Raspberry Pi Programming

Written by Mark Webster, and ??

Summary: Learn basic Python programming for the Raspberry Pi small board computer.

Introduction

The Raspberry Pi small board computer supports programming in most of the main computer languages such as C, C++, Python, Java, Scratch, Javascript, Ruby, HTML5, and others. The easiest language to write Raspberry Pi specific programs is Python. This workshop focuses on programming in Python 3 which is the newest and dominant version of Python.

Python Basics

The Python language is an interpreted language that doesn’t compile down to machine code like C or C++. Instead of compiling an interpreter program is run on the local computer which reads and executes the text based python program. This allows Python code to be cross platform. Since Python interpreters have been written for most major computers this make Python almost universal.
The advantages of Python iare:

1. cross platform
2. many support libraries
3. Faster programming. Often a Python program can be written in half the time of a C program.

Disadvantages of Python are:

1) Slower execution
2) More memory required
3) Slower start up of a program since the interpreter must be loaded first.

A tutorial for Python on the Raspberry Pi can be found at:
https://www.raspberrypi.org/documentation/usage/python/

Python Comments

Comments are essential in any language. Lines in Python that begin with a # symbol are ignored by the interpreter as a comment. Multiline comments are bounded by triple quotes.

```
# this line is a comment

“””
Both these lines
Are comments
“””
```

Python Line Grouping
Python does not use curly braces { } to bound groups of lines. Instead line indentation is used. Don’t use tabs, use the same number of spaces to group lines. The two indented lines are inside the for loop.

```
for i in range(3):
        print(i)
        counter = counter + 1
```

Python lines do not end with a semicolon. The new line is the end of a line.

Python Variables
Like all computer languages Python assigns memory locations to named variables. Python variables are not strongly typed so there is no declaration like “int”, “float”. In Python a value is just assigned to a variable name and that creates the variable. The interpreter automatically determines the type of the variable, string, integer, float, etc. For example, two variables Lastname and Pi can be created:

```
Lastname = “Webster”
Pi = 3.14159
```

Variable types can be changed dynamically, for example

```
Pi = 3.14159
Pi = “Raspberry”
```

Executing Python Programs

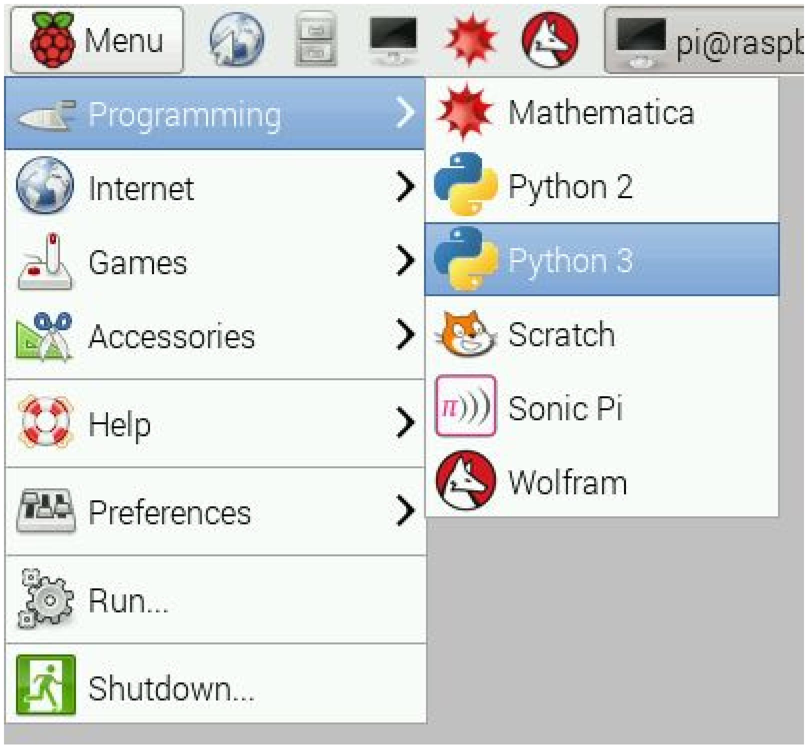
The print() function in Python will display the variable value.

