

# Aliagrid

Febbraio 2025

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



**Mael Vittorio Vena**

CTO & Co-Founder @ **Aliagrid**

2-time founder, esperienza  
pluriennale nel settore informatico,  
IoT, AI e delle startup.

Techstars 23

**B4i** BOCCONI FOR  
INNOVATION



UNIVERSITÀ  
DEGLI STUDI  
DI MILANO

**techstars\_**



**Prysmian**  
Group

A collage of three images: a colorful digital neural network or cloud visualization, a server room with rows of server racks, and a close-up of a green printed circuit board with a black integrated circuit chip.

## Mi occupo di:

- FW
- ML – Edge Computing
- IoT
- Cloud
- Networking
- HPC



**<https://bit.ly/workshop-micropython>**

# IoT

Dispositivi connessi in rete

Per **acquisire** dati

Per **intraprendere** azioni



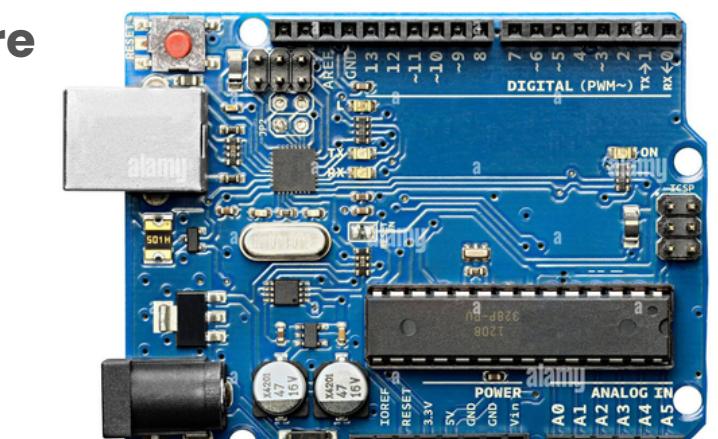
# Microcontrollori vs Single-Board Computer

## Microcontrollore

Microcontrollore:

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

**Board:** la scheda su cui alloggia il **microcontrollore**



## Single-Board Computers

Hanno **tutti gli elementi** di un **computer**, in una singola board

Es: Raspberry PI

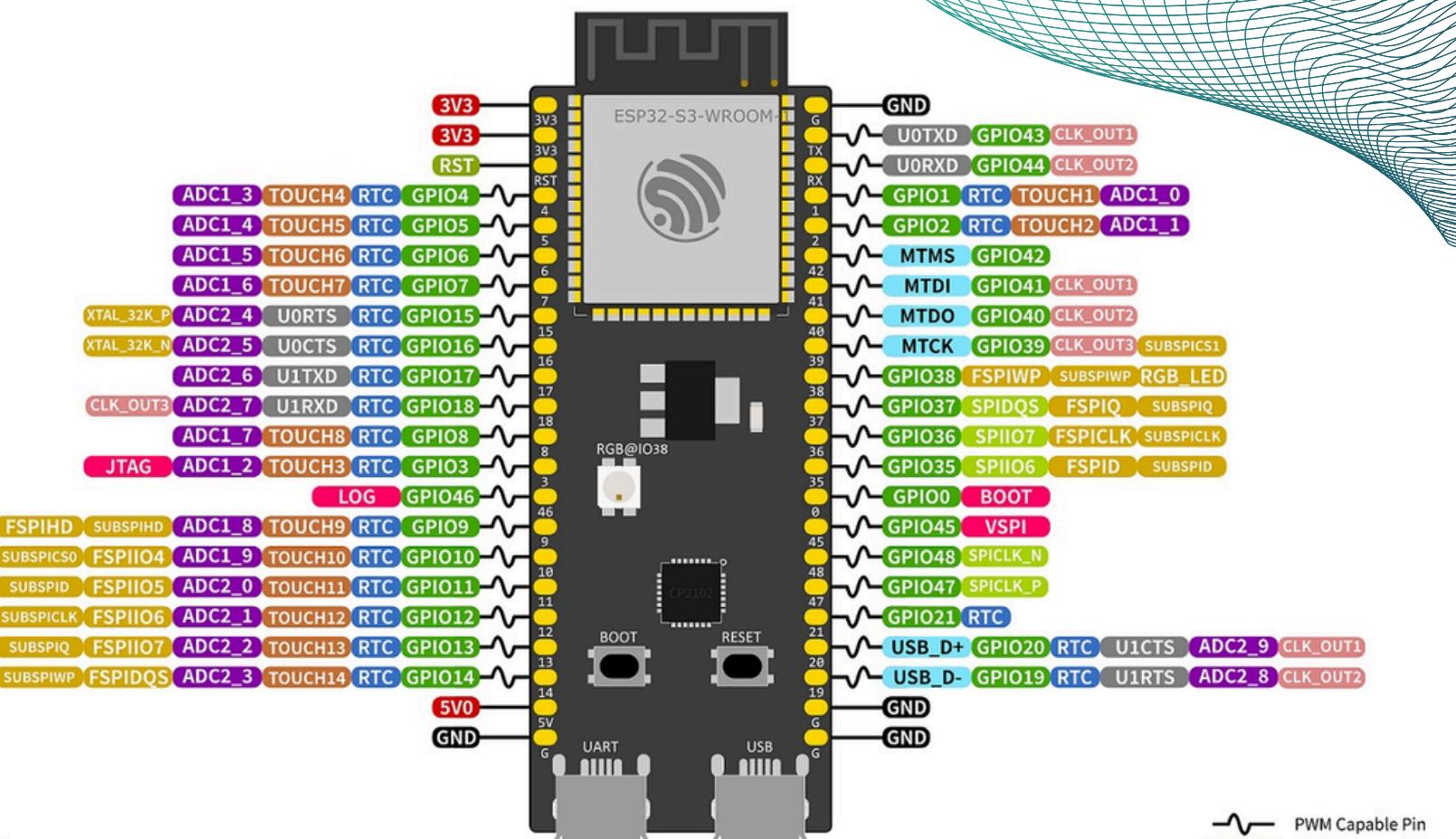


# Esp32

**“Pinout”: schema dei pin  
con nome e finalità**

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

ESP32-S3-DevKitC-1

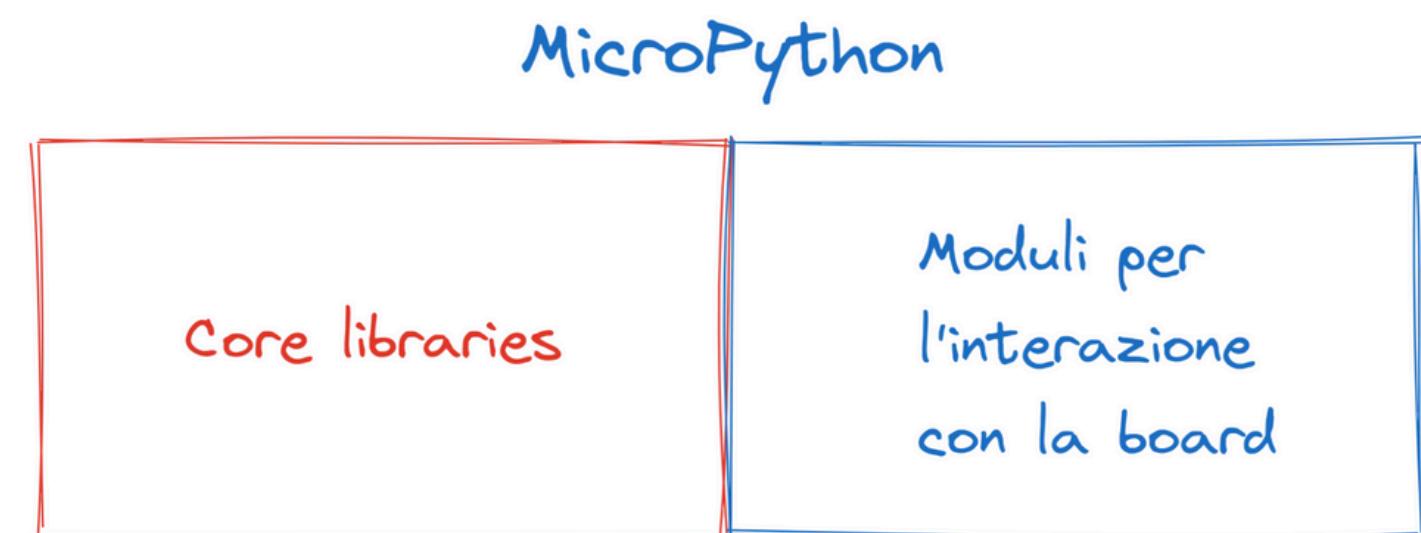


# Cos'è Micropython?

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

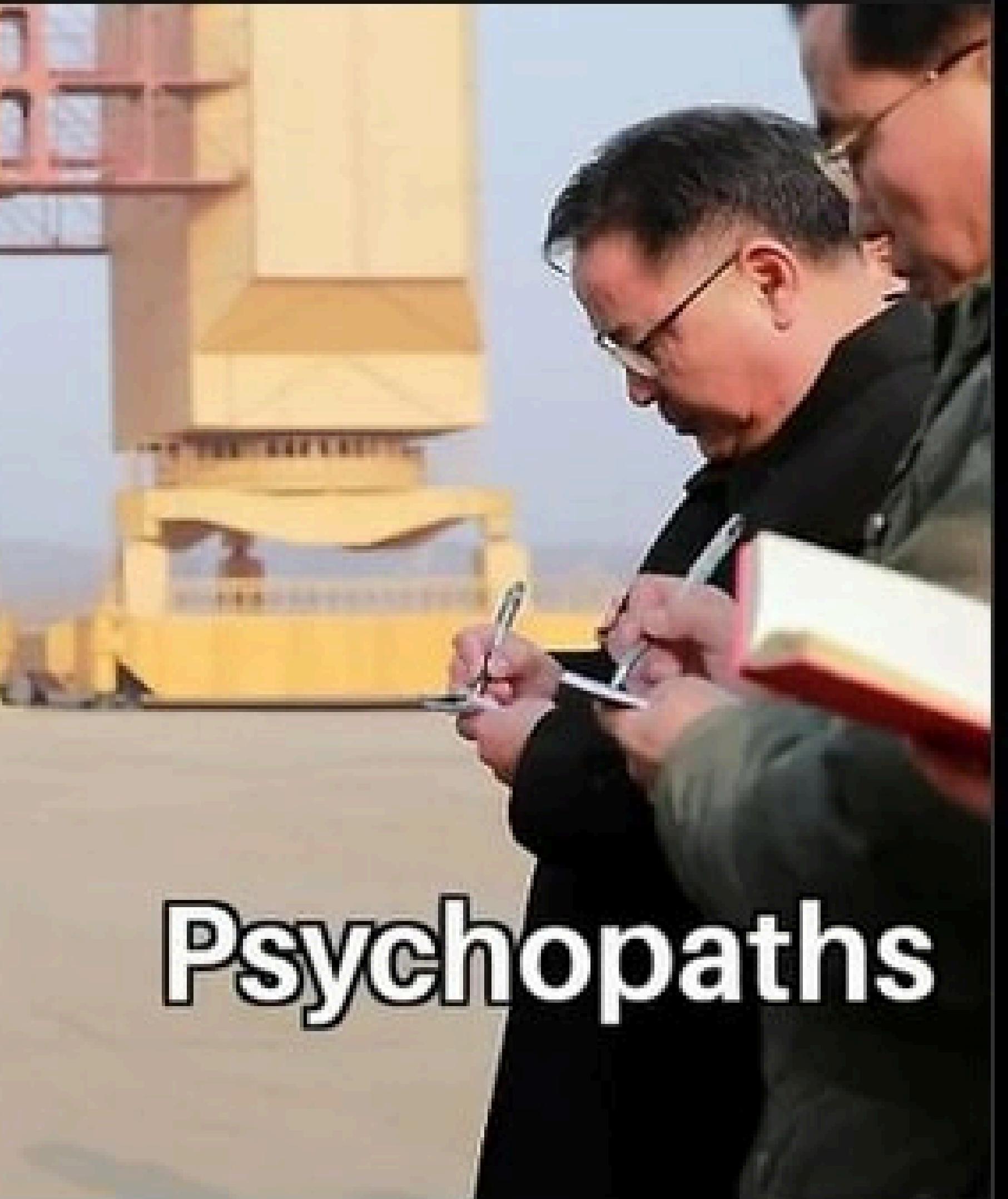
- entra in **256k** di **codice** e **16kb** di **RAM**
- vuole essere più compatibile possibile con Python
- Permette di **interagire** con le **componenti** della **board**

Nota: non tutte le board lo supportano!





