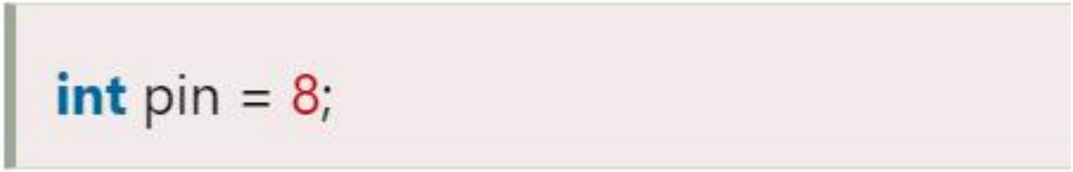
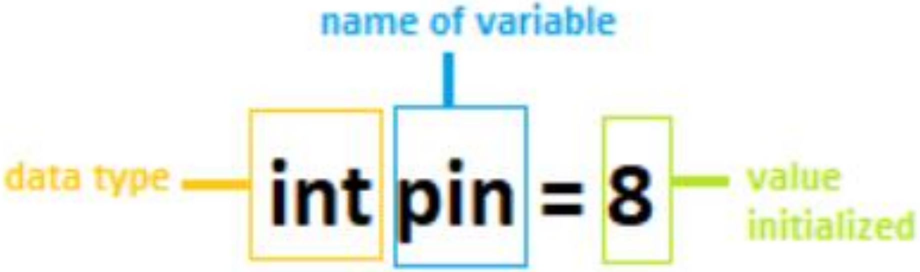
Arduino Variables

- The variables are defined as the place to store the data and values. It consists of a name, value, and type.
- The variables can belong to any data type such as int, float, char, etc.



```
int pin = 8;
```



Advantages of Variables

- We can use a variable many times in a program.
- The variables can represent integers, strings, characters, etc.
- It increases the flexibility of the program.
- We can easily modify the variables. For example, if we want to change the value of variable LEDpin from 8 to 13, we need to change the only point in the code.
- We can specify any name for a variable. For example, greenpin, bluePIN, REDpin, etc.

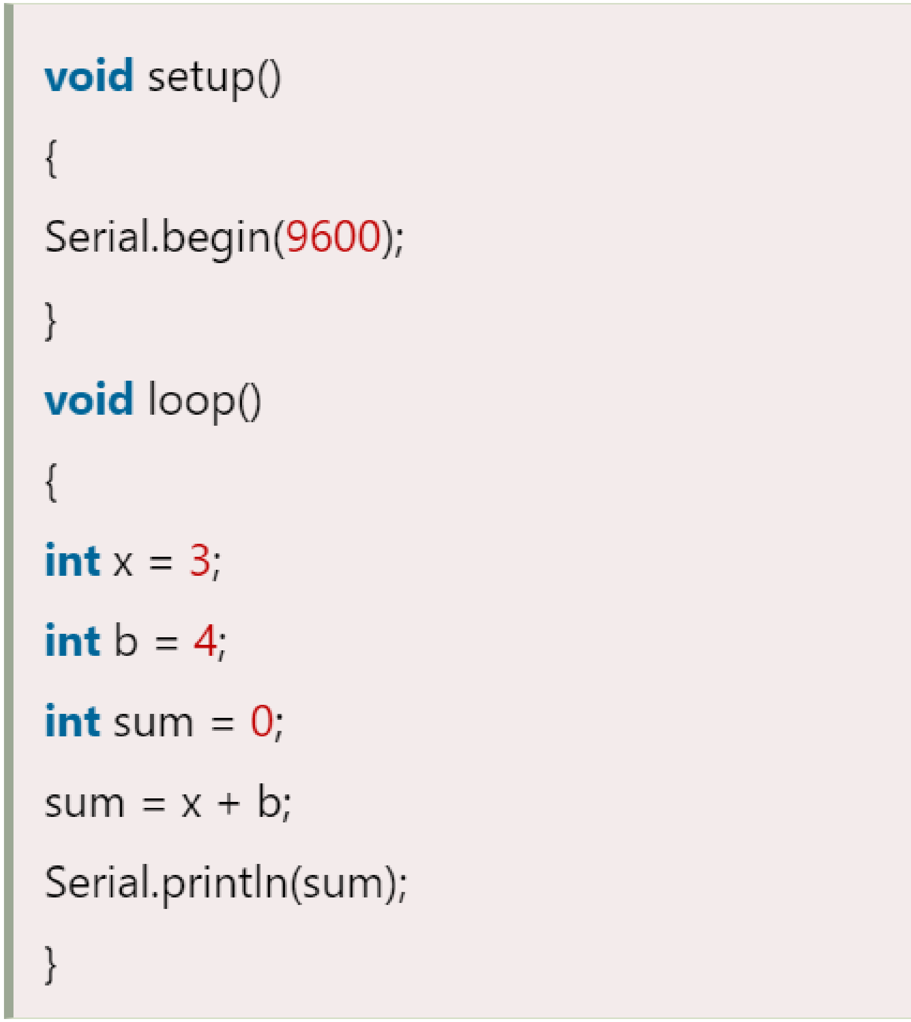
- The variables can be declared in two ways in Arduino, which are listed below:
- **Local variables**
- **Global variables**

