FAB LAB FROSINONE / 29-30 TH MARCH 2014

Arduino Basic Workshop

Nozioni fondamentali

ARDUINO DAY 2014

FAB LAB FROSINONE
OFFICINE GIARDINO

“Con **Physical Computing** si intende la realizzazione di ambienti FISICI e INTERATTIVI utilizzando hardware e software in grado di PERCEPIRE quello che avviene nell’ambiente circostante e di REAGIRE di conseguenza”.

Maietta / Aliverti, Il manuale del maker

Le informazioni sono ovunque e oggi possono essere tradotte in un segnale di informazione elettrica.

In ogni ogetto c'è un computer. Un cervello elettronico che interpreta le informazioni.

Gli oggetti non sono più statici e inermi, ma possono avere un comportamento guidato dal contesto.

Pinokio è una versione reale della lampada della Pixar, esplora le potenzialità comportamentali ed espressive della robotica.

Source: http://vimeo.com/53476316

NEST è un termostato che autoapprende le abitudini domestiche e imposta automaticamente temperature e comportamento. La società è stata recentemente acquistata da Google.

Source: https://nest.com/thermostat

Responsive Facade System è un rivestimento interattivo, in grado di modificarsi in base alla quantità di luce che colpisce la facciata.

Source: http://vimeo.com/61911111

“In ogni oggetto si può nascondere un piccolo computer in grado di interagire con noi o con gli altri oggetti. L’estensione di questi meccanismi di INTERAZIONE ATTRAVERSO INTERNET ci porta al concetto di **Internet of Things**”.

Maietta / Aliverti, Il manuale del maker

Little Printer è una mini stampante connessa ad internet. Può essere programmata attraverso un'app e stampa sms, foto o le notizie più importanti prese da un feed stabilito dall'utente.

Source: http://littleprinter.com/

Jawbone è un sistema che monitora come dormi, ti muovi e mangi e fornisce approfondimenti che ti aiutano a prendere decisioni per sentirti al meglio.

Source: https://jawbone.com/

Smart Citizen è un sistema di monitoraggio ambientale in crowd. Ovvero il sistema è direttamente collegato alla rete e mette a sistema i dati provenienti da ogni songola Smart Citizen distribuita sul territorio.

Source: http://www.smartcitizen.me/

Community

Tinkering

Hardware

Software

“**Arduino is** an OPEN-SOURCE electronics PROTOTYPING platform based on flexible, easy-to-use HARDWARE and SOFTWARE. It’s intended for artists, designers, hobbysts and anyone interested in creating interactive objects or environments”.

Definizione ufficiale, arduino.cc

Hardware

Arduino è un piccolo computer, che può essre collegato a dei **sensori**, da cui ottiene informazioni e che può mettere in funzione degli **attuatori**, ovvero altri meccanismi che compiono delle azioni.

ANALIZZA
(Arduino)

Il modo in cui Arduino analizza e interpreta i dati provenienti dai sensori e/o decide come gli attuatori devono agire è detto: **comportamento**.

REAGISCE
(Attuatori)

PERCEPISCE
(Sensori)

Esiste un'intera gamma di **board** Arduino le quali differiscono per funzioni e dimensione. Inoltre esistono le **shield**, circuiti aggiuntivi che estendono le funzionalità della board di partenza.

BOARD

Uno

Yun

Due

LilyPad

Nano

SHIELD

Ethernet

WiFi

Touch Screen

GSM

Per questo corso useremo Arduino Uno.

Data Sheet

Microcontroller	ATmega328
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limits)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
Analog Input Pins	6
DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (ATmega328) of which 0.5 KB used by bootloader
SRAM	2 KB (ATmega328)
EEPROM	1 KB (ATmega328)
Clock Speed	16 MHz

Software

Il secondo pilastro di arduino è l'ambiente software dotato di una **IDE per la programmazione** e un **linguaggio semplificato**.

L'ambiente di programmazione è ciò che permette di stabilire il comportamento di arduino rispetto ad un determinato contesto.

Il punto di forza sta nel fatto che è possibile **ottenere comportamenti diversi mantenendo lo stesso circuito di partenza**.

Si scrive uno **sketch**, si Verifica che sia corretto e si clicca su **Carica** per fare l'upload delle istruzioni su Arduino.

Verifica / Carica

Nuovo / Apri / Salva

Sketch

Status area

L'ambiente è una derivazione dell'ambiente di Processing e il linguaggio una derivazione del C.

Community

Arduino è totalmente Open Source, questo ha dato vita ad una community ricchissima dove poter trovare informazioni, codice, soluzioni, progetti a cui partecipare.

I luoghi più importanti della community sono il forum e il wiki Arduino Playground.

> **“Tinkering**: un modo di agire e di definire le persone che intraprendono un percorso sul sentiero dell’ESPLORAZIONE.
> Si cerca di fare qualcosa, che non si sa bene come fare, guidati dall’immaginazione e dalla curiosità.
> Quando si agisce in questo modo non esiste un modo giusto e uno sbagliato di fare le cose. Si tratta di scoprire come funzionano le cose e manipolarle”.
>
> *Massimo Banzi*, Getting Started With Arduino

Struttura della materia

"Strappando" tramite una forza (tensione elettrica) gli elettroni al conduttore metallico genero un flusso degli stessi che prende il nome di "corrente elettrica".

Legge di Ohm: V = R x I

Unità di misura:
V: **V** (volt) – I: **A** (ampere) - R: **Ω** (ohm)

Circuiti elettrici

"Un insieme di uno o più generatori elettrici collegati tra di loro e ai dispositivi utilizzatori di energia elettrica"

Circuito elettrico

Generatore di tensione (alimentatore)

Resistenza elettrica (simbolo più utilizzato in letteratura)

Batteria (o accumulatore)

Resistenza elettrica (simbolo più utilizzato dai CAD e web)

Equazione ai nodi
(1° legge di Kirchhoff)

La somma algebrica delle correnti entranti ed uscenti da un nodo è pari a 0:

I1 - I2 + I3 = 0 A

Equazione di maglia
(2° legge di Kirchhoff)

La somma algebrica delle d.d.p. in un maglia è pari a 0:

E1 - R1•I1 - R2•I2 = 0

Sensori ed attuatori elettrici

Sensore elettrico:
dispositivo che trasforma una grandezza fisica in un segnale elettrico

Attuatore elettrico:
dispositivo che trasforma un segnale elettrico in “movimento”, come ad esempio i motori elettrici, relè, etc.

Sensore di temperatura (NTC)

Sensore di pressione

Sensore di luminosità (fotoresistore)

Sensore di movimento

Motore elettrico (pompa idraulica)

Servomotore (modellismo)

Relè (commutazioni di potenza)

Motore “passo-passo” o “stepper” (stampanti, scanner, stampanti 3D)

Il cuore della filosofia Arduino è la prototipazione. Si basa sul concetto di creare progetti anziché limitarsi a parlarne. Quindi, cominciamo!

Photo credits CC: Play baking on a rainy day di victoire_za

Esercizio 1

Facciamo lampeggiare il led L.
Ci chiederemo come abbiamo fatto in un secondo momento.

Il **LED L** è integrato direttamente sulla board arduino.

E' direttamente **collegato al PIN 13** ed è già provvisto di una resistenza.

Prendete il cavo USB e collegate Arduino al vostro computer.

```
1  /*
2   Questo è un commento su più righe.
3   Tutto il testo inserito fra questi due comandi, viene completamente
4   Ignorato da Arduino.
5  */
6
7
8  // Questo invece è un commento su una sola riga
9
10
11 /*
12  Setup e Loop sono le funzioni principali di Arduino
13  Dentro setup imposto tutte le condizioni iniziali che vanno
14  impostate una volta Sola.
15 */
16
17
18 void setup() {
19   pinMode(13, OUTPUT);      //Dico ad A. che il PIN 13 funziona come OUTPUT
20 }
21
22 /*
23  La funzione loop è il luogo in cui inserisco tutte le istruzioni
24  che arduino deve eseguire. Le istruzioni vengono eseguite dalla cima al fondo
25  e poi vengono ripetute di nuovo, all'infinito.
26 */
27
28 void loop() {
29   digitalWrite(13, HIGH);   // Accende il led al PIN 13
30   delay(1000);              // Aspetta 1 secondo
31   digitalWrite(13, LOW);    // Spegne il led al PIN 13
32   delay(1000);              // Aspetta un secondo
33 }
34
```

Aprite l'IDE arduino
e copiate questo codice.
Per ora **solo il testo colorato**.

Sketch è il nome con cui chiamiamo un programma eseguito da Arduino.

Un programma è un insieme di istruzioni che vengono eseguite sequenzialmente dal microcontrollore per risolvere un problema o eseguire un compito.

```
1  /*
2    Questo è un commento su più righe.
3    Tutto il testo inserito fra questi due comandi, viene completamente
4    Ignorato da Arduino.
5   */
6
7
8  // Questo invece è un commento su una sola riga
9
10
11 /*
12   Setup e Loop sono le funzioni principali di Arduino
13   Dentro setup imposto tutte le condizioni iniziali che vanno
14   impostate una volta Sola.
15  */
16
17
18 void setup() {
19   pinMode(13, OUTPUT);
20 }
21
22
23
24
25
26
27
28 void loop() {
29   digitalWrite(13, HIGH);
30   delay(1000);
31   digitalWrite(13, LOW);
32   delay(1000);
33 }
34
```

Istruzione che contiene un'altra istruzione

Istruzione

Istruzione che contiene un'altra istruzione

Istruzioni

Istruzione è la singola riga dello sketch che abbiamo appena scritto e che dice ad Arduino di fare qualcosa.

Nei linguaggi di programmazione, a differenza dei linguaggi naturali, una frase sintatticamente scorretta non ha significato.

Operatore aritmetico

Fine istruzione

Espressione

```
9 + 2 + b + c ;
```

Parola Chiave

Operatore di assegnazione

Assegnazione

```
somma = 9 + 2 ;
```

Operatore di confronto

Confronto

```
somma == 11 ;
```

Blocco di istruzioni

```
{
    istruzione1;
    istruzione2;
}
```

Delimitatore blocco

Quali sono gli operatori e quale è il loro significato. Guardiamo alcuni esempi.