People who program  
embedded systems  
with Python



Psychopaths

# Perchè Python per micro?

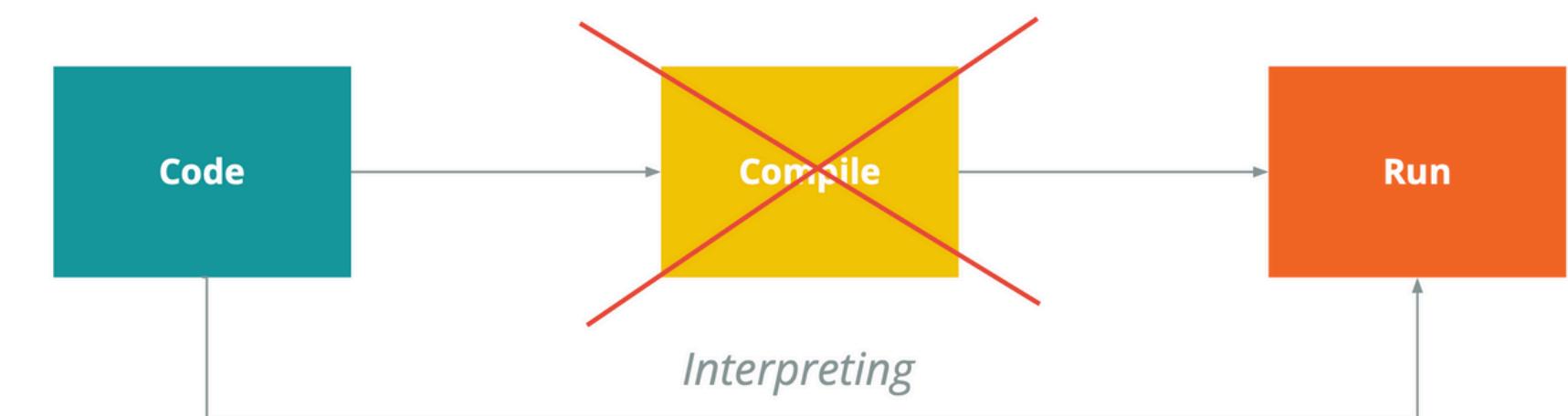
- Facile da scrivere e testare rispetto ai linguaggi C/C++.
- Consente una prototipazione rapida con meno codice boilerplate.
- La leggibilità e la sintassi lo rendono una scelta eccellente per i principianti.

<https://intechhouse.com/blog/top-programming-languages-for-embedded-systems/>



# Perchè è potente?

- Facile da leggere
  - Progettato con la semplicità in mente
- Flessibilità multipiattaforma
  - Funziona su diverse piattaforme hardware
- Ciclo di sviluppo più rapido
  - Nessun lungo processo di compilazione e flashing
- Comunità e risorse
  - Linguaggio di programmazione ampiamente utilizzato con una community attiva
- Ampio accesso alle librerie Python
  - Disponibilità di numerose librerie Python



# Python essentials

- Sintassi semplice (niente ; o {})
- Tipizzazione dinamica (nessuna dichiarazione di tipo)
- Gestione automatica della memoria (nessuna malloc o free)
- Indentazione obbligatoria (invece di {})

```
1  # Nessuna necessità di punto e virgola o parentesi graffe 🌟
2
3  def greet(name): # Definizione di funzione con indentazione
4      message = f"Hello, {name}!" # Tipizzazione dinamica: non serve dichiarare il tipo
5      return message # La memoria viene gestita automaticamente
6
7  # Liste dinamiche e loop leggibili
8  names = ["Alice", "Bob", "Charlie"]
9  for name in names:
10     print(greet(name)) # Output leggibile con f-string
11
12 # Creazione di una classe con costruttore senza dichiarazione esplicita dei tipi
13 class Animal:
14     def __init__(self, name):
15         self.name = name # Assegnazione diretta, senza specificare il tipo
16
17     def speak(self):
18         return f"{self.name} makes a sound."
19
20 dog = Animal("Doggo") # Nessuna necessità di allocazione esplicita della memoria
21 print(dog.speak())
22
23 # Esempio di lambda function (funzione anonima)
24 square = lambda x: x ** 2
25 print(square(5)) # Output: 25
```

# uPy essentials

- ✓ Sintassi pulita (niente {} o ;)
- ✓ Tipizzazione dinamica (nessuna dichiarazione esplicita dei tipi)
- ✓ Gestione della memoria automatica (nessun malloc o free)
- ✓ Indentazione obbligatoria (Pythonic!)
- ❤ Supporto per hardware embedded (machine.Pin, utime.sleep)

```
1 import machine
2 import utime # MicroPython usa utime invece di time
3
4 # Funzione con sintassi semplice e indentazione obbligatoria
5 def blink_led(pin_number):
6     led = machine.Pin(pin_number, machine.Pin.OUT) # Nessuna dichiarazione di tipo
7     while True:
8         led.value(1) # Accende il LED
9         utime.sleep(0.5) # Aspetta 500ms
10        led.value(0) # Spegne il LED
11        utime.sleep(0.5)
12
13 # Oggetto senza dichiarazione esplicita dei tipi
14 class Sensor:
15     def __init__(self, pin_number):
16         self.pin = machine.Pin(pin_number, machine.Pin.IN)
17
18     def read_value(self):
19         return self.pin.value() # Ritorna dinamicamente 0 o 1
20
21 # Esempio di uso di una lambda function
22 square = lambda x: x ** 2
23
24 # Esempio di esecuzione
25 print("MicroPython Ready 🚀")
26 sensor = Sensor(2) # Creazione dell'oggetto senza allocazione esplicita della memoria
27 print(f"Sensor reading: {sensor.read_value()}")
28
29 # Per testare il LED (rimuovere il commento se si vuole eseguire)
30 # blink_led(2) # Funzione bloccante
```

**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.

# Custom Bin

- 💡 Perché un binario custom?
  - **Includere librerie Python** direttamente nel firmware
  - **Aggiungere moduli C** per migliori prestazioni
  - **Ottimizzare memoria** e funzionalità per il proprio hardware

## 1 Aggiungere librerie Python

Copiare gli script Python nella cartella `modules/` della build  Ricompilare MicroPython con `make`

```
mkdir -p ports/esp32/modules  
cp my_library.py ports/esp32/modules/  
make BOARD=ESP32
```

## 2 Integrare moduli C personalizzati

Scrivere un modulo C (`my_module.c`):

```
#include "py/runtime.h"  
STATIC mp_obj_t my_function() {  
    return mp_obj_new_int(42);  
}  
STATIC MP_DEFINE_CONST_OBJ_0(my_function_obj, my_function);  
STATIC const mp_map_elem_t my_module_globals_table[] = {  
    { MP_QSTR_my_function, MP_OBJSIZEOF(mp_obj_new_int), MP_OBJ_FROM_PTR(&my_function_obj) },  
};  
STATIC MP_DEFINE_CONST_DICT(my_module_globals, my_module_globals_table);  
const mp_obj_module_t my_module = {  
    .base = { &mp_type_module },  
    .globals = (mp_obj_dict_t *)&my_module_globals,  
};  
MP_REGISTER_MODULE(MP_QSTR_my_module, my_module);
```

