**Lab 2. Digital I/O in Raspberry PI**

**Objectives**

* Performing digital input/output operation in Raspberry Pi
* Interfacing Switches and LEDs

**Equipment and Tools**

* Monitor
* HDMI to VGA converter
* IDLE IDE
* Raspberry Pi 3 with Raspbian Installed
* Micro usb cable
* Switches
* 270 Ω Resistor
* LEDs

**Raspberry Pi GPIOs**

Pins in Raspberry Pi, as shown in Figure.1, are used to perform digital input/output operations. Raspberry Pi B+ and 2 have 40 pins. Of these 40 pins, 26 are GPIO pins and the others are power or ground pins. GPIO pins can be imagined of as software controlled switches. These can be turned ON/OFF programmatically. Also, these can be used to read status of a pin that is configured as input.

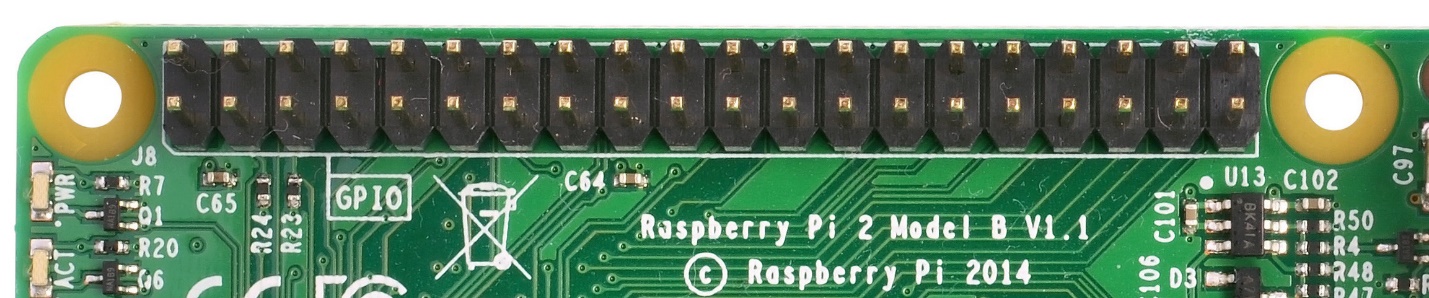
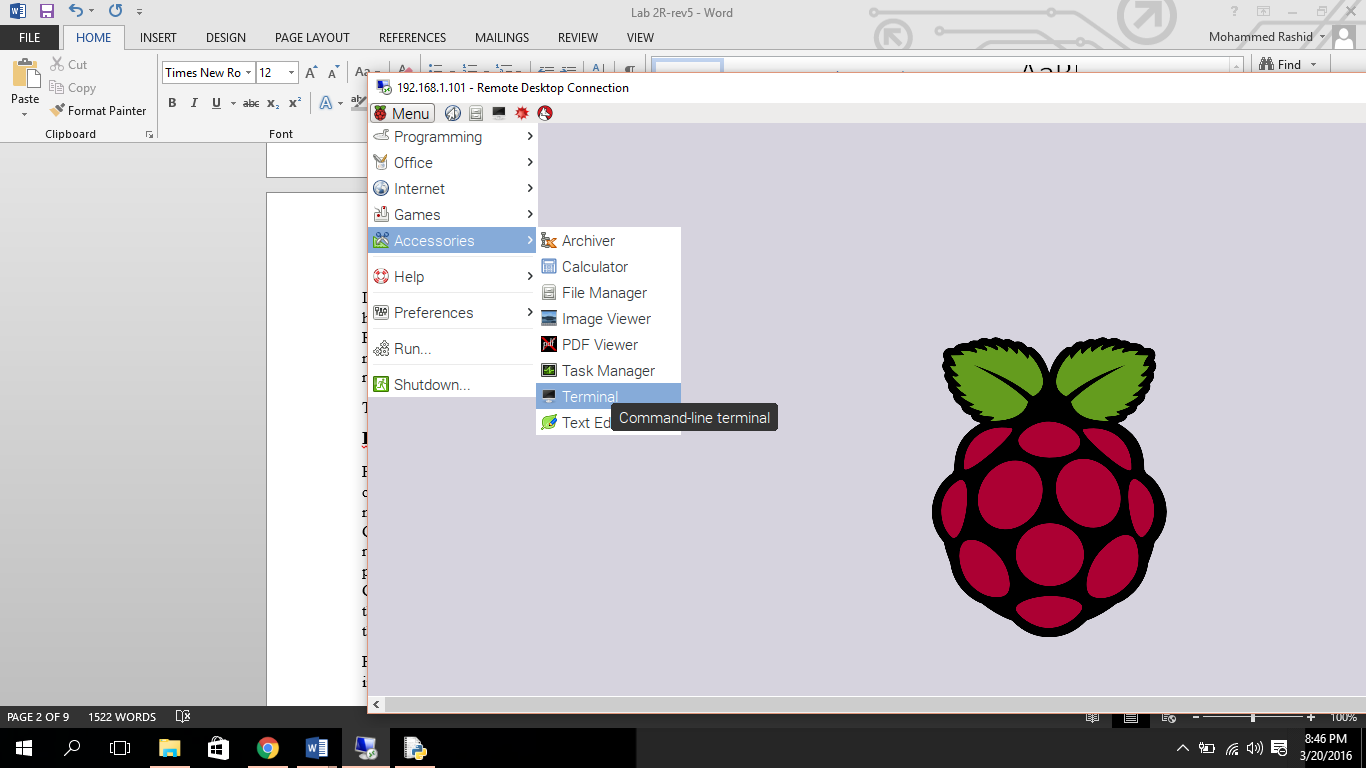


