Controlling Hardware with the Raspberry Pi using Python

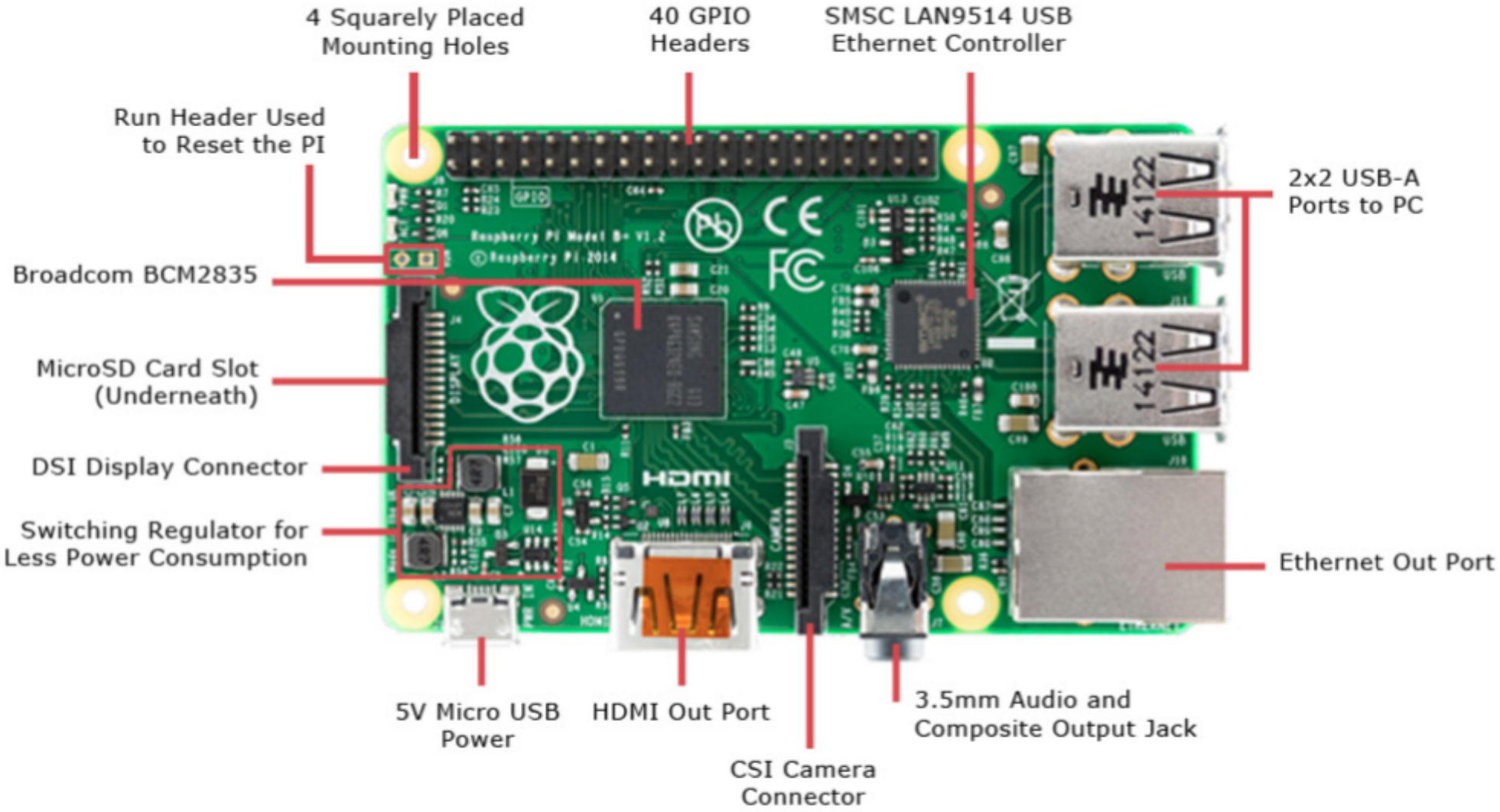
Frontiers in Neurophotonics Summer School 2018

Jamie Boyd
Dongsheng Xiao
Tim Murphy

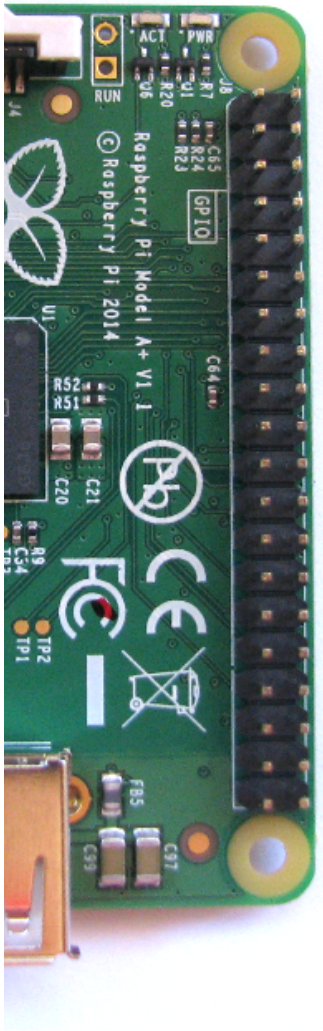
Raspberry Pi Specs (current model is Pi 3B+)

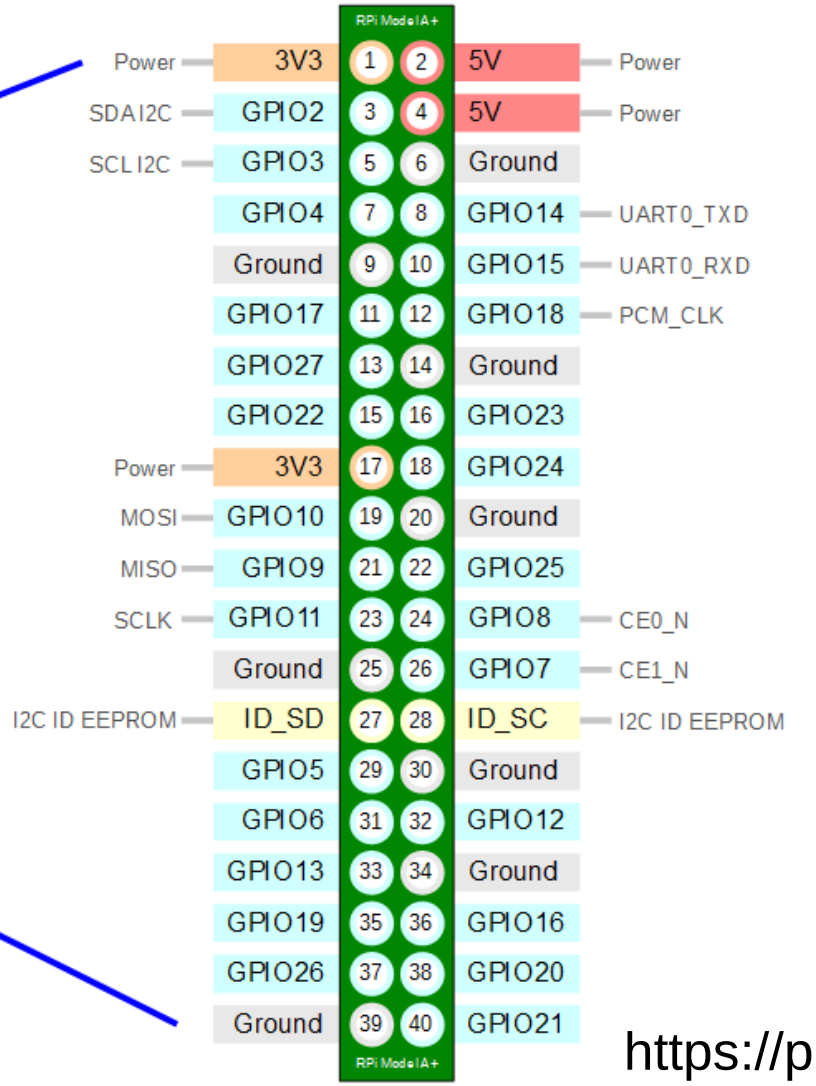
- **A low cost, credit-card sized computer designed to enable people of all ages to explore computing (raspberrypi.org)**
  - **1.2 GHZ quad-core ARM Cortex A53 (900MHz A7 on Pi 2)**
  - **1 GB 900 MHz SDRAM (450MHz on Pi 2)**
- **Runs Raspbian, a Linux-based operating System**
  - **Most of the Linux toolset supported**
  - **Programming in Python, C/C++, Java, ...**
- **Standard Connectivity**
  - **10/100 MBPS Ethernet**
  - **802.11n Wireless LAN (not on Pi 2)**
  - **Bluetooth (not on Pi 2)**
  - **4 USB Ports**
  - **HDMI video**
  - **micro SD card (no disks)**
  - **40 GPIO Pins for General Purpose Input/Output plus some specialized communication protocols (serial, i^2c, spi)**

