

Aliagrid

Marzo 2024

Workshop hands-on: "MicroPython, dal codice al cloud"



<https://bit.ly/workshop-unimi>

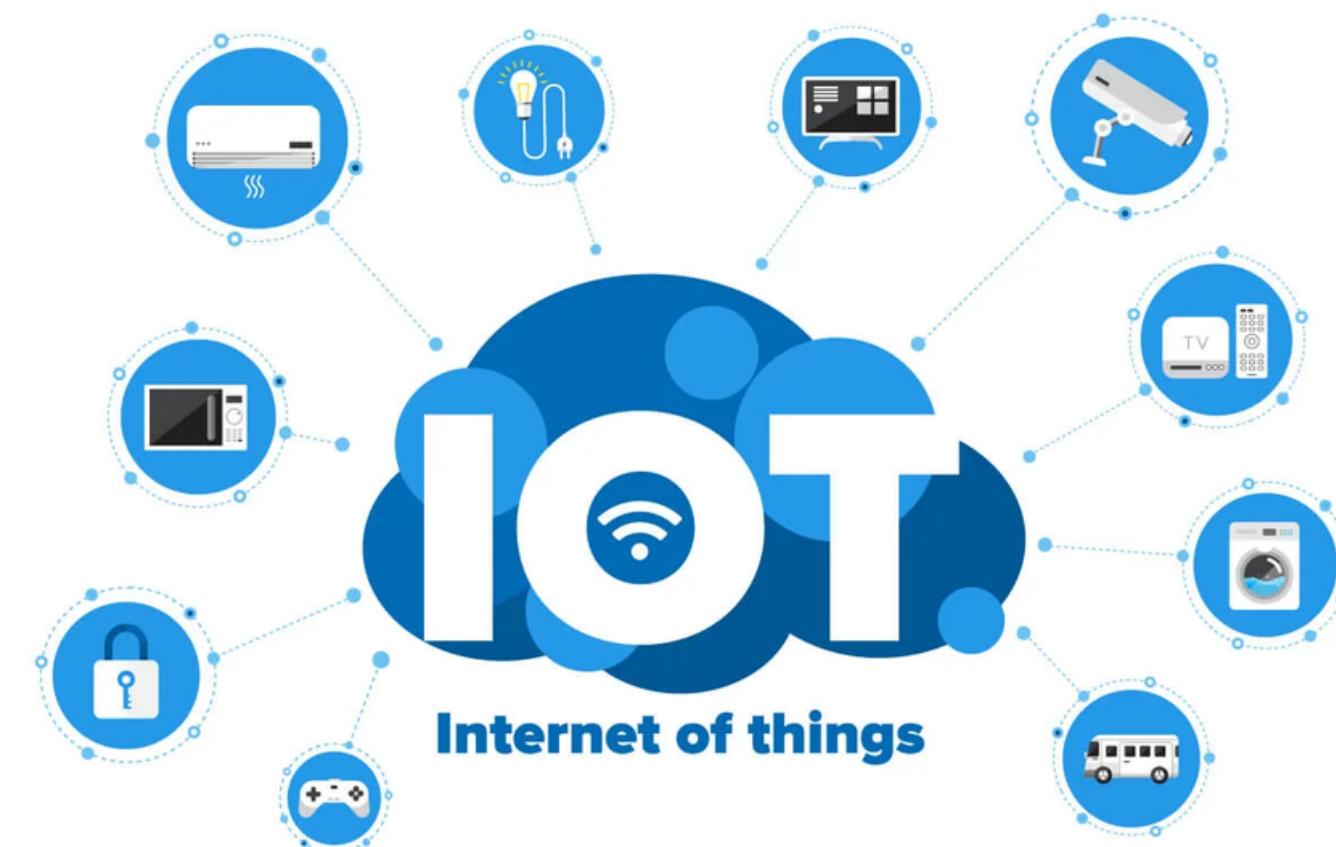
IoT

Dispositivi connessi in rete

I dispositivi comunicano

Per scambiare dati

Per intraprendere azioni



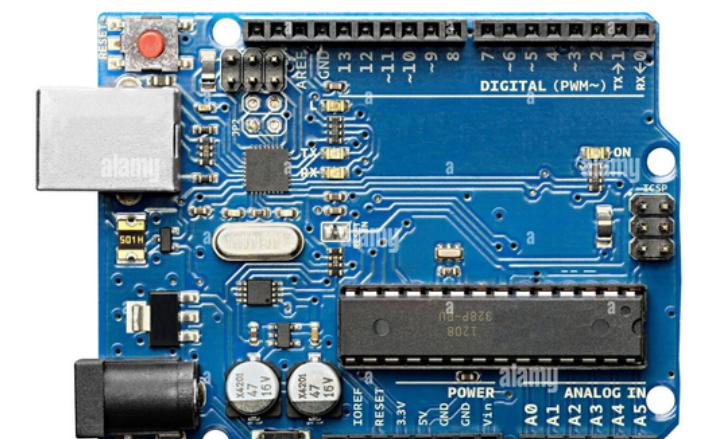
Microcontrollori vs Single-Board Computer

Microcontrollore

Microcontrollore: paragonabile ad un “computer molto piccolo”

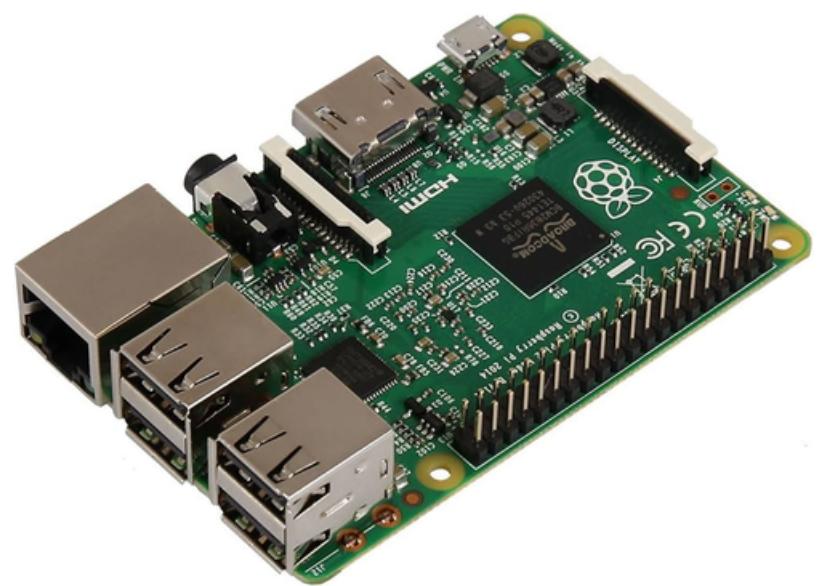
- Interagisce con il mondo esterno tramite un programma in memoria interna
- Con pin specializzati o configurabili dal programmatore (Wikipedia)

Board: la scheda su cui alloggia il microprocessore



Single-Board Computers

Hanno tutti gli elementi di un computer, in una singola board
Es: Raspberry PI

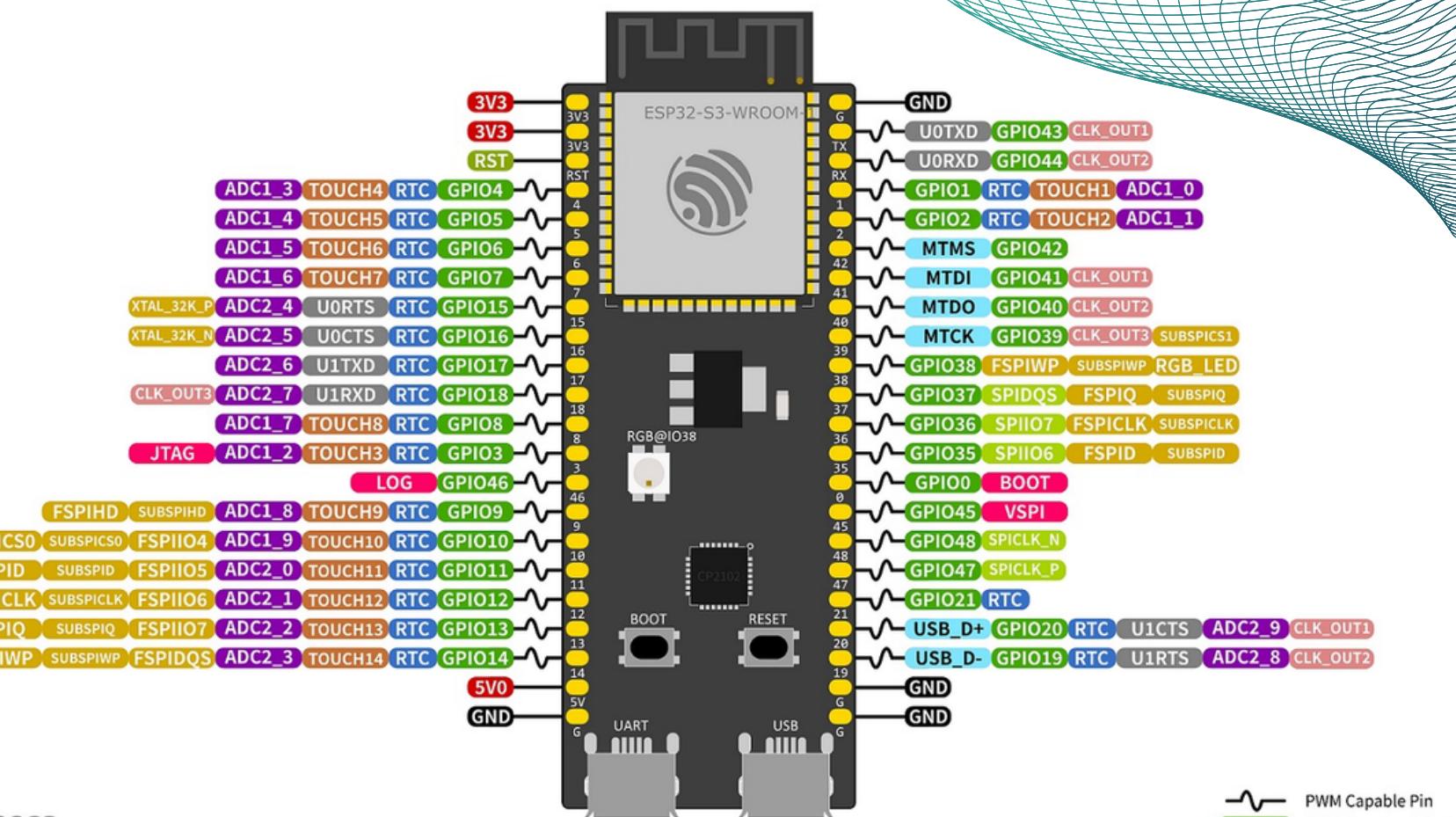


Anatomia di una board

“Pinout”: schema dei pin con nome e finalità

<https://docs.wokwi.com/guides/esp32>

ESP32-S3-DevKitC-1



ESP32-S3 Specs

32-bit Xtensa® dual-core @240MHz
Wi-Fi IEEE 802.11 b/g/n 2.4GHz + BLE 5 Mesh
512 KB SRAM (16 KB SRAM in RTC)
384 KB ROM
45 GPIOs, 4x SPI, 3x UART, 2x I2C,
14x Touch, 2x I2S, RMT, LED PWM, USB-OTG,
TWAI®, 2x 12-bit ADC, 1x LCD interface, DVP

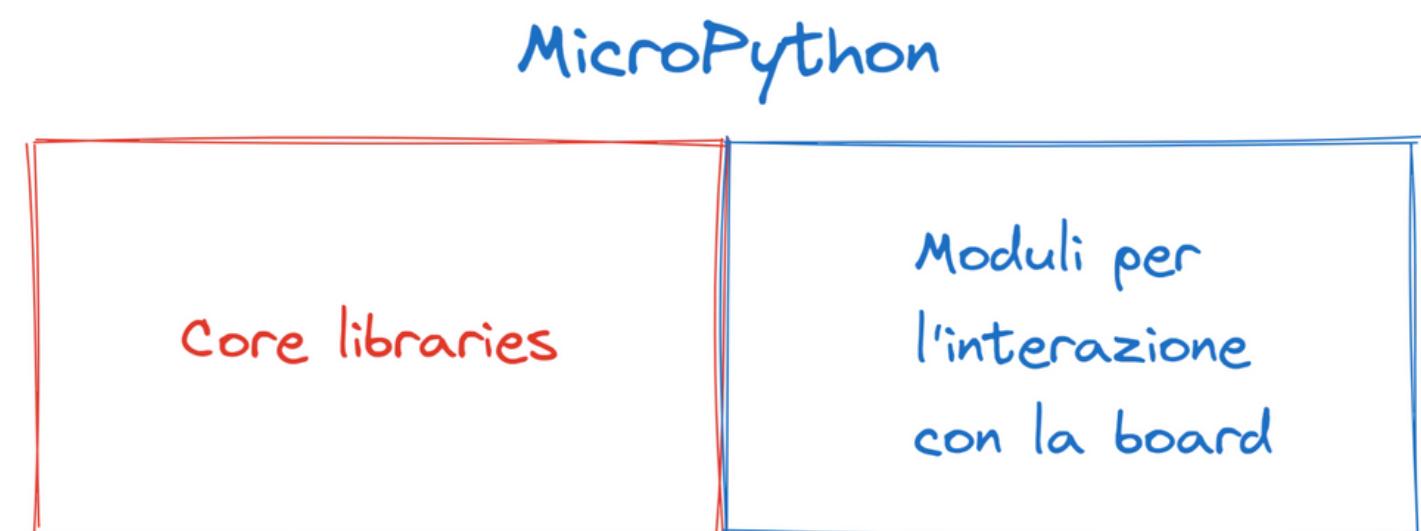
- PWM Capable Pin
 - GPIOx
 - JTAG/USB
 - ADCx_CH
 - TOUCHx
 - OTHER
 - SERIAL
 - STRAP
 - RTC
 - MISC
 - GND
 - PWD
- Miscellaneous/SPI functions
- Clock Output

Cos'è Micropython?

