

Corso Coding con Arduino



Arduino nasce nel 2005 come progetto **open-source** destinato principalmente agli studenti di design e ingegneria. Il fondatore principale è **Massimo Banzi**, un ingegnere italiano, che ha progettato Arduino per semplificare la **prototipazione elettronica**. Nel corso degli anni, Arduino è diventato uno degli strumenti di prototipazione più popolari al mondo grazie alla sua facilità d'uso e alla comunità di sviluppatori attivi.



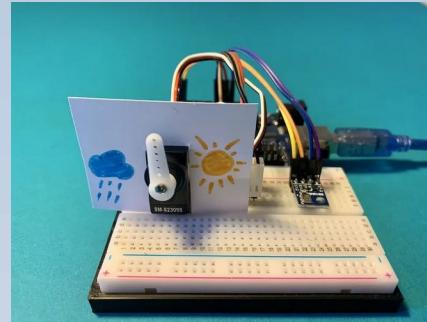
Arduino nella didattica

Arduino rappresenta una risorsa educativa estremamente potente, in grado di trasformare l'insegnamento della tecnologia, della matematica e delle scienze in un'esperienza pratica e coinvolgente.

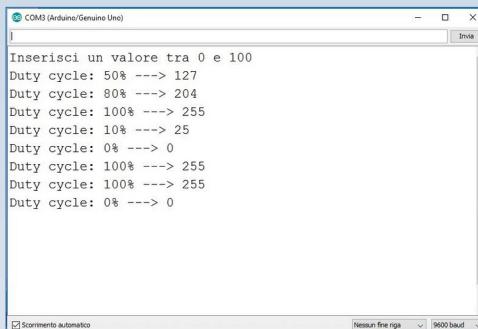
Utilizzando Arduino, gli studenti non solo apprendono nozioni teoriche, ma imparano anche a metterle in pratica attraverso la progettazione e la costruzione di circuiti, scrittura di codice, e creazione di progetti interattivi.

Arduino nella didattica

- **Apprendimento pratico:**
 - Teoria e pratica integrate
 - Sperimentazione
- **Sviluppo del pensiero critico:**
 - Analizzare e risolvere problemi
 - Testare e modificare
 - Gestire la complessità
- **Sviluppo della creatività:**
 - Progetti personalizzati
 - Esplorazione di idee



Com'è fatto Arduino



A screenshot of a computer screen showing the serial monitor window for an Arduino connection named "COM3 (Arduino/Genuine Uno)". The window displays a series of text messages indicating the mapping of duty cycles to digital pin values:

```
Inserisci un valore tra 0 e 100
Duty cycle: 50% ---> 127
Duty cycle: 80% ---> 204
Duty cycle: 100% ---> 255
Duty cycle: 10% ---> 25
Duty cycle: 0% ---> 0
Duty cycle: 100% ---> 255
Duty cycle: 0% ---> 0
```

The window also includes standard serial monitor controls at the bottom: "Scorrimento automatico" (Auto scroll), "Nessun fine riga" (No line break), and "9600 baud".

- Arduino è composto da una serie di componenti che permettono di interagire con il mondo esterno (sensori, motori, LED, etc.) tramite i pin di **input** e **output**. Il **microcontrollore** è il cuore del sistema e può essere programmato tramite un computer.
- Pin digitali e analogici: I **pin digitali** possono leggere o inviare segnali accesi/spenti (HIGH/LOW) e PWM tra 0 e 255 (8 bit), mentre i **pin analogici** possono leggere segnali con valori compresi tra 0 e 1023 (10 bit).
- Alimentazione: Può essere **alimentato tramite USB** o una fonte esterna (ad esempio una batteria o alimentatore esterno).
- Comunicazione seriale: Viene utilizzata per il **debugging** o per **comunicare con il computer**, ad esempio, per leggere i valori dei sensori tramite il monitor seriale.

Differenze con altri hardware simili

- Arduino vs Raspberry Pi: **Raspberry Pi** è un computer vero e proprio che può eseguire un **sistema operativo** (Linux), mentre Arduino è un microcontrollore che esegue un singolo programma. Raspberry Pi è più potente e adatto a **compiti complessi** come il multimedia, mentre Arduino è più adatto per interagire con dispositivi elettronici tramite pin di I/O.
- Arduino vs ESP32: L'**ESP32** è un **microcontrollore** che include supporto nativo per **Wi-Fi** e **Bluetooth**, mentre Arduino base non ha questa funzionalità senza moduli aggiuntivi. L'ESP32 è più potente, ma richiede più risorse per la programmazione e gestione.



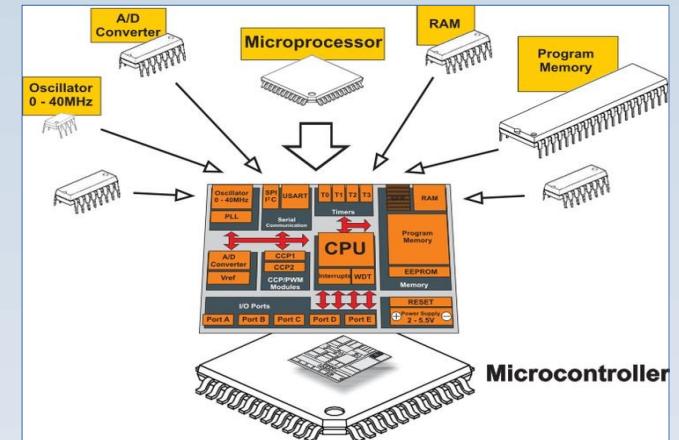
Attività pratica !

Migliorare e aumentare le conoscenze tecniche di questi dispositivi utilizzando motori di ricerca o interfacce di IA:

Componenti interni e/o esterni (memoria, oscillatore, ADC/DAC), vantaggi, potenzialità, anche alla luce del consumo energetico, funzioni *sleep*.



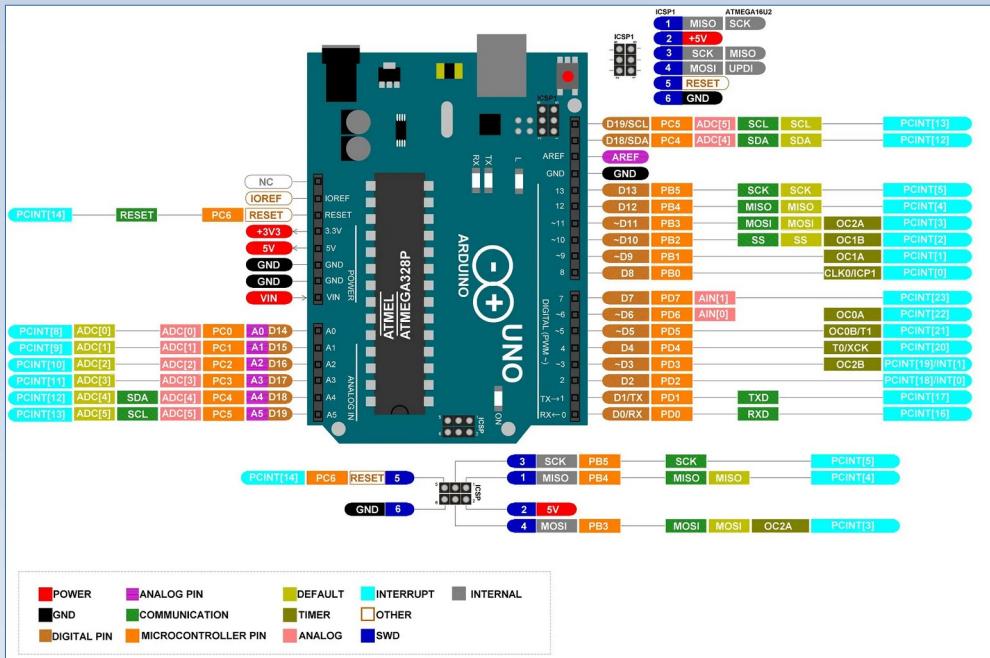
Arduino	Microcontrollore	Frequenza		Memoria			Connettori di I/O			Tipo interfaccia USB	Dimensioni in pollici	Dimensioni in millimetri	Anno di introduzione ^[21]
		MHz		Flash KB	EEPROM KB	SRAM KB	Pin di I/O digitale	...di cui con PWM	Pin di Input analogico				
Diecimila	ATmega168		16	0,5	1	14	6	6		FTDI	2,7 x 2,1	68,6 x 53,3	2007
Due ^[22]	Atmel SAM3X8E		512		96	54	12		12	ATmega16U2 + native host	4 x 2,1	101,6 x 53,3	
Duemilano	ATmega168/328P		16/32	0,5/1	1/2	14	6	6		FTDI	2,7 x 2,1	68,6 x 53,3	2008
Uno	ATmega328P	16	32	1	2	14	6	6		ATmega8U2	2,7 x 2,1	68,6 x 53,3	2010
Leonardo	Atmega32u4		32	1	2,5	20	7	12		Atmega32u4 integrato	2,7 x 2,1	68,6 x 53,3	2012
Mega	ATmega1280		128	4	8	54	14	16		FTDI	4 x 2,1	101,6 x 53,3	2009
Mega2560	ATmega2560	16	256	4	8	54	14	16		ATmega8U2	4 x 2,1	101,6 x 53,3	2009
Fio	ATmega328P		32	1	2	14	6	8		Nessuno	1,6 x 1,1	40,6 x 27,9	2010
Nano	ATmega168 o ATmega328		16/32	0,5/1	1/2	14	6	8		FTDI	1,70 x 0,73	43 x 18	2008
LilyPad	ATmega168V o ATmega328V	8	16	0,5	1	14	6	6		Nessuno	ø 2	ø 50	2007



Attività pratica !

[https://it.wikipedia.org/wiki/Arduino_\(hardware\)](https://it.wikipedia.org/wiki/Arduino_(hardware))

<https://it.wikipedia.org/wiki/Microcontrollore>

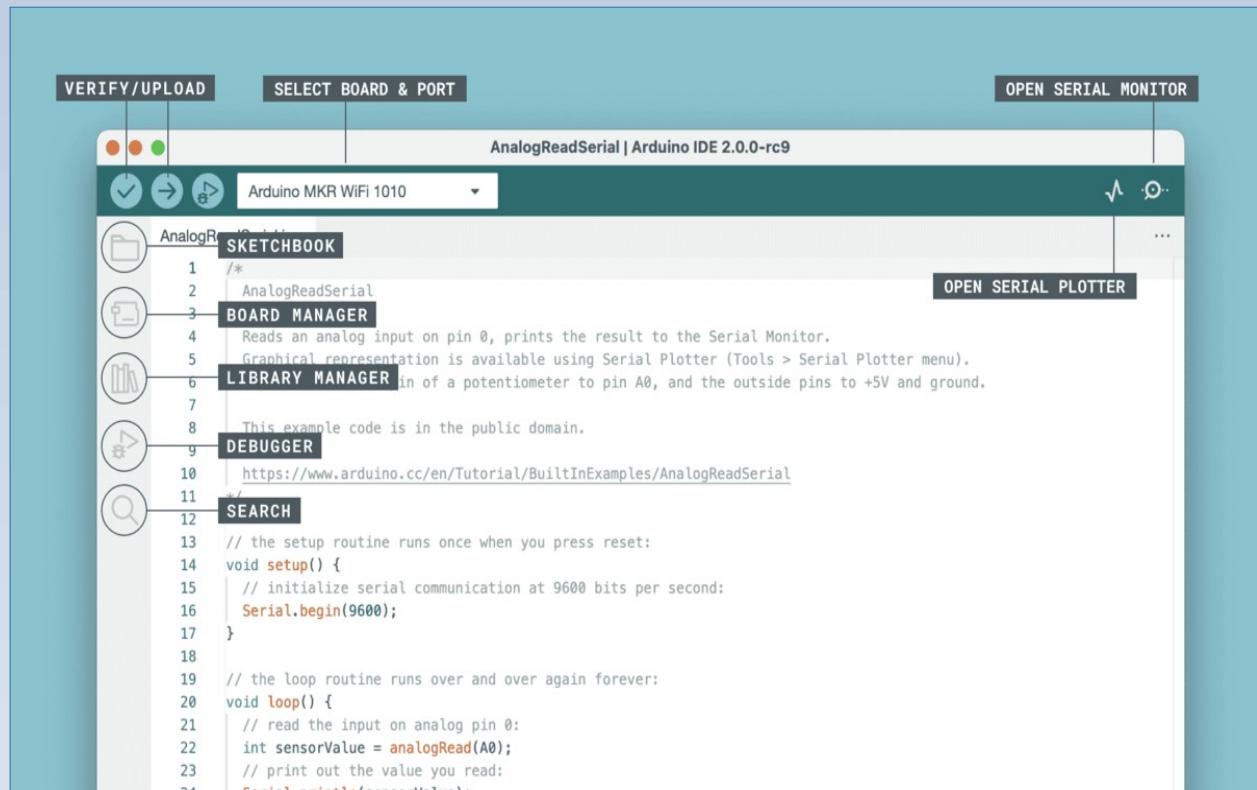


Ambiente di sviluppo integrato (IDE)

Nuova IDE 2.x

Funzioni principali:

- Verifica e caricamento.
- Monitoraggio seriale e plotter grafico.
- Cartella sketch.
- Board manager.
- Library manager.
- Selezione porta e scheda.
- Selezione velocità porta seriale.



