Engineering in One Video (EIOV)

Watch video on YouTube EIOV

B.Tech

Internet of Things (IoT)

Unit-4

in one video

With Notes

Programming the Arduino

(Arduino IDE, coding, using emulator, using libraries, etc.)

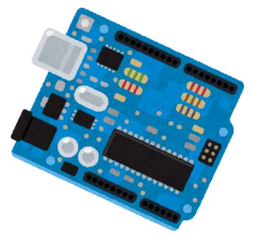
(KOT-501, KCS-712)

AKTU Exam

Internet of Things

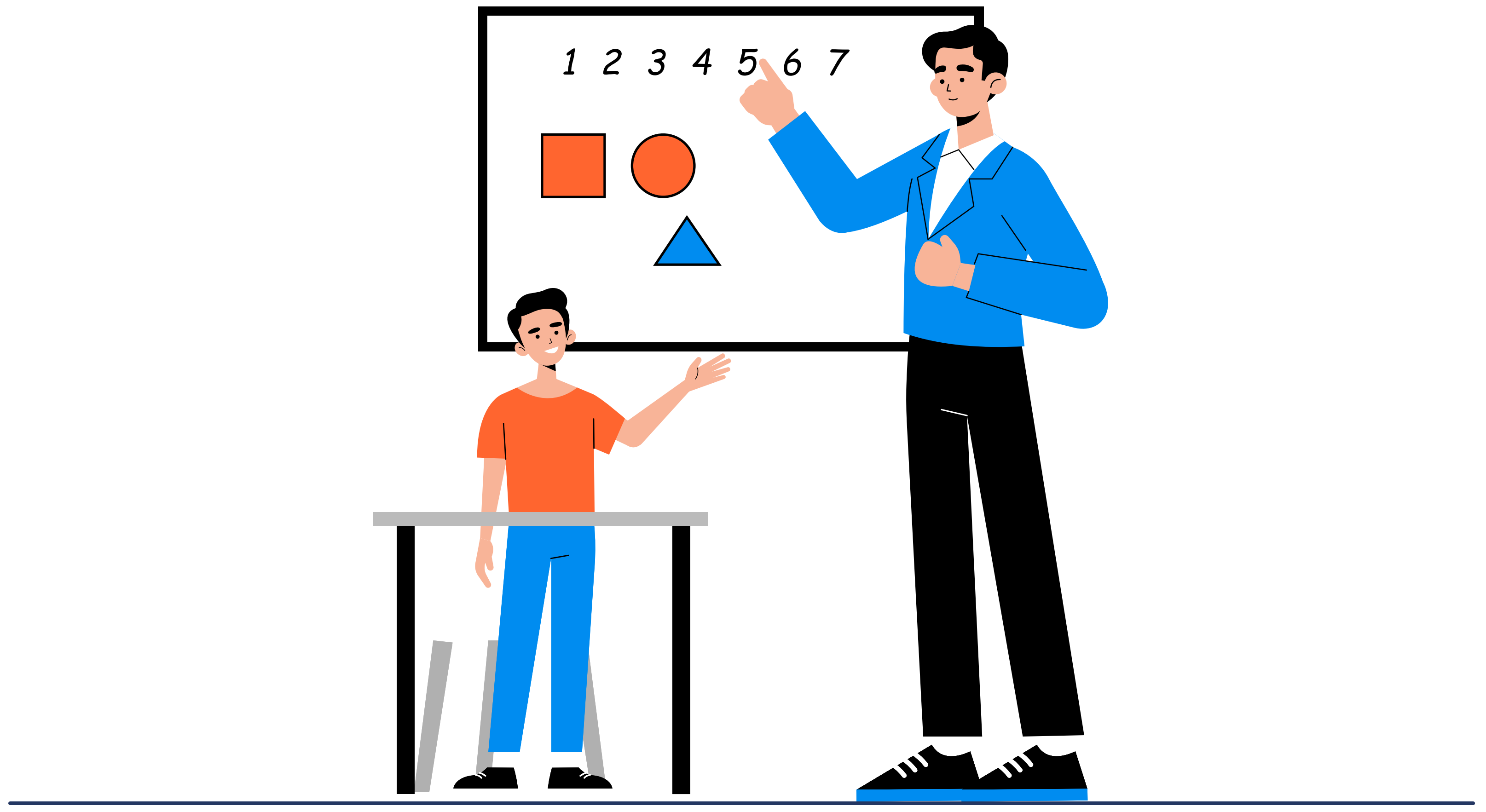
Unit-4

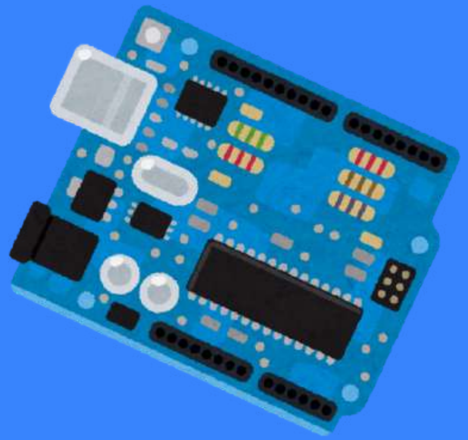
Programming the Arduino



Topics to be covered...

Anatomy/Components of Arduino platform board
Arduino IDE
Arduino coding syntax
Variable and its scope
Data types
Function libraries
Arduino emulator
Decision making statement(if, else)
Operator
Additions in Arduino
Programming the Arduino for IoT
Happy Ending!





Anatomy/Components of Arduino platform board

Anatomy/Components of Arduino board

- **Microcontroller:** The "brain" of the Arduino board, which processes and executes code. For most Arduino boards, this is an ATmega microcontroller that performs logical operations and controls other components.

- **Digital I/O Pins:** Pins that can be programmed as either input or output. They are used to interface with digital sensors, LEDs, or other devices that require digital signals (high/low).

- **Analog Input Pins:** These pins read varying voltage levels, allowing the Arduino to work with analog sensors (e.g., temperature or light sensors) and convert those signals to digital values.

- **Power Pins:** The power pins provide voltage (typically 3.3V and 5V) and ground connections, supplying power to sensors and other external devices.

- **ICSP (In-Circuit Serial Programming) Header:** Allows you to program the microcontroller directly and is useful for bootloading or low-level programming.

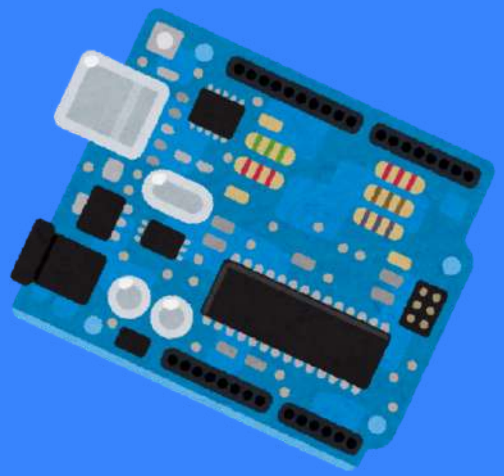
Anatomy/Components of Arduino board

- **USB Port:** Used to connect the Arduino to a computer for programming and powering the board. The USB interface is also used for serial communication, allowing the board to send and receive data with a computer.

