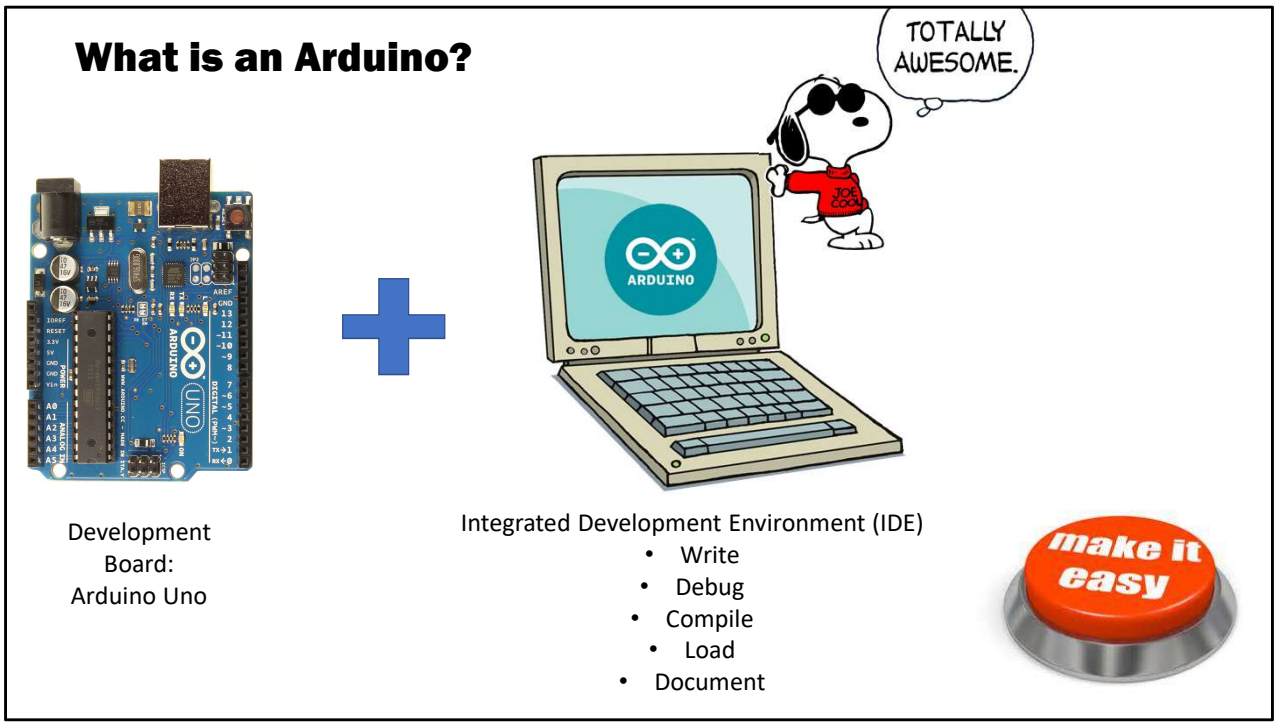

SnoCo
MAKERSPACE
Arduino 101



Arduino 101

Arduino is one of the most known development boards out there and, because of that, a common mistake is to assume the name Arduino refers only to the board itself.

Being a trademark, the name Arduino can only be used by the boards made by Arduino.org. However, given Arduino is an open-source Hardware, any company can take the schematics and create their own version of Arduino; the Arduino compatible.

Arduino Compatible boards: they look and behave like an Arduino board, and also allow you to use the Arduino IDE (integrated development environment).

This way, besides several types of boards, the name Arduino also refers to the IDE which is a crucial partner of the board because it will allow you to:

1. Write code: Arduino IDE uses a mix of C and C++.
2. Debug the code: it is very hard to write an error-free code, and the debugger will help you find all of the syntax errors you code might have.
3. Load: allow your computer to load the code to the Arduino board connected to it (you have to select the right board and the right communication port).

4. Compile: before loading you code to the Arduino board, the IDE needs first to transform/compile it into something the Arduino can understand (hexadecimal code).
5. Document: you can also use the IDE to write comments and make your code easier to understand (which is useful even to yourself in the future).

Is it easy to use?

It can be easy. However you need to bear in mind an Arduino project will require both electronics concepts (Power, tension, current, and all the different components part of your circuit) and also programming concepts (code that runs in the Arduino board interacting with the components).

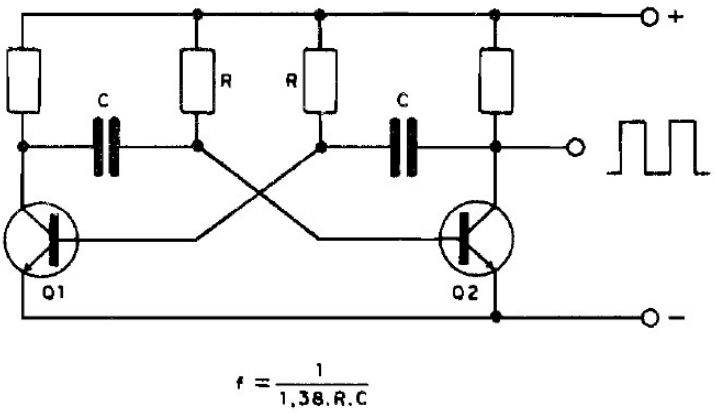
If you're building, for instance, a robot then you would have to add mechanic concepts on top of the electronics and programming.

For this training, given there are no pre-requisites, we will abstract several of the basic concepts so we can achieve its main goal:

To give you an overview of what an Arduino board is and some examples of what it can do.

The old way (Why do I need an Arduino?)

- Blink an LED
- Astable multivibrator
- Frequency is given by the resistors and capacitors



$$f = \frac{1}{1{,}38.R.C}$$

Why do I need an Arduino board?

Before we can answer that question, we need to do a quick look back in time and back to basics electronics.

Basic Circuit

Imagine we need to blink an LED (for whatever reason). The simplest circuit to do so would be an astable multivibrator as shown in the picture, where the frequency is given by the resistors and capacitors (according to the formula).

You would have to physically build this circuit in order to blink the LED.

Now imagine that you need a faster blinking LED. How can you achieve this? You would have to go back to the formula to define the new values for resistors and capacitors then replacing the ones in the original circuit.