Una versione “light” di **Python** che può girare sui **microcontrollori**

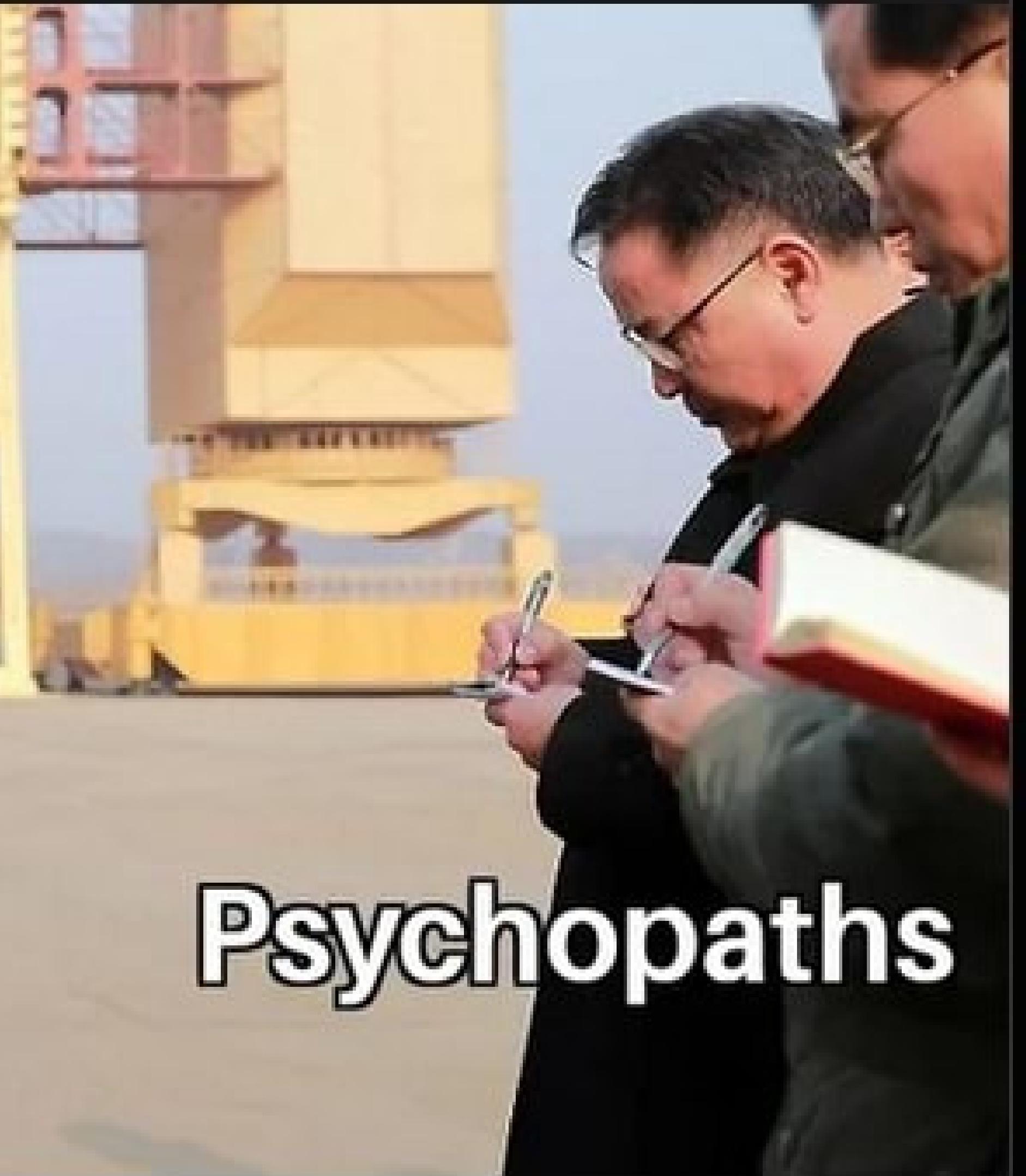
- entra in **256k** di **codice** e **16kb** di **RAM**
- vuole essere più compatibile possibile con Python
- Permette di **interagire** con le **componenti** della **board**

Nota: non tutte le board lo supportano!





People who program
embedded systems
with Python

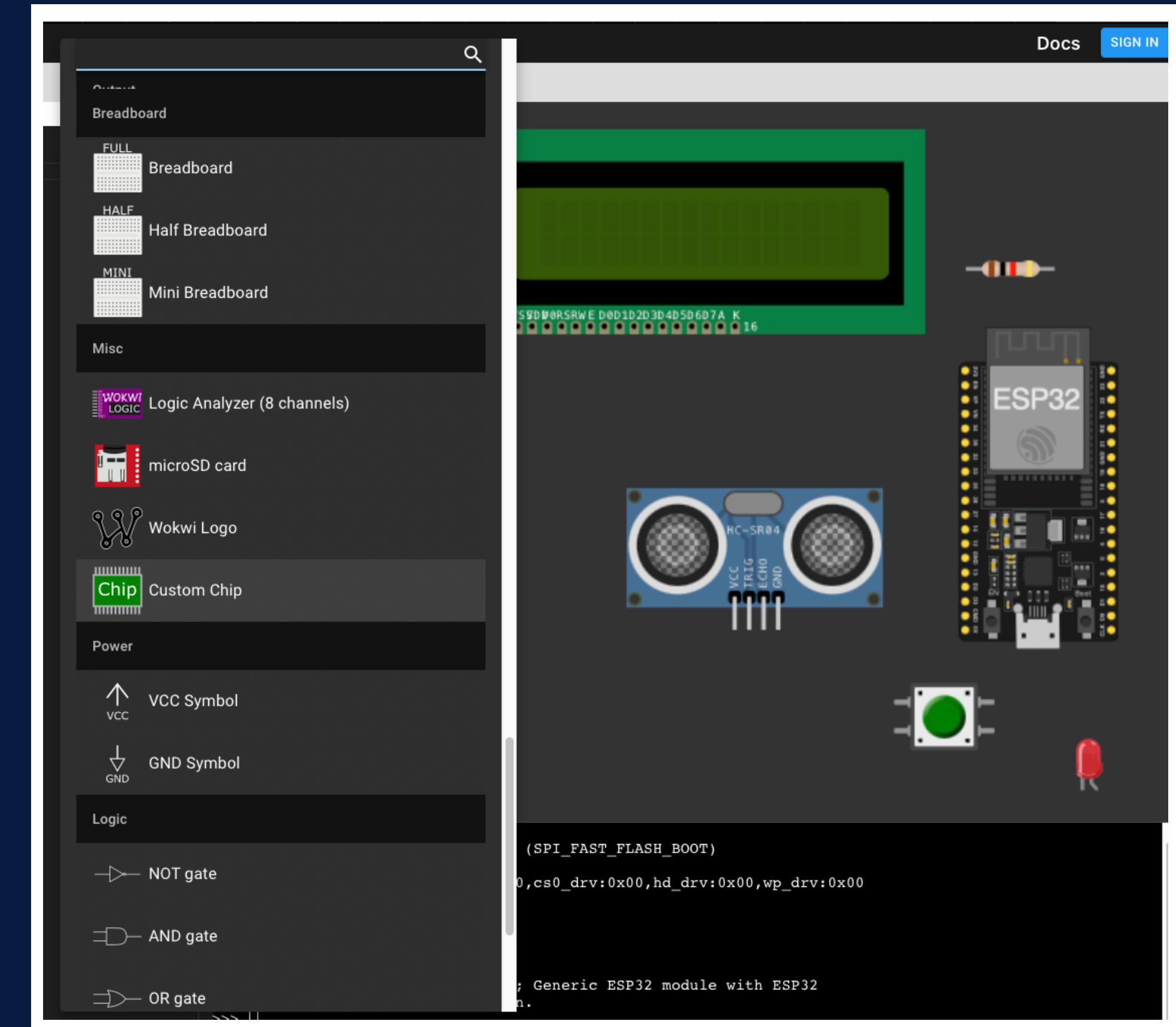


Psychopaths

Simulatore di circuiti

- Permette di fare prove senza paura di rovinare l'hardware
- Offre tantissimi componenti senza doverli andare a comprare

<https://wokwi.com/>

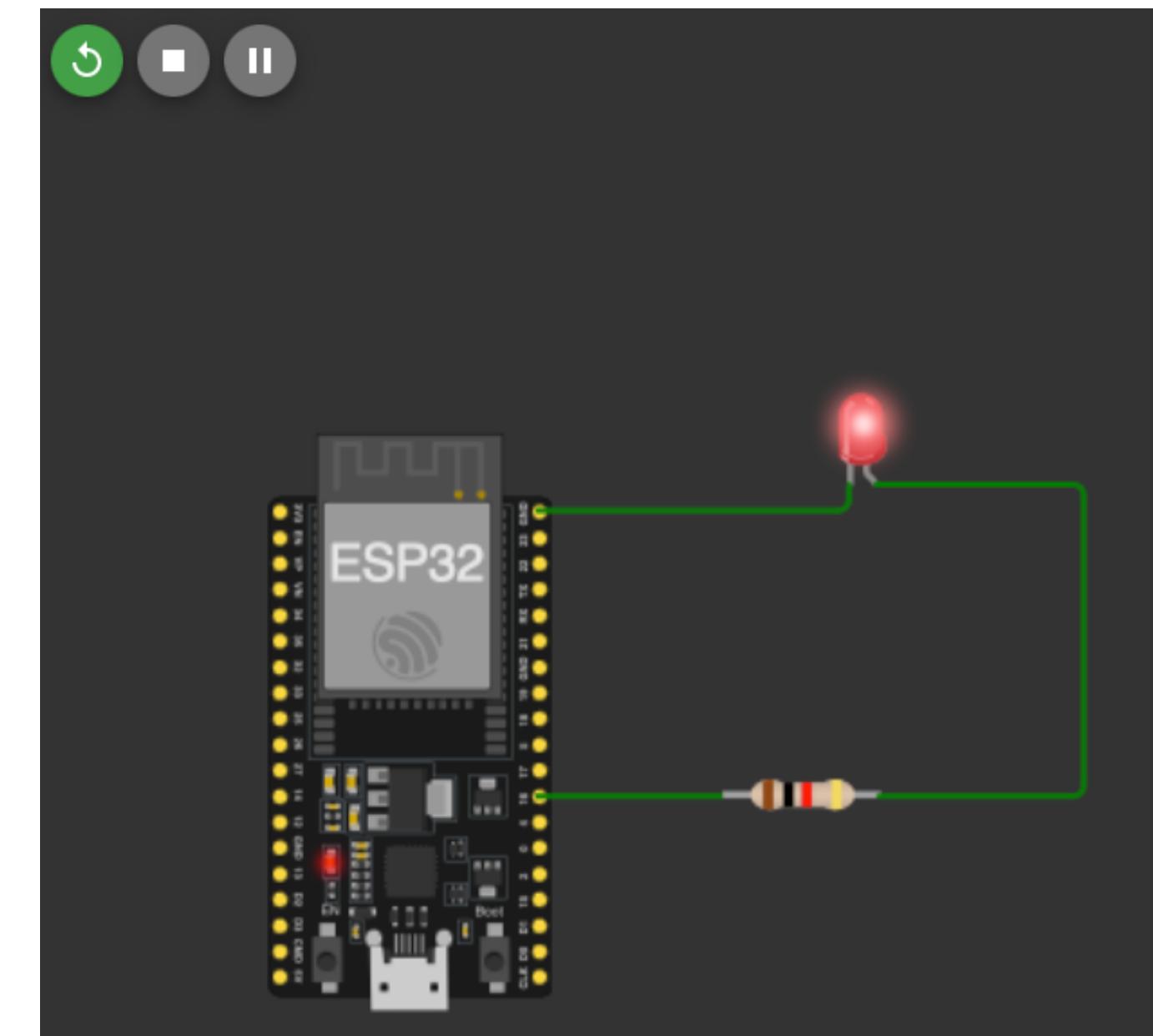


Accendere un led

- **Differenza di potenziale** (DDP): genera un flusso di corrente
- Una **pila** induce una differenza di potenziale – es: “una pila da 9V”
- La corrente circola solo in un circuito chiuso

Esempio

```
from machine import Pin  
led = Pin(16, Pin.OUT)  
led.value(1)
```



Termometro NTC

- **NTC** (Negative Temperature Coefficient): la **resistenza decresce con l'aumentare della temperatura**
- 3 pin:
 - **VCC**, l'alimentazione
 - **OUT**, la "lettura"
 - **GND**, la connessione al GND dell'ESP32

