Realizzare una Pila con una Patata

Obiettivo: Realizzare una Pila con una Patata ed accendere un LED.

Componenti elettronici:

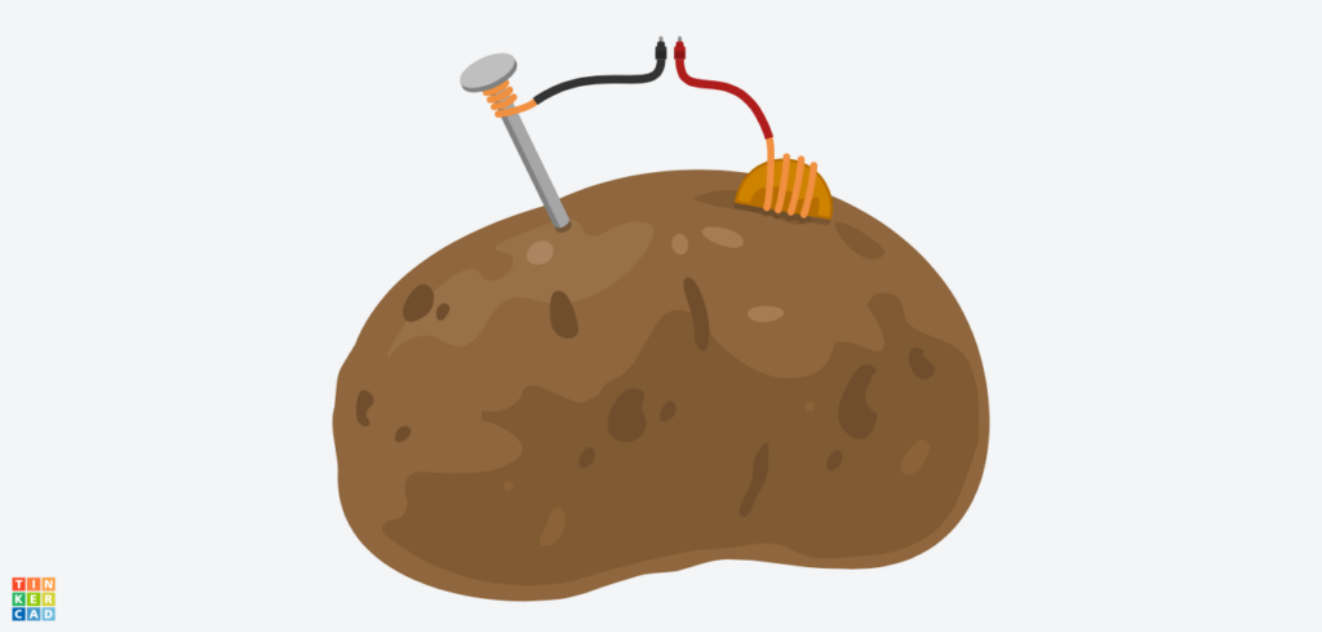
- 1 Patata
- 1 moneta da 5 centesimi (rame)
- 1 chiodo (zinco)
- Cavi
- 1 Led (opzionale – utilizzato per verificare il funzionamento della pila)
- 1 Multimetro (opzionale – utilizzato per verificare il funzionamento della pila)

Teoria: Dal punto di vista teorico, una pila è costituita da una soluzione elettrolitica nella quale sono immersi due differenti metalli (come ad esempio rame e zinco).
E' importante considerare che ogni agrume possiede al proprio interno succhi che possono fungere da soluzioni elettrolitiche in quanto ricchi di ioni.
Per questo motivo, elementi come limoni, arance e patate possono essere facilmente trasformati in pile.

Procedimento: Viene in seguito riportato un procedimento step by step per realizzare la pila utilizzando una patata

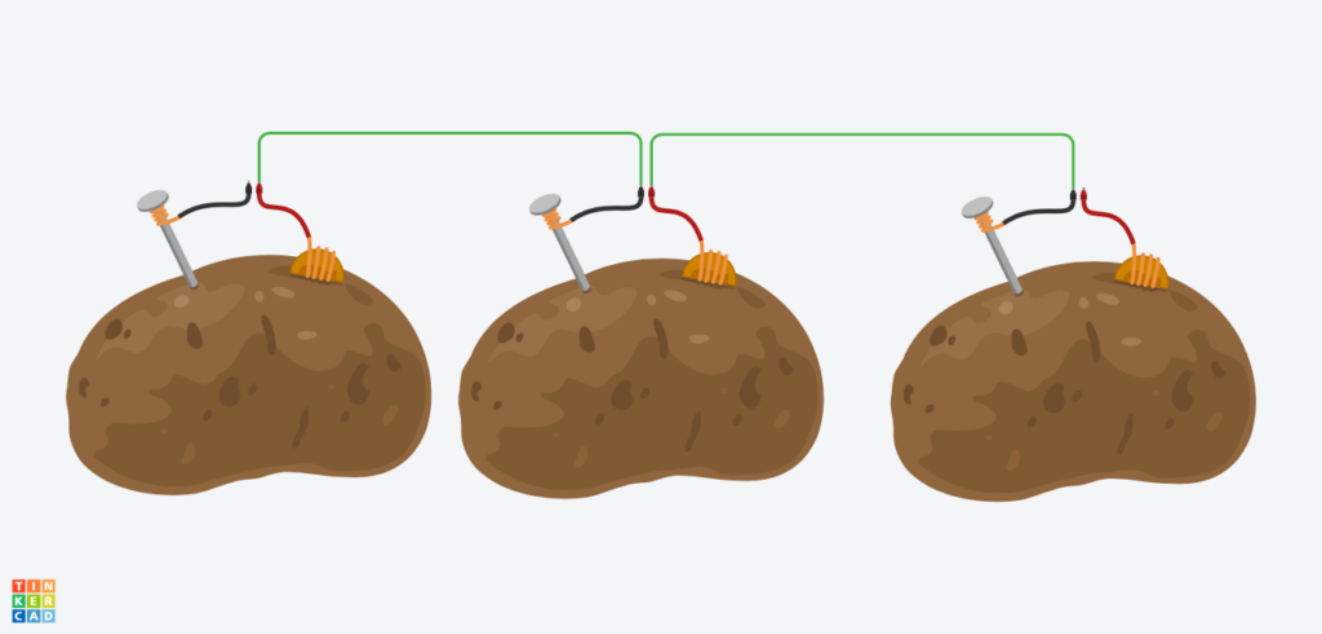
- Prima di iniziare, fare pressione con i palmi delle mani sulla patata appoggiata sul tavolo in modo schiacciarla e rompere i legami interni che producono il succo (questo permette di generare più energia).
- Introdurre alle due estremità della patata la moneta da 5 centesimi (è possibile anche utilizzare un chiodo di rame) ed il chiodo di zinco.
- Evitare che all'interno della patata i due elementi conduttori si tocchino tra di loro.
- Utilizzare il multimetro per determinare il livello di differenza di potenziale (tensione) prodotto.

