

CSCI 4907

Introduction to IoT and Edge Computing Applications

Prof. Kartik Bulusu, CS Dept.

Week 3 [02/03/2023]

- Setting up the Edge Lab
- Some more programming constructs
- Recap on RPi skeleton code
- Bash script to run RPi-python code
- Raspberry Pi programming [Blinking LEDs using Thonny]
- Setting up a Python virtual environment [venv] - Sandboxing your code
- Blinking LEDs on boot

```
git clone git@github.com:gwu-csci3907/Spring2024.git
```

```
git clone https://github.com/gwu-csci3907/Spring2024.git
```

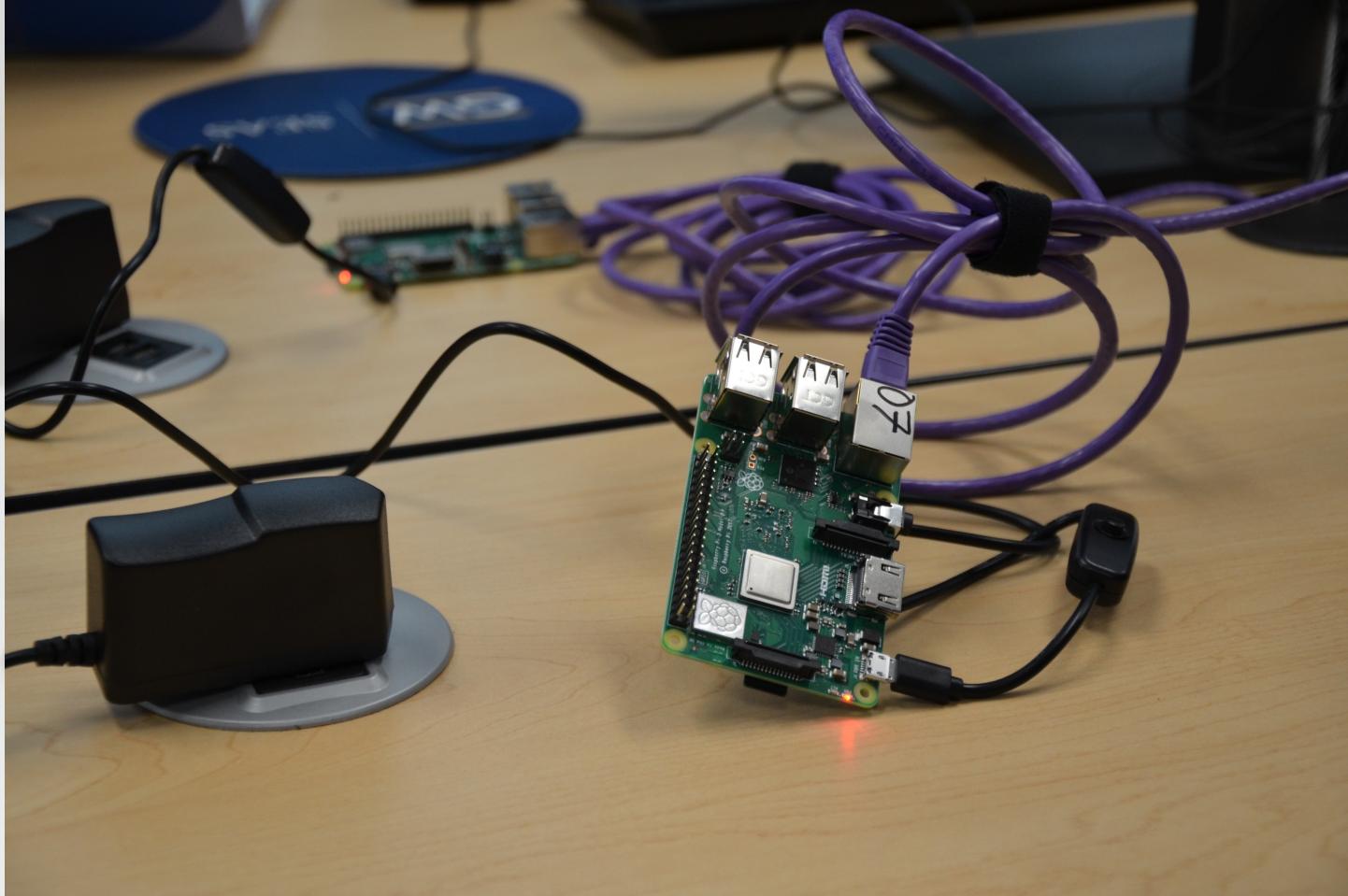


School of Engineering
& Applied Science

Set up lab the Edge-lab

STEP [1]:

Connect the RPis to each desk power outlet as shown



- Make sure there is a microSD card installed in the RPi
- Connect the RPi using the microUSB cable provided
- Connect the purple colored ethernet cables specifically for RPi connections
- LEDs on the RPi will start blinking indicating that it is booting up

STEP [2]:

Access the RPi in the Edge-lab

2.1 Open up remote desktop connection (using the VNC server)

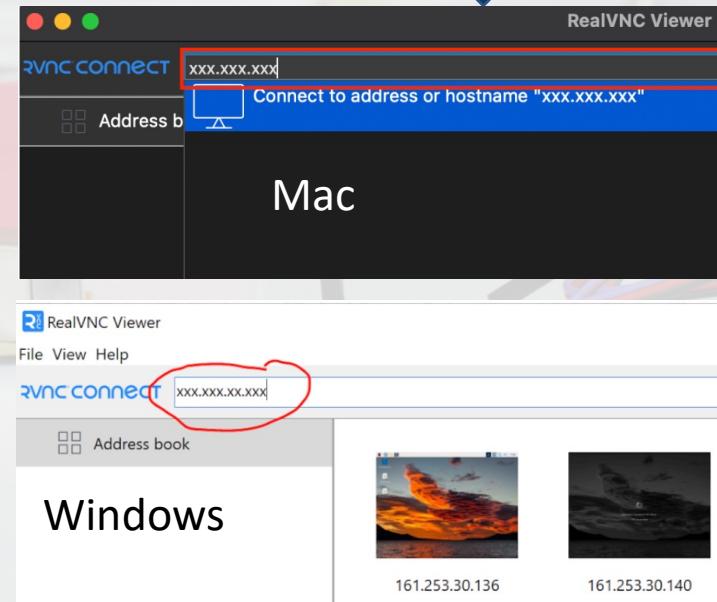
Each RPi has unique alpha-numeric name (e.g., Pi07, Pi152 etc)

- Locate the Pi-name and the IP address on the <128.164.139.xx>

OR

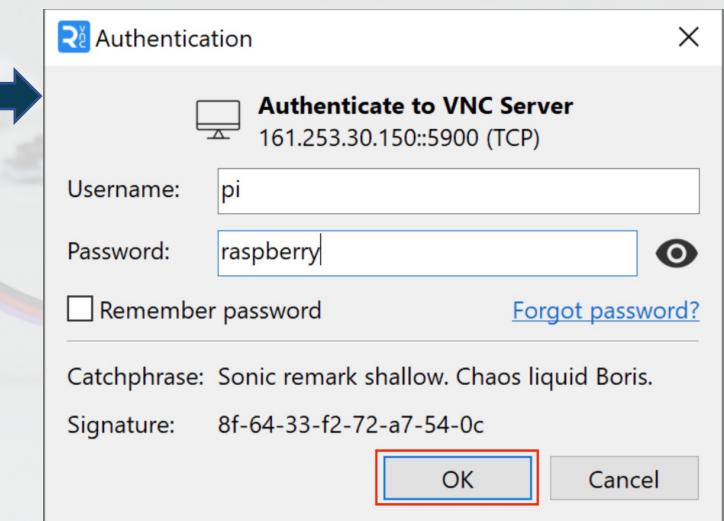
Each RPi connected using an ethernet cable directly to your laptops

- raspberrypi.local



2.2 Once you are connected you will

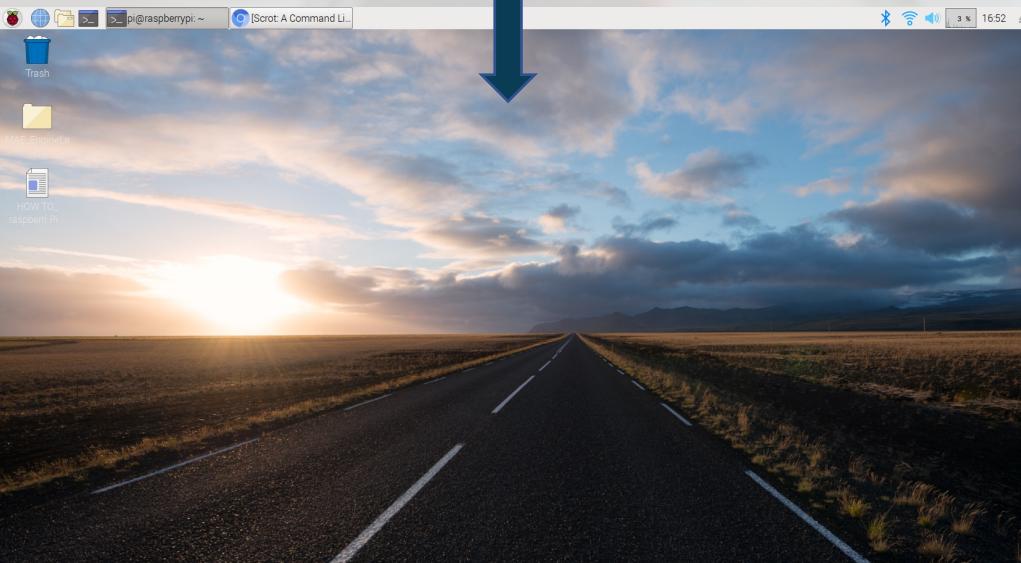
- See Authentication box below
- Type in the Username and Password



STEP [3]:

Now that you accessed the RPi...

