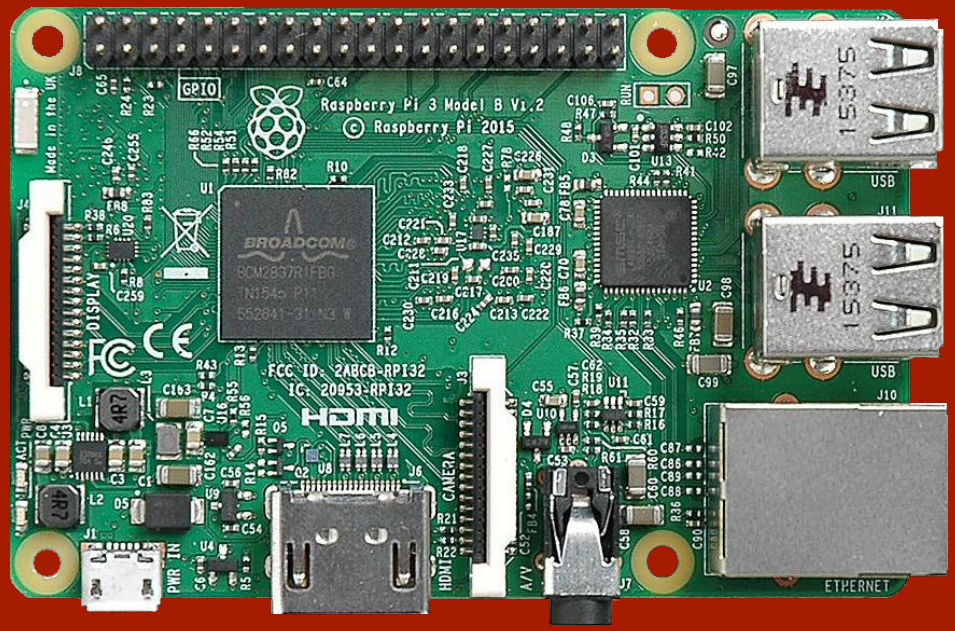
IoT and Practical Applications in Data Science

Raspberry Pi Hardware 1



TINKERCADEMY

An Infocomm Club Appointed Vendor

Raspberry Pi as an IoT Device?

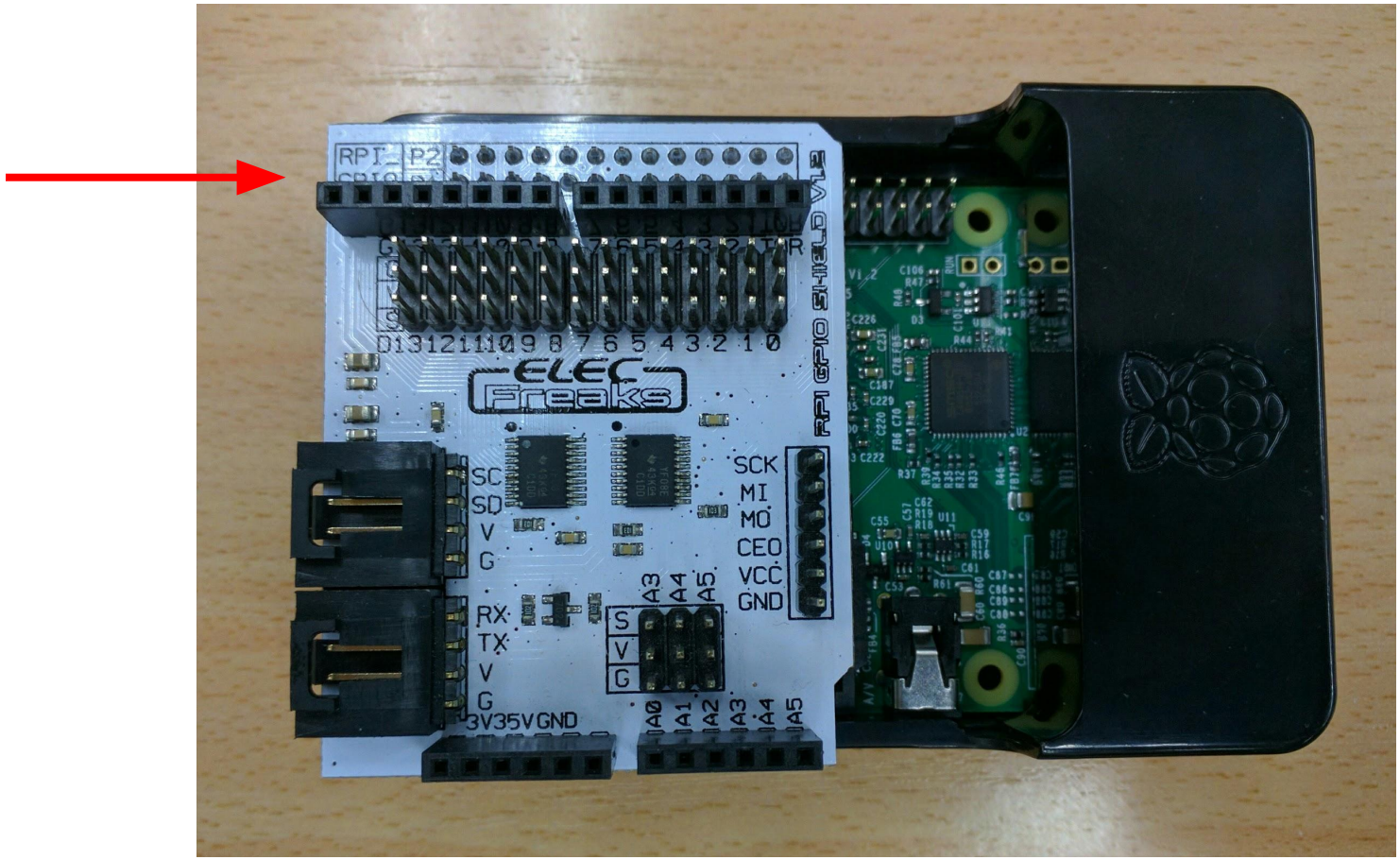
It can be!