Dati due valori x = 5 e y = 10

OP. ARITMETICI
Gli operatori aritmetici permettono di effettuare le operazioni classiche di somma, prodotto, sottrazione e divisione.

```
x = y        // x = 10
x + y        // 15
x - y        // -5
x * y        // 50
y / x        // 2
```

OP. CONFRONTO
Gli operatori di confronto, mettono a confronto un elemento con un altro e il risultato del confronto è sempre una risposta vera o falsa.

```
x == y       // false
x != y       // true
x < y        // true
x > y        // false
x <= y       // true
x >= y       // false
```

OB. BOOLEANI
Gli operatori booleani permettono di combinare insieme più operazioni di confronto. Il risultato è sempre una risposta vera o falsa.

```
x < y && x > y    // false
x < y || x > y    // true
!(x < y)          // false
```

L’ordine di esecuzione delle istruzioni si chiama **Flusso di esecuzione**, e viene eseguito dall’alto verso il basso.

Alcuni comandi particolari possono modificare il normale flusso di esecuzione. Questi comandi sono le **funzioni** e le **strutture di controllo**.

Istruzione decisionale

```
if(){} else{}
```

Funzioni

```
nomefunzione(){}
```

Cicli

```
for(){}
while(){}
```

Ci occuperemo di queste istruzioni nel corso del workshop, ma è necessario capire il concetto per introdurre le funzioni principali di uno sketch.

```
1  /*
2    Questo è un commento su più righe.
3    Tutto il testo inserito fra questi due comandi, viene completamente
4    Ignorato da Arduino.
5   */
6
7
8  // Questo invece è un commento su una sola riga
9
10
11 /*
12   Setup e Loop sono le funzioni principali di Arduino
13   Dentro setup imposto tutte le condizioni iniziali che vanno
14   impostate una volta Sola.
15  */
16
17
18 void setup() {
19   pinMode(13, OUTPUT);
20 }
21
22
23
24
25
26
27
28 void loop() {
29   digitalWrite(13, HIGH);
30   delay(1000);
31   digitalWrite(13, LOW);
32   delay(1000);
33 }
34
```

Eseguito una sola volta, contiene le istruzioni iniziali

Eseguito ripetutamente finché Arduino è acceso, contiene il comportamento fondamentale dello sketch

Le due **funzioni fondamentali** di uno sketch Arduino sono **setup** e **loop**. Devono essere sempre presenti.

setup() e loop() sono le funzioni fondamentali di uno sketch arduino. Arduino esegue prima di tutto le istruzioni contenute in setup, poi esegue le istruzioni contenute in loop, all'infinito.

Lo sketch più semplice possibile è formato dalle funzioni setup e loop vuote.

```
void setup()
{
   ...
}

void loop()
{
   ...
}
```

Esercizio 2

Facciamo lampeggiare un Led esterno, questa volta collegato al PIN 9

Breadboard (o basetta sperimentale) è uno strumento utilizzato per creare prototipi di circuiti elettrici in maniera facile e veloce.

Uno dei modelli più comuni di breadboard....

....ed il suo schema di collegamento interno

LED e resistenza di limitazione della corrente. Il diodo LED (acronimo inglese di "Light Emitting Diode") è un dispositivo che sfrutta le proprietà ottiche di alcuni materiali semiconduttori per produrre fotoni...ovvero luce..

Per utilizzare un LED nel modo corretto è necessario **inserire una resistenza di protezione in serie** che riduce la corrente che lo attraversa, per evitare di danneggiarlo.

La corrente di esercizio per pilotare un diodo LED oscilla normalmente tra i 15 e i 20 mA.

La caduta di tensione ai capi del LED varia in funzione del colore e viene chiamata Vf (tensione diretta)

Tutte queste informazioni sono reperibili sul datasheet di ogni diodo LED.

$$R = \frac{V_A - V_f}{i}$$

Codice colori resistori

Resistenza 200Ω

Creiamo un circuito tra il PIN 9 e GND (terra). In questo circuito inseriamo un LED e abbassiamo l'intensità con una resistenza da 200Ω.

```
1   /*
2    Blink
3    Turns on an LED on for one second, then off for one second, repeatedly.
4
5    This example code is in the public domain.
6    */
7
8   /*
9    Questo è un commento su più righe.
10   Tutto il testo inserito fra questi due comandi, viene completamente
11   Ignorato da Arduino.
12   */
13
14  int led = 9;    //Diamo un nome al PIN 9
15
16  /*
17   Setup e Loop sono le funzioni principali di Arduino
18   Dentro setup imposto tutte le condizioni iniziali che vanno
19   impostate una volta Sola.
20   */
21
22
23  void setup() {
24    pinMode(led, OUTPUT);     //Dico ad A. che il PIN 13 funziona come OUTPUT
25  }
26
27  /*
28   La funzione loop è il luogo in cui inserisco tutte le istruzioni
29   che arduino deve eseguire. Le istruzioni vengono eseguite dalla cima al fondo
30   e poi vengono ripetute di nuovo, all'infinito.
31   */
32
33  void loop() {
34    digitalWrite(led, HIGH);  // Accende il led al PIN
35    delay(1000);              // Aspetta 1 secondo
36    digitalWrite(led, LOW);   // Spegne il led al PIN 9
37    delay(1000);              // Aspetta un secondo
38  }
39
```

Il codice non cambia, tranne che per il numero del PIN e per l'uso della Variabile.

```
1  /*
2    Blink
3    Turns on an LED on for one second, then off for one second, repeatedly.
4
5    This example code is in the public domain.
6   */
7
8  /*
9    Questo è un commento su più righe.
10   Tutto il testo inserito fra questi due comandi, viene completamente
11   Ignorato da Arduino.
12  */
13
14 int led = 9;    //Diamo un nome al PIN 9
15
16
17
18
19
20
21
22
23 void setup() {
24   pinMode(led, OUTPUT);
25 }
26
27
28
29
30
31
32
33 void loop() {
34   digitalWrite(led, HIGH);
35   delay(1000);
36   digitalWrite(led, LOW);
37   delay(1000);
38 }
39
```

Variabile. Le variabili sono porzioni di memoria, come cassetti, in cui memorizzo un valore che mi è utile o su cui lavoro durante l'esecuzione dello sketch.

In una variabile memorizzo un valore di un determinato tipo. Nel corso dello sketch, ogni volta che ho bisogno di quel valore o di modificare quel valore, mi basta dire ad arduino di utilizzare la variabile in cui l'ho memorizzato.

Per poter usare una variabile, occorre prima crearla o, meglio, dichiararla.

```
int x = 100;
```

Tipo di dato

Parola chiave

Assegnazione

Valore

Una variabile è, come si è detto, un contenitore che **può contenere valori di un determinato tipo**; vi sono tipi diversi di valori, come interi, reali, caratteri, ecc.; ogni variabile è etichettata con un certo tipo, e può contenere soltanto valori di quel tipo.

Alcuni esempi dei tipi di dato più usati.

```
int x = 100;
float x = 2.576;
boolen x = true;
```

Introduciamo brevemente il concetto di funzione. **Una funzione è un blocco di istruzioni**, a cui diamo un nome.

Una funzione **ci permette di scrivere una serie di operazioni ripetitive una volta sola** e poi, usare la funzione nel programma attraverso il nome che gli abbiamo assegnato, senza dover riscrivere tutte le istruzioni.

Le funzioni, una volta richiamate, **possono restituire un valore o compiere un'azione**. In questo caso è consuetudine chiamarle procedure.

Vediamo rapidamente un esempio di procedura e uno di funzione.

Funzione somma

Tipo del valore restituito

Blocco di istruzioni

```
int somma(a, b){
  int x = a+b;
  return x;
}
```

Fine del flusso della funzione

Procedura accendi led

Blocco di istruzioni

```
void accendi_led(pin){
  digitalWrite(led, HIGH);
  delay(1000);
  digitalWrite(led, LOW);
}
```

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

  This example code is in the public domain.
 */

/*
  Questo è un commento su più righe.
  Tutto il testo inserito fra questi due comandi, viene completamente
  Ignorato da Arduino.
 */

int led = 9;    //Diamo un nome al PIN 9 ------------------
```

All'inizio di uno sketch **ho necessità di dire ad Arduino quali pin utilizzerò e in che modo li utilizzerò**. Utilizzo la funzione specifica pinMode(). Questa funzione è l'esempio classico di istruzione che va inserita nel setup().

Tipo di uso del Pin

```
INPUT
OUTPUT
```

```
pinMode(pin, mode);
```

Funzione

Numero del pin che sto inizializzando

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

  This example code is in the public domain.
 */

/*
  Questo è un commento su più righe.
  Tutto il testo inserito fra questi due comandi, viene completamente
  Ignorato da Arduino.
 */

int led = 9;    //Diamo un nome al PIN 9 ...

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

void loop() {
  digitalWrite(led, HIGH);  // Accende il led al PIN 9
  delay(1000);              // Aspetta 1 secondo
  digitalWrite(led, LOW);   // Spegne il led al PIN 9
  delay(1000);              // Aspetta un secondo
}
```

Funzione **digitalWrite()**, “accende” o “spegne” un pin di Arduino.

Letteralmente **scrive un valore alto o basso su un pin digitale**.
Praticamente vuol dire che attiva o disattiva un pin come un interruttore, inviando al pin il valore di HIGH (generalmente 5v) o quello di LOW (generalmente gnd).

Valore che determina l'azione sul pin

```
HIGH
LOW
```

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

Funzione

Numero del pin che sto azionando

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

  This example code is in the public domain.
 */

/*
  Questo è un commento su più righe.
  Tutto il testo inserito fra questi due comandi, viene completamente
  Ignorato da Arduino.
 */

int led = 9;    //Diamo un nome al PIN 9 ...

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

void loop() {
  digitalWrite(led, HIGH);  // Accende il led al PIN 9
  delay(1000);              // Aspetta 1 secondo
  digitalWrite(led, LOW);   // Spegne il led al PIN 9
  delay(1000);              // Aspetta un secondo
}
```

Funzione **delay()**, sospende l'esecuzione del programma per il tempo stabilito.

Sospende l'esecuzione del programma per la quantità di millisecondi espressi come parametro.

*E' **importante** tenere a mente il fatto che questa funzione sospende l'esecuzione del programma. In seguito vedremo come è possibile sfruttare il fattore temporale senza utilizzare una funzione che causa la sospensione dell'esecuzione.

