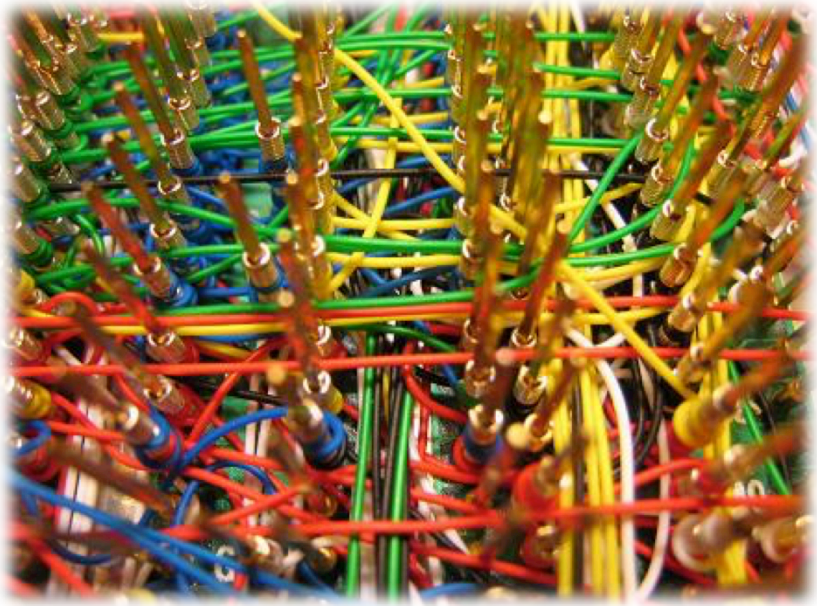
Electronic and electromechanical prototyping

Electronic and electromechanical prototyping

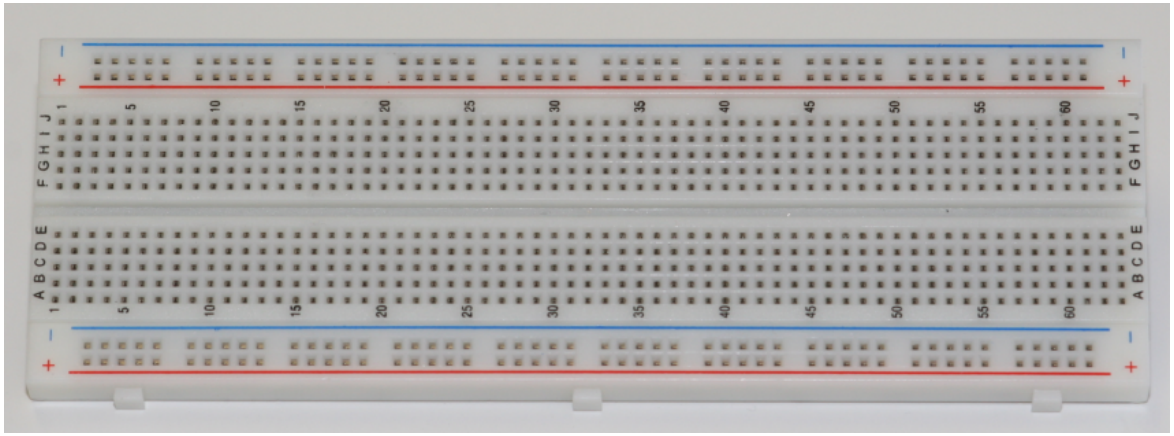
If you wanted to build a circuit prior to the 1960s, chances are you would have used a technique called **wire-wrap**.

Wire wrap is a process that involves wrapping wires around conductive posts attached to a **perfboard**. As you can see, the process can get rather complex very quickly. Although this method is still used today, there is something that makes prototyping much easier, **breadboards**!



What is a Breadboard?

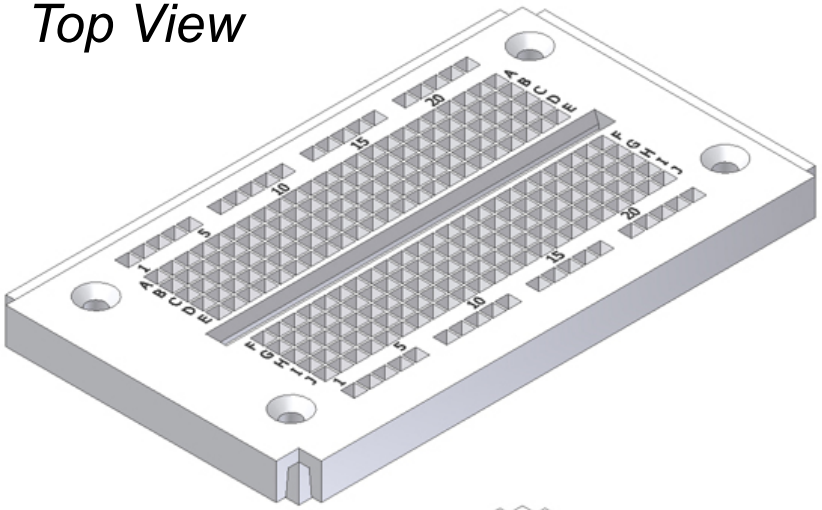
A breadboard, sometimes called a proto-board, is a reusable platform for temporarily built electronic circuits.



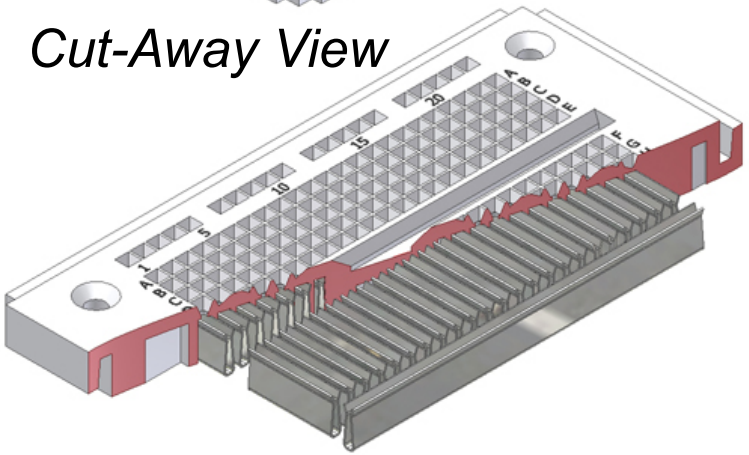
How A Breadboard Works

- Electric component leads and the wire used to connect them are inserted into holes that are arranged in a grid pattern on the surface of the breadboard.
- A series of internal metal strips serve as jumper wires. They connect specific rows of holes.

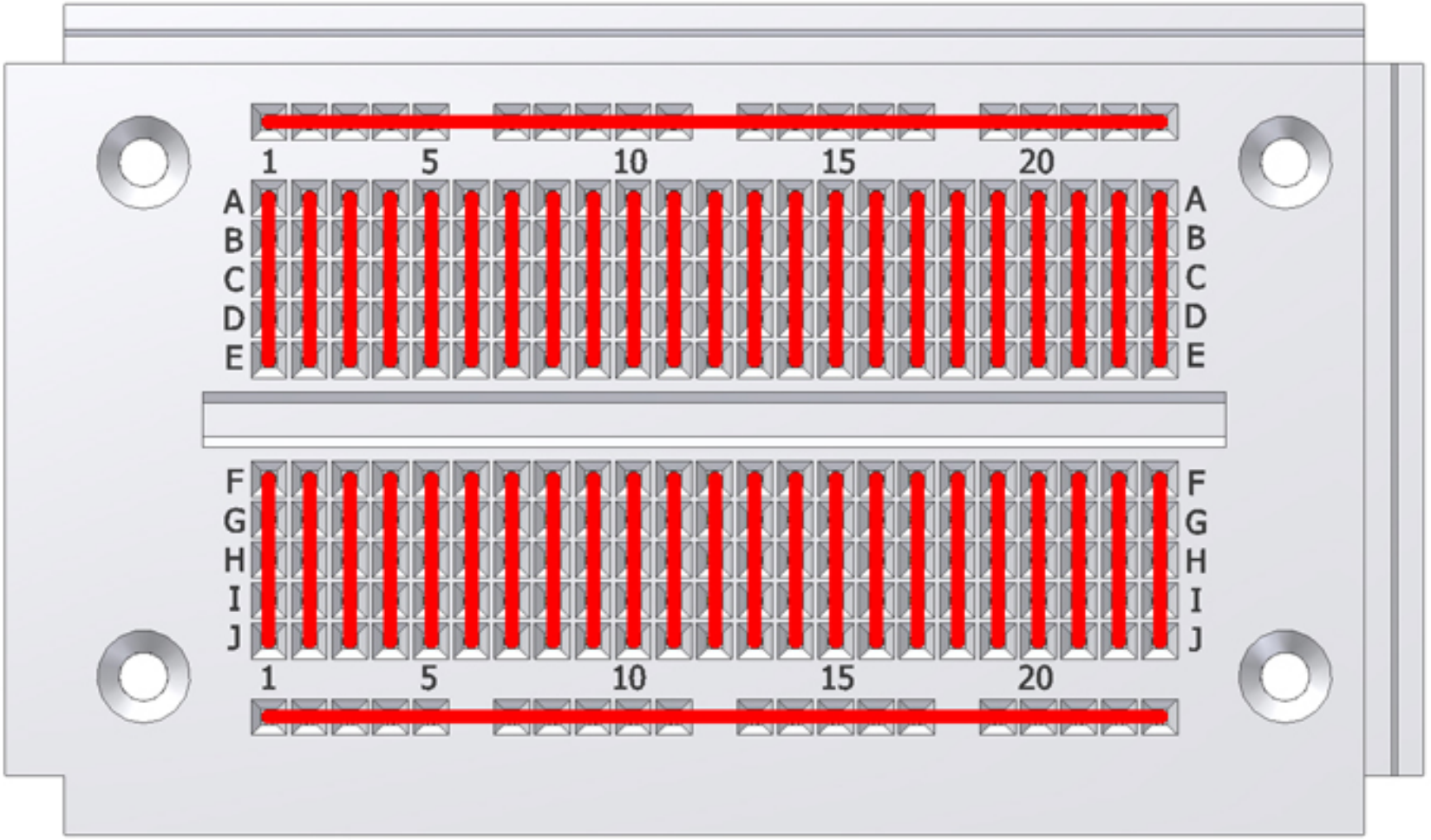
Top View



Cut-Away View



Breadboard Connections



Why Breadboard?

1) It takes less time (and money) to breadboard a circuit than to design and fabricate a printed circuit board (PCB).

   Because of the cost, a PCB should be reserved for the final working design.

2) As a complement to circuit simulation, breadboarding allows the designer to see how, and if, the actual circuit functions.

Why Breadboard?

3) Breadboards give the designer the ability to quickly change components during development and testing, such as swapping resistors or capacitors of different values.

4) A breadboard allows the designer to easily modify a circuit to facilitate measurements of voltage, current, or resistance.

Components: Resistances

The principal job of a resistor within an electrical or electronic circuit is to "resist" or **regulate the flow of electrons (current)** through them by using the type of conductive material from which they are composed.

Resistors can also be connected together in various **series and parallel combinations** to form resistor networks which can act as voltage droppers, voltage dividers or current limiters within a circuit.

Resistors are what are called "**Passive Devices**", that is they contain no source of power or amplification but only attenuate or reduce the voltage or current signal passing through them. This *attenuation results in electrical energy being lost in the form of hea*t as the resistor resists the flow of electrons through it.

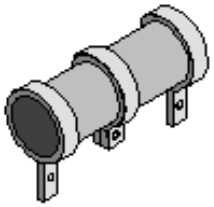
Components (2): Types of Resistance

All modern fixed value resistors can be classified into four broad groups:

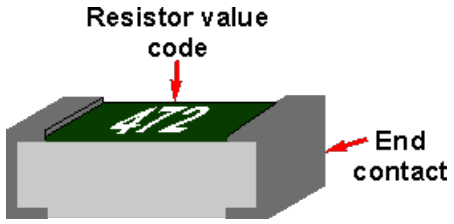
- Carbon Composition Resistor – Made of *carbon dust or graphite paste*, low wattage values
- Film or Cermet Resistor – Made from *conductive metal oxide paste*, very low wattage values
- Wire-wound Resistor – *Metallic bodies* for heatsink mounting, very high wattage ratings
- Semiconductor Resistor – High frequency/precision *surface mount thin film technology*

Carbon

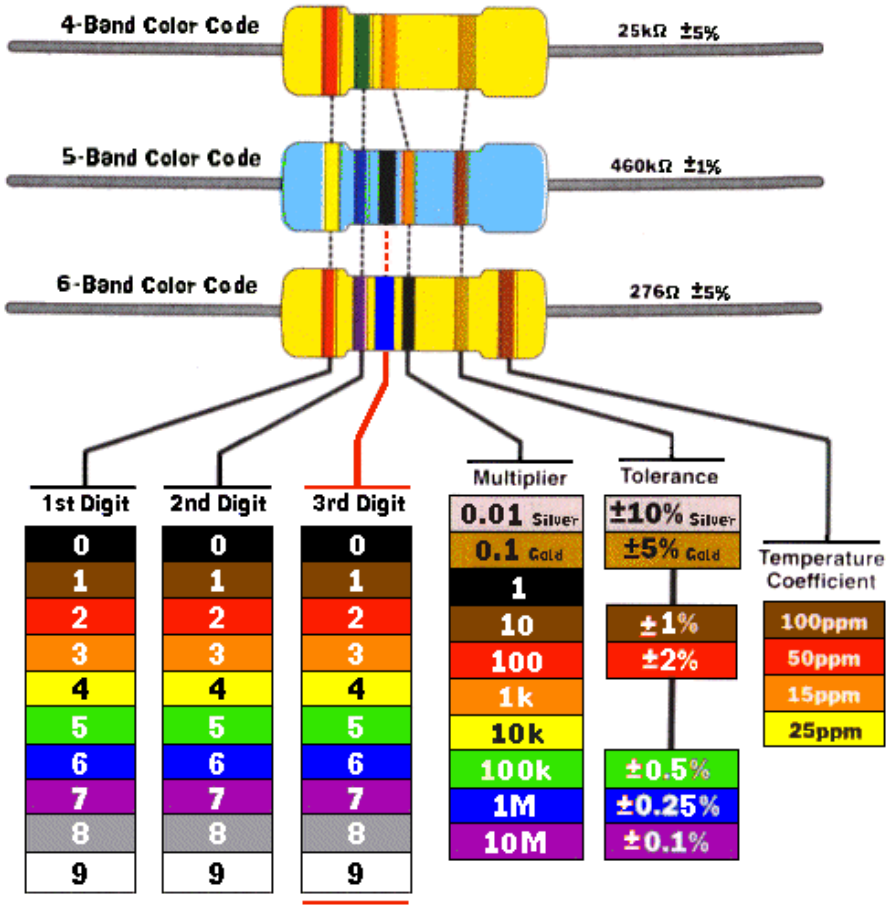
Film



Adiustable wirewound



Resistor Code



Components (3): Capacitors

Basically a capacitor is formed *from two conducting plates separated by a thin insulating layer*. They are manufactured in many forms, styles, and from many materials. Capacitors are widely used in electrical and electronic circuits.

In electronic circuits, small value capacitors are used to couple signals between stages of amplifiers, as components of electric filters and tuned circuits, as parts of power supply systems to smooth rectified current.

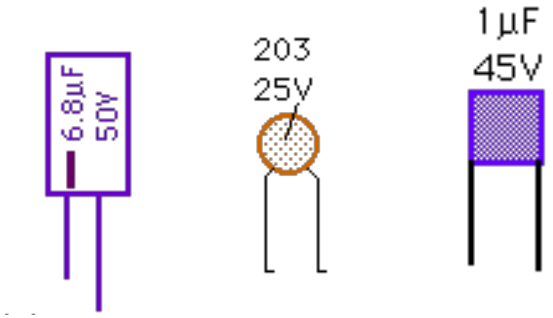
In electrical circuits, larger value capacitors are used for energy storage in such applications as strobe lights, as parts of some types of electric motors, for power factor correction in AC power distribution systems

Components (4): Types of Capacitors

a) Electrolytic:
Electrolytic capacitors are a type of capacitor that is *polarised*. They are able to offer *high capacitance values* - typically above 1µF, and are most widely used for low frequency applications (frequency limit if around 100 kHz) - power supplies, decoupling and audio coupling applications.

b) Ceramic capacitors:
Values range from a few picofarads to around 0.1 microfarads. Ceramic capacitor types are by far the most commonly used type of capacitor being cheap and reliable and their loss factor is particularly low although this is dependent on the exact dielectric in use.

c) Polymer capacitors: Polystyrene, Polyester, Metallised Polyester, Polycarbonate, Polypropylene.

