"Go Green for a Sustainable World"
Erasmus+ project 2025-1-FR01-KA220-SCH-00036613
Co-funded by the European Union
GO GREEN
By Ecomuseum Zagori S.C.E.
ECO MUSEUM ZAGORI

IoT practical: How to set up a simple experiment for schools

- MicroComputer Systems Laboratory team
- Prof. Charis Papadopoulos
- Dr.[2] Sotirios Kontogiannis

https://kalipso.math.uoi.gr

Μικροεπεξεργαστές

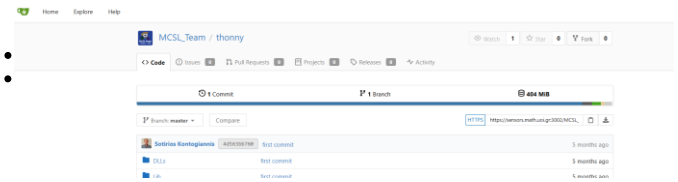
Presentation Outline

1. **Python Programming Fundamentals: Introduction to essential programming concepts, syntax, data structures, and control flow in Python.**
2. **Internet of Things Fundamentals and ESP8266 Microcontroller Platforms: Study of IoT architectures, embedded connectivity, and practical implementation using ESP8266-based development boards.**
3. **Sensor Technologies and Interfacing for Irrigation Control: Integration of digital and analog sensors for environmental monitoring, data acquisition, and automated irrigation decision support.**
4. **Embedded Programming with MicroPython and Thonny IDE: Development of IoT-oriented sensing and control applications using MicroPython in resource-constrained embedded systems.**
5. **Telemetry, Data Transmission, and Cloud Connectivity: Design and implementation of Wi-Fi-based communication protocols for transmitting sensor data to cloud platforms.**
6. **Data Visualization and IoT Dashboard Development: Introduction to ThingsBoard for cloud-based data visualization, telemetry monitoring, and interactive dashboard design.**

1. Embedded programming-MicroPython Basics

Thonny IDE

https://sensors.math.uoi.gr:3002/MCSL_Team/thonny



First install git tool using the following link:
https://git-scm.com/install/windows
Add to the path using cmd:
Alt+R cmd
>setx PATH "%PATH%;C:\Program Files\Git\cmd“
>git --version
>git config --global http.sslVerify false
>git config --global --get http.sslVerify
>cd \
>git clone https://sensors.math.uoi.gr:3002/MCSL_Team/thonny
>setx PATH “%PATH%;c:\thonny;c:\thonny\bin”
>thonny

Adding MicroPython to the ESP8266 device

1. Install esptool.py

>python -m pip install esptool

2. Find the COMM port of the device from the device manager

>wmic path Win32_SerialPort get DeviceID,Name,Description

3. Erase Flash

>python -m esptool --port COM5 erase_flash

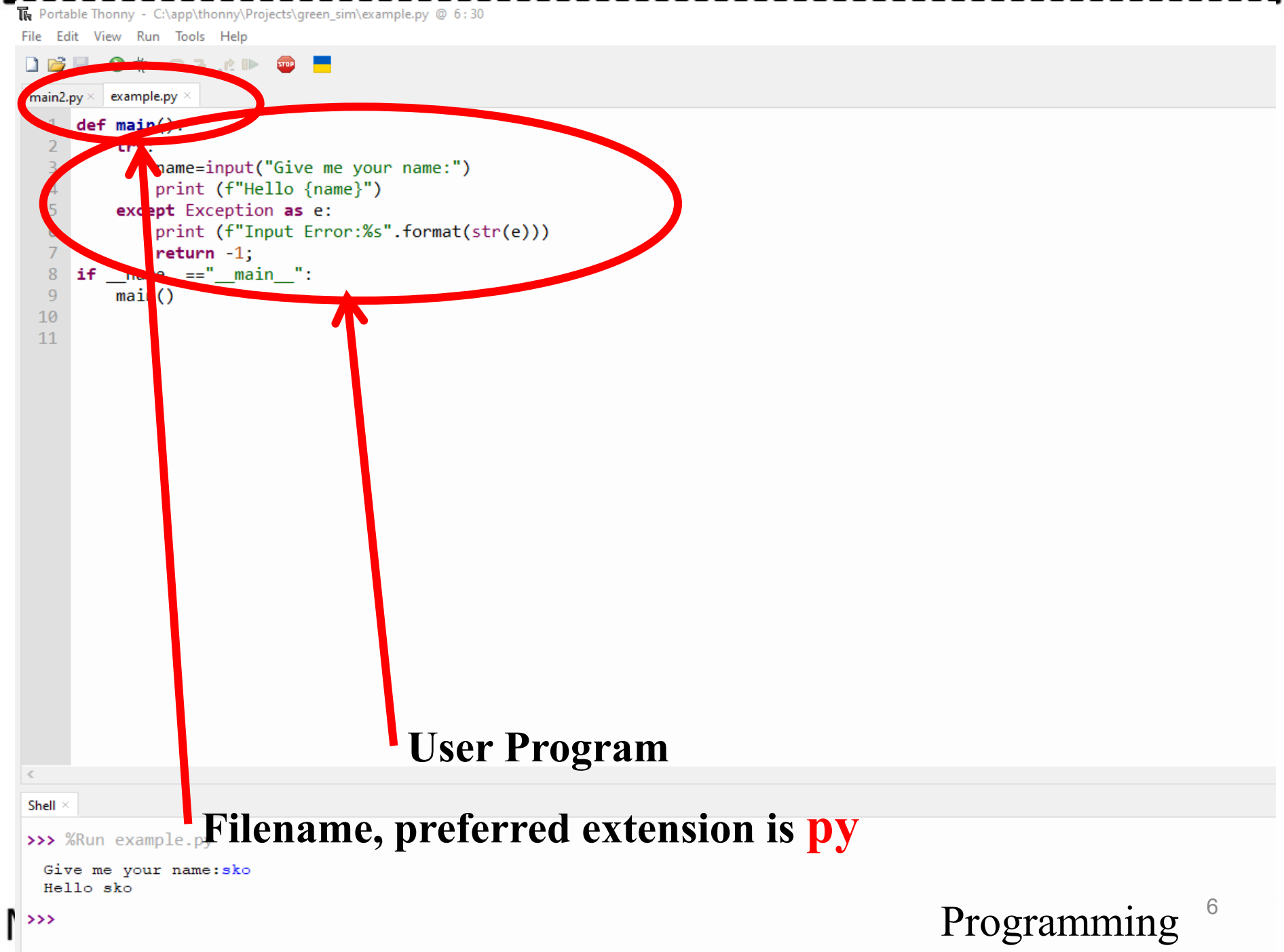
If it does not connect, hold FLASH/BOOT, press RST, release RST, then release FLASH/BOOT.

4. Flash the MicroPython firmware

>python -m esptool --port COM5 --baud 460800 write_flash --flash_size=detect 0 ESP8266_GENERIC-20251209-v1.27.0.bin

https://micropython.org/download/ESP8266_GENERIC/

ESP8266_GENERIC-20251209-v1.27.0.bin


Portable Thonny - C:\app\thonny\Projects\green_sim\example.py @ 6 : 30
File Edit View Run Tools Help
main2.py
example.py
def main():
try:
name=input("Give me your name:")
print (f"Hello {name}")
except Exception as e:
print (f"Input Error:%s".format(str(e)))
return -1;
if __name__=="__main__":
main()
User Program
Filename, preferred extension is py
Shell
>>> %Run example.py
Give me your name:sko
Hello sko
>>>

Types in Python

int

Bounded integers, e.g. 732 or -5

float

Real numbers, e.g. 3.14 or 2.0

long

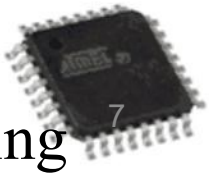
Long integers with unlimited precision

str

Strings, e.g. ‘hello’ or ‘C’

bool

True or False



Types in Python

Scalar

Indivisible objects that do not have internal structure

int (signed integers), **float** (floating point), **bool** (Boolean), ***NoneType***

NoneType is a special type with a single value

The value is called **None**

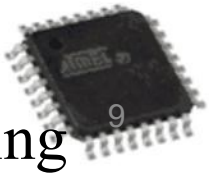
Non-Scalar

Objects having internal structure

str (strings)

Operators

Arithmetic	+	-	*	//	/	%	**
Comparison	==	!=	>	<	>=	<=	
Assignment	=	+=	-=	*=	//=	/=	%=	**=
Logical	and	or	not				
Bitwise	&	|	^	~	>>	<<	
Membership	in	not in					
Identity	is	is not					



Variables

A name associated with an object

Assignment used for binding

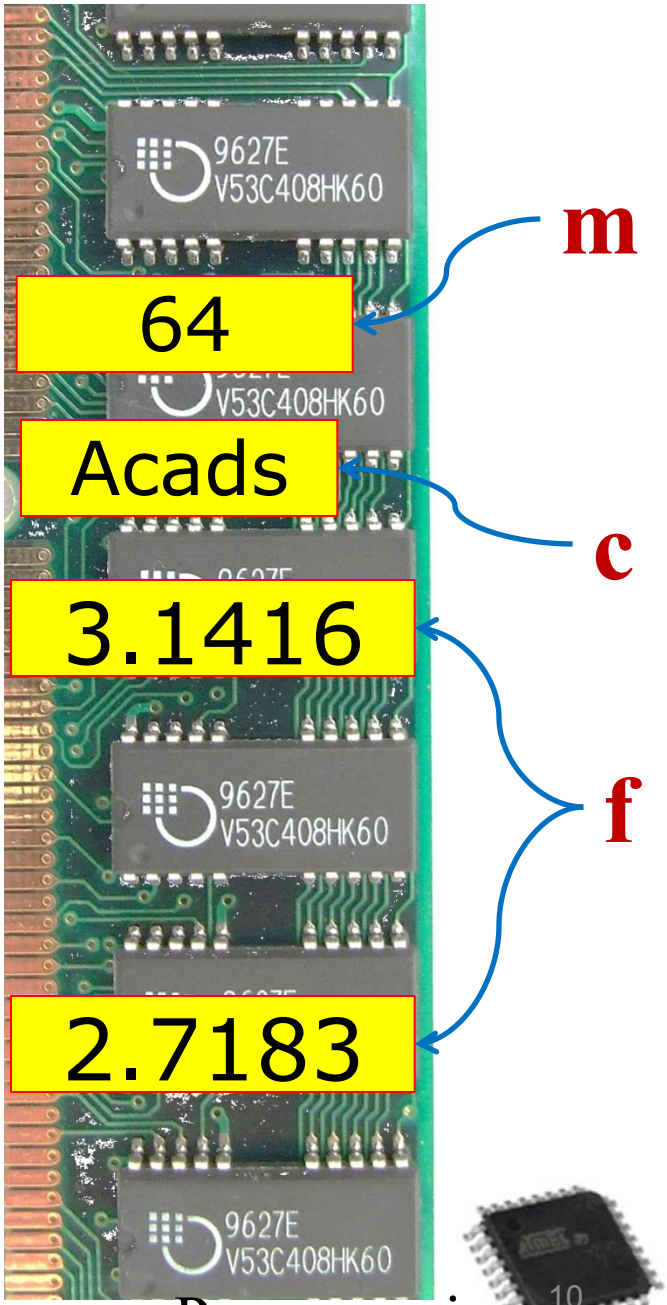
m = 64;

c = ‘Acads’;

f = 3.1416;

Variables can change their bindings

f = 2.7183;



Assignment Statement

A simple assignment statement

Variable = Expression;

Computes the value (object) of the expression on the right hand side expression (RHS)

Associates the name (variable) on the left hand side (LHS) with the RHS value

= is known as the assignment operator.

Multiple Assignments

Python allows multiple assignments

```
x, y = 10, 20
```

Binds x to 10 and y to 20

Evaluation of multiple assignment statement:

All the expressions on the RHS of the = are first evaluated **before any binding happens**.

Values of the expressions are bound to the corresponding variable on the LHS.

```
x, y = 10, 20
x, y = y+1, x+1
```

x is bound to 21 and y to 11 at the end of the program

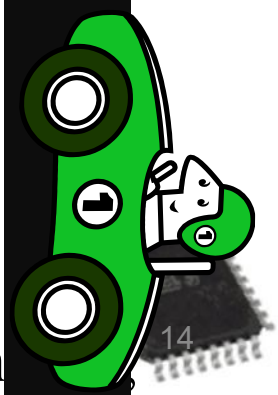
Binary Operations

Op	Meaning	Example	Remarks
+	Addition	9+2 is 11	
		9.1+2.0 is 11.1	
-	Subtraction	9-2 is 7	
		9.1-2.0 is 7.1	
*	Multiplication	9*2 is 18	
		9.1*2.0 is 18.2	
/	Division	9/2 is 4.25	In Python3
		9.1/2.0 is 4.55	Real div.
//	Integer Division	9//2 is 4	
%	Remainder	9%2 is 1	

Conditional Statements

In daily routine

- If it is very hot, I will skip exercise.
- If there is a quiz tomorrow, I will first study and then sleep. Otherwise I will sleep now.
- If I have to buy coffee, I will go left. Else I will go straight.

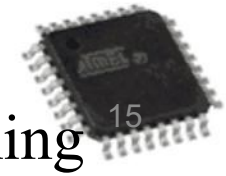


if-else statement

Compare two integers and print the min.

```
if  x < y:
        print (x)
else:
        print (y)
print ('is the minimum')
```

1. Check if x is less than y.
2. If so, print x
3. Otherwise, print y.



Elif

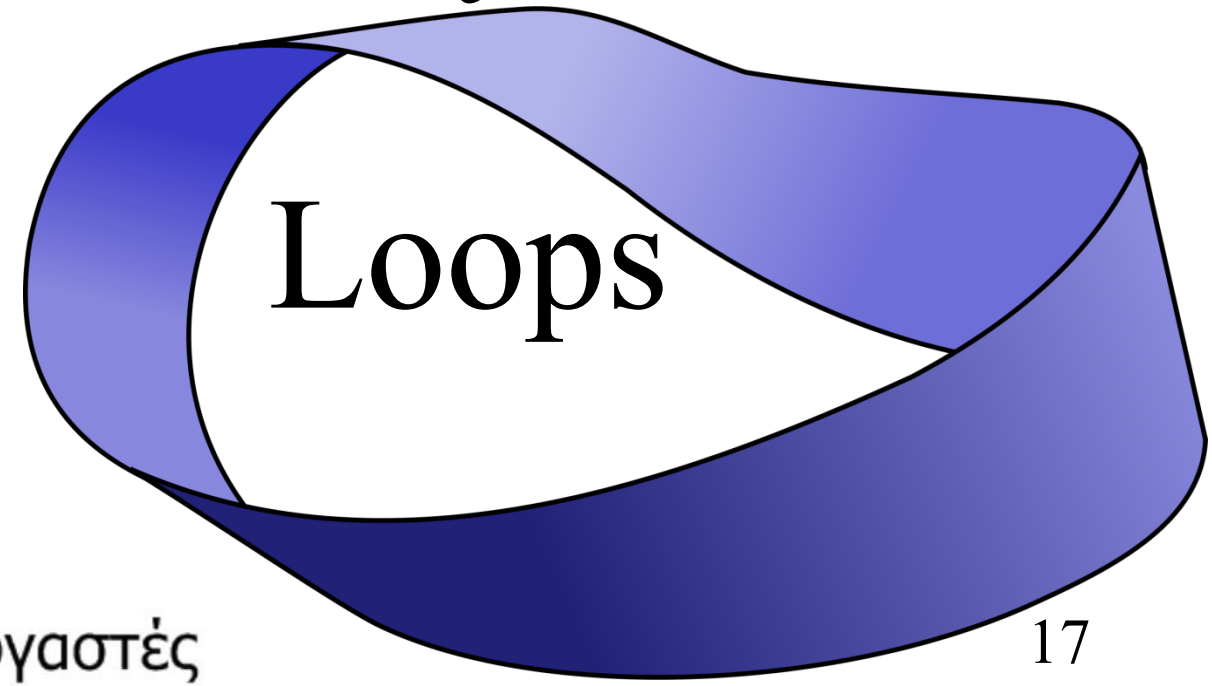
A special kind of nesting is the chain of if-else-if-else-… statements

Can be written elegantly using if-elif-..-else

```
if cond1:
    s1
else:
    if cond2:
        s2
    else:
        if cond3:
            s3
        else:
            …
```

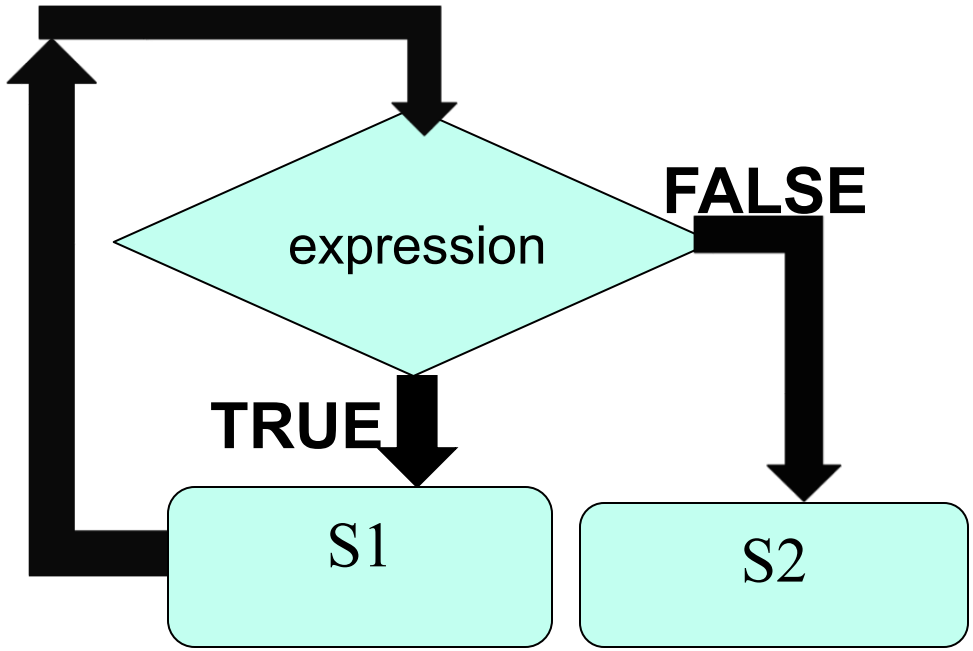
```
if cond1:
    s1
elif cond2:
    s2
elif cond3:
    s3
elif  …
else
    last-block-of-stmt
```

Programming using Python



While Statement

```
while (expression):
    S1
S2
```

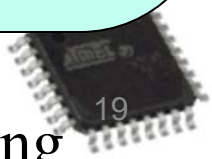


1. Evaluate expression
2. If TRUE then
   a) execute statement1
   b) goto step 1.
3. If FALSE then execute statement2.

For Loop

- Print the sum of the reciprocals of the first 100 natural numbers.

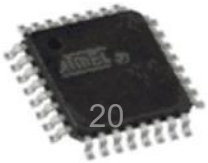
```
rsum=0.0# the reciprocal sum

# the for loop
for i in range(1,101):
    rsum = rsum + 1.0/i
print ('sum is', rsum)
```



For loop in Python

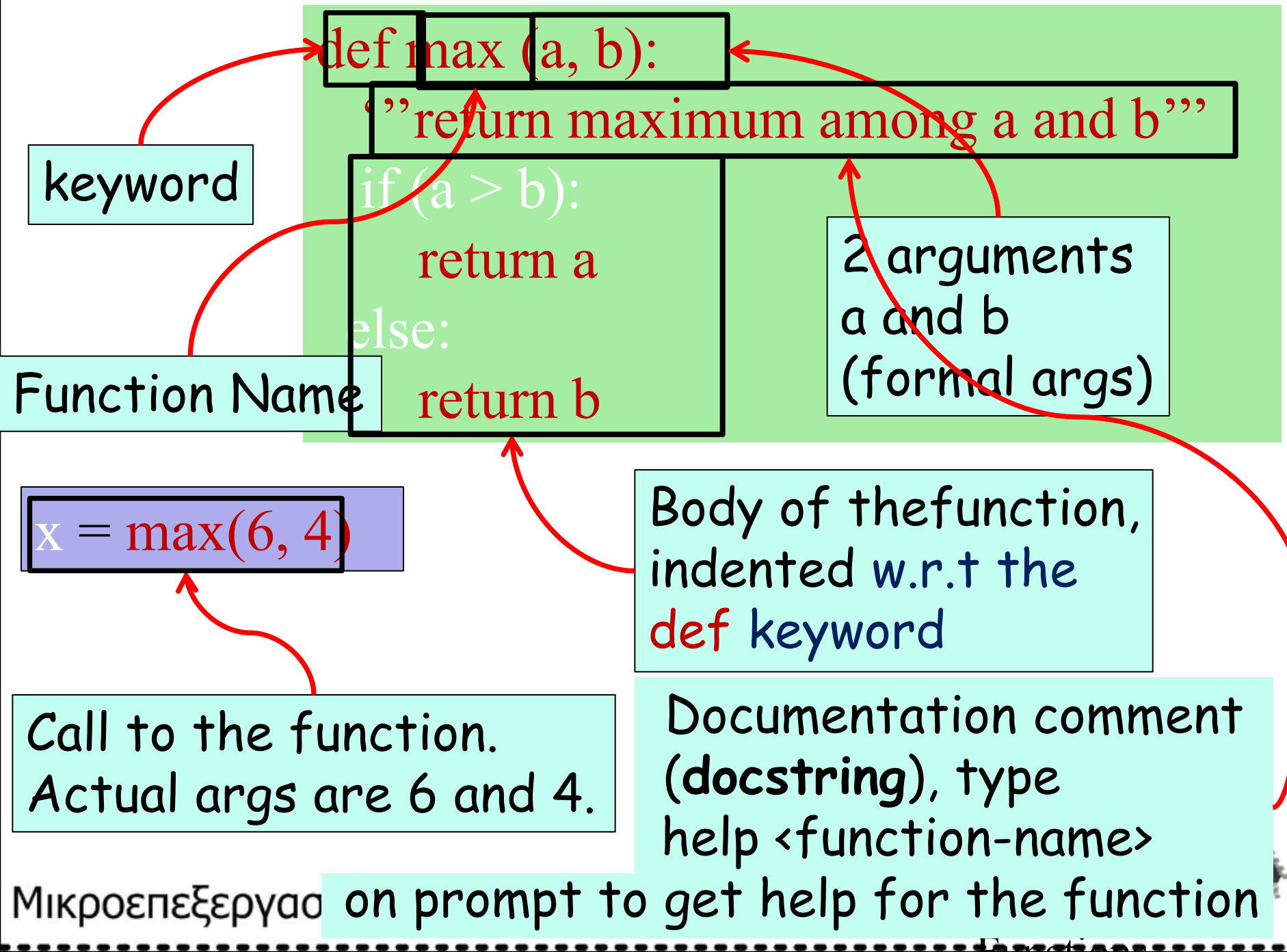
General form

```
for variable in sequence:
    stmt
```



Programming using Python

f(unctions)


def max (a, b):
''' return maximum among a and b '''
if (a > b):
return a
else:
return b
keyword
Function Name
2 arguments
a and b
(formal args)
x = max(6, 4)
Body of thefunction,
indented w.r.t the
def keyword
Call to the function.
Actual args are 6 and 4.
Documentation comment
(docstring), type
help <function-name>
on prompt to get help for the function

Globals

- Globals allow functions to communicate with each other indirectly
  - Without parameter passing/return value
- Convenient when two seemingly "far-apart" functions want to share data
  - No *direct* caller/callee relation
- If a function has to update a global, it must re-declare the global variable with global keyword.