Aggiungerlo a `mpconfigport.h`:

```
#define MICROPY_PY_MY_MODULE (1)
```

Modificare il `CMakeLists.txt` o `Makefile` per includere il modulo

# OTA

- ◆ Cos'è?
  - Aggiorna gli script di un dispositivo MicroPython da remoto
  - Ideale per dispositivi IoT senza accesso fisico
- ◆ Come funziona?
  - Connessione Wi-Fi
  - Download degli script aggiornati ↓
  - Sostituzione dei file esistenti
  - Riavvio automatico
- ◆ Vantaggi
  - Nessuna connessione fisica necessaria
  - Aggiornamenti rapidi e automatizzati
  - Perfetto per dispositivi remoti

```
# URL del server OTA (modifica con il tuo server)
OTA_SERVER = "http://tuo_server.com/micropython_updates/"

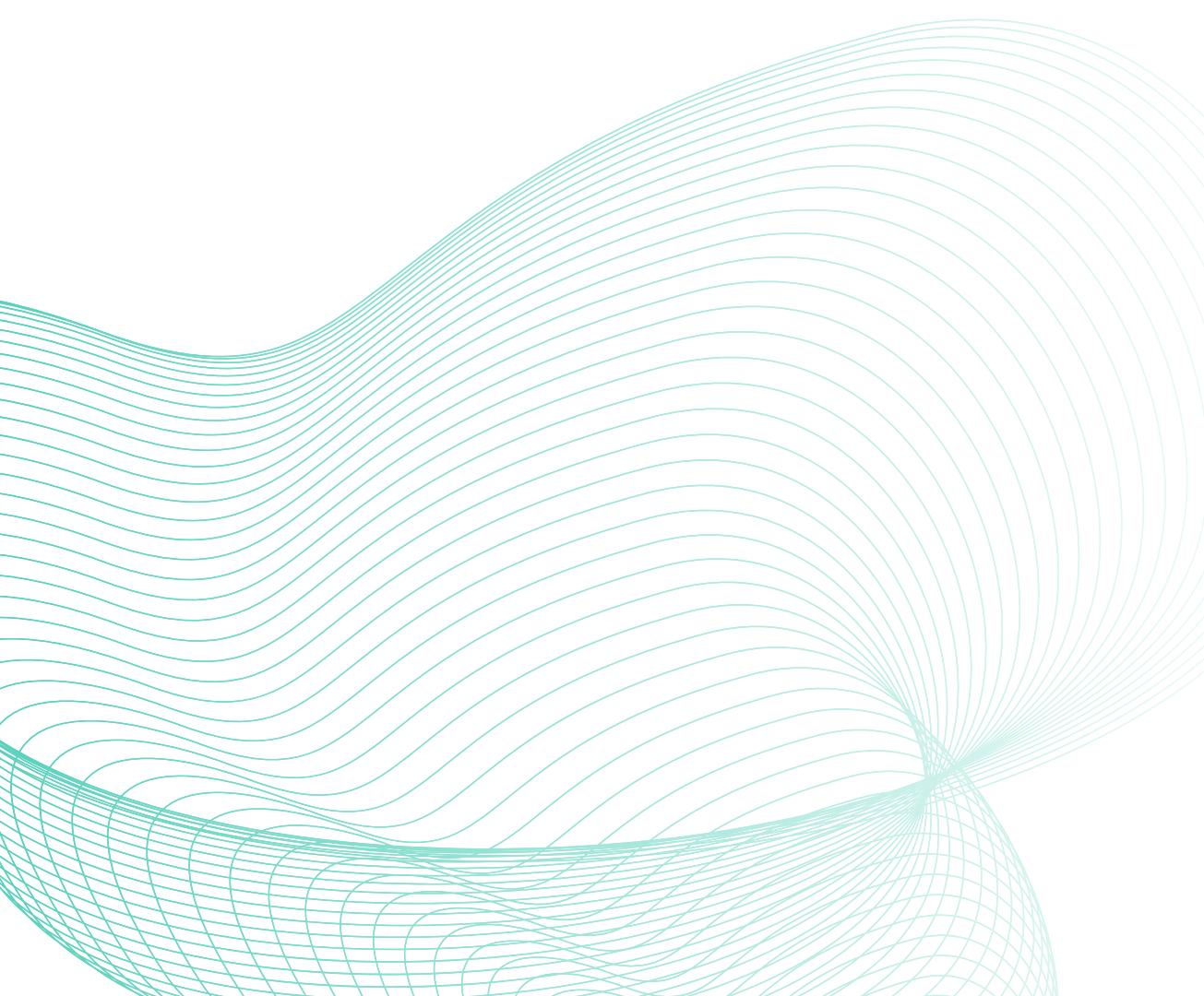
# Lista dei file da aggiornare
FILES_TO_UPDATE = ["main.py", "config.py"]

def download_file(filename):
    """ Scarica un file dal server OTA """
    url = OTA_SERVER + filename
    print(f"Scaricando {filename} da {url}...")
    try:
        response = urequests.get(url)
        if response.status_code == 200:
            with open(filename, "wb") as f:
                f.write(response.content)
            print(f"{filename} aggiornato con successo!")
        else:
            print(f"Errore nel download di {filename}: {response.status_code}")
        response.close()
    except Exception as e:
        print(f"Errore durante il download di {filename}: {e}")

def update_firmware():
    """ Aggiorna tutti i file richiesti """
    for file in FILES_TO_UPDATE:
        download_file(file)

    print("Aggiornamento completato! Riavvio in corso...")
    utime.sleep(2)
    machine.reset() # Riavvia il microcontrollore
```