Python programs can be written in a text editor and executed at the command line with:

```
python3 myProgramName
```

However, a common way to test and develop Python programs is with an IDLE. Start the IDLE environment. Load the module (program) from the File and then Run it from the menu on the module window.

Python can be interactively executed line by line in the interactive development environment called IDLE. Raspberry Pi lists both Python 2 and Python 3 IDLE environments on the main menu in the category “Programming”



Python Loops
Python loops are based on the idea of iterators instead of a loop index variable like C or C++. In the C language a loop might be:

```
for( i=0; i<3; i++) {
// some stuff here
}
```

In Python the loop would be (notice the semicolon at the end of the line)

```
For i in range(3):
   // Some stuff goes here
```

The power of an iterator is shown by an example of looping through all the characters in a name and printing them:

```
myName = “Fred Flintstone”
For aChar in myName:
        print(aChar)
```

Python Logical Branching

If statements are simple, just follow the indentation rule.

```
name = "Joe"

if len(name) > 3:
   print("Nice name,")
   print(name)
else:
   print("That's a short name,")
   print(name)
```

Installing Python Modules

One of the huge benefits of Python is the large number of libraries or modules that have been created for the language. These can either be installed from the command line, such as:

```
sudo apt update
sudo apt install python-picamera
```

If the Python library is not available in the Raspbian repository, then the PIP installer can be used.

```
sudo apt install python3-pip
```

An example of using pip to install the simplejson library is:

```
sudo pip3 install simplejson
```

Example 1: Write a Python program to count to 10

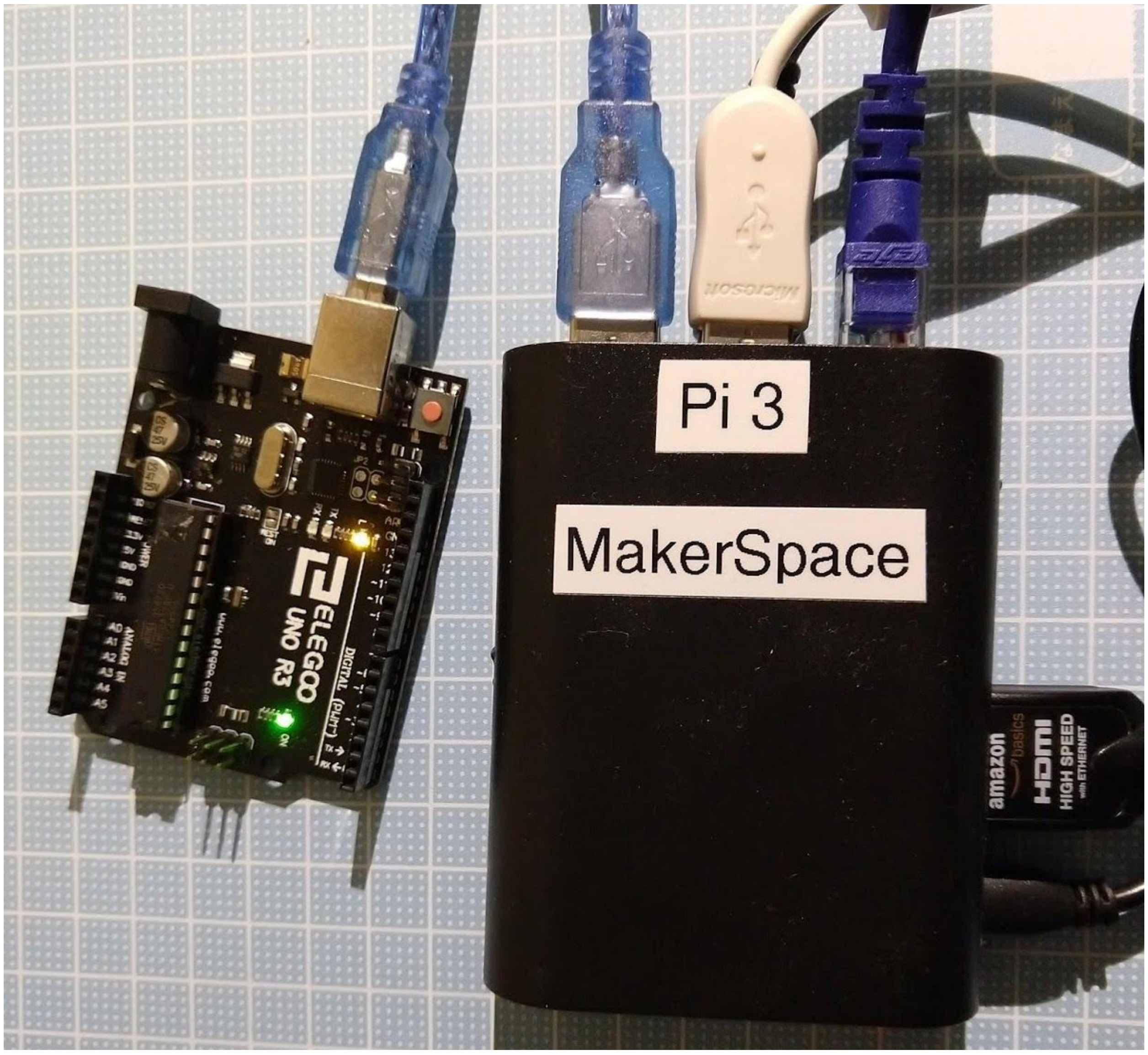
Using a text editor write a Python program with a for loop. Use the range() function to iterate from 0 to 100. Then print the value with print(i)

```
for i in range(100):
   print(i)
```