Globals

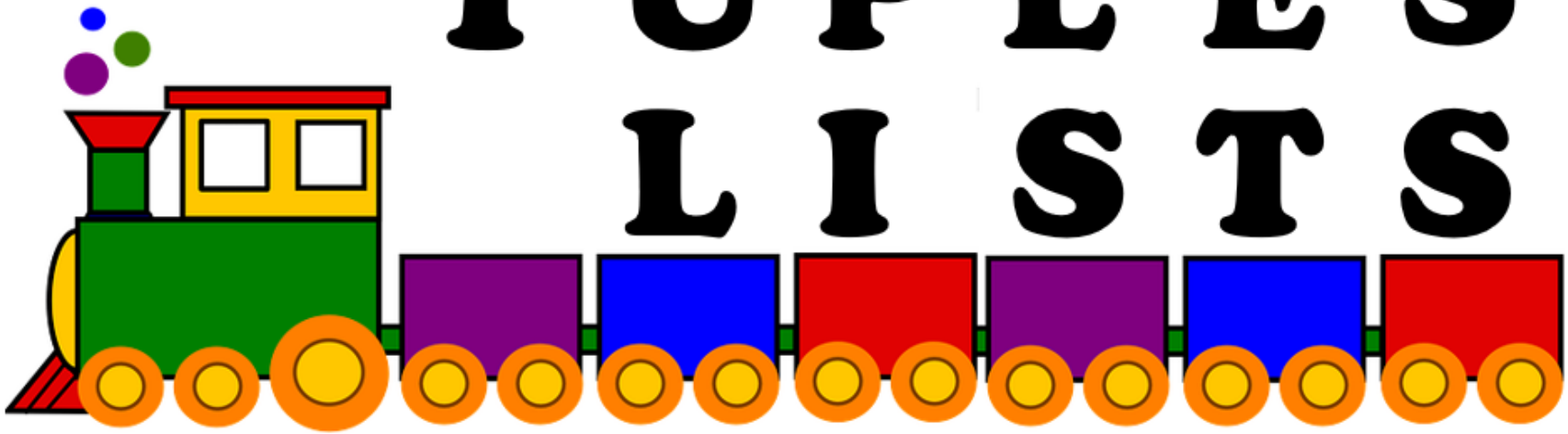
```
PI = 3.14
def perimeter(r):
    return 2 * PI * r
def area(r):
    return PI * r * r
def update_pi():
    global PI
    PI = 3.14159
```

```
>>> print(area (100))
31400.0
>>> print(perimeter(10))
62.800000000000004
>>> update_pi()
>>> print(area(100))
31415.999999999996
>>> print(perimeter(10))
62.832
```

defines PI to be of float type with value 3.14. PI can be used across functions. Any change to PI in update_pi will be visible to all due to the use of global.

Programming with Python

STRINGS
TUPLES
LISTS



Strings

```
>>> name='intro to python'
>>> descr='acad\'s first course'
>>> name
'intro to python'
>>> descr
"acad's first course"
```

- More readable when print is used

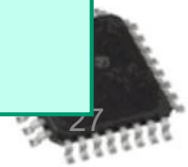
```
>>> print descr
acad's first course
```

Length of a String

len function gives the length of a string

```
>>> name='intro to python'
>>> empty=''
>>> single='a'
>>> len(name)
15
>>> len(single)
1
>>> len(empty)
0
>>> special='1\n2'
>>> len(special)
3
```

\n is a **single** character: the special character representing newline



Concatenate and Repeat

In Python, + and * operations have special meaning when operating on strings

- + is used for concatenation of (two) strings
- * is used to repeat a string, an int number of time
- Function/Operator Overloading

Indexing

Negative indices start counting from the right

Negatives indices start from -1

```
>>> name='Acads'
>>> name[-1]
's'
>>> name[-5]
'A'
>>> name[-2]
'd'
```

d last, ...

Slicing

To obtain a substring

s[start:end] means substring of s starting at index start and ending at index end-1

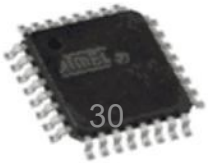
s[0:len(s)] is same as s

Both start and end are optional

If start is omitted, it defaults to 0

If end is omitted, it defaults to the length of string

s[:] is same as s[0:len(s)], that is same as s



Tuples

A tuple consists of a number of values separated by commas

```
>>> t = 'intro to python', 'amey karkare', 101
>>> t[0]
'intro to python'
>>> t[2]
101
>>> t
('intro to python', 'amey karkare', 101)
>>> type(t)
<type 'tuple'>
```

items = ("ESP8266", "DHT22", "DS18Bitems = ("ESP8266", "DHT22", "DS18B20")
result = ", ".join(items)
print(result)

Lists

Ordered sequence of values

Written as a sequence of comma-separated values between square brackets

Values can be of different types

usually the items all have the same type

```
>>> lst = [1,2,3,4,5]
>>> lst
[1, 2, 3, 4, 5]
>>> type(lst)
<type 'list'>
```

More Operations on Lists

L.append(x)

L.extend(seq)

L.insert(i, x)

L.remove(x)

L.pop(i)

- L.pop()
- L.index(x)
- L.count(x)
- L.**sort**()
- L.**reverse**()

x is any value, seq is a sequence value (list, string, tuple, …),
i is an integer value

Mutable and Immutable Types

Tuples and List types look very similar

However, there is one major difference: Lists are **mutable**

- Contents of a list can be modified

Tuples and Strings are **immutable**

- Contents can not be modified

Dictionaries

Unordered set of *key:value* pairs,

Keys have to be unique and immutable

Key:value pairs enclosed inside curly braces {...}

Empty dictionary is created by writing {}

Dictionaries are mutable

- add new key:value pairs,
- change the pairing
- delete a key (and associated value)

Operations on Dictionaries

Operation	Meaning
len(d)	Number of key:value pairs in d
d.keys()	List containing the keys in d
d.values()	List containing the values in d
k in d	True if key k is in d
d[k]	Value associated with key k in d
d.get(k, v)	If k is present in d, then d[k] else v
d[k] = v	Map the value v to key k in d (replace d[k] if present)
del d[k]	Remove key k (and associated value) from d
for k in d	Iterate over the keys in d

Importing ALL Functions

To import *all* functions from a module, in the current symbol table

```
>>> from fib import *
>>> fib_upto(6)
[0, 1, 1, 2, 3, 5]
>>> fib_iter(8)
21
```

This imports all names except those beginning with an underscore (_).

Beginners Start

```
import random
def main():
    print("Hello, world!")


if __name__ == "__main__":
    try:
        main()
    except Exception as e:
        print(f"An error occurred: {e}")
```

JSON package

JSON is an open standard file and data interchange format that uses human-readable text to store and transmit data objects consisting of attribute-value pairs and arrays.

It is a common data format with diverse uses in electronic data interchange, including web applications communicating with servers.

JSON (Java Script Object Notation) is a popular data format used for representing structured data. It's common to transmit and receive data between a server and web application in JSON format.

In Python, JSON exists as a string.

For example: p = '{"name": "Boby", "languages": ["Python", "Java"]}' It's also common to store a JSON object in a file.

JSON package

Import json Module

To work with JSON (string, or file containing JSON object), you can use Python's json module.

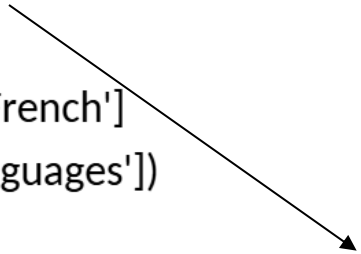
You need to import the module before you can use it: import json

The json module makes it easy to parse JSON strings and files containing JSON object

JSON package

You can parse a JSON string using json.loads() method. The method returns a dictionary.

```
import json

person = '{"name": "Boby", "languages": ["English", "French"]}'
person_dict = json.loads(person)

# Output: {'name': 'Boby', 'languages': ['English', 'French']}
print( person_dict)

# Output: ['English', 'French']
print(person_dict['languages'])
```



{'name': 'Boby', 'languages': ['English', 'French']}

JSON package

Convert dictionary to JSON string

```
person_dict = {'name': 'Boby', 'age': 12, 'children': None}
person_json = json.dumps(person_dict)

# Output: {"name": "Boby", "age": 12, "children": null}
print(person_json)
```

Exercise 1
Generate random sensory measurements

```
import random
def generate_sensor_values():
    """
    Generate random sensor values:
    - temperature: float value
    - humidity: float value
    - soil_moisture: integer value from 0 to 1024
    """
    temperature = random.uniform(15.0, 40.0)     # Celsius
    humidity = random.uniform(20.0, 90.0)        # Percentage
    soil_moisture = random.randint(0, 1024)      # Analog ADC value
    return {
        "temperature": round(temperature, 2),
        "humidity": round(humidity, 2),
        "soil_moisture": soil_moisture
    }
def main():
    values = generate_sensor_values()
    print("Random sensor values:")
    print(f"Temperature: {values['temperature']} °C")
    print(f"Humidity: {values['humidity']} %")
    print(f"Soil moisture: {values['soil_moisture']} / 1024")
if __name__ == "__main__":
    try:
        main()
    except Exception as e:
        print(f"Error: {e}")
```

Exercise 2

Generate ThingBoard JSON payload of random variables

```
import random
import json
from datetime import datetime
DEVICE_KEY = "ESP8266_GREENHOUSE_001"
def generate_thingsboard_payload():
    temperature = random.uniform(15.0, 40.0)
    humidity = random.uniform(20.0, 90.0)
    soil_moisture = random.randint(0, 1024)
    # Local system time
    local_time = datetime.now()
    # Timestamp in milliseconds
    timestamp_ms = int(local_time.timestamp() * 1000)
    payload = {
        "device_key": DEVICE_KEY,
        "ts": timestamp_ms,
        #"local_time": local_time.strftime("%Y-%m-%d %H:%M:%S"),
        "values": {
            "temperature": round(temperature, 2),
            "humidity": round(humidity, 2),
            "soil_moisture": soil_moisture
        }
    }
    return payload
def main():
    payload = generate_thingsboard_payload()
    json_string = json.dumps(payload)
    print(json_string)
if __name__ == "__main__":
    try:
        main()
    except Exception as e:
        print(f"Error: {e}")
```

2. IoT and MCU-MPU architectures

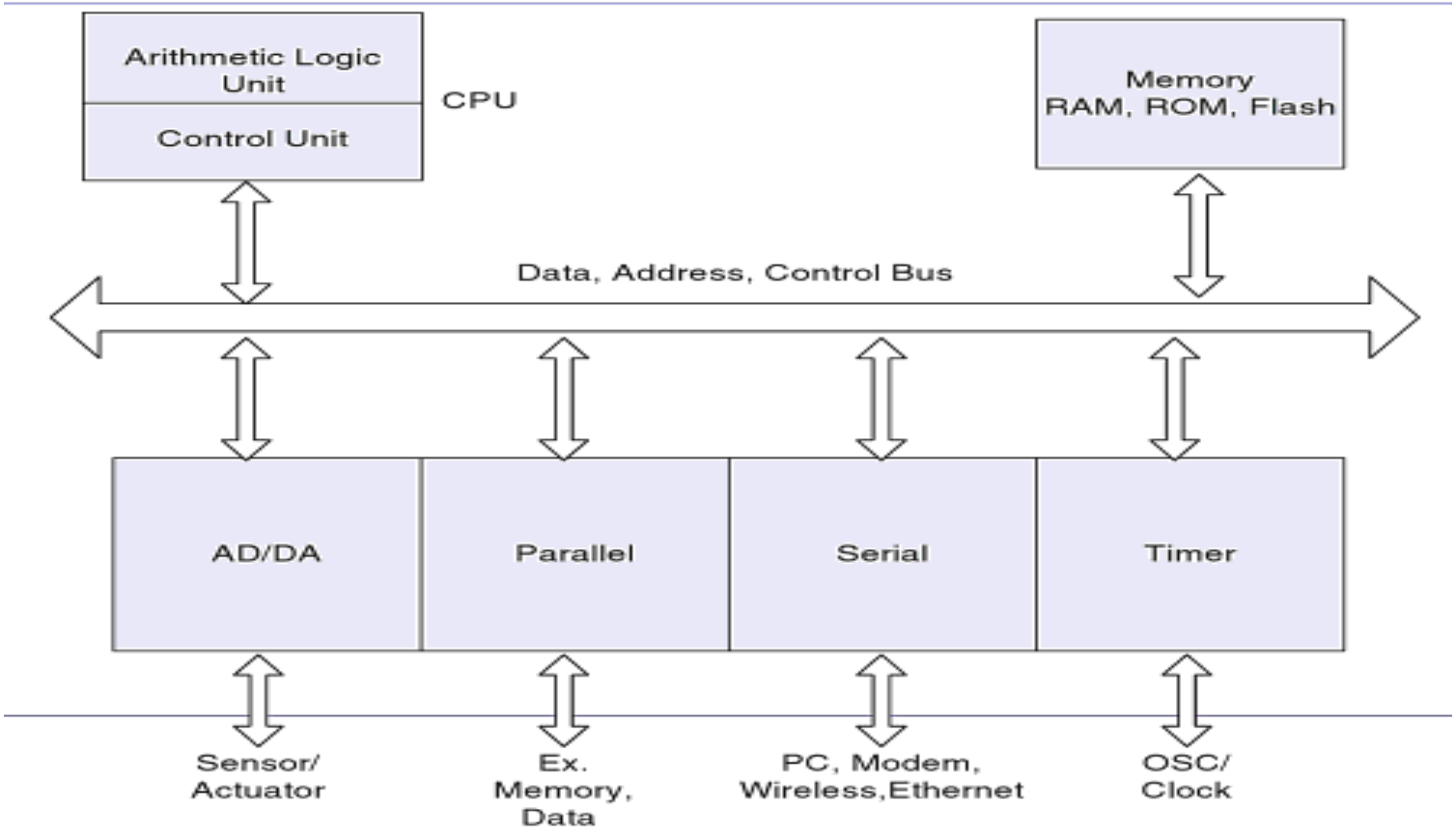
Internet of Things

IoT is a distributed infrastructure of interconnected physical entities, systems, and information resources, together with intelligent services that can process and react to information from both the physical and virtual worlds and influence activities in the physical world.

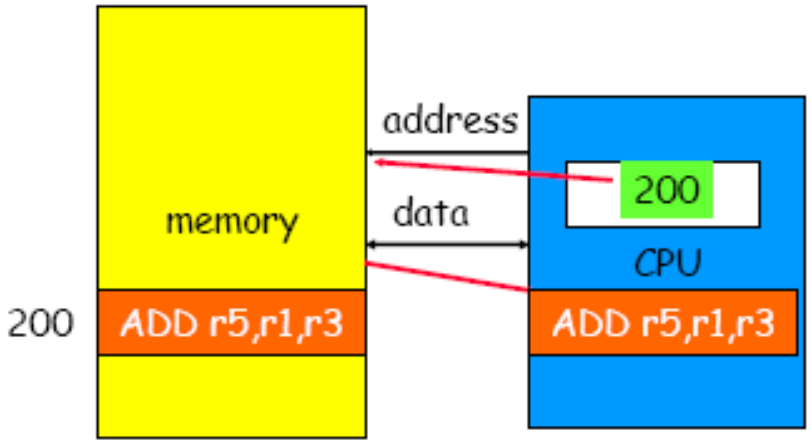
Characteristics

Grouping	1st Level
IoT System Characteristics	Auto-configuration/Autonomous operation
	Low power
	Highly distributed systems (probably autonomous systems)
	Network communication
	Low cost
	Real-time capability
	Security (Authentication-Encryption)
	Service subscription (Telemetry protocols)

Micro Controller/Processing Unit



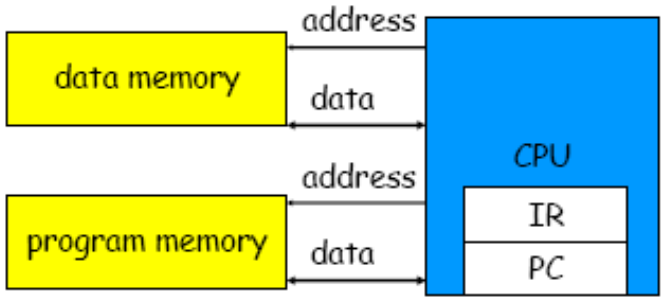
von Neumann



von Neumann vs. Harvard

- von Neumann
  - Same memory holds data, instructions.
  - A single set of address/data buses between CPU and memory
- Harvard
  - Separate memories for data and instructions.
  - Two sets of address/data buses between CPU and memory

Harvard Architecture



von Neumann vs. Harvard

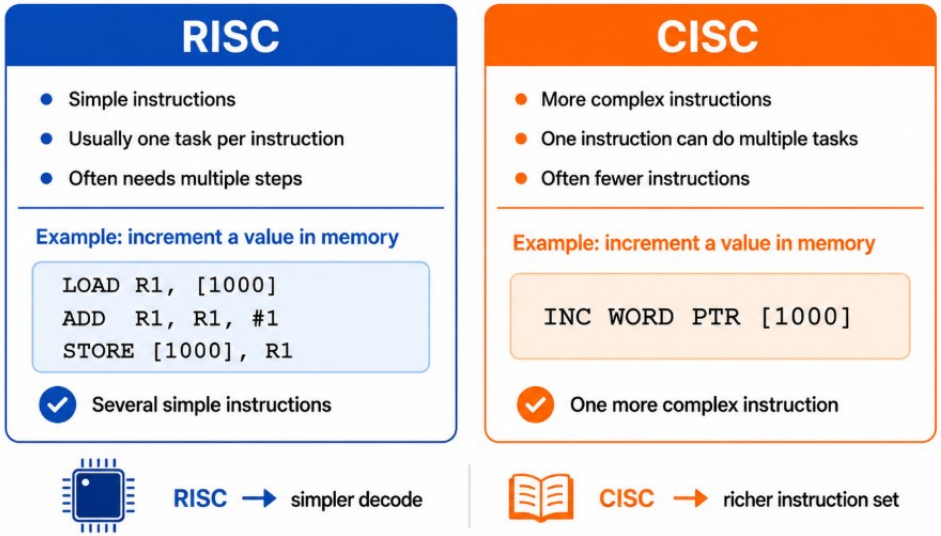
- Harvard allows two simultaneous memory fetches.
- Most DSPs use Harvard architecture for streaming data:
  - greater memory bandwidth;
  - more predictable bandwidth.

RISC vs. CISC

- Reduced Instruction Set Computer (RISC)
  - Compact, uniform instructions → facilitate pipelining
  - More lines of code → large memory footprint
  - Allow effective compiler optimization
- Complex Instruction Set Computer (CISC)
  - Many addressing modes and long instructions
  - High code density
  - Often require manual optimization of assembly code for embedded systems

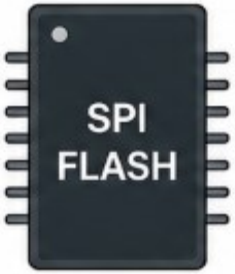
Microprocessors

	von Neumann	Harvard
RISC	ARM7	ARM9
CISC	Pentium	SHARC (DSP)



ESP32 Code Execution and Memory Use

SPI Flash



- Stores most program code and firmware

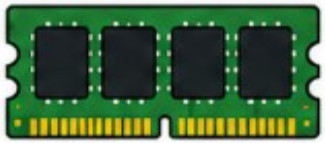
Code / Instructions

CPU

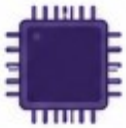
- Runs the program

Read / Write Data

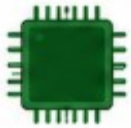
RAM



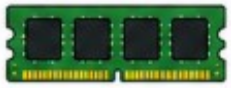
- Stores variables, stack, heap, and buffers while running



SPI Flash
(Non-volatile)



CPU
(Executes code)



RAM
(Working memory)

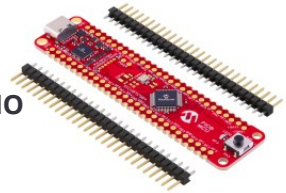
Most ESP32 code lives in SPI Flash and uses RAM as working memory.

What is ARM and PIC/AVR?

Low power Microcontrollers-Processors Using **RISC and Harvard** architecture

1. Peripheral Interface Controller (PIC) (RISC) is microcontroller developed by Microchip (Microchip MPLAB)

PIC18F57Q84 CURIOSITY NANO



ATTiny 85

2. AVR microcontroller was developed in the year of 1996 by Atmel Corporation as well as ATMega microprocessors (ATSAMD) (RISC) – **Arduino IDE**

ATmega4809 8-bit

Arduino UNO Wi-Fi R4 (ESP32)

Arduino Q

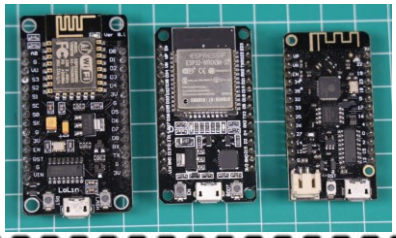
Qualcomm Dragonwing ARM+

STM32 ARM M3

3. An ARM microprocessor is also one of a family of CPUs based on the RISC (reduced instruction set computer) architecture developed by Advanced RISC Machines (ARM) (M0, M3, ESP).

(**STM32CubeIDE PlatformIO**)

Μικροεπεξεργαστές



ARM families

ARM family	ARM architecture
ARM7	ARM v4
ARM9	ARM v5
ARM11	ARM v6
Cortex-A	ARM v7-A
Cortex-R	ARM v7-R
Cortex-M	ARM v7-M

ARMv9 architecture has been added for Cortex A-M. From cortex-A all models of ARM are now 64bit

A-profile (Application): Offers high performance for complex applications like PCs, laptops, and servers (RPi).

R-profile (Real-Time): Optimized for real-time systems requiring deterministic responses, such as medical devices and automotive controls (RPi pico).

