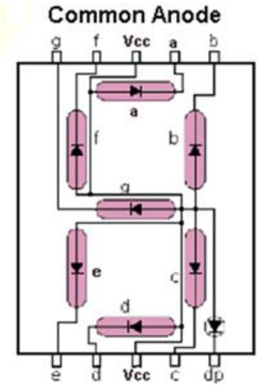
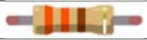
Seven-segment LED



Licenced under CC BY-NC-ND

Things used in the project

Hardware

	Raspberry Pi 3 Model B+
	7-segment LED (common anode)
	Jumper wires x 8
	Breadboard
	350Ω resistor x 1 (orange, orange, brown, gold)

Software

- Python 3
- RPi.GPIO (included in standard Raspbian installation)

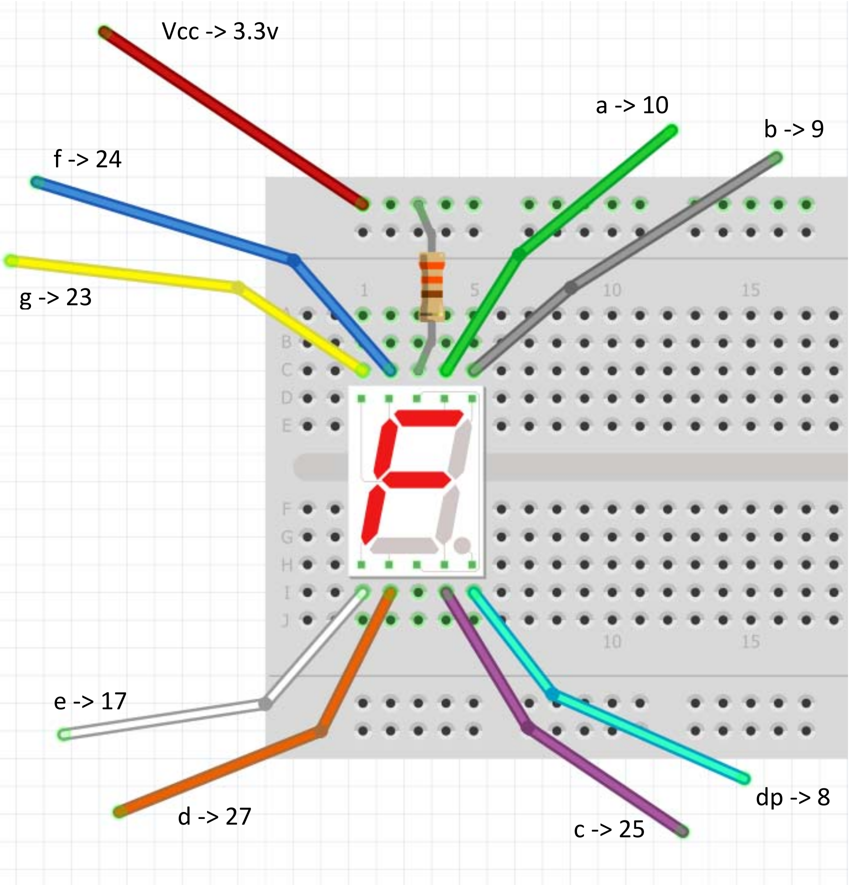


Diagram created using Fritzing (http://fritzing.org)

Wiring (GPIO)

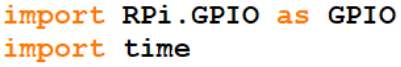
- 3.3v -> Vcc (with resistor)
- GPIO 10 (Pin #19) -> a
- GPIO 9 (Pin #21) -> b
- GPIO 25 (Pin #22) -> c
- GPIO 27 (Pin #13) -> d
- GPIO 17 (Pin #11) -> e
- GPIO 24 (Pin #18) -> f
- GPIO 23 (Pin #16) -> g
- GPIO 8 (Pin #24) -> dp

Created by Rebeca D'Cruz rebecca@d-cruz.me.uk

Seven-segment LED

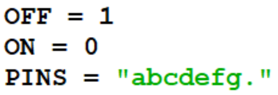
NB As always, when using GPIO pins, you must run the Python program using elevated privileges (**su**peruser **do**) at a command prompt, e.g. **`sudo python3 test_SSeg.py`**

Import GPIO and time



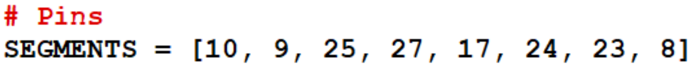
```
import RPi.GPIO as GPIO
import time
```

Set up constants, note that OFF is 1 and ON is 0



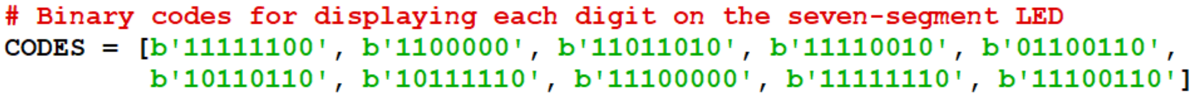
```
OFF = 1
ON = 0
PINS = "abcdefg."
```

Create constants for the PINs, change these if you have wired differently.



