SOFTWARE TUTORIAL

To control the Arduino we need an IDE (Integrated Development Environment). You can download Arduino IDE's latest version from https://www.arduino.cc.

1- Basic Arduino Functions

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

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.

Note that Serial.begin(9600) has 9600 baud rate. On the Serial Port monitor you must select the same baud as well. ***begin()*** : Sets the data rate in bits per second (baud) for serial data transmission. For communicating with the computer, use one of these rates: 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600, or 115200. You can, however, specify other rates - for example, to communicate over pins 0 and 1 with a component that requires a particular baud rate.

print() : Prints data to the serial port as human-readable ASCII text. This command can take many forms. Numbers are printed using an ASCII character for each digit. Floats are similarly printed as ASCII digits, defaulting to two decimal places. Bytes are sent as a single character. The ***println()*** method prints the string and moves the cursor to a new line. The ***print()*** method instead prints just the string , but does not move the cursor to a new line. Hence, subsequent printing instructions will print on the same line. The ***println()*** method can also be used without parameters, to position the cursor on the next line.

delay() : Pauses the program for the amount of time (in milliseconds) specified as parameter. (There are 1000 milliseconds in a second.)

pinMode() : Configures the specified pin to behave either as an input or an output.

Syntax

```
pinMode(pin, mode)
```

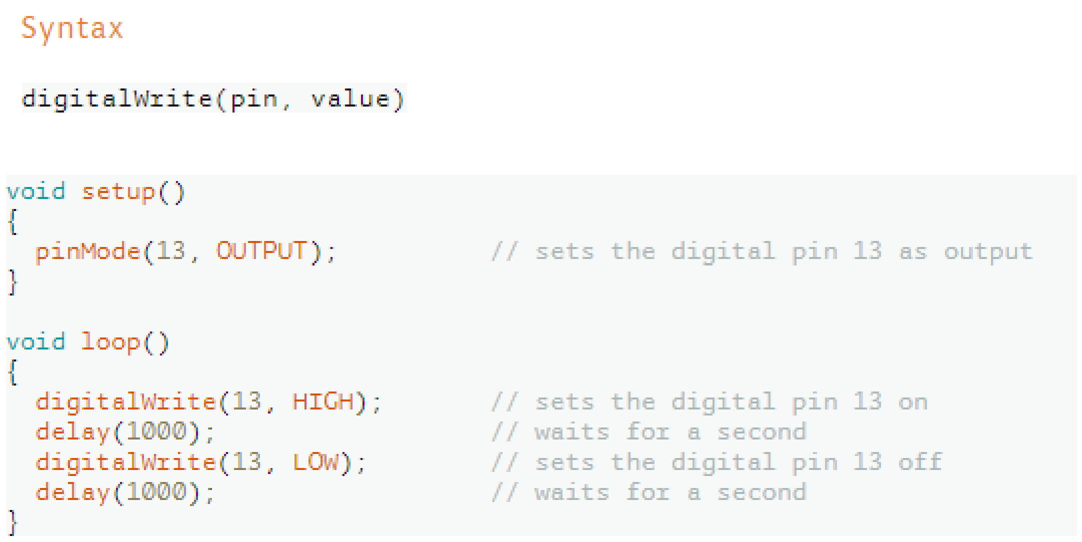
digitalRead():Reads the value from a specified digital pin, either HIGH or LOW.

Syntax

```
digitalRead(pin)
```

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.If the pin is configured as an INPUT, digitalWrite() will enable (HIGH) or disable (LOW) the internal pullup on the input pin. It is recommended to set the pinMode() to INPUT_PULLUP to enable the internal pull-up resistor. See the digital pins tutorial for more information.

If you do not set the pinMode() to OUTPUT, and connect an LED to a pin, when calling digitalWrite(HIGH), the LED may appear dim. Without explicitly setting pinMode(), digitalWrite() will have enabled the internal pull-up resistor, which acts like a large current-limiting resistor.



Syntax

```
digitalWrite(pin, value)
```

```
void setup()
{
  pinMode(13, OUTPUT);          // sets the digital pin 13 as output
}

void loop()
{
  digitalWrite(13, HIGH);       // sets the digital pin 13 on
  delay(1000);                  // waits for a second
  digitalWrite(13, LOW);        // sets the digital pin 13 off
  delay(1000);                  // waits for a second
}
```

analogRead() : Reads the value from the specified analog pin. Arduino boards contain a multichannel, 10-bit analog to digital converter. This means that it will map input voltages between 0 and the operating voltage (5V or 3.3V) into integer values between 0 and 1023. On an Arduino UNO, for example, this yields a resolution between readings of: 5 volts / 1024 units or, 0.0049 volts (4.9 mV) per unit.