M-profile (Microcontroller): Focused on low power consumption and size, suitable for IoT devices and wearables (ESP32-ESP8266)

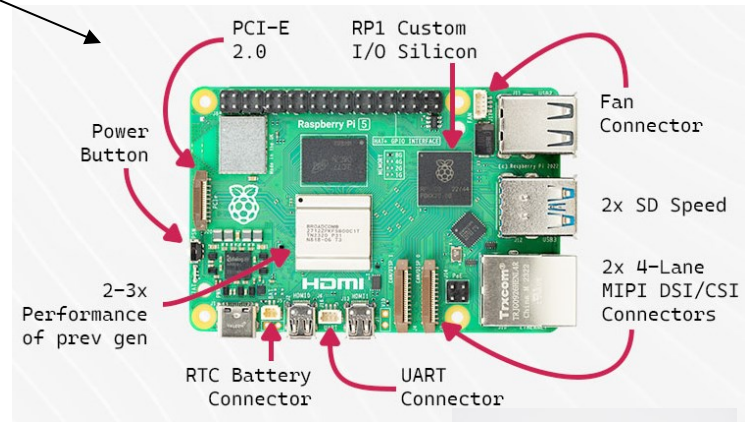
ARM (RISC processors)

RPi pico

Dual-core Arm Cortex-M0+
running at up to 133 MHz
264KB on-chip SRAM and 2MB
on-board QSPI flash memory

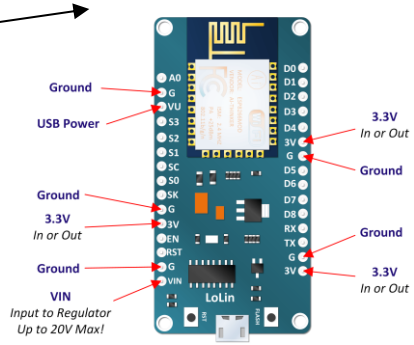
RPi 4/5

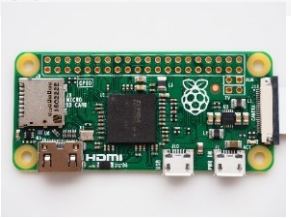
Broadcom BCM2712, a 64-bit quad-core Arm Cortex-A76 running at 2.4GHz
4GB or 8GB or 16GB LPDDR4X-4267 SDRAM
BLE/WiFi Gigabit Ethernet
800MHz VideoCore 4K video output



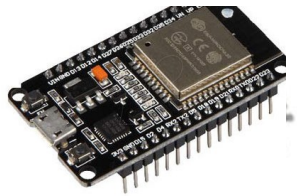
ESP32

Tensilica LX106 32-bit RISC CPU
CPU Clock 80-160 MHz
WiFi-BLE
RAM 64-128KB
Flash Memory: 4MB





Tensilica LX106 32-bit RISC CPU dual core
CPU Clock 160-240 MHz
WiFi-BLE
RAM 520KB
Flash Memory: 4-8MB



Power Consumption & Estimated Hours of Operation

Power available: 2W solar panel, 8h full sunlight per day | Battery: 18650 Li-ion 4000mAh (3.7V, ~14.8Wh usable)

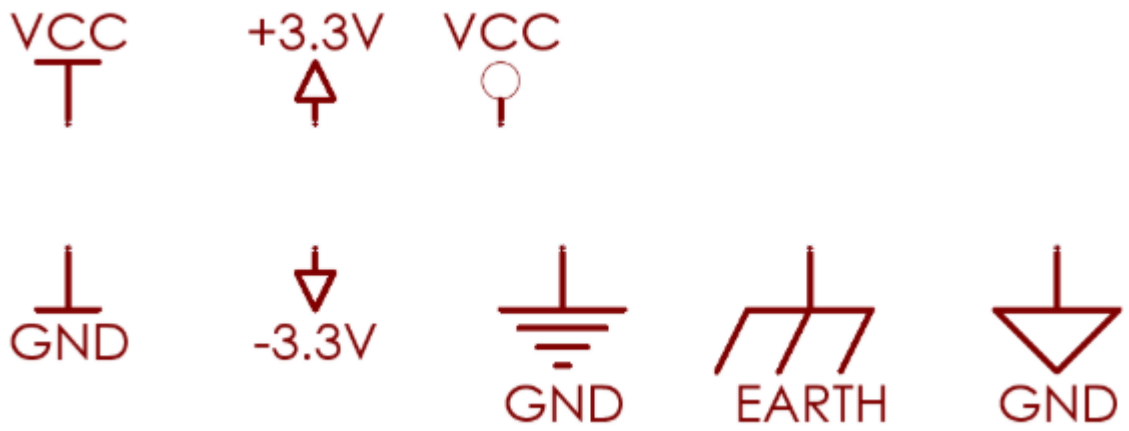
Assumptions: • 2W × 8h × 0.75 (system losses) = ~12Wh energy harvested per day • Devices powered at 3.3V or 5V as typical

Device (Main MCU / Board)	Typical Active Power (Fully On) (mW)	Typical Sleep Power (Deep/Light Sleep) (mW)	Total Power with OLED (Active) (mW)	Total Power with OLED (Sleep) (mW)	Hours of Operation (Always On) With 2W Panel + 4000mAh Battery (~12Wh/day)	Hours of Operation (5% Active / 95% Sleep) With 2W Panel + 4000mAh Battery (~12Wh/day)
ESP8266 (NodeMCU) 	80 – 170 mW	0.3 – 1.0 mW	110 – 200 mW	0.6 – 1.3 mW	~60 – 110 hours (2.5 – 4.6 days)	~4,400 – 7,600 hours (183 – 317 days)
ESP32 (DevKitC) 	120 – 240 mW	1 – 5 mW	150 – 270 mW	1.5 – 5.5 mW	~44 – 80 hours (1.8 – 3.3 days)	~2,800 – 5,000 hours (117 – 208 days)
Arduino Uno (ATmega328P) 	45 – 70 mW	0.1 – 0.3 mW	75 – 100 mW	0.7 – 1.0 mW	~120 – 190 hours (5 – 7.9 days)	~7,600 – 12,500 hours (317 – 521 days)
Arduino Uno WiFi (ATmega4809 + ESP8266) 	120 – 180 mW	0.6 – 1.5 mW	150 – 210 mW	1.1 – 1.8 mW	~57 – 85 hours (2.4 – 3.5 days)	~4,200 – 6,600 hours (175 – 275 days)
Arduino UNO Rev4 (Microchip RA4M1) 	70 – 110 mW	0.2 – 0.5 mW	100 – 140 mW	0.8 – 1.3 mW	~85 – 120 hours (3.5 – 5 days)	~6,000 – 9,000 hours (250 – 375 days)
Arduino GIGA R1 (ESP32-S3) 	200 – 350 mW	2 – 8 mW	230 – 380 mW	2.5 – 8.5 mW	~31 – 52 hours (1.3 – 2.2 days)	~2,000 – 3,600 hours (83 – 150 days)
Raspberry Pi Zero 2 W 	250 – 350 mW	50 – 70 mW	260 – 360 mW	55 – 75 mW	~28 – 46 hours (1.2 – 1.9 days)	~520 – 750 hours (22 – 31 days)
Raspberry Pi 4 Model B (2GB) 	800 – 1,200 mW	200 – 300 mW	810 – 1,210 mW	205 – 305 mW	~8.5 – 15 hours (0.35 – 0.62 days)	~160 – 260 hours (6.6 – 10.8 days)

Notes: • OLED used: 0.96" I2C SSD1306 (typ. 20–30 mW active, ~0.5 mW in sleep) • Sleep power varies by library, configuration, and peripherals • Actual results will vary with usage, sensors, WiFi, and environment

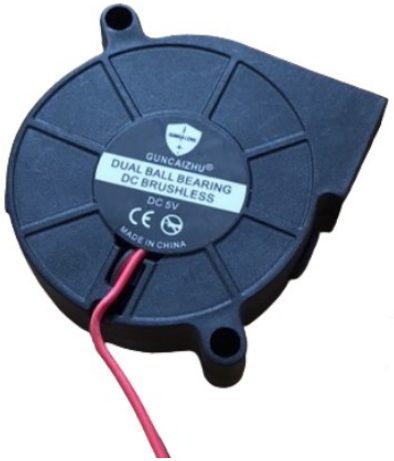
3. Electronics Basics for Sensors Interfacing Sensor Technologies

Data Inputs - Ground

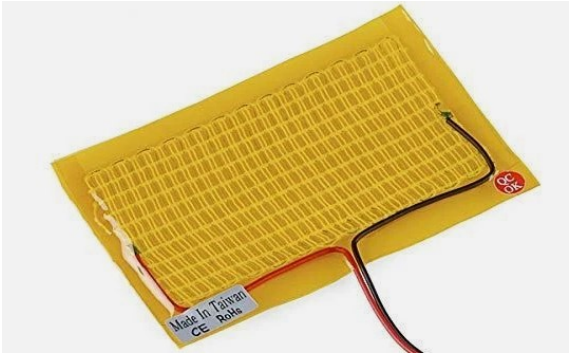


DC Input/Output Devices (HIGH=3.3V/LOW=0V)
GND = ALWAYS LOW=0V
COMMON GROUND - Keep sensors and MPU device Grounds Connected to avoid flapping
COMMON GROUND – Between different input voltage actuators

Actuators



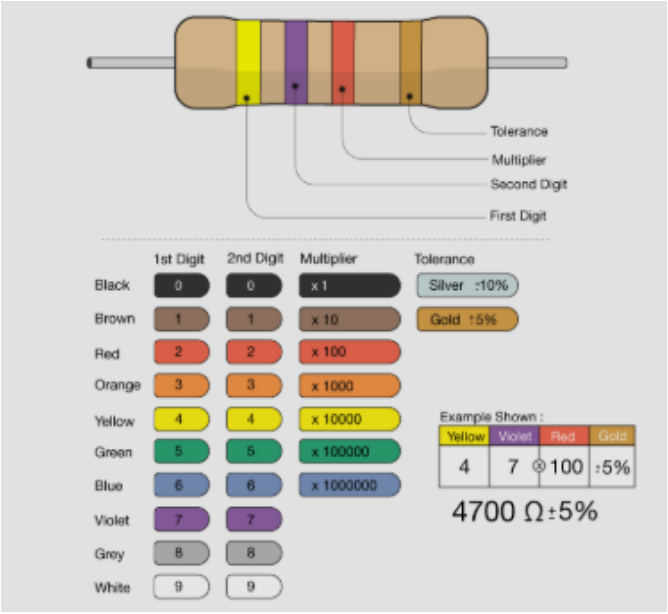
5-12V DC



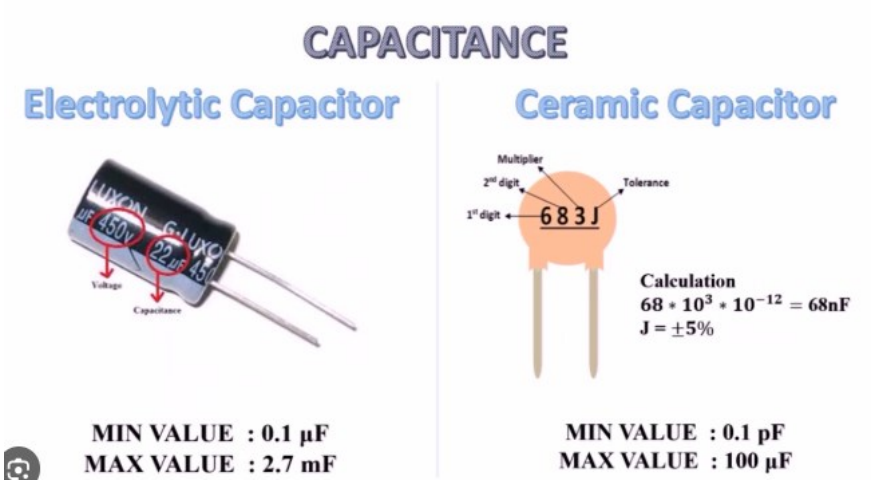
Μικροεπεξεργαστές

Electronics Components

1. Resistors

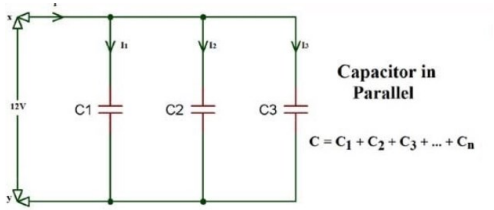


2. Capacitors



Series: R_1 R_2 R_3 = $R_{eq}=R_1+R_2+R_3$

Parallel: R_1, R_2, R_3 = $R_{eq}=(1/R_1+1/R_2+1/R_3)^{-1}$

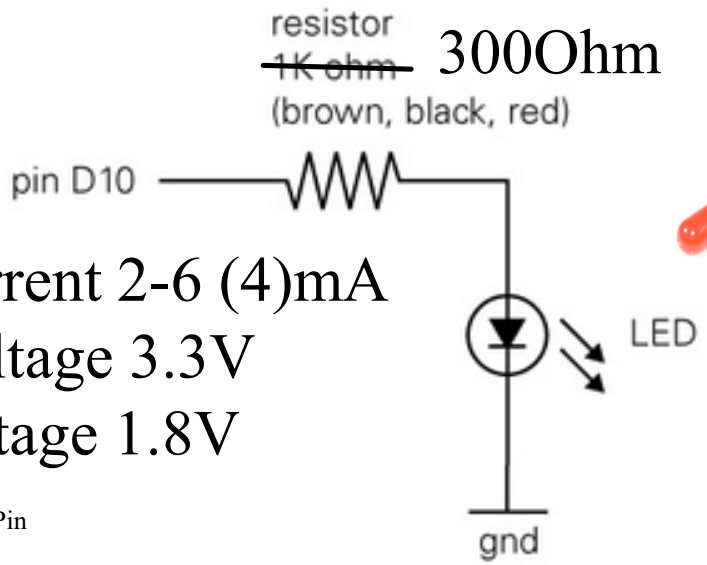


Μικροεπεξεργαστές

Electronics Basics

Connect Output LEDs –Digital Output

LED color	Typical voltage drop
Red	**1.8–2.2 V**
Orange / Yellow	**2.0–2.2 V**
Green	**2.0–3.2 V**
Blue(No R)	**3.0–3.4 V**
White(No R)	**3.0–3.4 V**
Infrared(1KΩ)	**1.1–1.5 V**



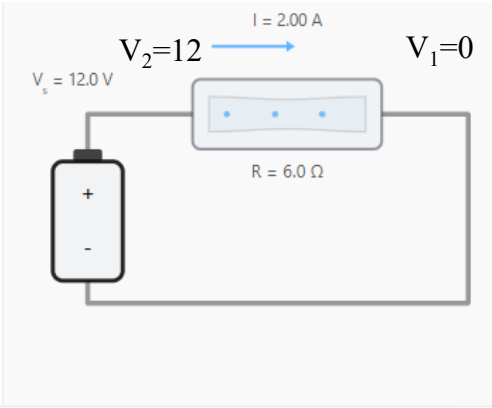
Max current 2-6 (4)mA
Max Voltage 3.3V
Min voltage 1.8V

Ohm's Law

```
from machine import Pin
import time
led = Pin(5, Pin.OUT)   # D1 = GPIO5
led = Pin(2, Pin.OUT) # D4 = GPIO2, onboard LED

while True:
    led.value(1)       # LED ON
    time.sleep(1)
    led.value(0)       # LED OFF
    time.sleep(1)
```

$$V = IR$$

$$I = \frac{V_2 - V_1}{R}$$



Μικροεπεξεργαστές

Quantity	Symbol	Unit
Power	(P)	Watt (W) (J/s)
Voltage	(V)	Volt (V)
Current	(I)	Ampere (A)
Resistance	(R)	Ohm(Ω)
Energy	(E)	Joules (Ws)

Electrical Power Law

$$E = P \cdot t$$

$$P = VI$$

Also, using Ohm's law:

$$\text{Watt-seconds (Ws)} = \text{Ah} \times V \times 3600$$

$$P = I^2R$$

$$\text{Wh} = \text{Ah} \times V$$

$$P = \frac{V^2}{R}$$

Voltage expresses the electrical potential difference between two points.

$$V = \frac{W}{Q}$$

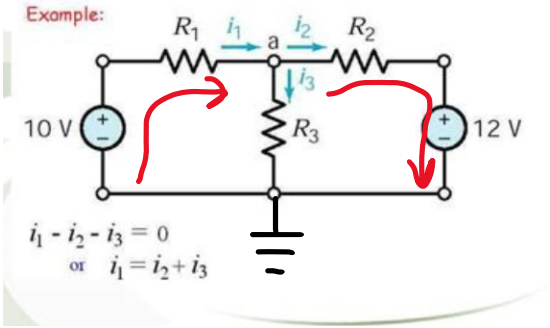
Current is the rate at which electric charge flows per second.

$$I = \frac{dQ}{dt}$$

Kirchhoff's Current Law - KCL

At any node, the total current entering equals the total current leaving.

$$\sum I_{in} = \sum I_{out}$$



if i_2 comes out negative, it means the real current through R_2 flows from the 12 V source toward node a, opposite to the arrow.

$$V_a = \frac{\frac{10}{R_1} + \frac{12}{R_2}}{\frac{1}{R_1} + \frac{1}{R_2} + \frac{1}{R_3}}$$

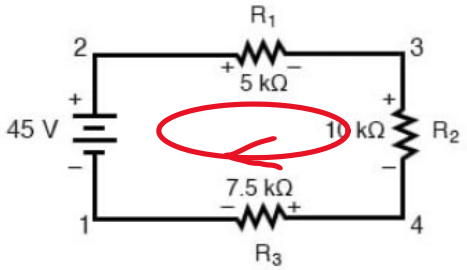
$$i_1 = \frac{10 - V_a}{R_1}$$

$$i_2 = \frac{V_a - 12}{R_2}$$

$$i_3 = \frac{V_a}{R_3}$$

Kirchhoff's Voltage Law - KVL

Around any closed loop, the sum of voltages is zero. $\sum V = 0$



Using Ohm's law:

$$45 - I(5\,k\Omega) - I(10\,k\Omega) - I(7.5\,k\Omega) = 0$$

$$I = \frac{45}{22.5\,k\Omega} = 2\,mA$$

VR1=10V,VR2=20V,VR3=15V

KVL check:

$$45 - 10 - 20 - 15 = 0$$

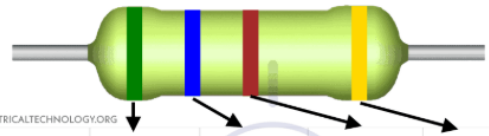
Electronics Components

3. Inductors

4 Band Inductor Color Code

1st Digit . 2nd Digit x Multiplier + Tolerance

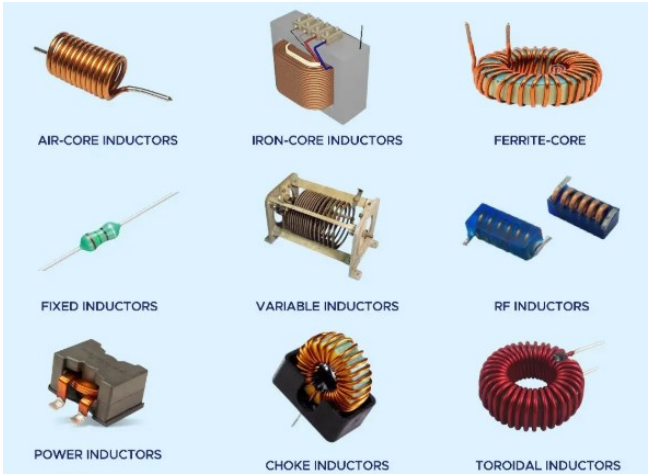
$5\ 6 \times 10^{2} = 560\ \mu H - \pm 5\%$

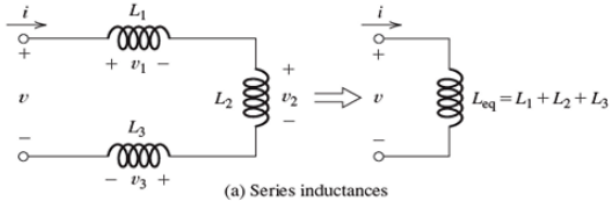


WWW.ELECTRICALTECHNOLOGY.ORG

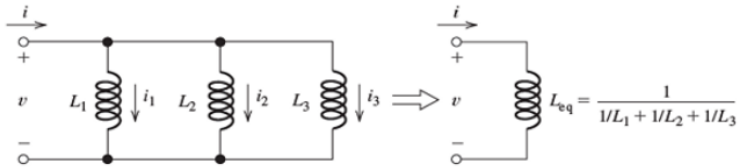
Band Color	1st Digit	2nd Digit	Multiplier	Tolerance (%)
BLACK	0	0	1	± 20%
BROWN	1	1	10	± 1%
RED	2	2	100	± 2%
ORANGE	3	3	1,000	± 3%
YELLOW	4	4	10,000	± 4%
GREEN	5	5	100,000	–
BLUE	6	6	1,000,000	–
VIOLET	7	7	–	–
GREY	8	8	–	–
WHITE	9	9	–	–
GOLD	–	–	0.1	± 5%
SILVER	–	–	0.01	± 10%

4. Other Inductors





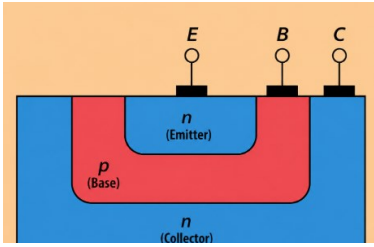
(a) Series inductances

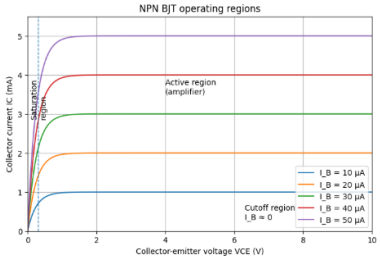


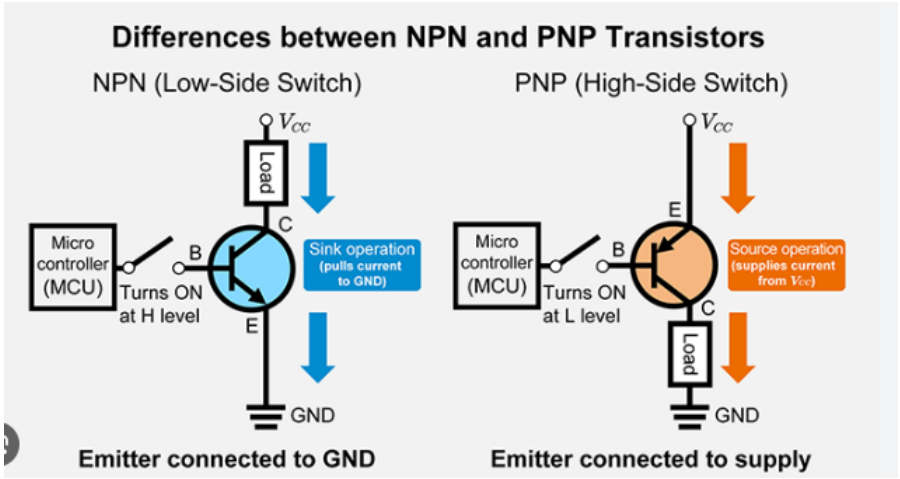
(b) Parallel inductances

Electronics Components

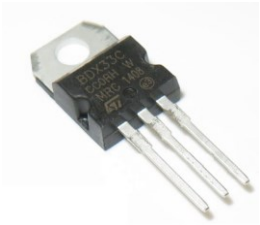
5. NPN Transistors

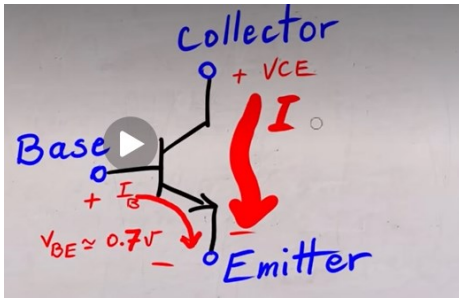


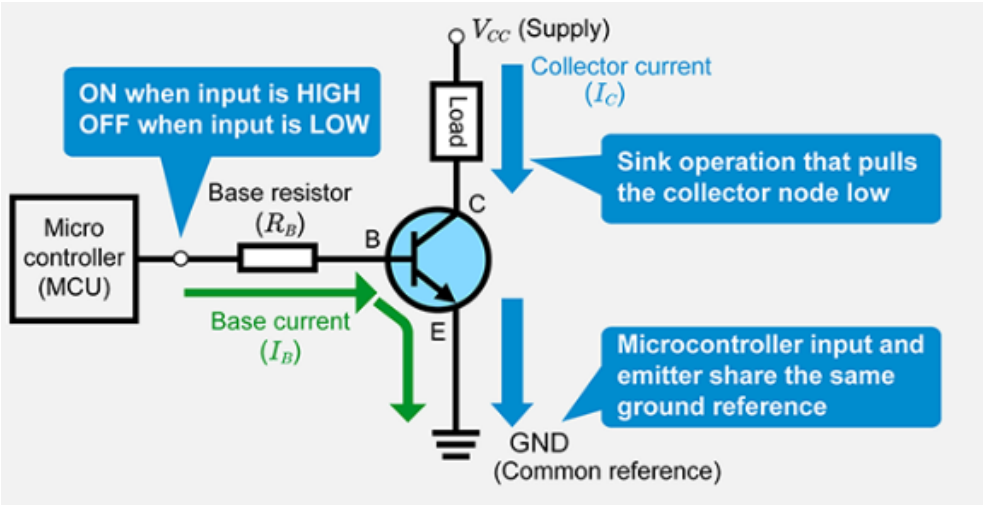


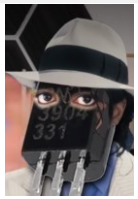


BDX33C
TIP31C BJT
TIP120 Darlington

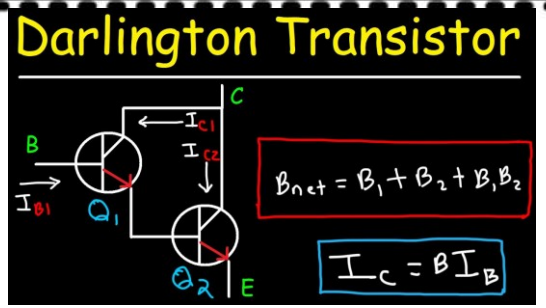




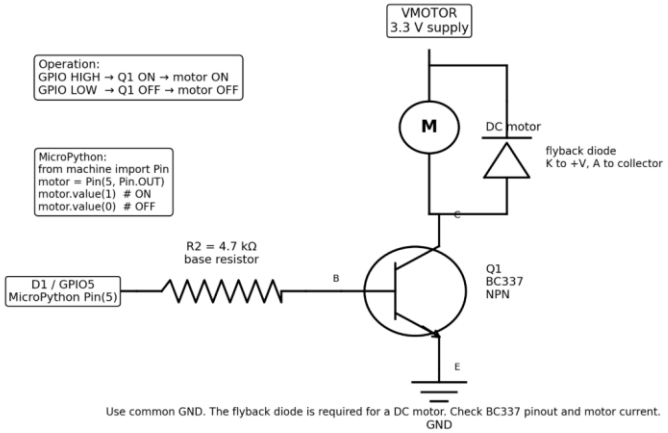


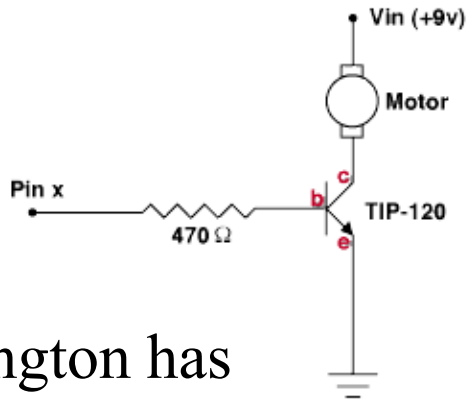


Electronics Basics



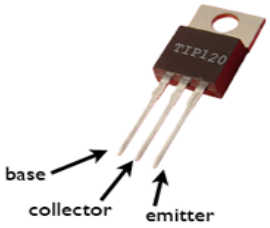
Connect NPN Transistor – Motor Drive – External DC Power