Linguaggio di programmazione

Il linguaggio di programmazione di Arduino si basa su **C/C++**. Il suo **IDE** (Integrated Development Environment) semplifica la scrittura e il caricamento del codice nei dispositivi.

Struttura di un programma Arduino

Ogni programma Arduino si compone di due funzioni principali:

- **setup()**: Eseguito una sola volta all'avvio.
Qui vengono configurati i pin e le comunicazioni.
- **loop()**: Eseguito continuamente. Qui avvengono le operazioni principali (ad esempio, **leggere sensori** e **controllare attuatori**, eseguire calcoli, manipolare dati).

```
cpp

void setup() {
    pinMode(13, OUTPUT); // Imposta il pin 13 come uscita
}

void loop() {
    digitalWrite(13, HIGH); // Accende il LED
    delay(1000);           // Aspetta 1 secondo
    digitalWrite(13, LOW);  // Spegne il LED
    delay(1000);           // Aspetta 1 secondo
}
```

Dichiarazione di variabili

```
int temperatura = 25; // Variabile intera  
float umidita = 60.5; // Variabile decimale  
boolean allarme = false; // Variabile booleana
```

Tipi di dichiarazione	Rappresentazione	N. di byte	Intervallo
Boolean			True – False / On – Off / High - Low
Char	Carattere	1 (8 bit)	-127 +127
Byte	Carattere	1 (8 bit)	0 +255
Int	Numero intero	2 (16 bit)	-32.768 + 32.767
Unsigned int	Numero intero	2 (16 bit)	0 + 65.535
Short	Numero intero "corto"	2 (16 bit)	
Long	Numero intero "lungo"	4 (32 bit)	-2.147.483.648 + 2.147.483.647
Float	Numero reale	4 (32 bit)	-3.4028235E+38 a + 3.4028235E+38
Double	Numero reale "lungo"	8 (64 bit)	1.7976931348623157 x 10 ³⁰⁸

Dichiarazione di variabili e operatori logici

```
int a = 10;  
int b = 5;  
int somma = a + b; // somma = 15  
int differenza = a - b; // differenza = 5  
int prodotto = a * b; // prodotto = 50  
int divisione = a / b; // divisione = 2  
int resto = a % b; // resto = 0
```

Operatore	Descrizione	Esempio	Risultato
<code>==</code>	Uguaglianza	<code>a == b</code>	Restituisce true se <code>a</code> è uguale a <code>b</code>
<code>!=</code>	Diversità	<code>a != b</code>	Restituisce true se <code>a</code> è diverso da <code>b</code>
<code><</code>	Minore di	<code>a < b</code>	Restituisce true se <code>a</code> è minore di <code>b</code>
<code>></code>	Maggiore di	<code>a > b</code>	Restituisce true se <code>a</code> è maggiore di <code>b</code>
<code><=</code>	Minore o uguale a	<code>a <= b</code>	Restituisce true se <code>a</code> è minore o uguale a <code>b</code>
<code>>=</code>	Maggiore o uguale a	<code>a >= b</code>	Restituisce true se <code>a</code> è maggiore o uguale a <code>b</code>

OOP - Programmazione orientata agli oggetti

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

Funzione

```
class Calcolatrice {  
public:  
    int somma(int a, int b) {  
        return a + b;  
    }  
};
```

Classe

Funzione -> metodo

```
Calcolatrice calc;  
int risultato = calc.somma(5, 3); // risultato sarà 8
```

Oggetto (istanza)

Programmazione imperativa e funzionale

```
int somma = 0;
for (int i = 1; i <= 10; i++) {
    somma += i;
}
Serial.println(somma); // Stampa la somma
```

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

int somma_tutti(int arr[], int n) {
    int risultato = 0;
    for (int i = 0; i < n; i++) {
        risultato = somma(risultato, arr[i]);
    }
    return risultato;
}
```

Paradigmi di programmazione

Differenze principali tra OOP, Programmazione Imperativa e Funzionale

Caratteristica	OOP (Programmazione Orientata agli Oggetti)	Imperativa	Funzionale
Focalizzazione	Oggetti (classi, istanze, metodi)	Azioni e modifiche dello stato del programma	Funzioni pure, trasformazione dei dati
Stato	L'oggetto incapsula lo stato	Lo stato è modificato direttamente	Dati immutabili, no cambiamenti di stato
Modularità	Alto grado di modularità grazie a classi e oggetti	Bassa modularità, codice sequenziale	Alto grado di modularità grazie a funzioni
Ereditarietà e Polimorfismo	Sì, tramite classi e oggetti	No, non c'è un concetto di ereditarietà	No, funzioni pure indipendenti
Complessità del programma	Alta, ma gestibile con buone pratiche di design	Potrebbe diventare complesso in progetti grandi	Bassa, grazie alla chiarezza e purezza delle funzioni
Adatto a	Sistemi complessi, applicazioni con interazioni tra entità	Applicazioni semplici e controllo dettagliato del flusso	Applicazioni matematiche, data transformation, calcoli paralleli

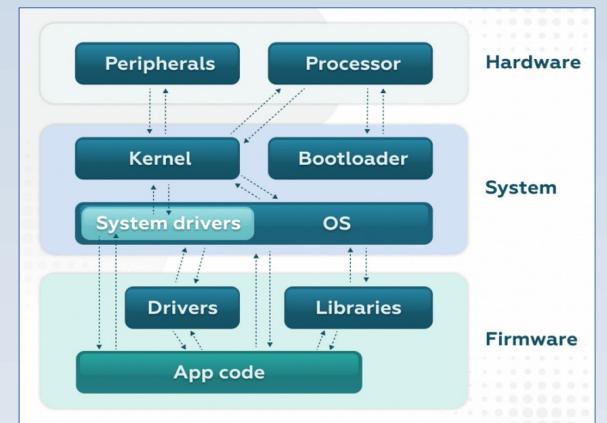
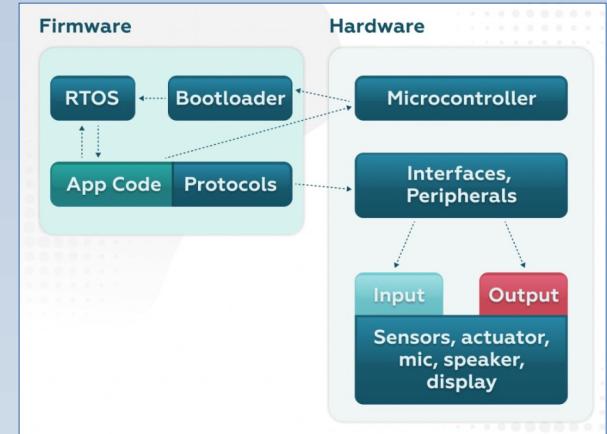
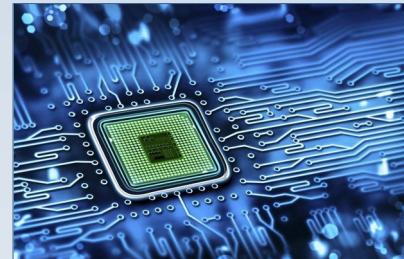
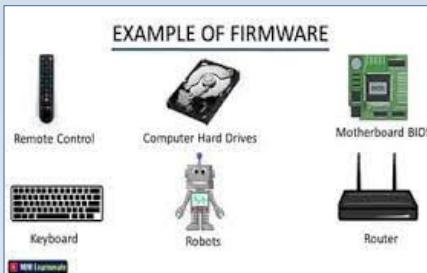
Ogni paradigma ha i suoi vantaggi, e la scelta dipende dal tipo di progetto e dai requisiti specifici. Nella pratica, molti linguaggi moderni, come C++ (usato anche per **Arduino**), supportano tutti e tre i paradigmi, consentendo di scegliere quello più adatto al contesto.

Attività pratica !

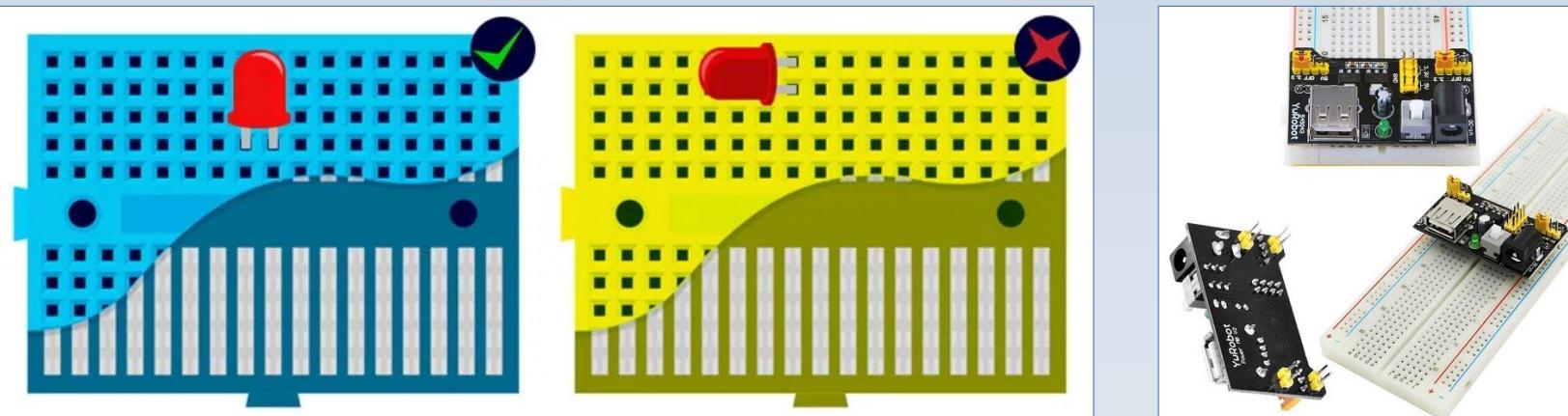
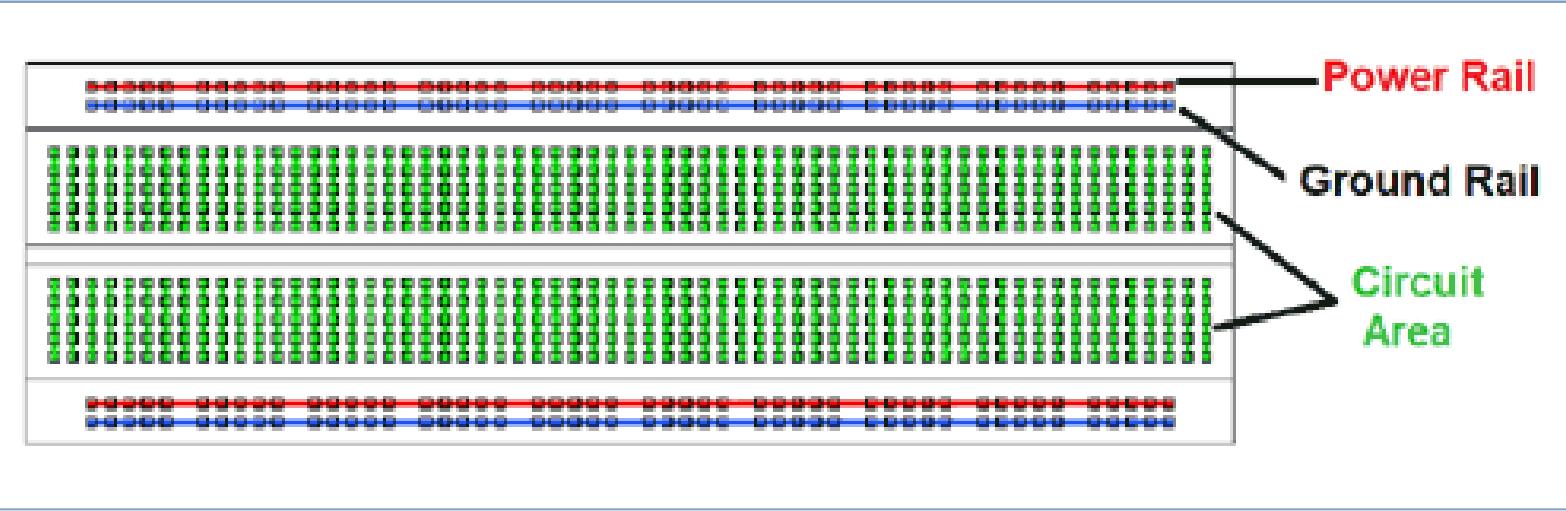
Breve ricerca o approfondimenti sui concetti di software, firmware, driver, hardware.

Provare a scrivere un breve codice per una calcolatrice nei tre paradigmi diversi di programmazione.

<https://vakoms.com/blog/what-is-firmware-definition-types-and-uses/>



La breadboard



Elettronica di Base per Arduino

Componenti Elettronici Fondamentali

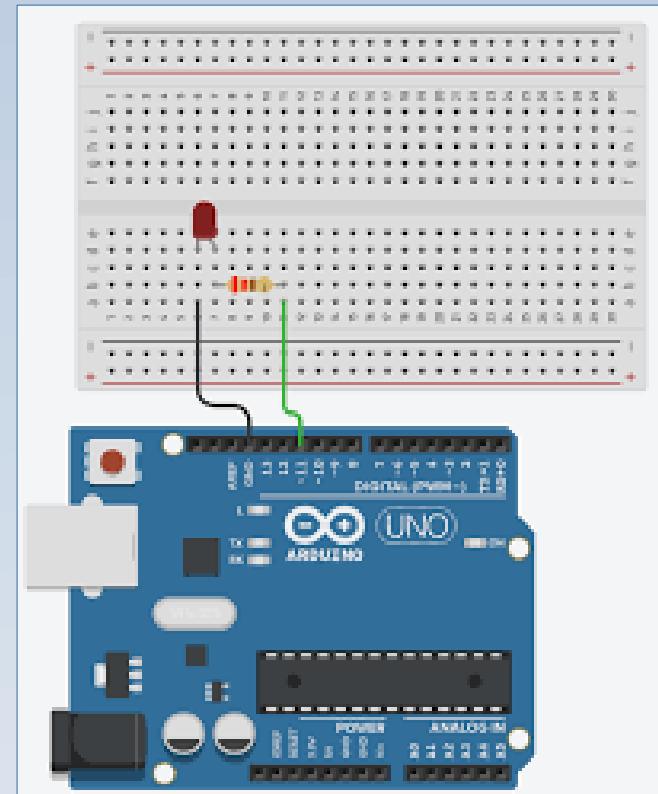
LED: Un LED è un dispositivo che emette luce quando attraversato da corrente elettrica. Viene utilizzato per visibilizzare lo stato del circuito.

Resistenze: Le resistenze limitano il flusso di corrente, proteggendo i componenti elettronici.

Collegamento di un LED

Collega il pin lungo del LED (anodo) al pin digitale 13 di Arduino.

Collega il pin corto del LED (catodo) alla resistenza da 220 ohm, quindi collega l'altro lato della resistenza al GND di Arduino.



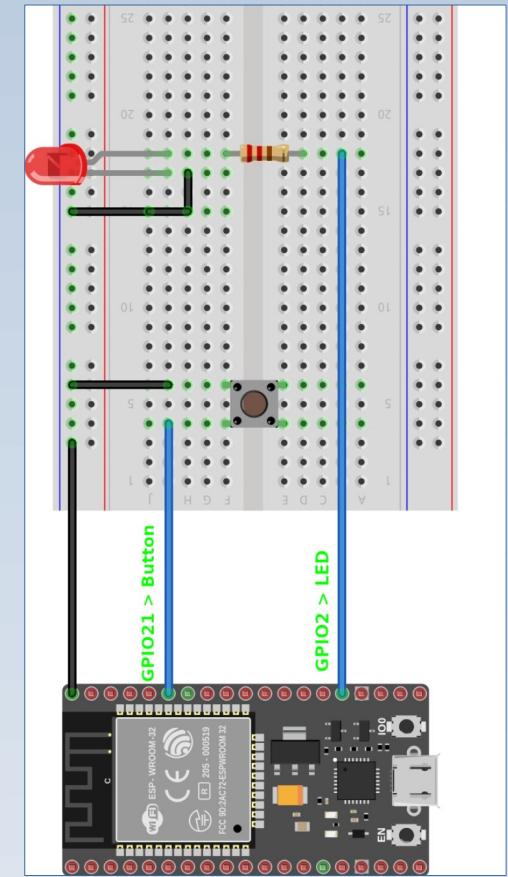
OUTPUT E INPUT DIGITALE

