

| **CODEDU’s Teachers’ Training Curriculum** | | | | | | |
| --- | --- | --- | --- | --- | --- | --- |
| **Section 3**  **Theoretical introduction to Arduino** | | | | | | |
| **Subject:**  **Theoretical introduction to Arduino** | **Duration (in hours):** | | | | **10 hours for Module 3.1**  **6 hours for Module 3.2** | |
| **Target audience:**  **Upper-primary and Secondary School Students** | | | | | | |
| **Training methodology: Online and Flipped Learning,  Demonstration** | | | | | | |
| **Level (and cycle, if applicable) of the learning experience:** | | | | | | |
| **Assessment method: Multiple Questions, Project** | | **Form of participation in the learning activity: Physical or online** | | | | |
| **Expected Learning outcomes:**  • Understanding basics  • Programming skills  • Hands-on project experience  • Problem-solving and creativity | | * Team collaboration * Arduino Basic concepts * Inclusive learning * Coding basics | | | | |
| **Prerequisites needed to enrol in the learning activities (if needed): N/A** | | | | | | |
| **Supervision and identity verification during an assessment:** | | | |  | |  |
| • Unsupervised with no identity verification. | | |  |  | |  |
| • Supervised with no identity verification. | | |  |  | |  |
| • Supervised online or onsite with identity verification. | | | X |  | |  |
| **Further information: The course introduces educators to the fundamentals of Arduino, focusing on its hardware, software, and applications in STEM education. Participants will learn to write, upload, and debug Arduino code, using sensors, actuators, and control structures to create interactive projects like smart home systems. Emphasizing hands-on, inclusive, and project-based learning, the course fosters problem-solving, creativity, and teamwork among participants.**    ***Key Words: Arduino programming, Microcontroller, Sensors and actuators, Input/output operations, STEM education, Hands-on projects, Problem-solving, Inclusive learning, Smart home systems, Troubleshooting*** | | | | | | |

