Computadoras y Lenguajes de programación

Unidad 3
Fundamentos de Sistemas Embebidos

¿Qué es una computadora?

¿Qué es una computadora?

Programa (instrucciones)

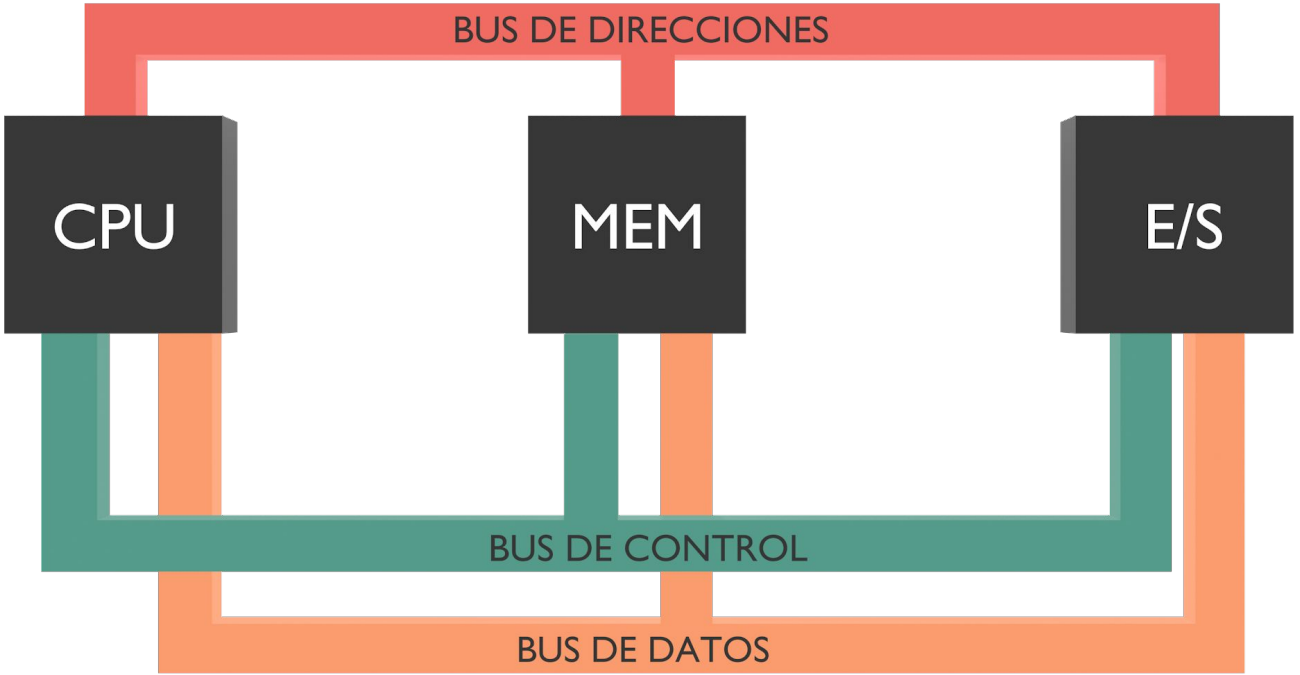
Datos

Resultado

Almacenamiento
Comunicaciones

Software: Soporte lógico
Hardware: Soporte físico

Bloques Funcionales



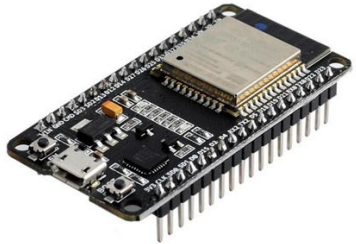
Tipos de computadoras

Propósitos generales (PC, Laptops, etc)

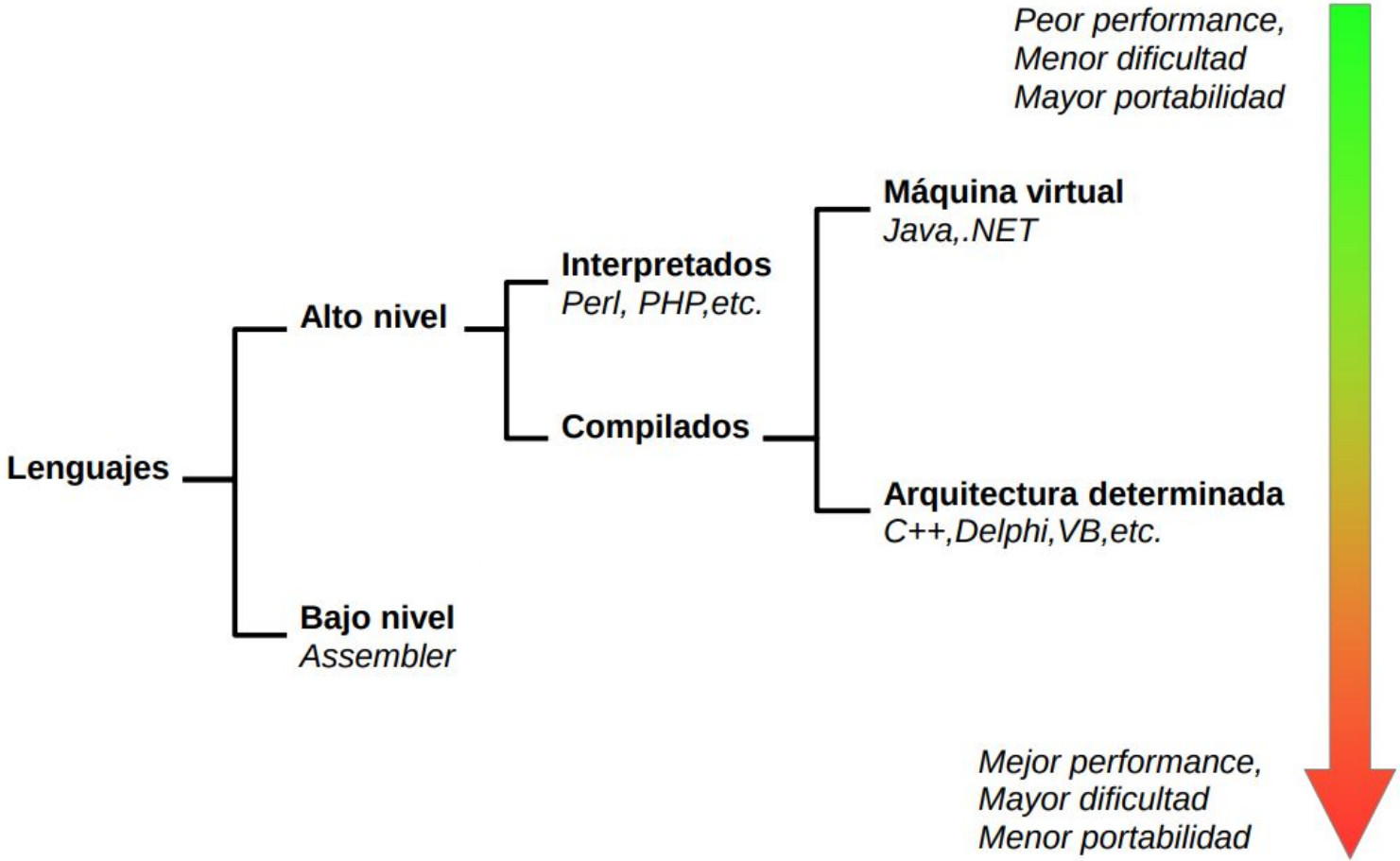
- Un único dispositivo permite cambiar su **aplicación** según la necesidad del usuario
  - Jugar
  - Acceder a la web
  - Correo electrónico
  - Apps de oficina
  - Programar
- Suele tener un sistema operativo
- Se puede expandir la capacidad de memoria

Fines específicos (microcontroladores)

- Un dispositivo posee un único **programa**
  - Heladeras
  - Autos
  - Controles industriales
- No siempre poseen un sistema operativo (Bare-Metal)
- No siempre se puede extender la capacidad de memoria



Clasificación de los lenguajes de programación



Alto nivel

```
1  int main(){
2      int a=8;
3      int b=10;
4      int c=0;
5
6     while (a>0){
7         c=c+b;
8         a--;
9     }
10
11    return c;
12 }
```

Bajo nivel

```
addi x2 x2 -32
sw x8 28 x2
addi x8 x2 32
addi x15 x0 8
sw x15 -20 x8
addi x15 x0 10
sw x15 -28 x8
sw x0 -24 x8
jal x0 32
lw x14 -24 x8
lw x15 -28 x8
add x15 x14 x15
sw x15 -24 x8
lw x15 -20 x8
addi x15 x15 -1
sw x15 -20 x8
lw x15 -20 x8
blt x0 x15 -32
lw x15 -24 x8
addi x10 x15 0
lw x8 28 x2
addi x2 x2 32
jalr x0 x1 0
```

Máquina

```
10074:      fe010113
10078:      00812e23
1007c:      02010413
10080:      00800793
10084:      fef42623
10088:      00a00793
1008c:      fef42223
10090:      fe042423
10094:      0200006f
10098:      fe842703
1009c:      fe442783
100a0:      00f707b3
100a4:      fef42423
100a8:      fec42783
100ac:      fff78793
100b0:      fef42623
100b4:      fec42783
100b8:      fef040e3
100bc:      fe842783
100c0:      00078513
100c4:      01c12403
100c8:      02010113
100cc:      00008067
```

