AsyncIO & GPIO

Asynchronous Programming & Jetson Nano GPIO

【110上】嵌入式系統技術實驗課程

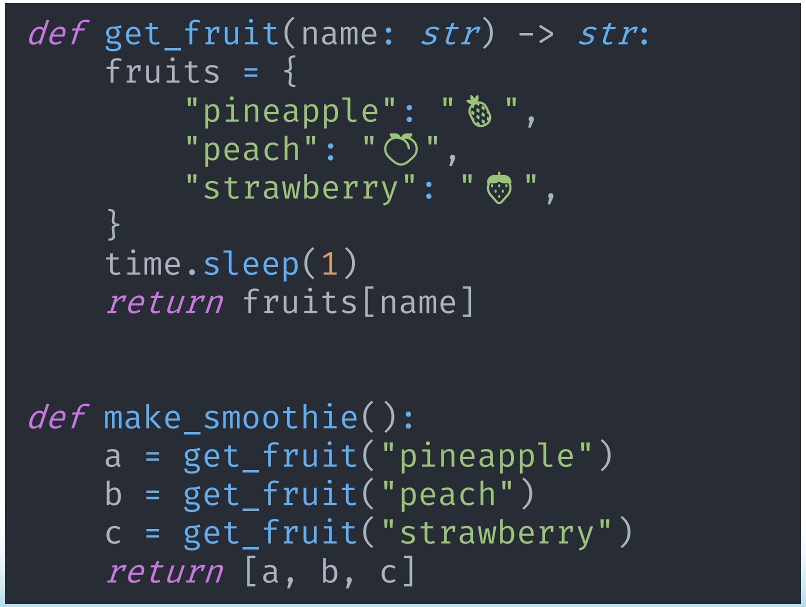
TA: 陳翰群 hanz1211.ee09@nycu.edu.tw

Asynchronous IO

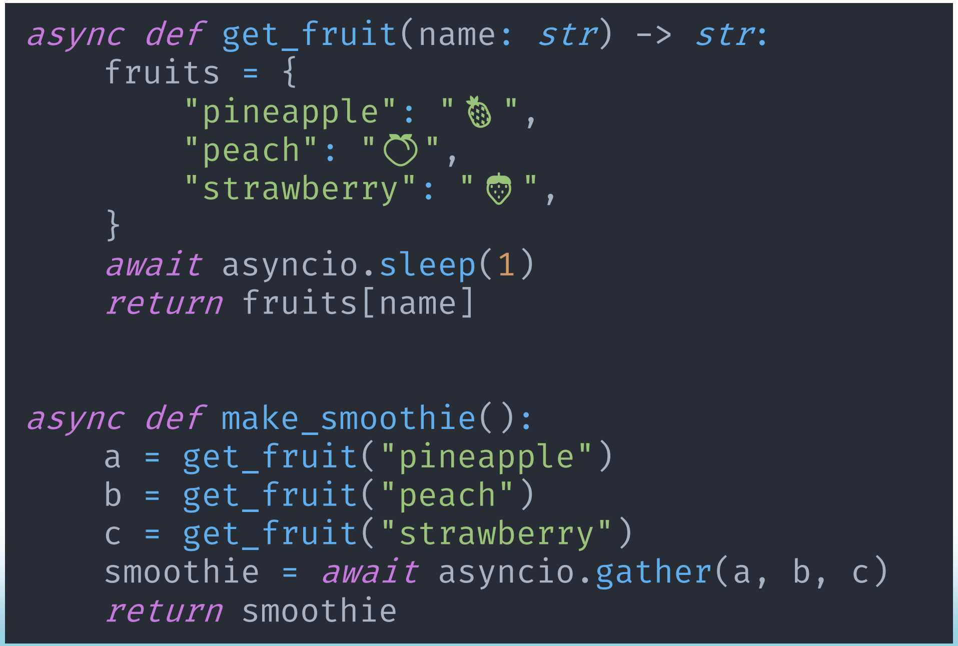
- Python 3.5 and later support async programming
  - Nonblocking concurrency
    - Prevent CPU busy waiting
    - Easier to maintain than multi-thread program
    - Mostly use in modern program to handle network communication
    - Some common terms: callback, event loop, promise, future
  - This is not parallelism
    - If you need parallel programming, use threading or multiprocessing
    - It's much harder to design and debug a parallel program
  - Use async when you have chance, use threading when you can't avoid
- More resource: https://realpython.com/async-io-python/

Python asyncio Package

- smoothie_sync.py & smoothie_async.py
- Modern Python support async programming
  - 2 new keywords: async / await
  - In synchronous version, it takes 3 seconds to finish
  - In asynchronous version, only 1 second



