Unit-2: Introduction

Integrated Development Environment(Ide) And Programming: Basics of Embedded C Programming, Data Types, Arithmetic & Logical Operators, Loops, Functions, #define Macros, Structures (Declaration and Accessing data members). Integrated Development Environment tools: Editor, Compiler, Linker, Loader, Debugger (Definitions only). Practice: Working with Arduino IDE(Simple programs on Operators, Loops and Functions).

Integrated Development Environment(Ide) and Programming:

- An Integrated Development Environment (IDE) is a software application that provides a comprehensive set of tools and features for writing, testing, and debugging code.
- When it comes to Arduino programming, there are specific IDEs tailored for working with Arduino boards.
- The official Arduino IDE is the most commonly used IDE for Arduino development, but there are also alternative IDEs available, such as PlatformIO and Visual Studio Code with Arduino extensions.

Key components and functionalities of an Arduino IDE:

- **Code Editor**: The IDE provides a code editor where you can write your Arduino programs. It offers features like syntax highlighting, auto-completion, and indentation to assist you in writing clean and error-free code. The code editor is where you write the instructions that control the behavior of your Arduino board.

- **Library Manager**: Arduino libraries are collections of pre-written code that provide additional functionality and simplify complex tasks. The IDE includes a library manager that allows you to easily search, install, and manage libraries required for your project. Libraries provide ready-to-use functions for tasks like controlling LEDs, reading sensors, or communicating with other devices.

- **Compiler and Uploader**: Once you have written your Arduino code, the IDE compiles it into machine-readable instructions that can be understood by the Arduino board's microcontroller. The compiler checks for syntax errors and other issues in your code. After successful compilation, the IDE uploads the compiled code to the Arduino board, making it ready to run.

Key components and functionalities of an Arduino IDE: (Cont.)

- **Serial Monitor:** The IDE provides a serial monitor tool that allows you to communicate with your Arduino board through the serial port. It displays the data sent by the Arduino and allows you to send commands or instructions from the computer to the board. The serial monitor is useful for debugging and monitoring the behavior of your Arduino programs.

- **Integrated Examples:** Arduino IDE comes with a set of pre-built examples that demonstrate various functionalities of the Arduino board and its peripherals. These examples can be accessed from the IDE's menu and serve as a starting point for learning and building your own projects.

Basics of Embedded C Programming

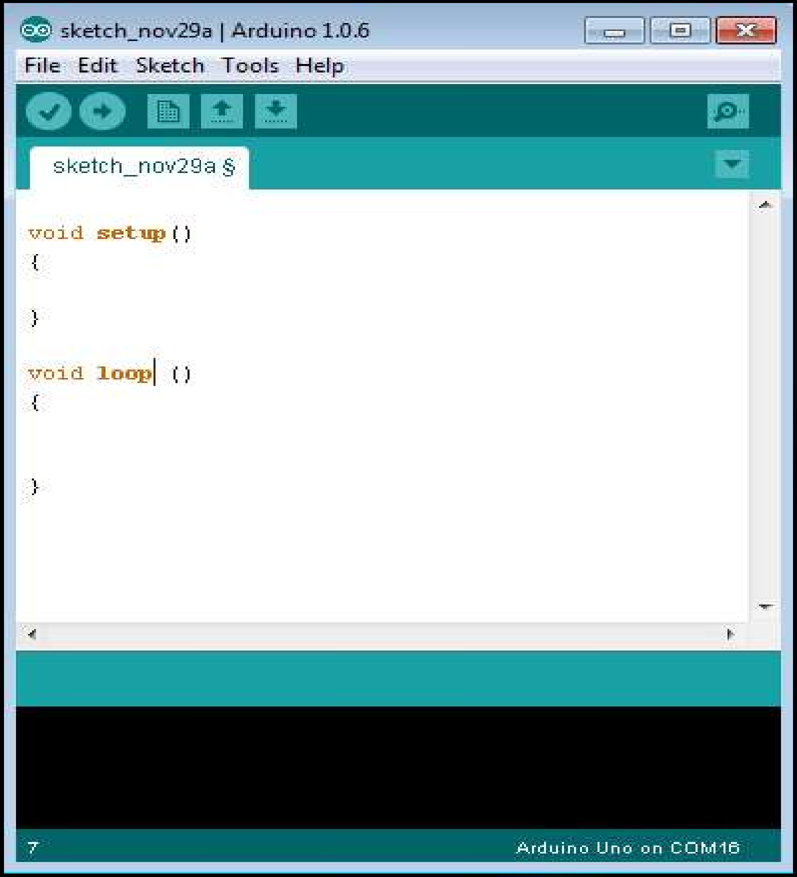
- Embedded C programming with respect to Arduino involves writing code in the C programming language to control and interact with Arduino boards.