You will see a screen like the one shown below

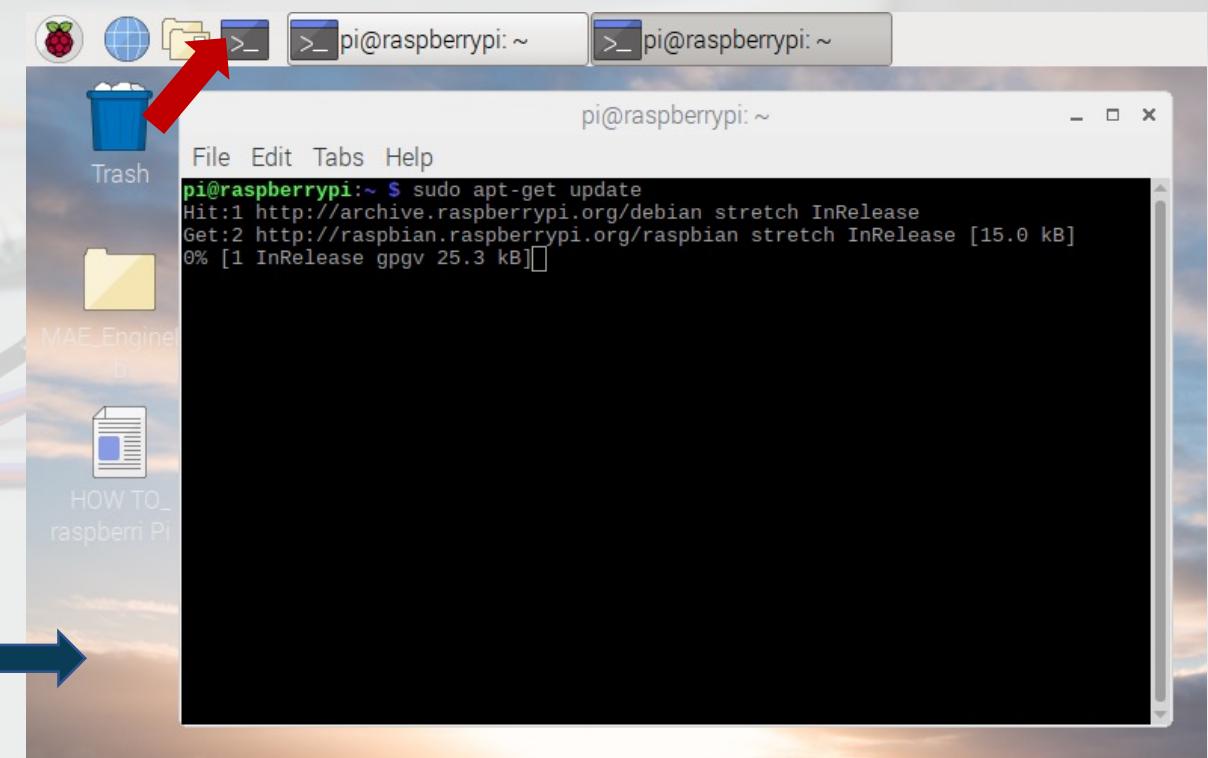


3.2 Testing is complete when you get to this step.

- Students should get the RPis to this step before the laboratory modules begin.

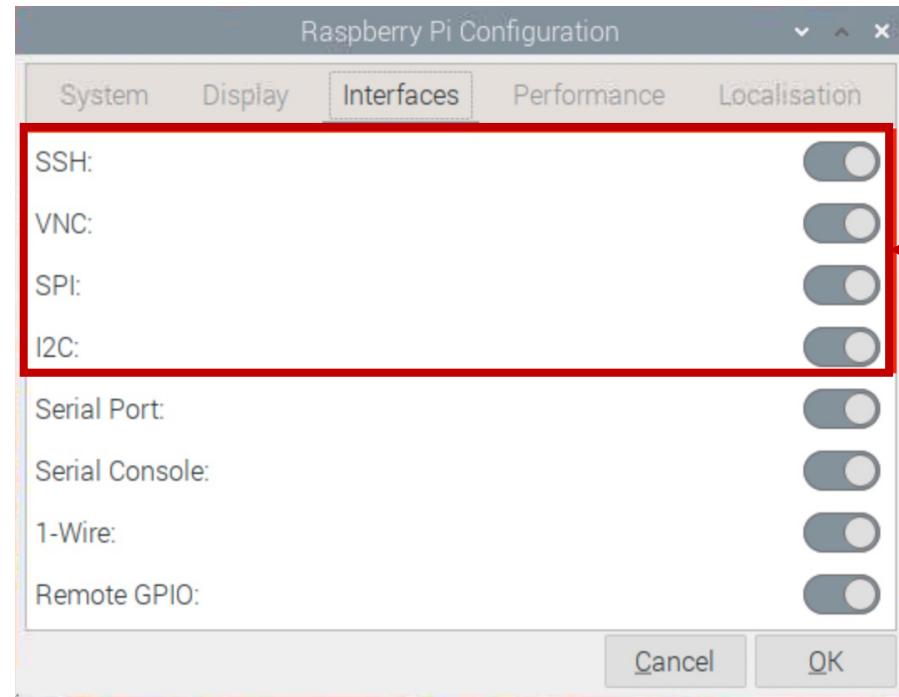
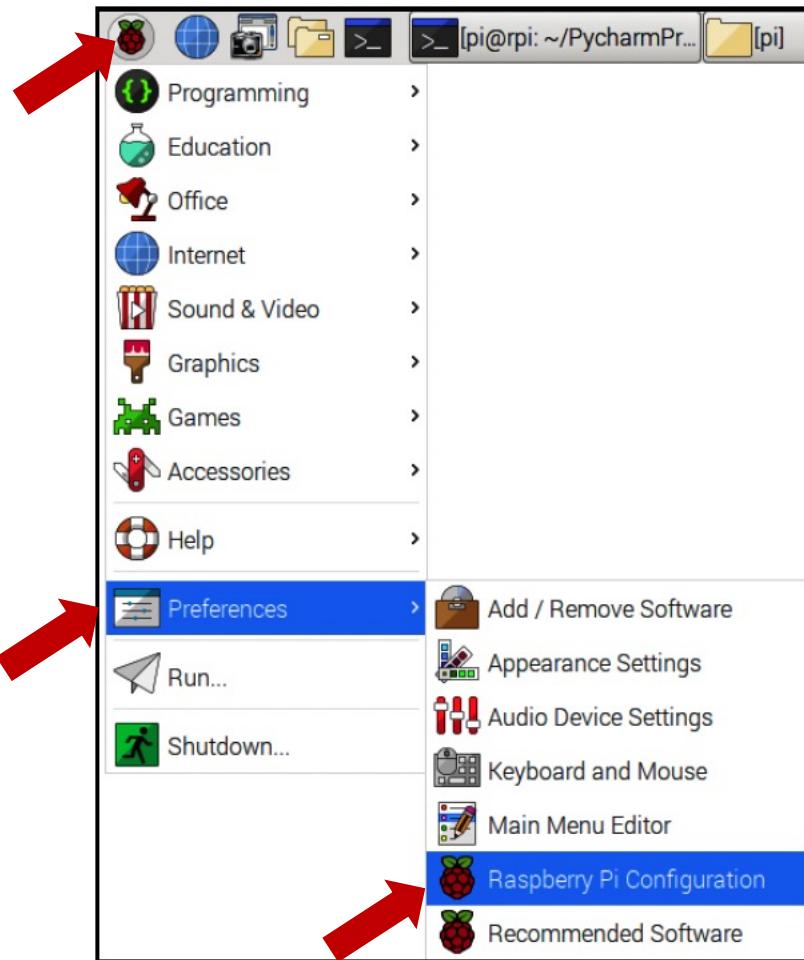
3.1 Click on terminal(shown with a red arrow below)