```
// Dichiarazione dei pin
const int pinPulsante = 2;      // Pulsante collegato al pin digitale 2
const int pinLED = 13;          // LED collegato al pin digitale 13 (LED integrato)

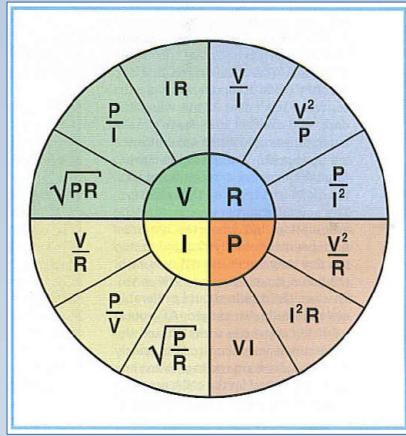
void setup() {
    pinMode(pinPulsante, INPUT); // Imposta il pin del pulsante come ingresso
    pinMode(pinLED, OUTPUT);    // Imposta il pin del LED come uscita
}

void loop() {
    int statoPulsante = digitalRead(pinPulsante); // Legge lo stato del pulsante

    if (statoPulsante == HIGH) {
        digitalWrite(pinLED, HIGH); // Accende il LED
    } else {
        digitalWrite(pinLED, LOW); // Spegne il LED
    }
}
```

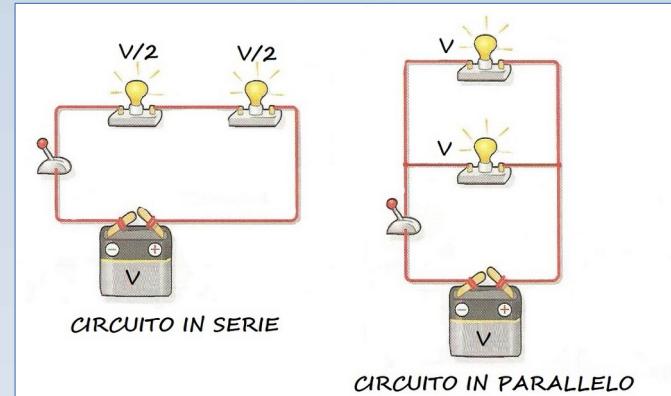


Informazioni utili su led e resistenze



$$R = \frac{(V_i - V_f)}{I}$$

Caduta di tensione	
Infrarosso	1,3
Rosso	1,8
Arancio	2,0
Giallo	1,9
Verde	2,0
Blu	3,0
Ultravioletto	3,0
Bianco	3,0



Attività pratica !

Il codice morse

L'insegnante può predisporre un messaggio che gli alunni dovranno decodificare.

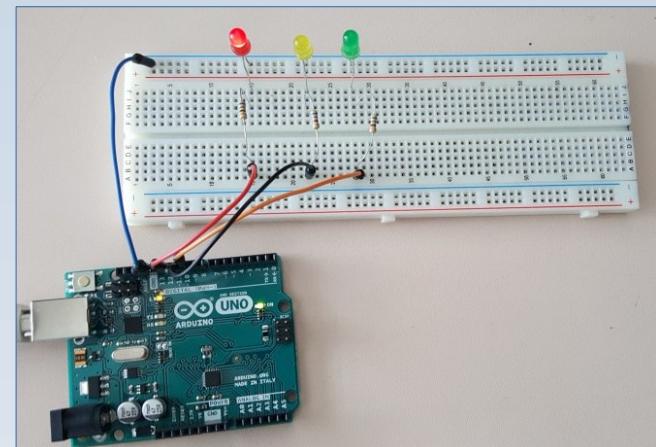
Il semaforo

Due o più gruppi di alunni possono predisporre uno scenario in cui più semafori devono essere sincronizzati.

La frequenza della vista umana

È possibile utilizzare il lampeggio per scoprire a quale frequenza l'occhio umano può distinguere l'accensione e lo spegnimento di un led ad un determinato intervallo di tempo.

A	--	J	----	S	...	2	-----
B	-....	K	---	T	-	3
C	---	L	U	...	4
D	...	M	--	V	----	5
E	.	N	--	W	---	6
F	---	O	---	X	---	7	-----
G	---	P	---	Y	---	8	-----
H	Q	----	Z	---	9	-----
I	..	R	---	1	----	0	-----



Sensori di temperatura

Sensori di Temperatura e Umidità

Un esempio comune è il sensore **LM35** (per la temperatura) o **DHT11/DHT22** (per temperatura e umidità) o i più nuovi **AHT10/AHT20**.

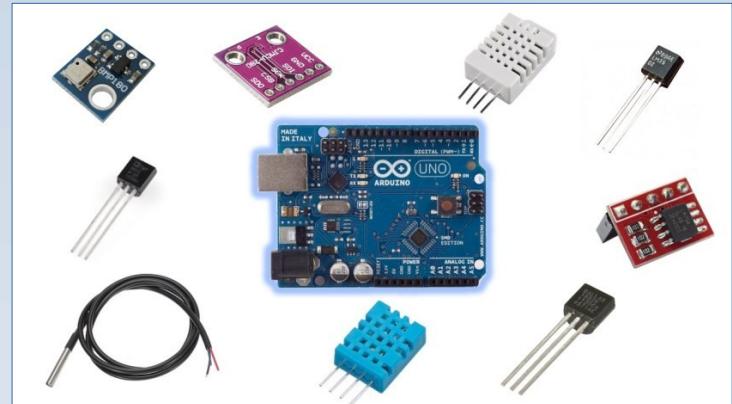
Il sensore rileva l'**umidità relativa** e la **temperatura** misurando la resistenza elettrica tra i due elettrodi. La conduttività tra gli elettrodi aumenta all'aumentare dell'umidità relativa.

Il minuscolo sensore ha un intervallo di alimentazione di **1,8–3,6 V**, ma 3,3 V è la tensione operativa consigliata.

```
int sensorPin = A0;
float temperature;

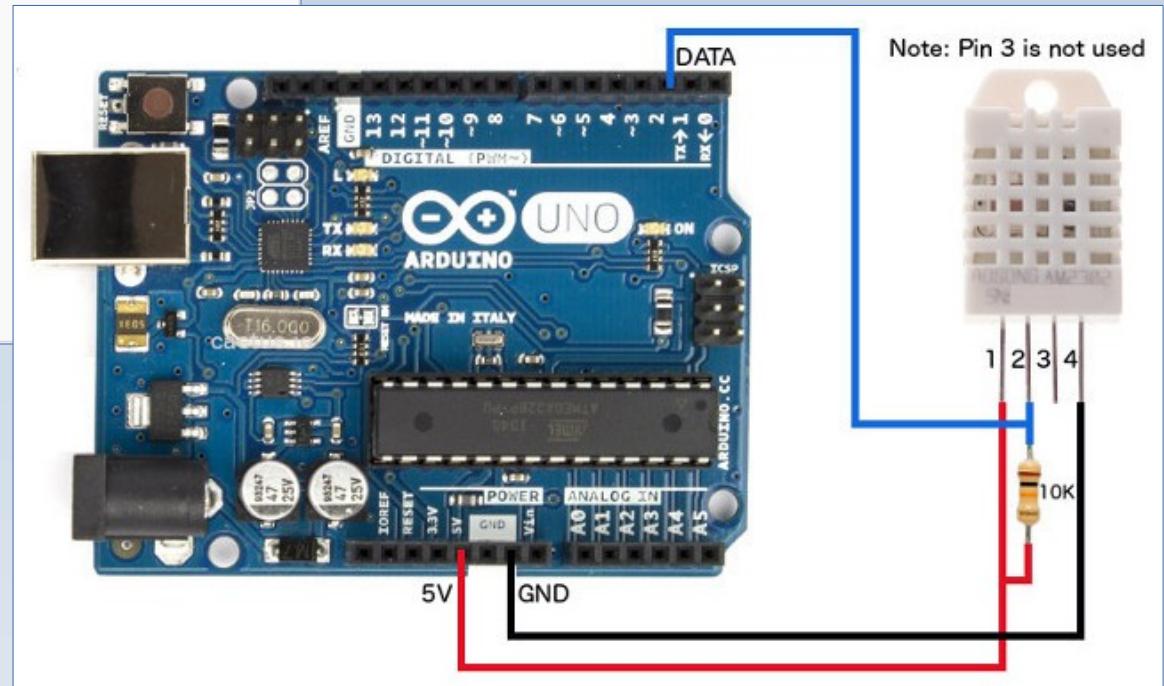
void setup() {
  Serial.begin(9600);
}

void loop() {
  temperature = analogRead(sensorPin) * (5.0 / 1023.0) * 100; // Conversione in °C
  Serial.println(temperature); // Visualizza la temperatura nel monitor seriale
  delay(1000); // Attendere 1 secondo
}
```



DHT11 e DHT22

```
#include <DHT.h>  
  
#define DHTPIN 2          // Pin connesso al sensore  
#define DHTTYPE DHT11    // Tipo di sensore  
  
DHT dht(DHTPIN, DHTTYPE);  
  
void setup() {  
  Serial.begin(9600);  
  dht.begin();  
}  
  
void loop() {  
  float temp = dht.readTemperature(); // Leggi la temperatura  
  Serial.print("Temperatura: ");  
  Serial.println(temp);  
  delay(2000);  
}
```



SERIE BME180/280/680

