Introduction to the Raspberry Pi

MakerspaceCT

The Raspberry Pi Foundation

Founded in 2009, the Raspberry Pi Foundation was created in response to a decline of computer science students in UK schools, with the goal of fostering the computer science programming in education.

The Raspberry Pi Foundation has developed a series of low-cost computers using open standards, intended to be an easy and affordable for school children to get exposed to computers and programming.

What is the Raspberry Pi?

- “Raspberry Pi” describes a line of low-cost ($10-$35USD) computers produced by the Raspberry Pi Foundation,
- Due to its low cost, large community, and use of open source software, Raspberry Pis have exploded in popularity since the release of the Raspberry Pi 1 Model B in March, 2012 (yes, the Model B came before the Model A).
- The most current high end versions (the Raspberry PI 3 Model B+ and Raspberry PI 4 Model B) have 64 bit CPUs, dedicated GPU’s, Ethernet and WiFi connectivity and at least 1GB of memory which is comparable with Personal Computers from about 10 years ago.
- As of February 2019, more than 25 million Raspberry Pi units have been sold worldwide!

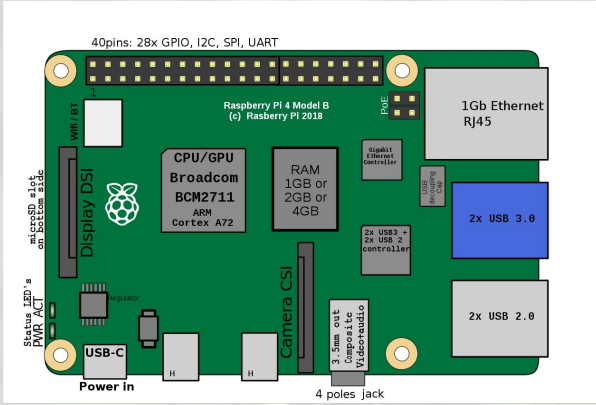
Raspberry Pi vs. Arduino

Adapted from MakeZine: (https://makezine.com/2015/12/04/admittedly-simplistic-guide-raspberry-pi-vs-arduino/)

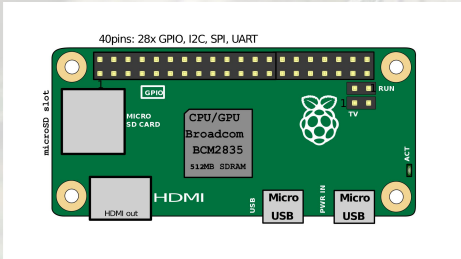
- An Arduino is a **microcontroller** motherboard. A microcontroller is a simple computer that can run **one program** at a time, over and over again. Arduino compatible microcontrollers are available from many manufacturers and in many formats. Suitable for Real Time activities because the processor is committed to that one task.
- A Raspberry Pi is a **full general-purpose computer** which can run **multiple programs** at one time. It uses a full operating system (usually Linux based) and can connect to a monitor, mouse, keyboard and network. Unlike most “personal computers” it also has an easy to access expansion bus (General Purpose Input and Output - GPIO) which makes it easier to interface to electronics projects. Due to multitasking, it may not be suitable for Real Time activities.

Which Pi ?

- Raspberry Pi 3 Model B+ and Raspberry Pi 4
  - Pros:
    - Faster Processor and more RAM
    - HDMI, Ethernet and USB connections
  - Cons:
    - Higher Power Consumption
    - More Expensive than RasPi Zero
- Zero-series (small form factor - Raspberry Pi Zero, Zero W, Zero WH
  - Pros:
    - $10 or less,
    - low power consumption, tiny form factor, great for IoT
  - Cons:
    - Too slow for desktop use, requires adaptors for HDMI/USB



By Jstrom99 - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=83463602

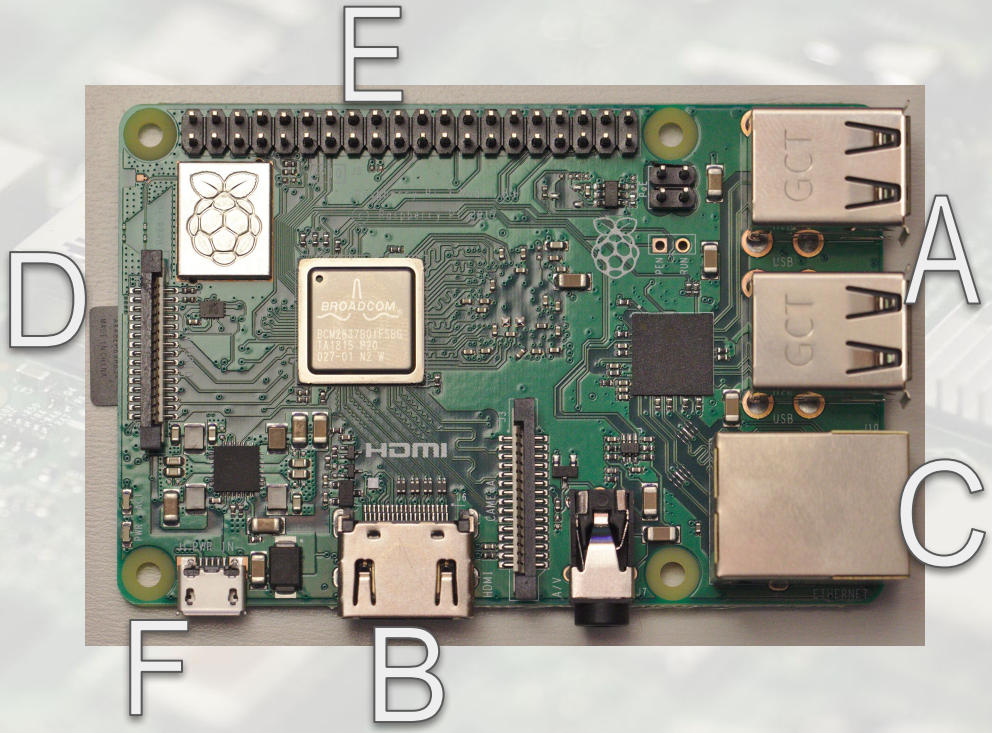


Efa [CC BY-SA 3.0 (https://creativecommons.org/licenses/by-sa/3.0)]