| **Module 3.1** |
| --- |
| **Getting Started with Arduino** |
| Arduino is an open-source platform designed to make it easier for creators, from hobbyists to engineers, to prototype electronic projects. At its core, Arduino combines both hardware and software, offering a simple yet powerful environment to develop interactive devices. Whether you're interested in building a basic LED blink circuit or designing a sophisticated IoT system, Arduino provides the necessary tools and components to bring your ideas to life.  This module will guide you through the basics of Arduino hardware and software, focusing on essential concepts such as digital and analogue signals, input and output operations, and key components like sensors and actuators. You will learn how to set up the Arduino IDE, write your first Arduino sketch, and understand the structure of a typical Arduino program. By the end of this module, you'll be ready to start experimenting with your own projects and exploring the vast world of possibilities that Arduino offers. 3.1.1. What is Arduino? Arduino is an open-source electronics platform that combines easy-to-use hardware and software, designed to make it accessible for anyone interested in creating interactive projects.  Arduino Uno Rev3 — Arduino Online Shop  *Figure 1: Arduino UNO Rev3 (Source: arduino.cc)*  It consists of small circuit boards, called microcontrollers, that can be programmed to control electronic components like sensors, motors, LEDs, and other devices. With Arduino, users can build projects that sense their environment, process inputs, and trigger specific actions.  The programming is done using the Arduino IDE, a simple development environment where you write code in a language similar to C++. Arduino is especially popular in education because it allows students to explore concepts in science, technology, engineering, and mathematics (STEM) through hands-on experimentation.  As teachers you can use Arduino to teach programming, electronics, and problem-solving, while fostering creativity as students design projects such as robots, weather stations, or automated systems. Its modular and flexible design makes it ideal for learners of all ages and experience levels, encouraging them to engage with technology in an approachable and fun way. 3.1.2. Basics of Arduino hardware Arduino hardware forms the backbone of its open-source platform, designed to simplify electronics prototyping and programming. Here are the basics:  Inside the Arduino UNO Board: A Comprehensive Tour  *Figure 2: Arduino UNO Board (Source: hackerearth.com)*   1. **Microcontroller**: At the heart of every Arduino board is a microcontroller, such as the ATmega328P (used in Arduino UNO). This small integrated circuit acts as the "brain" of the board, executing the instructions written in the Arduino IDE. 2. **Power Supply (Power Port)**: Arduino boards can be powered via USB from a computer or an external power supply (e.g., a battery or AC adapter). The boards typically operate at 5V or 3.3V depending on the model. 3. **Pins**:    * **Digital Pins**: Used for digital input or output (e.g., turning an LED on or off).    * **Analog Pins**: Designed to read analogue signals from sensors, like temperature or light sensors, which are then converted to digital data.    * **PWM Pins**: Specific digital pins can simulate analogue outputs using Pulse Width Modulation, useful for controlling things like motor speed or LED brightness.    * **Power Pins**: Supply voltage (3.3V or 5V) and ground (GND) to external components. 4. **Communication Interfaces**:    * **USB Port**: Allows uploading of code and communication with a computer.    * **Serial Pins (TX/RX)**: Enable communication with other devices or modules, such as Bluetooth or GPS.    * Some boards also support additional protocols like I2C and SPI for advanced communication. 5. **Reset Button**: Restarts the program running on the microcontroller, useful for troubleshooting. 6. **Voltage Regulators**: Ensure stable voltage supply, protecting the board from power fluctuations. 7. **Built-in LEDs**: Most Arduino boards have at least one built-in LED (connected to pin 13 on many models) for basic output testing. 8. **Form Factor and Shields**: Arduino boards come in different sizes and shapes, from the compact Nano to the versatile Mega. Shields are additional boards that stack on top of Arduino boards to extend their functionality, such as adding motor control, Wi-Fi, or GPS.   Arduino's hardware design is open-source, meaning its schematics are freely available. This allows for customization and fosters a broad ecosystem of compatible components and clones. Boards like the Arduino UNO are great for beginners, while others like the Arduino Mega or Portenta are suited for more complex projects. 3.1.3. Basics of Arduino software Arduino software consists primarily of the **Arduino Integrated Development Environment (IDE)**, which is used for writing, compiling, and uploading code to Arduino boards.  Getting Started with Arduino IDE 2 | Arduino Documentation  *Figure 3: Arduino IDE (Source: arduino.cc)*  Here are the key aspects:   1. **Arduino IDE**: This is the primary tool used for programming Arduino boards (<https://www.arduino.cc/en/Guide>). It provides a simple interface that allows users to write code in C/C++ and upload it to the board. The IDE handles all the interactions with the board, including the compilation of code into machine language that the microcontroller can execute. 2. **Programming Language**: Arduino uses a simplified version of C++ designed to make it easier for beginners. Code written for Arduino is typically broken into two main functions:    * **setup()**: Runs once when the board is powered on or reset. It's used for initialization, such as setting pin modes.    * **loop()**: Runs continuously after setup() and contains the main logic of the program, allowing for real-time interactions. 