Arduino Tutorial 1: Primi passi

Per Arduino Software (IDE) / Arduino Web Editor

Release 1.1
31/03/2017

Muoviamo i primi passi con Arduino!

Con questo tutorial saremo noi a costruire i circuiti per i nostri progetti: certo bisogna fare molta più attenzione di quando si usano degli shield come quello di TinkerKit, ma capiremo meglio come funzionano i componenti elettrici e Arduino stesso.

Trovi questo tutorial, tutti gli schemi elettrici ed altre informazioni sul nostro sito:

coderdojotrento.it/arduino1

1. Alcune note, prima di cominciare

1) ricordarsi che per poter funzionare, ogni circuito deve essere “chiuso”: è proprio come se fosse uno scivolo, e non si può sperare di arrivare in fondo alla discesa, se lo scivolo è interrotto in qualche punto;

2) gli elettroni (piccolissime particelle cariche che si trovano in tutti gli atomi) sono degli eccellenti scivolatori: se prepariamo un circuito costituito di materiale che li lascia passare -come dei fili di metallo o altro materiale conduttore- questi si muoveranno attraverso i fili.
Gli elettroni in movimento determinano la corrente elettrica. Vedi nota 1

3) ogni circuito deve essere alimentato, cioè ci deve essere qualcosa che mantiene la pendenza: su uno scivolo appoggiato orizzontalmente a terra, non si diverte nessuno! Chi compie il lavoro di riportare in alto le cariche per farle scivolare ancora verso il basso è l’alimentatore (batterie, pile, oppure il nostro Arduino);

4) ogni elemento del circuito funziona bene (e si diverte!) ad una specifica pendenza, quindi dobbiamo scegliere con attenzione gli elementi da collegare per evitare che gli scivolatori si brucino il sedere, che l’alimentatore faccia troppa fatica nel rimettere in circolo gli elettroni o che le sezioni del nostro scivolo si surriscaldino perché attraversate da troppa corrente;

5) in questo tutorial lavoreremo sempre con circuiti che **non sono pericolosi** per noi (sono alimentati con tensioni intorno ai 5 Volt, simili a quelle delle porte USB del nostro computer): stiamo alla larga invece da altri fili elettrici, che possono produrre correnti pericolose!

CIRCUITO CHIUSO e CONDUTTORI

CARICHE IN MOVIMENTO = CORRENTE [Ampere]

DISLIVELLO = DIFFERENZA di POTENZIALE [Volt]

ALIMENTATORI, PILE ed altri ELEMENTI dei CIRCUITI ELETTRICI

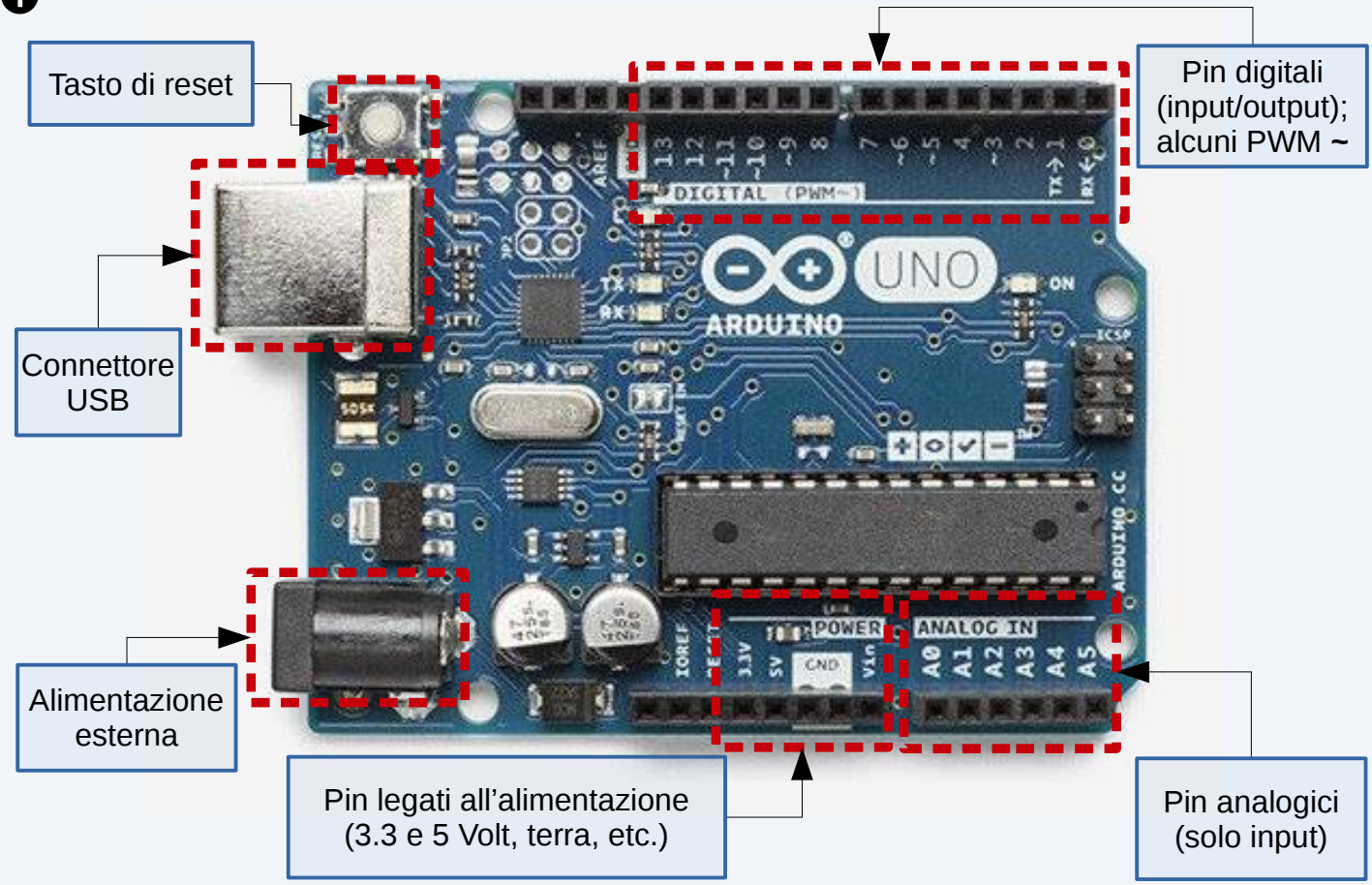
TENSIONI di 5 V: PER NOI NON SONO PERICOLOSE MA STIAMO ATTENTI!

Nota 1 – Per essere precisi, la carica degli elettroni è negativa: per questa loro caratteristica, hanno la capacità di … scivolare in salita! Così, anche se diciamo che la corrente elettrica scorre dal punto più in alto a quello più basso, loro vanno nella direzione contraria.

2. Piacere, sono Arduino!

Guardiamo con attenzione la nostra scheda Arduino UNO (vedi figura ❶): sul lato di sinistra è facile riconoscere il tasto di reset, il connettore USB che attaccheremo al nostro computer e lo spinotto per l'alimentazione esterna.
Sui due lati lunghi ci sono due serie di pin (connettori) che potremo usare per alimentare i nostri componenti elettrici o per leggere i valori di tensione dai nostri sensori: ricordati infatti che potremo collegare dei componenti in modo che funzionino come attuatori o come sensori, e questi potranno essere analogici o digitali (vedi prospetto nella figura ❷).