Raspberry Pi Setup - Connections

- Plug In:
  A. Keyboard and Mouse
  B. Video
  C. Network Cable (or use WiFi)
  D. MicroSD Card (Storage)
  E. 40 Pin GPIO Ribbon Cable
- When Ready To Start plug in Power (USB)

Note: There is no On/Off switch



File:Raspberry Pi 3 B+.jpg. (2019, July 5). *Wikimedia Commons, the free media repository*. Retrieved 05:33, November 28, 2019 from https://commons.wikimedia.org/w/index.php?title=File:Raspberry_Pi_3_B%2B.jpg&oldid=357124249.

Raspberry Pi Operating Systems

- The Raspberry Pi requires a MicroSD or SD card for the operating system and standard internal file storage. An SD card must be purchased or created in order to boot up and run the Raspberry Pi.
- There are two ways to create a bootable SD card:
  - NOOBS and NOOBS Lite
    - An easy to use operating system installer/recovery tool, sometimes provided on microSD cards sold with Raspberry Pi's. Downloadable at https://www.raspberrypi.org/downloads/noobs/
    - Easier to install: simply copy the NOOBS files to the root of a FAT32 formatted microSD card, insert card, boot and select the OS you want.
    - Standard NOOBS (about 2.3GB) includes Raspbian, NOOBS Lite (about 37MB) will download OS's from the Internet during the install.
  - Build a OS 'image disk' using a personal computer (as described at https://www.raspberrypi.org/documentation/installation/installing-images/README.md)
    - Download a ISO image of the OS formatted for the Raspberry PI.
    - Use an operating specific application (such as BalenaEtcher https://www.balena.io/etcher/ on the personal computer to transfer the ISO image to the SD drive.

Raspbian and other Pi OSs

- Standard Operating Systems Distributions
  - ***Raspbian - A LINUX Debian Distribution configured with popular tools is the most popular Raspberry Pi OS - It is our recommended OS.***
  - Ubuntu MATE - Ubuntu-based Linux Distro - is probably the next most popular
- Special Purpose OS's
  - Windows10 IOT Core - Windows 10 Universal Applications (aka Modern Apps from the Windows app store') from Microsoft
  - Kali Linux - Digital Forensics & Penetration Testing specific distribution
  - OpenWRT - Lightweight Linux distribution for Networking/Routers (Note: Raspberry Pi communication ports are not ideal for routing)
  - And many more…

Note: Linux distributions (Distro's) typically include the OS, a Graphical User Interface (GUI) and a selection of applications

Installing The Operating System

Insert the MicroSD card in the slot on the bottom of the Raspberry pie (label facing down, contacts up toward the board) and plug in the power to turn the system on.

- Follow the prompts to select the Country (United States), Language (American English) and time zone (New York) and check off ‘Use US Keyboard (just in case…)
- For the purpose of the class, leave the default username of “**pi**” and password of “**raspberry**”
- If there is a black section of the screen around the desktop, check off the box.
- Connect to the WiFi network if needed,
- SKIP, DO NOT RUN THE UPDATE routines at this time. (It takes too long for the class to wait.)
- Reboot if instructed

The Background of GNU/Linux, in a nutshell

The Linux kernel (*kernel being the lowest-level component of the operating system*) was originally released in 1991 by Linus Torvalds as a Unix-compatible system that would run on a 386 PC.

The GNU Project formed in the early ‘80s in an attempt to create a free alternative to UNIX, a then popular closed-source operating system from Bell Labs/AT&T. GNU is a recursive acronym that stands for GNU’s Not Unix.

The Linux kernel is normally paired with the *GNU* Project’s set of tools and programs that together form a complete operating system.

Raspbian Basics

Desktop Environment - A user friendly GUI shell

- The Raspberry Icon in the upper left is the main menu
- The Task Bar is on the top, default icons include:
  - The globe icon launches the Chromium Internet Browser (The Open Source root of Google's Chrome browser.)
  - Folders opens the File Manager including 'Find Files' under the Tools menu
  - Black box with prompt opens the 'Terminal' aka Linux Command Line

Connect to a Network

- Direct Ethernet connections usually start by themselves, otherwise use raspi-config from the command line or system menu
- WiFi access easily configured by clicking the radio icon on the upper right corner

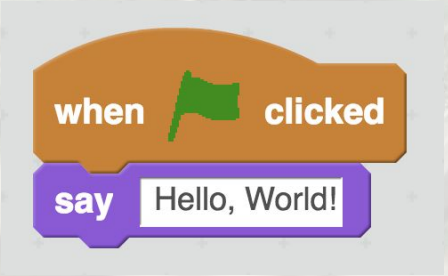
Raspbian Basics Continued

The Terminal - access to the Linux Command Prompt (aka BASH shell)

- Linux is a command oriented OS, the GUI was added on to make it more “user friendly” but the GUI in not required for OS functions.
- Most commands are in lowercase - UPPERcase and lowercase matter!
- Some commands need extra permissions (sudo - Super User Do - pronounced Sue-Doh).
- The interactive command lines keeps an editable history - use the up and down arrow keys to go up and down in the list, left and right keys etc. to edit the line and enter to accept the line (and avoid typing!)
- Commands include:
  - Running programs (**sudo raspi-config, python3, minecraft-pi, nano, zip**)
  - Installing and updating software (**sudo apt-get update, sudo apt-get install python3**)
  - Disk and file commands (**dir, ls, free -h, mkdir, chdir, pwd,** etc.)
  - Documentation (**man i2cdetect, man man , man gpio, <command> --help**)