- Arduino uses a simplified version of C/C++ that provides libraries and functions specifically designed for the Arduino platform.

- **Sketch** – The first new terminology is the Arduino program called “sketch”.

Structure of an Arduino Program:

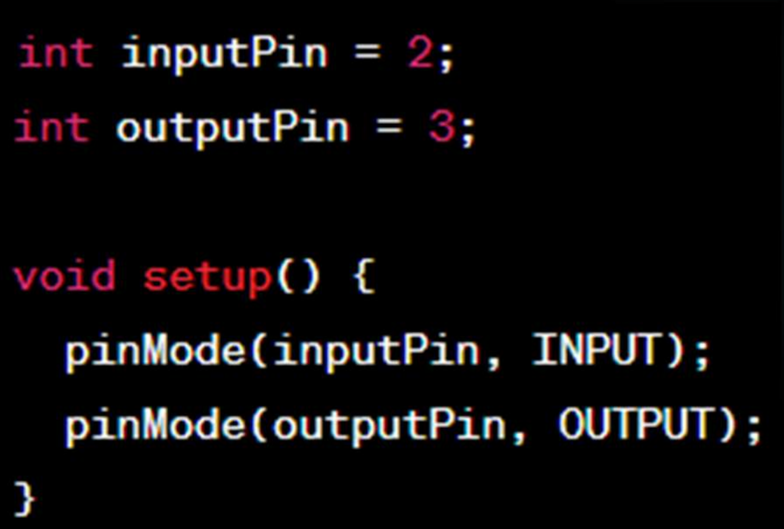
- Every Arduino program must have two essential functions:
  - **setup() and loop().**
- The **setup()** function is called once when the Arduino board starts.
- It is used for initializing variables, setting pin modes, and configuring any necessary settings.
- The **loop()** function is called repeatedly after the setup() function.
- It contains the main logic of your program that will execute in a continuous loop until the board is powered off.



```
void setup()
{

}

void loop()
{


}
```

Pin Definitions:

- Arduino boards have a set of **digital input/output (I/O) pins and analog input pins.**
- **Digital pins** can be used for both input and output operations. They are typically labeled with numbers (e.g., 2, 3, 4, etc.).
- **Analog input** pins can read analog values from sensors or other analog devices. They are labeled with an 'A' followed by a number (e.g., A0, A1, A2, etc.).
- **Pin mode** should be set using pinMode() function as either INPUT or OUTPUT before using the pin for reading or writing operations.

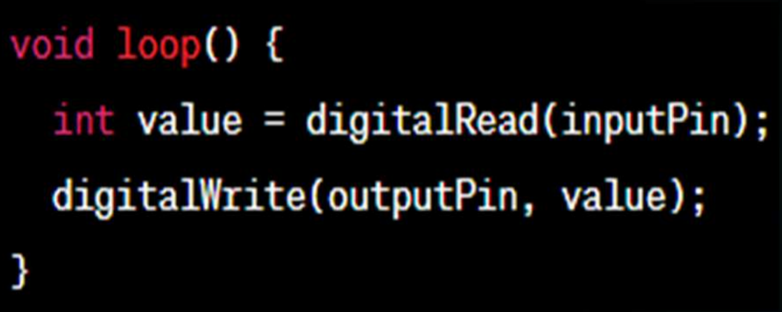


```
int inputPin = 2;
int outputPin = 3;

void setup() {
  pinMode(inputPin, INPUT);
  pinMode(outputPin, OUTPUT);
}
```