Programas Compilados

Alto nivel

```
1  int main(){
2      int a=8;
3      int b=10;
4      int c=0;
5
6      while (a>0){
7          c=c+b;
8          a--;
9      }
10
11     return c;
12 }
```

Necesita ser traducido para ser ejecutado (compilado o interpretado)

Bajo nivel

```
addi x2 x2 -32
sw x8 28 x2
addi x8 x2 32
addi x15 x0 8
sw x15 -20 x8
addi x15 x0 10
sw x15 -28 x8
sw x0 -24 x8
jal x0 32
lw x14 -24 x8
lw x15 -28 x8
add x15 x14 x15
sw x15 -24 x8
lw x15 -20 x8
addi x15 x15 -1
sw x15 -20 x8
lw x15 -20 x8
blt x0 x15 -32
lw x15 -24 x8
addi x10 x15 0
lw x8 28 x2
addi x2 x2 32
jalr x0 x1 0
```

Necesita ser traducido para ser ejecutado (ensamblado)

Máquina

```
10074:    fe010113
10078:    00812e23
1007c:    02010413
10080:    00800793
10084:    fef42623
10088:    00a00793
1008c:    fef42223
10090:    fe042423
10094:    0200006f
10098:    fe842703
1009c:    fe442783
100a0:    00f707b3
100a4:    fef42423
100a8:    fec42783
100ac:    fff78793
100b0:    fef42623
100b4:    fec42783
100b8:    fef040e3
100bc:    fe842783
100c0:    00078513
100c4:    01c12403
100c8:    02010113
100cc:    00008067
```

Es entendido por la CPU

Programas Compilados

Programas Compilados

Alto nivel

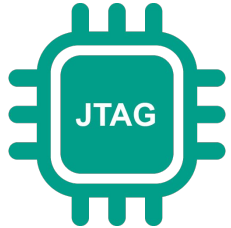
```
1  int main(){
2      int a=8;
3      int b=10;
4      int c=0;
5
6      while (a>0){
7          c=c+b;
8          a--;
9      }
10
11     return c;
12 }
```

Bajo nivel

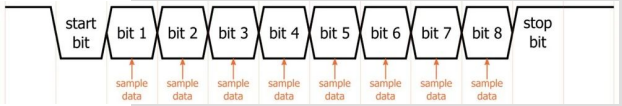
```
addi x2 x2 -32
sw x8 28 x2
addi x8 x2 32
addi x15 x0 8
sw x15 -20 x8
addi x15 x0 10
sw x15 -28 x8
sw x0 -24 x8
jal x0 32
lw x14 -24 x8
lw x15 -28 x8
add x15 x14 x15
sw x15 -24 x8
lw x15 -20 x8
addi x15 x15 -1
sw x15 -20 x8
lw x15 -20 x8
blt x0 x15 -32
lw x15 -24 x8
addi x10 x15 0
lw x8 28 x2
addi x2 x2 32
jalr x0 x1 0
```

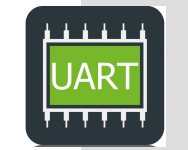
Máquina

```
10074:      fe010113
10078:      00812e23
1007c:      02010413
10080:      00800793
10084:      fef42623
10088:      00a00793
1008c:      fef42223
10090:      fe042423
10094:      0200006f
10098:      fe842703
1009c:      fe442783
100a0:      00f707b3
100a4:      fef42423
100a8:      fec42783
100ac:      fff78793
100b0:      fef42623
100b4:      fec42783
100b8:      fef040e3
100bc:      fe842783
100c0:      00078513
100c4:      01c12403
100c8:      02010113
100cc:      00008067
```



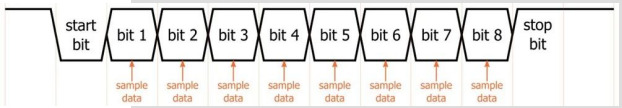
```python
import machine
from machine import Pin
import time

ledRojo = Pin(12, Pin.OUT)
while True:
    ledRojo.value(1)
    time.sleep(1)
    ledRojo.value(0)
    time.sleep(1)
```

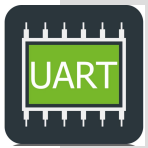




Programas Interpretados

```
1  import machine
2  from machine import Pin
3  import time
4
5  ledRojo = Pin(12, Pin.OUT)
6  while True:
7    ledRojo.value(1)
8    time.sleep(1)
9    ledRojo.value(0)
10   time.sleep(1)
```





El código fuente (source code) se transmite al intérprete que se encuentra corriendo en el dispositivo (ESP32). El intérprete utiliza el puerto serie (UART) para comunicarse con la computadora. Puede ejecutar línea a línea el código fuente o almacenarlo en un archivo y luego ejecutarlo. **El intérprete en sí mismo es un programa compilado que está ejecutando en el ESP32.** Se encarga de interpretar nuestro código fuente y ejecutarlo línea a línea.

Programas Interpretados

python™

MicroPython

Python es un lenguaje de programación interpretado. Comenzó en los 90s y soporta múltiples paradigmas de programación (estructurada, objetos, funcional). Al ser interpretado, en cualquier lugar donde se pueda ejecutar el intérprete de python entonces se puede correr código escrito en python. El intérprete en sí mismo es un programa escrito en C, y puede interactuar con el sistema operativo (DLLs, bibliotecas so, etc). Posee una extensa biblioteca estándar de funciones pero puede ser expandido fácilmente utilizando un manejador de paquetes. Al ser un lenguaje interpretado, su mayor problema es la performance. Si bien puede generarse una versión objeto de un programa python, el mismo es ejecutado por el intérprete. Python tiene un Zen con 20 principios que influyen en su diseño.

Es una versión de python (mayormente 3.4) reducida para poder correr en microcontroladores. Dado que NO interactúa contra un sistema operativo, o sea es Bare Metal, necesitamos utilizar una versión de MicroPython compilada a medida para el microcontrolador que usemos.
MicroPython contiene algunas bibliotecas que son específicas para MicroPython. Ej: La interacción con pines de entrada/salida.

```
>>> help()
Welcome to MicroPython on the ESP32!

For generic online docs please visit http://docs.micropython.org/

For access to the hardware use the 'machine' module:

import machine
pin12 = machine.Pin(12, machine.Pin.OUT)
pin12.value(1)
pin13 = machine.Pin(13, machine.Pin.IN, machine.Pin.PULL_UP)
print(pin13.value())
i2c = machine.I2C(scl=machine.Pin(21), sda=machine.Pin(22))
i2c.scan()
i2c.writeto(addr, b'1234')
i2c.readfrom(addr, 4)

Basic WiFi configuration:

import network
sta_if = network.WLAN(network.STA_IF); sta_if.active(True)
sta_if.scan()                             # Scan for available access points
sta_if.connect("<AP_name>", "<password>") # Connect to an AP
sta_if.isconnected()                      # Check for successful connection

Control commands:
  CTRL-A        -- on a blank line, enter raw REPL mode
  CTRL-B        -- on a blank line, enter normal REPL mode
  CTRL-C        -- interrupt a running program
  CTRL-D        -- on a blank line, do a soft reset of the board
  CTRL-E        -- on a blank line, enter paste mode

For further help on a specific object, type help(obj)
For a list of available modules, type help('modules')
>>>
```

https://micropython.org/download/

Se debe elegir un binario (.bin) compatible con el chip o el kit de desarrollo. Obviamente al ser de código abierto podemos compilar el intérprete, pero mayormente vamos a utilizar un kit donde sabemos que existe soporte.
Una vez descargado el binario, se escribe el mismo en la memoria flash del microcontrolador.

Los pasos para escribir el intérprete en la flash suelen ser:

- Descargar esptool
- Conectar el kit al puerto USB-Serie
- Detectar el puerto COM (windows) donde aparece el kit.
- Borrar la flash con flash_erase
- Escribir el binario

Luego podemos usar un programa terminal (PuTTY) para conectarse al intérprete. Recomendamos Thonny para programar.

Conceptos Básicos

- Cada línea de código es leída y ejecutada por el intérprete.
- El código se organiza en bloques, los cuales mantienen sangría (indentación)
  - El nivel de indentación crece hacia la derecha
  - Todo código dentro de un nivel de indentación se considera un bloque.
  - **Si** el bloque de código es condicional:
    - Crece la indentación para crear un nuevo bloque
  - **Sino:**
    - También crece la indentación
  - Luego vuelve a la normalidad para finalizar la condición.
- Existen palabras reservadas (ej: import, from, while)
- Utilizamos Pines para Entrada/Salida
- Por ahora solo vamos a simular lógica combinatoria.
- Soporta comentarios con #
- Con print(“....”) podemos imprimir datos en pantalla
- `from machine import Pin`

```
1   import machine
2   from machine import Pin
3   import time
4
5   ledRojo = Pin(12, Pin.OUT)
6   while True:
7     ledRojo.value(1)
8     time.sleep(1)
9     ledRojo.value(0)
10    time.sleep(1)
```

4.4 DC Characteristics (3.3 V, 25 °C)

Table 4-4. DC Characteristics (3.3 V, 25 °C)