Local Variables

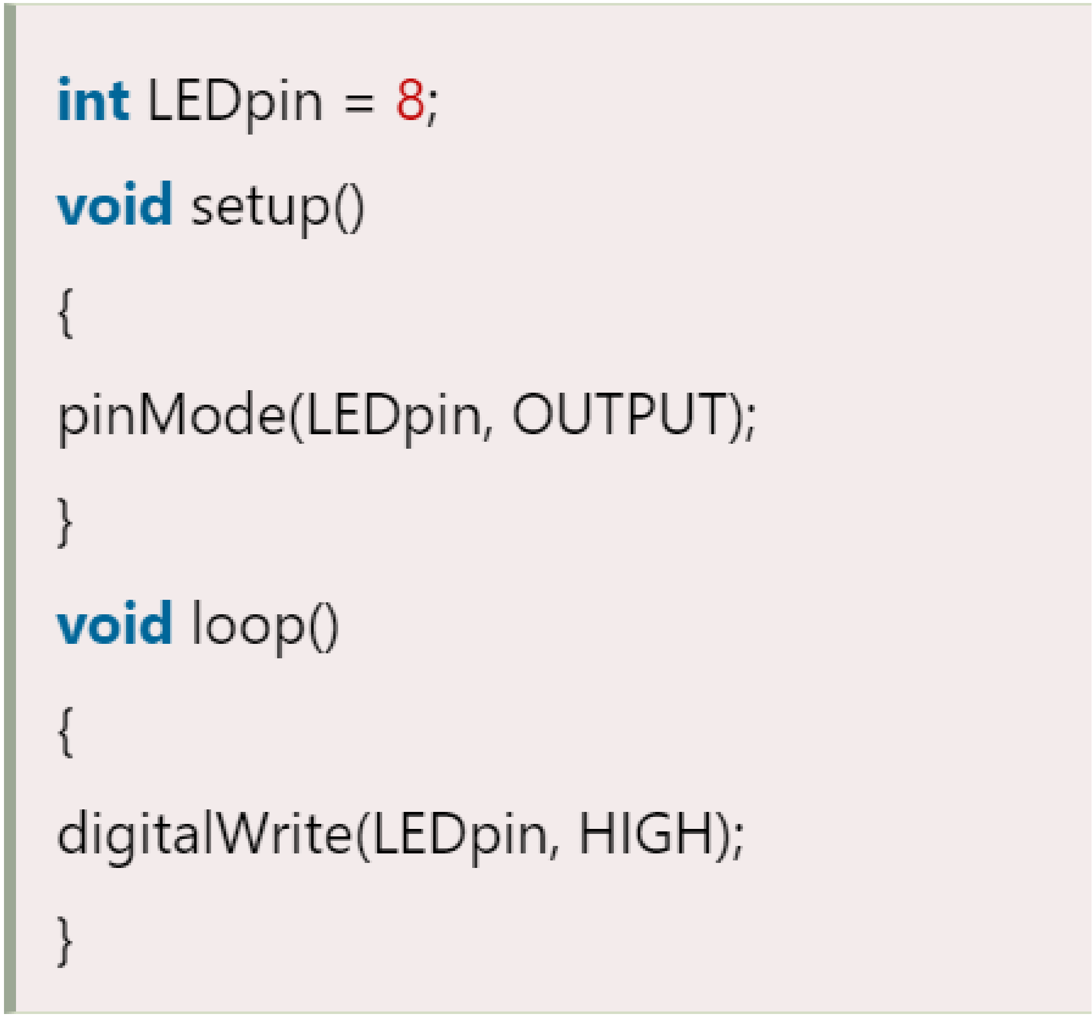
- The local variables are declared within the function. The variables have scope only within the function. These variables can be used only by the statements that lie within that function.

Global Variables

- The global variables can be accessed anywhere in the program.
- The global variable is declared outside the setup() and loop() function.



```
void setup()
{
Serial.begin(9600);
}
void loop()
{
int x = 3;
int b = 4;
int sum = 0;
sum = x + b;
Serial.println(sum);
}
```



```
int LEDpin = 8;

void setup()

{

pinMode(LEDpin, OUTPUT);

}

void loop()

{

digitalWrite(LEDpin, HIGH);

}
```

constants

- The constants in Arduino are defined as the predefined expressions. It makes the code easy to read.
- The constants in Arduino are defined as:

Logical level Constants

- The logical level constants are **true** or **false**.
- The value of true and false are defined as 1 and 0.
- Any non-zero integer is determined as true in terms of Boolean language.
- The true and false constants are type in lowercase rather than uppercase (such as HIGH, LOW, etc.).

Pin level Constants

- The digital pins can take two value **HIGH** or **LOW**.
- In Arduino, the pin is configured as INPUT or OUTPUT using the pinMode() function.
- The pin is further made HIGH or LOW using the digitalWrite() function.

HIGH

- The board includes two types of voltage pins to provide HIGH value, which are listed below:
- 5V
- 3V

LOW

- The pin configured as **LOW** is set at 0 Volts.

Arduino Operators

- The operators are used to solve logical and mathematical problems. For example, to calculate the temperature given by the sensor based on some analog voltage.
- The types of Operators classified in Arduino are:
- Arithmetic Operators
- Compound Operators
- Boolean Operators
- Comparison Operators
- Bitwise Operators

Arithmetic Operators

- There are six basic operators responsible for performing mathematical operations in Arduino, which are listed below:

Assignment Operator (=)

- The Assignment operator in Arduino
- is used to set the variable's value. It is quite different from the equal symbol (=) normally used in mathematics.

Addition (+)

- The addition operator is used for the addition of two numbers. For example, P + Q.

Subtraction (-)

- Subtraction is used to subtract one value from the another. For example, P - Q.

Multiplication (*)

- The multiplication is used to multiply two numbers. For example, P * Q.

Division (/)

- The division is used to determine the result of one number divided with another. For example, P/Q.