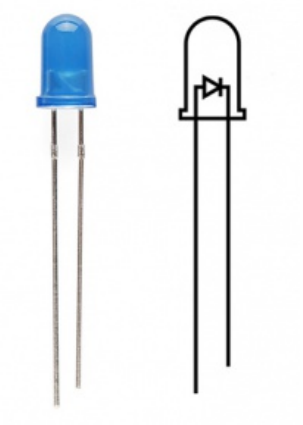


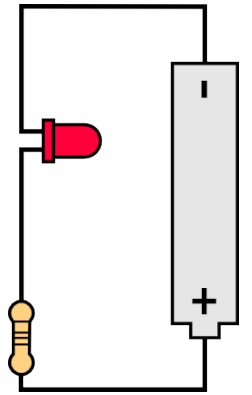
Components (5): LEDs

- LEDs (Light Emitting Diodes), being diodes, will only allow current to flow in one direction. The positive side of the LED is called the “anode” and is marked by having a longer leg. The other, negative side of the LED is called the “cathode.” Current flows from the anode to the cathode and never the opposite direction.

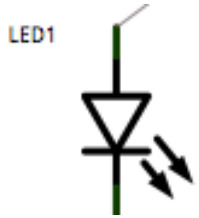
- More Current, More Light: The brightness of an LED is directly dependent on how much current it draws. This means that you can control the brightness of a LED by controlling the amount of current through it.

- If you connect an LED directly to a current source it will try to dissipate as much power as it’s allowed to draw and it will destroy itself. That’s why it’s important to limit the amount of current flowing across the LED with resistors. Resistors limit the flow of electrons in the circuit and protect the LED from trying to draw too much current.

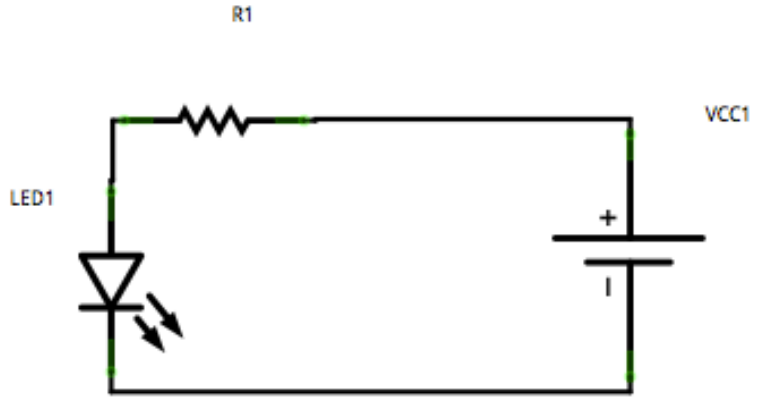


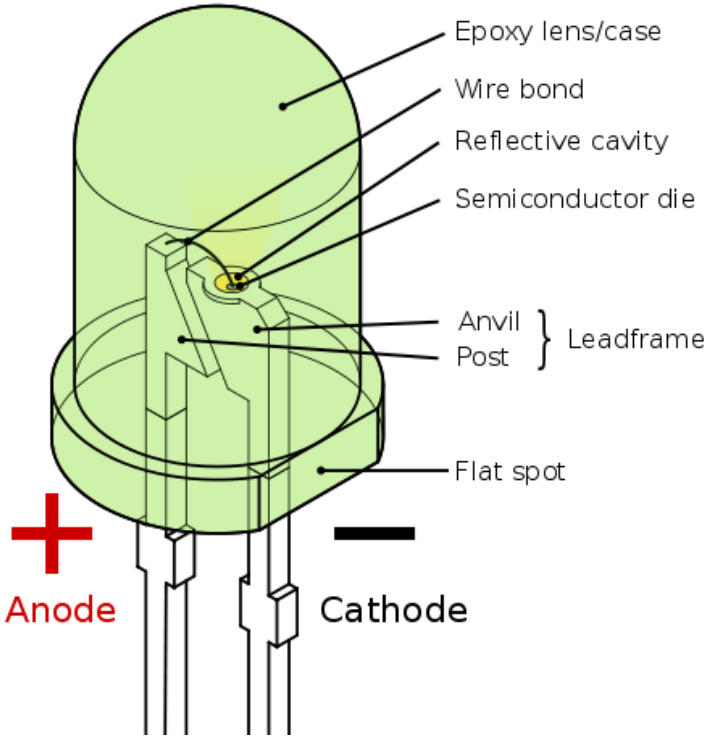


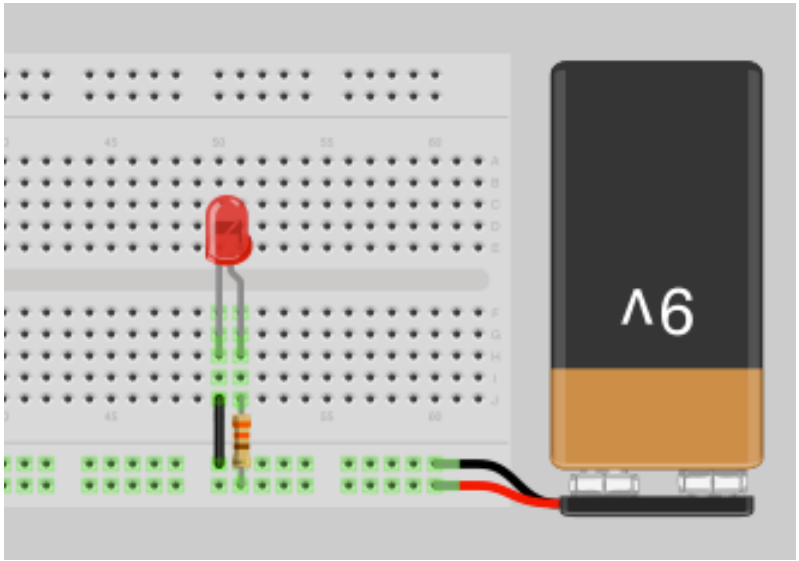
What are LEDs?



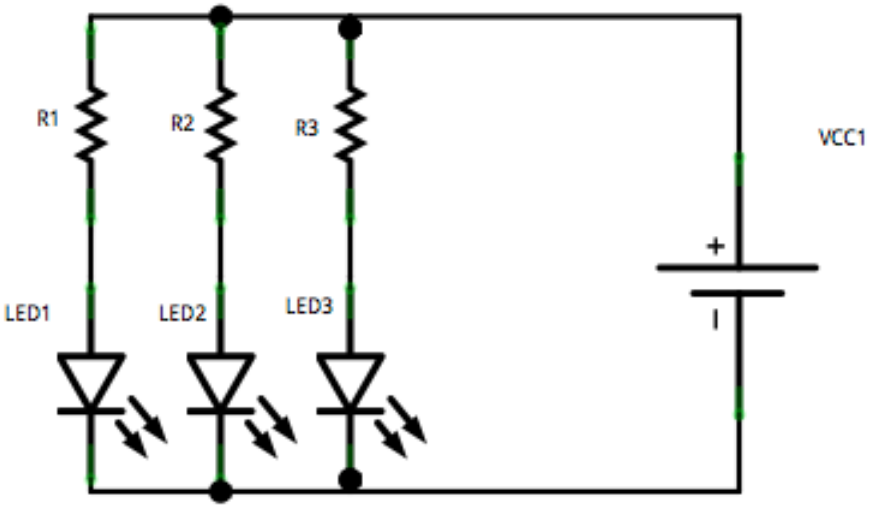
Light Emitting Diodes
Diode Symbol + Arrows for light
Points to ground

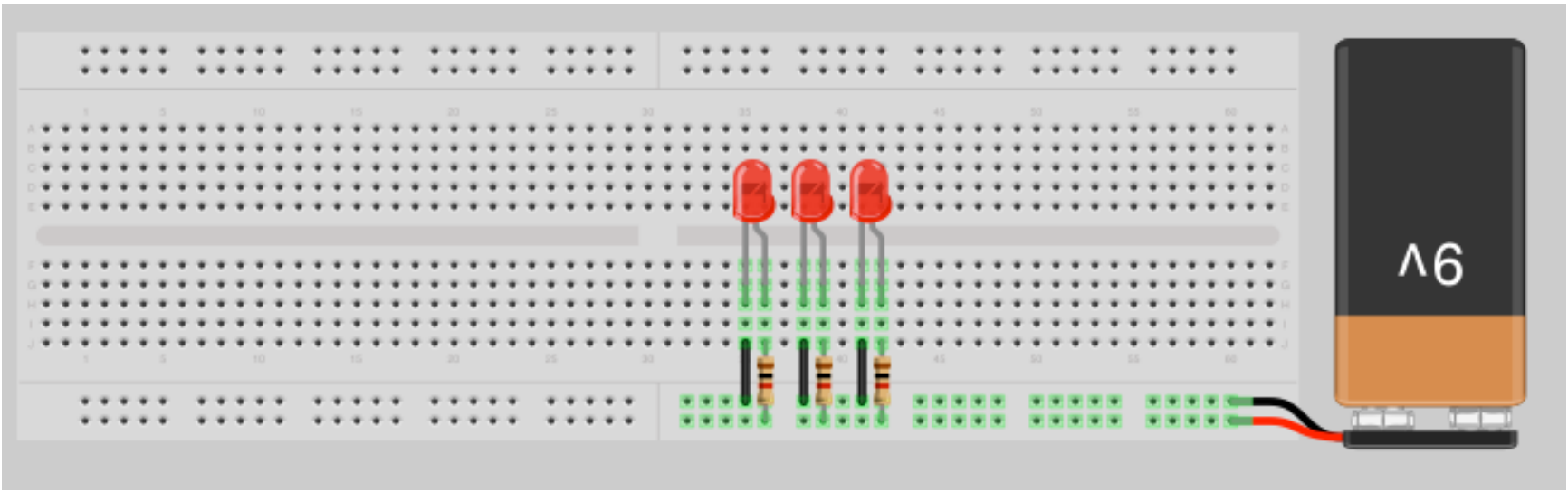




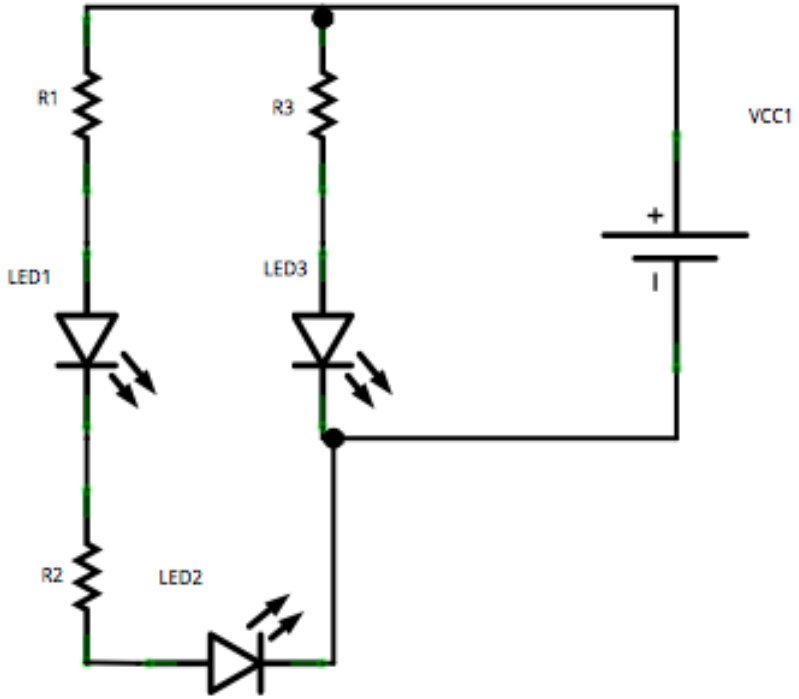


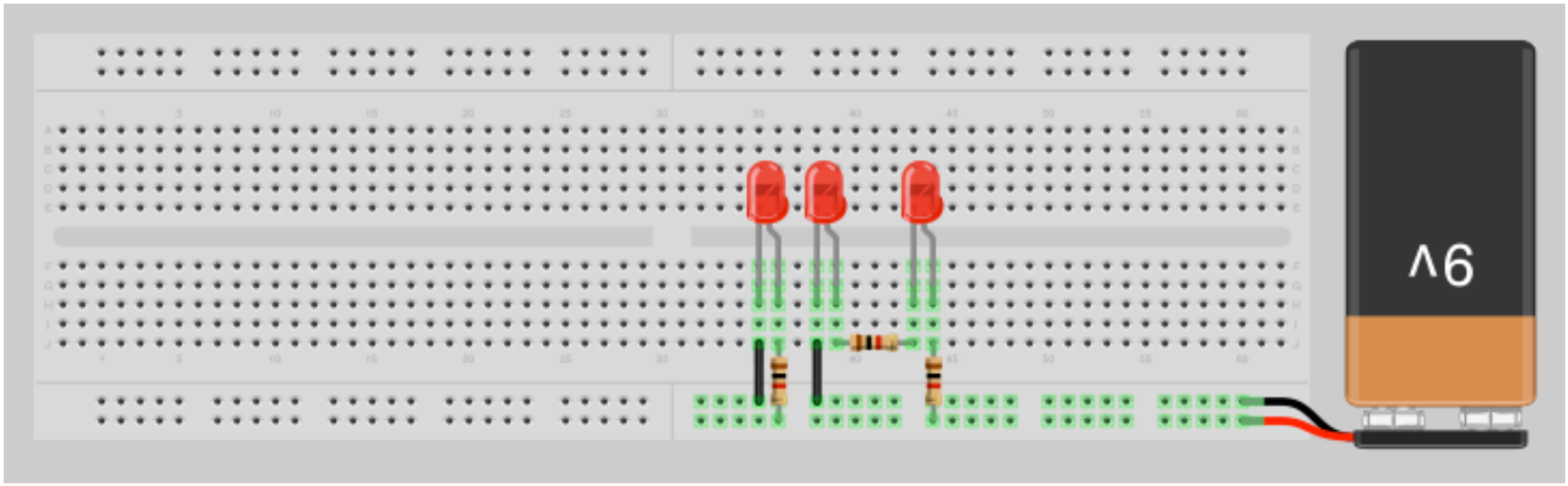
Parallel Circuit example





Parallel and Series Circuit Example





µP and µC

Microprocessor is an IC which has **only the CPU** (central processing unit) inside them i.e. only the processing powers. These microprocessors **don't have RAM, ROM and other peripheral on the chip**. A system designer has to add them externally to make them functional. Application of microprocessor includes Desktop PC's, Laptops, notepads etc.

Microprocessor find applications where **tasks are unspecific** like developing software, games, websites, photo editing, creating documents etc. In such cases the *relationship between input and output is not defined*. They need *high amount of resources* like RAM, ROM, I/O ports etc. The clock speed of the Microprocessor is quite high as compared to the microcontroller. Whereas the microcontrollers operate from a few MHz to 30 to 50 MHz, today's microprocessor operate above 1GHz as they perform complex tasks.

Microcontroller **has a CPU, in addition with a fixed amount of RAM, ROM and other peripherals** all embedded on a single chip. At times it is also termed as a mini computer or a computer on a single chip. Today different manufacturers produce microcontrollers with a wide range of features available.

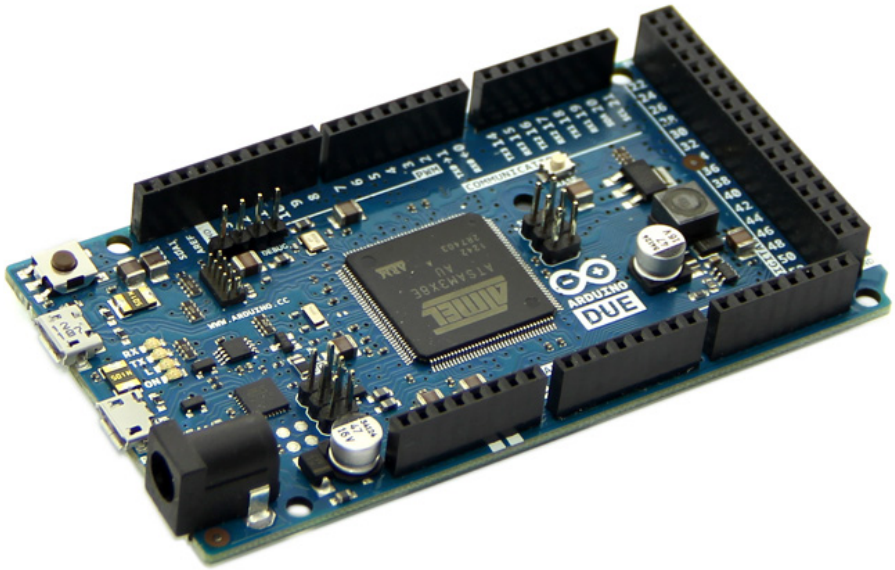
Microcontrollers are designed to perform **specific tasks**. Specific means applications where the *relationship of input and output is defined*. Depending on the input, some processing needs to be done and output is delivered. For example, keyboards, mouse, washing machine, digicam, pendrive, remote, microwave, cars, bikes, telephone, mobiles, watches, etc. Since the applications are very specific, they need *small resources* like RAM, ROM, I/O ports etc and hence can be embedded on a single chip. This in turn *reduces the size and the cost*.