Raspberry Pi 2



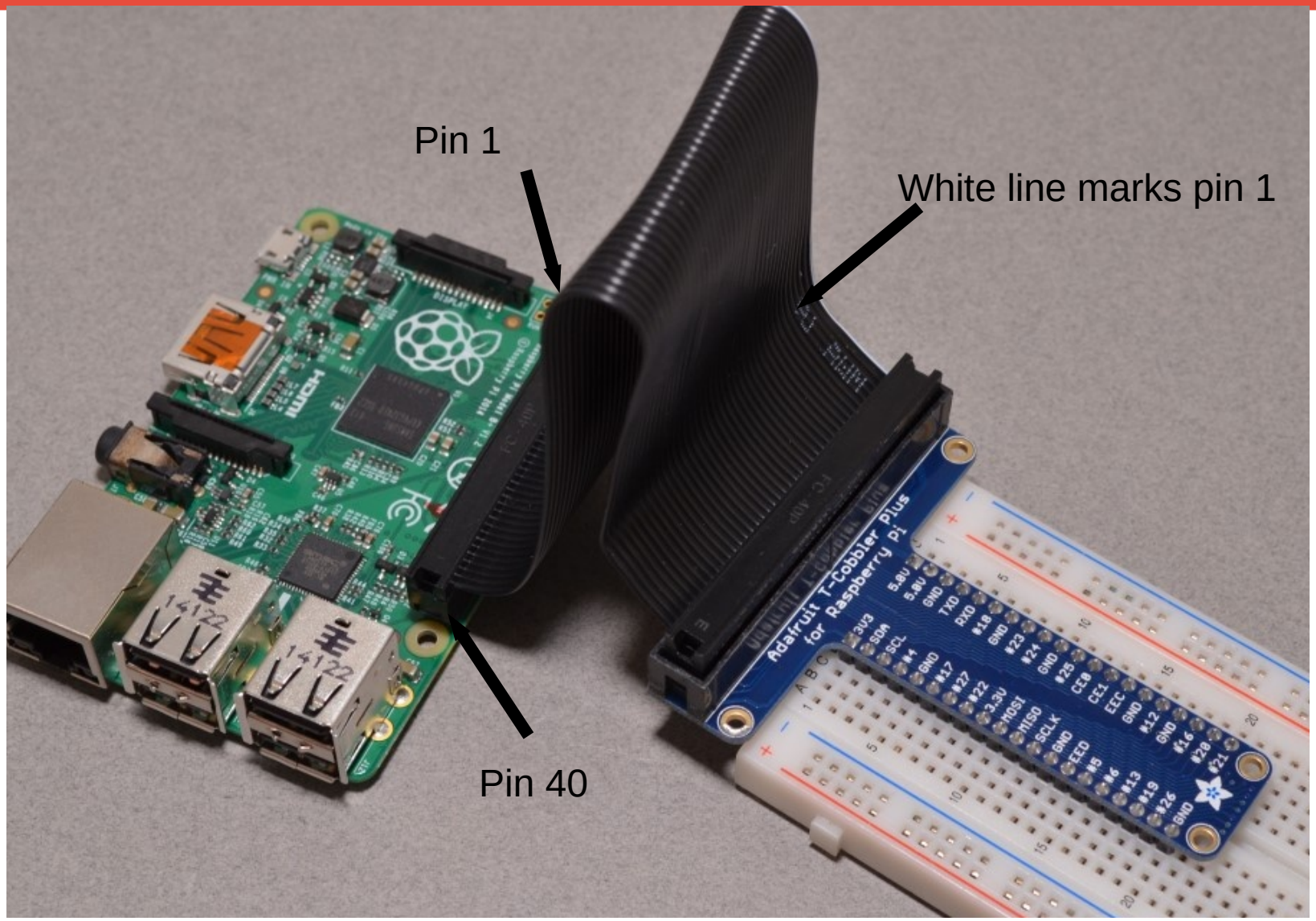
Rasberry Pi GPIO Header Pin Out





https://pinout.xyz/pinout/

T-Cobbler Plus and Bread-board



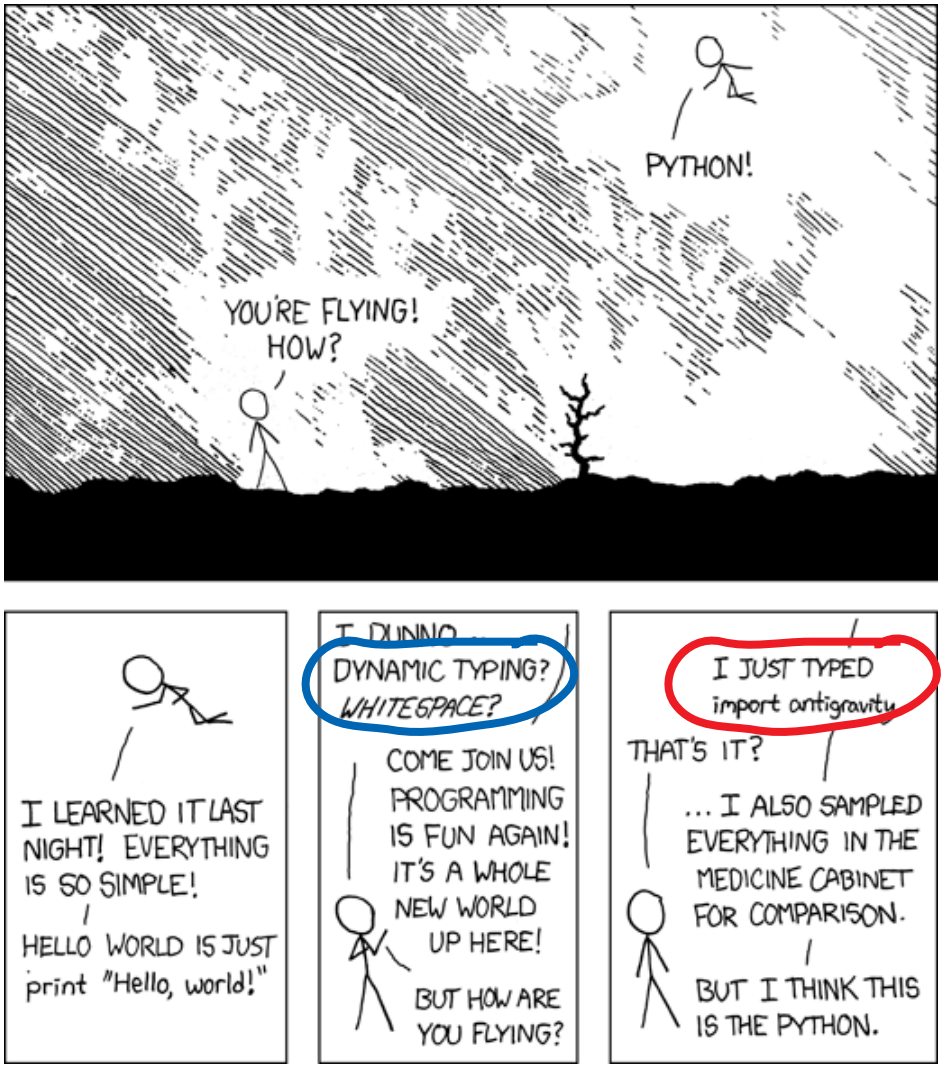
Booting Up

- **Connect micro-USB power to boot up**
  - Watch text scroll by until log-in prompt is presented
  - Login: pi
  - Password: raspberry
- **Launch GUI from the command line**
  - `$startx` ($ is the command prompt, don't type $)
- **Open a terminal window to get a command line back**
  - Raspberry menu->accessories->terminal
  - or select the picture of a computer monitor on task bar at top of screen. Also open a directory window from a task bar icon

Python Programming Language

- Python (both 3.x and legacy 2.7x) is installed in Raspbian by default
  - Always use Python 3
- Python is an interpreted language
  - each line of a program executed in turn by python interpreter (slower than C)
  - Can be used interactively, running code as you type it
- Python has great library support

  import RPi.GPIO



https://xkcd.com/353/

Running Python in a Terminal Window

- **Run a python file with python interpreter, specifying python version 3**
  - **`$python3 hello_world.py`**

    Hello World!

    ```
    print ('Hello World !')
    ```
- **Run python interpreter interactively in a terminal**

  `$python3`

  Python 3.53 (default, Jan 19 2017, 14:11:04)

  [GCC 6.3.0 20170124] on Linux

  Type “help”, “copyright”, “credits” or “license” for more information

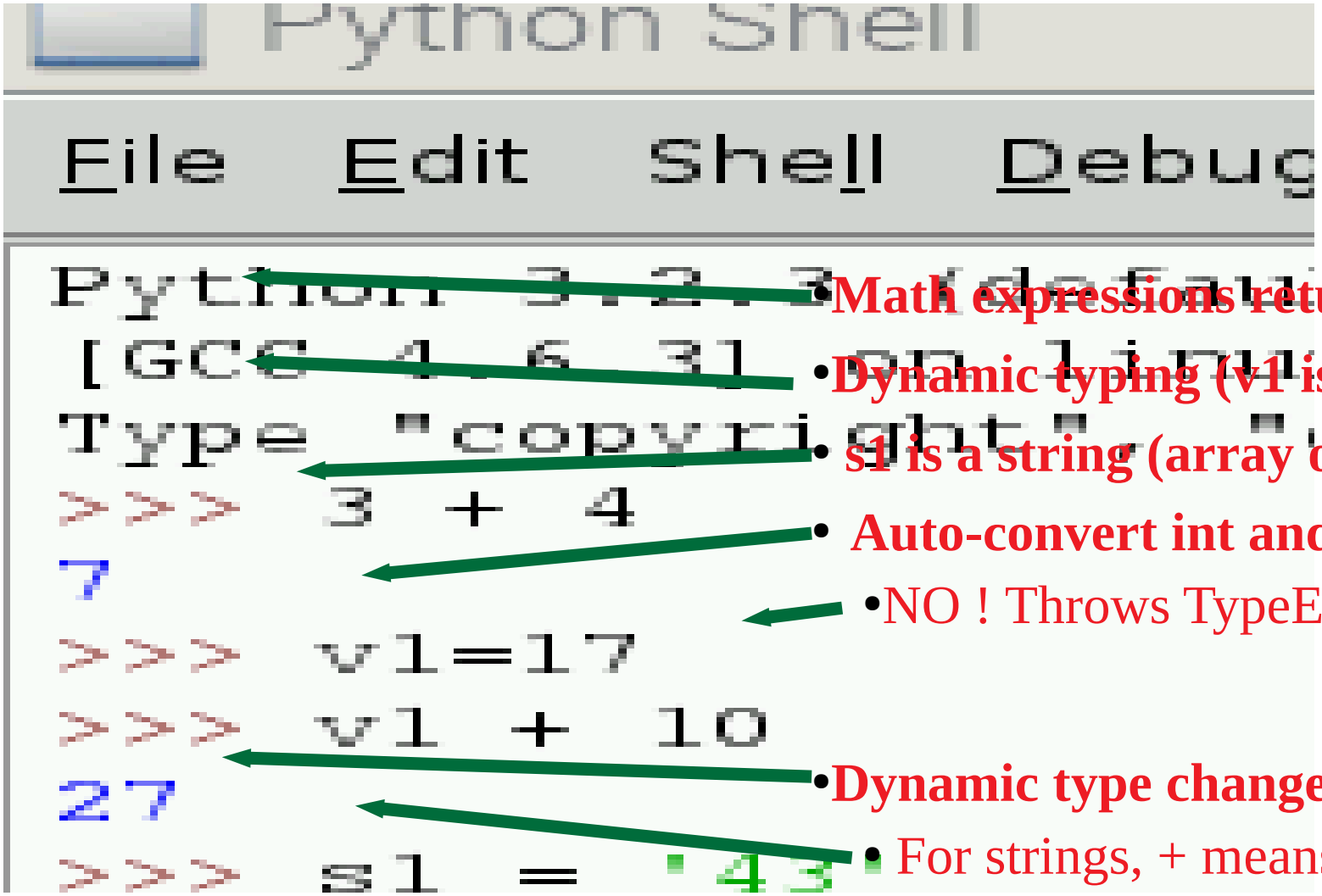
  >>> **(>>> is the python interpreter prompt, don’t type >>>)**