❶



❷

Tipi di componenti

1) di input (sensore) oppure di output (attuatore);

2) digitale (acceso/spento) oppure analogico (valori continui).

3. Il primo circuito: accendiamo un LED

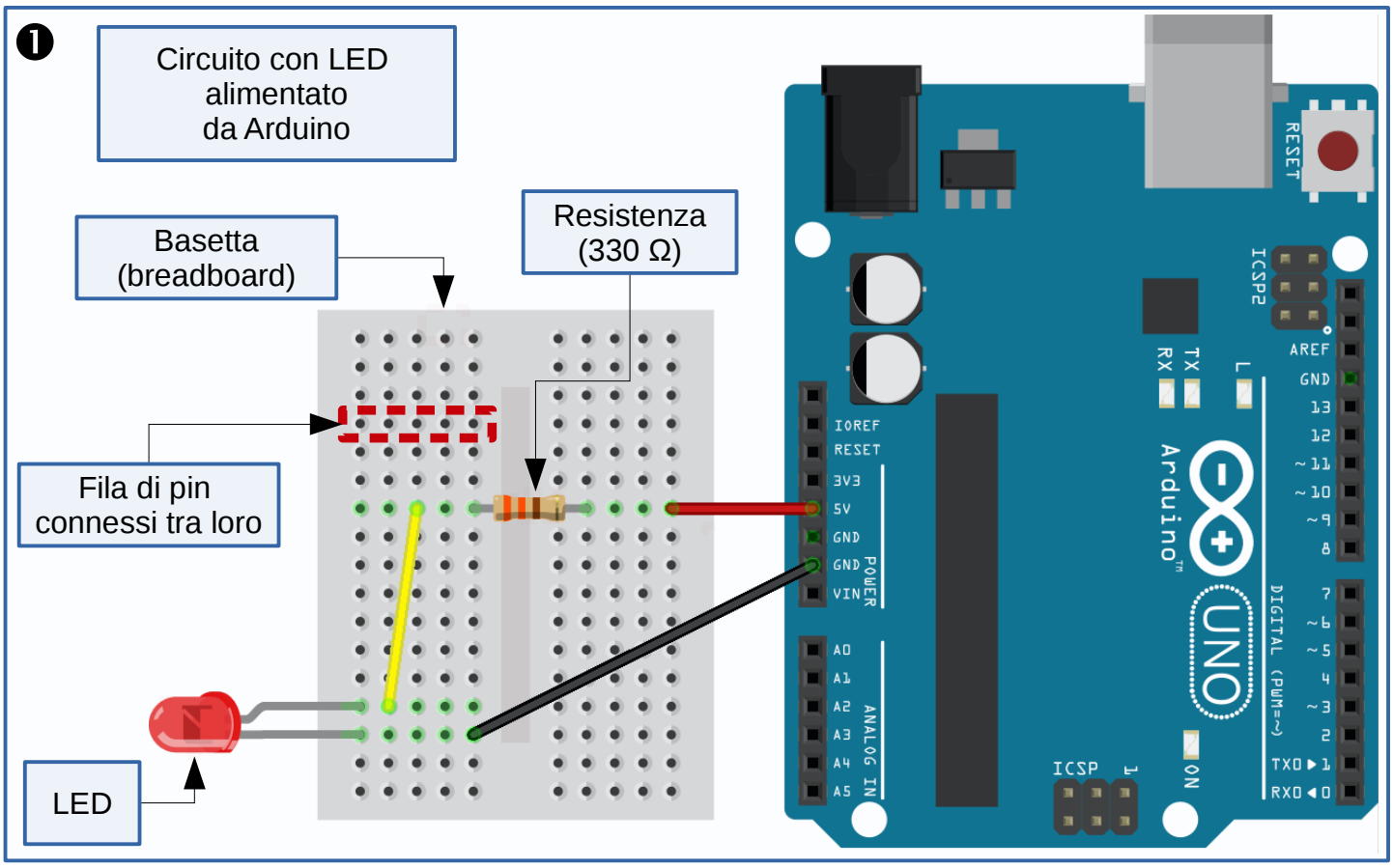
Per ora utilizziamo Arduino solo come alimentatore: di tutti i suoi pin usiamo infatti solo quello che fornisce i +5 Volt (segnato sulla scheda come “5V”) e quello di terra (“GND”). Li usiamo come estremi del circuito costituito dai tre cavetti di connessione, da un LED e una resistenza da 330 Ω (vedi figura ❶).

Tutti gli elementi sono applicati su una **basetta** (*breadboard*, in inglese) che permette di costruire rapidamente dei circuiti semplicemente inserendo le estremità dei componenti nei fori presenti sulla base di plastica: ciascuna fila di 5 fori è infatti collegata con del materiale conduttore, quindi è come se ci fossero dei cavetti sotto la plastica che collegano tutti i componenti che hanno le estremità infilate in una stessa fila.

In questo caso la corrente dentro il nostro circuito parte “in alto” dal punto che si trova a +5 Volt su Arduino, prosegue lungo il filo rosso, dentro la basetta (settima fila di destra) arriva all’estremità destra della resistenza, la attraversa, passa dal filo giallo, attraversa il LED ed arriva “a terra” (*ground*, in inglese) attraverso il filo nero.

Ed ora collega il cavo USB al computer, in modo che scheda e circuito risultino alimentati!

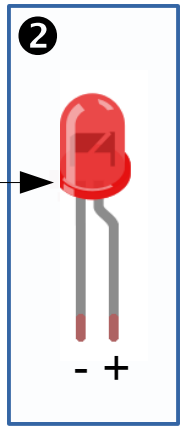
❶



Alcune note su questo primo semplice ma istruttivo circuito:

- il LED (“*light emitting diode*”: diodo che emette luce) come tutti i diodi funziona soltanto se la corrente lo attraversa in una particolare direzione: dobbiamo collegare l’estremità più lunga alla parte del circuito col voltaggio più alto (vedi figura ❷); l’altra estremità è segnata con una zona piatta nel cappuccio di plastica.
- la resistenza in serie con il LED serve per limitare un poco la corrente: il “dislivello” di 5 Volt è troppo alto per il LED, quindi lasciamo che gli elettroni perdano un po’ di energia lungo la discesa passando attraverso questo componente passivo (è semplicemente una strozzatura nel circuito che rallenta un po’ la discesa);
- la resistenza da 330 Ω (si legge *ohm*) è proprio quella giusta per far lavorare il LED al corretto voltaggio: se l’avessimo scelta più grande, l’avremmo “protetto” di più ma la luce prodotta dal LED sarebbe stata molto fioca.