If we connect sensors to it, it can function as an IoT device that collects data.

We also have the option of using it as a computer, but connecting a monitor and peripherals (mouse, keyboard, etc.)

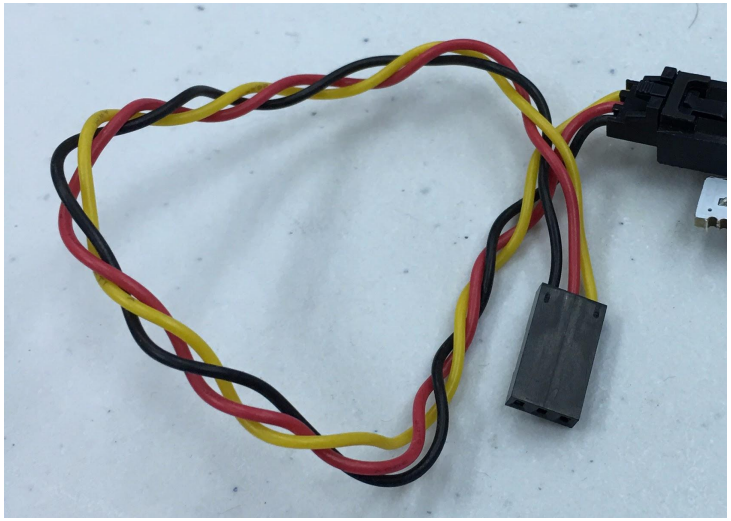
How to Connect the GPIO Shield

The GPIO Shield connects to the leftmost of the 40 pins:



Connecting Sensors

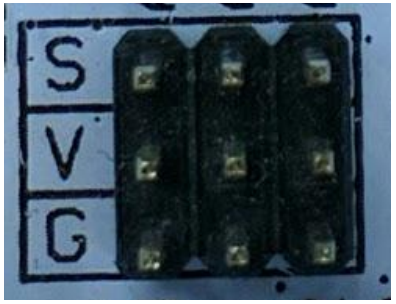
First, we need to connect sensors properly. Notice how each module has three different coloured wires? We need to connect these to the proper pins:



Yellow > S (Signal)

Red > V (Voltage)

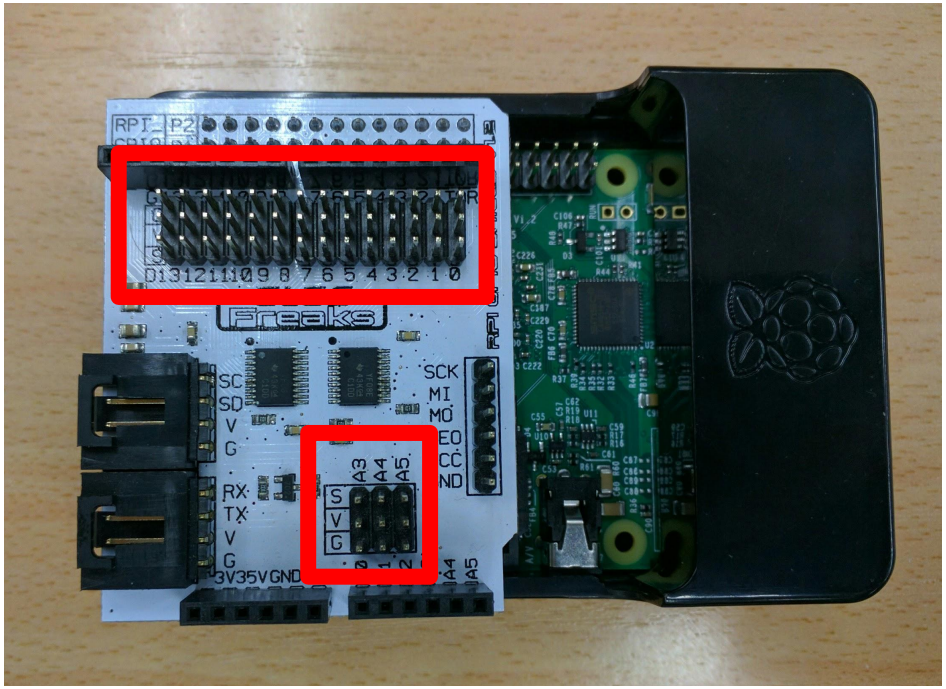
Black > G (Ground)



Analog vs Digital

Notice that we have analog and digital pins.

- Digital pins are labelled D0 to D13.
- Analog pins are labeled A3, A4, and A5.



Analog vs Digital

An analog signal has an infinite amount of possibilities.

Analog →

Digital →

23:59:59

While a digital signal has a limited number of possibilities.