# Esempi di ML su uPy



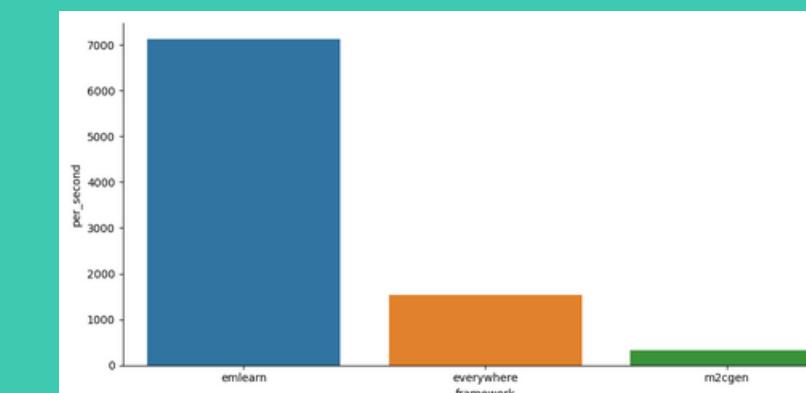
## TensorFlow Lite Micro

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

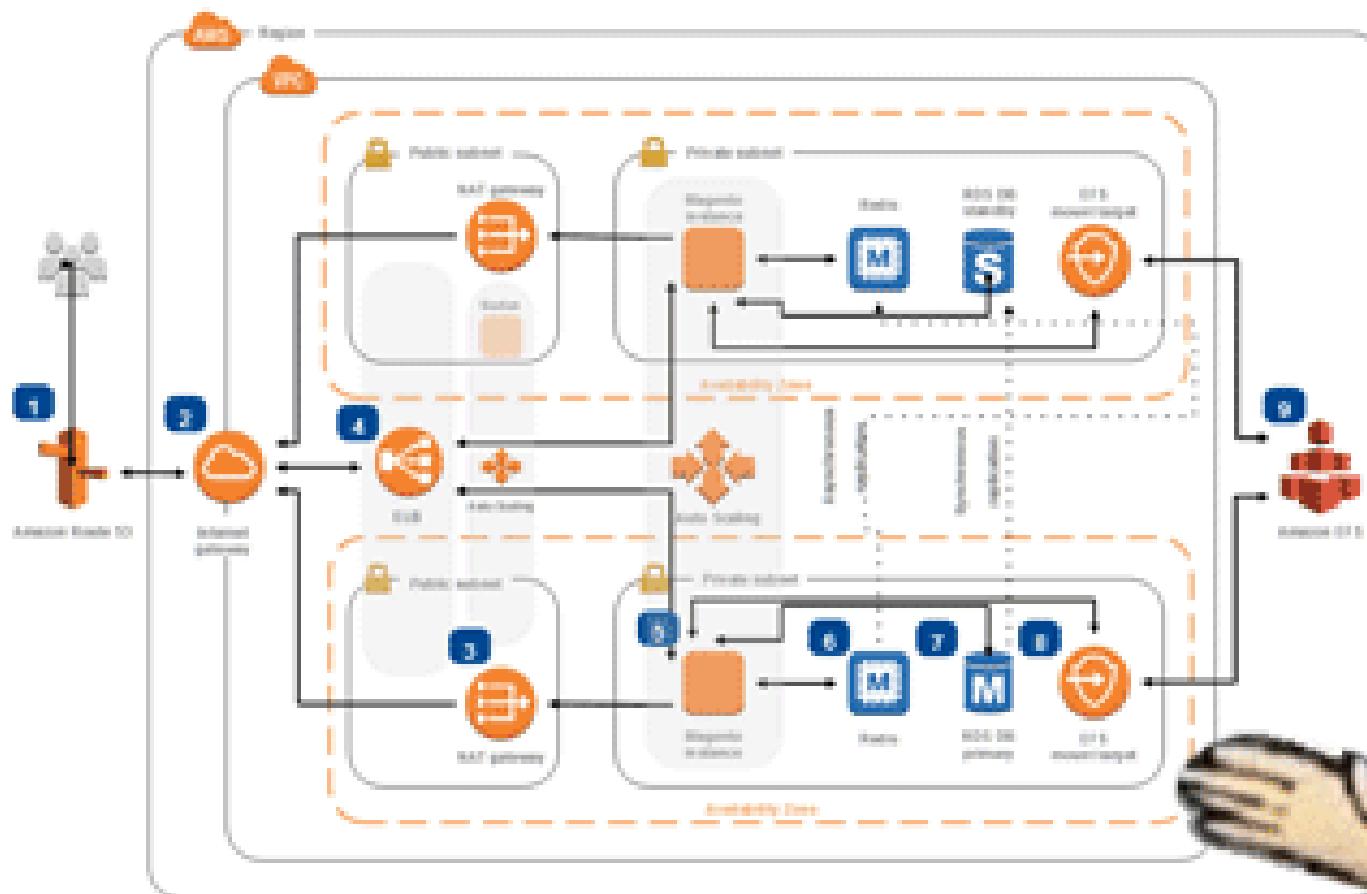
[Progetto](#) open source che tenta di riprodurre dei modelli di [Scikit-learn](#) in [codice nativo](#) di un set di linguaggi: Python, C, Java, Go, JavaScript, Visual Basic, C#, PowerShell, R, PHP, Dart, Haskell, Ruby, F#, Rust, Elixir.

