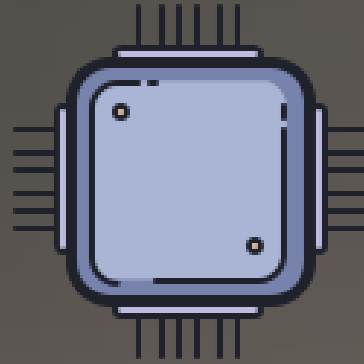
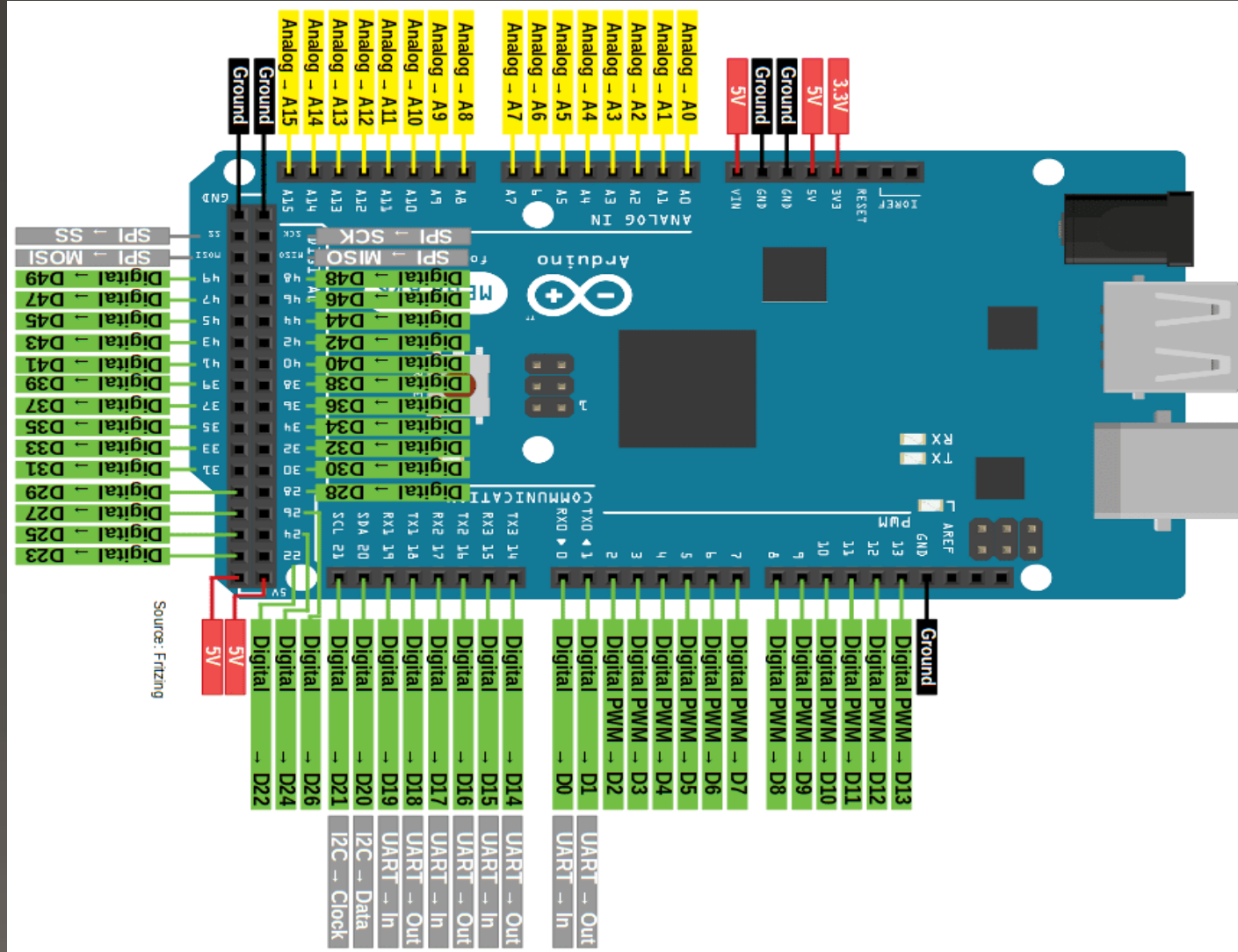


Embedded Systems

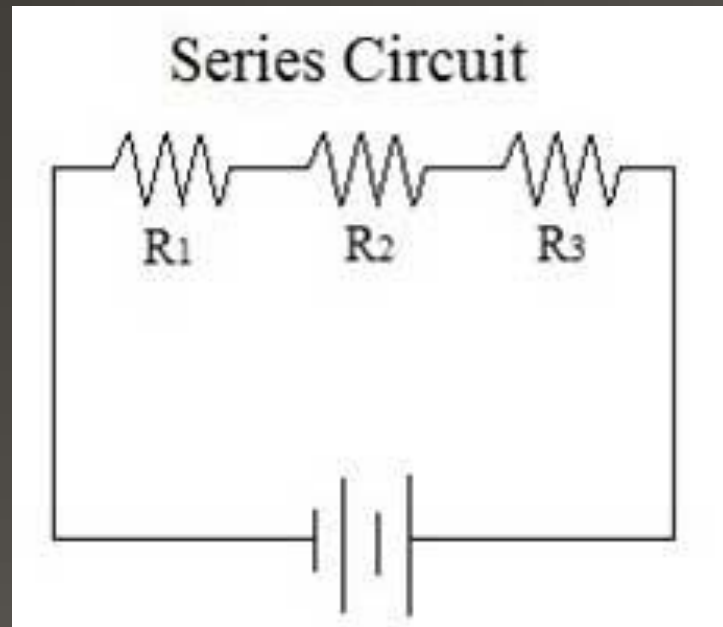


Low Speed Self-driving Vehicles- ITI

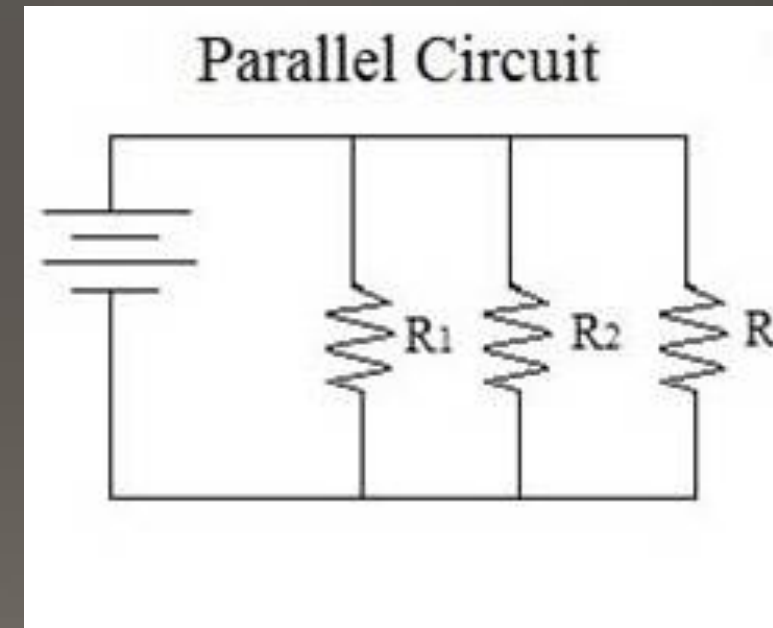
Input & output pins (I/O)



