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**.

Arrays

An array is a collection of values that are accessed with an index number. Any value in the array may be called upon by calling the name of the array and the index number of the value. Arrays are zero indexed.

An array needs to be declared and optionally assigned values before they can be used.

```
int myArray={34,23,67,1,34};
int x;
x=myArray[3];//x now equals 10
```

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

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

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

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

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

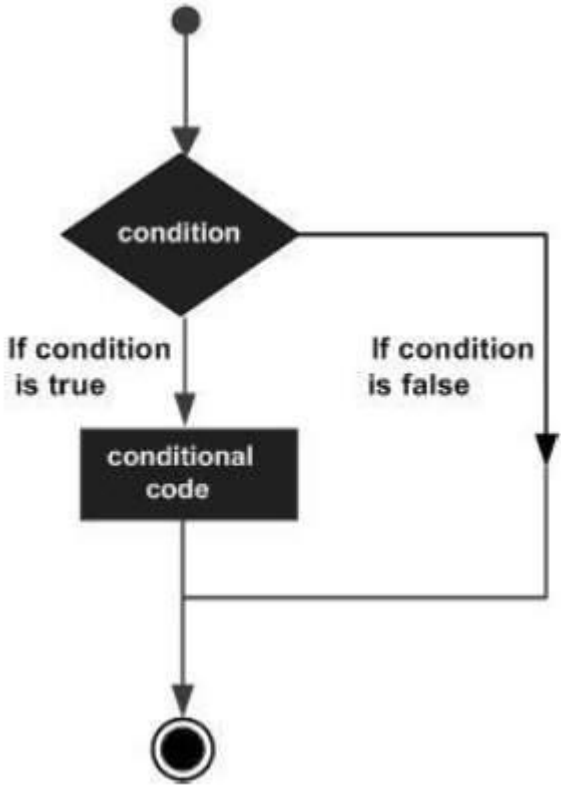
Operator name	Operator simple	Description	Example
increment	++	Increment operator, increases integer value by one	A++ will give 11
decrement	--	Decrement operator, decreases integer value by one	A-- will give 9
compound addition	+=	Add AND assignment operator. It adds right operand to the left operand 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

Arduino - Control Statements

Decision making structures require that the programmer specify one or more conditions to be evaluated or tested by the program. It should be along with a statement or statements to be executed if the condition is determined to be true, and optionally, other statements to be executed if the condition is determined to be false.

Following is the general form of a typical decision making structure found in most of the programming languages −



If statement

It takes an expression in parenthesis and a statement or block of statements. If the expression is true then the statement or block of statements gets executed otherwise these statements are skipped.

```
//Let's build a simple if conditional
int a =67;
int b = 78;
void setup(){
   Serial.begin(115200);
}
void loop(){
   if(b>a){
      Serial.println("b is more than a");
   }
}
```

If …else statement

An **if** statement can be followed by an optional else statement, which executes when the expression is false.

```
//Let's build a simple if conditional
```

```
int a =67;
int b = 78;
void setup(){
  Serial.begin(115200);
}
void loop(){
  if(b>a){
    Serial.println("b is more than a");
  }
  else{
    Serial.println("a is more than b");
  }
}
```

If…else if …else statement

The **if** statement can be followed by an optional **else if...else**statement, which is very useful to test various conditions using single if...else if statement.

```
//Let's build a simple if conditional
int a =67;
int b = 78;
void setup(){
  Serial.begin(115200);
}
void loop(){
  if(b>a){
    Serial.println("b is more than a");
  }
  else if (b=a){
    Serial.println("The numbers are equal");
  }
  else{
    Serial.println("a is more than b");
  }

  delay(2000);
}
```

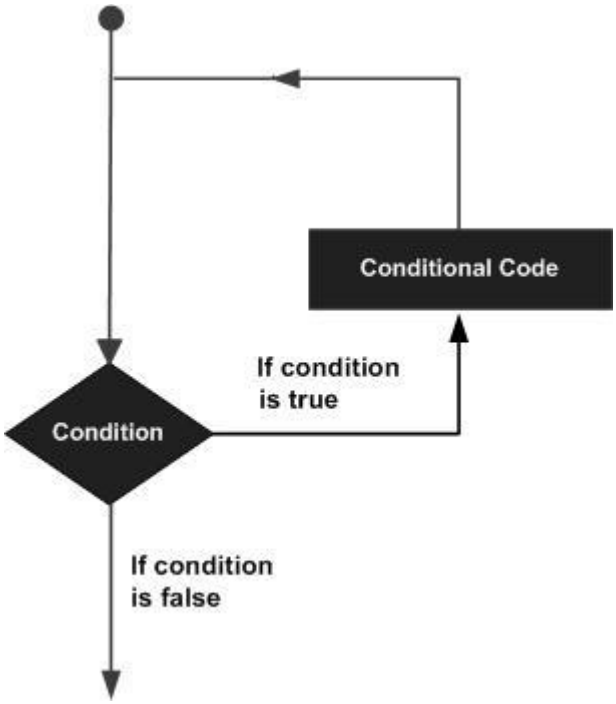
switch case statement

Similar to the if statements, **switch...case** controls the flow of programs by allowing the programmers to specify different codes that should be executed in various conditions.

Arduino - Loops

Programming languages provide various control structures that allow for more complicated execution paths.

