# Introduction to 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.

The key features are −

Arduino boards are able to read analog or digital input signals from different sensors and turn it into an output such as activating a motor, turning LED on/off, connect to the cloud and many other actions.

You can control your board functions by sending a set of instructions to the microcontroller on the board via Arduino IDE (referred to as uploading software).

Unlike most previous programmable circuit boards, Arduino does not need an extra piece of hardware (called a programmer) in order to load a new code onto the board. You can simply use a USB cable.

Additionally, the Arduino IDE uses a simplified version of C++, making it easier to learn to program.

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

## The Structure

The basic structure of Arduino code is fairly simple and straightforward. I all Arduino programs, you must have a void setup and void loop functions

Void setup()-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.

Void loop()-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.

#include <DHT.h>

/\*

A header file is a file containing C declarations and macro definitions

to be shared between several source files. You request the use of

a header file in your program by including it, with the C preprocessing directive ' #include '.\*/

#define some\_variable 25.5 //some constants defined in the program

void setup(){

//some statements to initialize your program

}

void setup(){

//some statements you want to always repeat

}

//The above code also shows the use of comments, both single line comments and multi-line comments

## 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 |

We’ll take a look at the most used data types in Arduino Programming

## **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(){//the void data type is shown here

## **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 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".

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

## **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).

float b=25.556; //decleration of variable with type float

Variables in C programming language, which Arduino uses, have a property called scope. A scope is a region of the program and there are three places where variables can be declared. They are −

* Inside a function or a block, which is called **local variables**.
* In the definition of function parameters, which is called **formal parameters**.
* Outside of all functions, which is called **global variables**.

### **Local Variables**

Variables that are declared inside a function or block are local variables. They can be used only by the statements that are inside that function or block of code. Local variables are not known to function outside their own. Following is the example using local variables −

### **Global Variables**

Global variables are defined outside of all the functions, usually at the top of the program. The global variables will hold their value throughout the life-time of your program.

A global variable can be accessed by any function. That is, a global variable is available for use throughout your entire program after its declaration.

The following example uses global and local variables −

int T , S ;

float c = 0 ; //Global variable declaration

void setup () {

}

void loop () {

int x , y ;

int z ; //Local variable declaration

x = 0;

y = 0; //actual initialization

z = 10;

}

## Operators

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

## **Arithmetic Operators**

Assume variable A holds 10 and variable B holds 20 then −

[Show Example](https://www.tutorialspoint.com/arduino/arduino_arithmetic_operators.htm)

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator name** | **Operator simple** | **Description** | **Example** |
| assignment operator | = | Stores the value to the right of the equal sign in the variable to the left of the equal sign. | A = B |
| addition | + | Adds two operands | A + B will give 30 |
| subtraction | - | Subtracts second operand from the first | A - B will give -10 |
| multiplication | \* | Multiply both operands | A \* B will give 200 |
| division | / | Divide numerator by denominator | B / A will give 2 |
| modulo | % | Modulus Operator and remainder of after an integer division | B % A will give 0 |

## **Comparison Operators**

Assume variable A holds 10 and variable B holds 20 then −

[Show Example](https://www.tutorialspoint.com/arduino/arduino_comparison_operators.htm)

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator name** | **Operator simple** | **Description** | **Example** |
| equal to | == | Checks if the value of two operands is equal or not, if yes then condition becomes true. | (A == B) is not true |
| not equal to | != | Checks if the value of two operands is equal or not, if values are not equal then condition becomes true. | (A != B) is true |
| less than | < | Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true. | (A < B) is true |
| greater than | > | Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true. | (A > B) is not true |
| less than or equal to | <= | Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true. | (A <= B) is true |
| greater than or equal to | >= | Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true. | (A >= B) is not true |

## **Boolean Operators**

Assume variable A holds 10 and variable B holds 20 then −

[Show Example](https://www.tutorialspoint.com/arduino/arduino_boolean_operators.htm)

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator name** | **Operator simple** | **Description** | **Example** |
| and | && | Called Logical AND operator. If both the operands are non-zero then then condition becomes true. | (A && B) is true |
| or | || | Called Logical OR Operator. If any of the two operands is non-zero then then condition becomes true. | (A || B) is true |
| not | ! | Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false. | !(A && B) is false |

## **Bitwise Operators**

Assume variable A holds 60 and variable B holds 13 then −

[Show Example](https://www.tutorialspoint.com/arduino/arduino_bitwise_operators.htm)

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator name** | **Operator simple** | **Description** | **Example** |
| and | & | Binary AND Operator copies a bit to the result if it exists in both operands. | (A & B) will give 12 which is 0000 1100 |
| or | | | Binary OR Operator copies a bit if it exists in either operand | (A | B) will give 61 which is 0011 1101 |
| xor | ^ | Binary XOR Operator copies the bit if it is set in one operand but not both. | (A ^ B) will give 49 which is 0011 0001 |
| not | ~ | Binary Ones Complement Operator is unary and has the effect of 'flipping' bits. | (~A ) will give -60 which is 1100 0011 |
| shift left | << | Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand. | A << 2 will give 240 which is 1111 0000 |
| shift right | >> | Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand. | A >> 2 will give 15 which is 0000 1111 |

## **Compound Operators**

Assume variable A holds 10 and variable B holds 20 then −

[Show Example](https://www.tutorialspoint.com/arduino/arduino_compound_operators.htm)

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator name** | **Operator simple** | **Description** | **Example** |
| increment | ++ | Increment operator, increases integer value by one | A++ will give 11 |
| decrement | -- | Decrement operator, decreases integer value by one | A-- will give 9 |
| compound addition | += | Add AND assignment operator. It adds right operand to the left operand and assign the result to left operand | B += A is equivalent to B = B+ A |
| compound subtraction | -= | Subtract AND assignment operator. It subtracts right operand from the left operand and assign the result to left operand | B -= A is equivalent to B = B - A |
| compound multiplication | \*= | Multiply AND assignment operator. It multiplies right operand with the left operand and assign the result to left operand | B\*= A is equivalent to B = B\* A |
| compound division | /= | Divide AND assignment operator. It divides left operand with the right operand and assign the result to left operand | B /= A is equivalent to B = B / A |
| compound modulo | %= | Modulus AND assignment operator. It takes modulus using two operands and assign the result to left operand | B %= A is equivalent to B = B % A |
| compound bitwise or | |= | bitwise inclusive OR and assignment operator | A |= 2 is same as A = A | 2 |
| compound bitwise and | &= | Bitwise AND assignment operator | A &= 2 is same as A = A & 2 |