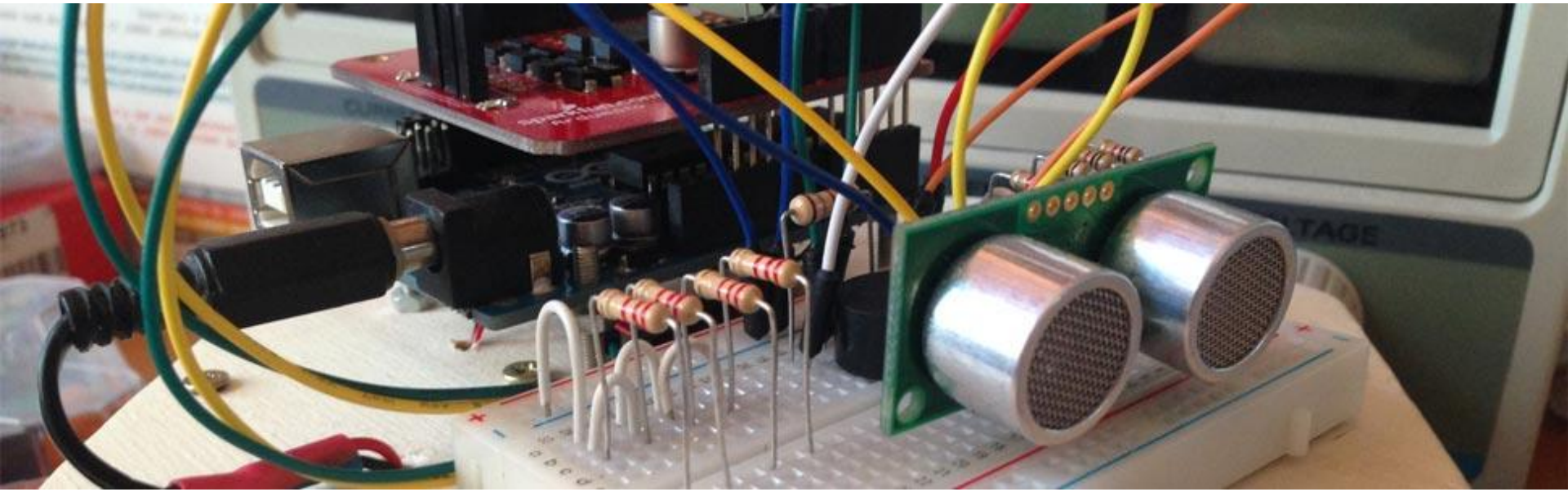
L'alfabeto di Arduino

Introduzione all'uso di Arduino
lezione 2

Prof. Michele Maffucci



Struttura della lezione

Argomenti

- Cos’é Arduino
- La scheda Arduino
- Terminologia essenziale
- Il software Arduino
- Comunicare con Arduino
- Programmazione
- Il primo programma
- Uscite digitali
- Ingressi digitali
- Modulazione di larghezza di impulso (PWM)

Il codice e le slide utilizzate sono suscettibili di variazioni/correzioni che potranno essere fatte in ogni momento.

Introduzione

Il seguente corso intende fornire le **competenze di base** per la realizzazione di lezioni di didattica delle robotica nella scuola secondaria di secondo grado.

Il corso ben si adatta a tutti i maker, studenti ed adulti, che per passione nell'elettronica necessitano di un'introduzione all'uso di Arduino.

Il docente che intendesse sviluppare un percorso didattico in cui si desidera realizzare dispositivi elettronici in grado di interfacciarsi col mondo fisico, potrà utilizzare queste lezioni come base per implementare moduli didattici aggiuntivi, pertanto questo corso è da intendersi come il mio personale tentativo di strutturare un percorso iniziale e modellabile a seconda del tipo di indirizzo della scuola. Chi vorrà potrà effettuare miglioramenti su quanto da me scritto.

Il percorso scelto è un estratto delle lezioni svolte durante i miei corsi di elettronica, sistemi ed impianti elettrici. Nelle slide vi sono cenni teorici di elettrotecnica che non sostituiscono in alcun modo il libro di testo, ma vogliono essere un primo passo per condurre il lettore ad un approfondimento su testi specializzati.

Il corso è basato sulla piattaforma Open Source e Open Hardware **Arduino** e fa uso dell'**Arduino starter kit**. Questa scelta non implica l'adozione di queste slide in corsi che non fanno uso di questo kit, ma è semplicemente una scelta organizzativa per lo svolgimento di questo corso di formazione. Alle proposte incluse nel kit ho aggiunto ulteriori sperimentazioni. Tutti i componenti possono essere acquistati separatamente.

Ulteriori approfondimenti e risorse a questo corso possono essere trovate sul mio sito personale al seguente link:

http://www.maffucci.it/area-studenti/arduino/

Nella sezione dedicata ad Arduino, sul mio sito personale, oltre ad ulteriori lezioni, di cui queste slide ne sono una sintesi, è possibile consultare un manuale di programmazione, in cui vengono dettagliate le istruzioni. Per rendere pratico l'utilizzo del manuale ne è stata realizzata anche una versione portable per dispositivi mobili **iOS** e **Android**, maggiori informazioni possono essere trovate seguendo il link.

Esempi utilizzati nel corso.

Tutti i programmi utilizzati nel corso possono essere prelevati al seguente link:

https://github.com/maffucci/LezioniArduino/tree/master/corso01

Gli sketch Arduino sono da scompattare nella cartella sketchbook.

Questo corso è nato in brevissimo tempo (circa 15 giorni) e quindi possibile che siano presenti delle imperfezioni, ringrazio fin d’ora chi vorrà segnalarmi correzioni e miglioramenti.

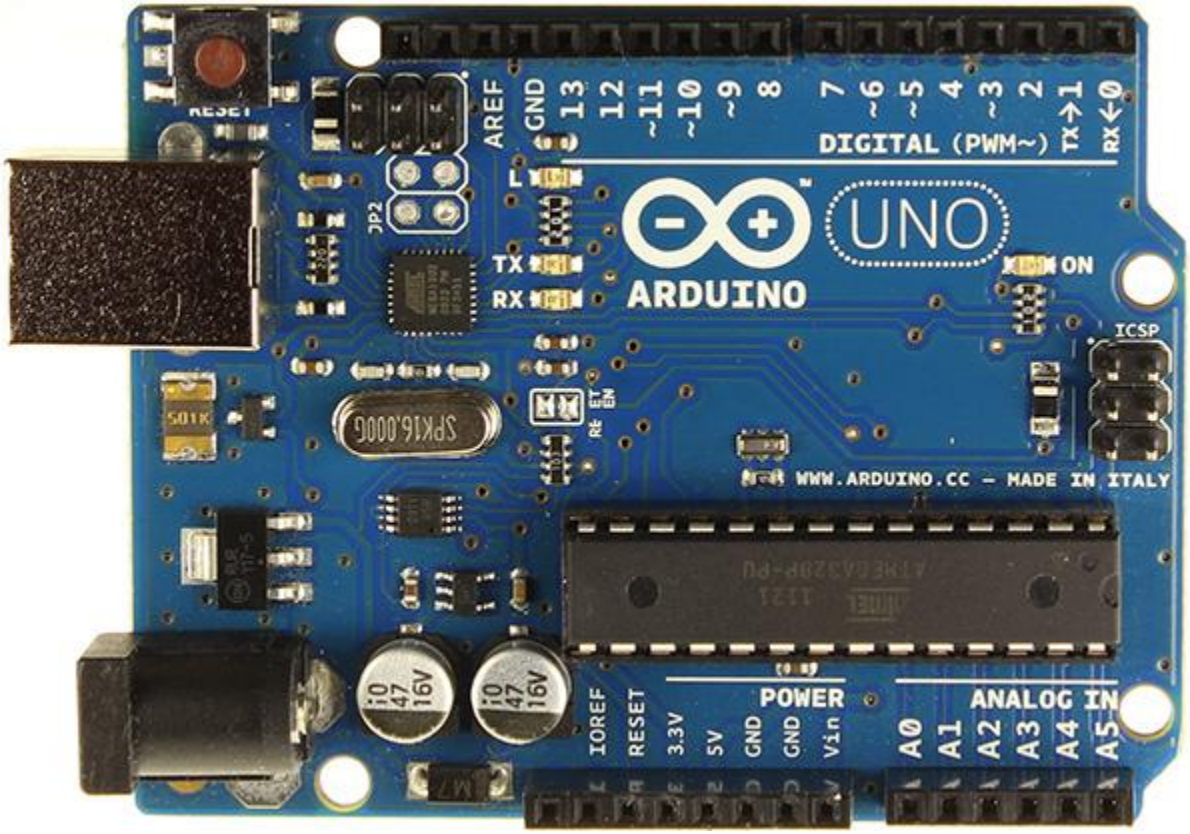
Per contatti ed ulteriori informazioni rimando alle ultime pagine di queste slide.

Grazie

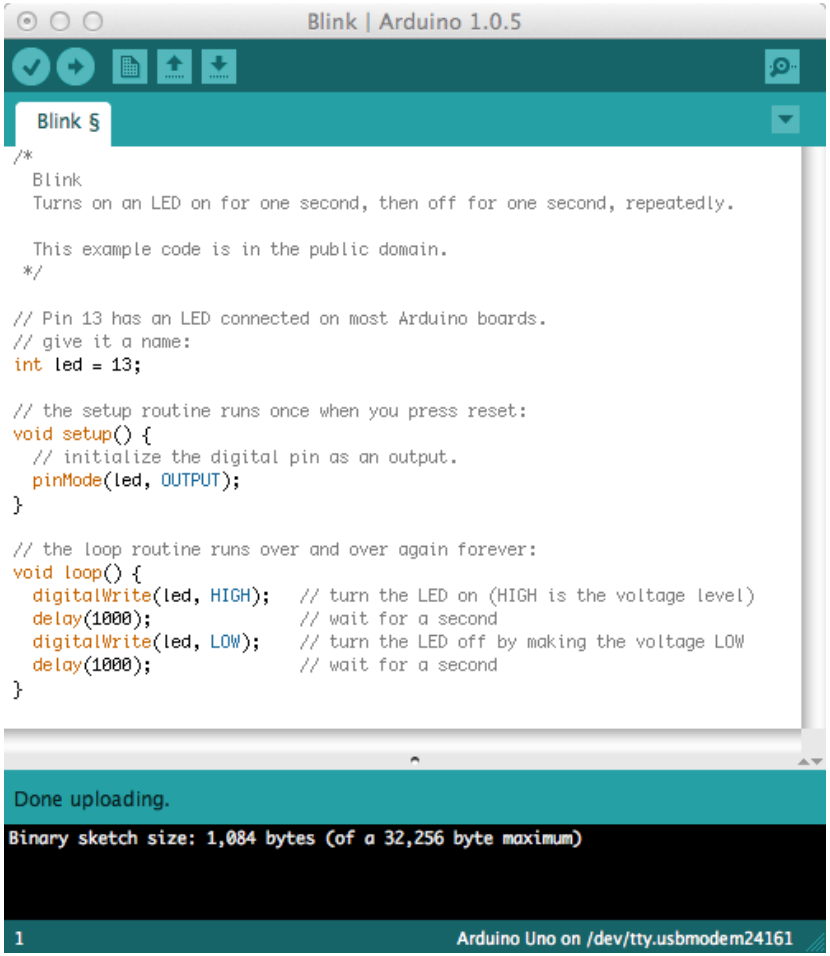
Cos’è Arduino

Arduino vuol dire 3 cose

Un oggetto fisico

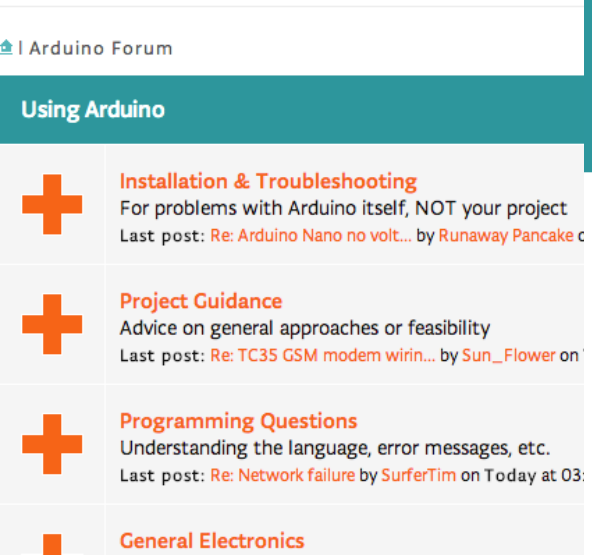


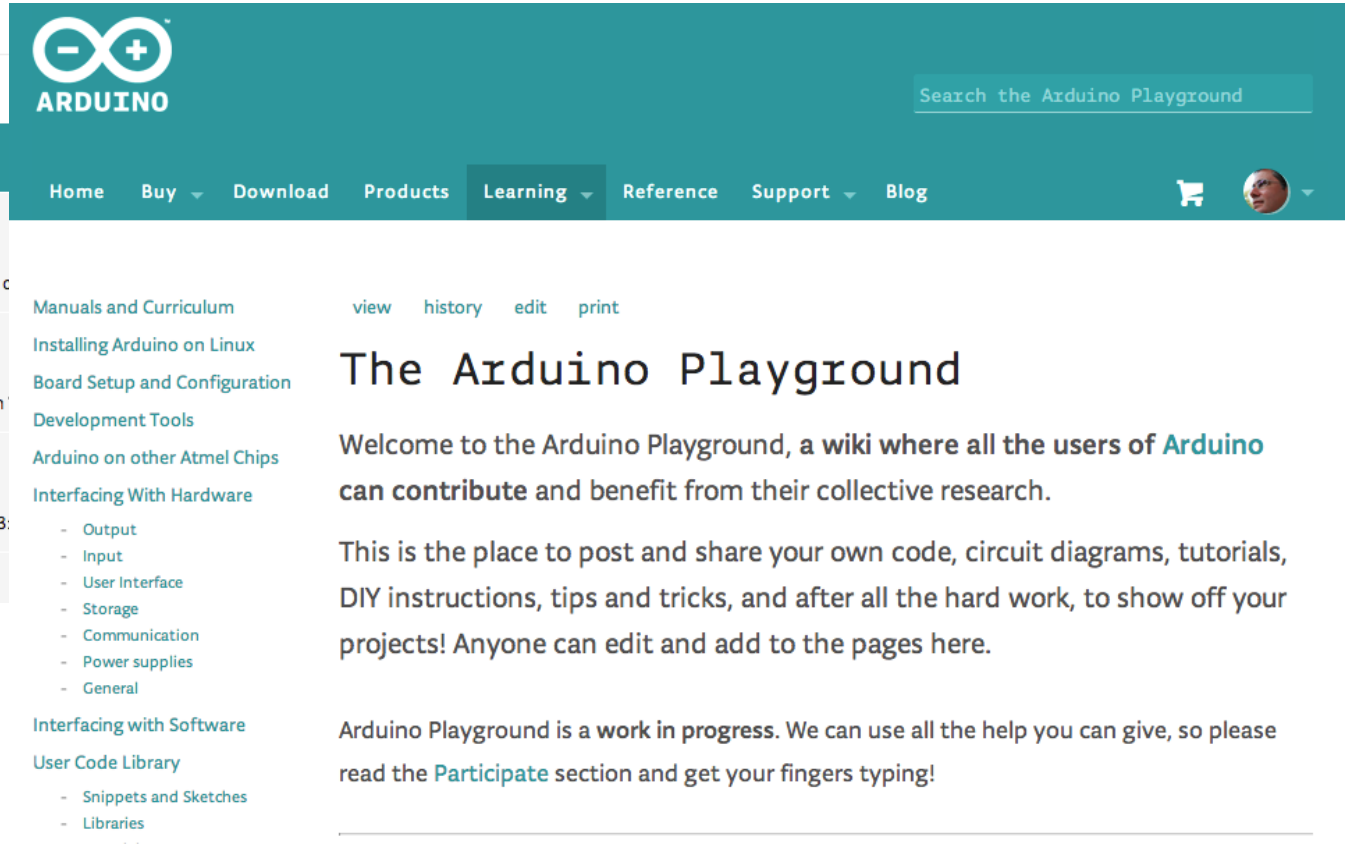
Un ambiente di sviluppo (di programmazione)



Arduino vuol dire 3 cose 3/3

una comunità ed una filosofia di sviluppo

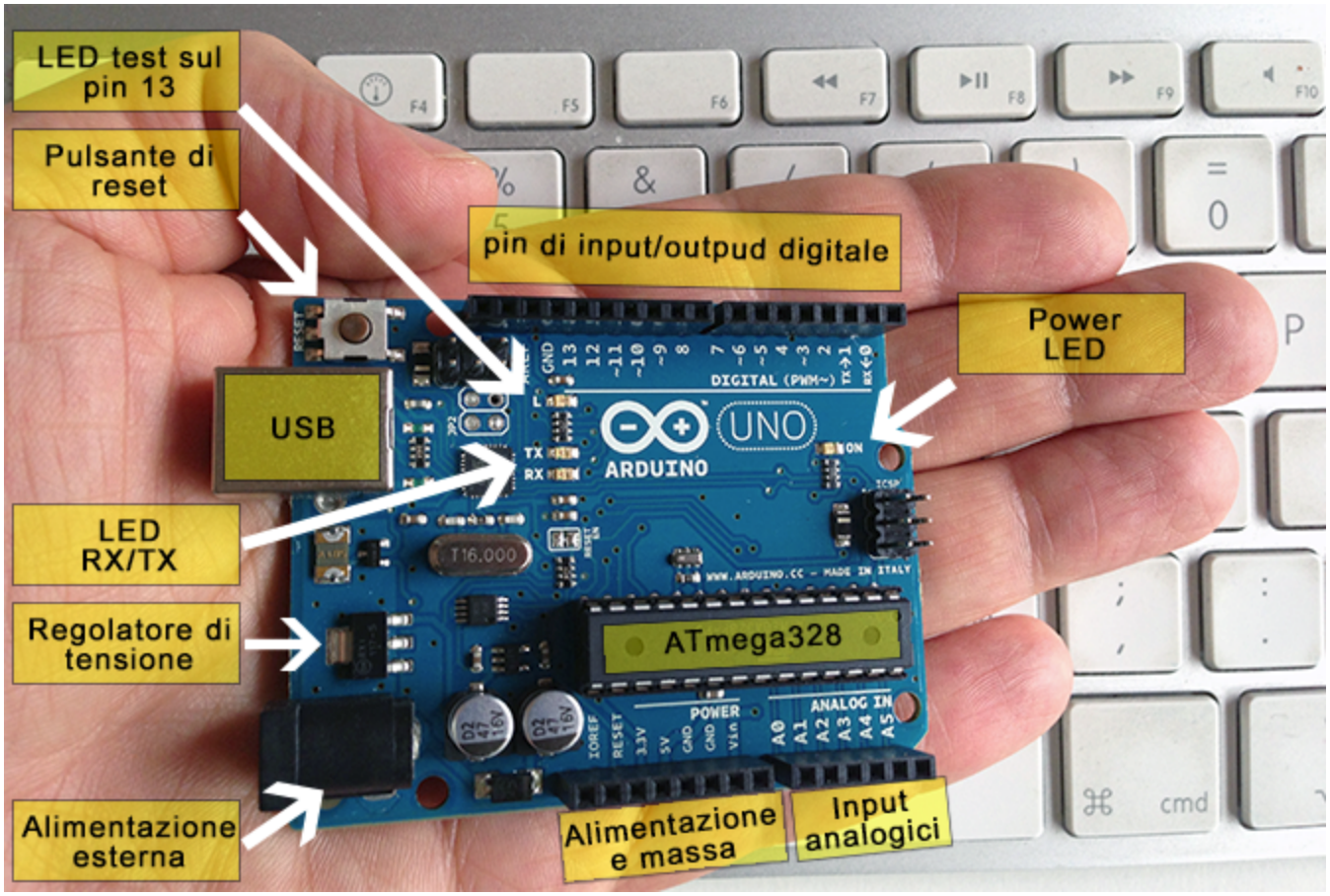




La scheda Arduino

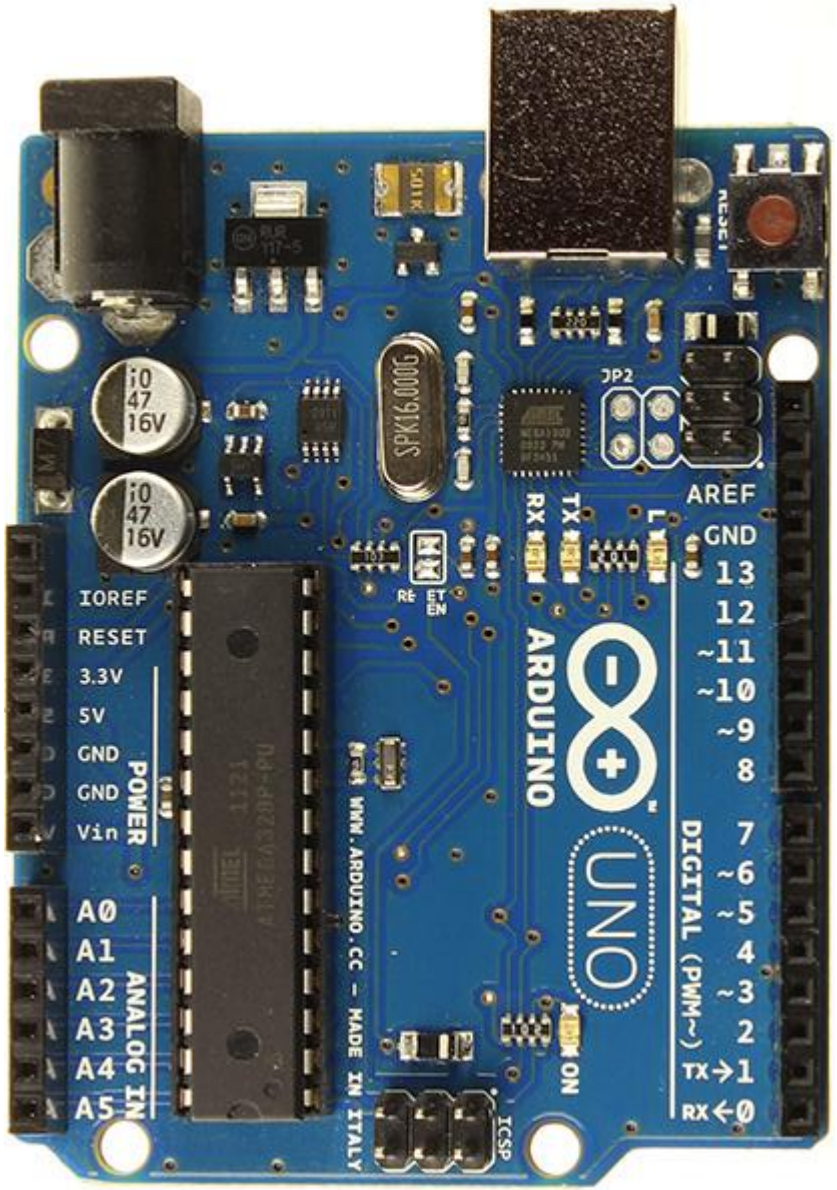
Prof. Michele Maffucci

Elementi di base



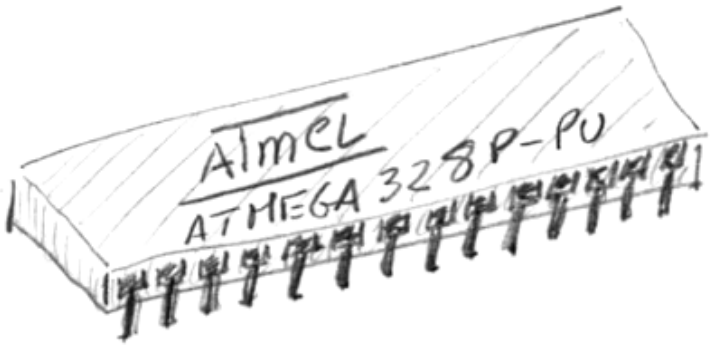
Caratteristiche tecniche

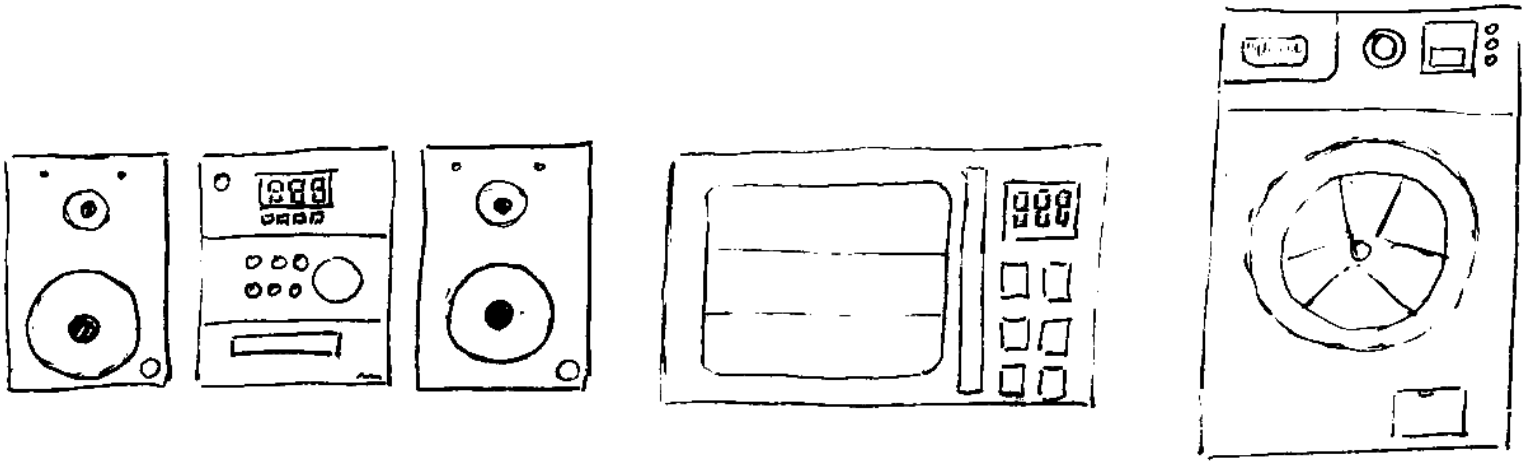
- Microcontroller: ATmega328
- Tensione di lavoro: 5V
- Tensione di ingresso (raccomandata): 7-12V
- Tensione di ingresso (limiti): 6-20V
- Pin digitalio I/O: 14 (di cui 6 forniscono un'uscita PWM)
- Pin analogici: 6
- Corrente Continua per i pin I/O: 40 mA
- Corrente continua per l'uscita a 3.3V: 50 mA
- Flash Memory: 32 KB (ATmega328) di cui 0.5 KB usata per bootloader
- SRAM: 2 KB (ATmega328)
- EEPROM: 1 KB (ATmega328)
- Velocità del clock: 16 MHz



Il microcontrollore

Il cuore della scheda Arduino è il **microcontrollore**, un dispositivo elettronico molto simile ad un computer in miniatura che potete trovare in molti degli elettrodomestici che usate ogni giorno: lavatrice, cellulare, forno a microonde, impianto HiFi, ecc...