```
#include <Wire.h>
#include <Adafruit_BMP280.h>

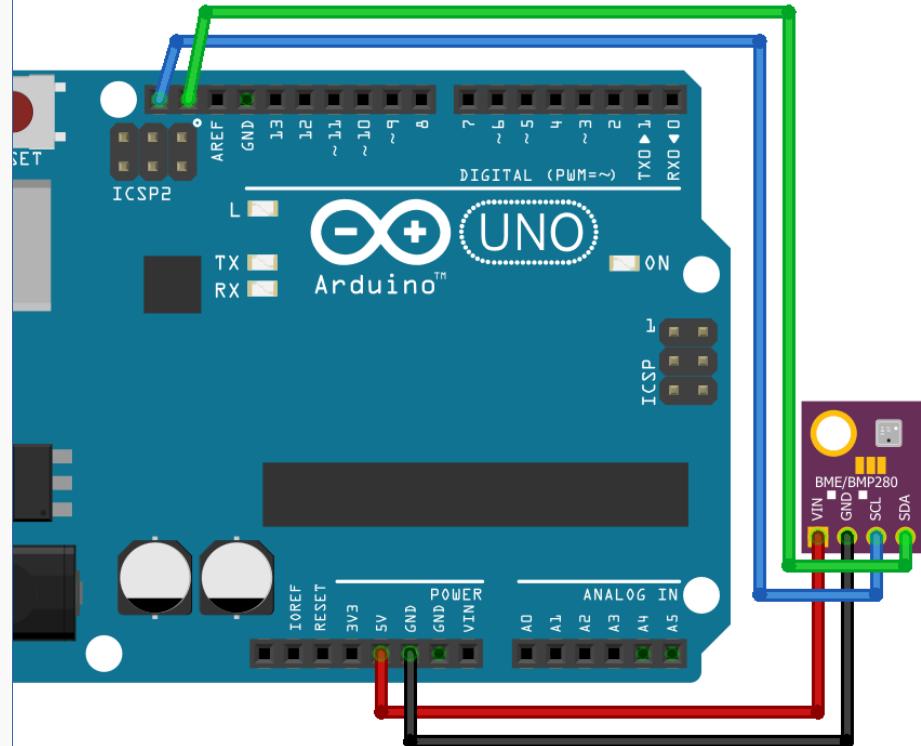
Adafruit_BMP280 bmp; // I2C

void setup() {
  Serial.begin(9600);
  if (!bmp.begin(0x76)) { // alcuni modelli usano 0x77
    Serial.println("Errore: BMP280 non trovato. Controlla indirizzo e cablaggio.");
    while (1);
  }
}

void loop() {
  Serial.print("Temperatura = ");
  Serial.print(bmp.readTemperature());
  Serial.println(" °C");

  Serial.print("Pressione = ");
  Serial.print(bmp.readPressure() / 100.0F); // da Pa a hPa
  Serial.println(" hPa");

  delay(2000);
}
```



fritzing

Obiettivi didattici

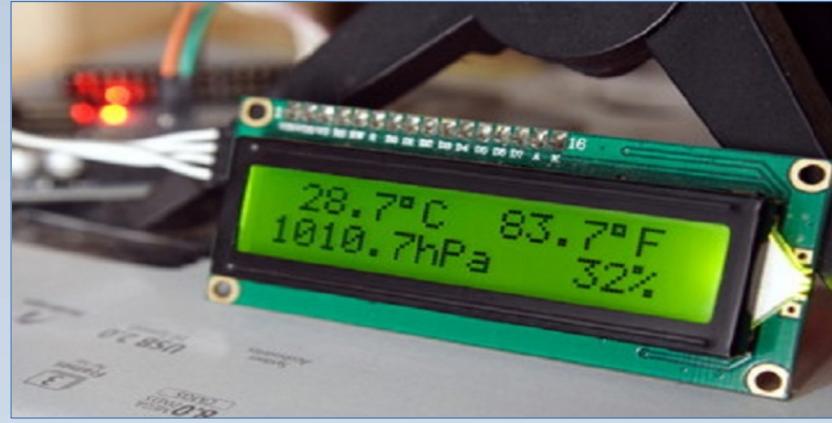
Fisica: Leggere pressione atmosferica e discutere il principio di Pascal.

Scienze: Misurare umidità relativa e temperatura ambiente.

Tecnologia: Utilizzare un sensore digitale via I2C.

Competenze trasversali: Acquisire dati, elaborarli e interpretarli.

Attività pratica !



Idee per attività didattiche

Esperimento sull'altitudine: sposta il sensore su più piani di un edificio per vedere il cambiamento di pressione e calcolare l'altezza teorica.

Clima della classe: monitoraggio continuo per verificare se umidità e temperatura sono adeguate.

Progetto meteo: confrontare i dati con quelli di una stazione meteo online.

Laboratorio di fisica: studiare la relazione tra temperatura e pressione in un ambiente controllato.

Sensore ad ultrasuoni

Sensori ad ultrasuoni

Il HC-SR04 è un sensore a ultrasuoni che misura la distanza tra sé e un oggetto riflettente usando onde sonore ad alta frequenza (~ 40 kHz).

Come funziona?

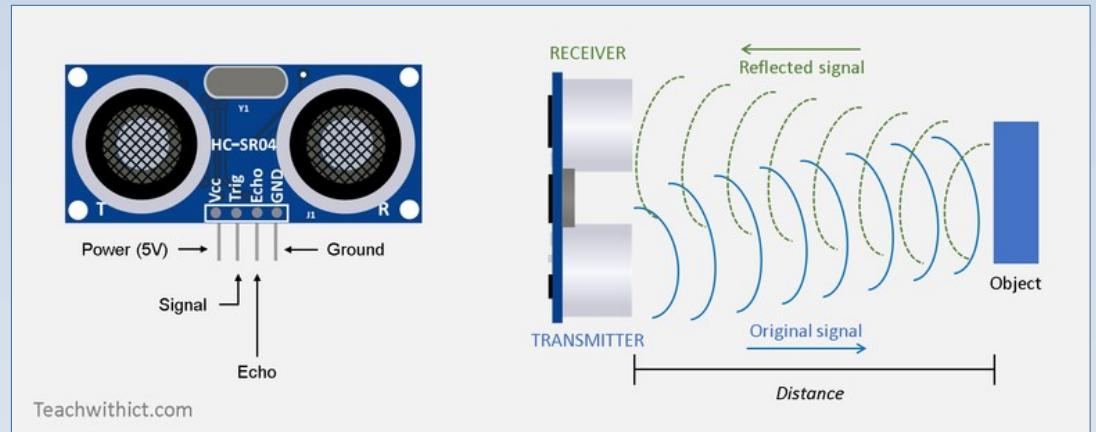
Arduino invia un impulso Trigger.

Il sensore emette un'onda ultrasonica.

L'onda rimbalza su un ostacolo e torna indietro.

Il sensore restituisce un impulso Echo: la durata indica il tempo di andata e ritorno del suono.

Con la velocità del suono (~343 m/s), si calcola la distanza



Sensore ad ultrasuoni HC SR04

```
#define TRIG_PIN 9
#define ECHO_PIN 10

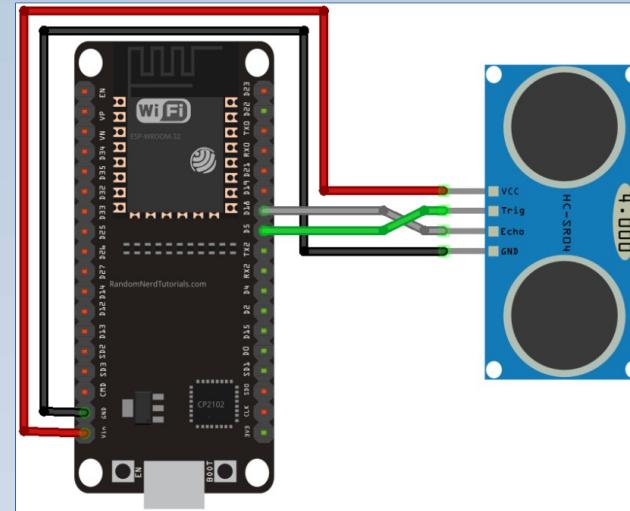
void setup() {
    Serial.begin(9600);
    pinMode(TRIG_PIN, OUTPUT);
    pinMode(ECHO_PIN, INPUT);
    Serial.println("Misuratore di distanza attivo");
}

void loop() {
    long duration;
    float distance;

    // Pulisce il pin trigger
    digitalWrite(TRIG_PIN, LOW);
    delayMicroseconds(2);

    // Invia un impulso di 10 microsecondi
    digitalWrite(TRIG_PIN, HIGH);
    delayMicroseconds(10);
    digitalWrite(TRIG_PIN, LOW);

    // Legge il tempo di ritorno dell'eco
    duration = pulseIn(ECHO_PIN, HIGH);
```



```
// Calcola la distanza in centimetri
distance = (duration * 0.0343) / 2;

Serial.print("Distanza: ");
Serial.print(distance);
Serial.println(" cm");

