Introduzione a Micropython

Da zero al Cloud

con **Juna Salviati**

Chi sono

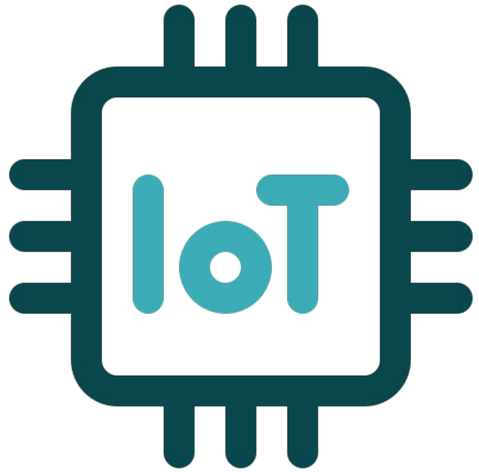
Juna Salviati
Solutions Architect
Technical communities enthusiast!

@1littleendian
https://medium.com/@1littleendian
https://dev.to/antigones

Un po’ di contesto: IoT, microcontrollori, board

IoT

- Dispositivi connessi in rete
- I dispositivi comunicano
  - per scambiare dati
  - per intraprendere azioni



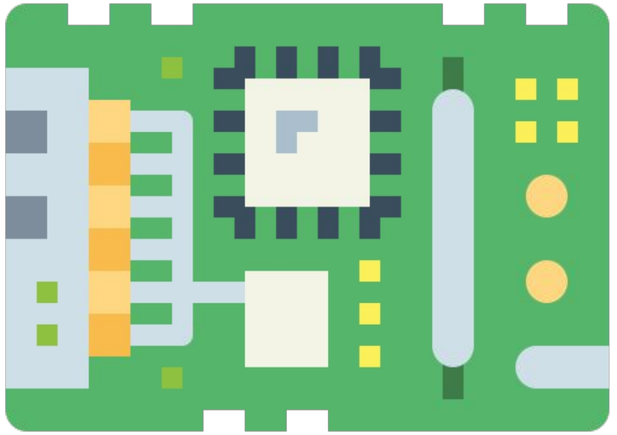
(image: Flaticon.com)

IoT per tutti: microcontrollori e board

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

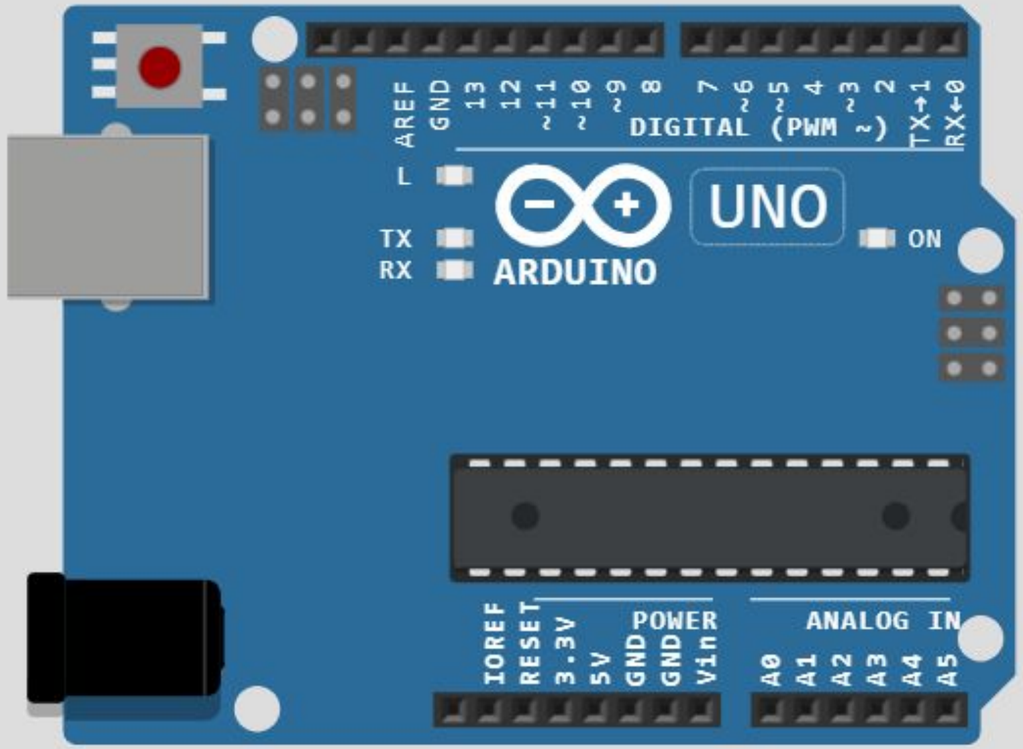


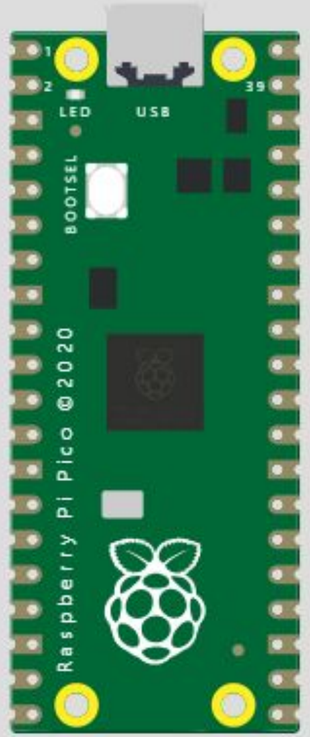
Single-board computers

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

Es: Raspberry PI

Board “in the wild”

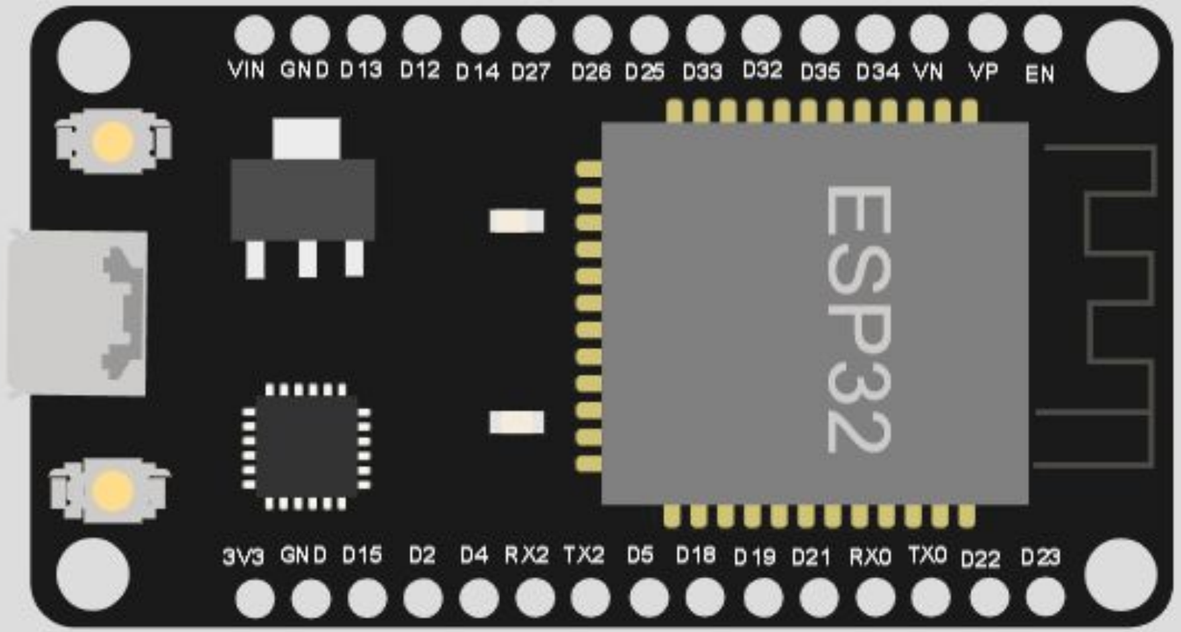




Perchè l’elettronica dei microcontrollori “è facile”

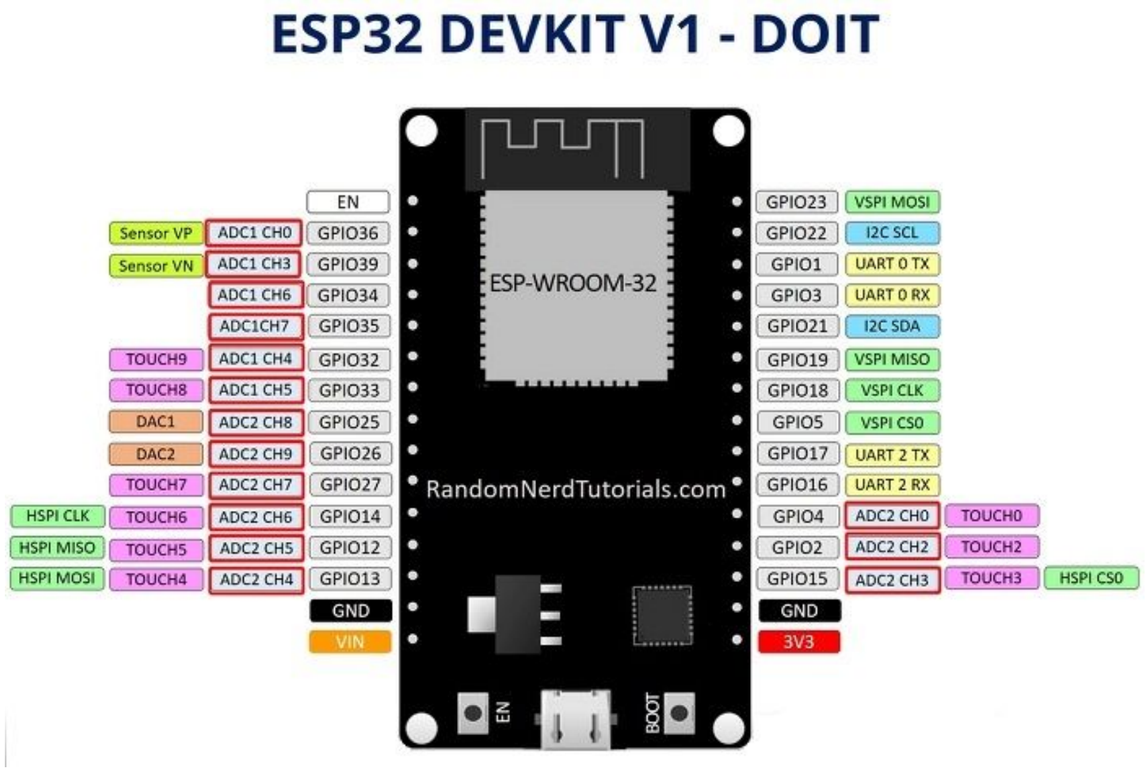
- Sposta il “carico” della conoscenza dall’hardware al software
- Conoscere pochi componenti permette di ottenere già risultati “apprezzabili” in termini di applicazioni
- C’è un ecosistema che semplifica ulteriormente il lavoro