## 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](#)

# 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?

# 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**:

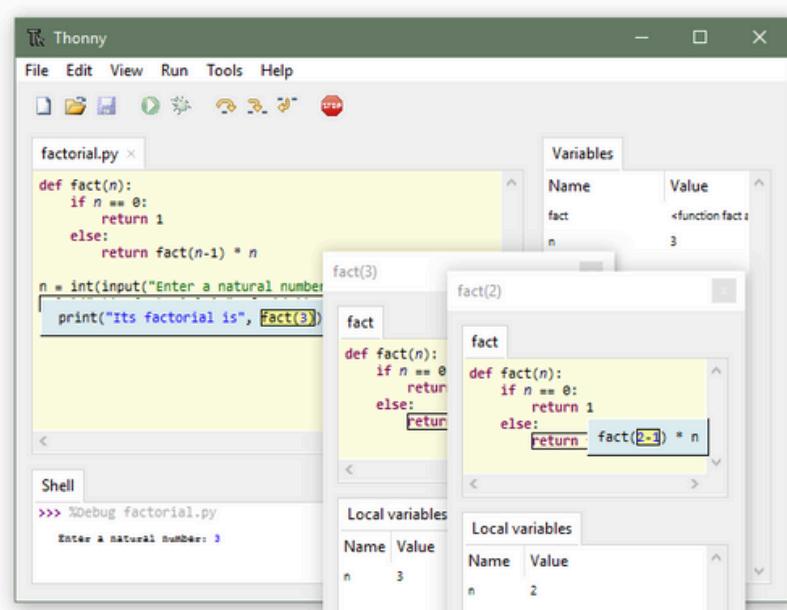
```
pip install esptool
```

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

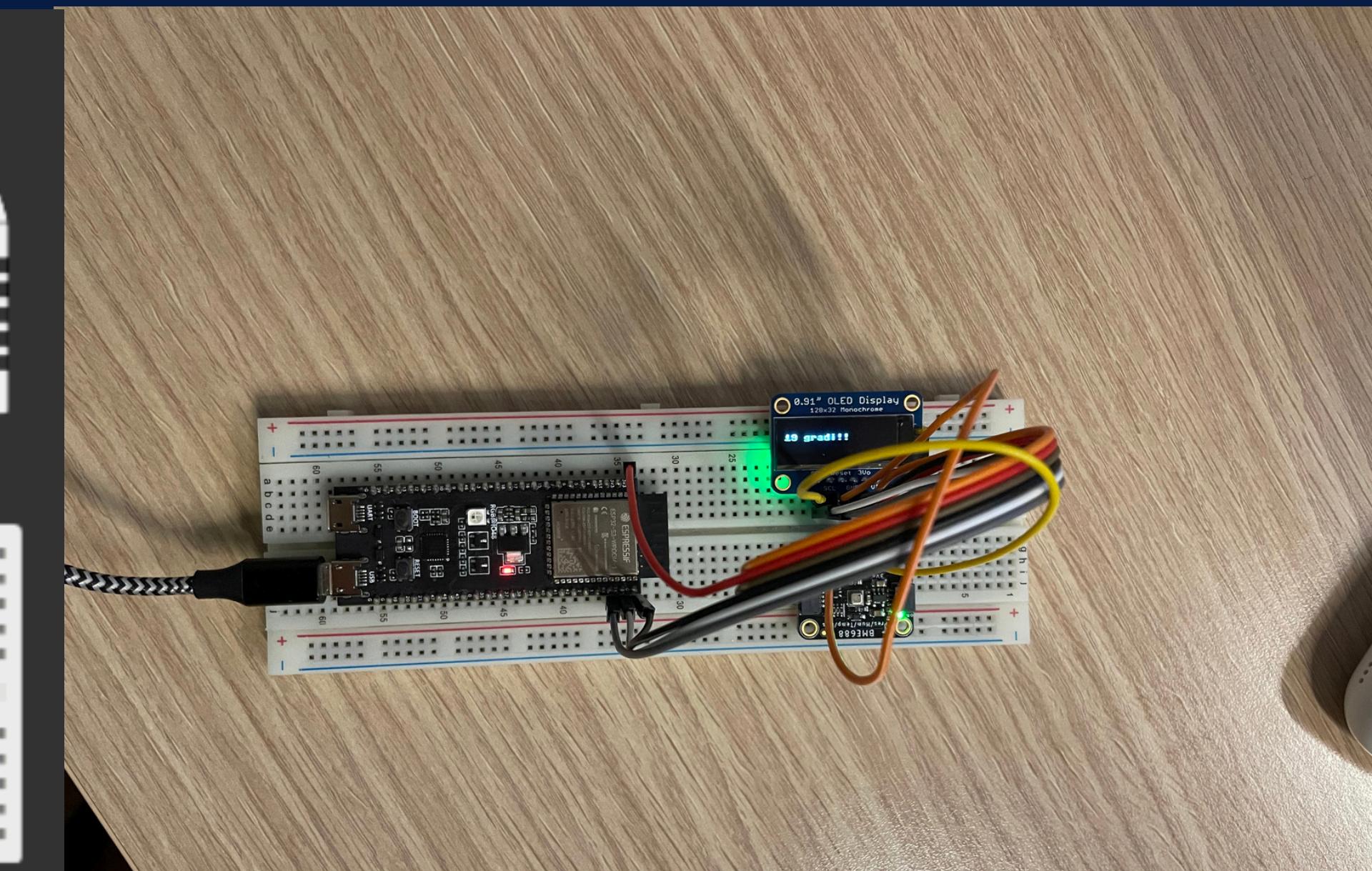
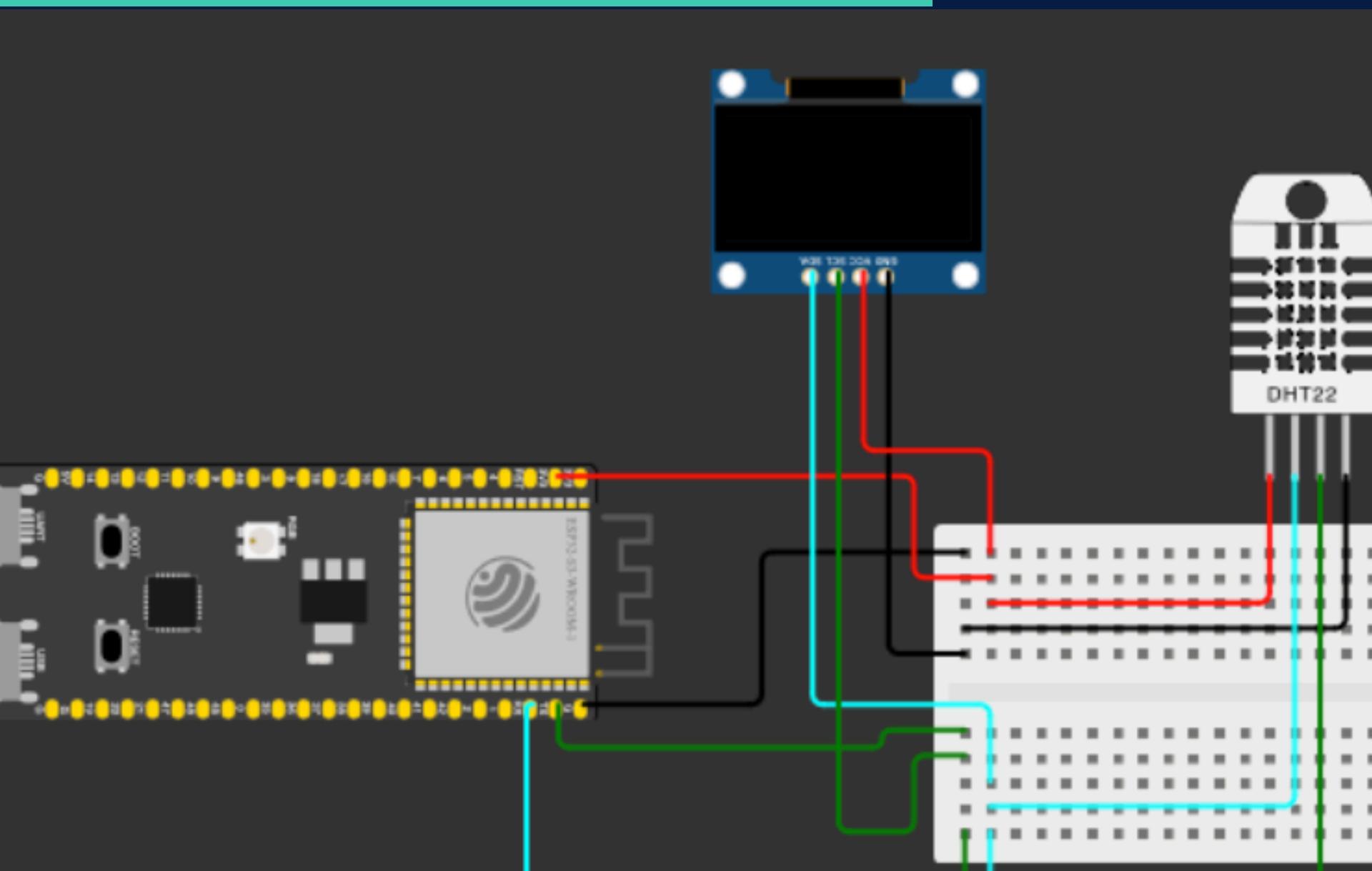
(Specificate la porta)

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.