E' molto probabile che se l'elettrodomestico possiede pulsanti e display e rileva grandezze fisiche (temperatura, pressione, ecc...) abbia al suo interno un microcontrollore.

Esercizio

Provate a fare un elenco,

contate quanti dispositivi con microcontrollore usate in una giornata tipo.

Terminologia essenziale

Terminologia essenziale

sketch

il programma che scrivete e fate girare sulla scheda Arduino

pin

i connettori di input o output

digital

vuol dire che può assumere solo due valori: ALTO o BASSO, in altro modo: ON/OFF oppure 0 o 1. Sequenza di numeri presi da un insieme discreto di valori (nel nostro caso 0 o 1)

analog

quando i valori utili che rappresentano un segnale sono continui (infiniti)

Il software Arduino

Prof. Michele Maffucci

Il software

L'ambiente di sviluppo viene comunemente chiamato Arduino e ciò può trarre in confusione, perché si identifica con Arduino anche la scheda hardware.

In queste lezioni per indicare l'ambiente di sviluppo software useremo le parole:

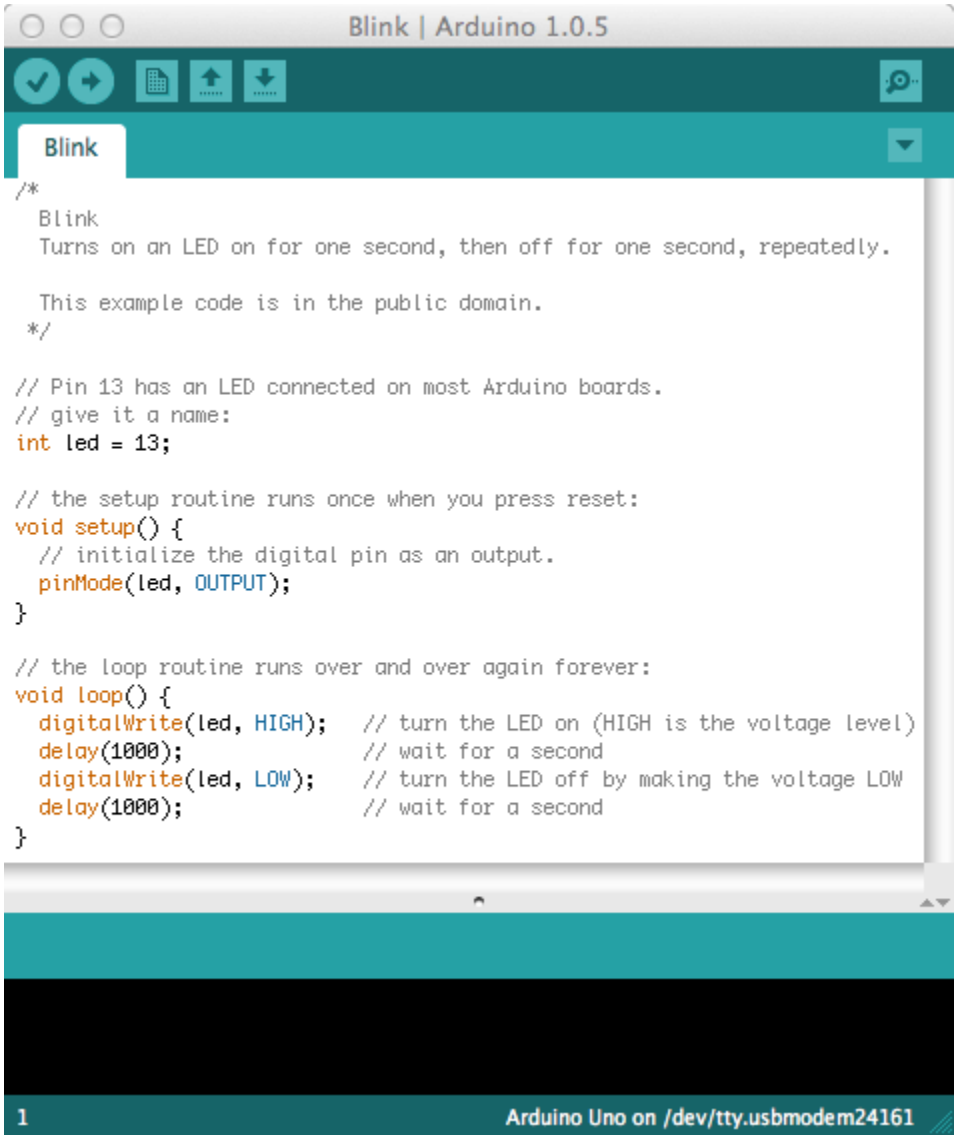
software Arduino

o con stesso significato

IDE

dove l'acronimo **IDE** indica: ***Integrated Development Enviroment***,
in italiano: ***ambiente di sviluppo integrato per la realizzazione di programmi***.

Il software

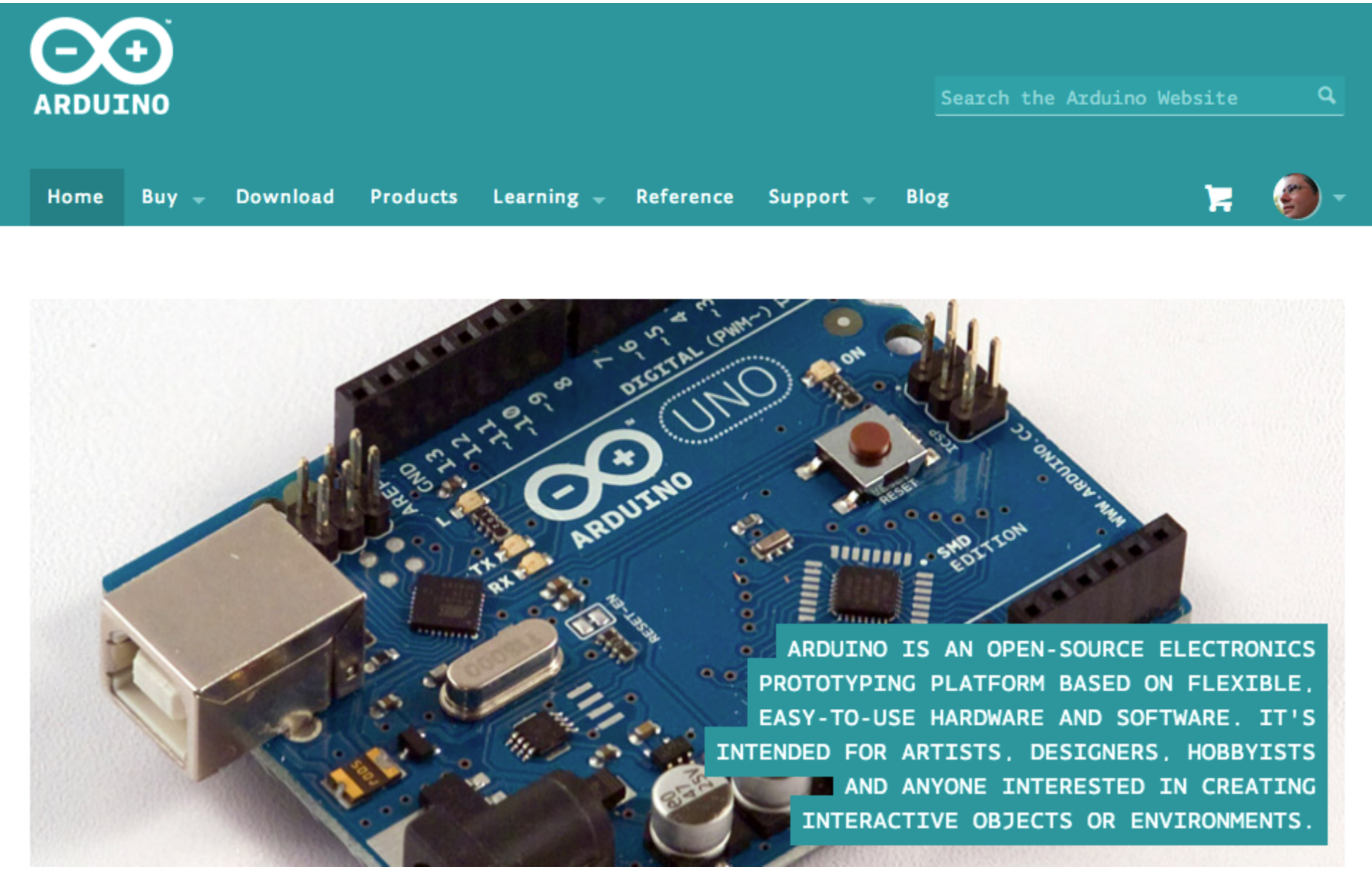


- Simile ad un editor di testo;
- potete scrivere, visualizzare, verificare la sintassi;
- potete trasferire il vostro sketch sulla scheda.

1. prelevare il software Arduino dal sito arduino.cc
2. collegate la scheda Arduino al computer
3. installare i driver
4. riavviate il computer
5. avviare il software Arduino
6. scrivere uno sketch
7. eseguire lo sketch facendo l'upload sulla scheda Arduino

Il software

collegarsi al sito arduino.cc



Il software | Download

ARDUINO

Search the Arduino Website

Home | Buy | Download | Products | Learning | Reference | Support | Blog

Download the Arduino Software

The open-source Arduino environment makes it easy to write code and upload it to the i/o board. It runs on Windows, Mac OS X, and Linux. The environment is written in Java and based on Processing, avr-gcc, and other open source software.

THE ARDUINO SOFTWARE IS PROVIDED TO YOU "AS IS," AND WE MAKE NO EXPRESS OR IMPLIED WARRANTIES WHATSOEVER WITH RESPECT TO ITS FUNCTIONALITY, OPERABILITY, OR USE, INCLUDING, WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR INFRINGEMENT. WE EXPRESSLY DISCLAIM ANY LIABILITY WHATSOEVER FOR ANY DIRECT, INDIRECT, CONSEQUENTIAL, INCIDENTAL OR SPECIAL DAMAGES, INCLUDING, WITHOUT LIMITATION, LOST REVENUES, LOST PROFITS, LOSSES RESULTING FROM BUSINESS

Arduino IDE

Arduino 1.0.5

Download

Arduino 1.0.5 (release notes), hosted by Google Code:

NOTICE: Arduino Drivers have been updated to add support for Windows 8.1, you can download the updated IDE (version 1.0.5-r2 for Windows) from the download links below.

- Windows Installer, Windows (ZIP file)
- Mac OS X
- Linux: 32 bit, 64 bit
- source

Next steps

Getting Started

Reference

Environment

Examples

Foundations

FAQ

Arduino 1.5.6-r2 BETA (with support for Arduino Yún and Arduino Due boards)

Windows

arduino.cc/windows
installazione per: Windows 7, Vista, e XP

Mac OS X

arduino.cc/mac
installazione per: OS X 10.5 e successive

Linux

arduino.cc/linux
installazione per: disponibile per moltissime distribuzioni Linux

approfondimenti su installazione **Mac** e **Linux** su:
www.maffucci.it/area-studenti/arduino/

Comunicare con Arduino

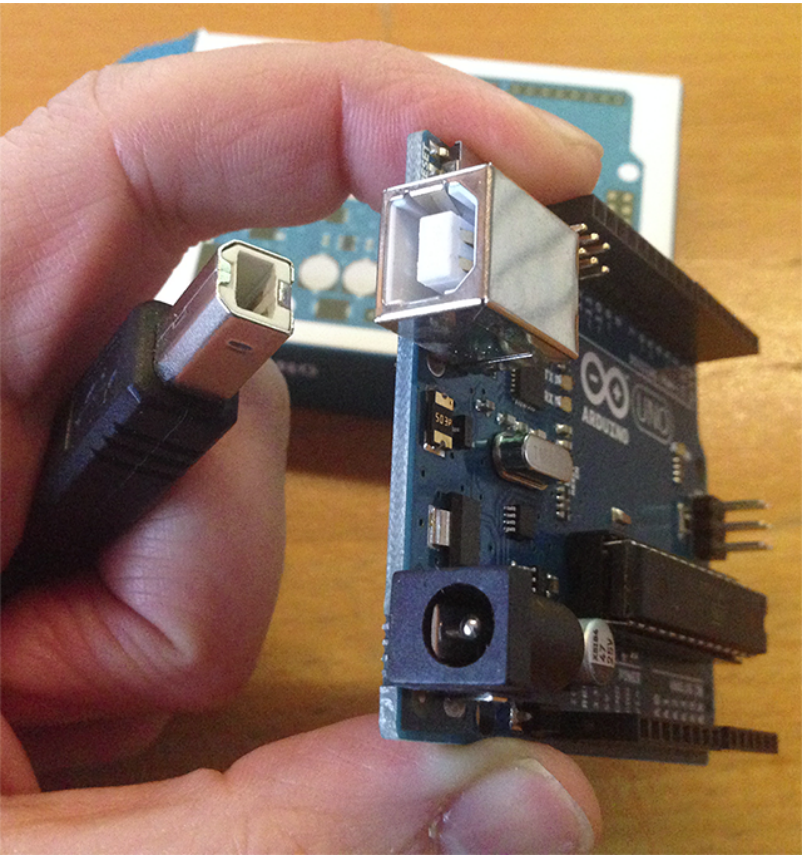
Prof. Michele Maffucci

Avviare l'IDE

Avviate l'IDE di programmazione facendo doppio click sull'icona di Arduino

Collegare la scheda

Collegare la scheda Arduino al computer mediante cavo USB (tipo B)

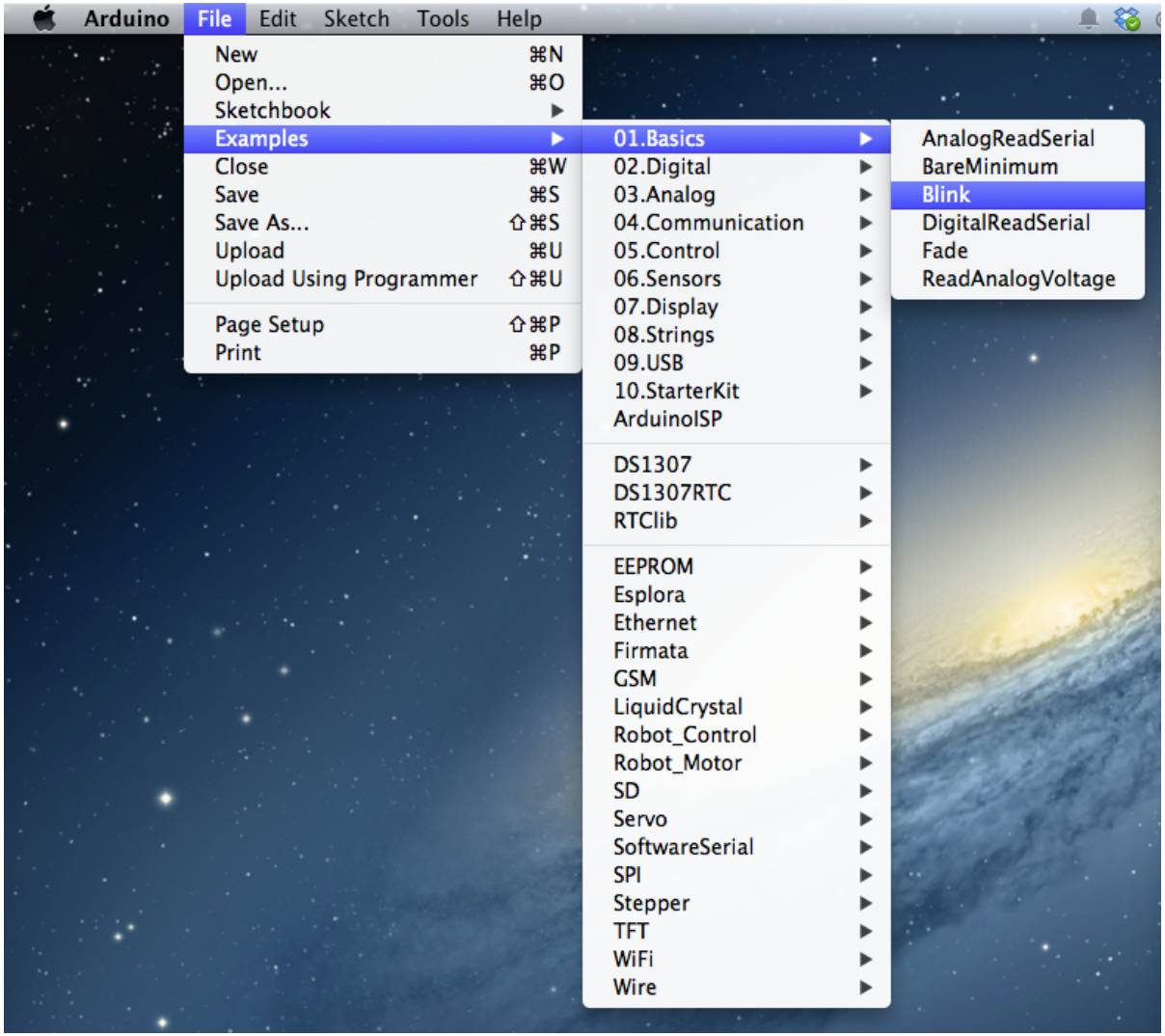


aprire sketch blink

3/8

A

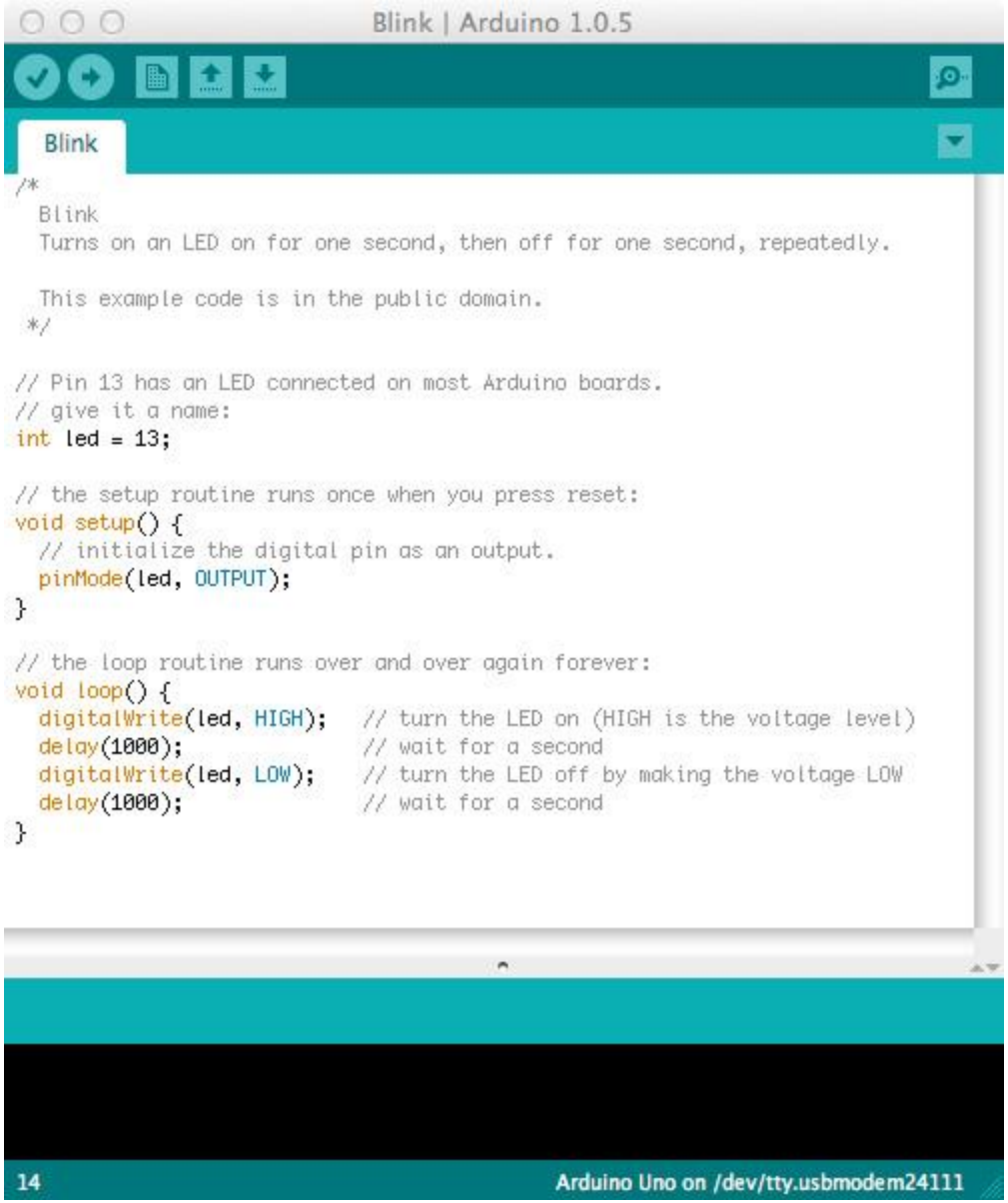
Aprire lo sketch di esempio **blink** che fa lampeggiare il LED presente sulla scheda. Lo sketch può essere aperto da: **File** > **Examples** > **01. Basics** > **Blink**



aprire sketch blink 3/8 B

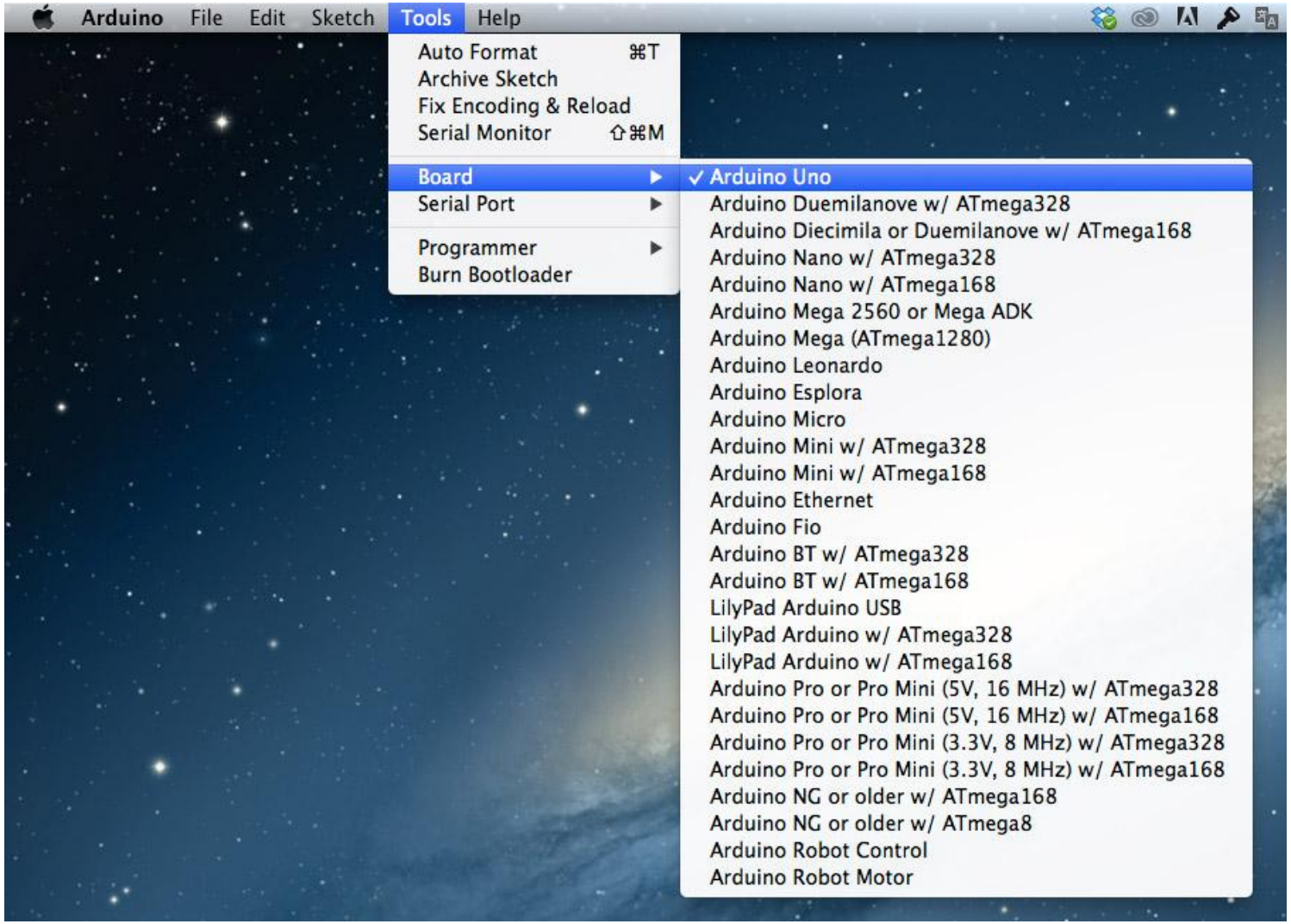
Si aprirà una finestra con il codice del programma blink.