Collegamento Circuitale:



Collegamento Circuitale

Personalizzazioni: E' possibile collegare più patate in serie al fine di incrementare il livello di tensione.



Mappa Concettuale di Arduino

Obiettivo: Viene presentata una mappa concettuale di Arduino utile per studenti, DSA, BES, e curiosi vari.

E’ possibile scaricare la mappa in formato PDF qui.

Riavviare Arduino in modo

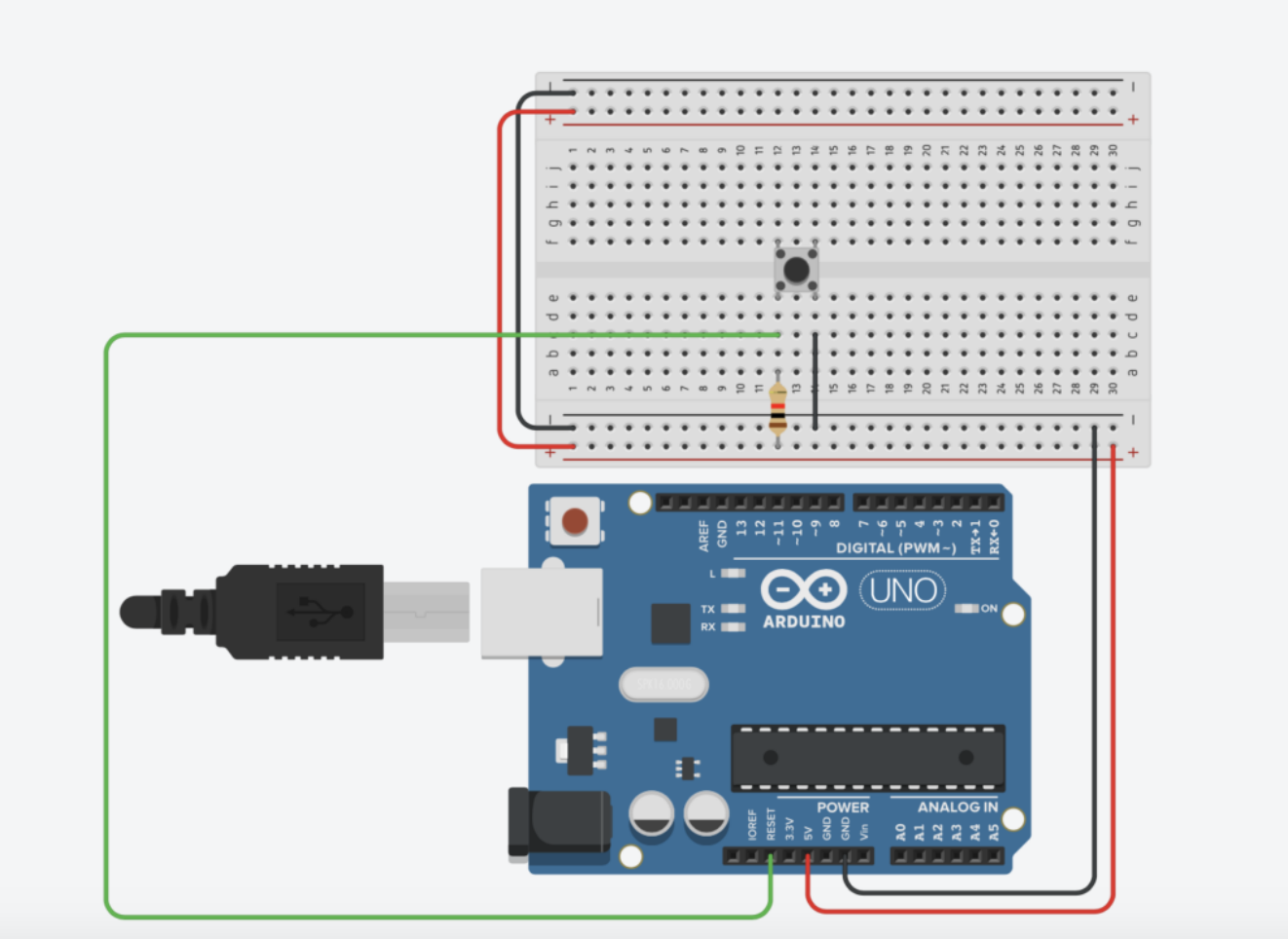
Hardware

Obiettivo: Utilizzare la porta Reset per riavviare Arduino.

Teoria: In alcune applicazioni potrebbe essere utile avere la possibilità di riavviare l’esecuzione dello sketch via software. In questo modo si può ripristinare la board ad una condizione iniziale certa.
Un esempio potrebbe essere la gestione del meccanismo di sicurezza di una macchina industriale dove, premendo il pulsante di arresto, il dispositivo deve ritornare ad uno stato iniziale ben definito.
In questo specifico caso, è possibile **utilizzare** la porta Reset di arduino, che se opportunamente attivata, permette di riavviare il programma.
Nel dettaglio è importante specificare che la porta di reset è **attiva bassa** ovvero se portato a livello basso (0V) esegue il reset della scheda.

Per effettuare il reset della scheda viene introdotto un pulsante esterno collegato in serie con una resistenza di pull-up. In questa configurazione il pulsante permette di gestire in modo corretto (attivo basso) la porta di reset.

Collegamento Circuitale:



Codice: Viene in seguito riportato il codice necessario per riavviare via hardware il microcontrollore Arduino. Nello specifico il programma, scrive sul monitor seriale la scritta “Start” ogni volta che Arduino viene riavviato.

Riavviare Arduino in modo Software

Obiettivo: Utilizzare una “funzione di reset” per riavviare Arduino via codice.