Anatomia di una board



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

Identificare i pin



“Pinout”: schema dei pin con nome e finalità

MicroPython

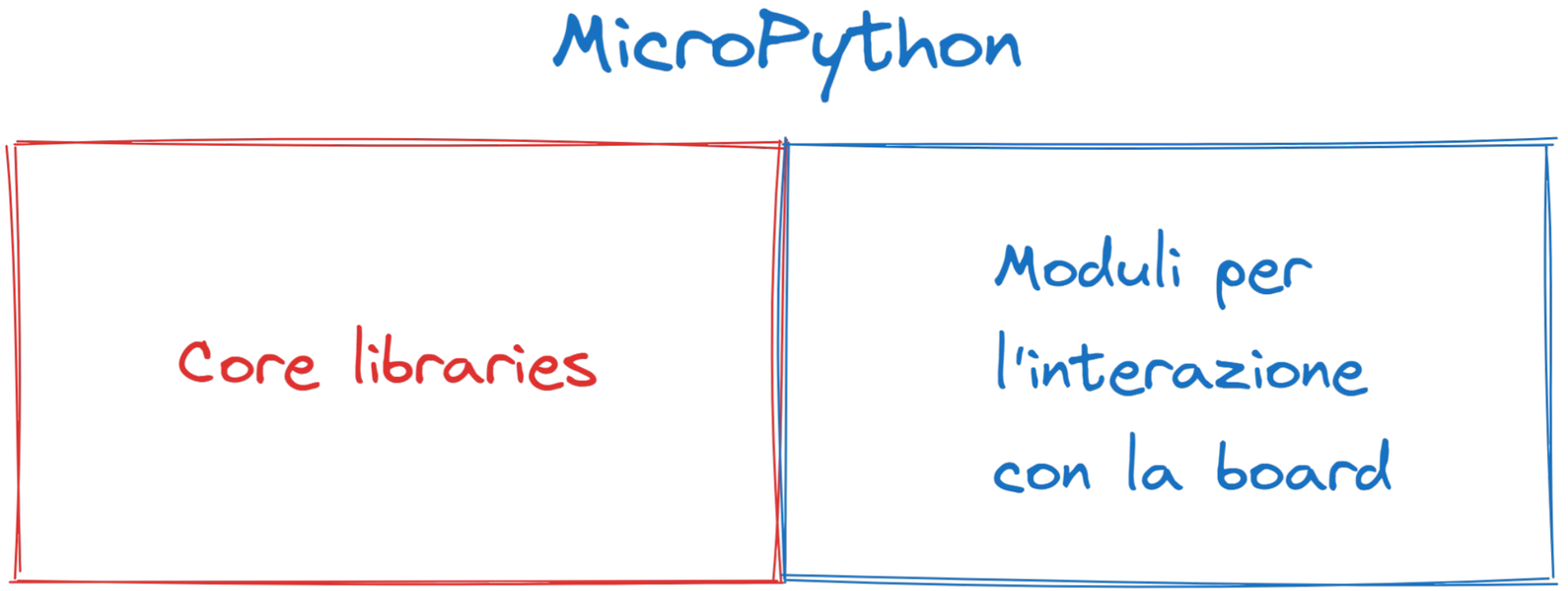
Una versione “light” di Python che può girare sui microcontrollori

- 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!

