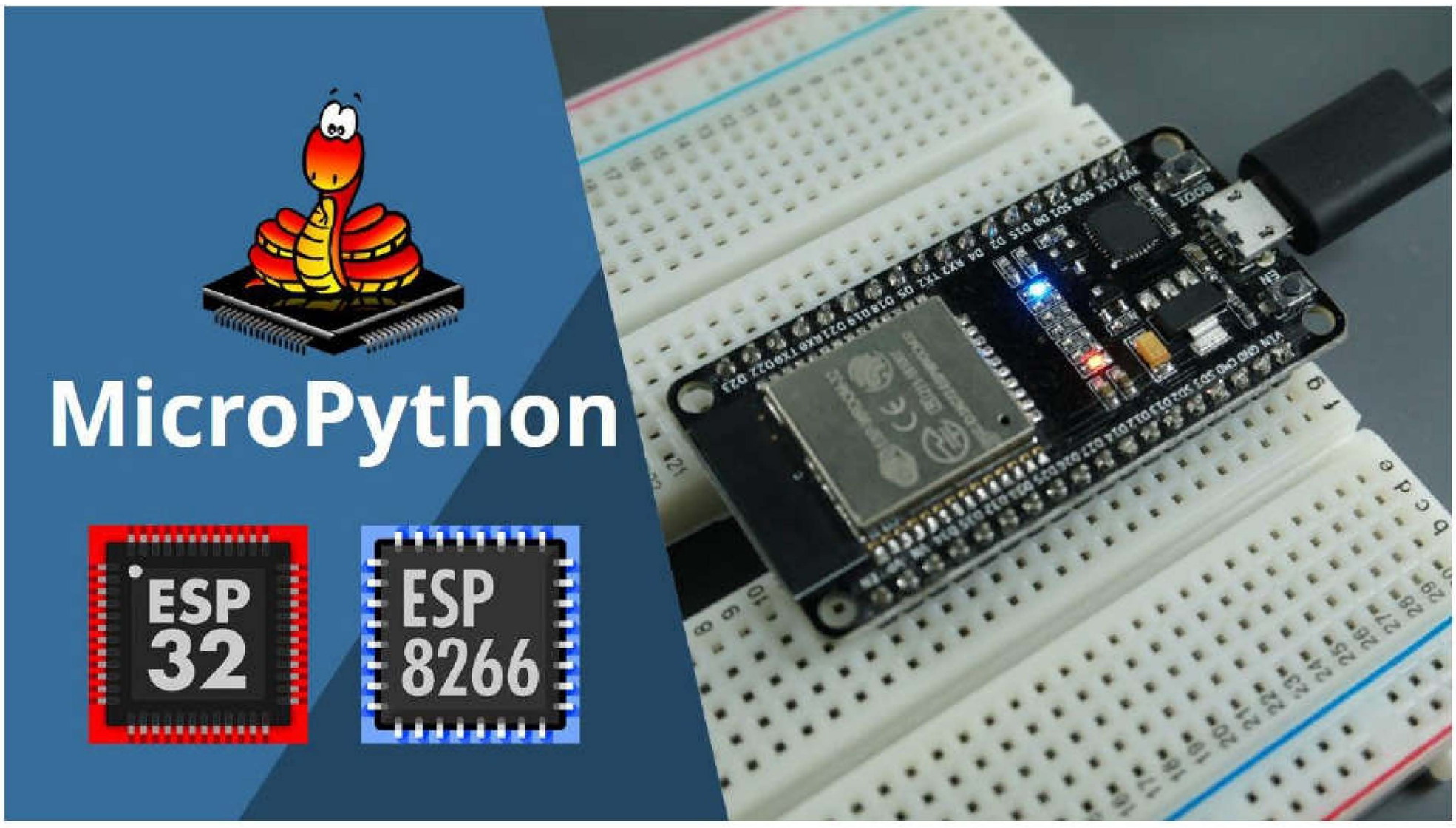
Blinking an LED



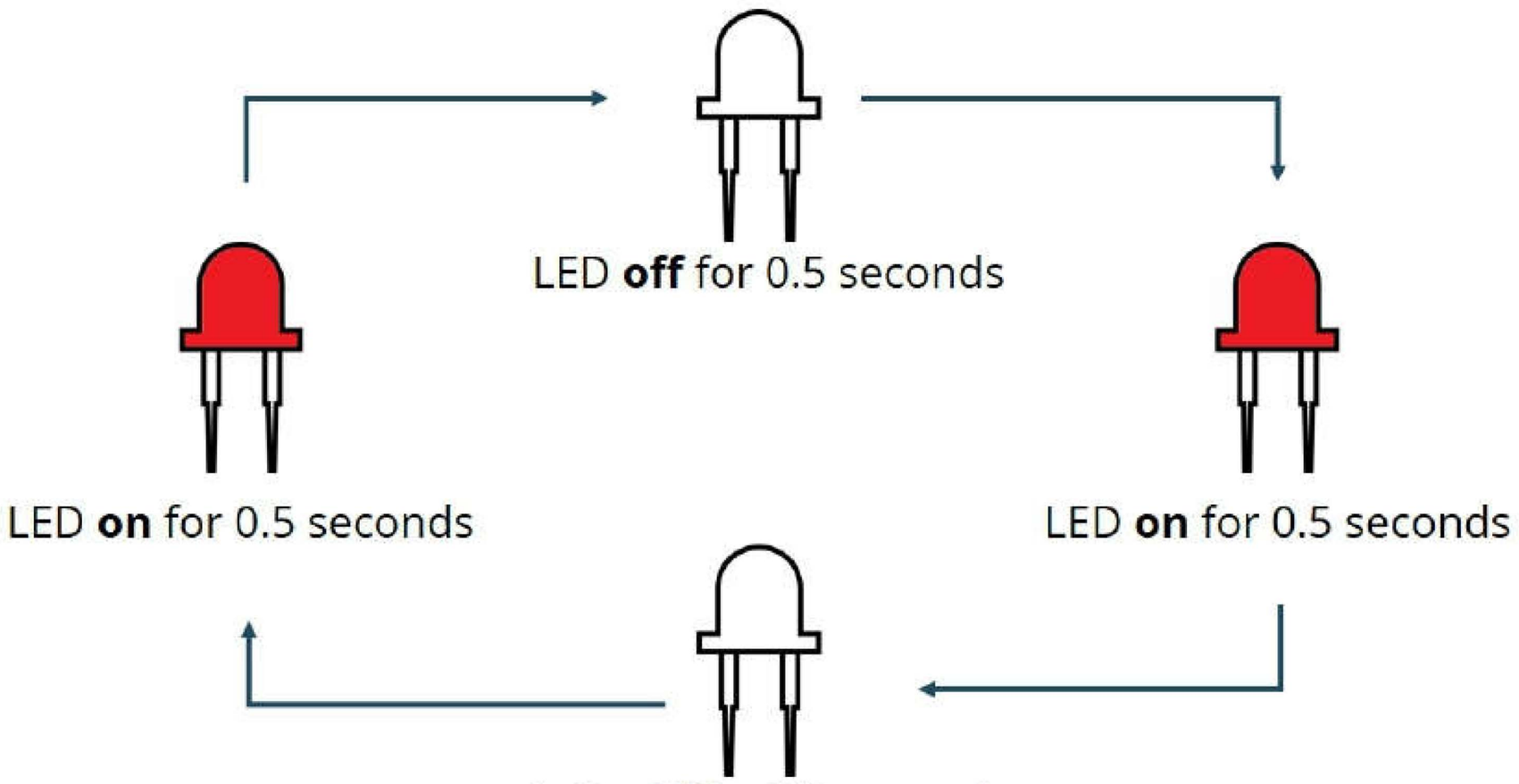
After all the theory we've been showing you, let's finally do something to interact with the physical world. The first thing we usually do when starting something new with electronics is building the blinking LED project. The blinking LED project is a great way to get users familiar with the programming language.