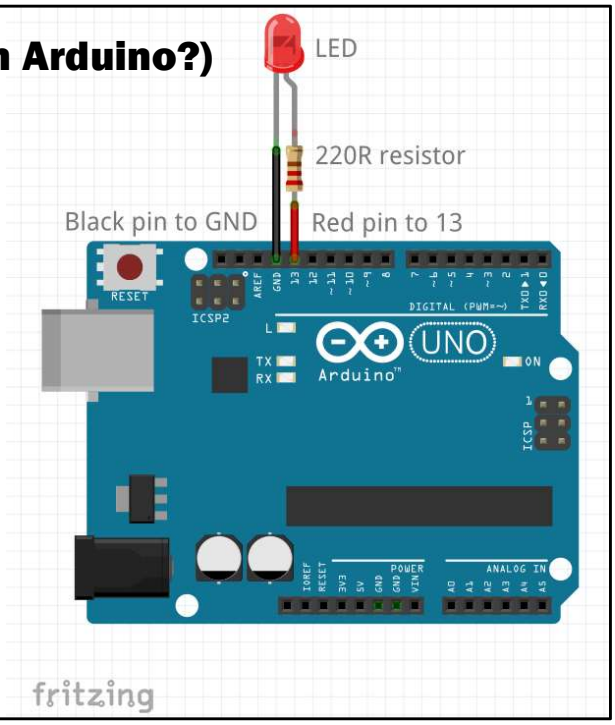
For each change in the specification you will have to redesign your circuit, choose the appropriate components and assemble the circuit.

Not exactly a simple task. As you might imagine, for more complexes scenarios the picture is not prettier.

Back to basics (Why do I need an Arduino?)

Additional scenarios

- What if we want to blink the LED faster?



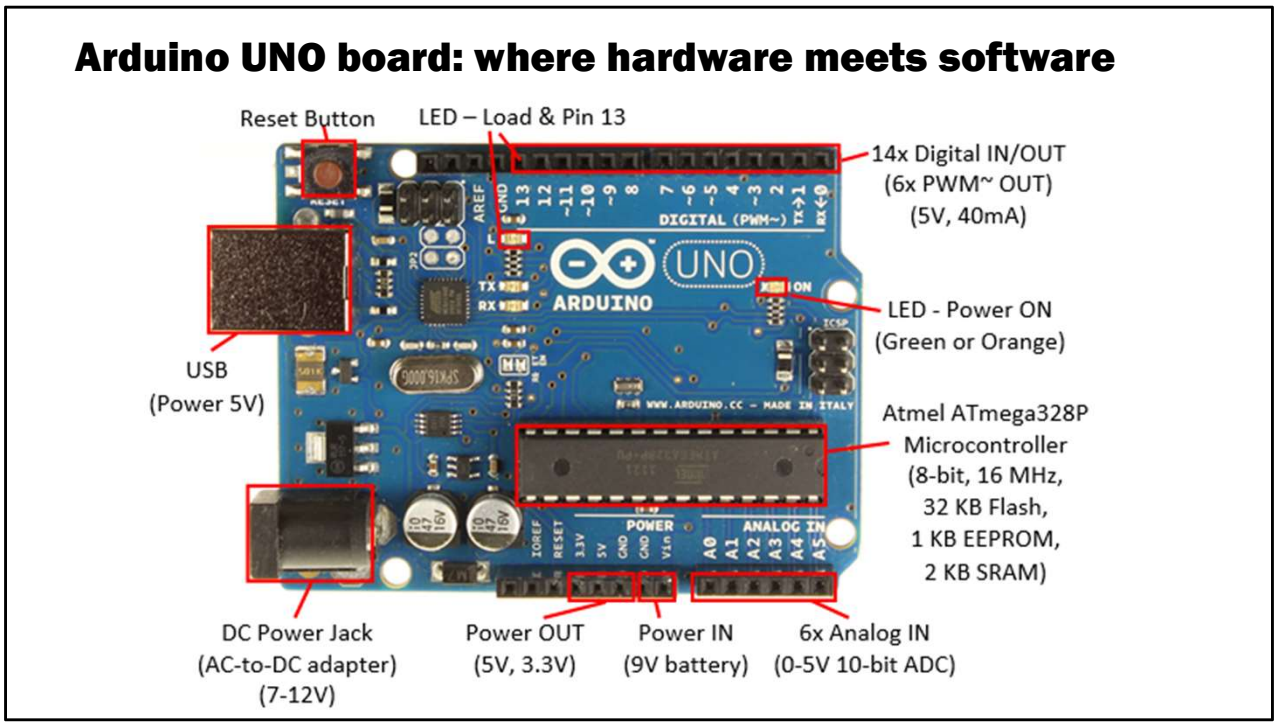
Blinking LEDs with the Arduino

Now looking at the very same example but this time using an Arduino board, let's try to recreate the experiment.

First thing you'll notice is that in the blinking scenario, rather than 8 discrete components, we only need 2: the LED and a resistor (not a counting contest, because obviously the Arduino board has dozens of components) which makes the assembly way simpler.

Given the Arduino board is a microcontroller, what it does is to control things. Things like the LED. In order to do so, we have to write a code that tells the Arduino microcontroller to turn the port where we have the LED on and off.

Likewise, for any other circuit and combination of the Arduino board and components, you'll have to define what the behaviors will be by writing the code.



Arduino UNO

Introducing the most common (and famous) of the Arduino boards: Arduino Uno.

Over the years this board became the "*de facto*" board for learners all over the planet. Because of the fact it is also open source, several other companies created their own versions of the Uno board (the use of the name and the logo are forbidden though).

When buying boards you should consider getting at least some original boards to support the project, even though we know the Chinese manufacturers would sell you the same functional board for a fraction of the official price.

In the next page you will find the Arduino schematics, available for download in the official Arduino site: https://store.arduino.cc/usa/arduino-uno-rev3

If you have a background in electronics this will help you understand how an Arduino Uno board works, which components were used, how to connect them and tweak it for your own projects. If you don't know electronics yet, skip it for now: it might be useful for you in the future.