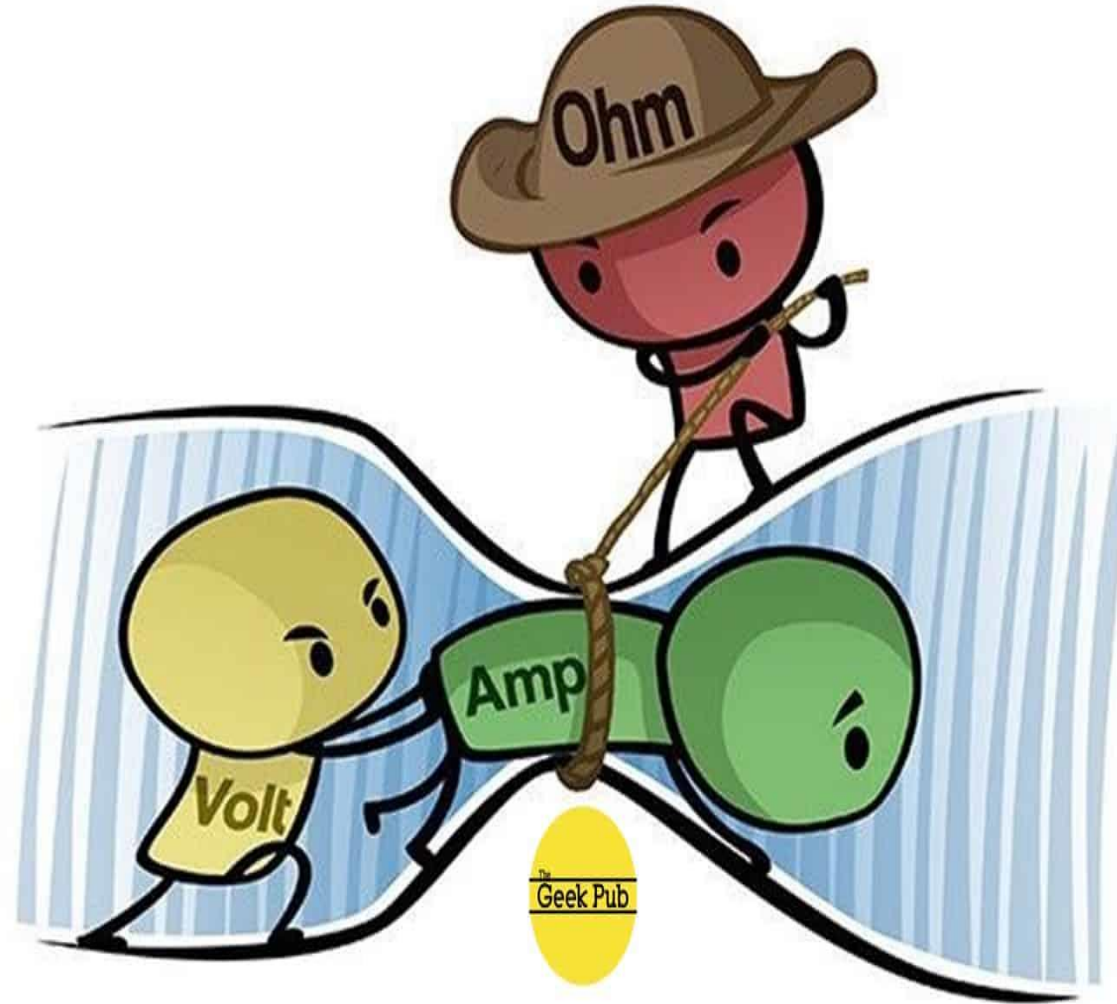
“series” connection is that components are connected end-to-end in a line to form a single path through which current can flow



“parallel” connection, on the other hand, is that all components are connected across each other’s leads.