Picture from https://learn.sparkfun.com/tutorials/analog-vs-digital

Pin Map

D0 = RX

D1 = TX

D2 = GPIO 17

D3 = GPIO 18

D4 = GPIO 27

D5 = GPIO 22

D6 = GPIO 23

D7 = GPIO 24

D8 = GPIO 25

D9 = GPIO 4

D10 = GPIO 8

D11 = MOSI

D12 = MISO

D13 = SCLK

A3 = GPIO 7

A4 = SDA

A5 = SCL

How to Interact with the GPIO Shield

Your SD card comes with GPIO libraries pre-installed!

We do need to import them though:

```
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BCM)
GPIO.setup(pin number, GPIO.OUT)
GPIO.output(pin number, True/False)
```

Note that we can't run our python programs using IDLE.
Instead, we need to run them via Terminal.

Running Your Program

1. Open Terminal
2. Navigate to the directory with your python program
3. Type: `python filename.py`

   where filename is your actual filename.

Try to make an LED blink

Hint: use ‘import time’
and ‘time.sleep()’

Making an LED Blink

Example code to light up one LED (plug LED into D2):

```
import RPi.GPIO as GPIO
import time
GPIO.setmode(GPIO.BCM)
GPIO.setup(17, GPIO.OUT)
while True:
    GPIO.output(17, True)
    time.sleep(1)
    GPIO.output(17, False)
    time.sleep(1)
```

Making an LED Blink

How to stop your program from running:

Ctrl + C - sends a signal to kill the program but gives the program a chance to clean itself up first before closing

Ctrl + Z - sends a signal to kill the program, without giving it the chance to cleanup first

Allowing Cleanup

```
def loop():
    ...

if __name__ == '__main__':
    try:
        print('Press Ctrl-C to quit.')
        while True:
            loop()
    except KeyboardInterrupt:
        GPIO.cleanup()
```

Try to make a traffic light

- Your traffic light should:
  - Turn the first LED on for 3 seconds (green light)
  - Turn the first LED off and the second LED on for 1 seconds (yellow light)
  - Turn the second LED off and the third LED on for 3 seconds (red light)
  - And then repeat
- Feel free to customise your traffic light patterns after!

Active vs Passive Buzzer

Passive Buzzer

Requires an AC signal to make a sound; changing the input signal produces different tones.

Active Buzzer

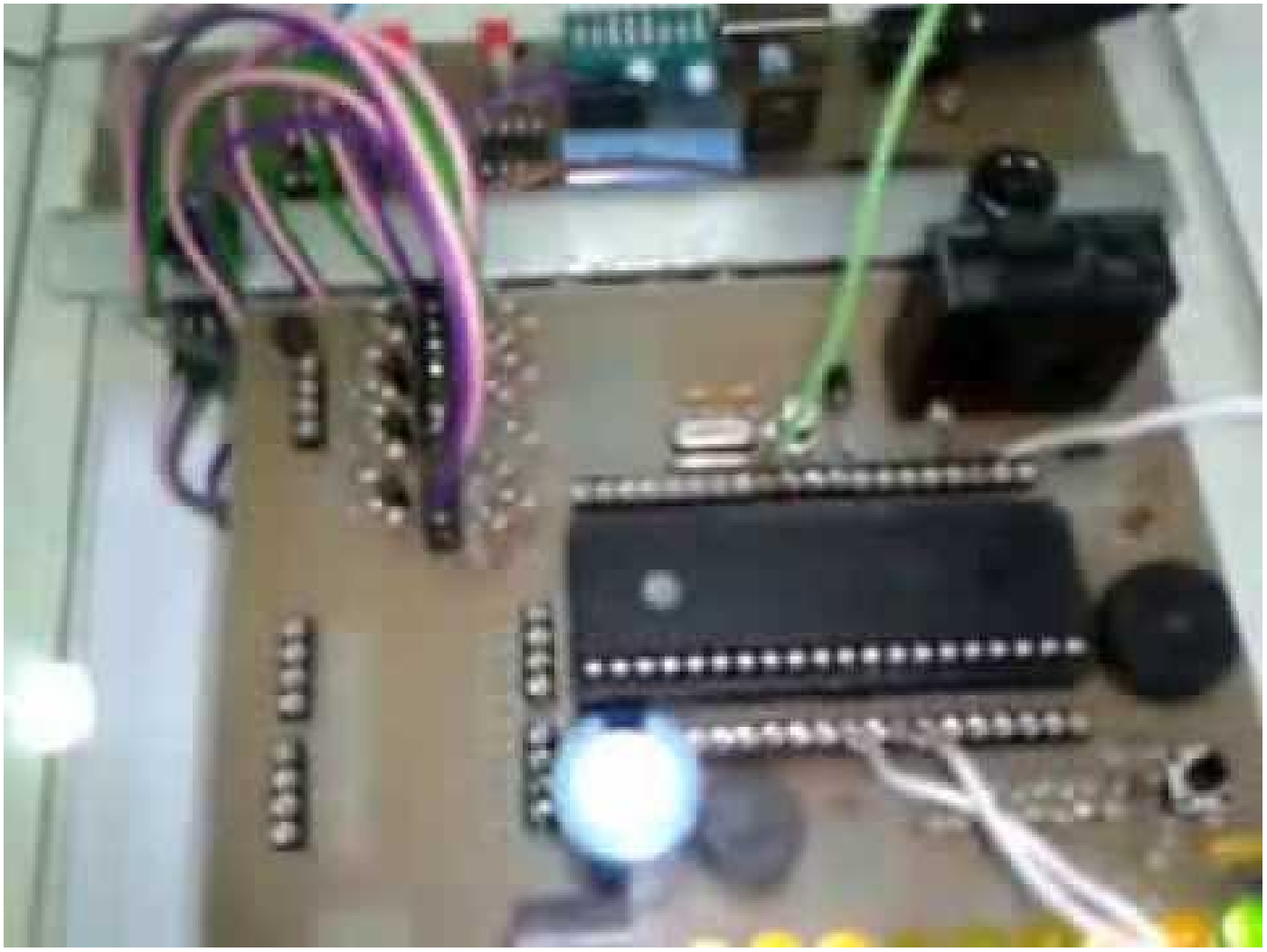
Generates one tone only, can be toggled on/off

Active vs Passive Buzzer

Active Buzzer - generates a tone using an internal oscillator, all you have to do is turn it on/off

Passive Buzzer - requires an AC signal to make a sound; changing the input signal produces the sound, rather than producing a tone automatically. In practice, we need to use some physics knowledge to produce sounds with this.