Esempio



Pin names

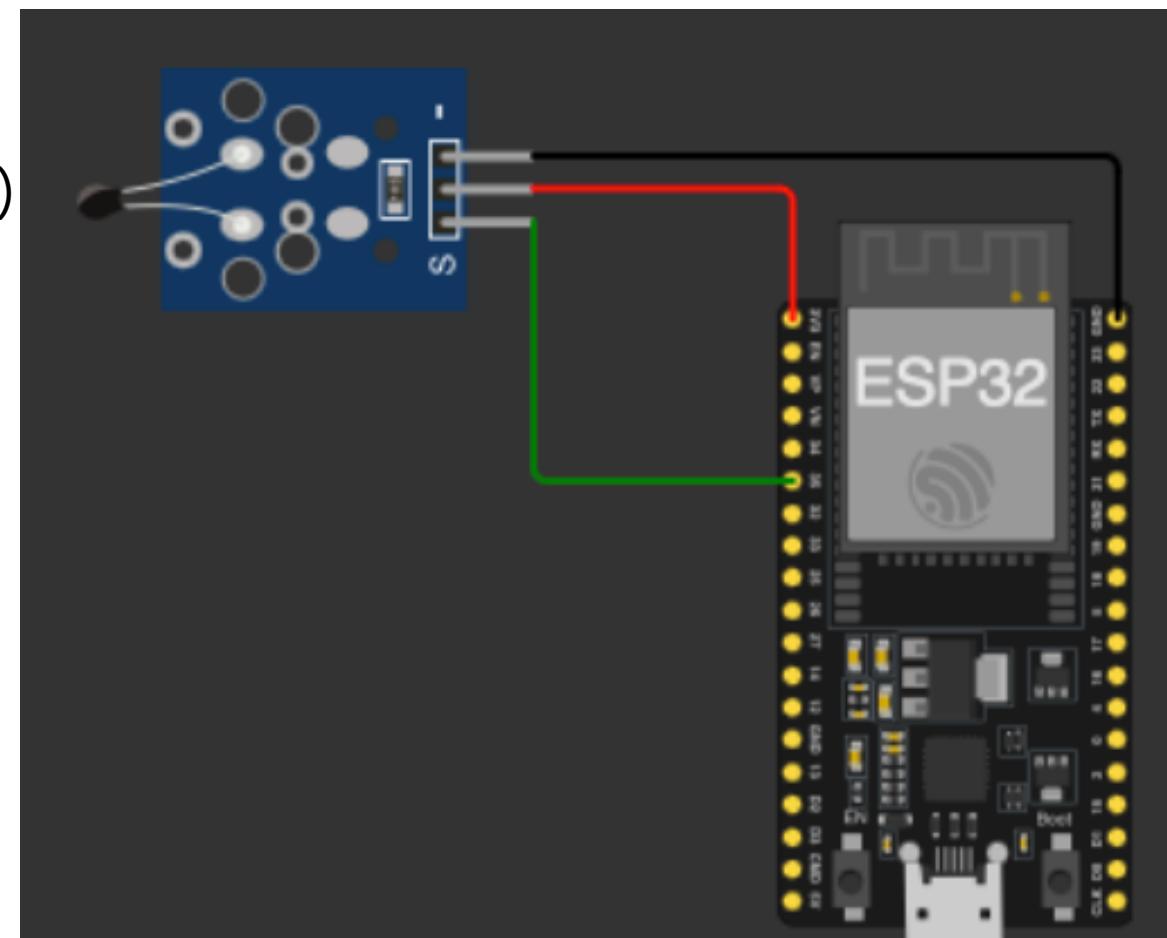
Name	Description
VCC	Positive power supply
OUT	Output signal (analog)
GND	Ground

```
from machine import Pin, ADC
import time
import math

BETA = 3950 #Beta coefficient of the
thermistor

p35 = ADC(Pin(35))
p35.width(ADC.WIDTH_10BIT)

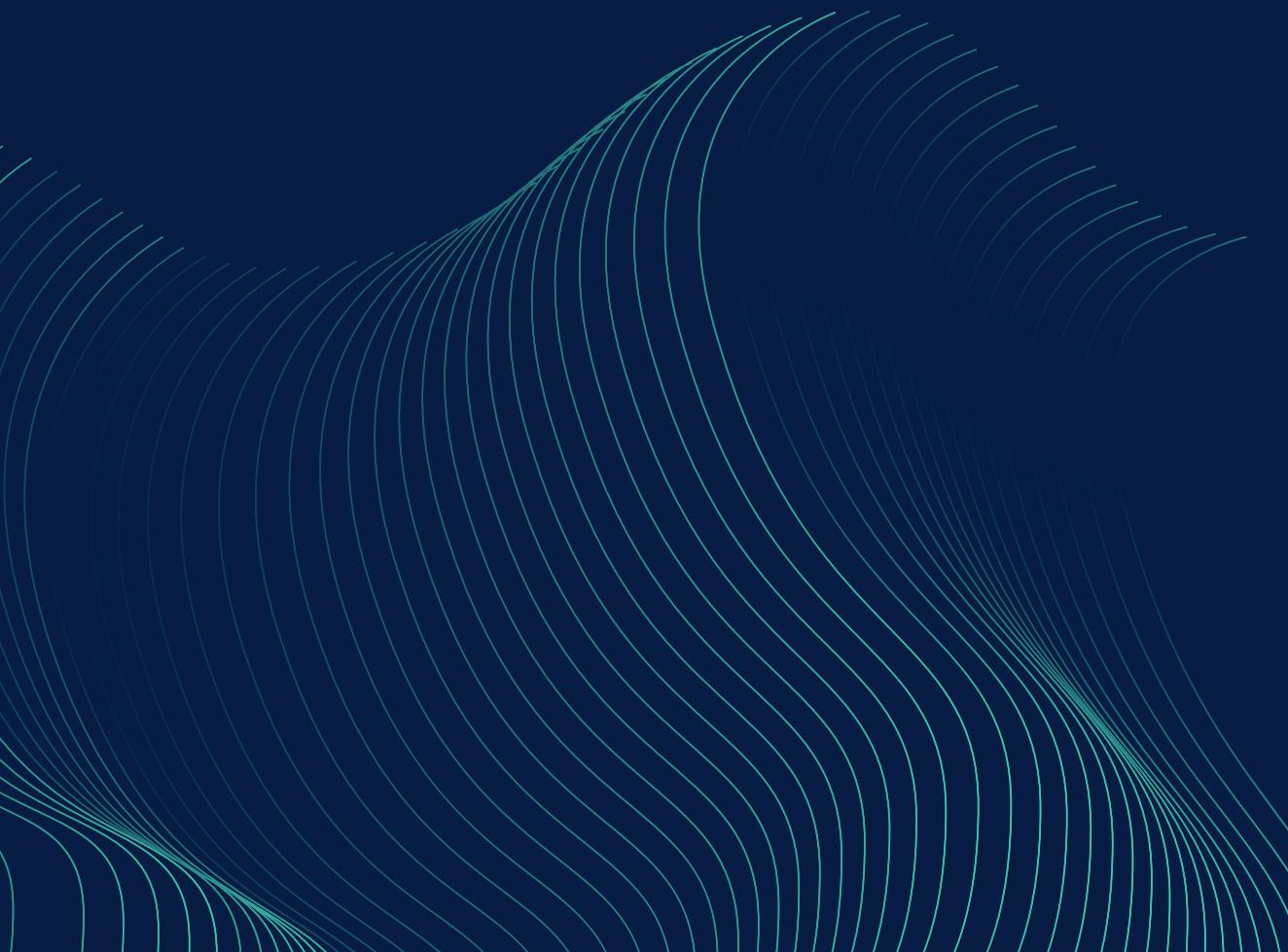
while True:
    v = p35.read()
    print(v)
    celsius = 1 / (math.log(1 / (1023 / v - 1))
    / BETA + 1.0 / 298.15) - 273.15
    print(celsius)
    time.sleep(2)
```



Breadboard pt1

TM1637

- **CLK**: serve a capire quando iniziano/finiscono le trasmissioni di dati
- **DIO**: il pin su cui scrivere i dati
- **VCC**: l'alimentazione
- **GND**: la connessione al GND dell'ESP32



Connessione in orizzontale

Connessione in verticale

1 e 2 non sono connessi



Pin names

Name	Description
CLK	Clock input
DIO	Data input *
VCC	Supply voltage
GND	Ground

Breadboard pt2

- Un “driver” scrive sul bus al posto nostro:
- ...secondo un “quasi” protocollo I2C
- ...sequenze di start/stop dei comandi/dati ...
- come accendere i segmenti

Specifica nei datasheet

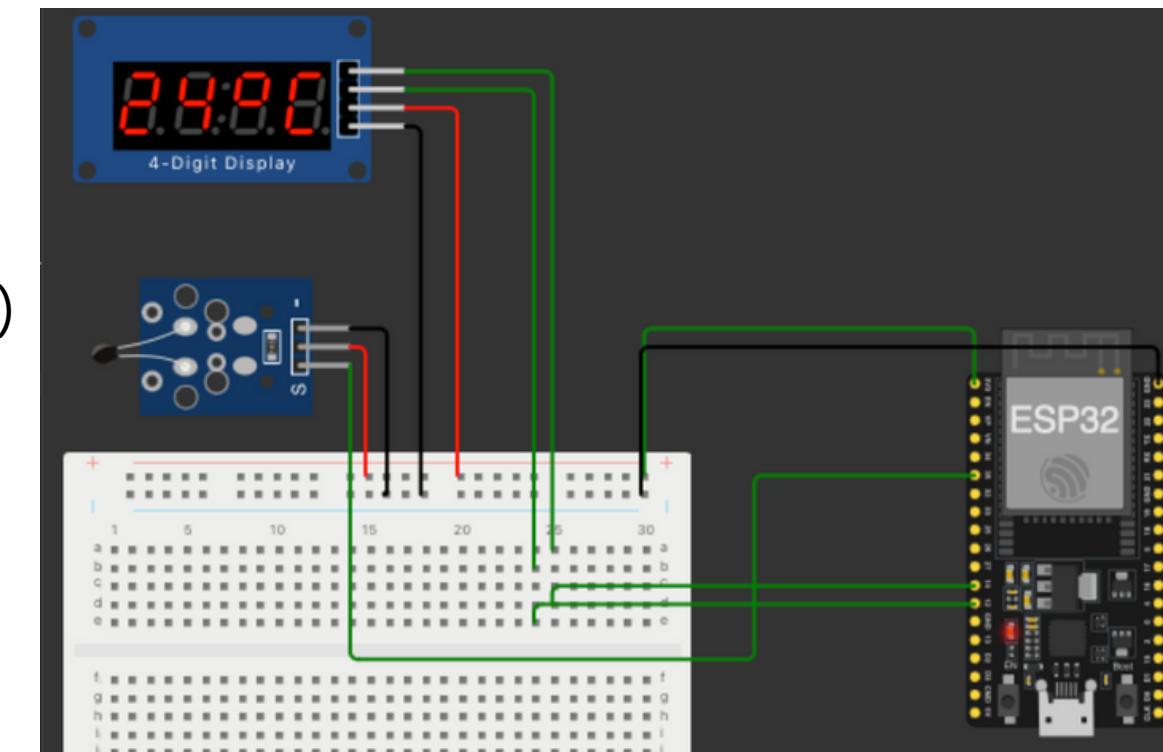
Esempio

```
from machine import Pin, ADC
import time
import math
import tm1637
display = tm1637.TM1637(clk = Pin(14),
dio = Pin(12))
```

BETA = 3950 #Beta coefficient of the
thermistor

```
p35 = ADC(Pin(35))
p35.width(ADC.WIDTH_10BIT)
```

```
while True:
    v = p35.read()
    print(v)
    celsius = 1 / (math.log(1 / (1023 / v - 1))
    / BETA + 1.0 / 298.15) - 273.15
    print(celsius)
    display.temperature(round(celsius))
    time.sleep(2)
```