Arduino Pinout

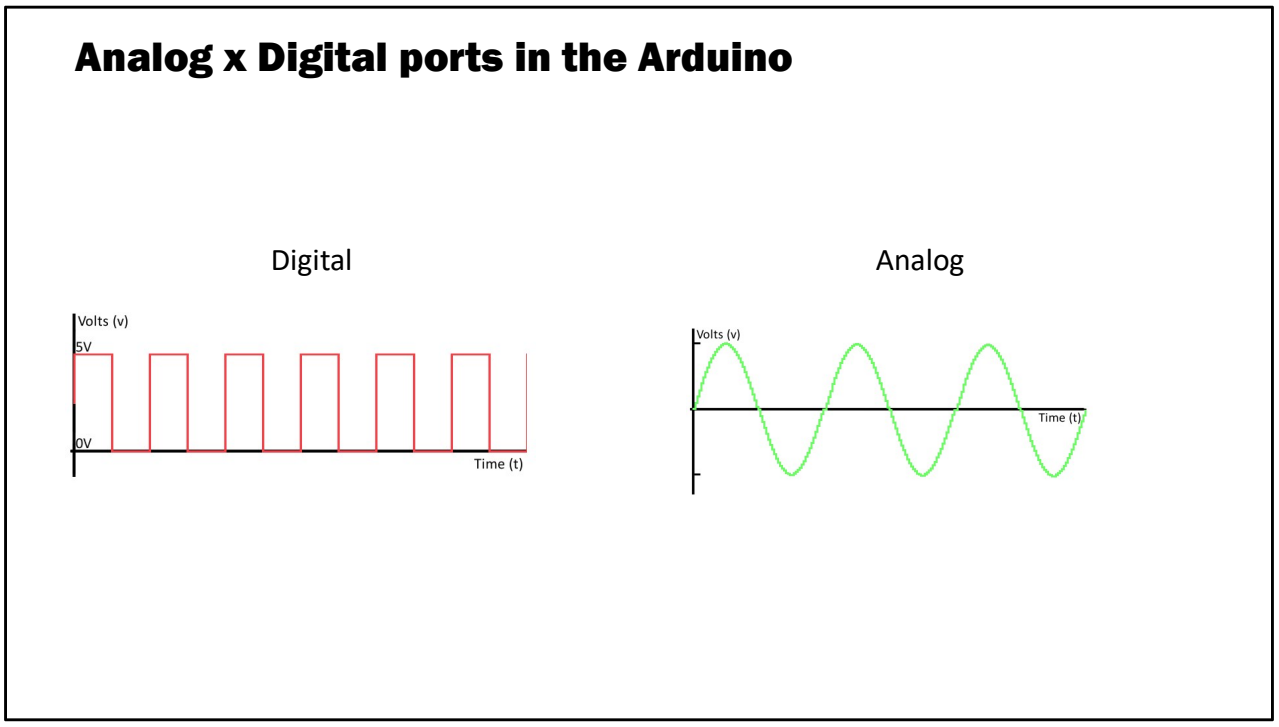
This layout shows all Arduino Uno pins and explain what's their function in the board. It breaks down the pins by different categories:

- Power pins / GND: pins you can use to power up small loads. Please note there is a limit per pin and total for the board that should be observed.
- Serial pins: these are the pins you can use for serial communication.
- Analog pins: Arduino Uno brings you 6 analog input ports.
- Control pin: the only control pin in the Uno board is the reset (same function as the Reset button)
- Physical pins: identifies the connection between the external pin and the Microcontroller pin.
- Port pins: identifies the port pins names the Microcontroller uses.
- Pin function: shows the function for some specialty pins
- Interrupt pins: the Uno brings you 2 physical interrupts for use.
- PWM pins: shows which pins are capable of operating with PWM (Pulse Width modulation)

As you can see several pins in the Arduino Uno have multiple functions, even though most of them cannot operate in multiple functions at the same time.

The Arduino board can be powered up using the USB cable connected to your computer, however for bigger consumption scenarios the DC jack should be used instead.

There are also two headers for ICSP (In circuit serial programming) in the board. One for the AT Mega 326 (the big chip sitting in a socket) and another one for the tiny AT Mega 32u, used in this board as a USB to serial conversion circuit.



Digital x Analog ports

The Arduino Uno board has 14 digital ports (ports 0 to 13) and 6 Analog input ports (A0 to A5).

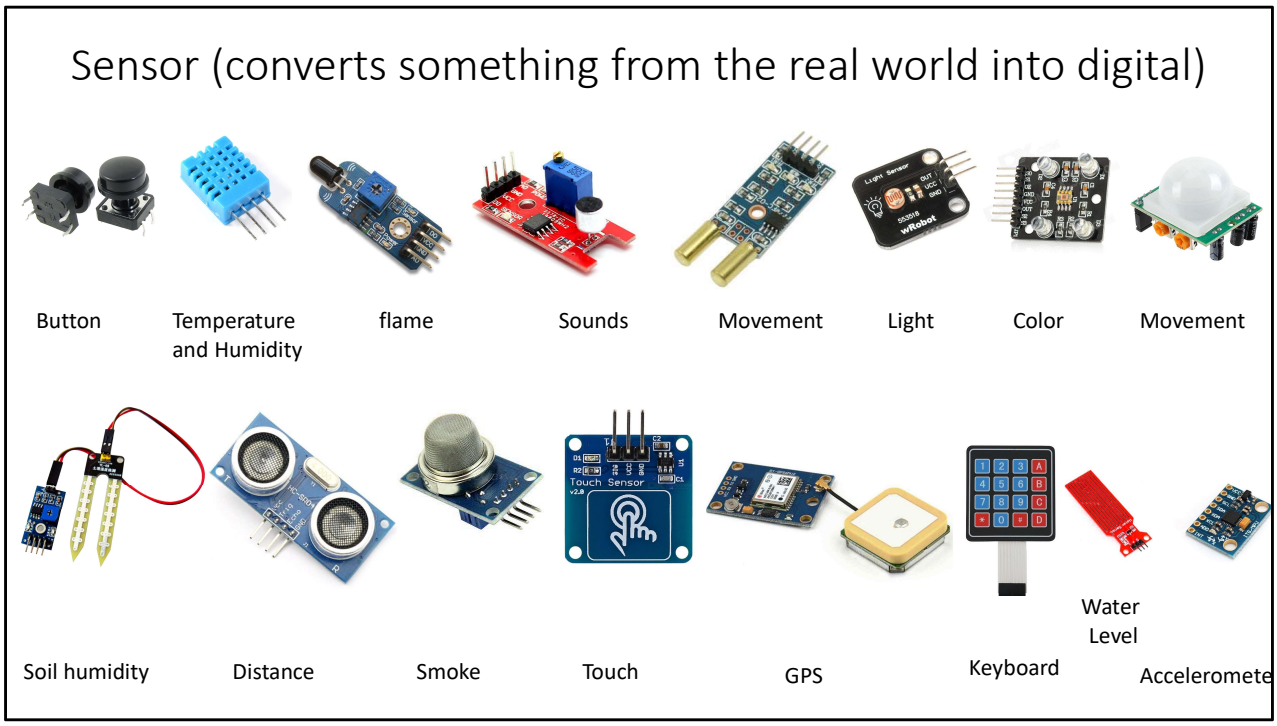
They are fundamentally different in terms of the way they translate real world signals to the Arduino.

