

Dear student

There are significant efforts behind the completion of this book, in the process of its preparation, review, and development, and there are funds spent to print it and deliver it to you, to support your learning and your scientific and moral advancement, so, try to be faithful to this effort, appreciating it by preserving your book.



وزارة التعليم
Ministry of Education
2026 - 1448

Deposit number: 1446/19524
ISBN: 978-603-514-073-7

Internet of Things

Second year - Secondary stage - Pathways System

1448 - 2026 Edition



المملكة العربية السعودية

رؤية
2030
المملكة العربية السعودية
KINGDOM OF SAUDI ARABIA

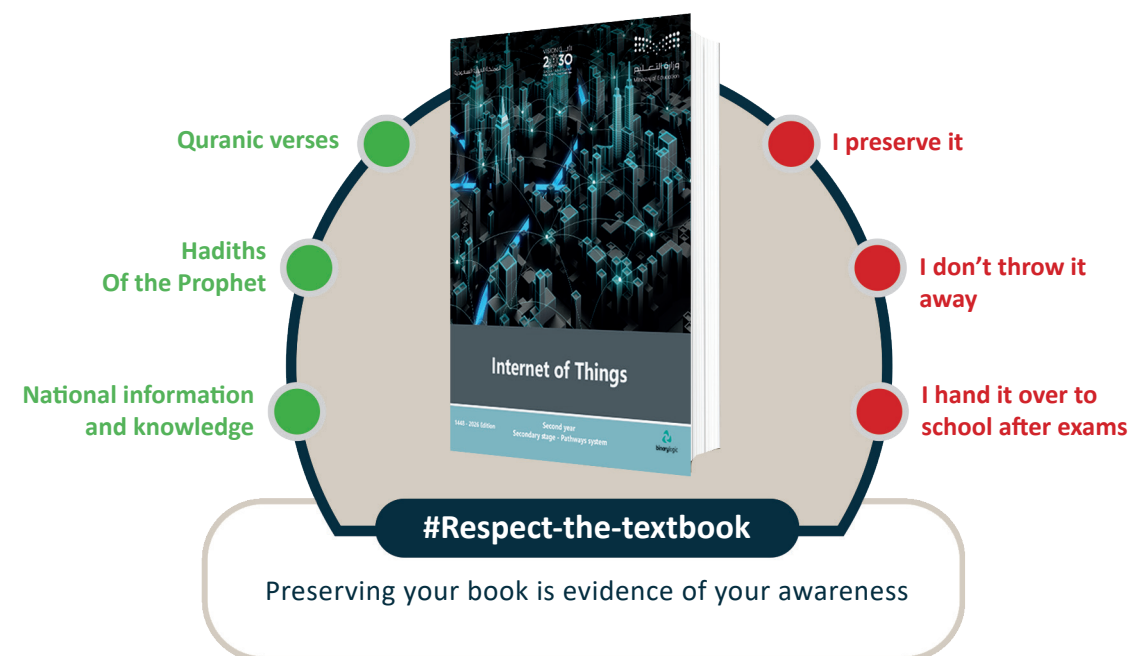
وزارة التعليم
Ministry of Education

Internet of Things

1448 - 2026 Edition

Second year
Secondary stage - Pathways system





قررت وزارة التعليم تدريس
هذا الكتاب وطبعه على نفقتها



المملكة العربية السعودية

Internet of Things

Secondary stage - Pathways system

Second year



وزارة التعليم
Ministry of Education
2026 - 1448

The book is distributed freely and cannot be sold.

1448 - 2026 Edition

Publisher: Tatweer Company for Educational Services

Published under a special agreement between Binary Logic SA and Tatweer Education Services Company (Contract No. 0003/2022) for use only in the Kingdom of Saudi Arabia

Copyright © 2026 Binary Logic SA

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, without permission in writing from the publishers.

Please note: This book contains links to websites that are not maintained by Binary Logic. Although we make every effort to ensure these links are accurate, up-to-date and appropriate, Binary Logic cannot take responsibility for the content of any external websites.

Trademark notice: Product or corporate names mentioned herein may be trademarks or registered trademarks and are used only for identification and explanation without intent to infringe. Binary Logic disclaims any affiliation, sponsorship, or endorsement by the respective trademark owners. Excel is a registered trademark of Microsoft Corporation. Tinkercad is a registered trademark of Autodesk Inc. "Python" and the Python logos are registered trademarks of Python Software Foundation. Jupyter is a registered trademark of Project Jupyter. PyCharm is a trademark of JetBrains s.r.o. MultisimLive is a trademark of National Instruments Corporation. CupCarbon is a registered trademark of CupCarbon. Arduino is a registered trademark of Arduino SA. Micro:bit is a registered trademark of Micro:bit Educational Foundation.

The above companies or organizations do not sponsor, authorize, or endorse this book.

The publisher has made every effort to trace all copyright holders, but if they have inadvertently overlooked any they will be pleased to make the necessary arrangements at the first opportunity.



©National Curriculum Center, 2025

National Curriculum Center

Internet of Things. / National Curriculum Center .-

Riyadh, 2025

352p.; 210*25.5cm

L.D. no.: 1446/19524

ISBN: 978-603-514-073-7

Copyright reserved to the Ministry of Education

www.moe.gov.sa

Enrichment and support materials on the “iEN Ethraia Platform”



ien.edu.sa

Dear students, parents and anyone interested in education, we welcome your communication to improve our textbooks. Your suggestions are our top priorities.



fb.ien.edu.sa



وزارة التعليم
Ministry of Education
2026 - 1448

Introduction:

The progress and development of countries is measured by the ability to invest in education, and the extent to which their educational system responds to the requirements and changes of the generations. In the interest of the Ministry of Education sustaining the development of its educational systems, and in response to the vision of the Kingdom of Saudi Arabia 2030, the Ministry of education has taken the initiative to adopt the “Secondary Education Pathways” system to bring about an effective and comprehensive change in high school.

The secondary education pathways system provides a distinguished and modern educational model for high school in the Kingdom of Saudi Arabia, which efficiently contributes to:

- Strengthening the values of belonging to our homeland “the Kingdom of Saudi Arabia” and loyalty to its wise leadership “may God protect him” based on a pure belief supported by the tolerant teachings of Islam.
- Strengthening the values of citizenship by focusing on them in school subjects and activities, in line with the demands of sustainable development, and the development plans in the Kingdom of Saudi Arabia that emphasize the consolidation of both values and identity, based on the teachings of Islam and its moderation.
- Qualifying students in line with future specializations in universities or the required jobs; ensuring the consistency of education outputs with the labor market requirements.
- Enabling students to pursue education in their preferred path at early stages, according to their interests and abilities.
- Enabling students to join specific scientific and administrative disciplines related to the labor market and future jobs.
- Participation of students in an enjoyable and encouraging learning environment in school based on a constructive philosophy and applied practices within an active learning environment.
- Delivering students through an integrated educational journey from the primary level to the end of the high school level and facilitating their transition process to post-general education.
- Providing students with technical and personal skills that help them deal with life and respond to the requirements of their level.
- Expanding opportunities for graduate students through various options in addition to universities, such as: obtaining professional certificates, joining applied faculties, and earning job diplomas.

The pathways system consists of nine semesters that are taught over three years, including a common first year in which students receive lessons in various scientific and humanities fields, followed by two specialized years, in which students study a general path and four specialized paths consistent with their interests and abilities, which are: the Rightful path, Business Administration path, Computer Science and Engineering path, Health and Life path, which makes this system the best for students in terms of:

- The existence of new study subjects that match the requirements of the Fourth Industrial Revolution and development plans, and the Kingdom’s Vision 2030, which aims to develop higher-order thinking, problem-solving, and research skills.
- Elective field programs that are consistent with the needs of the labor market and students’ interests, as they enable students to join a specific elective field according to a specific job skill.
- Scale as it ensures the achievement of students’ efficiency and effectiveness, and helps them identify their tendencies and interests, and reveal their strengths, which enhances their chances of success in the future.
- Volunteer work designed specifically for students in line with the philosophy of activities in schools, and is one of the graduation requirements; which helps to promote human values, and build society (its development and cohesion).

- Bridging which enables students to move from one path to another according to specific mechanisms.
- Proficiency classes through which skills are developed and the achievement level improved, by providing enrichment and remedial mastery classes.
- The options of integrated learning and distance learning, which are built in the paths system based on flexibility, convenience, interaction and effectiveness.
- The graduation project that helps students integrate theoretical experiences with applied practices.
- Professional and skill certificates granted to students after completing specific tasks, and certain tests compatible with specialized organizations.

Accordingly, the computer science and engineering path as one of the updated paths at the secondary level contributes to achieving best practices by investing in human capital, and transforming the student into a participating and productive individual for science and knowledge, while providing him with the skills and experience necessary to complete his studies in fields that meet his interests and abilities, or to join the labor market.

The Internet of Things is one of the main subjects in the course of Computer science and Engineering Pathway, as it contributes to clarifying the concepts of the Internet of Things and the technologies associated with it. These help to employ these technologies in several areas of life, such as smart cities, education, agriculture, medicine, and other various economic fields. This course aims to introduce the student to the importance of the Internet of Things and its role in Industry 4.0, with the definition of policies and legislation related to the safe and ethical use of Internet of Things technologies. It also focuses on enhancing the skills of connecting Internet of things devices, how to send and receive data between them, and their role in smart systems and environments. This course also includes projects and practical exercises for what the student learns. There are also realistic exercises for the student to solve that simulate his cognitive levels under the guidance and supervision of the teacher.

The Internet of Things book is characterized by modern engagement methods, which make students can learn and interact with it through the various exercises and activities it provides. This book also emphasizes important aspects of Internet of Things education and learning, which are:

- The connection between the content and real-life problems.
- Diversity of ways to display engaging content.
- Highlights the role of the learner in the teaching and learning processes.
- Attention to the contents' structure and coherence.
- The skill of employing appropriate techniques in different situations.
- The ability to employ various methods in evaluating students in proportion to their individual differences.

To be on pace with global developments in this field, the Internet of Things book will provide the teacher with an integrated set of diverse educational materials that take into account the individual differences between students, in addition to educational software and websites, which provide students with the opportunity to employ modern technologies and practice-based communication; This solidifies its role in the teaching and learning process.

As we present this book to our dear students, we hope it will capture their interest, meet their requirements, and make learning this material more enjoyable and useful.

God grants success



بِسْمِ اللَّهِ الرَّحْمَنِ الرَّحِيمِ



Contents

Part 1

1. IoT Fundamentals..... 10

Lesson 1 IoT Concepts	11
Exercises	18
Lesson 2 The IoT Devices	21
Exercises	30
Project	34

2. The IoT in our Lives 36

Lesson 1 The IoT Platform	37
Exercises	47
Lesson 2 IoT Applications and Challenges ...	50
Exercises	62
Project	66

3. Building IoT Applications with Arduino 68

Lesson 1 Building a Smart Home System	69
Exercises	85
Lesson 2 Build a Plant Watering System	88
Exercises	98
Lesson 3 Build a Gas Leak System.....	100
Exercises	111
Project	114

4. Building an IoT Cloud Application 116

Lesson 1 Setup Arduino Development Environment	117
Exercises	133
Lesson 2 Programming Arduino with Python	136
Exercises	148
Lesson 3 Interacting with a Cloud Web Service	150
Exercises	161
Project	164



Part 2

5. IoT Advanced Applications 168

Lesson 1	IoT Application Areas	169
	Exercises	176
Lesson 2	IoT Networking Technologies	180
	Exercises.....	190
Lesson 3	Security and Privacy of IoT Systems	194
	Exercises.....	200
Project		204

6. IoT Programming with C++206

Lesson 1	Smart Security Applications with C++	207
	Exercises	223
Lesson 2	From Tinkercad Blocks to C++	228
	Exercises.....	238
Lesson 3	Microcontroller Programming with C++	242
	Exercises	258
Project		260

7. IoT Messaging 262

Lesson 1	Smart Cities and the MQTT Protocol	263
	Exercises	269
Lesson 2	Designing and Programming a Smart Waste IoT Device	273
	Exercises	289

Lesson 3	Building a Smart Waste Management Solution	292
	Exercises	304
Project		306

8. IoT Wireless Sensor Network Simulation308

Lesson 1	Introduction to CupCarbon.....	309
	Exercises	320
Lesson 2	Communication in an IoT Network	323
	Exercises.....	335
Lesson 3	IoT and Automated Mobile Devices	336
	Exercises.....	349
Project		350



Part 1

Unit 1
IoT Fundamentals

Unit 2
The IoT in our Lives

Unit 3
Building IoT Applications with Arduino

Unit 4
Building an IoT Cloud Application



1. IoT Fundamentals

In this unit, students will learn about the fundamental concepts behind the Internet of Things (IoT), its history throughout recent years, and how it is an integral part of emerging technologies. Students will also learn about the main components of smart objects, which form the basis of an Internet of Things system.

Learning Objectives

In this unit, you will learn to:

- > Define what the Internet of Things is.
- > Discuss the evolution of the Internet of Things.
- > Understand what the Internet of Things does.
- > Discuss the Internet of Thing's impact on emerging technologies.
- > Define what a smart object is.
- > Classify smart objects and their usage.
- > Distinguish between the various types of sensors for smart objects.
- > Distinguish between the various types of actuators for smart objects.



Lesson 1

IoT Concepts

Link to digital lesson



www.ien.edu.sa

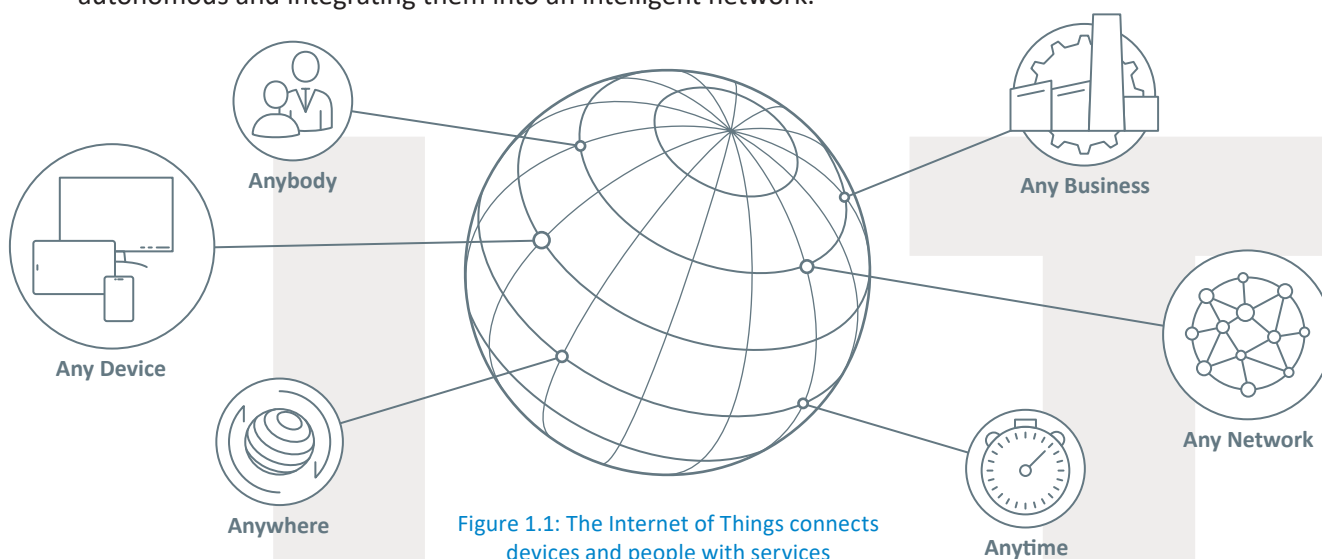
What is the Internet of Things?

We currently live in a world where virtually anything imaginable can be online, communicate with other devices or people and facilitate new services that improve our lives. The world is undergoing a significant technological transformation from self-driving drones carrying food orders to wearable sensors monitoring our health. This progression is collectively known as the Internet of Things (IoT).

The main objective of the IoT is to connect devices not already part of a computer network, whether private or public like the Internet, so that they may share data and interact with people and other objects. IoT is a change in technology that will enable devices to perceive and manage the physical environment by making objects autonomous and integrating them into an intelligent network.

Internet of Things (IoT)

A network of devices that are autonomously able to sense, monitor, or interact with the surrounding environment, in addition to collecting and exchanging data.



When devices and equipment can detect their environment and be remotely controlled across a network, the physical world and computers can be tightly integrated. This integration permits productivity, automation and cost reduction enhancements in almost every aspect of our lives. Now, the goal is to "connect the unconnected" and convert all objects to IoT devices, creating new applications on an intelligent network.

IoT Device

A physical object connected to a network that can gather and transmit data, be identified across that network and communicate with other IoT devices or platforms.



Smart Objects

Connected objects, or smart objects, are the objects that exchange data over a network. The interface with a user can be a simple one like a thermostat switch, a more complex one like that in a modern car, or an app with many features on a smartphone. However, there will also be many cases where the smart object does not have any "user interface", but instead has autonomous sensors and actuators interacting with their environment without human intervention. A sensor learns and measures its environment and an actuator can alter the physical world. The connected objects are divided into two categories: digital-first and physical-first. Digital-first objects are devices like smartphones, smartwatches, or home alarm systems that are necessarily designed to include digital/electronic interaction with the environment, while physical-first objects are everyday objects that require the addition of sensors or actuators to become smart objects. Refrigerators or lamps do not use and exchange data unless we enhance them by adding specific sensors, microcontrollers and antennas to connect them to the digital world of the Internet of Things.



Figure 1.2: Digital-first and Physical-first smart objects

The History of the Internet of Things

The concept of using sensors on physical objects and interacting with them over a network is not new. In the 1980s, some university students tried remotely tracking the contents of a soda vending machine. Technology at that time was more limited, and the Internet was not available. As networks grew to become accessible to any computer in the world and hardware companies advanced the miniaturization of chips, Central Processing Units (CPUs) and sensors, more technology applications were developed. The Internet, and the World Wide Web (WWW), evolved out of the original ARPAnet (Advanced Research Projects Agency Network), established in 1969, into a significantly more sophisticated network based on the Internet Protocol (IP) and Transmission Control Protocol (TCP). In April 1995, when the US government decommissioned the network backbone, an open framework for worldwide connectivity was built, and the Internet, as we know it today, was born. The unique IP address for each network device is the foundation for this connected world. Every device can connect to other devices using a unique code: smartphones, tablets, gaming consoles, cars, washing machines, lighting systems, front door locks, and credit card terminals. Some IoT devices use IP addresses to connect to a network, while others connect through wireless technologies such as Bluetooth or Zigbee via a gateway or hub. Therefore, not all IoT devices have a direct IP address that is accessible over the Internet.

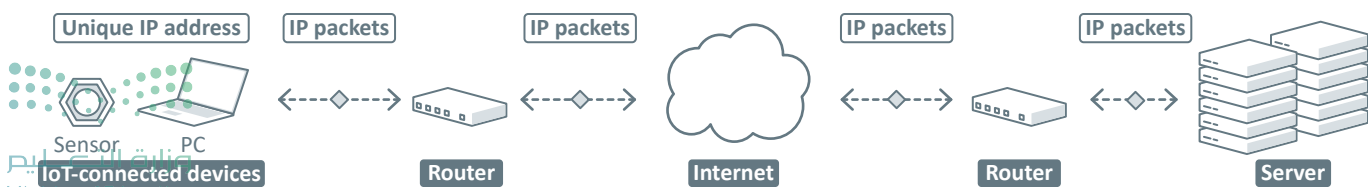


Figure 1.3: IP routing process over the Internet

The decrease in size and increase in capabilities in just a few years is obvious for anyone that used a desktop computer a few years ago and is using a smartphone today. Although much smaller, the smartphone is faster, has more memory and can easily connect to a wireless network with its own battery. Additionally, it integrates several sensors: camera, microphone, Global Positioning System (GPS), magnetometer, accelerometer, gyroscope, proximity sensor, and ambient light sensor. So, suppose a computer has limited requirements in processing power, local data storage and a few sensors. In that case, it can be a tiny device that can even be embedded in other physical objects.

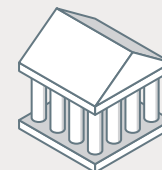
The Internet of Things era is commonly considered to have begun around 2008. Around this time, more devices became Internet-connected, and the Internet of Things became a reality. Kevin Ashton, a computer scientist, is credited with using the term "Internet of Things" for the first time in 1999 while working for a large multinational company. He used the term to describe a new concept involving tracking tags and computers with sensors connected to the Internet that can gather data to enhance the company's supply chain operations.

The Phases of the Evolution

The evolution of the Internet went through four phases, which also defined the development of the Internet of Things.

Connectivity Phase

In the early years of the Internet, only organizations and universities had Internet connectivity. Internet connection was uncommon for the typical individual. This phase linked individuals to the Web to easily access information.



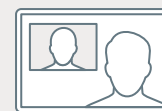
Networked Economy

With the advance of technology, connection speeds continued to increase, and connectivity was no longer the primary obstacle. This phase focused on maximizing efficiency and profit through networking.



Immersive Experiences

This era is defined by the emergence of social media, collaboration, and widespread device availability. Human interactions have been digitized, and applications are gradually transitioning to cloud infrastructure.



Internet of Things

The final phase concerns the communication and transaction of data between almost every device connected to the Internet to provide solutions and experiences for various industries.



Connectivity

Networked Economy

Immersive Experiences

Internet of Things

Figure 1.4: The evolution of the Internet

It is essential to recognize that the IoT is a collection of interconnected technologies and frameworks. Like the Web connects computers and content, the IoT connects devices, data, and people. Moreover, as more technologies and systems interconnect, and data gathering accelerates, this interconnected world gains power and provides value. Emerging technologies, such as the IoT, Artificial Intelligence (AI) and robotics, drive the digital transformation known as the Fourth Industrial Revolution or Industry 4.0.



What the Internet of Things Does

The IoT has led to technological breakthroughs and the creation of new types of products and services. The most important point is that IoT enables the transmission of gathered data from a room to a data center across the globe. The data can be processed along with other information in a central location, and an action can be made to happen in the room via an actuator. For example, a thermostat senses the temperature and humidity in your room. The algorithm that processes these two values combines them with weather forecast data in your city and switches on the air conditioning system based on an AI algorithm that minimizes energy consumption. All this happens in real-time and without human intervention.

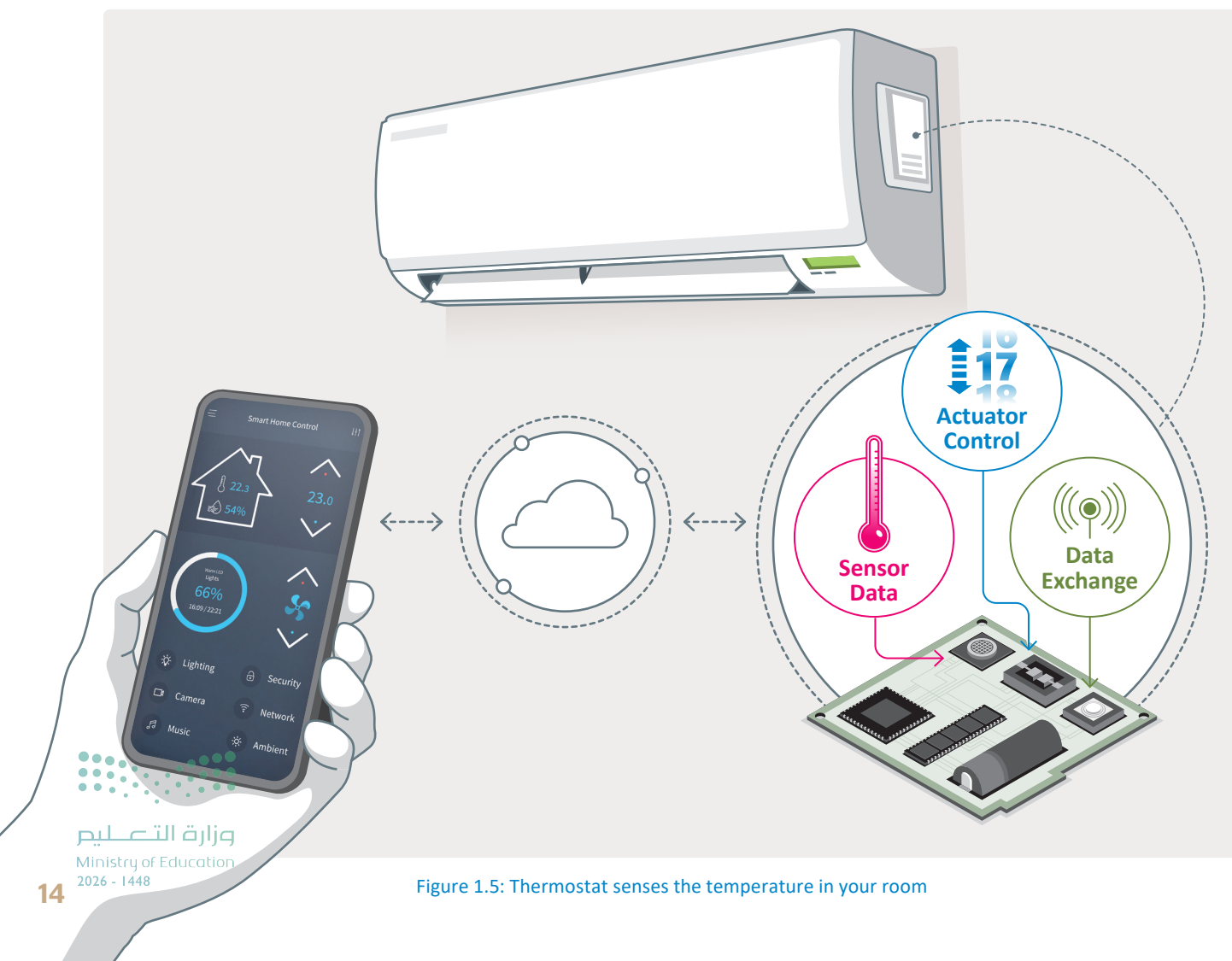


Figure 1.5: Thermostat senses the temperature in your room

IoT applications are categorized into four main domains:
Consumer, Commercial, Industrial, Infrastructure.

Wearables or smart homes fall into the consumer IoT domain, commercial IoT is found in schools, offices and retail shops, industrial IoT is the large scale use of IoT devices in factories, farms and transportation, and energy and water management comprise the infrastructure IoT domain.



Figure 1.6: Four main domains of IoT applications

As IoT becomes an integral part of technology solutions, it is increasingly mixing with other technologies like artificial intelligence and robotics, either using them to improve IoT applications or augmenting them with IoT-enabled objects. The result is that a collection of existing and emerging technologies is applied to solve old or new problems in the most efficient way. Table 1.1 shows applications of emerging technologies that are enhanced through IoT technologies.

Table 1.1: Enhanced applications through IoT

Application	Description
Automation	Machines and technologies that aid in automation can be found throughout history. The capacity to automate an activity frequently results in increased speed, efficiency, safety, and cost. Today, automation is what enables intelligent homes, intelligent buildings, and intelligent factories. It includes lighting controls, smart speakers, security systems, intelligent home products, and robots.
Computer Vision	AI-powered computer vision technologies enable computers to analyze images and videos and extract information from them, such as recognizing objects, faces, and movement. Face recognition, helping drones and vehicles steer and avoid collisions, and improving machine learning models for assessing the accuracy of chemotherapy and other treatments through the analysis of photos and scans are all current applications of the technology. In industrial applications, this approach can increase the fault detection rate by 90 percent or more in various processes.
Natural Language Processing (NLP)	This field employs linguistics, computing, and Artificial Intelligence to understand and implement human language. Alexa, Siri, and Google Assistant are exemplary NLP user interfaces. Nonetheless, voice interfaces are becoming increasingly prevalent on linked devices and equipment. This technique is also applied to chatbots and automated web services requiring typing or speaking. Nowadays, researchers are developing methods for systems to recognize emotions and sentiment.

Application	Description
Machine Learning	This technology, a subfield of AI, interprets and predicts future outcomes for various scenarios using mathematical models trained on "training data." Machine learning helps globally distributed systems inside the Internet of Things complete tasks without explicit programming. It is particularly beneficial for monitoring, forecasting, and telemetry applications.
Edge AI	Increasing numbers of digital devices are capable of performing local processing. Sensors on a device, such as a robot or autonomous car, transmit this data as the edge AI system performs the calculation, and the device stores the results. In certain instances, it may also transmit this data to the cloud. This architecture enables devices to run faster, more intelligently and with less energy. It drastically alters the operation of autonomous devices and extends the battery life of sensors by years.
Advanced Analytics	Because data is frequently dispersed, the IoT alters how analytic procedures are carried out. This implies the requirement for software to assemble and interpret the appropriate data. Manufacturing, healthcare, transportation, financial services, energy, telecommunications, and home automation are among the industries that benefit from IoT-centered analytics.
Robotics	Autonomous machines, such as drones, mobile robots, and autonomous vehicles, are growing rapidly due to onboard artificial intelligence and powerful sensor technology. A new concept has been introduced: the Internet of Robotic Technologies (IoRT). It refers to systems that observe events around them, calculate the data onboard and via the cloud, and then use this information to act in the real world.
Augmented Reality (AR)	AR's strength lies in its capacity to integrate, modify, and augment the virtual and actual worlds. In the consumer market, AR smartphone apps are used to improve images, digitally try on clothing, and play games. Various glasses and goggles aid in various jobs, from training to engineering. Text and graphics are generated by a rendering engine that receives the appropriate data from the IoT and delivers it to a device.
Virtual Reality (VR)	Immersive and intuitive, 3D computer-generated simulations require an IoT infrastructure. Today's videoconferencing systems, for instance, are transforming into VR places where individuals from all over the world can join a meeting, participate in a webinar, or attend a virtual conference through a 2D screen such as a laptop or smartphone or dedicated headgear. VR enables the assembly of pieces from various physical locations within a single virtual world.
Blockchain	The distributed ledger technology, which was first connected with digital currency, plays a crucial role in the Internet of Things. It can monitor and authenticate data as it traverses devices, databases, and microservices. Thus, it can aid with automation and detect infractions such as tampering. This is particularly beneficial in a highly decentralized IoT context, where data continually passes across organizations, servers, and systems.

The Components of an IoT Application

An IoT application consists of hardware, software and infrastructure components. Some of them are essential, and others depend on the type of application. The key component is the "thing", the IoT device that interacts with its environment in different ways. The IoT device may have sensors or actuators, but it must have a microcontroller with its power supply, memory and a network connection module that exchanges data through a network. Some IoT devices may have screens or buttons for user interaction, and others are completely hidden from the human eye. Connectivity is critical and usually is accomplished through the Internet via specialized IoT gateways and optimized network protocols. It enables the IoT device to connect to local or centralized computing devices, the IoT cloud.

They can be simple computers or cloud servers in large data centers, even in other countries. The server handles data storage and processing to decide if commands for specific actions will be sent back to the IoT device. Data analytics are generated to provide useful insights on the IoT application use, especially when thousands or, in some cases, even millions of devices are involved. You will explore all the components of an IoT application in detail in the next lessons.

The "thing": the IoT device

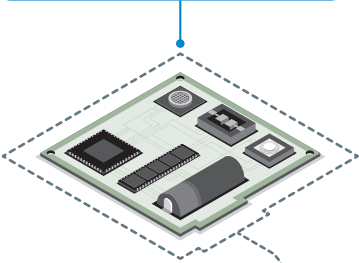


Table 1.2: Examples of IoT devices around us

Smart TV
Smart earbuds
Smart home controller (like Alexa)
Mesh WiFi system
Smart thermostat with room sensors
Fitness or health tracker
Smart air conditioner
Lights with smart switches and intelligent lamps
Smart power plugs
Air quality monitor
Electricity consumption monitor
Internet-connected refrigerator
Home security system with smart doorbell camera
Garage door with smart lock



Figure 1.7: Google Home Smart Assistant

Exercises

1

Read the sentences and tick ✓ True or False.	True	False
1. The IoT is a network of interconnected devices that communicate with each other.	☐	☐
2. The IoT includes devices that are not connected to the Internet.	☐	☐
3. Smart objects work exclusively without human input.	☐	☐
4. Digital-first objects are devices that only send and receive data.	☐	☐
5. The WWW originally evolved from the ARPAnet project.	☐	☐
6. During the Immersive Experiences phase, social media helped accelerate the use of big data.	☐	☐
7. IoT and AI research are advanced simultaneously to develop common applications.	☐	☐
8. Data from IoT sensors cannot be used for VR applications.	☐	☐
9. Smart objects communicate exclusively with each other.	☐	☐
10. Something as simple as a garage door cannot be part of an IoT system.	☐	☐

2

Define the main characteristic of the Internet of Things that separates it from other emerging technologies. Present your ideas below.



- 3 Consider which was the most critical technological advancement in recent history that made the IoT possible? Present your ideas below.

- 4 Which of the four phases of the Internet do you think was the most disruptive technologically and economically? Present your ideas below.

- 5 Which of the emerging technologies that drive Industry 4.0 do you believe is most impactful to the economy? Present your ideas below.



- 6 Search the Internet for an example of an IoT application that utilizes Computer Vision and NLP and describe it. Present your findings below.

- 7 Search the Internet for information about a VR application that generates virtual environments with input data from IoT sensors.

- 8 Search the Internet for information about how blockchain technologies help create decentralized IoT systems and applications.



Lesson 2

The IoT Devices

Link to digital lesson



www.ien.edu.sa

What is a "Thing"?

The Smart Objects

Things or smart objects are the building blocks of IoT. They are small and low-cost computers that interact with the physical environment around them by gathering data from sensors and providing real-time interactions via actuators. Sensors and actuators transform everyday objects into intelligent objects that are able to get information from and interact with the environment in a meaningful way. The real power of smart objects in an IoT solution comes from their being networked together rather than being isolated as standalone devices.



Smart objects are powered by a power source like the electrical grid, a battery or are self-powered through solar or wind energy. Their power consumption is so low that sometimes a smart object can run for months or years on batteries. There is a new generation of smart objects for the human body that can be powered by the electrical current of the body.



Every smart object has a communication device that sends the data collected from the sensors and receives the necessary instructions for the actuators. The communication device connects the smart object to the cloud.

The unifying component is a processing unit in the form of a microcontroller.

The microcontroller coordinates the sensors, the actuators and the communication device.

The microcontrollers used in development or for academic purposes, such as Arduino or Raspberry Pi, are very small computers.

Actual IoT applications use substantially smaller microcontrollers, sometimes as small as 2 by 2 millimeters in size. One example is "smart dust," which is a wireless network of computing and sensing platforms that are no bigger than a grain of sand and can work on their own. Smart dust can sense things like light, temperature, sound, and the presence of toxins or vibrations and record and send that information wirelessly to larger computer systems.

Tiny Low Cost Computer
Embedded into objects to make them smart. Can be organized into networks.

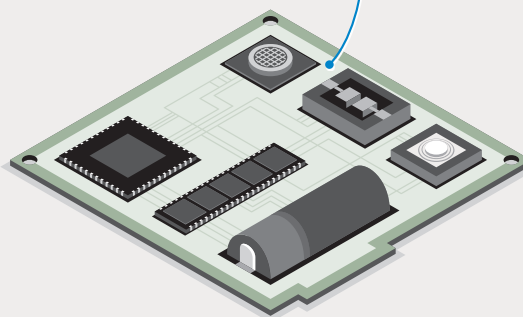


Figure 1.8: Tiny Low Cost Computer



Classifications of Smart Objects



Self-powered or power-connected

The object may carry its own energy source, a battery or solar panel, or is powered continuously by an external source. Self-powered objects can be very mobile but batteries restrict the object's usage period and affect data collection frequency and data transmission range.



Mobile or static

The smart object can be moved or has to remain always in the same place. An object can be mobile if it is attached to a bigger moving object.



Low or high reporting frequency

The frequency of reporting monitored parameters of the smart object can be low or high. A rust sensor on a bridge may report monthly values. A motion sensor in a car may report acceleration values hundreds of times per second. Higher reporting frequencies result in greater energy consumption, which may impose limitations on the power source.



Simple or rich data

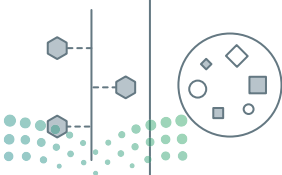
This classification is based on the amount of data collected and exchanged during each report cycle. A humidity sensor in a field can record simple daily index values, whereas an engine sensor may record hundreds of parameters, such as temperature, pressure, gas velocity, and compression speed. Data transfer rate is determined based on two factors: data classification (simple or complex), and data transmission rate (low to high). Throughput is a combined metric.

A medium-throughput object may transmit simple data at a relatively high rate (in which case the flow structure appears continuous) or rich data at a relatively low rate (in which case the flow structure looks bursty).



Report range

The distance between the smart object and the data collector defines the report range. For a fitness band sending data to your phone, for instance, the report range is up to a few meters. In contrast, a moisture sensor embedded in a road's asphalt surface may need to communicate with an antenna located hundreds of meters or even kilometers away.



Object density per cell

This classification is based on the amount of smart objects with similar communication needs connected to the same gateway. An oil pipeline may utilize one single sensor every few miles. In contrast, astronomy telescopes use hundreds or even thousands of mirrors over a small area, each equipped with multiple gyroscopes, gravity and vibration sensors.

The Main Components of a Smart Object

A smart object is a device with at least the four components listed below. The smart object may have just one sensor or one actuator, more than one sensor or more than one actuator or a combination of sensors and actuators, depending on the IoT application.

Processing unit

A smart object contains a processing unit for gathering data, processing and analyzing sensing information received by the sensor(s), coordinating control signals to any actuator, and operating a variety of operations, including the communication and power systems. Depending on the processing requirements of a given application, the type of processing unit employed can vary significantly. Microcontrollers are the most prevalent due to their compact size, versatility, programming ease, ubiquity, low power consumption, and low cost.

Sensors and Actuators

A smart object is able to interact with the physical world via its sensors and actuators. It is not necessary for a smart object to incorporate both sensors and actuators. Depending on the application, a smart object may contain one or more sensors and/or actuators.

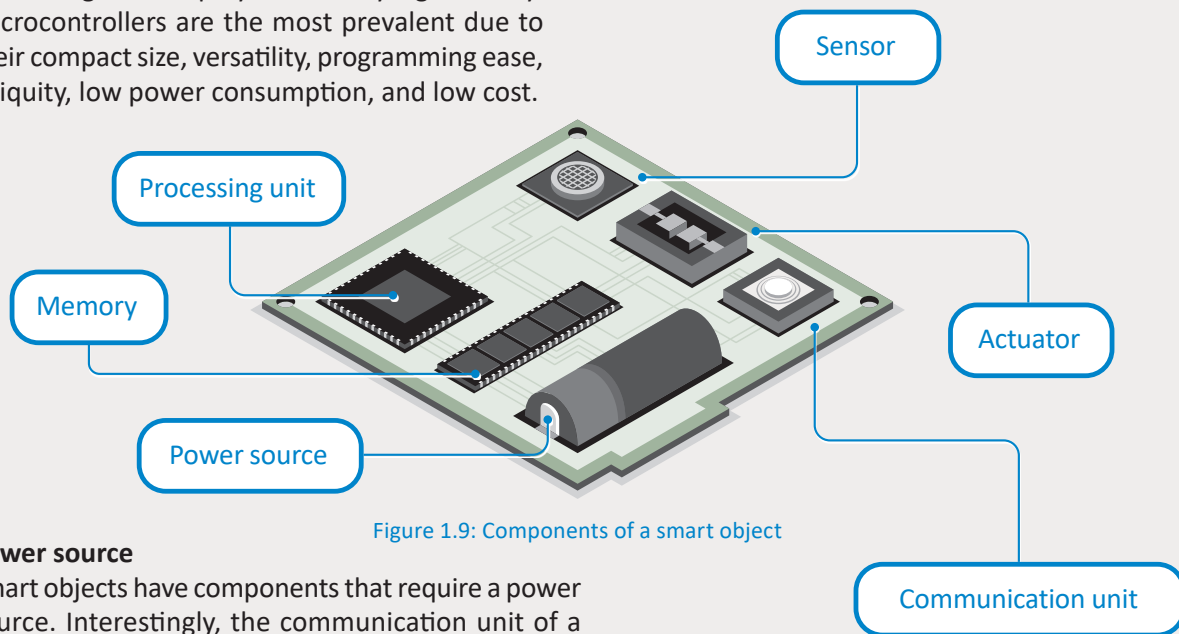


Figure 1.9: Components of a smart object

Power source

Smart objects have components that require a power source. Interestingly, the communication unit of a smart item typically consumes the greatest amount of energy. As with the other three elements of intelligent objects, the power needs vary substantially between applications. Smart objects typically have limited power, are deployed for an extended period of time, and are difficult to access.

This combination necessitates power efficiency, prudent power management, sleep modes, ultra-low power consumption hardware, etc., especially when the intelligent object relies on battery power. For long-term installations when smart items are, for all intents and purposes, unavailable, scavenging sources are typically used to provide power.

Communication unit

The communication unit is responsible for linking an intelligent item to other intelligent objects and the outside world (via the network). Smart object communication devices can be either wired or wireless. In IoT networks, intelligent items are interconnected wirelessly for a variety of reasons, including cost, limited infrastructure availability, and ease of implementation. There are numerous communication protocols for intelligent items.

The Sensors

A sensor does exactly what its name indicates: it senses. More specifically, a sensor measures a physical quantity, converts that measurement into data and passes it to be used by intelligent devices or humans. Sensors are not limited to gathering human-like sensory data. They provide a wide spectrum of measurement data with greater precision than human senses. Sensors can be embedded in any physical object and connected to the Internet by wired or wireless networks.

A modern car has an impressive collection of sensors that provide an immense amount of data that can be consumed by intelligent systems as well as shared with other vehicles on the road. The driver can check and control everything in the car with sensors of all types, like water and oil temperature, location, tire pressure, and velocity, that provide relevant data to improve safety and vehicle maintenance.

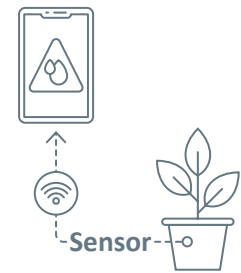


Figure 1.10: A sensor collecting and sending data from a plant pot to an IoT system



Figure 1.11: Sensors of a modern car

Classification of Sensors

Active or passive

Sensors can be classified based on whether they require an external power source to operate, and transmit and detect the energy at the same time (active) or do not require any external power source and do not transmit the energy but only detect it (passive).

Invasive or non-invasive

Sensors can be a part of the environment they are measuring (invasive) or be an external component (non-invasive).

Contact or no-contact

Sensors may require physical contact with the object being measured (contact) or not (no-contact).

Absolute or relative

Sensors can gather data on an absolute scale or relative to a reference value.

Area of application

Sensors can be categorized according to the specific industry in which they are utilized.

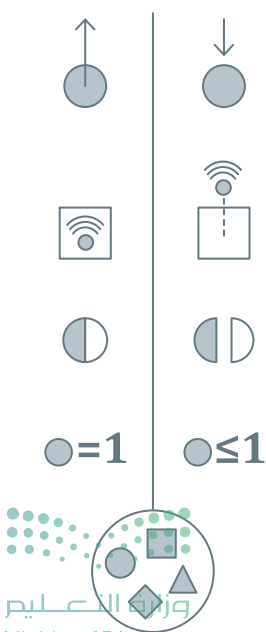
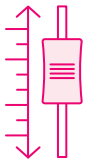
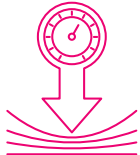
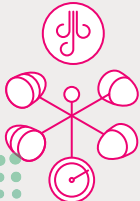
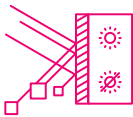


Table 1.3: Sensor types with examples

Type	Description	Examples
 Position	A position sensor measures the position of an object; the measurement can be in absolute or relative terms. There are three types of position sensors: linear, angular, and multi-axis.	Potentiometer, inclinometer, proximity sensor
 Occupancy and motion	In a surveillance area, occupancy sensors detect the presence of people and animals, while motion sensors detect the movement of people and objects. In contrast to motion sensors, occupancy sensors generate a signal even when a person is inactive.	Electric eye, radar
 Velocity and acceleration	Velocity sensors may be linear or angular, indicating how quickly an object moves in a straight line or how quickly it rotates. Acceleration sensors measure velocity changes.	Accelerometer, gyroscope
 Force	Force sensors determine if a physical force is applied.	Force gauge, viscometer, touch sensor
 Pressure	Similar to force sensors, pressure sensors measure the force exerted by liquids or gases.	Barometer, piezometer
 Flow	Flow sensors detect the fluid flow rate.	Anemometer, mass flow sensor, water meter

Type	Description	Examples
 Acoustic	Acoustic sensors measure sound levels in the environment.	Microphone, geophone, hydrophone
 Humidity	Humidity sensors measure the amount of humidity in the air or in a mass.	Hygrometer, humidity sensor, soil moisture sensor
 Light	Light sensors are capable of detecting the presence of light.	Infrared sensor, photodetector, flame detector
 Radiation	Radiation sensors detect environmental radiation.	Geiger-Müller counter, neutron detector
 Temperature	Temperature sensors quantify the amount of heat or cold within a system. Contact temperature sensors must be in physical contact with the target object. Non-contact temperature sensors measure temperature from a distance.	Thermometer, calorimeter, temperature gauge
 Chemical	Chemical sensors determine the chemical concentration within a system.	Breathalyzer, smoke detector
 Biosensor	Biosensors can detect biological properties in living organisms.	Blood glucose biosensor, pulse oximetry, electrocardiograph

The Actuators

Actuators are the complement of sensors.

Actuators receive a control signal, most commonly an electrical signal or digital command that triggers a physical effect in the system.

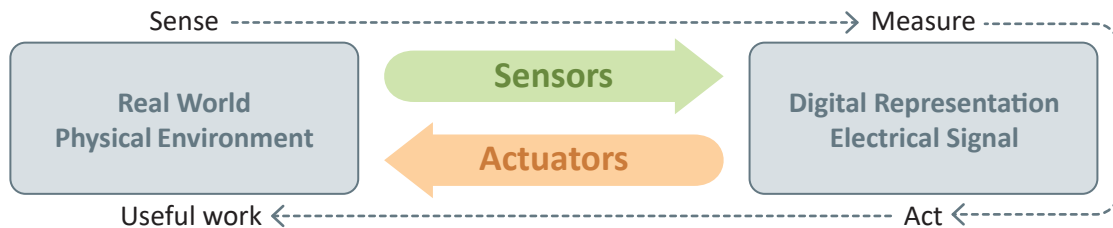


Figure 1.12: Sensor and actuator relationship

Human Analogy

Humans use their five senses to sense and measure their environment. The sensory organs convert this information into electrical pulses that the nervous system sends to the brain for processing. Likewise, IoT sensors are devices that sense and measure the physical world and send their measurements as electrical signals to a microprocessor or microcontroller for additional processing.

The human brain signals motor function and movement, and the nervous system carries that information to the appropriate part of the muscular system. Correspondingly, a processor can send an electrical signal to an actuator that converts the signal into physical action and has a measurable impact on its environment. This interaction between sensors, actuators, and processors and the similar functionality in biological systems is the basis for the fields of robotics and biometrics.

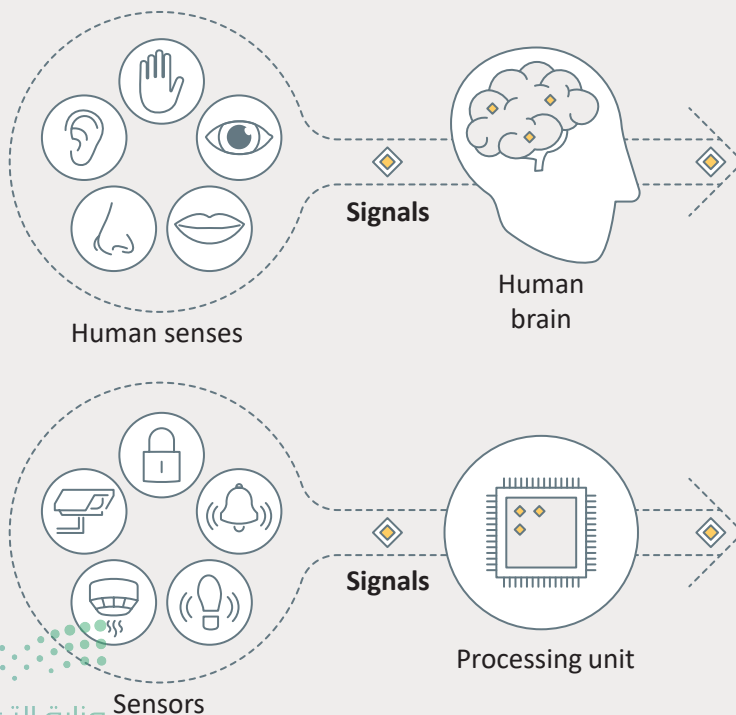


Figure 1.13: Human analogy

Classification of Actuators



Type of motion

Actuators can be categorized according to the type of motion they produce.

Examples: Linear, rotary, one/two/three-axes



Force Output

Actuators can be categorized according to the force exerted.

Examples: High power, low power, micro power



Output type

Actuators can be classified according to the number of stable-state outputs.

Examples: Binary, continuous



Area of application

Actuators can be categorized according to the specific industry in which they are used.

Examples: Manufacturing, automotive, medicine



Type of energy

Actuators can be categorized based on the type of energy they utilize.

Examples: Electrical, chemical, kinetic

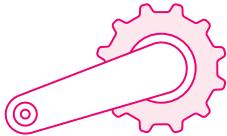
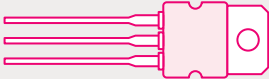
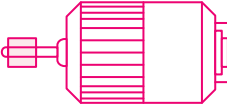
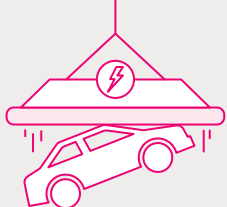
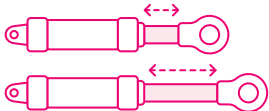


Figure 1.14: Tiny actuator (switch) for mechanical movement



Figure 1.15: Hydraulic actuator (piston) system on heavy machinery

Table 1.4: Actuator types with examples

Actuator type	Examples
 Mechanical actuator	Lever, screw jack, hand crank
 Electrical actuator	Thyristor, bipolar transistor, diode
 Electromechanical actuator	AC motor, DC motor, step motor
 Electromagnetic actuator	Electromagnet, linear solenoid
 Hydraulic and pneumatic actuator	Hydraulic cylinder, pneumatic cylinder, piston, pressure control valve
 Thermal and magnetic actuators	Magnetorestrictive material, bimetallic strip, piezoelectric bimorph
 Microactuators and nanoactuators	Electrostatic motor, microvalve, comb drive



Exercises

1

Read the sentences and tick ✓ True or False.	True	False
1. Smart objects are very complex electronic devices that require large amounts of processing power.	☐	☐
2. Smart objects are powered exclusively by renewable energy sources.	☐	☐
3. Smart objects can send data through a wide variety of communication frequencies	☐	☐
4. The processing unit sends the collected sensor data to external services on the Internet.	☐	☐
5. Velocity sensors can detect the presence of an object in an environment.	☐	☐
6. Radiation sensors can detect thermal readings in an environment.	☐	☐
7. The relationship between a processing unit and an actuator resembles the relationship between the human senses and the brain.	☐	☐
8. Actuators can take as input only discrete data.	☐	☐
9. Actuators can take input from sensors directly, without the need for external data services.	☐	☐



2 Describe the main components of a smart object.

3 Analyze which types of applications would require self-powered sensors and which would require power-connected sensors. Present your ideas below.

4 Different IoT applications require different types of smart objects. List the features that categorize smart objects.



5

Select three sensor types that are important for environment measurements and discuss their usage.

[illegible]

6

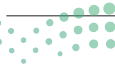
Consider how the study of the communication between the various systems of the human body could help engineers build better interconnected systems for IoT solutions.



7 Distinguish which types of actuators might be more commonly found in robotics applications.

This image shows a blank sheet of white paper with horizontal ruling lines. The lines are evenly spaced and run across the width of the page. There are no margins, text, or other markings on the paper.

8 Consider which actuator types require more complex data to accomplish their required tasks. Briefly explain why.



Project

The Internet of Things is expanding to cover most aspects of our daily and work lives. When integrated into an IoT application, common devices become smart objects that produce and consume IoT data.

1

Choose a common electronic device you use daily and create a proposal for an IoT application that begins with this device. This device sends and receives data from the IoT system to create forecasts and optimize its effectiveness.

2

Create a PowerPoint presentation that describes your proposal and how it could scale to more devices of the same type or in conjunction with other types of smart objects.



Wrap up

Now you have learned to:

- > Define the IoT and describe its history.
- > Distinguish the various uses of IoT applications.
- > Understand how the IoT enhances emerging technologies.
- > Describe smart objects and their uses.
- > Classify the sensors and the actuators that exist in smart objects.

KEY TERMS

Actuator	IoT	Sensor
Digital-First Object	IoT Device	Smart Object
Internet of Things	Physical-First Object	Thing



2. The IoT in our Lives

In this unit, students will learn about how networking and communication technologies enable IoT platforms and systems. They will also learn about their social impact and their expected evolution in the near future. Finally, we will cover the main concerns and limitations of IoT systems and how they are regulated.

Learning Objectives

In this unit, you will learn to:

- > Identify the cloud, fog, and edge layers for IoT applications.
- > Describe the main enablers of IoT systems.
- > Distinguish the networking protocols and technologies that are the basis of IoT communications.
- > Classify the main applications of IoT solutions.
- > Recognize the importance of the IoT in the near future.
- > Describe the security concerns of complex IoT systems.
- > State the technological challenges of IoT systems.
- > Identify how IoT applications are regulated.





The Internet in the Internet of Things

The term Internet of Things contains two keywords: Internet and Things. We explained what Things (smart devices) are and now we will explore the Internet part of an IoT solution. Cloud connectivity and services enable smart objects to gather measurements from the sensors and send commands to control the actuators. IoT devices are usually connected to a cloud IoT service using a communication protocol and through this service, the main IoT application will take decisions based on the collected data.

In this lesson you will learn about the Cloud-Fog-Edge architecture, the networks and protocols used, and the type of data exchanged to support an effective IoT solution.

Cloud, Fog and Edge

The most common cloud computing infrastructure is called the Cloud-Fog-Edge architecture. Briefly, this model describes three levels of storage, connectivity, and applications where "cloud" is the data center infrastructure, "edge" is the data processing that happens at the network's edge, close to the physical object creating the data, and "fog" is the mediator between the edge and the cloud for various purposes. You already know how cloud computing enables the storage and processing of data for a range of applications. Now, you will learn about the two other parts of the IoT computing infrastructure.

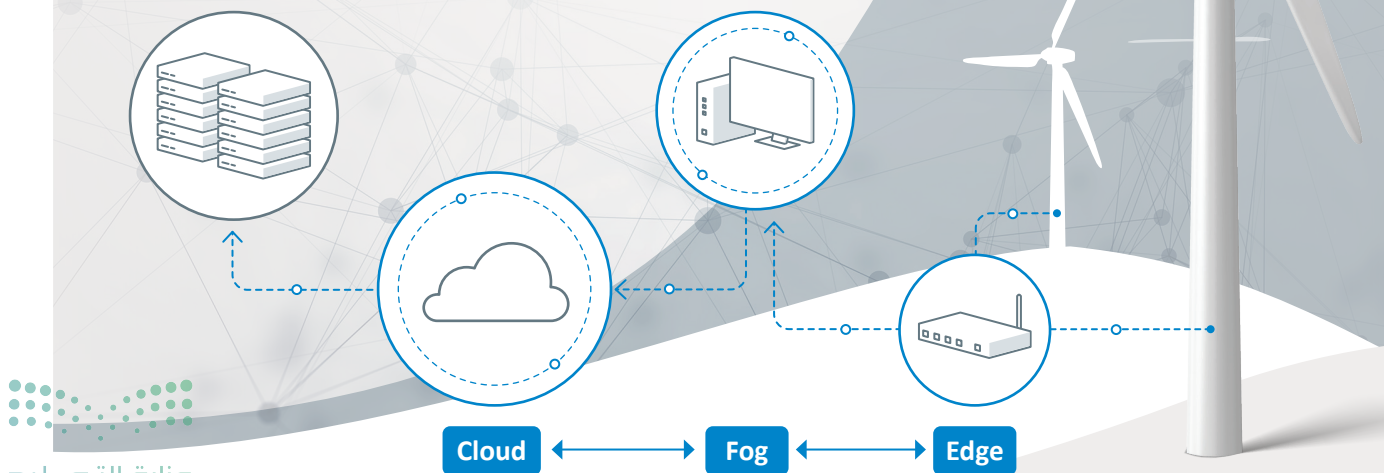


Figure 2.1: Edge to fog to cloud architecture

Fog Computing Fundamentals

A constant technological goal of IoT systems is to distribute data management as close to the sensor/actuator nodes. Fog computing is the most well-known example of edge services in IoT, that is closer to the things that generate the IoT data. A fog node can be any device with computing, storage, and network connectivity. Some examples are industrial controllers, switches, routers, embedded servers, and IoT gateways. Analyzing IoT data close to its origin reduces latency, offloads gigabytes of network traffic from the core network, and keeps sensitive data within the local network.

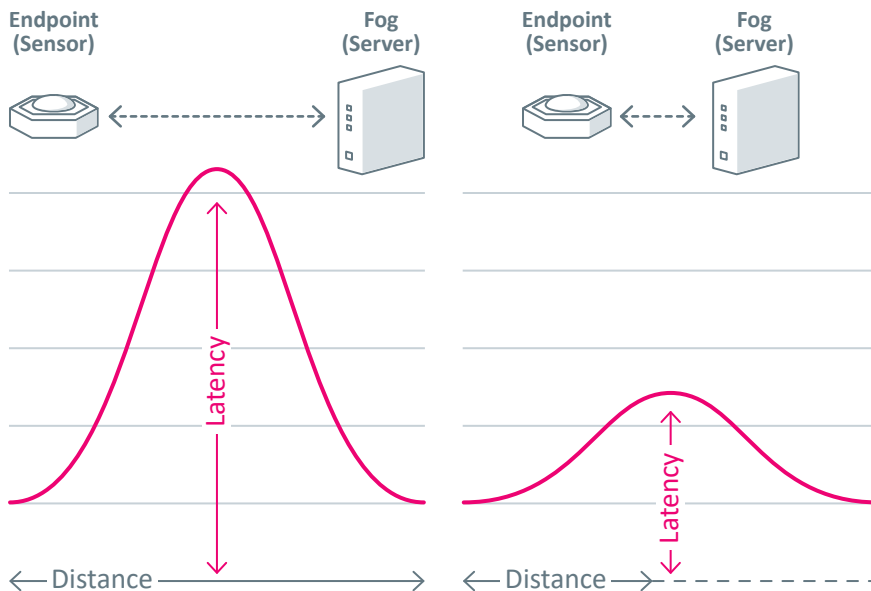


Figure 2.2: Example of latency increasing by the distance

Typically, fog services are performed very close to the IoT device, as close to the edge endpoints as possible. One significant benefit is that the fog node has contextual awareness of the sensors it manages due to its geographic proximity to those sensors. Because the fog node can analyze data from all the sensors on that section, it can provide contextual analysis of the messages it receives and may choose to send only relevant data to the cloud. As a result, the volume of data sent upstream is significantly reduced, making it much more helpful to cloud-based applications and analytics servers.

Furthermore, having contextual awareness allows fog nodes to respond to events in the IoT network much faster than the traditional cloud model, which would likely cause higher latency and slower response times. Thus, the fog layer provides a distributed network capability, allowing devices to be monitored, controlled, and analyzed in real-time without waiting for communication from the cloud's central application and analytics servers.

Latency

is the delay in the processing of data over a network connection or the delay between a user action and the response.

Endpoint

A data routing service that can receive and send data from and to other services. It can be a program on a computer or a dedicated hardware device.

Gateway

A gateway enables connectivity for devices that cannot connect directly to the Internet. A WiFi hotspot works as a gateway.

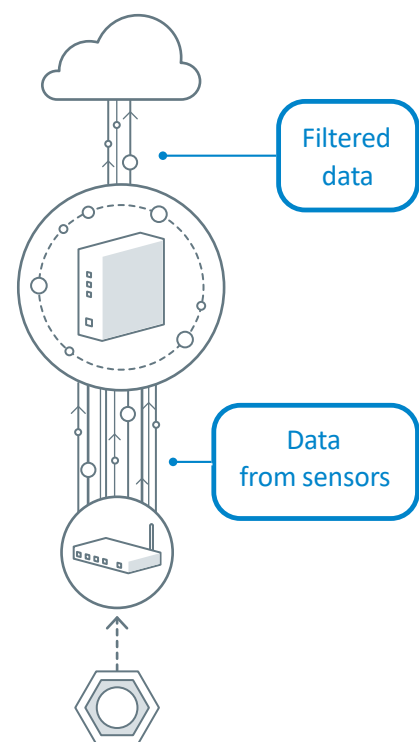


Figure 2.3: Fog data analysis

Fog Computing Advantages

Fog applications are as diverse as the IoT itself. Their typical responsibilities include data reduction, monitoring, and analyzing real-time data from network-connected devices.

Table 2.1: Fog Computing Advantages

Advantage	Description
Contextual location awareness and low latency	The fog node is located as close as possible to the IoT endpoint to provide distributed computing.
Geographic distribution	In direct contrast to the more centralized cloud, fog node-targeted services and applications require widely spread installations.
Deployment near IoT endpoints	Typically, fog nodes are deployed in the presence of several IoT endpoints. Typical metering deployments typically consist of 3000 to 4000 nodes per gateway, which also serves as a fog computing node.
Wireless communication between the fog and the IoT device	Although it is possible to connect wired nodes, the benefits of fog are greatest when a large number of endpoints are involved, and wireless access is the simplest way to achieve scalability.
Use for real-time interactions	Important fog applications involve interactions in real-time, as opposed to batch processing. The preprocessing of data in fog nodes enables upper-layer applications to process a subset of the larger data packets.

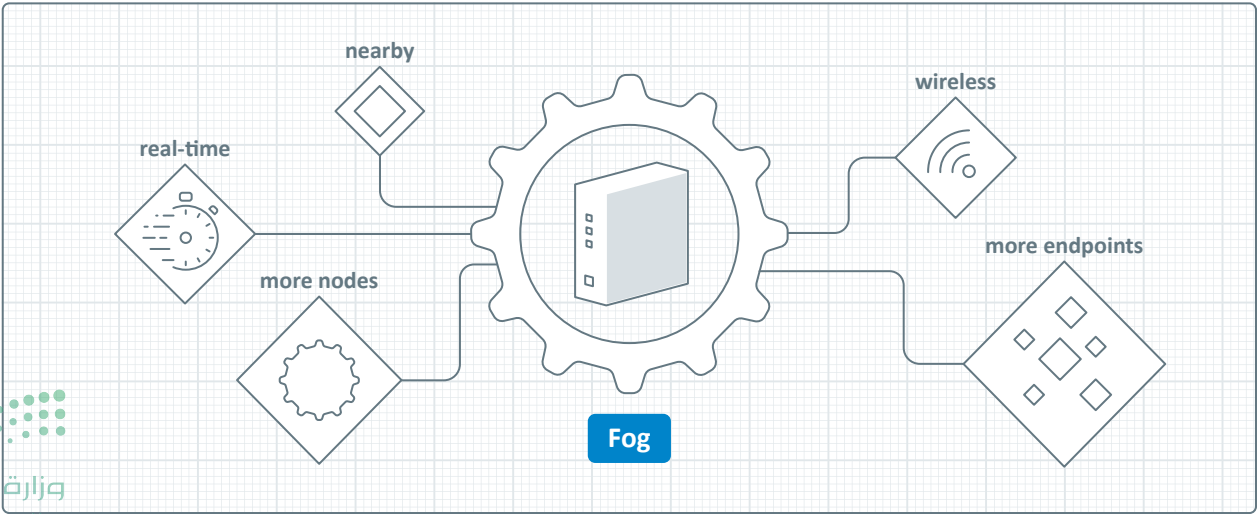


Figure 2.4: Features of the fog layer

Edge Computing Endpoints

Newer types of IoT endpoints have sufficient computing power to perform low-level analytics and filtering. These are called edge computing endpoints or edge devices. This layer in the cloud-fog-edge architecture provides more efficiency to the IoT solution. The cloud is not replaced by edge or fog computing. Instead, all these layers complement one another. The edge and fog computing layers assist in filtering, analyzing, and managing data. They prevent the cloud from being queried for each event by each IoT device. This model suggests that network bandwidth, computation, and data storage resources are organized hierarchically. Data is collected, analyzed, and sent at each stage based on the capabilities of the resources at each layer. The latency decreases as more data is sent to edge endpoints closer to the IoT devices. The benefit of this hierarchy is that responses to events from resources close to the IoT device are quick with immediate results. At the same time, big data storage and processing resources in cloud data centers are available when needed.

Edge Device

Edge devices are intelligent gateways capable of processing data locally. IoT devices can connect to edge devices over local networks like WiFi and Bluetooth.

Edge and Fog Working Together

Edge and fog computing necessitates using an abstraction layer to enable applications to communicate with one another. The abstraction layer exposes standardized Application Programming Interfaces (APIs) for monitoring, provisioning, and controlling physical resources. To support flexibility and consistency across the IoT system, the abstraction layer also requires a mechanism to support virtualization, with the ability to run multiple operating systems or service containers on physical devices. Regarding architecture, fog nodes closest to the network edge receive data from IoT devices. The fog IoT application then directs various data types to the best location for analysis. The most time-sensitive data is analyzed closest to the smart objects generating the data on the edge or fog node. Data that can be acted on in seconds or minutes is routed to an aggregation node for analysis and action. Less time-critical data is sent to the cloud for historical analysis, big data analytics, and long-term storage. For example, thousands of fog nodes could send data summaries to the cloud for historical analysis and storage. Considering these factors will help decide whether edge or fog computing would improve the system's efficiency.

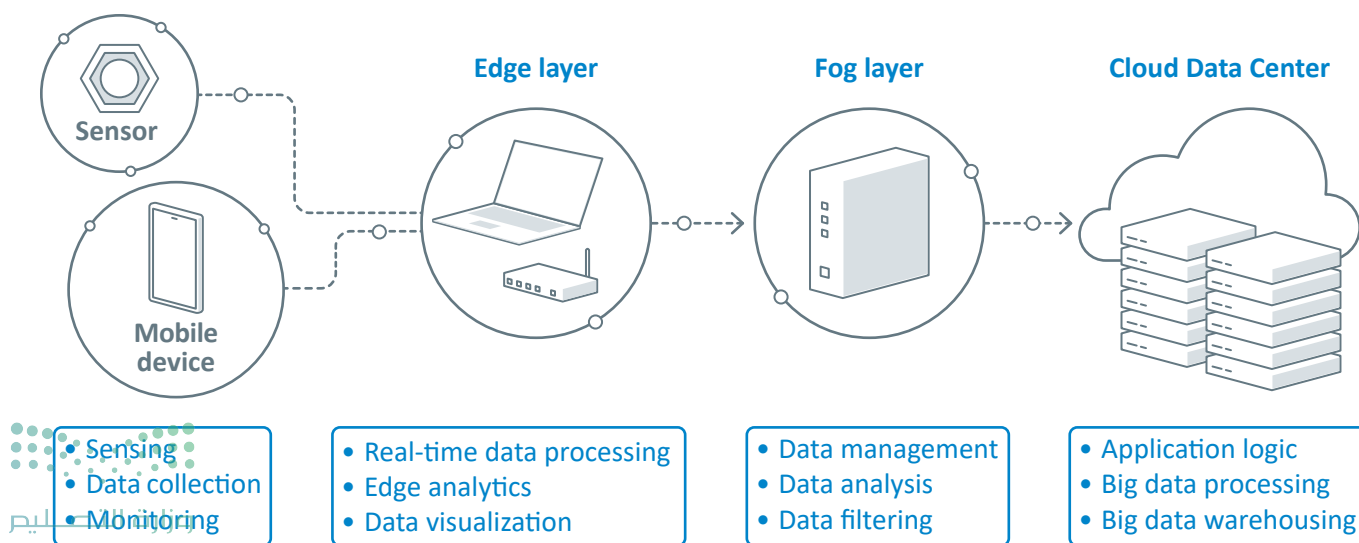


Figure 2.5: Edge and fog layers in an IoT system

IoT Enablers

IoT Data

In IoT, data, generated by billions of IoT devices, creates considerable value since it allows organizations to analyze sensed data in order to offer new services that improve user experience, cut costs or open up new revenue streams. Even though unstructured data makes up the majority of IoT-generated data, the insights it offers through analytics can transform operations and help to develop new business models. However, managing and evaluating all this data can be challenging. To solve this problem, IoT deployments are designed to filter noncritical data generation or consumption, cut back on upstream data that is not necessary, and respond to devices as quickly as possible.

Consider a smart city with hundreds of thousands of smart lighting devices connected via an IoT system. Most information transferred between lighting network modules and the control center is of little interest. However, patterns in this data might give helpful insights that can assist in forecasting when lights need repairs or when they can be turned on or off, reducing operational expenses.

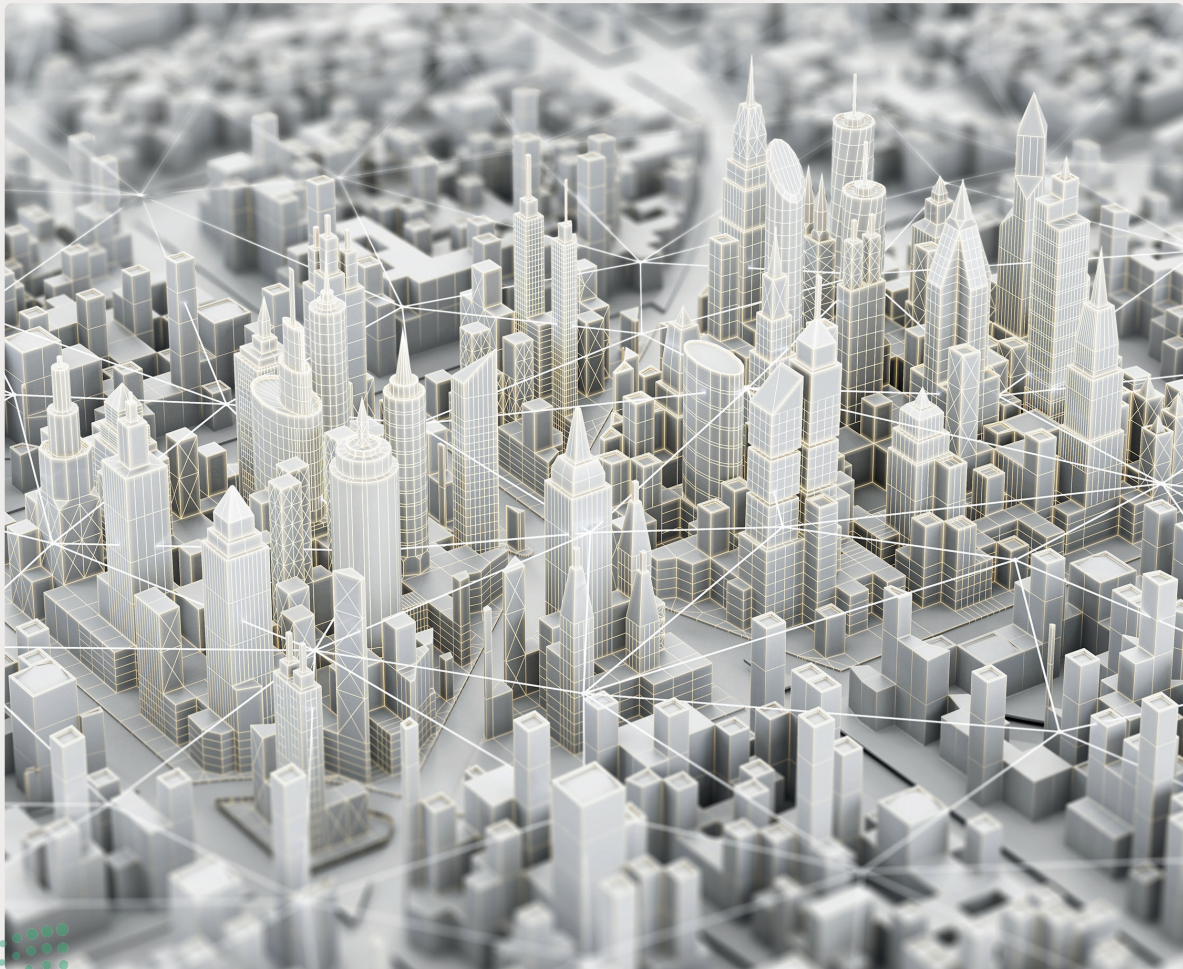


Figure 2.6: A smart city with connected smart devices



Data Classification

Not all data should be treated in the same way; it can be classified and evaluated in several ways. Several data analytics tools and processing techniques can be utilized. IoT-relevant classifications include whether the data is structured or unstructured and whether it is in motion or at rest.

Data in Motion and Data at Rest

Data is either in transit, and called "data in motion", or stored somewhere and called "data at rest". Network traffic and traditional client/server exchanges, such as web browsing, file transfers, and emails are examples of data in motion. Data at rest is data saved on a simple USB drive, a hard drive on your notebook or a storage array in a cloud data center.

From the standpoint of the IoT, data from smart objects is considered data in motion as it travels across the network to its final destination. This data is frequently handled at the edge devices or fog nodes. When data processing happens at an edge device, it can be filtered or destroyed, or it may be transmitted to a fog node or a data center for additional processing and possible storage.

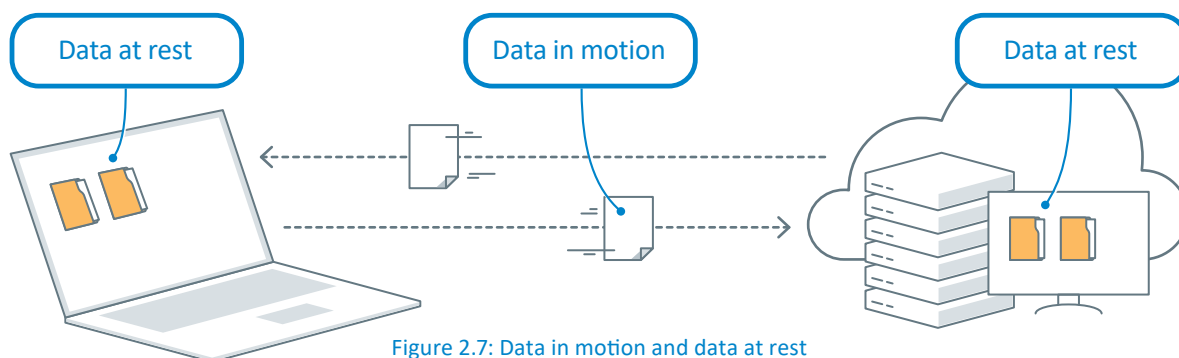


Figure 2.7: Data in motion and data at rest

Edge Analytics

The adoption of cloud services has been an area of evolution for IoT in recent years and data analytics is a vital element. In the world of IoT, huge amounts of data are collected on the devices and must frequently be analyzed and acted upon in real-time. Not only is the volume of data generated at the edge massive, requiring increased network bandwidth to the cloud, but the data may be so time sensitive that it requires urgent attention, and waiting for deep analysis in the cloud is not viable. A relatively new technology, edge analytics, solves this issue by providing data analytics functions within the IoT device. The data analysis is performed on the device for minimal latency before the data is sent to the cloud.



Figure 2.8: Edge analytics processing

Networking Protocols

Basic Networking Protocols

The fundamental Internet networking protocols, Internet Protocol (IP), TCP, and UDP, also provide connectivity for Internet of Things networks. Data transmission between smart objects and any other system in an IoT application is handled via higher-level protocols. These new protocols have been developed to address the requirements of IoT data transfer. Some IoT networks rely on a push model, such as a sensor reporting at regular intervals or responding to a local trigger. Others rely on a pull model, such as an application that queries the sensor for data across the IoT network.

Table 2.2: How TCP and UDP protocols work

Transmission Control Protocol (TCP)	User Datagram Protocol (UDP)
This connection-oriented protocol needs the formation of a session between the source and destination prior to data transmission. It can be compared to a standard telephone conversation, in which two phones must be connected and a communication link established before the two parties can communicate.	With this protocol, data can be sent quickly from source to destination, but there is no guarantee that it will get there. This is like sending mail, in which a letter is mailed to the right person but it is not known for sure whether this letter is received until another letter is sent to the initial sender as a response.

Wireless Access Protocols

NFC

Near-field communication (NFC) is a collection of protocols with a range of up to 4 centimeters. The small range and the low cost of the technology make NFC ideal for implementation in personal, everyday objects. Some applications include contactless payment systems using an NFC-enabled credit card or smartphone.

Bluetooth

Bluetooth is radio frequency wireless technology for exchanging data across short distances, removing the need for cables. Bluetooth is used for communication between devices of up to 10 meters. It can be found in various devices, appliances, gadgets, and even vehicles that, after they have been paired together, can send data to each other at high speeds. Devices include headphones and speakers, wireless keyboards, smart locks, and wristband watches.

IEEE 802.15.4

IEEE 802.15.4 is a wireless access technology for low-cost, low-data-rate devices powered by electricity or running on batteries. This networking technology is inexpensive and can support a longer battery life. It is also easy to set up because it uses a small protocol stack and is simple and flexible.

IoT Networking Protocols

Table 2.3 contains some of the latest networking protocols that IoT devices use to communicate with each other and the Internet. These protocols build upon the basic Internet networking protocols.

Table 2.3: IoT networking protocols

Protocol name	Features
 6LoWPAN	6LoWPAN is an acronym of IPv6 over Low-Power Wireless Personal Area Networks. This protocol delivers low-cost and secure IoT communications.
 ZigBee	Zigbee is based on the IEEE 802.15.4 standard and provides a simpler and less expensive way of communication than Bluetooth and WiFi. Common applications include building automation, home automation, and healthcare.
 ISA100.11a	ISA100.11a protocol is a standard for industrial automation of wireless systems, used for process control.
 WirelessHART	WirelessHART is a protocol stack for creating a time-synchronized, self-organizing, and self-healing mesh architecture.
 Thread	Thread is a set of protocols for making a safe and reliable mesh network for connecting and controlling devices, mainly at home.

IoT Communication Technologies

The various communication technologies for IoT solutions are classified by the range of information and data transmitted through them. Keep in mind that devices that use long-range communication technologies consume much more energy than their short-range counterparts.

Table 2.4: IoT communication technologies classified by distance

Distance	IoT communication technologies
Short range	A serial cable is a classic example of a wired system. Wireless short-range solutions, with a maximum distance of tens of meters between two devices, are usually a replacement for serial cables. Short-range wireless technologies include Bluetooth, Near-Field Communication (NFC) and Radio Frequency IDentification (RFID).
Medium range	This is the most common type of IoT access technology. There are various implementations in the tens to hundreds of meters range. The maximum distance between two devices is frequently less than one kilometer, but Radio Frequency (RF) technologies have no predetermined maximum distances as long as the radio signal is broadcast and received properly. Medium-range wireless technologies include IEEE 802.11 Wi-Fi. Wired technologies such as IEEE 802.3 Ethernet and IEEE 1901.2 Narrowband Power Line Communications (PLC) may also be classified as medium-range.
Long range	Long-range technologies are required for distances larger than one kilometer between two devices. Cellular (2G, 3G, 4G, 5G) and Low-Power Wide-Area (LPWA) technologies are examples of wireless technologies. LPWA communications may communicate over a broad area while requiring little power. As a result, these technologies are appropriate for battery-powered IoT sensors. IEEE 802.3 via optical fiber and IEEE 1901 Broadband Power Line Communications are both categorized as long-range but are not considered IoT access technologies.

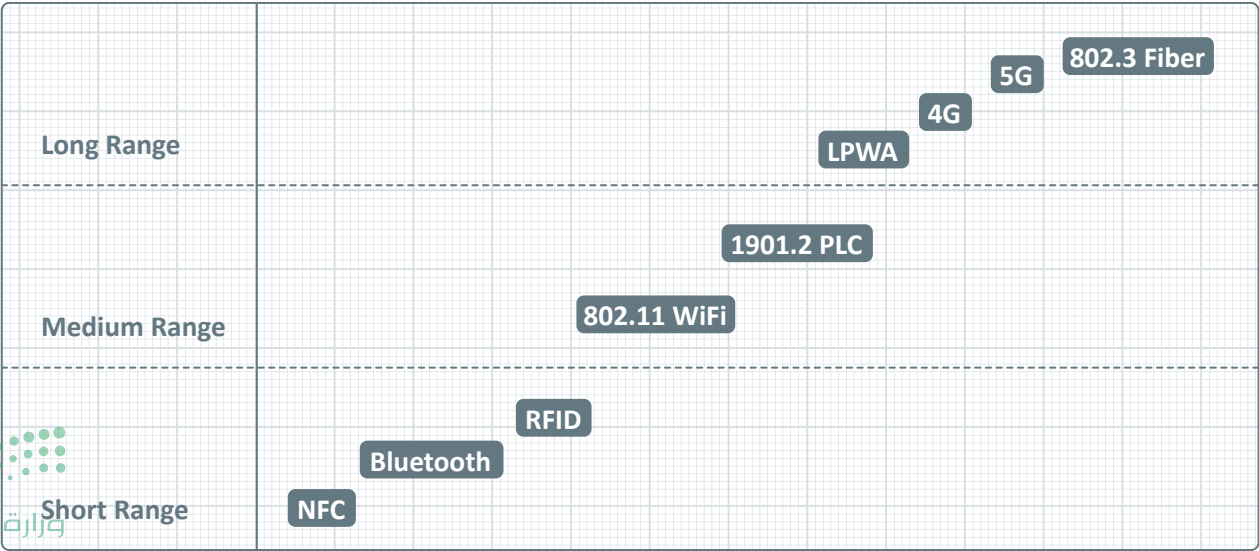


Figure 2.9: Ranges of communication technologies

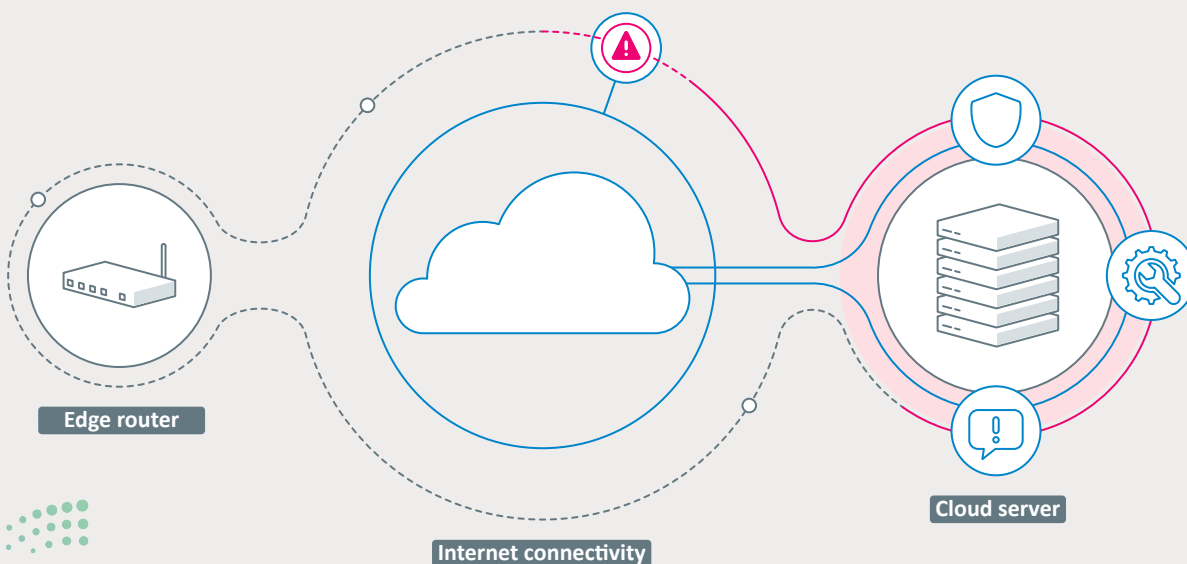
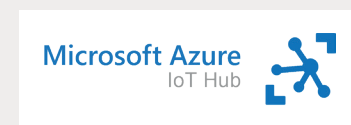
Connectivity Issues

An Internet connection can be unreliable and loss of connectivity for a short or longer period can happen unexpectedly. What should an IoT device do when the network connection is lost? The options are to drop the data or store the data locally until network connectivity is restored. The choice depends on the application, how critical the data is and the way the data is used to control actuators. It also depends on the amount of time the network connectivity is lost for.



Data can be temporarily stored locally if the connection is restored quickly, but otherwise, an IoT device cannot hold a large amount of data on its local storage. The code on the IoT device must be smart enough to know how to handle connectivity issues. Sometimes IoT devices can make limited decisions to control actuators without a connection to the main IoT application. The edge or cloud services must also be capable of functioning with incomplete or delayed data. For example, they should not send commands to the IoT device for data that are no longer relevant.

Many cloud issues include momentary loss of network connectivity or timeouts when a service is unavailable, and IoT devices can auto-recover from these problems. IoT cloud providers like AWS IoT Core and Microsoft Azure IoT Hub support the detection and handling of more persistent connectivity issues when they appear on systems implemented with these technologies. The IoT cloud services can diagnose the problem, try to provide a temporary solution, and assist in guidance for a permanent fix. The IoT system administrators are alerted when critical devices and infrastructure have issues and take action.



Exercises

1

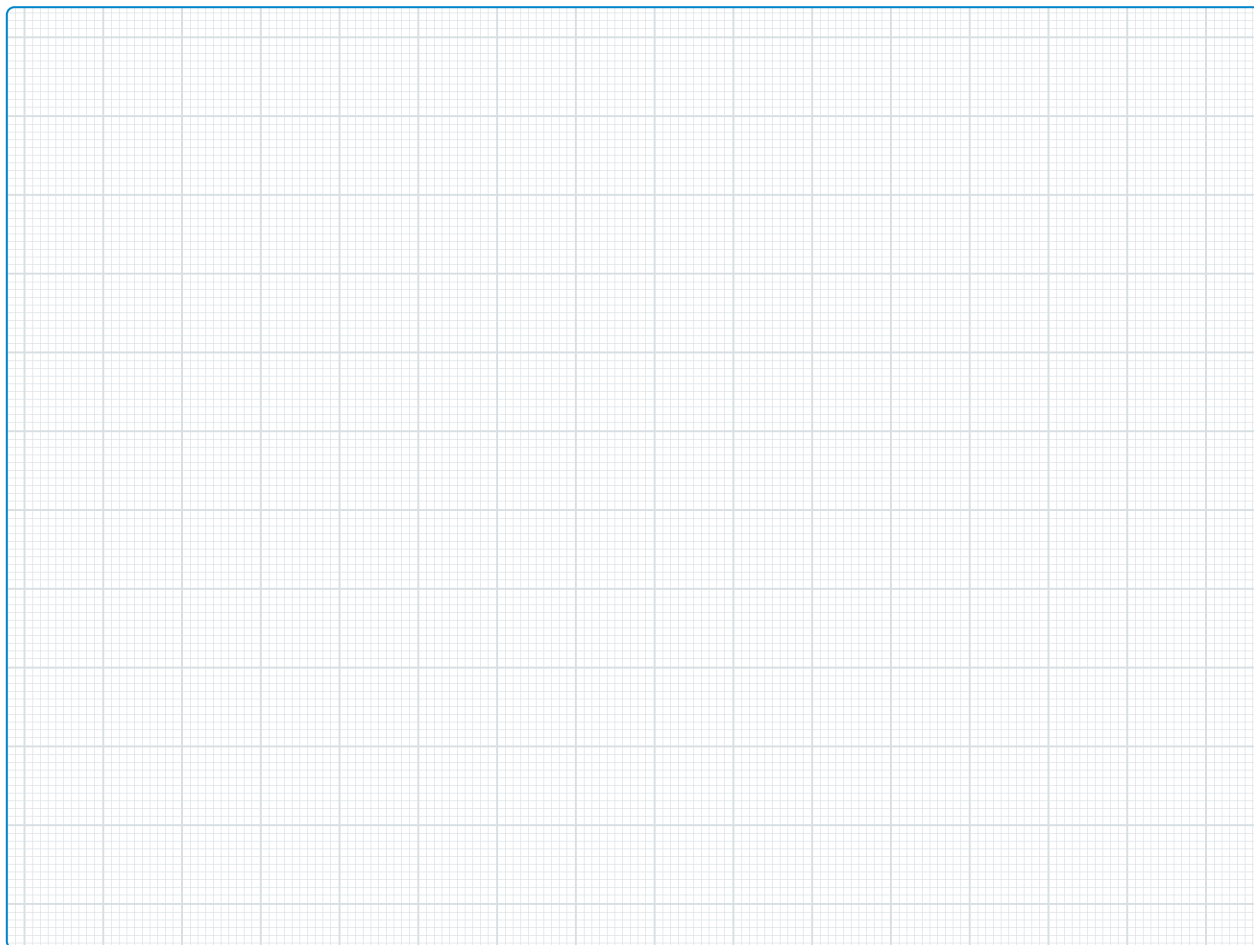
Read the sentences and tick ✓ True or False.	True	False
1. The fog layer is closer to the smart objects than the edge layer.	☐	☐
2. The Internet can connect directly to the edge layer.	☐	☐
3. The fog layer can communicate directly with cloud services.	☐	☐
4. Data processing can occur both in the fog and cloud layers.	☐	☐
5. Data transferred to a hard drive can be considered "data at rest".	☐	☐
6. Edge analytics replaces data processing in the cloud.	☐	☐
7. The UDP protocol awaits confirmation from the receiver that the package was received.	☐	☐
8. The ZigBee protocol sends more information about the sender object than other protocols	☐	☐
9. Cellular networks are used for short-range communication between smart objects	☐	☐
10. When connectivity issues occur, all the network protocols lose the data in transit.	☐	☐

2

Consider how adding the fog layer in IoT applications improves their effectiveness. Present your ideas below.



- 3 Draw a diagram to visualize the relationship between the cloud, fog and edge layers of the IoT architecture.



- 4 Discuss how edge analytics assist in effective IoT solutions.



- 5 Classify the types of applications using the TCP and UDP communication protocols, respectively.

- 6 Describe the main characteristics of the IEEE.802.15.4 network protocol that makes it so prominent in IoT applications. Present your ideas below.

- 7 Search the Internet for information about the main differences in communication between cellular networks and Bluetooth technologies.



Lesson 2

IoT Applications and Challenges

Link to digital lesson



www.ien.edu.sa

IoT Applications

IoT is one of the fastest-growing technologies, and as we move from the age of products to an age of services, IoT plays a major role in this technological revolution. The day is not too far when you will go home in a self-driving car and the entrance door will auto-detect your presence and open automatically. The following are some examples of areas where IoT changes how we live and work:



Figure 2.11: Wearable smart object

Wearables

Wearables are a very popular item on the market, but many people consider them simple gadgets. In fact, these devices are IoT devices offering a variety of functions, ranging from medical monitoring to wellness and fitness tracking. They can communicate with cloud services to provide real-time health insights to their user and even provide alerts for potential health problems.

Telemedicine

Telemedicine, or telehealth, has not yet reached its full potential. IoT-enabled telemedicine is exemplified by always-connected medical devices that healthcare professionals can monitor. Remote medical diagnosis happens proactively, saving valuable time for the proper treatment of patients. For example, heart attack detection systems can sense a person's heartbeat in real-time and send life-saving messages to a doctor.



Figure 2.12: Telemedicine application



Smart Homes

Smart homes, one of the best and most practical implementations of the IoT, significantly enhance convenience and home security. There are a variety of IoT applications for smart homes, but the most effective combine intelligent utility systems and entertainment. Home security is strengthened with improved locking systems and networked surveillance systems. As IoT evolves, we can be confident that our homes will become more intelligent. For example, an automatic lighting system will know when we are not home and conserve energy.



Figure 2.14: Saudi university implements an IoT-based security system

Education

An IoT-enabled school or campus can assist teachers and the management in the recording of daily attendance. The system can also notify the parents of absent students automatically. Other IoT devices used in the education sector are smart boards, door locks and fire or security systems.

Smart Grids

The electricity grid is a complex and critical system as it provides electricity to buildings, industries, transportation and almost everything in our daily life. A smart grid is an electricity grid that uses IoT technologies to reduce electricity waste, enhance electricity transmission efficiency, improve restoration time and reduce operating costs.



Figure 2.15: Smart grid communication



Figure 2.16: Self-Driving car on a highway

Self-Driving Cars

The big technology companies are developing versions of self-driving cars or other vehicles. For example, a telecom company has implemented a self-driving bus service for its Riyadh complex. King Abdullah University of Science and Technology is using a similar autonomous bus for its campus. Multiple sensors and embedded devices connected to the cloud generate massive data for decision-making based on machine learning algorithms. Even if it takes more years for the technology to fully develop and for countries to amend their laws and regulations, we are currently experiencing one of the fastest IoT implementations. However, since human lives are on the streets, we must ensure that the technology is ready to improve road safety.

Retail Shops

A new type of retail shop bridges the gap between online and physical stores. The IoT-enabled store adds products to your shopping cart in real-time as you select them from the shelves and facilitates cashless transactions by deducting funds from your digital wallet on your smartphone. If you change your mind and add another item to your cart, the prior item is removed and replaced with the new one. There is no cashier to charge you for your purchases, and you do not need to wait in line to pay.



Figure 2.17: Efficient digital shopping

Saudi IoT is the main regional event dedicated to the growth and development of IoT technologies. The conference focuses on current and future opportunities created by IoT technologies, applications, and services. Saudi IoT is aligned to the Saudi Vision 2030, which views emerging technologies as a requirement and the key to discovering more innovative methods of conducting business and enhancing the economy.

Smart Supply-Chain Management

Supply networks must be efficient and optimized, otherwise, they increase the cost of goods. IoT solutions for tracking items with electronic tags while in warehouses or on the road provide real-time information, reduce errors, and minimize delays in the supply chain.



Figure 2.18: Smart supply-chain networks



Figure 2.19: Industrial IoT application in manufacturing

Industrial IoT

The Industrial IoT, or IIoT, consists of sensors, instruments, and other IoT devices linked to production and energy management applications. Industry experts anticipate that IIoT will have the greatest potential of all IoT applications as it can improve product quality and production efficiency. For example, a commercial jetliner manufacturer has integrated sensors into tools and machines and given workers smart glasses to reduce errors and enhance safety in the workplace.

Smart Agriculture

The IoT will benefit the agricultural industry, and the future is undoubtedly optimistic. Tools for drip irrigation, recognizing crop patterns, water distribution and drones for farm surveillance are being continuously developed. These innovations enable farmers to become more productive and address concerns more effectively.



Figure 2.20: Smart garden IoT application

Smart Transportation

The Riyadh Metro and Riyadh Rapid Bus Transit System is a new public transportation system that enables individuals to navigate the city easily. Riyadh Metro uses IoT-enabled driverless trains and provides consumers with integrated solutions that enhance their commuting experience. The project includes a state-of-the-art center for monitoring and controlling surveillance systems for stations, lines, and other infrastructure.

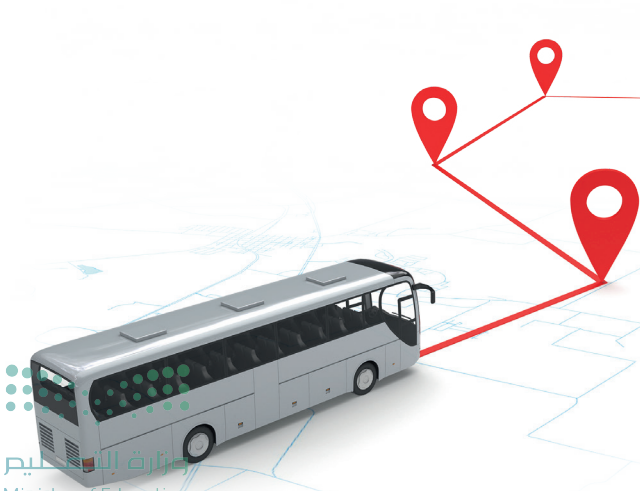


Figure 2.21: Smart transportation networks

Traffic Management

With the help of IoT technologies, traffic management in large cities can be enhanced. Using mobile phones as smart objects with their GPS sensors and apps such as Google Maps, and information from our vehicles through cloud systems like the Local Hazard Information system, IoT technologies can improve traffic and road safety. Long-term analysis of traffic patterns is another IoT application. Commuters will be better equipped to avoid traffic and delays if they are informed of alternative routes during busy hours.



Figure 2.22: Highway IoT application



Figure 2.23: Traffic monitoring with IoT

To improve road safety and traffic management, the Ministry of Transport and Logistic Services is actively pursuing the implementation of intelligent transport systems on its highways. The road and highway networks will be equipped with IoT systems to control the flow and efficiency of traffic. The ministry's plan defines the primary framework for the future implementation of intelligent transport systems.

Water/Waste Management

Numerous municipalities are implementing water recycling through the use of water treatment units. Using an IoT application, it is possible to determine how much water is consumed in a particular location, how much wastewater is being created, and how waste production has changed over time. With a smart waste management system, authorities can forecast, among other things, how much waste will be generated in a particular region, how it will be processed, when it will be cleared, and how to evaluate data for future planning. Similarly, the volume of garbage generated in each neighborhood over time can be analyzed. All this information can be utilized to plan the city's improvement initiatives. Data analytics systems enable optimal trash collection planning and forecasting of future trends.



Figure 2.24: Smart waste management minimizes solid waste

Example

More than two million Muslims gather yearly in Makkah during Ramadan and Hajj, and the local authorities must handle the increased waste generation. The Vision 2030 strategy aims to minimize all types of waste by implementing waste management systems. An effective solid waste management system based on IoT technologies improves waste collection at the source. Saudi universities propose the use of smart containers that allow the automation of waste classification for recycling.

