قــــررت وزارة الــتـعليــم تـدريــس
هــذا الـكـتـاب وطـبـعـه عـلـى نفقتها

وزارة الـتـعـلـيم
Ministry of Education

المملكة العربية السعودية

Internet of Things 1-2

Secondary stage - Pathways system

Second year

1444 - 2022 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

binarylogic

وزارة التعليم
Ministry of Education
2022 - 1444

King Fahd National Library Cataloging-in-Publication Data

Ministry of Education

Internet of Things 1.2 / Secondary Education - Pathway System Second Year / Ministry of Education - Riyadh, 2022

192p.; 210*25.5cm

ISBN: 978-603-511-351-9

1- Internet of Things   2- Curriculum

I-Title

370 dc 1444/4675

L.D. no.: 1444/4675

ISBN: 978-603-511-351-9

Enrichment and support materials on the “iEN Ethraia Platform”

IEN.EDU.SA

Submit your suggestions to enhance the textbook

FB.T4EDU.COM

وزارة التعليم
Ministry of Education
2022 - 1444

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, which is presented in two successive books, 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 data science 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

وزارة التعليم
Ministry of Education
2022 - 1444

Contents

بِسْمِ اللَّهِ الرَّحْمَٰنِ الرَّحِيمِ

وزارة التعليم
Ministry of Education
2022 - 1444

1. 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 IoT 1-1 book, 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.

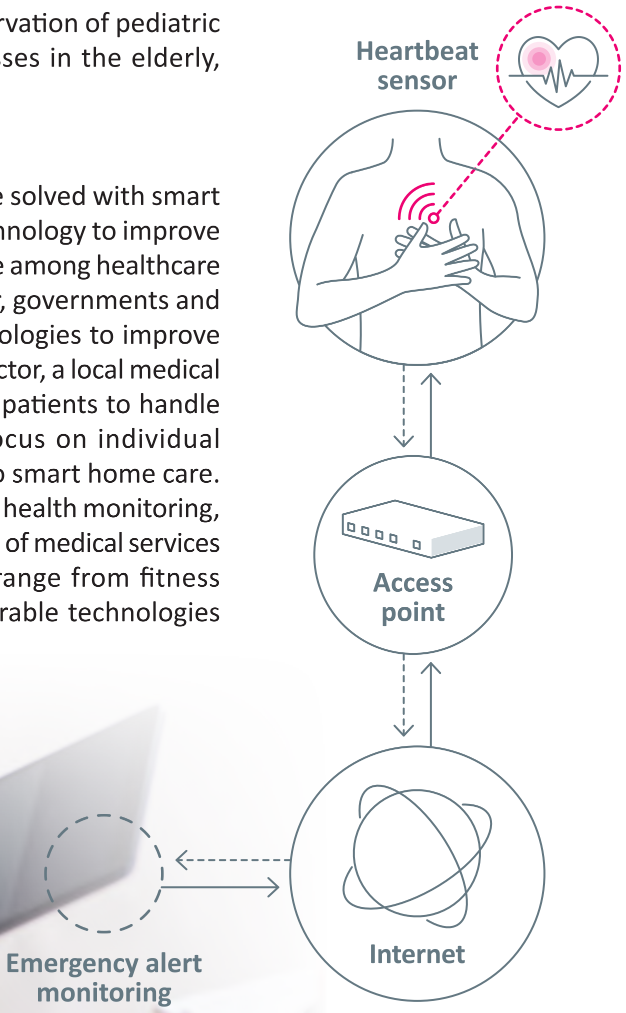


Figure 1.1: Monitoring health metrics

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

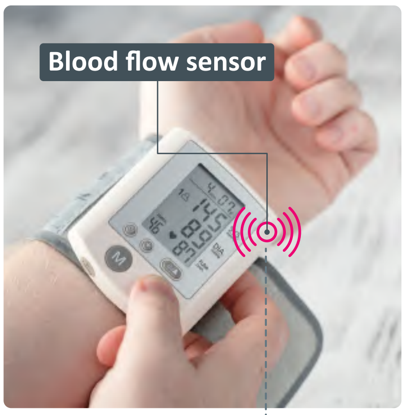
Wearables are smart objects placed on the human body. Wearable medical devices can gather, store, process, and analyze data to provide the required feedback and send alerts in emergency scenarios. The primary users are patients with temporary or permanent disabilities, the elderly, and babies. Biosensors 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), and movements, and geographics 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 1.2, 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.



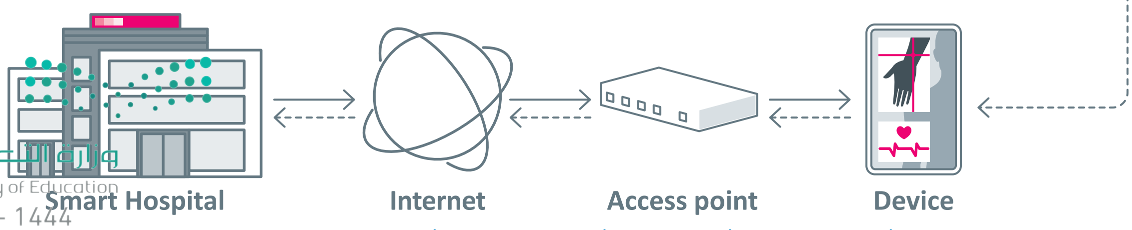


Figure 1.2: Body Sensor Networks connected to IoHT networks

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.

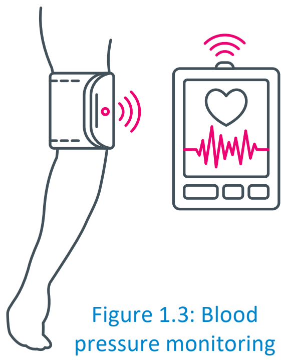


Figure 1.3: Blood pressure monitoring

Pain monitoring

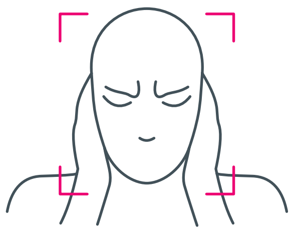


Figure 1.4: Pain monitoring

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

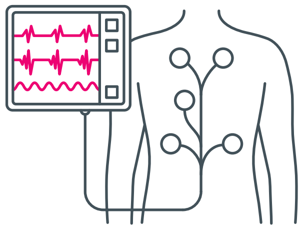


Figure 1.5: Electrocardiogram monitoring

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.

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.

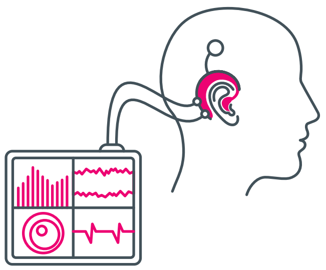


Figure 1.6: Sleep monitoring

Figure 1.7: 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.

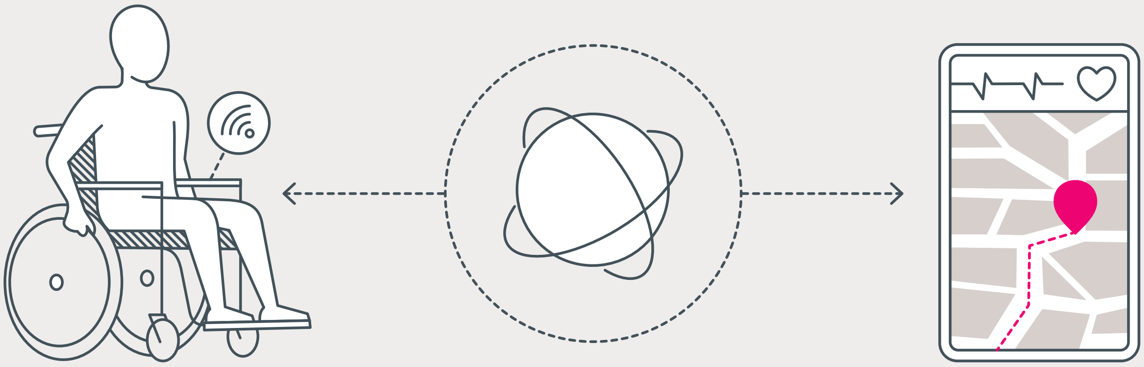


Figure 1.8: Disabled persons monitoring

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 1.9: 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.

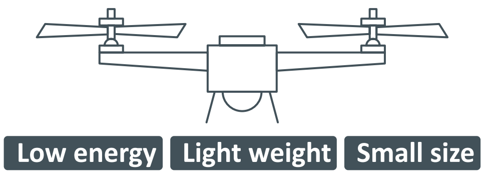


Figure 1.10: UAV essential requirements

Table 1.1: 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 1.11: 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 the 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.

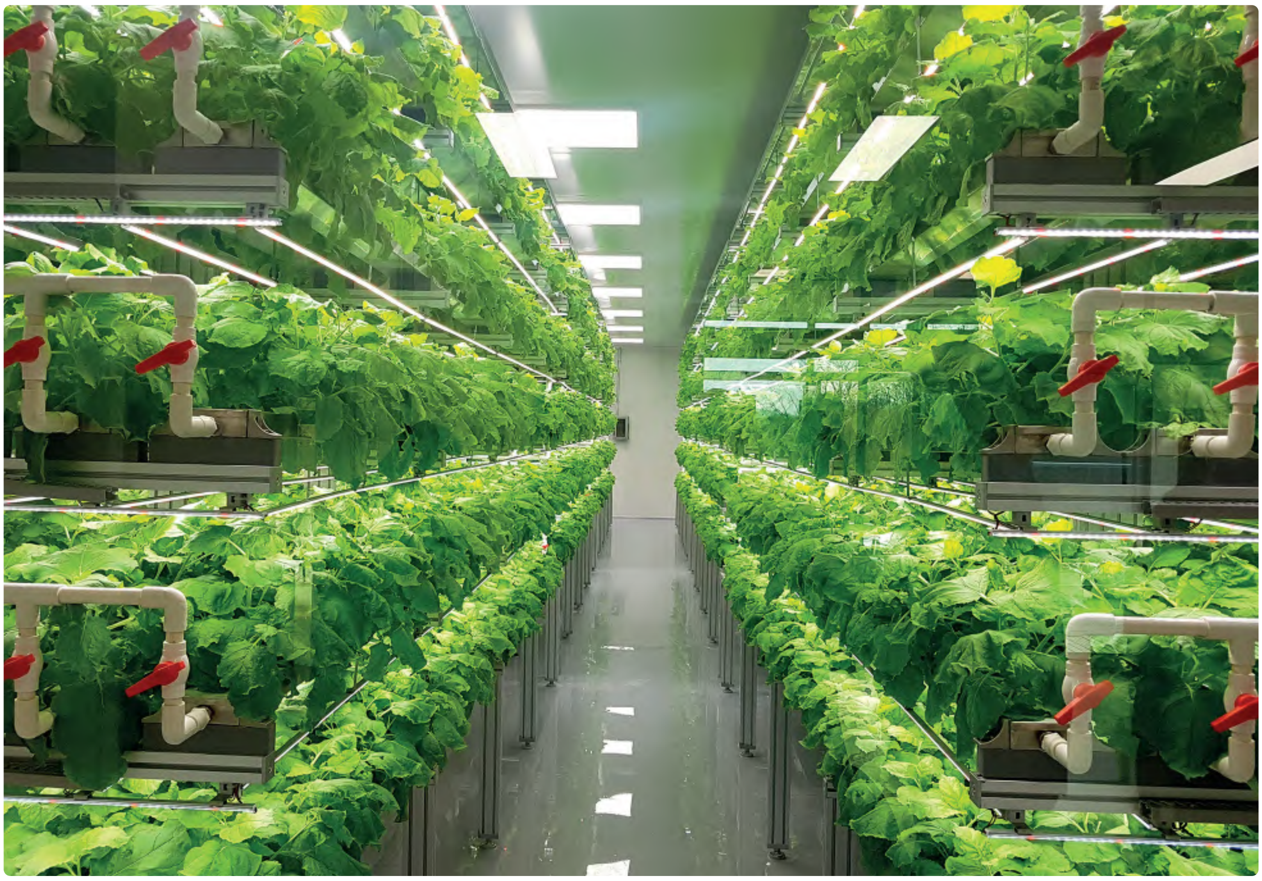


Figure 1.12: Vertical farming application

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.

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.

7 Classify the various types of UAV sensors.

8 Describe how IoT systems enable precision irrigation applications.

9 Analyze how vertical farming is dependent on effective IoT solutions.

Link to digital lesson

www.ien.edu.sa

Lesson 2

IoT Networking Technologies

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.

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.

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.

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.

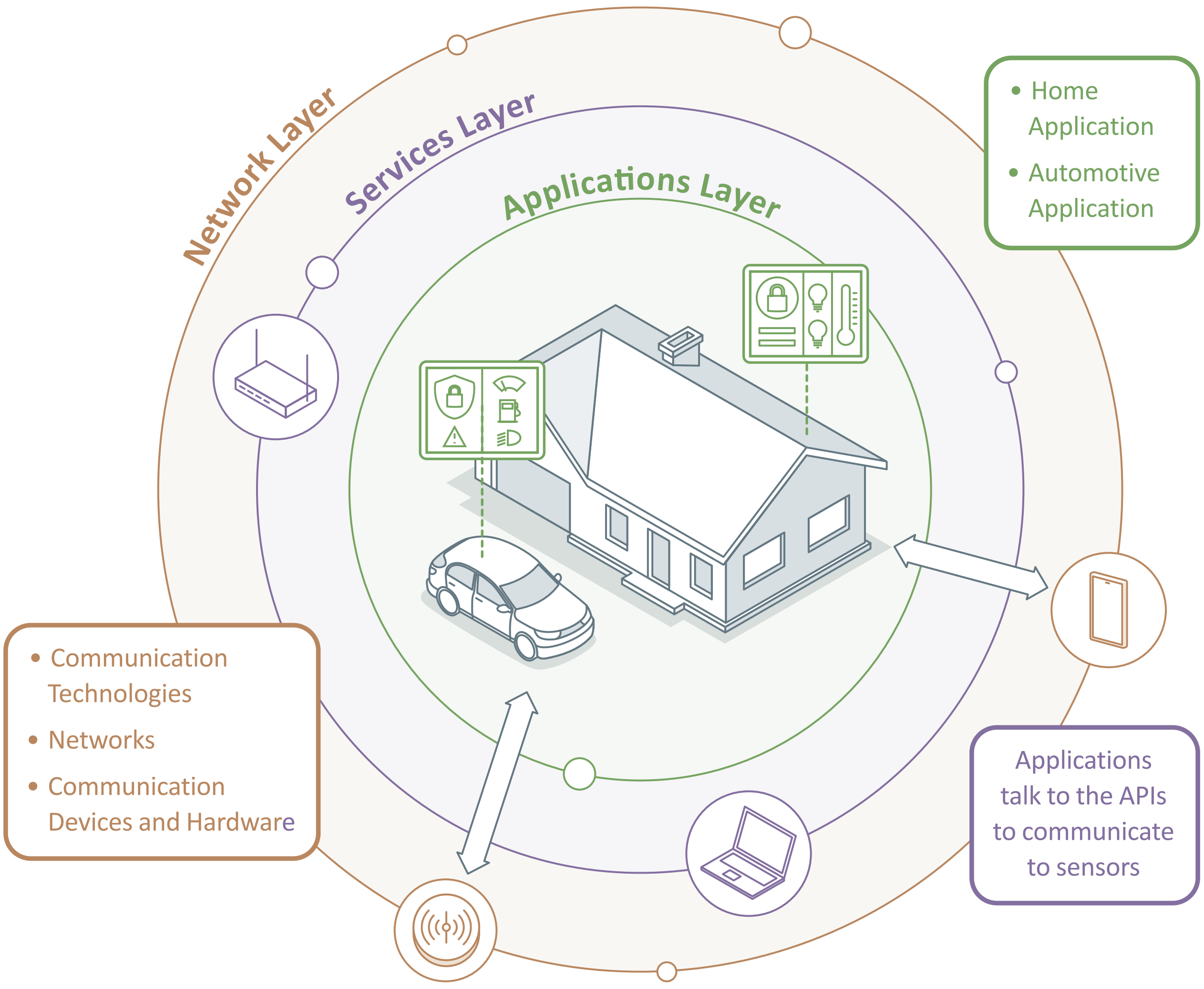

Network Layer
Services Layer
Applications Layer
• Home Application
• Automotive Application
• Communication Technologies
• Networks
• Communication Devices and Hardware
Applications talk to the APIs to communicate to sensors

Figure 1.13: 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.

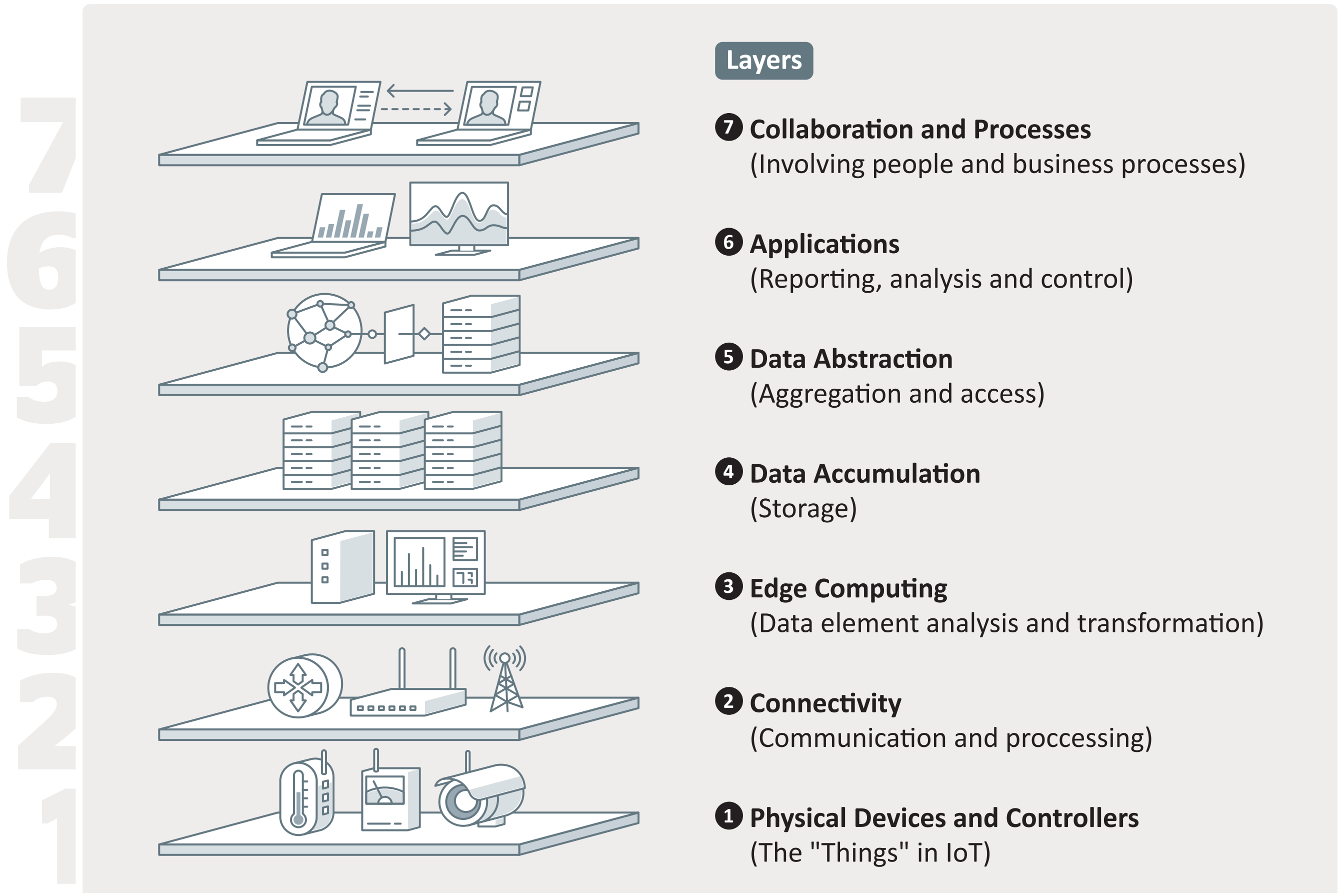
IoT World Forum Architecture

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.



7
6
5
4
3
2
1
Layers
7 Collaboration and Processes
(Involving people and business processes)
6 Applications
(Reporting, analysis and control)
5 Data Abstraction
(Aggregation and access)
4 Data Accumulation
(Storage)
3 Edge Computing
(Data element analysis and transformation)
2 Connectivity
(Communication and proccessing)
1 Physical Devices and Controllers
(The "Things" in IoT)

Figure 1.14: 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.

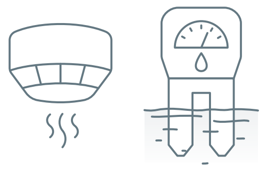


Figure 1.15: Devices & Controllers layer

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.

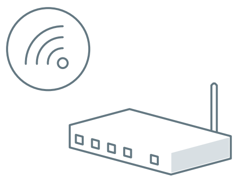


Figure 1.16: Connectivity layer

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.

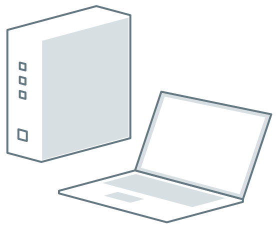


Figure 1.17: Edge Computing layer

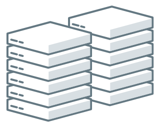


Figure 1.18: Data Accumulation layer

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.

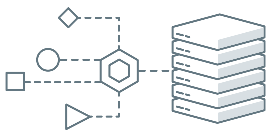


Figure 1.19: Data Abstraction layer

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.

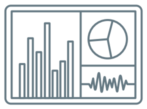


Figure 1.20: Applications layer

Layer 6: Applications Layer

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

Figure 1.21: Collaboration & Processes layer

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's 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 4 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 1.2: 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.

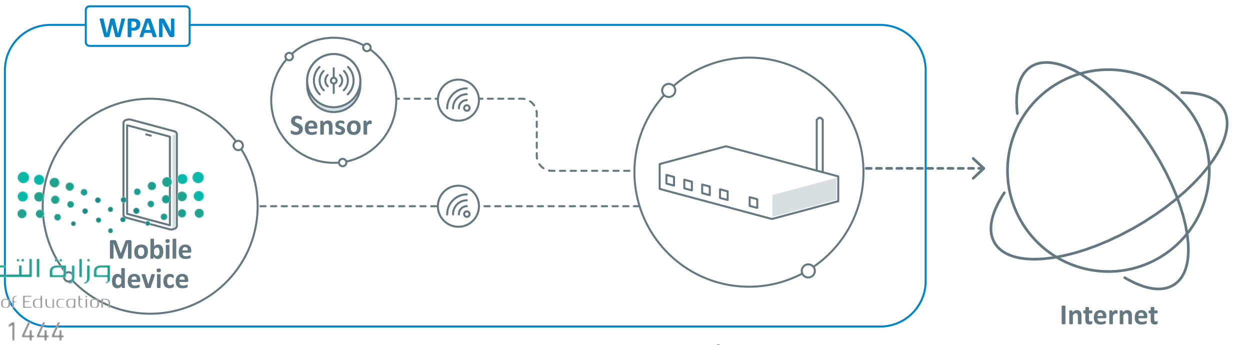


Figure 1.22: WPAN network

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 1.3: 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.

Sensor

2. **Intermittent data:** A light switch is a nice example of intermittent data ideal for Zigbee.

Light switch

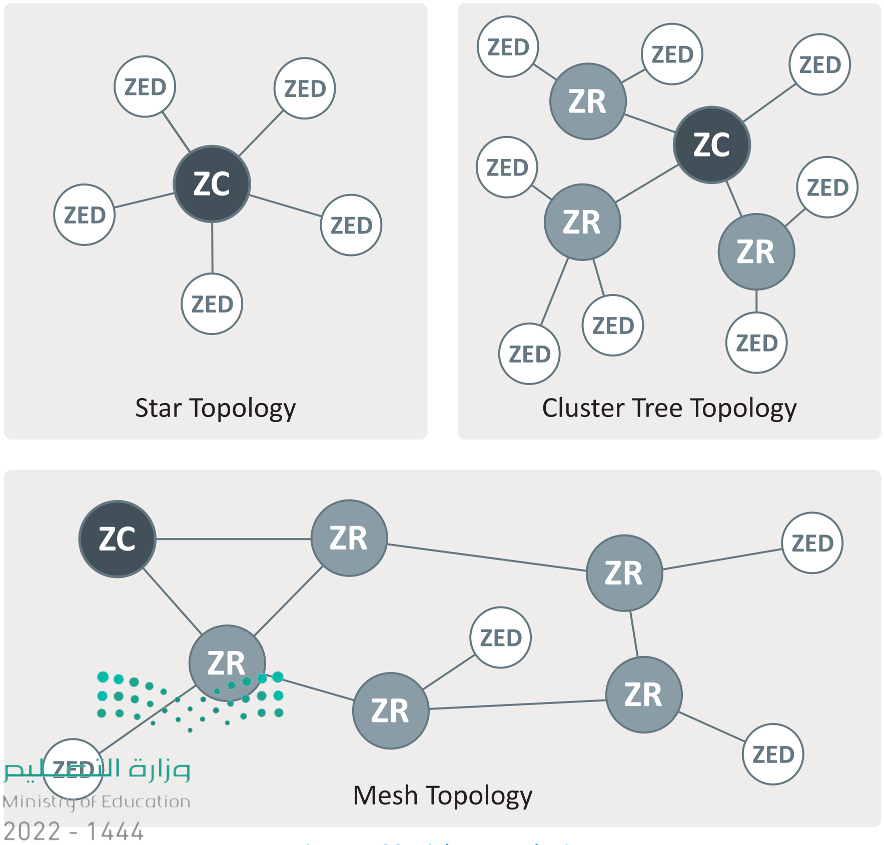
3. **Repeated low-latency data:** Zigbee assigns transmission time slots and can have very low latency, making it suitable for computer mice and keyboards.

Mouse

There are three fundamental Zigbee topologies:

Table 1.4: 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 has 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 1.23: 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.

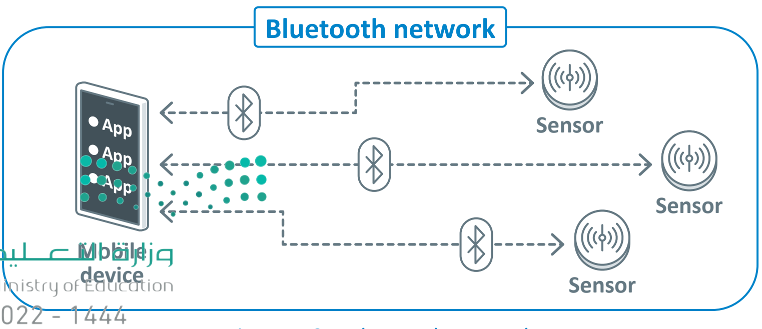


Figure 1.24: Bluetooth network

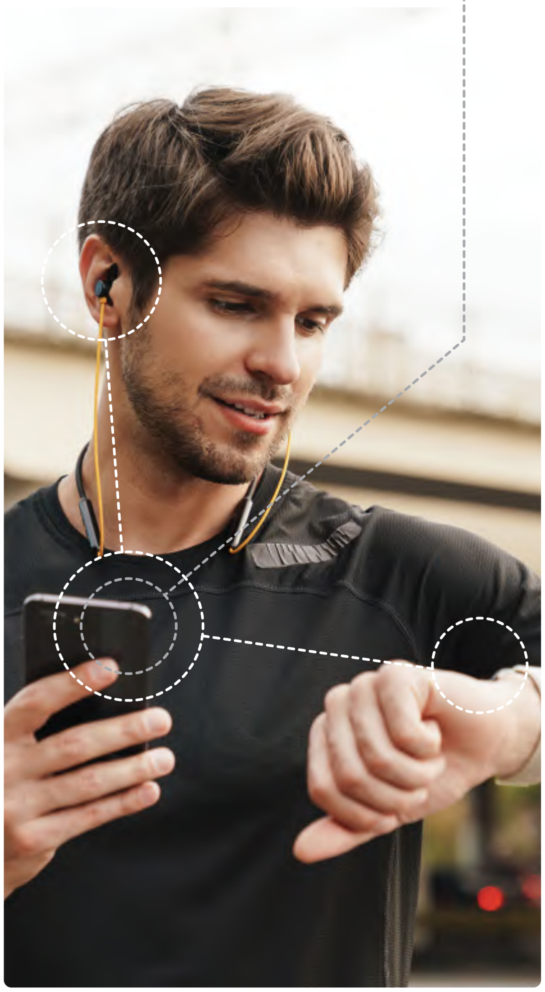


Figure 1.25: Bluetooth connectivity

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 1.26: 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 cities operators, broadcasters, and mobile and non-mobile services providers are addressing the need for regional or national IoT infrastructures, which are vital for enabling use cases for consumer markets.

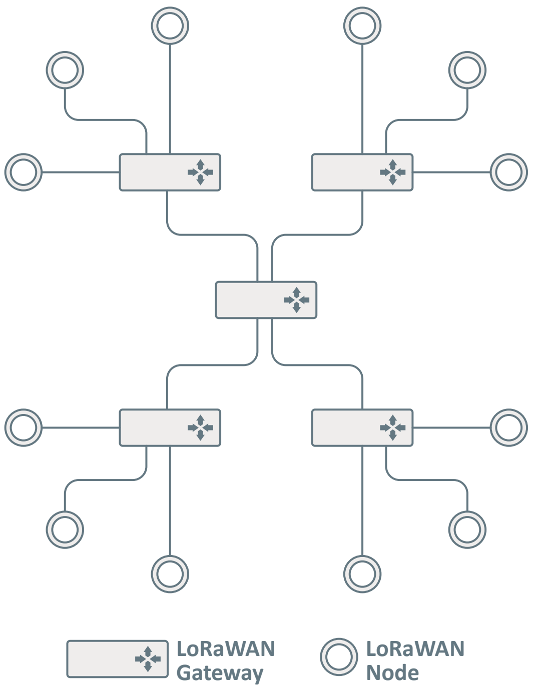


Figure 1.27: LoRaWAN "star of stars" topology

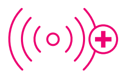
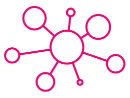
Cellular Networks (5G)

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 1.5: 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.

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

Lesson 3

Security and Privacy of IoT Systems

Link to digital lesson

www.ien.edu.sa

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 1.6: 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.

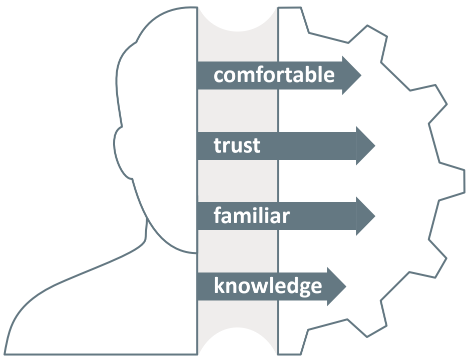


Figure 1.28: Accountability in the IoT

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 1.29: 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.

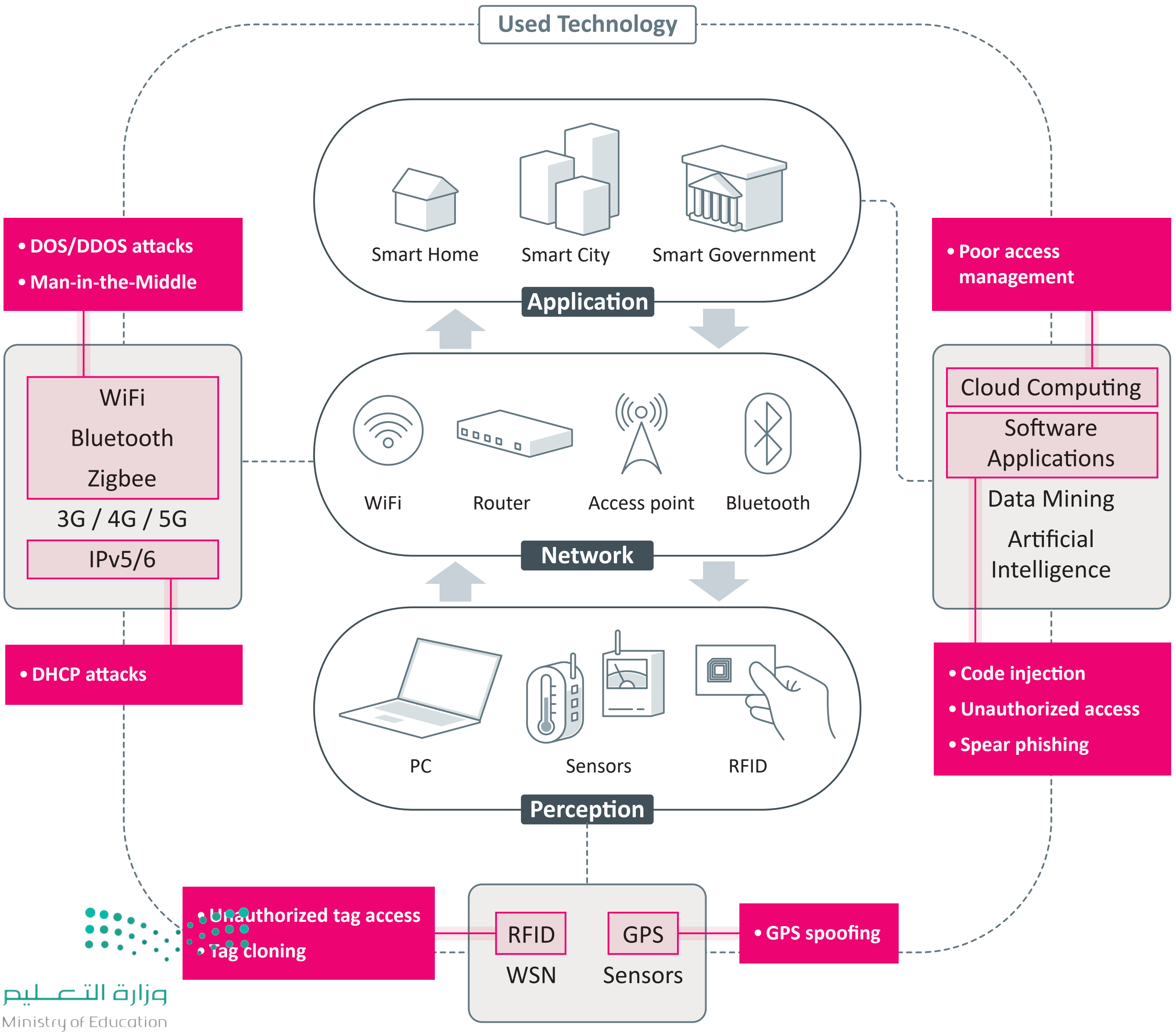


Figure 1.30: Security vulnerabilities in IoT systems

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 1.7: 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.

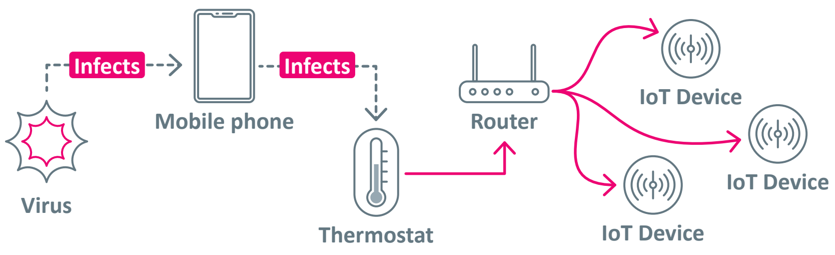


Figure 1.31: Infection of a network

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 1.32: 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 1.8: 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 Classify the main principles of IoT security.