Parameter	Description	Min	Typ	Max	Unit
C_{IN}	Pin capacitance	—	2	—	pF
V_{IH}	High-level input voltage	0.75 × VDD[1]	—	VDD[1]+ 0.3	V
V_{IL}	Low-level input voltage	–0.3	—	0.25 × VDD[1]	V
I_{IH}	High-level input current	—	—	50	nA
I_{IL}	Low-level input current	—	—	50	nA
V_{OH}[2]	High-level output voltage	0.8 × VDD[1]	—	—	V
V_{OL}[2]	Low-level output voltage	—	—	0.1 × VDD[1]	V
I_{OH}	High-level source current (VDD[1]= 3.3 V, V_{OH} >= 2.64 V, PAD_DRIVER = 3)	—	40	—	mA
I_{OL}	Low-level sink current (VDD[1]= 3.3 V, V_{OL} = 0.495 V, PAD_DRIVER = 3)	—	28	—	mA
R_{PU}	Internal weak pull-up resistor	—	45	—	kΩ
R_{PD}	Internal weak pull-down resistor	—	45	—	kΩ
V_{IH_nRST}	Chip reset release voltage CHIP_EN voltage is within the specified range)	0.75 × VDD[1]	—	VDD[1]+ 0.3	V
V_{IL_nRST}	Chip reset voltage (CHIP_EN voltage is within the specified range)	–0.3	—	0.25 × VDD[1]	V

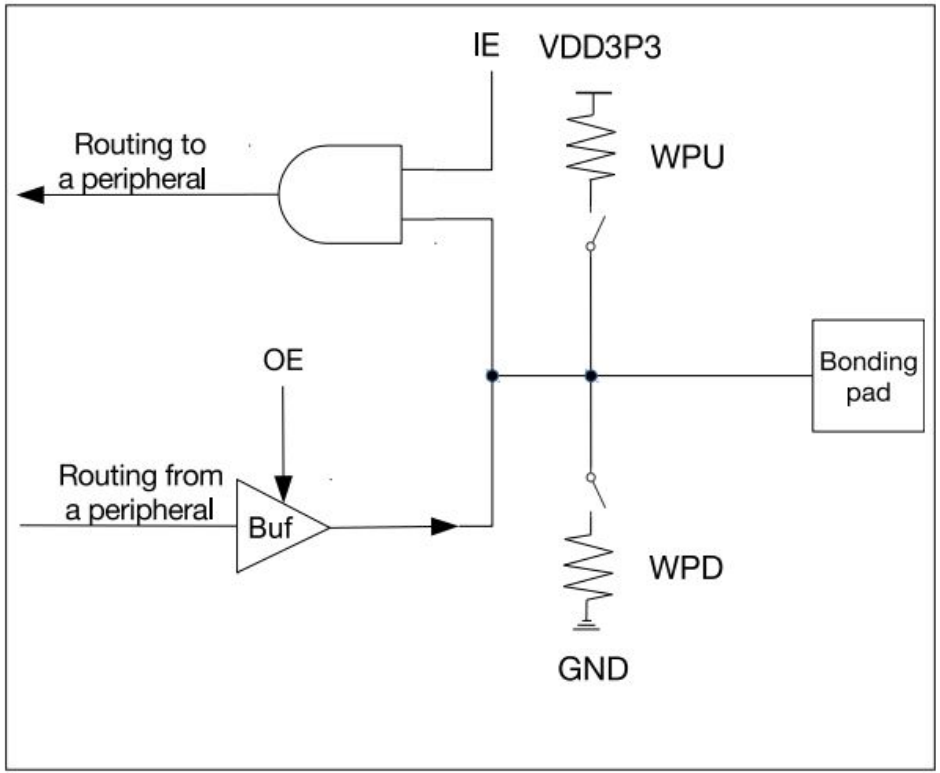
[1] VDD – voltage from a power pin of a respective power domain.

[2] V_{OH} and V_{OL} are measured using high-impedance load.

Table 4-1. Absolute Maximum Ratings

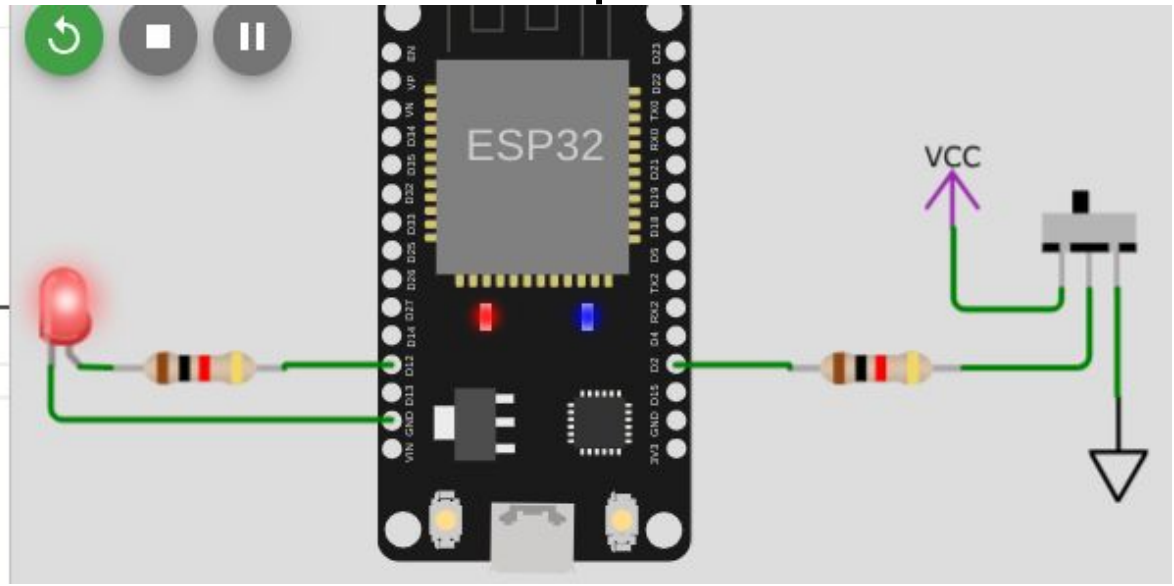
Parameter	Description	Min	Max	Unit
Input power pins[1]	Allowed input voltage	–0.3	3.6	V
I_{output}[2]	Cumulative IO output current	—	1000	mA
T_{STORE}	Storage temperature	–40	150	°C

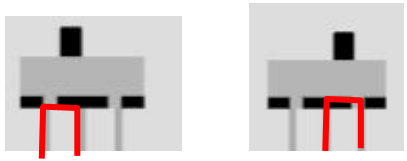
Pines de E/S (GPIO) - Entrada con/sin Pull Up o Down



Pines de E/S (GPIO) - Entrada sin Pull Up o Down

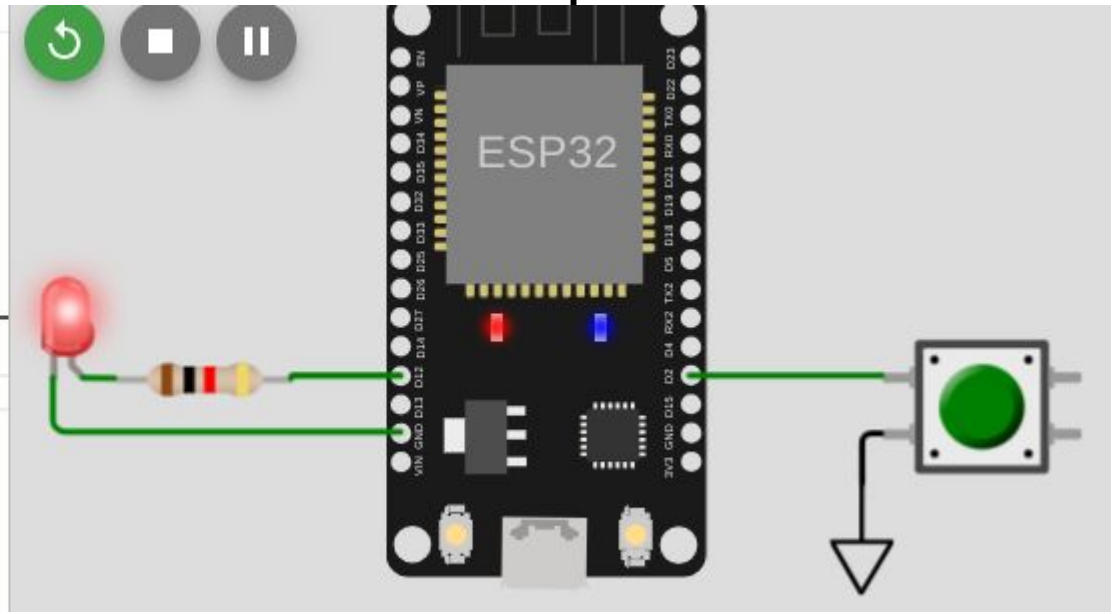
```
1   import machine
2   from machine import Pin
3   #definimos salida
4   ledRojo = Pin(12, Pin.OUT)
5   #definimos entrada (sin pullup)
6   boton = Pin(2,Pin.IN)
7   while True:
8     valor = boton.value()
9     ledRojo.value(valor)
10
```



