

ECE40862: Software for Embedded Systems

Fall 2020

Lab 1 – Introduction to Python and – Digital Input and Output using GPIOs

To be done individually; Due by 11:59pm, Wednesday, September 30, 2020.

1. Overview

This assignment will familiarize you with Python programming language and the **Thonny Python IDE editor**, which you will use throughout the semester. Python is an easy-to-learn but powerful programming language and it is the fastest growing language for embedded computing. In this course, you will be using **MicroPython** to program **ESP32 board**. MicroPython is a lean and efficient implementation of Python3 programming language. Before working with MicroPython on your board, acquaint yourself with Python basics.

2. Getting started with Python and MicroPython

The following sections describe various software installation procedures.

2.1. Installing Thonny IDE on your own PC

Thonny is a Python IDE meant for learning programming. You can find more details about this IDE on <https://thonny.org> and <https://github.com/thonny/thonny/wiki>. Installation instructions for **Windows, Linux, Mac OS** are provided in this webpage: <https://randomnerdtutorials.com/getting-started-thonny-micropython-python-ide-esp32-esp8266/>. You might get a *Windows Smart Screen Filter* warning when trying to install Thonny IDE on **Windows**, you can safely ignore it and go ahead with the installation. More details can be found at <https://github.com/thonny/thonny/wiki/Windows>. **Install LATEST VERSION 3.2.7**. By default, Thonny comes with Python 3.7 built in, so, separate installation of Python is **NOT** necessary. However, if you want to use a separate Python installation, you can find Python installation instructions for different OS on <https://realpython.com/installing-python/>.

2.2. Install USB Driver on your own PC (Windows and Mac only)

Download and install **CP210x USB to UART Bridge VCP Drivers** so that your machine identifies when you connect ESP32 board to your PC. You can download files here: <https://www.silabs.com/products/development-tools/software/usb-to-uart-bridge-vcp-drivers>.

Linux machines usually do not need this driver, however, if your Linux machine is unable to detect the board, you might want to install Linux version of this driver.

2.3. Flashing MicroPython on ESP32 (using Thonny)

The latest version of Thonny (3.2.7) comes preinstalled with the package ‘esptool’, which is a Python-based, open source, platform independent, utility to communicate with the ROM bootloader in ESP32. **Esptool** can be used to load MicroPython on your ESP32 board.

2.3.1. Acquiring MicroPython Firmware

Download latest **ESP32 firmware** from <http://micropython.org/download>. As of today, the latest available firmware is ‘[esp32-idf3-20200902-v1.13.bin](#)’ (you can click this link directly to download the firmware).

2.3.2. Thonny Setup and Flashing

- Plug in your ESP32 board to your computer using a microUSB cable, launch Thonny and navigate to “Run > Select Interpreter”.
- Under the “Which interpreter or device should Thonny use for running your code?” label, select “MicroPython (ESP32)”.
- Select the COM port to which your ESP32 is connected under “Port” (see Figure 1).

