Co-funded by the
Erasmus+ Programme
of the European Union

Erasmus+ KA210-VET

Partenariati su piccola scala nella

istruzione e formazione professionale

Titolo del progetto:“Using Arduinos in Vocational Training”

Acronimo del progetto: “UsingARDinVET”

Nr del progetto: “2023-1-RO01-KA210-VET-000156616”

***** *UsingARDinVET LIBRO GUIDA*****

Co-funded by the Erasmus+ Programme of the European Union

“Questo progetto è finanziato dal Programma Erasmus+ dell'Unione Europea. Tuttavia, la Commissione europea non può essere ritenuta responsabile per l'uso che può essere fatto delle informazioni in esso contenute”.

COORDINATORE:

LICEUL TEHNOLOGIC ELENA CARAGIANI (Tecuci, Romania)

PARTNER:

Ahi Evran Mesleki Eğitim Merkezi (Ankara, Türkiye)

2 EK Peiraia (Piraeus / Greece)

Instituto De Educaiión Secundaria "María Molıner" (SEGOVIA, Spain)

IPIAS G. GIORGI (Potenza, Italy)

Co-funded by the
Erasmus+ Programme
of the European Union

Sintesi del progetto

“Using Arduinos in Vocational Training”

Obiettivi:

Il modo migliore per imparare qualcosa è farlo e sperimentarlo. I materiali didattici più importanti sono i set sperimentali nell'istruzione e nella formazione professionale.

Quando si analizzano i programmi delle scuole di formazione professionale, si nota che è difficile fornire una formazione su Arduino con questi set. Abbiamo preparato questo progetto per superare questi problemi, per fornire un ambiente più efficiente e set sperimentali per i nostri studenti nelle lezioni di Arduino e per garantire un apprendimento più duraturo.

Implementazione:

*5 incontri transnazionali; 5 TPM si terranno durante i due anni del progetto. I partecipanti a questi incontri sono i team di progetto dei partner.
*Workshop “Il nostro progetto si incontra con scuole di formazione professionale, elettronica, industrie ICT, mercati del lavoro”: I risultati, i prodotti, saranno presentati ai partecipanti al workshop.

-Creazione di un team di progetto e di strumenti.
-Sito web del progetto.
-Kit e set di formazione.
-Guida all'uso diARDinVET.
-Video di formazione.
-DVD del progetto.
-5 Newsletter.
-Piantare alberi Erasmus.

Risultati:

-Cambiare la percezione degli studenti su “Insegnamento, apprendimento e utilizzo di Arduino nella formazione professionale”.
-Diminuire il livello di assenteismo nelle lezioni con Arduino.
-Far apprendere agli insegnanti metodologie innovative su Arduino.
-Fornire servizi educativi migliori ai loro studenti.
-Creare un clima scolastico positivo e migliorare l'ambiente di apprendimento.
-Migliorare i laboratori e le lezioni di Arduino nelle scuole.
-Aumentare l'occupazione dei laureati.
-Sviluppo di dialoghi interculturali.

Indice:

Co-funded by the
Erasmus+ Programme
of the European Union

Erasmus+ KA210-VET

Partenariati su piccola scala nella istruzione e formazione professionale

Titolo del progetto: “Using Arduinos in Vocational Training”

Acronimo del progetto: “UsingARDinVET”

Nr del progetto: “2023-1-RO01-KA210-VET-000156616”

Co-funded by the
Erasmus+ Programme
of the European Union

INTRODUZIONE ad ARDUINO

(Fondamenti di ARDUINO)

INTRODUZIONE AD ARDUINO

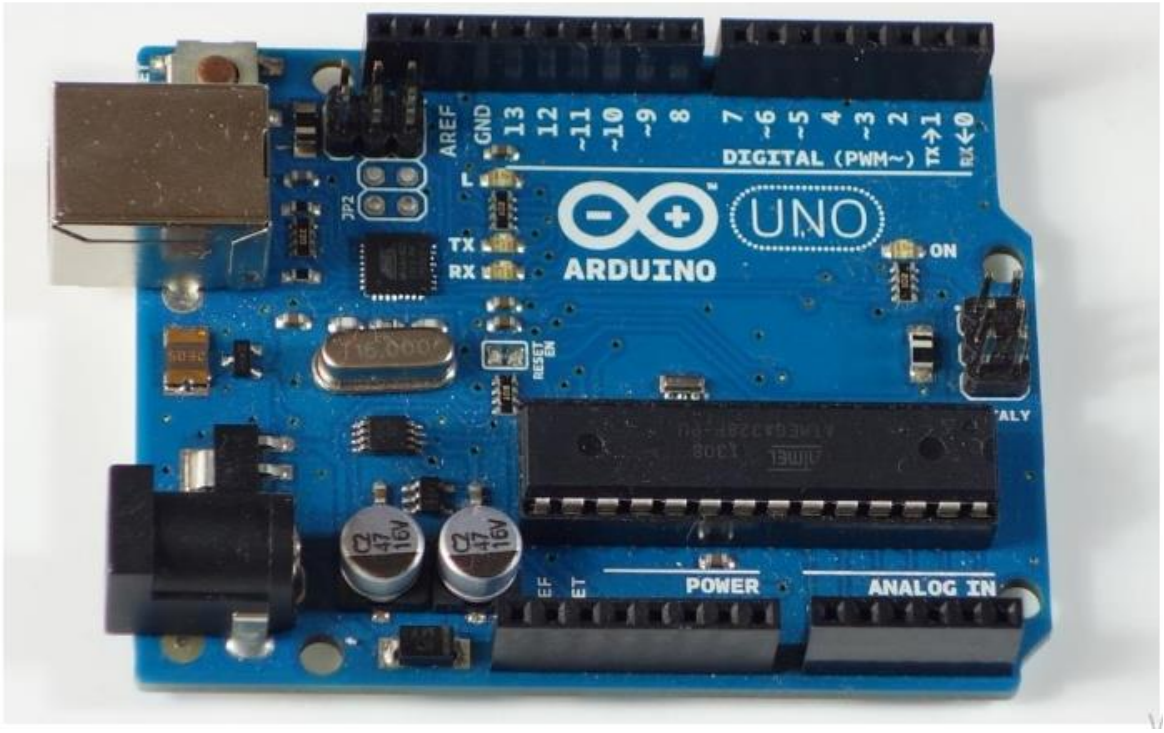
Arduino è una piattaforma open-source per la prototipazione, che combina hardware e software di facile utilizzo. Si compone di una scheda elettronica con microcontrollore, che può essere programmata, e di un software chiamato Arduino IDE (Integrated Development Environment), utilizzato per scrivere e caricare il codice dal computer sulla scheda fisica

In altre parole, Arduino è un piccolo "computer" che si può programmare per leggere informazioni dal mondo circostante e inviare istruzioni al mondo esterno. Tutto questo è possibile perché è possibile collegare diversi dispositivi e componenti ad Arduino per fare ciò che si desidera. Si possono realizzare progetti straordinari, non c'è limite a ciò che si può fare e con l'immaginazione tutto è possibile!

In parole povere, Arduino è una piattaforma basata su un microcontrollore che può essere programmato con le vostre istruzioni per interagire con varie forme di input e output. L'attuale modello di Arduino Uno ha dimensioni più piccole rispetto a quelle di una mano umana media

Che cos'è Arduino?

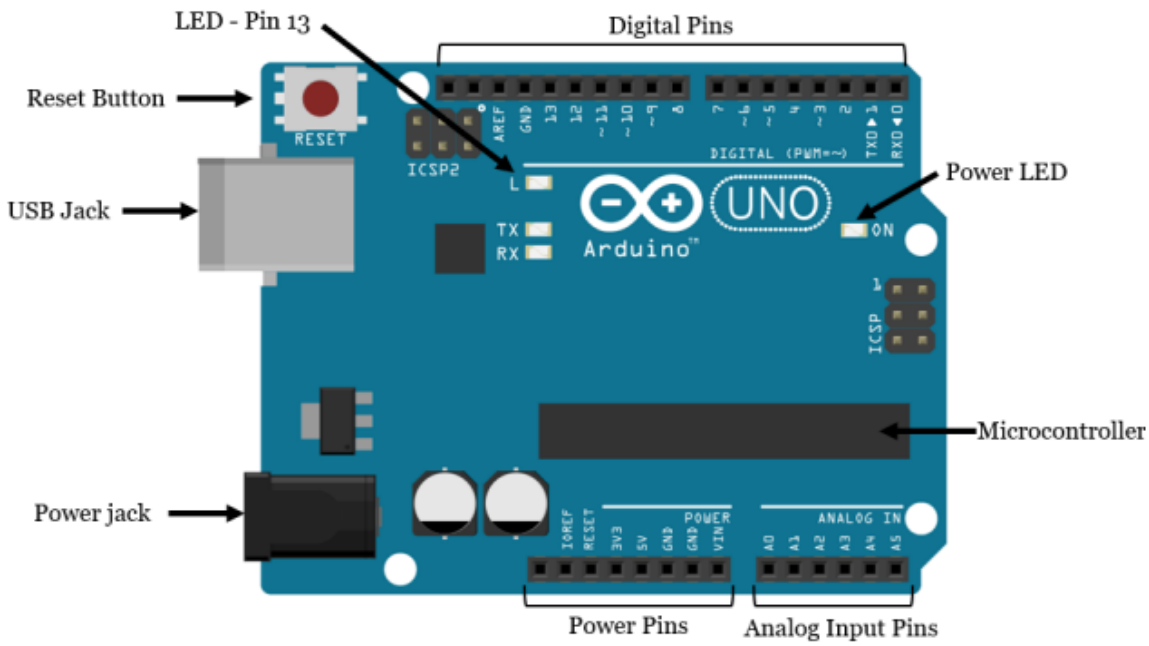
Arduino è la scheda mostrata nella figura seguente.



In pratica, Arduino è una piccola scheda elettronica di sviluppo con un 'cervello' (chiamato microcontrollore) che può essere collegato ai circuiti elettronici. In questo modo, è possibile leggere gli ingressi (ovvero raccogliere dati dall'esterno) e controllare le uscite (invio di istruzioni all'esterno). Il cervello di questa scheda, l'Arduino Uno, è un chip ATmega328p, dove vengono memorizzati i programmi che indicano ad Arduino cosa fare.

Esplorazione della scheda Arduino Uno

Nella figura che segue sono riportati i componenti principali di una scheda Arduino Uno. Vediamo cosa fa ogni parte.



· **Microcontrollore:** l'ATmega328p è il cervello di Arduino. Tutto ciò che è presente sulla scheda Arduino è destinato a supportare questo microcontrollore. È qui che si memorizzano i programmi per dire ad Arduino cosa fare.

· **Pin digitali:** Arduino dispone di 14 pin digitali, etichettati da 0 a 13, che possono essere configurati come ingressi o uscite. Quando sono impostati come ingressi, questi pin possono leggere la tensione. Possono leggere solo due stati: Quando sono impostati come uscite, questi pin possono applicare una tensione. Possono applicare solo 5V (HIGH) o 0V (LOW).

· **Pin PWM:** Sono i pin digitali contrassegnati da un ~ (pin 11, 10, 9, 6, 5 e 3). PWM è l'acronimo di “pulse width modulation” (modulazione dell'ampiezza degli impulsi) e consente a questi pin di emettere una tensione variabile, simulando "quantità di tensione" diverse. Più avanti si scoprirà qualcosa di più sulla PWM.

· **Pin TX e RX:** pin digitali 0 e 1. La T sta per “trasmissione” e la R per “ricezione”. Arduino utilizza questi pin per comunicare con altri dispositivi elettronici tramite comunicazione seriale. Arduino utilizza questi pin anche per comunicare con il computer durante il caricamento del codice. Evitare di utilizzare questi pin per altri compiti diversi dalla comunicazione seriale, a meno che non si sia a corto di pin.

· **LED collegato al pin digitale 13:** è utile per facilitare il debug degli sketch di Arduino.

· LED TX e RX: questi LED lampeggiano quando vengono inviate informazioni tra il computer e Arduino.

· **Pin analogici:** i pin analogici sono etichettati da A0 ad A5 e sono spesso utilizzati per leggere i segnali dei sensori analogici. Possono misurare tensioni variabili comprese tra 0 e 5V. Inoltre, possono essere configurati come pin di output/input digitale analogamente ai pin digitali.

· **Pin di alimentazione:** Arduino fornisce 3,3 V o 5 V attraverso questi pin. Questo è molto utile perché la maggior parte dei componenti richiede 3,3 V o 5 V per funzionare. I pin etichettati come "GND" sono i pin di massa.

· **Pulsante di reset:** quando si preme questo pulsante, il programma attualmente in esecuzione in Arduino viene riavviato. È presente anche un pin di reset accanto ai pin di alimentazione che funge da pulsante di reset. Applicando una breve tensione a questo pin, si resetta Arduino.

· **LED di accensione:** è acceso da quando Arduino è alimentato.

· **Presa USB:** per caricare i programmi dal computer alla scheda Arduino è necessario un cavo da USB A a USB B (mostrato nella figura seguente). Questo cavo alimenta anche Arduino.

· **Presa di alimentazione:** è possibile alimentare Arduino tramite la presa di alimentazione. La tensione di input consigliata è compresa tra 7V e 12V. Esistono diversi modi per alimentare Arduino: ad esempio, batterie ricaricabili, batterie usa e getta, alimentatori e pannelli solari.

Caratteristiche di Arduino

Arduino Uno monta il microcontrollore Atmel ATmega328P e dispone di una porta USB, un ingresso per la presa di alimentazione e un pulsante di reset. Arduino ha tutto ciò che un microcontrollore dovrebbe avere.

Microcontroller	Atmega328P
Tensione di lavoro	5V
Tensione di input (consigliata)	7-12V
Tensione di input (limite)	6-20V
Pin di input/output digitale	14
Pin di input/output PWM	6
Pin di input analogico	6
Corrente CC per pin di input/output	20mA
Corrente CC per 3,3 V	50mA
Memoria flash	32 KB
SRAM	2KB
EEPROM	1 KB
Velocità di clock	16 MHz
Lunghezza	68.6 mm
Larghezza	53.4 mm
Peso	25 g
Figura: Arduino Uno Features	

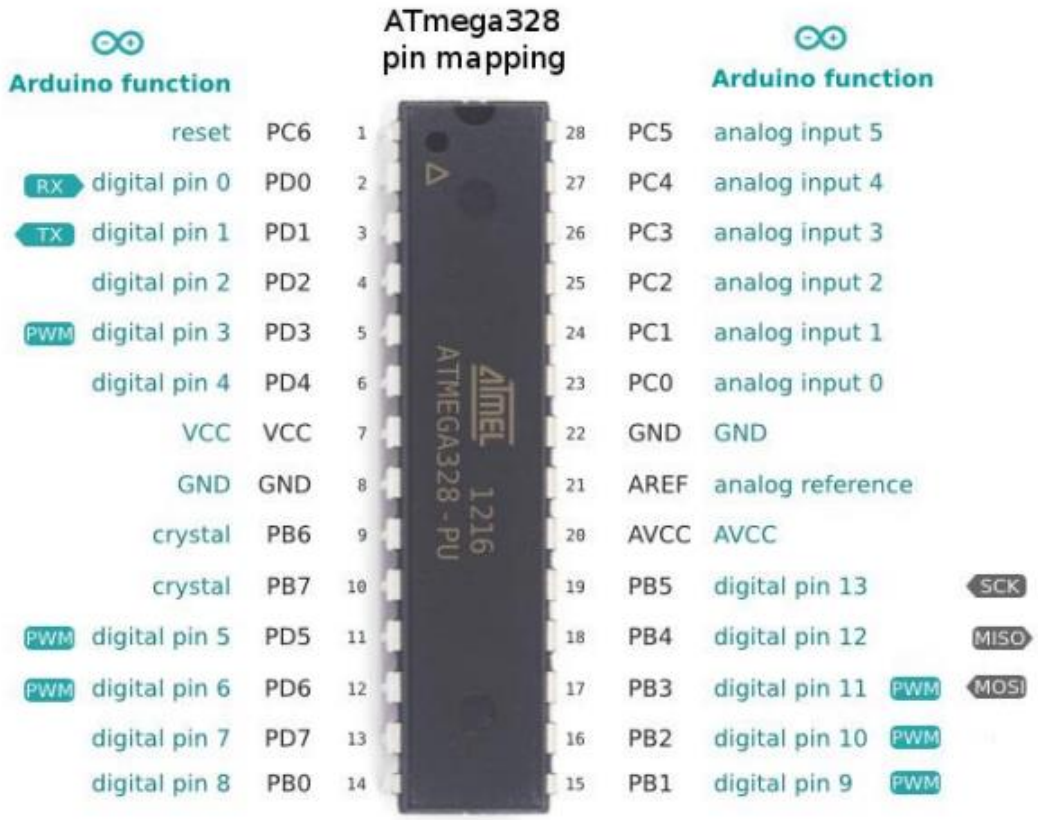


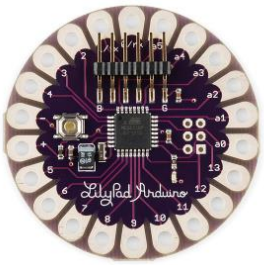
Figura: Atmega 328P Pins

ALTRI TIPI di ARDUINO

ARDUINO MEGA

È dotato di un microcontrollore Atmega 2560. Dispone di 54 pin di input e output digitali, 16 ingressi analogici, 4 porte seriali hardware e un oscillatore a cristallo da 16 mhz. È alimentata sia da USB che tramite adattatore CC. In genere questa scheda, che ha le stesse caratteristiche dell'Arduino UNO, viene preferita nei progetti più grandi perché dispone di un maggior numero di pin.	

ARDUINO LILYPAD

Lilypad è stato progettato per essere cucito su abiti e tessuti. In questo modo, può essere utilizzato in progetti creativi pensati per essere indossati. È dotato di un microcontrollore Atmega 168V.	

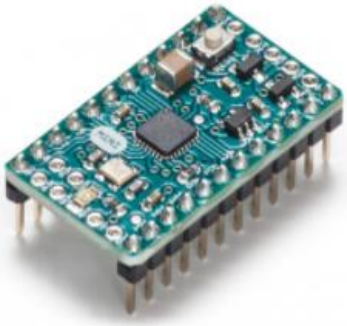
ARDUINO ETHERNET

Dispone di un chip Ethernet e di una porta Ethernet per la realizzazione di progetti connessi a Internet. È presente anche uno slot per scheda SD, mentre il microcontrollore utilizzato è il modello Atmega 328.	

ARDUINO BLUETOOTH

Su Arduino BT è presente un modulo Bluetooth, ideale per realizzare applicazioni che comunicano con il protocollo Bluetooth. Questo modulo può anche essere utilizzato per programmare Arduino tramite Bluetooth.	

ARDUINO MİNİ

È un modello Arduino progettato per essere utilizzato su una breadboard o integrato in un altro progetto. È dotato di microcontrollore modello Atmega 168 o Atmega 328. È ideale per le applicazioni in cui le dimensioni ridotte sono particolarmente importanti.	

ARDUINO NANO

Si tratta di un modello molto piccolo e progettato per applicazioni su circuito stampato, dotato di microcontrollore Atmega 328 o Atmega 168, regolatore di tensione, chip di conversione da seriale a USB, porta di input per tensione CC e porta mini USB.	

ARDUINO LEONARDO

È una delle schede Arduino che contiene un microcontrollore Atmega 32u4 il quale consente la connessione USB senza la necessità di un chip aggiuntivo. il quale consente la connessione USB senza la necessità di un chip aggiuntivo. Grazie alla connessione USB integrata, Arduino Leonardo può essere riconosciuto dal computer come un mouse o una tastiera (dispositivo HID.	

ARDUINO ESPLORA

Esplora è una scheda Arduino che contiene diversi sensori, a differenza delle altre. Grazie ai sensori presenti sulla scheda, è possibile eseguire molte applicazioni senza bisogno di altre aggiunte e di eccessive conoscenze elettroniche. Esplora è dotata di un potenziometro a scorrimento, un sensore di luce e suono, un sensore di temperatura, un generatore di suoni, un mini joystick analogico a 2 assi, un LED a 3 colori e un accelerometro. Esplora è inoltre dotato di un microcontrollore AVR Atmega 32U4 come Leonardo. È possibile sviluppare applicazioni che emulano mouse o tastiera quando la scheda è collegata a un computer tramite la sua connessione micro USB.	

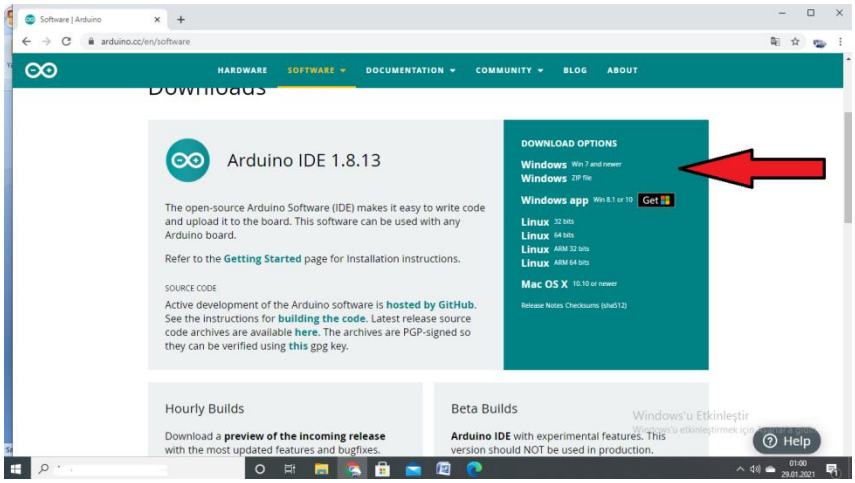
Scaricare l'IDE Arduino

L'IDE (Integrated Development Environment) di Arduino è l'ambiente in cui vengono sviluppati i programmi che diranno ad Arduino cosa fare.

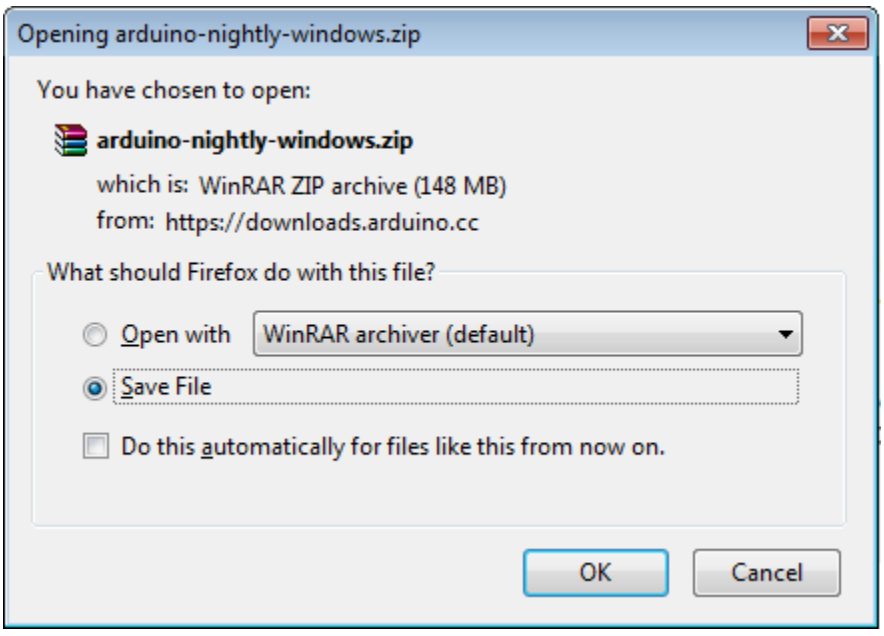
Per installare l'IDE di Arduino su Windows, è necessario seguire le istruzioni fornite.

Utilizzando l'IDE Arduino, è possibile caricare nuovi programmi sul microcontrollore, l'ATmega328P, tramite la connessione USB.

Per scaricare l'IDE Arduino, fare clic sul seguente link: https://www.arduino.cc/en/Main/Software.

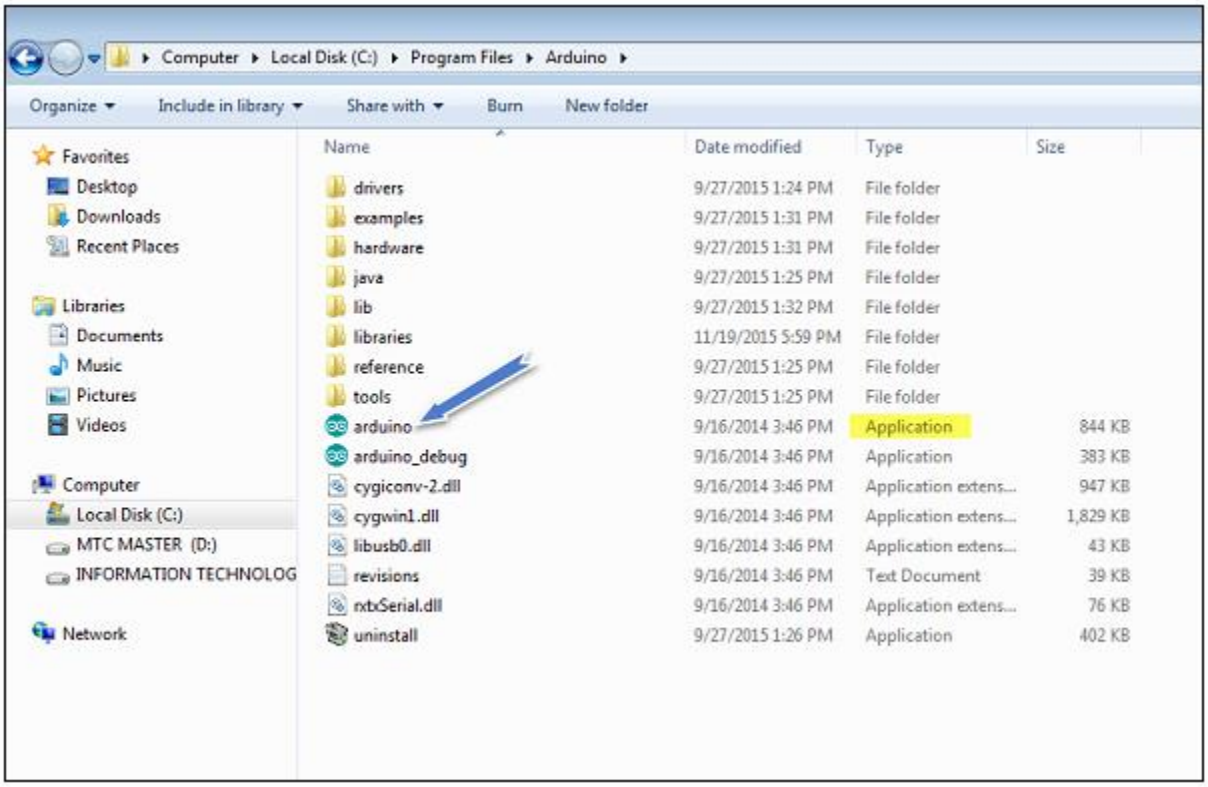


Selezionare il sistema operativo in uso e procedere con il download. Dopo aver scaricato il software Arduino IDE, dobbiamo decomprimere la cartella.



All'interno della cartella si trova l'icona dell'applicazione contrassegnata dal simbolo infinito (application.exe). Fare doppio clic sull'icona per avviare l'IDE.

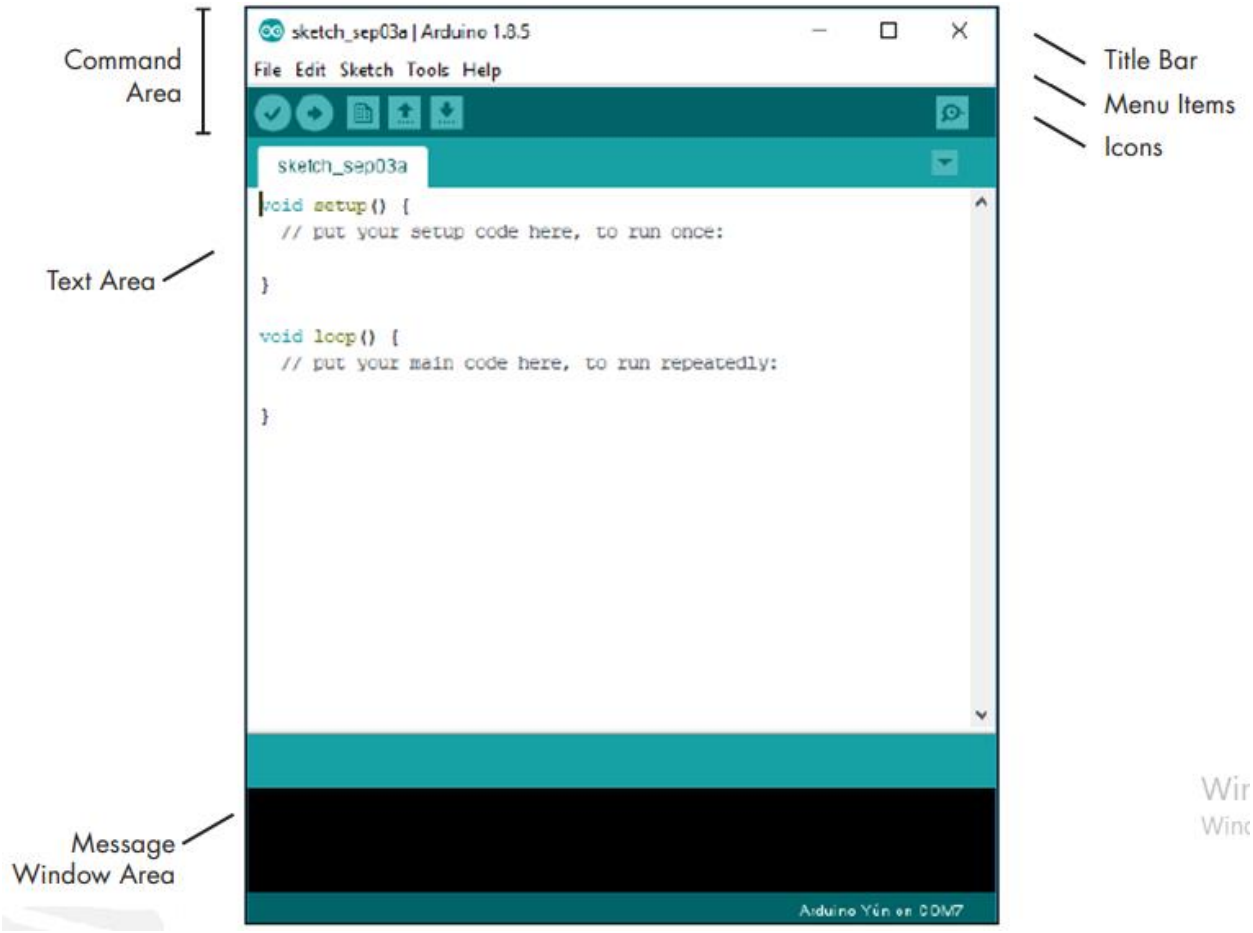
Successivamente, basta seguire la procedura guidata per completare l'installazione di Arduino IDE



Finestra dell'Arduino IDE utilizzata per scrivere i programmi

Quando si apre per la prima volta l'IDE Arduino, si dovrebbe vedere qualcosa di simile alla figura seguente.

Come mostrato nella figura seguente, l'IDE Arduino assomiglia a un semplice word processor. L'IDE è diviso in tre aree principali: l'area dei istruzioni, l'area del codice e l'area della finestra dei messaggi.



Voci di menu

Come in qualsiasi elaboratore di testi o editor di testo, è possibile fare clic su una delle voci di menu per visualizzarne le varie opzioni.

File: contiene le opzioni per salvare, caricare e stampare gli schizzi, una serie completa di schizzi di esempio da aprire e il sottomenu Preferenze.

Modifica: contiene le consuete funzioni di copia, incolla e ricerca comuni a qualsiasi elaboratore di testi.

Schizzo: Contiene la funzione di verifica dello schizzo prima di caricarlo su una scheda e alcune opzioni di cartella e importazione dello schizzo.

Strumenti: Contiene una serie di funzioni e di istruzioni per selezionare il tipo di scheda Arduino e la porta USB.

Aiuto: Contiene collegamenti a vari argomenti di interesse e alla versione dell'IDE.

Che cos'è lo sketch

Uno schizzo Arduino è un insieme di istruzioni create per svolgere un compito particolare; in altre parole, uno sketch è un programma.

Lo sketch non è altro che un insieme di istruzioni che Arduino deve eseguire. Gli schizzi creati con l'IDE Arduino vengono salvati come file .pde. Per creare uno sketch, è necessario realizzare le tre parti principali: la dichiarazione delle variabili, la funzione di impostazione e la funzione di loop principale.

Pulsanti della barra degli strumenti dell'IDE Arduino

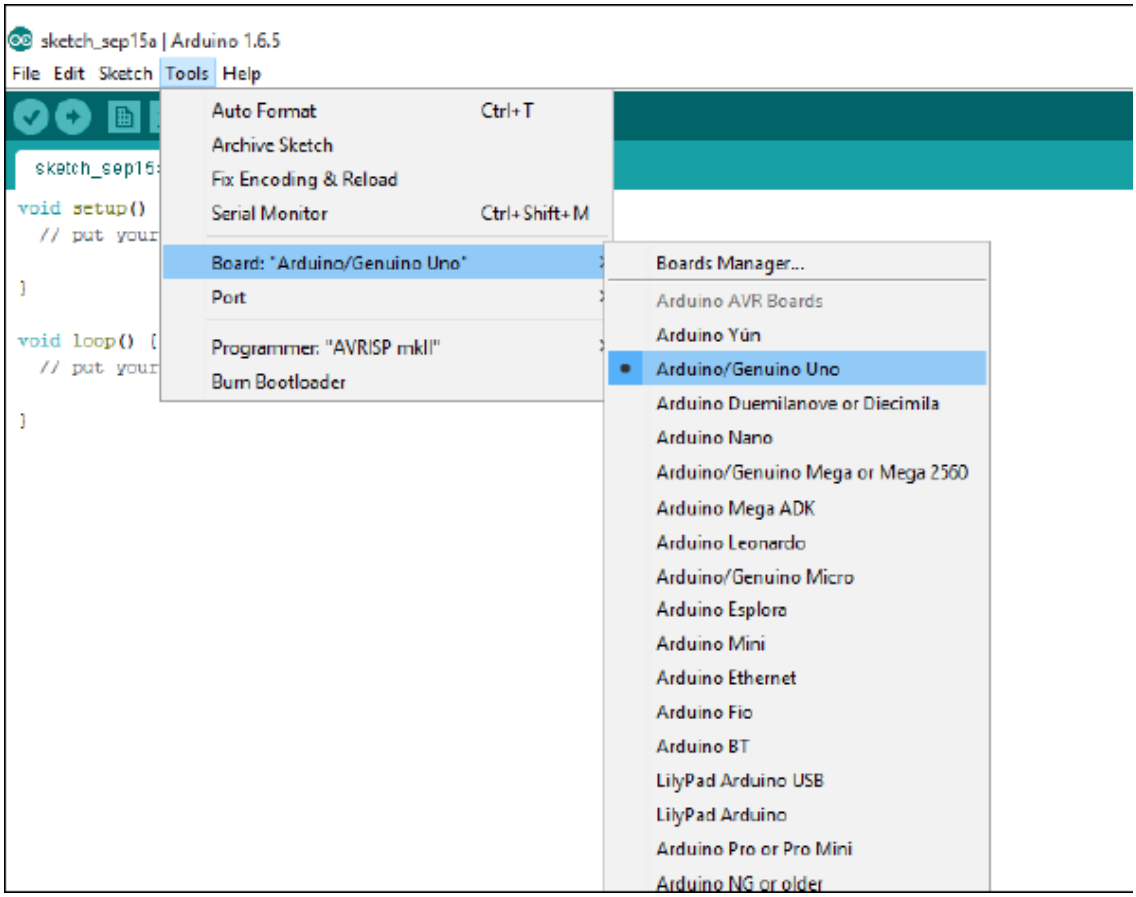
Sotto la barra degli strumenti del menu ci sono sei icone. Passare il mouse su ciascuna icona per visualizzarne il nome. Le icone, da sinistra a destra, sono le seguenti:

	**Verifica (Compilazione):** Fare clic su questa opzione per verificare che lo schizzo di Arduino sia valido e non contenga errori di programmazione.
	**Nuovo:** Fare clic su questa opzione per aprire un nuovo schizzo vuoto in una nuova finestra.
	**Apri:** fare clic su questa opzione per aprire uno schizzo salvato.
	**Salva:** Fare clic su questo per salvare lo schizzo aperto. Se lo schizzo non ha un nome, verrà richiesto di crearne uno.
	**Carica:** Fare clic su questa opzione per verificare e caricare lo sketch sulla scheda Arduino.
	**Monitor seriale:** Fare clic su questa opzione per aprire una nuova finestra da utilizzare per inviare e ricevere dati tra Arduino e l'IDE.

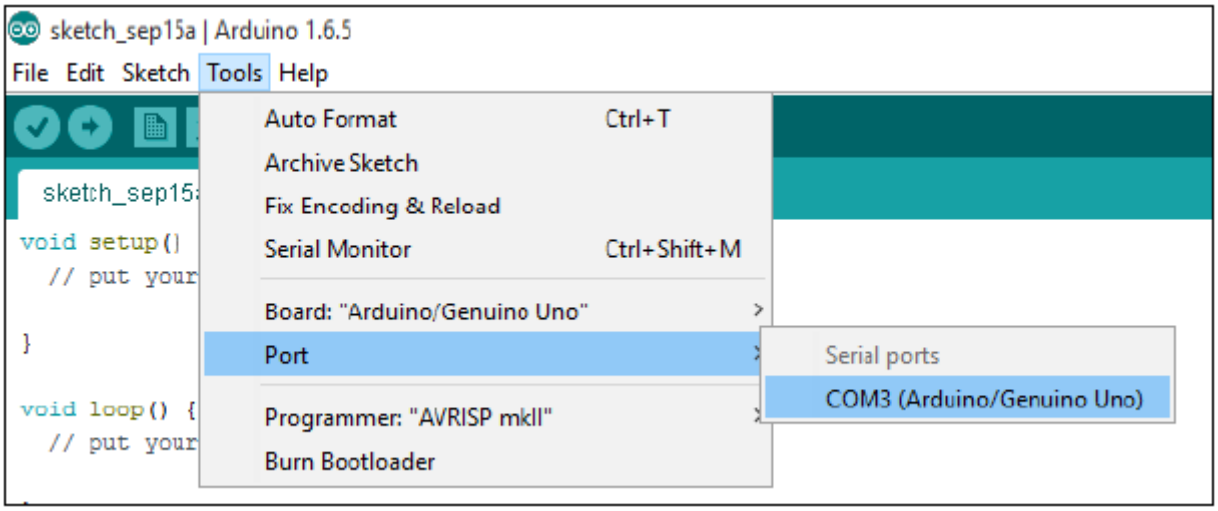
Collegamento di Arduino

Collegare Arduino UNO al computer tramite USB.
Dopo aver collegato Arduino con un cavo USB, è necessario assicurarsi che l'IDE Arduino abbia selezionato la scheda giusta.

Nel nostro caso, stiamo usando Arduino Uno, quindi dobbiamo andare su **Tools → Board: → Arduino/Genuino Uno.**



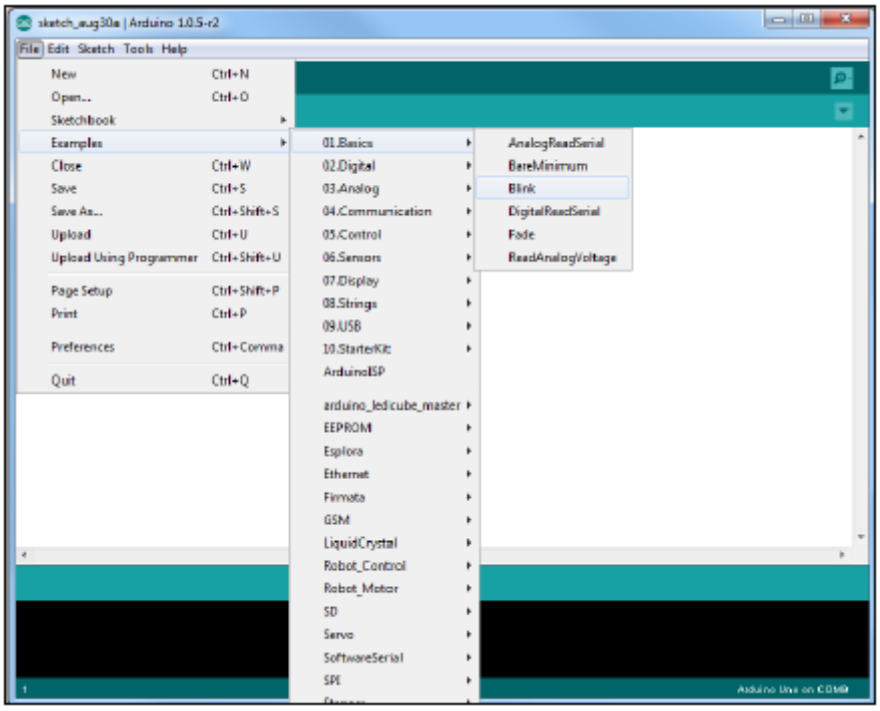
Quindi, è necessario selezionare la porta seriale a cui è collegato Arduino. Andare su **Tools→Port** e selezionare la porta giusta.



Caricamento di uno schizzo Arduino

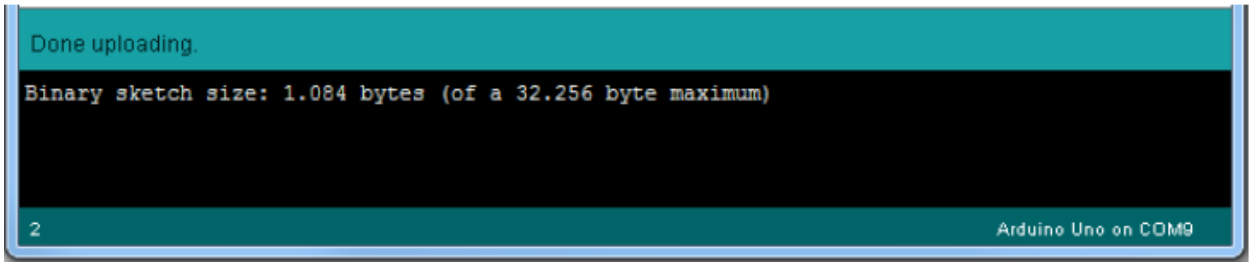
Per mostrarvi come caricare il codice sulla vostra scheda Arduino, vi mostreremo un semplice esempio. Si tratta di uno degli esempi più elementari: consiste nel far lampeggiare il LED di bordo o il pin digitale 13 ogni secondo.

1. Aprite l'IDE Arduino.
2. Andate al **File →Examples → 01.Basics → Blink**

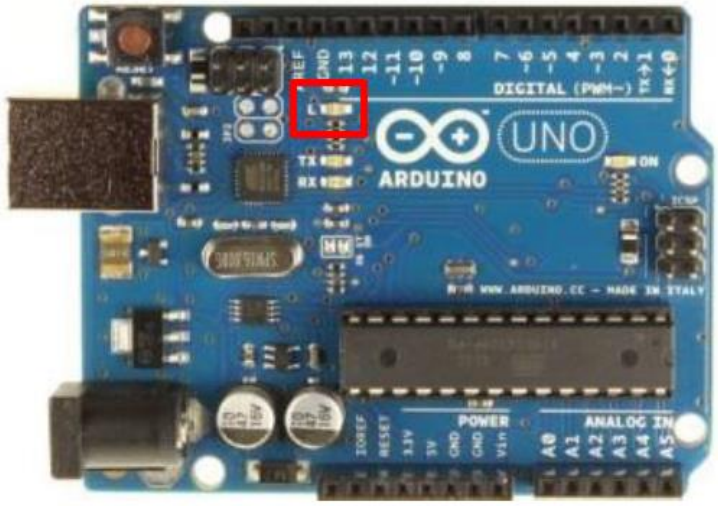


Per impostazione predefinita, l'IDE Arduino è preconfigurato per Arduino UNO. Fare clic sul pulsante **Upload** e attendere qualche secondo.

Dopo qualche secondo, dovrebbe apparire un messaggio di **Done uploading.**



Questo codice non fa altro che far lampeggiare il LED di bordo di Arduino UNO (evidenziato in rosso). Si dovrebbe vedere il piccolo LED accendersi per un secondo e spegnersi per un altro secondo, ripetutamente.



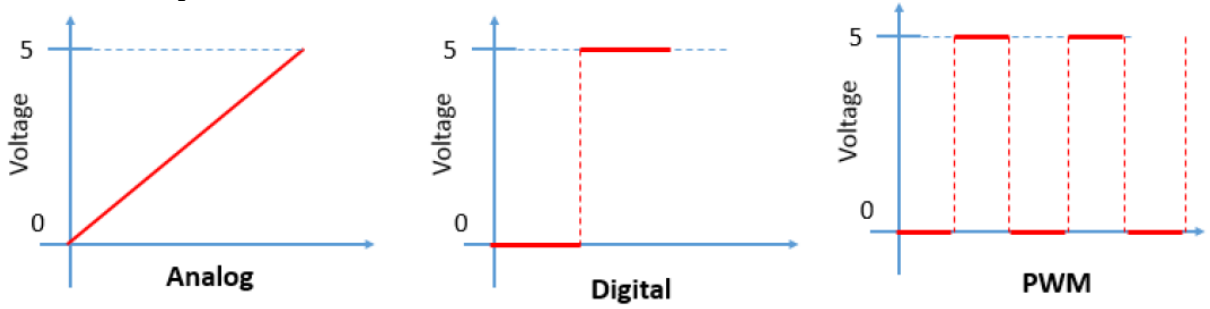
Controllo di un'output e lettura di un input
Una scheda Arduino dispone di pin digitali, analogici e PWM.

Differenza tra digitale, analogico e PWM
Nei pin digitali sono possibili solo due stati: acceso o spento. Questi possono anche essere indicati come Alto o Basso, 1 o 0 e 5V o 0V.

Ad esempio, se un LED è acceso, il suo stato è Alto o 1 o 5V. Se è spento, il suo stato è Basso o 0 o 0V.

Nei pin analogici, gli stati possibili sono 1024 e compresi tra 0 e 1023. Ciò consente di leggere i valori dei sensori. Ad esempio, utilizzando un sensore di luce, si otterrà un valore di 1023 in condizioni di oscurità totale e 0 in caso di luce intensa. Per livelli di luminosità intermedi, il valore sarà compreso tra 0 e 1023.

I pin PWM sono **pin digitali**, quindi emettono 0 o 5V. Tuttavia, grazie alla tecnica di 'modulazione di larghezza di impulso' (PWM), questi pin possono simulare valori di tensione intermedi tra 0 e 5V. La PWM consente di creare livelli di potenza variabili facendo oscillare rapidamente la tensione di output tra 0 e 5V.



Controllo di un'output

Per controllare un'output digitale si usa la funzione digitalWrite() e tra le parentesi si scrive il pin che si vuole controllare e poi HIGH o LOW.

Per controllare un pin PWM si usa la funzione analogWrite() e tra le parentesi si scrive il pin che si vuole controllare e un numero compreso tra 0 e 255.

Lettura di un input

Per leggere un input analogico si utilizza la funzione analogRead() e per un input digitale si utilizza digitalRead().

Nota: il modo migliore per imparare Arduino è fare pratica. Quindi, realizzate molti progetti e iniziate a costruire qualcosa.

Co-funded by the
Erasmus+ Programme
of the European Union

Erasmus+ KA210-VET

Partenariati su piccola scala nella

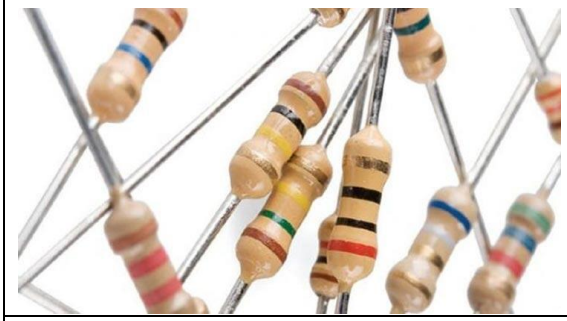
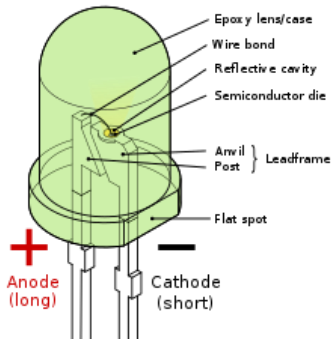
istruzione e formazione professionale

Titolo del progetto: "Using Arduinos in Vocational Training"

Acronimo del progetto: "UsingARDinVET"

Nr del progetto: "2023-1-RO01-KA210-VET-000156616"

Modulo di input/output Arduino e kit di formazione

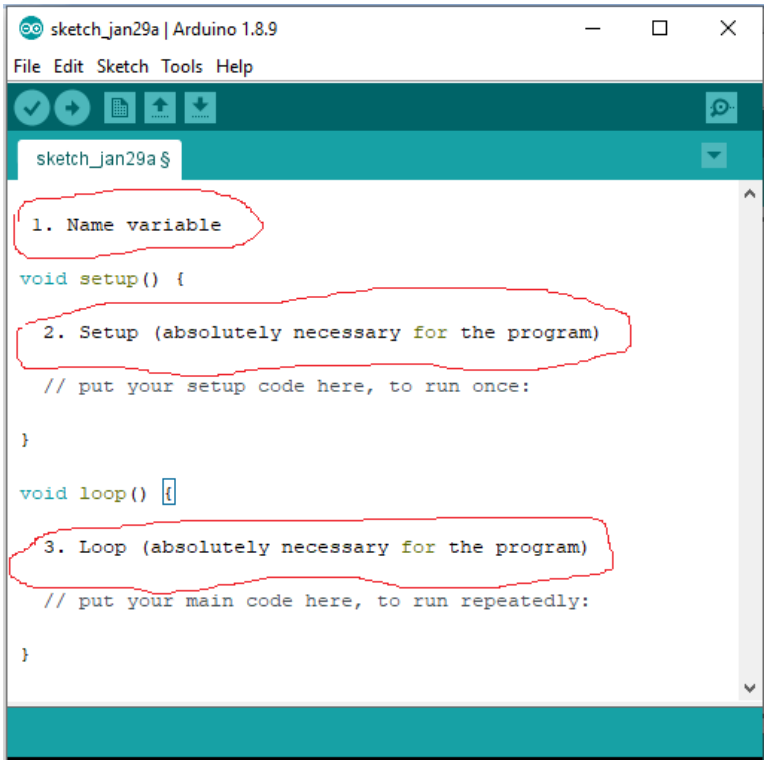
<table>
<tr>
<td>  </td>
<td>  </td>
</tr>
<tr>
<td> 
</td>
<td>  </td>
</tr>
</table>

Pianificare i nostri progetti

Quando si iniziano i primi progetti, si potrebbe essere tentati di scrivere lo sketch subito dopo aver avuto una nuova idea. Ma prima di iniziare a scrivere, è necessario eseguire alcune fasi preparatorie di base. Dopo tutto, la scheda Arduino non legge nel pensiero; ha bisogno di istruzioni precise e, anche se queste istruzioni possono essere eseguite da Arduino, i risultati potrebbero non essere quelli attesi se si trascura anche un piccolo dettaglio.
Sia che stiate creando un progetto che si limita a far lampeggiare una luce o un segnale ferroviario automatizzato, un piano dettagliato è la base del successo. Quando progettate i vostri progetti Arduino, seguite questi passaggi fondamentali:
1. Definire l'obiettivo. Stabilite cosa volete ottenere.
2. Scrivere l'algoritmo. Un algoritmo è un insieme di istruzioni che descrive come realizzare il progetto. L'algoritmo elenca i passaggi necessari per raggiungere l'obiettivo del progetto.
3. Selezionare l'hardware. Stabilite come collegarlo ad Arduino.
4. Scrivere lo sketch. Creare il programma iniziale che indichi ad Arduino cosa fare.
5. Collegare il tutto. Collegare l'hardware, i circuiti e altri elementi alla scheda Arduino.
6. Test e debug. Funziona? In questa fase si identificano gli errori e se ne individuano le cause, che siano nello sketch, nell'hardware o nell'algoritmo.
Più tempo si dedica alla pianificazione del progetto, più facile sarà la fase di test e debug.

Struttura di base di uno schizzo

Il programma di Arduino è chiamato “schizzo”. Uno schizzo può essere suddiviso in tre parti.



1. Nome della variabile:
Nella prima parte vengono nominati gli elementi del programma. Questa parte non è assolutamente obbligatoria.

2. Setup (assolutamente necessaria per il programma):
L'impostazione verrà eseguita una sola volta. Qui si dice al programma, ad esempio, quale (slot per i cavi) deve essere un input e un output sulle schede.
Definito come output: Il pin deve emettere una tensione. Ad esempio: Con questo pin un LED si accende.
Definito come input: La scheda deve leggere una tensione. Ad esempio: Un interruttore viene azionato. La scheda lo riconosce perché riceve una tensione sul pin di input.

