

Raspberry Pi GPIO

Author: Nick Lee

Nano shortcuts

Ctrl-O to save
Ctrl-X to exit

Ctrl-W to find
Alt-W to find again
Ctrl-_ to go to a line number
Ctrl-\ to find and replace

Ctrl-^ to set mark, then move cursor to select
Alt-^ to copy
Ctrl-K to cut
Ctrl-U to paste (uncut)

Alt-} to block indent
Alt-{ to block unindent

Edit /etc/nanorc to change settings.

Recommended settings:

```
set tabsize 4
set tabstospaces
set smooth
```

```
$ python3
```

```
>>> 1 + 3
```

```
>>> 9 - 7
```

```
>>> 6 * 3
```

```
>>> 3 / 2
```

```
>>> 3 // 2
```

```
>>> 7 % 3
```

```
>>> 4 + 2 * 3
```

```
>>> (4 + 2) * 3
```

```
>>> 2 ** 3
```

```
>>> type(3)
```

```
>>> type(3.0)
```

```
>>> type('3')
```

```
>>> 3.0 == 3
```

```
>>> '3' == 3
```

```
>>> # comment
```

```
>>> x = 1
```

```
>>> y = 2
```

```
>>> x + y
```

```
>>> x * y
```

```
>>> import time
```

```
>>> time.time()
```

```
>>> x = 25
```

```
>>> 'I am {} years old'.format(x)
```

```
>>> street = 'Nathan Road'
```

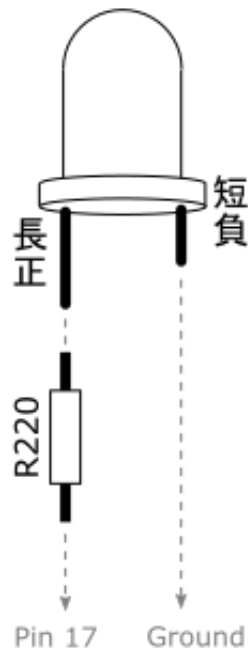
```
>>> area = 'Mongkok'
```

```
>>> 'I live on {} {}, {}'.format(x, street, area)
```

```
>>> humidity, temperature = 50.00001, 22.34567
```

```
>>> 'Today is {}% {:.2f} degree'.format(humidity, temperature)
```

LED



Raspberry Pi interacts with the outside world via **General-purpose Input Output (GPIO)** pins. We are going to control the pins using Python.

```
$ python3
```

```
>>> from RPi import GPIO
>>> GPIO.setmode(GPIO.BCM)      # use Broadcom numbering
>>> GPIO.setup(17, GPIO.OUT)    # setup pin 17 as output
>>> GPIO.output(17, 1)          # turn it on
>>> GPIO.output(17, 0)          # turn it off
>>> GPIO.cleanup()
```

Can you write a program that blinks the LED?

blink.py

```
import time
from RPi import GPIO

led = 17 ❶

GPIO.setmode(GPIO.BCM)
GPIO.setup(led, GPIO.OUT)

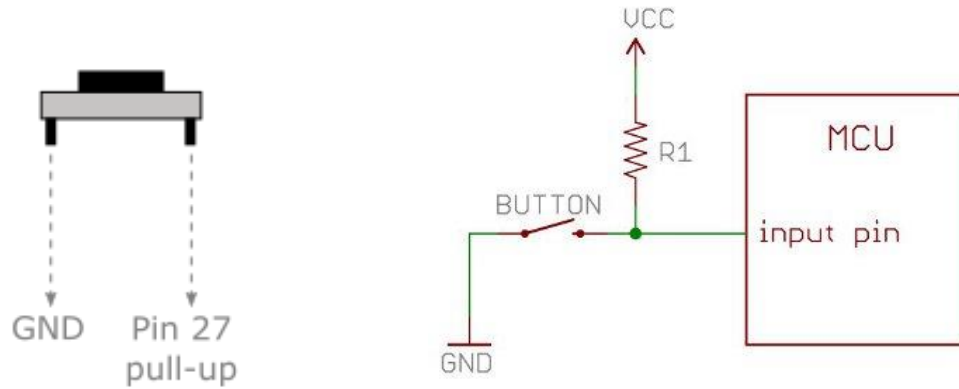
while 1:
    GPIO.output(led, 1)
    time.sleep(0.5) ❷
    GPIO.output(led, 0)
    time.sleep(0.5) ❸
```

❶ Define the LED pin number. If you change pin, this is the only place you need to change.

❷ Keep the light on for 0.5 second.

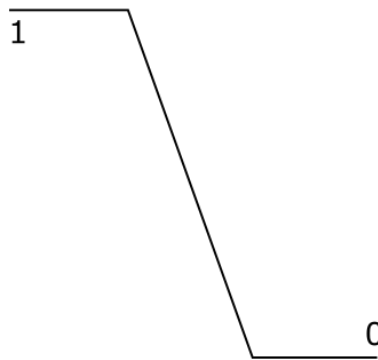
❸ Keep the light off for 0.5 second.

Button



Because it is pulled up, pin 27 is normally **HIGH (1)**.
On pressed, it becomes **LOW (0)**.

A button press is a change of voltage from HIGH to LOW.



