# Corso Arduino 2015

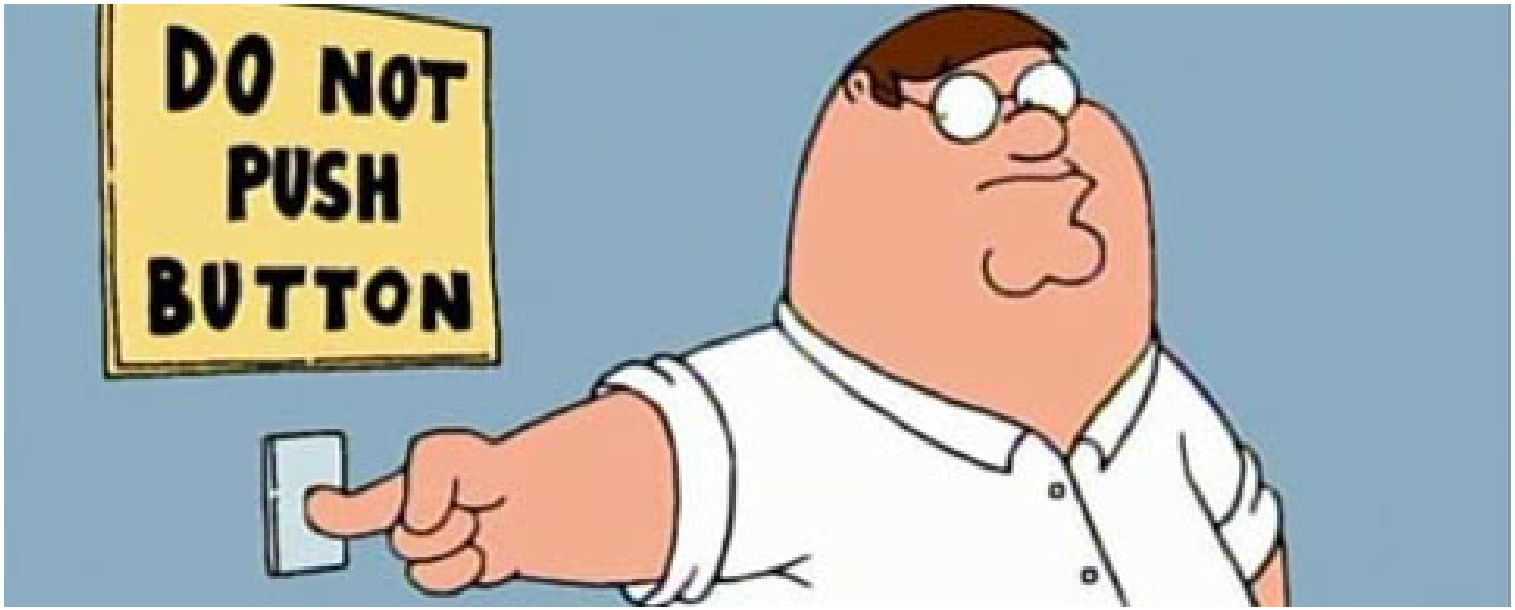
18 Maggio 2015

Riccardo Binetti
arduino@rbino.com

Lezione 1 - Blinking LEDs with style

# Fate domande

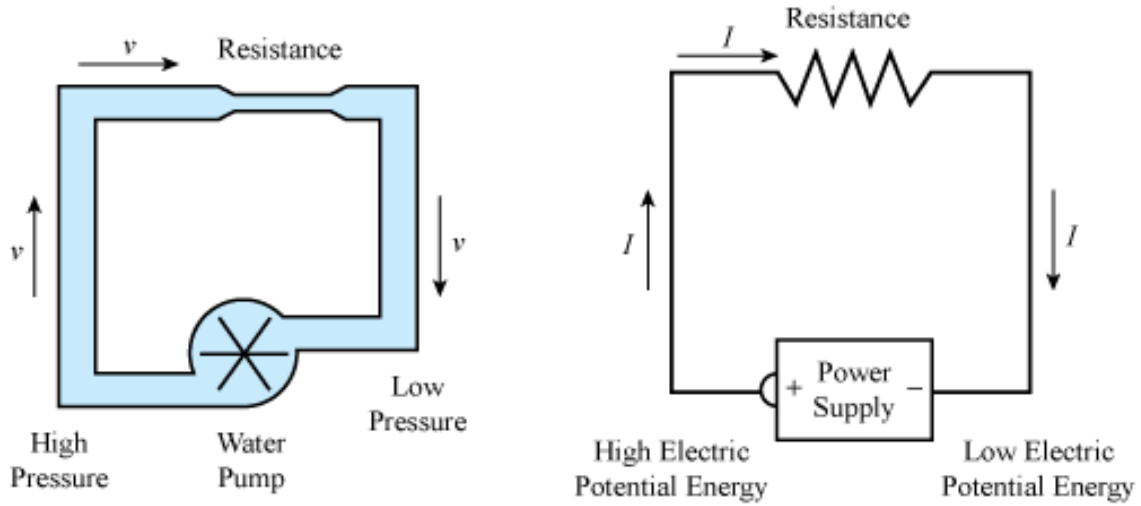
# Nel dubbio, non fatelo



# Iscrivetevi al gruppo

`http://bit.ly/arduino2015`

# Voltaggio, corrente e resistenza (metafora acquatica)



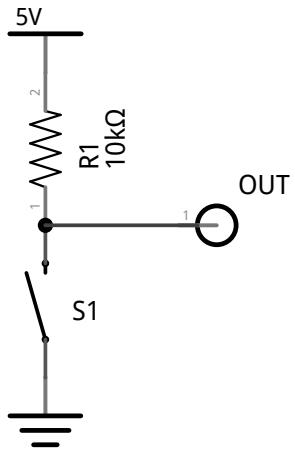

- Immaginiamo di avere un circuito chiuso d'acqua: l'acqua rappresenta gli elettroni.
- La velocità del flusso d'acqua rappresenta la corrente (simbolo I)
- La differenza di pressione tra due punti rappresenta la tensione (o differenza di potenziale) tra due punti (simbolo V)
- Una larghezza del tubo inferiore al normale rappresenta una resistenza (simbolo R)

# La legge di Ohm

$$V = R \cdot I \qquad\qquad I = \frac{V}{R}$$

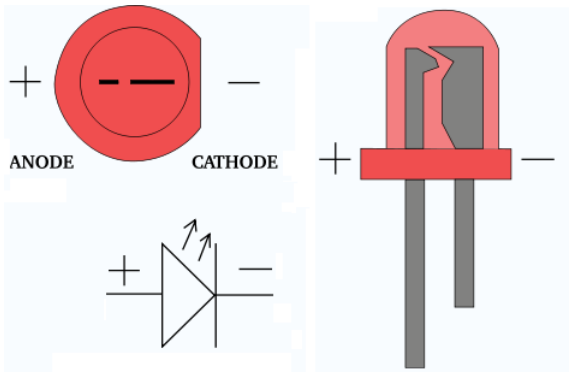
- Ci tornerà utile tra poco (ed è bene ricordarsela quando si lavora con componenti elettronici)
- A parità di differenza di potenziale (caso in cui ci si trova di solito):
  - Se aumento la resistenza, diminuirà la corrente
  - Se diminuisco la resistenza, aumenterà la corrente
- Casi limite:
  - Se la resistenza è infinita (circuito aperto) non passa corrente
  - Se la resistenza è 0 (corto circuito) passa "infinita" corrente

# Resistenza di pull-up (o pull-down)




- Serve a dare sempre un valore definito ad un'uscita collegata ad un bottone