Arduino DUE board

The Arduino Due is a microcontroller board based on the *Atmel SAM3X8E with a ARM Cortex-M3 CPU*.

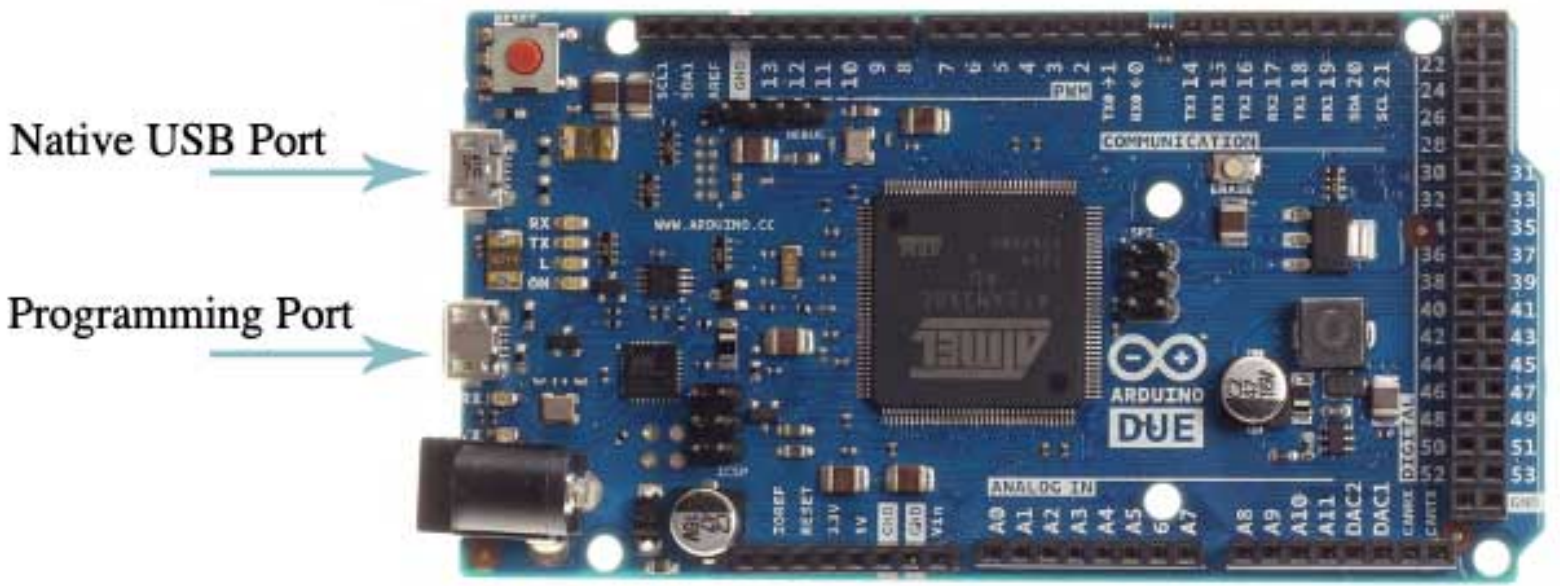
It is the first Arduino board based on a **32-bit ARM core microcontroller**.

It has *54 digital input/output pins* (of which 12 can be used as PWM outputs), *12 analog inputs*, *4 UARTs* (hardware serial ports), a *84 MHz clock*, an *USB OTG capable connection*, 2 DAC (digital to analog), 2 TWI, a power jack, an SPI header, a JTAG header, a reset button and an erase button.



https://www.arduino.cc/en/Main/arduinoBoardDue

Arduino DUE

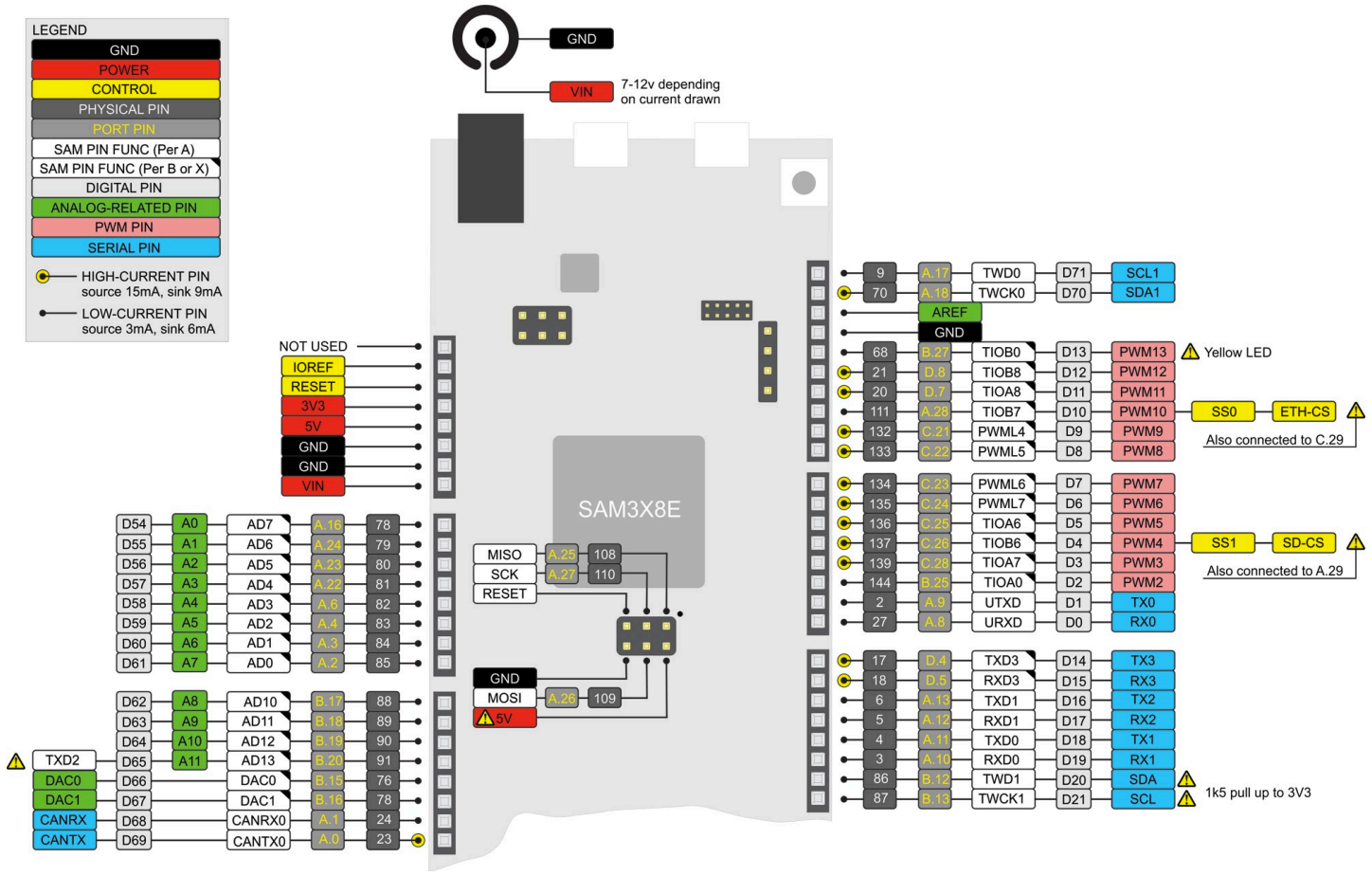

Native USB Port
Programming Port
ARDUINO
DUE
COMMUNICATION
ANALOG IN
DIGITAL

Serial Communication – USB (2)

- **Programming port**: The programming port is connected to an ATmega16U2, which provides a virtual COM port to software on a connected computer. It uses the 16U2 as a USB-to-serial chip connected to the first UART of the SAM3X (RX0 and TX0).
  The 16U2 has two pins connected to the Reset and Erase pins of the SAM3X. Opening and closing the Programming port connected at 1200bps triggers a "*hard erase" procedure of the SAM3X chip*, activating the Erase and Reset pins on the SAM3X before communicating with the UART. This is the recommended port for programming the Due.

- **Native port**: The Native USB port is connected directly to the SAM3X. Opening and closing the Native port at 1200bps triggers a *'soft erase*' procedure: the flash memory is erased and the board is restarted with the bootloader. Opening and closing the native port at a different baud rate will not reset the SAM3X.

Pin map


LEGEND
GND
POWER
CONTROL
PHYSICAL PIN
PORT PIN
SAM PIN FUNC (Per A)
SAM PIN FUNC (Per B or X)
DIGITAL PIN
ANALOG-RELATED PIN
PWM PIN
SERIAL PIN
HIGH-CURRENT PIN source 15mA, sink 9mA
LOW-CURRENT PIN source 3mA, sink 6mA
GND
VIN
7-12v depending on current drawn
NOT USED
IOREF
RESET
3V3
5V
GND
GND
VIN
SAM3X8E
MISO
SCK
RESET
GND
MOSI
5V
AREF
GND
Yellow LED
SS0
ETH-CS
Also connected to C.29
SS1
SD-CS
Also connected to A.29
1k5 pull up to 3V3
TXD2
DAC0
DAC1
CANRX
CANTX

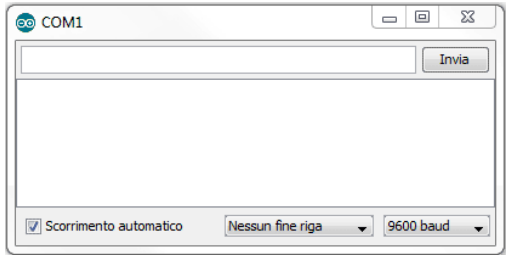
Arduino IDE

The IDE (**Integrated development environment**) allows us to write, compile and transfer our programs on the arduino board.

Verify: performs verification of the written code and its compilation;

Load : it loads the compiled firmware on board;

Serial Opening



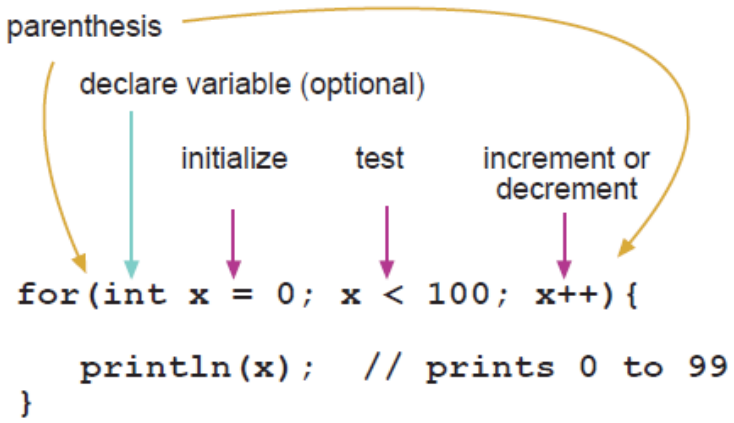
For Loop

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.

There are *three parts to the for loop header*:

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

- 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.



While Loop

while loops will loop continuously, until the expression inside the parenthesis becomes false.
This could be in your code, such as an incremented variable or an external condition, such as testing a sensor.

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

Example:

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

If/else Instruction

if/else allows greater control over the flow of code than the basic if statement, by allowing **multiple tests** to be grouped together. 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.

```
If (pinFiveInput < 500)
{  // action A
}
else if (pinFiveInput >= 1000)
{  // action B
}
else
{  // do Thing C
}
```

Switch

switch...case controls the flow of programs by allowing programmers **to specify different code that should be executed in various conditions**. In particular, a switch *statement compares the value of a variable to the values specified in case statements*. When a case statement is found whose value matches that of the variable, the code in that case statement is run.

The break keyword exits the switch statement, and is typically used at the end of each case. Without a break statement, the switch statement will continue executing the following expressions ("falling-through") until a break, or the end of the switch statement is reached.

```
switch (var) {
   case 1:
     //do something when var equals 1
     break;
   case 2:
     //do something when var equals 2
     break;
   default:
     // if nothing else matches, do the default
     // default is optional
   break;
 }
```

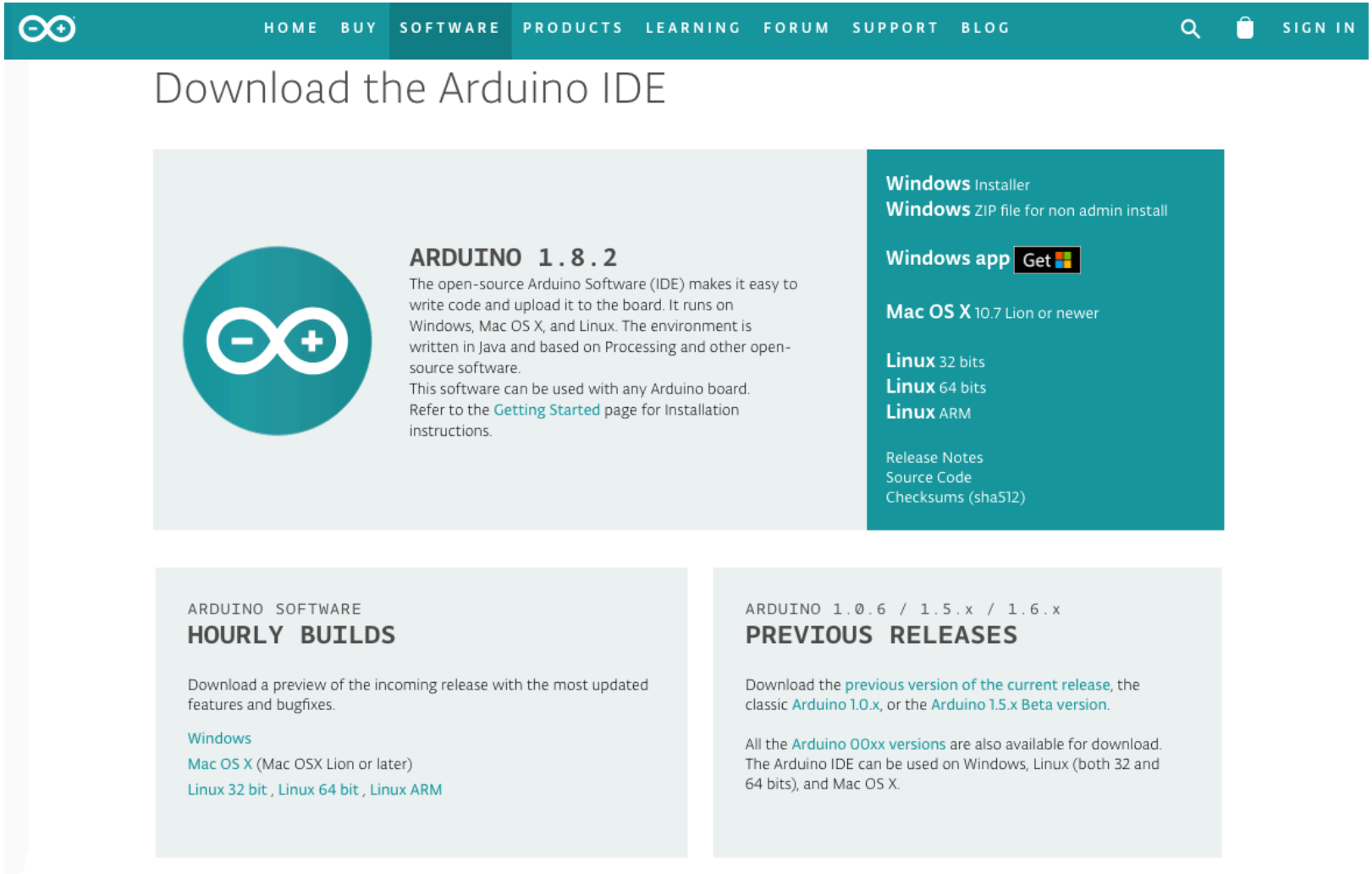
PROGRAMMING WITH ARDUINO

Software Arduino

The Arduino Programming Environment (IDE)

The Arduino Integrated Development Environment - or Arduino Software (IDE) - contains a text editor for writing code, a message area, a text console, a toolbar with buttons for common functions and a series of menus. It connects to the Arduino hardware to upload programs and communicate with them.