Modulo (%)

- The Modulo operator is used to calculate the remainder after the division of one number by another number.

Compound Operators

- The compound operators perform two or more calculations at once.
- The result of the right operand is assigned to the left operand, as already discussed above. The same condition will apply to all the compound operators, which are listed below:
- Let's consider a variable **b**.

b + +

- Here, b = b + 1. It is called the **increment operator**.

b + =

- For example, b + = 4. It means, b = b+ 4.

b - -

- Here, b = b - 1. It is called as the **decrement operator**.

b - =

- For example, b - = 3. It means, b = b - 3.

b * =

- For example, b * = 6. It means, b = b * 6.

b / =

- For example, b / = 5. It means, b = b / 5.

b % =

- For example, b % = 2. It means, b = b % 2.

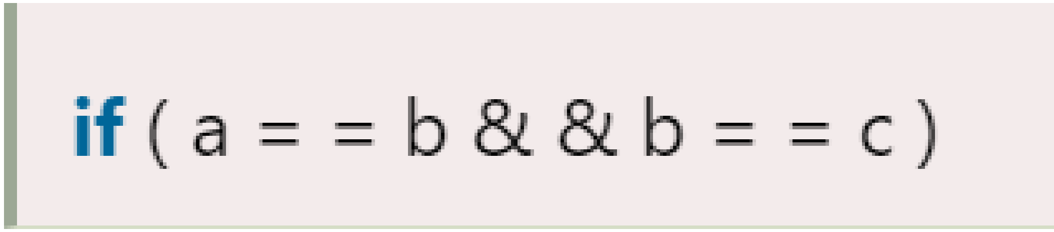
Boolean Operators

The Boolean Operators are

- **NOT (!)**
- **Logical AND (& &)**
- **Logical OR (| |)**

Logical AND (& &)

- The result of the condition is true if both the operands in the condition are true.
- Consider the below example:

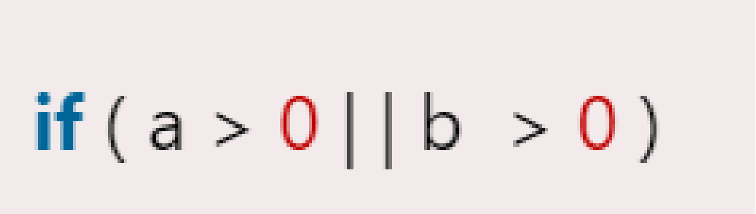


```
if ( a = = b & & b = = c )
```

- Above statement is true if both conditions are true. If any of the conditions is false, the statement will be false.

Logical OR (| |)

- The result of the condition is true, if either of the variables in the condition is true.
- Consider the below example.



```
if ( a > 0 || b  > 0 )
```

- The above statement is true, if either of the above condition (a> 0 or b > 0) is true.

NOT (!)

- It is used to reverse the logical state of the operand.

Comparison Operators

- The comparison operators are used to compare the value of one variable with the other.
- The comparison operators are listed below:

1 . less than (<)

- The less than operator checks that the value of the left operand is less than the right operand. The statement is true if the condition is satisfied.

2 greater than (>)

- The less than operator checks that the value of the left side of a statement is greater than the right side. The statement is true if the condition is satisfied.
- For example, a > b.
- If a is greater than b, the condition is true, else false.

3 equal to (= =)

- It checks the value of two operands. If the values are equal, the condition is satisfied.
- For example, a = = b.
- The above statement is used to check if the value of a is equal to b or not.

4 not equal to (! =)

- It checks the value of two specified variables. If the values are not equal, the condition will be correct and satisfied.
- For example, a ! = b.

5 less than or equal to (< =)

- The less or equal than operator checks that the value of left side of a statement is less or equal to the value on right side. The statement is true if either of the condition is satisfied.
- For example, a < = b
- It checks the value of a is less or equal than b.

6 greater than or equal to (> =)

- The greater or equal than operator checks that the value of the left side of a statement is greater or equal to the value on the right side of that statement. The statement is true if the condition is satisfied.
- For example, a > = b
- It checks the value of a is greater or equal than b. If either of the condition satisfies, the statement is true.

Bitwise Operators

- The Bitwise operators operate at the **binary level**.

1 bitwise NOT (~)

- The bitwise NOT operator acts as a complement for reversing the bits.
- For example, if b = 1, the NOT operator will make the value of b = 0.

2 bitwise XOR (^)

- The output is 0 if both the inputs are same, and it is 1 if the two input bits are different.

3 bitwise OR (|)

- The output is 0 if both of the inputs in the OR operation are 0. Otherwise, the output is 1. The two input patterns are of 4 bits.

4 bitwise AND (&)

- The output is 1 if both the inputs in the AND operation are 1. Otherwise, the output is 0. The two input patterns are of 4 bits.

5 bitwise left shift (< <)

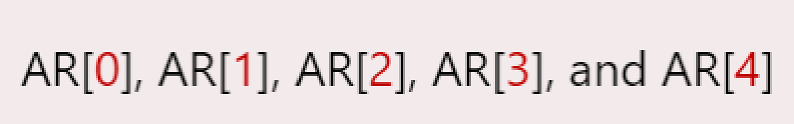
- The left operator is shifted by the number of bits defined by the right operator.

6 bitwise right shift (> >)

- The right operator is shifted by the number of bits defined by the left operator.