Project Overview

The project we'll build consists of simply blinking the ESP32/ESP8266 on-board LED. The on-board LED corresponds to GPIO 2 in both ESP32 and ESP8266. In simple terms, the blinking LED project works as follows:

- The LED turns on for 0.5 seconds—GPIO 2 set to HIGH.
- The LED turns off for 0.5 seconds—GPIO 2 set to LOW.
- The LED is on again for 0.5 seconds—GPIO 2 set to HIGH.
- The LED is off again for 0.5 seconds—GPIO 2 set to LOW.

This pattern continues until you tell the program to stop.



Script

The following script does precisely what we've described:

```
from machine import Pin
from time import sleep
led = Pin(2, Pin.OUT)
while True:
  led.value(not led.value())
  sleep(0.5)
```

SOURCE CODE

https://github.com/RuiSantosdotme/ESP-MicroPython/blob/master/code/Blink_LED/Blink_LED.py

How the Code Works

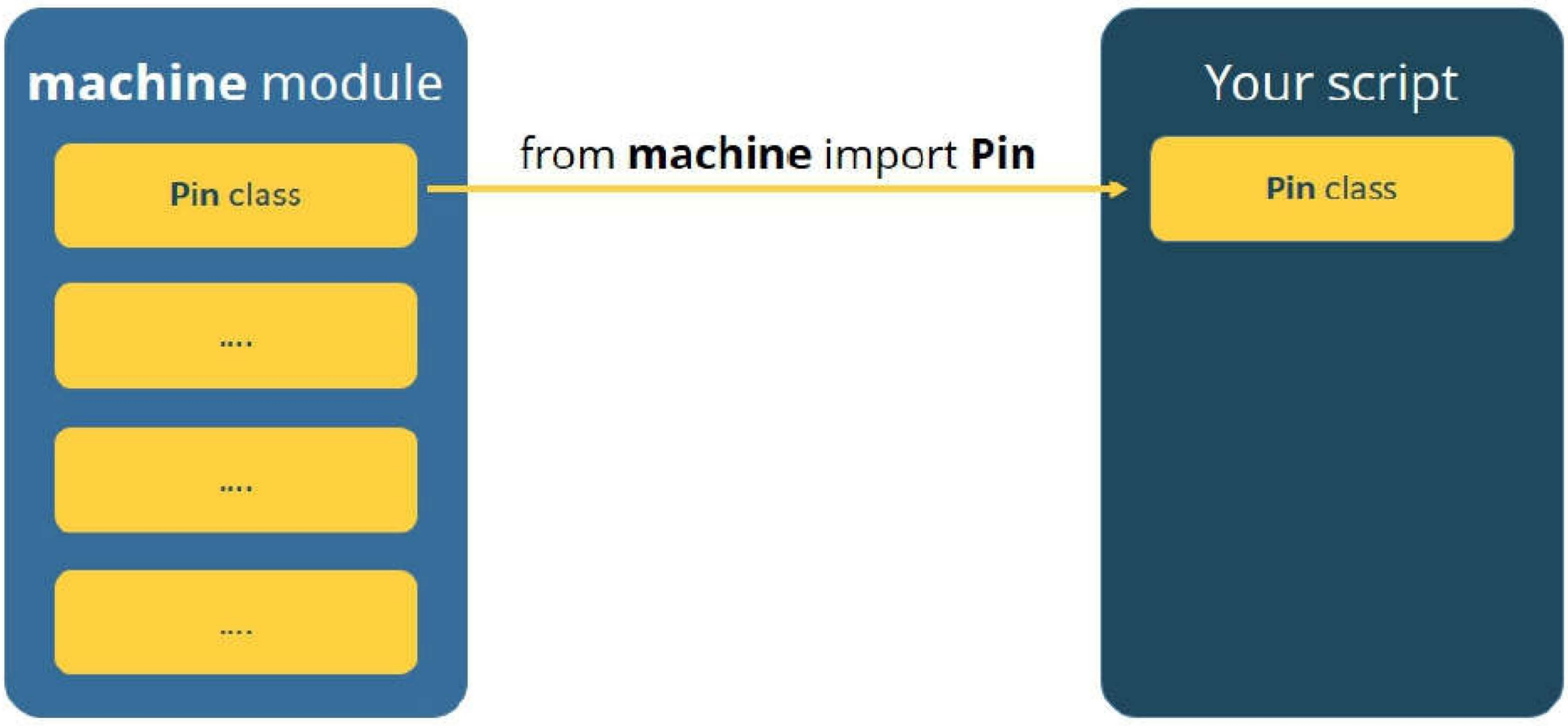
Let's take a closer look at the code to understand how it works.