- Se lo switch è aperto:
  - Nella resistenza non passa corrente (il circuito è aperto)
  - Quindi la differenza di potenziale ai due capi della resistenza è 0
  - Quindi l'uscita è a 5 Volts
- Se lo switch è chiuso, è in corto con GND quindi l'uscita sarà a 0 Volts

# il LED




- Il LED si accende quando l'anodo si trova almeno a ~2.0 V in più rispetto al catodo
- In soldoni, il - sarà collegato a GND e il + sarà collegato in serie ad una resistenza, che poi andrà al pin
- Quasi tutte le schede Arduino hanno un LED integrato (nel caso di Arduino Leonardo è sul pin 13), per oggi useremo quello

**Attenzione**

Non collegate mai un LED senza una resistenza in serie

# Resistenze per il LED

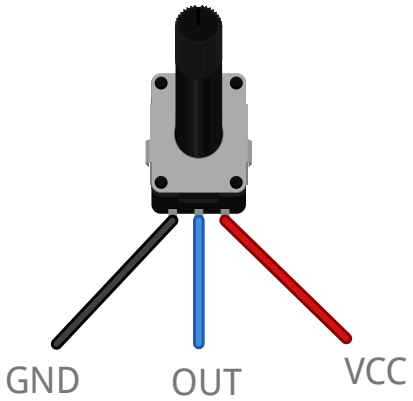
- Se volete calcolarvi la resistenza giusta per il vostro LED:
  - Controllate sul datasheet la forward current del vostro LED
  - Guardate che tensione c'è ai capi della serie di LED e resistenza (sottraendo la tensione di accensione del LED)
  - Legge di Ohm: $R = \frac{V}{I}$
- Oppure usate questa pratica tabellina per i voltaggi più comuni
  - La tabella assume un voltaggio di accensione di 2 V e una forward current di 20mA (valori dei LED rossi standard da 5 o 3 mm)

| Vcc (V) | 3.3 | 5 | 9 | 12 |
|---|---|---|---|---|
| R | 68Ω | 150Ω | 390Ω | 560Ω |