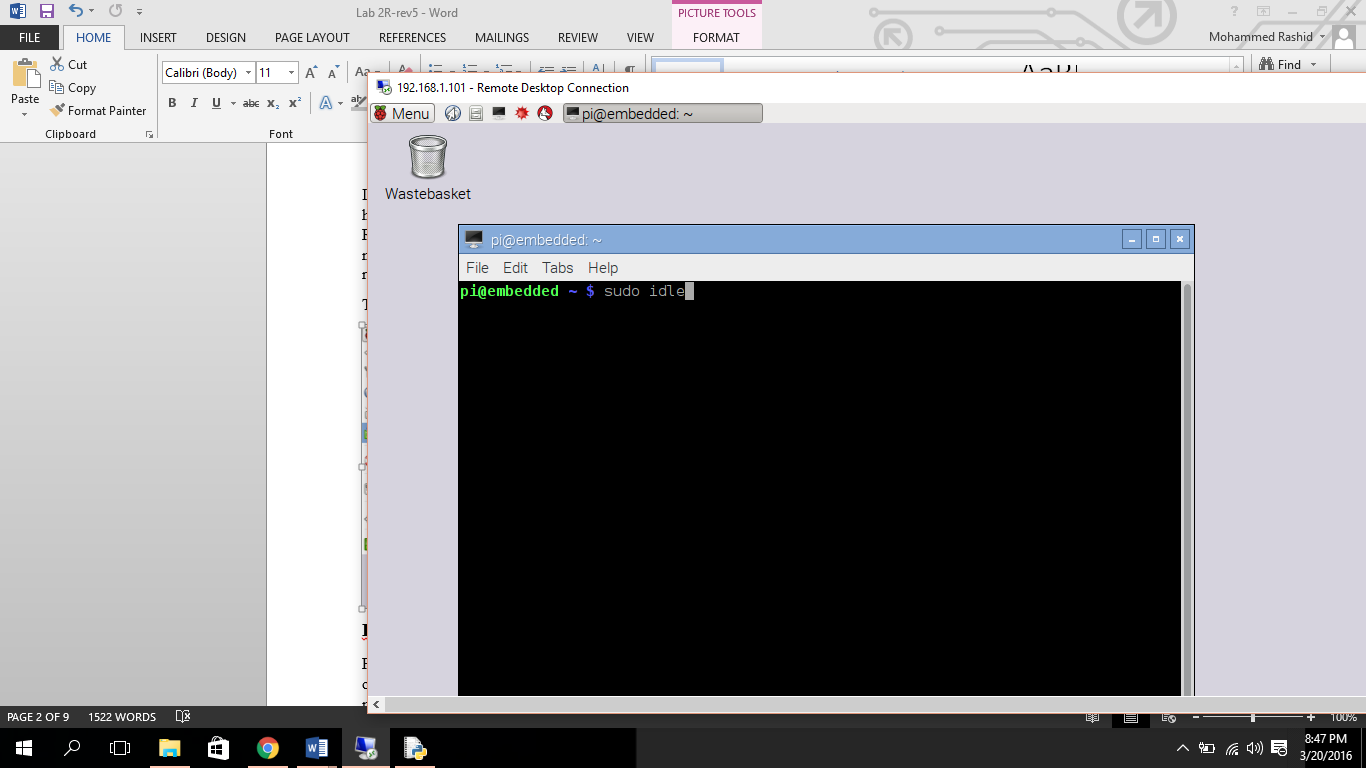
Figure 1. Raspberry Pi Pins

In order to control General Purpose Input Output (GPIO) pins from python, a library call RPi.GPIO has to be imported. This library provides functions to control the operation of GPIO pins in Raspberry Pi. This library comes pre-installed in Raspbian. For GPIO pins to be accessible, we need super user permission. To acquire super user permission, the program should be executed as root.

This is achieved by starting IDLE as superuser. To start as super user, go to terminal.

