



ARDUINO COURSE GUIDE



What is Programming Languages?

A programming language is a formal language, which comprises a set of instructions that produce various kinds of output. Programming languages are used in computer programming to implement algorithms.

Arduino Programming Language:

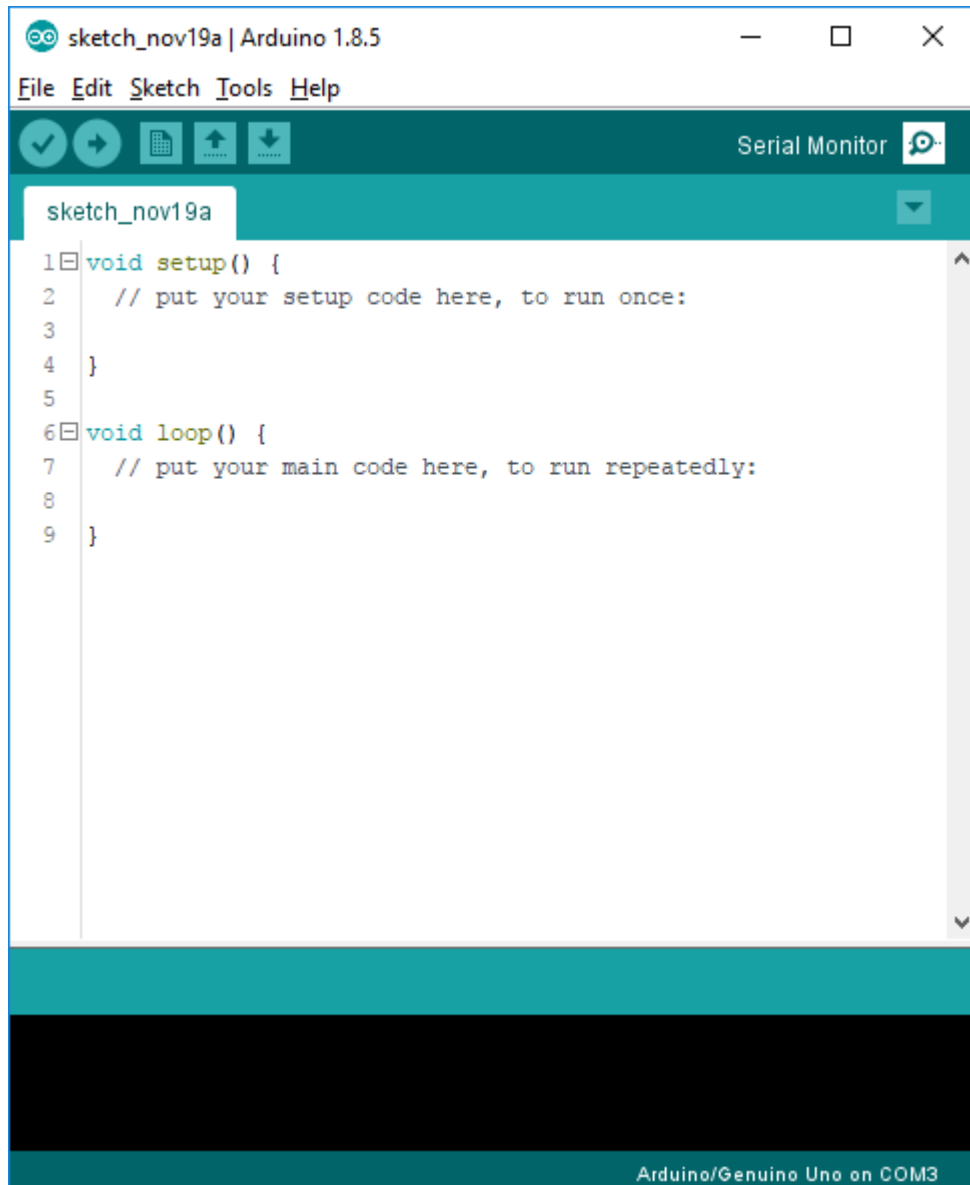
Arduino Programming may be divided into three main blocks:

1. Structure
2. Values (Variables and Constant)
3. Functions

1. Structure

The structure of Arduino program is pretty simple. Arduino programs have a minimum of 2 blocks:

- **Preparation → void setup()**
The setup function is the first to execute when the program is executed, and this function is called only once. The setup function is used to initialize the pin modes and start serial communication. This function has to be included even if there are no statements to execute.
- **Execution → void loop()**
After the “void setup ()” function is executed, the execution block runs next. The execution block hosts statements like reading inputs, triggering outputs, checking conditions etc.
As the name suggests, the “void loop ()” function executes the set of statements (enclosed in curly braces) repeatedly.