Studieremo più avanti il funzionamento.

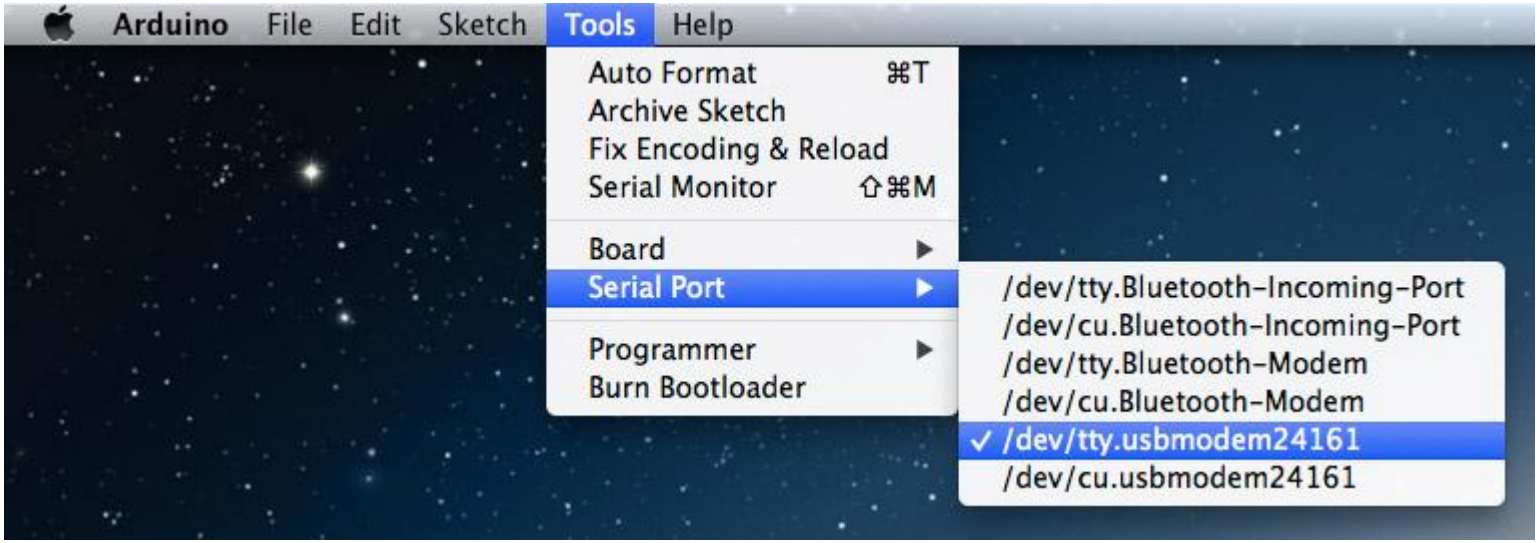


aprire sketch blink

Selezinate la scheda in vostro possesso, nel nostro caso Arduino Uno: **Tools > Board > Arduino Uno**



Selezionate la porta seriale da utilizzare per la comunicazione tra computer ed Arduino: **Tools > Serial port**



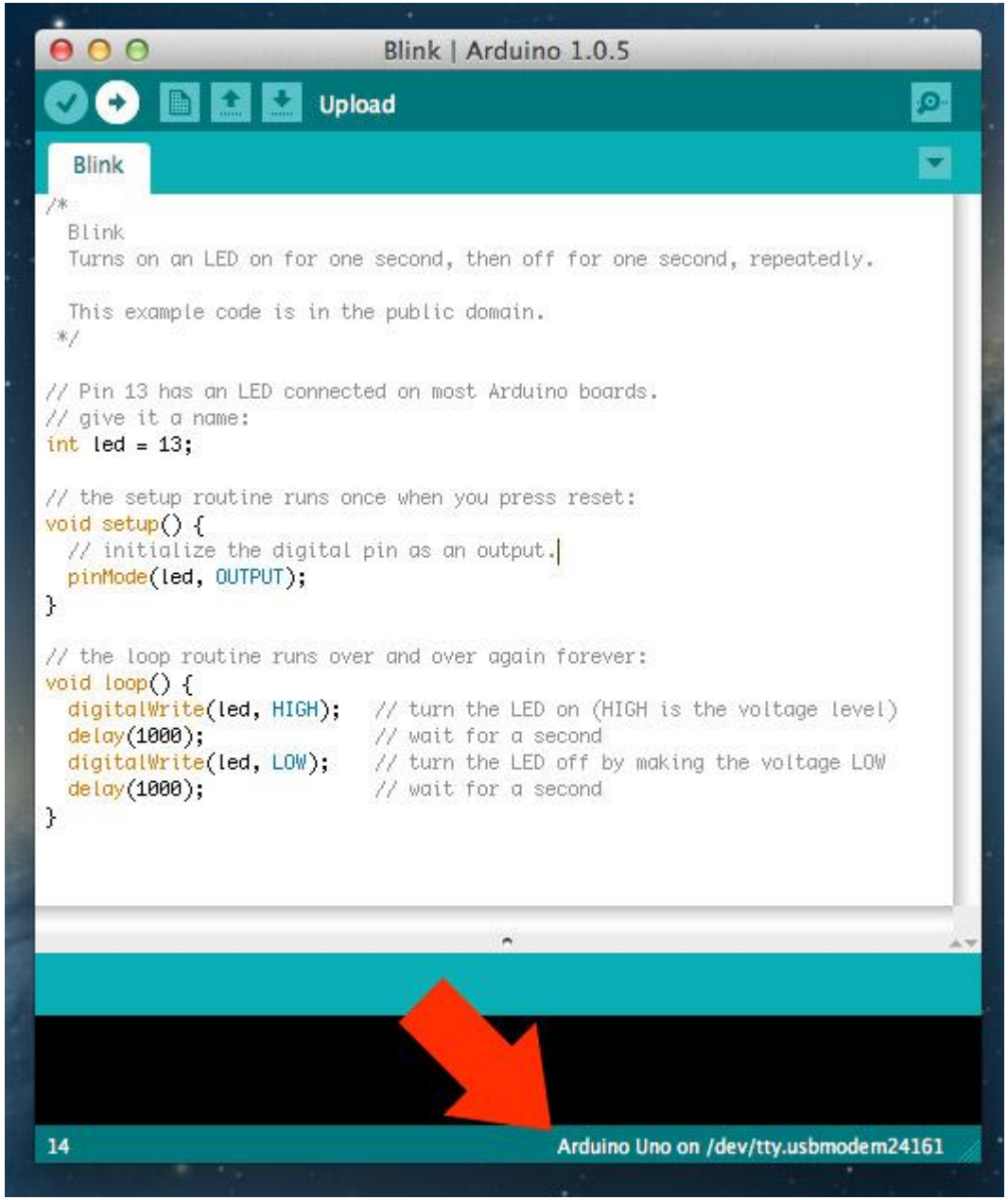
Su **Mac** potete selezionare indifferentemente /dev/tty.usbmodemXXXXX oppure /dev/cu.usbmodemXXXXX

Su **Windows** dovreste notare una o più porte COM, selezionate quella con numero più elevato, se non dovesse funzionare provate con le altre proposte.

Su **Linux** (Ubuntu) dovreste vedere una ttyACM0. Per maggiori informazioni consultare la sezione Arduino su Ubuntu su: www.maffucci.it/area-studenti/arduino/

aprire sketch blink | 5/8 | B

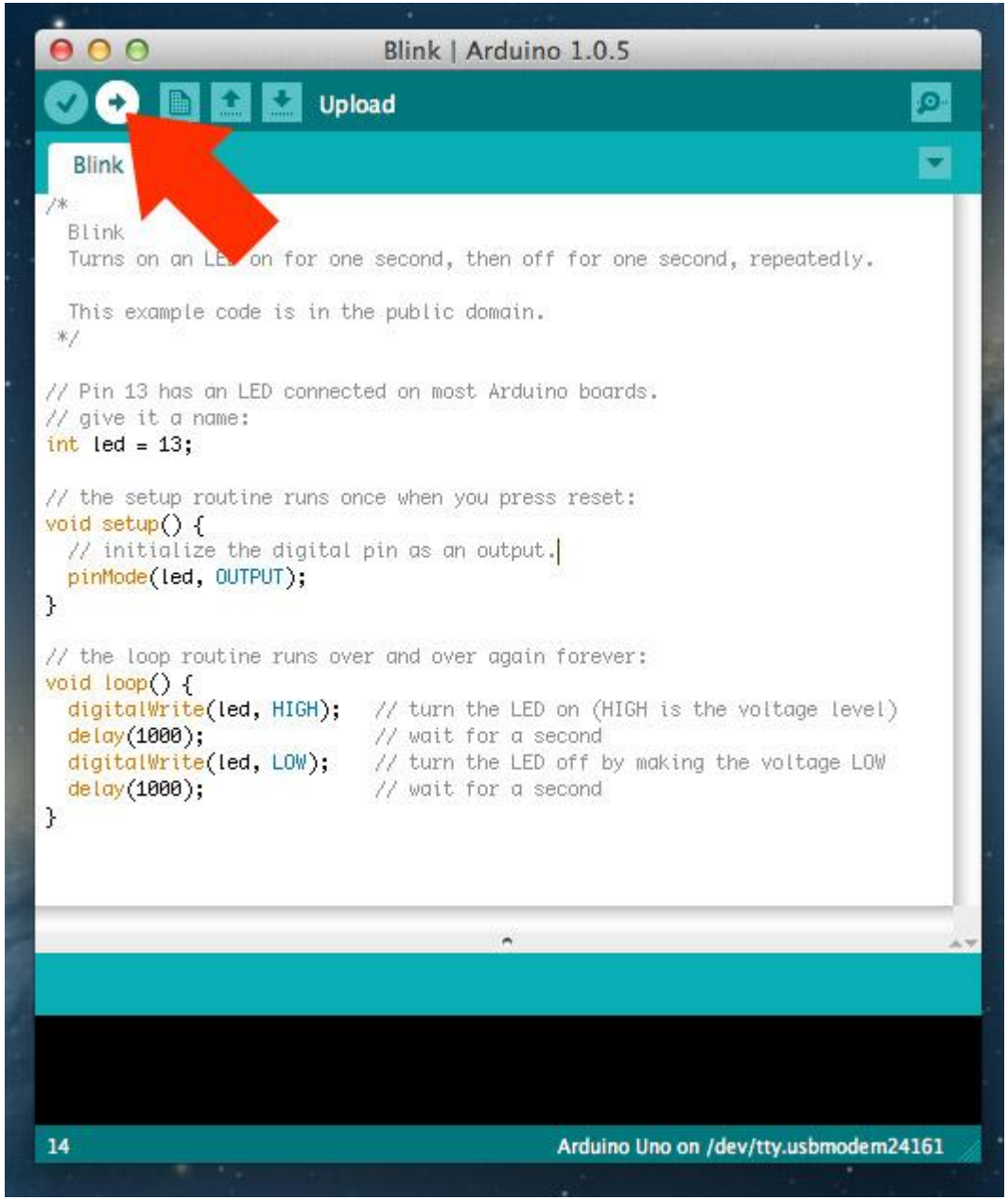
Il collegamento alla porta seriale viene segnalato nella finestra del codice in basso a destra



aprire sketch blink

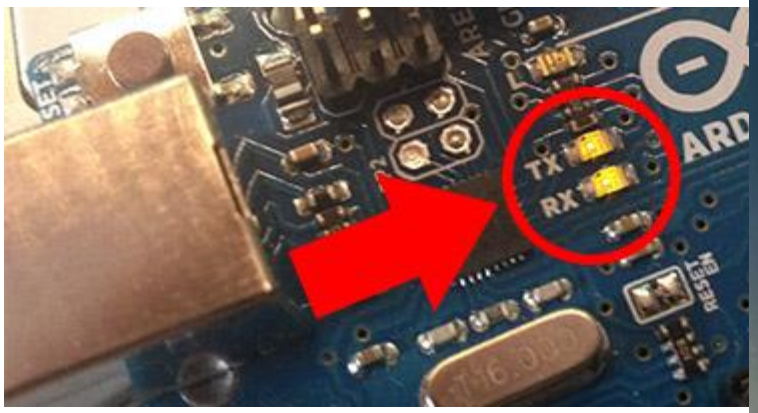
6/8 A

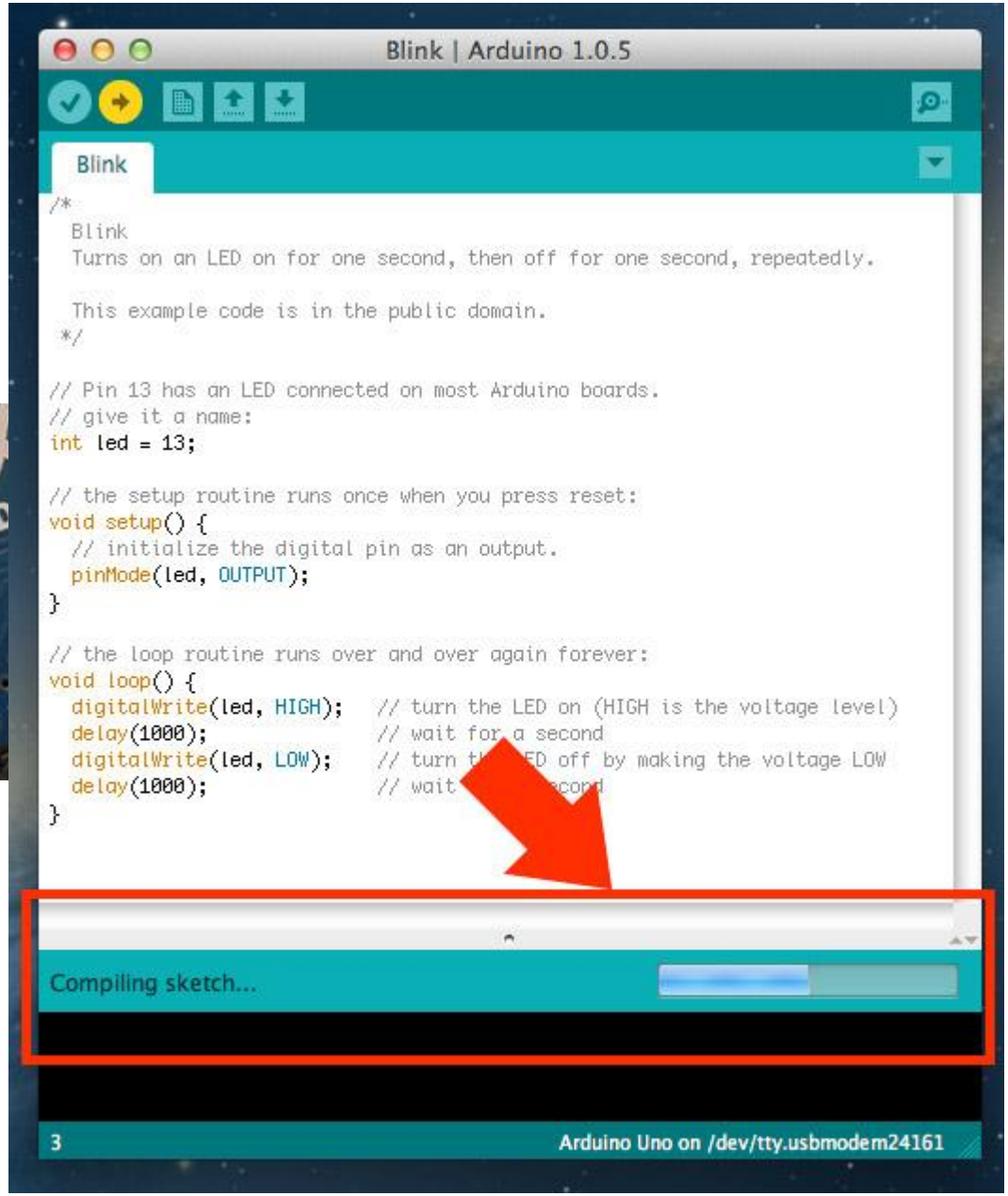
Procedere con il caricamento dello sketch Blink sulla scheda mediante il pulsante Upload nella finestra in cui compare il codice:



aprire sketch blink 6/8 B

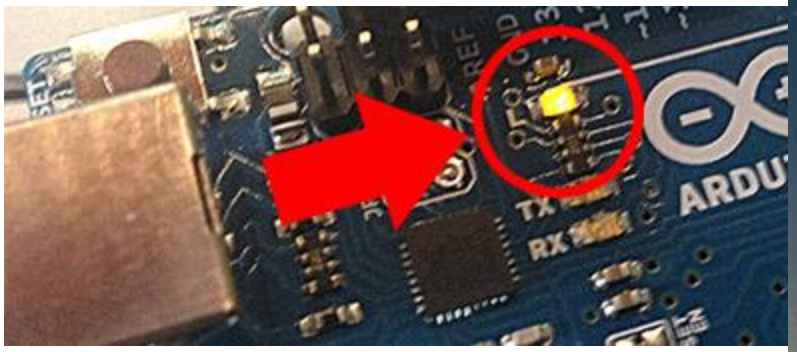
Ci vorrà qualche secondo, durante questa operazione vedrete che i led RX e TX (ricezione e trasmissione) lampeggiano.

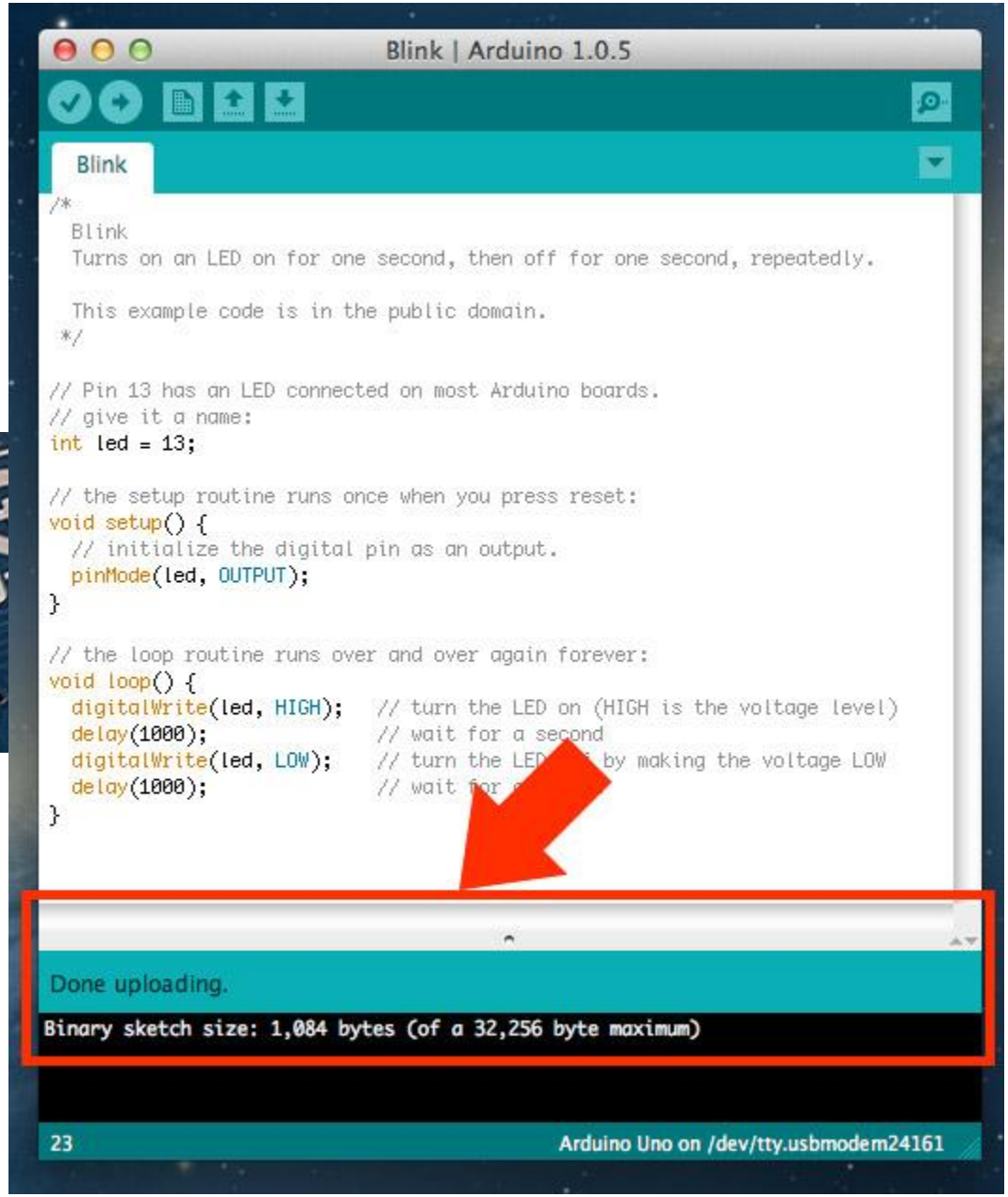




aprire sketch blink

Se tutto andrà a buon fine vi verrà restituito il messaggio "Done uploading." nella staus bar ed il LED L incomincia a lampeggiare

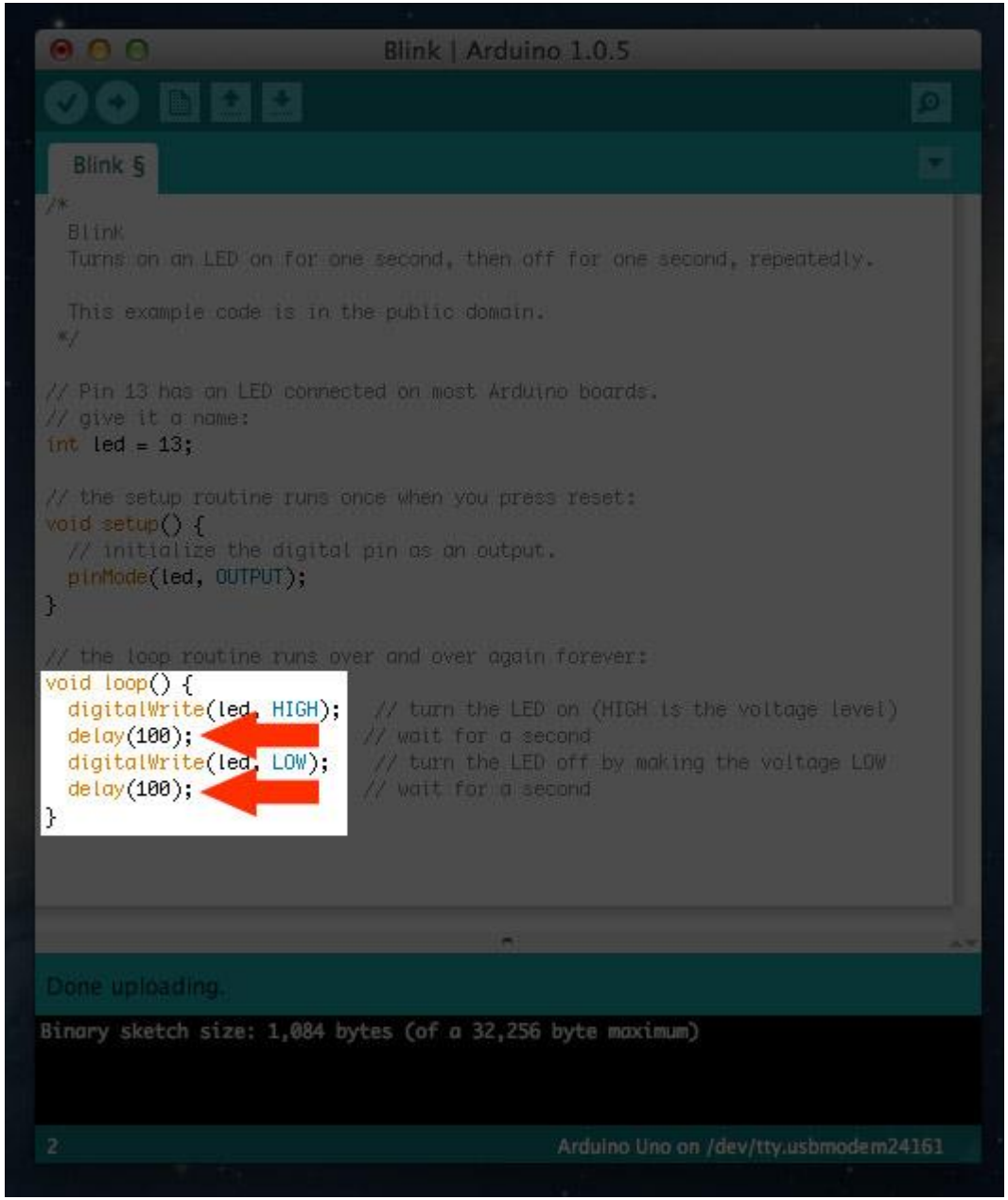




aprire sketch blink

Sulla scheda Arduino, se nuova e mai utilizzata, viene precaricato lo sketch Blink, quindi appena viene collegata la scheda al computer il LED L lampeggia.