Writing the traditional simple first program (3 Ways)

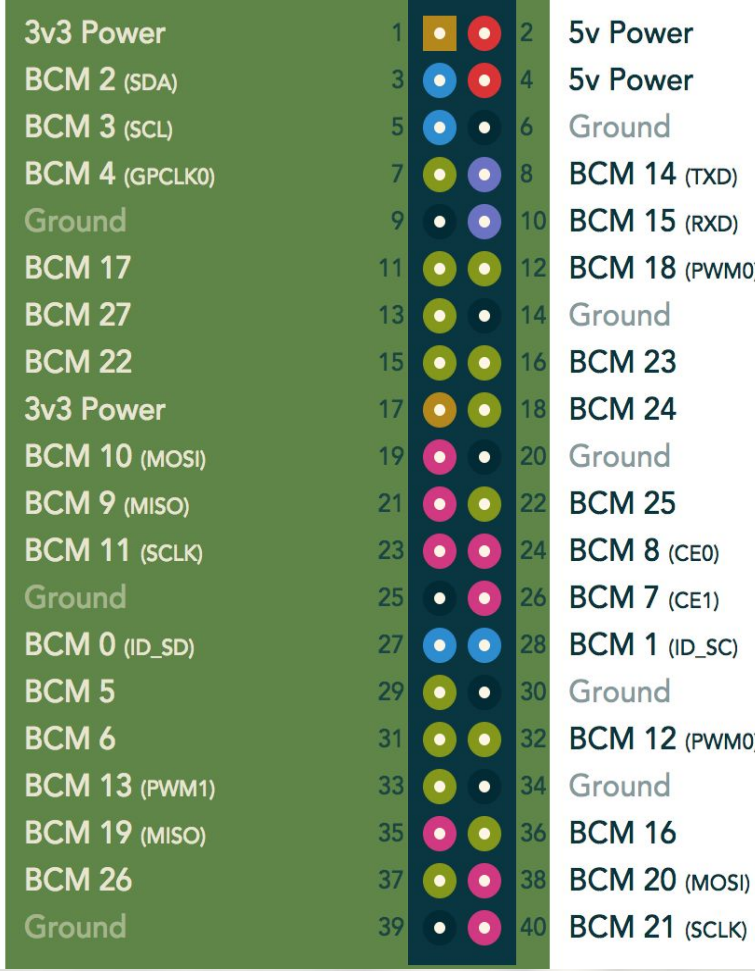
- Bash -> Linux Command Scripts - automating commands and running programs
  - The BASH Shell can run “shell scripts” - text file “programs” that run a series of commands, one line at a time
    - Open Terminal/Command session
    - Type “**echo Hello World**”
- Scratch2 -> Block-based Drag and Drop programming
  - Run application **Scratch2** (from Menu or Terminal/Command line)
  - Enter blocks (**/Events/When Flag Clicked and /Looks/Say text**)
  - Click on the green flag



- Python -> Established cross-platform language with support for more complex tasks and functions.
  - Run application **python3** (from Menu or Terminal/command line)
  - Type the following at the prompt: **print ("Hello World")** then press Return/Enter key
  - **exit()**

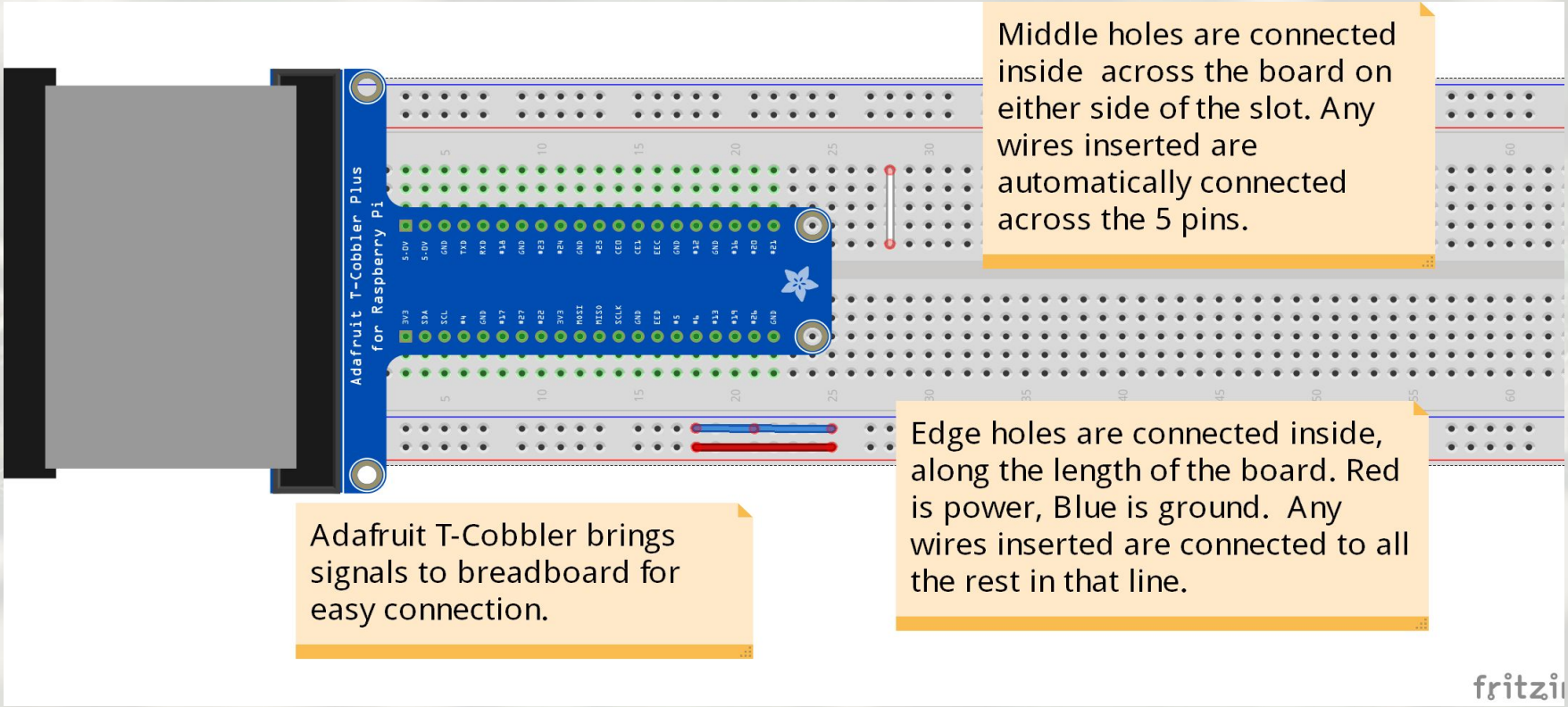
Stuff you can't do on your laptop: GPIO (General Purpose Input-Output)