The *machine* module

In all our scripts, we'll use the `machine` module. The `machine` module contains classes to interact with the physical world - this means classes to interact with the GPIOs.

The `machine` module contains many classes. We don't need all those classes in every script we create. In MicroPython, because we are writing code in constrained conditions, we should import only what we need. So, instead of importing all the

classes in the `machine` module, we just need to import the `Pin` class in this example.



The following line tells us to import only the `Pin` class from the `machine` module:

```
from machine import Pin
```

The *time* module

In most of our scripts, we'll use the `time` module. The `time` module, as the name suggests, allows us to deal with time:

- get time and date;
- use delays;
- measure how many seconds have passed since the program started;
- etc.

In this script, we need to add a delay to create the blinking effect. To add a delay we need to use the `sleep()` method. Because we only need that method from the `time` module, we import it as follows:

```
from time import sleep
```

Instantiating Pin objects

After importing all the necessary modules, we instantiate a `Pin` object called `led`.

The `Pin` object accepts these attributes in the following order:

```
(Pin number, pin mode, pull, value)
```

Here's what these arguments mean:

- **Pin number**: the GPIO number;
- **Pin mode**: a pin can be an input, output, or open-drain (we won't use open-drain mode in any of our examples):
    - **Pin.IN**
    - **Pin.OUT**
    - **Pin.OPEN_DRAIN**

Because we want GPIO 2 to act as an output, we need to set the mode to `Pin.OUT`.

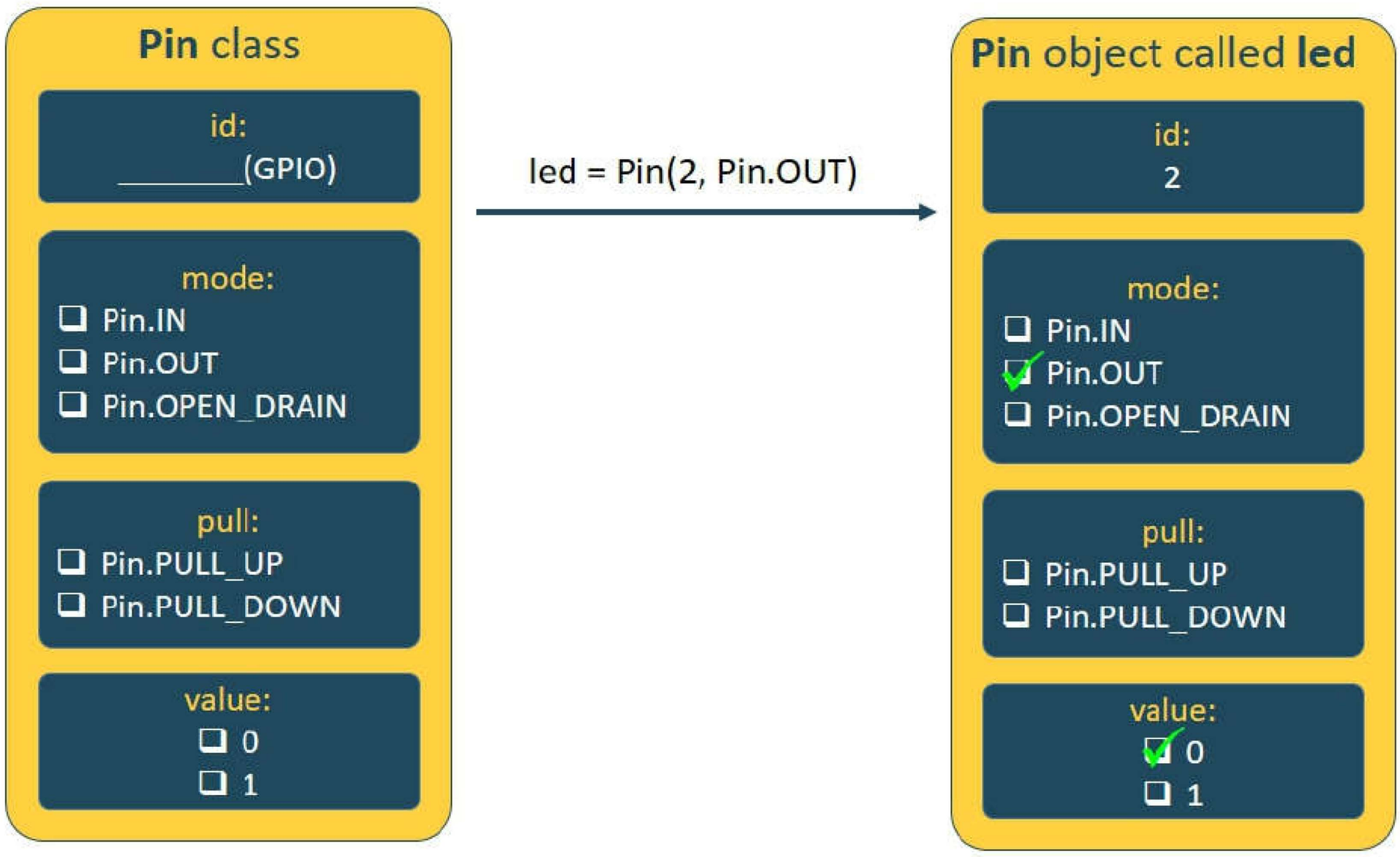
- **Pull**: this argument is used if we want to activate a pull up or pull-down internal resistor:
    - **Pin.PULL_UP** — pull-up
    - **Pin.PULL_DOWN** — pull-down