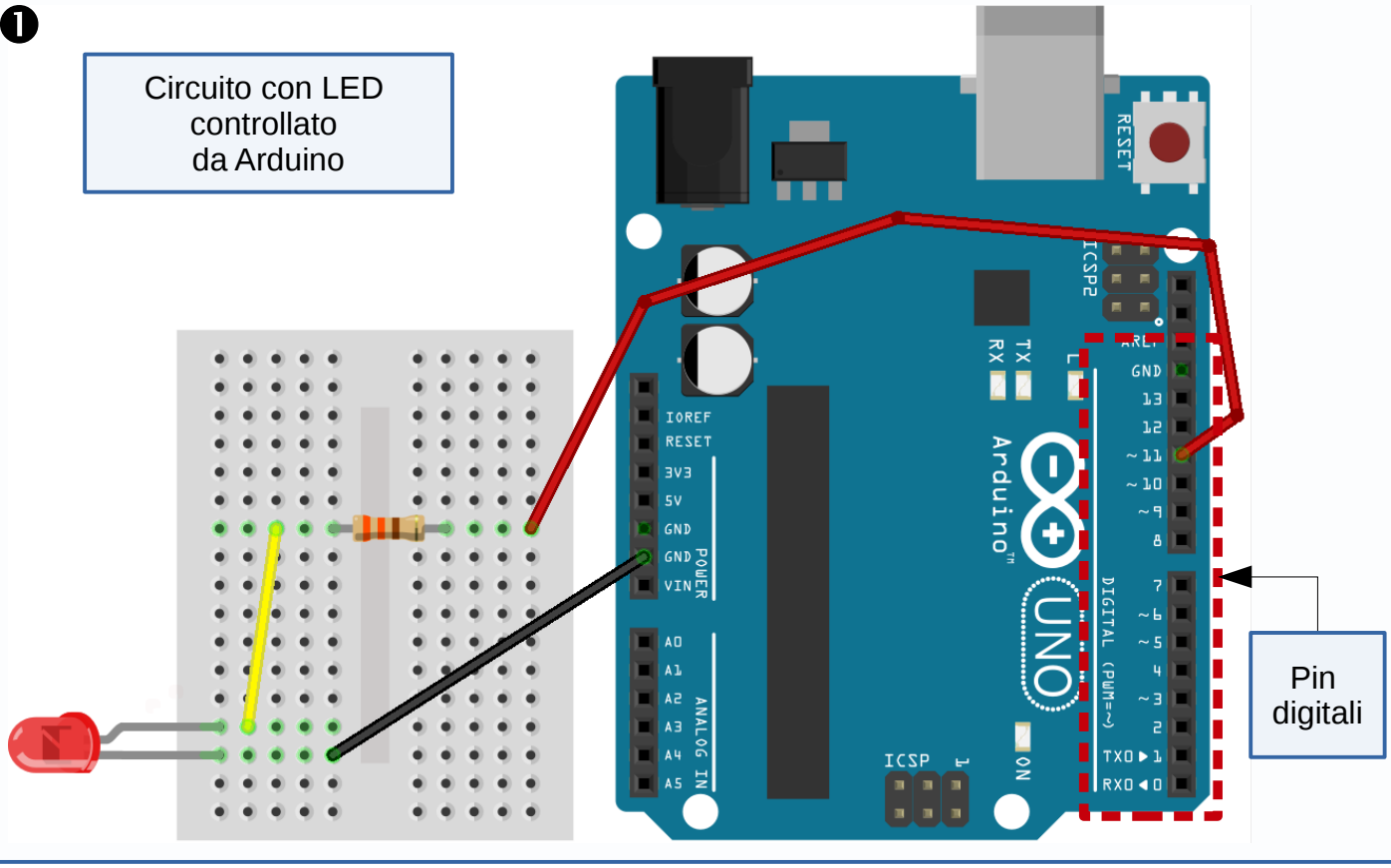
❷



4. Il secondo circuito: controlliamo un LED

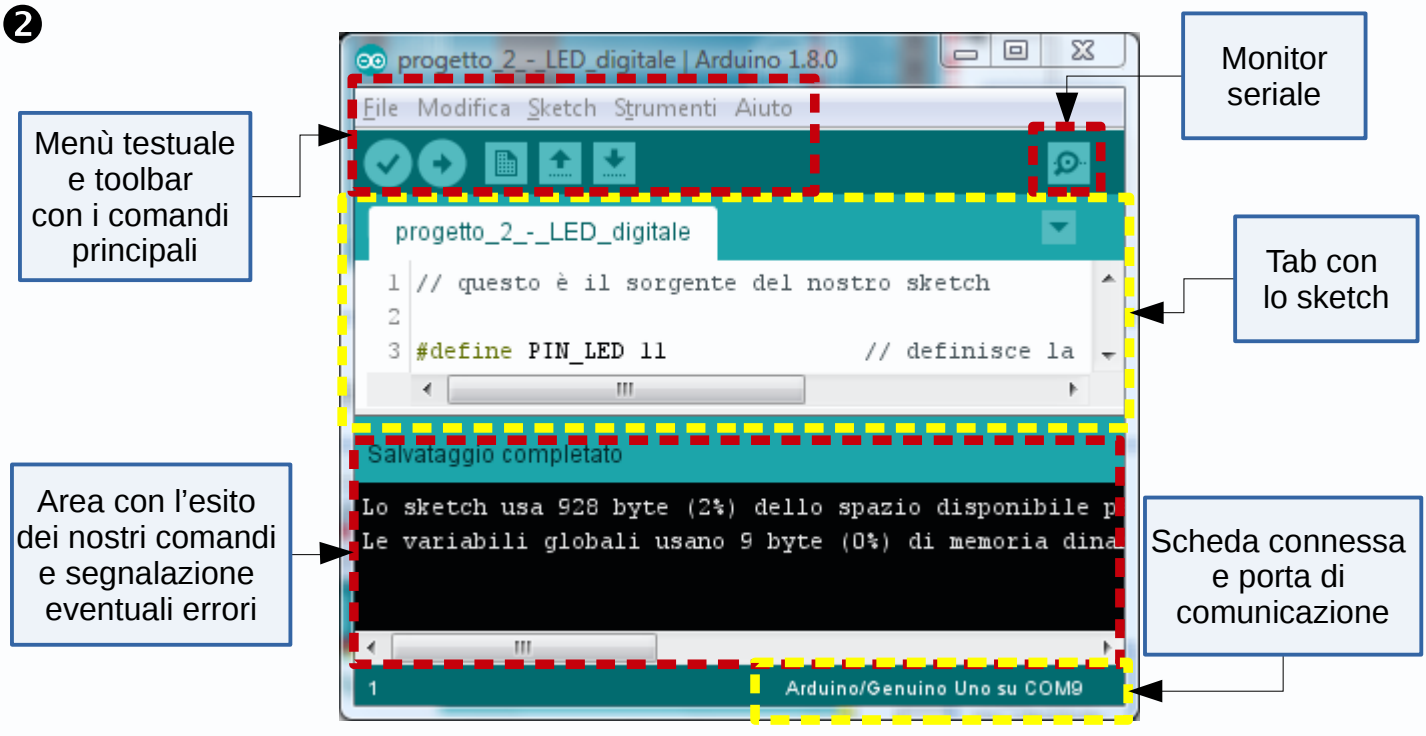
I componenti del secondo circuito sono gli stessi di prima; stavolta però li colleghiamo a due pin di Arduino che ci permetteranno di controllare l'accensione e lo spegnimento del LED utilizzando un programma.
Il cavetto nero rimane attaccato a terra ("GND"), mentre il cavetto rosso si sposta sul pin numero 11 della fila di destra: è uno dei pin digitali di Arduino (vedi figura ❶).

❶



Facciamo partire l'IDE di Arduino e familiarizziamo con l'interfaccia (vedi figura ❷).