The image shows the Arduino IDE interface. The title bar reads "sketch_nov19a | Arduino 1.8.5". The menu bar includes "File", "Edit", "Sketch", "Tools", and "Help". Below the menu bar is a toolbar with icons for checking, running, saving, and uploading. The "Serial Monitor" button is visible on the right. The main editor area shows a sketch named "sketch_nov19a" with the following code:

```
1 void setup() {  
2   // put your setup code here, to run once:  
3  
4 }  
5  
6 void loop() {  
7   // put your main code here, to run repeatedly:  
8  
9 }
```

The status bar at the bottom right indicates "Arduino/Genuino Uno on COM3".

2. Values:

Variables:

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**.

Constants:

Constants are predefined expressions in the Arduino language. They are used to make the programs easier to read. We classify constants in groups.

Defining Digital Pins modes:

Digital pins can be used as **INPUT**, or **OUTPUT**. Changing a pin with `pinMode()` changes the electrical behavior of the pin.

Pins Configured as INPUT

Arduino (ATmega) pins configured as **INPUT** with `pinMode()` are said to be in a *high-impedance* state. Pins configured as **INPUT** make extremely small demands on the circuit that they are sampling, equivalent to a series resistor of 100 Megohms in front of the pin. This makes them useful for reading a sensor.

Pins Configured as OUTPUT

Pins configured as **OUTPUT** with `pinMode()` are said to be in a *low-impedance* state. This means that they can provide a substantial amount of current to other circuits. ATmega pins can source (provide current) or sink (absorb current) up to 40 mA (milliamps) of current to other devices/circuits. This makes them useful for powering

LEDs because LEDs typically use less than 40 mA. Loads greater than 40 mA (e.g. motors) will require a transistor or other interface circuitry.

Defining built-ins: LED_BUILTIN

Most Arduino boards have a pin connected to an on-board LED in series with a resistor. The constant `LED_BUILTIN` is the number of the pin to which the on-board LED is connected. Most boards have this LED connected to digital pin 13.

Defining Logical Levels: true and false (Boolean Constants)

There are two constants used to represent truth and falsity in the Arduino language: `true`, and `false`.

`false`

`false` is the easier of the two to define. `false` is defined as 0 (zero).

`true`

`true` is often said to be defined as 1, which is correct, but `true` has a wider definition. Any integer which is non-zero is `true`, in a Boolean sense. So -1, 2 and -200 are all defined as `true`, too, in a Boolean sense.

Note that the `true` and `false` constants are typed in lowercase unlike `HIGH`, `LOW`, `INPUT`, and `OUTPUT`.

3. 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.

SOME IMPORTANT FUNCTIONS:

delay() :

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

Syntax: delay(ms)

Digital I/O

pinMode() :

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

Syntax: pinMode(pin, mode)

digitalWrite() :

Set the state of the pin whether to be HIGH/on/1 or LOW/off/0.

Syntax: digitalWrite(pin, value)

digitalRead() :

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

Syntax: digitalRead(pin)

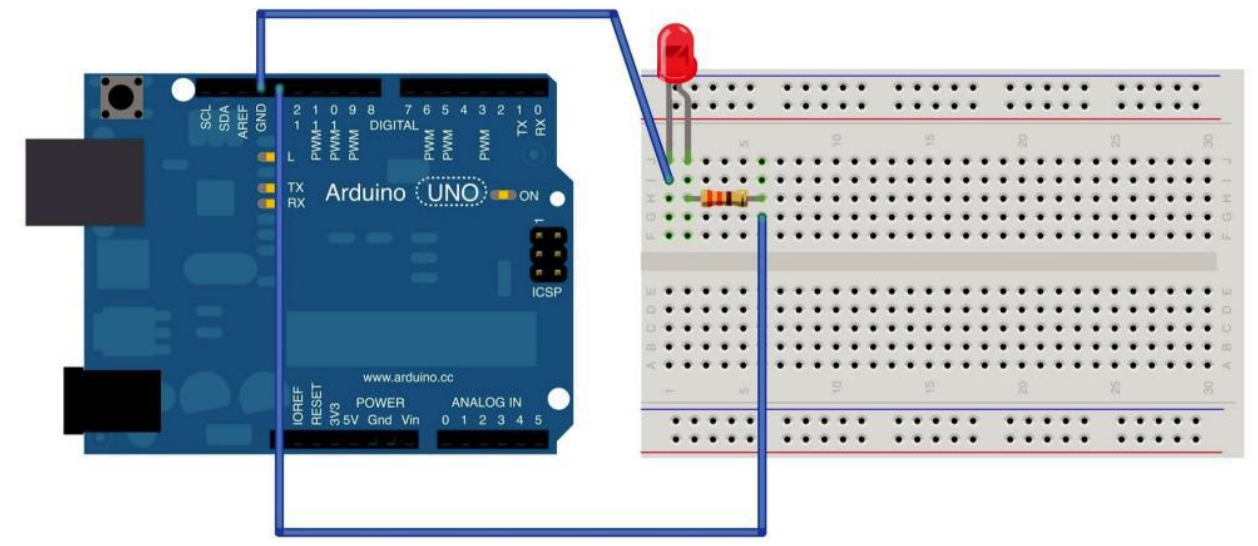
1. LED BLINK PROJECT (Single LED)