3. Loop (assolutamente necessario per il programma):
Questa parte di loop verrà ripetuta continuamente dalla scheda. Esegue lo sketch dall'inizio alla fine e ricomincia dall'inizio e così via.

Ulteriori regole di sintassi

È necessario prestare attenzione a queste regole durante la scrittura dei programmi Arduino. In caso contrario, il nostro programma fallirà.

; (Punto e virgola):

; (Punto e virgola) viene utilizzato per terminare un'istruzione. Se si dimentica di terminare una riga con un punto e virgola, si ottiene un errore del compilatore.

Esempio: int a=13;

{} (parentesi graffe):

Le parentesi graffe sono una parte importante del linguaggio di programmazione Arduino. Vengono utilizzate in diversi costrutti e questo può talvolta confondere i principianti. Una parentesi graffa di apertura "{" deve sempre essere seguita da una parentesi graffa di chiusura "}".

I principali usi delle parentesi graffe: Funzioni, cicli, dichiarazioni condizionali

Esempio:

```
void myfunction(datatype argument) { statements(s) }
```

// (commento a una riga) e /* */ (commento a più righe):

Si tratta di righe del programma utilizzate per informare se stessi o altri sul funzionamento del programma. Vengono ignorati dal compilatore e non vengono esportati nel processore, quindi non occupano spazio sul chip Atmega.
I commenti hanno il solo scopo di aiutare l'utente a capire (o ricordare) il funzionamento del

programma o di informare gli altri sul funzionamento del programma. Esistono due modi diversi per contrassegnare una riga come commento:

Esempio:

x = 5; // Questo è un commento su una sola riga. Tutto ciò che si trova dopo gli slash è un commento // fino alla fine della riga.

x = 5; /*Questo è un commento su più righe. ..... Questa è la fine di un commento su più righe. */

#define:

#define permette al programmatore di dare un nome a un valore costante prima che il programma venga compilato. Le costanti definite in arduino non occupano spazio nella memoria del programma sul chip. In generale, la parola chiave const è preferibile per definire le costanti e dovrebbe essere usata al posto di #define.

Esempio:

#define ledPin 3 // Il compilatore sostituirà ogni riferimento a ledPin con il valore 3 al momento della compilazione.

#include:

#include viene utilizzato per includere librerie esterne nello sketch. Ciò consente al programmatore di accedere a un ampio gruppo di librerie C standard (gruppi di funzioni preconfezionate) e anche a librerie scritte appositamente per Arduino.

Esempio:

#include <servo.h>

Arduino - Tipi di dati

I tipi di dati vengono utilizzati per dichiarare variabili o funzioni di tipo diverso. Il tipo di variabile determina lo spazio che occupa nella memoria e come il modello di bit memorizzato viene interpretato.

Le espressioni utilizzate per memorizzare qualsiasi informazione in memoria e che possono cambiare il valore durante il flusso del programma sono chiamate variabili. Le variabili possono essere numeri, caratteri o espressioni logiche. Il tipo di dati appropriato deve essere selezionato in base al tipo di variabile. A seconda del tipo di dati utilizzati per definire la variabile in memoria, viene allocata una determinata area.

La tabella seguente fornisce tutti i tipi di dati che verranno utilizzati durante la programmazione di Arduino.

Tipo	**Contenuto**
boolean	può contenere sia vero che falso
char	da -128 a 127
byte	da 0 a 255
unsigned char	da 0 a 255
int	da -32,768 a 32,767
unsigned int	da 0 a 65,535
word	(come unsigned int)
long (or long int)	da -2,147,483,648 a 2,147,483,647
unsigned long	da 0 a 4,294,967,295
float	da -3.4028235E+38 a 3.4028235E+38
double	(come float)

Arduino - Variabili e costanti

Durante la definizione della variabile, è necessario determinare il nome della variabile, il valore della variabile e il tipo di dati appropriato per la variabile.

La definizione viene effettuata come si vede nell'esempio seguente:

int LED =12;

Ecco: **int**=tipo di dati **LED**= nome della variabile **12**=valore variabile

Le variabili, che Arduino utilizza, hanno una proprietà chiamata *scope* (ambito). L'ambito definisce una regione del programma e ci sono tre posti in cui le variabili possono essere dichiarate. Essi sono:

- All'interno di una funzione o di un blocco, che si chiama variabile locale.
- Nella definizione dei parametri di una funzione, chiamati parametri formali.
- Al di fuori di tutte le funzioni, chiamate variabili globali.

Una costante è un qualificatore di variabile che modifica il comportamento della variabile, rendendola "di sola lettura". Ciò significa che la variabile può essere utilizzata come qualsiasi altra variabile del suo tipo, ma il suo valore non può essere modificato. Se si tenta di assegnare un valore a una variabile const, si ottiene un errore del compilatore.
Le costanti definite con la parola chiave const **seguono le stesse regole di scoping delle variabili** Per questo motivo e per le insidie dell'uso di #define, la parola chiave const è un metodo migliore per definire le costanti ed è preferibile all'uso di #define.

Esempio:

```
const float pi = 3.14;
```

Note:

#define o const: È possibile utilizzare const o #define per creare costanti numeriche o stringhe. Per gli array, è necessario usare const. In generale, const è preferibile a #define per definire le costanti

Arduino - Operatori

Un operatore è un simbolo che indica al compilatore di eseguire specifiche funzioni matematiche o logiche. In Arduino si possono usare i seguenti tipi di operatori.

Operatori aritmetici:

Supponiamo che la variabile A contenga 10 e che la variabile B contenga 20, allora -

Nome dell'operatore	**Simbolo dell'operatore**	**Esempio**
operatore di assegnazione	=	A = B
addizione	+	A + B darà 30
sottrazione	-	A - B darà -10
moltiplicazione	*	A * B darà 200
divisione	/	B / A darà 2
modulo	%	B % A darà 0

Operatori di confronto:

Supponiamo che la variabile A contenga 10 e la variabile B contenga 20, allora:

Nome dell'operatore	**Simbolo dell'operatore**	**Esempio**
uguale a	==	(A == B) è falso
non uguale a	!=	(A != B) è vero
minore di	<	(A < B) è vero
maggiore di	>	(A > B) è falso
minore di o uguale a	<=	(A <= B) è vero
maggiore di o uguale a	>=	(A >= B) è falso

Operatori booleani

Supponiamo che la variabile A contenga 10 e la variabile B contenga 20, allora:

Nome dell'operatore	**Simbolo dell'operatore**	**Esempio**
e	&&	(A && B) è vero
o	||	(A || B) è vero
non	!	!(A && B) è falso

Operatori bit a bit

Si supponga che la variabile A contenga 60 e la variabile B contenga 13, allora:

Nome dell'operatore	Simbolo dell'operatore	Esempio
e	&	(A & B) darà 12 che è 0000 1100
o	|	(A | B) darà 61 che è 0011 1101
xor	^	(A ^ B) darà 49 che è 0011 0001
non	~	(~A) darà -60 che è 1100 0011
spostamento a sinistra	<<	A << 2 darà 240 che è 1111 0000
spostamento a destra	>>	A >> 2 darà 15 che è 0000 1111

Arduino - Funzioni di I/O (istruzioni)

Le funzioni consentono di strutturare i programmi per eseguire singoli compiti. Il caso tipico per la creazione di una funzione è quando è necessario eseguire la stessa azione più volte in un programma. Queste funzioni saranno più chiare quando mostreremo esempi reali di circuiti e programmi su Arduino. Per facilitare la spiegazione delle varie funzioni di istruzione, le abbiamo suddivise in istruzioni separate.

I pin della scheda Arduino possono essere configurati come ingressi o uscite. Di seguito spiegheremo il funzionamento dei pin in queste modalità. È importante notare che la maggior parte dei pin analogici di Arduino può essere configurata e utilizzata esattamente come i pin digitali.

Funzione pinMode():

La funzione pinMode() viene utilizzata per configurare un pin specifico in modo che si comporti come input o come output. È possibile abilitare le resistenze di pull-up interne con la modalità INPUT_PULLUP.

Sintassi: pinMode(pin, mode)

```
Void setup () {
 pinMode (pin , mode);
                }
```

Parametri: pin: il numero del pin di cui desideri impostare la modalità INPUT, OUTPUT, or INPUT_PULLUP.

Esempi:

```
pinMode(13, OUTPUT);      // imposta il pin digitale 13 come output
pinMode(5, INPUT);        //imposta il pin digitale 5 come input
```

Funzione digitalWrite()

La funzione digitalWrite() serve a scrivere un valore HIGH o LOW su un pin digitale. Se il pin è stato configurato come OUTPUT con pinMode(), la sua tensione sarà impostata sul valore corrispondente: 5V per HIGH, 0V (massa) per LOW. Se il pin è configurato come INPUT, digitalWrite() abilita (HIGH) o disabilita (LOW) la resistenza di pull-up interna sul pin. Si consiglia di impostare il pin utilizzando pinMode() con l'opzione INPUT_PULLUP, per abilitare la resistenza di pull-up interna.

Se non si imposta pinMode() su OUTPUT e si collega un LED a un pin, quando si richiama digitalWrite(HIGH), il LED potrebbe apparire poco luminoso. Questo accade perché, senza una configurazione esplicita tramite pinMode(), la funzione digitalWrite() abilita la resistenza di pull-up interna, che agisce come una resistenza di limitazione della corrente.

Sintassi:

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

pin: il numero del pin di cui si vuole impostare la modalità

value:HIGH (1), or LOW(0).

Esempio:

```
digitalWrite(LED, HIGH);      // accensione del led
digitalWrite(LED, LOW);       // spegnimento del led
```

Funzione digitalRead()

La funzione digitalRead() legge il valore di un pin digitale specificato, restituendo un valore HIGH (alto) o LOW (basso)

Sintassi: digitalRead(pin) pin: il numero del pin di Arduino che si desidera leggere.

Esempi:

```
val = digitalRead(inPin); // legge il pin di ingresso
```

Nota: i pin di ingresso analogici possono essere utilizzati come pin digitali, denominati A0, A1, ecc.

Funzione delay()

La funzione dalay() mette in pausa il programma per il tempo (in millisecondi) specificato come parametro. (In un secondo ci sono 1000 millisecondi).

Sintassi: delay(ms):

ms: il numero di millisecondi di pausa. Tipi di dati ammessi: unsigned long.

Esempi:

```
delay(1000);                    // attende per 1 secondo
delay(2000);                    // attende per 2 secondi
```

Funzione analogWrite():

La funzione analogWrite() scrive un valore analogico (onda PWM) su un pin. Può essere utilizzata per regolare la luminosità di un LED o per controllare la velocità di un motore. Dopo una chiamata a analogWrite(), il pin genera un'onda rettangolare costante del duty cycle specificato fino alla successiva chiamata a analogWrite() (o a digitalRead() o digitalWrite()) sullo stesso pin.

Esempi:

Imposta l'uscita al LED in modo proporzionale al valore letto dal potenziometro.

```
val = analogRead(analogPin);   //leggere il pin di input
analogWrite(ledPin, val / 4);   // I valori di lettura analogica vanno da 0 a 1023, i valori di scrittura analogica da 0 a 255.
```

Funzione analogRead( )

Arduino è in grado di rilevare se c'è una tensione applicata a uno dei suoi pin e di segnalarlo attraverso la funzione digitalRead(). Esiste una differenza tra un sensore on/off (che rileva la presenza di un oggetto) e un sensore analogico, il cui valore cambia continuamente. Per leggere questo tipo di sensore, abbiamo bisogno di un tipo diverso di pin.

Nella parte inferiore destra della scheda Arduino si trovano sei pin contrassegnati dalla dicitura "Analog In". Questi pin speciali non solo dicono se c'è una tensione applicata ad essi, ma anche il suo valore. Utilizzando la funzione analogRead(), possiamo leggere la tensione applicata a uno dei pin.

Questa funzione restituisce un numero compreso tra 0 e 1023, che rappresenta tensioni tra 0 e 5 volt. Ad esempio, se al pin analogico 0 è applicata una tensione di 2,5 V, analogRead(0) restituirà 512.

Sintassi: analogRead(pin);

pin: il numero del pin di ingresso analogico da leggere, da 0 a 5.

Esempio:

```
val = analogRead(analogPin);    // leggere il pin input
Serial.println(val);            // valore di debug
```

La funzione if:

L'istruzione if controlla una condizione ed esegue l'istruzione o l'insieme di istruzioni seguenti se la condizione è "vera".

Sintassi:

if (condition)

{ //statement(s) }

condizione: un'espressione booleana (cioè, può essere vera o falsa).

Esempi: (Le parentesi possono essere omesse dopo un'istruzione if. In questo caso, la riga successiva (definita dal punto e virgola) diventa l'unica istruzione condizionata.

```
if (x > 120) digitalWrite(LEDpin, HIGH);
```

```
if (x > 120)
digitalWrite(LEDpin, HIGH);
```

```
if (x > 120) {digitalWrite(LEDpin, HIGH);}

if (x > 120) {
  digitalWrite(LEDpin1, HIGH);
  digitalWrite(LEDpin2, HIGH);
}
            // sono tutte corrette
```

Istruzione if-else:

L'istruzione if…else consente un maggiore controllo sul flusso del codice rispetto alla semplice istruzione if, permettendo di raggruppare più test. Una clausola else (se presente) verrà eseguita se la condizione nell'istruzione if risulta falsa. L'else può proseguire con un altro test if, in modo da poter eseguire contemporaneamente test multipli che si escludono a vicenda**.**

Ogni test procederà al successivo fino a quando non verrà trovato un test vero. Quando si trova un test vero, viene eseguito il blocco di codice associato, e il programma salta poi alla riga successiva all'intera costruzione if/else. Se nessun test risulta vero, viene eseguito il blocco else predefinito, se presente, e definisce il comportamento predefinito. È consentito un numero illimitato di ramificazioni else if.

Sintassi:

```
if (condition is TRUE) {
  // eseguire operazione A
}
else
  //se NO, eseguire operazione B
}
```

```
if (condition1) {
  // eseguire operazione A
}
else if (condition2) {
  // eseguire operazione B
}
else {
  // eseguire operazione C
}
```

Esempi: (Di seguito è riportato un estratto di un codice per un sistema di monitoraggio della temperatura)

```
if (temperature >= 70) {

  //Pericolo! Spegnere il sistema.

}

else if (temperature >= 60) { // 60 <= temperature < 70
  // Attenzione! Richiesta attenzione dell'utente.
```

```
}
else { // temperatura < 60
  // Sicuro! Continuare le attività abituali..
}
```

Istruzione for:
L'istruzione for viene utilizzata per ripetere un blocco di istruzioni racchiuso tra parentesi graffe. Di solito, si usa un contatore, che viene incrementato ad ogni iterazione, per terminare il ciclo L'istruzione for è utile per qualsiasi operazione ripetitiva e viene spesso utilizzata in combinazione con gli array per operare su insiemi di dati o pin.

Sintassi:

```
for (inizializzazione; condizione; incremento) {

 // istruzione(i);

}
```

Parametri:

inizializzazione: avviene per prima e soltanto una volta.

condizione: ogni volta che il ciclo viene eseguito, la condizione viene testata; se è vera, il blocco di istruzioni viene eseguito, quindi l'incremento viene applicato e la condizione viene testata di nuovo. Quando la condizione diventa falsa, il ciclo termina.

incremento: viene eseguito ad ogni iterazione del ciclo quando la condizione risulta ver.

Esempi:

```
 for (int i = 0; i <= 255; i++)              {
     analogWrite(PWMpin, i);                  }

for (int x = 2; x < 100; x = x * 1.5) {
          println(x);                          }

 for (int i = 0; i > -1; i = i + x)   {
   analogWrite(PWMpin, i);        }
```

Istruzione switch...case

Come le istruzioni if, il costrutto switch/case controlla il flusso dei programmi permettendo ai programmatori di specificare diversi blocchi di codice che devono essere eseguiti in base a condizioni differenti. In particolare, un'istruzione switch confronta il valore di una variabile con i valori specificati nelle istruzioni case. Quando viene trovata un'istruzione case il cui valore corrisponde a quello della variabile, viene eseguito il codice contenuto in quella istruzione case.

La parola chiave break consente di uscire dall'istruzione switch ed è generalmente utilizzata alla fine di ogni case. Senza un'istruzione break, l'istruzione switch continuerà ad eseguire le espressioni successive ("fall-through") fino a quando non viene raggiunto un break, o la fine dell'istruzione switch.

Sintassi:

```
switch (var) {
  case label1:
    // istruzioni
    break;
  case label2:
    // istruzioni
    break;
  default:
    // statements
    break;
}
```

Codice di esempio:

```
switch (var) {
  case 1:
    //Eseguiere un'azione quando la variabile è uguale a 1;
  case 2:
    //Esegui un'azione quando la variabile è uguale a 2
    break;
  default:
    // Se nessun altro caso corrisponde, esegui l'operazione predefinita.
    // L'operazione predefinita è optionale
    break;
}
```

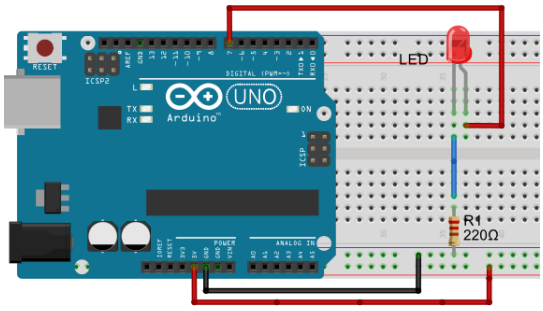
var: una variabile il cui valore deve essere confrontato con vari casi. Tipi di dati consentiti: int, char.
label1, label2: costanti. Tipi di dati consentiti: int, char.

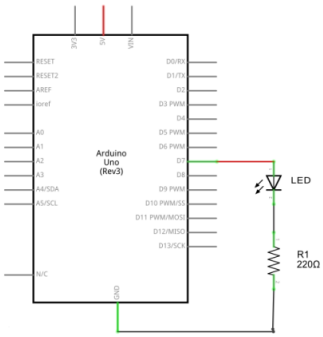
Nota:

I circuiti di Arduino sono mostrati in due modalità:

1-) Vista su breadboard

2-) Vista schema





1-) Vista su Breadboard

2-) Vista schema

Useremo la vista Breadboard, che è quella più utilizzata.

Basta! Facciamo qualcosa!

Circuiti del Kit Modulo Pulsante e LED di Arduino

La maggior parte dei pin di Arduino possono essere configurati come input o output. Il Kit Modulo Pulsante e LED è il primo KIT di formazione per l'uso di ARDinVET, progettato per insegnare agli studenti i sistemi I/O di Arduino. Pertanto, può essere definito come un KIT di formazione I/O per Arduino. In questo Kit di formazione, come mostrato di seguito, 6 pulsanti sono collegati ad Arduino come input. Un buzzer (cicalino), un connettore a 2 pin e 6 LED sono collegati come output.

Poiché gli studenti possono utilizzare questo kit di formazione per apprendere i sistemi I/O di Arduino, questo kit di formazione rende più facile testare i circuiti Arduino. Inoltre, possono utilizzare una breadboard per testare i circuiti e fare esperimenti con Arduino.

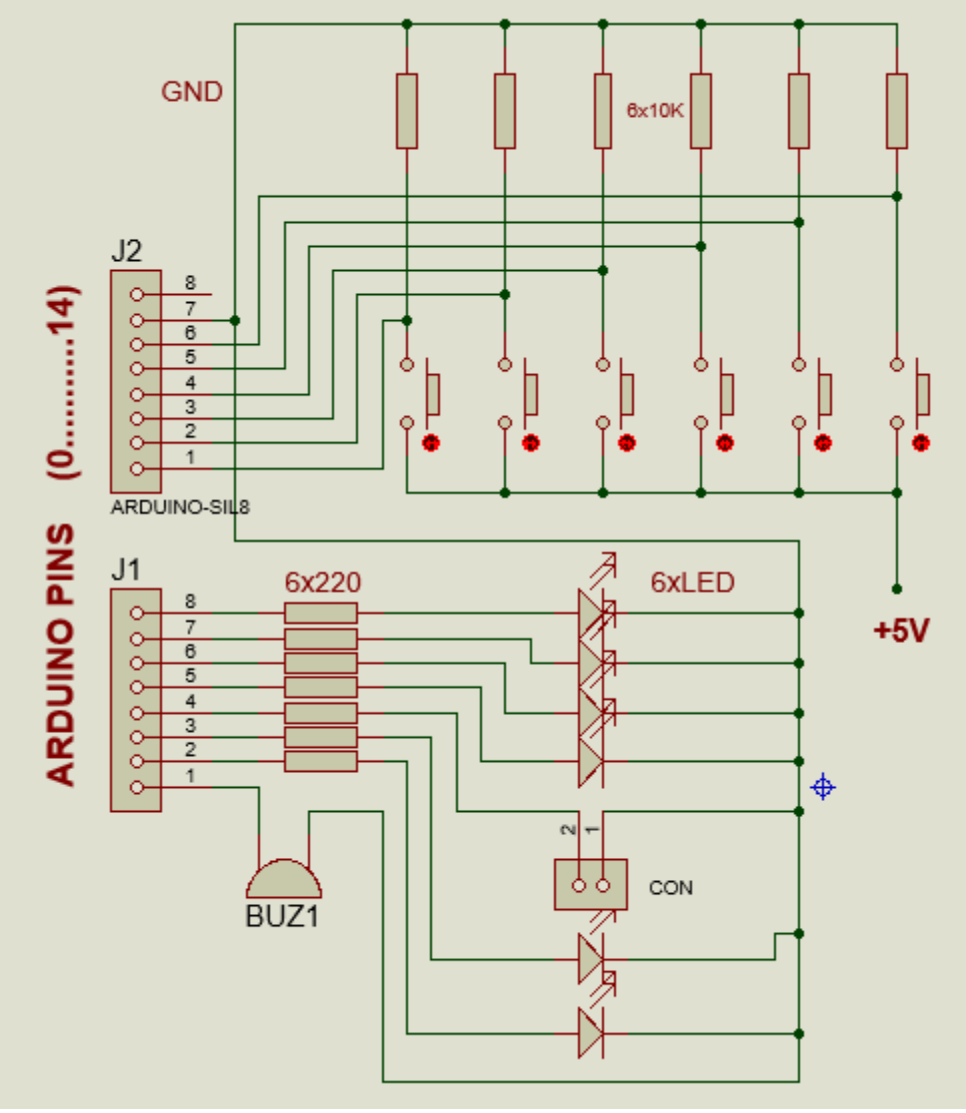


Figura: Circuito elettrico del Kit Modulo Pulsante e LED

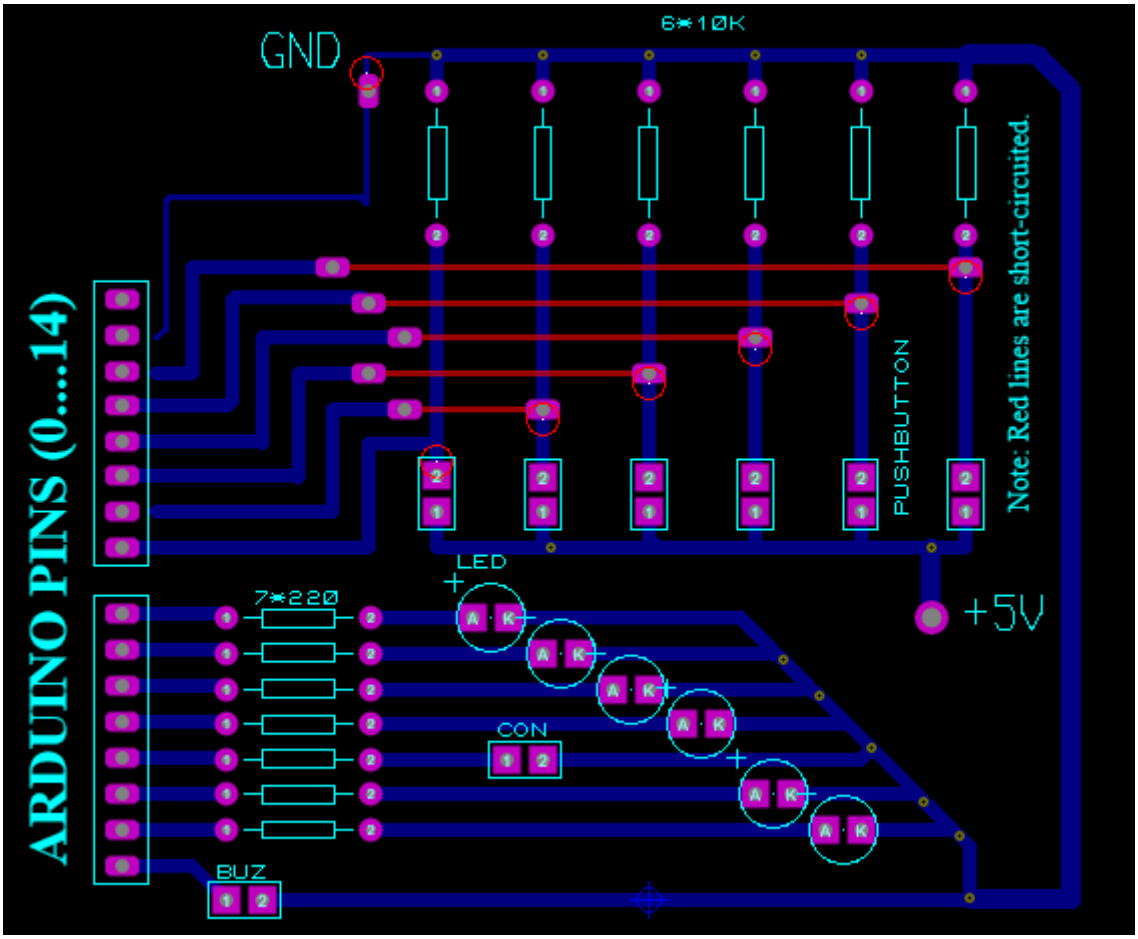


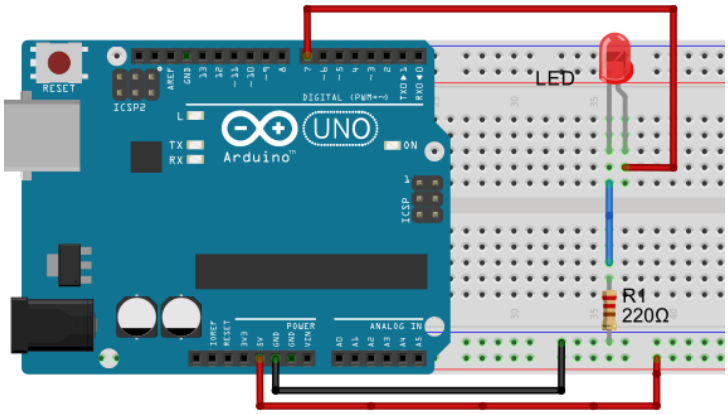
Figura: Schema PCB del Kit Modulo Pulsante e LED

Circuiti e Programmi esemplificativi per Arduino

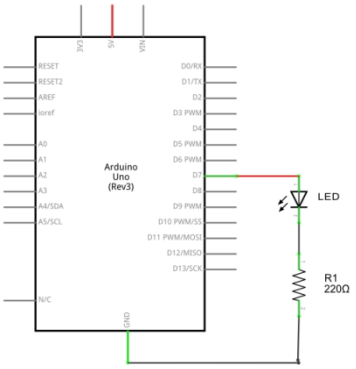
Circuito 1:

Titolo del circuito: Programma LED lampeggiante

Descrizione del circuito: Un LED è collegato al pin 13 di Arduino. Il LED lampeggia continuamente con un intervallo di 1 secondo.



1-) Vista da Breadboard



2-) Vista schematica

```
/*      Programma per il lampeggio del LED, accensione e spegnimento di un LED   */
int led = 7;                    //variabile integer led è dichiarata
void setup() {                  // il metodo setup() viene eseguito solo una volta
pinMode(led, OUTPUT);    // il led PIN è dichiarato come output digitale
}
void loop() {                   // il metodo loop() viene ripetuto
digitalWrite(led, HIGH);    // accensione del led
delay(1000);                    // arresto del programma per 1000 millisecondi
digitalWrite(led, LOW);       // spegnimento del led
delay(1000);                    // arresto del programma per 1000 millisecondi
 }
```

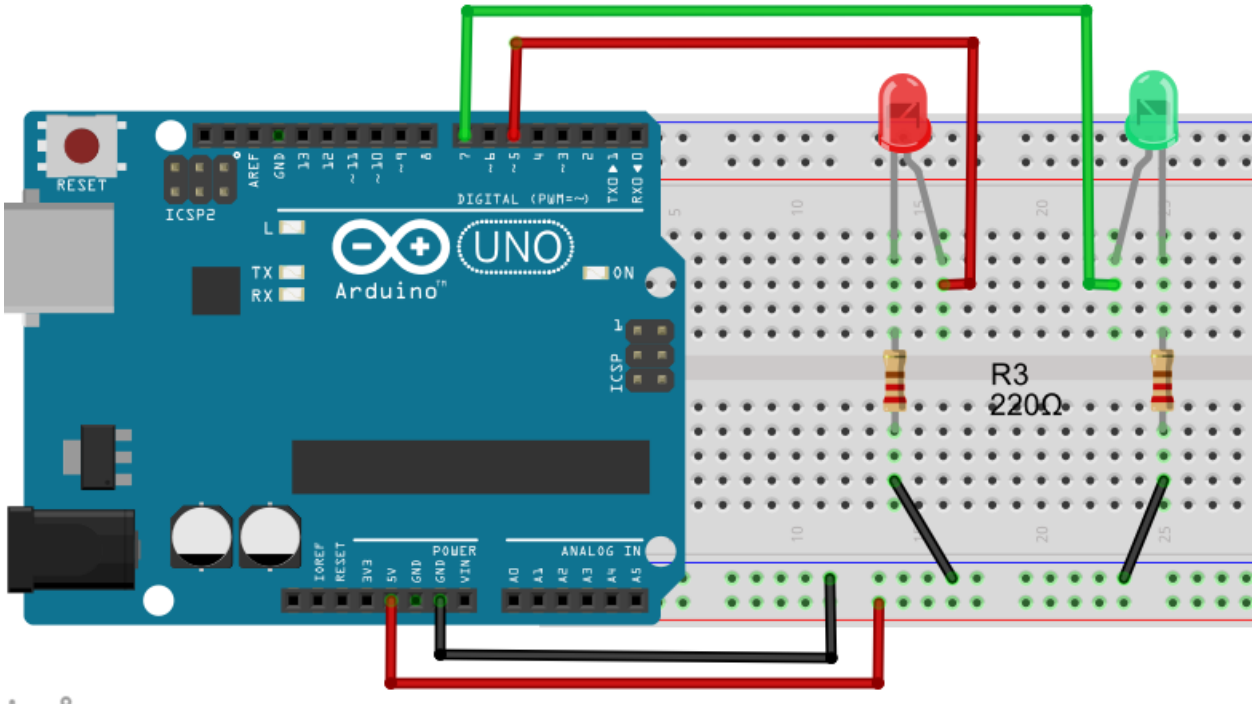
Circuit 2:

Titolo del circuito: Flip-Flop, programma per far lampeggiare 2 LED

Descrizione del circuito:

Nome del circuito: 2 LED lampeggiano in sequenza.

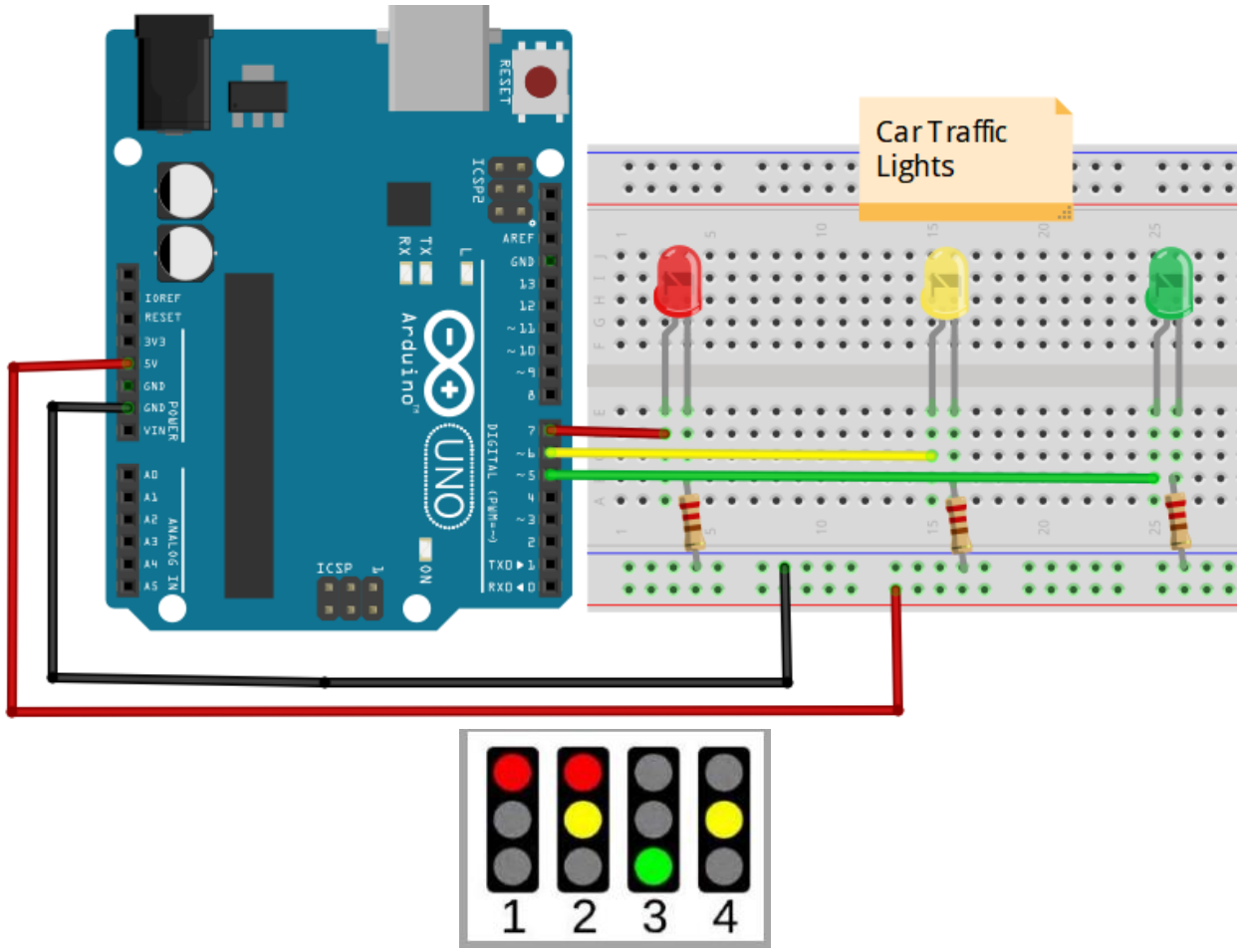
Descrizione del circuito: 2 LED lampeggiano in sequenza con intervalli di 2 secondi.




```
/*    Flip Flop     */

int greenLED=5;                    // // Pin a cui è collegato il LED verde
int redLED=7;                      // // Pin a cui è collegato il LED rosso

void setup()  {
pinMode(greenLED, OUTPUT);         // Il pin del LED verde è inizializzato come OUTPUT
pinMode(redLED, OUTPUT);           //Il pin del LED rosso è inizializzato come OUTPUT
                  }

void loop(){
digitalWrite(greenLED,  HIGH);          // accendi il LED verde
digitalWrite(redLED,  LOW);             // spegni il LED rosso
deley(2000);

digitalWrite(greenLED,  LOW);           // spegni il LED verde
digitalWrite(redLED,  HIGH);            // accendi il LED rosso
deley(2000);
                  }
```

Circuito 3:

Titolo del circuito: Semafori
Descrizione del circuito: In questo progetto realizzeremo un semaforo. Ci sono 3 LED di colori diversi (verde, giallo e rosso)..



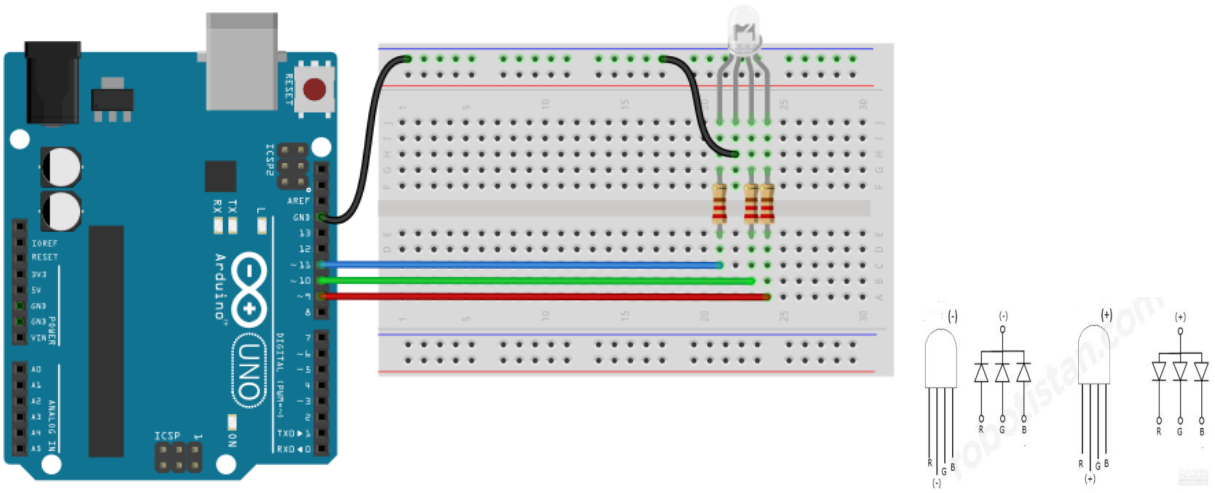

```
/*      Semaforo    */

int redLED = 7;
int yellowLED =6;
int greenLED = 5;

void setup() {                          // Qui stiamo inizializzando i nostri pin come outputs
pinMode(redLED,  OUTPUT);
pinMode(yellowLED,  OUTPUT);
pinMode(greenLED,  OUTPUT);
}

void loop() {
```

```
digitalWrite(redLED,  HIGH);          // Il LED rosso rimane ACCESO per 9 secondi.
digitalWrite(yellowLED,  LOW);
digitalWrite(greenLED,  LOW);
delay(9000);

digitalWrite(redLED,  HIGH);          // Il LED rosso e il LED giallo rimangono ACCESI per 2 secondi.
digitalWrite(yellowLED,  HIGH);
digitalWrite(greenLED,  LOW);
delay(2000);

digitalWrite(redLED,  LOW);           //Il LED verde rimane ACCESO per 9 secondi.
digitalWrite(yellowLED,  LOW);
digitalWrite(greenLED,  HIGH);
delay(9000);

digitalWrite(redLED,  LOW);           // Ancora una volta, il LED giallo è ACCESO per 2 secondi
digitalWrite(yellowLED,  HIGH);
digitalWrite(greenLED,  LOW);
delay(2000);

/* Il ciclo ricomincia   */
 }
```

Circuito 4:

Titolo del circuito: Applicazione LED RGB

Spiegazione del circuito: Il LED RGB è composto da 3 LED, con un terminale in comune, posizionati in un unico involucro. È possibile controllare digitalmente l'intensità luminosa dei tre colori. Inoltre, è possibile ottenere i colori desiderati utilizzando la tecnica PWM (Modulazione di Larghezza d'Impulso).



/* Faremo lampeggiare ciascun colore del LED RGB a intervalli di 1 secondo. Se vogliamo visualizzare la luce bianca, dobbiamo accendere tutti i LED.*/

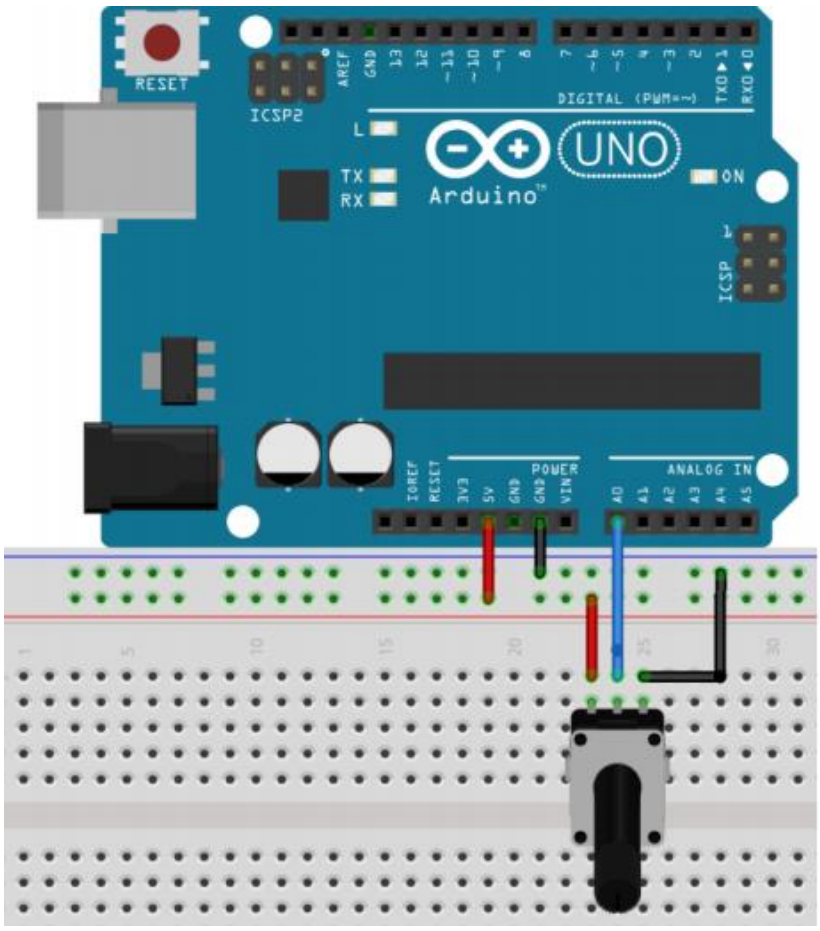
```
const int BlueLed=11;          // Colleghiamo il LED blu al pin 11.                    const
int GreenLed=10;      //  Colleghiamo il LED verde al pin--10                         const
int RedLed=9;          // Colleghiamo il LED rosso al pin-9

        // Impostiamo i pin ai quali sono collegati i LED come output.                 void
setup() {                                                                  pinMode(BlueLed,OUTPUT);
pinMode(GreenLed,OUTPUT);
pinMode(RedLed,OUTPUT);     }

        //  Il ciclo ha inizio qui.

void loop() {                                                                  digitalWrite(BlueLed,
LOW);          // Il LED rosso è ACCESO.                                      digitalWrite(GreenLed,
LOW);                                                                  digitalWrite(RedLed, HIGH);
delay(1000);

 digitalWrite(BlueLed,  LOW );   //   Il   LED   verde   è   ACCESO.
digitalWrite(GreenLed, HIGH);                                          digitalWrite(RedLed,
LOW );                                                                 delay(1000);

 digitalWrite(BlueLed,      HIGH);                                  //  Il  LED  blu  è  ACCESO.
digitalWrite(GreenLed,  LOW);                                          digitalWrite(RedLed,
LOW);                                                                  delay(1000);

        //   Mostriamo   il   colore   bianco   attivando   tutti   i   led.
digitalWrite(BlueLed, HIGH);                                           digitalWrite(GreenLed,
HIGH);                                                                 digitalWrite(RedLed,  HIGH);
delay(1000);                                                                                  }
```

Circuito 5:

Titolo del Circuito: Lettura della tensione analogica, lettura del valore dal potenziometro

Spiegazione del circuito: In questo circuito, impariamo come leggere un ingresso analogico sul pin analogico 0. L'input viene convertito tramite la funzione analogRead() in un valore numerico e stampato sul monitor seriale dell'IDE di Arduino**.**

Potenziometro: Un potenziometro (o pot) è un semplice trasduttore elettro-meccanico. Converte il movimento rotatorio o lineare della manopola (o del cursore) in una variazione di resistenza. Tale variazione viene (o può essere) utilizzata per controllare qualsiasi cosa, dal volume di un sistema hi-fi alla direzione di una grande nave mercantile.	



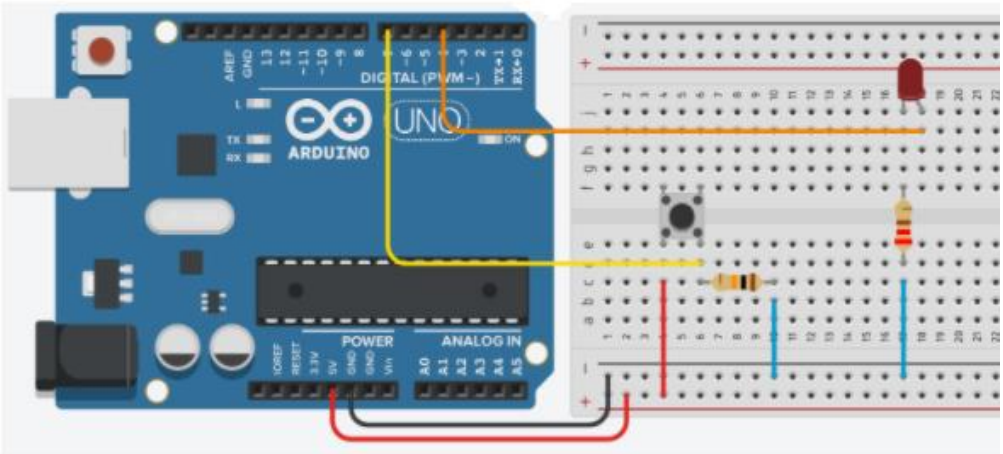
/* ReadAnalogVoltage : Legge un segnale analogico sul pin A0, lo converte in numero e stampa il risultato sul monitor seriale. La rappresentazione grafica è disponibile utilizzando il serial plotter(menu Tool > menu Serial Plotter). Collega il pin centrale di un potenziometro al pin A0, e i pin esterni a +5V e a massa. */

```
// La routine di setup viene eseguita una sola volta quando si preme il pulsante di reset:
void setup()            {
// Inizializza la comunicazione seriale a 9600 bit al secondo:
  Serial.begin(9600); }
// La routine loop viene eseguita continuamente, in modo ripetuto, all'infinito:
void loop() {
// leggi l'input sul pin analogico 0:
  int sensorValue = analogRead(A0);
// stampa il valore che hai letto:
  Serial.println(sensorValue);
  delay(1000);          // ritardo tra le letture per stabilità
}
```

Circuito 6:

Titolo del circuito: Utilizzo dei pulsanti su Arduino

Descrizione del circuito: Premendo un pulsante collegato al pin 7, il pulsante accende e spegne un LED collegato al pin digitale 4.



/* Il pulsante, connesso al pin 7, consente di accendere e spegnere il LED (diodo a emissione di luce) collegato al pin digitale 4*/

```
void setup()
{
  pinMode(4, OUTPUT);  // pin-4 rappresenta l'output
  pinMode(7, INPUT);     //  pin7 rappresenta l'input.
}
void loop()
{
   if (digitalRead(7) == HIGH)                 //  Se il pin7 è alto,
    digitalWrite(4, HIGH);                     //  il Led è ACCESO,
    if (digitalRead(7) == LOW)                 // Se il pin7 è basso,
     digitalWrite(4, LOW);                    // il Led è SPENTO.
}
```

NOTA: L'istruzione **IF-THEN** può essere utilizzata anche per gestire i pulsanti collegati ai pin di ingresso (INPUT) degli Arduino. Questi collegamenti possono essere realizzati in due modi diversi: collegamento **Pull-Up** e collegamento **Pull-Down**..

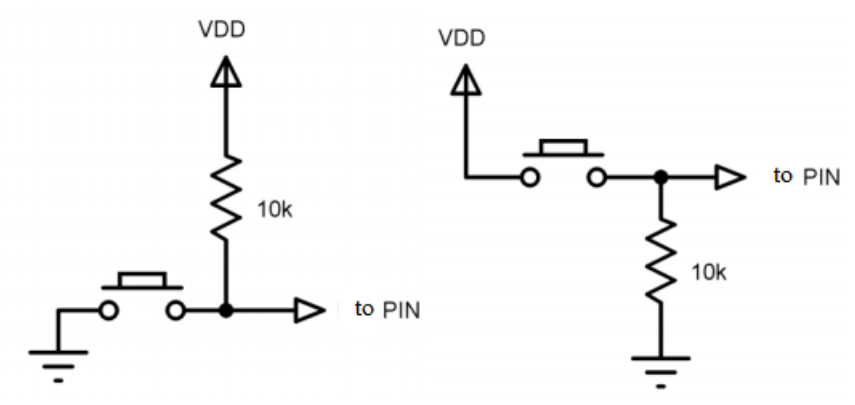
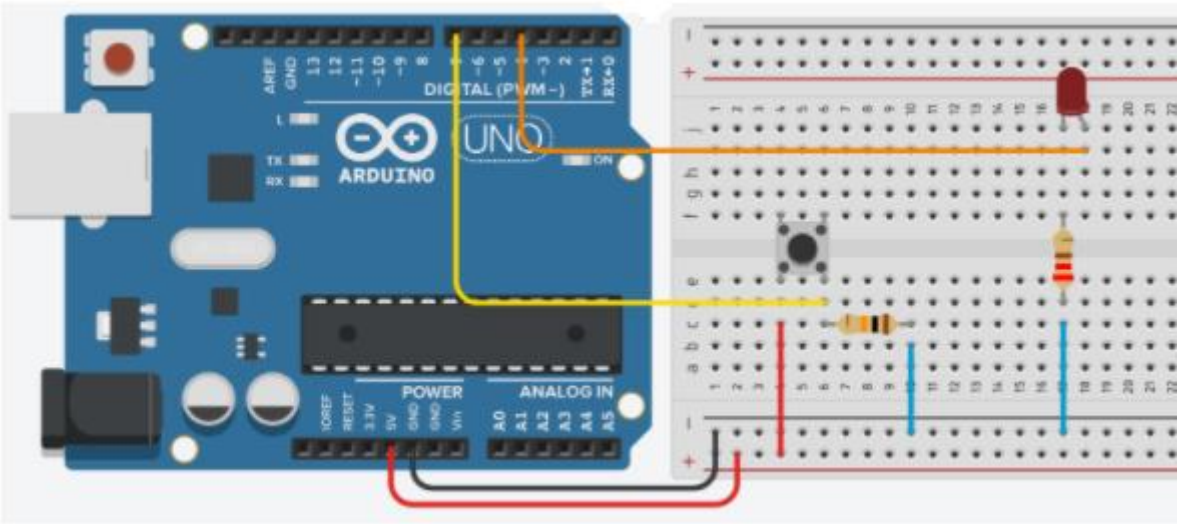


Figura: Schema di collegamento di interruttori o pulsanti utilizzati con l'istruzione IF…THEN

Circuito 7:

Titolo del circuito: Utilizzo dell'istruzione ELSE su Arduino

Spiegazione del circuito: Premendo un pulsante collegato al pin 7, il pulsante accende e spegne un LED collegato al pin digitale 4. Inoltre, impareremo a definire i pin utilizzando il tipo di dato "int".