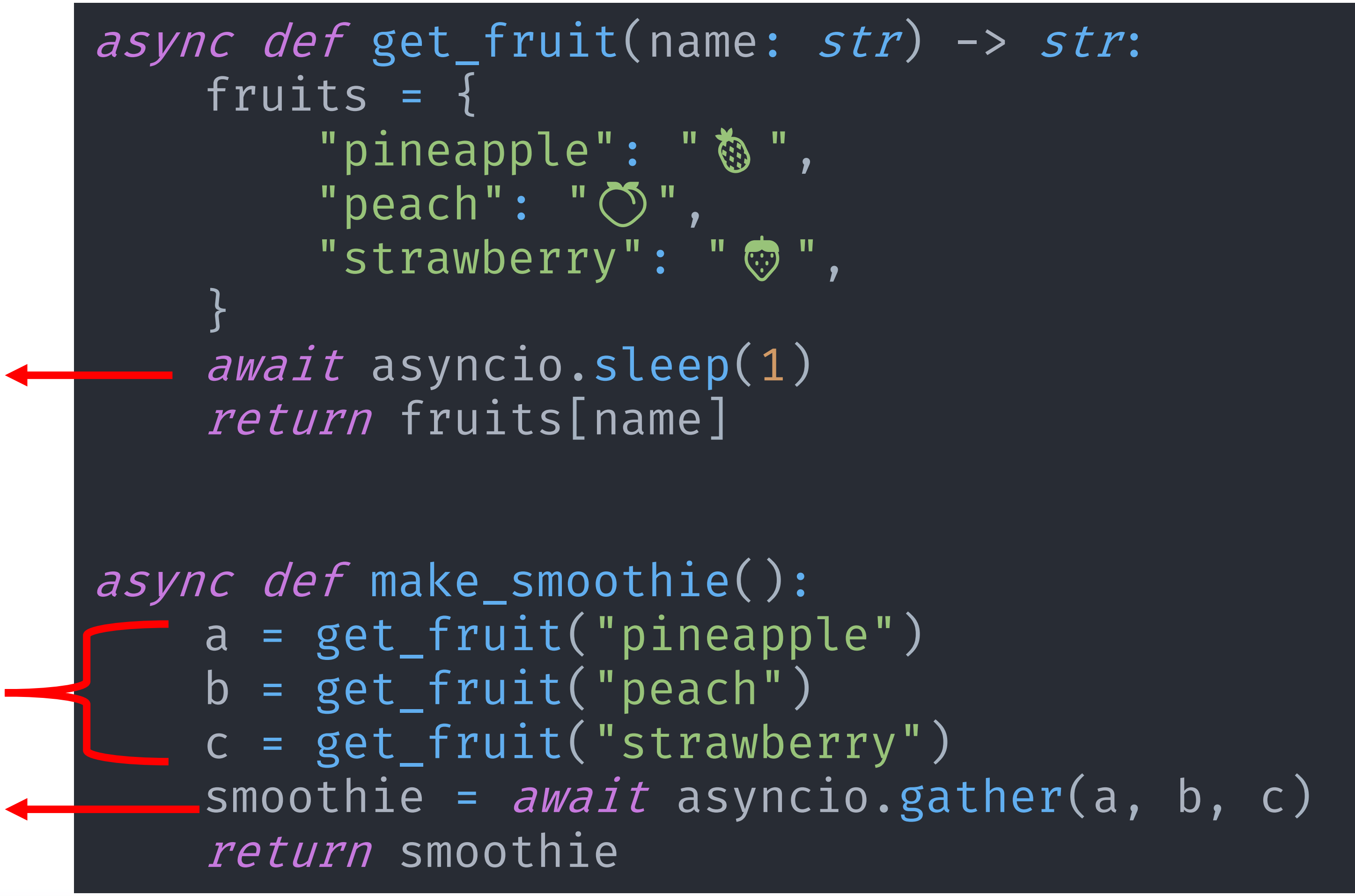
```python
def get_fruit(name: str) -> str:
    fruits = {
        "pineapple": "🍍",
        "peach": "🍑",
        "strawberry": "🍓",
    }
    time.sleep(1)
    return fruits[name]


def make_smoothie():
    a = get_fruit("pineapple")
    b = get_fruit("peach")
    c = get_fruit("strawberry")
    return [a, b, c]
```



```python
async def get_fruit(name: str) -> str:
    fruits = {
        "pineapple": "🍍",
        "peach": "🍑",
        "strawberry": "🍓",
    }
    await asyncio.sleep(1)
    return fruits[name]


async def make_smoothie():
    a = get_fruit("pineapple")
    b = get_fruit("peach")
    c = get_fruit("strawberry")
    smoothie = await asyncio.gather(a, b, c)
    return smoothie
```

Async Design Pattern

- smoothie_async.py
- async will return immediately
- Program only stop when await the value



```python
async def get_fruit(name: str) -> str:
    fruits = {
        "pineapple": "🍍",
        "peach": "🍑",
        "strawberry": "🍓",
    }
    await asyncio.sleep(1)
    return fruits[name]


async def make_smoothie():
    a = get_fruit("pineapple")
    b = get_fruit("peach")
    c = get_fruit("strawberry")
    smoothie = await asyncio.gather(a, b, c)
    return smoothie
```

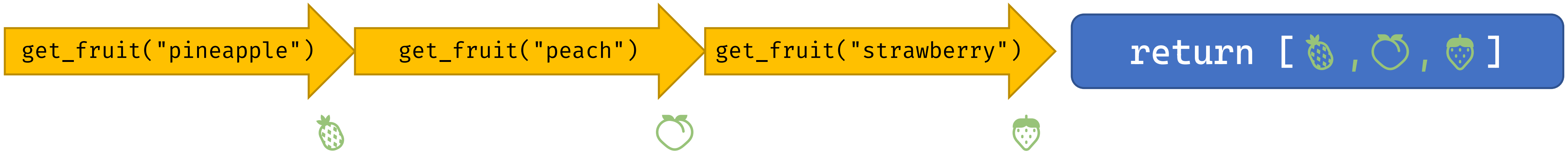
In real world app, this can be the network delay