A loop statement allows us to execute a statement or group of statements multiple times and following is the general form of a loop statement in most of the programming languages –



C programming language provides the following types of loops to handle looping requirements.

while loop

while loops will loop continuously, and infinitely, until the expression inside the parenthesis, () becomes false. Something must change the tested variable, or the while loop will never exit.

```
void setup(){
Serial.begin(115200);
}
void loop(){
   int a;
   a=45;
```

```
  while(a<56){
    Serial.println("The number is less than 56");//this is ideally an infinite loop
  }
}
```

do…while loop

The **do…while** loop is similar to the while loop. In the while loop, the loop-continuation condition is tested at the beginning of the loop before performed the body of the loop.

```
void setup(){
  Serial.begin(9600);
}
void loop(){
  float number 45.6;
  do {
    Serial.println("The number is still big");
    number=number-10;
  }
  while(number<50);///Remeber our infinite loop? This solves the problem as the
statement will become false at one point
}
```

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.

```
#define no 6
int count;

void setup(){
  Serial.begin(9600);

}
void loop(){
  for(count=0;count<=no;count++){  // count must be initialized before we can start using it
    Serial.println(count);//You should see the count printed 6 times with increasing
    delay(1000);
  }
}
```

Arduino - Functions

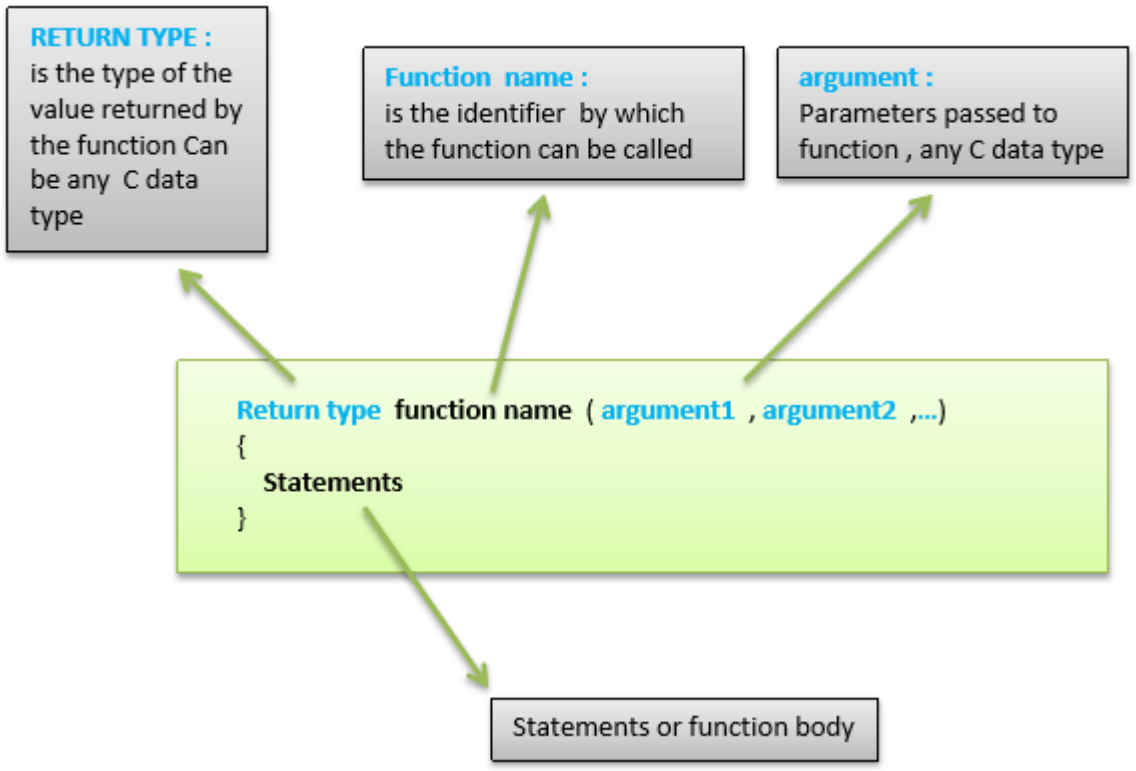
Functions allow structuring the programs in segments of code to perform individual tasks. The typical case for creating a function is when one needs to perform the same action multiple times in a program.

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.

```
void fun(void){
   Serial.println("Would you look at me, I was declared before the setup() function, and am
not even returning anything");

   delay(3000);
}

void setup(){
   Serial.begin(9600);
}
void loop(){
int holder;
holder=result(3,1,0);

Serial.println(holder);

fun();
}

int result(int a, int b, int c){
   int answer;
   answer=a+b+c;
   return answer;
}
```

Reading Inputs and Outputs

pinMode()

Configures the specified pin to behave either as an input or an output

Syntax

pinMode(pin, mode)

Parameters

pin: the number of the pin whose mode you wish to set

mode: INPUT, OUTPUT

digitalWrite()

Write a HIGH or a LOW value to a digital pin.

If the pin has been configured as an OUTPUT with pinMode(), its voltage will be set to the corresponding value: 5V (or 3.3V on 3.3V boards) for HIGH, 0V (ground) for LOW

Syntax

digitalWrite(pin, value)

Parameters

pin: the pin number

value: HIGH or LOW

digitalRead()

Description

Reads the value from a specified digital pin, either HIGH or LOW.

Syntax

digitalRead(pin)

Parameters

pin: the number of the digital pin you want to read

Returns

HIGH or LOW

analogRead()

Reads the value from the specified analog pin.

Syntax

analogRead(pin)

Parameters

pin: the number of the analog input pin to read from (0 to 5 on most boards, 0 to 7 on the Mini and Nano, 0 to 15 on the Mega)

Returns

int(0 to 1023)