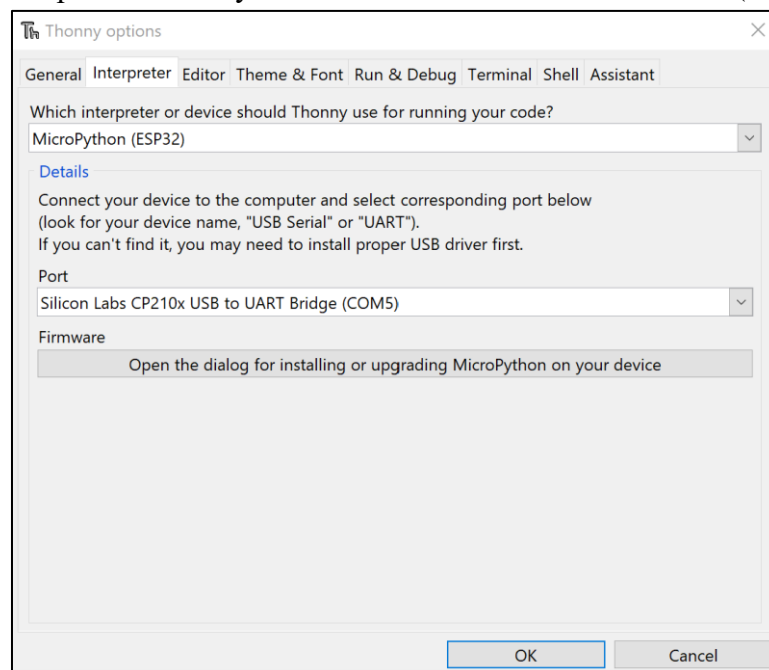


Figure 1. Set the interpreter and port

- After setting the interpreter and the port, click on the button: “Open the dialog for installing or upgrading MicroPython on your device”. Select your device’s port again and browse for the MicroPython firmware that you downloaded earlier for the “Firmware” field. **If you are putting MicroPython on your board for the first time**, then make sure that “Erase flash before installing” is checked, then click “Install”, as shown in Figure 2.

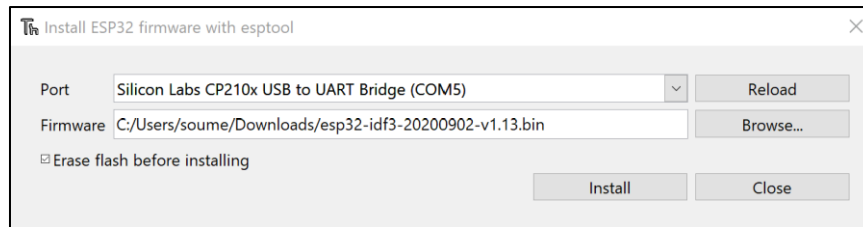


Figure 2. Select the port and downloaded MicroPython firmware

Thonny will now erase any existing firmware, and then flash the new MicroPython firmware on your ESP32, as you can see in Figure 3 and 4. This may take a few minutes. After the firmware has finished installing, you may close all the open dialogs.

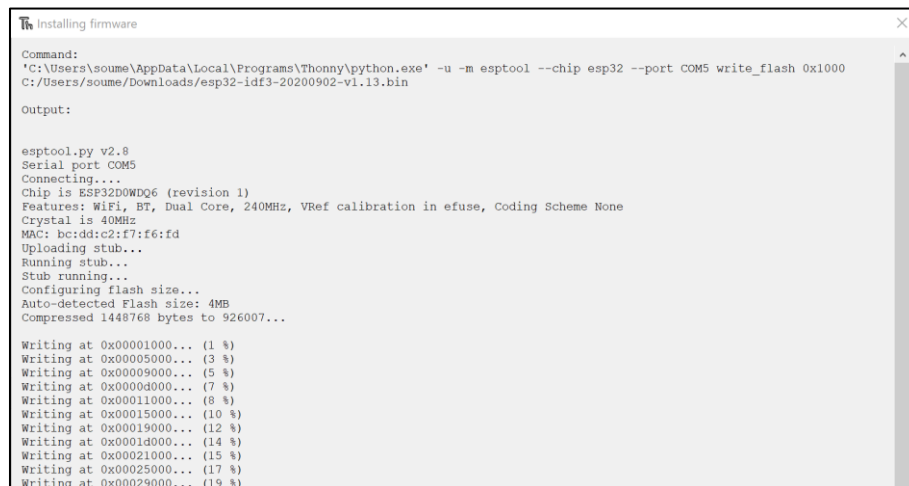


Figure 3. Flashing progress

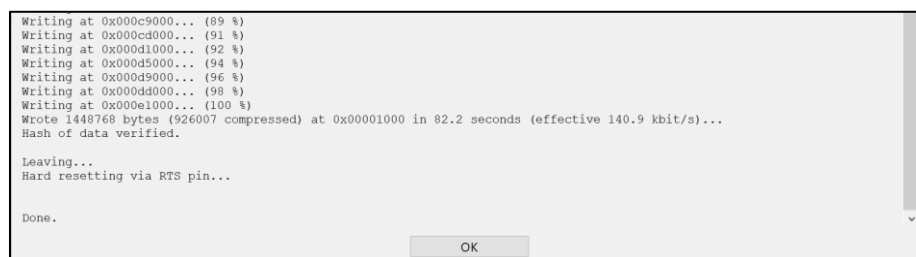


Figure 4. Thonny has completed flashing the firmware on the microcontroller.