Components:

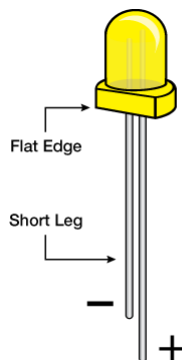
- Arduino Uno
- USB cable
- Breadboard
- LED
- 220 Ohm Resistor (Color Code: Red Red Brown Gold)
- Jumper wires

Operating Voltage: 5 volts

Circuit Diagram:



Note – To find out the polarity of an LED, look at it closely. The shorter of the two legs, towards the flat edge of the bulb indicates the negative terminal.



Code:

```
int led=13; //global variable

void setup() {
    pinMode(led, OUTPUT); //initialize digital pin 13 as an output.
}

void loop() {
    digitalWrite(led, HIGH); // turn the LED on (HIGH is the voltage level)
    delay(1000); //wait for a second
    digitalWrite(led, LOW); //turn the LED off by making the voltage LOW
    delay(1000); //wait for a second
}
```

SERIAL COMMUNICATION:

Used for communication between the Arduino board and a computer or other devices. All Arduino boards have at least one serial port (also known as a UART or USART), and some have several.



```
Simple Serial Print using Arduino
1 void setup() {
2   Serial.begin(9600);
3 }
4
5 void loop() {
6   Serial.println("Hello from Umesh");
7   delay(1000);
8 }
```