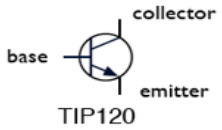




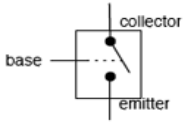
D1:1N4001..1N4009 (500mA)

Q1: TIP 120 (1-3A)





schematic symbol

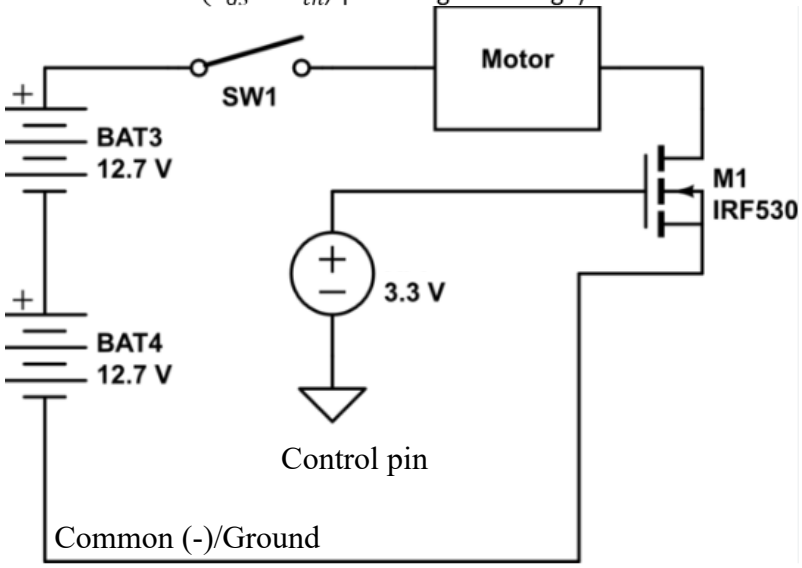


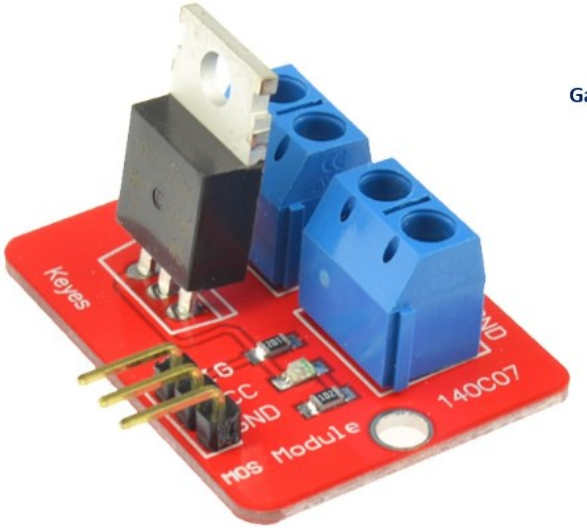
how it kind of works

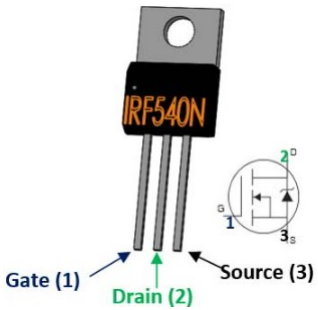
Electronics Components

6. N-MOSFETS

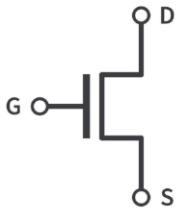
Turns on when ($V_{GS} > V_{th}$) positive gate voltage)







IRLZ44N
IRL540N

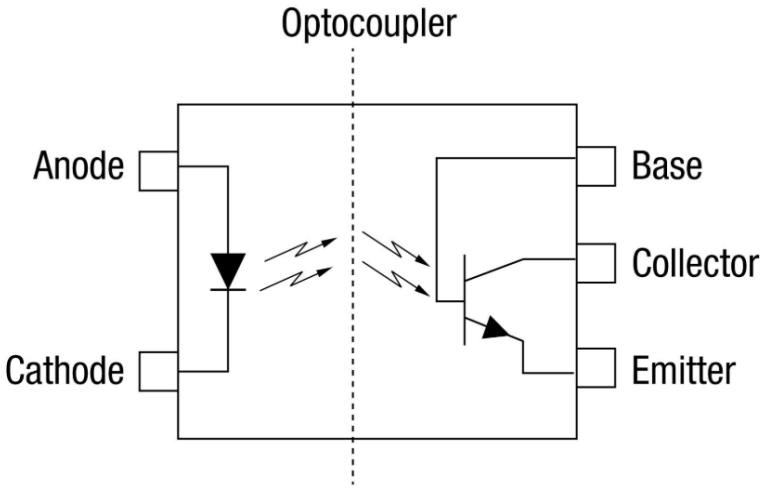


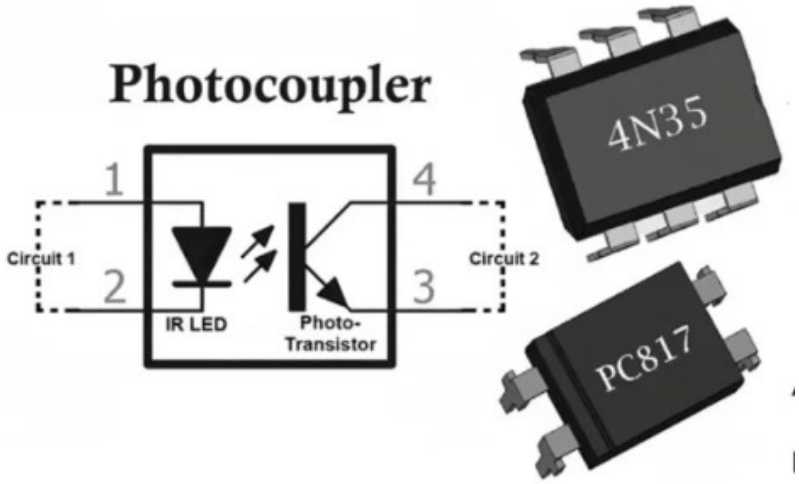
Choosing Switching MOSFET for 3.3v powered system

Μικροεπεξεργαστές IRF510 2N7000 BS170 BSS138 SI2302

Electronics Components

6. Optocouplers

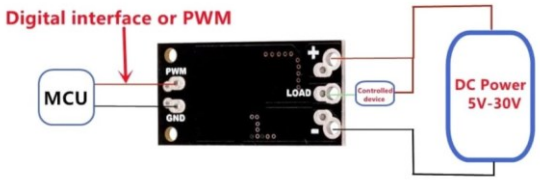




It is used in relay components (Up to 50mA Collector-Emitter current)

N-MOSFET Control Module 100V 9.4A - FR120N

N-MOSFET+Optocoupler to offer isolation

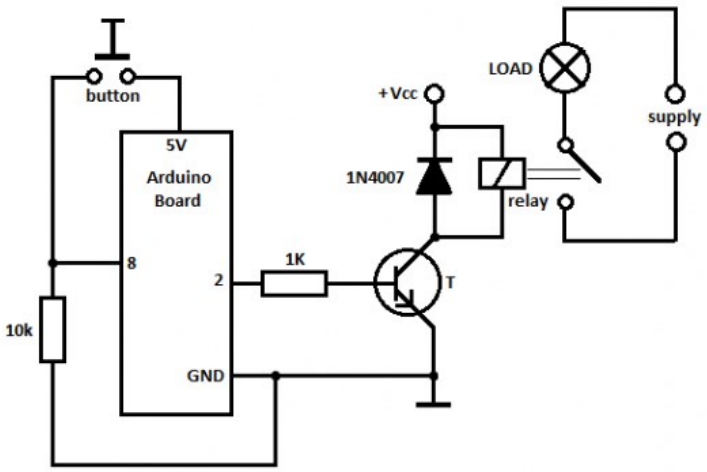


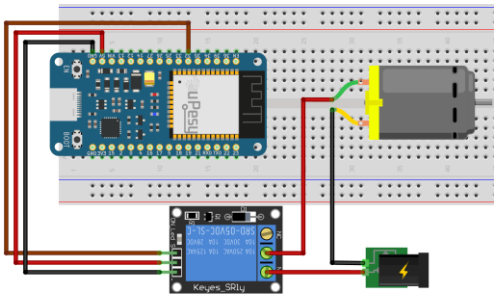
External load circuit	FR120N module
External power supply positive	+ / **DC+**
Load positive wire	+ / **DC+**
Load negative wire	**LOAD**
External power supply negative	- / **DC-**

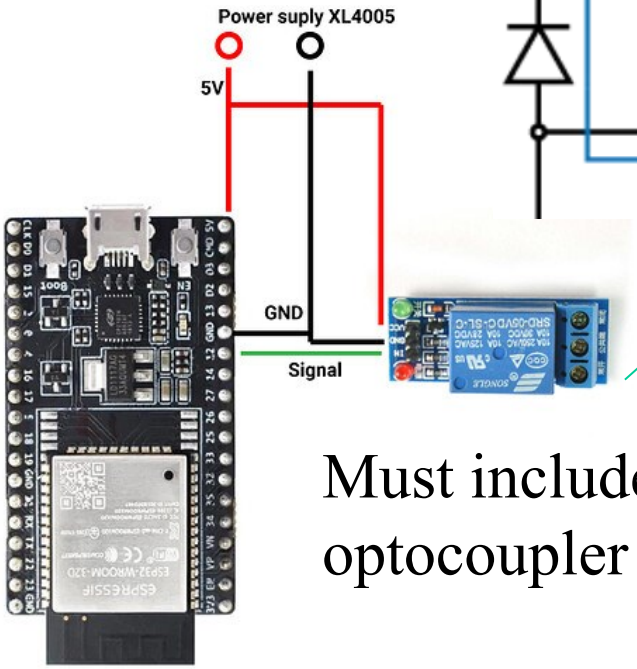
Electronics Basics

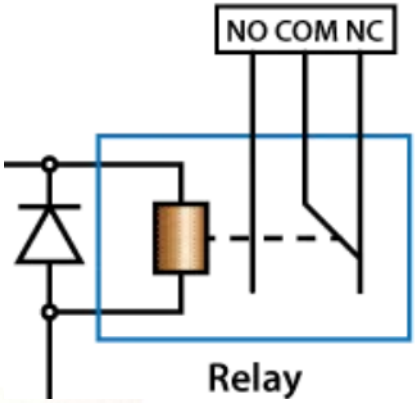
7. Connect Relay

More than 100mA
power to drive relay









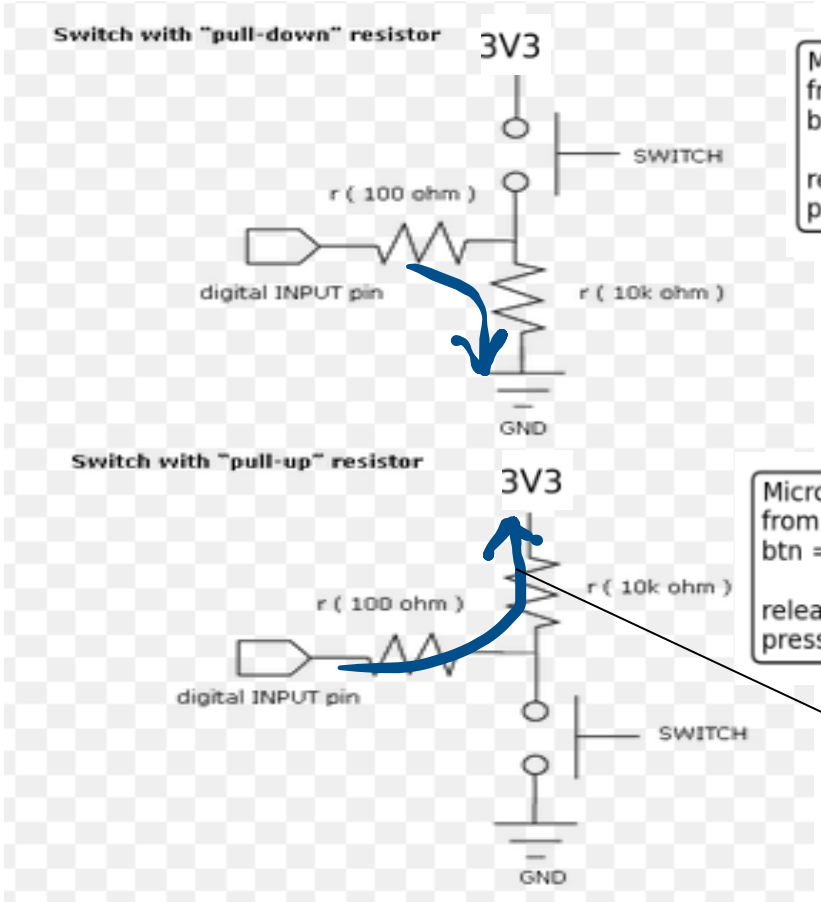
Must include
optocoupler circuit

Drivers Comparison Table

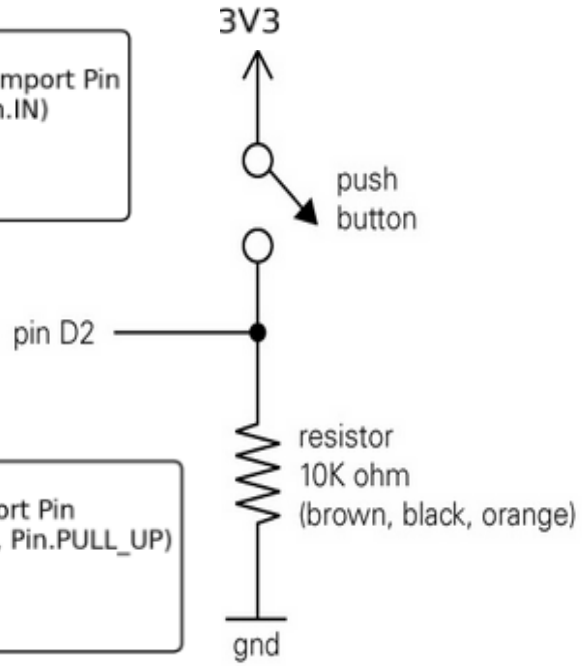
Feature	Logic-level MOSFET	NPN transistor	TIP120 Darlington	Relay
Good for 3.3 V GPIO	Yes, if logic-level	Sometimes	Yes	Sometimes, if relay module supports 3.3 V input
GPIO current needed	Very low	Medium/high	Low	Low for relay module; high for bare relay coil
Voltage loss	Very low	Medium	High	Very low at contacts
Heating	Low	Medium	High	Low at contacts; coil consumes power
Switching speed	Very fast	Fast	Fast	Slow
Good for motors/pumps	Yes	**Small ones only(0.01-0.25A)**	**Works but inefficient(0.1-3A)**	Yes, ON/OFF only
Best general choice	**Yes (DC 1-5A)**	No	No	**Yes for AC loads/ DC >5A**

Electronics Basics

Connect Digital Input Switch-Push Button



MicroPython:
```
from machine import Pin
btn = Pin(5, Pin.IN)

released = 0
pressed  = 1
```



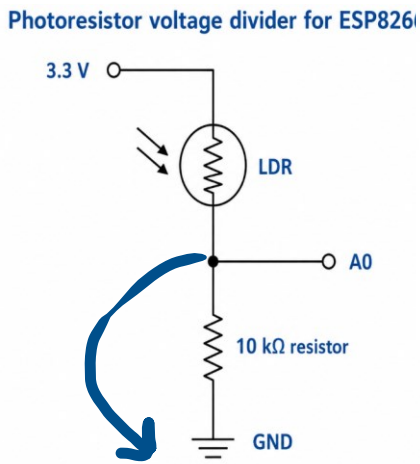
MicroPython:
```
from machine import Pin
btn = Pin(5, Pin.IN, Pin.PULL_UP)

released = 1
pressed  = 0
```

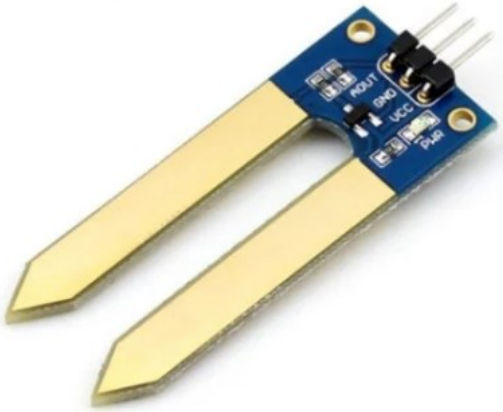
≈ 50 kΩ internal pull-up

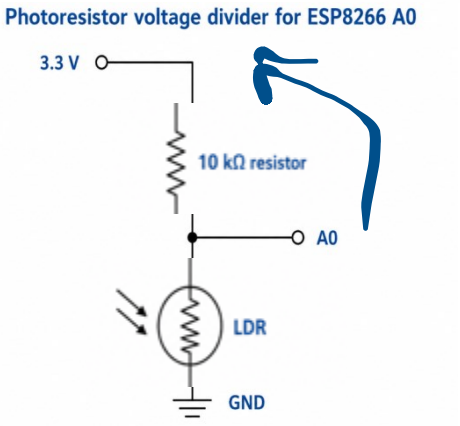
Electronics Basics

Dry soil → high resistance between probes → lower AOUT voltage
Wet soil → lower resistance between probes → higher AOUT voltage

Connect Analog Input







Light condition	LDR resistance
Strong light	~1 kΩ to 10 kΩ
Normal room light	~10 kΩ to 50 kΩ
Low light	~50 kΩ to 200 kΩ
Darkness	~500 kΩ to several MΩ

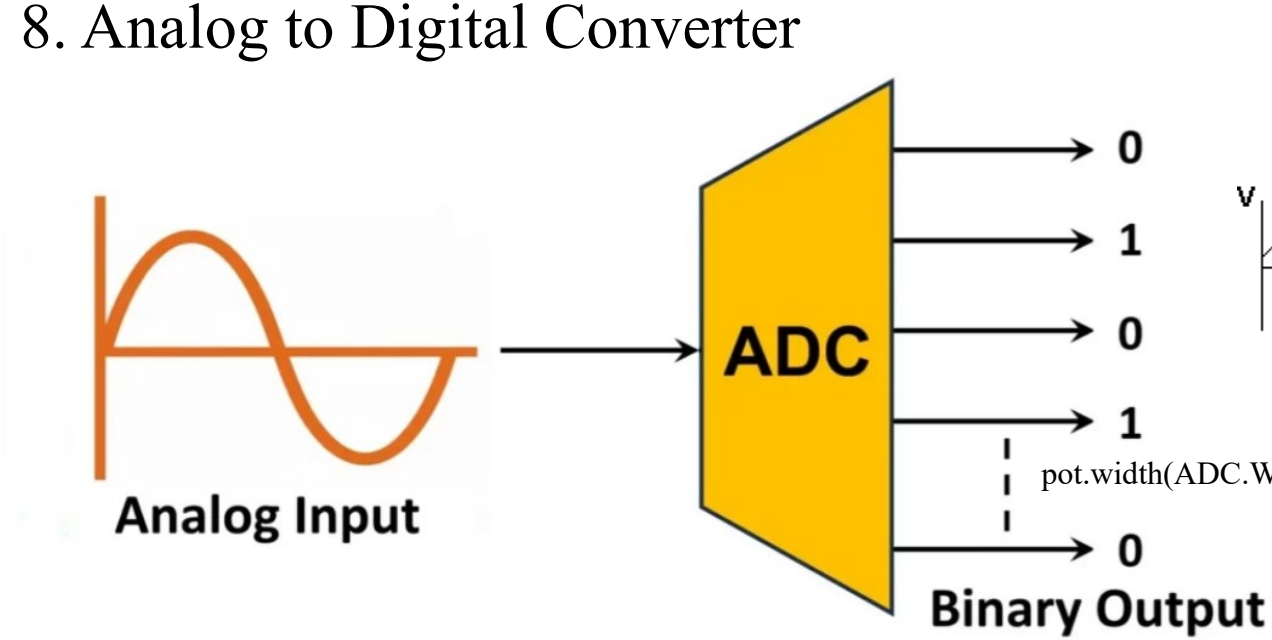
```
def map_value(x, in_min, in_max, out_min, out_max):
return int((x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min)
```

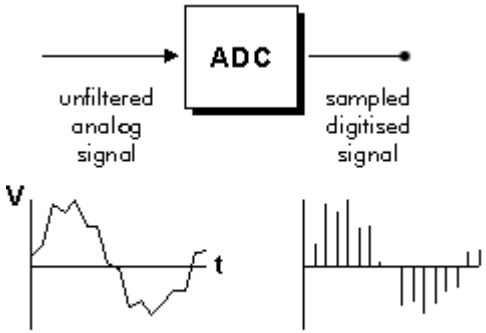
```
value10 = map_value(value12, 0, 4095, 0, 1023)
```

Electronics Basics

10bit==1024 encoded states/on board Voltage divider (0-3.3V input)

8. Analog to Digital Converter






```
pot.width(ADC.WIDTH_10BIT) # 10-bit ADC: values 0–1023
```

Analog to Digital Converter (ADC)

voltage = adc_value * 3.3 / 1023=32mV

Valid values (and approximate linear measurement ranges) are:

ESP32: pot=ADC(Pin(35)), 12-bit, 0–4095, supports attenuation
ESP8266: pot=ADC(0), 10-bit, 0–1023, no attenuation setting

- pot.atten(ADC.ATTN_0DB): No attenuation (100mV – 950mV)
- pot.atten(ADC.ATTN_2_5DB): 2.5dB attenuation (100mV – 1250mV)
- pot.atten(ADC.ATTN_6DB): 6dB attenuation (150mV – 1750mV)
- pot.atten(ADC.ATTN_11DB): 11dB attenuation (150mV – 2450mV)

Only 1 PIN A0

Μικροεπεξεργαστές

Analog photoresistive sensor

```
from machine import ADC,Pin
import time

# create an ADC object acting on a pin
photoresistor = ADC(Pin(0, Pin.IN))
# Configure the ADC attenuation to 11dB for full range
#photoresistor.atten(photoresistor.ATTN_11DB)
while True:
    # read a raw analog value in the range 0-4095
    value = photoresistor.read()
    print(value)
    time.sleep(0.05)
```