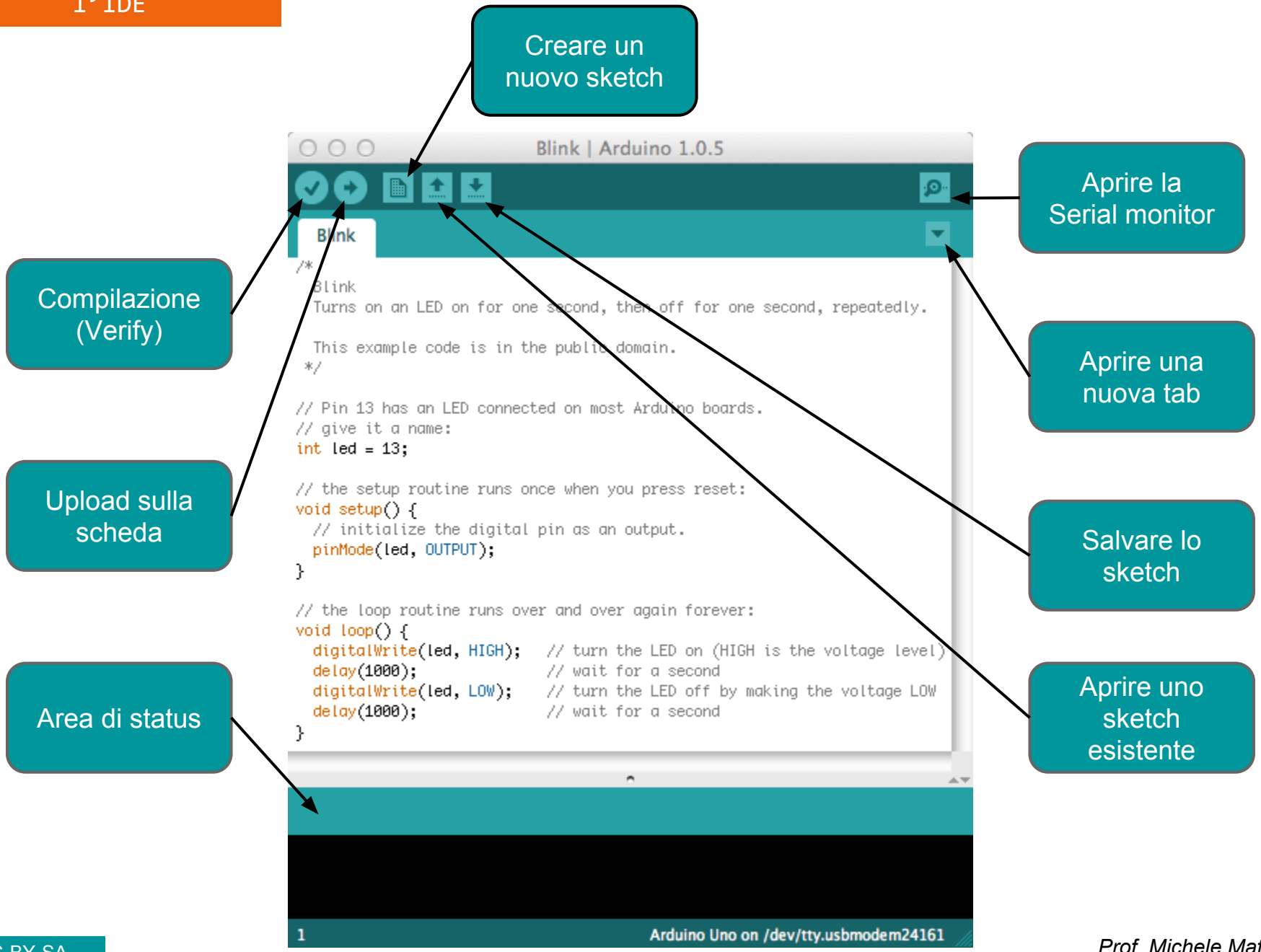
Per essere certi che lo sketch è stato caricato sulla scheda provate a variare il numero all'interno del comando delay, ponete il valore 100. Effettuate l'upload dello sketch, al termine dovreste notare che il LED L lampeggia molto più velocemente.



Programmazione

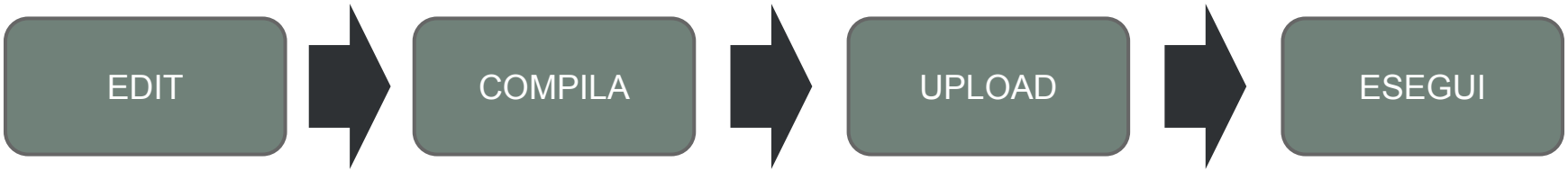
Prof. Michele Maffucci

l'IDE


Creare un nuovo sketch
Aprire la Serial monitor
Compilazione (Verify)
Aprire una nuova tab
Upload sulla scheda
Salvare lo sketch
Area di status
Aprire uno sketch esistente
Blink | Arduino 1.0.5
Blink
/*
Blink
Turns on an LED on for one second, then off for one second, repeatedly.
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
}
1
Arduino Uno on /dev/tty.usbmodem24161

Ciclo di sviluppo

Il ciclo di sviluppo è suddiviso in 4 fasi:



Compila. *Compilare vuol dire tradurre lo sketch in linguaggio macchina, detto anche codice oggetto*
Esegui. *uno sketch Arduino viene eseguito non appena termina la fase di upload sulla scheda*

Il linguaggio

Il linguaggio di programmazione è un C standard (ma molto più semplice)

Le funzioni più usate e che impareremo ad utilizzare durante le lezioni sono:

`pinMode()`
impostare un pin come input o come output

`digitalWrite()`
impostare un pin digitale a HIGH o LOW

`digitalRead()`
legge lo stato di un pin digitale

`analogRead()`
legge un pin analogico

`analogWrite()`
scrive in valore analogico

`delay()`
mette in attesa il programma per un determinato tempo

`millis()`
restituisce l'ora corrente (tempo di accensione di Arduino)

Altre funzioni con esempi di utilizzo potete trovarle seguendo il link.

Il primo programma

analisi del codice | 1/8 | sketch00

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

  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
}
```

A — Commento su più linee

B — Commento su una linea

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

  This example code is in the p
 */

// Pin 13 has an LED connected on most Arduino boards.
// give it a name:
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);               //
}
```

; — identifica dove termina un'istruzione

{ … } — identifica un blocco di istruzioni

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

  This example code is in the pu
 */

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

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

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

`int led = 13;`

Una variabile è un modo per nominare e memorizzare un valore numerico per un successivo utilizzo da parte del programma.

Tutte le variabili devono essere dichiarate prima di poter essere utilizzate. Dichiarare una variabile significa:

- definire il tipo del valore che può assumere: int, long, float, ecc…
- assegnare un nome
- e opzionalmente assegnargli un valore iniziale

queste operazioni vengono fatte una volta sola nel programma, ma il valore della variabile può essere modificato in qualsiasi momento usando l'aritmetica o utilizzando delle assegnazioni.

Nell'esempio che segue viene dichiarato che *led* è un int, (tipo intero) e che il suo valore iniziale è uguale a 13. Questo è chiamata assegnazione semplice.

Per approfondimenti seguire il link.

```
/*
  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;
}
```

La struttura base di un programma Arduino è abbastanza semplice e si sviluppa in almeno due parti. Queste due parti, o funzioni, necessarie racchiudono parti di istruzioni.

Dove **setup()** indica il blocco di settaggio e **loop()** è il blocco che viene eseguito. Entrambe le sezioni sono necessarie per far si che uno sketch funzioni.

setup() è la prima funzione ad essere invocata verrà eseguita una volta sola e in essa vengono dichiarate le variabili usate nel programma, è usata per impostare il pinMode o inizializzare la comunicazione seriale.

La funzione **loop()** contiene il codice che deve essere eseguito ripetutamente, in essa vengono letti gli input, i segnali di output ecc…
Questa funzione è la parte principale di un programma Arduino (sketch), esegue la maggior parte del lavoro.

```
void setup()
{
  istruzioni;
}

void loop()
{
  istruzioni;
}
```

Per approfondimenti seguire il link.

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

  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 whe
void setup() {
  // initialize the digital pin as
  pinMode(led, OUTPUT);
}

// the loop routine runs over and
void loop() {
  digitalWrite(led, HIGH);   // tu
  delay(1000);               // wa
  digitalWrite(led, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);               // wait for a second
}
```

`pinMode(led, OUTPUT);`

pinMode è un'istruzione che dice ad Arduino come usare un determinato pin.

Tra parentesi tonde vengono specificati gli argomenti che possono essere numeri e lettere.

I pin digitali possono essere utilizzati sia come **INPUT** che come **OUTPUT**.
Nel nostro caso poiché vogliamo far lampeggiare il diodo LED dobbiamo definire il pin di OUTPUT.

Le parole INPUT e OUTPUT sono costanti definite, che non variano mai nel linguaggio di Arduino.

Per approfondimenti seguire il link.

```
/*
  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);      rn
  delay(1000);                  // wait
  digitalWrite(led, LOW);    // turn
  delay(1000);               // wait
}
```

digitalWrite(led, HIGH);

L'istruzione digitalWrite possiede due argomenti:

il primo definisce il pin,
il secondo indica lo stato.

digitalWrite è un'istruzione in grado di impostare un pin definito come OUTPUT ad un valore HIGH o ad un valore LOW, in modo più semplice permette di accendere o spegnere un led connesso al pin specificato nel primo argomento, nel nostro caso LED.

Il 'pin' può essere specificato come una variabile o una costante (0-13).

Tenete conto che dire che su un determinato pin vi è uno stato HIGHT, vuol dire che su di esso viene applicata una tensione di +5 V, mentre se lo stato e LOW vuol dire che sul pin è applicata una tensione di 0V.

Per approfondimenti seguire il link.

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

  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
void setup() {
  // initialize the digital pin as o
  pinMode(led, OUTPUT);
}

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

`delay(1000);`

delay() è un'istruzione che interrompe per un determinato tempo l'esecuzione del programma.

L'istruzione ha un solo argomento numerico che indica il numero di **millisecondi** di attesa.

Con **"`delay(1000)`"** il programma si bloccherà per 1000 millisecondi ovvero 1 secondo.

Per approfondimenti seguire il link.

Esercizio 1
Variate il tempo di accensione e spegnimento, con tempi uguali di accensione e spegnimento (superiore ad 1 secondo).

Esercizio 2
Variate il tempo di accensione e spegnimento, con tempi diversi di accensione e spegnimento (superiore ad 1 secondo).

Esercizio 3
Provate ad eseguire il programma con tempi di accensione e spegnimento del LED con tempi di:

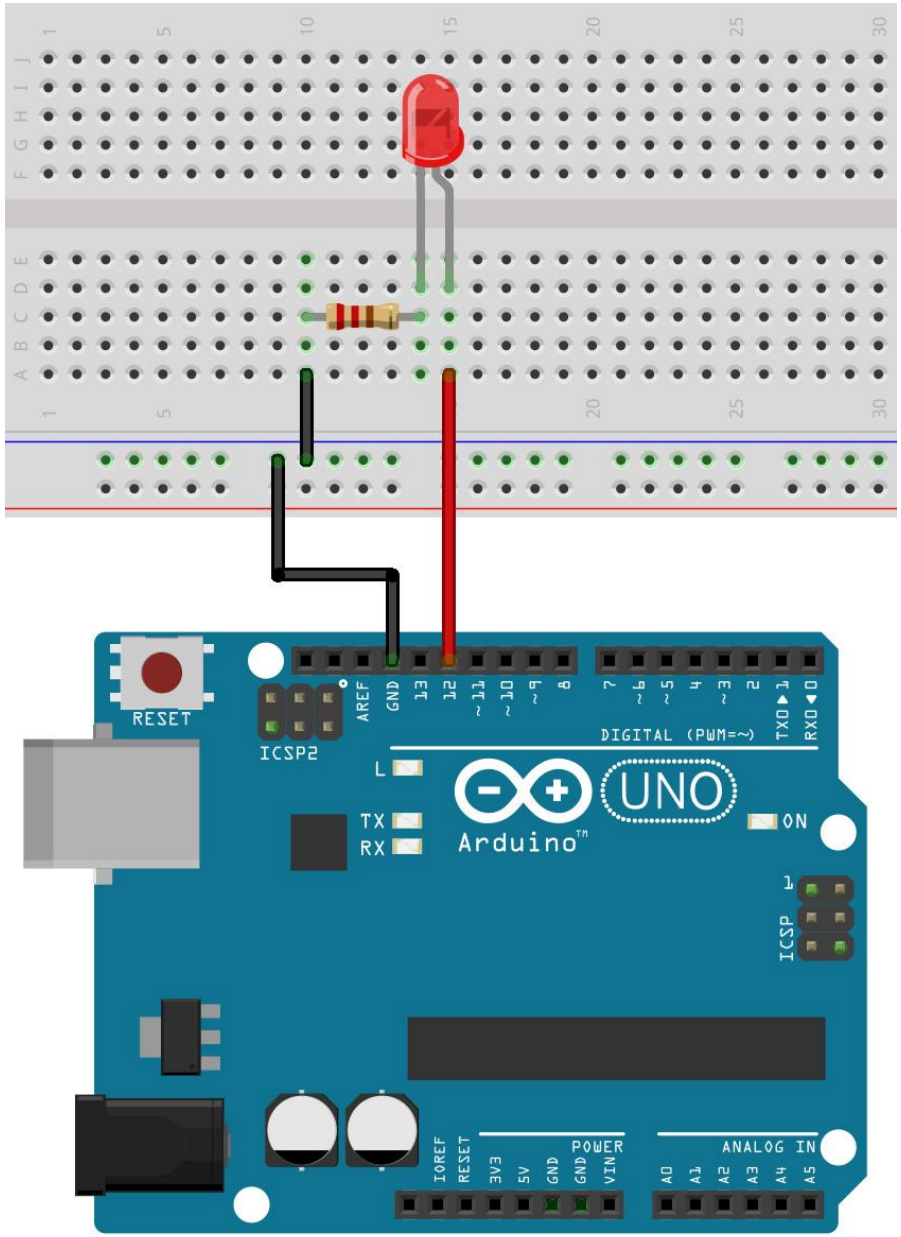
- 500 ms
- 250 ms
- 100 ms
- 50 ms

Verificare cosa accade al diminuire del tempo di accensione e spegnimento.

esercizio | 2/4 | sketch04

Esercizio 4
Utilizzando un LED esterno (diverso dal LED L della scheda) ed una resistenza da collocare in serie al LED per limitare la corrente del circuito, far funzionare il programma Blink. Utilizzare un qualsiasi pin digitale dal 3 al 12 (nell'esempio è stato usato il pin 12) e verificatene il funzionamento con tempi di accensione e spegnimento a piacere.

R = 220 Ohm



Per la realizzazione dello schema di montaggio è stato utilizzato il software Fritzing, per maggiori informazioni seguire il link.

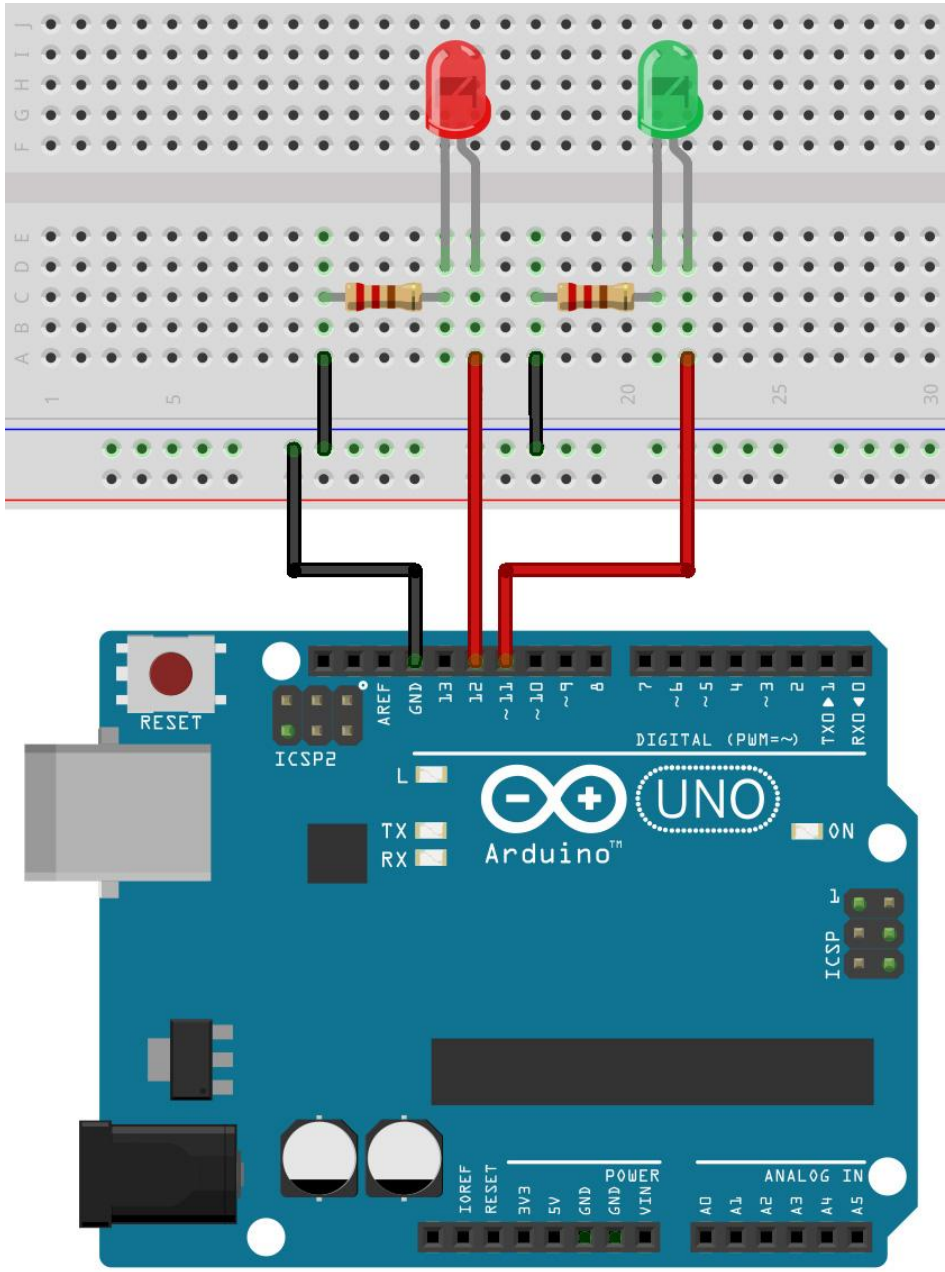
esercizio | sketch05 da a-d

Esercizio 5
Utilizzando due LED (che chiameremo LED01 e LED02) realizzare un doppio lampeggiatore in tre modalità:

1. accensione e spegnimento contemporaneo;
2. accensione e spegnimento alternato;
3. tre accensioni e spegnimenti del LED01 e successivamente 5 accensioni e spegnimenti del LED02.
4. Accensione in sequenza alternata di 10 LED

Utilizzare un qualsiasi pin digitale dal 3 al 12 (nell' esempio è stato usato il pin 12 e 11) fissare i tempi di accensione e spegnimento ad 1 secondo.

R = 220 Ohm



ciclo FOR | 1/2 | sketch05 da a-d

Nel precedente esercizio 5 alla domada 3, con le nozioni a vostra disposizione, avrete realizzato una sequenza di accensioni e spegnimenti utilizzando ripetutamente, nel caso del LED01 una sequenza di tre blocchi di codice:

```
digitalWrite(LED01, HIGH);   // accende il LED01 (livello di tensione HIGH)
delay(1000);                 // attesa di 1 secondo
digitalWrite(LED01, LOW);    // spegne il LED01 (livello di tensione LOW)
delay(1000);
```

nel caso del LED02 una seuenza di 5 blocchi di codice:

```
digitalWrite(LED02, HIGH);   // accende il LED02 (livello di tensione HIGH)
delay(1000);                 // attesa di 1 secondo
digitalWrite(LED02, LOW);    // spegne il LED02 (livello di tensione LOW)
delay(1000);
```

Esiste un metodo più semplice per poter ripetere per un numeo di volte stabilito una sequenza di istruzioni, questa operazione la si ottiene mediante il ciclo **for**.

L'istruzione **for** è usata per ripetere, per un determinato numero di volte, una serie di istruzioni, la ripetizione avviene se risulta vera una certa condizione logica, nel caso risulti falsa si esce dal ciclo.

ciclo FOR 2/2 sketch05 da a-d

Per poter eseguire l'iterazione è indispensabile inizializzare una variabile usata come contatore che verrà incrementata o decrementata dopo l'esecuzione delle istruzioni all'interno del for. Ad ogni ciclo viene controllato, all'interno del blocco "condizione", se la variabile contatore soddisfa una determinata condizione logica in modo che si possa proseguire attraverso il ramo "**Vero**" oppure il ramo "**Falso**".

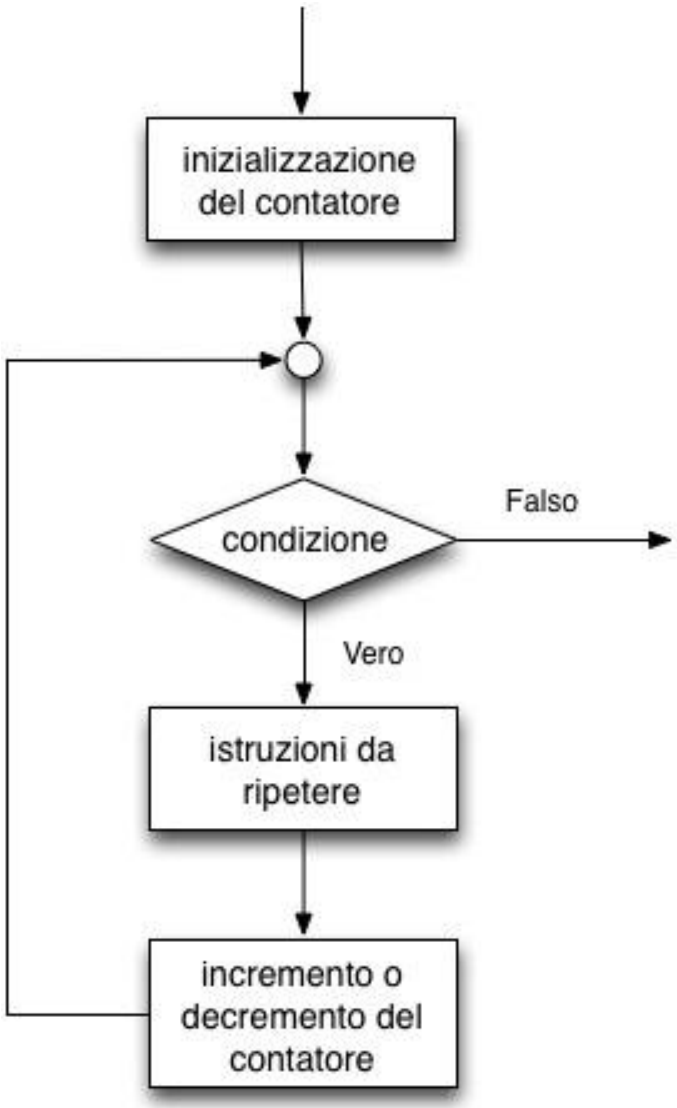
Sintassi

```
for (inizializzazione; condizione; espressione)
{
  qualcosa;
}
```

"**inizializzazione**" è una variabile locale che viene incrementata o decrementata da una espressione "**esperssione**" ad ogni ciclo di del loop.

Ad ogni ciclo la condizione "**condizione**" viene verificata, se risulta **VERA** vengono eseguite le istruzioni incluse nelle parentesi graffe che seguono il **for** e successivamente viene nuovamente verificata "**condizione**", quando "**condizione**" risulta **FALSA** il loop termina.

Per approfondimenti seguire il link.



Esercizio 5.3

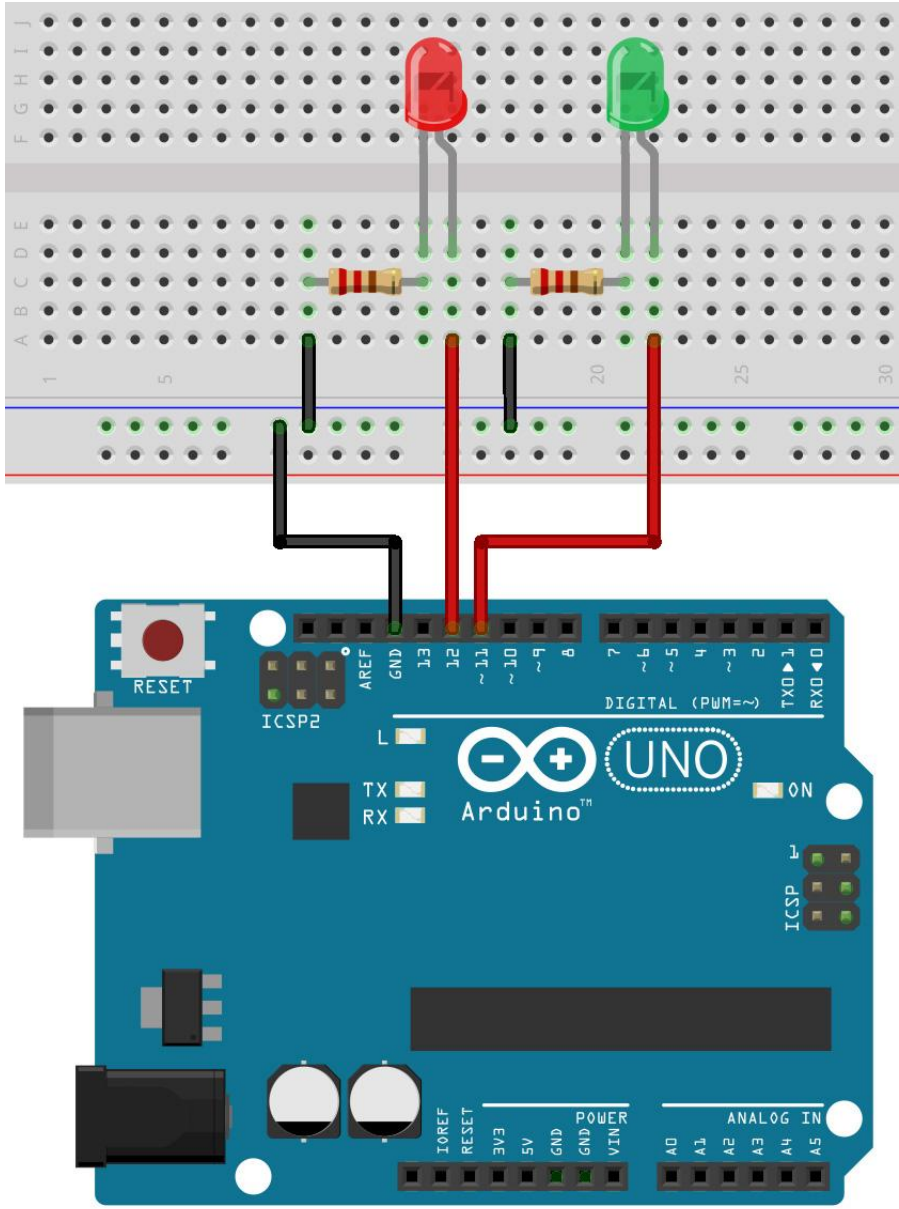
Utilizzando due LED (che chiameremo LED01 e LED02) realizzare un doppio lampeggiatore iun doppio lampeggiatore con la seguente funzionalità:

- tre accensioni e spegnimenti del LED01 e successivamente 5 accensioni e spegnimenti del LED02.

Utilizzare:

- l'istruzione **for** per la realizzazione dello sketch.
- un qualsiasi pin digitale dal 3 al 12 (nell' esempio è stato usato il pin 12 e 11) fissare i tempi di accensione e spegnimento ad 1 secondo.

R = 220 Ohm



Usiamo il con ciclo FOR

sketch05 da a-d

```
int LED01 = 11;
int LED02 = 12;

void setup() {
  // inizializzazione dei pin digitali come output.
  pinMode(LED01, OUTPUT);
  pinMode(LED02, OUTPUT);
}

void loop() {

  // ciclo LED01
  for (int i=0; i<3; i++)       // variabile i usata per contare i cicli di accensione
  {                             // e spegnimento
    digitalWrite(LED01, HIGH); // accende il LED01 (livello di tensione HIGH)
    delay(1000);                // attesa di 1 secondo
    digitalWrite(LED01, LOW);  // spegne il LED01 (livello di tensione LOW)
    delay(1000);                // attesa di 1 secondo
  }

  // ciclo LED02
  for (int i=0; i<5; i++)       // variabile i usata per contare i cicli di accensione
  {                             // e spegnimento
    digitalWrite(LED02, HIGH); // accende il LED02 (livello di tensione HIGH)
    delay(1000);                // attesa di 1 secondo
    digitalWrite(LED02, LOW);  // spegne il LED02 (livello di tensione LOW)
    delay(1000);                // attesa di 1 secondo
  }
}
```

Usiamo il con ciclo FOR

sketch05 da a-d

Nel **primo** ciclo **for** la variabile intera i, usata come **inizializzazione** del ***contatore di ciclo*** e viene inizializzata a **0**.

Le istruzioni incluse nel ciclo **for** verranno eseguite fino a quando la **condizione** di controllo **i<3** risulta vera.

Ad ogni ciclo il contatore i viene incrementato di 1 mediante l'istruzione i++ (analogo all'istruzione i=i+1)

La variabile **i** assumerà i valori **0, 1, 2**.

```
for (int i=0; i<3; i++)        // variabile i usata per contare i cicli di accensione
 {                             // e spegnimento
   digitalWrite(LED01, HIGH); // accende il LED01 (livello di tensione HIGH)
   delay(1000);               // attesa di 1 secondo
   digitalWrite(LED01, LOW);  // spegne il LED01 (livello di tensione LOW)
   delay(1000);               // attesa di 1 secondo
 }
```

Usiamo il con ciclo FOR

sketch05 da a-d

Nel **secondo** ciclo **for** la variabile intera i, usata come **inizializzazione** del ***contatore di ciclo*** e viene inizializzata a **0**.

Le istruzioni incluse nel ciclo **for** verranno eseguite fino a quando la **condizione** di controllo **i<5** risulta vera.

Ad ogni ciclo il contatore i viene incrementato di 1 mediante l'istruzione i++ (analogo all'istruzione i=i+1)

La variabile **i** assumerà i valori **0, 1, 2, 3, 4**.

```
for (int i=0; i<5; i++)        // variabile i usata per contare i cicli di accensione
 {                             // e spegnimento
   digitalWrite(LED02, HIGH); // accende il LED02 (livello di tensione HIGH)
   delay(1000);               // attesa di 1 secondo
   digitalWrite(LED02, LOW);  // spegne il LED02 (livello di tensione LOW)
   delay(1000);               // attesa di 1 secondo
 }
```

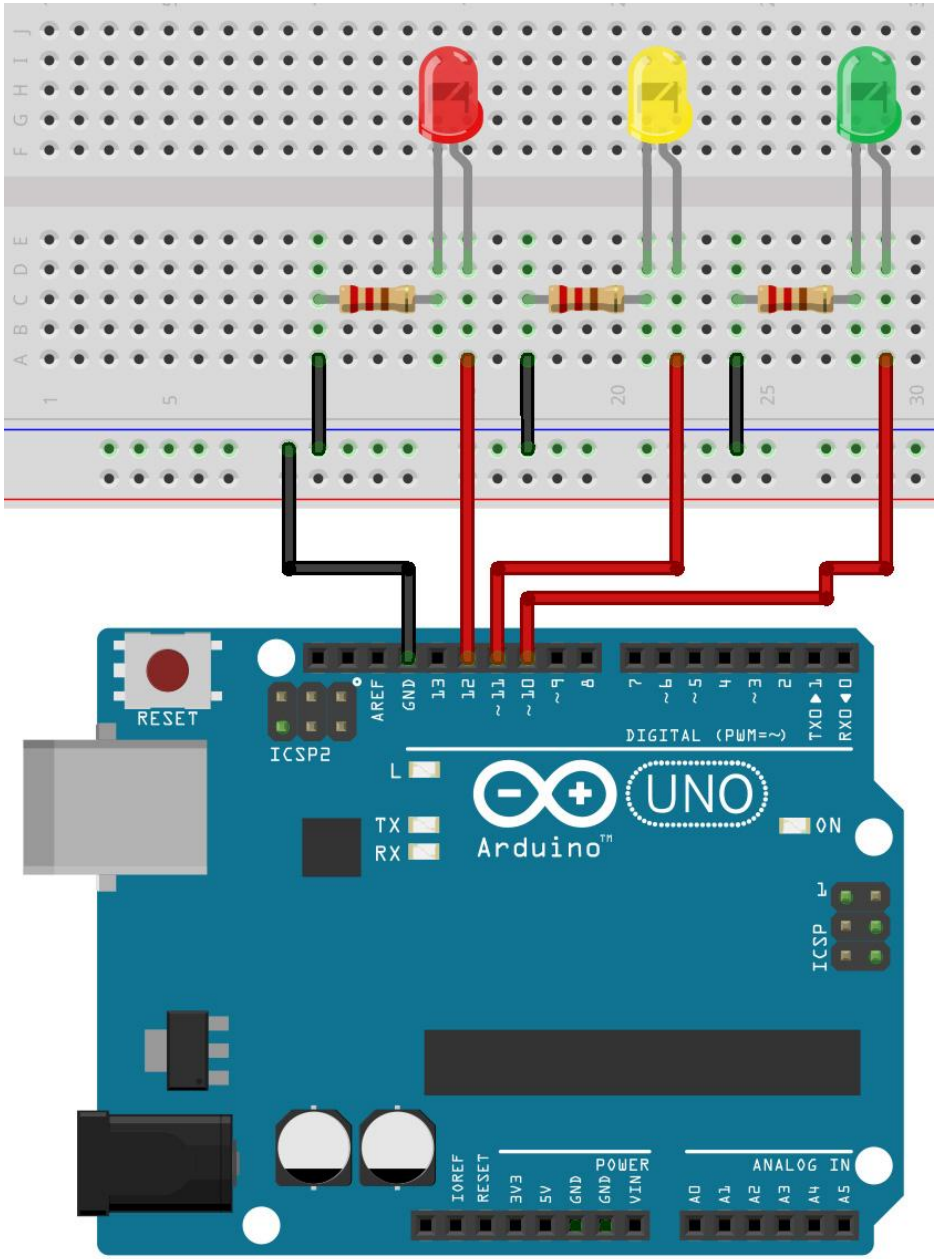
esercizio | 1/4 | sketch06

Esercizio 6
Simulare il funzionamento di un semaforo

Nota bene: *i tempi di accensione (cambiamenti di stato) non sono identici a quelli di un semaforo reale al fine di ridurre i tempi di attesa durante la sperimentazione.*

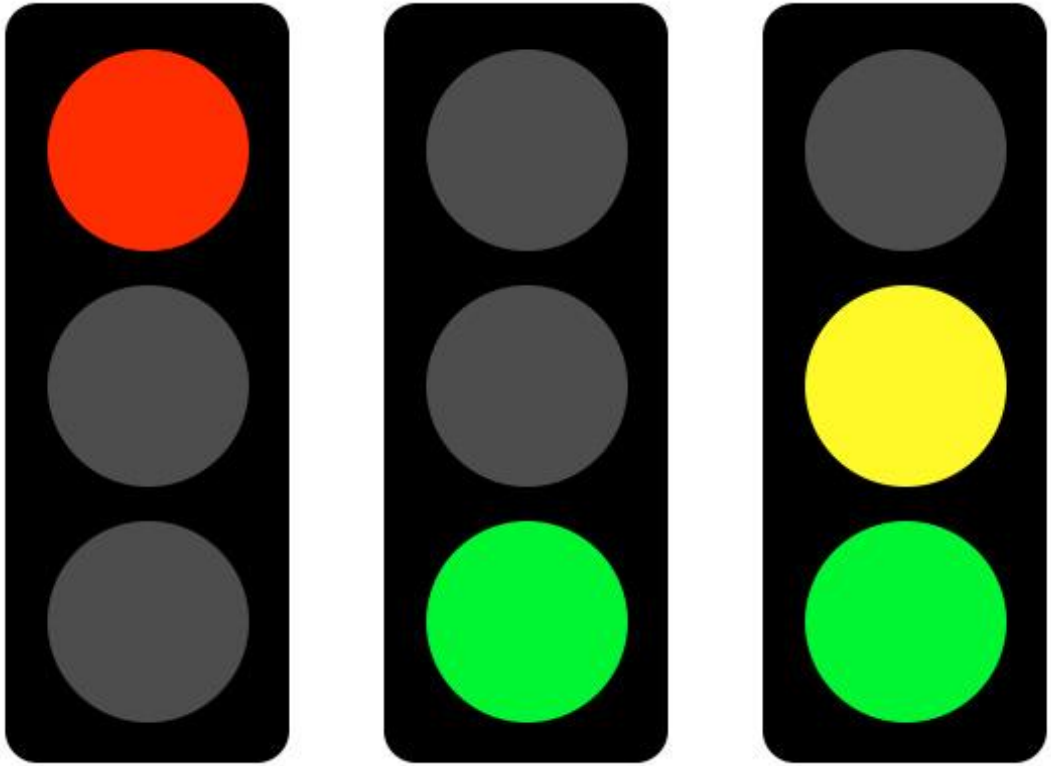
- Breadboard
- n. 1 diodo LED rosso
- n. 1 diodo LED giallo
- n. 1 diodo LED verde
- n. 3 resistenze da 220Ω
- cavi di connessione

Costruite il circuito utilizzando i **pin digitali 10, 11, 12** ricordate di collegare i **catodi dei LED a GND**.



Specifiche del programma

- durata del rosso **10 secondi**
- durata del verde **10 secondi**
- durata del giallo **5 secondi**



Struttura del programma

- blocco - **rosso** per 10 secondi
- blocco - **verde** per 10 secondi
- blocco - **verde** e **giallo** per 5 secondi

Commentare il programma

- Il programma dovrà essere commentato:
- dovrà avere una intestazione costituita dallo scopo del progetto
- il nome del programmatore
- la data di creazione
- ogni blocco logico dovrà essere commentato spiegandone l'azione

Da ricordare

I commenti possono essere di due tipi:

blocco commento su più linee

```
/* questo è un blocco commento chiuso
non dimenticare di chiudere il commento
anche i segni di blocco commento sono bilanciati
*/
```

blocco commento su singola linea

```
// questo è un commento su una sola linea
```

Da ricordare

La struttura base di un programma Arduino è abbastanza semplice e si sviluppa in almeno due parti.
Queste due parti, o funzioni, necessarie racchiudono parti di istruzioni.

```
void setup()
{
  istruzioni;
}

void loop()
{
  istruzioni;
}
```

Molto spesso per la realizzazione del programma (sketch) è necessario usare dichiarare delle variabili globali prima delle funizione setup()