  >>> quit() **(quits the python interpreter)**

Python Integrated DeveLopment Environment

- **IDLE is a Simple Python Integrated Development Environment (IDE)**
  - Python shell – behaves like Python used interactively in a terminal window
  - Multi-window text editor with
    - syntax highlighting
    - Autocompletion
    - smart indent.
- **Launch IDLE from raspberry menu->programming or from the command line:**
- **`$gksudo idle3 &`**
  - Use gksudo to launch with elevated permissions to access /dev/mem to use GPIO
  - Use & so terminal window can be used while IDLE runs

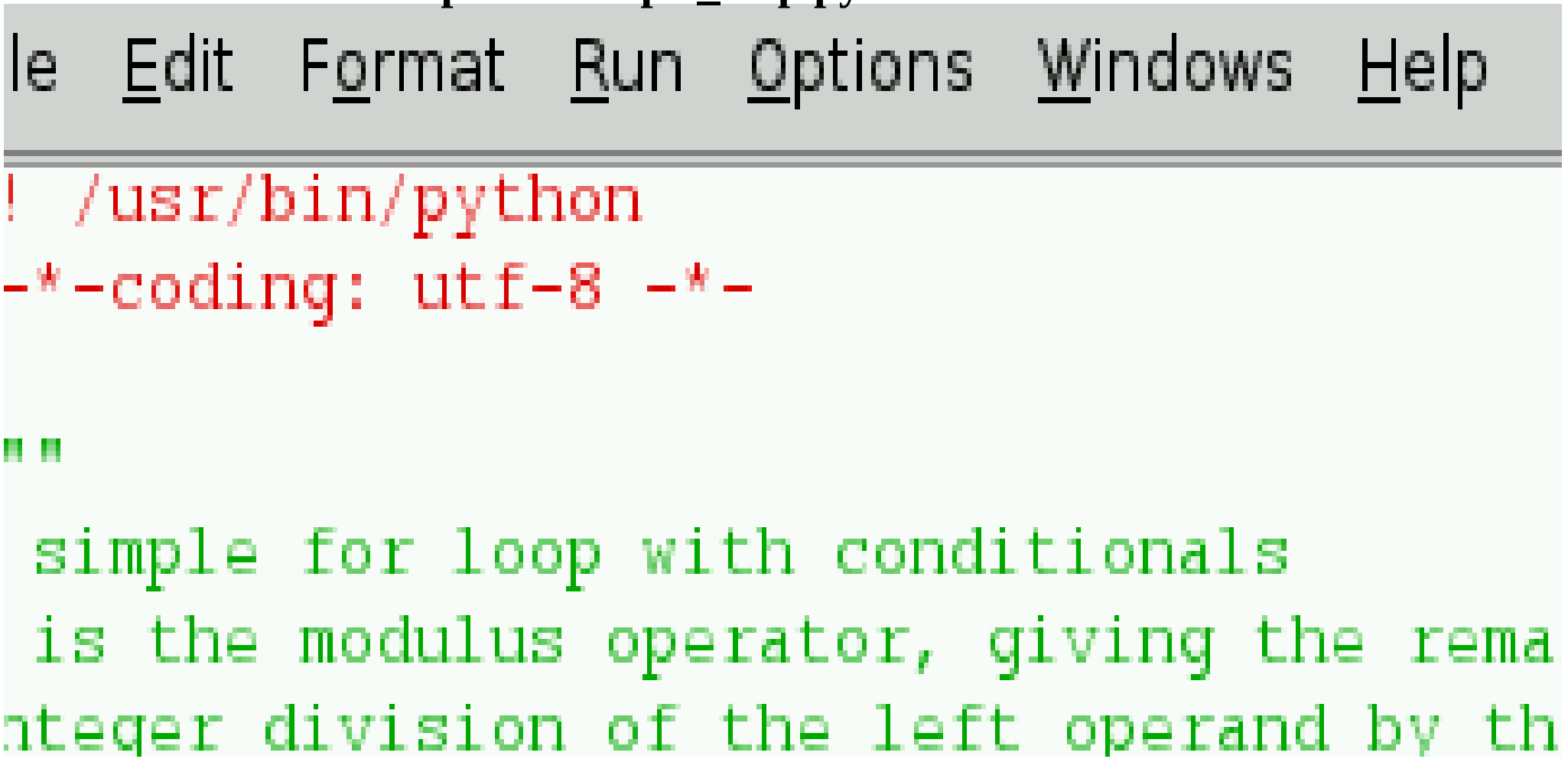
Interactive Python in IDLE Shell



- **Math expressions return result**
- **Dynamic typing (v1 is an integer)**
- **s1 is a string (array of characters)**
- **Auto-convert int and str?**
  - NO ! Throws TypeError exception
- **Dynamic type change of v1 to string**
  - For strings, + means concatenation

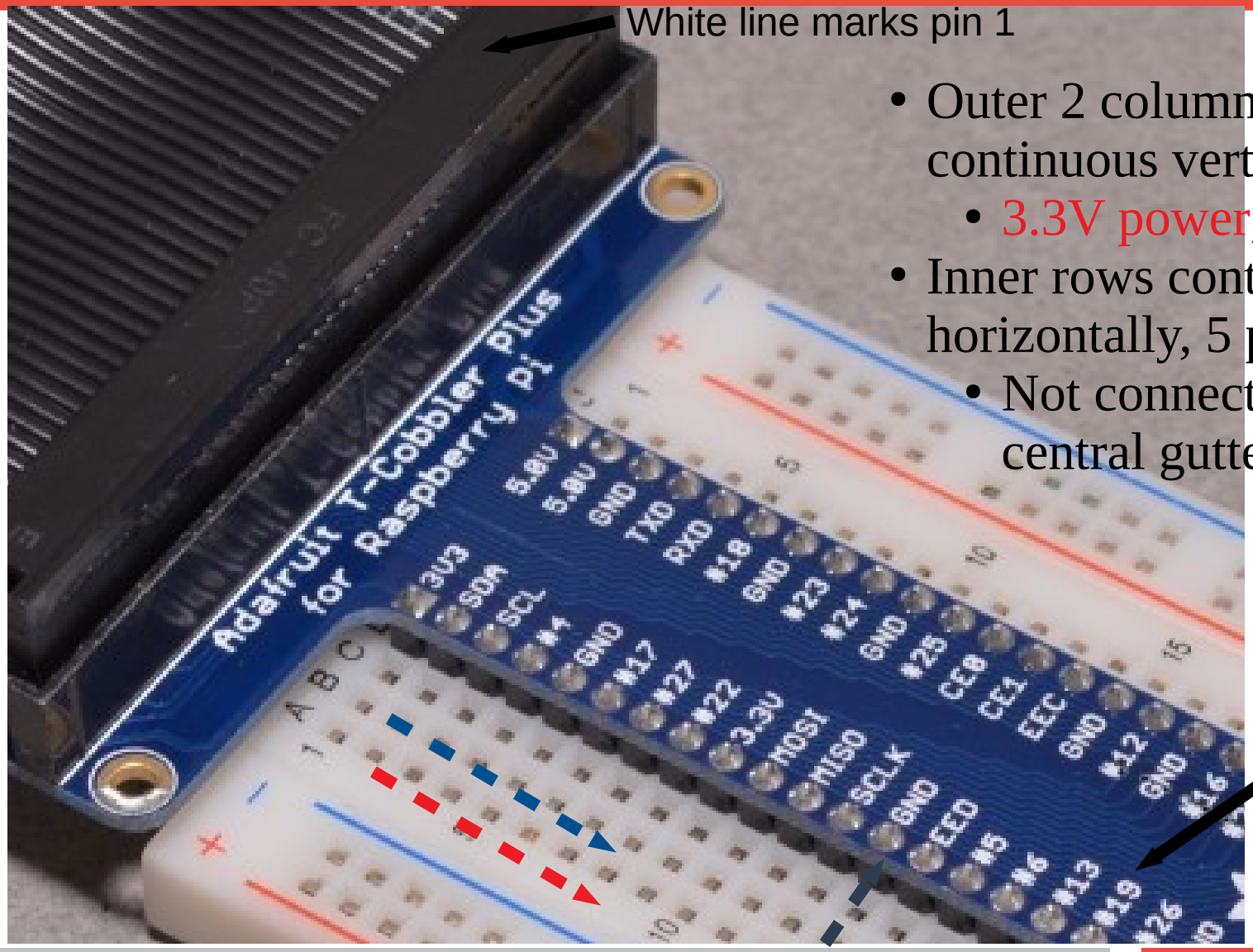
Loops and Conditionals in a Python Program