3. **Libraries**: Arduino provides a wide range of pre-written libraries to control sensors, motors, displays, and more. These libraries (<https://docs.arduino.cc/libraries/>) simplify programming by handling complex tasks in the background. For example, the **Servo** library (<https://docs.arduino.cc/libraries/servo/>) makes it easy to control a servo motor without needing to manage timing and motor position manually. 4. **Sketches**: The code written for Arduino is called a **sketch**. Sketches are written in the Arduino IDE and saved with the \*.ino extension. Once written, the sketch is uploaded to the board via USB or serial connections. 5. **Arduino Cloud Editor**: The Arduino IDE can also be run in a browser through the Arduino Cloud/Web Editor (<https://docs.arduino.cc/learn/starting-guide/the-arduino-web-editor/>), which allows for cloud-based project management and easy access to Arduino's vast library of examples and documentation. 6. **Serial Monitor**: This feature of the Arduino IDE allows real-time communication between the Arduino board and the computer. It’s useful for debugging and monitoring outputs from sensors or for sending commands to the Arduino from a computer. 7. **Arduino Create**: A cloud-based platform that integrates with the Arduino Web Editor. It offers project sharing, code management, and collaboration features, enhancing the Arduino experience by connecting users with a global maker community.  3.1.4. Overview of Arduino Board **Arduino Boards** are microcontroller-based platforms used to create interactive electronic projects. They provide a flexible and easy-to-use environment for users to control a wide range of physical devices, from sensors and actuators to motors and displays.  Here are some key Arduino boards and components:  **1. Arduino Boards**  **Arduino Uno**: One of the most popular models, the Uno features an ATmega328P microcontroller, offering 14 digital I/O pins, 6 analog inputs, and a USB connection for programming. It’s ideal for beginners and used in various simple projects, such as LED blinkers and sensor readings.    *Figure 4: Arduino UNO (Source: why.gr)*  **Arduino Mega 2560**: This board offers more I/O pins and memory than the Uno, with 54 digital I/O pins and 16 analog inputs. It's well-suited for more complex projects like robotics and automation systems.    *Figure 5: Arduino Mega 2560 (Source: electroschematics.com)*  **Arduino Nano**: A smaller, more compact version of the Uno, the Nano is often used for projects where space is limited. It features similar functionality but comes in a smaller form factor, making it perfect for embedded systems.  Arduino compatible Nano with CH340 USB IC  *Figure 6: Arduino Nano (Source: hobbycomponents.com)*  **Arduino Leonardo**: This board integrates a USB controller, allowing it to act as a computer peripheral like a keyboard or mouse. It's often used for human-computer interaction projects.  Arduino Leonardo with Headers — Arduino Official Store  *Figure 7: Arduino Leonardo (Source: arduino.cc)*  **Arduino Portenta H7**: A high-performance board designed for industrial applications, featuring dual ARM Cortex-M7 and M4 processors. It supports advanced tasks like artificial intelligence (AI) and machine learning.  Portenta H7 — Arduino Official Store  *Figure 8: Arduino H7 (Source: arduino.cc)*  **Arduino Nano 33 IoT**: Designed for IoT applications, it features built-in Wi-Fi and Bluetooth, making it suitable for connected devices.  Arduino Nano 33 IoT — Arduino Online Shop  *Figure 9: Arduino Nano 33 (Source: arduino.cc)*  **Arduino UNO R4**: The latest iteration of the iconic UNO series, available in two variants: UNO R4 WiFi and UNO R4 Minima. It features a 32-bit Renesas RA4M1 processor, increased memory, USB-C connectivity, CAN bus support, and enhanced analog capabilities. It's backward-compatible with existing shields and libraries, maintaining the traditional UNO form factor.  Arduino® UNO R4 WiFi — Arduino Official Store  *Figure 10: Arduino UNO R4 WiFi (Source: arduino.cc)*  In Cyprus, a notable innovation is the **Kypruino UNO+**, the first Arduino-compatible board developed in Cyprus. It is a local adaptation of the popular **Arduino UNO R3**, featuring enhanced specifications like a four-layer PCB design, integrated components such as buzzers and RGB LEDs, and full compatibility with the Arduino ecosystem. It was launched as part of the Robotex Cyprus event to inspire innovation and technological advancement on the island.  KYPRUINO UNO+ – ROBO CY  *Figure 11: Kypruino UNO+ Robotex CY Limited Edition (Source: robo.com.cy)* 3.1.5. Overview of Arduino Components  * **Sensors**: Sensors are key components used to detect changes in the environment. Examples include temperature sensors (like the **DHT11**), motion sensors (e.g., **PIR sensors**), and light sensors (e.g., **LDR**). These allow Arduino projects to interact with the real world by taking in environmental data. * **Actuators**: Actuators control the physical output of a project. These include **motors** (e.g., DC motors, stepper motors), **servos** for precise movement, and **LEDs** for visual feedback. Motors can be controlled with motor driver modules to regulate speed and direction. * **Shields**: Arduino shields are additional boards that can be stacked onto the main Arduino board to add functionality. Common shields include the **motor shield** for controlling motors, the **Ethernet shield** for network connectivity, and the **Wi-Fi shield** for wireless communication. * **Power Supply**: Arduino boards can be powered via USB or an external source. Common external power sources include a 9V battery or an external power adapter. Some boards, like the **Arduino Uno**, include a voltage regulator to manage the power supply efficiently. * **Breadboard and Jumper Wires**: A **breadboard** is used for prototyping and building circuits without soldering. **Jumper wires** connect components on the breadboard, enabling the creation of more complex circuits.  