Cosa c’è in MicroPython



Simulazione di circuiti

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

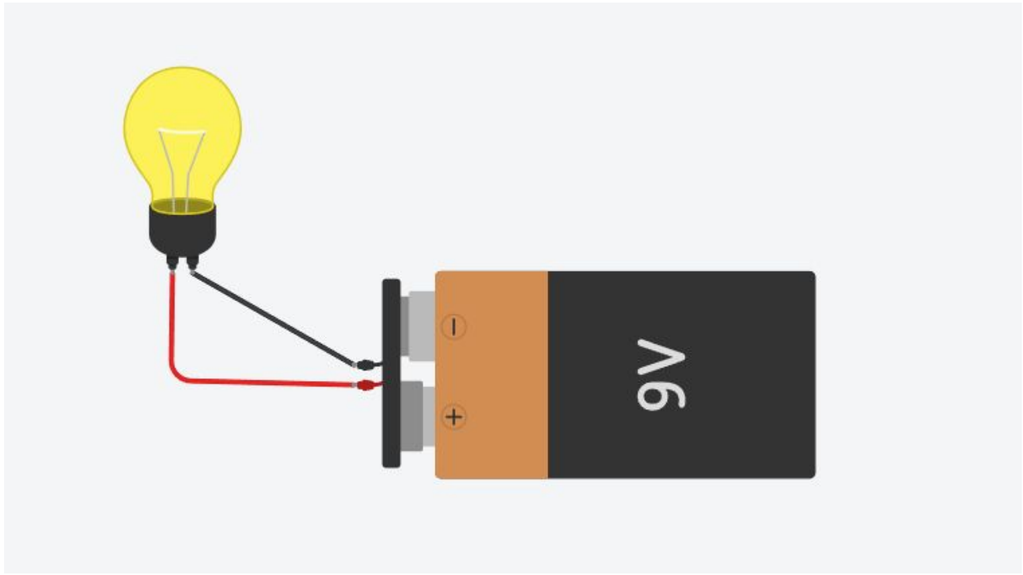
https://wokwi.com/

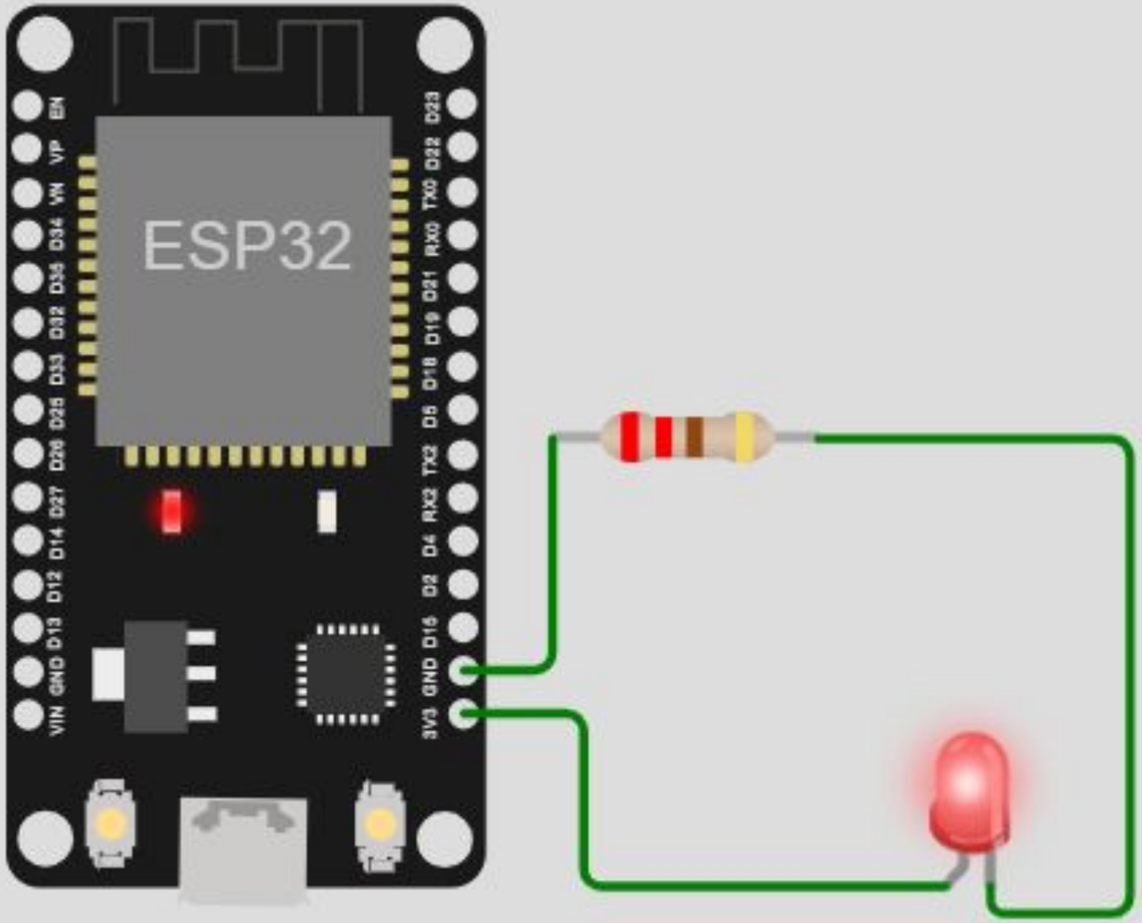
Elettronica 101

Come accendere una lampadina

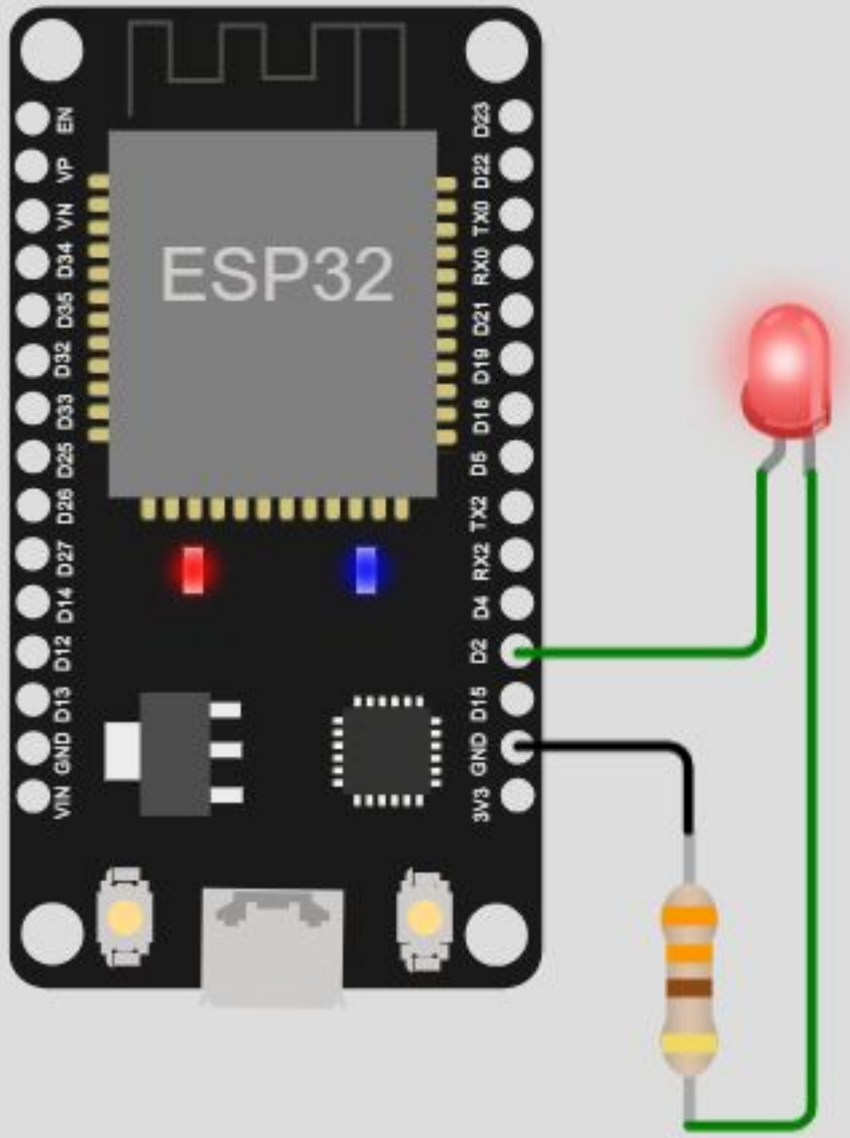
Alcuni fatti importanti:

- 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




ESP32
EN
VP
VN
D34
D35
D32
D33
D25
D26
D27
D14
D12
D13
GND
VIN
D23
D22
TX0
RX0
D21
D19
D18
D5
TX2
RX2
D4
D2
D15
GND
3V3

```
from machine import Pin


led = Pin(2, Pin.OUT)
led.value(1)
```



Termometro

Il termistore NTC

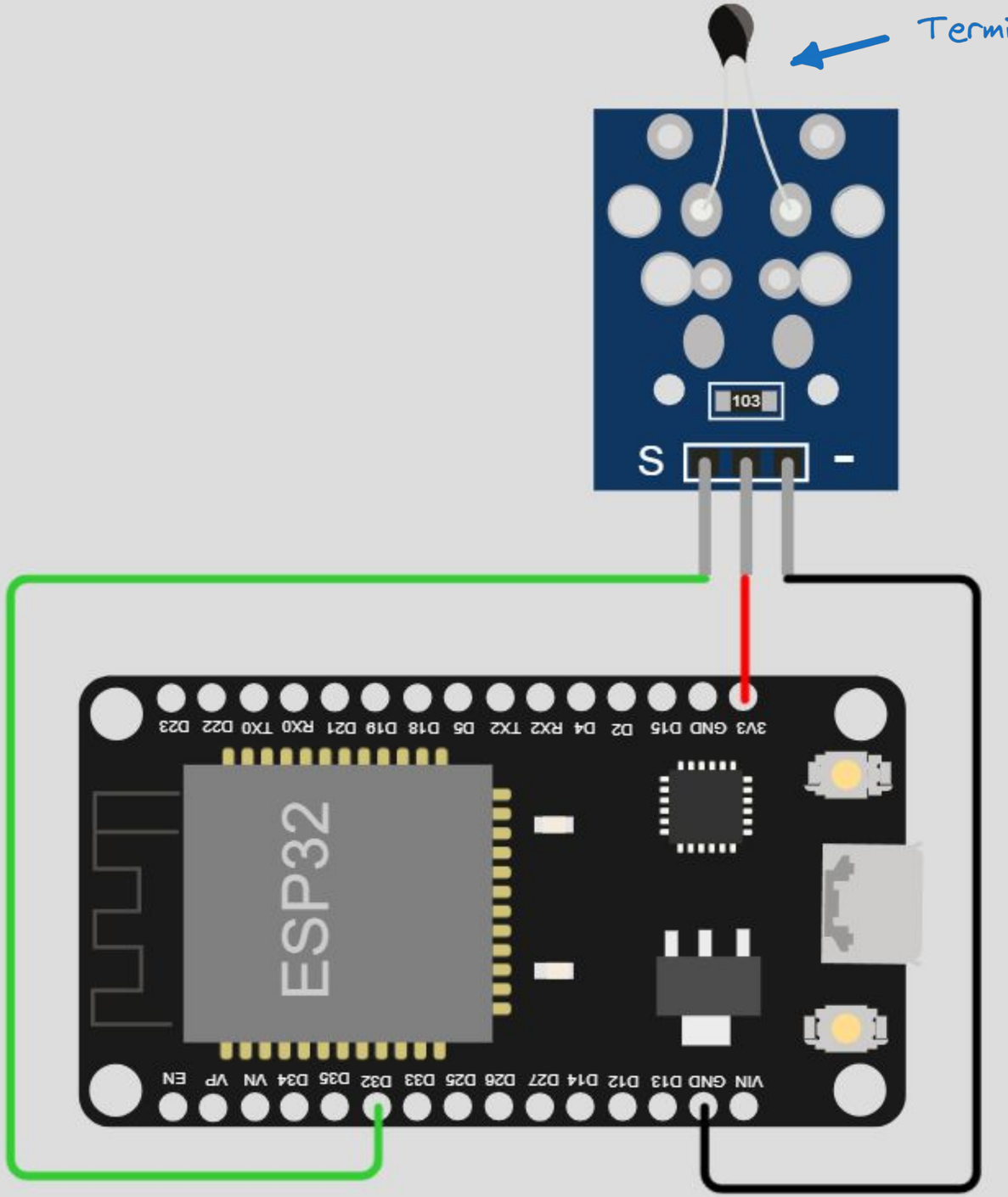
NTC (**Negative Temperature Coefficient**): la resistenza decresce con l'aumentare della temperatura

3 pin:

- VCC, l’alimentazione
- OUT, la “lettura”
- GND, la connessione al GND dell’ESP32