```
delay(millisecondi);
```

Funzione

Millisecondi di sospensione

```
1  /*
2    Blink
3    Turns on an LED on for one second, then off for one second, repeatedly.
4
5    This example code is in the public domain.
6   */
7
8  /*
9    Questo è un commento su più righe.
10   Tutto il testo inserito fra questi due comandi, viene completamente
11   Ignorato da Arduino.
12  */
13
14 int led = 9;    //Diamo un nome al PIN 9
15
16
17
18
19
20
21
22
23 void setup() {
24   pinMode(led, OUTPUT);     //Dico ad A. che il PIN 13 funziona come OUTPUT
25 }
26
27
28
29
30
31
32
33 void loop() {
34   digitalWrite(led, HIGH);
35   delay(1000);
36   digitalWrite(led, LOW);
37   delay(1000);
38 }
39
```

I **Commenti** servono a spiegare il codice. Arduino non ne tiene conto.

I commenti sono l'istruzione più semplice. Sono utilizzati per rendere più leggibile lo sketch. Tutto ciò che è un commento **non viene considerato dal compilatore** di Arduino.

Commento multiriga

```
/* Questo è un commento multiriga, posso
continuare a scrivere finché lo ritengo
necessario e finché non raggiungo l'istruzione di
fine commento. */
```

Commento su una sola riga

```
x = 5;  // Questo è un commento a singola riga.
        // Questo è un altro.
```

Esercizio 3

Variamo la luminosità del LED al PIN 9, modificando il Delay. Giochiamo con i valori e cerchiamo di capire cosa succede.

```
13
14  int led = 9;  //Diamo un nome al PIN 9
15
16  /*
17    Setup e Loop sono le funzioni principali di Arduino
18    Dentro setup imposto tutte le condizioni iniziali che vanno
19    impostate una volta Sola.
20   */
21
22
23  void setup() {
24    pinMode(led, OUTPUT);   //Dico ad A. che il PIN 13 funziona come OUTPUT
25  }
26
27  /*
28    La funzione loop è il luogo in cui inserisco tutte le istruzioni
29    che arduino deve eseguire. Le istruzioni vengono eseguite dalla cima al fondo
30    e poi vengono ripetute di nuovo, all'infinito.
31   */
32
33  void loop() {
34    digitalWrite(led, HIGH);  // Accende il led
35    delay(1);                 // Aspetta
36    digitalWrite(led, LOW);   // Spegne il led
37    delay(20);                // Aspetta
38  }
39
```

Manteniamo inalterato il circuito e lo sketch. Variamo solo i valori del delay.

Proviamo con valori diversi e molto bassi.

La cosa più evidente, se avete utilizzato bene il delay, è che **il led varia la luminosità** al variare dei valori del ritardo. Questo effetto si ottiene **grazie alla modulazione di larghezza d'impulso** o pwm.

> La modulazione di larghezza di impulso (o **PWM**, acronimo del corrispettivo inglese pulse-width modulation), è un tipo di modulazione digitale che permette di **ottenere una tensione media variabile dipendente dal rapporto tra la durata dell' impulso positivo e di quello negativo**.
>
> *fonte wikipedia*

> La modulazione a larghezza di impulso è largamente utilizzata anche per regolare la potenza elettrica inviata ad un carico, per esempio negli inverter, per regolare la velocità dei motori in corrente continua e per variare la luminosità delle lampadine.
>
> *fonte wikipedia*

Esercizio 4

Variamo la luminosità del LED utilizzando il PIN PWM. Proviamo a creare un effetto Fade.

Vi ricordate l'onda quadra che abbiamo generato giocando con il delay? Bene. Arduino dispone di alcuni pin, detti **pwm** (i pin con la tilde) che **generano direttamente un'onda quadra della larghezza d'impulso desiderata**.

analogWrite() è la funzione che permette di scrivere sui pin PWm. Funziona allo stesso modo di digitalWrite, ma il funzionamento non è acceso, spento, ma una modulazione d'impulso che varia da 0 a 255. Il grafico mostra il valore di analogWrite e l'onda quadra corrispondente.

La sintassi della funzione **analogWrite()**.

Valore di pwm

0...255

```
analogWrite(pin, value);
```

Funzione

Numero del pin che sto utilizzando

Comparison Operators. Gli operatori di comparazione permettono di comparare un elemento con un altro e il risultato dell'espressione è sempre vero o falso.

```
x = 10;
y = 20;

x == y   // false
x != y   // true
x <  y   // true
x >  y   // false
x <= y   // true
x >= y   // false
```

Un'espressione di comparazione viene utilizzata nei costrutti decisionali e nei cicli. Il computer la utilizza come test per prendere delle decisioni.

Introduciamo il primo ciclo. Il ciclo FOR. **Il ciclo for esegue un blocco di istruzioni per un numero definito di volte**. E‘ utile quando sappiamo per quante volte le operazioni devono essere eseguite.

Incremento

Inizializzazione

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

  istruzione 1;
  istruzione 2;
}
```

Controllo: quando i diventa maggiore di 255, l’espressione diventa false e il ciclo si interrompe.

Blocco istruzioni

```
// Inizializiamo le variabili
int LED = 9;

// Settiamo il pin 9 come pin di output
void setup(){
  pinMode(LED, OUTPUT);
}


//creiamo un ciclo
void loop(){

  for(int i = 0; i<255; i++){
    analogWrite(LED, i);    //attivo il pin 9 con un valore di pwm variabile
    delay(15);              //aggiungo un ritardo per vedere l'effetto
  }

  for(int i = 255; i>=0; i--){
    analogWrite(LED, i);    //attivo il pin 9 con un valore di pwm variabile
    delay(15);              //aggiungo un ritardo per vedere l'effetto
  }

  delay(500);               //aggiungo un ritardo finale per simulare il respiro
}
```

Il modo più semplice per creare un fade è utilizzare un ciclo for che imposta valori crescenti e un ciclo for con valori decrescenti per il PIN 9.

```
/*
 Fade

 This example shows how to fade an LED on pin 9
 using the analogWrite() function.

 This example code is in the public domain.
 */

int led = 9;           // the pin that the LED is attached to
int brightness = 0;    // how bright the LED is
int fadeAmount = 5;    // how many points to fade the LED by

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

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

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

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

La soluzione più elegante non usa un ciclo For, ma sfrutta il loop standard di Arduino.

Esercizio Bonus

Utilizziamo un LED RGB e scriviamo due sketch. Uno che mostri i tre colori primari. Uno che faccia assumere al led tutte le sfumature di colore possibili. Il circuito è lo stesso per entrambi gli sketch.

LED rgb (red-green-blue), un particolare diodo LED che contiene al suo interno 3 diodi LED di colore rosso, verde e blu e che possiamo comandare in maniera simultanea o indipendente.

LED RGB

Modulando la “quantità” di illuminazione dei singoli led e sommandole cromaticamente insieme possiamo ottenere il colore desiderato nello spettro di luce visibile.

Abbiamo 3 pin in ingresso, uno per colore e uno solo in uscita.

Usiamo i PIN pwm, per accedere a tutti i 255 valori disponibili.

Il circuito è identico al circuito del singolo LED. La variazione di percentuale del singolo colore sommata agli altri definisce il colore risultante.

```
/*
  Mostra i tre colori primari ripetutamente
*/

// Assegno il numero di PIN ai led che userò
// Assegno un nome di variabile che mi ricordi il colore su cui agisco
int RED_LED_PIN = 11;
int GREEN_LED_PIN = 10;
int BLUE_LED_PIN = 9;


// Creo delle variabili in cui memorizzo il valore del singolo colore
// Questo valore varierà da 0 a 255
int redIntensity = 255;
int greenIntensity = 255;
int blueIntensity = 255;

// Definisco una variabile con il tempo di delay
// In questa maniera posso fare delle prove senza cambiare il valore
// in ogni punto del codice
int DISPLAY_TIME = 1000;


void setup() {
  pinMode(RED_LED_PIN, OUTPUT);
  pinMode(GREEN_LED_PIN, OUTPUT);
  pinMode(BLUE_LED_PIN, OUTPUT);
}

void loop() {

  analogWrite(RED_LED_PIN, redIntensity);
  analogWrite(GREEN_LED_PIN, 0);
  analogWrite(BLUE_LED_PIN, 0);
  delay(DISPLAY_TIME);

  analogWrite(GREEN_LED_PIN, greenIntensity);
  analogWrite(RED_LED_PIN, 0);
  analogWrite(BLUE_LED_PIN, 0);
  delay(DISPLAY_TIME);

  analogWrite(BLUE_LED_PIN, blueIntensity);
  analogWrite(RED_LED_PIN, 0);
  analogWrite(GREEN_LED_PIN, 0);
  delay(DISPLAY_TIME);
}
```

Nel loop ogni blocco di funzione determina un colore. Ogni colore è dato da tutti e tre i valori di Rosso / Verde / Blu.

```
// Assegno il numero di PIN ai led che userò
// Assegno un nome di variabile che mi ricordi il colore su cui agisco
int RED_LED_PIN = 11;
int GREEN_LED_PIN = 10;
int BLUE_LED_PIN = 9;


// Creo delle variabili in cui memorizzo il valore del singolo colore
// Questo valore varierà da 0 a 255
int redIntensity = 0;
int greenIntensity = 0;
int blueIntensity = 0;

// Definisco una variabile con il tempo di delay
// In questa maniera posso fare delle prove senza cambiare il valore
// in ogni punto del codice
int DISPLAY_TIME = 100;


void setup() {
  pinMode(RED_LED_PIN, OUTPUT);
  pinMode(GREEN_LED_PIN, OUTPUT);
  pinMode(BLUE_LED_PIN, OUTPUT);
}

void loop() {
  // Primo ciclo dal ROSSO al VERDE
  // Il ciclo aumenta progressivamente il valore di verde e diminuisce quello di rosso

  for (greenIntensity = 0; greenIntensity <= 255; greenIntensity+=5) {
        redIntensity = 255-greenIntensity;
        analogWrite(GREEN_LED_PIN, greenIntensity);
        analogWrite(RED_LED_PIN, redIntensity);
        delay(DISPLAY_TIME);
  }

  // Secondo ciclo dal VERDE al BLU
  // Il ciclo aumenta progressivamente il valore di blu e diminuisce quello di verde

  for (blueIntensity = 0; blueIntensity <= 255; blueIntensity+=5) {
        greenIntensity = 255-blueIntensity;
        analogWrite(BLUE_LED_PIN, blueIntensity);
        analogWrite(GREEN_LED_PIN, greenIntensity);
        delay(DISPLAY_TIME);
  }

  // Terzo ciclo dal BLU di nuovo al ROSSO
  // Il ciclo aumenta progressivamente il valore di rosso e diminuisce quello di blu

  for (redIntensity = 0; redIntensity <= 255; redIntensity+=5) {
        blueIntensity = 255-redIntensity;
        analogWrite(RED_LED_PIN, redIntensity);
        analogWrite(BLUE_LED_PIN, blueIntensity);
        delay(DISPLAY_TIME);
  }
}
```

Stesso codice di partenza con alcune variazioni. I tre cicli For permettono di scorrere tutta la gamma di colori disponibili.

Esercizio Bonus

Proviamo a mettere in pratica tutto quello che abbiamo imparato fino ad ora, simulando un'onda di luce fra 4 LED. Aggiungiamo qualche difficoltà e una nuova struttura dati.

Vi ricordate le luci colorate di SuperCar? Ecco, proviamo a simulare l’effetto onda di Kit.

Source: http://youtu.be/9rGyGGFiyrg

Usiamo la striscia orizzontale della breadboard per creare un unico canale di terra.

Usiamo i PIN pwm, in questa maniera possiamo simulare un effetto onda con luminosità variabile.

Il circuito è molto semplice. Non è altro che il circuito del singolo LED dell'esercizio 2, ripetuto per 4 volte.

La nuova struttura dati è l'array.
Un **array è una collezioni di variabili**, a cui possiamo accedere attraverso un indice.

Dichiarazione: ognuna delle seguenti è una dichiarazione valida.