❷



L'IDE (*Integrated Development Environment* = Ambiente integrato di sviluppo) di Arduino è il programma installato sul computer grazie al quale prepareremo i progetti che controlleranno i nostri circuiti: il cuore del progetto è un file di testo che contiene tutte le istruzioni che devono essere eseguite da Arduino, proprio come in Scratch abbiamo gli script dei vari personaggi. Ciascuno di questi programmi si chiama *sketch* ed è salvato da Arduino IDE in file che hanno estensione .INO; la sintassi del linguaggio di programmazione utilizzato è molto simile a quella di C/C++ ma gli sketch sono molto molto semplici dato che è necessario realizzare solo due funzioni: la prima si chiama **setup()** e viene invocata una volta sola all'inizio del programma; la seconda invece si chiama **loop()** e viene invocata ciclicamente, fino a che la nostra scheda viene alimentata.

Facendo un parallelo con uno script in Scratch, possiamo immaginare che il comportamento di Arduino sia come quello indicato in figura ❶: quando il programma viene caricato sulla scheda, viene per prima cosa eseguita la funzione **setup()**, quindi -all'interno di un ciclo infinito- la funzione **loop()**.
Nella prima funzione inseriremo quindi tutti i comando che riguardano la preparazione della scheda, la scelta dei pin di lavoro e la loro attivazione; nella seconda funzione metteremo le istruzioni che devono essere eseguite continuamente, per tutta la durata del programma: lettura dei sensori, calcoli su tali valori, comandi per controllare gli attuatori, etc.

❶ Schematizzazione sketch Arduino

A proposito: anche Arduino, come Scratch, ha una versione che funziona off-line (*Arduino IDE* installata sul computer, appunto) ed una versione on-line. Si chiama *Arduino Web Editor* e la troviamo all'indirizzo https://create.arduino.cc/: una volta registrata la nostra utenza, potremo salvare in rete i nostri progetti ed accedere ad essi da qualunque computer connesso ad Internet.

Vediamo ora in dettaglio lo sketch che controlla l'accensione e lo spegnimento del LED (figura ❷): c'è un abbondante uso di commenti (la parte di testo che comincia con "//") che vengono ignorati da Arduino e quindi non cambiano il funzionamento del programma, ma che permettono di aggiungere spiegazioni o memo all'interno del codice (importantissime, non solo per gli altri, ma anche per noi stessi!).

❷ Sketch per il LED digitale

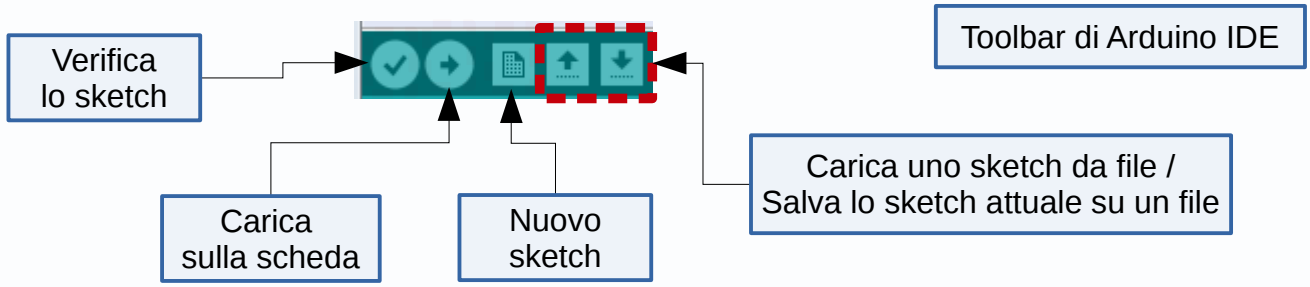
```
1  // questo è il sorgente del nostro sketch
2
3  #define PIN_LED 11              // definisce la costante PIN_LED valorizzata a 11
4
5  void setup() {
6    pinMode(PIN_LED, OUTPUT);     // predispone il pin digitale 11 per l'output
7  }
8
9  void loop() {
10   digitalWrite(PIN_LED, HIGH);  // accende il LED (HIGH è il voltaggio di 5 V)
11   delay(1000);                  // aspetta un secondo (1000 millisecondi)
12   digitalWrite(PIN_LED, LOW);   // spegne il LED (LOW è il voltaggio pari a 0 V)
13   delay(1000);                  // aspetta un altro secondo
14 }
```

Per ricordarci il pin al quale abbiamo collegato il LED definiamo una costante PIN_LED (a riga 3): utilizziamo questo valore sia nel setup() quando dichiariamo che lavoreremo con quel pin usandolo come un output digitale (riga 6), sia nel loop() quando portiamo il voltaggio sul pin a 5 Volt (riga 10) ed a 0 Volt (riga 12).
Ricordati le parentesi graffe che racchiudono il corpo delle funzioni e le parentesi rotonde per elencare gli argomenti delle funzioni (nel caso di setup e loop, non ci sono argomenti, ma le parentesi vanno comunque messe). Tieni presente che ogni istruzione è terminata da un ";" e presta attenzione a maiuscole e minuscole (il compilatore degli sketch di Arduino è *case sensitive*).

Siamo pronti per verificare la correttezza del codice utilizzando il simbolo della spunta nella toolbar dell'interfaccia (vedi figura ❶): eventuali errori presenti nel codice sono riportati in rosso nell'area della finestra in basso (vedi figura ❷ di pagina 4).
Se la verifica termina correttamente, in basso potrai leggere alcune informazioni sulla dimensione dello sketch (Arduino ha una memoria molto piccola, quindi bisogna risparmiare spazio!) e comparirà la scritta "Compilazione completata".

Appena colleghiamo la scheda alla porta USB, Arduino IDE la riconosce e la segnala nella parte in basso a destra della finestra (se il riconoscimento non avviene in automatico, dobbiamo scegliere noi la scheda e la porta utilizzando la voce "Strumenti" del menù). Una volta stabilita la connessione, possiamo caricare il programma utilizzando la seconda icona della toolbar: se tutto va bene, i LED con nome TX/RX presenti sulla scheda dovrebbero lampeggiare rapidamente e quindi il nostro sketch dovrebbe partire. Con quest'ultima operazione l'abbiamo infatti caricato nella memoria del processore di Arduino e ne abbiamo avviato l'esecuzione.