Arduino Array

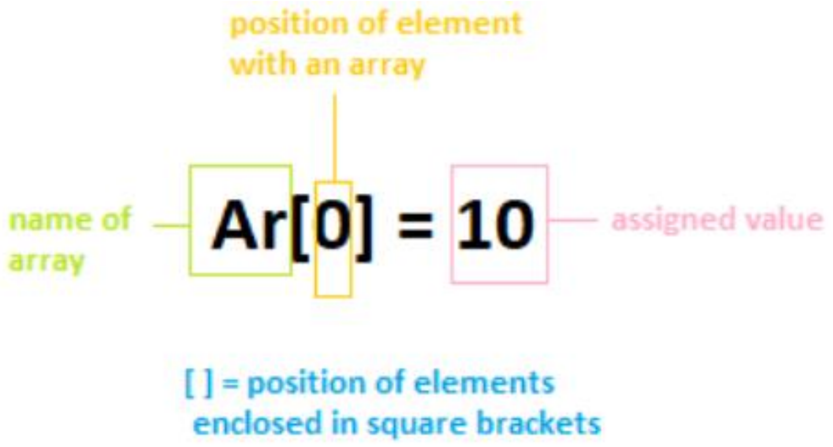
- What are Arrays?
- The arrays are defined as the data structures that allow multiple values to be grouped together in a simple way. This is an easily access method.
- The array is normally built from the data types like **integer, reals, characters, and boolean**. It refers to a named list of finite number (n) of similar data elements.
- The set of consecutive numbers usually represent the elements in the array, which are **0, 1, 2, 3, 4, 5, 6,.......n.**
- For example, if the name of an array of 5 elements is AR, the elements will be referenced as shown below:



AR[0], AR[1], AR[2], AR[3], and AR[4]

Arrays in Arduino

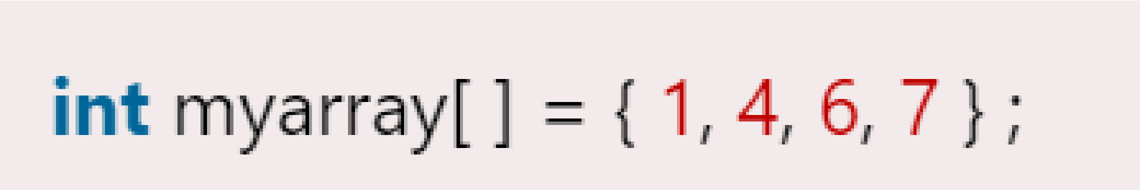
- The array in Arduino is declared with the integer data type.
- It is also defined as the collection of variables, which is acquired with an index number.
- The array is represented as:


position of element with an array
name of array
Ar[0] = 10
assigned value
[ ] = position of elements enclosed in square brackets

- We can specify any name according to our choice. The array name is the individual name of an element.

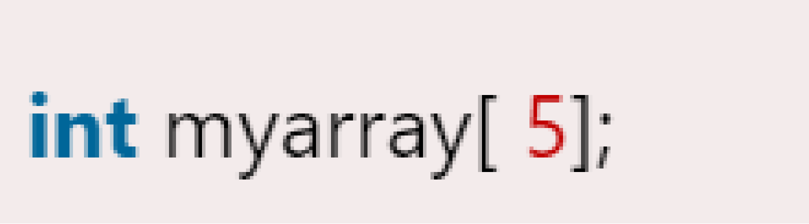
Array Declaration

- There are different methods to declare an array in Arduino, which are listed below:
- We can declare the array without specifying the size.



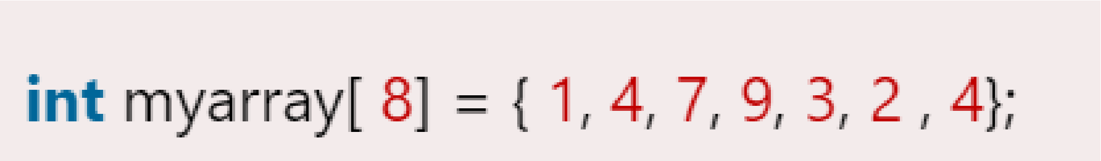
```
int myarray[ ] = { 1, 4, 6, 7 } ;
```

- We can declare the array without initializing its elements.
- For example,



```
int myarray[ 5];
```

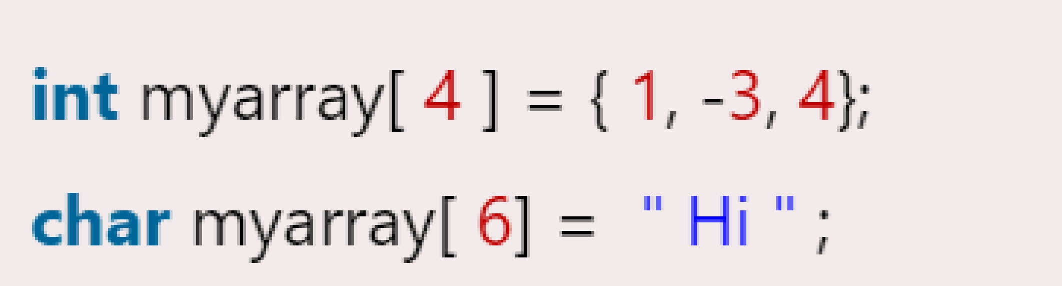
- We can declare the array by initializing the size and elements.



```
int myarray[ 8] = { 1, 4, 7, 9, 3, 2 , 4};
```

Features of Array

- The elements of the array can be characters, negative numbers, etc.
- For example,



```
int myarray[ 4 ] = { 1, -3, 4};
char myarray[ 6] =  " Hi " ;
```