3.1.6. Basic Concepts: Digital vs. Analog, Input vs. Output **Digital vs. Analog**   * **Digital** signals represent information in discrete binary values (0s and 1s), essentially representing two states: **ON** (1) and **OFF** (0). Arduino's digital pins can be configured as inputs or outputs, allowing devices to communicate or respond using simple high (1) or low (0) states. For example, an LED connected to a digital pin can be turned on or off based on these two states.   **Example**: An Arduino digital pin might read the state of a button (pressed = 1, not pressed = 0) or control a relay (on = 1, off = 0).   * **Analog** signals, on the other hand, can represent a continuous range of values, such as voltage variations. Arduino boards have analogue pins that can read or produce analogue signals, allowing for finer control and measurement. For instance, an analogue sensor (like a temperature or light sensor) might return values like 0-1023, with a range corresponding to a variable voltage (0V to 5V) depending on the intensity of the signal being measured.   **Example**: The **analogRead()** function on Arduino takes an analogue input (e.g., from a potentiometer) and converts it to a digital value, which can then be used in programming.  **Digital** signals are discrete (on/off, 0/1), while **Analog** signals are continuous (with a range of values).  **Input vs. Output**   * **Input** refers to the ability of the Arduino to receive data from external sources, such as sensors or switches. The Arduino can read data from input devices, allowing the board to respond dynamically based on environmental conditions.   **Example**: An Arduino input pin can read the value of a temperature sensor (like a DHT11 or LM35) and use that value to perform actions (e.g., turn on a fan if the temperature is too high).   * **Output** refers to the ability of the Arduino to send data or control other devices. The board sends signals to devices like motors, LEDs, or displays, triggering actions in response to the inputs it has processed.   **Example**: A digital output pin can be used to control an LED, turning it on or off depending on the logic defined in the program.  **Input** allows the Arduino to sense its environment (e.g., sensors, switches), while **Output** allows it to control devices based on the received data (e.g., motors, LEDs). 3.1.7. Setting up the Arduino IDE The **Arduino IDE** (Integrated Development Environment) is the software used to write code (called "sketches") and upload it to the Arduino board. The programming language is based on C/C++ and is simplified to help beginners get started quickly. It includes built-in libraries and examples for various components, making it easy to interface with sensors, actuators, and other devices.  Setting up the Arduino environment is a crucial first step in getting started with Arduino projects. The process involves installing the **Arduino IDE (Integrated Development Environment)**, which is used to write, compile, and upload code to the Arduino board. Here's how to set it up:  **Download the Arduino IDE**   * **Official Website**: Go to the official Arduino website (<https://www.arduino.cc/en/software>) and download the Arduino IDE suitable for your operating system (Windows, macOS, or Linux). * **Installation**: Follow the installation instructions for your platform. On Windows, the installation is straightforward (just follow the prompts). On macOS and Linux, you may need to provide additional permissions or dependencies.   **Install Drivers (for Windows)**  If you're using Windows, you may need to install drivers for the Arduino board to be recognized by your computer. These drivers are typically bundled with the IDE during installation, but if not, they can be found on the Arduino website or in the board's folder after installation.  **Connect Your Arduino Board**   * Use a **USB cable** to connect your Arduino board (such as the Arduino Uno or Arduino Nano) to your computer. The USB cable will both power the board and allow it to communicate with the IDE.   **Select the Arduino Board and Port**  Once the IDE is open:   * **Select your Board**: Go to **Tools > Board** and select the specific model of Arduino you're using (e.g., Arduino Uno, Arduino Mega). * **Select the Port**: Go to **Tools > Port** and choose the correct port where your Arduino is connected (usually labelled with "Arduino" in the name).  3.1.8. Introduction to Arduino Programming Language The **Arduino programming language** is a simplified version of C/C++ designed to be easy to learn, even for beginners. It allows you to control hardware components such as sensors, motors, and LEDs, making it ideal for creating interactive electronics projects. 3.1.9. Structure of an Arduino Sketch An Arduino program, called a **sketch**, consists of two main functions:   * **setup()**: This function runs once when the Arduino is powered on or reset. It’s used for initialization, such as setting pin modes and starting serial communication. * **loop()**: This function runs continuously after setup(), and it’s where the main logic of your program goes. It allows your program to react to sensor inputs, control outputs, and manage timing.   A screenshot of a computer program  Description automatically generated  *Figure 12: Sketch Example – Setup and Loop*   * **Arduino Cloud Editor**: If you prefer not to install anything, you can use the **Arduino Cloud Editor**. This is a cloud-based version of the Arduino IDE, allowing you to write and store code online (<https://create.arduino.cc/>). * **Libraries**: Arduino's vast library of pre-written code can help simplify interfacing with sensors and components. You can install libraries from **Sketch > Include Library > Manage Libraries**.   For more detailed guidance, you can refer to the [official Arduino setup page](https://www.arduino.cc/en/Guide/HomePage). |