❶



5. Funziona tutto? Bene, allora … rompi qualcosa!

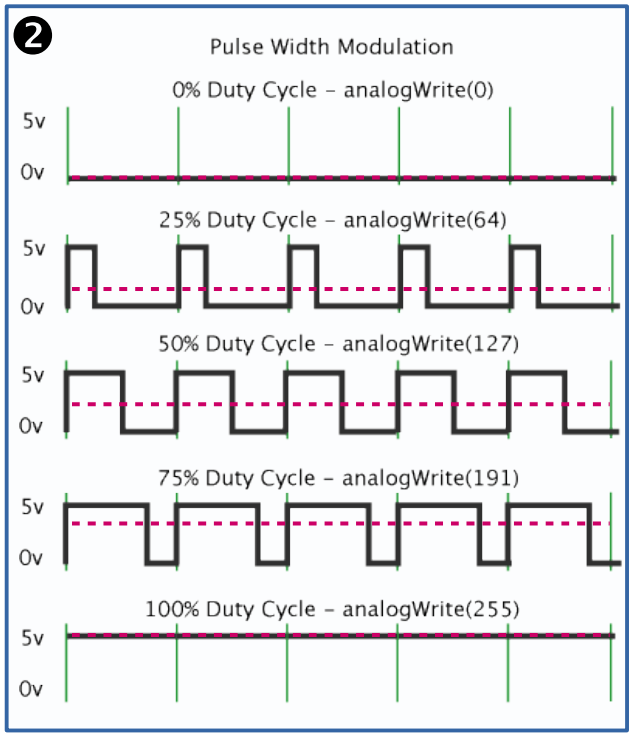
Se sei riuscito a superare la verifica di correttezza dello sketch, complimenti! Significa che sei molto attento ai particolari. Prova però ad introdurre apposta degli errori, per vedere come si comporta Arduino IDE al momento della verifica dello sketch.
Certe volte le spiegazioni che riporta sono chiare (prova a togliere un ";", oppure a riga 12 prova a scrivere ad esempio "digitalwrite" ignorando la maiuscola: l'errore dice esplicitamente che hai usato una parola che non conosce e che non è stata dichiarata), altre volte -soprattutto quando ci scordiamo delle parentesi- gli errori possono venir segnalati parecchie righe più in basso.

6. Sempre lo stesso LED, ma stavolta *in analogico*

Cambiamo ora il comportamento del nostro LED, modificando soltanto lo sketch: il circuito che abbiamo preparato può infatti andare bene anche per controllare il LED come se fosse un **attuatore analogico**.
Non ci limiteremo più alle tipiche azioni che si possono compiere con un componente digitale (accenderlo portando il voltaggio sul pin a 5 Volt, o spegnerlo mettendo il voltaggio a 0 Volt), ma ne cambieremo la luminosità utilizzando una gamma continua di valori di tensione (il dislivello del nostro scivolo).

Per ottenere questa serie di valori dobbiamo utilizzare la funzione **analogWrite** su un canale digitale in uscita, ma bisogna sceglierne uno che abbia il simboletto **~** a fianco (ovvero quelli dei pin 3, 5, 6, 9, 10 o 11): questi pin sono i **PWM** (*Pulse Width Modulation* = *Modulazione di larghezza d'impulso*) ed in uscita producono un'onda quadra compresa sempre tra 0 e 5 Volt, ma la larghezza dell'impulso determina in uscita un **voltaggio medio** che può assumere anche i valori intermedi (vedi righe ----- nella figura ❷).

❷



Utilizziamo lo sketch riportato in figura ❶: a riga 13 ed a riga 20 si nota l'utilizzo della funzione **analogWrite** con i due argomenti che indicano il pin sul quale settare il voltaggio in uscita ed un numero intero compreso tra 0 e 255 (come indicato nel prospetto nella figura ❷ della pagina precedente, il valore 0 corrisponde a 0 Volt, il valore 255 a 5 Volt ed i valori intermedi producono voltaggi proporzionali a questi due voltaggi minimo e massimo).
Per costruire la rampa crescente (righe 12-15) e quella discendente (righe 19-22) abbiamo utilizzato due **cicli 'for'**: ricordano un po' i blocchi "*ripeti N volte*" di Scratch, ma sono molto più ricchi, dato che permettono anche la definizione al volo di variabili che possono essere utilizzate proprio per scandire i passi all'interno del ciclo (nello sketch, le variabili di nome "valore").

❶ Sketch per il LED analogico

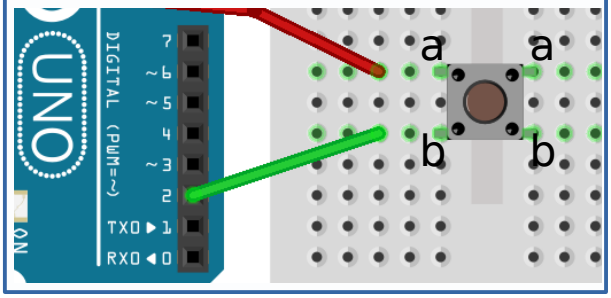
```
1  // questo è il sorgente del nostro secondo sketch: "LED analogico"
2
3  #define PIN_LED 11              // definisce la costante PIN_LED valorizzata a 11
4
5  void setup() {
6    pinMode(PIN_LED, OUTPUT);     // predispone il pin 11 per l'output
7  }
8
9  void loop() {
10   /*  Effettua un ciclo, appoggiandosi ad una variabile intera di nome "valore" che parte da 0 e si
11    *  incrementa ad ogni passo di 5 unità; il ciclo continua finchè "valore" è minore o uguale a 255  */
12   for (int valore = 0; valore <= 255; valore += 5) {
13     analogWrite(PIN_LED, valore);  // scrive sul pin del LED un voltaggio proporzionale a "valore"
14     delay(20);                     // aspetta un cinquantesimo di secondo (20 millisecondi)
15   }
16
17   /*  Effettua un secondo ciclo, stavolta con la variabile "valore" che scende da 255 a 0 decrementando
18    *  di volta in volta di 5 unità; il ciclo continua finchè "valore" è positivo */
19   for (int valore = 255; valore >= 0; valore -= 5) {
20     analogWrite(PIN_LED, valore);  // scrive sul pin del LED un voltaggio proporzionale a "valore"
21     delay(20);                     // aspetta un cinquantesimo di secondo (20 millisecondi)
22   }
23 }
```

7. Passiamo agli input: il digitale

In questo progetto utilizzeremo il sensore più semplice: un **bottone**. È di sicuro un sensore digitale, perché potrà essere posizionato in due soli stati: aperto o chiuso (visto che sarà inserito all'interno di un circuito, significa che metterà o meno in contatto due parti del circuito, chiudendo quindi un percorso). I bottoni più comuni sono come quello della figura ❷: hanno quattro piedini che sono già collegati a due a due (nel disegno, quelli sulla stessa riga, caratterizzati dalla stessa lettera **a** oppure **b**), e -quando si preme il pulsante- quelli sopra e quelli sotto si mettono in contatto.

Per leggere lo stato del bottone, predisporremo uno dei pin digitali (ad esempio il numero 2) in modalità di acquisizione grazie all'istruzione **pinMode(2, INPUT)** e poi leggeremo la tensione presente sul pin utilizzando l'istruzione **digitalRead(2)**: questa funzione restituirà un valore intero che può essere *HIGH* oppure *LOW* (ovvero il pin in oggetto si trova in prossimità del punto più alto dello scivolo, o viceversa sarà vicino a terra: cioè a dire, il voltaggio del pin sarà vicino a 5 Volt oppure a 0 Volt).