The Importance of the Internet of Things Now and in the Future

For decades, humans have been required to input data into computer systems. This data is processed to automate tasks or generate valuable information for decisions that must be taken. IoT technologies are changing this computing paradigm because nowadays, computers are autonomously perceiving environments and gathering data through their sensors. IoT technologies represent a significant technical transition, but there are also considerations regarding its implications for future technological advances.

IoT changes how individuals, systems and organizations interact with their surroundings. Using real-time connectivity to manage and monitor intelligent devices offers a new level of data-driven decision-making. This leads to optimized systems and processes and the delivery of new services that save individuals and businesses time and enhance overall quality of life. The number of smart objects as separate IoT devices or embedded in existing physical objects in everyday life will increase enormously in the coming years. Since 1990, when the first IoT device was created, a toaster connected to the Internet, we have managed to convert almost any possible physical object into a smart object.

IoT Analytics, an IoT market research organization, estimates that we are now using 14 billion IoT devices worldwide and projects this number to reach 27 billion by 2025. These numbers do not include computers and smartphones, very simple sensors, or single-directional communications devices. Industry solutions in energy, water and waste management, retail and wholesale, and transportation will drive the adoption of IoT technologies.

Technological Trends in Smart Objects

- **Size is decreasing:** Microcontrollers and sensors continue to shrink in size. Some of them are so small that they cannot be seen with the human eye. This reduced size makes integrating intelligence and network connectivity into commonplace physical objects easier.
- **Power consumption is decreasing:** The hardware components of an IoT device require less energy over time. This is important for sensors, as many of them are entirely passive. Some battery-powered sensors have a battery life of 10 years or longer.
- **Processing power is increasing:** CPUs continue to increase in processing power and decrease in size. This is a significant development for smart objects, which are becoming increasingly sophisticated with local capabilities like, as we have discussed, edge analytics.
- **Communication capabilities are improving:** In addition to speed increases in data transfer, wireless connections are also improving in range while keeping power consumption low.
- **Communication is increasingly standardized:** The IoT promotes the development of increasingly specialized communication protocols that support various use cases. The industry is making a great effort to build open standards for IoT communication protocols.

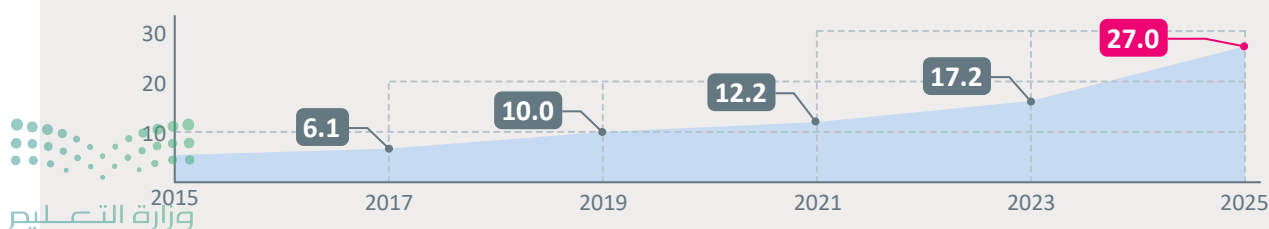
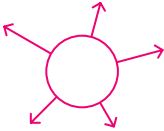


Figure 2.25: Global active IoT connections in billions, excluding computers, laptops, smartphones or tablets.

Challenges of Internet of Things Systems

While the vast array of IoT components is intended to generate significant productivity and automation benefits, it also poses new problems. Numerous aspects of IoT have become a reality, but some challenges must be addressed for IoT to become prevalent in various industries and in our daily lives. Table 2.5 shows some of the most common problems and challenges faced by every technological advancement, including IoT systems.

Table 2.5: Common IoT challenges

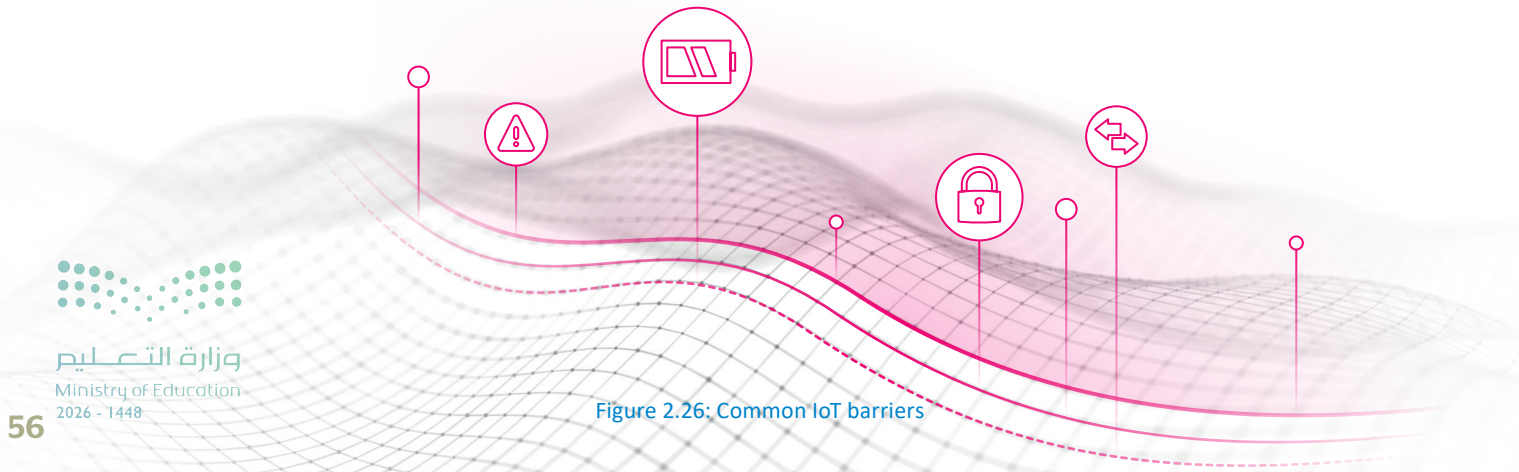
Challenge	Description
 Scale	Traditional IT networks might be big, but IoT networks can be several orders of magnitude larger. As the number of devices in a system increases, the complexity of the connections increases and scaling becomes an issue. Problems arise in delayed response and processing time as networks scale making it difficult to manage real-time systems.
 Security	As more smart objects are connected to one another and to users, IoT security becomes an increasingly difficult task. The security risk has increased dramatically, and if a device's connectivity is compromised, it poses a big problem. A hacked device may be used to attack other devices and systems.
 Privacy	As sensors become more prevalent in our daily lives, a growing share of the data they collect will be specific to individuals and their actions. This data may include health information, purchasing patterns, and retail activities. Businesses can monetize this information, and companies continually debate who owns it and how individuals can control access to it.
 Big data and data analytics	The IoT and its various sensors will generate a massive amount of data that must be analyzed. If efficiently processed, this data will yield vital information and insights. The problem, however, is how to evaluate massive amounts of data from many sources and formats before they become worthless.
 Interoperability	As with any developing technology, IoT protocols and architectures strive for market share and standardization. Some protocols and implementations are based on commercial components, while others are based on open standards.

Other IoT Barriers

The Table below shows current obstacles that slow down the deployment and development of IoT systems. Technology evolves, best practices and experience from failures are integrated into new systems and government regulation provides security and privacy guidelines to overcome these barriers.

Table 2.6: Common IoT barriers

IoT barriers	Examples
Deployment of IPv6	The IoT connects billions of small devices, each of which must have a unique IP address. Only IPv6 can support the current number of IoT devices. The migration to the new IP standard delays the rapid development of the Internet of Things ecosystem, and it will result in an increase in network security vulnerabilities.
Sensor Power Consumption	IoT sensors must be autonomous. Changing batteries on the billions of deployed devices is time-consuming, and the sensors must also be power-efficient to avoid rising energy costs.
Legal and Regulatory Issues	Utilizing IoT devices causes several legal complications and magnifies numerous Internet-related privacy issues. One issue includes the transmission of data across international borders.
Flexibility and Evolution of Applications	Every day sensors and devices continue to expand their capabilities. This results in the development of new, enhanced services. Modern apps with an intuitive user experience that support these services are becoming more complex and demanding for developers and user experience designers.
Integration of Data from Multiple Sources	Several data sources, including sensors, mobile devices, social network feeds, and other web resources, are streamed in IoT applications generating data in a variety of formats. It is increasingly difficult to filter and process this data effectively.



IoT Security Issues

One of the biggest issues arising from the universal use of the Internet, the rapid increase of devices in the IoT and the use of the cloud is the security of this entire digital global environment. Data networks have existed for decades, but most were publicly inaccessible, with special security protocols and access devices. The connection of billions of devices on data networks that are eventually part of the Internet is the reason for increased security breaches. IoT devices may simply switch on and off street lights to conserve energy, but they may also interact with sensitive data like personal medical data. It is imperative to address these security concerns from the beginning of the system design. IoT networks are exposed to a greater variety of attacks than other networks, and the quality and complexity of these attacks are increasing daily.

IoT systems should ensure user interactions are conducted in a secure environment. To avoid unwanted access to private data, IoT security experts should consider the following:

- Trustworthy and decentralized authentication models
- Technologies for encryption and data protection that save energy
- Cloud computing security and trustworthiness
- Data control
- Legal and liability concerns
- Communication and networking vulnerabilities
- Access and user rights and rules for sharing value additions
- Secure, inexpensive hardware
- Privacy policies management



Figure 2.27: IoT security standards

Examples: Security Issues with RFID Technologies

Radio-Frequency Identification is one of the most commonly used communication protocols for processing identification data for smart objects. RFID employs radio frequency waves to interact and exchange data without needing physical contact. Two components comprise an RFID system: transponders (RFID tags) and transceivers (RFID readers). The Electronic Product Code (EPC) is the unique identifier of the smart object. RFID tags are characterized as either active or passive. The inbuilt battery of an active tag permits the remote interaction of its unique EPC with surrounding EPCs at a limited distance. Passive tags operate without a battery, and data is read only when a transceiver within a short range activates the tag. Unfortunately, the data within the RFID is vulnerable to tampering, corruption, and deletion, although cryptographic techniques are used to ensure privacy and proof of originality of the transferred data.

Table 2.7: Examples of security vulnerabilities of IoT systems through RFID exploitation

Security vulnerability	Attack example
Attack on Authenticity Unauthorized Tag Disabling	These attacks cause RFID tags to malfunction and behave incorrectly when an RFID reader scans them. The EPC flags false information that does not match the RFID tag's destination properties. Typically, this attack is conducted remotely, allowing the attacker to modify the tag's behavior from a distance.
Attack on Integrity Unauthorized Tag Cloning	The theft of EPC identity information and modification of tags by unauthorized RFID readers comes under this category. Once the identity information of a tag has been obtained, it is easy to replicate the tag, which can be used to circumvent or counterfeit security mechanisms and create new vulnerabilities during automatic verification processes.
Attack on Confidentiality	Tracing an RFID tag through unauthorized readers may result in the disclosure of sensitive information.
Attack on Availability Denial of Service (DoS) Attack	Jamming the system with noise interference, blocking radio signals, or removing or disabling RFID.



Security Issues with Wireless Sensor Networks Technologies

Wireless sensor networks (WSN) are responsible for transferring data and information between smart objects in IoT systems. They are composed of autonomous nodes that communicate with limited frequency and capacity. The communication node consists of a battery, sensor, memory, radio transceiver, and microprocessor. Due to the limited communication range of each sensor node, information is relayed between the source and the base station in multiple stages. Wireless sensors gather and transmit the required data in coordination with other nodes for routing to the central system. Wireless sensors have limited power and computing capabilities, making many traditional security methodologies difficult or impossible to implement.

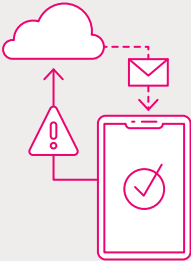
Wireless sensor network

A wireless sensor network (WSN) consists of dispersed autonomous sensors monitoring physical or environmental conditions that collectively transmit data to a central location.

Security and Privacy Concerns

Any connected device could be a potential open door to the IoT infrastructure or sensitive data. Concerns about data security and privacy are very important as complexity can create more vulnerabilities related to the provision of IoT services.

Table 2.8: Security concerns based on IoT system levels

IoT system levels	Security concerns
 Device level	IoT devices must prove their identity to maintain authenticity and limit locally stored data to protect privacy. Because IoT devices are present everywhere in the environment, physical security is also important. This creates the need to design resistance to device breaches so that it is difficult to extract sensitive items such as personal data, cryptographic keys, or credentials. To support a long service life, software updates must be frequently applied.
 Network level	This level of the IoT system represents connectivity and messaging between IoT devices and cloud services. Internet communications are usually a combination of private and public networks, so securing traffic is critical. Many IoT devices also communicate through protocols other than WiFi. The IoT gateway is responsible for maintaining confidentiality, integrity and availability when translating between different wireless protocols.
 Service level	This level represents the IoT management system and is responsible for managing devices and users, implementing policies and rules, and coordinating automation across devices. Role-based access control to manage the identity of users and devices and the actions they are authorized to take are critical at this level. Tracking of actions must be enabled to ensure that potentially compromised devices can be identified when abnormal behavior is detected.
 Data level	Large data analysis of aggregated data generated by IoT is often described as the most valuable aspect of IoT for service providers. Maintaining data privacy is a top priority for government agencies with the Communications, Space and Technology Commission (CST) in Saudi Arabia, the Federal Trade Commission (FTC) in the USA and the European Union Agency for Cybersecurity (ENISA) in the European Union, which release the corresponding privacy-related security requirements guidelines.

Approaches to Solving Security Challenges

Security must be considered during the design phase of an IoT system. Building security throughout the design phase begins at the hardware level, communication infrastructure, the operating system level, followed by the design level, and should be extended until application deployment. Companies and state organizations should enforce data protection policies and comply with their country's legislation. Specialized network and security engineers, with experience in IoT systems design, test and secure IoT devices and networks by implementing best practices in cybersecurity. These engineers must combine technical knowledge and field experience from various computing domains.

Privacy Concerns

The concept of privacy differs in different cultures and has evolved and transformed over time. Initially, installing surveillance cameras was considered intrusive; however, this practice is now widespread and accepted. The Internet of Things is a blend of public and commercial applications, and who will have access to and authority over the collected data is a concern. In IoT systems, privacy should be enforced over personally identifiable information, and storage and disclosure restrictions should be imposed. An adequate privacy and protection framework must be in place, and consumers must be guaranteed that their data is private and secure.



Governmental Regulation

The rapid growth of IoT applications has raised several concerns around data privacy issues and malicious attacks that can disrupt critical operations of the industry or government. Governments worldwide have started to focus on solving this problem with regulation and legislation initiatives covering IoT ecosystems.



هيئة الاتصالات والفضاء والتقنية
Communications, Space &
Technology Commission



IoT Regulatory Framework

The Kingdom of Saudi Arabia aims to become a leading country in developing and applying IoT technologies and services. The Communications, Space and Technology Commission (CST) has developed the IoT Regulatory Framework to regulate IoT service provisioning requirements to support this endeavor. The framework defines regulations specific to IoT equipment, IoT identifiers, such as IP addresses, that uniquely identify objects to simplify communications, and other IoT technologies. In addition, the IoT Regulatory Framework includes other fundamental standards for IoT service providers, such as communication with end-users regarding the significance of network and data security and data protection guidelines.



وزارة التعليم

Ministry of Education

2026 - 1448



Figure 2.28: CST IoT downloadable documents

Emerging Technology Regulatory Sandbox

CST has also established the Emerging Technology Regulatory Sandbox to develop and deliver innovative applications in Saudi Arabia. The initiative is part of the authority's responsibility to oversee and control the ICT sector, including licensing and regulating telecommunications applications that integrate IoT technologies. This regulatory sandbox aims to support, facilitate, and sustain the expansion of the IoT application development ecosystem in Saudi Arabia and benefit all sector stakeholders, including companies and customers.

The Emerging Technology Regulatory Sandbox tries to decrease the time required to bring applications to market and reduce the cost-of-service delivery while application developers can test novel products and services in a controlled environment. CST also contributes to developing an ecosystem hospitable to innovation by improving access to funding. These efforts align with the Saudi Vision 2030 to assist the growth of the private sector and the creation of technology-related jobs.



Exercises

1

Read the sentences and tick ✓ True or False.	True	False
1. Telemedicine is one of the IoT applications that is on the decline.	☐	☐
2. Smart grid IoT applications are used only to save energy.	☐	☐
3. The most critical innovations in IoT technologies have occurred in the last 20 years.	☐	☐
4. Smart objects will consume more energy in the future.	☐	☐
5. Implementing the same security standards for all IoT systems is the least complex problem in IoT systems.	☐	☐
6. The IPv6 protocol will not be able to support the expected number of future smart objects.	☐	☐
7. The amount of input data sources for smart objects will increase.	☐	☐
8. RFID and WSN technologies are invulnerable to DoS attacks.	☐	☐
9. IoT applications and systems are unregulated by government authorities.	☐	☐
10. Privacy of IoT data is a significant concern to governments and organizations.	☐	☐

2

Compare and contrast IoT-enabled self-driving cars and smart public transportation. Are these IoT applications separated or complement each other? Present your ideas below.



- 3 Solar or wind renewable energy sources introduce variability in power generation. Do you think smart grid technologies can make the distribution and management of renewable energy more effective? Present your reasons below.

- 4 Identify which technological trend in smart objects will be the most important in developing IoT systems. Present your ideas below.



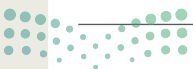
- 5 Discuss which of the common IoT challenges is the most difficult and expensive to overcome. Present your findings below.

- 6 Search the Internet for an event where a security vulnerability led to a cyberattack on an IoT system. What were the damages caused, and how could it have been prevented? Present your findings below.



7 Consider the most vulnerable point in Wireless Sensor Networks and how it affects IoT systems. Present your ideas below.

8 Describe what actions the CST board took to regulate the adoption of IoT applications in the Kingdom of Saudi Arabia.



Project

IoT applications are complex systems with many technical and operational layers to function properly and effectively.

1

Choose an industry where IoT is commonly used but is vulnerable to cyberattacks and data exploitation. Describe how a security vulnerability could be used to attack this system and what the repercussions are for the end users.

2

Create a PowerPoint presentation that describes the industry that you have chosen, illustrates the vulnerability problem, and contains a proposal for a solution to this problem.



Wrap up

Now you have learned to:

- > Distinguish cloud, fog, and edge layers for IoT applications.
- > Classify the main technological enablers of IoT systems.
- > Classify the networking protocols and systems that connect IoT applications.
- > Identify how IoT applications are used in the economic and government sectors.
- > Describe how IoT applications will evolve in the near future.
- > Define the main technological challenges that IoT engineers are facing.
- > Distinguish the security concerns of IoT systems.
- > Describe how IoT applications are currently regulated.

KEY TERMS

Authentication

Authorization

Data at Rest

Data in Motion

**Denial of Service (DoS)
Attack**

Edge Analytics

Edge Computing

Edge Device

**Electronic Product Code
(EPC)**

Endpoint

Fog Computing

Gateway

Internet Protocol (IP)

IoT Enabler

IoT Services

IPv6

Latency

**Near-Field
Communication (NFC)**

Protocols

**Radio Frequency
Identification (RFID)**

Regulations

Regulatory Framework

Smart Grid

**Transmission Control
Protocol (TCP)**

**User Datagram Protocol
(UDP)**

**Wireless Sensor
Networks**

3. Building IoT Applications with Arduino

In this unit, you will learn about the basic properties of an Internet of Things application and how to design and build practical applications using an Arduino microcontroller and the Tinkercad Circuits environment.

Learning Objectives

In this unit, you will learn to:

- > Recognize the components of microcontrollers and how to program them.
- > Measure data collected from various input sensors.
- > Understand how data from sensors and algorithms from programs work together.
- > Use data from sensors to trigger alerts and automated responses.
- > Utilizing a gas sensor to create a gas leak alarm for environments with hazardous conditions.
- > Design IoT circuits with an Arduino microcontroller in Tinkercad Circuits.
- > Program the Arduino microcontroller using the block-based coding language in Tinkercad Circuits.

Tools

- > Autodesk Tinkercad Circuits





Arduino Microcontrollers

Microcontrollers are designed for single-board minicomputers rather than for desktop or personal computers. For example, Arduino produces entirely self-contained microcontrollers, often with a set of built-in microprocessors. Arduino microcontrollers are characterized by their extensive functionality. However, they are limited in functionality compared to personal computers, as they are designed to perform only simple tasks.

Let's take a look at the most commonly used Arduino boards:

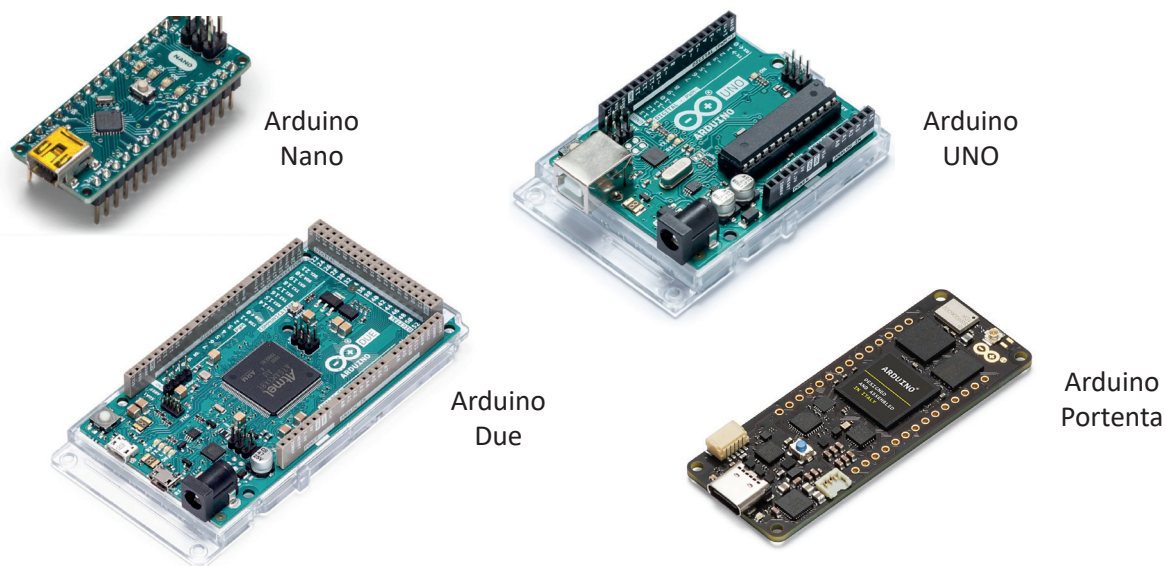


Figure 3.1: Arduino microcontroller boards

Table 3.1: Specifications of Arduino models

Model	USB Type	I/O	Clock speed	Flash memory	SRAM memory
Arduino Nano 33	Mini-B	26 pins	48 MHz	256 KB	32 KB
Arduino UNO R3	Type B	20 pins	16 MHz	32 KB	2 KB
Arduino Due	Micro-B	68 pins	84 MHz	512 KB	96 KB
Arduino Portenta H7	Type C	80 pins	480 MHz	up to 128 MB	up to 64 MB

INFORMATION

Compared with personal computers or laptops, microcontrollers can be manufactured quickly and easily reducing the production cost significantly.



Arduino UNO R3

The Arduino UNO R3 is a microcontroller board based on the ATmega microcontroller.

It has 14 digital input/output pins, of which 6 can be used as PWM (Pulse-Width Modulation) outputs and 2 are used for transmitting (Tx→1) and receiving serial data (Rx←0), 6 analog inputs, a USB connection, a power jack, and a reset button.

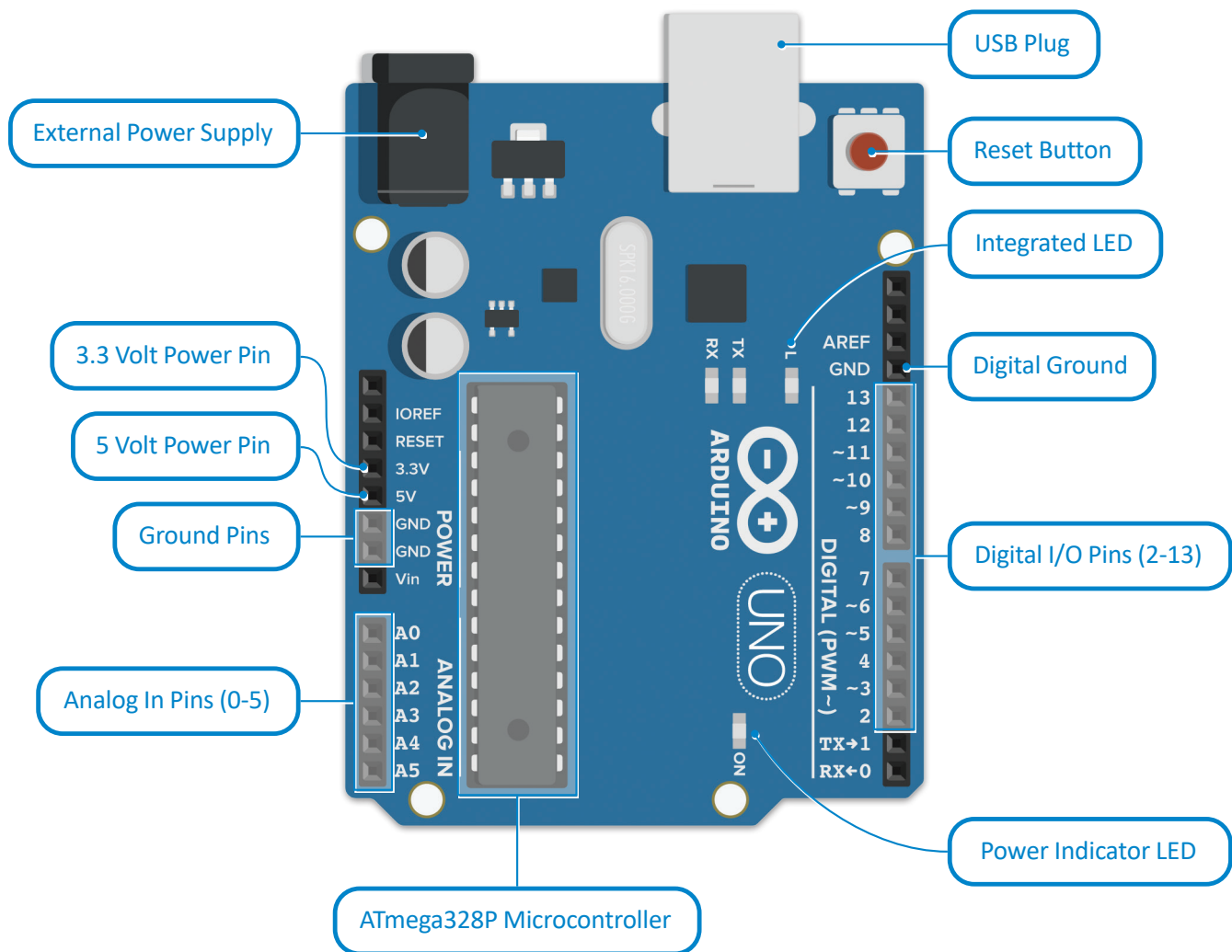
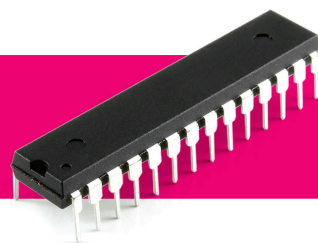


Figure 3.2: Arduino Uno R3

The ATmega328P is a single-chip microcontroller that is frequently found in Arduino products. It achieves high performance with low power consumption.



Some Examples of External Sensors for Microcontrollers

Soil Moisture Sensors

Soil Moisture Sensors are sensors that measure the soil's volumetric water content. Since direct measurement involves a significant amount of soil processing, soil moisture is measured indirectly utilizing other properties of the soil, such as electrical resistance (the greater the moisture the lower the resistance). Soil Moisture Sensors are crucial in agriculture and are used for monitoring applications such as irrigation control in industry, watering house and office plants as well as landscape irrigation.

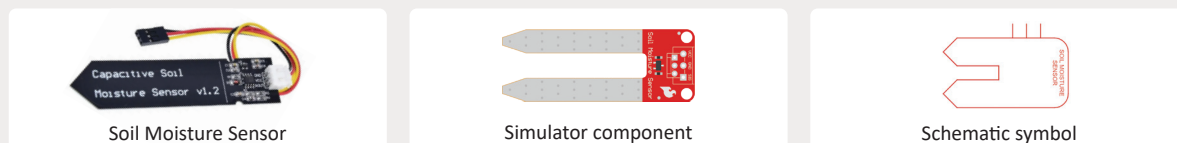


Figure 3.3: Soil Moisture Sensor

Temperature Sensors

The TMP36 temperature sensor is a temperature measuring device that generates an analog output voltage proportional to the temperature which can be simply converted into a temperature reading in degrees Celsius. This sensor can measure temperatures in the range of -40 to 125 C°. The TMP36 is used mainly in applications involving thermostats and temperature measurement, and since it doesn't need external calibration, it can be used without any additional parts.

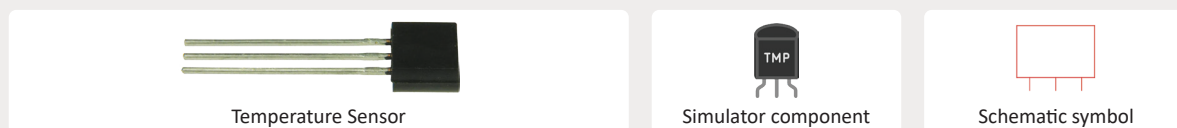


Figure 3.4: Temperature Sensor

PIR Sensors

Passive Infrared (PIR) sensors are electronic sensors that can detect objects in a specific Field Of View (FOV). They work by measuring the infrared radiation signals that are present in the FOV that they examine.

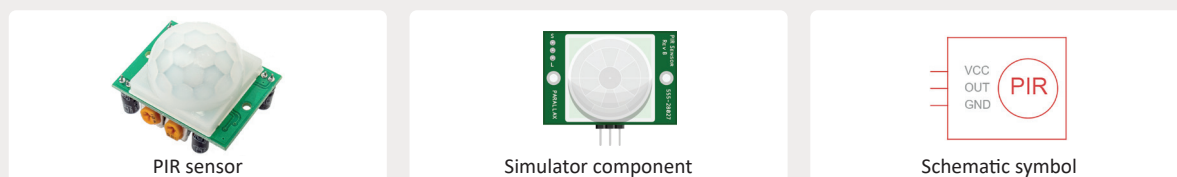


Figure 3.5: PIR sensor

Gas Sensors

Gas Sensors are chemical resistors that detect high levels of smoke and other gases, such as propane, hydrogen, and carbon monoxide. When the gas comes into contact with the resistor's material, its electrical resistance changes. These sensors can detect concentrations between 200 and 10,000 ppm. They are used for monitoring areas where there is a risk of fire or poisonous gas emissions.

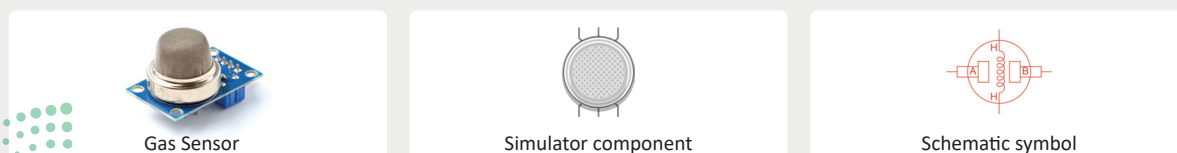


Figure 3.6: Gas Sensor



Build a Smart Home System

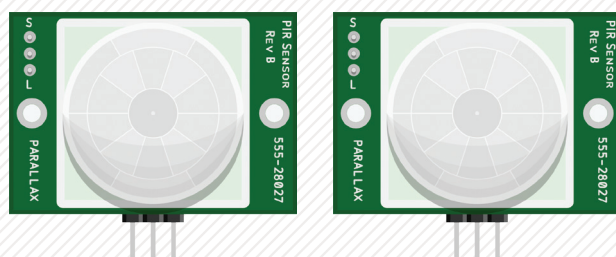
Every day, houses become increasingly equipped with smart sensors to enhance the quality of our lives and make our activities easier. One "smart" technology is implementing home lights that automatically turn on and off by sensing a human presence in the room.

In this example, you will use an **Arduino UNO R3** microcontroller to simulate, in Tinkercad, an automatic room lighting system for implementation in a Smart Home. You will use 2 PIR sensors that detect if an object has entered the field of view (FOV) in either of two rooms. When there is a person in the FOV of a sensor, its LED will light up and when they leave, the LED light will turn off. The second sensor will represent another room waiting for a person to enter.

For this project you will use the following components:

- 1 Arduino UNO R3
- 2 Passive Infrared (PIR) Motion Sensors
- 2 LEDs
- 2 Resistors
- 1 Breadboard Small

Components that you will use in this project

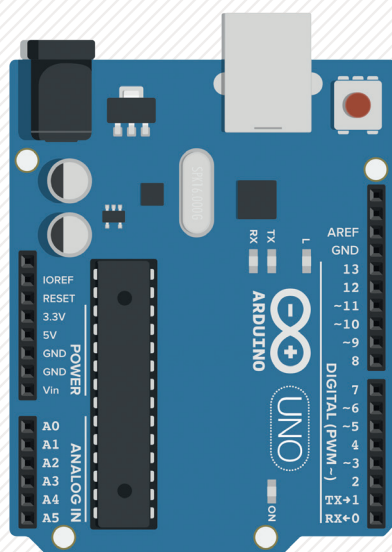


PIR Sensors

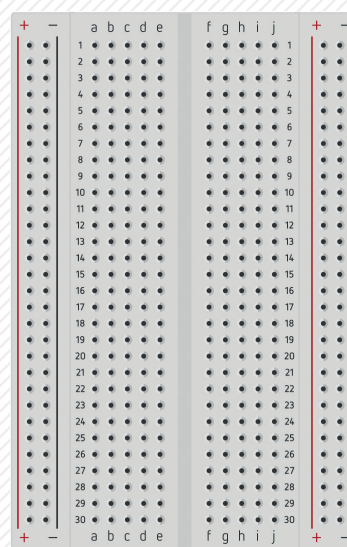


Resistors

LEDs



Arduino Uno R3



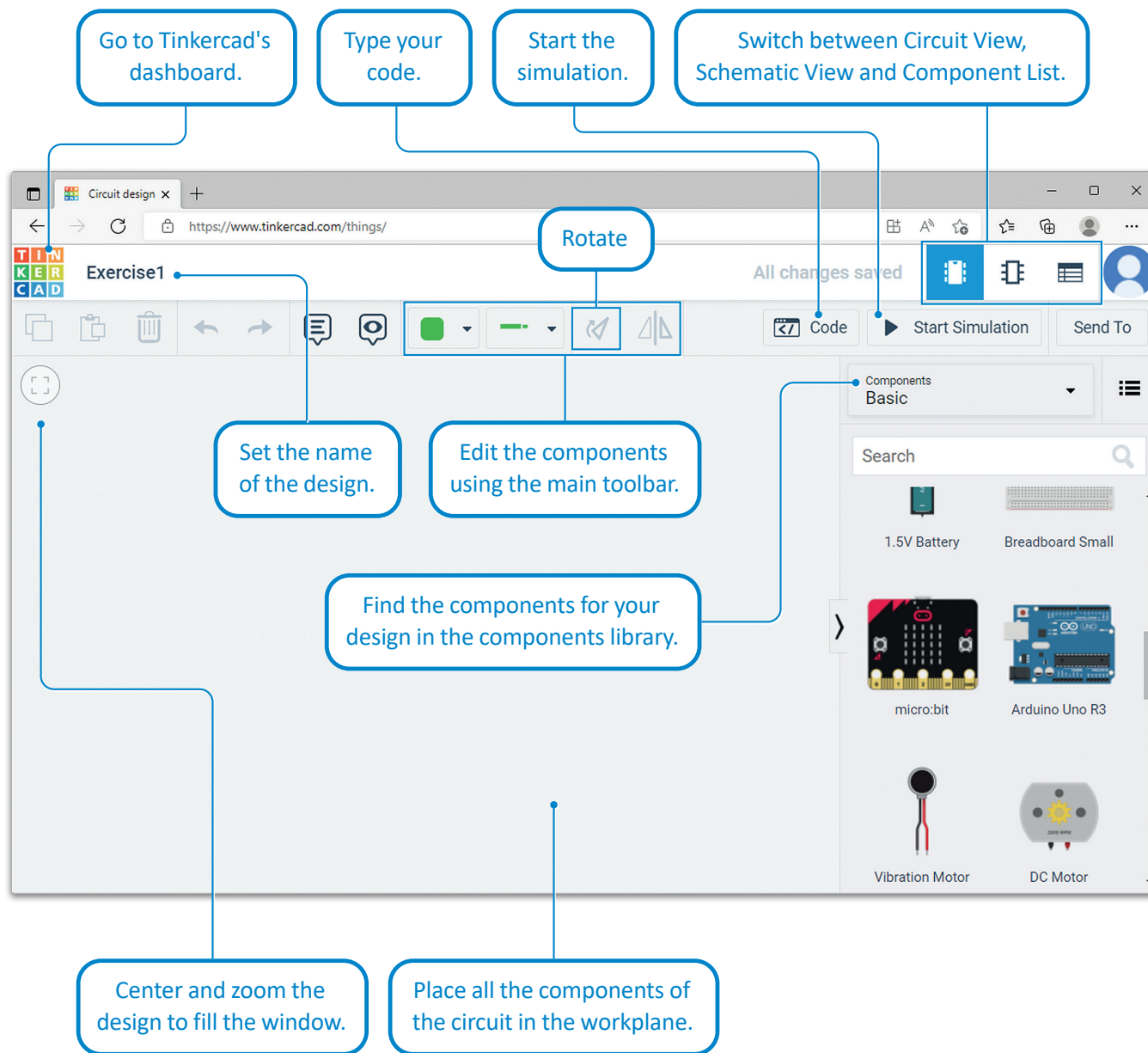
Breadboard Small

Figure 3.7: Smart Home project components

Now, let's begin a new circuit design in Tinkercad.

Tinkercad Circuits is a web-based simulator to create and program prototypes of electronic circuits with microcontrollers like Arduino.

Start Tinkercad Circuits at <https://www.tinkercad.com> and select Circuits to open the main window of the simulator.



Add a component to the design by clicking on a component in the Components library and then clicking anywhere on the workplane.

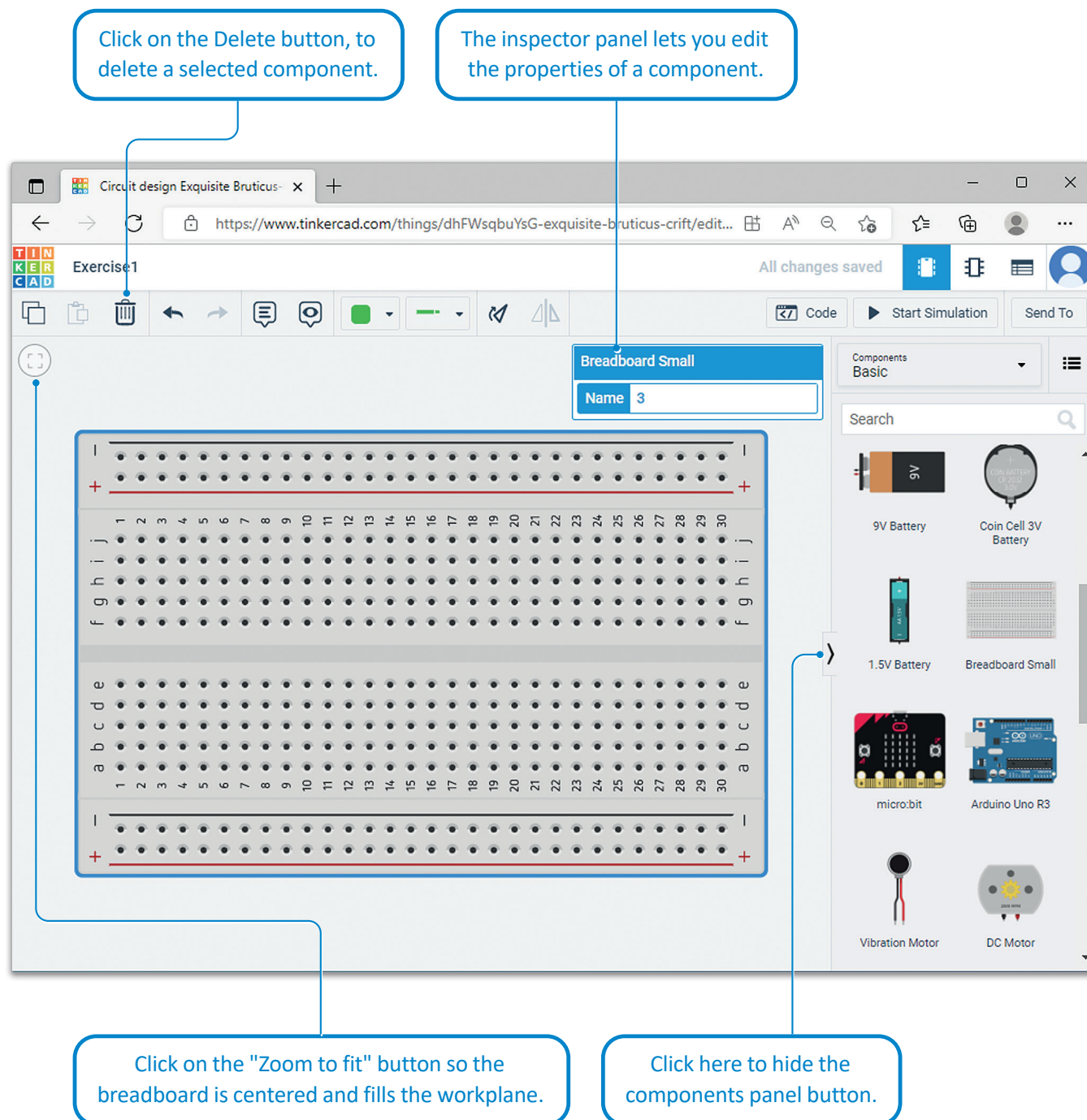


Figure 3.9: Edit components

Find the components that you will need for this project, and move them to the workplane. Find and add an Arduino UNO R3, two PIR sensors, two LEDs and two resistors, and a Breadboard Small in the workplane.

To add components to the workplane:

- > Find the **Arduino UNO R3** in the **components library** ¹ and drag and drop it in the **workplane**. ²
- > Find the **Breadboard Small** in the **components library** and drag and drop it in the **workplane**. ³

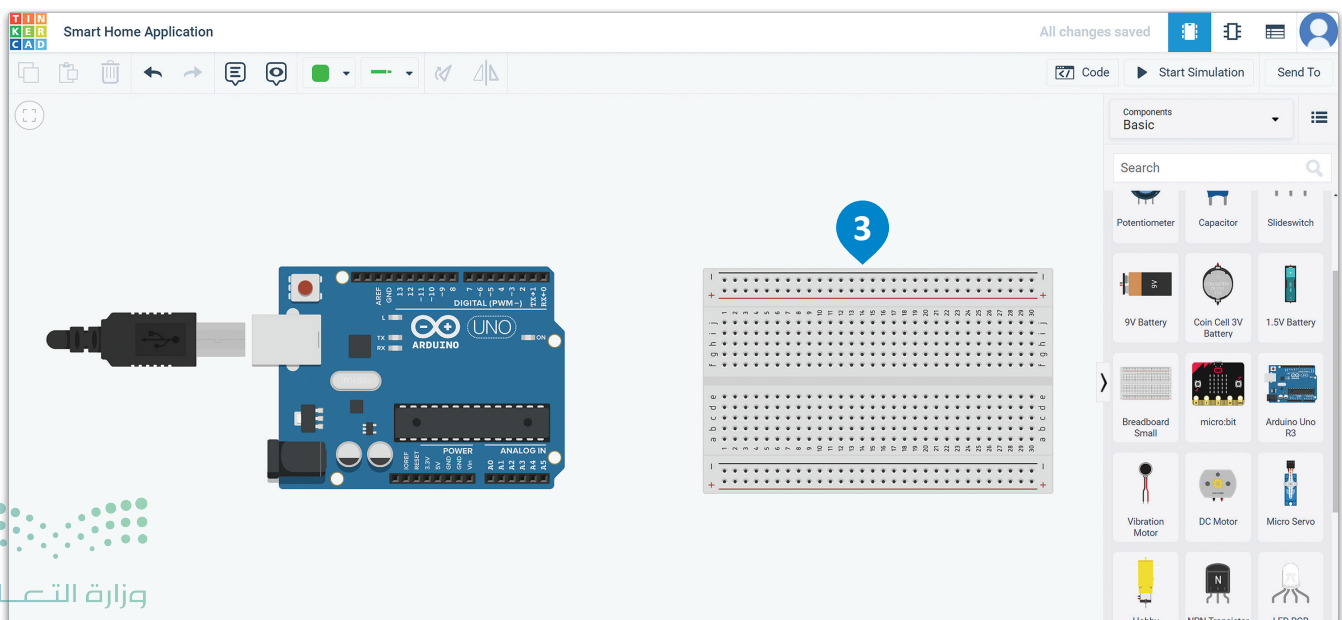
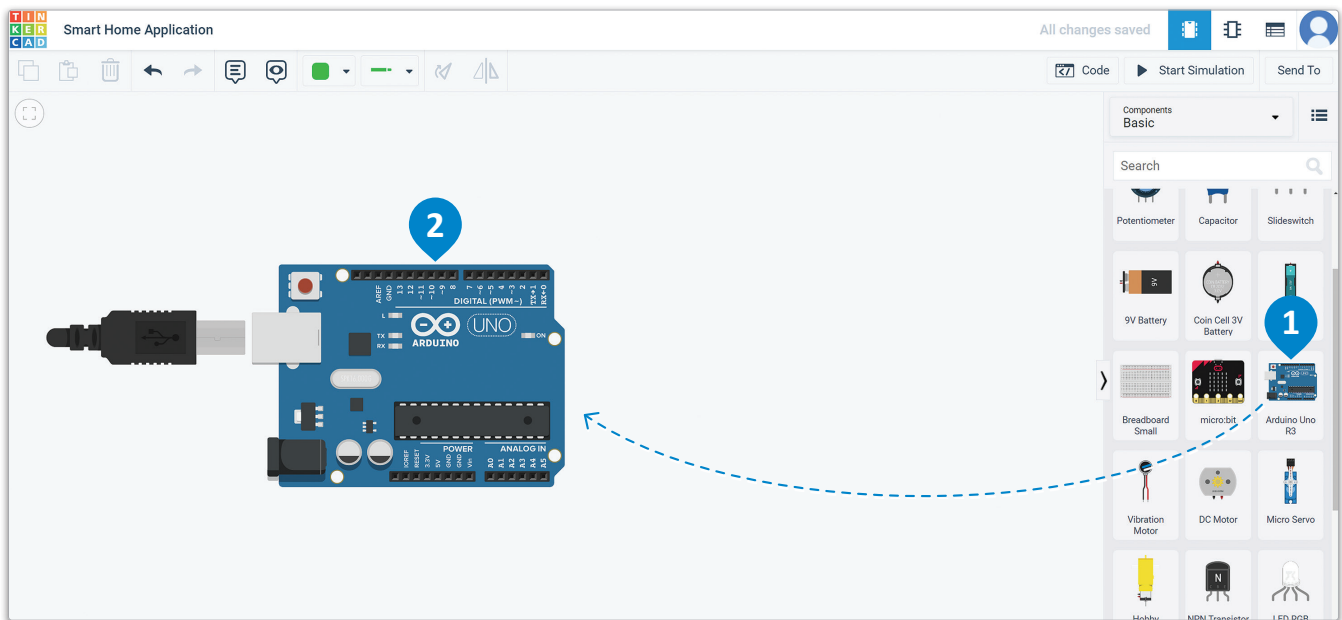


Figure 3.10: Add components to the workplane

Now, you will add two resistors to the breadboard.

To add resistors to the breadboard:

- > Drag and drop the **resistors** from the **components library** onto the **Breadboard**. **1**
- > From the inspector panel, set **Resistance** for both resistors to **220** **2** and set the **unit** **Ω** . **3**

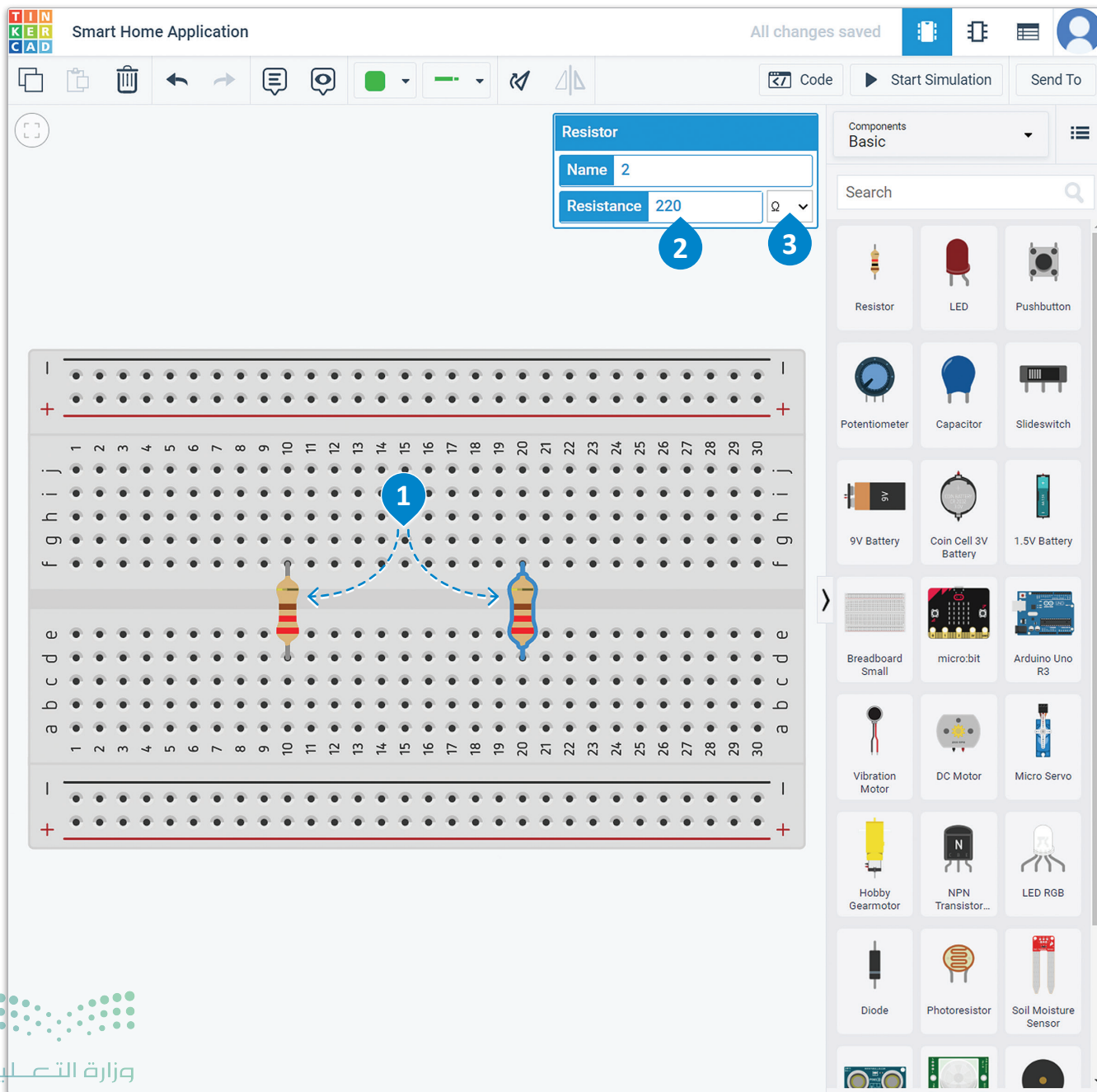


Figure 3.11: Add resistors to the breadboard

Continue by adding two LEDs to the breadboard and two PIR Sensors.

You need to connect the anode of each LED in series with its corresponding resistor.

To add components to the breadboard:

- > Drag and drop the **PIR Sensors** from the **components library**, and place them on the **Breadboard**. **1**
- > Drag and drop the **LEDs** from the **components library**, and place the **anodes** of each LED in parallel with the corresponding resistor in the **Breadboard**. **2**

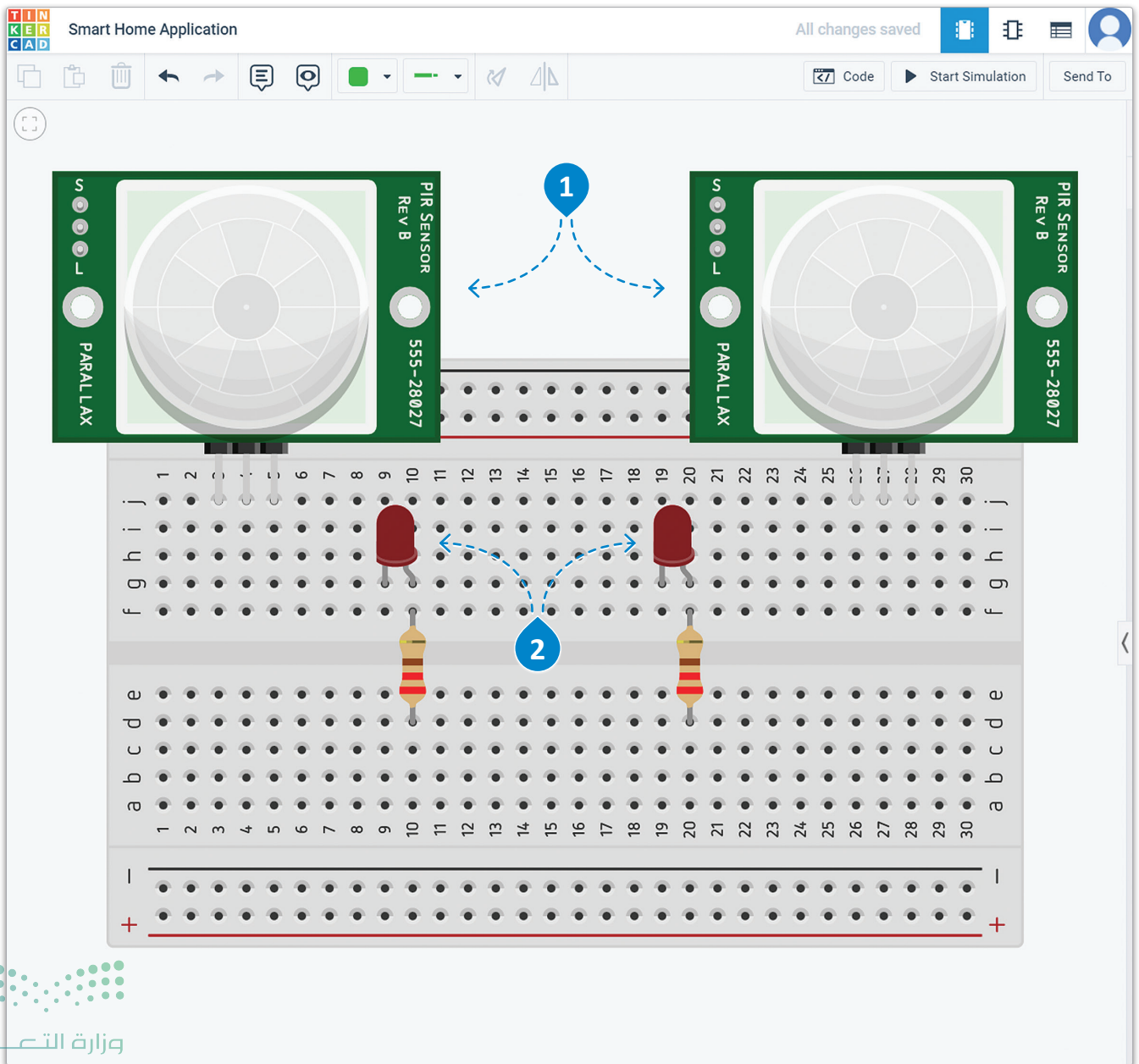
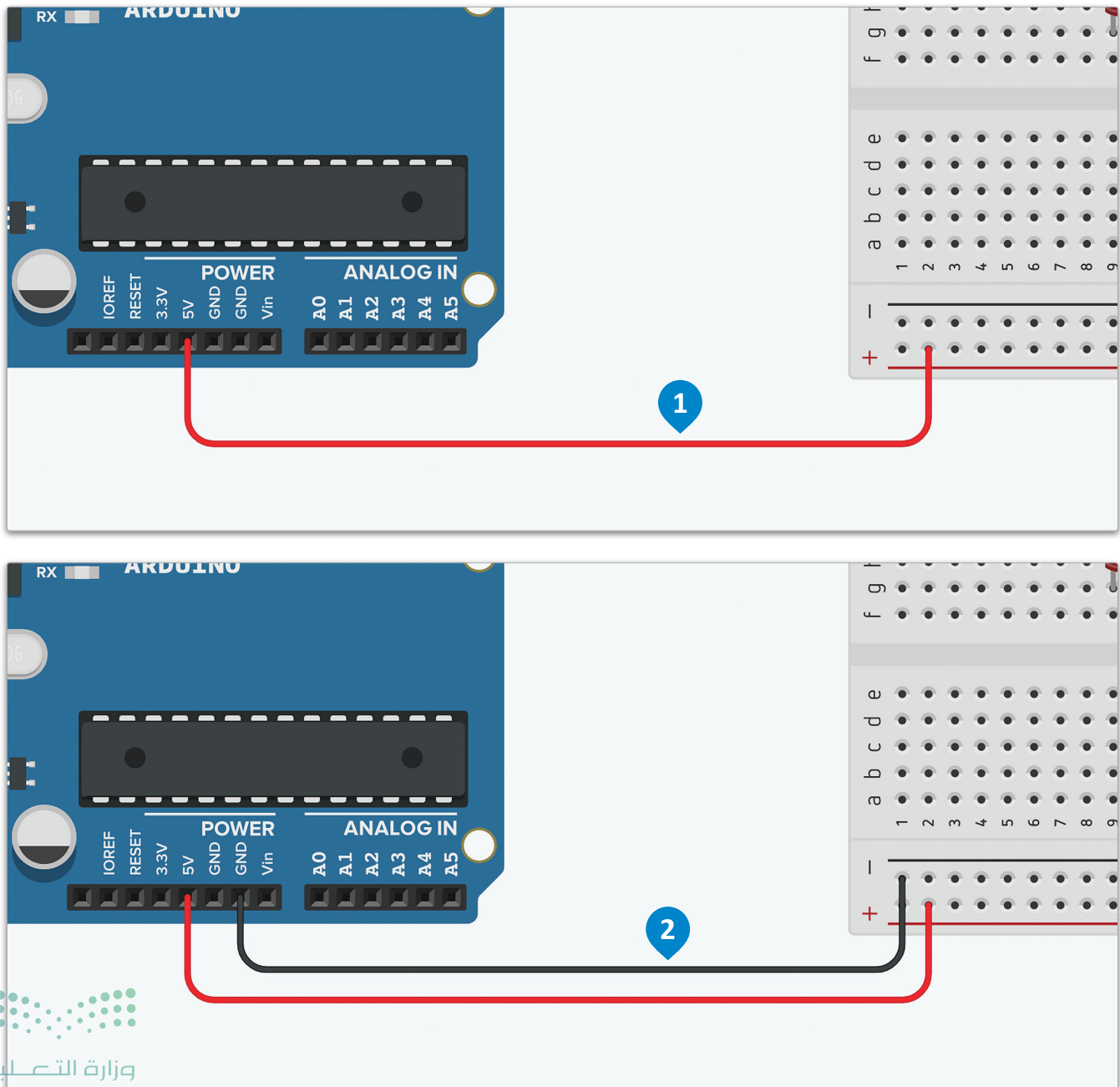


Figure 3.12: Add LEDs to the breadboard

Next, wire the Arduino UNO R3 to the breadboard by connecting the 5V pin to the positive column and the GND pin to the negative column.

To wire the Arduino UNO R3:

- > Connect the **Arduino UNO R3 5V pin** to the **positive column** of the **Breadboard** and change the wire color to **red**. **1**
- > Connect the **Arduino UNO R3 ground pin** to the **negative column** of the **Breadboard** and change the wire color to **black**. **2**



Then, connect the LED resistors to the Arduino digital pins 6 and 7.

To connect the LED resistors:

- > Connect one of the **resistors** to **Digital pin 6** of the **Arduino UNO R3** and change the wire color to **brown**. **1**
- > Connect the other **resistor** to Digital **pin 7** of the **Arduino UNO R3** and change the wire color to **brown**. **2**
- > Connect the two **LED cathodes** to the **negative column** of the **Breadboard**. **3**

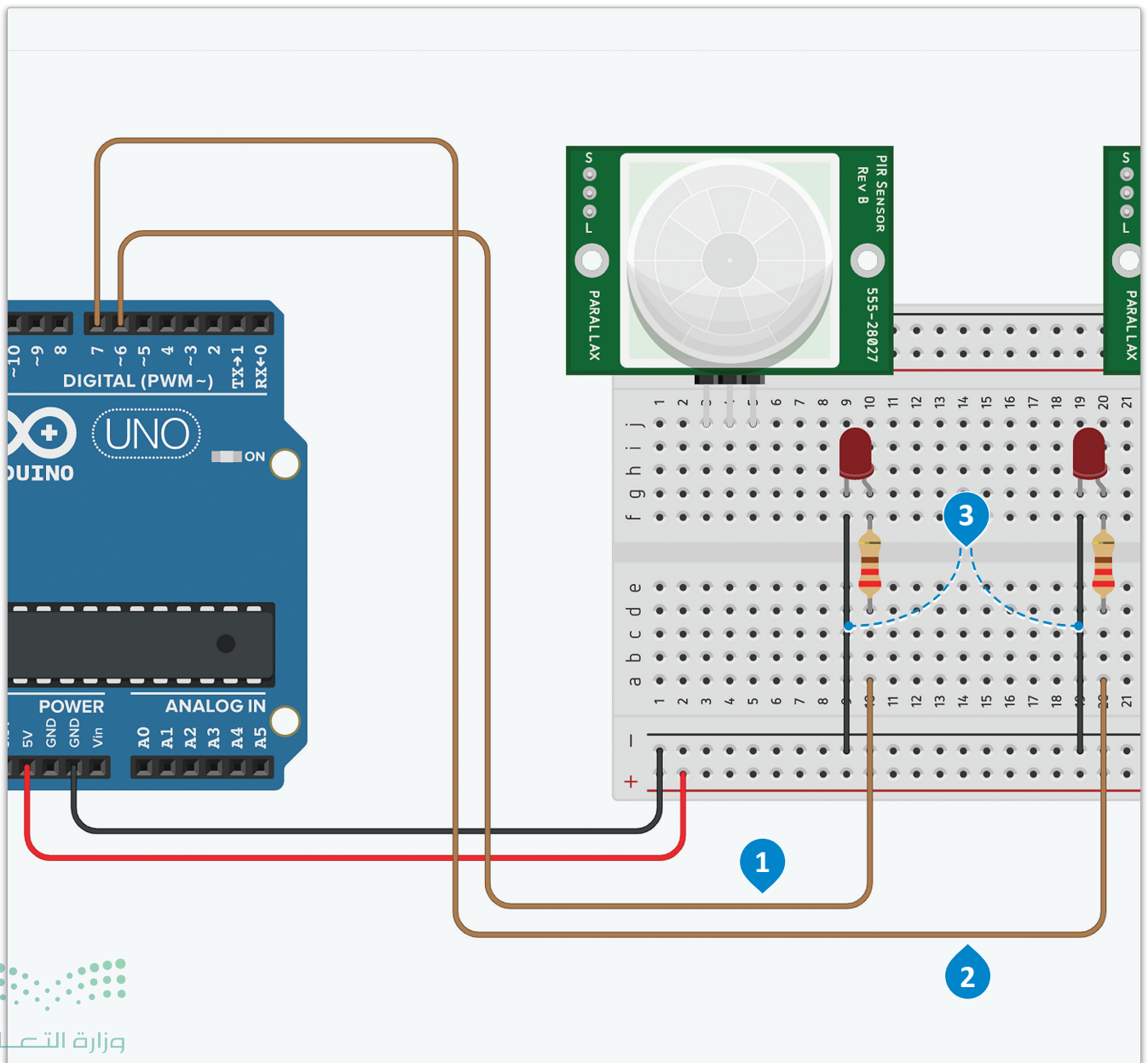


Figure 3.14: Wiring LED Resistor pairs

Finally, to complete the connections, you need to connect PIR "Room 1" and PIR "Room 2" to the Arduino Uno R3 and to the negative and positive columns of the breadboard.

To wire PIR sensors:

- > Connect the **PIR sensor Signal** pin to **Digital pin 8** and change the wire color to **green**. **1**
- > Connect the **PIR sensor Power** pin to the **positive column** of the **Breadboard Small** and change the wire color to **red**. **2**
- > Connect the **PIR sensor Ground** pin to the **negative column** of the **Breadboard Small** and change the wire color to **black**. **3**
- > Connect the **PIR sensor Signal** pin to **Digital pin 9** and change the wire color to **orange**. **4**
- > Connect the **PIR sensor Power** pin to the **positive column** of the **Breadboard Small** and change the wire color to **red**. **5**
- > Connect the **PIR sensor Ground** pin to the **negative column** of the **Breadboard Small** and change the wire color to **black**. **6**

Just like when you're coding, it's important to add comments to your work. In Tinkercad, you can add comments on the workplane. For example, you can add comments to indicate the two rooms by using the note tool.

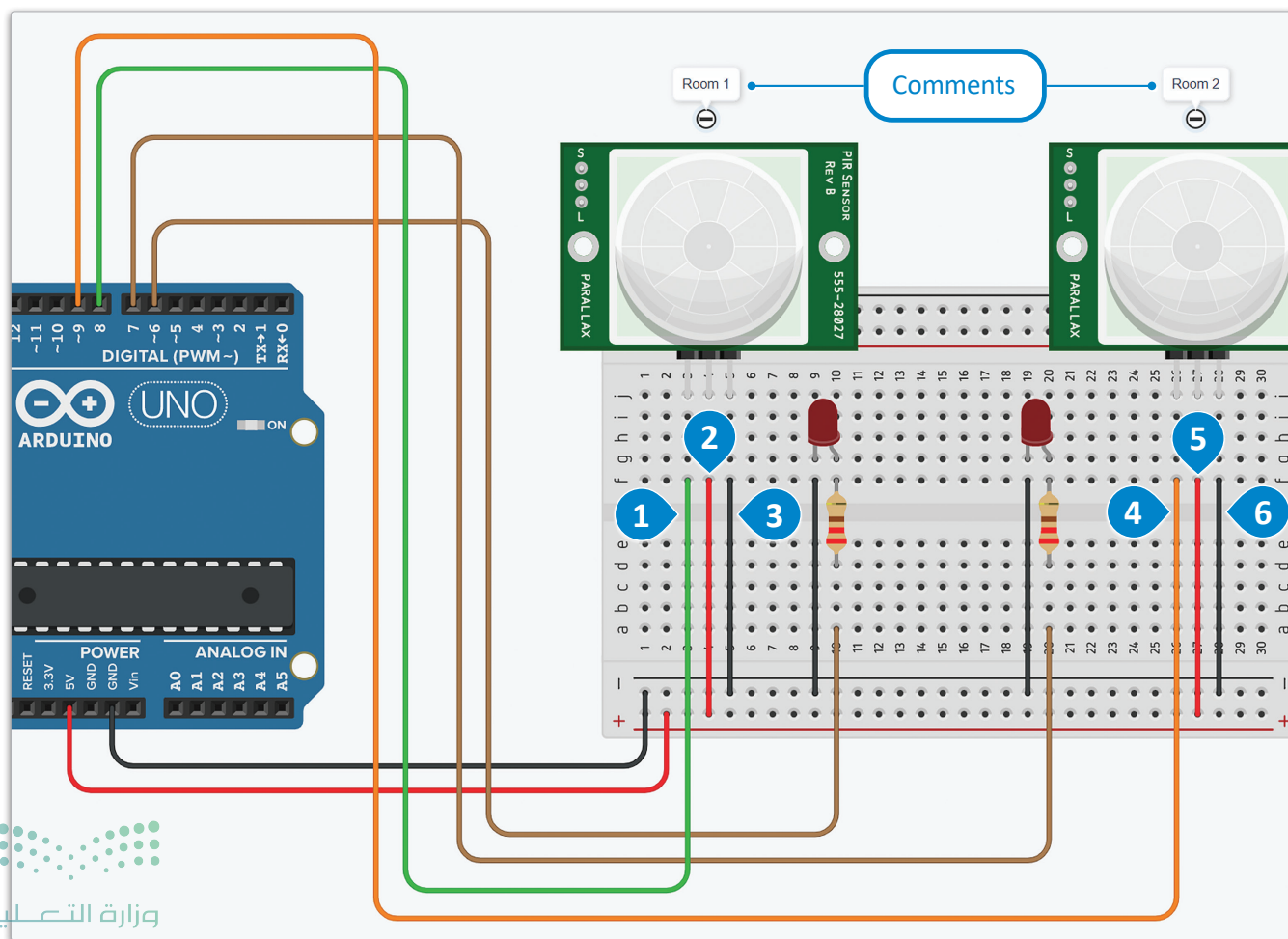
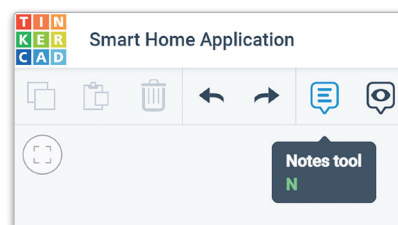


Figure 3.15: Wiring PIR sensors

Code Blocks

Now you have finished with the component preparation, let's explore the programming environment that you will use in this unit. Tinkercad provides Block-Based Coding technology to simplify microcontroller programming.

Toggle code editor

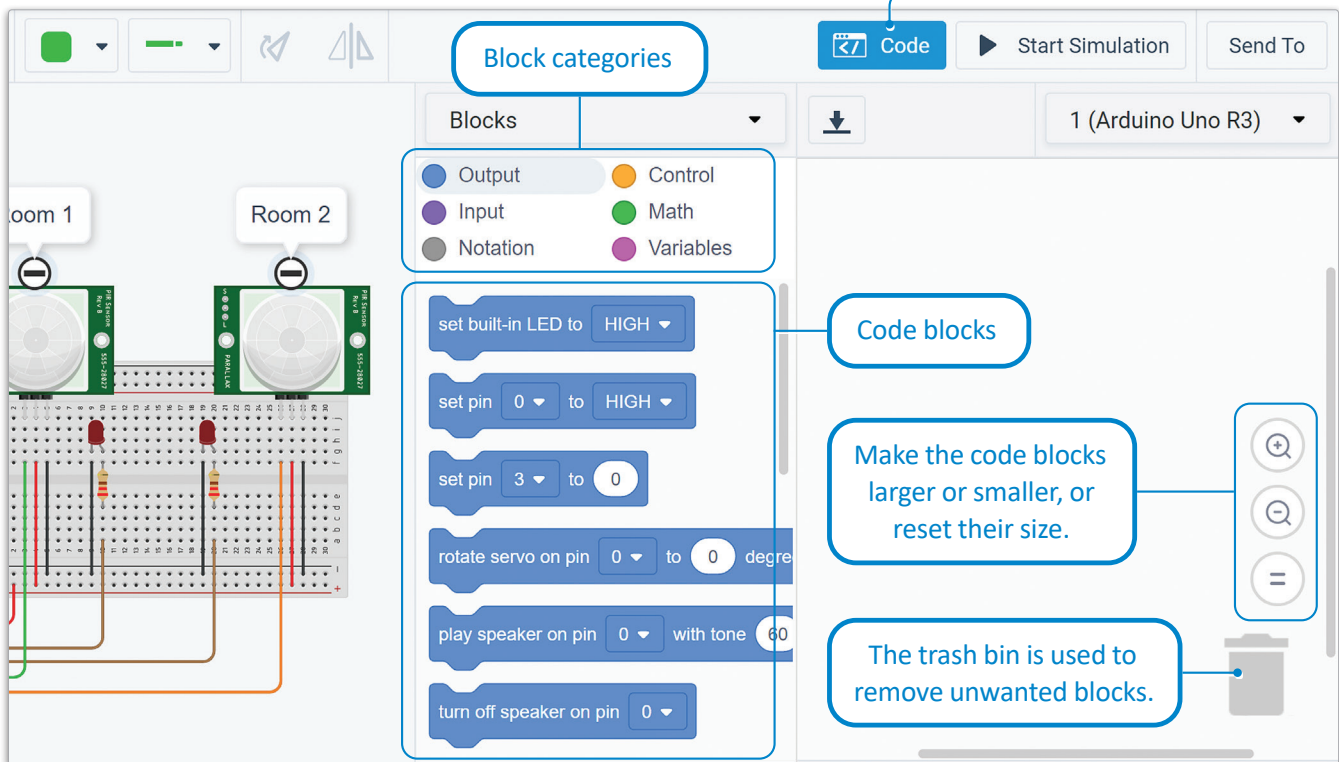


Figure 3.16: Code blocks

Blocks categories

Output

The **Output** blocks category allows you to define the digital and analog pins and send commands to the components of the microcontroller.

Input

The **Input** blocks category allows you to read data from the microcontroller.

Notation

The **Notation** blocks category allows you to use comments for your code.

Control

The **Control** blocks category allows you to add events, define loops to repeat actions and use statements to make decisions.

Math

The **Math** blocks category allows you to use mathematical symbols and operations.

Variables

The **Variables** blocks category allows you to create variables.

INFORMATION

Tinkercad's visual code blocks are useful for creating programs for the Arduino while avoiding common errors such as syntax errors, mistyping function names, forgetting a semicolon (;), and so on.

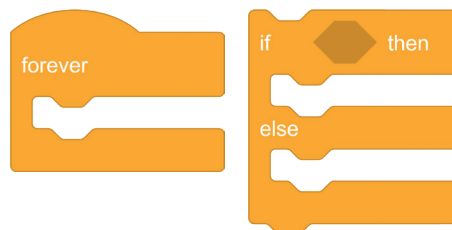
Let's take a look at some important blocks that you will use for this project:

The **forever** and the **if () then else** blocks can be found in the **Control** blocks category.

The **forever** block is essential to every program, because it is the block in which the rest of the commands are located. All other blocks inside it will run forever, in sequence, until the IoT device is switched off.

The **if () then else** block checks whether the criteria of the condition are met. If so, the microcontroller executes the blocks between **if () then** and **else**. Otherwise, the blocks below **else** are executed.

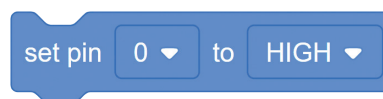
Control



The **set pin () to ()** block can be found in the **Output** blocks category.

The **set pin () to ()** block can set a digital or analog pin state to High or Low

Output



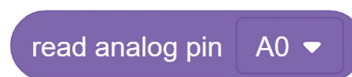
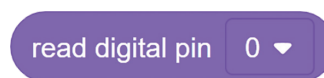
Analog pins can have a range of voltage values from 0V to 3.3V or 5V. This means that when interpreted by the program, these voltages correspond to a large variety of values.

The **read digital pin ()** and the **read analog pin ()** blocks can be found in the **Input** blocks category.

The **read digital pin ()** block is a boolean block that reads the state of the digital pin on the device ("High" or "Low").

The **read analog pin ()** block can read a voltage level that ranges from the supply voltage of your board, 3.3V or 5V, down to 0V (GND).

Input



Digital pins have two states; High, when voltage passes through (3.3V or 5V), and Low, when it does not (0V). Those two states are interpreted by the program as 1 (True) and 0 (False) respectively.



Creating the Code

Your program will execute forever and implement two if/else blocks, one for Room 1 and the other for Room 2. The PIR sensors will check for movement inside the rooms. If a PIR sensor detects a "person" it will turn its LED on and when the PIR sensor stops sensing the "person" then it will turn its LED off. This circuit simulates a Smart Home Application that turns the lights on automatically when someone enters a room.

The screenshot displays the Arduino IDE interface for a Smart Home Application. The 'Blocks' panel on the left lists various components: Output (blue), Input (purple), Notation (grey), Control (orange), Math (green), and Variables (pink). The 'Code' panel on the right shows the implementation of the program. The code starts with a 'forever' loop (orange block) that runs continuously. Inside the loop, there are two if/else blocks. The first if/else block checks for movement in Room 1 using digital pin 8. If the pin is HIGH, it sets pin 6 to HIGH (turns the LED on). If the pin is LOW, it sets pin 6 to LOW (turns the LED off). The second if/else block checks for movement in Room 2 using digital pin 9. If the pin is HIGH, it sets pin 7 to HIGH (turns the LED on). If the pin is LOW, it sets pin 7 to LOW (turns the LED off). Annotations explain the 'forever' loop and the room checks. The interface also includes a 'Start Simulation' button and a 'Send To' button. The bottom left corner shows the logo of the Ministry of Education, 2026 - 1448.

Blocks inside the forever block will run continuously in sequence until you stop the simulation.

Check for any movements in Room 1.

Check for any movements in Room 2.

Figure 3.17: Code implementation

Start the simulation by pressing the **Start Simulation** button to test your program.

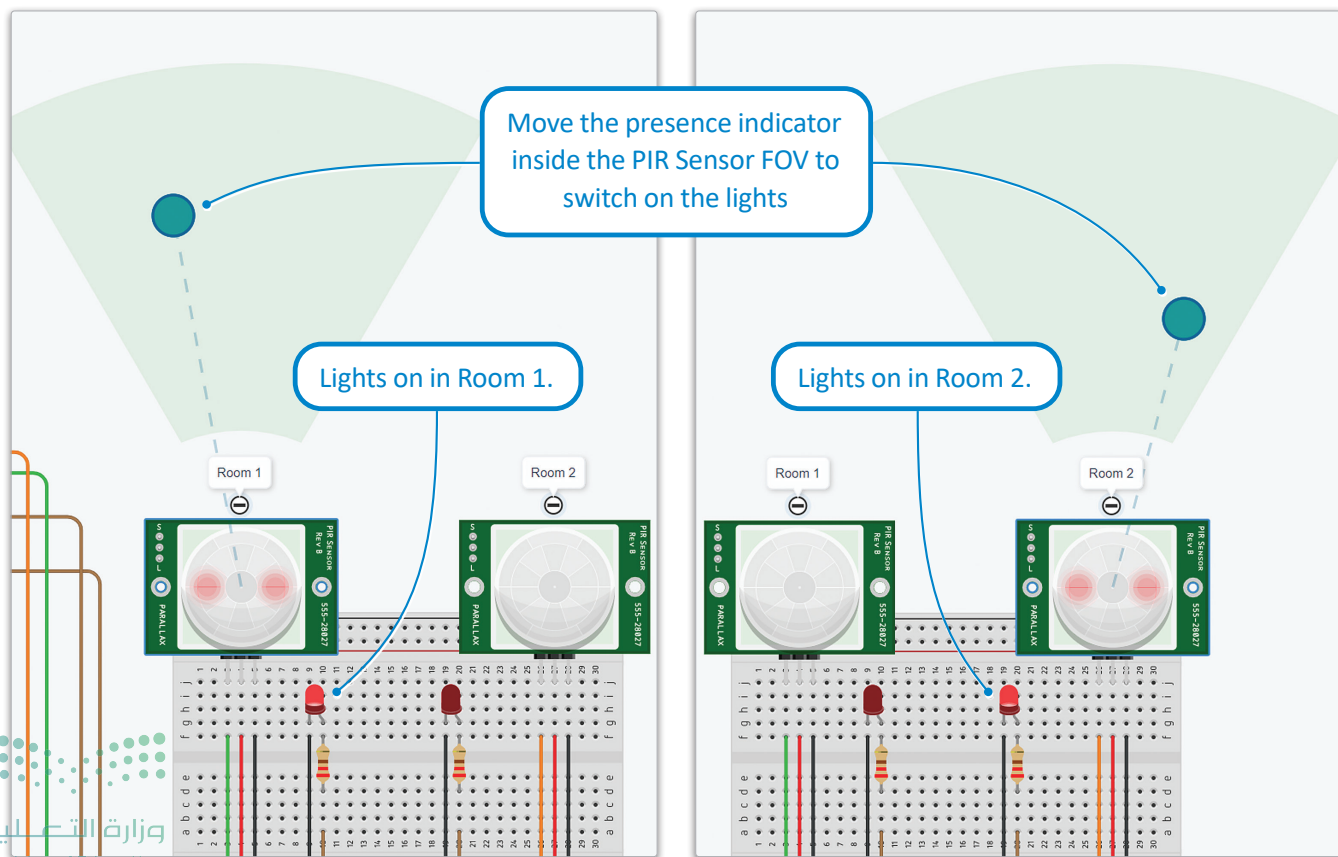
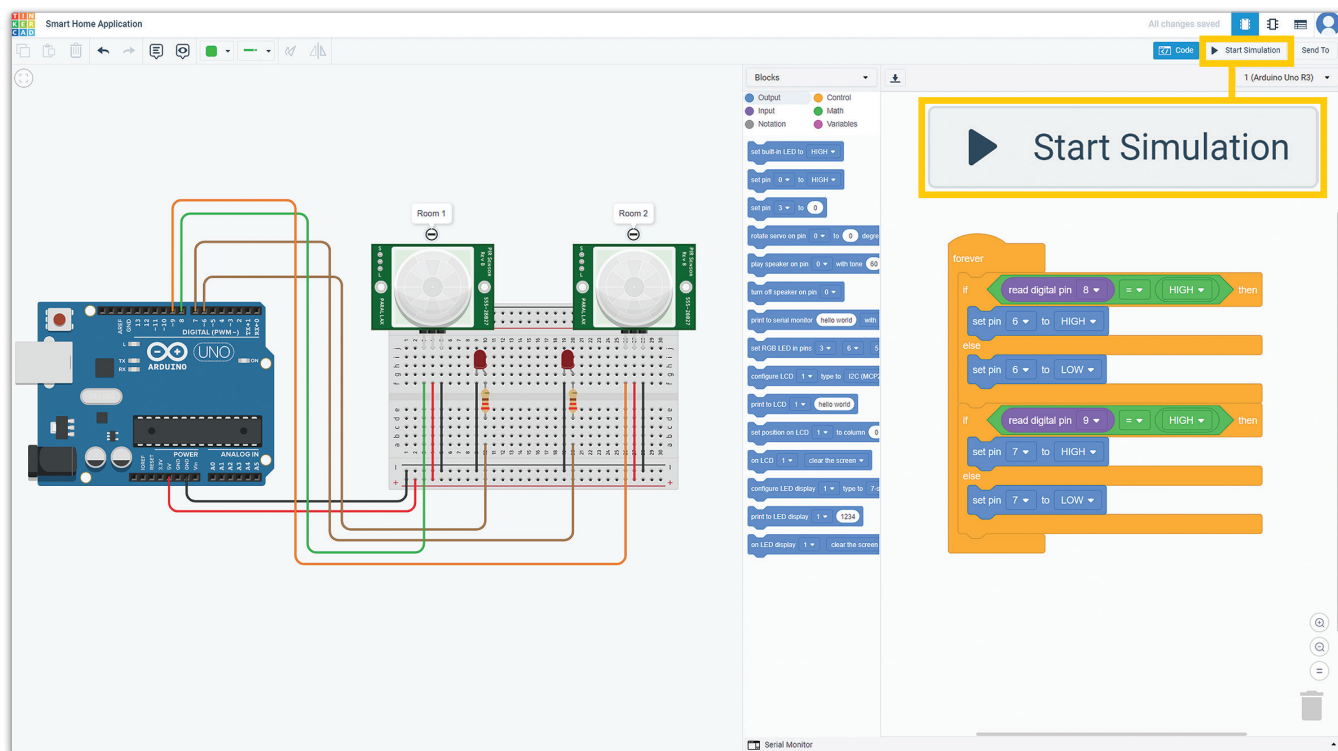
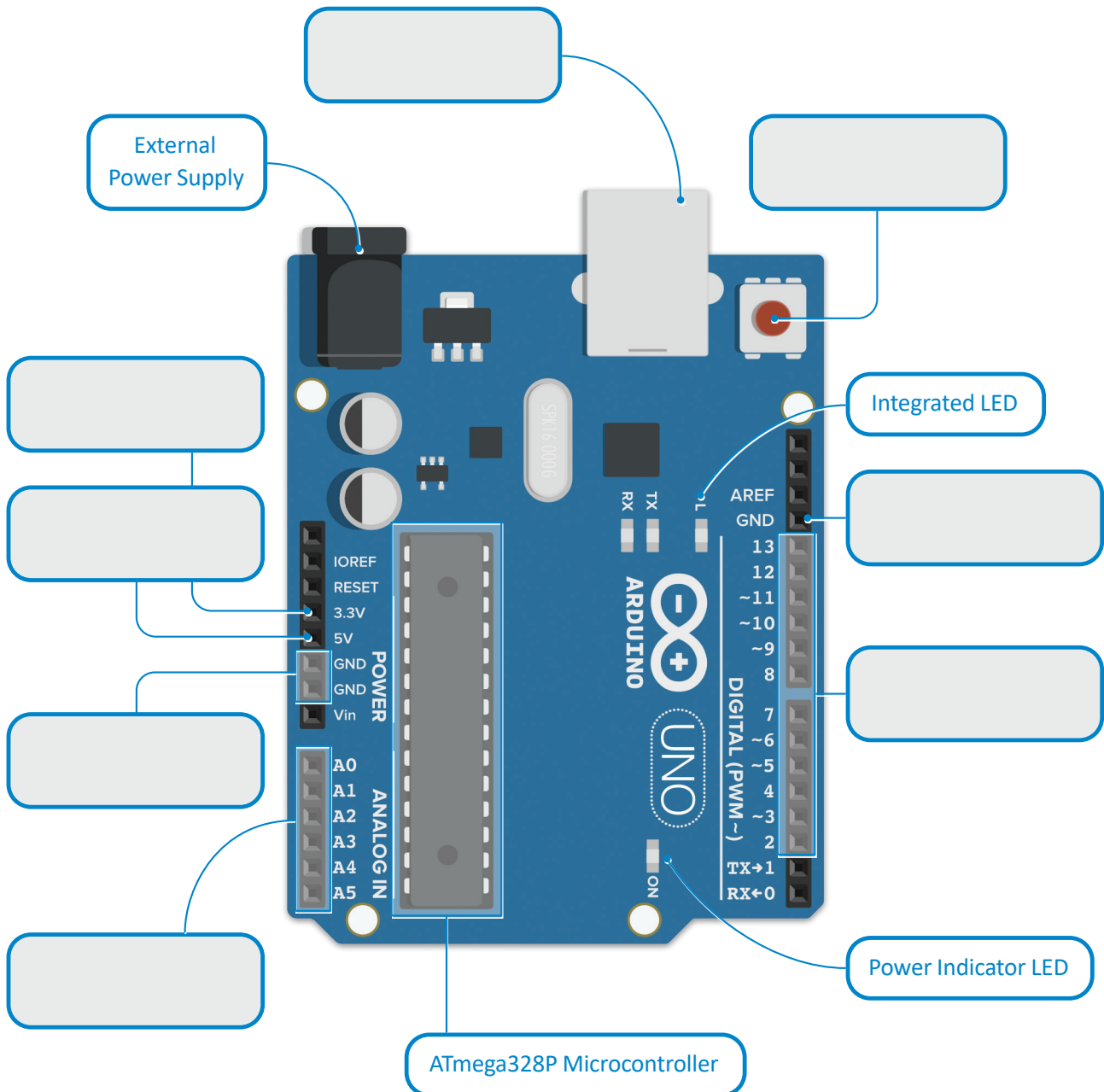


Figure 3.18: Program simulation

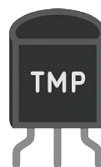
Exercises

- 1 Fill the empty boxes with the missing component names.



2 a. Match the items in the first row with those in the second.

Simulator
component



Name

Temperature

Gas

Soil Moisture

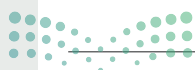
b. Write in which cases we can use each sensor.

1. Temperature

2. Gas

3. Soil Moisture

3 What is the difference between digital and analog pins?



- 4 Search the Internet for the main differences between the Arduino Uno and the Arduino Nano series. For what types of applications is each series used? Present your findings below.

- 5 What is the advantage of programming an Arduino microcontroller with the Codeblocks language, instead of text-based code in Tinkercad Circuits? Can you mention a disadvantage?

- 6 Extend the Smart Home System to monitor 3 rooms.

- 7 Change the Smart Home System so that it has 2 LEDs (1 red and 1 green) and 1 PIR Sensor. When the PIR sensor detects motion, only the green LED will be turned on, and when it does not detect motion, only the red LED will be turned on.



Lesson 2

Build a Plant Watering System

Build a Plant Watering System

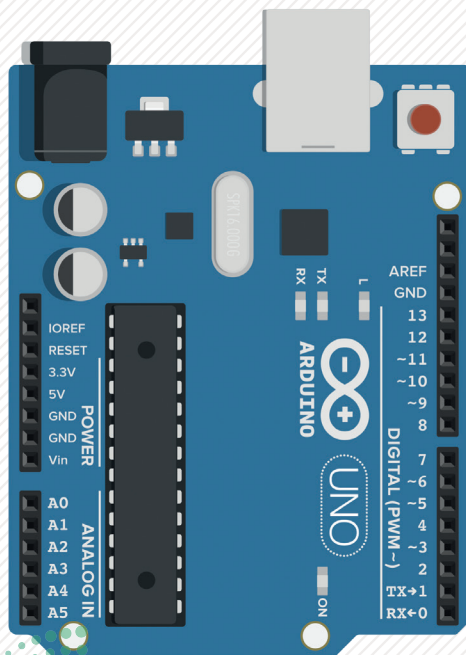
As the population is rising, more crops are needed to provide food and other essential products, but the large quantity of plants to be taken care of cannot be handled manually. By using automated systems for irrigation, efficiency is increased, and agriculture benefits greatly.

In this lesson, you will use the Arduino UNO R3 to simulate a Plant Watering System. The system will use a motor to drive a watering system when the sensors detect that soil moisture has dropped and the temperature is high.

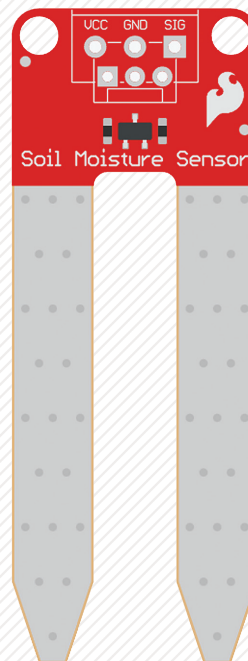
For this project you will use the following components:

- 1 DC Motor
- 1 Temperature Sensor [TMP36]
- 1 Soil Moisture Sensor
- 1 Arduino UNO R3

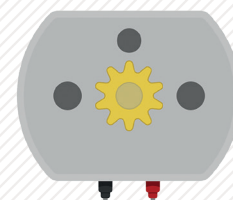
Components that you will use in this project



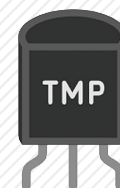
Arduino Uno R3



Soil Moisture Sensor



DC Motor



Temperature Sensor

Let's start by adding the components that you will need for this project to the workplane. First, find the Arduino UNO R3, add it to the workplane and rotate it 90 degrees.

To add the Arduino Microcontroller:

- > Drag and drop the **Arduino UNO R3** from the **components library** into the **workplane**. **1**
- > Click on the **Rotate** button three times. **2**

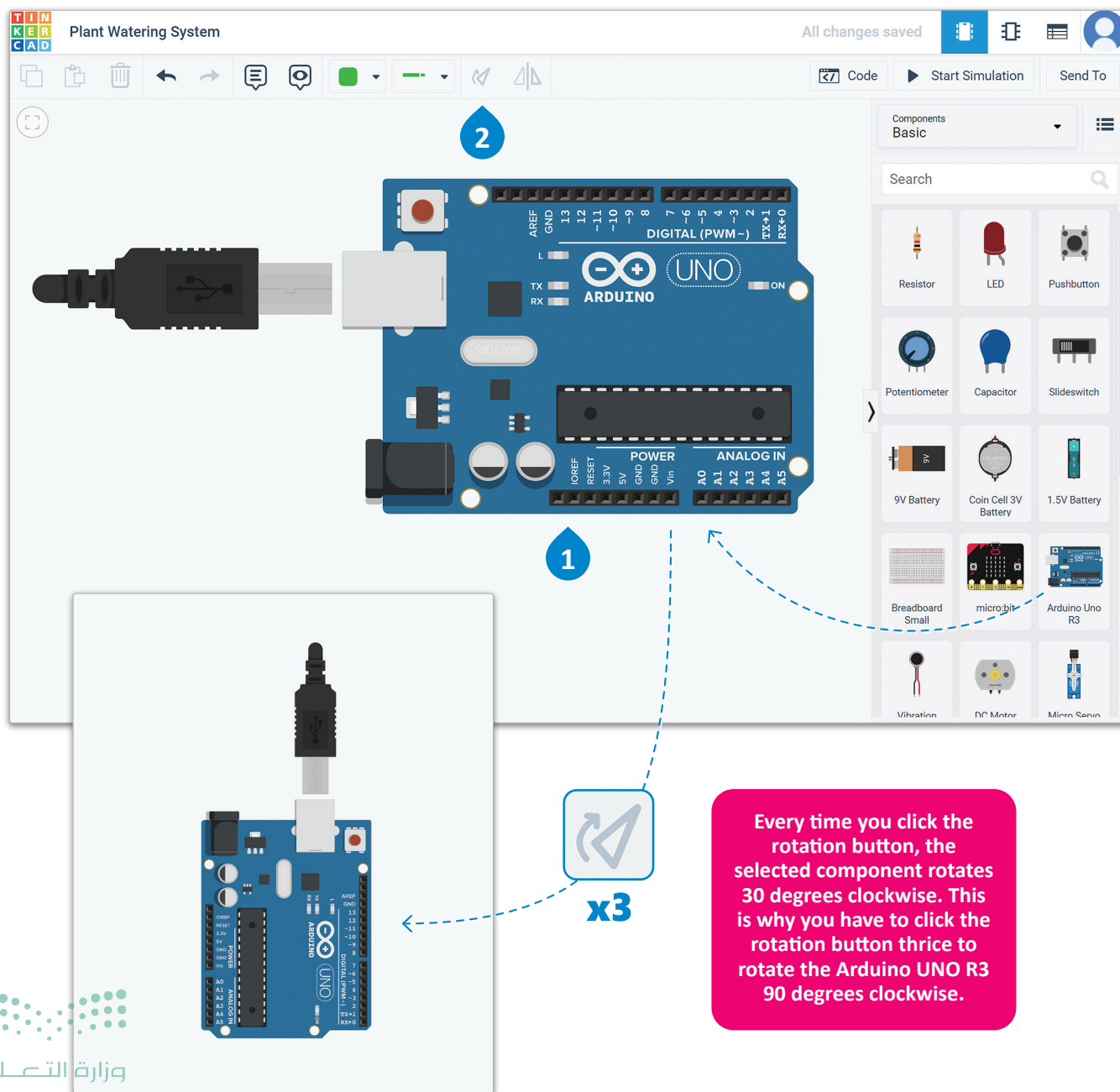
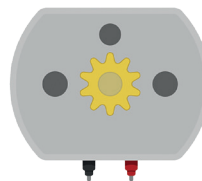


Figure 3.20: Add the Arduino Microcontroller

Next, find the rest of the components, the DC Motor, the Temperature Sensor and the Soil Moisture Sensor, and add them to the workspace.

To add components to the workspace:

- > Find the **DC Motor** from the **components library** and drag and drop it into the **workspace**. **1**
- > Find the **Temperature Sensor** from the **components library** and drag and drop it into the **workspace**. **2**
- > Find the **Moisture Sensor** from the **components library** and drag and drop it into the **workspace**. **3**



In this project, the DC Motor is used as a mean to turn on a water pump, which could provide water to the plants.