- At the bottom of the Thonny window in the tab labeled “Shell”, you should see text that resembles Figure 5.

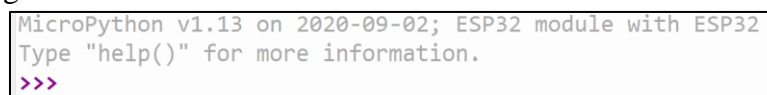


Figure 5. MicroPython REPL ready for input

Your ESP32 board is now ready for programming.

NOTE: You should be able to flash MicroPython easily using the steps mentioned above. If you are unsuccessful for any reason, you can follow the alternative method outlined in the next section.

2.4. Alternative Method to flash MicroPython

2.4.1. Installing esptool on your PC

You can read further details on esptool here: <https://github.com/espressif/esptool>. You can install esptool using any one of the following methods:

1. Using python package installer, pip:

```
>> pip3 install esptool
```

2. Using Thonny's package manager:

To access the package manager, open Thonny and go to the menu bar and select “Tools > Manage Packages”... This should pop open a new window with a search field. Type ‘esptool’ into that field and click the “Find package from PyPI” button. Go ahead and click “Install” to install this package. You will see a small window pop up showing the system’s logs while it installs the package. After the installation is finished, the plug-ins window can be closed. Then close and reopen Thonny. Now, you are ready to use esptool.

NOTE: You can use this method in future labs to install any additional packages.

2.4.2. Flashing MicroPython on ESP32 Board

Program your board using the **esptool.py** program. If you are putting MicroPython on your board (identified as `/dev/ttyUSBx*`, `x=0/1/2`) for the first time, then erase the entire flash using:

```
>> esptool.py --chip esp32 --port /dev/ttyUSB0 erase_flash
```

Then program the firmware starting at address **0x1000** using:

```
>> esptool.py --chip esp32 --port /dev/ttyUSB0 --baud 460800 write_flash -z 0x1000 <path to micropython .bin>
```

3. Python programming exercises

You **DO NOT** need to use ESP32 for this part. You can use the default python interpreter.

3.1. Character Input (Upload as program1.py)

Write a program that asks the user to enter name and age. Prints out a message that tells the year the user will turn 100 years old.

Hint: Use **input** command to get user input

```
>>> python program1.py
What is your name? Xavier
How old are you? 25
Xavier will be 100 years old in the year 2094
```

3.2. Lists and Conditional Statements (Upload as program2.py)

Write a program to initiate a list of numbers, print the list, and ask the user for a number and return a list that contains only elements from the original list that are smaller than the number given by the user.

```
>>> python program2.py
a = [1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
Enter number: 25
The new list is [1, 1, 2, 3, 5, 8, 13, 21]
```

3.3. Loops

3.3.1. While loop: (Upload as program3a.py)

Write a program to get Fibonacci series between 0 to a **user input number** using **while** loop

```
>>> python program3a.py
How many Fibonacci numbers would you like to generate? 10
The Fibonacci Sequence is: 1, 1, 2, 3, 5, 8, 13, 21, 34, 55
```

3.3.2. For loop: (Upload as program3b.py)

Write a program that generates a random number (0-10) and asks the user to guess it within three chances. If user guesses correctly, print 'You win!', otherwise print 'You lose!'

```
>>> python program3b.py
Enter your guess:3
Enter your guess:2
Enter your guess:10
You win!
```