Software - Download



Software - Installation

- Windows

  arduino.cc/windows

  Installation for: Windows 7, Vista, e XP

- Mac OS X

  arduino.cc/mac

  Installation for: OS X 10.7 or newer

- Linux

  arduino.cc/linux

Communication with Arduino

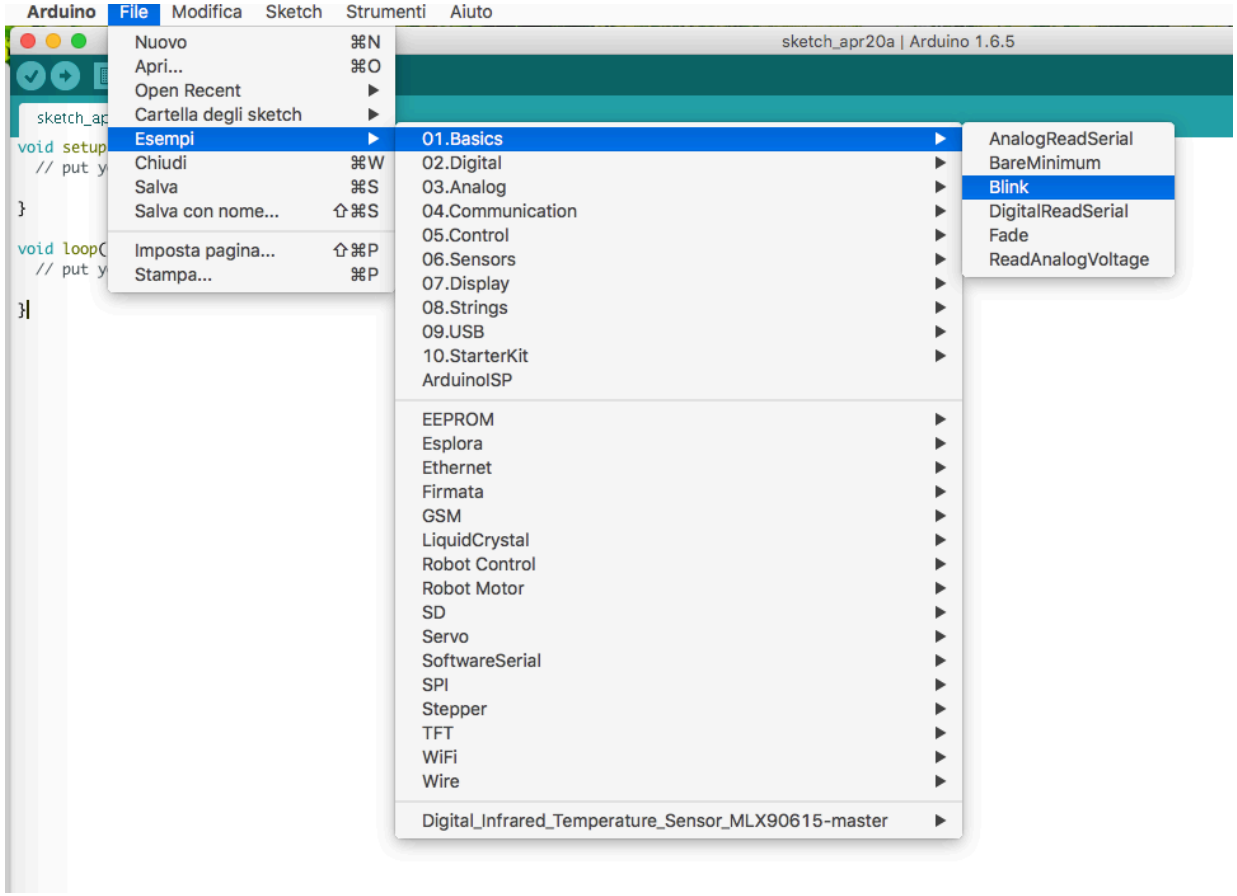
- Launch the Arduino IDE (double click)

Arduino Program Development

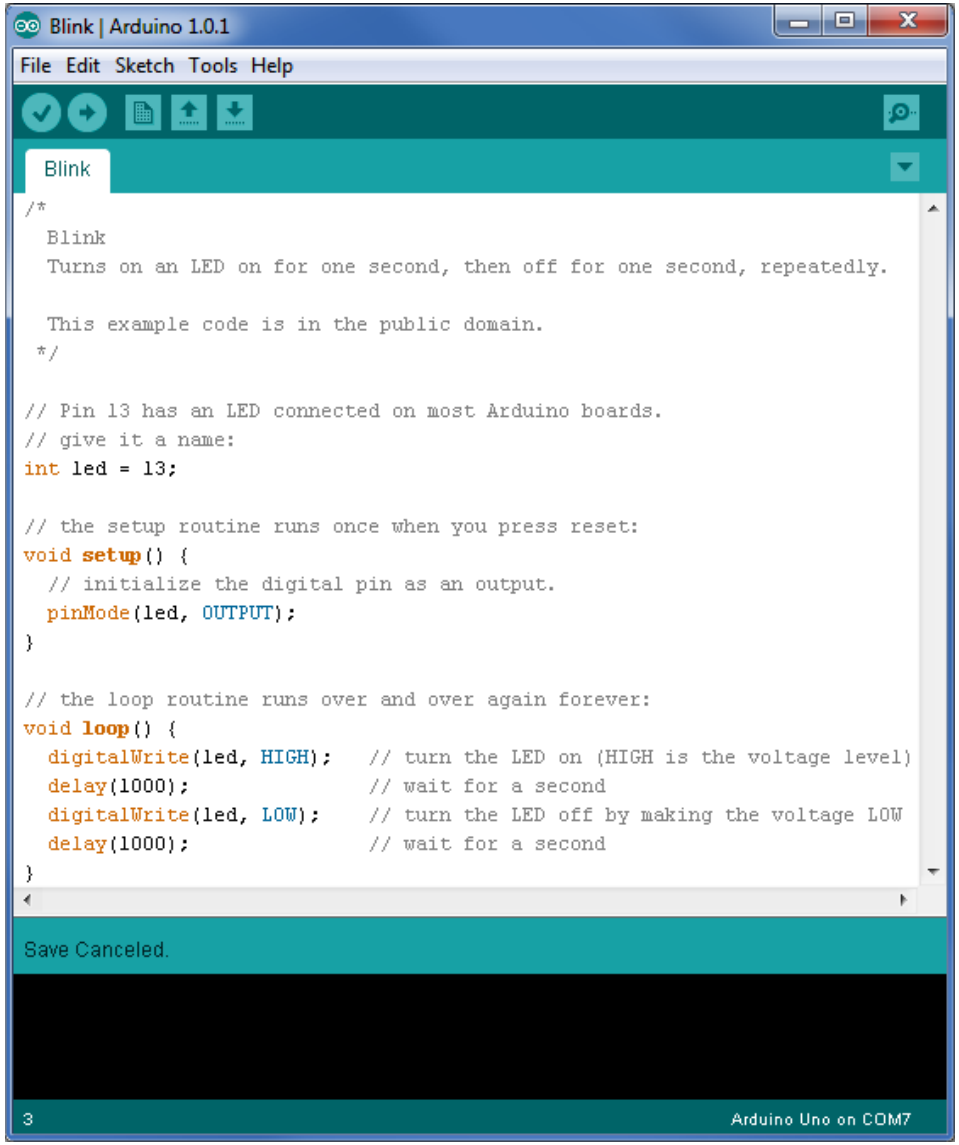
- Based on C++ without 80% of the instructions.
- A handful of new commands.
- Programs are called 'sketches'.
- Sketches need two functions:
  - void setup( )
  - void loop( )
- setup( ) runs first and once.
- loop( ) runs over and over, until power is lost or a new sketch is loaded.

• Open the sketch

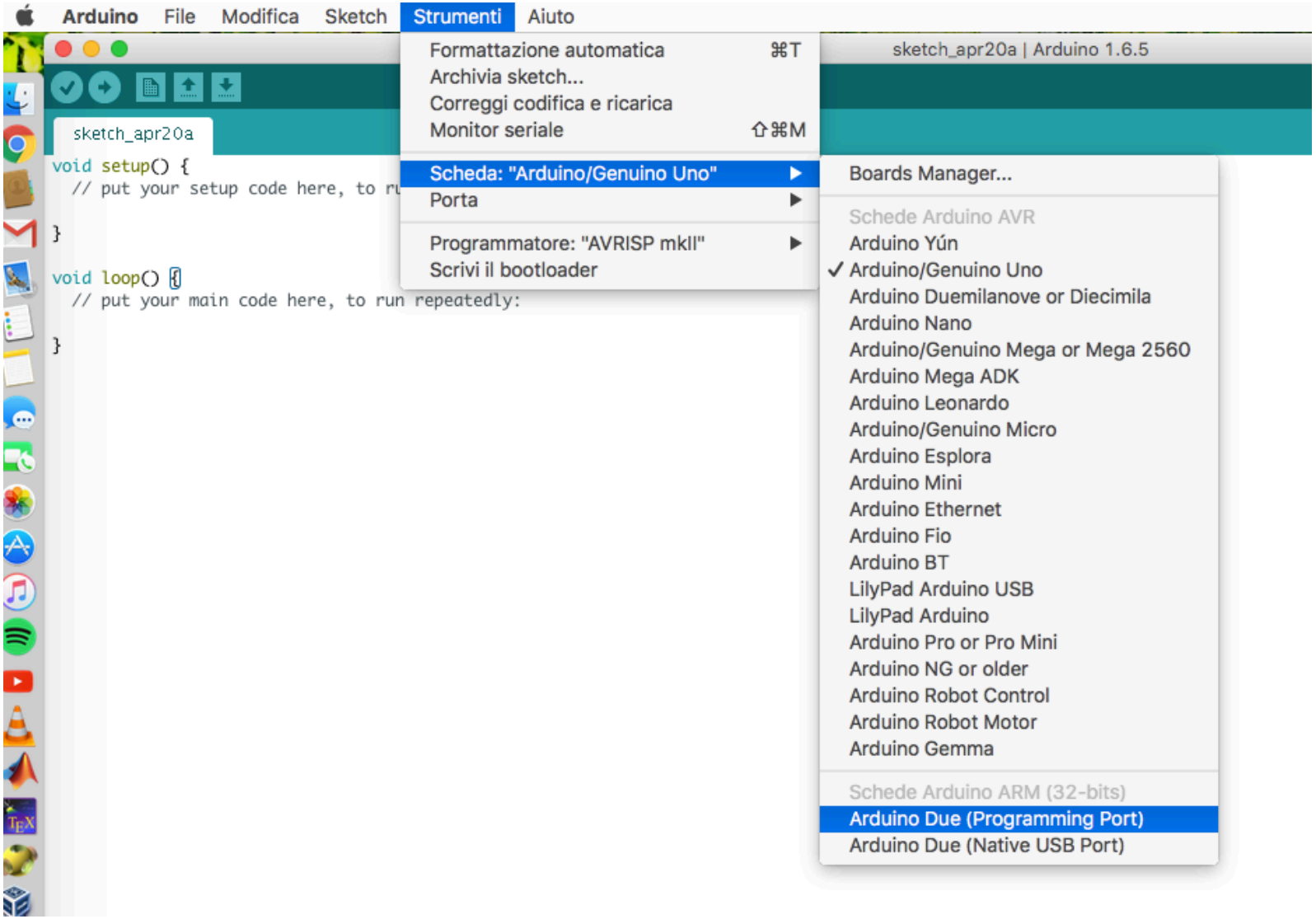
- Numerous sample sketches are included in the compiler
- Located under File, Examples
- Once a sketch is written, it is uploaded by clicking on File, Upload, or by pressing <Ctrl> U