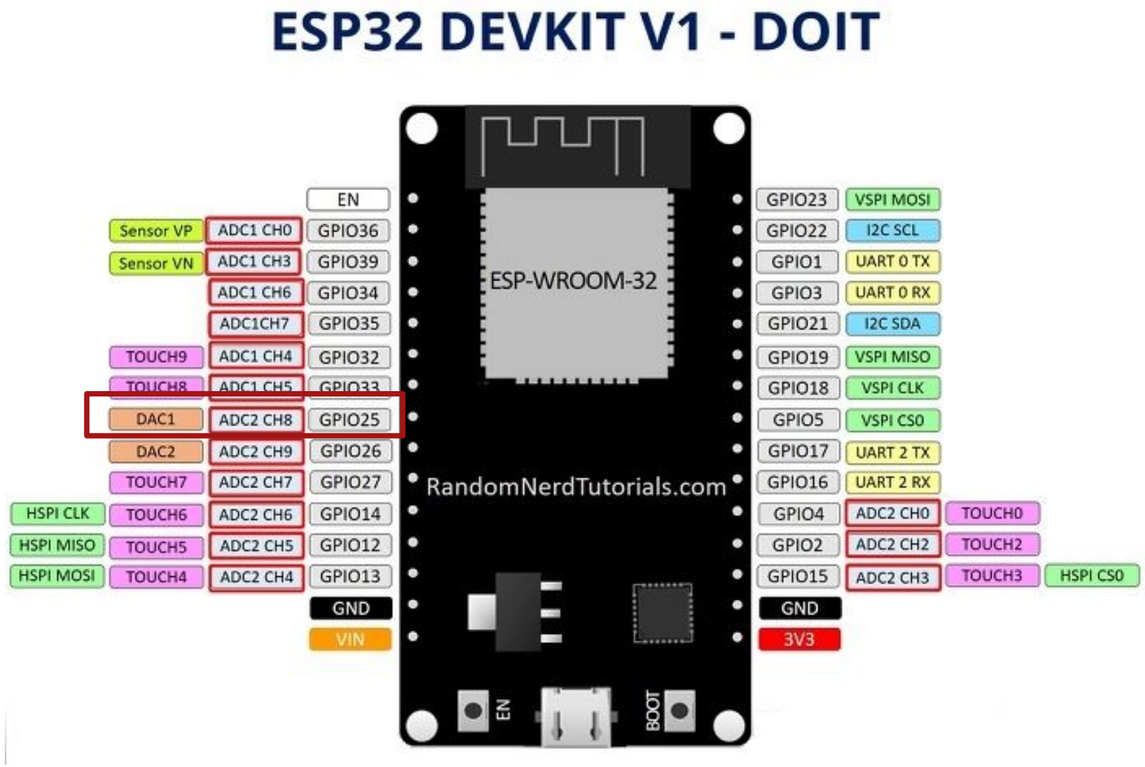
Pin names

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

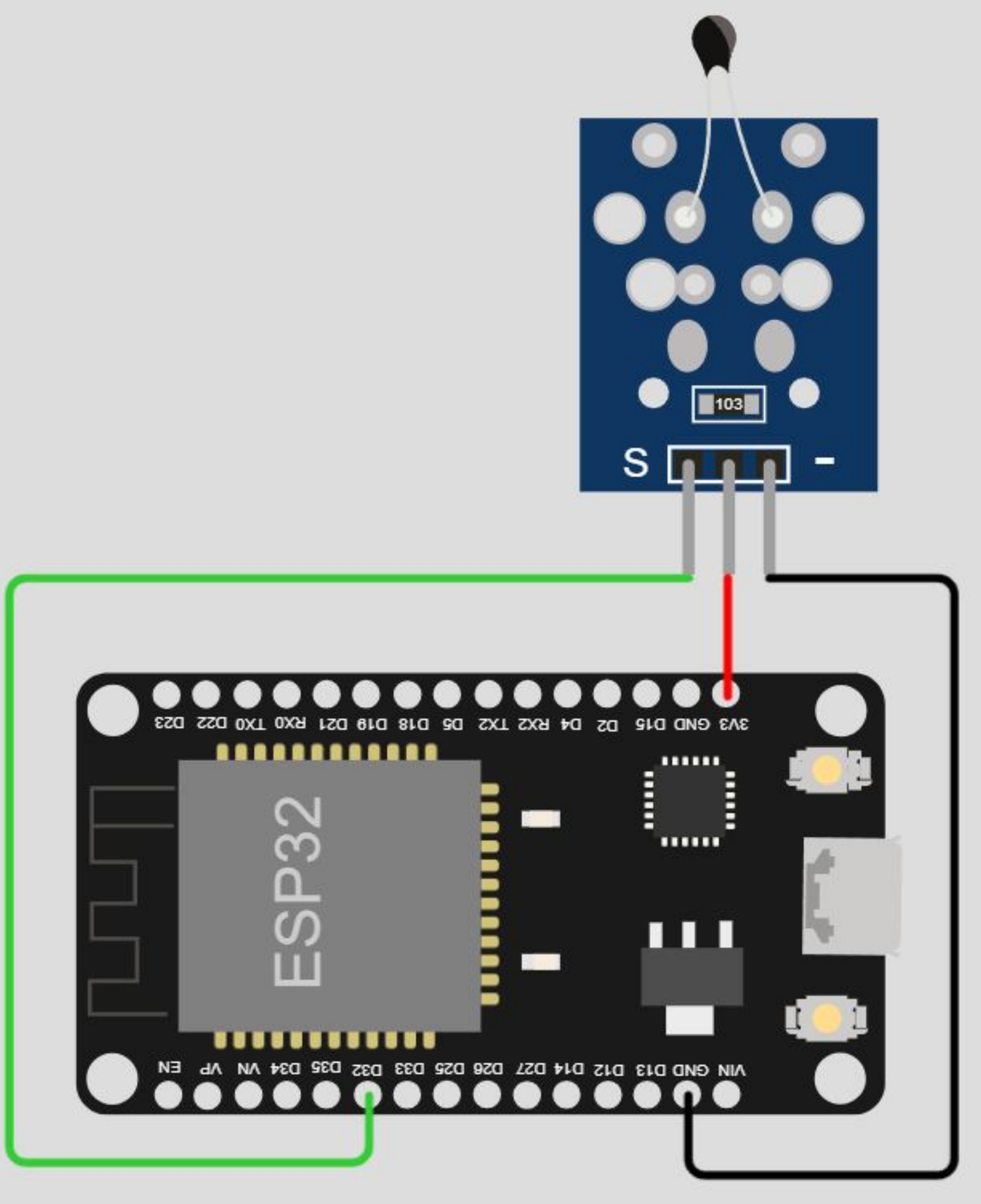


Solo pin ADC1 (problema del wifi)

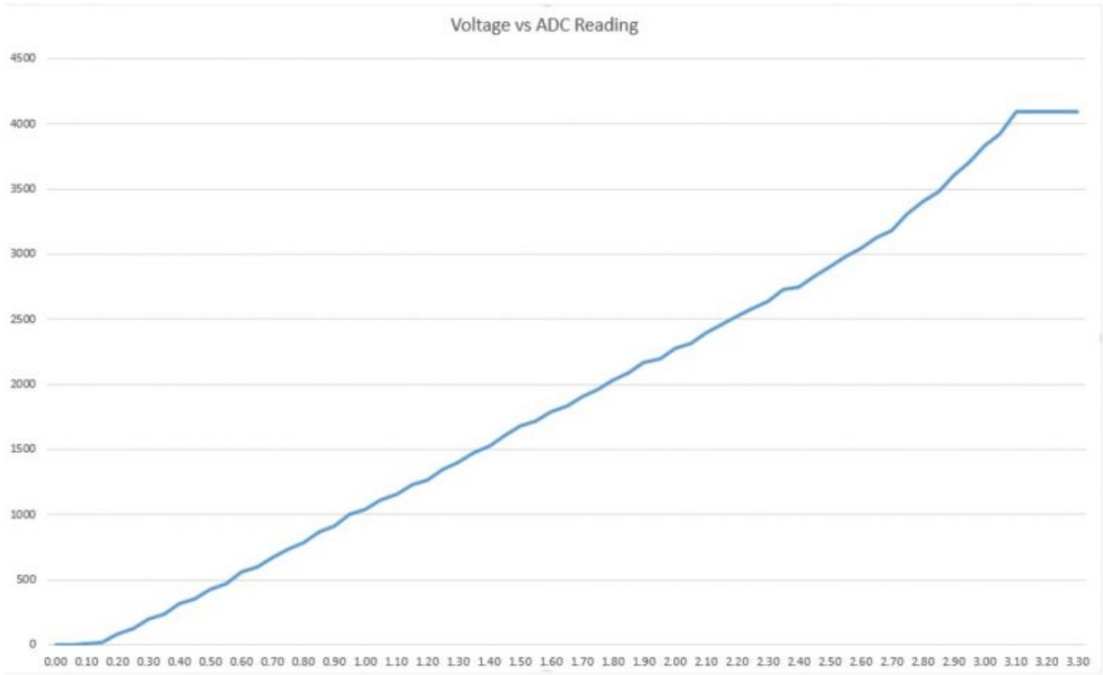
ESP32 - Pinout



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


p32 = ADC(Pin(32))

while True:
    print(p32.read())
    time.sleep(2)
```



Lettura di un pin ADC



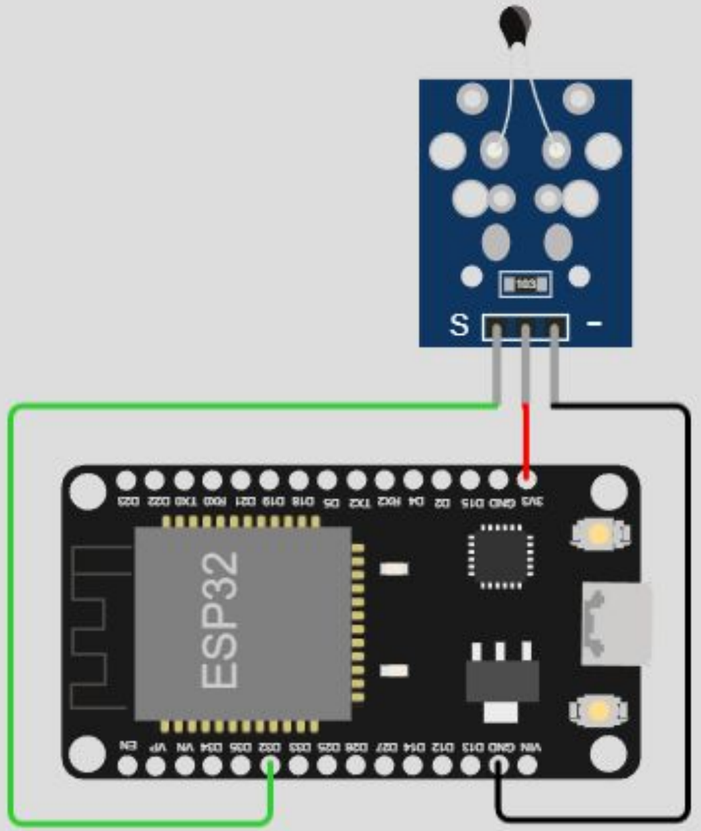
non lineare (https://randomnerdtutorials.com/esp32-adc-analog-read-arduino-ide/)

WOKWI SAVE SHARE temperature Docs

main.py diagram.json

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



p32 = ADC(Pin(32))


while True:
    v = p32.read()
    print(v)
    time.sleep(2)