4 Describe the main challenge of IoT security and cybercrime on the Internet. How can this issue be addressed?

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

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

7 Analyze how Big Data technologies create new privacy challenges.

8 Classify the current privacy concerns present in IoT systems.

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

2. 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

Link to digital lesson

www.ien.edu.sa

Lesson 1

Smart Security Applications with C++

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.

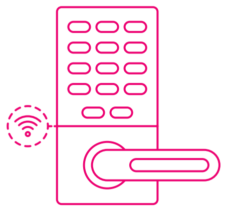
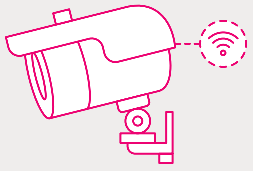
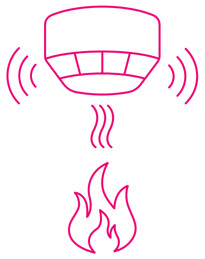
Risks

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.

Figure 2.1: Using a smartphone to open automatic gate machine in an office building

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

Table 2.1: 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++

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 2.2: 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];
```

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 2.3: Arithmetic operators

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

Table 2.4: 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 2.5: 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**

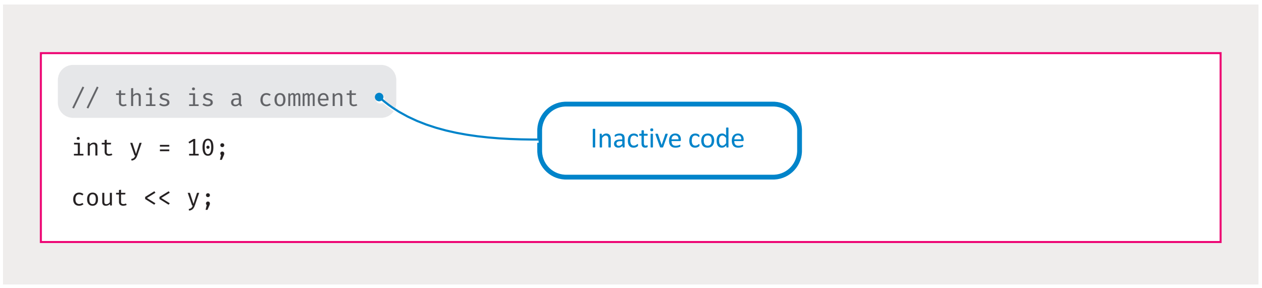
Table 2.6: Logical operators

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++

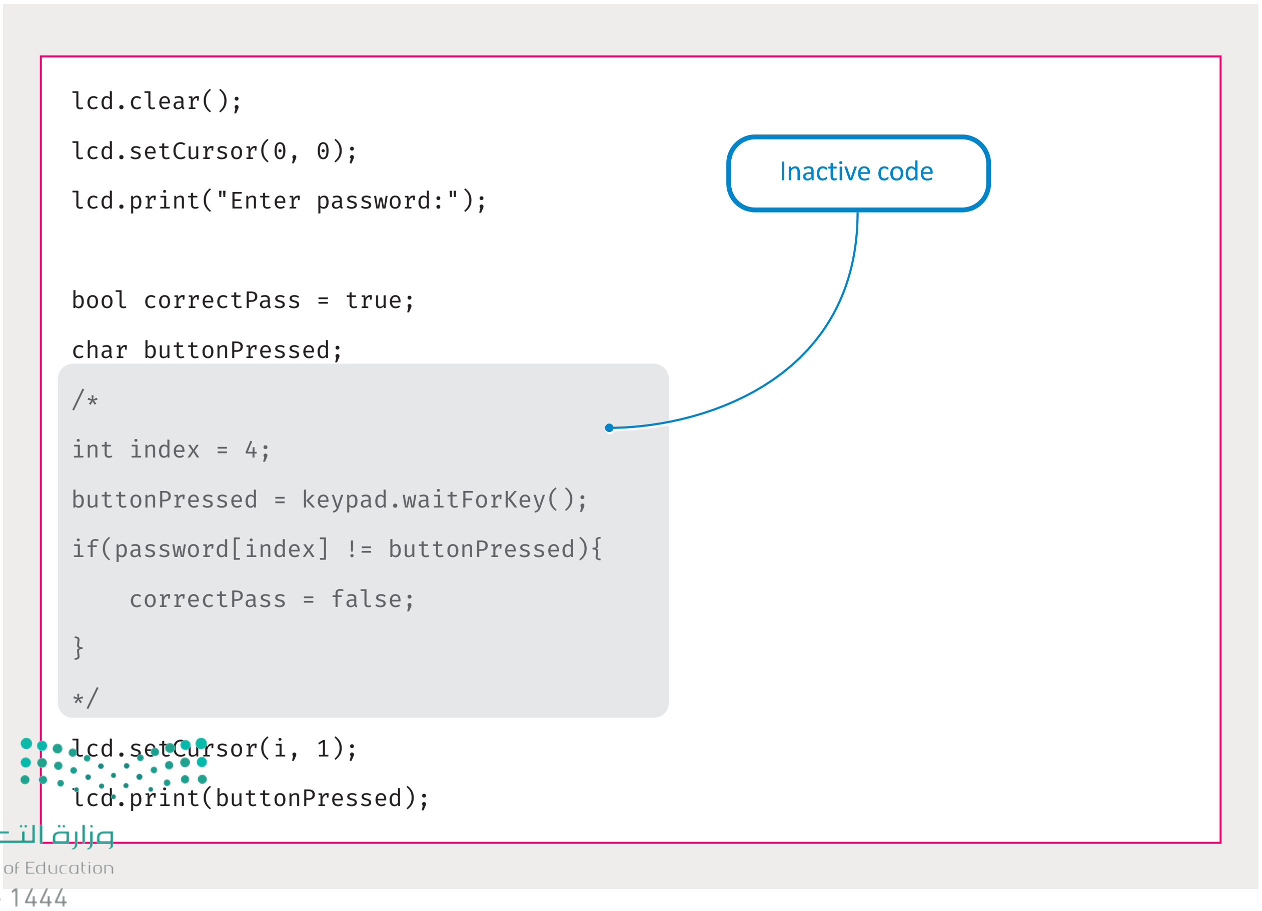
Another basic feature every programming language supports is comments. These comments are ignored by the compiler and are used to improve the readability of the code, making it easier for programmers or code reviewers to understand the functionality of the code. There are two ways to add a comment in 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;
```

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);
```

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
}
```

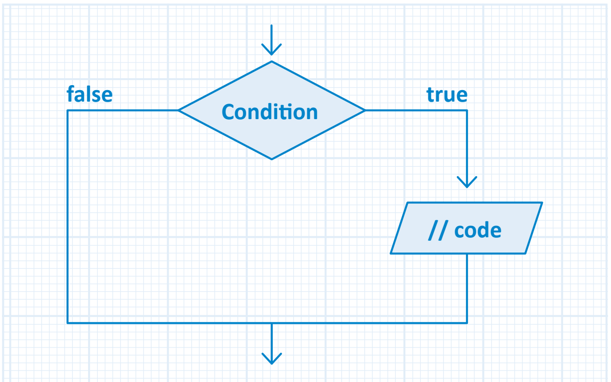


Figure 2.2: **if** statement flowchart

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
}
```

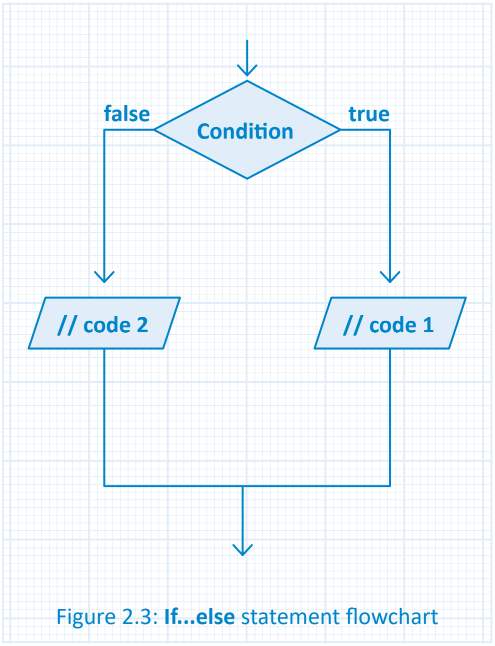


Figure 2.3: **If...else** statement flowchart

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) {          Executed
    // code
}
else {                     Skipped
    // code
}


// code after if...else
```

Condition is **false**

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


// code after if...else
```

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 if...else statement is:

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

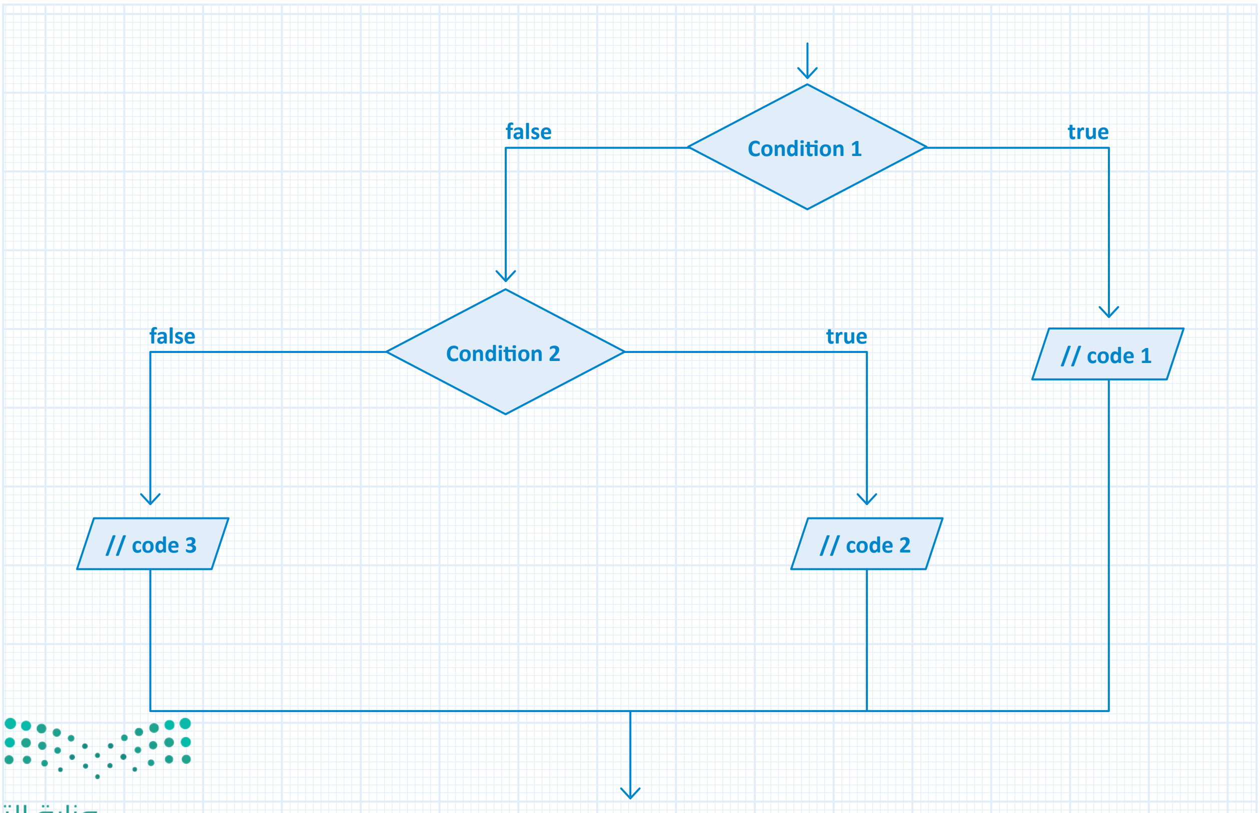


Figure 2.4: **If...else if ... else** statement flowchart

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 = 0;
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;
}
```

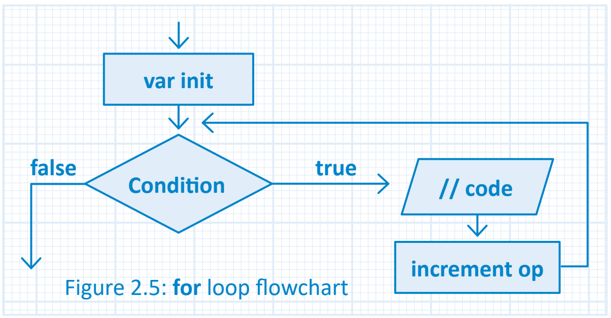


Figure 2.5: **for** loop flowchart

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.

"while" loop

The syntax of a while loop is:

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

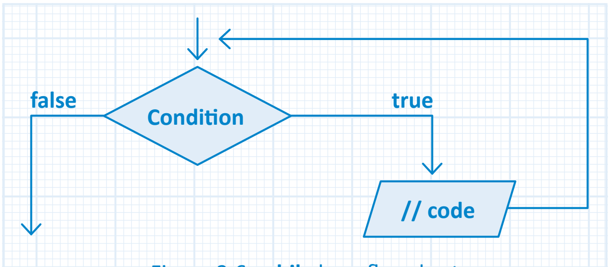


Figure 2.6: **while** loop flowchart

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.

"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);
```

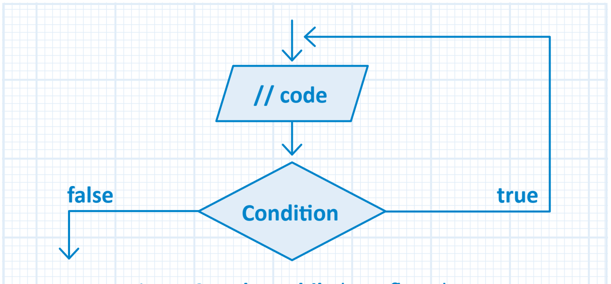


Figure 2.7: **do...while** loop flowchart

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.

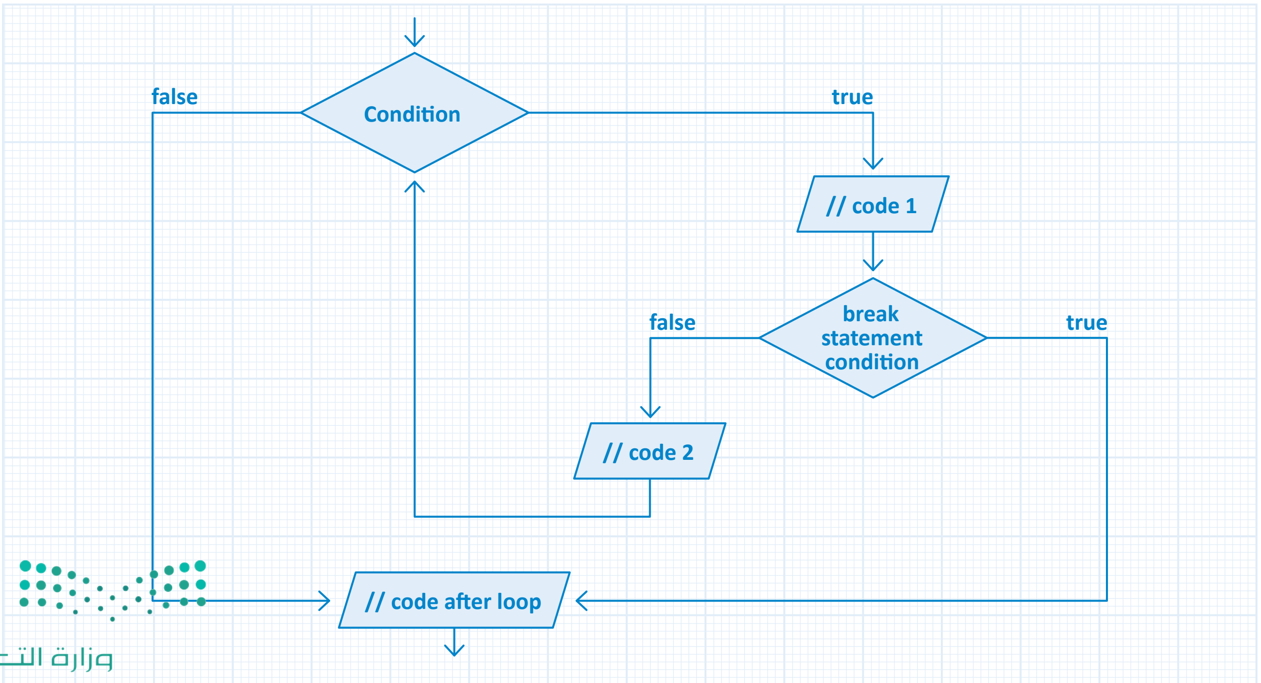


Figure 2.8: **break** statement flowchart

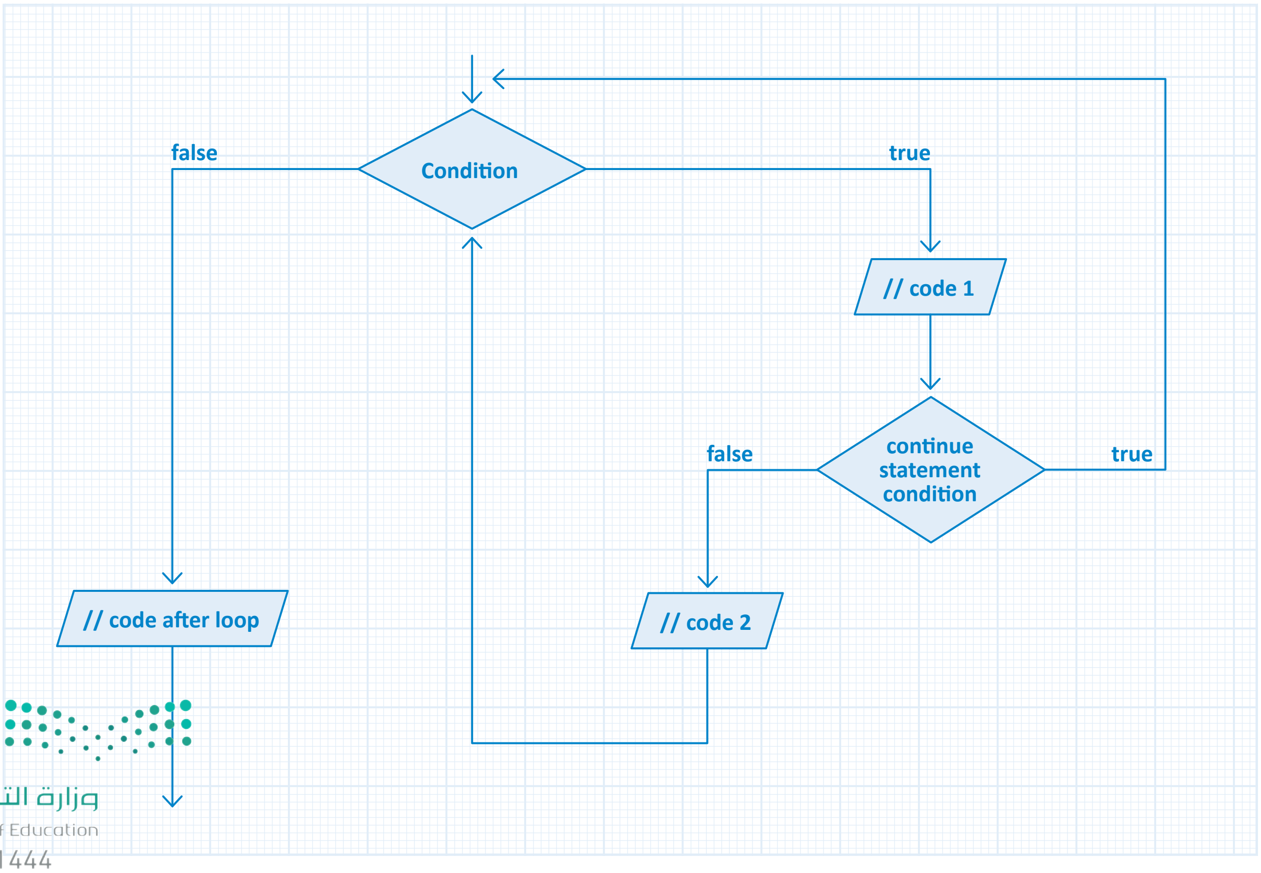
"continue" Statement

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.

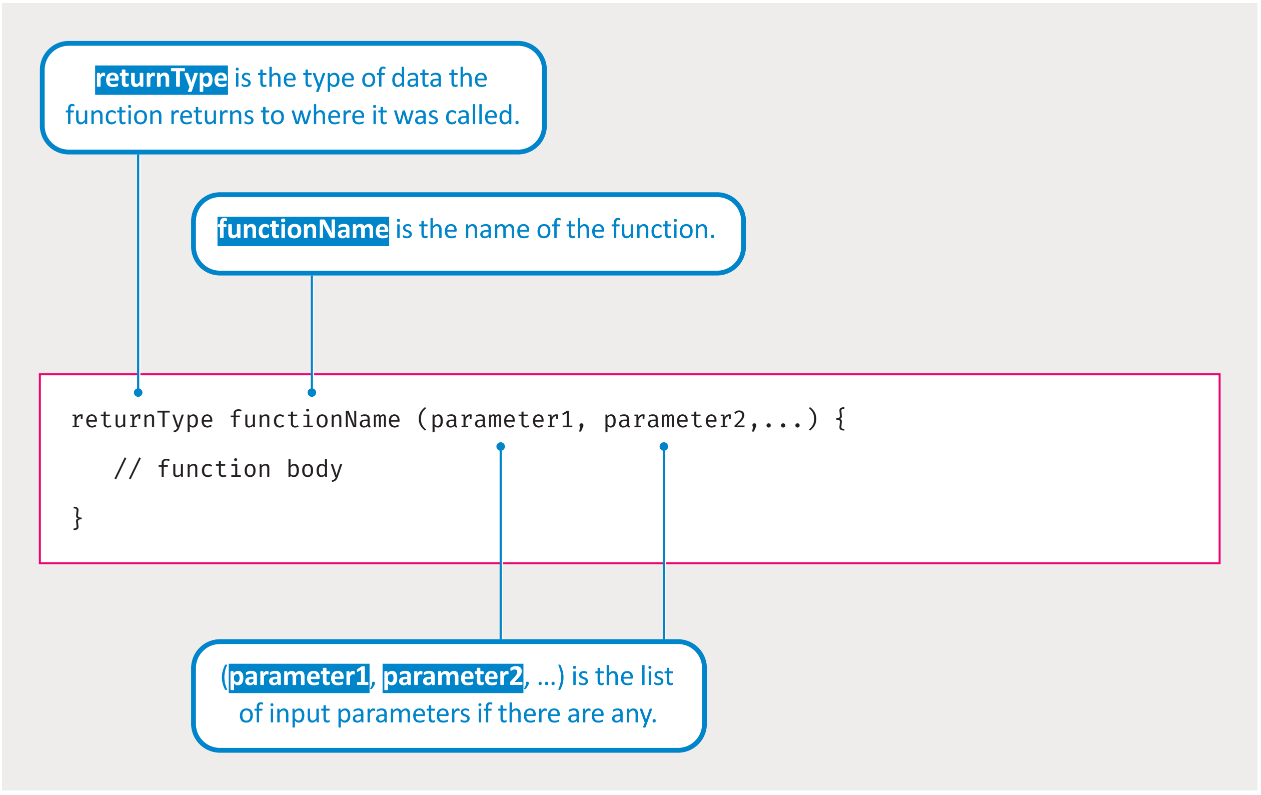


Functions in C++

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:


returnType is the type of data the function returns to where it was called.
functionName is the name of the function.
(**parameter1**, **parameter2**, ...) is the list of input parameters if there are any.

```
returnType functionName (parameter1, parameter2,...) {
   // function body
}
```

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

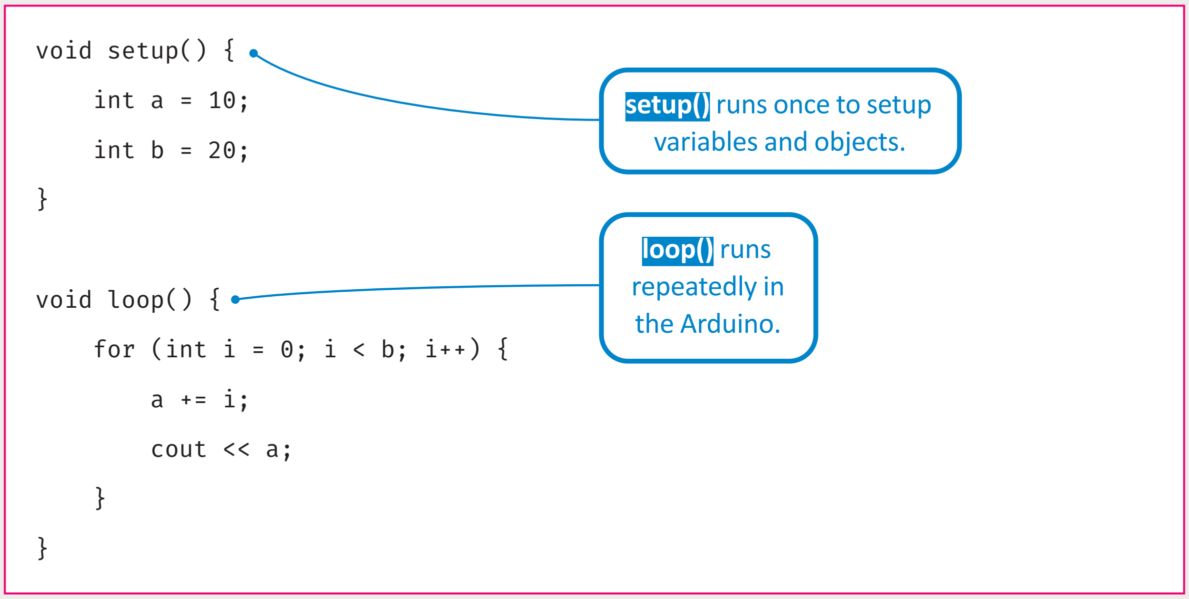
When writing code for Arduino in the Tinkercad platform, there are two functions that are called to execute the code of the circuit. These functions are called automatically when the program starts its execution unlike the rest of the functions that need to be called explicitly by your code.

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.



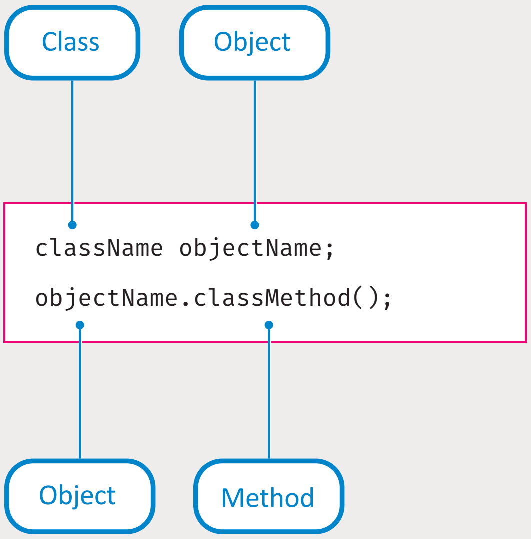
```
void setup() {
    int a = 10;
    int b = 20;
}

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 the 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.



```
className objectName;
objectName.classMethod();
```

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 Identify the potential risks of advanced smart security IoT applications.

4 Classify the most common IoT-enabled smart home devices

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

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

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

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.

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++

Link to digital lesson

www.ien.edu.sa

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++.

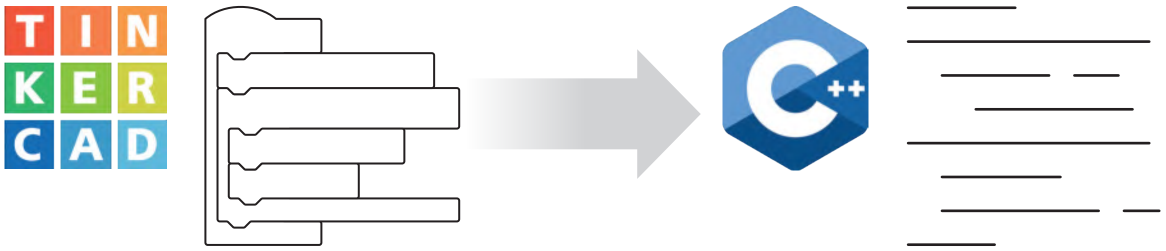


Figure 2.10: Tinkercad blocks to C++ code

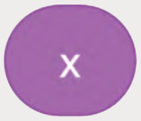
Variable Assignments and Operations

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.

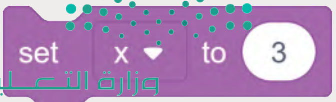
Command groups used:

● Output	● Control
● Input	● Math
● Notation	● Variables

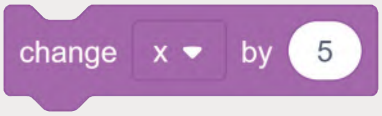
Declaring a variable named **x**.

Tinkercad block	C++
	`int x = 0;`

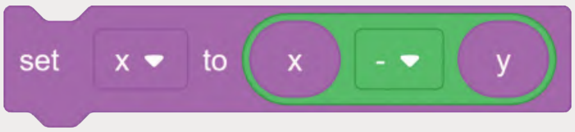
Assigning a value to a variable.

Tinkercad block	C++
	`x = 3;`

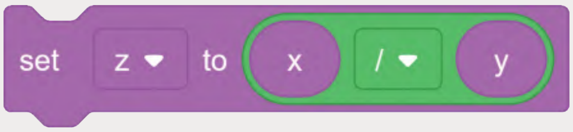
Changing a variable by a specific value.

Tinkercad block	C++
	`x += 5;`

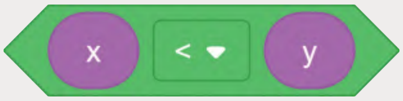
Performing a mathematical operation between variables **x** and **y**.

Tinkercad block	C++
	`x = x - y;`

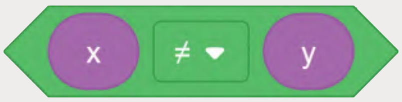
Assigning a third variable z to a mathematical operation between variables **x** and **y**.

Tinkercad block	C++
	`z = x / y;`

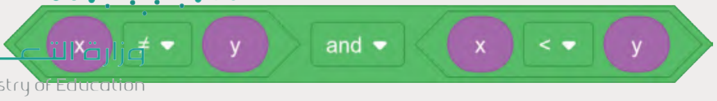
Performing a mathematical comparison between variables **x** and **y**.

Tinkercad block	C++
	`x < y`

Performing a logical comparison between variables **x** and **y**.

Tinkercad block	C++
	`x != y`

Performing a logical operation between two statements.

Tinkercad block	C++
	`x != y && x < y`

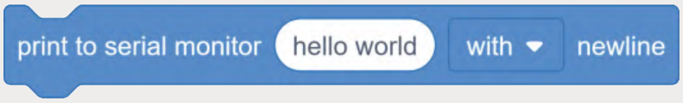
Conditional Statements, Loops and Output Messages

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.

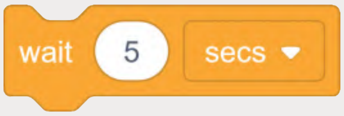
Command groups used:

● Output	● Control
● Input	● Math
● Notation	● Variables

Print a message to the **Serial Monitor**.

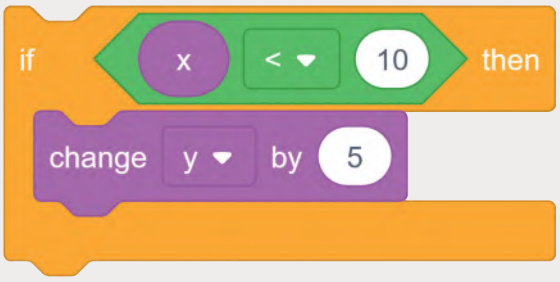
Tinkercad block	C++
	`Serial.println("hello world");`

Wait for 5 seconds.

Tinkercad block	C++
	`delay(5000);`

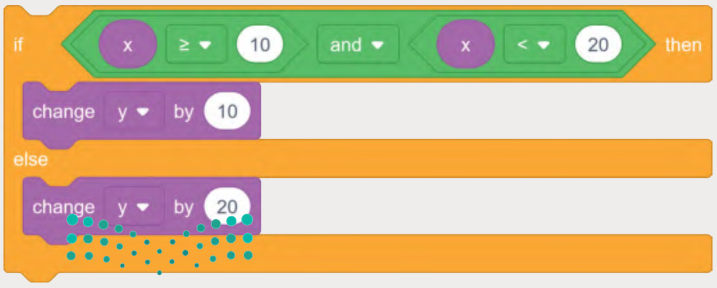
Execute the blocks in the **if** codeblock if the logical condition is true.

Tinkercad blocks	C++

	`if (x < 10) {`
	`    y += 5;`
	`}`

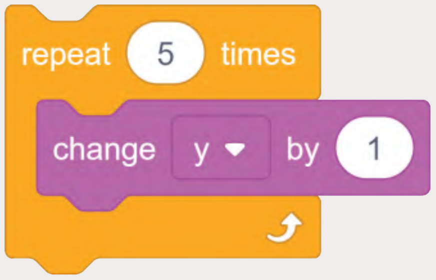
Execute the blocks in the **if** codeblock if the logical condition is true.

Tinkercad blocks	C++

 if
x
≥
10
and
x
<
20
then
change
y
by
10
else
change
y
by
20
	`if (x >= 10 && x < 20) {`
	`    y += 10;`
	`}`
	`else {`
	`    y += 20;`
	`}`

Execute the blocks in the **for** codeblock if the logical condition is true.

Tinkercad blocks

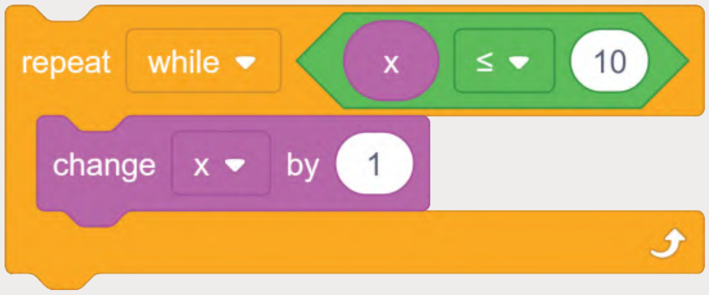


C++

```
for (counter = 0; counter < 5; ++counter) {
    y += 1;
  }
```

Repeat a while loop under the following condition.

Tinkercad blocks



C++

```
while (x <= 10) {
   x += 1;
}
```

Arduino Digital and Analog Pin I/O

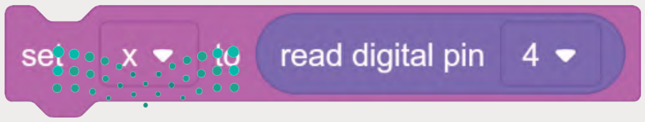
Interacting with the Arduino board's digital and analog pins in Tinkercad blocks is done through the **Input**, **Output**, and **Math** command groups. Each time that a pin is used, whether it is analog or digital, Tinkercad blocks recognizes if it is used for digital or analog I/O. To use a pin, you 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**.