- **Voltage Regulator:** Ensures the board receives a stable voltage, typically 5V, from external sources, protecting components from voltage spikes.

- **Crystal Oscillator:** Sets the clock speed of the microcontroller, ensuring it runs at a steady pace (e.g., 16 MHz for most Arduinos), which is crucial for timing and synchronization.

- **Reset Button:** This button resets the microcontroller, restarting the program from the beginning. It's useful for testing and debugging code.



Arduino IDE

Arduino IDE

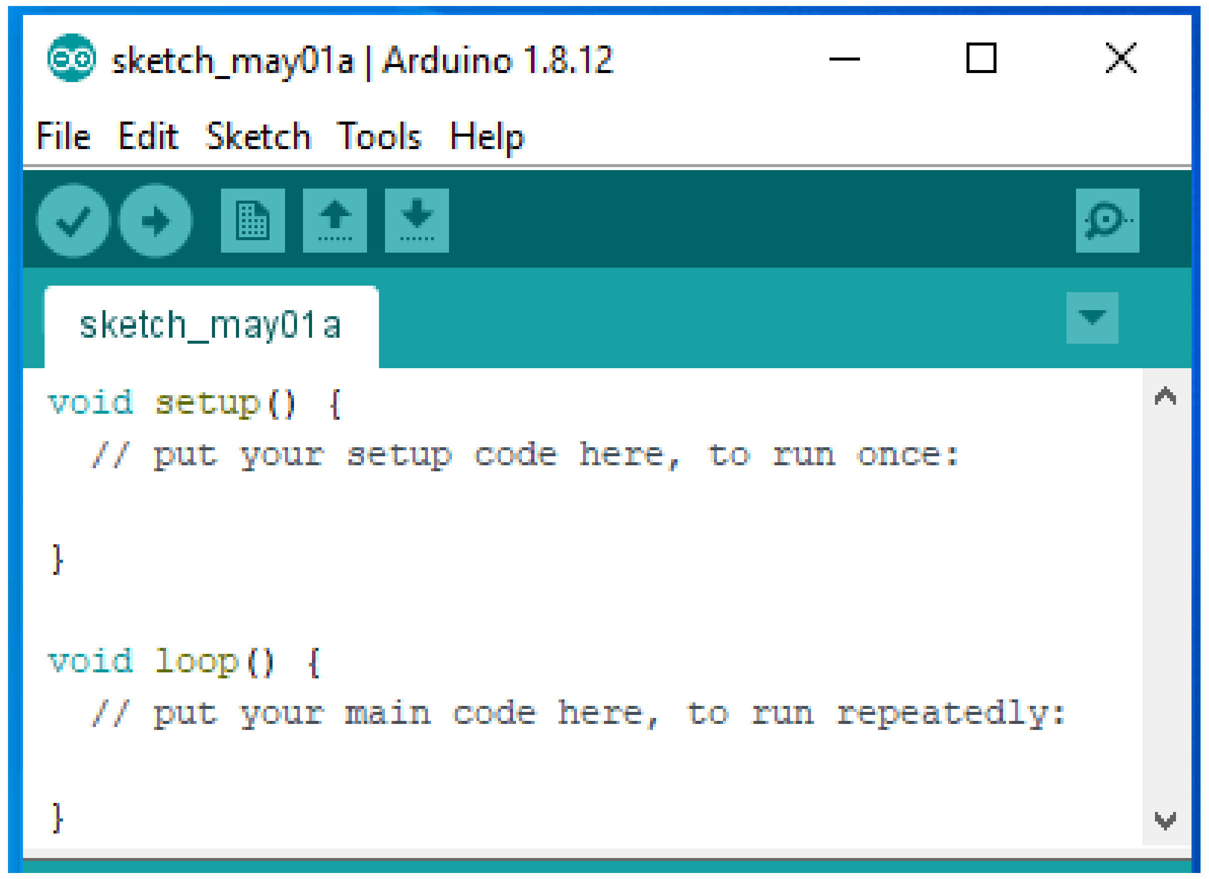
- The Arduino IDE (Integrated Development Environment) is the software platform used to write, compile, and upload code to Arduino boards. It's where you create the instructions (code) that tell the Arduino what to do.

- **Steps to Set Up an Arduino Board**
  - Download and Install the Arduino IDE
  - Connect the Arduino to Your Computer
  - Open the Arduino IDE
  - Select the Arduino Board Model
  - Select the Correct Port
  - Write or Open a Sketch (Code)
  - Verify and Upload the Code