delay(500);
}
```

Sensore ad ultrasuoni

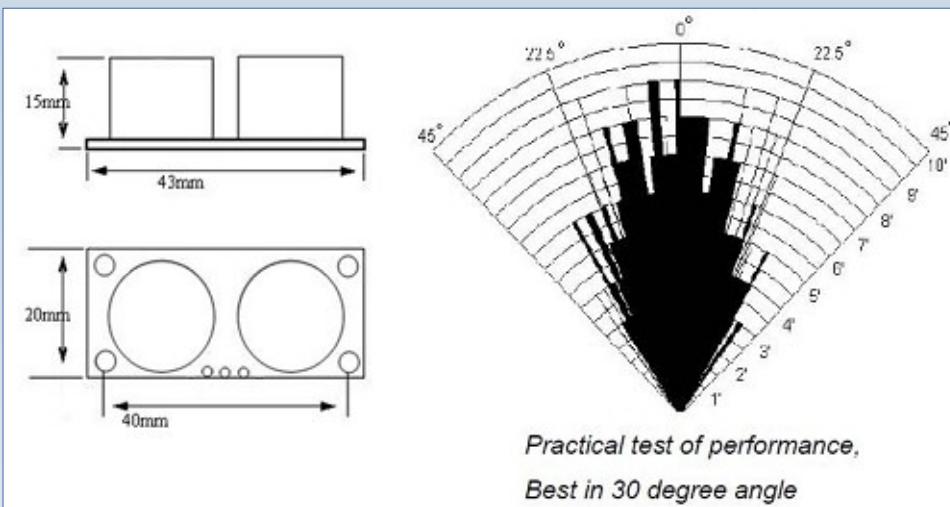
Finalità didattiche

Fisica: Studio della velocità del suono, tempo/distanza, onde meccaniche

Tecnologia: Sensori, sistemi di misura, automazione

Informatica: Input/output digitale, temporizzazione, calcoli con microcontrollore

Matematica: Proporzioni, conversioni, formula della velocità



Power Supply	5V DC
Working Current	15 mA
Working Frequency	40 kHz
Maximum Range	4 meters
Minimum Range	2 cm
Measuring Angle	15°
Resolution	0.3 cm
Trigger Input Signal	10uS TTL pulse
Echo Output Signal	TTL pulse proportional to the distance range
Dimensions	45mm x 20mm x 15mm

Attività pratica !

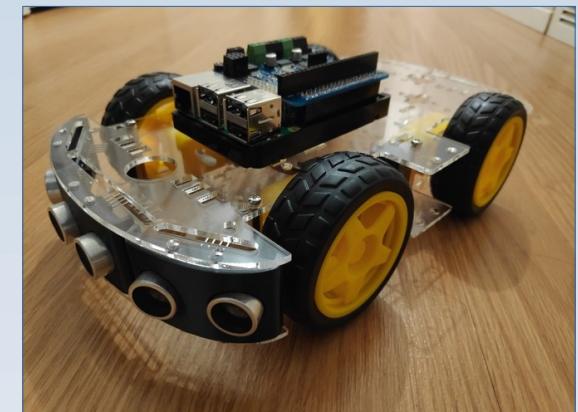
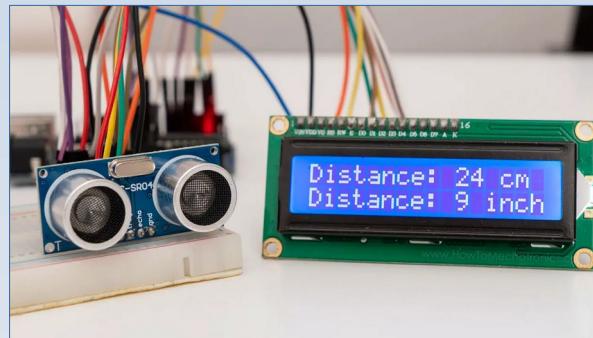
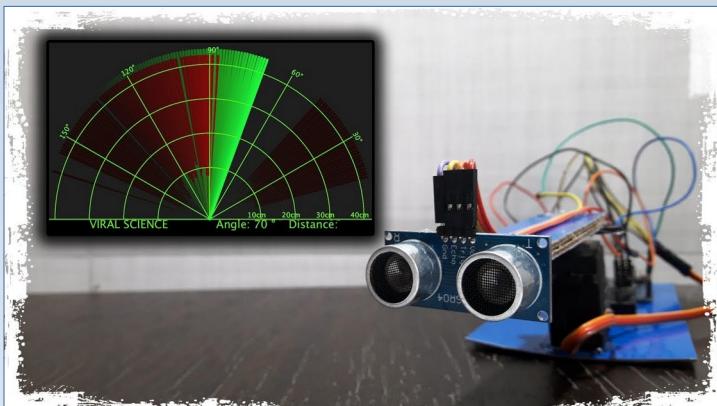
Idee per progetti

Contapassi automatico: posizionato vicino a una porta, conta i passaggi.

Misura della velocità: posizionando due sensori in sequenza.

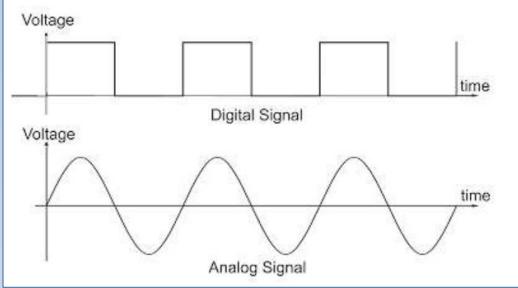
Radar semplificato: conservo i dati su PC e creo grafico della distanza.

Parcheggio assistito: LED o buzzer in base alla distanza misurata.



Ingressi analogici e PWM

Obiettivi didattici



Comprendere la differenza tra segnali analogici e digitali

Imparare a usare le porte analogiche (input) e PWM (output)

Applicare concetti di elettronica e informatica (duty cycle, modulazione)

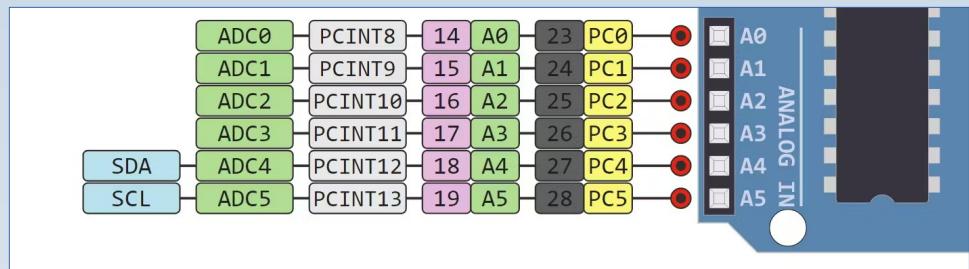
Realizzare esperimenti interattivi (LED, motori, sensori)

Porte analogiche (input)

Su Arduino Uno: da **A0** ad **A5**

Usate per leggere segnali di **input analogici** da sensori di temperatura, luce, potenziometri, ecc.

Usano un convertitore ADC a 10 bit → valori da 0 a 1023

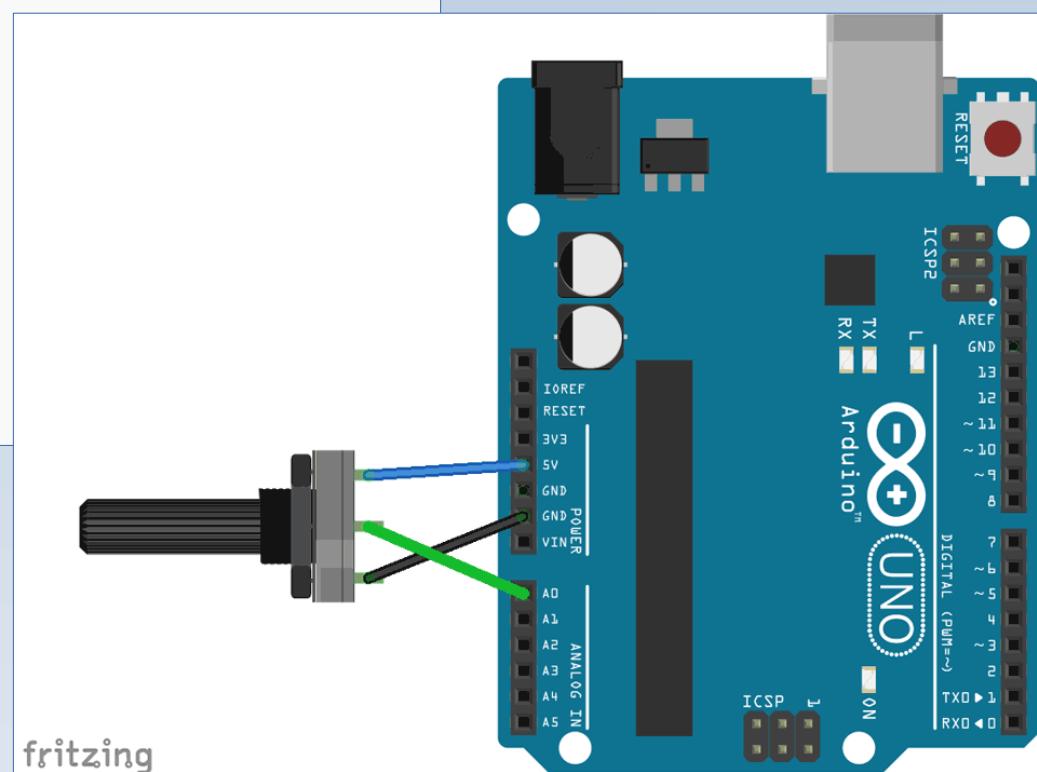


Lettura di un potenziometro

```
int potPin = A0; // collegato al cursore del potenziometro
int valore;

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

void loop() {
  valore = analogRead(potPin); // legge valore tra 0 e 1023
  Serial.println(valore);
  delay(100);
}
```



PWM - Pulse Width Modulation

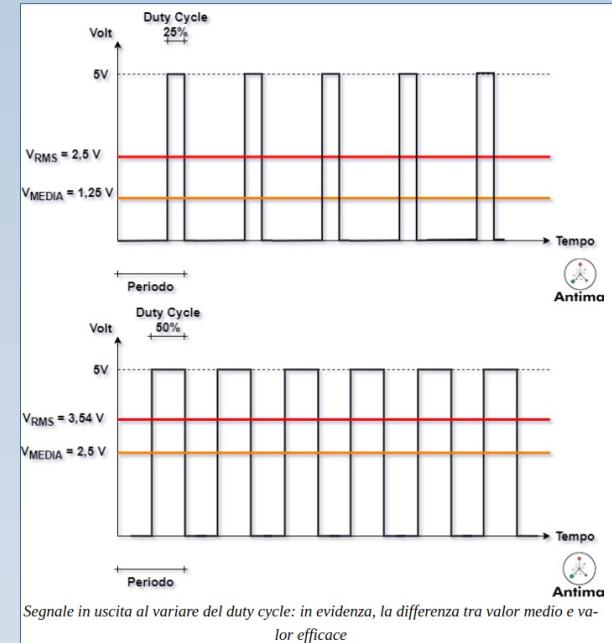
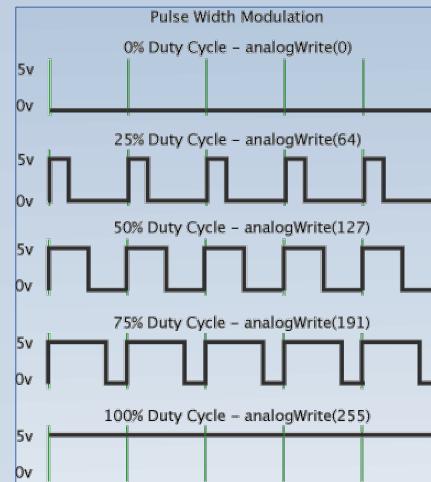
Che cosa significa Pulse Width Modulation?

Significa simulare un'**uscita analogica** con un segnale digitale.

Si basa su **impulsi digitali variabili** nel duty cycle.

Ciò che caratterizza e rende interessante la tecnica PWM è la possibilità di **variare il duty cycle**, ovvero l'intervallo di tempo in cui si ha lo stato alto rispetto al periodo totale.

In genere, questi valori (alto e basso) sono fissati a priori in funzione del tipo di dispositivo e di architettura che si usa e che li genera: un Arduino Uno lavora con livelli logici a 5V, dunque i valori saranno 0V e 5V, invece un ESP8266 lavora a 3,3V.



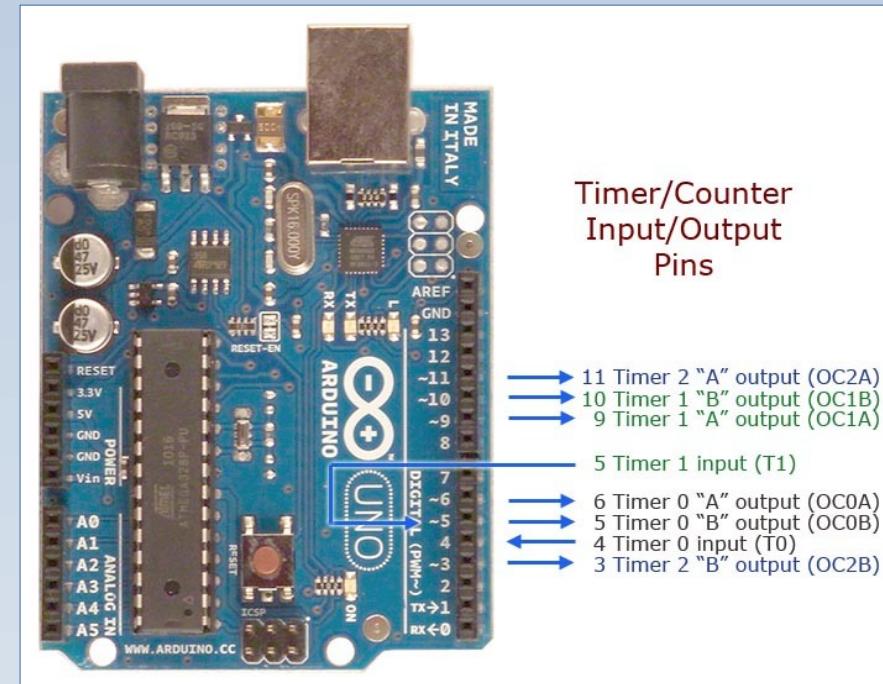
$$D = \frac{t_{on}}{T}$$

$$V_{RMS} = \sqrt{D(V_{max})^2}$$

PWM - Pulse Width Modulation

Il periodo del segnale è legato alla frequenza del dispositivo (microcontrollore) che si sta utilizzando e dunque varierà tra dispositivi differenti. Inoltre, non è detto che lo stesso microcontrollore utilizzi la stessa frequenza per ogni pin. Ad esempio, per un Arduino Uno, la frequenza del segnale PWM sui pin 5 e 6 sarà di circa 980Hz, mentre sugli altri pin sarà 490Hz. Questo è un fattore da tener conto in alcuni casi.

Il numero e la disposizione di questi pin, riconoscibili dal simbolo tilde ~ stampato su pcb, varia da scheda a scheda. Ad esempio, l'Arduino Uno e Nano presentano PWM ai pin 3, 5, 6, 9, 10 e 11. Arduino Mega invece dispone di pin PWM dal 2 al 13!

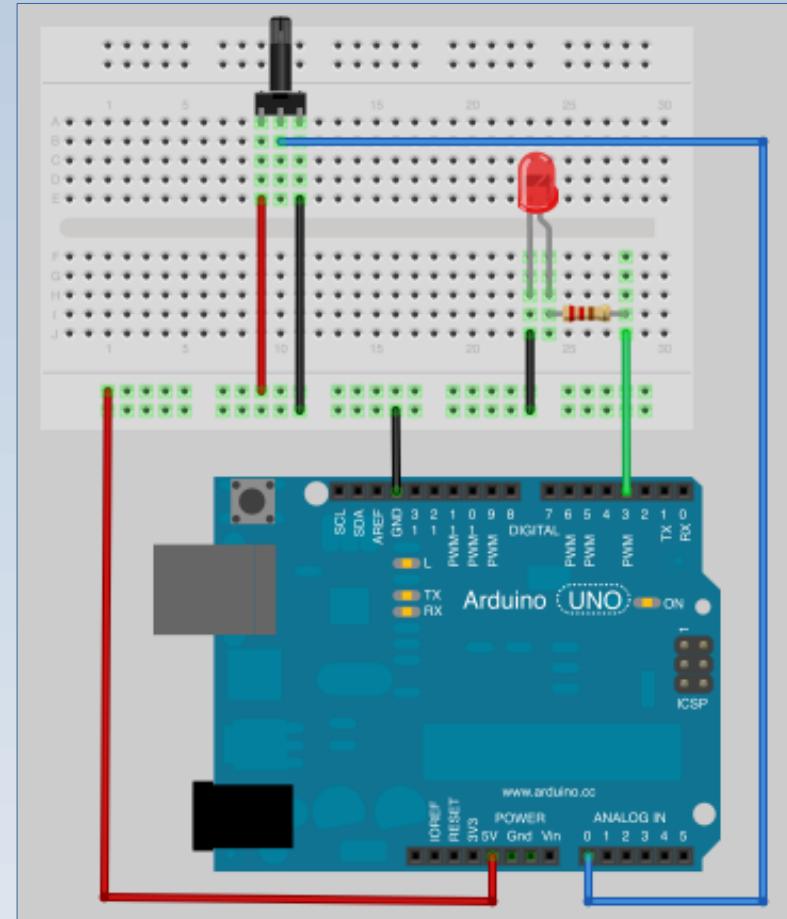


PWM: Dimmer per LED

