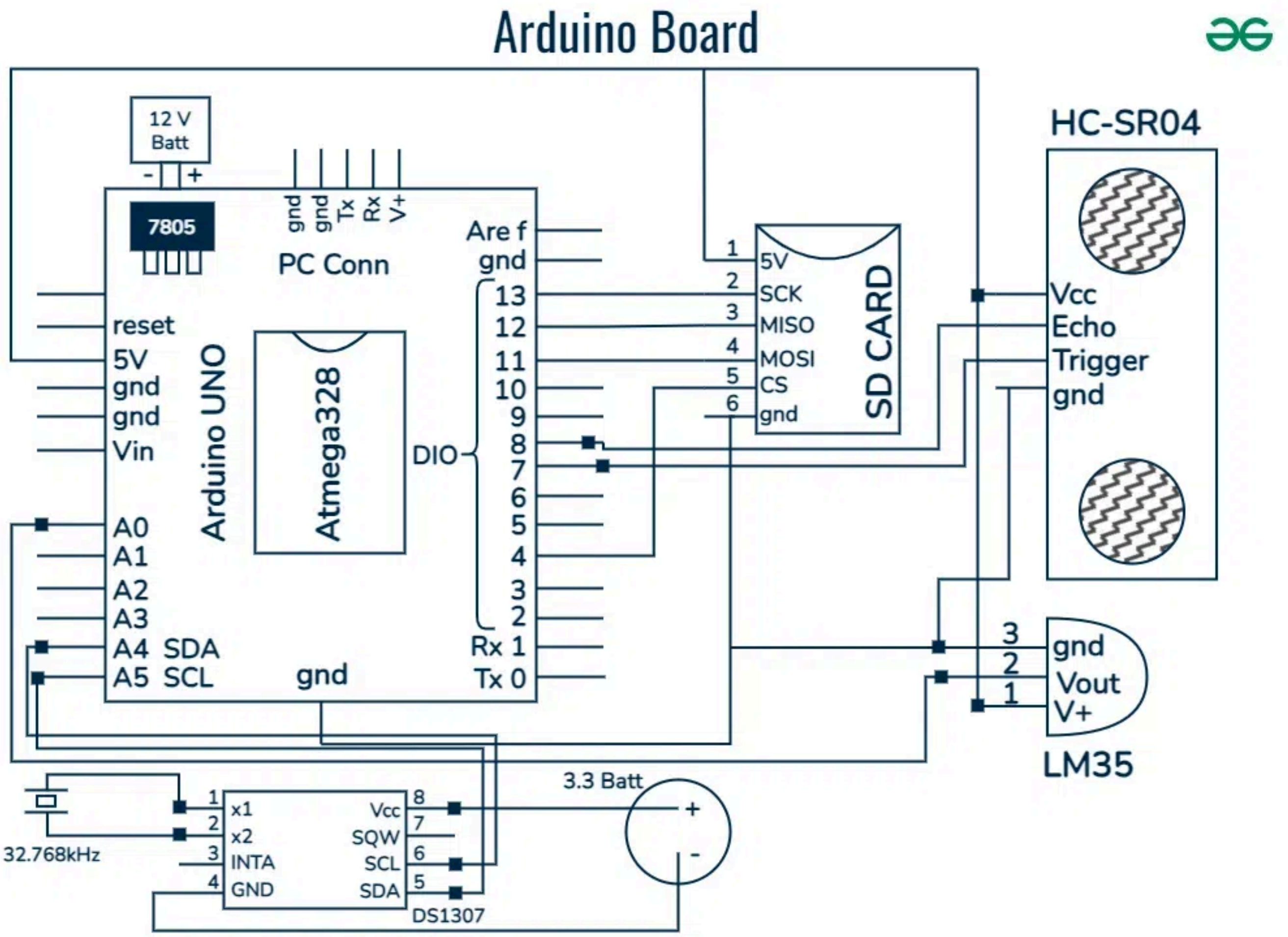
UNIT - 4 IOT Notes

UNIT 4: Arduino Platform, Anatomy, and IDE



1. Programming the Arduino for IoT

- **Popular Platform:** Arduino is a popular open-source platform widely used for developing IoT applications.
- **Accessibility:** It provides a simple and accessible way to program microcontrollers and create interactive projects.