Digital Input/Output:

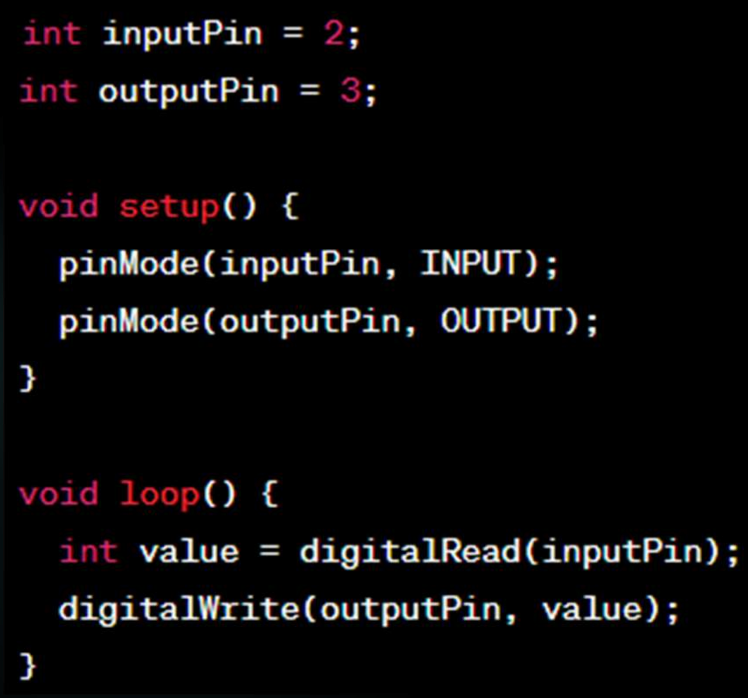
- To read from a digital pin, you can use the digitalRead() function, which returns either HIGH or LOW.

- To write to a digital pin, use the digitalWrite() function. You can set the pin to HIGH or LOW.



```
void loop() {
  int value = digitalRead(inputPin);
  digitalWrite(outputPin, value);
}
```

Summarize the working of the following code:

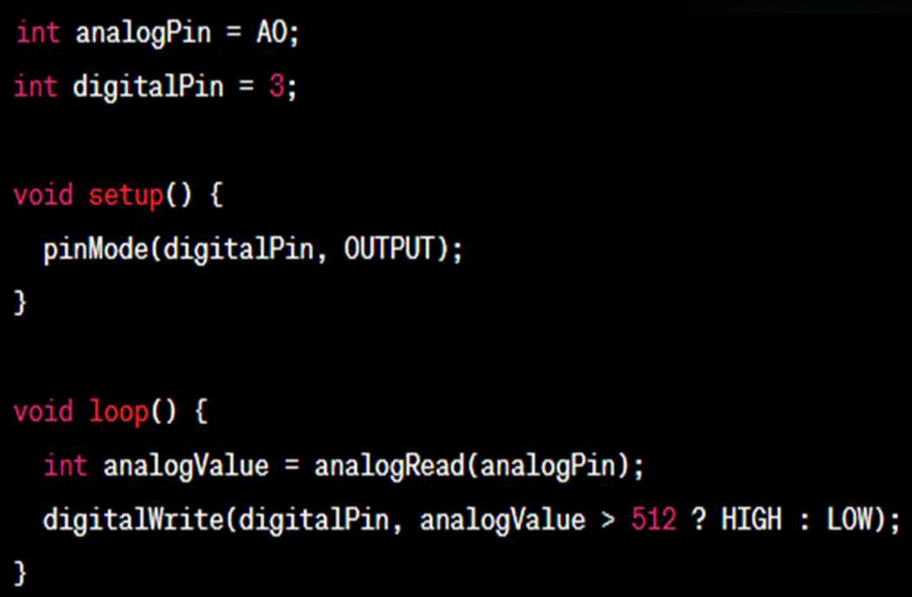


```
int inputPin = 2;
int outputPin = 3;

void setup() {
  pinMode(inputPin, INPUT);
  pinMode(outputPin, OUTPUT);
}

void loop() {
  int value = digitalRead(inputPin);
  digitalWrite(outputPin, value);
}
```