Soil Moisture Resistive sensor

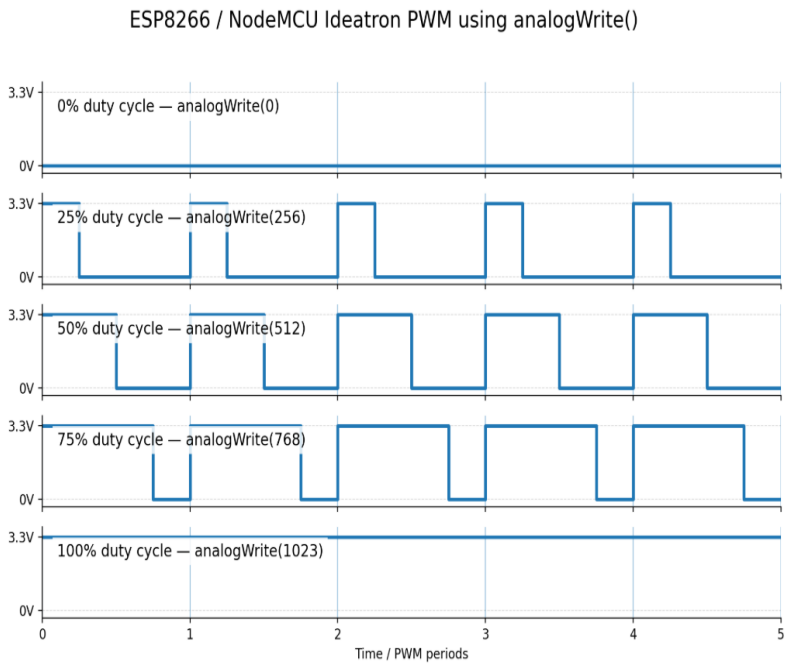
```
from machine import ADC
from time import sleep
soil = ADC(0)
DRY = 250   # value measured in dry air/dry soil
WET = 850   # value measured in water/wet soil
while True:
    try:
        raw = soil.read()
    except Exception as e:
        break
    moisture = (raw - DRY) * 100 / (WET - DRY)
    if moisture < 0:
        moisture = 0
    elif moisture > 100:
        moisture = 100
    print("Raw:", raw, "Moisture:", moisture, "%")
    sleep(1) #1second sleep for accuracy
```

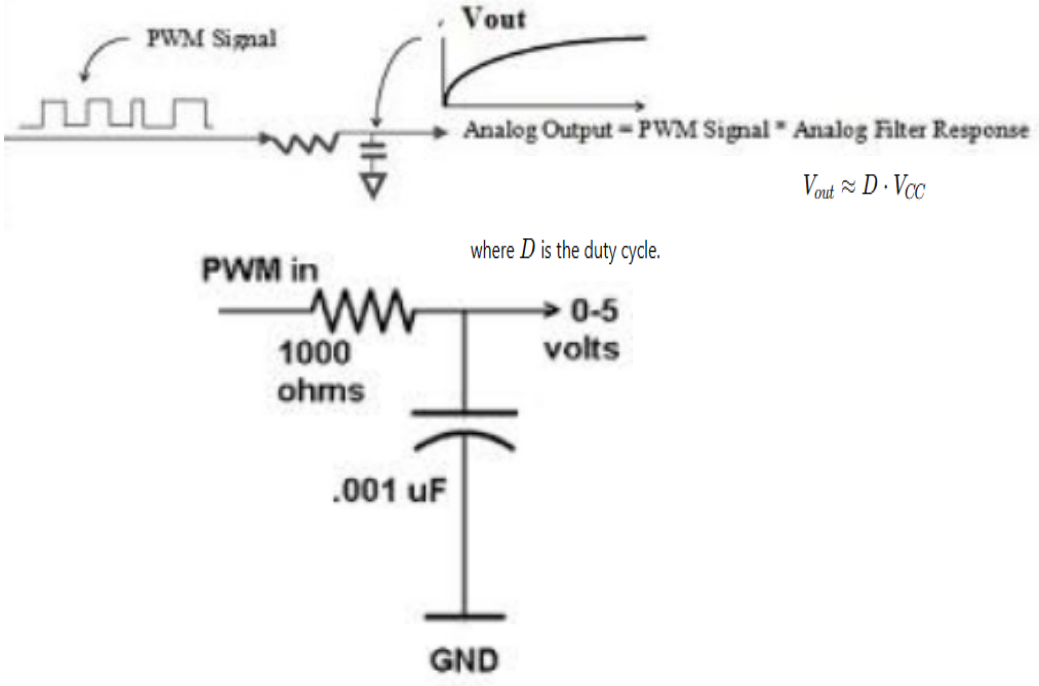
```
from machine import ADC
import time
photoresistor = ADC(0)
VREF = 3.3      # NodeMCU A0 board-level voltage range
MAX_ADC = 1023  # 10-bit ADC
while True:
    value = photoresistor.read()
    voltage = value * VREF / MAX_ADC
    print("ADC:", value, "Voltage:", voltage, "V")
    time.sleep(0.1) #100ms sleep
```

Electronics Basics

9. Digital Output PWM – ESP8266

There is a single frequency for all channels in the range fc=1–1000 Hz





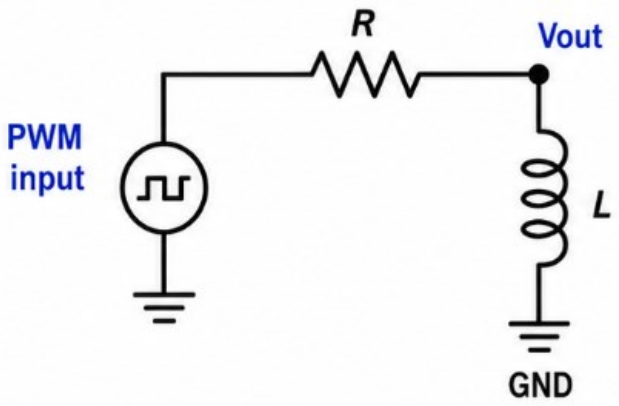
PWM up to 40 kHz

from machine import Pin, PWM

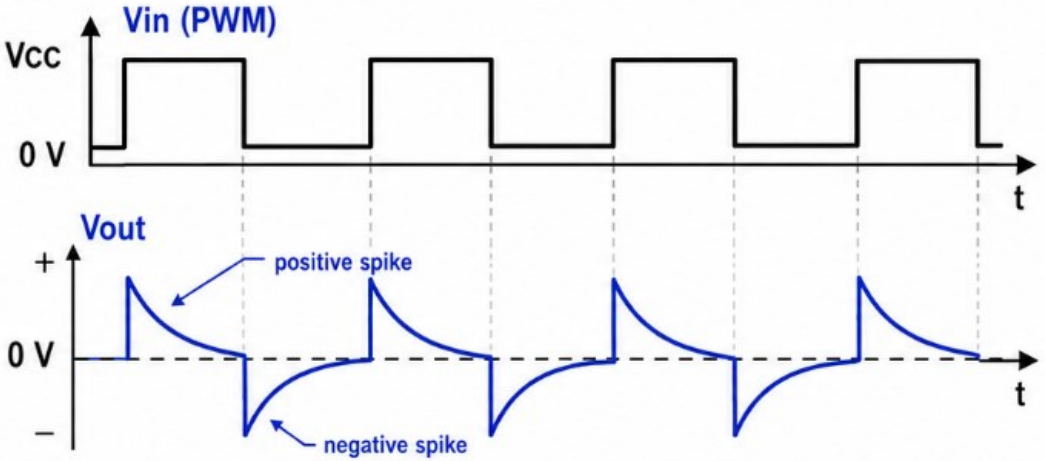
pwm = PWM(Pin(14), freq=1000, duty=512) # 1 kHz, 50% duty cycle (10bit analogWrite)

9. Digital Output PWM – ESP8266

PWM with Inductor to Ground (R-L configuration)



Circuit: PWM -> R -> Vout, with L from Vout to GND



Behavior of R-L to Ground

- Inductor to ground does **NOT** smooth PWM into analog DC.
- Vout is spike-like / high-pass behavior.
- Average output is approximately 0 V.
- Time constant: tau = *L* / *R*.

Transfer Function and Response

$$\frac{V_{out}}{V_{in}} = \frac{sL}{R + sL}$$

After a rising edge:

$$V_{out}(t) = V_{CC}\, e^{-Rt/L}$$

Where:

s = complex frequency (Laplace variable)

R = series resistance (ohms)

L = inductor (henries)

tau = time constant = L / R

RC filter: $V_{out} \approx$ duty cycle $\times V_{CC}$

R-L-to-ground circuit: V_{out} = spikes, not analog smoothing

Electronics Basics

9. I2C -Inter-Integrated Circuit

Line	Meaning	Function
SDA	Serial Data	Transfers data between the microcontroller and the device
SCL	Serial Clock	Provides the clock signal for synchronization

Each device has its own unique address
It supports multiple devices using two wires
I²C speed = 100 kHz /Conversion time: 10-30ms
1Byte address size - I²C address byte = [7-bit address][R/W bit]

```
from machine import Pin, I2C
i2c = I2C(scl=Pin(5), sda=Pin(4), freq=100000)
```

Electronics Basics

9. Multiple I2C -Inter-Integrated Devices with the OLED pins (D1-D2 – GPIO4/5)

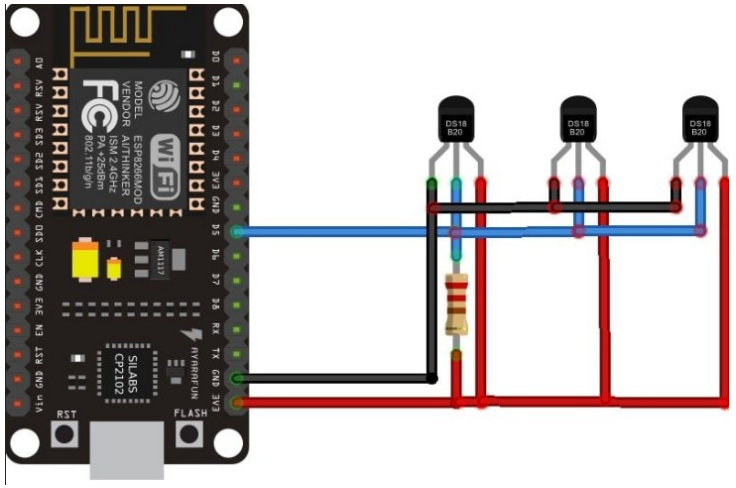
Device	Type / Use	Typical I2C address
OLED SSD1306	Display	`0x3C` or `0x3D`
BME280	Temperature / humidity / pressure sensor	`0x76` or `0x77`
ADS1115	16-bit analog-to-digital converter	`0x48`, `0x49`, `0x4A`, `0x4B`
DS3231	Real-time clock module	`0x68`
BMP280	Pressure / temperature sensor	`0x76` or `0x77`
MPU6050	Accelerometer + gyroscope	`0x68` or `0x69`

Electronics Basics

10. One-Wire

Unlike I²C it uses just one digital line

Supports multiple devices on one wire
Standard One-Wire speed ≈ 16 kbps
Conversion time -> 750ms




```
from machine import Pin, I2C
i2c = I2C(scl=Pin(5), sda=Pin(4), freq=100000)
```

```
import machine
import onewire
import ds18x20
import time
# D5 = GPIO14
data_pin = machine.Pin(14)
ds_bus = ds18x20.DS18X20(onewire.OneWire(data_pin))
sensors = ds_bus.scan()
print("DS18B20 sensors found:", sensors)
while True:
    ds_bus.convert_temp()
    time.sleep_ms(750)
    for sensor in sensors:
        address = ''.join('{:02x}'.format(byte) for byte in sensor)
        print("One-Wire address:", address)
        print("Family code:", hex(sensor[0]))
        temperature = ds_bus.read_temp(sensor)
        print("Sensor:", sensor, "Temperature:", temperature, "C")
    time.sleep(2)
```

For a DS18B20, each sensor has a unique 64-bit address, stored as 8 bytes.
[bytearray(b'(\xffd\x1eq\x16\x04\x9c')]

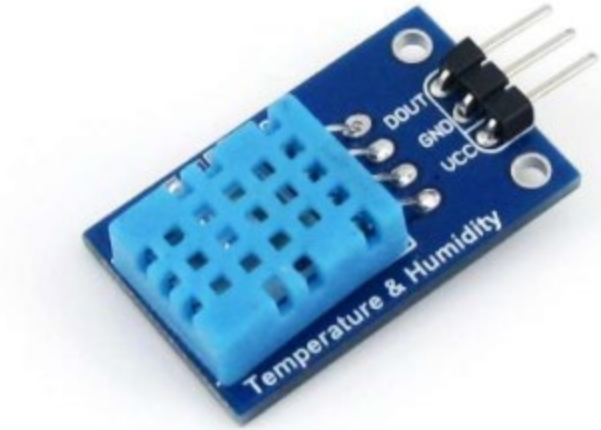
Electronics Basics

Can multiple DHT22 sensors share one data pin?
Normally, no.
We cannot share the same pin with other devices also!

11. DHT22/DHT11 one-wire proprietary

Unlike I²C it uses just one digital line

Supports only one device– No addressing
40-bit data frame (16bit T/16bit H/8bit Checksum)
Conversion time -> 2seconds

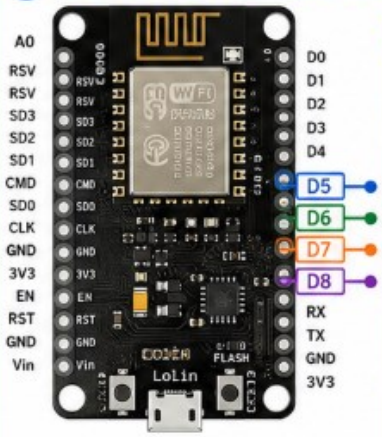


```
from machine import Pin
import dht
import time

# D5 = GPIO14
sensor = dht.DHT22(Pin(14))

while True:
    try:
        sensor.measure()

        temperature = sensor.temperature()
        humidity = sensor.humidity()

        print("Temperature:", temperature, "C")
        print("Humidity:", humidity, "%")

    except OSError as e:
        print("DHT22 read error:", e)

    time.sleep(2)
```

12 .Synchronous Peripheral Interface

ESP8266 SPI Specs, Speeds, and Pins

1 ESP8266 NodeMCU & HSPI Pins



Pin (NodeMCU)	GPIO	SPI Signal
D5	GPIO14	SCLK / CLK
D6	GPIO12	MISO
D7	GPIO13	MOSI
D8	GPIO15	CS / SS

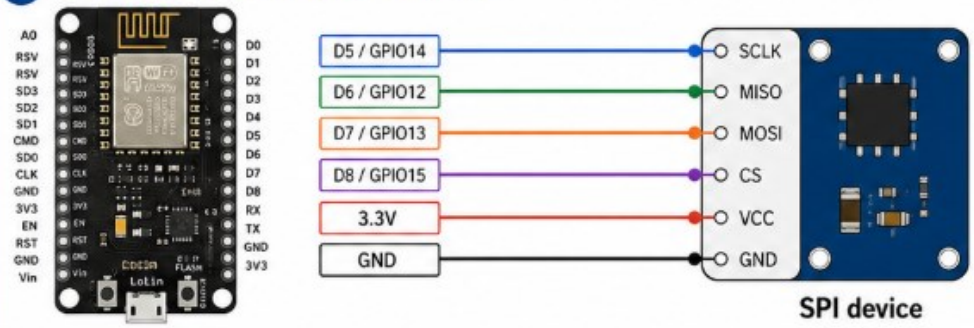
These are the default HSPI pins typically used for external SPI devices.

Logic level: 3.3 V

2 SPI characteristics

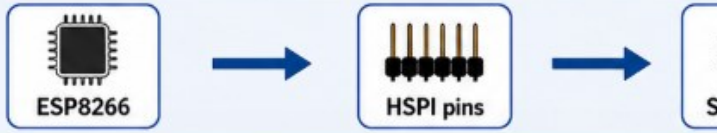
Bus used for peripherals	HSPI
Internal flash bus	SPI0 (reserved for onboard flash)
Communication	Full duplex
Roles	Master commonly used
SPI modes	Mode 0, 1, 2, 3
Bit order	MSB first
Typical clock settings	1 MHz, 4 MHz, 8 MHz, 10 MHz, 20 MHz
Maximum clock (theoretical)	up to 80 MHz
Practical note	many displays and sensors use lower speeds for reliability

3 ESP8266 NodeMCU to SPI Device Wiring



Important notes

- Use 3.3 V SPI devices or proper level shifting.
- SPI0 is normally connected to flash memory, so use HSPI for sensors, displays, and ADC/DAC modules.
- Keep wires short for higher clock speeds.
- CS selects the target device; multiple SPI devices can share SCLK, MOSI, and MISO with separate CS lines.

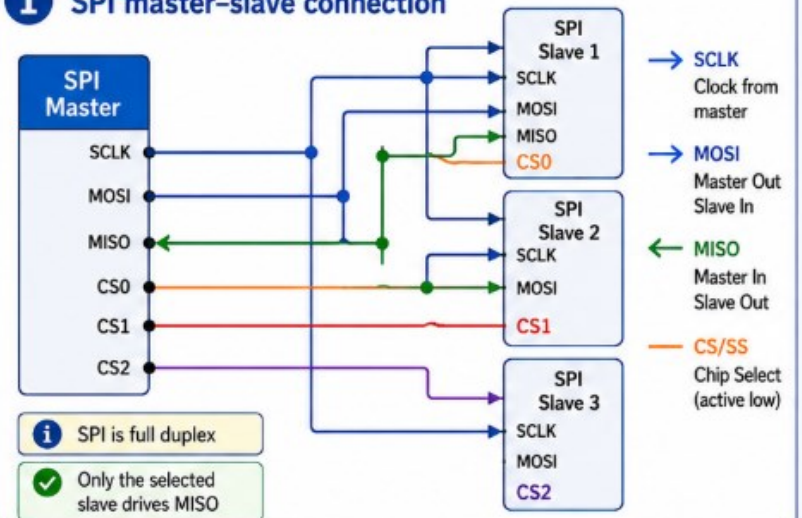


Default external SPI on NodeMCU:
D5=CLK, D6=MISO, D7=MOSI, D8=CS

12 .Synchronous Peripheral Interface

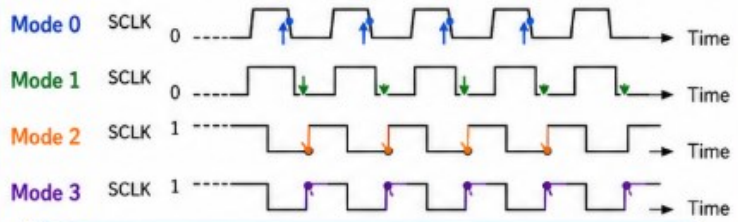
SPI Modes, Master–Slave, and Addressing

1 SPI master–slave connection



2 SPI modes (CPOL / CPHA)

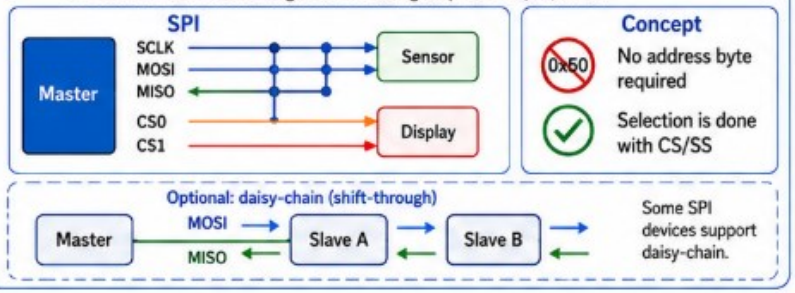
Mode	CPOL	CPHA	Clock idle level	Data sampled on
Mode 0	0	0	Low (0)	Rising edge
Mode 1	0	1	Low (0)	Falling edge
Mode 2	1	0	High (1)	Falling edge
Mode 3	1	1	High (1)	Rising edge



Master and slave must use the same SPI mode.

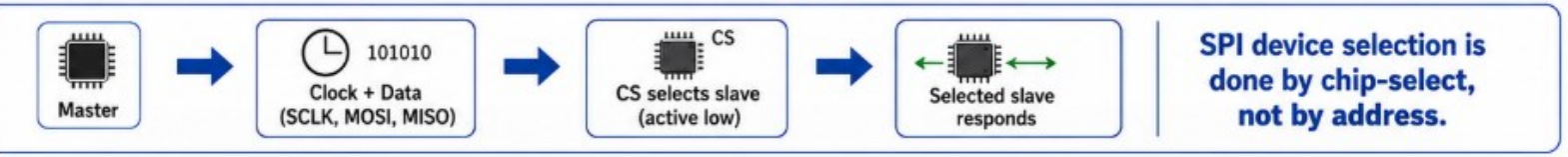
3 Addressing and slave selection

SPI does not embed addresses in the data frame like I²C.
The master selects the target device using chip-select (CS) lines.



4 Important notes

- One master can control many slaves.
- SCLK, MOSI, and MISO can be shared.
- Each slave usually needs a separate CS/SS line.
- SPI is faster than I²C in many applications.
- No built-in acknowledgement or addressing field.
- Typical uses: displays, ADCs, DACs, flash memory, sensors.



12 .Synchronous Peripheral Interface

```
from machine import Pin, SPI
import time
# Initialize SPI (Bus 1)
# sck: GPIO14, mosi: GPIO13, miso: GPIO12 (common ESP8266 mapping)
spi = SPI(1, baudrate=1000000, polarity=0, phase=0, sck=Pin(14), mosi=Pin(13), miso=Pin(12))
# Define Chip Select (CS) pin (GPIO15 is typical, but can be changed)
cs = Pin(15, Pin.OUT)
cs.value(1)  # Deselect device initially
print("SPI Initialized")
```

Half duplex

```
# Example: Send command 0x01 and data 0x42
command = bytearray([0x01, 0x42])
# Pull CS low to select device
cs.value(0)
# Send data
spi.write(command)
# Pull CS high to deselect
cs.value(1)
print("Data sent")
# --- Read Example ---
# Send a read command (e.g., 0x03) and then read 4 bytes of data
read_cmd = bytearray([0x03])
cs.value(0)
spi.write(read_cmd)            # Send read instruction
data = spi.read(4)             # Read 4 bytes
cs.value(1)
print("Received:", data)
```

Full duplex

```
# Send 0xFF and store received data
tx_data = b'\xFF\xFF\xFF\xFF'  # 4 bytes of dummy data
rx_data = bytearray(4)        # Buffer for received data

cs.value(0)
spi.write_readinto(tx_data, rx_data)  # Writes tx_data and reads into rx_data at the same time
cs.value(1)

print("Received data:", rx_data)
```

13. Universal Asynchronous Receiver Transmitter

UART on ESP8266 (NodeMCU)

Which UART to use and how to communicate with a serial device using Python (pySerial)

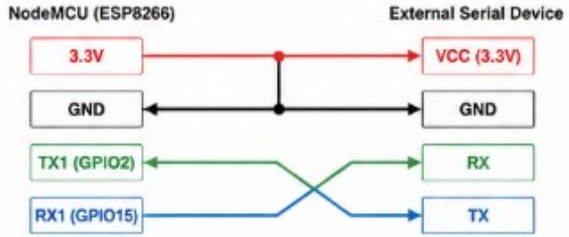
1. UART Interfaces on ESP8266

✓ Use UART0 for programming/serial console (default)

✗ Use UART1 (TX/RX pins) to communicate with external serial devices

UART	Type	TX Pin	RX Pin	Default Use
UART0	Primary	TX0 (GPIO1)	RX0 (GPIO3)	Flash programming Serial console (REPL) Debug logs
UART1	Secondary	TX1 (GPIO2)	RX1 (GPIO15)	User serial communication
UART2	(Limited)	N/A	N/A	Not available (separate HW UART not broken out)

2. UART1 Pinout (Recommended for Devices)



⚠ **ESP8266 is 3.3V only. Do NOT connect 5V devices directly.**

3. How to Communicate with a Serial Device Using Python (pySerial)

Step 1: Install pySerial

```
pip install pyserial
```

Step 2: Python Example

```
import serial
import time

# Change 'COM5' to your actual port
# On Windows: 'COM5', on Linux: '/dev/ttyUSB0', on Mac: '/dev/tty.usbserial-xxxx'
ser = serial.Serial(
    port='COM5',
    baudrate=9600,
    bytesize=8,
    parity='N',           # No parity
    stopbits=1,
    timeout=1             # Read timeout in seconds
)
time.sleep(2)  # Wait for the connection to initialize

# Send data to the device
ser.write(b'HELLO\r\n')    # Send bytes (\r\n for newline)

# Read response from device
response = ser.readline().decode(errors='ignore').strip()
print('Received:', response)

# Read all available data (non-blocking)
# data = ser.read(ser.in_waiting).decode(errors='ignore')
# print('Received:', data)

ser.close()
```

Common Methods

```
ser.write(b'...')
# send bytes

ser.readline()
# read a line
(ends with \n)

ser.read(size)
# read fixed
number of bytes

ser.in_waiting
# bytes waiting
in RX buffer

ser.close()
# close port
```

4. UART1 Communication Settings (Typical)

Parameter	Typical Value	Description
Baud Rate	9600 (common) 4800 / 19200 / 115200	Must match device setting
Data Bits	8	Usually 8
Parity	N (None)	N = None, E = Even, O = Odd
Stop Bits	1	Usually 1
Flow Control	None	ESP8266 UART1 has no HW flow control

Example Common Baud Rates

- 4800
- 9600
- 19200
- 38400
- 57600
- 115200

5. Notes

- Use UART0 (GPIO1/GPIO3) only for flashing and debug console.
- Use UART1 (GPIO2 TX1, GPIO15 RX1) for external serial devices.
- Ensure GND is common between ESP8266 and the device.
- Use 3.3V logic levels only.
- If device is 5V, use a level converter.