2. Arduino Platform Boards Anatomy [PYQ Q8 (June 2024)]

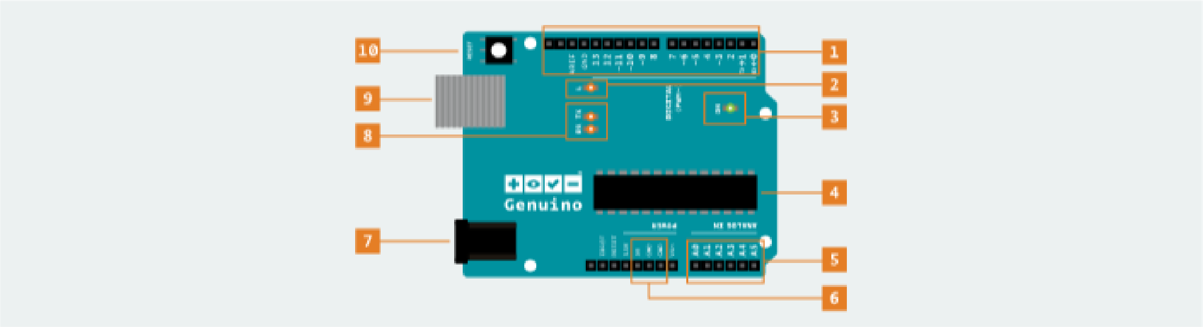
- **Variations:** Arduino boards come in various forms and configurations.
- **Common Components:** They share some common components:
  - **A. Microcontroller:**
    - **Central Component:** The core of an Arduino board; an integrated circuit that can be programmed.
    - **Function:** Executes the code and controls the board's functionality.

        - **Common MCUs:** ATmega328P (e.g., Uno), ATmega2560 (e.g., Mega), ARM Cortex-M (e.g., Due, MKR series).
    - **B. Digital I/O Pins:**
        - **Purpose:** Used to connect sensors, actuators, and other devices.
        - **Configuration:** Can be configured as either inputs or outputs.
        - **Values:** Can read or write digital values (HIGH or LOW).
        - **Special Functions:** Some digital pins support PWM (Pulse Width Modulation), indicated by a tilde (~) symbol. `analogWrite()` works on these PWM pins.
    - **C. Analog Input Pins:**
        - **Purpose:** Allow Arduino boards to read analog voltages from sensors (e.g., temperature sensors, light sensors, potentiometers).
        - **ADC:** These pins have an analog-to-digital converter (ADC) that converts the analog signal (voltage) into a digital value (typically 0-1023 for a 10-bit ADC).
        - **Function:** Use with `analogRead()`.
    - **D. Power Pins:**
        - **Purpose:** Provide regulated voltage to power the microcontroller and connected components.
        - **Common Pins:**
            - **VCC/5V:** Positive voltage supply (typically 5V for Uno).
            - **3.3V:** Lower positive voltage supply.
            - **GND:** Ground pins.
            - **Vin:** Input voltage (can be used to power the board from an external unregulated source).
            - **AREF:** Analog Reference pin.
            - **Reset:** Resets the microcontroller.
            - **IOREF:** Provides voltage reference for shields.
    - **E. Communication Interfaces:**
        - **Purpose:** Allow the Arduino to communicate with other devices, sensors, or modules.
        - **Common Protocols:**
            - **UART (Universal Asynchronous Receiver/Transmitter):** Serial communication (TX/RX pins).
            - **I2C (Inter-Integrated Circuit):** Two-wire interface (SDA/SCL pins).
            - **SPI (Serial Peripheral Interface):** Four-wire interface (MOSI, MISO, SCK, SS pins).
        - These are commonly used in embedded systems and microcontroller-based devices.

  - **Example (Arduino Uno):** A popular board featuring an ATmega328P microcontroller, 14 digital I/O pins, 6 analog input pins, and a USB connection for programming and power.