| **Activity** | |
| --- | --- |
| **Activity 3.1.1** | **Research and Presentation Task** |
| Assign students to research one of the Arduino boards presented in Chapter 3.1.4 |
| **Sources:**   * CODEDU - Module 3.1 for teachers |
| **Extra contents:**  **Hint:** Have them present its features, capabilities, and example applications to the class. |

| **Activity 3.1.2 (Assessment of Module 3.1)** |
| --- |
| **Type**: Multiple choice |
| **Question:** What is the primary function of the microcontroller on an Arduino board? |
| **Answers**: (Correct answer in Bold)  A. To supply power to external components **B. To act as the "brain" of the board, executing instructions**  C. To provide communication interfaces for external modules D. To regulate voltage on the board |

| **Activity 3.1.3 (Assessment of Module 3.1)** |
| --- |
| **Type**: Interactive Activity |
| **Activity:** Provide students with an Arduino UNO board diagram (or a physical board) |
| **Ask them to identify and label the following:**   * Microcontroller * Power port * Digital pins * Analog pins * USB port * Reset button |

| **Activity 3.1.4 (Assessment of Module 3.1)** |
| --- |
| **Type**: Short-Answer question |
| **Activity:** Digital vs. Analog |
| **Question:**  Explain the difference between digital and analog signals.  Provide an example of each type of signal in an Arduino project. |

| **Activity 3.1.5 (Assessment of Module 3.1)** |
| --- |
| **Type**: Short-Answer question |
| **Activity:** Input vs. Output |
| **Question:**  Describe what is meant by input and output in an Arduino system.  Give an example of each using sensors and actuators. |

| **Activity 3.1.6 (Assessment of Module 3.1)** |
| --- |
| **Type**: Quiz |
| **Activity:** Libraries and IDE Usage |
| **Question:** What is the purpose of the setup() and loop() functions in an Arduino sketch? |
| **Follow-Up:** Name two libraries available in the Arduino IDE and their use. |