What Does the Passive Buzzer Sound Like?



Even Floppy Drives Make Better Music...

Better in Surround Sound?

Left as an exercise to the reader...

Pitch and Frequency

The pitch of a sound is determined by the frequency of vibrations of the particles of the medium that the sound wave is passing through.

Frequency measures the number of complete back and forth vibrations over a given time period (i.e. 1 vibration/sec)

We often use Hertz (Hz) to measure frequency.

1 vibration/sec = 1 Hz

If we change the frequency, we can change the pitch.

Passive Buzzer Example Code

Example code:

```
import RPi.GPIO as GPIO
import time
GPIO.setmode(GPIO.BCM)
GPIO.setup(17, GPIO.OUT)
pitch = 523  #C note
duration = 1
#(cont. on next slide)
```

Passive Buzzer Example Code

Example code (cont.):

```
    period = 1.0/pitch
    delay = period / 2
    cycles = int(duration * pitch)

    for i in range(cycles):
        GPIO.output(17,True)
        time.sleep(delay)
        GPIO.output(17, False)
        time.sleep(delay)
```

Passive Buzzer Exercise

Try it out:

Program the passive buzzer to play ‘Mary Had a Little Lamb’

(or your favorite song, just make sure it’s not too complicated!)

Precise musical note frequencies can be found here:

http://www.phy.mtu.edu/~suits/notefreqs.html

Hint: one option is to use lists to store multiple pitches and durations

Passive Buzzer Answer

Example code to play multiple short songs. View the code at the Gist linked below:

tk.sg/passivebuzzer

Add a buzzer to your traffic light

- Make the buzzer sound when the traffic light is red

Connecting the Button

The e-lock push button connects to a Digital port just like our LEDs, but we need to activate the pin to be an input rather than an output:

```
GPIO.setup(25, GPIO.IN)
```

The pin will either have a value of True or False depending on if the button is pressed.

Hint: Use conditional statements to test if True or False!

Add a button to activate your traffic light

- Your traffic light should only loop through when the button is pressed

Traffic Light Answer

```
...

while True:
    if GPIO.input(23):
        GPIO.output(17, True)
        time.sleep(1)
        GPIO.output(17, False)
        GPIO.output(18, True)
        time.sleep(1)
        GPIO.output(18, False)
        GPIO.output(27, True)
        GPIO.output(22, True)
        time.sleep(1)
    else:
        GPIO.output(17, False)
        GPIO.output(18, False)
        GPIO.output(27, False)
        GPIO.output(22, False)
```

Pulsing Lights

When the Macbook added the “breathing” pulse light, it was a big feature! How do we recreate this on the Pi?

What is Analog?

An analog signal has an infinite amount of possibilities.

Picture from https://learn.sparkfun.com/tutorials/analog-vs-digital

While a digital signal has a limited number of possibilities.

Analog Output

We've seen that the Raspberry Pi can write HIGH (3.3V) and LOW (0V) digital voltage levels to light an LED. What if we wanted a brightness somewhere in between?

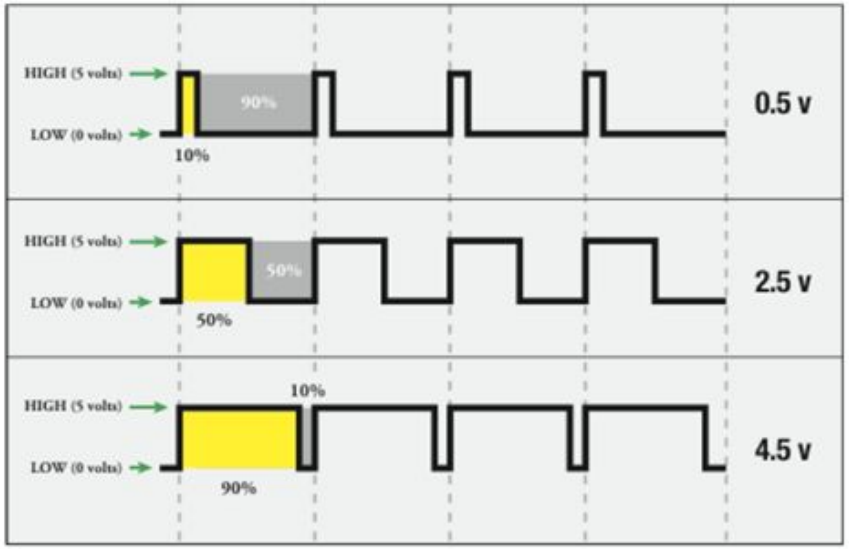
We can use **analog output** to set an arbitrary voltage between LOW (0V) and HIGH (3.3V), for instance 2V.