- At the prompt type: **sudo apt-get update**
- Wait for the updates to complete
- Then type: **sudo apt-get upgrade**
- If you get the following prompt
 - **Do you want to continue [Y/n]**
 - Type: **y**
 - And hit "Enter" on your keyboard and let the upgrades complete

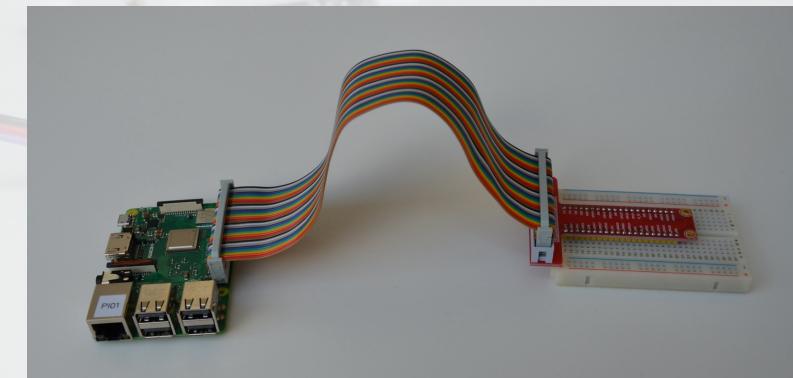
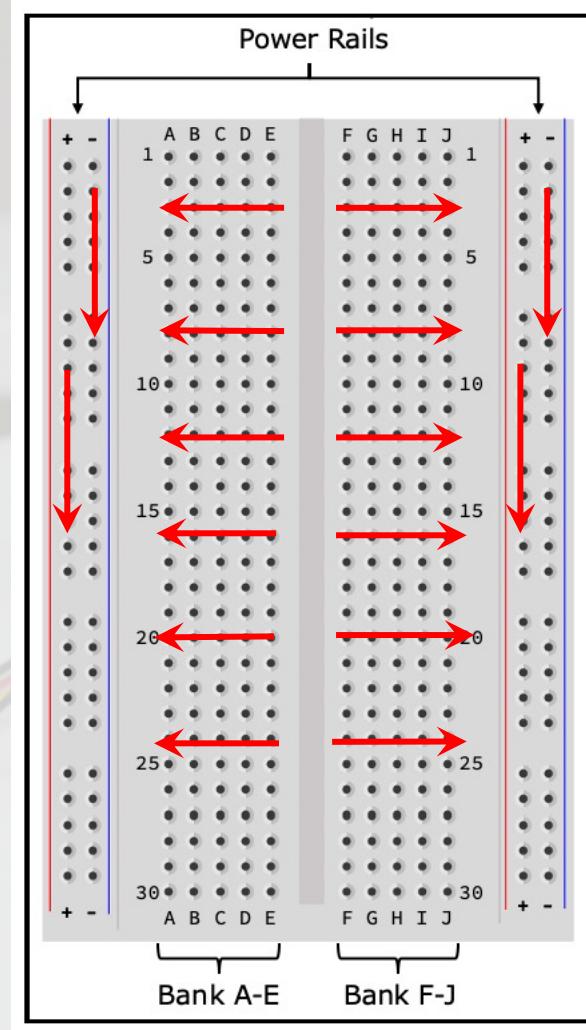


Couple of checks and balances

Make sure these are enabled



Connect the Raspberry Pi Model 3 B+ (RPi) to a bread board



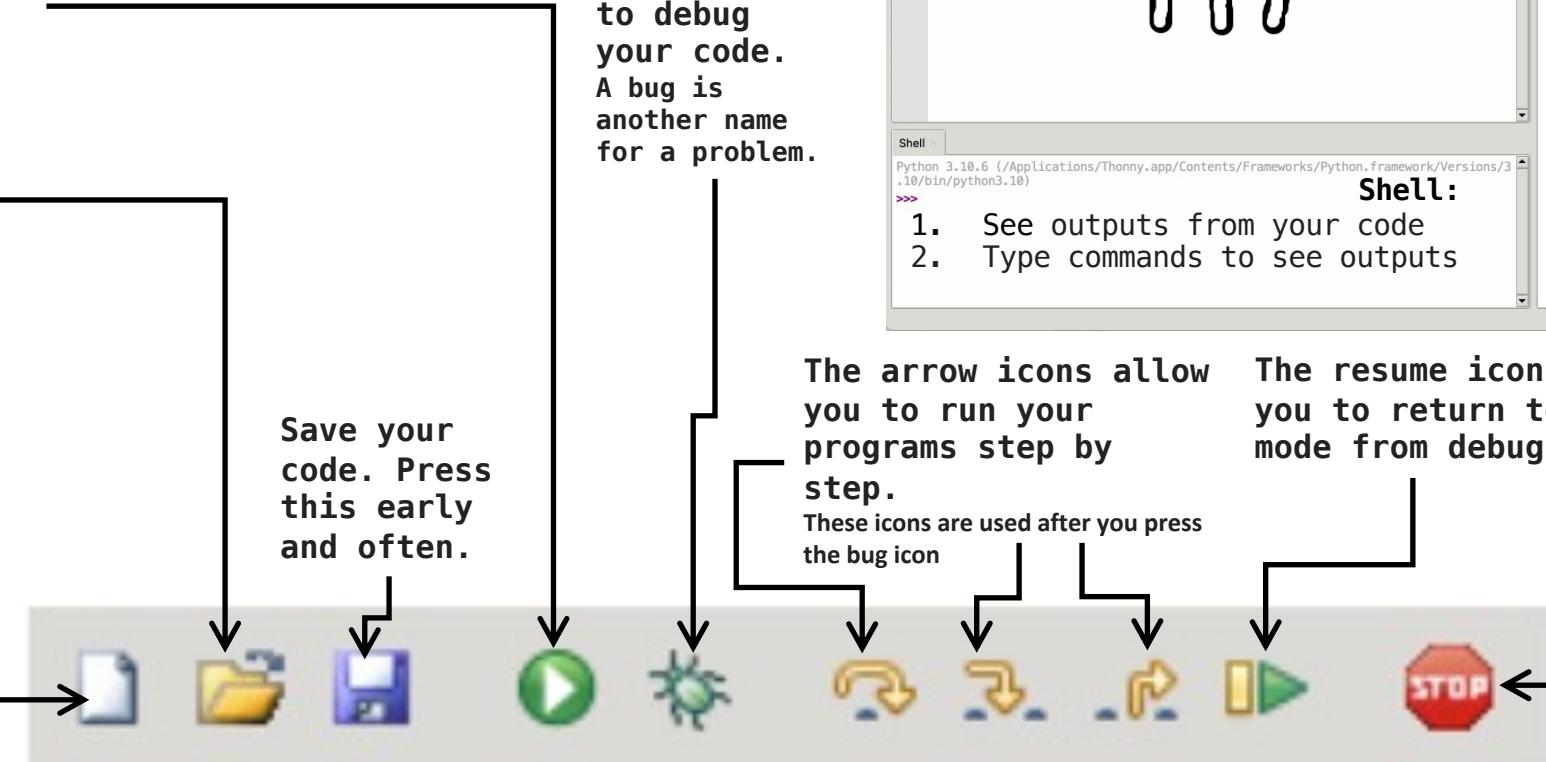
Let's get to know some programming
paradigms

Thonny integrated development environment (IDE)