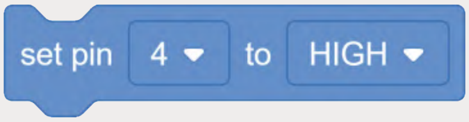
Tinkercad block



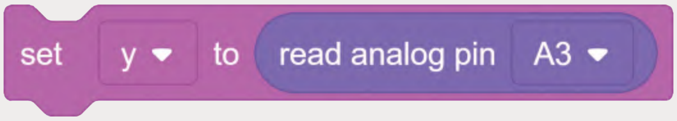
C++

```
pinMode(4, INPUT);
x = digitalRead(4);
```

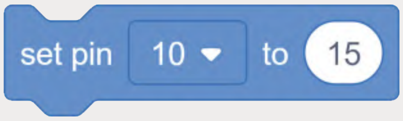
Setting the value of the digital pin **4** to **HIGH**.

Tinkercad block	C++
 set pin 4 to HIGH	`pinMode(4, OUTPUT);` `digitalWrite(4, HIGH);`

Getting the value of the analog pin **A3** and storing it in the variable **y**.

Tinkercad block	C++
 set y to read analog pin A3	`pinMode(A3, INPUT);` `y = analogRead(A3);`

Setting the value of the pin **10** to the analog value **15** using **PWM**.

Tinkercad block	C++
 set pin 10 to 15	`pinMode(10, OUTPUT);` `analogWrite(10, 15);`

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

Components that you will use in this project

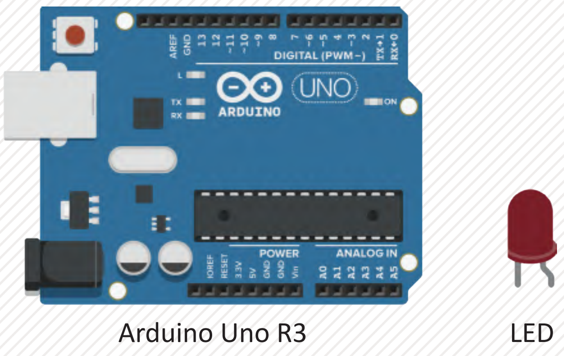


Figure 2.11: Project components

Connecting 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

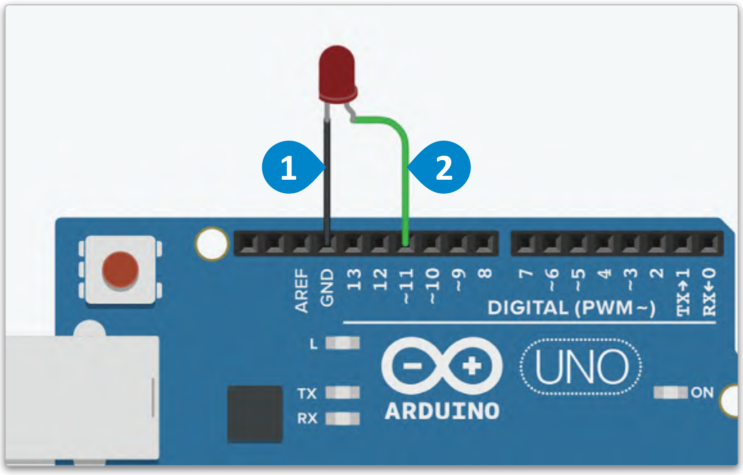
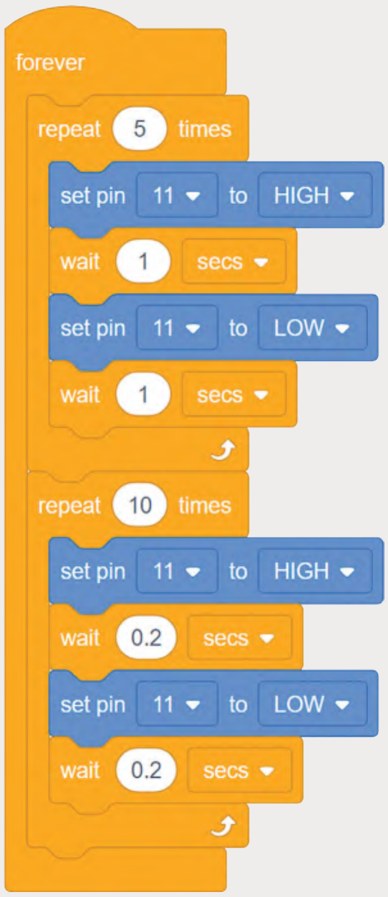


Figure 2.12: Connecting the LED

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



```
forever
  repeat 5 times
    set pin 11 to HIGH
    wait 1 secs
    set pin 11 to LOW
    wait 1 secs
  repeat 10 times
    set pin 11 to HIGH
    wait 0.2 secs
    set pin 11 to LOW
    wait 0.2 secs
```

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

Creating 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

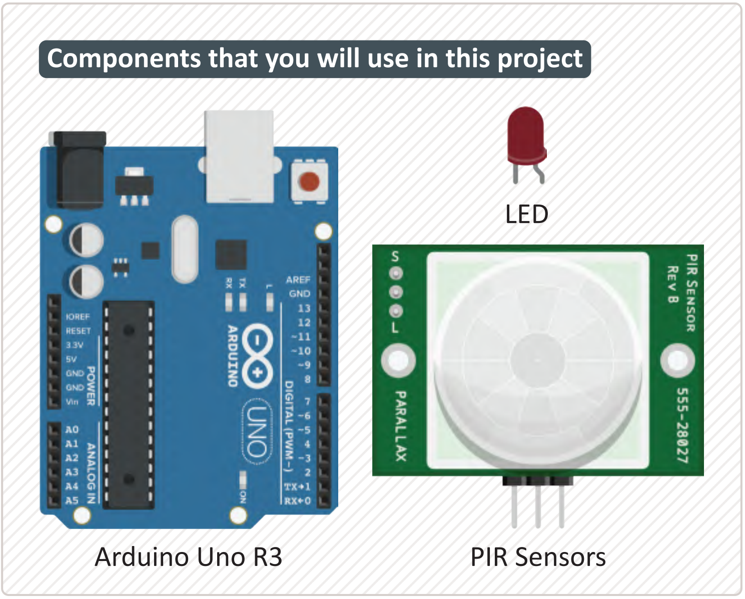


Figure 2.13: Project components

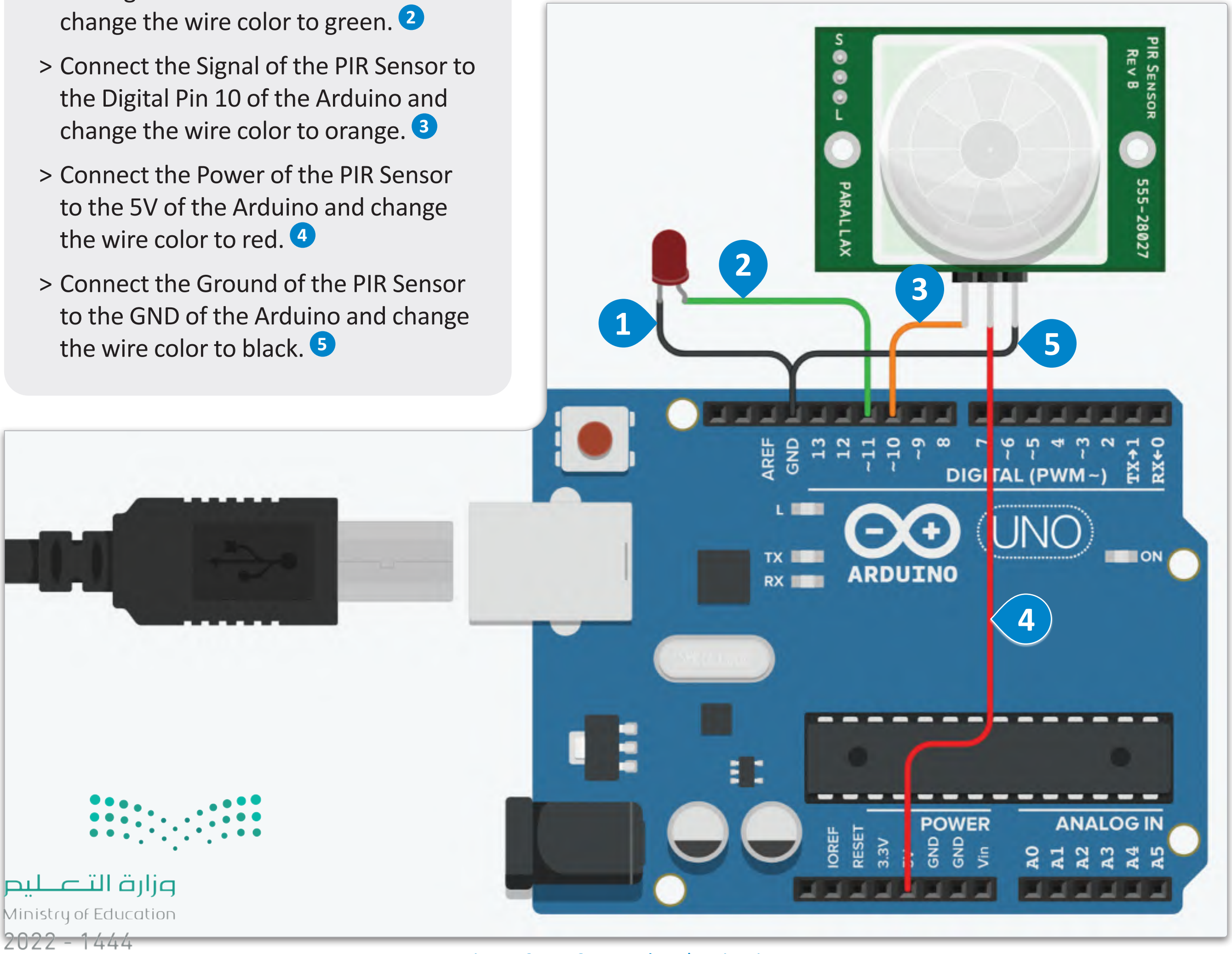
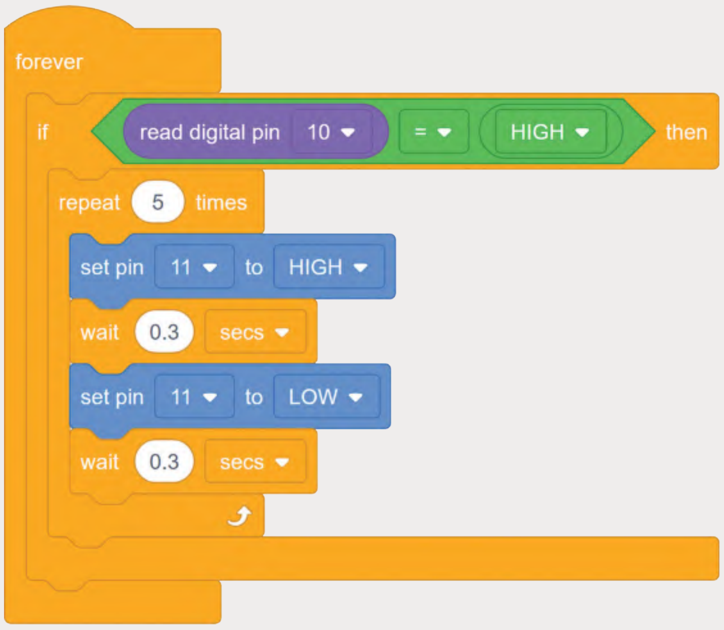


Figure 2.14: Connecting the circuit

Tinkercad blocks



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.

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)
    }
  }
}
```

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)

Creating 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 Servo motor to the Digital pin 11 of the Arduino and change the wire color to red. 5

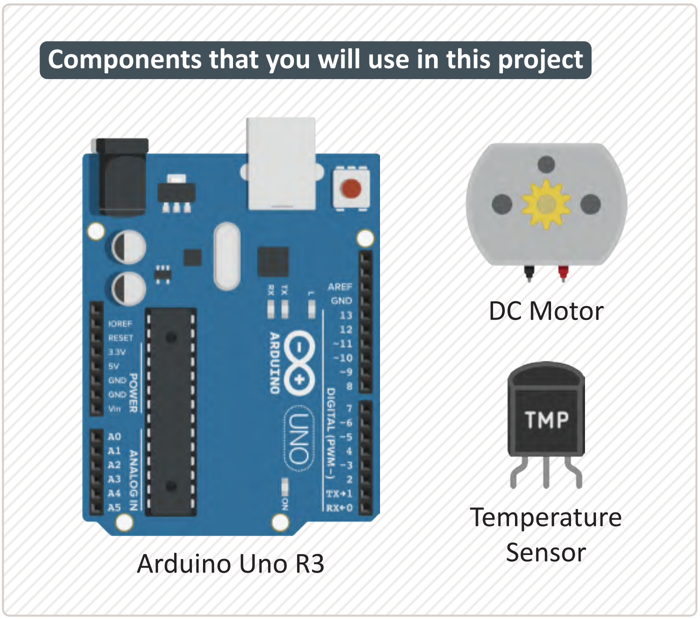


Figure 2.15: Project components

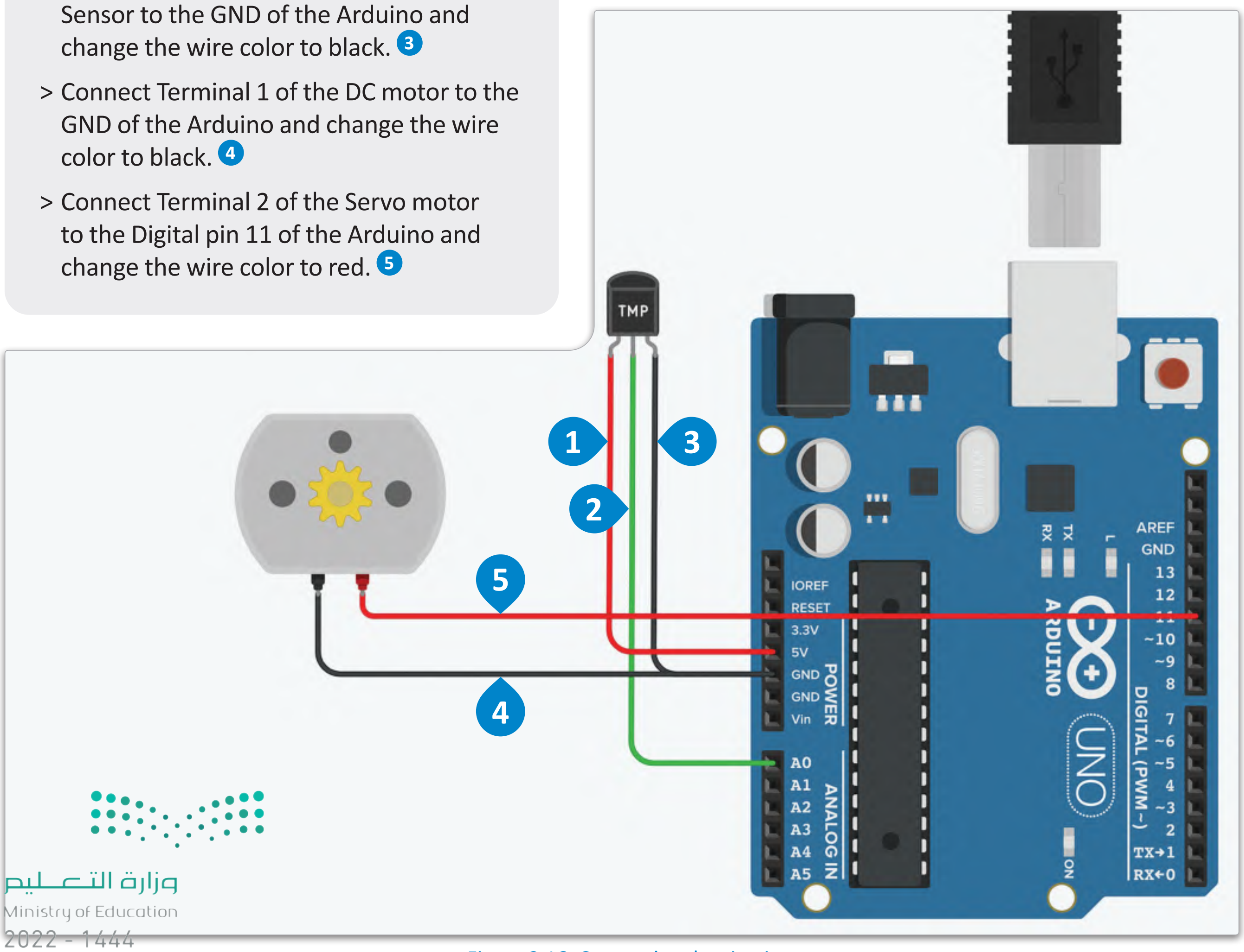
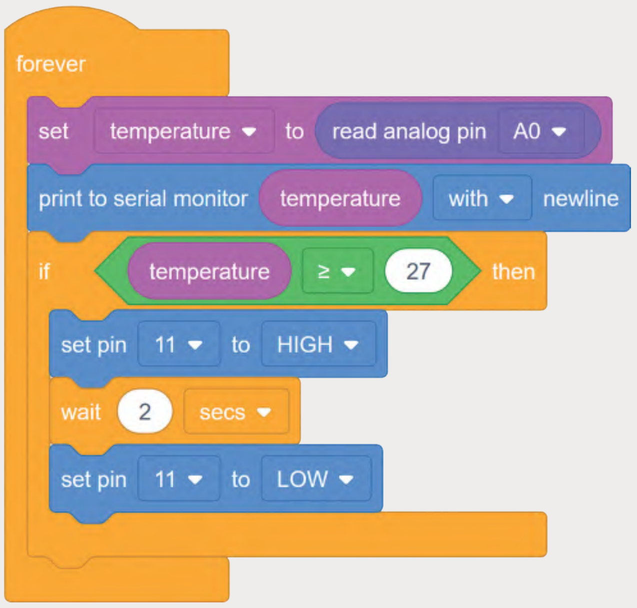


Figure 2.16: Connecting the circuit

Tinkercad blocks


forever
set temperature to read analog pin A0
print to serial monitor temperature with newline
if temperature ≥ 27 then
set pin 11 to HIGH
wait 2 secs
set pin 11 to LOW

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.

C++

```
int temperature = 0;
void setup() {
  pinMode(A0, INPUT);
  Serial.begin(9600);
  pinMode(11, OUTPUT);
}
void loop() {
  temperature = analogRead(A0);
  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 TMP 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

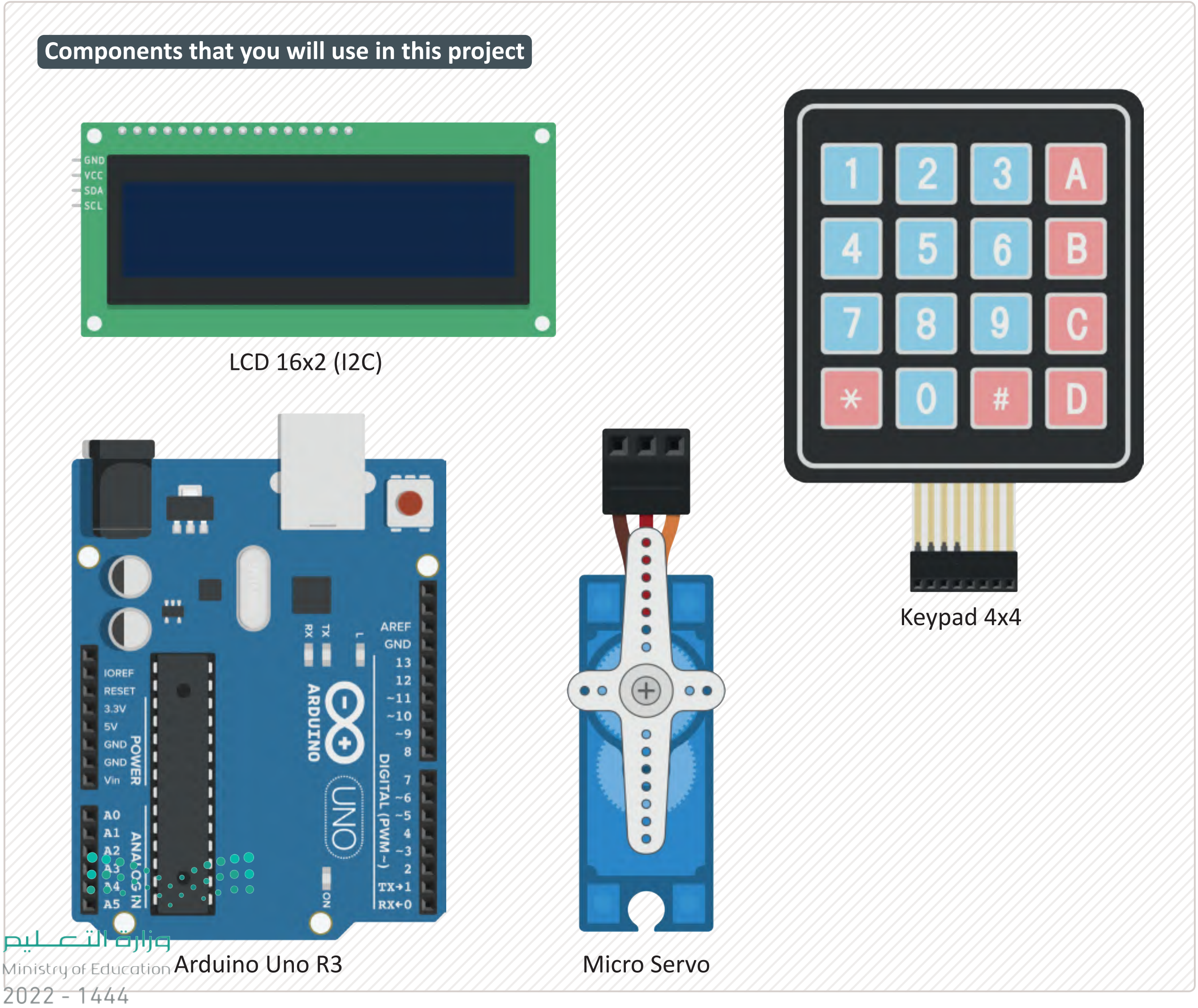


Figure 2.17: Project components

You will start by adding the keypad from the input category in the components and connect it to Arduino.

Connecting the Keypad:

> Find the **Keypad 4x4** component from the Input category in the components list and drag and drop it in the workplane. 1

> Connect the Row 1 of the Keypad to the Digital Pin 9 of the Arduino. 2

> Connect the Row 2 of the Keypad to the Digital Pin 8 of the Arduino. 3

> Connect the Row 3 of the Keypad to the Digital Pin 7 of the Arduino. 4

> Connect the Row 4 of the Keypad to the Digital Pin 6 of the Arduino. 5

> Connect the Column 1 of the Keypad to the Digital Pin 5 of the Arduino. 6

> Connect the Column 2 of the Keypad to the Digital Pin 4 of the Arduino. 7

> Connect the Column 3 of the Keypad to the Digital Pin 3 of the Arduino. 8

> Connect the Column 4 of the Keypad to the Digital Pin 2 of the Arduino. 9

> Change all wires color to green.

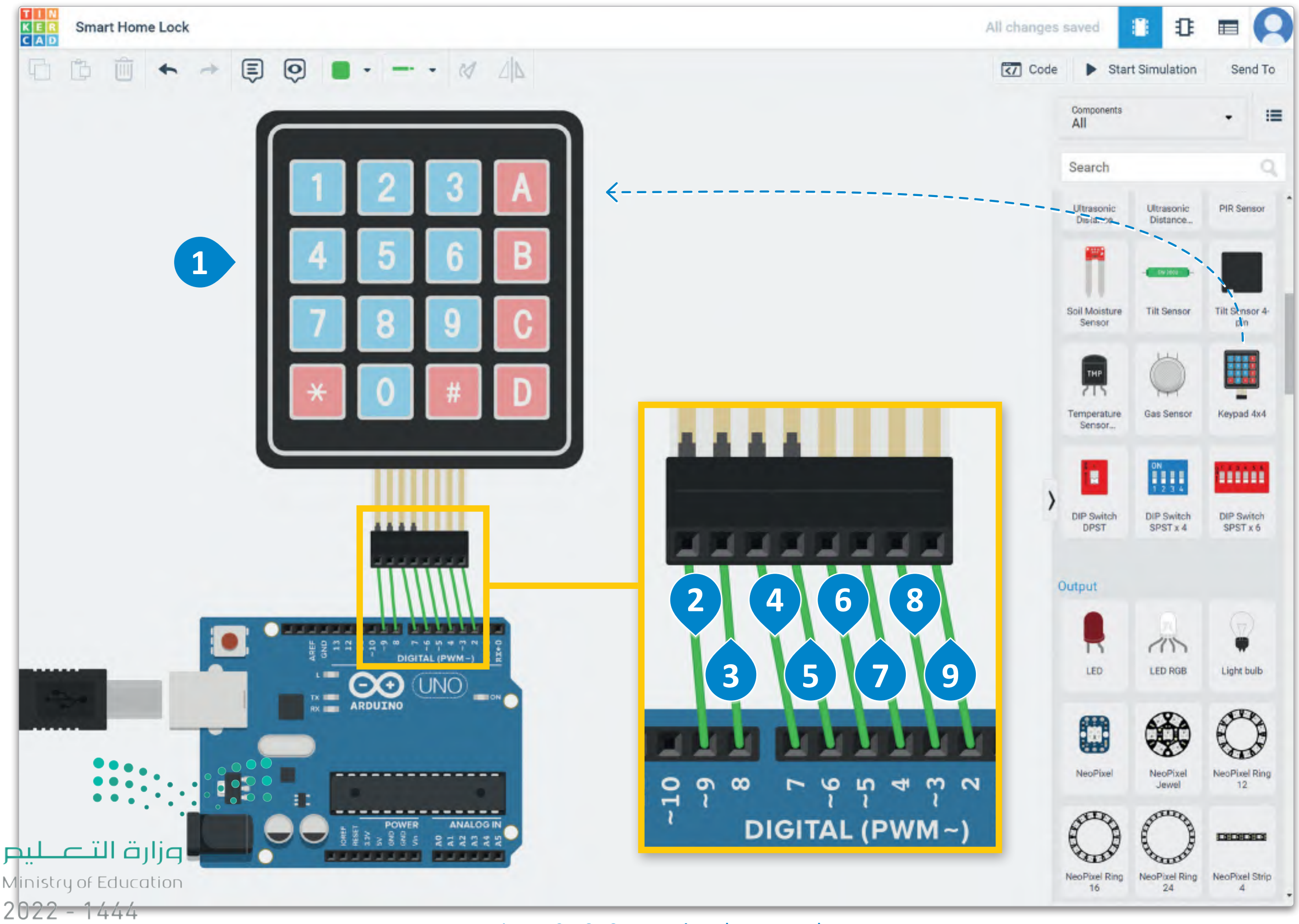


Figure 2.18: Connecting the Keypad

Now, find the LCD display from the ouput category in the components and connect it to the Breadboard.

Connecting 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

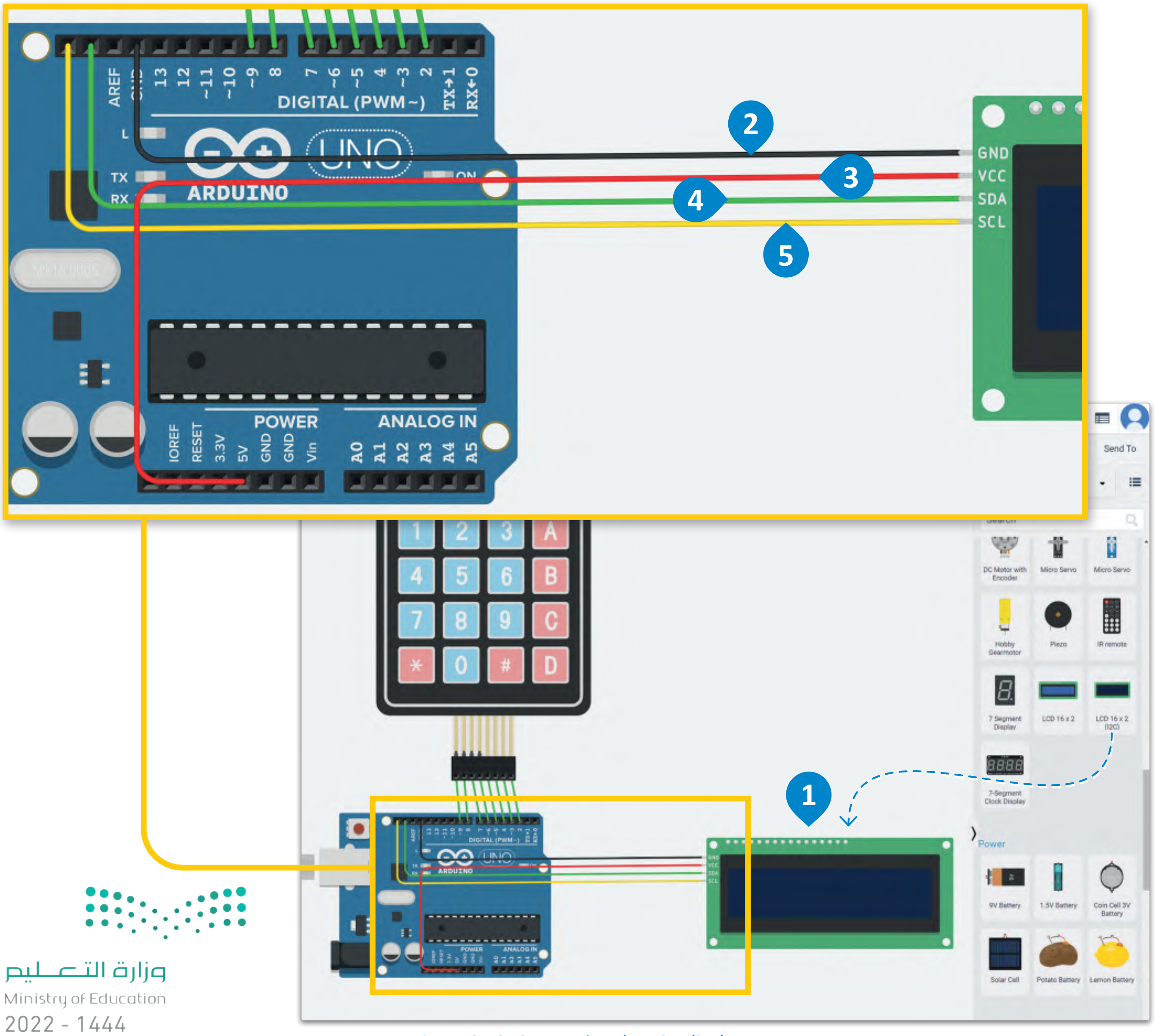


Figure 2.19: Connecting the LCD display

Finally, you will wire the Servo motor.

Find the **Servo motor** from the ouput category in the components and connect it to the Breadboard.

Connecting the Servo motor:

> 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 Servo motor to the GND of the Arduino and change the wire color to black. 2

> Connect the Power of the Servo motor to the 5V of the Arduino and change the wire color to red. 3

> Connect the Signal of the Servo motor to the Digital Pin 11 of the Arduino and change the wire color to orange. 4

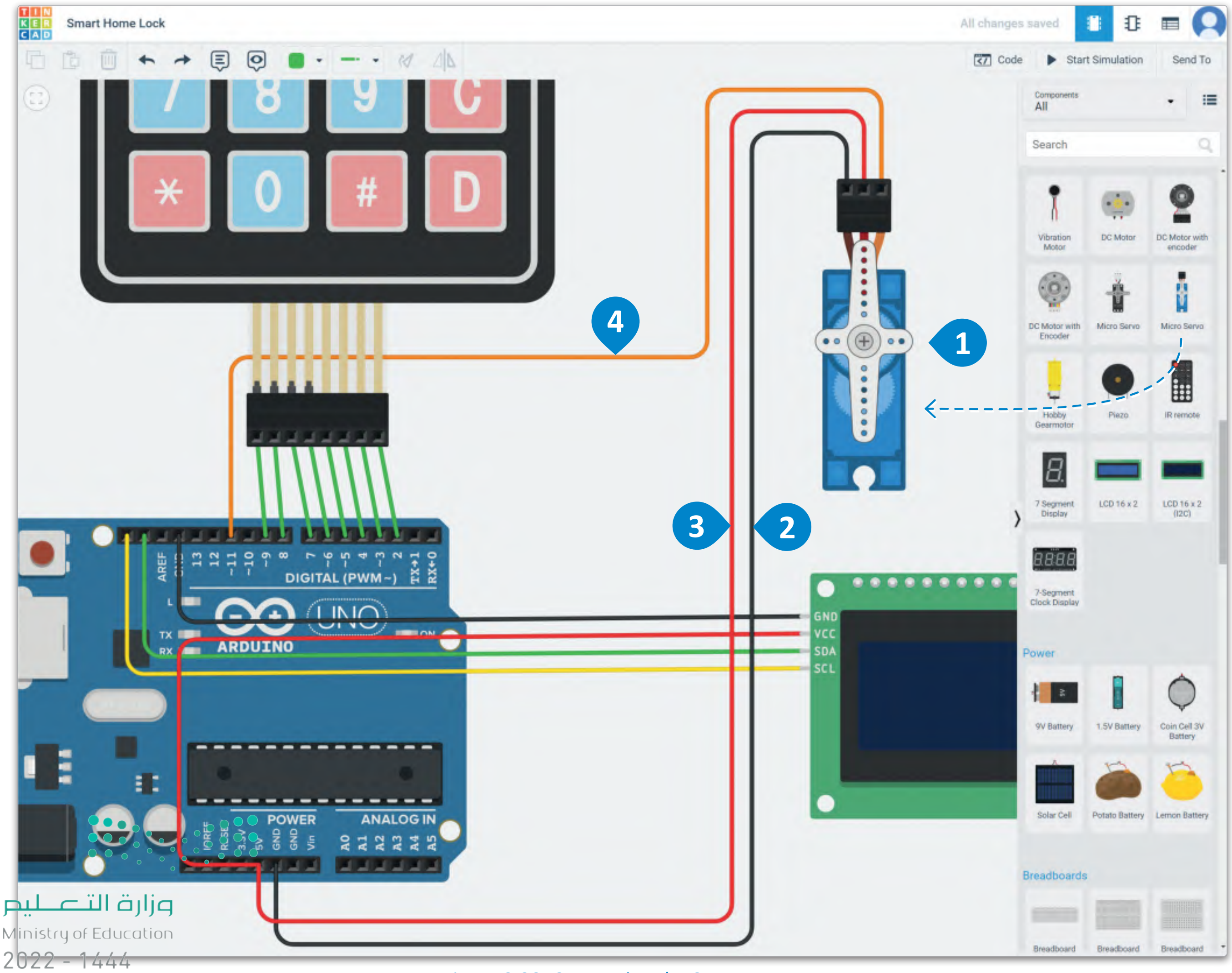


Figure 2.20: Connecting the Servo motor

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.

servo motor 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.

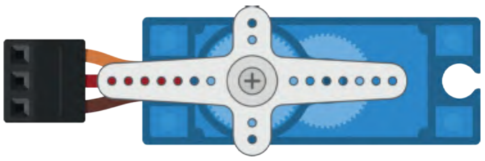


Figure 2.21:
Servo motor (Tinkercad object)

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 2.22: LCD display (Tinkercad object)

Keypad object

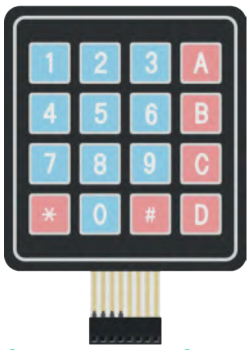


Figure 2.23:
Keypad (Tinkercad object)

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.