- The size of the array should not be less than the number of elements. For example,
- **int myarray[5] = { 1, 4, 6, 7 } ;** can be written as **int myarray[8] = { 1, 4, 6, 7 } ;**
- But, it cannot be written as
- **int myarray[2] = { 1, 4, 6, 7 } ;**

- The total elements, while specifying the **char** type should be **(n - 1),** where **n is the size of the array.** It is because one element is required to hold the null character in the array.
- For example,
- Let's specify the array as
- **char abc[8] = " Arduino";**

Access of array in Arduino

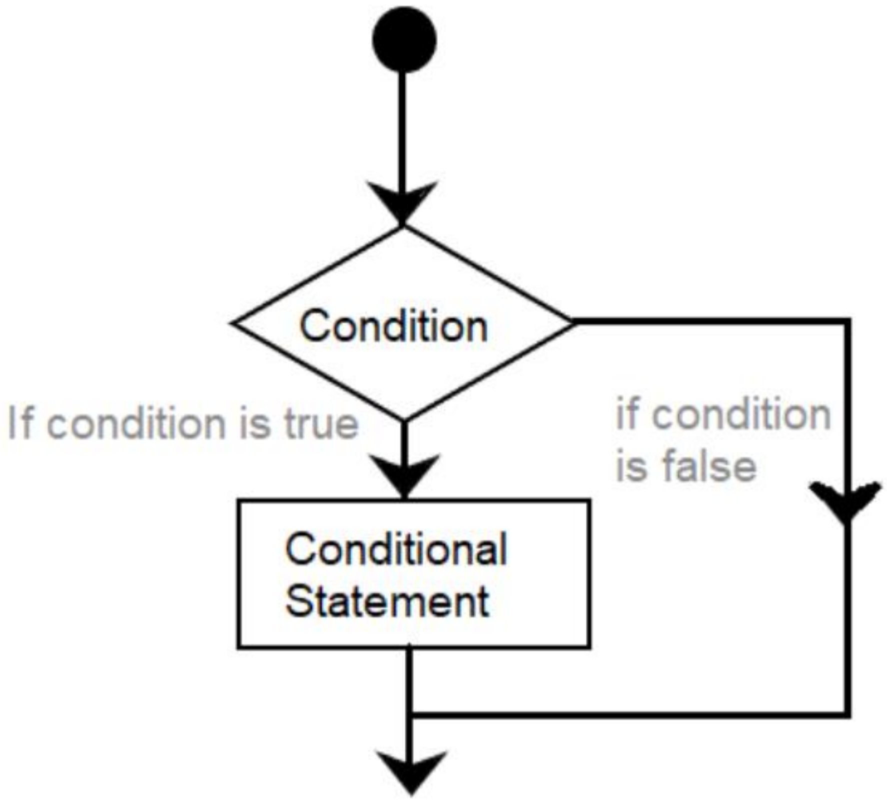
- The array in Arduino has zero index. It means that the first element of the array is indexed as 0.
- For example,
- Let's take an array (**ARarduino**) of **6** elements. The elements of the array are shown below:
- **ARarduino[0], ARarduino[1], ARarduino[2], ARarduino[3], ARarduino[4],**
- and **ARarduino[5].**
- The last element of the array is ARarduino[5].
- The last element of the array will be n-1, where n is the declared size of an array.

Arduino - Control Statements

- Control Statements are elements in Source Code that control the flow of program execution.

Arduino If statement

- The if ( ) statement is the conditional statement, which is the basis for all types of programming languages.
- If the condition in the code is true, the corresponding task or function is performed accordingly. It returns one value if the condition in a program is **true**. It further returns another value if the condition is **false**.
- It means that if ( ) statement checks for the condition and then executes a statement or a set of statements.



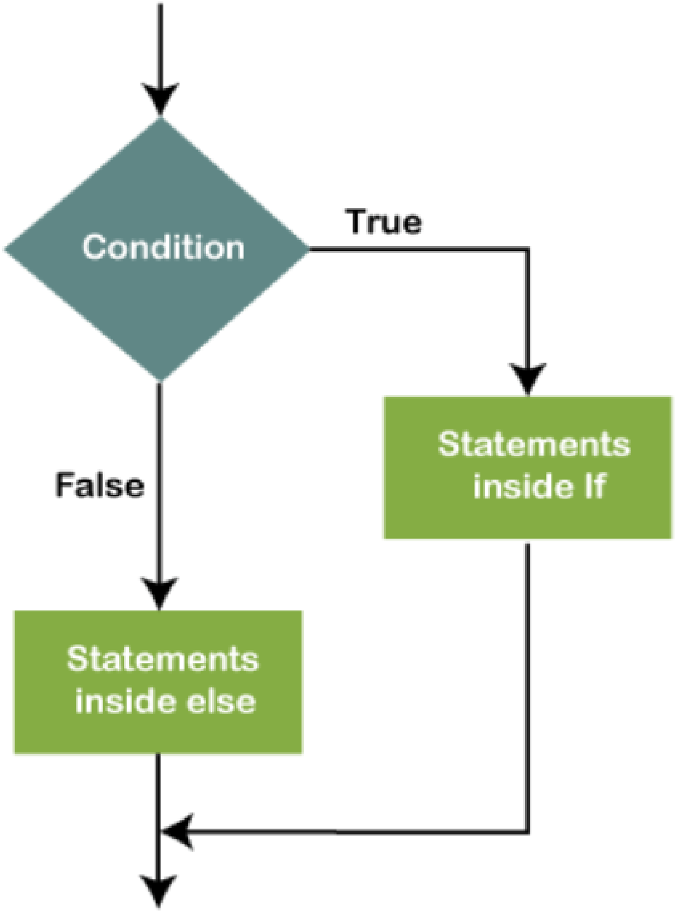
- It clearly explains the process of execution of a statement. If the condition is False, it comes out of the if ( ) statement. If the condition is true, the function is performed.