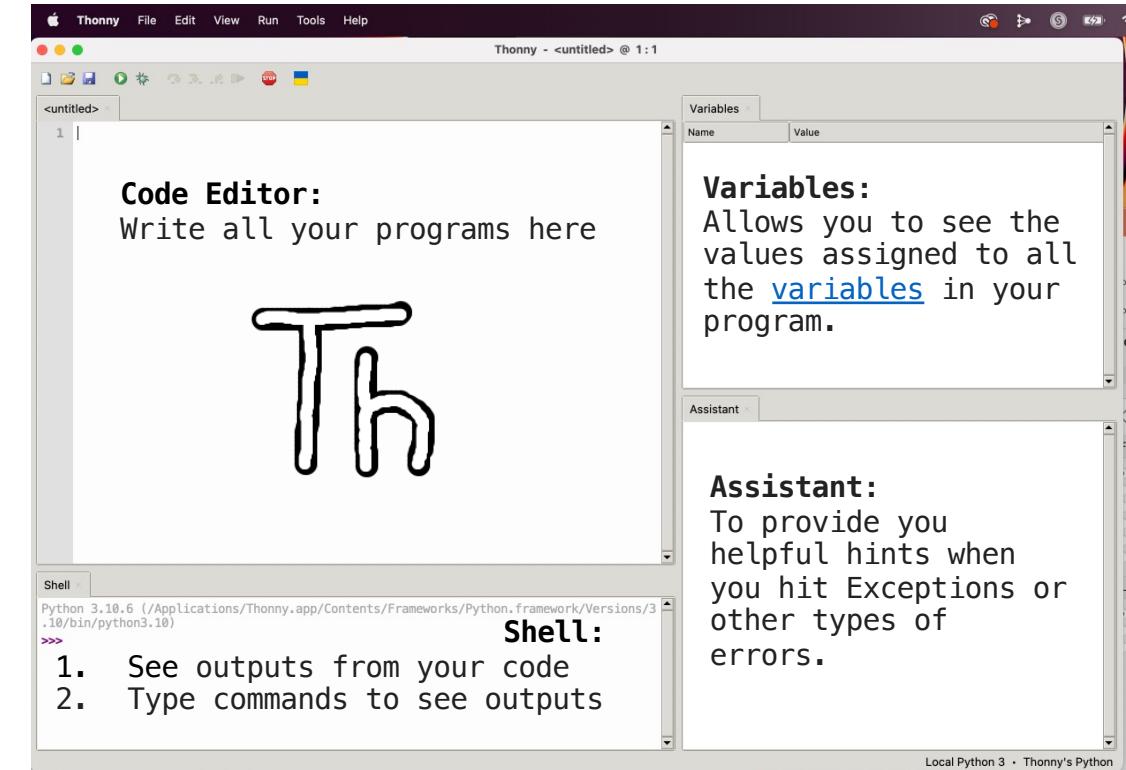
Allows you to run the code, i.e., "Do what I told you do do!".

Allows you to open a file that already exists on your computer

Create a new file



Sources:
Thonny (IDE): <https://en.wikipedia.org/wiki/Thonny>
Thonny: The Beginner-Friendly Python Editor: <https://realpython.com/python-thonny/>



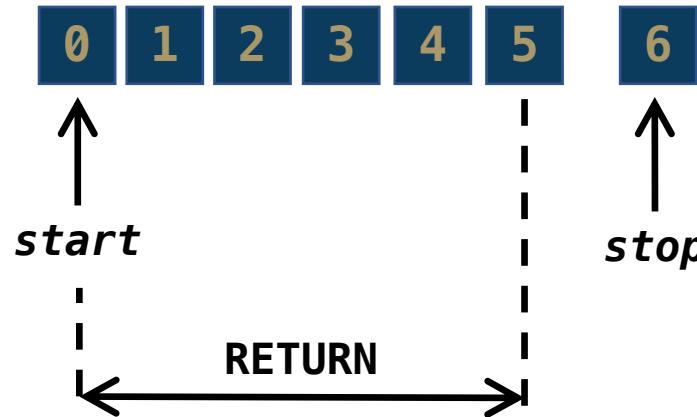
Built-in function **range()**

Python's **range()** function
returns a sequence of numbers
works only with integers

start: at the value (default = 0)
step: up or down at the increment value (default = 1)
stop: at the value but not including it

range(stop)

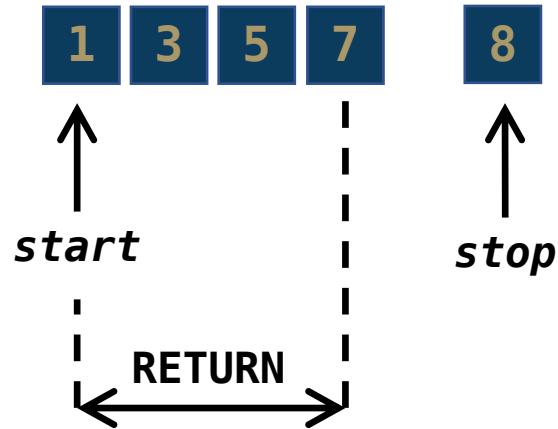
```
>>> range(6)
```



range(start, stop)

range(start, stop, step)

```
>>> range(1,8,2)
```



range()

- can be utilized in (for) loops
- to specify a range to iterate or do repetitions

Demos

Syntax and Skeleton of a user-defined function

```
def name(parameters):  
    statement  
    statement  
    ...  
    return value
```

Functions are blocks of reusable pieces of code

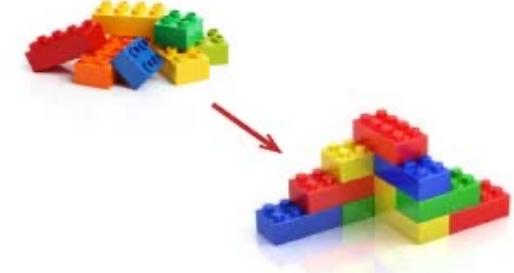


Image source:
<http://www.thescoopformummies.com/teaching-colors-to-preschoolers/lego-blocks-clip-art-uclody-clipart/>
<https://www.123rf.com/clipart-vector/lego.html?ttkey=7436ca74>

Function name: Identifier
by which it is called in the program

Function Declaration:
Starts with “def” that is not indented

Indentation: Tab or
4 spaces for each statement

statement
statement
...
return value

(Optional) Arguments:
values passed to the function

Body: Statements executed each time a function is called
(Optional) return value:
Can end function call and send data back to the main program

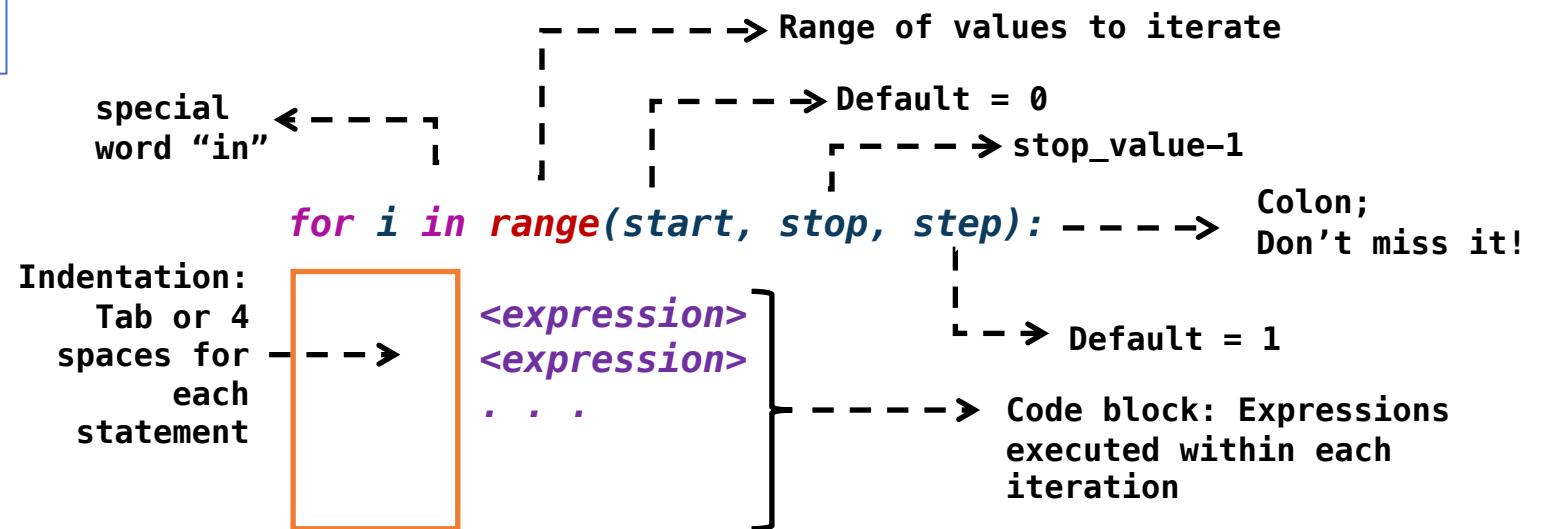
func_name()

Function call

Skeleton of the for-loop

```
for i in range(start, stop, step):  
    <expression>  
    <expression>  
    ...
```

Demos



Skeleton of
try:
except:
else:
finally:

```
try:  
    statement  
    function  
except KeyboardInterrupt:  
    statement  
    function  
else:  
    statement  
    function  
finally:  
    statement  
    function
```

The **try** block lets you test a block of code for errors.

The **except** block lets you handle the error.

The **else** block lets you execute code when there is no error.

The **finally** block lets you execute code, regardless of the result of the try- and except blocks.