- Se non avete il valore esatto usate una resistenza più grossa, non più piccola

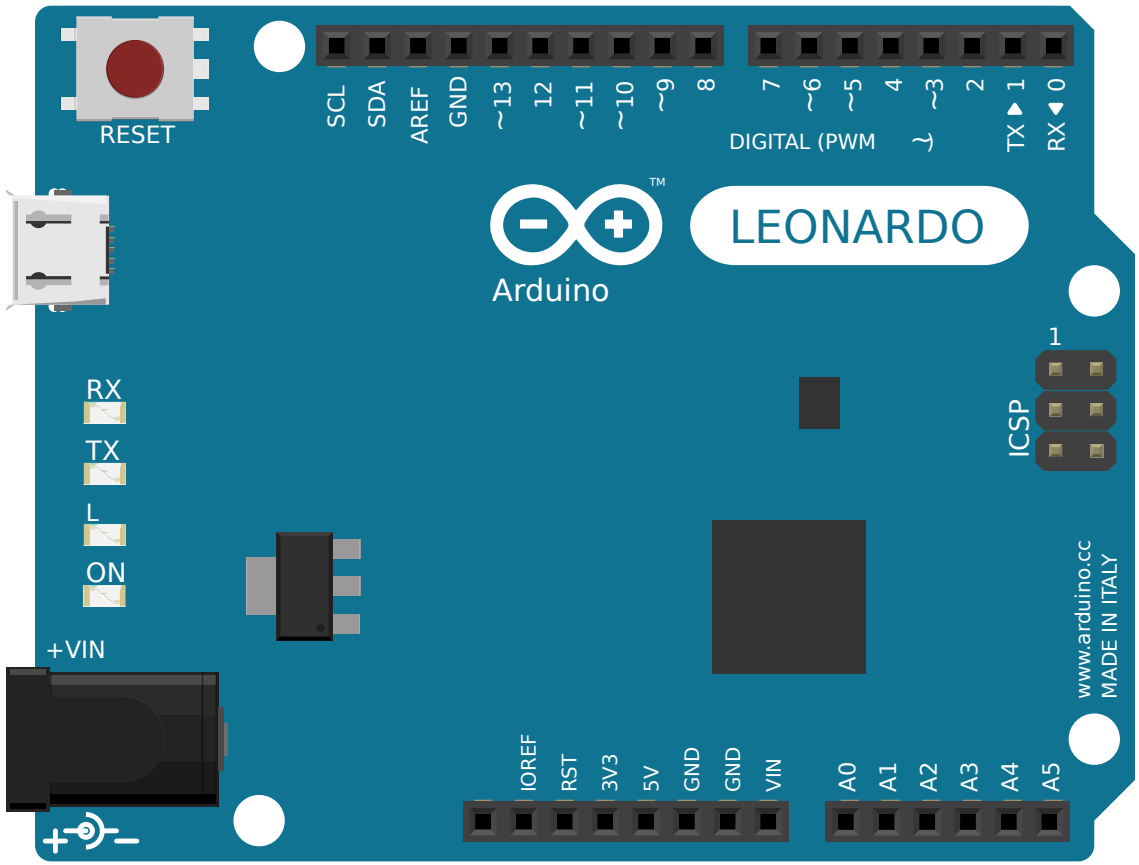
# Potenziometro

Come possiamo generare un segnale analogico che varia continuamente?
Ad esempio, con un potenziometro (o trimmer)