```

Simulation




```
730
730
730
730
730
```

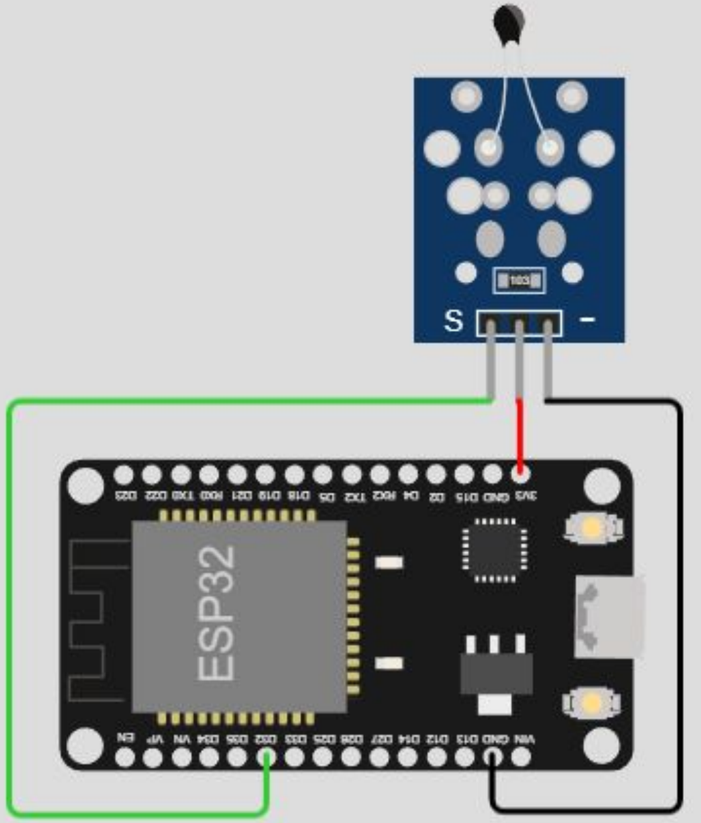
Python Biella Group

WOKWi SAVE SHARE temperature Docs

main.py ● diagram.json ●

```
{
  "version": 1,
  "author": "Juna Salviati",
  "editor": "wokwi",
  "parts": [
    {
      "type": "wokwi-esp32-devkit-v1",
      "id": "esp",
      "top": 108.7,
      "left": -14,
      "rotate": 270,
      "attrs": { "env": "micropython-20230426-v1.20.0" }
    },
    {
      "type": "wokwi-ntc-temperature-sensor",
      "id": "ntc1",
      "top": 35.27,
      "left": 27.7,
      "rotate": 90,
      "attrs": {}
    }
  ],
  "connections": [
    [ "esp:TX0", "$serialMonitor:RX", "", [] ],
    [ "esp:RX0", "$serialMonitor:TX", "", [] ],
    [ "esp:GND.2", "ntc1:GND", "black", [ "v19.32", "h64.02", "v-144.1" ] ],
    [ "ntc1:OUT", "esp:D32", "limegreen", [ "h-163.61", "v141.03", "h86.6" ] ],
    [ "esp:3V3", "ntc1:VCC", "red", [ "v0" ] ]
  ],
  "dependencies": {}
}
```

Simulation



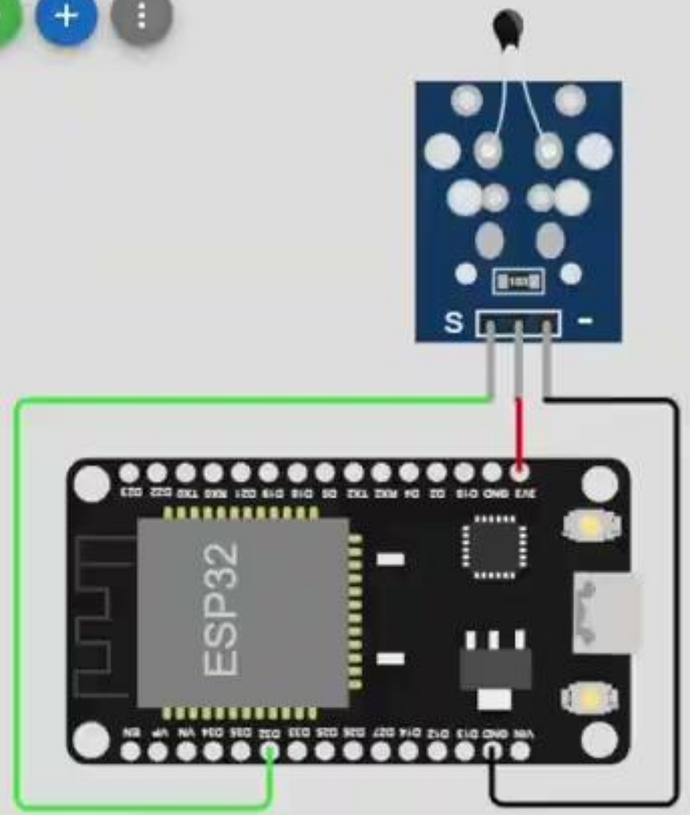

```
730
730
730
730
730
```

WOKWI SAVE SHARE temperature

main.py diagram.json

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

BETA = 3950 # Beta Coefficient of the thermistor


p32 = ADC(Pin(32))
p32.width(ADC.WIDTH_10BIT) # reduce pin reading resolution to match Arduino Uno


while True:
  v = p32.read()
  print(v)
  time.sleep(2)

```

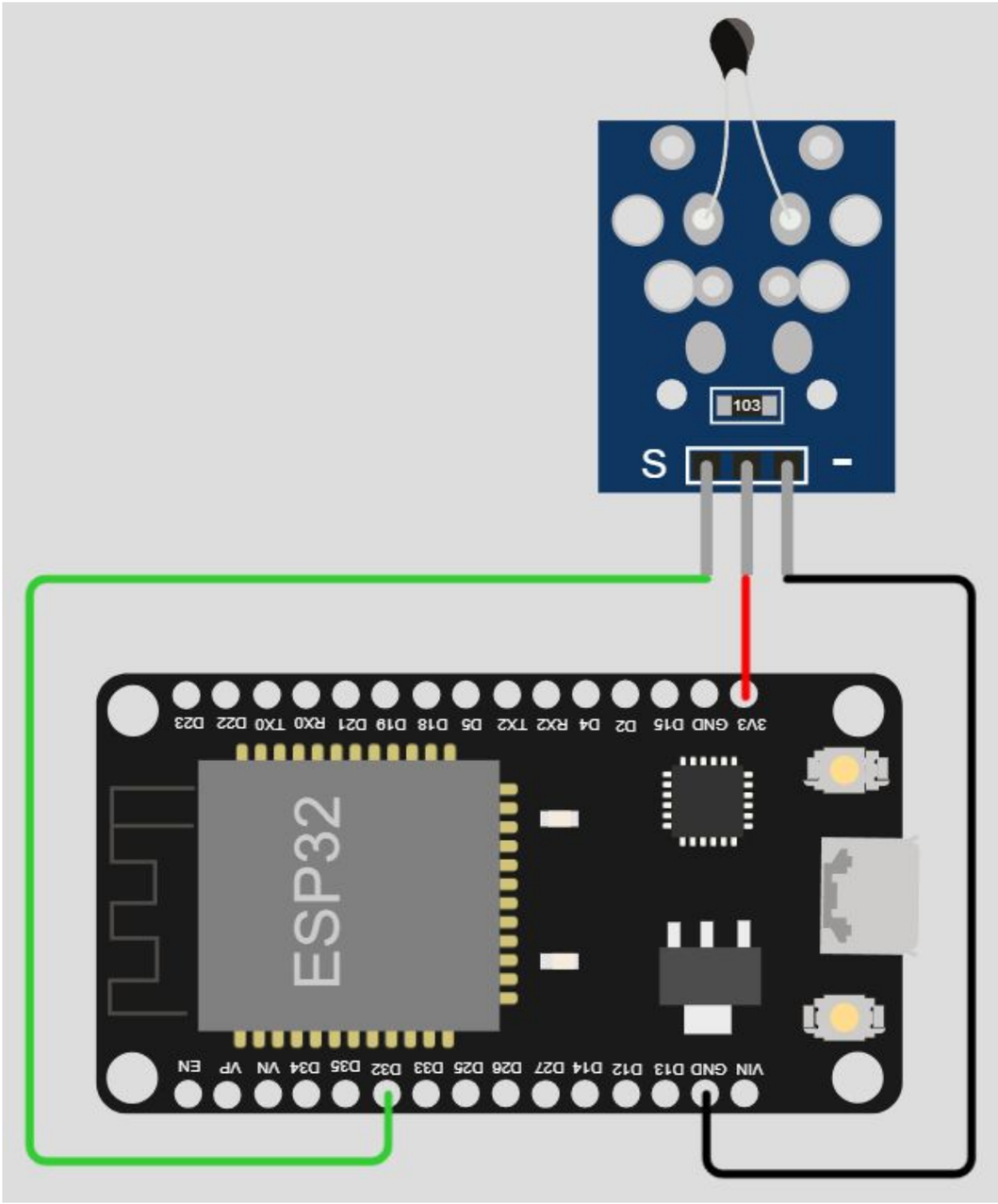
Simulation




```
26.01556
500
26.01556
500
26.01556
500
26.01556
500
26.01556
500
26.01556
```

Da Volt a temperatura

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

BETA = 3950 # Beta Coefficient of the thermistor


p32 = ADC(Pin(32))
p32.width(ADC.WIDTH_10BIT) # Change pin resolution

while True:
  v = p32.read()
  print(v)
  celsius = 1 / (math.log(1 / (1023 / v - 1)) / BETA + 1.0 / 298.15) -
273.15
  print(celsius)
  time.sleep(2)
```