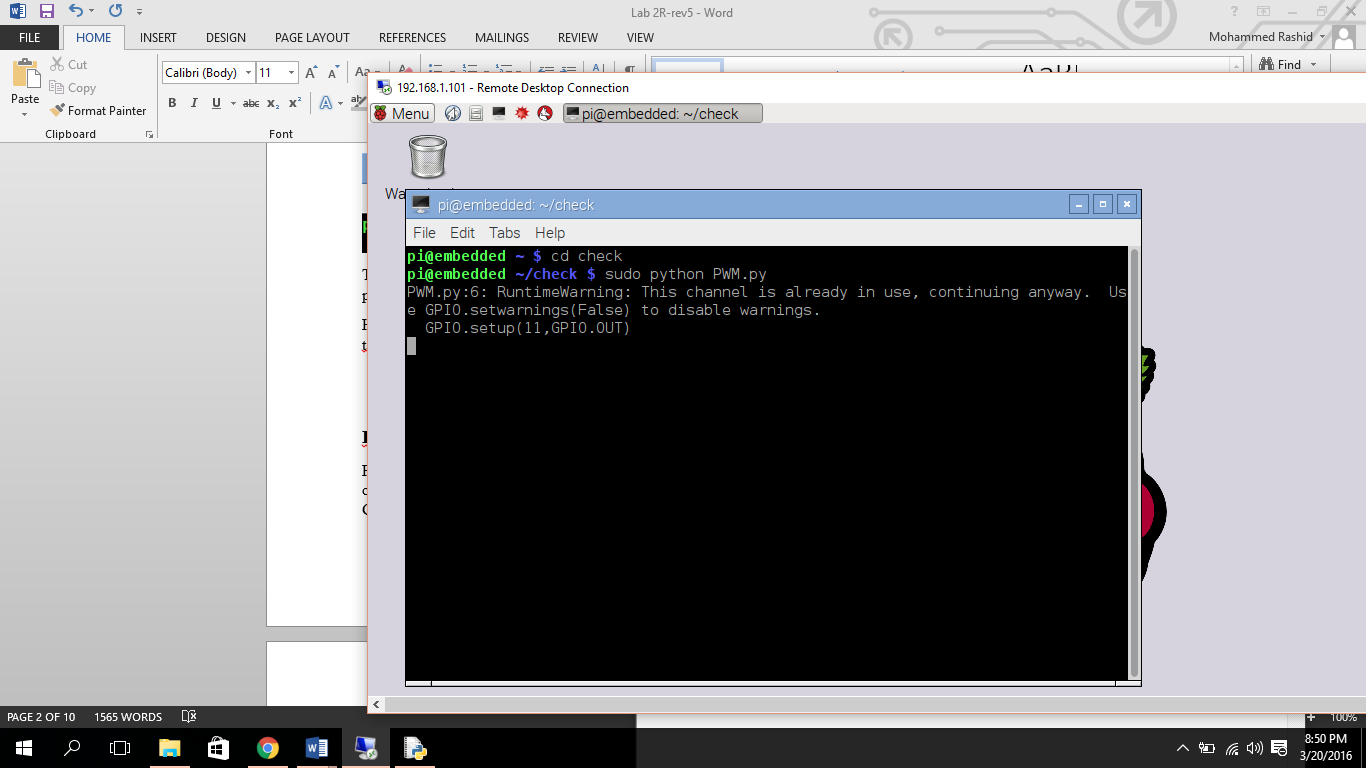


Once in terminal, type sudo idle. This is done to execute program as super user.

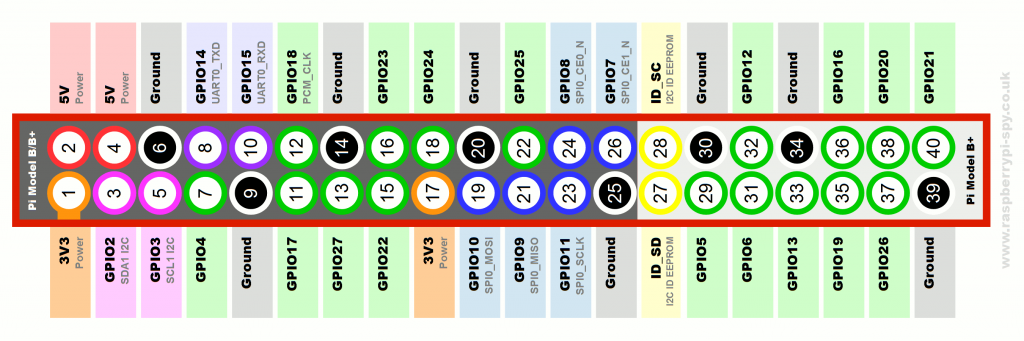


This gives programs executed via idle the permission to access GPIO pins.

However, if you want to execute the .py script from terminal itself, you do as following:



**RPi.GPIO Library**

For the GPIO Pins to be usable, pin mode has to be configured first. There are two modes of operation, namely, BOARD mode and BCM mode. The GPIO.BOARD option specifies that you are referring to the pins by the number of the pin the plug. This numbers goes from 1-40. Whereas, GPIO.BCM option means that you are referring to the pins by the "Broadcom SOC channel" number, these are the numbers after "GPIO".

For instance GPIO25 in BCM corresponds to Pin 22 in BOARD mode.

**Setting PIN Mode**

**Syntax:** GPIO.setmod(x), where x is GPIO.BCM or GPIO.BOARD

To set the GPIO mode we pass the board mode to setmode() function.

**Setting up pins as Input or Output**

**Syntax:** GPIO.setup(y,x), where y is pin number in BCM or BOARD format and x is GPIO.OUT or GPIO.IN

Next step is to initialize pin as either input or output. This done passing pin number followed by port direction (input or output) to setup() function.

Figure 2. *Pin Numbering*

To configure pin 7 as output and GPIO14 as input

|  |  |
| --- | --- |
| In BCM mode | In BOARD mode |
| GPIO.setmode(GPIO.BCM) | GPIO.setmode(GPIO.BOARD) |
| GPIO.setup(4,GPIO.OUT) | GPIO.setup(7,GPIO.OUT) |
| GPIO.setup(14,GPIO.IN) | GPIO.setup(8,GPIO.IN) |

**Reading Values (Digital Input Port)**

**Syntax:** GPIO.input(x), where x is pin number in BCM or BOARD format

To read value from an input pin, we use input function. Input function takes pin number as parameter. For instance, to read value from pin 8 [BOARD mode], we write GPIO.input(8).

**Writing Values (Digital Output Port)**

**Syntax**: GPIO.output(y,x), where y is pin number in BCM or BOARD format and x is (1/GPIO.HIGH/True (ON) or 0/GPIO.LOW/False (OFF) )

To write **(Output)** value to a output pin, output() function is used. Output function takes pin number and (1/GPIO.HIGH/True (ON) or 0/GPIO.LOW/False (OFF)) value as the parameters. To write to pin 8 high or to turn on pin 8, we write GPIO.output(8,True).

**Sleep Function (Delay Function)**

**Syntax:** time.sleep(y), where y is time in seconds

sleep() method suspends execution for the given number of seconds. The parameter can also be floating point number. For sleep function to be usable, time library has to be imported. To make the program sleep for a second we write, time.sleep(1). To make it sleep for 1 minute, use time.sleep(60).

**Cleanup function**

**Syntax:** GPIO.cleanup()

Before exiting a program, it is a good idea to clean up the pin usage. Clean up function releases the hold on GPIO pins. Also if you are reusing the port, make sure to clean up before using it.