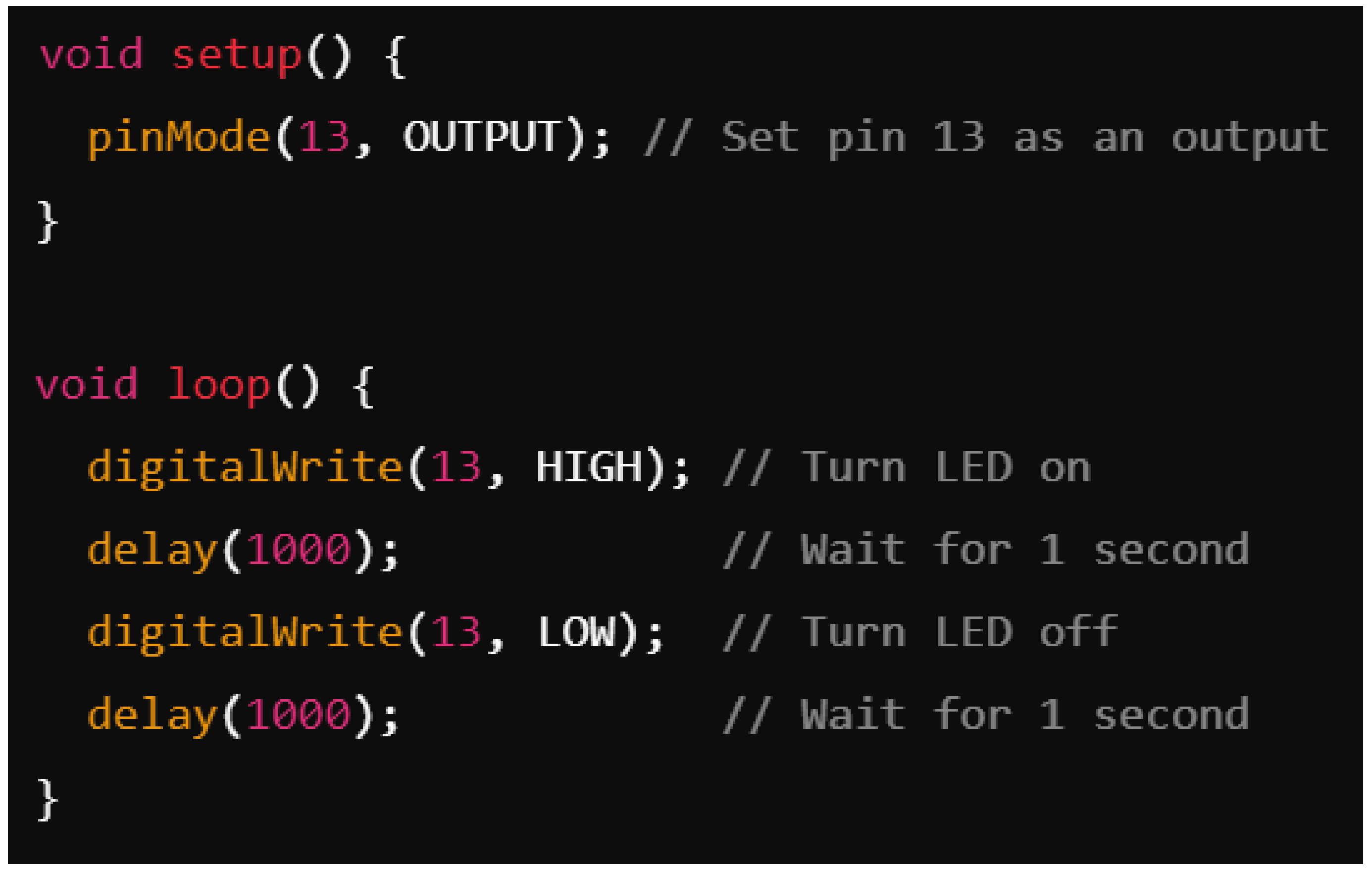
Arduino coding syntax

EIOV

Arduino coding syntax



```
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

}
```



```
void setup() {
  pinMode(13, OUTPUT); // Set pin 13 as an output
}

void loop() {
  digitalWrite(13, HIGH); // Turn LED on
  delay(1000);            // Wait for 1 second
  digitalWrite(13, LOW);  // Turn LED off
  delay(1000);            // Wait for 1 second
}
```

•

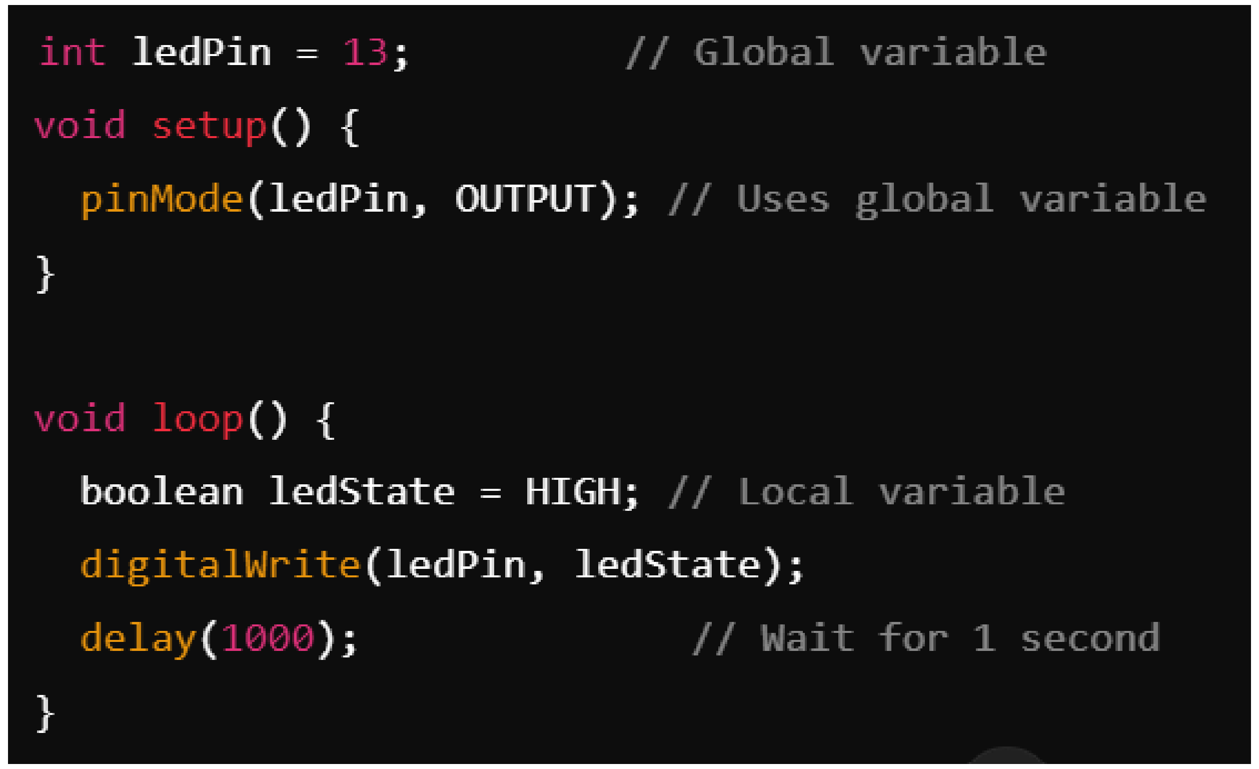
Variable and its scope

Variable and its scope

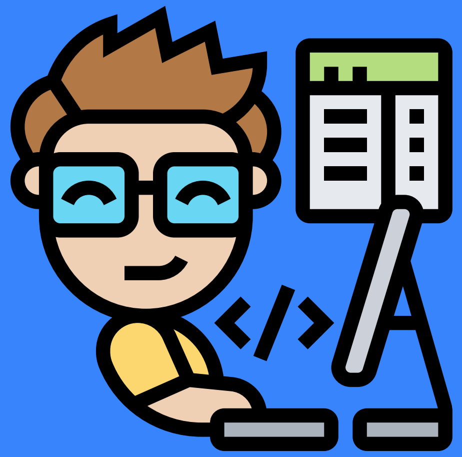
- A variable is a named storage location in memory that holds a value, which can be modified as the program runs.
- Variables make it easy to store and manipulate data, like sensor readings or control states, in your code.
- `int ledPin = 13;`
- **Scope:**
  - **Global variables** are declared outside functions and can be accessed anywhere in the code.
  - **Local variables** are declared within functions (like setup() or loop()) and can only be used within that function.

Variable and its scope

- **Data Types:**
  - int (integer) for whole numbers, like 10.
  - float for decimal numbers, like 3.14.
  - char for single characters, like 'A'.
  - boolean for true/false values.



```
int ledPin = 13;          // Global variable
void setup() {
  pinMode(ledPin, OUTPUT); // Uses global variable
}

void loop() {
  boolean ledState = HIGH; // Local variable
  digitalWrite(ledPin, ledState);
  delay(1000);              // Wait for 1 second
}
```



Data types

Data types

1. int (Integer)

- **Description:** Holds whole numbers (no decimal points) from -32,768 to 32,767.
- **Uses:** Ideal for counters, pin numbers, and sensor readings where you need integers.