Caso d'esempio

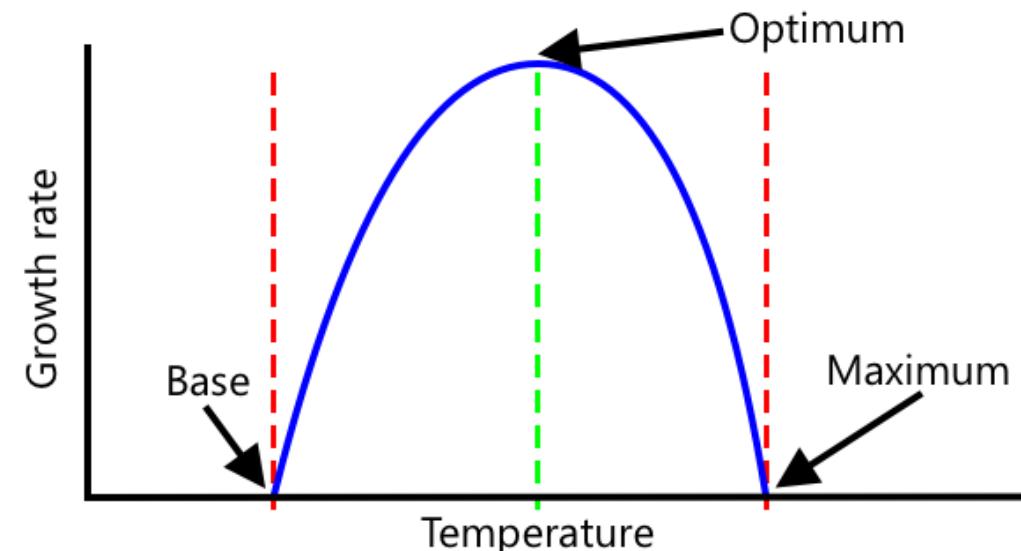
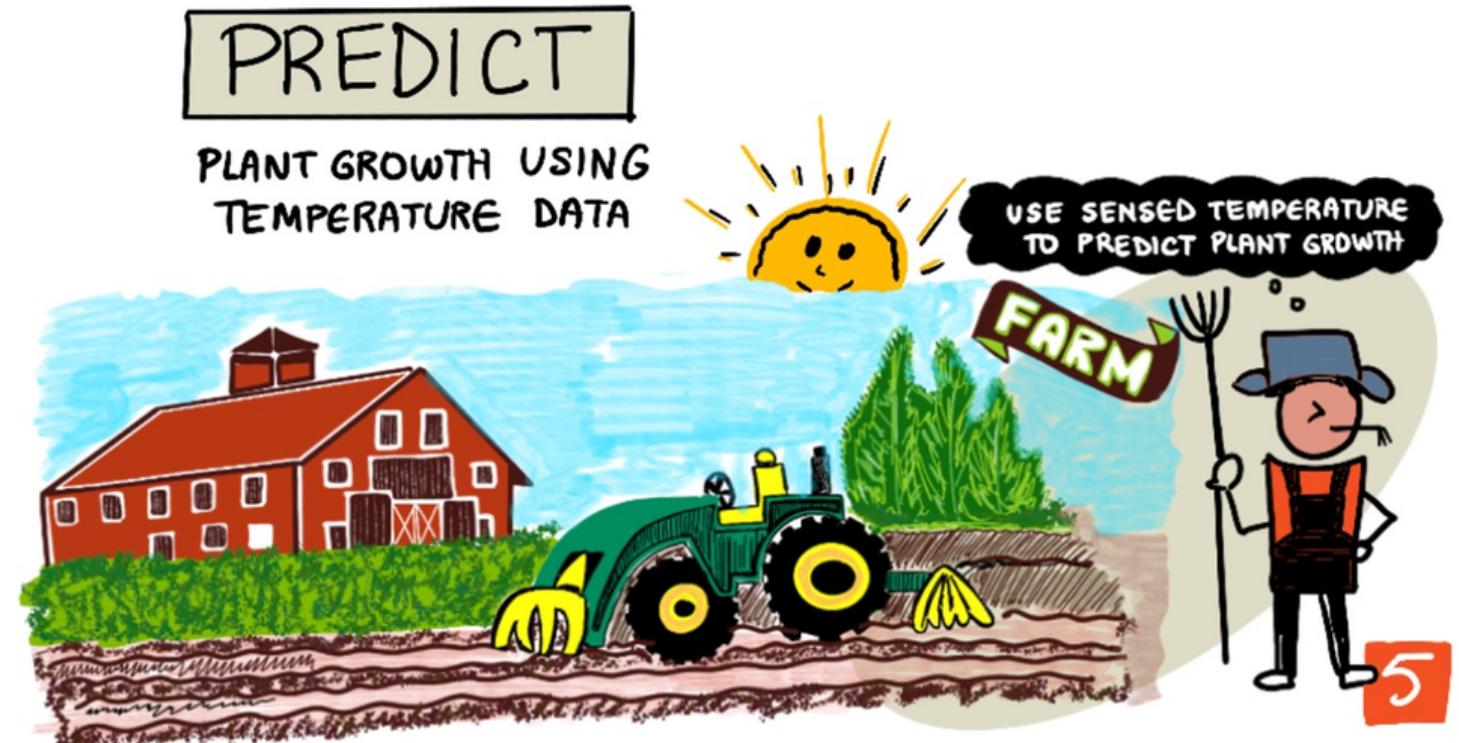
Growing Degree Days (GDD): misurano la crescita delle piante in base alla temperatura

Sono calcolati:

- su base quotidiana
- con la temperatura media al di sopra di una certa baseline
 - la baseline dipende dalla coltura

Ogni coltura necessita di un certo numero di GDD per crescere/fiorire/produrre raccolto

<https://github.com/microsoft/IoT-Farm-Beginners/blob/main/2-farm/lessons/1-predict-plant-growth/README.md>



**YOU BLINKED A LED
WITH A DOUNC?**



**TELL ME MORE ABOUT HOW YOU'RE
AN EMBEDDED SYSTEMS ENGINEER**

Binario micropython

I "binari di MicroPython" si riferiscono ai **file eseguibili** o immagini del firmware che contengono **l'implementazione di MicroPython** e che possono essere caricati su un microcontroller per eseguire il codice Python.



Interprete

L'interprete MicroPython è il cuore del sistema e consente al microcontroller di **eseguire** il codice **Python**. **Include un'implementazione delle specifiche del linguaggio Python** che è **adattata** per funzionare in **contesti embedded**, dove potrebbero essere presenti **limitazioni di memoria, potenza di calcolo e altre risorse**.

Flash e caricamento:

Dopo la compilazione, il **firmware MicroPython** deve essere **caricato** sulla **memoria flash** del microcontroller. Questo processo può variare a seconda del tipo di microcontroller e della piattaforma di sviluppo utilizzata. In molti casi, vengono utilizzati strumenti come **esptool** per caricare il firmware sulla memoria flash del microcontroller.

Cross-compilazione:

Spesso, il **firmware** MicroPython viene **compilato** in un **ambiente di sviluppo diverso** da quello in cui verrà eseguito effettivamente sul microcontroller. Questo processo è noto come **cross-compilazione**.

Estensioni e moduli aggiuntivi:

I binari di MicroPython possono includere diverse **estensioni** o **moduli specifici** per il **supporto hardware**, consentendo l'interazione diretta con i **componenti** del microcontroller, come **sensori, attuatori** e altro ancora.

Thonny ed esptool

Per utilizzare Micropython sulla vostra scheda **scaricate il binario** corrispondente da [questo link](#)

Se utilizzate ESP32 come Port, dovete installare esptool per poter interagire con la scheda.

Per farlo potete usare **PiP**:

```
pip install esptool
```

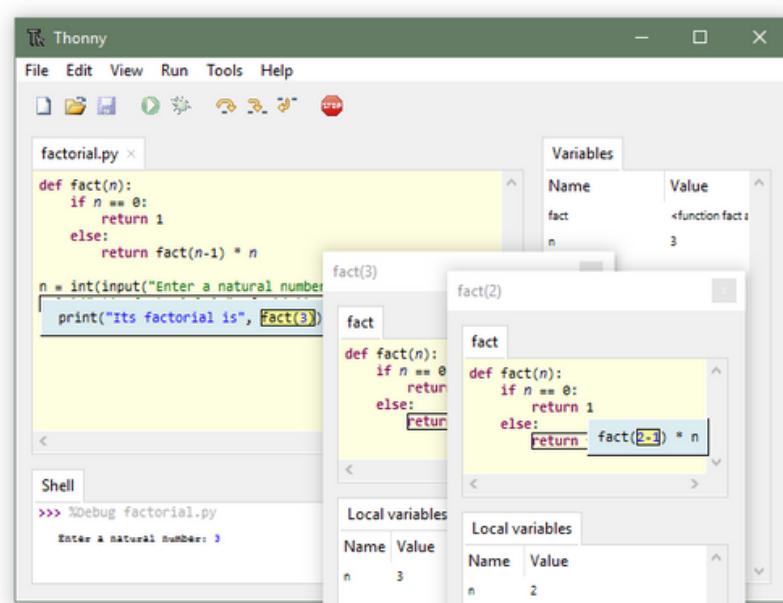
A questo punto dovete prima **"brasare"** la flash con :
`esptool.py --chip esp32 --port /dev/ttyUSB0 erase_flash`

(Specificate la porta)

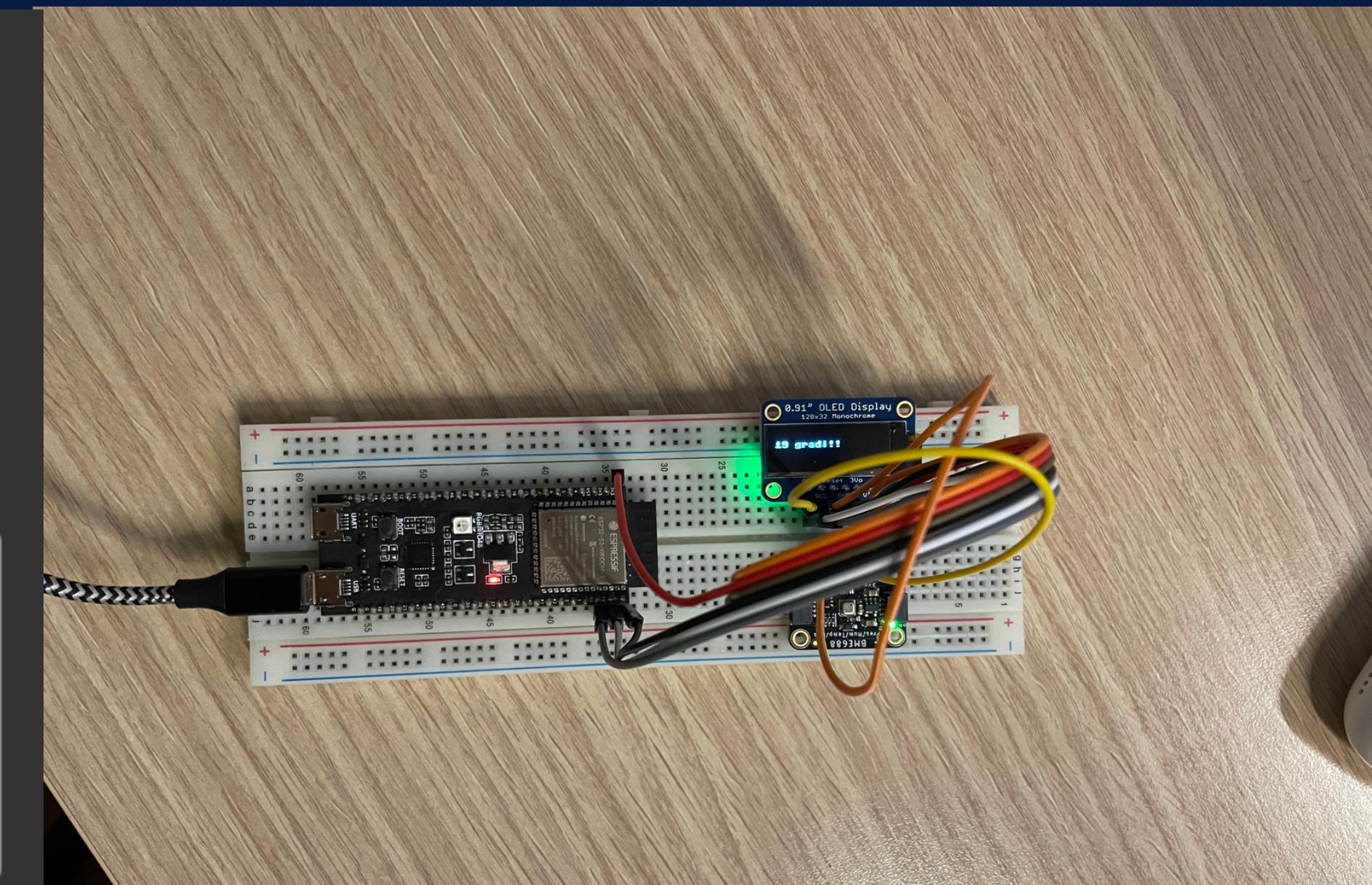
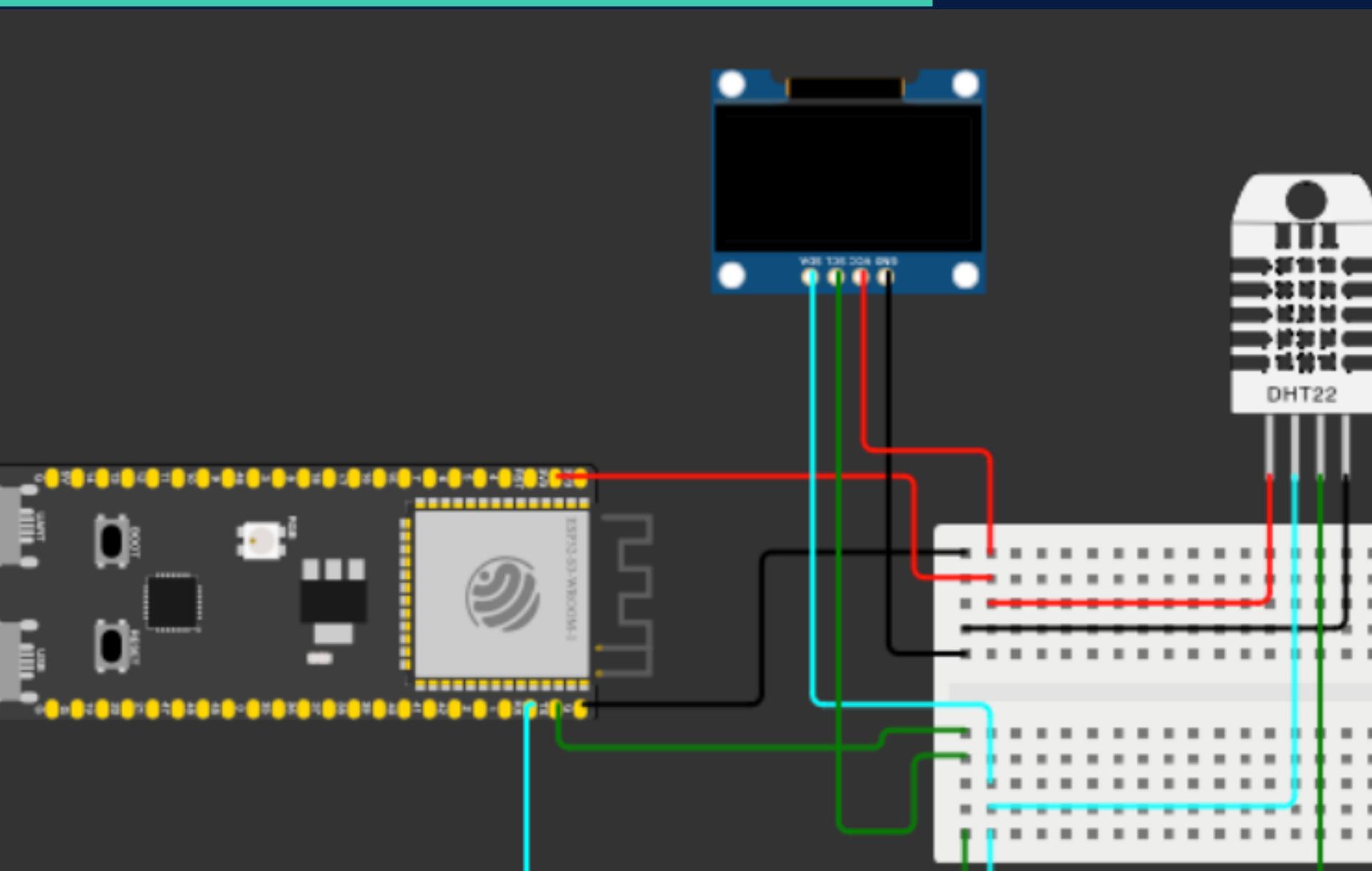
Infine **caricate** il binario sulla scheda con:
`esptool.py --chip esp32 --port /dev/ttyUSB0 --baud 460800 write_flash -z 0x1000 esp32-20190125-v1.10.bin`

(Specificate la porta e la posizione del binario nel vostro file System)

In ultimo, per **agevolare la scrittura del codice** consigliamo l'utilizzo di [Thonny](#). Un **IDE** brutta, ma che ha quanto serve.