# **Example #1**

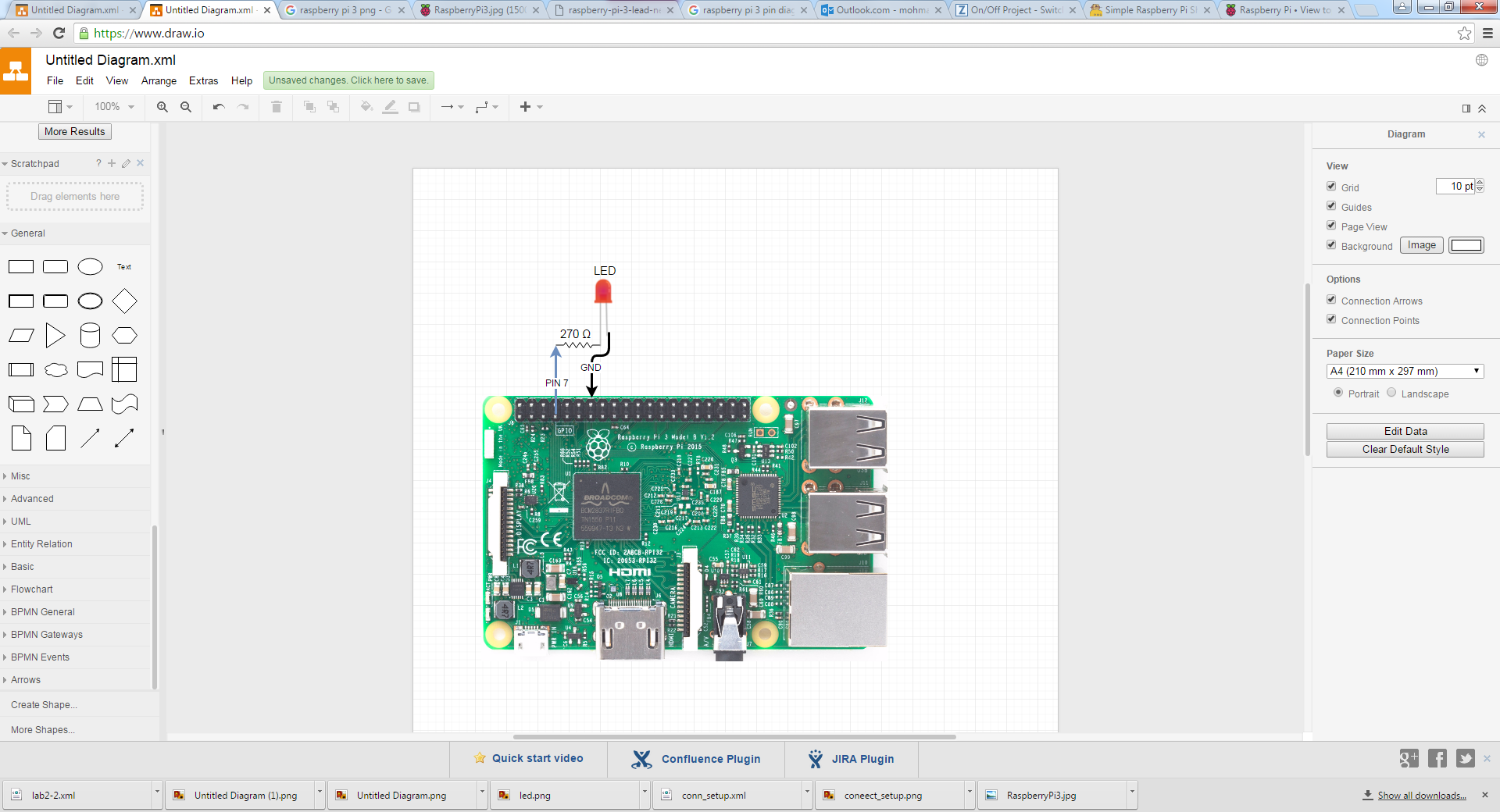


Figure 3. Connection Diagram

Write a python program to flash LEDs with an interval of 1 second

//Controlling LEDs with a Switch:

import RPi.GPIO as GPIO # a must to initialize input/ output ports

import time # to import time to implement sleep function

# also to implement other timer functions

GPIO.setwarnings(False) # to ignore warnings

GPIO.setmode(GPIO.BOARD) # to set board to BCM mode or Board mode

GPIO.setup(7,GPIO.OUT) # set pin 7 as an output (LEDs)

while True:

GPIO.output(7, GPIO.HIGH) # write to port 7

time.sleep(1) #sleep for 1s

GPIO.output(7, GPIO.LOW) # write to port 7

time.sleep(1) #sleep for 1s

# **Example #2**

C:\Users\b00042500\Downloads\Untitled Diagram (1).png

Figure 5. Connection Diagram

Write a python program to take input from switch and turn a LED light ON/OFF based on switch input.

//Controlling LEDs with a Switch:

import RPi.GPIO as GPIO # a must to initialize input output ports

import time # to import time to implement sleep function

# also to implement other timer functions

GPIO.setwarnings(False) # to ignore warnings

GPIO.setmode(GPIO.BOARD) # to set board to BCM mode or Board mode

GPIO.setup(7,GPIO.OUT) # set pin 7 as an output (LEDs)

GPIO.setup(16,GPIO.IN) #set pin 16 as input (Switches)

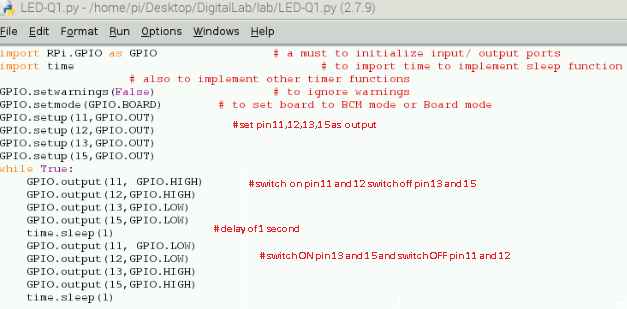
while True:

GPIO.output(7,GPIO.input(16)) # read from port 16 and write to port 7

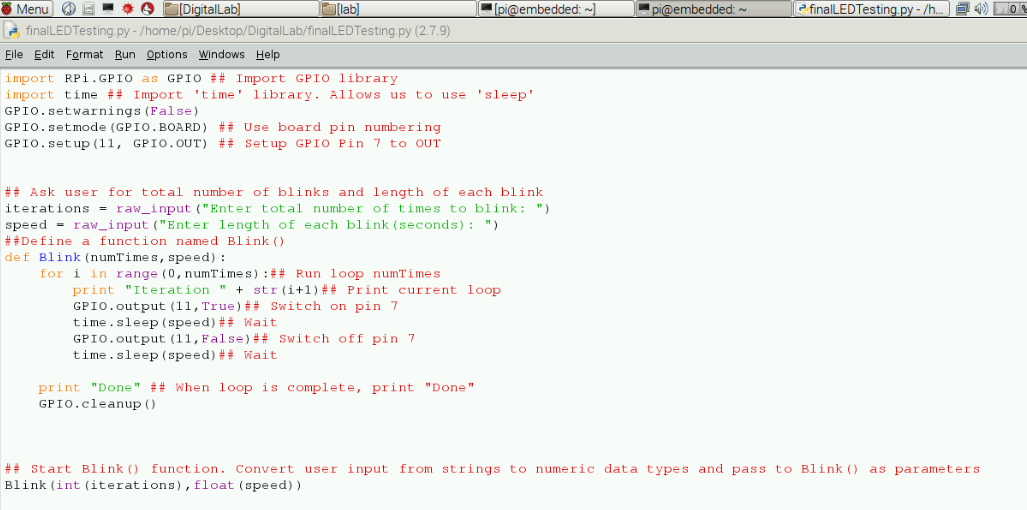
time.sleep(0.05) #sleep for 0.05

**Lab Questions**

* 1. Write a program in python to output the 4 LEDs alternatively. Switch ON LED 11,12 while LED 13,15 are OFF. Alternatively, switch ON LED 13,15 while LED 11,12 are OFF. Keep the delay for 1 second. Use the BOARD mode for accessing GPIOs

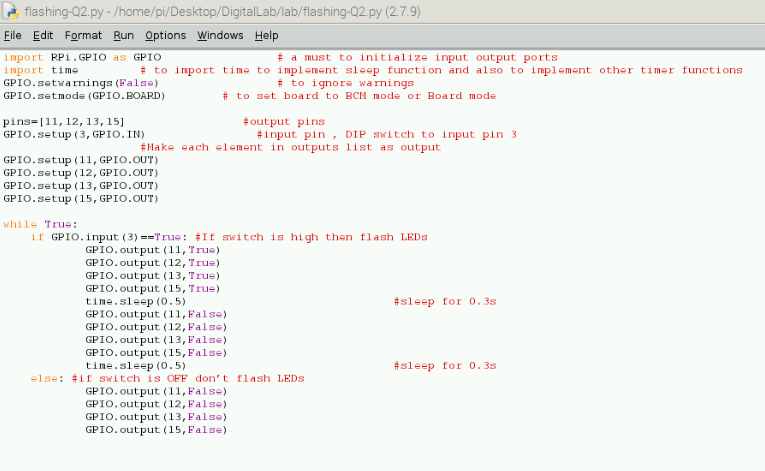


* 1. Write a program in python to blink LED 11, such that user is prompted to enter the number of times the LED blinks and also the length of each blink in seconds.



* 1. Write a python program to flash the 4 LEDs if an input connected to DIP switch number 3 is high. The program should turn off LEDs when the input is low.

**Answer:**



**OR : using a for loop:-**

import RPi.GPIO as GPIO # a must to initialize input output ports

import time # to import time to implement sleep function and also to implement other timer functions

GPIO.setwarnings(False) # to ignore warnings

GPIO.setmode(GPIO.BOARD) # to set board to BCM mode or Board mode

inputs=3 #input pin

outputs=[11,12,13,15] #output pins

GPIO.setup(inputs,GPIO.IN)

for i in outputs: #Make each element in outputs list as output

GPIO.setup(i,GPIO.OUT)

while True:

if GPIO.input(inputs)==True: #If switch is high then flash LEDs

for i in range(0,len(outputs)): #From 0 to length of array outputs

GPIO.output(outputs[i],True)

time.sleep(0.3) #sleep for 0.3s

GPIO.output(outputs[i],False)

time.sleep(0.3) #sleep for 0.3s

else: #if switch is OFF don’t flash LEDs

for i in range(0,len(outputs)): #From 0 to length of array outputs

GPIO.output(outputs[i],False)

* 1. Read input from 3 switches and update 3 LEDs based on the state of the switch.

Hints:

* + - 1. Use array/list to store pin numbers
      2. Use loops to quickly declare the input and output types
      3. Use **len(variable)** to get length of arrays, strings, etc.