Run the program at the command line with python3 myloop.py

Python Serial Port

Software: python3, python-serial package, another laptop or an Arduino connected to a USB port on the Raspberry Pi. The laptop/Arduino should be sending perioding lines of text.



Example Arduino Uno code:

```
int i = 0;
void setup() {
  Serial.begin(9600);
}

void loop() {
   i++;
  Serial.println(i);
  delay(1000);
}
```

Note: the Arduino code can be debugged using the serial monitor tool.

From a laptop a terminal emulation program like Putty can be used to send strings to the Raspberry Pi.

If not already done, first install the serial port package. Normally it is installed by default in Raspbian.

```
sudo apt-get install python-serial
```

Once installed Python code can be used to read and write data to a serial port, for example, to read and write to an Arduino.

First identify the serial port the Arduino or laptop is on. For example on the Raspberry Pi the port might be “/dev/ttyACM0” and on Windows might be “COM1”. The port name will be stored in the variable “thePort”.

```
thePort = “/dev/ttyACM0”
```

At the start of the program import the serial library

```
import serial
```

To open a port whose name is stored in the variable “thePort” at 9600 baud use

```
serialport = serial.Serial(str(thePort), 9600, timeout=0.5)
```

Note: the baud rate in Python must be the same as the baud rate on the Arduino or laptop.

To read and entire line (until the EOL) from a port use:

```
line = serialport.readline()
```

To write to a serial port use :

```
serialport.write("Say something:")
```

====Example Python3 code=============

```
import serial
thePort = “/dev/ttyACM0”

serialport = serial.Serial(str(thePort), 9600, timeout=0.5)

while True:
   line = serialport.readline()
# trim off the \r\n at the end of the string
   line = line.rstrip()
# convert bytes to a string
   line = line.decode(“utf-8”)
# if not a blank line then print
   if len(line)>0 :
        print(line)
```