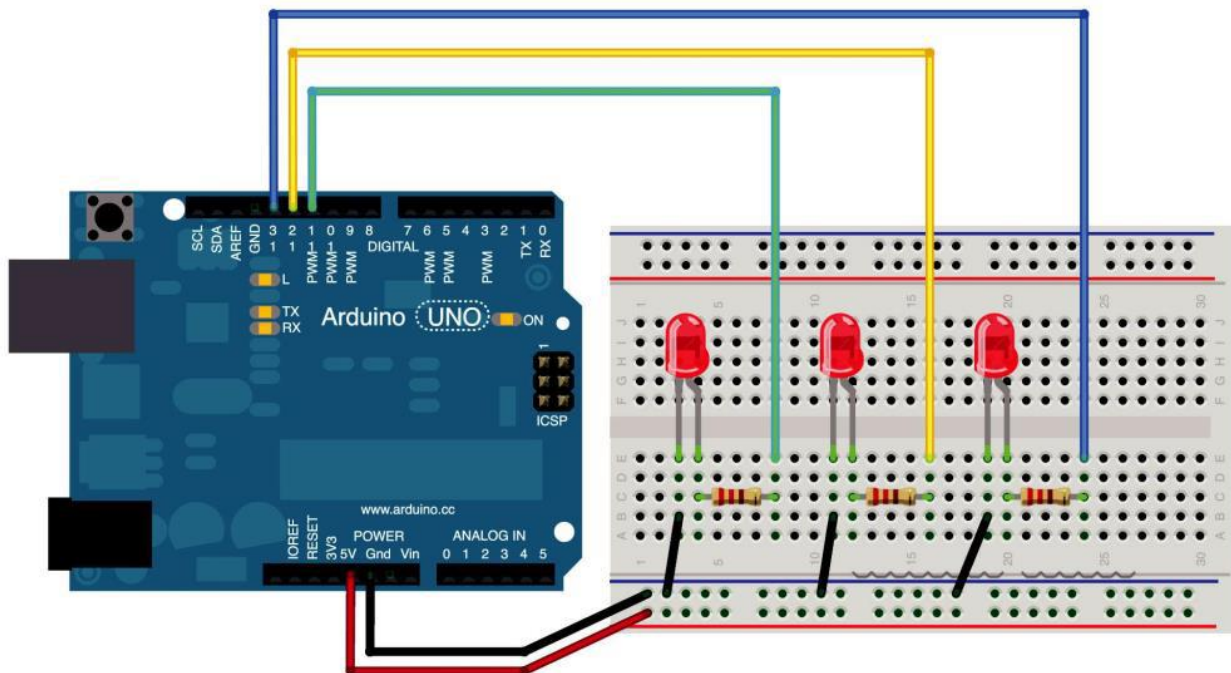

2. LED BLINK PROJECT (3 LEDs)

Components:

- Arduino Uno
- USB cable
- Breadboard
- LED x3
- 220 Ohm Resistor x3 (Color Code: Red Red Brown Gold)
- Jumper wires

Operating Voltage: 5 volts

Circuit Diagram:



Code:

```
int led1=11; //global variable

int led2=12;

int led3=13;

void setup() {
    pinMode(led1, OUTPUT); //initialize digital pin 13 as an output.
    pinMode(led2, OUTPUT);
    pinMode(led3, OUTPUT);
}

void loop() {
    digitalWrite(led1, HIGH); // turn the LED on (HIGH is the voltage level
    digitalWrite(led2, HIGH);
    digitalWrite(led3, HIGH);
    delay(1000); //wait for a second
    digitalWrite(led1, LOW); //turn the LED off by making the voltage LOW
    digitalWrite(led2, LOW);
    digitalWrite(led3, LOW);
    delay(1000); //wait for a second
}
```

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 –

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 –

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

Boolean Operators

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

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

Compound Operator:

compound assignments

Compound assignments combine an arithmetic operation with a variable assignment. These are commonly found in for loops as described later. The most common compound assignments include:

```
x ++      // same as x = x + 1, or increments x by +1
x --      // same as x = x - 1, or decrements x by -1
x += y     // same as x = x + y, or increments x by +y
x -= y     // same as x = x - y, or decrements x by -y
x *= y     // same as x = x * y, or multiplies x by y
x /= y     // same as x = x / y, or divides x by y
```

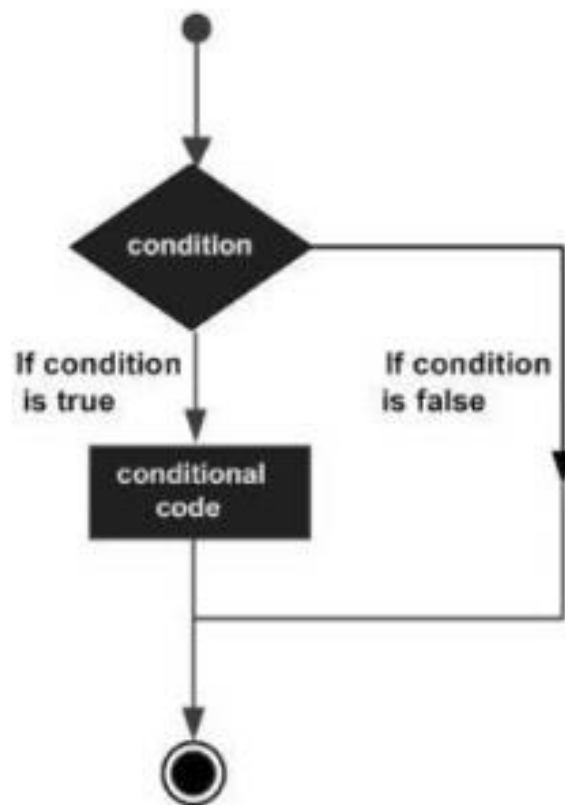
Note: For example, `x *= 3` would triple the old value of x and re-assign the resulting value to x.

Control Statements:

1. If

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)
}`



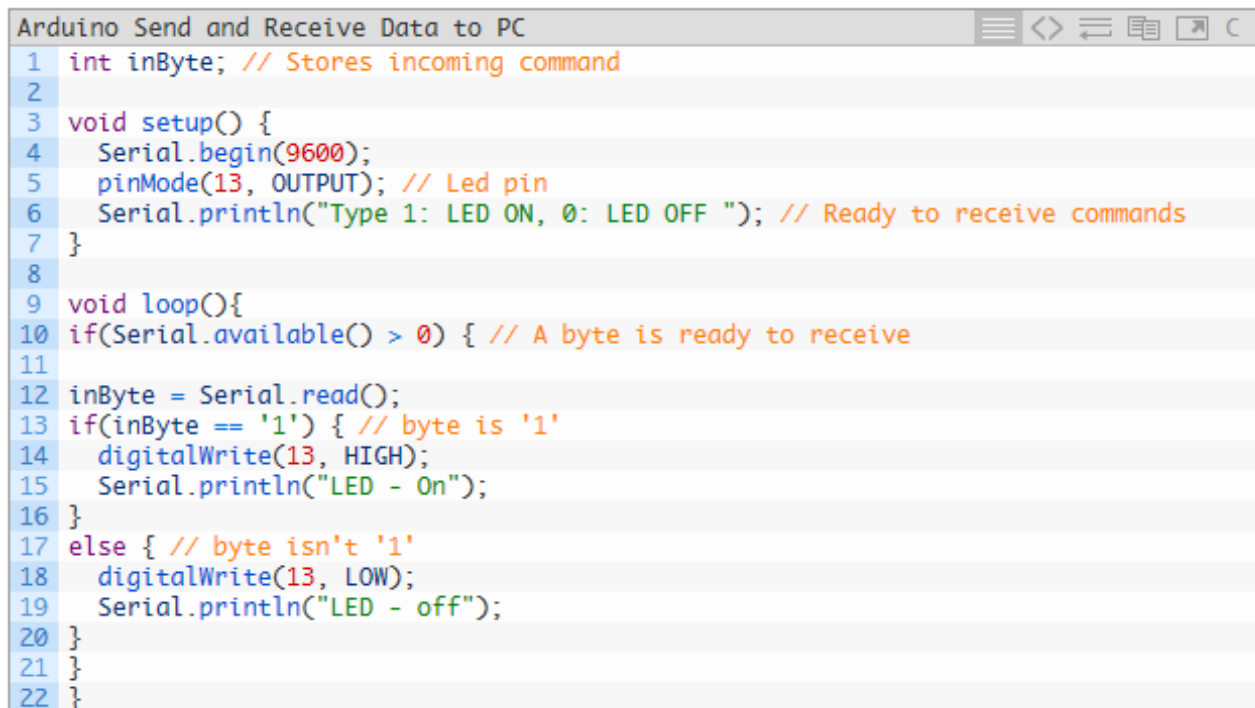
Serial.available():

Get the number of bytes (characters) available for reading from the serial port. This is data that's already arrived and stored in the serial receive buffer (which holds 64 bytes).

Serial.read():

Reads incoming serial data.

3. LED Control with Serial Monitor



```
1 int inByte; // Stores incoming command
2
3 void setup() {
4   Serial.begin(9600);
5   pinMode(13, OUTPUT); // Led pin
6   Serial.println("Type 1: LED ON, 0: LED OFF "); // Ready to receive commands
7 }
8
9 void loop(){
10  if(Serial.available() > 0) { // A byte is ready to receive
11
12    inByte = Serial.read();
13    if(inByte == '1') { // byte is '1'
14      digitalWrite(13, HIGH);
15      Serial.println("LED - On");
16    }
17    else { // byte isn't '1'
18      digitalWrite(13, LOW);
19      Serial.println("LED - off");
20    }
21  }
22 }
```