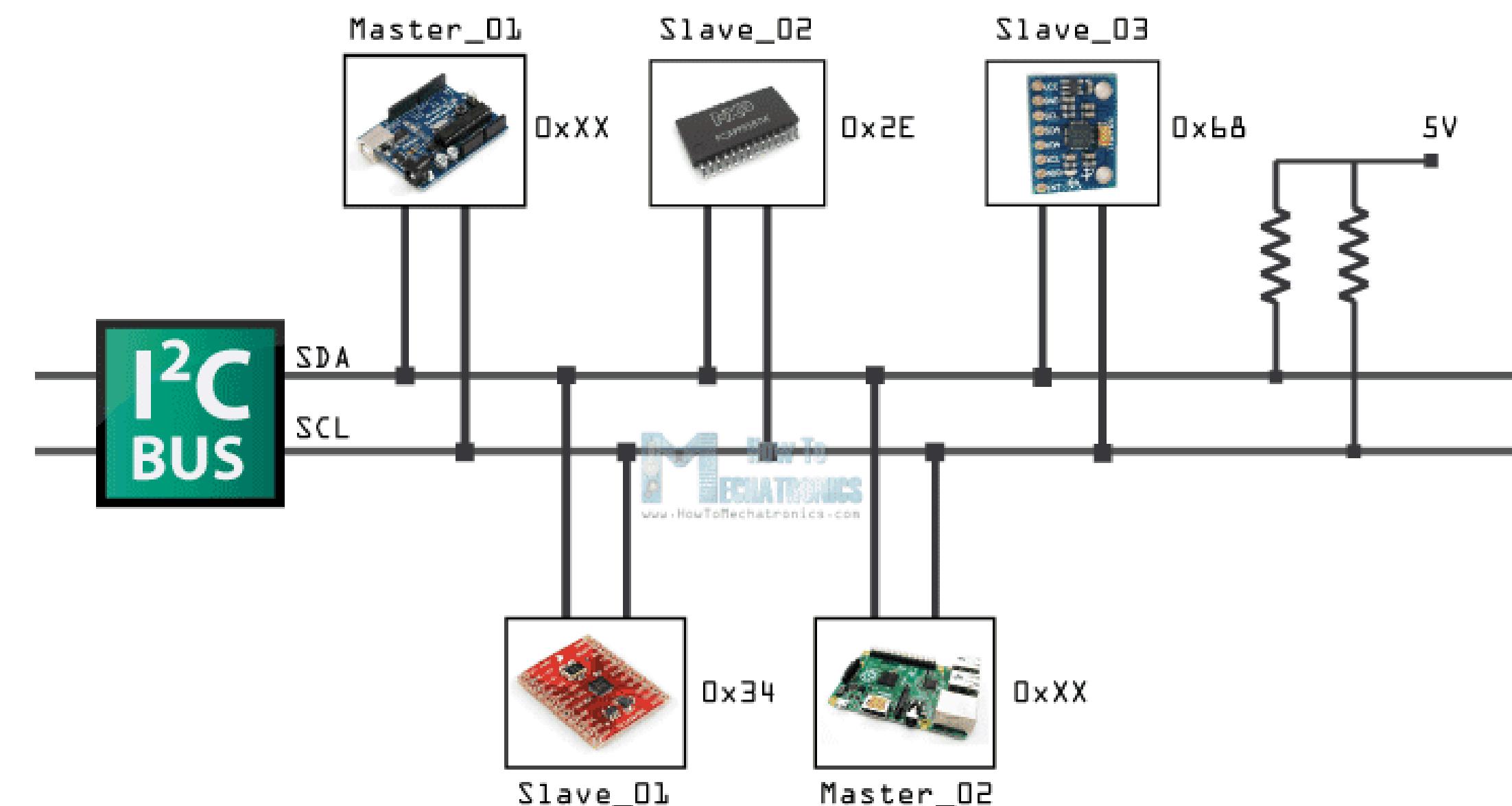
# Collegamento su una breadboard vera



# I<sup>2</sup>C

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

Con **I<sup>2</sup>C**, un sistema di **comunicazione seriale** bifilare utilizzato tra circuiti integrati.