Teoria:

In alcune applicazioni potrebbe essere utile avere la possibilità di riavviare l’esecuzione dello sketch via software. In questo modo si può ripristinare la board ad una condizione iniziale certa.
Un esempio potrebbe essere la gestione del meccanismo di sicurezza di una macchina industriale dove, premendo il pulsante di arresto, il dispositivo deve ritornare ad uno stato iniziale ben definito.
In questo specifico caso, è possibile **definire** ed **utilizzare** una funzione di Reset, che se richiamata, permette di riavviare il programma.
Nel dettaglio prima della funzione di setup viene dichiarato un **puntatore a funzione** il quale punta alla posizione zero. Richiamando questa funzione Arduino esegue il codice come se fosse stato appena avviato.

Codice: Viene in seguito riportato il codice necessario per riavviare via software il microcontrollore Arduino. Nello specifico il programma, una volta avviato, attende per cinque secondi prima di effettuare il reset software. Il monitor seriale viene utilizzato per fornire i feedback relativi alle fasi di start e di reset.

Leggere informazioni dal GPS BN-880 o uBlox M8N

Obiettivo:

Leggere tutte le informazioni provenienti dal GPS BN-880 utilizzando la libreria TinyGPS++

Video:

Componenti elettronici:

- Arduino UNO
- 1 GPS BN-880 (link 1 – link 2) o uBlox NEO-7M/M8N (link 3 – link 4)

Teoria:

Alla base di questa esercitazione c’è il modulo GPS Baitian

BN-880, ma andrebbe benissimo anche un modulo uBlox NEO-7M o M8N o equivalenti.

- 

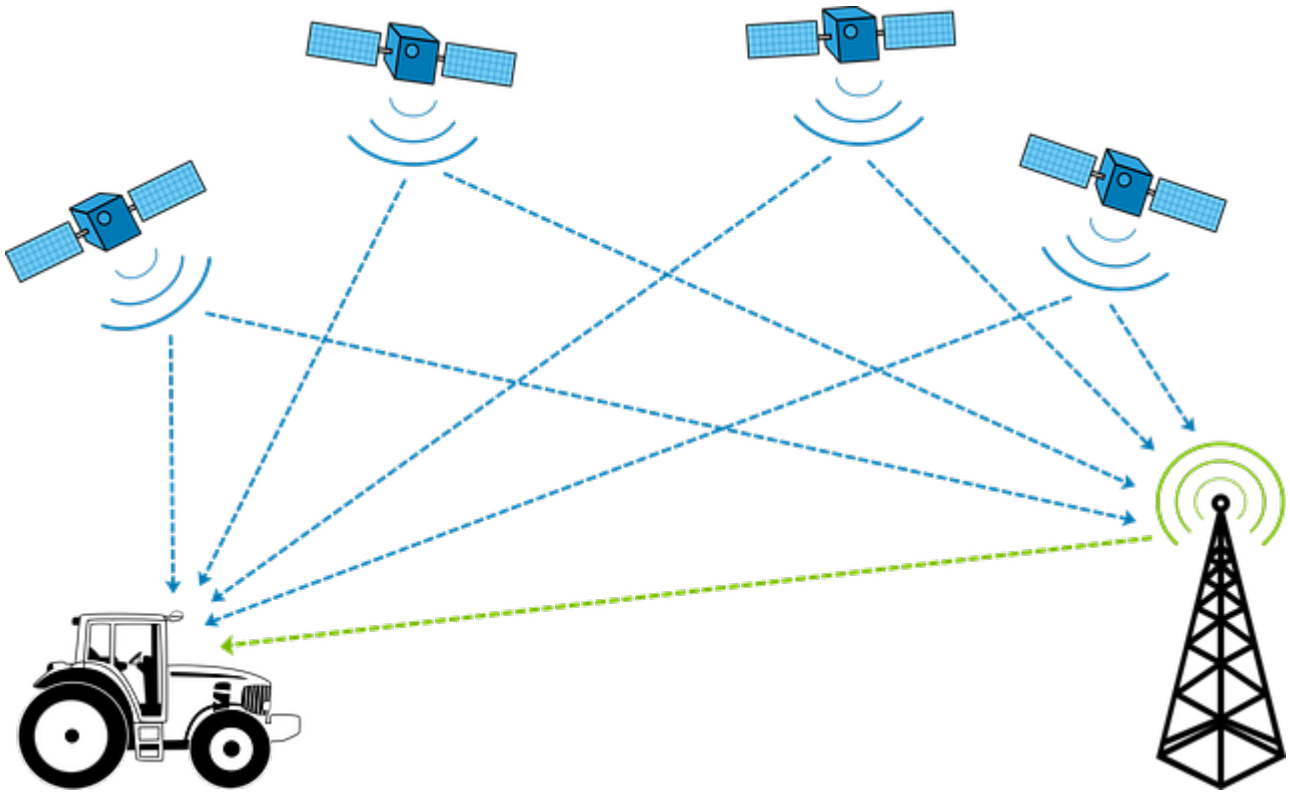
- 

Il modulo in oggetto oltre ad avere integrato il GPS, oggetto dell'esercitazione, ha anche integrato una bussola (compass) HMC5883l comunicante
tramite il bus/protocollo I2C.

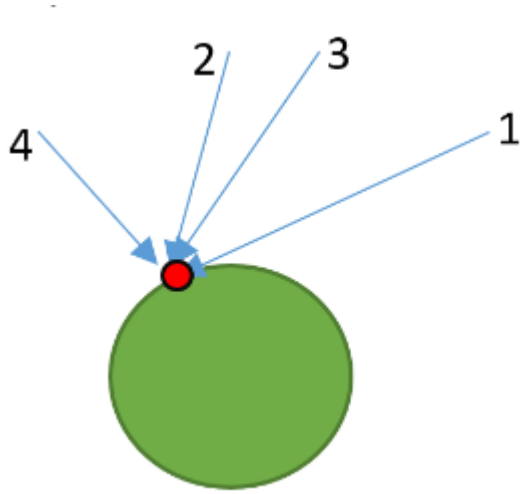
Ecco alcune caratteristiche:

- Inexpensive,
  light, Dual Mode GPS and GPS/Compass using UBLOX M8N module
- Receiving
  Format: GPS, GLONASS, Galileo, BeiDou, QZSS and SBAS
- Frequency:
  GPS L1,GLONASS L1,BeiDou B1,SBAS L1,Galileo E1
- Channels: 72
- Data Protocol: NMEA-0183 or UBX, Default
  NMEA-0183
- Single GNSS: 1Hz-18Hz
- Concurrent GNSS: 1Hz-10Hz
- Compass IC: HMC5883L