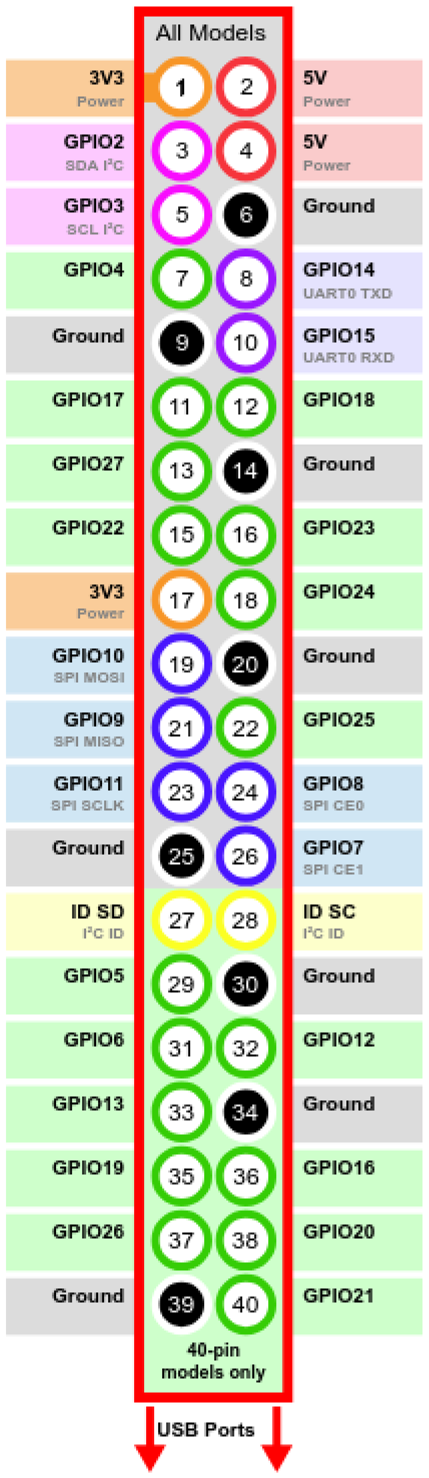
Execute the program with python3 myFilename.py
Note: this program runs forever. Terminate with a control-C

Python GPIO

Hardware: LED +470 ohm resistor on pin 17. Push button on pin 2 connected to ground.
Software: Python GPIO library installed. By default it is installed with Raspbian.
A Raspberry Pi differs from most laptop or desktop computers because it has GPIO (general purpose IO) pins. These are not as powerful as the pins on a microcontroller like Arduino, but they can work like digital pins and allow some direct interfaces with external electronics.

A confusing fact about the GPIO is there are two different numbering schemes: the hardware pin number and the software visible pin number. The software numbers are GPIO17, etc. The hardware numbers are 1,2,3…. Notice that GPIO17 = pin 11. The Python library uses either numbering system but defaults to GPIO numbering. For example, red = LED(17) means GPIO17 not pin 17.

The GPIO port can provide 3.3 volt or 5 volt power.



Official documentation for Python access to GPIO pins can be found at:
https://gpiozero.readthedocs.io/en/stable/
Use the menu at left of the documentation webpage to dive into details.

A brief tutorial can be found at:
https://www.raspberrypi.org/documentation/usage/gpio/python/README.md

The library used for GPIO access from Python is installed by default with Raspbian. An example of blinking an LED on pin 17 is:

To control an LED connected to GPIO17, you can use this code:

```
from gpiozero import LED
from time import sleep

led = LED(17)

while True:
   led.on()
   sleep(1)
   led.off()
   sleep(1)
```

There are other methods if you want the process to run in a separate thread

```
from gpiozero import LED
from signal import pause

red = LED(17)

red.blink()

pause()
```

Or to fade the LED with something like PWM on the Arduino,

```
from gpiozero import PWMLED
from signal import pause

led = PWMLED(17)

led.pulse()
```

```
pause()
```