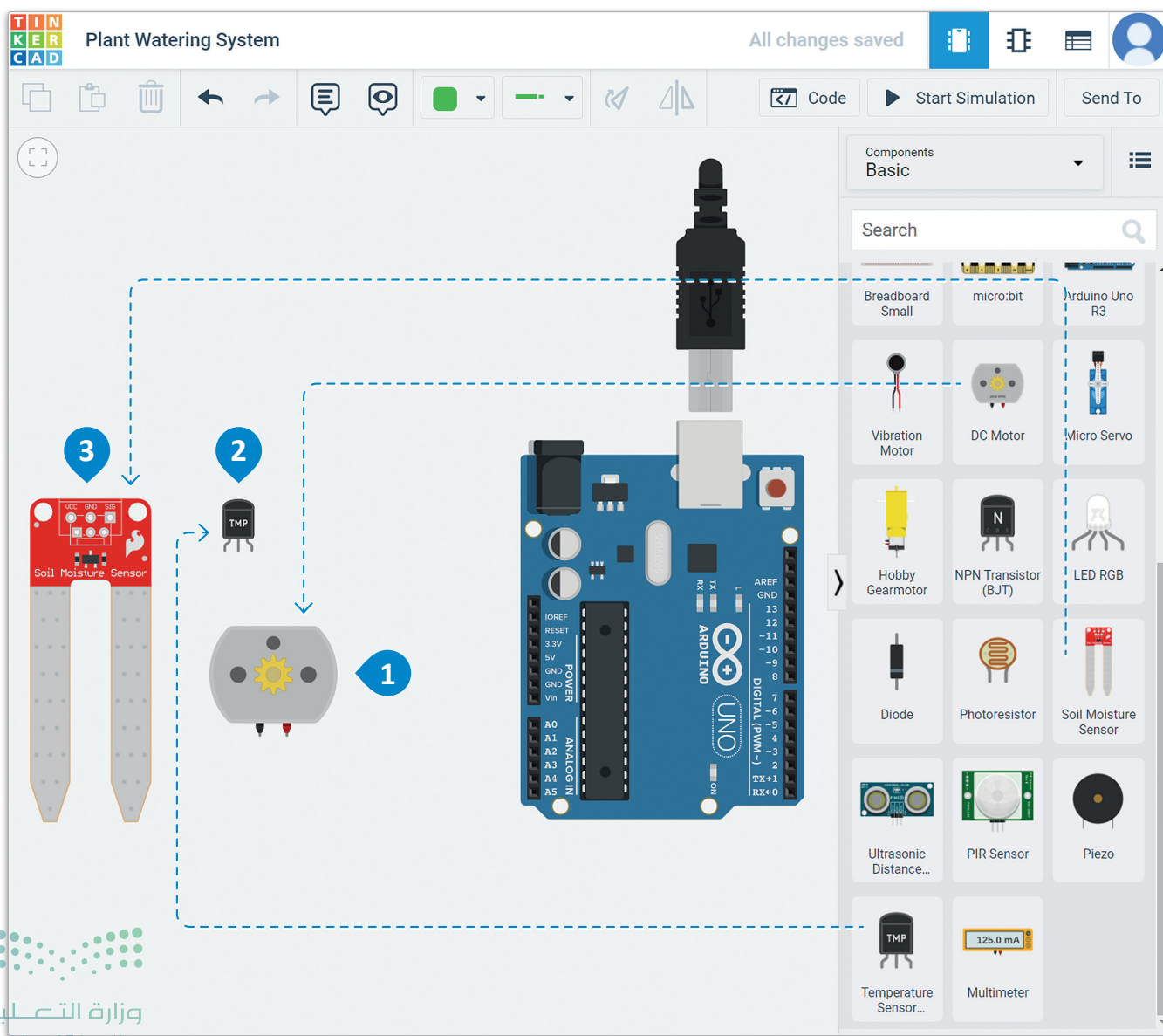


Figure 3.21: Add components to the workspace

Now, wire the DC Motor to the Arduino UNO R3.

To wire the DC Motor:

- > Connect **Terminal 1** of the **DC Motor** to an **Arduino GND pin** and change the wire color to **black**. **1**
- > Connect **Terminal 2** of the **DC Motor** to the **Arduino Analog pin A5** and change the wire color to **red**. **2**

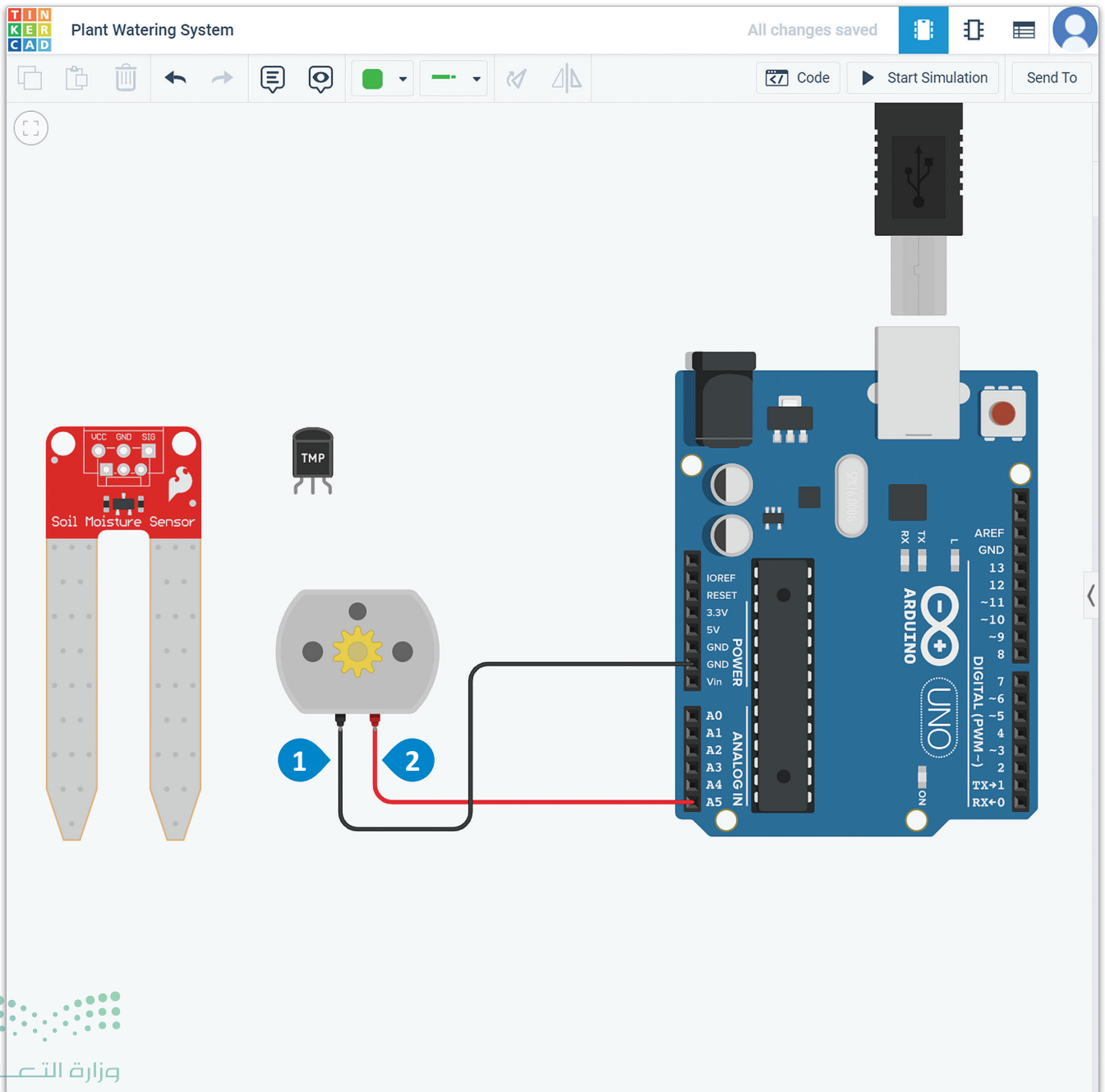


Figure 3.22: Wire the DC Motor

Next, wire the Temperature Sensor to the Arduino UNO R3.

To wire the Temperature Sensor:

- > Connect the **Power** pin of the **Temperature Sensor** to the **Arduino 5V pin** and change the wire color to **red**. **1**
- > Connect the **Vout** pin of the **Temperature Sensor** to the **Arduino Analog pin A3** and change the wire color to **yellow**. **2**
- > Connect the **GND** pin of the **Temperature Sensor** to an **Arduino GND pin** and change the wire color to **black**. **3**

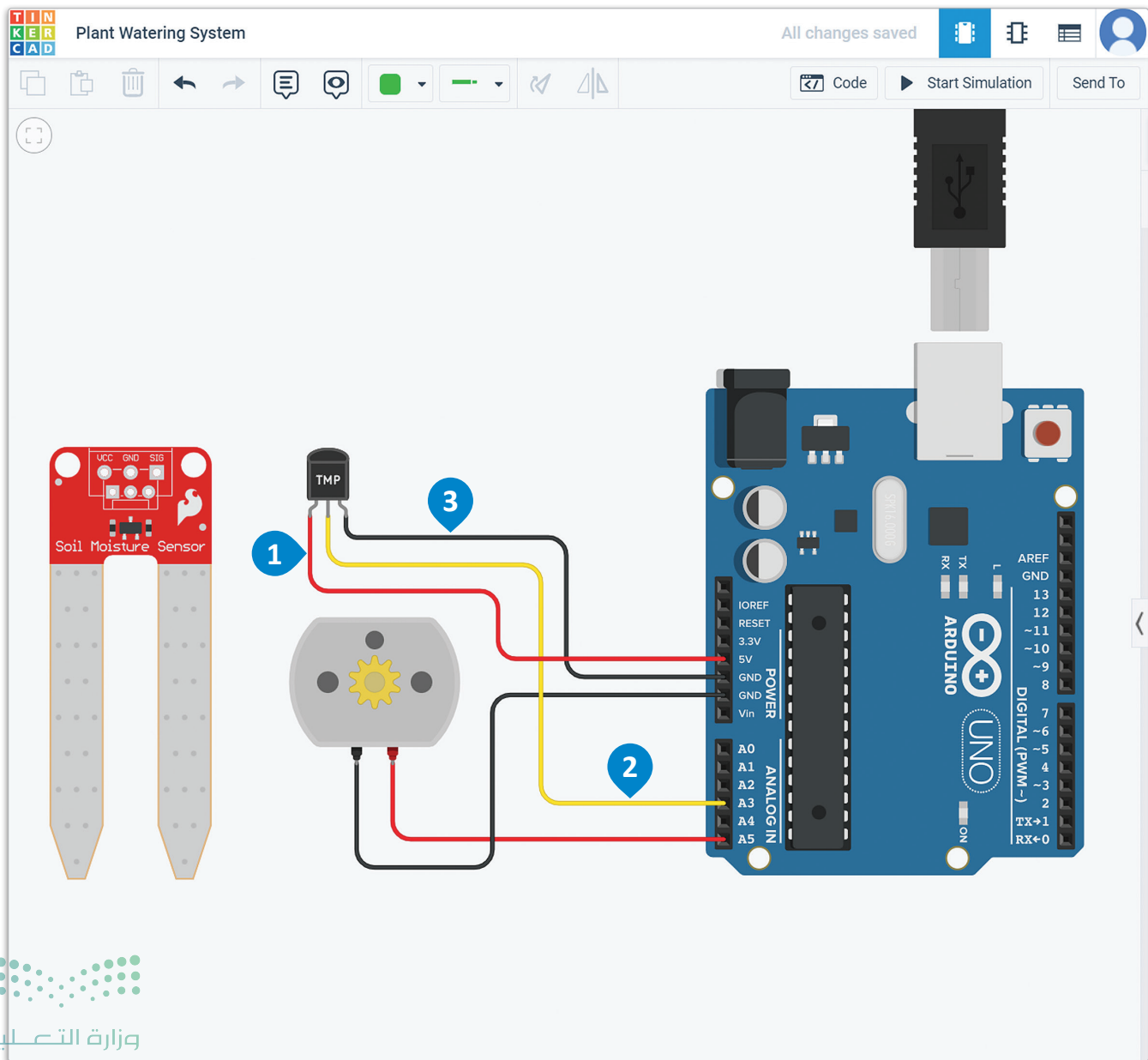
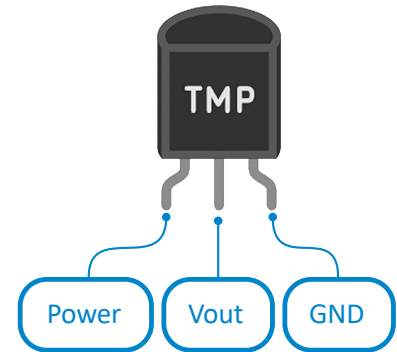


Figure 3.23: Wire the Temperature Sensor



Continue by wiring the Soil Moisture Sensor to the Arduino UNO R3.

To wire the Soil Moisture Sensor:

- > Connect the **Power** connector of the **Soil Moisture Sensor** to the **Arduino 5V pin** and change the wire color to **red**. **1**
- > Connect the **Ground** connector of the **Soil Moisture Sensor** to an **Arduino GND pin** and change the wire color to **black**. **2**
- > Connect the **Signal** connector of the **Soil Moisture Sensor** to the **Arduino Analog pin A0** and change the wire color to **green**. **3**

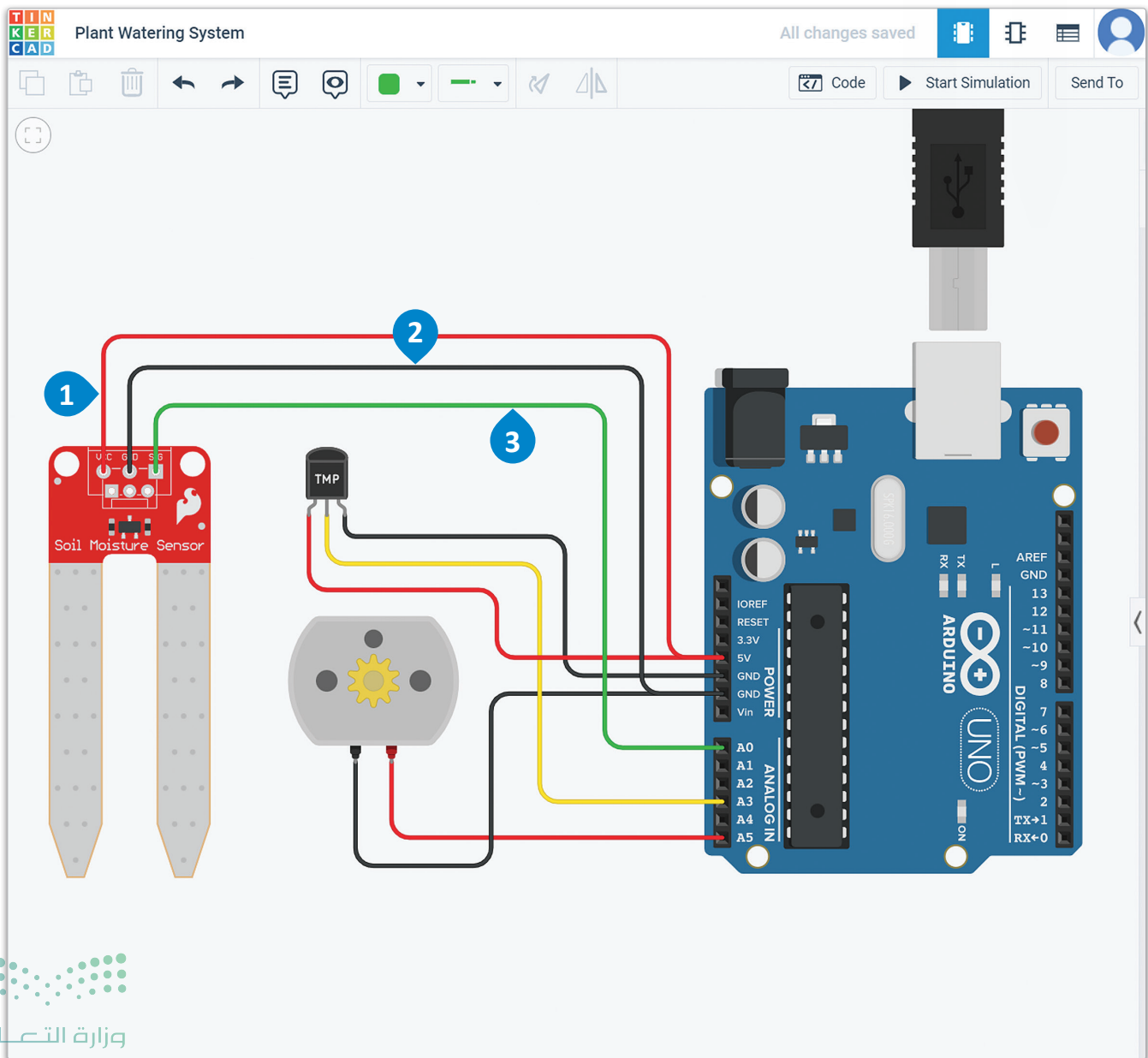
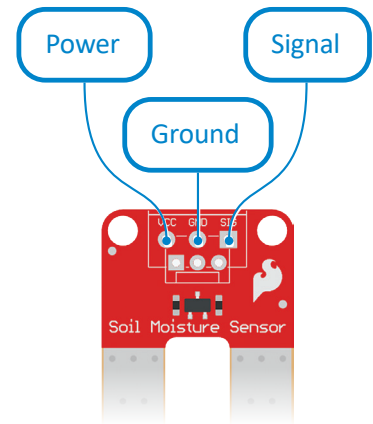


Figure 3.24: Wire the Soil Moisture Sensor

The Serial Monitor

The Serial Monitor is a tool we use to observe the values that we receive from the sensors. It makes the process of identifying problems in our circuit and/or code much easier by allowing the user to read output from the program and sensors. It can also be used for aesthetic reasons, to print a special message to the user, or to provide useful information and instructions. The Serial Monitor can be accessed from the bottom of the Code panel.

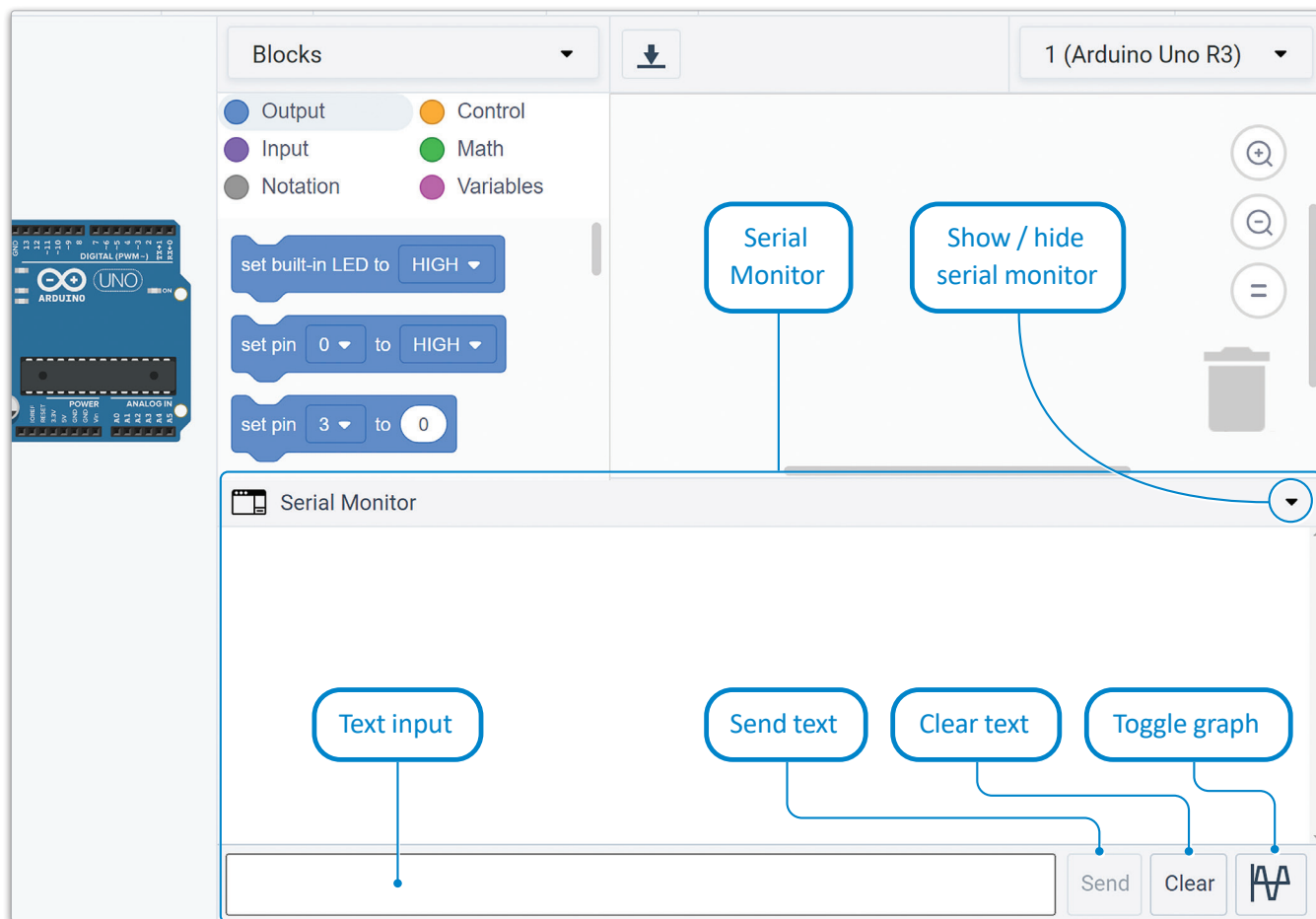
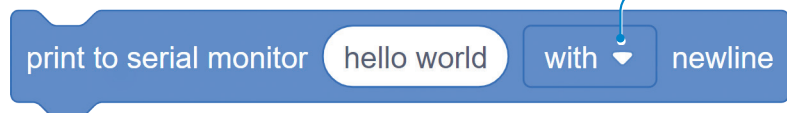


Figure 3.25: Serial Monitor



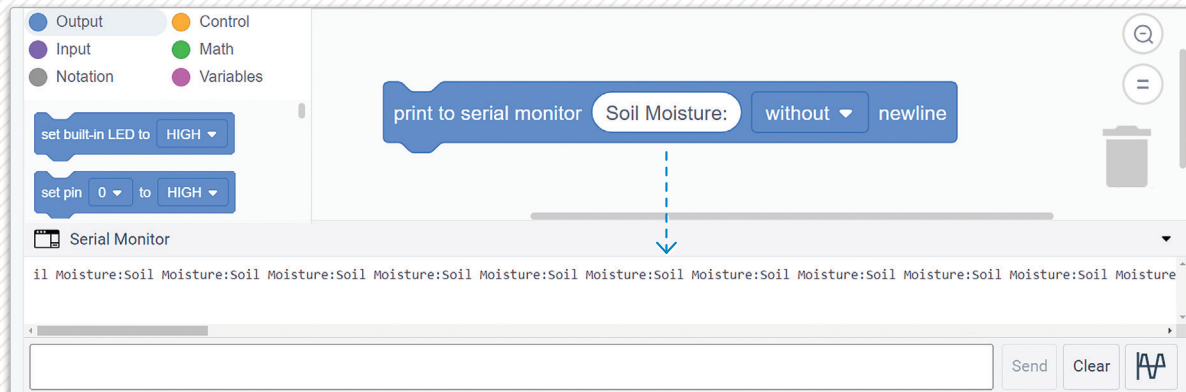
Choose "with" to start a newline after each block's message or "without" to continue on the same line.

The **print to serial monitor () () newline** block can be use to print messages in the **Serial Monitor**. The messages can be either character strings or numeric values.

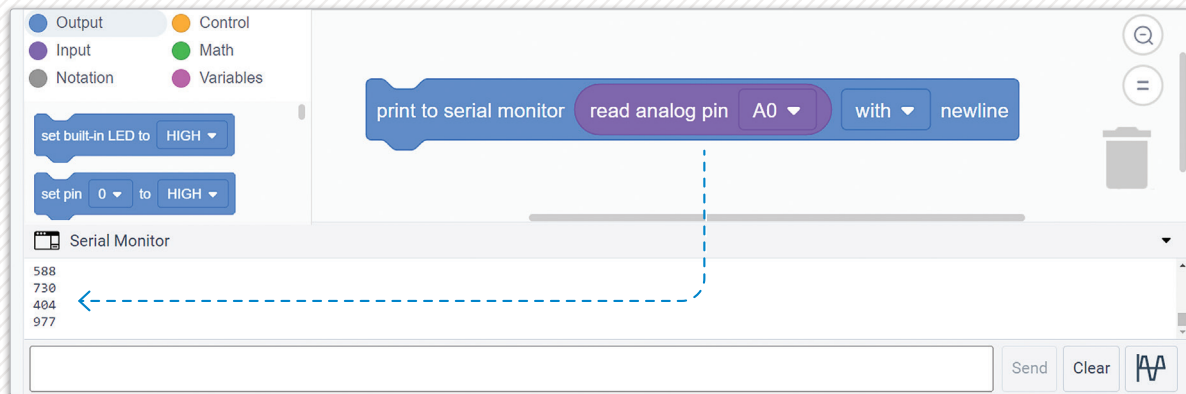


Example

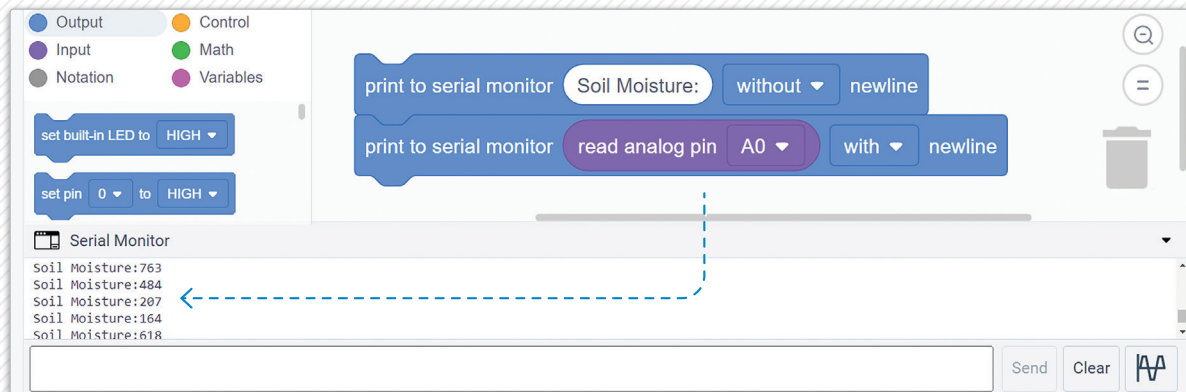
As an example, if you want to print the string "Soil Moisture: " without a newline, use:



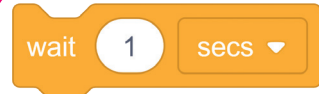
And if you want to print the value of analog pin A0 with a newline use:



If you combine those two you will get more complete serial monitor output:



wait () block delays the execution time of the program and its print outputs, thus providing a more pleasant output to the user. The block can delay the program for a defined number of seconds or milliseconds.



The Temperature Sensor sends analog signals but it uses a special block so we can specify which temperature unit (Celsius or Fahrenheit) we want. To read the temperature in Celsius from analog pin A3 use:

```

print to serial monitor Temperature: without newline
print to serial monitor read temperature sensor on pin A3 in units °C with newline
  
```

Code Implementation

First, the program prints the values of the soil moisture and the temperature to the serial monitor. Then, it checks if both conditions are evaluated true, and if so it turns the motor on, else it turns it off. Finally, the program waits for a second to avoid flooding the serial monitor with output.

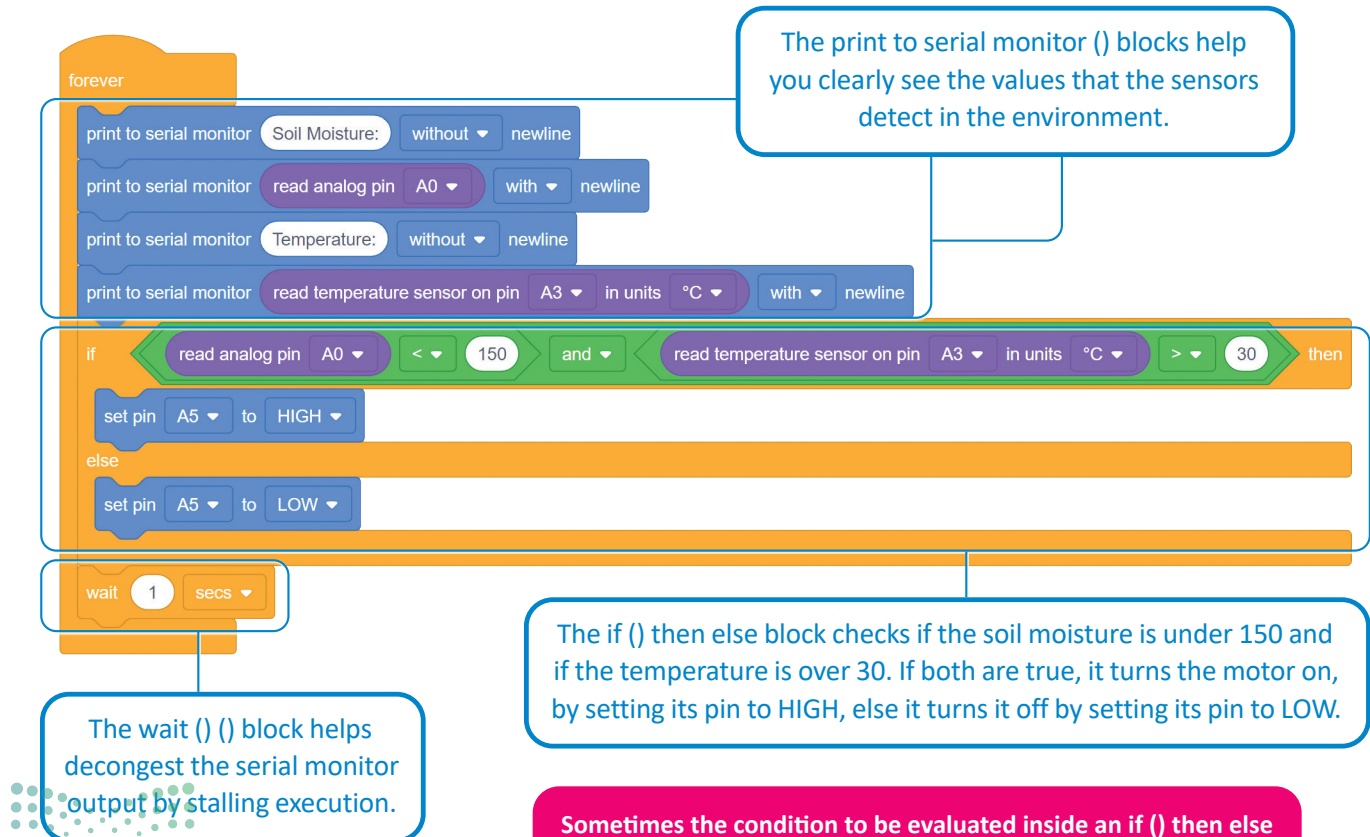
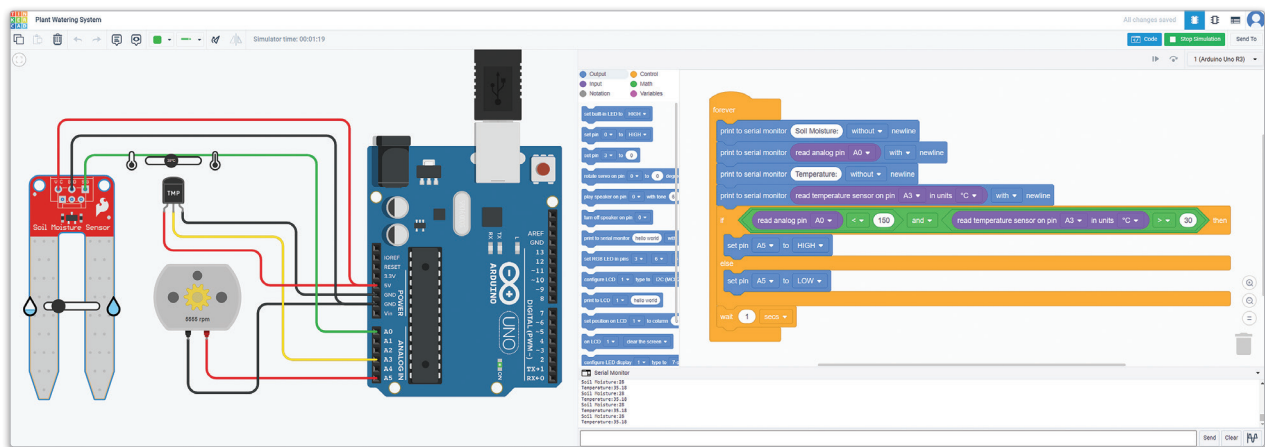


Figure 3.27: Code implementation

Table 3.2: System input values and output states

if read analog pin A0 > 150 and read temperature sensor on pin A3 in units °C > 30 then		
Soil moisture (pin A0)	Temperature (pin A3)	Output to motor (pin A5)
≥ 150	≤ 30	LOW
≥ 150	> 30	LOW
< 150	≤ 30	LOW
< 150	> 30	HIGH

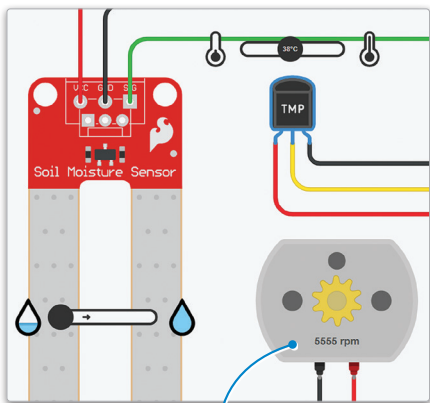
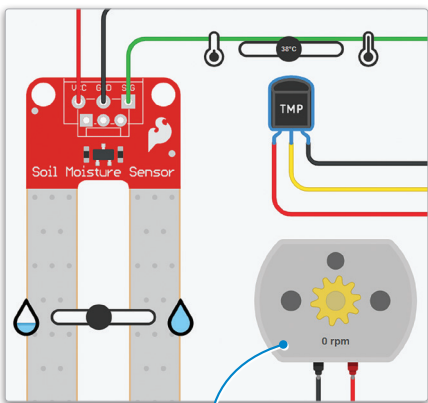
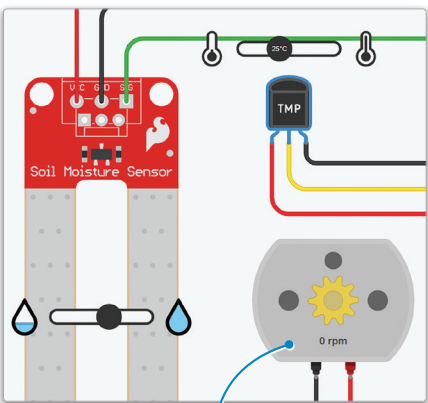
Now that you have finished with the coding blocks, you are ready to execute the program. Start the simulation by pressing the **Start Simulation** button. You can experiment and see the different states of the circuit by selecting a sensor's slider and adjusting its value:



Soil Moisture 600
and Temperature 25°C

Soil Moisture 300
and Temperature 38 °C

Soil Moisture 0
and Temperature 38 °C



DC Motor 0 rpm

DC Motor 0 rpm

DC Motor 5555 rpm

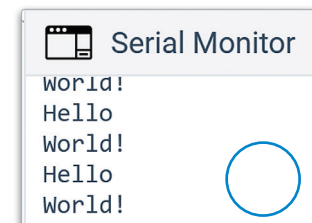
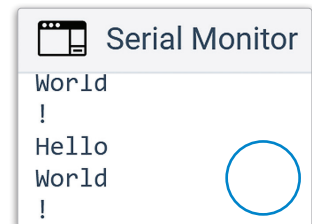
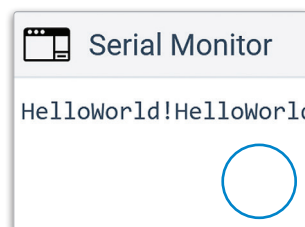
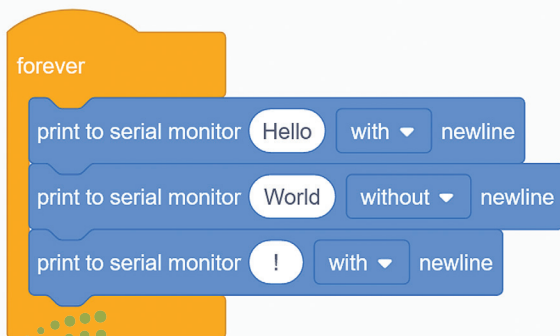
Figure 3.28: Motor speed at different Moisture and Temperature values

Exercises

- 1 If we wanted to set up the Smart Watering System in a more arid area where the water evaporates faster and the atmosphere is drier, what changes should we make? Describe a possible solution and present your ideas below.

- 2 In this lesson's project, the sensors and actuator did not require a breadboard to connect to the Arduino, in contrast with other projects. Describe the reasons why below.

- 3 Check with a tick ✓ the output of the following blocks.



- 4 Evaluate the following conditions of the code blocks as either True or False, by taking into consideration the values of the analog pins given:

Read the sentences and tick ✓ True or False.	True	False
	☐	☐
1. Soil Moisture Sensor Input A0: 180 - Temperature Sensor Input A3: 32	☐	☐
	☐	☐
2. A0 (Soil Moisture): 167 - A3 (Temperature): 43	☐	☐
	☐	☐
3. A0 (Soil Moisture): 225 - A3 (Temperature): 35	☐	☐
	☐	☐
4. A0 (Soil Moisture): 58 - A3 (Temperature): 41	☐	☐
	☐	☐
5. A0 (Soil Moisture): 150 - A3 (Temperature): 35	☐	☐

- 5 Extend the Smart Watering System to inform the user when the plant is being watered by printing the message "Watering Plant Now!" only when moisture is under 150 and temperature is over 30.

Don't forget to add a newline in the message so that the monitor is intelligible.

- 6 Extend the Smart Watering System so the DC Motor stays on for longer if the soil moisture levels are extremely low (under 50). Use wait blocks to make the DC Motor operate for longer.

Lesson 3

Build a Gas Leak System

Link to digital lesson



www.ien.edu.sa

Build a Gas Leak Alarm System

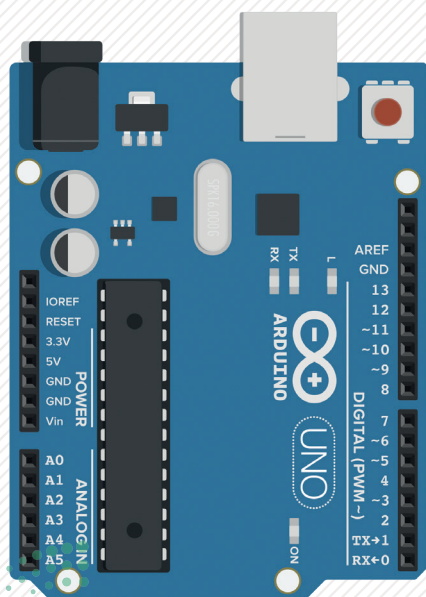
Many facilities, such as houses and working environments, are susceptible to gas leaks from appliances or have flammable materials that impose fire hazards. One method to prevent accidents is to equip those areas with smart gas detection systems.

This is a circuit that uses the Arduino UNO R3 to simulate a Gas Leak Alarm System. When the Arduino receives a signal from the gas sensor indicating high smoke concentration, it will send signals to the LEDs to flash alternately and to the Piezo buzzer to make a short beeping sound.

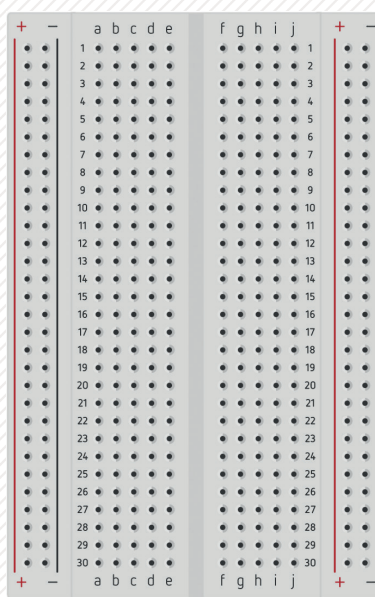
In this project you will be using the following components:

- 1 Piezo
- 1 Gas Sensor
- 2 LEDs
- 3 Resistors
- 1 Arduino Uno R3
- 1 Breadboard Small

Components that you will use in this project



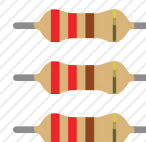
Arduino Uno R3



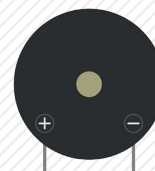
Breadboard Small



LEDs



Resistors



Piezo



Gas Sensor

Figure 3.29: Project components

We will start by adding the components that you will need for this project to the workplane. First, find the Arduino UNO R3 and the Breadboard.

To add the Arduino UNO R3 and the Breadboard:

- > Drag and drop the **Arduino UNO R3** from the **components library** to the **workplane**. **1**
- > Drag and drop the **Breadboard Small** from the **components library** to the **workplane**. **2**

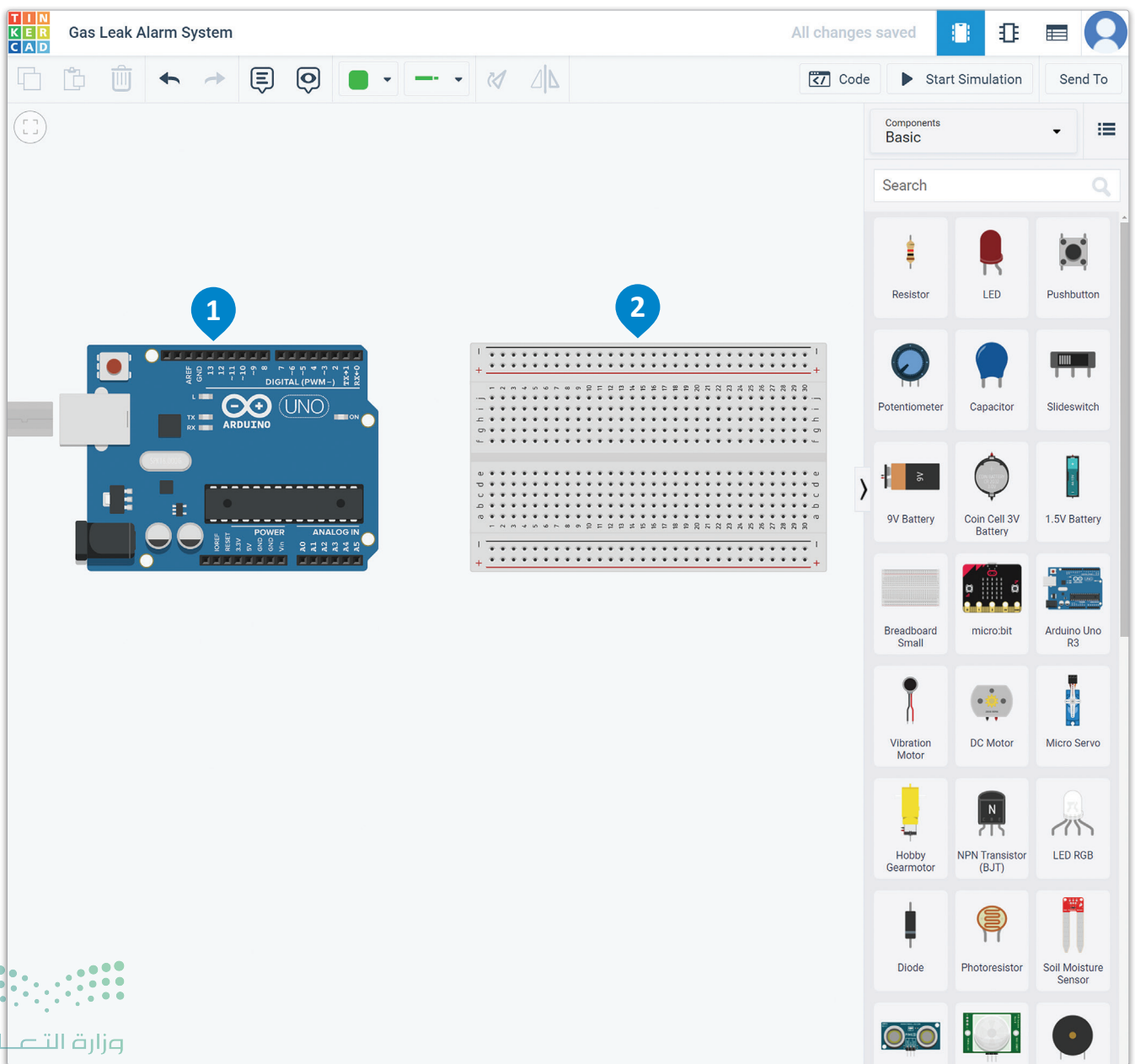


Figure 3.30: Add the Arduino UNO R3 and the Breadboard

Now, find the Gas Sensor and the Piezo buzzer, and connect them to the Breadboard.

To add components to the Breadboard Small:

- > Find the **Gas Sensor** from the **components library** ¹ and drag and drop it into the **Breadboard**. ²
- > Drag and drop the **Piezo** from the **components library** into the **Breadboard**. ³

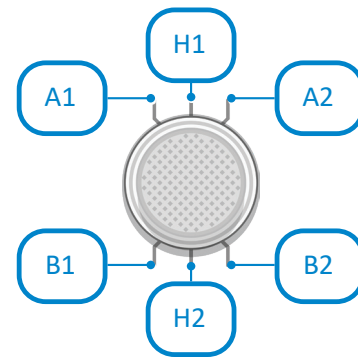


Figure 3.31: Gas Sensor

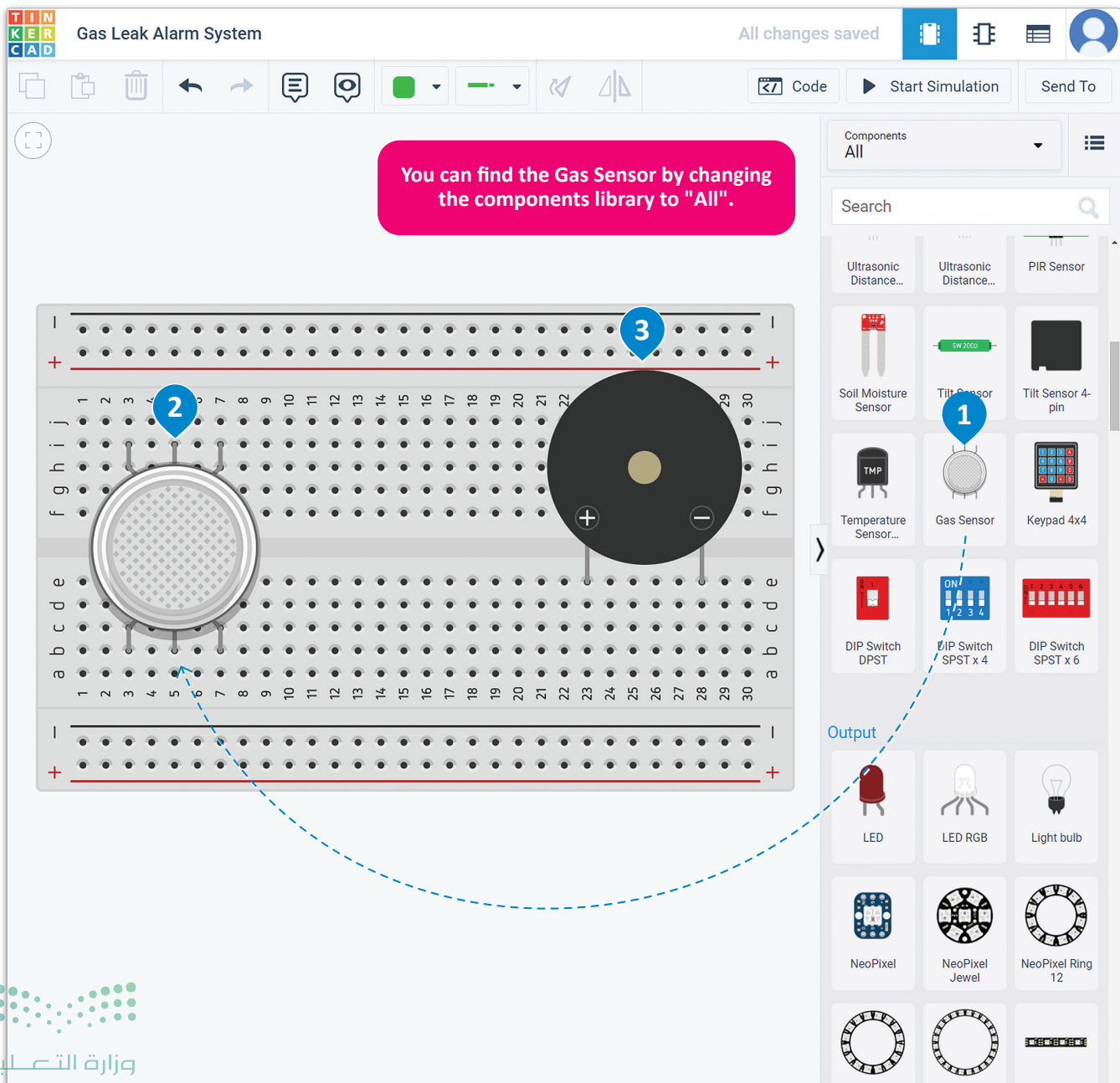


Figure 3.32: Add components to the Breadboard

Continue by adding two LEDs to the Breadboard and changing their colors.

To add LEDs:

- > Drag and drop an **LED** from the **components library** into the **Breadboard** and change its color to **orange**. **1**
- > Drag and drop another **LED** into the **Breadboard** and change its color to **green**. **2**

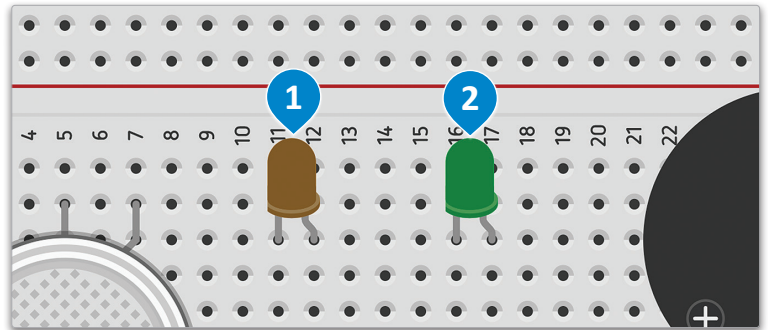


Figure 3.33: Add LEDs

Next, add resistors to the circuit and change their resistance to 220Ω. You will need three resistors; one for each LED and one for the Gas Sensor.

All LEDs have two states, on and off. When they are not lit, their color is darker. For example, the orange LED has a brown color.

To add Resistors:

- > Drag and drop two **resistors** from the **components library** and connect them to the **anodes** of the LEDs. **1**
- > Drag and drop another **resistor** from the **components library** and connect it to the **negative column** of the **Breadboard** and to the **Gas Sensor pin A1**. **2**
- > Set the **Resistance** of each resistor to **220Ω**. **3**

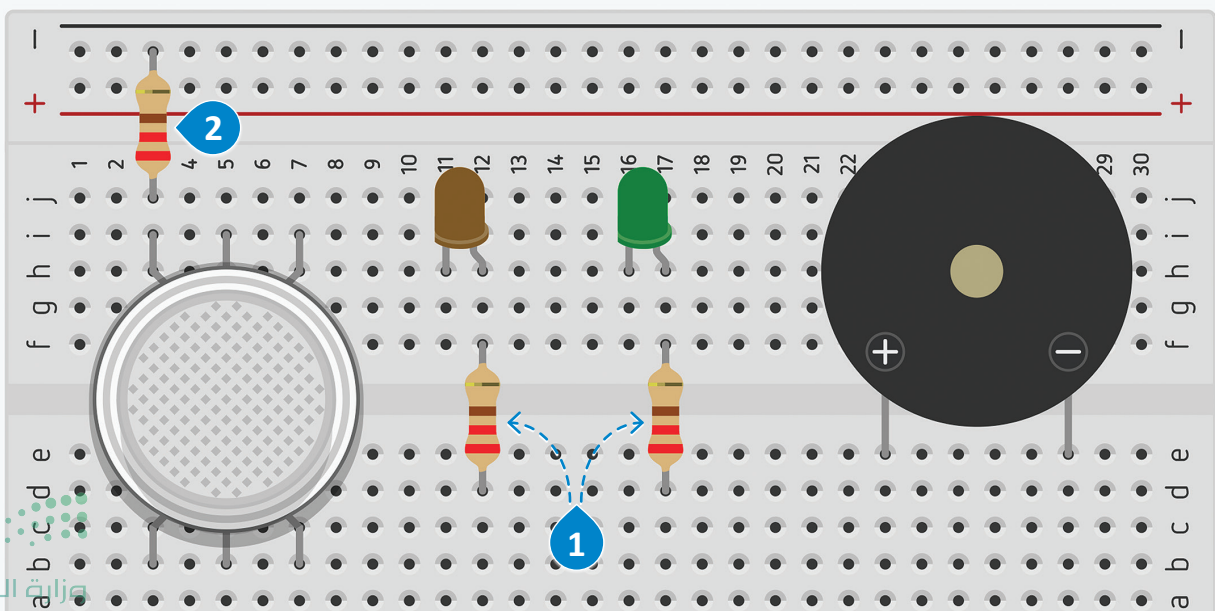
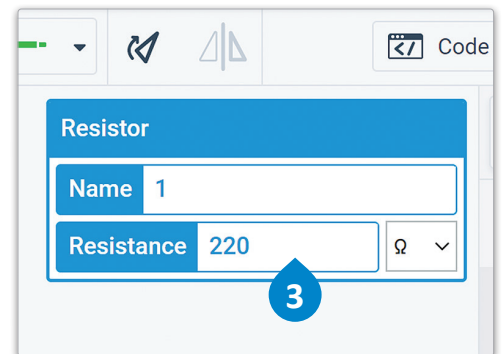


Figure 3.34: Add Resistors

Proceed with wiring the circuit.
First, you will connect the Arduino to the Breadboard.

To wire the Arduino Microcontroller:

- > Connect the **Arduino UNO R3 5V** to the **positive column** of the **Breadboard** and change the wire color to **red**. **1**
- > Connect the **Arduino UNO R3 GND** to the **negative column** of the **Breadboard** and change the wire color to **black**. **2**
- > Connect the two **negative columns** of the **Breadboard** and change the wire color to **black**. **3**

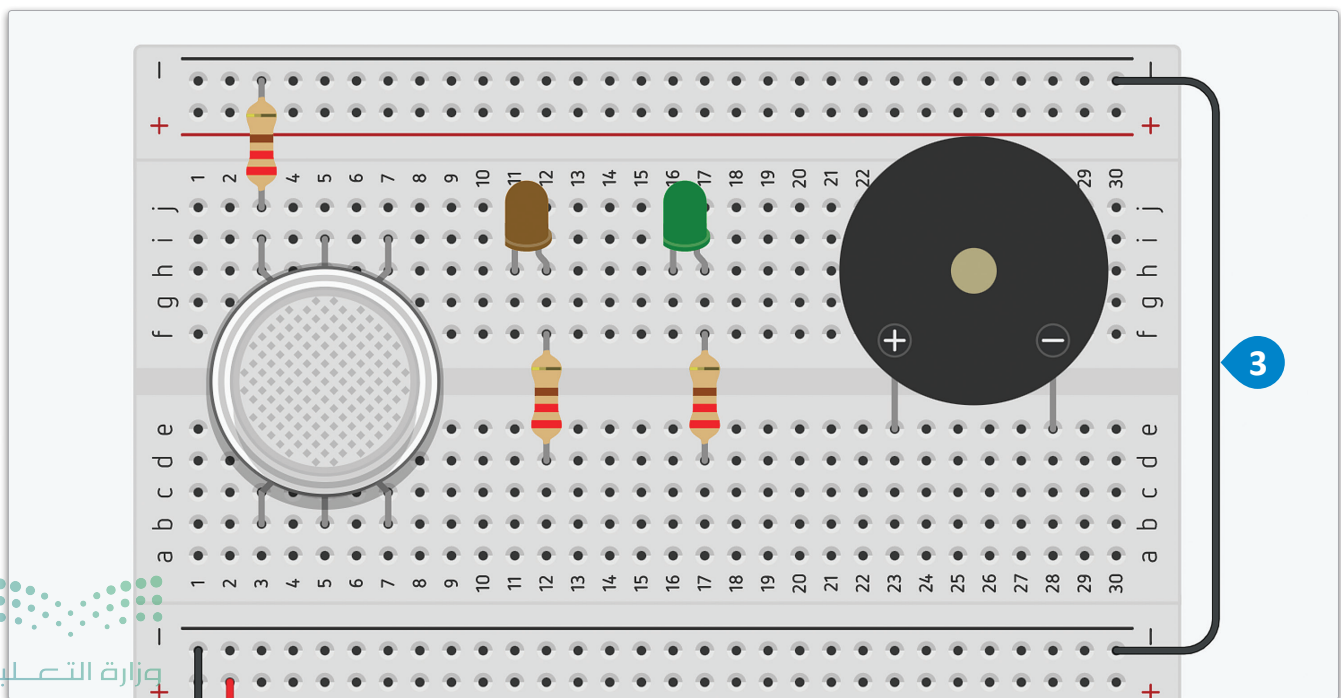
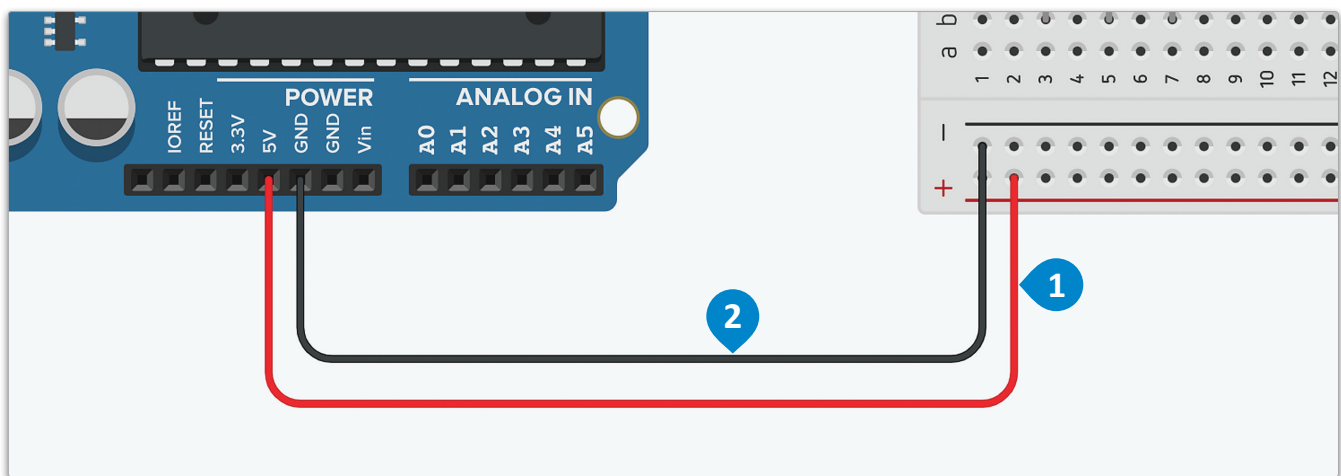


Figure 3.35: Wire the Arduino Microcontroller

Continue by wiring the Piezo buzzer.

To wire the Piezo:

- > Connect the **Piezo Positive pin** to **Analog pin A2** of the **Arduino** and change the wire color to **yellow**. **1**
- > Connect the **Piezo Negative pin** to the **negative column** of the **Breadboard** and change the wire color to **black**. **2**

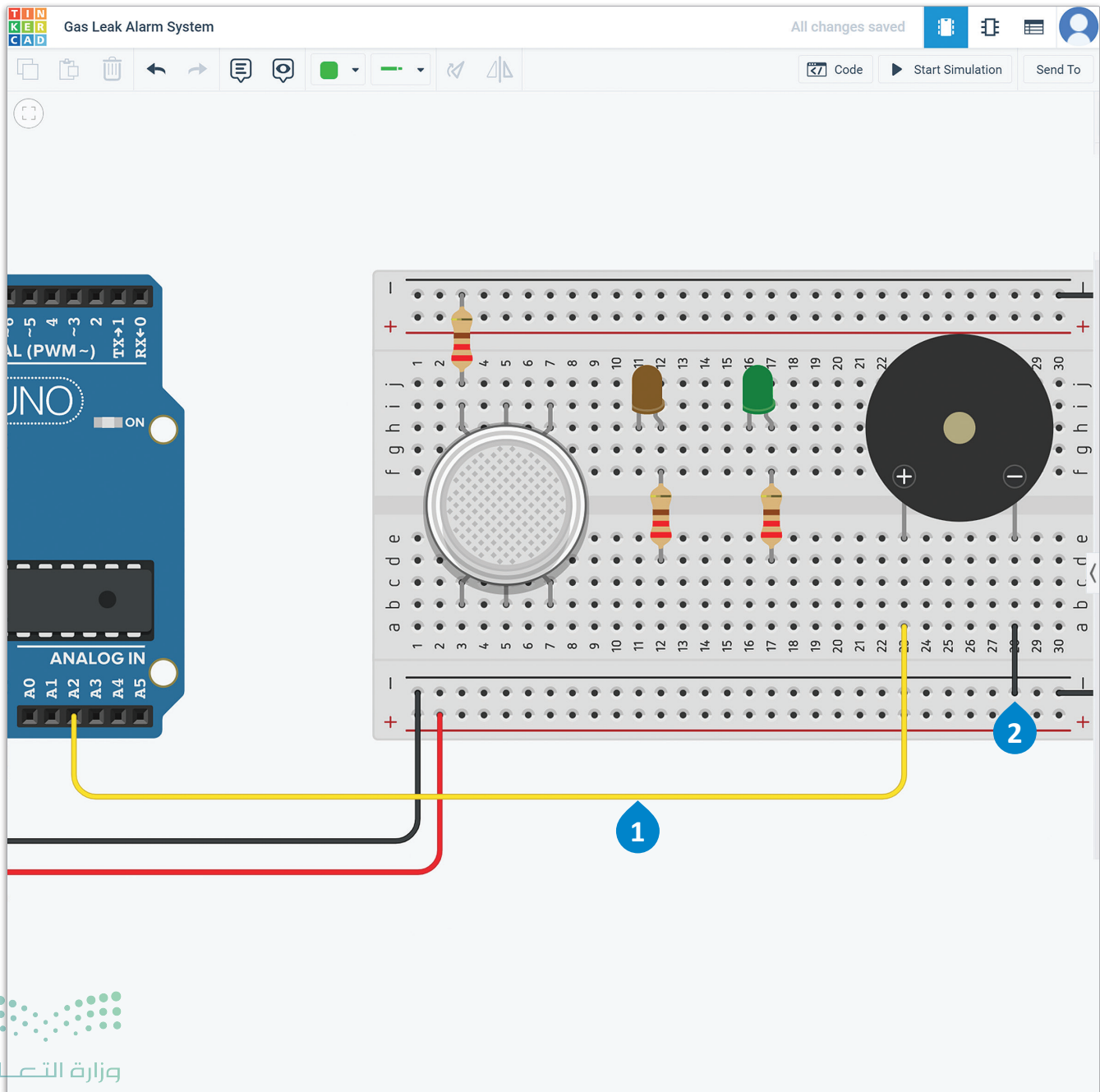


Figure 3.36: Wire the Piezo

Next, you will connect the LEDs.

To wire the LEDs:

- > Connect the **cathode** of the **Orange LED** to the **negative column** of the **Breadboard** and change the wire color to **black**. **1**
- > Connect the **resistor** of the **Orange LED** to **Digital pin 3** and change the wire color to **orange**. **2**
- > Connect the **cathode** of the **Green LED** to the **negative column** of the **Breadboard** and change the wire color to **black**. **3**
- > Connect the **Resistor** of the **Green LED** to **Digital pin 2** and change the wire color to **green**. **4**

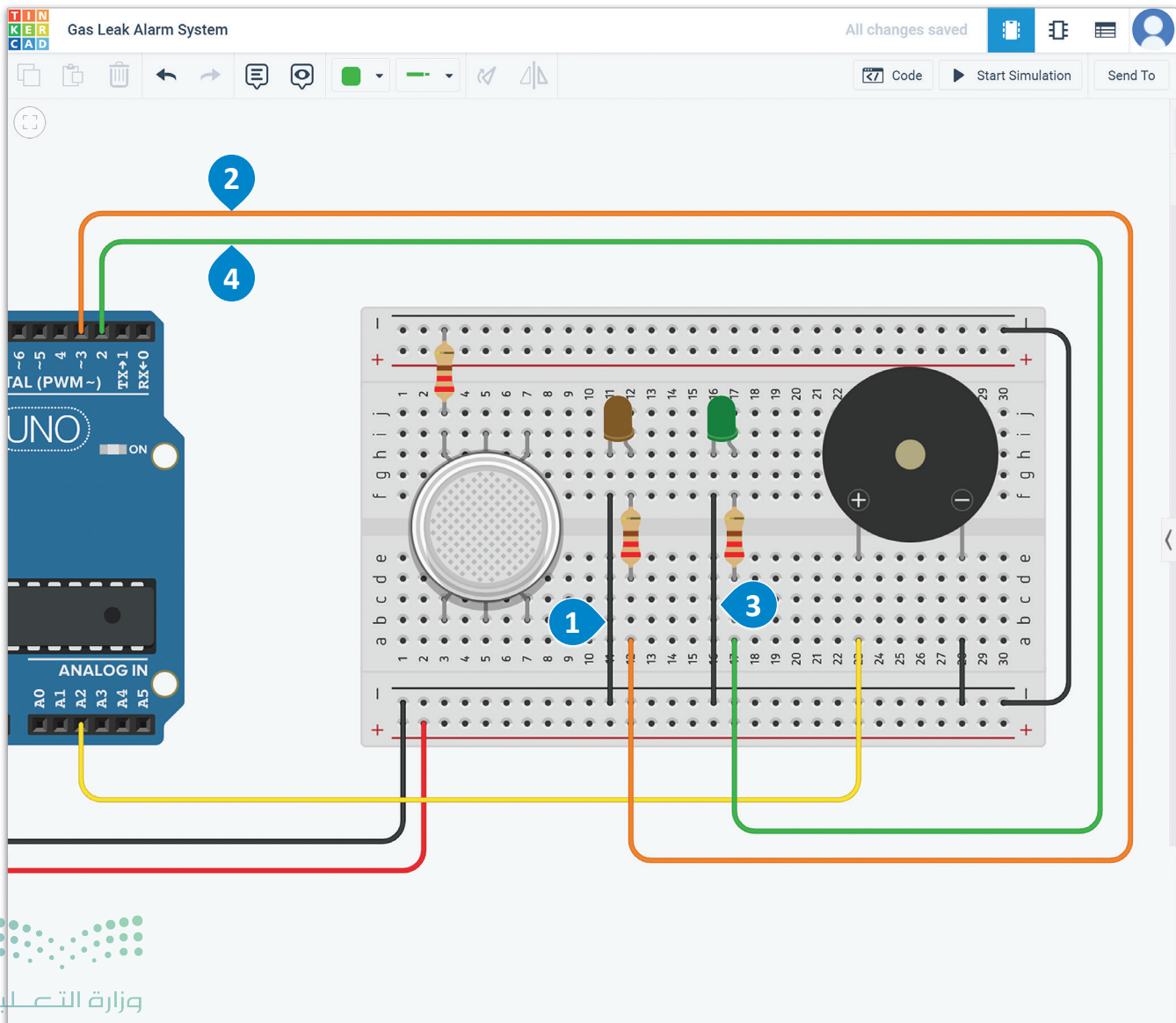


Figure 3.37: Wire the LEDs

Finally, you will wire the Gas Sensor.

To wire the Gas Sensor:

- > Connect the **Gas Sensor H1 pin** to the **negative column** of the **Breadboard** and change the wire color to **black**. ①
- > Connect the **Gas Sensor A2 pin** to the **Arduino Analog pin A4** and change the wire color to **purple**. ②
- > Connect the **Gas Sensor B1 pin** to the **positive column** of the **Breadboard** and change the wire color to **red**. ③
- > Connect the **Gas Sensor H2 pin** to the **positive column** of the **Breadboard**. ④
- > Connect the **Gas Sensor B2 pin** to the **positive column** of the **Breadboard**. ⑤

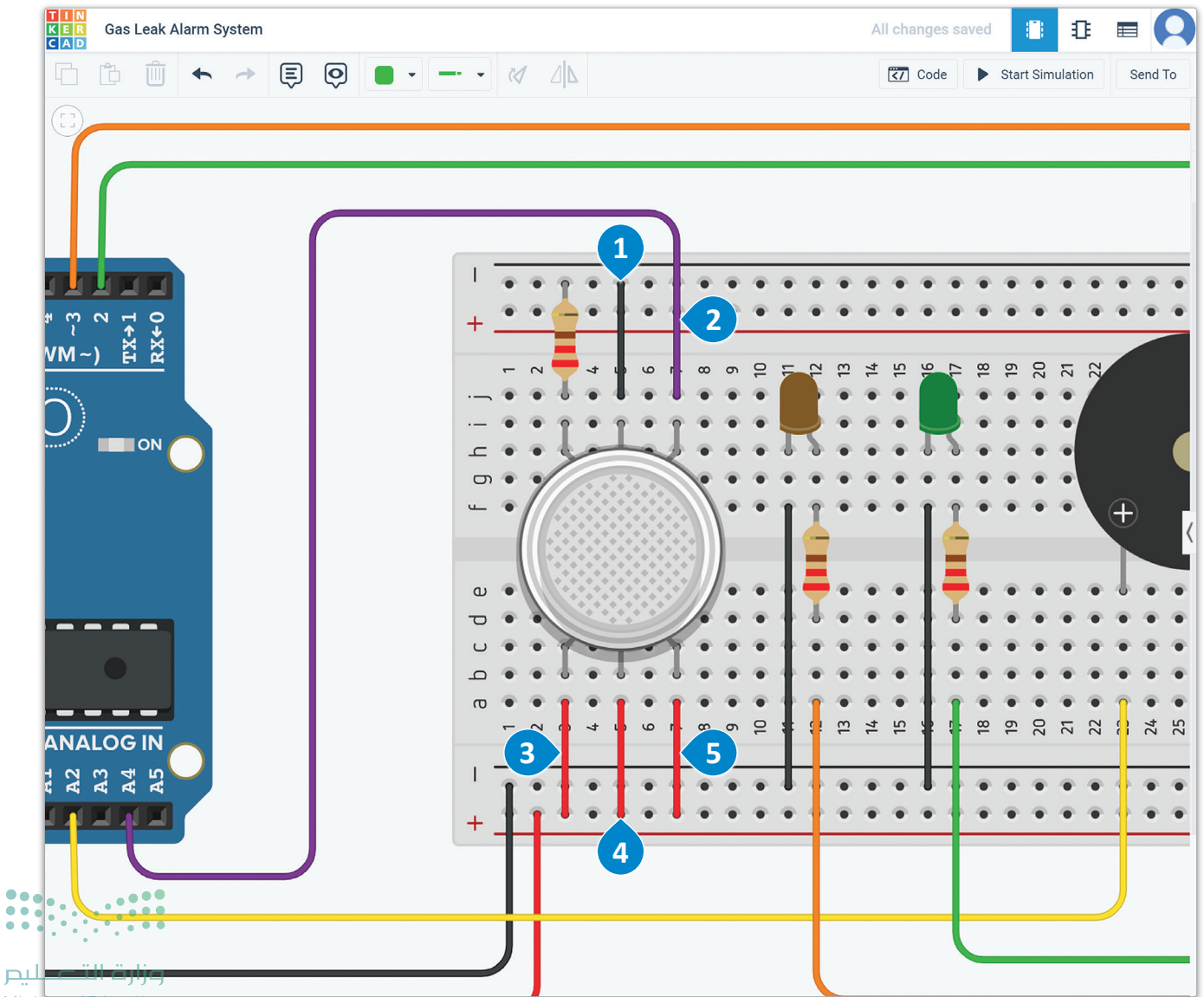


Figure 3.38: Wire the Gas Sensor

How the Gas Sensor Works:

The Gas Sensor has 6 pins; two A, two B and two H pins. The way it works is by detecting gas particles and transforming different concentrations to different electric voltages.

The purpose of the H pins is to warm the heater coil, which in turn activates the electrochemical sensor. One H pin must connect to a VCC (e.g. 5V or 3.3V) and the other H pin to Ground.

To transmit data from the Gas Sensor to be transmitted to the Arduino, either the A or the B pair has to be connected. One pin of the pair must connect to a VCC and the other pin to Ground through a resistor so the sensor's sensitivity can be adjustable. The unused pins must be connected to a VCC.

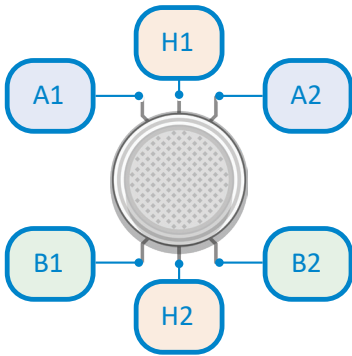


Figure 3.39: Gas Sensor

The Piezo Buzzer

The Piezo buzzer can produce a wide array of sounds of different tone and duration. To make the speaker on pin A2 play a tone of 110Hz for 1 second, use:

play speaker on pin A2 ▼ with tone 110 for 1 sec

ToneDuration

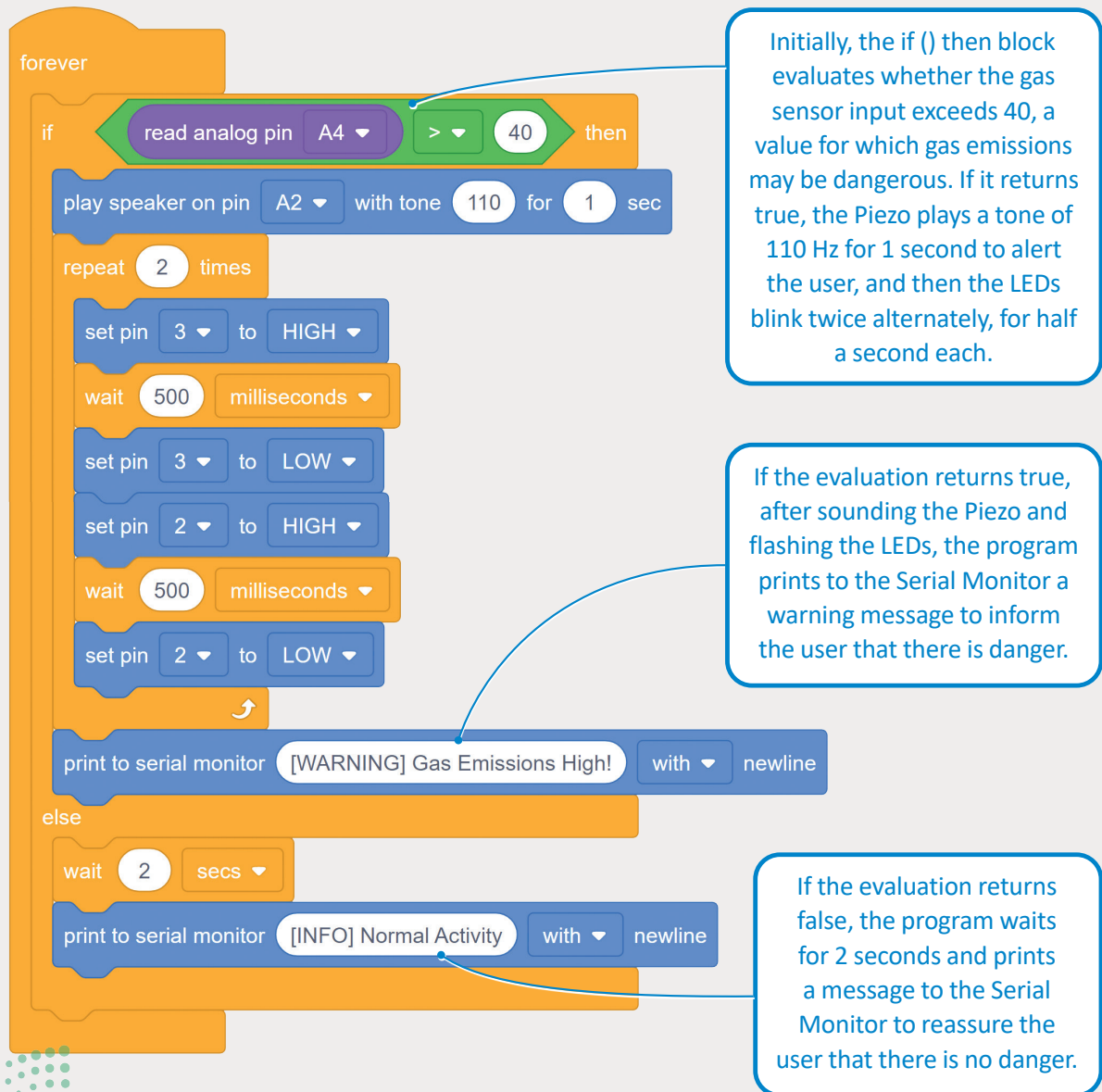
Duration is in seconds, but you may be less familiar with the units for the tone. These values correspond to the frequencies of musical notes, measured in Hertz (Hz). Some musical notes and their tones, if you want to try them, are shown here:

Table 3.3: Sound frequencies	
Notes	Frequencies
LA	110 Hz
DO	131 Hz
RE	147 Hz
FA	175 Hz

Gas Leak Alarm System Code

Now that we have wired our circuit and looked at how Gas Sensors and Piezo Buzzers work, it's time to write the code for our alarm system. The program monitors the output from the Gas Sensor to check if there is a fire hazard. If it detects a hazard, it raises the alarm by sounding the Piezo and flashing the LEDs. If not, the program waits before checking again. If it is, it alarms by playing a frequency with the Piezo and by blinking the LEDs. If it is not, the program just waits some time.

Create the following program in the coding area and press Start Simulation to run the Gas Leak Alarm System:



Run the program to test it.

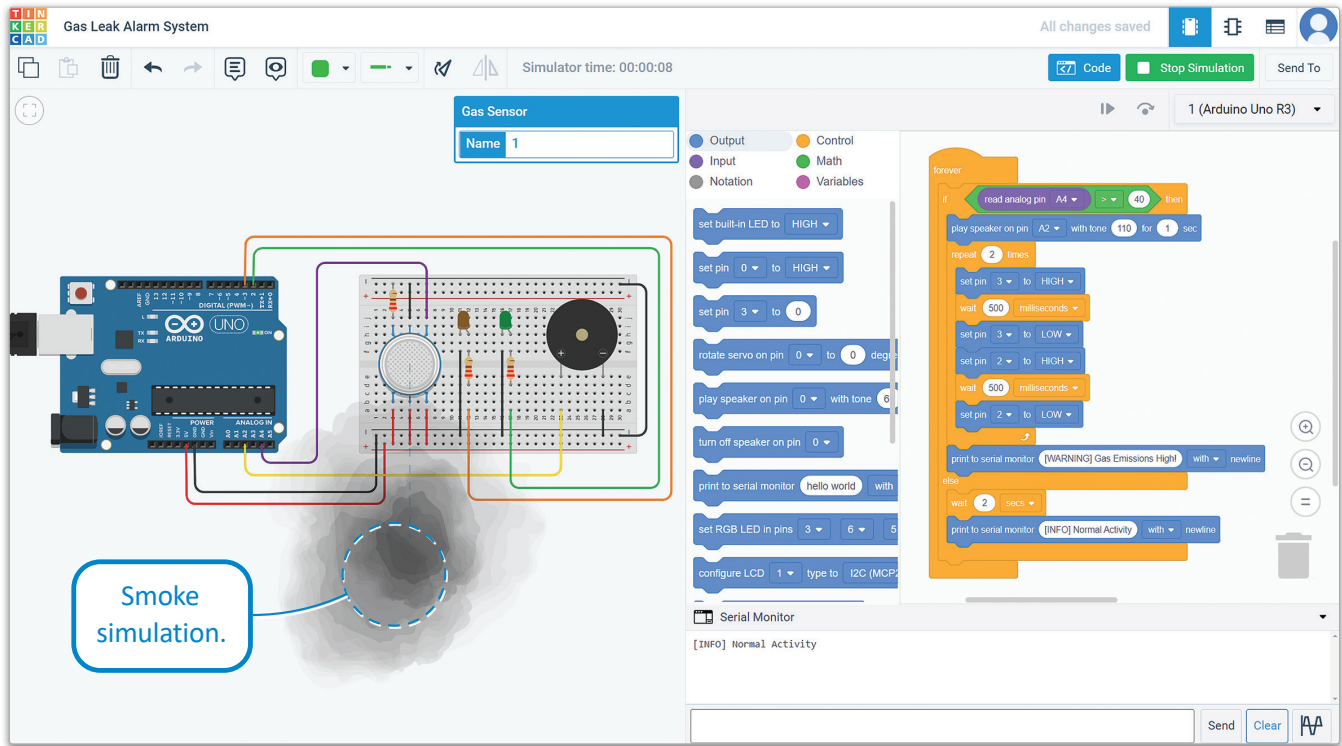


Figure 3.41: Code implementation

When the smoke cluster is away from the Gas Sensor, the Piezo does not make a sound and the LEDs do not flash.

But when the smoke cluster is close to the sensor, the Piezo starts beeping and the orange and green LEDs flash alternately.

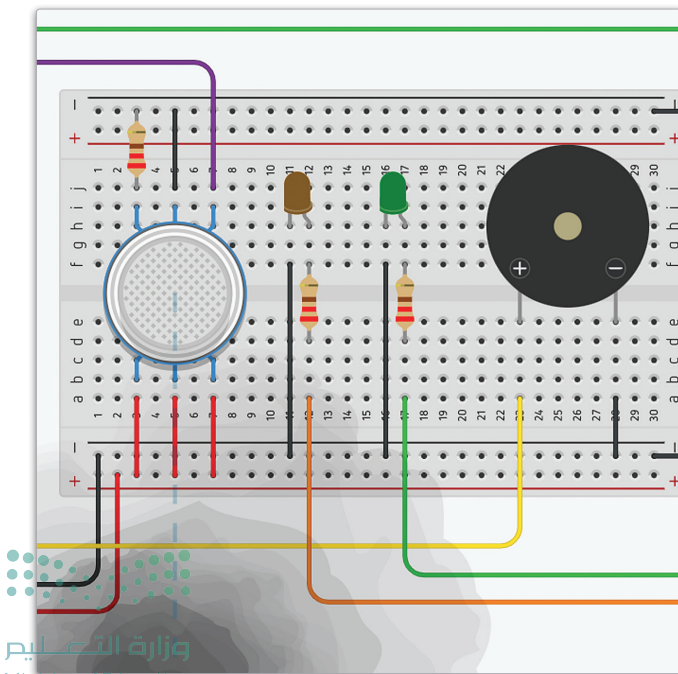


Figure 3.42: No gas detected

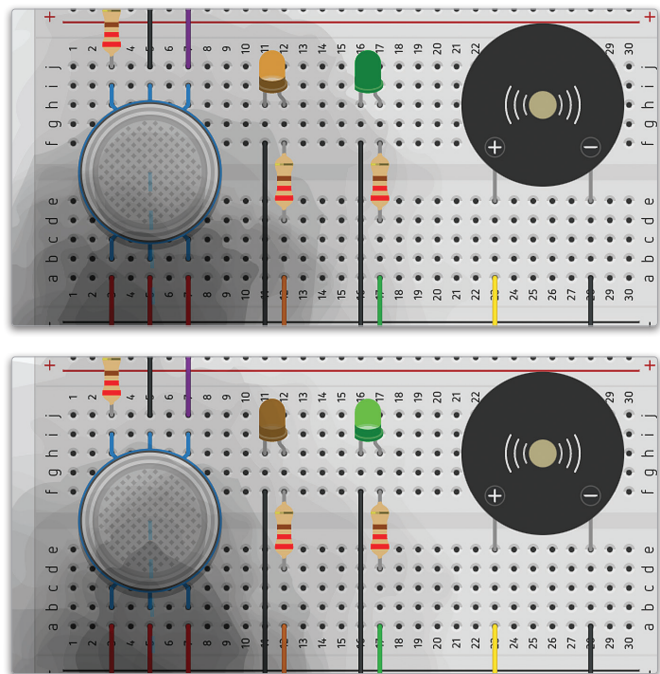


Figure 3.43: Gas detected

Exercises

- 1 Search the Internet for different types of gas hazards that could be detected with a gas sensor and analyze them. Present your findings below.

[illegible]

- 2 Consider how different tones from the Piezo buzzer could signal different alarms for different types of hazards. Present your ideas below.



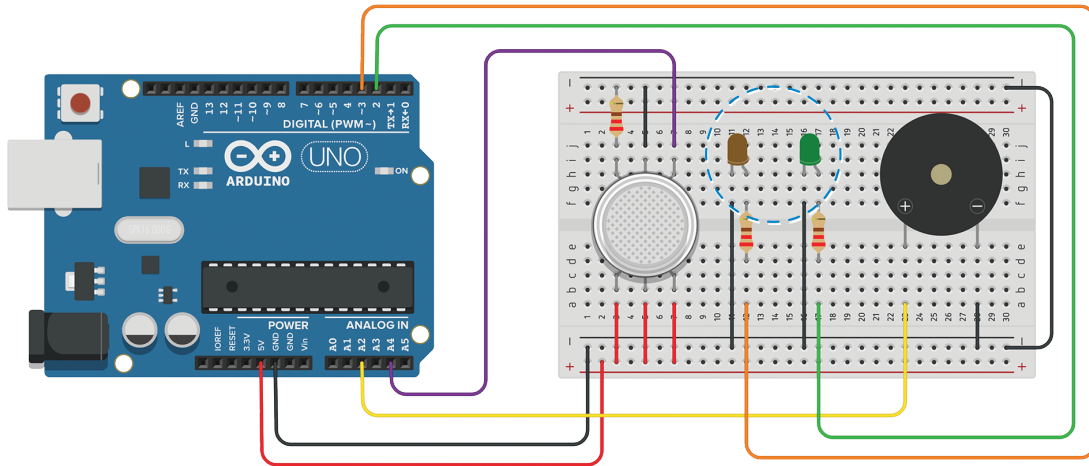
3

This image shows a single sheet of white paper with horizontal blue or grey ruling lines. The lines are evenly spaced and run across the width of the page. There are approximately 20 lines visible. The paper has a slight shadow on the right side, suggesting it's resting on a surface.

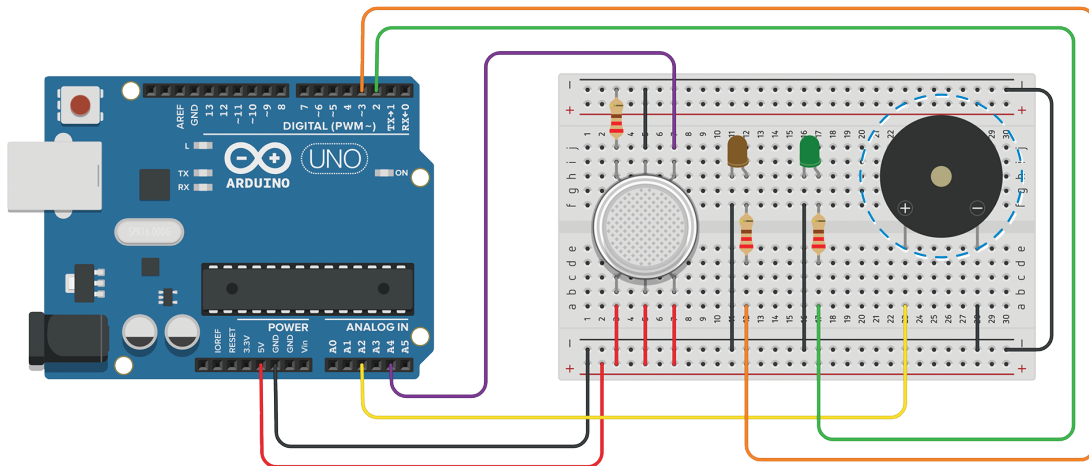
4



- 5 Change the flashing pattern of the LEDs so that, instead of each LED turning on and off twice for 500ms, only the green LED turns on and off, 3 times, for 1 second each time.



- 6 Change the beeping pattern of the Piezo so that, instead of beeping with a tone of 110Hz for 1 second, it beeps with a tone of 220Hz for 700ms.



- 7 Extend the circuit so it also outputs an additional warning when gas values are over 100.

Project

Greenhouses are very important in the field of agriculture, especially in areas where the climate and the conditions hinder agricultural investments. Greenhouses need to be monitored in order to be able to adjust conditions and protect them.

1

Design and implement a circuit in Tinkercad using an Arduino microcontroller to imitate a monitoring console of a greenhouse that will inform the user, using LEDs, whenever a sensor detects a change in the environment (e.g. motion, temperature, soil moisture, smoke).

2

Use different colors for each sensor's LED to make it easier for the user to understand what is happening.

3

Extend the design so it also outputs messages in the console when the conditions have been met. For example, when the Gas Sensor detects smoke, it could output "Fire Hazard!".



Wrap up

Now you have learned to:

- > Recognize the components of microcontrollers and to program them.
- > Measure data collected from various input sensors.
- > Understand how data from sensors and algorithms from programs work together.
- > Use threshold trigger alerts and automated responses.
- > Design IoT circuits with an Arduino microcontroller in Tinkercad Circuits.
- > Program the Arduino microcontroller using the code blocks language in Tinkercad Circuits.

KEY TERMS

Gas Sensor

Soil Moisture Sensor

Temperature Sensor

Pulse Width Modulation



4. Building an IoT Cloud Application

In this unit, you will create an IoT application that monitors an environment and sends data through a web service to a cloud platform. You will build a real circuit using an Arduino microcontroller that you will program using Python.

Learning Objectives

In this unit, you will learn to:

- > Use Python to program an Arduino microcontroller with PyFirmata.
- > Design a circuit for an IoT application using an Arduino microcontroller.
- > Monitor an environment and collect sensor data in real-time.
- > Use a web service to send the collected data to a cloud platform.
- > Monitor a remote environment with data provided in a cloud platform.
- > Understand how data from sensors and cloud datasets are used for data-driven decision making.
- > Understand how IoT applications can be scaled to evolve into complex solutions.

Tools

- > Arduino IDE
- > JetBrains PyCharm
- > Binary IoT Cloud



Lesson 1

Setup Arduino Development Environment

Link to digital lesson



www.iien.edu.sa

Using Python with Arduino

The official programming language for the Arduino microcontroller is **C++** but you can use **Python** through the **Firmata** protocol. Python is a high level programming language and its greatest strength is the wide range of libraries that can support any scenario. The Firmata protocol allows for a connection between a microcontroller and the commands provided by another programming language. In our case, we will use Python with the **PyFirmata** library which is an interface for the Firmata protocol.

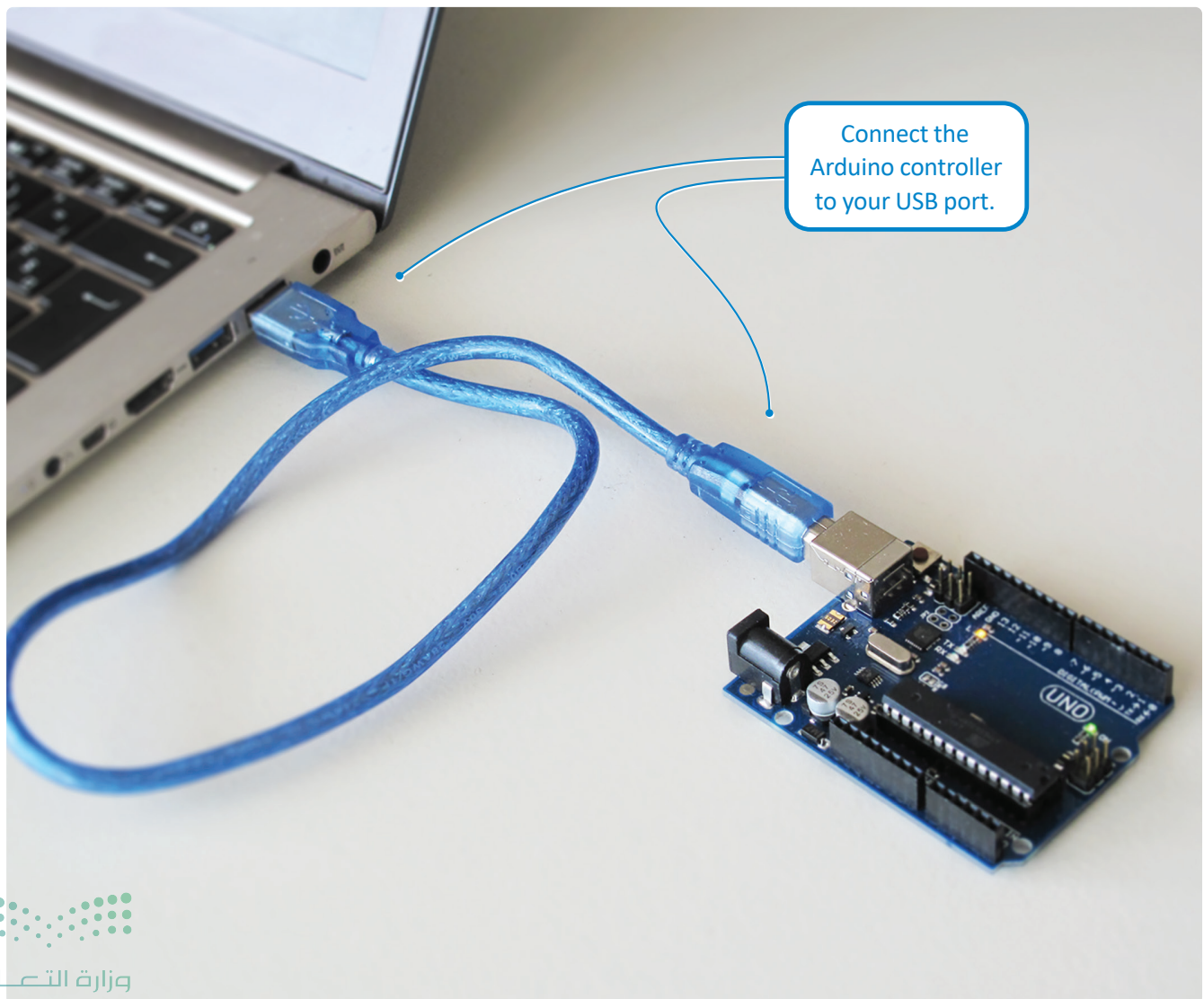


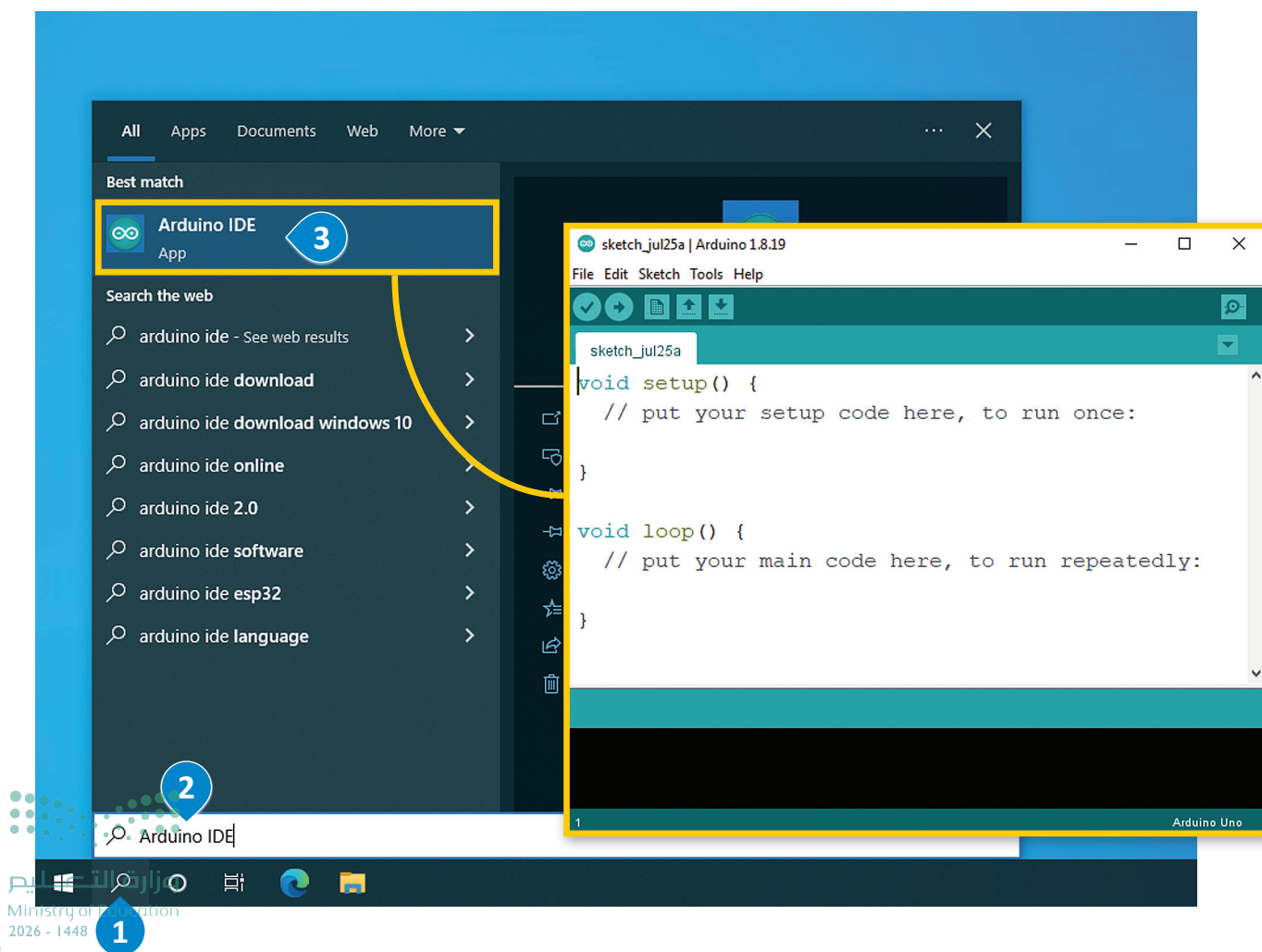
Figure 4.1: Arduino connected to laptop USB port

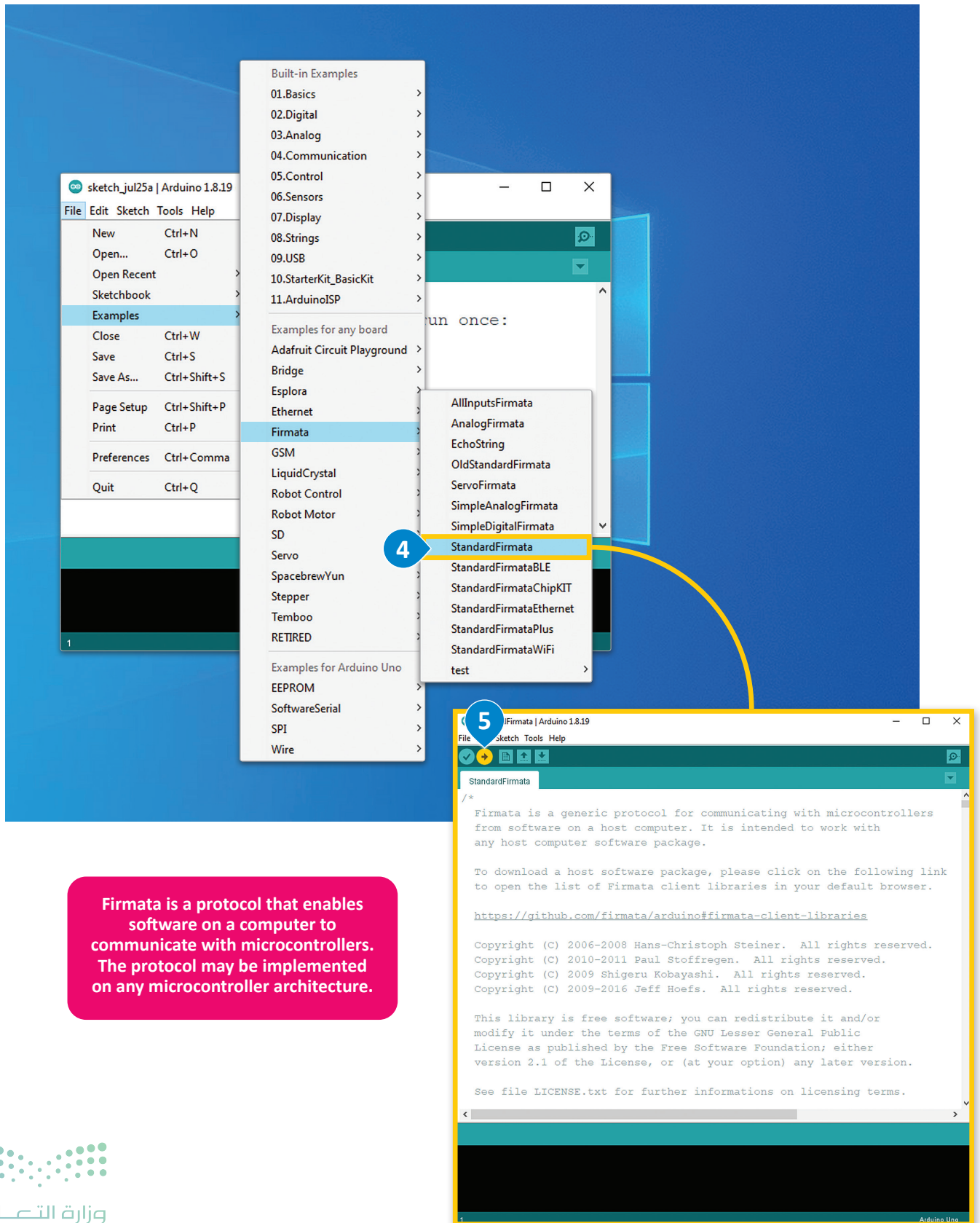
The Arduino Integrated Development Environment (Arduino IDE) is a text editor specifically designed for Arduino microcontrollers. It comes with additional tools for interacting with an Arduino and contains pre-built programs for specific Arduino tasks. To install the Arduino IDE go to the website: <https://www.arduino.cc/en/software> and download the latest version. Run the installer to complete the installation process. You will now open the Arduino IDE in order to load the StandardFirmata sketch to set up communication between the Arduino and your Python code. Make sure that the Arduino IDE version you are downloading is compatible with your Arduino microcontroller device.