Summary:

Analog Input:

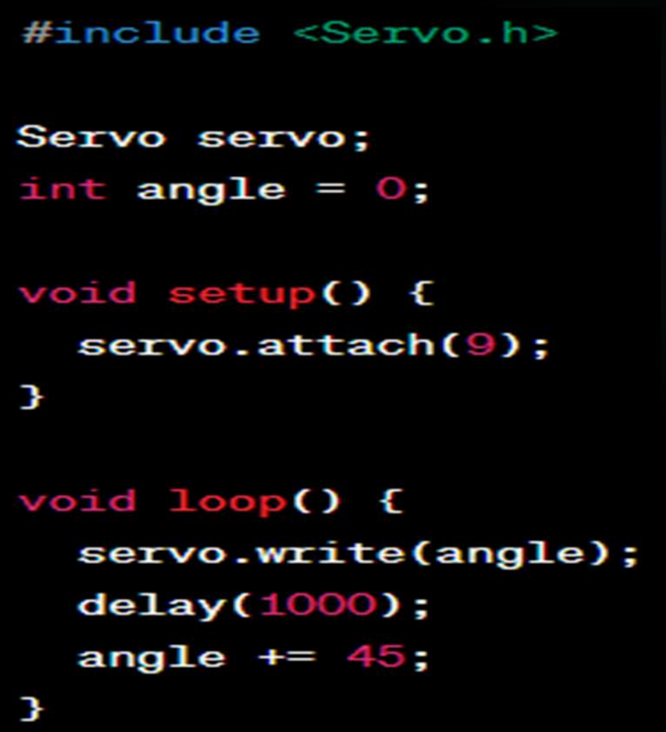
- To read analog values from an analog input pin, use the analogRead() function. It returns a value between 0 and 1023.
- For example, to read from analog pin A0 and write the value to a digital pin:



```
int analogPin = A0;
int digitalPin = 3;

void setup() {
  pinMode(digitalPin, OUTPUT);
}

void loop() {
  int analogValue = analogRead(analogPin);
  digitalWrite(digitalPin, analogValue > 512 ? HIGH : LOW);
}
```

Functions and Libraries:

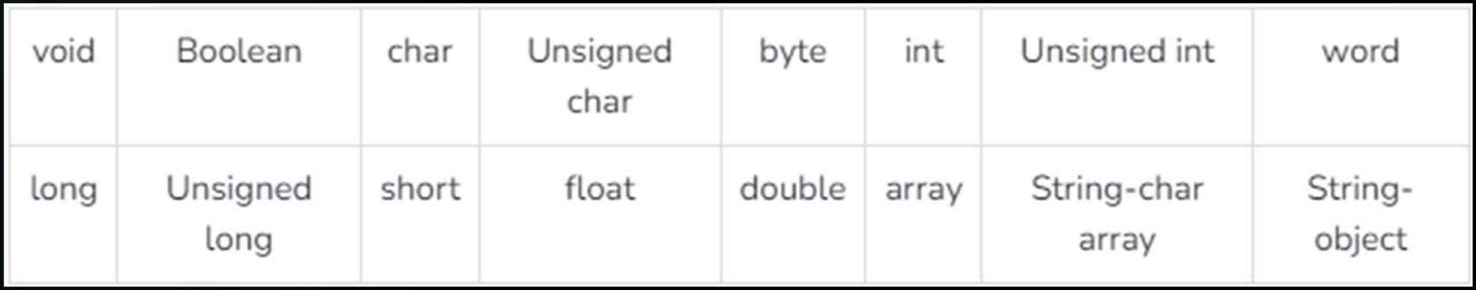
- Arduino provides various built-in functions and libraries that simplify common tasks.
- Examples include delay() for introducing delays, Serial library for serial communication, and libraries for specific sensors or devices.
- Libraries can be included using the #include directive at the beginning of the program.
- For example, to use the Servo library for controlling a servo motor:



```
#include <Servo.h>

Servo servo;
int angle = 0;

void setup() {
  servo.attach(9);
}

void loop() {
  servo.write(angle);
  delay(1000);
  angle += 45;
}
```