```
int led = 9;          // pin PWM
int potPin = A0;      // potenziometro

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

void loop() {
    int val = analogRead(potPin);          // 0 - 1023
    int pwm = map(val, 0, 1023, 0, 255);   // converte per PWM
    analogWrite(led, pwm);                 // imposta luminosità
    delay(10);
}
```



INPUT E OUTPUT DIGITALI E ANALOGICI

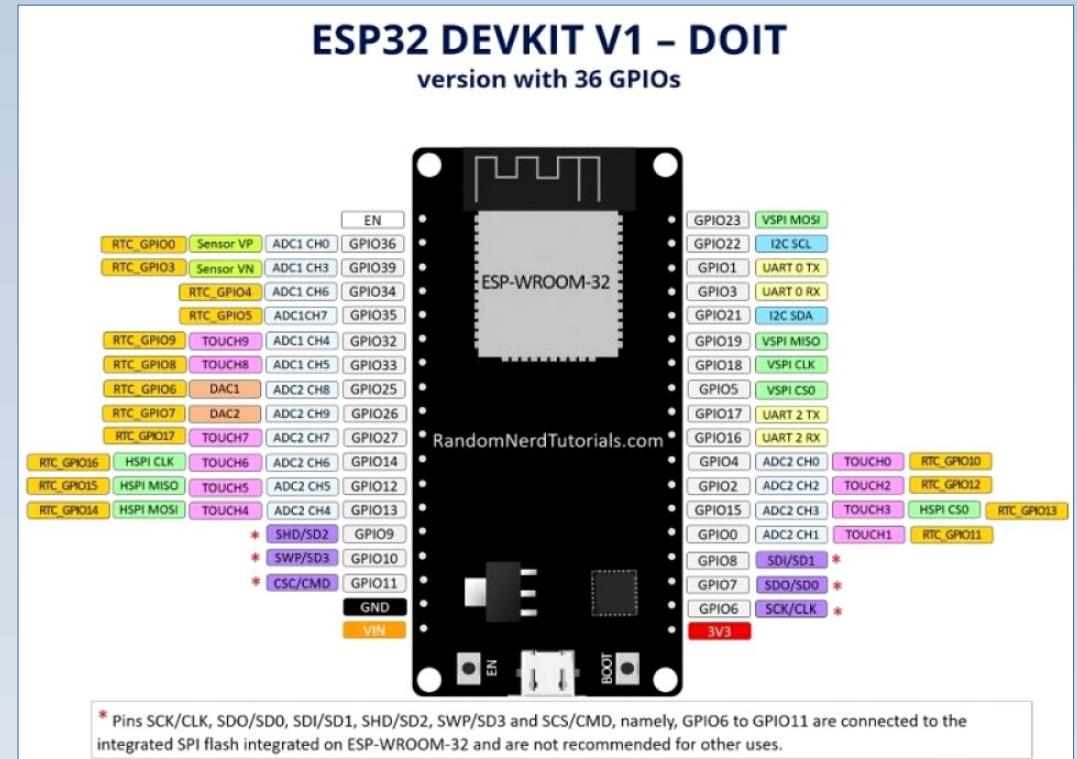
Finalità didattiche interdisciplinari

Fisica: Tensione, corrente, resistenza, potenza

Tecnologia: Sensori, attuatori, segnali

Informatica: Programmazione con variabili, cicli, funzioni

Matematica: Map, proporzioni, conversioni tra intervalli



RIEPILOGO COMANDI INPUT E OUTPUT

```
// Dichiarazioni dei pin
const int ledDigital = 2;      // LED acceso/spento (uscita digitale)
const int pulsante = 3;        // Pulsante (ingresso digitale)

const int ledPWM = 9;          // LED con luminosità variabile (uscita PWM)
const int potPin = A0;         // Potenziometro (ingresso analogico)

void setup() {
    // Configurazione dei pin
    pinMode(ledDigital, OUTPUT); // Pin digitale in uscita
    pinMode(pulsante, INPUT);   // Pin digitale in ingresso
    pinMode(ledPWM, OUTPUT);   // Pin PWM in uscita

    Serial.begin(9600);        // Per monitorare valori sul seriale
}
```

RIEPILOGO COMANDI INPUT E OUTPUT

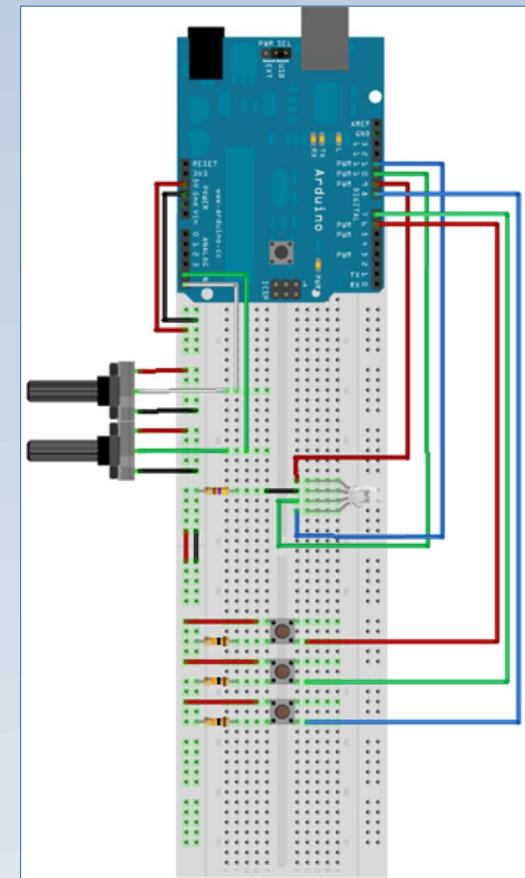
```
void loop() {
    // --- 1. Lettura digitale: pulsante ---
    int statoPulsante = digitalRead(pulsante); // HIGH o LOW
    digitalWrite(ledDigital, statoPulsante); // Accende/spegne LED

    // --- 2. Lettura analogica: potenziometro ---
    int valoreAnalogico = analogRead(potPin); // 0-1023

    // --- 3. Scrittura analogica (PWM): controlla luminosità LED ---
    int valorePWM = map(valoreAnalogico, 0, 1023, 0, 255); // conversione per PWM
    analogWrite(ledPWM, valorePWM); // Modula luminosità

    // Monitor seriale per vedere i valori
    Serial.print("Pulsante: ");
    Serial.print(statoPulsante);
    Serial.print(" | Potenziometro: ");
    Serial.print(valoreAnalogico);
    Serial.print(" | PWM: ");
    Serial.println(valorePWM);

    delay(100); // piccola pausa
}
```



Attività pratica !

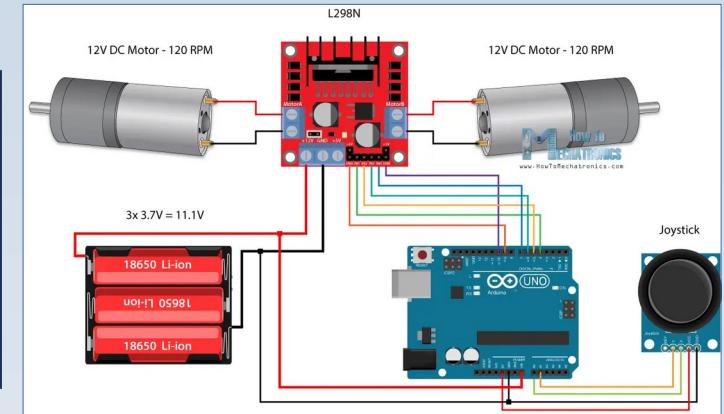
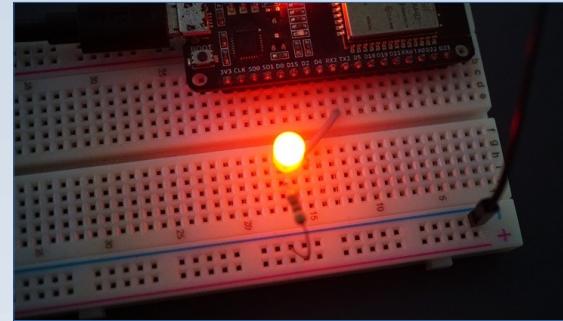
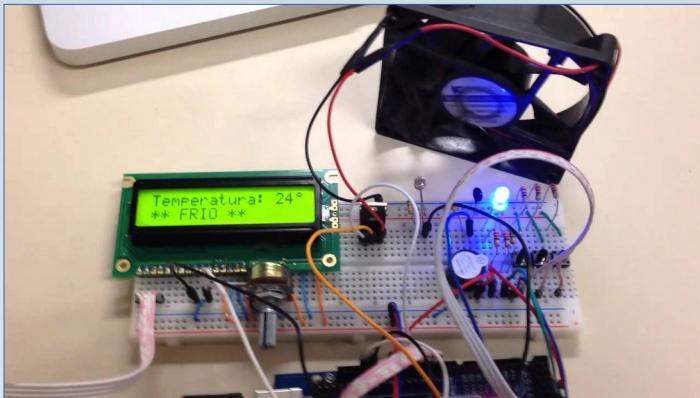
Idee per progetti

Dimmer LED con potenziometro: PWM, analogRead, analogWrite

Ventilatore con motore DC: PWM su motore, controllo velocità

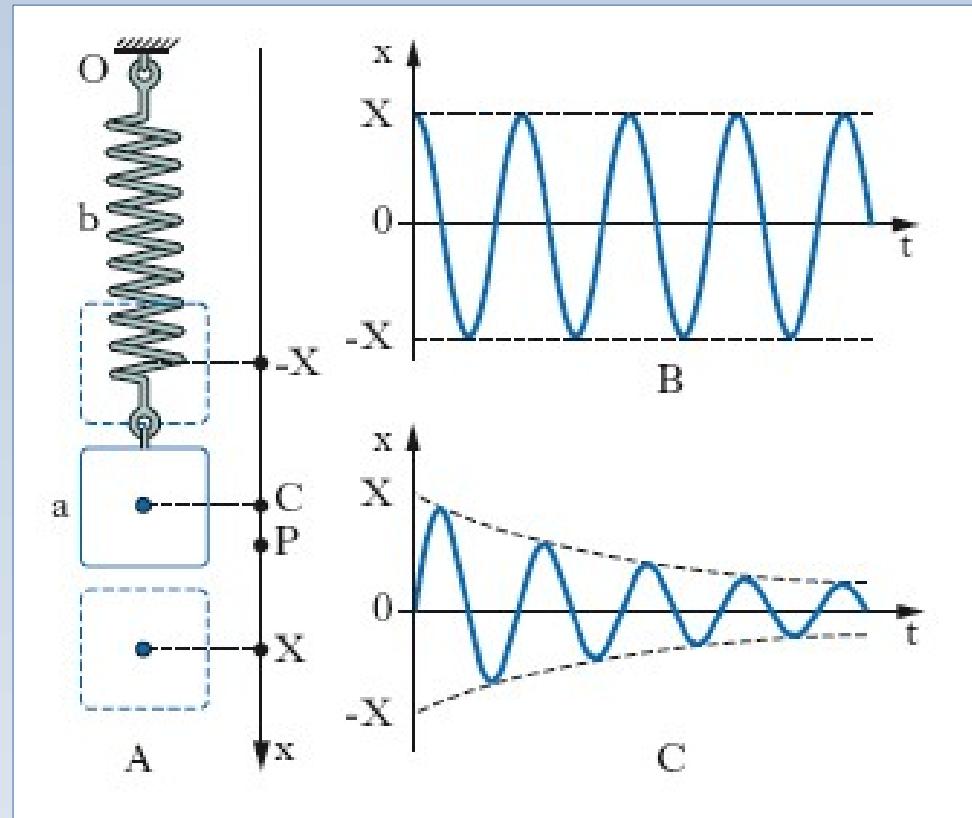
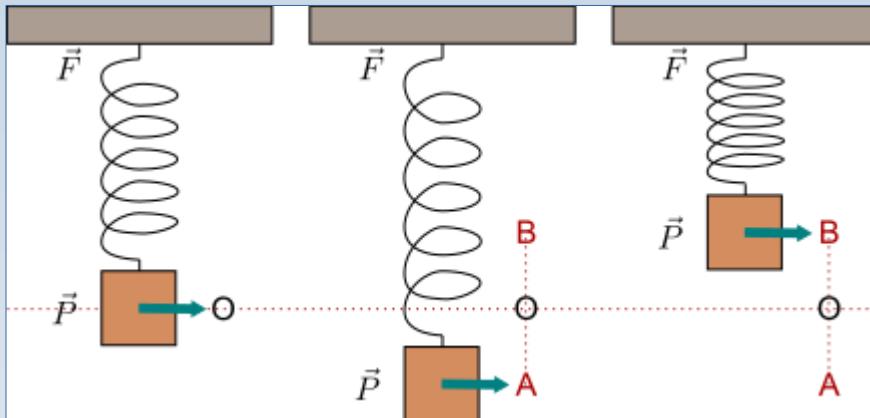
Sensore di luce con LED: LDR su A0, output su LED

Semaforo con LED: Digital + PWM per LED di potenza



Attività pratica !

Realizzazione di un misuratore di **Periodo e Frequenza** di un oggetto in oscillazione su modello e ispirazione del progetto del prof. Organtini Giovanni pubblicato sul libro “**Fisica con Arduino**”.



Attività pratica !

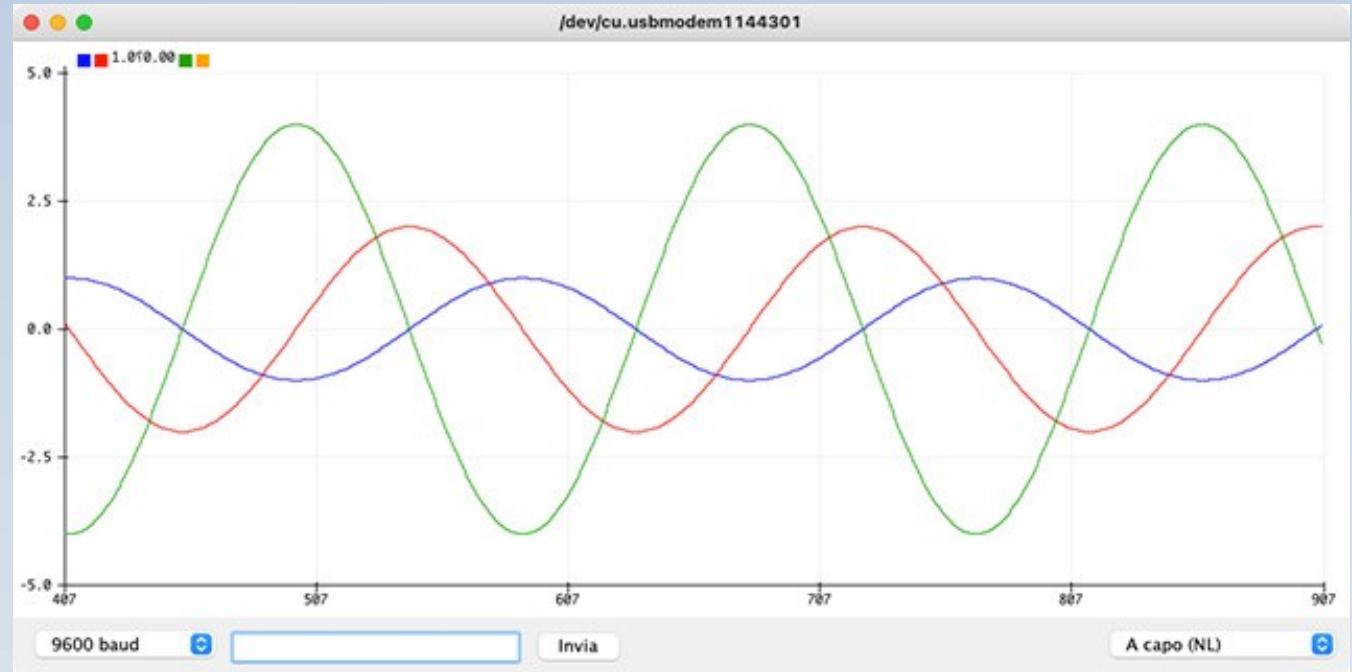
```
void loop() {  
    // Trigger dell'impulso  
    digitalWrite(trigPin, LOW);  
    delayMicroseconds(2);  
    digitalWrite(trigPin, HIGH);  
    delayMicroseconds(10);  
    digitalWrite(trigPin, LOW);  
  
