

La **Programación Orientada a Objetos (POO)** es un **paradigma de programación**, es decir una forma de programar basada en **objetos** y sus interacciones.

Los **objetos** son *elementos* que agrupan valores – *atributos*– y funcionalidades –*métodos*– y son creados (generalmente) a través de unas *plantillas* de software denominadas **clases**.

Cuando nos iniciamos en la **POO** es mejor dejar las definiciones formales de lado y utilizar un ejemplo práctico para entender que son y para que sirven las **clases** y los **objetos**. Para ello vamos a programar el encendido de un LED con el *microcontrolador* ESP32.

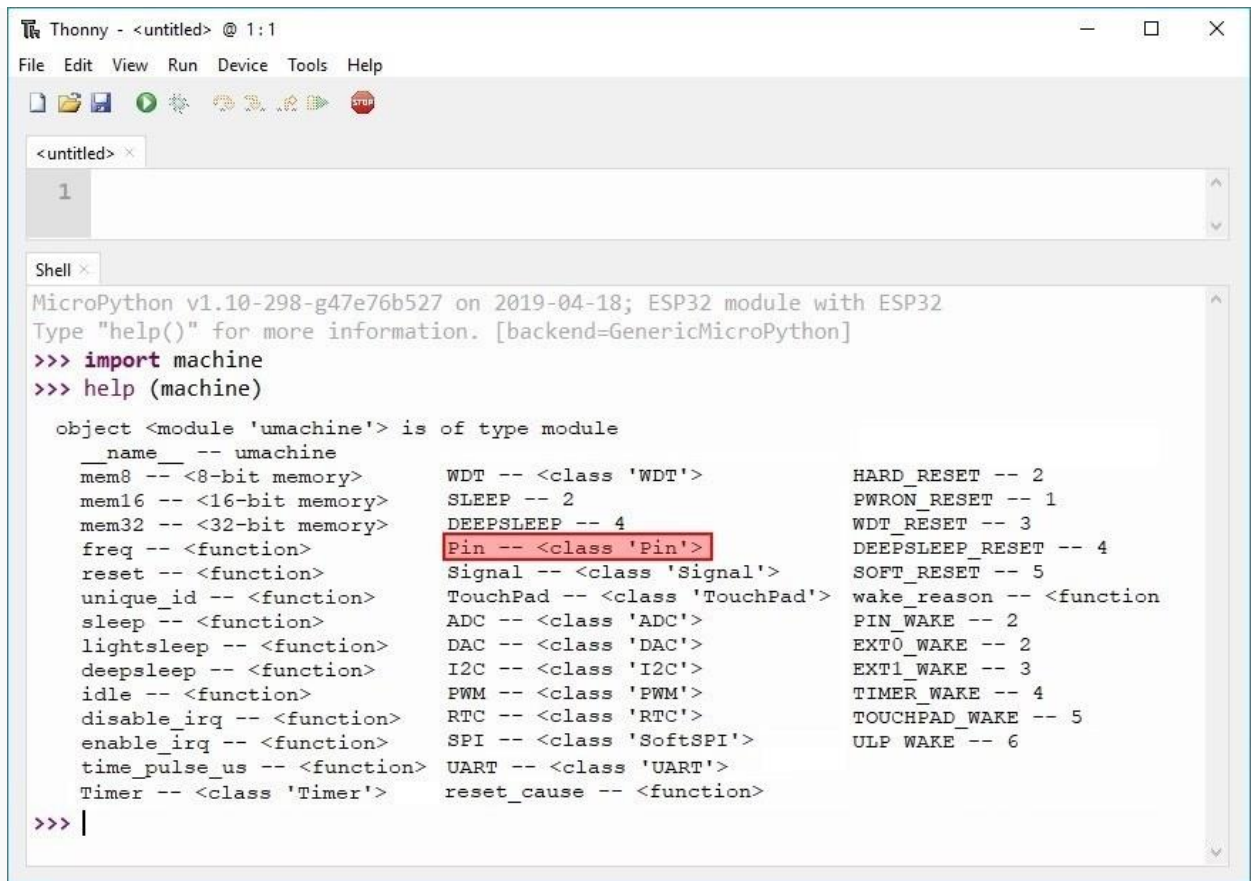
Como habíamos dicho, el *módulo* **machine** de MicroPython, que es la “**biblioteca**” de **funciones** y **clases**, para controlar el *hardware* del microcontrolador.