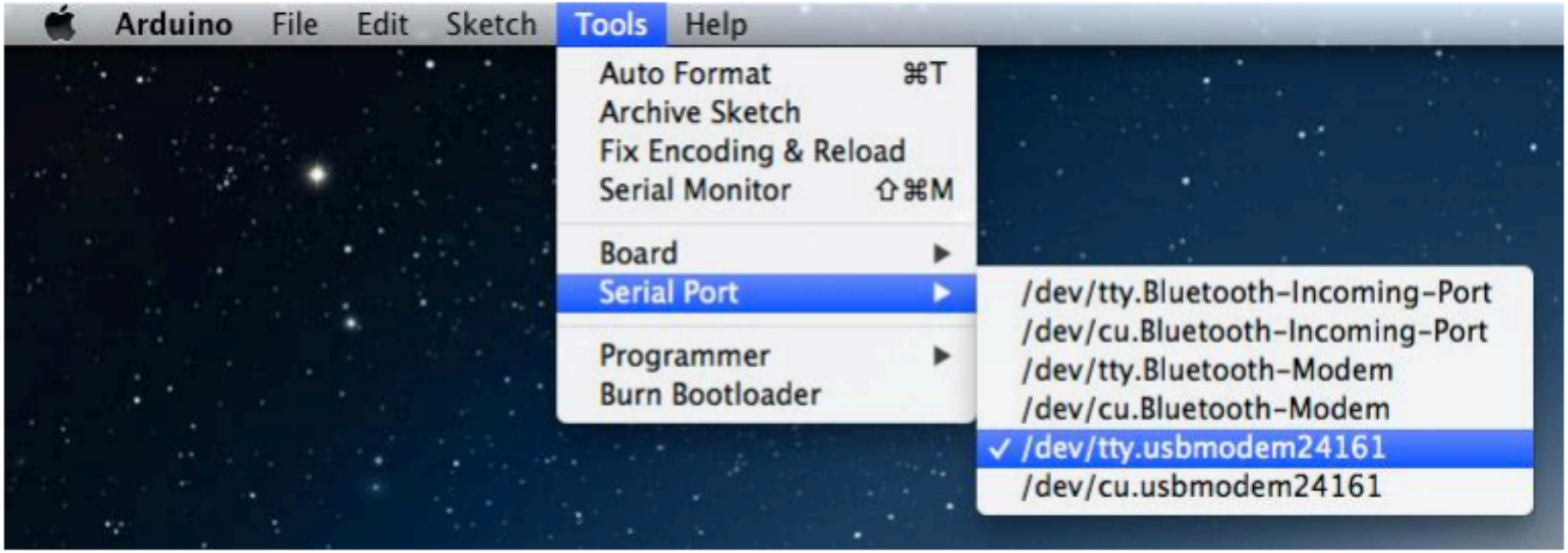
- Open the Blink sketch



- Select the Board

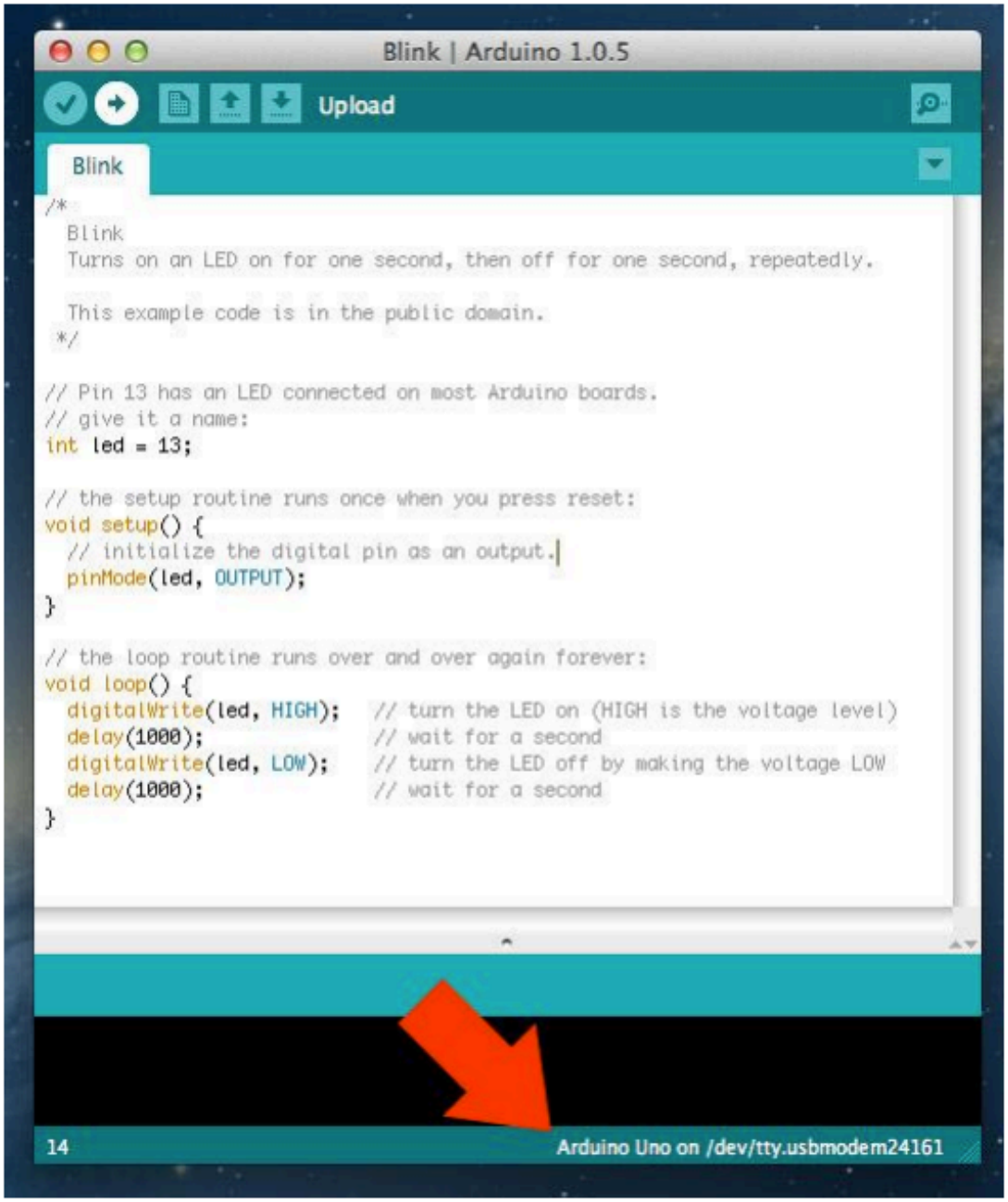


- Select the Serial Port

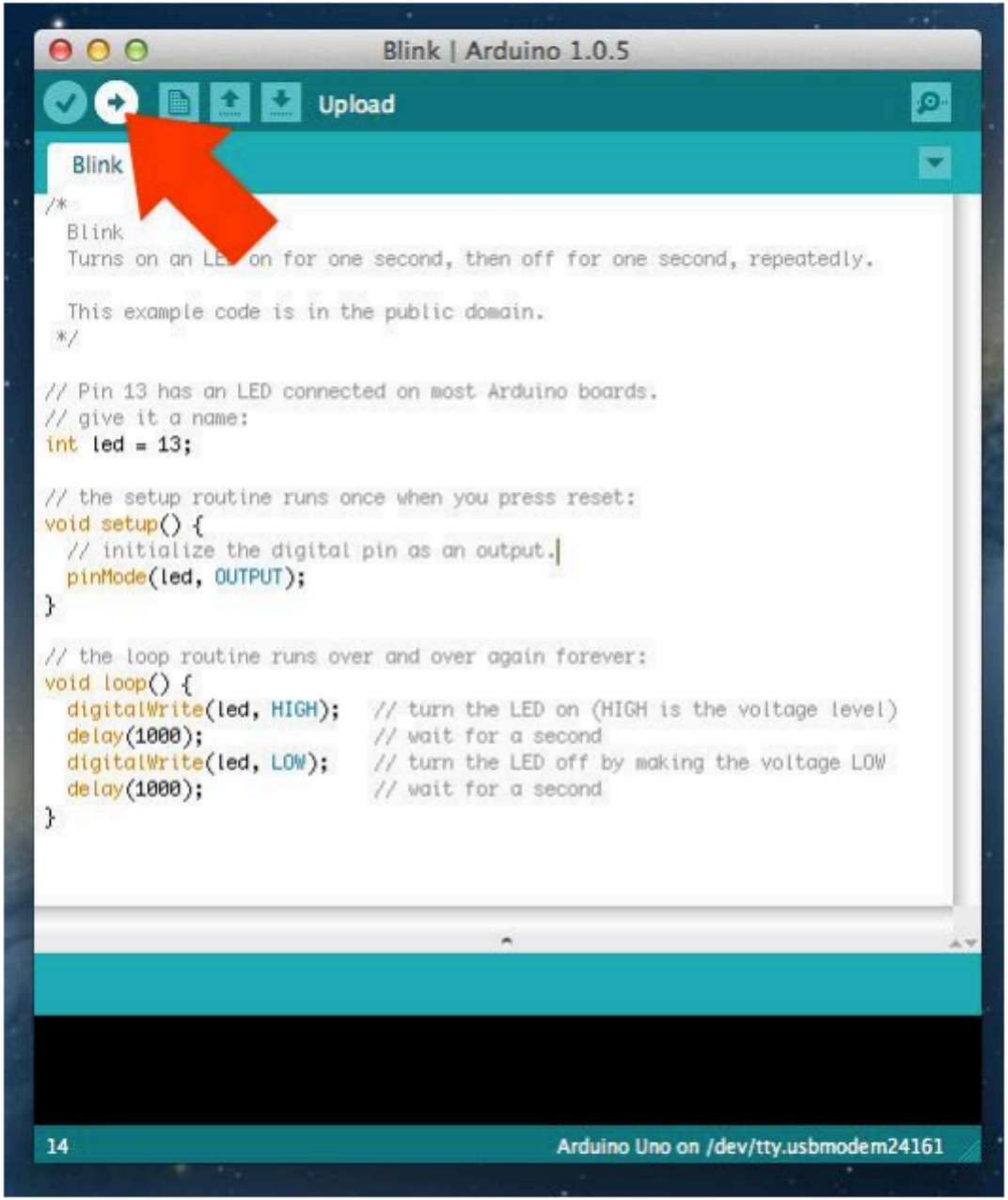


- **Mac**:
  You can indifferently choose between /dev/tty.usbmodemXXXXX or /dev/cu.usbmodemXXXXX
- **Windows**:
  There are one or more COM ports:
  choose the one with the higher number if it does not work try with the other proposals.

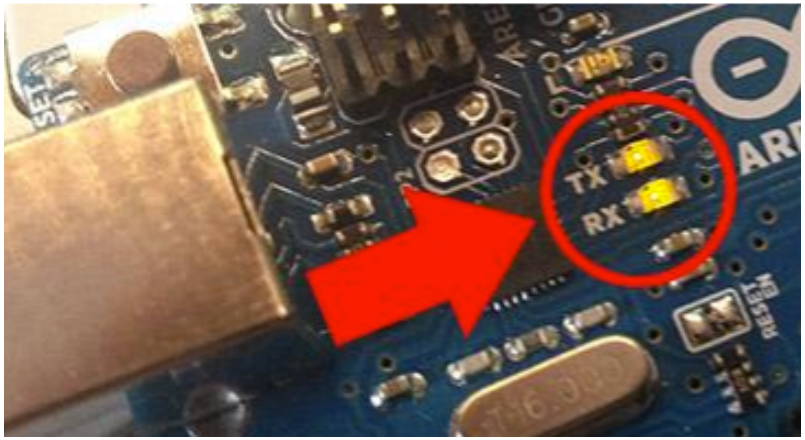
- The connection to the serial port is reported in the code window in bottom right

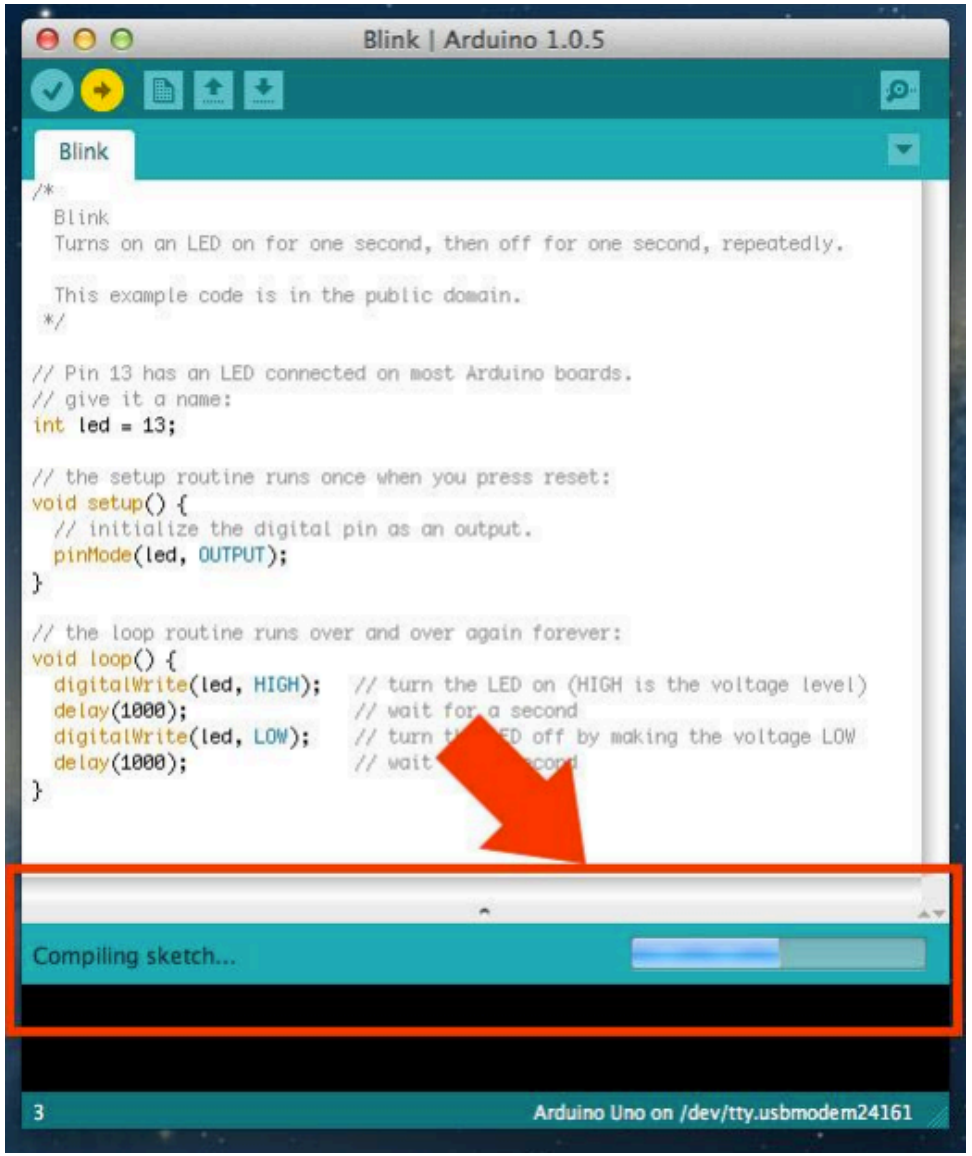


- Loading the Blink sketch on the board through the Upload button

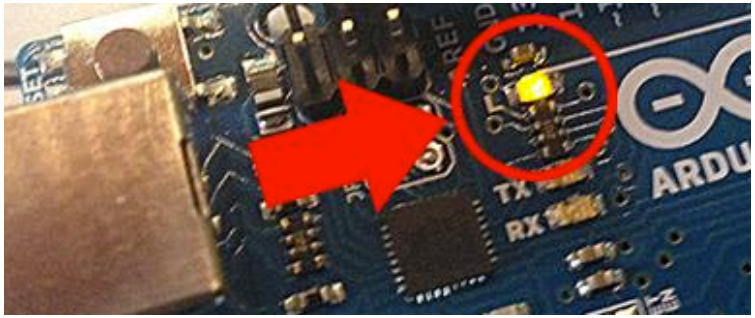


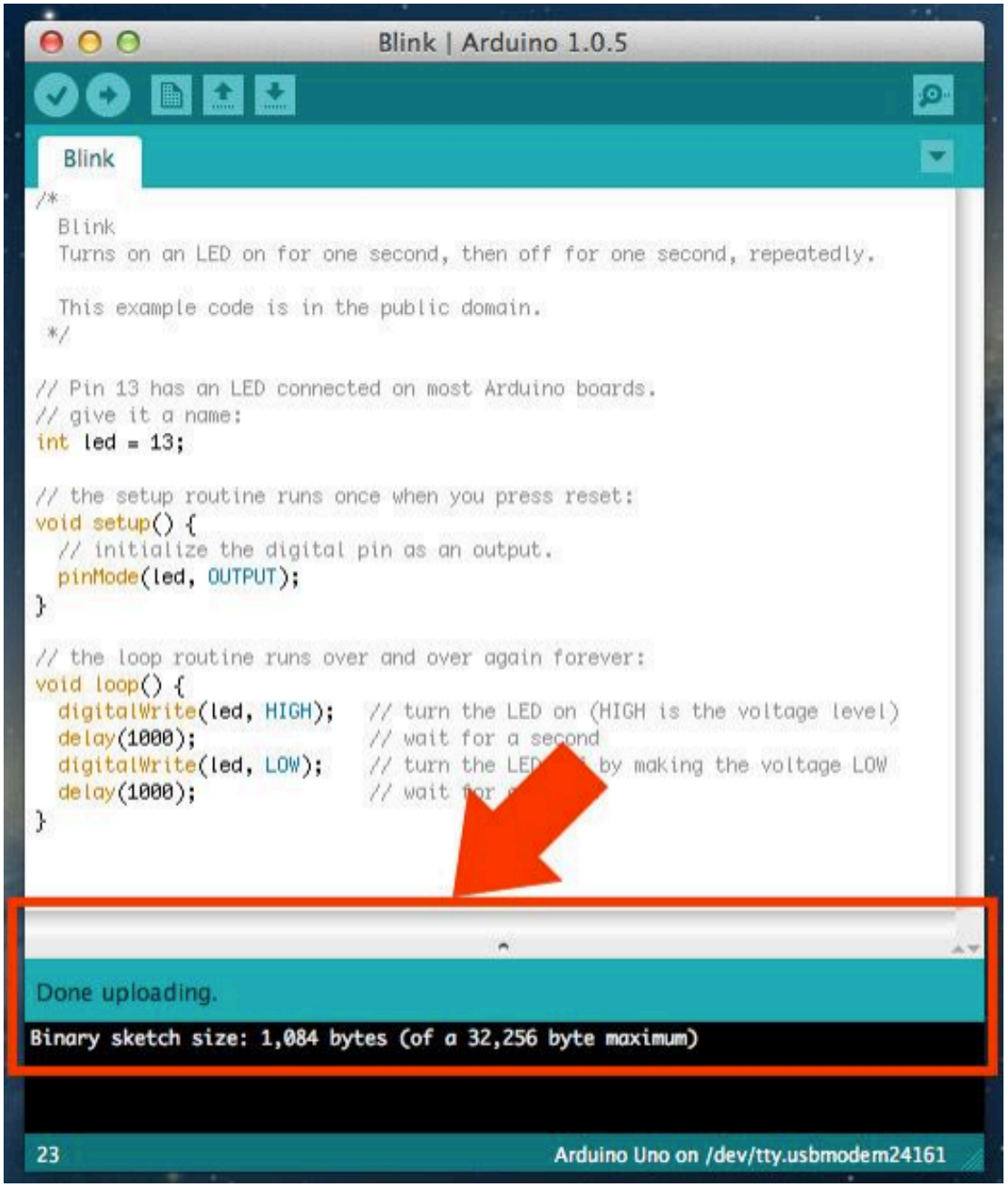
- It will take a few seconds, during this operation you will see that the LEDs RX and TX (receive and transmit)flash.