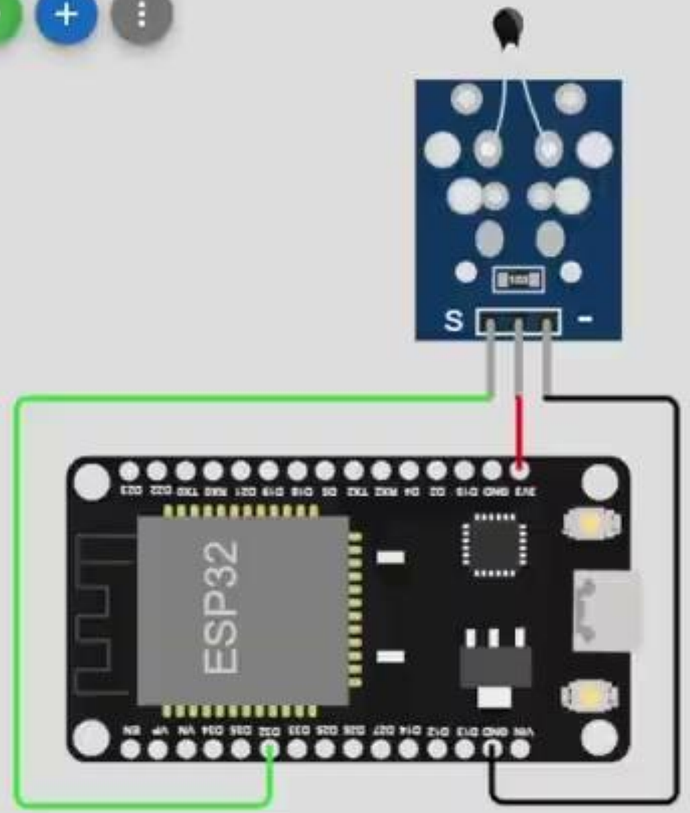
Python Biella Group

WOKWI SAVE SHARE temperature

main.py diagram.json

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

BETA = 3950 # Beta Coefficient of the thermistor


p32 = ADC(Pin(32))
p32.width(ADC.WIDTH_10BIT) # reduce pin reading resolution to match Arduino Uno


while True:
  v = p32.read()
  print(v)
  celsius = 1 / (math.log(1 / (1023 / v - 1)) / BETA + 1.0 / 298.15) - 273.15
  print(celsius)
  time.sleep(2)
```

Simulation




```
ho 0 tail 12 room 4
load:0x40080400,len:3712
entry 0x4008064c
523
23.99133
523
23.99133
523
23.99133
449
30.63168
```

Utilizzare la breadboard ed evolvere il termometro

La breadboard

Connessione in orizzontale

Connessione in verticale

1 e 2 non sono connessi

TM1637 (7 segmenti)

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

Pin names

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

“Sotto al cofano”

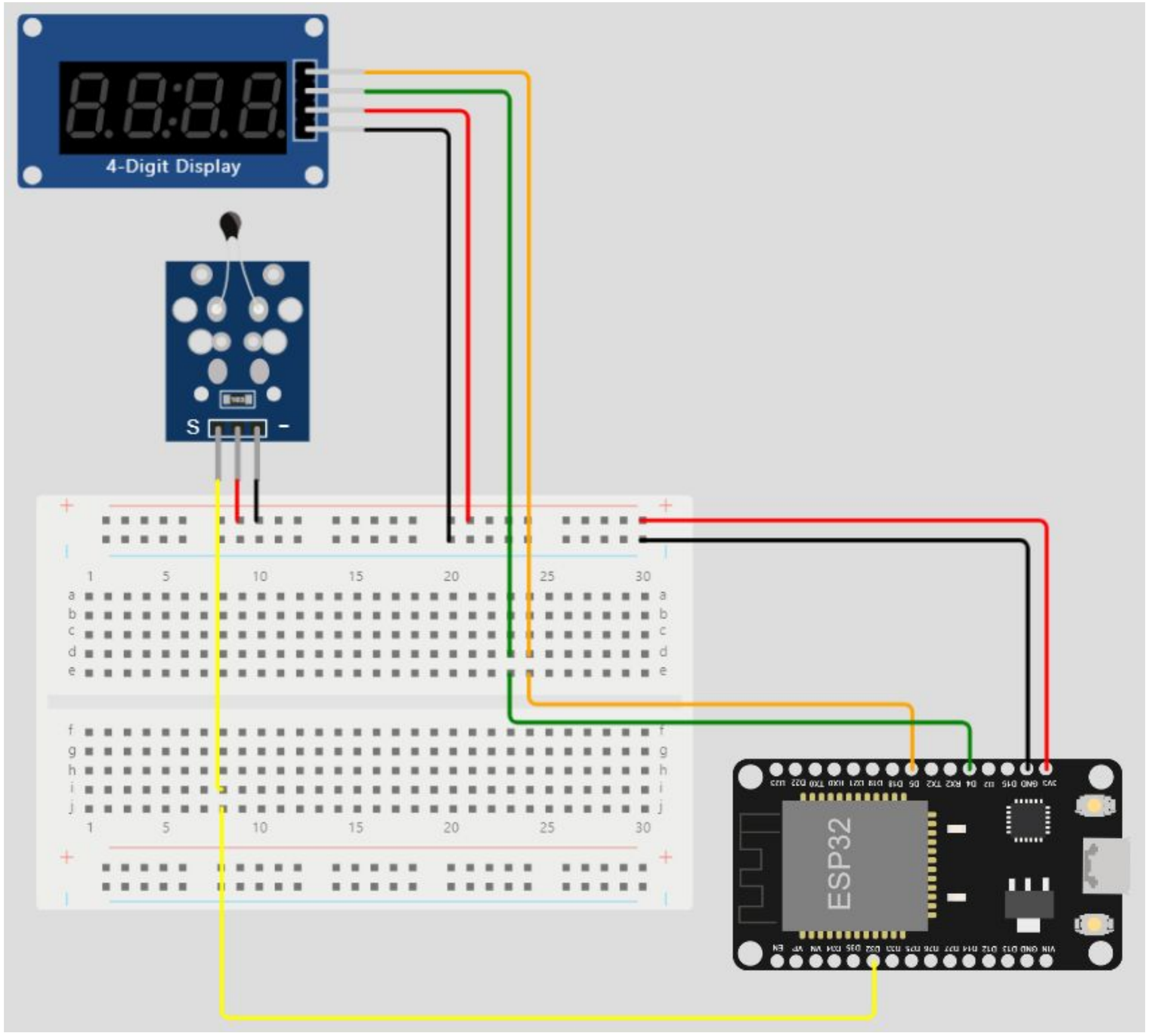
Un “driver” scrive sul bus al posto nostro:

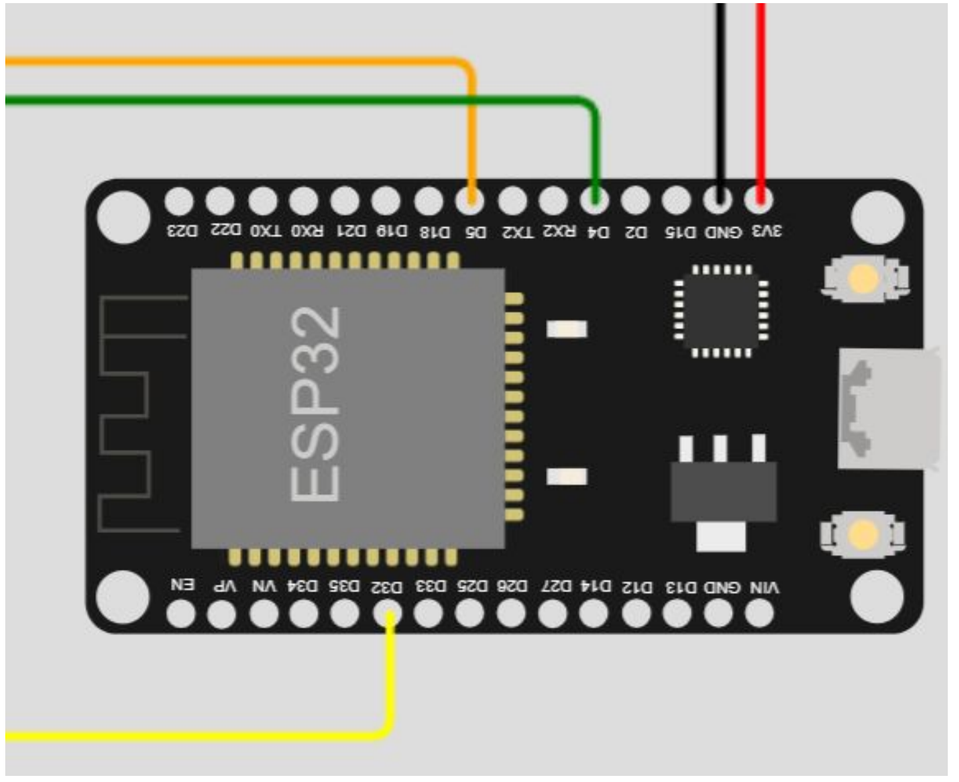
- ...secondo un “quasi” protocollo I2C
- ...sequenze di start/stop dei comandi/dati
- ...come accendere i segmenti

Specifica nei datasheet

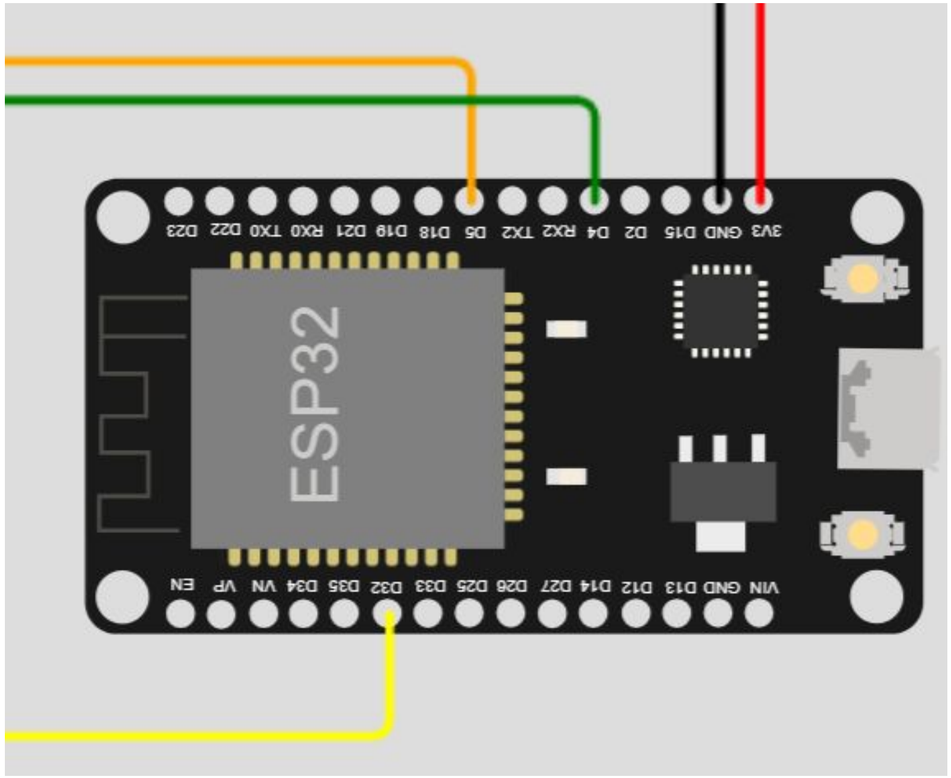
“Sotto al cofano”