2. float (Floating Point)

- **Description:** Holds decimal values with single-precision (up to 6-7 significant digits).
- **Uses:** Useful for measurements such as temperature, voltage, and distance.

3. boolean

- **Description:** Stores true or false values, represented by 1 or 0 respectively.
- **Uses:** Typically used in condition checks, toggling switches, on/off.

4. char (Character)

- **Description:** Holds a single character, such as 'A' or '3', stored as an 8-bit ASCII value.
- **Uses:** Useful for handling single characters or simple text elements.

Data types

5. String

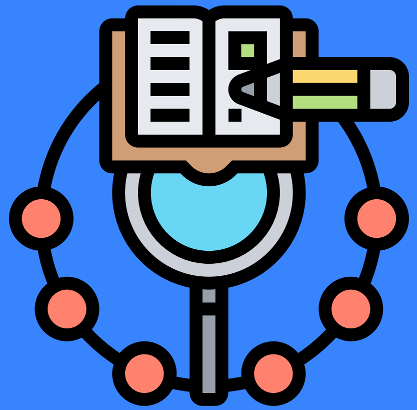
- **Description:** Allows for handling sequences of characters (text strings).
- **Uses:** Useful for displaying messages, communicating with serial monitors, or passing text data.

6. long

- **Description:** Holds larger integer values than int, ranging from -2,147,483,648 to 2,147,483,647.
- **Uses:** Useful when working with numbers larger than what int can store, such as long-distance measurements or timer values.

7. unsigned int and unsigned long

- **unsigned int:** Stores whole numbers from 0 to 65,535 (no negatives).
- **unsigned long:** Stores whole numbers from 0 to 4,294,967,295, making it ideal for very large positive values.



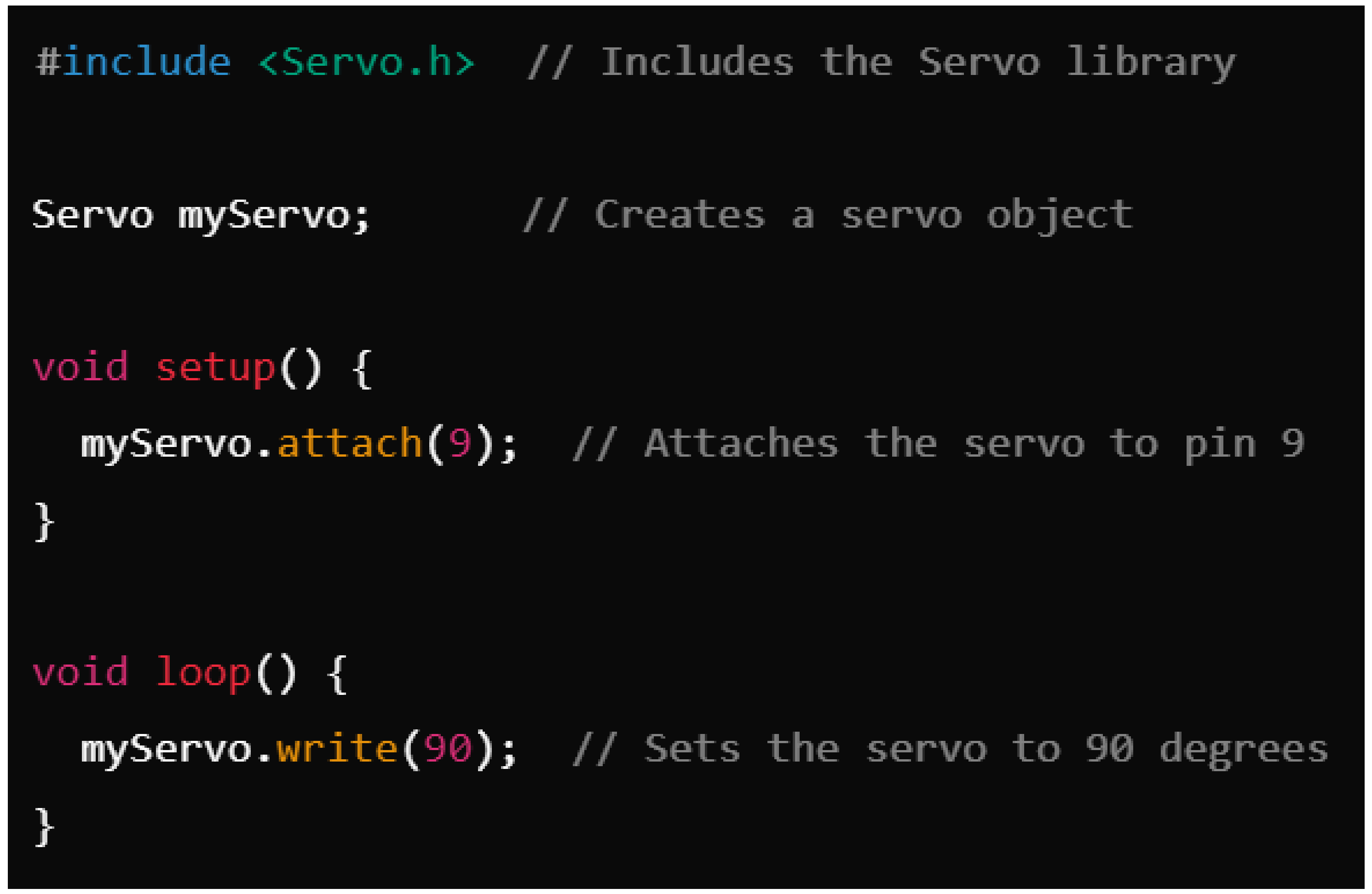
Function libraries

Function libraries

- Libraries are collections of pre-written functions that simplify complex tasks and allow you to use additional hardware or perform specialized actions in your code without writing everything from scratch.
- Libraries save time and make coding more efficient by providing reusable code that's well-documented and tested.

- **Key Points about Arduino Libraries**
  - **Purpose of Libraries:** Libraries simplify complex tasks by providing pre-written code.
  - **Use Libraries in Code:** Include it beginning of your code with #include <libraryName.h>.
  - **Library Structure:** Libraries have organized files with functions for specific hardware.
  - **Common Libraries:** Frequently used libraries include Wire, SPI, and WiFi.
  - **Installing Libraries:** Libraries can be added through the Arduino Library Manager.