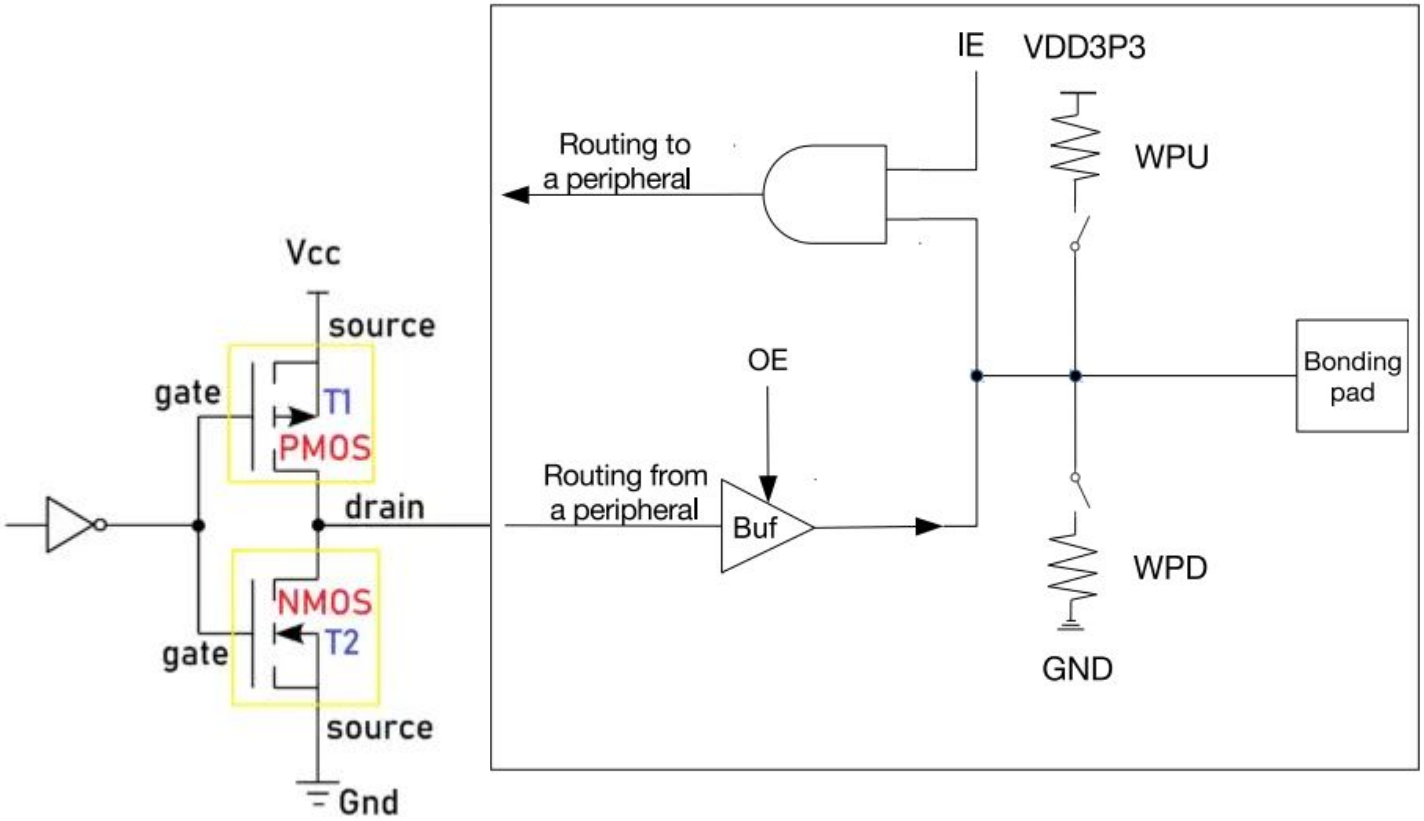


Pines de E/S (GPIO) - Entrada con Pull Up

```
1   import machine
2   from machine import Pin
3   #definimos salida
4   ledRojo = Pin(12, Pin.OUT)
5   #definimos entrada (con pullup)
6   boton = Pin(2,Pin.IN,Pin.PULL_UP)
7   while True:
8     valor = boton.value()
9     ledRojo.value(valor)
10
```

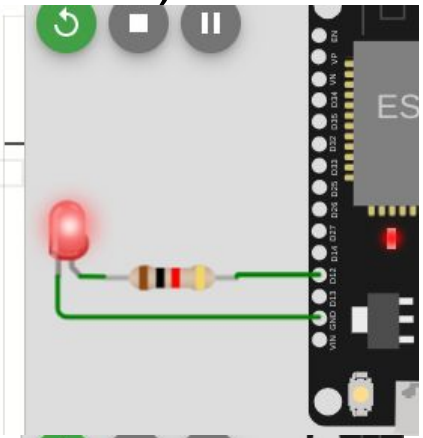


Pines de E/S (GPIO) - Salida Push Pull

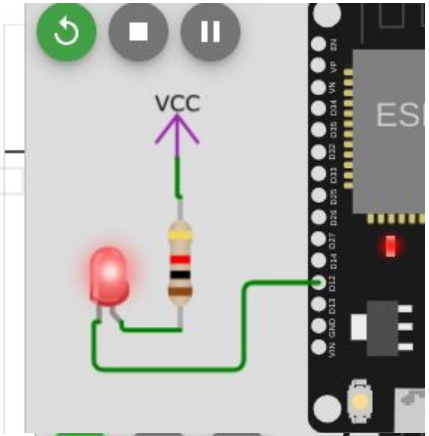


Pines de E/S (GPIO) - Salida Push Pull

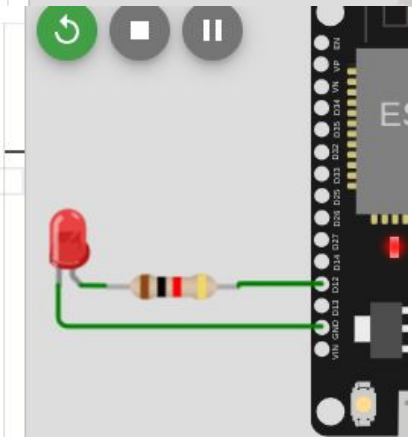
```
1  import machine
2  from machine import Pin
3  #definimos salida
4  ledRojo = Pin(12, Pin.OUT)
5  while True:
6    ledRojo.value(1)
```



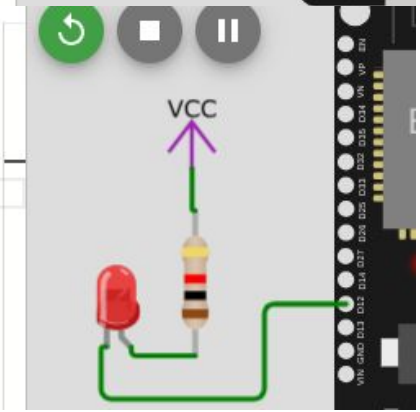
```
1  import machine
2  from machine import Pin
3  #definimos salida
4  ledRojo = Pin(12, Pin.OUT)
5  while True:
6    ledRojo.value(0)
```



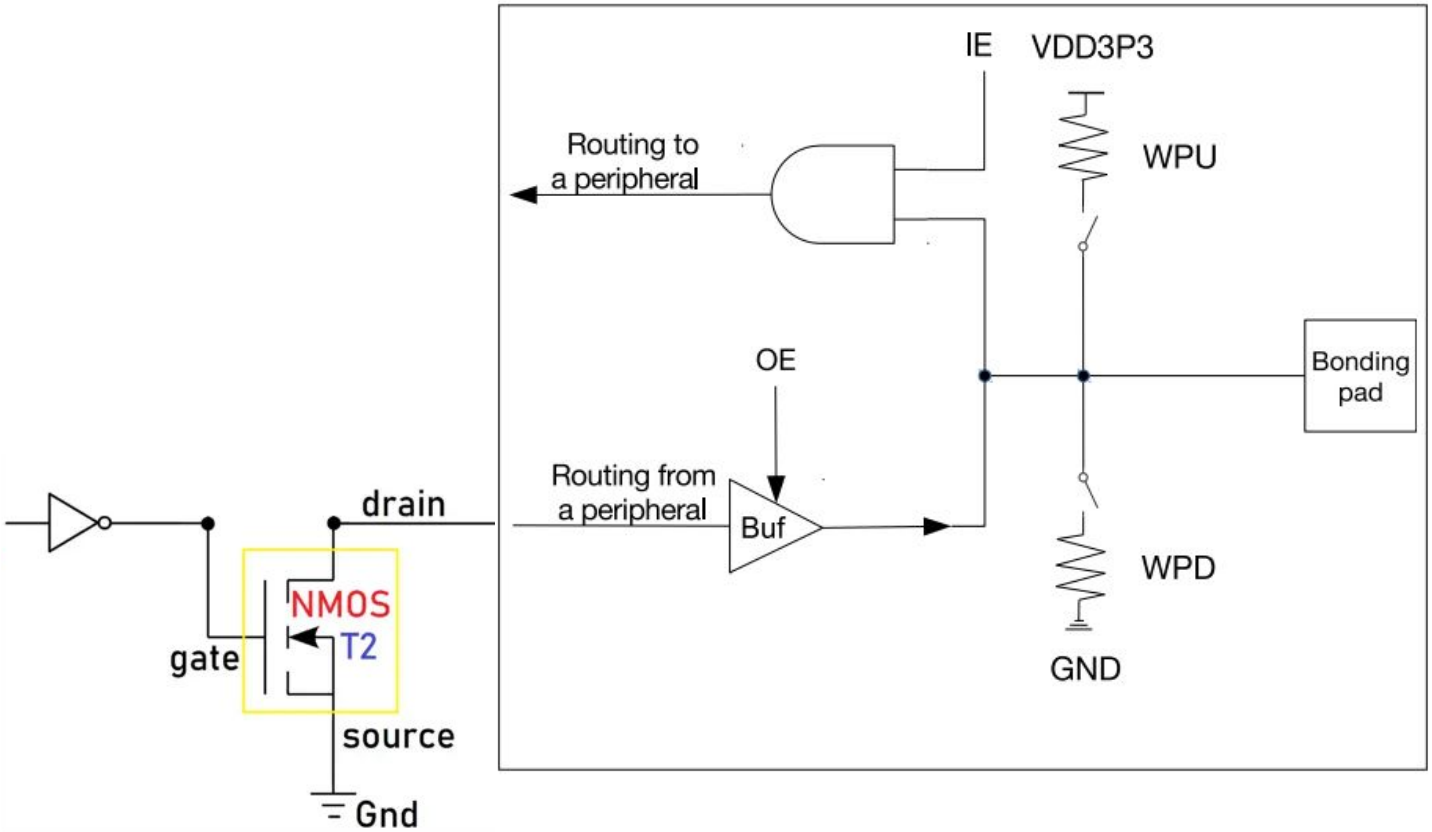

```
1  import machine
2  from machine import Pin
3  #definimos salida
4  ledRojo = Pin(12, Pin.OUT)
5  while True:
6    ledRojo.value(0)
```