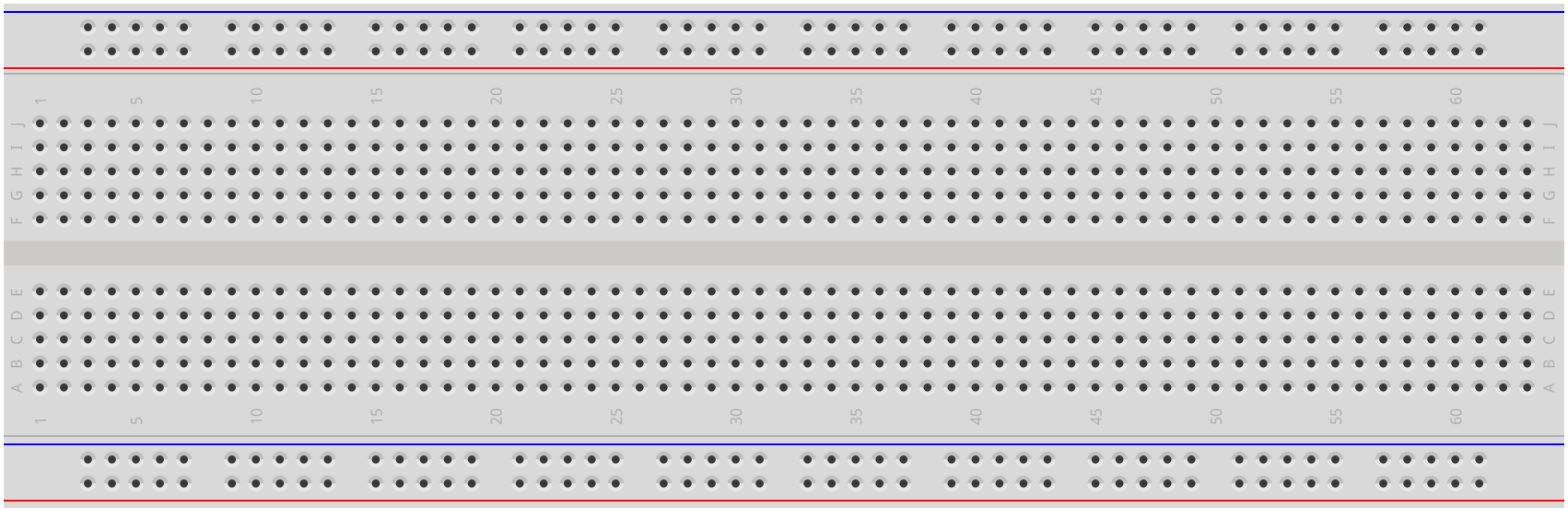

Collegato in questo modo, facendolo ruotare da un estremo all'altro OUT varierà continuamente tra GND e VCC.

# Arduino Leonardo



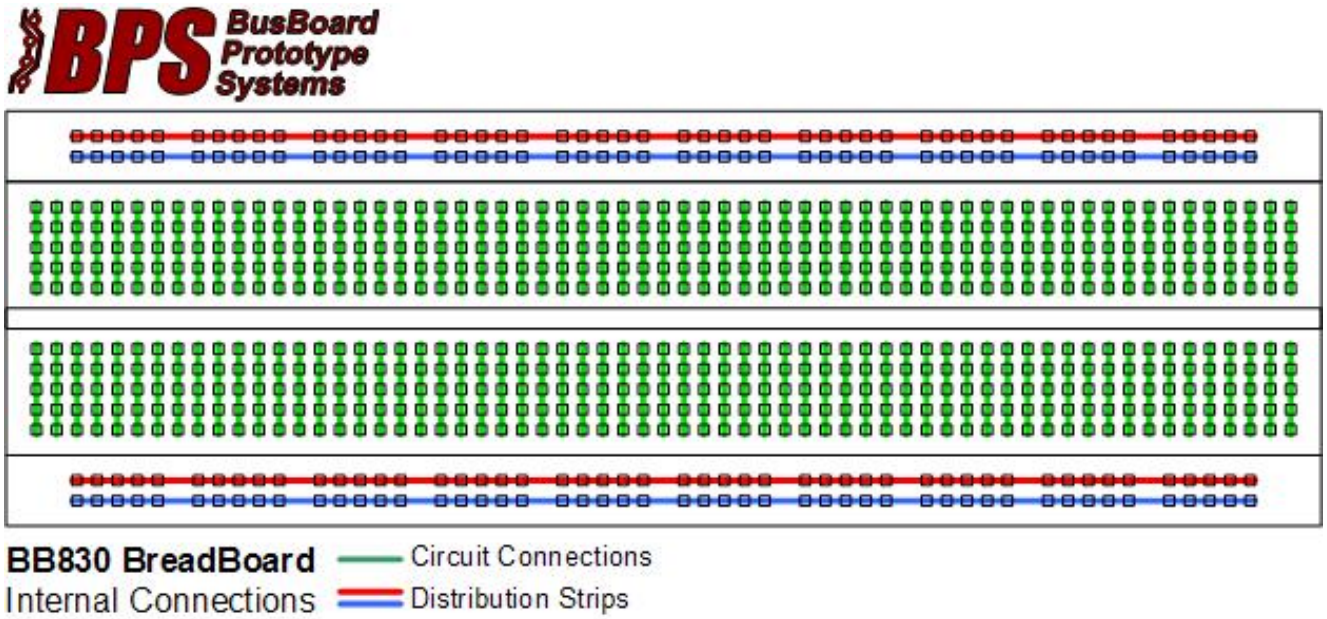

- Sarà il cervello dei nostri progetti
- Leonardo rispetto a Uno permette di essere vista come un diverso dispositivo USB

# Breadboard



- Serve per connettere i vari componenti tra di loro
- Non richiede saldatura
- Ce ne sono di varie misure e tipi