Function libraries



```
#include <Servo.h>  // Includes the Servo library

Servo myServo;      // Creates a servo object

void setup() {
  myServo.attach(9);  // Attaches the servo to pin 9
}

void loop() {
  myServo.write(90);  // Sets the servo to 90 degrees
}
```

Arduino emulator

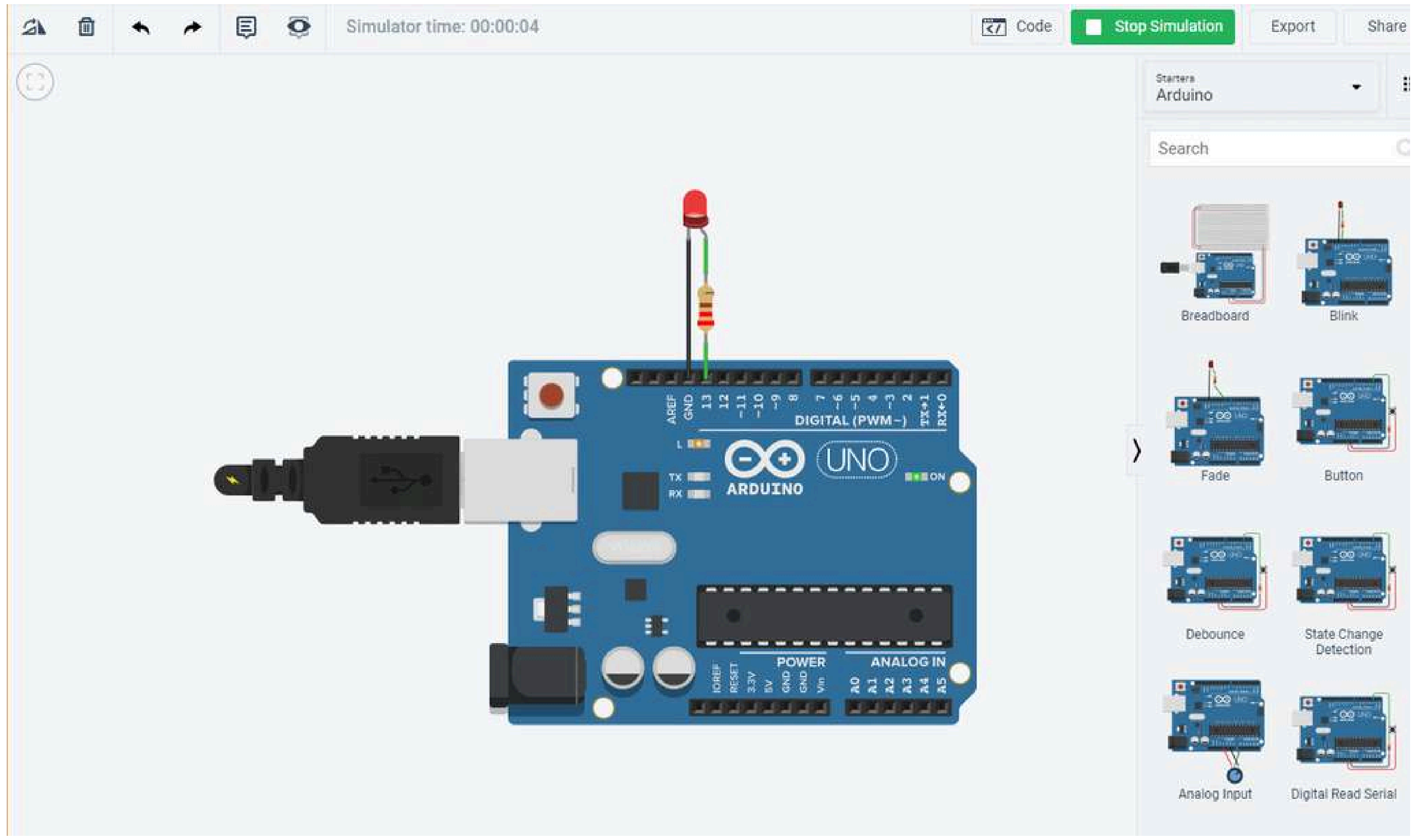
Arduino emulator

- An Arduino emulator is a software tool that allows you to simulate an Arduino board on your computer without needing physical hardware.

- Emulators are helpful for testing and debugging code in a virtual environment, which can be especially useful if you don’t have access to a specific Arduino board or component.

- **Key Points about Arduino Emulators:**
  - **Purpose of Emulators**
  - **Features**
  - **Popular Arduino Emulators:** Tinkercad Circuits, Proteus Design Suite, SimulIDE
  - **Benefits of Using an Emulator:** Cost-effective, Time-saving, Risk-free

Arduino emulator



Decision making statement(if, else)

Decision making statement(if, else)

- Decision-making statements are used to control the flow of a program based on conditions.
- These statements allow the Arduino to make decisions and execute certain parts of code only when specific conditions are met.

1. if Statement

```
if (condition) {
  // code to execute if condition is true
}
```

```
int sensorValue = analogRead(A0);
if (sensorValue > 500) {
  digitalWrite(13, HIGH);
}
```

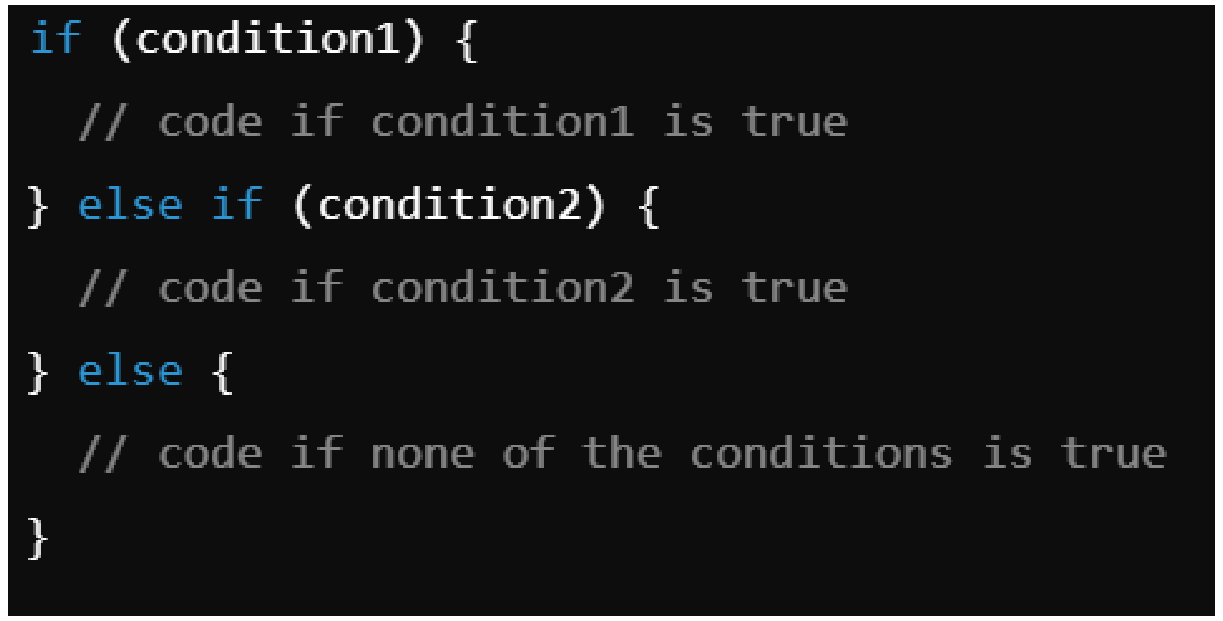
2. if-else Statement

```
if (condition) {
  // code to execute if condition is true
} else {
  // code to execute if condition is false
}
```