However, the Raspberry Pi does not have any analog outputs so it cannot truly output an arbitrary analog voltage, but it can *fake* an analog voltage.

Pulse Width Modulation (PWM)

PWM varies the amount of time that the blinking pin outputs HIGH vs. the time it outputs LOW.

Because the pin is blinking much faster than your eye can detect, the RPi creates the illusion of a “true” analog output.



However, do take note that only GPIO pin 18 (D3) is PWM-capable.

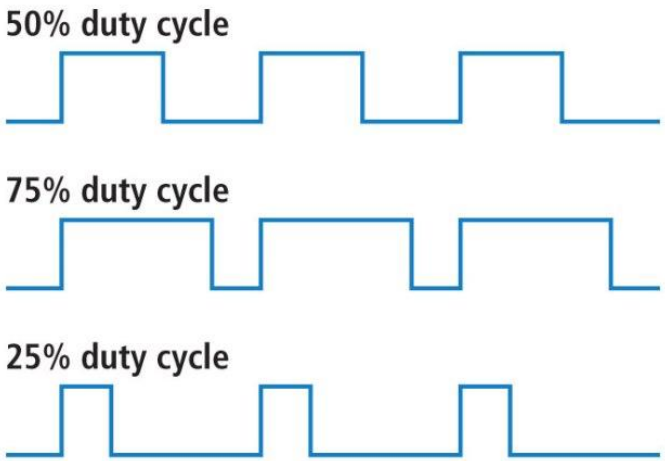
PWM commands

- *To create a PWM instance:*

```
p = GPIO.PWM(channel, frequency)
```

- *To start PWM:*

```
p.start(dc) #dc=duty cycle, 0.0<=dc<=100.0
```



Picture from: https://learn.sparkfun.com/tutorials/pulse-width-modulation

PWM commands

- *To change the frequency:*

  ```
  p.ChangeFrequency(new_freq)
  ```

- *To change the duty cycle:*

  ```
  p.ChangeDutyCycle(new_dc)
  ```

- *To stop PWM:*

  ```
  p.stop()
  ```

PWM Example

```
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BCM)
GPIO.setup(18, GPIO.OUT)

p = GPIO.PWM(18, 50) #frequency=50Hz
p.start(50) #duty-cycle = 50
```

PWM Exercise

Try it out:

Program your LED to pulse slowly on and off.

PWM Answer (part 1)

```
import time
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BCM)
GPIO.setup(18, GPIO.OUT)
p = GPIO.PWM(18, 50)
# pin=18 frequency=50Hz
p.start(0)
while True:
    p.ChangeDutyCycle(20)
    time.sleep(.5)
    p.ChangeDutyCycle(40)
    time.sleep(.5)
    p.ChangeDutyCycle(60)
    time.sleep(.5)
```

(cont. in next column)

```
p.ChangeDutyCycle(80)
time.sleep(.5)
p.ChangeDutyCycle(100)
time.sleep(.5)
p.ChangeDutyCycle(80)
time.sleep(.5)
p.ChangeDutyCycle(60)
time.sleep(.5)
p.ChangeDutyCycle(40)
time.sleep(.5)
p.ChangeDutyCycle(20)
time.sleep(.5)
p.ChangeDutyCycle(0)
time.sleep(.5)
```

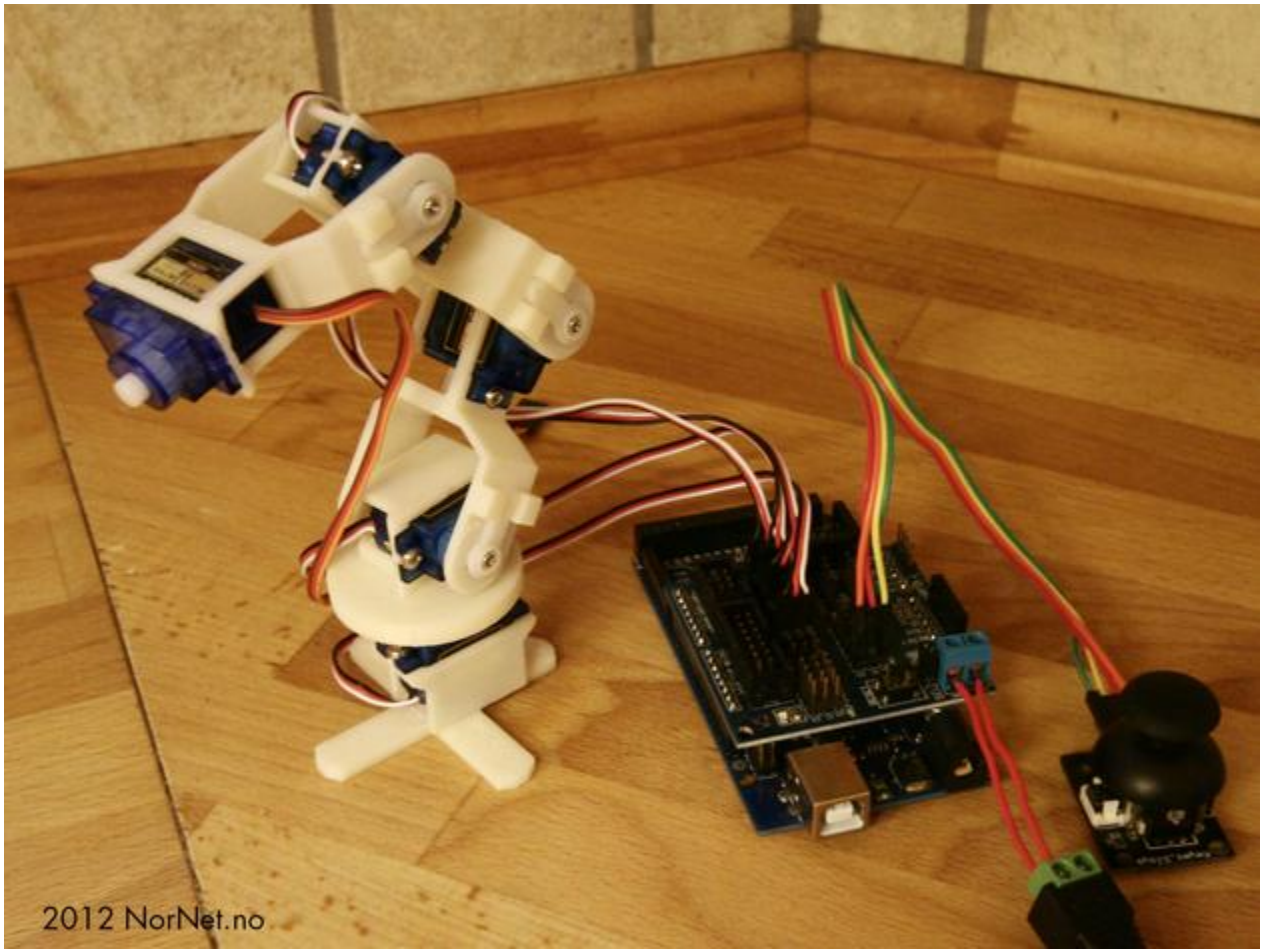
PWM Alternative Answer

```
import time
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BCM)
GPIO.setup(18, GPIO.OUT)
p = GPIO.PWM(18, 50) #channel=18 frequency=50Hz
p.start(0)
try:
    while True:
        for dc in range(0, 101, 5):
            p.ChangeDutyCycle(dc)
            time.sleep(0.1)
        for dc in range(100, -1, -5):
            p.ChangeDutyCycle(dc)
            time.sleep(0.1)
except KeyboardInterrupt:
    pass
p.stop()
GPIO.cleanup()
```