> **Digital ports** only recognize two states: 1 (also known as High) or 0 (also know as Low). You may consider state 1 as On because when a port is on a high state, it will provide 5 Volts and when a port is in a low state it will provide 0 volts.
>
> We use digital ports for any sensor that is either on or off like a button, an infrared sensor and so on. We'll also use digital pins to sensors sending information via I2C (Inter-integrated circuit).
>
> **Analog ports** are internally connected to an 10Bit ADC (analog to digital converter) which mean these ports can read values that varies from 0 to 1023 (thus a 1024 level resolution).

We use analog ports for any sensor that varies like a light sensor, a pressure sensor (used in scales), noise sensor and so on.

Note that analog ports A0 to A5 will also work as digital ones if you need additional digital ports to your project.

Reference: https://learn.sparkfun.com/tutorials/analog-vs-digital



Arduino I/O

A typical Arduino board brings you 6 analog inputs and 14 digital I/O (input/output) pins. They can be used to connect several sensors and actuators.

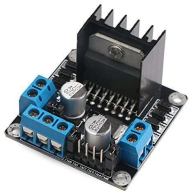
Sensors provide information to the board (e.g. the temperature reading of some place).

Input sensors

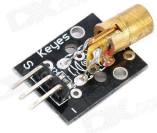
- Ultrasonic Sensors: pair of transmitter /receiver that allow you to calculate distances (just like the ultrasonic system of bats)
- Light sensor: detects the amount of light hitting the sensor
- Temperature: provide readings from the environment temperature. They range from home use to industrial applications.
- Humidity: provide readings from the environment humidity. Just like temperature sensors, you'll find expensive, very precise models but also cheap ones for home use.
- Sound: detects the presence of sounds. Can be adjusted to which intensity will trigger the Arduino.

- GPS: provides the location coordinates of the sensor to the Arduino.
- Button: there are several types of buttons, push buttons, on-off switches, reed switches, etc.
- Touch screens: part of modern screens, the touch screen can also be used as an input method for the Arduino allowing you to create nice GUI – graphic user interfaces.
- Keyboard: you can use any combination of single keys, to multiple keys (limited to the number of I/O ports, unless you use some matrix mechanism).

Actuator (converts something digital into real world)



Motor driver



Laser emitter

LED / RGB / Neo pixel

Display LCD

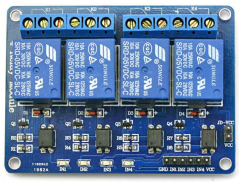
Buzzer

Water valve

Liquid pump

Audio player

Stepper motor



Relays

Arduino I/O

A typical Arduino board brings you 6 analog inputs and 14 digital I/O (input/output) pins. They can be used to connect several sensors and actuators.

Actuators will perform some action in the physical world based on information provided by the board (e.g. turning a motor on/off).

Output actuators

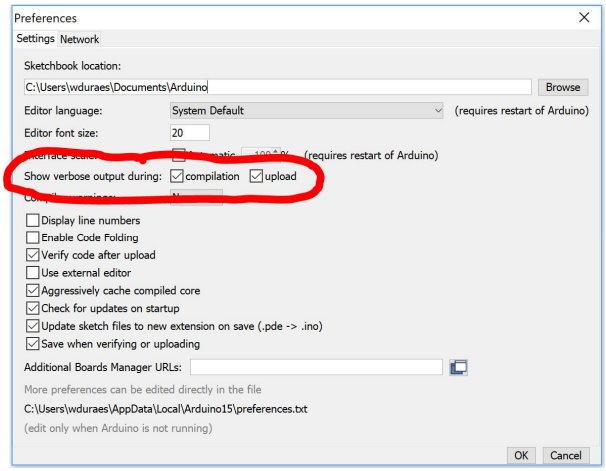
- Lights: varies from tiny LEDs, to bigger lights. If used in conjunction with a relay, the light power is – almost – unlimited.
- Neopixels: a special type of LED where red, green and blue LEDs are integrated alongside a driver chip into a tiny surface-mount package controlled through a single wire. They can be used individually, chained into longer strings.
- Motors: just like lights, there are large amount of different motors. For the smaller ones, you can drive them directly from the Arduino, bigger ones will require a driver, same for stepper motors.
- Relays: acts as switches to power big electric loads while isolating them from

the Arduino circuitry.

- Displays: varies from simple 7 segment displays (used in alarm clocks) to liquid crystal displays (2 – 4 lines and 20 characters) and even color monitors.
- Speakers: buzzers and other similar devices capable of providing sound feedback.

Arduino IDE Pre-req:

1. In the Arduino IDE, go to File/Preferences, check both boxes for compilation and upload.
2. Connect the board you have
3. Go to Tool / Port and select the COM port you found at the device manager
4. Go to Tool/Board and select "Arduino/Genuino Uno"



Arduino IDE

(Integrated development environment) is provided as a free download at: https://www.arduino.cc/en/Main/Software in versions for Mac, Linux and Windows.

Currently they support two different types of IDE:

Arduino Web editor: Allows you to use a web interface to create, build, test and all other functionalities using a browser. However, this version still requires you install a plugin in your computer to allow the serial communication.

Arduino 1.8.3 (currently the latest one): is the standard software you will install and use. This is also the one we recommend for the use in this workshop.

NOTE: This material was created using Windows and uses only Windows references.

Anatomy of a sketch:

```
//Use this area for declaring variables, adding
//libraries and references.

void setup() {
/*Place here everything you want to happen once you
power you Arduino board.
This will also happen once every time you press the
Reset button on the board.*/
}

void loop() {
//Code here will repeat forever, while the unit has
//power.
}
```

{ between braces }

//Comment

Sketch

All Arduino sketches will follow this structure:

Before setup: this is the area for variable declaration (for global usage), for adding libraries and any reference.