3. Arduino UNO Board Anatomy (Detailed from Page 3 & 4, Doc: Unit 4) [PYQ Q8 (June 2024)]

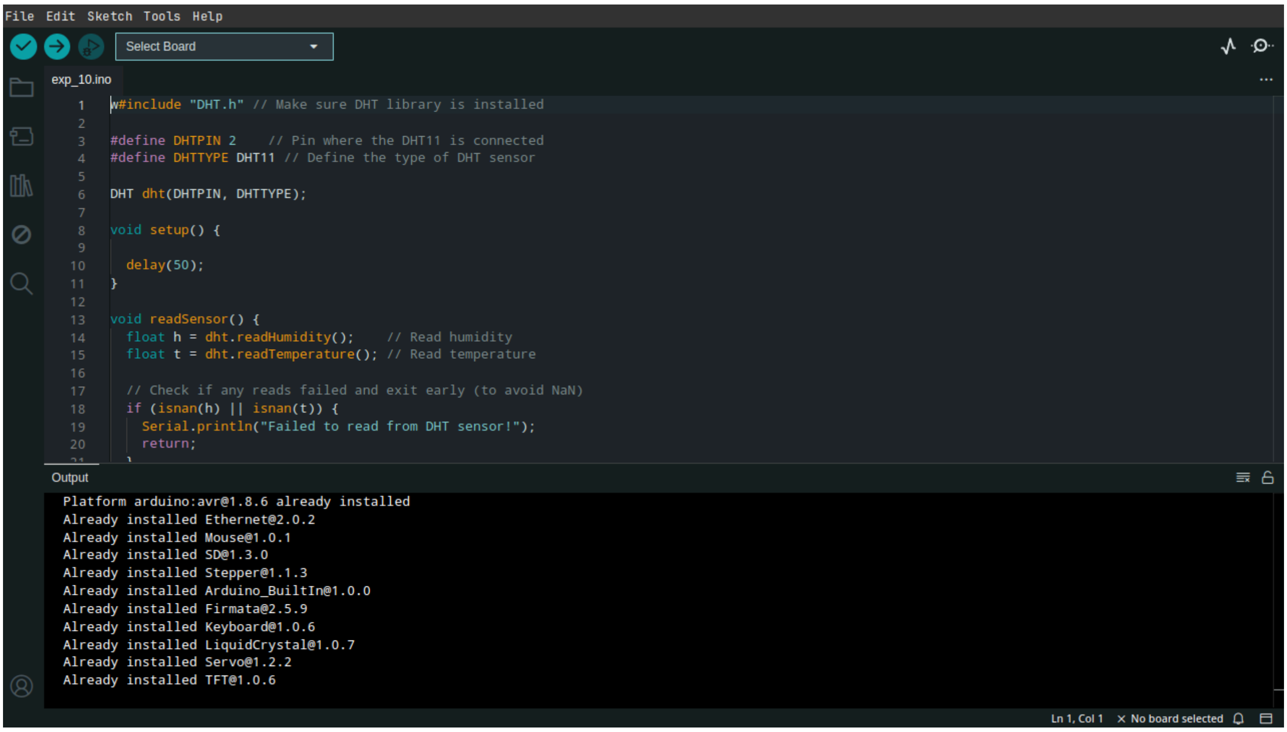
- **Overview:** Arduino boards sense the environment by receiving inputs from sensors and affect surroundings by controlling lights, motors, and other actuators. They are microcontroller development platforms.



- **Specific Components (UNO):**
  - **1. Digital pins (0-13):** Use with `digitalRead()`, `digitalWrite()`. `analogWrite()` works only on pins with the PWM symbol (~3, 5, 6, 9, 10, 11 on Uno).
  - **2. Pin 13 LED:** The only actuator built-in to the board. Handy for first blink sketch and debugging.
  - **3. Power LED:** Indicates Arduino is receiving power. Useful for debugging.
  - **4. ATmega microcontroller (e.g., ATmega328P):** The "heart" or "brain" of the board; processes and executes code, performs logical operations, and controls other components.
  - **5. Analog in (A0-A5):** Use these pins with `analogRead()` to read analog sensor values.
  - **6. GND and 5V pins:** Provide +5V power and ground to circuits. (Also 3.3V pin available).
  - **7. Power connector (Barrel Jack):** How you power your Arduino when not plugged into a USB port. Can accept voltages between 7-12V (recommended).
  - **8. TX and RX LEDs:** Indicate serial communication between Arduino and computer. Flicker during sketch upload and serial communication. Useful for debugging.
  - **9. USB port:** Used for powering Arduino UNO, uploading sketches, and communicating with the sketch (via `Serial.println()`, etc.).
  - **10. Reset button:** Resets the ATmega microcontroller, restarting the program from the beginning. Useful for testing/debugging.
  - **ICSP (In-Circuit Serial Programming) Header:** Allows programming the microcontroller directly (e.g., burning bootloader) or for low-level programming. (Visible on Uno board, mentioned on Page 4).
  - **Voltage Regulator:** Ensures the board receives a stable voltage (typically 5V from USB or external power), protecting components from voltage spikes.
  - **Crystal Oscillator:** Sets the clock speed of the microcontroller (e.g., 16 MHz for most Arduinos), crucial for timing and synchronization of operations.