6. Typical Use Cases

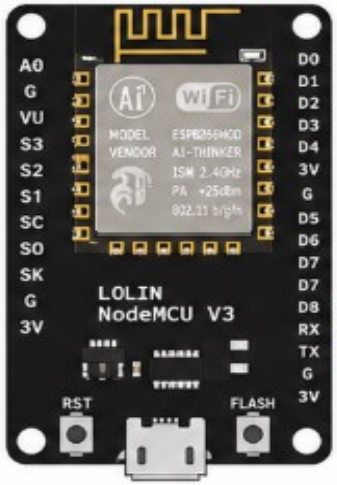
- GPS Modules (u-blox, NEO-6M)
- RF Modules (HC-12, RFM69 with UART mode)
- Serial Displays (Nextion in UART mode)
- Any TTL Serial Device

💡 **Summary: Use UART1 (GPIO2 TX1, GPIO15 RX1) to communicate with external serial devices. Use pySerial on your PC to send/receive data.**

13. Universal Asynchronous Receiver Transmitter

ESP8266 UART Example: Serial.begin() and Sending Random Numbers

1 ESP8266 UART pins



UART0 (default)

- TX0 = GPIO1
- RX0 = GPIO3

Use UART0 / Serial for USB serial monitor and TTL serial communication

Logic level: 3.3 V

2 Arduino code on ESP8266

```
void setup() {
  Serial.begin(9600);
  randomSeed(micros());
}

void loop() {
  int value = random(0, 1000);
  Serial.print("Random: ");
  Serial.println(value);
  delay(1000);
}
```

Serial.begin(9600) sets baud rate to 9600 bps.

randomSeed(micros()) initializes randomness based on microsecond timer.

Serial.println(value) sends the random number followed by newline once per second.

3 What is sent over UART

ESP8266

TX

PC Serial Monitor

4 Python pySerial receiver

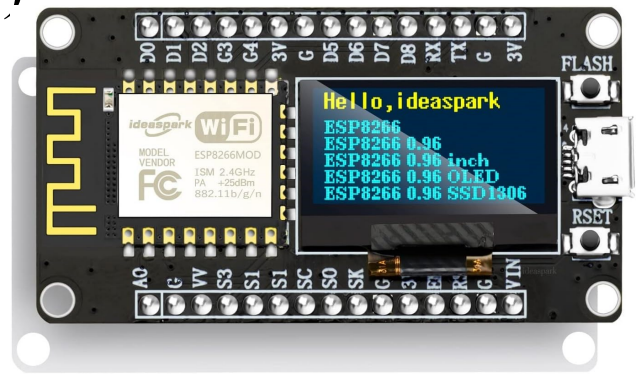
```
import serial
ser = serial.Serial('COM5', 9600, timeout=1)
while True:
    line = ser.readline().decode().strip()
    if line:
        print(line)
```

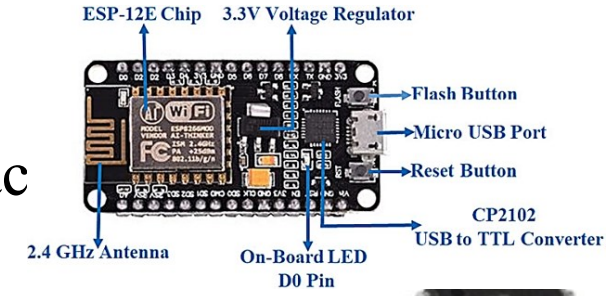
Replace COM5 with your port, e.g. /dev/ttyUSB0 on Linux.

Flow: ESP8266 generates random number -> Serial.println() sends text -> PC reads it with pySerial

Development Board

3.3V ARM 32bit quad-core device (ESP8266)
CPU:32-bit Tensilica L106 RISC processor
CPU frequency:80MHz
Wi-Fi: Integrated 2.4GHz IEEE 802.11 b/g/n
Security: WPA2 supported
Typical IoT protocols: HTTP/HTTPS/MQTT
Flash memory:4MB (program memory)
RAM: 128 KB
EEPROM: Emulated in Flash memory (4096 bytes / 4 KB)
ADC:resolution:10bit
PWM:supported in all pins
OLED: SSD1306, 128 × 64 pixels, I²C interfac

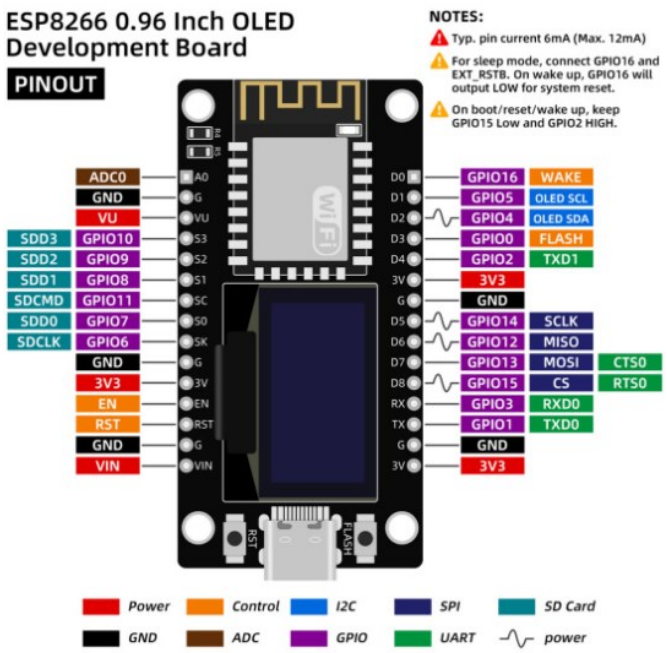




Cost:7-12€

https://manuals.plus/asin/B0BVM3H46B

Development Board

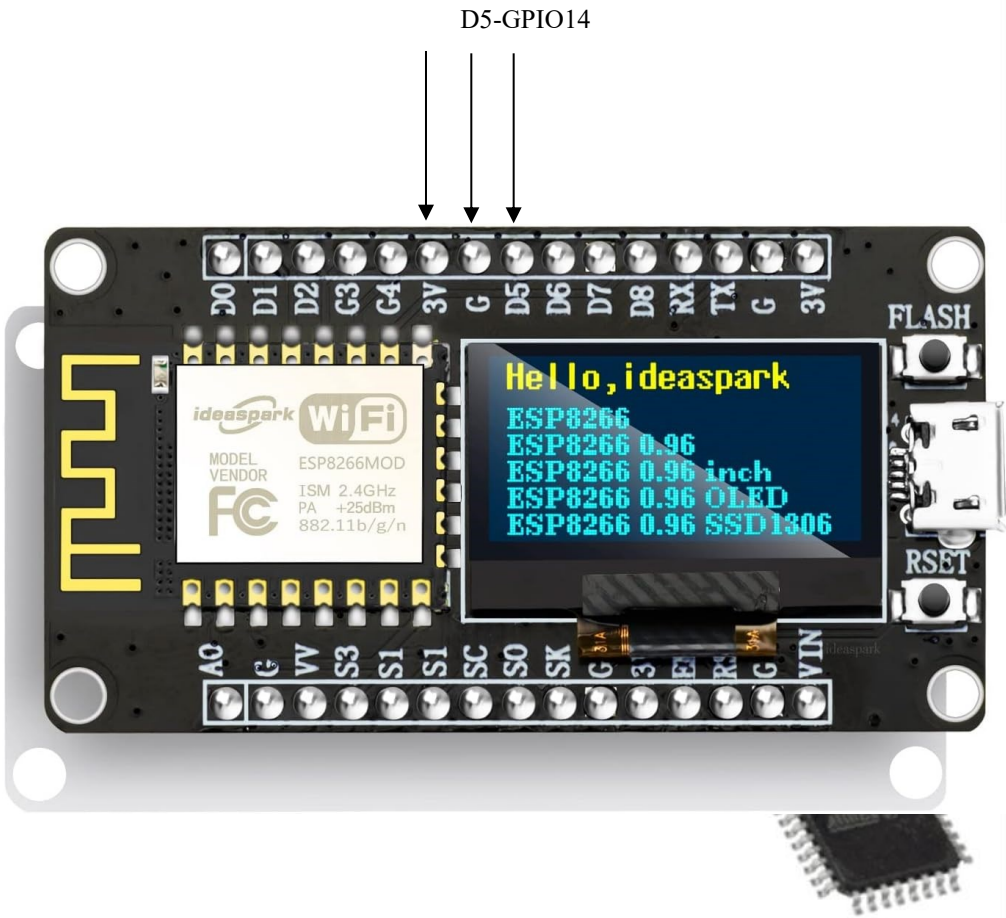


NODEMCU ESP8266 0.96inch OLED display(ESP12E)

PIN	GPIO	MicroPython	Common use
D0	GPIO16	Pin(16)	Digital I/O, deep-sleep wake
D1	GPIO5	Pin(5)	I²C SCL, digital I/O-OLED
D2	GPIO4	Pin(4)	I²C SDA, digital I/O-OLED
D3	GPIO0	Pin(0)	Boot-sensitive pin
D4	GPIO2	Pin(2)	Built-in LED on many boards, boot-sensitive pin
D5	**GPIO14**	**Pin(14)**	**SPI SCLK, digital I/O**
D6	**GPIO12**	**Pin(12)**	**SPI MISO, digital I/O**
D7	**GPIO13**	**Pin(13)**	**SPI MOSI, digital I/O**
D8	GPIO15	Pin(15)	SPI CS
RX	GPIO3	Pin(3)	UART receive
TX	GPIO1	Pin(1)	UART transmit
A0	**ADC0**	**ADC(0)**	**Analog input**

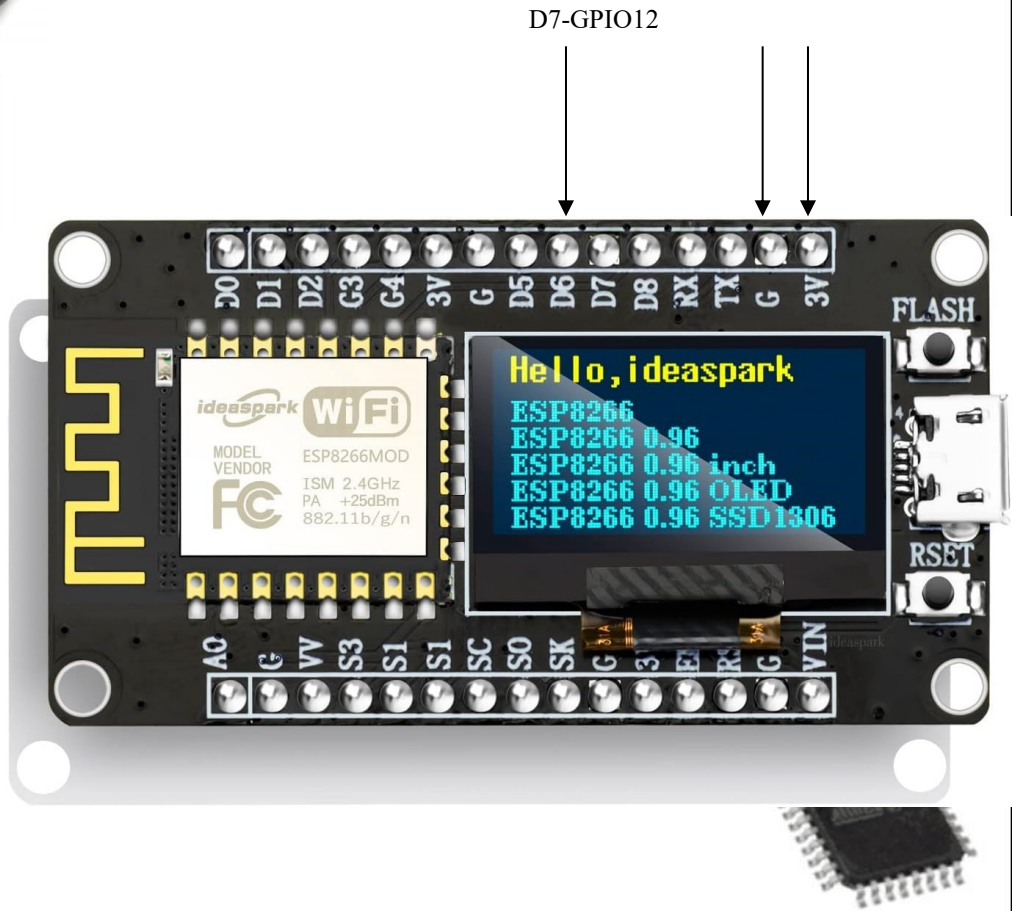
Let’s grab the blocks and connect

1. DHT22 one-wire proprietary



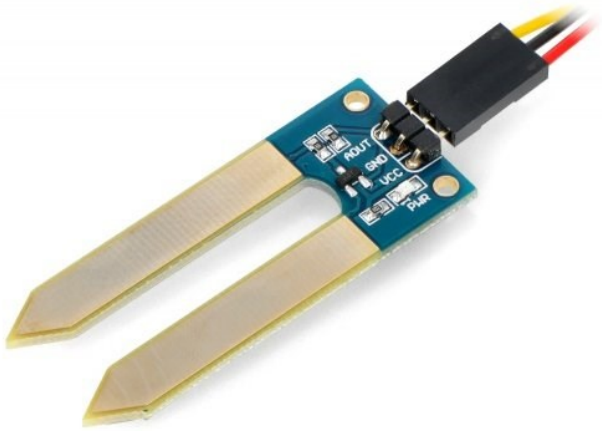
Let’s grab the blocks and connect

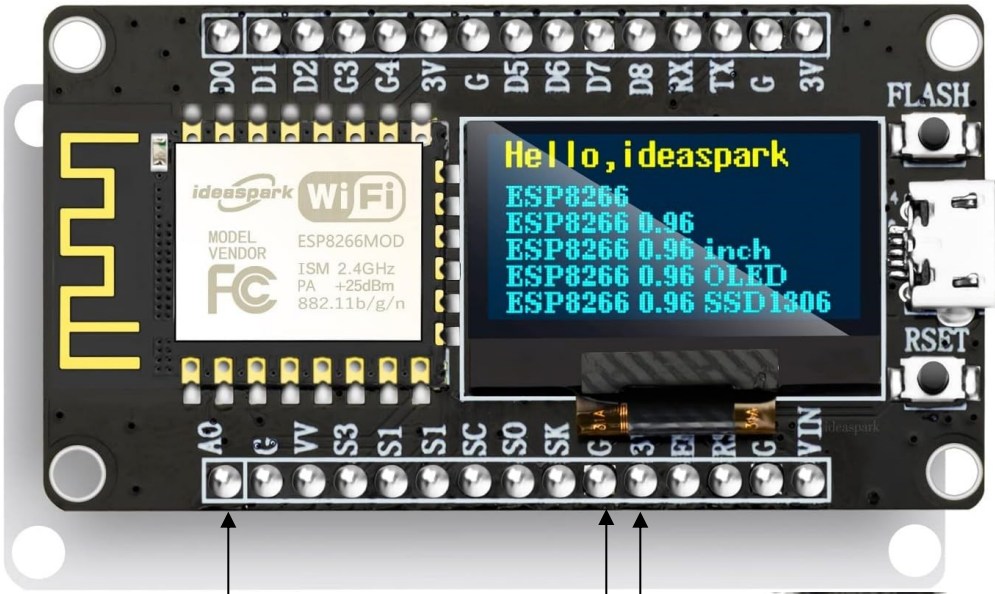
2. DS18B20 one-wire temperature probe



Let’s grab the blocks and connect

3. Waveshare soil moisture sensor



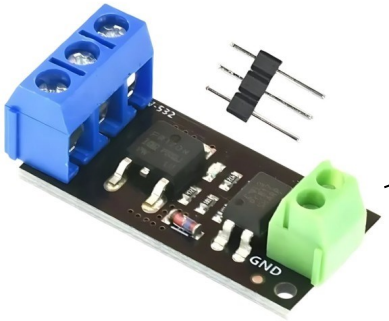


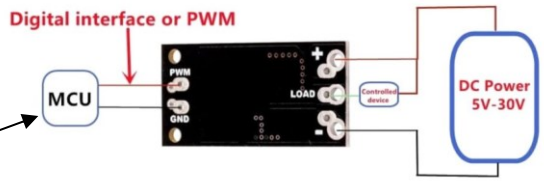
A0-GPIO0/ADC0

Let’s grab the blocks and connect

Load

4. Valve Actuator + driver

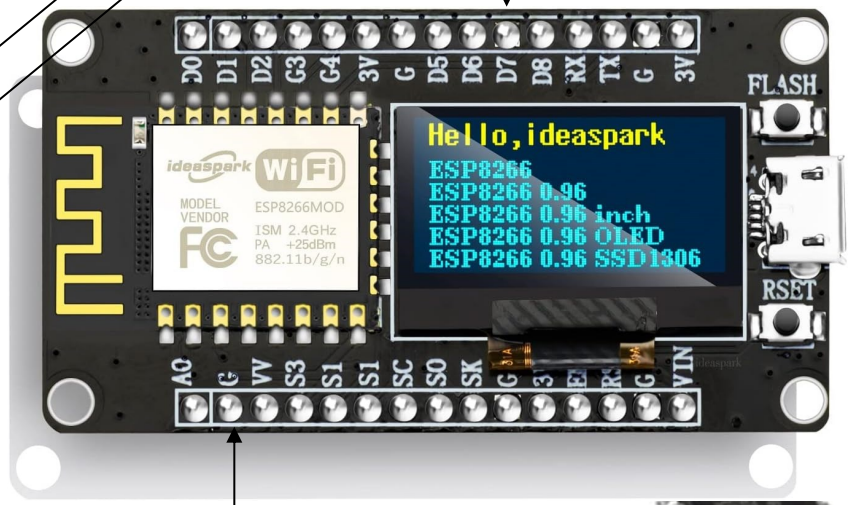




Positive Load =red
Negative Load =black

D7-GPIO13

External load circuit	FR120N module
External power supply positive (5V)	**+ / DC+**
Load positive wire	**+ / DC+**
Load negative wire	**LOAD**
External power supply negative (0V)	**- / DC-**



NOW Lets program the Board using MicroPython!!

Equipment	Qty	Cost (€)
ESP8266 NodeMCU with integrated 0.96" OLED	1	8.43
DHT22 temperature/humidity sensor	1	5.10
DS18B20 waterproof temperature sensor	1	2.70
Waveshare analog soil moisture sensor	1	3.50
FR120N MOSFET control module	1	2.40
Mini 3–5 V water pump	1	2.40
Jumper wires set	1	4.90
Total cost		**29.43**

4. NOW Lets program the Board using MicroPython!!

Initial Stuff:

```
# Valve rule thresholds
TEMP_THRESHOLD = 18.0
HUMIDITY_THRESHOLD = 70.0
SOIL_THRESHOLD_RAW = 150   # calibrated from your values
# Soil sensor calibration
SOIL_DRY_RAW = 4
SOIL_WET_RAW = 360
# ============================================================
# PIN MAPPING (NodeMCU ESP8266)
# D1 = GPIO5   -> OLED SCL
# D2 = GPIO4   -> OLED SDA
# D5 = GPIO14  -> DHT22
# D6 = GPIO12  -> DS18B20
# D7 = GPIO13  -> MOSFET / valve control
# A0           -> Analog soil moisture
# ============================================================
PIN_OLED_SCL = 5
PIN_OLED_SDA = 4
PIN_DHT22 = 14
PIN_DS18B20 = 12
PIN_VALVE = 13
OLED_WIDTH = 128
OLED_HEIGHT = 64
```

HW Init:

```
# ============================================================
# HARDWARE INIT
# ============================================================
oled = None
try:
    i2c = I2C(sda=Pin(PIN_OLED_SDA), scl=Pin(PIN_OLED_SCL), freq=50000)
    devices = i2c.scan()
    print("I2C devices:", devices)
    if 0x3C in devices:
        oled = ssd1306.SSD1306_I2C(OLED_WIDTH, OLED_HEIGHT, i2c, addr=0x3C)
    elif 0x3D in devices:
        oled = ssd1306.SSD1306_I2C(OLED_WIDTH, OLED_HEIGHT, i2c, addr=0x3D)
    else:
        print("OLED not found")
except Exception as e:
    print("OLED init error:", e)
    oled = None
dht_sensor = dht.DHT22(Pin(PIN_DHT22))
ow = onewire.OneWire(Pin(PIN_DS18B20))
ds = ds18x20.DS18X20(ow)
ds_roms = ds.scan()
print("DS18B20 ROMs:", ds_roms)
adc = ADC(0)
valve = Pin(PIN_VALVE, Pin.OUT)
valve.value(0)
```

Read F:

```
def read_ds18b20():
    if not ds_roms:
        return None
    try:
        ds.convert_temp()
        time.sleep_ms(750)
        return ds.read_temp(ds_roms[0])
    except Exception as e:
        print("DS18B20 error:", e)
        return None

def read_dht22():
    try:
        dht_sensor.measure()
        return dht_sensor.temperature(), dht_sensor.humidity()
    except Exception as e:
        print("DHT22 error:", e)
        return None, None

def read_soil_raw():
    try:
        return adc.read()
    except Exception as e:
        print("ADC error:", e)
        return None
```

Corr. F:

```
def soil_percent(raw, dry=SOIL_DRY_RAW, wet=SOIL_WET_RAW):
    if raw is None:
        return None
    if wet == dry:
        return 0
    pct = (raw - dry) * 100 / (wet - dry)
    if pct < 0:
        pct = 0
    if pct > 100:
        pct = 100
    return int(pct)
def soil_error(soil_pct):
    return soil_pct == 0
```

```
# Valve control
    valve_on = valve_rule(
        control_temp,
        humidity,
        soil_raw,
        soil_pct)
    if valve_on:
        valve.value(1)
    else:
        valve.value(0)
```

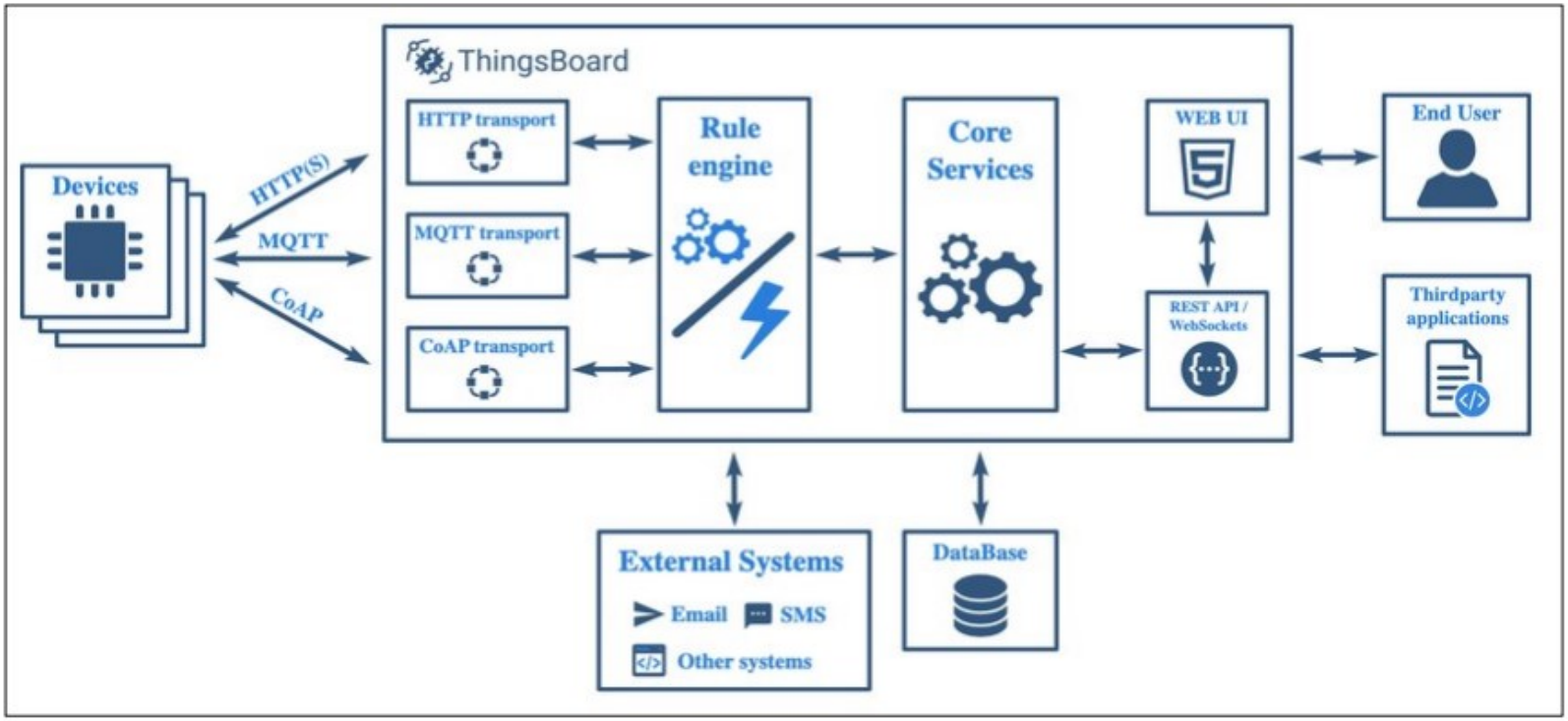
```
def valve_rule(temp_c, humidity, soil_raw, soil_pct):
    # Safety: if soil sensor reports 0%, treat as error and keep valve OFF
    if soil_error(soil_pct):
        return False

    if temp_c is None or humidity is None or soil_raw is None:
        return False

    return (
        temp_c > TEMP_THRESHOLD and
        humidity < HUMIDITY_THRESHOLD and
        soil_raw < SOIL_THRESHOLD_RAW)
```

5. ThingBoard and HTTP telemetry/Wi-Fi connectivity

ThingsBoard IoT Cloud Platform

- **ThingsBoard** is an open-source IoT platform for data collection, processing, visualization, and device management.
- **Open-Source Advantage:** allows customization, and community support.
- **ThingsBoard Core Functionalities:**
  - ❑ **Device Management and Data Collection:** ThingsBoard can handle a large number of devices and high data volumes, ensuring robust performance for critical IoT deployments at all times.
  - ❑ **Data Visualization:** Users can create custom dashboards to visualize data, monitor device performance, and gain insights from IoT data streams.
  - ❑ **Data Processing and Alerting:** The rule engine enables automated responses to specific events, integrating seamlessly with other systems for automated IoT operations.