- 40-pin connector that allows for straightforward interfacing to 5v or 3.3v electronics hardware - under the control of the Raspberry Pi software!
- 'Software Interfaces' are needed to talk to these ports, including:
  - WiringPi, for interaction from the command line and BASH scripts: http://wiringpi.com/the-gpio-utility/
- An alternative chart is available at: https://learn.adafruit.com/assets/28879



- Image: https://pinout.xyz

Breadboard Basics and a Helper

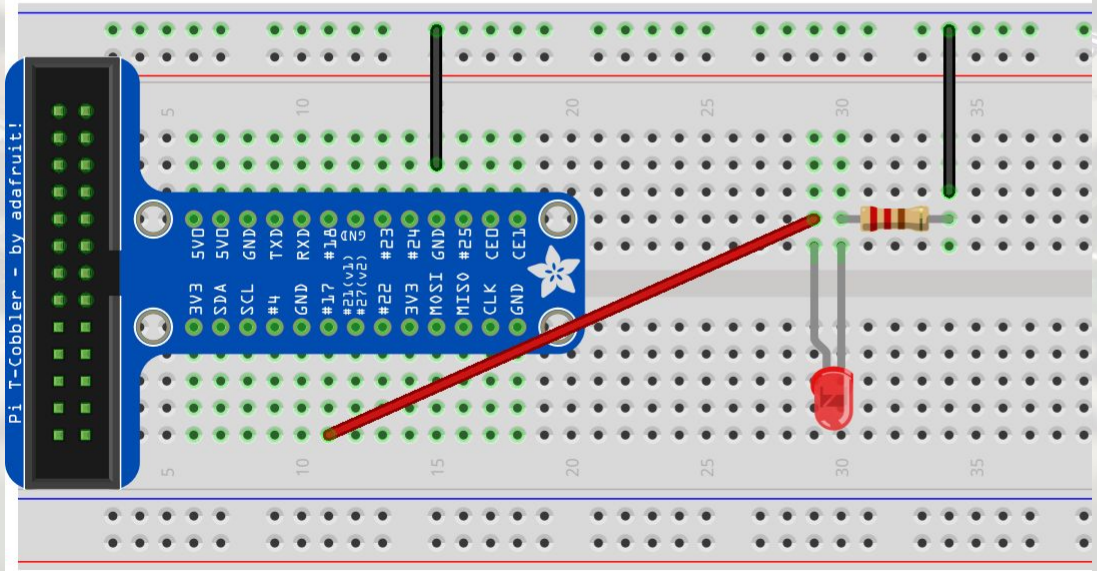


Fritzing is an amazing tool for documenting breadboard circuits and converting them to circuit boards. https://fritzing.org

Breadboard connections for controlling a LED

Wire color doesn't affect function, pick ones that work for you.

Part placement isn't critical, this is just one way to position parts and route wires.



Connections for 'Blinky" LED control:

- Ribbon Cable from RasPi to T-Cobbler provides easy access to all signals.
- Any ground (GND pin) connected to one end of an about 330 ohm resistor.
- The other end of the resistor connects to the straight (short) side of LED ***.
- The bent (long) side of LED connects to Pin 11 (BCM 17) from the RasPi (note: 17 is right next to 27!)

Blink a LED from the RasPi - Three Ways

- Bash -> Native Linux Script, great for simple and quick projects
- Scratch (2) -> Block-based programming (syntax-free)
- Python -> Established cross-platform language with support for more complex tasks and functions.

I bet you can you think of another way to blink the LED on and off. We could simply connect the LED to pin 1 (3 volts) and it would light. Disconnect and it goes off. We could even add a button in that circuit and the button would control the LED. Sometimes computers make easy things harder....

bash

- Run `**gpio readall**` to view status of GPIO pins.
  - If it's not available install it (sudo apt install wiringpi)
  - If it gets an error saying "Unable to determine board type" then install update from: http://wiringpi.com/wiringpi-updated-to-2-52-for-the-raspberry-pi-4b/
- Run `**man gpio**` to view documentation for gpio.
  - (-g option says to use BCM GPIO numbers)
- Run `**gpio -g mode 17 out**` to set pin 17 to output.
- Run `**gpio -g write 17 1**` to set the value of pin 17 to "1" (HIGH) On
- Run `**gpio -g write 17 0**` to set the value of pin 17 to "0" (LOW) Off

Create bash ‘shell script’ (.sh extension)

- Open Terminal
- Run `nano lightonoff.sh’ to open the nano editor and create a file
- Type in the following lines:

```
#!/bin/bash
gpio -g mode 17 out
echo “light On”
gpio -g write 17 1
sleep 1s
echo “light Off”
gpio -g write 17 0
```