# Collegamenti della Breadboard

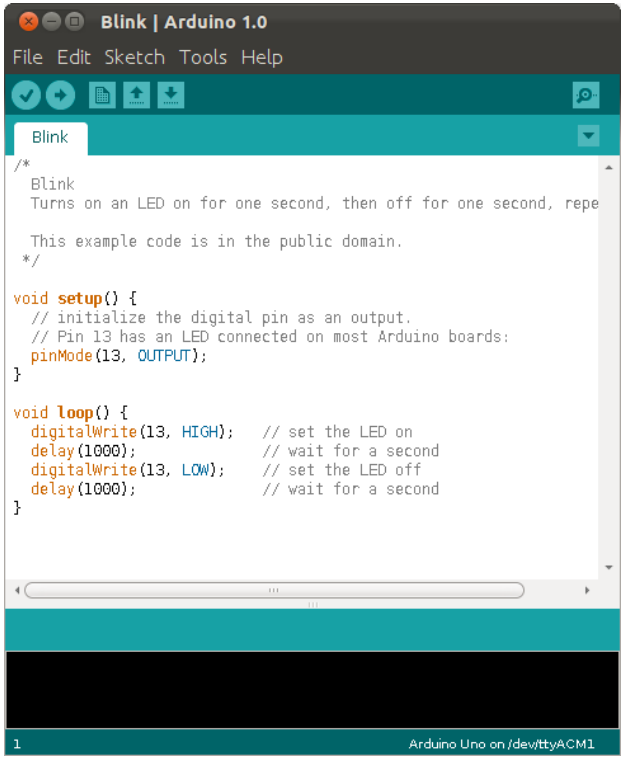



- I due blocchi centrali sono collegati tra di loro a blocchi di 5 (lato corto)
- I due blocchi centrali non sono collegati tra di loro
- Le 4 (o 2) file sul lato lungo sono tutte collegate tra di loro
- Su alcune breadboard, le file sul lato lungo sono interrotte a metà

# Fili in rame

- Servono a connettere Arduino alla Breadboard e i componenti quando la breadboard non basta
- Esistono dei set pretagliati per le breadboard se siete pigri
- Altrimenti basta del cavo rigido e un paio di pinze

# Arduino IDE




- È il programma che ci permette di scrivere sketch Arduino e di caricarli
- Contiene anche molti esempi già pronti
- Gira su Linux, Mac e Windows

# Test 1,2,3

```
int ledPin = 13;
void setup(){
  pinMode(ledPin, OUTPUT);
}
void loop(){
  digitalWrite(ledPin, HIGH);
}
```

Il risultato dovrebbe essere il LED acceso fisso

**Attenzione**

Prima di caricare il codice, selezionate la board Arduino Leonardo (Strumenti -> Tipo di Arduino)

# Analisi

- `int ledPin = 13;`
  - Assegna alla variabile ledPin il numero 13
  - Non è indispensabile ma appena i pin usati superano i 2 diventa comodo
  - Va definito fuori dalle funzioni di setup e loop (variabile globale)
- `void setup(){...}`
  - È una delle due funzioni che devono essere definite in ogni sketch
  - Tutto ciò che è al suo interno viene eseguito solo una volta appena Arduino viene accesa
  - Di solito la si usa appunto per il setup iniziale
- `void loop(){...}`
  - L'altra funzione che deve venire definita in ogni sketch
  - Come si intuisce dal nome, viene eseguita tutta e poi si rinizia da capo, finchè non si spegne Arduino

# Analisi/2

- `pinMode(ledPin, OUTPUT);`
  - Imposta la modalità del pin
  - Il primo argomento è il numero del pin che si vuole impostare
  - Il secondo è la modalità (INPUT, OUTPUT o INPUT_PULLUP)
- `digitalWrite(ledPin, HIGH);`
  - Setta un'uscita digitale
  - Il primo argomento è il numero del pin che si vuole settare
  - Il secondo è lo stato che si vuole impostare (HIGH o LOW)
  - Su Arduino Leonardo, HIGH significa che il pin sarà a 5 V, LOW che sarà a 0 V

# Blinking LED

- `delay(int nMilliseconds);`
  - Interrompe l'esecuzione per nMilliseconds

Con questa funzione dovreste ora essere in grado di scrivere l'Hello World per eccellenza di Arduino: il blinking LED. Dovete fare accendere e spegnere il LED ogni secondo.

# Digital Input

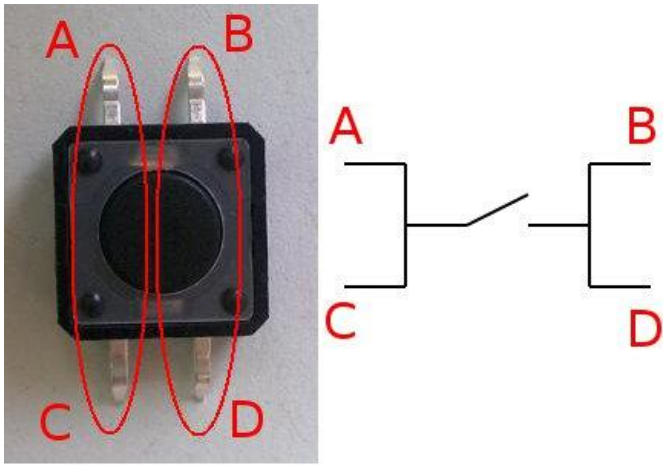
Finora abbiamo usato solo i pin come output. Come avrete intuito possono essere usati anche come input.