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();
    }
}
```

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 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);**

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:

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);
```

```
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!");
  }
}
```

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++	LCD display	setup()
Class	loop()	Smart Security
High Level Programming Language	Object Oriented Programming Language	
Keypad	Object	

3. 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**

Lesson 1

Smart Cities and the MQTT Protocol

Link to digital lesson

www.ien.edu.sa

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.

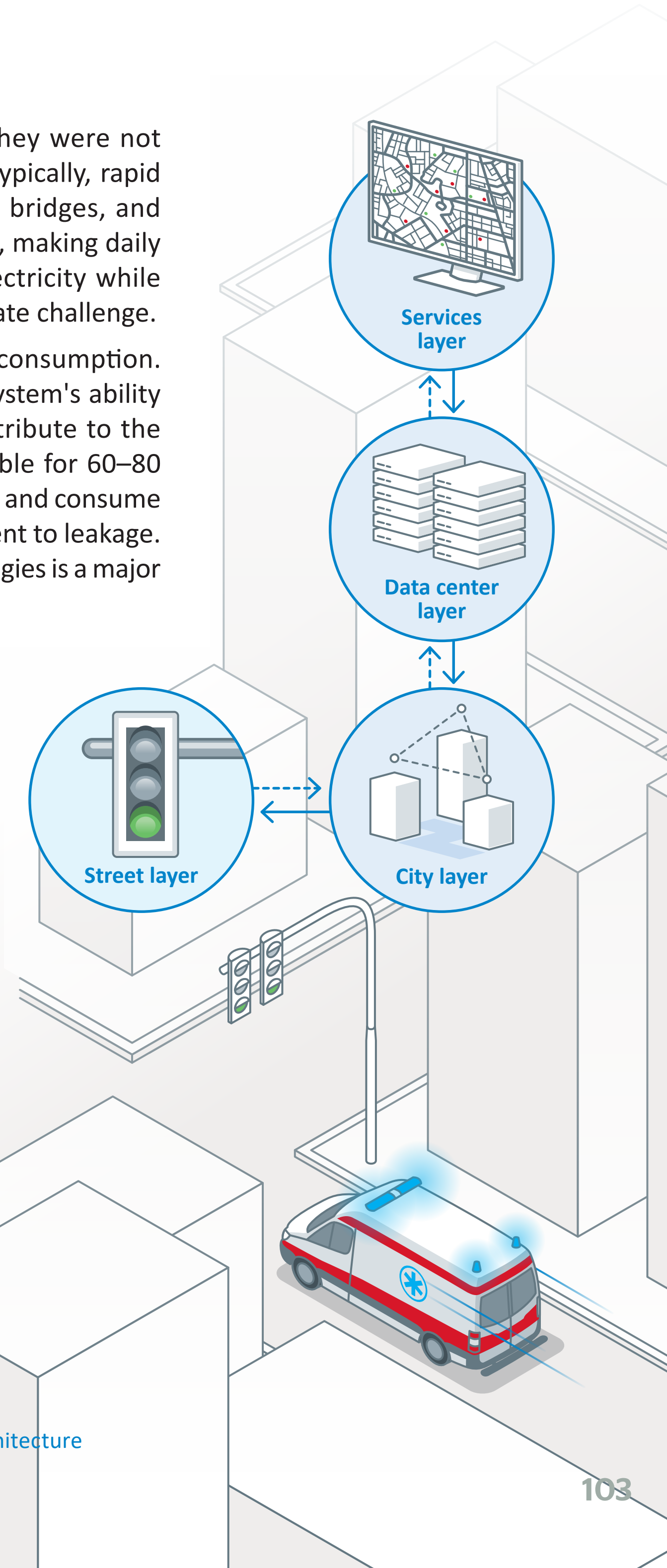


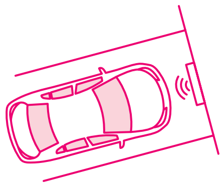
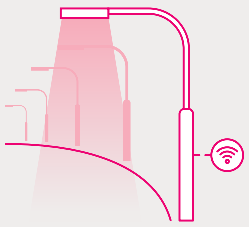
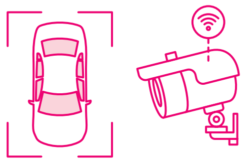
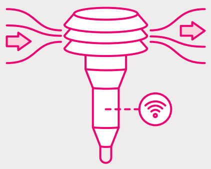
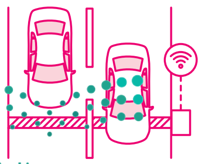
Figure 3.1: A Smart City IoT Architecture

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 3.1: 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 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.

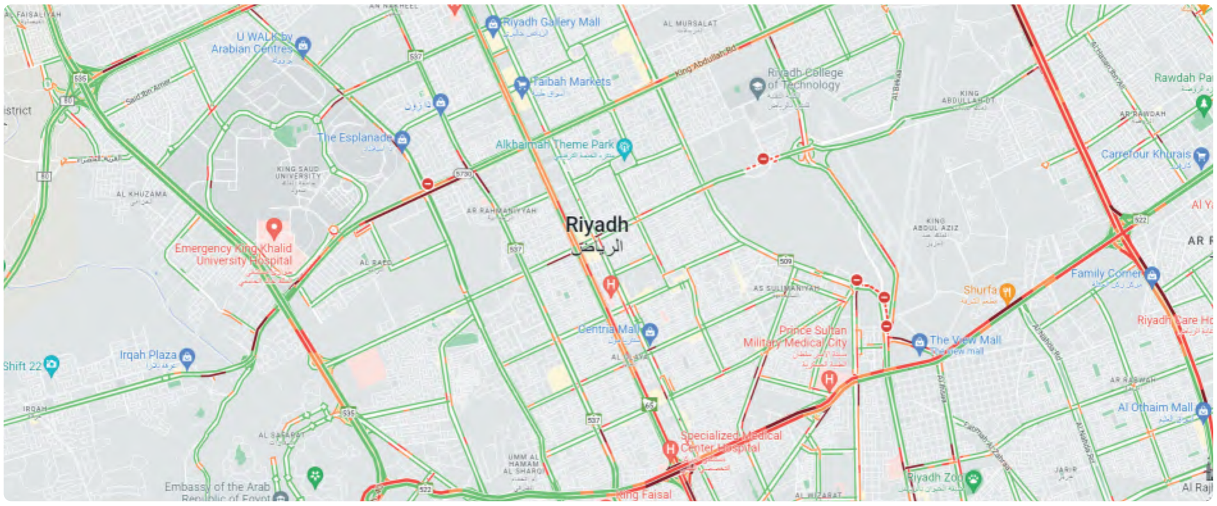


Figure 3.2: Traffic update in real time

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 3.3: Connected Street Lighting

Figure 3.4: 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 3.5: Smart Air-quality Station

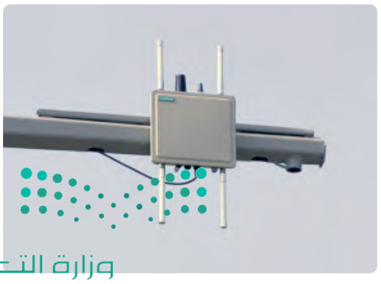


Figure 3.6: Roadside Unit (RSU)

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.

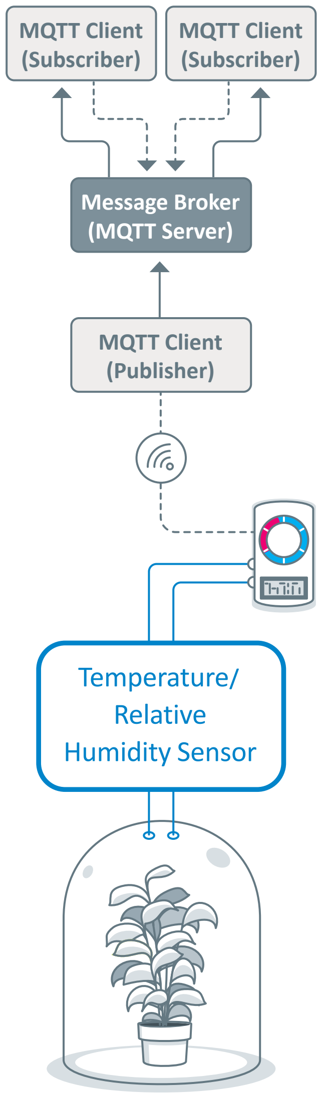


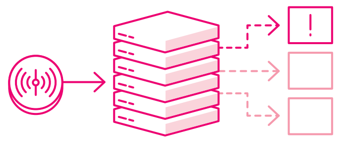
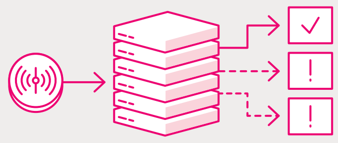
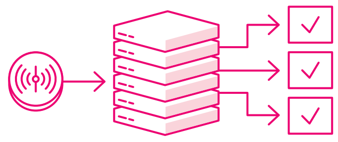
Figure 3.7: MQTT functionality

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 3.2: 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 Create a diagram showing how data flows in a smart city IoT architecture.

4 Provide examples of how sensors are used in a smart city street layer.

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 Classify the three degrees of Quality Of Service for the MQTT protocol.

9 Create a diagram with an example of three devices connected by the MQTT protocol, one publisher, and two subscribers.

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.

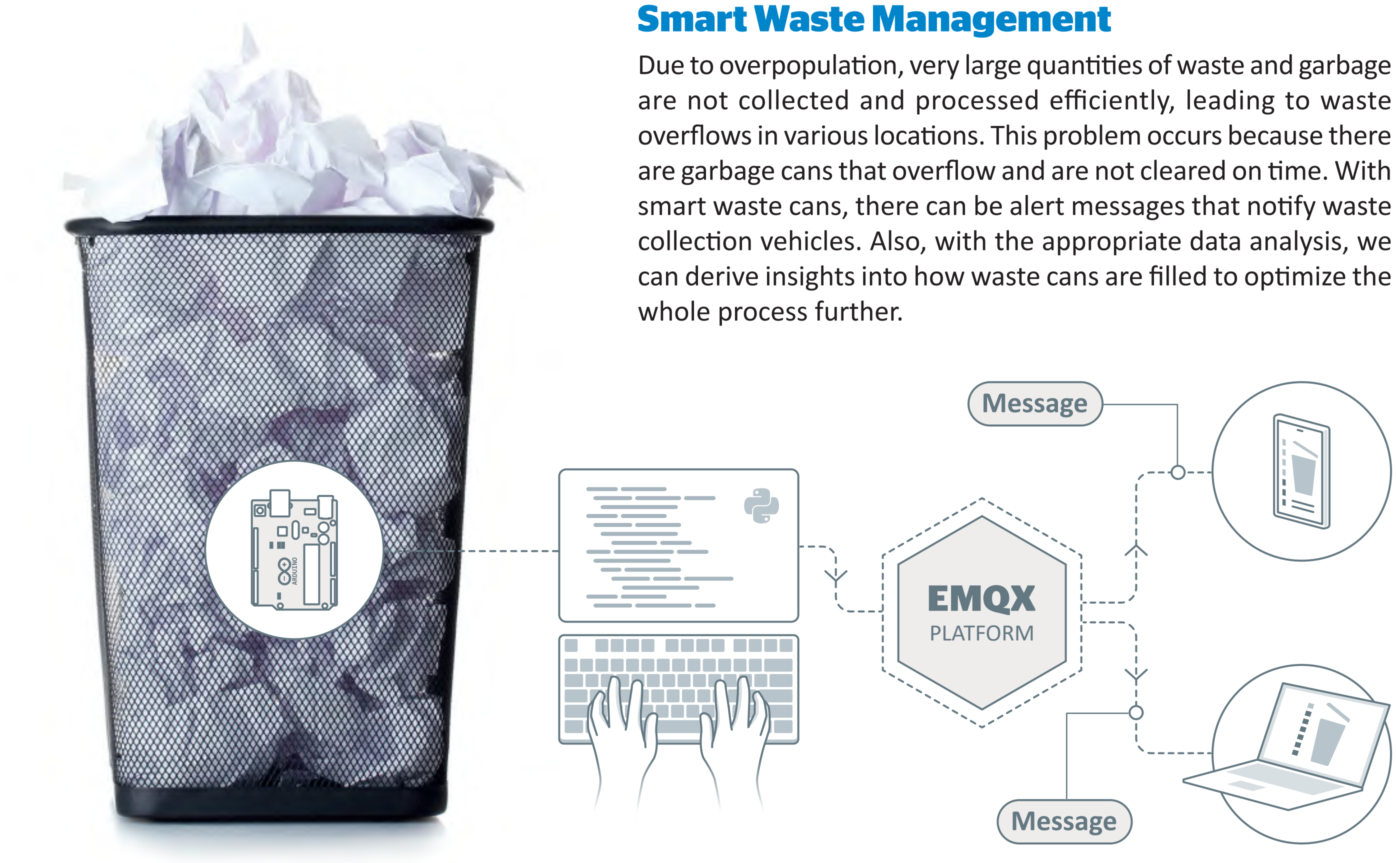


Figure 3.8: Smart Waste Management Arduino & MQTT project

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

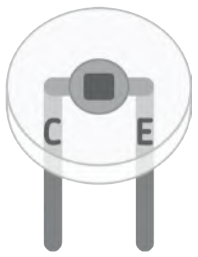
EMQX is an open-source MQTT broker with a high-performance real-time message processing engine. It is used to support event streaming for an extremely large number of IoT devices.

Components & Tools for Project

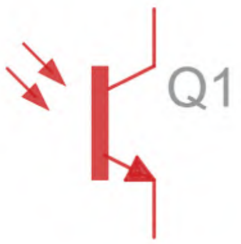
Phototransistor

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



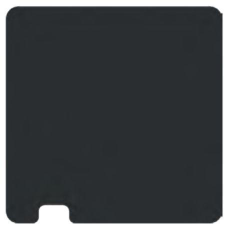
Schematic symbol

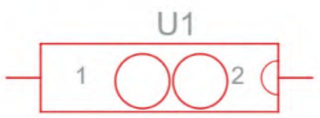
Figure 3.9: Phototransistor

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 3.10: 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 (1kΩ)

Components that you will use in this project

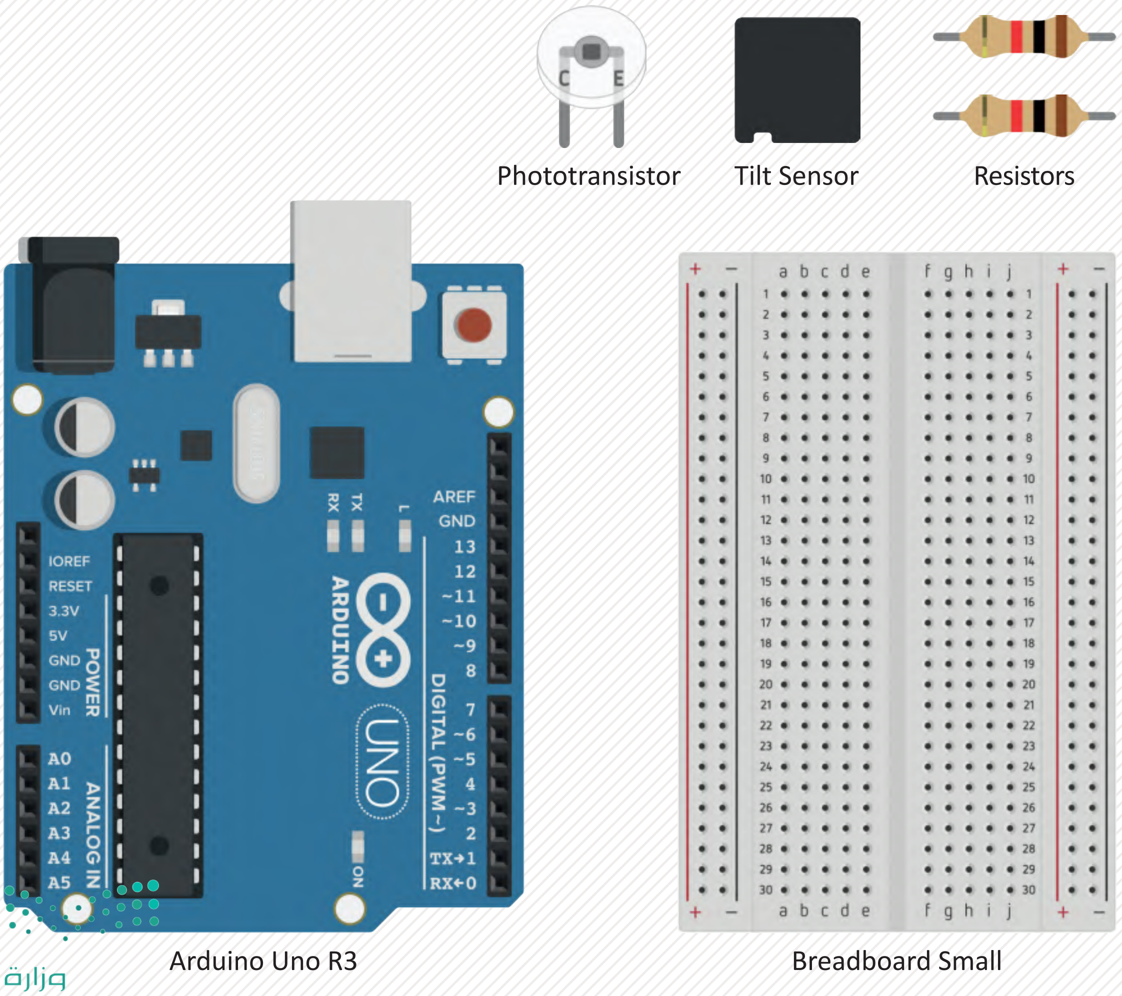


Figure 3.11: Smart Waste project components

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.

Setting up the connection to the EMQX broker with MQTTX:

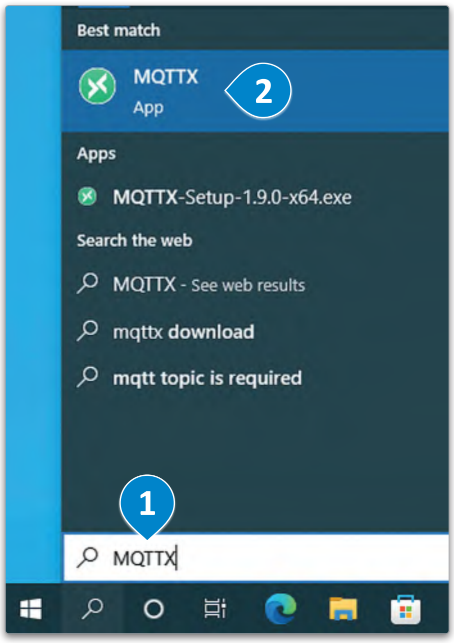
> Click the Windows search button and type MQTTX. 1

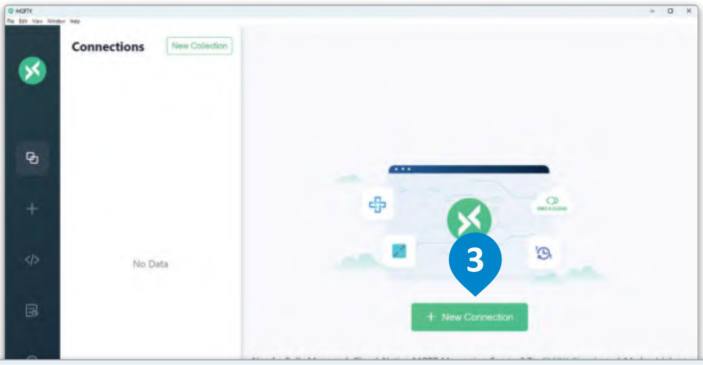
> Open the MQTTX client desktop app. 2

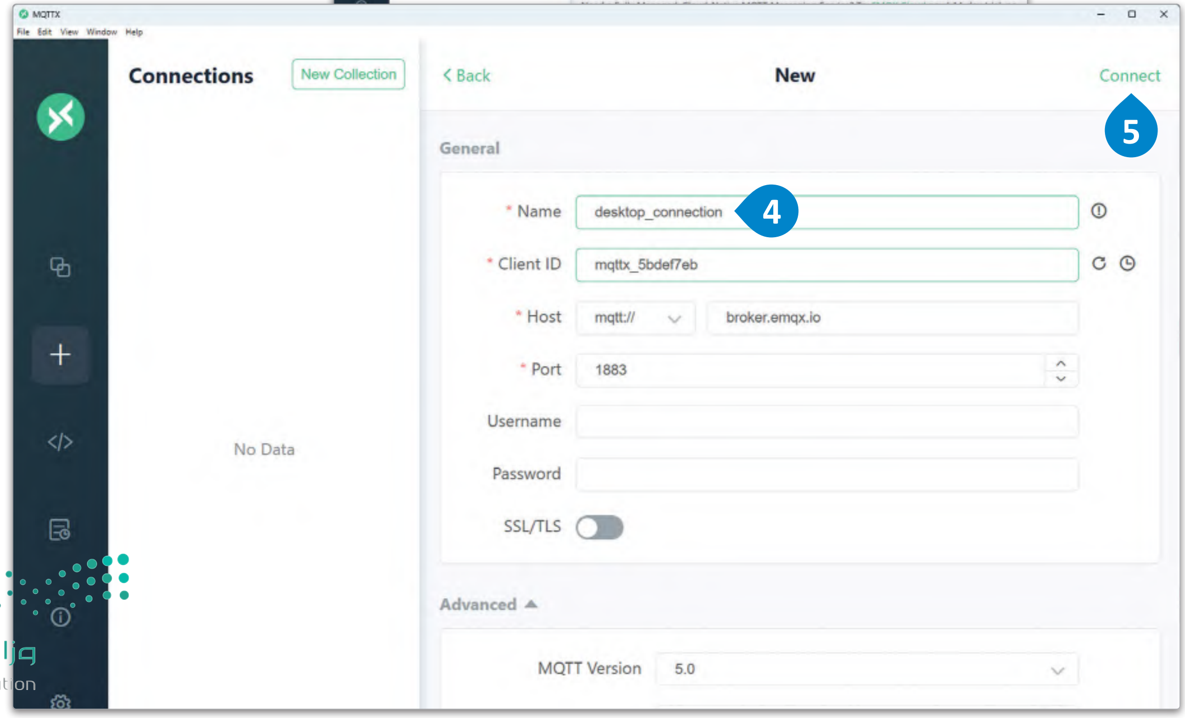
> Click **New Connection**, to create a new connection. 3

> Type a name for the connection e.g. **desktop_connection**. 4

> Click on the **Connect** button. 5







* Name: desktop_connection
* Client ID: mqttx_5bdef7eb
* Host: mqtt:// broker.emqx.io
* Port: 1883

Username

Password

SSL/TLS

MQTT Version: 5.0

Figure 3.12: 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.

Loading 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

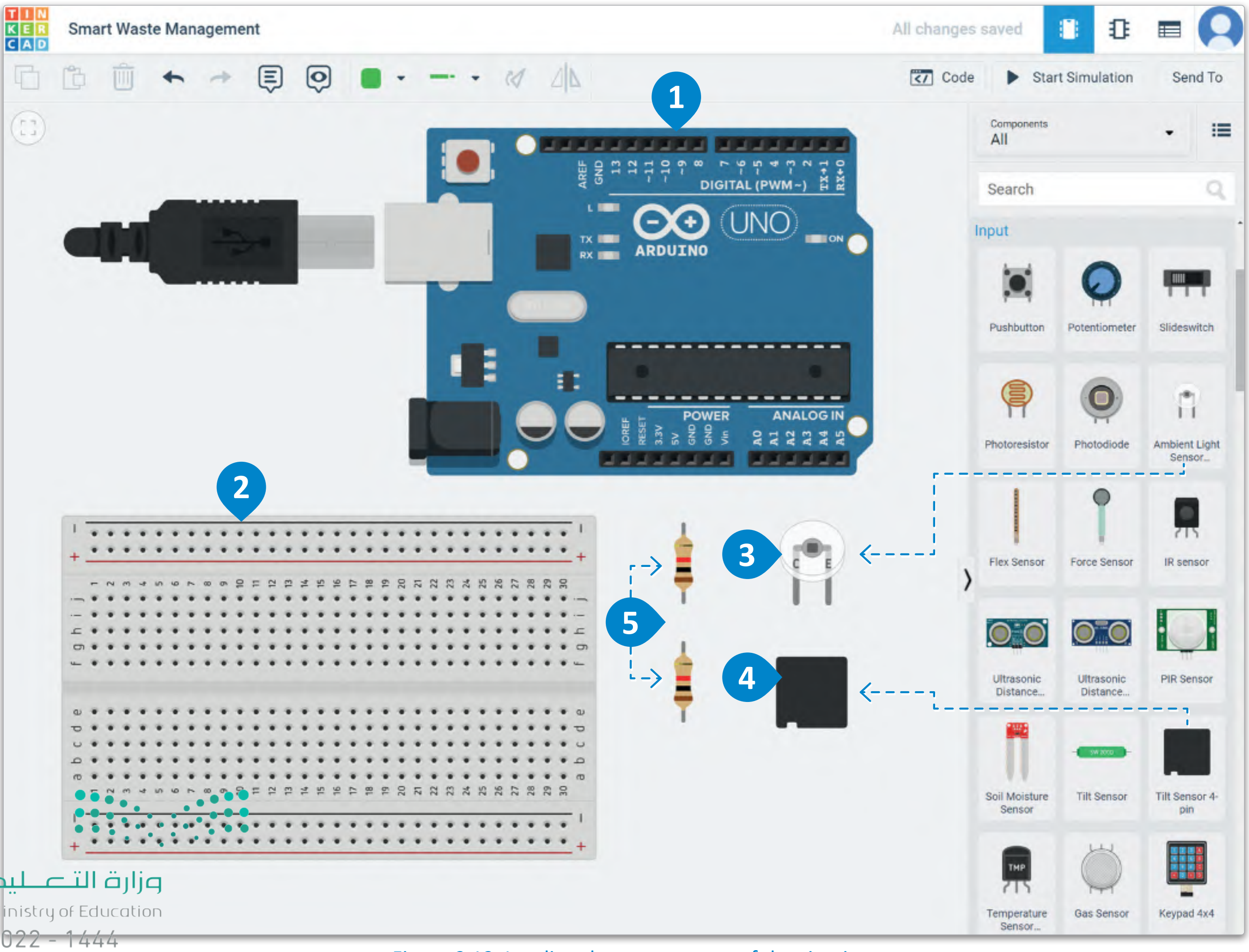


Figure 3.13: Loading the components of the circuit

Connecting 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

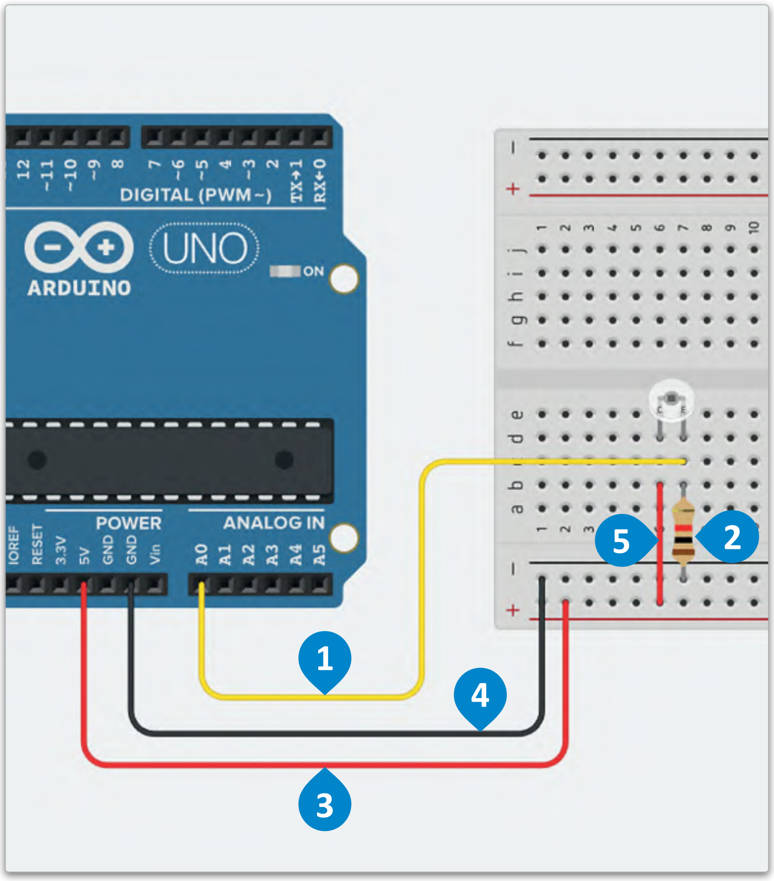


Figure 3.14: Connect the Phototransistor

Connecting 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

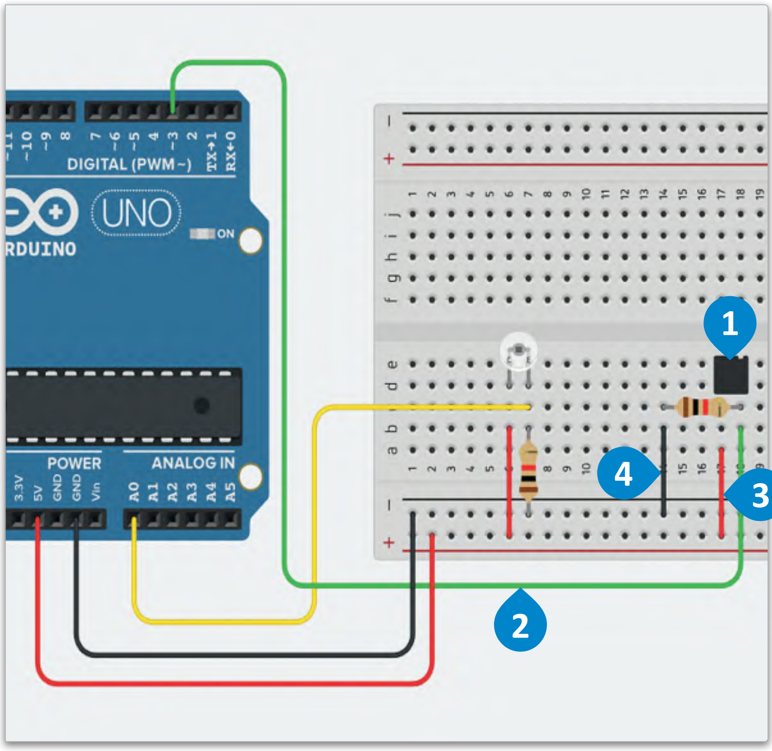


Figure 3.15: Connect the Tilt Sensor

Complete Circuit

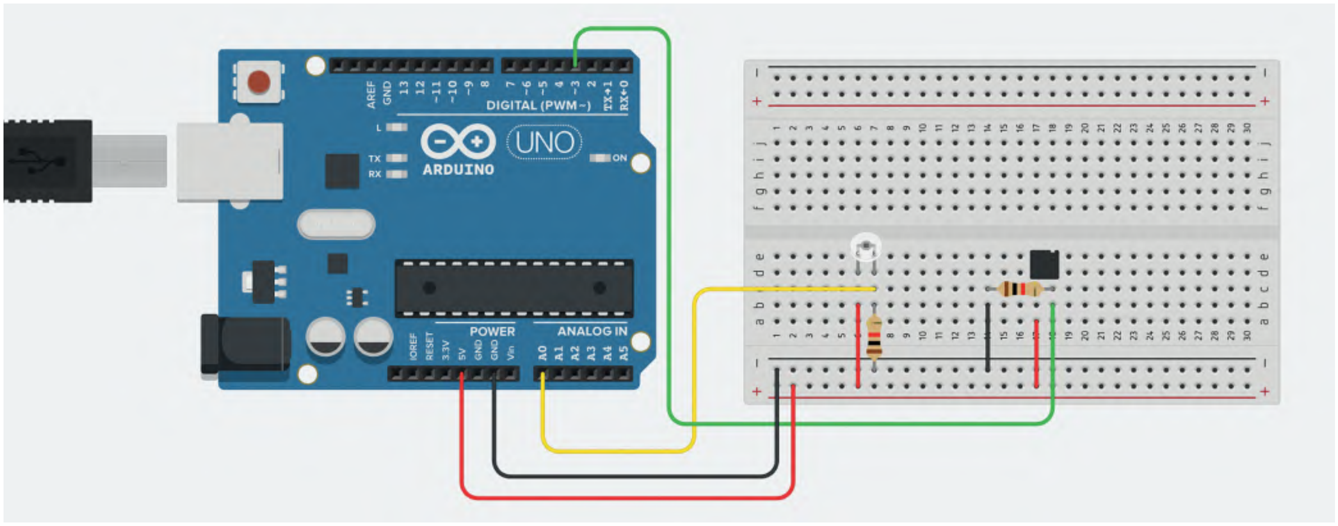


Figure 3.16: Complete circuit in Tinkercad

Physical Circuit

This photo represents what the physical circuit will look like.

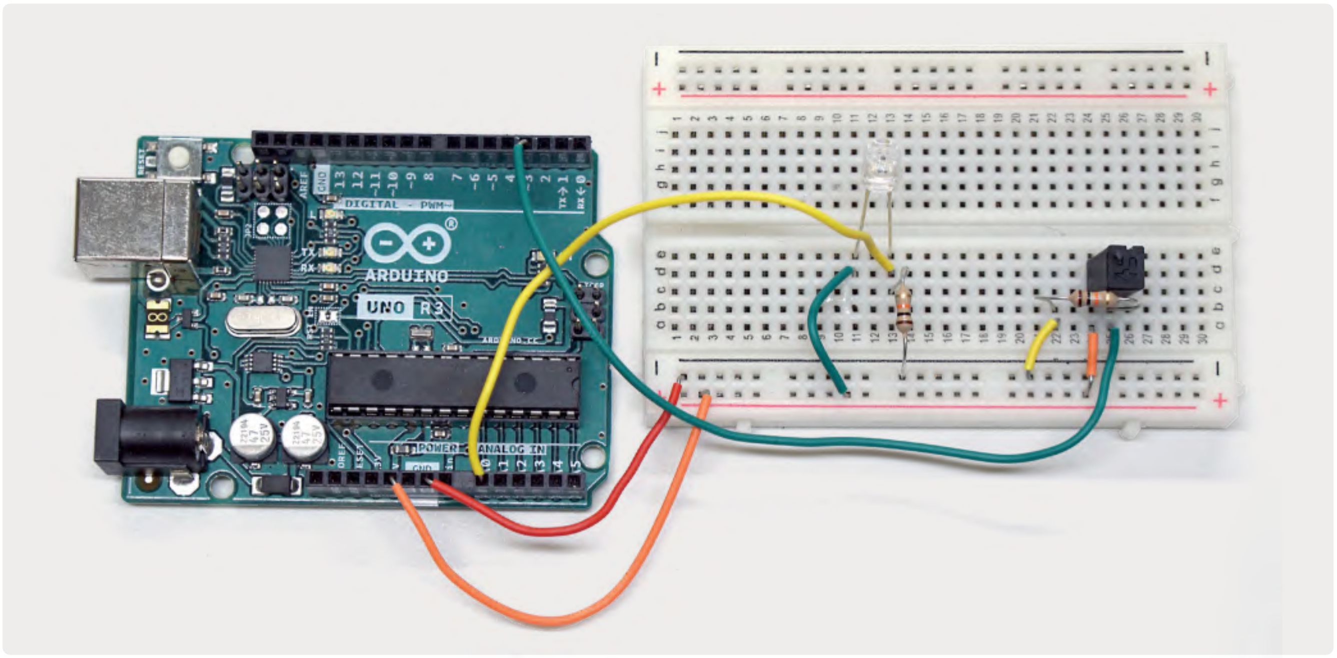


Figure 3.18: Photo of the physical circuit

The components are connected to the following pins:

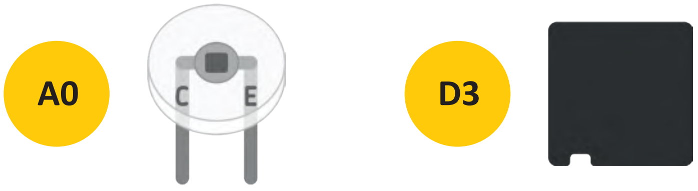


Figure 3.17: Pins connected to components

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

```
pip install paho.mqtt.client
```

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 3.3: 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 protypes 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.