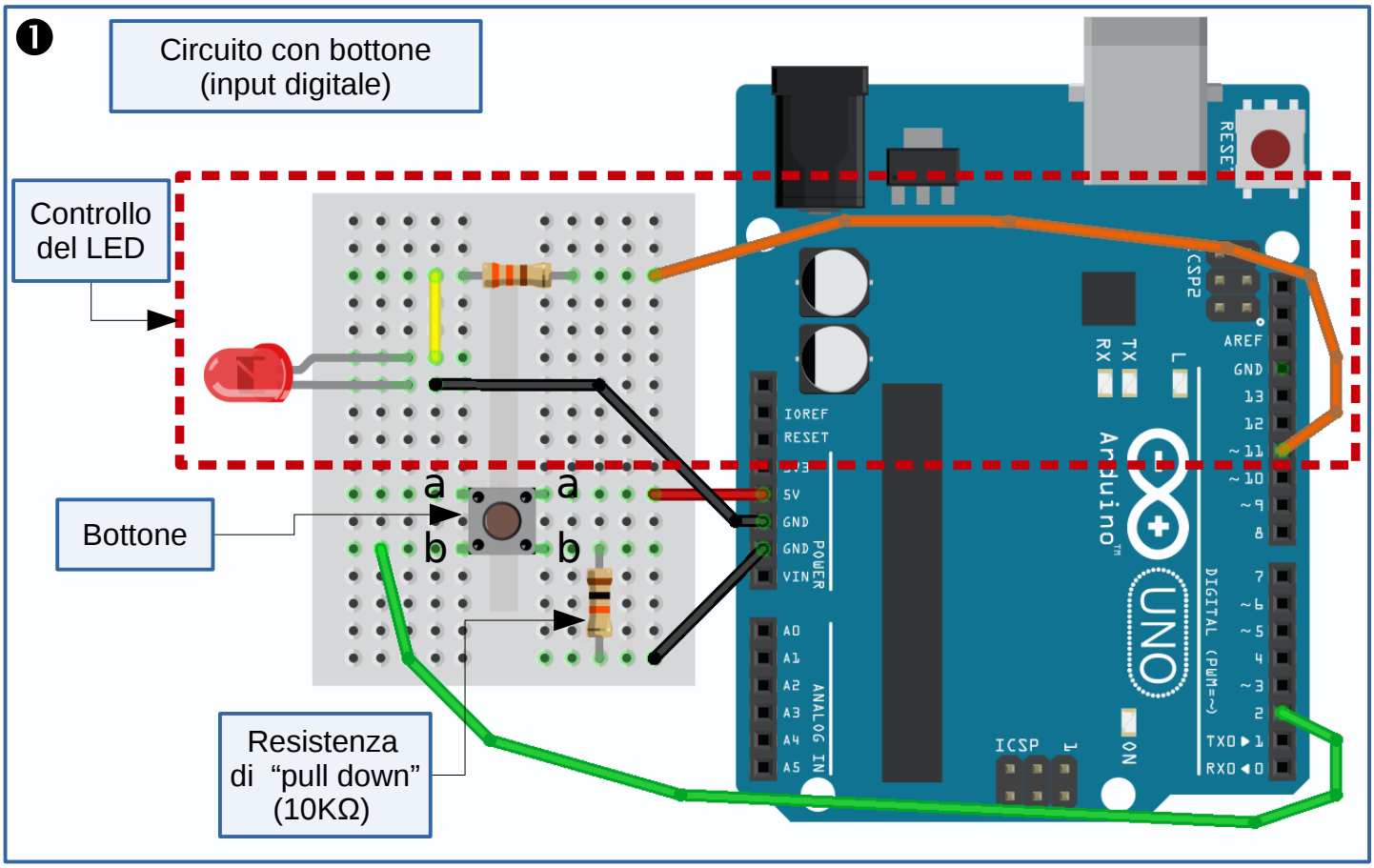
❷ Bottone collegato **male**



Diciamo subito che il primo tipo di circuito che ci viene in mente di provare (ovvero quello descritto in figura ❷) non funziona: se portiamo i 5 Volt col cavetto rosso su un lato **a** del bottone e proviamo a collegare l'altra estremità **b** direttamente sul pin che andremo a leggere (cavetto verde), otterremo spesso letture casuali.
Chiudendo l'interruttore anche il cavetto verde andrà a 5 Volt e leggeremo *HIGH*; ma quando l'interruttore è aperto, il filo verde è **volante**: non è collegato cioè a nessun circuito, quindi su questa estremità si può generare del rumore e il valore letto sarà indefinito.

Il modo giusto per utilizzare un bottone è quello descritto nella parte bassa dello schema riportato in figura ❶. Non spaventiamoci per tutti quei fili: della parte alta sappiamo già tutto! È infatti il circuito che serve per controllare il LED che abbiamo descritto nella figura ❶ di pagina 4.
La parte nuova si trova nella zona bassa della basetta: abbiamo arrangiato in modo un po' diverso i contatti del bottone, ed abbiamo aggiunto una resistenza che chiude a terra il circuito del filo verde dell'input digitale (pin 2): in questo modo il contatto non è più *volante*, e -quanto l'interruttore è aperto- leggiamo un chiaro *LOW*. Viceversa, quando premiamo il pulsante, i punti **a** e **b** si mettono in contatto, così anche **b** arriverà ai 5 Volt del cavetto rosso: il valore letto sarà quindi *HIGH*.

❶ Circuito con bottone (input digitale)



Lo script per controllare questo circuito si trova nella figura ❷ qui sotto.

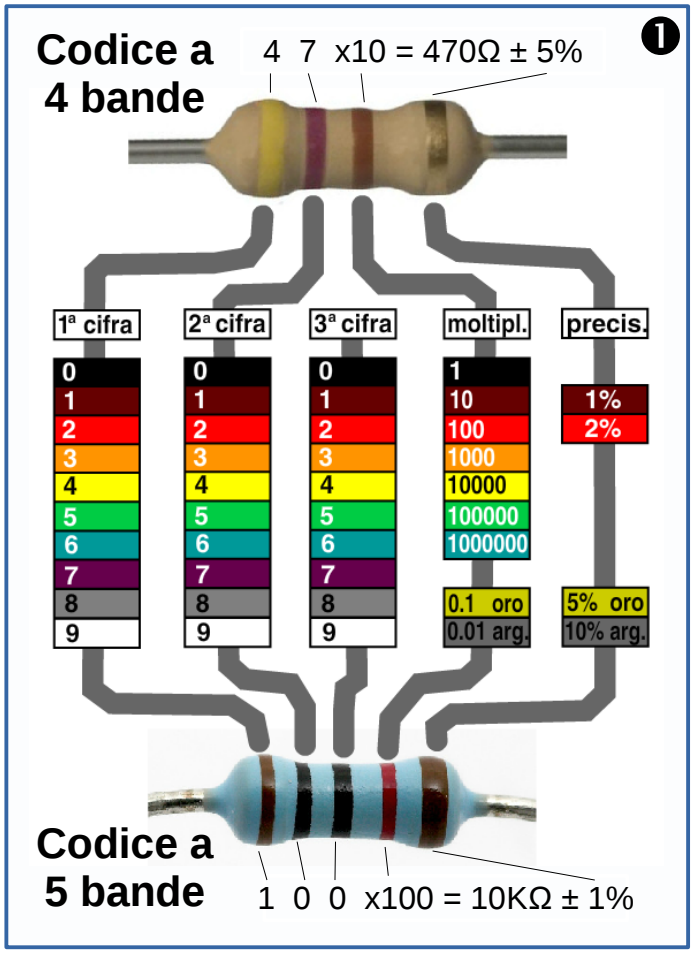
❷ Sketch per la lettura dello stato di un bottone

```
// definiamo ed inizializziamo due costanti di tipo intero per memorizzare i valori dei pin usati:
const int pinBottone = 2;      // il numero del pin al quale abbiamo collegato il bottone
const int pinLED = 11;         // il numero del pin al quale abbiamo collegato il LED

// questa invece è una variabile (nel corso del programma, può cambiare valore):
int statoBottone = LOW;        // variabile per leggere lo stato (premuto/sollevato) del bottone

void setup() {
  pinMode(pinLED, OUTPUT);     // inizializza il pin del LED come un output
  pinMode(pinBottone, INPUT);  // inizializza il pin del bottone come un input
}

void loop() {
  statoBottone = digitalRead(pinBottone);  // legge lo stato del bottone e lo assegna alla variabile

  // controlliamo ora se il pulsante è premuto...
  if (statoBottone == HIGH) {     // se lo è, la variabile statoBottone è HIGH:
    digitalWrite(pinLED, HIGH);   // accendi il LED
  } else {                        // altrimenti
    digitalWrite(pinLED, LOW);    // spegni il LED
  }
}
```