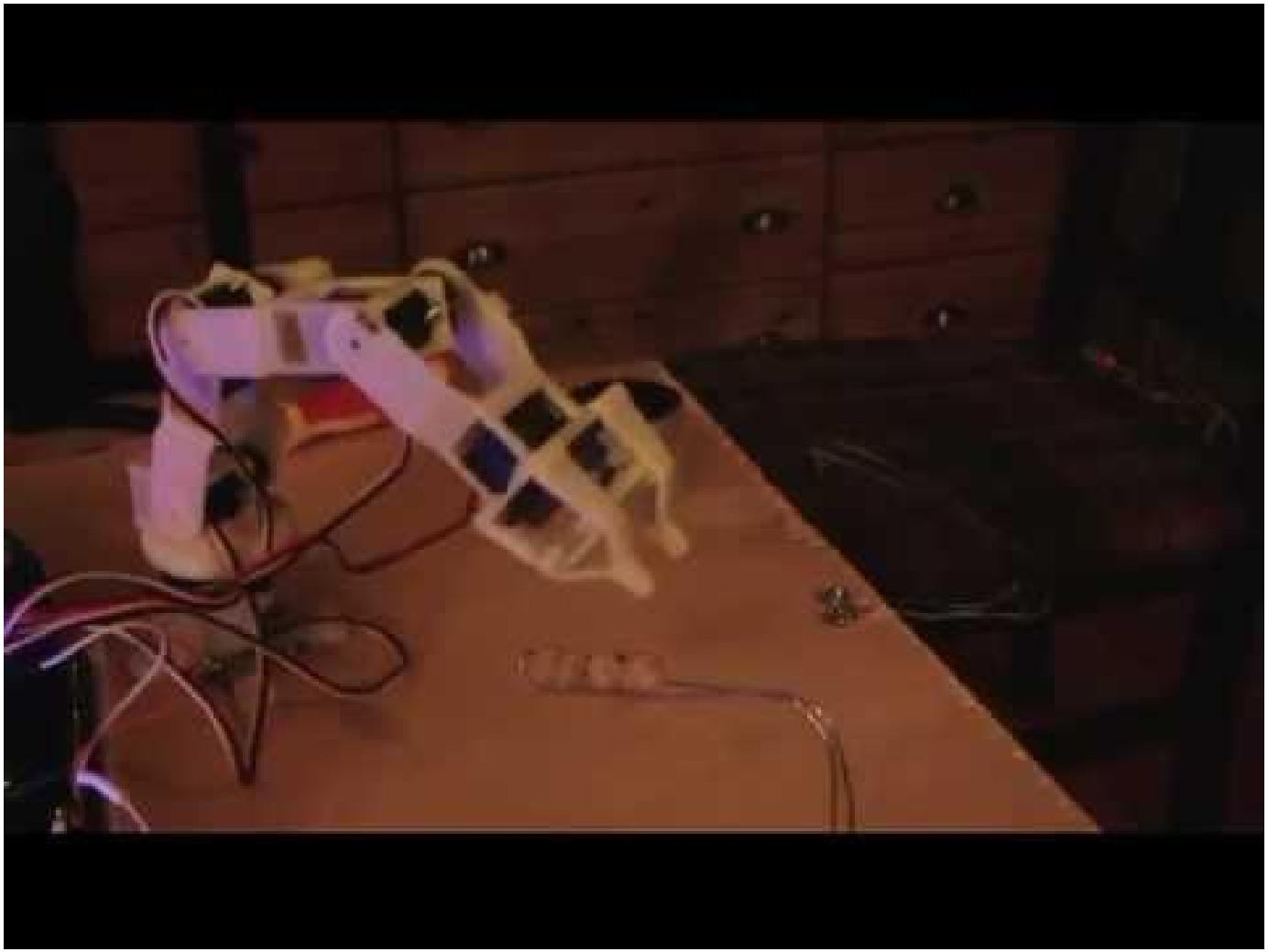
Servo

A servo is similar to a motor, except they typically do not rotate all the way around. Instead, they can be given commands to move precisely to a given angle.