3.4. Dictionary (Upload as program4.py)

Write a program to create a dictionary of names and birthdays. Upon execution, it should ask the user to enter a name, and return the birthday of that person back to them.

```
>>> python program4.py
Welcome to the birthday dictionary. We know the birthdays of:
Albert Einstein
Benjamin Franklin
Ada Lovelace
Who's birthday do you want to look up?
Benjamin Franklin
Benjamin Franklin's birthday is 01/17/1706.
```

3.5. Class and Functions (Upload as program5.py)

Write a program to create a Python class to find a pair of elements (indices of the two numbers) from a given list of numbers whose sum equals a specific target number.

Use this list in your program: [10,20,10,40,50,60,70]

Hint: You might create dictionaries of array index as *keys* and array item as *values* as you scan through the array. Use conditional statements, loops, etc.

```
>>> python program5.py
What is your target number? 60
index1=1, index2=3
```

4. Digital input/output using GPIOs

This section explains how to get started with the **Adafruit HUZZAH32 - ESP32 Feather** board and how to program the board using MicroPython and Thonny IDE. The goal of this experiment is to walk you through the *embedded software development* flow for the ESP32, where you will learn how to use microcontroller I/O, the inputs and outputs, to sense and actuate devices in the world outside the board.

The assignment is subdivided into two sections. In the first section, you will learn to implement a simple LED blink program that will give you a high-level idea of how you can accomplish the remaining labs. In the second section, you will interface two **LEDs** (Red and Green) and 2 **Push Button Switches** with the board and implement a simple pattern function using them.

The ESP32-WROOM-32 module on the Feather Board has 38 pins, many of which are GPIO (general-purpose input/output) pins. However, the feather board has only 28 pins and has its own adhoc pin numbering (marked e.g. A0, A1...) with many peripheral pins (like TX, RX, SCL, SDA...) indicated on the board itself. You need to explore the [ESP32-WROOM-32 datasheet](#) and [Feather board manuals](#) and [Schematics](#) to figure out the mapping between physical chip pins and board logical pins, and interface LEDs and Switches with the board pins.

4.1. Programming ESP32 using Thonny IDE

Before proceeding further, you should have completed installing any necessary software (explained in Section 2) and flashed MicroPython on the ESP32 board. The implementation procedure explained in this section demonstrates a typical procedure to run any program on the board.

4.2. Connect the Board to PC

Connect the board to your PC using the Micro USB cable (just plug into the jack) and the Feather will regulate the 5V USB down to 3.3V which will power the board. In addition, you can also connect the Lithium ion polymer (Lipo/Lipoly) battery to the JST jack (if you have ordered it), which will get continuously charged by the USB Power (OPTIONAL).

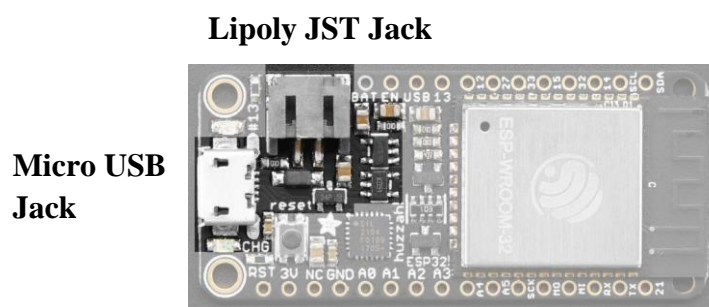


Figure 6. Board Power and PC Connections

You can get more information on starting MicroPython on your ESP32 board on this webpage: <http://docs.micropython.org/en/latest/esp32/tutorial/intro.html#esp32-intro>.

4.3. Testing Thonny IDE Installation

Important: Before testing the installation, your ESP32 board needs to be flashed with **MicroPython firmware**. Connect the board to your PC and open Thonny IDE. To test the installation, you need to tell Thonny that you want to run MicroPython Interpreter and select the board you are using.