- Use the **Ctrl-o** key combination to write the file out
- **Ctrl-x** to exit
- Run ‘**chmod +x lightonoff.sh**’ to allow executing the file
- Run ‘**bash lightonoff.sh**’ to run the script

Create looping BASH script

- Run `nano lightonoff.sh' and change the file:

```
#!/bin/bash
gpio -g mode 17 out
gpio -g write 17 0
sleep 0.5s
for (( i = 1; i < 6; i++ ))
do
  echo "light Off" $i
  gpio -g write 17 1
  sleep 1s
  echo "light Off"
  gpio -g write 17 0
  sleep 1s
done
```

- Use the **Ctrl-o** key combination to write the file out, Changing the name to '**looponoff.sh**'
- Run '**chmod +x looponoff.sh**' to allow executing the file
- Run '**bash looponoff.sh**' to run the script

Scratch (version 2)

Open Scratch2 (**Menu->Programming->Scratch2**)

A. Add GPIO extension library
   1. **/Scripts/more blocks/"add an extension"**
   2. Select '**Pi GPIO**' and hit Enter

B. Create Scratch program
   1. Grab and move **/Scripts/Events/"when space key pressed"** to the work area
   2. Grab and move **/Scripts/More Blocks/"set gpio"** and set under "**event**"
      a. Change number to **17** and keep **Output High**
   3. Grab and move **/Scripts/Control/"wait 1 secs"** and set under "**set gpio**"
   4. Grab and move **/Scripts/More Blocks/Set gpio** and set under "**wait**"
      a. Change number to **17** and set "**output low**"
   5. Grab and move **/Scripts/Control/"wait 1 secs"** and set under **"set gpio"**

C. Run Program by pressing the **<space bar>**

Scratch with a loop

A. Update Scratch program

1. Grab and disconnect all the set and wait blocks from the event
2. Grab new **/Scripts/Control/"Repeat"** block and set it under **'When Space…"**
3. Grab and move all the set and wait blocks and set inside the **Repeat loop**

B. Run the code by pressing the **<spacebar>**

If time, explore changing repeat and wait times, adding cat actions, etc.

Python3 - Using the Thonny Python IDE

- Open Thonny (**/Menu-/Programming/Thonny Python IDE** or **Thonny** on command line)
- Click on **New** to open a new working area
- Enter the following code into the editor (copy and paste)

```
import RPi.GPIO as GPIO
import time
GPIO.setmode(GPIO.BCM)  #use BCM pin numbering to match T-Cobbler board.
GPIO.setup(17, GPIO.OUT)
GPIO.output(17,True)
print("On")
time.sleep(5)
GPIO.output(17,False)
print("Off")
GPIO.setup(17, GPIO.IN)  #resets pin to avoid error messages
```

- Click on the **Run** icon to run the program (give it a name on the first run)

Note: In Python, “#” is the delimiter for a comment - whatever rather than code that will be interpreted.

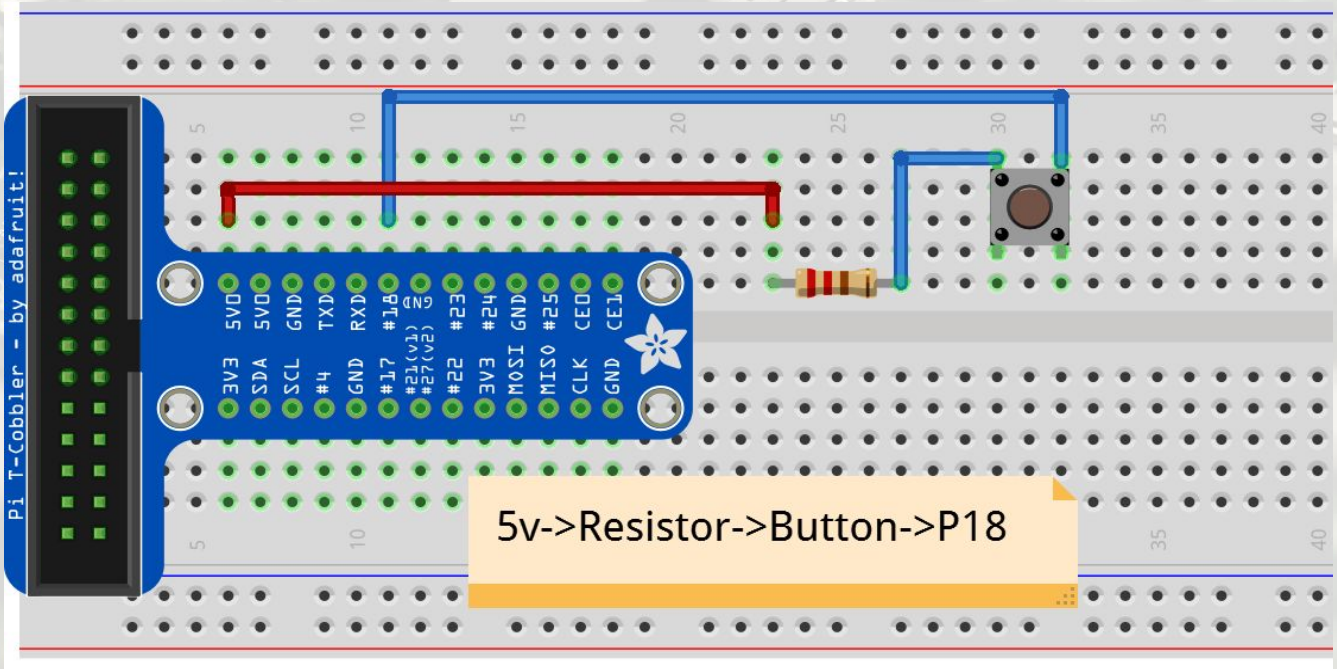
Python loop and function example

```
#! /user/bin/python3
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM) #use BCM pin numbering to match T-Cobbler board.
GPIO.setup(17, GPIO.OUT) # use GPIO 17 to connect the LED

# blink count times, delay starts at pauseStart seconds, wait ramp second less each
loop
def BlinkRampDown(count,pauseStart, ramp):
    print("Loops:",count, "Delay at start:", pauseStart, "Ramp:", ramp)
    pause = pauseStart

    for i in range(0,count):
        GPIO.output(17,True)
        time.sleep(pause)
        GPIO.output(17,False)
        time.sleep(pause)
        pause = pause - ramp
        if (pause <= 0):
            pause = 0

# Experiment with various calls
BlinkRampDown(10,1,0.1)
BlinkRampDown(20,0.25,0.015)
BlinkRampDown(10,0.25,-0.1)

GPIO.setup(17, GPIO.IN) #resets pin to avoid error messages
```

Remember:
Indentation is important!