ThingsBoard Architecture Diagram



ThingsBoard Visualization Options

1. **Real-Time Monitoring:** Use real-time widgets to monitor live data streams, providing instant feedback on device performance and environmental conditions from the environment.
2. **Historical Data Analysis:** Analyze historical data to identify trends, patterns, and anomalies, enabling predictive maintenance and optimized operations over time.
3. **Alerting and Notifications:** Set up alerts and notifications based on data thresholds, ensuring timely responses to critical events and potential issues on the go.

https://parallela.math.uoi.gr:8443/

ESP8266 Python Process: Sensor Readings to ThingsBoard via HTTP POST

1 Start / Boot ESP8266

MicroPython runtime starts

2 Initialize sensors and modules

DHT22

DS18B20

Soil Moisture Sensor

Setup GPIO, ADC, I2C, OneWire, etc.

3 Take sensory measurements

air_temperature = 24.8 C

humidity = 48.5 %

ds18b20_temperature = 24.6 C

soil_moisture_raw = 512

4 Create Python dictionary

```
telemetry = {
 "air_temperature": 24.8,
 "humidity": 48.5,
 "ds18b20_temperature": 24.6,
 "soil_moisture_raw": 512
}
```

5 Convert to JSON string

```
payload = json.dumps(telemetry)
```

JSON string (payload)

```
{"air_temperature":24.8,
"humidity":48.5,
"ds18b20_temperature":24.6,
"soil_moisture_raw":512}
```

6 Connect to Wi-Fi

```
wlan.connect(ssid, password)
```

Connected to Wi-Fi

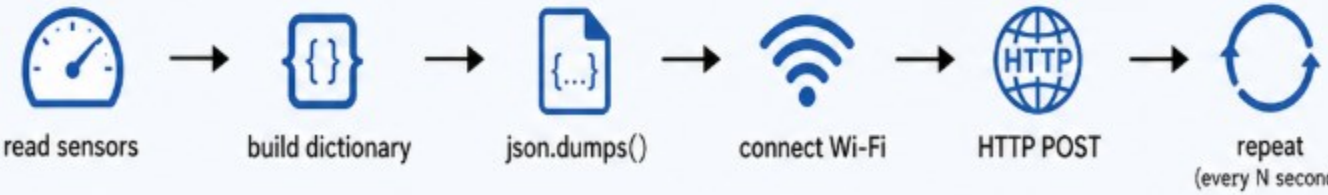
7 HTTP POST to ThingsBoard

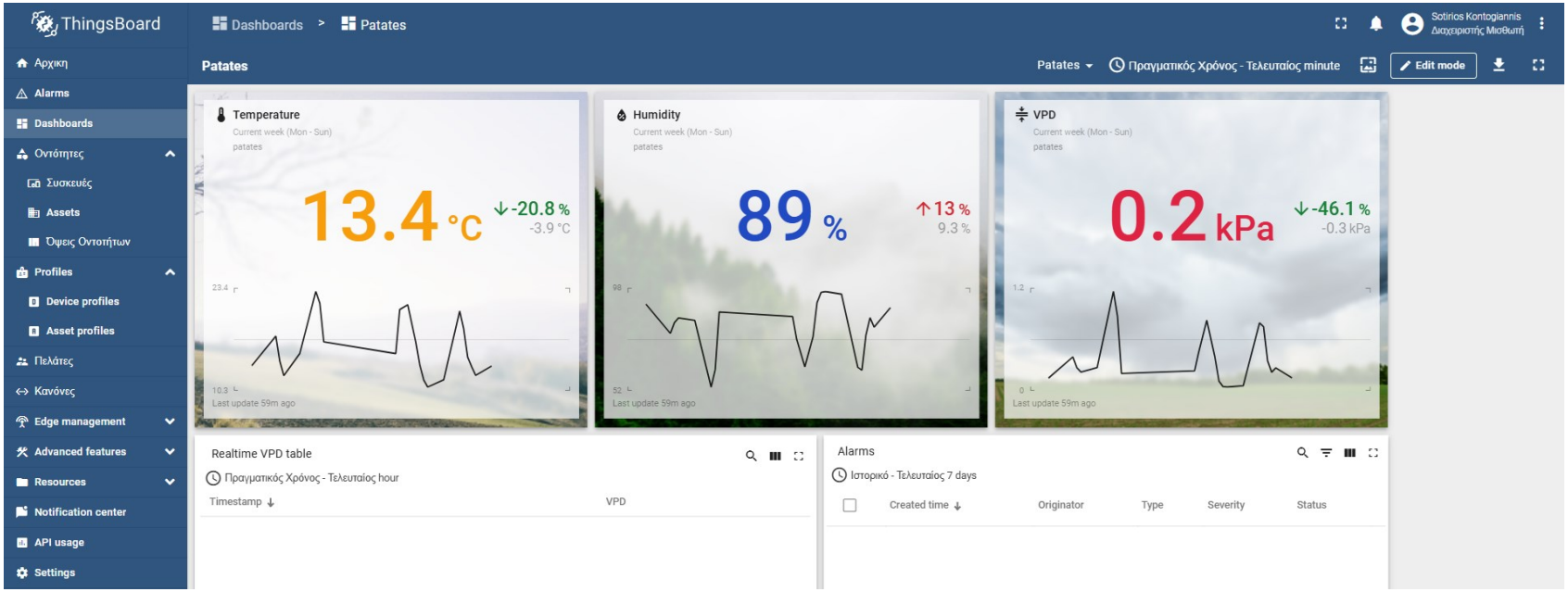
ThingsBoard

```
POST /api/v1/ACCESS_TOKEN/telemetry
Host: thingsboard.server
Content-Type: application/json

Body: payload
```

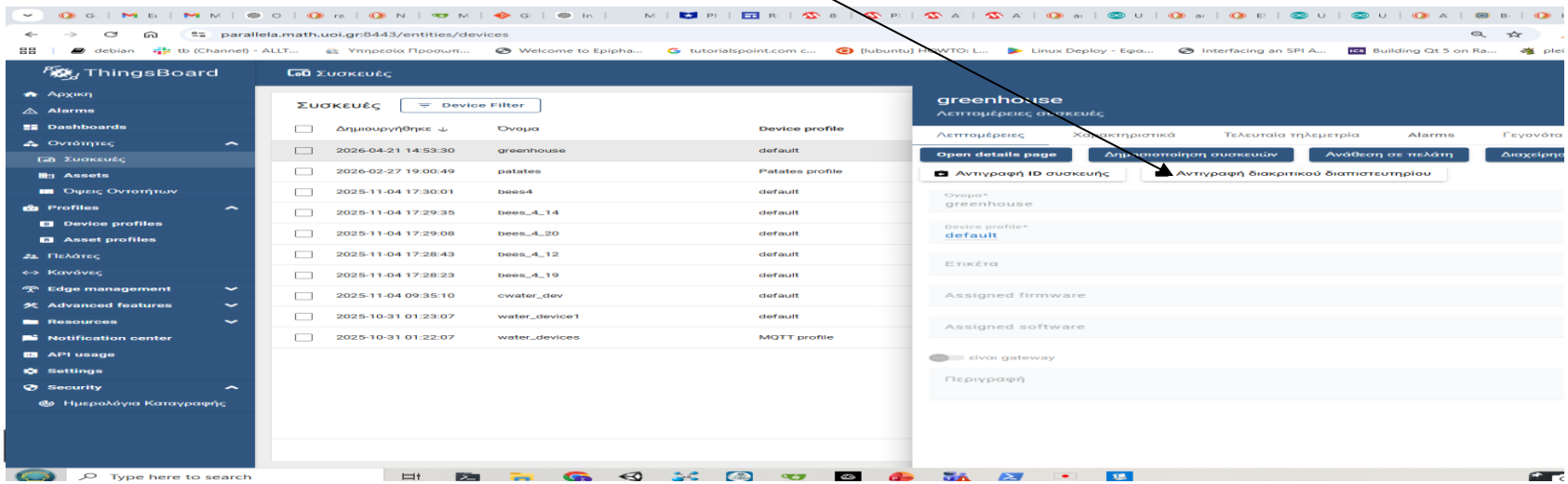
Telemetry stored and visualized on dashboard

Typical MicroPython flow





We send telemetry using HTTP:
http://parallela.math.uoi.gr:8080/api/v1/<token>/telemetry



Μικροε

Network connection

We first need to connect the device to a Wi-Fi network and get an IPV4 address

```
import network
import time

WIFI_SSID = "your_wifi_name"
WIFI_PASSWORD = "your_password"

wlan = network.WLAN(network.STA_IF)
wlan.active(True)

wlan.connect(WIFI_SSID,
WIFI_PASSWORD)

while not wlan.isconnected():
    time.sleep(1)

print("Connected:", wlan.ifconfig())
```

Telemetry protocol

We can use direct requests or RAW sockets playing with the HTTP header parameters (requests->urequests/sockets->usockets).

ESP8266 → ThingsBoard HTTP POST Telemetry

ESP8266 Sensor Node

- air_temperature = 24.80001
- ds18b20_temperature = 24.625
- humidity = 48.5
- soil_moisture_percent = 0
- soil_moisture_raw = 4
- soil_sensor_error = 1
- valve_status = 0

Wi-Fi / HTTP

ThingsBoard HTTP Service

parallela.math.uoi.gr:8080

HTTP POST Request

POST http://parallela.math.uoi.gr:8080/api/v1/dBPZ506CoqBkM9SQWtWJ/telemetry

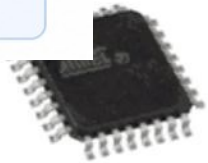
Headers

```
Host: parallela.math.uoi.gr:8080
Content-Type: application/json
Content-Length: <payload length>
Connection: close
```

JSON Body

```
{
  "air_temperature": 24.80001,
  "ds18b20_temperature": 24.625,
  "humidity": 48.5,
  "soil_moisture_percent": 0,
  "soil_moisture_raw": 4,
  "soil_sensor_error": 1,
  "valve_status": 0
}
```

Telemetry sent every 60 seconds



Simulation Example 1 of 3

```
import socket
import json
import time
import random
import requests
# ------------------------------
# ThingsBoard settings
# ------------------------------
ACCESS_TOKEN = "dBPZ506CoqBkM9SQWtWJ"
THINGSBOARD_HOST = "parallela.math.uoi.gr"
THINGSBOARD_PORT = 8080
TELEMETRY_PATH = "/api/v1/{}/telemetry".format(ACCESS_TOKEN)
BASE_URL = "http://{}:{}{}".format(THINGSBOARD_HOST,
THINGSBOARD_PORT, TELEMETRY_PATH)
```

```
# ------------------------------
# Time functions
# ------------------------------
def get_timestamp_ms():
    """
    Returns current timestamp in milliseconds
    """
    return int(time.time() * 1000)
# ------------------------------
# Simulated sensor values
# ------------------------------
def read_sensors():
    air_temperature = 20.0 + random.random() * 10.0
    ds18b20_temperature = 20.0 + random.random() * 10.0
    humidity = 35.0 + random.random() * 40.0
    soil_moisture_raw = random.randint(0, 1024)
    soil_moisture_percent = int((soil_moisture_raw / 1024.0) * 100)
    if soil_moisture_raw < 10 or soil_moisture_raw > 1015:
        soil_sensor_error = 1
    else:
        soil_sensor_error = 0
    if soil_moisture_percent < 30 and soil_sensor_error == 0:
        valve_status = 1
    else:
        valve_status = 0
    return {
        "air_temperature": round(air_temperature, 2),
        "ds18b20_temperature": round(ds18b20_temperature, 2),
        "humidity": round(humidity, 2),
        "soil_moisture_percent": soil_moisture_percent,
        "soil_moisture_raw": soil_moisture_raw,
        "soil_sensor_error": soil_sensor_error,
        "valve_status": valve_status
    }
```

Simulation Example 2 of 3

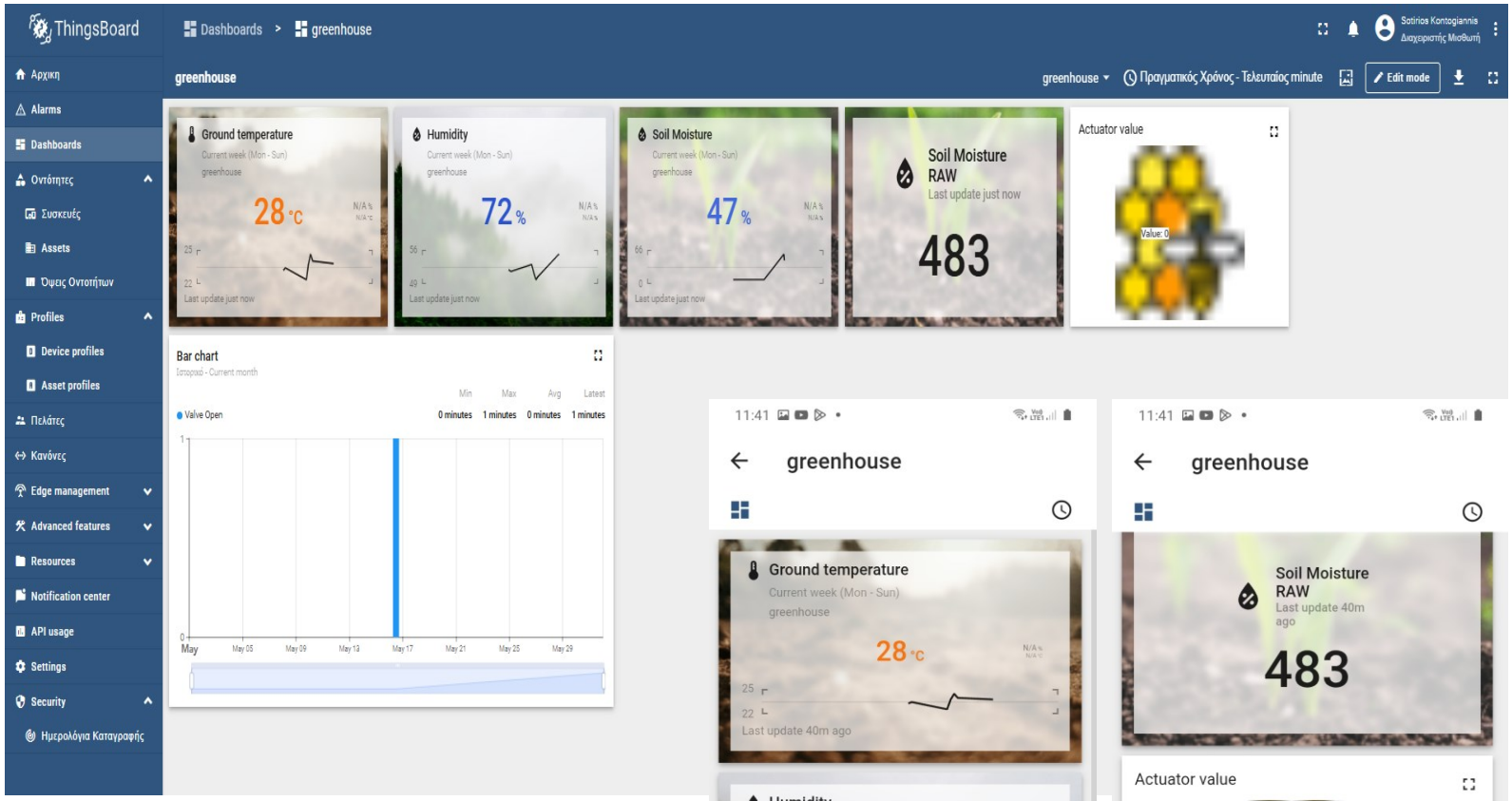
```
# ------------------------------
# Send telemetry using requests (recommended)
# ------------------------------
def send_telemetry_requests(values):
    """
    Send telemetry using the requests library (simpler and more robust)
    """
    payload = {
        "ts": get_timestamp_ms(),
        "values": values
    }

    print("Current time:", get_time_string())
    print("POST {}".format(BASE_URL))
    print("Payload:", json.dumps(payload, indent=2))

    try:
        response = requests.post(
            BASE_URL,
            json=payload,
            headers={"Content-Type": "application/json"}
        )

        print("Response status code:", response.status_code)
        if response.text:
            print("Response body:", response.text)

        response.raise_for_status()  # Raise exception for bad status codes
        return True

    except requests.exceptions.RequestException as e:
        print("Requests error:", e)
        return False
```

```
# ------------------------------
# Send telemetry using raw socket (alternative)
#-----------------------------------
def send_telemetry_socket(values):
    """
    Alternative: Send telemetry using raw socket (closer to original MicroPython code)
    """
    payload = {
        "ts": get_timestamp_ms(),
        "values": values
    }
    json_payload = json.dumps(payload)
    print("Current time:", get_time_string())
    print("POST http://{}:{}{}".format(
THINGSBOARD_HOST,THINGSBOARD_PORT,TELEMETRY_PATH))
    print("Payload:", json_payload)
    try:
        # Create socket connection
        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        sock.settimeout(10)  # Set timeout
        # Connect to server
        sock.connect((THINGSBOARD_HOST, THINGSBOARD_PORT))
        # Create HTTP request
        request = (
            "POST {} HTTP/1.1\r\n"
            "Host: {}:{}\r\n"
            "User-Agent: Python-Socket\r\n"
            "Content-Type: application/json\r\n"
            "Content-Length: {}\r\n"
            "Connection: close\r\n"
            "\r\n"
            "{}"
        ).format(TELEMETRY_PATH,THINGSBOARD_HOST,THINGSBOARD_PORT,
            len(json_payload.encode('utf-8')),
            json_payload
        )
        # Send request
        sock.send(request.encode())
        # Receive response
        response = b""
        while True:
            try:
                data = sock.recv(4096)
                if not data:
                    break
                response += data
            except socket.timeout:
                break
        print("Server response:")
        print(response.decode())
        return True
    except Exception as e:
        print("Socket error:", e)
        return False
    sock.close()
```

Simulation Example 3 of 3

```
# ------------------------------
# Main program
# ------------------------------
def main():
    print("Starting sensor simulation...")
    print("Using requests library for HTTP requests\n")

    iteration = 0

    try:
        while True:
            iteration += 1
            print(f"\n--- Iteration {iteration} ---")

            # Read sensor values
            telemetry = read_sensors()

            # Send telemetry (using requests - recommended)
            success = send_telemetry_requests(telemetry)

            # Uncomment below to use socket method instead
            # success = send_telemetry_socket(telemetry)

            if success:
                print("Telemetry sent successfully")
            else:
                print("Failed to send telemetry")

            # Wait 10 seconds before next reading
            time.sleep(10)

    except KeyboardInterrupt:
        print("\n\nProgram stopped by user")


if __name__ == "__main__":
    main()
```

6. ThingBoard Data Visualization



MCSL ThingsBoard App source code and Android APK available at:

https://sensors.math.uoi.gr:3002/MCSL_Team/MCSL_ThingBoard_app

ESP8266 OLED Visualization with MicroPython SSD1306

1 Read sensor values

DHT22

`air_temperature`
24.8 C

DS18B20

`humidity`
48.5 %

`ds18b20_temperature`
24.6 C

Soil Moisture Sensor

`soil_moisture_raw`
512

2 Prepare display strings

```
line1 = "Air: 24.8C"
line2 = "Hum: 48.5%"
line3 = "DS18: 24.6C"
line4 = "Soil: 512"
line5 = "WiFi OK TB OK"
```

Create the text lines that will be drawn on the OLED.

3 Draw into framebuffer

bytearray buffer (1024 bytes) for 128x64 (128*64/8)

00	00	00	00	...	...	FF	81	00
00	10	08	...	...	...	00	00	00

FrameBuffer

```
fill(0)       # clear
text(...)     # draw text
pixel(...)    # draw pixels
scroll(...)   # scroll content
```

Text and graphics are drawn into RAM (the buffer) first.

4 OLED driver

SSD1306 (base class)

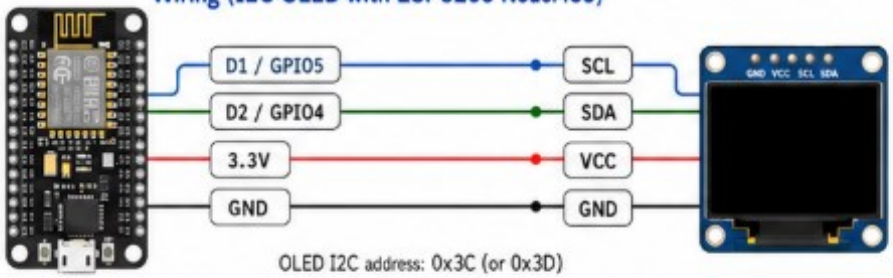
- `SSD1306_I2C` (width, height, i2c, addr=0x3C)
- `SSD1306_SPI` (width, height, spi, dc, res, cs)

- `init_display()` sends configuration commands to the OLED controller.
- `show()` sends the framebuffer (buffer) to the display.

5 Update OLED screen

`display.show()`
Sends the buffer and refreshes the OLED.