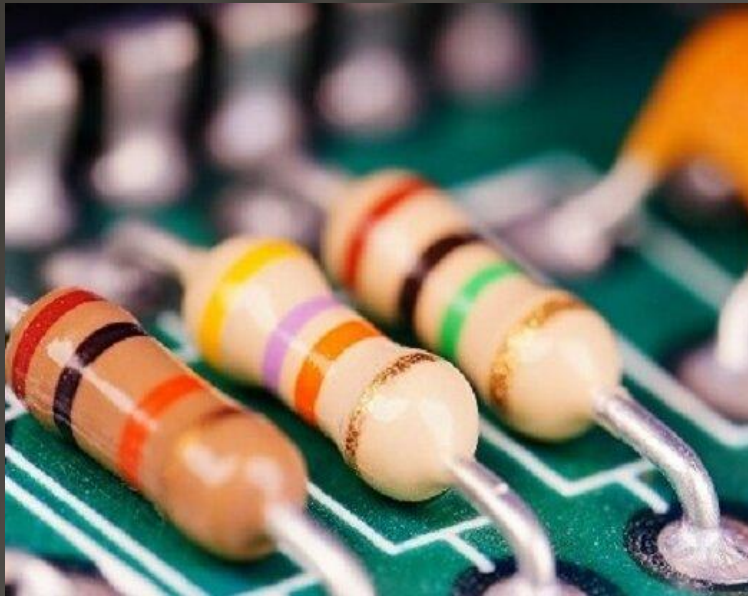


Resistance



Type of Resistance

Fixed Value



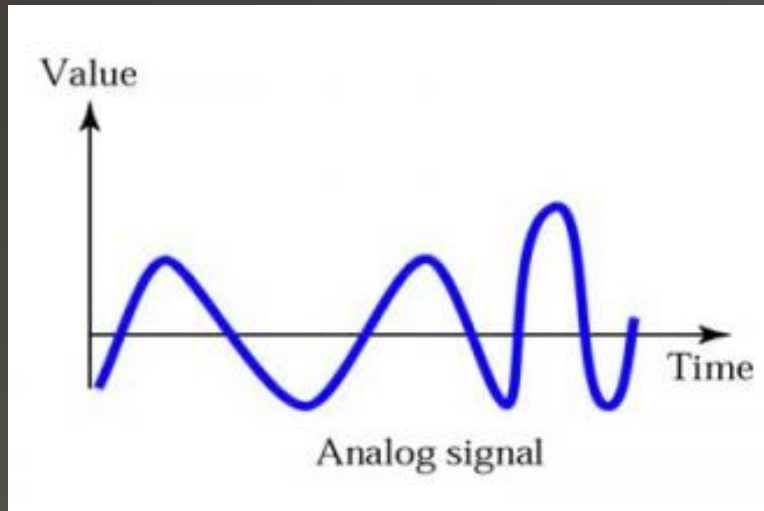
Variable resistance



Analog Vs Digital

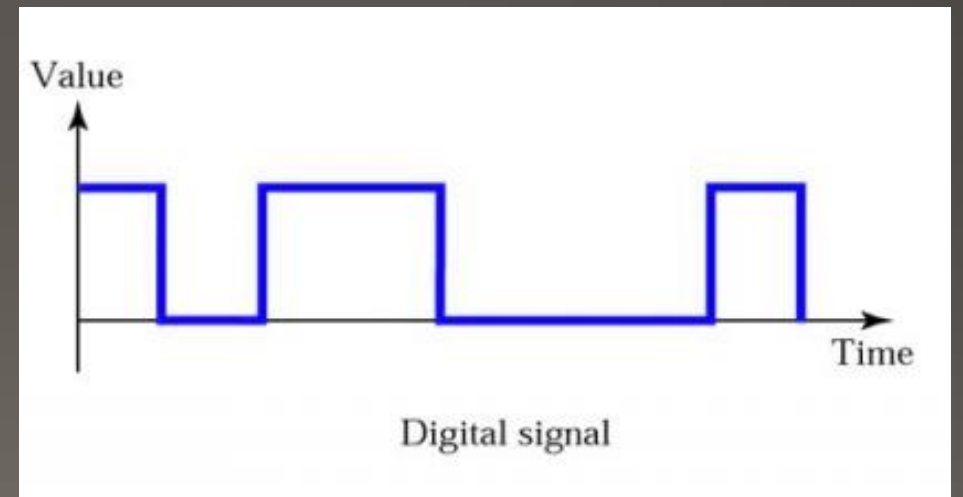
Analog signal is a continuous signal which represents physical measurements.

Denoted by sine waves

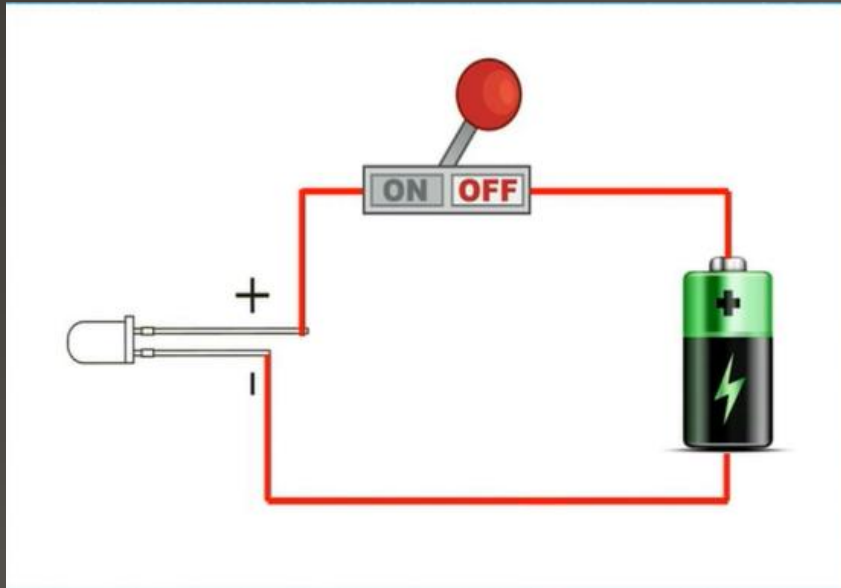


Digital signals are discrete time signals generated by digital modulation.

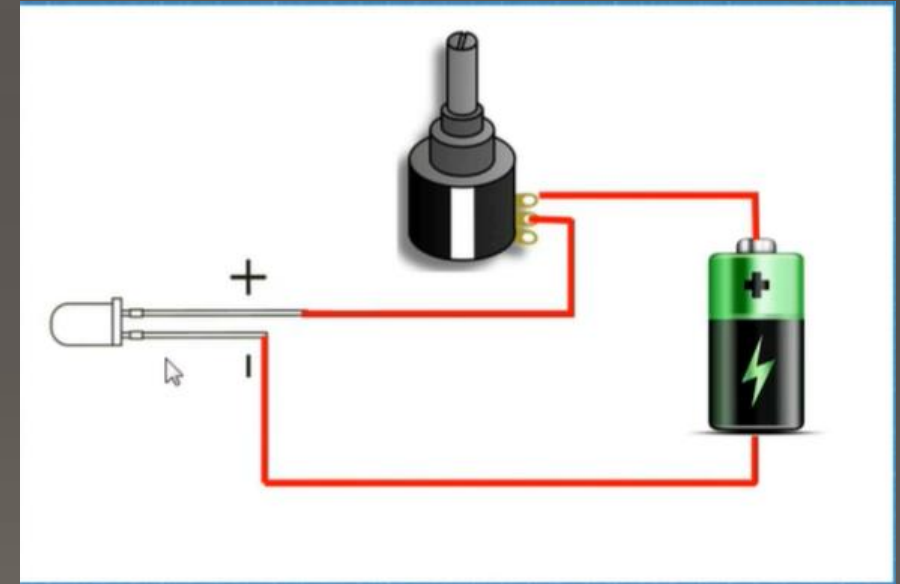
Denoted by square waves



Digital Circuit (On – Off)