    // Lettura eco  
    durata = pulseIn(echoPin, HIGH);  
    distanza = durata * 0.034 / 2;
```

```
    currentTime = millis();  
  
    // Rilevazione del passaggio (solo quando si entra nella soglia)  
    if (distanza < soglia_passaggio && !oggettoVicino) {  
        oggettoVicino = true;  
  
        // Calcola periodo se non è il primo passaggio  
        if (lastPassTime != 0) {  
            periodo = (currentTime - lastPassTime) / 1000.0; // in secondi  
            frequenza = 1.0 / periodo;  
        }  
        lastPassTime = currentTime;  
    }  
  
    // Rileva uscita dalla soglia  
    if (distanza >= soglia_passaggio) {  
        oggettoVicino = false;  
    }
```

Attività pratica !

```
// Output su Serial Plotter
Serial.print("Distanza:");
Serial.print(distanza);
Serial.print(" Periodo:");
Serial.print(periodo);
Serial.print(" Frequenza:");
Serial.println(frequenza);

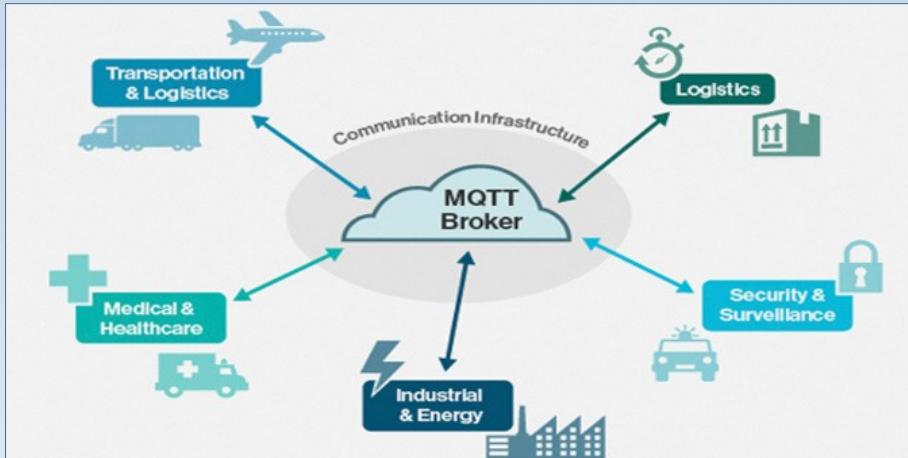
delay(50);
}
```



Introduzione a MQTT

MQTT e Comunicazione Wireless con ESP32

In questo modulo esploreremo il protocollo **MQTT**, molto utilizzato nell'ambito dell'Internet of Things (IoT), e impareremo a usarlo con la scheda **ESP32**, un microcontrollore potente con connettività wireless integrata. Impareremo a inviare e ricevere dati tra dispositivi tramite una rete, sviluppando sistemi intelligenti e connessi.



Introduzione a MQTT

MQTT: Un protocollo pensato per l'IoT

MQTT (Message Queuing Telemetry Transport) è un protocollo leggero per la trasmissione di dati su reti TCP/IP. È basato su un meccanismo di **Pub/Sub** (publish-subscribe), dove i dispositivi possono:

- Pubblicare messaggi su un argomento (**topic**)
- Iscriversi a un **topic** per ricevere aggiornamenti

È perfetto per dispositivi con risorse limitate e connessioni non sempre affidabili.

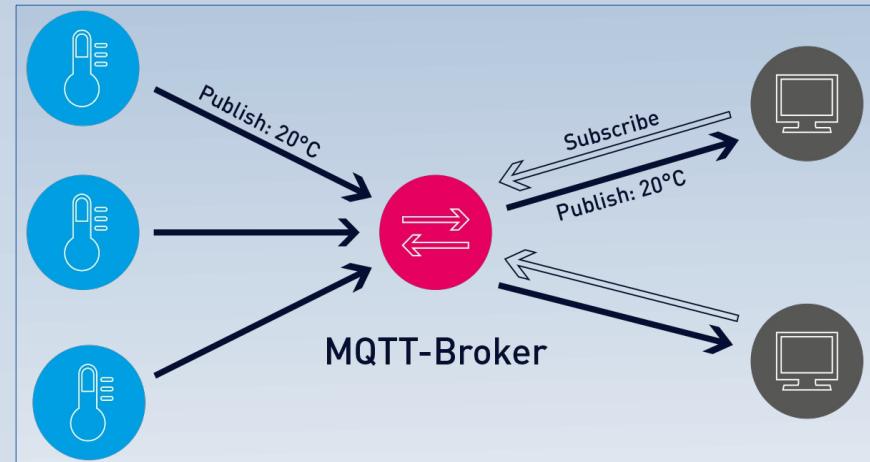
TCP/IP Layers	IoT Protocols		
Application	CoAP HTTP	MQTT AMQP	Others
Transport	TCP	UDP	DTLS
Internet	IP	6LoWPAN	RPL
Network	IEEE 802.15.4 IEEE802.11/b/g/n/ac/ad/ah/ax IEEE 802.3 Ethernet	GSM	LTE LPWAN

Architettura MQTT

Come funziona MQTT?

L'architettura si basa su tre elementi:

- **Broker:** Server centrale che gestisce e smista i messaggi.
- **Client Publisher:** Dispositivo che invia (pubblica) messaggi.
- **Client Subscriber:** Dispositivo che riceve i messaggi pubblicati su uno o più topic.



Esempio:

ESP32 legge temperatura → la invia al broker → un'app o un altro dispositivo la riceve.

Applicazioni reali

MQTT nel mondo reale

Esempi reali d'uso:

- **Domotica**: Controllo luci, termostati, prese intelligenti.
- **Stazioni meteo**: Sensori che inviano dati a un server centrale.
- **Tracciamento veicoli**: GPS che trasmette la posizione.
- **Smart farming**: Sensori di umidità del terreno, temperatura, ecc.

MQTT vs altri protocolli

- HTTP: ogni richiesta comporta l'invio completo dei dati (più lento e pesante).
- MQTT: mantiene la connessione e invia solo messaggi rilevanti.

Metafora:

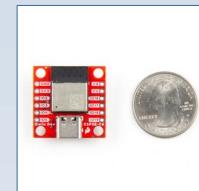
- *HTTP è come una telefonata dove ogni volta devi ridire chi sei.*
- *MQTT è una chat continua tra amici: si parla solo quando serve.*

Introduzione ad ESP32

Cos'è l'ESP32?

L'ESP32 è un microcontrollore sviluppato da Espressif. È la “versione avanzata” di Arduino:

- Include **Wi-Fi e Bluetooth**
- **Dual-core, a 240 MHz**
- Molto più potente, con più GPIO e memoria
- Programmazione compatibile con Arduino IDE
- Memoria flash fino a 4 MB
- >30 GPIO disponibili



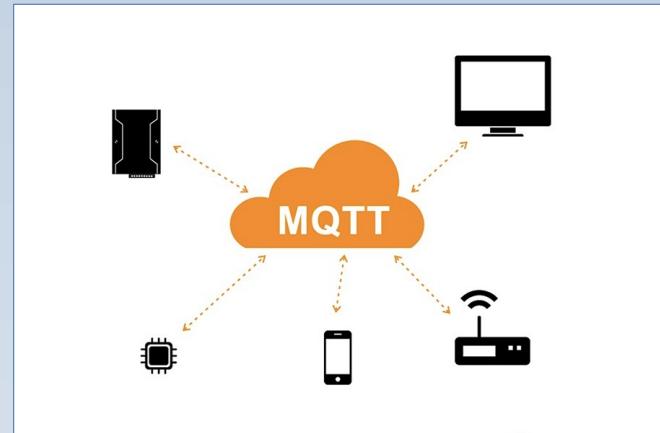
Costruzione rete MQTT

Cos'è un broker MQTT?

Il broker è il cuore del sistema, un **server centrale** che riceve tutti i messaggi dai client e li ridistribuisce a chi è iscritto.

Esempi di broker:

- Mosquitto (open source)
- HiveMQ
- Cloud MQTT (online)
- EMQX, VerneMQ, RabbitMQ (con plugin MQTT)
- Soluzioni cloud: Adafruit IO, AWS IoT, Thingsboard



Il broker è il "postino centrale" del sistema MQTT. Non fa altro che ricevere tutti i messaggi pubblicati dai dispositivi (client) e inoltrarli solo a chi è iscritto (sottoscritto) al topic corrispondente.

Costruzione rete MQTT

Caratteristiche di configurazione e funzionamento

- Porta predefinita: **1883** (senza cifratura), 8883 (con TLS/SSL)
- **QoS** (Quality of Service): 0, 1, 2 – indica quanto deve essere affidabile la consegna
- **Topic**: Canali testuali (es. scuola/aula1/temp)
- Retained messages: Il broker conserva l'ultimo messaggio per ogni topic
- Last Will: Messaggio inviato se il client si disconnette in modo improvviso
- Client ID: Ogni client MQTT deve avere un identificativo univoco

Esempi:

scuola/lab/movimento → sensore PIR

classe3a/luci/comando → controllare accensione luci

scuola/+/temperatura → riceve tutti i messaggi di temperatura di ogni aula



Pubblicazione dati da ESP32

```
#include <WiFi.h>
#include <PubSubClient.h>
#include "DHT.h"

const char* ssid = "NOME_RETE";
const char* password = "PASSWORD";
const char* mqttServer = "192.168.1.100";