Come impostiamo il pin?

# Digital Input/2

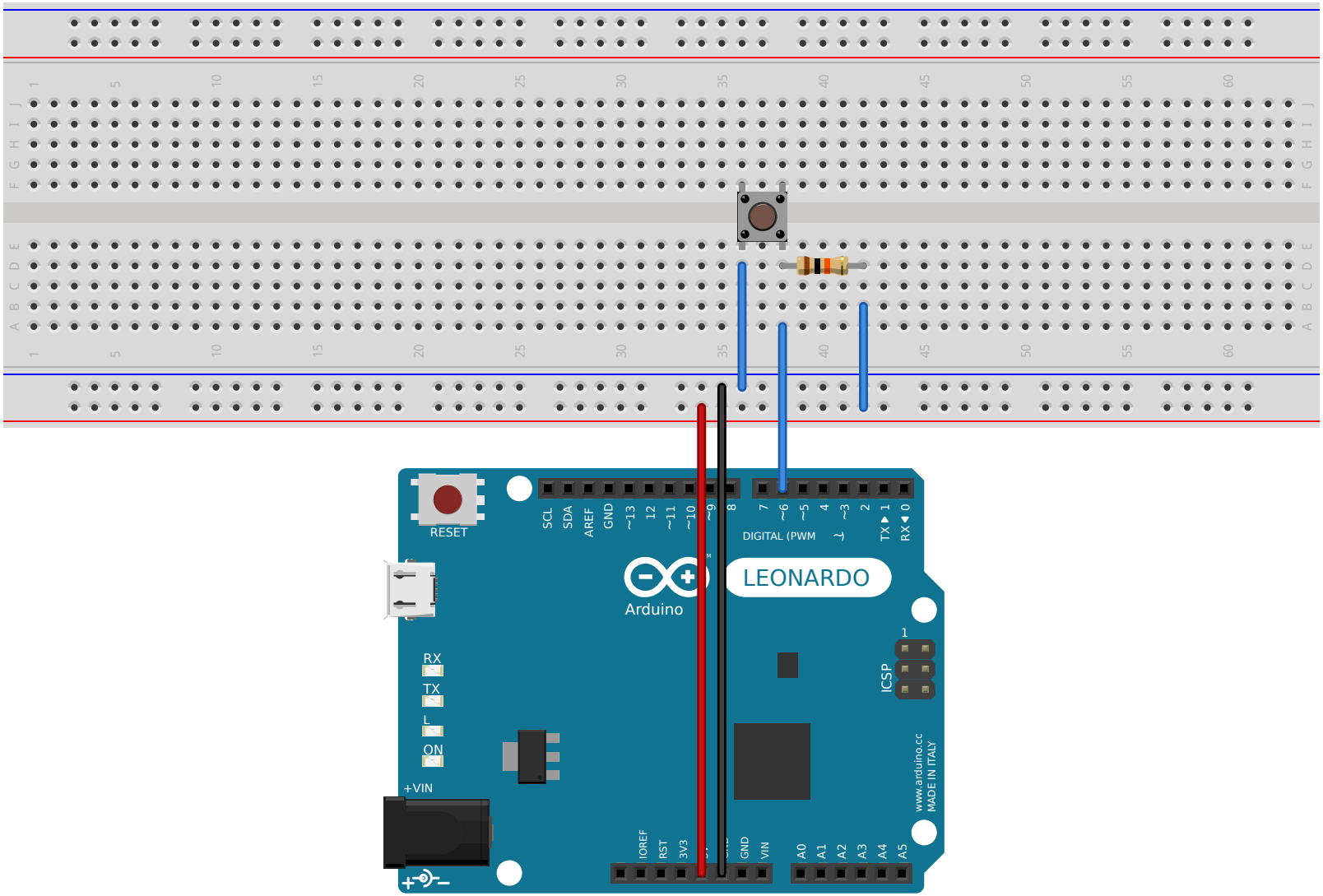
- `pinMode(ledPin, INPUT);`
- `digitalRead(pin);`
  - Legge il pin (indicato con un numero) e ritorna HIGH o LOW
- `if (condizione) {...} else {...}`
  - se `condizione` è `true` esegue il codice tra la prima coppia di parentesi graffe, altrimenti il codice tra la seconda coppia di parentesi graffe (non è obbligatorio avere l'`else`)
- `A == B`
  - Se A è uguale a B, ritorna `true`, altrimenti `false`

# Bottone




- A e C sono sempre connessi, B e D sono sempre connessi. Quando si schiaccia il bottone sono tutti connessi
- Cercate sempre informazioni sul bottone che state usando per esserne sicuri
- Ricordate quello che avevamo detto all'inizio sulla resistenza di pullup!