Activity Assessment criteria:

Activity 3.1.1 (40%), Activity 3.1.2 (10%), Activity 3.1.3 (20%), Activity 3.1.4 (10%), Activity 3.1.5 (10%), Activity 3.1.6 (10%)

| **Module 3.2** |
| --- |
| **Fundamentals of Coding with Arduino** |
| Coding with Arduino allows you to bring electronics to life by programming the Arduino board to interact with the physical world. This module covers the key concepts you need to understand to get started with Arduino programming, including variables, control structures, functions, and libraries. By mastering these basics, you'll be equipped to create a wide variety of projects, from simple LEDs to complex robots. 3.2.1. Basic programming concepts with Arduino. **Variables and Data Types**  Arduino uses variables to store data. These can hold values like numbers, text, or sensor readings. Some common data types are:   * **int**: Stores integers (whole numbers) * **float**: Stores decimal numbers * **char**: Stores single characters * **string**: Stores a series of characters (text)   A white background with black text  Description automatically generated  *Figure 13: Example – int and float*  **Control Structures**  The Arduino language supports several control structures used to control the flow of the program:   * **if/else:** Conditional statements used to check if something is true or false * **for, while:** Loop structures used to repeat actions * **switch:** Used to execute one of many code blocks based on the value of a variable   A close-up of a computer screen  Description automatically generated  *Figure 14: Example – if and else*  A computer code with text  Description automatically generated  *Figure 15: Example – for* 3.2.2. Functions Functions are reusable blocks of code that perform specific tasks. They help organize and simplify your program, making it easier to debug and maintain. Functions are defined using the **void** keyword when they do not return a value.  A computer code with text  Description automatically generated  *Figure 16: Example – void* 3.2.3. Libraries Arduino libraries (<https://docs.arduino.cc/libraries/>) are collections of pre-written code that simplify the interaction with different components, such as sensors, displays, or motors. By using libraries, you can avoid writing complex code from scratch. Libraries like **Servo**, **Wire (for I2C communication)**, and **SPI** help control motors, read sensors, and manage communication with other devices. To use a library, you can include it at the top of your sketch.  A white background with black text  Description automatically generated  *Figure 17: Example – Servo library* 3.2.4. Functions for Hardware Interaction Arduino offers a variety of built-in functions to interact with hardware:   * **pinMode()**: Configures a pin as an input or output * **digitalWrite()**: Sets a digital pin HIGH (on) or LOW (off) * **digitalRead()**: Reads the state of a digital input pin * **analogWrite()**: Sends a PWM signal to a pin (useful for dimming LEDs or controlling motor speed) * **analogRead()**: Reads the value of an analogue sensor   A close-up of a white background  Description automatically generated  *Figure 18: Example – pinMode and analogWrite* 3.2.5. Serial Communication Arduino has built-in functions for serial communication, allowing it to send and receive data from a computer or other devices. This is useful for debugging, displaying sensor values, or sending commands.   * **Serial.begin()**: Initializes serial communication. * **Serial.print():** Sends data to the serial monitor. * **Serial.read():** Reads incoming data from the serial port.   A close-up of a computer code  Description automatically generated  *Figure 19: Example – serial.begin and serial.print* 3.2.6. Basic Arduino Sketch Structure Open the **Arduino IDE** and write your first program, called a **sketch**. A simple example is the "Blink" program, which blinks an LED connected to pin 13 on the Arduino.  The basic structure of a sketch is:  A screenshot of a computer program  Description automatically generated  *Figure 20: Example – “Blink”sketch*  This simple sketch will cause the onboard LED (on pin 13) to blink once every second. 3.2.7. Upload the Sketch After writing the code, click the **Upload** button (right arrow) in the Arduino IDE. This will compile your sketch and upload it to the Arduino board via USB. The **TX** (transmit) and **RX** (receive) LEDs on the board will blink, indicating that the code is being uploaded. 3.2.8. Serial Monitor If you want to communicate with the Arduino or debug your code, you can use the **Serial Monitor**. To access it, go to **Tools > Serial Monitor** or click the magnifying glass icon in the top right. The Serial Monitor is useful for sending and receiving messages from the Arduino, especially for reading sensor values or debugging outputs. 3.2.9. Lesson planning that utilises Arduino boards for various coding concepts (e.g., sensor input, output control, programming logic). The following lesson ideas introduce fundamental coding concepts using Arduino, an open-source electronics platform that simplifies building interactive projects. These lessons are designed to guide students through essential programming principles while incorporating real-world applications and hands-on experiences. By using Arduino boards, students will explore concepts such as sensor input, output control, logic programming, and loops, helping them understand the basics of how code interacts with hardware.  These lessons build on each other, providing students with a comprehensive understanding of basic coding and electronics. By the end, they will have learned how to interact with sensors and control devices based on input, laying the foundation for more advanced projects and deeper coding knowledge. 3.2.10. Writing and Uploading Your First Program (Blink LED) One of the first and most fundamental exercises for anyone starting with Arduino is writing and uploading the "Blink" program. This simple yet effective program allows users to blink an LED on and off at regular intervals, providing an immediate demonstration of how to write, compile, and upload code to an Arduino board. Below is a step-by-step guide to writing and uploading this program, with an explanation of the key concepts and steps involved.  Once the IDE is installed and your Arduino is connected, you can begin writing your first program. Open the Arduino IDE and create a new sketch by selecting **File > New**. In the IDE, you will see two main sections where you will write your code: setup() and loop().  Here is the code for the "Blink" program:  A screenshot of a computer program  Description automatically generated  *Figure 21: Example – “Blink” program*  **Key Components of the Code**   * **setup() function**: This function is called once when the program starts. It's used to initialize settings and configure pins. In this case, pinMode(LED\_BUILTIN, OUTPUT) sets the built-in LED (on pin 13) as an output. * **loop() function**: This function is repeatedly called as long as the board is powered. It contains the main logic of your program. The digitalWrite(LED\_BUILTIN, HIGH) command turns the LED on, and digitalWrite(LED\_BUILTIN, LOW) turns the LED off. The delay(1000) command pauses the program for 1000 milliseconds (1 second) before repeating.   **Upload the Program to the Arduino**  With the program written, the next step is to upload it to the Arduino board.   * **Select the Board**: Go to **Tools > Board** and select the correct model of Arduino you are using (e.g., Arduino Uno, Arduino Nano). * **Select the Port**: Go to **Tools > Port** and choose the port where your Arduino is connected. * **Upload the Code**: Click the **Upload** button (the right arrow icon in the top-left corner of the IDE). The Arduino IDE will compile the sketch, convert it into machine code, and upload it to the board. You will see the **TX** and **RX** lights on the board flash during this process.   