Il sistema GPS (Global Positioning System):



GPS sta per **Global Positioning System** cioè un Sistema di Posizionamento Globale, la cui funzione primaria è quella di determinare con precisione la posizione sul pianeta Terra di un ricevitore (come ad esempio uno smartphone o un apparecchio dedicato). Il ricevitore GPS riceve continuamente segnali da lontani satelliti in orbita attorno alla Terra. Attorno alla Terra, a circa 20.200 km di altitudine, orbitano 31 satelliti del sistema GPS. Ogni satellite ha al suo interno un precisissimo orologio atomico. Ogni satellite invia continuamente un segnale radio contenente la sua posizione, e l'orario di trasmissione del segnale. Il ricevitore GPS confronta questi segnali, e tramite alcune equazioni riesce a stabilire la sua posizione.



Il pallino rosso è il ricevitore GPS. Alle ore 12:00 tutti i satelliti GPS inviano un segnale che dice "io mi trovo nella posizione XYZ, e
sono le 12:00". Poiché i satelliti hanno distanze diverse dal ricevitore, il
segnale "io mi trovo nella posizione XYZ, e sono le 12:00" arriva al ricevitore
in momenti diversi: quello del satellite 4 arriva per primo; poi quello del

satellite 2; poi 3; e poi 1. Sapendo quanto ci ha messo il segnale ad arrivare,
e quindi la distanza fra il ricevitore e i vari satelliti, il ricevitore GPS,
con una relativamente semplice triangolazione, riesce a stimare la propria
posizione sul pianeta Terra.

Che cosa trasmette il modulo GPS:

Il modulo GPS trasmette tramite Seriale TTL (0-5 Volt) delle "sentenze" tramite il protocollo NMEA0183.

NMEA 0183 (o più comunemente NMEA) è uno standard di comunicazione di dati utilizzato soprattutto in nautica e nella comunicazione di dati satellitari GPS. L'ente che gestisce e sviluppa il protocollo è la National Marine Electronics Association. Questo protocollo si basa sul principio che la fonte, detta talker, può soltanto inviare i dati (sentences) e la ricevente, detta listener, può soltanto riceverli.

Ad esempio la seguente sentenza $GGA trasmette tantissime informazioni oltre alla Latitudine e Longitudine:

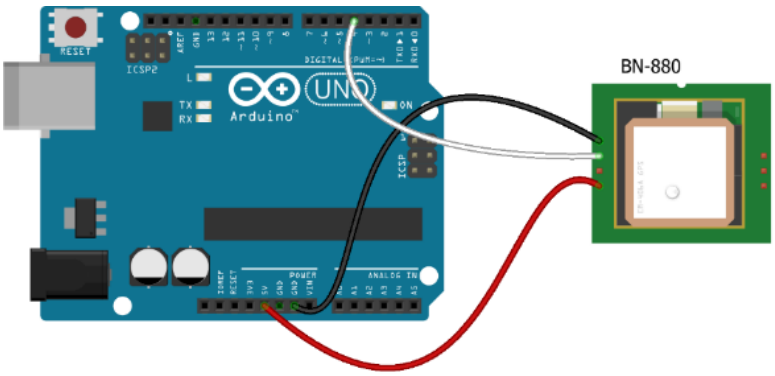
GGA **Global Positioning System Fix Data. Time, Position and fix related data for a GPS receiver**

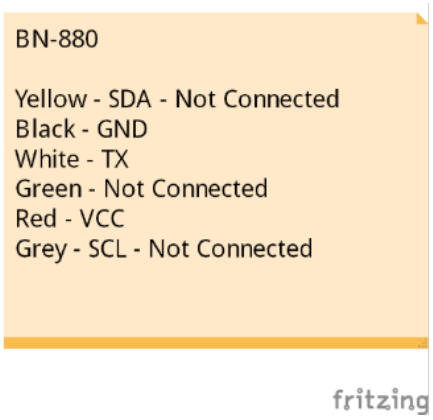
```
                                                      11
       1         2       3 4        5 6 7  8   9  10 |  12 13  14   15
       |         |       | |        | | |  |   |   | |   | |   |    |
$--GGA,hhmmss.ss,llll.ll,a,yyyyy.yy,a,x,xx,x.x,x.x,M,x.x,M,x.x,xxxx*hh
```

1) Time (UTC)
2) Latitude
3) N or S (North or South)
4) Longitude
5) E or W (East or West)
6) GPS Quality Indicator,
   0 - fix not available,
   1 - GPS fix,
   2 - Differential GPS fix
7) Number of satellites in view, 00 - 12
8) Horizontal Dilution of precision
9) Antenna Altitude above/below mean-sea-level (geoid)
10) Units of antenna altitude, meters
11) Geoidal separation, the difference between the WGS-84 earth ellipsoid and mean-sea-level (geoid), "-" means mean-sea-level below ellipsoid
12) Units of geoidal separation, meters
13) Age of differential GPS data, time in seconds since last SC104 type 1 or 9 update, null field when DGPS is not used
14) Differential reference station ID, 0000-1023
15) Checksum

Tramite
la libreria TinyGPS++ è possibile intercettare tutti i tipi di sentenze GPS e
inoltre nell'ultima versione anche di elaborare le sentenze particolari, ad
esempio prelevate da altre fonti NMEA0183.