Exception Clauses

```
try:  
    # Runs first  
    < code >  
except:  
    # Runs if exception occurs in try block  
    < code >  
else:  
  
finally:
```

Exception Clauses

```
try:  
    # Runs first  
    < code >  
except:  
    # Runs if exception occurs in try block  
    < code >  
else:  
    # Executes if try block *succeeds*  
    < code>  
finally:
```

Exception Clauses

```
try:  
    # Runs first  
    < code >  
except:  
    # Runs if exception occurs in try block  
    < code >  
else:  
    # Executes if try block *succeeds*  
    < code>  
finally:  
    # This code *always* executes  
    < code >
```



Skeleton of the updated Python program written for Raspberry Pi

```
import library1 as name1
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BORD)
GPIO.setup(12, GPIO.OUT)

def function_name(arguments):
    statement
    statement
    ...
    return value

if __name__ == "__main__":
    try:
        function_name1()
    except KeyboardInterrupt:
        function_name2()
```

The diagram illustrates the structure of a Python program for a Raspberry Pi. It uses curly braces and dashed arrows to map specific code snippets to their intended purpose. The annotations are as follows:

- Imports:** The first three lines of code (import statements) are grouped by a brace on the left and an arrow pointing to the right, which points to the text: "Import libraries that are relevant for interaction with the Raspberry Pi hardware such as GPIO pins, camera ports etc."
- GPIO Configuration:** The two lines of code involving GPIO (setmode and setup) are grouped by a brace on the left and an arrow pointing to the right, which points to the text: "Set up GPIO pins as data outputs or inputs for sensors and actuators"
- User-defined Functions:** The entire block from the start of the def statement to the return value is grouped by a large brace on the left and an arrow pointing to the right, which points to the text: "Create user-defined functions to modularize your code and make it easy to work." Below this, another brace groups the return value line and its corresponding arrow points to the text: "Create user-defined functions to release the GPIO pins"
- Entry Point and Exception Handling:** The final block from the if __name__ == "__main__" check to the end of the program is grouped by a brace on the left and an arrow pointing to the right, which points to the text: "Create entry point into the program and pull all functions in". Below this, another brace groups the exception handling and its arrow points to the text: "Create a keyboard-interrupt exception clause"



Let's blink some LEDs

Preliminaries:

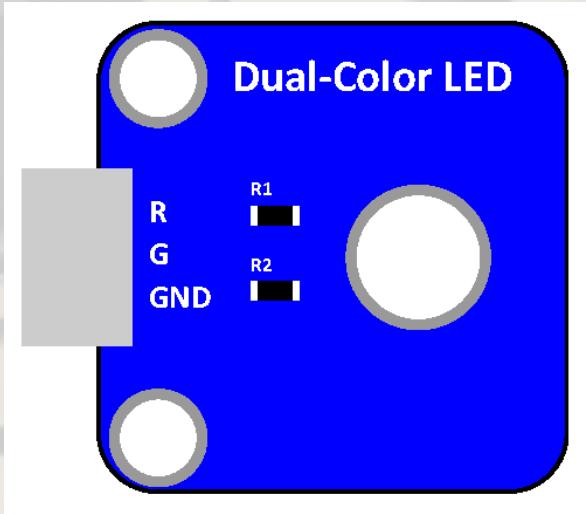
1. **git-clone the repo preferably on the desktop**

```
git clone git@github.com:gwu-csci3907/Spring2024.git
```

```
git clone https://github.com/gwu-csci3907/Spring2024.git
```

2. **I have the files on a USB in case there is a slowdown!**

Know your Light Emitting Diode (LED)



Source:

<https://www.sunfounder.com/learn/lesson-1-dual-color-led-sensor-kit-v2-0-for-b.html>

A dual-color light emitting diode (LED) is capable of emitting two different colors of light, typically red and green.

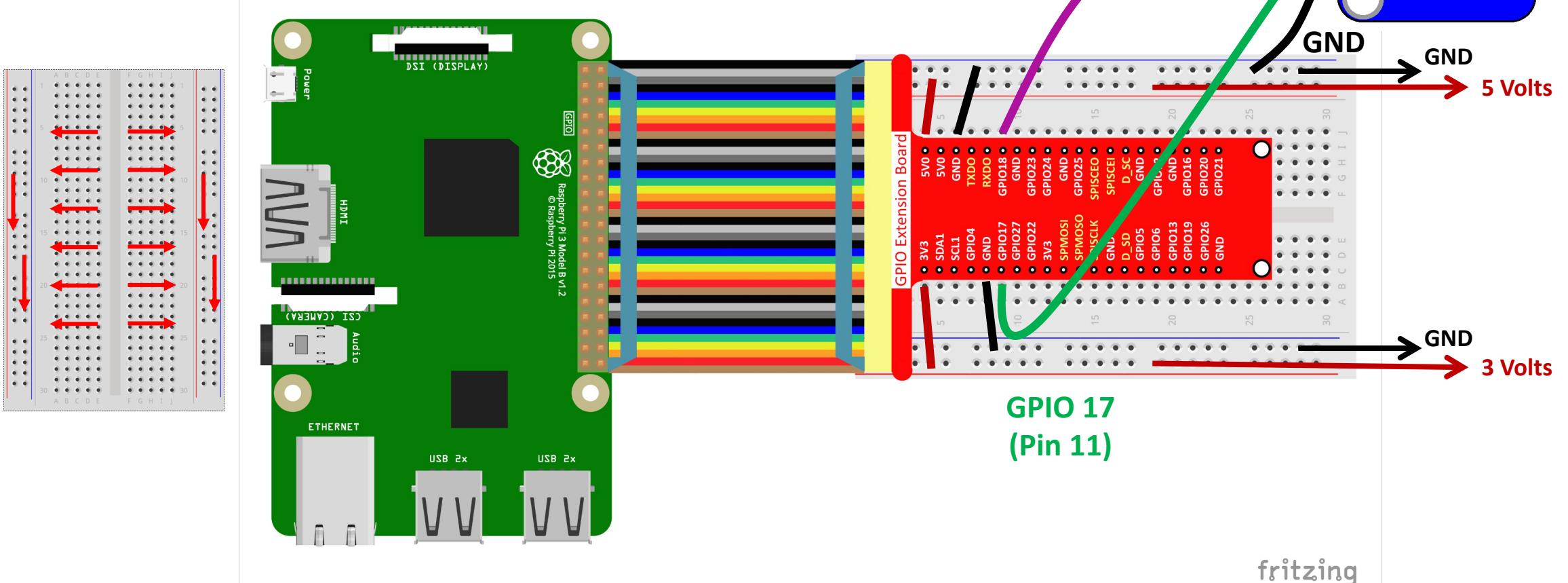
Application:

Variety of devices, such as televisions, digital cameras, and remote controls deploy these type LEDs.

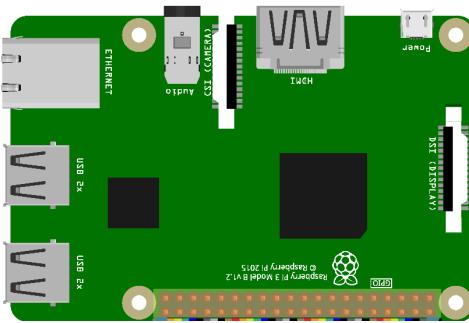
Connector:

3-pin anti-reverse cable

Light up an LED with your Raspberry Pi 3 B +



How a python code can
light up your LED with
Raspberry Pi Model 3 B+
(RPi)



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

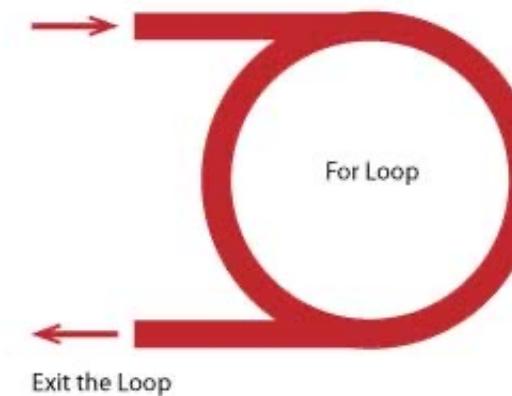
```
GPIO.setmode(GPIO.BCM)
```

GPIO Extension Board	
3	3V3
5	SDA1
7	SCL1
9	GPIO4
11	GND
13	GPIO17
15	GPIO27
17	GPIO22
19	3V3
21	SPMOSI
23	SPMOSO
25	SPISCLK
27	GND
29	D_SD
31	GPIO5
33	GPIO6
35	GPIO13
37	GPIO19
39	GPIO26
	GND
2	5V0
4	5V0
6	GND
8	TXDO
10	RXDO
12	GPIO18
14	GND
16	GPIO23
18	GPIO24
20	GND
22	GPIO25
24	SPISCEO
26	SPISCEI
28	D_SC
30	GND
32	GPIO12
34	GND
36	GPIO16
38	GPIO20
	GND
40	GPIO21

```
GPIO.setup(12, GPIO.OUT)
```

(For) How many times do you want to execute a piece of code ?