# Circuito bottone



# Tocca a voi

- Per prima cosa realizzate il classico blinking LED.
- Poi realizzate un blinking LED che blinka soltanto quando il bottone è premuto, mentre rimane spento se il bottone viene rilasciato.
- Se finite anche il secondo esercizio, provate ad accendere il LED con il bottone usato come "toggle": se schiacciato e rilasciato una volta accende il LED, se schiacciato e rilasciato un'altra volta lo spegne. Osservate le problematiche che sorgono. Perché sorgono?

Go!

# Soluzione blinking LED

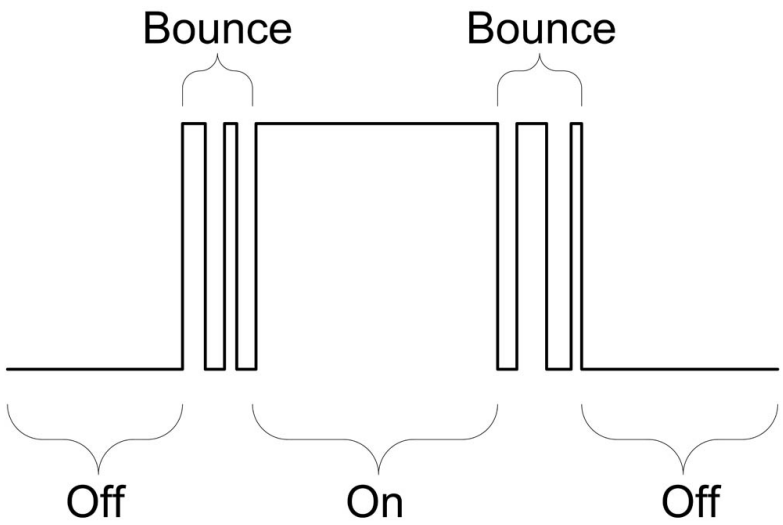
```
int ledPin = 13;

    int interval = 1000;
    void setup(){
      pinMode(ledPin, OUTPUT);
    }
    void loop(){
      digitalWrite(ledPin, HIGH);
      delay(interval);
      digitalWrite(ledPin, LOW);
      delay(interval);
    }
```

# Soluzione LED + bottone

```
int ledPin = 13;
int buttonPin = 6;
int interval = 1000;
void setup(){
  pinMode(buttonPin, INPUT);
  pinMode(ledPin, OUTPUT);
}
void loop(){
  if (digitalRead(buttonPin) == LOW){
    digitalWrite(ledPin, HIGH);
    delay(interval);
    digitalWrite(ledPin, LOW);
    delay(interval);
  } else {
    digitalWrite(ledPin, LOW);
  }
}
```

# Bonus level: bouncing




Quando il bottone viene premuto, in realtà gli switch meccanici fanno un po' di falsi contatti prima di stabilizzarsi.

# Debouncing in software, v1

Come si può risolvere questo problema? Vediamo alcuni approcci.

- Via software (bloccante): quando rileviamo lo stato che vogliamo, inseriamo un delay successivo per evitare i contatti dopo (5ms dovrebbero bastare)

```
...
currState = digitalRead(buttonPin);
if (currState != prevState){
  delay(5);
  if (currState == LOW){
    ledState = !ledState;
    digitalWrite(ledPin, ledState);
}
prevState = currState;
...
```

- Lo svantaggio è che si "perdono" 5ms ogni volta che si schiaccia il bottone

# Debouncing in software, v1: spiegazione

- Dobbiamo avere 3 variabili
  - currState, che conterrà ad ogni loop il valore di lettura del bottone
  - prevState, che conterrà il valore di lettura del loop precedente del bottone
  - ledState, che conterrà lo stato del LED
  - le variabili saranno di tipo boolean, ovvero con valore false o true: false viene visto come LOW, true viene visto come HIGH
- Se lo stato del bottone è cambiato rispetto al loop precedente (!= significa "non uguale"), vuol dire che è stato o premuto o rilasciato
- Aspettiamo 5 millisecondi per evitare il rumore del bouncing
- Se lo stato attuale è LOW vuol dire che il bottone è stato schiacciato, quindi invertiamo lo stato del LED
  - ledState = !ledState significa "Assegna a ledState il valore inverso di ledState" (! significa not). Quindi se era true (HIGH) diventerà false (LOW) e viceversa.