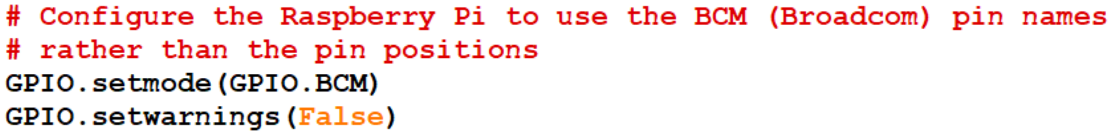
```
# Pins
SEGMENTS = [10, 9, 25, 27, 17, 24, 23, 8]
```

Each digit has a binary code (<u>not</u> a binary value). Create a list for these. Index 0 contains the binary code pattern for digit 0 etc.



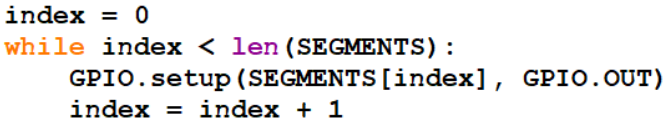
```
# Binary codes for displaying each digit on the seven-segment LED
CODES = [b'11111100', b'1100000', b'11011010', b'11110010', b'01100110',
         b'10110110', b'10111110', b'11100000', b'11111110', b'11100110']
```

Configure to use BROADCOM (BCM) numbering. These are the GPIO numbers on the outside of a pinout diagram. Turn off warnings.



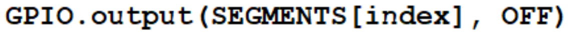
```
# Configure the Raspberry Pi to use the BCM (Broadcom) pin names
# rather than the pin positions
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
```

Set up all the pins to be for output.



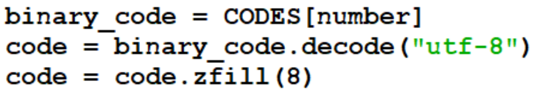
```
index = 0
while index < len(SEGMENTS):
    GPIO.setup(SEGMENTS[index], GPIO.OUT)
    index = index + 1
```

Then turn off all the LEDs. You could add this to the loop above.



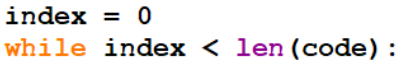
```
GPIO.output(SEGMENTS[index], OFF)
```

Given a 0-9 integer, ***number***, get the binary code from the list, decode it and pad it to 8 digits.



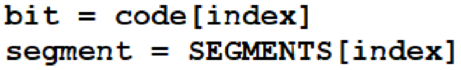
```
binary_code = CODES[number]
code = binary_code.decode("utf-8")
code = code.zfill(8)
```

Then iterate over the binary code



```
index = 0
while index < len(code):
```

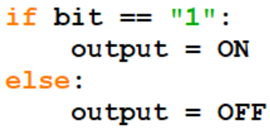
In the loop, extract and bit and get the pin for that segment.



```
bit = code[index]
segment = SEGMENTS[index]
```

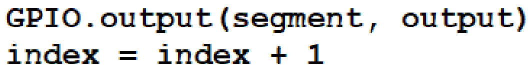
Flip the output.

Seven-segment LED



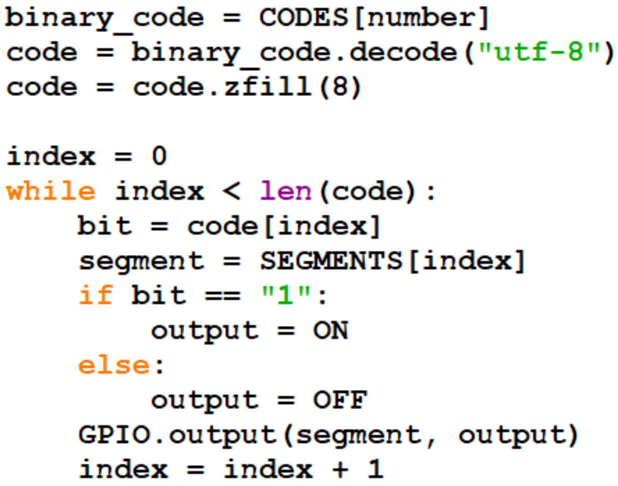
```
if bit == "1":
    output = ON
else:
    output = OFF
```

Turn that LED on or off and increment the loop counter.



```
GPIO.output(segment, output)
index = index + 1
```

The whole loop should look like this:



```
binary_code = CODES[number]
code = binary_code.decode("utf-8")
code = code.zfill(8)

index = 0
while index < len(code):
    bit = code[index]
    segment = SEGMENTS[index]
    if bit == "1":
        output = ON
    else:
        output = OFF
    GPIO.output(segment, output)
    index = index + 1
```

You could use this code for digital clocks, numeric output, countdowns etc. You could also work out the binary codes for letters.