WiFiClient espClient;
PubSubClient client(espClient);
DHT dht(5, DHT11); // pin 5

void setup() {
    Serial.begin(115200);
    WiFi.begin(ssid, password);
    client.setServer(mqttServer, 1883);
    dht.begin();
}
```

```
void loop() {
    if (!client.connected()) {
        client.connect("ESP32Client");
    }
    float t = dht.readTemperature();
    char msg[10];
    dtostrf(t, 1, 2, msg);
    client.publish("scuola/temperatura", msg);
    delay(5000);
}
```



```
$ mosquitto_sub -h localhost -t scuola/temperatura -v
scuola/temperatura 23.56
scuola/temperatura 23.47
scuola/temperatura 23.58
scuola/temperatura 23.51
scuola/temperatura 23.49
scuola/temperatura 23.46
scuola/temperatura 23.50
```

Attività pratica !

Statistica e variazioni ambientali con sensore BME280 e Arduino

Primaria e Secondaria di primo grado

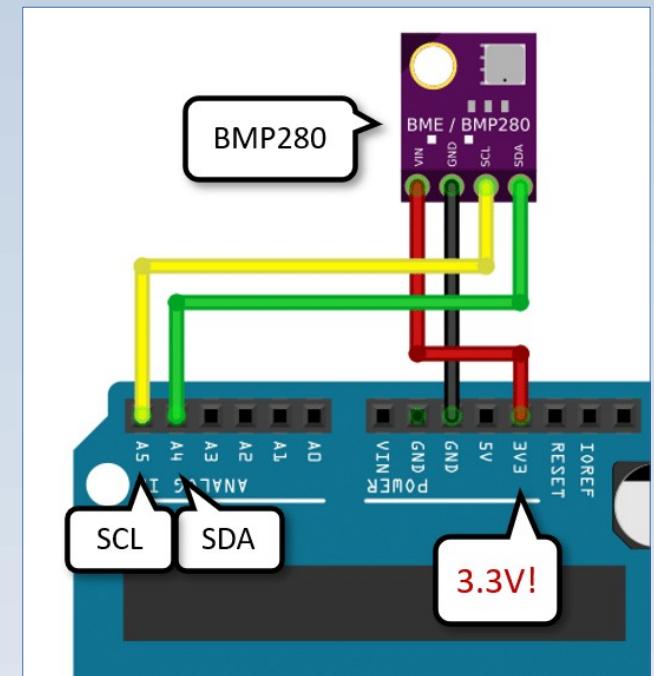
Obiettivi didattici:

- **Fisica:** comprendere grandezze fisiche ambientali (temperatura, pressione, umidità) e il concetto di variazione nel tempo e fenomeni correlati.
- **Matematica/Statistica:** raccogliere e analizzare dati reali, calcolare media, moda, deviazione standard, rappresentare graficamente l'andamento di una variabile. Concetto di campionamento.
- **Educazione scientifica:** sviluppare un atteggiamento sperimentale, confrontare misure, comprendere l'importanza della precisione e del contesto ambientale.

Attività pratica !

Materiale necessario e collegamenti

- Arduino UNO o Nano
- Sensore BME280 (I₂C)
- Cavi jumper
- Breadboard (opzionale)
- PC con IDE Arduino
- Foglio di calcolo (LibreOffice Calc, Excel o Google Fogli)
- Plotter seriale per grafico in tempo reale



Attività pratica !

```
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BME280.h>

Adafruit_BME280 bme;

void setup() {
  Serial.begin(9600);
  if (!bme.begin(0x76)) {
    Serial.println("Errore: BME280 non trovato!");
    while (1);
  }
}
```

```
void loop() {
  float temperatura = bme.readTemperature();
  float pressione = bme.readPressure() / 100.0F; // hPa
  float umidita = bme.readHumidity();

  Serial.print(temperatura); Serial.print(";");
  Serial.print(pressione); Serial.print(";");
  Serial.println(umidita);

  delay(5000);
}
```



Il sensore può avere indirizzo **0x76** o **0x77**. Se non funziona, prova entrambi.

Attività pratica !

Parte 1 – Acquisizione e registrazione dati

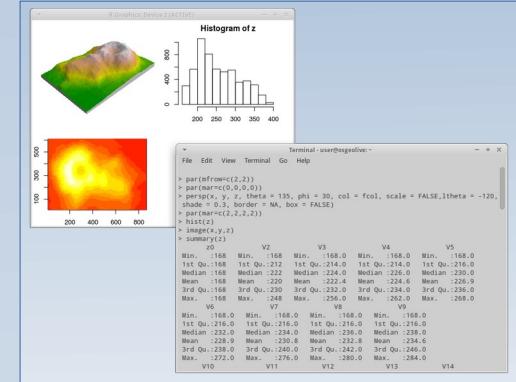
- Aprire il Plotter seriale o Monitor seriale.
- Acquisire 30–50 letture (una ogni 5 secondi).
- Copiare i dati in un foglio elettronico separando i valori (es. con ; come delimitatore).

Parte 2 – Analisi matematica/statistica

- Media aritmetica per temperatura, umidità, pressione.
- Moda (valore più frequente) e mediana.
- Deviazione standard per valutare la variabilità dei dati.
- Grafico temporale per ogni grandezza ($x = \text{tempo}$, $y = \text{valore}$).

Parte 3 – Riflessione fisica

- Relazione tra temperatura e umidità?
- La pressione varia se apriamo una finestra?
- Che effetto ha la presenza di più persone nella stanza?
- Discussione su accuratezza vs precisione.



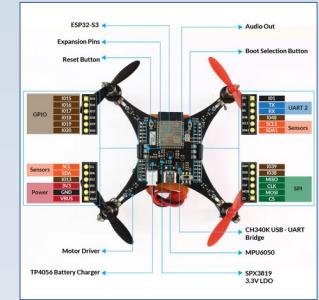
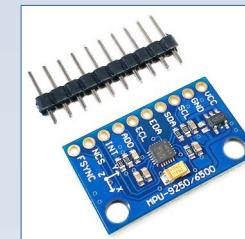
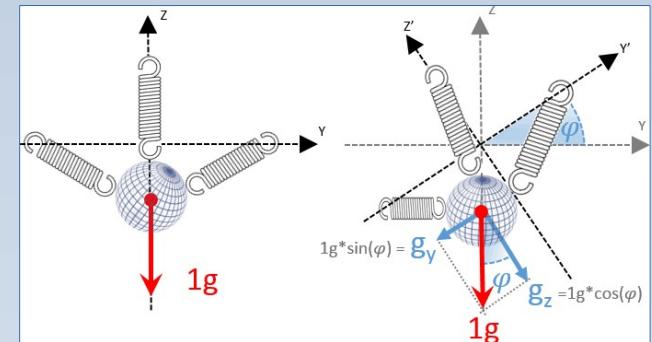
Attività pratica !

Studio del moto e accelerazione (ESP32 + accelerometro + MQTT)

Secondaria di secondo grado

Obiettivi didattici

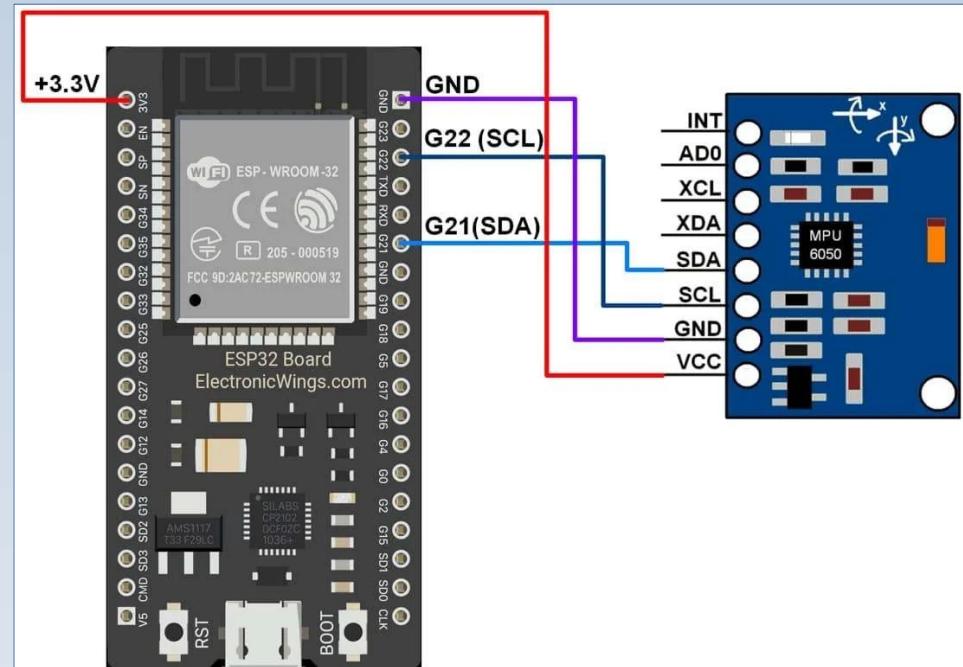
- Studiare la variazione dell'accelerazione in funzione del tempo.
- Applicare le leggi del moto (MRU, MRUA, caduta libera).
- Sviluppare competenze di acquisizione ed elaborazione dati tramite microcontrollori.
- Comprendere i concetti di sistema di riferimento, accelerazione vettoriale e composizione dei moti.
- Utilizzare strumenti digitali per osservare fenomeni reali (data logging e grafici in tempo reale).



Attività pratica !

Materiale necessario e collegamenti

- ESP32 DevKit
- Sensore MPU6050 (accelerometro + giroscopio)
- Cavi jumper
- Breadboard (facoltativa)
- PC con Arduino IDE e Plotter seriale (o MQTT + Grafana, se si estende)
- Supporto meccanico (per esempio carrellino su binario, pendolo, altalena, o una persona che cammina)



Attività pratica !

```
#include <Wire.h>
#include <MPU6050.h>

MPU6050 mpu;

void setup() {
    Serial.begin(115200);
    Wire.begin(21, 22); // SDA, SCL
    mpu.initialize();
    if (!mpu.testConnection()) {
        Serial.println("Errore: MPU6050 non trovato");
        while (1);
    }
    Serial.println("tempo(ms);acc_X(m/s^2);acc_Y(m/s^2);acc_Z(m/s^2)");
}
```

```
void loop() {
    int16_t ax, ay, az;
    mpu.getAcceleration(&ax, &ay, &az);

    // Converti in m/s2 (1g ≈ 9.81 m/s2, e sensore output: 16384 LSB/g)
    float accX = ax * 9.81 / 16384.0;
    float accY = ay * 9.81 / 16384.0;
    float accZ = az * 9.81 / 16384.0;

    Serial.print(millis()); Serial.print(";");
    Serial.print(accX); Serial.print(";");
    Serial.print(accY); Serial.print(";");
    Serial.println(accZ);

    delay(100); // 10 campioni al secondo
}
```

Attività pratica !

Parte 1 – Esperimento di osservazione

- Fissare l'ESP32 + MPU6050 a un oggetto mobile:
- Pendolo semplice
- Carrello su piano inclinato
- Ascensore o spostamento verticale
- Altalena o mano di uno studente in movimento
- Acquisire i dati accelerometrici per almeno 30 secondi.
- Esportare i dati dal monitor seriale in CSV per analisi.

Parte 2 – Elaborazione matematica/statistica

- Grafici delle componenti X, Y, Z dell'accelerazione.
- Valore assoluto dell'accelerazione totale:
$$a_{tot} = \sqrt{a_x^2 + a_y^2 + a_z^2}$$
- Media mobile per eliminare rumore.
- Analisi dei massimi, minimi e picchi per determinare fasi del moto.
- Verifica della seconda legge di Newton (eventuale confronto con forza esercitata).

Attività pratica !

Parte 3 – Discussione fisica

- Il sensore rileva accelerazione relativa rispetto al sistema di riferimento.
- Durante il moto rettilineo uniforme → accelerazione ≈ 0 .
- Durante il moto accelerato → variazione coerente con inclinazione o spinta.
- L'accelerazione lungo l'asse Z riflette la gravità, se il dispositivo è fermo.

Varianti ed estensioni

- Confronto accelerazione teorica e sperimentale su piani inclinati.
- Studio del moto armonico di un pendolo o oscillazione della mano.
- Integrazione temporale dell'accelerazione per ricavare velocità (con filtraggio).
- Versione con MQTT + Grafana/Node-RED per osservare a distanza (classe connessa).
- Calcolo energia cinetica se si stima la massa dell'oggetto mobile.



```

#include <Wire.h>
#include <Adafruit_BMP280.h>

Adafruit_BMP280 bmp; // I2C

void setup() {
  Serial.begin(9600);
  if (!bmp.begin(0x76)) { // alcuni modelli usano 0x77
    Serial.println("Errore: BMP280 non trovato. Controlla indirizzo e cablaggio.");
    while (1);
  }
}

#define TRIG_PIN 9
#define ECHO_PIN 10

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

```

The OSI Model



DHCP, DNS, FTP, HTTP, HTTPS, POP, SMTP, SSH, etc...

TCP UDP

IP Address: IPv4, IPv6

MAC Address

Ethernet cable, fibre, wireless, coax, etc...

The TCP/IP Model

Application

Transport

Internet

Network Access