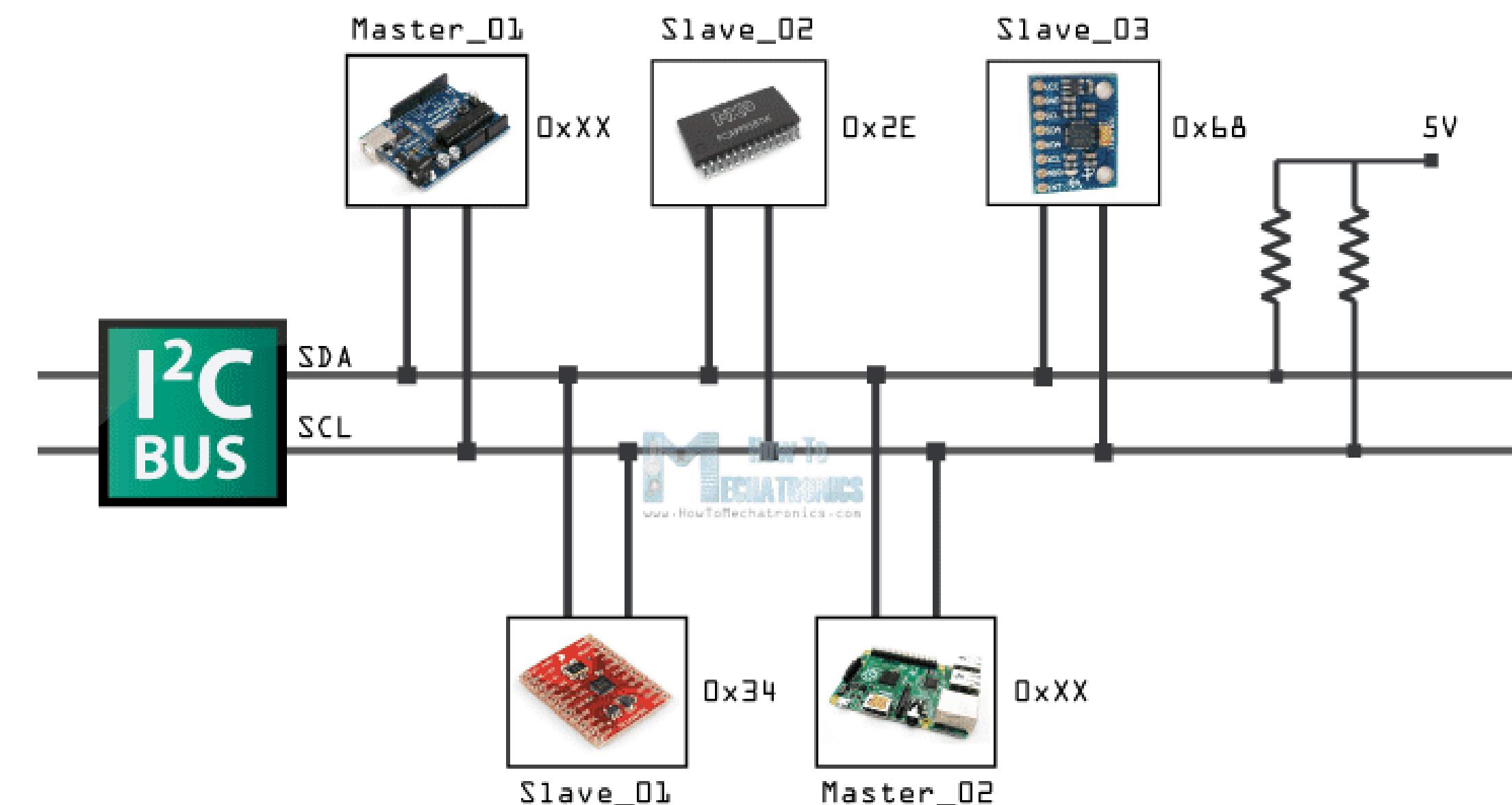
Collegamento su una breadboard vera



I²C

Come fanno i 2 sensori ad utilizzare gli **stessi pin** dell'ESP ?

Con **I²C**, un sistema di **comunicazione seriale** bifilare utilizzato tra circuiti integrati.



Termometro I2C

BME688

- Integrato con sensori di **pressione**, **umidità** e **temperatura** ad alta linearità e alta precisione.
- Il **sensore di gas** può rilevare Composti Organici Volatili (**voc**), composti solforati volatili (**vsc**) e altri gas come monossido di carbonio e idrogeno.
- La capacità di rilevamento si estende nell'intervallo delle parti per miliardo (**ppb**).

- Per interagire con il sensore abbiamo realizzato una **libreria** che rende semplice la lettura dei valori
- Potete scaricarla da [qui](#) ed inserirla nella cartella lib
- Occhio che **non tutti i pin dell'esp supportano I2C**



Webserver

Come leggiamo i dati senza schermo?

→ ⌂ ▲ Non sicuro 192.168.197.118

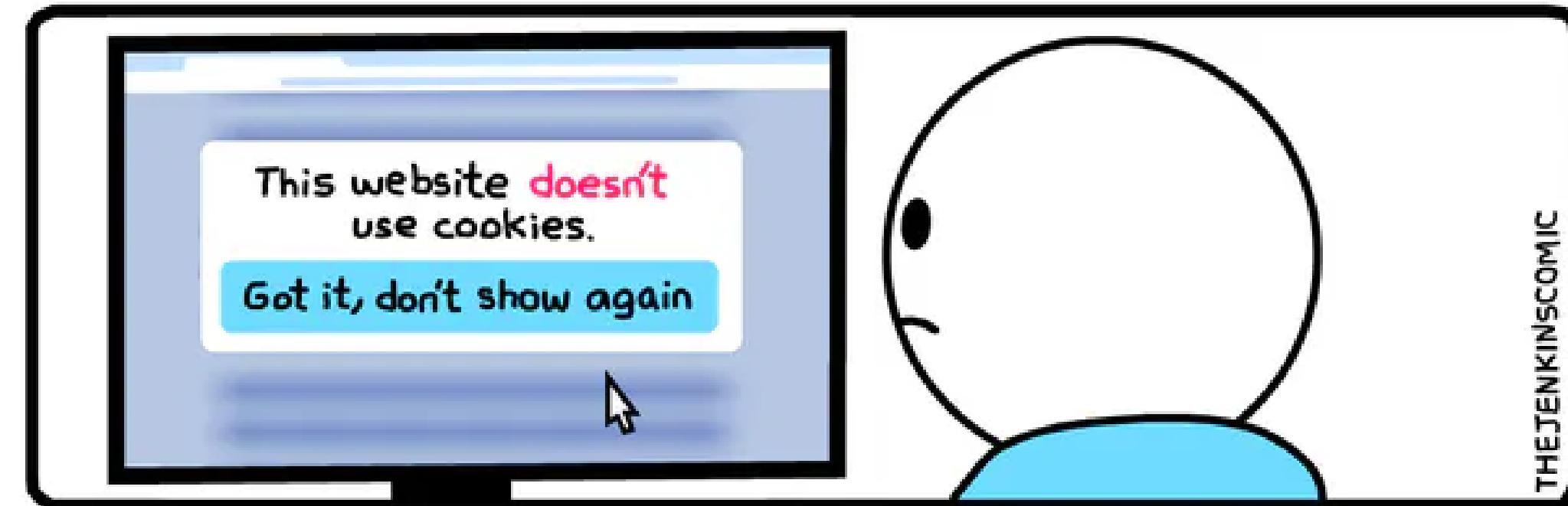
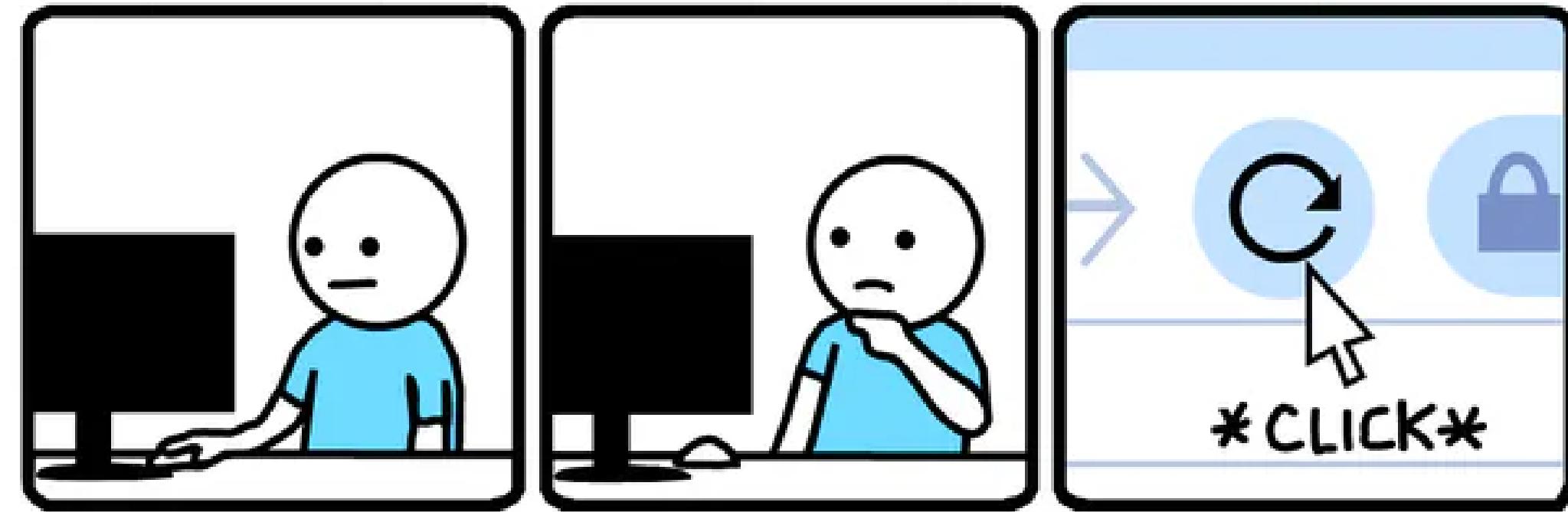
- Potremmo utilizzare un **webserver** che, connesso ad una **rete locale**, esponga il valore di temperatura nel payload html.
- Per farlo ci sono varie librerie, ad esempio **micropdot**.
- Per esplorare maggiormente le possibilità di micropython lo creeremo tramite **socket**

BME688 Demo

Temperature in Celcius: 19,5

Gas in Ohm: 57,9

Pressure in Pascal: 987.0700073242188



AWS

Amazon Web Services (AWS) è una piattaforma di servizi cloud offerta da Amazon.

OLTRE 200 SERVIZI DISPONIBILI

OLTRE 30 AVAILABILITY ZONE
WORLDWIDE



and more...



AWS IoT Core

Cos'è?

Amazon IoT Core è un servizio offerto da Amazon Web Services (**AWS**) progettato per **semplificare** la **gestione** dei dispositivi Internet of Things (**IoT**) e per consentire la **comunicazione** tra questi dispositivi e le applicazioni cloud.

Scalabilità e performance

- AWS IoT Core è progettato per gestire **grandi flotte** di **dispositivi IoT** in modo scalabile ed efficiente (**>1Bln**).
- Supporta la **comunicazione bidirezionale** tra i dispositivi e le applicazioni cloud, garantendo una **bassa latenza** e **un'alta affidabilità**.

Gestione dei dispositivi

- AWS IoT Core semplifica la **registrazione**, la gestione e la **comunicazione** tra i dispositivi IoT e le applicazioni cloud.
- Supporta l'utilizzo di protocolli standard, come **MQTT** e **HTTPS**, e offre funzionalità avanzate come la **gestione delle chiavi** e l'autenticazione dei dispositivi.

Integrazione con Altri Servizi AWS

- AWS IoT Core si **integra facilmente** con **altri servizi AWS**, consentendo agli sviluppatori di costruire soluzioni IoT complesse e complete.
- Ad esempio, è possibile utilizzare AWS **IoT Rules Engine** per **filtrare** e instradare i dati IoT, o sfruttare servizi come AWS Lambda per eseguire codice senza gestire l'infrastruttura.