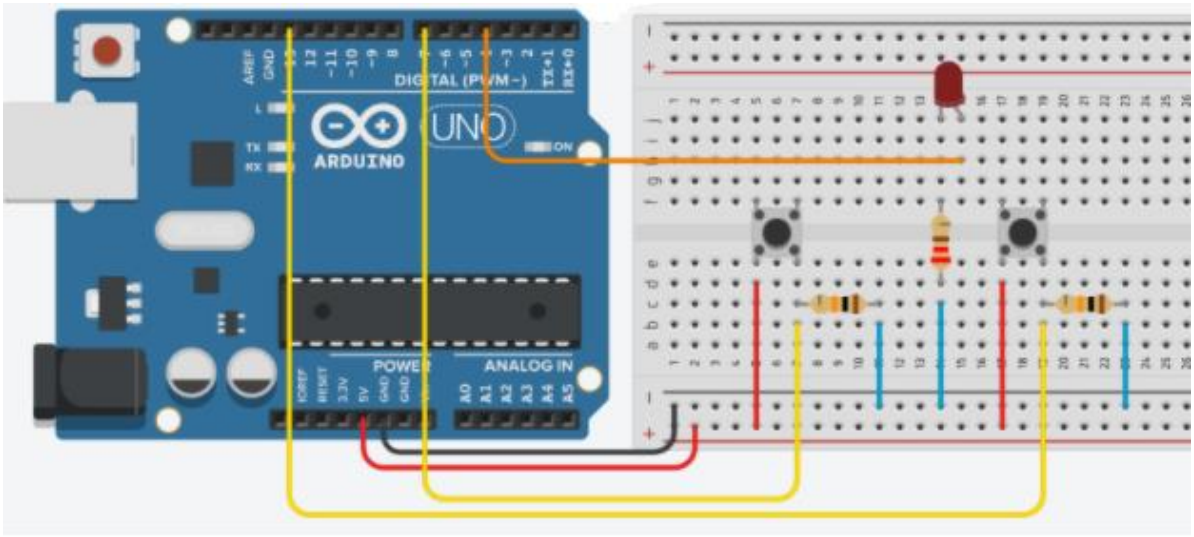



```
 int led = 4;
int buton =7;
void setup()
{
  pinMode(led, OUTPUT);
  pinMode(buton, INPUT);
}
void loop()
{
        if (digitalRead(buton) == HIGH)        // Il pulsante è ACCESO,
        digitalWrite(led, HIGH);                // Il Led è ACCESO
            else                                        // Se no
          digitalWrite(led, LOW);                // İl Led è SPENTO.

}
```

Circuito 8:

Titolo del circuito: 2 pulsanti contro 1 LED

Spiegazione del circuito: Un pulsante accende il LED, un altro pulsante spegne il LED.




```
*          2 pulsanti contro 1 LED
           I pulsanti sono collegati con resistori da 10K verso massa.          */
int led = 4;                    //  Il pin-4 è assegnato come "led"
int button1 = 7;                // Il  pin-7 è assegnato come   "pulsante1"
int button2 = 13;               // Il pin-13  è assegnato come "pulsante2"
void setup()
{
  pinMode(led, OUTPUT);                 // led=Output
  pinMode(button1, INPUT);              //  pulsante1=Input
  pinMode(button2, INPUT);              // pulsante2=Input
}
void loop()
{
  if (digitalRead(button1) == HIGH)     // SE  il "pulsante1" è ACCESO (attivo),
        digitalWrite(led, HIGH);                  // Il Led è ACCESO.
```

```
if (digitalRead(button2) == HIGH)        // SE  il "pulsante2"  è ACCESO (attivo),
  digitalWrite(led, LOW);                // Il Led è SPENTO.  }
```

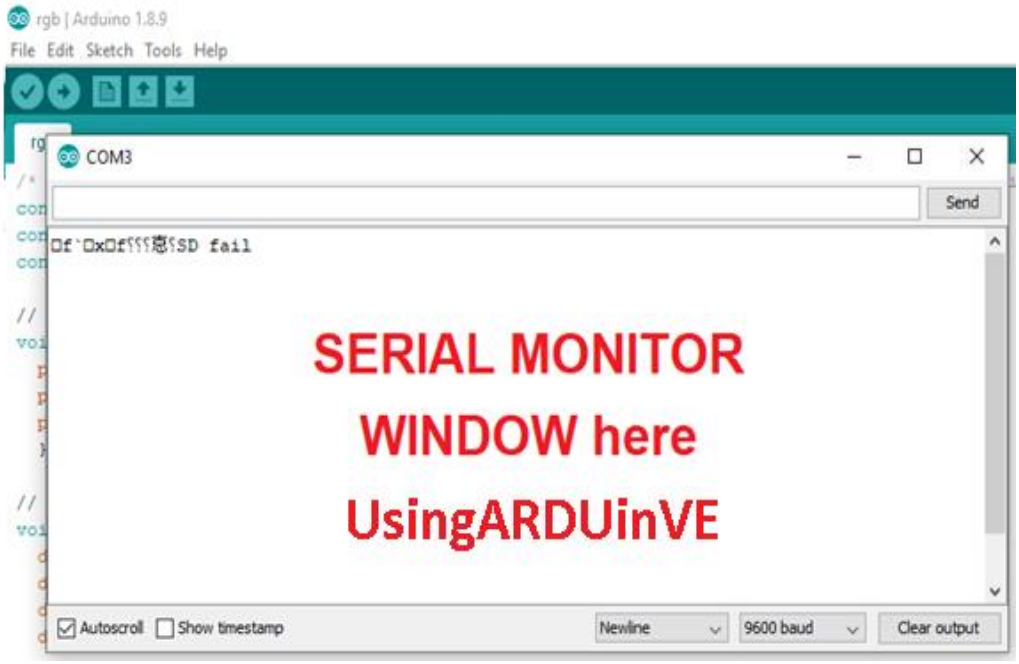
COME UTILIZZARE IL MONITOR SERIALE DI ARDUINO

Il Monitor Seriale è una finestra separata nell'IDE di Arduino che funge da terminale indipendente per inviare e ricevere dati seriali. È possibile accedervi cliccando sull'icona, illustrata nell'immagine seguente, posizionata all'estrema destra della barra degli strumenti.

I dati seriali vengono trasmessi su un singolo conduttore (anche se, nel nostro caso, generalmente tramite connessione USB) e consistono in una sequenza di bit (1 e 0) inviati in serie. La comunicazione seriale può avvenire in entrambe le direzioni, utilizzando due linee separate per la trasmissione e la ricezione dei dati.

Il Monitor Seriale viene utilizzato per eseguire il debug degli sketch di Arduino o per visualizzare i dati inviati da uno sketch in esecuzione. Per attivare il Monitor Seriale, è necessario che la scheda Arduino sia collegata al computer tramite USB.

Per aprire il Monitor Seriale, clicca sull'icona 'Monitor Seriale' nell'IDE. Si aprirà una finestra, come mostrato nello screenshot qui sotto. Verifica che il baud rate (velocità di comunicazione) sia impostato su 9600, nell'angolo in basso a destra. È importante che la velocità di comunicazione nel programma corrisponda a quella nel Monitor Seriale. Poiché il valore predefinito è 9600, mpostiamo questa stessa velocità anche nel nostro programma.



Istruzioni principali per utilizzare il Monitor Seriale:

Serial.begin(); Imposta la velocità di trasmissione dei dati in bit per secondo (baudrate) per la comunicazione seriale. Per comunicare con il Monitor Seriale, assicurati di utilizzare una delle velocità di baud presenti nel menu nell'angolo in basso a destra della finestra. Tuttavia, è possibile specificare altre velocità, ad esempio, per comunicare tramite i pin 0 e 1 con un componente che richiede una velocità di baud particolare.

Sintassi: Serial.begin(velocità);
Esempio: Serial.begin(9600);

Serial.print(); invia i dati alla porta seriale come una sequenza di caratteri ASCII. I numeri interi sono rappresentati come una sequenza di caratteri ASCII per ogni cifra. I numeri in virgola mobile (float) vengono convertiti in caratteri ASCII, con un formato predefinito che include due cifre decimali. I byte vengono trasmessi come singoli caratteri. Caratteri e stringhe vengono inviati senza alcuna modifica.

Per esempio:

Serial.print (78); stampa "78"

Serial.print('N'); stampa "N"

Serial.print("Hello Arduino"); stampa "Hello Arduino"

Serial.println(); invia i dati sulla porta seriale come una sequenza di caratteri ASCII, aggiungendo alla fine un carattere di ritorno a capo (ASCII 13, '\r') e un carattere di nuova linea (ASCII 10, '\n'). Questo comando supporta gli stessi tipi di dati e formati di **Serial.print()**

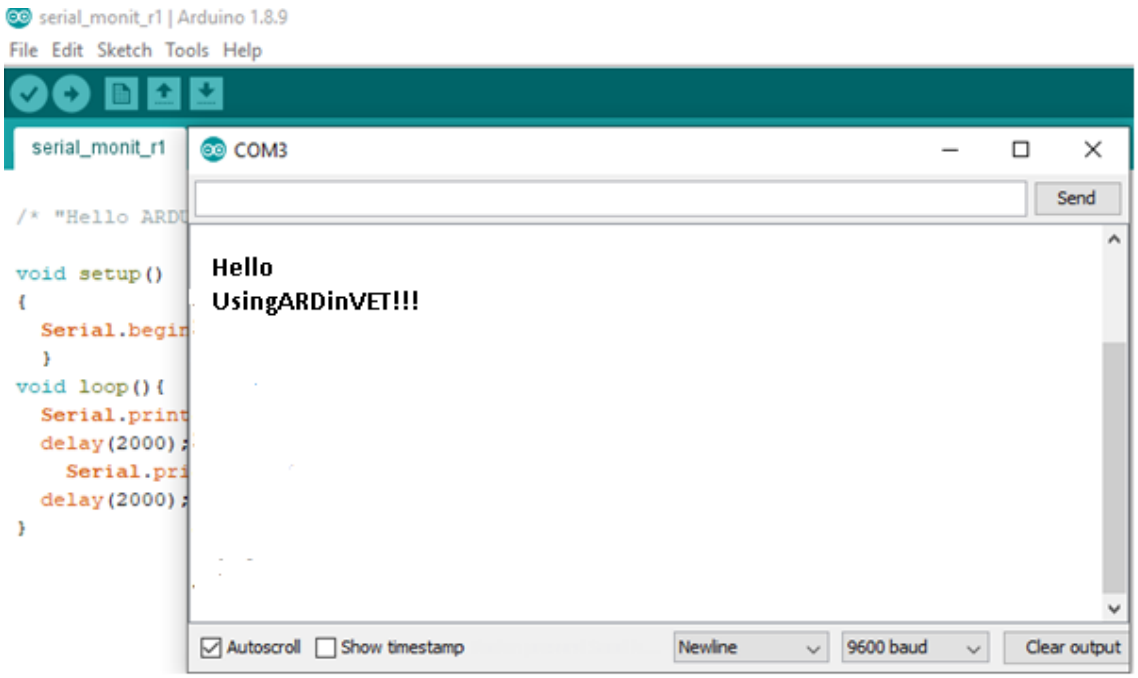
Serial.println(val);
Serial.println(val, format);
Serial.printle(13, BIN); stampa "1101", valore binario di 15 su Monitor seriale**.**

NOTA: L'unica differenza tra **Serial.print** e **Serial.println** è che **Serial.println** aggiunge un ritorno a capo e una nuova linea alla fine del messaggio, quindi il prossimo dato inviato inizierà su una nuova riga. Un'altra novità che potresti aver notato sono le virgolette (" "). Queste vengono utilizzate per indicare una **stringa**.

Circuito 9:
Titolo del circuito: "Applicazione 'Hello UsingARDinVET' nel Monitor Seriale"

Spiegazione del circuito: La scritta '**Hello UsingARDinVET'** sarà visibile nel Monitor Seriale.

```
/*        "Applicazione "Hello  UsingARDinVET" su Monitor Seriale  */
void setup()
{
Serial.begin(9600);     // Velocità della comunicazione seriale
}
void loop()
{
Serial.println("  HELLO ");
delay(2000);
Serial.println(" UsingARDinVET!!!");
delay(2000);
}
```

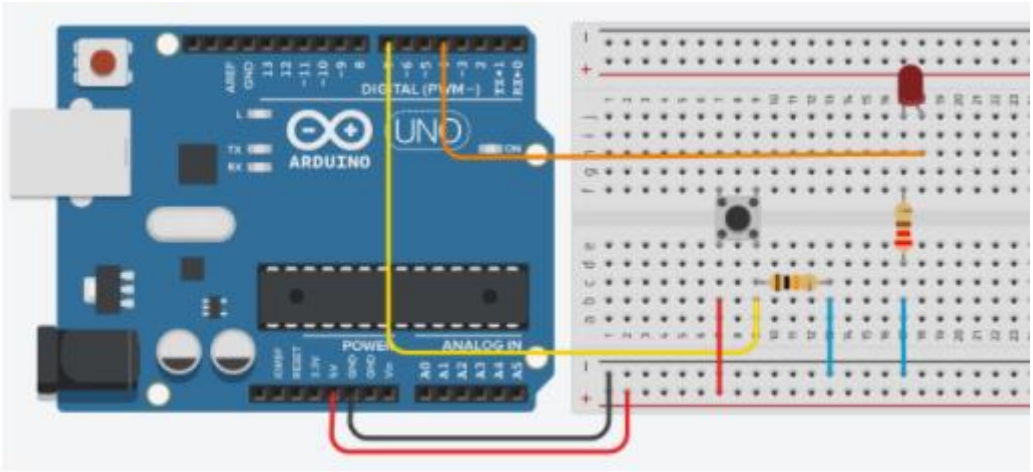


Circuito 10: Analisi dello Stato del Pulsante sul Monitor Seriale

Spiegazione del Circuito:

Se il pulsante è premuto, il messaggio **"LED acceso, LED=ON"** appare sul monitor seriale e il LED si accende.

Se il pulsante non è premuto, il messaggio **"Pulsante non premuto"** appare sul monitor seriale e il LED si spegne.




```
/*        Analisi dello Stato del Pulsante sul Monitor Seriale   */
void setup()                    {
pinMode(4, OUTPUT);
pinMode(7, INPUT);
Serial.begin(9600);          }
void loop()                          {
if (digitalRead(7) == HIGH)            // Legge il segnale digitale sul Pin 7
{
digitalWrite(4, HIGH);
Serial.println("Il Led è acceso, LED=ON");
delay(250);
}
else                                 // Se la condizione precedente non è soddisfatta
{
digitalWrite(4, LOW);
Serial.println("Il Pulsante non è premuto");
```

delay(1000);

}

} //L'output del monitor seriale è mostrato di seguito.

```
Button is not pressed
Button is not pressed
Led is ligthing, LED=ON
Led is ligthing, LED=ON
Led is ligthing, LED=ON
Led is ligthing, LED=ON
Led is ligthing, LED=ON
Button is not pressed
```

Circuito 11:

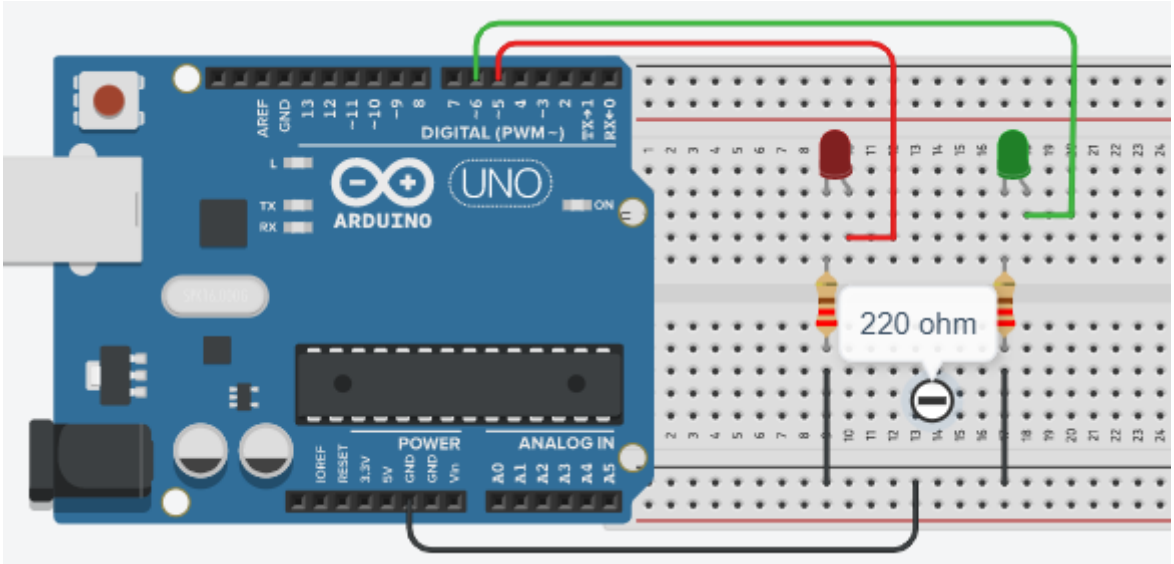
Titolo del circuito: "Invio di dati dal monitor seriale."

Spiegazione del circuito:

Se viene premuto il carattere "A" sulla tastiera, il LED1 si accende.

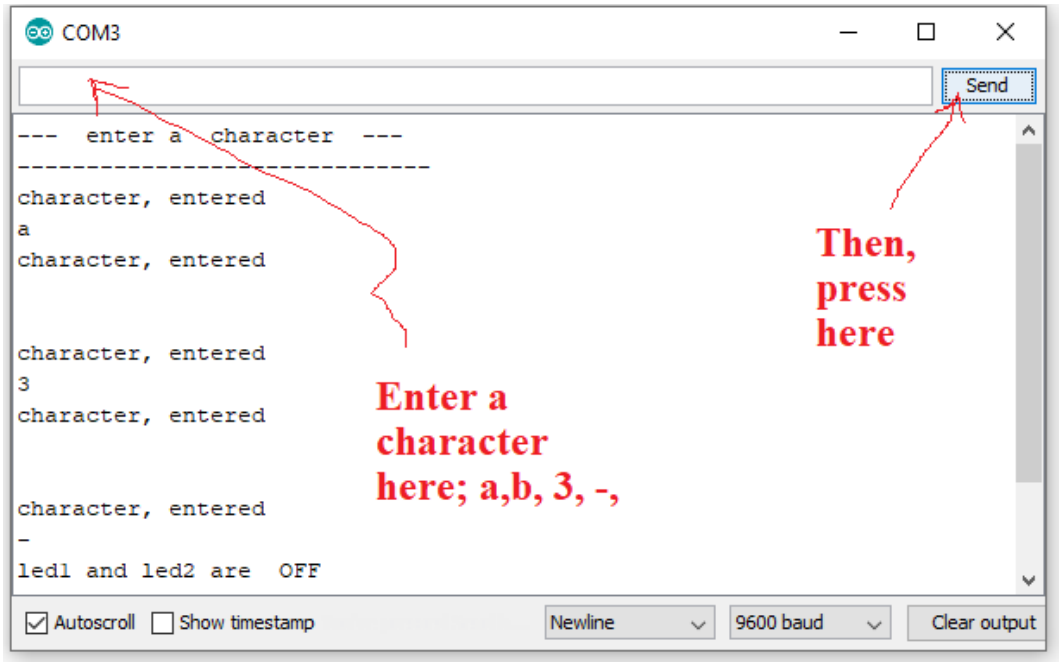
Se viene premuto il carattere "3" sulla tastiera, il LED2 si accende.

Se viene premuto il carattere "-" sulla tastiera, sia il LED1 che il LED2 si spengono.



```
/*    Invio di dati dal monitor seriale   */

char character;
#define  led1 5
#define  led2 6

void setup()   {
pinMode(led1, OUTPUT);
pinMode(led2, OUTPUT);
Serial.begin(9600);
Serial.println ("---  enter a  character  --- ");
Serial.println ("------------------------------");
}

void loop()
{
if (Serial.available()>0)
{
character=Serial.read();
Serial.println ("character, entered ");
Serial.println (character);

if (character=='A') digitalWrite (led1, 1);

if (character=='a') digitalWrite (led1, 1);

if (character=='3') digitalWrite (led2, 1);

if (character=='-')
{
digitalWrite(led1,0);
digitalWrite(led2,0);
Serial.println ("led1 and led2 are  OFF ");
}
}
}
```

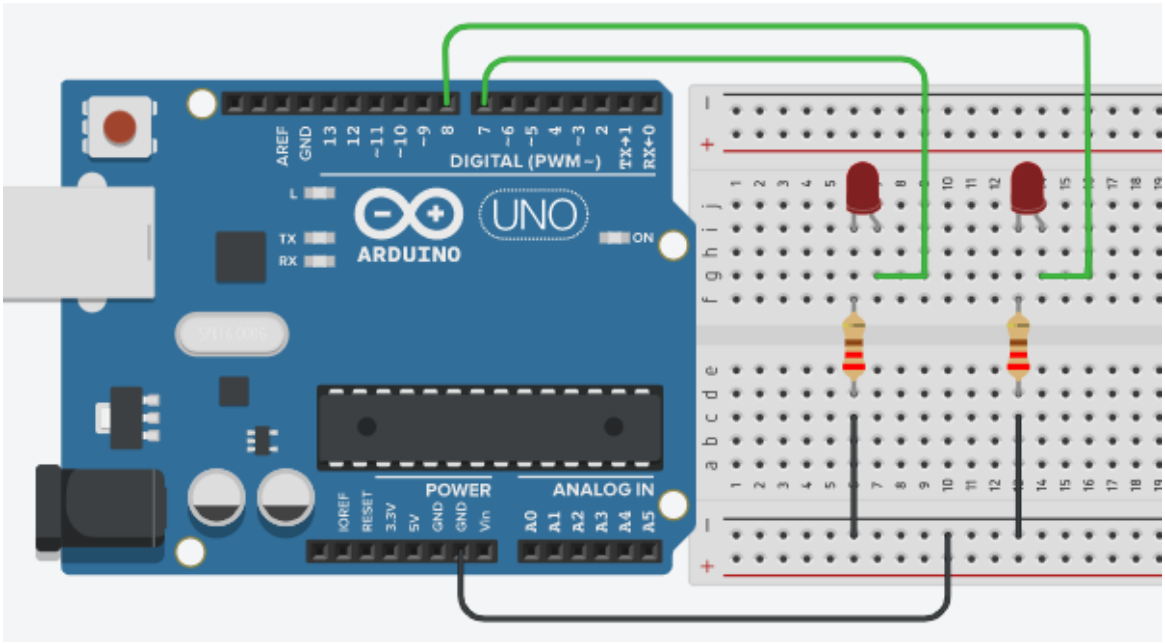
“



Circuito 12:

Titolo del circuito: "Apprendere l'istruzione FOR"

Spiegazione del circuito:



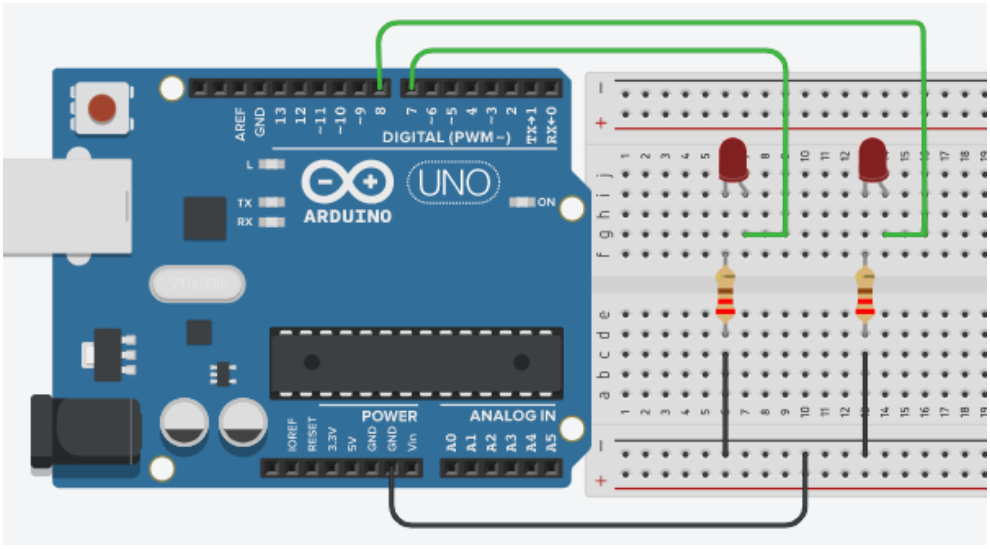
```
/*     "Apprendere l’istruzione FOR”   */

int   led1 = 7;
int   led2 = 8;

void setup() {
Serial.begin(9600);
pinMode(7,OUTPUT);
pinMode(8,OUTPUT);
}

void loop()
{
for(int i=1; i<6; i++)
{
Serial.println(i);
digitalWrite(led1, 1);
digitalWrite(led2, 1);
delay(1000);

digitalWrite(led1, 0);
digitalWrite(led2, 0);
delay(1000);
}
}
```

Circuito 13:

Titolo del circuito: "Istruzione FOR versus Monitor Seriale"

Spiegazione del circuito: In questa applicazione, i LED lampeggeranno sei volte, mentre i numeri 1, 2, 3, 4, 5 e 6 verranno visualizzati sul monitor seriale. Successivamente, il programma entrerà in un ciclo while e uscirà dal ciclo for. A questo punto, il programma si fermerà. Per avviare nuovamente l'esecuzione, sarà necessario premere il pulsante di reset.

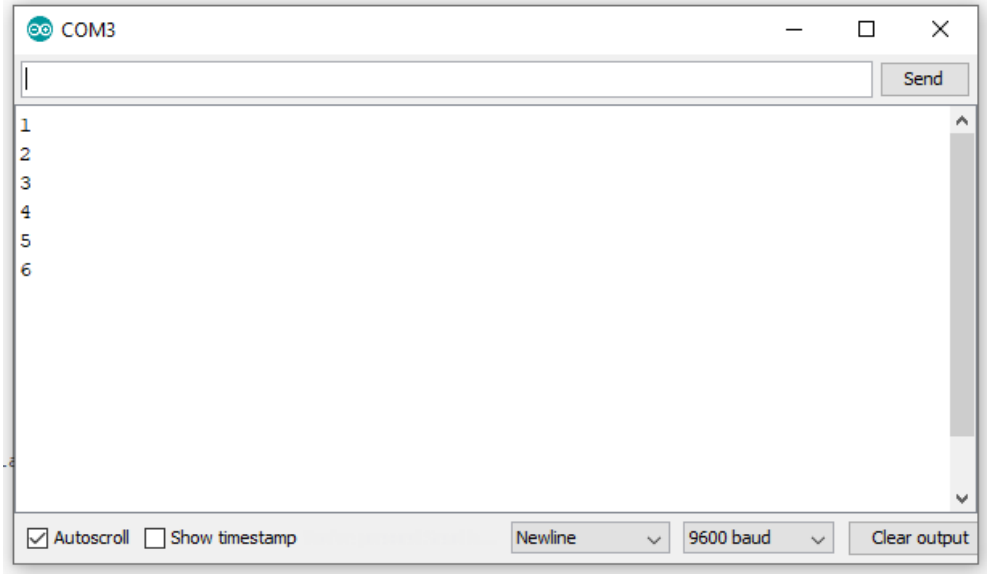
Qui utilizziamo lo stesso circuito del precedente.




```
/*    "Istruzione FOR versus Monitor Seriale"   */

int   led1 = 7;
int   led2 = 8;

void setup()                {
Serial.begin(9600);
pinMode(7,OUTPUT);
pinMode(8,OUTPUT);      }

void loop() {

for(int i=1; i<=6; i++)                  // il valore di i = 1, 2,3, 4,5, 6

{

Serial.println(i);                       // Scrive i valori di i sul monitor seriale
```

```
digitalWrite(led1, 1);
digitalWrite(led2, 1);
delay(1000);
digitalWrite(led1, 0);
digitalWrite(led2, 0);
delay(1000);
}
while(1);                    //  il ciclo for non genera un ciclo infinito.

}
```

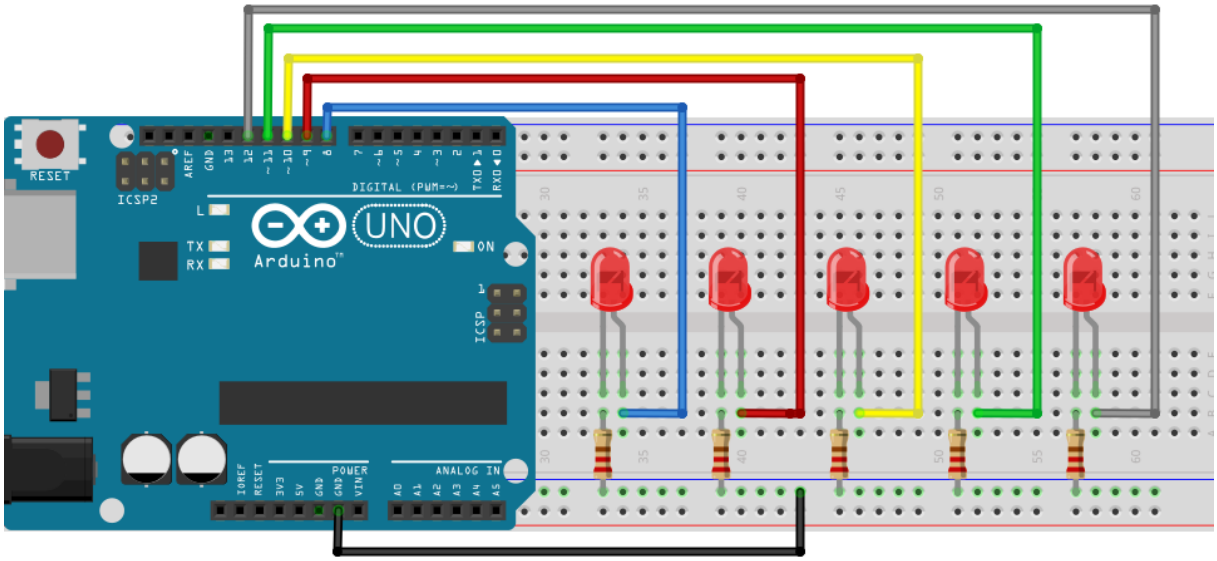
NOTA: Per riavviare, si deve premere il pulsante reset.



Circuito 14:

Titolo del circuito: "Knight Rider con 5 LED utilizzando l'istruzione FOR"

Spiegazione del circuito:




```
/*     Knight Rider con 5 LED utilizzando l'istruzione FOR  */

void setup() {
pinMode(8, OUTPUT);
pinMode(9, OUTPUT);
pinMode(10, OUTPUT);
pinMode(11, OUTPUT);
pinMode(12, OUTPUT);     }

void loop() {

for (int b=8; b<=12; b++)              // Qui parte l'incremento del contatore

{

digitalWrite(b, HIGH);
delay(150);
digitalWrite (b, LOW);
delay(150);

}

for (int b=12; b>=8; b--)                    // Qui avviene il decremento del contatore

{
```

```
digitalWrite (b, HIGH);
delay(150);
digitalWrite (b, LOW);
delay(150);

}
}
```

Circuito 15:

Titolo del circuito: " Generazione di colori casuali con un LED RGB, utilizzando l'istruzione switch/case"

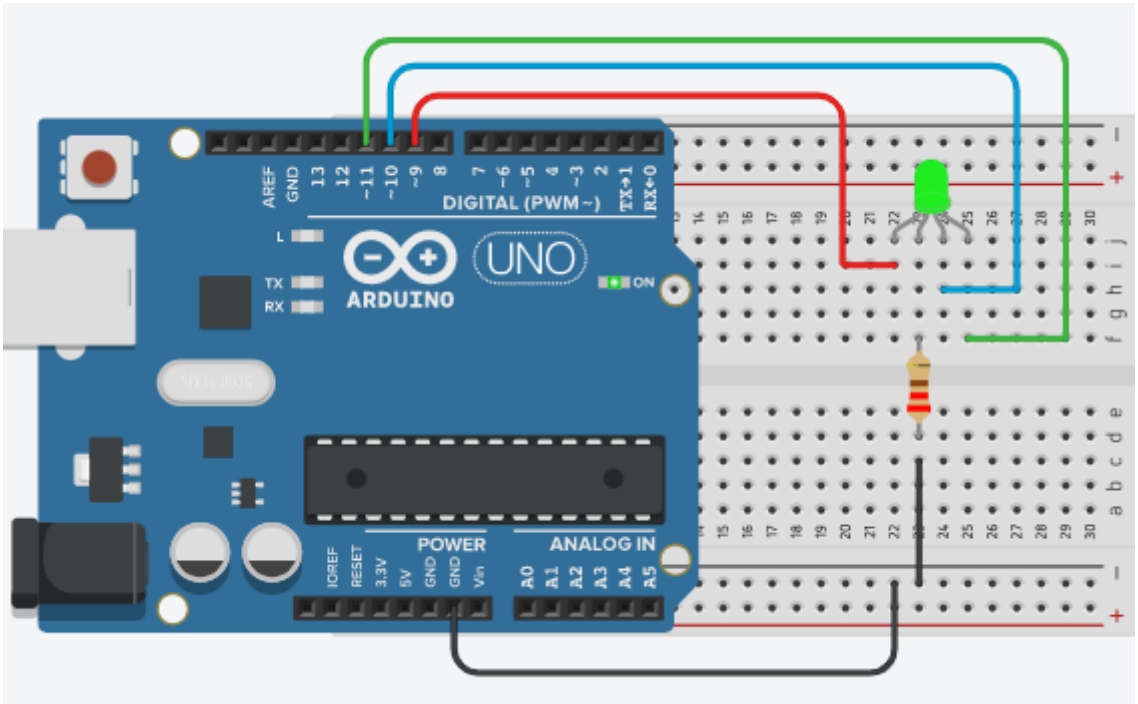
Spiegazione del circuito: In questo circuito vengono utilizzate l'istruzione Random e l'istruzione Switch/Case. In questo circuito, i colori rosso, verde e blu si otterranno in modo casuale.

NOTA:

random() : La funzione random genera numeri casuali.
Sintassi: random(max), random(min, max)

min: limite inferiore del valore casuale (opzionale).

max: limite superiore del valore casuale.



/* Generazione di colori casuali con un LED RGB, utilizzando l'istruzione switch/case */

```
#define R  9
#define G 10
#define B 11
int colour;
int dly=3000;

void setup() {
pinMode(R, OUTPUT);
pinMode(G, OUTPUT);
pinMode(B, OUTPUT);
Serial.begin(9600);
}

void loop()
{
colour=random(4);
Serial.println(colour);

switch(colour)  {

case 0:
digitalWrite (R, 1);              //LED rosso = ACCESO
digitalWrite (G, 0);
digitalWrite (B, 0);
delay(dly);
break;

case 1:
digitalWrite (R, 0);            //LED verde = ACCESO
digitalWrite (G, 1);
digitalWrite (B, 0);
delay(dly);
break;

case 2:                          //LED blu = ACCESO
digitalWrite (R, 0);
digitalWrite (G, 0);
digitalWrite (B, 1);
delay(dly);
break;

case 3:                           //Nessun colore
digitalWrite (R, 0);
digitalWrite (G, 0);
digitalWrite (B, 0);
delay(dly);
```

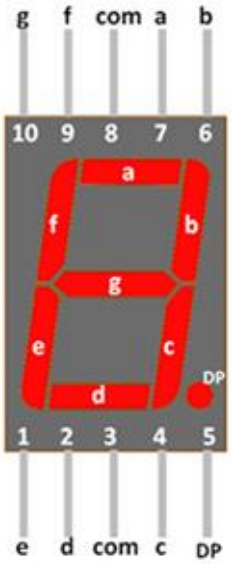
```
break;
}
}
```

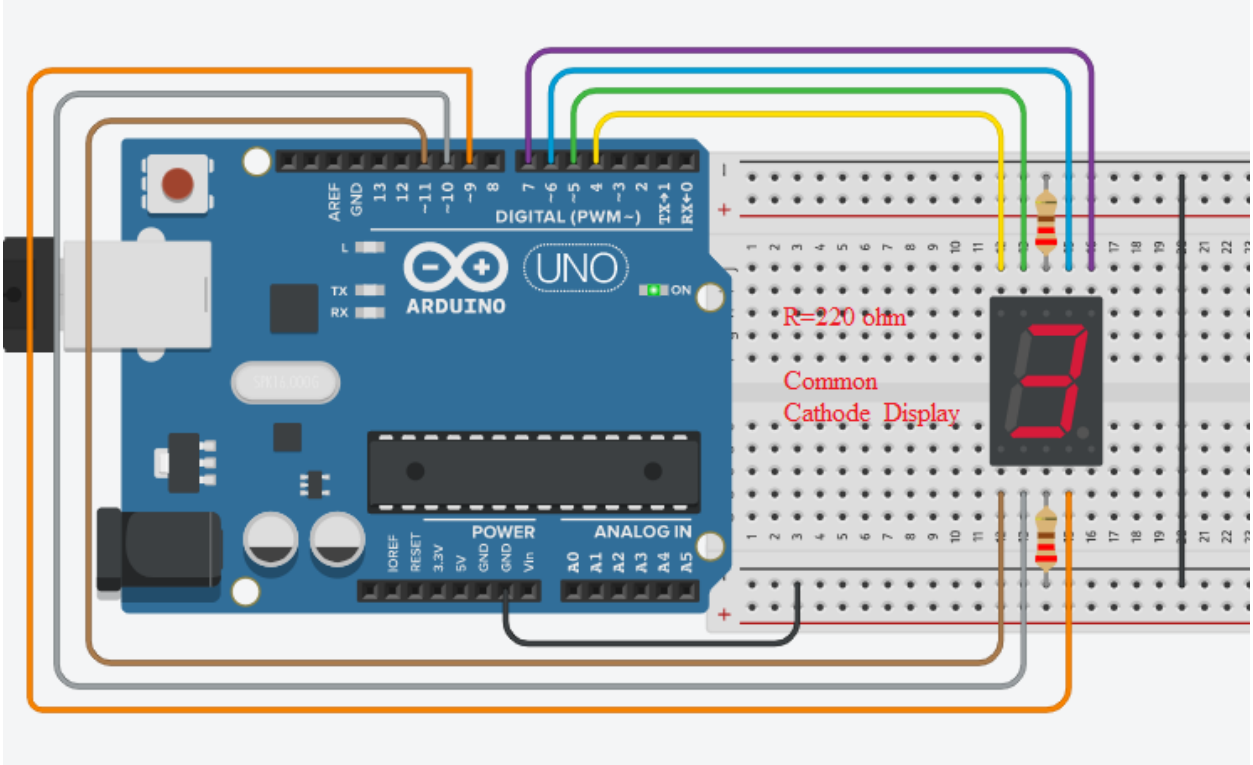
Circuito 16:

Titolo del circuito: "Contatore crescente 0-9 con display a 7 segmenti a catodo comune"

Spiegazione del circuito: Per visualizzare un numero sul display, vengono accesi i segmenti (LED) necessari tra i sette disponibili, contrassegnati dalle lettere a, b, c, d, e, f, g.

NOTA: Un display a sette segmenti è un dispositivo elettronico utilizzato per visualizzare numeri decimali.






```
/*    " Contatore crescente 0-9 con display a 7 segmenti a catodo comune. "    */

int       a=6,   b=7,   c=9,    d=10,     e=11,   f=5,    g=4;
int  number;

void setup() {
pinMode(a,  OUTPUT);
pinMode(b,  OUTPUT);
pinMode(c,  OUTPUT);
pinMode(d,  OUTPUT);
pinMode(e,  OUTPUT);
pinMode(f,  OUTPUT);
pinMode(g,  OUTPUT);
}

void loop()  {
for(number=0;  number< =9;  number++)    {
delay(1000);
switch(number)  {

 case 0:
digitalWrite (a, HIGH);
digitalWrite (b, HIGH);
digitalWrite (c, HIGH);
digitalWrite (d, HIGH);
digitalWrite (e, HIGH);
digitalWrite (f, HIGH);
digitalWrite (g, LOW);
break;

case 1:
digitalWrite (a, LOW);
```

```
digitalWrite (b, HIGH);
digitalWrite (c, HIGH);
digitalWrite (d, LOW);
digitalWrite (e, LOW);
digitalWrite (f, LOW);
digitalWrite (g, LOW);
break;

case 2:
digitalWrite (a, HIGH);
digitalWrite (b, HIGH);
digitalWrite (c, LOW);
digitalWrite (d, HIGH);
digitalWrite (e, HIGH);
digitalWrite (f, LOW);
digitalWrite (g, HIGH);
break;

case 3:
digitalWrite (a, HIGH);
digitalWrite (b, HIGH);
digitalWrite (c, HIGH);
digitalWrite (d, HIGH);
digitalWrite (e, LOW);
digitalWrite (f, LOW);
digitalWrite (g, HIGH);
break;

case 4:
digitalWrite (a,LOW );
digitalWrite (b, HIGH);
digitalWrite (c, HIGH);
digitalWrite (d, LOW);
digitalWrite (e, LOW);
digitalWrite (f, HIGH);
digitalWrite (g, HIGH);
break;

case 5:
digitalWrite (a, HIGH);
digitalWrite (b, LOW);
digitalWrite (c, HIGH);
digitalWrite (d, HIGH);
digitalWrite (e, LOW);
digitalWrite (f, HIGH);
digitalWrite (g, HIGH);
break;

case 6:
digitalWrite (a, HIGH);
digitalWrite (b, LOW);
digitalWrite (c, HIGH);
digitalWrite (d, HIGH);
digitalWrite (e, HIGH);
digitalWrite (f, HIGH);
digitalWrite (g, HIGH);
break;

case 7:
digitalWrite (a, HIGH);
digitalWrite (b, HIGH);
digitalWrite (c, HIGH);
digitalWrite (d, LOW);
digitalWrite (e, LOW);
digitalWrite (f, LOW);
digitalWrite (g, LOW);
break;

case 8:
digitalWrite (a, HIGH);
digitalWrite (b, HIGH);
digitalWrite (c, HIGH);
digitalWrite (d, HIGH);
digitalWrite (e, HIGH);
digitalWrite (f, HIGH);
digitalWrite (g, HIGH);
break;

case 9:
digitalWrite (a, HIGH);
digitalWrite (b, HIGH);
digitalWrite (c, HIGH);
digitalWrite (d, HIGH);
digitalWrite (e, LOW);
digitalWrite (f, HIGH);
digitalWrite (g, HIGH);
break;
}
}
}
```

Circuito 17:

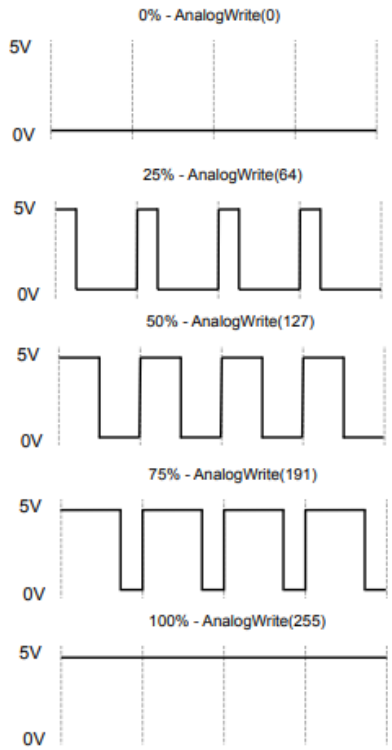
Titolo del circuito: "Utilizzo della Tecnica PWM"

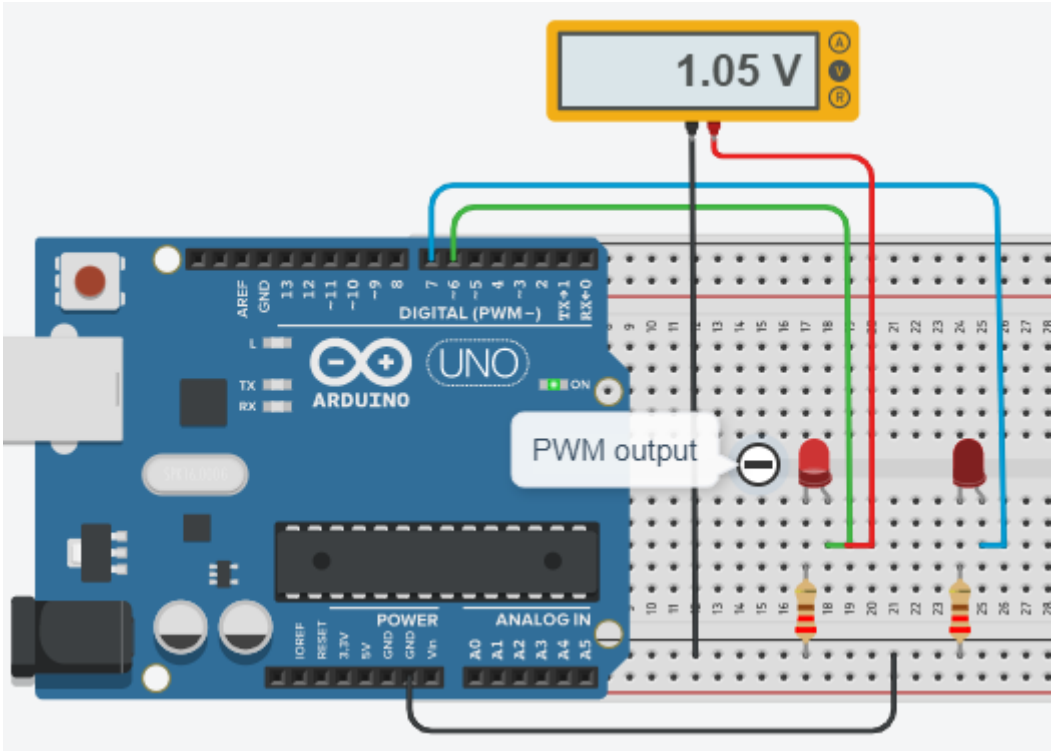
Spiegazione del circuito:

In pratica, vengono utilizzati il pin 6 come uscita PWM e il pin 7 come uscita digitale. L'obiettivo è osservare la differenza tra i due. Con tecnica PWM è possibile realizzare applicazioni come la regolazione della luminosità di un LED o il controllo della velocità di un motore.

NOTA: Arduino supporta la modulazione di larghezza di impulso (PWM) su specifici pin contrassegnati da una tilde (~) sulla scheda (pin 3, 4, 5, 9, 10 e 11), con una frequenza del segnale di 500 Hz (500 impulsi al secondo). Il rapporto tra il tempo in cui il segnale è "ON" e il tempo totale del ciclo è chiamato duty cycle (ciclo di lavoro). Ad esempio, un'uscita PWM che rimane attiva per metà del tempo ciclo ha un duty cycle del 50%. Su Arduino, è possibile assegnare un valore compreso tra 0 e 255 per impostare il duty cycle dell'uscita PWM. Il valore 0 corrisponde a un duty cycle del 0%, con l'uscita sempre a 0V, mentre il valore 255 rappresenta un duty cycle del 100%, con l'uscita costantemente a 5V. Per configurarlo, si utilizza la funzione **analogWrite(),** assegnando il valore desiderato

Il segnale PWM può essere considerato attivo per una frazione x/255, dove x è il valore passato alla funzione **analogWrite().** Di seguito è riportato un esempio che mostra come varia la forma degli impulsi in funzione del valore di x:



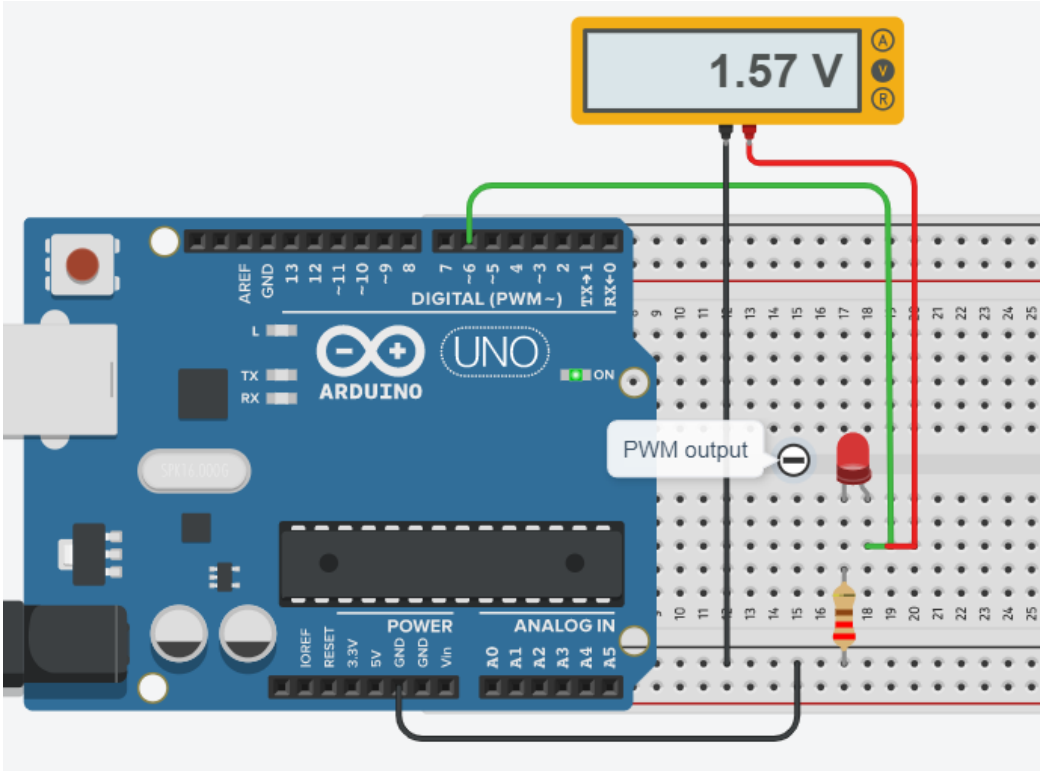



```
/* Apprendere la tecnica PWM utilizzando la funzione analogWrite()   */

#define    led1  6
#define    led2  7

void setup()   {
pinMode(led1, OUTPUT);
pinMode(led2, OUTPUT);       }

void loop()                 {

analogWrite(led1, 60);          //  Il valore di 60 viene inviato al pin Led1, cioè 1,05 V.

analogWrite(led2,60);          //  Il valore di 60 viene inviato al pin Led2, cioè 1,05 V.

delay (100);                        // Soltanto il led1 si accende.
}
```

Circuito 18:

Titolo del circuito: "Modificare la luminosità di un LED dal minimo al massimo."

Spiegazione del circuito: Utilizzando la tecnica PWM, è possibile variare la luminosità di un LED da un valore minimo a uno massimo. In questo caso, verranno utilizzate insieme le istruzioni *analogWrite()* e *for*



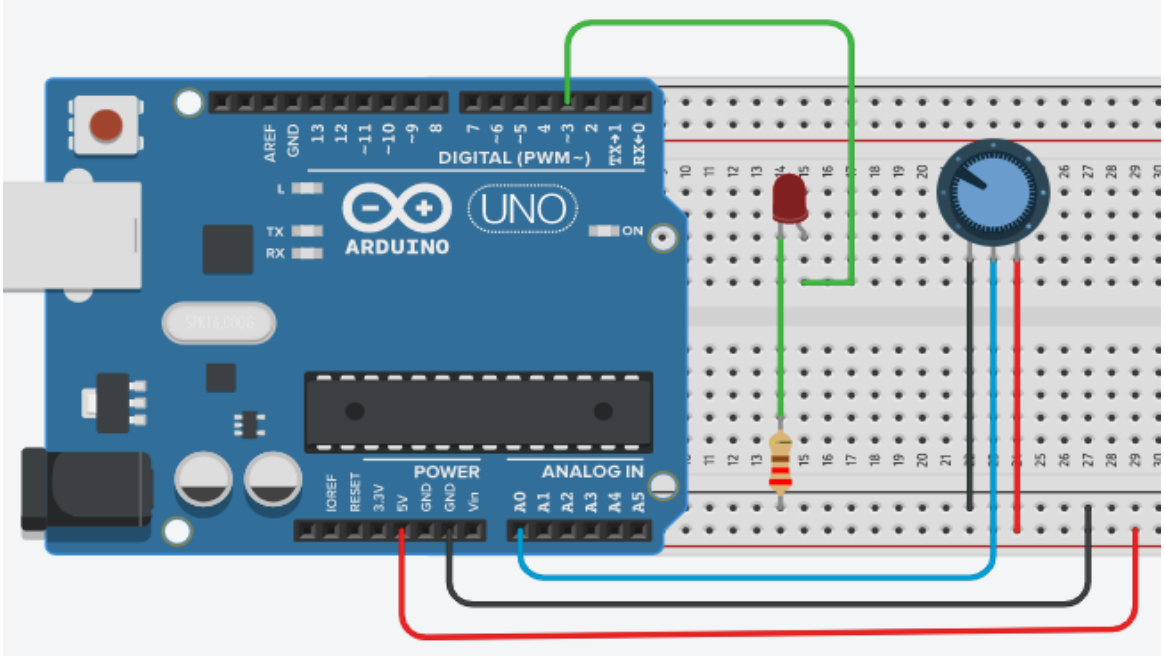

```
/* Modificare la luminosità di un LED dal minimo al massimo   */

#define led1    6
int a;

void setup()          {
Serial.begin(9600);
pinMode(led1, OUTPUT);    }

void loop() {

for (a=0; a<=255; a++) {

Serial.println(a);

analogWrite(led1, a);

delay (100);
          }              }
```

NOTA: Il voltmetro visualizza i valori di tensione tra 0 e 5 Volt, mentre il monitor seriale mostra i numeri interi compresi tra 0 e 255.

Circuito 19:

Titolo del circuito: "Il potenziometro regola la luminosità del led"

Spiegazione del circuito: Utilizzando il potenziometro (10K), è possibile regolare la luminosità di un LED da un'intensità minima a una massima.In questo caso, viene utilizzata la funzione map ().



Nota:
Funzione map():

Rimappa un numero da un intervallo a un altro. Ovvero, un valore pari a *fromLow* verrà mappato su *toLow*, un valore pari a *fromHigh* su *toHigh*, e i valori intermedi verranno mappati proporzionalmente.

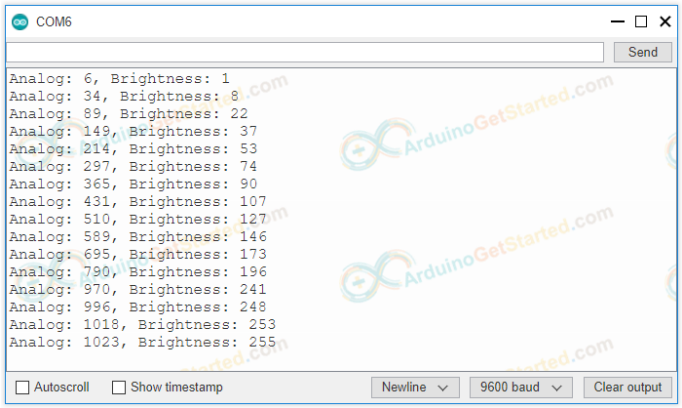
La funzione map() non limita i valori all'interno dell'intervallo specificato, poiché in alcuni casi è utile ottenere valori al di fuori di tale intervallo. È possibile utilizzare la funzione constrain(), prima o dopo l'uso di map(), per restringere i valori all'interno degli intervalli desiderati.