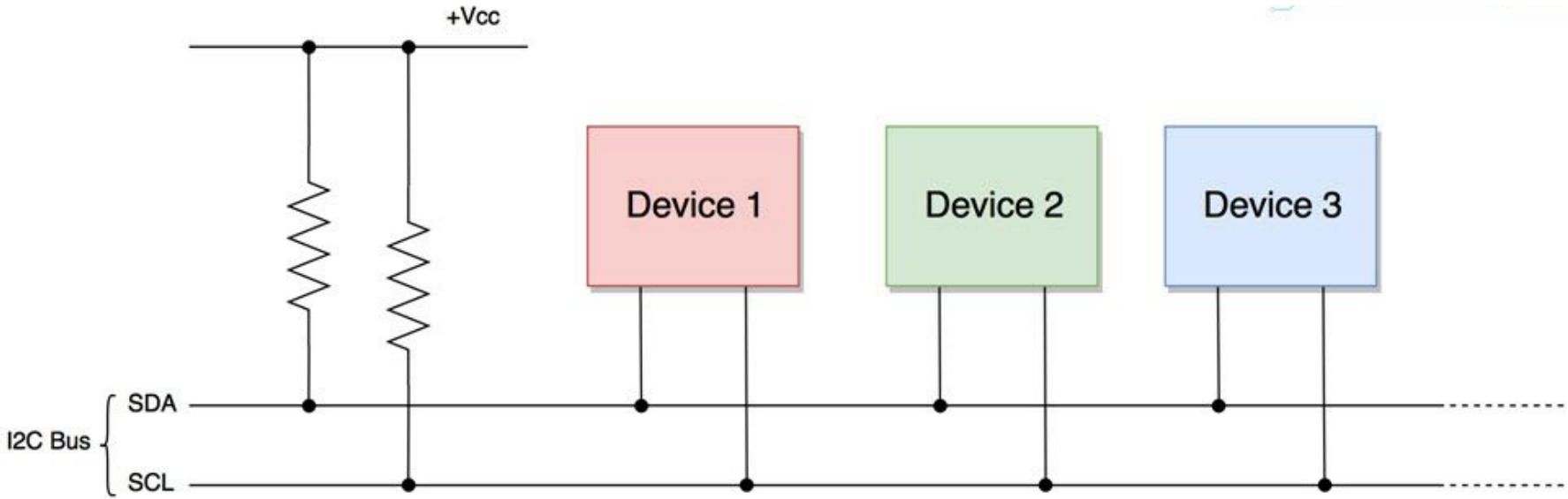
```
1  import machine
2  from machine import Pin
3  #definimos salida
4  ledRojo = Pin(12, Pin.OUT)
5  while True:
6    ledRojo.value(1)
```



Pines de E/S (GPIO) - Salida Open Drain

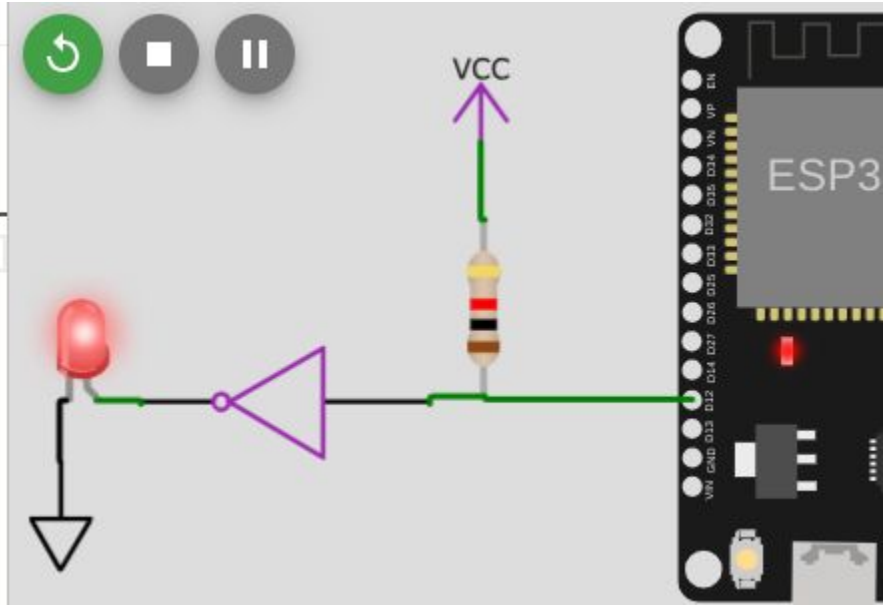


Pines de E/S (GPIO) - Salida Open Drain



Pines de E/S (GPIO) - Salida Open Drain

```
1  import machine
2  from machine import Pin
3  #definimos salida
4  salida = Pin(12, Pin.OPEN_DRAIN)
5  while True:
6    salida.value(0)
```



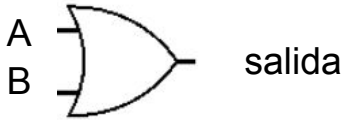
Operadores Booleanos

Podemos utilizar las operaciones básicas del álgebra de boole (AND, OR, NOT).

Ejemplos

```
salida = ( not A and B )
```


```
salida = ( A or B )
```

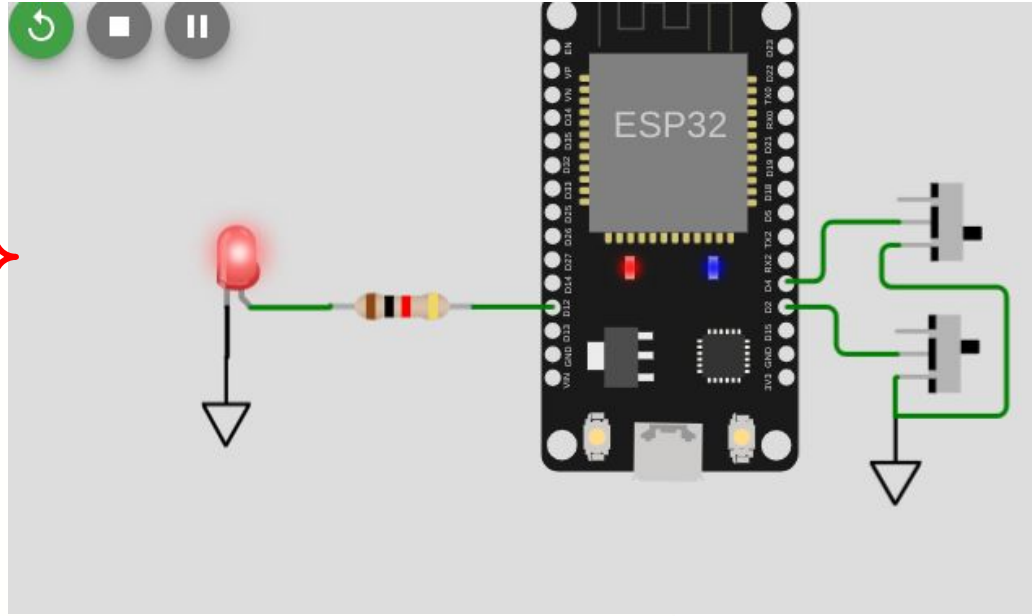


La variable salida almacena el resultado de la operación booleana temporalmente, luego podemos escribir este resultado en una salida.

```
A = pinA.value()
B = pinB.value()
salida = ( A and B )
led.value( salida)
```