```
int myInts[6];
int myPins[] = {2, 4, 8, 3, 6};
int mySensVals[5] = {2, -8, 3, 2};
```

Assegnare un valore a una variabile della collezione.

```
mySensVals[2] = 10;
```

Richiamare il valore di una variabile della collezione.

```
x = mySensVals[4];
```

```
1  /*...
13
14
15 int myPins[] = {0, 0, 0, 0, 11, 10, 9, 6, 0, 0, 0, 0};  // Array con valori cuscinetto ai lati
16 int ritardo = 20;                                        // Uso il ritardo come variabile
17
18
19 void setup() {
20
21   for(int i = 4; i < 8; i++){      // Inizializza l'array con un ciclo for
22       pinMode(myPins[i], OUTPUT);
23   }
24
25 }
26
27 void loop() {
28
29  //
30  for(int i = 3; i < 9; i++){       // Uso un ciclo for per creare una onda
31    analogWrite(myPins[i], 250);    // di valori sull'array di pin
32    delay(ritardo);
33    analogWrite(myPins[i-1], 50);
34    delay(ritardo);
35    analogWrite(myPins[i-2], 0);
36    delay(ritardo);
37    analogWrite(myPins[i-3], 0);
38    delay(ritardo);
39
40  }
41
42
43  for(int i = 8; i >= 3; i--){
44    analogWrite(myPins[i], 250);
45    delay(ritardo);
46    analogWrite(myPins[i+1], 50);
47    delay(ritardo);
48    analogWrite(myPins[i+2], 0);
49    delay(ritardo);
50    analogWrite(myPins[i+3], 0);
51    delay(ritardo);
52  }
53
54 }
55
```

Utilizziamo un array per memorizzare il numero dei pin che useremo.
Questo ci permette di automatizzare le funzioni di variazione dei valori.

Utilizzo dei valori cuscinetto per non eccedere i limiti dell'array. Il codice può essere migliorato ;-)

Esercizio 5

Cominciamo ad utilizzare elementi di input digitale. Aggiungiamo un pulsante al circuito del led del primo esercizio e manteniamo acceso il led finché si tiene premuto il pulsante.

Pulsante elettrico, il pulsante è un dispositivo elettrico simile all'interruttore ma con una molla che lo riporta alla posizione di partenza appena viene rilasciato..

Micro-pulsante da circuito stampato

Simbolo circuitale del pulsante NO ed NC

Pulsanti per illuminotecnica

Pulsanti per applicazioni industriali

Esistono due tipi di pulsanti:

Normally open (NO): "normalmente aperto", appena viene premuto chiude il circuito;
Normally closed (NC): “normalmente chiuso", appena viene premuto apre il circuito.

Fra gli usi più comuni: accendere e spegnere la luce per mezzo di un relè, suonare il campanello o il citofono, chiamare l'ascensore, etc...

Per mostrare la similitudine, mantengo lo schema originale del led blink e aggiungo il circuito del pulsante.

Resistenza da 10KΩ

Per mostrare la similitudine, mantengo lo schema originale del led blink e aggiungo il circuito del pulsante.

Il pulsante quando non è premuto crea un circuito che parte da 5V e si chiude in GND

Per mostrare la similitudine, mantengo lo schema originale del led blink e aggiungo il circuito del pulsante.

Quando il pulsante è premuto il circuito viene creato tra 5V e il PIN 2, il quale ne legge il valore.

Ora **utilizzeremo un pulsante come dispositivo di input digitale**.
Il pulsante sarà collegato con un pin digitale che utilizzeremo come segnalatore della pressione.

Innanzitutto dobbiamo dire ad arduino che il pin che stiamo utilizzando verra utilizzato per leggere e non per scrivere.

```
pinMode(pin, INPUT);
```

Dopodiché introduciamo la funzione **digitalRead() che legge il valore di un pin**. Questo valore può essere HIGH o LOW.

Funzione

```
digitalRead(pin);
```

RETURN
La funzione, quando viene utilizzata, restituisce un valore. Il valore è la presenza o meno di tensione nel PIN indicato. Quindi HIGH o LOW.

Numero del pin che voglio leggere

Istruzione condizionale: **IF()** effettua un controllo su un'espressione condizionale, se questa è vera esegue il primo blocco di istruzioni, se è falsa, esegue il secondo.

Espressione condizionale

```
if (qualcosa < 500)
{
    // action A;
}
else
{
    // action B;
}
```

Se l'espressione è vera esegue questo blocco.

Altrimenti esegue questo blocco.

```
/*
 Button

 Accendiamo un led attraverso un bottone.
Il led rimane acceso finché il bottone è premuto.


The circuit:
* LED attached from pin 9 to ground
* pushbutton attached to pin 2 from +5V
* 10K resistor attached to pin 2 from ground

by DojoDave and Tom Igoe

This example code is in the public domain.

http://www.arduino.cc/en/Tutorial/Button
*/

int led = 9;              //Diamo un nome al PIN 9
int buttonPin = 2;        // Agganciamo un piedino del bottone al PIN 2

int buttonState = 0;     // Definiamo una variabile in cui memorizzeremo
                         // lo stato del bottone
                         // 0 = non premuto, 1 = premuto


void setup() {
  pinMode(led, OUTPUT);       // Dico ad A. che il PIN 9 funziona come OUTPUT
  pinMode(buttonPin, INPUT);  // Dico ad A. che il PIN 2 funziona come INPUT
}



void loop() {
  buttonState = digitalRead(buttonPin);  // 1. leggolo stato del bottone e lo memorizzo
                                         // nella variabile che abbiamo creato

                                          // 2. uso un costrutto decisionele
                                          // se il bottone è premuto, accende il led
                                          // altrimenti spegne il led
  if (buttonState == HIGH) {
    // turn LED on:
    digitalWrite(led, HIGH);
  }
  else {
    // turn LED off:
    digitalWrite(led, LOW);
  }

}
```

digitalRead() legge il valore del pulsante. L'istruzione IF() determina se il pulsante è premuto o no.
Se lo è, accende il led.

Esercizio 6

Trasformiamo il nostro circuito con pulsante a pressione, in un pulsante a stato definito. Se premo la prima volta il led si accende, se premo la seconda volta il led si spegne.

Un automa a stati finiti è un modello che permette di descrivere con precisione e in maniera formale il comportamento di molti sistemi.

	HIGH	LOW
S1 (acceso)	S2	S1
S2 (spento)	S1	S2

Un automa a stato può essere descritto da un **diagramma degli stati** o da una **tabella di transizione**.

```
1   /*...
19
20  int led = 9;              //Diamo un nome al PIN 9
21  int buttonPin = 2;        // Agganciamo un piedino del bottone al PIN 2
22
23  int buttonState = 0;     // Dove conserviamo lo stato del bottone
24
25  int ledState = 0;        // Dove conserviamo lo stato del led
26
27
28  void setup() {
29    pinMode(led, OUTPUT);        // Dico ad A. che il PIN 9 funziona come OUTPUT
30    pinMode(buttonPin, INPUT);   // Dico ad A. che il PIN 2 funziona come INPUT
31  }
32
33
34
35  void loop() {
36    buttonState = digitalRead(buttonPin);   // 1. leggolo stato del bottone e lo memorizzo
37                                            // nella variabile che abbiamo creato
38
39                                            // 2. controllo che il bottone sia premuto
40                                            // se lo è modifico lo stato del LED.
41    if (buttonState == HIGH){
42      ledState = !ledState;                 // 3. vario lo stato del led al variare dello stato precedente
43    }
44
45                                            // 4. In base allo stato del LED lo accendo o lo spengo
46    if (ledState == HIGH) {
47      // turn LED on:
48      digitalWrite(led, HIGH);
49    }
50    else {
51      // turn LED off:
52      digitalWrite(led, LOW);
53    }
54
55  }
56
```

Modifichiamo il codice dell'esercizio precedente. Aggiungiamo una variabile che memorizza lo stato del led. Lo stato può essere 0: spento o altro: acceso

```
1   /*...

19

20  int led = 9;              //Diamo un nome al PIN 9
21  int buttonPin = 2;        // Agganciamo un piedino del bottone al PIN 2
22
23  int buttonState = 0;     // Dove conserviamo lo stato del bottone
24
25  int ledState = 0;        // Dove conserviamo lo stato del led
26
27
28  void setup() {
29    pinMode(led, OUTPUT);        // Dico ad A. che il PIN 9 funziona come OUTPUT
30    pinMode(buttonPin, INPUT);   // Dico ad A. che il PIN 2 funziona come INPUT
31  }
32
33
34
35  void loop() {
36    buttonState = digitalRead(buttonPin);   // 1. leggolo stato del bottone e lo memorizzo
37                                            // nella variabile che abbiamo creato
38
39                                            // 2. controllo che il bottone sia premuto
40                                            // se lo è modifico lo stato del LED.
41    if (buttonState == HIGH){
42      ledState = !ledState;                 // 3. vario lo stato del led al variare dello stato precedente
43    }
44
45                                            // 4. In base allo stato del LED lo accendo o lo spengo
46    if (ledState == HIGH) {
47      // turn LED on:
48      digitalWrite(led, HIGH);
49    }
50    else {
51      // turn LED off:
52      digitalWrite(led, LOW);
53    }
54
55  }
56
```

Ad ogni ciclo controllo se il pulsante è schiacciato. Se lo è inverto lo stato di ledstate.

Al passo successivo se lo stato di ledstate è acceso, allora accendo il led, altrimenti lo spengo.

Esercizio 7

Se provate a premere il pulsante rapidamente, il sistema impazzisce.
Questa soluzione ha due problemi: non tiene conto dello stato precedente del led e ha problemi di “bouncing”.
Proviamo ad eliminarli.
Stesso circuito, stesso funzionamento.

> **Pushbuttons are very simple** devices: two bits of metal kept apart by a spring. When you press the button, the two contacts come together and electricity can flow. This sounds fine and simple, but in real life **the connection is not that perfect**, especially when the button is not completely pressed, and **it generates some spurious signals called bouncing**.
>
> *M. Banzi*, Getting started with arduino

Si chiamano tecniche di debouncing tutti quegli accorgimenti che ci permettono di eliminare o controllare le imperfezioni del segnale. Esistono diverse tecniche di debouncing, alcune anche molto complesse, noi ci limiteremo ad usarne una molto semplice, ma efficace per i nostri scopi.