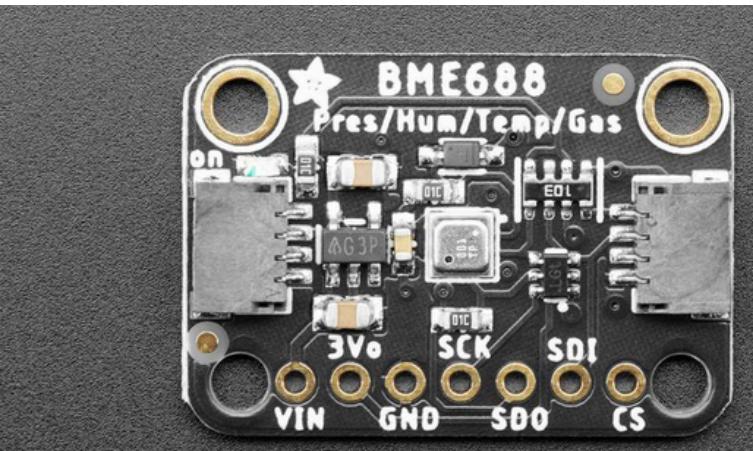


# Termometro I2C

## 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**

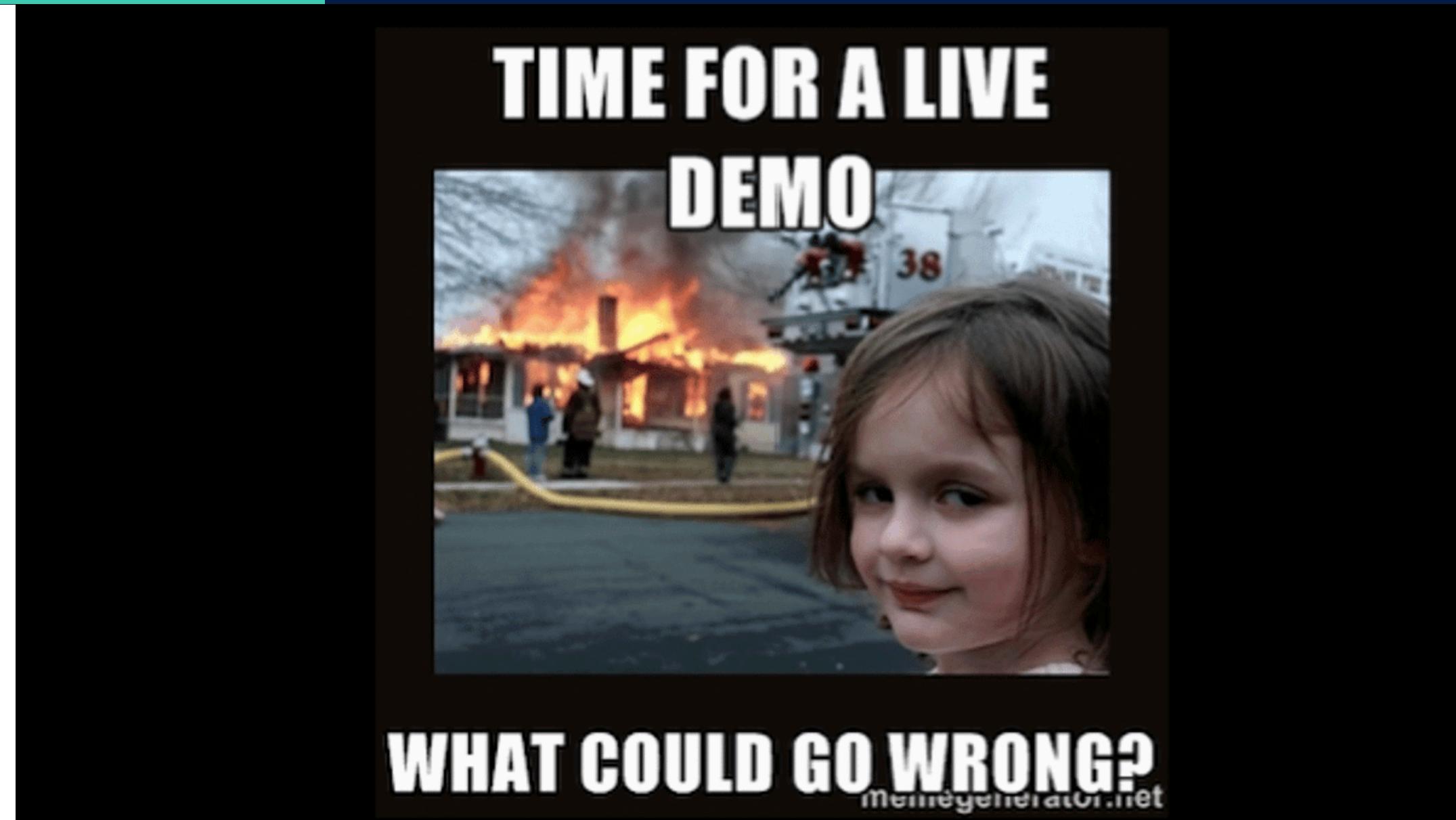
## BME688 Demo

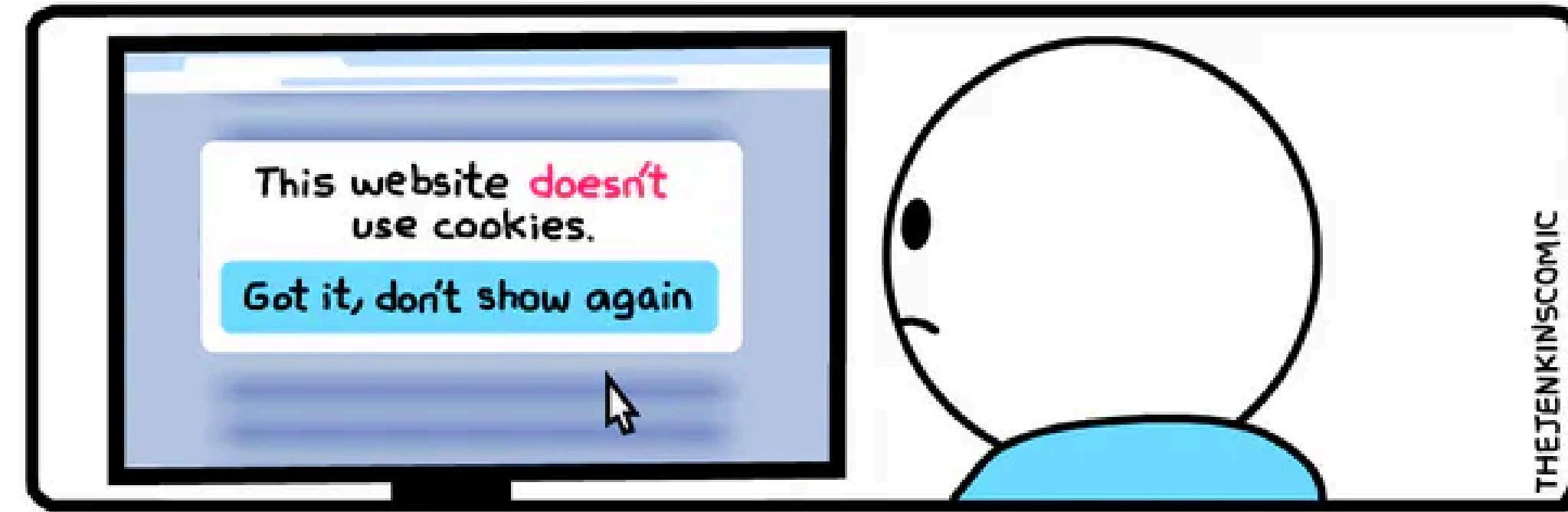
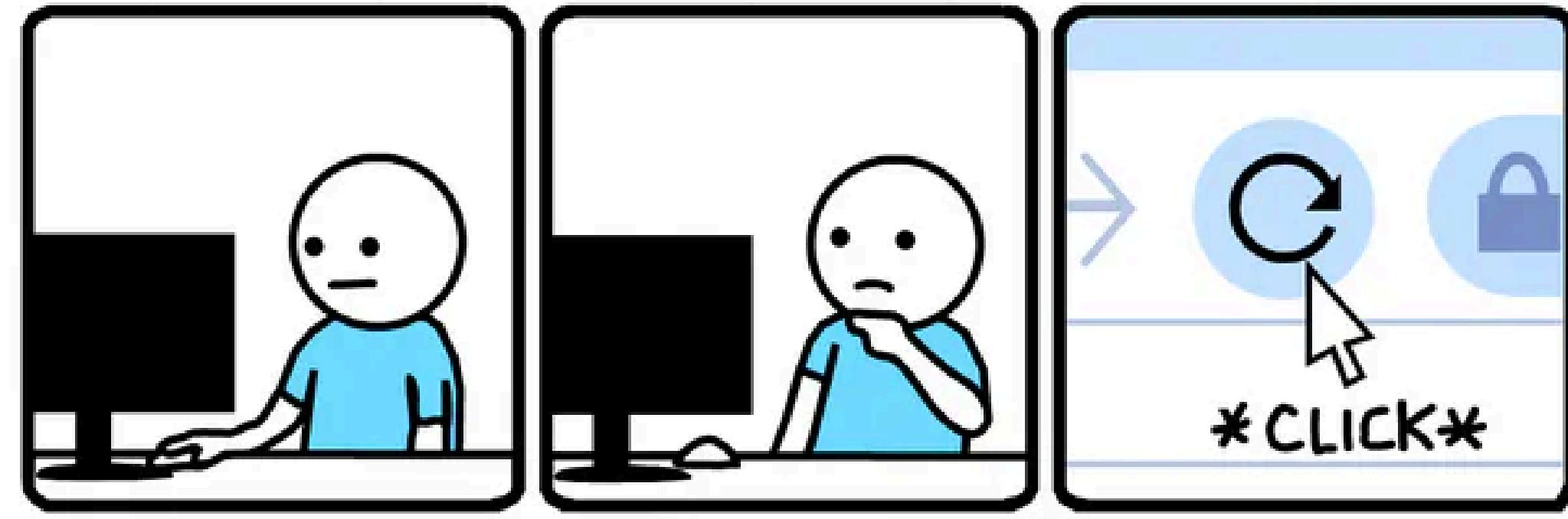
Temperature in Celcius: 19,5

Gas in Ohm: 57,9

Pressure in Pascal: 987.0700073242188

# Demo time!





# Grazie!