The brightness of the LED can be changed with led.value,

To read the state of a button on GPIO pin 2 here is an example Python code:

```
from gpiozero import Button
from time import sleep

button = Button(2)

while True:
    if button.is_pressed:
        print("Pressed")
    else:
        print("Released")
    sleep(1)
```

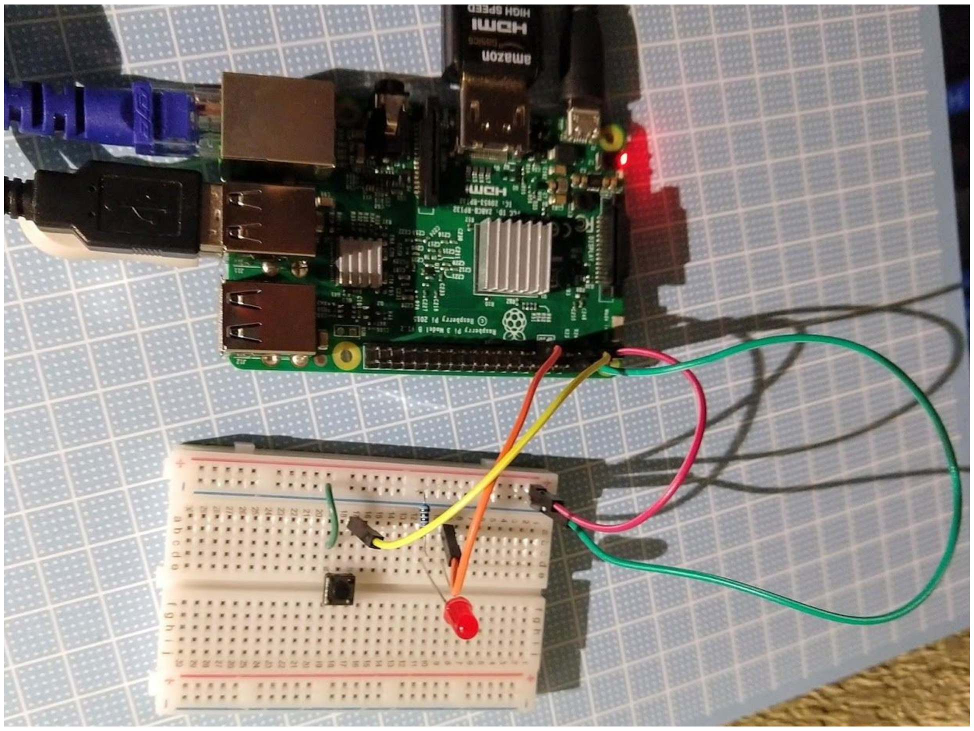
Note the indentation.

To run in a separate process the LED and button objects can be can be combined

```
from gpiozero import LED, Button
from signal import pause

led = LED(17)
button = Button(3)

button.when_pressed = led.on
button.when_released = led.off

pause()
```

Example 2: Turn on an LED with a button

Hardware: Raspberry Pi 3, breadboard, push button, LED, about 350 ohm resistor, M-F jumper wires
Software: Rasbian, Python3, python gpio installed (by default)



Use the code examples above to turn on and off an LED using a push button. This example uses the python gpiozero library code. To write lower level code with concepts more like the Arduino use lines like: GPIO.input(10) == GPIO.HIGH

The two button and LED examples above can be combined with:

```
from gpiozero import LED, Button
from time import sleep

led = LED(17)
button = Button(2)

while True:
    if button.is_pressed:
        led.on()
    else:
        led.off()

    sleep(0.01)
```

When the button is pressed, the LED turns on, and turns off when the button is released. Terminate the program with control-C

Python Graphics

There are many, many graphics libraries for Python. Generally a GUI application revolves around an infinite event loop, like the “void loop()” function for Arduino microcontrollers.