Let's start by finding and running the Arduino IDE from your computer's search button.

To load the StandardFirmata sketch:

- > Click the **Windows search button** ¹ and type **Arduino IDE**. ²
- > Open the **Arduino IDE**. ³
- > Load the **StandardFirmata** sketch into the Arduino IDE by clicking on **File > Examples > Firmata > StandardFirmata**. ⁴
- > **Upload** the sketch to your Arduino. ⁵





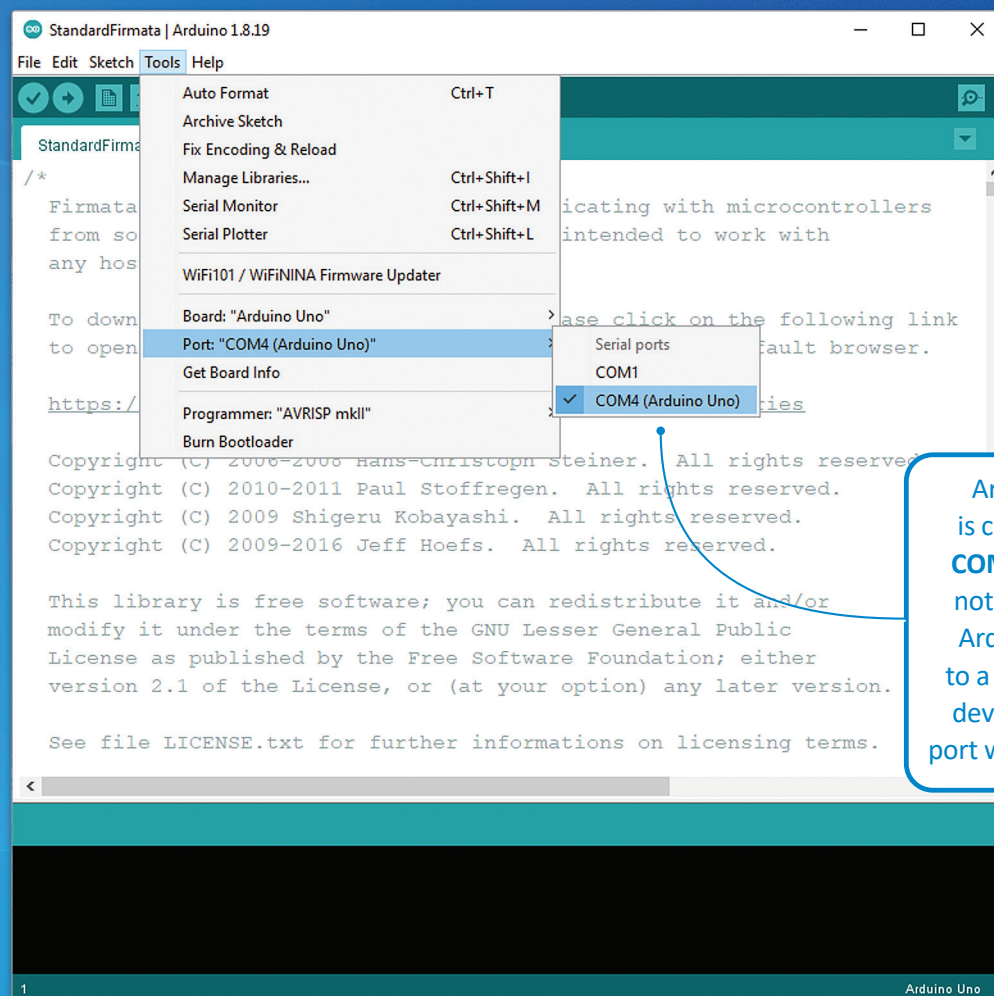
Firmata is a protocol that enables software on a computer to communicate with microcontrollers. The protocol may be implemented on any microcontroller architecture.



Figure 4.2: Loading the Firmata sketch

You can get the **communication port** from your PC to Arduino by clicking on *Tools > Port > Serial ports* as demonstrated below.

In this example, the communication port is COM4, on your PC it might be different, e.g. COM3, COM5. Make a note of the communication port because you will use it in your Python code to communicate with Arduino.



Arduino Uno is connected to **COM4**. If you do not connect the Arduino device to a USB port in a device, then the port will be inactive.

Figure 4.3: Viewing the Arduino communication port

Whenever you control an Arduino microcontroller with Python, you need to keep the StandardFirmata sketch running so that the Python code that you write can communicate with Arduino.



Now that you have loaded **StandardFirmata** onto the Arduino microcontroller, you need to follow these steps for each Python project:

Open **PyCharm** and install the **pyfirmata** Python package through **pip**. In **PyCharm** open the terminal in your working directory and enter the following command:

```
pip install pyfirmata
```

Create a new Python file and at the beginning of your code import the **pyfirmata** package with the line:

```
import pyfirmata
```

Create a variable called *communication_port* which will store the name of the **communication port** of your PC where the Arduino is connected:

```
communication_port = "COM4"
```

Use the following commands to set up the connection between your Python program and your Arduino, and to access the pins of the Arduino:

```
# Set the Arduino port to read from
board = pyfirmata.Arduino(communication_port)

# Set up access to the inputs of the circuit
it = pyfirmata.util.Iterator(board)
it.start()
```

For the commands that you give the Arduino, you need an infinite loop so that they will be executed continuously.

```
while True:
    # add your code here
```

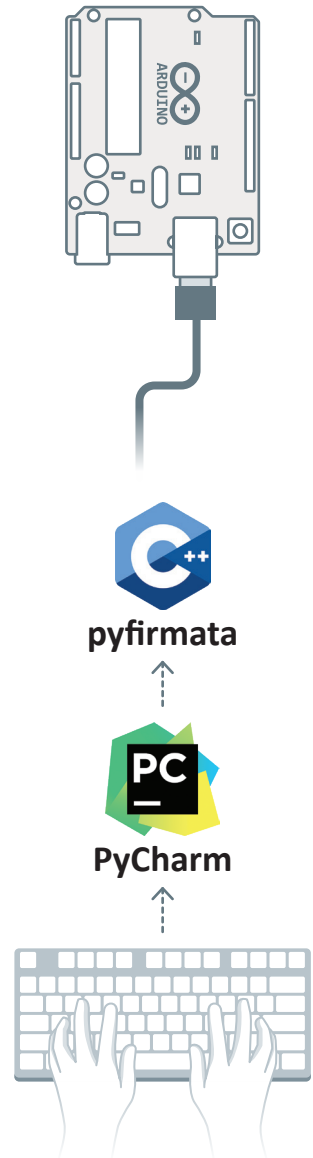


Figure 4.4: Coding and Arduino with Python through pyfirmata

Although the connection is through a USB cable, Windows is using a traditional serial port connection to exchange data with the Arduino device. The OS takes care of the virtual serial port.



The code for every Python program for Arduino starts like this:

```
import pyfirmata

communication_port = "COM4"

# Set the Arduino port to read from
board = pyfirmata.Arduino(communication_port)

# Set up pyfirmata to access the status of the inputs of the circuit
it = pyfirmata.util.Iterator(board)
it.start()

while True:
    # add your code here
```

The code that you implement is run on your own PC, not on the Arduino, which means that you can access functions that are not available on the Arduino microcontroller.

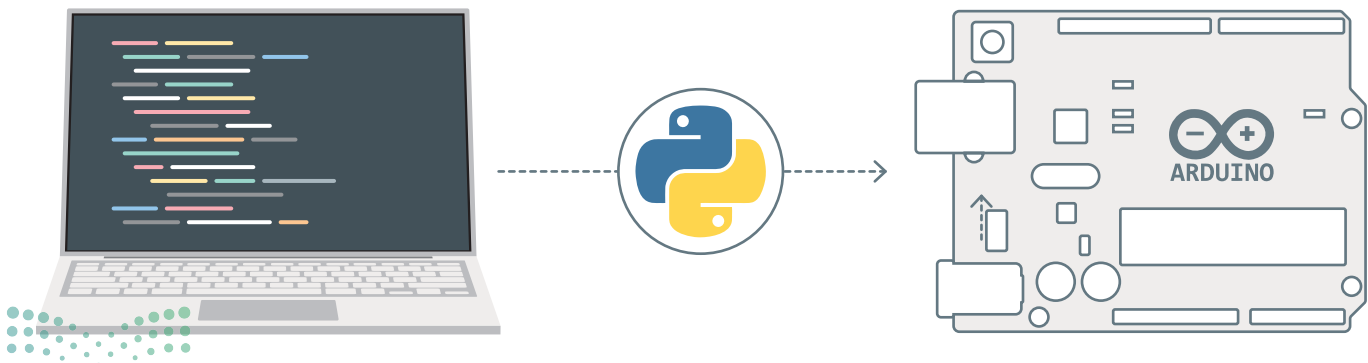


Figure 4.5: Controlling an Arduino microcontroller with Python code

Working with PyFirmata

The main functionalities that we need to implement in Python with the use of PyFirmata are the reading and the writing of values to both the analog and digital pins of the Arduino microcontroller.

In **pyfirmata** the above action is done by the **board.get_pin()** method which takes 3 character arguments separated by a colon.

The first argument is 'a' or 'd' which means **analog** or **digital** pin.

The second argument is the number of the pin that you want.

- Digital pins are numbered **0-12**.
- Analog pins are numbered **A0-A5**.

The third argument is the type of interaction with the Arduino microcontroller.

- For digital pins it is 'i' or 'o' for **input** or **output**.
- For analog pins it is 'i' or 'p' for **input** or **PWM (Pulse Width Modulation)**.

Pulse Width Modulation (PWM)

Pulse Width Modulation (PWM) is a type of modulation that uses a digital output to make an analog signal with varying power.

Pulse Width Modulation

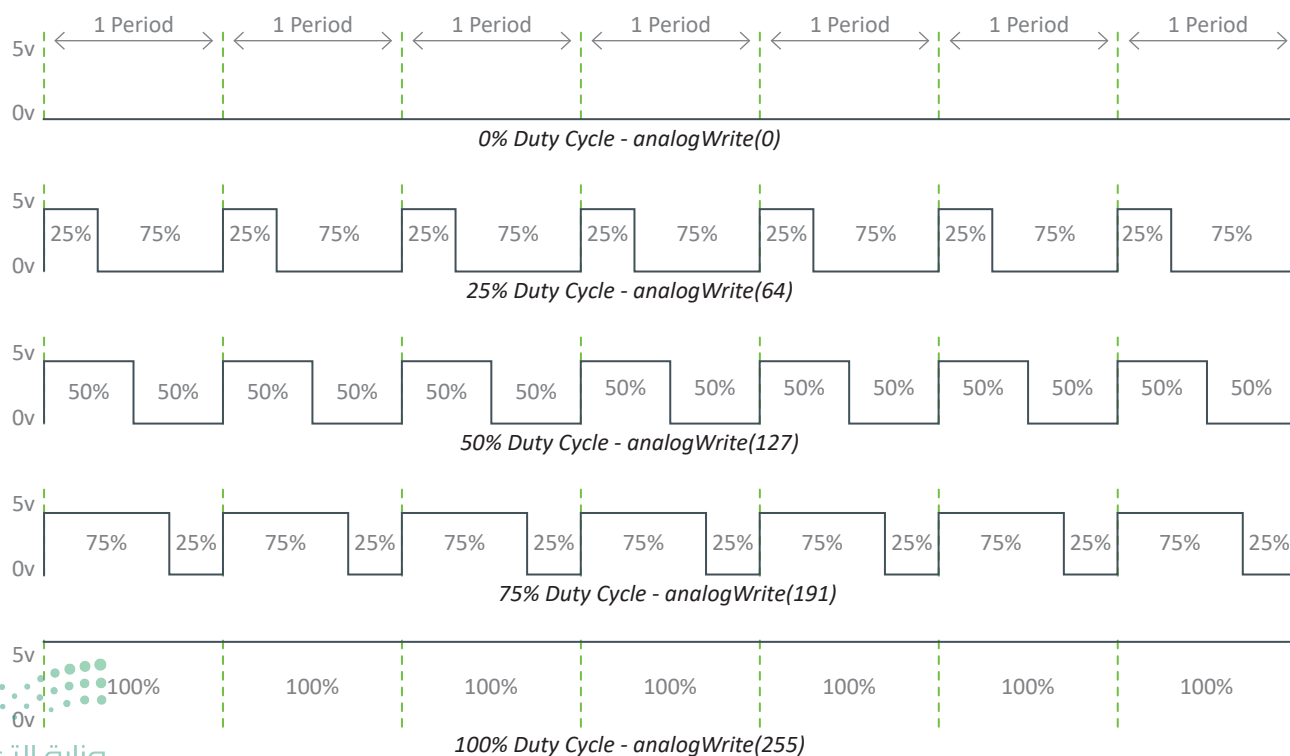


Figure 4.6: Pulse Width Modulation

Interacting with Digital Pins

Reading a value from a digital pin (digital pin 10).

```
digital_pin = board.get_pin("d:10:i")  
pin_value = digital_pin.read()
```

Writing a value to a digital pin (digital pin 10).

```
digital_pin = board.get_pin("d:10:o")  
digital_pin.write(1)  
digital_pin.write(0)
```

Interacting with Analog Pins

Reading a value from an analog pin (analog pin 4).

```
analog_pin = board.get_pin("a:4:i")  
pin_value = analog_pin.read()
```

Writing a value to an analog pin (analog pin 4).

```
analog_pin = board.get_pin("a:4:p")  
analog_pin.write(0.75)
```

Each sensor or actuator
require different values
to work properly.

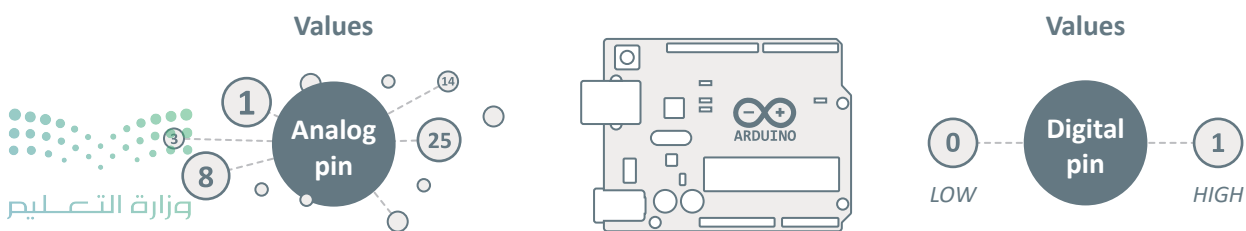


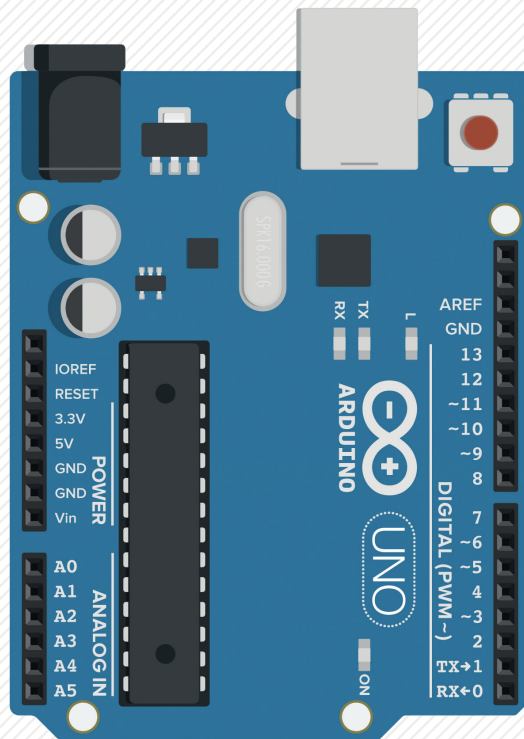
Figure 4.7: The difference between reading data from digital and analog pins

A Simple Arduino Project with PyFirmata

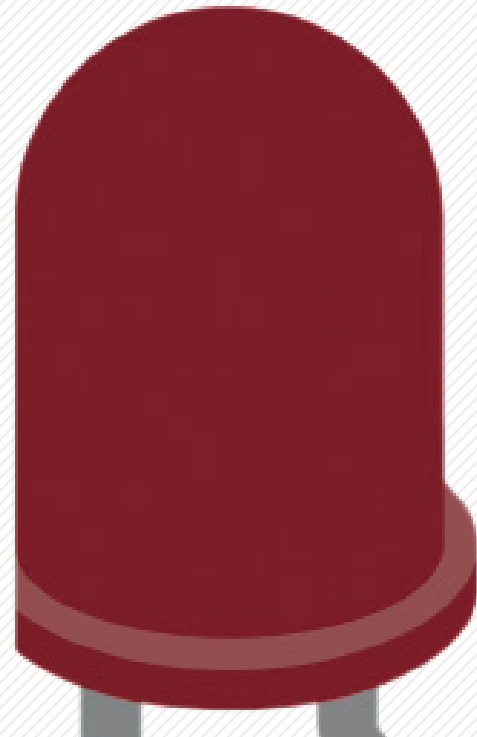
To demonstrate the use of the PyFirmata library, we will create a simple Arduino project using an external LED light and the integrated LED pin of an Arduino. We will use Python to program each LED light to blink alternately. First we will simulate the circuit in Tinkercad Circuits. You will need the following components:

- 1 Arduino Uno R3
- 1 Breadboard Small
- 1 LED
- 1 Resistor

Components that you will use in this project



Arduino Uno R3



Breadboard Small



LED



Resistor

Let's start by putting the required components into the Tinkercad Circuits workspace.

To load the components:

- > Find the **Arduino Uno R3** from the components library and drag and drop it into the **workplane**. **1**
- > Find the **Breadboard Small** from the components library and drag and drop it into the **workplane**. **2**

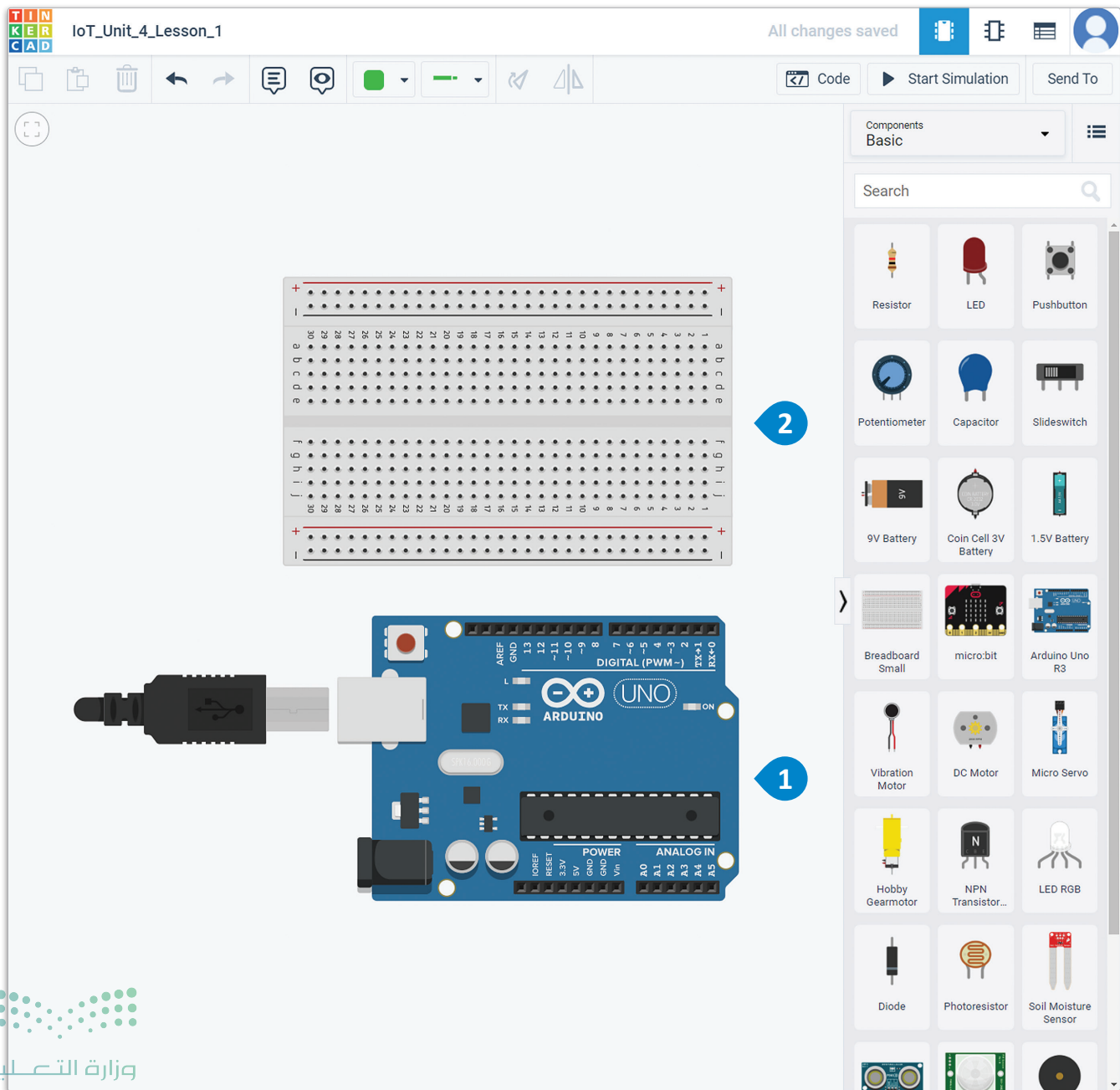


Figure 4.9: Loading the components

Now you will complete the wiring to connect the Arduino to the external LED.

To wire the Arduino:

- > Connect the **Arduino Uno R3 GND** to the **negative column** of the **Breadboard Small** and change the wire color to **black**. **1**
- > Find the **Resistor** from the components library, place it on the **Breadboard Small** **2**, connect the **Digital pin 12** to **Terminal 1** of the **Resistor** and change the wire color to **red**. **3**

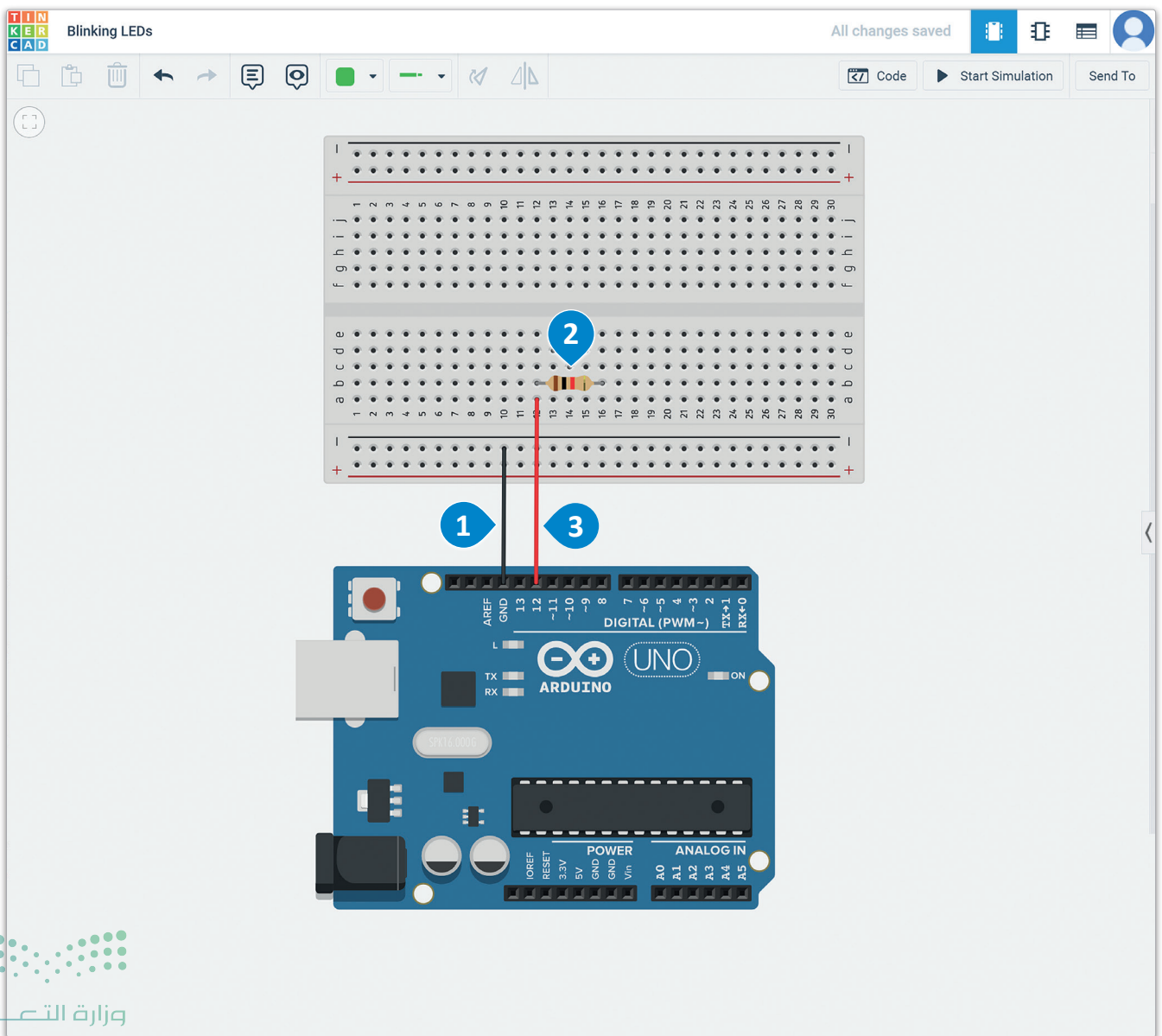


Figure 4.10: Wiring the Arduino

To connect the LED:

- > Find the **LED** from the components library, place it on the **Breadboard Small**. **1**
- > Connect **Terminal 2** of the **Resistor** to the **Anode** of the LED. **2**
- > Connect the **Cathode** of the LED to the **negative column** of the **Breadboard Small**. **3**

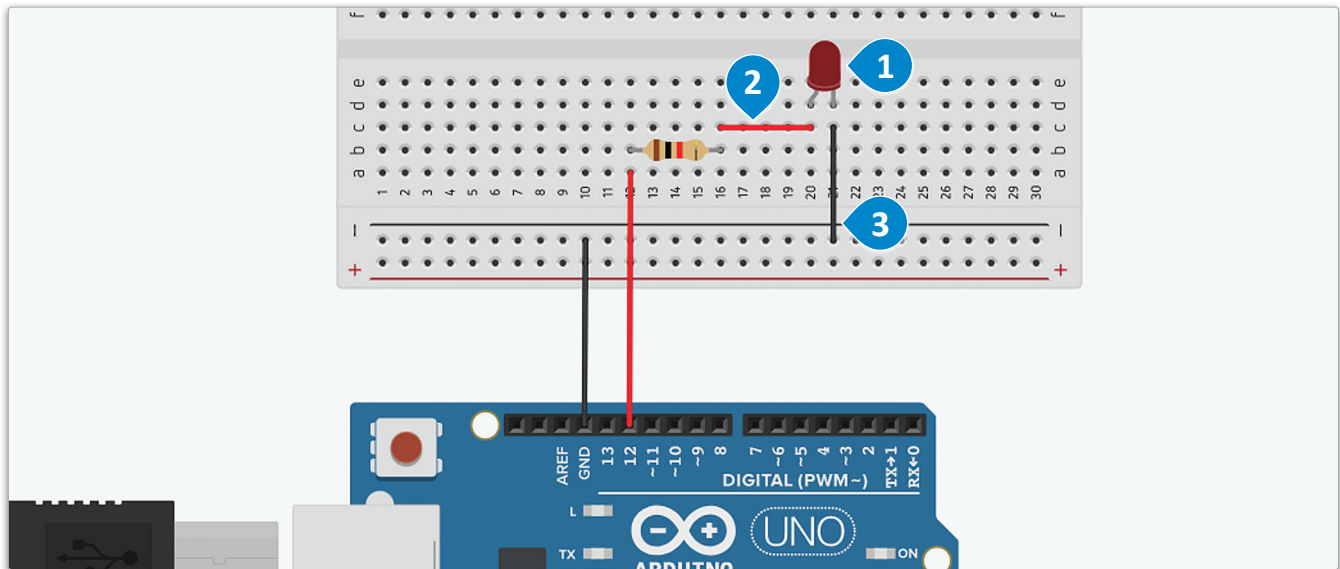


Figure 4.11: Connecting the LED

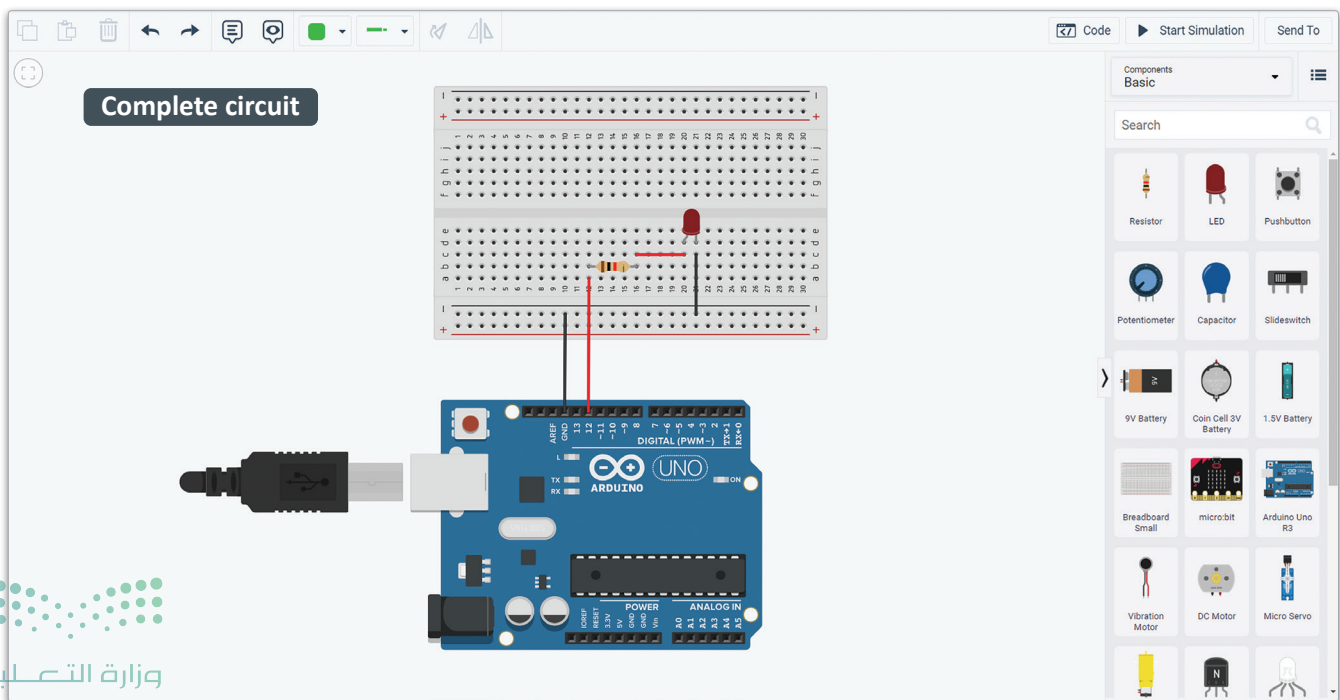


Figure 4.12: Complete circuit in Tinkercad

Physical Circuit

This photo represents the physical circuit's look.

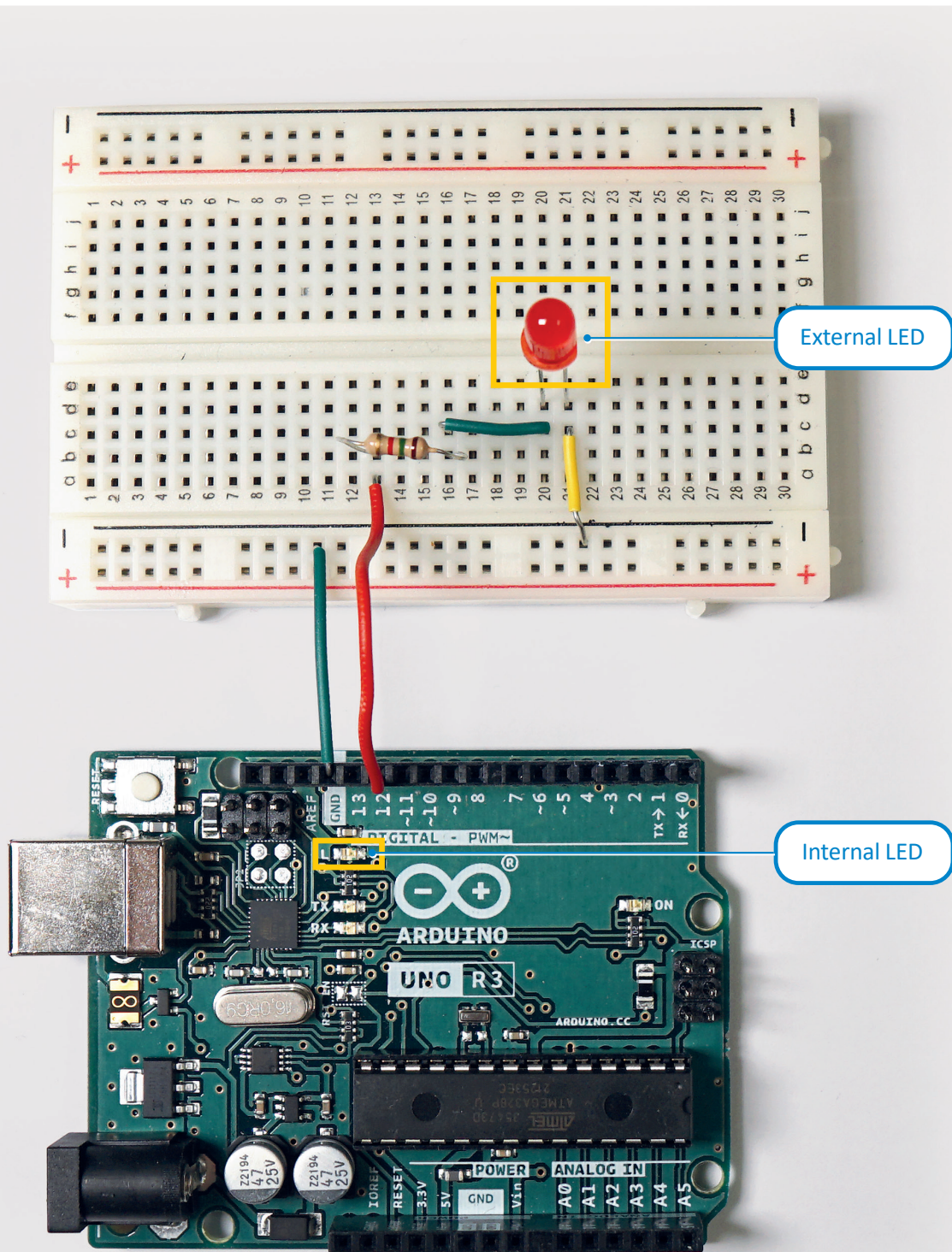


Figure 4.13: Photo of the physical circuit

Programming the Arduino to Blink

We will program the two LEDs to blink one after the other with a time difference of one second. The built-in LED on the Arduino is connected to digital pin 13, and the external LED is connected to the digital pin 12. Inside the infinite loop, we will send a HIGH (1) signal to the LED that we want to shine and a LOW (0) signal to the other LED. After one second, we will reverse the signals.

Open **PyCharm**, create a new Python file and import the required libraries.

```
import pyfirmata
import time
```

Set up the communication port.

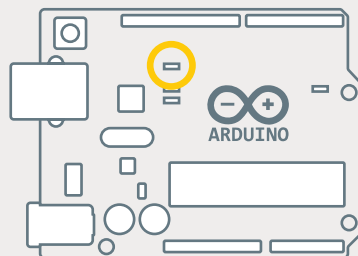
```
communication_port = 'COM4'
```

Set up the communication between PyFirmata and the board.

```
board = pyfirmata.Arduino(communication_port)
it = pyfirmata.util.Iterator(board)
it.start()
```

Get the pins for the external LED and the internal LED.

```
external_led = board.get_pin("d:12:o")
internal_led = board.get_pin("d:13:o")
```



D13

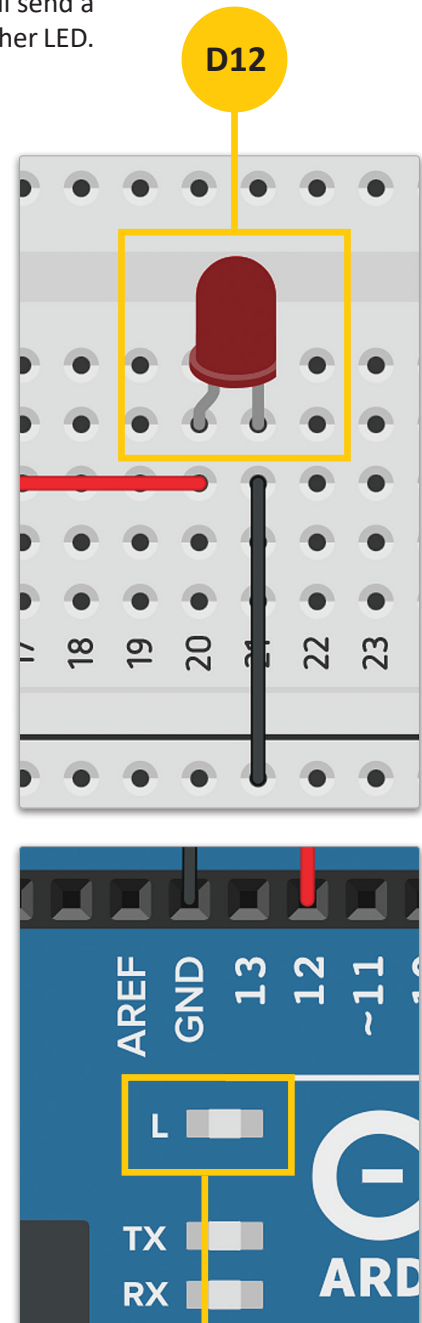


Figure 4.14: Connected pins to components



Write the logic to make the lights blink.

```
while True:
    external_led.write(1)
    internal_led.write(0)

    time.sleep(1)

    external_led.write(0)
    internal_led.write(1)

    time.sleep(1)
```

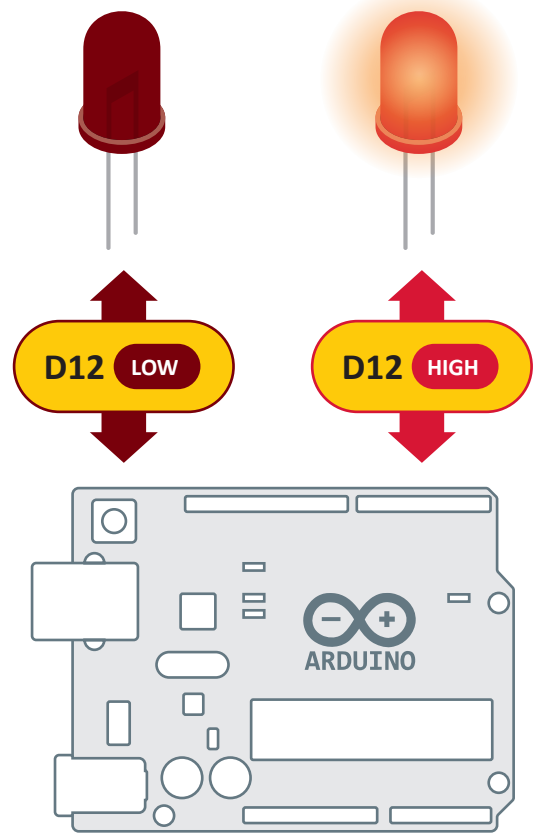
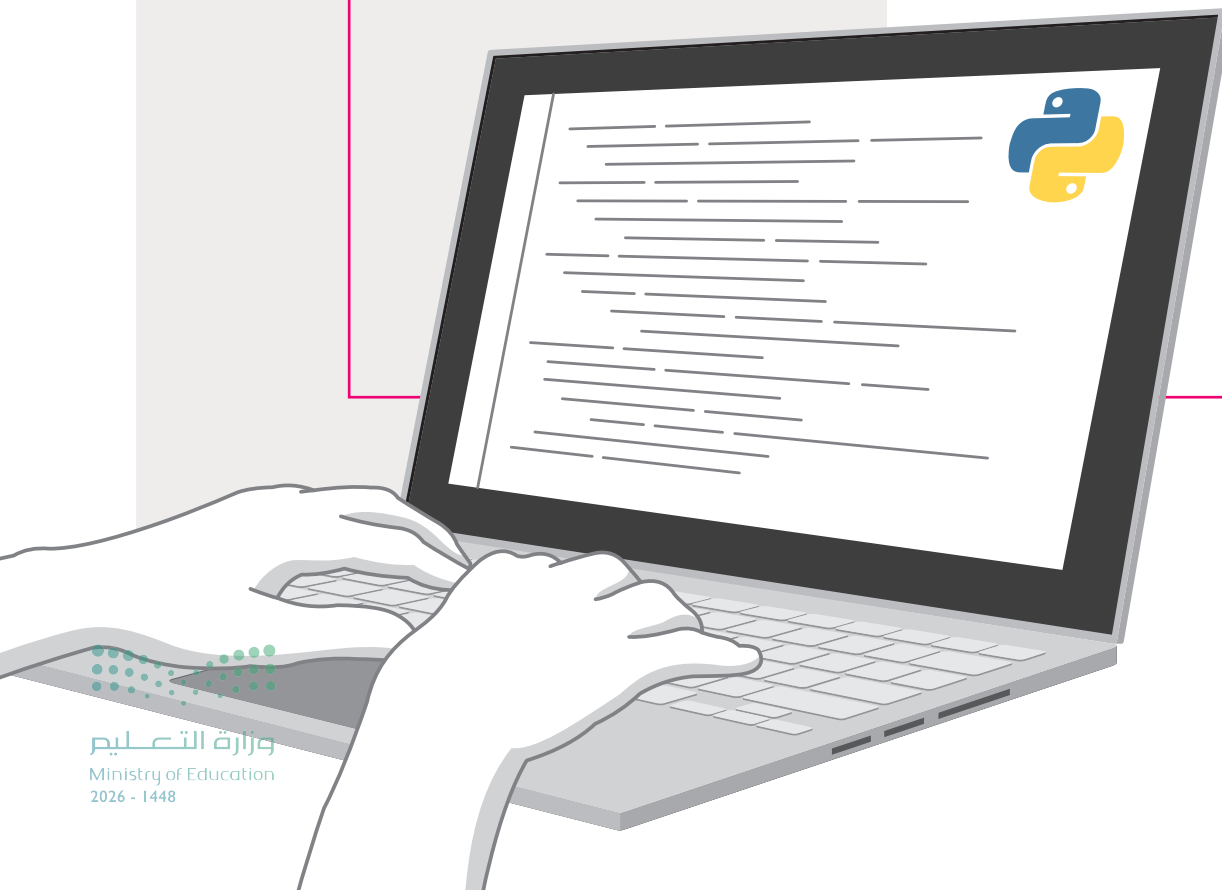


Figure 4.15: Arduino sending digital signal to pins



Complete Code

```
import pyfirmata
import time

communication_port = 'COM4'

board = pyfirmata.Arduino(communication_port)
it = pyfirmata.util.Iterator(board)
it.start()

external_led = board.get_pin("d:12:o")
internal_led = board.get_pin("d:13:o")

while True:
    external_led.write(1)
    internal_led.write(0)

    time.sleep(1)

    external_led.write(0)
    internal_led.write(1)

    time.sleep(1)
```

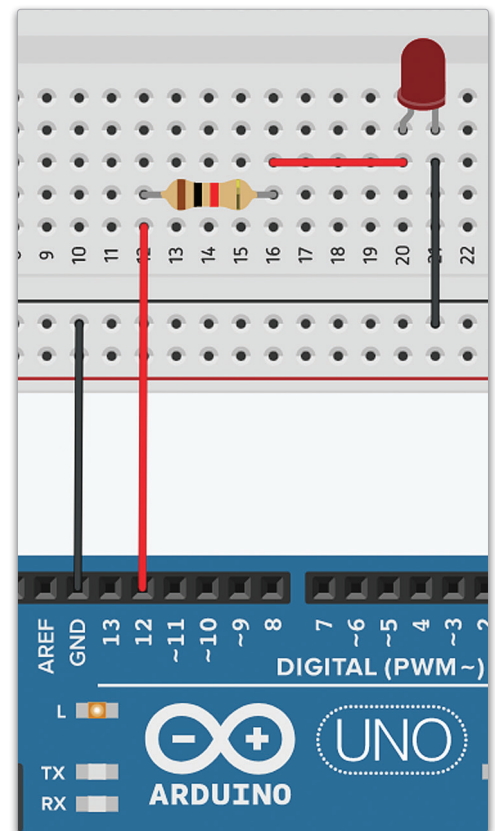
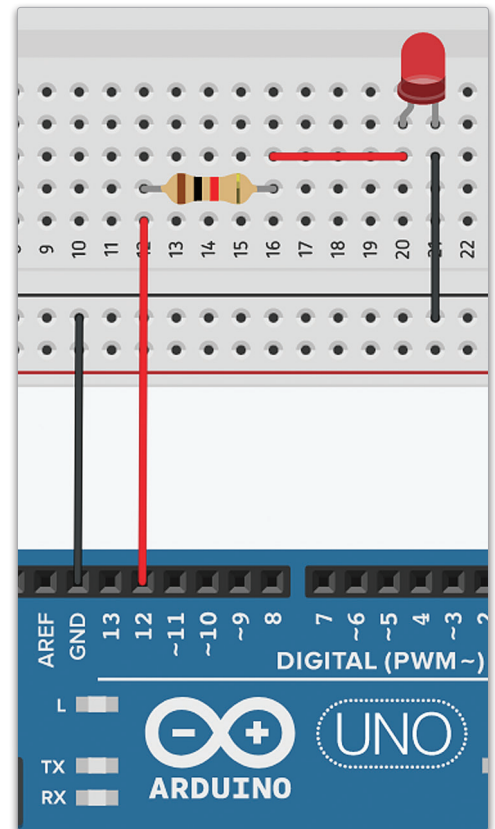


Figure 4.16: LEDs blinking interchangeably



Exercises

1

Read the sentences and tick ✓ True or False.	True	False
1. An Arduino microcontroller can be programmed only with C++.	☐	☐
2. Firmata is a cryptography protocol.	☐	☐
3. PyFirmata is a library used only with Python.	☐	☐
4. To open communication between an Arduino and a PC, you need to upload the ServoFirmata sketch.	☐	☐
5. The PyFirmata code automatically recognizes the communication port the Arduino uses.	☐	☐
6. Analog pins use Pulse Width Modulation instead of standard output.	☐	☐
7. External LED lights can emit light with both digital and analog signals.	☐	☐
8. The internal LED light of the Arduino is connected to the digital 10 pin.	☐	☐
9. The Python code with PyFirmata runs on the Arduino microcontroller.	☐	☐
10. The main benefit of using a real breadboard is that if the components are placed incorrectly then they can simply be moved to the correct location on the breadboard.	☐	☐

2 Describe the process of setting up a development environment for Arduino and mention what are the software and hardware environments.

3 What are the advantages of programming an Arduino with Python? Justify your answer.

4 What is the disadvantage of working with the Firmata protocol and the PyFirmata library?



5 What do the following lines of code do?

```
pin = board.get_pin("a:4:p")  
pin.write(0.75)
```

6 Give two examples of a sensor or an actuator that work more effectively with digital signals and two more examples of sensors or actuators which work better with analog signals. Consider the sensors and actuators you have been taught about.



Lesson 2

Programming Arduino with Python

Link to digital lesson



www.ien.edu.sa

Smart Garden with an Arduino

Due to climate change, smart gardening is promoted as an agricultural method that is both very sustainable and scalable. To cover the agricultural needs of an increasing population with climate change in mind, there needs to exist more efficient agricultural methods such as smart gardening. We will simulate an Arduino circuit that monitors a smart garden and sends data to an online cloud platform. The Arduino will constantly send data to the cloud and if a specific set of temperature and moisture conditions are met, we will simulate turning on the watering system. We will first simulate the circuit in Tinkercad Circuits to show clearly the circuit connections and then you will use the simulation to guide you to build the exact same circuit with a real Arduino microcontroller.

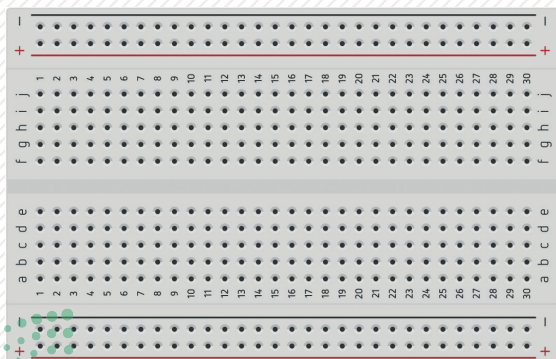
You will need the following components:

- 1 Arduino Uno R3
- 1 Breadboard Small
- 1 Soil Moisture Sensor
- 1 Temperature Sensor
- 1 DC Motor

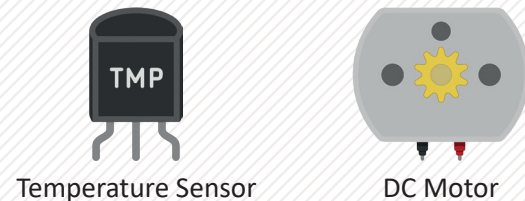
Cloud Platform

A cloud platform is a server in an Internet-based data center that makes it possible for software and hardware services to work together remotely and in large distributions.

Components that you will use in this project

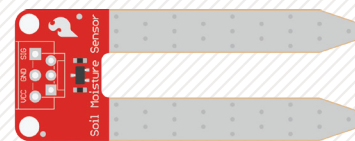


Breadboard Small

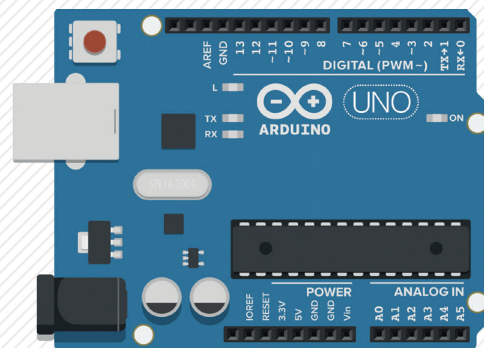


Temperature Sensor

DC Motor



Soil Moisture Sensor



Arduino Uno R3

Figure 4.17: Smart garden project components

To load the components:

- > Find the **Arduino Uno R3** from the **components library** and drag and drop it into the **workplane**. **1**
- > Find the **Breadboard Small** from the **components library** and drag and drop it into the **workplane**. **2**

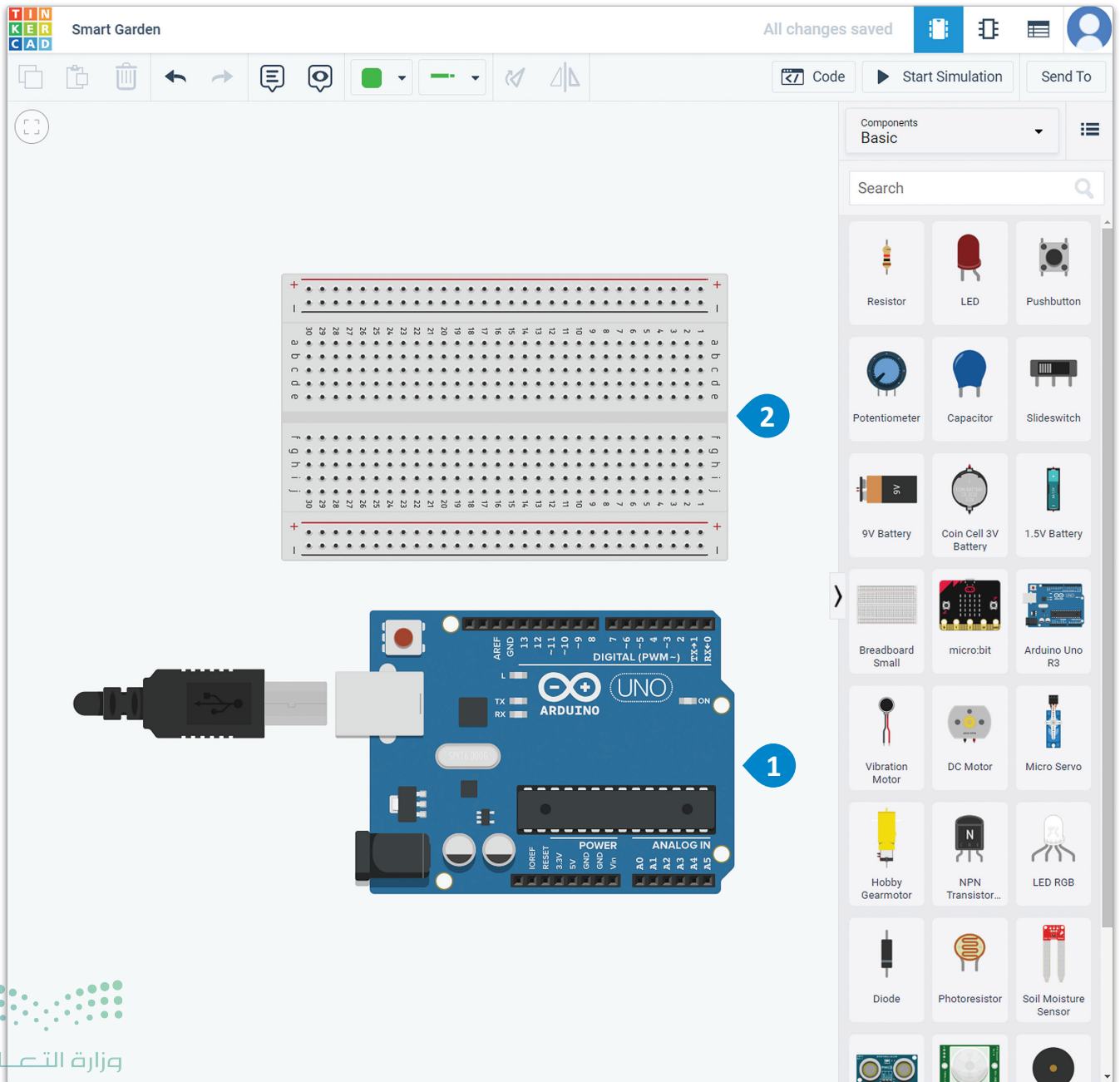


Figure 4.18: Loading the components

Now you will connect the Arduino to the breadboard.

To connect the Arduino to the Breadboard:

- > Connect the **Arduino Uno R3 GND** to the **negative column** of the **Breadboard Small** and change the wire color to **black**. **1**
- > Connect the **Arduino Uno R3 5V** to the **positive column** of the **Breadboard Small** and change the wire color to **red**. **2**

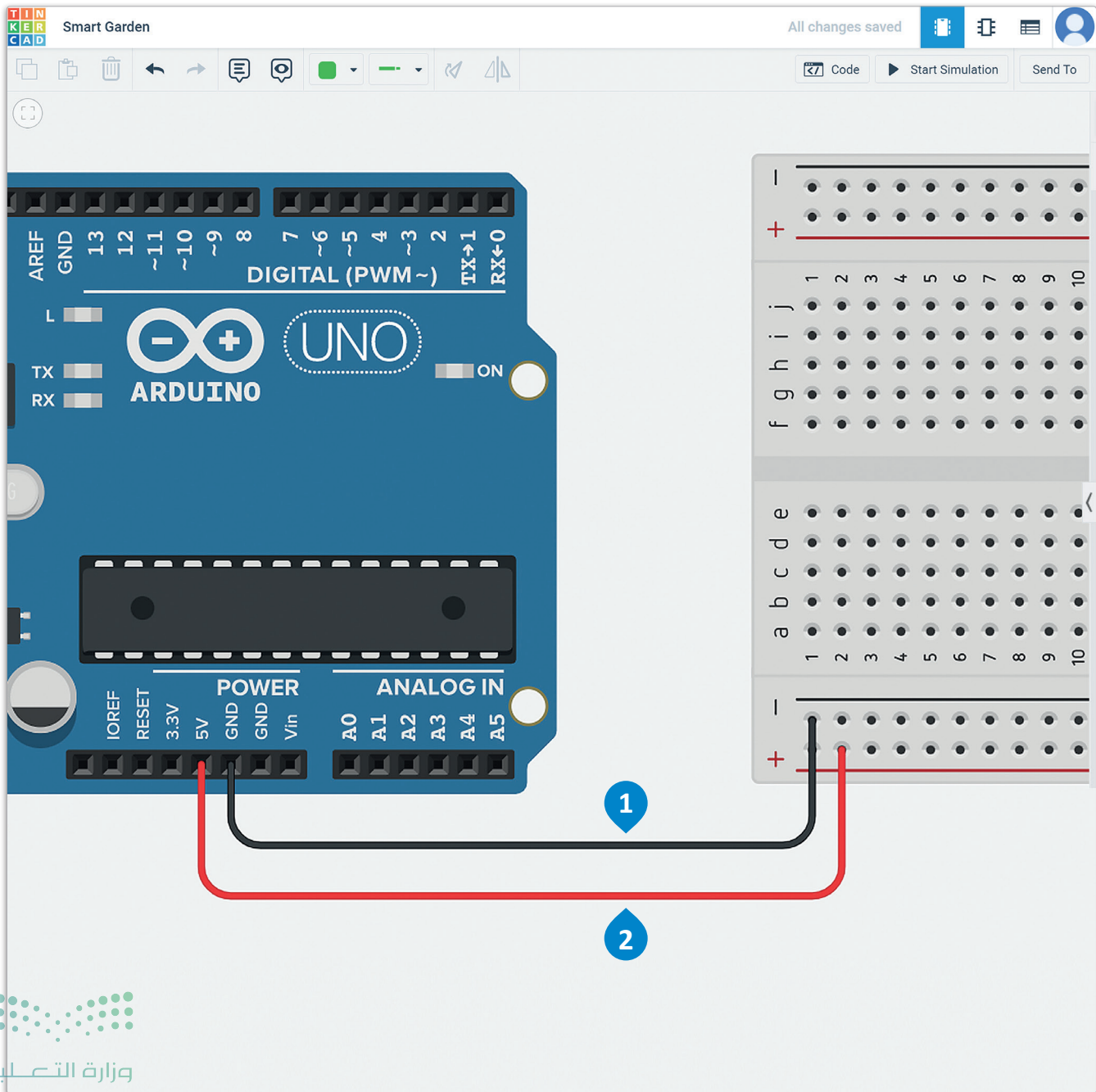


Figure 4.19: Connecting the Arduino to the Breadboard

Now you will connect the DC motor to a digital port in the Arduino.

To connect the DC motor:

- > Connect the **Arduino Uno R3 GND** to **Terminal 1** of the **DC motor** and change the wire color to **black**. **1**
- > Connect the **Digital pin 3** to the **Terminal 2** of the **DC motor** and change the wire color to **red**. **2**

The DC motor will emulate opening a watering system valve which would be activated when a specific set of conditions are met.

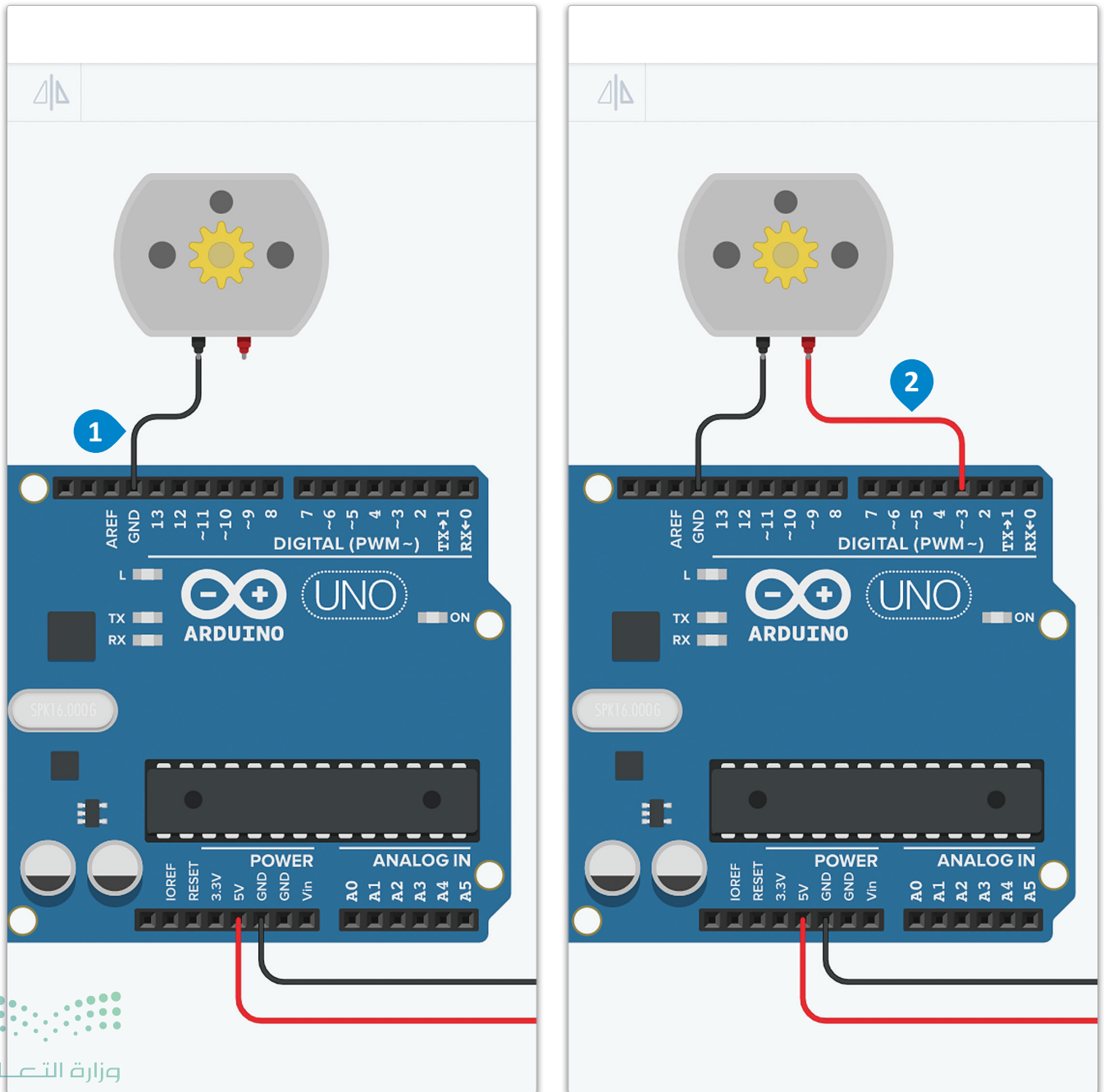


Figure 4.20: Connecting the DC motor

Now you will connect the Temperature sensor to an analog port in the Arduino.

To connect the Temperature sensor:

- > Connect the **Power** end of the **Temperature sensor** to the **positive column** of the **Breadboard Small** and change the wire color to **red**. **1**
- > Connect the **Vout** end of the **Temperature sensor** to the **Analog pin A2** of the Arduino and change the wire color to **green**. **2**
- > Connect the **GND** end of the **Temperature sensor** to the **negative column** of the **Breadboard Small** and change the wire color to **black**. **3**

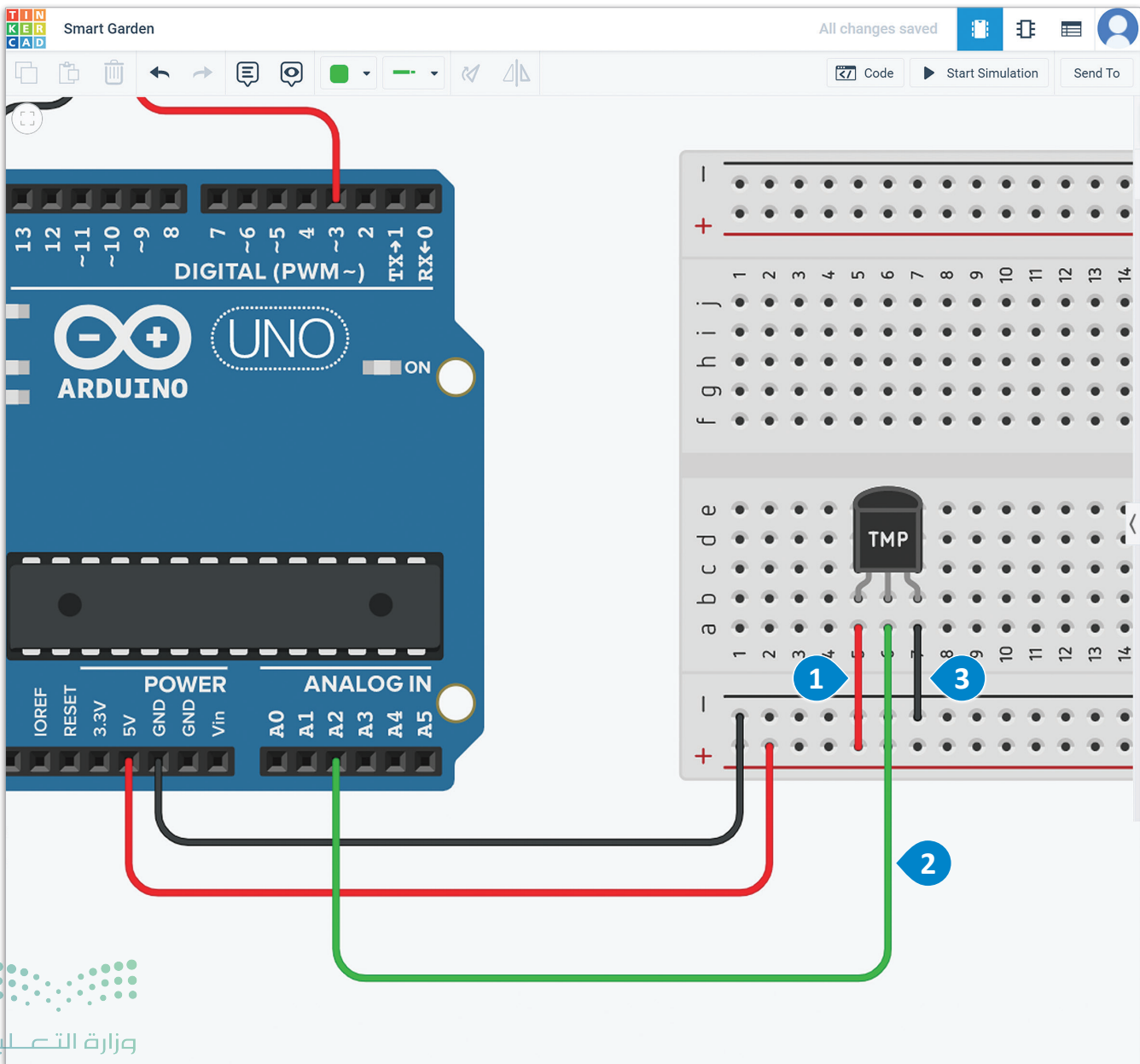


Figure 4.21: Connecting the Temperature sensor

Finally you will connect the Soil Moisture Sensor to an analog port in the Arduino.

To connect the Soil Moisture Sensor:

- > Connect the **Power** end of the **Soil Moisture Sensor** to the **positive column** of the **Breadboard Small** and change the wire color to **red**. **1**
- > Connect the **Ground** end of the **Soil Moisture Sensor** to the **negative column** of the **Breadboard Small** and change the wire color to **black**. **2**
- > Connect the **Signal** end of the **Soil Moisture Sensor** to the **Analog pin A4** of the Arduino and change the wire color to **green**. **3**

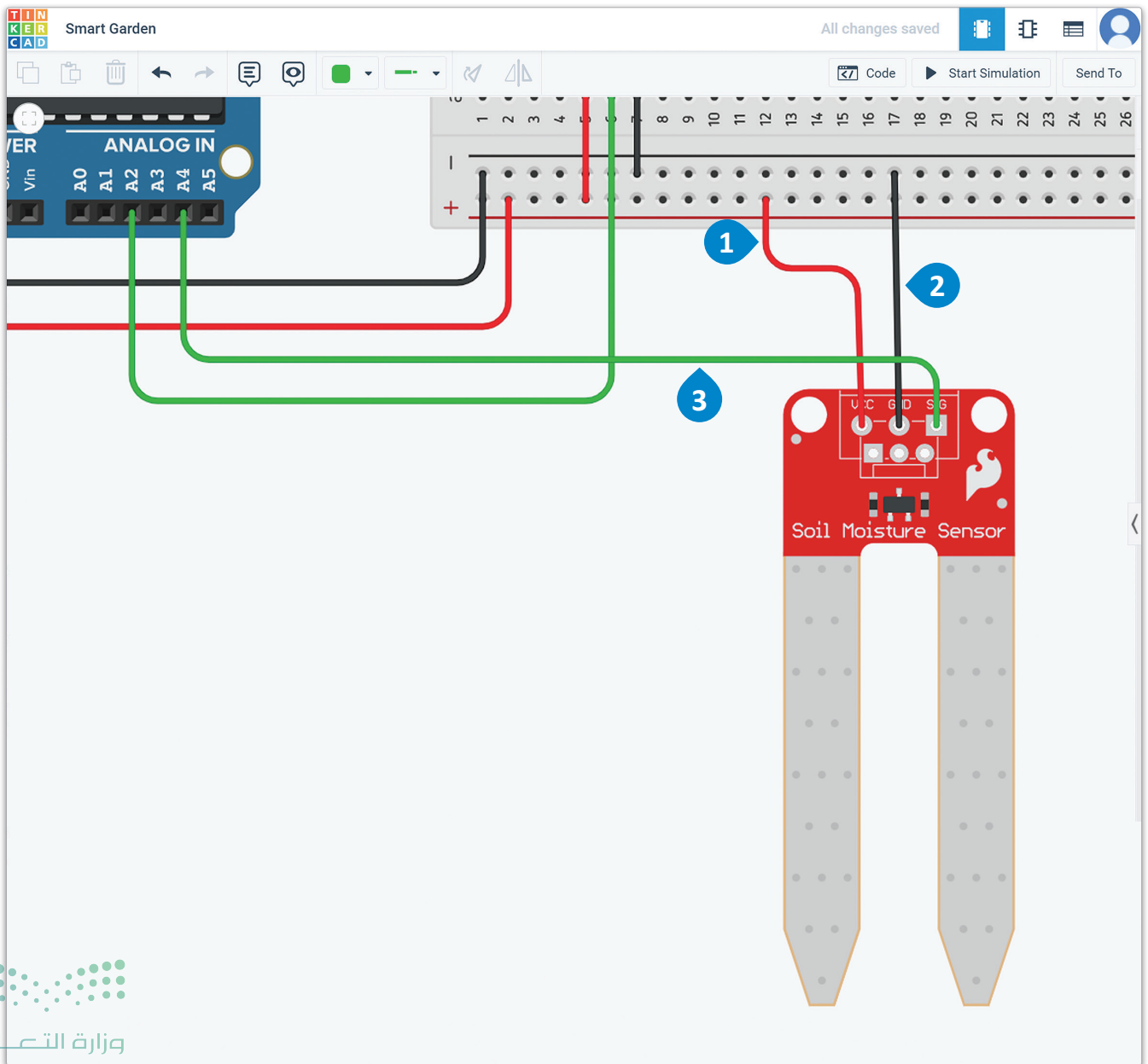
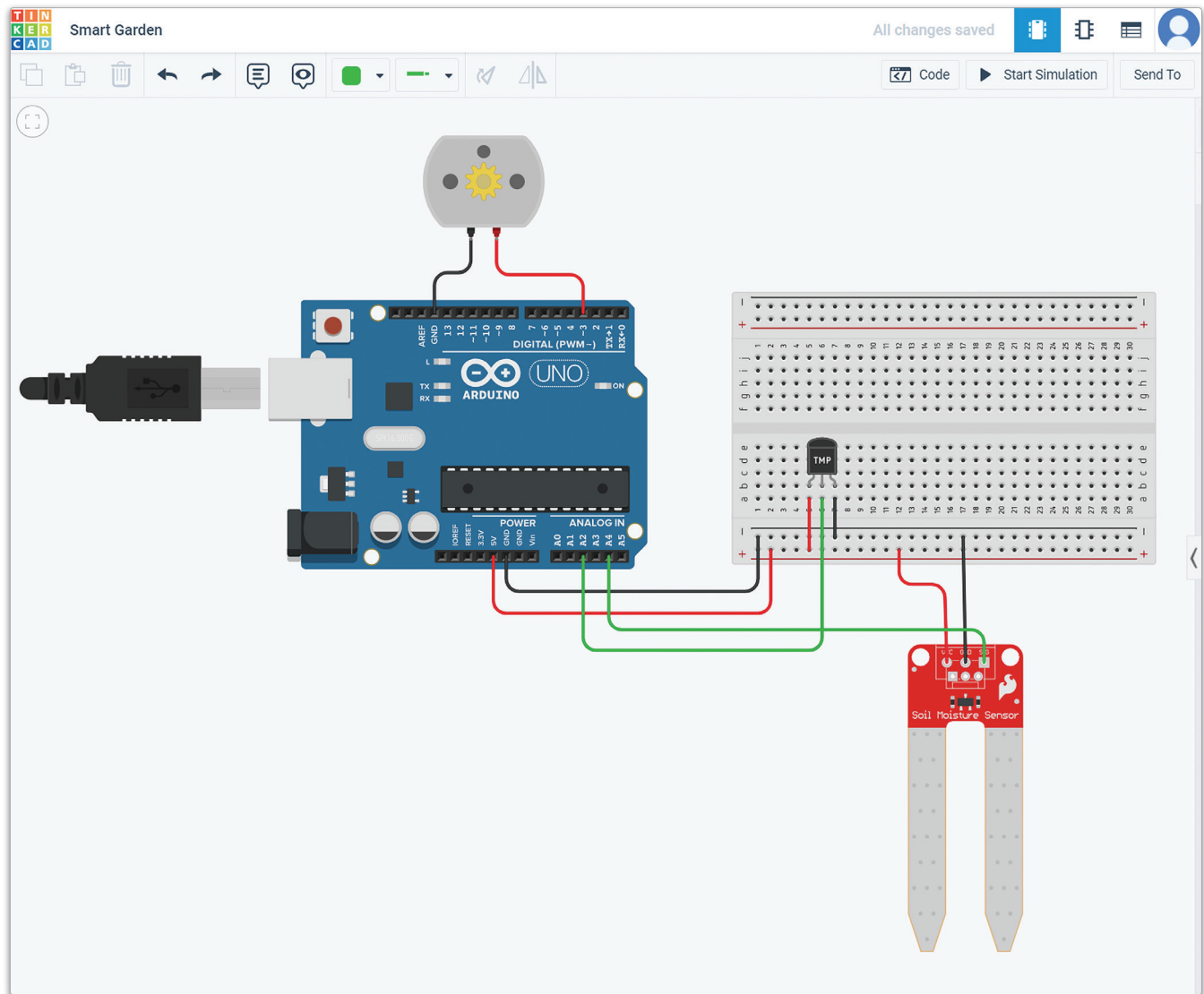


Figure 4.22: Connecting the Soil Moisture sensor

Complete Circuit



The components are connected to the following pins:

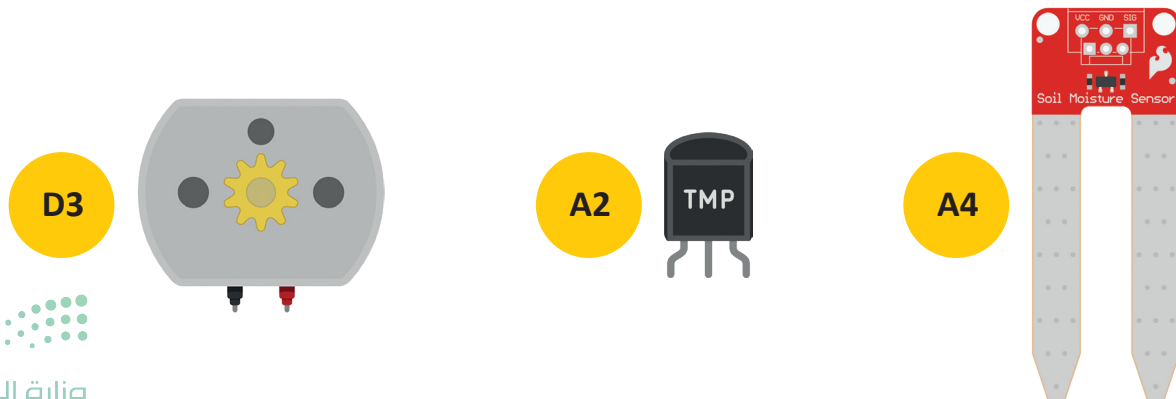


Figure 4.23: Pins connected to components

Physical Circuit

This photo represents what the physical circuit will look like.

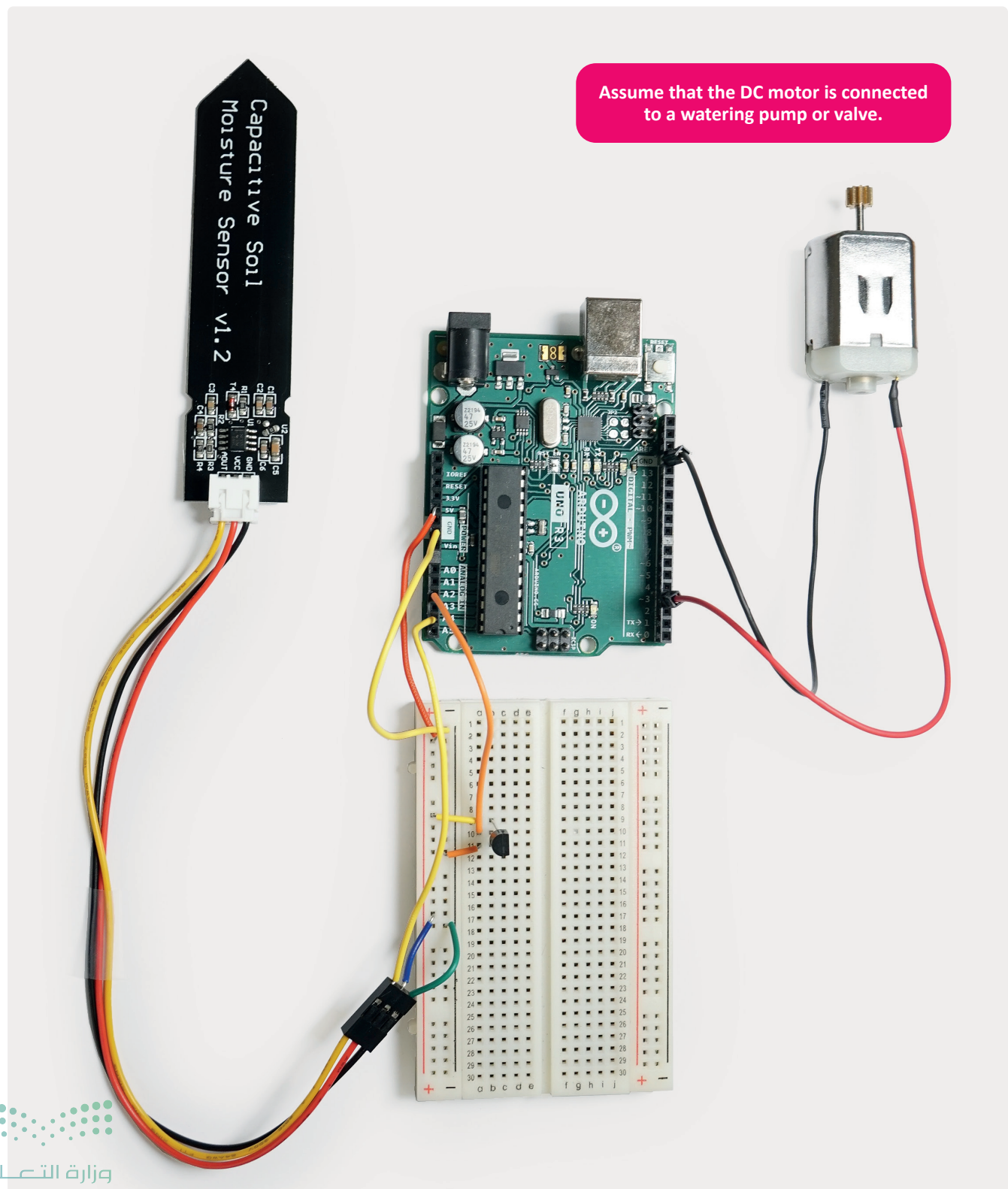


Figure 4.24: Photo of the physical circuit

Programming the Arduino Smart Garden Sensors and Motor

You will now program the Arduino to read the temperature and soil moisture sensor pins. If a specific set of temperature and moisture values are met, then the DC motor will be activated through a function, turning on for 5 seconds, then stopping, mimicking the automatic monitoring and watering of a plant in a smart garden.

Open **PyCharm**, create a new Python file and import the required libraries.

```
import pyfirmata
import time
```

Setup the communication port and the required pins.

```
communication_port = 'COM4'
dc_motor_pin = board.get_pin('d:3:o')
temperature_sensor_pin = board.get_pin('a:2:i')
moisture_sensor_pin = board.get_pin('a:4:i')
```

Setup the communication between PyFirmata and the board.

```
board = pyfirmata.Arduino(communication_port)
it = pyfirmata.util.Iterator(board)
it.start()
```

Implement the following method to control the DC motor.

```
def water_plant(dc_motor_pin):
    print("--- Watering plant ---")
    dc_motor_pin.write(1)
    time.sleep(5)
    dc_motor_pin.write(0)
```

This method sends a HIGH digital signal for 5 seconds to the DC motor and then sends a LOW digital signal to stop the rotation.



Create an infinite loop under which we will write our code.

```
while True:  
    # add your code here
```

Read the unprocessed temperature and moisture input values that you get from the analog pins.

```
temperature_value = temperature_sensor_pin.read()  
moisture_value = moisture_sensor_pin.read()
```

Check if the input values from the pins are null. The rest of the logic is implemented below this condition.

```
if (temperature_value is not None) and (moisture_value is not None):
```

Sometimes microcontroller sensors send null values, so we add another check so as to avoid errors in the program.

Create the following variables that map the unprocessed input values to the appropriate temperature and moisture values using mathematical formulas. For the temperature, we first get the 3 first decimal digits of the analog signal that comes from the temperature sensor and we then convert that value to the voltage that is applied by the temperature sensor on the signal pin. The next step is to convert the voltage to Celsius degrees which is a specific formula for this sensor type.

```
temperature_value = float(temperature_value) * 1000  
voltage = (temperature_value / 1024) * 5  
temperature = (voltage - 0.5) * 100  
moisture = (1.0 - float(moisture_value)) * 100
```

Converting the temperature to degrees Celsius and the moisture to a percentage.

Insert the temperature and moisture conditions for the watering of the plant.

```
if (temperature >= 24.0) and (moisture <= 40.0):  
    water_plant(dc_motor_pin)
```

If the temperature is hotter than 24 degrees Celsius and the soil moisture level is lower than 40%, activate the plant watering mechanism.

Create report messages to display through the terminal when the program runs and collects data.

```
temperature_report = "Temperature : " + str(temperature) + " C"  
moisture_report = "Moisture : " + str(round(moisture, 2)) + "%"  
  
print(temperature_report)  
print(moisture_report)
```

Complete Code

```
import time  
import pyfirmata  
  
board = pyfirmata.Arduino('COM4')  
it = pyfirmata.util.Iterator(board)  
it.start()  
  
dc_motor_pin = board.get_pin('d:3:o')
```



```

temperature_sensor_pin = board.get_pin('a:2:i')
moisture_sensor_pin = board.get_pin('a:4:i')

def water_plant(dc_motor_pin):
    print("--- Watering plant ---")
    dc_motor_pin.write(1)
    time.sleep(5)
    dc_motor_pin.write(0)

while True:
    temperature_value = temperature_sensor_pin.read()
    moisture_value = moisture_sensor_pin.read()

    if (temperature_value is not None) and (moisture_value is not None):

        temperature_value = float(temperature_value) * 1000
        voltage = (temperature_value / 1024) * 5
        temperature = (voltage - 0.5) * 100
        moisture = (1.0 - float(moisture_value)) * 100

        if (temperature >= 24.0) and (moisture <= 40.0):
            water_plant(dc_motor_pin)

        temperature_report = "Temperature : " + str(temperature) + " C"
        moisture_report = "Moisture : " + str(round(moisture, 2)) + "%"

        print(temperature_report)
        print(moisture_report)

        time.sleep(10)

```

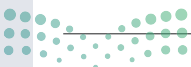


Exercises

- 1 Consider whether an analog output with Pulse Width Modulation would be a more efficient output to control a DC motor. Present your ideas below.

- 2 Considering the number of I/O pins of an Arduino and a micro:bit which one is a better microcontroller for a smart garden system. Present your ideas below.

- 3 Explain why the sensor input values taken from the analog pins have to be processed with a different mathematical formula depending on the sensor.



- 4 Describe why it is important to perform checks on the collected data to see whether the sensors send null or corrupt data.

- 5 Consider whether a servo motor would water the plants more precisely and effectively. Present your ideas below.

- 6 Expand the Python program with print commands to generate a report of the environment readings every 30 seconds.



Lesson 3

Interacting with a Cloud Web Service

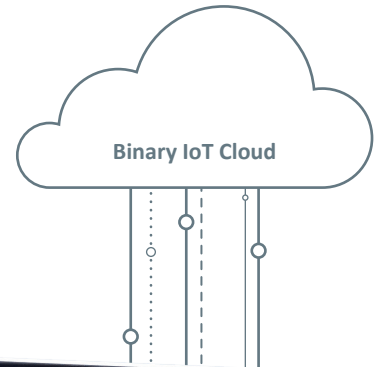
Link to digital lesson



www.ien.edu.sa

Interacting with Web Services

We will extend the project from the previous lesson to enable it to send data via a web service. Binary IoT Cloud is an educational online cloud platform that allows you to view the environmental data that is collected by the Arduino circuit you created in the previous lesson.



وزارة التعليم
Ministry of Education
2026 - 1448

Figure 4.25: Binary IoT platform with collected data from Arduino

Firstly, you will need to register and create a new user in the Binary IoT Cloud online platform.

To register on the Binary IoT Cloud online platform:

- > Go to the Binary IoT Cloud online platform at <https://ksa-iot.azurewebsites.net/Login.aspx> and on the welcome page click on **Register**. ①
- > Enter a **Username** ② and a **Password** ③ of your choosing and the **PIN** code **174563**. ④
- > Click the **Register** button. ⑤



binarylogic Binary IoT Cloud

Username

Password

LOGIN

① Register >

< Login

Username
sharif73 ②

Password
..... ③

Confirm Password
.....

Pin
..... ④

REGISTER ⑤

Figure 4.26: Registering in the Binary IoT Cloud online platform



Calling a Web API with Python

You will create a **JSON** object containing the environment data and send it to **Binary IoT Cloud** through a function that interacts with a **Web API**.

Install the Python **requests** package through pip. Open the terminal on your working directory and enter the following command:

```
pip install requests
```

At the start of your Python code, import the **requests** package with the line:

```
import requests
```

Get the current date and time and convert it to a string by importing the **datetime** module and calling the following method:

```
from datetime import datetime

date_time = str(datetime.now())
```

Create 2 variables **username** and **password** that are the credentials you used previously to register and login to **Binary IoT Cloud**.

```
username = "your_username_here"
password = "your_password_here"
```

Create a new function called **send_data()** which takes the following arguments:

```
def send_data(username, password, temperature, moisture):
```

Web API

A Web API (Application Programming Interface) is an endpoint that allows a program to access a service from another program that resides on a server on the Internet.

Javascript Object Notation (JSON)

JSON (JavaScript Object Notation) is an open data format type that is used to transfer data between services. Objects in JSON are key-value pairs that can store data types such as strings, integers, floats, arrays, and even other objects.



Figure 4.27: An application communicates with a cloud platform through a Web API



Create a new variable named `api_url` with the following link which is the cloud platform's API web endpoint:

```
api_url = "https://ksa-iot-api.azurewebsites.net/api/readings"
```

Create an object with the method variables which will be passed to the API as a JSON request body:

```
reading = {  
    "username": username,  
    "password": password,  
    "temperature": temperature,  
    "moisture": moisture,  
    "datetime": date_time  
}
```

Call the API with the **`requests.post()`** method which has as arguments an API URL and a JSON body object:

```
response = requests.post(api_url, json=reading)
```

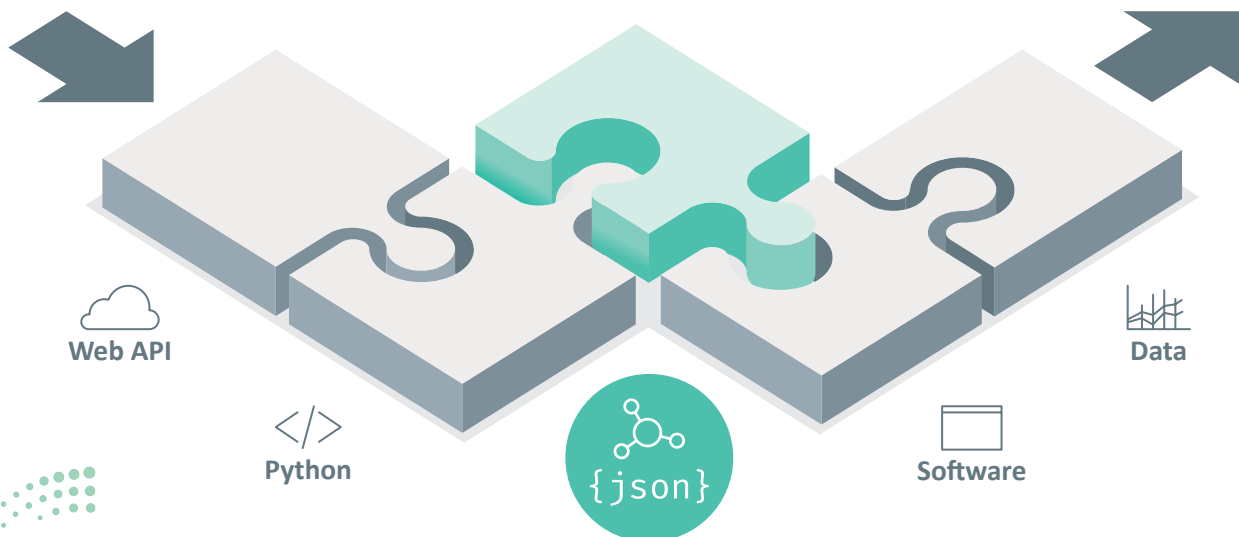


Figure 4.28: JSON is an open data format type to transfer data between services

The complete code for the **send_data()** method is shown bellow:

```
def send_data(username, password, temperature, moisture):  
    api_url = "https://ksa-iot-api.azurewebsites.net/api/readings"  
    date_time = str(datetime.now())  
  
    reading = {  
        "username": username,  
        "password": password,  
        "temperature": temperature,  
        "moisture": moisture,  
        "datetime": date_time  
    }  
    response = requests.post(api_url, json=reading)
```

In the main program, add the **send_data()** method to send the environment data you collected every 30 seconds.

```
send_data(username, password, temperature, moisture)  
time.sleep(30)
```

Complete Code

```
from datetime import datetime  
  
import time  
import requests  
import pyfirmata
```



```

board = pyfirmata.Arduino('COM4')
it = pyfirmata.util.Iterator(board)
it.start()

dc_motor_pin = board.get_pin('d:3:o')
temperature_sensor_pin = board.get_pin('a:2:i')
moisture_sensor_pin = board.get_pin('a:4:i')

username = "your_username_here"
password = "your_password_here"

def send_data(username, password, temperature, moisture):
    api_url = "https://ksa-iot-api.azurewebsites.net/api/readings"
    date_time = str(datetime.now())

    reading = {
        "username": username,
        "password": password,
        "temperature": temperature,
        "moisture": moisture,
        "datetime": date_time
    }

    response = requests.post(api_url, json=reading)

def water_plant(dc_motor_pin):
    print("--- Watering plant ---")
    dc_motor_pin.write(1)
    time.sleep(5)
    dc_motor_pin.write(0)

```



```

while True:
    temperature_value = temperature_sensor_pin.read()
    moisture_value = moisture_sensor_pin.read()

    if (temperature_value is not None) and (moisture_value is not None):

        temperature_value = float(temperature_value) * 1000
        voltage = (temperature_value / 1024) * 5
        temperature = (voltage - 0.5) * 100
        moisture = (1.0 - float(moisture_value)) * 100

        if (temperature >= 24.0) and (moisture <= 40.0):
            water_plant(dc_motor_pin)

        temperature_report = "Temperature : " + str(temperature) + " C"
        moisture_report = "Moisture : " + str(round(moisture, 2)) + "%"

        print(temperature_report)
        print(moisture_report)

        send_data(username, password, temperature, moisture)

        time.sleep(30)

```

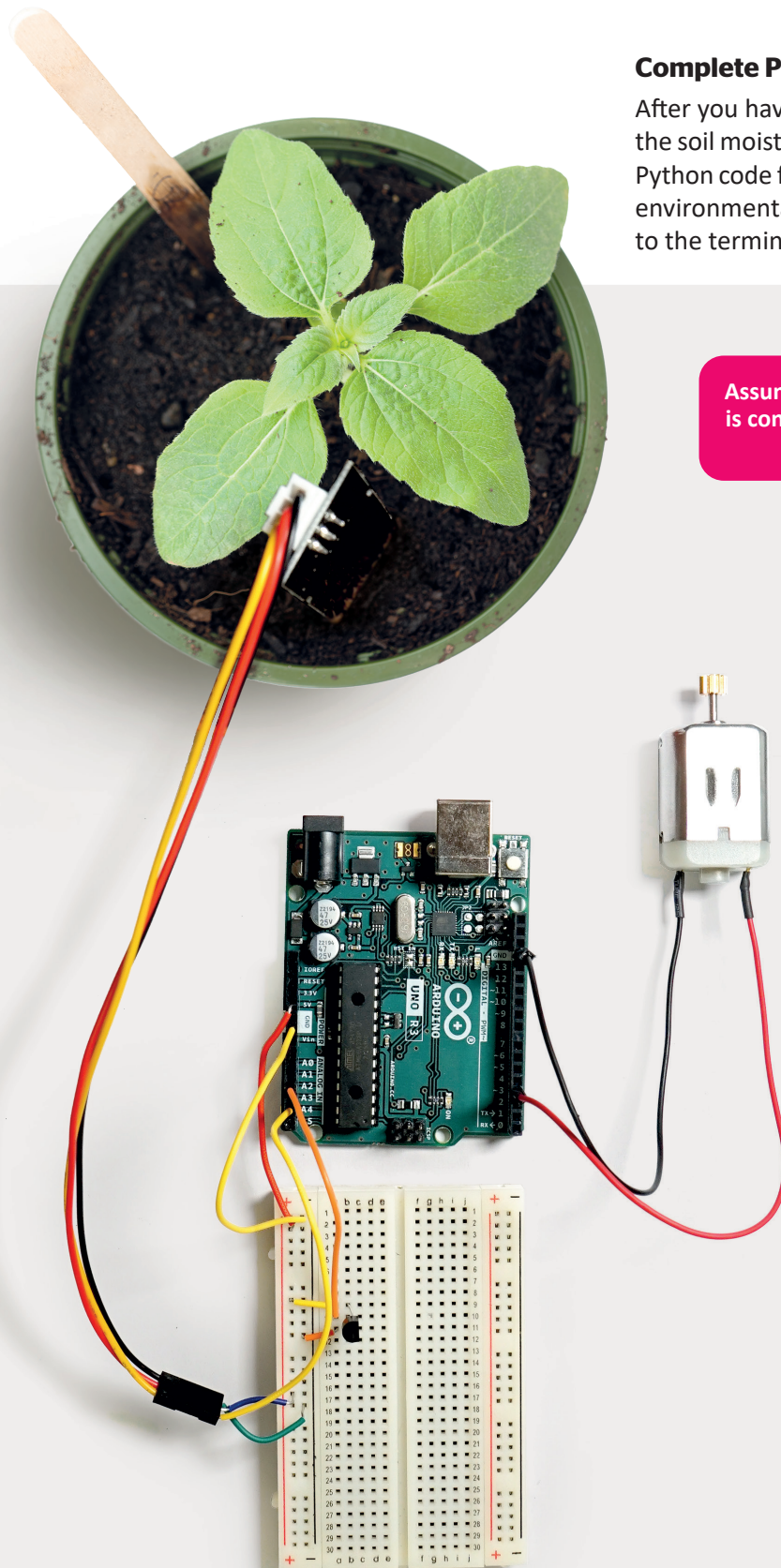


Figure 4.29: Arduino communicating with PC and Binary IoT Cloud

Complete Project

After you have written the program place the soil moisture sensor in a plant, run the Python code from your PC and you will see environmental readings being outputted to the terminal.