Analog Circuit



Open Circuit

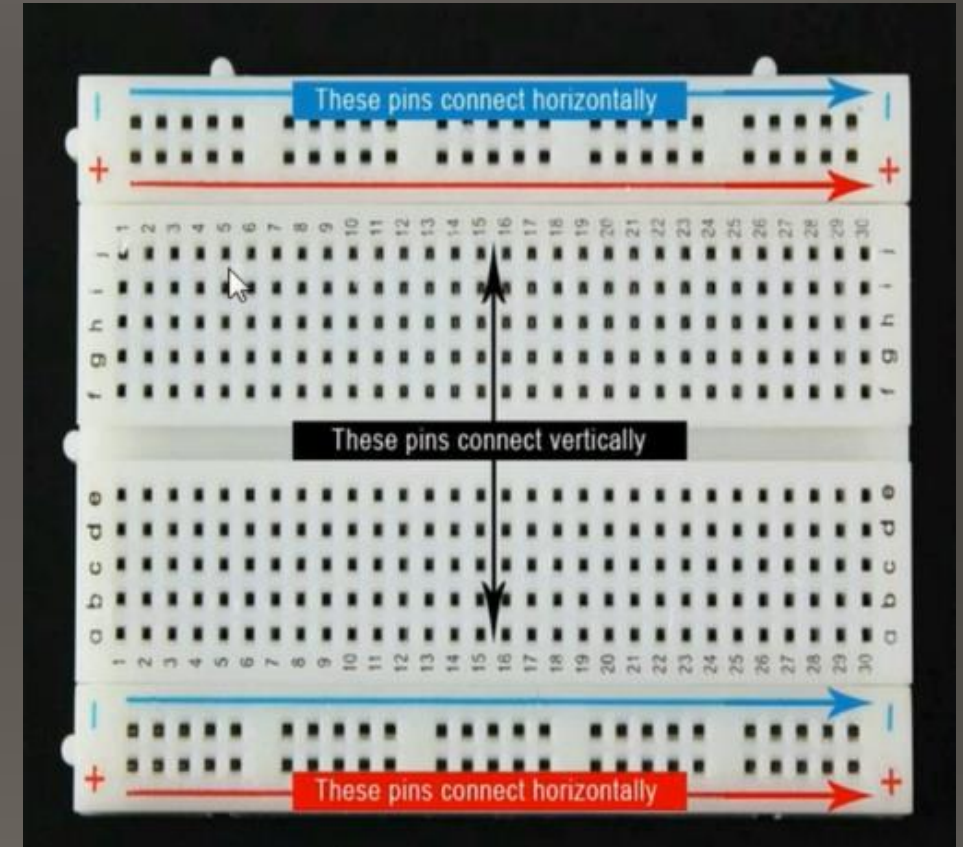


Closed Circuit



A very important item when working Arduino is a solderless breadboard.

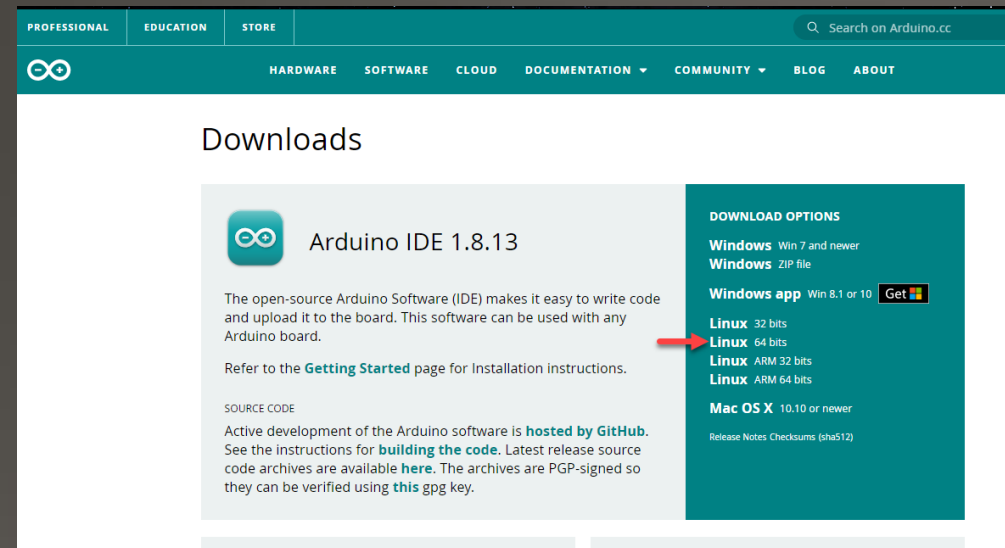
This device allows you to prototype your Arduino project without having to permanently solder the circuit together. Using a breadboard allows you to create temporary prototypes and experiment with different circuit designs. Inside the holes (tie points) of the plastic housing, are metal clips which are connected to each other by strips of conductive material.



Introduction To Arduino

setup and installation

<https://www.arduino.cc/en/software>



```
sudo ./install.sh
```






```
./arduino
```

```
sudo chmod a+rw /dev/ttyACMo
```



Arduino – Program Structure

IDE Controls

-  **Verify** : Checks your Code for errors compiling it .
-  **Upload** : Compiles your code and upload it to the configured board.
-  **New** : Creates a new sketch.
-  **Open** : Present a menu of all the sketches in your sketchbook.
-  **Save** : Saves your sketch.