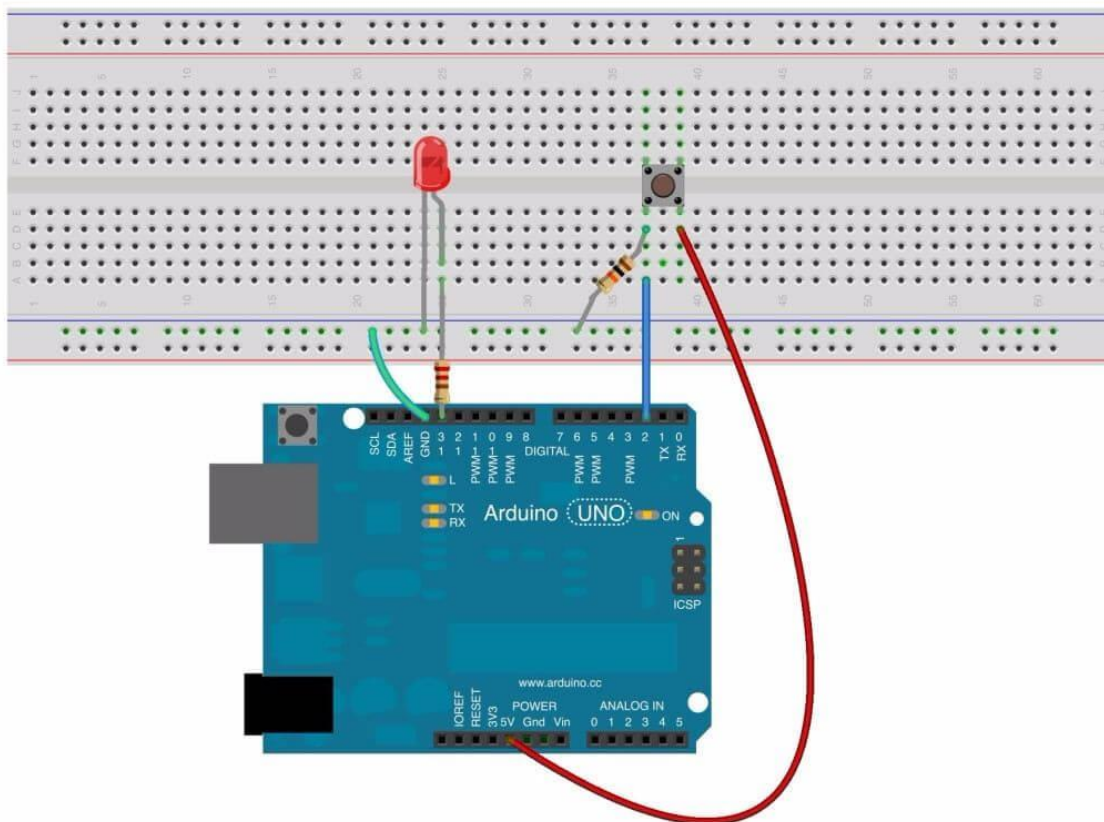

4. LED WITH PUSH BUTTON

Components:

- Arduino Uno
- USB cable
- Breadboard
- LED
- Push Button
- 220 Ohm Resistor x3 (**Color Code:** Red Red Brown Gold)
- 10K Resistor (**Color Code:** Brown Black Orange Gold)
- Jumper wires

Operating Voltage: 5 volts

Circuit Diagram:



Code:

```
// constants won't change. They're used here to
// set pin numbers:

const int buttonPin = 2;    // the number of the pushbutton pin
const int ledPin = 13;      // the number of the LED pin

// variables will change:
int buttonState = 0;        // variable for reading the pushbutton status

void setup() {
  // initialize the LED pin as an output:
  pinMode(ledPin, OUTPUT);
  // initialize the pushbutton pin as an input:
  pinMode(buttonPin, INPUT);
}

void loop() {
  // read the state of the pushbutton value:

  buttonState = digitalRead(buttonPin);

  // check if the pushbutton is pressed.
  // if it is, the buttonState is HIGH:
  if (buttonState == HIGH) {
    // turn LED on:
    digitalWrite(ledPin, HIGH);
  } else {
    // turn LED off:
    digitalWrite(ledPin, LOW);
  }
}
```

Analog I/O

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. (2^{10})

Syntax: analogRead(pin)

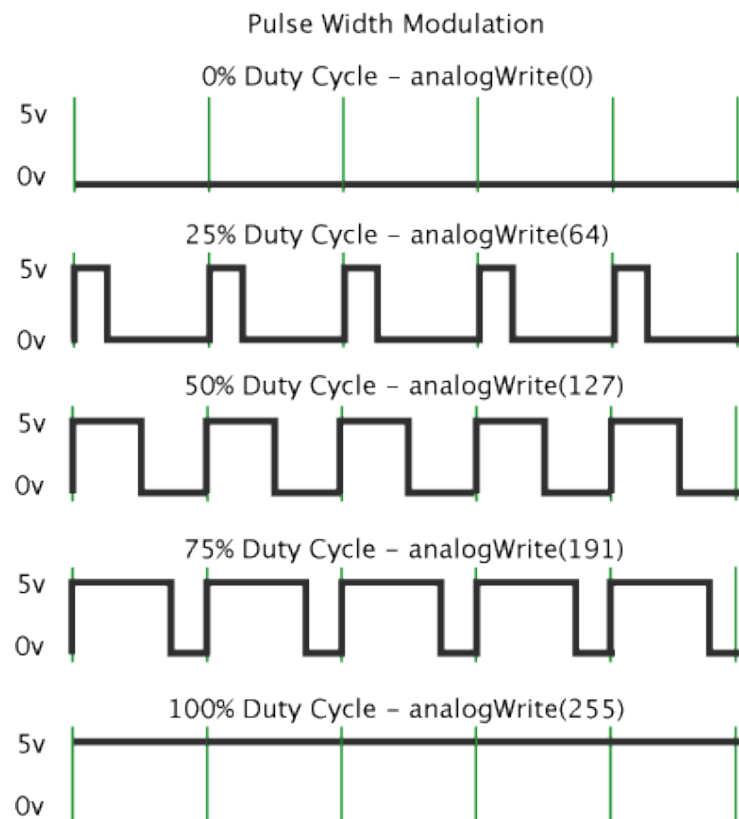
analogWrite() :