4. Arduino IDE (Integrated Development Environment) [PYQ Q1d (June 2024 - general IDE concept)]

- **Definition:** The software platform (application) used to write, compile, and upload code (called "sketches") to Arduino boards. It's where you create the instructions that tell the Arduino what to do.
- **Steps to Set Up an Arduino Board using IDE:**
  1. Download and Install the Arduino IDE (from arduino.cc).
  2. Connect the Arduino to Your Computer (via USB).
  3. Open the Arduino IDE.
  4. Select the Arduino Board Model (Tools > Board).
  5. Select the Correct Port (Tools > Port).
  6. Write or Open a Sketch (Code).
  7. Verify (Compile) the Code (√ button).
  8. Upload the Code to the Arduino (→ button).



```
w#include "DHT.h" // Make sure DHT library is installed

#define DHTPIN 2    // Pin where the DHT11 is connected
#define DHTTYPE DHT11 // Define the type of DHT sensor

DHT dht(DHTPIN, DHTTYPE);

void setup() {

  delay(50);
}

void readSensor() {
  float h = dht.readHumidity();    // Read humidity
  float t = dht.readTemperature(); // Read temperature

  // Check if any reads failed and exit early (to avoid NaN)
  if (isnan(h) || isnan(t)) {
    Serial.println("Failed to read from DHT sensor!");
    return;
```

```
Platform arduino:avr@1.8.6 already installed
Already installed Ethernet@2.0.2
Already installed Mouse@1.0.1
Already installed SD@1.3.0
Already installed Stepper@1.1.3
Already installed Arduino_BuiltIn@1.0.0
Already installed Firmata@2.5.9
Already installed Keyboard@1.0.6
Already installed LiquidCrystal@1.0.7
Already installed Servo@1.2.2
Already installed TFT@1.0.6
```

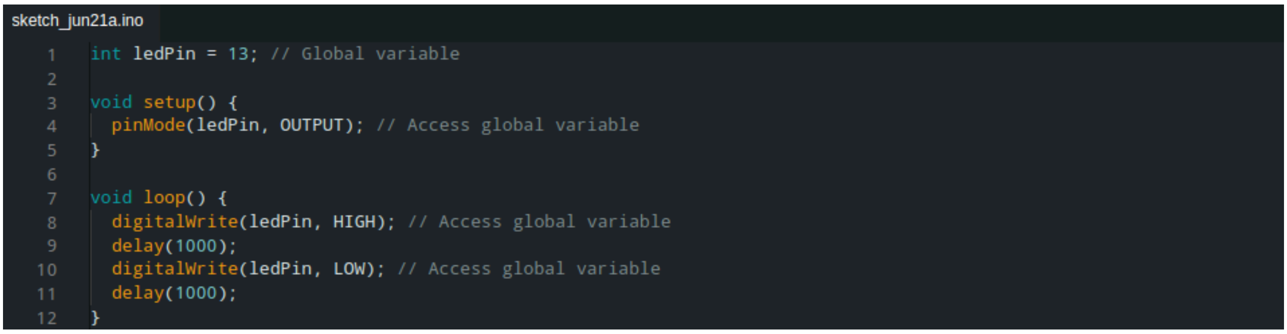
5. Arduino Coding Syntax (Basic Structure)

- `void setup() { ... }`:
  - This function runs once when the Arduino board starts up or is reset.
  - Used for initializing settings, pin modes, libraries, serial communication, etc.
  - *Example:* `pinMode(13, OUTPUT);` // Sets pin 13 as an output.
- `void loop() { ... }`:
  - This function runs repeatedly and continuously after `setup()` has finished.
  - Contains the main logic of the program that the Arduino will execute over and over.
  - *Example:*

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

6. Variables and Scope in Arduino

- **Variable:** A named storage location in memory that holds a value, which can be modified as the program runs. Used to store and manipulate data like sensor readings or control states.
    - *Example:* `int ledPin = 13;`
- **Scope:** Determines where a variable can be accessed in the code.
    - **Global variables:** Declared outside any function (including `setup()` and `loop()`). Can be accessed anywhere in the code.
    - **Local variables:** Declared within a function (like `setup()` or `loop()`) or a block of code. Can only be used (accessed) within that specific function or block.