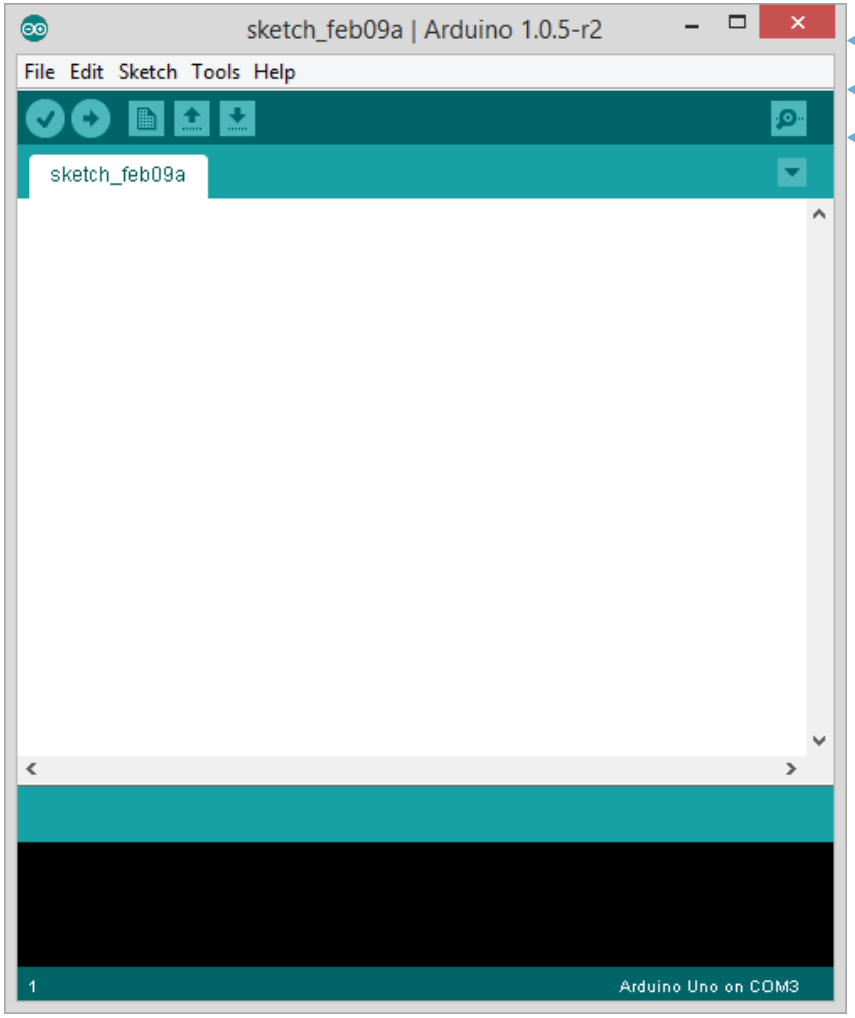
If everything will be succesfull you will be returned message "Done uploading.” in the staus bar, and the LED L starts flashing





Programming

Parts of the IDE main screen



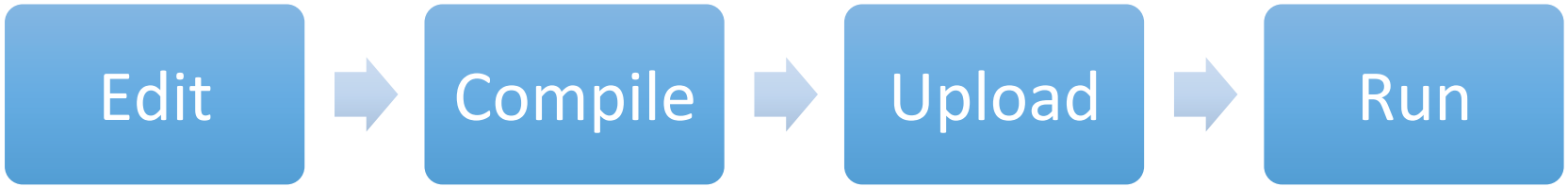
Name of current sketch

Main menus

Action buttons/icons

- Verify (AKA compile)
- Upload (send to Arduino)
- Start a new sketch
- Open a sketch (from a file)
- Save current sketch (to a file)
- Open Serial Monitor window

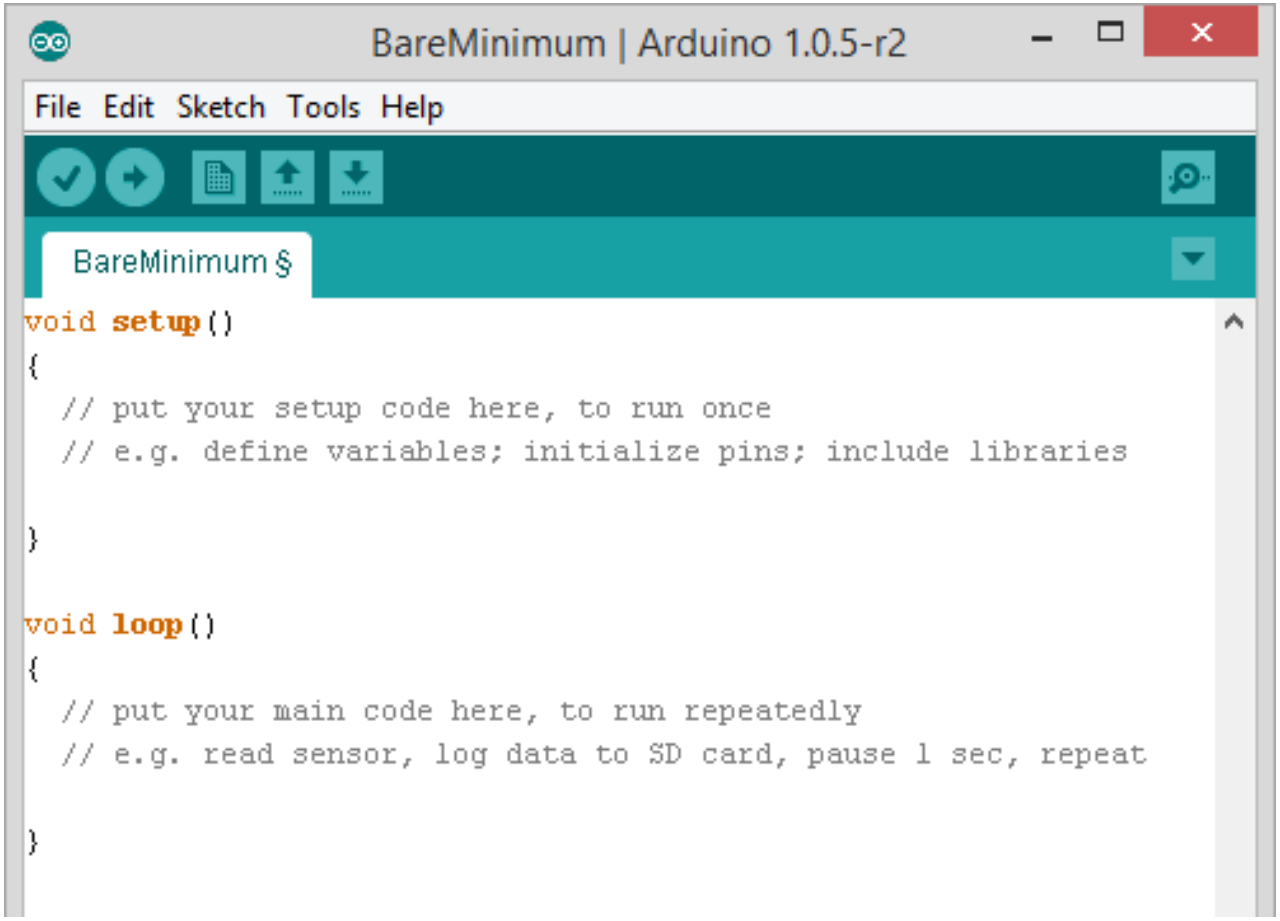
The development cycle is divided into 4 phases:



Compile: Compile means to translate the sketch into machine language, also known as object code

Run: Arduino sketch is executed as soon as terminates the step of uploading on the board

The structure of an Arduino Sketch



- The first one is “**setup()**”. Anything you put in this function will be executed by the Arduino just once when the program starts.

- The second one is “**loop()**”. Once the Arduino finishes with the code in the **setup()**function, it will move into **loop()**, and it will continue running it in a loop, again and again, until you reset it or cut off the power.

Arduino Specific Functions

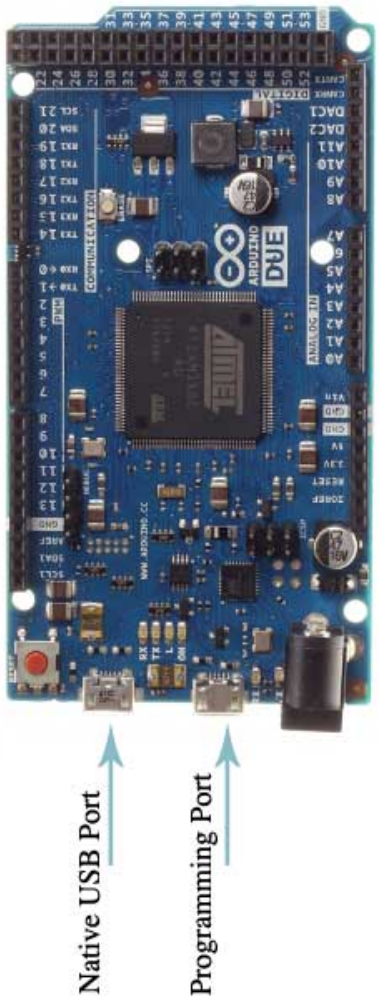
- **pinMode(*pin*, *mode*)**
  - Configures a digital pin to read (input) or write (output) a digital value
- **digitalWrite(*pin*, *value*)**
  - Writes the digital value (HIGH or LOW) to a pin set for output
- **digitalRead(*pin*)**
  - Reads a digital value (HIGH or LOW) on a pin set for input
- **analog versions of above**
  - **analogRead's** range is 0 to 1023 (for Arduino Uno)
  - The Due and the Zero have 12-bit ADC capabilities that can be accessed by changing the resolution to 12. This will return values from analogRead() between 0 and 4095.
- **serial commands**
  - print, println, write, delay
- Other example

  https://www.arduino.cc/en/Reference/HomePage

First Program

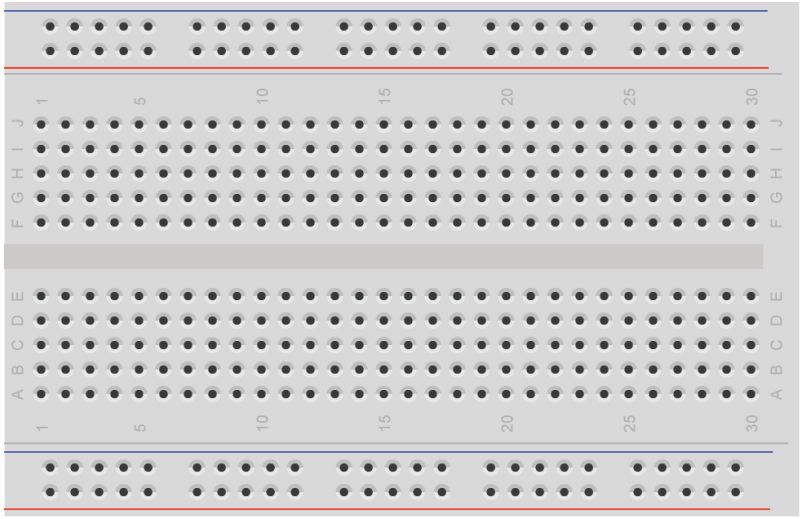
Blinking LED

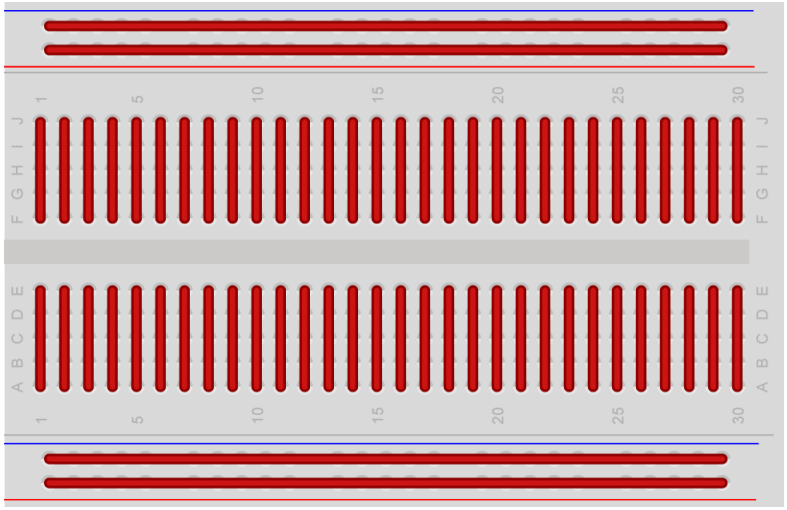
Arduino DUE



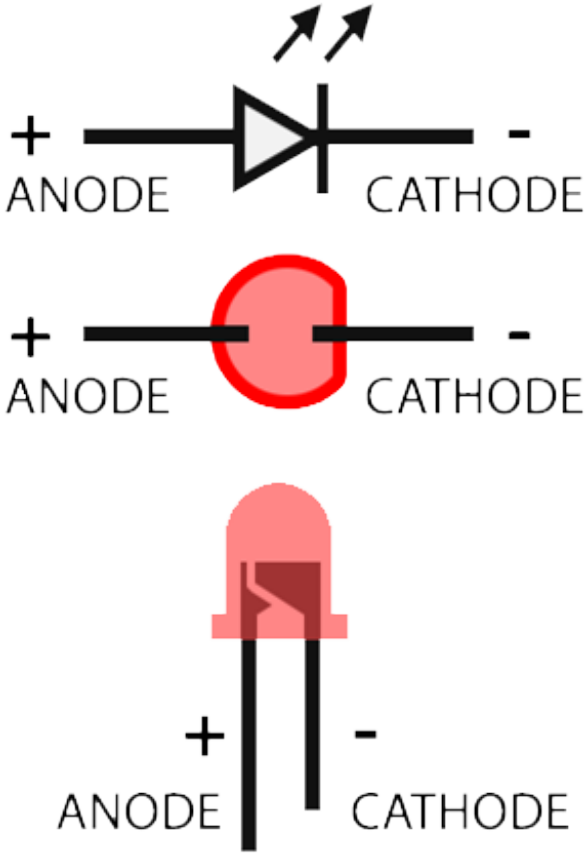
Microcontroller	AT91SAM3X8E
Operating Voltage	3.3V
Input Voltage (recommended)	7-12V
Input Voltage (limits)	6-16V
Digital I/O Pins	54 (of which 12 provide PWM output)
Analog Input Pins	12
Analog Output Pins	2 (DAC)
Total DC Output Current on all I/O lines	130 mA
DC Current for 3.3V Pin	800 mA
DC Current for 5V Pin	800 mA
Flash Memory	512 KB all available for the user applications
SRAM	96 KB (two banks: 64KB and 32KB)
Clock Speed	84 MHz
Length	101.52 mm
Width	53.3 mm
Weight	36 g