Wiring (I2C OLED with ESP8266 NodeMCU)



OLED I2C address: 0x3C (or 0x3D)

Typical update loop

```
read sensors
format text
oled.fill(0)                    # clear screen
oled.text(line1, 0, 0)
oled.text(line2, 0, 10)
oled.text(line3, 0, 20)
oled.text(line4, 0, 30)
oled.text(line5, 0, 40)
oled.show()                     # update display

repeat every N seconds
```

- Get latest sensor values
- Create display strings
- Clear the framebuffer
- Draw each line of text
- Transfer buffer to OLED
- Repeat periodically

What the driver does

- Initializes the SSD1306 controller.
- Stores pixels in `self.buffer` (RAM).
- Uses FrameBuffer for drawing.
- Supports `contrast()` and `invert()`.

Important methods

- `init_display()` – configure OLED
- `fill(color)` – clear or fill screen
- `text(s, x, y, c)` – draw text
- `show()` – push buffer to display
- `poweron()` – turn display on
- `poweroff()` – turn display off

Why framebuffer matters

- No direct "character LCD" mode.
- Graphics are composed in RAM first.
- `show()` transfers the whole frame over I2C or SPI.

Sensors → formatted text → framebuffer (RAM) → OLED display

OLED Visualization

1. Init

```
i2c = I2C( sda=Pin(PIN_OLED_SDA), scl=Pin(PIN_OLED_SCL), freq=50000 )
oled = ssd1306.SSD1306_I2C(
        OLED_WIDTH=128,
        OLED_HEIGHT=64,
        i2c,
        addr=0x3C
    )
# ============================================================
# HELPER FUNCTIONS
# ============================================================
def oled_message(lines):
    if oled is None:
        return

    oled.fill(0)

    y = 0
    for line in lines[:6]:
        oled.text(line[:21], 0, y)
        y += 10

    oled.show()
```

OLED Visualization

1. Startup

```
# ============================================================
# STARTUP
# ============================================================
oled_message([
    "ESP8266 starting",
    "Init..."
])
#------------------------------------------------
#Periodic output
#------------------------------------------------
# OLED output
    oled_message([
        "Temp:{}C".format(fmt1(control_temp)),
        "Hum :{}%".format(fmt1(humidity)),
        "Soil:{}".format(soil_display),
        "Raw :{}".format(fmti(soil_raw)),
        "Valve:{}".format("ON" if valve_on else "OFF"),
        "TB:{}".format("OK" if tb_ok else "ERR")
    ])
def fmt1(v):
    if v is None:
        return "--.-"
    return "{:.1f}".format(v)

def fmti(v):
    if v is None:
        return "---"
    return str(v)
```


END OF PRESENTATION
ANY QUESTIONS?

Pedagogical Wrap-Up

Tools Needed:

1. ThingsBoard App
https://thingsboard.io/docs/installation/docker-windows/
2. ThingsBoard Mobile App
https://thingsboard.io/products/mobile/
3. Thonny for embedded programming (Simulation using Python)
https://sensors.math.uoi.gr:3002/MCSL_Team/thonny
4. ESP8266 + Low-cost sensors (Real implementation using MicroPython)

4. WokWi Simulator (Cloud e-mail registration)
(https://wokwi.com)
5. Replace ESP8266 with ESP32 generic and MicroPython

ESP32 pinout

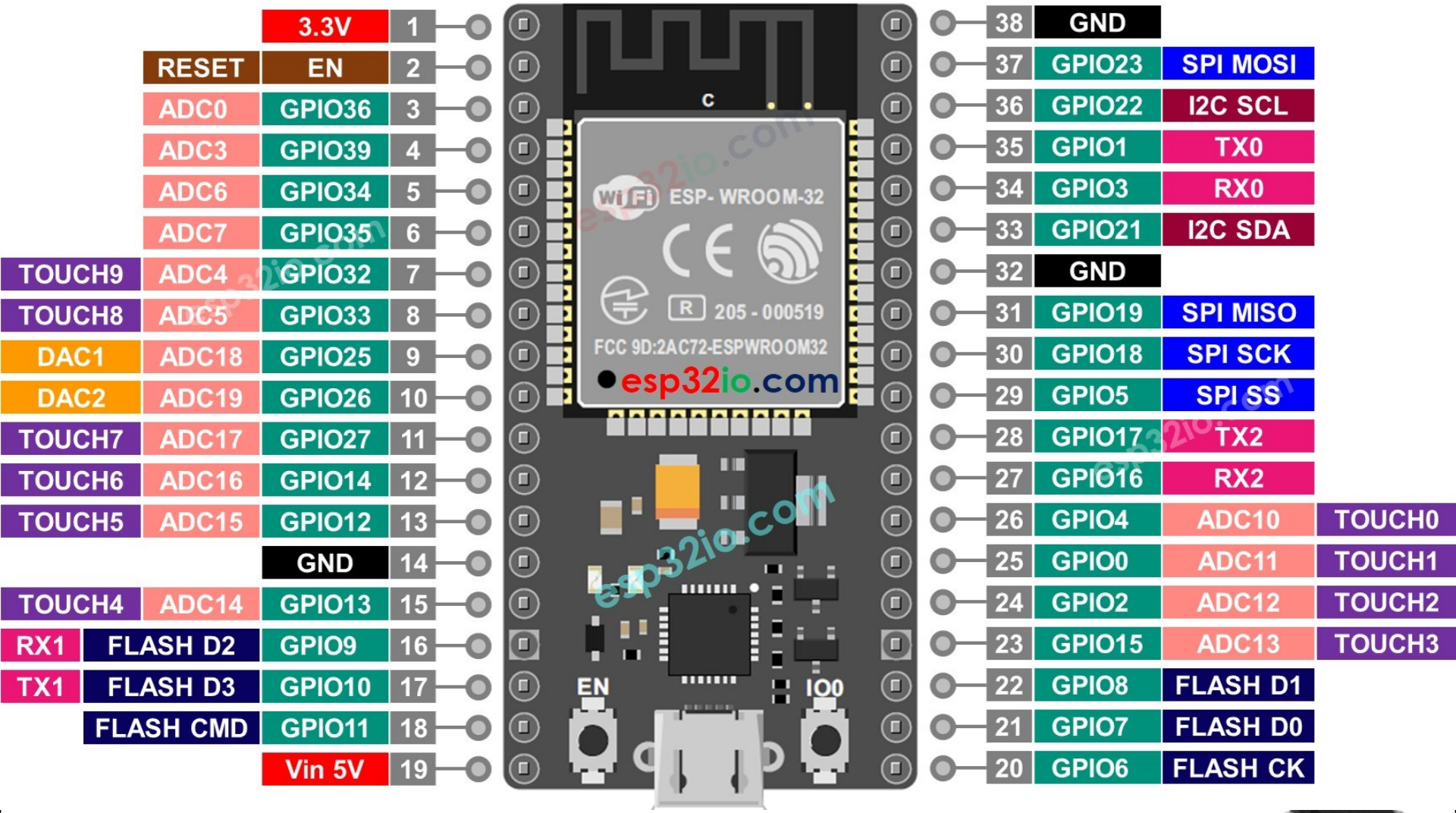
ESP32 GPIO	Use
GPIO2	**LED / output, but boot-sensitive on some boards**
GPIO4	digital I/O, PWM, ADC
GPIO5	digital I/O, PWM, SPI CS
GPIO12	**digital I/O, PWM, ADC, boot-sensitive**
GPIO13	digital I/O, PWM, ADC
GPIO14	digital I/O, PWM, ADC
GPIO15	**digital I/O, PWM, ADC, boot-sensitive**
GPIO16	**digital I/O, UART**
GPIO17	**digital I/O, UART**
GPIO18	SPI SCK, digital I/O
GPIO19	SPI MISO, digital I/O
GPIO21	**I2C SDA, digital I/O**
GPIO22	**I2C SCL, digital I/O**
GPIO23	SPI MOSI, digital I/O
GPIO25	DAC, ADC, PWM, digital I/O **(VALVE)**
GPIO26	DAC, ADC, PWM, digital I/O
GPIO27	ADC, PWM, digital I/O
GPIO32	ADC, PWM, digital I/O **(DS18B20)**
GPIO33	ADC, PWM, digital I/O **(DHT22)**

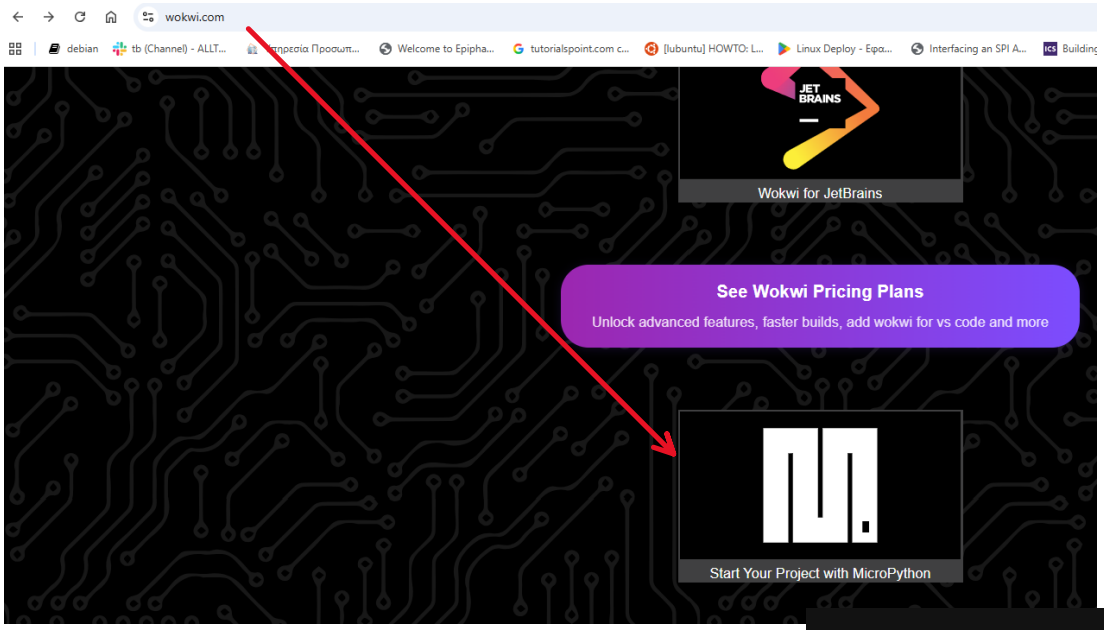
ESP32 GPIO	Use
GPIO34	input only, ADC **(SOIL SENSOR)**
GPIO35	input only, ADC
GPIO36 / VP	input only, ADC
GPIO39 / VN	input only, ADC

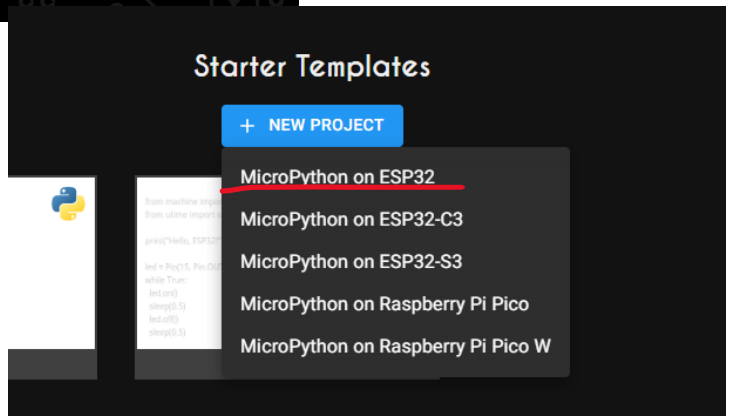
```
from machine import DAC, Pin
import time

dac = DAC(Pin(25))   # GPIO25 = DAC1

while True:
    dac.write(0)     # near 0V
    time.sleep(1)

    dac.write(128)   # about half voltage
    time.sleep(1)

    dac.write(255)   # near 3.3V
    time.sleep(1)
```

analog audio output
generating simple waveforms

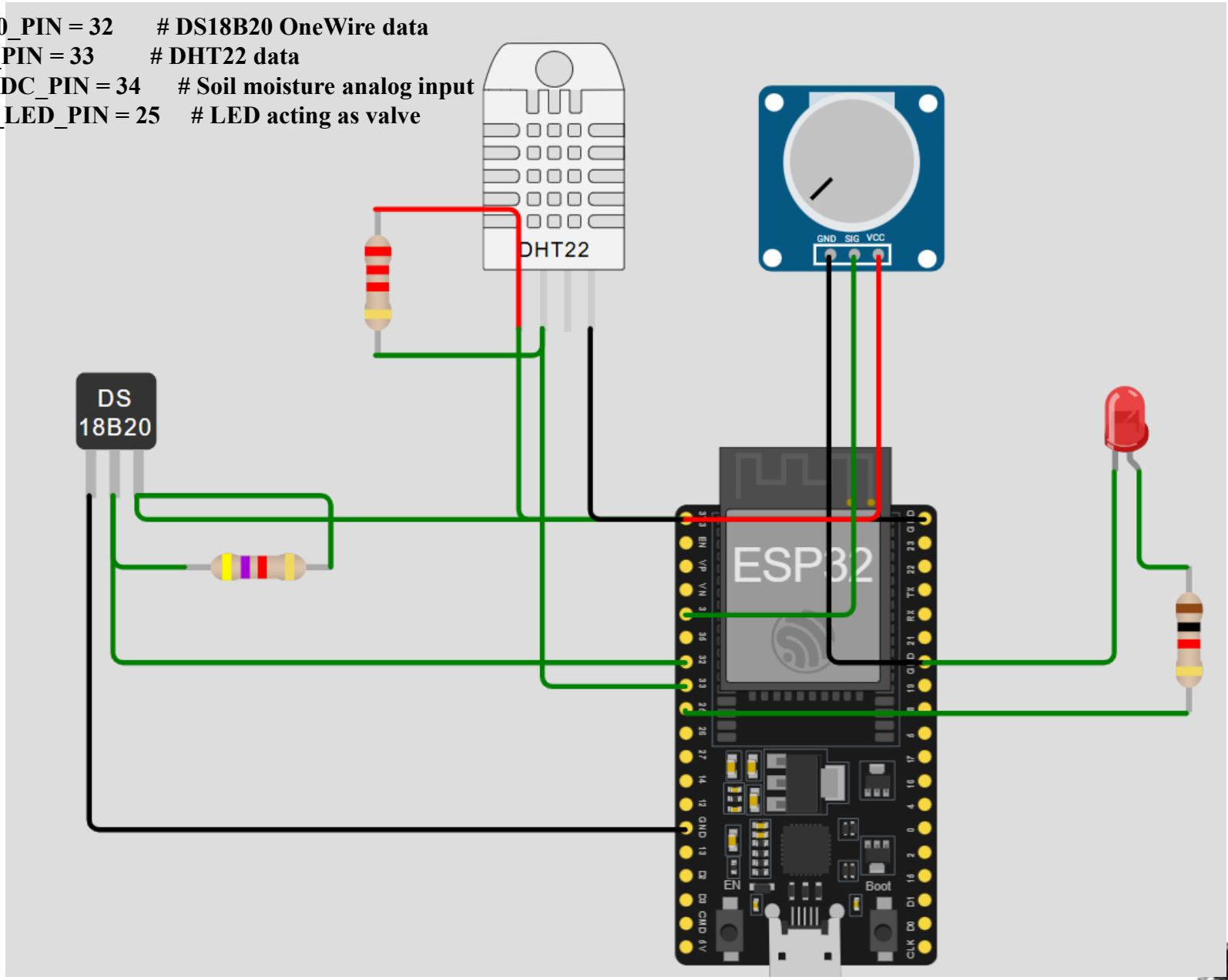
ESP32 pinout




wokwi.com
debian
tb (Channel) - ALLT...
Υπηρεσία Προσωπ...
Welcome to Epipha...
tutorialspoint.com c...
[lubuntu] HOWTO: L...
Linux Deploy - Εφα...
Interfacing an SPI A...
Building
JET BRAINS
Wokwi for JetBrains
See Wokwi Pricing Plans
Unlock advanced features, faster builds, add wokwi for vs code and more
Start Your Project with MicroPython

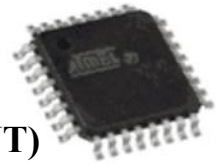

Starter Templates
NEW PROJECT
MicroPython on ESP32
MicroPython on ESP32-C3
MicroPython on ESP32-S3
MicroPython on Raspberry Pi Pico
MicroPython on Raspberry Pi Pico W

```
DS18B20_PIN = 32       # DS18B20 OneWire data
DHT22_PIN = 33         # DHT22 data
SOIL_ADC_PIN = 34      # Soil moisture analog input
VALVE_LED_PIN = 25     # LED acting as valve
```



Code:1/6

```
import network
import time
import ujson
import urequests
from machine import Pin, ADC
import onewire
import ds18x20
import dht
# ----------------------------
# Wokwi ESP32 Wi-Fi settings
# ----------------------------
WIFI_SSID = "Wokwi-GUEST"
WIFI_PASSWORD = ""
# ----------------------------
# ThingsBoard settings
# ----------------------------
ACCESS_TOKEN ="xYBwl4h8CRMPqNqwKWMO"
THINGSBOARD_HOST = "parallela.math.uoi.gr"
THINGSBOARD_PORT = 8080
TELEMETRY_PATH =
"/api/v1/{}/telemetry".format(ACCESS_TOKEN)
BASE_URL = "http://{}:{}{}".format(
    THINGSBOARD_HOST,
    THINGSBOARD_PORT,
    TELEMETRY_PATH
)
# ----------------------------
# ESP32 pin settings
# ----------------------------
DS18B20_PIN = 32       # DS18B20 OneWire data
DHT22_PIN = 33         # DHT22 data
SOIL_ADC_PIN = 34      # Soil moisture analog input
VALVE_LED_PIN = 25     # LED acting as valve
# ----------------------------
# DS18B20 setup
# ----------------------------
ds_pin = Pin(DS18B20_PIN)
ds_sensor = ds18x20.DS18X20(onewire.OneWire(ds_pin))
ds_roms = ds_sensor.scan()
# ----------------------------
# DHT22 setup
# ----------------------------
dht_sensor = dht.DHT22(Pin(DHT22_PIN))
# ----------------------------
# Soil moisture ADC setup
# ----------------------------
soil_adc = ADC(Pin(SOIL_ADC_PIN))
soil_adc.atten(ADC.ATTN_11DB)      # suitable for 0–3.3 V
input
soil_adc.width(ADC.WIDTH_10BIT)    # ESP32 ADC
range: 0–1023
SOIL_WET_VALUE = 0
SOIL_DRY_VALUE = 1023
# ----------------------------
# Valve LED setup
# ----------------------------
valve_led = Pin(VALVE_LED_PIN, Pin.OUT)
valve_led.value(0)
```



Code:2/6

```
# ----------------------------
# Wi-Fi connection
# ----------------------------
def connect_wifi():
    wlan = network.WLAN(network.STA_IF)
    wlan.active(True)
    if not wlan.isconnected():
        print("Connecting to Wi-Fi", end="")
        wlan.connect(WIFI_SSID, WIFI_PASSWORD)
        timeout = 40
        while not wlan.isconnected() and timeout > 0:
            print(".", end="")
            time.sleep(0.5)
            timeout -= 1
    if wlan.isconnected():
        print("\nWi-Fi connected")
        print("Network config:", wlan.ifconfig())
        return True
    print("\nWi-Fi connection failed")
    return False
```

```
# ----------------------------
# Timestamp
# ----------------------------
def get_timestamp_ms():
    """
    Returns timestamp in milliseconds.

    No ntptime is used.
    This uses the ESP32 local RTC time.
    """
    seconds = time.time()

    # Some MicroPython ports use 2000-01-01 as epoch.
    # ThingsBoard expects Unix epoch timestamp in
milliseconds.
    if time.gmtime(0)[0] == 2000:
        seconds += 946684800

    return int(seconds * 1000)
```

Code:3/6

```
# ----------------------------
# Read DS18B20
# ----------------------------
def read_ds18b20():
    if not ds_roms:
        print("DS18B20 error: no device found")
        return None
    try:
        ds_sensor.convert_temp()
        time.sleep_ms(750)
        temperature = ds_sensor.read_temp(ds_roms[0])
        return round(temperature, 2)
    except Exception as e:
        print("DS18B20 read error:", e)
        return None
# ----------------------------
# Read DHT22
# ----------------------------
def read_dht22():
    try:
        dht_sensor.measure()
        return {
            "temperature": round(dht_sensor.temperature(), 2),
            "humidity": round(dht_sensor.humidity(), 2)
        }
    except Exception as e:
        print("DHT22 read error:", e)
        return {
            "temperature": None,
            "humidity": None
        }
```

```
# ----------------------------
# Read soil moisture
# ----------------------------
def read_soil_moisture():
    raw = soil_adc.read()   # 0–1023

    # low raw value  = wet soil
    # high raw value = dry soil
    moisture_percent = 100 - int(
        (raw - SOIL_WET_VALUE) * 100 /
        (SOIL_DRY_VALUE - SOIL_WET_VALUE))
    if moisture_percent < 0:
        moisture_percent = 0
    if moisture_percent > 100:
        moisture_percent = 100
    return raw, moisture_percent
```

Code:4/6

```
# ----------------------------
# Read all sensors and control valve LED
# ----------------------------
def read_sensors():
    ds18b20_temperature = read_ds18b20()
    dht_values = read_dht22()
    soil_raw, soil_percent = read_soil_moisture()
    # Soil sensor error detection for 0–1023 ADC range
    if soil_raw < 10 or soil_raw > 1015:
        soil_sensor_error = 1
    else:
        soil_sensor_error = 0
    # Valve control rule
    if soil_percent < 30 and soil_sensor_error == 0:
        valve_status = 1
    else:
        valve_status = 0
    # Drive valve LED
    valve_led.value(valve_status)
    telemetry = {
        "air_temperature": dht_values["temperature"],
        "ds18b20_temperature": ds18b20_temperature,
        "humidity": dht_values["humidity"],
        "soil_moisture_percent": soil_percent,
        "soil_moisture_raw": soil_raw,
        "soil_sensor_error": soil_sensor_error,
        "valve_status": valve_status
    }
    return telemetry
```

Code:5/6

```
# ----------------------------
# Send telemetry to ThingsBoard
# ----------------------------
def send_telemetry(values):
    timestamp = get_timestamp_ms()
    payload = {
        "ts": timestamp,
        "values": values
    }
    json_payload = ujson.dumps(payload)
    print("Timestamp:", timestamp)
    print("POST", BASE_URL)
    print("Payload:", json_payload)
    response = None
    try:
        response = urequests.post(
            BASE_URL,
            data=json_payload,
            headers={
                "Content-Type": "application/json"
            }
        )

        print("Response status code:", response.status_code)
        try:
            print("Response body:", response.text)
        except Exception as e:
            print("HTTP error:", e)
            return False
        finaly:
            if response:
                response.close()
```

```
# ----------------------------
# Main program
# ----------------------------
def main():
    print("Starting ESP32 Wokwi greenhouse telemetry")
    print("DS18B20 on GPIO32")
    print("DHT22 on GPIO33")
    print("Soil moisture ADC on GPIO34, range 0–1023")
    print("Valve LED on GPIO25")
    print("DS18B20 devices found:", ds_roms)
    if not connect_wifi():
        print("Cannot continue without Wi-Fi")
        return
    iteration = 0
    while True:
        iteration += 1
        print("\n--- Iteration {} ---".format(iteration))
        telemetry = read_sensors()
        print("Air temperature:", telemetry["air_temperature"], "C")
        print("DS18B20 temperature:", telemetry["ds18b20_temperature"], "C")
        print("Humidity:", telemetry["humidity"], "%")
        print("Soil raw:", telemetry["soil_moisture_raw"])
        print("Soil moisture:", telemetry["soil_moisture_percent"], "%")
        print("Soil sensor error:", telemetry["soil_sensor_error"])
        print("Valve status:", telemetry["valve_status"])
        if telemetry["valve_status"] == 1:
            print("Valve LED: ON")
        else:
            print("Valve LED: OFF")
        success = send_telemetry(telemetry)
        if success:
            print("Telemetry sent successfully")
        else:
            print("Failed to send telemetry")
        time.sleep(10)
main()
```


User:ma547@uoi.gr
Pass:123456!@#