Instant launch & return

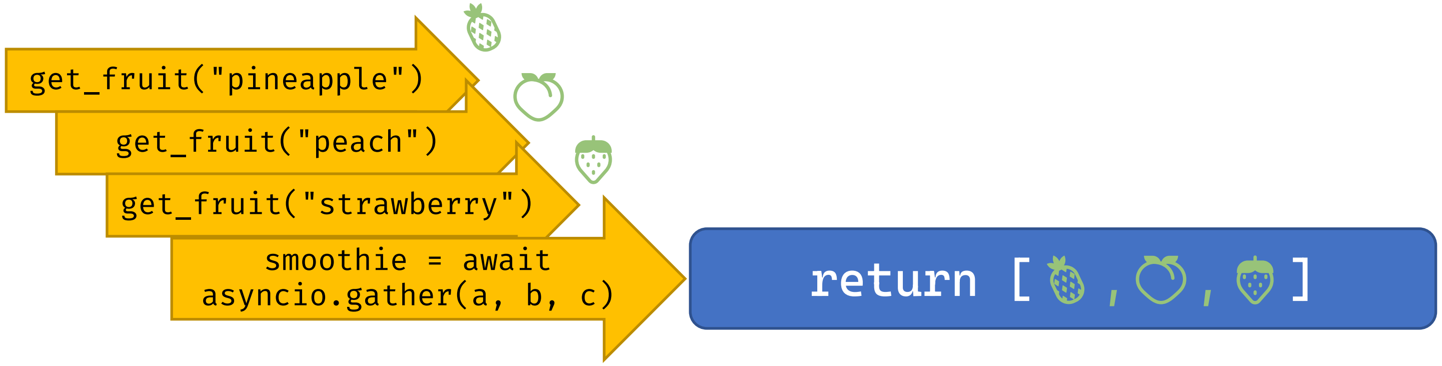
Await until all values ready

Async Timeline

- smoothie_async.py



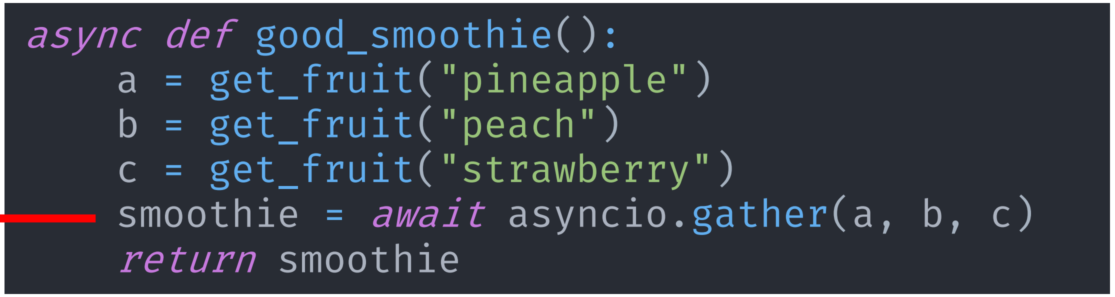
get_fruit is async
Tell CPU to do other stuff
don’t wait for this function to return



Wait for all get_fruit functions to finish

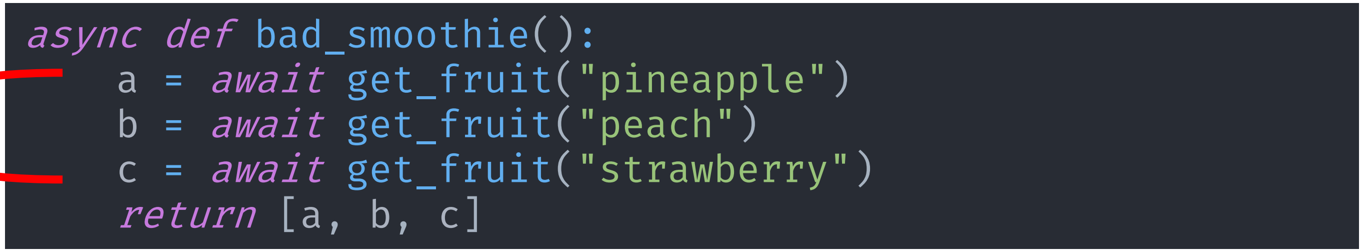
Avoid Multiple Await

- smoothie_async.py
- If you have multiple object to await, use gather to await them at the same time



```python
async def good_smoothie():
    a = get_fruit("pineapple")
    b = get_fruit("peach")
    c = get_fruit("strawberry")
    smoothie = await asyncio.gather(a, b, c)
    return smoothie
```

Wait for 3 lines at the same time

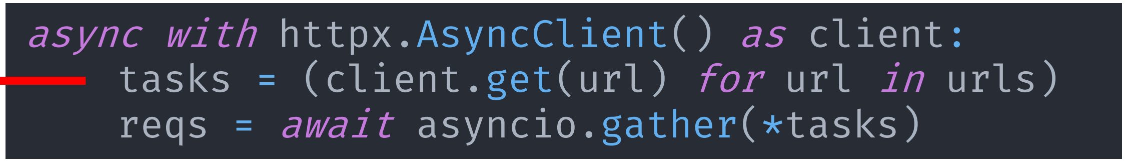


```python
async def bad_smoothie():
    a = await get_fruit("pineapple")
    b = await get_fruit("peach")
    c = await get_fruit("strawberry")
    return [a, b, c]
```

Wait 1 sec per line

Async Communication Example

- async_requests.py
  - This program is a simple web crawler that count the number of http & https URLs in top 15 popular websites
- Default requests library is synchronous, we need httpx library to create async http client



```python
async with httpx.AsyncClient() as client:
    tasks = (client.get(url) for url in urls)
    reqs = await asyncio.gather(*tasks)
```

Create 15 GET requests at the same time

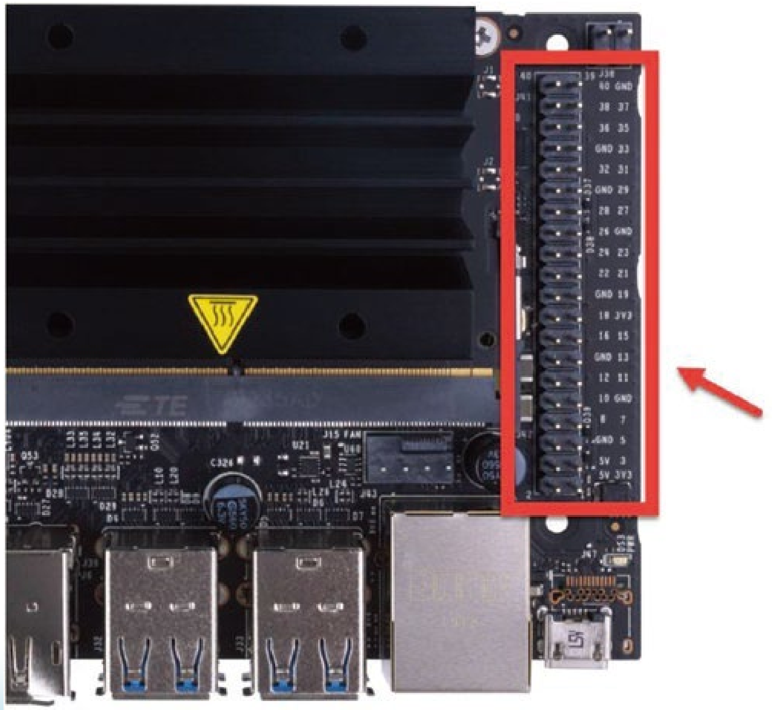
Unpacking argument lists with * operator

