**Arduino**

Arduino is a prototype platform (open-source) based on an easy-to-use hardware and software. It consists of a circuit board, which can be programed (referred to as a microcontroller) and a ready-made software called Arduino IDE (Integrated Development Environment), which is used to write and upload the computer code to the physical board.

Arduino provides a standard form factor that breaks the functions of the micro-controller into a more accessible package.

**Software structure functions in arduino**

Void setup ( ) {

}

* PURPOSE − The setup() function is called when a sketch starts. Use it to initialize the variables, pin modes, start using libraries, etc. The setup function will only run once, after each power up or reset of the Arduino board.
* INPUT − -
* OUTPUT − -
* RETURN − -

Void Loop ( ) {

}

* PURPOSE − After creating a setup() function, which initializes and sets the initial values, the loop() function does precisely what its name suggests, and loops consecutively, allowing your program to change and respond. Use it to actively control the Arduino board.
* INPUT − -
* OUTPUT − -
* RETURN − -

**If structure in arduino**

Description

The if statement checks for a condition and executes the proceeding statement or set of statements if the condition is 'true'.

Syntax

if (condition) {

//statement(s)

}

Parameters

condition: a boolean expression (i.e., can be true or false).

Example Code

The brackets may be omitted after an if statement. If this is done, the next line (defined by the semicolon) becomes the only conditional statement.

if (x > 120) digitalWrite(LEDpin, HIGH);

if (x > 120)

digitalWrite(LEDpin, HIGH);

if (x > 120) {digitalWrite(LEDpin, HIGH);}

if (x > 120) {

digitalWrite(LEDpin1, HIGH);

digitalWrite(LEDpin2, HIGH);

}

**Setup() and loop()**

Void setup ( ) {

}

* PURPOSE − The setup() function is called when a sketch starts. Use it to initialize the variables, pin modes, start using libraries, etc. The setup function will only run once, after each power up or reset of the Arduino board.
* INPUT − -
* OUTPUT − -
* RETURN − -

Void Loop ( ) {

}

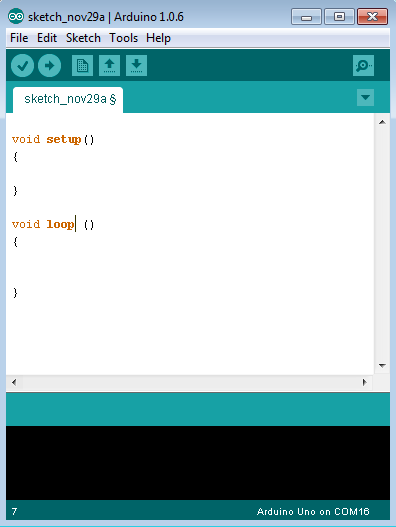
* PURPOSE − After creating a setup() function, which initializes and sets the initial values, the loop() function does precisely what its name suggests, and loops consecutively, allowing your program to change and respond. Use it to actively control the Arduino board.
* INPUT − -
* OUTPUT − -
* RETURN − -

**Structure of arduino programming**

Arduino programs can be divided in three main parts: Structure, Values (variables and constants), and Functions. In this tutorial, we will learn about the Arduino software program, step by step, and how we can write the program without any syntax or compilation error.

Let us start with the Structure. Software structure consist of two main functions −

* Setup( ) function
* Loop( ) function



Void setup ( ) {

}

* PURPOSE − The setup() function is called when a sketch starts. Use it to initialize the variables, pin modes, start using libraries, etc. The setup function will only run once, after each power up or reset of the Arduino board.
* INPUT − -
* OUTPUT − -
* RETURN − -

Void Loop ( ) {

}

* PURPOSE − After creating a setup() function, which initializes and sets the initial values, the loop() function does precisely what its name suggests, and loops consecutively, allowing your program to change and respond. Use it to actively control the Arduino board.
* INPUT − -
* OUTPUT − -
* RETURN − -

**Operators in arduino**

An operator is a symbol that tells the compiler to perform specific mathematical or logical functions. C language is rich in built-in operators and provides the following types of operators −

* Arithmetic Operators
* Comparison Operators
* Boolean Operators
* Bitwise Operators
* Compound Operators

You can explain each . it is same as c programming

**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 are 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 |

Boolean

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

Example

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

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

Char

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".

However, characters are stored as numbers. You can see the specific encoding in the [ASCII chart](https://www.arduino.cc/en/Reference/ASCIIchart). This means that it is possible to do arithmetic operations on characters, in which the ASCII value of the character is used. For example, 'A' + 1 has the value 66, since the ASCII value of the capital letter A is 65.

Example

Char chr\_a = ‘a’ ;//declaration of variable with type char and initialize it with character a

Char chr\_c = 97 ;//declaration of variable with type char and initialize it with character 97

unsigned char

Unsigned char is an unsigned data type that occupies one byte of memory. The unsigned char data type encodes numbers from 0 to 255.

Example

Unsigned Char chr\_y = 121 ; // declaration of variable with type Unsigned char and initialize it with character y

byte

A byte stores an 8-bit unsigned number, from 0 to 255.

Example

byte m = 25 ;//declaration of variable with type byte and initialize it with 25

int

Integers are the primary data-type for number storage. int stores a 16-bit (2-byte) value. This yields a range of -32,768 to 32,767 (minimum value of -2^15 and a maximum value of (2^15) - 1).