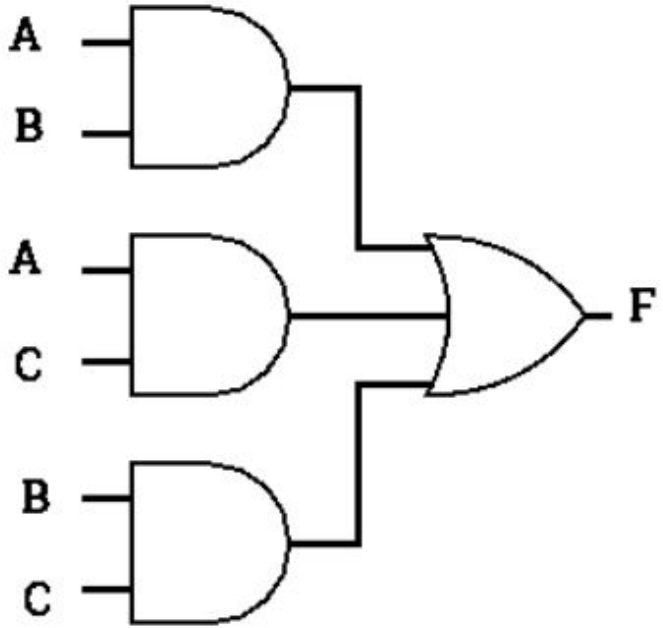
https://wokwi.com/ Simulamos ESP32 con MicroPython

```
1   import machine
2   from machine import Pin
3   #definimos salida
4   ledRojo = Pin(12, Pin.OUT)
5   SliderA = Pin(2,Pin.IN,Pin.PULL_UP)
6   SliderB = Pin(4,Pin.IN,Pin.PULL_UP)
7   while True:
8     A = SliderA.value()
9     B = SliderB.value()
10    #Tabla de verdad de XOR
11    #   A  |  B  |  XOR
12    #  -----------------
13    #   0  |  0  |  0
14    #   0  |  1  |  1
15    #   1  |  0  |  1
16    #   1  |  1  |  0
17    xor = (not A and B) or ( A and not B)
18    ledRojo.value(xor)
19    print("A="+str(bool(A))+" B="+str(bool(B))+" Xor="+str(bool(xor)))
```



Ejemplo: Función mayoría de 3 bits

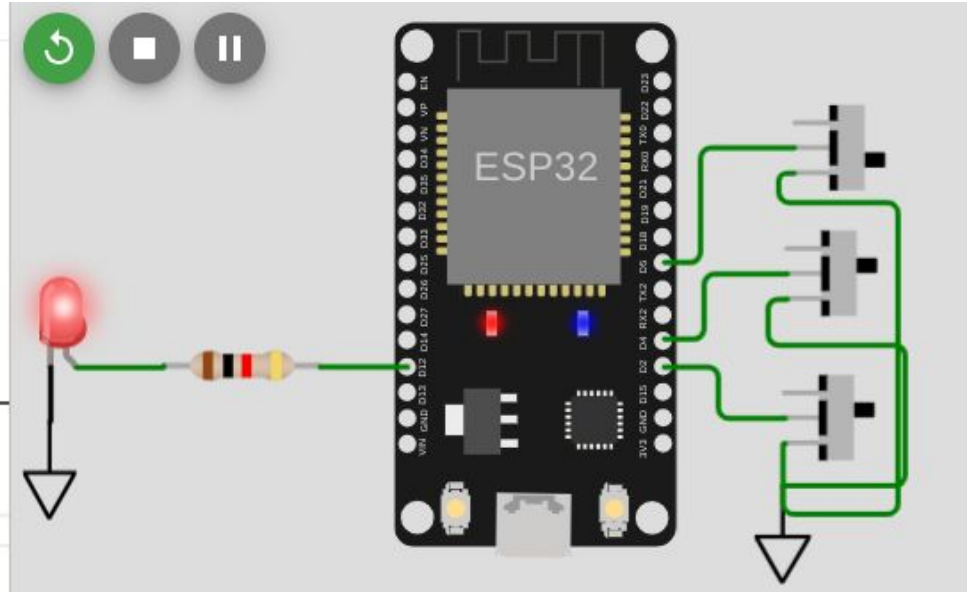
$$F(A,B,C)=B.C+A.C+A.B$$



Ejemplo: Función mayoría de 3 bits

$$F(A,B,C)=B.C+A.C+A.B$$

```
1   import machine
2   from machine import Pin
3   #definimos salida
4   ledRojo = Pin(12, Pin.OUT)
5   SliderA = Pin(2,Pin.IN,Pin.PULL_UP)
6   SliderB = Pin(4,Pin.IN,Pin.PULL_UP)
7   SliderC = Pin(5,Pin.IN,Pin.PULL_UP)
8   while True:
9     A = SliderA.value()
10    B = SliderB.value()
11    C = SliderC.value()
12
13    mayoria = ( (A and B) or (A and C) or (B and C))
14    ledRojo.value(mayoria)
15
```



Implementar los siguientes circuitos en micropython

- Compuerta AND de 3 entradas
- Sumador de dos números de 1 bit (dos entradas, dos salidas)
- Sumador de tres números de 1 bit (tres entradas, dos salidas)
- Sumador de dos números de 2 bits (cuatro entradas, tres salidas)
- Codificador simple (tres entradas, dos salidas)
- Codificador simple (siete entradas, tres salidas)
- Decodificador (dos entradas, cuatro salidas)
- Multiplexor (cuatro entradas de datos, dos de selección, una salida)
- Demultiplexor (una entrada de datos, dos de selección, cuatro salidas)

Micropython en ESP32

micropython.org

MicroPython

DOWNLOAD DOCS DISCORD DISCUSSIONS WIKI STORE

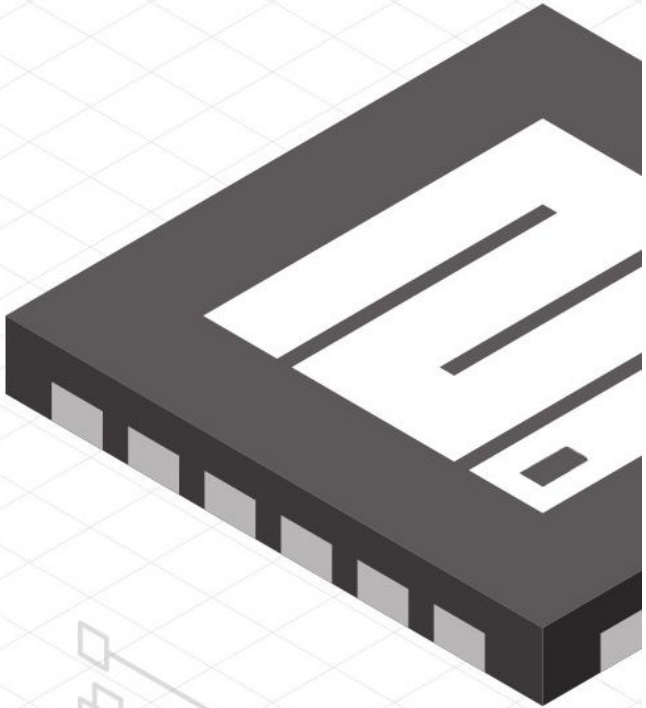
MicroPython

MicroPython is a lean and efficient implementation of the Python 3 programming language that includes a small subset of the Python standard library and is optimised to run on microcontrollers and in constrained environments.

The MicroPython pyboard is a compact electronic circuit board that runs MicroPython on the bare metal, giving you a low-level Python operating system that can be used to control all kinds of electronic projects.

MicroPython is packed full of advanced features such as an interactive prompt, arbitrary precision integers, closures, list comprehension, generators, exception handling and more. Yet it is compact enough to fit and run within just 256k of code space and 16k of RAM.

MicroPython aims to be as compatible with normal Python as possible to allow you to transfer code with ease from the desktop to a microcontroller or embedded system.



TEST DRIVE A PYBOARD

BUY A PYBOARD

USE MICROPYTHON ONLINE

MicroPython downloads

MicroPython is developed using git for source code management, and the master repository can be found on GitHub at github.com/micropython/micropython.

The full source-code distribution of the latest version is available for download here:

- micropython-1.20.0.tar.xz (73MiB)
- micropython-1.20.0.zip (151MiB)

Daily snapshots of the GitHub repository (not including submodules) are available from this server:

- micropython-master.zip
- pyboard-master.zip

Firmware for various microcontroller ports and boards are built automatically on a daily basis and can be found below.

Filter by:
Port: esp32 [x]
Feature: Audio Codec, BLE, Battery Charging, CAN, Camera, DAC, Display, Dual-core, Environment Sensor, Ethernet, External Flash, External RAM, Feather, IMU, JST-PH, JST-SH, LoRa, Microphone, PoE, RGB LED, SDCard, Secure Element, USB, USB-C, WiFi, microSD, mikroBUS
Vendor: Actinius, Adafruit, Arduino, BBC, Espressif, Espruino, Fez, George Robotics, HydraBus, I-SYST, LEGO, LILYGO, Laird Connectivity, LimiFrog, M5 Stack, Makerdiary, McHobby, Microchip, MikroElektronika, MiniFig Boards, NXP, Netduino, Nordic Semiconductor, OLIMEX, PJRC, Particle, Pimoroni, Pycom, Raspberry Pi, Renesas Electronics, ST Microelectronics, Seeed Studio, Silicognition, Sparkfun, Unexpected Maker, VCC-GND Studio, Vekatech, WeAct, Wemos, Wireless-Tag, Wiznet, nullbits, u-blox
MCU: cc3200, esp32, esp32c3, esp32s2, esp32s3, esp8266, mimxrt, nrf51, nrf52, nrf91, ra4m1, ra4w1, ra6m1, ra6m2, ra6m5, rp2040, samd21, samd51, stm32f0, stm32f4, stm32f7, stm32g0, stm32g4, stm32h7, stm32l0, stm32l1, stm32l4, stm32wb, stm32wl