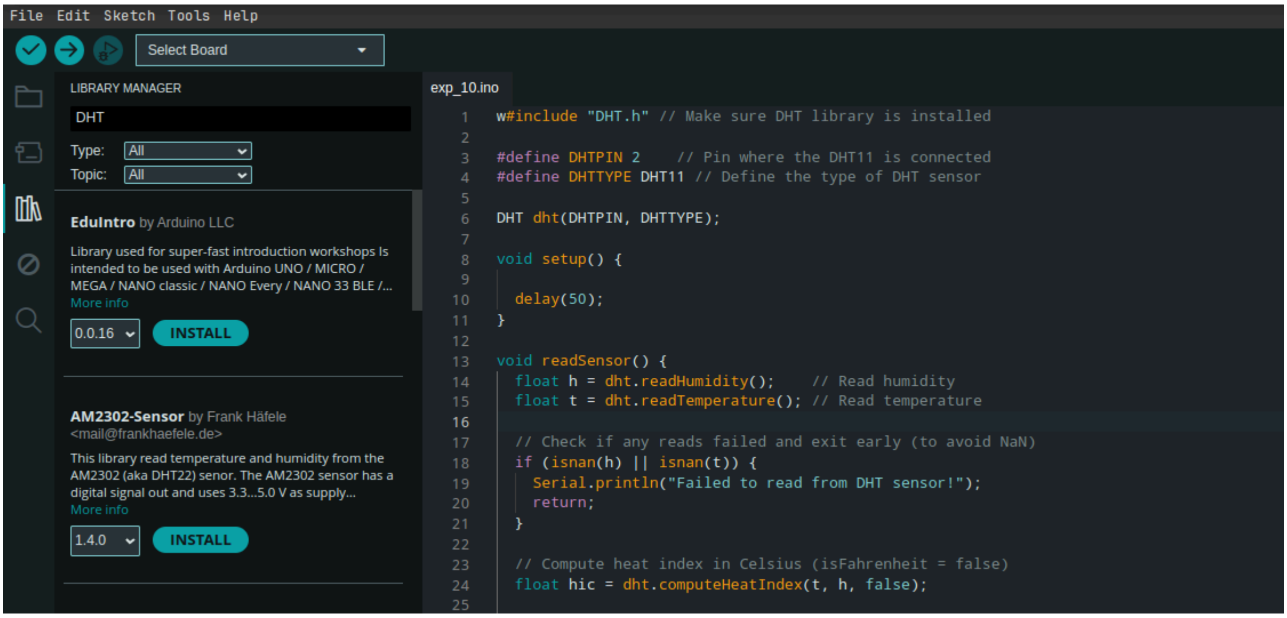


```
int ledPin = 13; // Global variable

void setup() {
  pinMode(ledPin, OUTPUT); // Access global variable
}

void loop() {
  digitalWrite(ledPin, HIGH); // Access global variable
  delay(1000);
  digitalWrite(ledPin, LOW); // Access global variable
  delay(1000);
}
```

- **Data Types:**
    - **`int` (integer):** For whole numbers (e.g., 10, -32768 to 32767 on most Arduinos). *Uses:* Counters, pin numbers, integer sensor readings.
    - **`float` (floating point):** For decimal numbers (e.g., 3.14, single-precision, 6-7 significant digits). *Uses:* Temperature, voltage, distance measurements.
    - **`char` (character):** For single characters (e.g., 'A', '3', stored as 8-bit ASCII). *Uses:* Handling single characters, simple text.
    - **`boolean` (boolean):** For true/false values (represented by 1 or 0). *Uses:* Condition checks, toggling switches.
    - **`String`:** Allows for handling sequences of characters (text strings). *Uses:* Displaying messages, serial communication.
    - **`long`:** Holds larger integer values than `int` (-2,147,483,648 to 2,147,483,647). *Uses:* Long-distance measurements, timer values.
    - **`unsigned int`:** Stores whole numbers from 0 to 65,535 (no negatives).
    - **`unsigned long`:** Stores whole numbers from 0 to 4,294,967,295. Ideal for very large positive values (e.g., `millis()`).