Writes an analog value (PWM wave) to a pin. Can be used to light a LED at varying brightness or drive a motor at various speeds. After a call to analogWrite(), the pin will generate a steady square wave of the specified duty cycle until the next call to analogWrite() (or a call to digitalWrite() or digitalWrite()) on the same pin. The frequency of the PWM signal on most pins is approximately 490 Hz. On the Uno and similar boards, pins 5 and 6 have a frequency of approximately 980 Hz.

Syntax: analogWrite(pin, value)

What is PWM?

Pulse Width Modulation, or PWM, is a technique for getting analog results with digital means. Digital control is used to create a square wave, a signal switched between on and off. This on-off pattern can simulate voltages in between full on (5 Volts) and off (0 Volts) by changing the portion of the time the signal spends ON versus the time that the signal spends OFF. The duration of "on time" is called the pulse width. To get varying analog values, we change, or modulate, that pulse width.



Arduino has 8 bit PWM module (2^8). A call to `analogWrite()` is on a scale of 0 - 255, such that `analogWrite(255)` requests a 100% duty cycle (always on), and `analogWrite(127)` is a 50% duty cycle (on half the time) for example. If PWM = 0, Voltage = 0v If PWM = 255, Voltage = 5v If PWM = 128, Voltage = 2.5v

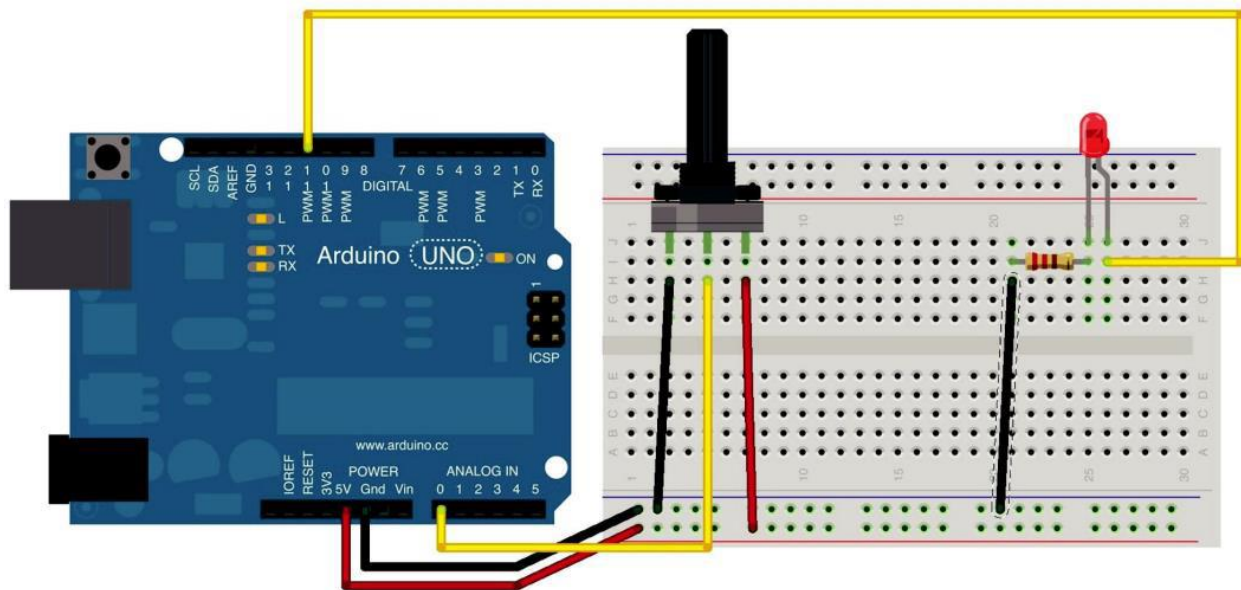
5. LED FADING WITH POT

Components:

- Arduino Uno
- USB cable
- Breadboard
- LED x5
- 220 Ohm Resistor x5 (Color Code: Red Red Brown Gold)
- Jumper wires