Concetti di IoT Core



Things

In AWS IoT Core, una "**Thing**" rappresenta una **risorsa virtuale** che può essere un **dispositivo** fisico, una parte di **attrezzatura**, **un'applicazione** o qualsiasi altra entità che può inviare o ricevere dati tramite Internet.

Certificates

I "**Certificates**" (certificati) sono utilizzati per **autenticare** le **Things** e garantire una **comunicazione sicura**.

Ogni Thing può avere un certificato X.509 che viene utilizzato per autenticare e **crittografare** la **comunicazione** con AWS IoT Core.

Gestione dei dispositivi

Le "**Policies**" in AWS IoT Core sono **regole di sicurezza** che specificano quali azioni possono essere eseguite su **quali risorse da parte di quali entità** (come Things o utenti).

Esempio: Una policy può **concedere** a una **Thing** il **permesso** di **pubblicare** dati su un determinato topic MQTT.

Message routing

Il "**Message Routing**" (instradamento dei messaggi) in AWS IoT Core permette di **definire regole** per il **filtraggio** e l'**instradamento automatico** dei messaggi tra le Things e altri servizi AWS.

Collegarsi con micropython



Autenticazione tramite certificati SSL

- Come discusso nelle precedenti slide, per poter inviare dati alla rete AWS è necessario **autenticarsi** con dei certificati.

- Per **cifrare la connessione** è possibile usare la classe **SSL** (anch'essa nativa dalla versione **1.22** di upy)
- é quindi necessario **passare i path** dei **certificati** e istanziare un oggetto SSL da passare poi alla classe mqtt,

```
76  
77 def _get_ssl_params(keyfile, certfile):  
78     #Get ssl parameters for MQTT  
79     # These keys must be in der format the keys  
80     # downloaded from AWS website is in pem format  
81     ssl = SSLContext(ssl.PROTOCOL_TLS_CLIENT)  
82     ssl.load_cert_chain(certfile, keyfile)  
83     return ssl  
84
```

Collegarsi con micropython



Uso della classe umqtt.simple

Per poter inviare messaggi al cloud è necessario scegliere uno dei due protocolli supportati da IoT Core: **MQTT** ed HTTP.

```
54 def mqtt_connect(client=client_id, endpoint=aws_endpoint, sslp=ssl_params):
55     mqtt = MQTTClient(client_id=client, server=endpoint, port=8883, keepalive=1200, ssl=True, ssl_params=sslp)
56     print("Connecting to AWS IoT...")
57     mqtt.connect()
58     print("Done")
59     return mqtt
60
61 def mqtt_publish(client, topic=topic_pub, message=''):
62     print("Publishing message...")
63     client.publish(topic, message)
64     print(message)
```

Usando **MQTT**, che abbiamo già visto in precedenza, si può utilizzare la classe (nativa) **umqtt.simple**, che astrae il protocollo rendendo molto semplice l'invio di **payload**.

Inviamo i dati via MQTT

Tramite la console di AWS è quindi **possibile vedere** i messaggi che stanno arrivando sul topi corrispondente

```
{  
    "Id_dev": 33,  
    "Temp": 18.9,  
    "Pressure": 983.327,  
    "Gas": 65.9  
}
```

► Properties

▼ test_topic/esp32

February 20, 2024, 12:24:35 (UTC+0100)

```
{  
    "Id_dev": 33,  
    "Temp": 19.3,  
    "Pressure": 980.139,  
    "Gas": 58.2  
}
```

► Properties

▼ test_topic/esp32

February 20, 2024, 12:23:31 (UTC+0100)

```
{  
    "Id_dev": 33,  
    "Temp": 19.4,  
    "Pressure": 988.339,  
    "Gas": 58.5  
}
```

Scriviamo su DynamoDB

Per贮are in modo semplice le misure,
il servizio più comodo è sicuramente
DynamoDB, un **Database noSQL** che è
direttamente integrato con IoT Core.

Test_esp32

Autopreview View table details

Scan or query items

Scan Query

Select a table or index Table - Test_esp32 Select attribute projection All attributes

Filters

Run Reset

Completed. Read capacity units consumed: 0.5

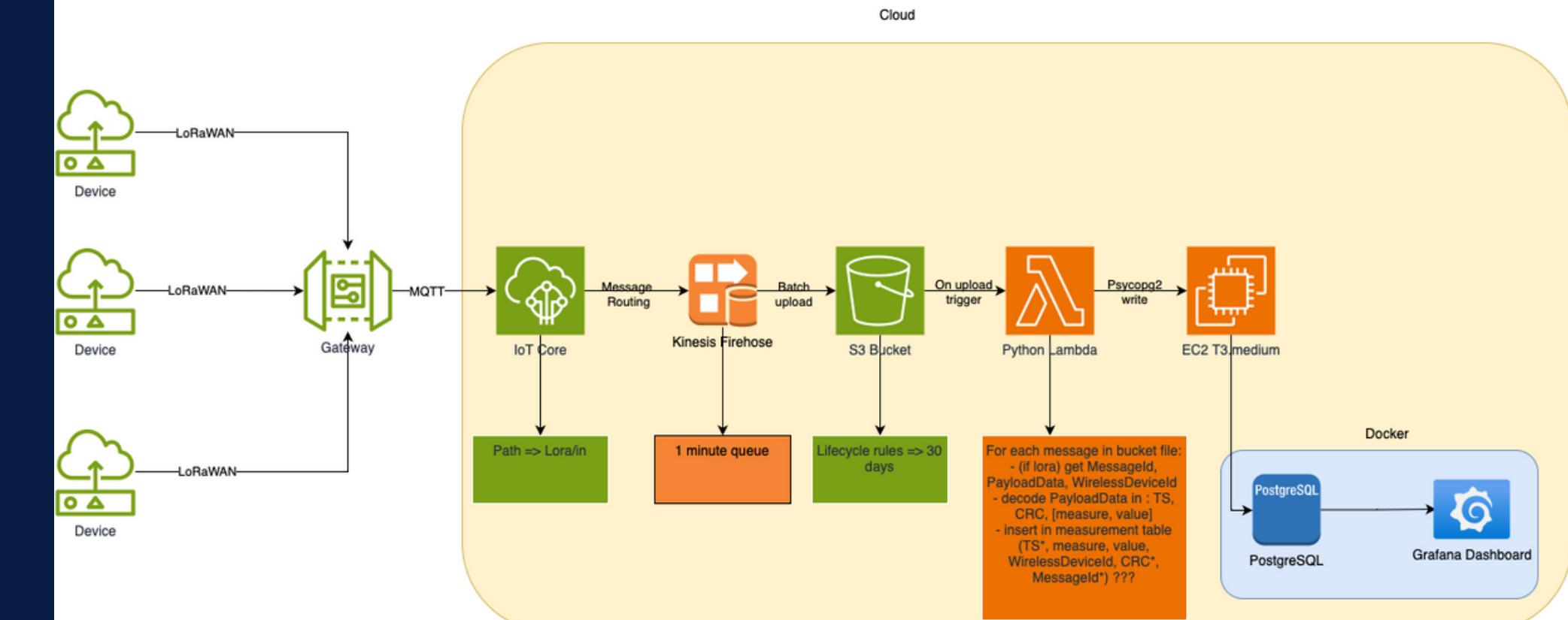
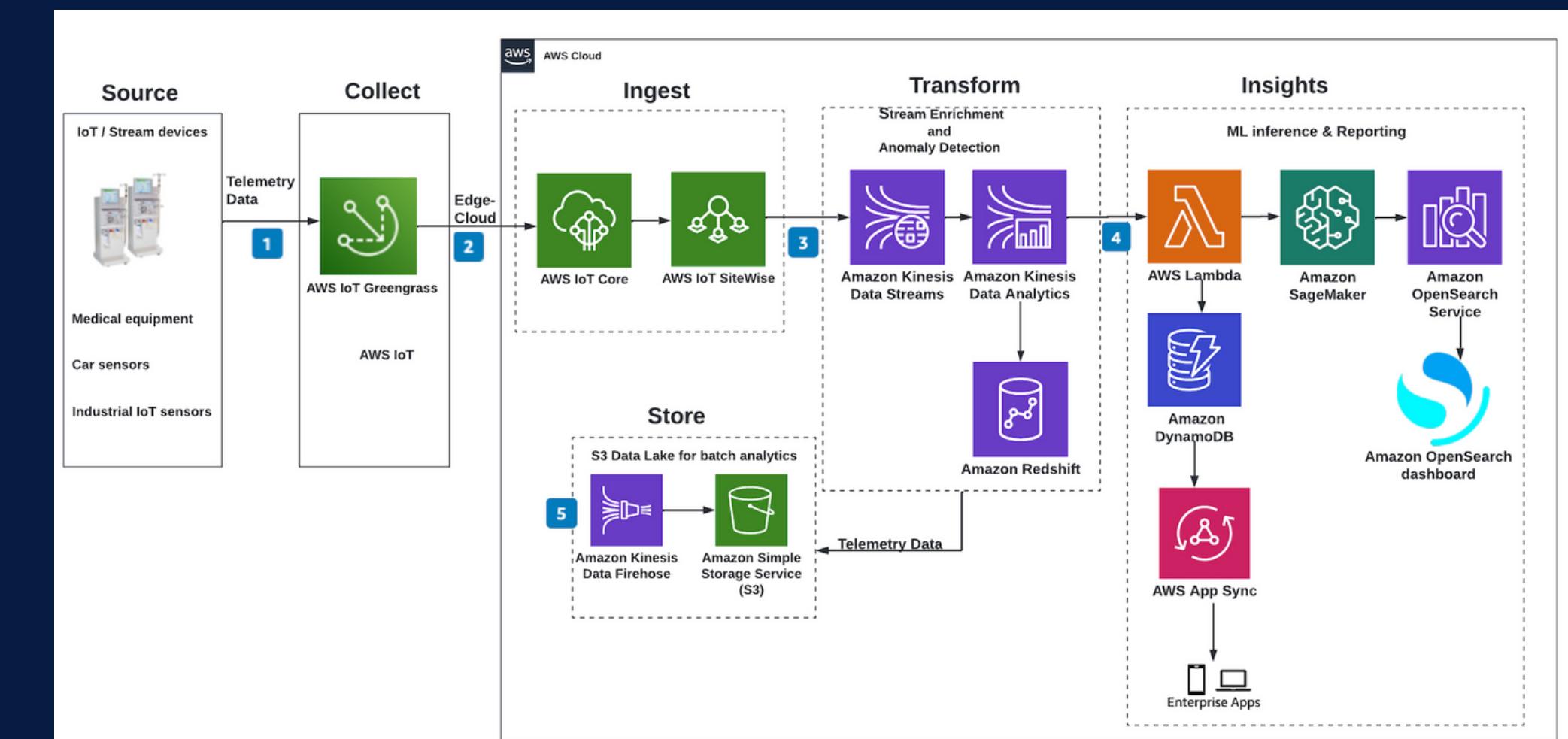
Items returned (3)

Actions Create item

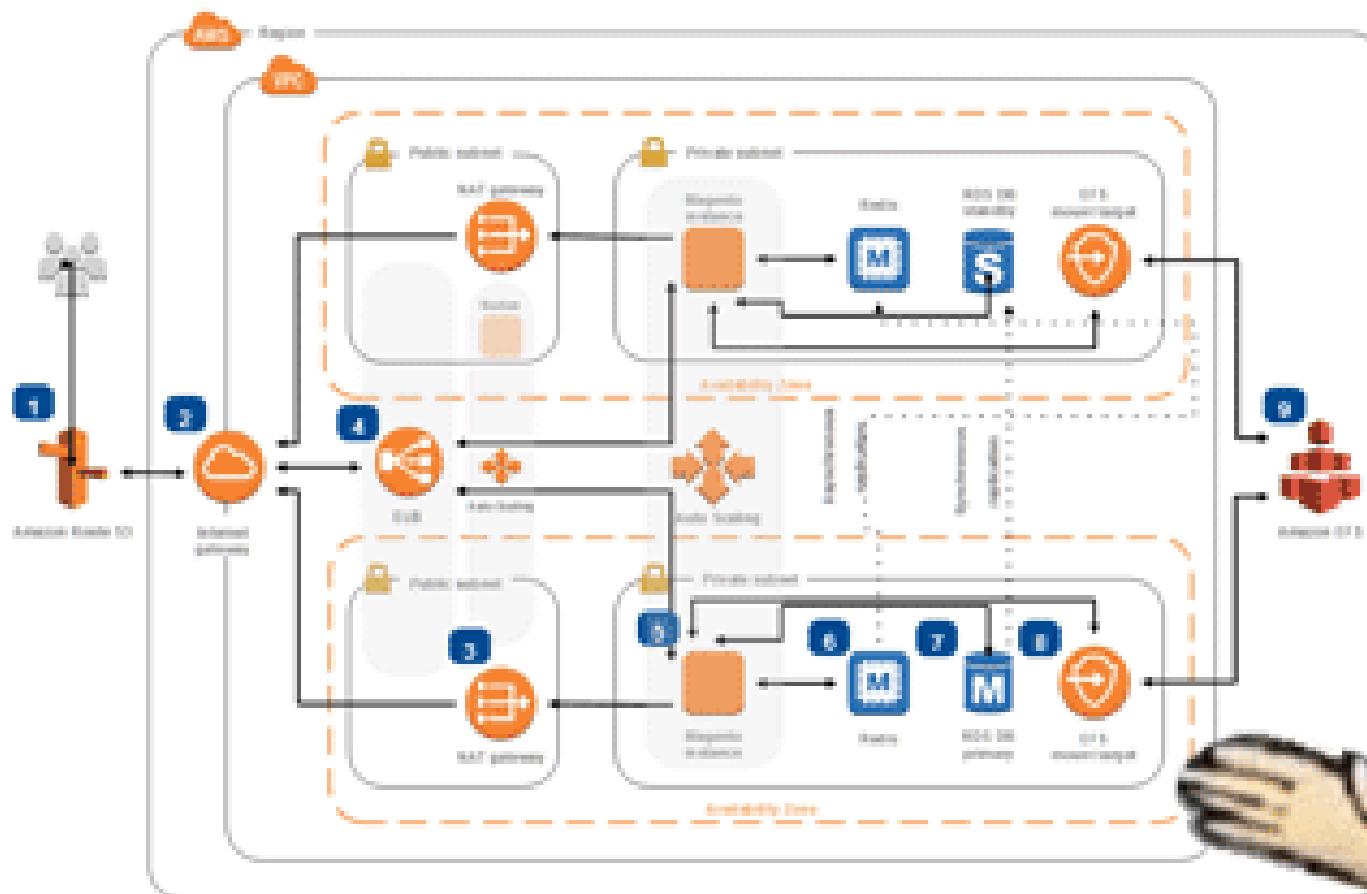
ts (String) payload

1708436040492	{ "temperature": { "N": "29.5" }, "humidity": { "N": "82.4" }, "pressure": { "N": "1013" } }
1708436033110	{ "temperature": { "N": "28.5" }, "humidity": { "N": "82" }, "pressure": { "N": "1011" } }
1708436003898	{ "temperature": { "N": "28" }, "humidity": { "N": "80" }, "pressure": { "N": "1013" } }