LE RESISTENZE

Prestiamo attenzione alla resistenza che abbiamo aggiunto vicino al bottone: questa ha un valore abbastanza "alto" (10KΩ, ovvero 10'000 ohm) e serve appunto per "portare a terra" quella parte di circuito che prima era volante: si chiama infatti **resistenza di "pull down"** (ovvero "che tira giù"). Se l'avessimo collegata al bottone in modo che il cavetto volante verde del sensore fosse portato ai +5 Volt dell'alimentatore, l'avremmo chiamata **resistenza di "pull up"** (useremmo quella configurazione nel caso volessimo un circuito che sta normalmente a livello *HIGH* e che viene portato a *LOW* nel momento della pressione del pulsante).

Ma come facciamo, per scegliere la resistenza da utilizzare in un circuito? Siccome è il componente che limita il passaggio della corrente, la scelta dipende dall'intensità di corrente che desideriamo nel circuito.
Per limitarla un poco e per proteggere il LED come nelle pagine precedenti, basta una resistenza di poche centinaia di ohm; se invece non dobbiamo pilotare alcun componente che dissipa energia (e che quindi necessita di corrente), ma ci vogliamo concentrare sulla misura del voltaggio, possiamo anche prendere resistenze ben più elevate (decine di migliaia di ohm).
E per capire il valore di una resistenza? Utilizziamo lo schema dei colori illustrato nella figura ❶: se la resistenza ha 4 bande colorate, due rappresentano le prime cifre, una è un moltiplicatore e l'ultima rappresenta la precisione della grandezza. Se invece le bande sono 5, la prima tre sono le cifre, quindi abbiamo moltiplicatore e precisione.
Per determinate la direzione di lettura delle bande colorate, ci possiamo basare sui colori (quelli della precisione sono limitati) oppure notiamo che il gruppo delle prime tre bande è distanziato dalle strisce di colore finali.

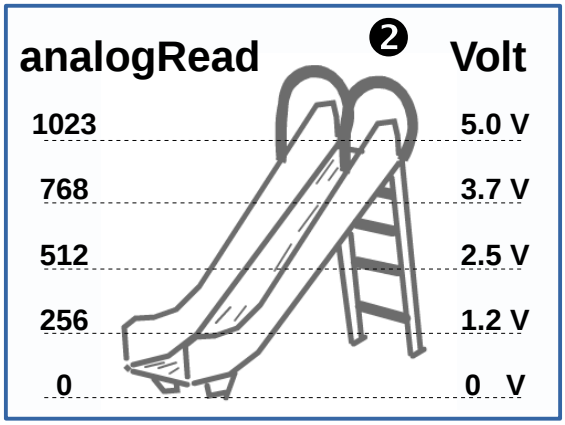


8. Gli input analogici: il potenziometro

Per avere in ingresso dei valori letti da un input analogico dobbiamo semplicemente:

- scegliere uno dei pin marcati come **A0**, **A1**, … **A5** (quelli che stanno appunto nell'angolo di Arduino dedicato agli "ANALOG IN": vedi zona in basso a destra della figura ❶ di pagina 2);
- ricordarci che la grandezza che viene letta in ingresso è sempre un voltaggio (nel parallelo dello scivolo, un cavetto collegato ad un input analogico permette di capire a che "altezza" si trova l'estremità del filo).

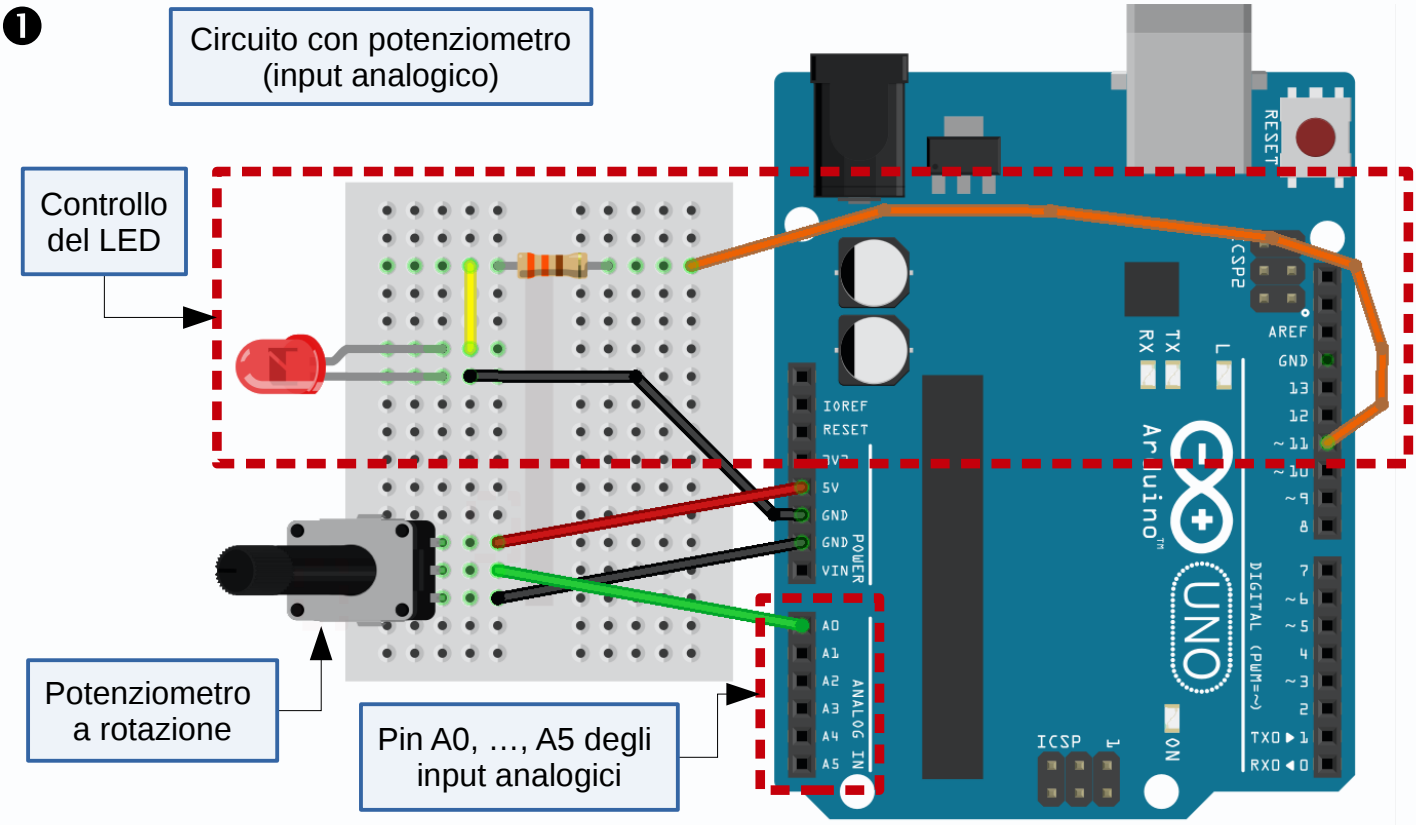
Come l'input digitale può leggere dei valori *HIGH* o *LOW* (ad indicare che il cavetto è in contatto con un punto vicino alla cima dello scivolo, o viceversa vicino al fondo), così l'input analogico potrà dirci non solo se il cavetto è ad uno dei due estremi, ma ci dirà anche in che posizione precisa ci troviamo anche se siamo ad un'altezza intermedia: grazie alla funzione **analogRead(A0)** leggeremo 1023 quando siamo in cima allo scivolo (ovvero a 5 Volt), otterremo invece 0 se siamo in fondo allo scivolo (cioè a terra), circa 512 se siamo a metà altezza, 256 se siamo ad un quarto, e così via, in proporzione (vedi figura ❷).