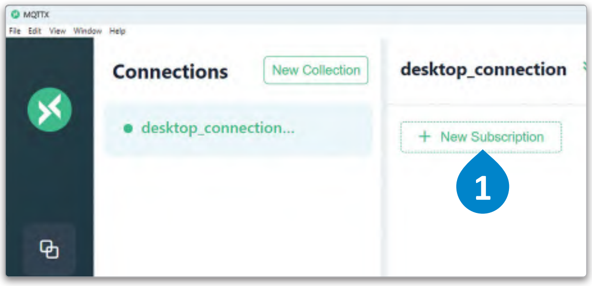
To test that your messages were published correctly you will use the **MQTTX** desktop client. Before executing the Python script you will use the MQTTX client to subscribe to the **"waste/drops"** topic. Now the client will wait to receive the messages that are published through the Python script and are distributed through the **EMQX** public broker.

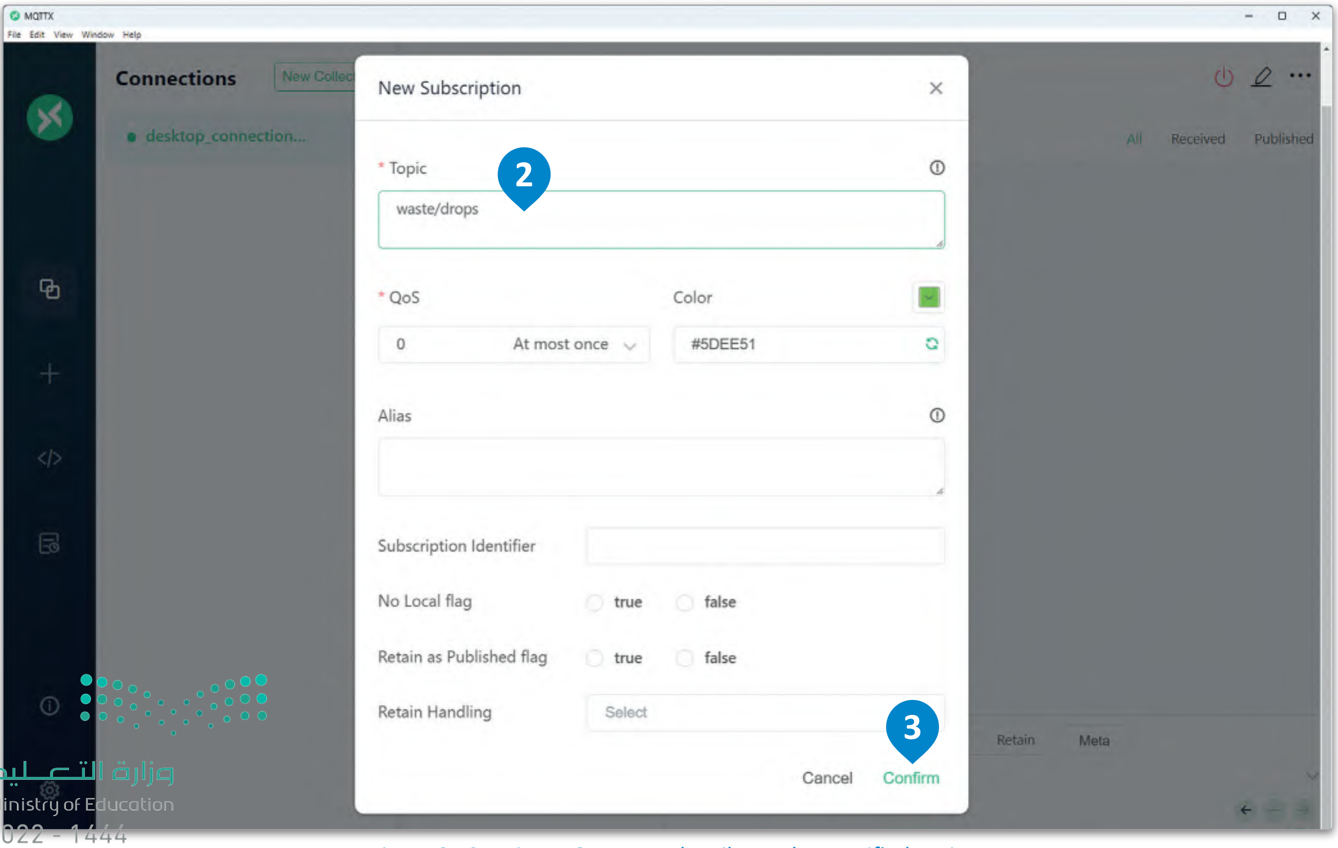
Using MQTTX to subscribe to the specified topic:

> 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





New Subscription

* Topic: waste/drops

* QoS: 0 At most once

Color: #5DEE51

Alias:

Subscription Identifier:

No Local flag: true false

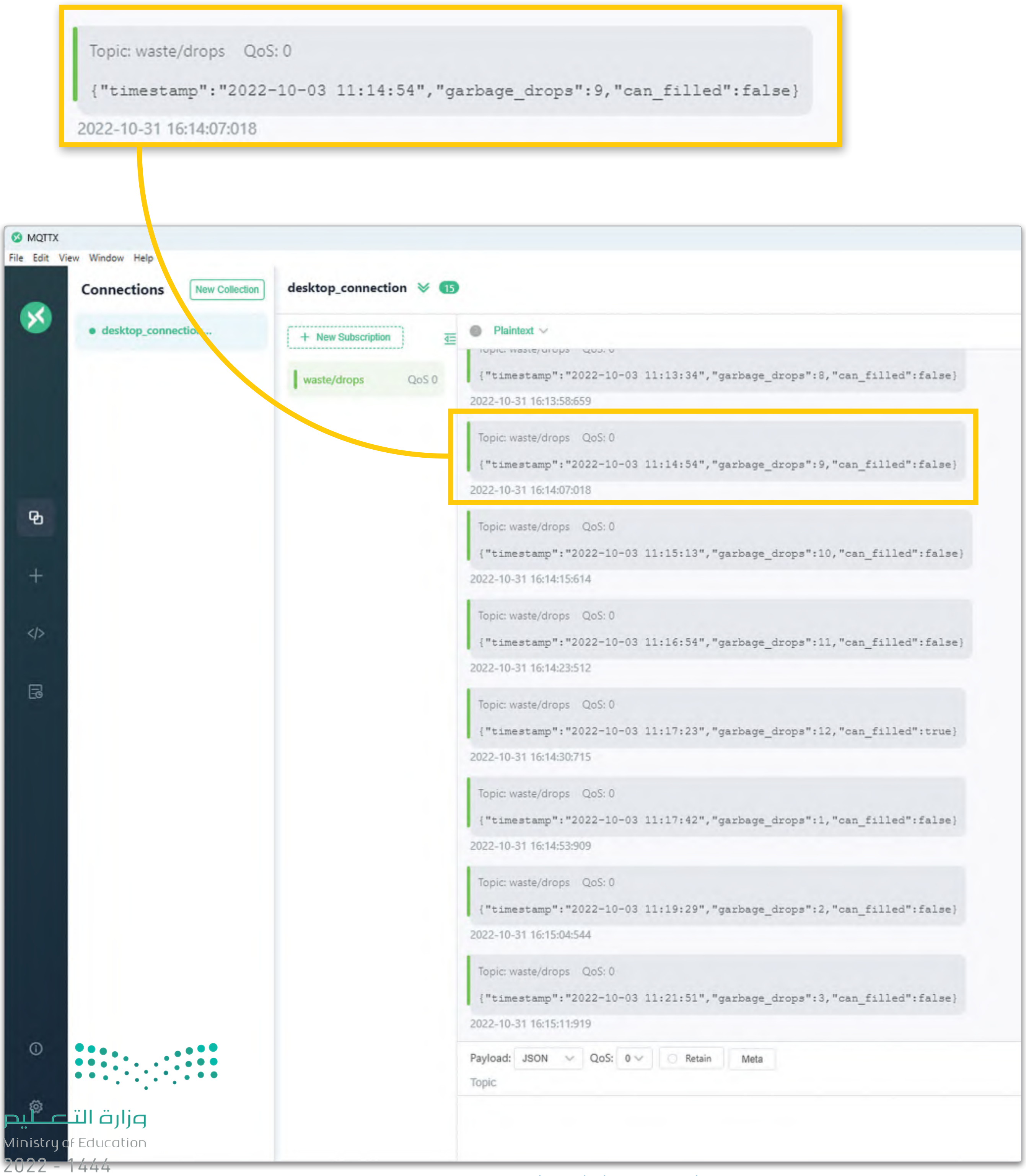
Retain as Published flag: true false

Retain Handling: Select

Figure 3.19: 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
```

```
{"timestamp":"2022-10-03 11:13:34","garbage_drops":8,"can_filled":false}
2022-10-31 16:13:58:659

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

Topic: waste/drops    QoS: 0
{"timestamp":"2022-10-03 11:15:13","garbage_drops":10,"can_filled":false}
2022-10-31 16:14:15:614

Topic: waste/drops    QoS: 0
{"timestamp":"2022-10-03 11:16:54","garbage_drops":11,"can_filled":false}
2022-10-31 16:14:23:512

Topic: waste/drops    QoS: 0
{"timestamp":"2022-10-03 11:17:23","garbage_drops":12,"can_filled":true}
2022-10-31 16:14:30:715

Topic: waste/drops    QoS: 0
{"timestamp":"2022-10-03 11:17:42","garbage_drops":1,"can_filled":false}
2022-10-31 16:14:53:909

Topic: waste/drops    QoS: 0
{"timestamp":"2022-10-03 11:19:29","garbage_drops":2,"can_filled":false}
2022-10-31 16:15:04:544

Topic: waste/drops    QoS: 0
{"timestamp":"2022-10-03 11:21:51","garbage_drops":3,"can_filled":false}
2022-10-31 16:15:11:919
```

Figure 3.20: 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.

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 publishes 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.

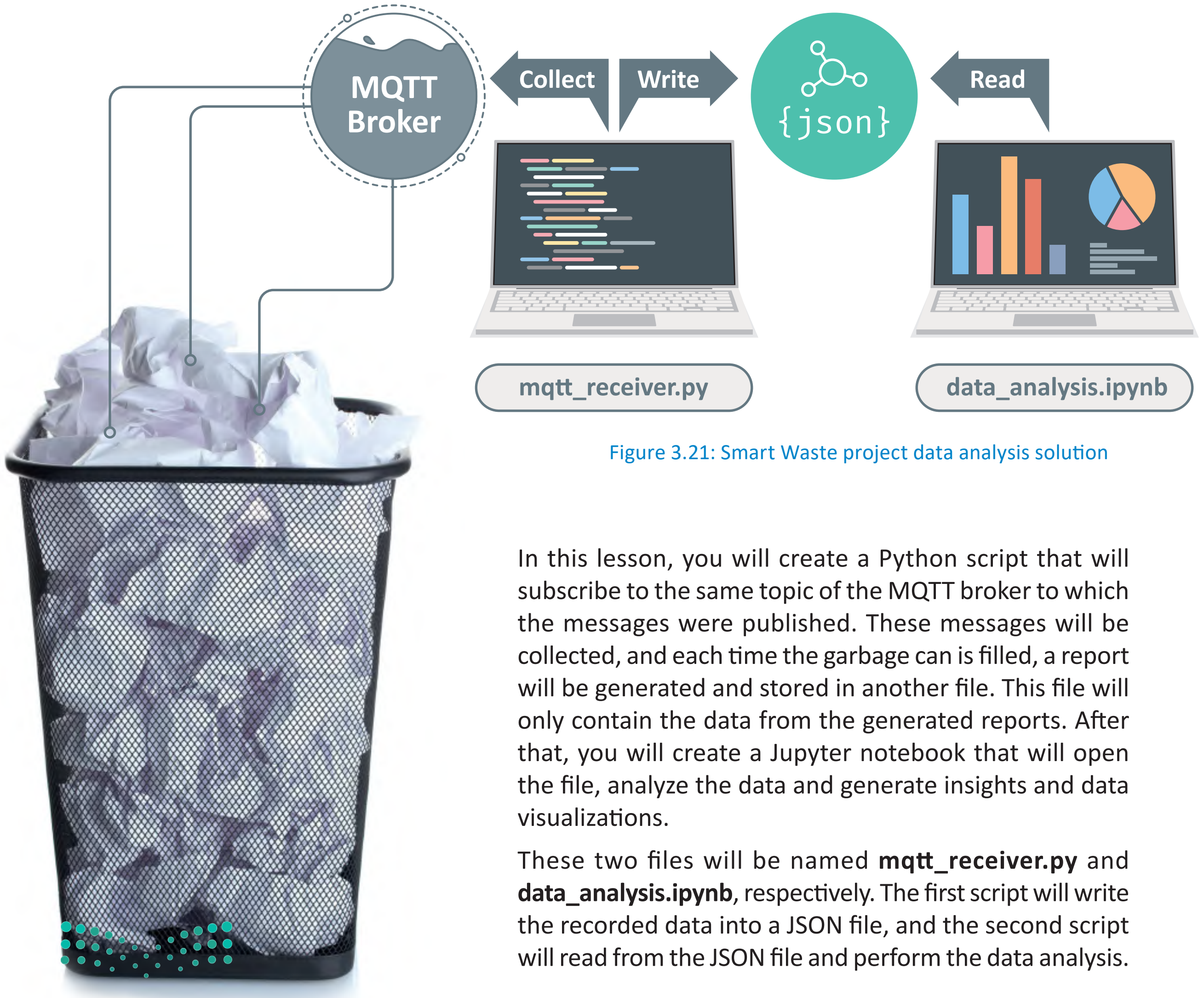


Figure 3.21: Smart Waste project data analysis solution

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.

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 mesages 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.

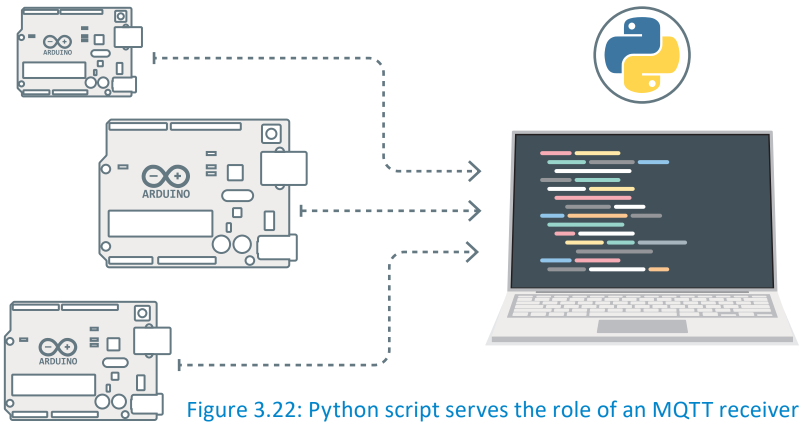


Figure 3.22: Python script serves the role of an MQTT receiver

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.

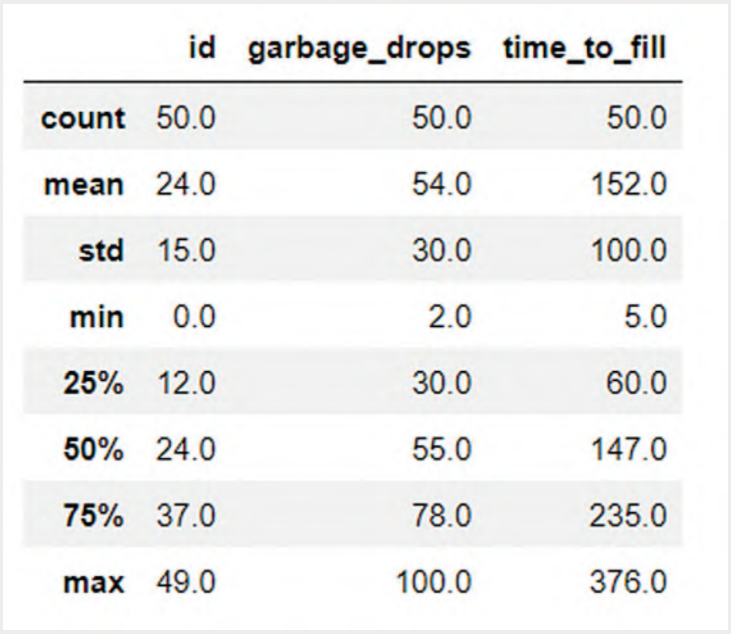
The JSON file is available for download here:
http://binary-academy.com/dnld/KSA/IOT2/U3_L3_DATA.json

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)
```



	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 3.23: 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)
```

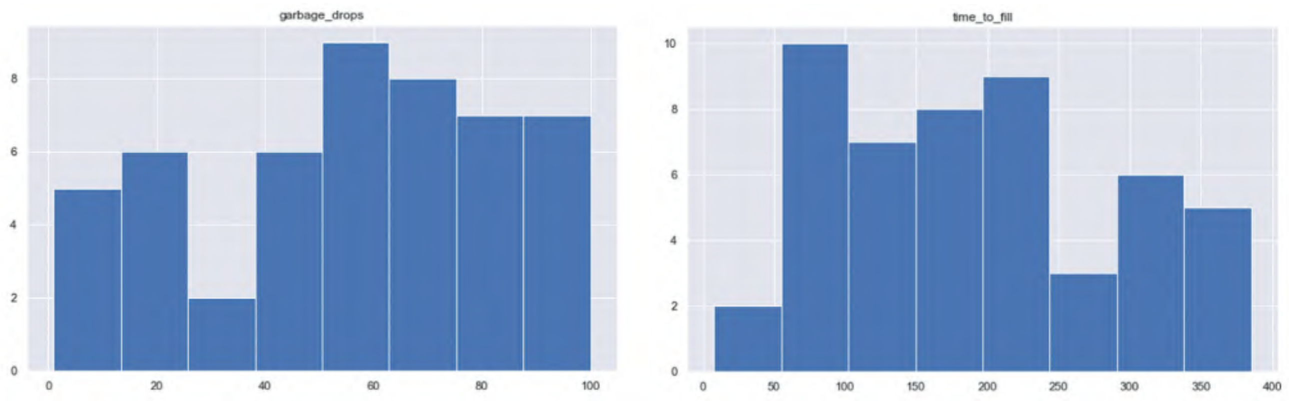


Figure 3.24: Histograms

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');
```

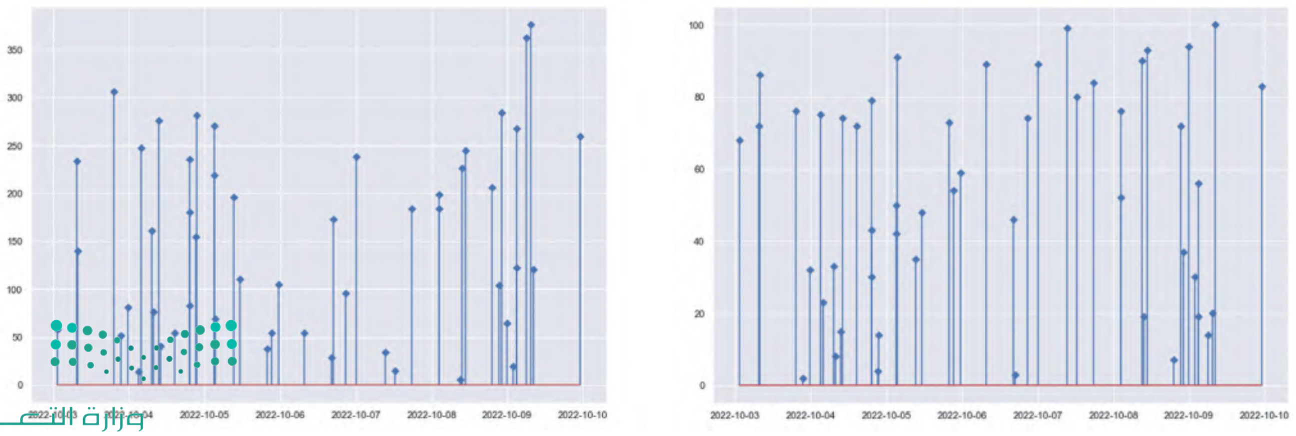


Figure 3.25: Stem plots

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);
```

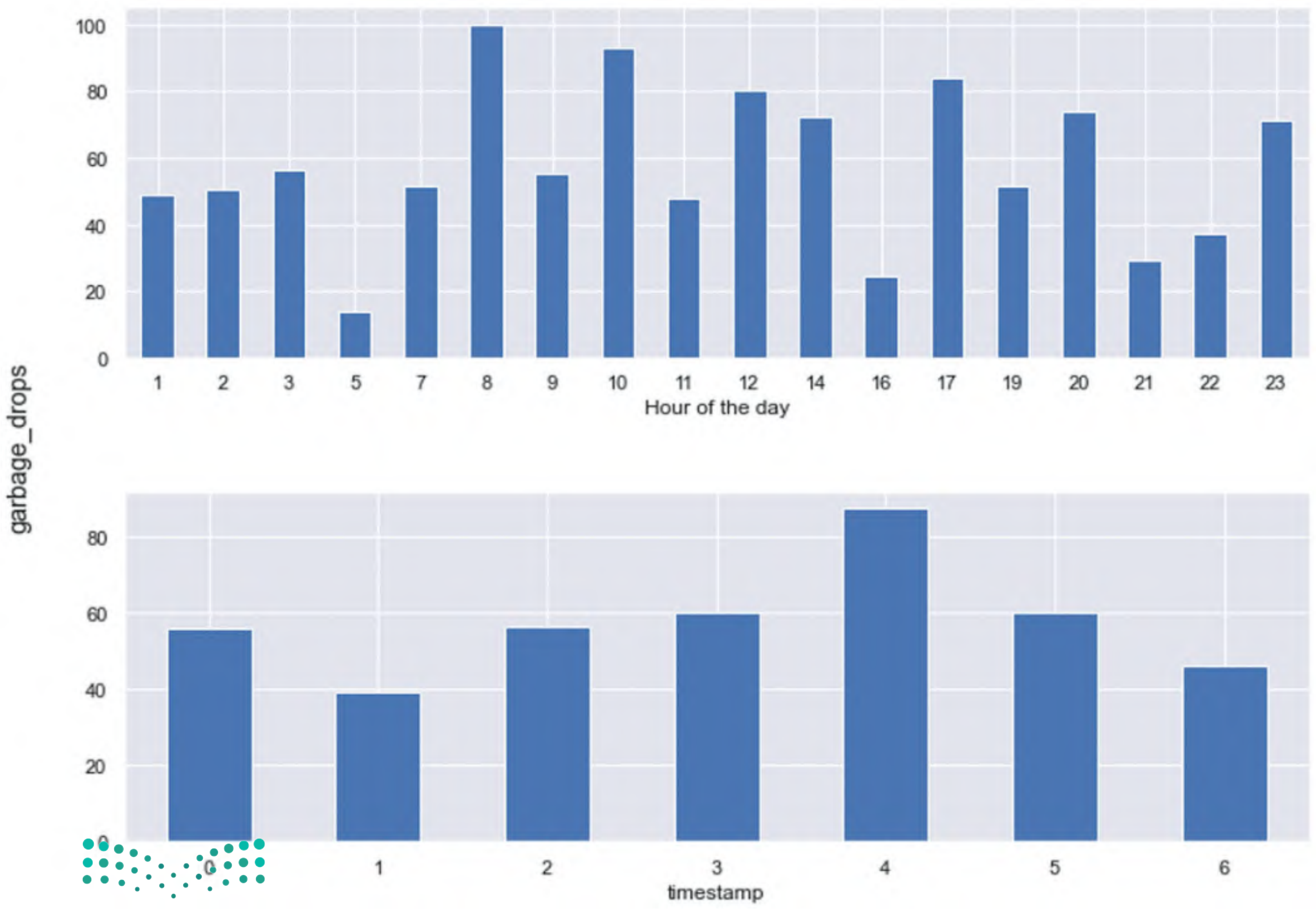


Figure 3.26: Group by plots

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	Tracker

4. 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.

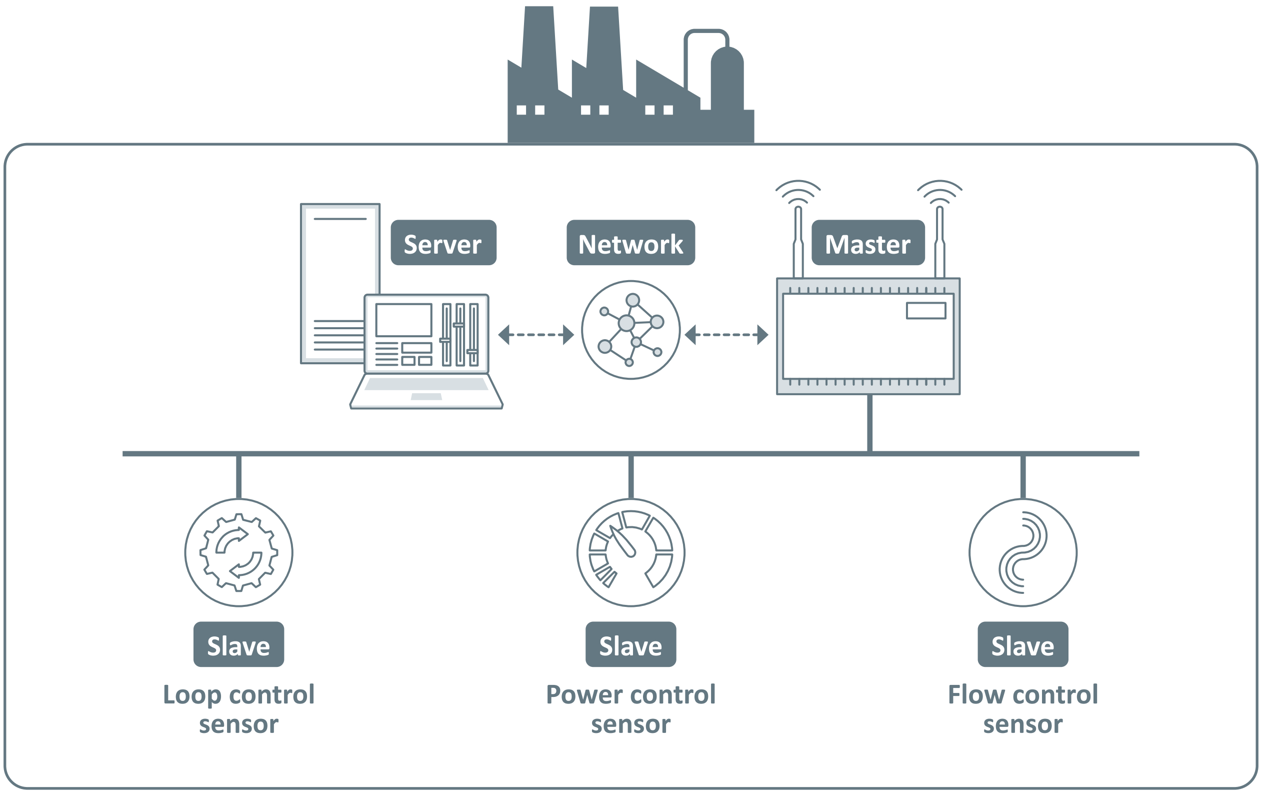


Figure 4.1: Modbus network protocol

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 4.2: 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

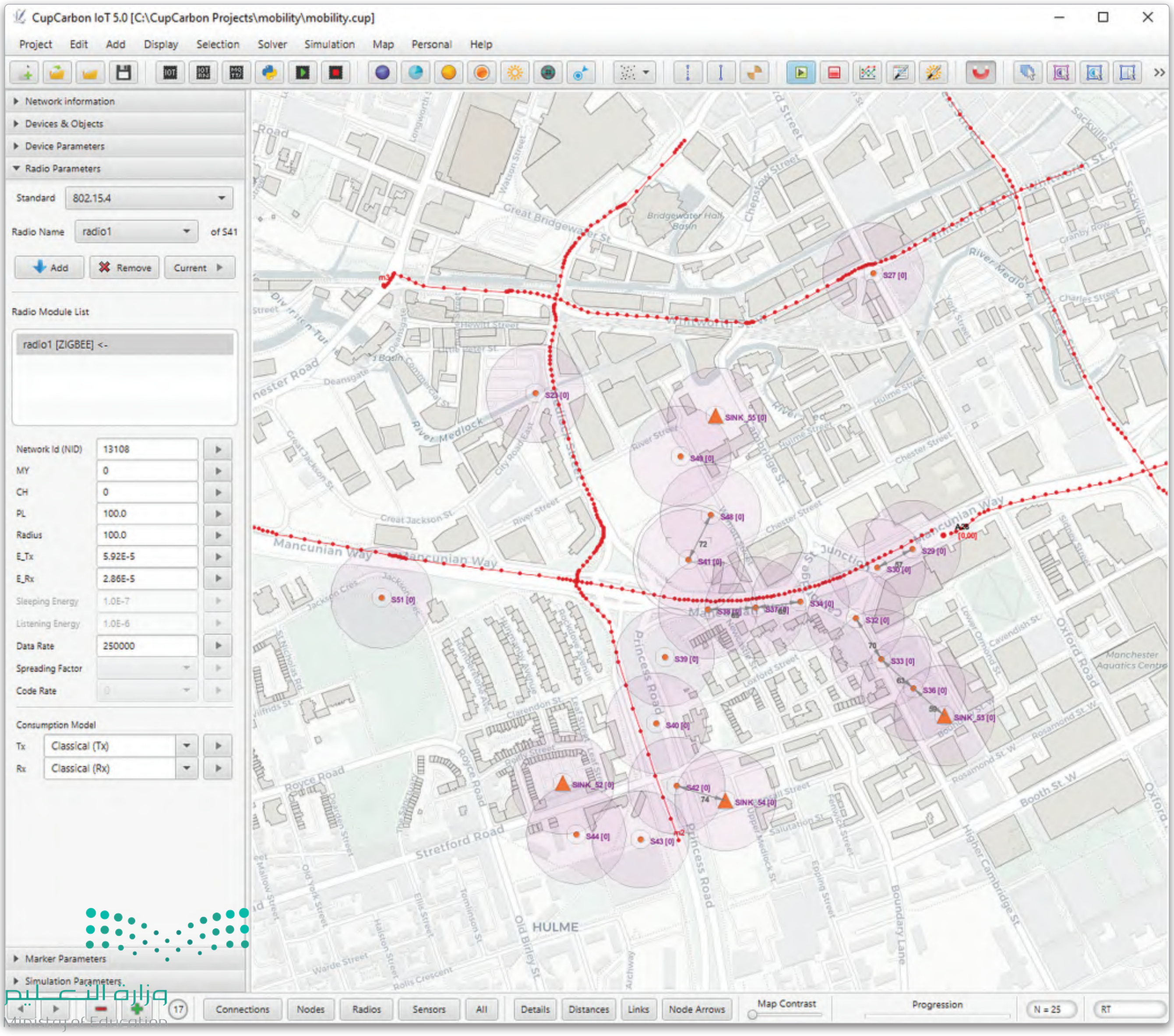
The Uthmaniyah Gas Plant is one of KSA's largest gas processing plants. IoT and AI solutions enhance productivity and facility reliability. Drones and smart objects monitor gas pipeline refinery equipment and use data analysis methods to optimize their usage. Thousands of IoT-enabled sensors monitor 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.

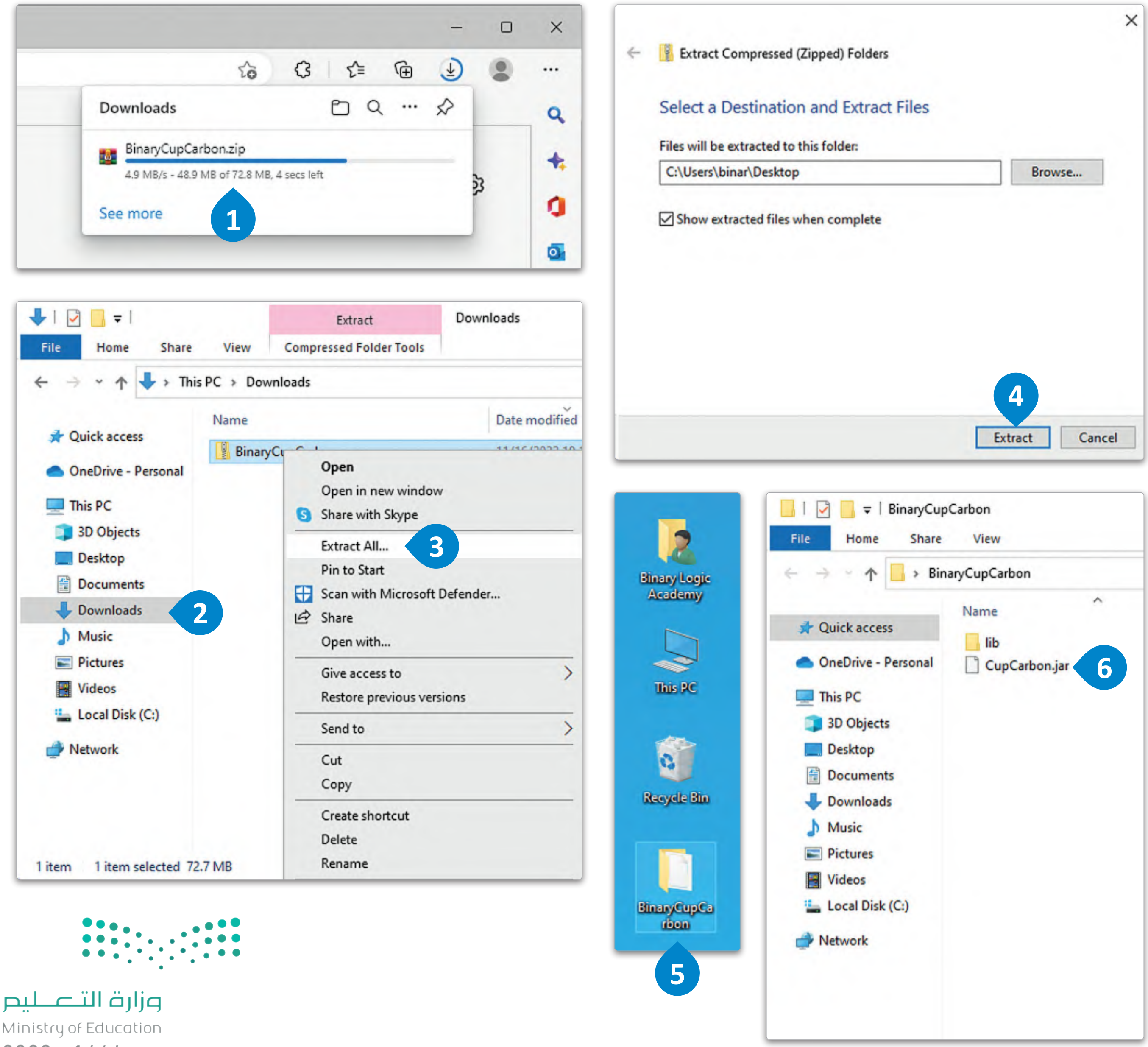


Standard: 802.15.4
Radio Name: radio1 of S41
radio1 [ZIGBEE] <-
Network Id (NID): 13108
MY: 0
CH: 0
PL: 100.0
Radius: 100.0
E_Tx: 5.92E-5
E_Rx: 2.86E-5
Sleeping Energy: 1.0E-7
Listening Energy: 1.0E-6
Data Rate: 250000
Spreading Factor:
Code Rate: 0
Consumption Model
Tx: Classical (Tx)
Rx: Classical (Rx)

Figure 4.3: 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



Files will be extracted to this folder: C:\Users\binar\Desktop
☑ Show extracted files when complete

Figure 4.4: Downloading CupCarbon

وزارة التعليم
Ministry of Education
2022 - 1444

The CupCarbon Windows

When you open the program you will notice two windows: the Main Window containing the Map and the Console.

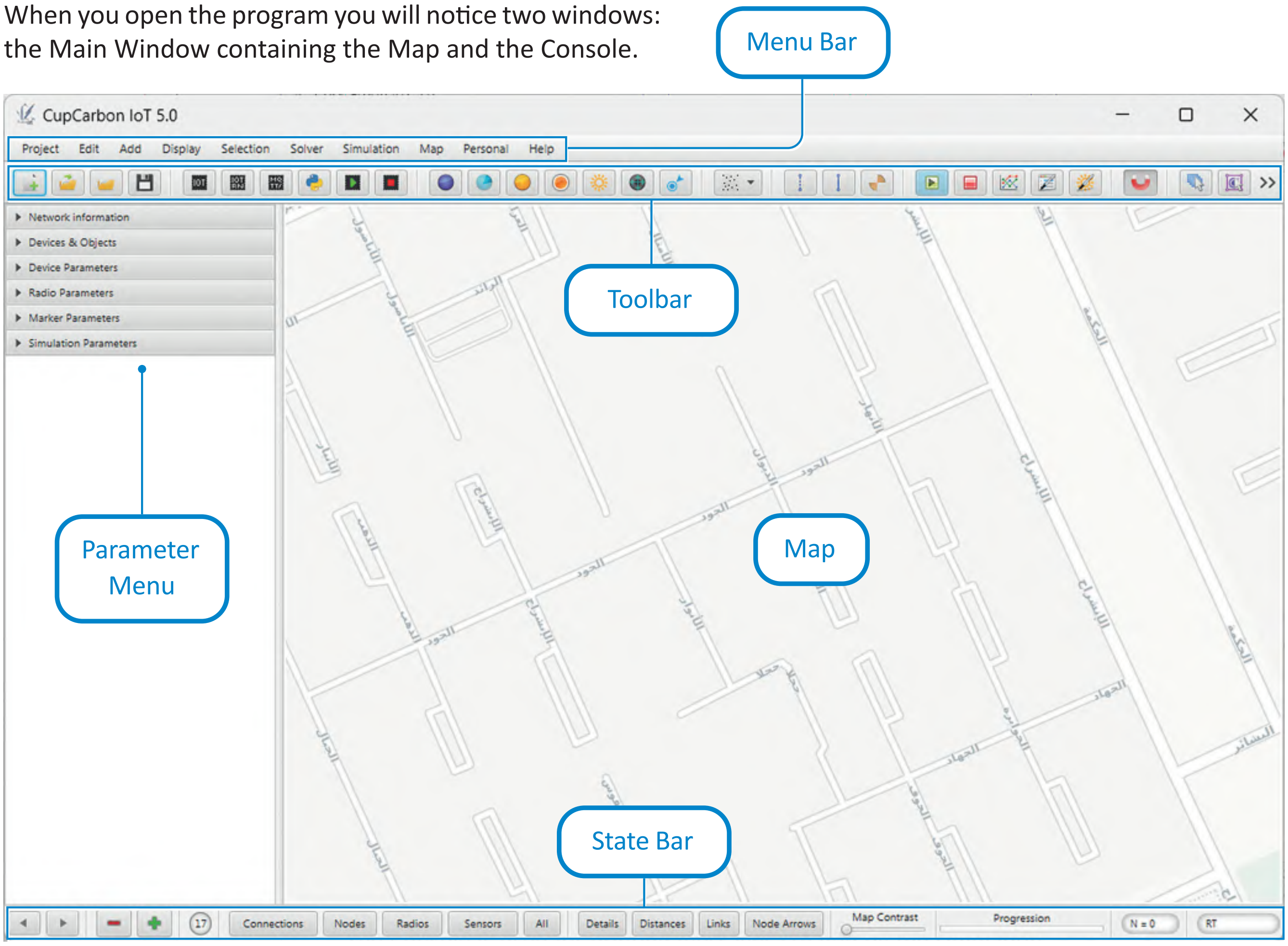
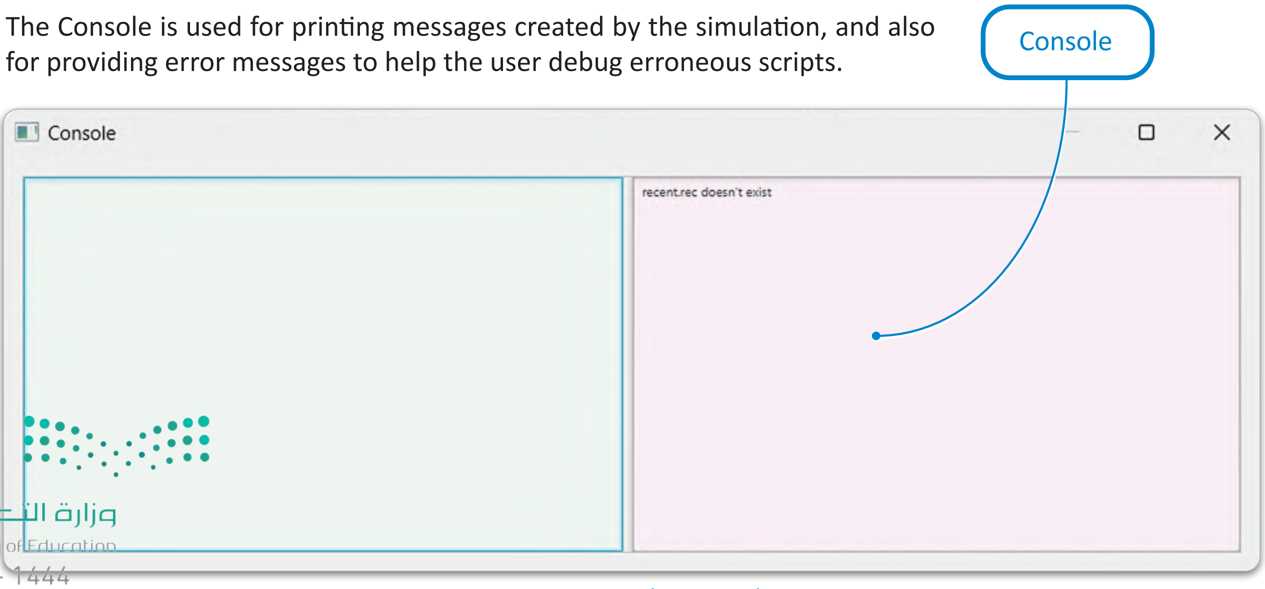


Figure 4.5: CupCarbon Main Window

The Console is used for printing messages created by the simulation, and also for providing error messages to help the user debug erroneous scripts.