- Output:

```
…
Sync requests finished in 14.93 seconds.
-------------------------------
…
Async requests finished in 2.61 seconds.
```

Jetson Nano GPIO

- A general-purpose input/output (GPIO) is an uncommitted digital signal pin on a board.
- On NVIDIA Jetson Nano, you can find GPIO pins on the J41 header.



Sysfs GPIO	Name	Pin	Pin	Name	Sysfs GPIO
	3.3 VDC *Power*	1	2	**5.0 VDC** *Power*	
	I2C_2_SDA *I2C Bus 1*	3	4	**5.0 VDC** *Power*	
	I2C_2_SCL *I2C Bus 1*	5	6	**GND**	
gpio216	**AUDIO_MCLK**	7	8	**UART_2_TX** */dev/ttyTHS1*	
	GND	9	10	**UART_2_RX** */dev/ttyTHS1*	
gpio50	**UART_2_RTS**	11	12	**I2S_4_SCLK**	gpio79
gpio14	**SPI_2_SCK**	13	14	**GND**	
gpio194	**LCD_TE**	15	16	**SPI_2_CS1**	gpio232
	3.3 VDC *Power*	17	18	**SPI_2_CS0**	gpio15
gpio16	**SPI_1_MOSI**	19	20	**GND**	
gpio17	**SPI_1_MISO**	21	22	**SPI_2_MISO**	gpio13
gpio18	**SPI_1_SCK**	23	24	**SPI_1_CS0**	gpio19
	GND	25	26	**SPI_1_CS1**	gpio20
	I2C_1_SDA *I2C Bus 0*	27	28	**I2C_1_SCL** *I2C Bus 0*	
gpio149	**CAM_AF_EN**	29	30	**GND**	
gpio200	**GPIO_PZ0**	31	32	**LCD_BL_PWM**	gpio168
gpio38	**GPIO_PE6**	33	34	**GND**	
gpio76	**I2S_4_LRCK**	35	36	**UART_2_CTS**	gpio51
gpio12	**SPI_2_MOSI**	37	38	**I2S_4_SDIN**	gpio77
	GND	39	40	**I2S_4_SDOUT**	gpio78

Setting Up GPIO

- The NVIDIA Jetson Nano GPIO can use from 1.8V to 3.3V. By default, all GPIO pins use 3.3V. Make sure you don’t use a pin input voltage of more than 3.3V. Otherwise, your board will be broken.
- To access the NVIDIA Jetson Nano GPIO, use the Jetson.GPIO library from Jetson. This library is modified from the RPI GPIO (Raspberry Pi) library.

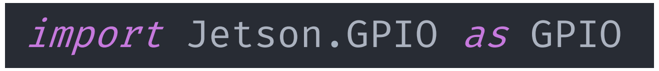
```
sudo apt-get install git-all
git clone https://github.com/NVIDIA/jetson-gpio.git
cd jetson-gpio
sudo python3 setup.py install
```

- Setting User Permissions

```
sudo groupadd -f -r gpio
sudo usermod -a -G gpio <your_user_name>
sudo cp lib/python/Jetson/GPIO/99-gpio.rules /etc/udev/rules.d/
sudo udevadm control --reload-rules && sudo udevadm trigger
```

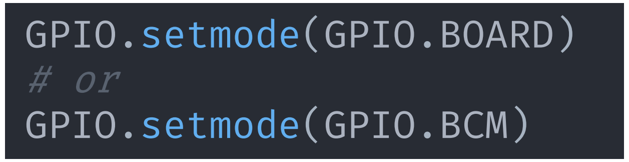
GPIO Mode

- https://github.com/NVIDIA/jetson-gpio#complete-library-api



```python
import Jetson.GPIO as GPIO
```