EL MÓDULO «MACHINE»

El *módulo* **machine** dispone de 12 **funciones** y 12 **clases**:

- **Funciones:** *reset()*, *reset_cause()*, *enable_irq()*, *disable_irq()*, *freq()*, *idle()*, *time_pulse_us()*, *sleep()*, *lightsleep()*, *deepsleep()*, *wake_reason()* y *unique_id()*.
- **Clases:** *Timer()*, *WDT()*, **Pin()**, *Signal()*, *TouchPad()*, *ADC()*, *DAC()*, *I2C()*, *PWM()*, *RTC()*, *SPI()* y *UART()*.

Podemos ver este listado en *Thonny* utilizando la función `help()`, después de importar el *módulo*:



```
Thonny - <untitled> @ 1:1
File Edit View Run Device Tools Help

<untitled> x
1


Shell x
MicroPython v1.10-298-g47e76b527 on 2019-04-18; ESP32 module with ESP32
Type "help()" for more information. [backend=GenericMicroPython]
>>> import machine
>>> help(machine)

object <module 'umachine'> is of type module
  __name__ -- umachine
  mem8 -- <8-bit memory>
  mem16 -- <16-bit memory>
  mem32 -- <32-bit memory>
  freq -- <function>
  reset -- <function>
  unique_id -- <function>
  sleep -- <function>
  lightsleep -- <function>
  deepsleep -- <function>
  idle -- <function>
  disable_irq -- <function>
  enable_irq -- <function>
  time_pulse_us -- <function>
  Timer -- <class 'Timer'>
  WDT -- <class 'WDT'>
  SLEEP -- 2
  DEEPSLEEP -- 4
  Pin -- <class 'Pin'>
  Signal -- <class 'Signal'>
  TouchPad -- <class 'TouchPad'>
  ADC -- <class 'ADC'>
  DAC -- <class 'DAC'>
  I2C -- <class 'I2C'>
  PWM -- <class 'PWM'>
  RTC -- <class 'RTC'>
  SPI -- <class 'SoftSPI'>
  UART -- <class 'UART'>
  reset_cause -- <function>
  HARD_RESET -- 2
  PWRON_RESET -- 1
  WDT_RESET -- 3
  DEEPSLEEP_RESET -- 4
  SOFT_RESET -- 5
  wake_reason -- <function>
  PIN_WAKE -- 2
  EXT0_WAKE -- 2
  EXT1_WAKE -- 3
  TIMER_WAKE -- 4
  TOUCHPAD_WAKE -- 5
  ULP_WAKE -- 6

>>> |
```

De entre las **clases** del *módulo*, la **clase** `Pin()` se utiliza para crear **objetos** para controlar las *entradas/salidas* del *microcontrolador* (General Purpose Input/Output –**GPIO**-), permitiendo tanto **leer la presencia/ausencia de tensión de entrada** como **suministrar una tensión (voltaje) de salida de 3.3V**, que nos servirá para encender el LED.

Adicionalmente si pedimos información sobre la **clase** `Pin()` podemos ver que tiene 5 métodos –`Pin.init()`, `Pin.value()`, `Pin.off()`, `Pin.on()` y `Pin.irq()`–:



```
Thonny - <untitled> @ 1:1
File Edit View Run Device Tools Help

<untitled> x
1

Shell x
MicroPython v1.10-298-g47e76b527 on 2019-04-18; ESP32 module with ESP32
Type "help()" for more information. [backend=GenericMicroPython]
>>> import machine
>>> help(machine.Pin)

object <class 'Pin'> is of type type
init -- <function>          IN -- 1          PULL_HOLD -- 4
value -- <function>        OUT -- 3          IRQ_RISING -- 1
off -- <function>          OPEN_DRAIN -- 7      IRQ_FALLING -- 2
on -- <function>           PULL_UP -- 2       WAKE_LOW -- 4
irq -- <function>          PULL_DOWN -- 1      WAKE_HIGH -- 5

>>> |
```

La documentación completa y actualizada sobre la **clase** `Pin()` la podemos localizar en el apartado de las librerías específicas de *MicroPython*: <http://docs.micropython.org/en/latest/library/machine.Pin.html>.

LA CLASE «PIN()»

Para poder usar esta **clase**, necesitamos saber como funcionan su **constructor** y sus **métodos**.