```
1  /*...
19
20 int led = 9;              //Diamo un nome al PIN 9
21 int buttonPin = 2;        // Agganciamo un piedino del bottone al PIN 2
22
23 int buttonState = 0;       // Dove conserviamo lo stato del bottone
24 int old_buttonState = 0;  // Dove memorizzo lo stato precedente del bottone
25
26 int ledState = 0;        // Dove conserviamo lo stato del led
27
28
29 void setup() {
30   pinMode(led, OUTPUT);        // Dico ad A. che il PIN 9 funziona come OUTPUT
31   pinMode(buttonPin, INPUT);  // Dico ad A. che il PIN 2 funziona come INPUT
32 }
33
34
35
36 void loop() {
37   buttonState = digitalRead(buttonPin);   // 1. leggolo stato del bottone e lo memorizzo
38                                           // nella variabile che abbiamo creato
39
40                                           // 2. controllo che il bottone sia premuto
41                                           // se lo è modifico lo stato del LED.
42
43   if (buttonState == HIGH && old_buttonState == LOW){  // Effettuo un debouncing
44     ledState = !ledState;                              // attraverso lo stato del bottone
45     delay(15);                                         // e un leggero ritardo in ms
46   }
47
48   old_buttonState = buttonState;  // memorizzo lo stato del bottone
49
50
51   if (ledState == HIGH) {
52     // turn LED on:
53     digitalWrite(led, HIGH);
54   }
55   else {
56     // turn LED off:
57     digitalWrite(led, LOW);
58   }
59
60 }
61
```

Esistono diverse tecniche di debouncing. Noi ne usiamo una semplicissima: al codice precedente **aggiungiamo un controllo sullo stato precedente del pulsante** e aggiungiamo un ritardo nell'esecuzione del ciclo.

Esercizio 8

Aggiungiamo all'esercizio precedente, un nuovo comportamento. Al click del pulsante il led si accende o si spegne, se invece lo tengo premuto, modifico la luminosità del led.

La funzione millis() calcola e restituisce i millisecondo trascorsi dall'inizio dell'esecuzione dello sketch. A differenza di delay() **non interrompe il flusso del programma**.

Funzione

```
millis();
```

RETURN
I millisecondi trascorsi dalla partenza dello sketch.

```
/*...

int led = 9;                //Diamo un nome al PIN 9
int buttonPin = 2;          // Agganciamo un piedino del bottone al PIN 2

int buttonState = 0;        // Dove conserviamo lo stato del bottone
int old_buttonState = 0;    // Dove memorizzo lo stato precedente del bottone

int ledState = 0;           // Dove conserviamo lo stato del led

int luminosita = 0;
int variazione = 5;

unsigned long starttime = 0;

void setup() {
  pinMode(led, OUTPUT);       // Dico ad A. che il PIN 9 funziona come OUTPUT
  pinMode(buttonPin, INPUT);  // Dico ad A. che il PIN 2 funziona come INPUT
}



void loop() {
  buttonState = digitalRead(buttonPin);                    // 1. leggolo stato del bottone e lo memorizzo
                                                           // nella variabile che abbiamo creato

                                                           // 2. controllo che il bottone sia premuto
                                                           // se lo è modifico lo stato del LED.

  if (buttonState == HIGH && old_buttonState == LOW){      // Effettuo un debouncing
    ledState = !ledState;                                  // attraverso lo stato del bottone
    starttime = millis();                                  // Memorizzo il tempo del momento in cui ho premuto il bottone

    delay(15);
  }

  if (buttonState == HIGH && old_buttonState == HIGH){     // 3. Verifico se il bottone è ancora premuto
    if (ledState == HIGH && (millis() - starttime) > 500){ // Verifico se è premuto da più di 500 secondi
      luminosita += variazione;                            // modifico la luminosità
      delay(30);

      if (luminosita < 5 || luminosita >250){              // inverto la variazione ai margini
        variazione = -variazione;
      }

    }
  }

  old_buttonState = buttonState;


  if (ledState == HIGH) {
    // turn LED on:
    analogWrite(led, luminosita);
  }
  else {
    // turn LED off:
    digitalWrite(led, LOW);
  }

}
```

Modifichiamo il codice precedente. Aggiungiamo un controllo che calcola quanto tempo rimane premuto il bottone e dopo un certo valore, modifica la luminosità.

Esercizio 9

Input analogico. Vogliamo cambiare l’intensità di un led utilizzando un potenziometro.

Segnale Analogico e Digitale

Un segnale digitale può assumere solo due valori: 0V e +5V.

Un segnale analogico può assumere tutti i valori compresi tra 0V e +5V (nel caso della nostra scheda Arduino).

Potenziometro (o reostato) è un resistore variabile, costituito da un cilindro isolante su cui è avvolto un filo metallico; le due estremità sono connesse a due morsetti. Longitudinalmente al cilindro e da un'estremità all'altra, scorre un cursore recante un contatto strisciante sul filo, a sua volta collegato ad un morsetto.

Fino ad ora abbiamo utilizzato un input digitale acceso/spento. Ora vogliamo utilizzare come input una serie di valori variabili. Arduino fornisce una serie di **PIN analogici**, e funzioni specifiche che **convertono l'intensità del segnale in entrata in un valore compreso tra 0 e 1023**.

Pin analogici IN

Il potenziometro lavora su un circuito separato, come il pulsante. E' una resistenza variabile che cambia girando la manopola.

Resistenza variabile, vuol dire che cambia la tensione sul circuito che parte dal PIN 5V e finisce in GND. Il PIN analogico A0 registra questa variazione in un valore che va da 0 a 1023

Lo sketch che scriveremo, interpreta quel valore e modifica lo stato del LED.

analogRead() traduce l'intensità in ingresso tra 0 e 5 volts in un valore intero compreso tra 0 e 1023. E' la funzione che utiliziamo per leggere valori provenienti da resistenze variabili: potenziometri, fotoresistenze, etc. collegate ad un pin analogico.

Funzione

```
analogRead(pin);
```

RETURN
La funzione, quando viene utilizzata, restituisce un intero compreso tra 0 e 1023.

Numero del pin che voglio leggere

La funzione **map()** rimappa un valore a partire **da una scala di valori data ad una nuova**. Ad esempio se ricevo un valore in un pin digitale in una scala 0/1023, per usarlo con un pin pwm devo convertirlo nel corrispettivo valore in una scala 0/255.

Funzione

Valore da convertire

Valore massimo corrente

Nuovo valore massimo

```
map(value, fromLow, fromHigh, toLow, toHigh)
```

RETURN
Restituisce il valore corrispondente in una nuova scala.

Valore minimo corrente

Nuovo valore minimo

```
/*...

int LED = 9;        // Diamo un nome al PIN 9
int POT = A0;       // Diamo un nome al PIN A0;

int valore = 0;     // Definiamo una variabile che useremo
                    // per il valore del potenzionetro

int valoreNormalizzato = 0;  // definiamo una variabile dove metteremo il risultato
                             // di VALORE una volta normalizzato



void setup() {
  pinMode(LED, OUTPUT);      // Dico ad A. che il PIN 9 funziona come OUTPUT
  pinMode(POT, INPUT);           // Dico ad A. che il PIN A0 funziona come INPUT
}



void loop() {
  valore = analogRead(POT);     // Questo valore può andare da 0 a 1023
                                // ho bisogno di portarlo in una scala da 0 a 255

  valoreNormalizzato = map(valore, 0, 1023, 0, 255);
  analogWrite(LED, valoreNormalizzato);


}

```

Il codice è molto semplice.
Le uniche novità sono la funzione analogRead() e la funzione map()

Cerchiamo di capire il concetto di normalizzazione o campionamento: stampiamo sul monitor seriale i valori del potenziometro al PIN A0 e il suo corrispettivo normalizzato dopo l'applicazione della funzione map().

La comunicazione seriale è il modo più semplice e flessibile per permettere ad Arduino di comunicare con qualsiasi dispositivo esterno
come, ad esempio, un computer.

Un layer intermedio, un proxy, scritto in un qualsiasi linguaggio (generalmente processing) ci permette di accedere a tutte le risorse del computer.

Arduino utilizza una libreria standard chiamata Serial. La quale mette a disposizione una serie di funzioni che facilitano la comunicazione seriale.

Vedremo in un esercizio più avanzato come usare l'oggetto Serial per comunicare con Processing.
Per ora ci limiteremo a capirne il funzionamento per abilitare il **monitor seriale**. Uno strumento che ci permette di stampare sul monitor una serie di valori del nostro sketch.

Primo passo, dire ad Arduino **a che velocità vogliamo parlare**. Avviamo la porta seriale ad una velocità stabilita.

Funzione

```
Serial.begin(speed);
```

Bits per second

Secondo passo, **stampare sul monitor seriale**: print() e println() stampano il valore che gli viene passato come parametro.

Funzione

Valore da stampare

```
Serial.print(value);
Serial.println(value);
```

Esempio di codice che stampa una stringa di testo