We call that a **falling edge**.

waitforedge.py

```
from RPi import GPIO

button = 27

GPIO.setmode(GPIO.BCM)
GPIO.setup(button, GPIO.IN, pull_up_down=GPIO.PUD_UP)

while True:
    GPIO.wait_for_edge(button,
                        GPIO.FALLING, ❶
                        bouncetime=300) ❷
    print('Edge falling')
```

❶ Blocks until a falling edge occurs

❷ Keep reading for an explanation of **bouncetime**

While `wait_for_edge()` is waiting, we can't do anything else. Another method is to set up a **callback**, and let the GPIO library invoke the callback when it detects a falling edge.

detect.py

```
import time
from RPi import GPIO

button = 27

GPIO.setmode(GPIO.BCM)
GPIO.setup(button, GPIO.IN, pull_up_down=GPIO.PUD_UP)

def button_pressed(channel): ❶
    print('Pressed on channel', channel)

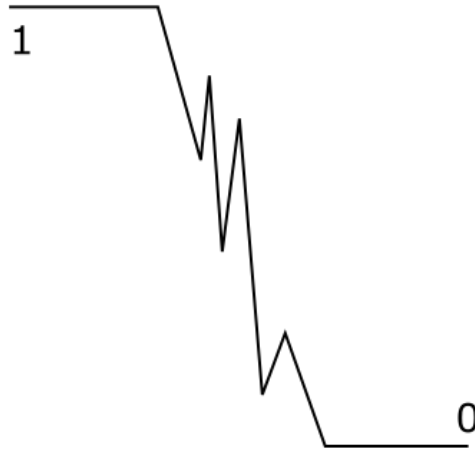
GPIO.add_event_detect(button,
                       GPIO.FALLING,
                       callback=button_pressed, ❷
                       bouncetime=300) ❸

while True: ❹
    time.sleep(1)
```

- ❶ Define a callback function
- ❷ Tell GPIO to invoke the callback function on a falling edge
- ❸ See the following page for an explanation of **bouncetime**
- ❹ Keep the program running. Otherwise, we would lose the edge detection.

What is bouncetime?

A push button is imperfect. The voltage change from HIGH to LOW is not a smooth edge as depicted before. In reality, it looks more like this:



The button **bounces**.

bouncetime=300 tells GPIO to ignore the bounces for 300 milliseconds.

If we don't tell it to ignore bounces, it would report multiple falling edges for a single button press.

Try removing **bouncetime=300** from the above code. What do you expect?

Experiment with various **bouncetime**. Which values do you prefer?

Quick reaction game

Now that you know how to handle LED and push button, let's check how quick your reaction is. Make a reaction timer as follows:

1. Turn on LED
2. User presses button to "start the clock". LED is turned off.
3. After a random number of seconds (say, 2-10 seconds), LED is turned back on!
4. User has to press button as soon as he can. Print out his reaction time.

Which strategy would you use to detect button press?

reaction.py

```
import random, time
from RPi import GPIO

GPIO.setmode(GPIO.BCM)

led = 17
button = 27

GPIO.setup(led, GPIO.OUT)
GPIO.setup(button, GPIO.IN, pull_up_down=GPIO.PUD_UP)

GPIO.output(led, 1)

GPIO.wait_for_edge(button, ❶
                    GPIO.FALLING,
                    bouncetime=300)
GPIO.output(led, 0)

delay = random.uniform(2, 10)
time.sleep(delay) ❷

light_on = time.time() ❸
GPIO.output(led, 1)

GPIO.wait_for_edge(button, ❹
                    GPIO.FALLING,
                    bouncetime=300)
pressed = time.time() ❺
GPIO.output(led, 0)

reaction = pressed - light_on
print('Reaction time: {:.3f} sec'.format(reaction))





















GPIO.cleanup()
```

- ① Wait for user to "start the clock"
- ② Delay for a random number of seconds (2-10)
- ③ Remember the time when LED is turned on
- ④ Wait for user to press
- ⑤ Remember the time when user presses the button

如果你想用 GPIO 針做更多嘢，我鼓勵你試吓 GPIO Zero

`https://gpiozero.readthedocs.io`

Raspberry Pi B+ J8 Header

| <i>Pin#</i> | <i>NAME</i> | | <i>NAME</i> | <i>Pin#</i> |
|-------------|-----------------------|---|-----------------------|-------------|
| 01 | 3.3v DC Power |  | DC Power 5v | 02 |
| 03 | GPIO02 (SDA1 , I2C) |  | DC Power 5v | 04 |
| 05 | GPIO03 (SCL1 , I2C) |  | Ground | 06 |
| 07 | GPIO04 (GPIO_GCLK) |  | (TXD0) GPIO14 | 08 |
| 09 | Ground |  | (RXD0) GPIO15 | 10 |
| 11 | GPIO17 (GPIO_GEN0) |  | (GPIO_GEN1) GPIO18 | 12 |
| 13 | GPIO27 (GPIO_GEN2) |  | Ground | 14 |
| 15 | GPIO22 (GPIO_GEN3) |  | (GPIO_GEN4) GPIO23 | 16 |
| 17 | 3.3v DC Power |  | (GPIO_GEN5) GPIO24 | 18 |
| 19 | GPIO10 (SPI_MOSI) |  | Ground | 20 |
| 21 | GPIO09 (SPI_MISO) |  | (GPIO_GEN6) GPIO25 | 22 |
| 23 | GPIO11 (SPI_CLK) |  | (SPI_CE0_N) GPIO08 | 24 |
| 25 | Ground |  | (SPI_CE1_N) GPIO07 | 26 |
| 27 | ID_SD (I2C ID EEPROM) |  | (I2C ID EEPROM) ID_SC | 28 |
| 29 | GPIO05 |  | Ground | 30 |
| 31 | GPIO06 |  | GPIO12 | 32 |
| 33 | GPIO13 |  | Ground | 34 |
| 35 | GPIO19 |  | GPIO16 | 36 |
| 37 | GPIO26 |  | GPIO20 | 38 |
| 39 | Ground |  | GPIO21 | 40 |

Rev. 1.1
16/07/2014

<http://www.element14.com>