Schema Elettronico (Fritzing):

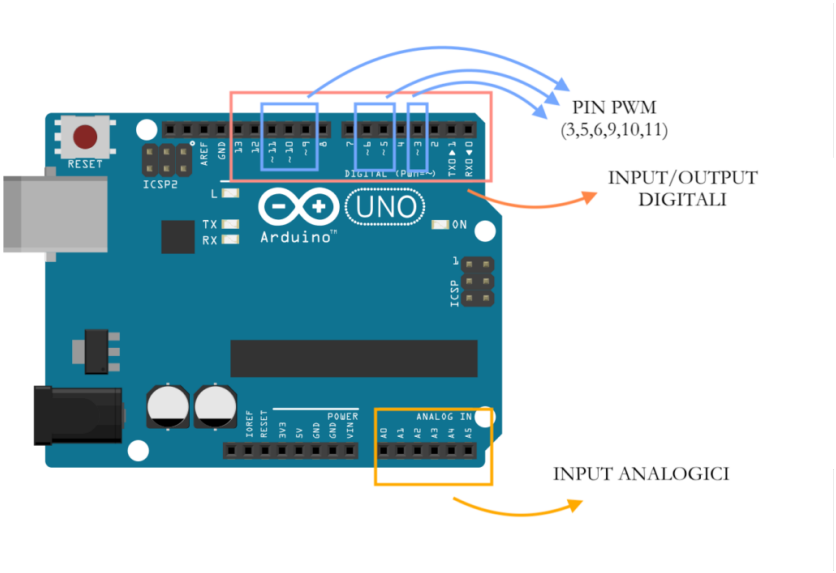




Codice:

Le Funzioni digitalWrite, digitalRead, analogWrite e analogRead

Obiettivo: Imparare ad utilizzare le principali funzioni di Arduino



Arduino Porte (Input, Output, Digitali, Analogici)

Teoria:

Le principali funzioni utilizzate da Arduino per comunicare con il mondo esterno sono quattro e si dividono in base alla **tipologia di azione:**

- Lettura: utilizzate per acquisire i dati dai differenti sensori (luminosità, temperatura, umidità, etc)
- Scrittura: utilizzate per comandare i differenti attuatori (motori, buzzer, display, etc)

ed in base alla **tipologia di segnale trattato:**

- Digitale: utilizzate per trattare segnali digitali che possono assumere solamente valori logici (i.e., LOW e HIGH)
- Analogico: utilizzate per trattare segnali analogici con valori compresi tra 0 e 5V.

Nello specifico queste quattro funzioni sono così definite:

Codice:

- **digitalWrite:** Funzione utilizzata per comandare attuatori mediante una logica LOW/HIGH come ad esempio motori, led o buzzer. Questa funzione prevede l'impiego di due parametri di input: il **PIN** (0-13) ed il **VALORE** (LOW/HIGH)

digitalWrite(pin, valore);

- **analogWrite:** Funzione utilizzata per comandare attuatori mediante una logica analogica (valori compresi tra 0V e 5V) come ad esempio motori o led. Questa funzione prevede l'impiego di due parametri di input: il **PIN** (0-13) ed il **VALORE** (0-255). Nel caso specifico il valore 0 corrisponde a 0V mentre 255 a 5V. Per tutti gli altri VALORI si può attuare la proporzione lineare. (Ad esempio volendo generare un riferimento di tensione pari a 3Volt il VALORE di input può essere così calcolato: (3/5)*255. E' importante considerare che i valori di tensione non sono "realmente" analogici ma generati attraverso la tecnica **PWM**. Inoltre, l'istruzione analogWrite può essere utilizzata solamente su alcuni pin digitali di output: i pin PWM (3,5,6,9,10,11).

analogWrite(pin, valore);

- **digitalRead:** Funzione utilizzata per leggere dati da sensori basati su una logica LOW/HIGH come ad esempio i pulsanti. Questa funzione prevede l'impiego di un parametro di input: il **PIN** (0-13) ed un parametro di output: il **VALORE** (LOW/HIGH) che viene restituito dalla funzione.

valore= **digitalRead**(pin);

- **analogRead:** Funzione utilizzata per leggere dati da sensori di tipo analogico (valori compresi tra 0V e 5V) come ad esempio fotoresistenze, sensori di temperatura, umidità etc. Questa funzione prevede l'impiego di un parametro di input: il **PIN** (A0-A5) ed un parametro di output: il **VALORE** (0-1023). Nel caso specifico il valore 0 corrisponde a 0V mentre 1023 a 5V. Per tutti gli altri VALORI si può attuare la proporzione lineare. (Ad esempio se viene letto il VALORE 512, la tensione di riferimento può essere così calcolata: (512/1023)*5.

valore = **analogRead**(pin);

Quiz:

A che valore di tensione corrisponde l'intero 818 letto attraverso la funzione analogRead

4V
3V
2V
5V
Correct! Wrong!
(818/1023)*4 = 1V

A che valore di tensione corrisponde l'intero 204 letto attraverso la funzione analogRead

2V
4V
3V
1V
Correct! Wrong!
(204/1023)*5 = 1V

A che valore di tensione corrisponde l'intero 511 letto attraverso la funzione analogRead

5V
4V
2,5V
1V
Correct! Wrong!
(511/1023)*5 = 2.5V

A che valore di tensione corrisponde l'intero 255 generato utilizzando la funzione analogWrite

5V
0V
2.5V
1V
Correct! Wrong!
(255/255)*5 = 5V

A che valore di tensione corrisponde l'intero 100 generato utilizzando la funzione analogWrite

1V
2V
1.66V
1.96
Correct! Wrong!
(100/255)*5 = 1.96V

La funzione Valore = digitalWrite(11) è corretta?

Vero
Falso
Correct! Wrong!
La funzione digitalWrite non ha tipi di ritorno

La funzione digitalWrite(11,HIGH) è corretta?

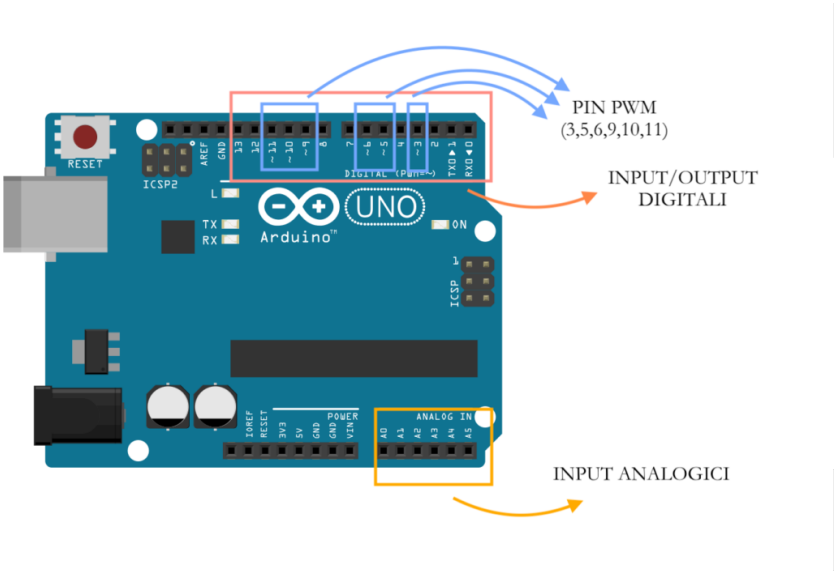
Vero
Falso
Correct! Wrong!