Once the code has been uploaded, the LED on your Arduino board should start blinking on and off every second. If the LED doesn't blink, check the board and port selection and make sure that the Arduino is connected properly.  **Understanding the Code**   * **pinMode(LED\_BUILTIN, OUTPUT)**: This line configures the pin to which the built-in LED is connected (usually pin 13) as an output, meaning it will send signals to the LED to turn it on or off. * **digitalWrite(LED\_BUILTIN, HIGH)**: This turns the **LED on** by setting the voltage on the pin to HIGH (5V). * **delay(1000)**: This pauses the program for one second before proceeding to the next command. * **digitalWrite(LED\_BUILTIN, LOW)**: This turns the **LED off** by setting the voltage on the pin to LOW (0V).   **Troubleshooting**  If the LED doesn’t blink, here are a few troubleshooting tips:   * **Check your connections**: Ensure the Arduino board is properly connected to your computer. * **Correct Board and Port**: Double-check that you've selected the correct board and serial port in the **Tools** menu. * **USB Cable**: Try using a different USB cable if the Arduino board isn't recognized by the computer. * **Board Status**: Make sure the board's **ON** LED is lit, indicating it's powered correctly.   **Next Steps: Experimenting with the Code**  Once you successfully upload the "Blink" program, you can modify the code to experiment with different delays or even use external components like a push button or additional LEDs. This can help you better understand how to control outputs and time events.   * **Change the blink rate**: Modify the delay() value to change the blink speed. For example, change delay(1000) to delay(500) for the LED to blink faster. * **Use an external LED**: You can connect an external LED to pin 12 (or any other pin) using a resistor and modify the code to control it instead of the built-in LED.   For further learning, consider exploring additional examples available in the Arduino IDE under **File > Examples**. These projects offer a wide range of exercises that can enhance your understanding of how to interact with various sensors, actuators, and hardware components. 3.2.11. Understanding Syntax and Commands When introducing Arduino programming to beginners, it's crucial to start with understanding **syntax** and **commands**. Syntax refers to the set of rules that defines the structure of statements in a programming language. Commands are the instructions given to the microcontroller to perform tasks, such as turning on an LED or reading a sensor value.  **Basic Syntax**  Arduino code is written in **C++**, but it's simplified to make it more accessible. The syntax consists of the following main parts:   * **Statements**: These are the individual instructions the microcontroller will follow. For example, pinMode(13, OUTPUT); tells the board to configure pin 13 as an output pin. Every statement ends with a **semicolon** (;), which indicates the end of the instruction. * **Functions**: Functions are prewritten blocks of code that perform specific tasks. For example, digitalWrite() is a function used to send a high or low signal to a pin. You call functions in your code to make the Arduino do something. * **Curly Braces { }**: Curly braces are used to group a set of instructions. For instance, the setup() and loop() functions must contain curly braces to define the start and end of the instructions inside them.   **Key Commands in Arduino Programming**  Here are a few key commands used frequently in Arduino sketches:   * **pinMode(pin, mode)**: This command sets a specific pin (e.g., pin 13) to either an input or output mode. For example, pinMode(13, OUTPUT); configures pin 13 as an output pin, which allows you to control an LED or other components connected to that pin. * **digitalWrite(pin, value)**: This command sets the state of a digital pin to either **HIGH** or **LOW**. **HIGH** means 5V (on), and **LOW** means 0V (off). For example, digitalWrite(13, HIGH); turns the LED connected to pin 13 on. * **delay(milliseconds)**: The delay() function pauses the program for a specified number of milliseconds. For example, delay(1000); pauses the program for 1 second.   **Understanding Structure**  Arduino programs are generally structured in two main functions: setup() and loop().   * **setup()**: This function runs once when the program starts. It's used for setting up configurations, like initializing pins or serial communication. * **loop()**: This function runs continuously after setup(). It's used for repeating actions, such as turning an LED on and off.   **Reading and Writing Commands**  In addition to writing values to pins with digitalWrite(), you can also read input from sensors or switches using digitalRead().   * **digitalRead(pin)**: Reads the value of a digital pin (HIGH or LOW). For example, int buttonState = digitalRead(2); reads the state of a button connected to pin 2.   By using commands like digitalWrite(), digitalRead(), and delay(), you can control hardware devices such as LEDs, motors, and sensors, as well as manage program flow in an Arduino project.  **Error Handling and Debugging**  While working with Arduino code, errors can arise due to incorrect syntax or logic. The Arduino IDE will often highlight errors and provide hints, making it easier for beginners to debug. A common mistake is forgetting to end a line with a semicolon or mismatched parentheses. 3.2.12. Variables, Data Types, and Operators In Arduino programming, understanding **variables**, **data types**, and **operators** is fundamental. These elements form the building blocks of writing efficient and functional code, enabling you to store and manipulate data, perform calculations, and control the flow of your program.  **Variables in Arduino Programming**  A **variable** is a named storage location in your program that holds a value. The value can change or be manipulated throughout the program. For example, you may use a variable to store a sensor reading or the status of a button.  To declare a variable in Arduino, you specify its **data type** and then its name.  **Data Types in Arduino Programming**  Data types define the type of data a variable can store. Different data types are used to represent various kinds of values, such as integers, floating-point numbers, or boolean values. Common data types used in Arduino programming include:   * **int**: Represents an integer (whole number), typically ranging from -32,768 to 32,767 * **float**: Represents a floating-point number, which includes decimals * **char**: Represents a single character * **boolean**: Represents true or false values. It is useful for binary decisions or flags * **long**: Represents larger integers that don't fit into the standard int data type   **Operators in Arduino Programming**  **Operators** are symbols used to perform operations on variables or values. In Arduino, operators allow you to perform arithmetic calculations, compare values, or combine logical expressions. The most common types of operators include:   * **Arithmetic Operators**: Used to perform basic math operations.   + + (addition)   + - (subtraction)   + \* (multiplication)   + / (division)   + % (modulo, or remainder of division) * **Comparison Operators**: Used to compare two values and return a boolean result (true or false).   + == (equal to)   + != (not equal to)   + > (greater than)   + < (less than)   + >= (greater than or equal to)   + <= (less than or equal to) * **Logical Operators**: Used to combine multiple conditions.   + && (AND): Returns true if both conditions are true.   + || (OR): Returns true if at least one condition is true.   + ! (NOT): Reverses the logical state (true becomes false, and false becomes true). * **Assignment Operators**: Used to assign a value to a variable.   + = (simple assignment)   + +=, -=, \*=, /=, etc. (combined assignment) |