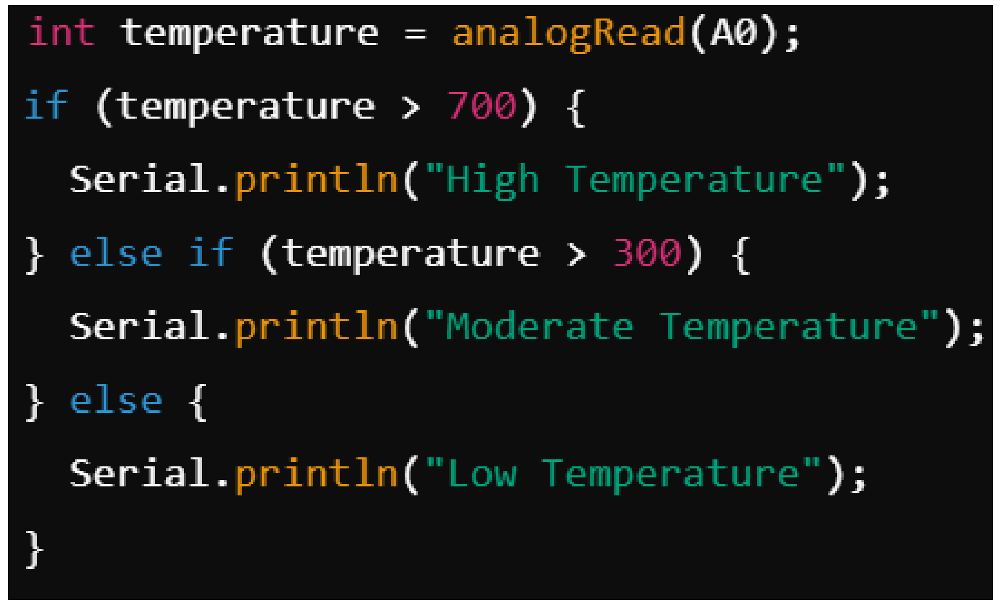
```
int sensorValue = analogRead(A0);
if (sensorValue > 500) {
  digitalWrite(13, HIGH);
} else {
  digitalWrite(13, LOW);
}
```

Decision making statement(if, else)

3. if-else if-else Statement

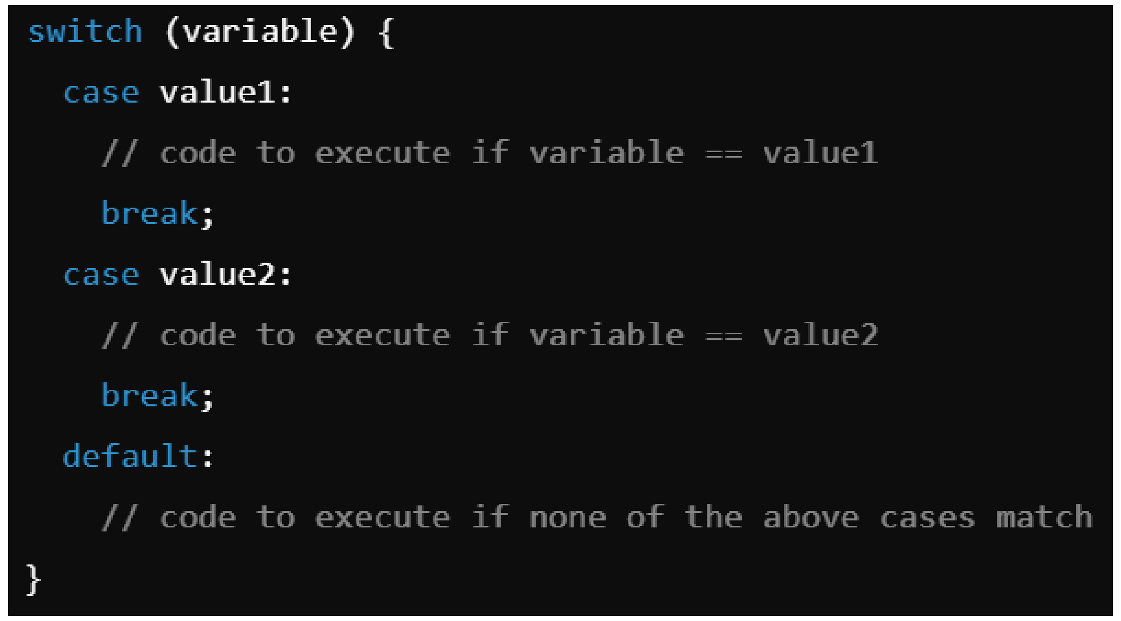


```
if (condition1) {
  // code if condition1 is true
} else if (condition2) {
  // code if condition2 is true
} else {
  // code if none of the conditions is true
}
```

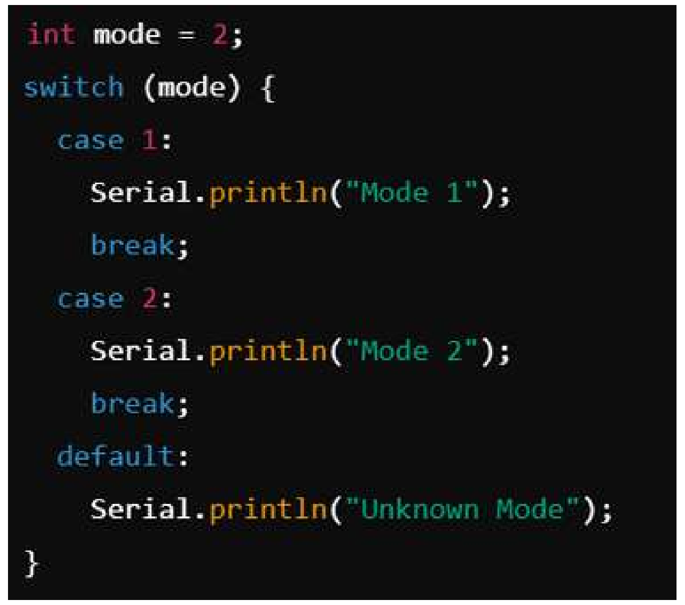


```
int temperature = analogRead(A0);
if (temperature > 700) {
  Serial.println("High Temperature");
} else if (temperature > 300) {
  Serial.println("Moderate Temperature");
} else {
  Serial.println("Low Temperature");
}
```