Setup: place here code to be executed once the board is powered up (and also every time Reset button is pressed – the onboard button present on every Arduino Uno board)

Loop: in the next example, Arduino will execute lines 5 to 8 and then go back to 5 as long as the board has power.

Comments: It is a great idea to document your code. To do so, you need to add double forward slashes before the comment and the IDE will ignore everything that goes beyond that in the line.

You can also comment blocks and to do that you'll need a forward slash followed by an asterisk /* and close the comment section by */ just like used in the slide above.

Code organization: as long as your code has all of the correct tags, parenthesis and syntax it will work even if you write your entire code in one single line.

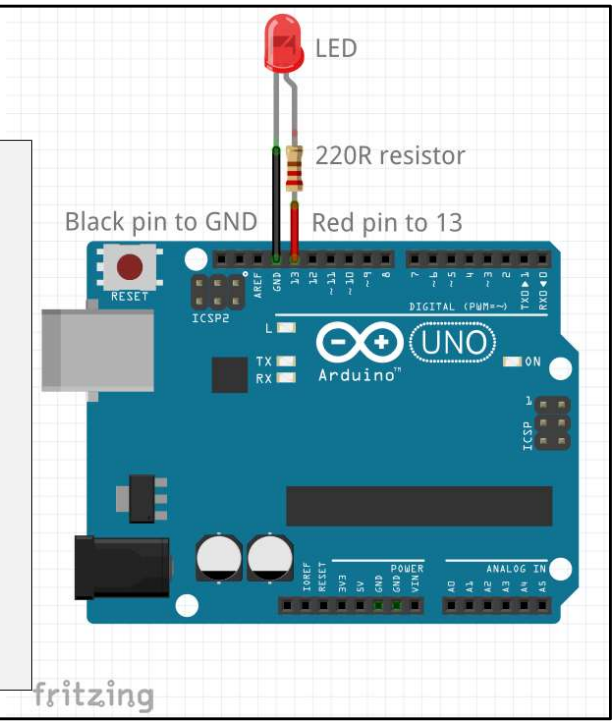
However this is a very bad coding practice because if you need to debug it (find errors) it will be way harder for you to do that. It will also be harder for other people to read and understand your code.

Breaking lines in your code and adding comments won't increase your code size so we encourage you to follow the structure we use in the following examples.

Your first sketch: Blinky

Code is case sensitive!!

```
1  void setup() {
2  pinMode(13, OUTPUT);
3  }
4  void loop() {
5  digitalWrite(13, HIGH);
6  delay(1000);
7  digitalWrite(13, LOW);
8  delay(1000);
9  }
```



Blinky

This is your first sketch / code / program. It will allow you to blink an LED.

Given the IDE is case sensitive you'll have to type the code exactly as it appears inside the grey box above.

Code explained:

Code in line 2 set pin 13 as an Output in the board.

In Line 5 we tell the Arduino to write a high logic level to pin 13 (high level means 5V), which will turn the LED on.

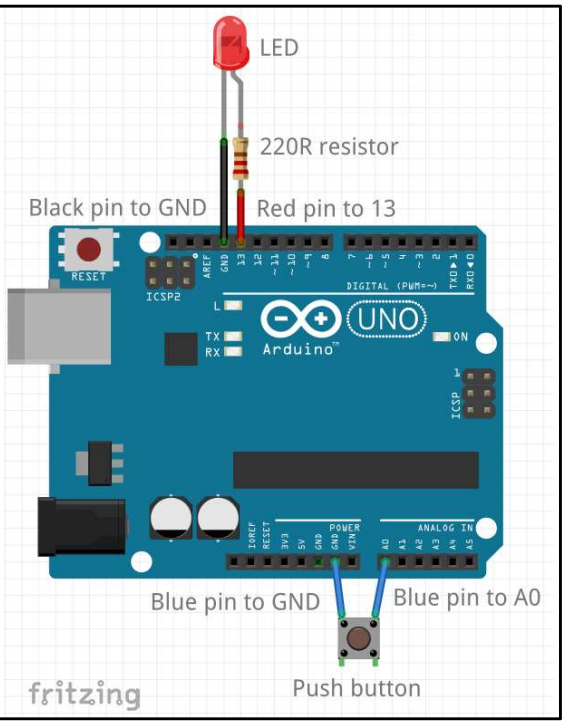
Line 6 tells the Arduino to add a 1 second (1,000 milliseconds) delay in the code execution, which will keep the LED on for that time.

In Line 7 we tell the Arduino to write a low logic level to pin 13 (low level means 0V), which will turn the LED off.

Finally, in line 8 we have another 1 second delay.

Adding a button

```
1  int button=0;
2  void setup() {
3    pinMode(A0, INPUT_PULLUP);
4    pinMode(13, OUTPUT);
5  }
6  void loop() {
7  button = digitalRead(A0);
8     if(button==LOW)
9     {digitalWrite(13, HIGH);}
10    else
11    {digitalWrite(13, LOW);}
12 }
```



Button: Arduino Inputs

In Blinky sketch we played with an output turning it on and off in a loop. Next challenge is to add a control to the LED: a button.

Unlike the LED, which takes an output pin, the button is a type of input: it will send information to the Arduino.

In order to make this code a little easier, we're going to use a variable which will hold the information if the button is pressed or not.

This is what we do at line 1: create a variable called button and assign the value 0 to it.

During setup (lines 3 and 4) we define pin A0 as an input – actually as input with an internal pullup resistor, and pin 13 as an output.

Inside the loop (the infinite repetition part of the code) line 7 we read what's the value in pin A0 (in this case 0 or 1) and then assign the button variable with this value.

Line 8 is the conditional statement which will evaluate the content of the button variable. If

it is low (zero volts or GND) then the statement in line 9 will be executed: turn the LED on. Otherwise code in line 11 will happen: turn the LED off.

IF Statement

```
if (condition to evaluate)
{what you want Arduino to do if the condition is true}
Else
{what you want Arduino to do if the condition is false}
```

The else part of the if statement is optional.

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)

What is going on? Serial monitor

```
1  Serial.begin(9600);
2
3  Serial.print("text");
4  Serial.println("text in a new
5  line");
6
7  Serial.print("Button value: ");
8  Serial.println(button);
9
10 delay(50);
```

Serial Monitor

So you wrote the code, there are no syntax errors, you load it and, somehow, it's not working. How to find out what is going on?

Your best friend in a situation like this is the Serial Monitor: it will allow your board to communicate back to the PC via the serial port.