We won't set any internal resistor, so we don't need to pass this parameter.

- **Value**: can be 0 or 1, or True or False. Setting 0 or False means the GPIO is off. Setting 1 or True means the GPIO is on. If we don't pass any parameter, its state is 0 by default. So, our `led` object looks as follows:

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

The following figure may help you better understand how to create our `led` object from the `Pin` class.



Then, you create a `while` loop that is always `True`. This is a little trick to make something run forever and ever.

```
while True:
```

Note: if you are used to program with Arduino IDE, this `while` loop is similar to the `loop()` function in an Arduino sketch.

Inside the loop, we set if the GPIO is on or off. The `Pin` class provides a method called `value()` that allows us to change the state of the GPIO. For example, to change the value of the led, you use the following:

```
led.value(state)
```

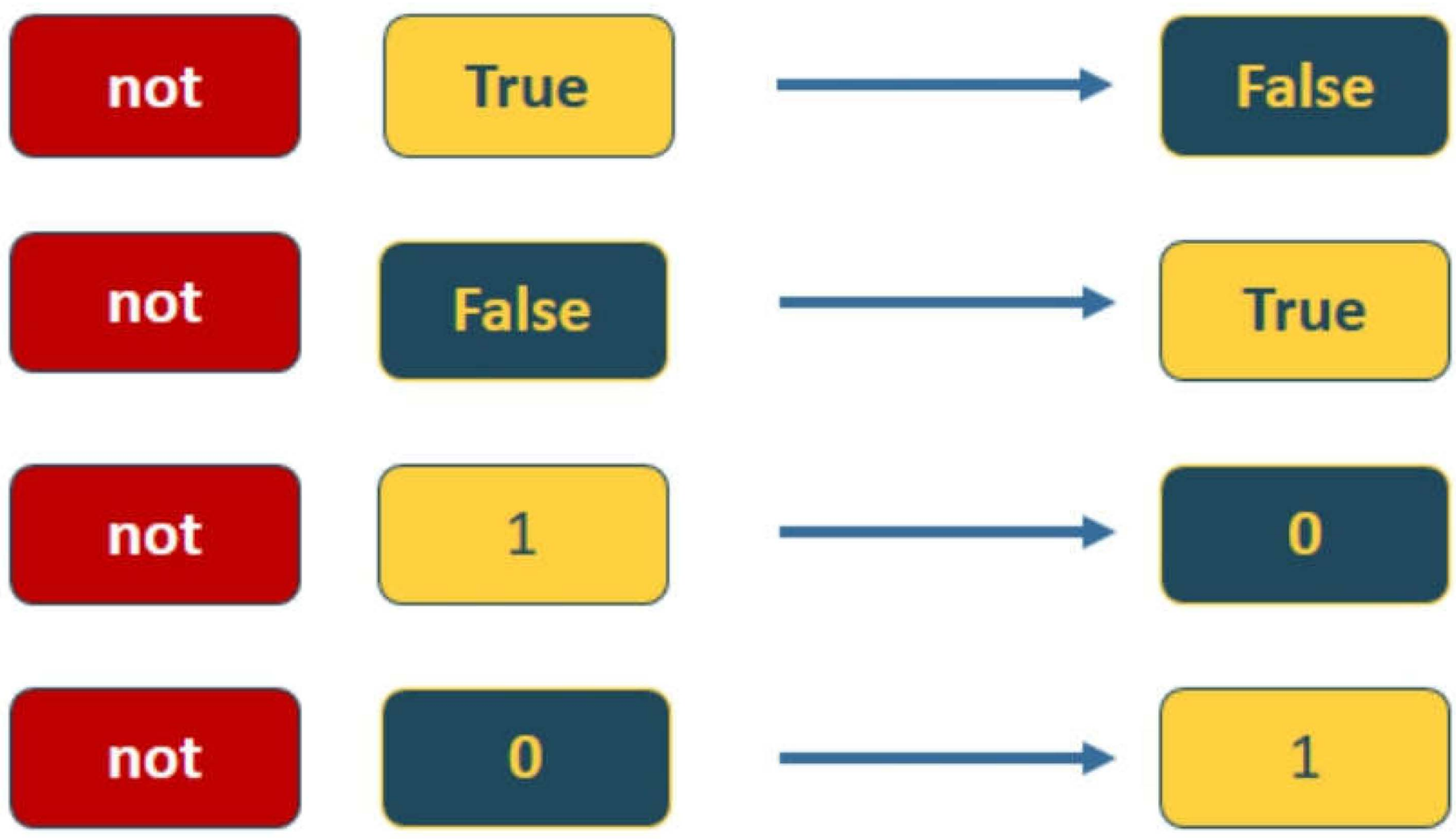
In which `state` corresponds to the state you want to set to the led: either 0 or 1 (or `True` or `False`).

As argument we pass the following expression:

```
not led.value()
```

The `led.value()` expression (without any arguments) returns the current state of the `led` object. In the first loop `led.value()` is 0. By applying the ***not*** logical operator, we get 1.