```
dichiarazione
variabili globali

void setup()
{
  istruzioni;
}

void loop()
{
  istruzioni;
}
```

esercizio | sketch06b

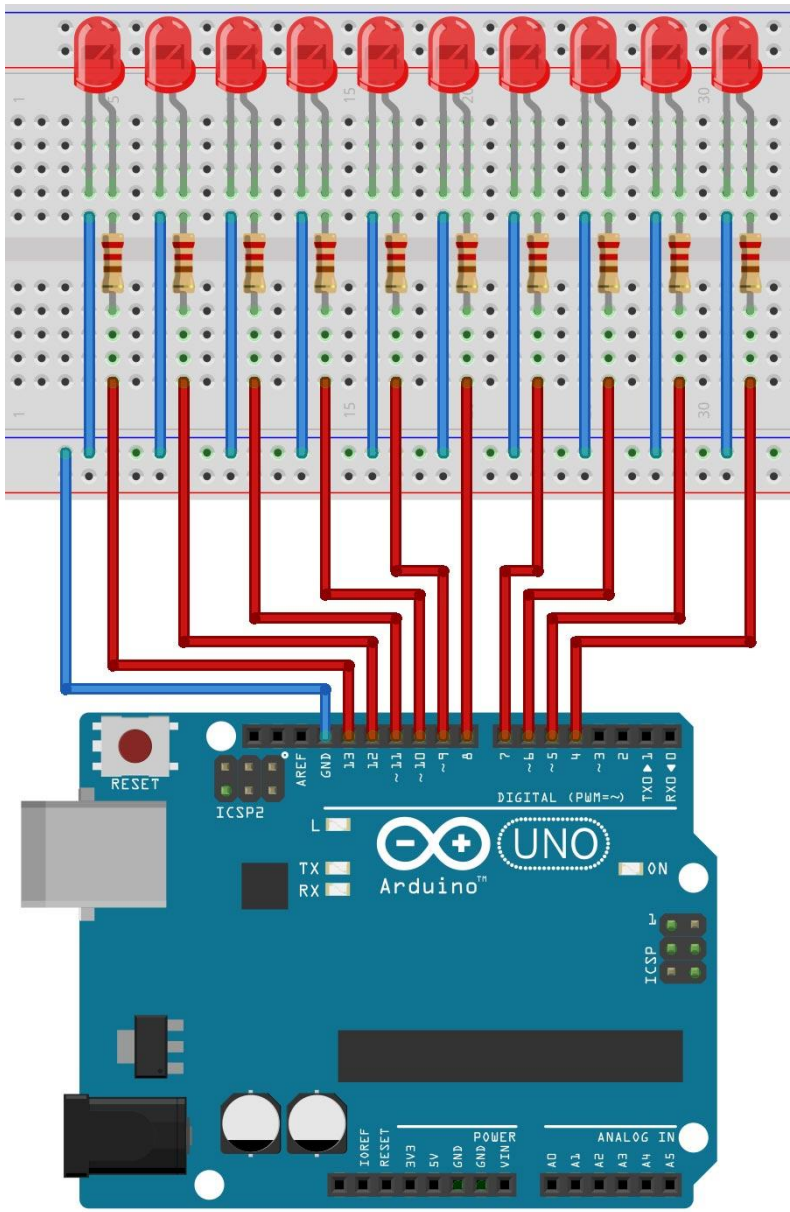
Esercizio 6b

Realizziamo una sequenza di accensione e spegnimento alternata.

Questo esercizio ha lo scopo di introdurre all'uso degli array e alla gestine del tempo con Arduino. Si consiglia di prelevare lo sketch e consultare i link che rimandano alle lezioni on-line.

- Breadboard
- n. 10 diodo LED
- n. 10 resistenze da 220Ω
- cavi di connessione

Costruite il circuito utilizzando i **pin digitali dal 4 al 13** ricordate di collegare i **catodi dei LED a GND**.

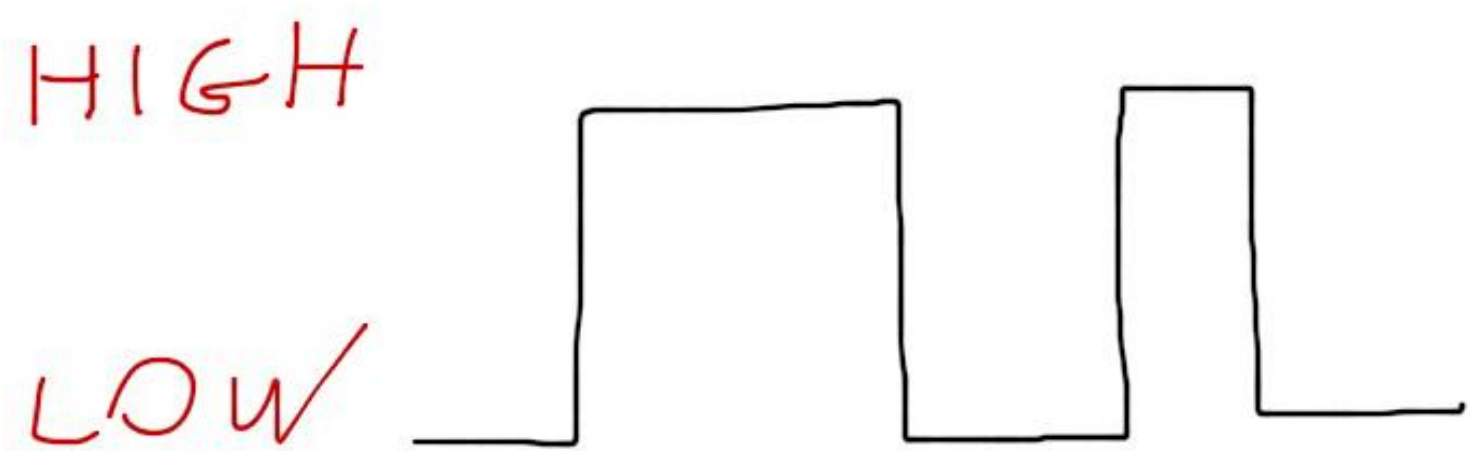


Input digitali

HIGH / LOW

Interruttori e pulsanti consentono di effettuare interruzioni e connessioni del passaggio di corrente all' interno di un circuito, ma Arduino per comprendere che un componente o un suo piedino è connesso o non connesso ha necessità di leggere una tensione elettrica e nello specifico:

- livello logico **ALTO** > **HIGH** > equivalente a 5 volt = Vcc
- livello logico **BASSO** > **LOW** > equivalente a 0 volt = GND



APERTO / CHIUSO

Interruttori e pulsanti vengono definiti **CHIUSI** (resistenza tra i terminali minori di 1 Ohm equivalente ad un cortocircuito), quando consentono il passaggio di corrente.

Se il passaggio di corrente non è consentito si definisce il collegamento **APERTO** (resistenza tra i terminali > 10 MOhm)

Interruttore

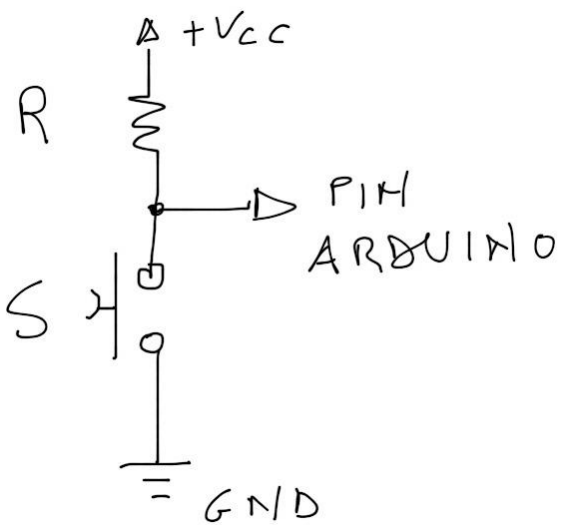
pulsante

Collegamento di pulsante normalmente aperto (N.O. Normaly Open)

Dire un pulsante è **normalmente aperto**, vuol dire che quando non premiamo il pulsante questo interrompe il circuito (non permette il passaggio di corrente), possiamo avere due tipologie di collegamento N.O.:

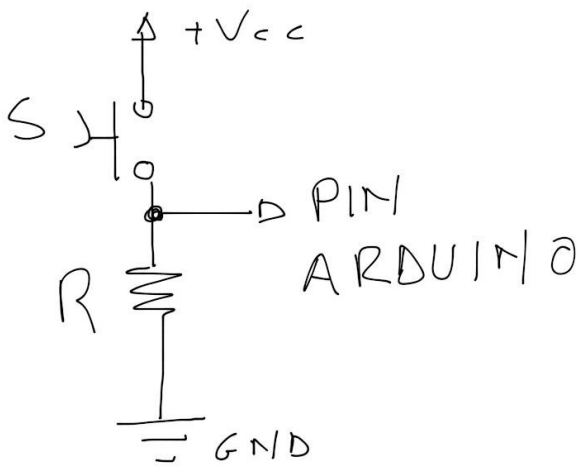
1. Con resistenza **pull-up** in cui la resistenza è collegata direttamente a +Vcc e il pulsante a GND

- Pulsante **premuto** > pin Arduino collegato a GND (0 V) > livello logico di uscita **0**
- Pulsante **rilasciato** > pin Arduino collegato a +Vcc (5 V) > livello logico di uscita **1**



2. Con resistenza **pull-down** in cui la resistenza è collegata direttamente a GND e il pulsante a +Vcc

- Pulsante **premuto** > pin Arduino collegato a +Vcc (5 V) > livello logico di uscita **1**
- Pulsante **rilasciato** > pin Arduino collegato a GND (0 V) > livello logico di uscita **0**

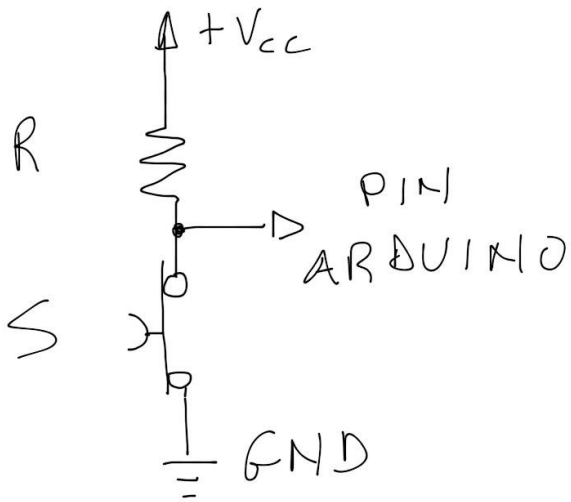


Collegamento di pulsante normalmente chiuso (N.C. Normaly Close)

Dire un pulsante è **normalmente chiuso**, vuol dire che quando non premiamo il pulsante questo non interrompe il circuito (permette il passaggio di corrente), possiamo avere due tipologie di collegamento N.C.:

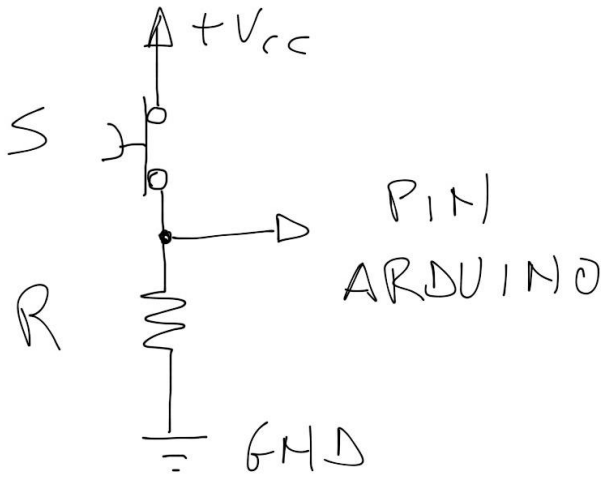
1. Con resistenza **pull-up** in cui la resistenza è collegata direttamente a +Vcc e il pulsante a GND

- Pulsante **premuto** > pin Arduino collegato a +Vcc (5 V) > livello logico di uscita **1**
- Pulsante **rilasciato** > pin Arduino collegato a GND (0 V) > livello logico di uscita **0**



2. Con resistenza **pull-down** in cui la resistenza è collegata direttamente a GND e il pulsante a +Vcc

- Pulsante **premuto** > pin Arduino collegato a GND (0 V) > livello logico di uscita **0**
- Pulsante **rilasciato** > pin Arduino collegato a +Vcc (5 V) > livello logico di uscita **1**



Scopo

Realizzare un programma che permette di accendere una LED quando premiamo un pulsante (N.O.) e quando viene nuovamente premuto il pulsante spegne il LED, comportamento analogo a quello che si ha per un impianto di illuminazione.

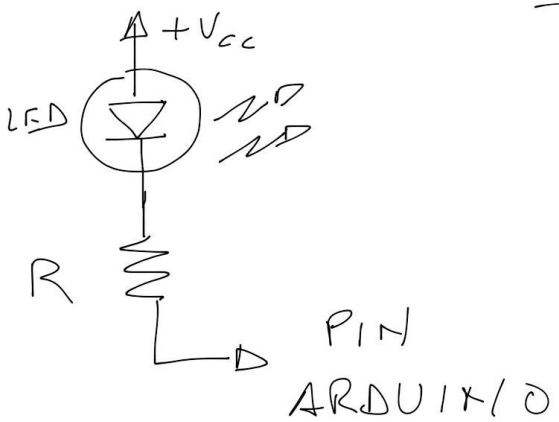
PREMESSA ALLA COSTRUZIONE

Ogni pin di Arduino è in grado di fornire una corrente di uscita di circa 40 mA sufficiente per poter controllare l'accensione di un LED (tipicamente di 20 mA).
Valori di corrente assorbiti o erogati superiori ai 40 mA o tensioni di lavoro superiori a 5V su uno qualsiasi dei pin possono danneggiare il microcontrollore o i dispositivi ad esso collegati.

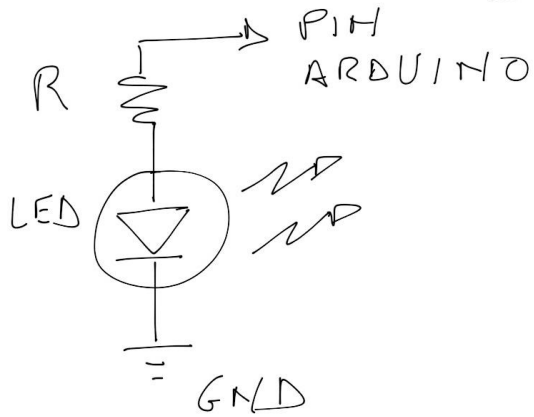
Tipicamente viene inserito in serie al LED una resistenza di circa 220 Ohm, ma LED di colore diverso hanno correnti di funzionamento di diverso valore, ciò implica valori di resistenza diversa, un valore di 220 Ohm può andar bene per quasi tutti i tipi di LED.

Per il calcolo preciso della resistenza da porre in serie al LED consultare il seguente link.

LED ACCESO CON LIVELLO LOW



LED ACCESO CON LIVELLO HIGH



controlliamo un LED con un pulsante

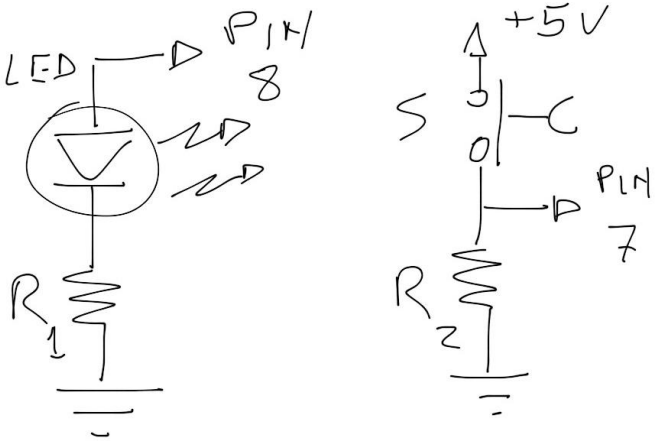
sketch07

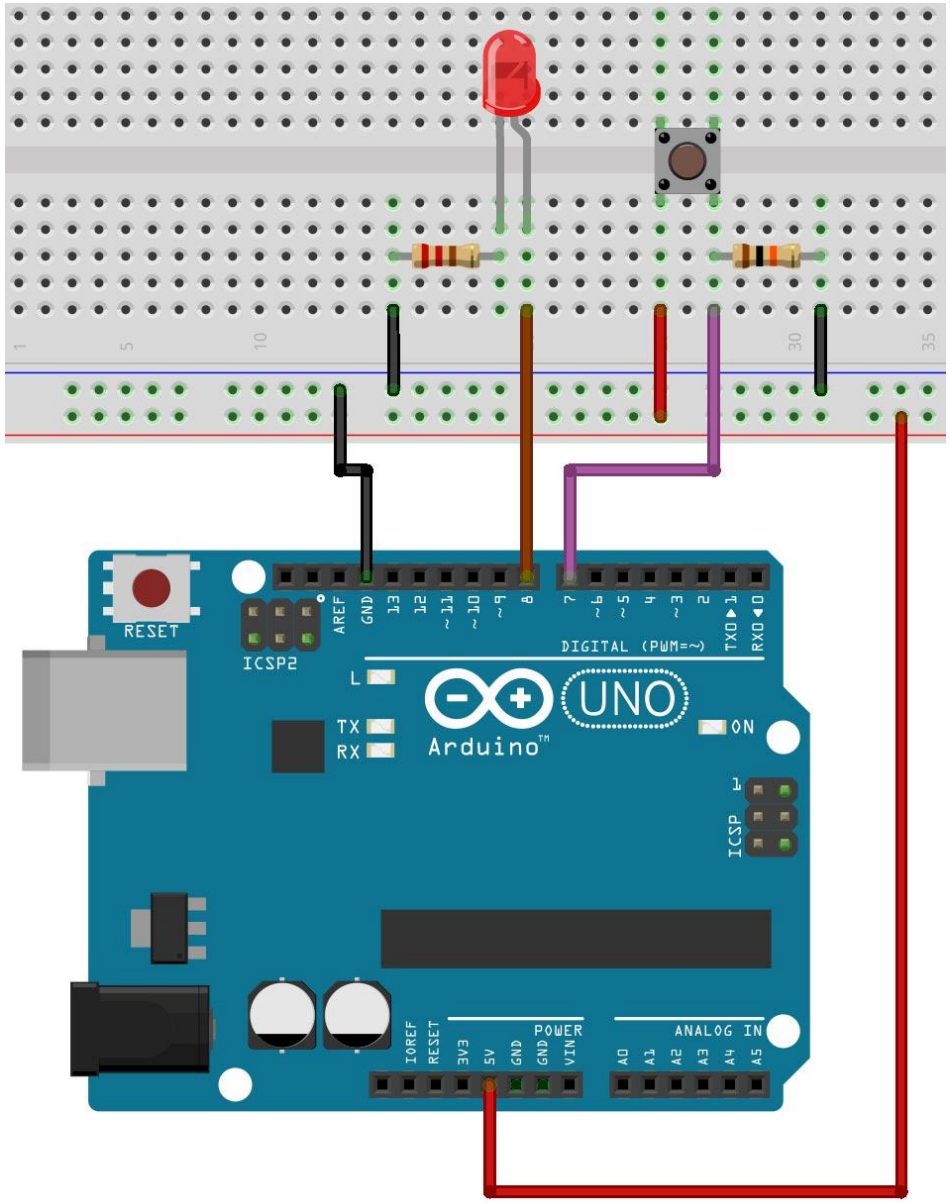
Per controllare lo stato di un pulsante utilizzeremo l'istruzione **digitalRead()**, questa istruzione legge il valore su uno specifico pin digitale che può assumere due valori, **HIGH** o **LOW**, detto in modo meno informatico e più elettronico, verifica se su un determinato pin è applicata una tensione di +5V (definito HIGH) o 0V (definito LOW).

Quindi con digitalRead() possiamo leggere uno stato di un sensore e memorizzare questo stato nella memoria di Arduino per fare qualcosa.

Realizziamo il circuito rappresentato nell'immagine.

R serie LED = 220 Ohm
R serie pulsante = 10 KOhm





controlliamo un LED con un pulsante | 3/6 | sketch07

```
/*
  Prof. Michele Maffucci
  07.03.2014

  Accensione del LED alla pressione d

  Questo codice è di dominio pubblico
*/

int LED = 8;                    // LE
int BUTTON = 7;                 // pi

int val = 0;                    si
                                // pi

void setup() {
  pinMode(LED, OUTPUT);         // imposta il pin digitale come output
  pinMode(BUTTON, INPUT);       // imposta il pin digitale come input
}

void loop() {
  val = digitalRead(BUTTON);    // legge il valore dell'input e lo conserva

  // controlla che l'input sia HIGH (pulsante premuto)
  if (val == HIGH) {
    digitalWrite(LED, HIGH);    // accende il LED (livello di tensione HIGH)
  }
  else {
    digitalWrite(LED, LOW);     // spegne il LED
  }
}
```

`int val = 0;`

Per ricordare lo stato del pulsante, definiamo una variabile **intera** di nome ***val*** a cui assegnamo inizialmente il valore **0**.

In ***val*** viene conservato **digitalRead()**

I valori di qualsiasi variabile usata con Arduino vengono inseriti nella memoria RAM, **valori che vengono persi una volta che si toglie l' alimentazione alla scheda Arduino**.

controlliamo un LED con un pulsante

sketch07

```
/*
  Prof. Michele Maffucci
  07.03.2014

  Accensione del LED alla pressione del pulsante

  Questo codice è di dominio pubblico
*/

int LED = 8;                    // LED collegato al pin digitale 8
int BUTTON = 7;                 // pin di input dove è collegato il pulsante

int val = 0;                    // si userà val per conservare lo stato del
                                // pin di input a cui è collegato il pulsante

void setup() {
  pinMode(LED, OUTPUT);         // impo
  pinMode(BUTTON, INPUT);       // impo
}

void loop() {
  val = digitalRead(BUTTON);

  // controlla che l'input sia HIGH (pulsante premuto)
  if (val == HIGH) {
    digitalWrite(LED, HIGH);    // accende il LED (livello di tensione HIGH)
  }
  else {
    digitalWrite(LED, LOW);     // spegne il LED
  }
}
```

`val = digitalRead(BUTTON);`

All'interno della codice di loop() viene assegnata alla variabile **val** lo stato del pulsante (valore in input al pin 7) mediante l'istruzione **digitalRead(BUTTON),** ovvero l'istruzione legge il valore dell'input (pulsante) e restituisce un valore 0 oppure 1

controlliamo un LED con un pulsante

sketch07

```
/*
  Prof. Michele Maffucci
  07.03.2014

  Accensione del LED alla pressione del pulsante

  Questo codice è di dominio pubblico
*/

int LED = 8;                    // LED collegato al pin digitale 8
int BUTTON = 7;                 // pin di input dove è collegato il pulsante

int val = 0;                    // si userà val per conservare lo stato del
                                // pin di input a cui è collegato il pulsante

void setup() {
  pinMode(LED, OUTPUT);         // imposta il pin digitale come output
  pinMode(BUTTON, INPUT);       // i
}