Arduino – Program Structure

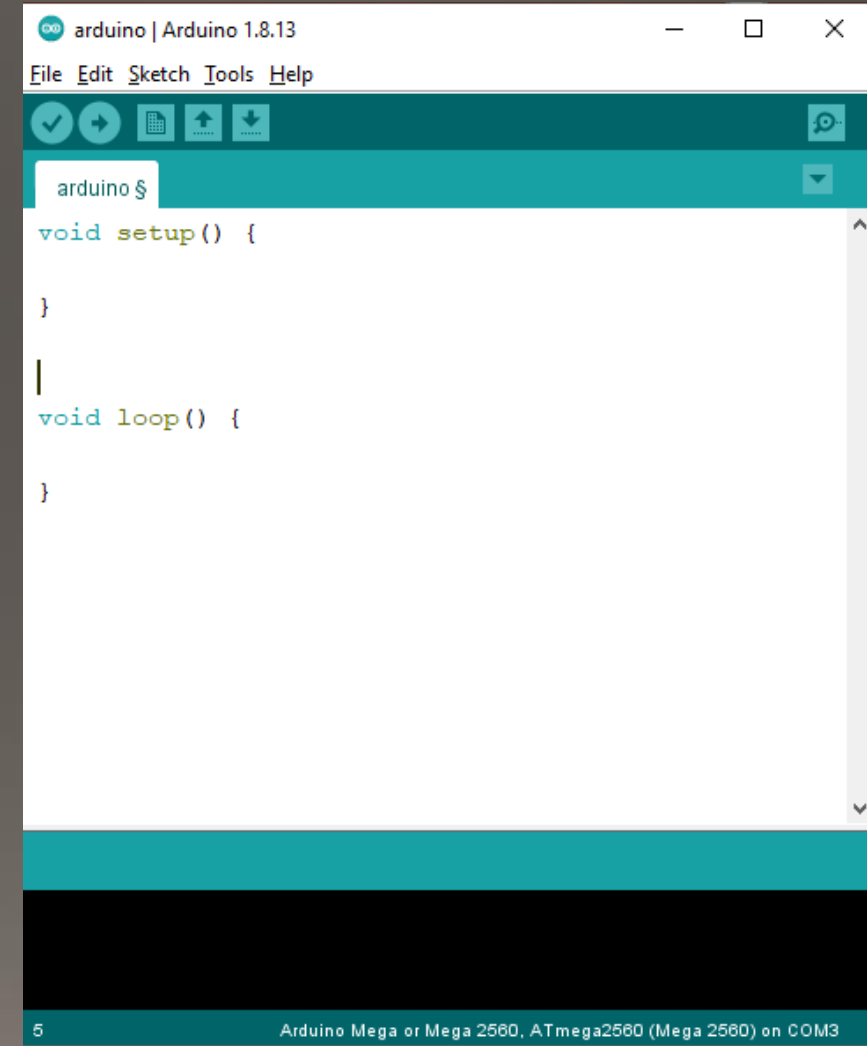
❑ Software Structure consist of two main Functions:-

❑ Void setup(){ }

Void setup is technically a function that you create at the top of each program. Inside the curly brackets is the code that you want to run one time as soon as the program starts running. You set things like pinMode in this section.

❑ Void Loop(){ }

The loop is another function that Arduino uses as a part of its structure. The code inside the loop function runs over and over as long as the Maker Board is turned on.



```

arduino | Arduino 1.8.13
File Edit Sketch Tools Help

[Icons: Checkmark, Arrow, Grid, Upload, Download, Search]

arduino $
void setup() {

}

|
void loop() {

}

5 Arduino Mega or Mega 2560, ATmega2560 (Mega 2560) on COM3
    
```



Arduino – I/O Functions

□ pinMode() Function

The pinMode() Function is used to configure a specific pin to behave either as an input or an output . It is possible to enable the internal pull-up resistors with the mode INPUT_PULLUP.

Additionally , the INPUT mode explicitly disables the internal pull-ups.

```
Void setup (){  
  pinMode(pin,mode);  
}
```

Pin – the number of pin whose mode you wish to set .

Mode – INPUT , OUTPUT , or INPUT_PULLUP



□digitalWrite() Function

- Digital write works with pins set as OUTPUTs in pinMode(). It's a function that sends either 5V or 0V to the pin you supply it. This function only has two settings: on or off. To specify on or off, though, you can use the words HIGH or LOW, true or false, 1 or 0.

Void loop ()

```
digitalWrite(pin,value);  
}
```

Pin – the number of pin whose mode you wish to set .

Value – High or Low



□ analogRead() Function

- Using analogRead() is very similar to digitalWrite(), but you should expect different results. Analog signals exist on a scale. With a Maker Board, that scale is from 0 to 1024. You can use analogRead along with analog pins that are set as pinMode(INPUT) or pinMode(INPUT_PULLUP).

Void loop ()

```
analogRead(pin);  
}
```

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)



□digitalRead() Function

- Arduino is able to detect whether there is a voltage applied to one of its pins and report it through the digitalRead() function. Read the value from a specified digital pin, either HIGH, LOW.

Void loop ()

digitalRead(pin);

}

Pin – the number of the digital input pin to read from.



Arduino – Variables

- ❑ **Variables** are used to store information to be referenced and manipulated in a computer program.
- ❑ **Variable Scope** : 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** .
- Outside of all functions , which is called **global variables** .



Arduino – Data types

❑ **Void** the 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(){}.

❑ **Boolean** A Boolean variable occupies one byte of memory

Example

```
boolean val = false;
boolean state = true;
```

Datatype	RAM usage
void keyword	N/A
boolean	1 byte
char	1 byte
unsigned char	1 byte
int	2 byte
unsigned int	2 byte
word	2 byte
long	4 byte
unsigned long	4 byte
float	4 byte
double	4 byte



Arduino – Data types

❑ **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'