Tkinter

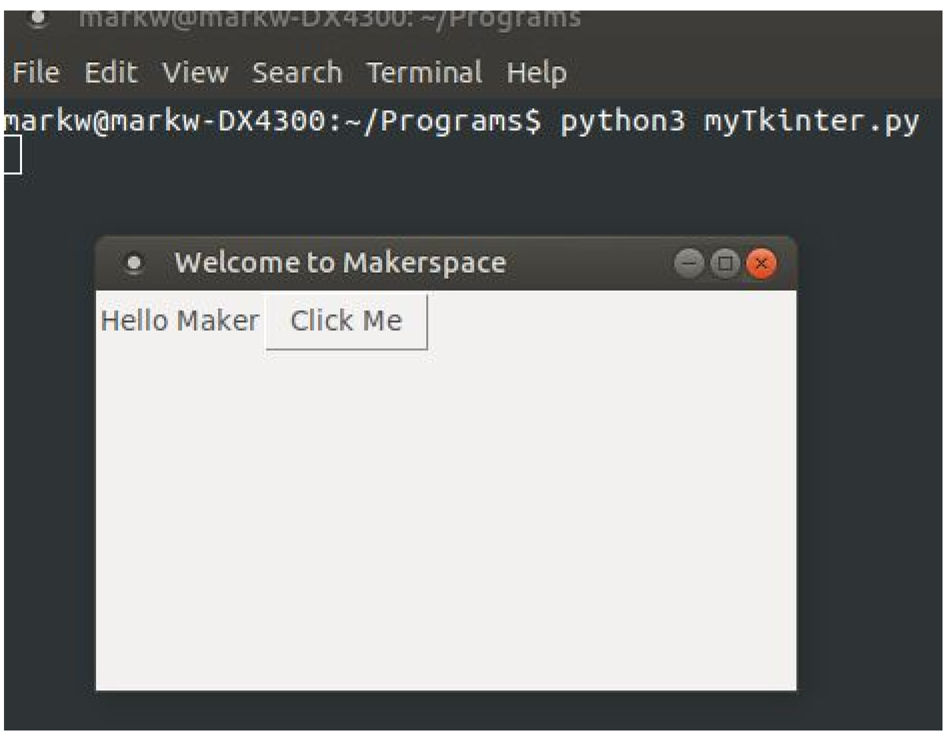
Business and application software requires windows, dialog boxes, listboxes, forms, and other graphical objects. The TK software provides a robust set of GUI objects for creation of application software.

Tkinter is the Python interface to the TK GUI system. Tkinter is available on most linux systems, Windows, and Raspberry Pi. Documentation can be found at:
https://docs.python.org/3/library/tkinter.html

Example 3: Python Tkinter dialog box

Software: Install the Tkinter python package. sudo apt-get install python-tk. Usually already installed.

Type this code into a text file called “myTkinter.py”. It will create a window with a title, put text in the window, create a button, create a function which is called if the button is clicked, and then go into the endless event loop.



```
markw@markw-DX4300:~/Programs$ python3 myTkinter.py
```

markw@markw-DX4300: ~/Programs
File Edit View Search Terminal Help
Welcome to Makerspace
Hello Maker
Click Me

```
from tkinter import *

#create a window, give it a title, define its size
window = Tk()
window.title("Welcome to Makerspace")
window.geometry('350x200')

#create a text label and define where it will be placed in the grid
lbl = Label(window, text="Hello Maker")
lbl.grid(column=0, row=0)

# function to be called when the button is clicked
def clicked():
   lbl.configure(text="Button was clicked !!")

#create a button and define its position on the grid
btn = Button(window, text="Click Me", command=clicked)
btn.grid(column=2, row=0)

#the main event loop
window.mainloop()
```

To execute the myTkinter.py program, at the command line type “python3 myTkinter.py”. Click away and enjoy.

Pygame

Game software requires a different GUI than business oriented applications since games are graphics intensive and don’t have the standard dialog boxes, forms, etc. A Python library has evolved to support game development.