Il potenziometro di solito è schematizzato con il simbolo qui a fianco: si tratta infatti di una resistenza molto grande (dai 10K ai 100K ohm) che viene alimentata dai 5 Volt di Arduino e che lascia passare poca corrente; su questa resistenza va ad “appoggiarsi” un terzo contatto (di solito è il pin intermedio) che può scivolare su e giù, in base a come ruotiamo la manopola.
Questo cursore (la freccia del disegno) è collegato al sensore analogico che andrà quindi a “pescare” la tensione su diverse posizioni; la funzione **analogRead()** converte poi il voltaggio in una serie di valori da 0 a 1023.

Il circuito che useremo è quello descritto nella figura ❶: in alto riconosciamo il solito circuito che serve per controllare il LED collegato al pin numero 11. In basso invece vediamo il potenziometro con il pin intermedio collegato ad **A0** tramite il cavetto verde.

❶



In figura ❷, lo script per controllare questo circuito. Nel metodo loop() notiamo l’acquisizione del valore grazie all’**analogRead** (riga 14) ed il suo impiego per pilotare il LED (riga 16): la moltiplicazione per il fattore 255.0/1023.0 serve per convertire la gamma di valori letti (da 0 a 1023) in valori utili per l’**analogWrite** (che accetta numeri da 0 a 255).

❷ Sketch per la lettura della tensione di un potenziometro

```
1  // definiamo ed inizializziamo due costanti di tipo intero per memorizzare ...
2  const int pinPot = A0;      // ... il numero del pin al quale abbiamo collegato il potenziometro
3  const int pinLED = 11;      // ... il numero del pin al quale abbiamo collegato il LED
4
5  // questa invece è una variabile (nel corso del programma, può cambiare valore):
6  int valorePot = 0;          // variabile che ospita il valore letto dal potenziometro
7
8  void setup() {
9    pinMode(pinLED, OUTPUT);  // inizializza il pin del LED come un output
10   // N.B. Non serve inizializzare il pin A0 come input, dato che gli analogici sono tutti solo degli ingressi
11 }
12
13 void loop() {
14   valorePot = analogRead(pinPot);  // legge il valore dal potenziometro e lo assegna alla variabile
15
16   analogWrite(pinLED, valorePot*(255.0/1023.0));  // scrive sul pin del LED un valore proporzionale a quello letto
17   delay(20);                 // aspetta un cinquantesimo di secondo (20 millisecondi)
18 }
```

9. Parliamoci!

Per quest'ultimo progetto del tutorial introduttivo non presenteremo nessun nuovo componente elettronico, ma scopriremo uno strumento utilissimo che ci permetterà di mettere in comunicazione Arduino con il nostro computer.
Useremo infatti **Serial**, una classe (ovvero un insieme di funzionalità offerte dall'ambiente di lavoro di Arduino) che è utilizzabile all'interno di ogni sketch: passando attraverso una porta seriale (ovvero il cavetto USB) potremo impiegare la classe Serial per leggere o scrivere informazioni verso/da il computer per svolgere dei compiti particolari o -più semplicemente- mandare al computer delle informazioni di *debug* quando il nostro Arduino non si comporterà come ci aspettiamo e dovremo … capire cosa sta combinando.
Per vedere Serial all'opera, va benissimo il circuito per leggere i valori dal potenziometro appena preparato: aggiungeremo solo alcune istruzioni (evidenziate in giallo) per ottenere lo sketch in figura ❶.

❶ Sketch per la lettura della tensione di un potenziometro + Uso della classe Serial

```
// definiamo ed inizializziamo due costanti di tipo intero per memorizzare ...
const int pinPot = A0;      // ... il numero del pin al quale abbiamo collegato il potenziometro
const int pinLED = 11;      // ... il numero del pin al quale abbiamo collegato il LED

// questa invece è una variabile (nel corso del programma, può cambiare valore):
int valorePot = 0;          // variabile che ospita il valore letto dal potenziometro

void setup() {
  pinMode(pinLED, OUTPUT);  // inizializza il pin del LED come un output
  Serial.begin(9600);       // apre la comunicazione seriale con il computer (alla velocità di 9600 bps)
}

void loop() {
  valorePot = analogRead(pinPot);  // legge il valore dal potenziometro e lo assegna alla variabile
  Serial.print("Valore letto dal potenziometro: ");   // scrive verso il computer la stringa riportata tra apici
  Serial.println(valorePot);       // scrive verso il computer il valore della variabile valorePot e va a capo
  analogWrite(pinLED, valorePot*(255.0/1023.0));  // scrive sul pin del LED un valore proporzionale a quello letto
  delay(20);                // aspetta un cinquantesimo di secondo (20 millisecondi)
}
```

Compiliamo e carichiamo lo sketch e -per leggere le scritte inviate da Arduino al PC- apriamo il **Monitor seriale** utilizzando l'icona in alto a destra dell'IDE di Arduino (quella col simbolo della lente d'ingrandimento, indicata nella figura ❷ di pagina 4): se la velocità di comunicazione indicata in basso a destra è la stessa specificata nell'istruzione *Serial.begin(9600)* dovremmo leggere i messaggi inviati verso il computer. Se abilitiamo lo scorrimento automatico (check in basso a sinistra) e giriamo il potenziometro, vedremo come cambiano i valori letti.

Ora che abbiamo imparato le basi, mettiamoci all'opera con i divertentissimi progetti descritti nel secondo tutorial su Arduino: buon hacking!

Trovi questo ed altri tutorial sul nostro sito: coderdojotrento.it/arduino1

Ringraziamenti

Questo tutorial:
* fa parte delle "risorse" di CoderDojo Trento,
* è stato scritto con il supporto di CoderDolomiti.

Alcune delle immagini utilizzate in questo tutorial derivano dal sito ufficiale di Arduino (arduino.cc), mentre gli schemi sono stati disegnati con il tool open source Fritzing (fritzing.org).