***Syntax of for loop***: for i in range(0,len(variable))

**Answer:**

import RPi.GPIO as GPIO # a must to initialize input output ports

import time # to import time to implement sleep function and also to implement other timer functions

GPIO.setwarnings(False) # to ignore warnings

GPIO.setmode(GPIO.BOARD) # to set board to BCM mode or Board mode

inputs=[3,7,8] #input pins

outputs=[11,12,13] #output pins

for i in inputs: #Make each element in inputs list as input

GPIO.setup(i,GPIO.IN)

for i in outputs: #Make each element in outputs list as output

GPIO.setup(i,GPIO.OUT)

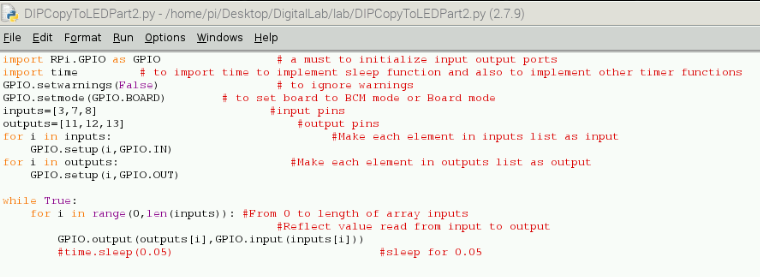
while True:

for i in range(0,len(inputs)): #From 0 to length of array inputs

#Reflect value read from input to output

GPIO.output(outputs[i],GPIO.input(inputs[i]))

time.sleep(0.05) #sleep for 0.05



**Programming in C:**

* 1. Write a program in C language to turn ON/OFF an LED for 0.5 seconds connected to PIN 11 (on BOARD mode)

Type nano blink.c from the terminal window and type the code shown below:

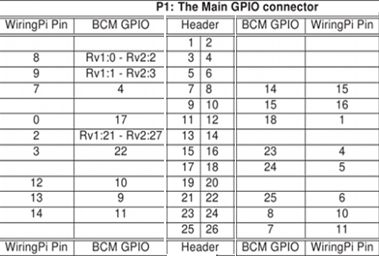
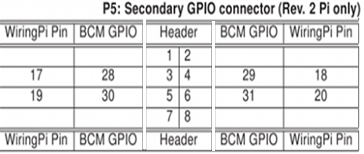


***Steps to follow:***

RPi.GPIO library is based on Python language and can be used only to access the GPIOs for a python script. To access the raspberry pi GPIOs through a C program, it is required to use a library in C language, i.e. Wiring Pi Library to access GPIOs. Following are the steps on executing installing and setting up Wiring Pi library:

1. First, we need to install this library by following the instructions as mentioned in **http://wiringpi.com/download-and-install/**
2. Once the library is installed, we need to get familiar with WiringPi API and functions:
3. ***Function # 1: wiringPiSetup (void) :*** This initializes wiringPi and assumes that the calling program is going to be using the wiringPi pin numbering scheme. So***wiringPi*** supports its own pin numbering scheme which should be followed. This is a simplified numbering scheme which provides a mapping from virtual pin numbers 0 through 16 to the real underlying Broadcom GPIO pin numbers. The table in next slide shows the mapping between the Raspberry BCM GPIO Pin and the WiringPi Pin.
4. ***Function # 2: void digitalWrite (int pin, int value) :*** Writes the value HIGH or LOW (1 or 0) to the given pin which must have been previously set as an output
5. ***Function # 3 int digitalRead (int pin) :*** This function returns the value read at the given pin. It will be HIGH or LOW (1 or 0) depending on the logic level at the pin.

To access GPIOs using Wiring Pi, it is important to note that Wiring Pi has its own virtual pin numbers which have to be mapped to the GPIOs BCM mode.



*Table 2: Mapping for Raspberry Pi board 2 revision and more*

*Table 1: Mapping for Raspberry Pi board 1*

**For example:** To write to Pin Number 11 (PIN 11 by BOARD mode) for LED, PIN 11 by Board mode corresponds to BCM\_GPIO 17 (by BCM mode) from figure 1 GPIO layout . This BCM\_GPIO 17 corresponds to wiringPin number 0 from table 1. So we say digitalWrite(0,1); to Turn ON the LED. To turn it off, we say digitalWrite(0,0).

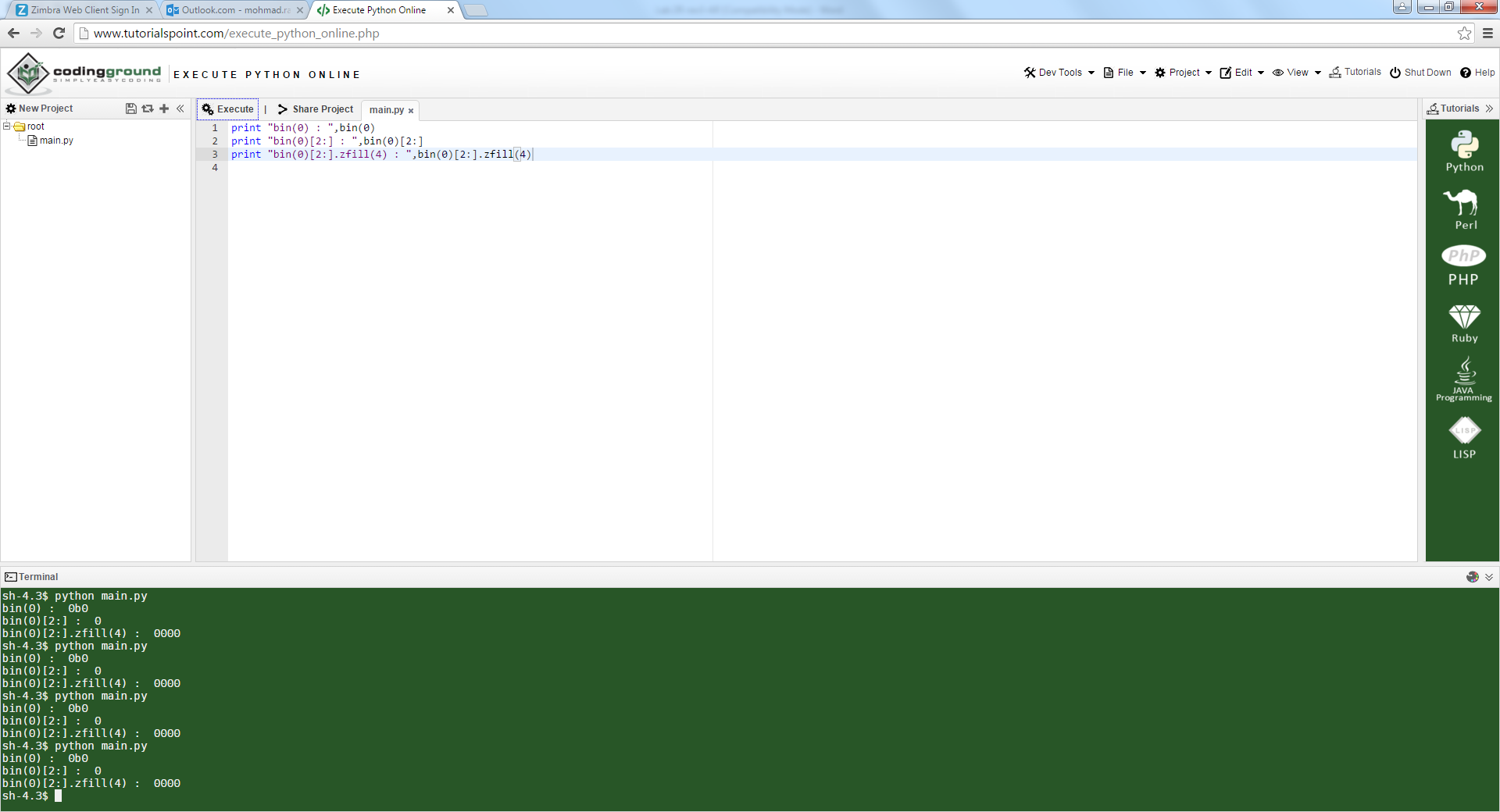
Next in the terminal window; type the following commands:

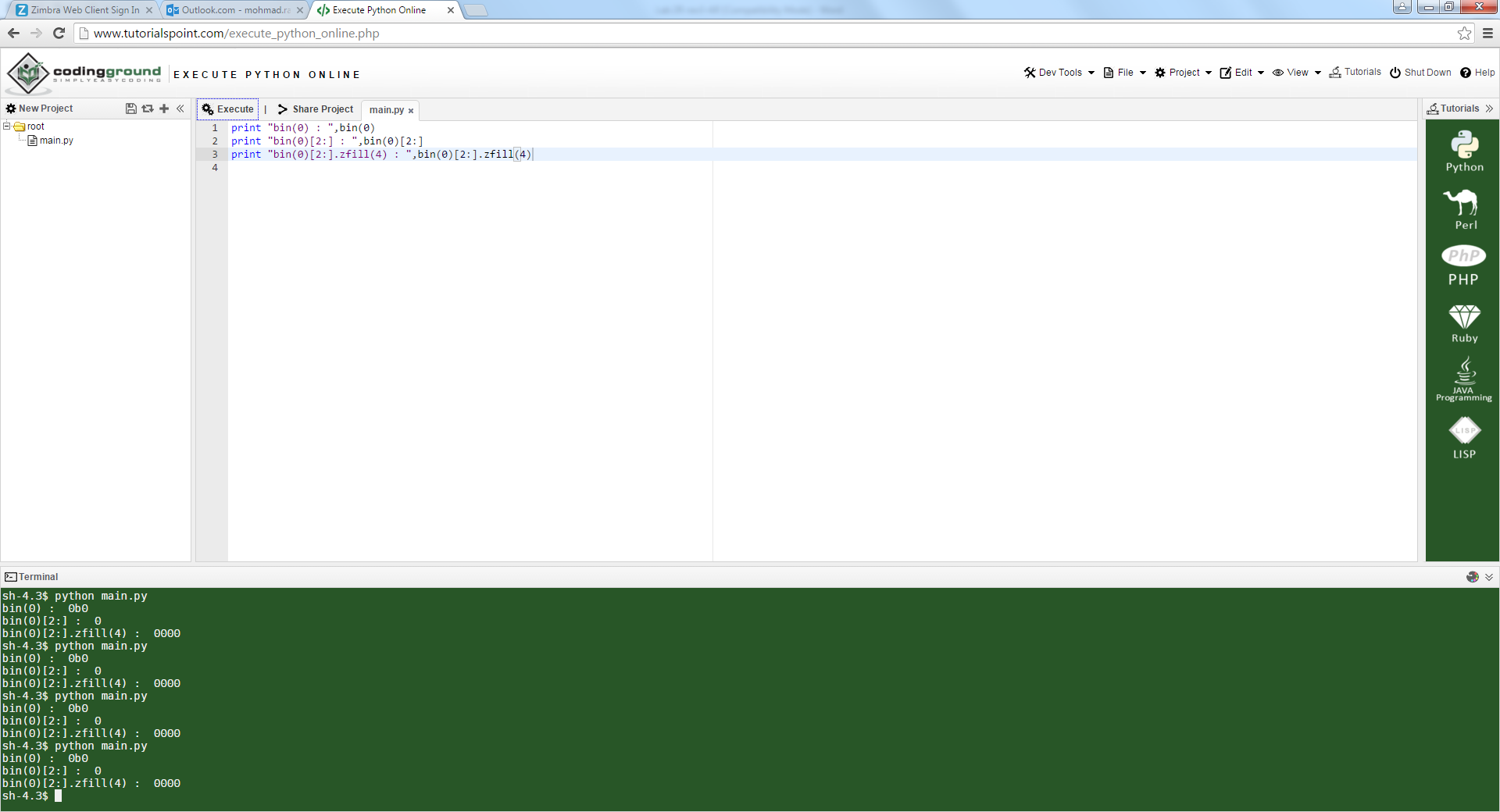
1. **nano blink.c** // this will open the text file editor to write the C code
2. **gcc blink.c –o blinkLED –lwiringpi** // this command will compile the blink.c file and create an executable file called blinkLED. The -o blinkLED part tells the compiler to take the source file (blink.c) and output another file (blinkLED) that we can make executable. The -lwiring command is used to run the executable file blinkLED for wiringPi. No need of chmod here as –lwiringPi does that for us. chmod in C is used to change the permissions of the file and make it executable.