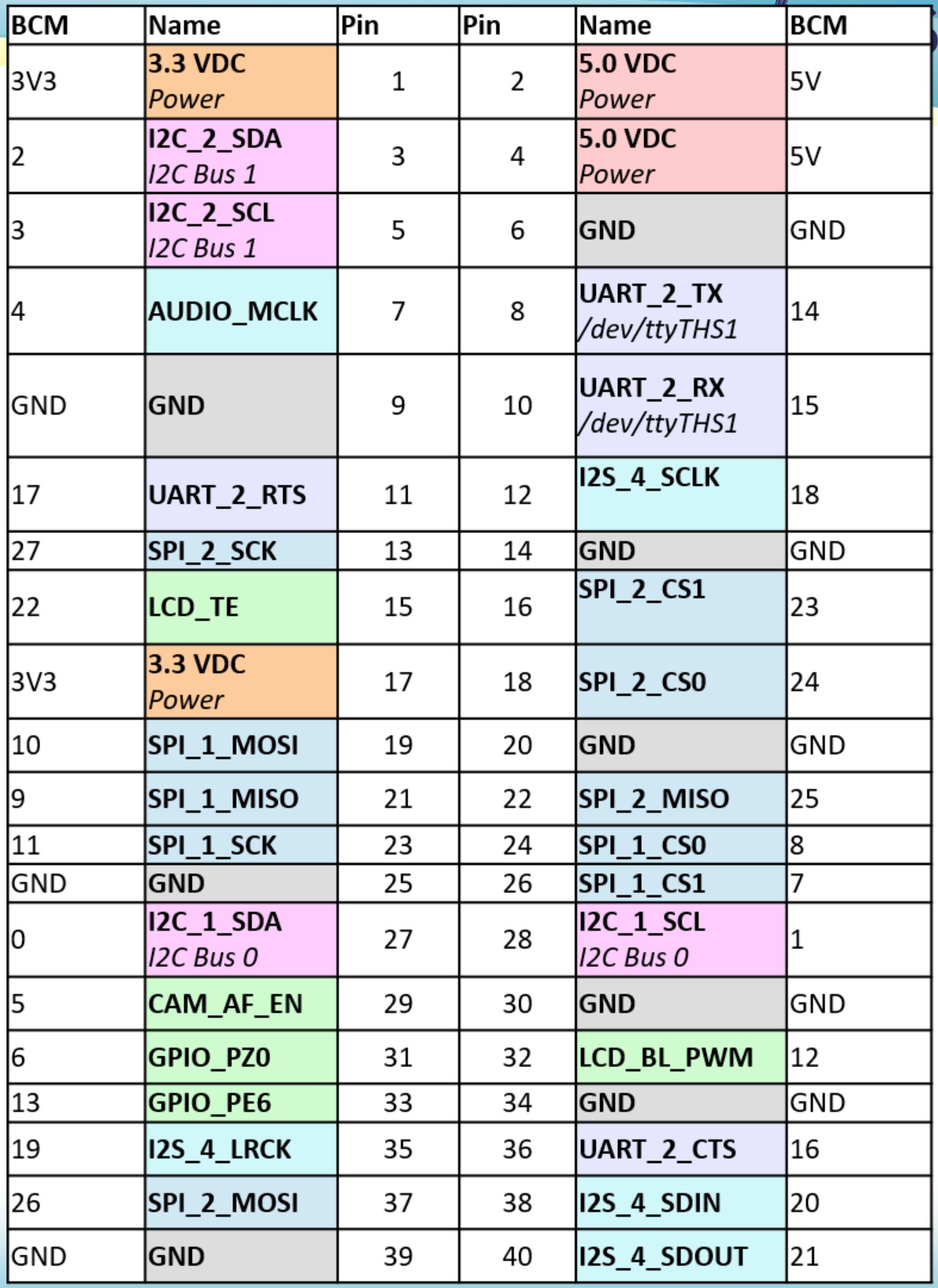
- There are 4 modes of GPIO: Board, BCM, CVM, Tegra SoC
- We normally use Board and BCM
- To specify which mode you are using, use the following function call:



```python
GPIO.setmode(GPIO.BOARD)
# or
GPIO.setmode(GPIO.BCM)
```

GPIO Pin

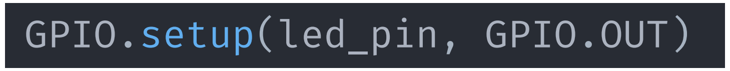
- In Board mode, use Pin column as ID
- In BCM mode, use BCM column as ID



BCM	Name	Pin	Pin	Name	BCM
3V3	**3.3 VDC** *Power*	1	2	**5.0 VDC** *Power*	5V
2	**I2C_2_SDA** *I2C Bus 1*	3	4	**5.0 VDC** *Power*	5V
3	**I2C_2_SCL** *I2C Bus 1*	5	6	**GND**	GND
4	**AUDIO_MCLK**	7	8	**UART_2_TX** */dev/ttyTHS1*	14
GND	**GND**	9	10	**UART_2_RX** */dev/ttyTHS1*	15
17	**UART_2_RTS**	11	12	**I2S_4_SCLK**	18
27	**SPI_2_SCK**	13	14	**GND**	GND
22	**LCD_TE**	15	16	**SPI_2_CS1**	23
3V3	**3.3 VDC** *Power*	17	18	**SPI_2_CS0**	24
10	**SPI_1_MOSI**	19	20	**GND**	GND
9	**SPI_1_MISO**	21	22	**SPI_2_MISO**	25
11	**SPI_1_SCK**	23	24	**SPI_1_CS0**	8
GND	**GND**	25	26	**SPI_1_CS1**	7
0	**I2C_1_SDA** *I2C Bus 0*	27	28	**I2C_1_SCL** *I2C Bus 0*	1
5	**CAM_AF_EN**	29	30	**GND**	GND
6	**GPIO_PZ0**	31	32	**LCD_BL_PWM**	12
13	**GPIO_PE6**	33	34	**GND**	GND
19	**I2S_4_LRCK**	35	36	**UART_2_CTS**	16
26	**SPI_2_MOSI**	37	38	**I2S_4_SDIN**	20
GND	**GND**	39	40	**I2S_4_SDOUT**	21

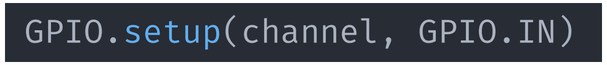
GPIO Mode

- gpio_led.py
- Connect board pin 7 and any GND to LED
- This will toggle LED every 2 seconds
- Define the GPIO pin as output



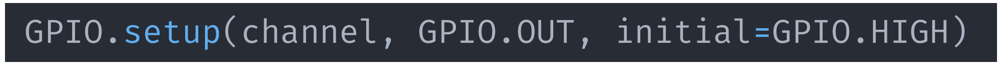
```
GPIO.setup(led_pin, GPIO.OUT)
```

  - Set as input:



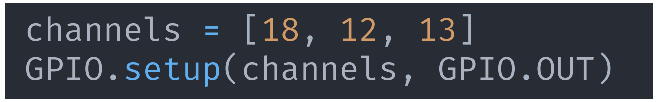
```
GPIO.setup(channel, GPIO.IN)
```

  - Set with initial value

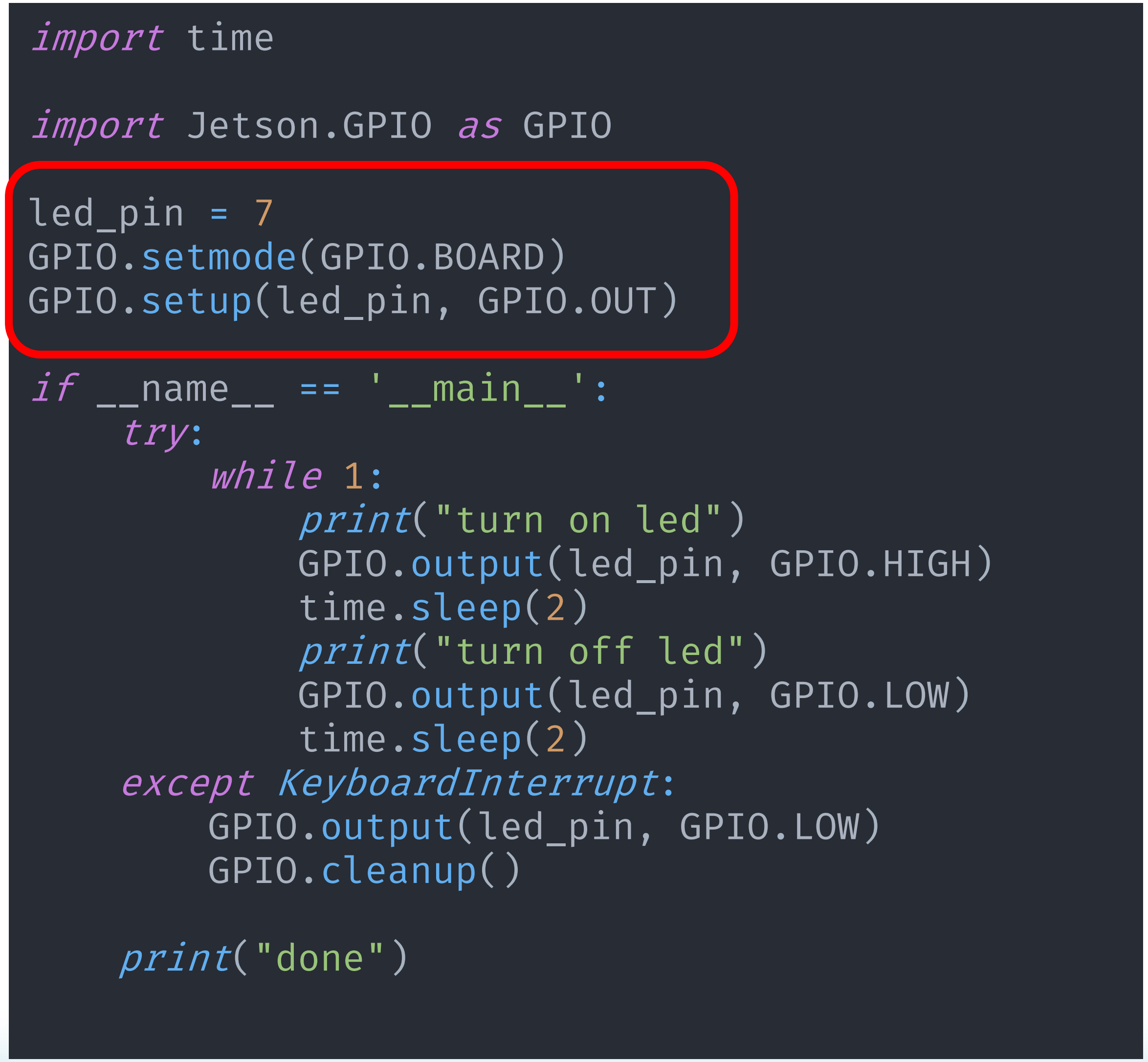


```
GPIO.setup(channel, GPIO.OUT, initial=GPIO.HIGH)
```

  - Set multiple pin



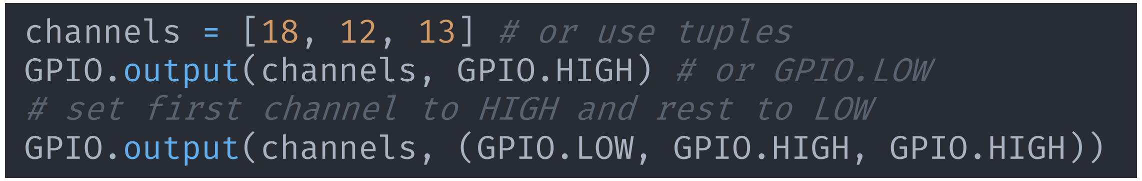
```
channels = [18, 12, 13]
GPIO.setup(channels, GPIO.OUT)
```