Adding/Wiring a Button

- Leave your LED set up, we're going to keep using it
- Add a Button and connect one side to Pin 12 (GPIO18)
- Add a Resistor between the other leg of the button and Pin 2 (5 volts)



Scratch Button Reader

1. Open Scratch2

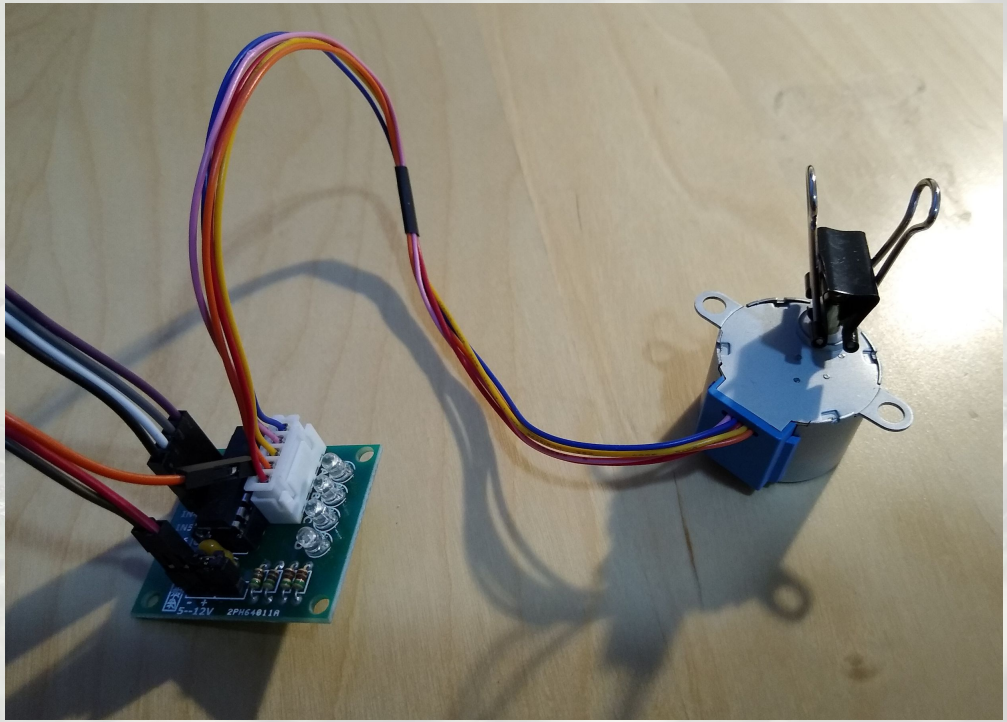
   (**Menu->Programming->Scratch2**)
2. B. Create Scratch program
   a. **Event/when flag clicked**
   b. **Control/loop forever** under when
   c. Put **control if-else** in **forever** loop
   d. **If** condition **<More/GPIO 18> is high** then
   e. Inside then:
      i. **Set gpio 17 to output.high**
      ii. **Move 10 steps**
      iii. **Next costume**
   f. Inside else
      i. **Set gpio 17 to output.low**
      ii. **Go to x:0 y:0**
3. Run the program by clicking on the flag
4. Press the button to see it work

Python Button Reader

```
import RPi.GPIO as GPIO
import time
GPIO.setmode(GPIO.BCM)  #use BCM pin numbering to match T-Cobbler board.

# Set GPIO18 as an input pin and set initial value to be pulled low (off)
GPIO.setup(18, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)

GPIO.setup(17, GPIO.OUT) # our LED is still connected to GPIO 17
GPIO.output(17,False)    # Turn the LEd off if it's on

while True:              # Run forever
   if GPIO.input(18) == GPIO.HIGH:
      print("Button was pushed!")
      GPIO.output(17,True)
   else:
      GPIO.output(17,False)

GPIO.setup(17, GPIO.IN)  #resets pin to avoid error messages
```

Stepper Motor Setup

- Stepper Motor 28BYJ-48
- Stepper Controller Board
  - IC ULN2003APG
  - 4 clear LEDS
  - White 5 pin connector
  - IN1 to IN7
  - Power - and + (5--12V)



Plug white 5 pin cable from Motor to controller

1. Connect IN1 to GPIO 5
2. Connect IN2 to GPIO 6
3. Connect IN3 to GPIO 13
4. Connect IN4 to GPIO 19

Stepper Motor continued

1. Connect Power Cables
   a. “-” to ground on breadboard
   b. ‘+” to +5 on breadboard
2. Go to: https://github.com/gavinlyonsrepo/RpiMotorLib/blob/master/RpiMotorLib/RpiMotorLib.py and copy the file **RpiMotorLib.py** (Or other locations as it may be available during class)
3. Open Thonny, open a new file, and paste library into editor
4. Save the file as **RpiMotorLib.py** into your current python directory (the default folder)
5. Open a new file and Copy in the short test program from the next page
6. Run it
7. Change some parameters and run it again