The int size varies from board to board. On the Arduino Due, for example, an int stores a 32-bit (4-byte) value. This yields a range of -2,147,483,648 to 2,147,483,647 (minimum value of -2^31 and a maximum value of (2^31) - 1).

Example

int counter = 32 ;// declaration of variable with type int and initialize it

**Local and global variables**

A global variable is one that can be seen by every function in a program. Local variables are only visible to the function in which they are declared. In the Arduino environment, any variable declared outside of a function (e.g. [setup()](https://www.arduino.cc/reference/en/language/structure/sketch/setup), [loop()](https://www.arduino.cc/reference/en/language/structure/sketch/loop), etc. ), is a *global* variable.

When programs start to get larger and more complex, local variables are a useful way to insure that only one function has access to its own variables. This prevents programming errors when one function inadvertently modifies variables used by another function.

It is also sometimes handy to declare and initialize a variable inside a [for](https://www.arduino.cc/reference/en/language/structure/control-structure/for) loop. This creates a variable that can only be accessed from inside the [for](https://www.arduino.cc/reference/en/language/structure/control-structure/for)-loop brackets.

Example Code

int gPWMval; // any function will see this variable

void setup() {

// ...

}

void loop() {

int i; // "i" is only "visible" inside of "loop"

float f; // "f" is only "visible" inside of "loop"

// ...

for (int j = 0; j < 100; j++) {

// variable j can only be accessed inside the for-loop brackets

}

}

**For loop**

A **for loop** executes statements a predetermined number of times. The control expression for the loop is initialized, tested and manipulated entirely within the for loop parentheses. It is easy to debug the looping behavior of the structure as it is independent of the activity inside the loop.

Each **for** loop has up to three expressions, which determine its operation. The following example shows general for loop syntax. Notice that the three expressions in the for loop argument parentheses are separated with semicolons.

## for loop Syntax

for ( initialize; control; increment or decrement) {

// statement block

}

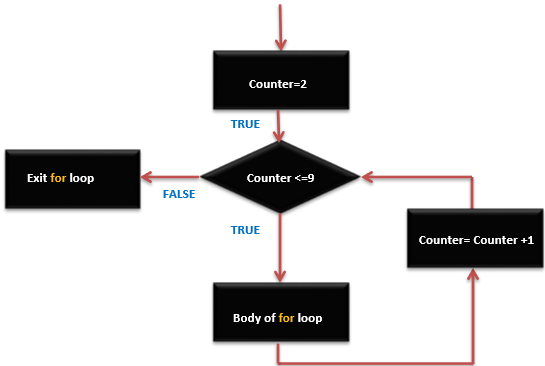
### Example

for(counter = 2;counter <= 9;counter++) {

//statements block will executed 10 times

}

## for loop Execution Sequence



**Functions in arduino programming**

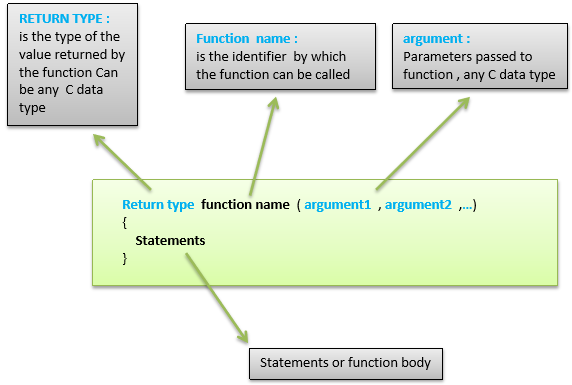
Functions allow structuring the programs in segments of code to perform individual tasks. The typical case for creating a function is when one needs to perform the same action multiple times in a program.

Standardizing code fragments into functions has several advantages −

* Functions help the programmer stay organized. Often this helps to conceptualize the program.
* Functions codify one action in one place so that the function only has to be thought about and debugged once.
* This also reduces chances for errors in modification, if the code needs to be changed.
* Functions make the whole sketch smaller and more compact because sections of code are reused many times.
* They make it easier to reuse code in other programs by making it modular, and using functions often makes the code more readable.

There are two required functions in an Arduino sketch or a program i.e. setup () and loop(). Other functions must be created outside the brackets of these two functions.

The most common syntax to define a function is −



**Function Declaration**

A function is declared outside any other functions, above or below the loop function.

We can declare the function in two different ways −

The first way is just writing the part of the function called **a function prototype** above the loop function, which consists of −

* Function return type
* Function name
* Function argument type, no need to write the argument name

Function prototype must be followed by a semicolon ( ; ).

The following example shows the demonstration of the function declaration using the first method.

**Example**

int sum\_func (int x, int y) // function declaration {

int z = 0;

z = x+y ;

return z; // return the value

}

void setup () {

Statements // group of statements

}

Void loop () {

int result = 0 ;

result = Sum\_func (5,6) ; // function call

}

The second part, which is called the function definition or declaration, must be declared below the loop function, which consists of −

* Function return type
* Function name
* Function argument type, here you must add the argument name
* The function body (statements inside the function executing when the function is called)

The following example demonstrates the declaration of function using the second method.

**Example**

int sum\_func (int , int ) ; // function prototype

void setup () {

Statements // group of statements

}

Void loop () {

int result = 0 ;

result = Sum\_func (5,6) ; // function call

}

int sum\_func (int x, int y) // function declaration {

int z = 0;

z = x+y ;

return z; // return the value

}