1. Go to **Tools > Options** and select the **Interpreter** tab. Select **MicroPython (ESP32)**.
2. Select the device serial port: **Silicon Labs CP210x USB to UART Bridge (COMPort)**

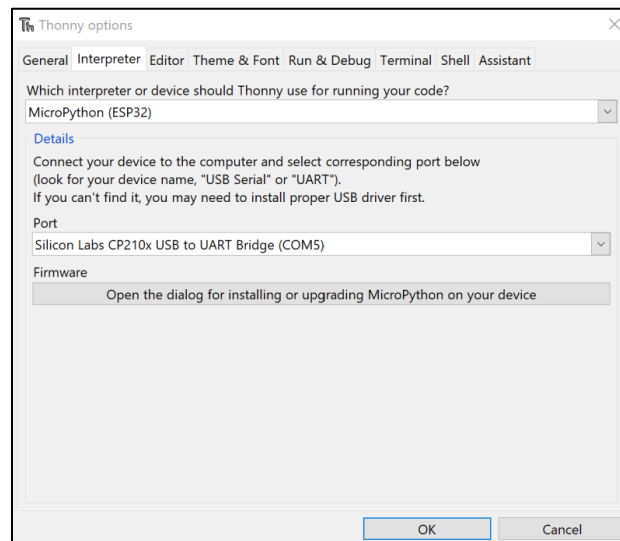


Figure 7. Thonny Interpreter and Port selection

Thonny IDE should now be connected to your board. The board will start and run the files **'boot.py'** and **'main.py'** (if any) automatically and provide you a MicroPython REPL shell. A read-eval-print-loop (REPL), also termed interactive top-level or language shell, is a simple, interactive computer programming environment that takes single user inputs (i.e., single expressions), evaluates (executes) them, and returns the result to the user, shown in Figure 8.

```
rst:0x1 (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
config: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:2
load:0x3fff0018,len:4
load:0x3fff001c,len:5008
ho 0 tail 12 room 4
load:0x40078000,len:10600
ho 0 tail 12 room 4
load:0x40080400,len:5684
entry 0x400806bc
I (538) cpu_start: Pro cpu up.
I (539) cpu_start: Application information:
I (539) cpu_start: Compile time: Sep 2 2020 03:00:08
I (542) cpu_start: ELF file SHA256: 0000000000000000...
I (548) cpu_start: ESP-IDF: v3.3.2
I (553) cpu_start: Starting app cpu, entry point is 0x40082f30
I (0) cpu_start: App cpu up.
I (563) heap_init: Initializing. RAM available for dynamic allocation:
I (570) heap_init: At 3FFAFF10 len 000000F0 (0 KiB): DRAM
I (576) heap_init: At 3FFB6388 len 00001C78 (7 KiB): DRAM
I (582) heap_init: At 3FFB9A20 len 00004108 (16 KiB): DRAM
I (588) heap_init: At 3FFBDB5C len 00000004 (0 KiB): DRAM
I (594) heap_init: At 3FFCA9E8 len 00015618 (85 KiB): DRAM
I (601) heap_init: At 3FFE0440 len 00003AE0 (14 KiB): D/IRAM
I (607) heap_init: At 3FFE4350 len 0001BCB0 (111 KiB): D/IRAM
I (613) heap_init: At 4009DE28 len 000021D8 (8 KiB): IRAM
I (619) cpu_start: Pro cpu start user code
I (303) cpu_start: Starting scheduler on PRO CPU.
I (0) cpu_start: Starting scheduler on APP CPU.
COMPLETED BOOT
Hello from main
MicroPython v1.13 on 2020-09-02; ESP32 module with ESP32
Type "help()" for more information.
>>>
```

Figure 8. MicroPython startup on ESP32 in Thonny IDE

4.4. Using the REPL

Once you have a prompt, you can start experimenting! Anything you type at the prompt will be executed after you press the *Enter* key. MicroPython will run the code on the board that you enter and print the result (if there is one). If there is an error with the text that you enter, then an error message is printed. You can find more information on REPL at <http://docs.micropython.org/en/latest/esp8266/tutorial/repl.html> (MicroPython execution is similar in ESP8266 and ESP32).