Example

```
char char_a='a';
```

```
Void loop(){}.
```

❑ **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 char_y = 121;
```

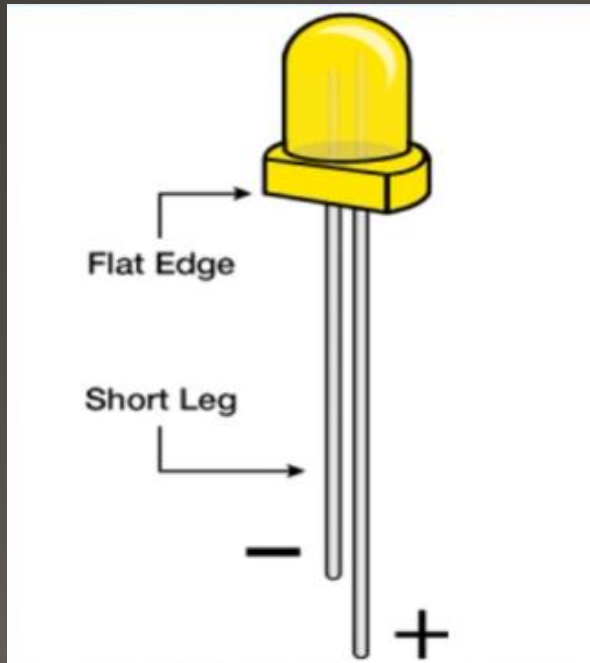
Datatype	RAM usage
void keyword	N/A
boolean	1 byte
char	1 byte
unsigned char	1 byte
int	2 byte
unsigned int	2 byte
word	2 byte
long	4 byte
unsigned long	4 byte
float	4 byte
double	4 byte



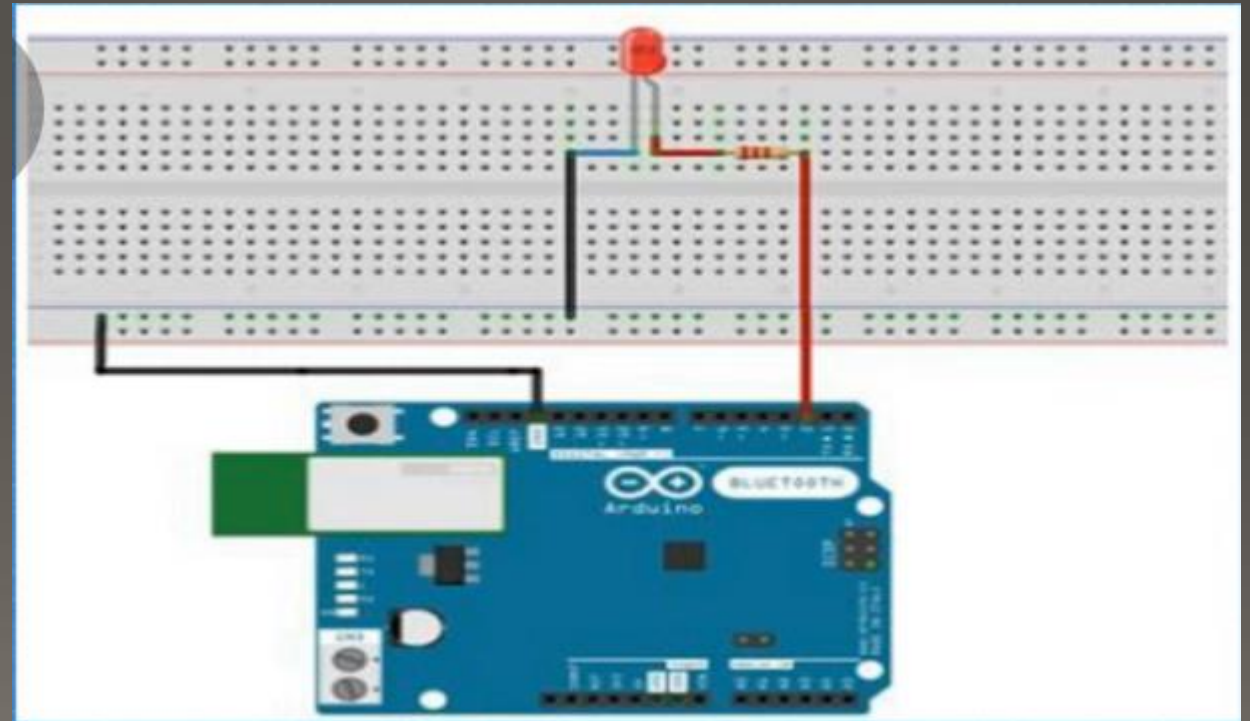
Arduino – Blinking LED

- LEDs are small, powerful lights that are used in many different applications.
- To start, we will work on blinking an LED, the Hello World of microcontrollers. It is as simple as turning a light on and off.

LED

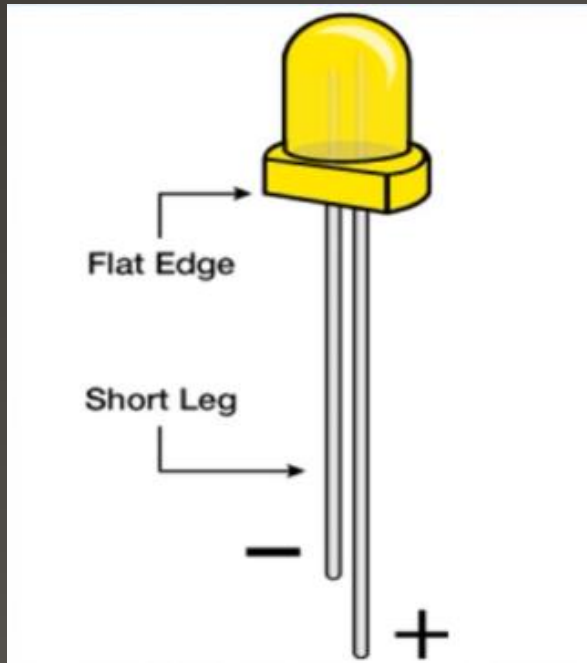


Arduino Connections

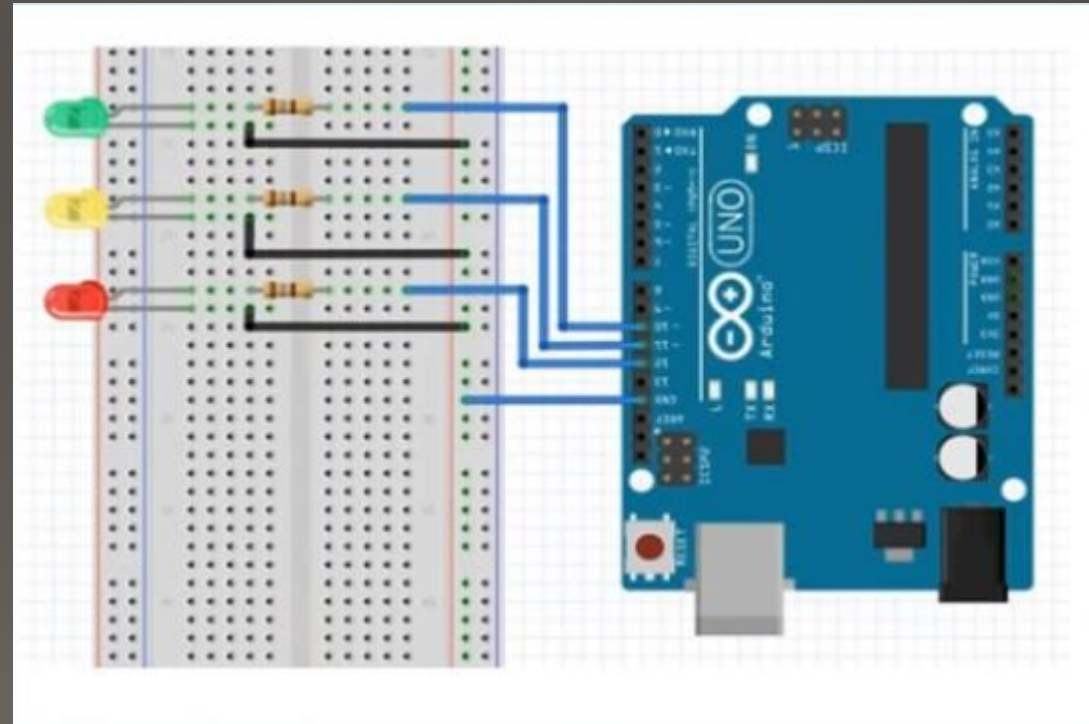


Arduino – traffic light

LED



Arduino Connections



Code