La funzione map() esegue operazioni su numeri interi, quindi non restituisce valori frazionari anche quando il calcolo lo richiederebbe. Le parti decimali vengono troncate, senza essere arrotondate o mediate:

Sintassi: map(value, fromLow, fromHigh, toLow, toHigh);

Esempio: val = map(val, 0, 1023, 0, 255);

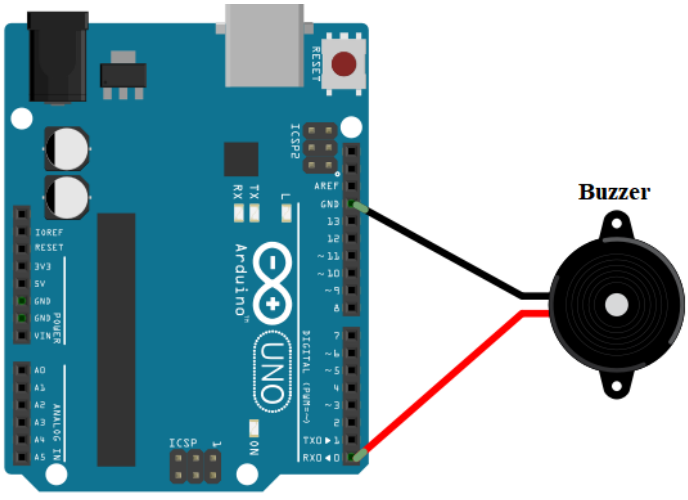
```
/*    Il potenziometro regola la luminosità del LED.    */

int LED_PIN = 3;

void setup() {                                                    Serial.begin(9600);
pinMode(LED_PIN, OUTPUT);      }

void loop() {

// legge il valore in ingresso dal pin analogico A0 (valore compreso tra 0 e 1023)

int analogValue = analogRead(A0);

// scala l'ingresso analogico in un valore di luminosità (valore tra 0 e 255)

int brightness = map(analogValue, 0, 1023, 0, 255);

// imposta la luminosità del LED collegato al pin 3

analogWrite(LED_PIN, brightness);

// visualizza il valore sul monitor seriale

Serial.print("Analog: ");

Serial.print(analogValue);

Serial.print(" Brightness: ");

Serial.println(brightness);

delay(100);                                                                    //
finestra del monitor seriale }
```



Circuito 20:

Titolo del circuito: "Utilizzare un buzzer"

Spiegazione del circuito: Un buzzer è un dispositivo di segnalazione acustica, che può essere meccanico, elettromeccanico o piezoelettrico (abbreviato in piezo). Nel settore industriale, i buzzer sono spesso utilizzati come dispositivi di allarme, emettendo suoni ronzanti o beep acustici per segnalare eventi particolari.



/* Utilizzare un buzzer nei circuiti Arduino*/

#define buzzer_pin 0

void setup() {

pinMode(buzzer_pin, OUTPUT); }

void loop() {

digitalWrite(buzzer_pin, HIGH);

delay(500);

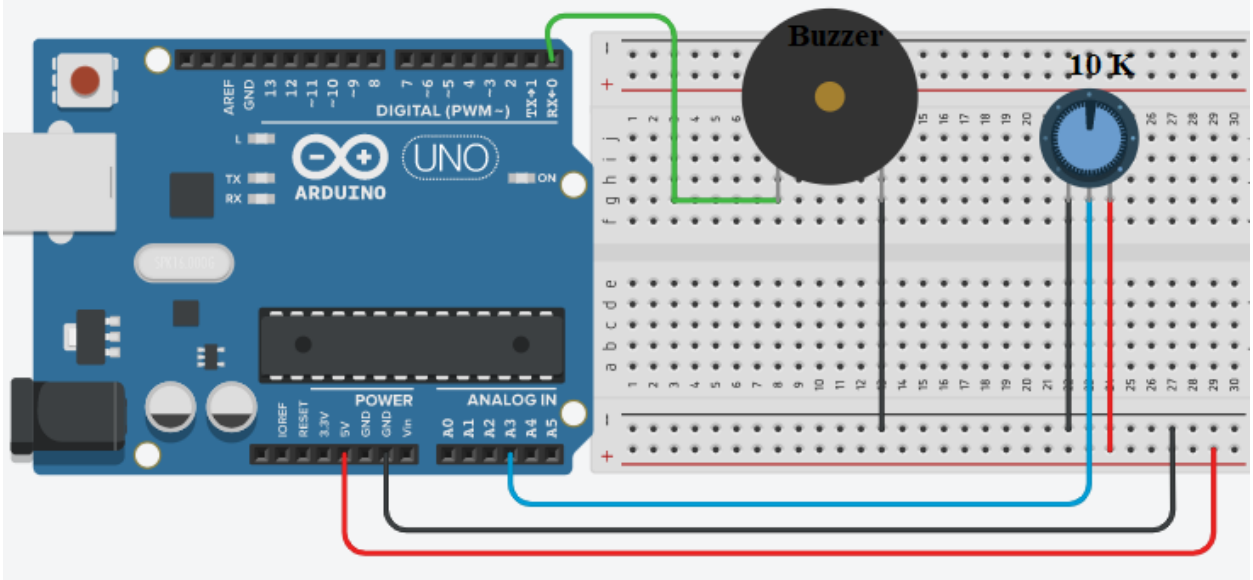
digitalWrite(buzzer_pin, LOW);

delay(500); }

Circuito 21:

Titolo del circuito: "Controllare un buzzer con un potenziometro"

Spiegazione del circuito: "Quando il valore del potenziometro è superiore a 500, il buzzer emette un suono.



/* Controllare un buzzer con un potenziometro */

const int POT_PIN = A3; // Pin di Arduino collegato al pin del potenziometro
const int BUZZER_PIN = 0; // Il pin di Arduino collegato al pin del Buzzer.
const int ANALOG_THRESHOLD = 500; **int analogValue;**

void setup() {

pinMode(BUZZER_PIN, OUTPUT); // imposta il pin di Arduino in modalità output

}

void loop() {

```
analogValue = analogRead(POT_PIN);          // Legge il valore in ingresso dal pin analogico

if(analogValue > ANALOG_THRESHOLD)
digitalWrite(BUZZER_PIN, HIGH);          // Accende il Buzzer

else digitalWrite(BUZZER_PIN, LOW);          // Spegne il Buzzer

}
```

Co-funded by the
Erasmus+ Programme
of the European Union

Erasmus+ KA210-VET

Small-scale partnerships in vocational education and training

Titolo Progetto: “Using Arduinos in Vocational Training”

Acronimo Progetto : “UsingARDinVET”

Progetto Numero: “2023-1-RO01-KA210-VET-000156616”

Modulo LCD e KIT di Formazione

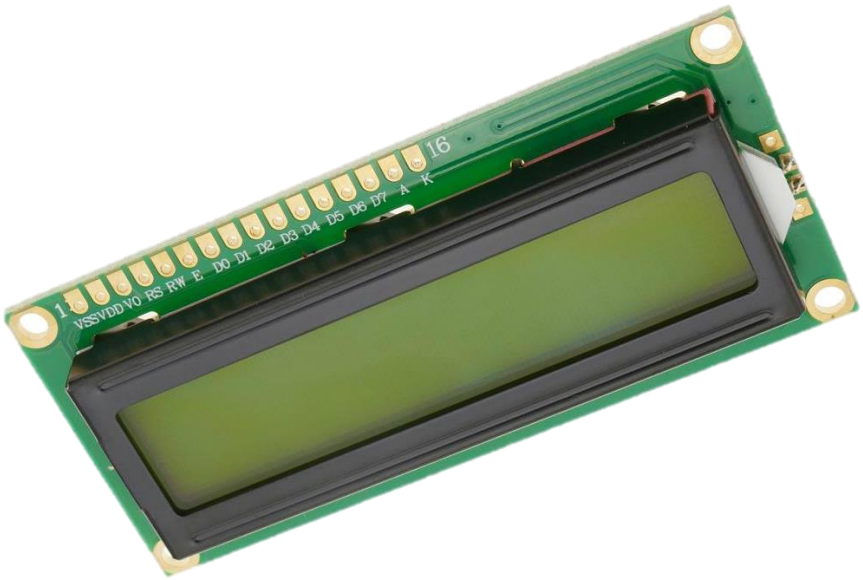
Modulo LCD e KIT di Formazione

In questo modulo vogliamo spiegare come visualizzare messaggi di stato o letture dei sensori di Arduino su display LCD. Sono estremamente comuni e rappresentano un modo rapido per aggiungere un'interfaccia leggibile al vostro progetto.

LCD è l'abbreviazione di Liquid Crystal Display (Display a Cristalli Liquidi). Si tratta di un display che utilizza cristalli liquidi per produrre un'immagine visibile. Quando viene applicata corrente a questo speciale tipo di cristallo, questo diventa opaco, bloccando la retroilluminazione che si trova dietro lo schermo. Di conseguenza, quella particolare area diventerà più scura rispetto alle altre. Ed è così che i caratteri vengono visualizzati sullo schermo.

Interfacciamento di un modulo LCD a 16x2 caratteri con Arduino

Esistono diversi tipi di display LCD che puoi collegare al tuo Arduino. Il più comune è basato sul chip controller LCD con interfaccia parallela di Hitachi, l'HD44780.
Questi LCD sono ideali per visualizzare solo testo/caratteri, da cui il nome "LCD a caratteri". Il display ha una retroilluminazione a LED e può visualizzare 32 caratteri ASCII su due righe, ciascuna composta da 16 caratteri.
Per ogni carattere sul display sono presenti piccoli rettangoli, ognuno dei quali è una griglia di 5x8 pixel.
Sebbene visualizzino solo testo, sono disponibili in diverse dimensioni e colori: ad esempio, 16x1, 16x4, 20x4, con testo bianco su sfondo blu, con testo nero su sfondo verde e molti altri.
La community Arduino ha già sviluppato una libreria per gestire gli LCD HD44780 (**LiquidCrystal Library**); Quindi li interfacceremo tra poco.



Di seguito è riportato il pinout del display LCD:

VSS: è un pin di massa e deve essere collegato alla massa di Arduino;

VDD: collegato a 5 V;

VD: per la regolazione del contrasto (collegato al pin centrale del potenziometro);

RS: controlla in quale zona di memoria del display LCD vengono memorizzati i dati inviati;

R/W: pin per selezionare la modalità di lettura/scrittura;

E: se abilitato, consente al modulo LCD di eseguire istruzioni speciali;

Da D0 a D7: trasmissione dati;

A e K: anodo e catodo per fornire retroilluminazione al modulo LCD.

Arduino - Funzioni LCD (comandi)

LiquidCrystal lcd() - Crea una variabile di tipo LiquidCrystal. Il display può essere controllato tramite 4 o 8 linee dati. Nel primo caso, omettere i numeri dei pin da d0 a d3 e lasciare tali linee scollegate. Il pin RW può essere collegato a massa anziché a un pin di Arduino; in tal caso, omettetelo dai parametri di questa funzione. Sintassi:

LiquidCrystal(rs, enable, d4, d5, d6, d7)
LiquidCrystal(rs, rw, enable, d4, d5, d6, d7)
LiquidCrystal(rs, enable, d0, d1, d2, d3, d4, d5, d6, d7)
LiquidCrystal(rs, rw, enable, d0, d1, d2, d3, d4, d5, d6, d7)

Parametri:
rs: il numero del pin Arduino collegato al pin RS sul display LCD

rw: il numero del pin Arduino collegato al pin RW sul display LCD (facoltativo)

enable: il numero del pin Arduino collegato al pin enable sul display LCD

d0, d1, d2, d3, d4, d5, d6, d7: i numeri dei pin Pin Arduino collegati ai pin dati corrispondenti sul display LCD. d0, d1, d2 e d3 sono opzionali; se omessi, il display LCD verrà controllato utilizzando solo le quattro linee dati (d4, d5, d6, d7).

lcd.begin() - Inizializza l'interfaccia per lo schermo LCD e specifica le dimensioni (larghezza e altezza) del display. begin() deve essere chiamato prima di qualsiasi altro comando della libreria LCD.

Sintassi:
lcd.begin(colonne, righe)

Parametri:
lcd: una variabile di tipo LiquidCrystal

cols: il numero di colonne del display

rows: il numero di righe del display

lcd.print() - Stampa del testo sul display LCD. Sintassi:

lcd.print(dati)
lcd.print(dati, BASE)

Parametri:
lcd: una variabile di tipo LiquidCrystal

data: i dati da stampare (char, byte, int, long o string)

BASE (facoltativa): la base in cui stampare i numeri: BIN per binario (base 2), DEC per decimale (base 10), OCT per ottale (base 8), HEX per esadecimale (base 16).

Returns
byte print() restituirà il numero di byte scritti, sebbene la lettura di tale numero sia facoltativa.

lcd.setCursor() - Posiziona il cursore LCD; ovvero, imposta la posizione in cui verrà visualizzato il testo successivo scritto sull'LCD. Sintassi:

lcd.setCursor(col, row)

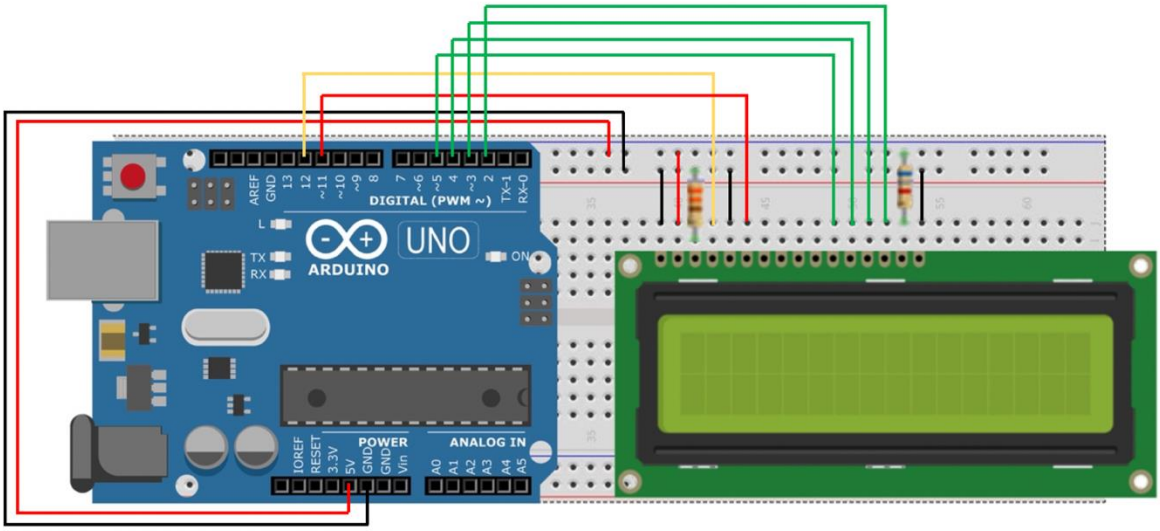
Parametri:
lcd: una variabile di tipo LiquidCrystal

col: la colonna in cui posizionare il cursore (con 0 come prima colonna)

row: la riga in cui posizionare il cursore (con 0 come prima riga)
lcd.clear(); - Cancella lo schermo LCD e posiziona il cursore nell'angolo in alto a sinistra.

Sintassi: lcd.clear()
Parametri: lcd: una variabile di tipo LiquidCrystal

Circuito 1:
Titolo del circuito: "Visualizza un messaggio sul display LCD"
Spiegazione del circuito: È possibile collegare un display LCD al microcontrollore e visualizzare i messaggi desiderati sullo schermo.
Nota: è necessario includere la libreria LiquidCrystal.h per utilizzare le funzioni LCD descritte di seguito.




```
/* Stampa un messaggio */

#include <LiquidCrystal.h> // include il codice della libreria

//costanti per il numero di righe e colonne nel display LCD

const int numRows = 2;

const int numCols = 16;

// inizializza la libreria con i numeri dei pin dell'interfaccia

LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

void setup()

{

lcd.begin(numCols, numRows);

lcd.print("hello, world!"); // Stampa un messaggio sul display LCD.

}
void loop() {
}
```

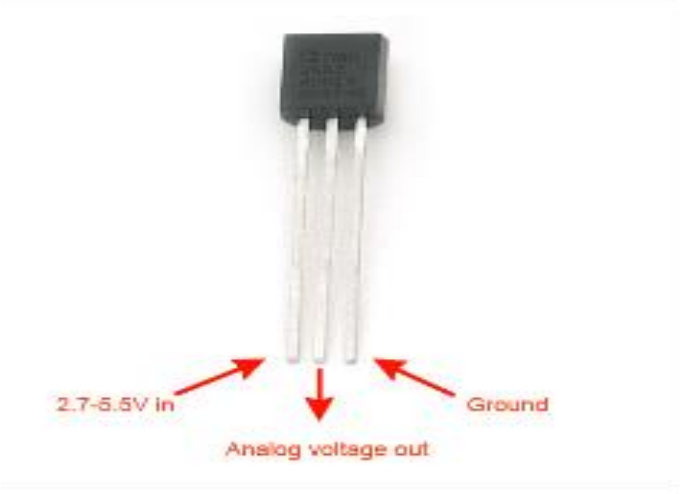
Circuito 2:

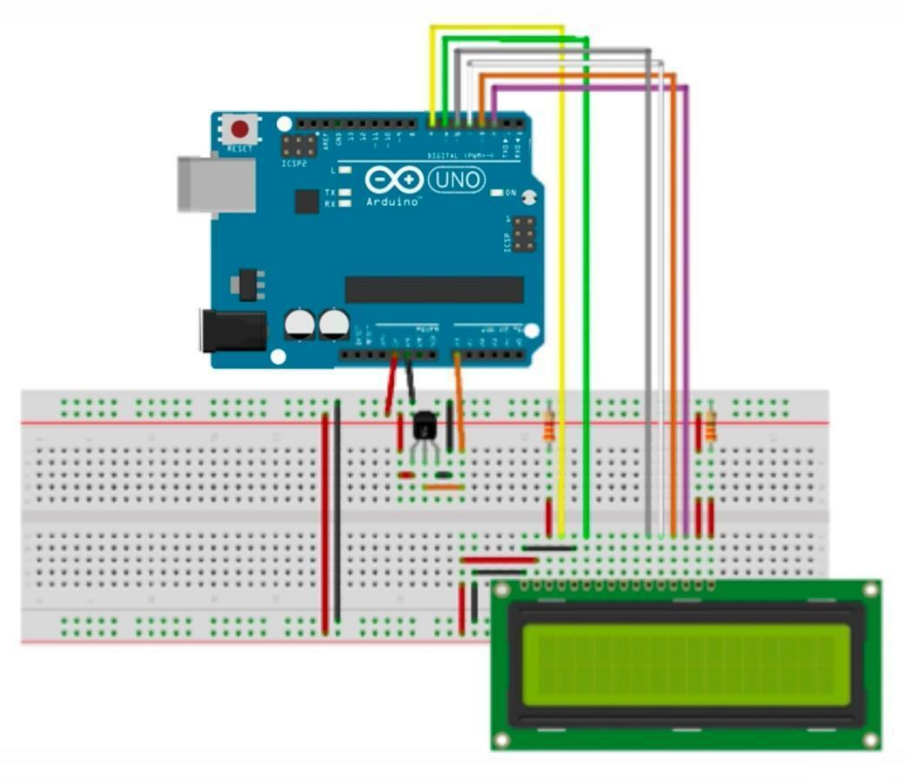
Titolo del circuito: "Termometro digitale"

Spiegazione del circuito: Per creare un termometro digitale con Arduino. La temperatura verrà misurata con il sensore di temperatura LM35 e dovrà essere visualizzata su un display LCD, aggiornato ogni secondo.

Il pin Vo del display LCD regola il contrasto dei caratteri visualizzati. Come accennato in precedenza, deve essere collegato a una resistenza da 3,3 kΩ collegata a GND. Tuttavia, in alcuni display potrebbe essere necessario collegare il pin Vo direttamente a GND.

Nota: il sensore LM35 ha 3 terminali: uno per l'alimentazione, uno per la massa e uno per l'uscita della tensione proporzionale alla temperatura rilevata, pari a 10 mV per ogni grado centigrado, calibrata in gradi Celsius.



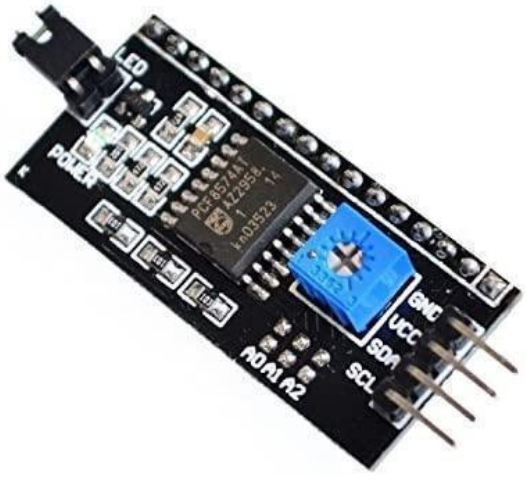



```
/* Termometro digitale */

#include <LiquidCrystal.h> //Libreria per il controllo del display LCD

#define pin_temp A0 //Pin di collegamento del sensore di temperatura Vout

float temp = 0; //Variabile in cui verrà memorizzata la temperatura rilevata

LiquidCrystal lcd(7, 6, 5, 4, 3, 2); //Inizializzazione della libreria con i pin del display LCD

void setup()

{
lcd.begin(16, 2); //Impostazione del numero di colonne e righe nel display LCD lcd.setCursor(0,
0); //Sposta il cursore sulla prima riga (riga 0) e sulla prima colonna lcd.print ("Temperatura:");
Stampa il messaggio "Temperatura:" sulla prima riga

//*Imposta Vref ADC a 1,1 V
(per una maggiore precisione nel calcolo della temperatura)

IMPORTANTE: se utilizzi Arduino Mega, sostituisci INTERNAL con INTERNAL1V1 */
analogReference(INTERNAL);
}
```

```
void loop()
{
/*calcola la temperatura ============================================*/
temp = 0;
for (int i = 0; i < 5; i++) { //Esegue l'istruzione successiva 5 volte
temp += (analogRead(pin_temp) / 9.31); // Calcola la temperatura e la somma nella variabile
'temp'

}
temp /= 5; //Calcola la media matematica dei valori di temperatura
/*=================================================================
=*/
/*Vedo la temperatura sul display LCD
================================================*/
lcd.setCursor(0, 1); //lcd.print(temp); Sposta il cursore sulla prima colonna e sulla seconda riga
lcd.print(temp);
// Temperatura visualizzata sul display LCD

lcd.print(" C"); // Visualizza uno spazio e il carattere 'C' sul display
/*============================================*/
delay(1000); // Ritardo di un secondo (modificabile)
}
```

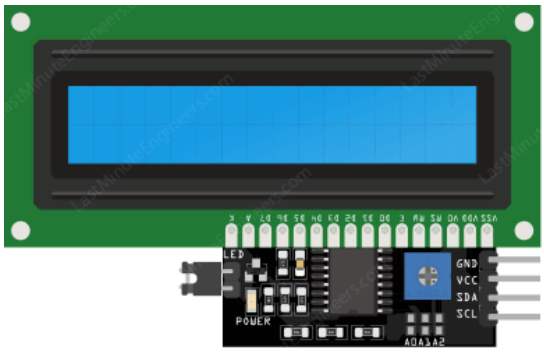
Interfacciamento di un LCD I2C con Arduino

Se si desidera collegare un display LCD ad Arduino, è necessario utilizzare molti pin su Arduino. Anche in modalità a 4 bit, Arduino richiede comunque un totale di sette connessioni, ovvero la metà dei pin I/O digitali disponibili.
La soluzione è utilizzare un display LCD che si interfaccia con il protocollo I2C. Occupa solo due pin I/O, che non possono nemmeno far parte di un set di pin I/O digitali e possono essere condivisi anche con altri dispositivi I2C.
Il display LCD I2C più diffuso è costituito da un display LCD a caratteri basato su HD44780 e da un adattatore LCD I2C.



La parte più importante dell'adattatore è il chip I/O Expander a 8 bit, PCF8574. Questo chip converte i dati I2C provenienti da un Arduino nei dati paralleli richiesti dal display LCD. La scheda è dotata anche di un piccolo potenziometro per regolare con precisione il contrasto del display. Se si utilizzano più dispositivi sullo stesso bus I2C, potrebbe essere necessario impostare un indirizzo I2C diverso per la scheda, in modo che non entri in conflitto con un altro dispositivo I2C. Per questo motivo, la scheda dispone di tre ponticelli a saldare (A0, A1 e A2).

Un LCD I2C ha solo 4 pin che lo interfacciano con Arduino.



Il pinout è il seguente:

GND: è un pin di massa e deve essere collegato alla massa di Arduino;

VCC: fornisce alimentazione al modulo e al LCD. Collegarlo all'uscita a 5 V di Arduino o a un alimentatore separato;

SDA: è un pin per i dati seriali. Questa linea viene utilizzata sia per la trasmissione che per la ricezione. Collegarlo al pin SDA di Arduino;

SCL: è un pin per il clock seriale. Si tratta di un segnale di temporizzazione fornito dal dispositivo Bus Master. Collegarlo al pin SCL di Arduino.

Sulle schede Arduino con layout R3, SDA (linea dati) e SCL (linea clock) si trovano sui pin header vicino al pin AREF. Sono anche noti come A5 (SCL) e A4 (SDA). Consultare la tabella sottostante per identificare i pin corretti in base al modello di Arduino.

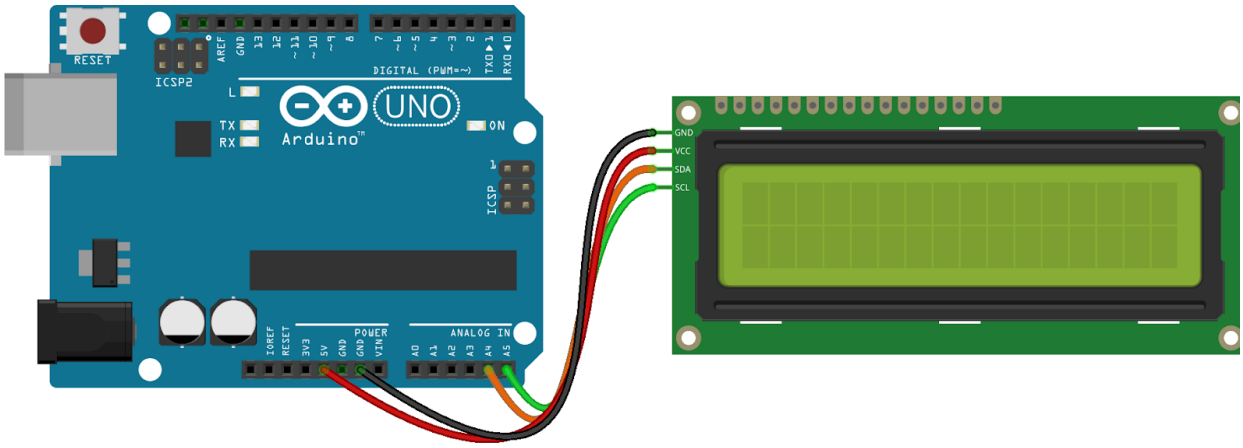
	SCL	SDA
Arduino Uno	A5	A4
Arduino Nano	A5	A4
Arduino Mega	21	20
Leonardo/Micro	3	2

Circuito 3:

Titolo del circuito: "Print a message to the I2C LCD"

Spiegazione del circuito: è possibile collegare un display LCD al microcontrollore con soli 4 pin e visualizzare i messaggi desiderati sullo schermo.

Nota: è necessario installare una libreria chiamata LiquidCrystal_I2C. Questa libreria è una versione migliorata della libreria LiquidCrystal inclusa nell'IDE Arduino.




```
/*  Print a message on a I2C Display */

#include <LiquidCrystal_I2C.h>

LiquidCrystal_I2C lcd(0x3F,16,2);  // set the LCD address to 0x3F for a 16 chars and 2 line display

void setup() {

  lcd.init();

  lcd.clear();

  lcd.backlight();      // Make sure backlight is on

  // Print a message on both lines of the LCD.

  lcd.setCursor(2,0);   //Set cursor to character 2 on line 0

  lcd.print("Hello world!");

  lcd.setCursor(2,1);   //Move cursor to character 2 on line 1

  lcd.print("with I2C protocol!");

}

void loop() {

}
```

Interfacciamento di un modulo display grafico OLED con Arduino

Un'altra possibilità per utilizzare il protocollo I2C è scegliere un display OLED (diodo organico a emissione di luce). Sono super leggeri e sottili e producono immagini più luminose e nitide.

Un display OLED funziona senza retroilluminazione. Per questo motivo, il display offre un contrasto così elevato, un angolo di visione estremamente ampio e può visualizzare livelli di nero profondi. L'assenza di retroilluminazione riduce significativamente la potenza necessaria per il funzionamento dell'OLED.

Come possiamo vedere dalla figura, il pin out è la classica interfaccia I2C a quattro pin già vista sopra.

La community Arduino ha già sviluppato alcune librerie per gestire questi display OLED, come la libreria SSD1306 di Adafruit. Per installare la libreria, andare su Sketch > Includi libreria > Gestisci librerie... Attendere che Library Manager scarichi l'indice delle librerie e aggiorni l'elenco delle librerie installate.

Arduino - OLED Graphic Display Module Functions (Commands)

pinMode();

display.begin()

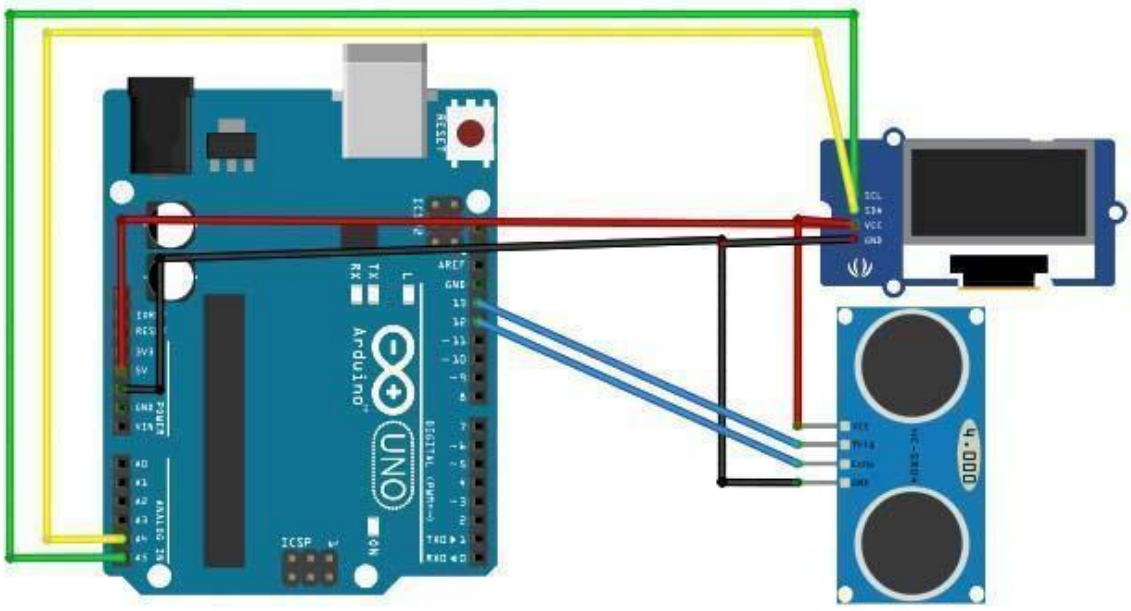
display.clearDisplay();

Circuito 4:

Titolo del circuito: "Sensore di distanza"

Spiegazione del circuito: La distanza verrà misurata con il sensore a ultrasuoni HC-SR04 e visualizzata su un display OLED.

Nota: ci sono solo quattro pin di cui occuparsi sull'HC-SR04: VCC (Alimentazione), Trig (Trigger), Echo (Ricezione) e GND (Terra).



/* Distance sensor */

#include <SPI.h> // this library allows you to communicate with SPI devices, with the Arduino as the master device.

#include <Wire.h> // this library allows you to communicate with I2C / TWI devices

#include <Adafruit_GFX.h> // the OLED display's libraries

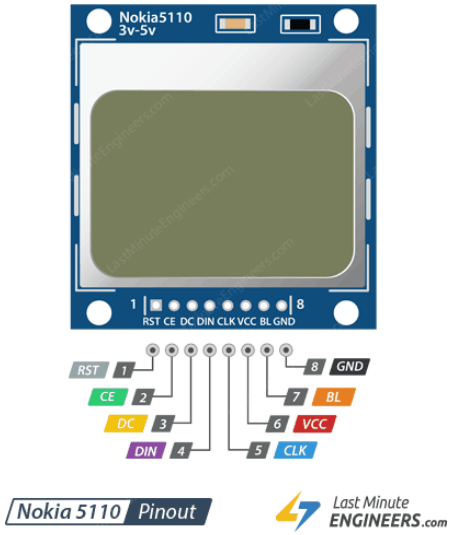
#include <Adafruit_SSD1306.h>

```
#define CommonSenseMetricSystem
#define trigPin 13       // define the pins of the sensor
#define echoPin 12

void setup() {
  Serial.begin (9600);
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
  display.begin(SSD1306_SWITCHCAPVCC, 0x3C); //initialize with the I2C addr 0x3C (128x64)
  display.clearDisplay();

}

void loop() {
  long duration, distance;

  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);

  duration = pulseIn(echoPin, HIGH);
  distance = (duration/2) / 29.1;
  display.setCursor(22,20);    //oled display setting cursor
  display.setTextSize(3);        //size of the text
```

```
 display.setTextColor(WHITE);        //if you write black it erases things
 display.println(distance);      //print our variable
 display.setCursor(85,20);
 display.setTextSize(3);


   display.println("cm");


 display.display();


 delay(500);
 display.clearDisplay();
  Serial.println(distance);//debug
}
```

Interfacing Nokia 5110 Graphic LCD Display with Arduino

È possibile interfacciare Arduino con piccoli LCD simili a quelli utilizzati dalla Nokia nei suoi cellulari 3310 e 5110. Questi display sono piccoli (solo circa 1,5"), poco costosi, facili da usare, consumano relativamente poco (solo 6-7 mA) e possono visualizzare sia testo che bitmap.



Si tratta di un display grafico da 84×48 pixel. Si interfaccia con i microcontrollori tramite un'interfaccia bus seriale simile all'SPI. Il display LCD è inoltre dotato di retroilluminazione in diversi colori come rosso, verde, blu e bianco. La retroilluminazione è costituita da quattro LED

disposti lungo i bordi del display. Dispone di 8 pin che lo interfacciano con Arduino; il pinout è il seguente:

- **RST**: resetta il display. È un pin attivo basso, il che significa che è possibile resettare il display tirandolo verso il basso. È anche possibile collegare questo pin al reset di Arduino in modo che reimposti automaticamente lo schermo;
- **CE (Chip Enable):** viene utilizzato per selezionare uno dei tanti dispositivi connessi che condividono lo stesso bus SPI;
- **D/C (Data/Command):** il pin indica al display se il dato ricevuto è un comando o un dato visualizzabile;
- **DIN:** è un pin dati seriali per l'interfaccia SPI;
- **CLK:** è un pin clock seriale per l'interfaccia SPI;
- **VCC:** fornisce alimentazione al display LCD che colleghiamo al pin a 3,3 V di Arduino;
- **BL (Backlight):** controlla la retroilluminazione del display. Per controllarne la luminosità, è possibile aggiungere un potenziometro o collegare questo pin a qualsiasi pin Arduino che supporti la tecnologia PWM;
- **GND:** è un pin di massa e deve essere collegato alla massa di Arduino.

È possibile collegare i pin di trasmissione dati a qualsiasi pin di I/O digitale. Il display LCD ha livelli di comunicazione di 3 V, quindi non è possibile collegare direttamente questi pin ad Arduino. Un modo è quello di aggiungere delle resistenze in linea con ciascun pin di trasmissione dati. Basta aggiungere delle resistenze da 10 kΩ tra i pin CLK, DIN, D/C e RST e una resistenza da 1 kΩ tra CE. Il pin di retroilluminazione (BL) è collegato a 3,3 V tramite una resistenza di limitazione della corrente da 330 Ω. È possibile aggiungere un potenziometro o collegare questo pin a qualsiasi pin Arduino che supporti PWM, se si desidera controllarne la luminosità. La community Arduino ha già sviluppato alcune librerie per gestire questi display NOKIA, come la libreria PCD8544 per LCD Nokia 5110 di Adafruit. Per installare la libreria, andare su Sketch > Includi libreria > Gestisci librerie... Attendere che Library Manager scarichi l'indice delle librerie e aggiorni l'elenco delle librerie installate.

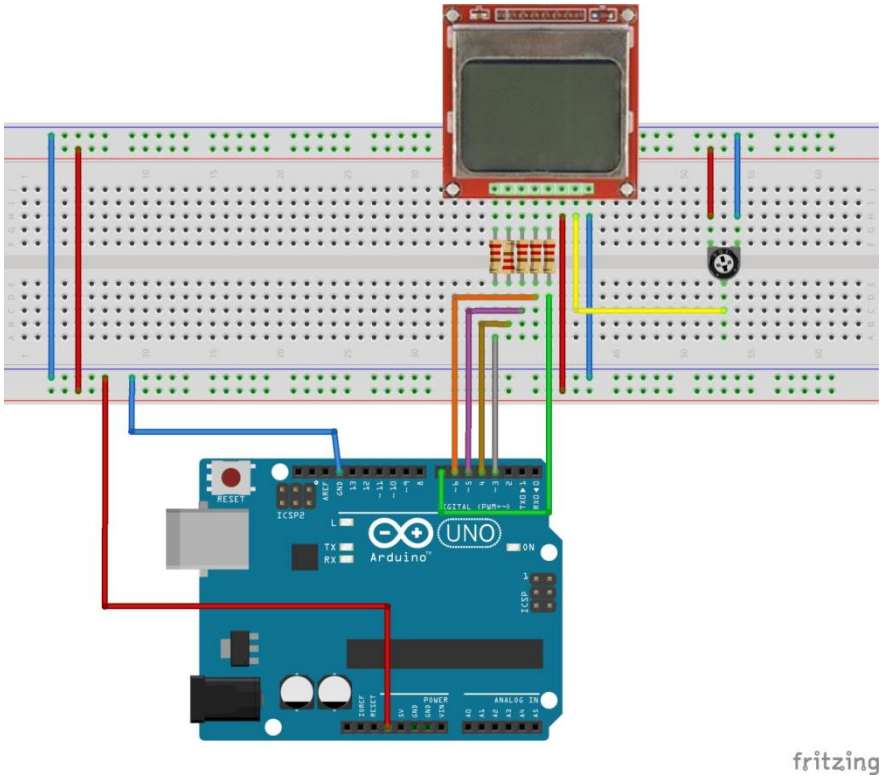
Circuito 5:

Titolo del circuito: "Rotazione del testo"

Spiegazione del circuito: È possibile ruotare il contenuto del display chiamando la funzione setRotation(). Permette di visualizzare il display in modalità verticale o capovolto.

Nota: la funzione accetta un solo parametro che corrisponde a 4 rotazioni cardinali. Questo valore può essere qualsiasi numero intero non negativo a partire da 0. Ogni volta che si aumenta il valore, il contenuto del display viene ruotato di 90 gradi in senso antiorario. Ad esempio:

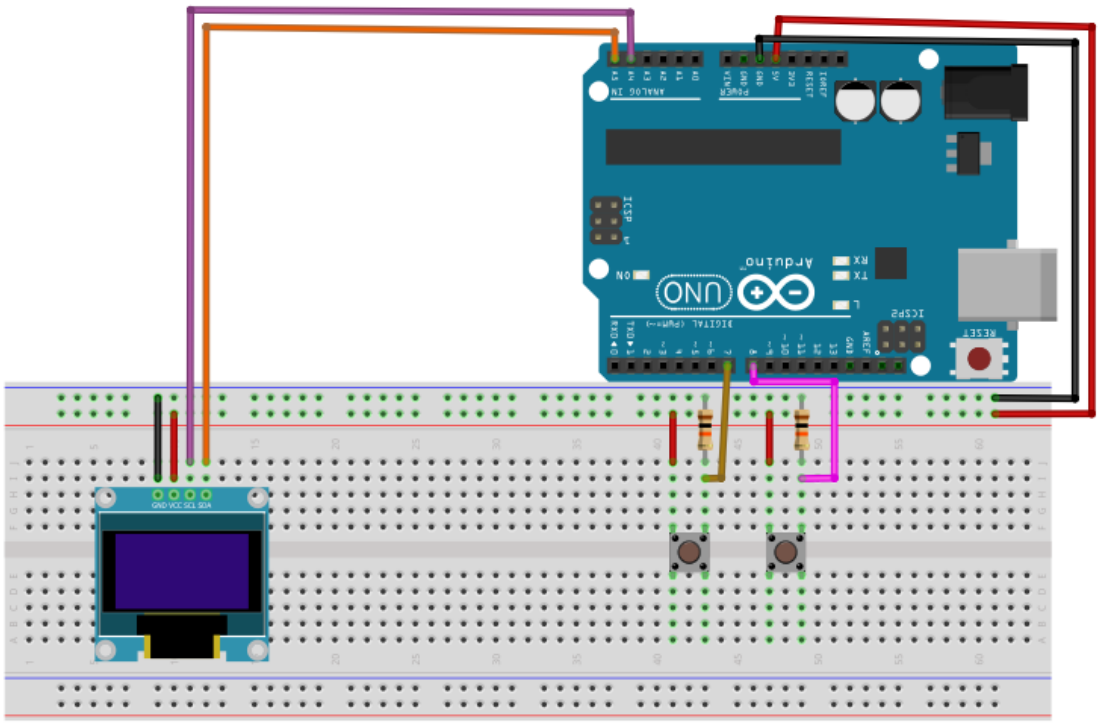
- 0 – Mantiene lo schermo nell'orientamento orizzontale standard.
- 1 – Ruota lo schermo di 90° verso destra.
- 2 – Capovolge lo schermo.
- 3 – Ruota lo schermo di 90° verso sinistra.




```
/*  Text Rotation  */
#include <SPI.h>        // this library allows you to communicate with SPI devices, with the Arduino as the master device.

#include <Adafruit_GFX.h>            // the OLED display's libraries
#include <Adafruit_PCD8544.h>

// Declare LCD object for software SPIAdafruit_PCD8544(CLK,DIN,D/C,CE,RST);
Adafruit_PCD8544 display = Adafruit_PCD8544(7, 6, 5, 4, 3);

void setup()   {
```

```
	Serial.begin(9600);

	//Initialize Display
	display.begin();

	display.setContrast(57);	// you can change the contrast around to adapt the display

	display.clearDisplay();	// clear the buffer.

// Text Rotation
while(1)
{
	display.clearDisplay();
	display.setRotation(rotatetext);
	display.setTextSize(1);
	display.setTextColor(BLACK);
	display.setCursor(0,0);
	display.println("Text Rotation");
	display.display();
	delay(1000);
	display.clearDisplay();
	rotatetext++;
}
}

void loop() {}
```

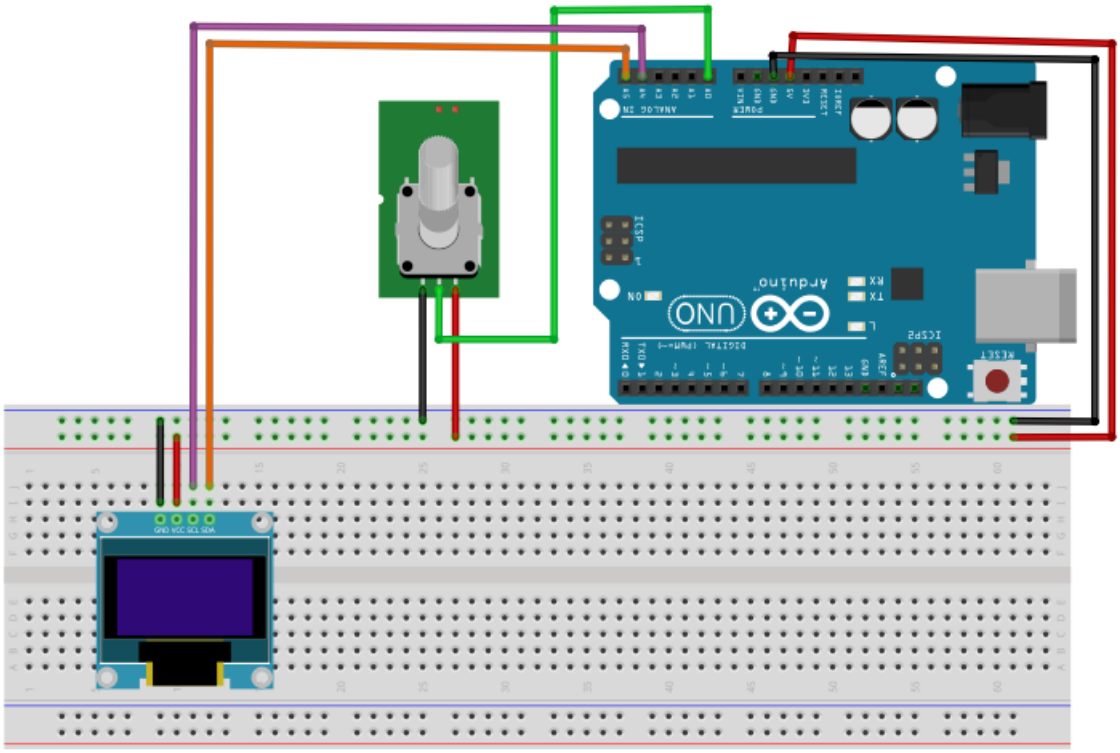
Circuito 6:
Titolo del circuito: "Contatore a pulsanti"
Spiegazione del circuito: un display OLED che mostra un numero, che può essere incrementato e decrementato premendo due pulsanti diversi.



```
#include <U8glib.h> //lcd libraries
#include "U8glib.h"
U8GLIB_SSD1306_128X64
u8g(U8G_I2C_OPT_NONE|U8G_I2C_OPT_DEV_0);  //display model
int button_plus = 8, button_minus = 7; //declaration pin buttons
int state_plus, state_minus, number=0;
void setup() {
  pinMode(button_plus,INPUT);
  pinMode(button_minus,INPUT);
  Serial.begin(9600);
  u8g.setFont(u8g_font_fub25n); //font to be used for the write of the number
}
void write_number(void) {
  u8g.setPrintPos(10,50);
  u8g.print(number);
}
void loop() {
```

```
  //buttons
  state_plus = digitalRead(button_plus);
  state_minus = digitalRead(button_minus);
if(state_plus==1)
{
   number++;
   delay(50);
}
  if(state_minus==1)
  {
   number--;
   delay(50);
  }
  //write de number sul display
  u8g.firstPage();
  do {
   write_number();
  } while  ( u8g.nextPage() );
   u8g.firstPage();
}
```

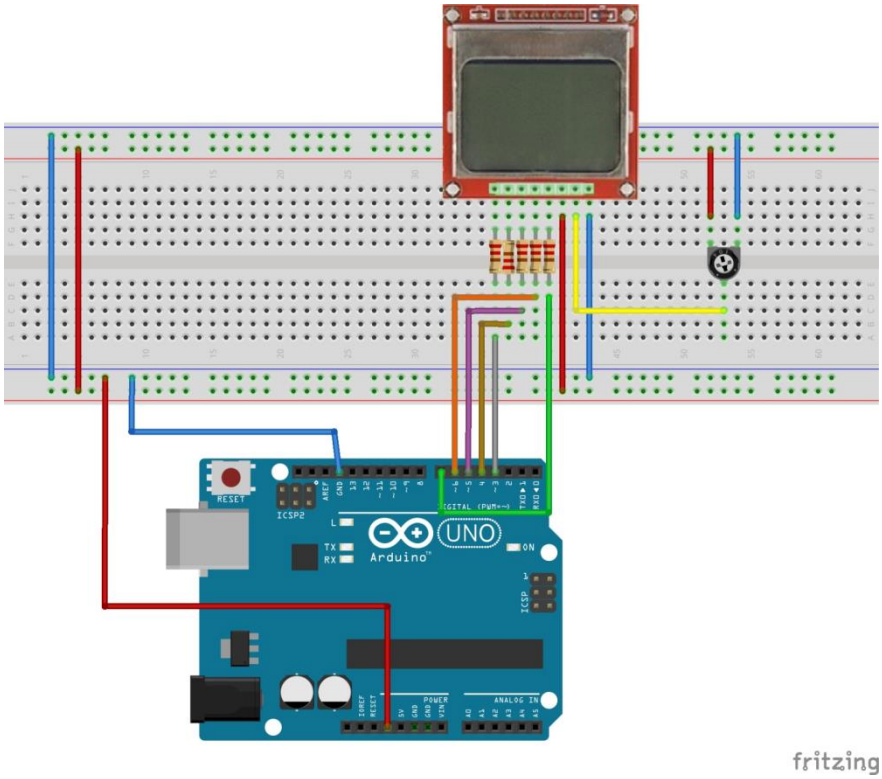
Circuito 7:
Titolo del circuito: "Contatore con potenziometro"
Spiegazione del circuito: un display OLED che mostra un numero, il cui valore aumenta e diminuisce ruotando un potenziometro.



```
#include <U8glib.h> //lcd libraries
#include "U8glib.h"
U8GLIB_SSD1306_128X64
u8g(U8G_I2C_OPT_NONE|U8G_I2C_OPT_DEV_0);  //display model

int potentiometer = 0; //declaration and potentiometer
int state_pot, stop_pot, difference, number=0;

void setup() {
  Serial.begin(9600);
  u8g.setFont(u8g_font_fub25n);//font to be used for the write of the number
}

void write_number(void) {
  u8g.setPrintPos(10,50);
  u8g.print(number);
}
```

```
void loop() {
  state_pot = analogRead(potentiometer);

  //potentiometer
  stop_pot = state_pot;//save the value when it is stop to see if the potentiometer turns clockwise or counterclockwise
  delay(100);
  state_pot = analogRead(potentiometer);
  difference = stop_pot-state_pot;
  if((difference>2)||(difference<2))//is used to avoid making trades if the potentiometer is stop
  {
    number=number-difference/5;//if difference is greater than 0 then it turns clockwise and adds otherwise it subtracts
    delay(100);
  }

  //write the number on display
  u8g.firstPage();
  do {
    write_number();
  } while
  ( u8g.nextPage() );
    u8g.firstPage();
}
```

Circuito 8:
Titolo del circuito: "Immagine Bmp di Marilyn"
Spiegazione del circuito: Come disegnare immagini bitmap sul display LCD del Nokia 5110. In questo esempio è presente un ritratto di Marilyn Monroe.
Nota: la risoluzione dello schermo del display LCD del Nokia 5110 è di 84×48 pixel, quindi immagini più grandi non verranno visualizzate correttamente. Per visualizzare un'immagine bitmap sul display LCD del Nokia 5110, dobbiamo chiamare la funzione drawBitmap(). Accetta sei parametri: coordinata X dell'angolo in alto a sinistra, coordinata Y dell'angolo in alto a sinistra, array di byte della bitmap monocromatica, larghezza della bitmap in pixel, altezza della bitmap in pixel e colore. Nel nostro esempio, l'immagine bitmap ha una dimensione di 84×48. Quindi, le coordinate X e Y sono impostate a 0, mentre larghezza e altezza sono impostate a 84 e 48.




```
/*  Marilyn Bmp Image */

#include <SPI.h>
#include <Adafruit_GFX.h>
#include <Adafruit_PCD8544.h>

Adafruit_PCD8544 display = Adafruit_PCD8544(7, 6, 5, 4, 3);

// 'Marilyn Monroe 84x48', 84x48px
const unsigned char MarilynMonroe [] PROGMEM = {
```