- **From File Menu -> Open -> simple_loop.py •From Run Menu -> Run Module**



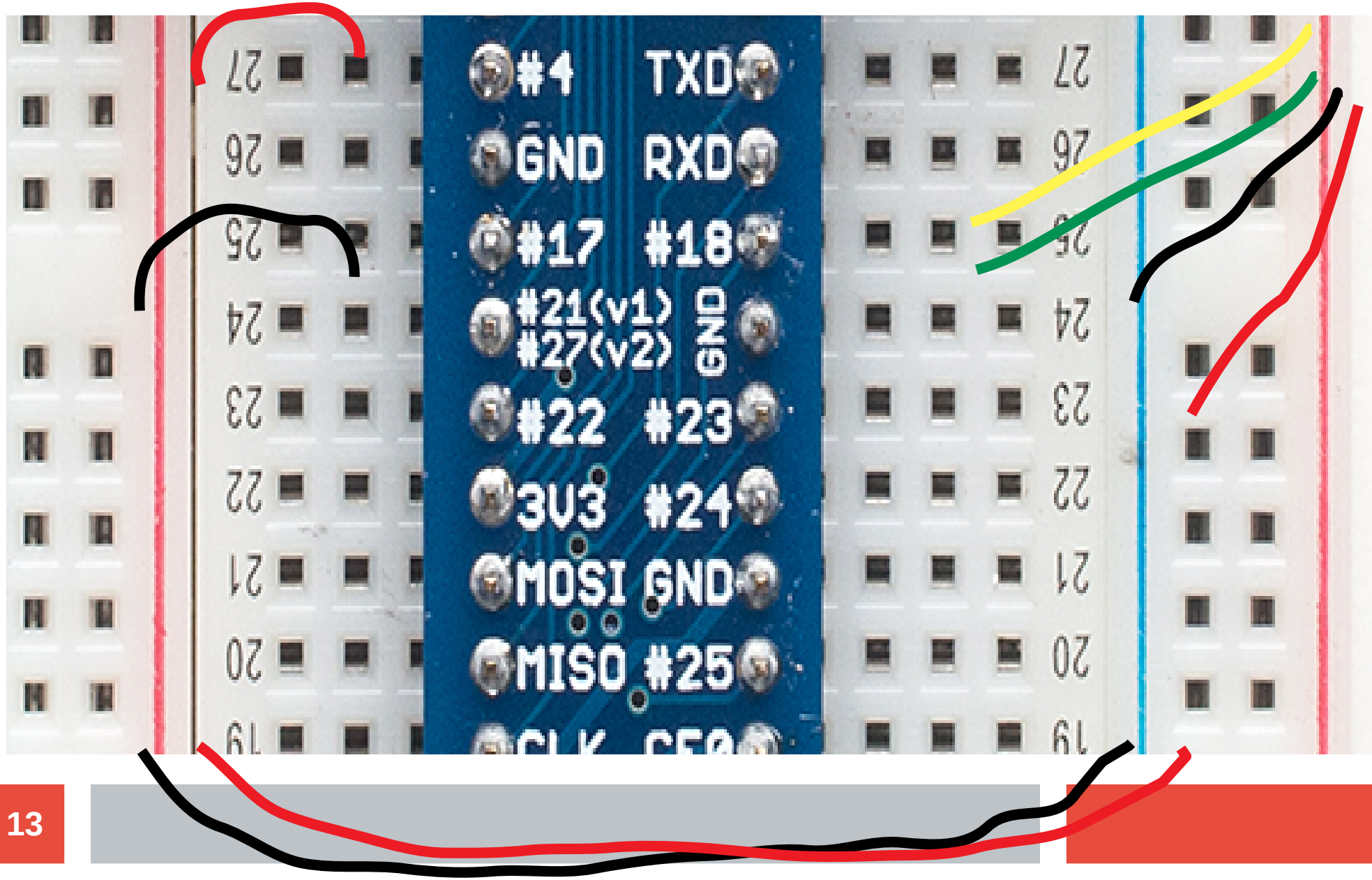
- **White space is syntax: loop body, and if blocks are indented using tab key**

T-Cobbler Plus and Bread-board



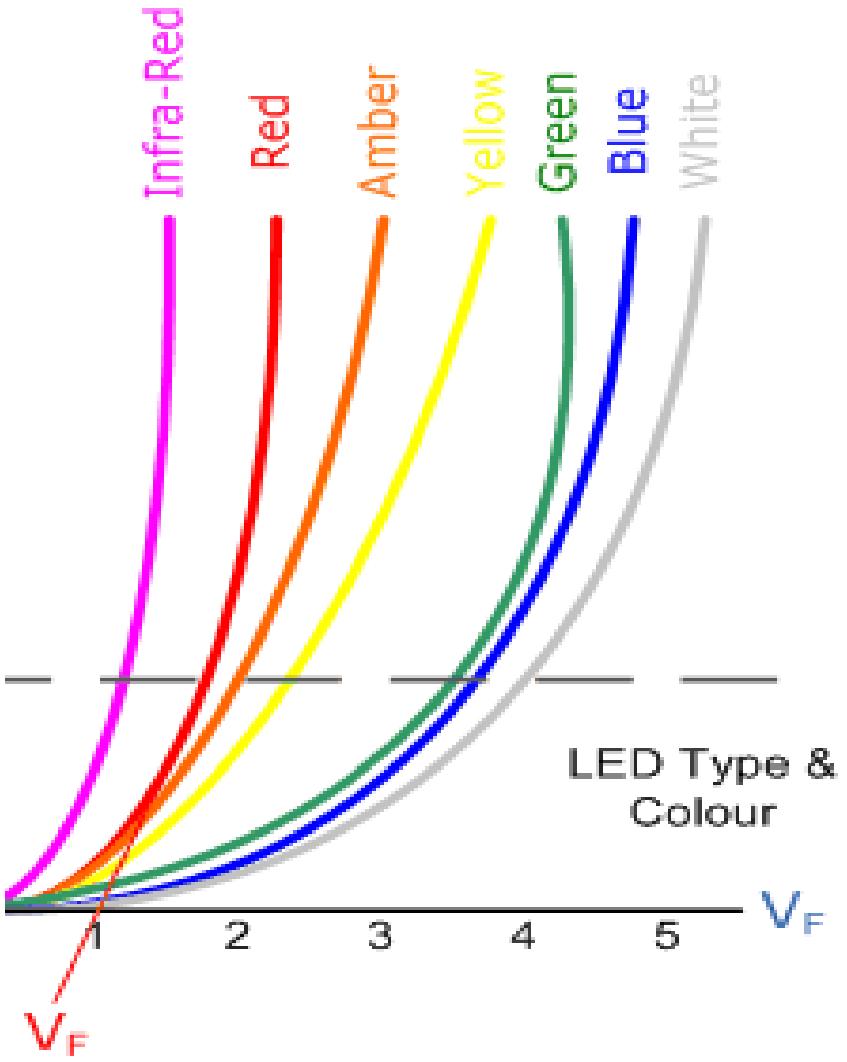
- Outer 2 columns continuous vertically
  - 3.3V power, ground
- Inner rows continuous horizontally, 5 pins/row
  - Not connected across central gutter

T-Cobbler Plus and Bread-board Serial Example



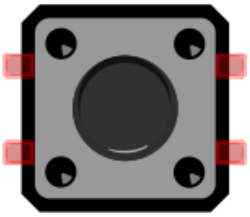
LED = Light Emiting Diode

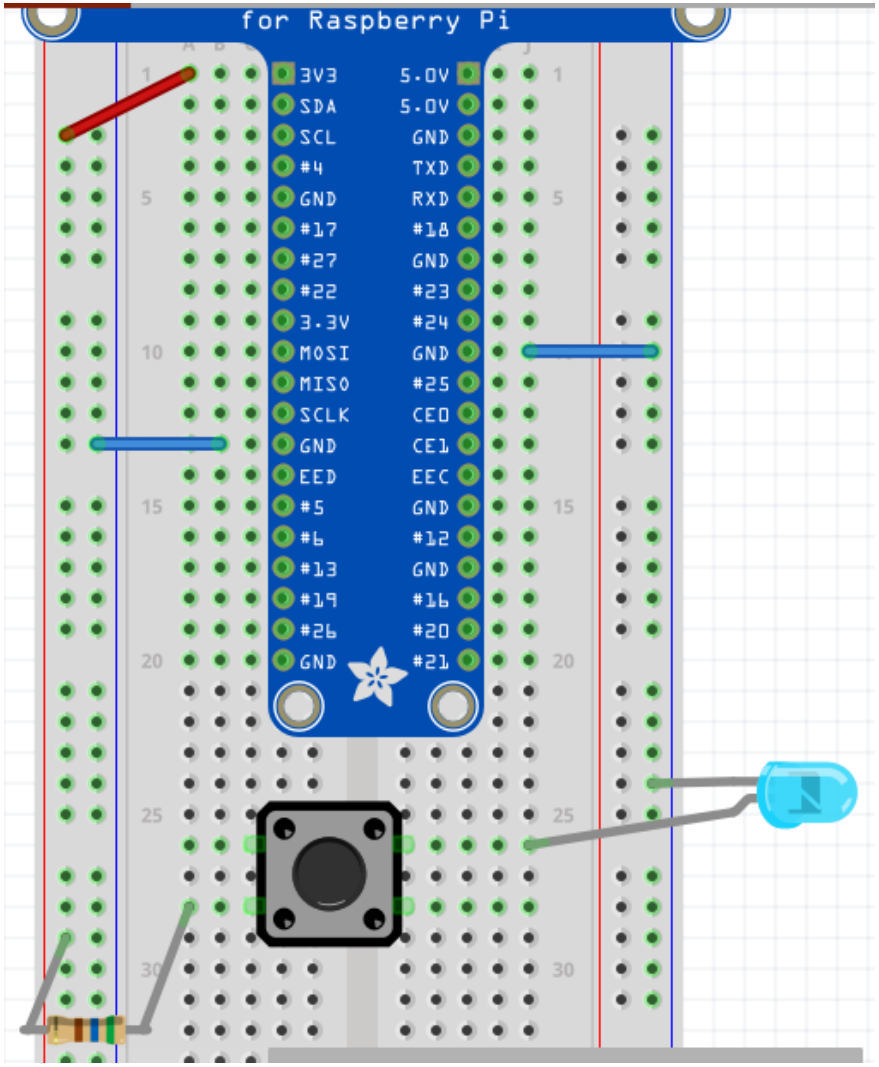
- LED has a polarity
  - Current only flows one way in a diode
  - Reverse breakdown voltage > 20V
- Forward voltage = 1.5 to 3.5 V
  - Red < Green < Blue
- max current = 20 mA. Too much current/ heat destroys LED
- I/V curve is exponential. Hard to limit Current by controlling Voltage
  - Use Ballast Resistor in series
  - Current = Voltage/Resistance
    - Ohm's Law
  - Assume voltage drop on LED = V_F
  - Current =(V – LED V_F)/ballast R