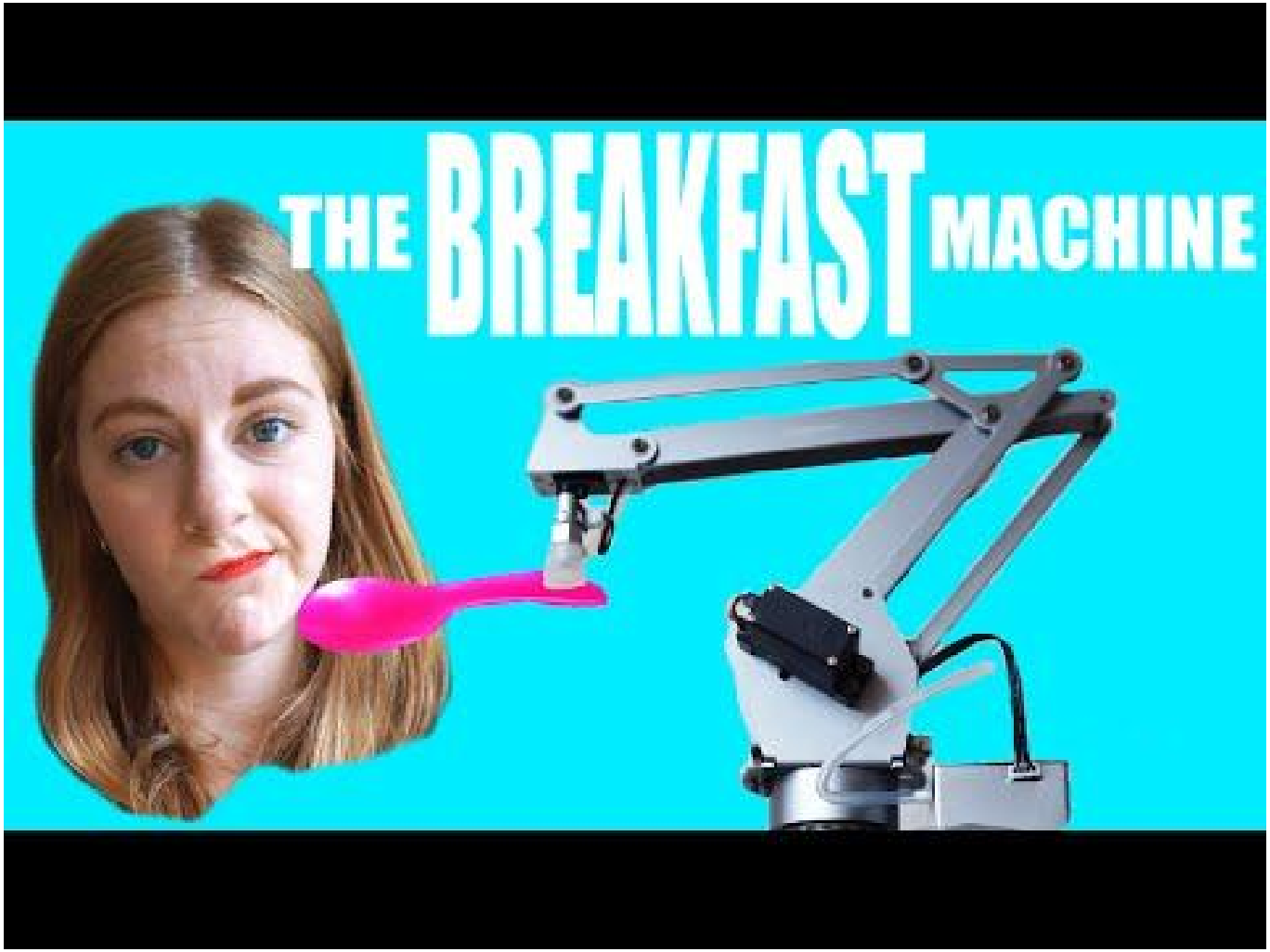
Servo



Servo



Servo



Servo

Servo

Servo

- Our servo isn't as powerful as those used in commercial applications.
- As a result, **BE CAREFUL** turning your servo to angles close to 0° and 180° (or numbers outside these boundaries).
- If the motor strains too much, it can burn out.