Assume that the DC motor is connected to a watering pump or valve.



Viewing the Smart Garden Data

Run the complete Python program and leave it for a few minutes to collect data. Then it will be uploaded to the Binary IoT Cloud platform.

Log in to the platform with your credentials and you can view the data that you have collected for your plant.

To log into the Binary IoT Cloud online platform:

- > Go to the Binary IoT Cloud online platform at <https://ksa-iot.azurewebsites.net/Login.aspx>. ¹
- > Enter the **Username** and the **Password** that you created. ²
- > Click the **Login** button. ³

The screenshot shows the login interface of the Binary IoT Cloud platform. At the top, there is a dark header with the 'binarylogic' logo on the left and 'Binary IoT Cloud' on the right. A blue circle with the number '1' points to the header area. Below the header, the login form is centered. It includes a 'Username' label above a text input field containing 'sharif73'. Below that is a 'Password' label above a text input field with masked characters '.....'. A blue circle with the number '2' points to the password field. Under the password field is a green 'LOGIN' button. Below the login button is a 'Register >' link. A blue circle with the number '3' points to the 'Register >' link. At the bottom left of the page, there is a logo for the Ministry of Education and the text 'وزارة التعليم' and 'Ministry of Education'.

Figure 4.31: Logging into the Binary IoT Cloud online platform

Welcome sharif73

The data that you view in this table are the environmental readings that are collected by the Arduino and sent to the platform via the Python program running on your PC.

Date ↑	Temperature	Moisture
15/08/2022 11:24	25.41	39.80
15/08/2022 11:24	25.42	39.40
15/08/2022 11:25	25.43	39.70
15/08/2022 11:25	25.41	39.60
15/08/2022 11:26	25.40	39.50
15/08/2022 11:26	25.40	39.43
15/08/2022 11:27	25.40	39.42
15/08/2022 11:27	25.41	39.41
15/08/2022 11:28	25.36	39.39
15/08/2022 11:28	25.40	42.34

Page 1 of 2 (16 items) < 1 2 >

EXPORT TO XLSX

EXPORT TO CSV

LOGOUT

Export your data into
.xlsx or .csv files

Comma Separated Values (CSV)

A CSV (Comma Separated Values) file is a text file whose values are separated by commas and every line in the file represents a data record.



Using Sensor Data and Big Datasets for Data-Driven Decision Making

The project we have just built implements basic functionality for monitoring a smart garden. However, this was just a small example with only one device collecting and sending data to an online platform. Suppose we had a large garden that required more devices to cover the various plants and environments, with each plant species and variety requiring a different environment configuration for its monitoring. Collecting different datasets would reveal patterns in the behavior of the garden as a whole system.

Now suppose that there were even more gardens in numerous locations across the city that the municipality parks management has to maintain with completely different systems. All these gardens collect vast amounts of data, which, when analyzed, can provide thorough insights and patterns about the operation and behavior of all those gardens compared to one another.

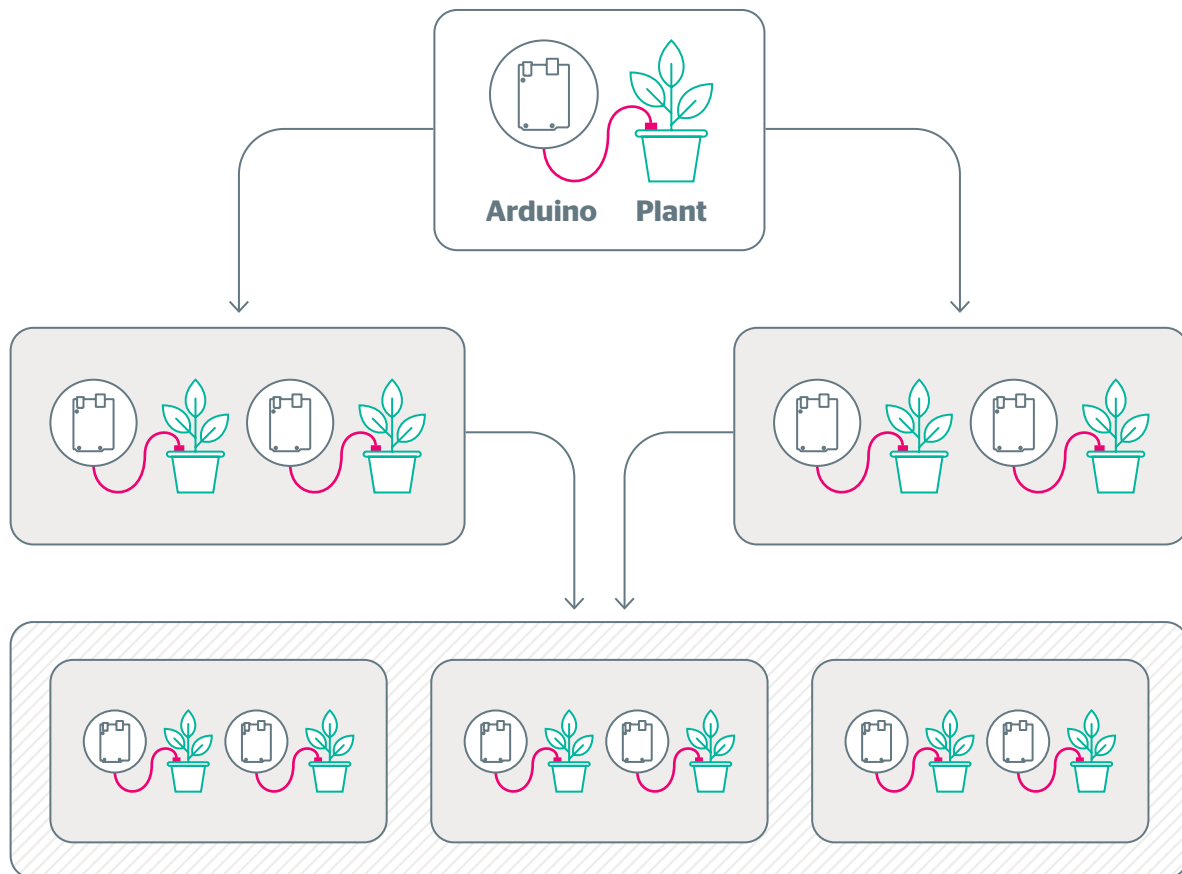


Figure 4.33: Scaling the smart garden project

Exercises

1 Explain the use of a Web API.

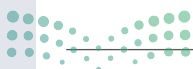
2 Why are the username and the password required to connect to the Web API?
Think of two reasons and write them down.



3 Which code lines do you have to change to modify the conditions required to start watering the plant and the duration of the watering?

This image shows a blank sheet of white paper with horizontal ruling lines. The lines are evenly spaced and run across the width of the page. There are no margins, text, or other markings on the paper.

4 Describe what readings other than temperature and humidity can be collected in a smart garden environment.



- 5 Suppose you could also read past data from an online cloud platform. You could not only have current environmental readings to use, but data from the past as well.

Analyze how this could lead to more efficient watering of your smart gardens and present your ideas below.

- 6 Compare the JSON and CSV data format types.



Project

Smart gardens are complex systems that accommodate various types of plants in the same environment, monitored by a single system.

1

Design an Arduino circuit in Tinkercad Circuits that builds upon the circuit you just created to monitor and water more plants.

2

Use the remaining analog pins of the Arduino to connect 3 more plants using soil moisture sensors.

3

Extend the Python code to get the moisture values from the new plants and create different environmental conditions for the watering of each plant.

4

Build the physical circuit and extend the Python code to accommodate the new plants.



Wrap up

Now you have learned to:

- > Install pyfirmata and use Python to program Arduino microcontrollers.
- > Build practical Internet of Things applications with a physical Arduino microcontroller.
- > Use cloud platforms to send and evaluate collected data.
- > Identify how data collected from Internet of Things solutions can affect the decision making process.
- > Understand how large Internet of Things solutions start from simple applications which are then expanded.

KEY TERMS

Comma Separated Values

Javascript Object Notation

Web API

Communication port

Protocol

Web service

Firmata Library

Pulse Width Modulation



Part 2

Unit 5

IoT Advanced Applications

Unit 6

IoT Programming with C++

Unit 7

IoT Messaging

Unit 8

IoT Wireless Sensor Network Simulation





5. IoT Advanced Applications

In this unit, you will learn about the implementation of IoT solutions in the healthcare industry as well as in the agricultural sector. Also, you will learn about IoT architectures, and you will explore different networking protocols. Finally, you will explore the concepts of security and privacy in IoT systems.

Learning Objectives

In this unit, you will learn to:

- > Describe how IoT technologies are used in the Internet of Healthcare Things (IoHT).
- > Identify different smart healthcare applications.
- > Describe how IoT technologies can improve the agriculture sector.
- > Classify the oneM2M IoT architecture layers.
- > Clarify the functionality of the IoT World Forum architecture layers.
- > Identify the main characteristics of NFC and RFID technologies.
- > Define the technologies and the protocols that are used in Wireless Personal Area Networks (WPANS).
- > Identify the 5G network security challenges in IoT systems.
- > Describe the IoT privacy concerns and their possible solutions.



Lesson 1

IoT Application Areas

Link to digital lesson



www.ien.edu.sa

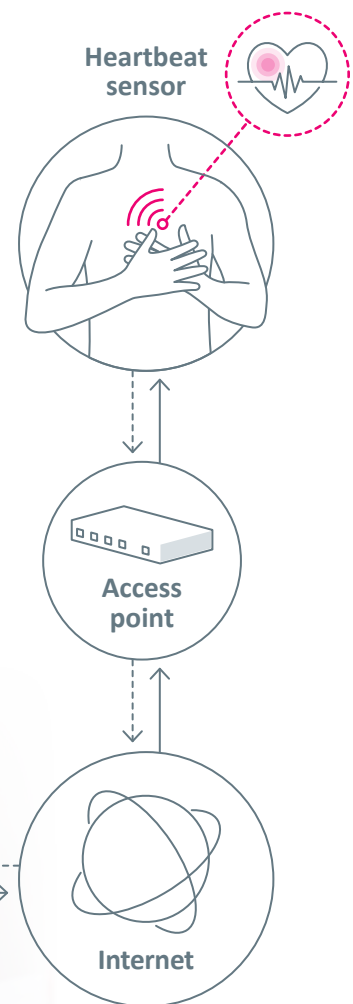
In the previous part (Part 1), you learned about the fundamental concepts behind the Internet of Things (IoT) and how it is an integral part of emerging technologies. You also learned some basic applications that use IoT technology. Now you will extend your previous knowledge by learning new IoT applications.

Smart Healthcare

The implementation of IoT in the healthcare industry has a significant impact on society. IoT devices, such as wearable sensors, provide remote health monitoring, emergency alerts, and human well-being systems. In addition to monitoring health metrics, health-tracking gadgets include innovative wearable technologies that enhance the quality of life. From the observation of pediatric patients to the diagnosis and monitoring of chronic illnesses in the elderly, effective healthcare services can be provided for all ages.

The Evolution of Healthcare

The rapid population rise creates new challenges that can be solved with smart healthcare. Smart healthcare refers to the application of technology to improve the quality of life. Due to the absence of digital-era knowledge among healthcare workers, the transition to smart healthcare is slow. However, governments and private institutions are investing in the integration of technologies to improve the healthcare system. Traditionally, a patient would visit a doctor, a local medical center, or a hospital when needed. Smart healthcare helps patients to handle certain emergency circumstances independently. The focus on individual healthcare has shifted from traditional hospital treatment to smart home care. With the use of IoT devices, smart healthcare delivers remote health monitoring, emergency alerts, cost-effective treatment, and the availability of medical services regardless of location. These health monitoring devices range from fitness trackers that measure health metrics to sophisticated wearable technologies that collect many metrics.



Emergency alert
monitoring

Figure 5.1: Monitoring health metrics

Internet of Healthcare Things

The Internet of Healthcare Things (IoHT) is an IoT-based solution that uses IoT technologies to link people with various healthcare services. Specialized physicians can remotely review medical reports and records and provide recommendations without being in the same location as the patient. IoHT consists of networked medical imaging, lab reports, and remote healthcare monitoring devices. Medical imaging could be X-ray, MRI (Magnetic resonance imaging) scan, computerized tomography (CT) scan, and other types of imaging. It also provides emergency services comparable to smart ambulances or smart clinics.

Wearables

Wearables are smart objects placed on the human body. Wearable medical devices can gather, store, process, and analyze data to provide the required feedback and send alerts in emergency scenarios. The primary users are patients with temporary or permanent disabilities, the elderly, and babies. Biosensors on the patient's clothing capture data and produce a digital electrical output that can be utilized to monitor their health indicators. A biosensor is a small analytical instrument combined with a biological component to recognize events. Sensors and actuators differ based on the monitoring systems. They can collect and transmit data, such as bio-signals, body temperature, oxygen saturation level (Pulse Oximetry), movements, and geographic location. There are many bio-signals generated from the body such as electrocardiogram (ECG), electroencephalogram (EEG), and Electromyography (EMG). Attached sensors can monitor physiological or biomechanical parameters, such as heart rate and muscle activity, respiration, body temperature, blood pressure, position, motion, and acceleration. The output of smart sensors and IoT devices is typically complex, necessitating the application of artificial intelligence, data analytics, and other technologies such as cloud computing.

Electroencephalogram (EEG)

An electroencephalogram (EEG) is a diagnostic tool to identify brain electrical activity abnormalities.

Body Sensor Network

A Body Sensor Network (BSN) is a Wireless Sensor Network (WSN) used for human body monitoring. It is a wearable sensor node network that can communicate with other nodes and smart objects. These sensor nodes have computing, storage, wireless transmission, and sensing capabilities. As shown in Figure 5.2, a blood flow sensor sends a patient's blood flow data to a smart device. This device is connected to the Internet and sends these data to the Smart Hospital. Even though BSN-based systems have a wide variety of applications, they can be used for continuous and non-invasive monitoring of vital signs, as tiny wireless sensors are placed on the skin and, in some cases, embedded in the clothing. This facilitates early disease identification and diagnosis. Typically, these sensors detect data on human body movement, body temperature, heart rate, skin conductance, and muscle functions.

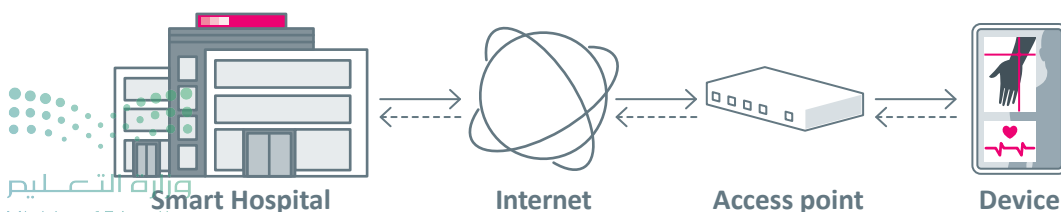
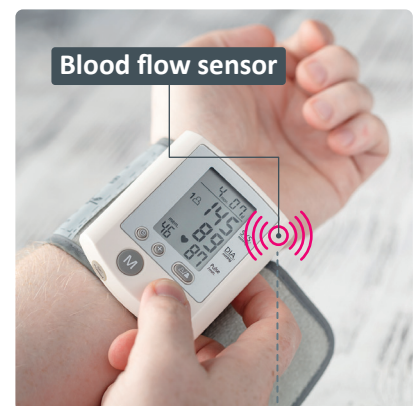


Figure 5.2: Body Sensor Networks connected to IoHT networks

Smart Healthcare Applications

Blood pressure monitoring

Variations in the typical rate at which the heart pumps blood are associated with high blood pressure in humans. Hypertension, another term for high blood pressure, is a worldwide health issue caused by elevated blood pressure in the arteries. Chronic hypertension causes many problems, including heart failure, chronic renal disease, and blindness. Smart watches are wearable IoT devices that besides, tracking a user's fitness and heart rate, can monitor other metrics like blood pressure and send the data for processing. IoT healthcare systems built on the cloud computing platform have become increasingly popular over time, allowing patients to monitor and control their blood pressure utilizing IoT devices.

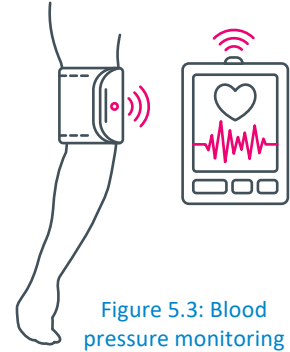


Figure 5.3: Blood pressure monitoring

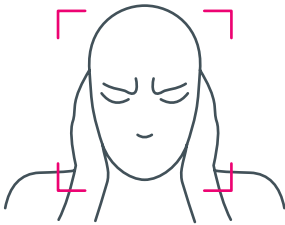


Figure 5.4: Pain monitoring

Pain monitoring

Identifying human emotions and pain is essential for delivering quality care to patients. Direct communication with patients or traditional means of interaction may not be adequate. Primarily youngsters, the elderly, and those with mental illness require this form of engagement. Expressions on the face are a behavioral indicator of pain. Since the feeling of pain generates changes in facial expressions, they can be utilized as an automatic technique for diagnosing human discomfort. Instead of standard self-reporting methods, it can be used for people who cannot self-report, such as intensive care unit patients and infants. Infants' facial expressions are frequently observed by their parents because they convey information about their health. A solution is an automated pain recognition system that uses physiological inputs from IoT sensors and data analysis to evaluate different kinds of pain and emotions.

Electrocardiogram monitoring

Sensors on the skin capture electrical signals caused by heartbeats. Electrodes are typically positioned on the chest when an ECG is used in clinics. However, this setup is not suitable for everyday use at home. There are various smart objects for remote ECG examinations, and hospital doctors can process patient data from these wearable devices. Such an application can be built as a warning system to offer people cardiac health alerts and recommendations.

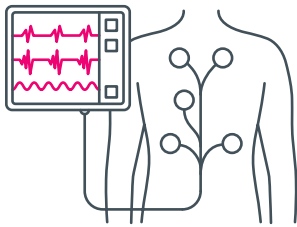


Figure 5.5: Electrocardiogram monitoring

Electrocardiogram (ECG)

An electrocardiogram (ECG) is a test that measures the heart's electrical activity to determine whether the heart is functioning appropriately.

Sleep monitoring

Sleep is a natural and periodic state of mental and physical rest, but many individuals suffer from sleep disorders. There are various sleep disorders, including insomnia, sleep apnea, and obstructive sleep apnea. Obstructive Sleep Apnea (OSA) is a potentially fatal respiratory disease during sleep. It impairs quality of life by producing personality and behavioral issues. Countless systems are available for detecting OSA. One solution is wearable in-ear electroencephalography (ear-EEG) connected to the room's IoT network. This is a continuous and unobtrusive method of 24/7 sleep monitoring that assesses sleep quality. The captured data is used to predict the sleep stages utilizing AI algorithms.

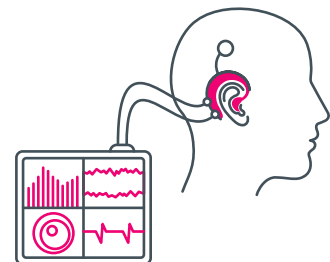


Figure 5.6: Sleep monitoring



Figure 5.7: Pathology monitoring

Pathology monitoring

Pathology is the scientific study of the origins and effects of disease and injury. In an EEG, this is accomplished by attaching small metal disks with thin wires to the scalp which send signals to a computer to store the results. EEG is frequently employed for this purpose because of its low cost and non-invasive nature. EEG can diagnose some brain-related disorders, such as epilepsy and stroke. Patients with these conditions require immediate attention because any delay can be fatal. An IoT system that monitors the patient's condition can be life-saving in such situations.

Disabled persons monitoring

Smart wheelchairs (SMW) connected to IoT systems is a new research topic. The design of these systems consists of two elements: a mapping service used for navigation and a client wheelchair. SMWs incorporate 3D LIDAR for mapping their external environs and autonomous movement without Global Positioning System (GPS). This technology employs both a control architecture for the motorized wheelchair and an embedded system for monitoring critically ill patients. The embedded system also uses the user's biometric characteristics to detect potentially dangerous situations. The wheelchair would generate a warning by activating the alarm upon measuring the heartbeat and blood pressure spikes at a certain interval.

Light Detection And Ranging (LIDAR)

Light Detection And Ranging (LIDAR) is a technique for measuring distances by pointing a laser at an item or surface and measuring the time required for the reflected light to return to the sender.

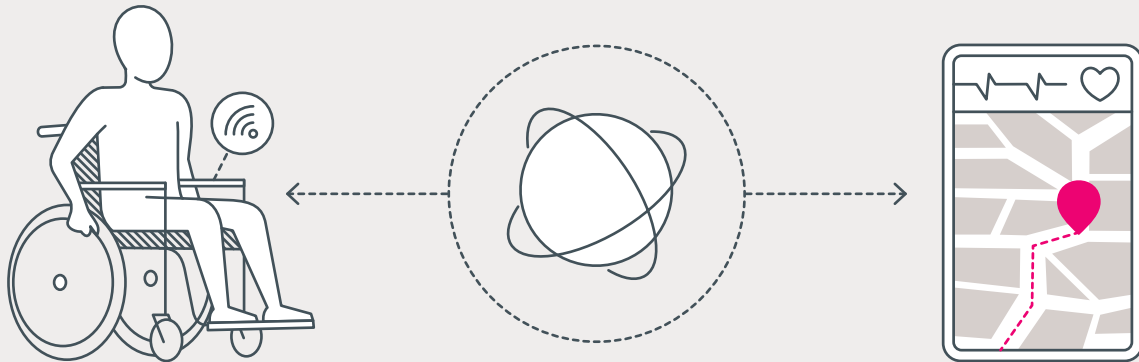


Figure 5.8: Disabled persons monitoring

Example

A Saudi telecommunication provider has launched the Virtual Clinic. It is used by doctors to make remote diagnoses for their patients. These services use IoT networking systems through wearables to help doctors collect the necessary data which is distributed to local hospitals and medical centers.

Smart Agriculture

In the previous unit, you took your first steps in smart agriculture by building a Plant Watering System. The agricultural sector can improve and optimize most workflows by utilizing many IoT technologies. The implementation of the IoT in today's agricultural sector has particular advantages, such as the efficient use of resources like land, water, fertilizers, and pesticides; an improvement in profitability, sustainability, food safety, and environmental protection; and a decrease in production costs.



Figure 5.9: Smart Agriculture with UAVs

Smart Agriculture Applications

Precision farming

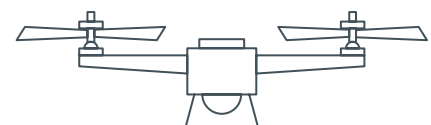
Precision farming refers to watering plants per their location and water amount requirements. This type of farming requires data from many sensors, such as plant location, humidity, and surface temperatures which may be obtained largely by aerial monitoring.

Remote-controlled aircraft, often known as Unmanned Aerial Vehicles (UAVs) or drones, have gained popularity for aerial monitoring. Over the past years, UAVs have been utilized extensively for tracking cultivated fields and providing effective precision agriculture solutions. Using remote sensing, it is possible to follow a variety of crop and vegetation metrics using images of varying wavelengths. Historically, remote sensing relied heavily on satellite imagery. UAV systems have proven effective in various precision farming applications, including pesticide, water deficit recognition, and disease identification. Numerous decisions can be made based on the data captured by the UAV for estimating yield in order to fix the identified problem and maximize production.

The role of UAVs is to capture data with precise spatial details. Many sensors are used depending on the agricultural parameters that must be monitored. UAV sensors must meet three essential requirements: low energy consumption, light weight, and small size. These techniques create environment maps that depict the soil morphology, allowing for more efficient irrigation planning for each crop. Global Positioning System (GPS) technologies are widely utilized to assist in the localization and georeferencing of objects captured by remote sensing. Since remote sensing information is a rich source of environmental data, it is usually imported into Geographic Information Systems (GISs) and combined with other datasets.

Unmanned Aerial Vehicle (UAV)

Unmanned Aerial Vehicles (UAVs) are aircrafts without human pilots, crew, or passengers.



Low energy **Light weight** **Small size**

Figure 5.10: UAV essential requirements

Table 5.1: Important types of sensors used by UAVs

Sensor Type	Description
 Visible Light Sensors	It can take images in various conditions, including sunny and cloudy weather. However, the quality of the photos depends on light conditions.
 Thermal Infrared Sensors	Infrared thermal sensors measure surface temperatures. Using infrared sensors and an optical lens, thermal cameras collect infrared energy. Thermal imaging cameras focus and detect radiation at the same wavelengths, transforming it into grayscale images representing heat. Multiple thermal imaging sensors can create a colored image.
 Multispectral Imaging Sensors	Multispectral sensors collect visible wavelengths as well as wavelengths that fall outside the visible spectrum, including near-infrared radiation (NIR), short-wave infrared radiation (SWIR) and others. UAVs with multispectral or hyperspectral sensors collect crop absorption of water information. Despite their increased cost, spectral data can be quite valuable for evaluating many biological and physical characteristics of crops.

Precision irrigation

Precision irrigation is a micro-irrigation technique that conserves nutrients and optimizes water required by plants. It slowly provides plant roots with water droplets below or above the surface. Crop productivity is increased by adopting precision irrigation IoT technologies. The installed sensors identify or read the physical and chemical aspects of the farmland, including the weather, temperature, humidity, plant health, soil moisture, soil acidity, and soil nutrients. The collected data are analyzed to inform farmers of the necessary adjustments. Data analysis assists in determining the appropriate nutrients and their quantities, as well as the water needed for irrigation.



Figure 5.11: Precision Irrigation application

Vertical farming

Vertical farming is the cultivation of plants at a vertical scale, not a horizontal one. Only a small area is needed for a crop to thrive, and multiple types of crops can be cultivated concurrently. Using IoT technologies, devices may be remotely handled using communication technologies such as Bluetooth, Wi-Fi, and RFID.

Vertical farming is typically meant to cultivate crops in urban environments. An indoor vertical farming system has a perfect climate and no external environmental concerns. IoT technologies are crucial for the farming environment and plant health monitoring and watering.

Vertical farming necessitates the processing and analysis of vast amounts of data for crops to develop effectively. With vertical farming, agricultural productivity can be optimized with technical assistance, such as automating the entire process from seed to harvest in an enclosed environment.



Figure 5.12: Vertical farming application

Example

The NEOM mega-city project in KSA is planned to be a vertical city that will utilize breakthrough technologies to solve the problems of pollution, transportation, and food sustainability. It will consist of two structures that are 500 meters tall, built 200 meters apart, and run in parallel for 170 kilometers. The area in between will host the advanced vertical city. NEOM aims to create the first integrated desert food self-sufficiency system. With scarce water availability, smart agriculture systems are needed to create self-sufficient communities. Circular agriculture and vertical farming techniques are enhanced by IoT and AI technologies to optimize the use of resources and enhance agricultural production. You can learn more about the Line in NEOM from here: <https://www.neom.com/ar-sa/about>

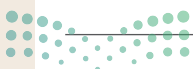
Exercises

1

Read the sentences and tick ✓ True or False.	True	False
1. IoT technologies have not enhanced the healthcare industry.	☐	☐
2. The Internet of Healthcare Things is an extension of the Internet of Things.	☐	☐
3. All wearable medical devices are constantly connected to the Internet.	☐	☐
4. Body sensor networks can be autonomous IoT systems.	☐	☐
5. A smart wheelchair includes an embedded system which uses the user's biometric characteristics to detect potentially dangerous situations.	☐	☐
6. UAVs can perform only one type of scan across an agricultural area.	☐	☐
7. Thermal Infrared Sensors detect radiation of radiant heat.	☐	☐
8. Precision irrigation is used to optimize the usage of the resources needed for an agricultural system.	☐	☐
9. Precision irrigation does not need a lot of sensors to be effective.	☐	☐
10. Vertical farming is used to optimize area usage.	☐	☐

2

Define what the Internet of Healthcare Things is.



3 Distinguish what data types can be collected by wearable smart objects.

4 Describe what a Body Sensor Network is comprised of.

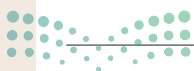
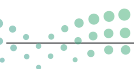
5 Analyze how AI solutions can be used for IoHT solutions for pain monitoring.



6 Describe how UAVs are used for precision farming IoT solutions.

This image shows a blank sheet of white paper with horizontal ruling lines. The lines are evenly spaced and run across the width of the page. There are no margins, text, or other markings on the paper.

7 Classify the various types of UAV sensors.



8

This image shows a single sheet of white paper with horizontal blue or grey ruling lines. The lines are evenly spaced and run across the width of the page. There are approximately 20 lines visible. The paper has a slight shadow on the right side, suggesting it's resting on a surface.

9



Lesson 2

IoT Networking Technologies

Link to digital lesson



www.ien.edu.sa

OneM2M Architecture Versus IoT World Forum Architecture

The rapid development of machine-to-machine (M2M) communications has resulted in the creation of IoT architectures. These architectures help accelerate the adoption of M2M applications and devices including the Internet of Things. The **oneM2M** architecture and the **IoT World Forum Architecture** are considered widely known IoT architectures. The **oneM2M** architecture designs IoT solutions with only the devices and their applications in mind. The **IoT World Forum Architecture** is used to design IoT applications while considering technologies such as data storage, data processing, network connectivity, and edge computing.

OneM2M Architecture

Dealing with the variety of devices, software, and access methods is one of the biggest issues when developing an IoT architecture. By creating a horizontal platform design, oneM2M architecture is building interoperability standards at all levels of the Internet of Things stack.

Based on the oneM2M architecture the IoT functions are separated into three layers: the application layer, the services layer, and the network layer. At first look, this architecture may appear basic and relatively generic; however, it is very rich, encourages interoperability via IT-friendly APIs, and supports a vast array of IoT technologies.

Applications layer

The oneM2M architecture prioritizes connections between devices and their respective applications. This domain contains application-layer protocols and integration with business intelligence (BI) systems.

Services layer

This layer is represented as a horizontal structure across industry-specific apps. Horizontal modules at this tier comprise the physical network on which IoT apps operate, the underlying management protocols, and the hardware. Examples include cellular backhaul communications, Multiprotocol Label Switching (MPLS) networks, Virtual Private Networks (VPNs), Software Defined Networks (SDNs), etc. The topmost layer is the common services layer.

Machine-to-Machine (M2M)

Machine-to-Machine, or M2M, is a term that describes any technology that enables networked devices to exchange data and carry out tasks without human intervention.



Software-Defined Networks (SDN)

Software-Defined Network (SDN) is a network architecture where the network is controlled through software-based controllers or Application Programming Interfaces (APIs) instead of specialized hardware devices.

Multiprotocol Label Switching (MPLS)

Multiprotocol Label Switching directs data between nodes based on specified labels and tags, not network addresses.

Network layer

This is the IoT devices and endpoints' communication domain. It consists of both the devices and the communications network that connects different types of networks like wireless mesh networks and point-to-multipoint systems.

Point-to-Multipoint System

A Point-to-multipoint system provides various pathways from a single network node to multiple destination nodes.

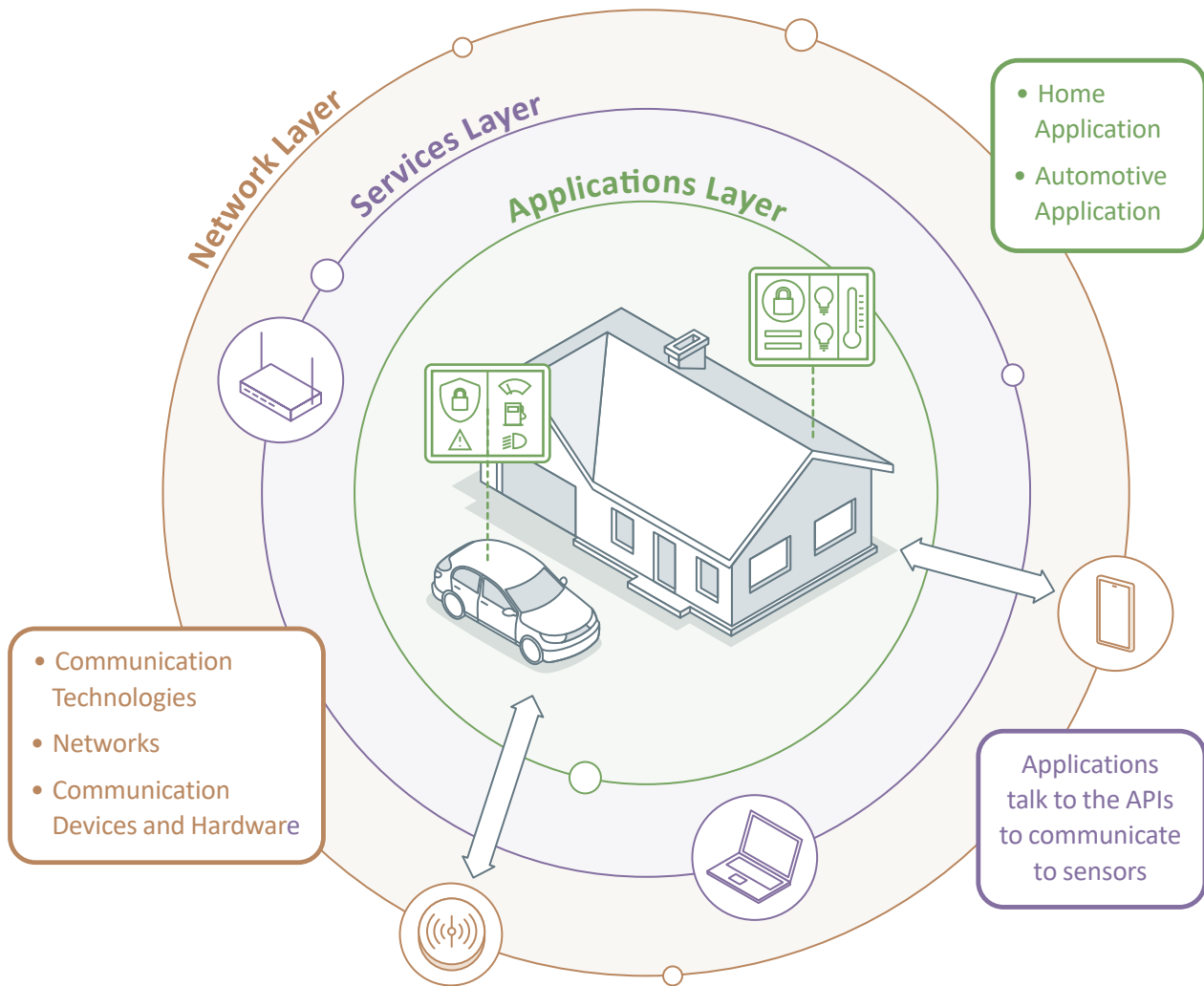


Figure 5.13: oneM2M architecture layers

Smart and non-smart gadgets frequently communicate with one another. In other cases, machine-to-machine communication is unnecessary, and devices merely connect with usecase-specific apps in the IoT application domain across a Field Area Network (FAN). The FAN is the most complex component of the communications network since it is primarily responsible for providing "last-mile" communications to end devices. The device domain also consists of the gateway device, which provides connections to the core network and serves as the boundary between the device and network domains.

IoT World Forum Architecture

The IoT Reference Model introduced at the IoT World Forum specifies a series of levels with control flowing from a center point to edge layers, which consists of sensors, devices, machines, and other intelligent end nodes. In general, data moves from the edge layers of the stack to the center.

By utilizing this reference model, we may accomplish the following:

- Divide the IoT challenge into subproblems.
- Determine the various technologies at each layer and their interrelationships.
- Define a system whose components can be supplied by several vendors.
- Define interfaces in a manner that promotes interoperability.
- Define a layered security paradigm that is enforced at level transition points

The IoT Reference Model is similar to the OSI Networking Model.

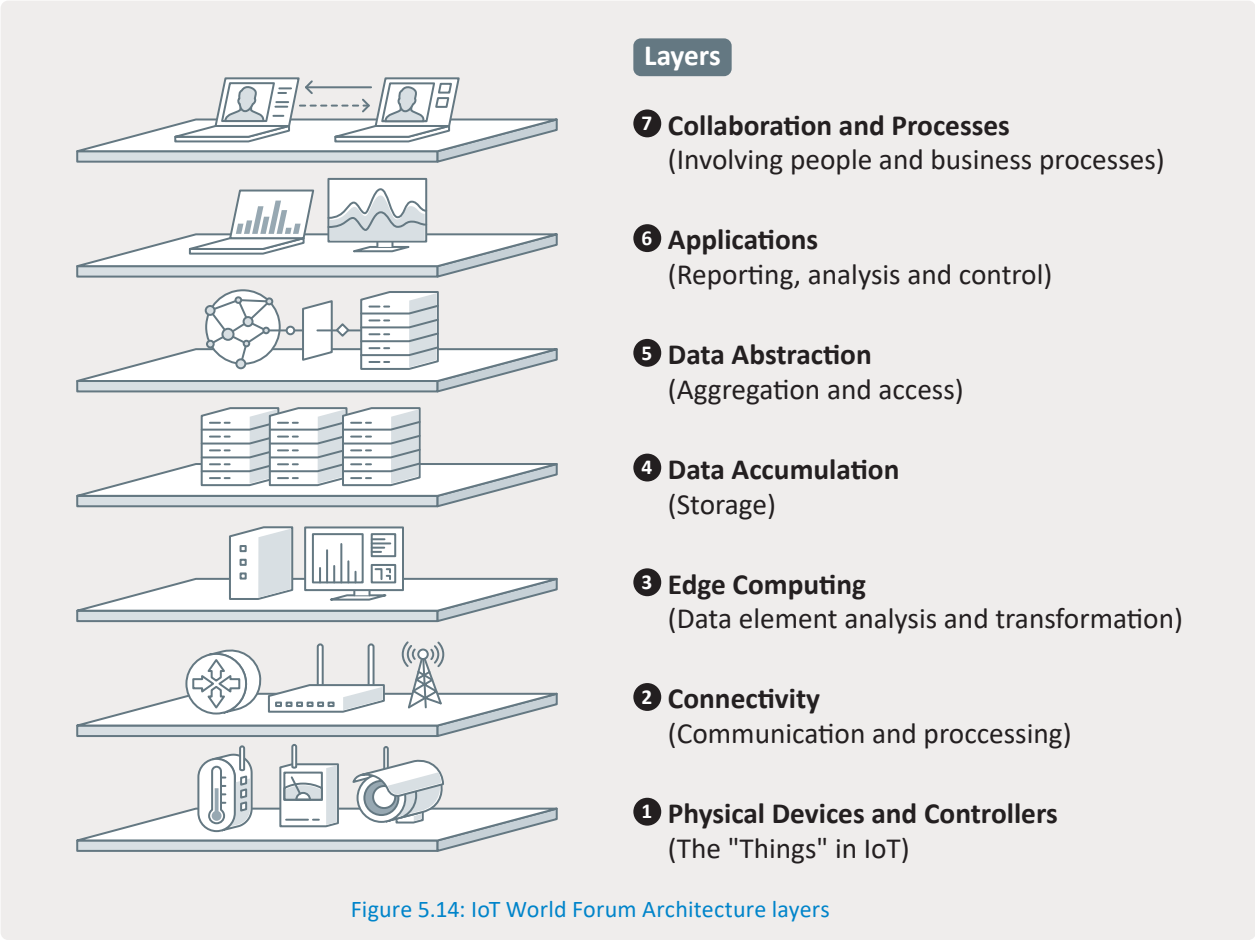


Figure 5.14: IoT World Forum Architecture layers

Layer 1: Physical Devices and Controllers Layer

The IoT Reference Model's initial layer is the physical devices and controllers layer. This layer contains the "things" of the Internet of Things, such as the many endpoint devices and sensors that send and receive data. These "things" can range in size from practically tiny sensors to enormous manufacturing machinery. Their main task is to generate data and allow control across a network.

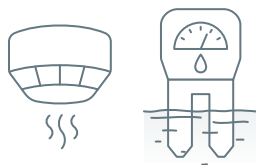


Figure 5.15: Devices & Controllers layer

Layer 2: Connectivity Layer

The role of the connectivity layer is the transfer of data in a reliable and timely manner. This covers transmissions between Layer 1 devices and the network, as well as transmissions between the network and Layer 3 information processing (the edge computing layer). As you may have seen, the connection layer comprises all networking parts of IoT and makes no distinction between the last-mile network (the network between the sensor/endpoint and the IoT gateway, addressed later in this chapter), the gateway network, and the backbone network.

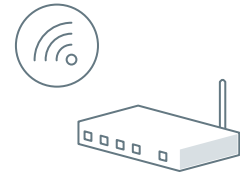


Figure 5.16: Connectivity layer

Layer 3: Edge Computing Layer

Edge computing is Layer 3's role. This layer focuses on data reduction and transforming network data flows into information that is ready to be stored and processed by higher levels. One of the fundamental ideas of this reference model is that information processing be initiated as close to the network's edge as is feasible and as quickly as is possible. Layer 3 also performs the examination of data to see whether it may be filtered or aggregated before being transferred to a higher layer. This also permits data to be reformatted or decoded, which facilitates further processing by other systems.

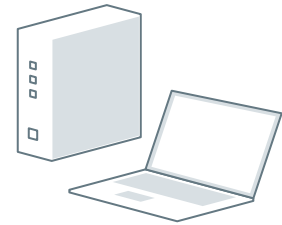


Figure 5.17: Edge Computing layer



Figure 5.18:
Data Accumulation layer



Figure 5.19:
Data Abstraction layer

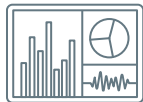


Figure 5.20:
Applications layer



Figure 5.21:
Collaboration & Processes layer

Layer 4: Data Accumulation Layer

Captures and saves data so that programs may access it when necessary. Converts event-based data to formats that can be queried by other services.

Layer 5: Data Abstraction Layer

Reconciles diverse data formats and ensures consistent semantics from varied sources. Using virtualization verifies that the data set is full and consolidates data into a single location or several data stores.

Layer 6: Applications Layer

Utilizes software programs to interpret data. Applications are able to monitor, regulate, and generate reports depending on the data analysis.

Layer 7: Collaboration and Processes Layer

Consumes and distributes application data. IoT's utility stems from the fact that sharing and collaborating on IoT data frequently involves numerous steps. This layer can alter company operations and offer IoT advantages.

Short Range Communication Network and Protocols

RFID and NFC

Radio-Frequency Identification (RFID) and Near-Field Communication (NFC) are communication technologies that enable the short-range connection between IoT devices and a network. RFID and NFC are used to store and retrieve data remotely. They consist of a radio transponder, radio receiver, and radio transmitter and use electromagnetic fields to automatically recognize and track tags attached to smart objects. Each tag sends or receives digital data when activated by an electromagnetic interrogation pulse from a nearby RFID or NFC reader device.

RFID enables the tracking of tools, equipment, inventories, assets, and people through tags that are attached to them. If a tag is close to a reader, it can be read even if it is not visible. These tags can be read in bulk, unlike barcodes which can only be read one at a time, and they can be read within a case, carton, box, or another container.

NFC is generally used to exchange data between devices within a range of around 10 centimeters. It is used for contactless credit card transactions, to replace digital office or hotel keys, and to simplify the connection and setup of devices such as headphones.

The main difference between RFID and NFC is that NFC is designed for secure data exchange, making it appropriate for financial transactions, while RFID is mainly used for applications where we need to identify unique items wirelessly.

Table 5.2: RFID vs NFC

Feature	RFID	NFC
Usage Frequency	125kHz ~ 2.45GHz	13.56MHz
Connection Range	Maximum 100m	Within 10cm (short distance)
Communication	One-way Communication	Two-way communication
Advantage	Able to recognize long distances	High security

Wireless Personal Area Networks (WPANS) and Protocols

Sensors and other Internet-connected objects require a means for transmitting and receiving data. This section discusses Personal Area Networks (PAN) and close-range communication. In an IoT ecosystem, sensors and actuators can communicate through copper lines or Wireless Personal Area Networks (WPANS).

Personal Area Network (PAN)

A Personal Area Network (PAN) is a computer network used to connect electronic devices within a user's workspace.

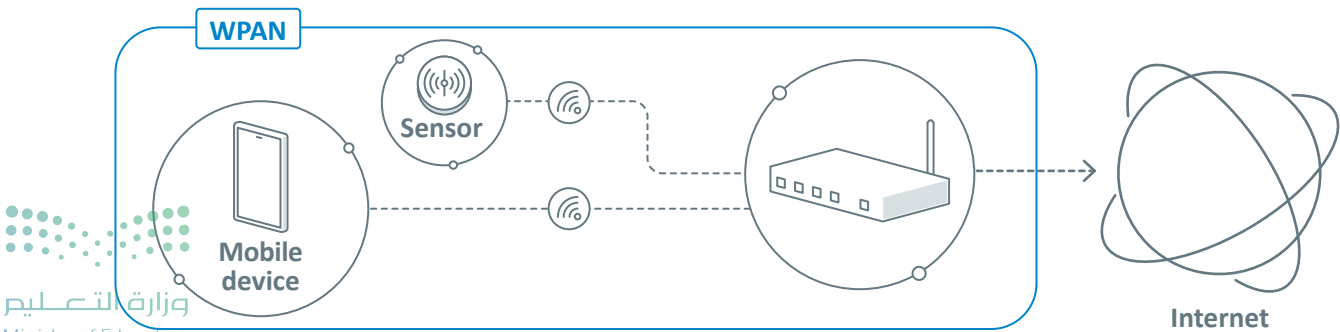


Figure 5.22: WPAN network

Non-IP Based WPANS Protocols

Zigbee

Zigbee is a WPAN protocol based on the IEEE 802.15.4 foundation that is designed for cost-, power-, and space-constrained commercial and residential IoT networking. Zigbee can form networks, discover devices, secure the network, and manage the network. Zigbee does not provide data transport services or an application execution environment. Zigbee is essentially a self-healing mesh network.

The following table illustrates the main components of a Zigbee network.



Table 5.3: Main components of a Zigbee network

Component	Description
Zigbee controller (ZC)	A highly capable device used to build and initiate network functions on a Zigbee network, able to assign logical network addresses and allow nodes to join or leave the mesh.
Zigbee router (ZR)	This optional component handles a portion of the mesh network by assigning logical network addresses and allowing nodes to join or exit the mesh.
Zigbee end device (ZED)	This is a simple straightforward endpoint device, such as a light switch or thermostat, that has the necessary capabilities to communicate with the coordinator.

Zigbee addresses three distinct data traffic types.

- 1 **Periodic data:** The rate of periodic data delivery or transmission is determined by the applications (for example, sensors periodically transmitting). When an application or external stimuli happens at a random pace, intermittent data is produced.
- 2 **Intermittent data:** A light switch is a nice example of intermittent data ideal for Zigbee.
- 3 **Repeated low-latency data:** Zigbee assigns transmission time slots and can have very low latency, making it suitable for computer mice and keyboards.



Sensor



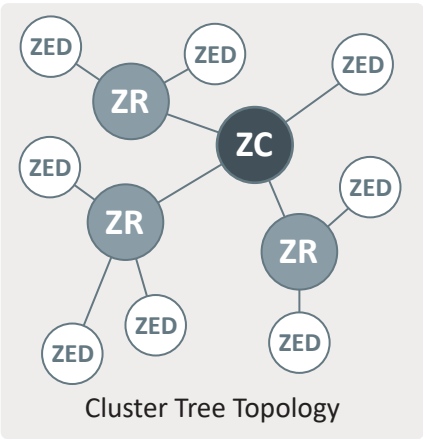
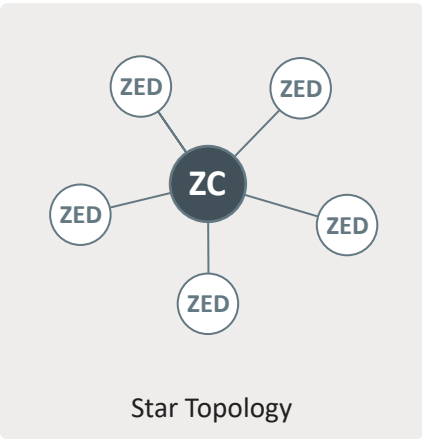
Light switch



Mouse

There are three fundamental Zigbee topologies:

Table 5.4: Zigbee topologies	
Component	Description
Star Topology	A ZC containing one or more ZEDs. Only extends two hops, limiting the distance between nodes. A dependable link with a single point of failure at the ZC is also required.
Cluster Tree topology	A multi-hop network that uses beaconing to extend coverage and range. ZEDs are endpoints, although ZC and ZR nodes can have child nodes. Child nodes communicate only with their parent nodes. Parent nodes can communicate upstream or downstream with their children. A central failure point remains a problem.
Mesh Topology	Any source device can be routed to any destination device. Utilizes tree-based and table-based routing methods. To execute routing functions, ZC and ZR radios must be powered at all times, draining battery life. Routers within a specific range of each other are permitted to interact directly. The primary benefit is that the network may expand outside the line of sight and have redundant pathways.



Hop
When a packet is passed from one network segment to the next, this is a hop.

Beaconing
Beaconing in networking is a periodic digital broadcast, like a lighthouse beacon.

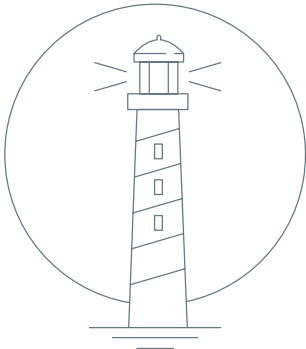
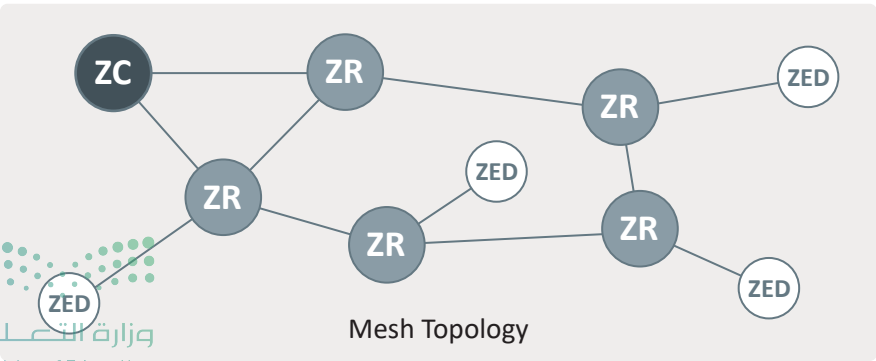


Figure 5.23: Zigbee topologies

Bluetooth

Bluetooth is a low-power wireless communication technology widely utilized in devices ranging from mobile phones to keyboards and gaming consoles. Bluetooth has been used extensively in IoT deployments for some time, as the primary device for beacons, wireless sensors, asset tracking systems, remote controls, health monitors, and alarms when operating in low energy mode (LE).



In a Bluetooth WPAN, a number of Bluetooth events can occur. The two fundamentals are:

Advertising

Initiated by a device to warn scanning devices of the existence of a device requesting to pair or relay a message contained in an advertising packet.

Connecting

This event describes the process of pairing a device with a host.

In Low Energy mode (LE), a device can carry out a complete communication utilizing only the advertising channel. Alternately, communication may involve pair-wise bidirectional communication and necessitate a formal connection between the devices. Devices required to make this sort of connection will begin the formation procedure by listening for advertising packets. In this situation, the listener is considered an initiator. If the advertiser transmits a connectable advertising event, the initiator may submit a connection request using the same physical channel it received the connectable advertising packet on.

The advertiser can then decide whether to establish a link. If a link is established, the advertising event concludes and the initiator is referred to as the master and the advertiser as the slave. Bluetooth terminology refers to this connection as a piconet, and connection events take place. The connection events between the master and slave all occur on the same beginning channel. After data has been transferred and the connection event has ended, frequency hopping can be used to select a new channel for the pair.

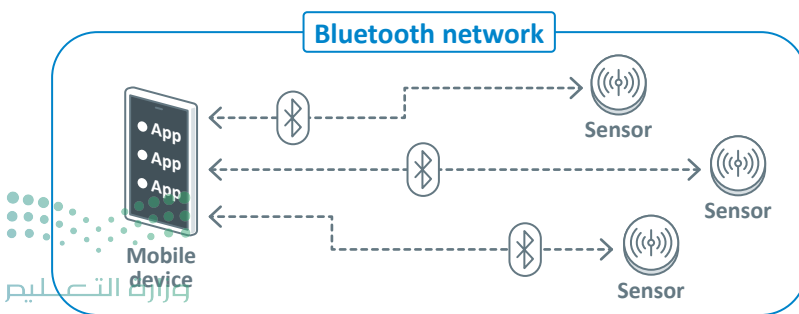
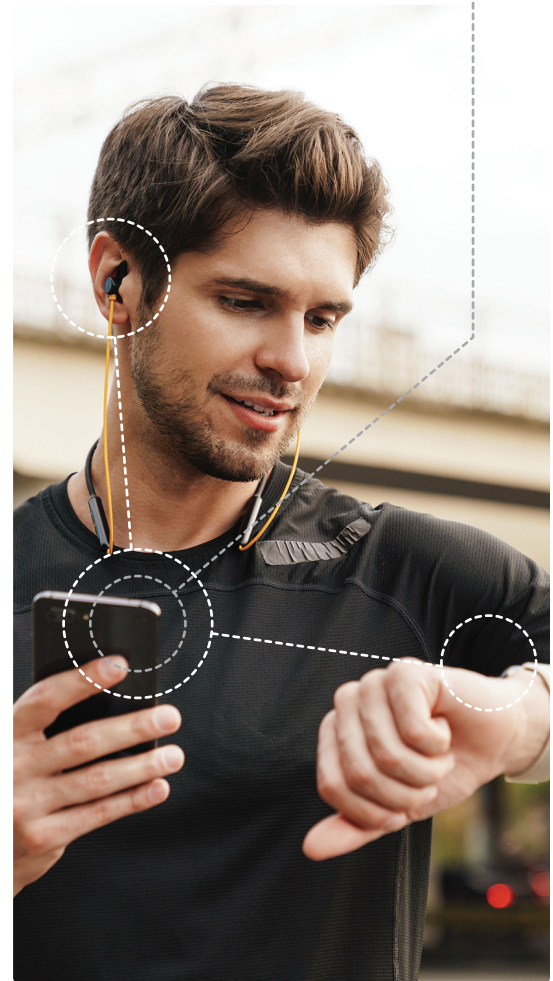


Figure 5.24: Bluetooth network

Figure 5.25: Bluetooth connectivity

IP Based WPANS Protocols

6LoWPAN

IP networking over low-power RF communication systems is intended for devices with limited power and space that do not require high bandwidth networking services. The protocol is compatible with various WPAN communications, including IEEE.802.15.4, Bluetooth, and sub-1 GHz RF technologies, as well as Power Line Controller (PLC). The primary benefit of 6LoWPAN is that even the most basic sensors may be IP-addressable and function as network citizens via 3G/4G/LTE/Wi-Fi/Ethernet routers. IPV6 can adequately cover an estimated 50 billion Internet-connected devices and continue to do so long into the foreseeable future. IPV6 is hence well-suited for IoT expansion.

6LoWPAN networks are mesh networks that exist on the outskirts of bigger networks. The topologies are adaptable, allowing for ad hoc and disjointed networks with no ties to the Internet or other systems, or they may be linked to the backbone or the Internet through edge routers. Various edge routers can connect multiple 6LoWPAN networks; this is known as multi-homing. In addition, ad hoc networks can emerge without the need for an edge router's Internet access.

Edge routers create 6LoWPAN mesh networks on the perimeters of larger, conventional networks. They can also facilitate IPV6-to-IPV4 swaps if necessary. Datagrams are handled similarly to an IP network, which offers some advantages over proprietary protocols. All nodes inside a 6LoWPAN network share the IPV6 prefix established by the edge router. Throughout the Network Discovery (ND) phase, nodes will register with the edge routers.

ND governs the interaction between hosts and routers in the local 6LoWPAN mesh. Multi-homing enables numerous 6LoWPAN edge routers to operate a network; for instance, when failover or fault tolerance requires various media (4G and Wi-Fi).

Thread

Thread is an IoT networking protocol based on IPV6 (6LoWPAN). Its primary objective is home automation and home networking. Thread is IP-addressable and is based on the IEEE 802.15.4 protocol and 6LoWPAN. It shares similarities with Zigbee and other 802.15.4 variations, but differs significantly in that it is IP-addressable. This IP protocol is based on the data and physical layers of 802.15.4 and the security and routing characteristics of 6LoWPAN.

Thread is also mesh-based, making it a viable option for residential lighting systems with up to 250 devices per mesh. The advantage of Thread is that by providing IP addressability in very small sensors and home automation systems, one may reduce power consumption because the protocol does not require application state persistence at the network layer. This also means that the edge router hosting a Thread mesh network does not need to handle application layer protocols, hence reducing its power and processing requirements. Being IPV6 compatible and having all communications encrypted using the Advanced Encryption Standard (AES), it is naturally secure.



Figure 5.26: WPAN networks

Long Range Communication Networks and Protocols

Wireless Personal Area Networks (WPAN) and Wireless Local Area Networks (WLAN) link sensors to a local network, but not necessarily to the Internet or other systems. The IoT ecosphere will encompass sensors, actuators, cameras, smart-embedded gadgets, vehicles, and robots at the most remote locations. Long-term, we must deal with the Wide Area Network (WAN).

LoRaWAN

Low-Power Wide-Area (LPWA) wireless technologies are ideally suited for long-range and battery-powered endpoints. Frequently, LoRaWAN topology is referred to as "star of stars" topology. Endpoints exchange packets via gateways functioning as bridges, with a central LoRaWAN network server. Endpoints communicate directly with one or more gateways, whereas gateways connect to the backend network via regular IP connections.

The same packets can be received and transported by many gateways. When duplicate packets are received, the network server is responsible for de-duplication.

Unlicensed LPWA technologies give new options for private corporate networks, broadcasters, and mobile and non-mobile service providers to deploy IoT infrastructures, solutions, and use cases. The ecosystem of endpoints is expanding fast and will undoubtedly be the deciding factor between the various LPWA technologies and solutions, such as LoRaWAN.

Smart city operators, broadcasters, and mobile and non-mobile service providers are addressing the need for regional or national IoT infrastructures, which are vital for enabling use cases for consumer markets.

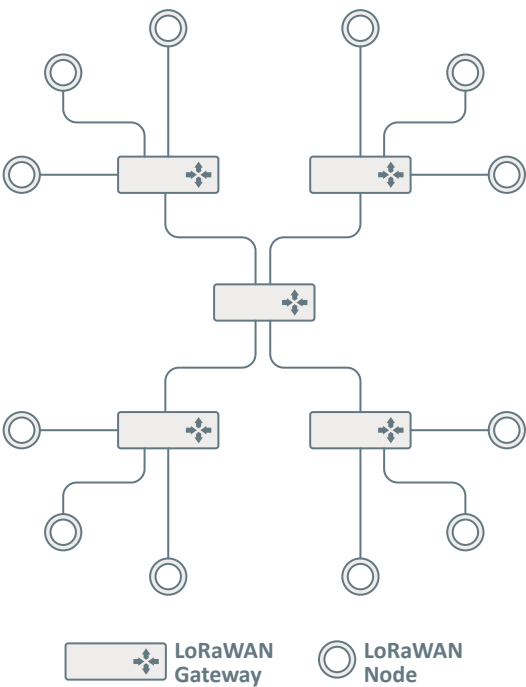


Figure 5.27: LoRaWAN "star of stars" topology

Cellular Networks (5G)

The most common kind of communication is cellular radio, especially cellular data. Prior to the development of cellular technology, mobile communication devices had limited coverage, shared frequency space, and were effectively two-way radios. Cellular Networks are excellent at carrying data in both directions at fast speeds, but at the expense of range and battery consumption.

5G is the next-generation IP-based communication technology that is being developed to succeed 4G cellular networks. Additionally, 5G enhances bandwidth, latency, density, and user expense.

5G aims to be a single umbrella standard that encompasses all cellular services and categories, as opposed to building distinct services and categories for each use case.

Table 5.5: Main features of modern 5G networks

Features	Description
	Enhanced Mobile Broadband (eMBB)
	Ultra-Reliable and Low-Latency Communications (URLLC)
	Massive Machine Type Communications (mMTC)

Exercises

1

Read the sentences and tick ✓ True or False.	True	False
1. The OneM2M architecture contains a data layer.	☐	☐
2. VPN services can be used in the services layer of an OneM2M architecture.	☐	☐
3. In the IoT World Forum architecture, the Applications layer can contain monitoring services.	☐	☐
4. NFC technologies are used for long-range communication between devices.	☐	☐
5. The Zigbee protocol communicates through UDP network channels.	☐	☐
6. The Zigbee router is responsible for the self-healing properties of mesh networks.	☐	☐
7. The advertising event of Bluetooth communications sends data packets to nearby devices.	☐	☐
8. Thread is not a mesh-based network protocol.	☐	☐
9. Smart city network systems do not need Long Range Communication Networks and Protocols.	☐	☐
10. The 5G network is a low-power technology.	☐	☐

2

Classify the key layers of the oneM2M architecture for IoT systems.



3 Analyze the main layers of the IoT World Forum system architecture.

4 Identify the main characteristics of RFID and NFC technologies.

5 Categorize the two main types of WPANS and provide examples for each type.



6 Identify the three key components of a Zigbee network.

7 Distinguish the two fundamental events occurring during a Bluetooth connection.

8 Describe the two main IP-based WPANS protocols.



9 Define the "star of stars" topology that LoRaWAN networks use.

[illegible]

10 Analyze how 5G technologies have evolved from 4G network technologies.





Lesson 3

Security and Privacy of IoT Systems

Security

Internet, IoT, Cloud-Based Services, Cyber-Physical Systems (CPSs), and mobile devices define modern life in the 21st century. Technology allows worldwide communication, which benefits society. However, as technology evolves, cybercriminals can exploit more vulnerabilities. IoT's impact on enterprises and business models grows. The success of IoT for businesses depends on consumer trust. Nevertheless, many technological products and services are rushed to market with low concern for users' security and privacy.

Security is a critical part of the design process from the starting level to each next level. Security policies, protocols, and standards must be created in parallel to support any technological development.

The following table presents IoT security principles

Cyber-Physical System (CPS)

A Cyber-Physical System (CPS) is a computer system that controls or monitors a mechanism using computer-based algorithms.

Table 5.6: IoT security principles

Principle	Description
 Trust	Allow only authorized users or services to access the device or data.
 Identity	Verify the identity of individuals, services, and "things."
 Privacy	Maintain the privacy of a user's device, personal information, and sensitive data.
 Protection	Safeguard devices and users against physical, financial, and reputational harm.

User-Centered Challenges of IoT Systems

Traditional security measures are insufficient to provide comprehensive security to the modern connected world. Unlike many traditional electronic devices, IoT devices interact with services on the Internet. Many potential benefits will not be realized until people become comfortable with these technologies. Accountability is critical for trust between end-users and the creators of IoT systems. The complexity of distributed data flows, inadequate consent mechanisms, and a lack of information to the user all contribute to the need to build accountability into the IoT.

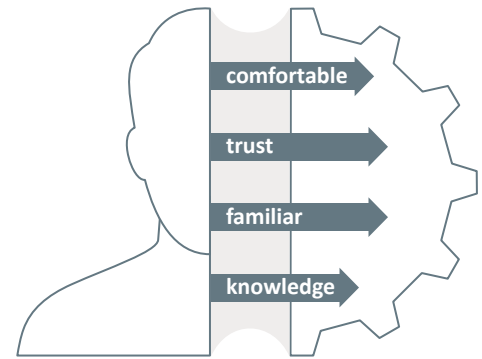


Figure 5.28: Accountability in the IoT

IoT Security and Cybercrime

Internet infrastructure is a physical construct inside sovereign nations' territorial boundaries. Nonetheless, the data flowing over this infrastructure traverses several national jurisdictions, which remains a cyberspace-specific concern. While illegal conduct in cyberspace easily crosses geographical borders, law enforcement does not.

The gap between legislation and technology is a major obstacle in combating cybercrime. The criminal justice system is inherently retrospective and time-consuming, creating significant difficulties for cyberspace regulation. The rate at which technology is integrated into our society outpaces the creation of policy and legislation. As a result, cyberspace is controlled by a patchwork of inadequate, underdeveloped, and competing laws.

Additionally, a consensus is difficult to achieve because each nation has its autonomous norms, beliefs, and practices, promoting different visions for cyberspace. Various nations, for instance, promote cyber sovereignty, arguing that national borders apply to cyberspace and that each nation should be able to regulate how individuals and corporations use the Internet within its borders.



Figure 5.29: Ransomware malware attack and breach

Architectural Challenges of IoT Security

IoT requires a set of standards and a well-defined architecture with interfaces, data models, and protocols due to the variety of devices, protocols, and services involved. Numerous attacks are possible when IoT devices connect with a cloud service and exchange data for the first time. Various IoT device characteristics might pose security risks and issues. Mobility, interdependence, and other similar characteristics introduce various challenges and dangers, such as firmware vulnerabilities, storage, processing power, network attacks, rules, and standards, that necessitate additional study. The Internet of Things necessitates more IPv4-to-IPv6 transitioning devices, necessitating an increase in bandwidth. The adoption of IPv6 and 5G, as well as the new generation of communication for improved speed, generate additional risks and difficulties.

The following illustrations show how a simple architecture evolves from a simple system to an increasingly complex one. Each layer of complexity has new vulnerabilities to the system's components.

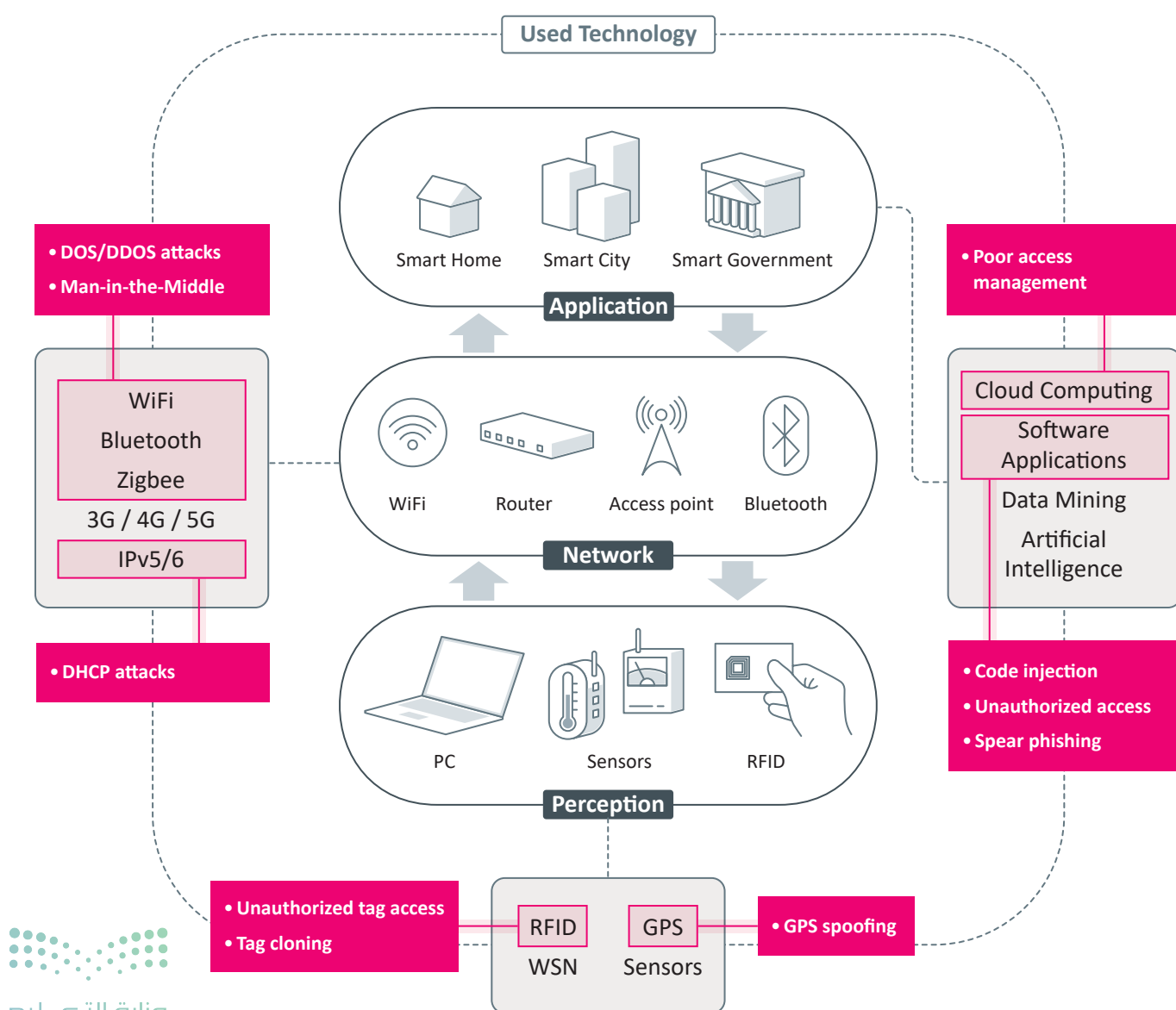


Figure 5.30: Security vulnerabilities in IoT systems

5G Networks and IoT Security

5G is a promising technology that has been identified as the next step in the global evolution of mobile communication over the long term. 5G is the primary component of a networked or IoT/M2M-oriented society. It will enable fast access to information and services. The objective of 5G is mobile connectivity for humans as well as mobile and ubiquitous connectivity for any computing device and application that can benefit from being connected to the Internet (IoT) and the Web (WoT—Web of Things).

Due to the development of 5g networks, it is reasonable to raise issues related to the impact of 5G on the communication security of IoT devices. There will be a need for IoT middleware and a security standard to implement new methods for interconnecting various cognitive networks and devices. With a better and faster network infrastructure, there will be greater interaction between things, particularly with the distribution of processing for cloud services, generating a high impact in terms of data security and enabling the development of new applications that improve people's lives.

The following table illustrates the main IoT 5G security concerns.

Table 5.7: IoT 5G security concerns	
Concern	Description
 Big Data Security	IoT systems continuously create large amounts of heterogeneous data. In addition, data traffic demands for mobile communication in IoT systems will expand considerably. Therefore, it is necessary to devise an effective method for managing these large amounts of data created by IoT systems. 5G network technologies deliver data at a substantially lower cost per bit than previous networks. Secure protocols are needed to properly manage and organize these massive amounts of data to establish a comprehensive security solution for a 5G-based IoT system.
 Device and application protection	Protecting numerous devices and applications is an additional difficulty. A crucial feature of 5G-based IoT systems is the potential to support a far larger number of devices and applications than is now possible. The connections of millions of additional devices and applications will introduce new security concerns. Even with a simple cyberattack, victims could be locked out of their homes, cars, and other linked devices.
 Communication Channels Protection	Maintain the privacy of a user's device, personal information, and sensitive data.

Privacy

While online security remains a major concern and challenge in the IoT environment, preserving privacy will also remain a significant challenge that requires additional attention. The privacy of IoT end-users could be jeopardized if personal data is leaked to unauthorized parties. Given the diversity of IoT-connected devices and the inherent vulnerabilities of hardware and software in some of them, protecting end-user privacy may present numerous security challenges.

The vast amount of personal data captured by big data systems allows organizations to combine different datasets, increasing the ability to identify individuals. The capacity to mine and analyze datasets grows in volume and variability daily. To overcome this, it is prudent to ensure that personal data is completely anonymized. Organizations that use anonymized data must demonstrate that they conducted a thorough risk assessment and implemented effective security techniques. This could include a variety of technical safeguards, such as data masking and pseudonymization, as well as legal and organizational safeguards.

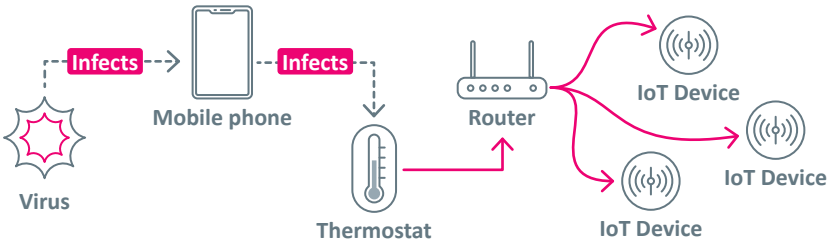


Figure 5.31: Infection of a network

Example

Hackers can infect an IoT network and collect private data by exploiting Universal Plug and Play (UPnP) devices. UPnP offers zero configuration, meaning no authentication is required to connect. Hackers exploit this feature to infect a device and then an IoT network. For example, a mobile phone infected with a virus could connect to a thermostat in your smart home residence through WiFi. This thermostat is connected through UPnP to the router of your smart home. The whole IoT network is infected with this virus, and now a data breach of private information has occurred.

Data masking

Data masking is the process of changing sensitive data. The data is of no value to unauthorized intruders but is still usable by software and authorized personnel for further analysis.

Pseudonymization

Pseudonymization is a data management and de-identification process that replaces personally identifiable information fields in a data record with one or more pseudonyms.

Differential Privacy

In differential privacy, a controlled amount of randomness is added to a data set without affecting dataset accuracy. This technique is used to prevent identifying any personal information of individuals in the data set.

Universal Plug and Play (UPnP)

Universal Plug and Play (UPnP) is a service that enables devices on the same local network to automatically find and connect to each other using standard networking protocols. Printers, routers, mobile devices, and smart TVs are all types of UPnP devices.



Personal Sensitive Data

This is the full data including personal and special data

Name	Ali Sami
Date of birth	24.02.84
Email	asami@mail.com
User ID	ASami_84
Health	Type 1 diabetes



Pseudonymous Data

IDs are replaced and sensitive data is encrypted.

Name	User 458230
Date of birth	24.02.84
Email	#Sd24@!04gTu
User ID	%UTopRg#Ku!1
Health	Type 1 diabetes



Anonymous Data

IDs removed and sensitive data randomised/generalised

Sex	Male
Age	30-49
Health	Type 1 diabetes

Figure 5.32: Pseudonymization and data masking

Data protection and security are difficult in an IoT environment because at the system's core is a communication interface between smart objects without human intervention. Given the rate of the evolution of such systems, it is not surprising that there is little evidence to suggest that data protection is keeping up. Even when legislators demonstrate an awareness of specific concerns in large-scale data processing, their understanding of risk implications may be insufficient in practice.

The following table shows the current IoT privacy concerns and their possible solutions.

Table 5.8: IoT privacy concerns and their possible solutions

Privacy concerns	Possible solutions
Data collection from various sources without careful verification of relevance or accuracy.	Utilize AI to validate the accuracy of acquired data.
Big data approaches enable organizations to merge multiple datasets, which enhances the possibility that data may identify living individuals.	Utilize a variety of technical precautions, such as data masking, anonymization, pseudonymization, and aggregation, in addition to legal and organizational safeguards.
The lack of openness in data processing and the complexity of Big Data analytics might contribute to mistrust.	Improve the level of openness by providing privacy disclosures before processing any data obtained.
The difficulty of determining if new uses are consistent with the original intent of data collection.	An organization may collect personal data for one purpose and subsequently analyze it for a completely different purpose. In such a case, the users must be informed of the change, and if necessary, further consent must be acquired.
Any breaches will threaten users' privacy and harm the creators' credibility, decreasing trust and causing users to lose faith in the organization and system.	Technical methods like encryption protocols and blockchain technology are utilized. Access control, video surveillance, and security records are physical security systems that can be implemented.
Design of systems with privacy protection in mind.	A privacy risk assessment will give an early warning system for detecting privacy issues.
Lack of IoT-related national, regional, and global policies and regulatory frameworks which can also conflict with technological development.	It is essential to bring together nations, international organizations, industrial partners, and security and IoT experts from industry and academia to develop solutions to protect IoT-generated personal data.

Exercises

1

Read the sentences and tick ✓ True or False.	True	False
1. A cyber-physical system is a system that only monitors a mechanism.	☐	☐
2. The IoT Protection principle includes the physical security of IoT devices.	☐	☐
3. Cybersecurity laws are implemented in the same way in each country	☐	☐
4. IPv6 and 5G technologies are completely secure.	☐	☐
5. M2M technologies can be created without human intervention.	☐	☐
6. Smart objects that are hackable can become a hazard for their users.	☐	☐
7. Middleware systems for communication between 5G networks are vulnerable to cyberattacks.	☐	☐
8. Personal data generated by any smart object is automatically encrypted.	☐	☐
9. Pseudonymization techniques introduce fake data to protect the real data.	☐	☐
10. Blockchain technologies can help protect data in distributed IoT systems.	☐	☐

2

What is the main concern about IoT systems' rapid development and deployment?



3

This image shows a blank sheet of white paper with horizontal ruling lines. The lines are evenly spaced and run across the width of the page. There are no margins, text, or other markings on the paper.

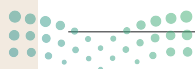
4



5 Distinguish various types of possible attacks on every layer of a simple IoT architecture.

This image shows a blank sheet of white paper with horizontal ruling lines. The lines are evenly spaced and run across the width of the page. There are no margins, text, or other markings on the paper.

6 What is the most significant technological security challenge created by 5G networks in IoT systems? Present your ideas below.



7

This image shows a blank sheet of white paper with horizontal ruling lines. The lines are evenly spaced and run across the width of the page. There are no margins, text, or other markings on the paper.

Project

Smart healthcare is one of the most important sectors that IoT technologies improve. A variety of devices and systems are interconnected and exchange large amounts of data. Medical and biological data are considered the most private personal data and must be protected by companies and governments.

1

In traditional healthcare, medical and biological data would be used by the patient, their doctors, hospitals, and medical centers. In smart healthcare, this data can be accessed from many more points. Write down the types of devices, services, and systems that transport, process, or store personal biological data through smart healthcare systems.

2

Technology companies that build IoT systems are not the only ones responsible for protecting biological data. Governments are accountable for providing legislation and regulation to protect citizens from personal data misuse or breach. Search the Internet for examples of the legislation supplied by the Kingdom of Saudi Arabia for smart healthcare systems. Search the Internet for similar legislation provided by another country of your choosing.

3

After you have written down your notes, use them to create a PowerPoint presentation that shows the potential issues of security and privacy in smart healthcare and the difference in legislation between the KSA and another country of your choosing.



Wrap up

Now you have learned to:

- > Describe how body sensor networks are utilized in smart healthcare applications.
- > Define the types of UAV sensors that are used in smart agriculture IoT applications.
- > Identify the main domains of the oneM2M architecture.
- > Distinguish the different layers of the IoT World Forum architecture.
- > Identify the differences between NFC and RFID technologies.
- > Identify the networking protocols that are used in Wireless Personal Area Networks (WPANS).
- > Classify the main principles of IoT security.
- > Identify security techniques that are established in IoT privacy.

KEY TERMS

6LoWPAN	Internet of Health Things	Personal Area Network
Bluetooth		Pseudonymization
Body Sensor Network	IoT World Forum Architecture	RFID
Cyber Physical System	IPv6	Thread
Data Masking	LoRaWAN	UAV
Edge Computing	Machine To Machine	Wireless Personal Area Network
Electrocardiogram	NFC	Zigbee
Electroencephalogram	oneM2M Architecture	

6. IoT Programming with C++

In this unit, you will learn about smart security applications. You will also learn the fundamentals of the C++ programming language for the Arduino microcontroller and how to transition from codeblocks to C++ in Tinkercad Circuits. Finally, you will build a smart security project with an Arduino microcontroller and program it with C++.

Learning Objectives

In this unit, you will learn to:

- > Identify the benefits and the risks of an IoT security system.
- > Name some common IoT devices used in smart security systems.
- > Recognize the common data types in C++.
- > Use operators in C++.
- > Use conditional statements in C++.
- > Use loops in C++.
- > Create a function in C++.
- > Convert the Tinkercad blocks into C++ commands.
- > Program an Arduino smart security system with C++.

Tools

- > Autodesk Tinkercad Circuits



Lesson 1

Smart Security Applications with C++

Link to digital lesson



www.ien.edu.sa

Smart Security

A smart security system is a way or a process of securing something using a network of cooperating parts and tools. IoT systems can handle inspections in and around your property and keep track of who has access to gates and doors with smart locks if they are installed. For example, smart doorbells can recognize and interact with visitors before unlocking the front door. Motion-activated high-definition cameras are integrated into these gadgets. To secure your home, smart security systems warn you of any anomalies and can set off an alarm or even contact the police.

Benefits

There are numerous reasons for implementing an intelligent home security system. IoT enables remote monitoring and management of your home via a mobile application. Nowadays, smart security devices employ AI to detect environmental changes and warn users. In response to the alert, the devices initiate a specified action. People invest in smart home security systems to make their residences more secure. These cutting-edge technologies provide you with keyless entry to your home and give you real-time security updates.

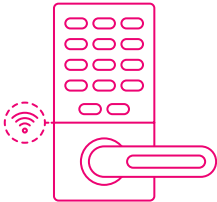
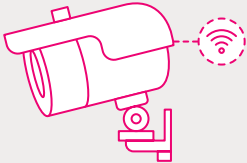
Risks

However, the lack of legislation surrounding the usage and security of IoT devices poses a serious threat to the deployment of IoT in a smart home. In the lack of worldwide security standards, privacy and data security risks arise while using IoT devices. Every Internet of Things gadget in your house collects data. If you want to keep your lifestyle private, you must safeguard every system that gathers and retains your personal information.



Keeping the risks in mind, let’s explore some of the most common IoT-enabled devices used in smart security systems.

Table 6.1: Common IoT-enabled devices

Devices	Uses in smart security systems
 Smart locks and alarms	Smart locks improve the security of your house and allow you to remotely operate the front door. You may set restrictions to enable visitors’ entry at certain periods of time. Some smart locks provide more advanced features such as fingerprint, face-scan, or even eye-scan authentication.
 Smart cameras	A home security system is incomplete without the integration of smart cameras. Cameras serve as the digital eyes of your home, allowing you to watch any activity inside and outside in real-time. There are several intelligent camera options available, including wireless IP cameras that can be monitored from any location with an internet connection. Surveillance video of the regions around the entrance gates may be captured by door or gate cameras.
 Fire and smoke sensors	It is crucial to install fire and smoke detectors to get notified instantly when there is a hazardous situation in your residence. Smart homes are often built with carbon monoxide detectors that provide alerts when they detect dangerously high amounts of gas. They may also activate the sprinkler system or notify the fire department to ensure that the fire does not spread uncontrolled and cause property loss or injuries.
 Motion sensors	Motion detectors are a vital component of a smart security system. Vibrations and inputs are recorded and analyzed in both two and three dimensions by these systems, which then can indicate any irregular motion. They may activate alarms to notify users of activities inside or in the local vicinity of the house.

C++ Language

Achieving robust security is hard and thus we need powerful languages such as C++ to program interfaces. C++ is a high level compiled programming language that has many object-oriented and functional features on top of many low-level memory-handling capabilities. Its main characteristics are performance speed and efficiency. C++ was designed as an extension of the C programming language.

Basic Data Types

In contrast to other programming languages, in C++ you need to specify the type of a variable before using it. Variables need to have a type that indicates what kind of data each variable holds. The C++ program needs this information to know how much memory is needed to be allocated for these data.

Table 6.2: The most common data types in C++

Type	Identifier	Example
Integers	(int)	5, -4
Floating-point numbers	(float or double)	3.14 , -7.5
Characters	(char)	'c'
Booleans	(bool)	bool flag = true;

You can alter a type by using a type modifier e.g. long int. The possible combinations are:

	char	int	double
signed	✓	✓	
unsigned	✓	✓	
short		✓	
long		✓	✓

The programmer can also define their own types based on their needs.

When creating a variable there are some naming rules that you need to follow.

Valid names:

- Can only have alphabets (A-Z, a-z), numbers (0-9), and the underscore (_).
- Cannot begin with a number.
- Cannot be a keyword. For example, int is a keyword that is used to denote integers.

Variables can be declared with or without being initialized.

Arrays

A very common data structure in C++ is the array. An array is essentially a variable that can hold multiple values of the **same** type.

The syntax for declaring an array is:

```
datatype arrayName[arraySize];
```

matrix type

matrix name

matrix size

After declaration, you cannot change the array's type or size. You can access its elements by using their indices.

For example, if you want to store 10 integer values you can create an array in which you will store the values. First, you must declare the array's type and size:

```
int values[10];
```

where int is the type of the elements stored in the array, the array's name is "values" and its size is 10. To fill it with values the command is:

```
values [10] = {0,1,2,3,4,5,6,7,8,9};
```

To access any of its elements, you just need its index. So, the command:

```
int a = values[3];
```

declares a variable called a, which is of integer type and its value is the 4th element of the values array (indexing in C++ starts from 0). Although you can have arrays of more than one dimension, the most common types are one-dimensional or two-dimensional. To create a 2d array, you need to declare the size of each dimension. For example:

```
char keys[4][2];
```

declares an array with 4 rows and 2 columns which can store values of type char. To fill the array with your values you work as in the 1d arrays:

```
keys[4][2] =  
{ {1,2},  
  {3,4},  
  {5,6},  
  {7,8}  
};
```

Here you need a pair of values for each row.



Basic Operators in C++

The basic types of operators are arithmetic, assignment, relational and logical.

Table 6.3: Arithmetic operators

Operator	Operation
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulo Operation (Remainder after division)

Table 6.4: Assignment operators

Operator	Example	Equivalent to
=	a = b;	a = b;
+=	a += b;	a = a + b;
-=	a -= b;	a = a - b;
*=	a *= b;	a = a * b;
/=	a /= b;	a = a / b;
%=	a %= b;	a = a % b;

In integer (int) arithmetic, "/" is used to calculate the quotient of the division and "%" is used for the remainder.
e.g. 5/2=2, 5%2=1

In floating-point (float, double) arithmetic, only "/" is used for the quotient.
e.g. 5.0/2.0=2.5

Table 6.5: Relational operators

Operator	Meaning	Example
==	Is Equal to	3 == 5 gives us false
!=	Not Equal to	3 != 5 gives us true
>	Greater than	3 > 5 gives us false
<	Less than	3 < 5 gives us true
>=	Greater Than or Equal to	3 >= 5 gives us false
<=	Less Than or Equal to	3 <= 5 gives us true

Table 6.6: Logical operators

Operator	Meaning	Example
&&	expression1 && expression2	Logical AND. True only if all operands are true.
	expression1 expression2	Logical OR. True if at least one of the operands is true.
!	!expression	Logical NOT. True only if the operands is false.

Comments in C++

Another basic feature every programming language supports is comments. These comments are ignored by the compiler and are used to improve the readability of the code, making it easier for programmers or code reviewers to understand the functionality of the code. There are two ways to add a comment in C++ depending on whether you want the comment to span multiple lines or not.

Use `//` for a single line of comment

```
// this is a comment
```

```
int y = 10;
```

```
cout << y;
```

Inactive code

Use `/*` to start a block comment and `*/` to end it. Block comments are also used to make a part of the code inactive while testing the functionality of the program. For example, in the following code, the if statement will be skipped by the compiler.

```
lcd.clear();
```

```
lcd.setCursor(0, 0);
```

```
lcd.print("Enter password:");
```

```
bool correctPass = true;
```

```
char buttonPressed;
```

```
/*
```

```
int index = 4;
```

```
buttonPressed = keypad.waitForKey();
```

```
if(password[index] != buttonPressed){
```

```
    correctPass = false;
```

```
}
```

```
*/
```

```
lcd.setCursor(i, 1);
```

```
lcd.print(buttonPressed);
```

Inactive code



Printing in C++

To print a variable **x** in C++ use the following command

```
cout << x;
```

Conditional Statements in C++

To execute certain blocks of code depending on whether a condition is true, you can use three types of conditionals:

- **if statements**
- **if...else statements**
- **if...else if...else statements**

if statement

This type of if statement is used when you want to execute a block of code in case a condition is met.

The syntax of a simple if statement in C++ is:

```
if (condition) {  
    // body of if statement  
}
```

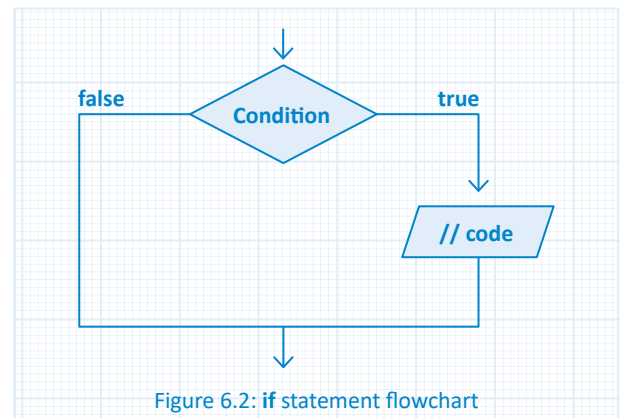


Figure 6.2: if statement flowchart

First, the condition in parentheses is evaluated and if its value is true then the code inside the { } is executed. If the condition is false the code inside the { } is skipped. How if statement works:

Condition is **true**

```
int number = 5;  
if (number > 0) {  
    // code  
}  
// code after if
```

Condition is **false**

```
int number = 5;  
if (number < 0) {  
    // code  
}  
// code after if
```


if...else statement

In this type of if...else statement, either the code block inside the if {} will be executed and then the code inside else {} is skipped or the if {} is skipped then the block of code inside the else {} will be executed.

The syntax of an if...else statement is:

```
if (condition) {  
    // block of code 1 if condition is true  
}  
else {  
    // block of code 2 if condition is false  
}
```

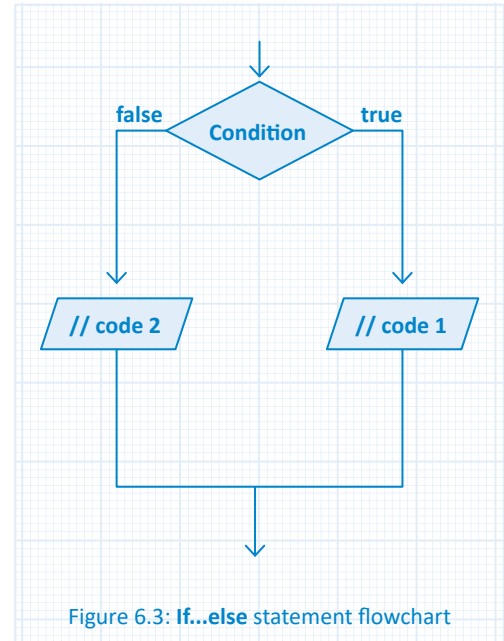


Figure 6.3: If...else statement flowchart

First, the condition in parentheses is evaluated and if its value is true then the code inside the if {} is executed. If the condition is false, the code inside the else {} is executed.

How if...else statement works:

Condition is **true**

```
int number = 5;  
if (number > 0) {  
    // code  
}  
else {  
    // code  
}  
  
// code after if...else
```

Executed

Skipped

Condition is **false**

```
int number = 5;  
if (number < 0) {  
    // code  
}  
else {  
    // code  
}  
  
// code after if...else
```

Skipped

Executed



if...else if...else statement

The last type of conditional if...else if...else is used when you need to check more than one condition or when you need 3 or more blocks of code to be executed depending on some conditions.

The syntax of an if .. else if .. else statement is:

```
if (condition1) {  
    // code block 1  
}  
else if (condition2) {  
    // code block 2  
}  
else {  
    // code block 3  
}
```

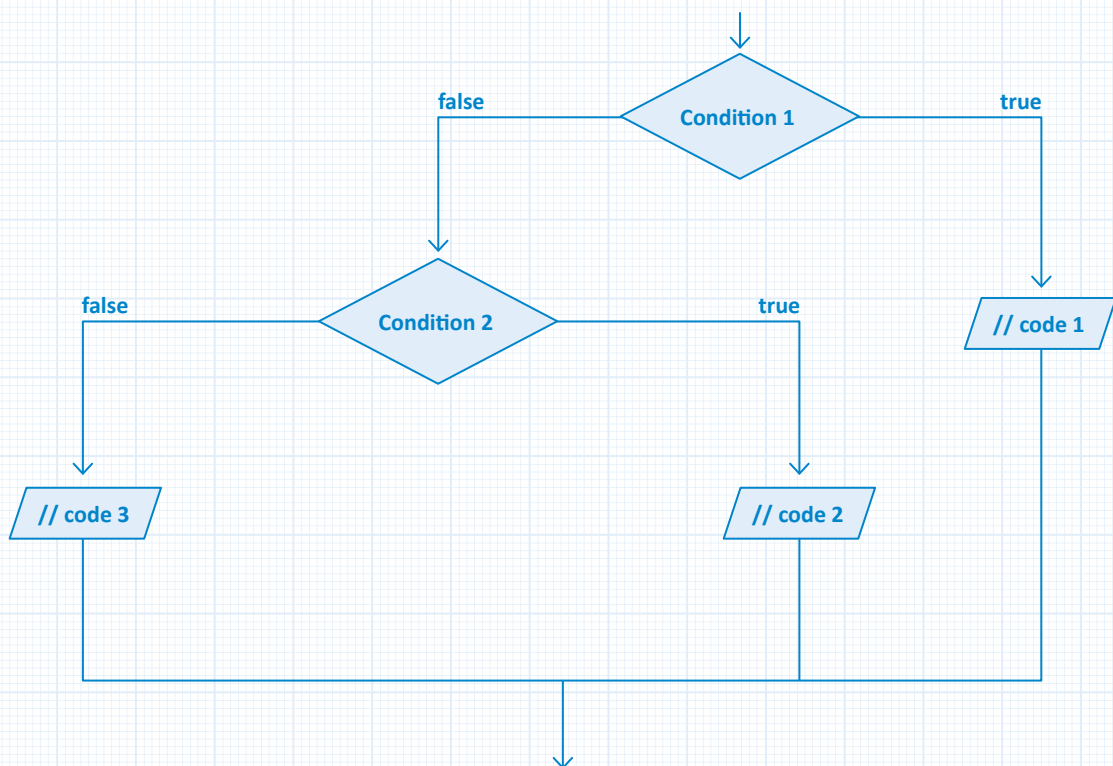


Figure 6.4: If...else if ... else statement flowchart

How if...else if...else statement works:

If condition1 is True, code block 1 will be executed and the rest of the code blocks are skipped.

1st Condition is **true**

```
int number = 2;
if (number > 0) { Executed
    // code
}
else if (number == 0) { Skipped
    // code
}
else {
    // code
}
// code after if
```

If condition1 is False and condition2 is True, code block 2 will be executed and code block 3 is skipped.

2st Condition is **true**.

```
int number = 0;
if (number > 0) { Skipped
    // code
}
else if (number == 0) { Executed
    // code
}
else { Skipped
    // code
}
// code after if
```

If neither condition1 nor condition2 is True, code block 3 will be executed.