This is great because you'll be able to see, real time, variable values, port logic state and much more.

In the example above, line 1 initiates the Serial Monitor and allow you to define at what speed the communication should happen (from 9600 all the way to 115200 bits per second).

Once you have the serial communication started, all you need to do is to declare in you code what do you want to see in the serial monitor window. Serial.print is the code you use to do that.

Serial.print("text") as shown in line 3 will send the string **text** through the serial connection.

Whatever you have inside the “ ” will be sent back to the computer.

Serial.print(button) as shown in line 8 will send the button’s value. Remember? In the previous example, button was a variable that read the value of a port. So, by doing this Serial.print(button) you’ll be able to see in your computer what is the value of that variable real time.

Notice a difference between print and println? Both will send information back to the computer, however println will send a carriage return signal after printing, creating a new line. So if you need to print things side by side, user print, if you want new lines after printing use println.

Obviously Serial monitor will only work while your computer is connected to the board.

Delay

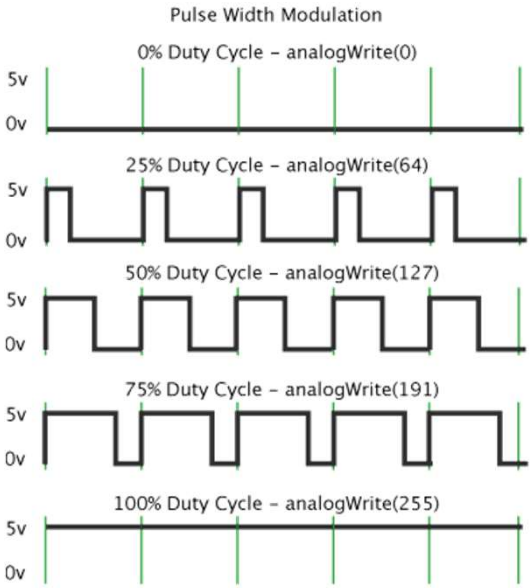
In line 10 we have a delay(50) instruction. It does what you would expect: create a delay in the code execution for the time expressed in the parenthesis. In this case we’re creating a 50 milliseconds delay (for one second use 1000).

When using the serial monitor it might be a good idea add a little delay in the code execution so the interface with the computer and the board have time enough to get a complete message before trying to show that.

Open Serial Monitor window: there are two ways to do that. You can use the keyboard shortcut “CTRL+SHIF+M” while you have the Arduino IDE open or you can select Tools/Serial Monitor.

```
1  int button=0;
2  void setup() {
3    pinMode(A0, INPUT_PULLUP);
4    pinMode(13, OUTPUT);
5    Serial.begin(9600);
6  }
7  void loop() {
8  button = digitalRead(A0);
9  Serial.print("Button value: ");
10 Serial.println(button);
11 delay(50);
12    if(button==LOW)
13    {digitalWrite(13, HIGH);}
14    else
15    {digitalWrite(13, LOW);}
16 }
```

PWM: Pulse Width Modulation



PWM

According to Wikipedia **Pulse-width modulation (PWM)** is a modulation technique used to encode a message into a pulsing signal. Although this modulation technique can be used to encode information for transmission, its main use is to allow the control of the power supplied to electrical devices, especially to inertial loads such as motors.

The average value of voltage (and current) fed to the load is controlled by turning the switch between supply and load on and off at a fast rate. The longer the switch is on compared to the off periods, the higher the total power supplied to the load.

Six of the Arduino Uno board pins are PWM enabled (The ones with the ~ symbol): 3, 5, 6, 9, 10 and 11.

The number of PWM ports varies from board to board. As we said, Arduino Uno has 6, Arduino Nano also 6 and Arduino Mega 14.

Now imagine we have a LED connected to one of these pins.

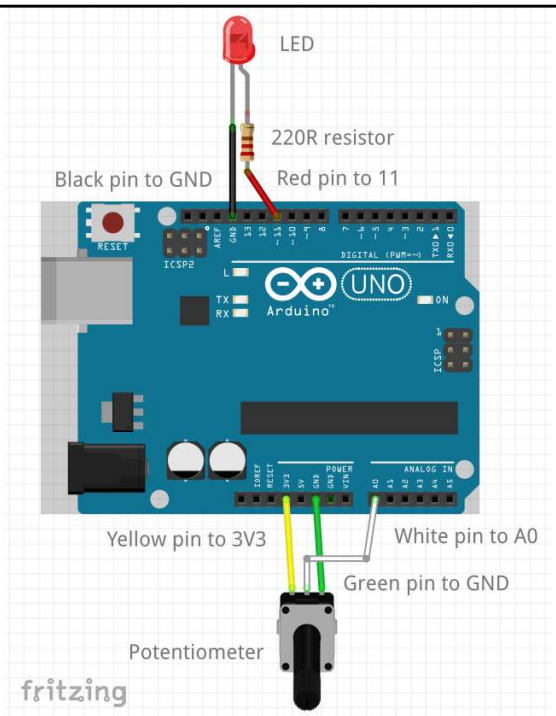
As you can see in the picture above, we can use a command such as analogWrite(0) to

determine a 0% duty cycle (power) to the pin. In this case, the LED would be off.

Given the range for any of these pins is 256 values, you can assign any value between 0 to 255. Second line in the picture shows analogWrite(64) which provides 25% of total power to the LED, going all the way to the last line where we have analogWrite(255) meaning full power.

Often we'll see project using PWM ports to control the speed of motors, but it can actually be used to limit the power delivered to any electronic/electric device (always observing the board limits).

Adding a potentiometer



Potentiometer: analog Input

In this example, we'll use a potentiometer to control an analog input and fade one LED.

Two main concepts are used here:

1. Analog Input – the potentiometer reading
2. PWM output – the LED brightness.

Adding a potentiometer

```
1  int light=0;
2  int light2=0;
3
4  void setup() {
5    pinMode(A0, INPUT);
6    pinMode(11, OUTPUT);
7  }
8  void loop() {
9  light = analogRead(A0);
10 light2 = map(light,0,1023,0,255);
11 analogWrite(11, light2);
12 }
```

Potentiometer: analog Input

Code explained:

Line 1 we create a variable to keep the light reading from the Analog pin.

In line 2 we create another variable to keep the light value (how bright we ant the LED to go).

Lines 5 and 6 inside the Setup is where we tell Arduino that Pin A0 will be an input and pin 11 an output.