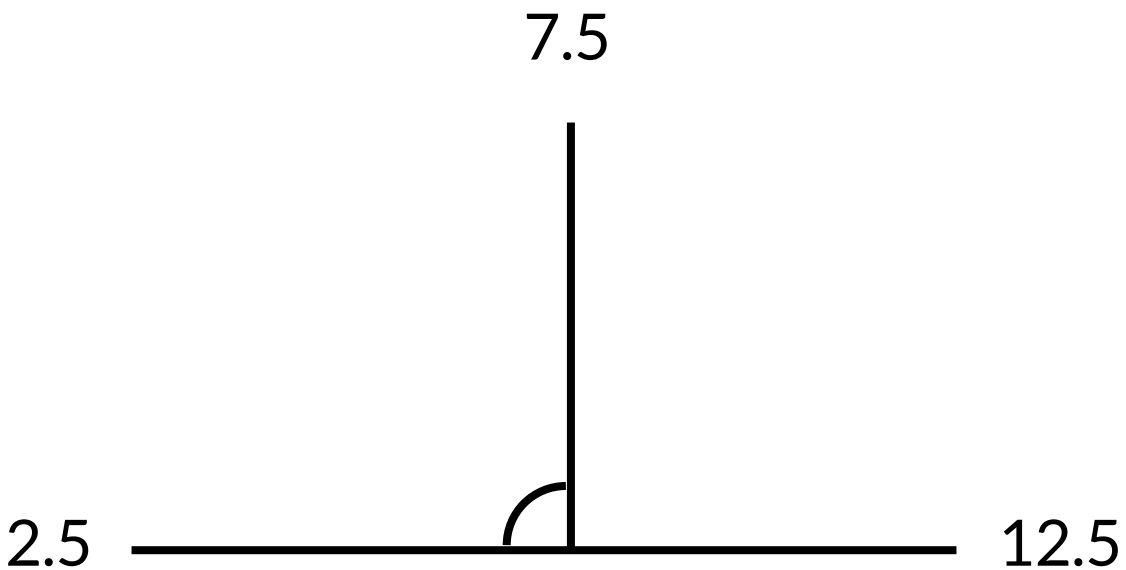
Servo Example

Servo Example

```
import time
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BCM)
GPIO.setup(18, GPIO.OUT)
p = GPIO.PWM(18, 50)
p.start(7.5)
while True:
    p.ChangeDutyCycle(7.5) #90 degrees
    time.sleep(1)
    p.ChangeDutyCycle(11.5) #just under 180 degrees
    time.sleep(1)
    p.ChangeDutyCycle(3.5) #just over 0 degrees
    time.sleep(1)
```

*Be careful about turning your servo to exactly 0° or 180°!

Servo Example



2.5 corresponds to 0 degrees

12.5 corresponds to 180 degrees

Servo Exercise

Try it out:

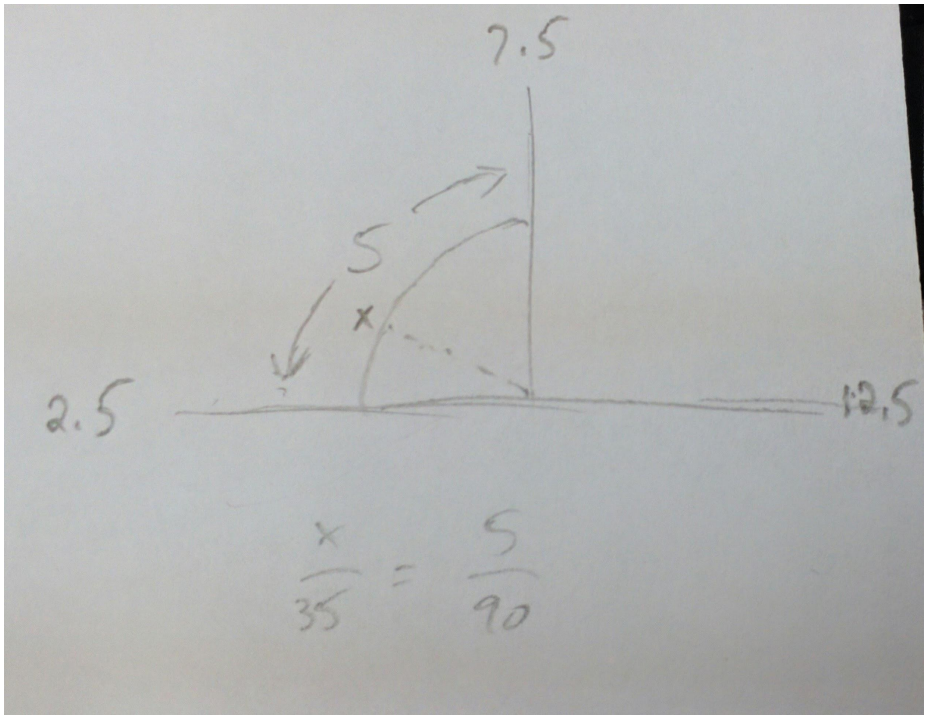
Program your servo to:

- Move to 90° at the start of the program
- Wait until a button is pressed, and then rotate back and forth between 35° and 145° **(be sure to add a short pause in between giving the serve a command to move)**

Servo Exercise – Hint

For precise angles, just use ratios.

(Remember that 2.5 corresponds to 0°, and 12.5 corresponds to 180°)



x=1.94, so the duty cycle for 35° = 1.94 + 2.5 = 3.44

Servo Exercise – Hint

Common angles:

Angle	#
0°	2.50
15°	3.33
30°	4.17
45°	5.00
60°	5.83
75°	6.67
90°	7.50
105°	8.33
120°	9.17
135°	10.00
150°	10.83
165°	11.67
180°	12.50

Servo (Incomplete) Answer

```
p = GPIO.PWM(18, 50)
p.start(7.5)
while True:
    if GPIO.input(27):
        p.ChangeDutyCycle(11.56)
        time.sleep(1)
        p.ChangeDutyCycle(3.44)
        time.sleep(1)
    else:
        p.ChangeDutyCycle(7.5)
        time.sleep(1)
```