All conditions are **false**.

```
int number = -1;
if (number > 0) { Skipped
    // code
}
else if (number == 0) { Skipped
    // code
}
else { Executed
    // code
}
// code after if
```

You can also nest an if statement inside the code block of another if statement. They don't have to be of the same type. For example:

```
// outer if statement
if (condition1) {

    // statements

    // inner if statement
    if (condition2) {
    }
}
// code after if
```

Loops

In C++ you can use three types of loops:

- for loop
- while loop
- do...while loop

for loop

The syntax of a for loop is:

```
for (variable initialization;  
condition; increment operation) {  
    // loop statements;  
}
```

while loop

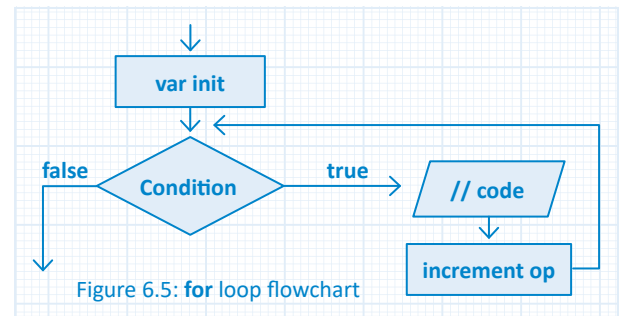
The syntax of a while loop is:

```
while (condition) {  
    // loop statements;  
}
```

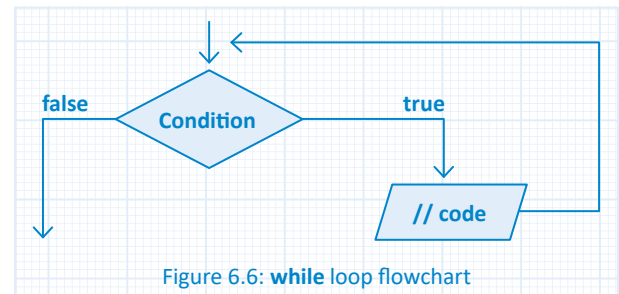
do...while loop

The third iteration type, do...while, is a variation of the while loop and its syntax is:

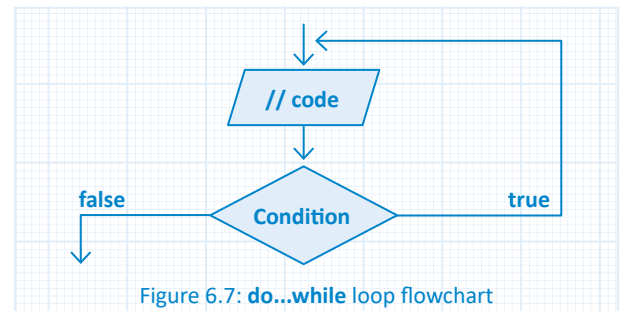
```
do {  
    // statement execution;  
} while (condition);
```



Where variable initialization is executed only once before the loop starts and it sets the starting values of the variables that are part of the condition. In this step, you can also declare and initialize a variable, usually the counter used for the loop iterations, by the condition. If the condition's value is True, the loop statements are executed and then the increment operation updates the values of the initialized variables. This continues until the condition's value changes to False.



Where the loop statements are executed while the condition evaluates to True. When the condition becomes False the iteration stops and the loop statements are skipped.



Its difference from the while loop is that in a do...while loop the condition is checked after the loop statements. This means that the code inside the body of the loop will be executed at least one time. The iteration stops when the condition evaluates to False.

"break" and "continue" Statements

When working with loops there are two very useful statements, "break" and "continue". They both work in every loop type.

break Statement

The break statement will terminate the loop in which it is encountered.

```
for (init; condition; update) {  
    // code block 1  
    if (condition to break) {  
        break  
    }  
    // code block 2  
}  
// code after loop
```

```
while (condition) {  
    // code block 1  
    if (condition to break) {  
        break  
    }  
    // code block 2  
}  
// code after loop
```

If the break statement is found inside the body of a nested loop, it terminates the inner loop.

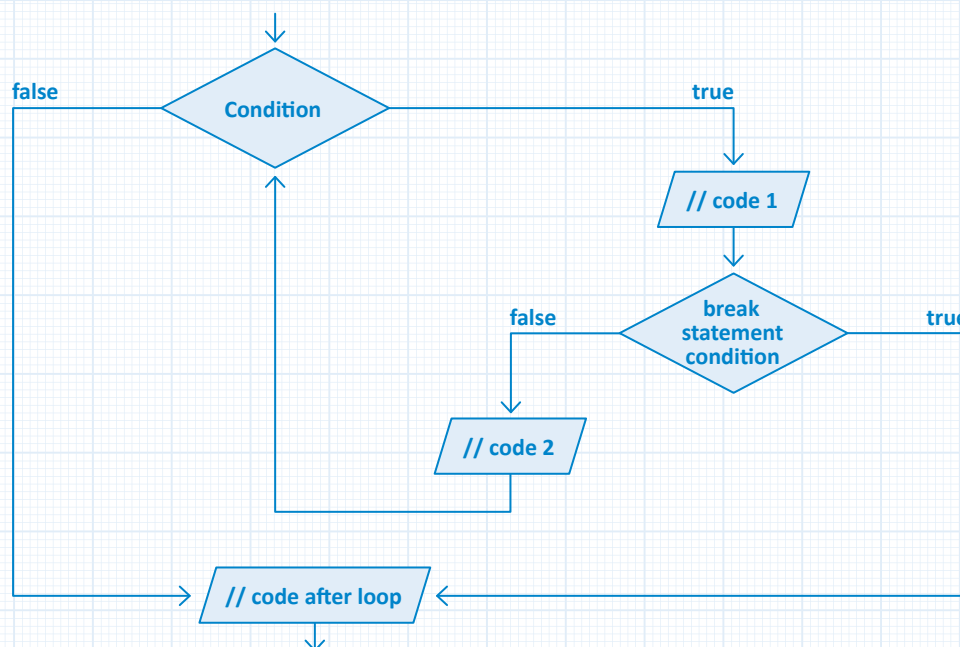


Figure 6.8: break statement flowchart



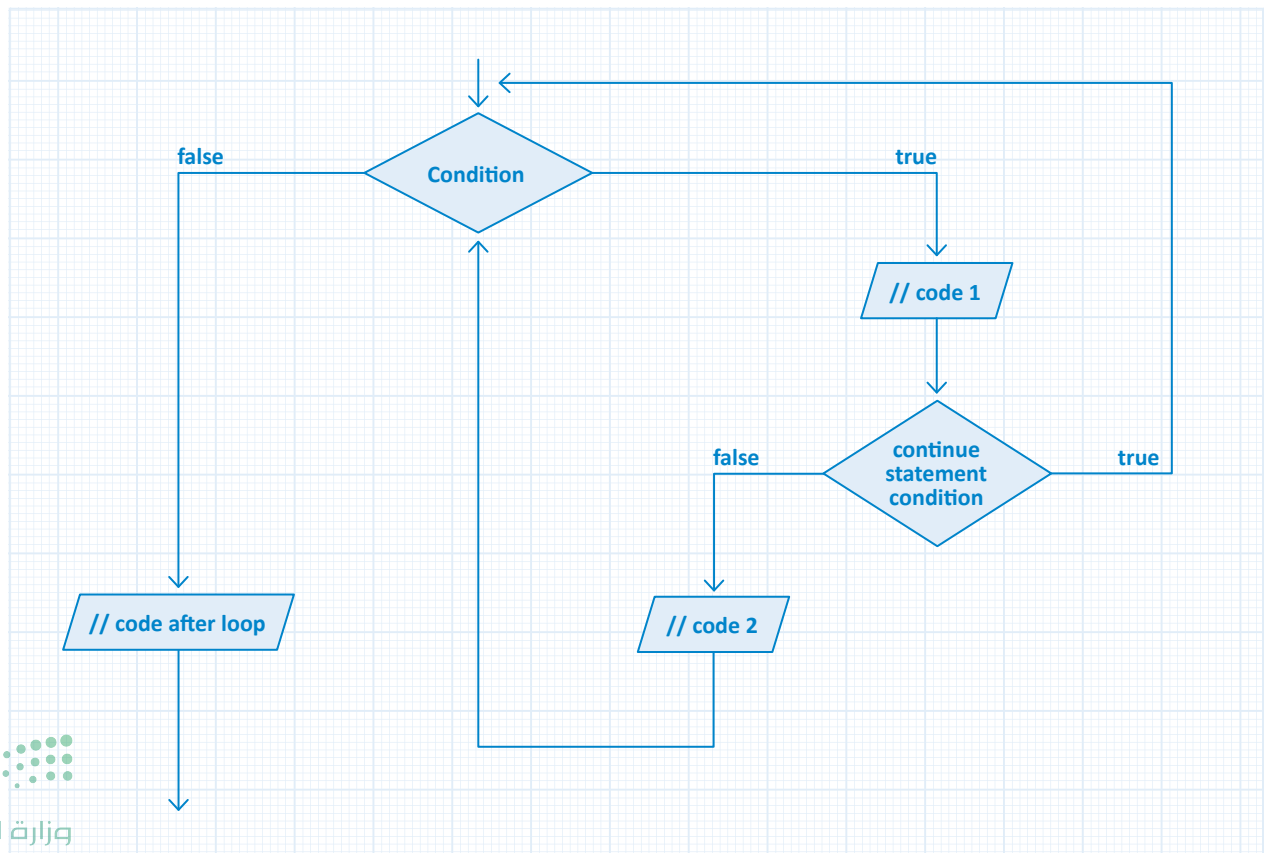
continue Statement

The continue statement will skip the rest of the code inside the loop and go to the next iteration.

```
for (init; condition; update) {  
    // code block 1  
    if (condition to continue) {  
        continue  
    }  
    // code block 2  
}  
// code after loop
```

```
while (condition) {  
    // code block 1  
    if (condition to continue) {  
        continue  
    }  
    // code block 2  
}  
// code after loop
```

If the continue statement is found inside the body of a nested loop it skips the current iteration of the inner loop.

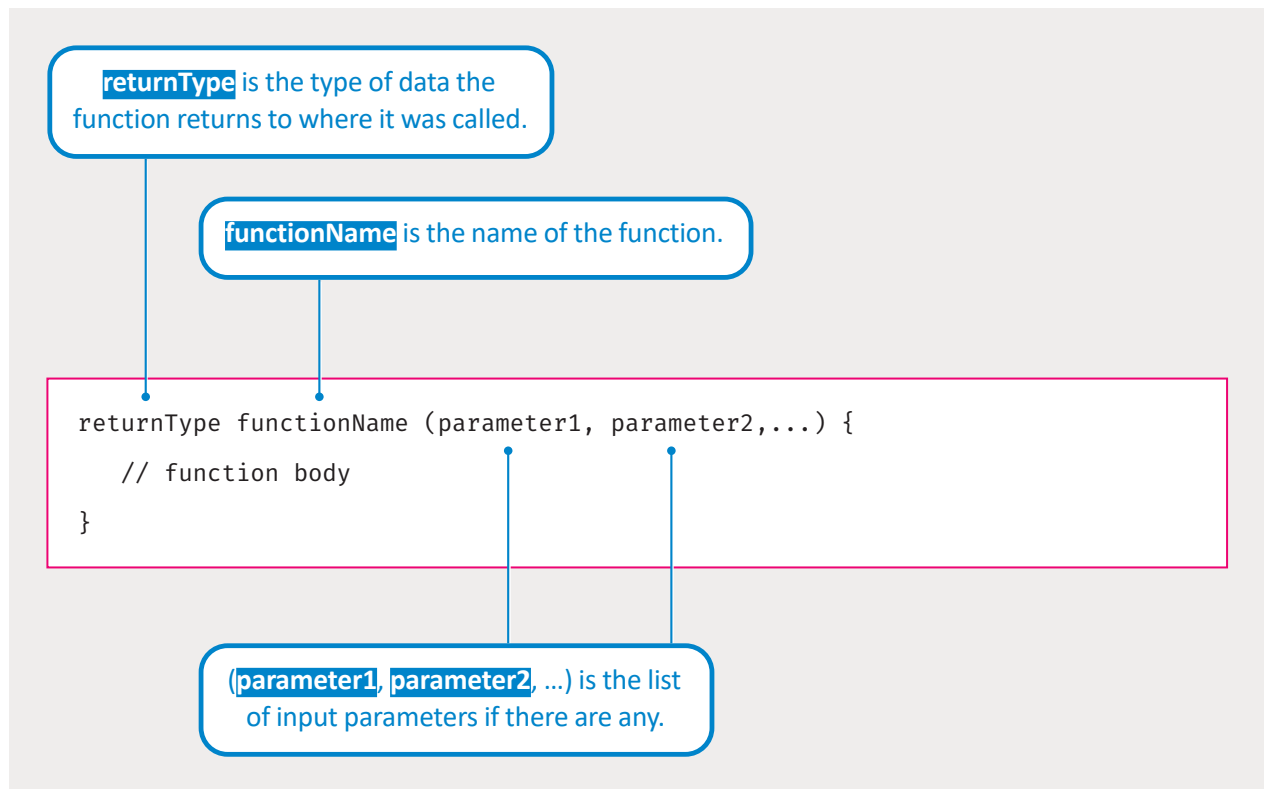


Functions in C++

Quite often in programming, there are tasks that need to be executed many times. One solution would be to write the same lines of code every time these tasks need to be performed. A better solution is to group these lines of code and create a function that performs these tasks. In C++ there are many standard library functions, which are functions that are predefined and can be used by programmers. There is also the ability for programmers to define their own functions based on their needs. These are called user-defined functions.

Every function can accept some variables as input parameters, execute some code which is enclosed in {}, and to end the function there is a return statement, which returns a value.

To create a function, you first need to declare it:



An example of a very simple function that receives two integers and returns their sum is:

```
// function declaration  
int adding (int a, int b) {  
    s = a+b;  
    return s  
}
```



To use this function in your main code you call it by its name and pass as parameters two integers:

```
int main () {  
    int a=2;  
    int b=5;  
    int c;  
    //calling the function and passing a, b as arguments  
    c = adding(a,b);  
    //cout will print the value of c  
    cout << c;  
    return 0;  
}
```

Only in the **main()** function, the return statement is optional and can be omitted.

As you can see, "**main**" is also a function that returns the value 0, hence its return type is int but accepts no input parameters in this case which is indicated by the empty parentheses (). "main" is a special kind of function in C++ where the main code of a program exists.

The type, number and order of arguments passed to a function must match the type of the corresponding parameters in the function declaration.

It is possible for a function to not return any value. In that case, the returnType will be "**void**".

```
void displyNumber () {  
    // code  
}
```

Setup() and Loop() Functions

When writing code for Arduino in the Tinkercad platform, there are two functions that are called to execute the code of the circuit. These functions are called automatically when the program starts its execution unlike the rest of the functions that need to be called explicitly by your code.

The first function that is executed is "void setup()". It is executed only once in the beginning, and it is responsible for setting up the various circuit parts. Things like setting the mode of each Arduino digital pin, establishing communication with the serial terminal and more are handled in the setup() function.

After setup() has been executed, the function "void loop()" is called repeatedly as long as the system is powered. This is the function that performs the main functionality of the circuit.

In general, you want to write the setting up code inside the body of `void setup()`, the main programming logic inside the body of `void loop()`, and any constant or function declarations outside the body of these two functions.

Example of an Arduino C++ program.

```
void setup() {  
    int a = 10;  
    int b = 20;  
}  
  
void loop() {  
    for (int i = 0; i < b; i++) {  
        a += i;  
        cout << a;  
    }  
}
```

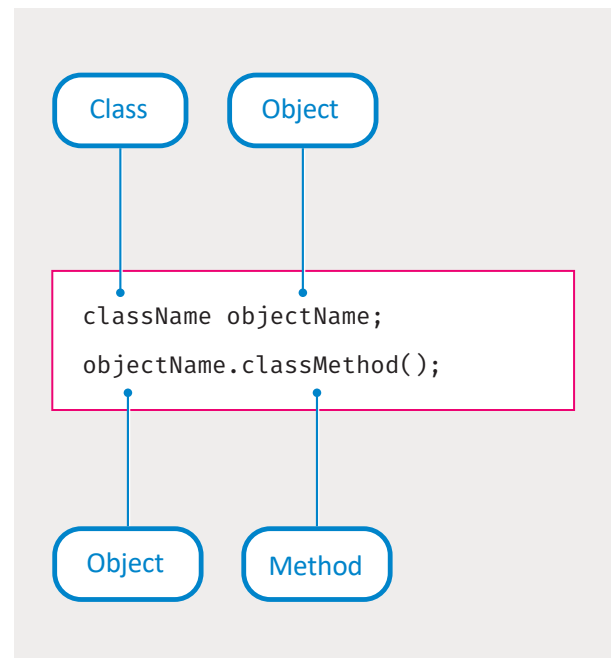
setup() runs once to setup variables and objects.

loop() runs repeatedly in the Arduino.

Classes, Objects, and Methods

In object-oriented programming, you basically want to perform all computations based on "objects". An object is the basic unit of object-oriented programming. Objects can have properties and can perform some basic actions. For example, a servo motor can be considered such an object. It has some properties (name, type) and can perform some basic actions such as reading from a digital pin, rotating its motor by a number of degrees, etc. These actions that each object can perform are called methods and in C++ they are basically functions that have been declared inside the body of an object.

Technically the properties and methods are being declared inside the body of a class and not an object. Avoiding many technical details, you can think of a class as a concept and the objects as the embodiment of that concept. For example, in a circuit simulation where there would be three servo motors, you would first need to declare a class "Servo" and each one of those three motors would be a Servo object, or as it is commonly called an instance of the class Servo.



Exercises

1

Read the sentences and tick ✓ True or False.	True	False
1. IoT devices can lock the doors of a residence.	☐	☐
2. You cannot monitor a smart home application from a smartphone.	☐	☐
3. Legislation is up to date on current issues regarding IoT smart security applications.	☐	☐
4. Smart camera systems can be accessed only by the residence's network.	☐	☐
5. Smart home systems can automatically contact first responder services.	☐	☐
6. Smart lock systems can use biometric data to identify users.	☐	☐
7. C++ is a completely different language than C.	☐	☐
8. C++ is an object-oriented language.	☐	☐
9. C++ arrays are always type-defined.	☐	☐
10. The setup() and loop() functions are not important in an Arduino program.	☐	☐

2

Identify the benefits provided by smart security IoT applications.



3

This image shows a single sheet of white paper with horizontal ruling lines. The lines are evenly spaced and run across the width of the page. There are no margins, text, or other markings on the paper.

4



5 Define the basic data types of a C++ program.

This image shows a single sheet of white paper with horizontal blue or grey ruling lines. The lines are evenly spaced and run across the width of the page. There are no margins, text, or other markings on the paper.

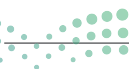
6 Analyze the fundamental rules of naming C++ variables.



7 Demonstrate how "for" loops are implemented in C++.

This image shows a single sheet of white paper with horizontal ruling lines. The lines are evenly spaced and run across the width of the page. There are no margins, text, or other markings on the paper.

8 Describe the difference between the "while" and "do ... while" loops in C++.



9 State the use of the `setup()` and `loop()` functions in an Arduino sketch.

[illegible]

10 Analyze how an electronic component connected to an Arduino board can be abstracted to a C++ class and object.





Lesson 2

From Tinkercad Blocks to C++

Migrating from Visual Blocks Programming to C++ Programming

In this lesson, we are going to learn how to move from programming an Arduino with Tinkercad blocks to programming it with C++. While Tinkercad blocks are useful for rapid prototyping, using native C++ is necessary for utilizing the full capabilities of the Arduino microcontroller. We are going to learn the basic functions and statements to start programming an Arduino microcontroller with C++.

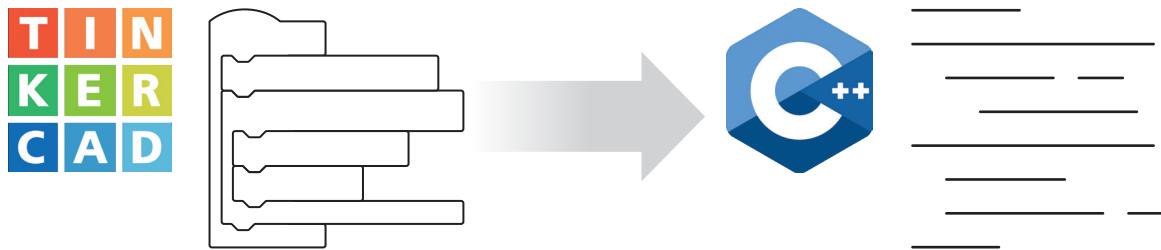


Figure 6.10: Tinkercad blocks to C++ code

Tinkercad provides a simulation environment for Arduino modeling, which does not require a physical Arduino to be connected to a computer.

Variable Assignments and Operations

Declaring and changing variables in Tinkercad blocks is done through the **Variables** and **Math** command groups. The following table illustrates examples of the available commands.

Command groups used:

- | | |
|----------|-----------|
| Output | Control |
| Input | Math |
| Notation | Variables |

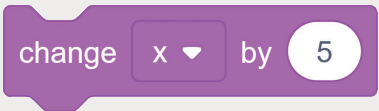
Declaring a variable named **x**.

Tinkercad block	C++
	<pre>int x = 0;</pre>

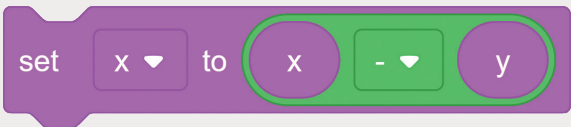
Assigning a value to a variable.

Tinkercad block	C++
	<pre>x = 3;</pre>

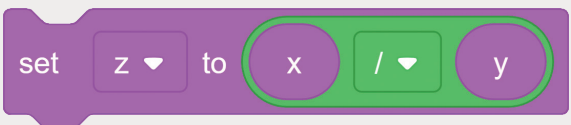
Changing a variable by a specific value.

Tinkercad block 	C++ <pre>x += 5;</pre>
--	-------------------------------

Performing a mathematical operation between variables **x** and **y**.

Tinkercad block 	C++ <pre>x = x - y;</pre>
--	----------------------------------

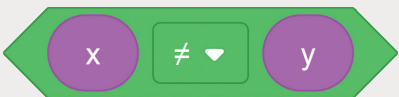
Assigning a third variable **z** to a mathematical operation between variables **x** and **y**.

Tinkercad block 	C++ <pre>z = x / y;</pre>
--	----------------------------------

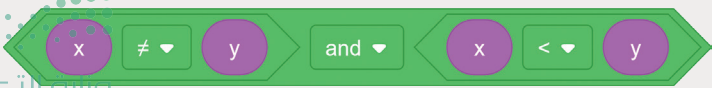
Performing a mathematical comparison between variables **x** and **y**.

Tinkercad block 	C++ <pre>x < y</pre>
--	--------------------------------

Performing a logical comparison between variables **x** and **y**.

Tinkercad block 	C++ <pre>x != y</pre>
--	------------------------------

Performing a logical operation between two statements.

Tinkercad block 	C++ <pre>x != y && x < y</pre>
--	--

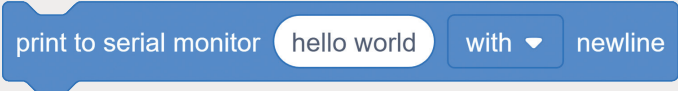
Conditional Statements, Loops and Output Messages

Building conditional statements, loops and output messages in Tinkercad blocks is done through the **Control** and **Output** command groups. The following table illustrates examples of the available commands.

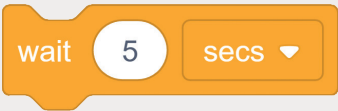
Command groups used:

 Output	 Control
 Input	 Math
 Notation	 Variables

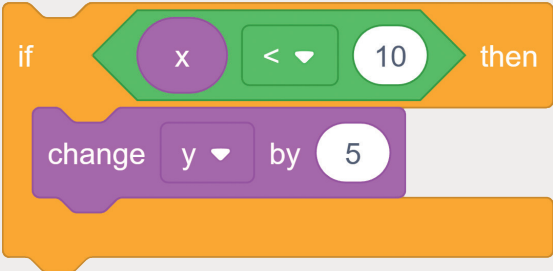
Print a message to the **Serial Monitor**.

Tinkercad block 	C++ <pre>Serial.println("hello world");</pre>
--	--

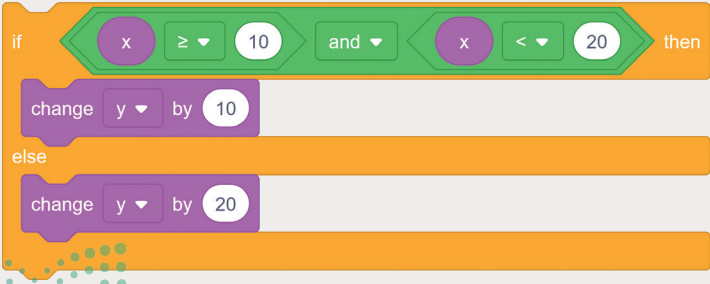
Wait for 5 seconds.

Tinkercad block 	C++ <pre>delay(5000);</pre>
--	------------------------------------

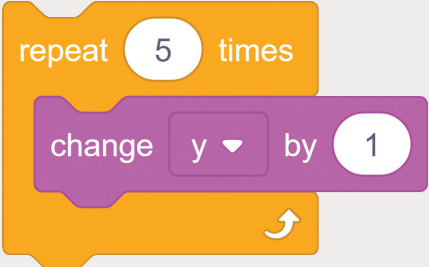
Execute the blocks in the **if** codeblock if the logical condition is true.

Tinkercad blocks 	C++ <pre>if (x < 10) { y += 5; }</pre>
--	--

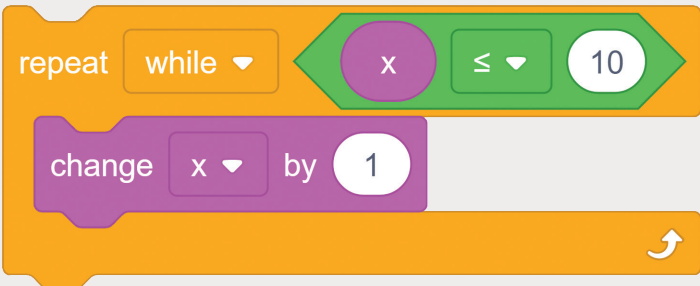
Execute the blocks above **else** in the **if** codeblock if the logical condition is true, else execute the blocks below **else**.

Tinkercad blocks 	C++ <pre>if (x >= 10 && x < 20) { y += 10; } else { y += 20; }</pre>
---	---

Execute the blocks in the **for** codeblock if the logical condition is true.

Tinkercad blocks	C++
	<pre>for (counter = 0; counter < 5; ++counter) { y += 1; }</pre>

Repeat a while loop under the following condition.

Tinkercad blocks	C++
	<pre>while (x <= 10) { x += 1; }</pre>

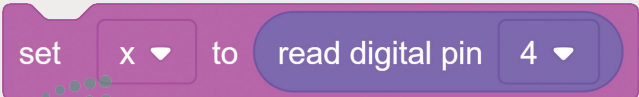
Arduino Digital and Analog Pin I/O

Interacting with the Arduino board's digital and analog pins in Tinkercad blocks is done through the **Input**, **Output**, and **Math** command groups. Each time that a pin is used, whether it is analog or digital, Tinkercad blocks recognize if it is used for digital or analog I/O. To use a pin, you need to specify in the **setup()** Arduino function whether it is used in **INPUT** or **OUTPUT** mode. For analog output, the pins **3, 5, 6, 9, 10, 11** are used with **Pulse Width Modulation (PWM)**. The following table illustrates examples of the available commands.

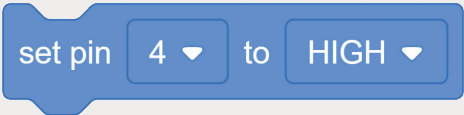
Command groups used:

- | | |
|--|---|
|  Output |  Control |
|  Input |  Math |
|  Notation |  Variables |

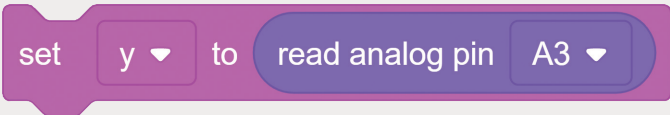
Getting the value of the digital pin **4** and storing it in the variable **x**.

Tinkercad block	C++
	<pre>pinMode(4, INPUT); x = digitalRead(4);</pre>

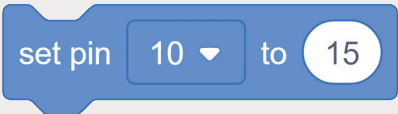
Setting the value of the digital pin **4** to **HIGH**.

Tinkercad block	C++
	<pre>pinMode(4, OUTPUT); digitalWrite(4, HIGH);</pre>

Getting the value of the analog pin **A3** and storing it in the variable **y**.

Tinkercad block	C++
	<pre>pinMode(A3, INPUT); y = analogRead(A3);</pre>

Setting the value of the pin **10** to the analog value **15** using **PWM**.

Tinkercad block	C++
	<pre>pinMode(10, OUTPUT); analogWrite(10, 15);</pre>

Examples of Migration from Tinkercad Blocks to C++

We will create simple examples in Tinkercad to showcase the migration of programming the Arduino board with Tinkercad blocks, to programming it with C++.

Blinking LEDs Example

We will build a simple program that creates two loops that blink an LED light 5 and 10 times at different rhythms.

Components needed:

- 1 Arduino Uno R3
- 1 LED

Programming in C++ in Tinkercad can start by converting the editing type from Blocks to Text.

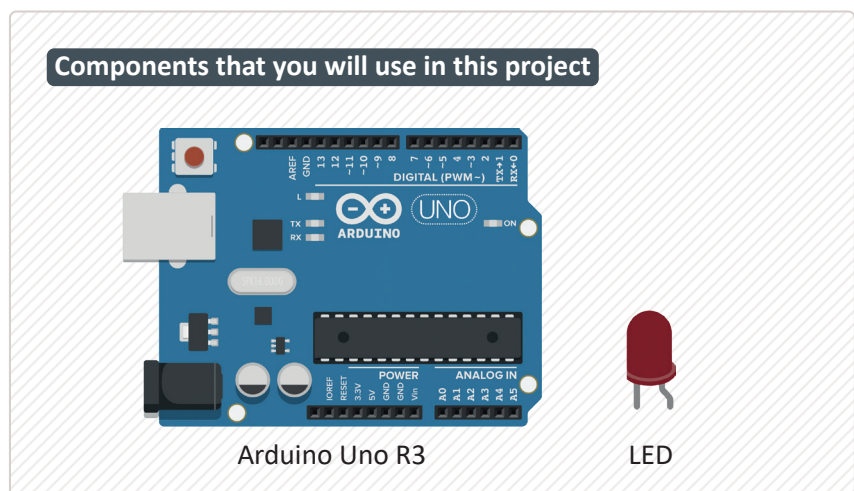


Figure 6.11: Project components

To connect the LED:

- > Connect the **Cathode** of the LED to the **GND** of the Arduino and change the wire color to **black**. ①
- > Connect the **Anode** of the LED to the **Digital Pin 11** of the Arduino and change the wire color to **green**. ②

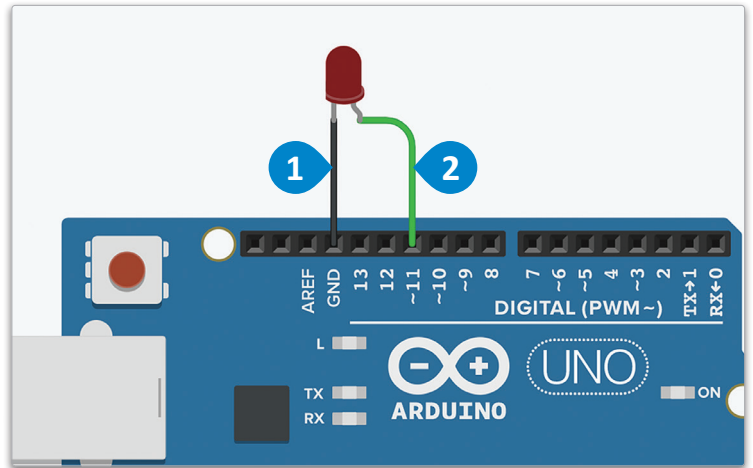
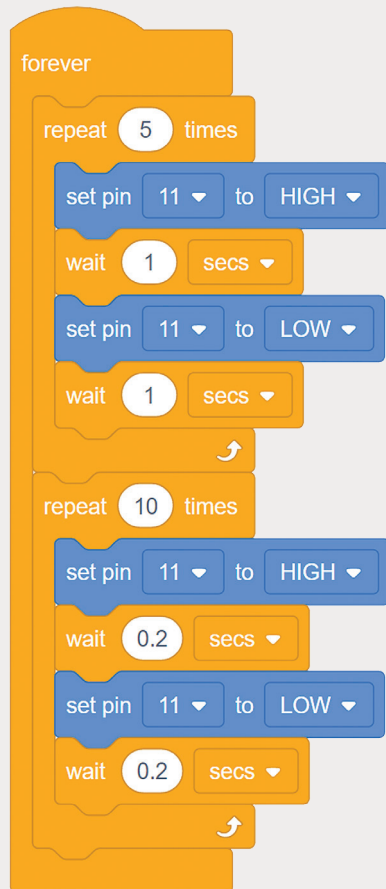


Figure 6.12: Connecting the LED

Programming the Arduino

The program will blink the LED light every 1 second 5 times, and then it will blink the LED light every 200 milliseconds 10 times.

Tinkercad blocks



C++

```
int counter;
int counter2;
void setup() {
  pinMode(11, OUTPUT);
}
void loop() {
  for (counter = 0; counter < 5; ++counter) {
    digitalWrite(11, HIGH);
    delay(1000); // Wait for 1000 millisecond(s)
    digitalWrite(11, LOW);
    delay(1000); // Wait for 1000 millisecond(s)
  }
  for (counter2 = 0; counter2 < 10; ++counter2) {
    digitalWrite(11, HIGH);
    delay(200); // Wait for 200 millisecond(s)
    digitalWrite(11, LOW);
    delay(200); // Wait for 200 millisecond(s)
  }
}
```



PIR Alarm Example

We will expand the previous project with a PIR alarm that will send a signal to light the LED 3 times in rapid succession.

- Components needed:
- 1 Arduino Uno R3
- 1 LED
- 1 PIR Sensor

To create the circuit:

- > Connect the **Cathode** of the **LED** to the **GND** of the Arduino and change the wire color to **black**. **1**
- > Connect the **Anode** of the **LED** to the **Digital Pin 11** of the Arduino and change the wire color to **green**. **2**
- > Connect the **Signal** of the **PIR Sensor** to the **Digital Pin 10** of the Arduino and change the wire color to **orange**. **3**
- > Connect the **Power** of the **PIR Sensor** to the **5V** of the Arduino and change the wire color to **red**. **4**
- > Connect the **Ground** of the **PIR Sensor** to the **GND** of the Arduino and change the wire color to **black**. **5**

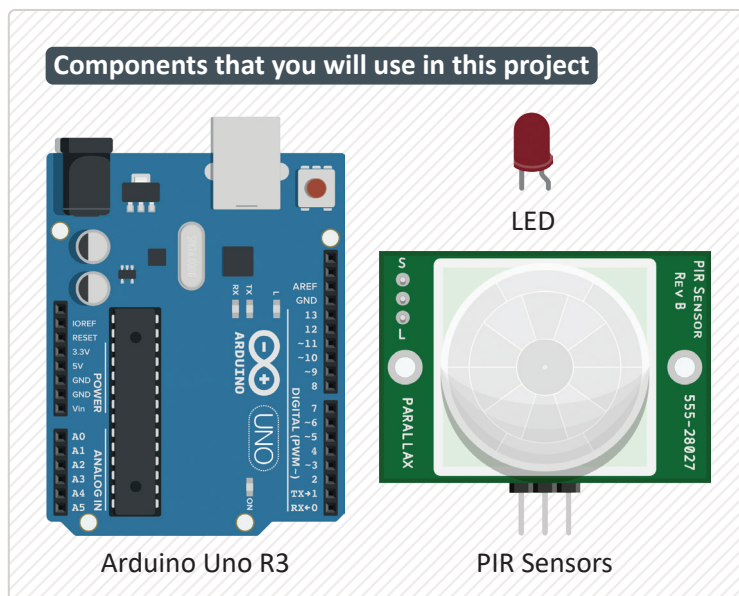


Figure 6.13: Project components

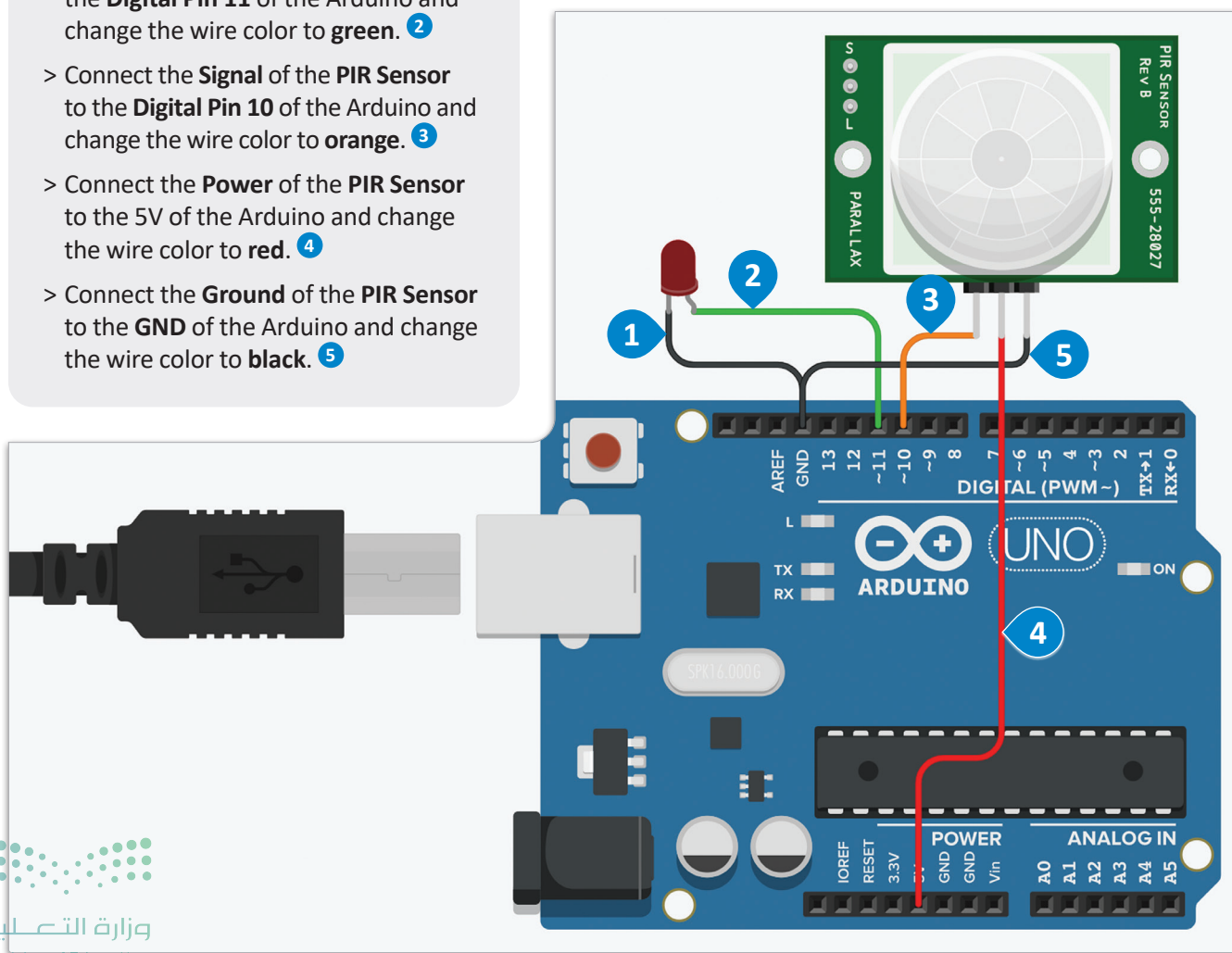
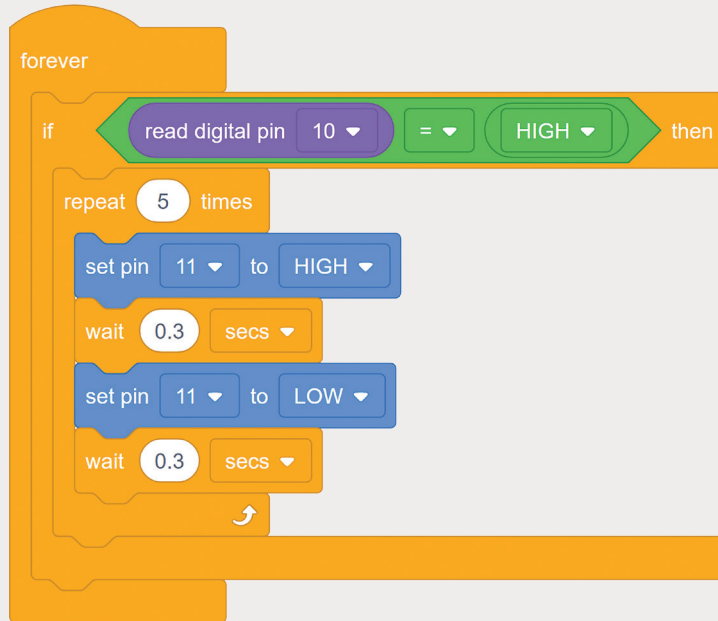


Figure 6.14: Connecting the circuit

Tinkercad blocks



C++

```
int counter;
void setup() {
  pinMode(10, INPUT);
  pinMode(11, OUTPUT);
}
void loop() {
  if (digitalRead(10) == HIGH) {
    for (counter = 0; counter < 5; ++counter) {
      digitalWrite(11, HIGH);
      delay(300); // Wait for 300 millisecond(s)
      digitalWrite(11, LOW);
      delay(300); // Wait for 300 millisecond(s)
    }
  }
}
```

Programming the Arduino

The program will check if the PIR sensor has detected an object in its Field of View. If it has detected an object, it will send a signal to blink the LED light 5 times in rapid succession.

Choose the programming mode of Text in the code editor to see the produced C++ text.



DC Motor Example

We will build a simple circuit to control a DC motor depending on the temperature of the environment. You will need the following components:

- 1 Arduino Uno R3
- 1 DC Motor
- 1 Temperature Sensor (TMP36)

To create the circuit:

- > Connect the **Power** pin of the **Temperature Sensor** to the **5V** of the Arduino and change the wire color to **red**. **1**
- > Connect the **Vout** pin of the **Temperature Sensor** to the **Analog pin A0** of the Arduino and change the wire color to **green**. **2**
- > Connect the **GND** pin of the **Temperature Sensor** to the **GND** of the Arduino and change the wire color to **black**. **3**
- > Connect **Terminal 1** of the **DC motor** to the **GND** of the Arduino and change the wire color to **black**. **4**
- > Connect **Terminal 2** of the **DC motor** to the **Digital pin 11** of the Arduino and change the wire color to **red**. **5**

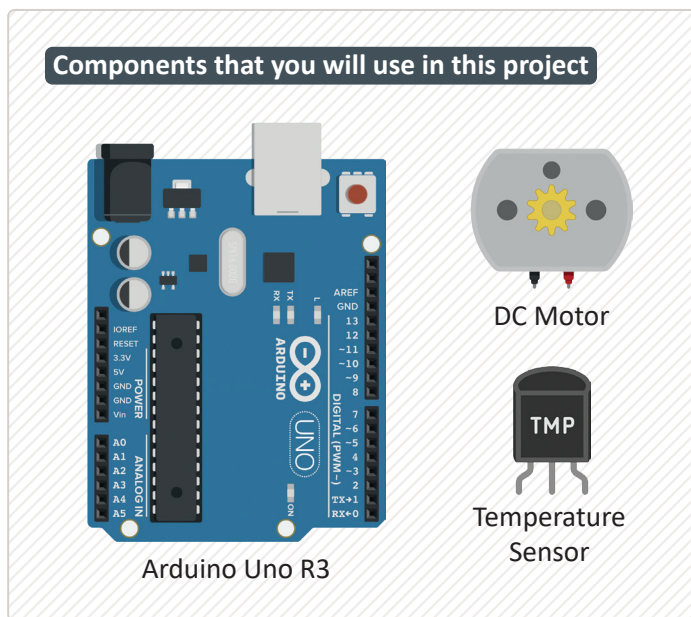


Figure 6.15: Project components

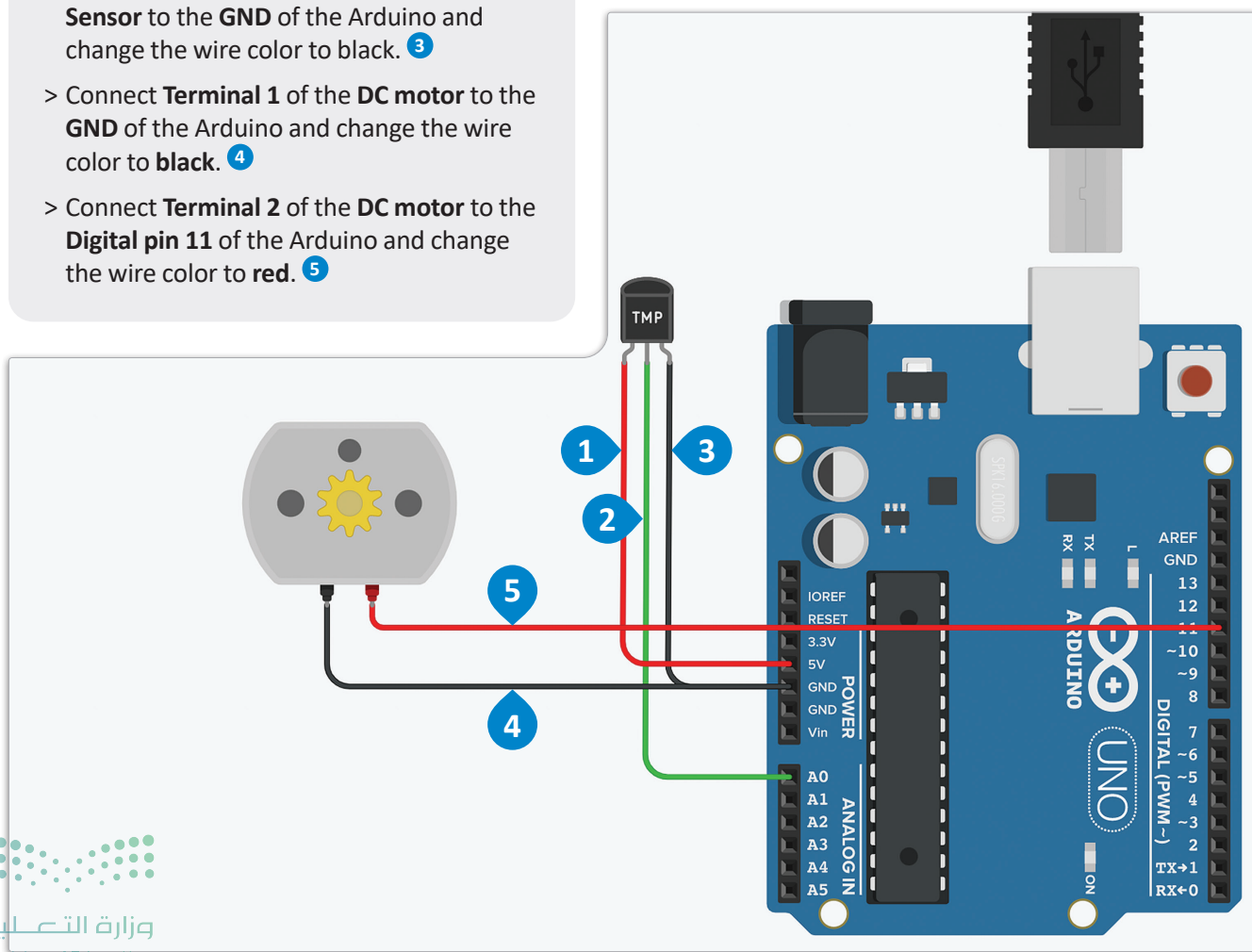
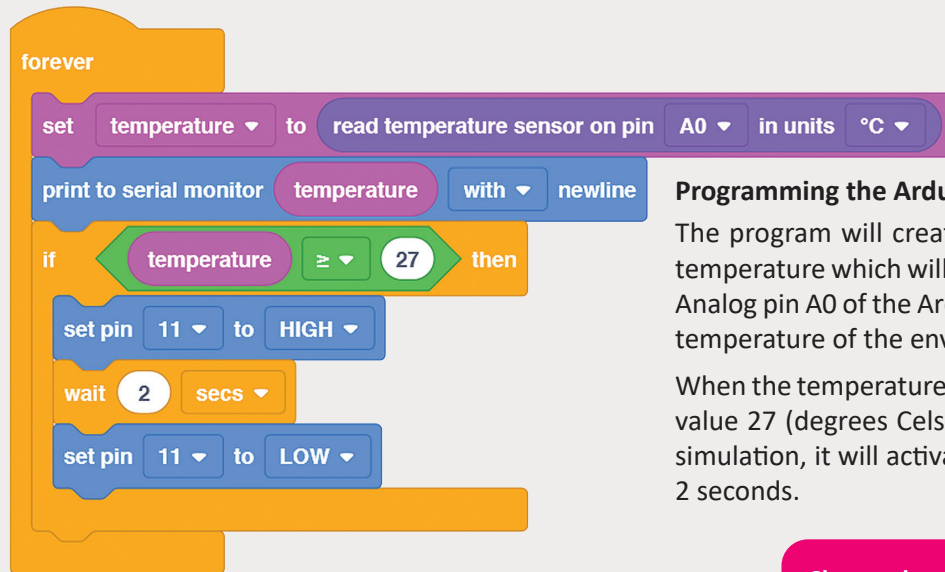


Figure 6.16: Connecting the circuit

Tinkercad blocks



Programming the Arduino

The program will create a variable named temperature which will be connected to the Analog pin A0 of the Arduino and record the temperature of the environment.

When the temperature variable reaches the value 27 (degrees Celsius) in the Tinkercad simulation, it will activate the DC motor for 2 seconds.

Choose the programming mode of Text in the code editor to see the produced C++ text.

C++

```

int temperature = 0;
void setup() {
  pinMode(A0, INPUT);
  Serial.begin(9600);
  pinMode(11, OUTPUT);
}
void loop() {
  temperature = (-40 + 0.488155 * (analogRead(A0) - 20));
  Serial.println(temperature);
  if (temperature >= 27) {
    digitalWrite(11, HIGH);
    delay(2000); // Wait for 2000 millisecond(s)
    digitalWrite(11, LOW);
  }
}
  
```

The Serial object is used for printing in the Serial Monitor. In the setup() function, the begin() function initializes the Serial Monitor so it can be used later. Then, the user can print values and messages to the Serial Monitor with the function print() or println(), with the latter also adding a newline at the end.



Exercises

- 1 Write a C++ function that takes two float data type arguments, an analog signal, and a multiplier. The function multiplies the signal and returns it.

- 2 Write a C++ sketch that reads an analog signal input from a pin that is a temperature reading in Fahrenheit. Create a function that converts this value to degrees Celsius and sends it to a pin as an analog output.



3 Find the syntax error and the logical error in the following code snippets:

```
void loop() {  
  for (counter = 0; counter < 5; --counter) {  
    digitalWrite(11, HIGH);  
    // Wait for 1000 millisecond(s)  
    delay("1000");  
    digitalWrite(11, LOW);  
    // Wait for 1000 millisecond(s)  
    delay("1000");  
  }  
}
```

Syntax Error:

Logical Error:

```
void loop() {  
  temperature = digitalRead(A0);  
  Serial.println(temperature);  
  if (temperature >= 270) {  
    digitalWrite(11, 1);  
    // Wait for 2000 millisecond(s)  
    delay(2000);  
    digitalWrite(11, 0);  
  }  
}
```

Syntax Error:

Logical Error:



4

Write a C++ sketch for the Arduino that uses the function from exercise 1 and reads an analog signal input. It then creates a for loop that uses the function from exercise 1 to amplify the original signal 5 times. Each time the signal is amplified, it is sent to a pin as an analog output.

5

Extend the Blinking LEDs example to accommodate another LED light of a different color that blinks every time the first LED light is off.



- 6 Extend the PIR alarm example to accommodate another PIR alarm and another LED light of another color. Each PIR alarm will be bound to an LED light which will blink depending on whether the PIR alarm detected an object.

- 7 Adjust the DC motor example to send an analog signal to the DC motor depending on the temperature that is detected by the Temperature sensor.



Lesson 3

Microcontroller Programming with C++

Link to digital lesson



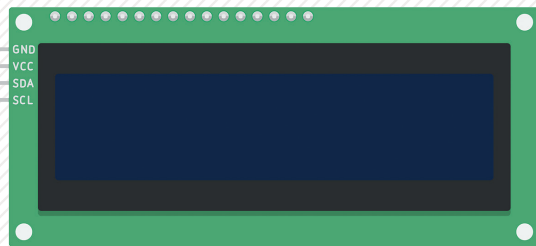
www.ien.edu.sa

Build a Smart Door Lock

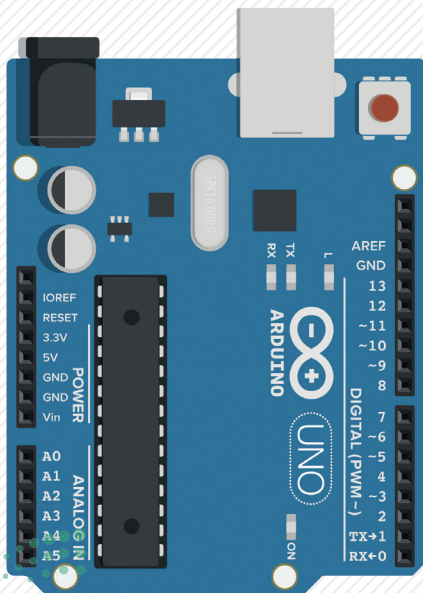
In this project the components you are going to use are:

- Arduino Uno R3
- Keypad 4x4
- LCD 16x2 (I2C)
- Micro Servo

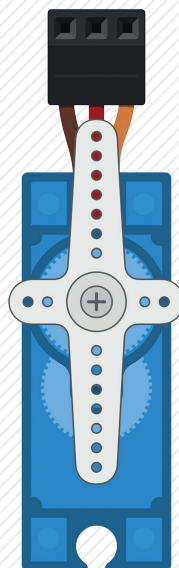
Components that you will use in this project



LCD 16x2 (I2C)



Arduino Uno R3



Micro Servo



Keypad 4x4

You will start by adding the keypad from the input category in the components and connecting it to the Arduino.

To connect the Keypad:

- > Find the **Keypad 4x4** component from the **Input** category in the components list and drag and drop it in the **workplane**. ❶
- > Connect the **Row 1** of the **Keypad** to the **Digital Pin 9** of the Arduino. ❷
- > Connect the **Row 2** of the **Keypad** to the **Digital Pin 8** of the Arduino. ❸
- > Connect the **Row 3** of the **Keypad** to the **Digital Pin 7** of the Arduino. ❹
- > Connect the **Row 4** of the **Keypad** to the **Digital Pin 6** of the Arduino. ❺
- > Connect the **Column 1** of the **Keypad** to the **Digital Pin 5** of the Arduino. ❻
- > Connect the **Column 2** of the **Keypad** to the **Digital Pin 4** of the Arduino. ❼
- > Connect the **Column 3** of the **Keypad** to the **Digital Pin 3** of the Arduino. ❽
- > Connect the **Column 4** of the **Keypad** to the **Digital Pin 2** of the Arduino. ❾
- > Change all wires color to **green**.

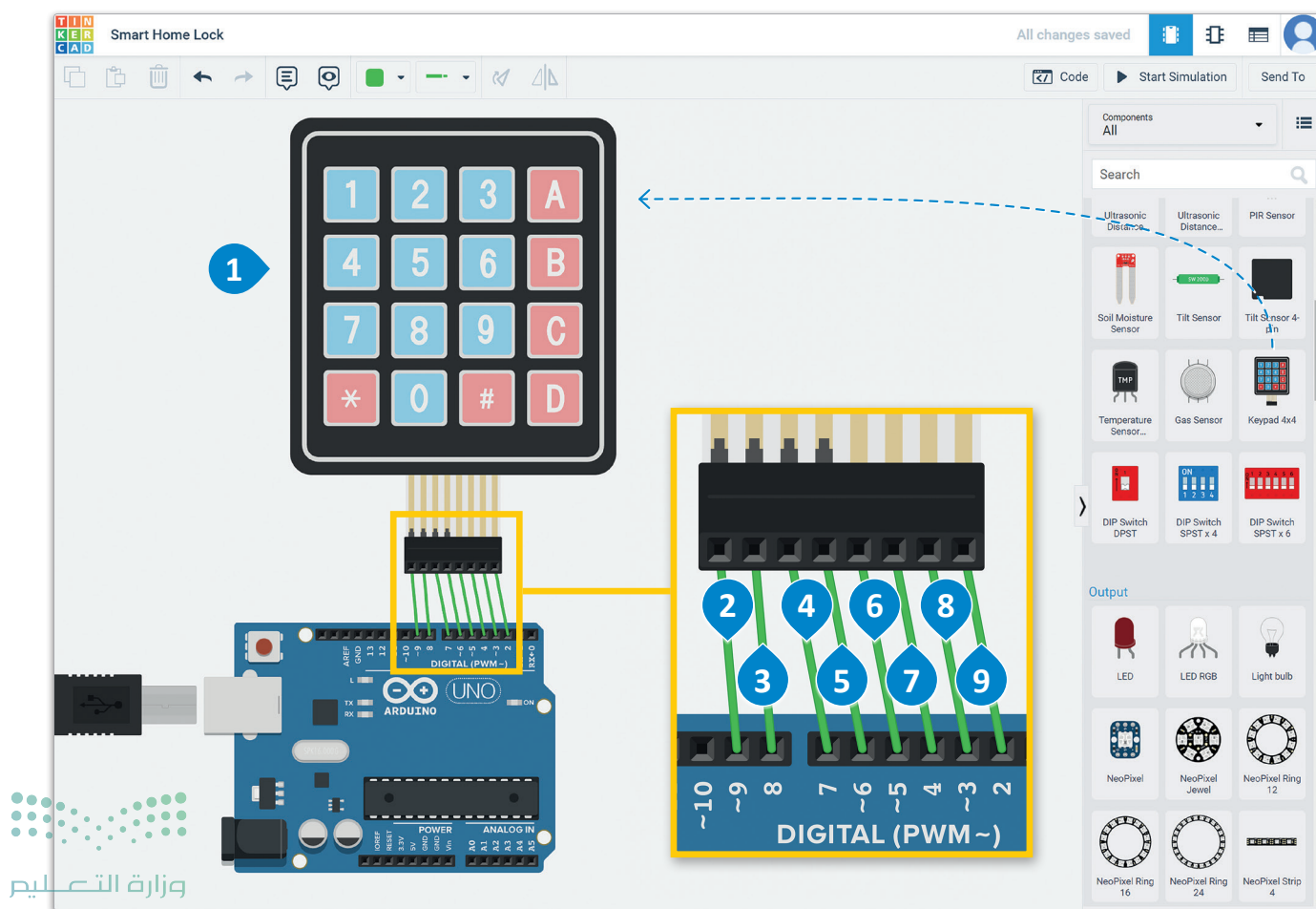


Figure 6.18: Connecting the Keypad

Now, find the LCD display from the output category in the components and connect it to the Arduino.

To connect the LCD display:

- > Find the **LCD 16x2 (I2C)** component from the **Output** category in the components list and drag and drop it in the **workplane**. **1**
- > Connect the **Ground** of the **LCD** to the **GND** of the Arduino and change the wire color to **black**. **2**
- > Connect the **Power** of the **LCD** to the **5V** of the Arduino and change the wire color to **red**. **3**
- > Connect the **SDA** of the **LCD** to the **SDA** of the Arduino and change the wire color to **green**. **4**
- > Connect the **SCL** of the **LCD** to the **SCL** of the Arduino and change the wire color to **yellow**. **5**

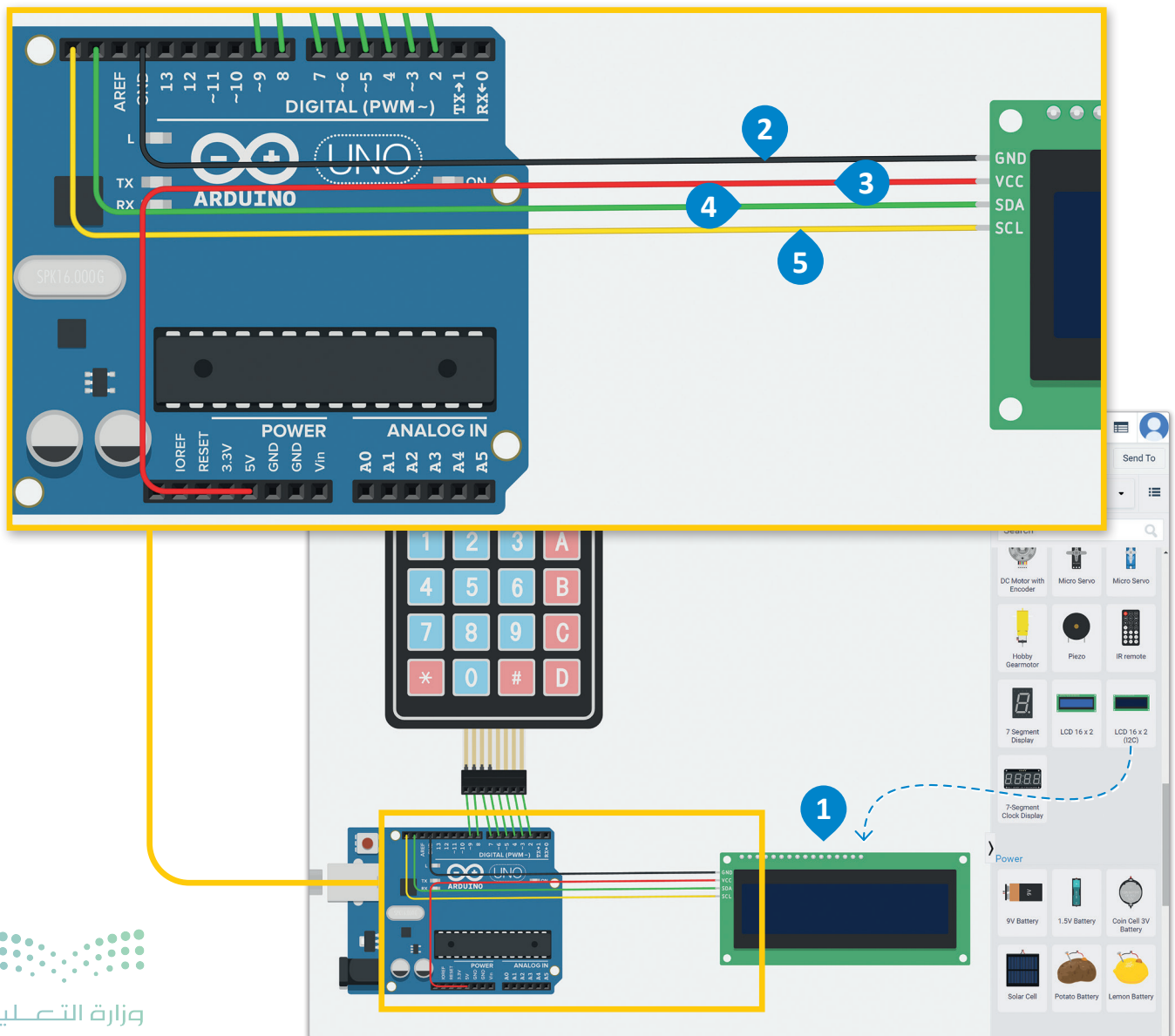


Figure 6.19: Connecting the LCD display

Finally, you will wire the servo motor.

Find the **Micro Servo** from the output category in the components and connect it to the Arduino.

To connect the Micro Servo:

- > Find the **Micro Servo** component from the **Output** category in the components list and drag and drop it in the **workplane**. ①
- > Connect the **Ground** of the **Micro Servo** to the **GND** of the Arduino and change the wire color to **black**. ②
- > Connect the **Power** of the **Micro Servo** to the **5V** of the Arduino and change the wire color to **red**. ③
- > Connect the **Signal** of the **Micro Servo** to the **Digital Pin 11** of the Arduino and change the wire color to **orange**. ④

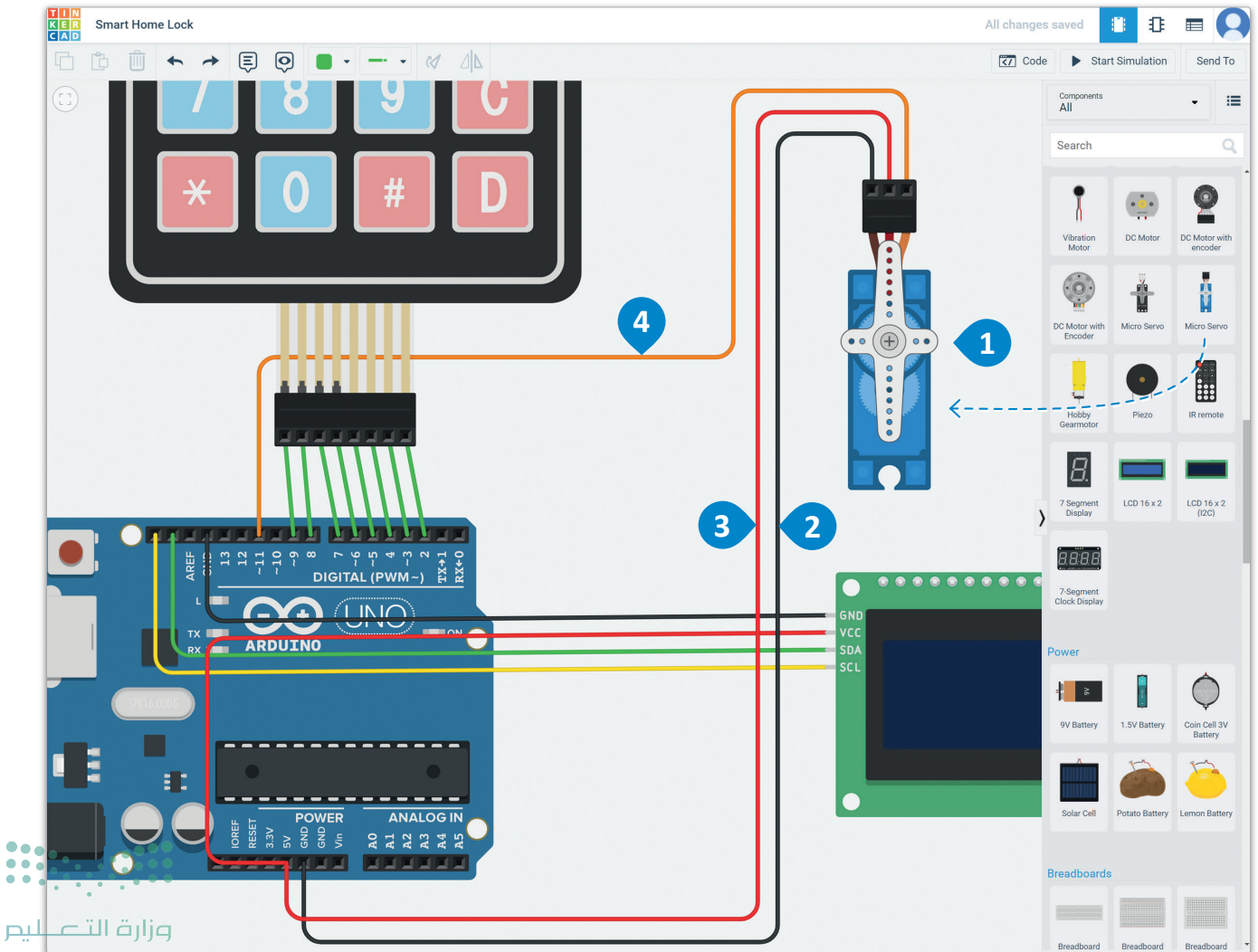


Figure 6.20: Connecting the Micro Servo

Include the Libraries

Apart from the Arduino controller, to use the rest of the components and program their logic in C++ you first need to include their libraries in the code section of the Tinkercad platform. These libraries provide many functions specific to each component.

To include a library in C++ you need to type:

```
#include <library name>
```

For the current project, you need the following libraries.

For the LCD panel

```
#include <Adafruit_LiquidCrystal.h>
```

for the keypad

```
#include <Keypad.h>
```

for the servo motor

```
#include <Servo.h>
```

Create the Objects

After including the necessary libraries, you need to create some objects and initialize some parameters. The objects that you need to create are

- an LCD object,
- a servo object,
- a keypad object.

When creating an object (or an instance) of a class, sometimes you need to provide some arguments to the constructor of this object. A constructor is a special class method that is called when an object is created, and its functionality is to initialize the object's parameters.



Micro Servo object

To create a **servo motor** object:

```
Servo servo;
```

Where "**Servo**" is the object type and "**servo**" is the actual object that you use in the code. You don't need to provide any initialization parameters.

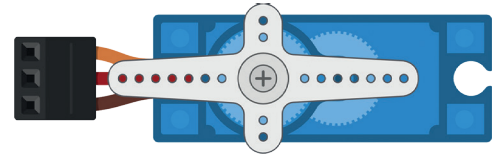


Figure 6.21:
Micro Servo (Tinkercad object)

LCD display object

To create an **LCD display** object:

```
Adafruit_LiquidCrystal lcd(0);
```

With this command, you initialize an object of type **Adafruit_LiquidCrystal** and pass its initial Arduino address (which is 0 by default) as an argument to the constructor of the class.

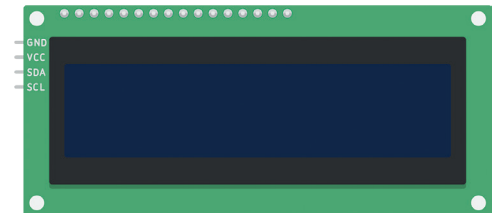


Figure 6.22: LCD display (Tinkercad object)

Keypad object

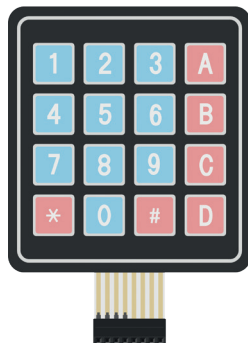


Figure 6.23:
Keypad (Tinkercad object)

The creation and initialization of the Keypad object needs some setup code. At first you need to specify the number of rows and columns the **Keypad** has. You do this with the commands:

```
const byte numRows = 4; // number of rows on the keypad  
const byte numCols = 4; // number of columns on the keypad
```

Here, you specify that the number of rows (numRows) is of the type "const byte" and its value is 4. The same applies to numCols.

Then you need to define the key pressed according to the row and column exactly as it appears on the keypad. The way to do this is:

```
// keymap defines the key pressed according to the rows and columns just as
    they appear on the keypad

char keymap[numRows][numCols] =
    {
        {'1', '2', '3', 'A'},
        {'4', '5', '6', 'B'},
        {'7', '8', '9', 'C'},
        {'*', '0', '#', 'D'}
    };
```

Here, you create the keymap array with the numRows and numCols you defined earlier and explicitly define the keys that are on the keypad.

After that, you need to setup the keypad connections to the Arduino terminals. You do this by defining two variables of type byte:

```
// Code that shows the the keypad connections to the arduino terminals

byte rowPins[numRows] = {9,8,7,6}; //Rows 0 to 3
byte colPins[numCols] = {5,4,3,2}; //Columns 0 to 3
```

The last step is to define a Keypad object by calling its constructor and providing the necessary arguments.

```
// initializes an instance of the Keypad class

Keypad keypad = Keypad(makeKeymap(keymap), rowPins, colPins, numRows, numCols);
```

To conclude the setup code, you define a variable called password that will store the password of the door lock and it is an array of characters with length 4.

```
char password[4];
```



Break down the Code

At this point, the setup code is completed.

As we explained in Lesson 1, the Arduino controller executes the setup() function only once, when it is powered on and then executes the loop() function repeatedly.

Let us break this code down.

We use two servo functions from the Servo library.

servo.attach(11), where we attach the Servo variable to pin 11.

servo.write(0), which is used to write a value to the servo, in this case, it writes the value 0, controlling the shaft accordingly. On a standard servo, this will set the angle of the shaft (in degrees), moving the shaft to that orientation.

After that, we use three functions of the Adafruit_LiquidCrystal library. These are:

lcd.begin(col,row) which initializes the interface to the LCD screen and specifies the dimensions (width and height) of the display. begin() needs to be called before any other LCD library commands.

The arguments are:

- cols, which is the number of columns that the display has
- rows, which is the number of rows that the display has

and because the LCD screen that you will use is 16x2 you give as arguments col=16 and row=2, hence lcd.begin(16,2);

The last piece of code in the setup() function is a for loop which stores the 4-character password the user types on the Keypad, in the variable password[4].

To do so the Keypad library function:

The code of the setup() function is:

```
void setup()
{
    //servo setup
    servo.attach(11);
    servo.write(0);

    //lcd setup and password set
    lcd.begin(16, 2);
    lcd.setCursor(0, 0);
    lcd.print("Set 4 character");
    lcd.setCursor(0, 1);
    lcd.print("password:");

    for(int i = 0; i < 4; i++) {
        password[i] = keypad.waitForKey();
    }
}
```

The next function is:

lcd.setCursor(col,row) which sets the location at which subsequent text written to the LCD will be displayed. So, to display the phrase "Set 4 character password" you need both rows of the lcd screen. On the first row, the phrase "Set 4 character" will be displayed and on the second row the phrase "password" will be displayed. That's why before displaying the first phrase you call the function as **lcd.setCursor(0, 0)**; and to display the second phrase you call the function as **lcd.setCursor(0, 1)**;

keypad.waitForKey() is called. This function will get the key that was pressed and store it in the password array.

Now for the main functionality of this project, the loop() function will be getting called repeatedly.

The code of the loop() function is:

```
void loop()
{
    // clear the screen and display the new message
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("Enter password:");

    bool correctPass = true;
    char buttonPressed;

    // this code checks each button pressed against the corresponding password
    // digit
    // e.g. it will check the 1st button pressed against the first digit of the
    // password and so on
    for (int i = 0; i < 4; i++) {
        buttonPressed = keypad.waitForKey();
        if(password[i] != buttonPressed){
            correctPass = false;
        }
        lcd.setCursor(i, 1);
        lcd.print(buttonPressed);
    }

    delay(1000);

    //this code will be executed if the password is correct
    if (correctPass) {
        // clear the lcd screen
        lcd.clear();
    }
}
```



```

    // set the cursor to the beginning of the 1st line
    lcd.setCursor(0, 0);
    lcd.print("Correct password!");
    // set the cursor to the beginning of the 2nd line
    lcd.setCursor(0, 1);
    lcd.print("Unlocking...");
    // write the angle by which the servo will rotate
    servo.write(180);
    // wait 5 sec and then rotate the servo to its original angle
    delay(5000);
    servo.write(0);
}
else {
    // this code will be executed if the password is wrong
    // clear the lcd screen
    lcd.clear();
    // set the cursor at the 1st col,row
    lcd.setCursor(0, 0);
    // print the message
    lcd.print("Wrong password!");
}
}
}

```

Break this Code down

To start with, there is some code to clear the lcd screen and display a message asking for the password.

```

// clear the screen and display the new message
lcd.clear();
lcd.setCursor(0, 0);
lcd.print("Enter password:");

```

Then there is the code that takes the user-typed password and checks whether this code is correct or not. The way it does this is by comparing the buttons pressed one by one sequentially against the digit of the password that is in the same position.

For example, let us say the password set in the beginning is "5456" and the user types the password "5453". Since every key that the user presses will be compared to the corresponding password key, what will happen is:

5 is compared to 5 (they are the same, so no problem so far)

4 is compared to 4 (still the same, still no problem)

5 is compared to 5 (still no problem)

3 is compared to 6 (they are not the same, so the password pressed is NOT correct).

Whenever the code compares two keys that are not the same it should update a variable with the information that the password is not correct. It does not matter if the wrong key occurs at the first digit, last digit or anywhere in between. The complete password will be wrong. So, to store this information you can use a boolean variable that will be initialized as true and whenever a wrong key occurs, its value will be set to false. After the comparison is done, you can check the value of this variable and if that value is true it means the user typed the correct password. If that value is false, it means the user typed a wrong password.

The functionality we just described is performed by this piece of code:

```
bool correctPass = true;
char buttonPressed;

// this code checks each button pressed against the corresponding password
// digit
// e.g. it will check the 1st button pressed against the first digit of the
// password and so on
for (int i = 0; i < 4; i++) {
    buttonPressed = keypad.waitForKey();
    if(password[i] != buttonPressed){
        correctPass = false;
    }
    lcd.setCursor(i, 1);
    lcd.print(buttonPressed);
}
```



What is left now, is to unlock the door (rotate the servo) if the password typed was correct and lock it again after a period of time, or to display a message saying the password was wrong.

This functionality is performed by the code:

```
// this code will be executed if the password is correct
if(correctPass){
    // clear the lcd screen
    lcd.clear();
    // set the cursor to the beginning of the 1st line
    lcd.setCursor(0, 0);
    lcd.print("Correct password!");
    // set the cursor to the beginning of the 2nd line
    lcd.setCursor(0, 1);
    lcd.print("Unlocking...");
    // write the angle by which the servo will rotate
    servo.write(180);
    // wait 5 sec and then rotate the servo to its original angle
    delay(5000);
    servo.write(0);
}
else {
    // this code will be executed if the password is wrong
    // clear the lcd screen
    lcd.clear();
    // set the cursor at the 1st col,row
    lcd.setCursor(0, 0);
    // print the message
    lcd.print("Wrong password!");
}
```



Finally, the complete code for the door lock project is:

Complete Code

```
// C++ code
//
#include <Adafruit_LiquidCrystal.h>
#include <Keypad.h>
#include <Servo.h>

Adafruit_LiquidCrystal lcd(0);
Servo servo;

const byte numRows = 4; //number of rows on the keypad
const byte numCols = 4; //number of columns on the keypad

// keymap defines the key pressed according to the rows and columns just as they
// appear on the //keypad
char keymap[numRows][numCols] =
{
  {'1', '2', '3', 'A'},
  {'4', '5', '6', 'B'},
  {'7', '8', '9', 'C'},
  {'*', '0', '#', 'D'}
};

// Code that shows the the keypad connections to the arduino terminals
byte rowPins[numRows] = {9,8,7,6}; //Rows 0 to 3
byte colPins[numCols] = {5,4,3,2}; //Columns 0 to 3

// initializes an instance of the Keypad class
Keypad keypad = Keypad(makeKeymap(keymap), rowPins, colPins, numRows, numCols);
```

Choose the programming mode
of Text in the code editor.



```

char password[4];

void setup()
{
    // servo setup
    servo.attach(11);
    servo.write(0);

    // lcd setup
    lcd.begin(16, 2);
    // lcd print 1st line
    lcd.setCursor(0, 0);
    lcd.print("Set 4 character");
    // lcd print 2nd line
    lcd.setCursor(0, 1);
    lcd.print("password:");

    // gets and stores the password
    for(int i = 0; i < 4; i++){
        password[i] = keypad.waitForKey();
    }
}

void loop() {
    // clear the screen and display the new message
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("Enter password:");

    bool correctPass = true;
    char buttonPressed;

```




```

// this code checks each button pressed against the corresponding password digit
// e.g. it will check the 1st button pressed against the first digit of the
password and so on
for(int i = 0; i < 4; i++) {
    buttonPressed = keypad.waitForKey();
    if(password[i] != buttonPressed) {
        correctPass = false;
    }
    lcd.setCursor(i, 1);
    lcd.print(buttonPressed);
}

delay(1000);

//this code will be executed if the password is correct
if (correctPass){
    // clear the lcd screen
    lcd.clear();
    // set the cursor to the beginning of the 1st line
    lcd.setCursor(0, 0);
    lcd.print("Correct password!");
    // set the cursor to the beginning of the 2nd line
    lcd.setCursor(0, 1);
    lcd.print("Unlocking...");
    // write the angle by which the servo will rotate
    servo.write(180);
    // wait 5 sec and then rotate the servo to its original angle
    delay(5000);
    servo.write(0);
}
else {
    // this code will be executed if the password is wrong

```

```
// clear the lcd screen  
lcd.clear();  
// set the cursor at the 1st col,row  
lcd.setCursor(0, 0);  
// print the message  
lcd.print("Wrong password!");  
}  
}
```

After you have completed the program,
click on the Start Simulation button.



Exercises

- 1 Create a circuit in Tinkercad that is connected to a temperature sensor and an LCD display. Then program it with C++ to display the temperature reading on the LCD display.

- 2 Create a circuit in Tinkercad that is connected to a 4x4 keypad and an LCD display. Then program it with C++ to display the pressed characters on the LCD display.



- 3 Create a circuit in Tinkercad that is connected to a 4x4 keypad and two LED lights, one RED and one green. The user will set a password and then will try to use it. If the input is right, the green light will light up, and if it is wrong, the red light will blink repeatedly.

- 4 Create a circuit in Tinkercard that is connected to a soil moisture sensor and servomotor. Then program it with C++ to turn the servomotor when the soil moisture reaches a certain dryness.



Project

A Smart security system is only one part of a complete IoT smart home system. There are other IoT applications for homes, but one of the most important is temperature regulation.

In this project you will extend the circuit and the code of the smart door lock project to accommodate more electronic components to control the temperature in the home.

1

The two main environmental readings that must be monitored are the temperature and the time of day. The temperature will be monitored by a temperature sensor and the time of day by a phototransistor that indicates the light levels outside of the residence.

2

Connect a DC motor to the circuit which will represent a thermostat and another LCD display. The LCD display will show the current temperature in degrees Celsius. The DC motor will be activated by an analog signal depending on the readings from the environment.

3

Create various tiers of temperature and light conditions which will send different analog values to the DC motor. Cooler environments need more output from the thermostat (DC motor). Build the circuit and program it with C++ to implement automatic temperature regulation.



Wrap up

Now you have learned to:

- > List the benefits and the risks of an IoT security system.
- > Cite examples of IoT devices used in smart security systems.
- > Use the basic commands in C++.
- > Program an Arduino microcontroller with C++.
- > Create an electronic circuit in Tinkercad and program it with C++.

KEY TERMS

**C++ Programming
Language**

Class

High Level

Programming Language

Keypad

LCD display

**Object Oriented
Programming**

Object

Smart Security



7. IoT Messaging

In this unit, you will learn about smart city applications and the (Message Queuing Telemetry Transport) MQTT communication protocol. You will also build an IoT application with an Arduino microcontroller and the MQTT protocol. Finally, you will perform data analysis operations on the built application.

Learning Objectives

In this unit, you will learn to:

- > List the layers of a Smart City Architecture.
- > Cite examples of a Smart City.
- > Describe the functionality of the MQTT protocol.
- > Classify the Quality of Service (QoS) of the MQTT protocol.
- > Use Python script to publish messages to MQTT X Client.
- > Create a JSON Data file to store reports.
- > Use a Jupyter Notebook to perform data analysis operations on the JSON data file.

Tools

- > Arduino IDE
- > JetBrains PyCharm
- > Autodesk Tinkercad Circuits
- > MQTT X Client





Smart Cities

The majority of cities began as modest urban centers. They were not originally planned to support a rapidly rising population. Typically, rapid expansion affects city infrastructure and services. Roads, bridges, and sewage systems frequently exceed their maximum capacity, making daily living difficult. Supplying essentials such as water and electricity while simultaneously lowering the carbon footprint is an immediate challenge.

As the global population increases, so do emissions and consumption. Population concentration in confined areas limits the ecosystem's ability to absorb pollutants. Increased emissions and waste contribute to the acceleration of climate change. Today, cities are accountable for 60–80 percent of the world's energy and greenhouse gas emissions and consume 60 percent of all drinkable water, losing as much as 20 percent to leakage. Optimizing resources, waste, and emissions with IoT technologies is a major priority for city authorities worldwide.

A Smart City IoT Architecture

The main challenge of smart IoT solutions is connecting multiple complex systems into one consolidated solution. There are many proposed smart city architectures. One of the most prominent divides a smart city IoT network into four layers. These are the **Street**, **City**, **Data Center**, and **Services** layers.

Data goes from devices at the street layer to the city network layer, where it is consolidated, normalized, and stored. The data center layer feeds information to the services layer, comprising city-service provider apps.

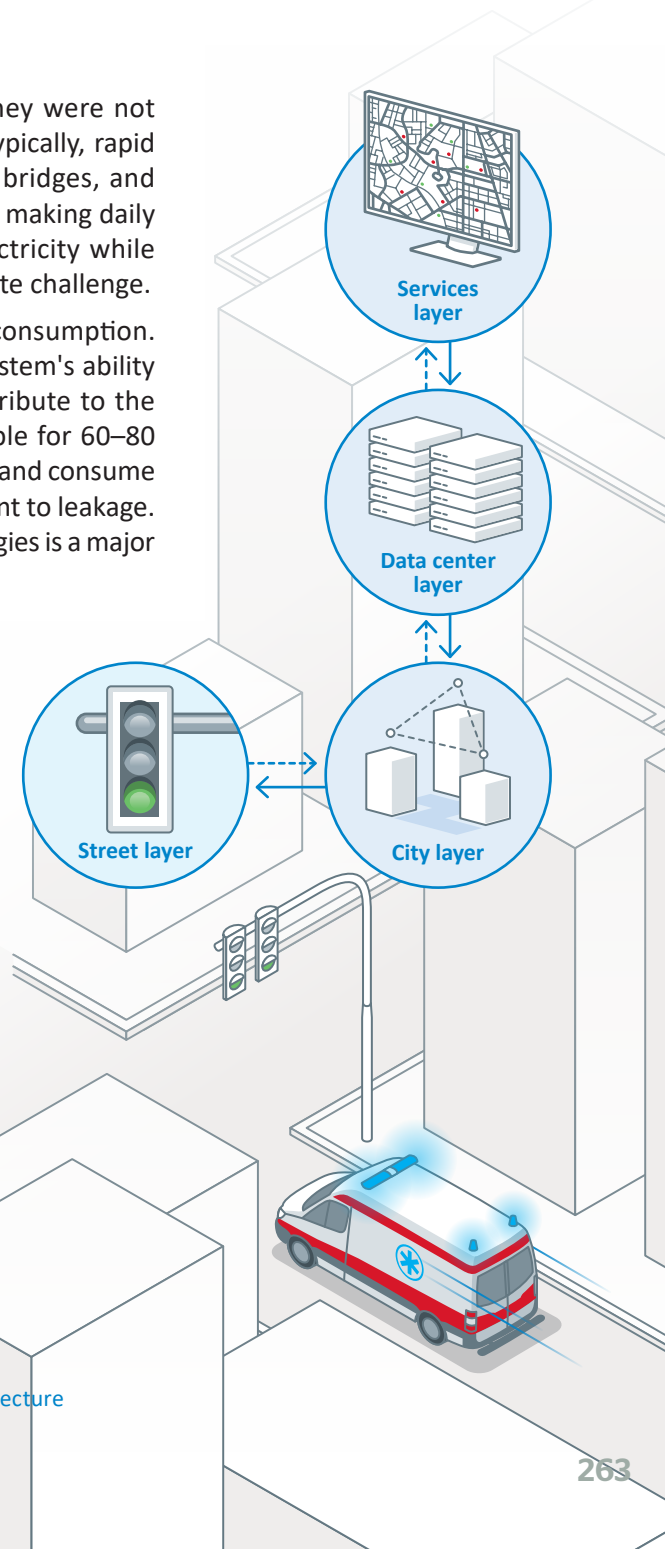


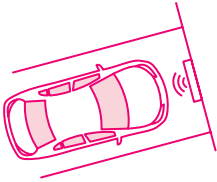
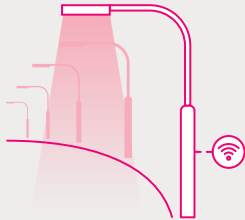
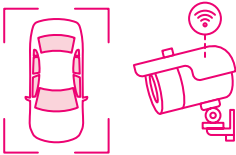
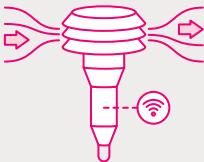
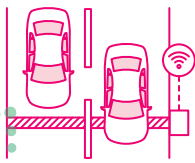
Figure 7.1: A Smart City IoT Architecture

Street Layer

The street layer consists of devices and sensors that gather data and operate depending on the overall solution's requirements and the networking components needed to collect and aggregate such data.

At the street layer, a range of devices is employed for various smart city use cases, such as:

Table 7.1: Street layer devices and sensors

Type	Description
 Magnetic sensor	A magnetic sensor may detect a parking event by monitoring changes in the magnetic field when a heavy metal object, such as a vehicle or a truck, approaches it.
 Lighting controller	A lighting controller can dim and brighten a light based on environmental variables and time.
 Video cameras	Video cameras paired with video analytics can recognize cars, faces, and traffic conditions for various traffic and security applications.
 Air quality sensor	An air quality sensor can detect and quantify quantities of gases and particulate matter to provide a hyper-localized view of pollution in a specific location.
 Counters	In order to provide traffic analytics, counters record the number of vehicles moving in the street or parked in a public parking area.

City Layer

The city layer can appear to be a straightforward transport layer between IoT edge devices and the data center or Internet. Network routers and switches must be deployed at the city layer above the street layer to support the transfer of big data. The city layer must transmit data through many types of protocols for various IoT applications. However, some applications are sensitive to delays or packet losses. A lost packet may trigger an alert or create an incorrect status report. Therefore, the city layer must be resilient to ensure that a data packet sent from a sensor or gateway will always reach its destination.

Data Center Layer

The gathered sensor data is sent to a data center for processing and storage. Based on this data processing, important information and patterns will be identified and insights will be generated. For instance, a data center can give a city-wide perspective of the traffic and assist authorities in determining the demand for additional or fewer mass transport vehicles. The same traffic data may be used to automatically manage and synchronize the city's traffic light durations to reduce traffic congestion. Cloud and data storage services are crucial for developing any comprehensive IoT solution. This data can be stored in data centers owned by city authorities or private companies depending on the local legislation.

Services Layer

Ultimately, the actual value of IoT systems is determined by the services delivered to authorities and citizens. The processed data should be displayed following the particular demands of each data consumer, the distinct user experience requirements, and the different use cases. Buses and other public transit systems can be redirected if needed to avoid known congestion locations. The number of subway trains can be dynamically increased in response to an increase in traffic congestion, anticipating the decisions of commuters to choose public transit instead of their automobiles on days with heavy traffic.

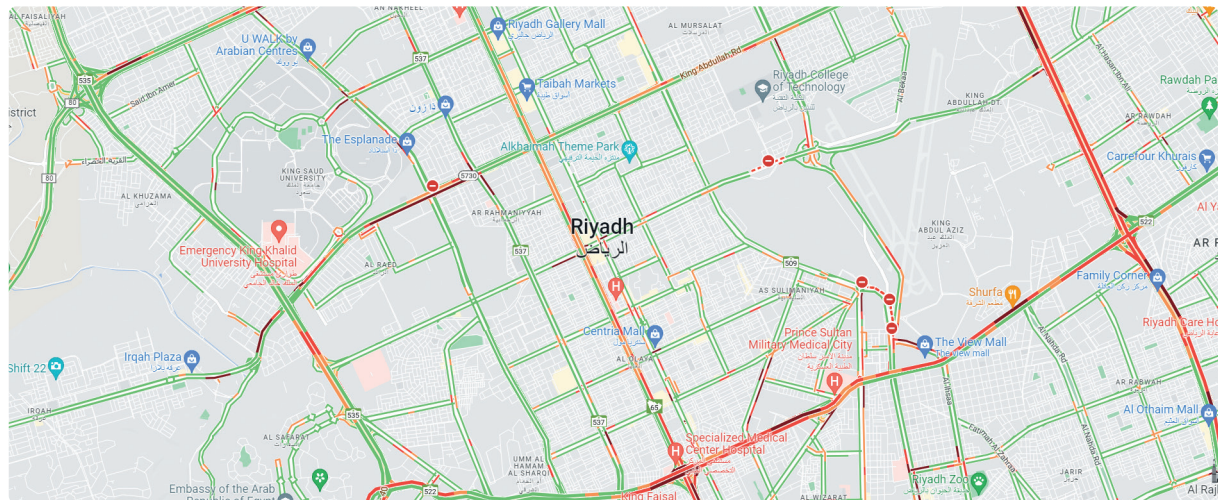


Figure 7.2: Traffic update in real time

Example

The Ministry of Municipal and Rural Affairs and Housing plans to implement more than 50 IoT smart city projects by 2030, including smart traffic management and parking, environmental preservation and trash disposal systems, smart housing, and land management systems. Improving citizens' quality of life, financial sustainability, and service quality are the main goals.

Smart City Applications

Connected Street Lighting

Street lighting is one of the most expensive metropolitan utilities, accounting for up to 40 percent of the overall utility bill. Commonly, cities search for ways to cut lighting costs while simultaneously improving operating efficiencies and lowering initial expenditure. Installing a smart street lighting system can result in substantial energy savings and can be leveraged to deliver new services. Light-Emitting Diode (LED) technology is at the forefront of the move from conventional to intelligent street lighting. LEDs with low power consumption are ideally suited for smart solution applications. For instance, LED color or light intensity can be modified according to the conditions.



Figure 7.3: Connected Street Lighting

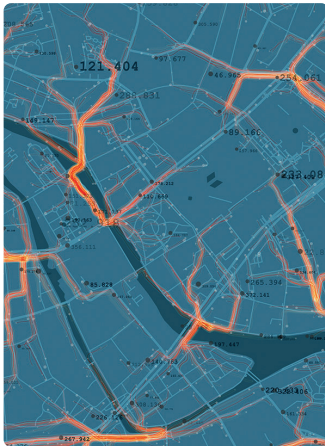


Figure 7.4: Smart Traffic Control

Smart Traffic Control

Traffic is one of the most well-known problems in every city. It is a major contributor to global pollution and loss of productivity. A smart city traffic solution would incorporate population counts, transit information, and vehicle counts on the road and forward the necessary data to traffic planners so they can take action. It is possible to enable traffic apps in cooperation with IoT sensors to control traffic and decrease congestion. Using historical data, urban planners may create more effective strategies to minimize traffic congestion. Consequences of heavy traffic waves include a rise in local accidents, which are often minor but increase general congestion. A common solution for stop-and-go traffic is regulating the standard flow speed based on vehicle density. An application that detects traffic density in real-time can regulate the length of the traffic light cycle to restrict or eliminate the wave effect by controlling the number of vehicles added to the flow on major roads.

Connected Environment

The majority of large cities monitor air quality. Costly and decades-old air quality monitoring stations are frequently used to collect data. These stations are quite precise in their readings, but their range is extremely limited. Thus, a metropolis is likely to have several blind spots without enough data to properly identify air-quality patterns. Considering the cost and size of air quality monitoring stations, communities cannot afford to acquire the necessary number of stations to provide reliable information on a localized level and track pollution flows as they move through the city over time.



Figure 7.5: Smart Air-quality Station



Figure 7.6: Roadside Unit (RSU)

Smart Safety Alerts

On the side of the roads, there is a Dedicated short-range communications (DSRC) communication unit which serves as a gateway between vehicle on-board unit OBUs and the communications infrastructure. A Roadside Unit (RSU) is a special wireless communicating device located on the roadside that provides connectivity and information support to passing vehicles, including safety warnings and traffic information.



A smart city always requires localized, real-time, distributed knowledge about air quality. For this data, smart cities require the following:

- Open-data systems that receive measurements of air quality from existing monitoring stations.
- IoT sensors that give the same level of precision as the air quality stations but are far less expensive.
- Environmental data visualization for authorities and citizens and storage of previous air quality data records to trace emissions through time and identify trends.

Example

The Line in the NEOM megacity in the KSA aims to incorporate the latest breakthrough smart city technologies to become the most technologically advanced urban environment.

NEOM will rely heavily on smart city IoT solutions to reach its goal of becoming a zero-emission city with no cars or traffic.

Message Queuing Telemetry Transport (MQTT)

Introduction to MQTT

During the end of the 1990s, engineers from IBM and Arcom searched for a dependable, lightweight, and cost-effective protocol to monitor and manage many sensors and their data from a central server location, as was customary in the oil and gas sectors. The outcome of their study was the development of the Message Queuing Telemetry Transport (MQTT) protocol that is now standardized by the Organization for the Advancement of Structured Information Standards (OASIS). The MQTT protocol is more commonly used in IoT applications than the HTTP protocol because it is easier to create complex architectures with devices that publish and receive data packets.

MQTT Basics

An MQTT client can be a "publisher" to send data to an MQTT server operating as a message server (message broker). The MQTT server receives the publishers' network connection and application messages. Additionally, it manages the subscription and unsubscription processes and delivers application data to MQTT clients serving as subscribers. Clients can subscribe to all or particular data from a publisher's information pool using MQTT. In this case, the MQTT client is called a "subscriber". The inclusion of a message broker in MQTT decouples the data transfer between publishers and subscribers. Publishers and subscribers are unaware of each other. The MQTT message broker guarantees that information may be delayed and stored in the event of network failure, which is an advantage of this decoupling. Due to this, publishers and subscribers are not required to be online simultaneously. Each client and server MQTT session consists of four phases: session establishment, authentication, data exchange, and session termination. Each client that connects to a server has a unique client ID that identifies the MQTT session between the two parties. The server treats each client individually when sending an application message to many clients. The drawbacks of the MQTT protocol are slower transmit cycles than HTTP, resource discovery and backup services must be implemented by the user, there is a lack of default security encryption and it is generally difficult to scale as the number of devices and brokers increases.

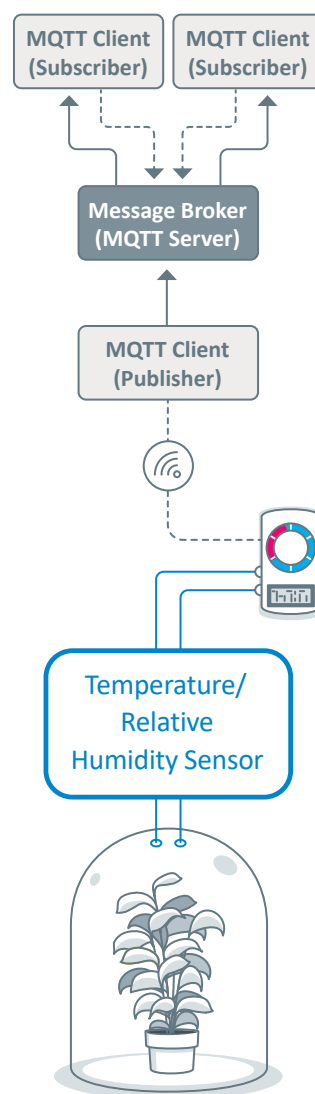
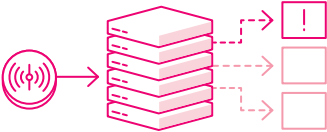
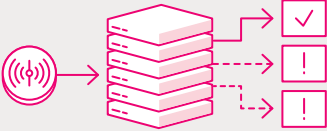
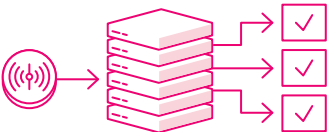


Figure 7.7: MQTT functionality

Quality of Service (QoS)

The MQTT protocol provides three degrees of service quality (QoS). QoS for MQTT is applied while exchanging application messages with publishers or subscribers. The delivery protocol primarily concerns application message delivery from a single sender to a single recipient.