```
recent.rec doesn't exist
```

Figure 4.6: CupCarbon Console

Getting started

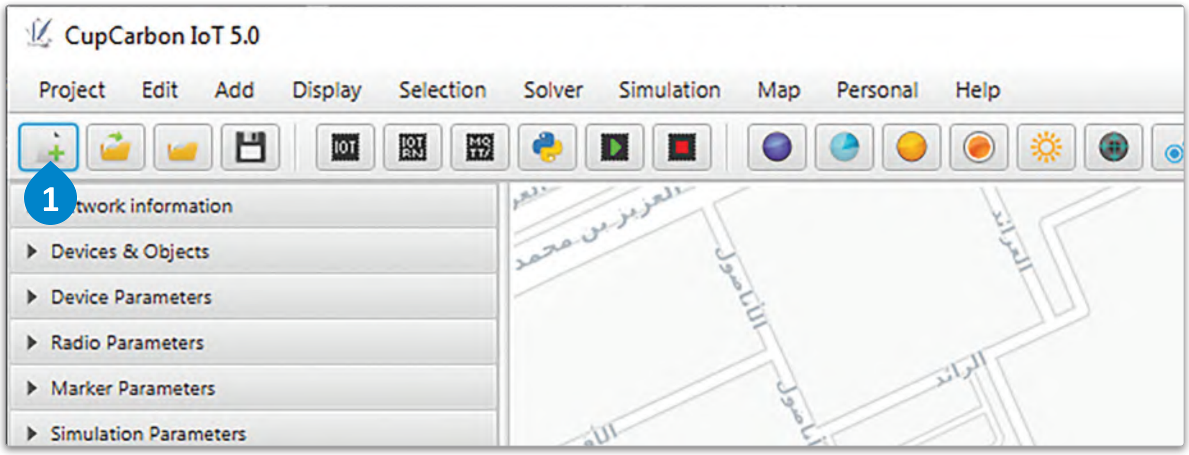
In this lesson, you are going to create a simple simulation of an IoT node printing messages in order to familiarize yourself with the CupCarbon environment.

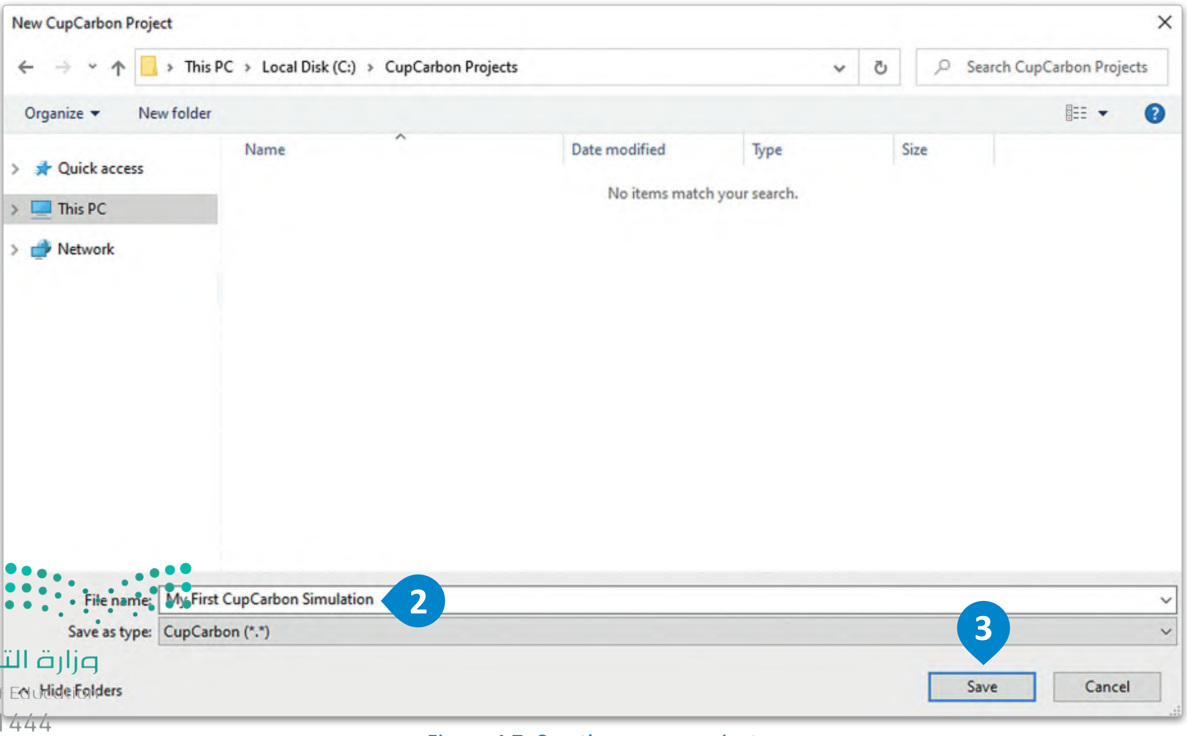
First, you must create a new project:

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





File name: My First CupCarbon Simulation

Save as type: CupCarbon (*.*)

Figure 4.7: 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**.

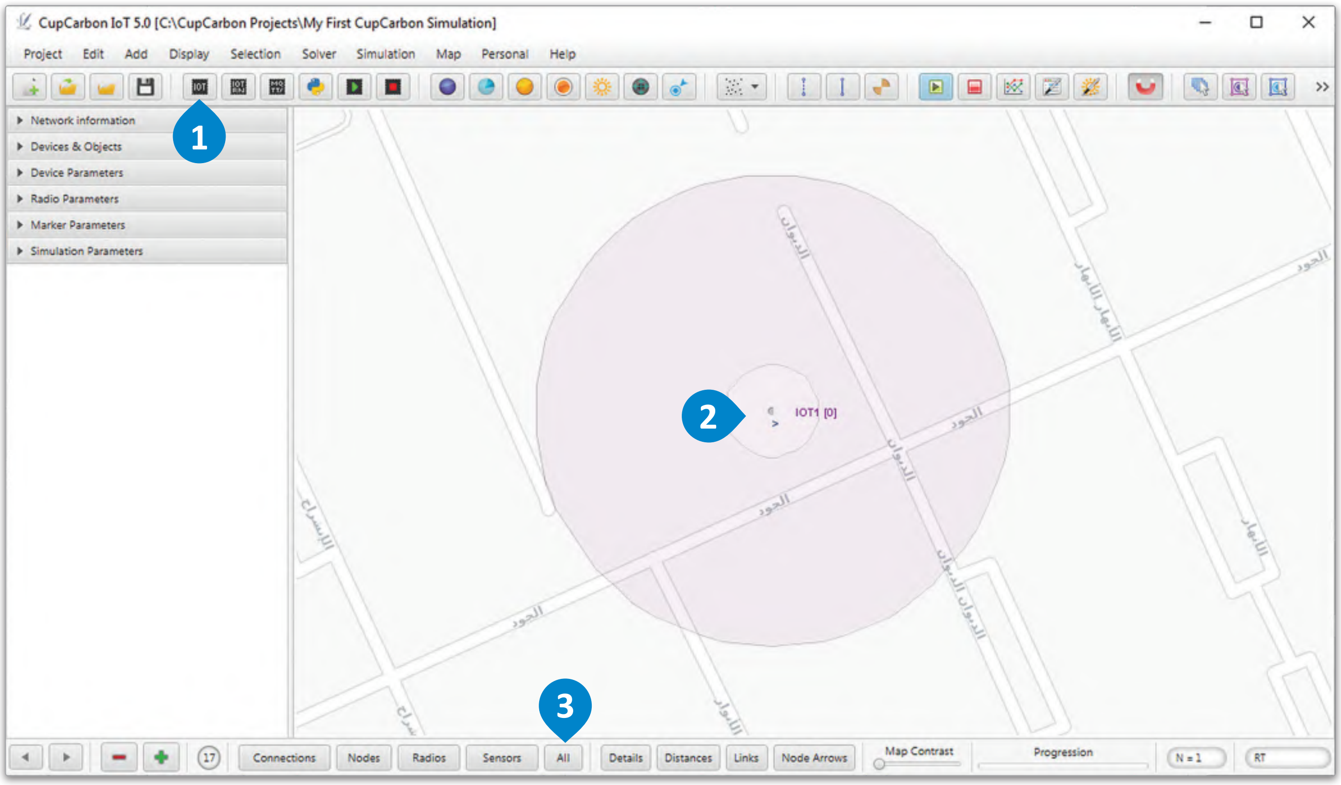

CupCarbon IoT 5.0 [C:\CupCarbon Projects\My First CupCarbon Simulation]
Project Edit Add Display Selection Solver Simulation Map Personal Help
Network information
Devices & Objects
Device Parameters
Radio Parameters
Marker Parameters
Simulation Parameters
IOT1 [0]
Connections Nodes Radios Sensors All Details Distances Links Node Arrows Map Contrast Progression N = 1 RT
1
2
3

Figure 4.8: 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:

```
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 by 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:

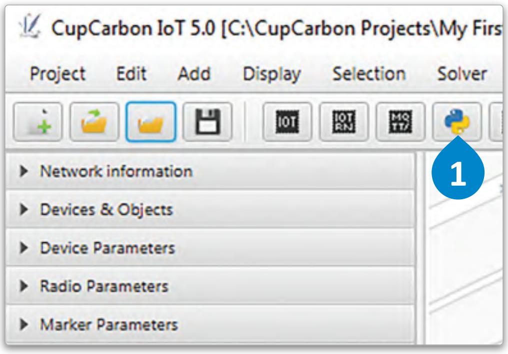
> Click on the **Python button** on the **Toolbar**. 1

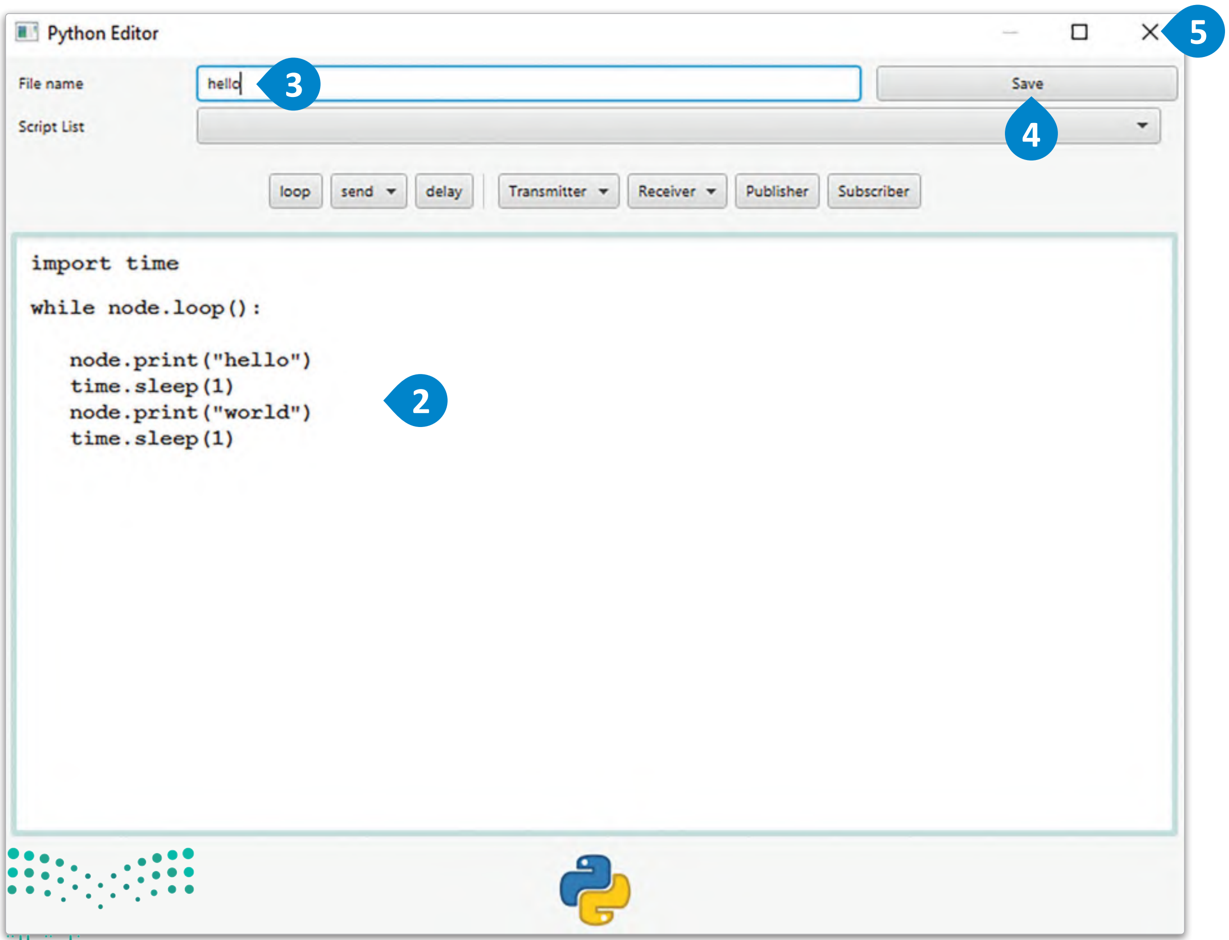
> Type the Python code into the field. 2

> Type **hello** in the **File name field**. 3

> Click **Save**. 4

> Close the Python Editor window. 5





File name: hello

```
import time
while node.loop():

   node.print("hello")
   time.sleep(1)
   node.print("world")
   time.sleep(1)
```

Figure 4.9: Python editor

To insert the script and run the simulation:

> Click on the node. 1

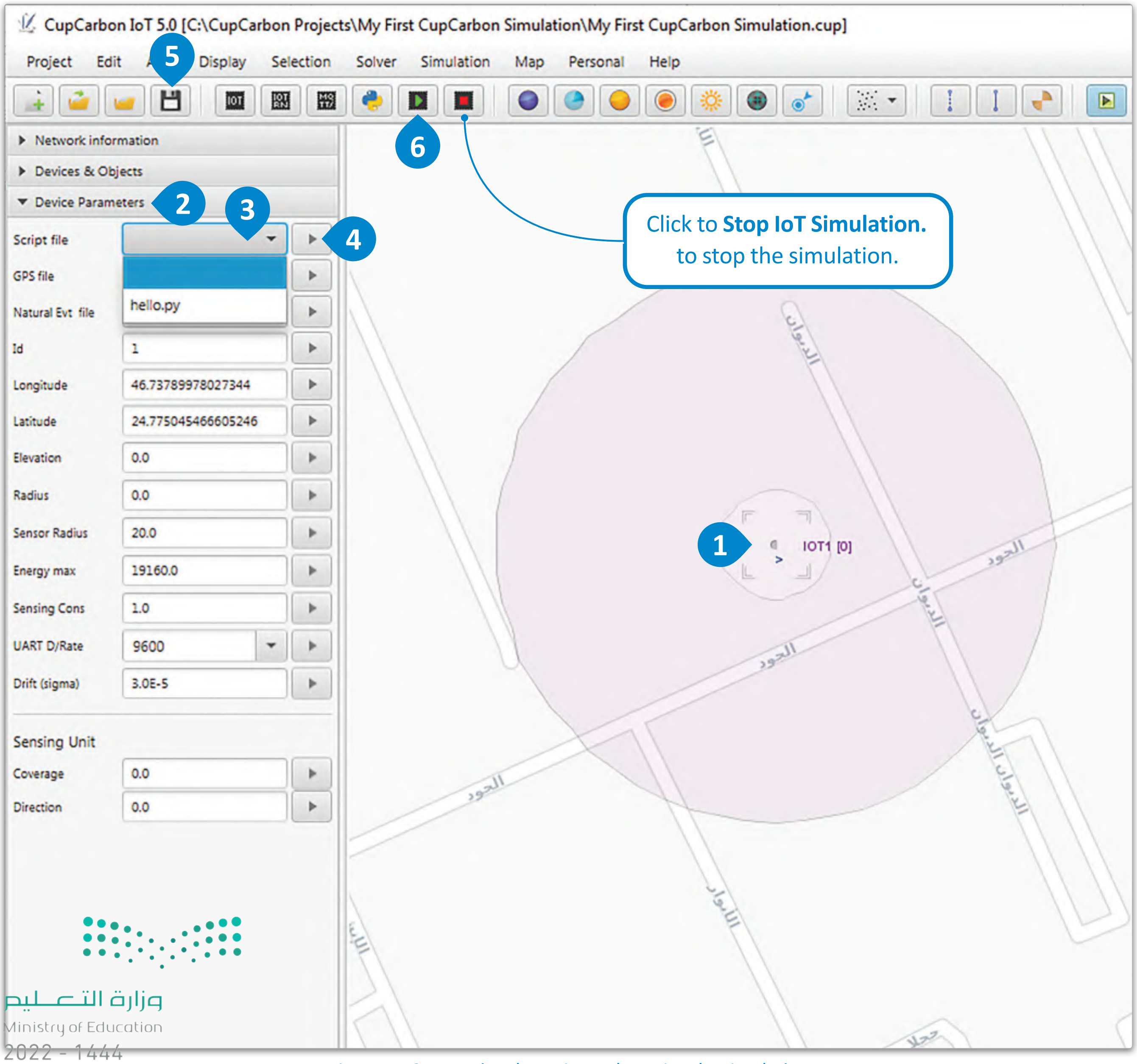
> Click on the **Device Parameters tab** on the **Parameter Menu**. 2

> 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

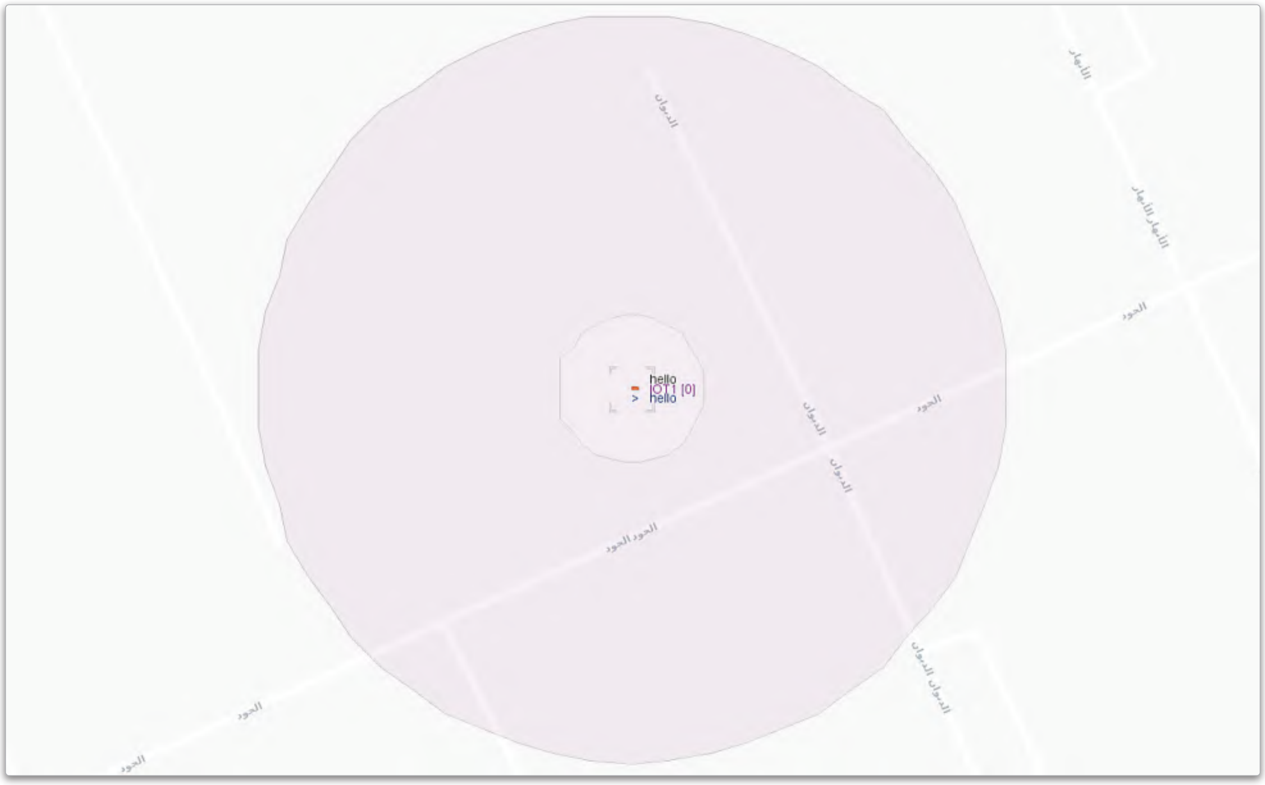


Id	1
Longitude	46.73789978027344
Latitude	24.775045466605246
Elevation	0.0
Radius	0.0
Sensor Radius	20.0
Energy max	19160.0
Sensing Cons	1.0
UART D/Rate	9600
Drift (sigma)	3.0E-5
Coverage	0.0
Direction	0.0

Click to **Stop IoT Simulation.** to stop the simulation.

Figure 4.10: Inserting the script and running the simulation

As expected, the node alternates between printing the strings "hello" and "world" for 1 second each.



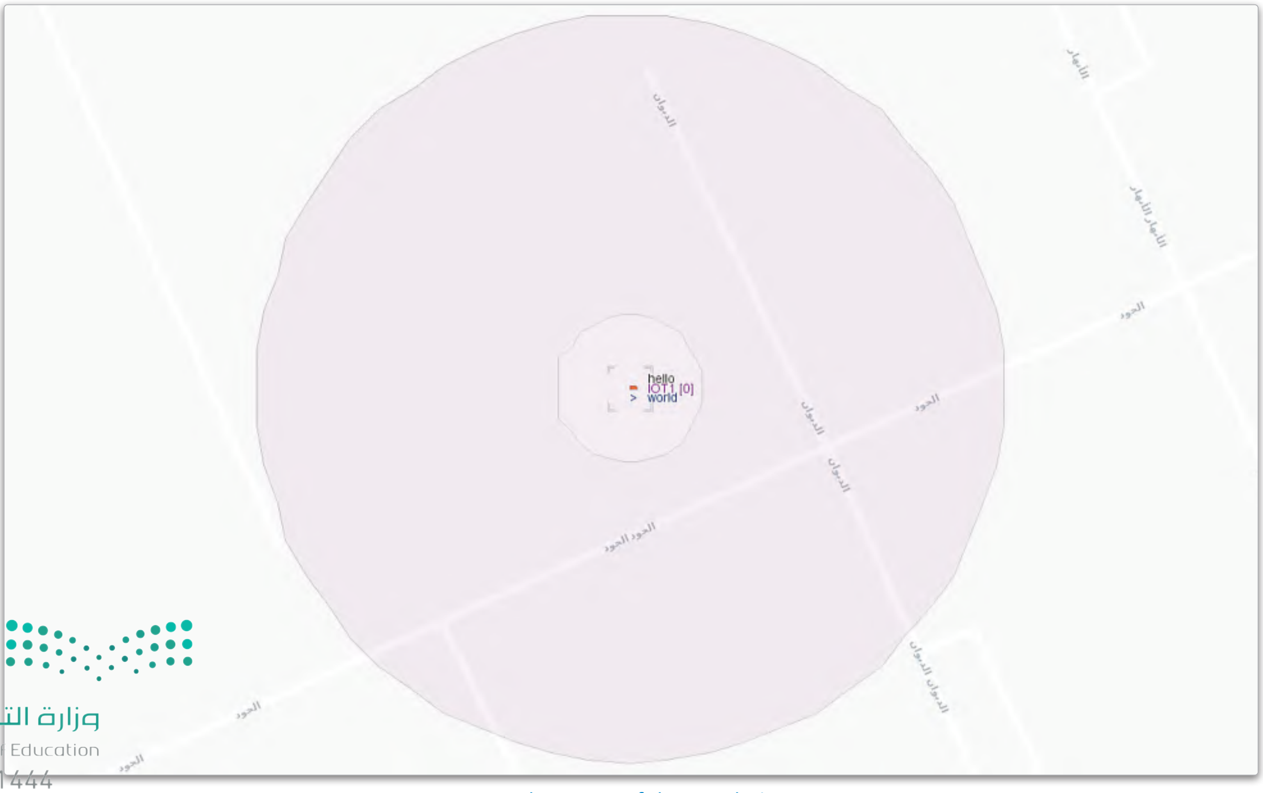


Figure 4.11: The states of the simulation

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.

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.

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

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.

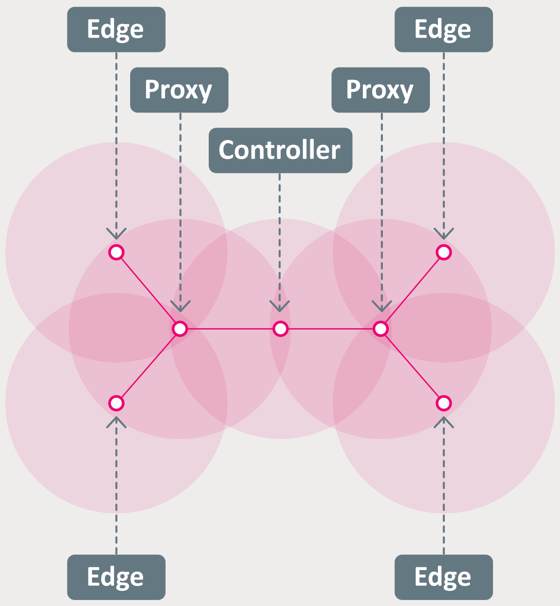


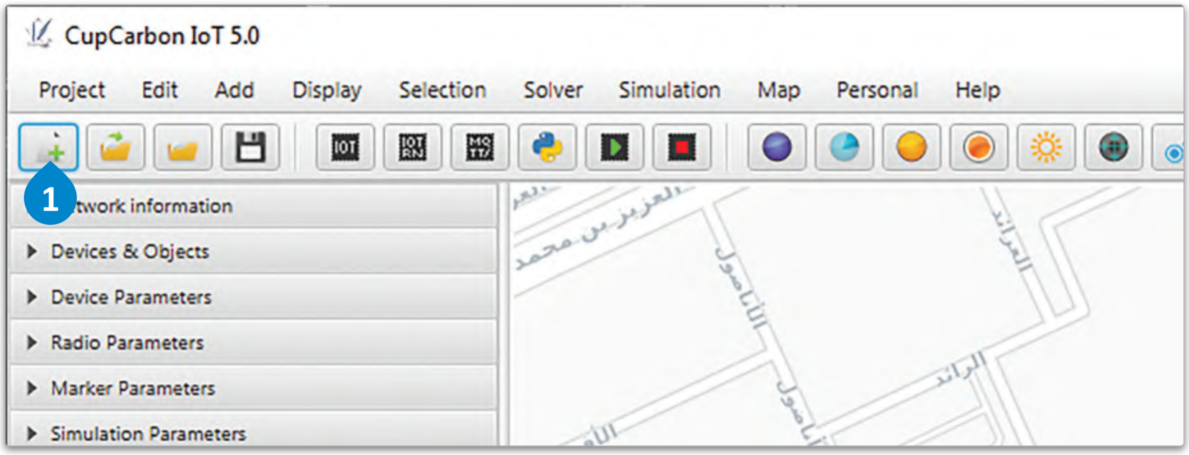
Figure 4.12: Nodes and their functions

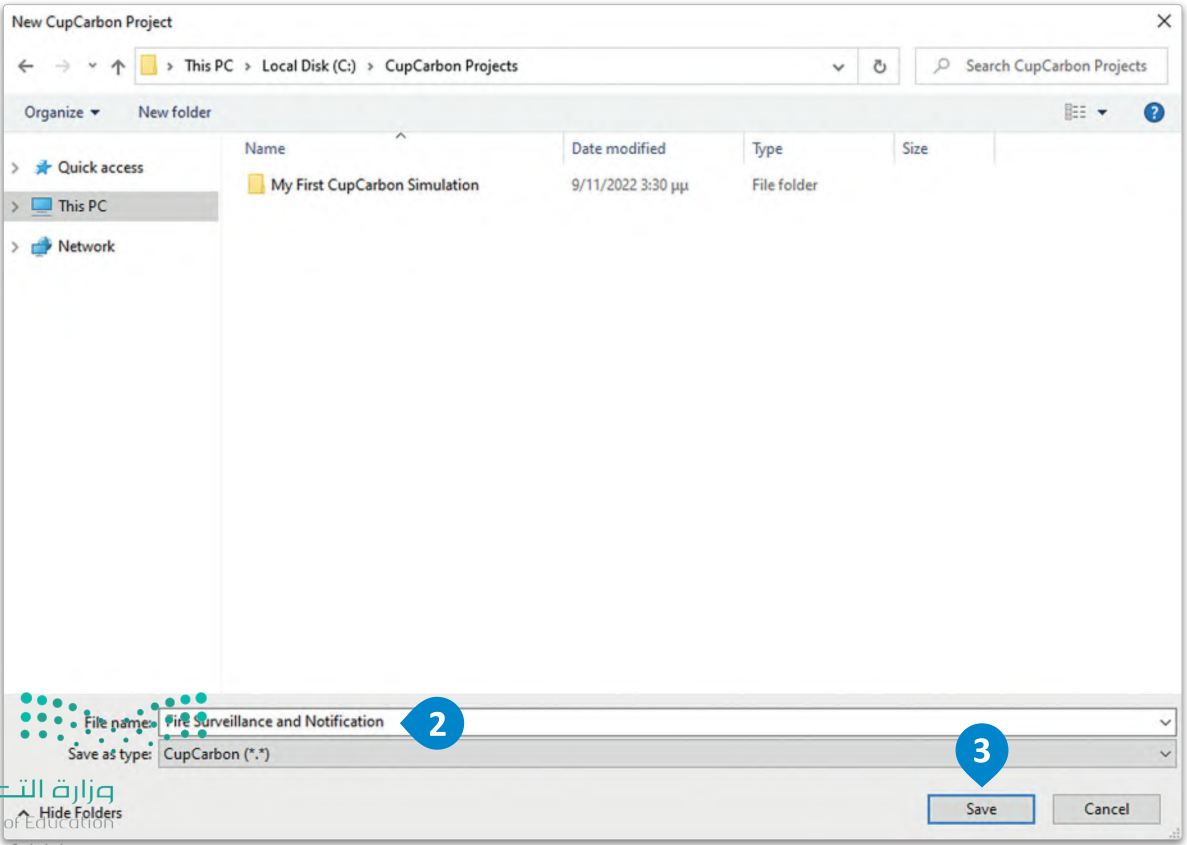
Let's start by creating a new project:

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





Name	Date modified	Type	Size
My First CupCarbon Simulation	9/11/2022 3:30 μμ	File folder	

File name: Fire Surveillance and Notification

Save as type: CupCarbon (*.*)

Figure 4.13: 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.