Enter the Loop

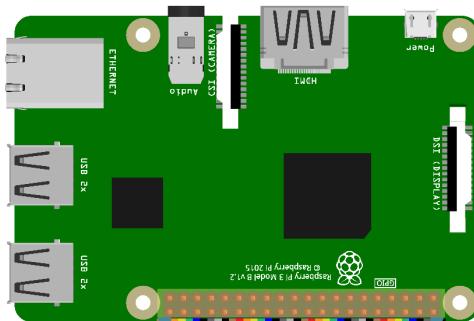


```
for i in range(0,15):
```

```
    GPIO.output(12, GPIO.HIGH)  
    time.sleep(0.5)  
    GPIO.output(12, GPIO.LOW)  
    time.sleep(0.5)  
    print(i)
```

```
GPIO.cleanup()
```

Another python code to kick start your Raspberry Pi Model 3 B+ (RPi)



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

```
GPIO.setmode(GPIO.BOARD)
```

GPIO Extension Board	
1	3V3
3	SDA1
5	SCL1
7	GPIO4
9	GND
11	GPIO17
13	GPIO27
15	GPIO22
17	3V3
19	SPMOSI
21	SPMOSO
23	SPICLK
25	GND
27	D_SD
29	GPIO5
31	GPIO6
33	GPIO13
35	GPIO19
37	GPIO26
39	GND
2	5V0
4	5V0
6	GND
8	TXDO
10	RXDO
12	GPIO18
14	GND
16	GPIO23
18	GPIO24
20	GND
22	GPIO25
24	SPICEO
26	SPICEI
28	D_SC
30	GND
32	GPIO12
34	GND
36	GPIO16
38	GPIO20
40	GPIO21

```
GPIO.setup(12, GPIO.OUT)
```

```
def loop():  
    while True:  
        GPIO.output(12, GPIO.HIGH)  
        time.sleep(0.5)  
        GPIO.output(12, GPIO.LOW)  
        time.sleep(0.5)
```

```
def destroy():  
    GPIO.output(12, GPIO.LOW)  
    # Turn off all leds  
    GPIO.cleanup()
```

```
if __name__ == "__main__":  
    try:  
        loop()  
    except KeyboardInterrupt:  
        destroy()
```

Setting up a Python virtual environment - “Sandbox” your Python projects

“Unwearingly working and making changes to the global Python environment can break Python-based system-level tools, and remedying the situation can become a major headache.”

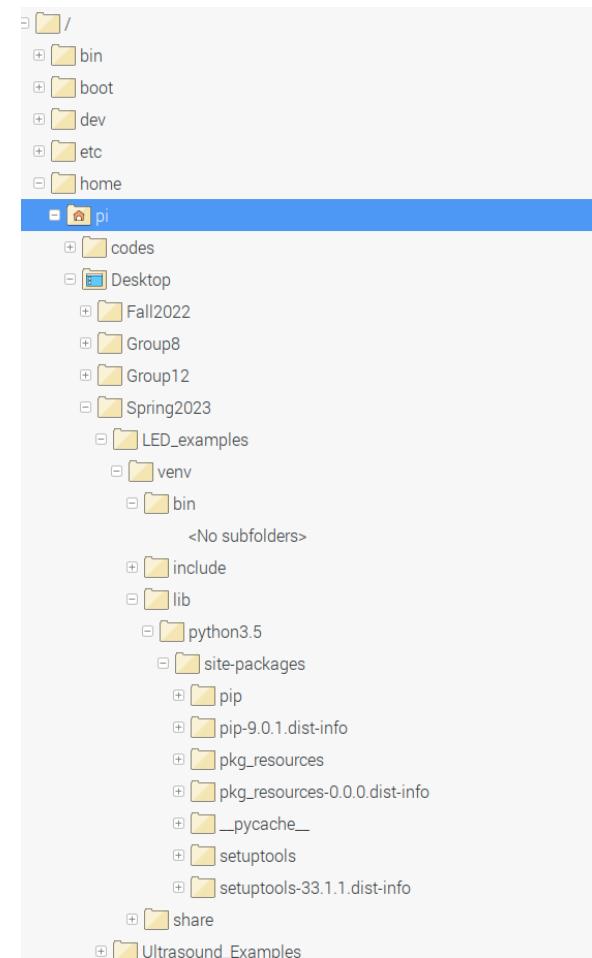
Gary Smart, Practical Python Programming for IoT

Create python virtual environments

- To “sandbox” our Python projects
- Avoid adverse interference with
 - system-level Python utilities
 - or other Python projects.

```
pi@raspberrypi: ~/Desktop/Spring2023/LED_examples
File Edit Tabs Help
pi@raspberrypi:~ $ cd Desktop/Spring2023/LED_examples/
pi@raspberrypi:~/Desktop/Spring2023/LED_examples $
pi@raspberrypi:~/Desktop/Spring2023/LED_examples $
pi@raspberrypi:~/Desktop/Spring2023/LED_examples $ python3 -m venv venv
pi@raspberrypi:~/Desktop/Spring2023/LED_examples $
pi@raspberrypi:~/Desktop/Spring2023/LED_examples $
pi@raspberrypi:~/Desktop/Spring2023/LED_examples $ source venv/bin/activate
(venv) pi@raspberrypi:~/Desktop/Spring2023/LED_examples $
(venv) pi@raspberrypi:~/Desktop/Spring2023/LED_examples $
(venv) pi@raspberrypi:~/Desktop/Spring2023/LED_examples $ which python
/home/pi/Desktop/Spring2023/LED_examples/venv/bin/python
(venv) pi@raspberrypi:~/Desktop/Spring2023/LED_examples $
(venv) pi@raspberrypi:~/Desktop/Spring2023/LED_examples $
(venv) pi@raspberrypi:~/Desktop/Spring2023/LED_examples $ python --version
Python 3.5.3
(venv) pi@raspberrypi:~/Desktop/Spring2023/LED_examples $
(venv) pi@raspberrypi:~/Desktop/Spring2023/LED_examples $
(venv) pi@raspberrypi:~/Desktop/Spring2023/LED_examples $
(venv) pi@raspberrypi:~/Desktop/Spring2023/LED_examples $ deactivate
pi@raspberrypi:~/Desktop/Spring2023/LED_examples $
pi@raspberrypi:~/Desktop/Spring2023/LED_examples $
pi@raspberrypi:~/Desktop/Spring2023/LED_examples $
```

Anatomy of the python virtual environments



Activate python-venv & Install pip

```
pi@raspberrypi:~/Desktop/Spring2023/LED_examples
File Edit Tabs Help
pi@raspberrypi:~/Desktop/Spring2023/LED_examples $ source venv/bin/activate
(venv) pi@raspberrypi:~/Desktop/Spring2023/LED_examples $ pip3 install --upgrade pip
Cache entry deserialization failed, entry ignored
Cache entry deserialization failed, entry ignored
Collecting pip
  Using cached https://files.pythonhosted.org/packages/27/79/8a850fe3496446ff0d584327ae44e7500daf6764ca1a382d
2d02789accf7/pip-20.3.4-py2.py3-none-any.whl
Installing collected packages: pip
  Found existing installation: pip 9.0.1
    Uninstalling pip-9.0.1:
      Successfully uninstalled pip-9.0.1
Successfully installed pip-20.3.4
(venv) pi@raspberrypi:~/Desktop/Spring2023/LED_examples $
(venv) pi@raspberrypi:~/Desktop/Spring2023/LED_examples $
(venv) pi@raspberrypi:~/Desktop/Spring2023/LED_examples $ pip list
DEPRECATION: Python 3.5 reached the end of its life on September 13th, 2020. Please upgrade your Python as Python 3.5 is no longer maintained. pip 21.0 will drop support for Python 3.5 in January 2021. pip 21.0 will remove support for this functionality.
Package           Version
-----
pip              20.3.4
pkg-resources     0.0.0
setuptools       33.1.1
(venv) pi@raspberrypi:~/Desktop/Spring2023/LED_examples $
```

Review of the code provided to you

The screenshot shows the Thonny IDE interface with a Python script named `gpio_pkg_check.py`. The code is a script designed to check for the availability of various Python GPIO library packages. It uses try-except blocks to attempt imports of `gpiozero`, `pigpio`, and `RPi.GPIO`. If an import fails, it prints that the package is unavailable and suggests installing it with pip. The script is annotated with comments explaining its purpose and dependencies.

```
author: Kartik Bulusu (CS Dept., GWU)
Modified by Kartik Bulusu (CS Dept., GWU)

This Python script checks for the availability of various Python GPIO Library Pac
It does this by attempting to import the Python package. If the package import is
we report the package as Available, and if the import (or import initialization)
we report the package as Unavailable.

Dependencies:
  pip3 install gpiozero pigpio RPi.GPIO

Built and tested with Python 3.7 on Raspberry Pi 4 Model B
Tested with Python 3.5.3 on Raspberry Pi 3B+
"""

try:
    import gpiozero
    print('GPIOZero Available')
except:
    print('GPIOZero Unavailable. Install with "pip install gpiozero"')

try:
    import pigpio
    print('PiGPIO Available')
except:
    print('PiGPIO Unavailable. Install with "pip install pigpio"')

try:
    import RPi.GPIO
    print('RPi.GPIO Available')
except:
    print('RPi.GPIO Unavailable. Install with "pip install RPi.GPIO"')
```

Install GPIO packages using the code provided.