| **Activity** | |
| --- | --- |
| **Activity 3.2.1** | **Variables and Data Types** |
| Write a sketch that stores a sensor reading in a variable, modifies it (e.g., adds 10), and displays the result in the Serial Monitor. What data type would you use to store the following values, and why? |
| **Sources:**   * CODEDU - Module 3.2 for teachers * Arduino website |
| **Extra contents:**   * A temperature reading with decimals (e.g., 23.4°C). * The state of a button (pressed or not pressed). * A series of characters, such as "Hello, Arduino!" |

| **Activity** | |
| --- | --- |
| **Activity 3.2.2** | **Control Structures** |
| Create a program using an if/else statement to blink an LED at a different rate depending on whether a button is pressed. |
| **Sources:**   * CODEDU - Module 3.2 for teachers * Arduino website |
| **Extra contents:** Explain the difference between a for loop and a while loop. Provide an example of when you’d use each. |

| **Activity** | |
| --- | --- |
| **Activity 3.2.3** | **Functions** |
| Write a reusable function named blinkLED that takes two parameters: the pin number and the delay time. Use it to blink two different LEDs at different rates. |
| **Sources:**   * CODEDU - Module 3.2 for teachers |
| **Extra contents:** Why are functions useful in programming, and how can they make debugging easier? |

| **Activity** | |
| --- | --- |
| **Activity 3.2.4** | **Write a basic sketch** |
| Ask students to write a simple Arduino program to blink the built-in LED on the board every second. |
| **Sources:**   * CODEDU - Module 3.2 for teachers * Arduino website |
| **Extra contents:**  **Hint:** The LED is connected to pin 13 on most Arduino boards. |

| **Activity** | |
| --- | --- |
| **Activity 3.2.5** | **Debugging exercice** |
| Provide students with a faulty Arduino sketch where an LED is supposed to blink but doesn’t.  Ask them to debug and fix the issue. |
| **Sources:**   * CODEDU - Module 3.2 for teachers * Arduino website |
| **Extra contents:**  **Hint:** Example Error: Missing pinMode() in setup() or incorrect pin assignment. |

| **Activity 3.2.6 (Assessment of Module 3.2)** |
| --- |
| **Type**: Multiple choice |
| **Question:** Which of the following data types is used to store a number with decimals in Arduino? |
| **Answers: (Correct answer in Bold)**   1. int 2. **float** 3. char 4. string |

| **Activity 3.2.7 (Assessment of Module 3.2)** |
| --- |
| **Type**: Multiple choice |
| **Question:** Which of the following control structures in Arduino is used to repeat a block of code until a condition is no longer true? |
| **Answers: (Correct answer in Bold)**   1. for 2. if/else 3. switch 4. **while** |

| **Activity 3.2.8 (Assessment of Module 3.2)** |
| --- |
| **Type**: Short-Answer question |
| **Activity:** pinmode |
| **Question:**  What is the purpose of the pinMode() function in Arduino? Give an example of its usage. |
| **Answer:**  The pinMode() function is used to configure a specific pin as either an input or output. Example: pinMode(13, OUTPUT); sets pin 13 as an output, often used for controlling an LED. |

| **Activity 3.2.9 (Assessment of Module 3.2)** |
| --- |
| **Type**: Multiple choice |
| **Activity:** Understanding the code |
| **Question:**  What will the following code do?  *int buttonState = digitalRead(7);*  *if (buttonState == HIGH) {*  *digitalWrite(13, HIGH);*  *} else {*  *digitalWrite(13, LOW);*  *}* |
| **Answers: (Correct answer in Bold)**   1. Turn the LED on when the button is not pressed 2. Turn the LED off when the button is pressed 3. **Turn the LED on when the button is pressed** 4. Nothing, because there is no setup function |

| **Activity 3.2.10 (Assessment of Module 3.2)** |
| --- |
| **Type**: Short-Answer question |
| **Activity:** Digital vs. Analog |
| **Question:**  Explain the difference between the digitalWrite() and analogWrite() functions in Arduino programming. |
| **Answer:**  digitalWrite() is used to set a digital pin either HIGH (on) or LOW (off), whereas analogWrite() sends a Pulse Width Modulation (PWM) signal to a pin to simulate varying voltage, often used to control the brightness of LEDs or the speed of motors. |

| **Activity 3.2.11 (Assessment of Module 3.2)** |
| --- |
| **Type**: Short-Answer question |
| **Activity:** Understannding funtions |
| **Question:** What does the Serial.begin() function do, and why is it important in Arduino programming? |
| **Answer:**  Serial.begin() initializes serial communication at a specified baud rate, allowing Arduino to send and receive data from a computer or other devices. It is important for debugging, monitoring sensor values, and sending commands. |

| **Activity 3.2.12 (Assessment of Module 3.2)** |
| --- |
| **Type**: Interactive Activity |
| **Activity:** Provide students with a breadboard, jumper wires, and an LED |
| **Ask them to:**   * Connect the LED to an Arduino digital pin. * Write a sketch to turn the LED on for 2 seconds and off for 1 second.Digital pins * Analog pins * USB port * Reset button |

Activity Assessment criteria:

Activity 3.2.1 (15%), Activity 3.2.2 (15%), Activity 3.2.3 (15%), Activity 3.2.4 (10%), Activity 3.2.5 (10%), Activity 3.2.6 (5%), Activity 3.2.7 (5%), Activity 3.2.8 (5%), Activity 3.2.9 (5%), Activity 3.2.10 (5%), Activity 3.2.11 (5%), Activity 3.2.12 (5%)