La funzione analogWrite(11,1023) è corretta?

Vero
Falso
Correct! Wrong!
Il valore massimo di tensione per la funzione digitalWrite è pari a 255

Arduino e i 6 PIN Digitali Segreti

Obiettivo: Scoprire ed utilizzare i 6 pin digitali segreti di Arduino nel proprio progetto.



Arduino Porte (Input, Output, Digitali, Analogici)

Teoria:

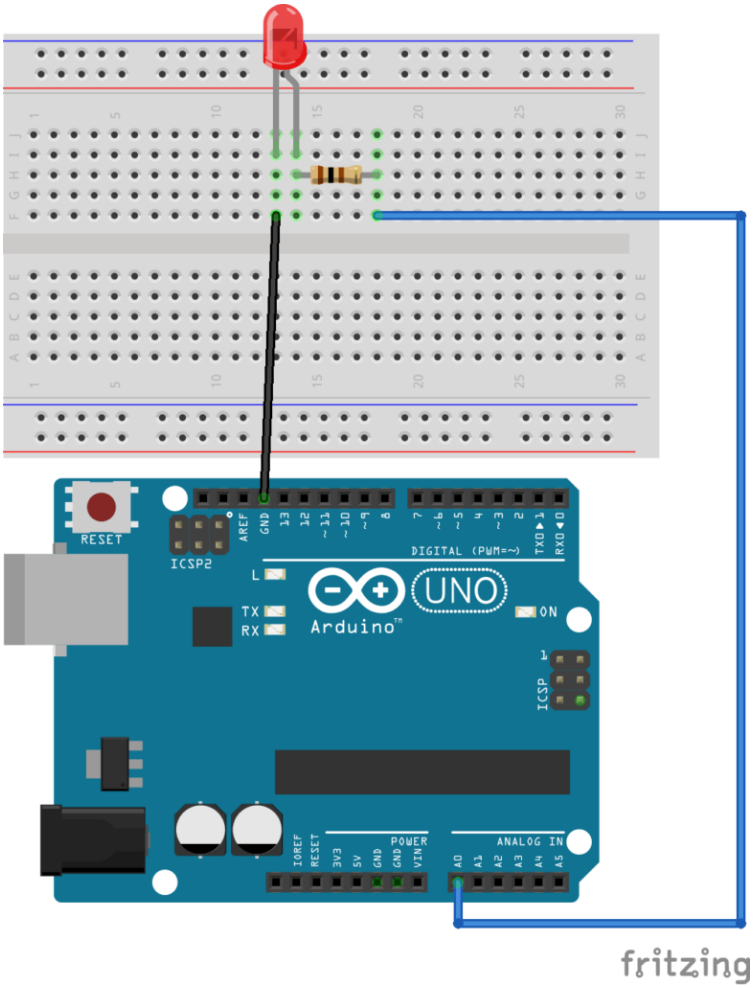
Uno dei problemi più comunemente incontrati dagli sviluppatori è quello di finire i pin digitali necessari per collegare Arduino ad eventuali periferiche (sensori/attuatori). I pin digitali sono 14 e potrebbero bastare pochi elementi per terminarli (e.g., display, led, pulsanti, keypad). E' importante inoltre considerare che i pin 0 ed 1 sono utilizzati per la comunicazione seriale pertanto ne è sconsigliato l'utilizzo.
Pertanto, le soluzioni alla carenza di pin digitali in un progetto che richieda più 14 pin sono:

- Utilizzare una scheda più efficace come Arduino Mega
- Utilizzare dei componenti elettronici aggiuntivi tipo buffer o registri
- **Utilizzare i 6 Pin Digitali Segreti (SOLUZIONE PROPOSTA)**

Nello specifico è possibile utilizzare i 6 pin di input analogici (quelli che si trovano in basso a destra nella scheda) come pin di input/output digitali. Pertanto i PIN di input/output sono in tutto 20 e non 14 come sempre pensato.
Utilizzare i pin di input analogici come pin di input/output digitale è particolarmente semplice basta semplicemente indicare nell'istruzione pinmode il numero corretto definito nella seguente tabella:

Input Analogico	**Input/Output Digitale**
A0	14
A1	15
A2	16
A3	17
A4	18
A5	19

Collegamento Circuitale:



Collegamento Circuitale

Codice: Viene in seguito riportato il codice necessario per accendere spegnere un led utilizzando il pin segreto 14.

Caricare il primo programma (Blinking Led)

Obiettivo: Utilizzare il Software Arduino per caricare il primo programma e fare lampeggiare un LED

Prerequisiti

Blinking led [Avanzato]

