

| **CODEDU’s Students’ Training Course** | | | | | | |
| --- | --- | --- | --- | --- | --- | --- |
| **Section 2: Introduction to Arduino, Physical Computing, and Coding** | | | | | | |
| **Subject: Introduction to Arduino** | **Duration (in hours):** | | | | **Module 2.1: 1 school period – 45 minutes**  **Module 2.2: 1 school period – 45 minutes**  **Project implementation: Depends on the project** | |
| **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:** | | **Form of participation in the learning activity:** | | | | |
| **Expected Learning outcomes:**  • Understanding the fundamental concepts of Arduino and its applications.  • Gaining familiarity with the Arduino hardware and software environment.  • Developing basic programming skills in the Arduino IDE.  • Learning to read and write simple Arduino programs. | | • Ability to control basic electronic components using Arduino.  • Developing critical thinking and problem-solving skills.  • Fostering creativity and innovation through hands-on projects.  • Developing a portfolio of Arduino projects.  • Gaining confidence in working with electronics and technology. | | | | |
| **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: This course introduces students to the exciting world of Arduino, a powerful open-source platform for physical computing. Students will learn about the hardware and software components of Arduino, develop basic programming skills, and engage in hands-on projects to create interactive projects that combine electronics and code. This course aims to foster creativity, problem-solving, and a deeper understanding of how technology works.**    ***Key Words: Arduino, Physical Computing, Microcontroller, Programming, Electronics, Sensors, Actuators, IDE,***  ***Coding, IoT*** | | | | | | |

| **Module 2.1** |
| --- |
| **Getting Started with Arduino** |
| ***Module 2.1: Getting Started with Arduino***   * ***Project Ideas:***   + ***Robotics:***     - *Build a simple line-following robot.*     - *Create a robotic arm that can pick up and move objects.*     - *Design a robot that can avoid obstacles.*   + ***Game Development:***     - *Build a custom game controller for your favorite video game.*     - *Create an interactive musical instrument.*     - *Develop a physical input device for a virtual reality experience.*   + ***Creative Expression:***     - *Design an interactive light show.*     - *Build a sound synthesizer.*     - *Create an interactive art installation that responds to user input.*  2.1.1. 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. 2.1.2. 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. 2.1.3. 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.   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. 2.1.4. 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* 2.1.5. 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* 2.1.6. 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* 2.1.7. 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* 2.1.8. 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* 2.1.9. 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. 2.1.10. 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. 2.1.11. 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. 2.1.12. 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. 2.1.13. Project ideas The Arduino platform offers a limitless canvas for creativity and innovation. With its versatility and ease of use, you can bring a wide range of projects to life, from simple interactive gadgets to complex robotic systems.  In the realm of robotics, Arduino provides the foundation for building a variety of fascinating creations. You can construct a line-following robot that autonomously navigates along a designated path, a robotic arm that can manipulate objects with precision, or an obstacle-avoiding robot that can navigate its surroundings safely. These projects not only provide hands-on experience with robotics principles but also encourage critical thinking and problem-solving skills.  The gaming world also provides fertile ground for Arduino-based projects. You can design custom game controllers that offer unique input options, creating a more immersive and personalized gaming experience. Alternatively, you can build interactive musical instruments that respond to your movements and gestures, transforming the act of music-making into a dynamic and engaging experience.  For those seeking creative expression, Arduino offers a unique medium for artistic exploration. You can design interactive light shows that respond to music or ambient sounds, creating a mesmerizing visual spectacle. Alternatively, you can build a sound synthesizer that generates unique and expressive soundscapes, pushing the boundaries of musical creativity.  The possibilities are truly endless. With Arduino, you have the power to transform your imagination into reality, turning abstract ideas into tangible creations that interact with the world around you. Whether you're a budding engineer, a creative artist, or simply someone who enjoys tinkering and exploring new technologies, Arduino offers a rewarding and enriching experience that will continue to inspire and amaze for years to come. |

| **Activity** | |
| --- | --- |
| **Activity 2.1.1** | **Research and Presentation Task** |
| Research one Arduino board and present its features, capabilities, and example applications to the class. |

| **Activity 2.1.2 (Assessment of Module 2.1)** |
| --- |
| **Type**: Multiple choice |
| **Question:** What is the primary function of the microcontroller on an Arduino board? |
| **Answers**:  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 2.1.3 (Assessment of Module 2.1)** |
| --- |
| **Type**: Interactive Activity |
| **Activity:** Identify and label the following on an Arduino UNO board diagram (or a physical board) |
| * Microcontroller * Power port * Digital pins * Analog pins * USB port * Reset button |

| **Activity 2.1.4 (Assessment of Module 2.1)** |
| --- |
| **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 2.1.5 (Assessment of Module 2.1)** |
| --- |
| **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 Assessment criteria:

Activity 2.1.1 (60%), Activity 2.1.2 (10%), Activity 2.1.3 (10%), Activity 2.1.4 (10%), Activity 2.1.5 (10%)

| **Module 2.2** |
| --- |
| **Basic Coding with Arduino** |
| 2.2.1. 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. 2.2.2. 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. 2.2.3. 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 2.2.1** | **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? |
| **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 2.2.2** | **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. |
| **Extra contents:** Explain the difference between a for loop and a while loop. Provide an example of when you’d use each. |

| **Activity** | |
| --- | --- |
| **Activity 2.2.3** | **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. |
| **Extra contents:** Why are functions useful in programming, and how can they make debugging easier? |

| **Activity** | |
| --- | --- |
| **Activity 2.2.4** | **Write a basic sketch** |
| Write a simple Arduino program to blink the built-in LED on the board every second. |
| **Extra contents:**  **Hint:** The LED is connected to pin 13 on most Arduino boards. |

| **Activity** | |
| --- | --- |
| **Activity 2.2.5** | **Debugging exercice** |
| Identify in the faulty Arduino sketch where an LED is supposed to blink but doesn’t.  Debug and fix the issue. |
| **Extra contents:**  **Hint:** Example Error: Missing pinMode() in setup() or incorrect pin assignment. |

| **Activity 2.2.6 (Assessment of Module 2.2)** |
| --- |
| **Type**: Multiple choice |
| **Question:** Which of the following data types is used to store a number with decimals in Arduino? |
| **Answers:**   1. int 2. **float** 3. char 4. string |

| **Activity 2.2.7 (Assessment of Module 2.2)** |
| --- |
| **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:**   1. for 2. if/else 3. switch 4. while |

| **Activity 2.2.8 (Assessment of Module 2.2)** |
| --- |
| **Type**: Short-Answer question |
| **Activity:** Digital vs. Analog |
| **Question:**  Explain the difference between the digitalWrite() and analogWrite() functions in Arduino programming. |

| **Activity 2.2.9 (Assessment of Module 2.2)** |
| --- |
| **Type**: Short-Answer question |
| **Activity:** Understannding funtions |
| **Question:** What does the Serial.begin() function do, and why is it important in Arduino programming? |

Activity Assessment criteria:

Activity 2.2.1 (15%), Activity 2.2.2 (15%), Activity 2.2.3 (15%), Activity 2.2.4 (15%), Activity 2.2.5 (10%), Activity 2.2.6 (5%), Activity 2.2.7 (5%), Activity 2.2.8 (10%), Activity 2.2.9 (10%)