The Loop:

Line 9. We do an Analog read from pin 10 → analogRead(A0) and store the value read into the light variable.

Line 10: convert the value from 10 bit down to 8bit.

Analog inputs are 10bits, thus they will vary from 0 to 1023. We need to

read this value and use it to define the output on pin 11.

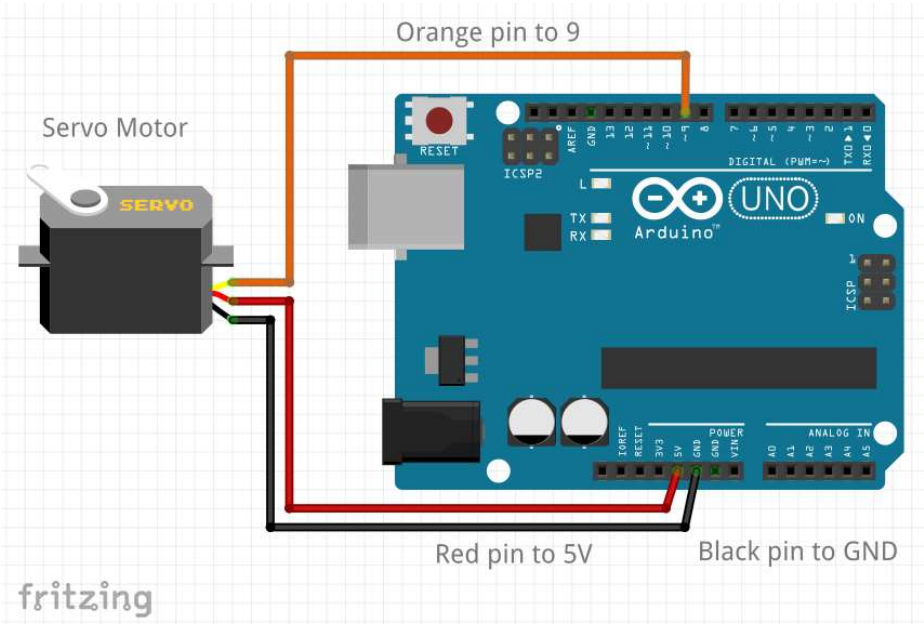
Pin 11 is a PWM one, which varies from 0 to 255 (8 bits). The way we use to adapt these different ranges is using the Map function.

So, the variable light2 is getting the result of a mapping.

Map function takes the value stored in variable light and we tell it this value varies from 0 to 1024 and assign the new range to these values which is from 0 to 255. The function will take care of the mapping.

Finally on line 11 we write a PWM value to the pin 11, according to what we have stored in the variable light2.

Servos – Motor control



Motor control: Servo library

According to the Arduino reference page the servo library allows an Arduino board to control RC (hobby) servo motors.

Library: The Arduino environment can be extended through the use of libraries, just like most programming platforms. Libraries provide extra functionality for use in sketches, e.g. working with hardware or manipulating data. To use a library in a sketch, select it from Sketch > Import Library.

Servos have integrated gears and a shaft that can be precisely controlled. Standard servos allow the shaft to be positioned at various angles, usually between 0 and 180 degrees. Continuous rotation servos allow the rotation of the shaft to be set to various speeds.

The Servo library supports up to 12 motors in the Arduino UNO board. The use of the library disables analogWrite() (PWM) functionality on pins 9 and 10, whether or not there is a Servo on those pins.

Servo example:

In this example, we'll be using a servo motor and we'll control the position we want the motor to turn.

Servos – Motor control

```
1  #include <Servo.h>
2  Servo myservo;
3
4  void setup() {
5    myservo.attach(9);
6  }
7  void loop() {
8    myservo.write(0);
9    delay(500);
10   myservo.write(180);
11   delay(1000);
12 }
```

Servo: motor control

Code explained:

Line 1 we include the servo library.

Line 2 is where we create an object named myservo based in the Servo library. Once we have this object created it will inherit all characteristics defined in the servo library.

Inside the setup we connect (attach) the motor to the pin 9, so Arduino will know there is a servo motor connected to that pin.

The loop is quite simple:

Line 8 write the value 0 to the myservo object we created. This will make the motor to move to 0 degree position.

Giving the movement is mechanical, it might takes some time until the servo reaches the

desired position, so we added a 500 milliseconds delay in line 9 to give the servo time to move.

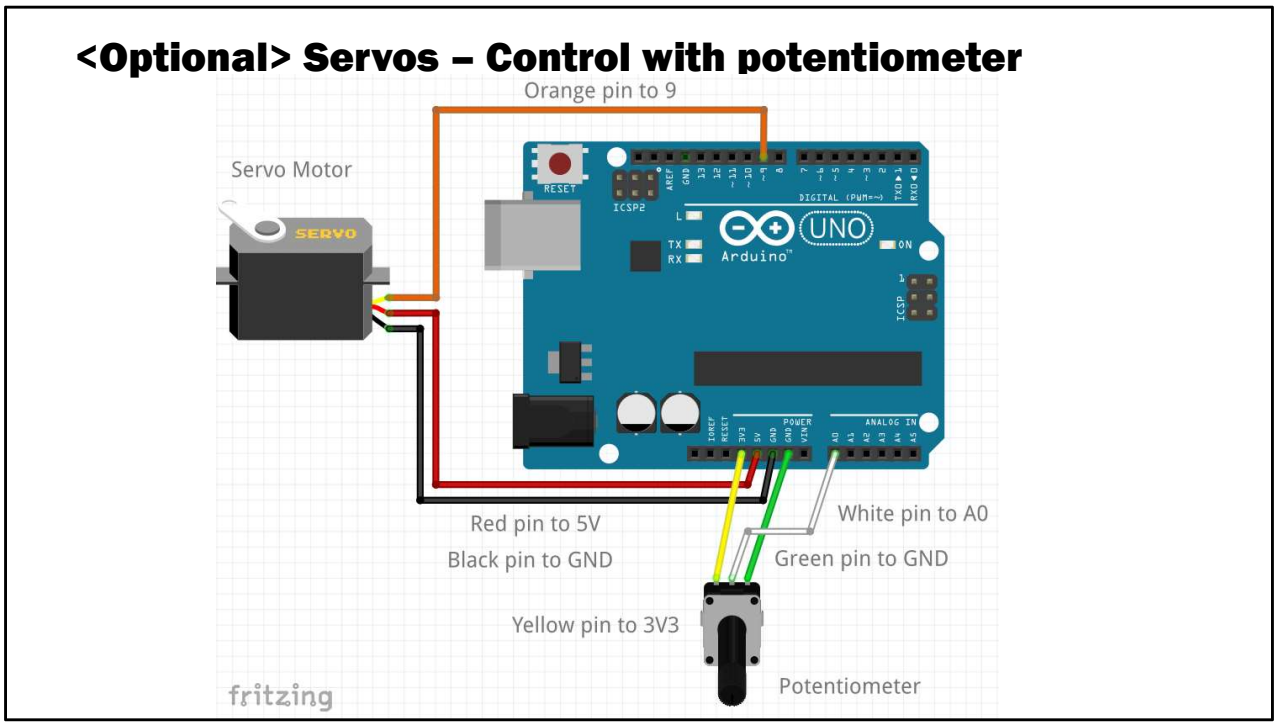
Line 10 write the value 180 to the myservo object we created. This will make the motor to move to 180 degrees position.