void loop() {
  val = digitalRead(BUTTON);    // l

  // controlla che l'input sia HIGH
  if (val == HIGH) {
    digitalWrite(LED, HIGH);    // ac
  }
  else {
    digitalWrite(LED, LOW);     // sp
  }
}
```

`if (val == HIGH)...`

Mediante l'istruzione **if...else** si controlla se il pulsante è stato premuto.

Se il valore di val è **HIGH** (valore logico 1) vuol dire che il pulsante è premuto e allora, tramite l'istruzione "**digitalWrite(LED, HIGH);**" viene acceso il LED, se il primo confronto risulta falso, ciò vuol dire che il pulsante non è premuto come conseguenza viene eseguita la parte di codice **else** che spegne il LED, ciò viene eseguito con l' istruzione: "**digitalWrite(LED, LOW);**"

Esercizio 7
Utilizzare due pulsanti per controllare l' accensione dei due LED secondo le seguenti regole:

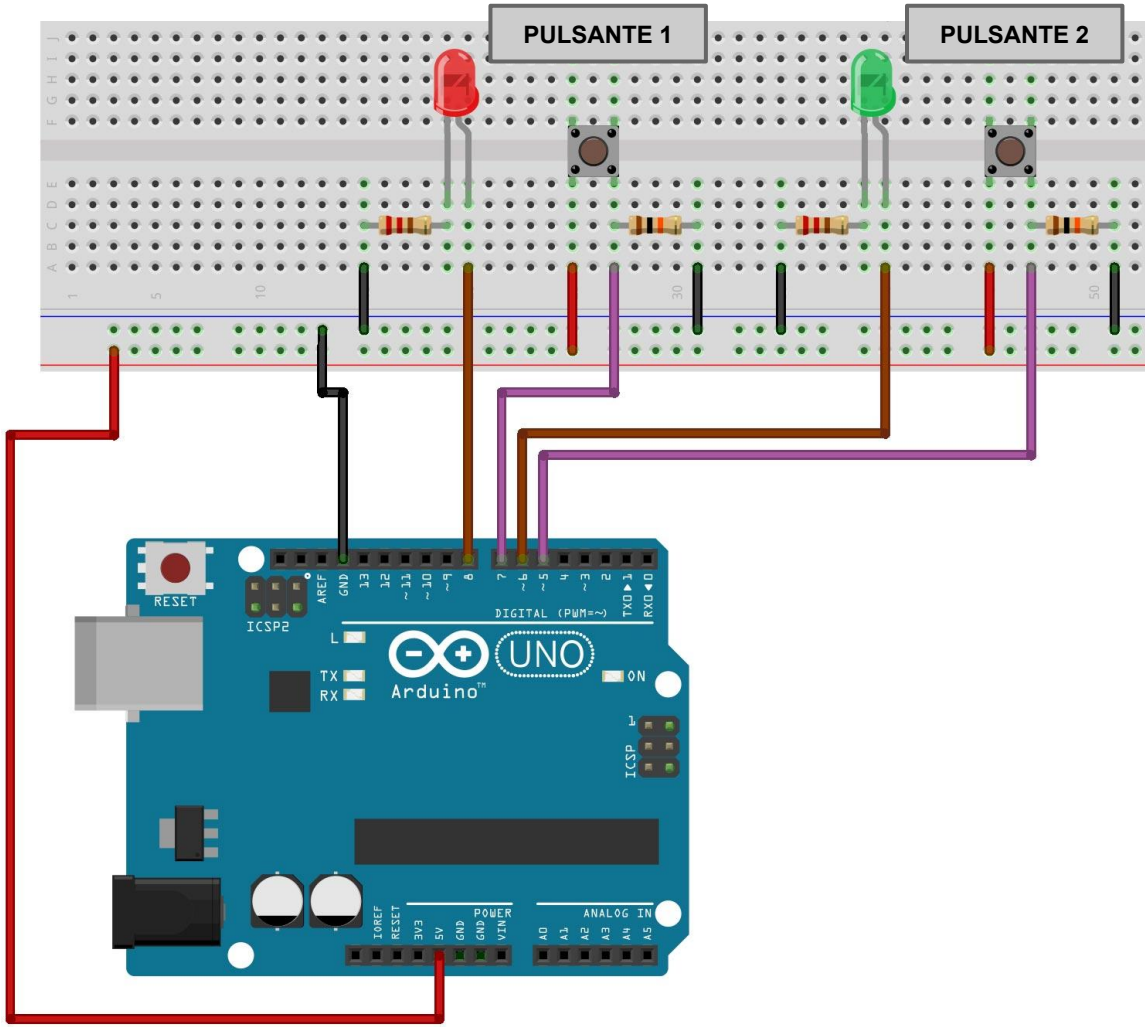
- premendo il pulsante 1 il LED rosso lampegga ad intervalli di 1 secondo
- premendo il pulsante 2 il LED verde lampeggia ad intervalli di 250 millisecondi

Componenti e strumentazione:

- Breadboard
- n. 1 diodo LED rosso
- n. 1 diodo LED verde
- n. 2 resistenze da 220Ω
- n. 2 resistenze da 1KΩ
-

cavi di connessione

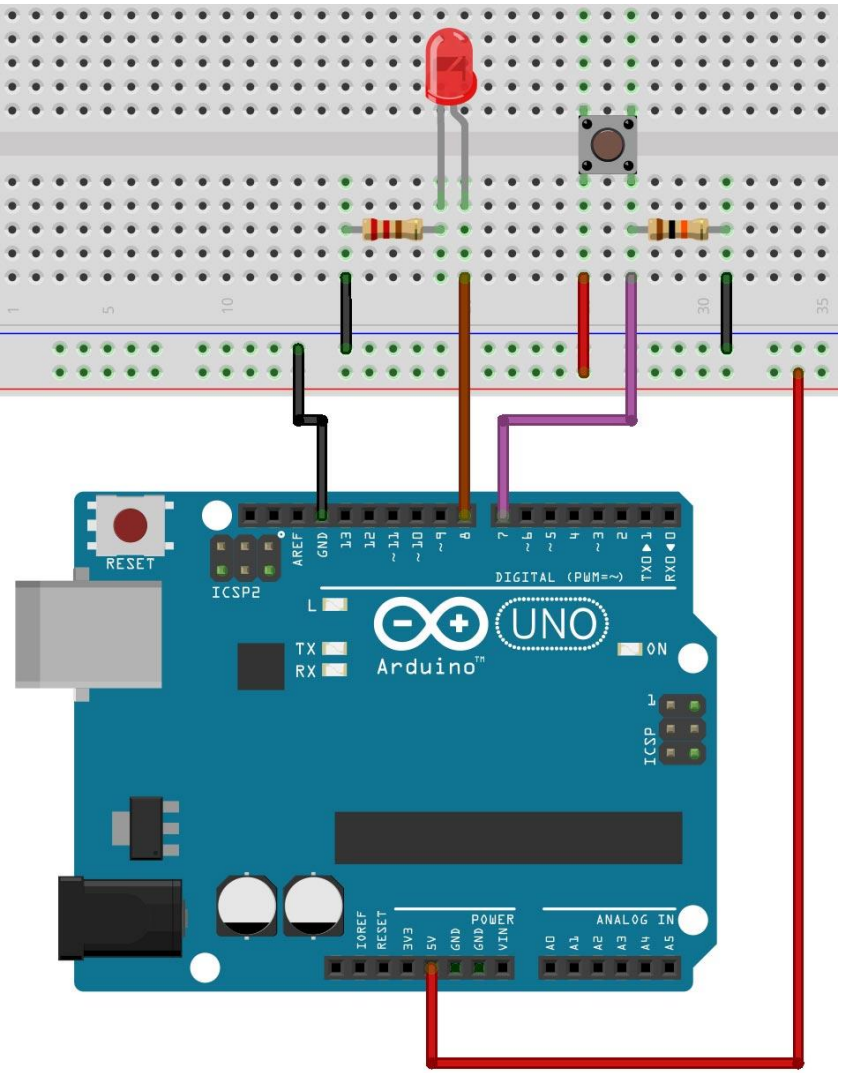
Costruite il circuito utilizzando i **pin digitali 5, 6, 7, 8**.

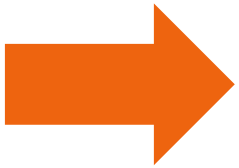


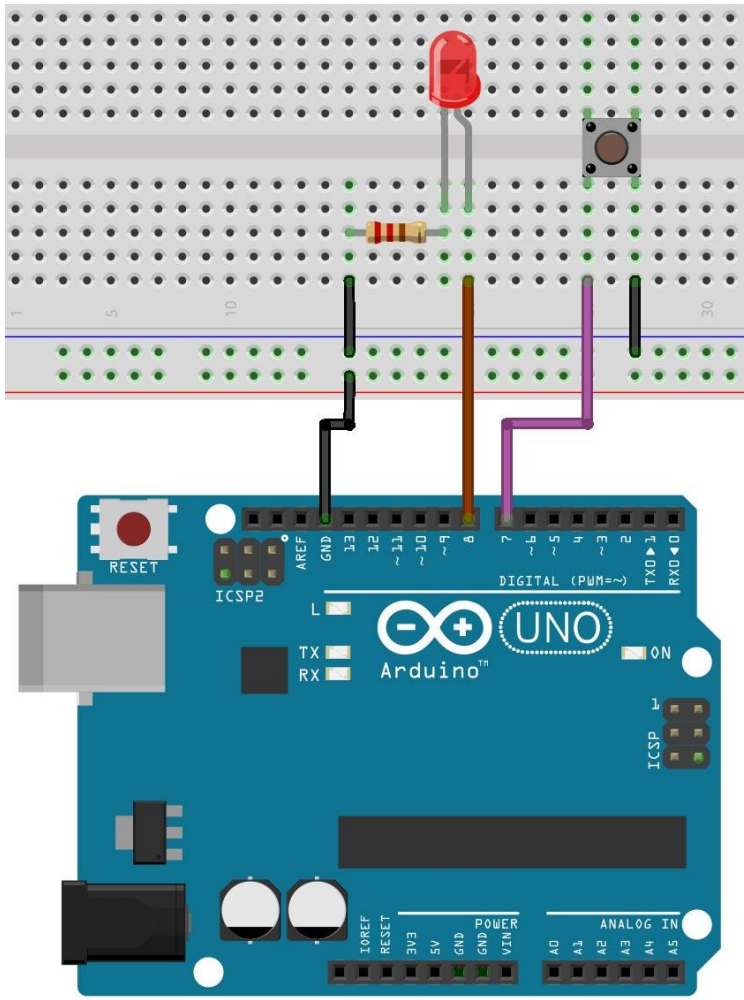
Cosa succede se premiamo tutti e due i pulsanti contemporaneamente?
Come può essere evitata la contemporaneità di stato?

eliminiamo la resistenza di pull-up

Si vuole mostrare come eliminare la resistenza di pull-up sul pulsante.







eliminiamo la resistenza di pull-up

Gli input digitali devono essere provvisti di una resistenza che mantenga i pin su un valore conosciuto, quando il pulsante non viene premuto.

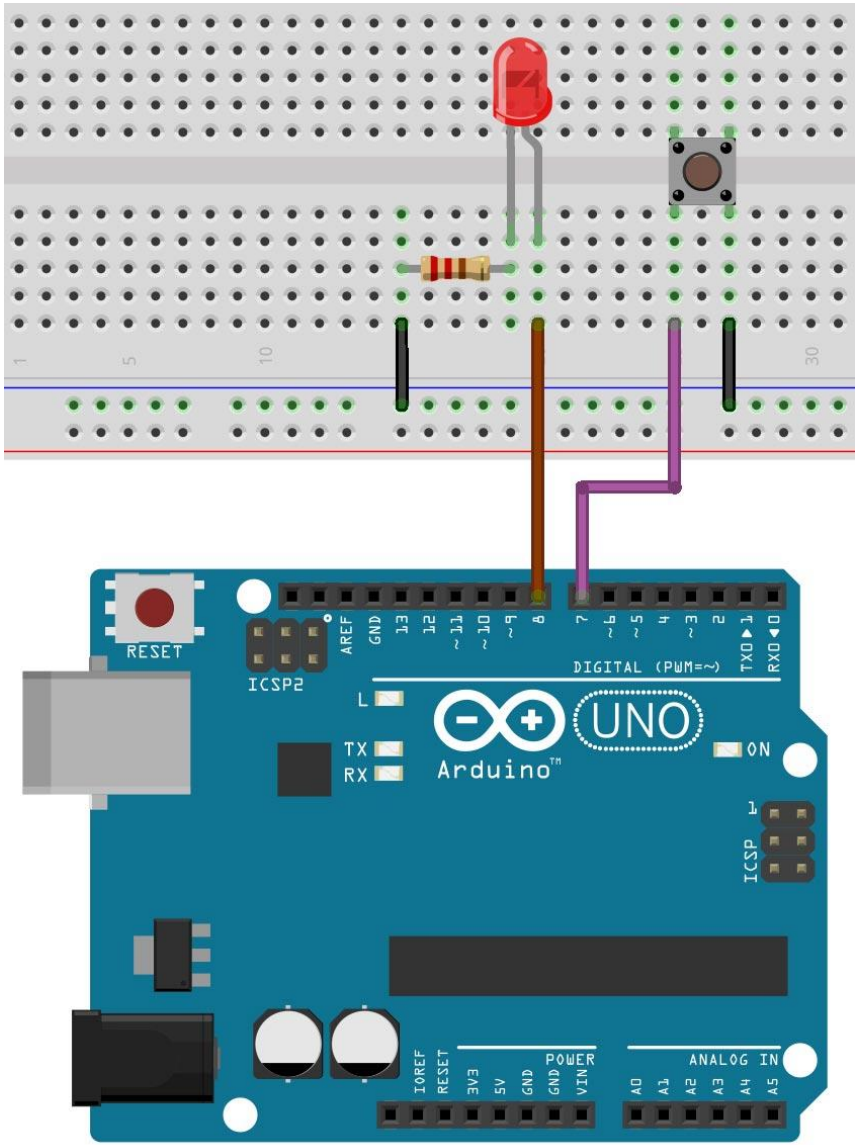
Arduino è dotato di resistenze interne di pull-up che possono essere attivate ponendo ad **HIGH** un pin che si trova in modalità **INPUT**.

Attenzione che se si imposta un pin di **OUTPUT** come **HIGH** e poi lo si passa in modalità **INPUT**, la resistenza di pull-up rimane attiva e se si va a leggere il valore del pin si ottiene **HIGH**.

Se impostiamo il pin su **LOW** in modalità **OUTPUT** usando il comando **digitalWrite(pin, LOW)** e poi si passa questo pin ad INPUT con il comando **pinMode (pin, INPUT)**, la resistenza di pull-up rimane disattivata.

Le resistenze di pull-up interne ad Arduino sono di circa 20 KOhm o più.

Per maggiori informazioni si consulti il seguente link.



eliminiamo la resistenza di pull-up

3/4

sketch09

```
/*
  Prof. Michele Maffucci
  08.03.2014

  Accensione del LED alla pressione del pulsante
  eliminando la resistenza di pull-up sul pulsante

  Questo codice è di dominio pubblico
*/

int LED = 8;                    // LED collegato al pin digitale 8
int BUTTON = 7;                 // pin di input dove è collegato il pulsante

int val = 0;                    // si userà val per conservare lo stato del
                                // pin di input a cui è collegato il pulsante

void setup() {
  pinMode(LED, OUTPUT);         // imposta
  pinMode(BUTTON, INPUT);       // imposta
  digitalWrite(BUTTON, HIGH);   //        a l
}

void loop() {
  val = digitalRead(BUTTON);    // legge il valore dell'input e lo conserva

  // controlla che l'input sia HIGH (pulsante premuto)
  if (val == HIGH) {
    digitalWrite(LED, HIGH);    // accende il LED (livello di tensione HIGH)
  }
  else {
    digitalWrite(LED, LOW);     // spegne il LED
  }
}
```

`digitalWrite(BUTTON, HIGH);`

Attivazione della resistenza di pull-up su BUTTON (pin 7)

eliminiamo la resistenza di pull-up

4/4

sketch09

```
/*
  Prof. Michele Maffucci
  08.03.2014

  Accensione del LED alla pressione del pulsante
  eliminando la resistenza di pull-up sul pulsante

  Questo codice è di dominio pubblico
*/

int LED = 8;                    // LED collegato al pin digitale 8
int BUTTON = 7;                 // pin di input dove è collegato il pulsante

int val = 0;                    // si user
                                // pin di

void setup() {
  pinMode(LED, OUTPUT);         // imposta
  pinMode(BUTTON, INPUT);       // imposta
  digitalWrite(BUTTON, HIGH);   // attiva
}

void loop() {
  val = digitalRead(BUTTON);    // legge i

  // controlla che l'input sia HIGH (pulsante premuto)
  if (val == HIGH) {
    digitalWrite(LED, HIGH);   // accende il LED (livello di tensione HIGH)
  }
  else {
    digitalWrite(LED, LOW);    // spegne il LED
  }
}
```

Nello schetch avete notato che il LED rimane acceso fino a quando non si preme il pulsante.

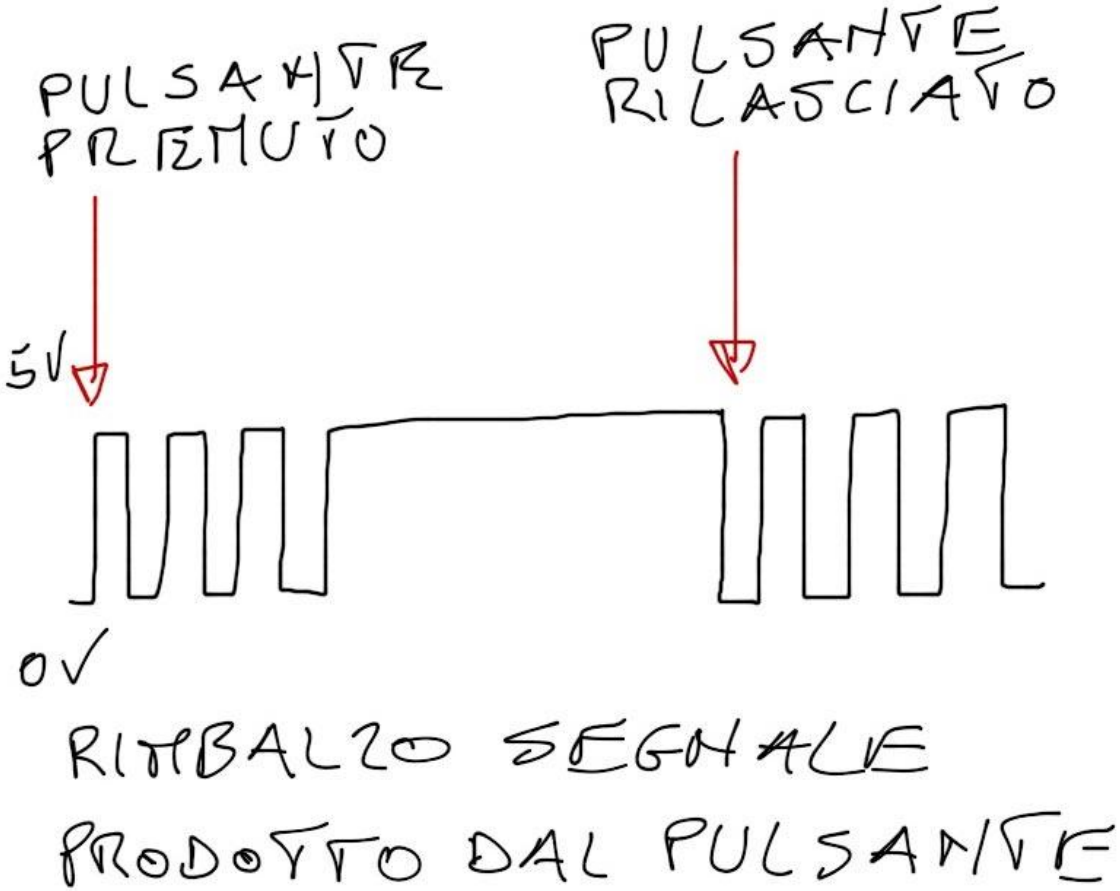
Come possiamo invertire questo comportamento?

Il LED deve essere normalmente spento alla pressione del pulsante si accende.

Rimbalzo del segnale elettrico prodotto dal pulsante

Non appena aumentiamo la frequenza con cui viene premuto il pulsante, si cade in una situazione di incongruenza per cui il LED non risponde più ai comandi. Questo problema avviene perché il pulsante è un apparato meccanico costituito da contatti elettrici ed una molla, quando premiamo e rilasciamo, si manifestano situazioni di rimbalzo del contatto che creano dei segnali non corretti, detti spuri, che modificano lo stato del diodo LED.

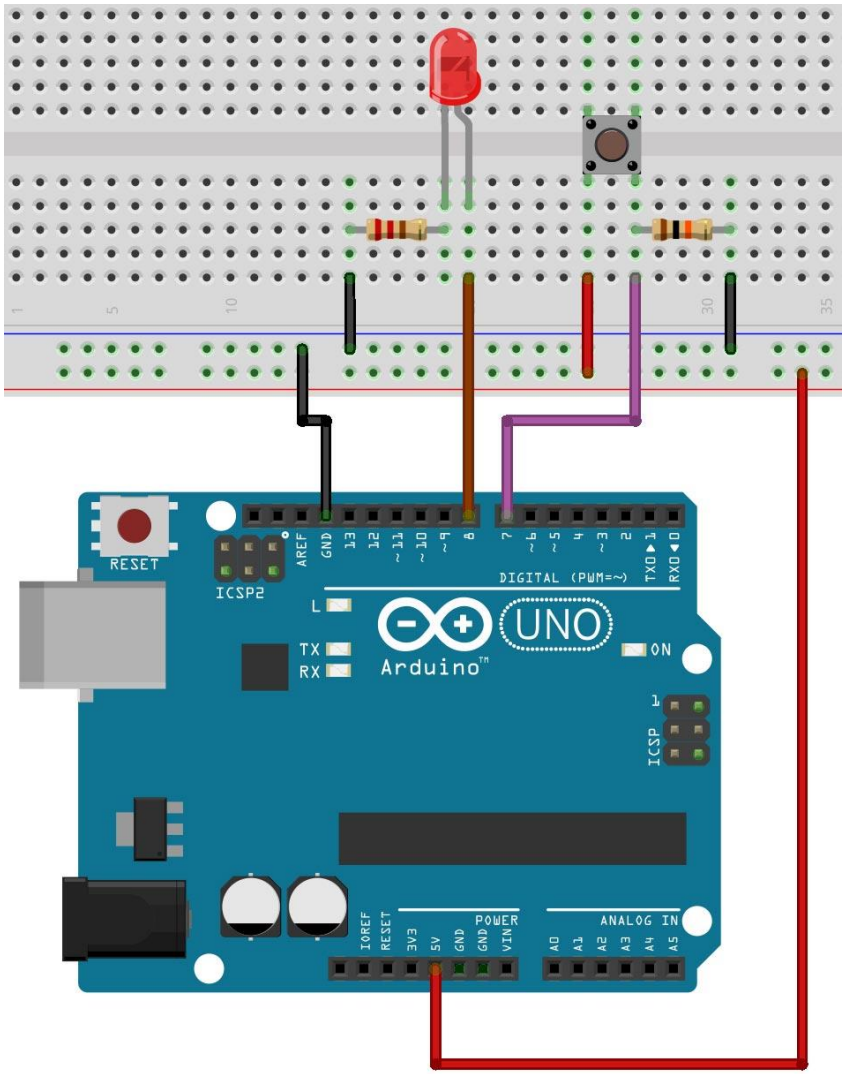
Per approfondimenti alla soluzione del problema del rimbalzo si consulti il seguente link.



antirimbalzo | 2/3 | sketch10

Realizziamo un sistema di antirimbalzo.
Si realizzi il circuito rappresentato in figura.

Diversamente dal circuito precedente **non si usa la resistenza di pull-up interna di Arduino** (si usi una resistenza in serie al pulsante).



antirimbalzo | 3/3 | sketch10

```
/*
  Prof. Michele Maffucci
  08.03.2014

   Antirimbalzo
   pulsante collegato al pin 7 LED al pin 8
   La logica di antirimbalzo impedisce il fraintendimento
   dello stato del pulsante.

   Questo codice è di dominio pubblico
*/

const int LED = 8;                    // LED collegato al pin digitale 8
const int BUTTON = 7;                 // pin di input dove è collegato il pulsante
const int ritardoRimbalzo = 10;       // millisecondi di attesa fino alla stabilità

// antirimbalzo restituisce lo stato quando l'interruttore è stabile
boolean antirimbalzo(int pin)
{
  boolean stato;
  boolean precedenteStato;

  precedenteStato = digitalRead(pin); // memorizza lo statoprecedente
  for(int contatore=0; contatore < ritardoRimbalzo; contatore++)
  {
      delay(1);                       // attende per 1 millisecondo
      stato = digitalRead(pin);       // legge il pin
      if( stato != precedenteStato)
      {
         contatore = 0;               // resetta il contatore se lo stato e cambiato
         precedenteStato = stato;     // salva lo stato corrente
      }
  }
  // a questo punto lo stato è stabile e viene restituito
  return stato;
}

void setup()
{
  pinMode(BUTTON, INPUT);
  pinMode(LED, OUTPUT);
}
```

or(int contatore=0; contatore < itardoRimbalzo; contatore++)...

iene verificato lo stato del pulsante in un ntervallo di 10 millisecondi.
Se lo stato precedente è diverso da quello ttuale viene effettuato un nuovo controllo di urata 10 millisecondi.

Modulazione di larghezza di impulso (PWM)

Prof. Michele Maffucci

Duty cycle

Molto spesso si ha la necessità di controllare la luminosità di LED o la velocità di motori elettrici.

Ma variazioni di questo genere sono di tipo analogico.
Come si effettua un controllo di tipo analogico con un dispositivo come Arduino in grado di gestire in output grandezze di tipo digitali?

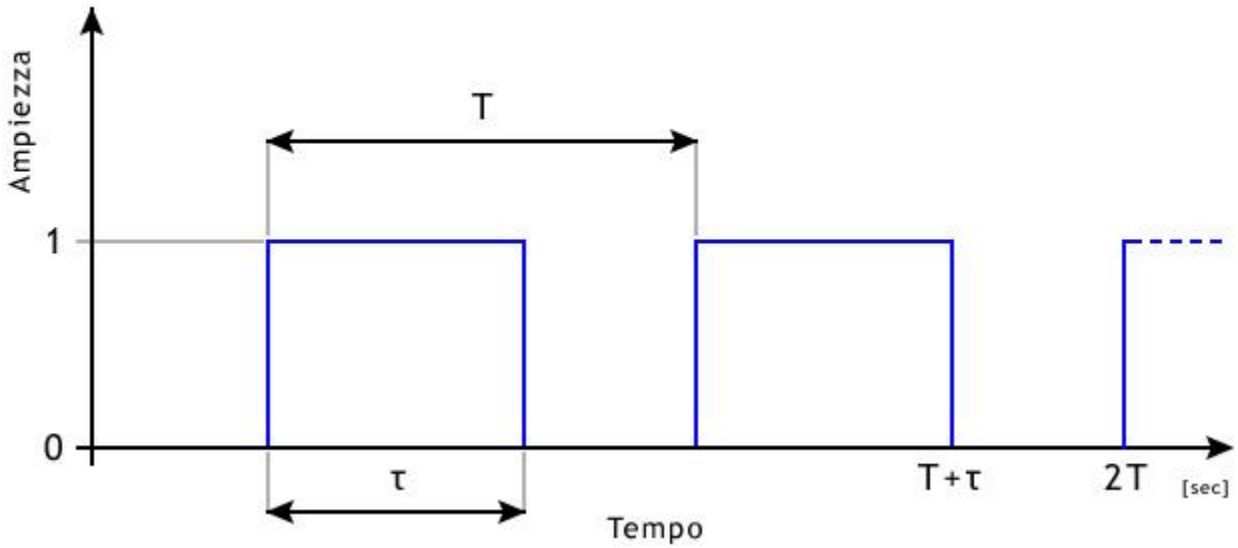
Per effettuare questa operazione Arduino usa la **modulazione di larghezza di impulso in inglese Pulse Width Modulation – PWM** sfruttando opportuni pin digitali.

Nel caso del controllo della luminosità di un LED
Per controllare la luminosità di un LED lo si fa lampeggiare ad una frequenza elevato, tanto da non far percepire all'occhio umano gli istanti in cui il LED è acceso e gli istanti in cui è spento, inoltre a secondo del rapporto del tempo di accensione e spegnimento potremo regolare la luminosità del LED.

Per i dettagli teorici e pratici sul PWM con esempi ed esercizi si segua il link.

Duty cycle

Il **duty cycle** di un onda quadra/rettangolare e il rapporto tra la durata (in secondi) del segnale quando è "**alto**" ed il **periodo totale** del segnale. In altre parole è un numero che esprime quant'è la parte di periodo in cui il segnale è alto.



Facendo riferimento al disegno la formula che esprime il duty cycle è:

$$d=\frac{\tau}{T}$$

Dove **T** è il periodo e **τ** la parte di periodo in cui il segnale è alto.

Duty cycle

Dalla formula si nota che il duty cycle può variare da un valore minimo di 0 a un valore massimo pari a **T**, ciò implica che il valore del duty cycle varia da 0 a 1:

$$d=\frac{\tau}{T}$$

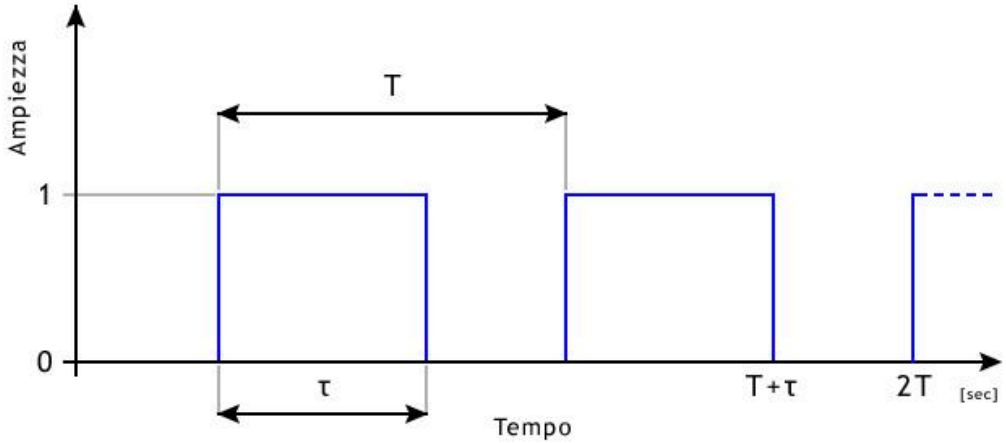
Caso 1: per $\boldsymbol{\tau = 0}$

$$\frac{0}{T}=0$$ livello basso per tutto il periodo

Caso 2: per $\boldsymbol{\tau = T}$

$$\frac{T}{T}=1$$ livello alto per tutto il periodo

in entrambi i casi siamo in presenza di segnali continui.

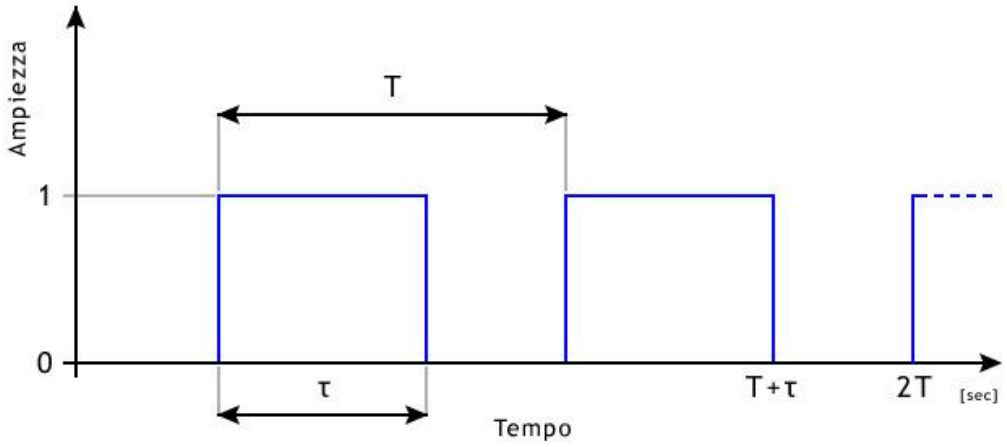


Duty cycle

Dalla formula possiamo comprendere quindi che il duty cycle è sempre un valore che varia tra 0 e 1.

Il duty cycle è spesso rappresentato in percentuale, D% e per ottenere la percentuale è sufficiente moltiplicare per 100 il rapporto τ/T, dire quindi che il D%=30% vuol dire che per il 30% del periodo totale il segnale si trova a livello alto, come conseguenza possiamo subito dire che il segnale sarà a livello basso per il restante 70% del periodo.

Dire quindi che il duty cycle è del 50% vuol dire che nel periodo T il segnale si mantiene alto per $T/2$ e per il restante $T/2$ a livello basso in questo caso siamo quindi in presenza di un'onda quadra.



analogWrite(pin, valore)

Arduino UNO offre la possibilità di usare i pin **3, 5, 6, 9, 10, 11** l'istruzione: **analogWrite()**, istruzione che permette di ottenere risultati analogici su pin digitali in altro modo poter usare il PWM.

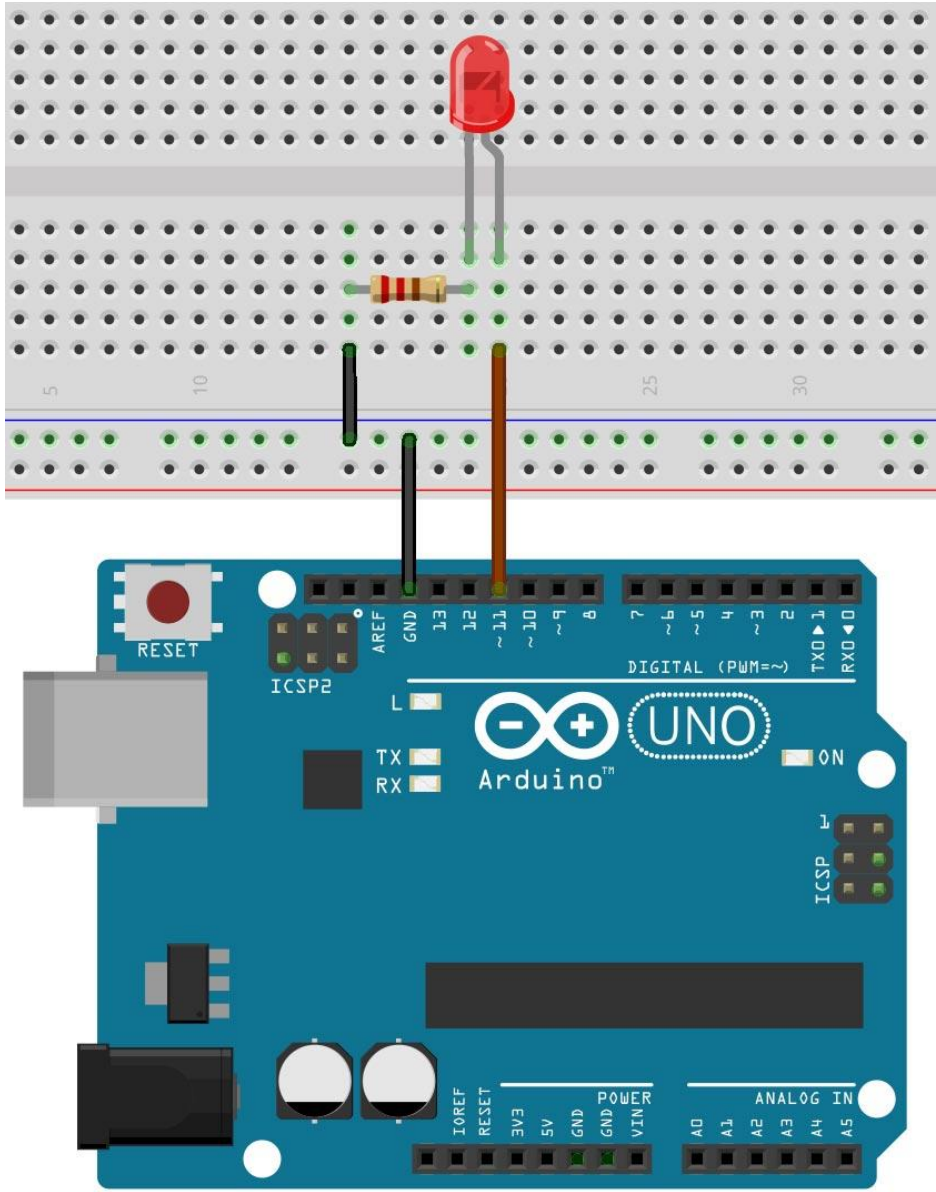
Sintassi:
analogWrite(pin, valore)

dove:

- **pin**: è il piedino su cui inviamo il segnale, per Arduino UNO i pin 3, 5, 6, 9, 10, 11
- **valore**: è il duty cycle compreso tra 0 (sempre off) a 255 (sempre on)

La funzione non restituisce nessun valore.

Realizziamo il seguente circuito riportato nell' immagine collegando il diodo LED sul pin 11. Il valore della resistenza è di 220 Ohm



analogWrite(pin, valore) 2/8 sketch11

analogWrite(11, 0)
il LED collegato al pin 11 avrà una luminosità dello 0% (duty cycle 0%)