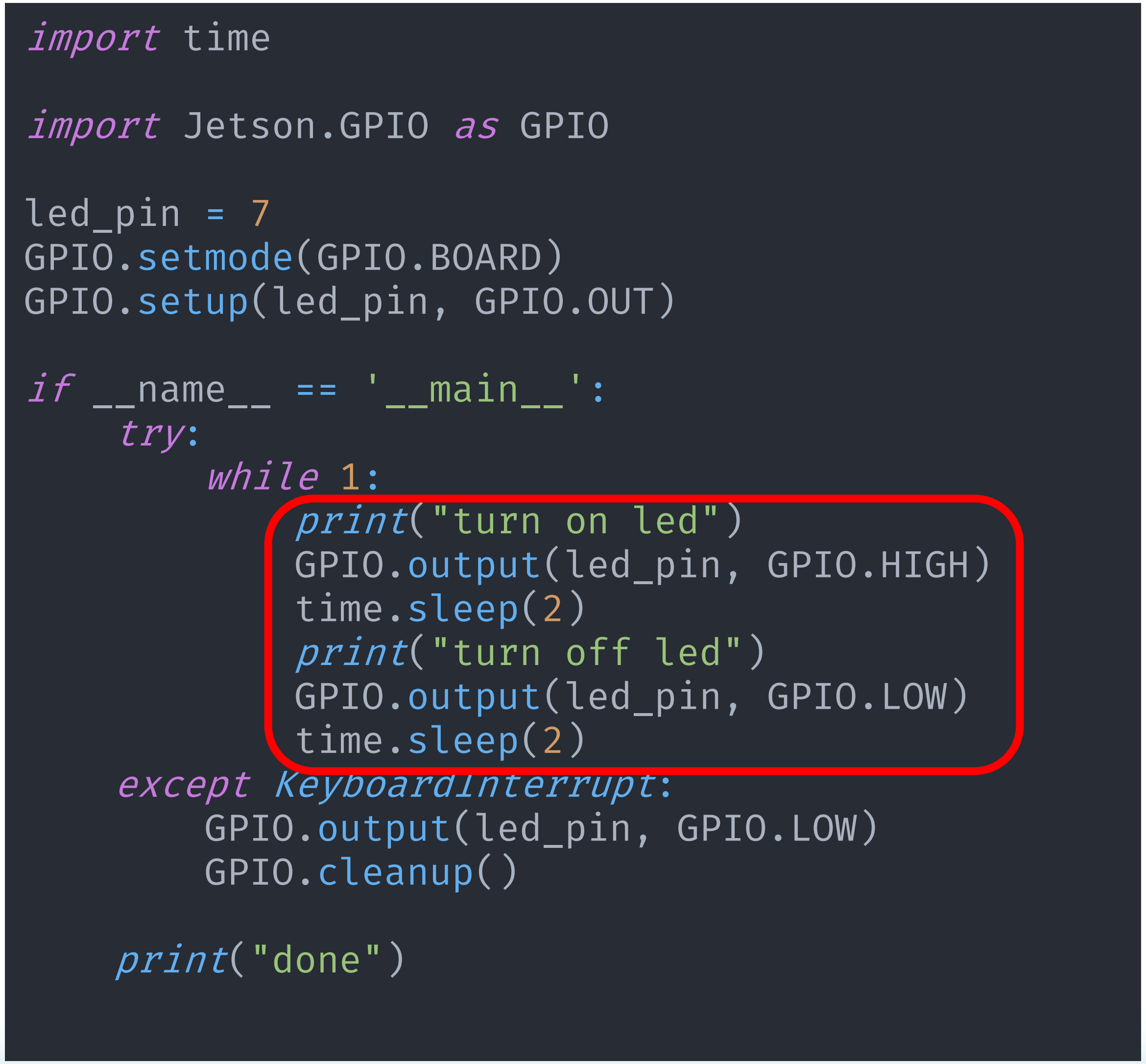
```
import time

import Jetson.GPIO as GPIO

led_pin = 7
GPIO.setmode(GPIO.BOARD)
GPIO.setup(led_pin, GPIO.OUT)

if __name__ == '__main__':
    try:
        while 1:
            print("turn on led")
            GPIO.output(led_pin, GPIO.HIGH)
            time.sleep(2)
            print("turn off led")
            GPIO.output(led_pin, GPIO.LOW)
            time.sleep(2)
    except KeyboardInterrupt:
        GPIO.output(led_pin, GPIO.LOW)
        GPIO.cleanup()

    print("done")
```

GPIO Output

- gpio_led.py
- After defining GPIO pins, you can send signals with GPIO.output
- State can be GPIO.LOW or GPIO.HIGH
  - Set multiple output at the same time



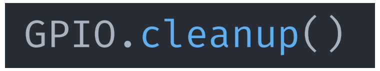
```
channels = [18, 12, 13] # or use tuples
GPIO.output(channels, GPIO.HIGH) # or GPIO.LOW
# set first channel to HIGH and rest to LOW
GPIO.output(channels, (GPIO.LOW, GPIO.HIGH, GPIO.HIGH))
```