Note: You are still in python-venv

Checks and balances:
pip list
pip freeze

```
pi@raspberrypi:~/Desktop/Spring2023/LED_examples
File Edit Tabs Help
(venv) pi@raspberrypi:~/Desktop/Spring2023/LED_examples $ python gpio_pkg_check.py
GPIOZero  Unavailable. Install with "pip install gpiozero"
PiGPIO   Unavailable. Install with "pip install pigpio"
RPi.GPIO  Unavailable. Install with "pip install RPi.GPIO"
(venv) pi@raspberrypi:~/Desktop/Spring2023/LED_examples $
(venv) pi@raspberrypi:~/Desktop/Spring2023/LED_examples $
(venv) pi@raspberrypi:~/Desktop/Spring2023/LED_examples $ pip3 install gpiozero pigpio RPi.GPIO
DEPRECATION: Python 3.5 reached the end of its life on September 13th, 2020. Please upgrade your Python as Python 3.5 is no longer maintained. pip 21.0 will drop support for Python 3.5 in January 2021. pip 21.0 will remove support for this functionality.
Looking in indexes: https://pypi.org/simple, https://www.piwheels.org/simple
Collecting gpiozero
  Using cached https://www.piwheels.org/simple/gpiozero/gpiozero-1.6.2-py2.py3-none-any.whl (148 kB)
Collecting pigpio
  Using cached https://www.piwheels.org/simple/pigpio/pigpio-1.78-py2.py3-none-any.whl (39 kB)
Collecting RPi.GPIO
  Using cached https://www.piwheels.org/simple/rpi-gpio/RPi.GPIO-0.7.1-cp35-cp35m-linux_armv7l.whl (68 kB)
Collecting colorzero
  Using cached https://www.piwheels.org/simple/colorzero/colorzero-2.0-py2.py3-none-any.whl (26 kB)
Requirement already satisfied: setuptools in ./venv/lib/python3.5/site-packages (from colorzero->gpiozero) (33.1.1)
Installing collected packages: colorzero, RPi.GPIO, pigpio, gpiozero
Successfully installed RPi.GPIO-0.7.1 colorzero-2.0 gpiozero-1.6.2 pigpio-1.78
(venv) pi@raspberrypi:~/Desktop/Spring2023/LED_examples $ 
```

```
pi@raspberrypi:~/Desktop/Spring2023/LED_examples
File Edit Tabs Help
(venv) pi@raspberrypi:~/Desktop/Spring2023/LED_examples $ pip list
DEPRECATION: Python 3.5 reached the end of its life on September 13th, 2020. Please upgrade your Python as Python 3.5 is no longer maintained. pip 21.0 will drop support for Python 3.5 in January 2021. pip 21.0 will remove support for this functionality.
Package          Version
-----
colorzero        2.0
gpiozero         1.6.2
pigpio           1.78
pip              20.3.4
pkg-resources    0.0.0
RPi.GPIO         0.7.1
setuptools       33.1.1
(venv) pi@raspberrypi:~/Desktop/Spring2023/LED_examples $ python gpio_pkg_check.py
GPIOZero  Available
PiGPIO   Available
RPi.GPIO  Available
(venv) pi@raspberrypi:~/Desktop/Spring2023/LED_examples $ pip freeze > requirements.txt
DEPRECATION: Python 3.5 reached the end of its life on September 13th, 2020. Please upgrade your Python as Python 3.5 is no longer maintained. pip 21.0 will drop support for Python 3.5 in January 2021. pip 21.0 will remove support for this functionality.
(venv) pi@raspberrypi:~/Desktop/Spring2023/LED_examples $ 
```

pip stands for “Python installs packages”.
pip queries the Python Package Index, or simply PyPi
Web source for PyPi: <https://pypi.org>

Run the python codes
Blink your LEDs
Note: You are still in python-venv

```
pi@raspberrypi:~/Desktop/Spring2023/LED_examples
File Edit Tabs Help
(venv) pi@raspberrypi:~/Desktop/Spring2023/LED_examples $ which python
/home/pi/Desktop/Spring2023/LED_examples/venv/bin/python
(venv) pi@raspberrypi:~/Desktop/Spring2023/LED_examples $
(venv) pi@raspberrypi:~/Desktop/Spring2023/LED_examples $ python --version
Python 3.5.3
(venv) pi@raspberrypi:~/Desktop/Spring2023/LED_examples $
(venv) pi@raspberrypi:~/Desktop/Spring2023/LED_examples $
(venv) pi@raspberrypi:~/Desktop/Spring2023/LED_examples $ python gpio_pkg_check.py
GPIOZero Available
PiGPIO Available
RPi.GPIO Available
(venv) pi@raspberrypi:~/Desktop/Spring2023/LED_examples $ python LED_1.py
0
1
2
3
4
(venv) pi@raspberrypi:~/Desktop/Spring2023/LED_examples $ 
```

deactivate your venv

```
pi@raspberrypi:~/Desktop/Spring2023/LED_examples
File Edit Tabs Help
(venv) pi@raspberrypi:~/Desktop/Spring2023/LED_examples $ deactivate
pi@raspberrypi:~/Desktop/Spring2023/LED_examples $
pi@raspberrypi:~/Desktop/Spring2023/LED_examples $
pi@raspberrypi:~/Desktop/Spring2023/LED_examples $
pi@raspberrypi:~/Desktop/Spring2023/LED_examples $
pi@raspberrypi:~/Desktop/Spring2023/LED_examples $ 
```

Running a python script at boot cron, the UNIX scheduler

Awesome IoT project will run automatically every time you start your Raspberry Pi

Review `run_on_boot.sh`

Note: You are no longer in `python-venv`

Let's make sure we have a
file-folder structure that
looks some what like this

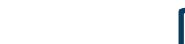


Let's open the `run_on_boot.sh`
on a terminal
to review the script



Absolute paths to

1. Python interpreter in venv
2. Python script
3. Locate the LOG-file



- The last two lines tie everything together
1. echo [updates LOG-file with string in quotes]
 2. Next line runs py-file and logs output on boot
 3. 2>&1 [logs any additional errors during run]



```
pi@raspberrypi:~/Desktop/Spring2023/LED_examples $ cat run_on_boot.sh
#!/bin/bash
#File: LED_examples/run_on_boot.sh

# Absolute path to Virtual Environment python interpreter
PYTHON=/home/pi/Desktop/Spring2023/LED_examples/venv/bin/python

# Absolute path to Python script
SCRIPT=/home/pi/Desktop/Spring2023/LED_examples/LED_1.py

# Absolute path to output log file
LOG=/home/pi/Desktop/Spring2023/LED_examples/LED_1.log

echo -e "\n##### STARTUP $(date) #####\n" >> $LOG
$PYTHON $SCRIPT >> $LOG 2>&1
pi@raspberrypi:~/Desktop/Spring2023/LED_examples $
```