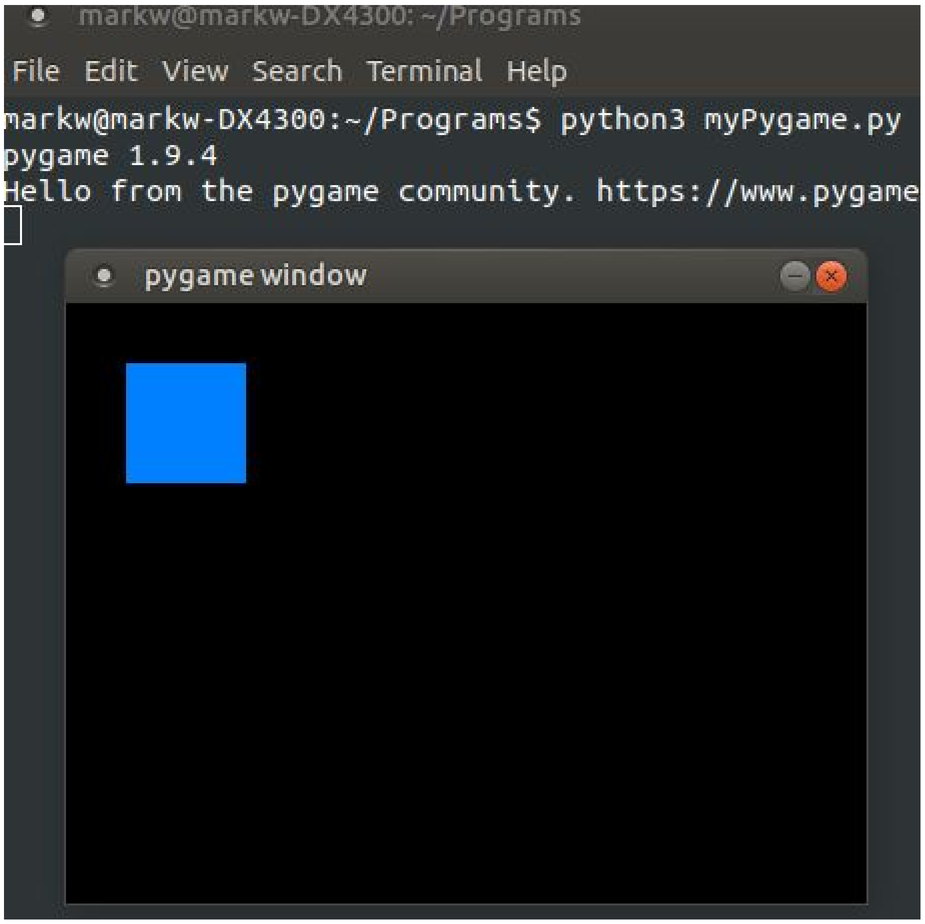
Many python games use the Pygame graphics and game development library.
Full documentation is at: https://www.pygame.org/docs/

Pygame is not installed by default and can be downloaded with the pip software.

```
python3 -m pip install -U pygame --user
```

Example 4: Python Pygame window

Software: pygame module



markw@markw-DX4300: ~/Programs
File Edit View Search Terminal Help
pygame window

```
markw@markw-DX4300:~/Programs$ python3 myPygame.py
pygame 1.9.4
Hello from the pygame community. https://www.pygame
```

After pygame is installed (which it usually is by default), type the following code into a text editor

```
import pygame

pygame.init()
screen = pygame.display.set_mode((400, 300))
done = False

#the main event loop
while not done:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            done = True

    pygame.display.flip()
```

```
# RGB description is (0,128,255)
# Rectangle is (upper leftx, upper lefty, lower rightx, lower righty) or (30,30,60,60)
        pygame.draw.rect(screen, (0, 128, 255), pygame.Rect(30, 30, 60, 60))
```

Execute at the command line with “python3 myPygame.py”