**Homework Assignment:**

* 1. Implement a BCD counter using LEDs.

Hints:





* + - 1. Use bin(x) to get binary representation of a number
      2. bin(x) gives value as 0bx, to remove 0b from result use bin(x)[2:]. This means make string starting from index 2 of the result given by bin(x)
      3. To get a constant representation of 4 bits, use zfill(4), so to get to 4 bit binary representation of a number use bin(x)[2:].zfill(4)

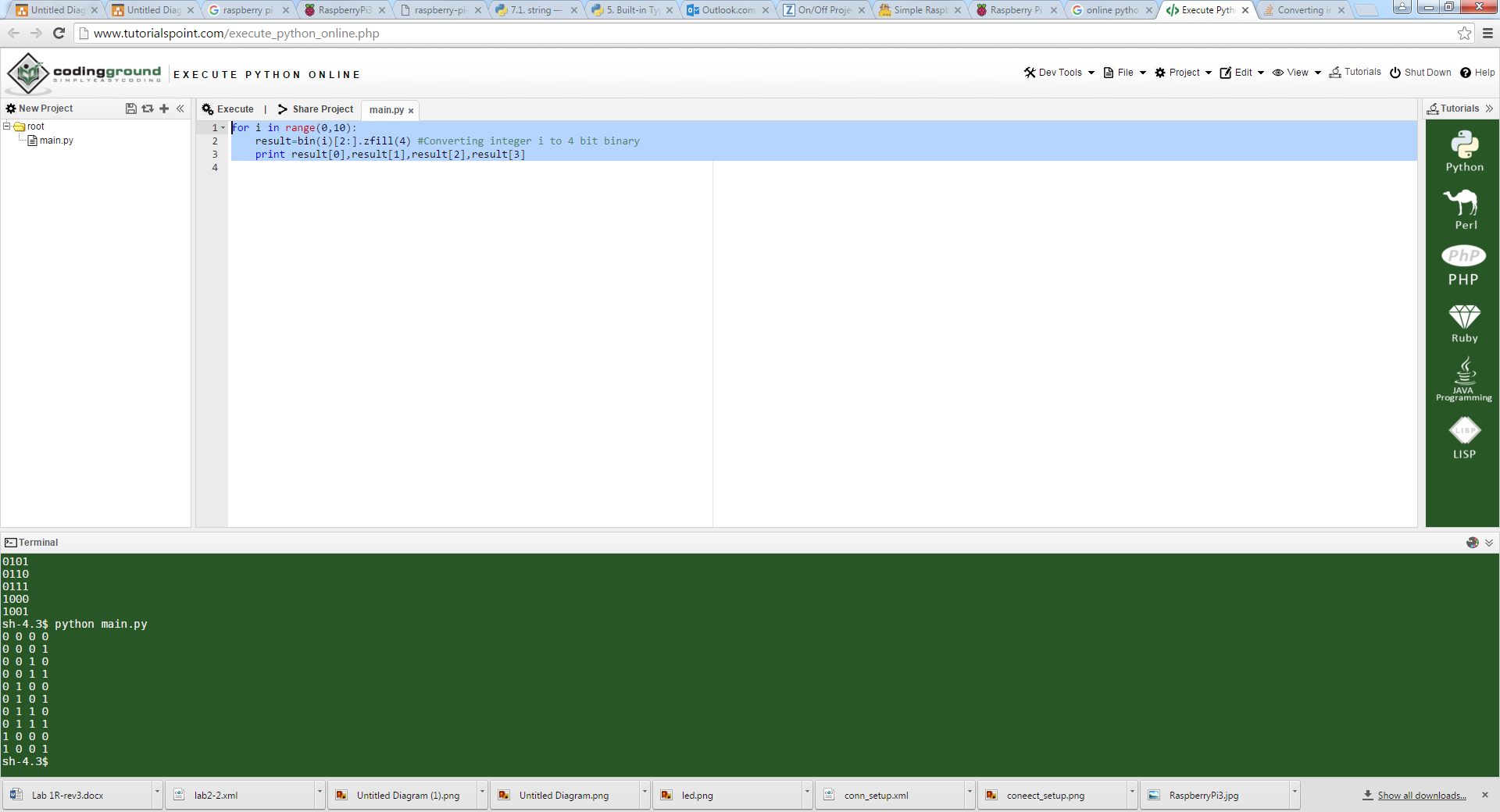
Code:

for i in range(0,10):

result=bin(i)[2:].zfill(4) #Converting integer i to 4 bit binary

print result[0],result[1],result[2],result[3]

Result:



**Answer:**

import RPi.GPIO as GPIO # a must to initialize input output ports

import time # to import time to implement sleep function and also to implement other timer functions

GPIO.setwarnings(False) # to ignore warnings

GPIO.setmode(GPIO.BOARD) # to set board to BCM mode or Board mode

outputs=[3,5,7,8] #output pins

bits=len(outputs) #This will make bits=4

for i in outputs: #Make each element in outputs list as output

GPIO.setup(i,GPIO.OUT) # set pin 7 as an output (LEDs)

while True:

for i in range(0,10): #Means from 0 to 9

result=bin(i)[2:].zfill(bits) #Converting integer i to 4 bit binary

#bin(0) by default gives as 0b0

#We don't need first two characters, so we use [2:], saying start from character 2

#Also we need 4 bit representation, so we fill unused places with 0s via zfill

for j in range(0,bits): #Writing to output pins in outputs list from 3 to 8

GPIO.output(outputs[j],int(result[j]))