Esempio di un'architettura di produzione

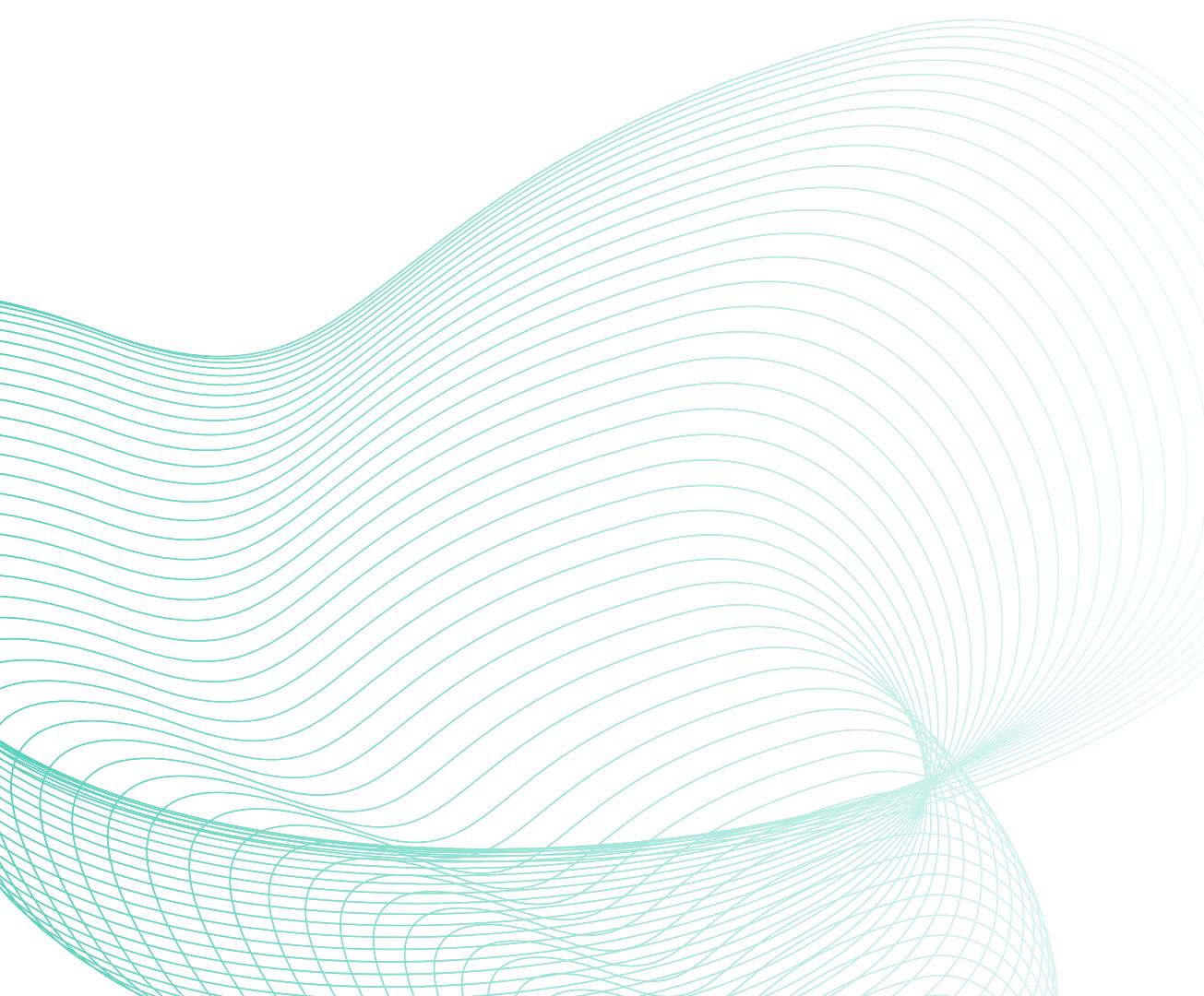


ME: I JUST NEED TO HOST 'HELLO WORLD' ON THE CLOUD.



AWS: NO PROBLEM. HAVE YOU
CHECKED ALL OF OUR COOL NAMED
PRODUCTS YOU'LL NEVER UNDERSTAND?

Esempi di ML su uPy



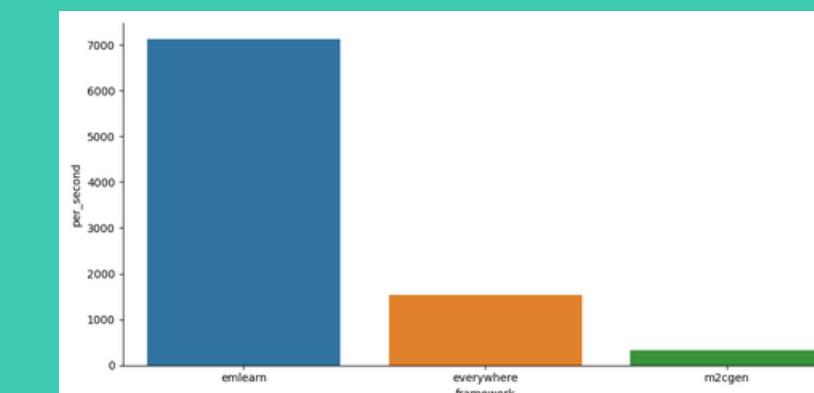
TensorFlow Lite Micro

Esiste un [progetto](#) open source che ha embeddato [TSLite Micro](#) su uPy. Non è purtroppo molto aggiornato e richiede di ricompilare il binario con delle librerie specifiche.

[Esempio](#) di riconoscimento vocale

Emlearn

Riproduzione della libreria Emlearn in micropython.
Facilmente integrabile [scaricando](#) i file ed inserendoli nella cartella lib. non ha tanti modelli, ma sono molto efficienti



M2CGEN

[Progetto](#) open source che tenta di riprodurre dei modelli di [Scikit-learn](#) in [codice nativo](#) di un set di linguaggi: Python, C, Java, Go, JavaScript, Visual Basic, C#, PowerShell, R, PHP, Dart, Haskell, Ruby, F#, Rust, Elixir.

A manina

- Allenare il modello TF su colab
- Esportare pesi e bias da TensorFlow su file
- Sviluppare la rete in MicroPython
- Implementare i layer e le funzioni di attivazione
- Convertire probabilità normalizzandola tra 0 e 1
- Scrivere una funzione che valuti l'accuratezza

[Un esempio](#)

Aliagrid

Marzo 2024

Progetti di gruppo, Aliagrid come Tutor

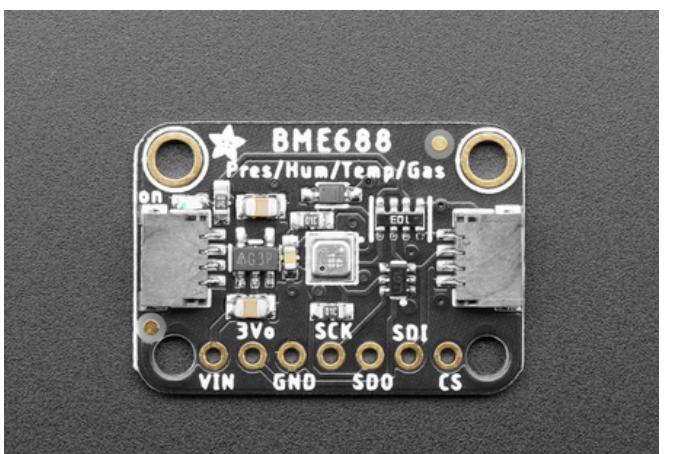
Proposta 1

Descrizione

Implementa un **sistema completo di monitoraggio ambientale** che raccoglie dati come **temperatura, umidità e pressione**. **Analizza le time series** per **identificare anomalie** nei dati, come **variazioni improvvise** o comportamenti non attesi.

Sensori:

Sensori gas: bme688



Proposta 2

Descrizione

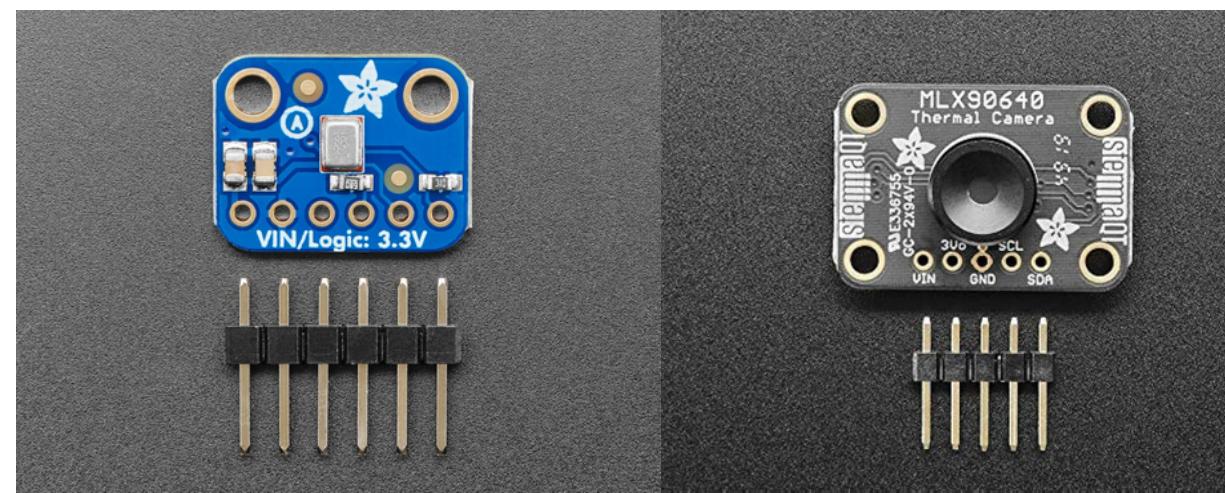
Realizza un **sistema di sicurezza** che utilizza **sensori** di **movimento** per rilevare **intrusioni**. Integra un modello per **classificare** il tipo di **movimento, differenziando** tra **animali domestici e persone**.

Sensori

Videocamera termica: mlx
microfono

Attuatori

Led



Proposta 3

Descrizione

Creare un **sistema di illuminazione intelligente** che si adatta alle condizioni ambientali.

Sensori

Videocamera termica: MLx

Attuatori

Led



Proposta 4

Descrizione

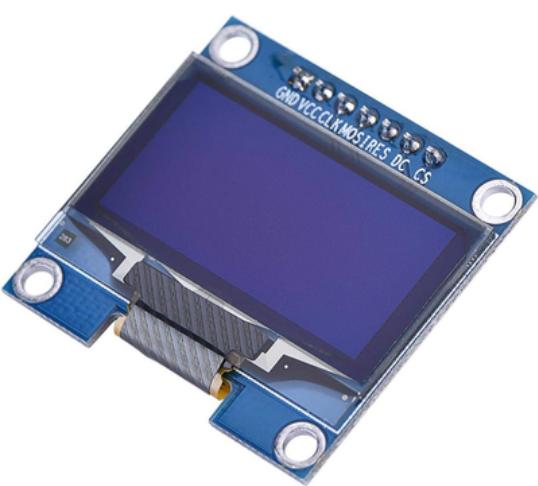
Implementa un **sistema di conteggio delle persone** che utilizza **sensori termici** facendo attenzione a distinguere persone diverse.

Sensori

Videocamera termica: MLx

Attuatori

Display



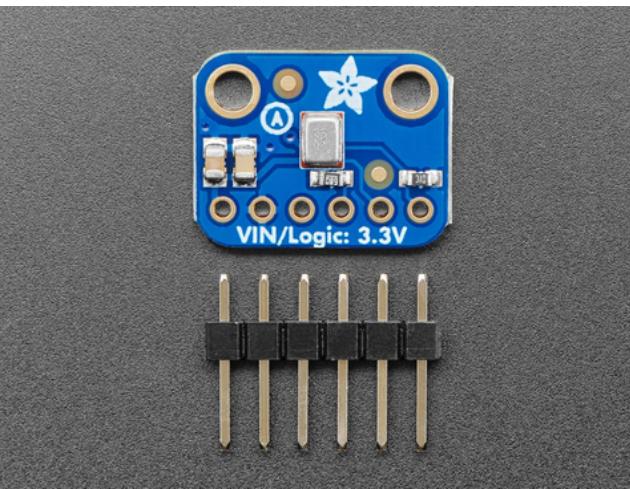
Proposta 5

Descrizione

Implementa un sistema di **rilevamento** dei **suoni anomali** che utilizza **microfoni**. Integra un modello di **analisi** dei **segnali** per **classificare i suoni**: identificando ad esempio **rumori sospetti** o situazioni di emergenza.