```
Serial.print("Hello world!");
```

```
1  /*
2
3   */
4
5  int LED = 9;     // Diamo un nome al PIN 9
6  int POT = A0;      // Diamo un nome al PIN A0;
7
8  int valore = 0;    // Definiamo una variabile che useremo
9                     // per il valore del potenzionetro
10
11 int valoreNormalizzato = 0;  // definiamo una variabile dove metteremo il risultato
12                              // di VALORE una volta normalizzato
13
14
15
16 void setup() {
17   pinMode(LED, OUTPUT);   // Dico ad A. che il PIN 9 funziona come OUTPUT
18   pinMode(POT, INPUT);          // Dico ad A. che il PIN A0 funziona come INPUT
19
20   Serial.begin(9600);          // Apro la porta seriale per inviare dati al computer
21                                // alla velocità di 9600 bit al secondo
22 }
23
24
25
26 void loop() {
27   valore = analogRead(POT);     // Questo valore può andare da 0 a 1023
28                                 // ho bisogno di portarlo in una scala da 0 a 255
29
30
31   valoreNormalizzato = map(valore, 0, 1023, 0, 255);
32   analogWrite(LED, valoreNormalizzato);
33
34   Serial.print(valore);                // Stampo i valori sul monitor seriale
35   Serial.print(" -- > ");
36   Serial.println(valoreNormalizzato);
37
38   delay(1000);                  // Aggiungiamo un ritardo di un secondo tale
39                                 // per osservare meglio il motior se
40
41 }
42
```

Aggiungiamo solo poche righe e abbiamo un sistema per verificare i nostri valori.

Con lo stesso sistema, in seguito, impareremo a comunicare con altri ambienti.

Esercizio 10

Input analogico. Vogliamo cambiare l'intensità di un led utilizzando una fotoresistenza e sfruttando il Partitore di Tensione.

Il fotoresistore è un componente elettrico la cui resistenza è inversamente proporzionale alla quantità di luce che lo colpisce.

Si comporta come un tradizionale resistore, ma il suo valore in ohm diminuisce quando aumenta l'intensità della luce che la colpisce.

Partitore di tensione (e di corrente).
Molte volte nella pratica si ha la necessità di derivare da un generatore di tensione una tensione minore di quella prodotta per applicarla ad un carico. Un metodo utilizzabile è quello del partitore di tensione che si compone di due resistenze in serie.
Stesso discorso dicasi per la corrente.

Partitore resistivo di tensione

Partitore resistivo di corrente

Lo schema è identico a quello del potenziometro. Aggiungiamo due resistenze in serie e otteniamo un Partitore di Tensione.

La fotoresistenza è una resistenza variabile. Collegando due resistenze in serie posso calcolare la tensione media fra le due. Se una delle due è variabile, varia anche la tensione media.

Il potenziometro, al suo interno, implementa un circuito simile, per cui lo sketch che scriveremo è simile a quello usato per il potenziometro.

La funzione map ha un piccolo difetto rispetto ai nostri scopi. Se riceve dei valori fuori range, li considera valori validi. **Per costringere i valori entro la scala che ci interessa, usiamo la funzione constrain()**.

Funzione

Limite superiore

```
constrain(x, a, b);
```

RETURN
La funzione restituisce X se x è compreso tra a e b. Altrimenti a se x è minore di a e b se x è maggiore di b.

Limite inferiore

```
1  /*
2
3   */
4
5  int LED = 9;     // Diamo un nome al PIN 9
6  int FOTO = A0;      // Diamo un nome al PIN A0;
7
8  int valore = 0;    // Definiamo una variabile che useremo
9                     // per il valore del potenzionetro
10
11 int valoreNormalizzato = 0;  // definiamo una variabile dove metteremo il risultato
12                              // di VALORE una volta normalizzato
13
14
15
16 void setup() {
17   pinMode(LED, OUTPUT);   // Dico ad A. che il PIN 9 funziona come OUTPUT
18   pinMode(FOTO, INPUT);          // Dico ad A. che il PIN A0 funziona come INPUT
19
20   Serial.begin(9600);          // Apro la porta seriale per inviare dati al computer
21                                // alla velocità di 9600 bit al secondo
22 }
23
24
25
26 void loop() {
27   valore = analogRead(FOTO);     // Questo valore può andare da 0 a 1023
28                                  // ho bisogno di portarlo in una scala da 0 a 255
29
30
31   valoreNormalizzato = map(valore, 15, 60, 0, 255);
32   valoreNormalizzato = constrain(valoreNormalizzato, 0, 255);
33   valoreNormalizzato = 255 - valoreNormalizzato;
34
35   analogWrite(LED, valoreNormalizzato);
36
37   Serial.print(valore);                // Stampo i valori sul monitor seriale
38   Serial.print(" -- > ");
39   Serial.println(valoreNormalizzato);
40
41   delay(10);                   // Aggiungiamo un ritardo di un secondo solo
42                                // per osservare meglio il motior seriale
43
44 }
45
```

Come vedete è identico al codice scritto per il potenziometro. Cambiamo solo i valori di map, perché dobbiamo regolare i valori sul valore del Partitore di Tensione.

Inoltre utilizziamo la funzione costrain() per eliminare i valori fuori scala.

Cambiamo la resistenza fissa e utilizziamone una di diverso valore, osserviamo come cambiano i valori medi e come è necessario effettuare una nuova mappatura.

Esercizio 11

A partire dal circuito precedente, costruiamo un teremin rudimentale, utilizzando un buzzer al posto del led.

Buzzer (o Beeper)

Il buzzer è un dispositivo di segnalazione audio di tipo meccanico, elettromeccanico o piezoelettrico. Un tipico utilizzo è come dispositivo di sveglia, temporizzatori, o nei pc.

Modelli di buzzer esistenti in commercio

Simbolo elettrico del buzzer

Usiamo lo stesso schema della fotoresistenza. Sostituiamo al LED il Piezo Buzzer.

Utilizziamo una resistenza molto alta, perché il buzzer è anche un sensore di pressione che produce una tensione di ritorno.

Spostiamo il pin di uscita del Buzzer al PIN 8, perché **i buzzer interferiscono con i PIN pwm**

Per far emettere un suono semplice al buzzer usiamo la funzione tone(). Tone crea un'onda quadra di una certa durata e una certa frequenza che il buzzer trasforma in suono.

Funzione

Durata del tono in millisecondi

```
tone(pin, frequency, duration);
```

Pin al quale è associato il buzzer e a cui applico la funzione tone().

Frequenza del tono in Hertz

```
/*

 */

int PIEZO = 8;     // Diamo un nome al PIN 8
int FOTO = A0;      // Diamo un nome al PIN A0;

int valore = 0;    // Definiamo una variabile che useremo
                   // per il valore del potenzionetro

int valoreNormalizzato = 0;  // definiamo una variabile dove metteremo il
                             // di VALORE una volta normalizzato



void setup() {
  pinMode(PIEZO, OUTPUT);   // Dico ad A. che il PIN 9 funziona come OUTPUT
  pinMode(FOTO, INPUT);          // Dico ad A. che il PIN A0 funziona come INPUT

  Serial.begin(9600);          // Apro la porta seriale per inviare dati al computer
                               // alla velocità di 9600 bit al secondo
}



void loop() {
  valore = analogRead(FOTO);     // Questo valore può andare da 0 a 1023
                                 // ho bisogno di portarlo in una scala da 31 a 4978
                                 // questi sono i valori in cui sono compresi toni e semitoni
                                 // gestiti da arduino.


  valoreNormalizzato = map(valore, 15, 60, 31, 4978);
  valoreNormalizzato = constrain(valoreNormalizzato, 31, 4978);
  valoreNormalizzato = 4978 - valoreNormalizzato;

  tone(PIEZO, valoreNormalizzato, 5);   // Non tutti i valori corrispondono a delle note
                                        // ma al momento ci interessa l'effetto teremin
                                        // piuttosto che un codice perfetto

  Serial.print(valore);
  Serial.print(" -- > ");
  Serial.println(valoreNormalizzato);

  delay(5);

}
```

Al posto della funzione classica digitalWrite() utilizziamo una funzione tone().

Campioniamo i valori della fotoresistenza, in quelli delle frequenze che vogliamo ottenere.

Senza modificare nulla al circuito, aprite l'esempio di arduino:
File > esempi > digital > toneMelody

Caricate lo sketch e fatelo partire, senza modificare nulla.

Esercizio 12

Servo motori. Proviamo ad agganciare un servo motore ad Arduino e facciamogli compiere ripetutamente un movimento destra sinistra di 180 gradi.

Servomotori

Particolare tipo di motore generalmente di piccola potenza, le cui condizioni operative sono soggette ad ampie e spesso repentine variazioni sia nel campo della velocità che della coppia motrice, alle quali si deve adattare con la massima rapidità e precisione.

I servomotori trovano applicazione nei controlli di posizione, sistemi automatici di regolazione e nelle periferiche di sistema, come stampanti e plotter.

PILOTAGGIO

Agganciamo direttamente il servomotore ad Arduino. La sua tensione di utilizzo è 5v, quindi non ha bisogno di resistenze.

Il servomotore ha 3 collegamenti. Uno che va al PIN analogico. Uno che va alla tensione di alimentazione e uno che finisce a terra.

Per semplificarci la vita possiamo usare le librerie. Le librerie sono pacchetti di funzioni scritte per uno scopo specifico. Alcune librerie sono interne e già pronte all'uso, altre sono esterne e vanno incluse.

Istruzione di inclusione

Nome della libreria

```
#include <Servo.h>
```

Esistono librerie di ogni tipo. Librerie per i motori, per il suono, per le connessioni wifi. A questo indirizzo c'è l'elenco ufficiale: http://arduino.cc/en/Reference/Libraries

Una volta inclusa la libreria nel nostro codice, **occorre dichiarare una variabile del tipo della libreria**. Dopodiché, attraverso la "notazione puntata", posso accedere alle funzioni contenute nella variabile.

Tipo della variabile.
Ha lo stesso nome della libreria, meno l'estensione.

Nome della variabile.

Dichiarazione

```
Servo qualsiasinome;
```

Oggetto dichiarato.

Utilizzo di una funzione

```
qualsiasinome.write();
```

Funzione

La libreria che abbiamo appena dichiarato è la **libreria che ci facilita l'uso dei servo motori.** Le due funzioni principali sono attach() e write().

Variabile che abbiamo appena dichiarato. Ha il nome che abbiamo deciso noi.

```
myservo.attach(numero pin)
```

La funzione attach ha la stessa funzione di pinMode(). Dice ad arduino che tipo di utilizzo viene fatto del pin. Più semplicemente possiamo dire che stiamo dichiarando a che pin agganciamo il servo motore.

La funzione write(), della libreria Servo, fa ruotare il servomotore di un angolo che indichiamo come parametro nella funzione.

Variabile che abbiamo appena dichiarato. Ha il nome che abbiamo deciso noi.

```
myservo.write(gradi)
```

La funzione fa ruotare il servomotore di un angolo che va da 0 a 180°

In questo esercizio introduciamo anche una nuova struttura di controllo, il ciclo while. Il ciclo while esegue un blocco di istruzioni finché l'espressione condizionale è vera.

Espressione condizionale che può essere true o false.

Istruzione del ciclo

```
while(expression){
  istruzione1;
  istruzione2;
  istruzionen;
}
```

Il blocco istruzioni viene eseguito solo e finché expression è true.

```
1  // Controlling a servo position using a potentiometer (variable resistor)
2  // by Michal Rinott <http://people.interaction-ivrea.it/m.rinott>
3
4
5  #include <Servo.h>  // Includiamo la libreria che ci permette di utilizzare il servo
6
7  Servo myservo;      // Creo un oggetto servo, come fosse una variabile di un tipo nuovo.
8
9  int posizione = 0;  //
10
11 void setup()
12 {
13   myservo.attach(9);  // collego il servo al pin 9
14
15
16 }
17
18 void loop()
19 {
20
21       while(posizione < 180){
22         posizione ++;
23         myservo.write(posizione);    // scrivo la posizione del motorino
24         delay(15);
25     }
26
27       while(posizione > 0){              // utilizzo un nuovo ciclo di controllo
28         posizione --;             // ma potevo usare anche un for
29         myservo.write(posizione);    // scrivo la posizione del motorino
30         delay(15);
31     }
32
33
34 }
```

Utilizziamo un ciclo while per far compiere al servomotore un arco di movimento di 180 gradi. Poi facciamo la stessa cosa per riportarlo nella posizione iniziale.