The following table presents the three MQTT QoS levels:

Table 7.2: Quality of Service Levels	
Level	Description
 QoS 0: at most once • Doesn't survive failures • Never duplicated	This is an unacknowledged and best-effort data service known as "at most once" delivery. The publisher delivers a single message to a server, which relays it to each subscriber. The recipient receives no answer, and the sender does not retry to send the data. The recipient receives the message either once or not at all.
 QoS 1: at least once • Survives connection loss • Can be duplicated	This QoS level assures messages are sent at least once between the publisher and server, then between the server and subscribers. This level guarantees at least one delivery.
 QoS 2: exactly once • Survives connection loss • Never duplicated	This is the highest QoS level and is used when neither message loss nor duplication is acceptable. This QoS level has an extra cost since each packet includes an optional variable header with a packet identification. This level provides a "guaranteed service" named "exactly once" delivery. The number of retries is irrelevant as long as the message is sent precisely once.

Example

In smart cities, IoT objects can be attacked due to their centralized architecture. Using traditional security methods may not be adequate for the evolving IoT environment. In the KSA, blockchain IoT technologies will be developed in large cities to minimize central points of network failure with a distributed architecture. The NEOM megaproject will base its network on blockchain IoT technologies to provide secure and accessible network infrastructure for its citizens.

Exercises

1

Read the sentences and tick ✓ True or False.	True	False
1. Smart city technologies are being developed only to optimize traffic flow.	☐	☐
2. City layer network routers must be resilient against potential packet losses.	☐	☐
3. Data from the street layer is sent directly to the Data Center layer.	☐	☐
4. Data stored in the Data Center layer can be stored on the servers of private companies.	☐	☐
5. The services layer contains the applications that the residents of the city use.	☐	☐
6. Connected street lighting systems require LED lights exclusively.	☐	☐
7. Historical data cannot be used to forecast future traffic.	☐	☐
8. Connected environment solutions can be used to reduce city emissions.	☐	☐
9. The MQTT protocol was created to connect many sensors through a single service point.	☐	☐
10. On a basic connection with the MQTT protocol, the publisher and the subscriber acknowledge each other's presence	☐	☐

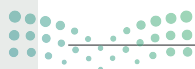
2

What is the primary driver behind smart city advancements? Present your ideas below.



3

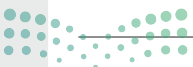
4



- 5 Describe how identical systems in the Data Center layer can be used in multiple applications. Present your ideas below.

- 6 Provide two examples of smart city applications and briefly describe them. Present your ideas below.

- 7 Describe briefly how the MQTT protocol works.



8

This image shows a blank sheet of white paper with horizontal ruling lines. The lines are evenly spaced and run across the width of the page. There are no margins, text, or other markings on the paper.

9



Lesson 2

Designing and Programming a Smart Waste IoT Device

Link to digital lesson



www.ien.edu.sa

Smart Waste Management

Due to overpopulation, very large quantities of waste and garbage are not collected and processed efficiently, leading to waste overflows in various locations. This problem occurs because there are garbage cans that overflow and are not cleared on time. With smart waste cans, there can be alert messages that notify waste collection vehicles. Also, with the appropriate data analysis, we can derive insights into how waste cans are filled to optimize the whole process further.

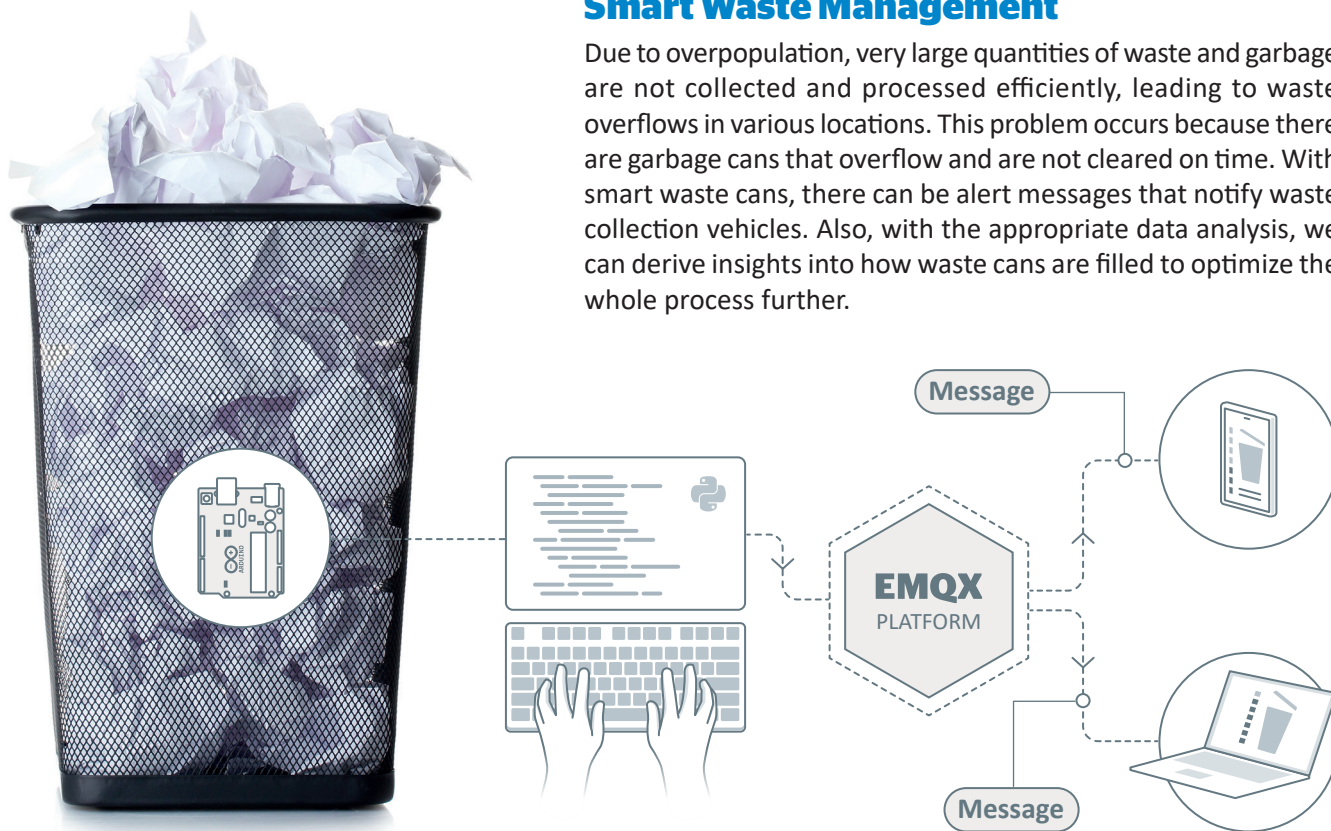


Figure 7.8: Smart Waste Management Arduino & MQTT project

In this lesson, you will build a prototype of a smart garbage can that counts how many actions on average are required to reach full capacity. A message is sent to an MQTT broker each time the can is used. When the can is full, another message is sent to the system's controller unit that generates reports for the can. For this project, you will use an Arduino microcontroller that represents the smart trash can, which you will program with the Firmata protocol and Python, and use the EMQX platform to distribute the messages.



EMQX is an open-source MQTT broker with a high-performance real-time message processing engine. It is used to support event streaming for an extremely large number of IoT devices.

Components & Tools for Project

Phototransistor

A Phototransistor is an electrical component that operates when exposed to light. When light falls upon the sensor, a proportionate amount of reverse current flows. Phototransistors are widely employed to detect and convert light pulses to electrical signals.



Phototransistor



Simulator component



Schematic symbol

Figure 7.9: Phototransistor

Tilt Sensor

A tilt sensor is used to measure a reference plane's tilt along multiple axes. Tilt sensors assess the tilting position relative to gravity and are employed in various applications. They make orientation or inclination detection simple.



Tilt Sensor



Simulator component



Schematic symbol

Figure 7.10: Tilt Sensor



Arduino Prototype

The Arduino microcontroller will monitor the state of the can and collect the action data to send through the Firmata protocol. The tilt sensor will be used to record each time the can is used, simulating the movement of the can's lid and the phototransistor as a light sensor, so that when it reaches a certain threshold, it will mean the can is full of garbage.

You will need the following components:

- 1 Arduino Uno R3
- 1 Breadboard Small
- 1 Phototransistor
- 1 Tilt Sensor
- 2 Resistors (1k Ω)

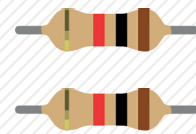
Components that you will use in this project



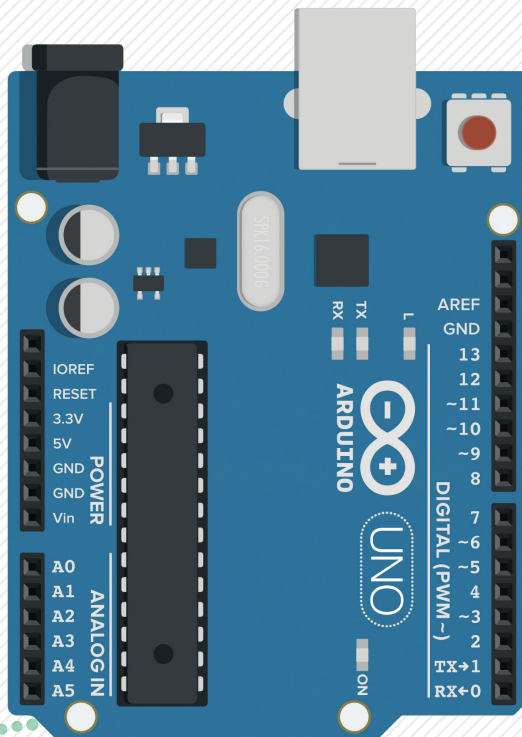
Phototransistor



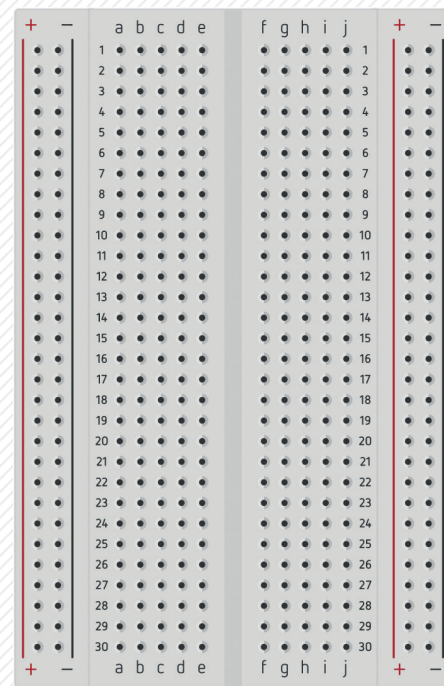
Tilt Sensor



Resistors



Arduino Uno R3

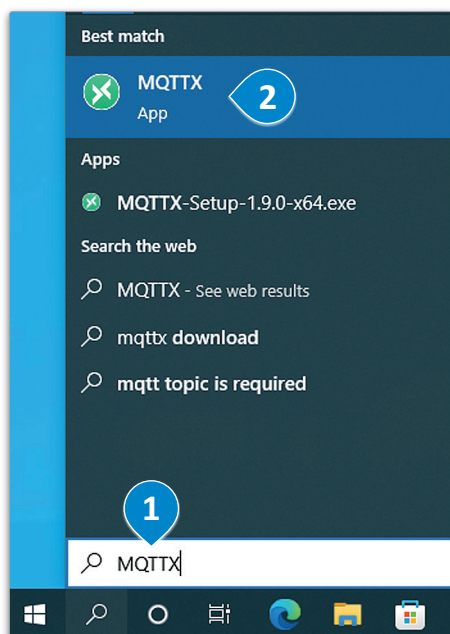


Breadboard Small

Connecting to the EMQX Public Broker

You will first need to install the **MQTTX** client desktop application and then test the connection with the EMQX public broker. To install the **MQTTX** client application go to the website: <https://mqttx.app/> and download the latest version.

Run the installer to complete the installation process. You will now open the client and create a new connection to the EMQX broker.



To set up the connection to the EMQX broker with MQTTX:

- > Click the Windows search button and type "MQTTX". **1**
- > Open the **MQTTX** client desktop app. **2**
- > Click **New Connection**, to create a new connection. **3**
- > Type a name for the connection e.g. **desktop_connection**. **4**
- > Click on the **Connect** button. **5**

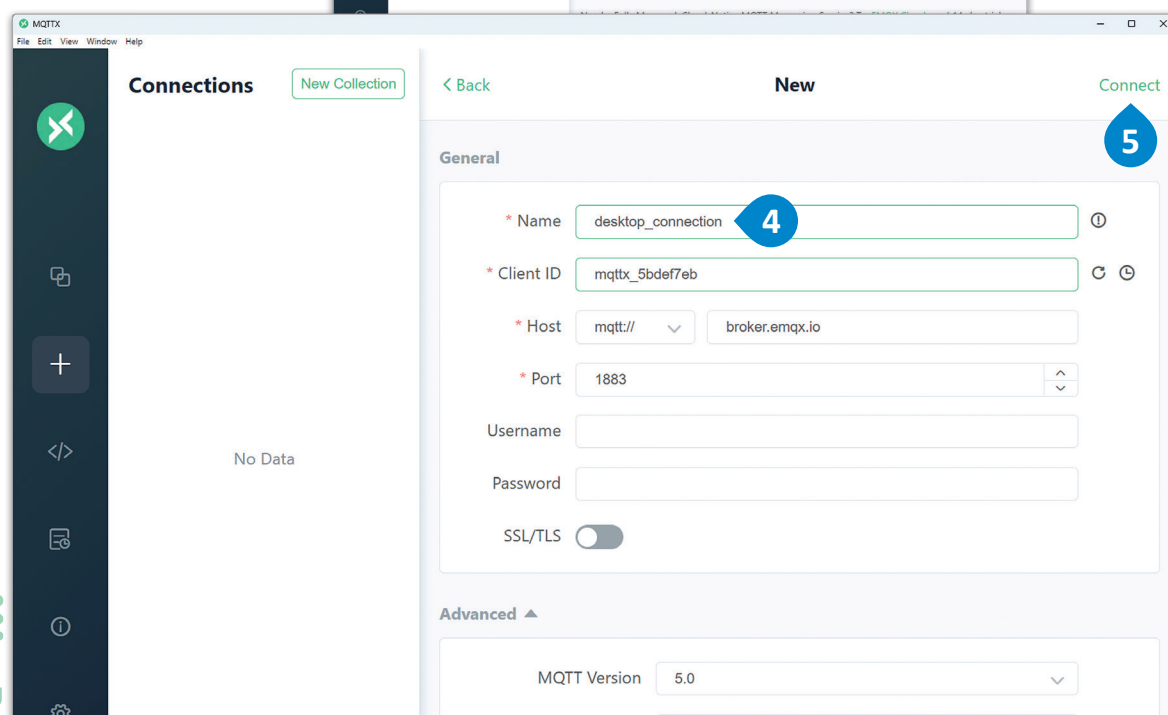
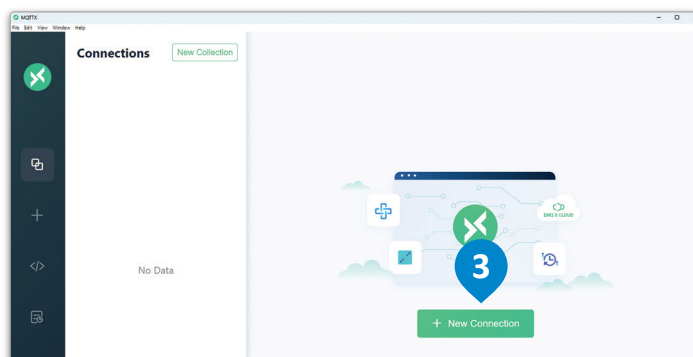


Figure 7.12: Setting up the connection to the EMQX broker with MQTTX

Arduino Circuit

You will begin building the Arduino circuit by putting the required components into the Tinkercad circuits workplane.

To load the components:

- > Find the **Arduino Uno R3** from the **components library** and drag and drop it into the **workplane**. **1**
- > Find the **Breadboard Small** from the **components library** and drag and drop it into the workplane. **2**
- > Find the **Ambient Light Sensor [Phototransistor]** from the **components library** and drag and drop it into the **workplane**. **3**
- > Find the **Tilt Sensor 4-pin** from the **components library** and drag and drop it into the **workplane**. **4**
- > Find the **Resistor** from the **components library** and drag and drop 2 of them into the **workplane**. **5**

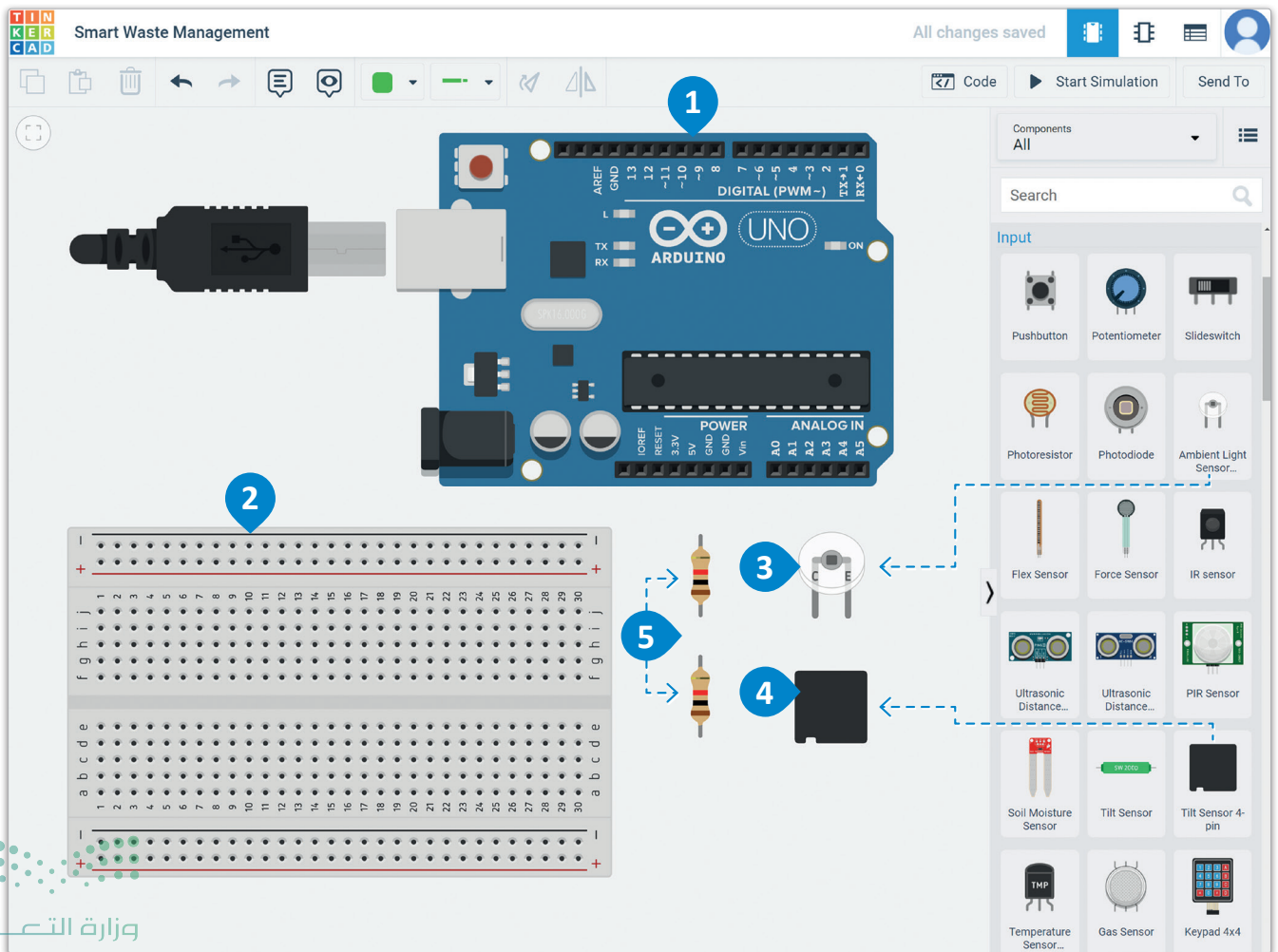


Figure 7.13: Loading the components of the circuit

To connect the Phototransistor:

- > Connect the **Emitter** end of the **Phototransistor** to **Analog Pin A0** of the Arduino and change the wire color to **yellow**. **1**
- > Connect **Terminal 2** of the one **Resistor** to the same row as the **Emitter** of the **Phototransistor** and connect **Terminal 1** of the **Resistor** to the **negative column** of the **Breadboard Small**. **2**
- > Connect the **Arduino UNO R3 5V pin** to the positive column of the Breadboard and change the wire color to **red**. **3**
- > Connect the **Arduino UNO R3 ground pin** to the negative column of the Breadboard and change the wire color to **black**. **4**
- > Connect the **Collector** end of the **Phototransistor** to the **positive column** of the **Breadboard Small**. **5**

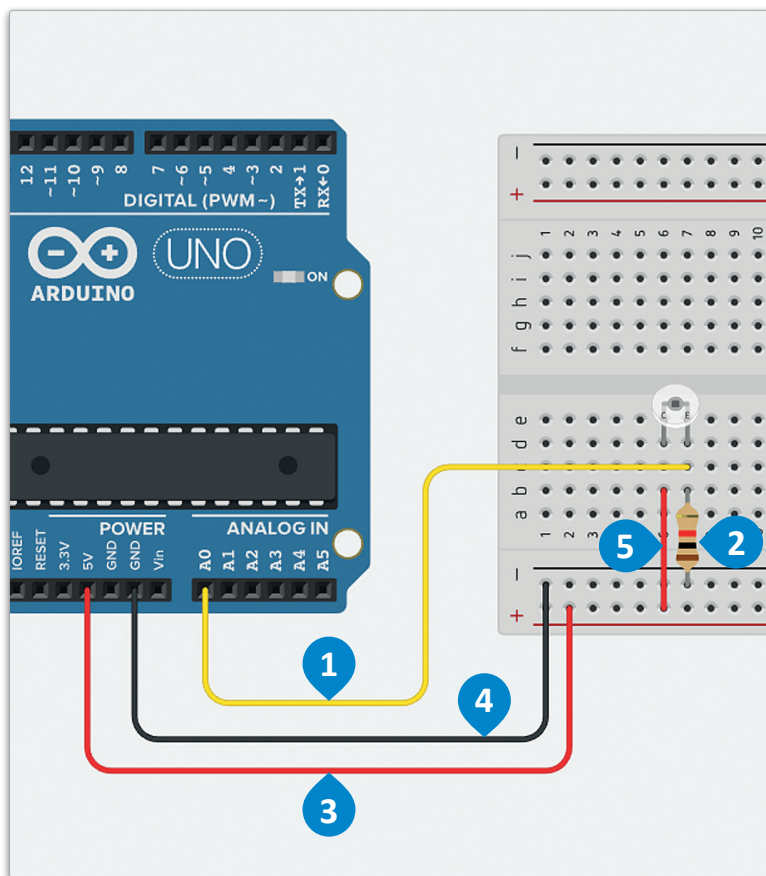


Figure 7.14: Connect the Phototransistor

To connect the Tilt Sensor:

- > Connect **Terminal 2** of the other **Resistor** to **Terminal 2** of the **Tilt Sensor**. **1**
- > Connect **Terminal 2** of the **Tilt Sensor** to **Digital Pin 3** of the Arduino and change the wire color to **green**. **2**
- > Connect **Terminal 4** of the **Tilt Sensor** to the positive column of the **Breadboard Small** and change the wire color to **red**. **3**
- > Connect **Terminal 1** of the **Resistor** to the **negative column** of the **Breadboard Small** and change the wire color to **black**. **4**

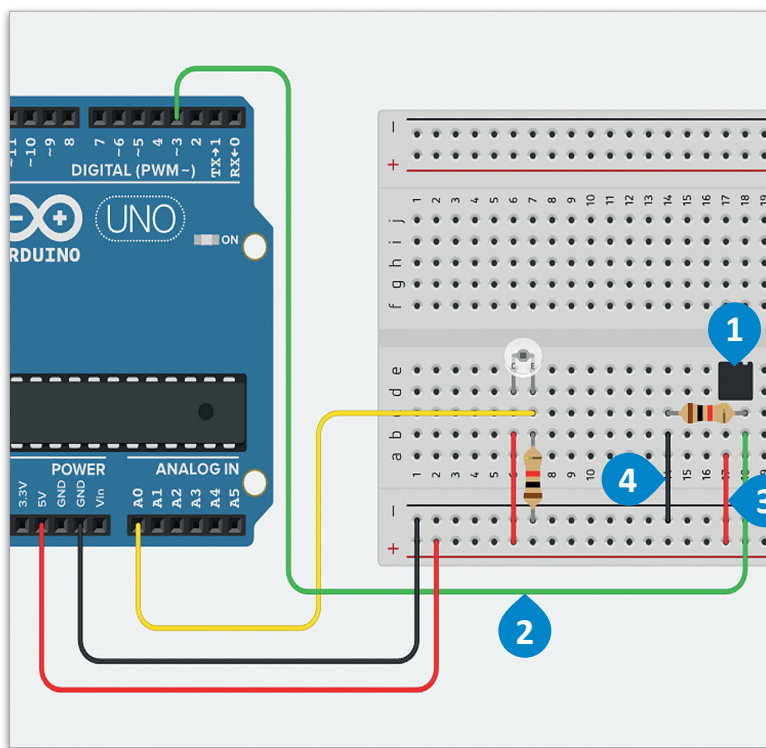
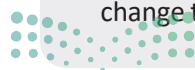


Figure 7.15: Connect the Tilt Sensor



Complete Circuit

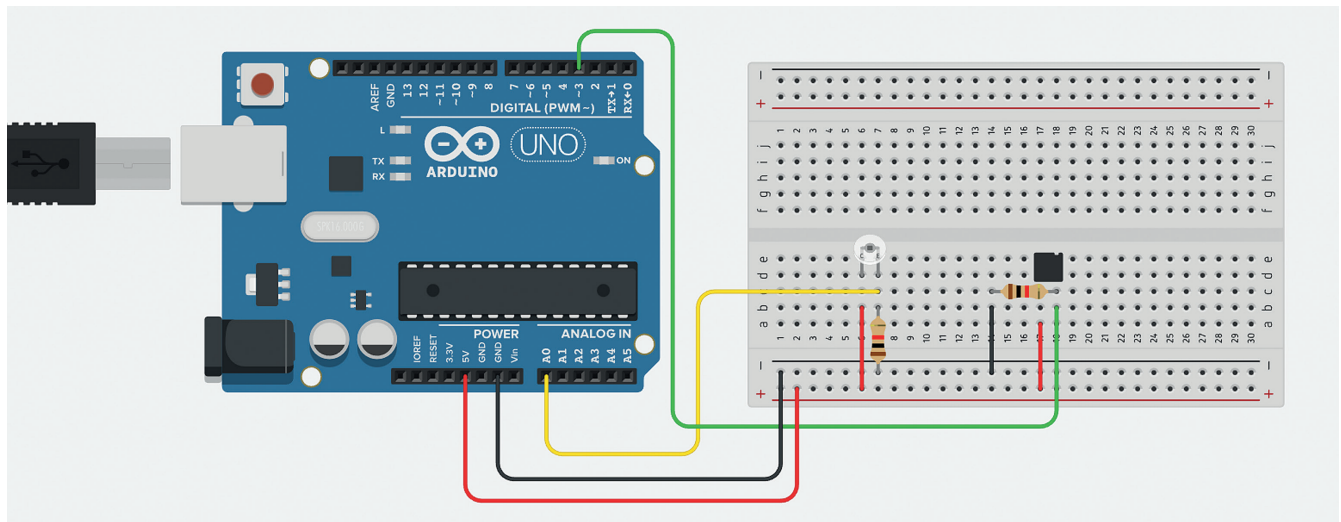


Figure 7.16: Complete circuit in Tinkercad

Physical Circuit

This photo represents what the physical circuit will look like.

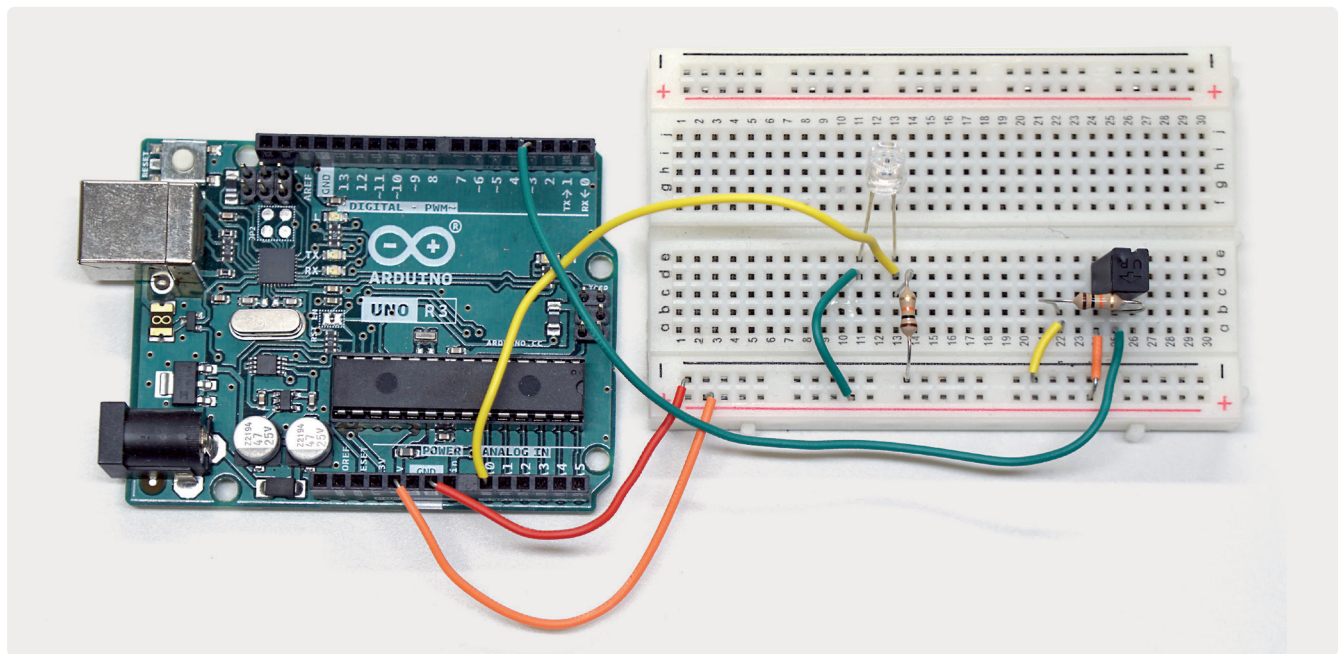


Figure 7.18: Photo of the physical circuit

The components are connected to the following pins:

A0



D3



Figure 7.17: Pins connected to components



Programming the Arduino

You will begin by uploading the **StandardFirmata** sketch through the **Arduino IDE** to setup a communication channel between the Arduino and the Python script that you will write.

Open **PyCharm** and install the **paho-mqtt** Python package through **pip**. In **PyCharm**, open the terminal in your working directory and enter the following command:

```
pip install paho-mqtt
```

Create a new Python file called **mqtt_arduino.py** and at the beginning of your code import the following packages:

- **datetime**: Create timestamps for the messages that we send.
- **time**: Control the program flow.
- **json**: Work with JSON objects.
- **pyfirmata**: Communicate with the Arduino board through the Firmata protocol.
- **paho.mqtt.client**: Create clients that communicate with MQTT brokers.

```
from datetime import datetime
import time
import json
import pyfirmata
import paho.mqtt.client as mq
```

Create the following variables which will be used for the MQTT client that we will create. **CLIENT_ID** is the name you will give to client. **MQTT_BROKER** is the address of the public broker provided by EMQX that you will connect to. **TOPIC** is the name of the topic that the client will subscribe to. **PORT** is the default server port to connect to the MQTT broker. **FLAG_CONNECTED** is a flag variable you will use on an event handler function later.

```
# Variables to setup MQTT client
CLIENT_ID = "PUBLISHER_01"      # ID of the client
MQTT_BROKER = "broker.emqx.io"  # Address of the broker
TOPIC = "waste/drops"           # Topic to subscribe to
PORT = 1883                     # Default server port
FLAG_CONNECTED = False          # Connection flag
```



Table 7.3: MQTT Broker Connection Variables

Variable	Description
CLIENT_ID	The name of the MQTT client.
MQTT_BROKER	Address of the target MQTT broker.
TOPIC	The topic that the client will subscribe to.
PORT	The server port to connect to.
FLAG_CONNECTED	Flag variable to check server connection.

Add the following lines that initialize a connection to the Arduino with the Firmata protocol and set the two pins for the light and tilt sensor we will use to get our data.

```
board = pyfirmata.Arduino('COM4')    # Specify communication port
it = pyfirmata.util.Iterator(board)   # Select the board to connect
it.start()                            # Connect to board

# Selecting the sensor pins
light_sensor_pin = board.get_pin('a:0:i')
tilt_sensor_pin = board.get_pin('d:3:i')
```

Create the following variables; **can_full** is a flag to set whether the can has been filled or not. **garbage_drops** is a counter to track how many uses were needed to fill the can.

```
can_full = False    # Flag to indicate whether the can is full
garbage_drops = 0   # Counter for the garbage drops
```

Create the following function which resets the **can_full** and **garbage_drops** variables every time the can is full, and next we publish a message to the client about it.

```
def reset_can():
    global garbage_drops    # Access the garbage_drops variable
    global can_full        # Access the can_full variable
    garbage_drops = 0      # Reset the counter to 0
    can_full = False       # Clear the can
```



Create the following function to publish a message that the can was used to the client. You will first create a **timestamp** variable to record the time and create a dictionary object with the **timestamp**, **garbage_drops** and **can_full** properties. You will convert this dictionary to a JSON object and then publish it to the subscribed **"waste/drops"** topic through the client.

```
def publish_message():
    global garbage_drops # Access garbage_drops variable
    global can_full      # Access can_full variable
    # Create a custom format for the timestamp
    timestamp = str(datetime.now().strftime("%H:%M:%S"))
    msg_dictionary = {    # Creating the JSON object
        "timestamp": timestamp,
        "garbage_drops": garbage_drops,
        "can_full": can_full
    }
    msg = json.dumps(msg_dictionary) # Convert dictionary to JSON
    try:
        result = client.publish(TOPIC, msg) # Publish message
    except:
        print("There was an error while publishing the message")
    time.sleep(2)
```

Create the following event handler function that prints a confirmation message to the terminal about whether or not the connection to the client was successful. The function's arguments are default arguments that must be used to bind that function to the appropriate event handler provided by the **paho.mqtt.client** library.

```
def on_connect(client, userdata, flags, rc):
    global FLAG_CONNECTED # Access the FLAG_CONNECTED variable

    if rc == 0:           # If rc is 0 the client connected successfully
        FLAG_CONNECTED = True
        print("Connected to MQTT Broker!")
    else:
        print("Failed to connect to MQTT Broker!")
```



In the main part of the program, you will initialize the MQTT client, bind the **on_connect** event handler to the above function, connect to the specified MQTT broker, and subscribe to the specified topic.

```
client = mq.Client(CLIENT_ID)      # Initialize an MQTT client
client.on_connect = on_connect     # Bind the on_connect event handler
client.connect(MQTT_BROKER, PORT)  # Connect to the specified MQTT broker
client.subscribe(TOPIC, 0)         # Subscribe to the specified topic
```

Create the main loop for the program. If the **light_value** has a value of less than 0.200 then the can is considered full.

```
while True:
    # Get sensor values
    light_value = light_sensor_pin.read()
    tilt_value = tilt_sensor_pin.read()

    if (light_value is not None) and (tilt_value is not None):
        print("Light levels : " + str(light_value))
        print("Tilt levels : " + str(tilt_value))
        print("Garbage drops : " + str(garbage_drops))

    # If there is a tilt, add 1 to the counter
    if (tilt_value == True):
        garbage_drops += 1
        # If there is a tilt and the can is full
        # publish a message and reset the can
        if (light_value <= 0.200):
            can_full = True
            publish_message()
            reset_can()
            publish_message()
        time.sleep(1)
```



Complete Code

```
from datetime import datetime
import time
import json
import pyfirmata
import paho.mqtt.client as mq

# Variables to setup MQTT client
CLIENT_ID = "PUBLISHER_01"      # ID of the client
MQTT_BROKER = "broker.emqx.io"  # Address of the broker
TOPIC = "waste/drops"           # Topic to subscribe to
PORT = 1883                      # Default server port
FLAG_CONNECTED = False          # Connection flag

board = pyfirmata.Arduino('COM4') # Specify communication port
it = pyfirmata.util.Iterator(board) # Select the board to connect
it.start()                        # Connect to board

# Selecting the sensor pins
light_sensor_pin = board.get_pin('a:0:i')
tilt_sensor_pin = board.get_pin('d:3:i')

can_full = False    # Flag to indicate whether the can is full
garbage_drops = 0    # Counter for the garbage drops

def reset_can():
    global garbage_drops # Access garbage_drops variable
    global can_full      # Access can_full variable
    garbage_drops = 0     # Reset the counter to 0
    can_full = False      # Clear the can
```



```

def publish_message():
    global garbage_drops    # Access garbage_drops variable
    global can_full         # Access can_full variable

    # Create a custom format for the timestamp
    timestamp = str(datetime.now().strftime("%Y-%m-%d %H:%M:%S"))

    # Creating the dictionary object
    msg_dictionary = {
        "timestamp": timestamp,
        "garbage_drops": garbage_drops,
        "can_full": can_full
    }
    msg = json.dumps(msg_dictionary) # Convert dictionary to JSON

    try:
        result = client.publish(TOPIC, msg) # Publish message
    except:
        print("There was an error while publishing the message")

    time.sleep(2)
    print("Message sent to the MQTT broker")

def on_connect(client, userdata, flags, rc):
    global FLAG_CONNECTED # Access the FLAG_CONNECTED variable

    if rc == 0:           # If rc is 0 the client connected successfully
        FLAG_CONNECTED = True
        print("Connected to MQTT Broker!")
    else:
        print("Failed to connect to MQTT Broker!")

```




```

client = mq.Client(CLIENT_ID)      # Initialize an MQTT client
client.on_connect = on_connect     # Bind the on_connect event handler
client.connect(MQTT_BROKER, PORT)  # Connect to the specified MQTT broker
client.subscribe(TOPIC, 0)         # Subscribe to the specified topic

while True:
    # Get sensor values
    light_value = light_sensor_pin.read()
    tilt_value = tilt_sensor_pin.read()

    if (light_value is not None) and (tilt_value is not None):
        print("Light levels : " + str(light_value))
        print("Tilt levels : " + str(tilt_value))
        print("Garbage drops : " + str(garbage_drops))

        # If there is a tilt, add 1 to the counter
        if (tilt_value == True):
            garbage_drops += 1

        # If there is a tilt and the can is filled
        # publish a message and reset the can
        if (light_value <= 0.200):
            can_full = True
            publish_message()
            reset_can()

        publish_message()

    time.sleep(1)

```



Testing the Broker

EMQX is a public MQTT broker for testing and developing MQTT applications. It aids in developing IoT application prototypes without the cost of infrastructure and the development of the broker. You will use the MQTT X client to test the publishing of our messages. You will later create another Python script that will receive the published messages, generate reports for the can and perform data analysis on those reports. After you have uploaded the StandardFirmata sketch to the Arduino, execute the Python script and move the breadboard to activate the tilt sensor. Every time it is activated, the garbage count tracker will count up. When you activate the tilt sensor with the light sensor covered, the program will publish a message that the can is full, resetting the garbage counter. In the next lesson you will perform data analysis on the data from the published messages.

To test that your messages were published correctly you will use the **MQTTX** desktop client. Before executing the Python script you will use the MQTTX client to subscribe to the "**waste/drops**" topic. Now the client will wait to receive the messages that are published through the Python script and are distributed through the **EMQX** public broker.

To use MQTTX to subscribe to the specified topic:

- > On the menu of the **desktop_connection** connection, click on the **New Subscription** button. **1**
- > On the **Topic** textbox, enter the text **waste/drops**. **2**
- > Click on the **Confirm** button. **3**

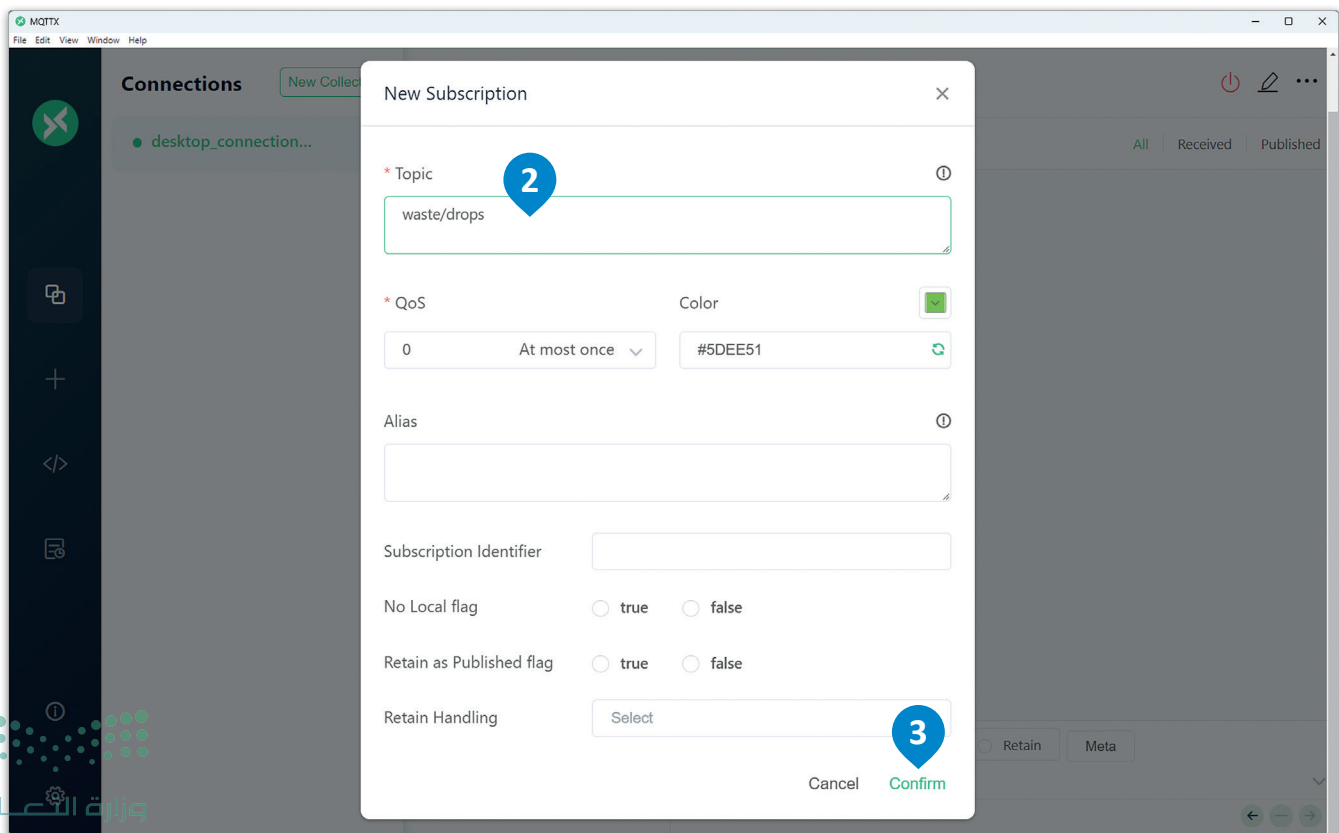
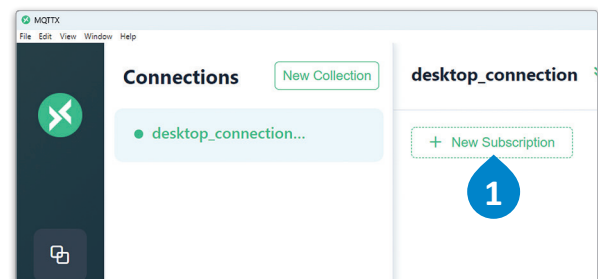


Figure 7.19: Using MQTTX to subscribe to the specified topic

Viewing Messages through the MQTTX Client

After you have executed the Python script and it begins to publish messages, you will receive those messages in the **MQTTX** desktop client like this.

```
Topic: waste/drops  QoS: 0  
{ "timestamp": "2022-10-03 11:14:54", "garbage_drops": 9, "can_filled": false }  
2022-10-31 16:14:07:018
```

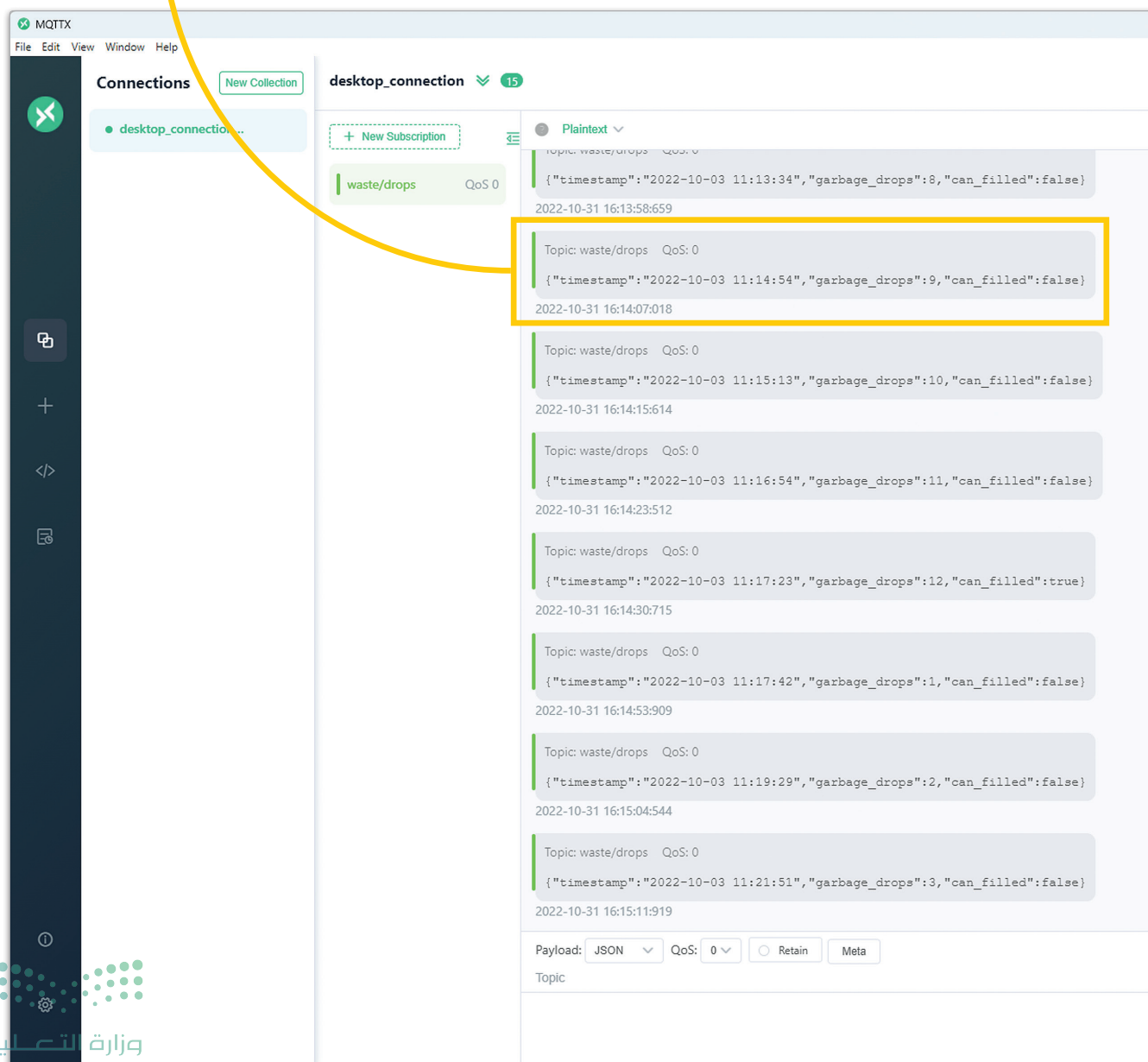


Figure 7.20: Viewing messages through the MQTTX client

Exercises

- 1 Create a diagram of an MQTT network with one Arduino as the publisher and two other Arduinos as the receivers.



- 2 Describe the phototransistor and tilt sensor components and their use cases.



3 Analyze what the public EMQX broker is and how it aids in developing IoT prototype applications.

This image shows a single sheet of white paper with horizontal ruling lines. The lines are evenly spaced and run across the width of the page. There are no margins, text, or other markings on the paper.

- 4 Update the event handler for the **on_connect** event that prints to the console the configuration info that you provided to the MQTT client.

5 Update the **publish_message()** function to publish a message with a timestamp that presents the complete date and time and a new property that contains the client identifier.

- 6 Create a Python script that lets the user type the topic they want to subscribe to and the message they want to send and then publish it through the public EMQX broker. Test your script with the MQTT X desktop client.



Lesson 3

Building a Smart Waste Management Solution

Link to digital lesson



www.ien.edu.sa

Smart Waste Management and Data Analysis

In the previous lesson, you created a smart waste management prototype for a garbage can with an Arduino microcontroller that monitors its environment, generates data from sensors, and publishes this data as messages to an MQTT topic. This data needs to be collected and processed to generate insights from it.

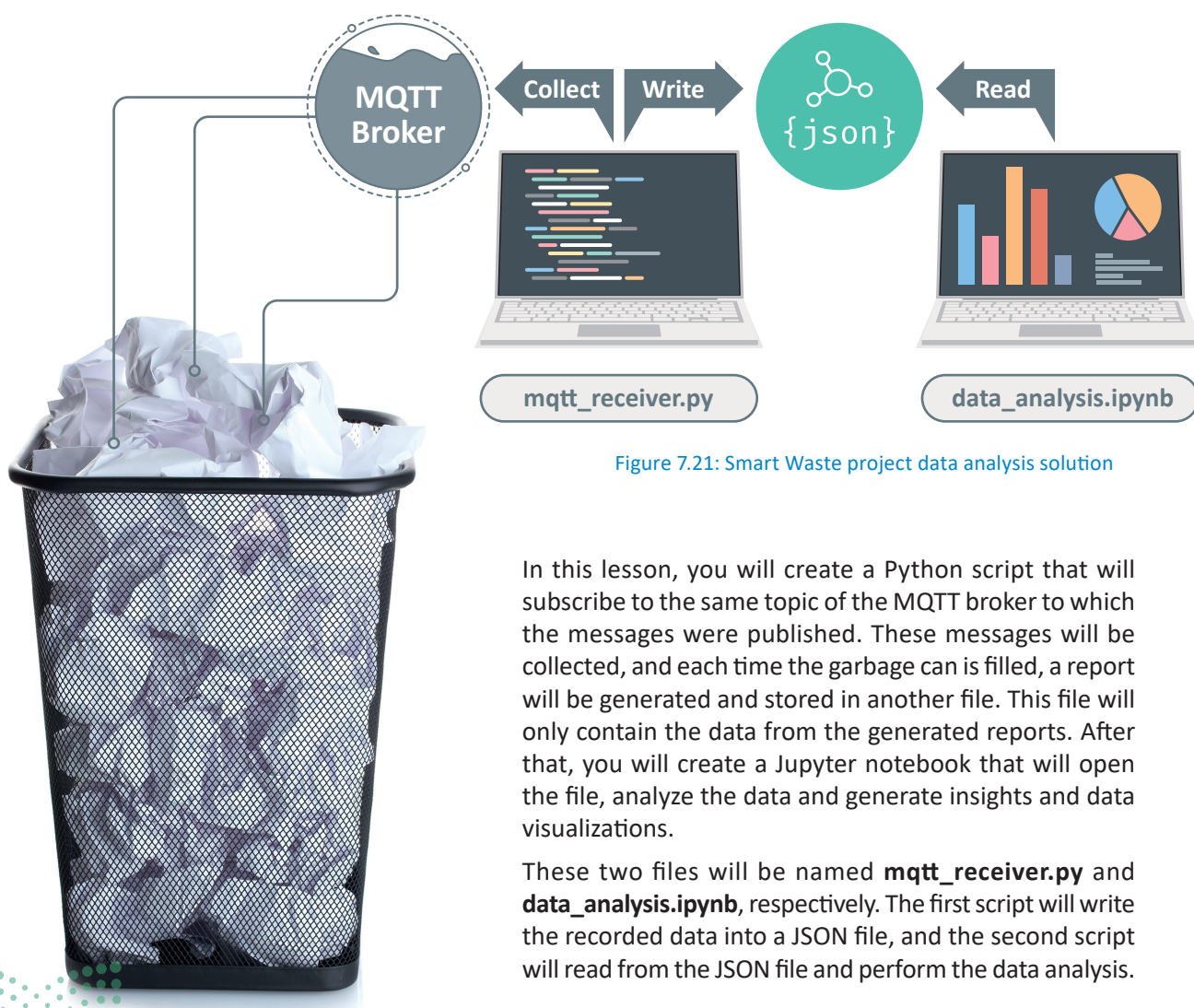


Figure 7.21: Smart Waste project data analysis solution

In this lesson, you will create a Python script that will subscribe to the same topic of the MQTT broker to which the messages were published. These messages will be collected, and each time the garbage can is filled, a report will be generated and stored in another file. This file will only contain the data from the generated reports. After that, you will create a Jupyter notebook that will open the file, analyze the data and generate insights and data visualizations.

These two files will be named **mqtt_receiver.py** and **data_analysis.ipynb**, respectively. The first script will write the recorded data into a JSON file, and the second script will read from the JSON file and perform the data analysis.



وزارة التعليم

Ministry of Education

2026 - 1448

Creating the JSON Data File

We are going to create a JSON data file with an empty array. The **mqtt_receiver.py** script will append each generated report as a JSON object to the array. The **data_analysis.py** will open this JSON file, read the JSON array contents and perform the data analysis operations.

Open **PyCharm** and in your working directory create a new file named **data.json**. Inside that file create an empty array object as shown in the box below. The **mqtt_receiver.py** will append JSON objects of the generated reports to the array presented below. Save and close the **data.json** file.

```
[ ]
```

Create a new Python file called **mqtt_receiver.py** and at the beginning of your code import the following packages:

- **datetime:** Create timestamps for the messages that we send.
- **json:** Work with JSON objects.
- **paho.mqtt.client:** Create clients that communicate with MQTT brokers.
- **os:** Work with files on your computer.

```
from datetime import datetime
import json
import paho.mqtt.client as mq
from os import path
```

Create the following variables **data_file** and **data_file_objects** which will be used to interact with the JSON data file.

```
data_file = "your_file_path" # Absolute path to the JSON data file
data_file_objects = [ ] # This contains the objects from the JSON data file
```



Create the following variables which will be used for the MQTT client that we will create. **CLIENT_ID** is the name we will give to our client. **MQTT_BROKER** is the address of the public broker provided by EMQX that we will connect to. **TOPIC** is the name of the topic that the client will subscribe to. **PORT** is the default server port to connect to the MQTT broker. **FLAG_CONNECTED** is a flag variable we will use on an event handler function later.

```
# Variables to setup MQTT client

CLIENT_ID = "RECEIVER_01"      # ID of the client
MQTT_BROKER = "broker.emqx.io" # Address of the broker
TOPIC = "waste/drops"          # Topic to subscribe to
PORT = 1883                     # Default server port
FLAG_CONNECTED = False         # Connection flag
```

Create the following variables **messages_stack** and **reports** which will be used to store information from the published messages.

```
messages_stack = [] # The array with the messages per can filling
reports = []        # The array with all the generated report objects
```

Create the following event handler function that prints a confirmation message to the terminal about whether or not the connection to the client was successful. The function's arguments are default arguments that must be used to bind that function to the appropriate event handler provided by the **paho.mqtt.client** library.

```
def on_connect(client, userdata, flags, rc):
    global FLAG_CONNECTED # Access the FLAG_CONNECTED variable

    if rc == 0:            # If rc is 0 the client connected successfully
        FLAG_CONNECTED = True
        print("Connected to MQTT Broker!")
    else:
        print("Failed to connect to MQTT Broker!")
```



Create the following event handler function that triggers each time a message is published to the subscribed topic. The function's arguments are default arguments that must be used to bind that function to the appropriate event handler provided by the **paho.mqtt.client** library.

```
def on_message(client, userdata, msg):  
    global messages_stack # Access the messages_stack variable  
  
    # Decode the message payload  
    payload = str(msg.payload.decode())  
  
    # Convert the payload to a JSON object and append it  
    # to the messages stack  
    payload_object = json.loads(payload)  
    messages_stack.append(payload_object)  
  
    # When you receive a message, print it to the terminal  
    print("||---- MESSAGE RECEIVED ----||\n")  
    print("Payload: " + str(payload_object))  
  
    # If the payload object has the can_filled flag set to True  
    # generate a report for the filled can  
    if payload_object["can_filled"] == True:  
        generate_report()
```

This Python script that serves the role of an MQTT receiver can collect messages from multiple Arduino publishers at the same time. This solution can be further expanded to process JSON data with even more fields containing information about the publishers as well.

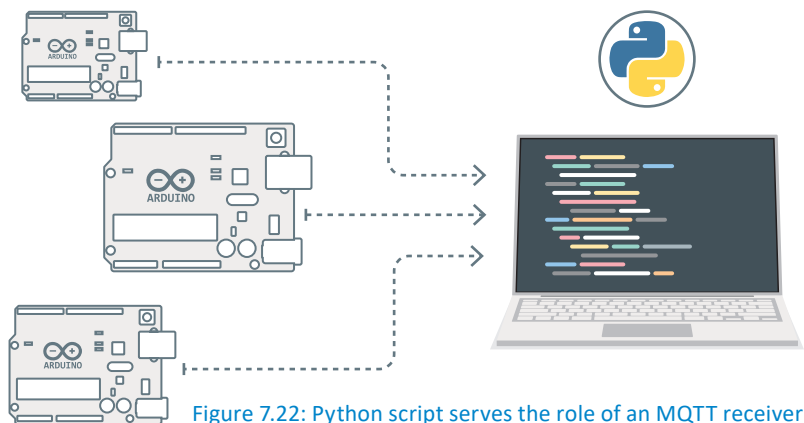


Figure 7.22: Python script serves the role of an MQTT receiver



Create the following **generate_report** function which will create a report JSON object and append it to the data file every time that a message is received that indicates that the garbage can is full.

```
def generate_report():  
    global data_file_objects # Access data_file_objects variable  
    global messages_stack    # Access messages_stack variable  
    global reports           # Access reports variable  
  
    # Getting the first and last objects from the messages_stack  
    first_msg = messages_stack[0]  
    last_msg = messages_stack[len(messages_stack) - 1]  
  
    # Converting the string attributes of the messages objects to datetime  
    time_filled_timestamp = last_msg["timestamp"]  
    first_timestamp = datetime.strptime(first_msg["timestamp"], "%H:%M:%S")  
    last_timestamp = datetime.strptime(last_msg["timestamp"], "%H:%M:%S")  
  
    garbage_drops = last_msg["garbage_drops"]  
  
    # Calculating the time_to_fill by comparing the timestamps  
    # of the first and last fillings  
    time_delta = last_timestamp - first_timestamp  
    time_to_fill = time_delta.total_seconds() / 60  
    report_id = len(reports) # This will be used for object indexing  
    # The JSON object that will be appended to the JSON data file  
    report = {  
        "id": report_id,  
        "timestamp": time_filled_timestamp,  
        "garbage_drops": garbage_drops,  
        "time_to_fill": time_to_fill  
    }
```



```

# Append the new report to the objects of the data file
# and write the data_file_objects array to the data file
data_file_objects.append(report)
with open(data_file, 'w') as file:
    json.dump(data_file_objects, file, indent=4, separators=(',', ': '))

# Append the report object to the reports array and to the JSON data file
# and clear the messages stack
reports.append(report)
messages_stack = []

```

In the main part of the program, we will check if the data exists, we will open it and then we will initialize the MQTT client, bind the **on_connect** and **on_message** event handlers to the above functions, connect to the specified MQTT broker, subscribe to the specified topic and listen for incoming messages.

```

# Check if the data file exists
if path.isfile(data_file) is False:
    raise Exception("Data file not found")
# Read the contents of the JSON data file
with open(data_file) as fp:
    data_file_objects = json.load(fp)

client = mq.Client(CLIENT_ID)      # Initialize an MQTT client
client.on_connect = on_connect     # Bind the on_connect event handler
client.on_message = on_message     # Bind the on_message event handler
client.connect(MQTT_BROKER, PORT)  # Connect on the specified MQTT broker
client.subscribe(TOPIC, 0)         # Subscribe to the specified topic
client.loop_forever()              # Listen continuously for messages

```



Complete Code

```
from datetime import datetime
import json
import paho.mqtt.client as mq
from os import path

data_file = "your_file_path" # Absolute path to the JSON data file
data_file_objects = [] # This contains the objects from the JSON data file

# Variables to setup MQTT client
CLIENT_ID = "RECEIVER_01" # ID of the client
MQTT_BROKER = "broker.emqx.io" # Address of the broker
TOPIC = "waste/drops" # Topic to subscribe to
PORT = 1883 # Default server port
FLAG_CONNECTED = False # Connection flag

messages_stack = [] # The array with the messages per can filling
reports = [] # The array with all the generated report objects

def on_connect(client, userdata, flags, rc):
    global FLAG_CONNECTED # Access the FLAG_CONNECTED variable

    if rc == 0: # If rc is 0 the client connected successfully
        FLAG_CONNECTED = True
        print("Connected to MQTT Broker!")
    else:
        print("Failed to connect to MQTT Broker!")

def on_message(client, userdata, msg):
    global messages_stack # Access the messages_stack variable
```



```

# Decode the message payload
payload = str(msg.payload.decode())

# Convert the payload to a JSON object and append it
# to the messages stack
payload_object = json.loads(payload)
messages_stack.append(payload_object)

# When you receive a message, print it to the terminal
print("||---- MESSAGE RECEIVED ----||\n")
print("Payload: " + str(payload_object))

# If the payload object has the can_filled flag set to True
# generate a report for the filled can
if payload_object["can_filled"] == True:
    generate_report()
def generate_report():
    global data_file_objects # Access data_file_objects variable
    global messages_stack    # Access messages_stack variable
    global reports           # Access reports variable

    # Getting the first and last objects from the messages_stack
    first_msg = messages_stack[0]
    last_msg = messages_stack[len(messages_stack) - 1]

    # Converting the string datetimes to datetime objects
    first_timestamp = datetime.strptime(first_msg["timestamp"], "%H:%M:%S")
    last_timestamp = datetime.strptime(last_msg["timestamp"], "%H:%M:%S")
    garbage_drops = last_msg["garbage_drops"]

    # Calculating the time_to_fill by comparing the timestamps
    # of the first and last fillings
    time_delta = last_timestamp - first_timestamp

```

```

time_to_fill = time_delta.total_seconds() / 60
report_id = len(reports) # This will be used for object indexing

# The JSON object that will be appended to the JSON data file
report = {
    "id": report_id,
    "garbage_drops": garbage_drops,
    "time_to_fill": time_to_fill
}

# Append the new report to the objects of the data file
# and write the data_file_objects array to the data file
data_file_objects.append(report)
with open(data_file, 'w') as file:
    json.dump(data_file_objects, file, indent=4, separators=(',', ': '))

# Append the report object to the reports array and to the JSON data file
# and clear the messages stack
reports.append(report)
messages_stack = []

# Check if the data file exists
if path.isfile(data_file) is False:
    raise Exception("Data file not found")

# Read the contents of the JSON data file
with open(data_file) as fp:
    data_file_objects = json.load(fp)

client = mq.Client(CLIENT_ID) # Initialize an MQTT client
client.on_connect = on_connect # Bind the on_connect event handler
client.on_message = on_message # Bind the on_message event handler
client.connect(MQTT_BROKER, PORT) # Connect on the specified MQTT broker
client.subscribe(TOPIC, 0) # Subscribe to the specified topic
client.loop_forever() # Listen continuously for messages

```

Data Analysis in Jupyter Notebook

Now we will use a Jupyter Notebook to perform data analysis operations on the JSON data file. Because it would take too long to collect the data needed to perform data analysis, there is a prefilled JSON dataset that we will use. This dataset simulates leaving the Arduino prototype running for a long period of time.

The JSON file is available for download here:

http://binary-academy.com/dnld/KSA/IOT2/U3_L3_DATA.json



In the beginning, we will import the required libraries and read the JSON data from the file.

```
import os
import pandas as pd          # library used for data manipulation
import matplotlib.pyplot as plt # library used for plotting data

# The data that will be used, extracted from the JSON dataset
data = pd.read_json('U3_L3_DATA.json', 'records', convert_dates=['timestamp'])
```

We will then describe the dataset to extract statistical properties.

```
data.describe().round(0)
```

This is the downloaded data file that must be placed in the same project folder.

	id	garbage_drops	time_to_fill
count	50.0	50.0	50.0
mean	24.0	54.0	152.0
std	15.0	30.0	100.0
min	0.0	2.0	5.0
25%	12.0	30.0	60.0
50%	24.0	55.0	147.0
75%	37.0	78.0	235.0
max	49.0	100.0	376.0

Figure 7.23: Data description

We will create two histograms grouped by the **garbage_drops** and the **time_to_fill** properties.

```
# Create histograms for the data using 8 groupings
hist = data.hist(['garbage_drops'],figsize=(10,6),bins=8)
hist = data.hist(['time_to_fill'],figsize=(10,6),bins=8)
```

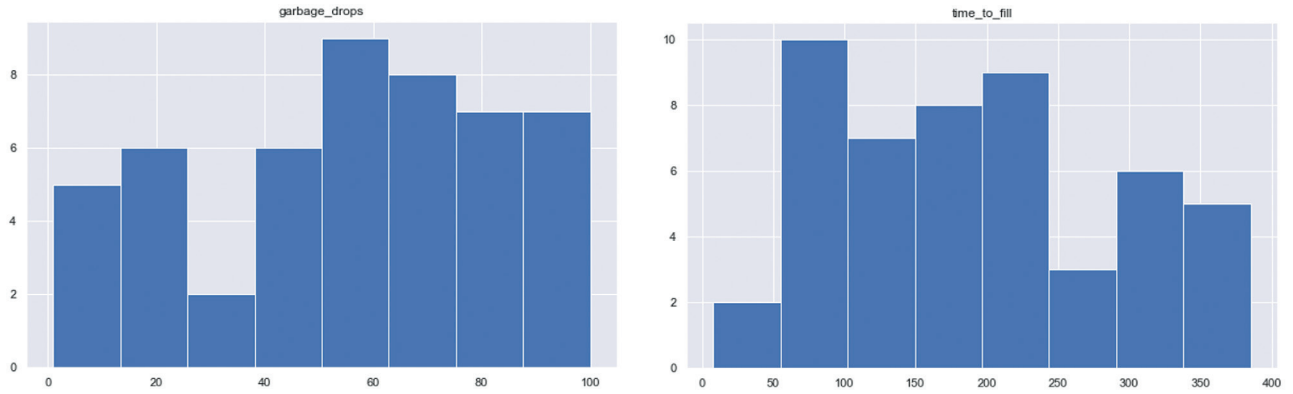


Figure 7.24: Histograms

We will then create two stem plots to display the **garbage_drops** and **time_to_fill** values at each time interval.

```
# Create stem plots for the data with diamond-shaped ('D') markers
plt.stem(data['timestamp'], data['time_to_fill'], markerfmt='D');
plt.stem(data['timestamp'], data['garbage_drops'], markerfmt='D');
```

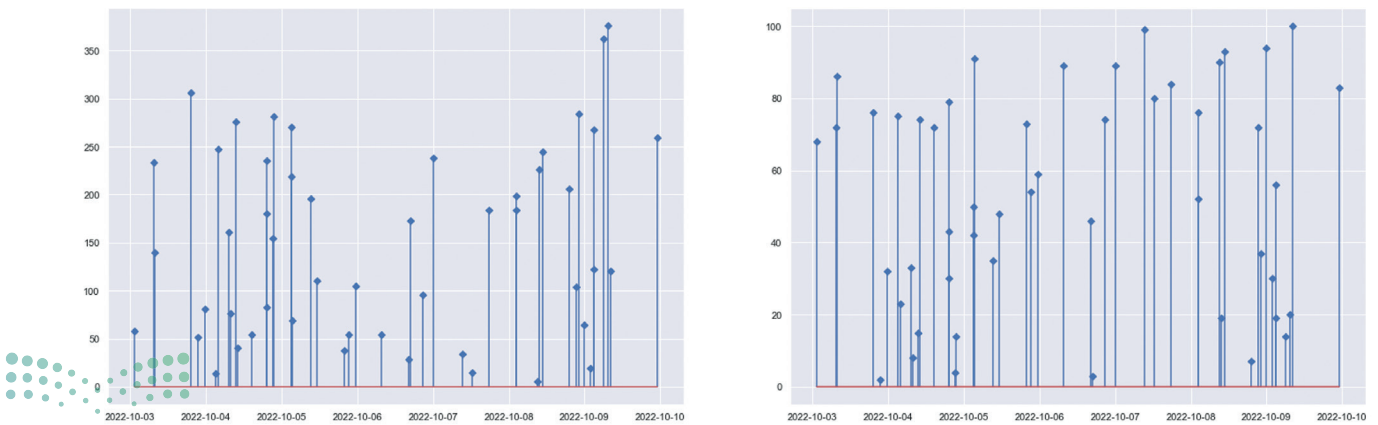


Figure 7.25: Stem plots

Finally, we will create two plots that group the number of **garbage_drops** for each hour of the day and for each hour of the week.

```
# Create 2 plots, one that groups mean garbage amount by hour and one by day
fig, (ax1,ax2) = plt.subplots(2,figsize=(12, 8))
fig.supylabel('garbage_drops')
data.groupby(data["timestamp"].dt.hour)["garbage_drops"].mean().plot(kind='bar',
rot=0, xlabel='Hour of the day', ax=ax1);

# Monday = 0, Sunday = 6
data.groupby(data["timestamp"].dt.day_of_week)["garbage_drops"].mean().
plot(kind='bar', rot=0, ax=ax2);
```

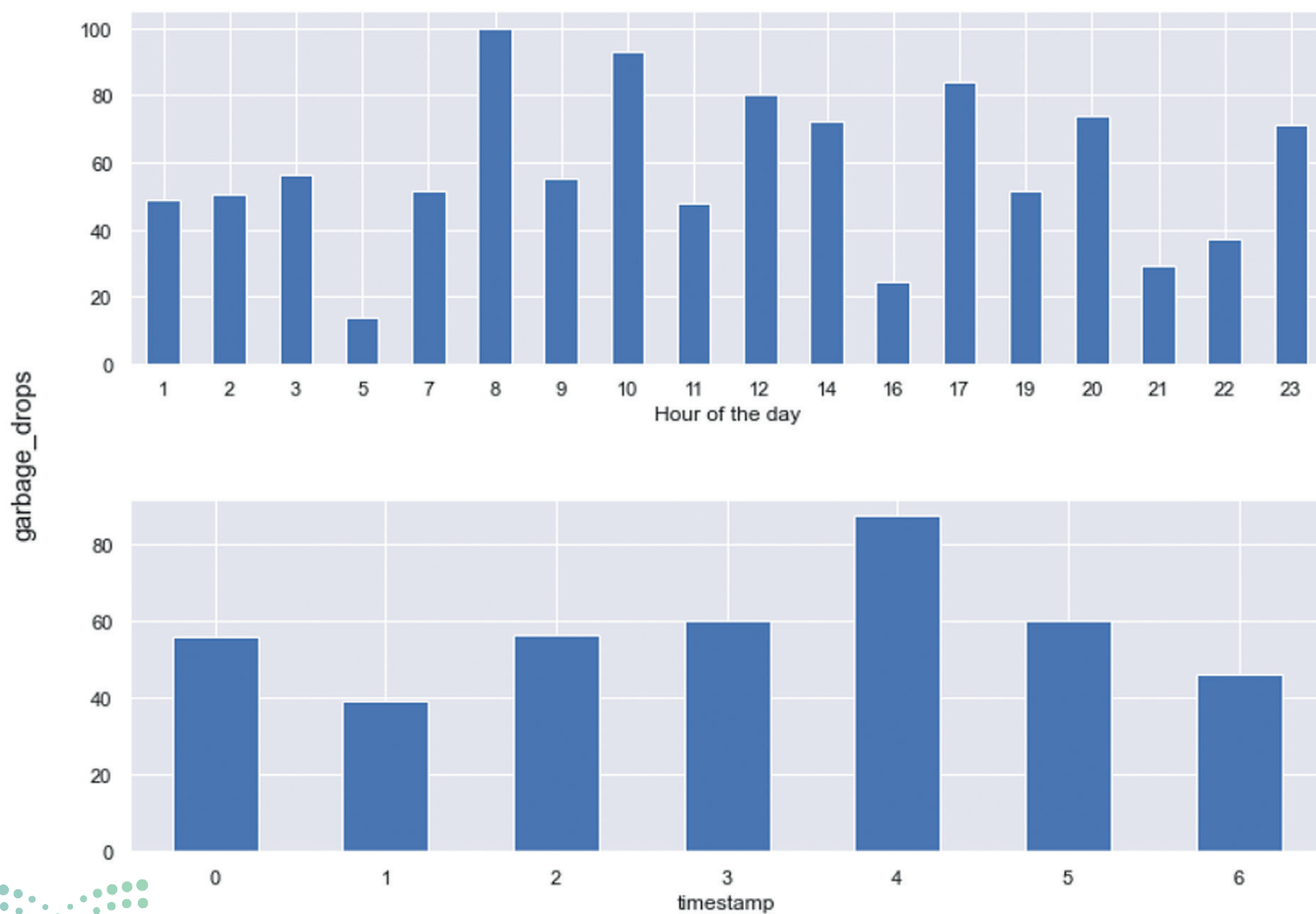
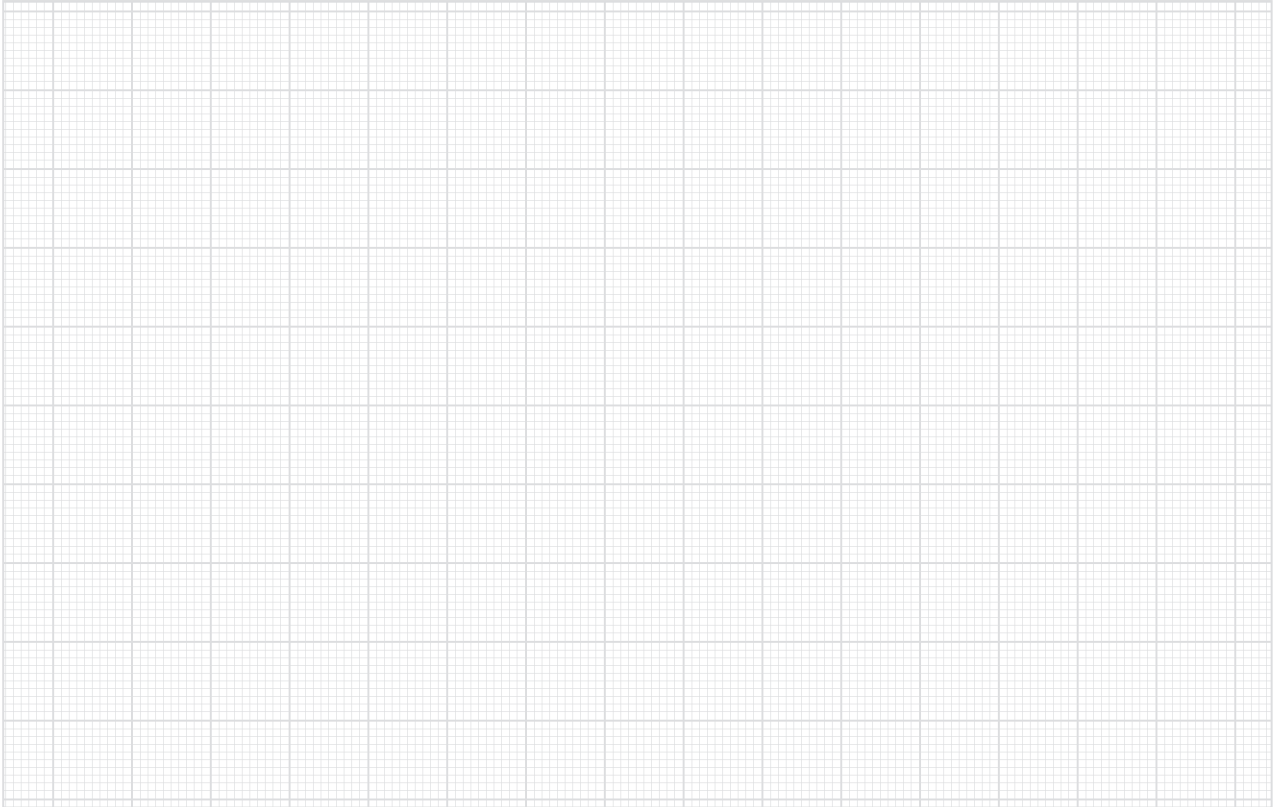


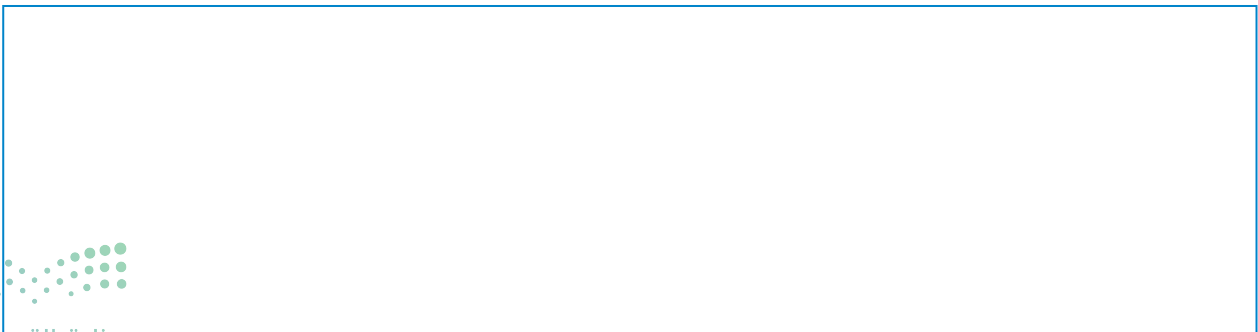
Figure 7.26: Group by plots

Exercises

- 1 Create a diagram that illustrates the relationship between the two Python scripts and the JSON file that holds the data.



- 2 Create a Python script that connects to three topics and write an **on_connect** event handler that prints the configuration info and the topics to which the client is subscribed to the terminal.



- 3 Update the **on_message** object to print info to the terminal about the client that published the data and the topic that the data was received from.

- 4 Create a new JSON file that will hold all the values from the messages stack and use the `generate_report()` function to append the values of the `messages_stack` to the new JSON file.

- 5 In the Jupyter notebook, create a new scatter plot for the same data that you processed in the lesson.

- 6 Add another Python script that will receive the messages that you published from the script in exercise 6 of lesson 2. When you receive a message print the info about the publisher, the receiver, and the subscribed topic to the terminal.



Project

Connections with the MQTT protocol are used in real-world projects. The same architectures for communication through publishers, brokers and receivers can be applied to various other domains. We will create a smart garden solution that is connected and monitored through the MQTT protocol. This architecture can then be scaled to a more industrial application for large smart gardens.

1

Create a new circuit with an Arduino board, a temperature sensor, a soil moisture sensor and a phototransistor.

2

Create another Python script that will be the receiver for the data collected by the Arduino. The user will be prompted to choose which topic to listen to and then it will create a client to subscribe to that topic. It will store the messages and it will display an alert if there is a spike in the latest values.

3

Create a final Python script that will subscribe to the topic with all the readings and will save them in a JSON file. The user will be prompted as to whether they want to listen to the broker and collect data or create visualizations for the readings that are already stored.

4

Run the three Python scripts simultaneously, adjust the environment of the Arduino to update the data readings and observe the outputted results.



Wrap up

Now you have learned to:

- > Analyze the layers of a Smart City Architecture.
- > Publish messages by using the MQTT protocol.
- > Create Python script to publish messages to MQTTX Client.
- > Store reports on a JSON Data file.
- > Perform data analysis operations on a JSON data file using Jupyter Notebook.

KEY TERMS

City Layer	Phototransistor	Services Layer
Client	Prototype	Street Layer
Data Center Layer	Publisher	Subscriber
MQTT Server	Quality of Service	Tilt Sensor
Message Broker	Receiver	



8. IoT Wireless Sensor Network Simulation

In this unit, you will learn the IoT technologies that are used in smart industry and manufacturing. Students will use CupCarbon to create and simulate sensor networks. Finally, you will create a Fire Surveillance and Notification prototype and a Smart Industry and Automation prototype.

Learning Objectives

In this unit, you will learn to:

- > Recognise IoT technologies in manufacturing.
- > Define the use of the Industrial Automation and Control Systems in the connected factory.
- > Create and visualize IoT networks using CupCarbon.
- > Simulate an IoT network using CupCarbon.
- > Create Python scripts to program the nodes of a network.
- > Create a Fire Surveillance and Notification IoT prototype.
- > Create a Smart Industry and Automation IoT prototype.

Tools

- > CupCarbon



Lesson 1

Introduction to CupCarbon

Link to digital lesson



www.ien.edu.sa

Smart Industry

IoT technologies affect all areas of life and industry is no different. Cost minimization and efficiency improvement have always been vital, but as industrial models evolve and competition rises, the emphasis is turning to innovation and enhanced business models. After decades of squeezing costs out of manufacturing processes and the supply chain, businesses are beginning to realize that additional cost reduction may hinder customer service and production quality.

Several IoT-related technologies are at the core of the digital disruption in manufacturing:



Data-Driven Manufacturing

Big data is transforming the industry. Manufacturers can have access to all machine-generated data to monitor quality control in real-time, enhance Overall Equipment Effectiveness (OEE), and decrease unscheduled downtime. OEE is a well recognized indicator of industrial productivity. Additionally, manufacturers are researching methods to use this data to facilitate quick retooling in response to market shifts and other demands.



Operational Technology (OT) and Information Technology (IT) Convergence

In the context of the Internet of Things (IoT) in a manufacturing environment, operational technology consists of Programmable Logic Controllers (PLCs), computers, and other technology that is frequently similar to IT technology but is controlled and owned by business operations outside of IT. IP networking enables deeper machine and factory integration, and the distinction between factory and business networks is disappearing. Manufacturers are seeking methods to integrate their processes under a unified networking infrastructure, pushing beyond traditional silos.



Improved Technology with Lower Costs

Connectivity, monitoring, and optimization of machines are becoming scalable, automated, and platform-based as a result of the emergence of new technologies. In this advanced stage of technology, machines may be considered part of a fully interconnected network system, as opposed to isolated systems. The convergence of computation, networking and security reduces the cost of connecting devices.

Enhanced Efficiency and Safety

Factories, especially in the food and beverage sector, are seeking ways to achieve zero-touch processes, to automate manual activities. IoT, along with robotics and image processing can enable a modern factory to improve efficiency and safety.



An Architecture for the Connected Factory

Companies are beginning to integrate their Industrial Automation and Control Systems (IACS) with IT applications and analytics tools to deliver operational and business-beneficial control and analytics capabilities. IACS are used either to control basic processes or to monitor that safety measures are employed when an abnormal condition occurs. The objectives of IACS are quality and efficiency in production while maintaining a high level of integrity and reliability.

Modbus/TCP

In the industrial sector, Modbus is most frequently employed for master/slave administration. Modbus has been converted to current communications protocols, such as Ethernet and TCP/IP, similar to other automation control technologies. Modbus is widely used because the protocol is an open, documented standard that is well-established worldwide. The Modbus master/slave arrangement is well-suited to the connection-oriented nature of TCP, however this way of communication is often less versatile.

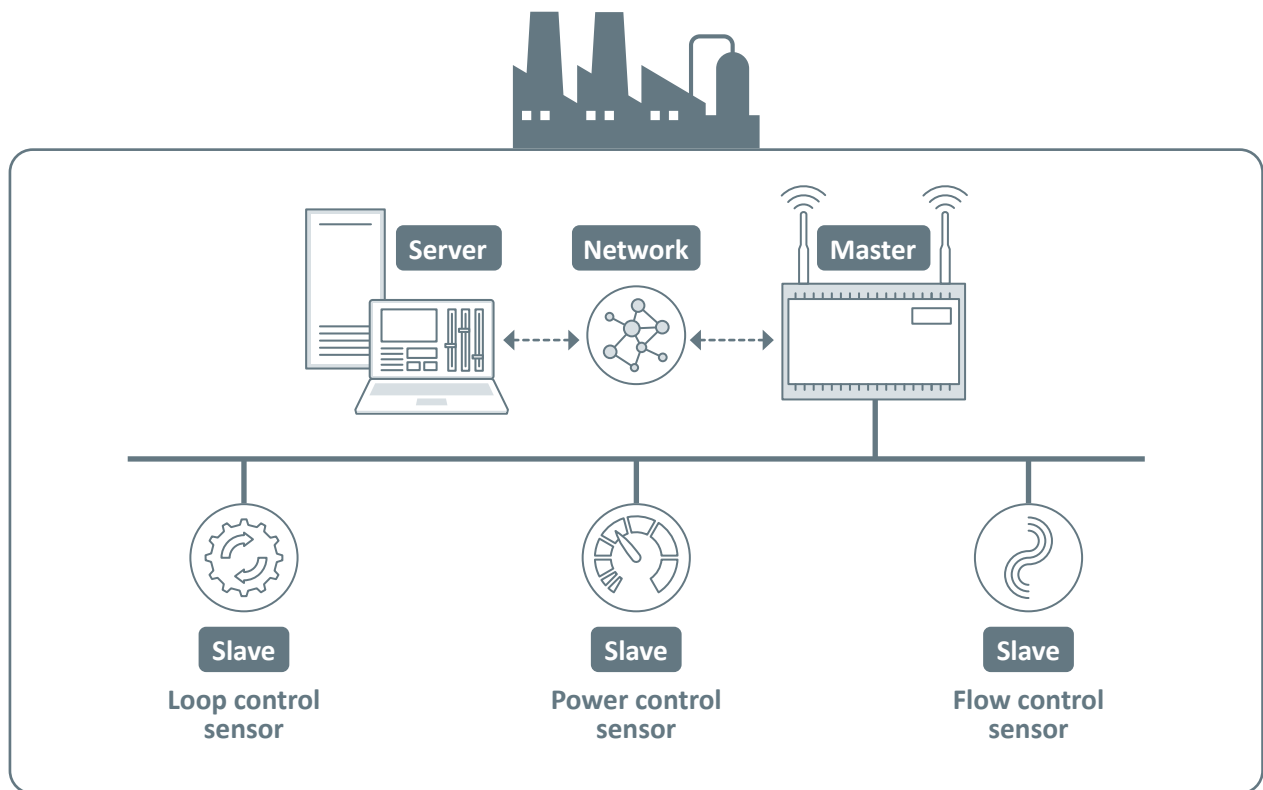


Figure 8.1: Modbus network protocol

Connected Factory Challenges

The manufacturing industry has been one of the primary targets of cybercriminals. The convergence of factory and business networks opens security holes to factory processes that were traditionally isolated. Frequently, the solution has been simply to isolate the factory floor network from the IT corporate network. However, a network disconnected from higher-layer processes will have limited capabilities and IoT-enabled business enhancements. In addition, several potential risks emerge from laptops and workstations on the plant floor that are physically available to contractors or workers with unrestricted access.

Edge Computing in the Connected Factory

The machines on the factory floor can generate vast quantities of data. Many factories have addressed this issue by deploying computers to capture this data. Collecting data from computers on the factory floor has created maintenance and security issues, as each computer requires operating system patches and upgrades. Hardware failures are also prevalent, as equipment is frequently not designed to withstand factory conditions. Clearly, this method makes it extremely challenging for manufacturing operations to efficiently gather, process, and respond to data. Such an approach is a huge hindrance to the visibility and potential commercial benefits that industrial data analytics might provide.

New developments in computation capability at the network's edge are aiding in the resolution of these issues. With machine-embedded and near-machine edge computing devices that incorporate switching, routing, and security in a single durable form factor, manufacturers are beginning to recognize the advantages of linking machines and edge computing services.

Oil and Gas Industry

Oil and gas are two of the most important resources utilized by modern society. From transportation infrastructure to the supply of plastics, almost every element of modern life depends on the availability of these commodities. Today, oil and gas firms are primarily concerned with reducing costs, increasing efficiency and speed, and maximizing returns on current investments. Controlling production costs and enhancing the overall health and safety of hazardous settings are among the most critical Key Performance Indicators (KPIs) of the sector.

Similarly to other sectors, oil and gas firms use the Internet of Things for a number of purposes, including the following:

- Monitoring the state or behavior of industrial equipment for visibility and control.
- Maximizing process and resource utilization.
- Improving business decision making.



Figure 8.2: Oil production monitoring

Industry Key Challenges as Digitization Drivers

The Internet of Things (IoT) and digitization—the process of utilizing breakthroughs in information technology to develop new solutions and technologies for operations, work processes, and methods—are paving the way for previously impossible efficiency gains and new business models.

- Advanced modeling and analytics.
- Big data.
- IT/OT convergence.
- Smart machines.
- Mobility and cloud.
- Asset performance management.

Example

The Uthmaniyah Gas Plant is one of KSA's largest gas processing plants. IoT and AI solutions enhance productivity and facility reliability. Drones and smart objects monitor gas pipeline refinery equipment and use data analysis methods to optimize their usage. Thousands of IoT-enabled sensors monitor the Khurais oil field to predict the oil's state in the wells. These methods reduce energy usage and maintenance expenses.

What CupCarbon is

CupCarbon is a Smart City and Internet of Things Wireless Sensor Network simulator. It can be used to design, create and visualize IoT networks consisting of nodes, devices and events, among others. It provides a plethora of tools in order to configure the networks created, test and optimize them.

Popular protocols you have been taught are included such as Zigbee, Wi-Fi and LoRa. Plus, with OpenStreetMap used as an interface so the simulations can take place anywhere, even in your neighborhood!



After successfully creating your desired network, CupCarbon can supply code for your physical Arduino microcontrollers, so you can try it out on real hardware. The nodes communicate with each other with scripts. The simulator uses its own programming language, SenScript and also supports Python. In this unit, you will program the nodes with Python.

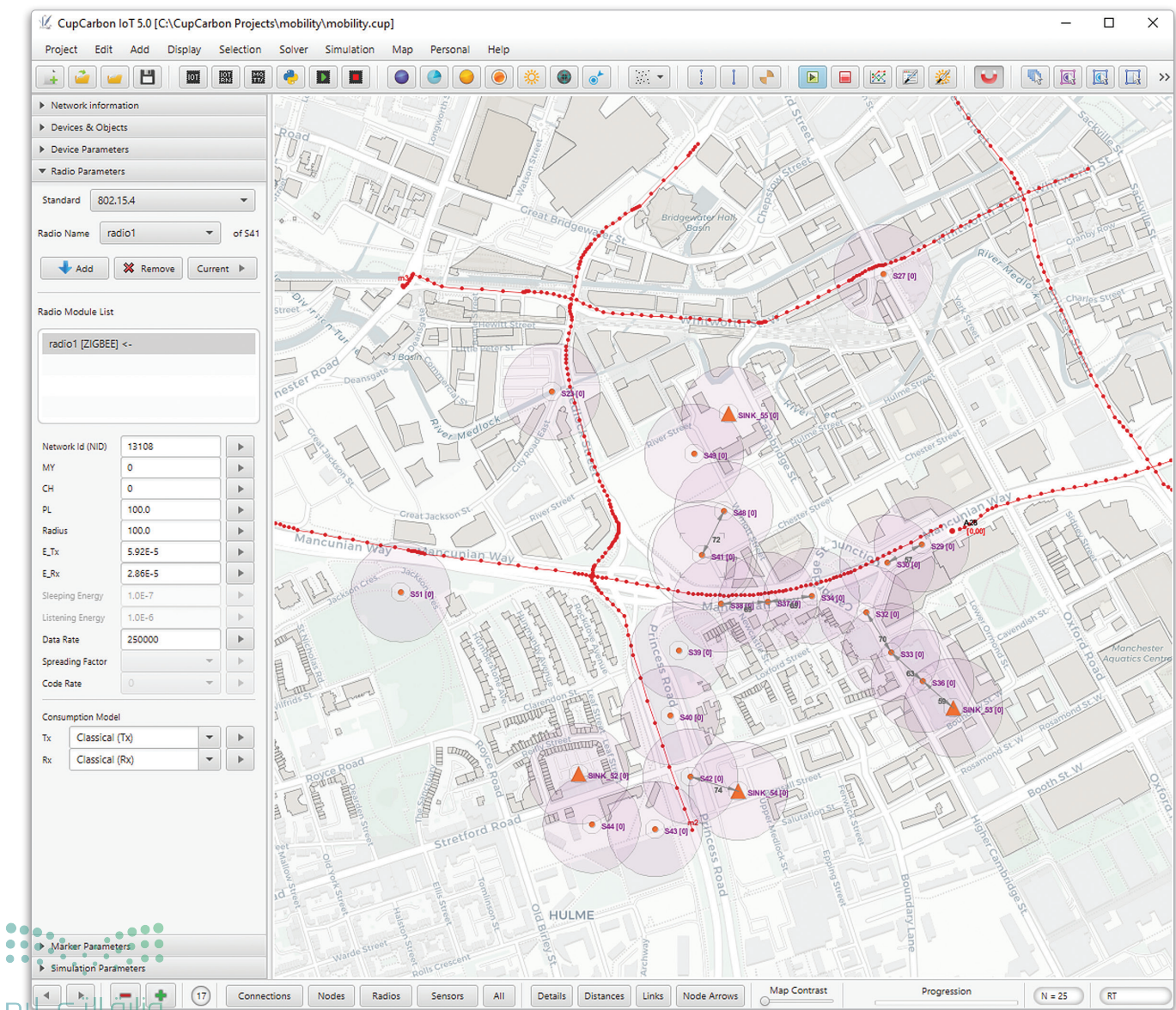
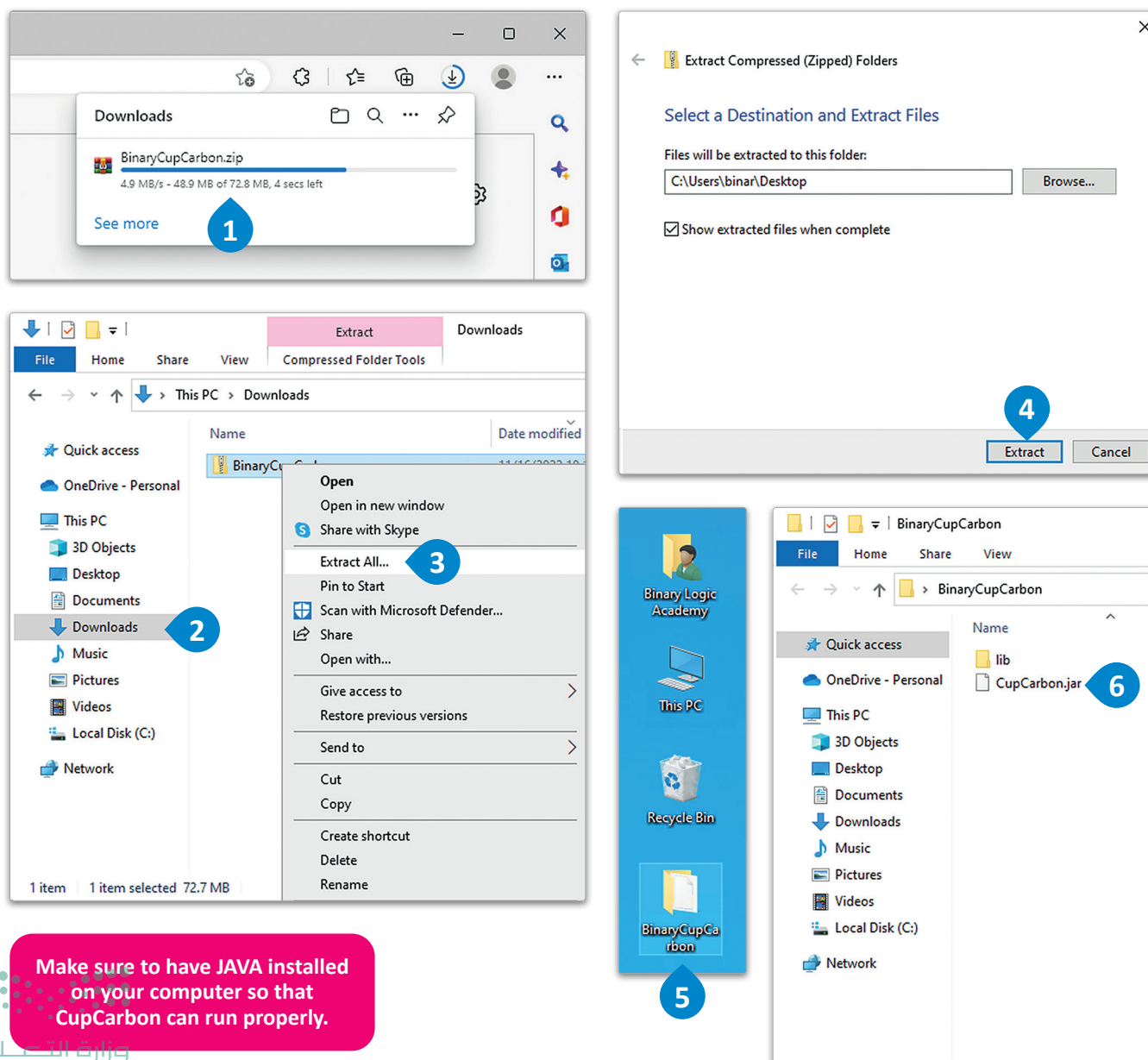


Figure 8.3: A CupCarbon project

To download and run CupCarbon:

- > Open your browser and download the file from <http://binary-academy.com/dnld/KSA/IOT2/BinaryCupCarbon.zip> ①
- > Open the **File Explorer** and find the downloaded file in **Downloads**. ②
- > Right click on the file and choose **Extract All...** ③
- > Choose your **Desktop** as a destination for the extraction and click **Extract**. ④
- > Find the **extracted folder** in you **Desktop** and open in. ⑤
- > Double-click **CupCarbon.jar** to start **CupCarbon**. ⑥



Make sure to have **JAVA** installed on your computer so that **CupCarbon** can run properly.