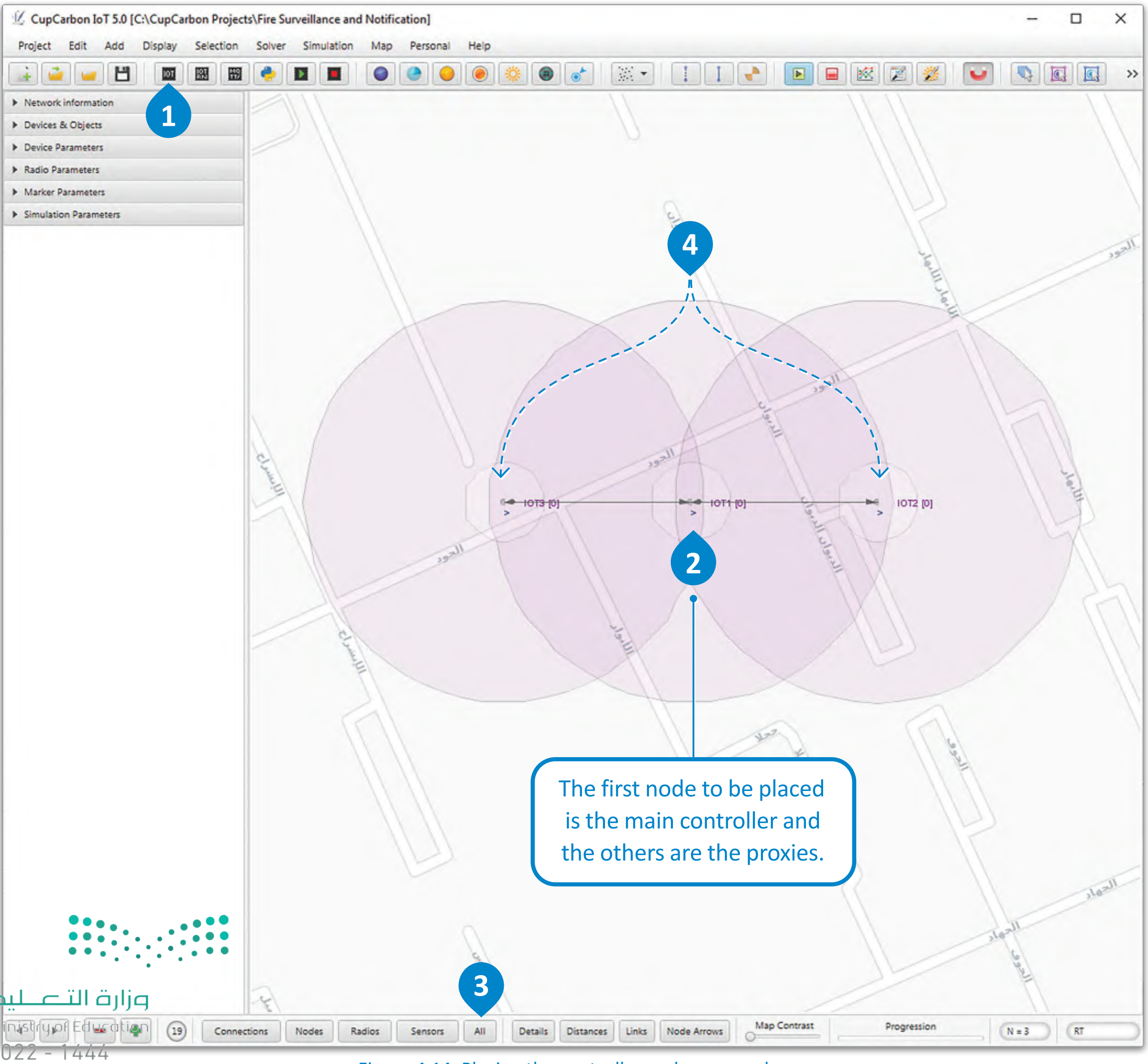


Figure 4.14: Placing the controller and proxy nodes

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**.

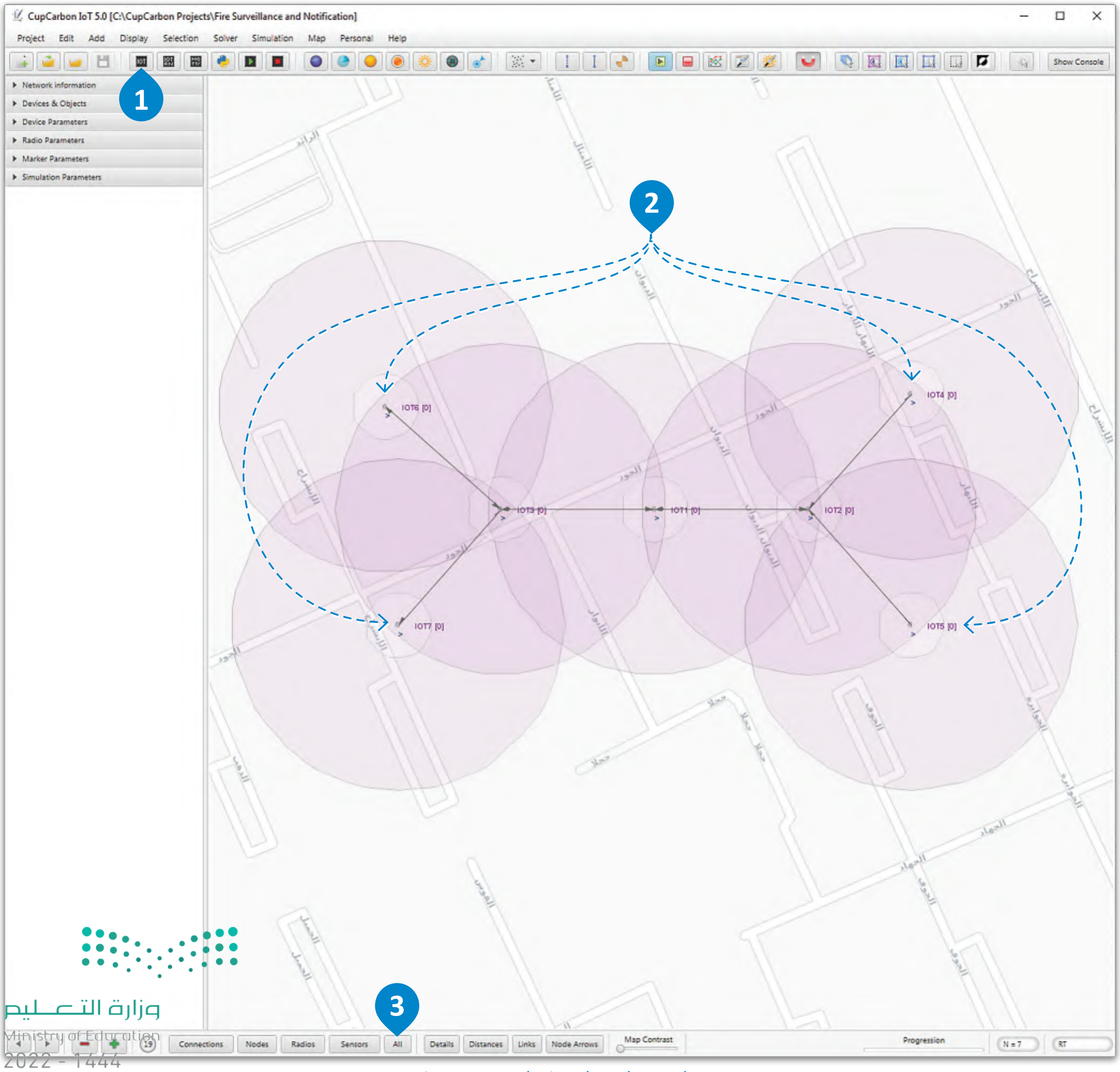


Figure 4.15: Placing the edge nodes

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 rangewhich 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, 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 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)
```

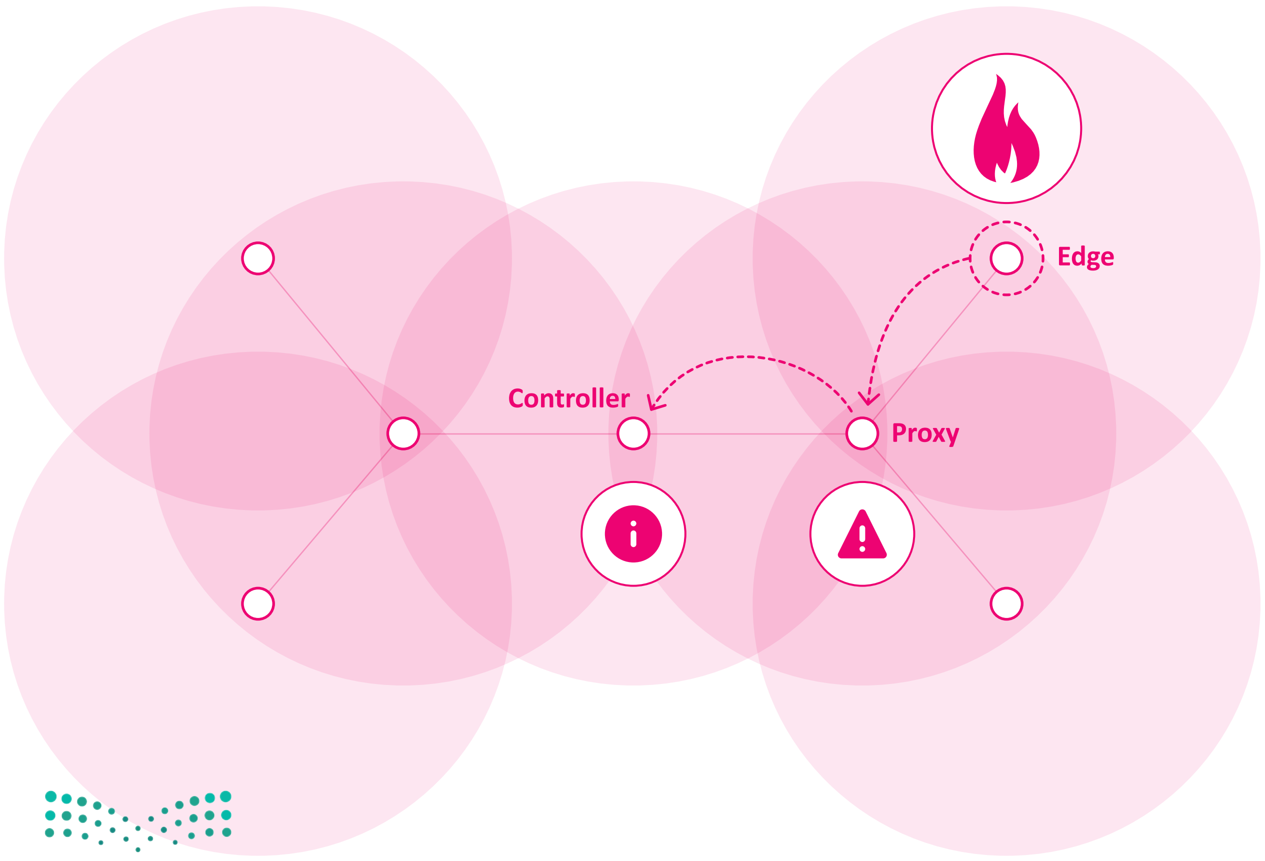


Figure 4.16: Complete network

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:

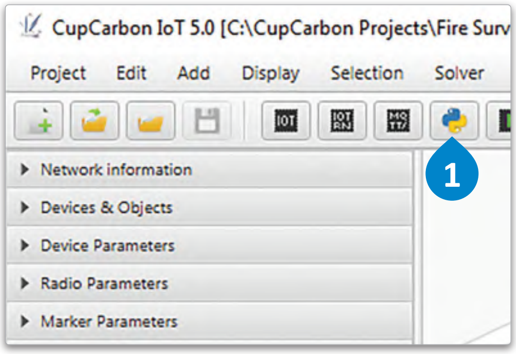
> Click on the **Python button** on the **Toolbar**. 1

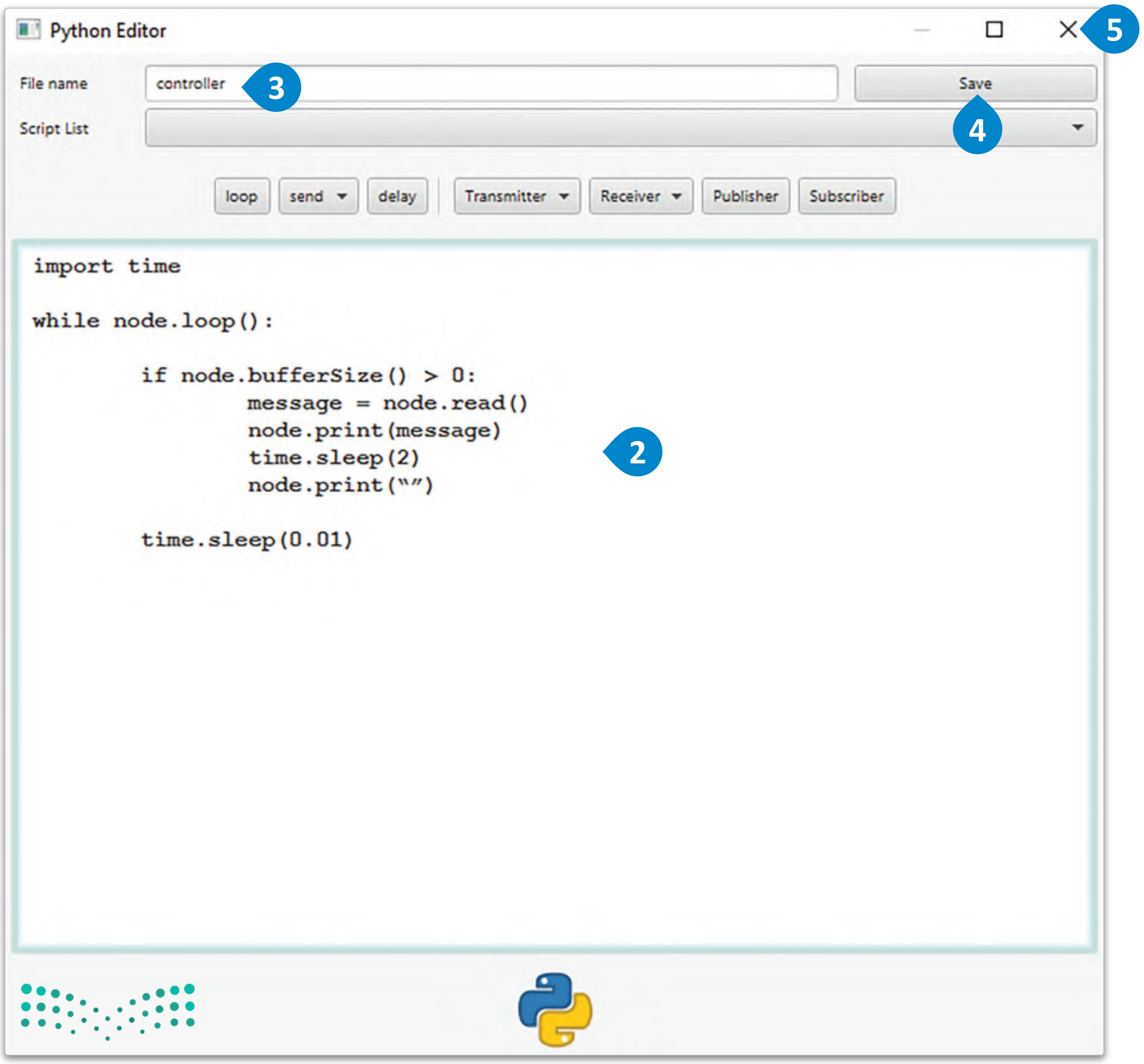
> Type the Python code into the field. 2

> Type **controller** in the **File name field**. 3

> Click **Save**. 4

> Close the Python Editor window. 5





```
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 4.17: Create the script

To insert the script:

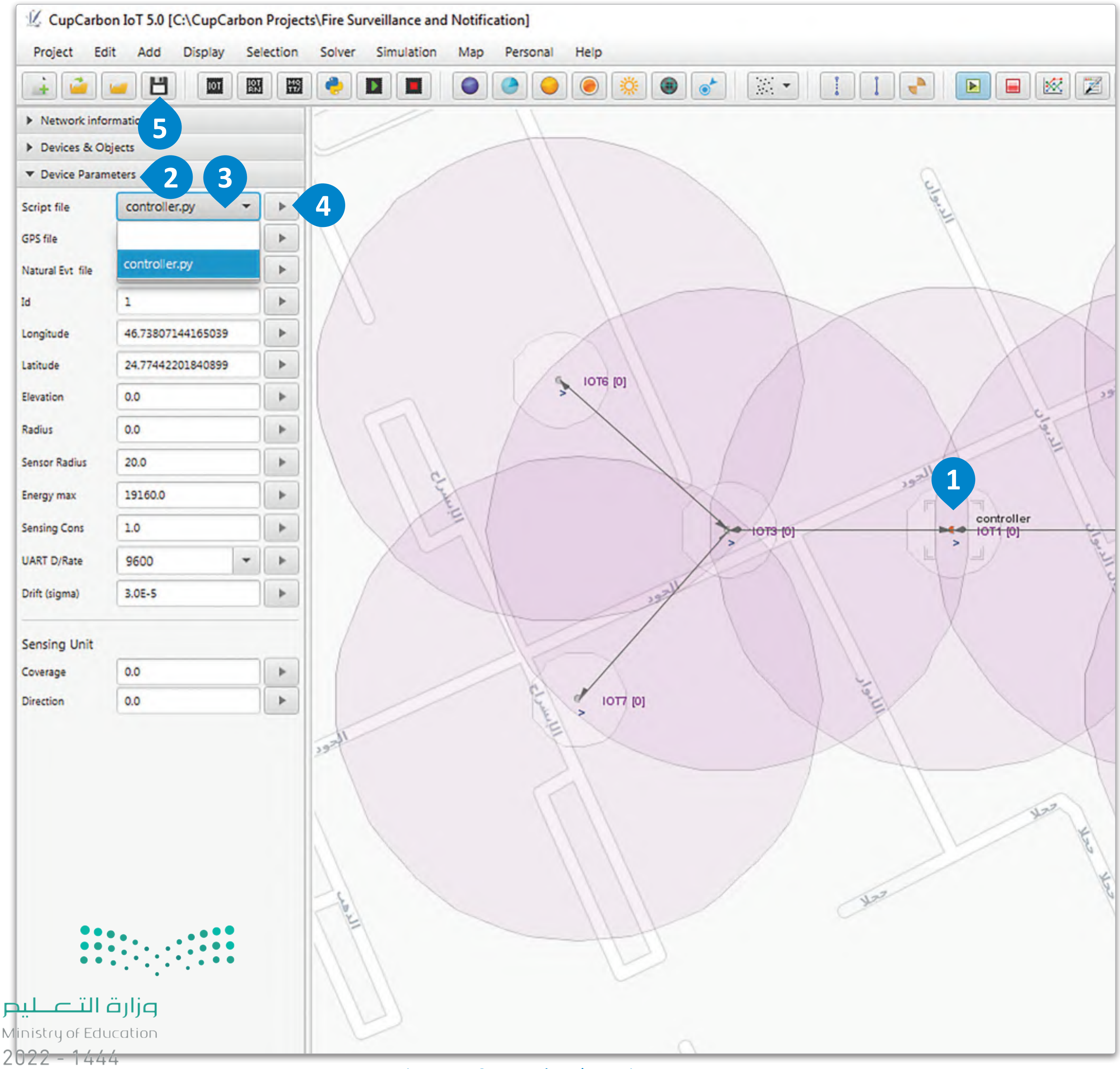
> Click on the node. 1

> 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


CupCarbon IoT 5.0 [C:\CupCarbon Projects\Fire Surveillance and Notification]
Project Edit Add Display Selection Solver Simulation Map Personal Help
Network information
Devices & Objects
Device Parameters
controller.py
IOT6 [0]
IOT3 [0]
IOT7 [0]
controller
IOT1 [0]
1
2
3
4
5

Script file	controller.py
GPS file	
Natural Evt file	
Id	1
Longitude	46.73807144165039
Latitude	24.77442201840899
Elevation	0.0
Radius	0.0
Sensor Radius	20.0
Energy max	19160.0
Sensing Cons	1.0
UART D/Rate	9600
Drift (sigma)	3.0E-5

Sensing Unit

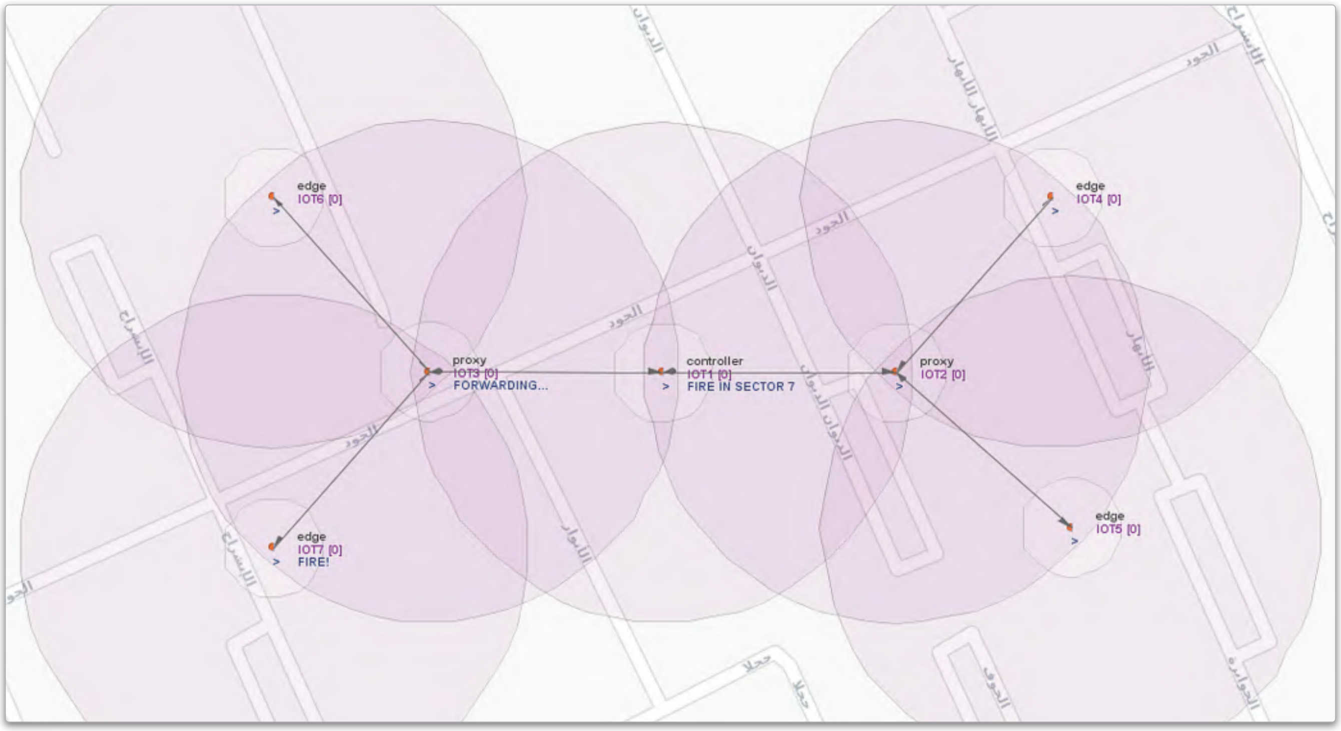
Coverage	0.0
Direction	0.0

Figure 4.18: Inserting the script

Likewise, create the other scripts, copy their codes and apply them to their corresponding nodes, so all nodes have 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.

Use self-explanatory script names such as edge.py and proxy.py.



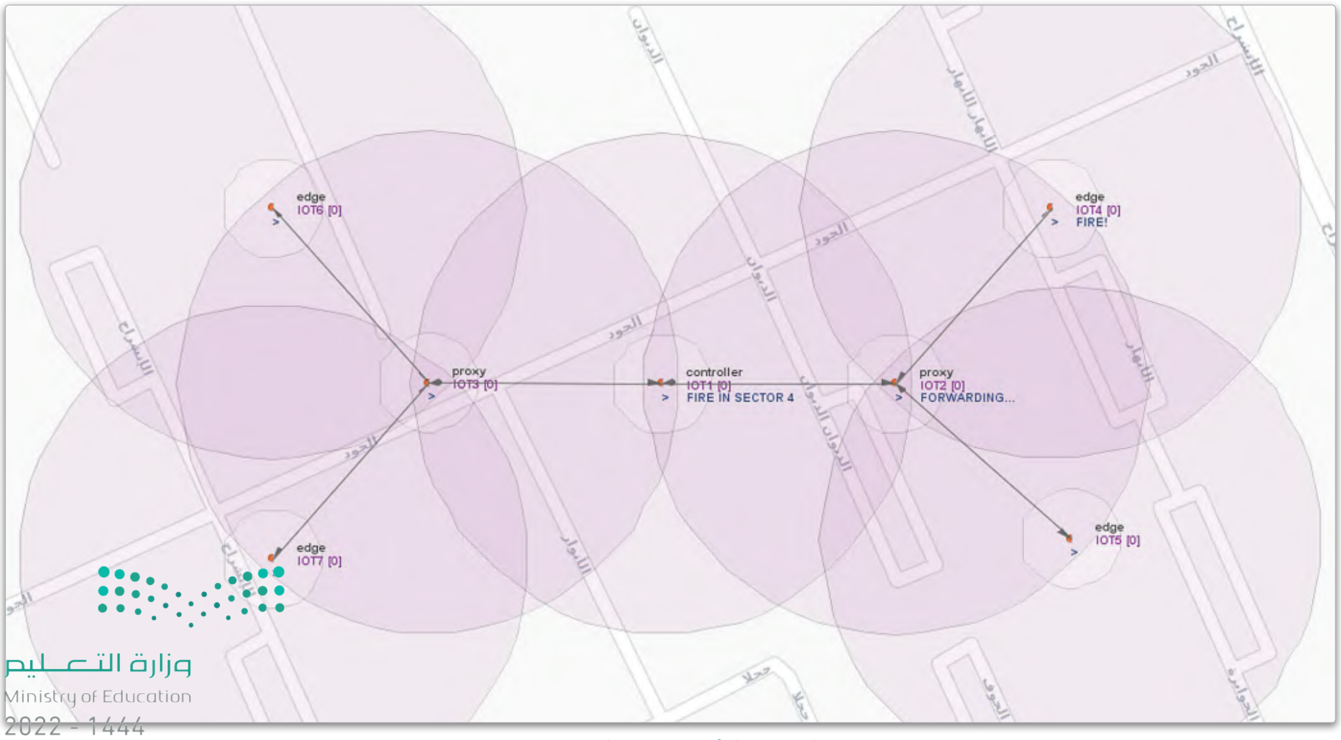


Figure 4.19: The states of the simulation

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.

Link to digital lesson

www.ien.edu.sa

Lesson 3

IoT and Automated Mobile Devices

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.

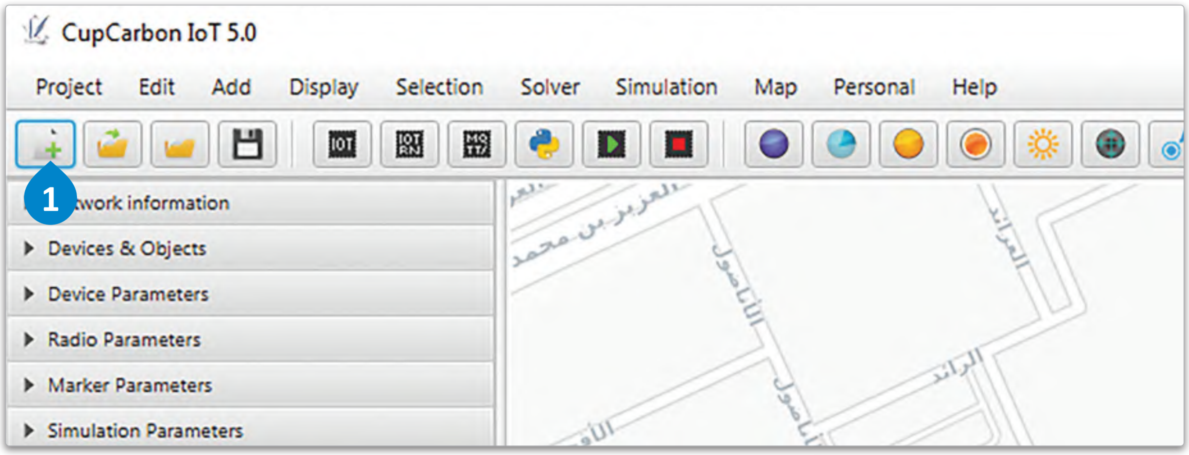
Figure 4.20: Automated Industry Vehicle

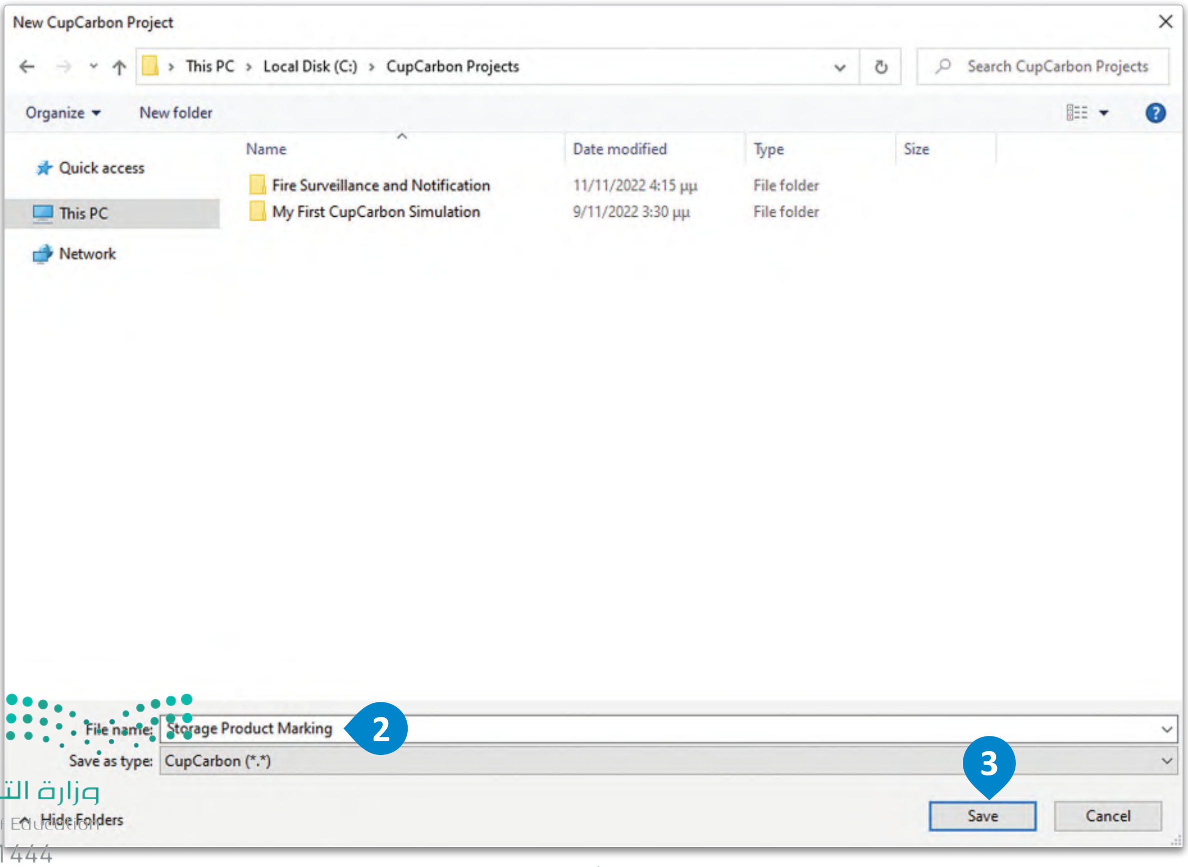
Let's start by creating a new project:

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





Name	Date modified	Type	Size
Fire Surveillance and Notification	11/11/2022 4:15 µµ	File folder	
My First CupCarbon Simulation	9/11/2022 3:30 µµ	File folder	

File name: Storage Product Marking

Save as type: CupCarbon (*.*)