El **constructor** es la subrutina cuya misión es inicializar el objeto (crear el objeto y establecer sus valores –atributos– y funcionalidades –métodos– iniciales mediante los argumentos). El **constructor** de la **clase** `Pin` es:

class machine.Pin(id, mode, pull, *, alt)

Los *argumentos* del **constructor** son:

- **id**: es el número del pin del *microcontrolador* ESP32 con el que se asocia el objeto. Los *pin*s disponibles son los de los siguientes rangos: 0-19, 21-23, 25-27, 32-39, pero no es recomendable el uso de los *pin*s 1 y 3 porque se utilizan en el puerto de comunicaciones *UART*, tampoco los *pin*s 6, 7, 8, 11, 16 y 17 que se utilizan

para conectar la *memoria flash*. Por último, el rango de pines 34-39 son únicamente entradas. En consecuencia los pines aconsejables para encender un LED son los siguientes: 2, 4-5, 9-10, 12-15, 18-19, 21-23, 25-27 y 32-33.

- **mode:** se refiere al modo en el que se puede configurar el pin. Los *argumentos* pueden ser **Pin.IN** –*pin de entrada*, para leer la tensión-, **Pin.OUT** –*pin de salida*, para suministrar una tensión- o **Pin.OPEN_DRAIN** –*pin de entrada o salida con colector abierto*-.
- **pull:** se refiere a la posibilidad de configurar resistencias para evitar errores en las lecturas por factores externos. Los *argumentos* pueden ser **None** -sin resistencias-, **Pin.PULL_UP** -con resistencia *pull-up* habilitada- o **Pin.PULL_DOWN** -con resistencia *pull-down* habilitada-.
- * **(value):** permite establecer tensión de salida. Los *argumentos* pueden ser «0» apagado -sin tensión- (0.0V) y «1» encendido -con tensión- (3.3V). Se pueden utilizar *booleanos* como *argumentos*.
- **alt:** identidad de una función alternativa.

Los *argumentos* para encender el LED azul integrado en la placa **DOIT ESP32 DEVKIT V1** serán los siguientes:

- **id** «2» ya que el LED está conectado con *pin* 02 (**GPIO 02**).
- **mode** «*Pin.OUT*» para hacer que el *pin* 02 se convierta en un *pin* de salida de tensión.
- **pull** «None» para no configurar resistencias *pull-up* o *pull-down* (solo son útiles en el caso de ser un *pin* de entrada –*Pin.IN*-). No es necesario definir este *argumento*.

- **value « 1 »** para encender el LED -salida de tensión de 3.3V-.

Utilizaremos el siguiente código en el caso de que queramos importar el **módulo** *machine* completo:

```
import machine
pin_02 = machine.Pin(2, mode=machine.Pin.OUT,
pull=None, value=1)
# pin_02 = machine.Pin(2, machine.Pin.OUT, value=1) es
equivalente
```

o el siguiente, si únicamente queremos importar la **clase** *Pin()* del **módulo** *machine*:

```
from machine import Pin
pin_02 = Pin(2, mode=Pin.OUT, pull=None, value=1)
# pin_02 = Pin(2, Pin.OUT, value=1) es equivalente
```

Una vez ejecutado el código habremos *inicializado* un **objeto**, al que se le ha asignado el **nombre** *pin_02*, cuyos **valores iniciales** convierten el pin 2 (GPIO 02) en un *pin de salida* y su estado es *encendido* (3.3V de tensión).

Los **métodos** que podemos utilizar una vez creado el **objeto** nos permiten *manipular* (modificar) sus funcionalidades. En la **clase** *Pin* tenemos los siguientes:

- **Pin.init():** modifica los argumentos iniciales del constructor. Se pueden modificar los argumentos **mode**, **pull**, ***(value)** y **alt** cuando se definen nuevos valores.
- **Pin.value():** permite leer la presencia/ausencia de tensión de entrada del pin (sin no se proporciona un argumento) o establecer una tensión de salida en

el pin proporcionando el argumento. Los argumentos pueden ser «0» apagado (0.0V) y «1» encendido (3.3V). Se pueden utilizar *booleanos* como argumentos.

- **Pin.off():** establece una tensión de salida del pin de 0.0V -apagado-.
- **Pin.on():** establece una tensión de salida del pin de 3.3V -encendido-.
- **Pin.irq():** habilita interrupciones externas en el pin , siempre y cuando se trate de un pin de entrada.