Switch an LED (Manually)

- Left Power rail wired to Pi 3.3 V
- Ground rails wired to Pi ground
- 560 Ohm resistor to limit current.
  - (3.3V - 2V)/560 Ohm= 2.3 mA
- Momentary switch connects when pressed.
  - 4 pins for mechanical stability. The 2 pins across each row are permanently connected.
  - Make it span the gutter, connect diagonals
- power->resistor->switch->LED->ground
  - Order of resistor, switch, LED not important
- LED has a polarity
  - No light, flip LED
  - short leg to ground

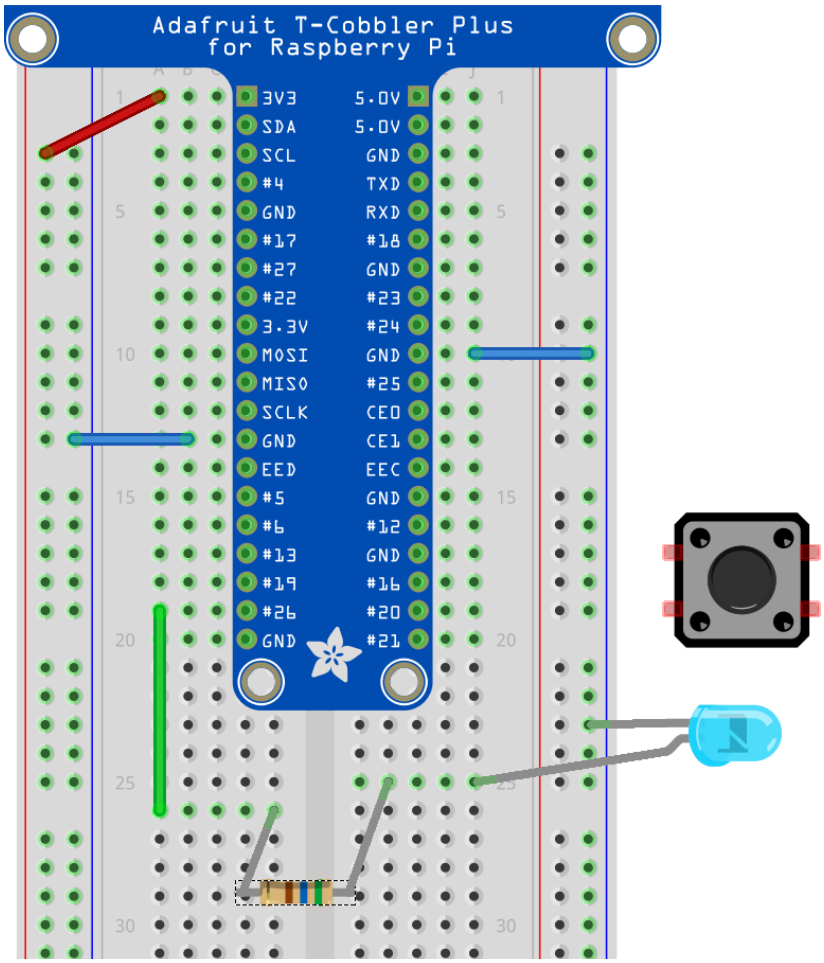




Raspberry Pi Digital Outputs

- Digital output is High or Low (3.3V CMOS)
  - Set High 3.3v -> GPIO pin -> load
  - pin sources current drives voltage to >= 2.4 V
  - Set Low load -> GPIO pin -> gnd
  - pin sinks current pulls voltage to <= 0.5 V
- How much Current can a GPIO pin sink/source and still maintain Voltage within specification?
  - configurable per pin from 2 to 16 mA
  - Default is 8 mA
  - Exceed that current and voltage on pin sags
  - The total current from all pins must not exceed 51mA or you may damage the pi
- Higher resistance load = lower current draw
  - Use a transistor to drive low resistance loads

GPIO 26 -> Resistor -> LED -> Ground



GPIO 26 -> Resistor -> LED -> 3.3V ?

Switch LED using Python library RPi.GPIO

sourceforge.net/p/raspberry-gpio-python/wiki/

In a terminal, $gksudo idle3 & to open idle with access to /dev/mem. In Python:

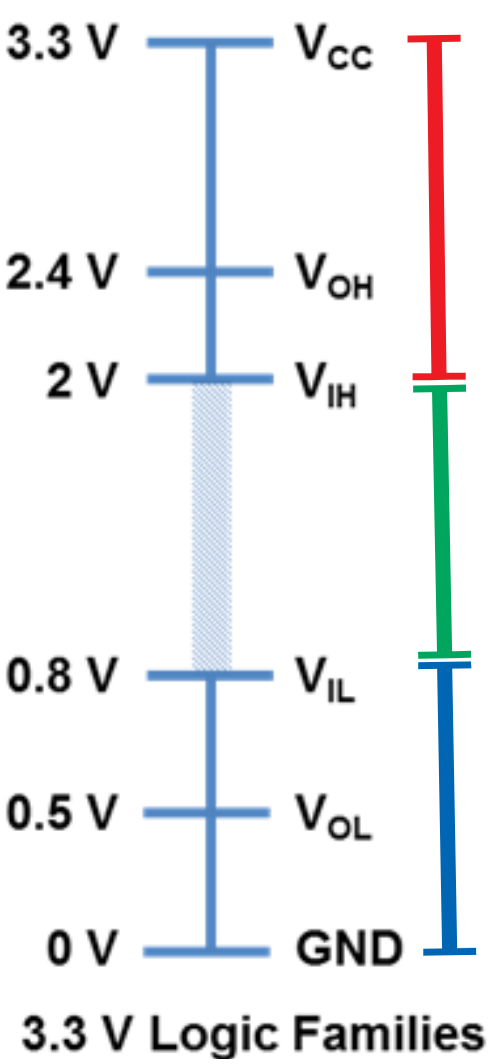
```
>>> import RPi.GPIO as GPIO "as" so you can refer to it as GPIO
>>> GPIO.setmode(GPIO.BCM) Broadcom numbers, not header numbers
>>> GPIO.setup(26, GPIO.OUT) set up GPIO 26 for output
>>> GPIO.output(26, GPIO.HIGH)sets GPIO 26 to 3.3 V
>>> GPIO.output (26, GPIO.LOW) sets GPIO 26 to ground
>>> GPIO.cleanup(26) when we are done with GPIO 26
```

Blink an LED using a Python Script blink.py

From File Menu -> Open -> blink.py From Run Menu -> Run Module

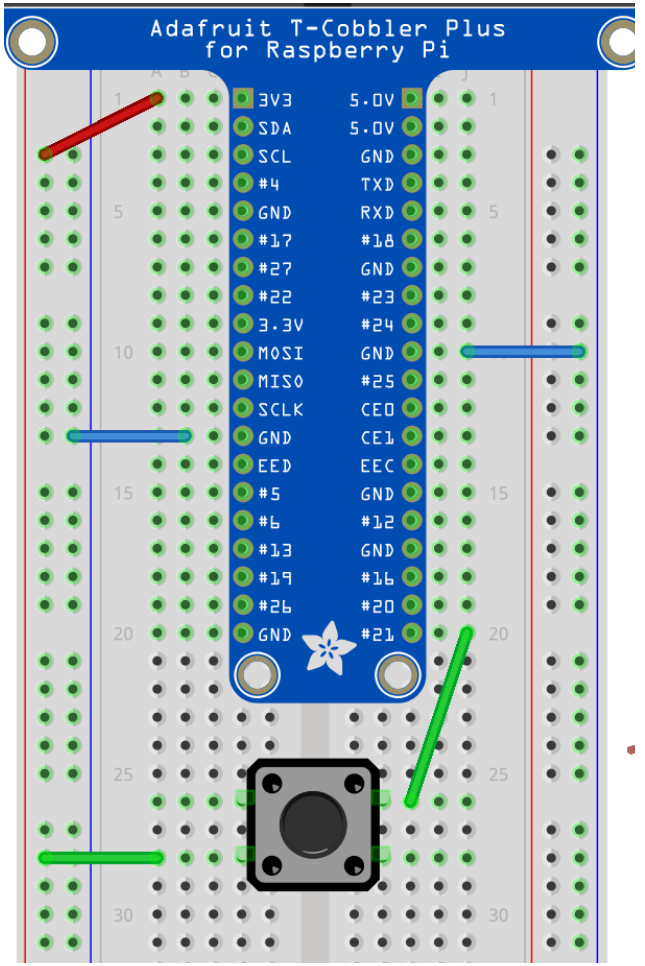
```
import RPi.GPIO as GPIO # import Rpi.GPIO library
from time import sleep  # for timing LED on and off periods
the_pin = 26            # declare variables
on_time = 0.5
off_time= 0.5
blinks = 10
GPIO.setwarnings(False) #warns if GPIO pin is already in use
GPIO.setmode(GPIO.BCM)  # always use Broadcom pin numbering
GPIO.setup(the_pin,GPIO.OUT)    # set pin for output
for blink in range (0, blinks): # loops for blinks times
    GPIO.output(the_pin,GPIO.HIGH) #LED turns on
    sleep (on_time)                # sleep for LED on time
    GPIO.output(the_pin,GPIO.LOW)  # LED turns off
    sleep (off_time)               # sleep for LED off time
GPIO.cleanup ()         # prevents warnings from pin in use
```