0x00, 0x00, 0x00, 0x7f, 0x00, 0x02, 0xfe, 0xf8, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0xbe, 0x00,
0x00, 0x1f, 0xe0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x20, 0x00, 0x00, 0x3f, 0x80, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0xf0, 0x00, 0x00, 0x1f, 0xe1, 0x80, 0x00, 0x00, 0x00, 0x00, 0x00, 0xc0,
0x00, 0x00, 0x0f, 0xf1, 0xc0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x0e, 0xd8, 0xe0,
0x00, 0x00, 0x00, 0x00, 0x00, 0x1f, 0x80, 0x00, 0x07, 0xe0, 0x70, 0x00, 0x00, 0x00, 0x00, 0x03,
0x3f, 0xe0, 0x00, 0x07, 0xf0, 0x78, 0x00, 0x00, 0x00, 0x00, 0x01, 0xe0, 0x70, 0x00, 0x0f, 0xee,
0x7c, 0x00, 0x00, 0x00, 0x00, 0x03, 0xc0, 0x00, 0x00, 0x0f, 0xf7, 0x1c, 0x00, 0x00, 0x00, 0x00,
0x07, 0x80, 0x00, 0x0f, 0xc7, 0xf3, 0x1e, 0x00, 0x00, 0x00, 0x00, 0x07, 0xc0, 0x00, 0x0f, 0xf3,
0xdf, 0x7f, 0x80, 0x00, 0x00, 0x00, 0x07, 0xfe, 0x00, 0x08, 0x7d, 0xef, 0xff, 0xc0, 0x00, 0x00,
0x00, 0x7f, 0xff, 0x80, 0x30, 0x0f, 0xfc, 0xe0, 0xc0, 0x00, 0x00, 0x01, 0x9e, 0x73, 0xc0, 0xe0,
0x07, 0xf8, 0xc1, 0xc0, 0x00, 0x00, 0x03, 0xfc, 0x00, 0x01, 0xc0, 0x0f, 0xfd, 0xe1, 0x80, 0x00,
0x00, 0x03, 0xf8, 0x00, 0x01, 0x9c, 0x0f, 0xff, 0xc1, 0xc0, 0x00, 0x00, 0x02, 0xc0, 0x00, 0x01,
0x9f, 0xbf, 0xfe, 0x01, 0x40, 0x00, 0x00, 0x02, 0x60, 0x00, 0x03, 0x07, 0xef, 0xff, 0x01, 0x40,
0x00, 0x00, 0x00, 0x60, 0x00, 0x07, 0x01, 0xf7, 0xff, 0x80, 0xc0, 0x00, 0x00, 0x00, 0x50, 0x01,
0xdf, 0x00, 0x7f, 0xff, 0x1c, 0x80, 0x00, 0x00, 0x00, 0x40, 0x01, 0xff, 0x00, 0x1f, 0xff, 0x1e,
0xe0, 0x00, 0x00, 0x02, 0x08, 0x00, 0x3f, 0x80, 0x07, 0xef, 0x03, 0xe0, 0x00, 0x00, 0x06, 0x08,
0x00, 0x03, 0xc0, 0x07, 0xdf, 0x07, 0xc0, 0x00, 0x00, 0x06, 0x08, 0x0f, 0x81, 0x80, 0x1f, 0xdf,
0x1f, 0x80, 0x00, 0x00, 0x03, 0x08, 0x1f, 0x98, 0x00, 0x3f, 0xfe, 0x19, 0x80, 0x00, 0x00, 0x18,
0x08, 0x3f, 0xfe, 0x00, 0x7f, 0xfe, 0x3f, 0x00, 0x00, 0x00, 0x08, 0x08, 0x30, 0x3f, 0x00, 0xff,
0xff, 0x3f, 0x00, 0x00, 0x00, 0x01, 0xe0, 0x76, 0x0f, 0x89, 0xff, 0xff, 0x9f, 0x00, 0x00, 0x00,

```
	0x03, 0xe0, 0x7f, 0xc3, 0x81, 0xff, 0xfe, 0x9f, 0x80, 0x00, 0x00, 0x03, 0xf0,
0x7f, 0xf3, 0xc3,
	0xff, 0xfe, 0x1f, 0x00, 0x00, 0x00, 0x03, 0xf0, 0x7f, 0xfd, 0xc3, 0xff, 0xfe, 0x5e,
0x00, 0x00,
	0x00, 0x03, 0xf0, 0x7f, 0xff, 0xc3, 0xff, 0xf3, 0x1e, 0x00, 0x00, 0x00, 0x03,
0xf0, 0x71, 0xff,
	0x87, 0xff, 0xe3, 0xff, 0x00, 0x00, 0x00, 0x07, 0xf0, 0x7c, 0x3f, 0x87, 0xff,
0xe3, 0xfe, 0x00,
	0x00, 0x00, 0x0f, 0xf0, 0x3c, 0xff, 0x05, 0xff, 0xf3, 0xfc, 0x00, 0x00, 0x00,
0x0f, 0xf0, 0x0f,
	0xfe, 0x09, 0xff, 0xf7, 0xfc, 0x00, 0x00, 0x00, 0x08, 0xf8, 0x01, 0xfc, 0x19,
0xff, 0xff, 0xf8,
	0x00, 0x00, 0x00, 0x0c, 0x78, 0x00, 0x00, 0x13, 0xff, 0xff, 0xf8, 0x00, 0x00,
0x00, 0x0e, 0x78,
	0x00, 0x00, 0x23, 0xff, 0xff, 0xf0, 0x00, 0x00, 0x00, 0x0e, 0xf8, 0x00, 0x00,
0x47, 0xff, 0xff,
	0xf0, 0x00, 0x00, 0x00, 0x0c, 0xfa, 0x00, 0x01, 0x8f, 0xff, 0xff, 0xe0, 0x00,
0x00, 0x00, 0x08,
	0x7b, 0x00, 0x03, 0x3f, 0xff, 0xff, 0xe0, 0x00, 0x00, 0x00, 0x0c, 0x3b, 0xf8,
0x0f, 0xff, 0xff,
	0xff, 0xe0, 0x00, 0x00, 0x00, 0x0f, 0xbb, 0xff, 0xff, 0xff, 0xff, 0xff, 0xf0, 0x00,
0x00, 0x00,
	0x07, 0xfb, 0xff, 0xff, 0xff, 0xff, 0xff, 0xf0, 0x00, 0x00, 0x00, 0x00, 0x71, 0xff,
0xff, 0xff,
	0xff, 0xff, 0xe0, 0x00, 0x00, 0x00, 0x00, 0x41, 0xff, 0xff, 0xff, 0xff, 0xff, 0xe0,
0x00, 0x00
};

void setup()  {
	Serial.begin(9600);

	display.begin();

	display.setContrast(57);

	display.clearDisplay();

	// Display bitmap
	display.drawBitmap(0, 0,  MarilynMonroe, 84, 48, BLACK);
	display.display();
```

```
        // Invert Display
        //display.invertDisplay(1);
}

void loop() {}
```

Co-funded by the
Erasmus+ Programme
of the European Union

Erasmus+ KA210-VET

Small-scale partnerships in vocational education and training

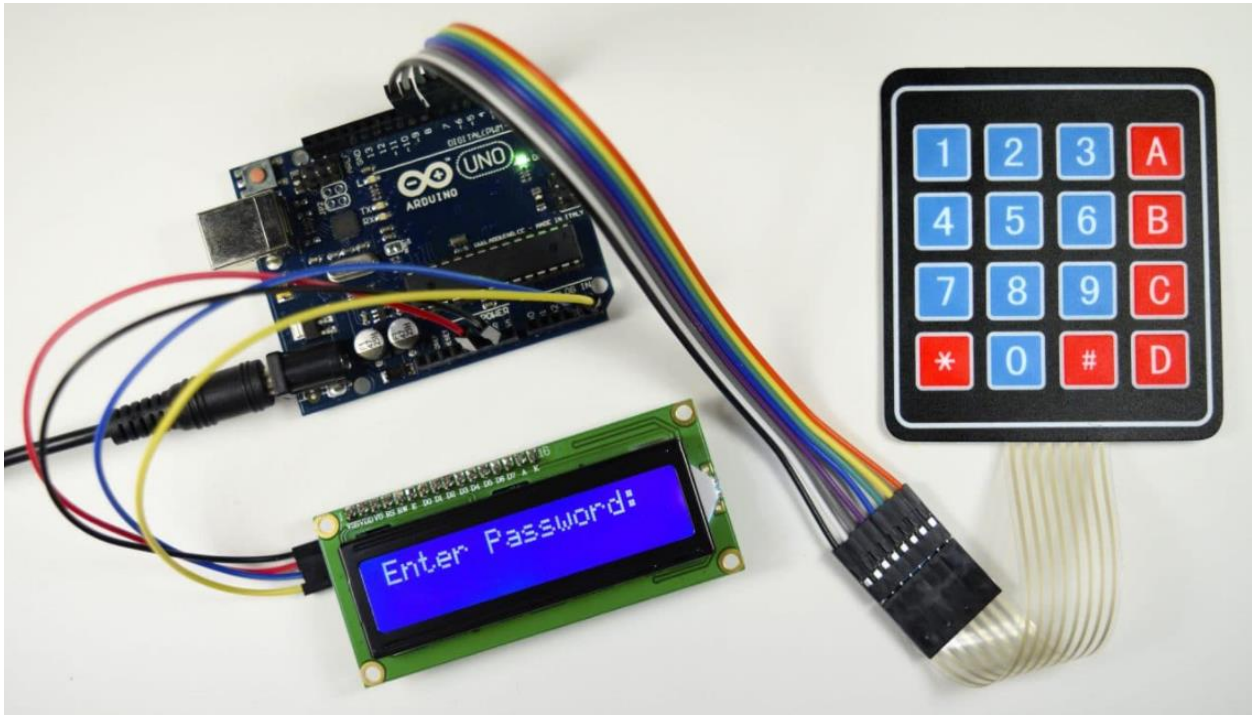
Project Title: “Using Arduinos in Vocational Training”

Project Acronym: “UsingARDinVET”

Project No: “2023-1-RO01-KA210-VET-000156616”

Modulo Tastiera e kit di formazione

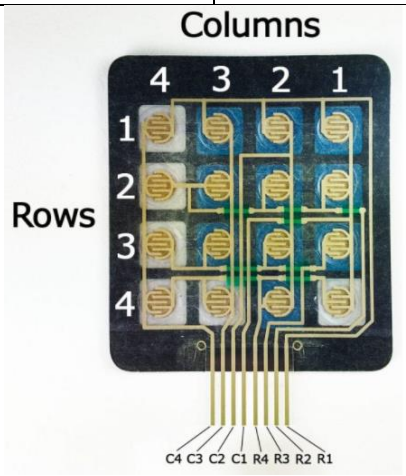
Modulo tastiera e kit di formazione



Cos'è una tastiera?

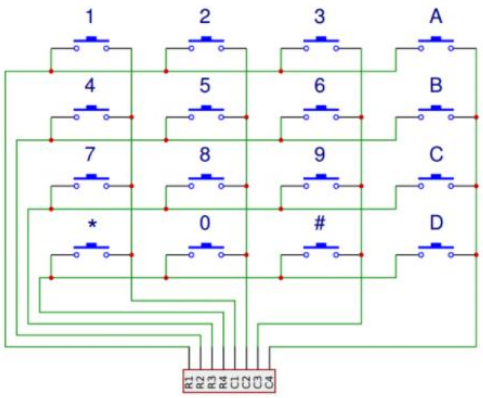
Una tastiera è un sistema di pulsanti disposti a matrice che funziona come un dispositivo di commutazione per collegare una riga a una colonna.

Utilizzeremo una tastiera a membrana con matrice 4x4, perché è sottile e dotata di supporto adesivo, in modo da poter essere incollata sulla maggior parte delle superfici piane. Sotto ogni tasto si trova un interruttore a membrana. Ogni interruttore di una fila è collegato agli altri interruttori della fila tramite una pista conduttiva posta sotto il tastierino. Ogni interruttore di una colonna è collegato allo stesso modo: un lato dell'interruttore è collegato a tutti gli altri interruttori di quella colonna tramite una pista conduttiva. Ogni riga e colonna è collegata a un singolo pin, per un totale di 8 pin su una tastiera 4x4.

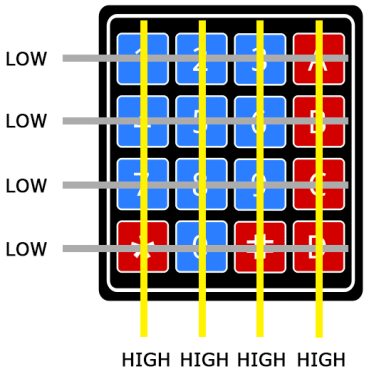
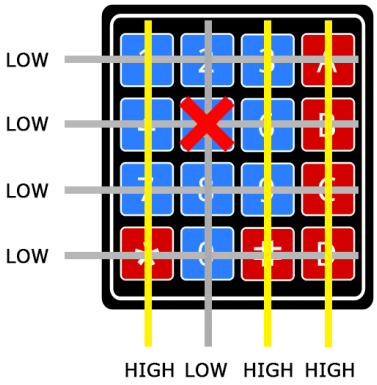


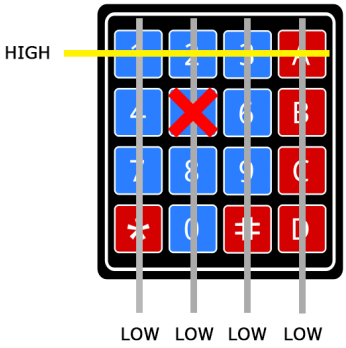
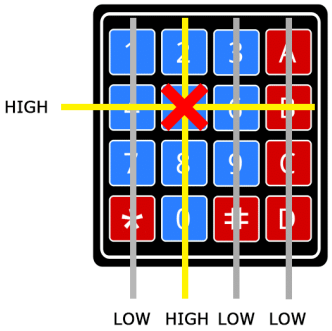
Premendo un pulsante si chiude l'interruttore tra una traccia di colonna e una di riga, consentendo il passaggio di corrente tra un pin di colonna e un pin di riga.

Lo schema di una tastiera 4x4 mostra come sono collegate le righe e le colonne:



Arduino rileva quale pulsante è stato premuto rilevando il pin di riga e di colonna collegato al pulsante.
Questo avviene in quattro fasi:

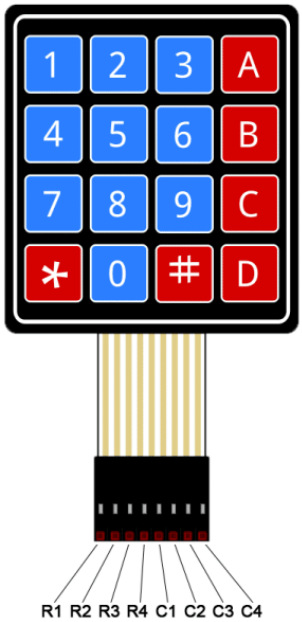
1. Innanzitutto, quando non viene premuto alcun pulsante, tutti i pin della colonna vengono mantenuti ALTI e tutti i pin della riga vengono mantenuti BASSI	
2. Quando si preme un pulsante, il pin della colonna viene portato a livello BASSO poiché la corrente dalla colonna ALTA scorre al pin della riga BASSO:	

3. Arduino ora sa in quale colonna si trova il pulsante, quindi deve solo trovare la riga in cui si trova. Lo fa impostando ogni pin di riga su HIGH e, allo stesso tempo, leggendo tutti i pin di colonna per rilevare quale pin di colonna torna su HIGH.	
4. Quando il pin della colonna torna in HIGH, Arduino ha trovato il pin della riga collegato al pulsante:	

Dal diagramma soprastante, è possibile vedere che la combinazione della riga 2 e della colonna 2 potrebbe significare solo che è stato premuto il pulsante numero 5.

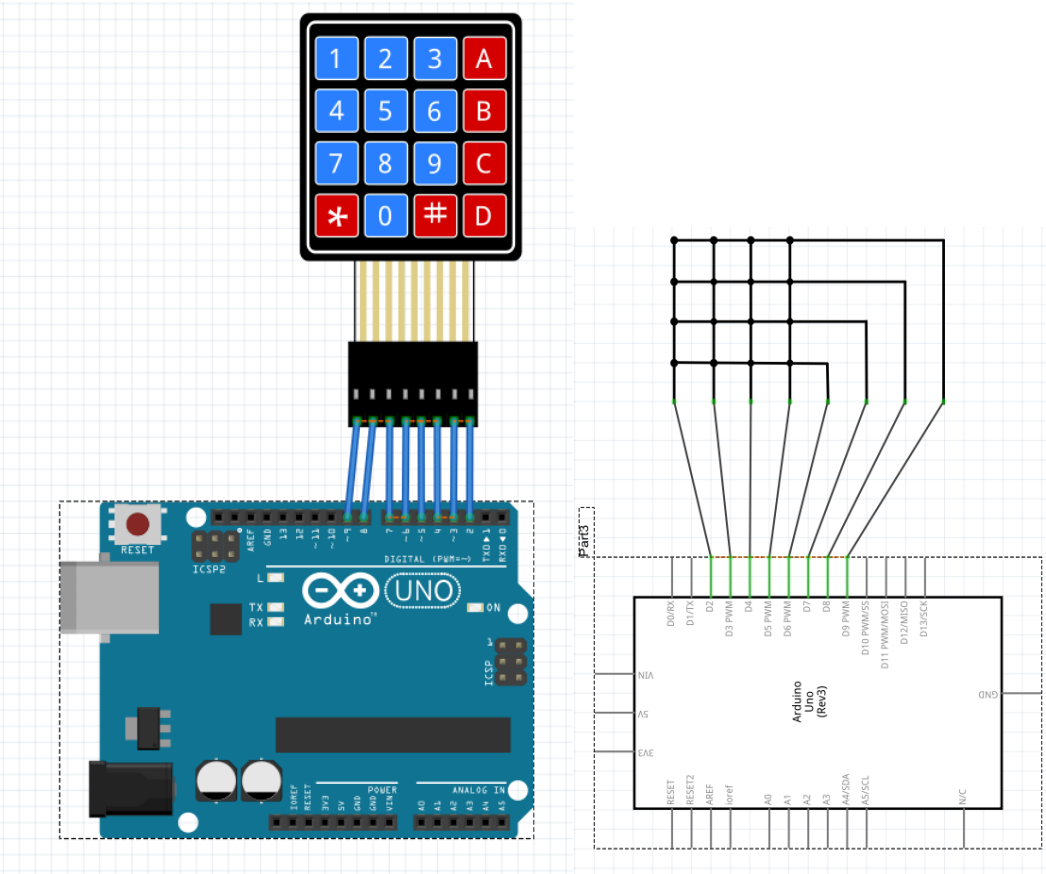
COLLEGARE LA TASTIERA AD ARDUINO

La disposizione dei pin della maggior parte delle tastiere a membrana sarà la seguente:



Circuito 1

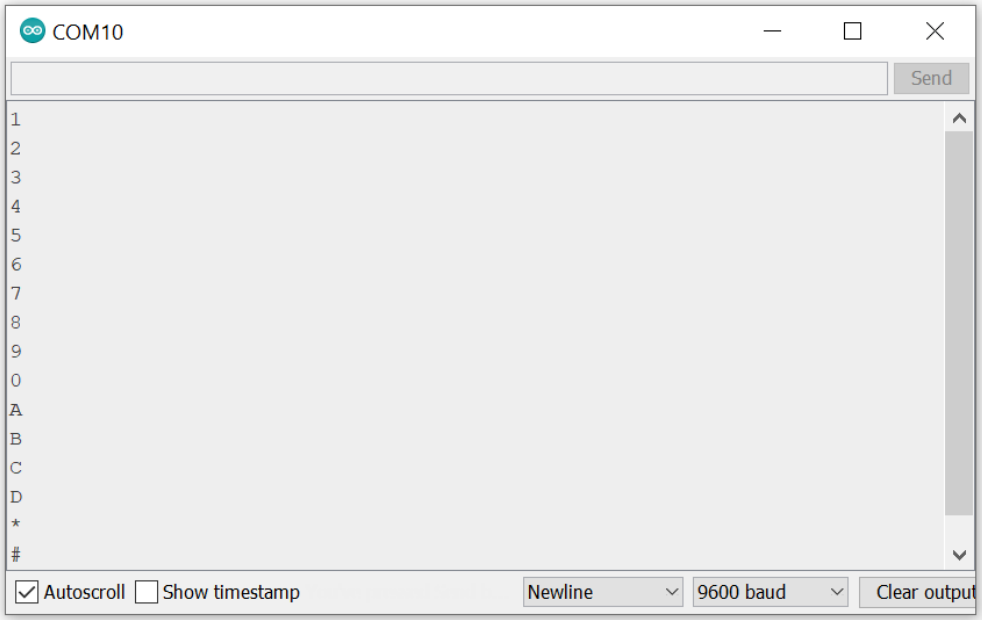
Titolo del circuito: Il monitor seriale visualizza il tasto premuto
Descrizione del circuito: Il programma ci mostrerà come visualizzare ogni pressione di tasto sul monitor seriale.



1-) Vista breadboard 2-) Vista schematica

```
/*    "Serial monitor display the pressed key"   */
#include <Keypad.h>
const byte ROWS = 4;
const byte COLS = 4;
char hexaKeys[ROWS][COLS] = {
  {'1', '2', '3', 'A'},
  {'4', '5', '6', 'B'},
  {'7', '8', '9', 'C'},
  {'*', '0', '#', 'D'}
};
byte rowPins[ROWS] = {9, 8, 7, 6};
byte colPins[COLS] = {5, 4, 3, 2};
Keypad customKeypad = Keypad(makeKeymap(hexaKeys), rowPins, colPins, ROWS, COLS);
void setup(){
  Serial.begin(9600);
}
void loop(){
```

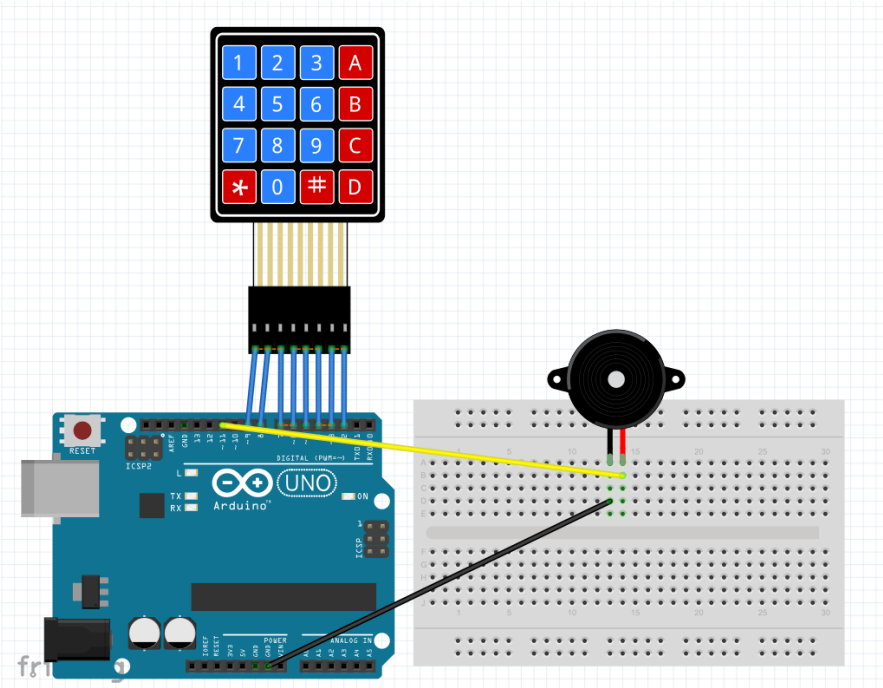
```
 char customKey = customKeypad.getKey();
 if (customKey){
   Serial.println(customKey);
 }
}
```



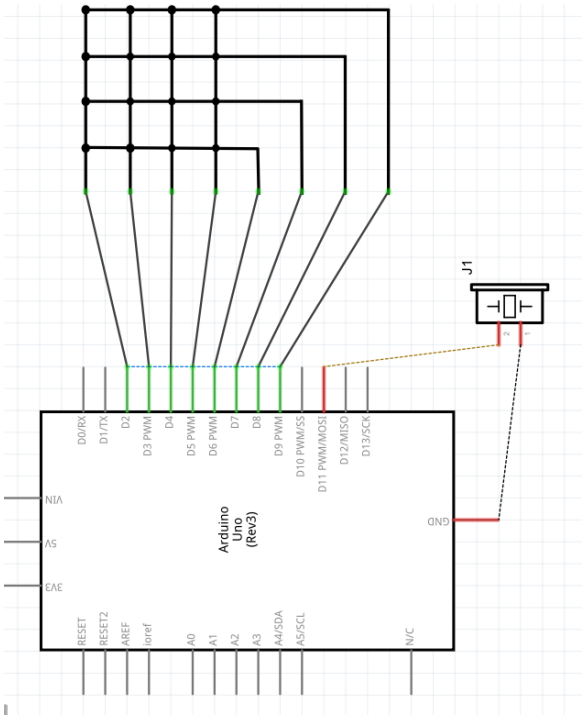
Circuito 2

Titolo del circuito: Segnale acustico del tastierino Arduino

Descrizione del circuito: Quando si preme un tasto sul tastierino, il cicalino piezoelettrico emette un segnale acustico



1-) Vista breadboard



2-) Vista schematica

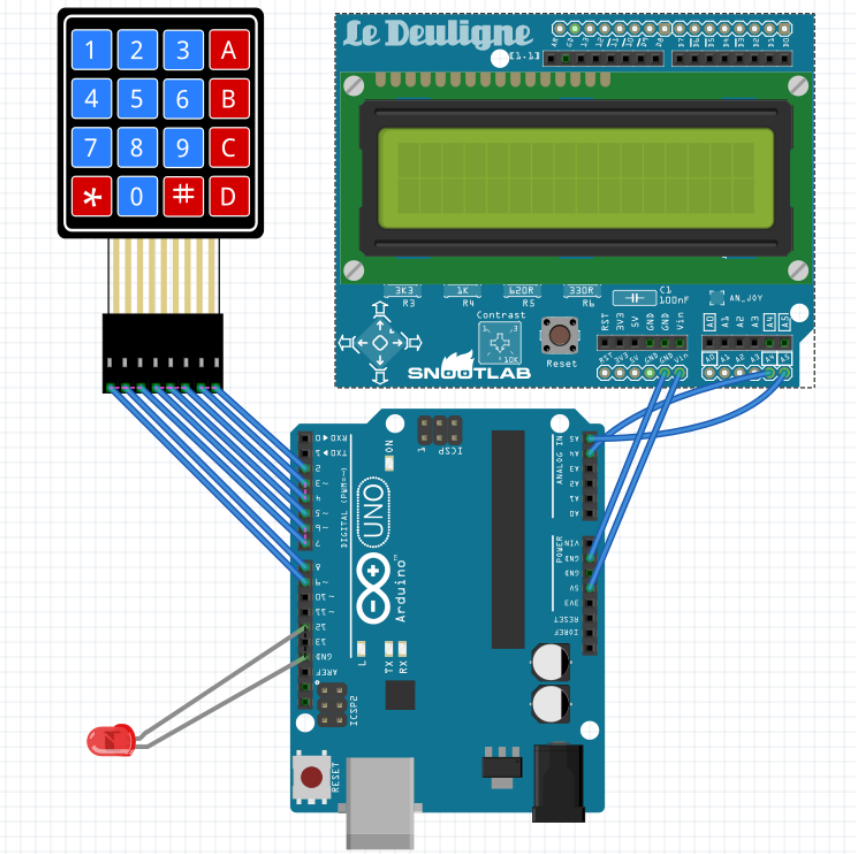
```
/*    "Arduino Keypad Beep"   */
#include <Keypad.h>
#include <ezBuzzer.h>
const int BUZZER_PIN = 11;
const int ROW_NUM    = 4; // four rows
const int COLUMN_NUM = 4; // four columns
char keys[ROW_NUM][COLUMN_NUM] = {
  {'1', '2', '3', 'A'},
  {'4', '5', '6', 'B'},
  {'7', '8', '9', 'C'},
  {'*', '0', '#', 'D'}
};
byte pin_rows[ROW_NUM] = {9, 8, 7, 6};      // connect to the row pinouts of the keypad
byte pin_column[COLUMN_NUM] = {5, 4, 3, 2}; // connect to the column pinouts of the
keypad
Keypad keypad = Keypad(makeKeymap(keys), pin_rows, pin_column, ROW_NUM,
COLUMN_NUM );
ezBuzzer buzzer(BUZZER_PIN); // create ezBuzzer object that attach to a pin;
void setup() {
  Serial.begin(9600);
}
void loop() {
```

```
 buzzer.loop(); // MUST call the buzzer.loop() function in loop()
 char key = keypad.getKey();
 if (key) {
   Serial.print(key); // prints key to serial monitor
   buzzer.beep(100);  // generates a 100ms beep
 }
}
```

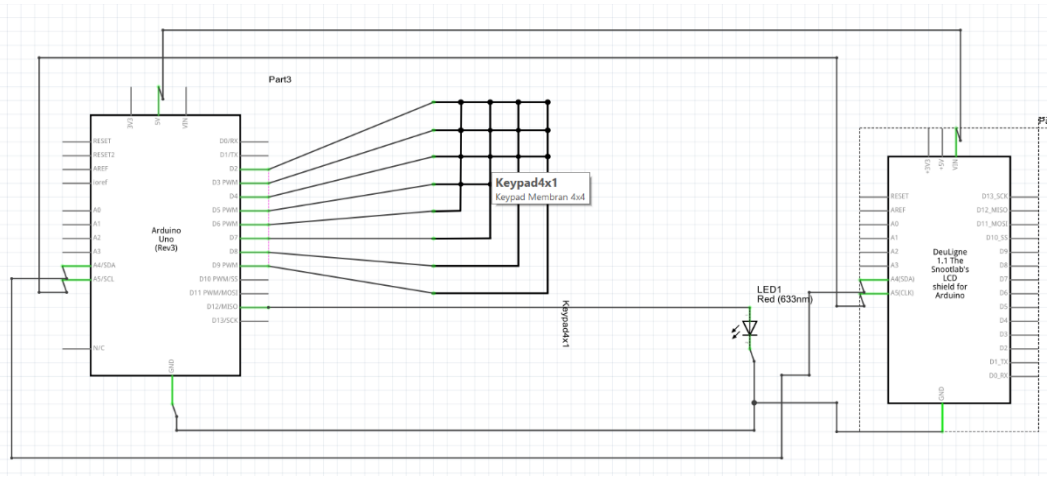
Circuito 3

Titolo del circuito: Codice di sblocco Arduino

Descrizione del circuito: Programma che imposta un tastierino numerico sull'Arduino. Un LED si accenderà quando si digita il codice corretto.



1-) Vista breadboard



2-) Vista schematica

/* Arduino Unlocking code */

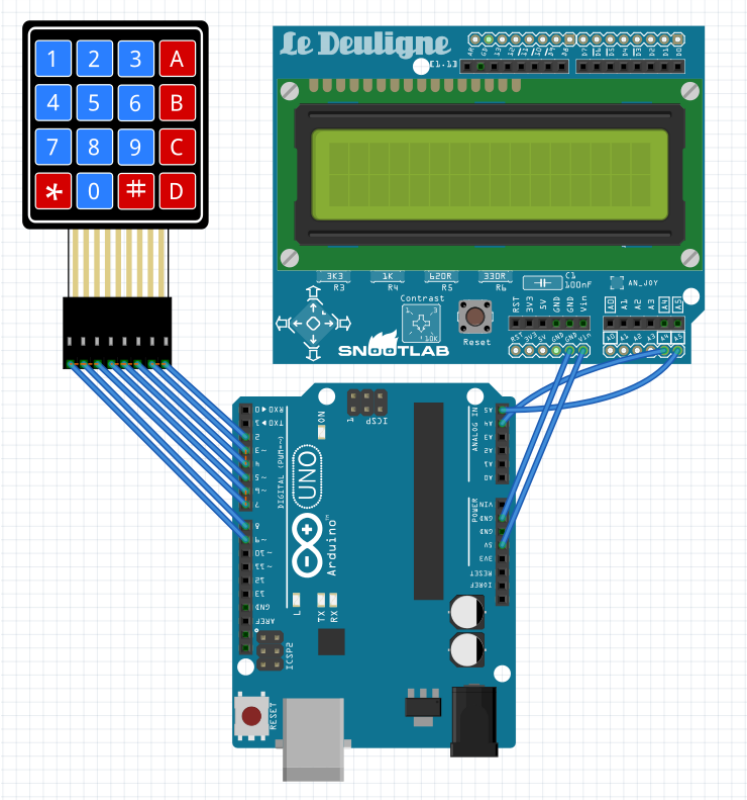
```
#include <Keypad.h> //Libraries you can download them via Arduino IDE
#include <Wire.h>
#include <LCD.h>
#include <LiquidCrystal_I2C.h>
#define Solenoid 12          //Actually the Gate of the transistor that controls the solenoid
                             //in my case I use a simple LED
#include <liquidcrystal_i2c.h></liquidcrystal_i2c.h></lcd.h></wire.h></keypad.h>
#define I2C_ADDR 0x27 // LCD i2c Adress and pins
#define BACKLIGHT_PIN 3
#define En_pin 2
#define Rw_pin 1
#define Rs_pin 0
#define D4_pin 4
#define D5_pin 5
#define D6_pin 6
#define D7_pin 7
LiquidCrystal_I2C
lcd(I2C_ADDR,En_pin,Rw_pin,Rs_pin,D4_pin,D5_pin,D6_pin,D7_pin);
const byte numRows= 4; //number of rows on the keypad
const byte numCols= 4; //number of columns on the keypad
int code = 1234;  //here is the code
int tot,i1,i2,i3,i4;
char c1,c2,c3,c4;
//keymap defines the key pressed according to the row and columns just as appears on the
keypad char keymap[numRows][numCols]=
{
{'1', '2', '3', 'A'},
```

```
{'4', '5', '6', 'B'},
{'7', '8', '9', 'C'},
{'*', '0', '#', 'D'}
};
//Code that shows the keypad connections to the arduino terminals
byte rowPins[numRows] = {9,8,7,6}; //Rows 0 to 3
byte colPins[numCols]= {5,4,3,2}; //Columns 0 to 3
//initializes an instance of the Keypad class
Keypad myKeypad= Keypad(makeKeymap(keymap), rowPins, colPins, numRows,
numCols);
void setup()
        {
         lcd.begin (16,2);
         lcd.setBacklightPin(BACKLIGHT_PIN,POSITIVE);
         lcd.setBacklight(HIGH);
         lcd.home ();
         lcd.print("ROMANIA DoorLock");
         lcd.setCursor(9, 1);
         lcd.print("Standby");
          pinMode(Solenoid,OUTPUT);
         delay(2000);
        }
void loop()
{
    char keypressed = myKeypad.getKey();  //The getKey fucntion keeps the program
runing, as long you didn't press "*" the whole thing bellow wouldn't be triggered
          if (keypressed == '*')          // and you can use the rest of you're code simply
            {
              lcd.clear();
              lcd.setCursor(0, 0);
              lcd.print("Enter Code");              //when the "*" key is pressed you can enter
the passcode
                  keypressed = myKeypad.waitForKey();  // here all programs are stopped until
you enter the four digits then it gets compared to the code above
                  if (keypressed != NO_KEY)
                    {
                     c1 = keypressed;
                     lcd.setCursor(0, 1);
                     lcd.print("*");
                     }
                  keypressed = myKeypad.waitForKey();
```

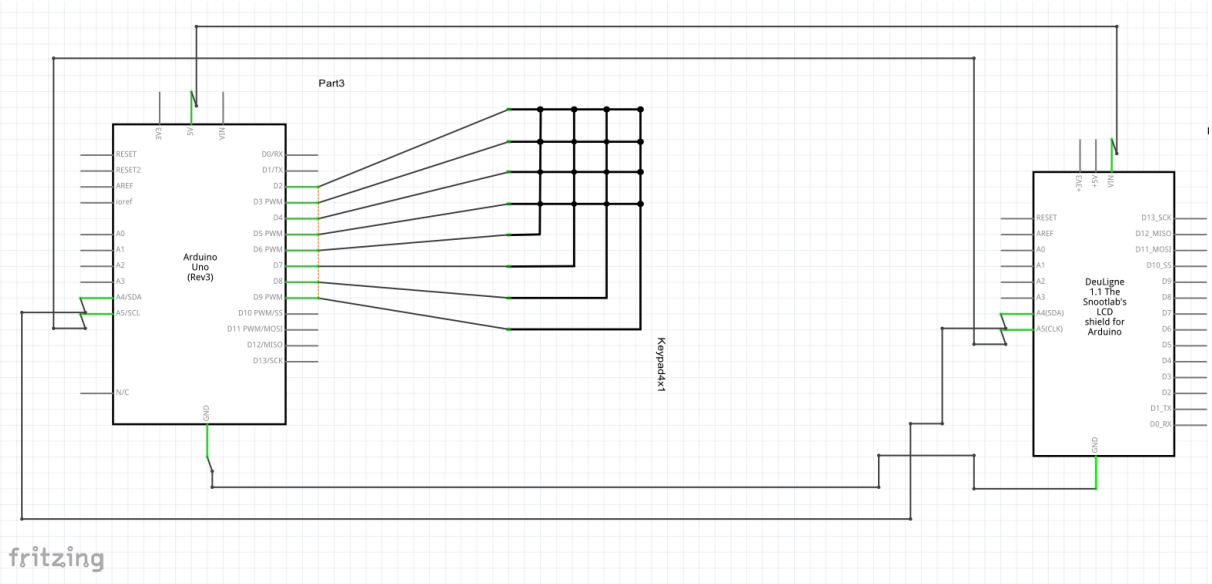
```
 if (keypressed != NO_KEY)
   {
    c2 = keypressed;
    lcd.setCursor(1, 1);
    lcd.print("*");
    }
  keypressed = myKeypad.waitForKey();
 if (keypressed != NO_KEY)
   {
    c3 = keypressed;
    lcd.setCursor(2, 1);
    lcd.print("*");
    }
   keypressed = myKeypad.waitForKey();
 if (keypressed != NO_KEY)
   {
    c4 = keypressed;
    lcd.setCursor(3, 1);
    lcd.print("*");
    }
  i1=(c1-48)*1000;        //the keys pressed are stored into chars I convert them to int then i did some multiplication to get the code as an int of xxxx
    i2=(c2-48)*100;
    i3=(c3-48)*10;
    i4=c4-48;
    tot=i1+i2+i3+i4;
  if (tot == code) //if the code is correct you trigger whatever you want here it just print a message on the screen
  {
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("Welcome");
    digitalWrite(Solenoid,HIGH);
delay(3000);
digitalWrite(Solenoid,LOW);
    lcd.setCursor(7, 1);
    lcd.print("ROMANIA DoorLock");

    delay(3000);
    lcd.clear();
    lcd.print("ROMANIA DoorLock");
```

```
          lcd.setCursor(9, 1);
          lcd.print("Standby");

         }
         else //if the code is wrong you get another thing
         {
          lcd.clear();
          lcd.setCursor(0, 0);
          lcd.print("WRONG CODE");
          delay(3000);
          lcd.clear();
          lcd.print("ROMANIA DoorLock");
          lcd.setCursor(9, 1);
          lcd.print("Standby");
          }
             }
}
```

Circuito 4
Titolo del circuito: Calcolatrice Arduino
Descrizione del circuito: Il programma esegue calcoli matematici di base



1-) Vista breadboard



2-) Vista schematica

/* Arduino calculator */

```
#include <Keypad.h>
#include <EEPROM.h>
#include <LCD.h>
#include <LiquidCrystal_I2C.h>
#define I2C_ADDR 0x27          //I2C adress
#define BACKLIGHT_PIN 3        // Declaring LCD Pins
#define En_pin 2
#define Rw_pin 1
#define Rs_pin 0
#define D4_pin 4
#define D5_pin 5
#define D6_pin 6
#define D7_pin 7
const byte ROWS = 4; // Four rows
const byte COLS = 4; // Four columns
// Define the Keymap
char keys[ROWS][COLS] = {
  {'1','2','3','A'},
  {'4','5','6','B'},
  {'7','8','9','C'},
  {'*','0','#','D'}
};
byte rowPins[ROWS] = {9,8,7,6}; //Rows 0 to 3
byte colPins[COLS]= {5,4,3,2}; //Columns 0 to 3
```

```
LiquidCrystal_I2C lcd(I2C_ADDR,En_pin,Rw_pin,Rs_pin,D4_pin,D5_pin,D6_pin,D7_pin);
Keypad kpd = Keypad( makeKeymap(keys), rowPins, colPins, ROWS, COLS ); //  Create the
Keypad
long Num1,Num2,Number;
 char key,action;
 boolean result = false;
 void setup()
        {
         lcd.begin (16,2);
         lcd.setBacklightPin(BACKLIGHT_PIN,POSITIVE);
           lcd.setBacklight(HIGH); //Lighting backlight
        lcd.print("Calculator Ready"); //Display a intro message
  lcd.setCursor(0, 1);   // set the cursor to column 0, line 1
  lcd.print("A=+ B=- C=* D=/"); //Display a intro message
   delay(6000); //Wait for display to show info
    lcd.clear(); //Then clean it
                    //          for(i=0 ; i<sizeof(code);i++){        //When you upload the code the first
time keep it commented
//            EEPROM.get(i, code[i]);            //Upload the code and change it to store it in the
EEPROM
//             }        //Then uncomment this for loop and reupload the code (It's done only once)
        }
void loop() {
  key = kpd.getKey(); //storing pressed key value in a char
if (key!=NO_KEY)
DetectButtons();
if (result==true)
CalculateResult();
DisplayResult();
}
void DetectButtons()
{
    lcd.clear(); //Then clean it
   if (key=='*') //If cancel Button is pressed
   {Serial.println ("Button Cancel"); Number=Num1=Num2=0; result=false;}
       if (key == '1') //If Button 1 is pressed
   {Serial.println ("Button 1");
   if (Number==0)
   Number=1;
   else
   Number = (Number*10) + 1; //Pressed twice
```

```
}
    if (key == '4') //If Button 4 is pressed
{Serial.println ("Button 4");
if (Number==0)
Number=4;
else
Number = (Number*10) + 4; //Pressed twice
}
    if (key == '7') //If Button 7 is pressed
{Serial.println ("Button 7");
if (Number==0)
Number=7;
else
Number = (Number*10) + 7; //Pressed twice
}
  if (key == '0')
{Serial.println ("Button 0"); //Button 0 is Pressed
if (Number==0)
Number=0;
else
Number = (Number*10) + 0; //Pressed twice
}
    if (key == '2') //Button 2 is Pressed
{Serial.println ("Button 2");
 if (Number==0)
Number=2;
else
Number = (Number*10) + 2; //Pressed twice
}
    if (key == '5')
{Serial.println ("Button 5");
 if (Number==0)
Number=5;
else
Number = (Number*10) + 5; //Pressed twice
}
    if (key == '8')
{Serial.println ("Button 8");
 if (Number==0)
Number=8;
else
```

```
Number = (Number*10) + 8; //Pressed twice
}
  if (key == '#')
{Serial.println ("Button Equal");
Num2=Number;
result = true;
}
    if (key == '3')
{Serial.println ("Button 3");
 if (Number==0)
Number=3;
else
Number = (Number*10) + 3; //Pressed twice
}
    if (key == '6')
{Serial.println ("Button 6");
if (Number==0)
Number=6;
else
Number = (Number*10) + 6; //Pressed twice
}
    if (key == '9')
{Serial.println ("Button 9");
if (Number==0)
Number=9;
else
Number = (Number*10) + 9; //Pressed twice
}
  if (key == 'A' || key == 'B' || key == 'C' || key == 'D') //Detecting Buttons on Column 4
{
Num1 = Number;
Number =0;
if (key == 'A')
{Serial.println ("Addition"); action = '+';}
 if (key == 'B')
{Serial.println ("Subtraction"); action = '-'; }
 if (key == 'C')
{Serial.println ("Multiplication"); action = '*';}
 if (key == 'D')
{Serial.println ("Division"); action = '/';}
delay(100);
```

```
  }
}
void CalculateResult()
{
  if (action=='+')
    Number = Num1+Num2;
  if (action=='-')
    Number = Num1-Num2;
  if (action=='*')
    Number = Num1*Num2;
  if (action=='/')
    Number = Num1/Num2;
}
void DisplayResult()
{
  lcd.setCursor(0, 0);   // set the cursor to column 0, line 1
  lcd.print(Num1); lcd.print(action); lcd.print(Num2);
    if (result==true)
  {lcd.print(" ="); lcd.print(Number);} //Display the result
    lcd.setCursor(0, 1);   // set the cursor to column 0, line 1
  lcd.print(Number); //Display the result
```

Co-funded by the
Erasmus+ Programme
of the European Union

Erasmus+ KA210-VET

Small-scale partnerships in vocational

education and training

Project Title: “Using Arduinos in Vocational Training”

Project Acronym: “UsingARDinVET”

Project No: “2023-1-RO01-KA210-VET-000156616”

Modulo Display e matrice a punti

Using Arduinos in Vocational Training
UsingARDinVET

IPSIA G.Giorgi - Potenza (Italy)

Dot Matrix Display Module

Nel contesto di Arduino, una matrice di punti è un display a LED costituito da una griglia bidimensionale di LED disposti in righe e colonne. Ogni LED può essere acceso o spento in modo indipendente, consentendo la creazione di pattern e animazioni complesse attraverso il controllo simultaneo di più LED. Una matrice di LED è utile per una vasta gamma di applicazioni. Una matrice 8x8, come quella mostrata in Figura 1, può essere utilizzata per visualizzare lettere o numeri. Se si dispone di più di questi moduli affiancati, è possibile creare un display con testo scorrevole.

Si noti che il modulo è progettato in modo che ci sia pochissimo spazio tra i LED e i bordi del modulo. Quando questi tipi di moduli a matrice vengono montati uno accanto all'altro, la distanza tra l'ultima colonna o riga di un modulo e la colonna o riga adiacente del modulo successivo è uguale alla distanza tra i LED situati al centro del modulo. Questo mantiene una spaziatura costante quando si utilizzano più moduli per creare display di grandi dimensioni.

Circuito 1. Creazione di un grafico a barre

Si consideri, ad esempio, un display a matrice di punti 10x1, composto da 10 LED singoli disposti su un'unica fila. È possibile collegare i LED come mostrato in Figura 2.
Lo schema seguente accende una serie di LED, il cui numero è proporzionale al valore di un potenziometro collegato a una porta di ingresso analogico.

```
const int analog Pin = A0 ;   // the pin at the potentiometer  is attached to
const int led Count = 10 ;     // the number of LEDs in the bar graph

// an array  of  pin numbers to which LEDs are  attached
```

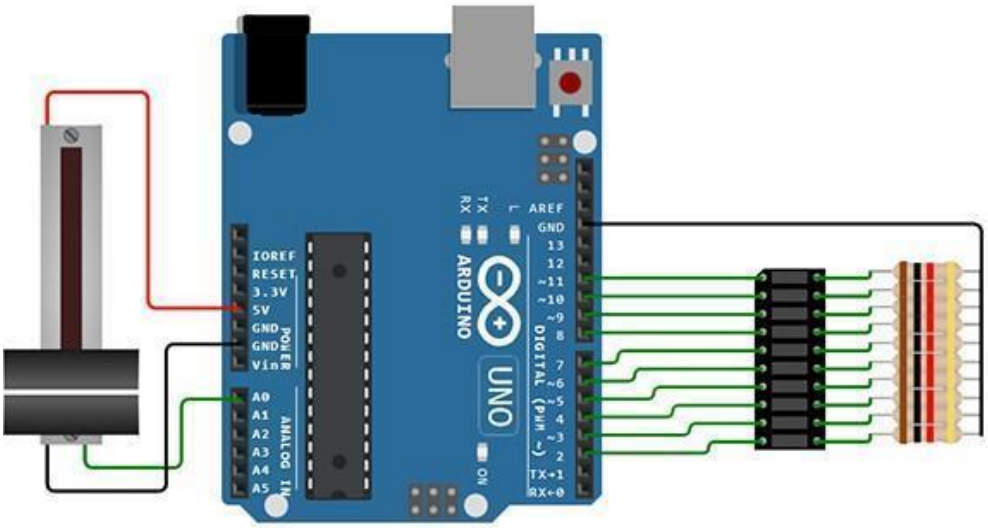


Figure 2: Bar Graph.

```
i n t  l e d Pi n s [ ] = { 2 ,  3 ,  4 ,  5 ,  6 ,  7 ,  8 ,  9 ,  10 ,  11 };

void setup ( ) {
//  l oop  over  the  pin  array and  s e t  them  a l l  to output :
f o r ( i n t  this Led = 0 ; this Led < led Count ;  this Led++) {
pinMode ( l e d Pi n s [ this Led ] , OUTPUT) ;
}
}

void loop ( ) {
// read the potentiometer :
i n t sensor Reading = analogRead ( analog Pin ) ;
// map the  r e s u l t  to a range from 0 to  the number o f LEDs :
 i n t  l e d L e v e l = map( sensor Reading , 0 , 1023 , 0 , led Count ) ;
```

```
// l oop over the LED array :
f o r  ( i n t  this Led = 0 ;  this Led < led Count ;  this Led++) {
// i f  the  array element ' s index  i s  l e s s  than  l e d Level ,
// turn  the  pin  for  t h is  element on :
 i f ( this Led < l ed Level ) {
d i g i t a l W r i t e ( l e d Pi ns [ this Led ] , HIGH ) ;
}
// turn  o f f  a l l  pins  h i g h e r than the  l e d L e v e l :
e l s e {
d i g i t a l W r i t e ( l e d Pi n s [ this Led ] , LOW) ;
}
}
}
```

I pin collegati ai LED sono contenuti nell'array ledPins. Per modificare il numero di LED, è possibile aggiungere (o rimuovere) elementi da questo array, ma assicurarsi che la variabile ledCount sia uguale al numero di elementi (che dovrebbe essere uguale al numero di pin). La funzione map di Arduino viene utilizzata per calcolare il numero di LED che devono essere accesi in proporzione al valore del sensore. Il codice esegue un ciclo su ogni LED, accendendolo se il valore proporzionale del sensore è maggiore del numero di LED. In un mondo ideale, un potenziometro al minimo restituirà zero, ma è probabile che si sposti nel mondo reale. Quando il sensore è al valore massimo, tutti i LED sono accesi. Se si nota che l'ultimo LED lampeggia quando il potenziometro è al massimo, provare ad abbassare il secondo argomento di map da 1023 a 1000 circa.

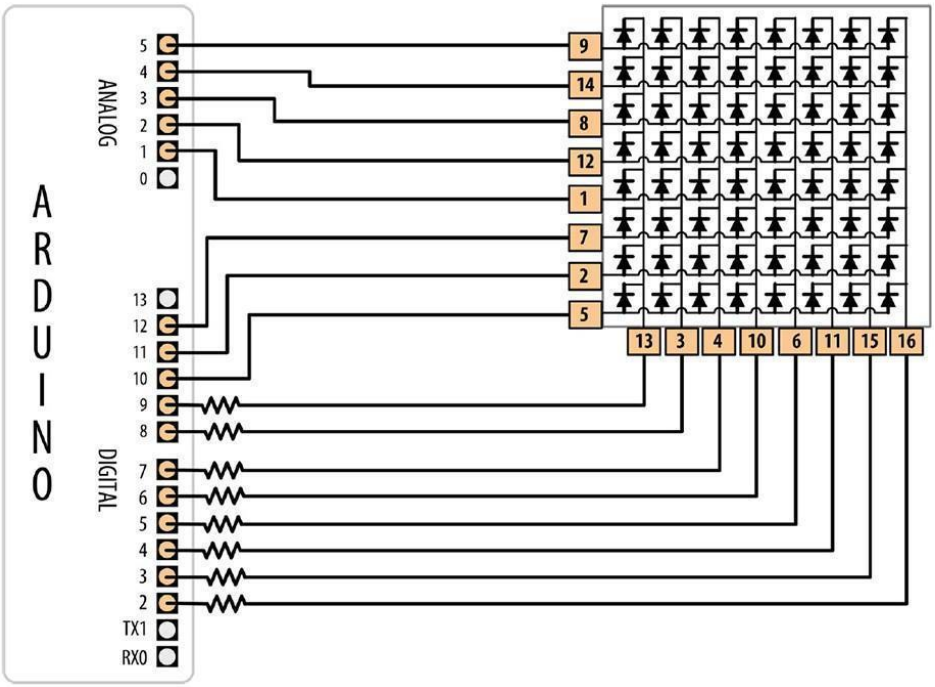


Figura 3: connessioni

Questo schema utilizza una matrice di 64 LED, con anodi collegati in file e catodi in colonne (come nel modello Jameco 2132349).