Figure 4.21: 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

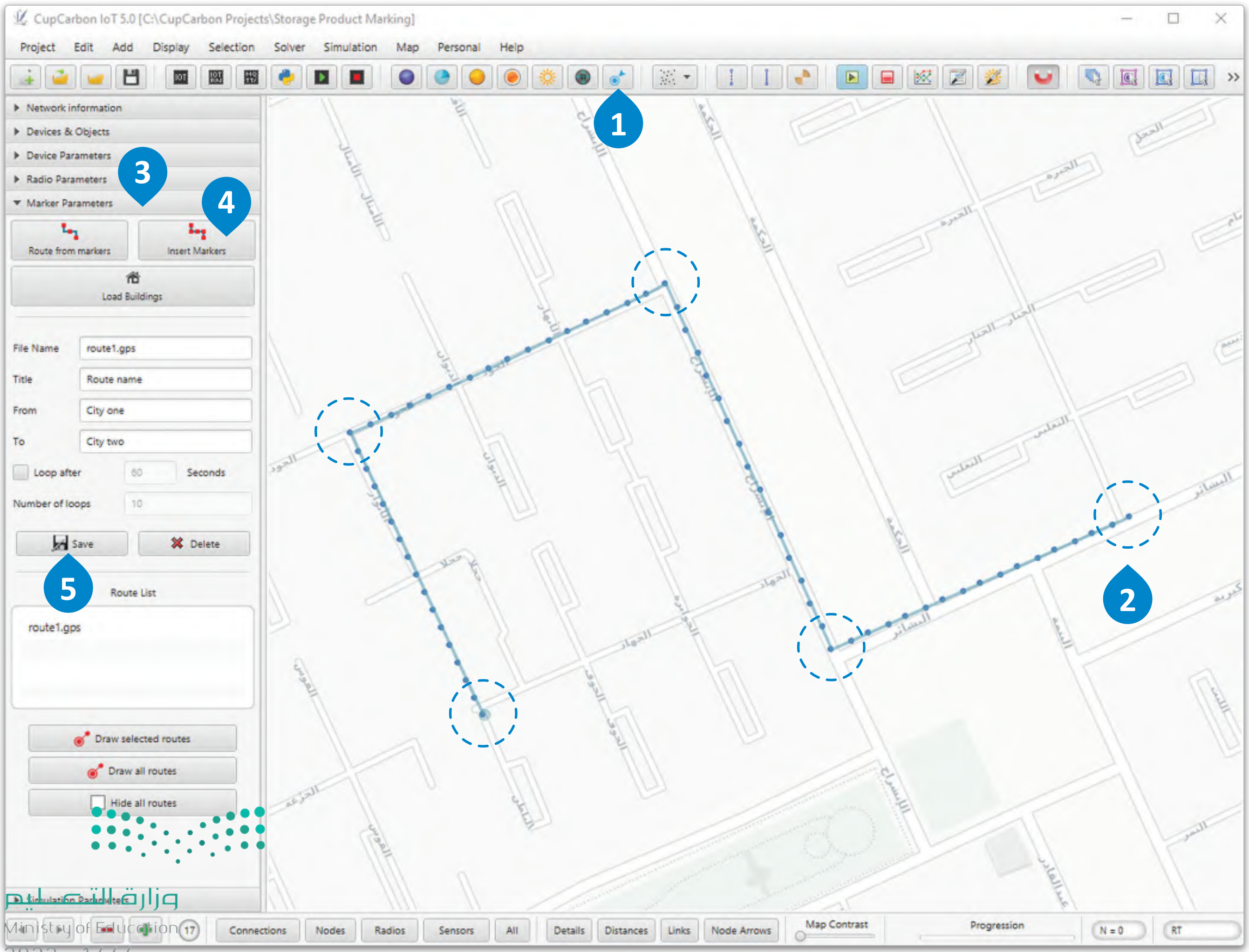
> Click on the map **5 times**, in a similar manner to the picture, **creating lines** along the aisles. 2

> 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



File Name: route1.gps
Title: Route name
From: City one
To: City two
Loop after: 60 Seconds
Number of loops: 10
route1.gps

Figure 4.22: 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

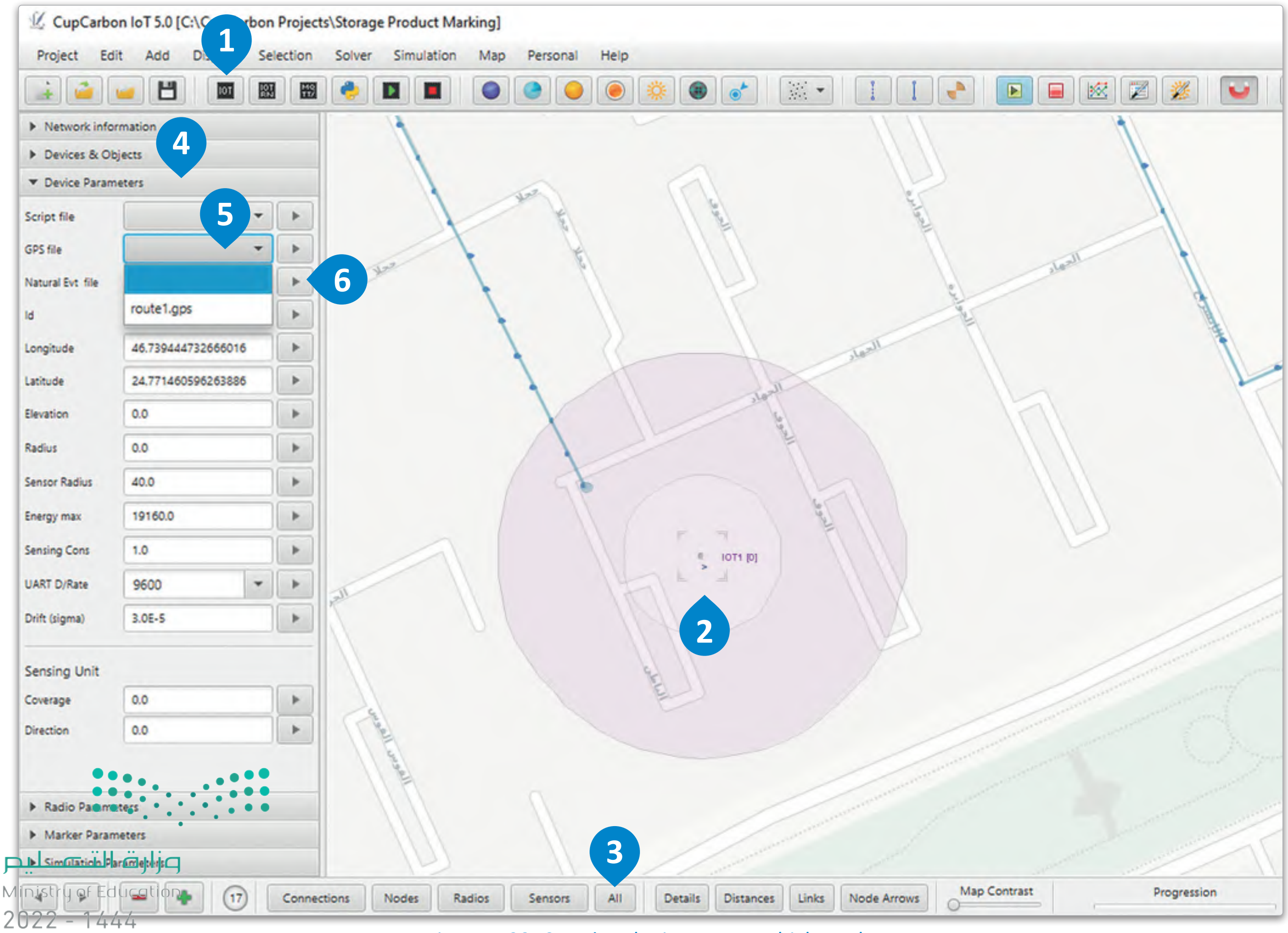
> Press **Esc**.

> 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 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



Longitude	46.739444732666016
Latitude	24.771460596263886
Elevation	0.0
Radius	0.0
Sensor Radius	40.0
Energy max	19160.0
Sensing Cons	1.0
UART D/Rate	9600
Drift (sigma)	3.0E-5
Coverage	0.0
Direction	0.0

Figure 4.23: 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 radius includes at least 1 marker from the route. 2

> Click on the **All button** from the **State bar**. 3

> Press **Esc**.

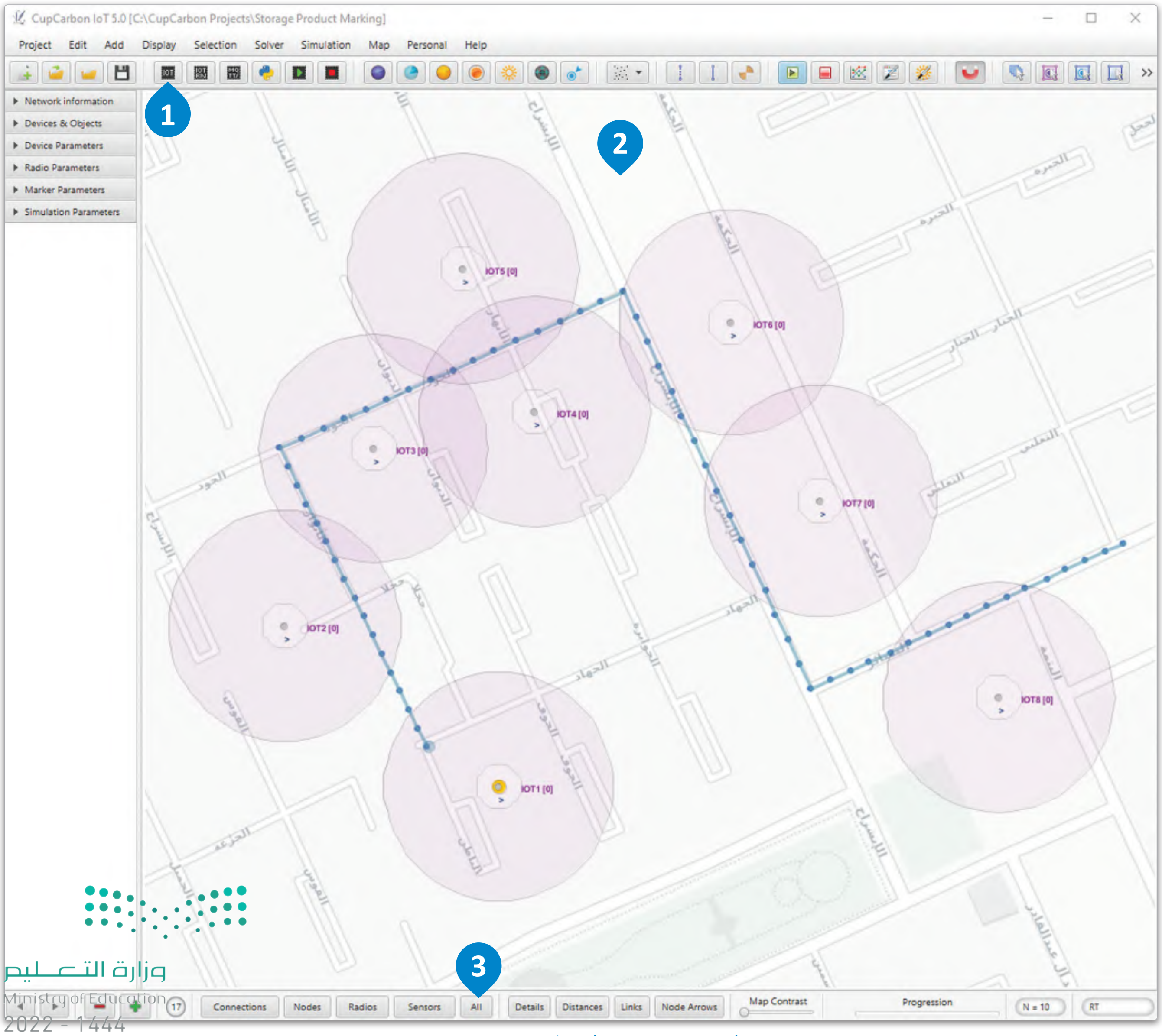


Figure 4.24: Creating the container nodes

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**.

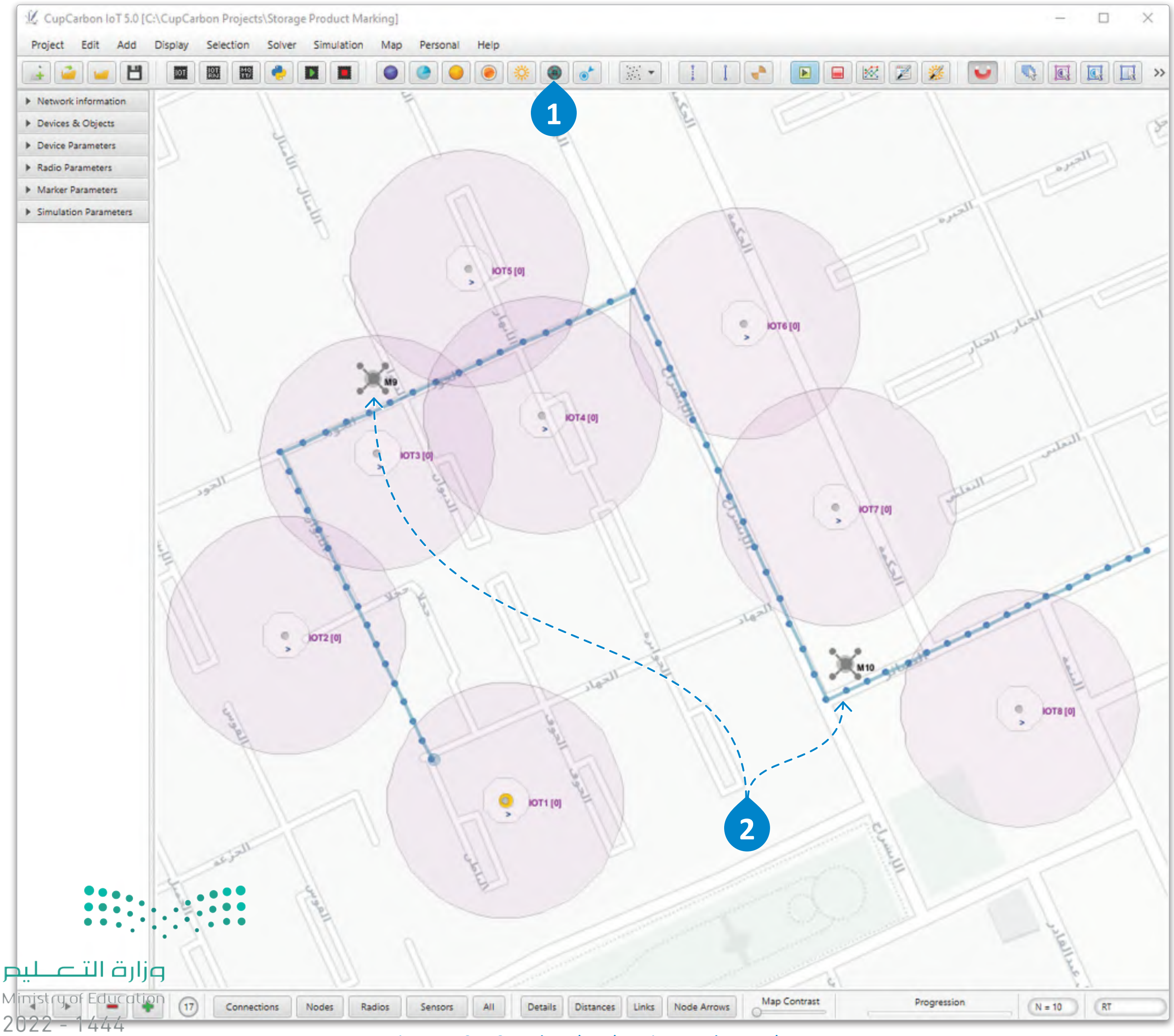

CupCarbon IoT 5.0 [C:\CupCarbon Projects\Storage Product Marking]
Project Edit Add Display Selection Solver Simulation Map Personal Help
Network information
Devices & Objects
Device Parameters
Radio Parameters
Marker Parameters
Simulation Parameters
1
2
IOT1 [0]
IOT2 [0]
IOT3 [0]
IOT4 [0]
IOT5 [0]
IOT6 [0]
IOT7 [0]
IOT8 [0]
M9
M10
Connections Nodes Radios Sensors All Details Distances Links Node Arrows Map Contrast Progression N = 10 RT

Figure 4.25: 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 nonconsumables is the string they send. In the nonconsumables' 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 inpector 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 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:

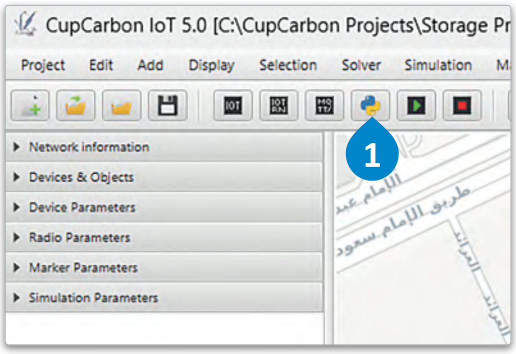
> Click on the **Python button** on the **Toolbar**. 1

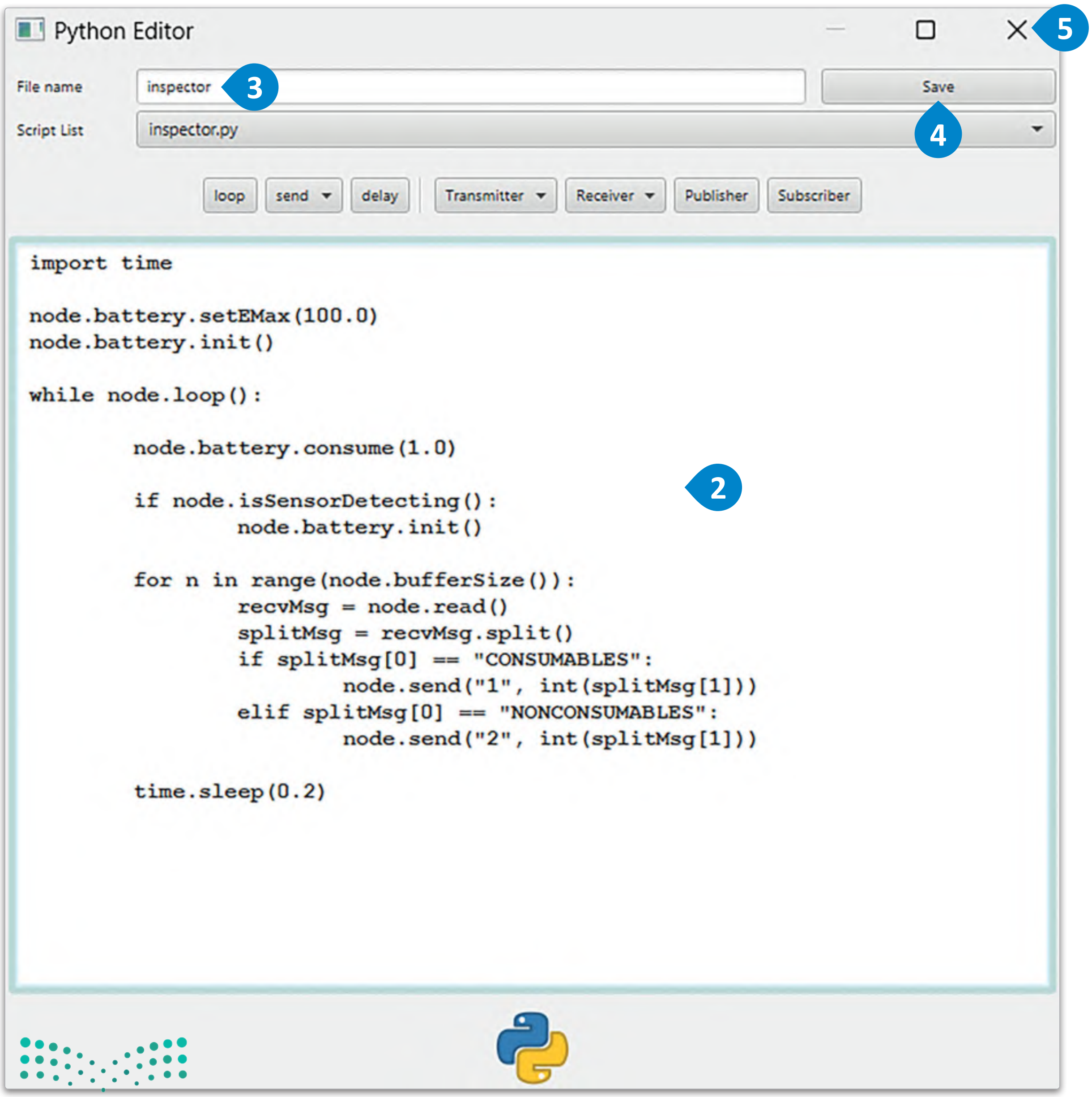
> Type the Python code into the field. 2

> Type **inspector** in the **File name field**. 3

> Click **Save**. 4

> Close the Python Editor window. 5





```
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)
```

Figure 4.26: Create the script

To insert the script:

> Click on the inspector vehicle node. 1

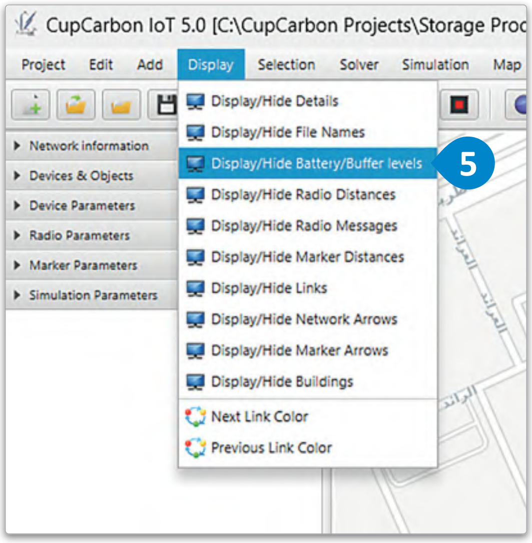
> Click on the **Device Parameters tab** on the **Parameter Menu**. 2

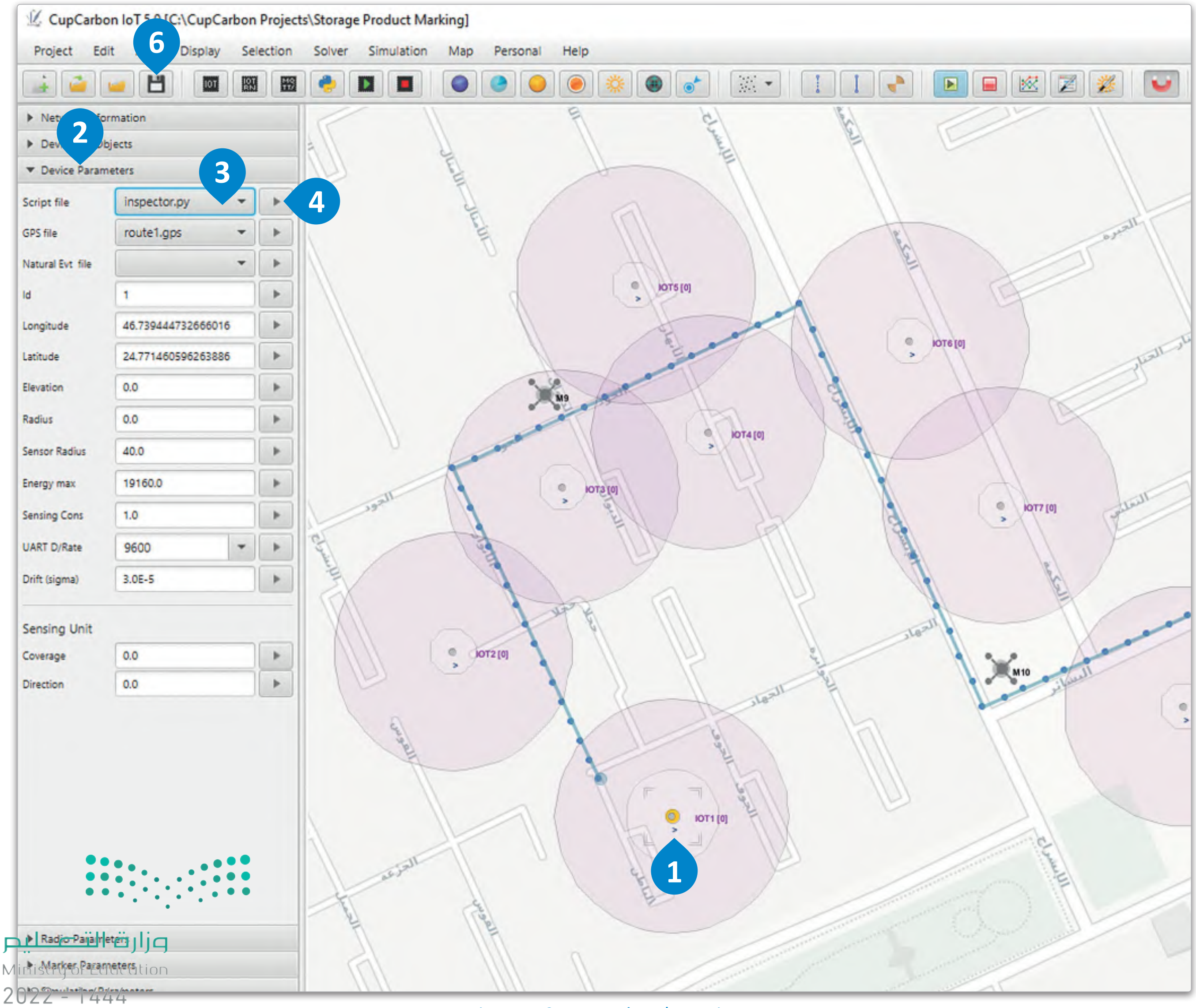
> Click on the **Script file** box. 3

> 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





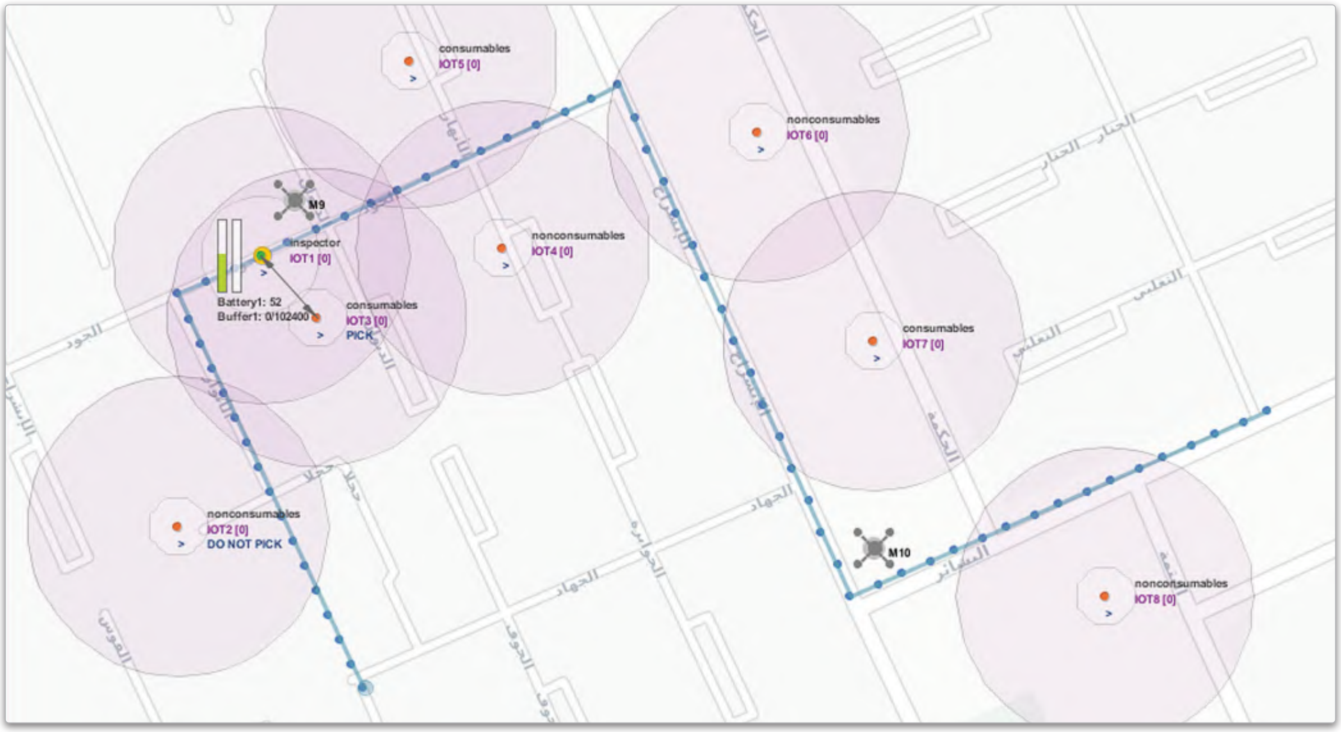
Script file: inspector.py
GPS file: route1.gps
Natural Evt file:
Id: 1
Longitude: 46.739444732666016
Latitude: 24.771460596263886
Elevation: 0.0
Radius: 0.0
Sensor Radius: 40.0
Energy max: 19160.0
Sensing Cons: 1.0
UART D/Rate: 9600
Drift (sigma): 3.0E-5

Sensing Unit
Coverage: 0.0
Direction: 0.0

Figure 4.27: Inserting the script

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.



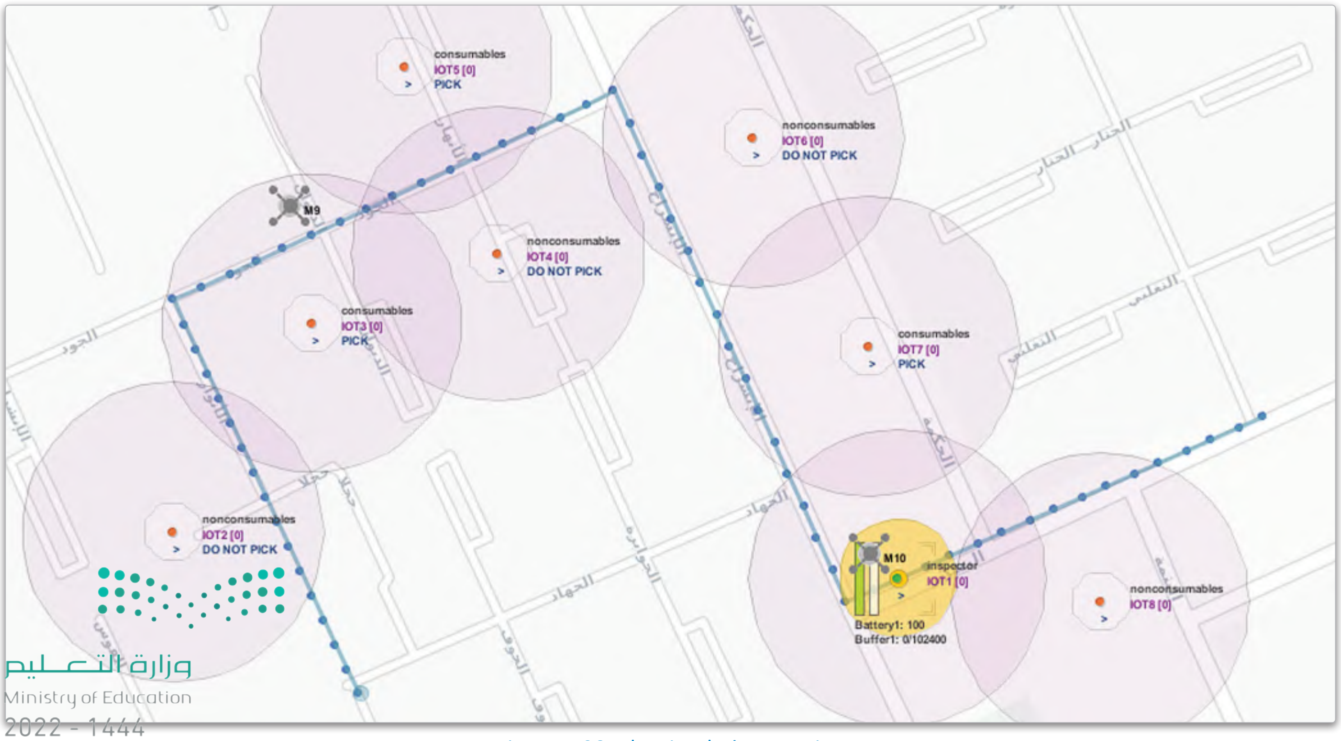


Figure 4.28: The simulation running

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.

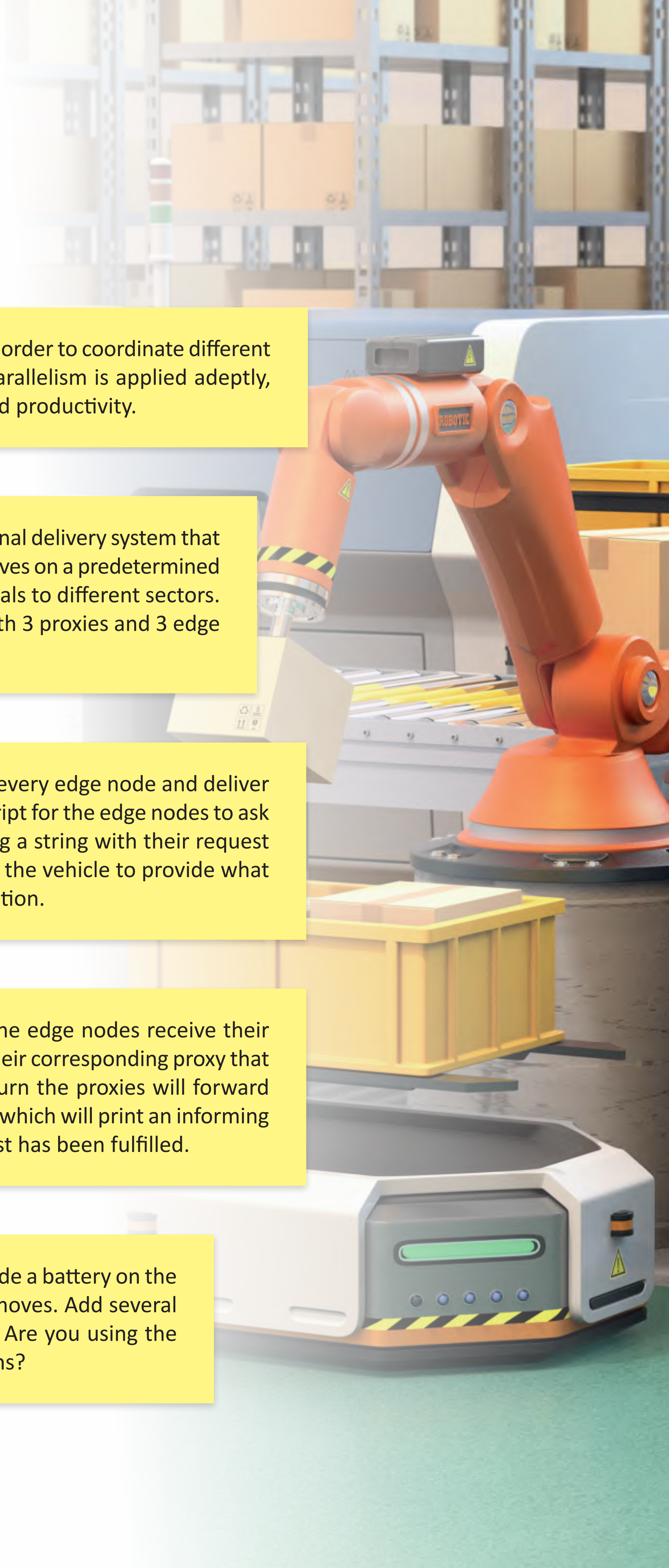
2 Identify whether your project is using the minimal number of charging stations. Try removing a station and relocating the others to test your hypothesis.

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	Edge Computing	Modbus
Data-Driven Manufacturing	Industrial Automation and Control Systems	Operational Technology
Digitization	Key Performance Indicators	Smart Industry

Notes

وزارة التعليم
Ministry of Education
2022 - 1444