Raspberry Pi Digital Inputs

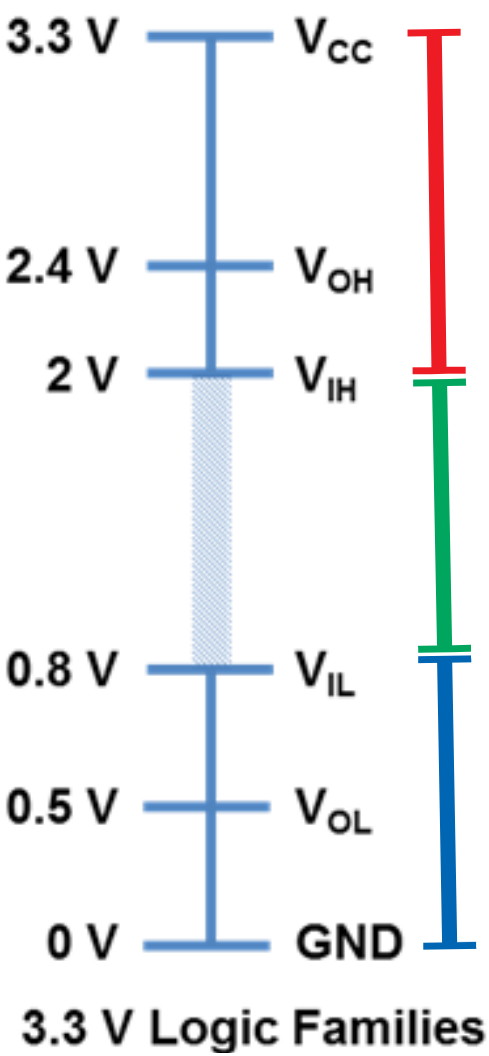


3.3 V Logic Families

- Inputs are read as high if above 2 volts
- Inputs are read as low if below 0.8 volts
- Inputs between those levels are undefined
- Can you see a problem with this circuit ?
- Keep 0 < input < 3.3 V to prevent damage to the Pi
  - The Pi GPIO pins are NOT 5V tolerant

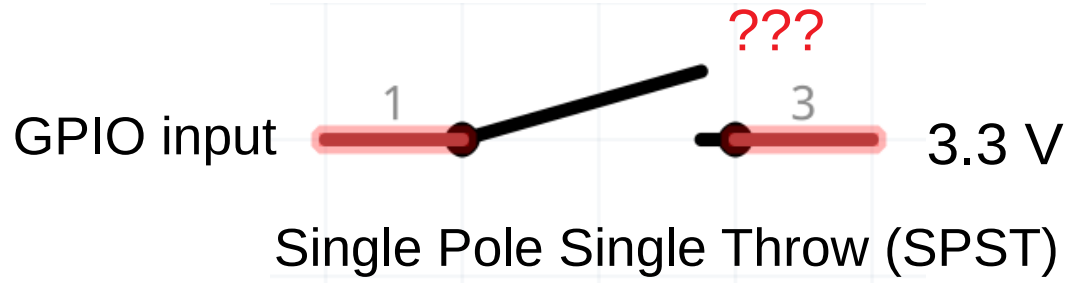


Raspberry Pi Digital Inputs

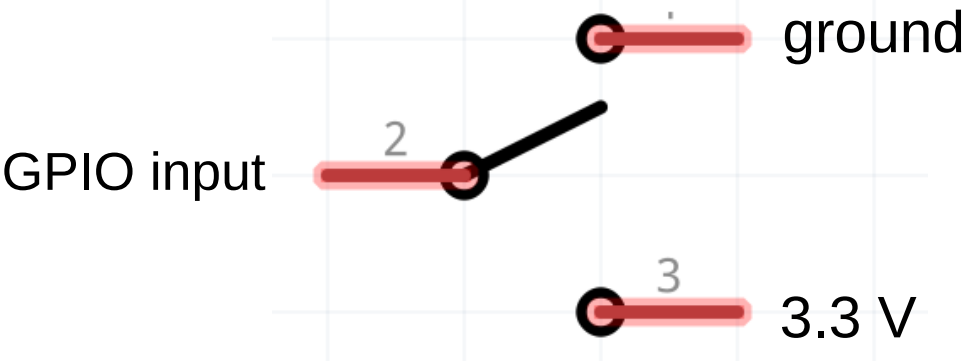


3.3 V Logic Families

- Your momentary switch is Single Throw (closed or open)
- How can we change the circuit to get reliable inputs ?
  - high when switch is pushed
  - low when switch is not pushed



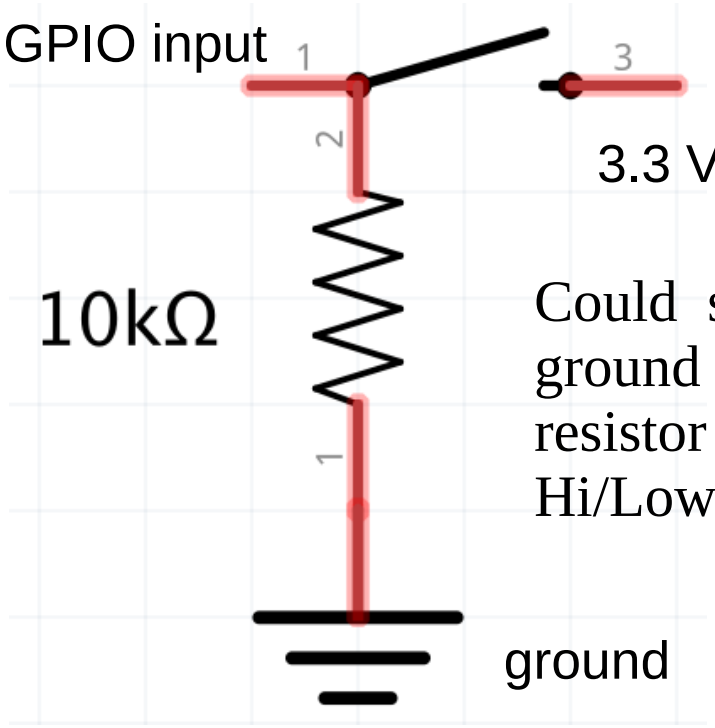
Single Pole Single Throw (SPST)



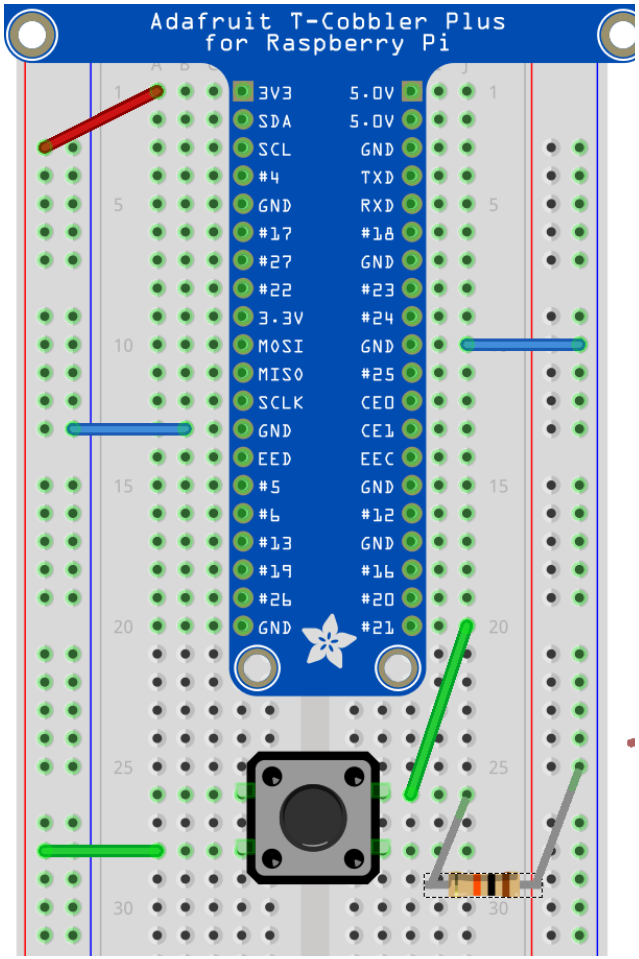
Single Pole Double Throw (SPDT)

GPIO input with a pull-down Resistor

- **With a pull-down resistor, input is pulled down to ground through resistor when switch is open**
- **When switch is closed, current draw from voltage source is 3.3 V/10 k = 0.33 mA**



Could swap the 3.3V and ground and have a pull-up resistor to 3.3V, with switch Hi/Low logic reversed



Read Digital Inputs using RPi.GPIO

```
>>> GPIO.setup(21, GPIO.IN)  # set up GPIO 21 for input
>>> GPIO.input (21)          # returns 1 if GPIO pin is High, 0 if pin is Low
```

From File Menu -> Open -> check_input.py **From Run Menu -> Run Module**

```
import RPi.GPIO as GPIO # gksudo idle3 for access to /dev/mem
from time import sleep
in_pin = 21
GPIO.setmode(GPIO.BCM)
# enable built in pull-down resistor on GPIO Pin (also PUD_UP)
GPIO.setup(in_pin,GPIO.IN, pull_up_down = GPIO.PUD_DOWN)
while True: # infinite loop
    try:       # exceptions try-catch
        if GPIO.input(in_pin) is GPIO.HIGH:
            print ('GPIO pin ' + str (in_pin) + ' is high')
        else:
           print ('GPIO pin ' + str (in_pin) + ' is low')
        sleep (0.1)                # 1/0.1 sec = 10Hz rate
    except KeyboardInterrupt: # ctrl-c triggers interrupt
        GPIO.cleanup()
        break                      # breaks out of infinite loop
```