La Figura 3 mostra le connessioni (i display a LED bicolore potrebbero essere più facili da ottenere, ed è possibile pilotare solo uno dei colori se è sufficiente). Figura 3: Una matrice di LED collegata a 16 pin digitali.

```
const int columnPins [ ] = { 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9 };
const int rowPins [ ] = { 10 , 11 , 12 , A1 , A2 , A3 , A4 , A5 } ;
int pixel = 0 ;    // 0 to 63 LEDs in the matrix
int column Level = 0 ;       // pixel value converted into LED column
int row Level = 0 ; // pixel value converted into LED row

void setup ( ) {
for ( int i = 0 ; i < 8 ; i ++) { pinMode ( columnPins [ i ] , OUTPUT) ; pinMode ( rowPins [ i ]
, OUTPUT) ;
}
}

void loop ( ) {
pixel = pixel + 1 ;
if ( pixel > 63 ) pixel = 0 ;
column Level = pixel / 8 ;// map to the number of columns row Level = pixel % 8 ; // get
the fractional value
for ( int column = 0 ; column < 8 ; column++) { digitalWrite ( columnPins [ column ]
, LOW) ;
for ( int row = 0 ; row < 8 ; row++)
{ if ( column Level > column ) {
digitalWrite ( rowPins [ row ] , HIGH ) ;
} else if ( column Level == column && row Level >= row ) { digitalWrite ( rowPins
[ row ] , HIGH ) ;
} else {
// turn off all LEDs in this row
digitalWrite ( columnPins [ column ] , LOW) ;
}
delay Microseconds ( 3 0 0 ) ;
digitalWrite ( rowPins [ row ] , LOW) ;       // turn off LED
}
// disconnect this column from Ground
digitalWrite ( columnPins [ column ] , HIGH ) ;
}
}
```

Il valore del resistore deve essere scelto in modo da garantire che la corrente massima attraverso un pin non superi i 40 mA su Arduino Uno. Poiché la corrente per un massimo di otto LED può fluire attraverso ciascun pin di colonna, la corrente massima per ciascun LED deve essere un ottavo di 40 mA, ovvero 5 mA. Ogni LED in una tipica piccola matrice rossa ha una tensione diretta di circa 1,8 volt. Calcolando il resistore che genera 5 mA con una tensione diretta di 1,8 volt, si ottiene un valore di 680 Ω. Consultare la scheda tecnica per trovare la tensione diretta della matrice che si desidera utilizzare. Ogni colonna della matrice è collegata tramite il resistore in serie a un pin digitale. Quando il pin di colonna passa a livello basso e un pin di riga passa a livello alto, il LED corrispondente si accende. Per tutti i LED in cui il pin di colonna è a livello alto o il pin di riga è a livello basso, non scorrerà corrente attraverso il LED e il LED non si accenderà. Il ciclo for scansiona ogni riga e colonna e accende i LED in sequenza finché tutti i LED non sono accesi. Il ciclo inizia con la prima colonna e riga e incrementa il contatore di riga finché tutti i LED di quella riga non sono accesi; quindi passa alla colonna successiva e così via, accendendo un altro LED a ogni passaggio del ciclo fino a quando tutti i LED non sono accesi. Non è necessario accendere un'intera riga contemporaneamente. Il seguente schema accenderà un LED alla volta durante la sequenza:

```
const int columnPins [ ] = { 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9 };
const int rowPins [ ] = { 10 , 11 , 12 , A1 , A2 , A3 , A4 , A5 } ;
int pixel = 0 ;    // 0 to 63 LEDs in the matrix

void setup ( ) {
for ( int i = 0 ; i < 8 ; i ++) {
pinMode ( columnPins [ i ] , OUTPUT) ;  // make all the LED pins outputs pinMode ( rowPins [ i ] , OUTPUT) ;
digitalWrite ( columnPins [ i ] , HIGH ) ;
}
}

void loop ( ) {
pixel = pixel + 1 ;
if ( pixel > 63 ) pixel = 0 ;
int column = pixel / 8 ;  // map to the number of columns
int row = pixel % 8 ; // get the fractional value
digitalWrite ( columnPins [ column ] , LOW) ;      // Connect this column to GND
digitalWrite ( rowPins [ row ] , HIGH ) ;    // Take this row HIGH
delay ( 1 2 5 ) ;       // pause briefly
digitalWrite ( rowPins [ row ] , LOW) ;      // Take the row low
digitalWrite ( columnPins [ column ] , HIGH ) ;    // Disconnect the column from GND
}
```

Circuito 2.
Visualizzazione di immagini su una matrice di LED

Si desidera visualizzare una o più immagini su una matrice di LED, magari creando un effetto di animazione alternando rapidamente più immagini. Questa soluzione può utilizzare lo stesso cablaggio della figura 3. Lo schizzo crea l'effetto di un cuore che batte accendendo brevemente i LED disposti a forma di cuore. Un cuore piccolo seguito da un cuore più grande viene acceso per ogni battito cardiaco (le immagini sono simili alla figura 4):

byte big Heart [ ] = { B01100110 , B11111111 , B11111111 , B11111111 , B01111110 ,

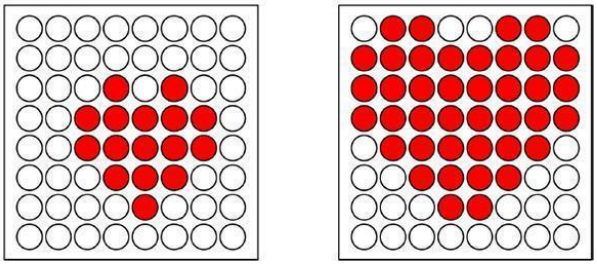


Figure 4: Le immagini dei due cuori riprodotta per ogni battito.

B00111100 , B00011000 , B00000000 } ;
byte small Heart [ ] = { B00000000 , B00000000 , B00010100 , B00111110 , B00111110 , B00011100 , B00001000 , B00000000 } ;
const i n t columnPins [ ] = { 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9 };
const i n t rowPins [ ] = { 10 , 11 , 12 , A1 , A2 , A3 , A4 , A5 } ;

```
void setup ( ) {
f o r ( i n t i = 0 ; i < 8 ; i ++) {
pinMode ( rowPins [ i ] , OUTPUT) ;        // make a l l the LED pins o utp uts pinMode ( columnPins [ i ] , OUTPUT) ;
d i g i t a l W r i t e ( columnPins [ i ] , HIGH ) ;    // disconnect column pins from Ground
}
}

void loop ( ) {
i n t pulse Delay = 800 ;       // m i l l i s e c o n d s to wait between b e ats
show ( small Heart, 8 0 ) ;    // show the small heart image fo r 80 ms
show ( big Heart, 16 0 ) ;     // followed by the big heart f o r 160 ms
delay ( pulse Delay ) ;        // show nothing between beats
}

// Show a frame o f an image s to re d in the array pointed to by the image
```

```
// parameter . The frame i s repeated f o r the given duration in millisecond s .
void show ( byte * image ),
unsigned long duration ) {

unsigned long s t a r t = m i l l i s ( ) ;          // b egin timing the animation
while ( s t a r t + duration > m i l l i s ( ) )   // loop u n t i l the duration has passed
{
f o r ( i n t row = 0 ; row < 8 ; row++) {
d i g i t a l W r i t e ( rowPins [ row ] , HIGH ) ;
// connect row to +5 volts
f o r ( i n t column = 0 ; column < 8 ; column++)
{bool p i x e l = bitRead ( image [ row ] , column ) ;
 i f ( p i x e l == 1 ) {
d i g i t a l W r i t e ( columnPins [ column ] , LOW) ;        // connect column to Gnd
}
delay Microseconds ( 3 0 0 ) ;          // a small delay f o r each LED d i g i t a l W r i t e (
columnPins [ column ] , HIGH ) ;    // disconnect column from Gnd
}
d i g i t a l W r i t e ( rowPins [ row ] , LOW) ;        // disconnect LEDs
}
}
}
```

Il valore scritto sul LED si basa sulle immagini memorizzate negli array bigHeart e smallHeart. Ogni elemento nell'array rappresenta un pixel (un singolo LED) e ogni riga dell'array rappresenta una riga nella matrice. Una riga è composta da otto bit rappresentati in formato binario (come indicato dalla B maiuscola all'inizio di ogni riga). Un bit con valore 1 indica che il LED corrispondente dovrebbe essere acceso; uno 0 significa 0. L'effetto animazione viene creato passando rapidamente da un array all'altro. La funzione loop attende un breve intervallo (800 ms) tra i battiti e quindi chiama la funzione show, prima con l'array smallHeart e poi con l'array bigHeart. La funzione show scorre ogni elemento in tutte le righe e colonne, accendendo il LED se il bit corrispondente è 1. La funzione bitRead viene utilizzata per determinare il valore di ciascun bit. Un breve ritardo di 300 microsecondi tra ciascun pixel consente all'occhio di avere tempo sufficiente per percepire il LED. La temporizzazione è scelta in modo da consentire a ogni immagine di ripetersi abbastanza velocemente (50 volte al secondo) da rendere impercettibile il battito delle palpebre.
Ecco una variante che modifica la frequenza cardiaca in base al valore di un sensore. È possibile testarla utilizzando una resistenza variabile collegata al pin di ingresso analogico 0. Utilizzare il cablaggio e il codice mostrati in precedenza, sostituendo però la funzione loop con questo codice:

```
void loop ( ) {
i n t sensor Value = analogRead ( A0 ) ;      // read the analog in value
 i n t pulse Rate =
```

```
map( sensor Value , 0 , 1023 , 40 , 2 4 0 ) ; //  convert  to  beats  /  minute
i n t pulse Delay = ( 60000 / pulse Rate ) ;   // m i l l i s e c o n d s to wait between beats
show ( small Heart , 8 0 ) ;// show  the  small  heart  image  f o r  100 ms
show ( big Heart , 16 0 ) ;     // f o l l o w e d by the b i g heart for 200 ms
delay ( pulse Delay ) ;          // show nothing between beats
}
```

Questa versione calcola il ritardo tra gli impulsi utilizzando la funzione mappa per convertire il valore del sensore in battiti al minuto.

Circuito 3.
Controllo di una matrice di LED tramite MAX72xx

Dovete controllare una matrice di LED 8x8 e volete ridurre al minimo il numero di pin Arduino necessari. Potete utilizzare un registro a scorrimento per ridurre il numero di pin necessari per controllare una matrice di LED. Questa soluzione utilizza il chip driver LED MAX7219 o MAX7221 per fornire questa funzionalità. Collegate Arduino, la matrice e il MAX72xx come mostrato nella figura 5.

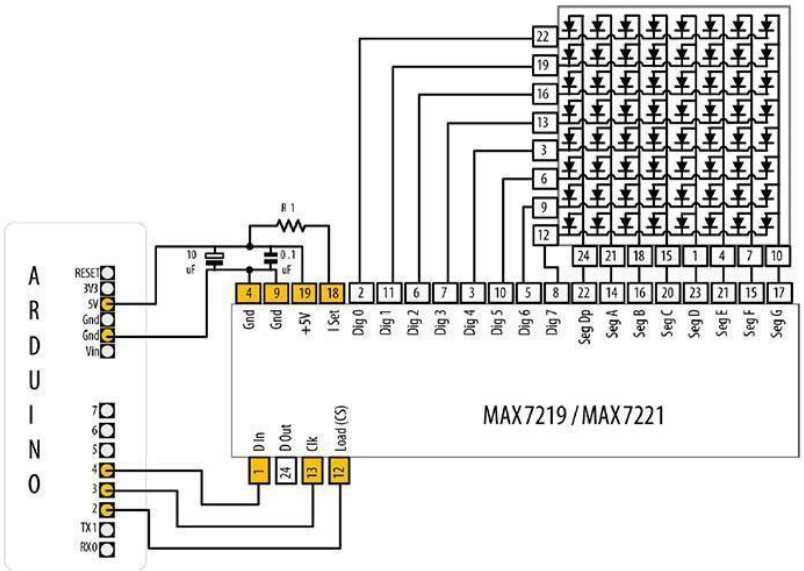


Figure 5: collegamenti tra scheda Arduino e matrice.

Questo sketch si basa sulla libreria MD_MAX72XX, che può visualizzare testo, disegnare oggetti sul display ed eseguire diverse trasformazioni. La libreria è disponibile nell'Arduino Library Manager.

```
#i n c l u d e <MD_MAX72xx. h>
// Pins to control 7219
```

```
#d e f i n e LOAD_PIN 2
#d e f i n e CLK_PIN 3
#d e f i n e DATA_PIN 4
// Configure the hardware
#d e f i n e MAX_DEVICES 1
#d e f i n e HARDWARE_TYPE MD_MAX72XX: :PAROLA_HW MD_MAX72XX mx =
MD_MAX72XX(HARDWARE_TYPE, DATA_PIN, CLK_PIN, LOAD_PIN,
MAX_DEVICES) ;

void  setup ( ) { mx. begin ( ) ;  }

void loop ( ) {
mx. c l e a r ( ) ;       //  Clear  the  d i s p l a y
// Draw rows and columns
f o r  ( i n t  r = 0 ;  r < 8 ;  r++)
{ f o r ( i n t c = 0 ; c < 8 ; c++) {
mx. setPoint ( r , c , true ) ;
// L ig ht each LED delay ( 5 0 ) ;
}
// Cycle through  available brightness levels
f o r ( i n t k = 0 ; k <= MAX_INTENSITY; k++) {
mx. c o n t r o l (MD_MAX72XX: : INTENSITY, k ) ; delay ( 1 0 0 ) ;
}
}
}
```

Una matrice viene creata passando il tipo di hardware, i numeri di pin per i pin dati, carico e clock, e anche il numero massimo di dispositivi (nel caso in cui si concatenano moduli). Il loop cancella il display, quindi utilizza il metodo setPoint per accendere i pixel. Dopo che lo sketch ha disegnato una riga, scorre ciclicamente le intensità di luminosità disponibili e passa alla riga successiva.
I numeri di pin mostrati qui si riferiscono ai LED verdi nella matrice bicolore 8x8, disponibile da Adafruit (codice articolo 458). Questo sketch funziona anche con una matrice monocolore, poiché utilizza solo uno dei due colori. Se noti che la matrice visualizza il testo al contrario o non nell'orientamento previsto, puoi provare a cambiare il tipo di hardware nella riga #define HARDWARE_TYPE MD_MAX72XX::PAROLA_HW da PAROLA_HW a GENERIC_HW, ICSTATION_HW o FC16_HW. Il resistore (contrassegnato con R1 nella figura 5) viene utilizzato per controllare la corrente massima che verrà utilizzata per pilotare un LED. Il datasheet del MAX72xx presenta una tabella che mostra un intervallo di valori. Il LED verde nella matrice LED mostrata nella figura 5 ha una tensione diretta di 2 volt e una corrente diretta di 20 mA. La tabella dei valori dei resistori (dal datasheet del MAX72xx) indica 28 kΩ, ma per aggiungere un piccolo margine di sicurezza, un resistore da 30 kΩ o 33 kΩ sarebbe una scelta

appropriata. I condensatori (0,1 µF e 10 µF) sono necessari per evitare picchi di rumore generati quando i LED vengono accesi e spenti.

RGB LEDs

Un diodo a emissione luminosa (LED) è un piccolo componente che si illumina quando è attraversato da corrente. I LED RGB (Figura 6) funzionano secondo lo stesso principio, ma contengono internamente tre LED (rosso, verde e blu) in grado di combinarsi per produrre praticamente qualsiasi colore.

Figura 6: RGB LEDs.

Il modello di colore RGB è un modo per rappresentare i colori miscelando la luce rossa, verde e blu (Figura 7). L'intensità di ciascun canale di colore determina il colore complessivo visualizzato. Combinando questi colori primari a diversi livelli si generano milioni di colori visibili all'occhio umano. Ad esempio, per creare una luce puramente blu, è necessario regolare il LED blu alla massima intensità e i LED verde e rosso alla minima. Tuttavia, per la luce bianca, tutti e tre i LED devono essere impostati alla massima intensità.

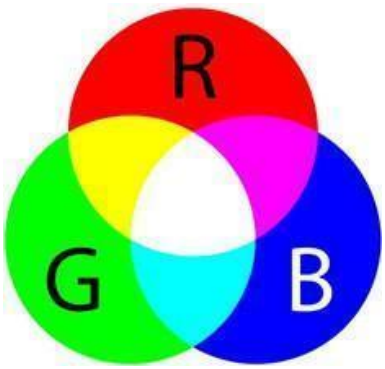


Figura 7: RGB Color Model

I LED RGB contengono tre LED al loro interno e, solitamente, questi tre LED condividono un anodo o un catodo comune. Questo permette di classificare i LED RGB come ad anodo comune o a catodo comune (Figura 8).

Circuito 5. Utilizzo di un LED RGB

Per ottenere colori diversi con un LED RGB, è necessario controllare la luminosità di ciascun LED interno. Questo può essere ottenuto utilizzando segnali PWM con un Arduino.
Nel circuito mostrato nella Figura 9, il catodo è collegato a GND e i tre anodi sono collegati a tre pin digitali sulla scheda Arduino tramite resistori da 220 Ohm. È importante che i pin utilizzati nel vostro Arduino possano emettere segnali PWM.
Il seguente codice farà cambiare alcuni colori al LED RGB:

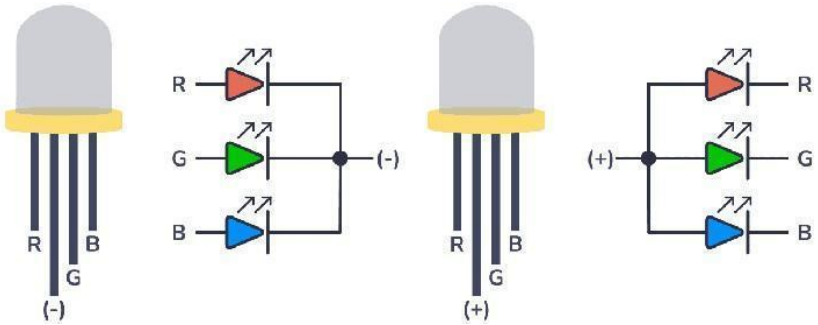


Figure 8: Types of RGB LEDs.

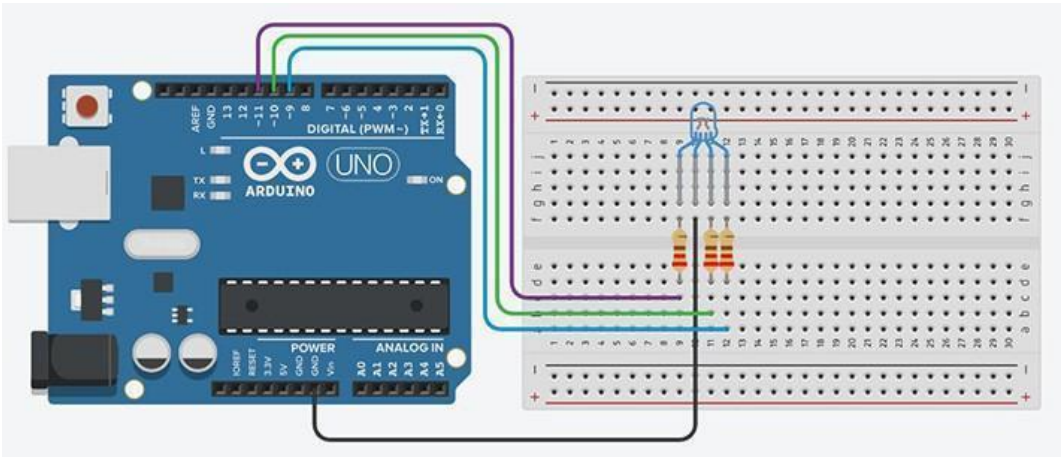


Figure 9: Connect an RGB LED to an Arduino.

```
// Declare the PWM LED pins
 i n t redLED = 9 ;
i n t greenLED = 10 ; i n t blueLED = 11 ;

void setup ( ) {
// Declare the pins f o r the LED as Output pinMode ( redLED , OUTPUT) ;
pinMode ( greenLED , OUTPUT) ; pinMode ( blueLED , OUTPUT) ;
}

// A simple  f u nc t io n  to  s e t  the  level for  each color from 0 to 255
```

```
void s tColor (
i n t  redValue ,
i n t greenValue ,
i n t blueValue ) {
analog Write ( redLED , red Value ) ; analog Write ( greenLED , green Value ) ; analog
Write ( blueLED , blue Value ) ;
}

void loop ( ) {
// Change a few colors

s e t Co l o r ( 255 , 0 , 0 ) ;  //  Red  Color
delay ( 10 0 0 ) ;
s e t Co l o r ( 0 , 255 , 0 ) ;  //  Green  Color
delay ( 10 0 0 ) ;
s e t Co l o r ( 0 , 0 , 2 5 5 ) ;//  Blue  Color
delay ( 10 0 0 ) ;
s e t Co l o r ( 255 , 255 , 0 ) ;         //  Yellow
delay ( 10 0 0 ) ;
s e t Co l o r ( 0 , 255 , 2 5 5 ) ;       //  Cyan
delay ( 10 0 0 ) ;
s e t Co l o r ( 255 , 0 , 2 5 5 ) ;       //  Magenta
delay ( 10 0 0 ) ;
s e t Co l o r ( 255 , 255 , 2 5 5 ) ;    //  White
delay ( 10 0 0 ) ;
}
```

Nella funzione di configurazione, i pin 9, 10 e 11 sono configurati come uscite. La funzione loop richiama ripetutamente la funzione setColor per visualizzare colori diversi a intervalli di un secondo. La funzione setColor accetta tre parametri (valori di rosso, verde e blu) che possono variare da 0 a 255. Questi valori vengono utilizzati nella funzione analogWrite, che emette segnali PWM per controllare l'intensità di ciascun colore del canale LED RGB.

Circuito 6. Controllo RGB LED con potenziometro

In questo esempio (figura 10), modificheremo il colore del LED RGB quando giriamo la manopola del potenziometro..

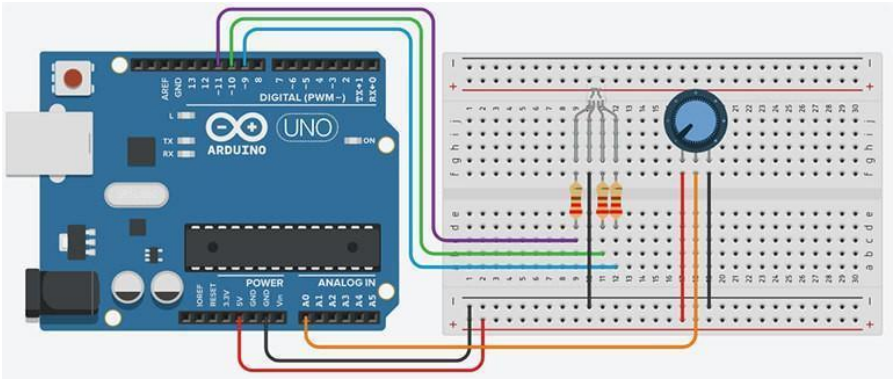


Figura 10: Circuito Arduino con LED RGB e potenziometro.

Scriviamo il codice per farlo.

#d e f i n e RGB_RED_PIN 11

#d e f i n e RGB_BLUE_PIN 10

#d e f i n e RGB_GREEN_PIN 9

#d e f i n e POTENTIOMETER_PIN A0

void setup ( )

{

pinMode (RGB_RED_PIN, OUTPUT) ; pinMode (RGB_BLUE_PIN, OUTPUT) ; pinMode (RGB_GREEN_PIN, OUTPUT) ;

}

void loop ( )

{

int potentiometer Value = analogRead (POTENTIOMETER_PIN);

i n t rgb Value = map(potentiometer Value, 0 , 1023 , 0 , 15 3 5) ;

i n t red ;

```
i n t blue; i n t green;


i f ( rgb Value < 256 ) { red = 255 ;
blue = rgb Value ; green = 0 ;
}
e l s e i f ( rgb Value < 512 ) { red = 511 = rgb Value ; blue = 255 ;
green = 0 ;
}
e l s e i f ( rgb Value < 768 ) { red = 0 ;
blue = 255 ;
green = rgb Value = 512 ;
}
e l s e i f ( rgb Value < 1024 ) { red = 0 ;
blue = 1023 = rgb Value ; green = 255 ;
}
e l s e i f ( rgb Value < 1280 ) { red = rgb Value = 1024 ; blue = 0 ;
green = 255 ;
}


e l s e {
red = 255 ;
blue = 0 ;
green = 1535 = rgb Value ;
}


analog Write (RGB_RED_PIN, red ) ; analog Write (RGB_BLUE_PIN, blue ) ; analog Write
(RGB_GREEN_PIN, green ) ;
}
```

Per prima cosa, creiamo un de ne per ogni pin che utilizzeremo. Uno per il potenziometro e uno per ogni colore del LED (scriviamo il codice come se stessimo controllando 3 LED diversi).

Nel void setup(), inizializziamo tutti i LED (in realtà, i 3 piedini del LED RGB) in modalità OUTPUT. Non c'è nulla da fare per il potenziometro, poiché un pin analogico è già in modalità input di default. Nel void loop(), leggiamo prima il valore del potenziometro con analogRead(). Questo ci restituisce un valore compreso tra 0 e 1023. Poiché vogliamo scegliere tra 1536 opzioni diverse, usiamo la funzione map() per trasformare questo valore dall'intervallo 0-1023 all'intervallo 0-1535.So, we have 6 different steps for changing the color. Also, you can note that the rst color and the last color are the same (red).

Per il 1° passaggio: impostiamo il rosso a 255 e aumentiamo il colore blu da 0 a 255, in base al valore rgbValue che abbiamo calcolato (nell'intervallo 0-1535).

Se rgbValue è maggiore di 255, passiamo al passaggio 2. Ora abbiamo valori da 256 a 511. Impostiamo il blu a 255 e poi diminuiamo il valore del rosso. Per farlo, dobbiamo sottrarre rgbValue al valore massimo per questo blocco, che è 511. Ad esempio, se inseriamo la struttura if con rgbValue = 400, avremo rosso = 511 - 400 = 111.

Per il passaggio numero 3, manteniamo il blu a 255 e questa volta aumentiamo il verde. Il valore rgbValue è ora compreso tra 512 e 767. Quindi, per partire da 0 e arrivare a 255, sottraiamo 512 a ogni valore ottenuto. I passaggi numero 4, 5 e 6 seguono la stessa logica dei passaggi precedenti.

Ora abbiamo 3 valori tra 0 e 255, memorizzati in 3 variabili diverse. Dopo il calcolo, usiamo analogWrite() su ciascuna gamba del RGB come se si trattasse di 3 LED diversi, con i rispettivi valori per rosso, blu e verde.

NeoPixel LED

I NeoPixel sono strisce LED RGB intelligenti i cui elementi possono essere controllati individualmente. Utilizzano driver WS2812, WS2811 o WS6812 e utilizzano un protocollo single-wire per controllare il colore del LED integrato. I LED sono integrati nel controller e sono venduti assemblati in vari formati: flessibili, a matrice, ad anello e come elementi singoli.

Ogni cella ha cinque pin (figura 11):

V_{CC}: 5V alimentazione per il circuito di controllo;

V_{DD}: 5V alimentazione per LED;

V_{SS}: terra;

D_{IN} : immissione dati;

D_{OUT} : uscita dati da collegare al LED successivo nella catena.

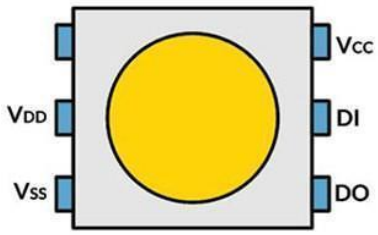


Figura 11: Pinout di un singolo modulo WS2812.

Nei prodotti NeoPixel, le connessioni sono semplificate e sono necessari solo tre pin:

- 5*V* : per alimentazione;
- *GND*: per la terra, da condividere con Arduino;
- D_{IN} : per la trasmissione dei dati.

Alimentare molti LED richiede molta potenza, quindi è necessario utilizzare un alimentatore o una batteria da 5 V in grado di fornire tutta la corrente richiesta dai LED. Tra 5 V e GND, è consigliabile inserire un condensatore elettrolitico da mille microfarad per fornire lo spunto necessario all'accensione dei vari LED. La linea dati deve essere collegata ad Arduino tramite un resistore da 470 Ω (figura 12).

Non c'è limite al numero di LED che un elemento NeoPixel può contenere. Le uniche limitazioni sono: la potenza assorbita dalla striscia aumenta per ogni LED aggiunto (ogni LED richiede un massimo di 60 mA);

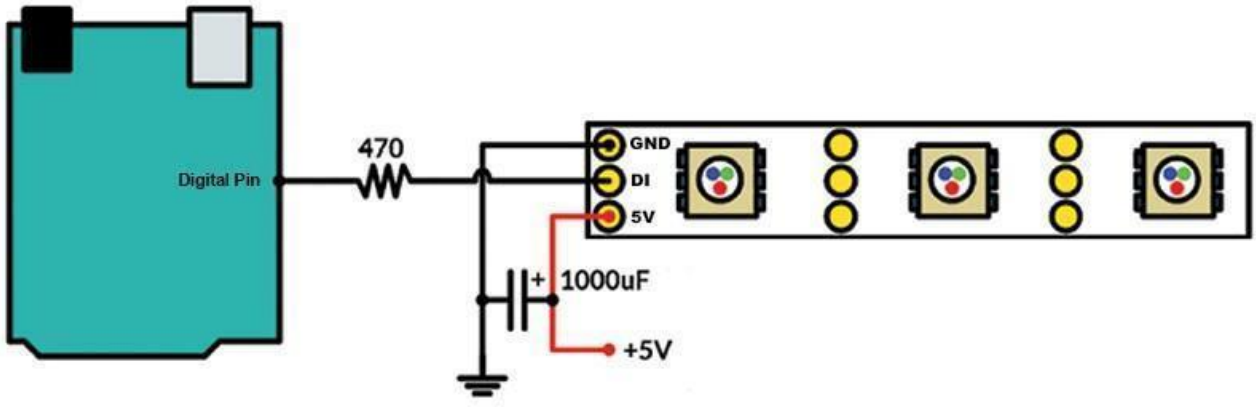


Figura 12: Interfacciamento tra NeoPixel e Arduino.

il tempo di risposta aumenta all'aumentare del numero di LED;

la memoria richiesta dal microcontrollore aumenta all'aumentare del numero di LED.

Esistono librerie per gestire la comunicazione con i singoli LED. La libreria di gestione che vedremo si chiama Adafruit NeoPixel di Adafruit e può essere installata tramite Arduino Library Manager. Per utilizzare la libreria, è necessario includerne la definizione all'inizio dello sketch:

```
#i n c l u d e <Adafruit_Neo Pixel . h>
```

L'inizializzazione dell'oggetto Adafruit_NeoPixel coinvolge una serie di parametri, quali il numero di LED da controllare, il pin utilizzato per la comunicazione e il modello del driver:

```
Adafruit_Neo Pixel p i x e l s = Adafruit_Neo Pixel (NUMPIXELS, PIN , NEO_RGB + NEO_KHZ800 ) ;
```

Il tipo di driver viene specificato combinando vari ag con il seguente significato:

- NEO_KHZ800: utilizza una velocità di trasmissione di 800 kHz;
- NEO_RGB: pixel collegati in modalità RGB.

I pixel vengono inizializzati con:

```
p i x e l s . begin ( ) ;
```

È possibile controllare il colore di ogni singolo pixel utilizzando il suo indice per impostare i valori RGB con:

```
p i x e l s . s e t P i x e l C o l o r ( num_pixel ,  0 , 15 0 , 0 ) ;
```

I colori vengono poi trasmessi con:

```
p i x e l s . show ( ) ;
```

Le funzioni della libreria includono anche una funzione per impostare la luminosità di tutti i LED:

```
s t r i p . s e t Brightness ( 1 0 0 ) ;
```

Circuito 7. Utilizzo della striscia NeoPixel

In questo esempio, impareremo come usare Arduino per controllare la striscia LED RGB NeoPixel e come utilizzare la libreria NeoPixel di Adafruit per configurare i NeoPixel. La Figura 13 mostra un esempio molto semplice di collegamento della striscia LED NeoPixel alla scheda Arduino. Per collegare una striscia di LED NeoPixel a una scheda Arduino, collega tre fili:

- Alimentazione (+5V) goes to the plus of a power source.
- Ground (GND) goes to power source ground and should be additionally connected to the board ground if powered from a separate power source.
- Data input (DIN) va al vantaggio di una fonte di alimentazione

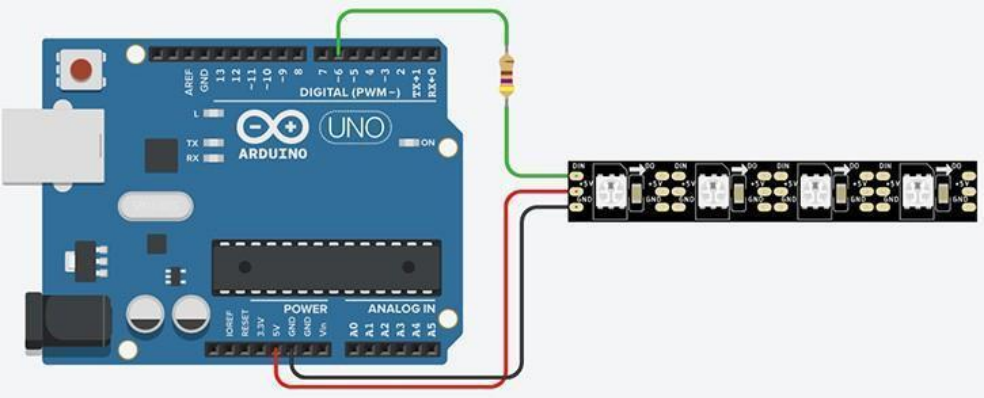


Figura 13: Collegamento di una striscia NeoPixel ad Arduino

La libreria Adafruit_NeoPixel permette di accendere facilmente un LED specifico con una certa intensità e colore, oppure di accenderlo. Ogni LED può essere controllato individualmente.

Qui abbiamo quattro LED, il primo dei quali è numerato 0.

Ad esempio, per accenderlo di rosso, si usa il comando: strip.setPixelColor (0, 255, 0, 0); Tuttavia, il LED risponderà a questo comando solo se seguito da strip.show().

#i n c l u d e <Adafruit_Neo Pixel . h>

#d e f i n e LED_PIN 6

#d e f i n e LED_COUNT 4

Adafruit_Neo Pixel s t r i p (LED_COUNT, LED_PIN, NEO_GRB + NEO_KHZ800) ;

void setup ( ) {

s t r i p . begin ( ) ; s t r i p . c l e a r ( ) ; s t r i p . show ( ) ;

}

void loop ( ) {

```
        s t r i p . s e t Brightness ( 5 0 ) ;
        s t r i p . s e t P i x e l C o l o r ( 0 , 255 , 0 , 0 ) ; s t r i p . show ( ) ;
        delay ( 10 0 0 ) ;
s t r i p . s e tPi x e lCo l o r (0 , 0 , 0 , 0 );
s t r i p . s e tPi x e lCo l o r ( 1 , 0 , 255 , 0 );
        s t r i p . show ();
        delay ( 10 0 0 );
s t r i p . s e tPi x e lCo l o r ( 1 , 0 , 0 , 0 ) ;
s t r i p . s e tPi x e lCo l o r (2 , 0 , 0 , 2 5 5 );
        s t r i p . show ();
        delay ( 10 0 0 );
s t r i p . s e tPi x e lCo l o r (2 , 0, 0, 0);
s t r i p . s e tPi x e lCo l o r (3 , 255, 255, 255);
        st r i p .
        show ( ) ;
delay ( 10 0 0 ) ;
st r i p . s e t P i x e lCo l o r ( 3 , 0 , 0 , 0 ) ;
}
```

Circuit 8. Illuminare un anello NeoPixel

L'anello NeoPixel è una disposizione circolare di LED RGB indirizzabili individualmente, che consente un'ampia gamma di combinazioni di colori e luminosità.

Nel circuito mostrato nella Figura 14, l'alimentatore è collegato al pin 5V, il pin GND è collegato alla terra del circuito e il pin DIN è collegato a un pin digitale sulla scheda Arduino.

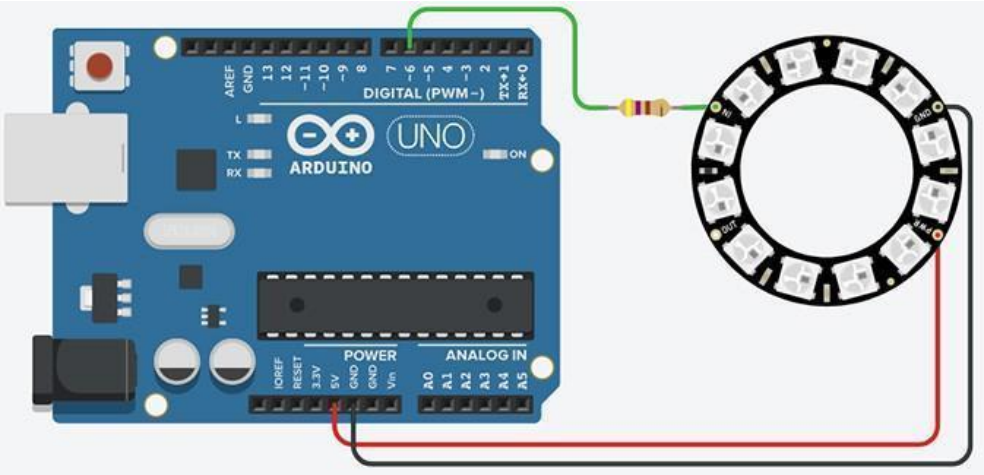


Figura 14: Interfacciamento dell'anello Neopixel con Arduino.

Come mostrato nel codice seguente, indichiamo il pin a cui è collegato l'ingresso dati NeoPixel e quanti LED sono presenti nel nostro anello. Il passo successivo è dichiarare effettivamente il nostro oggetto NeoPixel, che chiameremo "anello". Nella nostra funzione setup(), chiamiamo la funzione begin() su quell'anello. Quindi, chiamiamo show() per cancellare tutti i LED e, infine, impostiamo la luminosità con setBrightness(), che chiamiamo una volta all'inizio per indicare al nostro NeoPixel quale sarà la luminosità massima.

Ora, iniziamo con un ciclo for che va da 0 al numero di LED presenti nel nostro anello. Quindi, impostiamo il colore di ogni singolo LED. La funzione setPixelColor() accetta gli argomenti, in ordine: numLED, rosso, verde e blu. Inseriremo i come numLED in modo che il ciclo for li esegua tutti. Per il colore, utilizziamo valori casuali per rosso, verde e blu. In seguito, chiamiamo ring.show() per aggiornare effettivamente il colore nell'anello. Infine, aggiungiamo un ritardo di 50 millisecondi dopo l'applicazione di ciascun colore per creare un effetto di animazione di caricamento.

Next, we take the same loop and reverse it. To do this, we start from the last LED and count backwards to 0. Then, we set the pixel color to 0, 0, 0, which means no color, and add those same two lines..

```
#i n c l u d e <Adafruit_Neo Pixel . h>

#d e f i n e LED_PIN  6
#d e f i n e LED_COUNT 16

Adafruit_Neo Pixel r i n g (LED_COUNT, LED_PIN, NEO_RGB + NEO_KHZ800 ) ;

void setup ( )
{
 r i n g . begin ( ) ;
r i n g . show ( ) ;
r i n g . s e t B r i g h tn e s s ( 5 0 ) ;
}
void loop ( ) {
f o r ( i n t  i = 0 ;  i < r i n g . numPixels ( ) ;  i ++){
r i n g . s e t P i x e l C o l o r ( i , random ( 2 5 5 ) , random ( 2 5 5 ) , random ( 2 5 5 ) ) ;
r i n g . show ( ) ;
delay ( 5 0 ) ;
}
f o r ( i n t  i = r i n g . numPixels () = 1;  i >= 0 ;  i ==)
{ r i n g . s e t P i x e l C o l o r ( i , 0 , 0 , 0 ) ;
r i n g . show ( ) ; delay ( 5 0 ) ;
}
}
```

Circuito 9. Controllo di un anello NeoPixel

Questo sketch utilizza la libreria Neopixels di Adafruit (installata tramite Arduino Library Manager) per modificare il colore dei LED in base alle letture di un pin analogico. La Figura 15 mostra la connessione tra un anello NeoPixel e un potenziometro per il controllo del colore:

#i n c l u d e <Adafruit_Neo Pixel . h>

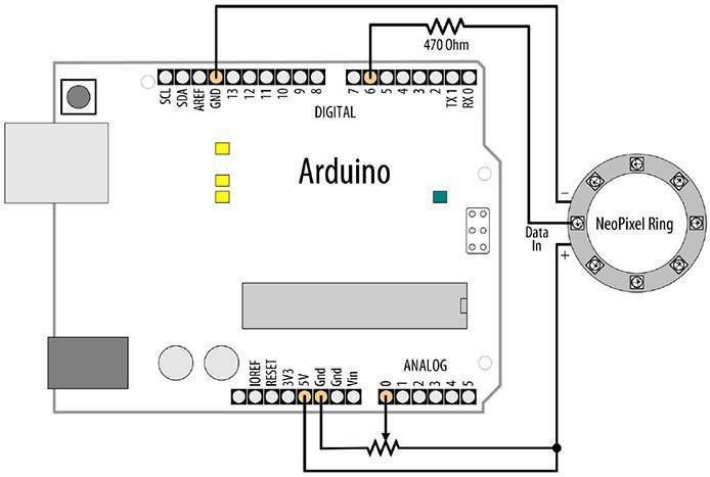


Figura 15: Collegamento di un anello NeoPixel.

```
const i n t sensor Pin = A0 ;   // analog pin for sensor
const i n t l ed Pin = 6 ;          // the pin the LED s t r i p i s connected to
const i n t count = 8 ;  // how many LEDs in the s t r i p
// d e c l are LED s t r i p
Adafruit_Neo Pixel l e d s = Adafruit_Neo Pixel ( count , led Pin , NEO_GRB + NEO_KHZ800
) ;

void setup ( ) {
l e d s . begin ( ) ;       // i n i t i a l i z e LED s t r i p f o r ( i n t i = 0 ; i < count ; i ++) {
l e d s . s e t P i x e l C o l o r ( i , l e d s . Color ( 0 , 0 , 0 ) ) ;          // turn each LED o f f
}
l e d s . show ( ) ;       // r e f r e s h the s t r i p with the new pixels values ( a l l o f f )
}
```

```
void loop ( ) {
s t a t i c unsigned i n t l a s t_r e a di n g = = 1;
 i n t reading = analogRead ( sensor Pin ) ;
i f ( reading != l a s t_r ea d i n g ) {   //  I f  the  v al ue  has  changed
// Map the analog reading  to  the  c o l o r range of  the Neo Pixel
unsigned i n t mappedSensorReading = map( reading , 0 , 1023 , 0 , 6 5 5 3 5 ) ;
// Update the  p i x e l s  with a  s l i g h t  d e l ay  to  c re ate a sweeping  e f f e c t
f o r ( i n t i = 0 ; i < count ; i ++) {
l e d s . s e t P i x e l C o l o r (i , l e d s . gamma32 ( l e d s . ColorHSV ( mappedSensorReading
, 255 , 1 2 8 ) ) ) ;
 l e d s . show ( ) ;
delay ( 2 5 ) ;
}
l a s t_r e a d i n g = reading ;
}
}
```

Si specifica il numero di LED nella striscia, il pin Arduino a cui è collegata la linea dati ledPin e il tipo di striscia LED utilizzata (in questo caso: NEO_GRB+NEO_KHZ800). Per impostare il colore di un singolo LED si utilizza il metodo led.setPixelColor. È necessario specificare il numero del LED (a partire da 0 per il primo) e il colore desiderato. Per trasferire i dati ai LED è necessario chiamare led.show. È possibile modificare i valori di più LED prima di chiamare led.show per modificarli contemporaneamente. I valori non modificati manterranno le impostazioni precedenti. Quando si crea l'oggetto Adafruit_NeoPixel, tutti i valori vengono inizializzati a 0.

La libreria NeoPixel include una funzione specifica per convertire una tonalità in un valore RGB: ColorHSV. Il primo argomento è la tonalità, il secondo la saturazione del colore e il terzo la luminosità. La funzione gamma32 esegue una conversione sull'output di ColorHSV per compensare il modo in cui i computer rappresentano i colori e il modo in cui gli esseri umani li percepiscono.

Co-funded by the Erasmus+ Programme of the European Union

Erasmus+ KA210-VET

Partnership su piccola scala nell'istruzione e

nella formazione professionale

Titolo progetto: “Using Arduinos in Vocational Training”

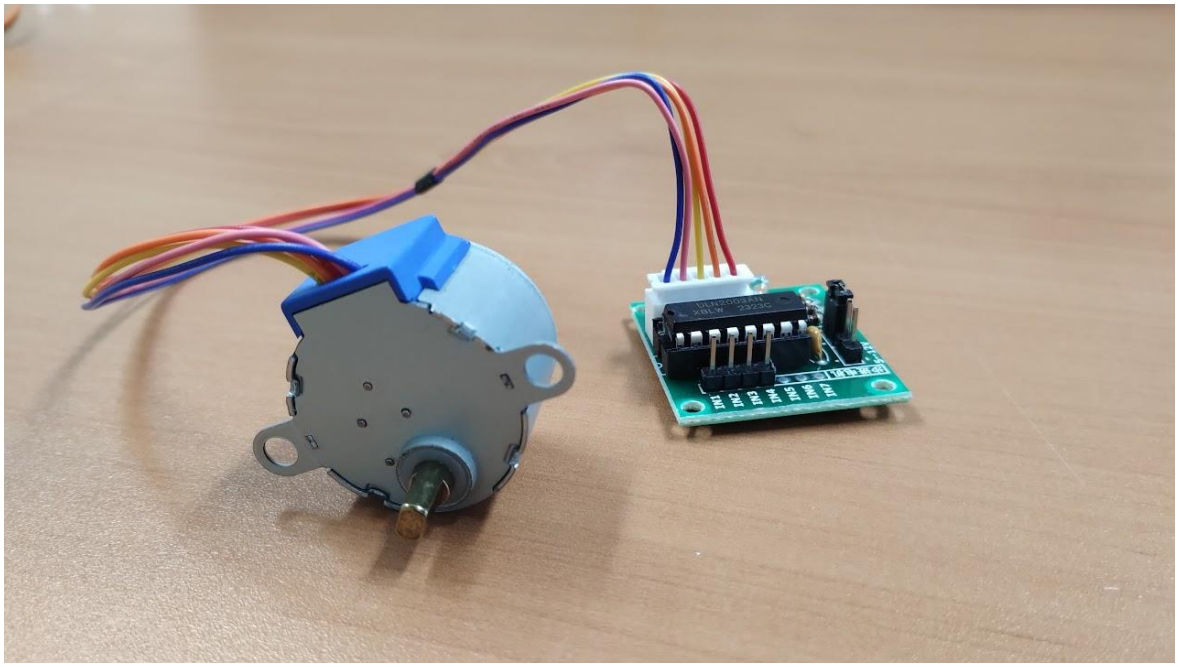
Acronimo del progetto: “UsingARDinVET”

Progetto No: “2023-1-RO01-KA210-VET-000156616”

Modulo motori e kit di addestramento

INTRODUZIONE ad ARDUINOS

(Modulo motori e kit di addestramento)



MOTORI CC

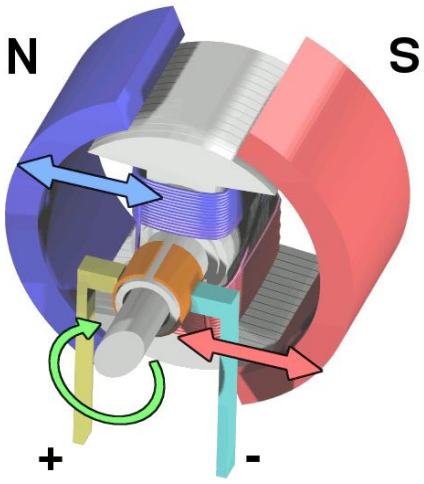
Panoramica

Un motore a corrente continua (CC) è un tipo di motore elettrico che funziona a corrente continua. È uno dei tipi di motore più utilizzati grazie alla sua semplicità, affidabilità e capacità di fornire una rotazione continua. I motori a corrente continua sono comunemente presenti in elettrodomestici, giocattoli, utensili e vari altri dispositivi che richiedono un movimento meccanico. In questo documento, spiegheremo il principio di funzionamento di base dei motori a corrente continua, i loro componenti e il loro funzionamento.

In questo documento spiegheremo il principio di funzionamento di base dei motori a corrente continua, i loro componenti e il loro funzionamento.

Principio di base del funzionamento del motore a corrente continua

Un motore a corrente continua funziona secondo il principio dell'induzione elettromagnetica, secondo il quale quando un conduttore percorso da corrente viene immerso in un campo magnetico, subisce una forza che lo fa muovere. Questa forza è nota come la forza di Lorentz.



Principio chiave:

Una corrente che scorre attraverso una bobina di filo genera un campo magnetico attorno alla bobina.

La bobina è immersa in un campo magnetico esterno (solitamente creato da magneti permanenti o elettromagneti).

L'interazione tra il campo magnetico della bobina e il campo magnetico esterno genera una forza che fa ruotare la bobina.

Questo moto rotatorio viene trasferito all'albero motore, generando un movimento meccanico.

Fattori che influenzano il funzionamento del motore CC

Diversi fattori possono influenzare le prestazioni di un motore a corrente continua:

- **Tensione**: la velocità e la coppia di un motore a corrente continua sono direttamente correlate alla tensione applicata. Una tensione più elevata generalmente si traduce in velocità e coppia maggiori.
- **Velocità**: aumentando la tensione applicata aumenta la velocità del motore perché aumenta la corrente attraverso gli avvolgimenti del rotore, producendo un campo magnetico più forte e una forza maggiore.
- **Coppia**: la coppia, o forza di rotazione, è proporzionale alla corrente. Una corrente più elevata significa una coppia più elevata.
- **Corrente**: la corrente determina la coppia che il motore può generare. Una corrente più elevata aumenta la coppia, ma può anche causare surriscaldamento se il motore non è adeguatamente raffreddato.
- **Resistenza**: la resistenza degli avvolgimenti del rotore influenza il flusso di corrente. Una resistenza più elevata limita la quantità di corrente che può fluire, riducendo la coppia del

motore. D'altra parte, una resistenza più bassa consente il flusso di una maggiore corrente, aumentando la coppia e la velocità.

- **Intensità del campo magnetico**: anche l'intensità del campo magnetico dello statore influisce sulle prestazioni del motore. Campi magnetici più intensi determinano una maggiore interazione tra statore e rotore, producendo più forza e quindi prestazioni più elevate.

Applicazioni

I motori elettrici a corrente continua (CC) vengono utilizzati quando è necessaria una rotazione continua e fluida. I normali motori a CC sono controllati in tensione, consentendo di decidere la velocità e la coppia che il motore produrrà. I normali motori a CC non possono essere utilizzati quando è necessario un controllo preciso della velocità o della posizione del motore. In questi casi, si scelgono solitamente motori passo-passo o servomotori.

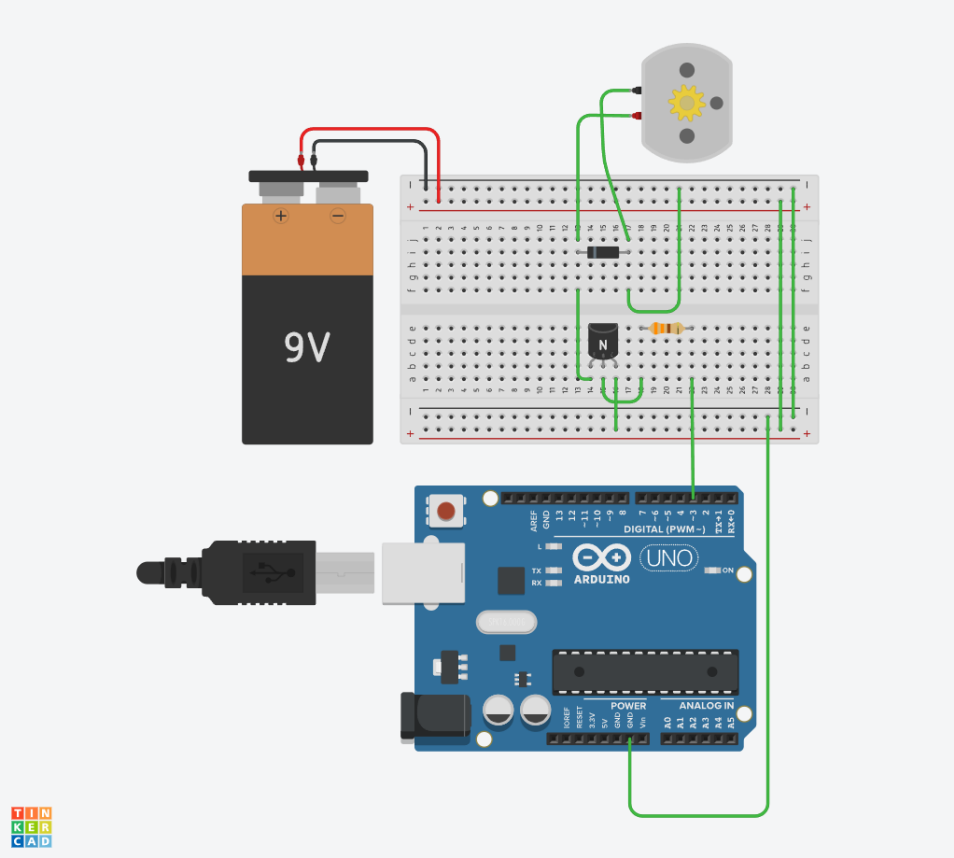
Requisiti di alimentazione

La maggior parte dei motori a corrente continua disponibili consumano una corrente superiore alla corrente massima erogabile da Arduino e può danneggiare le uscite digitali di Arduino. Pertanto, è necessario alimentare il motore esternamente e controllarlo tramite i pin di uscita di Arduino. Il modo più comune per controllare i motori a corrente continua con Arduino è utilizzare un controller per motori o costruire un controller personalizzato utilizzando un transistor.

Circuito 1

Titolo del circuito: pilotaggio di un motore a corrente continua con un transistor.

Descrizione del circuito: questo circuito utilizza un transistor PNP 2N2222 per controllare l'alimentazione a 9V tramite Arduino.



Circuito 1

Per costruirlo avrai bisogno di:

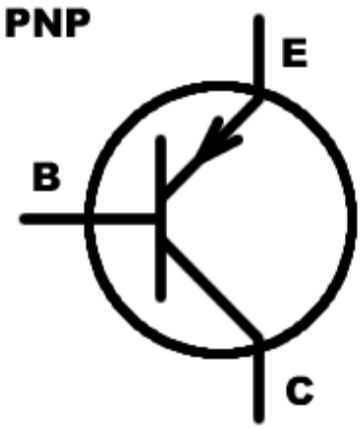
• 1 motore a corrente continua

• 1 transistor PNP 2N2222

• 1 resistore da 330Ω

• 1 diodo Zener

• 1 batteria da 9V



I transistor PNP possono essere utilizzati come interruttori a controllo elettronico. La corrente fluirà dal pin emettitore (E) al pin collettore (C) solo quando una corrente sufficiente fluirà dal pin base (B) al pin collettore (C). In altre parole, possiamo controllare la corrente tra i pin E e C utilizzando il pin B. Il circuito è molto semplice. Il transistor permetterà il passaggio di una corrente di 9 V dalla batteria al motore solo quando il pin 3 è attivo. Poiché il pin 3 è un pin PWM, possiamo inviare impulsi di diversa ampiezza per controllare la velocità del motore utilizzando il metodo analogWrite. Il diodo viene utilizzato per impedire che il motore invii corrente al circuito a causa del suo movimento inerziale. La resistenza viene utilizzata per adattare l'uscita del pin Arduino ai corretti valori di ingresso del transistor.