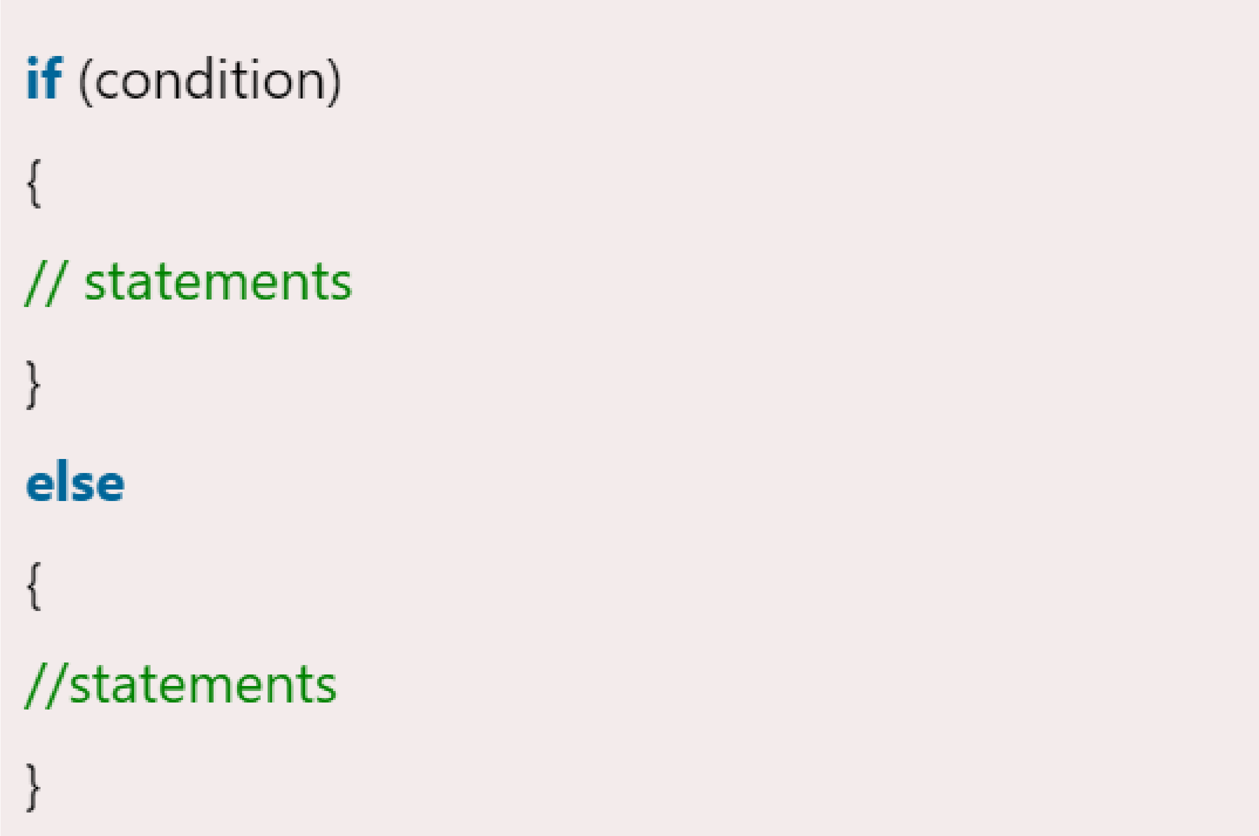
- Here,
- **condition** = It includes the boolean expressions, that can be true or false.
- We can also use one or more operators inside the parentheses.
- The comparison operators that can be used as a condition inside the parentheses are listed below:
- a ! = b (a not equal to b)
- a < b (a less than b)
- a > b (a greater than b)
- a = = b (a equal to b)
- a < = b (a less than or equal to b)
- a > = b (a greater than or equal to b)
- where,
- **a** and **b** are the variables.