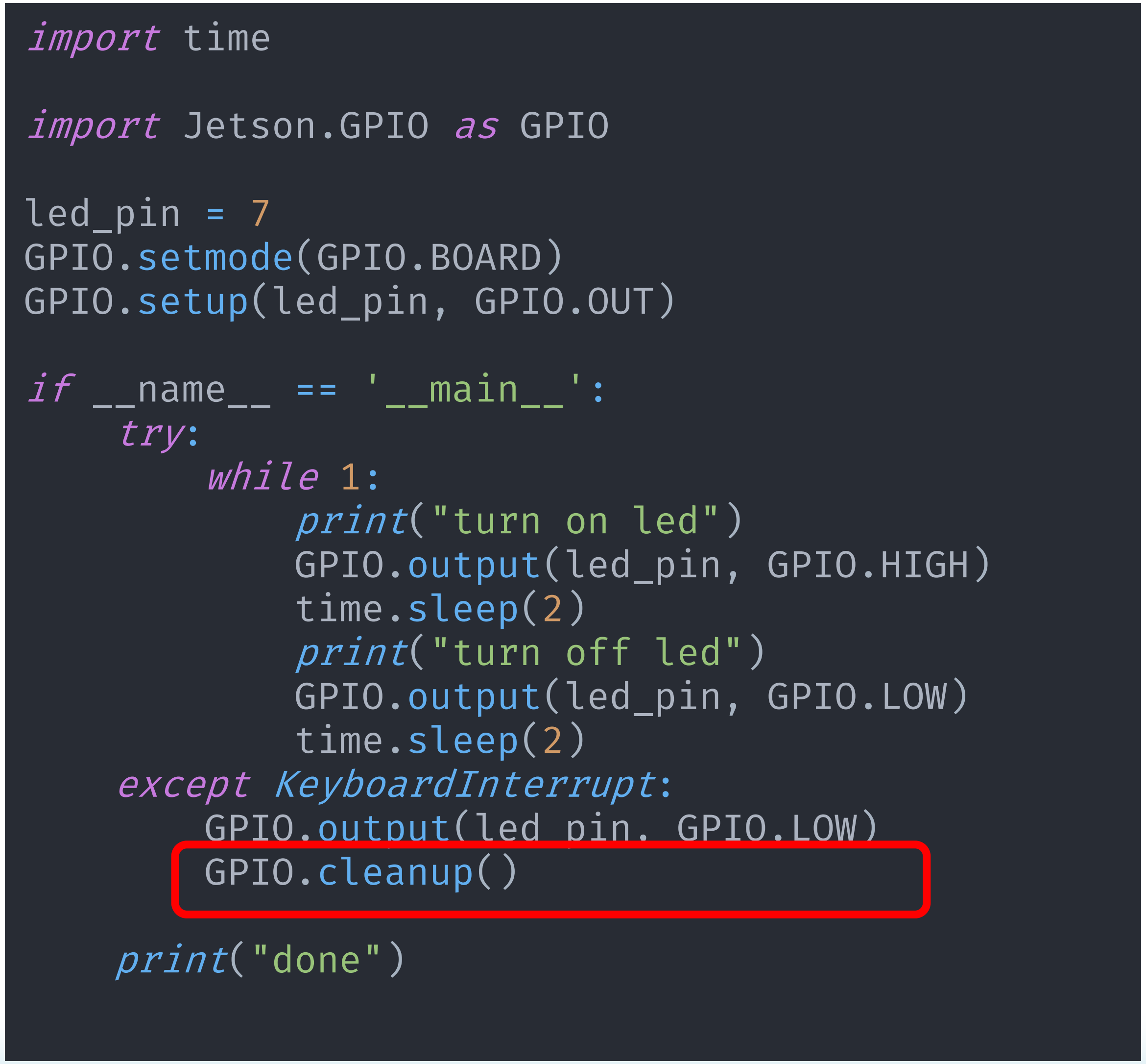
```
import time

import Jetson.GPIO as GPIO

led_pin = 7
GPIO.setmode(GPIO.BOARD)
GPIO.setup(led_pin, GPIO.OUT)

if __name__ == '__main__':
    try:
        while 1:
            print("turn on led")
            GPIO.output(led_pin, GPIO.HIGH)
            time.sleep(2)
            print("turn off led")
            GPIO.output(led_pin, GPIO.LOW)
            time.sleep(2)
    except KeyboardInterrupt:
        GPIO.output(led_pin, GPIO.LOW)
        GPIO.cleanup()

    print("done")
```

GPIO Clean Up

- gpio_led.py
- At the end of the program, it is good to clean up the channels so that all pins are set in their default state.
- To clean up all channels used, call:



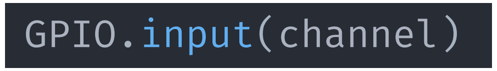
```
GPIO.cleanup()
```



```
import time

import Jetson.GPIO as GPIO

led_pin = 7
GPIO.setmode(GPIO.BOARD)
GPIO.setup(led_pin, GPIO.OUT)

if __name__ == '__main__':
    try:
        while 1:
            print("turn on led")
            GPIO.output(led_pin, GPIO.HIGH)
            time.sleep(2)
            print("turn off led")
            GPIO.output(led_pin, GPIO.LOW)
            time.sleep(2)
    except KeyboardInterrupt:
        GPIO.output(led_pin, GPIO.LOW)
        GPIO.cleanup()

    print("done")
```

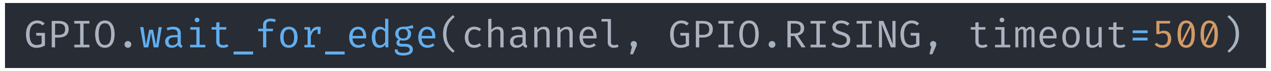
GPIO Input & Interrupts

- To read the value from input pin:



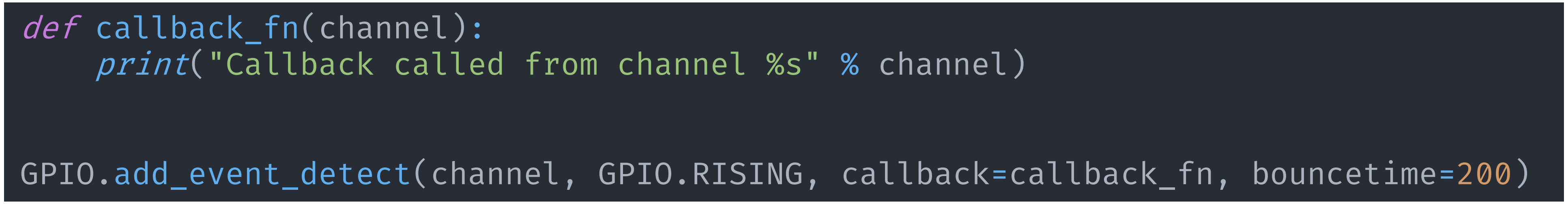
```
GPIO.input(channel)
```

  - This return GPIO.LOW or GPIO.HIGH
- Blocking wait function
  - This will make program wait for channel pin voltage rise for at most 500ms, if no timeout specify, will wait forever.



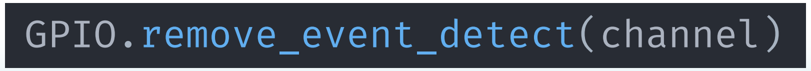
```
GPIO.wait_for_edge(channel, GPIO.RISING, timeout=500)
```

- Callback when edge is detected
  - Add an event listener, when detect voltage rising, will trigger the callback function, also debounce is optional to prevent multiple triggers



```
def callback_fn(channel):
    print("Callback called from channel %s" % channel)


GPIO.add_event_detect(channel, GPIO.RISING, callback=callback_fn, bouncetime=200)
```

  - If event listener is no longer required:



```
GPIO.remove_event_detect(channel)
```

GPIO PWM & More

- https://github.com/NVIDIA/jetson-gpio
  - Check official Git sample codes: samples/simple_pwm.py

- https://www.rs-online.com/designspark/jetson-nano-40-pin-gpio-cn