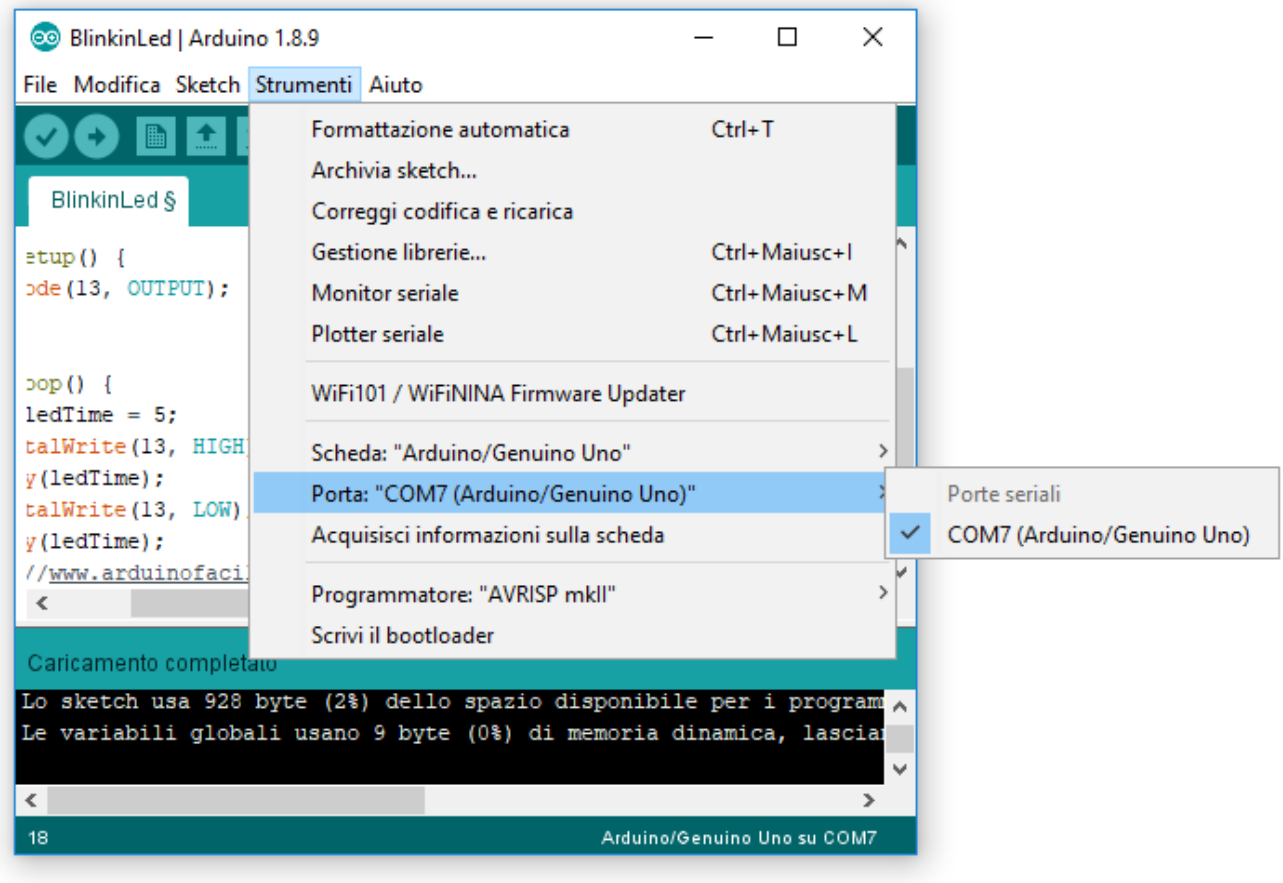
Come installare l'ambiente Software di Arduino

Teoria:

Attraverso il software di programmazione precedentemente scaricato è possibile caricare sulla piattaforma Arduino dei programmi ad hoc che abbiamo il compito di svolgere determinate operazioni utili per trasformare i propri input in output specifici.
A seguire vengono riportate le principali istruzioni utili per caricare un programma prestabilito (nel caso specifico

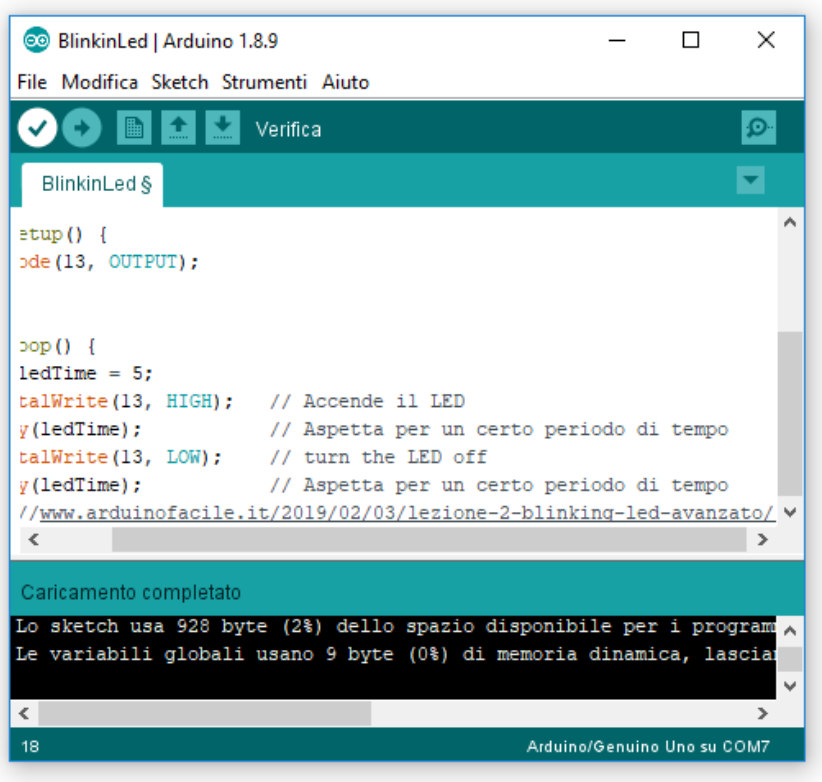
blinking led avanzato)

- Aprire il software Arduino
- Collegare la scheda Arduino al PC mediante cavo USB
- Impostare la porta di comunicazione corretta (**IMPORTANTE: se la porta non viene impostata non è possibile caricare il programma**)



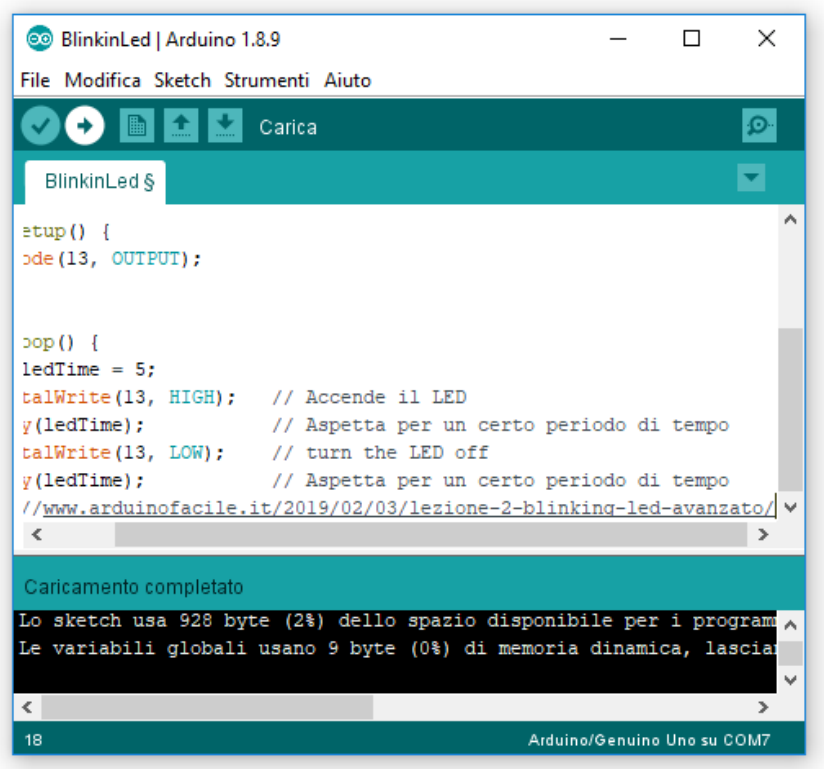
Configurare Porta Seriale Arduino

- r ePCompilare il codice: questa operazione serve per vedere se il codice scritto presenta degli errori. La compilazione si esegue premendo la spunta evidenziata in figura (in alto a sinistra).