Arduino if-else and else-if

If else

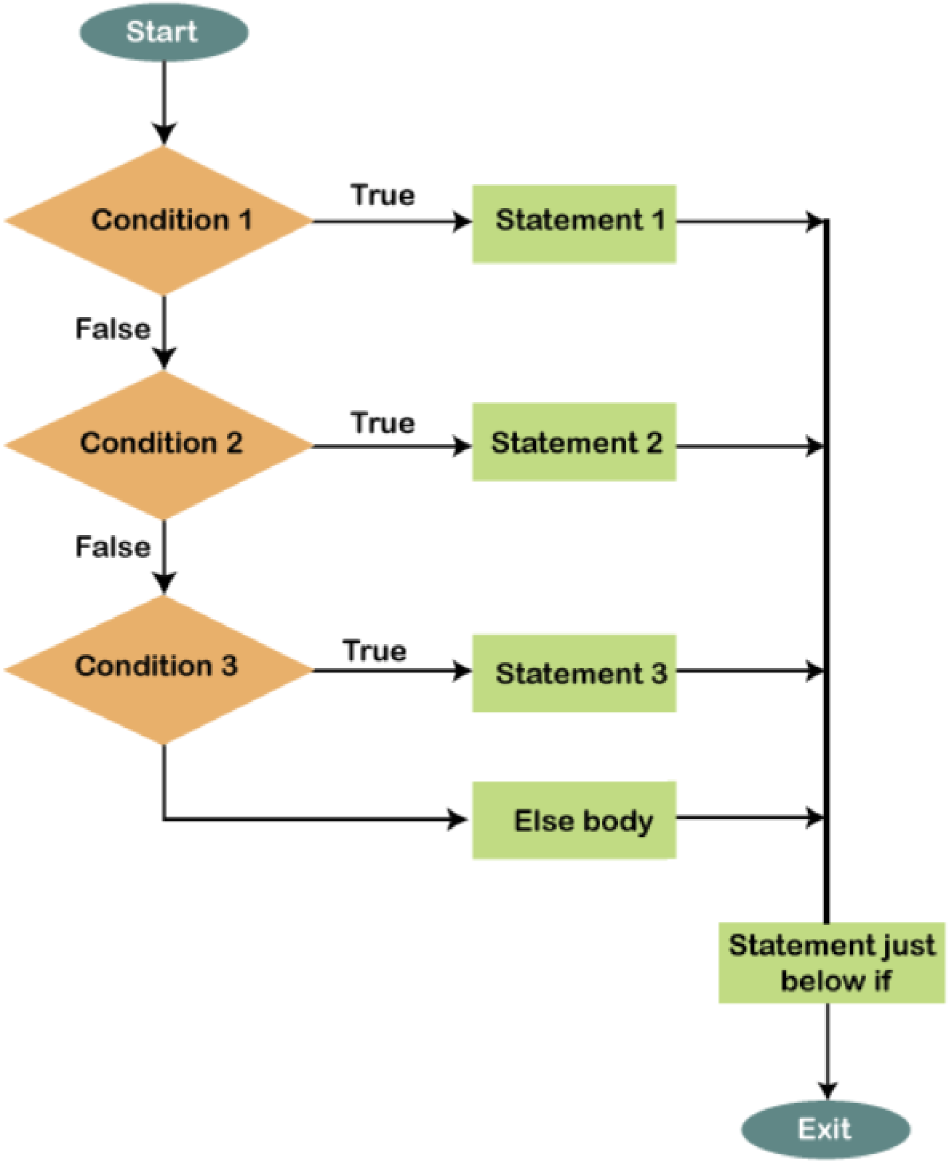
- The if-else condition includes if ( ) statement and else ( ) statement. The condition in the else statement is executed if the result of the If ( ) statement is false.





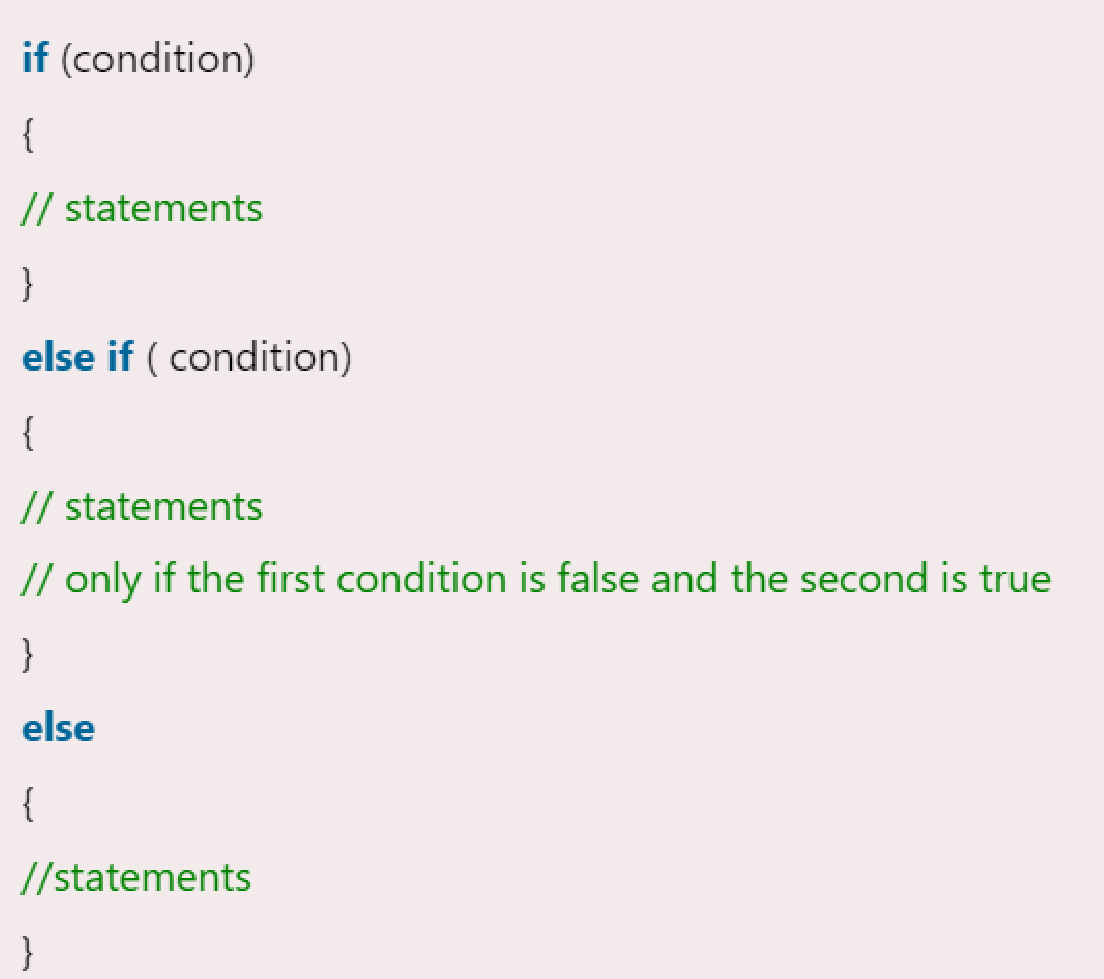
```
if (condition)
{
// statements
}
else
{
//statements
}
```