Il codice per controllare il motore utilizzando il pin 3 è:

```
int motorPin = 3;                    //The pin attached to the motor
void setup()
{
   pinMode(motorPin, OUTPUT);        //Defines the pin 3 as output
}
void loop()
{
    analogWrite(motorPin, 255);      //Full speed
    delay(2000);
    analogWrite(motorPin, 100);      //Medium Speed
    delay(2000);
    analogWrite(motorPin, 10);       //Low speed
    delay(2000);
 }
```

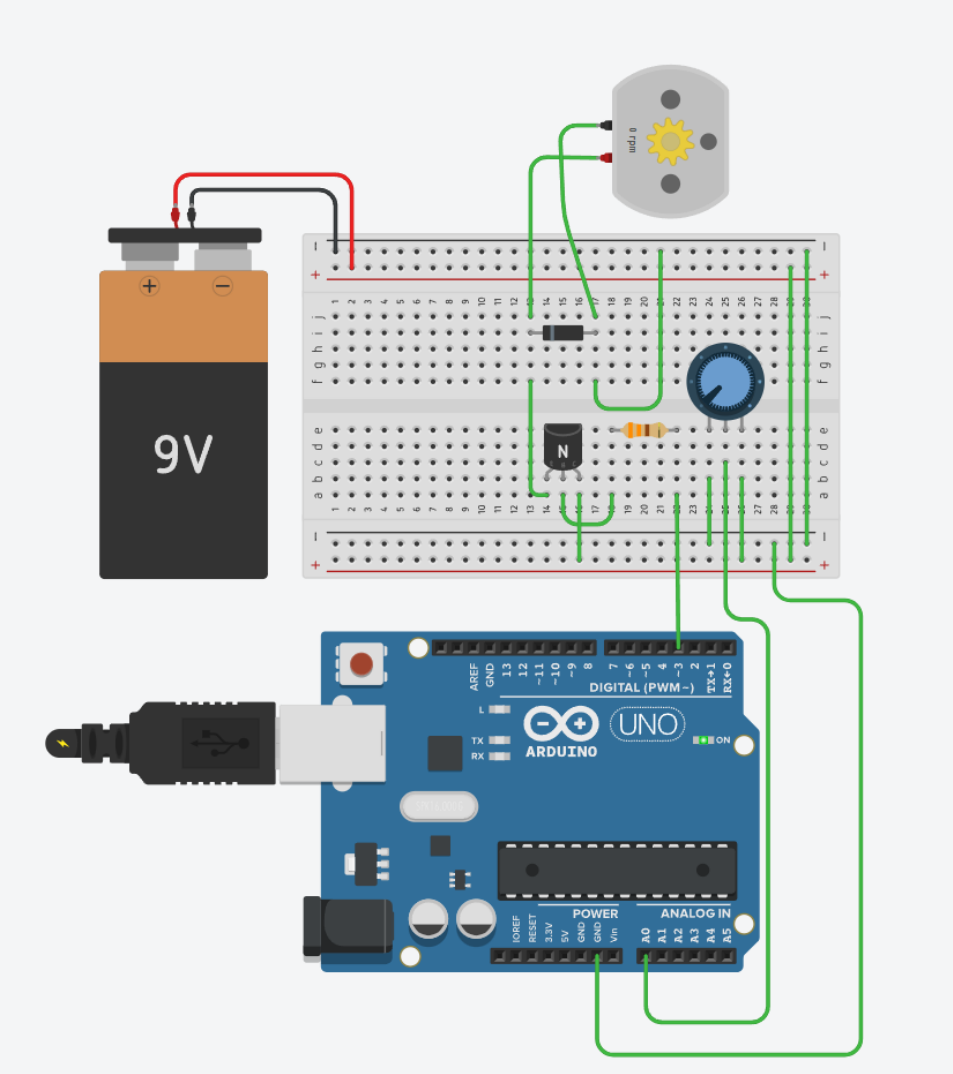
Circuito 2

Titolo Circuito: Controllo della velocità di un motore a corrente continua tramite un potenziometro

Descrizione del circuito: Questo circuito utilizza un transistor PNP 2N2222 per controllare l'alimentazione a 9 V tramite Arduino.

Inoltre, un potenziometro viene letto tramite il pin analogico A0 di Arduino e utilizzato per controllare la velocità del motore.

Modifichiamo il nostro circuito aggiungendo un potenziometro per controllare la velocità del motore.



Esempio di programma 2: Controllo del motore con un potenziometro

```
int motorPin = 3;                    //The pin attached to the motor
int controlPin = 0;                  //The pin attached to the pot
int speed = 0;                       //The speed assingned to the motor

void setup()
{
   pinMode(motorPin, OUTPUT);        //Set up the motor pin as output pin
}

void loop()
{
speed = analogRead(controlPin);    //Read the potentiometer value and store it on //speed variable
    analogWrite(motorPin, speed);    //Send speed to the motor
    delay(500);                      //Wait 500 millisecons
 }
```

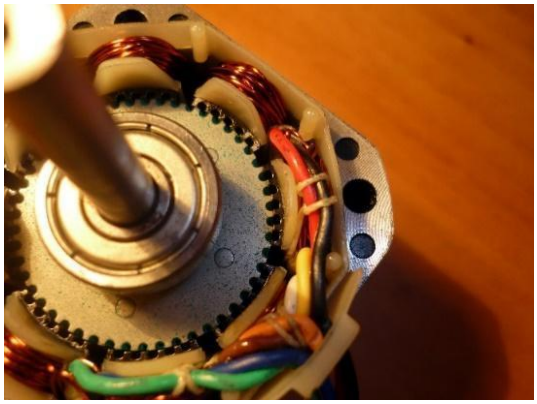
MOTORI PASSO-PASSO

Panoramica

Un motore passo-passo è un tipo di motore elettrico che si muove a passi discreti, rendendolo ideale per il controllo preciso di posizione, velocità e direzione. A differenza dei tradizionali motori a corrente continua, che ruotano ininterrottamente, i motori passo-passo suddividono una rotazione completa in più passi, ciascuno dei quali varia tipicamente da 0,9° a 1,8° per passo, a seconda del design del motore. Questa caratteristica unica consente loro di ottenere movimenti estremamente precisi e ripetibili senza la necessità di encoder o altri dispositivi di rilevamento della posizione.

Come funzionano i motori passo-passo

Il funzionamento di un motore passo-passo si basa sull'elettromagnetismo. All'interno di un motore passo-passo, sono presenti più bobine o avvolgimenti disposti attorno a un rotore centrale. Quando una corrente passa attraverso una bobina specifica, si crea un campo magnetico che interagisce con il rotore, facendolo ruotare di un certo angolo. Alimentando sequenzialmente diverse bobine secondo uno schema controllato, il rotore viene mosso con incrementi precisi (passi).



La chiave per controllare il movimento di un motore passo-passo risiede nel circuito di pilotaggio, che regola la sequenza e la temporizzazione degli impulsi di corrente inviati alle bobine. Il driver può funzionare in diverse modalità, come:

- Passo completo: eccita una bobina alla volta, consentendo un passo maggiore (ad esempio, 1,8° per passo).
- Mezzo passo: eccita due bobine alternativamente, consentendo passi più piccoli e un movimento più fluido.
- Microstepping: divide ogni passo completo in passi più piccoli, migliorando la fluidità e la risoluzione e riducendo le vibrazioni.

Vantaggi dei motori passo-passo

- **Precisione e accuratezza**: i motori passo-passo sono estremamente precisi, il che li rende ideali per applicazioni che richiedono un posizionamento preciso, come stampanti 3D, macchine CNC e bracci robotici.

- **Nessun feedback richiesto**: i motori passo-passo possono funzionare senza sistemi di feedback esterni (come gli encoder) poiché la posizione del rotore è determinata dal numero di passi effettuati.
- **Affidabilità**: questi motori sono semplici da costruire, riducendo il numero di componenti soggetti a guasti.
- **Basso costo**: i motori passo-passo sono relativamente economici, il che li rende una soluzione conveniente per molte applicazioni.

Svantaggi dei motori passo-passo

- **Bassa efficienza**: i motori passo-passo possono essere meno efficienti dal punto di vista energetico rispetto ad altri tipi di motori, soprattutto a velocità più elevate.
- **Calo di coppia ad alte velocità**: ad alte velocità, i motori passo-passo tendono a perdere coppia, il che può influire sulle prestazioni in determinate applicazioni.
- **Vibrazioni e rumore**: i motori passo-passo possono produrre vibrazioni e rumore, soprattutto quando funzionano a velocità inferiori o con bassa corrente.

Applicazioni per motori passo-passo

I motori passo-passo sono ampiamente utilizzati in applicazioni che richiedono un controllo preciso della posizione e della rotazione, tra cui:

- **Robotica**: per il controllo preciso di bracci robotici e altre parti mobili.
- **Macchine CNC**: per muovere l'utensile o il pezzo in lavorazione con incrementi controllati.
- **Piattaforme per telecamere**: per un controllo preciso di panoramica e inclinazione in fotografia e videografia.
- **Apparecchiature mediche**: per applicazioni come pompe, protesi e macchinari diagnostici.

I motori passo-passo sono un componente essenziale in molte applicazioni in cui precisione e controllo sono fondamentali. La loro capacità di suddividere la rotazione completa in piccoli passi discreti consente un posizionamento accurato e la loro semplicità li rende una scelta popolare sia per gli hobbisti che per gli ingegneri industriali. Sebbene non siano la scelta migliore per applicazioni ad alta velocità ed efficienza, i loro vantaggi in termini di controllo e affidabilità li rendono indispensabili in settori come la robotica, la produzione e l'automazione. Solitamente, i motori passo-passo utilizzano sensori per rilevare quando gli oggetti in movimento raggiungono l'inizio o la fine del percorso; questi sensori sono chiamati sensori di finecorsa e sono essenziali per determinare la posizione iniziale di un motore passo-passo.

Circuito 3:

Titolo del circuito: Spostamento di un motore passo-passo di un giro in avanti e di un giro all'indietro

Descrizione del circuito: Questo circuito utilizza un motore passo-passo 28BYJ-48 e un controller basato su ULD2003 per fornire alimentazione al motore passo-passo ed evitare di danneggiare la scheda Arduino.

In questo esempio utilizzeremo un piccolo motore passo-passo chiamato 28BYJ-48

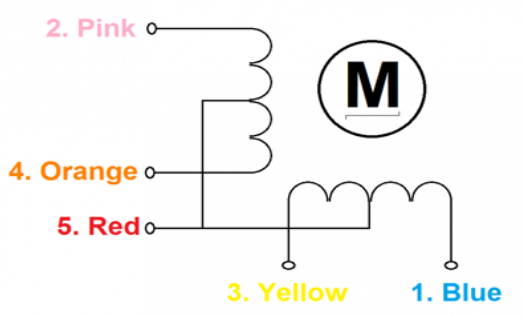


Figura 2 Collegamenti di 28BYJ-48

Questo motore ha quattro bobine con un cablaggio con schema unipolare ed è solitamente controllato da un circuito basato su ULN2003 che fornisce corrente a ciascuna bobina per evitare di danneggiare i pin I/O del nostro Arduino.

Il motore 28BYJ-48 ha 64 passi per giro quando utilizzato in modalità mezzo passo e ha un rapporto di trasmissione interno di 1:64, quindi 64 * 64 = 4096 passi totali per giro.

In questo progetto utilizzeremo:

- 1 motore passo-passo 28BYJ-48
- 1 controller basato su ULN2003
- Cavi
- Scheda Arduino Uno

Collegare il motore passo-passo al controller utilizzando la spina appropriata:

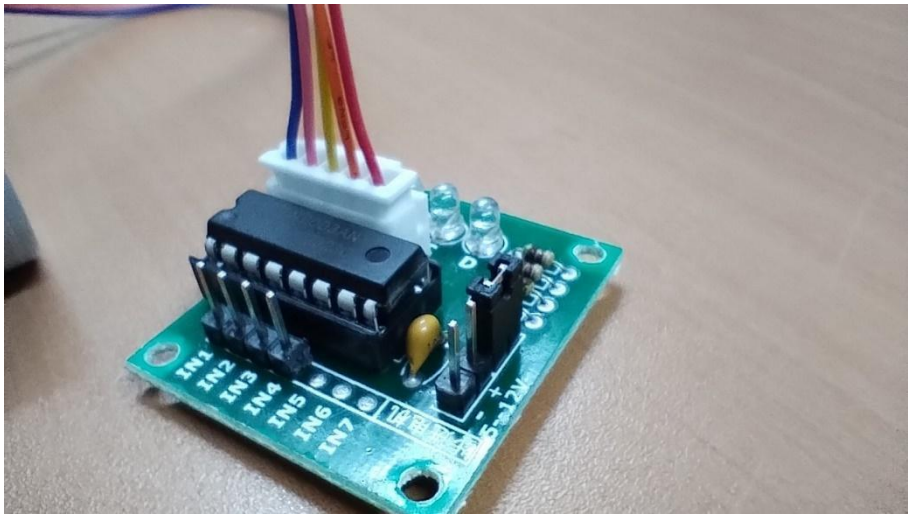
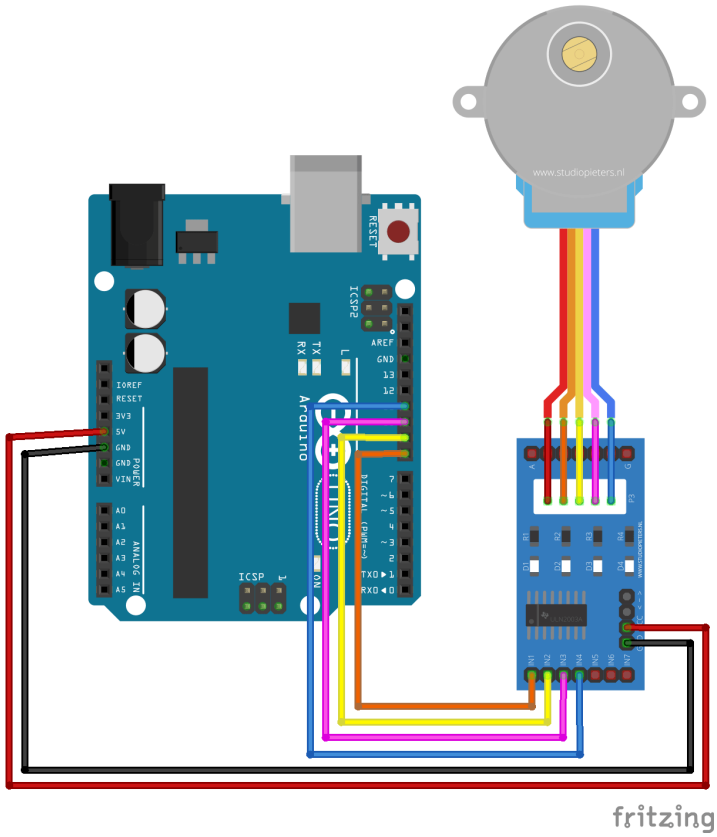


Figura 3: Motore passo-passo collegato al controller

Ora collega il controller alla scheda Arduino seguendo questo schema:



Circuito 3:

Per controllare il motore passo-passo utilizzeremo la libreria Stepper.

I metodi più utili della libreria Stepper sono:

stepper(steps, pin1, pin2, pin3, pin4)	Crea un nuovo stepper con i seguenti parametri: Passi: Numero di passi per giro Pin 1 e Pin 2: Pin collegati al motore Pin 3 e Pin 4: (Facoltativo): Pin collegati al motore se il motore è dotato di cavi
step(steps)	Spostare lo stepper di un numero specifico di passi
setSpeed(rpm)	Imposta la velocità del motore su un numero di giri al minuto specifico. Tieni presente la velocità massima del tuo motore.

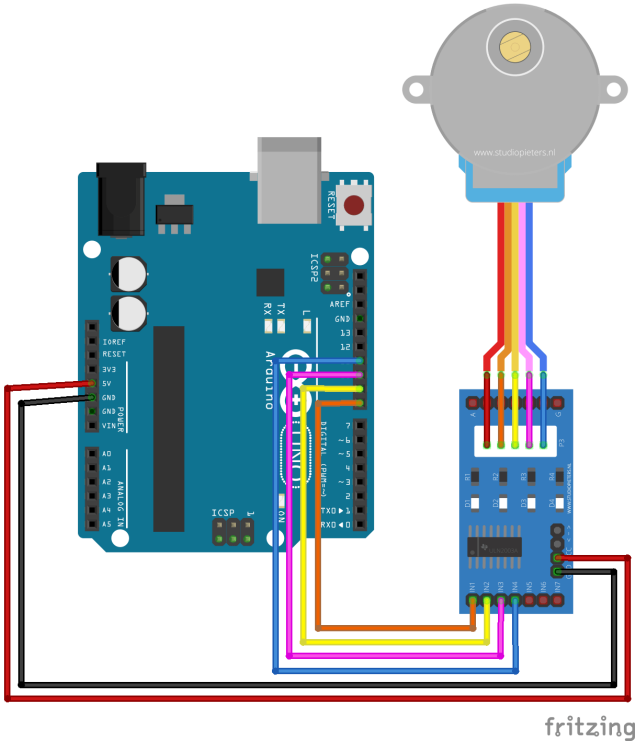
Il codice per muovere il motore di un giro in avanti e di un giro indietro è:

```
#include <Stepper.h>                    // Include the steper control library

Stepper stepper(4096, 8, 10, 9, 11);    // Create a stepper with 4096 steps per revolution


void setup() {
}

void loop {
   stepper.step(4096);                  //One complete revolution
   delay(5000);                         //wait for 5 seconds
   stepper.step(-4096);                 //One complete revolution backwards (negative)
   delay(5000);                         //wait for 5 seconds
}
```

Circuito 4:

Titolo del circuito: Muovere un motore passo-passo come la lancetta dei secondi di un orologio

Descrizione del circuito: utilizzando lo stesso stepper e lo stesso controller, modificheremo il codice per far muovere il motore come la lancetta dei secondi di un orologio: 1/60 di rivoluzione ogni secondo.



Il codice per muovere il motore passo-passo 1/60th giro ogni secondo è:

```
#include <Stepper.h>                    // Include the steper control library
Stepper stepper(4096, 8, 10, 9, 11);    //Create a stepper with 4096 steps per revolution
void setup() {
   stepper.setSpeed(240)                //240 rpm speed for quick seconds hand movement
}
void loop {
   stepper.step(68);                    //60 divided into 4096 equals aprox 68 steps
//per second
   delay(1000);                         //each second we move the motor
}
```

SERVOMOTORI

Panoramica

I servomotori sono ampiamente utilizzati in robotica, automazione e altri settori in cui è richiesto un controllo preciso della posizione angolare. Si differenziano dai motori tradizionali in quanto possono ruotare in una posizione specifica, anziché ruotare in modo continuo. Questo li rende ideali per compiti come il movimento di bracci robotici, il controllo della posizione di una telecamera o la regolazione dell'angolazione dei pannelli solari.

Come funzionano i servomotori?

Un servomotore è un piccolo motore dotato di un sistema di feedback che gli consente di ottenere un movimento angolare preciso. A differenza dei normali motori a corrente continua, che ruotano continuamente in entrambe le direzioni, un servomotore può ruotare solo entro un intervallo limitato (tipicamente da 0° a 180°) e la sua posizione può essere controllata con estrema precisione.

Il servomotore è costituito da tre componenti chiave:

- Motore: aziona la rotazione.
- Potenziometro di feedback: monitora la posizione del motore.
- Circuito di controllo: riceve il segnale di controllo e regola di conseguenza la posizione del motore.

I servomotori funzionano utilizzando segnali a modulazione di larghezza di impulso (PWM). Arduino invia un segnale PWM al servo, che regola la posizione del motore in base alla durata del segnale.

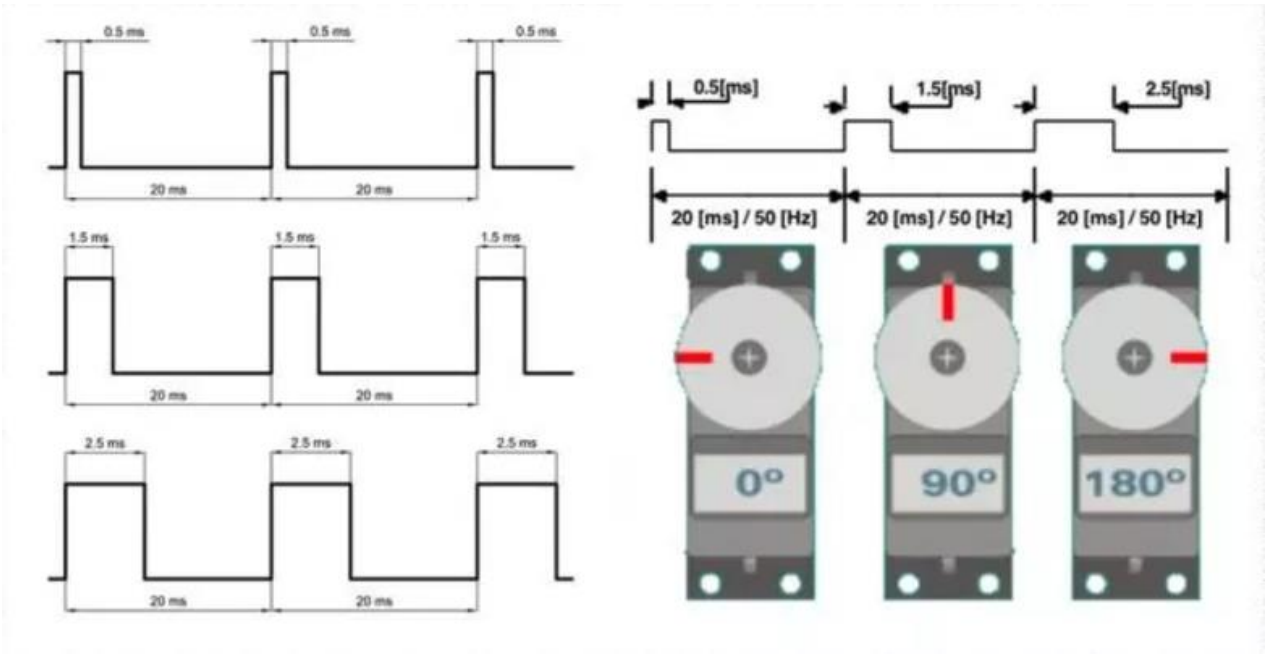


Figura 4 How PWD controlla la posizione del servomotore

Un impulso breve (1 ms) potrebbe impostare il servo a 0°.

Un impulso lungo (2 ms) potrebbe impostare il servo a 180°.

Un impulso di circa 1,5 ms generalmente posiziona il servo a 90°.

Variando la larghezza dell'impulso, il servomotore può essere posizionato in qualsiasi punto del suo intervallo operativo.

Applicazioni dei servomotori

- **Robotica**: i servomotori sono ampiamente utilizzati nei robot per controllare giunti, ruote o attuatori.
- **Gimbal per telecamere**: per stabilizzare una telecamera, i servomotori possono controllarne l'orientamento e mantenerla ferma.
- **Posizionamento dell'antenna**: i servomotori sono utilizzati nelle parabole radio e satellitari per regolare la direzione dell'antenna.
- **Veicoli RC**: auto, aerei e barche telecomandati utilizzano spesso i servocomandi per sterzare e controllare altri componenti meccanici.

I servomotori sono componenti essenziali per progetti che richiedono un controllo preciso degli angoli. Con Arduino, controllare un servo è un compito semplice, grazie alla libreria Servo, che semplifica la generazione di segnali PWM. Uno dei maggiori vantaggi dei servomotori rispetto ai motori passo-passo è che, poiché il servomotore è in grado di rilevare autonomamente la propria posizione, di solito non necessita di sensori di finecorsa come nei motori passo-passo o in corrente continua.

Vantaggi dei servomotori:

- Grande precisione senza bisogno di sensori di finecorsa.
- Mantengono la posizione anche in presenza di disturbi esterni.

Svantaggi dei servomotori:

- Gamma di movimento limitata, non adatta al movimento continuo
- Velocità massima limitata

Esempio di Progetto: spostare un servomotore

L'SG90 è un servomotore piccolo e pratico che può essere alimentato tramite i pin di Arduino. I servomotori più grandi richiedono un alimentatore esterno.

Per questo progetto utilizzeremo:

- 1 Servomotore SG90
- Cavi
- Scheda ArduinoUno

Il pin di controllo del servo (giallo) deve essere collegato a un pin digitale PWM; useremo il pin ~3.

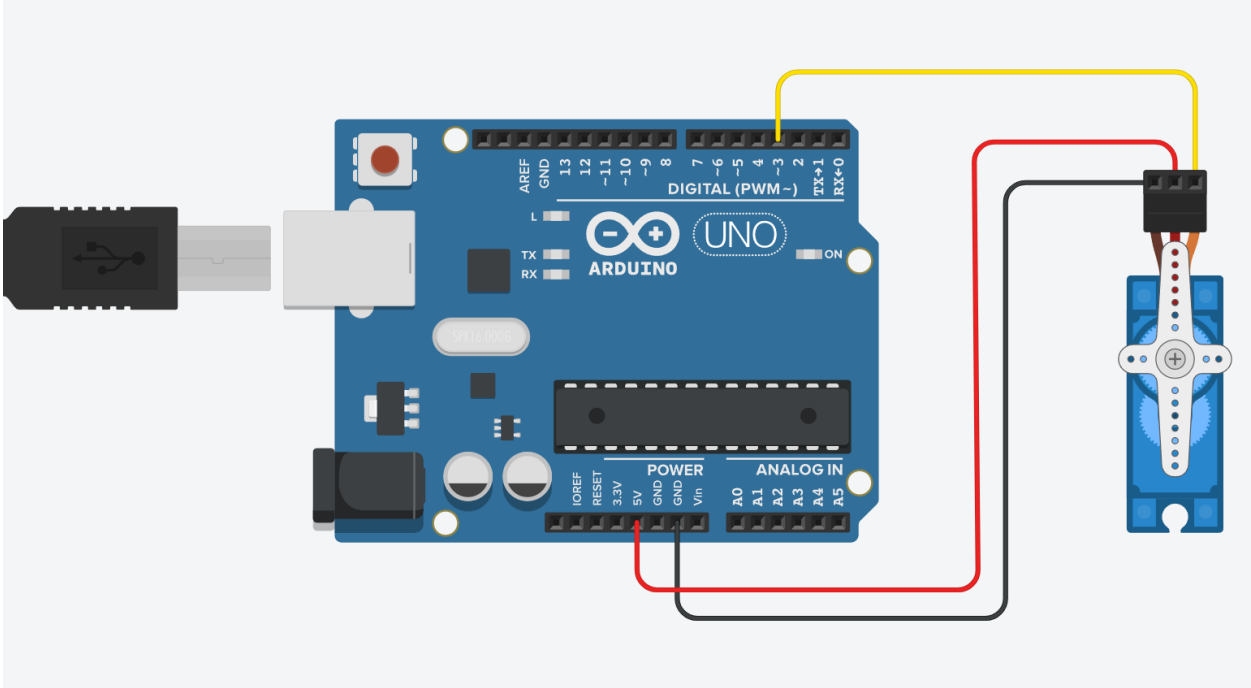
Utilizzeremo la libreria Servo.h per aiutarci a gestire il servo. I metodi Servo.h più importanti sono:

attach(pin, min, max)	Collega la variabile del servo a un pin. Il pin deve essere un pin digitale PWM. I parametri opzionali min e max impostano la durata dell'impulso, in microsecondi, che il servo si aspetta per impostare l'angolo minimo e massimo. Se non vengono specificati valori min o max, vengono utilizzati i valori predefiniti di 544 e 2400 millisecondi.
write(angle)	Impostare l'angolo del servo al valore desiderato. Il servo si muoverà in questa posizione.

Circuito 5:

Titolo del circuito: Movimento semplice di un servomotore

Descrizione del circuito: Utilizzando il pin digitale 3 di Arduino, posizioneremo il servomotore a 90, 180 e 0 gradi. Il servomotore è collegato a 5 V e GND per l'alimentazione e il pin digitale 3 in modalità PWM per controllare la posizione.



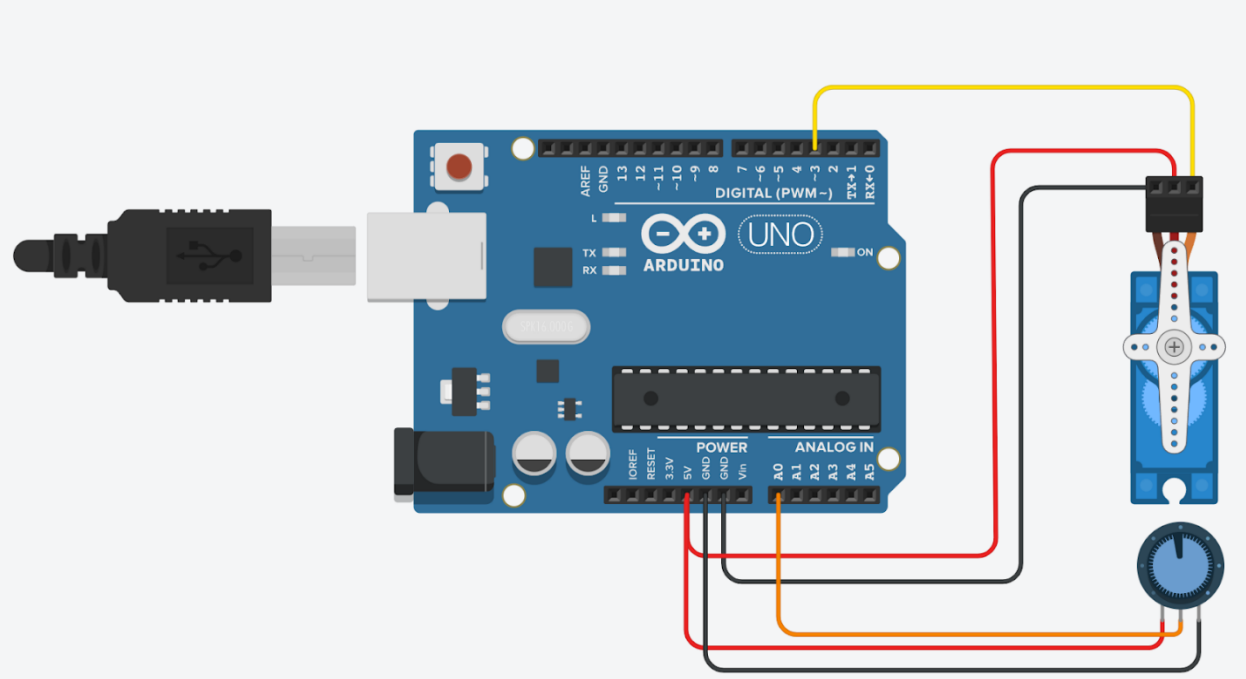
Circuito 5

```
#include <Servo.h>;          //The servo library
Servo myServo;               //We need to create a servo object
void setup()
{
  myServo.attach(3);         //Cofigure the servo pin
  myServo.write(0);          //Set servo at 0 degrees angle
}
void loop()
{
  myServo.write(90);         //set servo at 90 angle
  delay(1000);               //wait for 1 second
  myServo.write(180);        //set servo at 180 angle (max)
```

```
  delay(1000);              //wait for 1 second
  myServo.write(0);         //set servo at 0 angle (min)
  delay(1000);              //wait for 1 second
}
```

Circuito 6:

Titolo del circuito: Controllo di un servomotore tramite potenziometro



Circuito 6

In questo esempio, la posizione del servo è controllata dal potenziometro. Per tradurre la posizione del potenziometro nella posizione del servo, viene utilizzata la mappa delle funzioni:

map(value, from_min , from_max, to_min, to_max)	Mappa un valore dalla scala [from_min, from_max] alla scala [to_min, to_max].

```
#include <Servo.h>;
Servo myServo;                          //Creates the servo object
void setup()
```

```
{
  myServo.attach(3);                    //Attach the servo to pin number 3
  myServo.write(0);                     //position servo in 0 position
}

void loop()
{
  int position = map(analogRead(0),0,1023,0,175);//map the position of the potentiometer
  myServo.write(position);              //to a angle of the servo and store the
}                                       //mapped value in position variable
                                        //before sending it to the servo
```

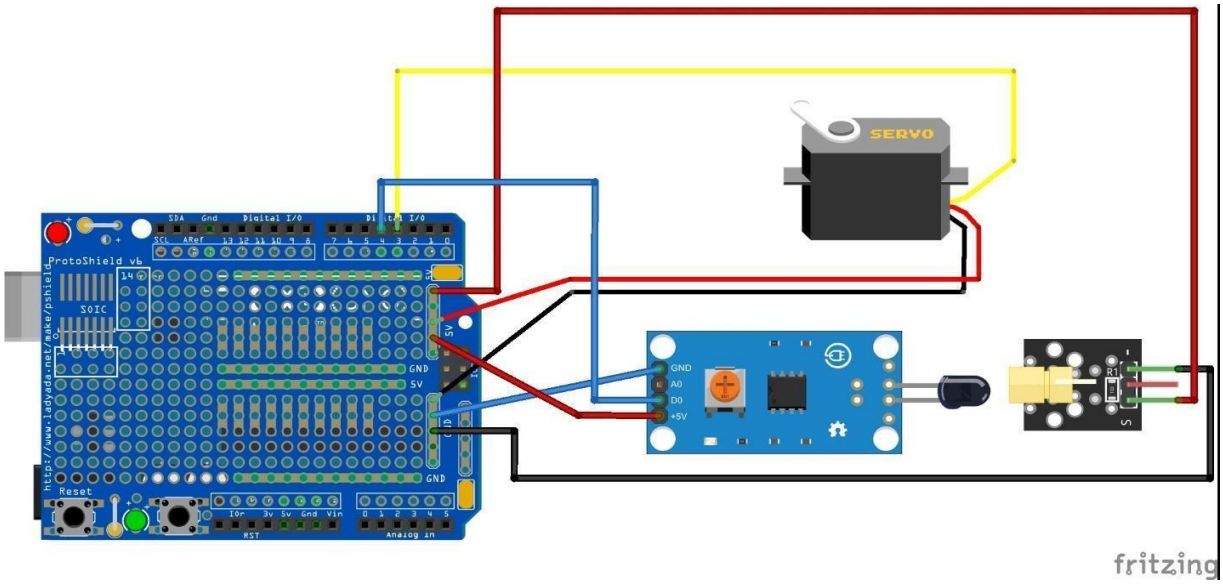
Circuito 7:

Titolo del circuito: Barriera per auto controllata da rilevatore laser

Descrizione del circuito: simuleremo una barriera per auto utilizzando un servo. Per rilevare la scheda vengono utilizzati un emettitore di luce LASER e un rilevatore di luce. Per facilitare le connessioni viene utilizzato un proto-shield.

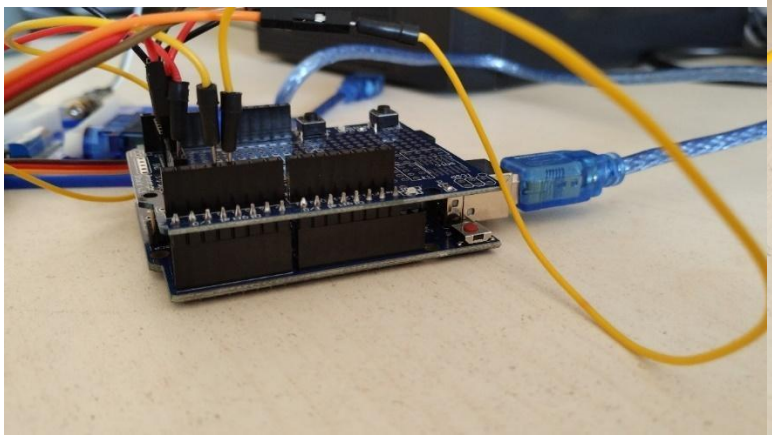
Per questo progetto utilizzeremo:

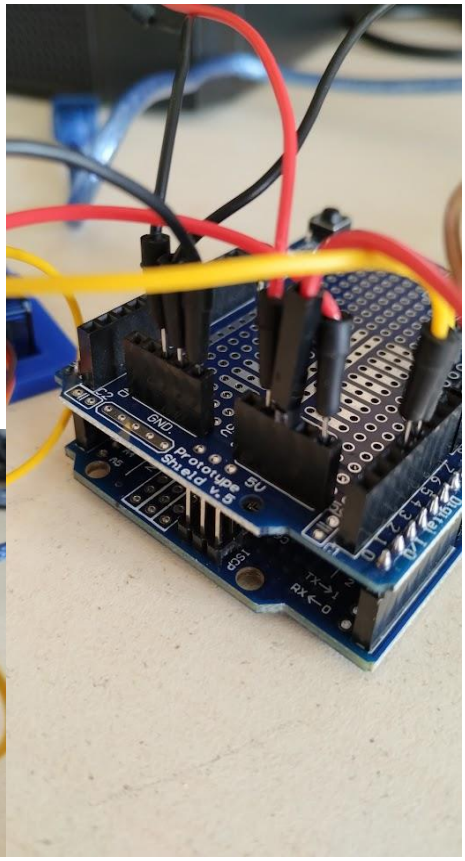
- 1 Modulo emettitore laser
- 1 Modulo rilevatore di luce
- 1 Servomotore SG90
- 1 Protoshield
- Cavi
- Scheda di supporto
- Barriera
- Auto
- Scheda Arduino Uno



Circuito 7

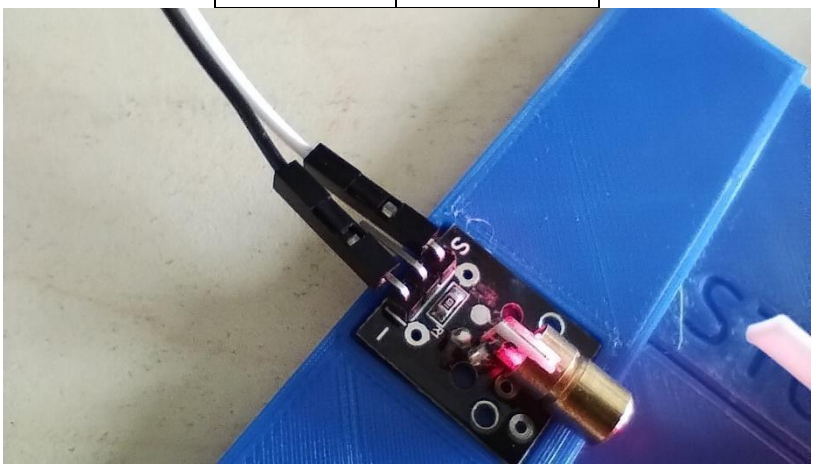
Collega il proto shield alla scheda Arduino. Nota che sono presenti due connettori per 5V e GND, molto comodi per collegare i cavi di alimentazione dei sensori e del servomotore.



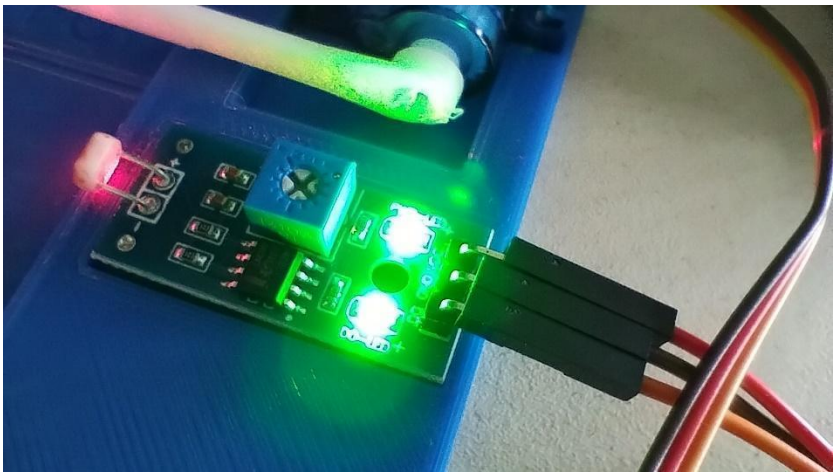


Collegare i pin – e S del modulo LASER ai connettori GND e 5V del Proto Shield:

Modulo emettitore LASER	
S	5V
Pin centrale	non connesso
-	GND



Collegare i pin VCC, GND e D0 del modulo rilevatore di luce ai pin corrispondenti del prototipo:



Modulo rilevatore di luce	
D0	Digital Pin 4
GND	GND
VCC	5V

Accendete Arduino, puntate il laser verso il sensore di luce e regolate la sensibilità della luce utilizzando un cacciavite sul potenziometro blu del modulo sensore di luce. La luce sinistra dovrebbe accendersi solo quando il laser colpisce il sensore, come nell'immagine sopra.

Collegare i pin del Servo al pin di alimentazione e ai pin dati del prototipo:

Servomotore	
Marrone	GND
Rosso	5V
Giallo	Digital Pin ~3

Fissare la barriera al servomotore. Potrebbe essere necessario regolare l'angolazione della barriera se il servomotore non è nella posizione corretta.

Posizionare tutti gli elementi sulla piastra di supporto e fissarli con nastro adesivo o Blue Tack per assicurarsi che rimangano in posizione.

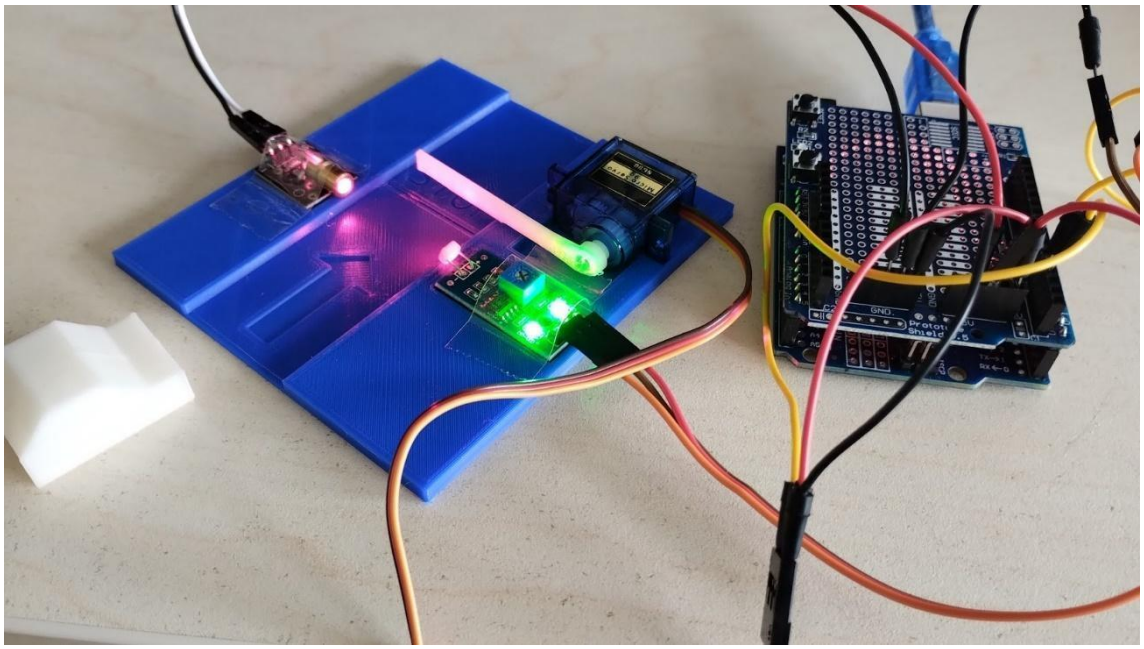


Figura 5 Barriera servocontrollata

Controllo di un servomotore con Arduino

Per controllare un servomotore con un Arduino, possiamo utilizzare la libreria Servo, che rende il processo semplice e pratico. La libreria fornisce funzioni per collegare il servo a un pin, impostarne la posizione e muoverlo lentamente all'angolazione desiderata. Per evitare falsi rilevamenti di

oggetti, il programma leggerà il sensore ogni 250 ms. Il programma considererà la presenza di un'auto in attesa solo quando 4 letture consecutive rilevano un ostacolo tra il LASER e il sensore di luce.

```
#include <Servo.h>
Servo myServo;
int detections;
int sensorPin = 4;              //Pin del sensore di luce
int servoPin = 3;               //Pin Servo Motore. Deve essere un pin PWM
int open = 0;                   //Posizione del servo motore nello stato aperto
int close = 128;                // Posizione del servo motore nello stato chiuso
int threshold = 4;              // Numero di letture di riga da considerare come rilevamento
int waitToPass = 4000;          //Tempo di attesa per il passaggio dell'auto
void setup() {
  myServo.attach(servoPin);          //Inizializza il servo motore
  myServo.write(close);              //Imposta la barriera in posizione di chiusura
  pinMode(sensorPin , INPUT);        // Imposta il pin del sensore come pin di input
  detections = 0;                    // Inizializza il numero di rilevamento degli oggetti
}

void loop() {

  if (digitalRead(sensorPin)==1){              //Se qualcosa viene rilevato
    detections = min ( threshold, detections +1);     //Incrementa il numero di rilevamenti di uno
                                               //fino alraggiungimento della soglia
  } else {                                     //If no object is detected
    detections = max ( 0, detections -1);      // Decrementa il numero di rilevamenti di uno
                                               // fino alraggiungimento di 0
  }
  delay (250);                                 //Quattro letture al secondo

  if (detections == threshold){                // Se viene raggiunto il livello di soglia di
rilevamento
    myServo.write(open);                       //Apri la barriera
    delay (waitToPass);                        //Aspetta che la macchina passi
  }

  if (detections == 0){                        // Se non viene rilevato alcun oggetto per le letture
                                               di soglia
     myServo.write(close);                     //Chiudi la barriera
  }

}
```

Co-funded by the
Erasmus+ Programme
of the European Union

Erasmus+ KA210-VET

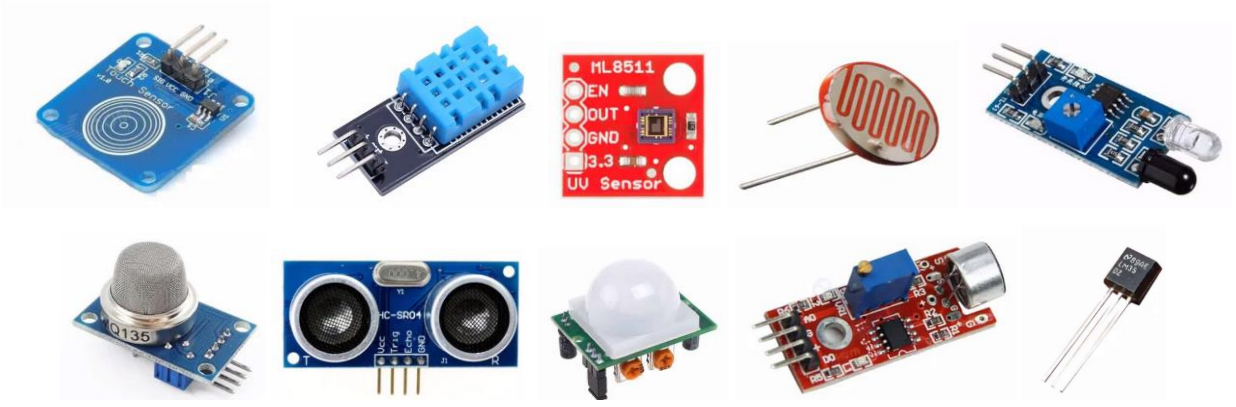
Small-scale partnerships in Vocational Education and Training

Project Title: “Using Arduinos in Vocational Training”

Project Acronym: “UsingARDinVET”

Project No: “2023-1-RO01-KA210-VET-000156616”

Modulo SENSORI e KIT di formazione



Co-funded by the
Erasmus+ Programme
of the European Union

Modulo SENSORI e KIT di formazione

L'obiettivo di questo modulo è spiegare come funzionano i sensori in interazione con Arduino. Verranno presentati diversi tipi di sensori basati su Arduino e verranno sviluppate applicazioni che li includono, al fine di essere utilizzate dagli insegnanti dell'istruzione secondaria professionale.

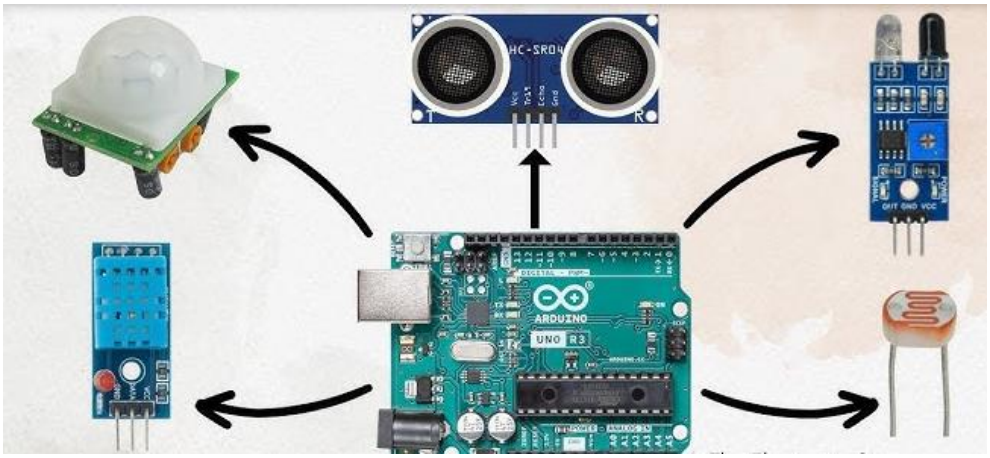
Informazioni sui sensori

Un sensore è un dispositivo con cui la scheda Arduino interagisce con l'ambiente. Un sensore riceve in ingresso un segnale e risponde fornendo in uscita un segnale elettrico.

Per essere più specifici, i sensori ricevono diversi tipi di segnali, ad esempio fisici, chimici o biologici, e rispondono convertendoli in un segnale elettrico, sotto forma di corrente, tensione o carica. Potremmo anche dire che un sensore è un traduttore che converte un valore non elettrico in un valore elettrico.

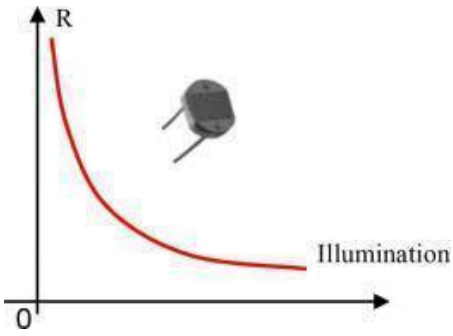
Esistono sensori di diversi tipi in base alle applicazioni, al segnale di ingresso, al meccanismo di conversione e al materiale utilizzato, con caratteristiche come costo, precisione o portata.

Li possiamo trovare ovunque, poiché il nostro mondo è pieno di diversi tipi di sensori e delle loro applicazioni, semplici o più complesse: nei nostri uffici, giardini, centri commerciali, case, automobili, giocattoli, ecc. Ecco perché è fondamentale comprenderne il funzionamento e le molteplici possibilità.



1. Fotoresistore

1.1 Informazioni sui fotoresistori



Un fotoresistore è un resistore variabile controllato dalla luce. Un'elevata intensità luminosa incidente sulla superficie causerà una resistenza inferiore, mentre una minore intensità luminosa causerà una resistenza maggiore. L'uso più comune è come interruttore di attivazione luce/buio. È anche chiamato LDR (Light Dependent Resistor).

Vediamo come reagisce un fotoresistore alla luce.

1.2 Circuito 1

Titolo del circuito: Rilevazione della luminanza

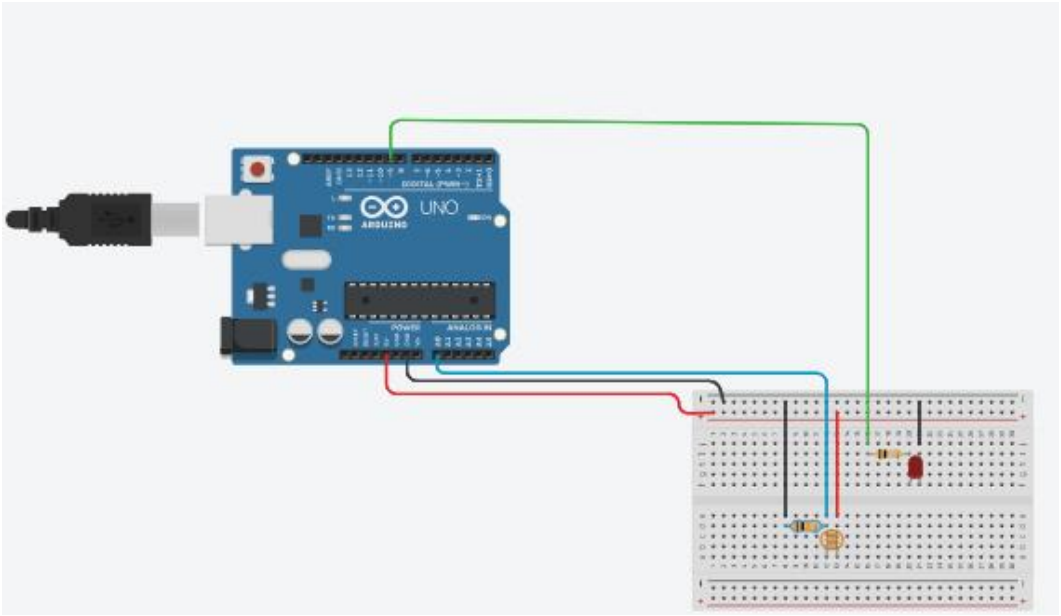
Descrizione del circuito: Il circuito rileva la luminosità della stanza, se scende sotto il limite impostato accende il led.