Firmware

Releases

v1.20.0 (2023-04-26) .bin [.elf] [.map] [Release notes] (latest)
v1.19.1 (2022-06-18) .bin [.elf] [.map] [Release notes]
v1.18 (2022-01-17) .bin [.elf] [.map] [Release notes]
v1.17 (2021-09-02) .bin [.elf] [.map] [Release notes]
v1.16 (2021-06-23) .bin [.elf] [.map] [Release notes]
v1.15 (2021-04-18) .bin [.elf] [.map] [Release notes]
v1.14 (2021-02-02) .bin [.elf] [.map] [Release notes]
v1.13 (2020-09-02) .bin [.elf] [.map] [Release notes]
v1.12 (2019-12-20) .bin [.elf] [.map] [Release notes]

Nightly builds

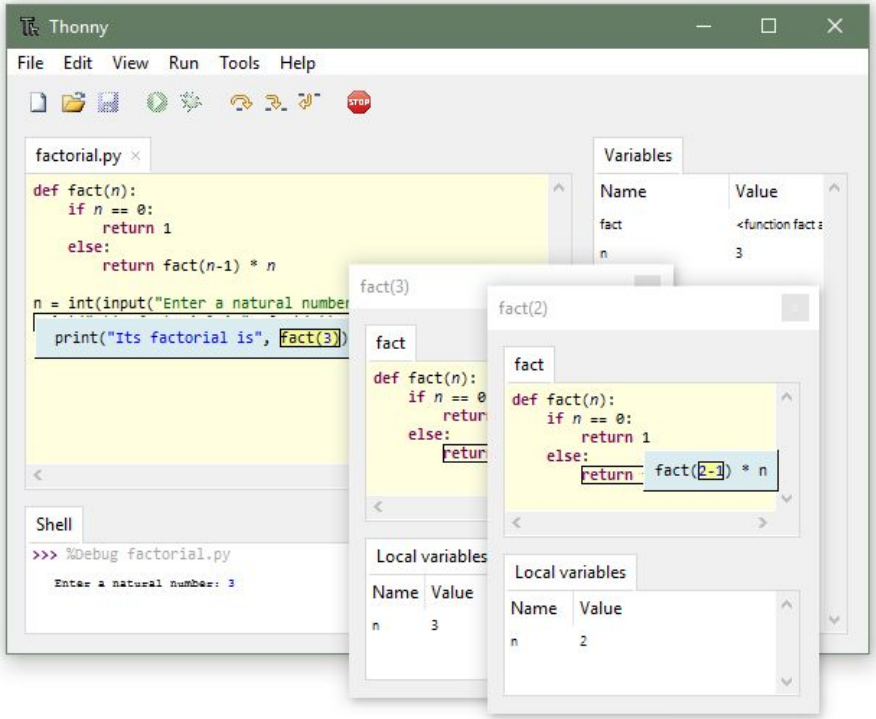
v1.20.0-396-g1dedb65e6 (2023-08-24) .bin [.app-bin] [.elf] [.map]
v1.20.0-394-g326dfd2a8 (2023-08-23) .bin [.app-bin] [.elf] [.map]
v1.20.0-384-g2919a9fbf (2023-08-23) .bin [.app-bin] [.elf] [.map]
v1.20.0-379-ga18d62e06 (2023-08-16) .bin [.app-bin] [.elf] [.map]