```
/* Prof. Michele Maffucci
   08.03.2014

   Utilizzo della funzione analgoWrite() - (duty cycle 0%)

   Questo codice è di dominio pubblico
*/

#define LED 11            // LED collegato al pin digitale 11

void setup() {
  pinMode(LED, OUTPUT);  // imposta il pin digitale come output
}

void loop() {
  analogWrite(LED, 0);   // LED spento
}
```

analogWrite(11, 64)
il LED collegato al pin 11 avrà una luminosità del 25% (duty cycle 25%)

```
/* Prof. Michele Maffucci
   08.03.2014

   Utilizzo della funzione analgoWrite() - (duty cycle 25%)

   Questo codice è di dominio pubblico
*/

#define LED 11            // LED collegato al pin digitale 11

void setup() {
  pinMode(LED, OUTPUT);  // imposta il pin digitale come output
}

void loop() {
  analogWrite(LED, 64);   // accende il LED
}
```

analogWrite(pin, valore) 4/8 sketch13

analogWrite(11, 128)
il LED collegato al pin 11 avrà una luminosità del 50% (duty cycle 50%)

```
/* Prof. Michele Maffucci
   08.03.2014

   Utilizzo della funzione analgoWrite() - (duty cycle 50%)

   Questo codice è di dominio pubblico
*/

#define LED 11            // LED collegato al pin digitale 11

void setup() {
  pinMode(LED, OUTPUT);  // imposta il pin digitale come output
}

void loop() {
  analogWrite(LED, 128); // accende il LED
}
```

analogWrite(11, 191)
il LED collegato al pin 11 avrà una luminosità del 75% (duty cycle 75%)

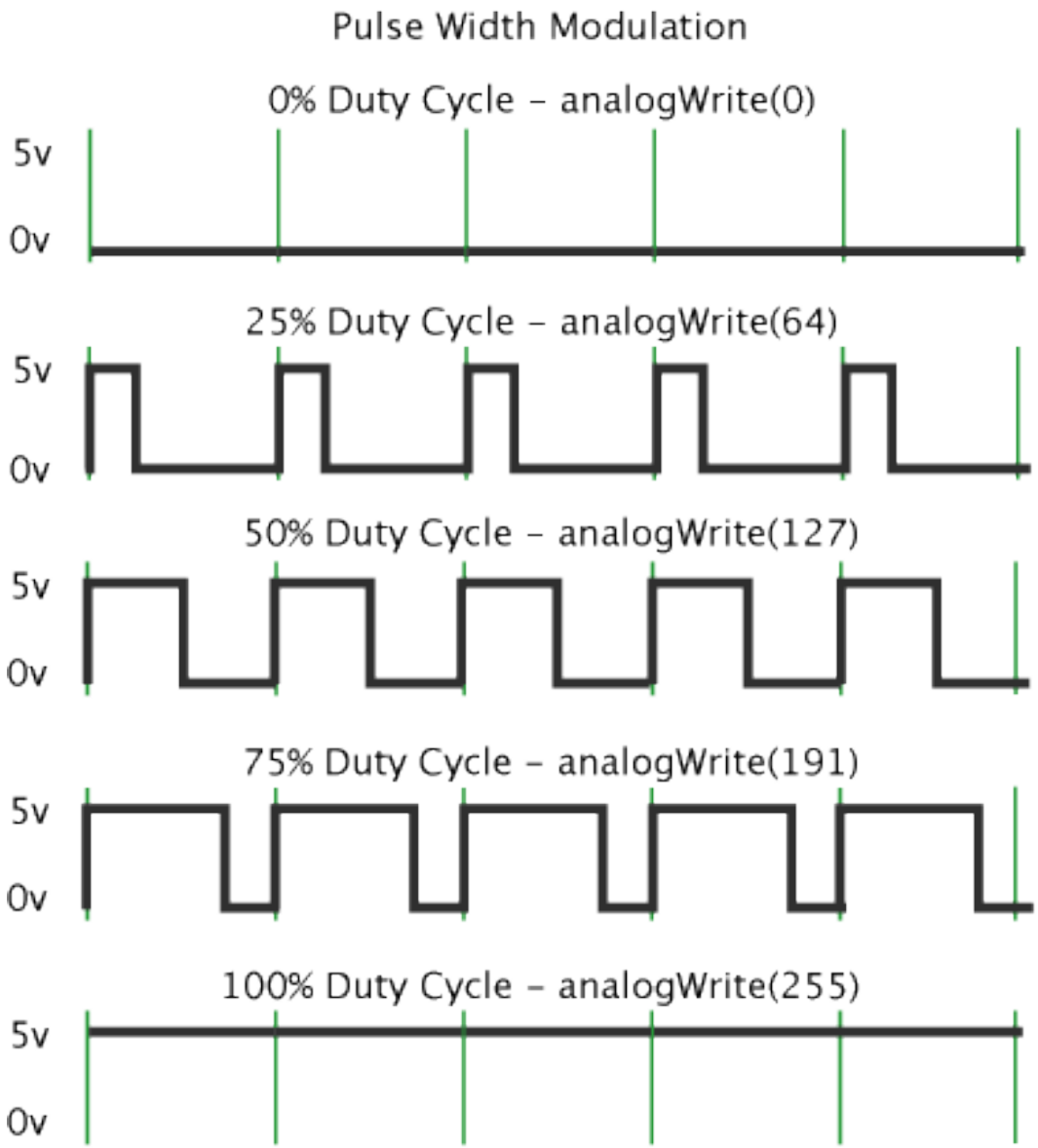
```
/* Prof. Michele Maffucci
   08.03.2014

   Utilizzo della funzione analgoWrite() - (duty cycle 75%)

   Questo codice è di dominio pubblico
*/

#define LED 11            // LED collegato al pin digitale 11

void setup() {
  pinMode(LED, OUTPUT);  // imposta il pin digitale come output
}

void loop() {
  analogWrite(LED, 191); // accende il LED
}
```

analogWrite(11, 255)
il LED collegato al pin 11 avrà una luminosità del 100% (duty cycle 100%)

```
/* Prof. Michele Maffucci
   08.03.2014

   Utilizzo della funzione analgoWrite() - (duty cycle 100%)

   Questo codice è di dominio pubblico
*/

#define LED 11            // LED collegato al pin digitale 11

void setup() {
  pinMode(LED, OUTPUT);  // imposta il pin digitale come output
}

void loop() {
  analogWrite(LED, 255); // accende il LED
}
```

analogWrite(pin, valore)

Utilizzando uno degli sketch precedenti (dall'11 al 15) variare il secondo parametro della digitalWrite() per valori di: 5, 10, 15, 20, 25 dovreste notare ancor di più la variazione di luminosità del LED.

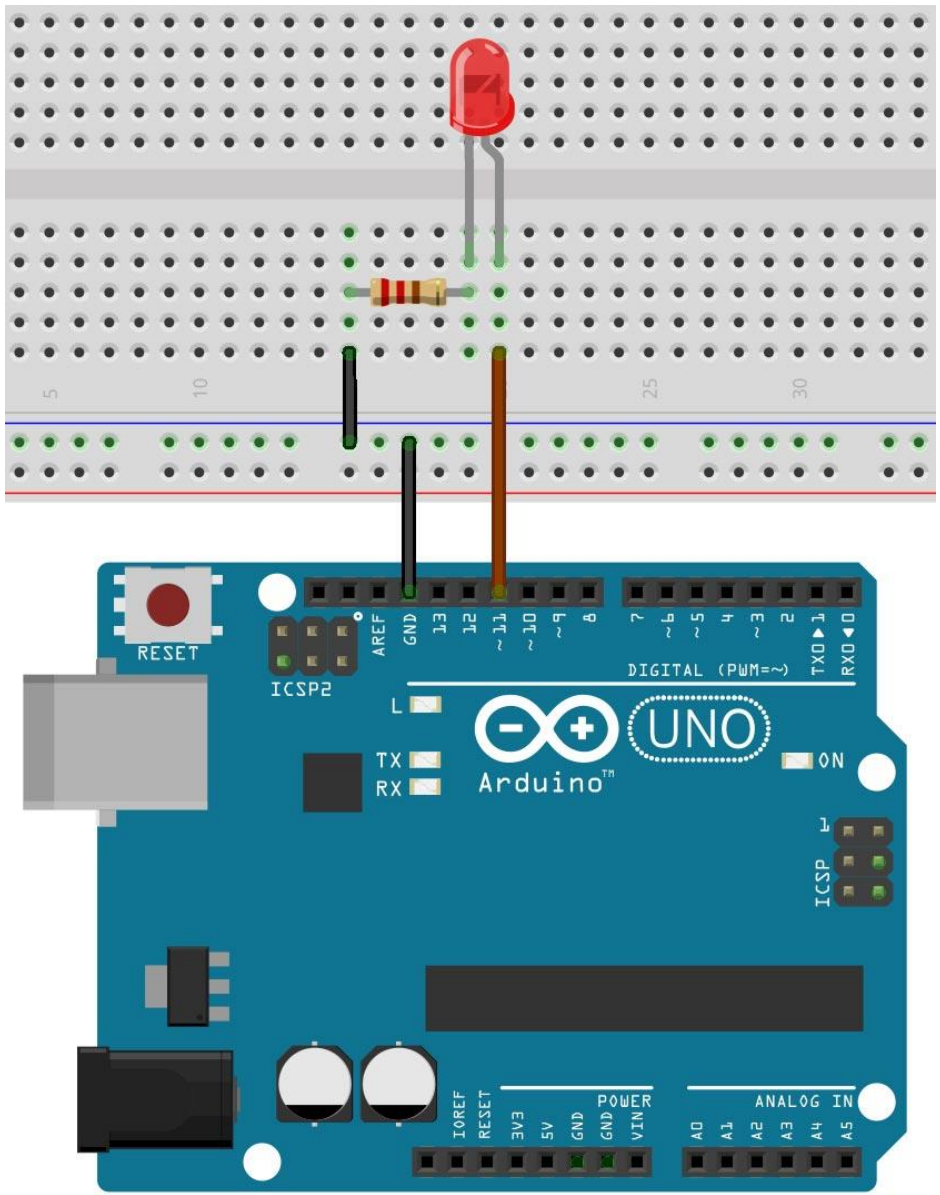


fade | 1/4 | sketch16

Utilizzando lo stesso circuito che abbiamo usato negli esempi precedenti, vediamo come realizzare il fade del LED ovvero l'accensione e lo spegnimento graduale, attenzione che questo modo di procedere sarà utile anche quando dovremo imparare a variare la velocità di un motorino elettrico.

Lo schetch allegato è già presente negli esempi disponibili dal menù:

File -> Examples -> 3.Analog -> Fading



fade 2/4 sketch16

```
/* Prof. Michele Maffucci
   08.03.2014

   Fading

   Questo codice è di dominio pubblico
*/

#define LED 11              // LED collegato al pin digitale 11
int valoreFade = 0;         // variabile usata per contare in avanti e indietro

void setup()  {
  pinMode(LED, OUTPUT);     // imposta il pin digitale come output
}

void loop()  {
  // procede ciclicamente da 0 a 254 (fade in -> aumento luminosità)
  for (valoreFade = 0 ; valoreFade < 255; valoreFade++) {
    analogWrite(LED, valoreFade);      //impostiamo la luminosità del LED
    delay(10);
    // aspettiamo 10ms per percepire la viariazione di luminosità,
    //perché analogWrite è istantaneo
  }
  // procede ciclicamente da 255 a 1 (fade out -> diminuzione della luminosità)
  for(valoreFade = 255 ; valoreFade > 0; valoreFade--) {
    analogWrite(LED, valoreFade);      //impostiamo la luminosità del LED
    delay(10);
    // aspettiamo 10ms per percepire la viariazione di luminosità,
    //perché analogWrite è istantaneo
  }
}
```

fade | 3/4 | sketch16

```
/* Prof. Michele Maffucci
   08.03.2014

   Fading

   Questo codice è di dominio pubblico
*/

#define LED 11              // LED collegato al pin digitale 11
int valoreFade = 0;         // variabile usata per contare in d

void setup()  {
  pinMode(LED, OUTPUT);     // imposta il pin digitale come out
}

void loop()  {
  // procede ciclicamente da 0 a 254 (fade in -> aumento lumino
  for (valoreFade = 0 ; valoreFade < 255; valoreFade++) {
    analogWrite(LED, valoreFade);      //impostiamo la luminosi
    delay(10);
    // aspettiamo 10ms per percepire la viariazione di luminosi
    //perché analogWrite è istantaneo
  }
  // procede ciclicamente da 255 a 1 (fade out -> diminuzione d
  for(valoreFade = 255 ; valoreFade > 0; valoreFade--) {
    analogWrite(LED, valoreFade);      //impostiamo la luminosi
    delay(10);
    // aspettiamo 10ms per percepire la viariazione di luminosi
    //perché analogWrite è istantaneo
  }
}
```

Ad ogni ciclo incrementiamo la variabile **valoreFade** di 1 (valoreFade++) partendo da 0 fino a 254.

valoreFade viene utilizzata in

analogWrite(LED, valoreFade)

per variare il valore del **duty cycle** ed incrementare la luminosità del LED. Poichè l'azione di analogWrite(LED, valoreFade) è immediata per percepire visivamente la variazione di luminosità introduciamo un piccolo ritardo di 10ms con delay(10).

Il ciclo terminerà non appena la condizione **valoreFade < 255** non è più vera, cioè **valoreFade non più minore di 255**.

fade | 4/4 | sketch16

```
/* Prof. Michele Maffucci
   08.03.2014

   Fading

   Questo codice è di dominio pubblico
*/

#define LED 11              // LED collegato al pin digitale 11
int valoreFade = 0;         // variabile usata per contare in avanti e indietro

void setup()  {
  pinMode(LED, OUTPUT);     // imposta il pin digitale come output
}

void loop()  {
  // procede ciclicamente da 0 a 254 (fade in -> aumento lumin
  for (valoreFade = 0 ; valoreFade < 255; valoreFade++) {
    analogWrite(LED, valoreFade);      //impostiamo la luminos
    delay(10);
    // aspettiamo 10ms per percepire la viariazione di luminos
    //perché analogWrite è istantaneo
  }
  // procede ciclicamente da 255 a 1 (fade out -> diminuzione
  for(valoreFade = 255 ; valoreFade > 0; valoreFade--) {
    analogWrite(LED, valoreFade);      //impostiamo la luminos
    delay(10);
    // aspettiamo 10ms per percepire la viariazione di luminos
    //perché analogWrite è istantaneo
  }
}
```

Ad ogni ciclo la variabie **valoreFade** viene decrementata di 1 (valoreFade--) facendo decrescere valoreFade da 255 a 1 e di conseguenza la luminosità del LED.

Il ciclo terminerà quando la condizione valoreFade > 0 non è più vera.

Usciti da questo secondo ciclo si ripartirà con il primo ciclo **for** che permetterà nuovamente l'aumento della luminosità del LED.

esercizio | sketch17

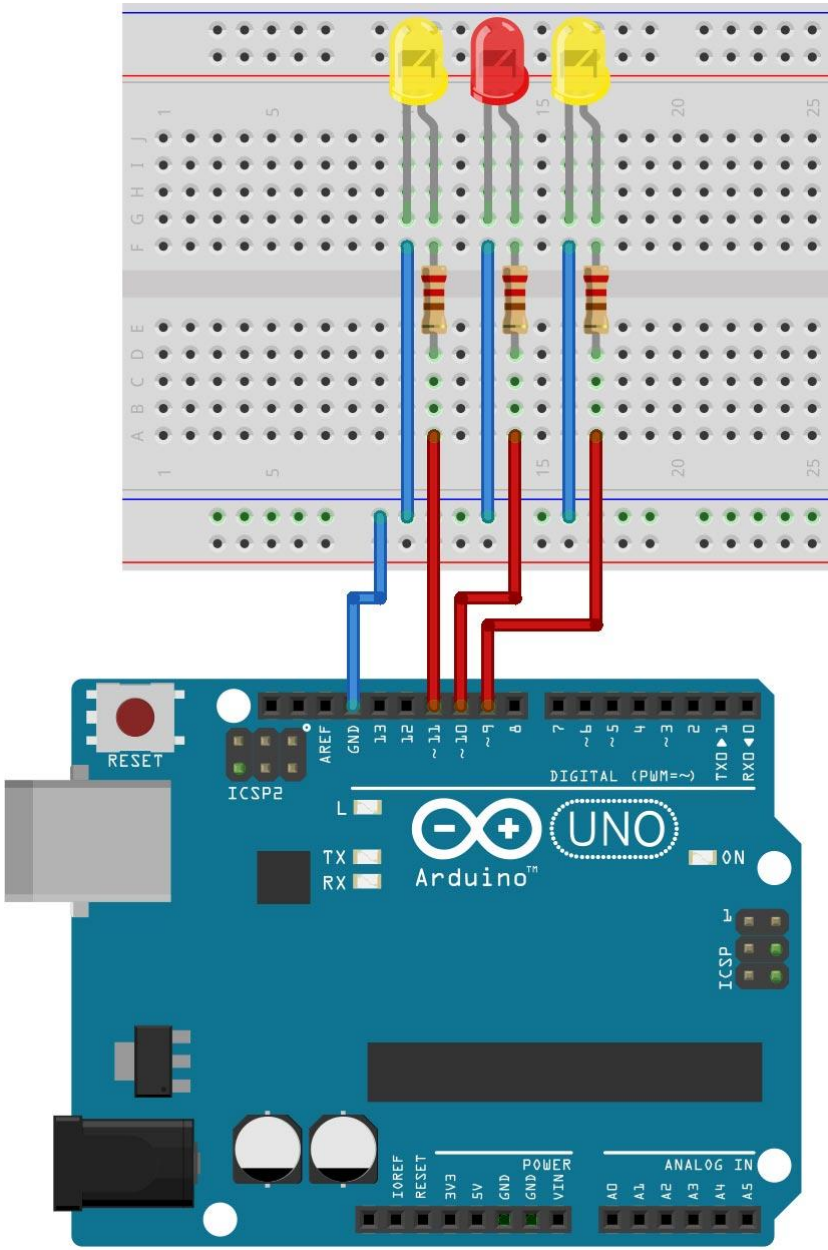
Esercizio 17

Realizzazione dell'effetto fuoco.

Questo esercizio ha lo scopo di utilizzare le uscite PWM e mostrare come generare numeri casuali con Arduino. Si consiglia di prelevare lo sketch e consultare i link che rimandano alle lezioni on-line.

- Breadboard
- n. 2 LED gialli
- n. 1 LED rosso
- n. 3 resistenze da 220Ω
- cavi di connessione

Costruite il circuito utilizzando i **pin digitali 9, 10, 11** ricordate di collegare i **catodi dei LED a GND**.



Grazie

Prof. Michele Maffucci

www.maffucci.it
michele@maffucci.it

www.twitter.com/maffucci/
www.facebook.com/maffucci.it/
plus.google.com/+MicheleMaffucci/

it.linkedin.com/in/maffucci