```
TM1637_CMD3 = const(128) # 0x80 display control command
TM1637_DSP_ON = const(8) # 0x08 display on


def _write_dsp_ctrl(self):
        # display on, set brightness
        self._start()
        self._write_byte(TM1637_CMD3 | TM1637_DSP_ON | self._brightness)
        self._stop()
```


4-Digit Display
S
ESP32


ESP32

```
from machine import Pin, ADC
import time
import math
import tm1637
display = tm1637.TM1637(clk=Pin(5), dio=Pin(4))

BETA = 3950 # Beta Coefficient of the thermistor

p2 = ADC(Pin(2))
p2.width(ADC.WIDTH_10BIT)

while True:
  v = p2.read()
  celsius = 1 / (math.log(1 / (1023 / v - 1)) / BETA +
1.0 / 298.15) - 273.15
  display.temperature(round(celsius))
  time.sleep(2)
```

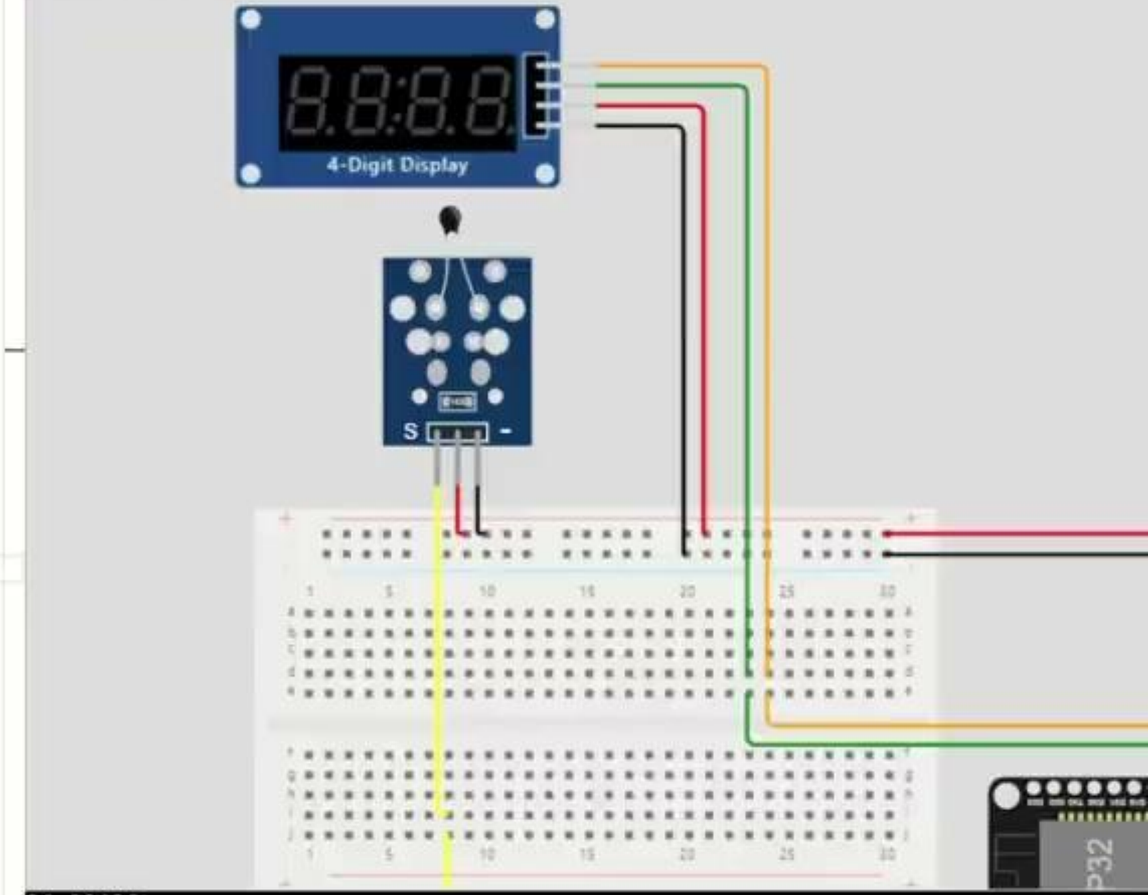


WOKWi SAVE SHARE temperature_with_display

main.py | diagram.json | tm1637.py

```python
from machine import Pin, ADC
import time
import math
import tm1637
display = tm1637.TM1637(clk=Pin(5), dio=Pin(4))

BETA = 3950 # Beta Coefficient of the thermistor


p32 = ADC(Pin(32))
p32.width(ADC.WIDTH_10BIT)


while True:
  v = p32.read()
  print(v)
  celsius = 1 / (math.log(1 / (1023 / v - 1)) / BETA + 1.0 / 298.15) - 273.15
  display.temperature(round(celsius))
  print(celsius)
  time.sleep(2)

```

Simulation