GPIO Input: Edge Detection

- An edge is the change in state of an electrical signal from LOW to HIGH (rising edge) or from HIGH to LOW (falling edge).
  - Edge changes are **events**, as opposed to **states** (high and low)
  - We are often more interested in events than states
- Checking GPIO input level repeatedly in a loop is called **polling**.
  - Processor intensive to poll at a high frequency
  - Easy to miss events polling at too low a frequency

From File Menu -> Open -> check_input_edge.py From Run Menu -> Run Module

```
while GPIO.input(in_pin) is GPIO.LOW:#polling for low-to-high
    sleep (sleep_time) # replace with pass for performance
    # pass
print ('GPIO pin ' + str (in_pin) + ' went HIGH')
```

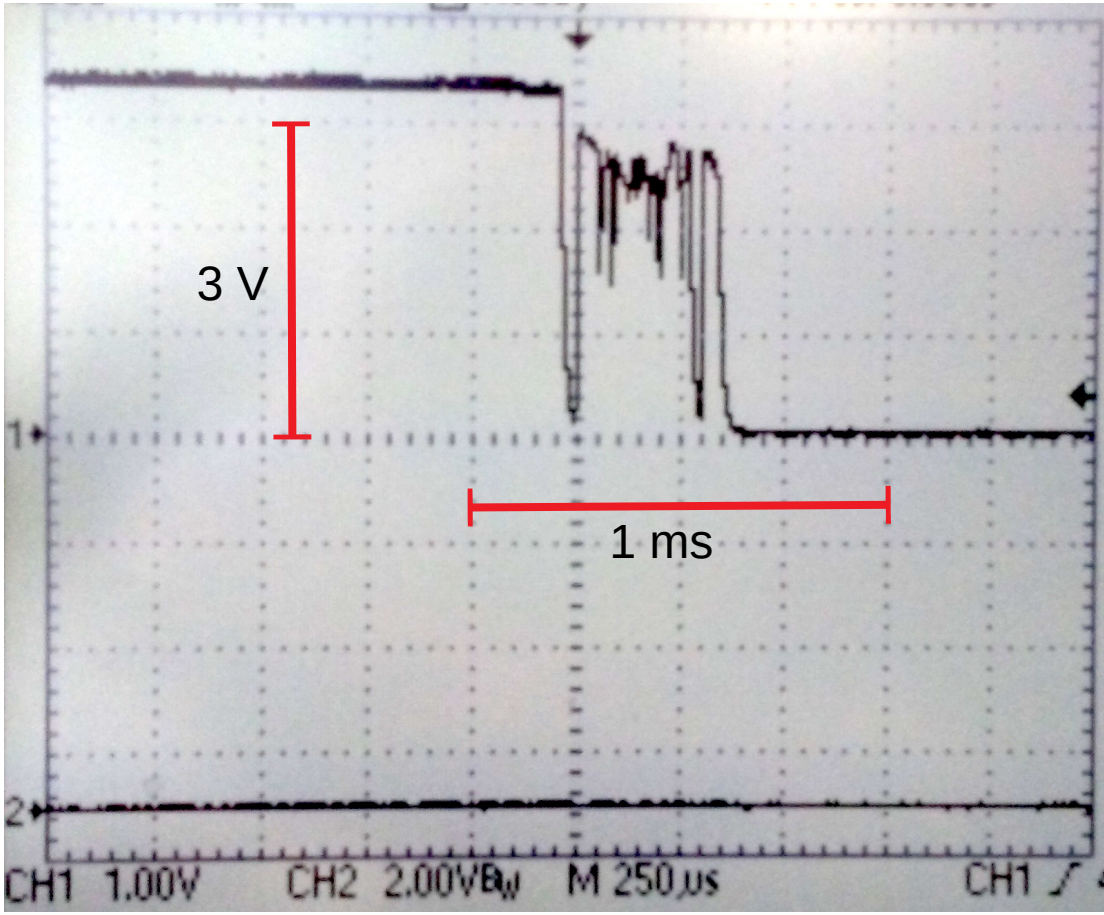
Rpi.GPIO functions for Edge Detection

`GPIO.wait_for_edge (GPIO pin, event,[bouncetime, timeout])`

- event can be `GPIO.RISING`, `GPIO.FALLING` or `GPIO.BOTH`
- bouncetime in milliseconds
  - transitions are ignored unless a constant state is maintained for this many milliseconds (to mitigate physical switch bounce)
  - Too short a bounce time gives extra events
  - Too long a bounce time may ignore real events
- Timeout in milliseconds
  - Function returns after the timeout expires, even if an event has not occurred
- Returns GPIO pin number if an event has occurred before the timeout, returns None (a special Python object) if there is a timeout.
- **Doesn't miss the event, doesn't use processor time while waiting**
- **Your program is blocked while waiting for event or timeout**

Edge Detection: Switch Debounce

- Mechanical switches bounce when switched, rapidly making and breaking contact until they settle (1 ms or more depending on switch).
  - Bounces give rise to phantom events
- Software debounce or filter with capacitor



GPIO input: wait_for_edge.py

```
# Set up a pin for input as usual
GPIO.setup (in_pin, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)

# each time through loop, call wait for edge
result = GPIO.wait_for_edge(in_pin, GPIO.BOTH, bouncetime=
bounce_time_ms, timeout=time_out_ms)

# check for timeout
if result is None:
  print ('No button press on GPIO pin ' + str (in_pin) + ' for '
  + str (time_out_ms/1000) + ' seconds.')

# As we waited for both rising and falling, check level
if GPIO.input (in_pin) is GPIO.HIGH:
  print ('GPIO pin ' + str (in_pin) + ' went HIGH')
else:
  print ('GPIO pin ' + str (in_pin) + ' went LOW')
```

GPIO input: edge_detected.py

GPIO.add_event_detect(pin, edge,[callback, bouncetime])

GPIO.event_detected(pin)

•Designed to be used in a loop where you want to do other things without constantly checking for an event.

- Your program is not blocked, and not constantly polling
- event_detected has a memory, but only for the most recent event
- If more than one event since last check, previous events will be lost

```
# Set up a pin for input as usual, then add event to detect
 GPIO.setup (in_pin, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
 GPIO.add_event_detect(in_pin, GPIO.BOTH, bouncetime =
bounce_time_ms)

 # in the loop, do some work, then check if an event happened
 sleep (processing_time) # simulates doing other processing
 result = GPIO.event_detected(in_pin) # if True, an event
```

GPIO input: edge_counter.py

- Callback functions are threaded (threads run concurrently with data shared)
  - A callback function runs at the same time as your main program, in immediate response to an edge. `my_callback (pin)`
  - Global variables share information with callback and main program

```
rising_edges =0 #define globals outside of any functions
falling_edges =0
# to share a global variable between two threads, you
need two functions. we define a main function
def main ():
    # body of main function goes here


# tell Python to run main() when file is opened
if __name__ == '__main__':
    main()
```

GPIO input: edge_counter.py

The callback function that runs in a separate thread when an edge is detected

It increments global variable rising_edges if a low-to-high transition, or falling_edges if a high-to-low transition

```
def counter_callback(channel):
    global rising_edges #global, same variable in main
    global falling_edges
    if GPIO.input (channel) is GPIO.HIGH:
        rising_edges +=1  #increment rising_edges
    else:
        falling_edges +=1 #increment falling_edges
```

GPIO input: edge_counter.py

In main(), we occasionally check how many events have occurred. We **only read** global variables in main (), **never write to them** as the callback runs concurrently

```
global rising_edges  #the global tells Python these variables
global falling_edges #are declared outside of the function

sleep (processing_time) # simulates doing other processing

#rising_edges – last_rising = rising edges in processing_time
print ('GPIO pin ' + str (in_pin) + ' went HIGH ' +
str (rising_edges - last_rising) + ' times in the last ' + str
(processing_time) + ' seconds')

last_rising = rising_edges     # keep track of values before
last_falling = falling_edges   # next period of sleeping.

print ('GPIO pin ' + str (in_pin) + ' went HIGH a total of ' +
str (rising_edges) + ' times') # at ctrl-c, print totals
```

GPIO: a Challenge

Think about how you would write a program that waits for a button press and lights an LED when the button is pressed.

Combine whichever of the GPIO input and output functions that you like. There's more than one way to do this.

Yes, you can light an LED from a switch without using a Raspberry Pi, but keep in mind that:

GPIO output to more "interesting" devices is the same as GPIO output for lighting an LED

GPIO input from a more "interesting" device is the same as processing input from a button press

PiCamera module v2.1

The PiCamera camera module has specs similar to a cell phone camera

- **Sony IMX219 silicon CMOS back-lit sensor**
- **8 megapixel (3280 × 2464 pixels)**
- **400 to 700 nm spectral response (also a noIR version)**
- **ISO settings between 100 and 800 in steps of 100, or auto gain**
- **exposure times between 9 µs and 6 s (Rolling Shutter)**
- **24 bit RGB output (10-bit A/D converter on chip, resampled to 8)**
- **Movies:**
  - **up to 15 frames per second (fps) full-frame imaging video**
  - **up to 30 fps in 1080p - faster, more reliable, with reduced image size**
  - **Frames will be dropped if Pi is taxed (pre-allocate an array ?)**