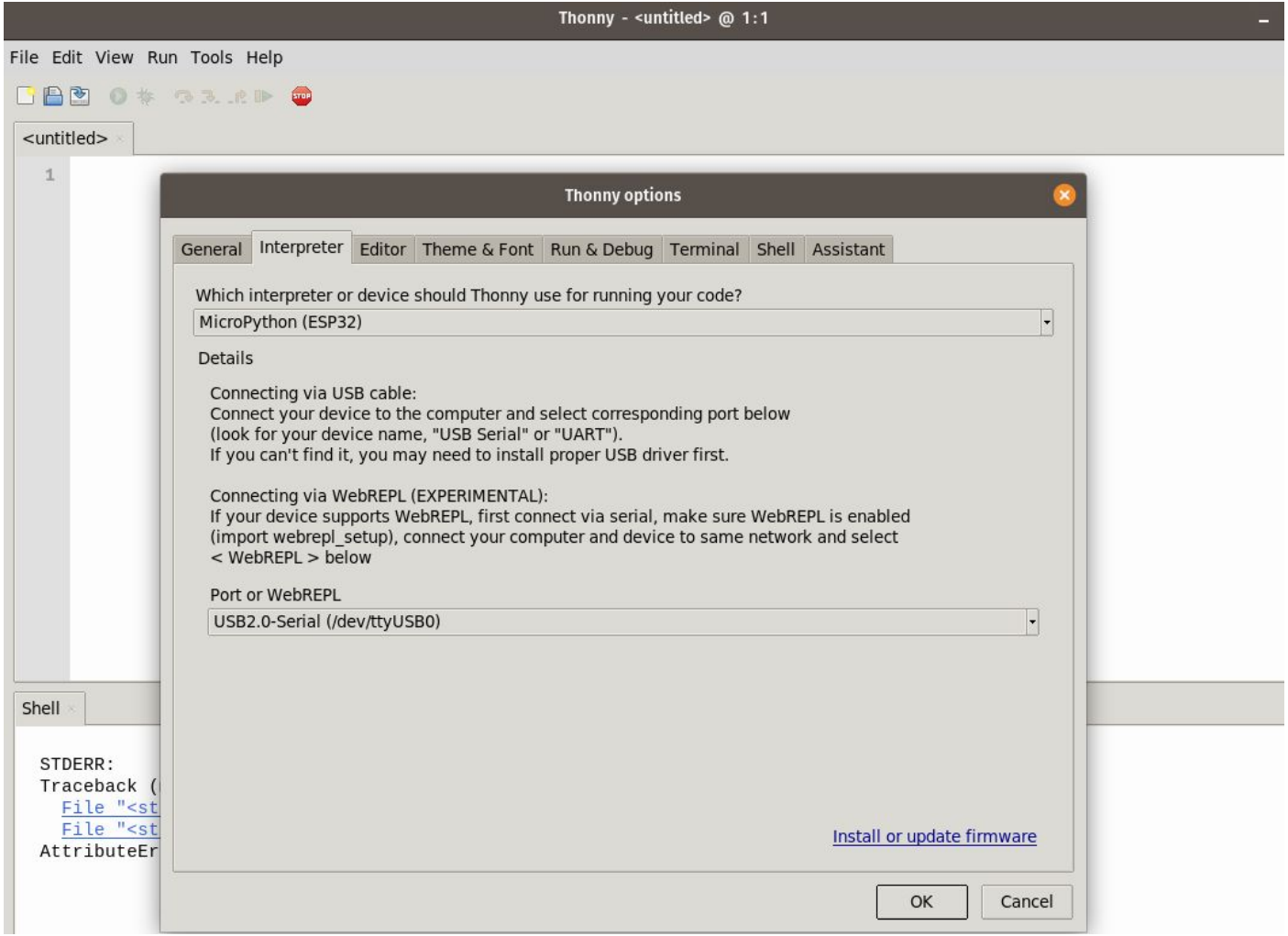
thonny.org

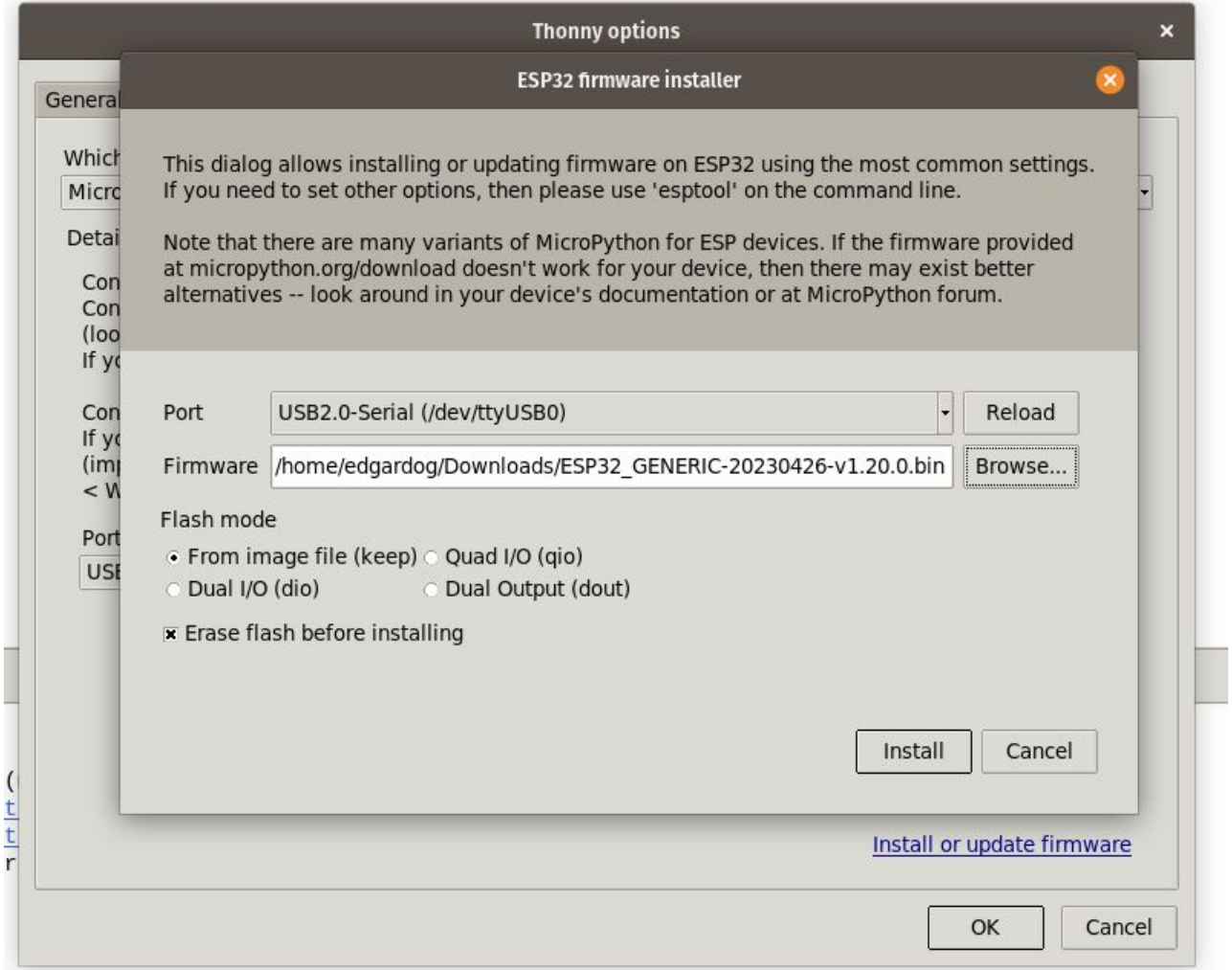
Thonny

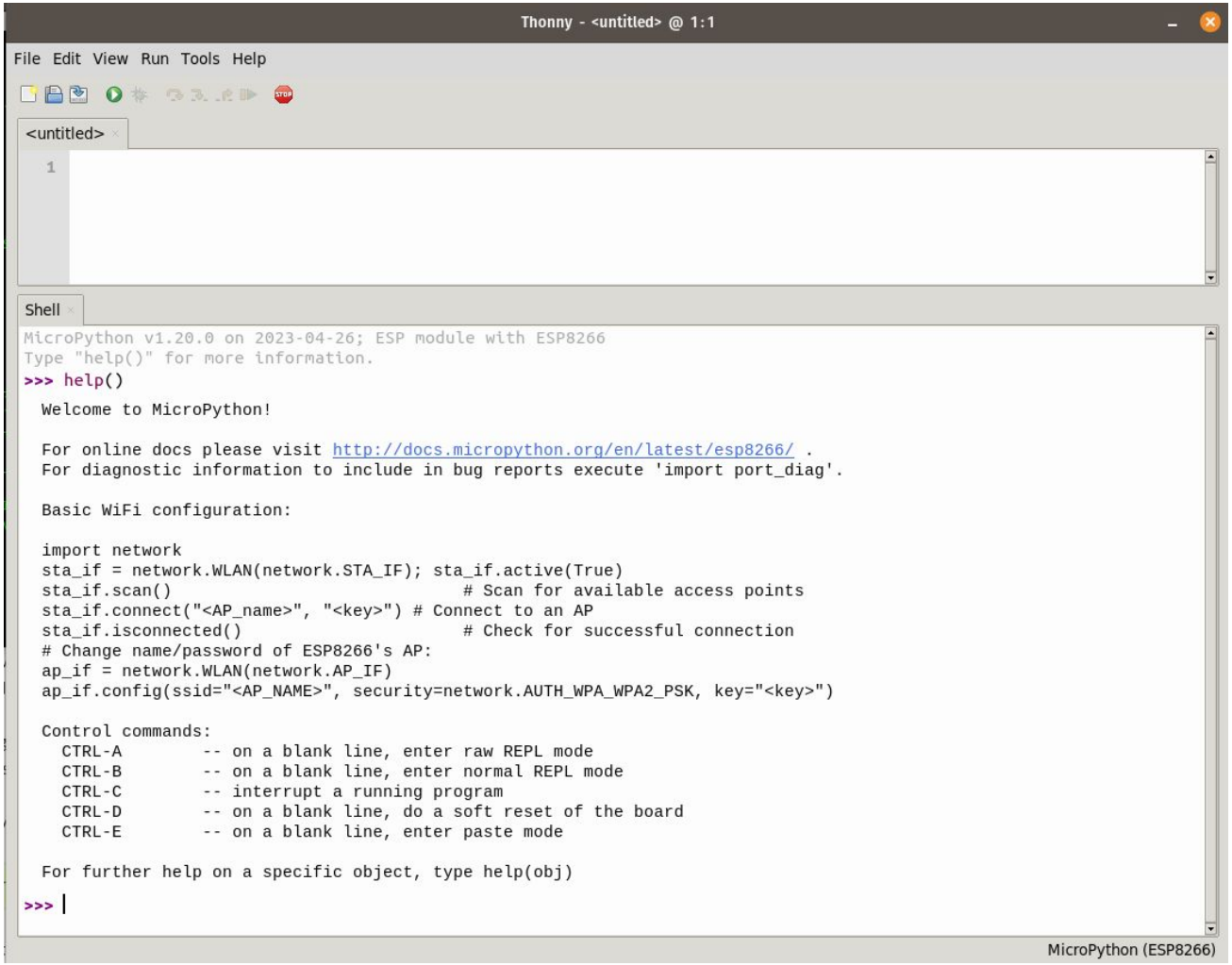
Python IDE for beginners

Download version 4.1.2 for Windows • Mac • Linux




Thonny - <untitled> @ 1:1
File Edit View Run Tools Help
<untitled>
1
Thonny options
General
Interpreter
Editor
Theme & Font
Run & Debug
Terminal
Shell
Assistant
Which interpreter or device should Thonny use for running your code?
MicroPython (ESP32)
Details
Connecting via USB cable:
Connect your device to the computer and select corresponding port below
(look for your device name, "USB Serial" or "UART").
If you can't find it, you may need to install proper USB driver first.
Connecting via WebREPL (EXPERIMENTAL):
If your device supports WebREPL, first connect via serial, make sure WebREPL is enabled
(import webrepl_setup), connect your computer and device to same network and select
< WebREPL > below
Port or WebREPL
USB2.0-Serial (/dev/ttyUSB0)
Install or update firmware
OK
Cancel
Shell
STDERR:
Traceback (
File "<st
File "<st
AttributeEr


Thonny options
ESP32 firmware installer
This dialog allows installing or updating firmware on ESP32 using the most common settings.
If you need to set other options, then please use 'esptool' on the command line.
Note that there are many variants of MicroPython for ESP devices. If the firmware provided at micropython.org/download doesn't work for your device, then there may exist better alternatives -- look around in your device's documentation or at MicroPython forum.
Port
USB2.0-Serial (/dev/ttyUSB0)
Reload
Firmware
/home/edgardog/Downloads/ESP32_GENERIC-20230426-v1.20.0.bin
Browse...
Flash mode
From image file (keep)
Quad I/O (qio)
Dual I/O (dio)
Dual Output (dout)
Erase flash before installing
Install
Cancel
Install or update firmware
OK
Cancel


Thonny - <untitled> @ 1:1
File Edit View Run Tools Help
<untitled>
1
Shell
MicroPython v1.20.0 on 2023-04-26; ESP module with ESP8266
Type "help()" for more information.
>>> help()
Welcome to MicroPython!
For online docs please visit http://docs.micropython.org/en/latest/esp8266/ .
For diagnostic information to include in bug reports execute 'import port_diag'.
Basic WiFi configuration:
import network
sta_if = network.WLAN(network.STA_IF); sta_if.active(True)
sta_if.scan() # Scan for available access points
sta_if.connect("<AP_name>", "<key>") # Connect to an AP
sta_if.isconnected() # Check for successful connection
# Change name/password of ESP8266's AP:
ap_if = network.WLAN(network.AP_IF)
ap_if.config(ssid="<AP_NAME>", security=network.AUTH_WPA_WPA2_PSK, key="<key>")
Control commands:
CTRL-A -- on a blank line, enter raw REPL mode
CTRL-B -- on a blank line, enter normal REPL mode
CTRL-C -- interrupt a running program
CTRL-D -- on a blank line, do a soft reset of the board
CTRL-E -- on a blank line, enter paste mode
For further help on a specific object, type help(obj)
>>>
MicroPython (ESP8266)

File Edit View Run Tools Help

<untitled> *

```
1 import machine
2 from machine import Pin
3 import time
4 ledAzul = Pin(14,Pin.OUT)
5 while True:
6     ledAzul.value(0)
7     time.sleep(1)
8     ledAzul.value(1)
9     time.sleep(1)
```

Where to save to?

This computer

MicroPython device

Shell

<untitled> *

```
1 import machine
2 from machine import Pin
3 import time
4 ledAzul = Pin(14,Pin.OUT)
5 while True:
6     ledAzul.value(0)
7     time.sleep(1)
8     ledAzul.value(1)
9     time.sleep(1)
```

Shell

MicroPython v1.20.0 on 2023-04
Type "help()" for more informa

Save to MicroPython device

MicroPython device

Name	Size (bytes)
boot.py	230

File name: Prueba.py

OK Cancel

[Prueba.py]

```
1 import machine
2 from machine import Pin
3 import time
4 ledAzul = Pin(2,Pin.OUT)
5 while True:
6     ledAzul.value(0)
7     time.sleep(1)
8     ledAzul.value(1)
9     time.sleep(1)
```