# Debouncing in software v2

- Via software (non bloccante): ad ogni giro del loop, se il bottone è nello stato "attivo" incrementiamo un contatore, se è nello stato "a riposo" lo mettiamo a zero. Impostiamo una soglia a quel contatore oltre cui il bottone è considerato premuto.

```
...
if (digitalRead(buttonPin) == LOW){
  counter++;
} else {
  counter = 0;
}
if (counter > 50){
  currState = LOW;
} else {
  currState = HIGH;
}
if (currState == LOW && prevState == HIGH){
  ledState = !ledState;
  digitalWrite(ledPin, ledState);
}
prevState = currState;
...
```

# Debouncing in software, v2: spiegazione

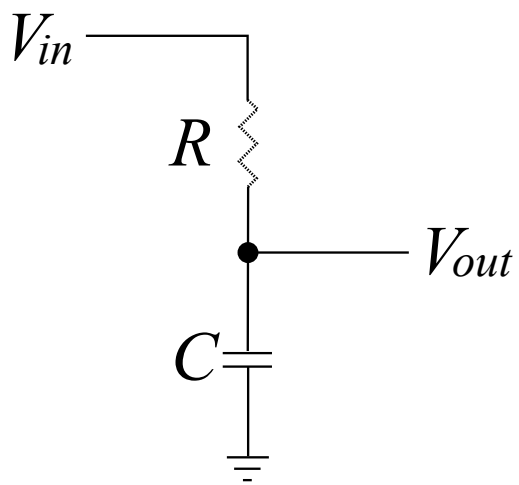
- Dobbiamo avere 3 variabili come prima, più una variabile counter di tipo unsigned int
- Se il bottone è premuto, incrementiamo il contatore, se rileviamo che è rilasciato (perché effettivamente rilasciato o per rumore di bouncing), riportiamo il contatore a 0
- Quando il contatore arriva ad una soglia (in questo caso 50), sappiamo che il bottone è stato almeno quel numero di loop premuto senza rumore, quindi dichiariamo che il suo stato è effettivamente "premuto" (LOW, per coerenza con gli esempi precedenti)
- A questo punto, entreremo nell'if (solo il primo giro che il bottone è stato dichiarato premuto) e invertiremo lo stato del LED

**Attenzione**

Questo metodo funziona male se ci sono dei delay nel codice. Ad esempio, se state facendo blinkare il LED con due delay da 1000ms, ogni "giro" del loop durerà ~2 secondi, quindi per arrivare alla soglia di 50 bisogna tenere il bottone premuto 100 secondi! La prossima lezione vedremo come evitare di usare la funzione delay e vivere felici.

# Debouncing in hardware

- Via hardware: con un filtro passa basso (con R=100kΩ e C = 47nF, $\tau$ = R×C = 4.7ms)

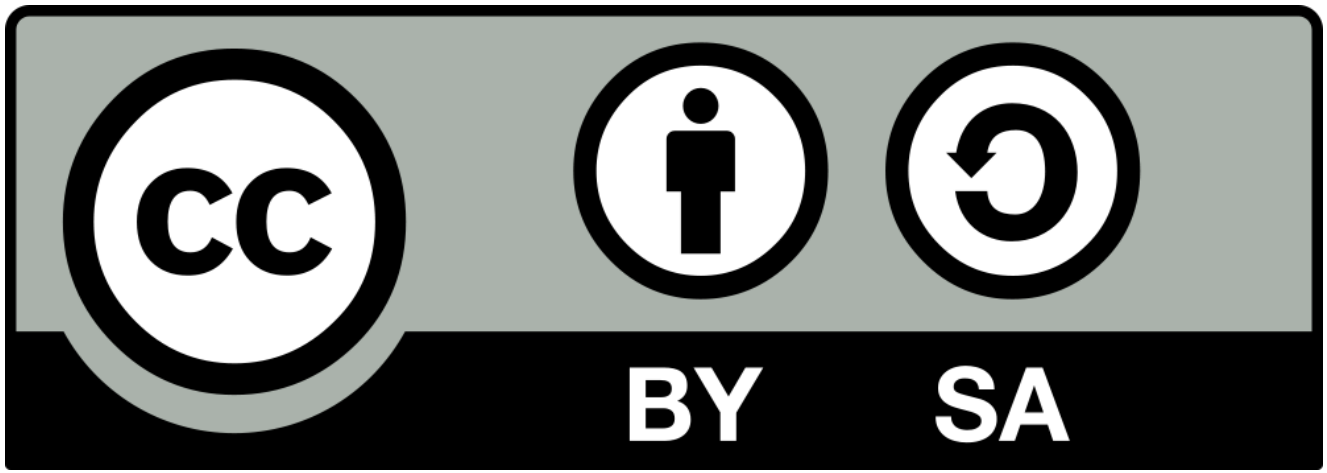


# Domande?

Se vi vengono in mente più tardi, fatele sul gruppo

# Per oggi è tutto

Ci vediamo settimana prossima






`https://www.poul.org`