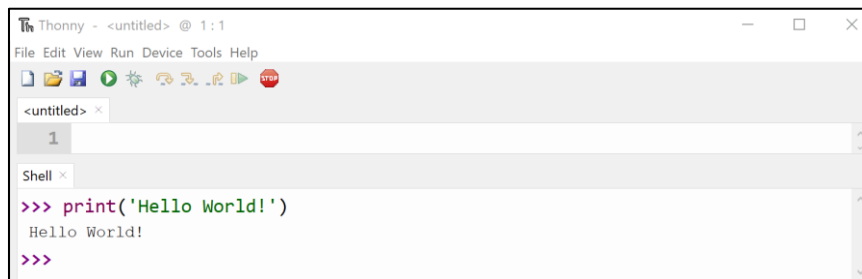


Figure 9. Experimenting with REPL

Type the command **help()** in the Shell and see if it responds back. If it responded back, everything is working fine.

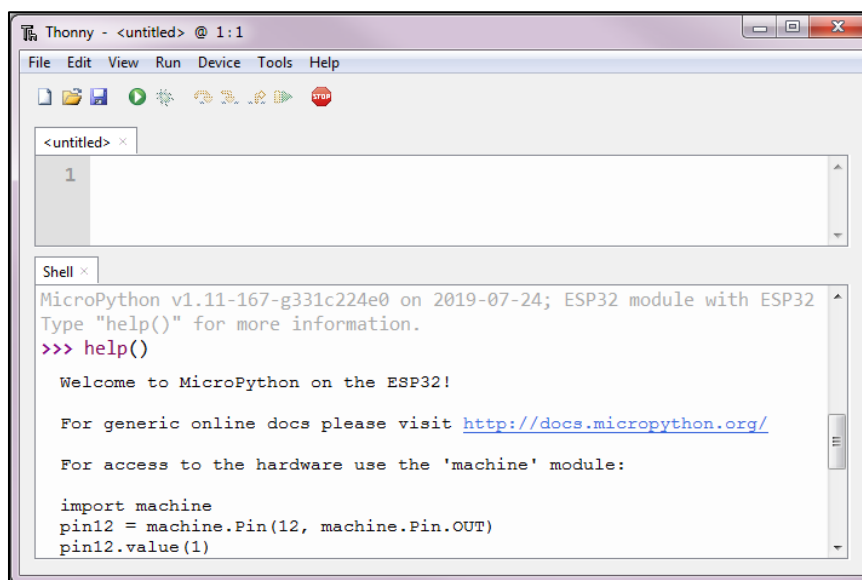


Figure 10. Testing the REPL

4.5. First Program on the ESP32

In the EDITOR of the Thonny IDE, write your first ***'Hello World'*** program and save it on the board as **main.py**. The IDE asks for location to save the file, **Select MicroPython device** to save the file on the flash memory on the board, as shown in Figure 11. Pressing **Ctrl+D** or selecting **Run > Send EOF/Soft Reboot in Thonny** will perform a soft reboot of the board and as mentioned earlier, it will run **main.py** and you will see the output on the MicroPython shell. You can also run the program **main.py** from the shell by calling the following command, which will run the code inside **main.py** and give you output on the shell, both of which are shown in

Figure 12. Similarly, any code you write for your labs, you can save them on the board and import your program module in MicroPython shell to run the code.

```
import main
```



Figure 11. Saving file on ESP32 board



Figure 12. Reboot and Running main.py

4.6. The internal filesystem

Since ESP32 Feather has 4MB flash memory, a default filesystem will be generated upon first boot, which uses the FAT format and is stored in the flash after the MicroPython firmware. You can open any file already stored inside the board using Thonny IDE, modify it and save it back on the board. More details on working with the internal filesystem are available at this webpage: <http://docs.micropython.org/en/latest/esp8266/tutorial/filesystem.html>.