- The else( ) statement can also include other if statements. Due to this, we can run multiple statements in a single program.
- The flowchart is shown below:



Else if

- The else if statement can be used with or without the else ( ) statement. We can include multiple else if statements in a program.



```
if (condition)
{
// statements
}
else if ( condition)
{
// statements
// only if the first condition is false and the second is true
}
else
{
//statements
}
```

- The else if ( ) statement will stop the flow once its execution is true.
- What is the difference between Else and Else If?
- The **Else ( )** part is executed if one or all the **If ( )** conditions present in the code comes out to be false.
- The else if ( ) will stop the program flow if it becomes true.

Arduino Functions

- The functions allow a programmer to divide a specific code into various sections, and each section performs a particular task. The functions are created to perform a task multiple times in a program.
- The function is a type of procedure that returns the area of code from which it is called.
- For example, to repeat a task multiple times in code, we can use the same set of statements every time the task is performed.

- The Arduino has two common functions **setup()** and **loop(),** which are called automatically in the background. The code to be executed is written inside the curly braces within these functions.
- **void setup()** - It includes the initial part of the code, which is executed only once. It is called as the **preparation block**.
- **void loop()** - It includes the statements, which are executed repeatedly. It is called the **execution block**.

Function Declaration

- The method to declare a function is listed below:

Function return type

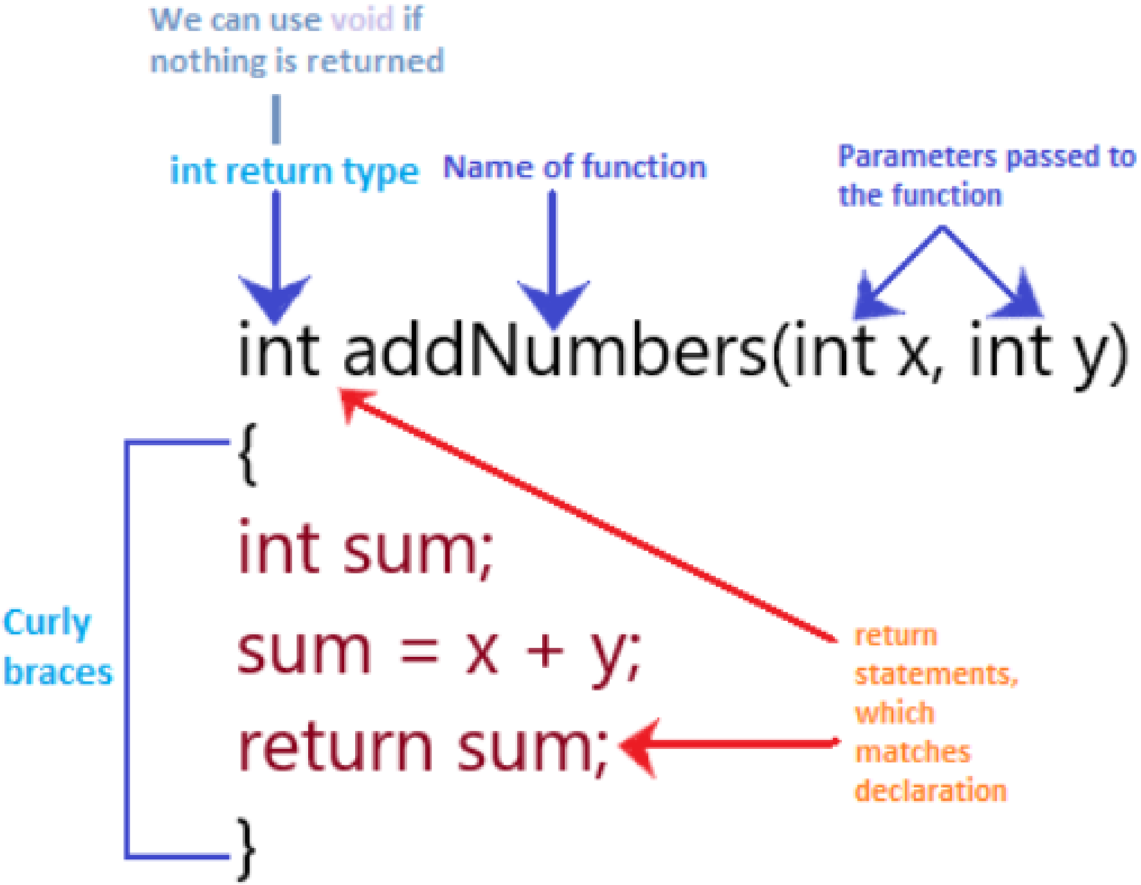
- We need a return type for a function. For example, we can store the return value of a function in a variable.
- We can use any data type as a return type, such as **float, char**, etc.

Function name

- It consists of a name specified to the function. It represents the real body of the function.

Function parameter

- It includes the parameters passed to the function. The parameters are defined as the special variables, which are used to pass data to a function.
- The function must be followed by **parentheses ( )** and the **semicolon ;**
- The actual data passed to the function is termed as an argument.



We can use void if nothing is returned

int return type

Name of function

Parameters passed to the function

```
int addNumbers(int x, int y)
{
int sum;
sum = x + y;
return sum;
}
```

Curly braces

return statements, which matches declaration