```
// Sweep
// by BARRAGAN <http://barraganstudio.com>
// This example code is in the public domain.


#include <Servo.h>

Servo myservo;  // create servo object to control a servo
                // a maximum of eight servo objects can be created

int pos = 0;    // variable to store the servo position
int servopin = 9;


void setup()
{
  myservo.attach(servopin);  // attaches the servo on pin 9 to the servo object
}


void loop()
{
  for(posizione = 180; posizione>=1; posizionepos-=1)
  {
      myservo.write(pos);
      delay(10);
  }

  for(posizione = 0; posizione < 180; posizione += 1)
  {
      myservo.write(pos);
      delay(10);
  }

}
```

Avremmo potuto fare la stessa cosa con un ciclo FOR, ottenendo lo stesso risultato.

Esercizio 13

Servo motori. Adesso vogliamo che il servomotore faccia lo stesso movimento, ma che sia controllabile da un potenziometro.

Il circuito sembra complicato, in realtà abbiamo messo tutto sulla breadboard per poter agganciare anche il potenziometro.

Il servomotore e il potenziometro formano due circuiti separati. Arduino si occupa di creare il comportamento a partire dai dati del potenziometro

Ricordiamoci che il potenziometro è una resistenza variabile e quindi il PIN A0, analogico, trasforma la variazione di tensione in 1023 valori.

```
// Controlling a servo position using a potentiometer (variable resistor)
// by Michal Rinott <http://people.interaction-ivrea.it/m.rinott>

#include <Servo.h>

Servo myservo;          // create servo object to control a servo

int potpin = A0;        // analog pin used to connect the potentiometer
int posizione;          // variable to read the value from the analog pin

void setup()
{
  pinMode(potpin, INPUT);
  myservo.attach(9);        // attaches the servo on pin 9 to the servo object
}

void loop()
{
  posizione = analogRead(potpin);                  // Leggo il valore del potenziometro
  posizione = map(posizione, 0, 1023, 0, 179);     // Riduco la gamma di valori a 180
  myservo.write(posizione);                        // Scrivo la posizione sul servo
  delay(15);
}
```

Il codice è estremamente semplice: leggo e memorizzo il valore del potenziometro e scrivo la posizione sul servomotore.

Esercizio 14

Pilotare carichi con tensione più elevata di quella con cui lavora arduino. Proviamo a collegare un motore dc e a controllarlo.

Transistor
(a giunzione bipolare)

Il transistor è composto da un materiale semiconduttore al quale sono applicati tre terminali che lo collegano al circuito esterno.

L'applicazione di una tensione elettrica o di una corrente elettrica a due terminali permette di regolare il flusso di corrente che attraversa il dispositivo, e questo permette di amplificare il segnale in ingresso.

Le principali funzioni che gli vengono affidate all'interno di un circuito elettronico sono:

L'amplificazione di un segnale in entrata.
Il funzionamento da interruttore (switch).

Transistor bipolare a giunzione tipo p-n-p e n-p-n con rappresentazione circuitale.

MosFET,

acronimo del termine inglese **metal-oxide-semiconductor field-effect transistor**, ovvero **transistor metallo-ossido-semiconduttore a effetto di campo**, scritto anche MOS-FET o MOS FET e spesso conosciuto come transistor MOS, è una tipologia di transistor ad effetto di campo largamente usata nel campo dell'elettronica digitale, ma diffusa anche nell'elettronica analogica.

Diodo

Il diodo è un dispositivo elettronico detto attivo la cui funzione ideale è quella di permettere il flusso di corrente elettrica solo in un verso e di bloccarla totalmente nell'altro.

Diodo di ricircolo o di Free-Wheeling

Nei circuiti viene utilizzato un diodo disposto in anti-parallelo con un carico induttivo (una bobina del relè o un motore elettrico). In questo caso è chiamato diodo di "ricircolo" o di "libera circolazione" (free wheeling).

Il diodo è collegato in modo che non conduca (sia cioè interdetto) quando il carico viene attivato.

Quando il carico induttivo viene disattivato, si avrebbe un picco di tensione nel senso inverso. Questo picco di tensione è conosciuto come “rimbalzo induttivo“ ed il diodo permette la dissipazione di questa energia sulla componente resistiva del carico induttivo impedendo che torni indietro danneggiando i semiconduttori.

Il mosfet separa il motore da arduino e funziona come un interruttore.
Il pin 9 pwm mi permette di aprire e chiudere il circuito con una frequenza pari al valore proveniente dal potenziometro.
Quello che faccio, quindi, è utilizzare analogWrite() direttamente sul pin di controllo del mosfet.

Utilizziamo un mosfet o un normale transistor per scollegare il circuito del motore da Arduino.

Usiamo un mosfet o un transistor semplicemente come interruttore. Viene attivato dal PIN 9.

Il potenziometro ha un circuito distinto, come abbiamo già visto.
Invia il suo valore al Pin A0.

Per il motore DC utilizziamo un diodo che impedisce un callback induttivo, ovvero una onda di ritorno di tensione causata dal movimento del motore.

Il motore in realtà potrebbe richiedere troppa corrente rispetto a quella fornita da Arduino, per cui possiamo anche usare una fonte di energia esterna.

Il circuito non cambia, tranne per il fatto che la fonte di alimentazione non proviene più dal PIN 5V.

Il collegamento alla massa di entrambi i circuiti deve essere lo stesso.

fritzing

```
1  /*
2      Il motore è guidato da un potenziometro che ne varia
3      il PWM.
4   */
5
6  int potenziometro = A0;     // potenziometro
7  int motore = 9;             // motore
8
9  void setup() {
10   // inizializza il motore come output
11   pinMode(motore, OUTPUT);
12   pinMode(potenziometro, INPUT);
13   Serial.begin(9600);
14 }
15
16 void loop(){
17   byte valore = map(analogRead(potenziometro),0,1023,0,255);
18   Serial.println(valore);
19
20   analogWrite(motore,valore);
21 }
22
23
24
```

Il codice è estremamente semplice: leggo e memorizzo il valore del potenziometro e abilito il motore attraverso il mosfet.

L'intermittenza di acceso/spento ad una certa frequenza determina la velocità di rotazione effettiva del motore dc.

Esercizio 15

Utilizziamo la porta seriale di Arduino per comunicare con il computer. Utilizziamo la fotoresistenza per modificare automaticamente un cerchio creato da processing.

Processing is a programming language, development environment, and online community. Since 2001, Processing has promoted software literacy within the visual arts and visual literacy within technology.

Processing.org

Processing è un linguaggio sviluppato per applicazioni grafiche e quindi fornisce una serie di vantaggi per tutte le applicazioni grafiche.

L'ambiente è simile ad Arduino, perché l'IDE arduino è una sua derivazione.

Verifica / Carica

Nuovo / Apri / Salva / Export

Sketch

Status area

Il linguaggio è derivato dal linguaggio JAVA. E' object oriented e nel complesso, più potente e strutturato del linguaggio usato da Arduino.

Riprendiamo il circuito dell’esercizio 10: la fotoresistenza che controlla la luminosità del led.

Ricostruiamo tutto il circuito, anche se per questo esercizio utilizzeremo solo la foto resistenza.

Le librerie per la comunicazione seriale via hardware o via software gestiscono la spedizione e la ricezione delle informazioni. Queste informazioni spesso sono costituite da gruppi di variabili che devono essere inviate tutte insieme. Perché le informazioni vengano interpretate correttamente, **la parte ricevente deve riconoscere dove comincia e dove finisce ciascun messaggio.** Una comunicazione significativa, che abbia un senso, così come ogni altro tipo di comunicazione tra una macchina e un'altra, è possibile solo se le parti che si occupano della spedizione e della ricezione sono d'accordo su come è organizzata l'informazione all'interno del messaggio. **L'organizzazione formale dell'informazione in un messaggio e la gamma di risposte appropriate da ricevere vengono dette protocollo di comunicazione.**

M. Margolis, Arduino Cookbook

CODICE ARDUINO

```
1  /*...
14
15
16 int FOTO = A0;      // Diamo un nome al PIN A0;
17
18 int valore = 0;    // Definiamo una variabile
19                    // dove memorizziamo i valori dell fotoresistenza
20
21
22
23 void setup() {
24   pinMode(FOTO, INPUT);          // Dico ad A. che il PIN A0 funziona come INPUT
25
26   Serial.begin(9600);           // Apro la porta seriale per inviare dati al computer
27                                 // alla velocità di 9600 bit al secondo
28 }
29
30
31
32 void loop() {
33   valore = analogRead(FOTO);  // Questo valore può andare da 0 a 1023
34
35   Serial.println(valore);     // Invio alla porta seriale il valore della fotoresistenza
36   delay(30);
37
38 }
39
```

Il codice di arduino legge il valore della foto resistenza, lo memorizza in una variabile e poi lo scrive sulla porta seriale. Il valore inviato termina con il fine riga.

CODICE PROCESSING

Il codice è molto simile a quello di arduino visto fino ad ora, ma con qualche variante. Anche processing ha un metodo setup() con la stessa funzione di quello visto per arduino e un metodo draw() che ha la stessa funzione del metodo loop() di arduino.

```
1   /*...
14
15
16  import processing.serial.*;
17
18  Serial port;
19  String myString;
20  char lf = '\n';     // Linefeed in ASCII
21
22  float raggio = 30;
23
24  void setup(){
25    frameRate(30);
26    size(640, 480);     // size(x, y)
27                        // imposta la finestra
28                        // lo zero è in alto a sinistra
29                        // la griglia è 0-639 e 0-479
30
31     smooth();           // antialias
32     strokeWeight(1);    // Spessore della linea
33     background(0);      // Colore di sfondo della finestra
34     fill(0);            // Colore riempimento
35     stroke(255);        // Colore del contorno
36
37
38    //println(Serial.list());
39    String portName = Serial.list()[3];          // memorizzo il nome della porta
40    port = new Serial(this, portName, 9600);     // inizializzo la porta
41    port.clear();                                // pulisco il buffer di memoria
42  }
43
44  void draw(){
45    background(0);
46
47    if (port.available() > 0) {
48      myString = port.readStringUntil(lf);       //  leggo la stringa sulla porta seriale fino al carattere \n
49      if (myString != null) {
50        myString = trim(myString);                       // elimino gli spazi vuoti
51        raggio = map(int(myString), 300, 900, 0, 255);   // faccio un mapping dei valori e li memorizzo
52      }                                                  // in una variabile raggio
53    }
54
55
56    fill(raggio);                                      // modifico il riempimento del cerchio
57    ellipse(width/2, height/2, raggio, raggio);        // e disegno il cerchio
58                                                       // in base ai colori della foto resistenza
59    println(raggio);
60
61  }
62
```

CODICE PROCESSING

Processing fornisce una serie di funzioni specifiche per il disegno. Nel setup() ho inserito una serie di funzione che modificano i parametri delle figure tracciate sulla finestra.

Ellipse() è la funzione che disegna il cerchio sullo schermo.