4. switch-case Statement



```
switch (variable) {
  case value1:
    // code to execute if variable == value1
    break;
  case value2:
    // code to execute if variable == value2
    break;
  default:
    // code to execute if none of the above cases match
}
```

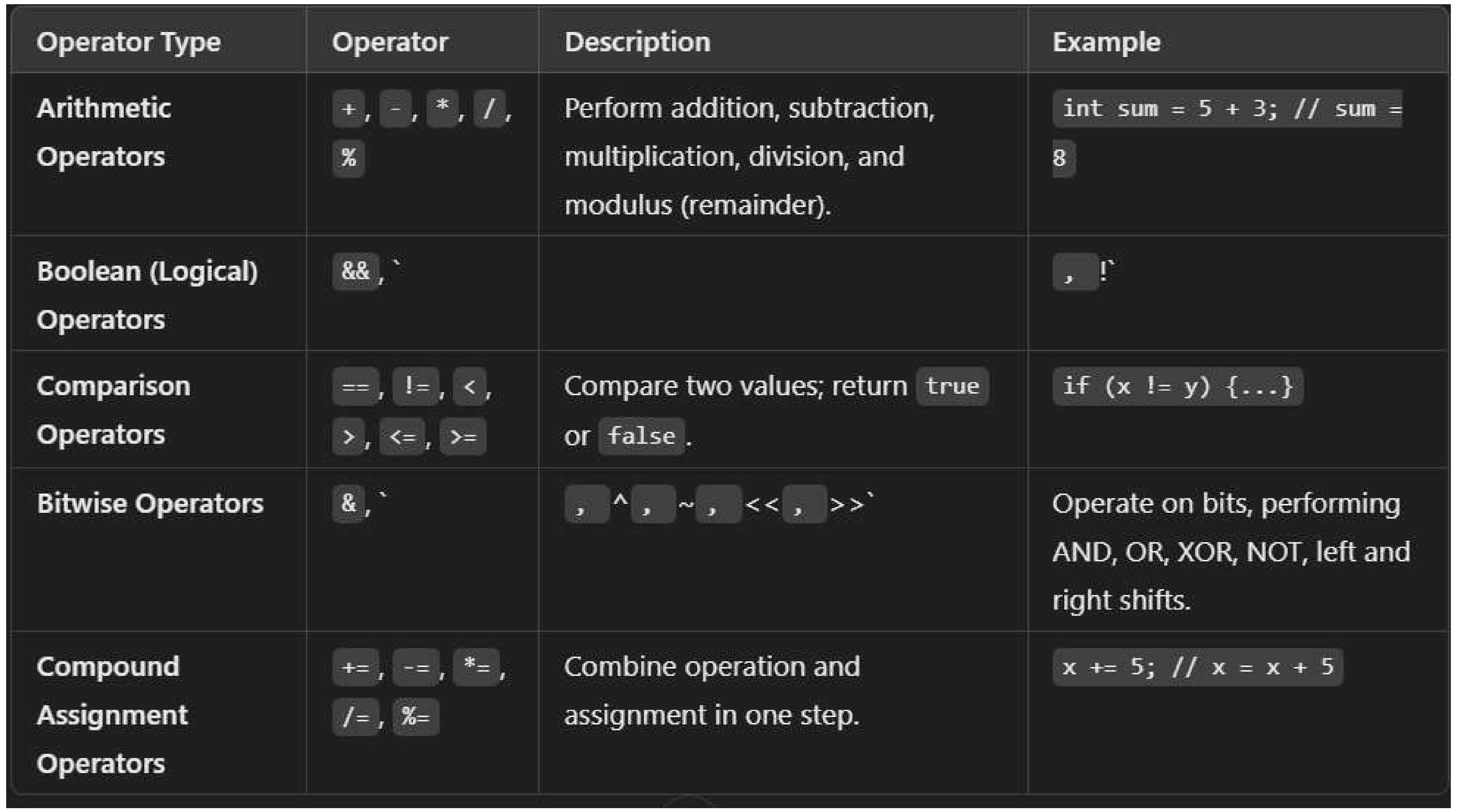


```
int mode = 2;
switch (mode) {
  case 1:
    Serial.println("Mode 1");
    break;
  case 2:
    Serial.println("Mode 2");
    break;
  default:
    Serial.println("Unknown Mode");
}
```

Operator

Operator

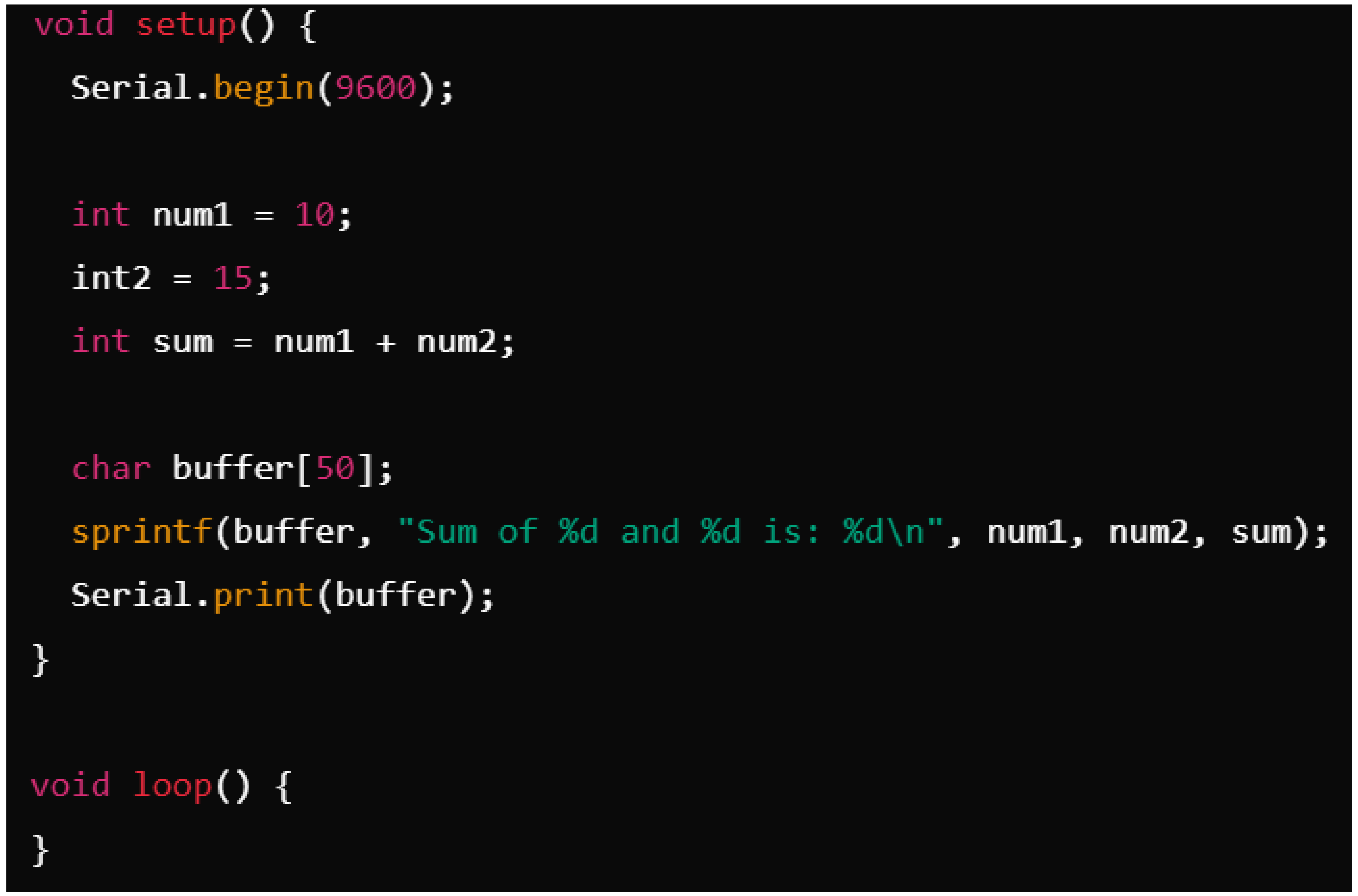
- An operator is a symbol that tells the compiler to perform a specific mathematical, relational, or logical operation on one or more operands.
- Operators are used to manipulate data and variables.