```
int red = 10;
int yellow = 9;
int green = 8;

void setup()
{
    pinMode(red, OUTPUT);
    pinMode(yellow, OUTPUT);
    pinMode(green, OUTPUT);
}
```

```
void loop(){
    digitalWrite(green, LOW);
    digitalWrite(red, HIGH);
    delay(2000);
    digitalWrite(red, LOW);
    digitalWrite(yellow, HIGH);
    delay(1000);
    digitalWrite(yellow, LOW);
    digitalWrite(green, HIGH);
    delay(2000);
}
```



Arduino – count from 1 to 10 and print them on the screen using while loop

```
Int counter =1;  
Voide setup(){  
  Serial.begin(9600);  
}  
Void loop()  
{  
  While (counter <= 10 )  
  {  
    Serial.print("Counting from 1 to 10 : ");  
    Serial.println(counter);  
    Counter ++;  
  }  
  Serial.println("done counting");  
  delay(2000);  
}
```



Arduino – count from 10 to 0 and print them on the screen using for loop

```
Int counter =10;  
Voide setup(){  
  Serial.begin(9600);  
  Serial.println("Counting from 10 to 0 : ");  
}  
Void loop()  
{  
  for(counter ; counter >=0;counter --){  
    Serial.println(counter);  
  }  
}  
Serial.println("done counting");  
delay(2000);  
}
```



Arduino – prints from A to Z on the screen using for loop

```
char counter =0;
Void setup(){
  Serial.begin(9600);
  Serial.println("Printing from A to Z : ");
}
Void loop()
{
  for(counter='A' ; counter <='Z';counter++){
    Serial.println(counter);
  }
}
Serial.println("done counting");
delay(2000);
}
```



Arduino – program that takes the score from user and prints the user grade

```
Int score =0;
Voide setup(){
  Serial.begin(9600);
  Serial.println("Please Enter Your Score: ");
}
Void loop()
{
  if (Serial.available()){
    score=Serial.read();
    switch(score)
    {
      case '9':
        Serial.println("Your Grade is : A");
        break;
      case '6':
      case '5': Serial.println("Your Grade is : D"); break;
    }
  }
}
```



Arduino – Functions

```
void setup ()
{
    Statements // group of statements
}
Void loop ()
{
    int result = 0 ;
    result = Sum_func (5,6) ; // function call
    Serial.print("Result =");
    Serial.println(result);
}
int sum_func (int x, int y) // function declaration
{
    int z = 0;
    z = x+y ;
    return z; // return the value
}
```



```
//Frist Program

const int LED = 13;
void setup ( )
{
  pinMode(LED, OUTPUT);
}

void loop()
{
  digitalWrite(LED, HIGH);
  delay(1000);
  digitalWrite(LED, LOW);
  delay(1000);
}
```



SYNTAX

The **const** keyword stands for constant. It is a variable qualifier that modifies the behavior of the variable, making a variable "read-only".

void setup(){ }

Void setup is technically a function that you create at the top of each program. Inside the curly brackets is the code that you want to run one time as soon as the program starts running. You set things like pinMode in this section.

void loop(){ }

The loop is another function that Arduino uses as a part of its structure. The code inside the loop function runs over and over as long as the Maker Board is turned on.

;(semicolon)

The end of a command or statement. The compiler doesn't look for spaces in your code, it looks for semicolons.



`{ }` curly braces

A group of code statements. You always need a closing curly brace to match each opening curly brace.

`//`single line comment

When you type two forward slashes, the code from that point until the end of that line is ignored by the compiler. Use comments to explain what your code does, help you remember how the hardware is hooked up, or to remove some code from the program without deleting it.

`/*` multi-line comment `*/`

A multi-line comment starts with the `/*` and doesn't end until the `*/` characters. When you have a lot to type, use a multi-line comment to make it easier to group the information together.



#define button 2

the #define command is used for a 'find and replace within your code. It is similar to creating a variable to replace a number, but it doesn't take up any memory. The downside of the #define is that, using the example above, a variable called buttonPress becomes 2 Press when the code is compiled.

#include <library.h> or #include "library.h"

In the functions section of this text, we spoke about many of the built-in Arduino functions and how useful they can be to simplify your code and make hardware easier to interface with. Libraries are files that you can include in your code to use even more methods and functions. Anyone can write a library, so there are thousands of Arduino libraries available to you.



HARDWARE FUNCTIONS

Arduino has a number of functions built in to help you control hardware. They give you a way to simply control the pins.

pinMode(pin, INPUT/OUTPUT/INPUT_PULLUP);

Pin mode sets up your hardware pin to receive power, allow power to flow through, or allow a resisted amount of power to flow through. Each time you use it, you have to supply the pin number you're addressing and the setting you want for that pin.

digitalWrite(pin, HIGH/LOW);

Digital write works with pins set as OUTPUTs in pinMode(). It's a function that sends either 5V or 0V to the pin you supply it. This function only has two settings: on or off. To specify on or off, though, you can use the words HIGH or LOW, true or false, 1 or 0.



digitalRead(pin);

The digitalRead() function only takes one argument: the pin number you want to read. Its job is to listen to the pin that the programmer provides and report back what its voltage level is. The voltage level that digitalRead sends back is often a trigger for a button press or other input and is used with pinMode(INPUT) and pinMode(INPUT_PULLUP) pins.

analogRead(pin);

Using analogRead() is very similar to digitalRead(), but you should expect different results. Analog signals exist on a scale. With a Maker Board, that scale is from 0 to 1024. You can use analogRead along with analog pins that are set as pinMode(INPUT) or pinMode(INPUT_PULLUP).



analogWrite(pin, value);

analogWrite sends a value between 0 and 255 to a pin with the pinMode(OUTPUT). 255 is completely on, meaning that the pin is receiving full power 100% of the time. Passing analogWrite() a value of 128 for the value, however, means that the pin is receiving full power 50% of the time. The power flickers on and off so quickly that the output (like a motor or LED) appears to be only half way powered.

tone(pin, freqhz);

Tone is a more specialized function created to work with speakers and buzzers. You supply it with a pin argument and a frequency argument measured in hertz. Using a method similar to analogWrite, tone sends a signal to the pin that creates your sound!

noTone(pin);

noTone only takes one argument: the pin you'd like to stop sending a tone to. It's not very interesting by itself, but you'll need it to stop making noise!



TIME FUNCTIONS

millis();

Millis is a built-in Arduino function that doesn't have any arguments. When you call millis, you simply type millis() and it returns to you the number of milliseconds since your program started running. Although not very useful by itself, millis() is really helpful when you want to make things happen at a certain time without delaying your program. It will run for about 50 days before the variable type can't hold any more milliseconds and will overflow back to zero.

delay(milliseconds);

The delay function pauses your program for the number of milliseconds you supply it as an argument. Because the Maker Board is running the code very quickly, you'll often want to slow things down by pausing the progress of the code for a few seconds.



MATH FUNCTIONS

`min(x, y); max(x, y);`

The min function takes in two values and returns the smaller of the two. Max returns the larger of the two. You can also enter variables as arguments and you can enter two different data types (like an int and a float) and these functions still work. When a function can accept multiple data types, it's called **overloaded**.

`abs(x);`

abs stands for absolute value, which is what this function returns when you supply it a number. You can supply it a negative or positive number, a variable with a number value, or a math equation and it will return back to you the whole number and up to two decimal places.



variable ++; variable --

Increment or decrement a variable. The longer version of this command might look something like: $(x = x + 1;)$ or $(y = x - 1;)$. When assigning the value of the incremented/decremented variable, to another variable, the placement of the $(++ / --)$ will generate a different result as it is passed to the receiving variable.

map(value, low, high, toLow, toHigh);

The map function is usually supplied a variable value or a sensor reading as the value, then the range you expect that value to have. For example you may expect light readings between 250 and 850. The last two values supplied to the function are the new scale you want to apply to the readings.

Map seems a little strange until you think of using it. Imagine you want to display the temperature on a strip of 10 LED lights. The temperature will probably range more than from 0 to 10 degrees, so you can map the temperature values of 50-85 degrees to the LED values of 0-10.



constrain (value, lowest_value, highest_value);

Constrain adds a floor and a ceiling to the value that a variable or reading can possibly have. First, you supply it the value you want to constrain. Then you give that value a floor and a ceiling. To use the constrained value in your code, it has to have a name, so you'll often see constrain used like this:

```
int reading = (reading, 100,200);
```

Constrain is useful when the value of your inputs can range more than the value of your outputs. You'll often see it used near a map command because you may want to change the scale of your variable and then limit it's range for an output. For example you may map a temperature range of 50-100 to a brightness of 0-255. Then you may constrain the temp to 50-100 so a temperature of 49 doesn't make strange things happen with your code.



`random(max); random(min, max);`

`random()` can be used when you want to add a little surprise inside your code or generate a number without knowing exactly what it will be. This function has two arguments, but one of them is optional. If you enter only one argument, the minimum number `random` will return is 0 and the maximum number is $(\text{max} - 1)$. If you enter two arguments, `random()` knows that the first argument is the minimum and the second is the max.



COMMUNICATION FUNCTIONS

The serial monitor is a window into what's happening on the Maker Board. It's a **class** with **methods**, which is why you use “Serial.” in front of the command. Printing information to the serial port will allow messages, variable values, or code checkpoints to show up on your screen.

Serial.begin(9600);

The serial port is not running by default, so use the .begin() method to start it in the void setup() section of your code. The argument 9600 has to do with the speed of communication between the Maker Board and the serial port. 9600 is the default. When you open the serial monitor by pressing the magnifying glass icon in the IDE, be sure that the bottom right-hand corner is also displaying 9600. Otherwise, the information is garbled.



```
Serial.print( "Message" );
```

```
Serial.print(variable);
```

```
Serial.print(":");
```

Once the serial port has been started, you can use the `Serial.print()` method to send strings of text or variable values to the serial monitor and watch them on the screen. If you want to show a friend instructions to play a game that you created with code, you could use `Serial.print("Here's how to play:")` at the end of your `setup()` and print the instructions to the serial monitor. Remember that Maker Board has to be plugged in to the computer to be able to send any data.



`Serial.println("message");`

`Serial.println()` is a method similar to `Serial.print` except that it starts a new line after each message. This is handy when you're printing many values and you don't want them to print in a straight line with no spaces.

CREATING YOUR OWN FUNCTIONS

```
void blinkLight(byte _pin) {  
  digitalWrite(_pin , HIGH);  
  delay(500);  
  digitalWrite(_pin, LOW);  
}
```



CONSTANTS

HIGH or LOW

When used with `pinMode(OUTPUT)` and `digitalWrite()`, HIGH means 5V is sent to the pin you specify. LOW means 0V is sent to the pin.

When used with `pinMode(INPUT)` or `pinMode(INPUT_PULLUP)` and `digitalRead()`, HIGH means that the pin is reporting at least 3 volts. LOW means the pin is reporting less than 3 volts.

true or false

The word false is converted to a 0 in the code. The word true is actually defined as all 'non-zero' numbers. You can do math with true and false. For example, $(\text{false} - \text{true})$ is equal to -1.



VARIABLES

int variable = -32,767 to 32,767; int stand for integer

bool variable = true (1) / false (0) bool stands for boolean

char variable = -128 to 127; char is usually pronounced like character

byte variable = 0 to 255;

unsigned int variable = 0 to 65,535;

long variable = -2,147,483,647 to 2,147,483,647;

unsigned long variable = 0 to 4,294,967,295;

float variable = - 3.4028235E+38 to 3.4028235E+38;



Arrays

type_of_variables name_of_array[number_of_variables_in_array] = { array values }

```
int guesses[7];
```

```
byte months[] = {0, 1, 2, 3, 4};
```

```
char variable[6] = {0, 1, 2, 3, 4};
```



STATEMENTS

if- else if -else

if(condition here is true){ run all of these code statements } else{ run these statements }

switch case

```
switch (var){  
case label:  
// statements
```

```
break;  
case label:  
// statements
```

```
break;  
default: (optional)  
// statements  
}
```



An example using switch to decide which pin is high based on the number of button presses:

```
switch(buttonPresses){  
  
  case 1:  
    digitalWrite(12,HIGH);  
    break;  
  case 2:  
    digitalWrite(10,HIGH);  
    break;  
  case 3:  
    digitalWrite(9,HIGH);  
    break;  
  //many more case statements can go in here...  
  default:  
    digitalWrite(2,HIGH);
```



ITERATION STATEMENTS

for loop

for(create_and_declare_variable ; condition_based_on_that_variable ;
modify_variable_here_until_condition_is_false){ do this until the second statement in
the for loop is false }

example:

```
for( int i = 0 ; i < 200 ; i ++){  
  analogWrite( 9, i );  
  delay(100);  
}
```



while loop

```
while( this_condition_is_true ){ run_this_code }
```

```
while ( brightness > 0 ){  
  analogWrite(3, (brightness -1));  
  delay(100);  
  brightness - - ;  
}
```