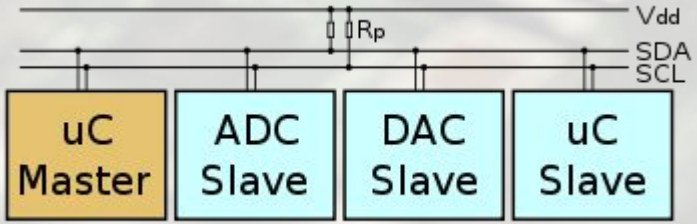
```
# Call Syntax: motor_run(gpiopins, wait, steps(512 = 360 degrees),
#     ccwise(False = clockwise), verbose, steptype(”full,"half" or ”wave”), initdelay)
```

Python Stepper Motor Test

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

GpioPins = [5, 6, 13, 19]    # GPIO pins to the controller
mymotortest = RpiMotorLib.BYJMotor("MyMotorOne", "28BYJ")

startTime = time.time()
# Call Syntax: motor_run(gpiopins, wait, steps(512 = 360 degrees),
#         ccwise(False = clockwise), verbose, steptype(”full,"half" or ”wave”),
initdelay)
mymotortest.motor_run(GpioPins , .002, 256, False, False, "full", .0005)
endTime = time.time()
print(f'Elapsed time in seconds {endTime - startTime:5.3}')

# avoid errors when running next program
GPIO.cleanup()
```

I^2C - Inter-Integrated-Circuit *(pronounce eye-squared-see)*

- I2C is a very common multi-master, multi-slave, serial protocol for two-wire interface to connect devices like microcontrollers, I/O interfaces and other similar peripherals in embedded systems. (Typically 100kbits/sec up to 1 meter)
- A Serial connection uses fewer wires than a parallel connection to connect multiple devices because only a few wires are needed to send data in a serial fashion.
- Each I2C slave device has a unique address.



monday

Updating/Installing Software Packages in Raspbian

- The `apt` package manager is the easiest way to install software.

Syntax of the command:

Prefixing a command with ‘`sudo`’ causes it to be run as the root user (superuser).

“`apt`” is the name of the package manager.

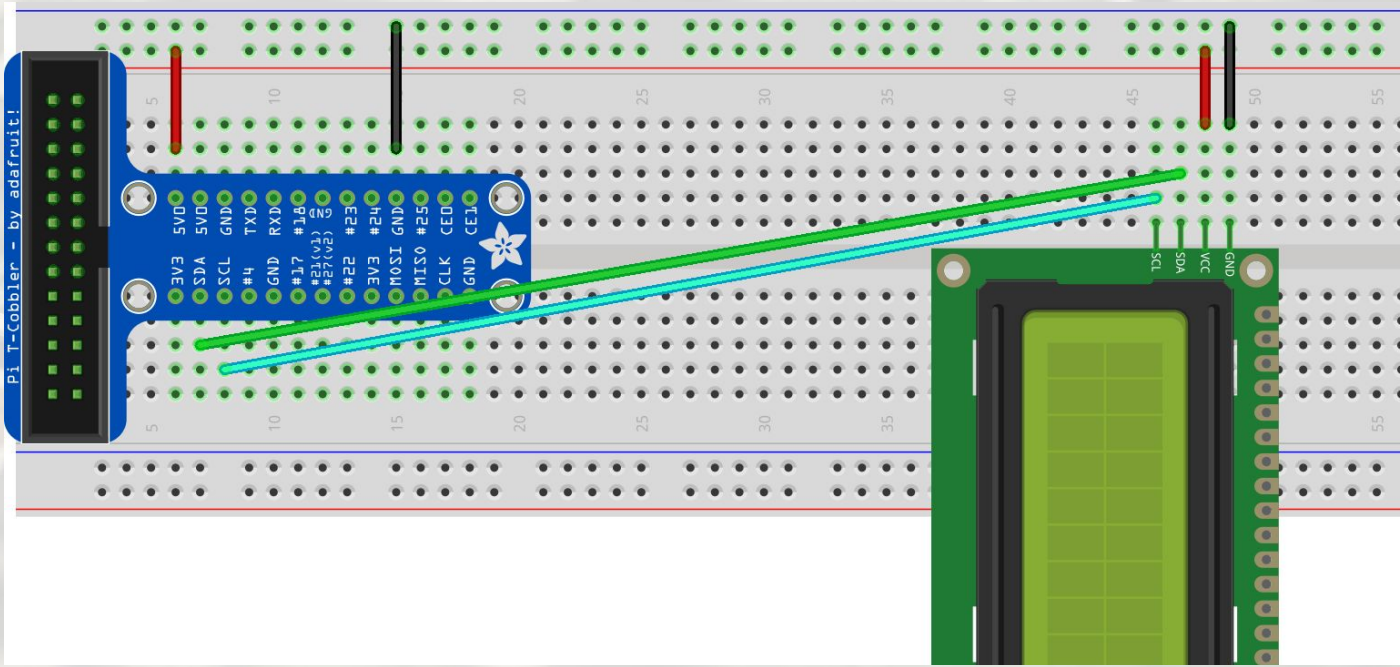
“`install`” is what you want apt to do to the package you name next

“`i2c-tools`” is the name of the Package to be installed.