5. ESP32 Programming Exercises

5.1. Simple Blink Program (main.py)

The example source code below blinks the Red LED on the ESP32 Feather board 5 times with the interval of roughly 0.5 seconds. Use this source code and save as **main.py** file on your board and perform a *soft reboot*. Use the Feather board **schematics** to figure out the pin X connected to the red LED on the board.

```
from machine import Pin
from time import sleep

# Onboard RED LED is connected to IO_X
# FIND OUT X FROM SCHEMATICS AND DATASHEET
# Create output pin on GPIO_X
led_board = Pin(X, Pin.OUT)

# Toggle LED 5 times
for i in range(10):
    # Change pin value from its current value, value can be 1/0
    led_board.value(not led_board.value())
    sleep(0.5)    # 0.5 seconds delay

print("Led blinked 5 times")
```

5.2. LED Pattern Display using Switches (pattern.py)

Now that you have gone through the warm-up program above, you can finally get to the actual problem statement for Lab 1. Implement a simple pattern function by interfacing two **push button switches** and two external **LEDs** (red and green) with the ESP32 Feather board that performs the operation given in Table 1. The **switches** are considered to be ON when they are pushed down and **OFF** when they are released.

Table 1. LED Pattern for Lab 1

Switch1	Switch2	Red LED	Green LED
OFF	OFF	OFF	OFF
OFF	ON	OFF	ON
ON	OFF	ON	OFF
ON	ON	OFF	OFF

In addition, you must determine the **number of times Switch1 and Switch2 are pressed** from the start of the program. If any one of the switches is **pressed 10 times**, the pattern in Table 1 should be stopped and the two **LEDs** should **start blinking alternatively**. This should continue until your program detects a **switch press** on the **other switch** (i.e., NOT the switch, which was pressed 10 times, pressing it again should not have any effect).

Sample Program Flow:

- The user starts *testing* the pattern function; the program starts turning the LEDs ON and OFF depending upon switch presses.

- Let us assume that throughout the *testing*, **Switch1** is pressed **9** times and **Switch2** is pressed **5** times. On the next **Switch1** press (the 10th press), the LEDs **start blinking alternatively**. The user presses **Switch2**, both LEDs turn **OFF** and the program displays a message on the MicroPython shell ‘**You have successfully implemented LAB1 DEMO!!!**’ and exits to the shell.

6. Submission

Make sure you follow these instructions precisely. Points will be deducted for any deviations. You need to turn in your code on Brightspace. Please create a directory named *username_lab1*, where username is your CAREER account login ID. Inside this directory, create two more directories, *part1* and *part2*.

- Put all the files created in section 3 (Python programming exercise) in *part1*.
- Put all the files created in section 5 in *part2*.

NOTE: These directories should contain only the source code files, i.e., no executables, no temporary files, etc.

- Zip the parent directory (i.e., *username_lab1*) and name it as *username_lab1.zip* and upload the .zip file to Brightspace.

REFERENCES

- [1] Getting started with MicroPython on the ESP32 <https://docs.micropython.org/en/latest/esp32/tutorial/intro.html>
- [2] ESP32 WROOM-32 Datasheet https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32_datasheet_en.pdf
- [3] ESP32 Technical Reference Manual https://www.espressif.com/sites/default/files/documentation/esp32_technical_reference_manual_en.pdf
- [4] Adafruit HUZZAH32 – ESP32 Feather Online Manual <https://learn.adafruit.com/adafruit-huzzah32-esp32-feather>
- [5] Adafruit ESP32 Feather Schematics https://cdn-learn.adafruit.com/assets/assets/000/041/630/original/feather_schem.png?1494449413
- [6] MicroPython GitHub <https://github.com/micropython/micropython>
- [7] REPL <https://docs.micropython.org/en/latest/esp8266/tutorial/repl.html>
- [8] Thonny IDE <https://thonny.org>