Podemos utilizar los **métodos** anteriores para apagar y volver a encender el LED de varias formas diferentes:

- Modificando los argumentos iniciales del **constructor** con el **método Pin.init()**:

```
pin_02.init(value=0)
```

```
pin_02.init(value=1)
```

- Estableciendo la tensión de salida del *pin* con el **método Pin.value()**:

```
pin_02.value(0)
```

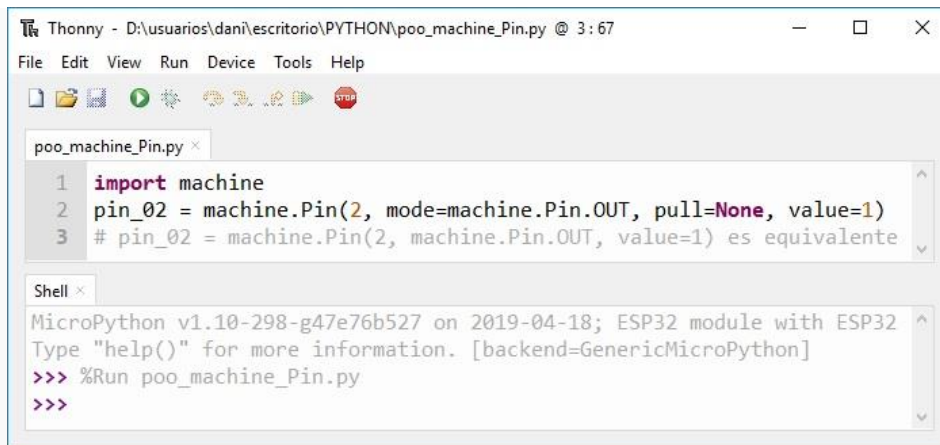
```
pin_02.value(1)
```

- Utilizando los **métodos Pin.off() y Pin.on()**:

```
pin_02.off()
```

```
pin_02.on()
```

La ejecución en **Thonny** del **constructor** para inicializar del **objeto** desde el **editor de código** se visualizará así:



```
Thonny - D:\usuarios\dani\escritorio\PYTHON\poo_machine_Pin.py @ 3:67
File Edit View Run Device Tools Help

poo_machine_Pin.py x
1 import machine
2 pin_02 = machine.Pin(2, mode=machine.Pin.OUT, pull=None, value=1)
3 # pin_02 = machine.Pin(2, machine.Pin.OUT, value=1) es equivalente

Shell x
MicroPython v1.10-298-g47e76b527 on 2019-04-18; ESP32 module with ESP32
Type "help()" for more information. [backend=GenericMicroPython]
>>> %Run poo_machine_Pin.py
>>>
```

Una vez *inicializado* se encenderá el LED.
Haciendo uso en el **intérprete activo (Shell)** de los diferentes **métodos**, a medida que los ejecutemos podremos apagar y encender el LED:



```
Thonny - D:\usuarios\dani\escritorio\PYTHON\poo_machine_Pin.py @ 3:67
File Edit View Run Device Tools Help

poo_machine_Pin.py x
1 import machine
2 pin_02 = machine.Pin(2, mode=machine.Pin.OUT, pull=None, value=1)
3 # pin_02 = machine.Pin(2, machine.Pin.OUT, value=1) es equivalente

Shell x
MicroPython v1.10-298-g47e76b527 on 2019-04-18; ESP32 module with ESP32
Type "help()" for more information. [backend=GenericMicroPython]
>>> %Run poo_machine_Pin.py
>>> pin_02.init(value=0)
>>> pin_02.init(value=1)
>>> pin_02.value(0)
>>> pin_02.value(1)
>>> pin_02.off()
>>> pin_02.on()
>>>
```

Hemos visto de una forma práctica que:

- Las **clases** son las *plantillas* de software que se utilizan para crear **objetos**.
- Podemos crear tantos **objetos** con una **clase** como sean necesarios.
- Los **métodos** son las *funciones* de la **clase** que nos permiten:
 - Cuando se *inicializa* el **objeto**, establecer sus funcionalidades iniciales (comportamiento).
 - Una vez *inicializado* el **objeto**, manipularlo (modificar sus funcionalidades).

- El **constructor** de la **clase** es la *subrutina* cuya misión es *inicializar* el **objeto**, es decir, crearlo y establecer sus valores –*atributos*– y funcionalidades –*métodos*– iniciales.
- Los **objetos**, por lo tanto, son los *elementos* que agrupan valores y funcionalidades.