Breadboard



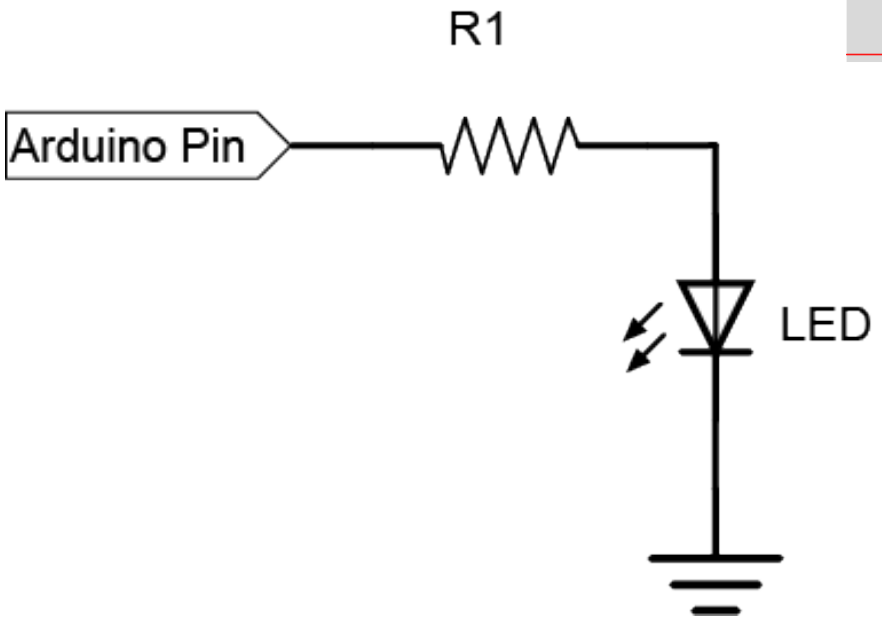


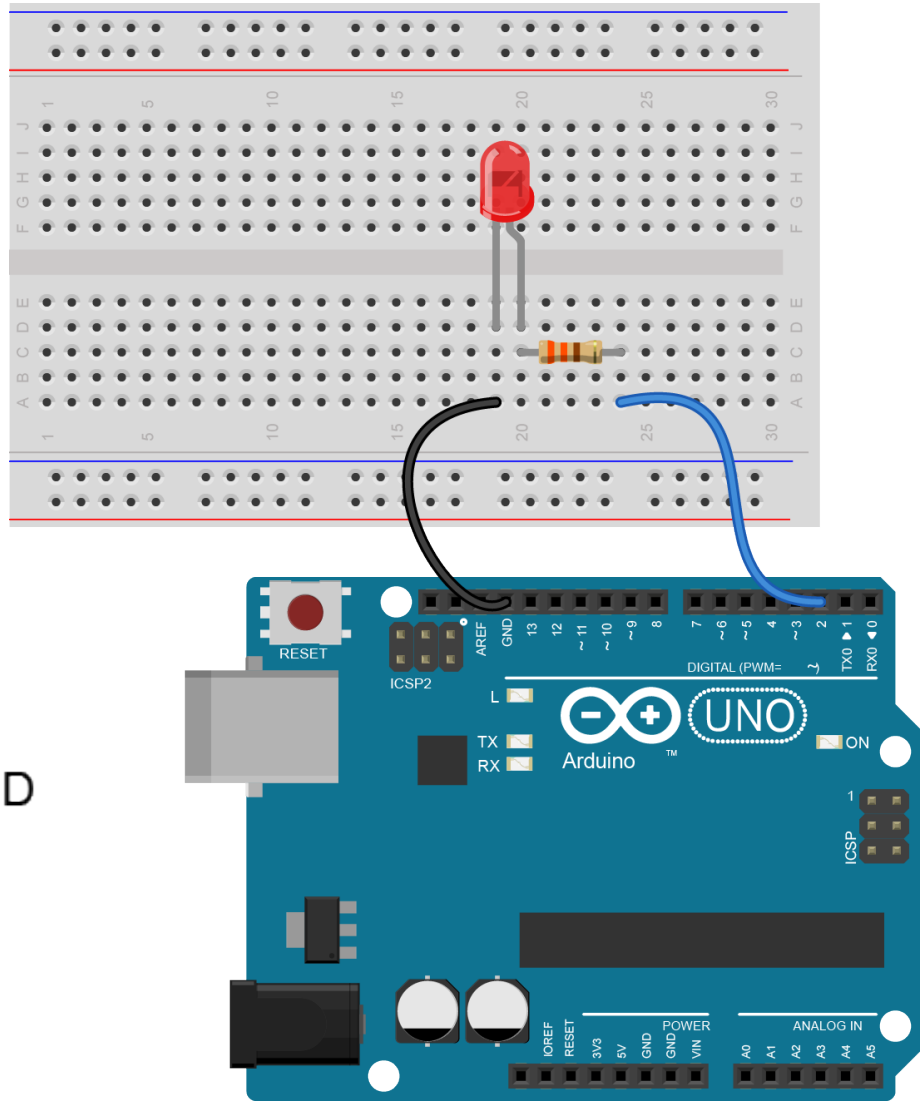
LED



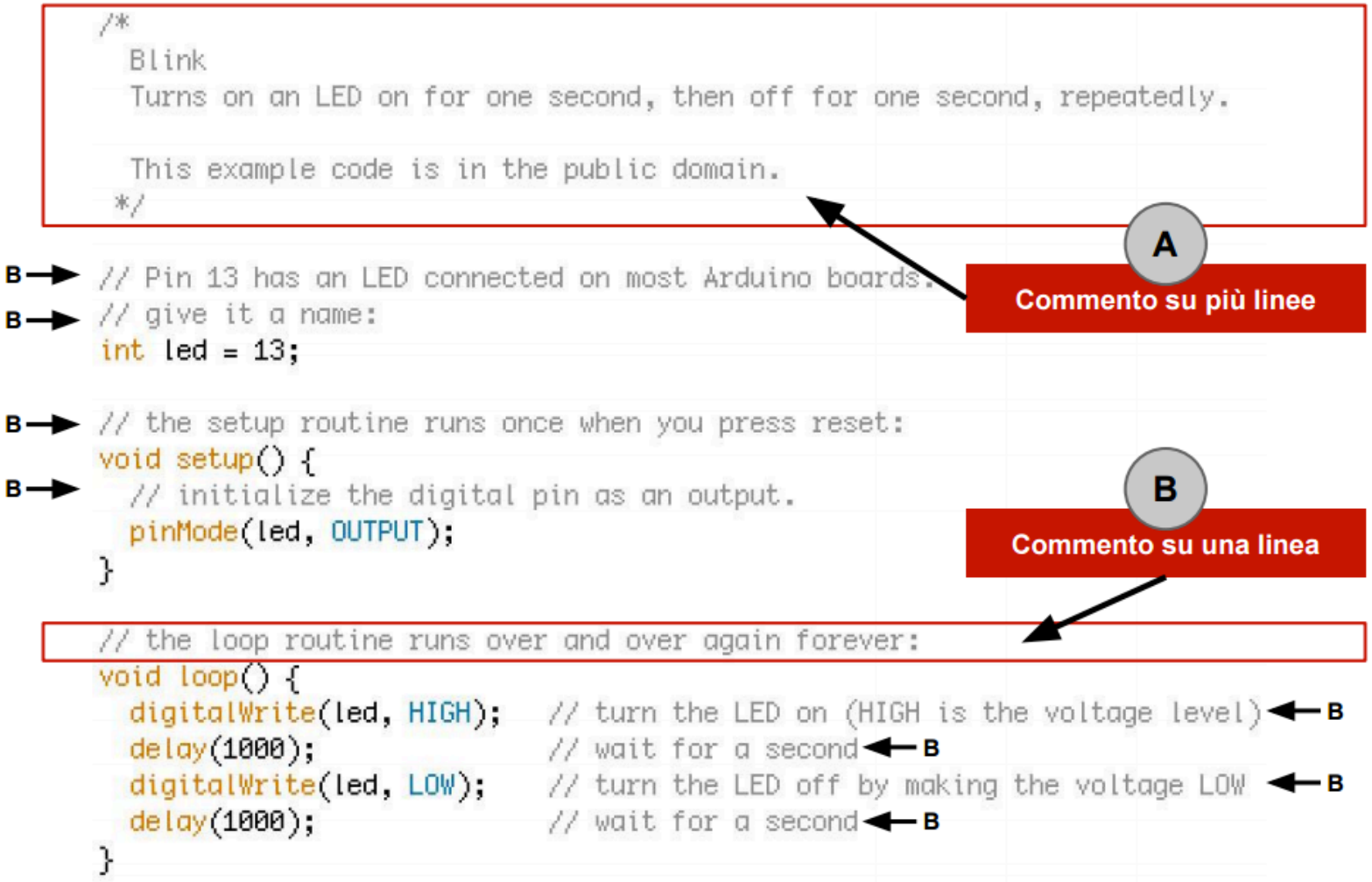
Circuit

- Arduino board
- 1 breadboard
- 1 led
- 1 resistor of 150 ohm
- wires

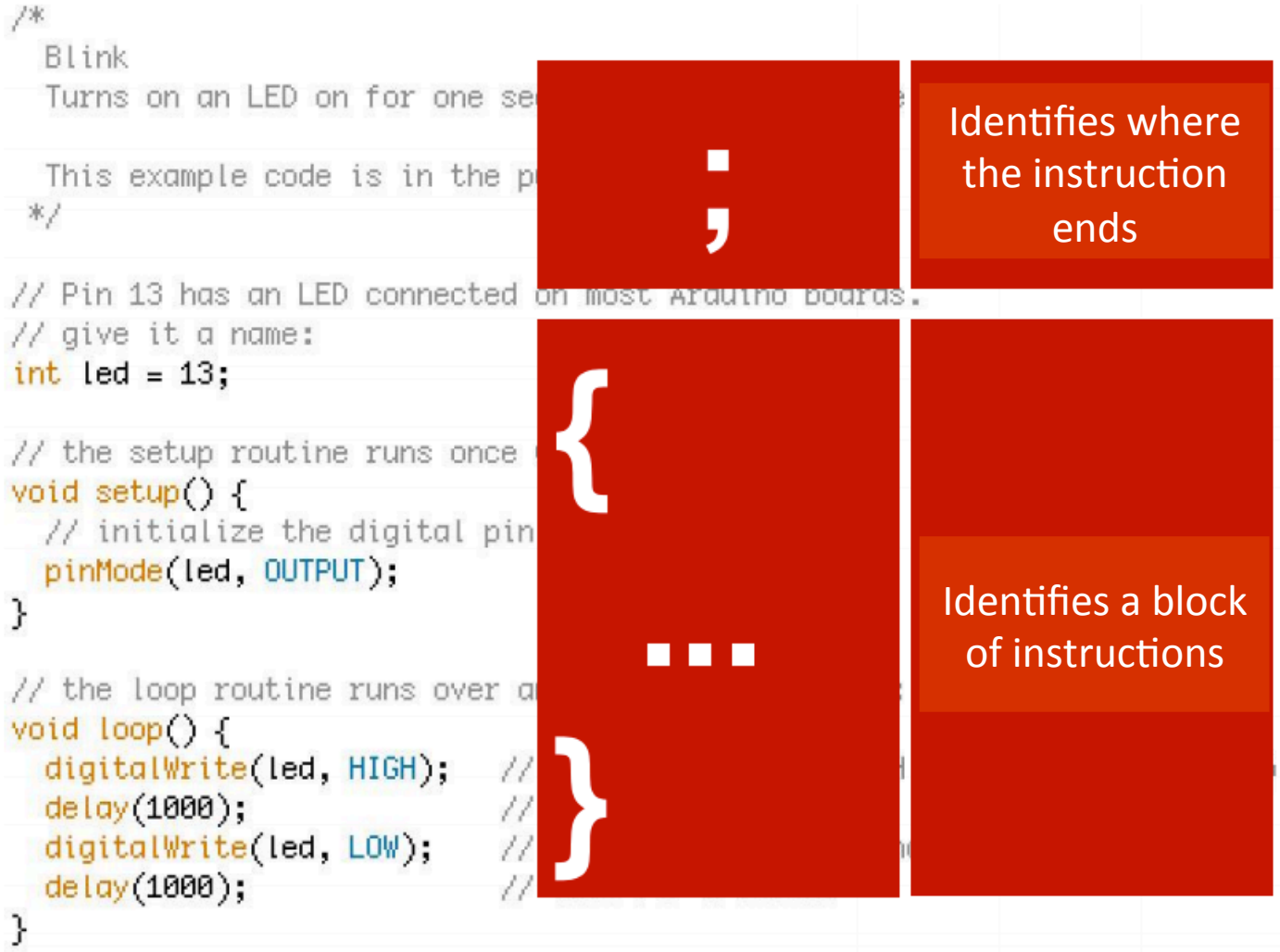




Code analysis 1/7



Code analysis 2/7



Code analysis 3/7

```
/*
  Blink
  Turns on an LED on for one se

  This example code is in the p
 */

// Pin 13 has an LED connected
// give it a nam
int led = 13;

// the setup routine runs once
void setup() {
  // initialize the digital pin
  pinMode(led, OUTPUT);
}

// the loop routine runs over a
void loop() {
  digitalWrite(led, HIGH);   //
  delay(1000);               //
  digitalWrite(led, LOW);    //
  delay(1000);               //
}
```

int led = 13;

A variable is a way for naming and storing a numerical value for later use by the program. All variables must be declared before they can be used. Declaring a variable means:

- define the type of value that can assume: int, long,float, etc …
- assign a name
- and optionally assign an initial value.

These operations are carried out only once in program, but the value of the variable can be changed at any time using the arithmetic or using assignments. The following example stated that LED is an int, (Integer type) and that its initial value is equal to 13. This is called a **simple assignment**.

Code analysis 4/7

```
/*
  Blink
  Turns on an LED on for one second, then off for one s

  This example code is in the public domain.
 */

// Pin 13 has an LED connected on most Arduino boards.
// give it a name:
int led = 13;

// the setup routine runs once when you press reset:
void setup() {
  // initialize the digital pin as an output.
  pinMode(led, OUTPUT);
}

// the loop routine runs over and over again forever:
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
}
```

Struttura di base

```
void setup()
{
  istruzioni;
}

void loop()
{
  istruzioni;
}
```

Code analysis 5/7

```
/*
  Blink
  Turns on an LED on for one seco

  This example code is in the pub
 */

// Pin 13 has an LED connected on
// give it a name:
int led = 13;

// the setup routine runs once wh
void setup() {
  // initialize the digita pin a
  pinMode(led, OUTPUT);
}

// the loop routine runs over and
void loop() {
  digitalWrite(led, HIGH);   // t
  delay(1000);               // w
  digitalWrite(led, LOW);    // t
  delay(1000);               // w
}
```

pinMode (LED, OUTPUT);

pinMode is an instruction that specifyes how a particular pin is defined. In the parentheses, topics that can be numbers and letters are specified. Digital pins can be used as INPUT or OUTPUT. In this case, we want to flash the diode, for this reason LED must be define as OUTPUT pin. The INPUT and OUTPUT words are defined constants

Code analysis 6/7

```
/*
  Blink
  Turns on an LED on for one second,

  This example code is in the public
 */

// Pin 13 has an LED connected on mos
// give it a name:
int led = 13;

// the setup routine runs once when y
void setup() {
  // initialize the digital pin as an
  pinMode(led, OUTPUT);
}

// the loop routine runs over and ove
void loop() {
  digitalWrite(led, HIGH);   // turn
  delay(1000);               // wait
  digitalWrite(led, LOW);    // turn
  delay(1000);               // wait
}
```

digitalWrite (led, HIGH);

The digitalWrite instruction has two arguments: the first one defines the pin, the second one indicates the status.
If the pin is configured as an OUTPUT, digitalWrite() will enable (HIGH) or disable (LOW) the internal pullup on the output pin (it turns on or off an LED). The 'pin' can be specified as a variable or a constant. If the pin state is HIGHT, it means that is applied a voltage of 3,3V (5V for Arduino Uno), while if the state is LOW the applied voltage is 0V.

Code analysis 7/7

```
/*
  Blink
  Turns on an LED on for or

  This example code is in t
 */

// Pin 13 has an LED connec
// give it a name:
int led = 13;

// the setup routine runs c
void setup() {
  // initialize the digital
  pinMode(led, OUTPUT);
}

// the loop routine runs ov
void loop() {
  digitalWrite(led, HIGH);
  delay(1000);
  digitalWrite(led, LOW);
  delay(1000);
}
```

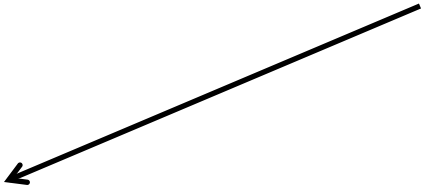
delay (1000);

delay () Pauses the program for the amount of time (in miliseconds) specified as parameter. (There are 1000 milliseconds in a second). The instruction has a single numeric argument; It indicates the number of milliseconds to wait.

Programming our sketch

```
void setup()
{
  Serial.begin(9600);
  pinMode (ledPin,OUTPUT);
}
```

Just like before we need to set our LED as an output.



We have a new piece of code here: ***Serial.begin ( )***

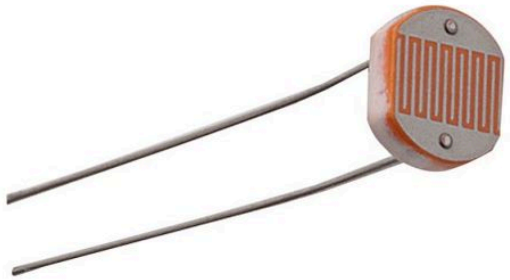
Sets the data rate in bits per second (baud) for serial data transmission. So basically we are going to transfer 9600 bits per second to the computer. Old modems attached to our phone lines used to work at this speed. This was considered FAST back in the day, until 14400 finally came out.

Second Program

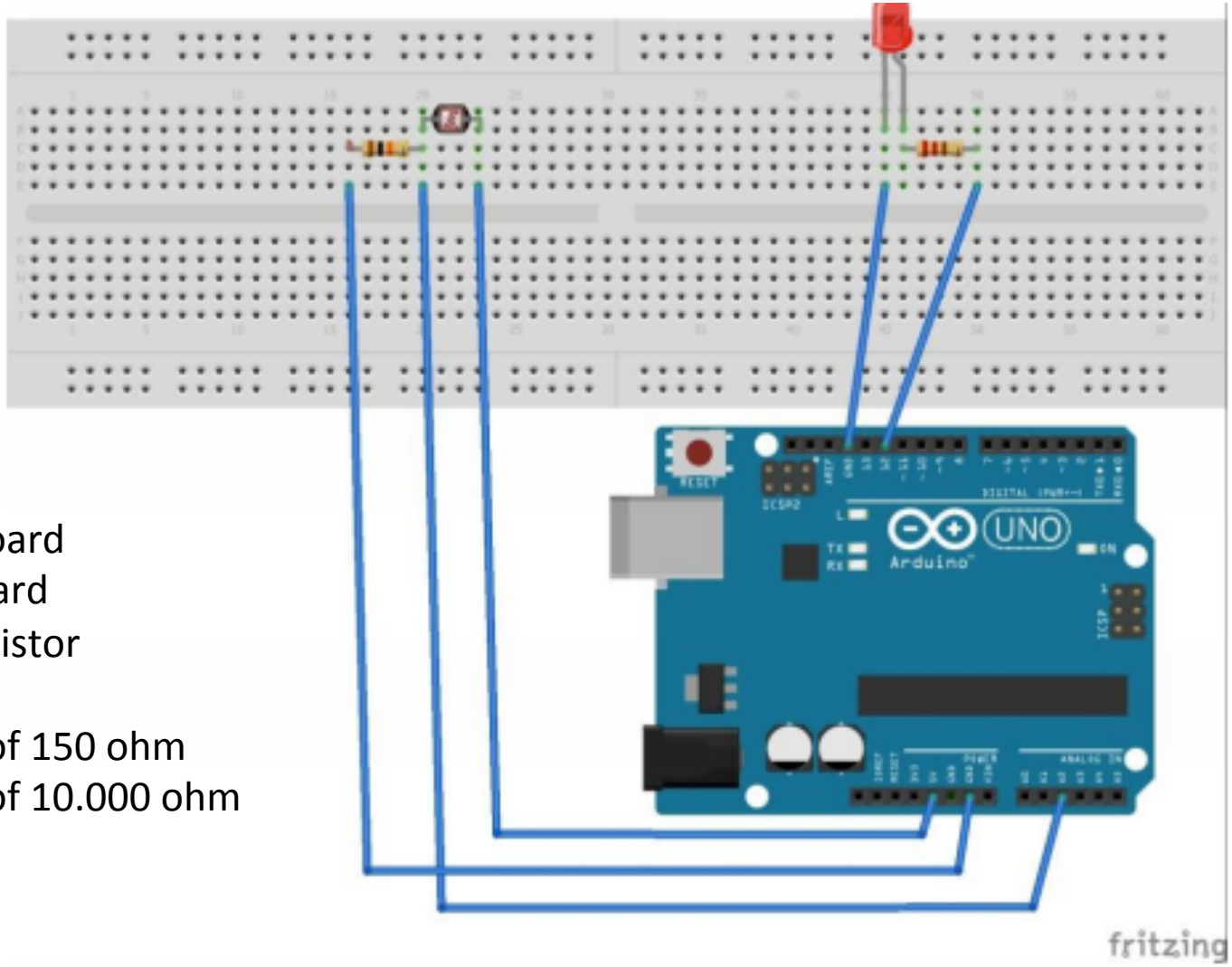
Photoresistor with LED

Photoresistor

A photoresistor and 'a resistance whose impedance (and that means' whose capacity 'to circulate electricity') varies according to the light that strikes it. While the light increase, the resistance decreases, and vice versa. It is typically an analog type sensor. To use in necessary to connect a leg to an analog port and, in parallel, to a 10k ohm resistor connected to ground while connecting the other leg power from 5 volts. The analog port returns a value from 0 to 1023 (or from 0 to 4095 in 12 bits ADC) that varies according to the light that strikes the photoresistor.