Sensori

Microfono



Proposta 6

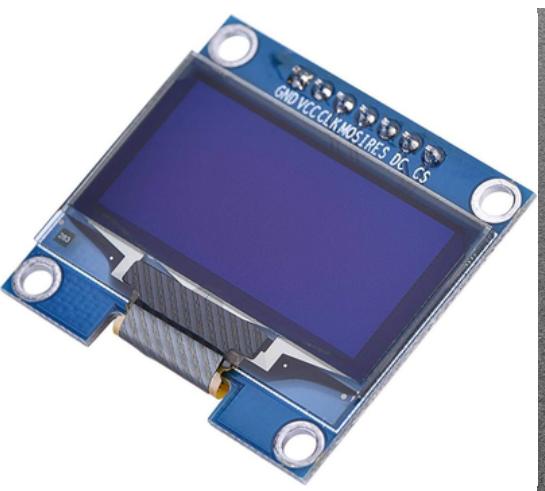
Descrizione

Fai la tua proposta!

Chi meglio di te sa cosa può risultarti interessante da scoprire

Sensori

Tutti quelli che hai visto



Aliagrid

Marzo 2024

Proposte di Tesi



I WANT YOU

TO PULL MY FINGER

"Sistema integrato per il monitoraggio e la predizione della qualità dell'aria urbana"

Descrizione:

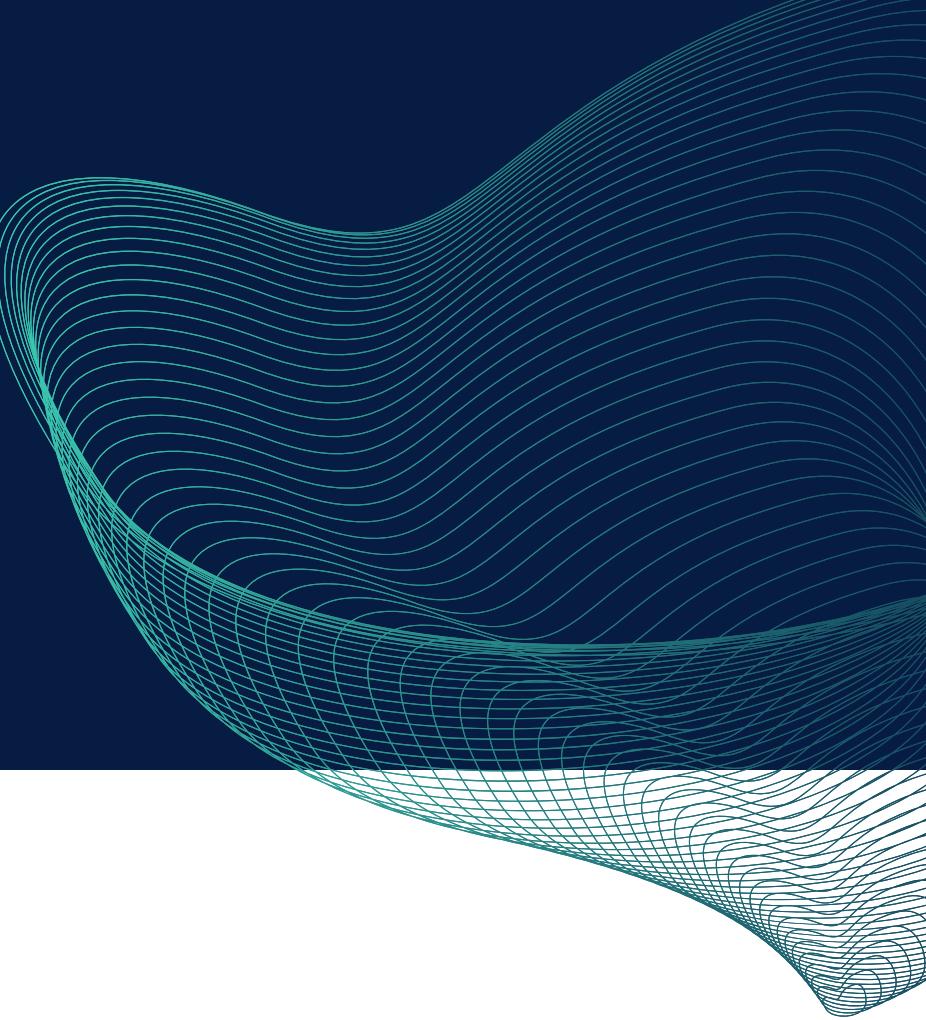
Lo studente dovrà: **progettare e implementare un sistema integrato** basato su microcontroller con MicroPython per il **monitoraggio continuo della qualità dell'aria nelle città**. Utilizzare sensori dedicati per raccogliere **dati ambientali** e implementare modelli di **machine learning** per la **previsione della qualità dell'aria**. Costruire un sistema distribuito che metta in connessione le misure ottenute dai vari sensori per **escludere eventuali outlier** e convergere verso un risultato condiviso.

Obiettivi Specifici:

- Configurare una **rete di sensori** per il monitoraggio della **qualità dell'aria** in diversi punti della città.
- Implementare **modelli di machine learning** per la **previsione** della qualità dell'aria **basati sui dati storici**.
- Sviluppare **un'interfaccia web** per visualizzare in tempo reale i dati raccolti e le previsioni.



"Rilevamento e quantificazione automatica delle portata del gas utilizzando microfoni e machine learning"



Descrizione

Lo studente dovrà: realizzare un sistema per il **monitoraggio** della **portata del gas in tempo reale** utilizzando **microfoni** e **modelli di machine learning**. Sviluppare algoritmi in grado di rilevare e **classificare automaticamente** anomalie nei gruppi di riduzione pressione del gas **basandosi sui suoni generati**. Applicare l'approccio MicroPython per implementare il sistema su microcontroller.

Obiettivi specifici

- **Acquisire e analizzare campioni audio** correlati alle diverse situazioni di passaggio di gas ed estrarne la correlazione con la portata.
- **Implementare algoritmi di machine learning** per la classificazione delle **anomalie sonore**.
- Integrare il **sistema** con **avvisi** e notifiche in tempo reale **per situazioni di emergenza**.



"Sistema Distribuito per il rilevamento precoce di terremoti utilizzando MicroPython e IMU"

Descrizione

Lo studente dovrà: progettare un **sistema distribuito** di **rilevamento precoce di terremoti** basato su microcontroller con MicroPython. Utilizzare una combinazione di sensori **sismici, microfoni e videocamere** per raccogliere dati e implementare **algoritmi di machine learning** per la **classificazione** delle anomalie. Esplorare la possibilità di inviare notifiche immediate.

Obiettivi specifici

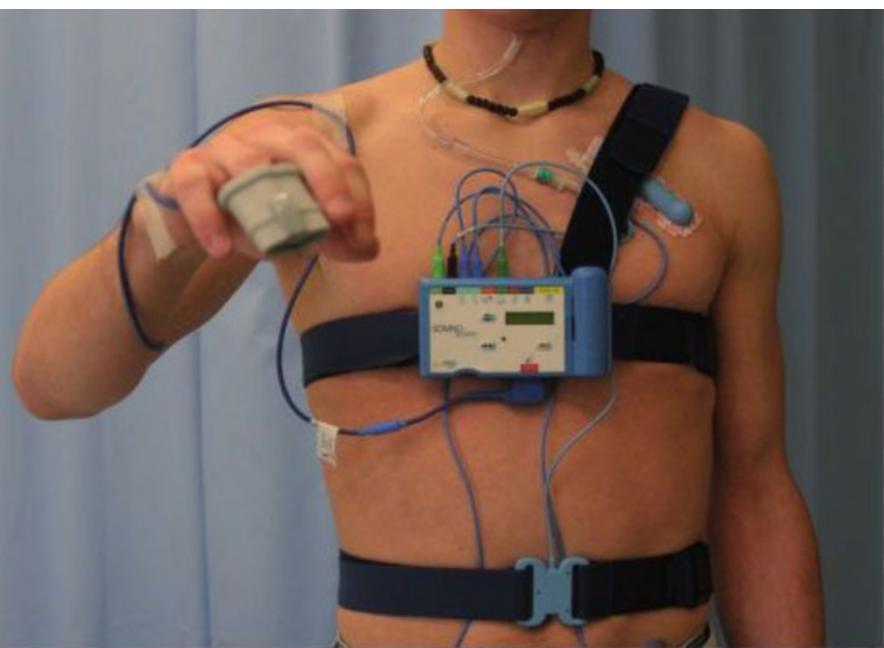
- **Integrazione** di **sensori** sismici, microfoni e videocamere per il monitoraggio completo.
- Sviluppo di **algoritmi** di machine learning per identificare e **classificare** le **anomalie sismiche**.
- Configurazione di un sistema di **notifica tempestiva** in caso di rilevamento di terremoti o eventi sismici significativi.



"Sistema di Sensori Inerziali per Monitoraggio Respiratorio"

Descrizione

Implementare un **dispositivo indossabile** basato su un set di **sensori inerziali**, come **accelerometri e giroscopi**. Questi sensori catturano dati relativi ai **movimenti del torace e dell'addome** durante la respirazione. Utilizzando algoritmi di elaborazione del segnale, si può estrarre la **frequenza respiratoria**, la profondità e altri parametri chiave. Il sistema dovrebbe essere in grado di **riconoscere pattern respiratori anomali**, quali **apnee o irregolarità** nella **frequenza**, e generare notifiche in tempo reale. L'architettura potrebbe integrare un microcontroller con **capacità di calcolo locale** per ridurre la dipendenza dal cloud, garantendo tempi di risposta più veloci.

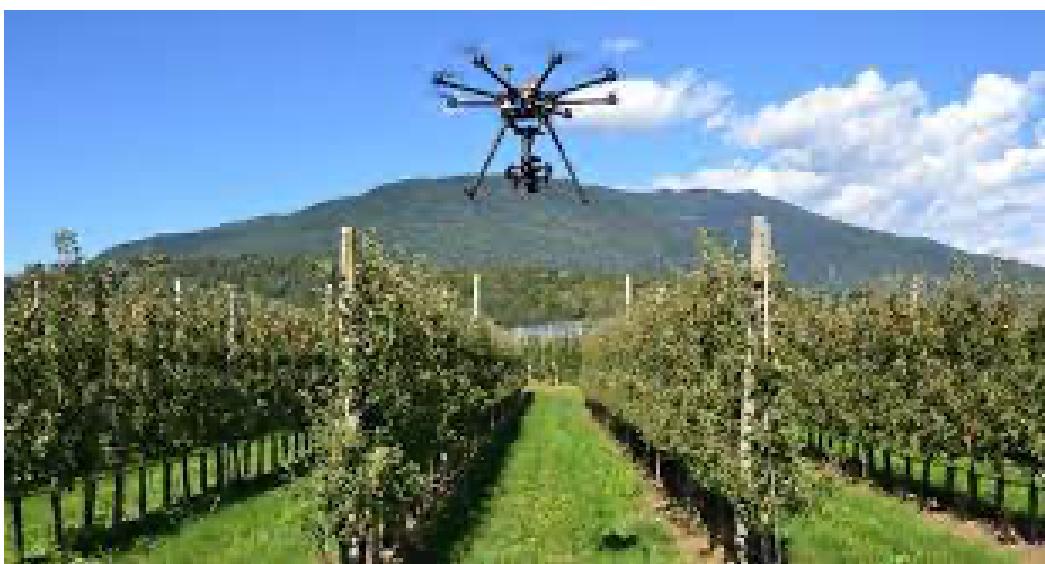


"Algoritmi di Visione Artificiale su Droni per l'Analisi delle Colture"

Descrizione

Creare algoritmi di **visione artificiale** per l'analisi delle immagini raccolte da **droni agricoli**. Utilizzare **reti neurali convoluzionali** (CNN) e tecniche di trasferimento di apprendimento per riconoscere dettagli importanti come lo **stato della vegetazione**, la **presenza di malattie** e **carenze nutrizionali** nelle **colture**. Integrare i dati raccolti con sensori in situ per una valutazione più completa.

L'implementazione richiede la **configurazione accurata** dei **parametri** delle **reti neurali**, la **gestione efficiente** delle **risorse computazionali** a **bordo** del drone e la **trasmissione** dati sicura e **affidabile** alla **stazione di controllo** a terra.



"Sistema Domotico basato su Microservizi e Protocollo Matter"

Descrizione

Sviluppare un **sistema domotico distribuito** basato su **microservizi** e utilizzando il **protocollo Matter** (precedentemente conosciuto come Project Connected Home over IP, o Project CHIP) per la comunicazione tra i dispositivi. **Ogni dispositivo** dovrebbe essere **implementato** come un **microservizio autonomo**, garantendo una **gestione modulare** delle **funzionalità**.

Il protocollo **Matter offre un'interoperabilità standardizzata** tra una varietà di dispositivi smart home, fornendo una **base comune** per l'**interazione** e la comunicazione affidabile. L'utilizzo di Matter assicura una connessione stabile e sicura tra i dispositivi, semplificando l'integrazione di nuovi dispositivi conformi a questo standard nell'ecosistema domotico come **Home Assistant**.