Data Types

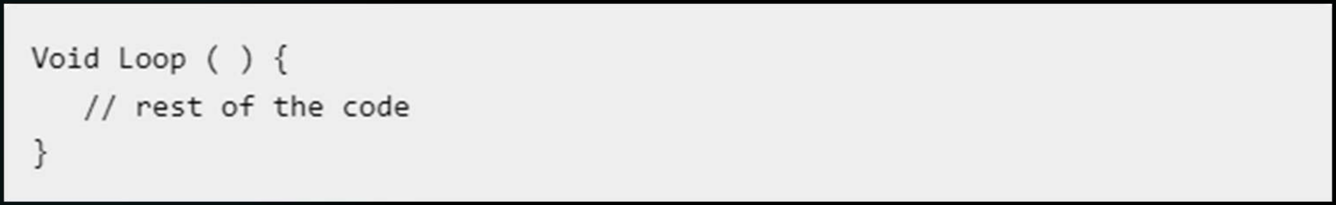
- Data types in C refers to an extensive system used for declaring variables or functions of different types. The type of a variable determines how much space it occupies in the storage and how the bit pattern stored is interpreted.
- The following table provides all the data types that you will use during Arduino programming.



void	Boolean	char	Unsigned char	byte	int	Unsigned int	word
long	Unsigned long	short	float	double	array	String-char array	String-object

void

- The void keyword is used only in function declarations.
- It indicates that the function is expected to return no information to the function from which it was called.



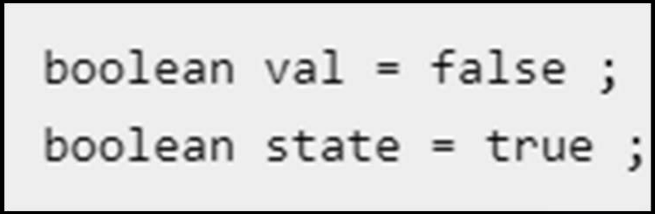
```
Void Loop ( ) {
   // rest of the code
}
```

Boolean

- A Boolean holds one of two values, true or false.
- Each Boolean variable occupies one byte of memory.

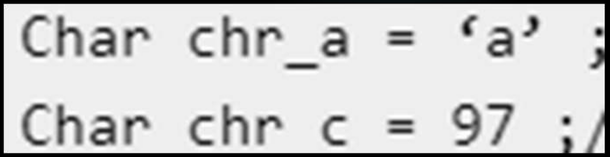
boolean val = false ; // declaration of variable ***"val"*** with type boolean and initialize it with false

boolean state = true ; // declaration of variable ***"state"*** with type boolean and initialize it with true



```
boolean val = false ;
boolean state = true ;
```

Char/Signed Char



```
Char chr_a = 'a' ;
Char chr c = 97 ;/
```

- A data type that takes up one byte of memory that stores a character value.
- Character literals are written in single quotes like this: **'A'** and for multiple characters, strings use double quotes**: *"ABC“.***
- The range of char is typically from -128 to 127.
- When used for arithmetic operations, char behaves as a signed data type.

Char chr_a = ‘a’ ;//declaration of variable “***char_a”*** with type char and initialize it with character a.

Char chr_c = 97 ;//declaration of variable ”***chr_c”*** with type char and initialize it with character 97.

Unsigned chr_h= -97 ;//declaration of variable ”***chr_h”*** with type signed char and initialize it with numeric number -97.

unsigned char

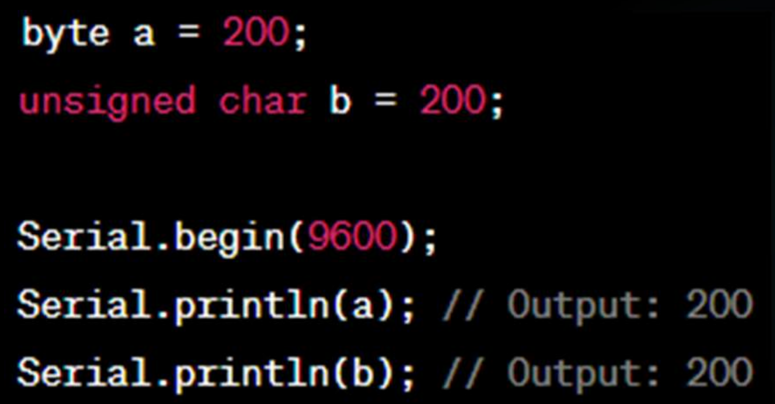
- It is also a 1-byte (8-bit) data type.
- It is explicitly specified as an unsigned data type, meaning it can only represent positive values.
- The range of unsigned char is typically from 0 to 255.
- When used for arithmetic operations, unsigned char behaves as an unsigned data type, allowing only non-negative results.