7. Function Libraries in Arduino

- **Definition:** Collections of pre-written functions that simplify complex tasks and allow use of additional hardware or specialized actions without writing everything from scratch.
- **Benefits:** Save time, make coding more efficient by providing reusable, well-documented, and tested code.
- **Key Points about Arduino Libraries:**
  - **Purpose:** Simplify complex tasks by providing pre-written code.
  - **Usage:** Include at the beginning of your code with `#include <libraryName.h>`.
    - *Example:* `#include <Servo.h>`
  - **Structure:** Have organized files with functions for specific hardware or tasks.
  - **Common Libraries:** `Wire` (for I2C), `SPI` (for SPI communication), `WiFi`, `Servo`.
  - **Installing Libraries:** Can be added through the Arduino IDE's Library Manager (Sketch > Include Library > Manage Libraries...).



DHT

Type: All

Topic: All

EduIntro by Arduino LLC

Library used for super-fast introduction workshops Is intended to be used with Arduino UNO / MICRO / MEGA / NANO classic / NANO Every / NANO 33 BLE /...

More info

0.0.16 INSTALL

AM2302-Sensor by Frank Häfele <mail@frankhaefele.de>

This library read temperature and humidity from the AM2302 (aka DHT22) senor. The AM2302 sensor has a digital signal out and uses 3.3...5.0 V as supply...

More info

1.4.0 INSTALL

```
w#include "DHT.h" // Make sure DHT library is installed

#define DHTPIN 2    // Pin where the DHT11 is connected
#define DHTTYPE DHT11 // Define the type of DHT sensor

DHT dht(DHTPIN, DHTTYPE);

void setup() {

  delay(50);
}

void readSensor() {
  float h = dht.readHumidity();    // Read humidity
  float t = dht.readTemperature(); // Read temperature

  // Check if any reads failed and exit early (to avoid NaN)
  if (isnan(h) || isnan(t)) {
    Serial.println("Failed to read from DHT sensor!");
    return;
  }

  // Compute heat index in Celsius (isFahrenheit = false)
  float hic = dht.computeHeatIndex(t, h, false);

```

8. Arduino Emulator

- **Definition:** A software tool that allows simulation of an Arduino board on a computer without needing physical hardware.
- **Usefulness:** Helpful for testing and debugging code in a virtual environment, especially without access to a specific Arduino board or component.
- **Key Points:**
  - **Purpose:** Simulate Arduino hardware and code execution.
  - **Features:** Virtual components (LEDs, sensors, motors), serial monitor simulation, code debugging capabilities.
  - **Popular Arduino Emulators:** Tinkercad Circuits, Proteus Design Suite, SimulIDE, Wokwi.
  - **Benefits:** Cost-effective (no hardware purchase), Time-saving (quick setup and testing), Risk-free (no risk of damaging physical components).

9. Decision Making Statements (if, else) in Arduino

- **Purpose:** Used to control the flow of a program based on certain conditions. Allow the Arduino to make decisions and execute specific blocks of code only when specific conditions are met.
- **Types:**
  - **1. `if` Statement:** Executes code block if the condition is true.

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

    - *Example (from Page 9):*

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

  - **2. `if-else` Statement:** Executes one block of code if the condition is true, and another block if it's false.

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

    - *Example (from Page 9):*

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

  - **3. `if-else if-else` Statement:** Allows checking multiple conditions sequentially.

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

    - *Example (from Page 10):*

```
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:** Selects one of many code blocks to be executed based on the value of an expression (variable).