Syntax

`analogRead(pin)`

```
int analogPin = A3; // potentiometer wiper (middle termi
                    // outside leads to ground and +5V
int val = 0;  // variable to store the value read

void setup() {
  Serial.begin(9600);           //  setup serial
}

void loop() {
  val = analogRead(analogPin);  // read the input pin
  Serial.println(val);          // debug value
}
```

2- Basic Coding Statements

if()…else

The if…else allows greater control over the flow of code than the basic if statement, by allowing multiple tests to be grouped together. An else clause (if at all exists) will be executed if the condition in the if statement results in false. The else can proceed another if test, so that multiple, mutually exclusive tests can be run at the same time. Each test will proceed to the next one until a true test is encountered. When a true test is found, its associated block of code is run, and the program then skips to the line following the entire if/else construction. If no test proves to be true, the default else block is executed, if one is present, and sets the default behavior. Note that an else if block may be used with or without a terminating else block and vice versa. An unlimited number of else if branches are allowed.

Syntax

```
if (condition1)
{
  // do Thing A
}
else if (condition2)
{
  // do Thing B
}
else
{
  // do Thing C
}
```

Comparison Operators:

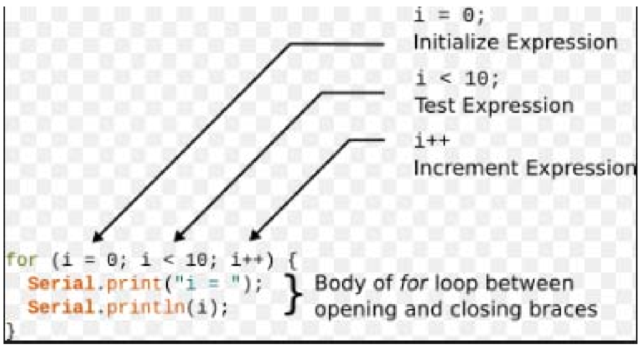
```
x == y (x is equal to y)
x != y (x is not equal to y)
x <  y (x is less than y)
x >  y (x is greater than y)
x <= y (x is less than or equal to y)
x >= y (x is greater than or equal to y)
```

for()

The for statement is used to repeat a block of statements enclosed in curly braces. An increment counter is usually used to increment and terminate the loop. The for statement is useful for any repetitive operation, and is often used in combination with arrays to operate on collections of data/pins.

Syntax

```
for (initialization; condition; increment) {
        //statement(s);
}
```



```
for (i = 0; i < 10; i++) {
  Serial.print("i = ");
  Serial.println(i);
}
```

while()

A while loop 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. This could be in your code, such as an incremented variable, or an external condition, such as testing a sensor.

Syntax

```
while(condition){
  // statement(s)
}
```

Example Code

```
var = 0;
while(var < 200){
  // do something repetitive 200 times
  var++;
}
```

array[]

An array is a collection of variables that are accessed with an index number. Creating an array:

```
  int myInts[6];
  int myPins[] = {2, 4, 8, 3, 6};
  int mySensVals[6] = {2, 4, -8, 3, 2};
  char message[6] = "hello";
```

You can declare an array without initializing it as in myInts. In myPins we declare an array without explicitly choosing a size. The compiler counts the elements and creates an array of the appropriate size. Finally you can both initialize and size your array, as in mySensVals. Note that when declaring an array of type char, one more element than your initialization is required, to hold the required null character.

int

Integers are your primary data-type for number storage. On the Arduino Uno (and other ATmega based boards) an 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). On the Arduino Due and SAMD based boards (like MKR1000 and Zero), 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). int’s store negative numbers with a technique called (2’s complement math). The highest bit, sometimes referred to as the "sign" bit, flags the number as a negative number. The rest of the bits are inverted and 1 is added. The Arduino takes care of dealing with negative numbers for you, so that arithmetic operations work transparently in the expected manner. There can be an unexpected complication in dealing with the bitshift right operator (>>) however.