Cosa ti servirà:

a) Breadboard
b) Fotoresistore
c) Resistore 220Ω* ,10KΩ **
d) Led 3mm
e) Arduino

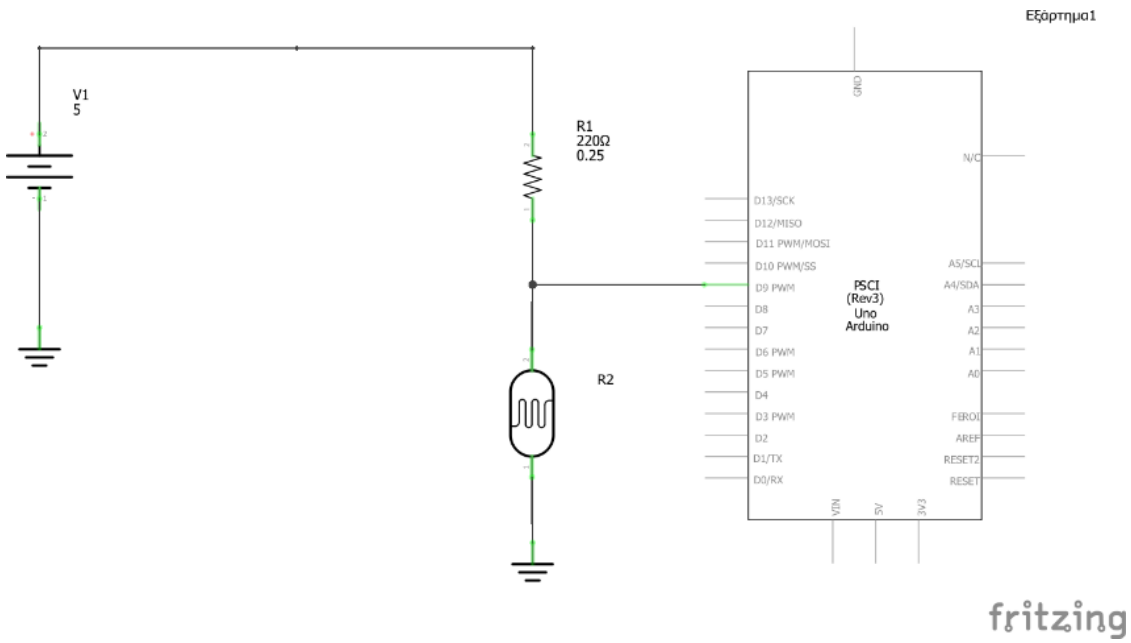
* La resistenza da 220Ω è la resistenza di protezione standard del LED a 5 volt.

** Il resistore da 10 KΩ viene utilizzato per proteggere Arduino dalla sovracorrente quando il valore del fotoresistore diminuisce significativamente.



Il LED collegato al pin 9 è impostato come uscita. Quando il pin 9 è alto, il LED emette luce; altrimenti, rimane spento.

Con il fotoresistore e il resistore da 10 kΩ, costruiamo un partitore di tensione. Il valore che verrà inviato al processore dipenderà dall'intensità della luce sul sensore.



1.3 Il Programma

```
1  const int photoantistasi = A0; // Fotoresistore sul pin analogico A0 di Arduino
2  const int ledPin = 9; // Pin LED sul pin 9 di Arduino
3
4  //Variables
5  int timi ; // Memorizza il valore dal fotoresistore
6
7  void setup (){
8   pinMode (ledPin , OUTPUT ); // Imposta lepPin - 9 pin come uscita
9   pinMode (fotoantistasi , INPUT ); // Imposta il pin fotoantistasi - A0 come input
          (facoltativo)
10  Serial . begin (9600); // // apre la porta seriale, imposta la velocità dei dati a 9600 bps
11
}
12
13 void loop (){
14 value = analogRead (fotoantistasi);
15
16  //È possibile modificare il valore "70".
17  if (timi > 70){
18    digitalWrite (ledPin , LOW ); //Spegni il led
19 }
20  else {
21    digitalWrite (ledPin , HIGH ); //Accendi il LED
22 }
23
24  delay (500); //Piccolo ritardo
25 }
```

Il programma legge l'ingresso analogico. Se il valore supera '70', il LED si accende; altrimenti, si spegne. Ogni ciclo di programma include un ritardo di 500 millisecondi (utilizzando il comando 'delay(500)') per dare al sensore il tempo di effettuare le misurazioni.

tips

Puoi modificare il tuo Programma e, tramite la porta seriale, monitorare i valori del sensore sul tuo computer.

```
26 const int photoantistasi = A0; // Fotoresistore sul pin analogico A0 di Arduino
27 const int ledPin = 9; // Pin LED sul pin 9 di Arduino
28
29 //Variabili
30 int timi ; // Memorizza il valore dal fotoresistore
31
32 void setup (){
33  pinMode (ledPin , OUTPUT ); // Imposta lepPin - 9 pin come uscita
34  pinMode (fotoantistasi , INPUT ); // Imposta il pin pResistor - A0 come input (facoltativo)
35  Serial . begin (9600); // // apre la porta seriale, imposta la velocità dei dati a 9600 bps
36
}
37
38 void loop (){
39 value = analogRead (fotoantistasi);
40
41   //È possibile modificare il valore "25".
42   if (timi > 70){
43     digitalWrite (ledPin , LOW ); //Spegni il led
44 }
45   else {
46     digitalWrite (ledPin , HIGH ); //Accendi il LED
47 }
48
49  delay (500); //Piccolo ritardo
50   Serial . println (timi); // stampa il valore (dopo ogni modifica del valore della variabile
                sulla riga sullo schermo).
```

2. Sensore di movimento

Il sensore di movimento funziona rilevando le variazioni della radiazione infrarossa o la presenza di calore e movimento all'interno della sua area di copertura. Il tipo più comune di sensore di movimento è un sensore a infrarossi passivi (PIR), che rileva le variazioni della radiazione infrarossa emessa dagli oggetti nel suo campo visivo.

2.1 Informazioni sul sensore PIR

Il sensore di rilevamento del movimento a infrarossi (sensori piroelettrici a infrarossi) è un sensore che consente di rilevare la presenza di un organismo vivente in movimento in un'area specifica.

Tutti gli esseri viventi emettono radiazioni infrarosse, che possono essere rilevate facilmente tramite appositi sensori.

Il sensore PIR è costituito da due aree che rilevano la radiazione infrarossa.

Quando un organismo vivente entra nella prima area, si crea una differenza di potenziale positiva nel circuito elettronico del sensore.

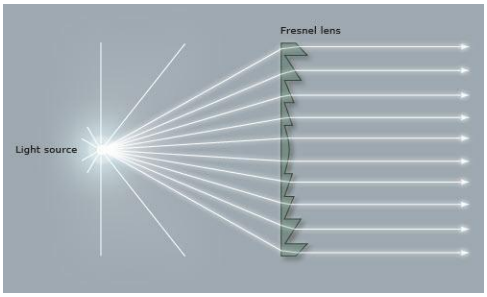
Quando esce dalla seconda area, si crea una differenza di potenziale negativa.

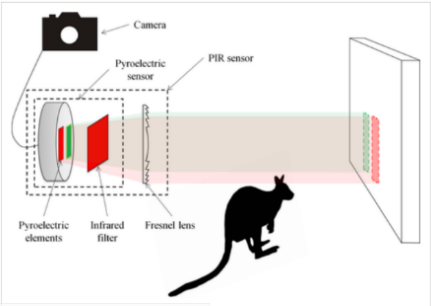
Questa tensione attiva il circuito elettronico del sensore e invia un '1' logico al processore (presenza di movimento nello spazio). Il problema di questa funzione è che rileva solo una piccola area, la sottile zona tra le due aree di rilevamento.

Per aumentare la portata dell'area, il sensore è coperto da una lente di Fresnel.

Augustin Fresnel scoprì la lente di Fresnel e la trovò applicazione nei fari nautici.

La sua proprietà è quella di concentrare la luce di una sorgente in una direzione specifica, a un livello particolare.





Qui lo usiamo al contrario, come si può vedere nell'immagine.

La radiazione infrarossa presente in uno spazio converge sul sensore e aumenta così l'area che può rilevare

2.2 Circuito 2:

Titolo del circuito: Rilevatore di onde

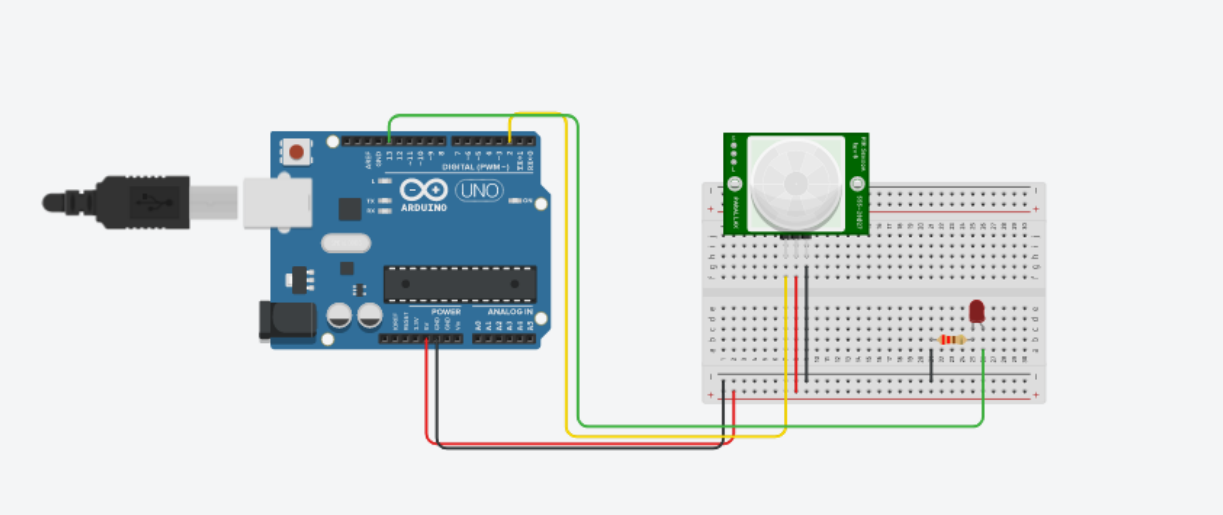
Descrizione del circuito: Il circuito monitora lo spazio. Se viene rilevato un movimento, attiva il LED.

Cosa ti servirà:

a) Breadboard
b) Sensore RIP
c) Resistore 220Ω* ,10KΩ **
d) Led da 3 mm
e) Arduino

* La resistenza da 220Ω è la resistenza di protezione standard del LED a 5 volt.

** Il resistore da 10 KΩ viene utilizzato per proteggere Arduino da sovracorrente quando il valore del fotoresistore diminuisce significativamente.



Alimentare il sensore con una tensione di 5 volt. Il segnale del pin è collegato al pin 2 dell'Arduino.

Il LED è collegato al pin 13 dell'Arduino.

Quando il sensore rileva un movimento, invia il bit '1' e il programma accende il led

Il sensore PIR ha tre pin: il pin di alimentazione, collegato a un alimentatore da 5 volt; il pin di terra, collegato al pin di terra (GR); e il pin di segnale. Il pin di segnale emette il valore 0 quando non viene rilevato alcun movimento e il valore 1 quando il sensore rileva un movimento.

2.3 Il Programma

```
int led = 13;                    // il pin a cui è collegato il LED
int sensor = 2;                  // il pin a cui è collegato il sensore
stato int = BASSO ;              // per impostazione predefinita, nessun movimento rilevato
int val = 0;                     // variabile per memorizzare lo stato del sensore (valore)

void setup () {
  pinMode (led , OUTPUT );       // inizializza il LED come uscita
  pinMode (sensore , INPUT );    // inizializza il sensore come input
  pinMode (cicalino , PRODUZIONE );
  tono (cicalino , 1000 , 2000);
  Serial . begin (9600);         // inizializza la seriale

}
void loop (){
valore = digitalRead (sensore);  // legge il valore del sensore
  if (valore == HIGH ) { /                    / controlla se il sensore è HIGH
    digitalWrite (led , HIGH );  // accende il LED
    delay (100);                 // ritardo 100 millisecondi
    if (stato == BASSO ) {
      Seriale . println ( "Movimento rilevato!" );
stato = HIGH ;                   // aggiorna lo stato della variabile a HIGH
}
}
  else {
      digitalWrite (led , LOW );  // spegne il LED
      noTone (cicalino);
      ritardo (200);              // ritardo 200 millisecondi

      if (stato == ALTO ){
        Seriale . println ( "Movimento interrotto!" );
stato = LOW ;                    // aggiorna lo stato della variabile a LOW
}
}
}
```

Il programma legge l'ingresso 2 di Arduino. Se il valore è '1', il LED si accenderà e, tramite la porta seriale, verrà visualizzato il messaggio "Movimento rilevato!"; in caso contrario, il LED si spegnerà e verrà visualizzato il messaggio "Movimento interrotto".

3. Sensore di fiamma

Un tema importante nell'automazione degli edifici è la rilevazione incendi. Vengono utilizzati due tipi di sensori: il rilevamento del fumo e il rilevamento della fiamma. Solitamente, quando si attiva il sensore di fumo, si attiva un allarme, mentre se si attiva il sensore di fiamma, si attiva il sistema di spegnimento.

Esistono diversi modi per rilevare un incendio, ad esempio rilevando le variazioni di temperatura, il fumo, ecc. In tutti questi, il rilevamento delle variazioni di temperatura sarebbe più accurato poiché alcuni incendi non produrranno nemmeno fumo rilevabile.

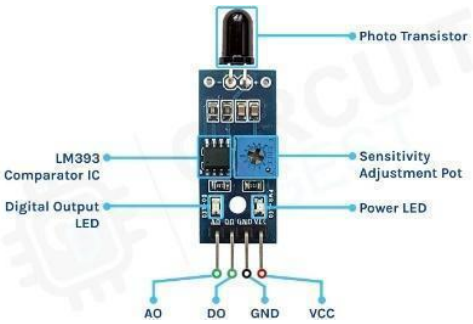
3.1 Informazioni sul sensore di fiamma

Il fotodiodo IR rileva la radiazione IR proveniente da qualsiasi corpo caldo e confronta questo valore con un valore specificato.

Possiamo visualizzare il valore dell'irradiazione sull'uscita analogica del sensore oppure, una volta che l'irradiazione raggiunge un valore di soglia, il sensore modificherà di conseguenza la sua uscita digitale.

Il modulo sensore di fiamma è composto da pochissimi componenti, tra cui un fotodiodo IR, un circuito integrato comparatore LM393 e alcuni componenti passivi.

Il LED di alimentazione si accende quando il modulo è alimentato e il LED D0 si spegne quando viene rilevata una fiamma. La sensibilità può essere regolata tramite il trimmer di resistenza integrato.



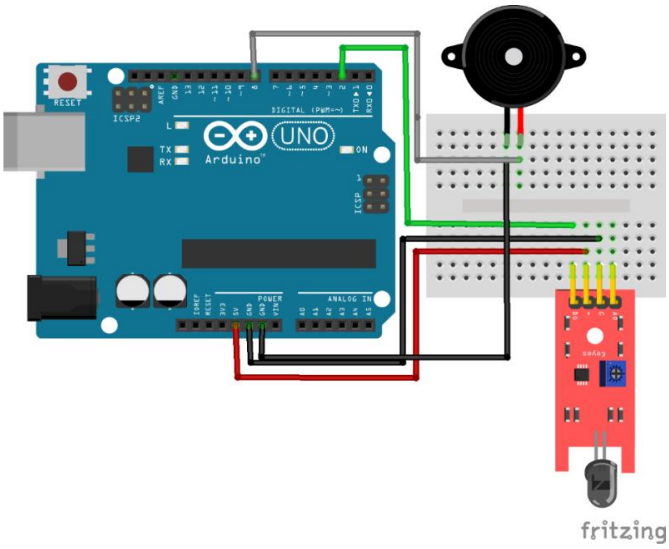
3.2 Circuito 3

Titolo del circuito: Rilevatore di fiamma

Descrizione del circuito: Il circuito monitora l'area. Se viene rilevata una fiamma, attiva il cicalino e l'allarme suona.

Cosa ti servirà:

a) Breadboard
b) Sensore di fiamma
c) Cicalino passivo
d) Arduino



Alimentare il sensore con una tensione di 5 volt. Il segnale digitale è collegato al pin 2 dell'Arduino. Il buzzer è collegato al pin 13 dell'Arduino.

Quando viene rilevata una fiamma, il sensore invia un "1" logico all'ingresso Arduino e si attiva, tramite l'uscita 8, il buzzer con una frequenza sonora di 1KHz

3.3 Il Programma

```
int flameSens = 2; //flameSens al pin 2 di Arduino
int buzzerPin = 8; //cicalino sul pin 8 di Arduino
void setup (){
  pinMode (flameSens , INPUT ); //inizializza il pin di uscita del sensore di fiamma collegato
come input
  pinMode (buzzerPin , OUTPUT ); // inizializza il pin digitale buzzerPin come output
  Serial . begin (9600); // inizializza la comunicazione seriale a 9600 baud
}
void loop (){
  if ( digitalRead (flameSens) == 1 )
{
    tone (buzzerPin , 1000); //Invia segnale sonoro a 1 KHz
    Seriale . println ( "** Attenzione!!!! Incendio rilevato!!! **" );
}
  else
{
    digitalWrite ( LED_BUILTIN , BASSO ); // Led spento
    noTone (buzzerPin); //interrompe il suono
    Seriale . println ( "Nessun incendio rilevato" );
```

```
}
  delay (100);
}
```

tips

Utilizzando la funzione tone(), puoi creare suoni diversi. Prova a generare frequenze sonore di 2400 Hz e 2900 Hz con un ritardo di 1000 secondi tra loro.

4. Sensore di umidità del terreno

I sensori di umidità del terreno sono molto diffusi in agricoltura, paesaggistica e giardinaggio. Le tecnologie di rilevamento dell'umidità offrono vantaggi ai coltivatori, consentendo loro di risparmiare tempo e denaro sui sistemi di irrigazione manuale.

Possono aiutare agricoltori e giardinieri a controllare i livelli di umidità in modo rapido, preciso e conveniente. Grazie a questo, puoi facilmente determinare quando innaffiare o irrigare il tuo campo.

L'analisi del terreno mediante un sensore riduce l'incertezza fornendo dati in tempo reale, consentendo di prendere decisioni più consapevoli su come gestire risorse come l'acqua.

4.1 Informazioni sul sensore di umidità del suolo

Il sensore di umidità del terreno funziona in modo semplice.
Il sensore è costituito da una sonda a forcella con due conduttori esposti, inserita nel terreno o nel punto in cui si desidera misurare il contenuto di umidità.

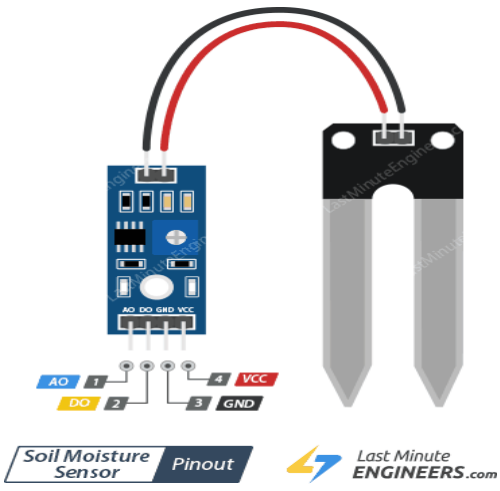
Questi agiscono come resistori variabili (simili ai potenziometri) la cui resistenza varia in base al contenuto di umidità del terreno. Questa resistenza varia inversamente all'umidità del terreno.

Inoltre, il sensore include un modulo elettronico che collega la sonda all'Arduino.

Il modulo genera una tensione di uscita basata sulla resistenza della sonda, disponibile su un pin di uscita analogica (AO).

Lo stesso segnale viene inviato a un comparatore ad alta precisione LM393, che lo digitalizza e lo rende disponibile su un pin di uscita digitale (DO).

Il modulo include un potenziometro per regolare la sensibilità dell'uscita digitale (DO).



Il sensore di umidità del terreno necessita solo di quattro pin per la connessione.

AO (Analog Output) genera una tensione di uscita analogica proporzionale al livello di umidità del terreno.

DO (uscita digitale) D0 diventa LOW quando il livello di umidità supera il valore di soglia (impostato dal potenziometro) e HIGH in caso contrario.

(VCC) fornisce alimentazione al sensore. Si consiglia di alimentare il sensore con una tensione compresa tra 3,3 V e 5 V. Si prega di tenere presente che l'uscita analogica varia a seconda della tensione fornita al sensore.

GND è il pin di terra

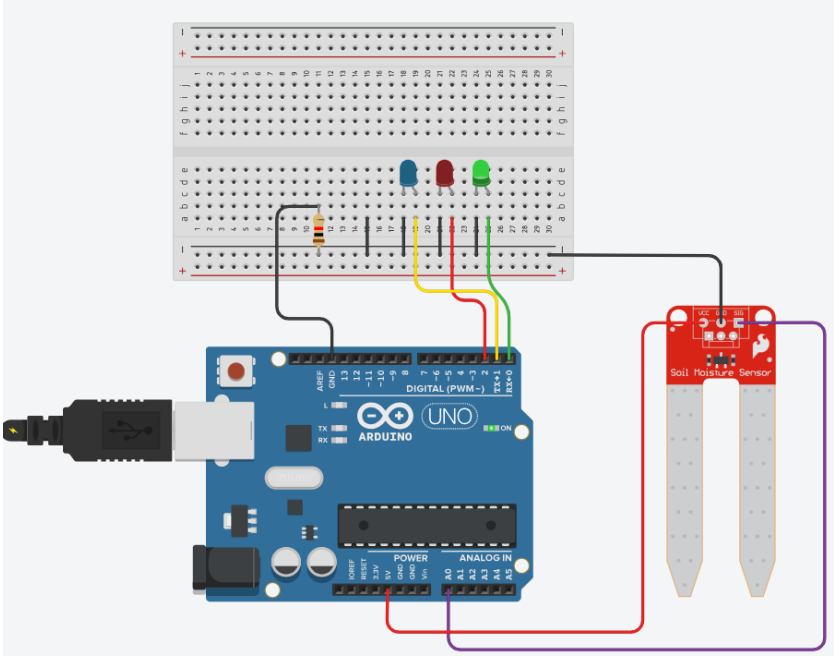
4.2 Circuito 4

Titolo del circuito: Rilevatore di umidità del suolo

Descrizione del circuito: Il circuito misura l'umidità del terreno in percentuale rispetto a un valore dato. Se l'umidità è compresa tra 0 e 30%, si accende la luce rossa; tra 30% e 60%, si accende la luce blu e sopra il 60%, si accende la luce verde.

Cosa ti servirà:

a) Breadboard
b) Sensore di umidità del suolo
c) Led verde, blu, rosso
d) è necessaria una resistenza da 220Ω per proteggere il LED
e) Arduino



Nota: all'inizio del programma, il sensore deve rilevare il livello massimo di umidità in modo da poter calcolare con precisione le percentuali durante tutto il ciclo del programma.

4.3 Il Programma

```
// Programma Arduino del sensore di umidità
int greenLight = 9; //Definizione dei pin
int luceblu = 8;
int luce rossa = 10;
                                        float maximumMoistureLevel; //Il livello massimo di
                                        umidità e i livelli di umidità attuali saranno necessari per i
                                        calcoli percentuali
                                      float currentMoistureLevel; // = Livello di umidità


void setup () {
    pinMode(8 , OUTPUT ); //avvia il pin 8 come output
   pinMode (9 , OUTPUT ); //avvia il pin 9 come output
   pinMode(10 , OUTPUT ); //avvia il pin 10 come output
   pinMode (A1 , INPUT ); //A1 è il pin utilizzato per il sensore di umidità del terreno
delay (100);
Livello di umidità massimo = analogRead (A1)
                                           digitalWrite (luce verde ,   HIGH ); //tutti i LED
                                        si accendono per 2 secondi per indicare che il
                                        programma è stato avviato.
   digitalWrite (luce blu , ALTO );
                                           digitalWrite (luce rossa , ALTO );  ;
```

```
  delay (100);
  digitalWrite (luce verde ,  BASSO );
  digitalWrite (luce blu , BASSO );
  digitalWrite (luce rossa , BASSO );
  Serial . begin (9600);
}

 void  loop() {
                                                    if (maximumMoistureLevel / currentMoistureLevel <= 0.3) //se il livello di umidità è inferiore al 30%
{
  digitalWrite (luce verde ,  BASSO );
  digitalWrite (luce blu , BASSO );
                                                digitalWrite (luce rossa , HIGH ); //Accendi la luce rossa, ma non far suonare il cicalino
}
    else if (maximumMoistureLevel / currentMoistureLevel <= 0.8 &&
    maximumMoistureLevel / currentMoistureLevel > 0.3) // se il livello di umidità è compreso tra il 30 e il 60%
   {
  digitalWrite (luce verde , BASSO );
  digitalWrite (luce blu ,  HIGH ); //Accendi semplicemente la luce gialla
  digitalWrite (luce rossa , BASSO );
}
    else   //Altrimenti il livello di umidità è superiore al 60%, e quindi è abbastanza buono
{
  digitalWrite (luce verde , HIGH ); //Accendi la luce verde
  digitalWrite (luce blu ,  BASSO );
  digitalWrite (luce rossa , BASSO );
}
livello di umidità attuale = analogRead (A1); //rinnovo delle misurazioni

  delay (500); //Breve ritardo per non sovraccaricare il programma
Serial.println( "maximumMoistureLevel" ); // Solo per poter vedere il livello massimo di umidità come una lettura tra 0-1023
Serial.println(livello di umidità massimo);
  Serial.println( "currentMoistureLevel" ); //Solo per farti vedere il livello di umidità come una lettura tra 0-1023
  Serial.println(livello di umidità corrente);
}
```

suggerimenti

Il sensore di umidità ha uno svantaggio: essendo costantemente alimentato in un ambiente umido, ha una durata breve. Pertanto, si consiglia di utilizzarlo solo quando è necessario effettuare una misurazione.

5. Protezione attiva antincendio (AFP)

5.1 Informazioni sulla protezione attiva antincendio (AFP)

La Protezione Antincendio Attiva (AFP) si riferisce a un insieme di sistemi che richiedono un intervento per funzionare efficacemente in caso di incendio. Questi interventi possono essere manuali, come l'utilizzo di un estintore, o automatizzati, come un sistema sprinkler. Indipendentemente dal metodo utilizzato, è necessario un intervento per il corretto funzionamento di questi sistemi.

Un tipico sistema di protezione antincendio attiva rileva l'incendio, attiva l'allarme e quindi attiva il sistema di spegnimento automatico.

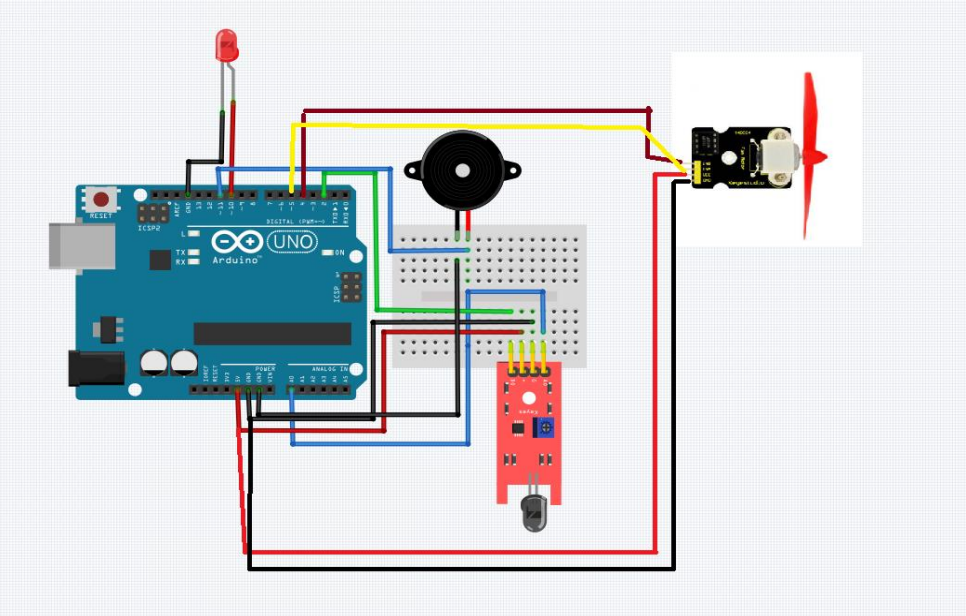
5.2 Circuito 5

Titolo del circuito: Protezione antincendio attiva (AFP)

Descrizione del circuito: Il circuito rileva la presenza di un incendio tramite un sensore di fiamma. Quando viene rilevata una fiamma, vengono attivati sia un allarme acustico che un avviso visivo , e l'allarme emetterà quattro suoni (cicalino + LED). Se la fiamma persiste, la ventola viene attivata per 4 secondi per spegnerla. Se la fiamma continua a essere presente, il processo si ripete.

Cosa ti servirà:

a) Breadboard
b) sensore di fiamma
c) Led rosso
d) è necessaria una resistenza da 220Ω per proteggere il LED
e) modulo buzzer passivo
f) ventola del modulo
g) Arduino
h) lumino da tè



Circuito di pinout

Pin 11 □ buzzer

Pin 10 □ Led rosso

Pin4 □ INA

Pin 5 □ INB

Nota: il nuovo componente nel circuito è il modulo ventola, che ha quattro pin: Vcc, Gnd, INA e INB. Una tensione di 5 volt è collegata a Vcc, mentre la massa è collegata a Gnd. I pin INA e INB sono utilizzati per il controllo del motore. Quando INA è basso e INB è basso, il motore si ferma. Quando INA è basso e INB è alto, il motore gira in senso orario.

5.3 Il Programma

```
 int flameSens = A0; //sensore di fiamma al pin 2 di Arduino
int buzzerPin = 11; //cicalino sul pin 8 di Arduino
int ledPin = 10; //LED sul pin 8 di Arduino
int INA = 4; // pin INA shelld motore su pin4 arduino
int INB = 5; // pin INB shelld del motore al pin 5 di Arduino
int flame = 0; // variabile per memorizzare lo stato del sensore (valore)

void setup (){
 pinMode(flameSens , INPUT ); //inizializza il pin di uscita del sensore di fiamma come input
per Arduino
 pinMode(buzzerPin , OUTPUT ); // inizializza il pin del buzzer come output
 pinMode(ledPin , OUTPUT ); // inizializza il pin led come output
 Serial.begin(9600); // inizializza la comunicazione seriale a 9600 baud
 pinMode(INA , OUTPUT ); // inizializza il pin led come output
 pinMode(INB , OUTPUT ); // inizializza il pin led come output
}

void loop () {
fiamma = analogRead (flameSens);
  Serial.println( "flame arxh" );
 Serial.println(fiamma); //visualizzazione della misurazione del sensore

if (fiamma < 100) // leggi il valore del sensore
{
 for ( int i = 1; i <= 4; i ++ ) // la sveglia suona 4 volte
{
   digitalWrite (ledPin , HIGH ); // accendi il LED
   tone (buzzerPin , 2400); //il buzzer suona a 2400 KHz
   delay (1000);
   digitalWrite (ledPin , LOW ); // spegne il LED
   tone (buzzerPin , 2900); //il buzzer suona a 2900 KHz
   delay (1000);
   Serial.println( "ciclo di fiamma" );
  Serial.println(fiamma);
}
  digitalWrite (INA , LOW ); // accende la ventola in senso orario
 digitalWrite (INB , HIGH ); // accende la ventola in senso orario
 Serial.println( "motore di fiamma" );
 Serial.println(fiamma);
 delay (4000);
}
digitalWrite (INA , LOW ); // accende la ventola in senso orario
 digitalWrite (INB , LOW ); // accende la ventola in senso orario
 noTone (cicalino);
 Serial.println( "flametelos" ); //visualizzazione della misurazione del sensore
 Serial.println(fiamma);
```

}

Operazione

Quando viene rilevata una fiamma, l'allarme suonerà (cicalino + LED) quattro volte, quindi la ventola si accenderà per 4 secondi.

6. Sistema di irrigazione automatica delle piante

Utilizzare un sistema di irrigazione automatico è un modo efficiente per risparmiare acqua e prendersi cura del proprio giardino. Questo sistema fornisce l'esatta quantità d'acqua necessaria alle piante, a differenza dell'irrigazione manuale o dell'utilizzo di un sistema di irrigazione a pioggia, che può facilmente causare eccessi o carenze d'acqua.

6.1 Titolo del circuito: Sistema automatico di irrigazione delle piante

Descrizione del circuito: Questo circuito misura il livello di umidità del terreno. Se il livello di umidità scende al di sotto di una soglia predeterminata, la valvola si apre, consentendo all'acqua di fluire verso la pianta e il LED verde si accende. Quando il livello di umidità supera la soglia specificata, la valvola si chiude per interrompere l'irrigazione e il LED blu si accende.

Cosa ti servirà:
a) Breadboard

b) sensore di umidità (con modulo)

c) Led verde, led blu

d) Per proteggere il LED è necessaria una resistenza da 220Ω.

e) elettrovalvola

f) piccolo serbatoio d'acqua

g) Arduino

h) piccolo vaso con pianta

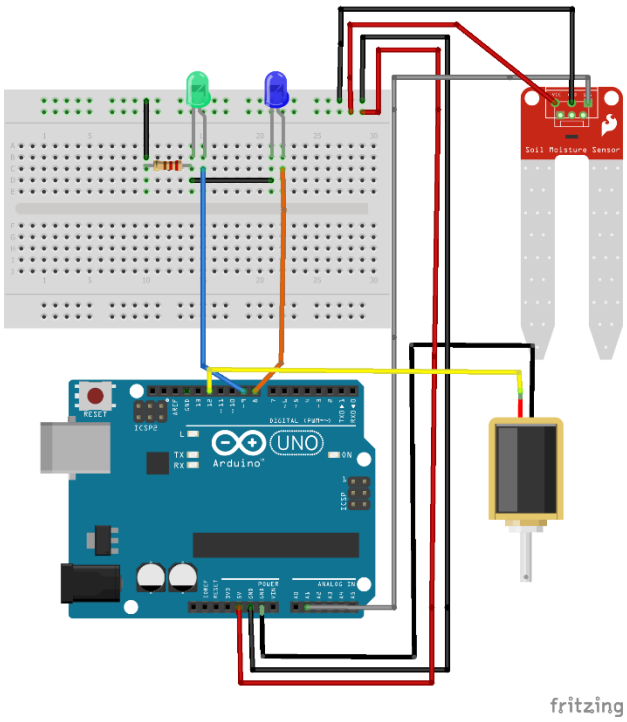
Circuito di pinout

Pin 8 □ led, verde

Pin 9 □ led blu

Sensore di umidità con ingresso analogico A1 □

Valvola pin 12 □



6.3 Il Programma

```
// Programma Arduino del sensore di umidità

int greenLight = 9; //Definizione dei pin
int blueLight = 8; //Definizione dei pin
int moistureLevel = A1; //Definizione dei pin
int valvePin = 10; //Definizione dei pin
int maxMoistureLevel; //Il livello massimo di umidità
int minMoistureLevel; //Il livello massimo di umidità
int currentMoistureLevel; // livello di umidità attuale


void setup () {
  pinMode (blueLight , OUTPUT ); //avvia il pin come output
  pinMode (greenLight , OUTPUT ); //avvia il pin 9 come output
   pinMode (livello di umidità , INPUT ); //A1 è il pin utilizzato per il sensore di umidità del terreno
   pinMode (valvePin , OUTPUT ); //inizializza il pin 9 come output
   Serial.begin (9600);); // inizializza la comunicazione seriale a 9600 baud:
}
void  loop () {
maxMoistureLevel = 700; //imposta la variabile a 700
minMoistureLevel = 400; //imposta la variabile a 400
currentMoistureLevel = analogRead (moistureLevel); //Assegna alla variabile il valore corrente
  if (currentMoistureLevel > 1000) //se il livello di umidità è inferiore a 1000 si accendono le luci verde e blu
```

```
{
   digitalWrite (greenLight , HIGH ); //Accendi la luce
   digitalWrite (blueLight , HIGH ); //Accendi la luce
   digitalWrite (valvePin , LOW ); //Spegni la luce
}
  else if ((currentMoistureLevel > minMoistureLevel) && (currentMoistureLevel < 1000)) //se il
livello di umidità è compreso tra il 30 e il 60%
{
   digitalWrite (luce verde , ALTO );
   digitalWrite (luce blu ,   BASSO ); // luce led accesa
   delay (3000);
   digitalWrite (valvePin , HIGH ); // aprire la valvola
   delay (500);
   digitalWrite (valvePin , LOW ); // chiudi la valvola
   delay (300);
   digitalWrite (luce verde , LOW ); // luce led spenta
}
  if (currentMoistureLevel > maxMoistureLevel) { //se il livello di umidità è inferiore al 30%
   digitalWrite (luceblu , ALTA );
}

  delay (500); //Breve ritardo
  Serial.println ( "Livello di umidità" );
  Serial.println (livello di umidità corrente);
}
```

7. Luce di sicurezza

Le luci di sicurezza sono luci elettriche, solitamente esterne a un edificio, che si accendono quando qualcuno o qualcosa si muove nelle loro vicinanze. I ladri cercano di entrare in casa nelle prime ore del mattino, ma i vicini possono individuarli quando le luci di sicurezza si accendono. Questo sistema dovrebbe essere più efficiente aggiungendo un sensore di luminosità; la luce si accenderà solo quando la luce circostante è scarsa, risparmiando energia.

7.1 Titolo del circuito: Luce di sicurezza

Descrizione del circuito: Questo circuito è dotato di due sensori: un sensore di movimento (PIR) e un sensore di luminosità. Quando il livello di luminosità dell'area monitorata è basso e viene rilevato un movimento, la luce si accende per tre secondi.

Cosa ti servirà:
a) Breadboard

b) fotoresistore

c) modulo Led

d) Resistenza da 10K

e) sensore PIR

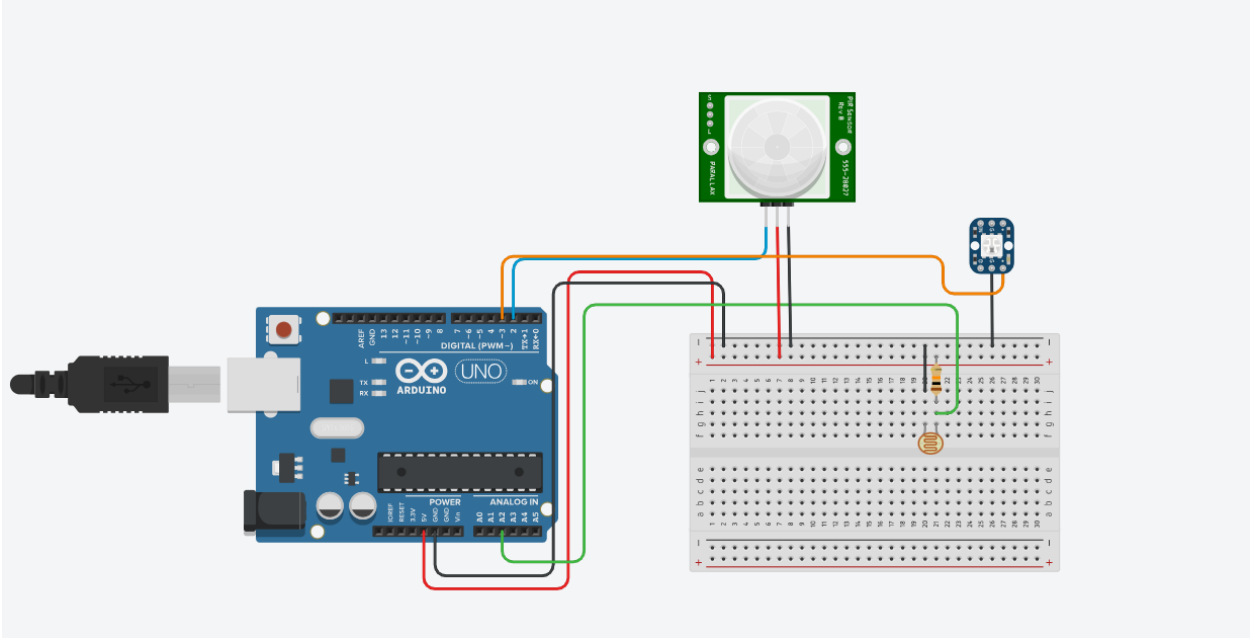
g) Arduino

Circuito di pinout

Pin 2 □ Ingresso PIR

Pin A2 □ sensore di luminosità in ingresso

Pin 3 □ Led



7.3 Il Programma

```
//Costanti
cost int pResistor = A2; // Fotoresistore sul pin analogico A2 di Arduino
cost int ledPin = 3; // Pin LED sul pin 9 di Arduino

//Variabili
int valore; // Memorizza il valore dal fotoresistore (0-1023)

void setup (){
 pinMode (ledPin , OUTPUT ); // Imposta lepPin - 9 pin come uscita
 pinMode (pResistore , INPUT ); // Imposta il pin pResistor - A0 come input (facoltativo)
 Serial.begin (9600); //velocità con cui Arduino comunica con il laptop
}

void loop (){
valore = analogRead (pResistor); //legge i dati analogici dal sensore

  if (valore < 700){
    digitalWrite (ledPin , LOW ); //Spegni il led
}
  else {
    digitalWrite (ledPin , HIGH ); //Accendi il LED
}

  delay (500); //Piccolo ritardo
  Serial . println (valore); //stampa sul monitor seriale
}
```

8. Applicazione

Tutti i progetti di sensori possono essere applicati alla costruzione corrispondente. Il pinout per questa costruzione è identico a quello del progetto individuale.

Circuito di inout

Pin A0 □ sensore di fiamma

Pin 2 □ Uscita PIR

Pin 3 □ Modulo Led

Pin4 □ Motore INA

Pin 5 □ Motore INB

Pin 8 □ led, verde

Pin 9 □ led blu

Pin 10 □ Led rosso

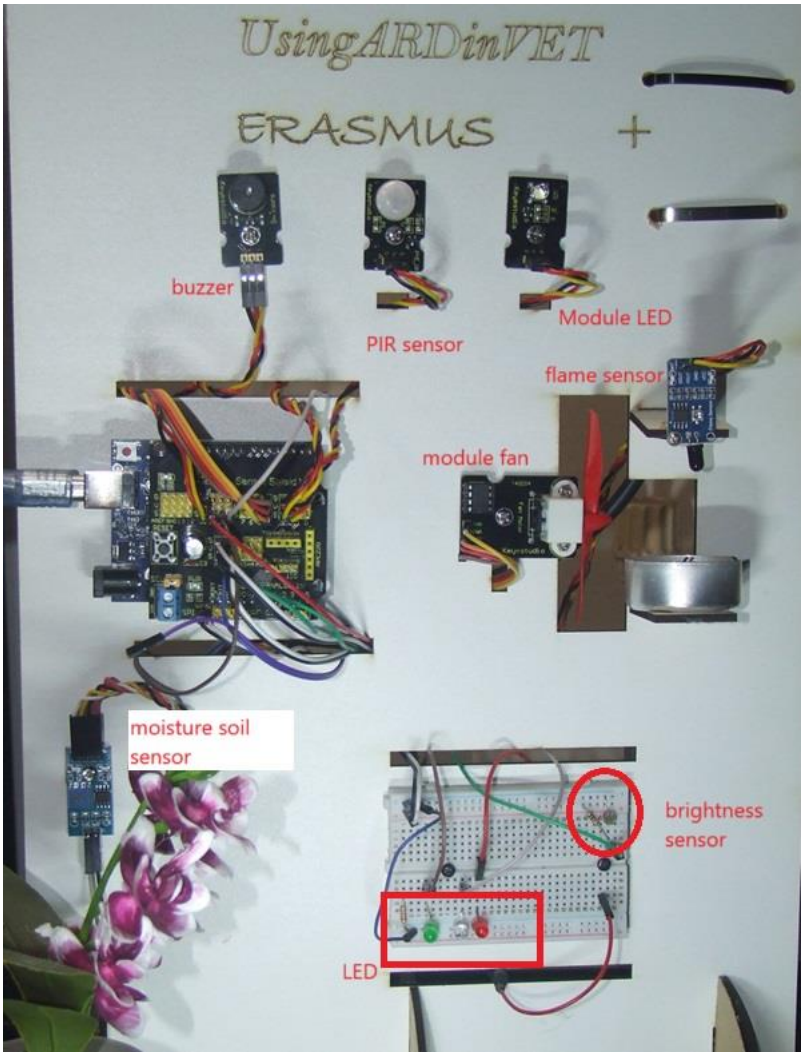
Pin 11 □ buzzer

Valvola pin 12 □

Pin A0 □ sensore di fiamma con ingresso analogico

Pin A1 □ sensore di umidità con ingresso analogico

Pin A2 □ sensore di luminosità di ingresso analogico



Con il seguente programma è possibile eseguire **contemporaneamente progetti come sistema di irrigazione automatica delle piante, protezione antincendio attiva** e luci di sicurezza, senza dover caricare i rispettivi programmi singolarmente.

```
// programma completo
int sensorPir = 2; //Sensore PIR sul pin 2 di Arduino
int ledPin = 3; // Pin LED sul pin 9 di Arduino
int INA = 4; // pin INA shelld motore su pin4 arduino
int INB = 5; // pin INB shelld del motore al pin 5 di Arduino
int blueLight = 8; //Definizione dei pin
int greenLight = 9; //Definizione dei pin
int redLedPin = 10; //Definizione dei pin
int buzzerPin = 11; //Definizione dei pin
```

```
int valvePin = 10; //Definizione dei pin
int flameSens = A0; //sensore di fiamma al pin A0 di Arduino
int moistureLevel = A1; //sensore di umidità al pin A1 di Arduino
int sensorPhotores = A2; // Fotoresistore sul pin analogico A2 di Arduino
int maxMoistureLevel; //variabile per il livello massimo di umidità
int minMoistureLevel; //variabile per il livello minimo di umidità
int currentMoistureLevel; // variabile per il livello di umidità attuale
int valuePhoto; //variabile per memorizzare il valore della fotoresistenza (0-1023)
int valuePir; // variabile per memorizzare il valore del sensore PIR (0-1023)
int flame = 0; // variabile per memorizzare lo stato del sensore (valore)

void setup () {
    pinMode (ledPin , OUTPUT ); // Imposta lepPin come output
    pinMode (sensorPhotores , INPUT ); // Imposta il pin sensorPhotores come input
    pinMode (sensorePir , INPUT ); // Imposta il pin sensorPhotores come input
    pinMode (flameSens , INPUT ); //inizializza il pin di uscita del sensore di fiamma come input
per Arduino.
    pinMode (buzzerPin , OUTPUT ); // inizializza il pin del buzzer come output
    pinMode (redLedPin , OUTPUT ); // inizializza il pin redled come output
    pinMode (INA , OUTPUT ); // inizializza il pin della ventola INA come uscita
    pinMode (INB , OUTPUT ); // inizializza il pin della ventola INB come uscita
    pinMode (blueLight , OUTPUT ); //avvia il pin della luce blu come output
    pinMode (greenLight , OUTPUT ); //avvia il pin della luce verde come output
    pinMode (livello di umidità , INPUT ); //A1 è il pin utilizzato per il sensore di umidità del
terreno
    pinMode (valvePin , OUTPUT ); //inizializza il pin 9 come output
maxMoistureLevel = 700; //imposta la variabile a 700
minMoistureLevel = 400; //imposta la variabile a 400
    Serial . begin (9600); // inizializza la comunicazione seriale a 9600 baud:
}
void securityLight(){ //metodo per la luce di sicurezza
   digitalWrite (ledPin , HIGH ); // accende il LED
```

```
  ritardo (5000);
  digitalWrite (ledPin , LOW ); // spegne il LED
}
void flameSens1(){ //metodo per il fuoco
  for ( int i = 1; i <= 4; i ++ ) // suona l'allarme 4 volte
{
    digitalWrite (redLedPin , HIGH ); // accendi il LED
    tone (buzzerPin , 2400); //il buzzer suona a 2400 KHz
    ritardo (1000);
    digitalWrite (redLedPin , LOW ); // spegne il LED
    tone (buzzerPin , 2900); //il buzzer suona a 2900 KHz
    ritardo (1000);
    Seriale . println ( "ciclo fiamma" ); //visualizza la variabile 'fiamma'
    Seriale . println (fiamma);
}
    digitalWrite (INA , LOW ); // accende la ventola in senso orario
    digitalWrite (INB , HIGH ); // accende la ventola in senso orario
    noTone (buzzerPin); // ferma il cicalino
    ritardo (4000);
    digitalWrite (INA , LOW ); // spegni la ventola
    digitalWrite (INB , LOW ); // spegni la ventola

}
void watering(){ //metodo per l'irrigazione
   if (currentMoistureLevel > 1000) //se il sensore di umidità non è a terra

{
      digitalWrite (greenLight , HIGH ); //Accendi la luce, mostra che il sensore non è a terra
      digitalWrite (blueLight , HIGH ); //Accendi la luce , mostra che il sensore non è a terra
      digitalWrite (redLedPin , LOW ); //Spegni la luce , mostra che il sensore non è a terra
}
```

```
    else if ((currentMoistureLevel > minMoistureLevel) && (currentMoistureLevel < 1000)) //se il livello di umidità è inferiore al livello minimo
{
        digitalWrite (luce verde , HIGH ); // luce led accesa
        digitalWrite (luce blu ,   LOW ); // luce led spenta
        ritardo (3000);
        digitalWrite (redLedPin , HIGH ); // aprire la valvola
        ritardo (500);
        digitalWrite (redLedPin , LOW ); // chiudi la valvola
        ritardo (300);
        digitalWrite (luce verde , LOW ); // luce led spenta
}
      if (currentMoistureLevel > maxMoistureLevel) { //se il livello di umidità è superiore al livello minimo
       digitalWrite (blueLight , HIGH ); // luce led accesa
}

      Serial . println ( "Livello di umidità" ); // visualizza l'umidità
      Seriale . println (livello di umidità corrente);
}



void loop () {
valoreFoto = analogRead (sensorPhotores); //Assegna alla variabile il valore corrente
valuePir = digitalRead (sensorPir); //Assegna alla variabile il valore corrente
flame = analogRead (flameSens); //Assegna alla variabile il valore corrente
currentMoistureLevel = analogRead (moistureLevel); //Assegna alla variabile il valore corrente
  if (valuePhoto > 800 && valuePir == HIGH ) //se c'è oscurità e movimento
{
lucedisicurezza();
}
```

```
 if fiamma < 450) //se c'è fuoco
{
fiammaSens1();
}


 if ((currentMoistureLevel > minMoistureLevel) && (currentMoistureLevel < 1000)) //se il
livello di umidità è inferiore al livello minimo e //se il sensore di umidità è a terra
{
irrigazione();
}
  Serial . println ( "valuePhoto" ); //visualizza la variabile 'valuePhoto'
 Serial . println (valuePhoto); //visualizza la variabile 'valuePhoto'
 Serial . println ( "fiamma" ); //visualizza la variabile 'valuePhoto'
 Serial . println (flame); //visualizza la variabile 'valuePhoto'
 delay (500);
}
```

Co-funded by the
Erasmus+ Programme
of the European Union

Allegato

Co-funded by the
Erasmus+ Programme
of the European Union

Per maggiori informazioni, visita il sito web del nostro progetto:

www.ardinvet.com

Co-funded by the
Erasmus+ Programme
of the European Union

Questo progetto è finanziato dal programma Erasmus+ dell'Unione Europea. Tuttavia, la Commissione Europea non può essere ritenuta responsabile per qualsiasi uso che possa essere fatto delle informazioni in esso contenute.

BIBLIOGRAFIA:

1. Arduino: la guida definitiva ad Arduino per principianti, con istruzioni per la programmazione, ricettari, suggerimenti, trucchi e molto altro! Craig Newport;

2. Introduzione ai microcontrollori e ai controllori logici programmabili - Autore Viorel-Constantin Petre, Editore: Matrixrom Publishing House;

3. Arduino: Introduzione ad Arduino e programmazione di base con progetti (Metodi avanzati per apprendere la programmazione Arduino) - Ernest Leclerc;

4. Programmazione Arduino – Editura Nelly BL International Consulting LTD., marzo 2020.