Circuit for photoresistor



- Arduino board
- 1 breadboard
- 1 photoresistor
- 1 led
- 1 resistor of 150 ohm
- 1 resistor of 10.000 ohm
- 5 wires

analogReadResolution()

Description

analogReadResolution() is an extension of the Analog API for the Arduino Due and Zero.

Sets the size (in bits) of the value returned by analogRead(). It defaults to 10 bits (returns values between 0-1023) for backward compatibility with AVR based boards.

The Due and the Zero have 12-bit ADC capabilities that can be accessed by changing the resolution to 12. This will return values from analogRead() between 0 and 4095.

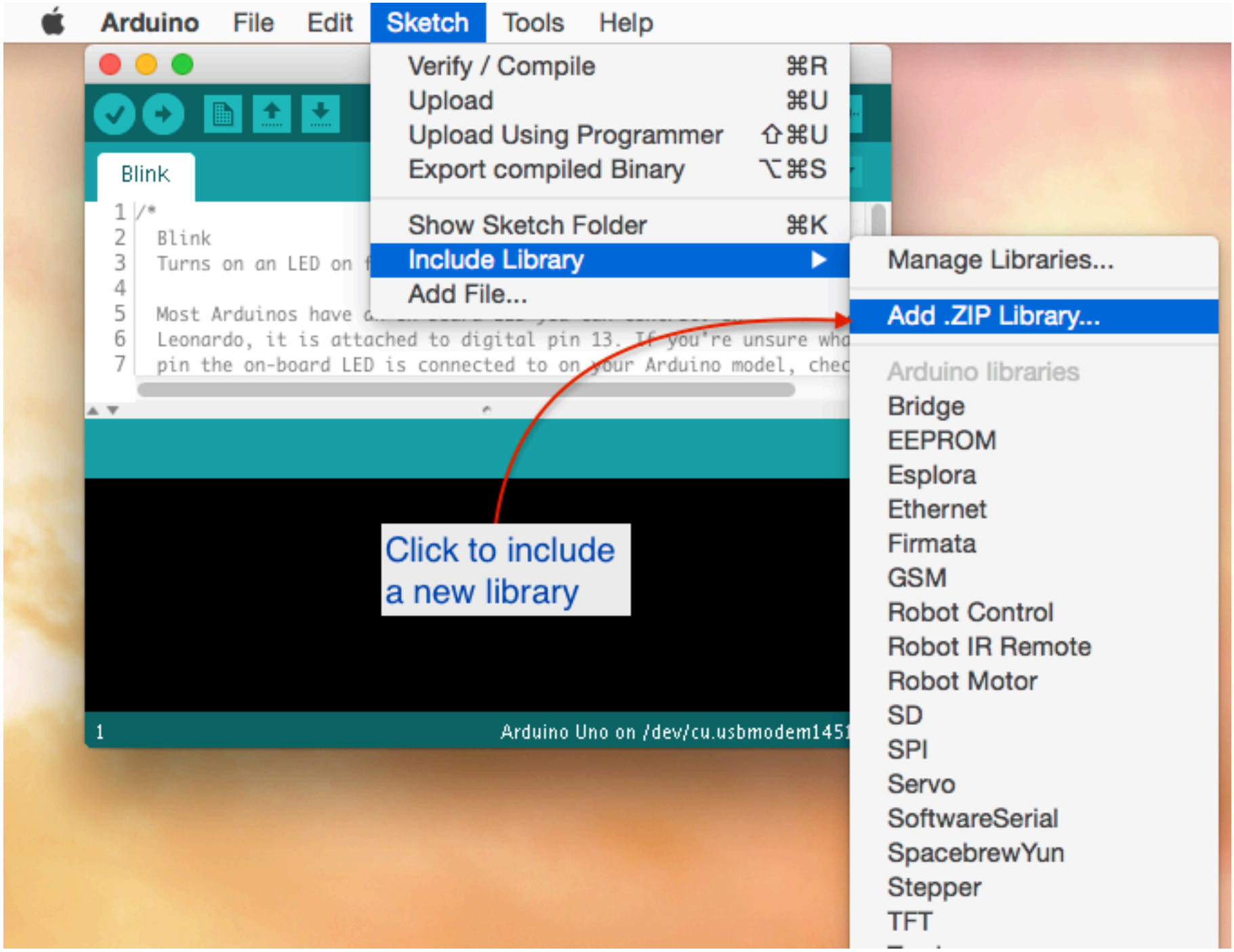
Sketch

```
 */
//
int valorefotocellula;      // variabile in cui viene inserito il valore analogico (da 0 a 1023)
                            //  della tensione rilevata sulla fotocellula.
//
//
void setup()
{
  pinMode(12, OUTPUT);     // definisce la porta 12 come porta di output
}
//
//
void loop()
{
  valorefotocellula = analogRead(2);    // legge il valore fornito dalla fotoresistenza
  if(valorefotocellula<=512)    // 512 e' un valore intermedio (la scala analogica va
                                // da 0 a 1023). Per rendere il sensore piu' o meno sensibile sara'
                                // sufficiente aumentare o diminuire questo parametro.
/* nota: in realta' sulla porta 2 arduino non legge il valore della luce ambientale, ma una
 tensione, che sara' bassa se l'impedenza della fotoresistenza (dipendente dalla luce ambientale)
 sara' alta e viceversa */
 {
    digitalWrite(12, HIGH);     // accende il led se l'impedenza della fotoresistenza (impedenza
        //  proporzionale alla luce rilevata) e' alta e quindi la luce ambientale e' bassa
  }
 else
 {
    digitalWrite(12, LOW);     // in caso contrario lo spegne
  }
}
```

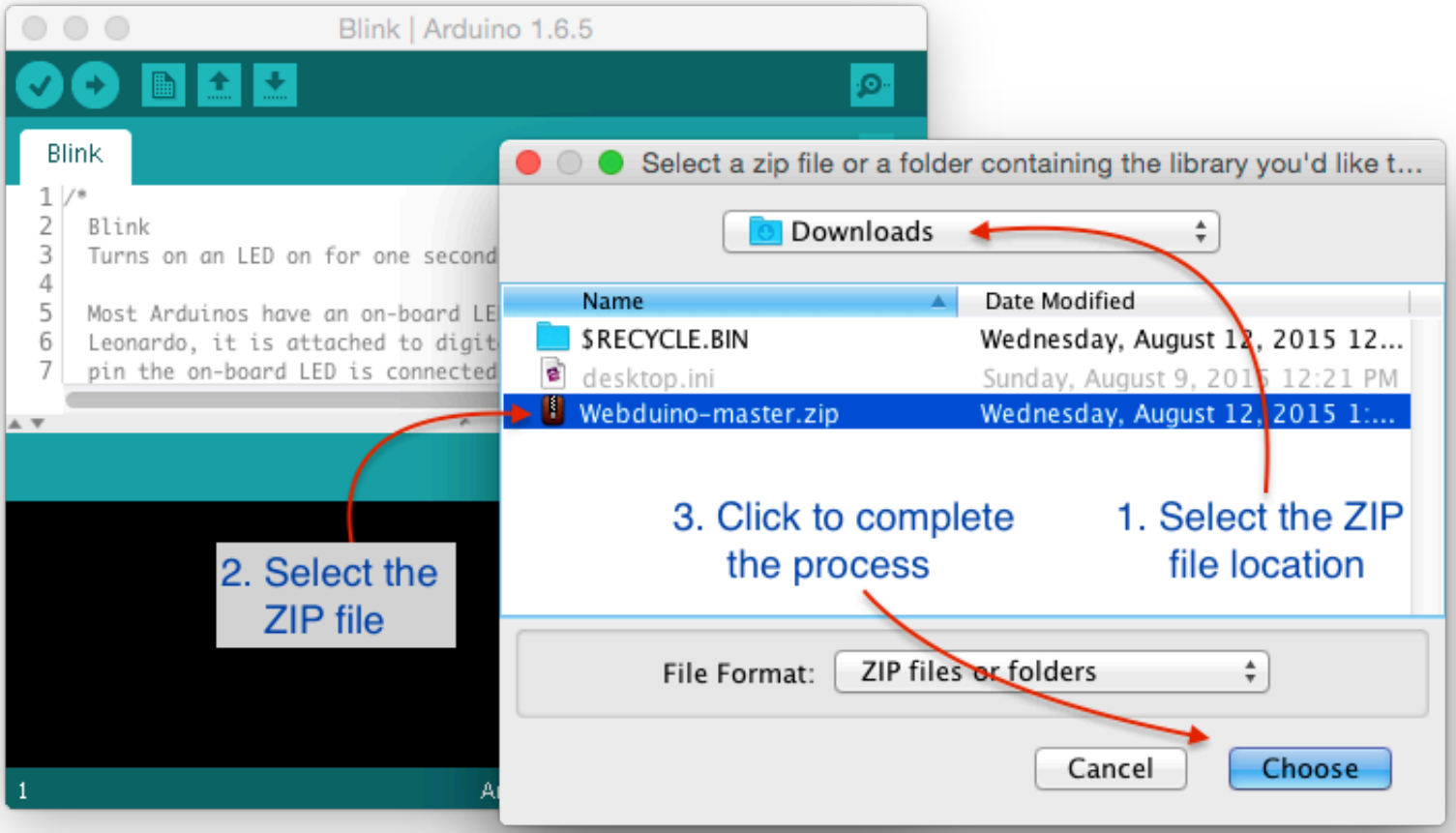
Libraries

Arduino Libraries

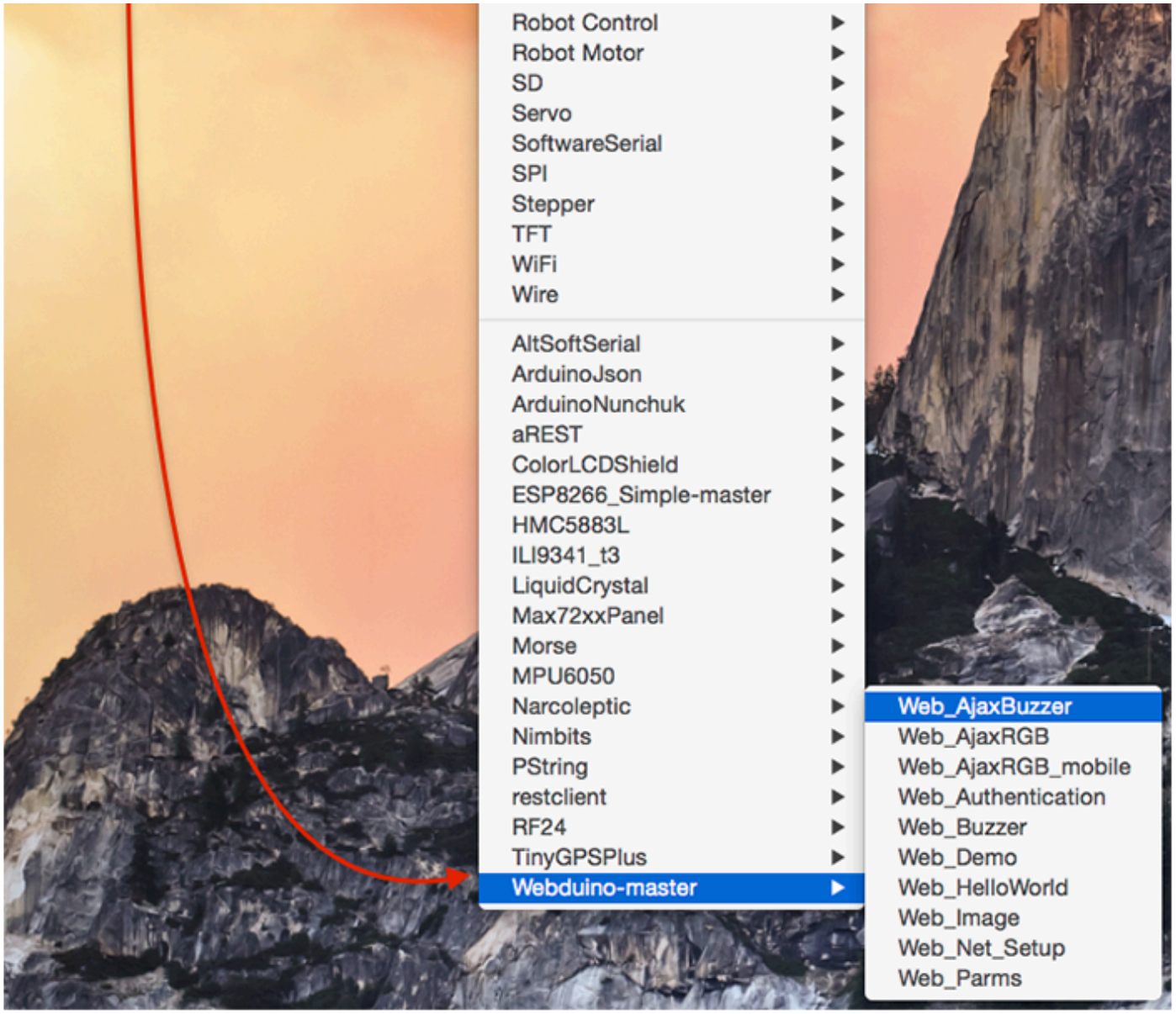
- If there is a library that you need but is not included with the IDE, you can install it. Let's look at an example.
- Download the ZIP file on your computer. It doesn't matter what platform you are on; the libraries work the same regardless of whether you are on Windows, Mac or Linux.
- Also, don't worry about extracting the files from the ZIP archive. The newer versions of the Arduino IDE have an easy library installer that takes care of extracting the library from the ZIP file and copying the files to the right location.
- Assuming the library ZIP file is in your Downloads folder, start the Arduino IDE. Then click on "Sketch → Include Library → Add .ZIP Library…", like this:


Arduino File Edit Sketch Tools Help
Verify / Compile ⌘R
Upload ⌘U
Upload Using Programmer ⇧⌘U
Export compiled Binary ⌥⌘S
Show Sketch Folder ⌘K
Include Library
Add File...
Manage Libraries...
Add .ZIP Library...
Arduino libraries
Bridge
EEPROM
Esplora
Ethernet
Firmata
GSM
Robot Control
Robot IR Remote
Robot Motor
SD
SPI
Servo
SoftwareSerial
SpacebrewYun
Stepper
TFT
Blink
Click to include a new library
Arduino Uno on /dev/cu.usbmodem1451

A new dialogue box will pop up. Browse to the location of the ZIP file, select it, and click on Choose to complete the process:



- When you click on “Choose”, the dialogue box will disappear, but nothing else is going to happen. No confirmation, no sound… To make sure that the Webuino library was actually installed, you can look for the example sketches that most libraries include.
- Go to File → Examples, and look at the bottom of the list for your new library:


Robot Control
Robot Motor
SD
Servo
SoftwareSerial
SPI
Stepper
TFT
WiFi
Wire
AltSoftSerial
ArduinoJson
ArduinoNunchuk
aREST
ColorLCDShield
ESP8266_Simple-master
HMC5883L
ILI9341_t3
LiquidCrystal
Max72xxPanel
Morse
MPU6050
Narcoleptic
Nimbits
PString
restclient
RF24
TinyGPSPlus
Webduino-master
Web_AjaxBuzzer
Web_AjaxRGB
Web_AjaxRGB_mobile
Web_Authentication
Web_Buzzer
Web_Demo
Web_HelloWorld
Web_Image
Web_Net_Setup
Web_Parms