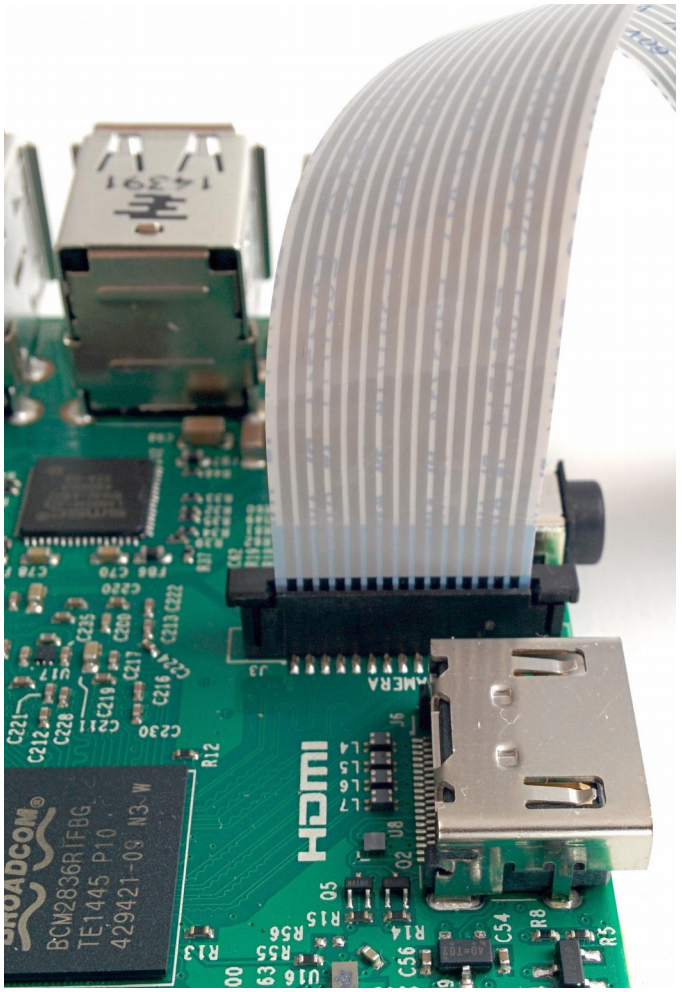
Proper PiCamera Cable Insertion

- Biggest cause of problems is reversed cable insertion
- On both the Pi and the camera
- Avoid kinks in the cable (can connect adjacent pins)

- Shutdown pi
  - Select shutdown from raspberry menu, or
  - **`$sudo shutdown -t 0 -h -P now`**
  - Unplug power before plugging in camera

- Plug ribbon cable into slot by HDMI connector
  - Avoid the similar display connector at front of pi
  - Ethernet and USB at rear of pi
  - Silver connectors side of cable faces to front of pi
  - Lift tabs, insert cable, press down tabs

- Reboot pi
- Login:pi
- Password:raspberry
- **`$startx`**



PiCamera Python Interface

picamera.readthedocs.io/en/release-1.13/

```
import picamera
camera = picamera.PiCamera()   # constructor for PiCamera object
camera.resolution = (640, 480) # horizontal size, vertical size
camera.framerate = 30
camera.preview_fullscreen = False
camera.preview_window = (20, 40, 680, 500) # left, top, right, bottom
camera.start_preview()#draws directly to the screen
camera.capture ('test.jpg') #takes a single image
camera.start_recording(name.h264)#extension sets filetype
```

many more settings than shown here (gain, white balance, ASA)
but defaults are o.k.

camera_test.py

```
Preview running at this point
Sleep the cpu, wait for keyboard interrupt
while True:
  try:
     time.sleep(0.1)
#Press CTRL+C to stop the preview and take picture
  except KeyboardInterrupt:
      camera.capture ('test.jpg') #take single image
      camera.stop_preview()
      camera.close()
      break
```

To see the picture, open a folder window from the task bar, and double-click on your picture (it's not your father's Linux anymore)

GPIO and movies: camera_on_button.py

```
>>> from time import time
>>> from datetime import datetime
```

- time() returns seconds since 1970/01/01 - use this to grab time stamps
- datetime is for human readable date/time formats. A datetime object has fields for year, month, day, hour, minutes, seconds, microseconds

  `datetime.fromtimestamp (time()).isoformat (' ')`

- fromtimestamp returns a datetime object from number of seconds since 1970/01/01

  .isoformat (' ') returns a string with formatted date, parameter is separator character, a space in this case

```
>>> now = time()
>>> dt = datetime.fromtimestamp(now)
>>> dt.isoformat(‘ ‘)
```

camera_on_button.py

```
result = GPIO.wait_for_edge(in_pin, GPIO.FALLING,
bouncetime= bounce_time_ms, timeout = 1000)
if result is None:
    continue   # continues at top of loop (ctrl-c)
camera.start_recording(base_name + str(trial_num) +
'.h264') # h264 is a video format, with compression
 startSecs= time()
 camera.start_preview()
 isRecording = True
 print ('Started recording ' + base_name +
str(trial_num) + '.h264 at ' + datetime.fromtimestamp
(int(startSecs)).isoformat (' '))
```

camera_on_button.py

```
timed_out = not GPIO.wait_for_edge(in_pin,
GPIO.FALLING, bouncetime= bounce_time_ms,
timeout=max_movie_time)
camera.stop_recording()
endSecs = time()
camera.stop_preview()
isRecording = False
if timed_out:
    print ('movie ' + base_name + str(trial_num) +
'.h264 stopped because of time out after ' +
str( endSecs -startSecs) + ' seconds')
else:
    print ('movie ' + base_name + str(trial_num) +
'.h264 stopped because of button up after ' +
str( endSecs -startSecs) + ' seconds')
$omxplayer trial_1.h264
```

GPIO: a Bigger Challenge

Write a program that lights an LED and then waits for a button press, timing the interval between LED and button press. Now you have a reaction time task.

Which GPIO methods would be best to use?

You can make it fancier with two buttons, making sure that the first button stays pressed until the LED is lit – so no cheating.

You can add a random element to time between trials with random.random () which returns a number between 0 and 1.

Care and Feeding of a Raspberry Pi

Frontiers in Neurophotonics Summer School 2018
Jamie Boyd
Dongsheng Xiao
Tim Murphy

Setting up an SD Card

- Some cards come with Rasbian (or other) OS installed
- Otherwise, you can install your own
- Start at https://www.raspberrypi.org/downloads/
  - Download a disk image
  - Copy the disk image to the SD card
  - Etcher is a graphical SD card writing tool that works on Mac OS, Linux and Windows https://etcher.io/
- Raspbian versions named after Toy Story characters
  - wheezy->jessie->stretch
  - Latest Pi needs latest OS, but latest OS will work with oldest Pi

sudo raspi-config

- Change default user password for security
- Set time zone, locale, and WiFi country code.
  - All options on these menus default to British or GB until you change them.
- Where did my !@#$% pound sign go?
  - Change keyboard layout to US layout to change £ into #
- Enable camera, i2c, spi, remote ssh login as needed
  - To use the serial port for hardware control, you have to disable the ability to login to the pi remotely over serial.

apt-get is for software package management

Command	What it Does
apt-get update	updates list of packages available from repositories
apt-get upgrade	Upgrades all installed packages to latest versions
apt-get dist-upgrade	Also updates dependent packages
apt-get install pkg	downloads and installs a software package pkg
apt-get remove --purge pkg	removes the package pkg wolfram-engine, libreoffice
apt-get -y autoremove	removes packages that are no longer required
apt-get install synaptic	installs a GUI package manager front-end

All if these need sudo

Sudo: super user-level security privileges

- sudo and gksudo allow users to run commands or launch programs with super user-level security privileges (access /dev/mem for GPIO)
- gksudo also sets HOME=/root, and copies .Xauthority to a tmp directory. This prevents files in your home directory becoming owned by root. May be better when launching applications with windowed (GUI) interfaces

```
$sudo python3 myprogram.py
```

runs myprogram.py with python interpreter permissions elevated to use GPIO library for hardware access

```
$gksudo idle3 &
```

launches idle for Python 3, whatever program you run has elevated permissions. The & lets you use the terminal window while idle runs

Sharing Data with PCs

- Why does this USB drive not work on my pi?
  - Linux native file system format is ext3 or ext4
  - Windows native file system is NTFS
  - Macintosh native file system is APFS, which replaced HFS+
- How can I make them all share a drive?
  - For small thumb drives, format the drive as FAT32 (4 GiB file limit)
  - Install NTFS support for Raspbian
    - **`$sudo apt-get install ntfs-3g`**
  - Consider exFAT, since Windows and Mac support it natively
    - **`$sudo apt-get install fuse-exfat`**
    - https://en.wikipedia.org/wiki/Comparison_of_file_systems

Controlling the Pi over a Network

Login remotely with ssh

```
$ssh pi@142.103.107.xxx
pi@142.103.107.xxx's password:

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Sun May  6 04:17:06 2018
pi@raspberrypi:~ $
```

Copying/Sending Data over a Network

- scp copies files over a secure, encrypted network connection
  - **From a remote computer to the local computer**
  - `$scp pi@192.168.0.102:RemotePath/RemoteFileName LocalPath`
  - **From the local computer to a remote computer**
  - `$scp LocalPath/LocalFileName pi@192.168.0.102:RemotePath`
  - **Recursively for all files in a directory (can also use wild cards *)**
  - `$scp -r pi@192.168.0.102:RemotePath LocalPath`
- **For GUI version for Windows or Mac:**
  - FileZilla, Cyberduck, WinSCP

The Pi and Keeping Time

- **Most computers have a real time clock with a battery that runs the clock when power is switched off**
- **The Pi must reset its clock when it boots up to get correct time**
  - It uses a network timeserver to get correct date and time, resynching occasionally
  - No network, and date/time will be same as when Pi was last shut down
- **You can set current date and time with date command**

  $sudo date -s '2018-06-13 15:30:00'

  - **The Pi will then keep fairly accurate time until it is shut off again**
- **You can add an i2c real time clock module to a raspberry pi:**
- **https://learn.adafruit.com/adding-a-real-time-clock-to-raspberry-pi**

Use git/github for Sharing Code

•git is a version control system that tracks changes in files and allows editing by multiple users

•Github hosts open source git repositories for free and provides a web interface for git. This presentation and the code referred to in it can be copied to your computer like this:

```
$git clone https://github.com/jamieboyd/neurophoto2018
```

•This downloads the documents, plus some hashed git database stuff.

•You can update to the latest version from github with the git pull command executed in your downloaded repository directory:

```
$cd neurophoto2018
$git pull
```

It is also easy to setup a git repository on your own servers