```
34.33496
410
34.33496
410
34.33496
410
34.33496
410
34.33496
410
34.33496
```

Inviare dati

Inviare dati

MQTT (MQ Telemetry Transport or Message Queuing Telemetry Transport)

- Protocollo di messaggistica “leggero” su TCP/IP
- Publisher/Subscriber
- Basso impatto/Banda limitata

Protocollo MQTT

Definisce due tipi di entità:

- Message broker
- Un certo numero di client

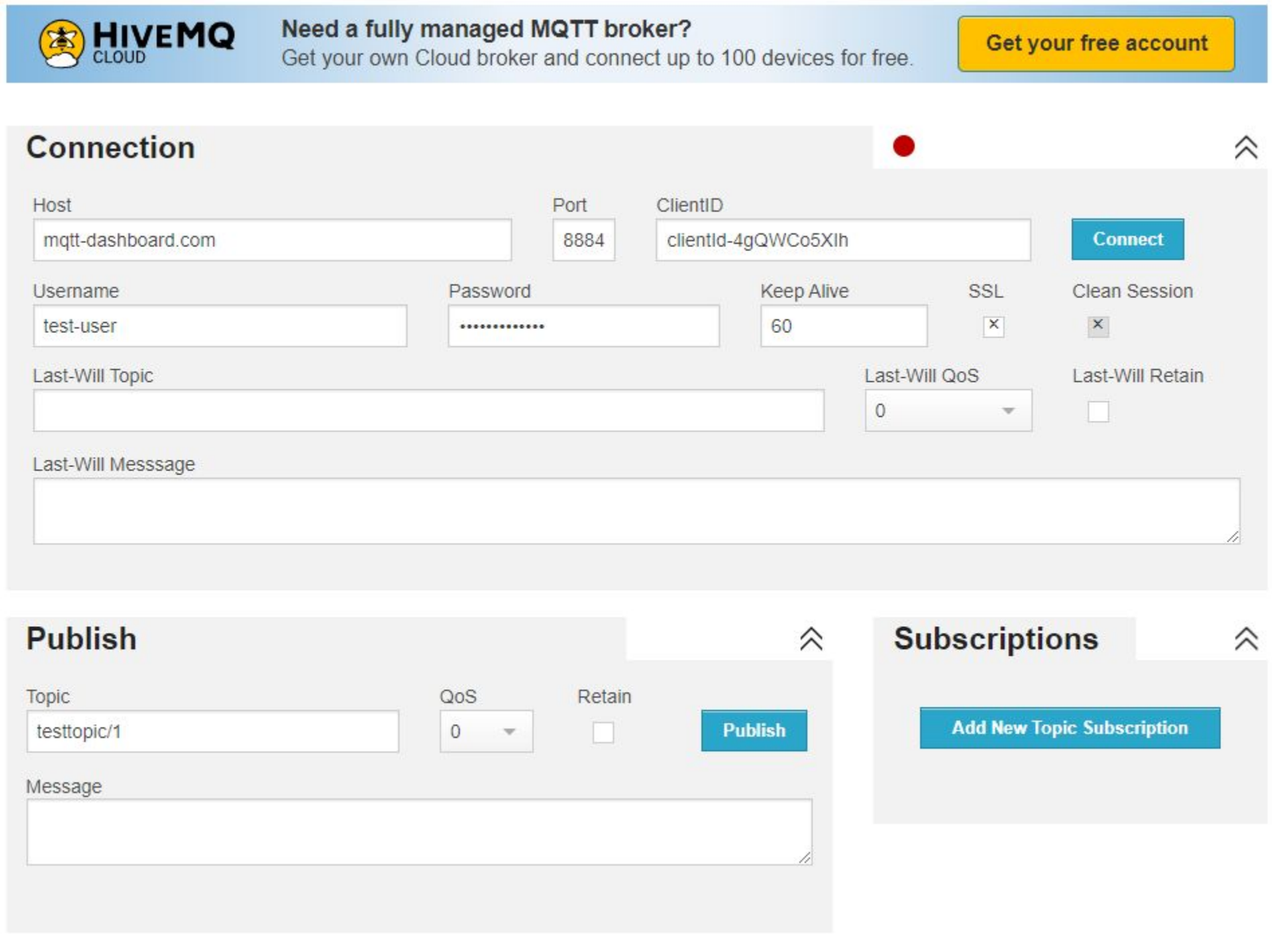
Cosa facciamo

1. Rileviamo la temperatura
2. Inviamo un messaggio json ad un servizio MQTT di demo
3. Usiamo una dashboard web per leggere i dati inviati

https://wokwi.com/projects/322577683855704658

Configurazione del servizio

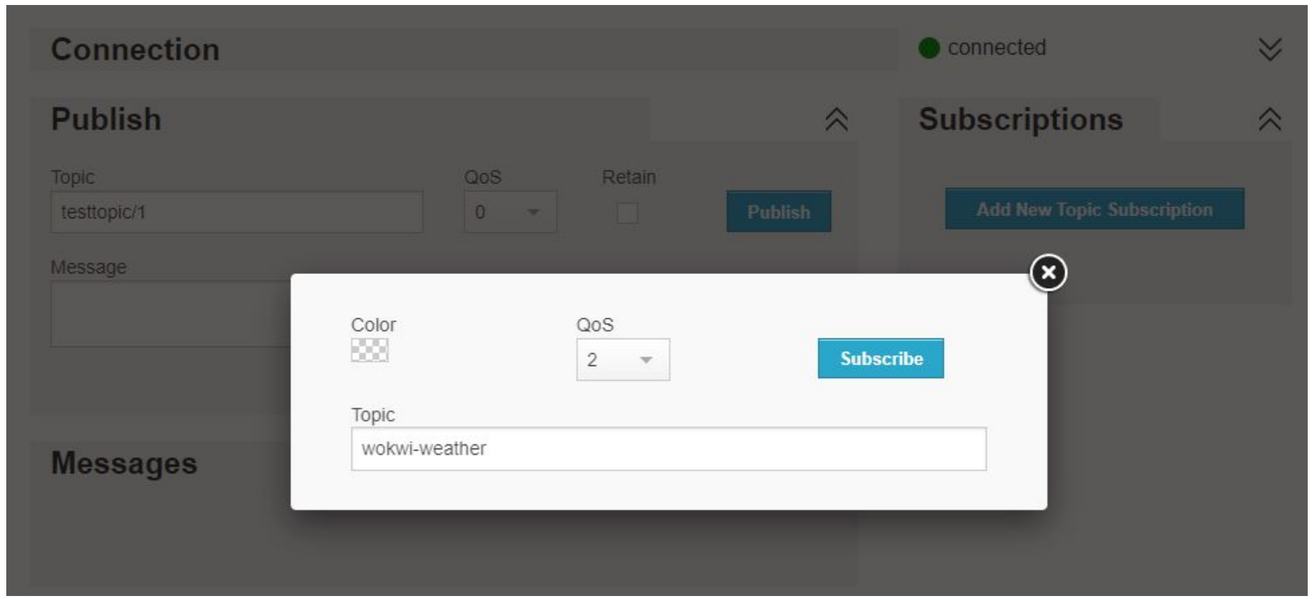
http://www.hivemq.com/demos/websocket-client/ + “Connect”



Configurazione del servizio

Click su "Add New Topic Subscription"

Topic: "wokwi-weather" + "Subscribe"



```
import network
import ujson
from umqtt.simple import MQTTClient
...
MQTT_CLIENT_ID = "micropython-weather-demo"
MQTT_BROKER    = "broker.mqttdashboard.com"
MQTT_USER      = ""
MQTT_PASSWORD  = ""
MQTT_TOPIC     = "wokwi-weather"
```

Connessione al WiFi (simulato)

```
print("Connecting to WiFi", end="")
display.scroll("Connecting to WiFi")
sta_if = network.WLAN(network.STA_IF)
sta_if.active(True)
sta_if.connect('Wokwi-GUEST', '')
loading_str = ''
while not sta_if.isconnected():
  print(".", end="")
  loading_str += '.'
  display.scroll(loading_str)
  time.sleep(0.1)
print("Connected!")
```

Attenzione: il WiFi del simulatore è monitorato, quindi non va bene per far viaggiare alcuni tipi di informazioni

Invio dei messaggi

```
p32 = ADC(Pin(32))
p32.width(ADC.WIDTH_10BIT)
prev_weather = ""
while True:
  v = p32.read()
  celsius = round(1 / (math.log(1 / (1023 / v - 1)) / BETA + 1.0 / 298.15) - 273.15)
  display.temperature(celsius)
  message = ujson.dumps({
    "temp": celsius
  })
  if message != prev_weather:
    client.publish(MQTT_TOPIC, message)
    prev_weather = message
  time.sleep(2)
```

```
rst:0x1 (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:2
load:0x3fff0030,len:4656
load:0x40078000,len:13284
ho 0 tail 12 room 4
load:0x40080400,len:3712
entry 0x4008064c
Connecting.....................................................................
.....................................................................
.....................................................................
.....................................................................
.....................................................Connected!
Connecting to MQTT server...Connected!
523
Updated!
Reporting to MQTT topic wokwi-weather: {"temp": 24}
```

Connection

connected

Publish

Topic

testtopic/1

QoS

0

Retain

Publish

Message

Subscriptions

Add New Topic Subscription

Qos: 2

wokwi-weather

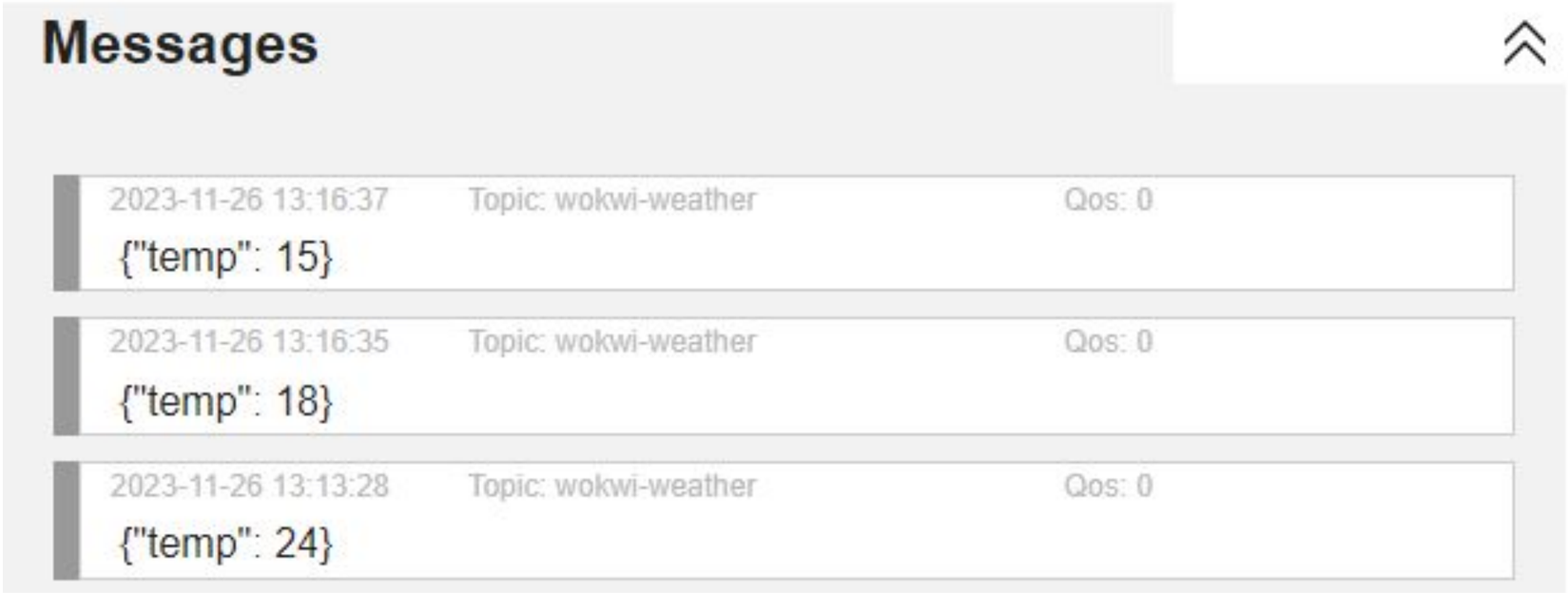
x

Messages

2023-11-26 13:13:28 Topic: wokwi-weather Qos: 0

{"temp": 24}

```
Updated!
Reporting to MQTT topic wokwi-weather: {"temp": 18}
627
Updated!
Reporting to MQTT topic wokwi-weather: {"temp": 15}
627
```



Un caso d’uso: predizione della crescita delle colture

Predire la crescita in agricoltura

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-For-Beginners/blob/main/2-farm/lessons/1-predict-plant-growth/README.md

edge computing

Grazie!