Unsigned Char chr_y = 121 ; // declaration of variable ***chr_y*** with type Unsigned char and initialize it with non negative number 121.

```
Unsigned Char chr_y = 121 ;
```

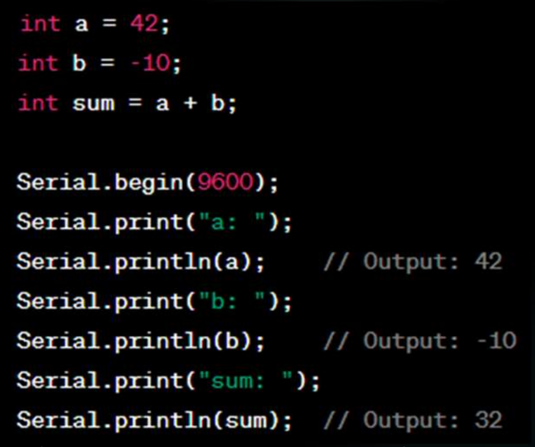
byte

- byte is an unsigned 8-bit data type.
- It can store values from 0 to 255.
- byte is specifically defined in the Arduino language and is often used to represent raw binary data or when you need to work with individual bits.



```
byte a = 200;
unsigned char b = 200;

Serial.begin(9600);
Serial.println(a); // Output: 200
Serial.println(b); // Output: 200
```

int



```
int a = 42;
int b = -10;
int sum = a + b;

Serial.begin(9600);
Serial.print("a: ");
Serial.println(a);    // Output: 42
Serial.print("b: ");
Serial.println(b);    // Output: -10
Serial.print("sum: ");
Serial.println(sum);  // Output: 32
```

Size and Range:

- The int data type is a signed 16-bit integer.
- It can store values from -32,768 to 32,767.
- The exact size of an int may vary depending on the Arduino board you are using. In most cases, it is a 16-bit signed integer.

Usage:

- The int data type is commonly used when you need to work with integer values within the range mentioned above.
- It is suitable for most general-purpose integer calculations and variables.
- int is the default data type for integer literals unless explicitly specified otherwise.

Signedness:

- The int data type is signed, which means it can represent both positive and negative values.
- If you need to work with only positive values, you can use the unsigned int data type, which extends the range to 0 to 65,535.Memory Usage:
- The int data type requires 2 bytes (16 bits) of memory to store the value.

Unsigned int

Unsigned int data type is used to store unsigned integer values. Here are some key points about the unsigned int data type:

Size and Range:

- The unsigned int data type is an unsigned 16-bit integer. It can store values from 0 to 65,535.
- Like the int data type, the size of an unsigned int may vary depending on the Arduino board you are using. In most cases, it is a 16-bit unsigned integer.

Usage:

- The unsigned int data type is commonly used when you only need to work with positive integer values within the specified range.
- It is suitable for scenarios where negative values are not required, such as representing sensor readings, loop counters, or array indices.