```
1   /*...
14
15
16  import processing.serial.*;
17
18  Serial port;
19  String myString;
20  char lf = '\n';     // Linefeed in ASCII
21
22  float raggio = 30;
23
24  void setup(){
25    frameRate(30);
26    size(640, 480);     // size(x, y)
27                        // imposta la finestra
28                        // lo zero è in alto a sinistra
29                        // la griglia è 0-639 e 0-479
30
31     smooth();           // antialias
32     strokeWeight(1);    // Spessore della linea
33     background(0);      // Colore di sfondo della finestra
34     fill(0);            // Colore riempimento
35     stroke(255);        // Colore del contorno
36
37
38    //println(Serial.list());
39    String portName = Serial.list()[3];          // memorizzo il nome della porta
40    port = new Serial(this, portName, 9600);     // inizializzo la porta
41    port.clear();                                // pulisco il buffer di memoria
42  }
43
44  void draw(){
45    background(0);
46
47    if (port.available() > 0) {
48      myString = port.readStringUntil(lf);       //  leggo la stringa sulla porta seriale fino al carattere \n
49      if (myString != null) {
50        myString = trim(myString);                          //elimino gli spazi vuoti
51        raggio = map(int(myString), 300, 900, 0, 255);      // faccio un mapping dei valori e li memorizzo
52      }                                                     // in una variabile raggio
53    }
54
55
56    fill(raggio);                                         // modifico il riempimento del cerchio
57    ellipse(width/2, height/2, raggio, raggio);           // e disegno il cerchio
58                                                          // in base ai colori della foto resistenza
59    println(raggio);
60
61  }
62
```

CODICE PROCESSING

```
1   /*...
14
15
16  import processing.serial.*;
17
18  Serial port;
19  String myString;
20  char lf = '\n';     // Linefeed in ASCII
21
22  float raggio = 30;
23
24  void setup(){
25    frameRate(30);
26    size(640, 480);     // size(x, y)
27                        // imposta la finestra
28                        // lo zero è in alto a sinistra
29                        // la griglia è 0-639 e 0-479
30
31    smooth();            // antialias
32    strokeWeight(1);     // Spessore della linea
33    background(0);       // Colore di sfondo della finestra
34    fill(0);             // Colore riempimento
35    stroke(255);         // Colore del contorno
36
37
38    //println(Serial.list());
39    String portName = Serial.list()[3];          // memorizzo il nome della porta
40    port = new Serial(this, portName, 9600);     // inizializzo la porta
41    port.clear();                                // pulisco il buffer di memoria
42  }
43
44  void draw(){
45    background(0);
46
47    if (port.available() > 0) {
48      myString = port.readStringUntil(lf);       //  leggo la stringa sulla porta seriale fino al carattere \n
49      if (myString != null) {
50        myString = trim(myString);                          // elimino gli spazi vuoti
51        raggio = map(int(myString), 300, 900, 0, 255);      // faccio un mapping dei valori e li memorizzo
52      }                                                     // in una variabile raggio
53    }
54
55
56    fill(raggio);                                           // modifico il riempimento del cerchio
57    ellipse(width/2, height/2, raggio, raggio);             // e disegno il cerchio
58                                                            // in base ai colori della foto resistenza
59    println(raggio);
60
61  }
62
```

Processing ha una classe serial che ci fornisce una serie di metodi per lavorare con la porta seriale.
La classe viene importata e dopo viene dichiarata una variabile con il tipo specifico della variabile.

CODICE PROCESSING

```
1   /*...
14
15
16  import processing.serial.*;
17
18  Serial port;
19  String myString;
20  char lf = '\n';    // Linefeed in ASCII
21
22  float raggio = 30;
23
24  void setup(){
25    frameRate(30);
26    size(640, 480);    // size(x, y)
27                       // imposta la finestra
28                       // lo zero è in alto a sinistra
29                       // la griglia è 0-639 e 0-479
30
31    smooth();           // antialias
32    strokeWeight(1);    // Spessore della linea
33    background(0);      // Colore di sfondo della finestra
34    fill(0);            // Colore riempimento
35    stroke(255);        // Colore del contorno
36
37
38    //println(Serial.list());
39    String portName = Serial.list()[3]; -------// memorizzo il nome della porta
40    port = new Serial(this, portName, 9600); ---// inizializzo la porta
41    port.clear();  ----------------------------// pulisco il buffer di
42  }
43
44  void draw(){
45    background(0);
46
47    if (port.available() > 0) {
48      myString = port.readStringUntil(lf); -----// leggo la stringa su
49      if (myString != null) {
50        myString = trim(myString);                      // elimino gli
51        raggio = map(int(myString), 300, 900, 0, 255); ---// faccio un ma
52      }                                                 // in una varia
53    }
54
55
56    fill(raggio);                                       // modifico il riempimento del cerchio
57    ellipse(width/2, height/2, raggio, raggio);-------  // e disegno il cerchio
58                                                        // in base ai colori della foto resistenza
59    println(raggio);
60
61  }
62
```

1. memorizzo il nome della porta disponibile
2. apro una porta seriale
3. pulisco il buffer della memoria

4. memorizzo in una stringa il contenuto della porta, fino al carattere di fine riga.
5. faccio una mappatura del valore e lo memorizzo in RAGGIO
6. disegno il cerchio con i valori ricevuti

Esercizio 16

Adesso facciamo l'esercizio inverso. Accendiamo o spegnamo un led controllato da arduino attraverso il movimento del mouse su una finestra di processing.

Stesso circuito
dell’esercizio precedente.
Utilizziamo la seconda
parte dello schema.

Accendiamo o spegnamo il LED

CODICE PROCESSING

```
/**
 * Simple Write.
 *
 * Check if the mouse is over a rectangle and writes the status to the serial port.
 * This example works with the Wiring / Arduino program that follows below.
 */


import processing.serial.*;

Serial myPort;  // Create object from Serial class
int val;        // Data received from the serial port

void setup()
{
  frameRate(30);
  size(200, 200);   // imposta la finestra

  String portName = Serial.list()[7];
  myPort = new Serial(this, portName, 9600);
}

void draw() {
  background(255);
  if (mouseOverRect() == true) {  // Se il mouse è dentro il box
    fill(204);                    // cambia colore del box
    myPort.write('H');            // invia il carattere H sulla porta seriale
  }
  else {                          // Se il mouse non è dentro il box
    fill(0);                      // cambia colore del box
    myPort.write('L');            // invia il carattere L sulla porta seriale
  }
  rect(50, 50, 100, 100);         // Disegna il quadrato
}

boolean mouseOverRect() { // Verifica se il mouse è dentro il box
  return ((mouseX >= 50) && (mouseX <= 150) && (mouseY >= 50) && (mouseY <= 150));
}
```

Dichiaro e creo la porta seriale, come abbiamo visto nel codice precedente. Dopodiché uso un costrutto IF per inviare sulla porta seriale una H o una L in base al fatto che il mouse sia o no nel box.

Il costrutto IF effettua un controllo sulla funzione mouseOverRect() che in realtà non fa altro che verificare se le coordinate del mouse si trovano all'interno di certi valori, che sono quelli del box. La funzione restituisce un valore True o False.

CODICE ARDUINO

```
/**
* Simple Write.
*
* Check if the mouse is over a rectangle and writes the status to the serial port.
* This example works with the Wiring / Arduino program that follows below.
*/

// Wiring/Arduino code:
// Read data from the serial and turn ON or OFF a li

char val;        // Data received from the serial po
int ledPin = 9;  // Set the pin to digital I/O 4

void setup() {
  pinMode(ledPin, OUTPUT);   // Inizializzo il pin 9 come output
  Serial.begin(9600);        // Inizializzo la porta seriale
}

void loop() {
  if (Serial.available()) {  // Controllo che la porta sia disponibile
    val = Serial.read();     // poi leggo il valore in byte, un carattere alla volta
  }

  if (val == 'H') {          // Se il valore è H
    digitalWrite(ledPin, HIGH);  // accende il LED
  } else {                     // Se il valore è un altro
    digitalWrite(ledPin, LOW);   // spegne il LED
  }

  delay(30);   // Aspetta prima di leggere di nuovo
}
```

Per arduino utilizziamo un codice semplificato che legge un carattere alla volta con la funzione Serial.read()

Memorizzo il carattere in VAL. Dopodiché utilizzo un costrutto IF. Se il carattere memorizzato è H accendo il Led, se è altro, spengo il LED.

Risorse

Arduino

Sito: http://www.arduino.cc/

Playground: http://playground.arduino.cc/

Forum: http://forum.arduino.cc/

Simulatore: http://123d.circuits.io/

Basic Connections: http://www.pighixxx.com/

Tutorials: http://learn.adafruit.com/

Processing

Sito: http://processing.org/

Video corso: http://hello.processing.org/

Tutorials: http://processing.org/tutorials/

Libri: http://processing.org/books/

Guida introduttiva: http://bit.ly/1kLGoMg

Elettronica

Fritzing: http://fritzing.org/home/

Eagle: http://www.cadsoftusa.com/

Simulatore: http://falstad.com/circuit/

Risorse

Servizi web per Internet of Things

Paraimpu: https://www.paraimpu.com/

Temboo: https://www.temboo.com/

ThingSpeak: https://thingspeak.com/

Bibliografia minima

ARDUINO

Guida base.

Utilizzo avanzato del linguaggio e dei componenti.

Panoramica generale e di facile lettura su elettronica e componenti.

Bibliografia minima

GENERALE

FAB LAB
FROSINONE
OFFICINE GIARDINO

W: officinegiardino.org
M: info@officinegiardino.org

CC 2014 Daniele Iori e Ivan De Cesaris
per Officine Giardino
Quest'opera è distribuita con Licenza Creative Commons
Attribuzione - Condividi allo stesso modo 4.0 Internazionale.
link: creativecommons.org/licenses/by-sa/4.0/