Not is a logical operator that reverses the state of the expression that follows. To better understand this concept, look at the following figure.



In the first loop, the `led.value(not led.value())` expression is the same as having `led.value(1)`. This sets the value of the `led` to 1 and turns GPIO 2 on. So, the on-board LED lights up.

After lighting up the LED, we wait for half a second (0.5 seconds).

```
sleep(0.5)
```

The `sleep()` function waits for the number of seconds you pass as argument. The code doesn't do anything else while you're in the `sleep()` function.

After waiting half a second, the loop restarts. This time, the `led.value()` is 1. By applying the `not` logical operator, it becomes 0. This is the same of writing `led.value(0)`, so the on-board LED turns off.

Then, we wait for half a second, and the loop repeats again and again, producing the blinking effect.

The following table may help you better understand what is going on in the code.

	led.value()	not led.value()	led.value(not led.value())
Loop number 1	0	1	led.value(1) → GPIO ON
Loop number 2	1	0	led.value(0) → GPIO OFF
Loop number 3	0	1	led.value(1) → GPIO ON
...	...	...	...

Upload the Code

Open uPyCraft IDE. Connect your ESP board to your computer using an USB cable. Make sure you select the right COM port in **Tools** ▸ **Serial**, and the right board in **Tools** ▸ **Board**. Then, upload the code to your board as we've shown in you in previous Units.

Demonstration

The on-board ESP32 LED should be blinking every half a second.