```
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
    break; // Optional for default
}
```

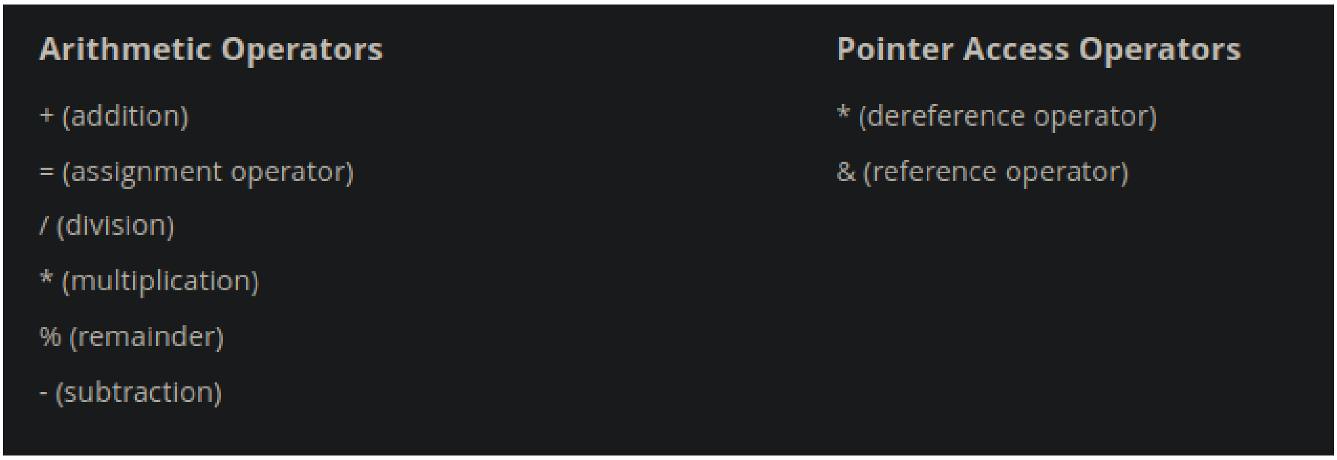
  - *Example (from Page 10):*

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

10. Operators in Arduino

- **Definition:** A symbol that tells the compiler to perform a specific mathematical or logical operation on one or more operands (values or variables).

- **Purpose:** Used to manipulate data and variables.



Arithmetic Operators

+ (addition)

= (assignment operator)

/ (division)

* (multiplication)

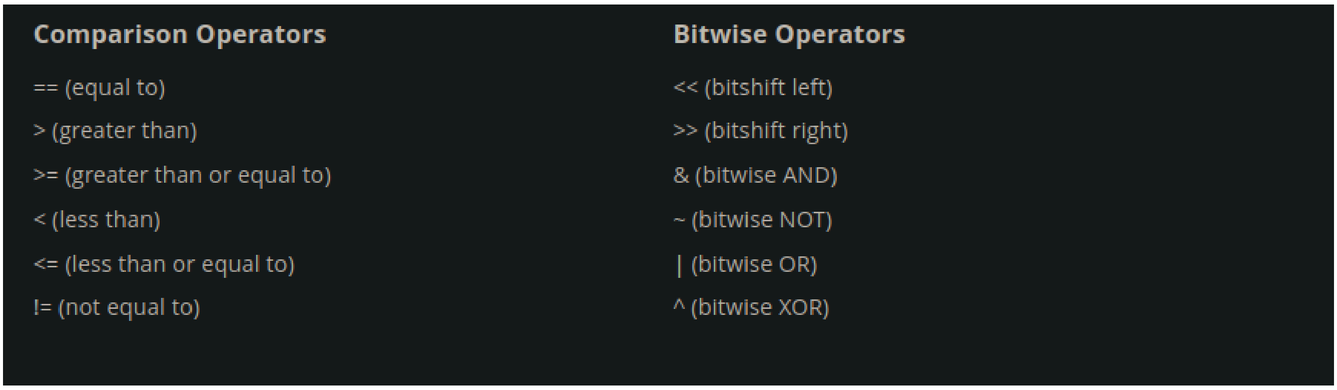
% (remainder)

- (subtraction)

Pointer Access Operators

* (dereference operator)

& (reference operator)



Comparison Operators

== (equal to)

> (greater than)

>= (greater than or equal to)

< (less than)

<= (less than or equal to)

!= (not equal to)

Bitwise Operators

<< (bitshift left)

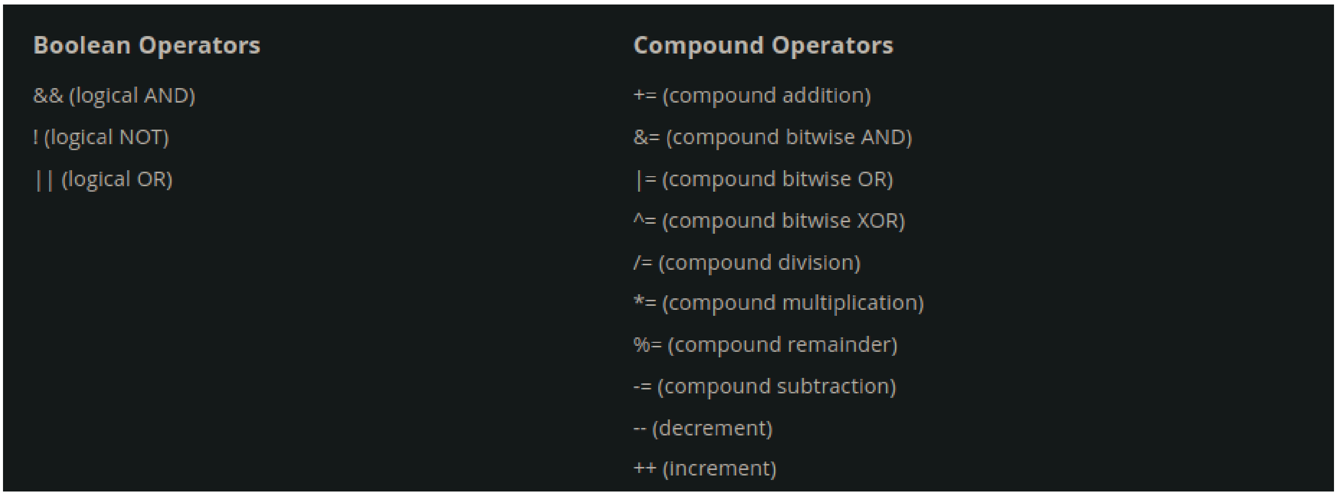
>> (bitshift right)

& (bitwise AND)

~ (bitwise NOT)

| (bitwise OR)

^ (bitwise XOR)



Boolean Operators

&& (logical AND)

! (logical NOT)

| | (logical OR)

Compound Operators

+= (compound addition)

&= (compound bitwise AND)

|= (compound bitwise OR)

^= (compound bitwise XOR)

/= (compound division)

*= (compound multiplication)

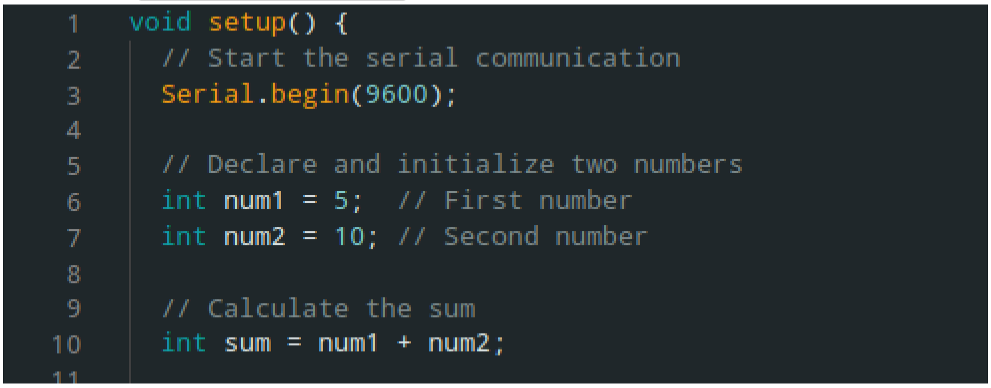
%= (compound remainder)

-= (compound subtraction)

-- (decrement)

++ (increment)

- **Types of Operators:**
  - **Arithmetic Operators:** `+` (addition), `-` (subtraction), `*` (multiplication), `/` (division), `%` (modulus/remainder).
    - *Example:* `int sum = 5 + 3;`