Add more 1000 milliseconds (1 second) delay in line 11.

Variation

Experiment changing the delay, or adding extra steps inside the loop. Something like this (the following code is based on the Arduino Sweep example):

```
void loop() {
  for (pos = 0; pos <= 180; pos += 1) { // goes from 0 degrees to 180 degrees
    // in steps of 1 degree
    myservo.write(pos);              // tell servo to go to position in variable 'pos'
    delay(15);                       // waits 15ms for the servo to reach the position
  }
  for (pos = 180; pos >= 0; pos -= 1) { // goes from 180 degrees to 0 degrees
    myservo.write(pos);              // tell servo to go to position in variable 'pos'
    delay(15);                       // waits 15ms for the servo to reach the position
  }
}
```



Servo: control with a potentiometer

This example will show how to control - real time - the position of a servo motor using an analog input connected to a potentiometer.

We'll map the reading from the potentiometer to the motor output.

As seen in the LED brightness example, the analog input varies from 0 to 1023, however the servo motor ranges from 0 to 180 degrees.

We'll need to use the Map function again to map the 1024 possible input values with the 181 possible output ones.

<Optional> Servos – Control with potentiometer

```
1  #include <Servo.h>
2  Servo myservo;
3  int pos = 0;
4  void setup() {
5    myservo.attach(9);
6    pinMode(A0,INPUT);
7  }
8  void loop() {
9    pos = map(analogRead(A0),0,1023,0,180);
10   myservo.write(pos);
11   delay(20);
12 }
```

Servo: motor control with potentiometer

Code explained:

Line 1 we include the servo library.

Line 2 is where we create an object named myservo based in the Servo library.

Line 3 we create a variable named pos (short for position) and attribute the value 0 to it.

Inside the setup we need to attach the motor (line 5) and define pin A0 as an input (line 6).

Line 9: we attribute to the variable pos the mapping result of the analog reading of the pin A0, originally ranging from 0 to 1024 into the new range of 0 to 180.

> Note the difference in this example and the potentiometer one: In that case we used two lines of code – one to read the analog pin and a second line to

do the mapping.

This case is shorter as we did it all in just one line of code.

You'll learn how to do optimizations like this with time, but until you're comfortable you can absolutely break this into 2 different lines.

Line 10 is where we write the content of the pos variable to the myservo object, giving the servo a 20 millisecond delay to get there (line 11).

What's next?

- What do you want to build?
- Different boards, different needs
  - Size, Speed, Memory, #Ports
  - Wifi - IoT projects, CNC, Robots
- Advanced topics:
  - Memory management
  - Manual multi threading
  - Interrupts / Semaphore controls
  - Bit shifting operations
  - Barebone projects

@Snoco Makers

- Electronics
- NeoPixels
- Radio control
- ATTiny microcontrollers (barebone)
- Fritzing (prototype software)
- IoT 101
- Digital analyzer
- Laser cutter
- Robotics club

Internet references:
YouTube channels (EEV blog, Educ8TV, etc)
Instructables, Hackster, sparkfun, Adafruit, Arduino.cc

Next Steps

In this workshop you've learned what is an Arduino, you setup your computer and a board, wrote some lines of code and could witness how the code you loaded is able to control physical devices.

This was only the beginning! Now what's next?

There is not a simple answer to this question. It depends on what types of project you want to build.

For each project there will be a different board:

- **Size**: boards comes in several different sizes, you can shrink your project to the point of having only one component → the Microprocessor itself! See AtTiny 85.
- **Speed**: if you're processing images, speed is really important. But that's not always the case. Remember: the faster the Microprocessor more power it will require!

- **Memory**: sketches like the ones we created in this workshops are super small, but when you create professional projects you might not be able to fit it in an Arduino Uno board just because there's not enough memory.
- **#Ports**: As we saw I this workshop, Arduino Uno has 14 I/Os. AtTiny will have only 6 while the Arduino Mega has 54. How many do you need?
- **WIFI**: the ability to connect your board to the Internet will allow you to think about connected devices and IoT scenarios.

At **Snoco Makerspace** space you'll have options to dig deeper in the Arduino world. Some of the Workshops / Meetings we have:

- **NeoPixels**: how to connect them, what libraries to use, how to get started with Neopixels
- **Radio control**: how to use radio expansion boards to allow P2P communication to your projects
- **ATTiny microcontrollers** (barebone): Shrink your project going barebone! How to use the AtTiny 85 to create a project.
- **Fritzing** (prototype software): how to draw your electronics diagram, the PCB board and send that for fabrication using Fritzing, a free software used in this material to create all wiring pictures.
- **Digital analyzer**: Sometimes the Serial monitor can't help you in your debugging adventures. Instruments like a multimeter are limited when it comes to high speeds. The digital analyzer is a tool that will allow you see what is happening on several Arduino pins at once. A must for more advanced projects.
- **Laser cutter**: Mandatory training if you're intending to use the laser cutter in your projects.
- **Robotics club**: an ongoing series of meeting for robot lovers! Create your own robot and have them compete against other robots in the club.

Some useful references where you can learn more:

1. Adafruit: www.adafruit.com
2. Instructables: www.instructables.com
3. Hackster: www.huckster.io
4. Sparkfun: www.sparkfun.com
5. Arduino; www.Arduino.cc

You tube channels worth watching:

1. EEV Blog

2. Explaining Computers
3. Andreas Spiess
4. DYI Perks
5. Ben Eater
6. Educ8s.tv