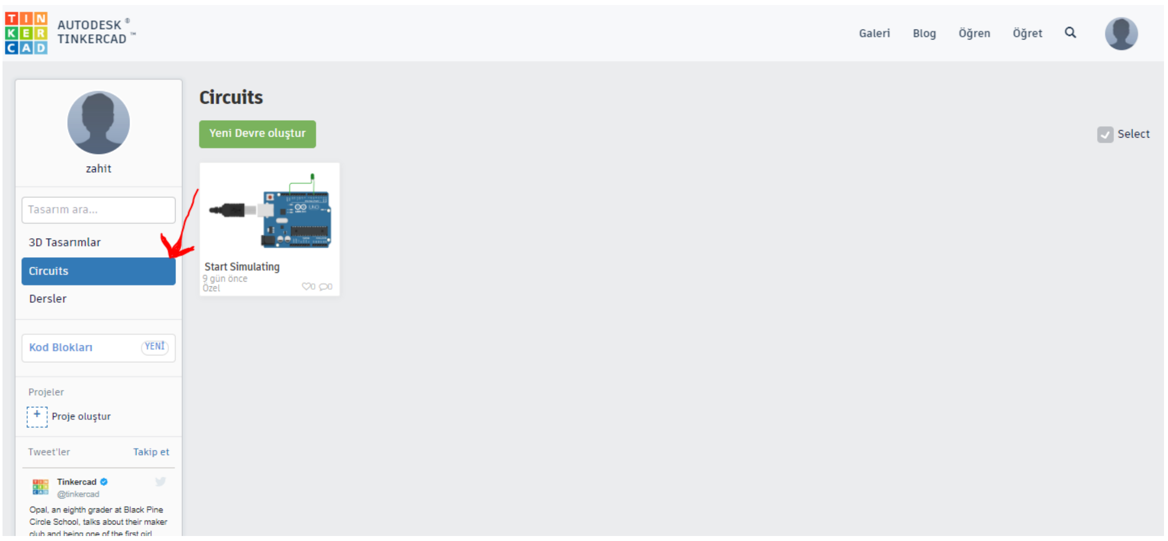
Example Code

```
int countUp = 0;            //creates a variable integer called 'countUp'

void setup() {
  Serial.begin(9600);       // use the serial port to print the number
}

void loop() {
  countUp++;                //Adds 1 to the countUp int on every loop
  Serial.println(countUp);  // prints out the current state of countUp
  delay(1000);
}
```

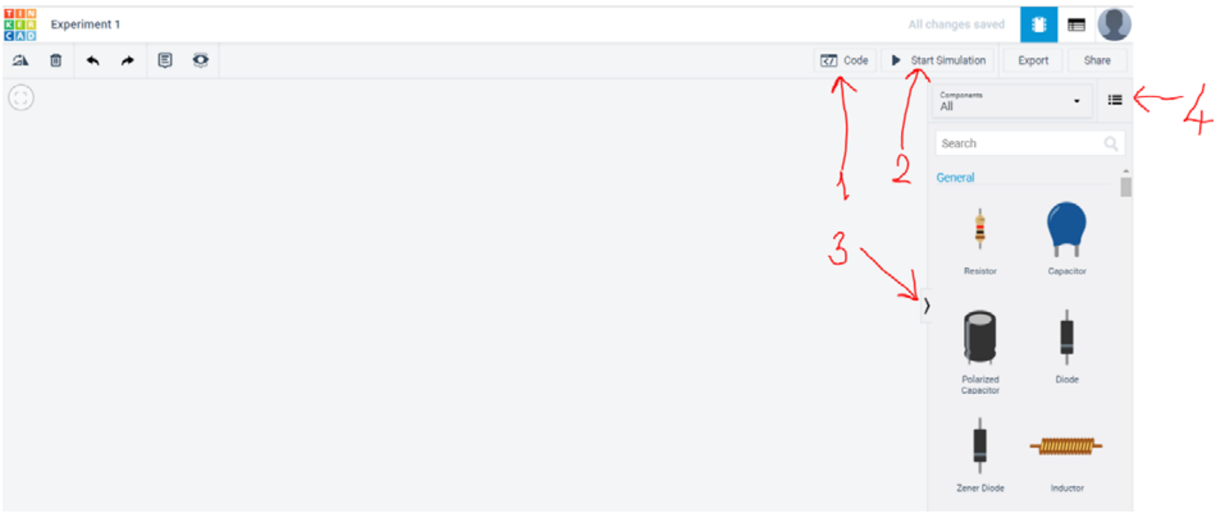
3- TinkerCad

TinkerCad is a simple, online 3D design and 3D printing software. It can also be used for electronics simulation using circuits.

Before starting, you need to create an account online by visiting the website www.tinkercad.com



Click on Circuits to switch from 3D Designs to Ciruits mode, and then click on Create new Circuit.



The section of a simple Tinkercad simulation consists of four parts as marked in the above figure and the explanation of each section is as follows.

1- In the “Code” section, you can write your codes.
2- After designing your circuit you can simulate it by clicking the “Start Simulation” button.
3- Arduino, Multimeter and electrical components can be picked from this by clicking this button.
4- You can read information about each component by clicking this button.