Figure 8.4: Downloading CupCarbon

The CupCarbon Windows

When you open the program you will notice two windows: the Main Window containing the Map and the Console.

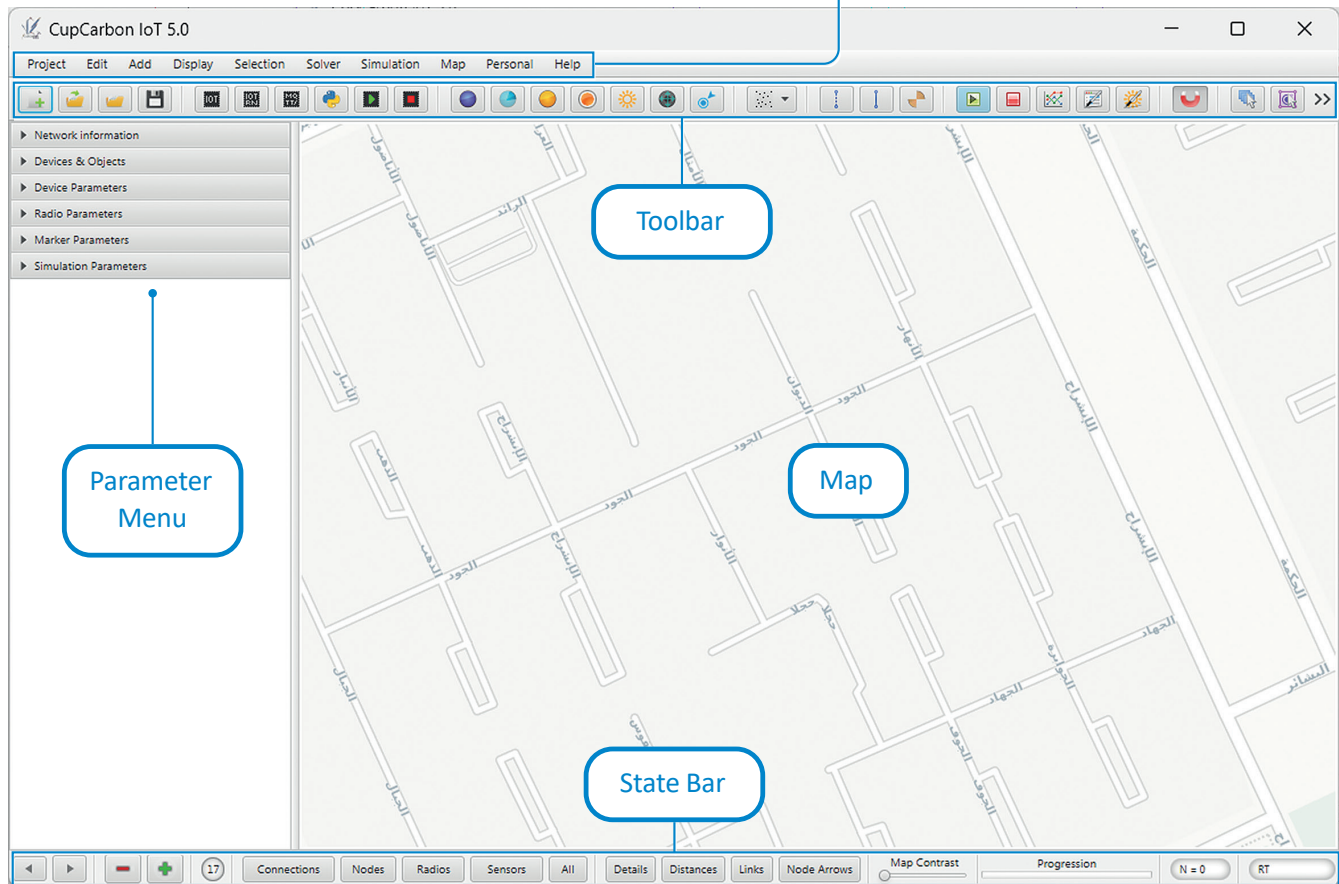


Figure 8.5: CupCarbon Main Window

The Console is used for printing messages created by the simulation, and also for providing error messages to help the user debug erroneous scripts.

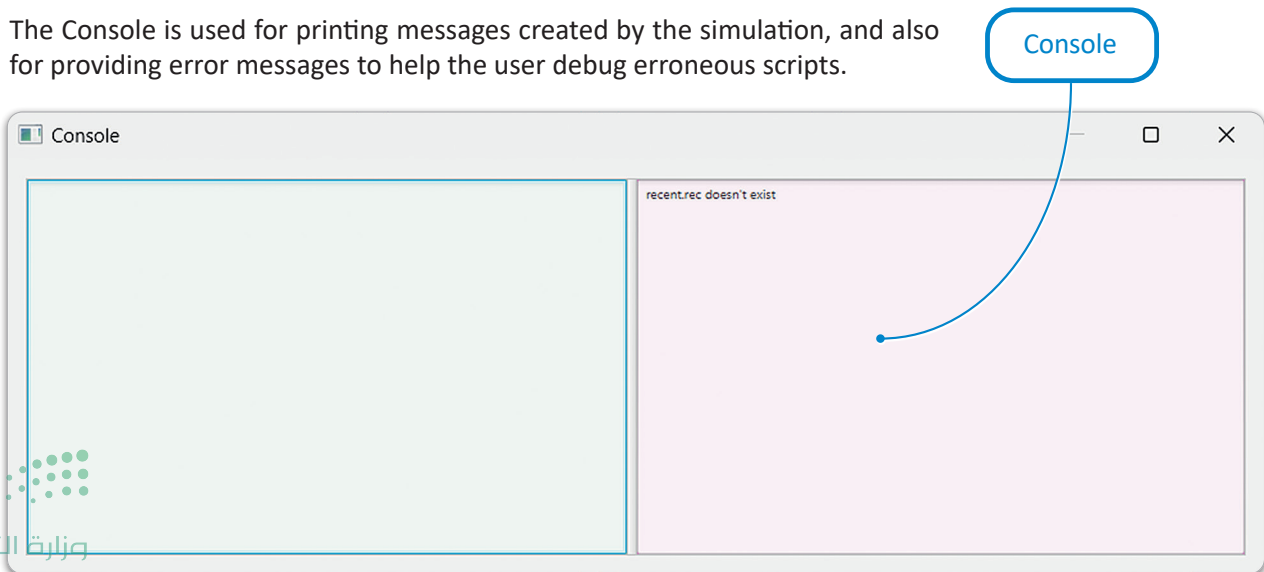


Figure 8.6: CupCarbon Console

Getting started

In this lesson, you are going to create a simple simulation of an IoT node printing messages in order to familiarize yourself with the CupCarbon environment.

First, you must create a new project:

To create a new project:

- > Click on the **New Project** button on the **Toolbar**. **1**
- > Choose your desired location for the project to be stored, type **"My First CupCarbon Simulation"** in the field **File name** **2** and click on **Save**. **3**

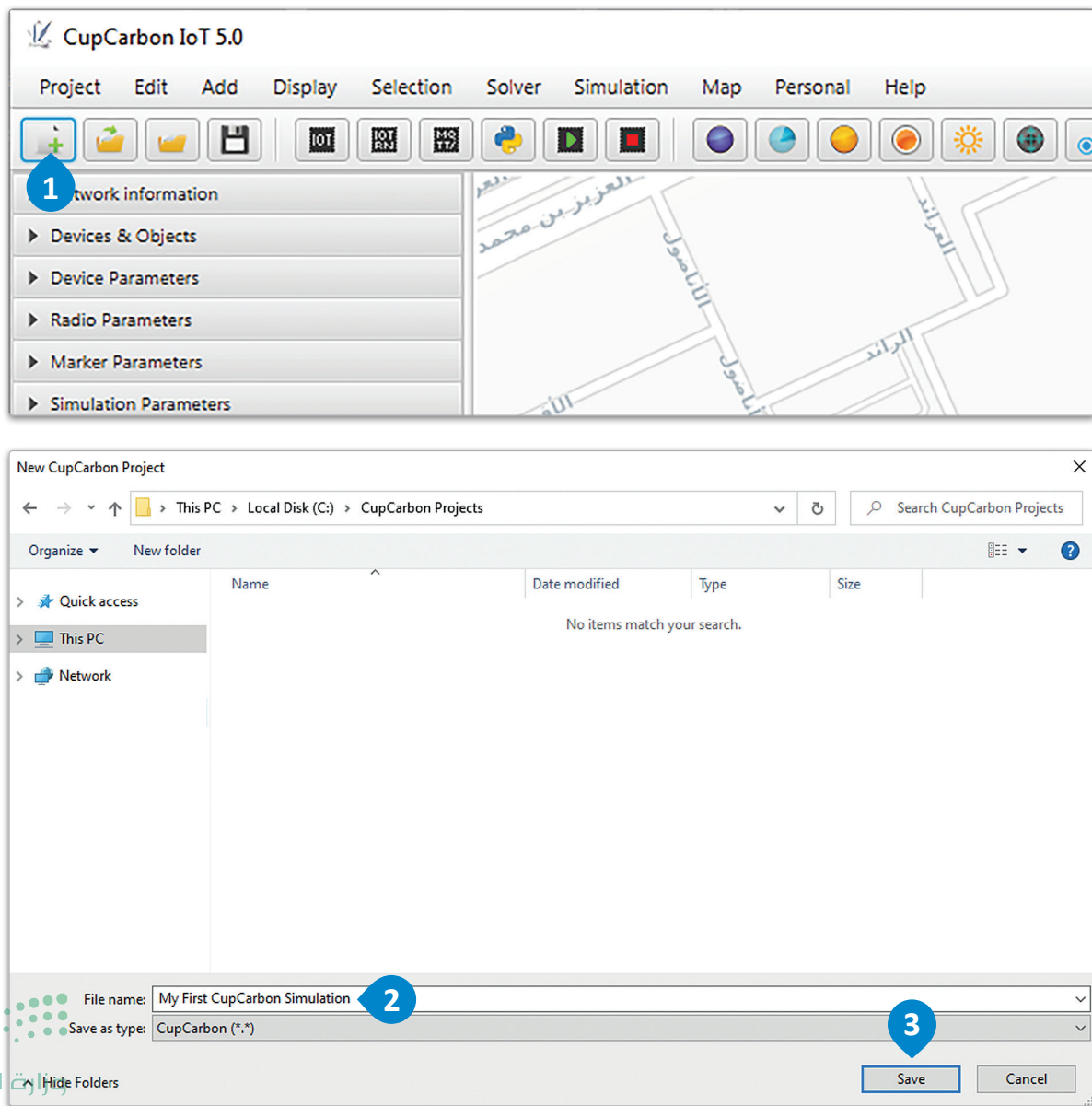


Figure 8.7: Creating a new project

Placing a node

On the toolbar you can find the various objects you are going to use in your projects, that either produce signals and communicate with each other or trigger events. One such object is the IoT Node which can be placed on the map and it can be given a Script file to run.

Nodes are the fundamental building blocks of CupCarbon. They display their ID number and have two circles around them: the inner is the sensor radius used for detecting sensors and the outer circle is used for detecting radio devices, like other nodes.

To place a node:

- > Click on the **IoT Node button** on the **Toolbar**. ①
- > Click on the **Map** to place the node. ②
- > Click on the **All button** from the **State Bar**. ③
- > Press **Esc**.

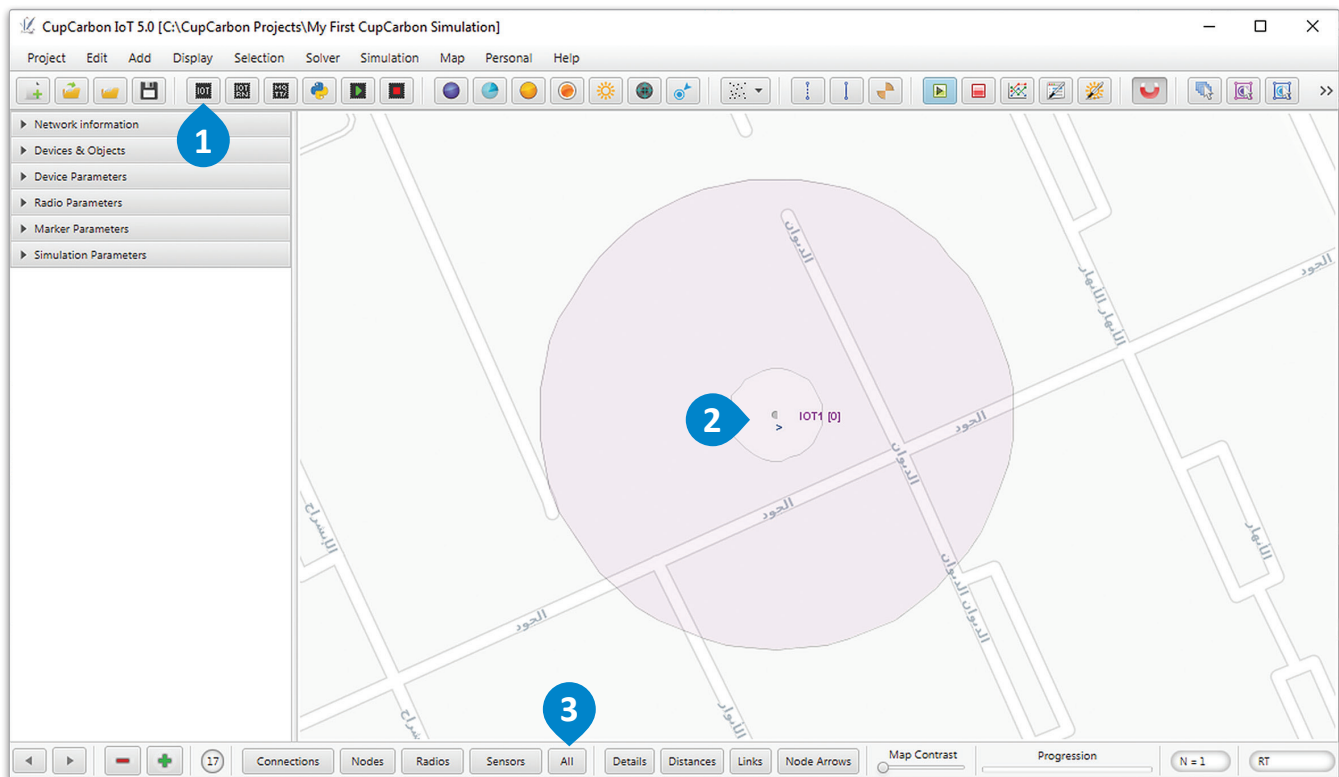


Figure 8.8: Placing a node

Creating a script

Now you will create a simple script that will be printing two messages in alternation on the node itself. The Python code you will use is the following:

Make sure to use proper indentation on the loop for the script to run properly.

```
import time
while node.loop():
    node.print("hello")
    time.sleep(1)
    node.print("world")
    time.sleep(1)
```

First you must include the Python time library, as you will be using the sleep function included to delay the prints. The actual instructions for the node have to be encapsulated inside the while node.loop():. The node can print on itself by using node.print() and it can "sleep" – do nothing – by using time.sleep(). The print() function takes as a parameter the message to be printed in string form e.g. node.print("hello world") and the sleep() function takes as a parameter a positive number, for as many seconds as you want the node to be delayed e.g. with time.sleep(3) the node will sleep for 3 seconds.

In your program, the node will print "hello", sleep for 1 second, print "world", sleep again for 1 second and then start from the top again. It will continue to run forever, unless you stop the simulation.

To create a script:

- > Click on the **Python button** on the **Toolbar**. **1**
- > Type the Python code into the field. **2**
- > Type hello in the **File name** field. **3**
- > Click **Save**. **4**
- > Close the **Python Editor** window. **5**

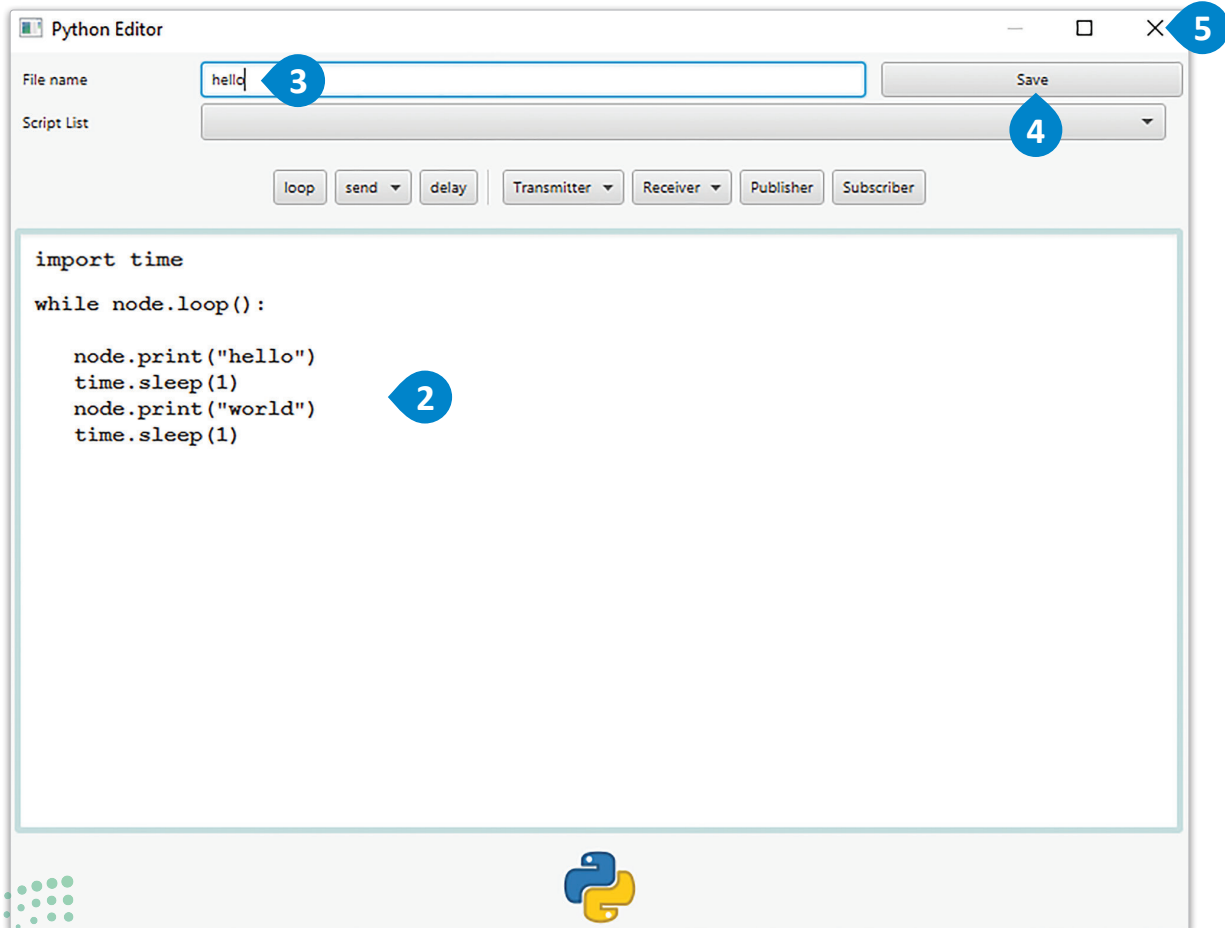
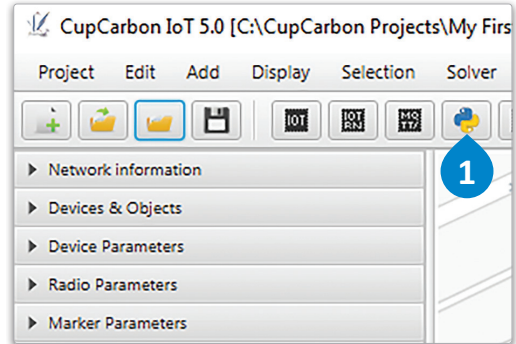


Figure 8.9: Python editor

To insert the script and run the simulation:

- > Click on the node. **1**
- > Click on the **Device Parameters** tab on the **Parameter** Menu. **2**
- > Click on the **Script file** box. **3**
- > From the drop down menu, choose the **hello.py** script and click on the button on the right to insert the script into the node. **4**
- > Click on the **Save Project** button on the Toolbar. **5**
- > On the Toolbar, click on the **Run IoT Simulation** button. **6**

- ### To insert the script and run the simulation:
- > Click on the node. **1**
 - > Click on the **Device Parameters** tab on the **Parameter** Menu. **2**
 - > Click on the **Script file** box. **3**
 - > From the drop down menu, choose the **hello.py** script and click on the button on the right to insert the script into the node. **4**
 - > Click on the **Save Project** button on the Toolbar. **5**
 - > On the Toolbar, click on the **Run IoT Simulation** button. **6**

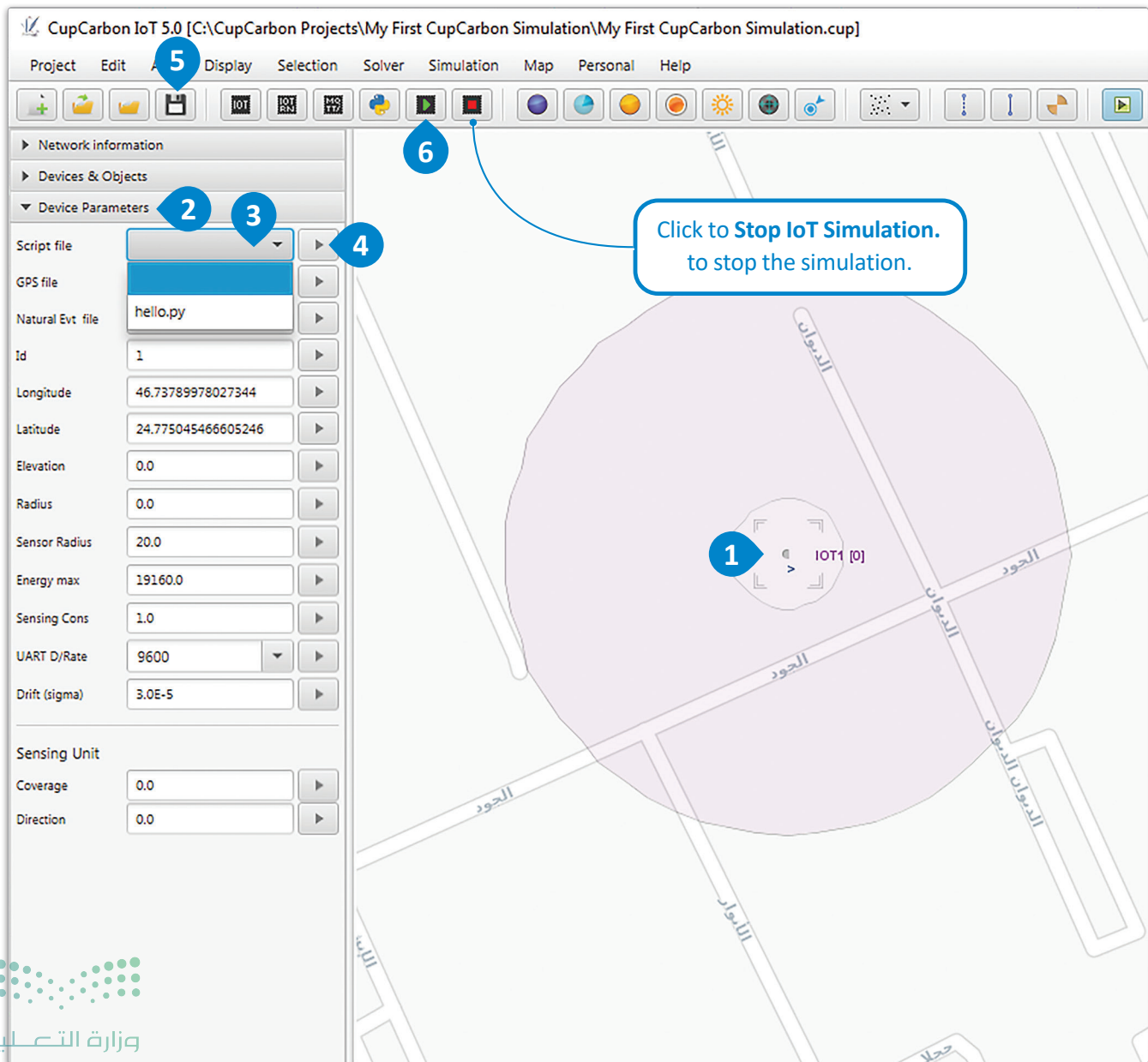


Figure 8.10: Inserting the script and running the simulation

As expected, the node alternates between printing the strings "hello" and "world" for 1 second each.



Figure 8.11: The states of the simulation

Exercises

1

Read the sentences and tick ✓ True or False.	True	False
1. Data monitoring cannot be used to enhance overall equipment effectiveness.	☐	☐
2. OT and IT departments can merge all manufacturing sectors under a single networking domain.	☐	☐
3. Connecting the devices of a factory under a single network can reduce costs.	☐	☐
4. Zero-touch automated processes in a food and beverage factory can improve the quality of the final product.	☐	☐
5. PC workstations on factory floors do not create security risks.	☐	☐
6. Devices in a factory that are not connected to an edge network can lose valuable data if they malfunction.	☐	☐
7. IoT systems in the oil and gas industries can prevent hazardous scenarios for workers.	☐	☐
8. CupCarbon can simulate the connection protocol Zigbee for smart objects.	☐	☐
9. CupCarbon nodes can be programmed exclusively with Python.	☐	☐
10. CupCarbon can produce sketches for physical microcontrollers such as Arduino.	☐	☐

2

Classify the main IoT technologies that disrupt traditional manufacturing operations.



3 Analyze how factories connected to IoT systems are vulnerable to cyberattacks.

[illegible]

4 Describe how edge computing on connected factories can improve their efficiency and production capabilities.



5 State how smart industry IoT solutions can be used to optimize processes in the oil and gas industry.

This image shows a single sheet of white paper with horizontal ruling lines. The lines are evenly spaced and run across the width of the page. There are no margins, text, or other markings on the paper.

6 In CupCarbon, create a node and a Python script for it. This script will have a loop that will make the node print the message "A" for 1 second, "B" for 2 seconds, and "C" for 3 seconds. Program the node and run the simulation.

7 In CupCarbon, create two nodes and a Python for each one. Each script will display a repeated "blink" message. Each node will send the message when the other is inactive. Program the nodes and run the simulation.



Lesson 2

Communication in an IoT Network

Link to digital lesson



www.iien.edu.sa

Communication between Devices

An IoT network consists of many devices that send and receive data between each other. These devices have different capabilities, such as different range, data bandwidth and power consumption, and thus run different sets of commands to provide a multitude of functions. In the following project, you will build such a network and explore the various fundamental blocks it consists of.

Fire Surveillance and Notification

In this lesson, you are going to create a project that simulates a fire surveillance system in a factory. The simulation will create random fires in the factory and the system will inform the factory's main control unit about which sector the fire is in. This will be achieved by using a variety of nodes, with different functions, that will communicate with each other to pass the messages. It will start from the outer nodes, the "edges", pass through the middle ones, the "proxies", to finally arrive at the inner node, the "main controller".

The Nodes and their Functions

To represent the different sectors of a factory, you will use three kinds of nodes in your project:

- In the edges a fire may occur, determined by a random number generator. In such cases, the node will print a message on itself and send a different message containing the sector number to its adjacent proxy node and then sleep for a specific interval.
- The proxies read any messages they may have received and forward them to the main controller. They also print a message on themselves to inform the user of their action.
- The main controller likewise reads any messages it may have received and prints the messages created by the edge nodes that inform the user of the fire.

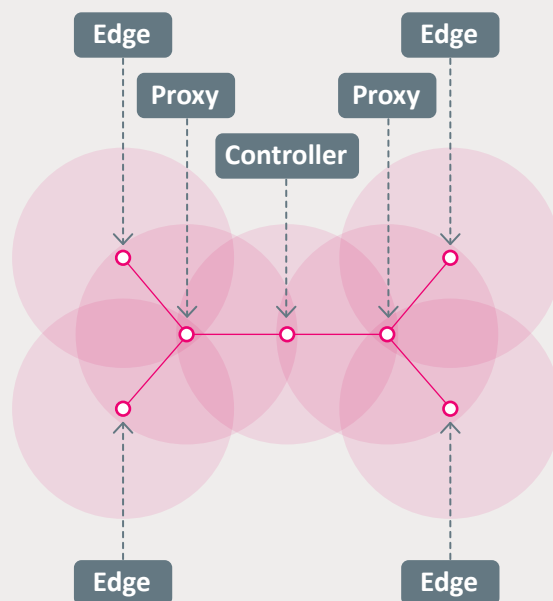


Figure 8.12: Nodes and their functions



Let's start by creating a new project:

To create a new project:

- > Click on the **New Project** button on the Toolbar. ①
- > Choose your desired location for the project to be stored, type "**Fire Surveillance and Notification**" in the field **File name** ② and click on **Save**. ③

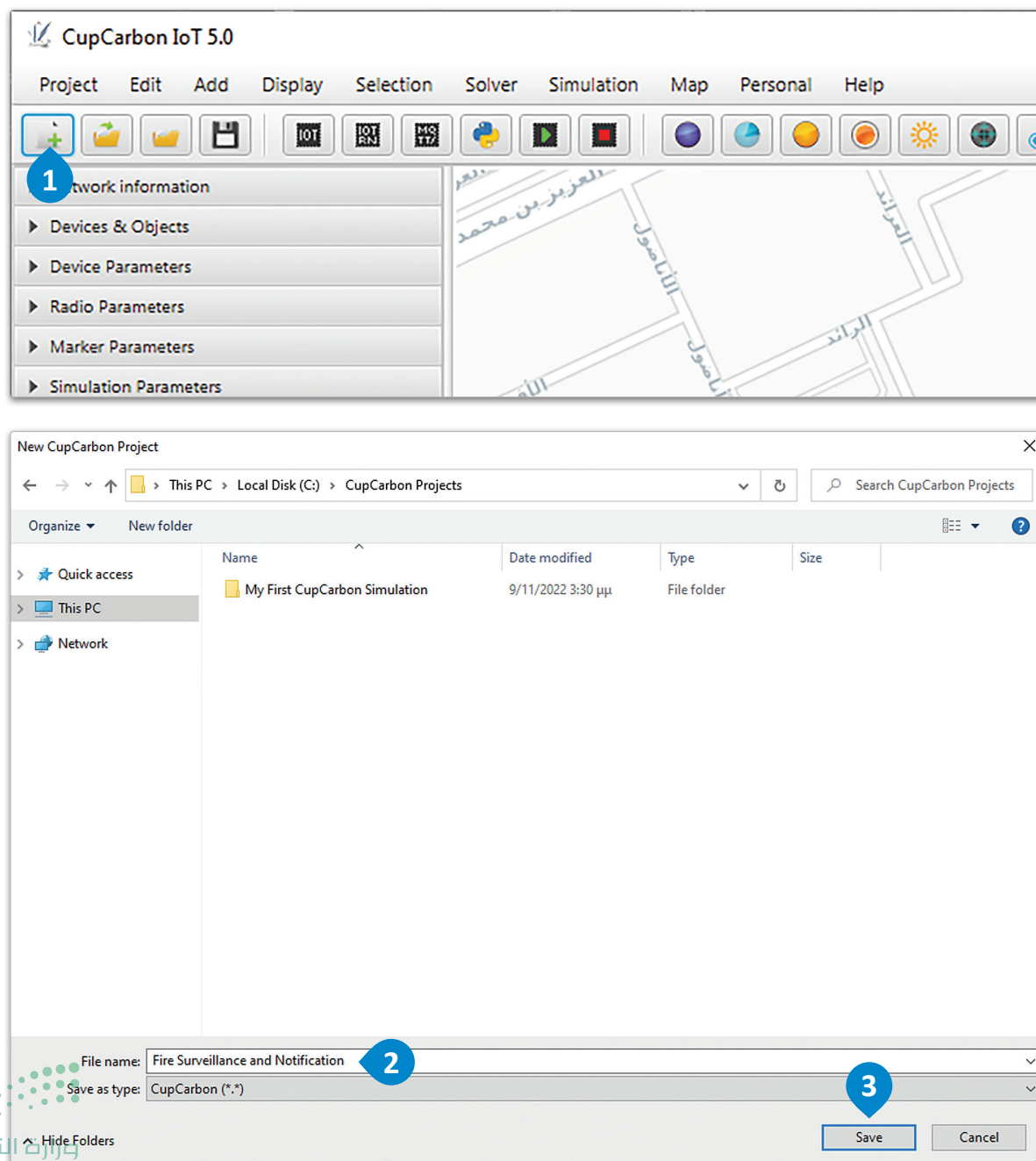


Figure 8.13: Creating a new project

Begin building the node network by placing the main controller and the proxies:

To place the controller and proxy nodes:

- > Click on the **IoT Node** button on the Toolbar. **1**
- > Click on the **Map** to place the node. **2**
- > Click on the **All** button from the **State Bar**. **3**
- > Place two more nodes on the **left** and **right** of the first one, inside its outer circle. **4**
- > Press **Esc**.

If the nodes are not placed inside the controller's radius, they will not be able to communicate. If this is the case, drag and drop them closer to the controller, until a two-directional arrow appears connecting the two nodes.

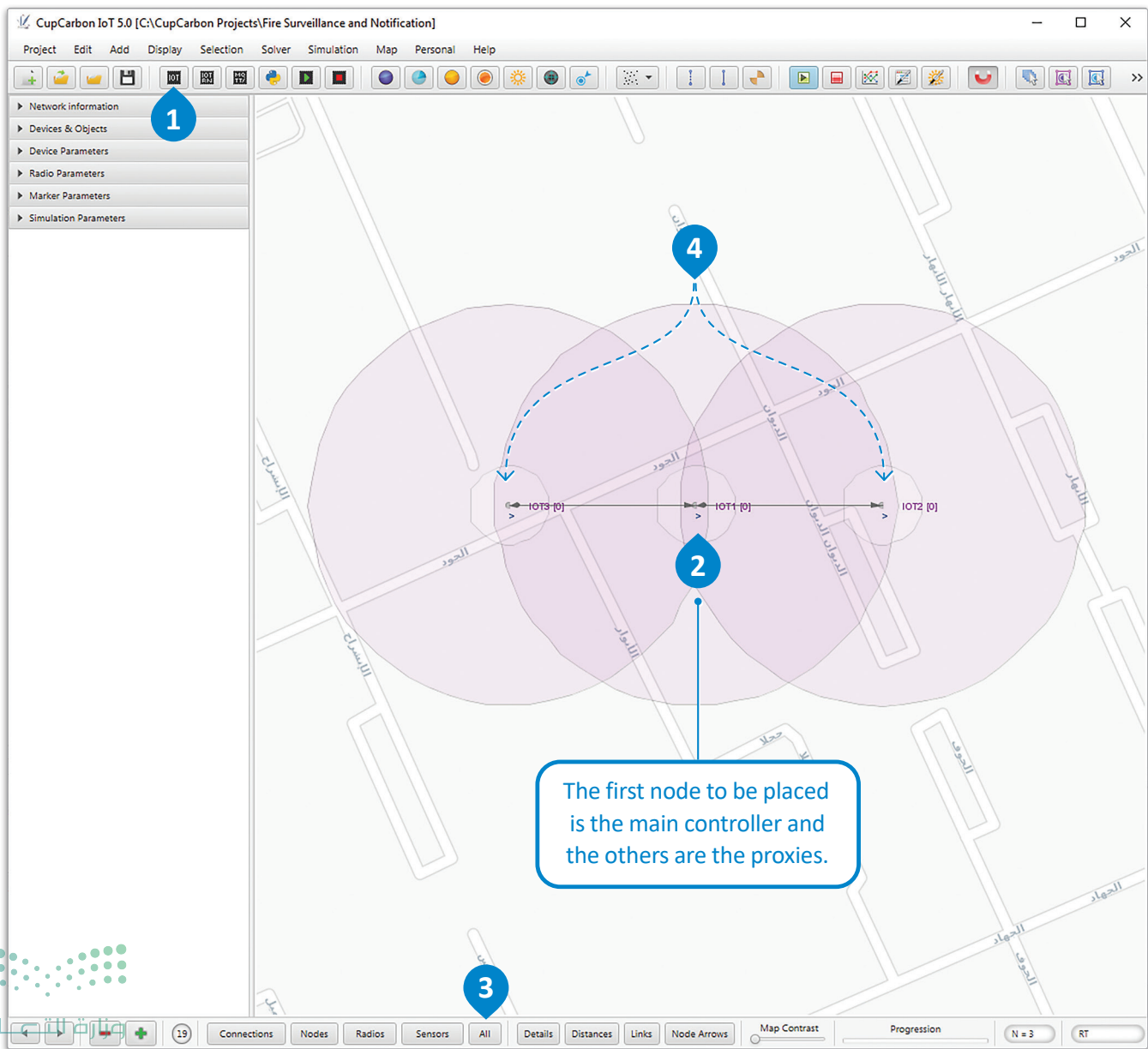


Figure 8.14: Placing the controller and proxy nodes

Continue by adding the edge nodes:

To place the edge nodes:

- > Click on the **IoT Node** button on the Toolbar. **1**
- > Place two nodes on each proxy node, inside its outer circle and not in the range of any other node. **2**
- > Click on the **All** button from the **State Bar**. **3**
- > Press **Esc**.

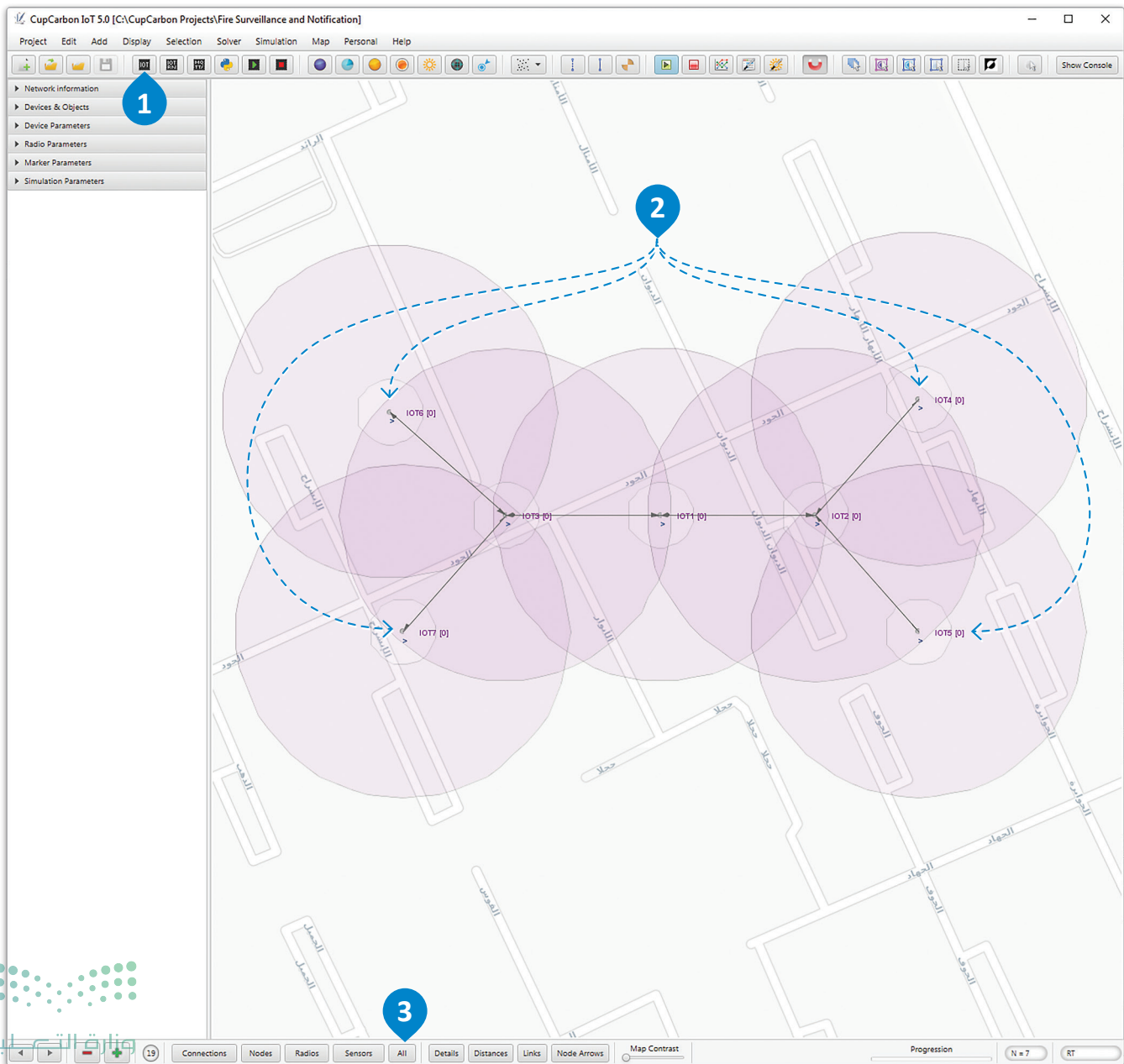


Figure 8.15: Placing the edge nodes

Creating the Scripts

You will now take a look at the scripts that the nodes will be running. Let's start with the edges' script.

First, include the necessary libraries.

```
import time
import random
```

The `randint()` function takes as argument two integers and returns an integer from the range which includes the two integers. For example, in our case `randint(1, 6)` randomly creates an integer from 1 to 6. This will be used as the random chance a sector may catch fire in each time interval. The integer will be stored in the variable `fire`.

```
while node.loop():
    fire = random.randint(1, 6)
```

Consider that when the random number generator returns 1, the sector will have caught fire. The script will check if the variable `fire` is equal to 1 and if so, it will run a series of commands. This includes printing on the node itself the message "FIRE!" and sending a message containing its sector ID to the proxy node it is adjacent to.

The sector's ID is the same as the node's ID number, the integer that makes it unique. If the sector's ID is 5, the message sent will be "FIRE IN SECTOR 5". The node's ID and therefore the sector's can be returned by the function `id()`. The ID is returned as a number, so it has to be "casted" as a string first, in other words converted to the string type, before being concatenated with the remaining message.

```
if fire == 1:
    node.print("FIRE!")
    message = "FIRE IN SECTOR " + str(node.id())
```

The nodes can send data to each other using the `send()` function. With only one argument, the function takes a string and broadcasts it to all the nodes inside its range.

```
node.send(message)
```



If the random number generator produces any other integer, in our case from 2 to 6, there is no fire in the sector and the node simply has to print an empty string on itself to clear any previous prints.

```
else:  
    node.print("")
```

Finally, the node will sleep for a random time interval, to more closely simulate the real life randomness of events and incidents. This will be achieved by using the `uniform()` function that works like the `randint()` function but produces real numbers and not only integers. In your project, the time intervals will be between 1 and 4 seconds.

```
time.sleep(random.uniform(1, 4))
```

Complete Code (edge.py)

```
import time  
import random  
  
while node.loop():  
  
    fire = random.randint(1, 6)  
  
    if fire == 1:  
        node.print("FIRE!")  
        message = "FIRE IN SECTOR " + str(node.id())  
        node.send(message)  
    else:  
        node.print("")  
  
    time.sleep(random.uniform(1, 4))
```



Next is the proxies' script.

When a node receives data, the data is stored to its buffer until read, thus initially the buffer's size must be checked to be greater than zero, meaning it is not empty. The buffer's size can be returned by the function `bufferSize()`.

```
import time

while node.loop():

    if node.bufferSize() > 0:
```

Then, the data received can be read using the function `read()`. After the message is read, it is stored to the variable `message`. The node also prints an informative message "FORWARDING..." to say that it forwards the message to the main controller.

```
    message = node.read()
    node.print("FORWARDING...")
```

As used in the previous script, the message stored in the variable will be sent to the main controller with the function `send()`. This time though, apart from the message it will take an extra argument, the receiving node's ID. Because the message is specific to one node with a known ID, the message does not have to be broadcasted, it can instead be unicasted (sent to only one node). In our case, the main controller was placed first, and thus has an ID equal to 1.

```
    node.send(message, 1)
```

Next, the node will sleep for 1 second, so as to give the user time to read the informative message printed on the node and then the node clears the message by printing an empty string.

```
    time.sleep(1)
    node.print("")
```

The script ends with the node sleeping for a very small interval (one hundredth of a second), so it can be responsive in case it receives a lot of data.

```
    time.sleep(0.01)
```



Complete Code (proxy.py)

```
import time
while node.loop():

    if node.bufferSize() > 0:
        message = node.read()
        node.print("FORWARDING...")
        node.send(message, 1)
        time.sleep(1)
        node.print("")

    time.sleep(0.01)
```

The number parameter in the send() function is the ID number of the controller node.

The main controller's script bears some resemblance to the proxies'. It also checks the buffer and reads the message received, but the message it prints on itself is the one originally created by the edge node.

```
import time
while node.loop():

    if node.bufferSize() > 0:
        message = node.read()
        node.print(message)
```

Then, like the proxy nodes, the main controller sleeps, but for 2 seconds this time and prints an empty string. Finally, it sleeps for a small interval, in the same fashion as the proxies.

```
time.sleep(2)
node.print("")
```

```
time.sleep(0.01)
```



Complete Code (controller.py)

```
import time
while node.loop():

    if node.bufferSize() > 0:
        message = node.read()
        node.print(message)
        time.sleep(2)
        node.print("")

    time.sleep(0.01)
```

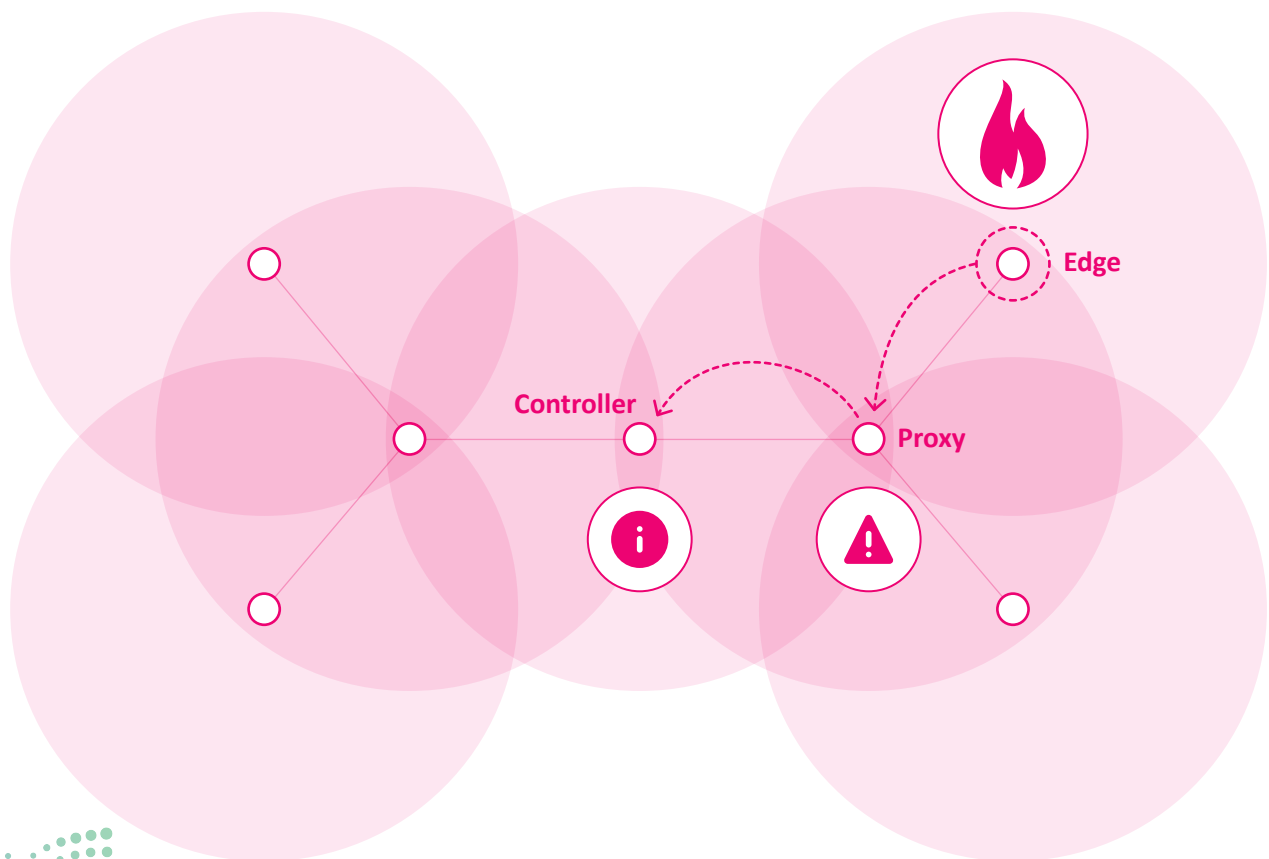


Figure 8.16: Complete network

Now that you have seen what the scripts do, go on and create them.

To create and apply the script for the controller node:

To create a script:

- > Click on the **Python** button on the Toolbar. **1**
- > Type the Python code into the field. **2**
- > Type "controller" in the **File name** field. **3**
- > Click **Save**. **4**
- > Close the **Python Editor** window. **5**

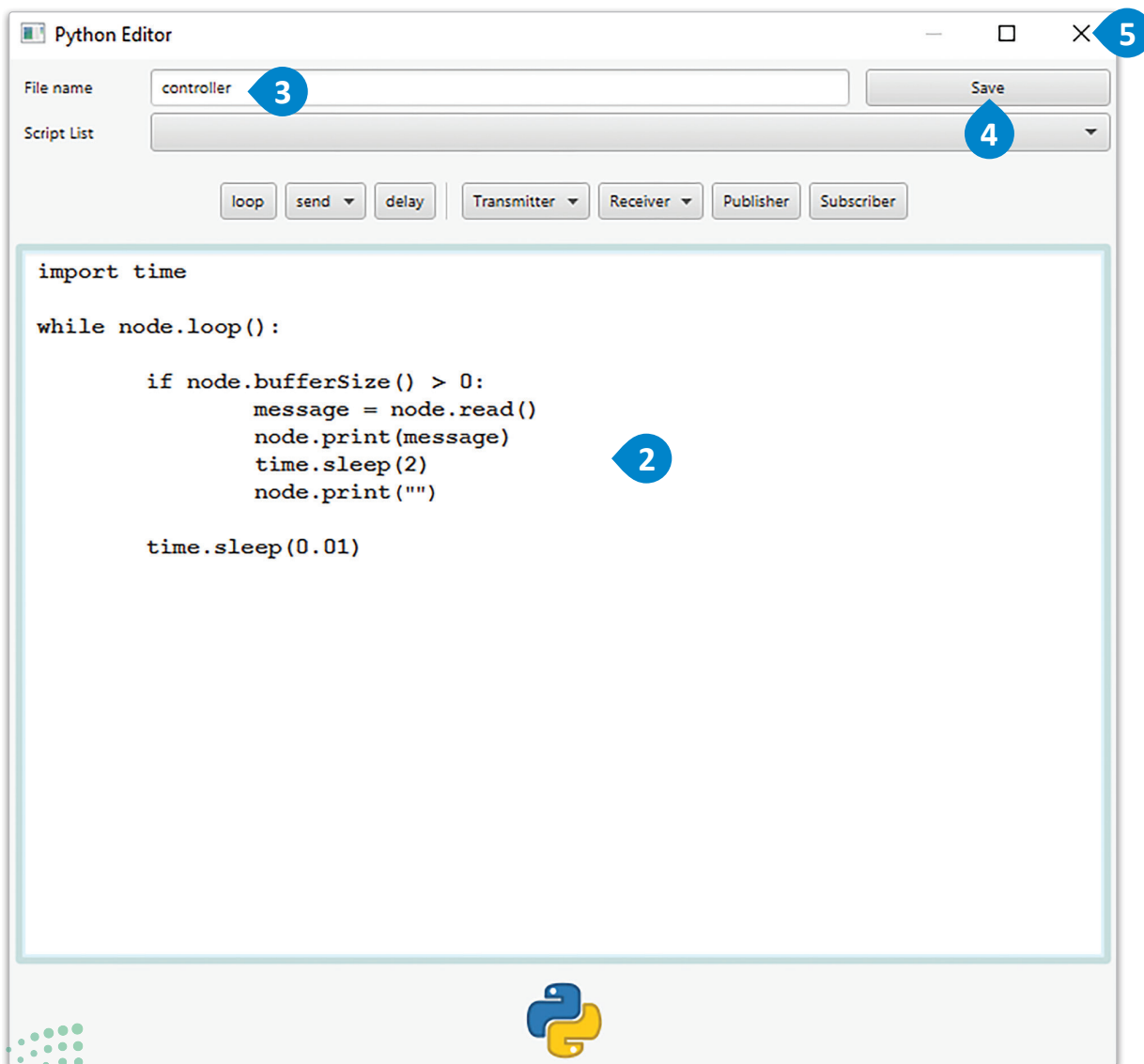
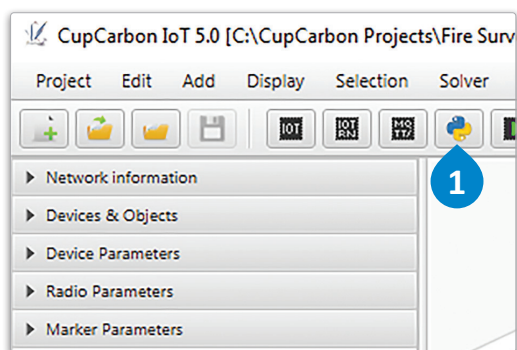


Figure 8.17: Create the script



To insert the script:

- > Click on the node. **1**
- > Click on the **Device Parameters** tab on the **Parameter Menu**. **2**
- > Click on the **Script file** box. **3**
- > From the drop down menu, choose the **controller.py** script and click on the button on the right to insert the script into the node. **4**
- > Click on the **Save Project** button on the Toolbar. **5**

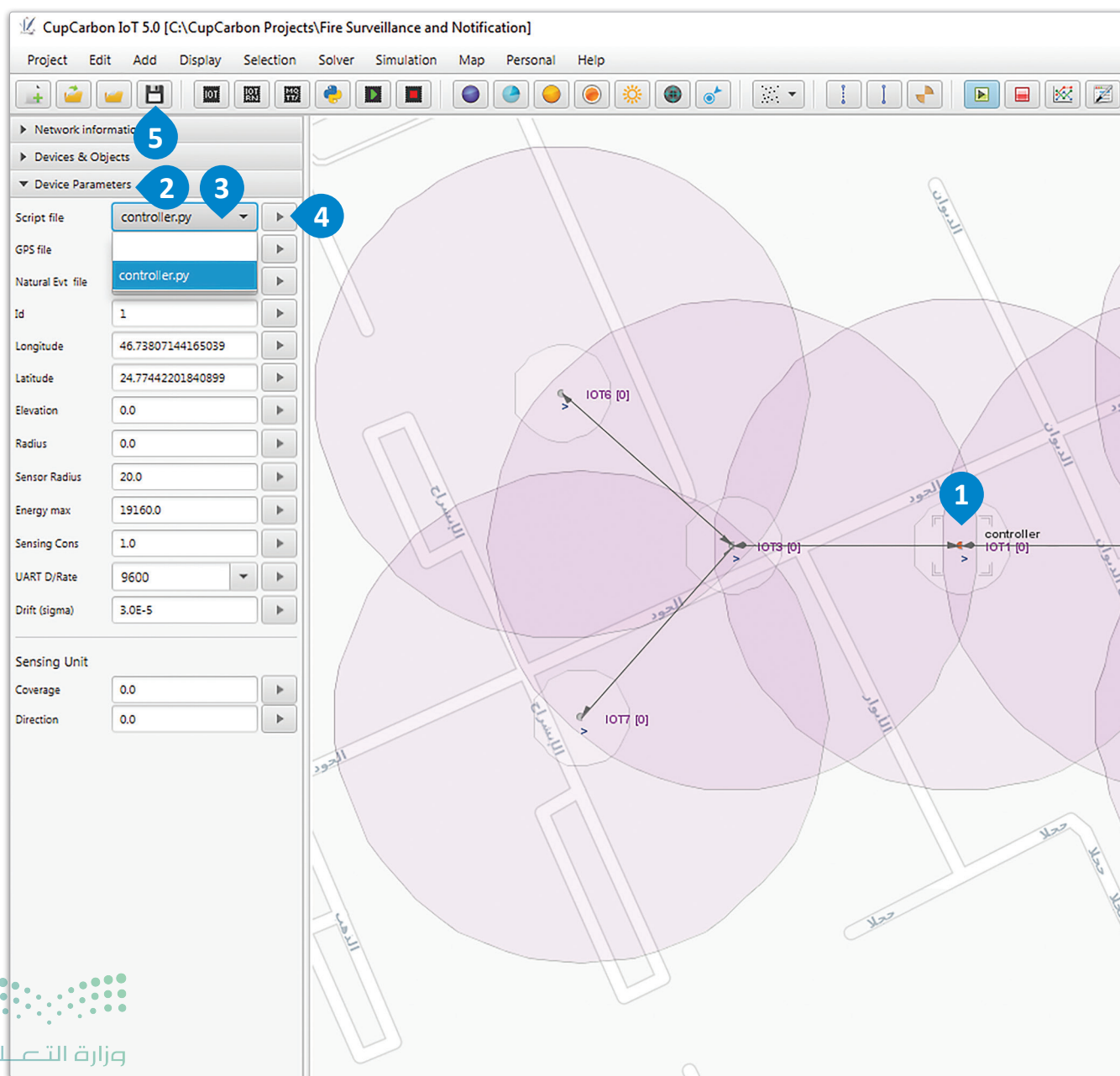


Figure 8.18: Inserting the script

When you are done, click the Run IoT Simulation button from the toolbar. Notice that because you used the random number generators, some sectors on the edges may catch fire more than others which may not catch fire at all.



وزارة التعليم
Ministry of Education
2026 - 1448

Exercises

- 1 Extend your project to support an additional "edge" node on each "proxy" node, so that each "proxy" has 3 "edges". Do not forget to insert scripts into the new nodes.
- 2 Extend your project to support an additional "proxy" node and add to this "proxy" 2 new "edge" nodes, so that the "main controller" has 3 "proxies" and each "proxy" has 2 "edges". Do not forget to insert scripts into the new nodes.
- 3 State which code section in the scripts influences the frequency that the fires occur. Then, in CupCarbon, modify your project so that fires are more likely to occur than before.
- 4 If there is latency in the factory's network, communication between nodes may be delayed. Modify the "proxy" nodes' script to make the nodes sleep for longer. Are any messages delayed and/or lost? Present your observations below.

- 5 Extend your project to also support floods. Modify the script of the sectors that are susceptible to fires, so when the random function returns the value 2, it will mean that a flood has occurred in that sector and the node will print and send the appropriate message.



Smart Industry and Automation

Automation is a significant advantage of modern technology and a major contributing factor to the Fourth Industrial Revolution. Smart Industry is enhanced by automation technologies which upscale business production, leading to larger profits. In the following project you are going to create a simulation of a system that will check a factory's storage area for containers with consumable items that will decay if left outside of refrigeration overnight, using an automated vehicle.

The automated inspector vehicle follows a predetermined route through the factory storage area, labeling the item containers according to their contents; consumables and non-consumables. Each container has an IoT tag that emits a message using its radio, and informs the vehicle of its contents. There are also some charging stations throughout the storage area to charge the vehicle's battery, as it decreases every time it moves.



Figure 8.20: Automated Industry Vehicle

Let's start by creating a new project:

To create a new project:

- > Click on the **New Project** button on the Toolbar. ①
- > Choose your desired location for the project to be stored, type **"Storage Product Marking"** in the field **File name** ② and click on **Save**. ③

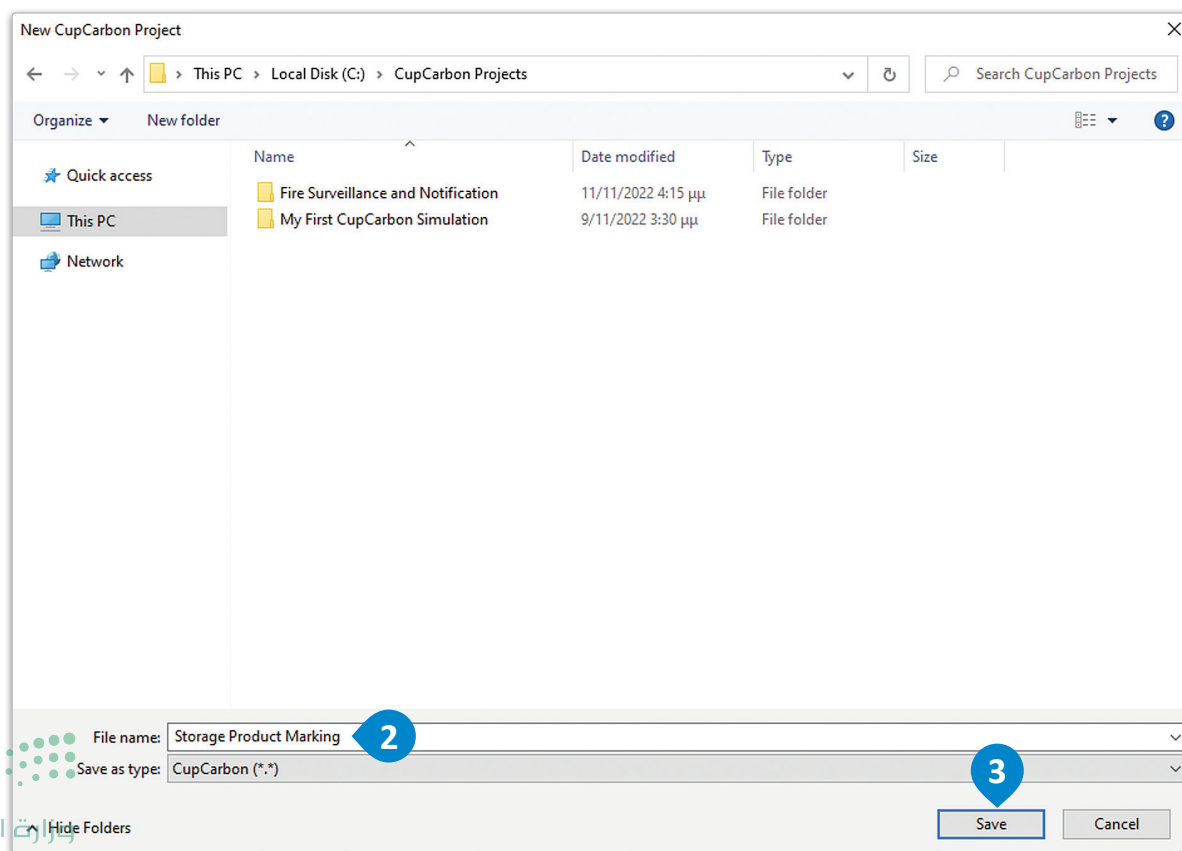
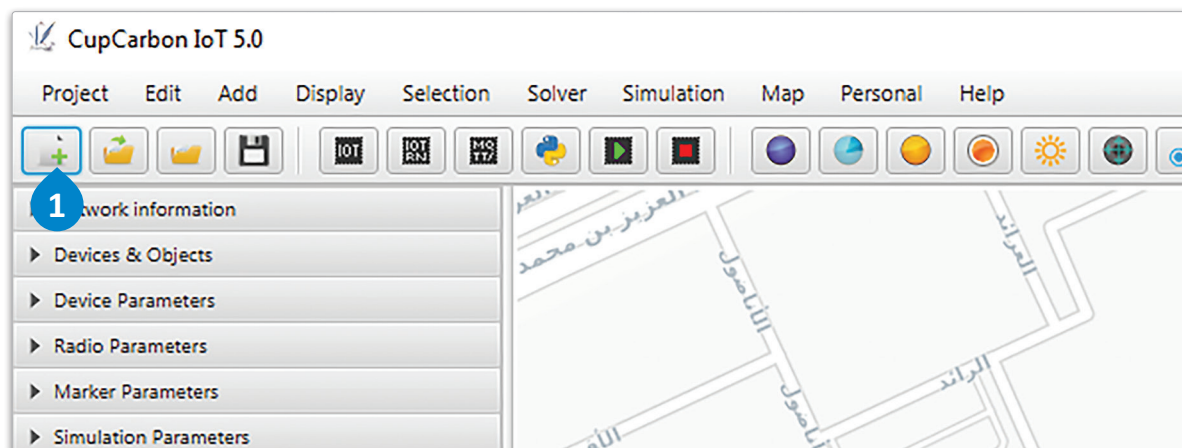


Figure 8.21: Creating a new project

Creating a predetermined route

First, you will create the route along which the inspector vehicle will move. You will initially add some markers on the map to define the route structure and then enrich it with some more markers.

To create the route:

- > Click on the **Marker** button on the Toolbar. **1**
- > Click on the map **5 times**, in a similar manner to the picture, creating lines along the aisles. **2**
- > Press **Esc**.
- > Click on the **Marker Parameters** tab on the parameter menu. **3**
- > Add more markers to the route by clicking on each of the first 4 markers placed and clicking **Insert Markers** 4 times. **4**
- > Click **Save**. **5**

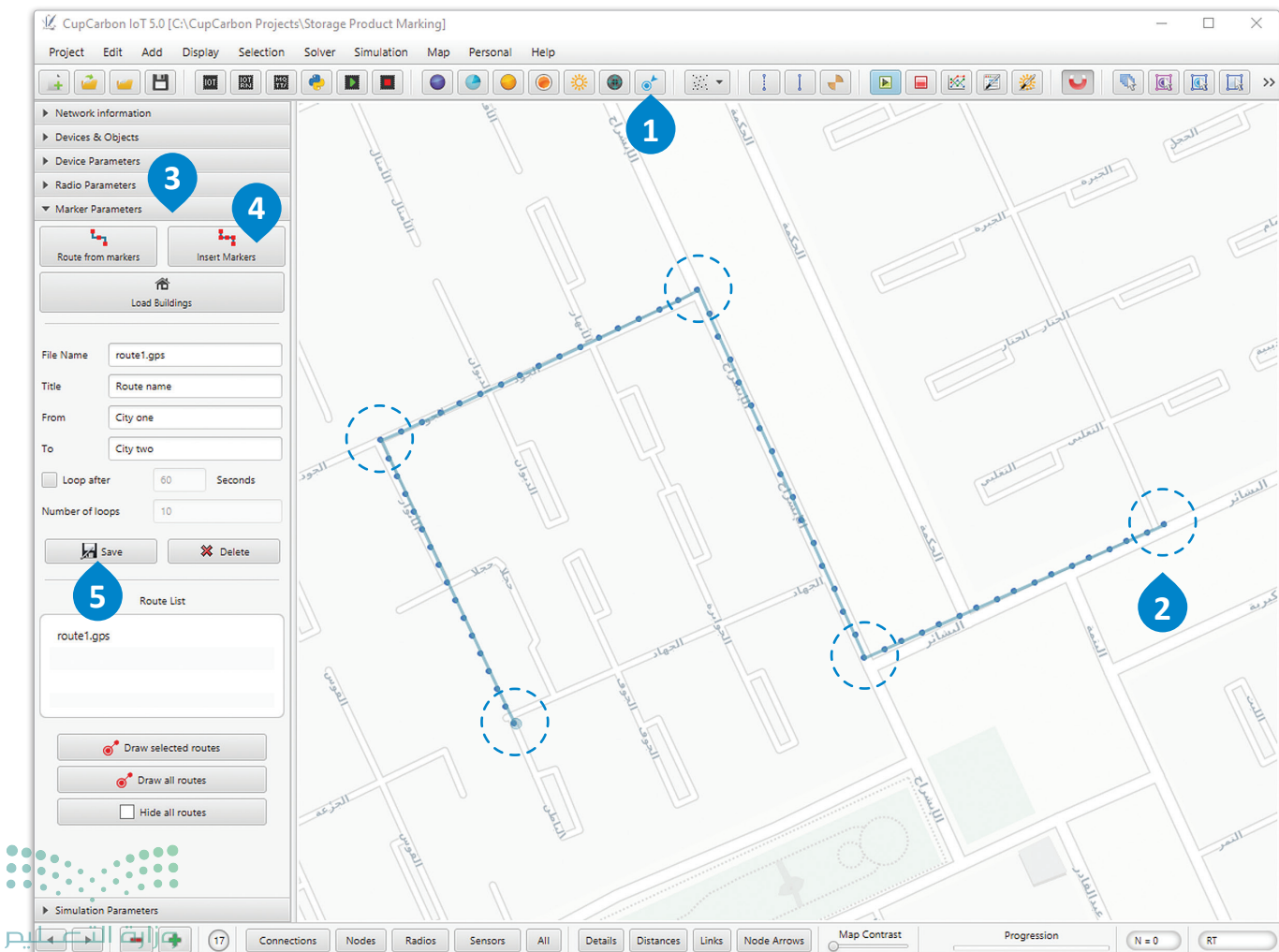


Figure 8.22: Creating a route

Adding the inspector vehicle node

The inspector vehicle will be simulated by a node moving along the route you just created. You will also increase the sensor radius of the node (inner circle) so that it can reach the charging station's range.

To add the inspector vehicle node:

- > Click on the **IoT Node** button on the Toolbar. **1**
- > Click on the **map** to place a node. **2**
- > Click on the **All** button from the **State bar**. **3**
- > Press **Esc**.
- > Click on the node and press **Shift** + **0** four times to increase the sensor radius.
- > Click on the **Device Parameters** tab on the parameter menu. **4**
- > Click on the box to the right of **GPS file**. **5**
- > From the drop down menu, choose the **route1.gps** script and click on the button on the right to insert the route into the node. **6**

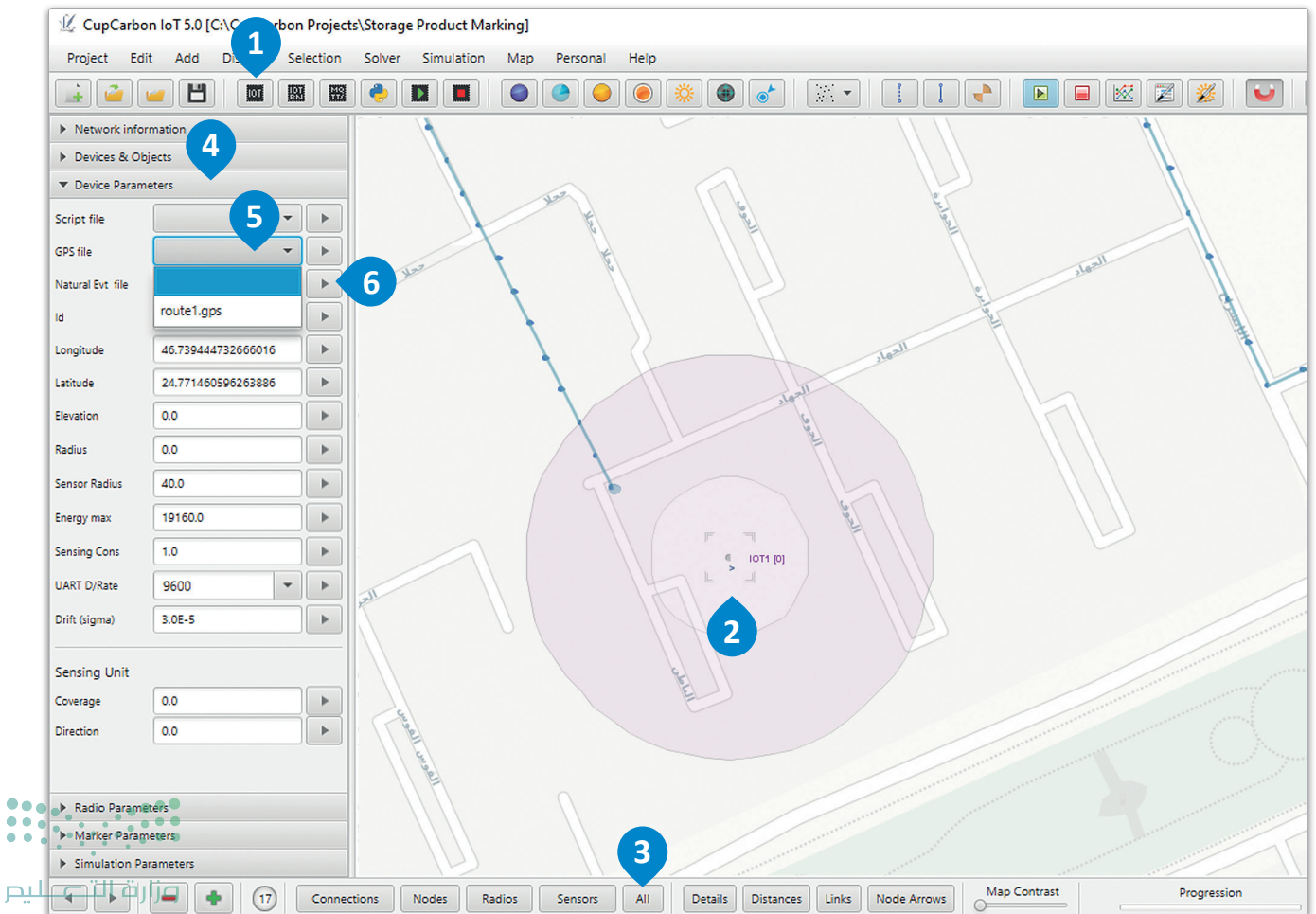


Figure 8.23: Creating the inspector vehicle node

Adding container nodes

Now it is time to add the nodes representing the containers.

To add the container nodes:

- > Click on the **IoT Node** button on the Toolbar. **1**
- > Click on the **map** and place 7 nodes near the route, so that their radio radius includes at least 1 marker from the route. **2**
- > Click on the **All** button from the **State bar**. **3**
- > Press **Esc**.

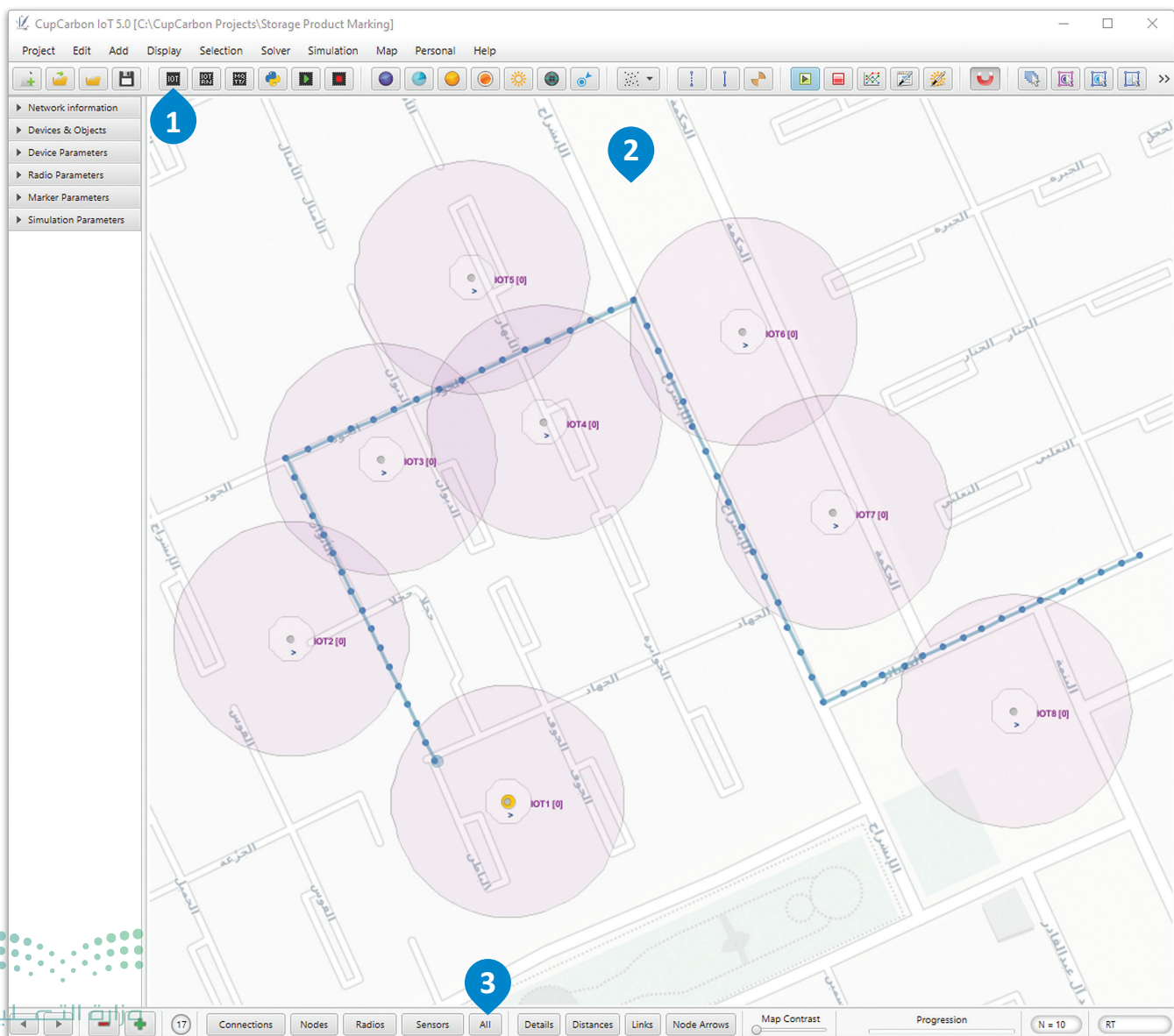


Figure 8.24: Creating the container nodes

Adding charging stations

As the inspector vehicle moves, it consumes energy and thus must be charged. You will place a couple of charging stations on the route that will recharge the vehicle's battery when it passes near them. For this purpose the inner radius, the sensor range, will be used this time.

To add the charging station nodes:

- > Click on the **Mobile** button on the Toolbar. **1**
- > Click on the **map** and place 2 nodes spread out along the route, so that the vehicle's sensor radius can reach them. **2**
- > Press **Esc**.

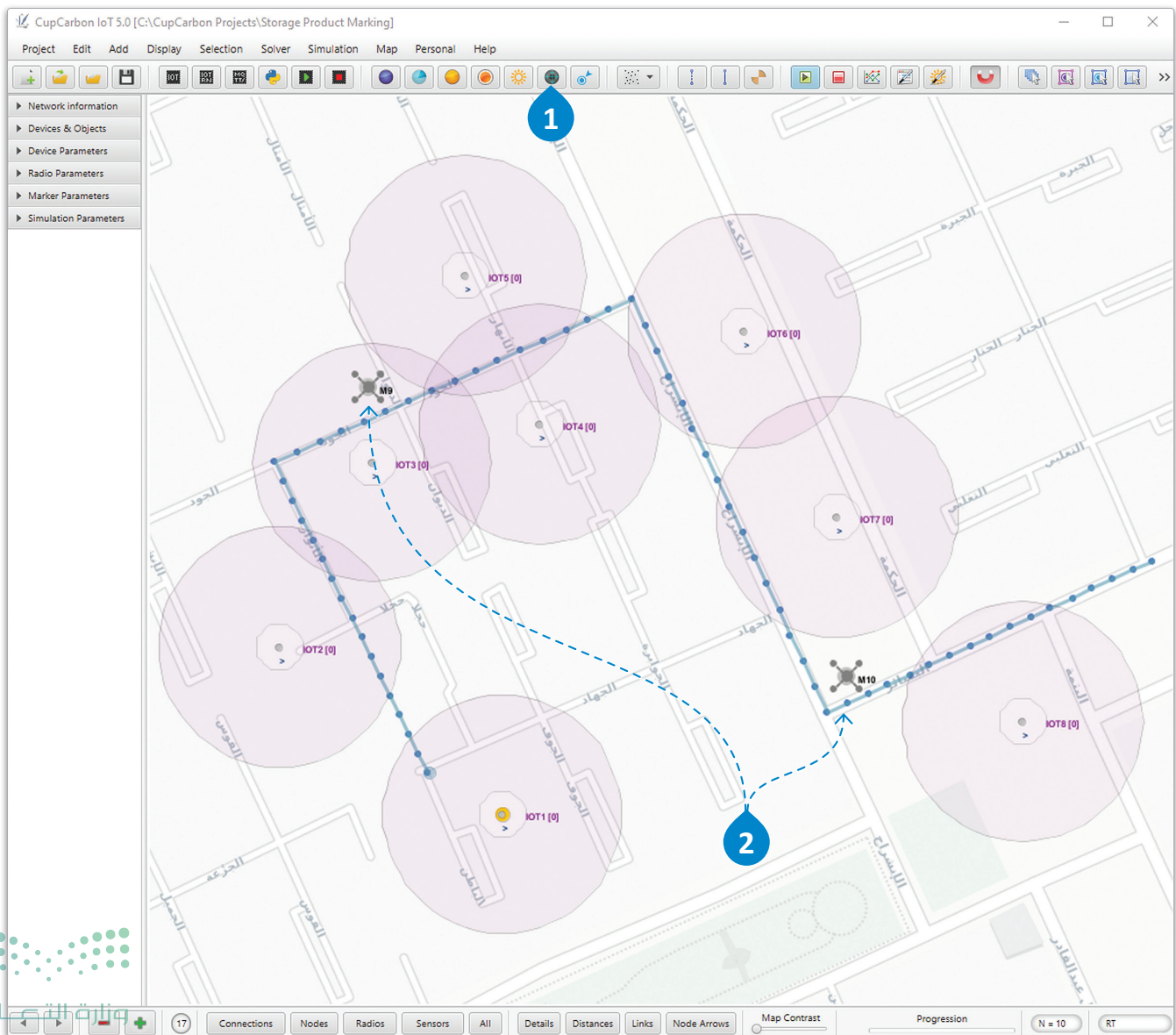


Figure 8.25: Creating the charging station nodes

Creating the Scripts

Let's now take a look at the scripts you will be using, starting with the code for the containers. The two types of containers will have similar scripts.

Start by including the library and printing an empty string on the node to remove any prints from previous executions.

```
import time

node.print("")
```

The consumables containers will be broadcasting a message, which will include its contents and ID, so it can be used by the inspector vehicle to categorize each container. The ID is an integer and has to be converted ("casted") into a string before being concatenated to the string. A space is placed between the contents info and the ID as only one string can be sent at one time with the send() function and you have to send two pieces of information here; the space will be used as a separator.

```
while node.loop():

    node.send("CONSUMABLES " + str(node.id()))
```

After the container analyzes the string it will either send back "1" meaning the containers must be picked or send "2" meaning it must not. In its turn the container prints on itself "PICK" or "DO NOT PICK" and then it sleeps for 1 second.

```
message = node.read()
if message == "1":
    node.print("PICK")
elif message == "2":
    node.print("DO NOT PICK")

time.sleep(1)
```



Complete Code (consumables.py)

```
import time

node.print("")
while node.loop():

    node.send("CONSUMABLES " + str(node.id()))
    message = node.read()
    if message == "1":
        node.print("PICK")
    elif message == "2":
        node.print("DO NOT PICK")

    time.sleep(1)
```

The difference in the code between the consumables and non-consumables is the string they send. In the non-consumables' script, the string "CONSUMABLES" becomes "NONCONSUMABLES".

Complete Code (nonconsumables.py)

```
import time

node.print("")
while node.loop():

    node.send("NONCONSUMABLES " + str(node.id()))
    message = node.read()
    if message == "1":
        node.print("PICK")
    elif message == "2":
        node.print("DO NOT PICK")

    time.sleep(1)
```



Next is the inspector vehicle's script. At the start, the battery is initialized by setting its max energy to 100 energy units with the `battery.setEMax()` function and then setting its current level to max with the `battery.init()` function.

```
import time

node.battery.setEMax(100.0)
node.battery.init()
```

As each time interval passes, the vehicle will consume a certain amount of energy. To simulate this, use the `battery.consume(1.0)` function to consume 1 energy unit for each time interval.

```
while node.loop():

    node.battery.consume(1.0)
```

To detect if any charging station is within range of the vehicle, use `isSensorDetecting()` function and if one is detected, use `battery.init()` to fill its energy to the max.

```
if node.isSensorDetecting():
    node.battery.init()
```

Now, the vehicle must check all the messages it has received and answer to their senders, the containers. First the read string is stored in the local variable `recvMsg` and then using the `split()` function the concatenated string is divided into two strings, according to the space used earlier, in the form of an array with the name `splitMsg`. This means that in the first cell of the array, `splitMsg[0]`, holds the contents of the container and the second cell, `splitMsg[1]`, holds the container's ID.

```
for n in range(node.bufferSize()):
    recvMsg = node.read()
    splitMsg = recvMsg.split()
```

If the contents string is "CONSUMABLES" it sends the string "1" with the `send()` function to the sender container using its ID. Else if it is "NONCONSUMABLES" it sends the string "2". In the end it sleeps for 200 ms, as it needs to be more responsive than the container nodes, because it communicates with more nodes.



```

if splitMsg[0] == "CONSUMABLES":
    node.send("1", int(splitMsg[1]))
elif splitMsg[0] == "NONCONSUMABLES":
    node.send("2", int(splitMsg[1]))

time.sleep(0.2)

```

Complete Code (inspector.py)

```

import time

node.battery.setEMax(100.0)
node.battery.init()

while node.loop():

    node.battery.consume(1.0)

    if node.isSensorDetecting():
        node.battery.init()

    for n in range(node.bufferSize()):
        recvMsg = node.read()
        splitMsg = recvMsg.split()
        if splitMsg[0] == "CONSUMABLES":
            node.send("1", int(splitMsg[1]))
        elif splitMsg[0] == "NONCONSUMABLES":
            node.send("2", int(splitMsg[1]))

time.sleep(0.2)

```



To create a script:

- > Click on the **Python** button on the Toolbar. 1
- > Type the Python code into the field. 2
- > Type "inspector" in the **File name** field. 3
- > Click **Save**. 4
- > Close the **Python Editor** window. 5

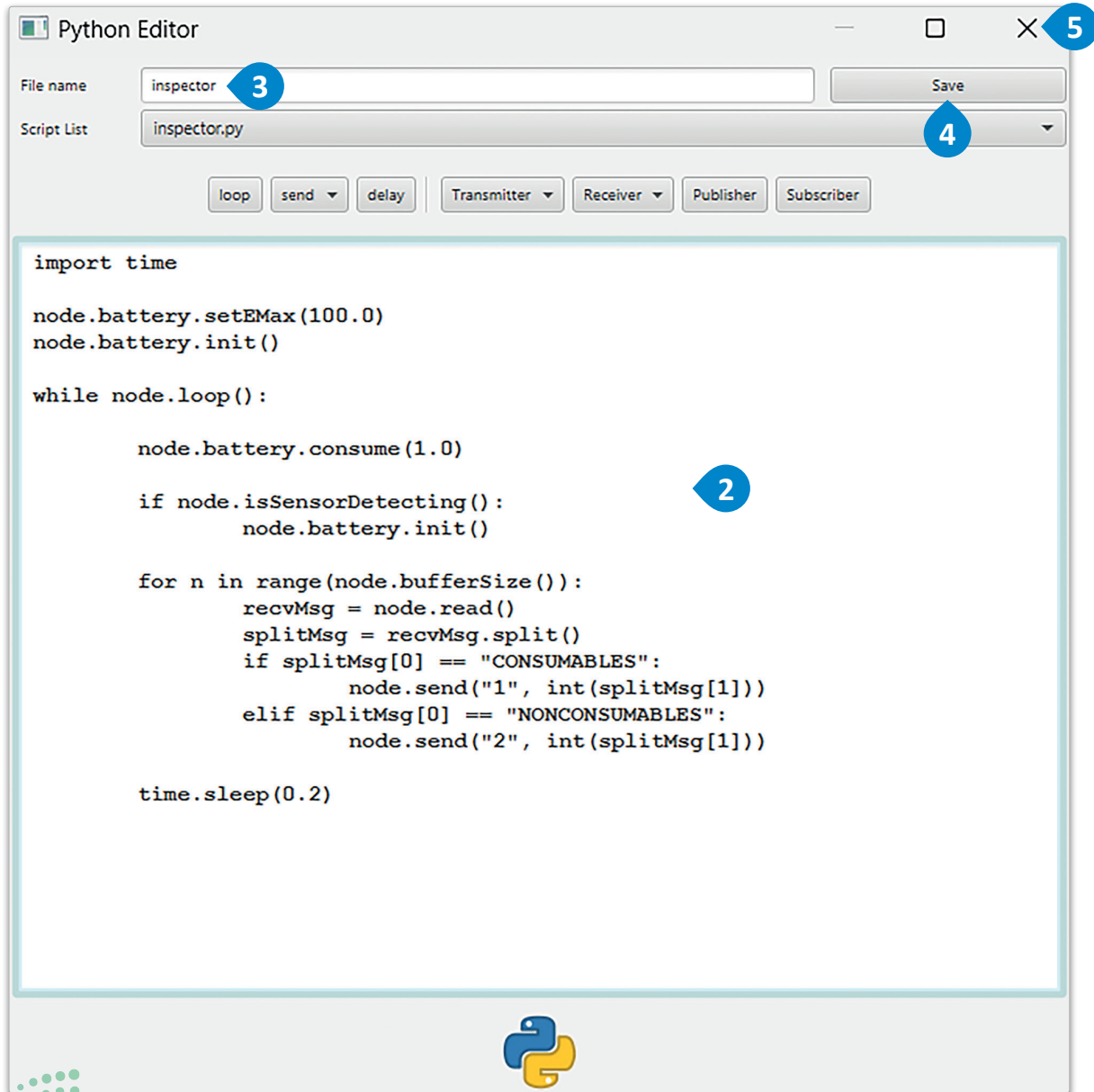
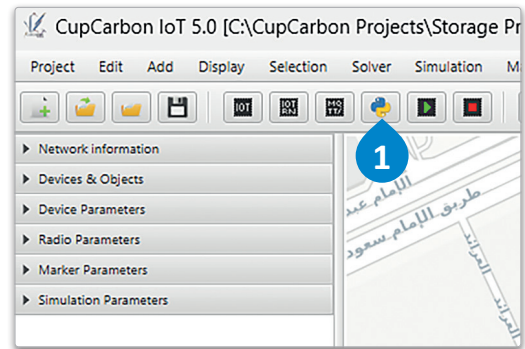
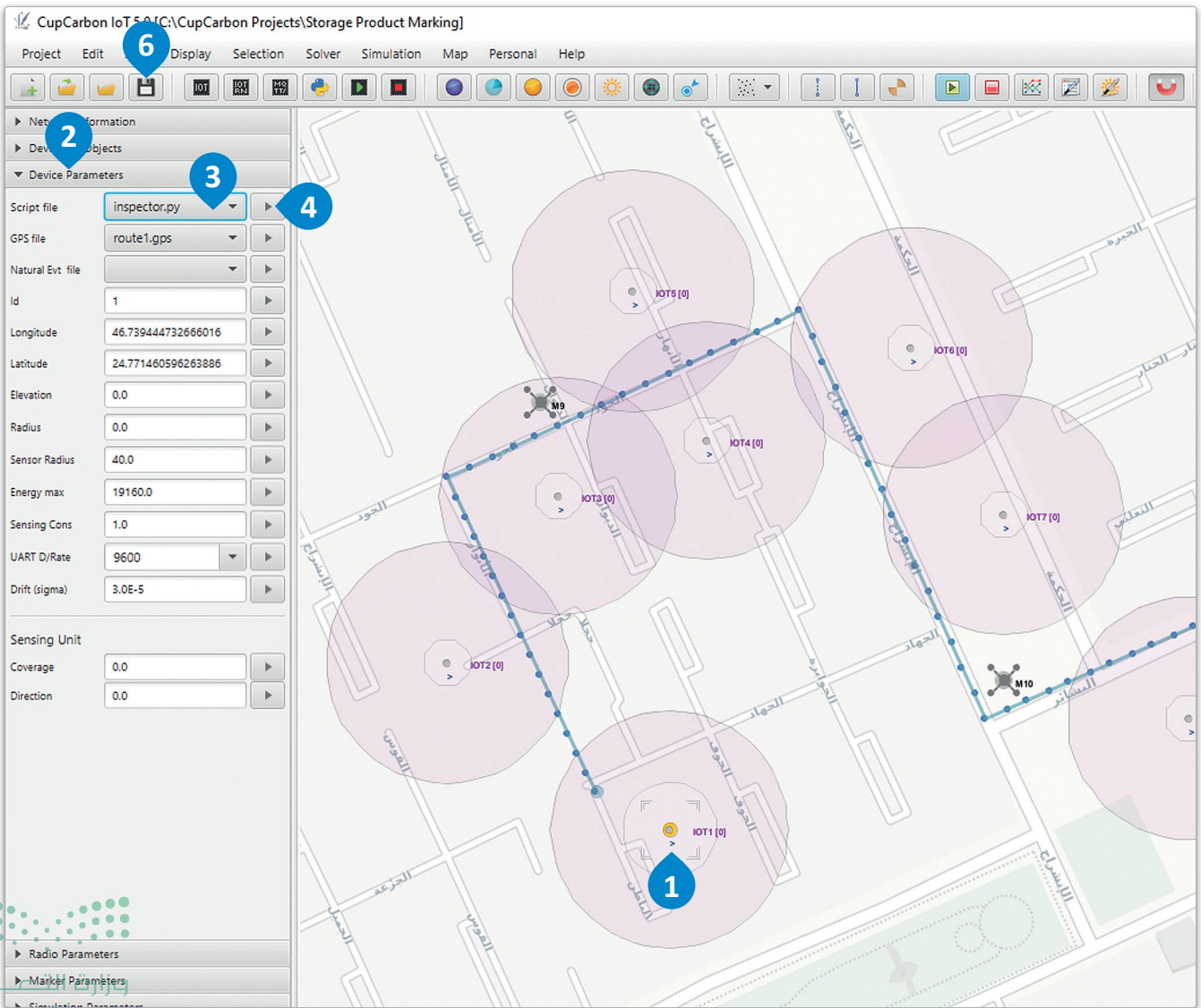


Figure 8.26: Create the script



To insert the script:

- > Click on the inspector vehicle node. 1
- > Click on the **Device Parameters** tab on the Parameter Menu. 2
- > Click on the **Script** file box. 3
- > From the drop down menu, choose the **inspector.py** script and click on the button on the right to insert the script into the node. 4
- > Click on **Display > Display/Hide Battery/Buffer levels**, from the Menu bar. 5
- > Click on the **Save Project** button on the Toolbar. 6



In a similar way, create the consumables.py and nonconsumables.py scripts and apply the first script to some of the container nodes and the second to the rest, so all container nodes have one of these two scripts.

When everything is set, you can click the Run IoT Simulation button from the toolbar to start the simulation.

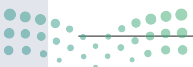


Figure 8.28: The simulation running

Exercises

- 1 Extend your project by adding more nodes and creating a route with more markers. Do not forget to insert scripts into the new nodes.
- 2 Identify whether your project is using the minimal number of charging stations. Try removing a station and relocating the others to test your hypothesis.
- 3 Modify the inspector vehicle's script to consume more energy and run out of battery quicker. Present your findings below.

- 4 Extend your project by creating a third type of container node, an empty container that will emit the string "EMPTY" and will not be marked by the inspector vehicle.
- 5 A slow network connection in the factory may have severe implications for the system's functionalities. Modify the script of the inspector vehicle's node to make the node sleep for longer. Are any messages delayed and/or lost? Present your observations below.



Project

Communication is key in a factory in order to coordinate different departments and sectors. When parallelism is applied adeptly, it can greatly increase efficiency and productivity.

1

You will be simulating a factory internal delivery system that consists of a delivery vehicle that moves on a predetermined route and delivers parts and materials to different sectors. Create a network of 1 controller with 3 proxies and 3 edge nodes on each proxy.

2

The delivery vehicle will pass from every edge node and deliver either parts or materials. Write a script for the edge nodes to ask for parts or for materials by sending a string with their request to the vehicle and write a script for the vehicle to provide what was requested by giving a confirmation.

3

Extend your project so that after the edge nodes receive their request the forward a message to their corresponding proxy that they can continue production. In turn the proxies will forward the message to the main controller, which will print an informing message stating the sector's request has been fulfilled.

4

Further extend your project to include a battery on the vehicle that depletes each time it moves. Add several charging stations across the route. Are you using the minimal amount of charging stations?



Wrap up

Now you have learned to:

- > Identify IoT technologies in manufacturing.
- > Use CupCarbon to simulate networks.
- > Create Python scripts to program network nodes.
- > Use CupCarbon to create IoT projects.

KEY TERMS

Connected Factory

**Data-Driven
Manufacturing**

Digitization

Edge Computing

**Industrial Automation
and Control Systems**

**Key Performance
Indicators**

Modbus protocol

Operational Technology

Smart Industry