chmod u+x run_on_boot.sh
current User, make the file executable

```
pi@raspberrypi: ~/Desktop/Spring2023/LED_examples
File Edit Tabs Help
pi@raspberrypi:~/Desktop/Spring2023/LED_examples $ cat run_on_boot.sh
#!/bin/bash
#File: LED_examples/run_on_boot.sh

# Absolute path to Virtual Environment python interpreter
PYTHON=/home/pi/Desktop/Spring2023/LED_examples/venv/bin/python

# Absolute path to Python script
SCRIPT=/home/pi/Desktop/Spring2023/LED_examples/LED_1.py

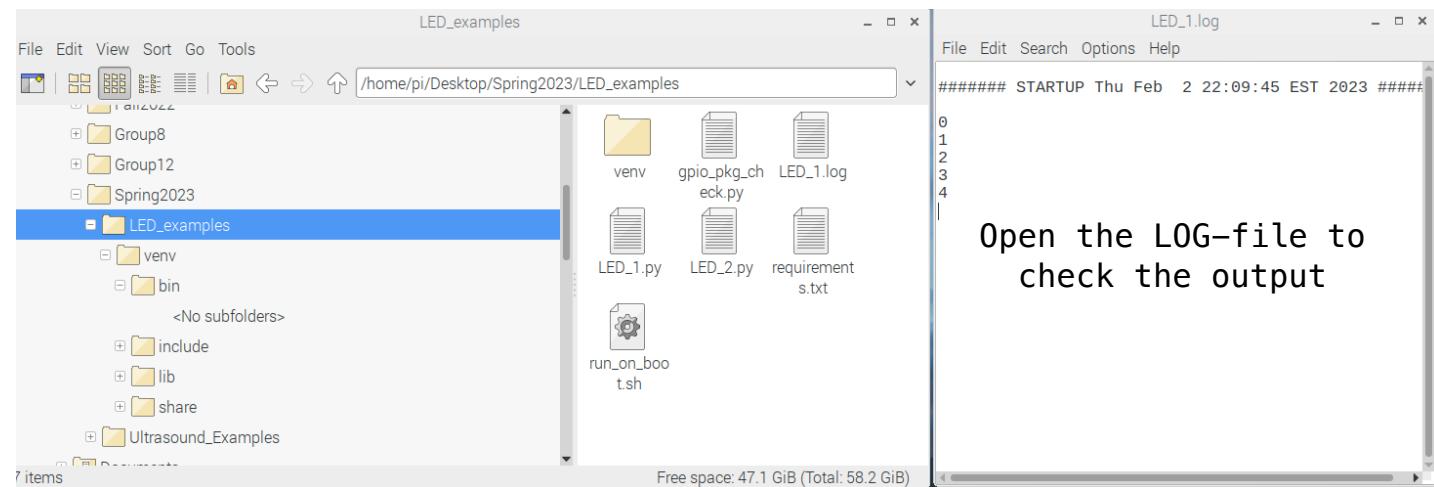
# Absolute path to output log file
LOG=/home/pi/Desktop/Spring2023/LED_examples/LED_1.log

echo -e "\n##### STARTUP $(date) #####\n" >> $LOG

$PYTHON $SCRIPT >> $LOG 2>&1
pi@raspberrypi:~/Desktop/Spring2023/LED_examples $
pi@raspberrypi:~/Desktop/Spring2023/LED_examples $ chmod u+x run_on_boot.sh
pi@raspberrypi:~/Desktop/Spring2023/LED_examples $ 
```

./run_on_boot.sh

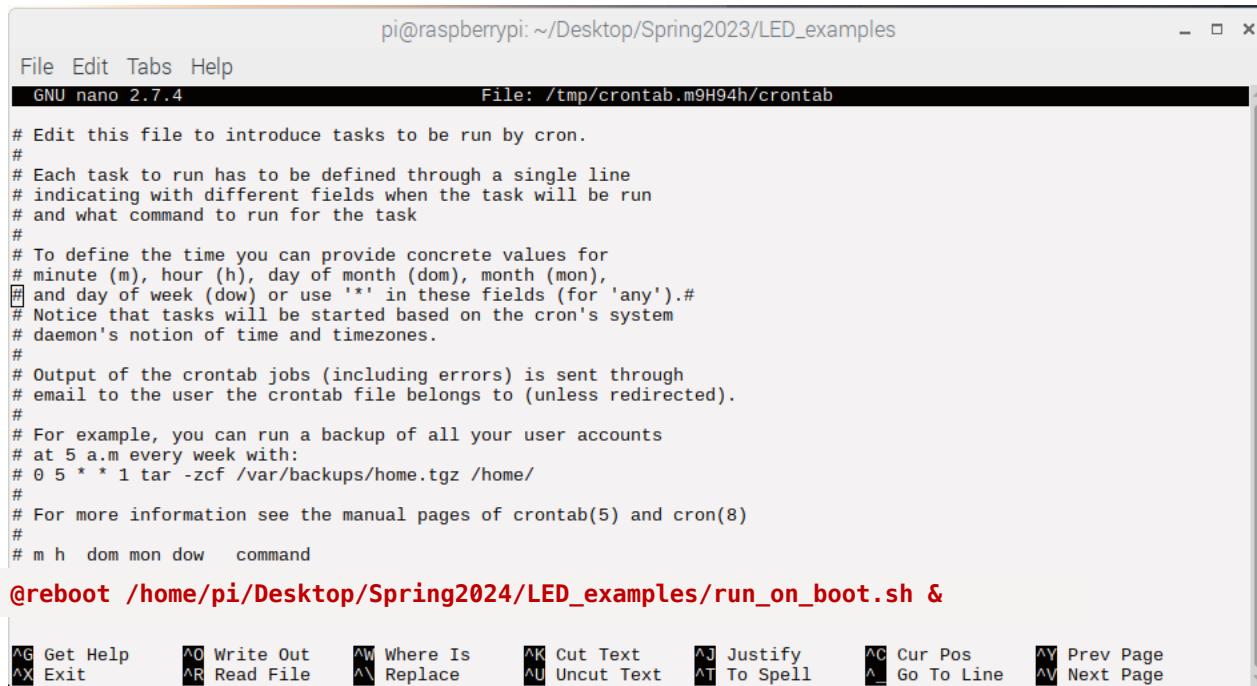
```
pi@raspberrypi: ~/Desktop/Spring2023/LED_examples
File Edit Tabs Help
pi@raspberrypi:~ $ cd Desktop/Spring2023/LED_examples/
pi@raspberrypi:~/Desktop/Spring2023/LED_examples $ ./run_on_boot.sh
pi@raspberrypi:~/Desktop/Spring2023/LED_examples $ 
```



crontab -e

```
$ crontab -e
```

```
@reboot /home/pi/Desktop/Spring2024/LED_examples/run_on_boot.sh &
```



```
pi@raspberrypi: ~/Desktop/Spring2023/LED_examples
File Edit Tabs Help
GNU nano 2.7.4
File: /tmp/crontab.m9H94h/crontab

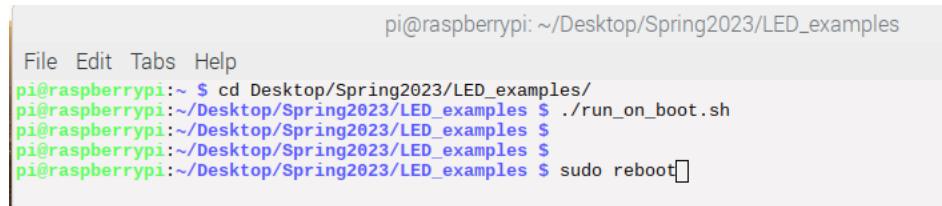
# Edit this file to introduce tasks to be run by cron.
#
# Each task to run has to be defined through a single line
# indicating with different fields when the task will be run
# and what command to run for the task
#
# To define the time you can provide concrete values for
# minute (m), hour (h), day of month (dom), month (mon),
# and day of week (dow) or use '*' in these fields (for 'any').#
# Notice that tasks will be started based on the cron's system
# daemon's notion of time and timezones.
#
# Output of the crontab jobs (including errors) is sent through
# email to the user the crontab file belongs to (unless redirected).
#
# For example, you can run a backup of all your user accounts
# at 5 a.m every week with:
# 0 5 * * 1 tar -zcf /var/backups/home.tgz /home/
#
# For more information see the manual pages of crontab(5) and cron(8)
#
# m h dom mon dow   command
@reboot /home/pi/Desktop/Spring2024/LED_examples/run_on_boot.sh &

^G Get Help      ^O Write Out    ^W Where Is      ^K Cut Text      ^J Justify      ^C Cur Pos      ^Y Prev Page
^X Exit          ^R Read File     ^V Replace       ^U Uncut Text    ^T To Spell     ^A Go To Line   ^V Next Page
```

"Do not forget the & character at the end of the line. This makes sure the script runs in the background."

Gary Smart, Practical Python Programming for IoT

Try manually ./run_on_boot.sh
then
sudo reboot



```
pi@raspberrypi: ~/Desktop/Spring2023/LED_examples
File Edit Tabs Help
pi@raspberrypi:~ $ cd Desktop/Spring2023/LED_examples/
pi@raspberrypi:~/Desktop/Spring2023/LED_examples $ ./run_on_boot.sh
pi@raspberrypi:~/Desktop/Spring2023/LED_examples $ 
pi@raspberrypi:~/Desktop/Spring2023/LED_examples $ 
pi@raspberrypi:~/Desktop/Spring2023/LED_examples $ sudo reboot
```