Compilare il codice

- Caricare il codice sulla piattaforma Arduino (questa operazione deve essere eseguita solo se il codice e' stato opportunamente compilato).
  Il caricamento si esegue premendo la freccia a destra evidenziata in figura (in alto a sinistra).



Variabili Locali e Globali

Obiettivo: Comprendere cosa è una variabile e quali sono le differenze tra variabili locali e globali.

Teoria: Le variabili sono gli strumenti utilizzati in tutti i linguaggi di programmazione per la memorizzazione dei dati. Da un punto di vista pratico, le variabili possono essere considerate come contenitori di dati situati in una determinata area della memoria. Le variabili sono caratterizzate da tre differenti elementi: il tipo di dato, il nome, ed il valore contenuto.

- **Tipo di dato**: A seconda del dato che una variabile deve memorizzare il programmatore deve scegliere il corretto tipo di riferimento. Nella seguente tabella vengono riportati i principali tipi di dati utilizzati in Arduino

Tipo di Dato	Dimensione	Descrizione
char	1 Byte	Contenitore per caratteri e valori alfanumerici (e.g., ‘a’, ‘b’, ‘1’, etc)
bool	1 Byte	Contenitore per valori booleani (e.g., true, false)
int	2 Byte	Contenitore per numeri interi nel range numerico, da -32768 a 32767.
unsigned int	2 Byte	Contenitore per numeri interi senza segno nel range numerico, da 0 a 65535.
long	4 Byte	Contenitore per numeri interi nel range numerico, da -2147483648 a 2147483647
unsigned long	4 Byte	Contenitore per numeri interi nel range numerico, da 0 to 4,294,967,295 (2^32 – 1)
float	4 Byte	Contenitore per numeri con la virgola nel range numerico da 3.4028235E+38 a -3.4028235E+38
double	4 Byte	Contenitore per numeri con la virgola nel range numerico da 3.4028235E+38 a -3.4028235E+38
String		Contenitore per testo (e.g., “Resistenza”)

- **<u>Identificativo (Nome della variabile):</u>** La variabile è caratterizzata da un nome attraverso il quale si fa riferimento al contenitore stesso dei dati. E' importante considerare che il nome della variabile non deve iniziare con un numero e soprattutto non può essere uguale ad una keyword del linguaggio di programmazione. Ad esempio il nome di una variabile non può essere loop.

- **Valore:** Le variabili sono utilizzate per contenere dei dati pertanto ad ogni variabile è associato un valore (nel caso in cui l'utente non assegni nessun valore molto probabilmente è il compilatore che assegna un valore di default).

Prima di potere essere utilizzate all'interno del codice le variabili devono essere dichiarate e se possibile inizializzate con un valore specifico. La sintassi utilizzata per la dichiarazione di una variabile è la seguente.

```
int lato = 10;
float peso = 43.5;
bool condizione = true;
char lettera = 'A';
String nome = "Andrea"
```

Le variabili possono essere definite sia all'interno dei blocchi loop e setup sia al loro esterno. In base a dove vengono definite, queste prendono il nome di **variabili globali** o **locali**.

- **Variabili Globali:** Sono variabili che hanno visibilità in tutto il programma. Le variabili globali sono

definite fuori dal corpo (fuori dalle parentesi graffe) delle funzioni loop e setup.

- **Variabili Locali:** Sono variabili che hanno visibilità solo all'interno di una funzione. Le variabili locali sono definite dentro il corpo (dentro le parentesi graffe) delle funzioni loop e setup.

Codice: Viene in seguito riportato l'esempio di un codice per la gestione di un un pulsate dove sono utilizzate sia variabili locali che variabili globali.

Le Funzioni Setup() e Loop()

Obiettivo: Conoscere le caratteristiche principali delle funzioni Setup() e Loop()

Teoria: Le funzioni **Setup()** e **Loop()** rappresentano il core principale di ogni programma sviluppato utilizzando il controllore Arduino. Ogni programma parte infatti dal codice definito all'interno di queste funzioni. Nello specifico le due funzioni sono così caratterizzate:

- **Setup():** In questa funzione è definita la configurazione

iniziale di Arduino. Attraverso l'istruzione **pinMode** è possibile stabilire quali pin di Arduino sono utilizzati per gestire ingressi e quali per gestire uscite. Vengono inoltre inizializzati i vari oggetti impiegati nel loop (e.g., **Serial.begin(9600)**). E' importante considerare che le istruzioni presenti nel setup vengono eseguite solamente una volta all'avvio del controllore Arduino.

- **Loop():** In questa funzione è definito il processing ovvero le operazioni che Arduino in modo ciclico. E' importante considerare che, a differenza dalla funzione di setup, le istruzioni presenti nel loop vengono eseguite ciclicamente: **terminata l'ultima istruzione si ricomincia con la prima** (per sempre). Le principali istruzioni presenti all'interno del corpo della funzione sono quelle utilizzate per gestire gli ingressi e le uscite (e.g., digitalRead, digitalWrite, analogRead, analogRead).

E' importante considerare che il corpo di una funzione è definito mediante le parentesi graffe. Nello specifico, la parentesi graffa aperta { rappresenta l'inizio della funzione, mentre la parentesi graffa chiusa } indica la sua fine.

Codice: Vengono in seguito riportati due esempi di codice. Il primo riguarda un progetto vuoto, mentre il secondo è relativo al codice di implementazione utilizzato per fare lampeggiare un led.