```
sudo apt install i2c-tools
```

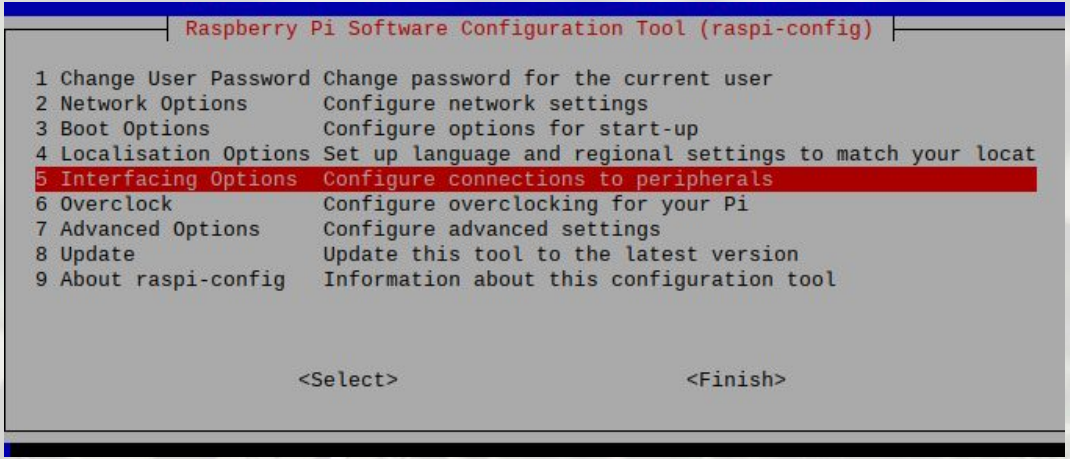
Wire the LCD module

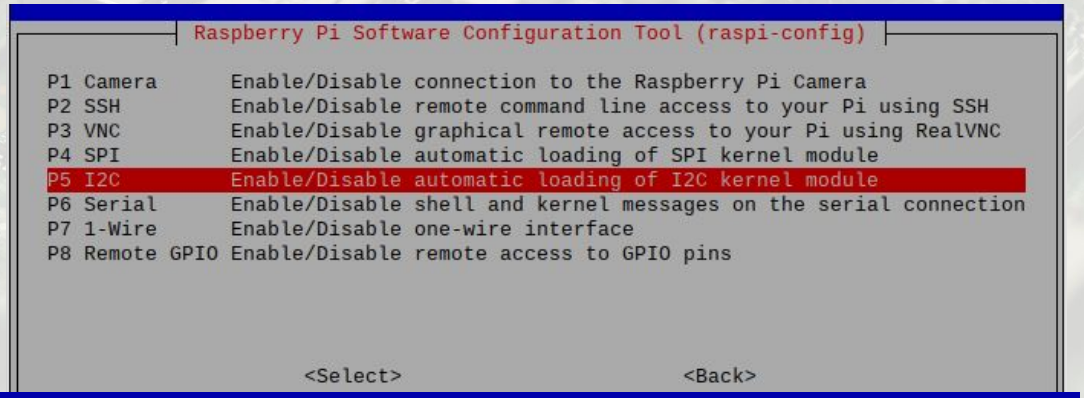
- GND goes to Ground
- VCC goes to +5 Volt
- SDA goes to SDA (Pin 3)
- SCL goes to SCL (Pin 5)

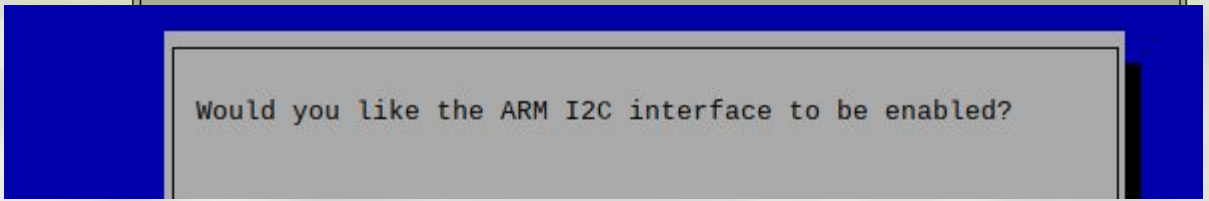


Enable I²C on the Raspberry Pi

- Run **sudo raspi-config**
- Select **#5 Interface Options**
- Select **#P5 I2C**
- Select **<Yes>**
- Select **<ok>**
- Select **<Finish>**







Ensure the SMBus and i2c software is loaded and Find the address of the i2c device

- Check that the SMBus software for Python is installed
  - **sudo apt install i2c-tools**
  - **sudo apt install python-smbus**
- Detect the address of the LCD
  - Run “**i2cdetect -y 1**”
  - The address will show on the grid as something other than “--”, write down the number.

Writing a message to a LCD from Python

- Open the LCD1602_guide.md file from https://github.com/MakerspaceCT/raspberry_pi_intro/blob/master/LCD1602_guide.md
  - Find the section labeled: "I**nstall the Python I2C Library**"
  - Copy and paste the first section of Python code into **Thonny**
  - Edit the BUS (probably **BUS = 1**) and ADDRESS values on lines 18 and 22.
  - Save the driver file as "**I2C_LCD_driver.py**". This file name must be exact (without the quotes).
- Create a python program to write to the LCD, save and run it.

```
import I2C_LCD_driver
from time import *
mylcd = I2C_LCD_driver.lcd()
mylcd.lcd_display_string("Hello World!", 1)
```

- Extra challenges:
  - Show the count of button pushes on the LCD
  - Show a list of quotes, one at a time. Push the button to change the text?
  - Push button to light up led, move characters along screen

How to host a web site on your Raspberry Pi

As documented at: https://www.raspberrypi.org/documentation/remote-access/web-server/ both Apache and NGINX are officially available for the Raspberry Pi . We will install a local NGINX server for this class as it tends to require less resources.

Note: It is unlikely that these systems will provide the capacity desired for a public web site, but if you do provide public access be sure to lock down your connection with a firewall and router rules that only allow the required access.

It is a good practice to ensure that only one web server is installed on a system, run the following to ensure Apache is not installed:

- **sudo apt-get remove apache2**

Let's follow the foundation's documented process at:https://www.raspberrypi.org/documentation/remote-access/web-server/nginx.md

MAKERSPACECT

EQUIPMENT. COMMUNITY. EDUCATION.

Questions about the class? Email Drew at drew@drewgates.com or Chuck at VenterCE@gmail.com

Thank you!