```
void setup() {
  // Start the serial communication
  Serial.begin(9600);

  // Declare and initialize two numbers
  int num1 = 5;  // First number
  int num2 = 10; // Second number

  // Calculate the sum
  int sum = num1 + num2;
```

    - **Boolean (Logical) Operators:** `&&` (logical AND), `||` (logical OR), `!` (logical NOT).
    - **Comparison (Relational) Operators:** `==` (equal to), `!=` (not equal to), `>` (greater than), `<` (less than), `>=` (greater than or equal to), `<=` (less than or equal to). Return `true` or `false`.
        - *Example:* `if (x == y) { ... }`
    - **Bitwise Operators:** `&` (bitwise AND), `|` (bitwise OR), `^` (bitwise XOR), `~` (bitwise NOT), `<<` (left shift), `>>` (right shift). Operate on individual bits of operands.
    - **Compound Assignment Operators:** Combine an arithmetic/bitwise operation with an assignment. E.g., `+=`, `-=`, `*=`, `/=`, `%=`, `&=`, `|=`, `^=`, `<<=`, `>>=`.
        - *Example:* `x += 5;` (equivalent to `x = x + 5;`)

11. Example: Blink LED in Arduino (Programming for IoT context)

- **Purpose:** A fundamental "hello world" program for microcontrollers, demonstrating basic output control.

```
void setup() {
  pinMode(13, OUTPUT); // Set digital pin 13 (often connected to an onboard LED) as an output.
}

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