Operating Voltage: 5 volts

Circuit Diagram:



Code:

```
int analogPin = A0; // potentiometer connected to analog A0
int ledPin = 11; // LED connected to digital pin 11
int val = 0; // variable to store the read value

void setup() {
  // put your setup code here, to run once:
  //setting all the leds to output
  pinMode(ledPin, OUTPUT);
}

void loop() {
  val = analogRead(analogPin); // read the input pin
  analogWrite(ledPin, val / 4)
  delay(50);
}
```

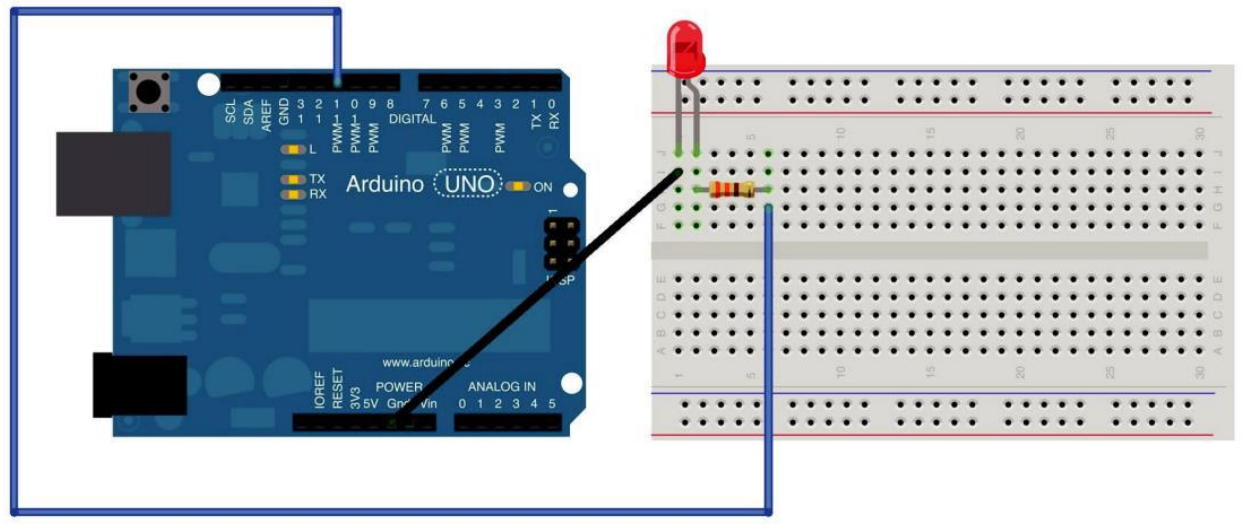
6. LED FADING WITHOUT POT

Components:

- Arduino Uno
- USB cable
- Breadboard
- LED
- 220 Ohm Resistor (Color Code: Red Red Brown Gold)
- Jumper wires

Operating Voltage: 5 volts

CIRCUIT DIAGRAM:



Code:

```
int led = 11;           // the PWM pin the LED is attached to
int brightness = 0;     // how bright the LED is
int fadeAmount = 5;     // how many points to fade the LED by

// the setup routine runs once when you press reset:
void setup() {
  // declare pin 9 to be an output:
  pinMode(led, OUTPUT);
}

// the loop routine runs over and over again forever:
void loop() {
  // set the brightness of pin 9:
  analogWrite(led, brightness);

  // change the brightness for next time through the loop:
  brightness = brightness + fadeAmount;

  // reverse the direction of the fading at the ends of the fade:
  if (brightness <= 0 || brightness >= 255) {
    fadeAmount = -fadeAmount;
  }
  // wait for 30 milliseconds to see the dimming effect
  delay(30);
}
```

Control Statements:

7. 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);
 }

Explanation:

The **initialization** happens first and exactly once. Each time through the loop, the **condition** is tested; if it's true, the statement block, and the **increment** is executed, then the **condition** is tested again. When the **condition** becomes false, the loop ends.

Code (Same fading, with for loop):

```
// Fade an LED using a PWM pin and for loop
int PWMpin = 11; // LED in series with 220 ohm resistor on pin 11

void setup() {
  pinMode(PWMpin, OUTPUT);
}

void loop() {
  int x = 1;

  for (int i = 0; i < 255; i = i + x) {

    analogWrite(PWMpin, i);

    if (i == 255) {
      x = -1; // switch direction at peak
    }
    delay(10);
  }
}
```

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