Operator Type	Operator	Description	Example
Arithmetic Operators	`+`, `-`, `*`, `/`, `%`	Perform addition, subtraction, multiplication, division, and modulus (remainder).	`int sum = 5 + 3; // sum = 8`
Boolean (Logical) Operators	`&&`, `		`,` !`
Comparison Operators	`==`, `!=`, `<`, `>`, `<=`, `>=`	Compare two values; return `true` or `false`.	`if (x != y) {...}`
Bitwise Operators	`&`, `	`,` ^ `,` ~ `,` << `,` >>`	Operate on bits, performing AND, OR, XOR, NOT, left and right shifts.
Compound Assignment Operators	`+=`, `-=`, `*=`, `/=`, `%=`	Combine operation and assignment in one step.	`x += 5; // x = x + 5`

Additions in Arduino

Additions in Arduino

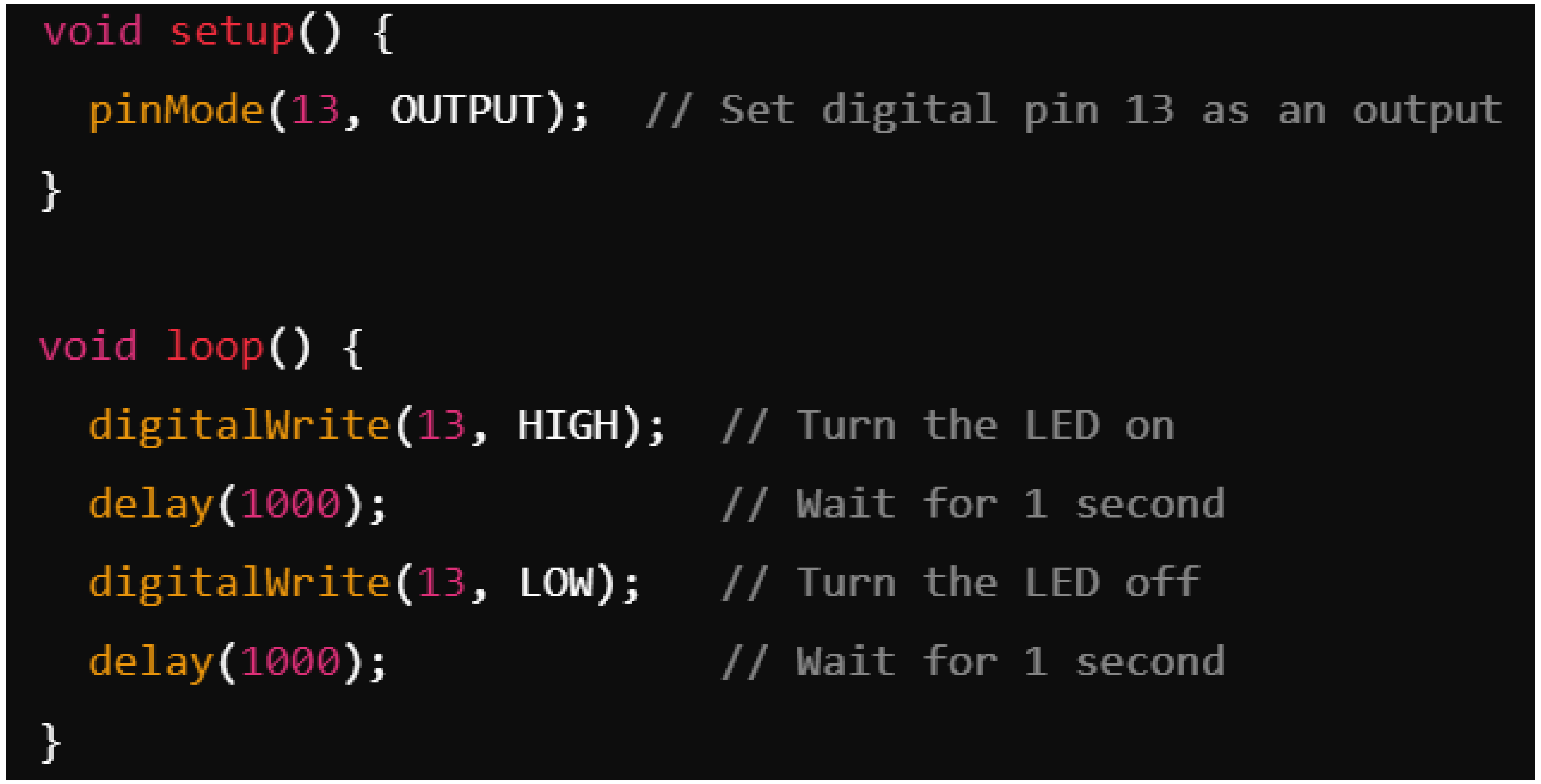


```
void setup() {
  Serial.begin(9600);

  int num1 = 10;
  int2 = 15;
  int sum = num1 + num2;

  char buffer[50];
  sprintf(buffer, "Sum of %d and %d is: %d\n", num1, num2, sum);
  Serial.print(buffer);
}

void loop() {
}
```

Programming the Arduino for IoT

Programming the Arduino for IoT

Blink LED in Arduino



```
void setup() {
  pinMode(13, OUTPUT);  // Set digital pin 13 as an output
}

void loop() {
  digitalWrite(13, HIGH);  // Turn the LED on
  delay(1000);             // Wait for 1 second
  digitalWrite(13, LOW);   // Turn the LED off
  delay(1000);             // Wait for 1 second
}
```

EIOV

Happy Ending!