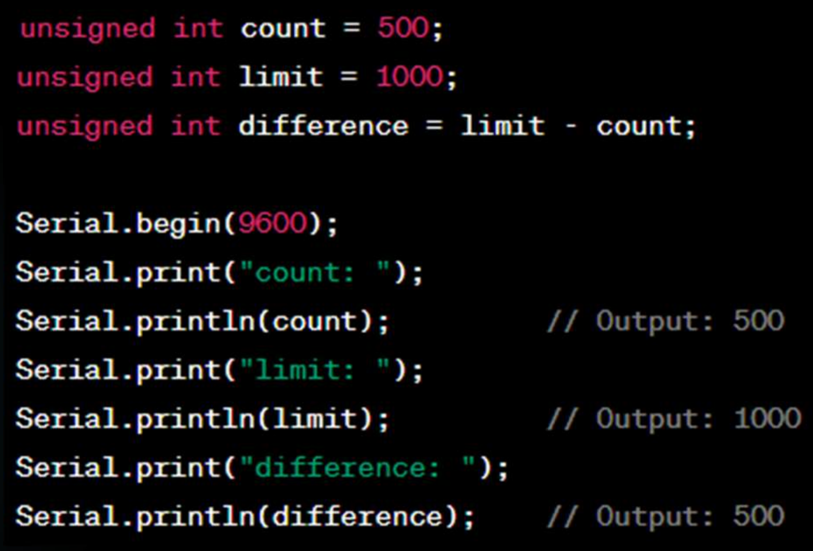
Signedness:

- Unlike the int data type, unsigned int is an unsigned data type and can only represent non-negative values.It cannot store negative values.
- If you need to work with both positive and negative values, you should use the int data type instead.

Memory Usage:

- The unsigned int data type requires 2 bytes (16 bits) of memory to store the value, similar to int.

Unsigned int (Cnt.)



```
unsigned int count = 500;
unsigned int limit = 1000;
unsigned int difference = limit - count;

Serial.begin(9600);
Serial.print("count: ");
Serial.println(count);          // Output: 500
Serial.print("limit: ");
Serial.println(limit);          // Output: 1000
Serial.print("difference: ");
Serial.println(difference);     // Output: 500
```

Arithmetic & Logical Operators

- These operators can be used to manipulate variables, control flow statements, and perform various calculations and logical operations in your Arduino code. Keep in mind the data types and the limitations of the microcontroller you are using to avoid overflow or unexpected behavior.

Arithmetic Operators:

- Addition: +
- Subtraction: -
- Multiplication: *
- Division: /
- Modulus (remainder): %

Logical Operators:

- AND: &&
- OR: | |
- NOT: !

Comparison Operators:

- Equal to: ==
- Not equal to: !=
- Greater than: >
- Less than: <
- Greater than or equal to: >=
- Less than or equal to: <=

Assignment Operators:

- Assignment: =
- Addition assignment: +=
- Subtraction assignment: -=
- Multiplication assignment: *=
- Division assignment: /=
- Modulus assignment: %=

Bitwise Operators:

- Bitwise AND: &
- Bitwise OR: |
- Bitwise XOR: ^
- Bitwise NOT: ~
- Left shift: <<
- Right shift: >>

Loops

- In embedded systems programming for Arduino, loops are used to repeat a block of code multiple times.
- Here are two commonly used loops: the ***for loop*** and the ***while loop***.

for Loop

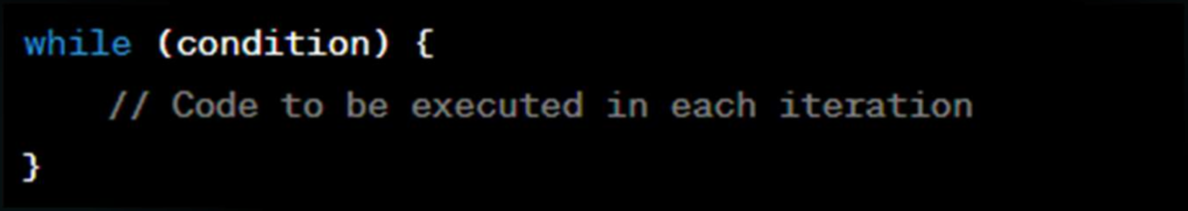
```
for (initialization; condition; increment) {
    // Code to be executed in each iteration
}
```

- Printing numbers from 1 to 5 using a for loop.

```
for (int i = 1; i <= 5; i++) {
    Serial.println(i);
    delay(1000); // Delay for 1 second
}
```

while Loop

- The while loop is useful when the number of iterations is not known in advance, and the loop continues until a certain condition is met.



```
while (condition) {
    // Code to be executed in each iteration
}
```