

DE1.3 Electronics 1

Lab 4A – Setting up MicroPython

Peter Cheung

Introduction

So far you have been using the signal generator running on the ESP32, the oscilloscope and the multimeter to investigate electronic circuits and learn about basic principles in electronics. From Lab 4 onwards, you will start writing code to control electronic components using the ESP32.

The signal generator application on the ESP32 was written in C++ under Visual Studio Code (VSC) intergrated development environment (IDE) with PlatformIO. The SIG_GEN application was flashed onto the Heltec ESP32 module before your parcel was sent. In that way, the signal generator is more or less ready for us after you added the rotary encoder for control.

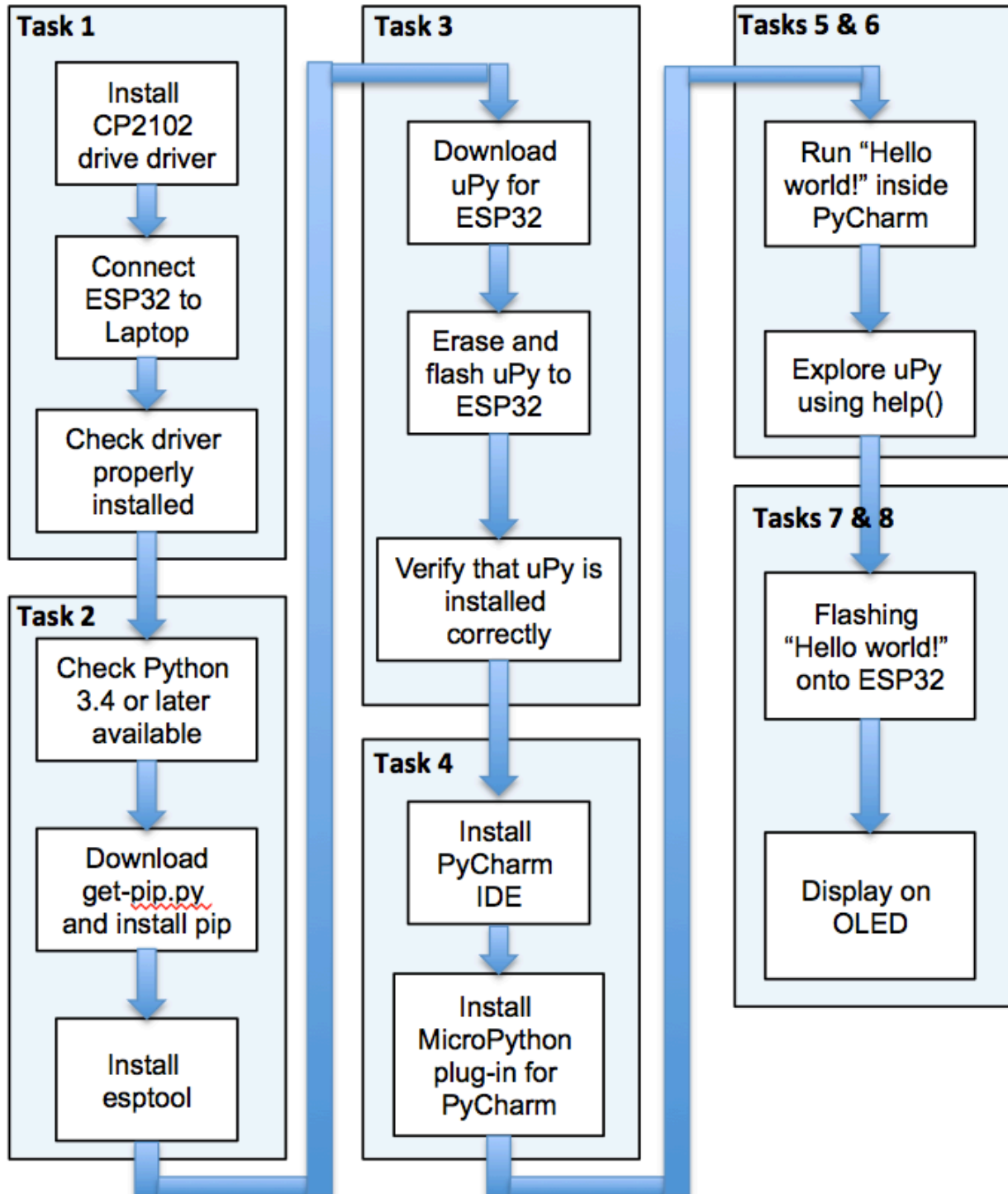
For the next part of the Home Lab, you will be programming the ESP32 with a version of Python, called **MicroPython** (or uPy). This choice is driven by three reasons: 1) you are already familiar with Python from Computing 1; 2) uPy is far easier to program and the code is shorter than using C++ or Sketch; 3) you are much more likely to write your own code than to download someone else's from the internet because uPy codes are not commonly available.

Lab 4 is in two parts. This document is for part A, which is mostly procedural. By the end of Lab 4B, you should be in a position start using the ESP32 and uPy. You will load the uPy intepreter program onto the ESP32 and overwrite the SIG_GEN code, and take control of the ESP32 module yourself. Although this instruction is quite long, the entire process should take around one hour if everything works perfectly.

IMPORTANT NOTE: Lab 4A requires you to have administrator's privilege on your laptop computer, which is required for installing various programs and device drivers. Further, read the instruction carefully and be patient – there are many steps and each step must be followed precisely, otherwise it won't work. However, the benefit is that you will learn how to install and set up a fairly complex environment – an experience that is beneficial in its own right. Once set up, you don't need to do it again for programming any EPS32 devices in the future.

Sequence of Tasks to set up MicroPython environment

A flowchart showing all steps required to set up the MicroPython environment for Lab 4.



Task 1 – Installing the CP2102 device driver

In Lab 1 to 3, you used the USB port on your laptop to provide power to the ESP32 module. From now on, you will also communicate with the ESP32 through the USB port on your laptop to flash a new program onto the ESP32 and to send or receive information to the ESP32. The mechanism relies on UART protocol of communication via the USB interface – this is known as the USB to UART bridge. For this to work, you need to install the CP2102¹ USB to UART Bridge VCP Driver (VCP = Virtual Communication Port). Go to Silicon Labs download page here:

<https://tinyurl.com/y5gl5fxr>

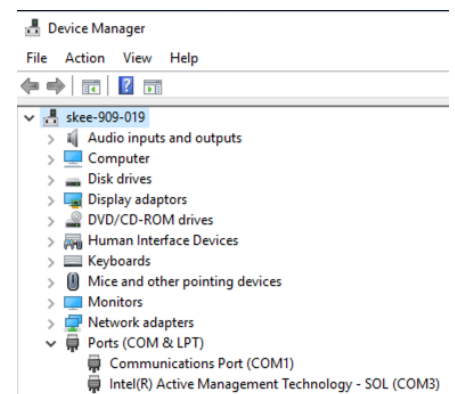
Follow the instruction and install the driver for your Windows 10 or Mac OSX laptop computer.

Next we need to check that the driver is installed properly. To do so, **you must plug your ESP32 module to the USB port of your computer with the supplied USB cable in the instrument bag.**


For Windows 10 PC




Once correctly installed, you may have to reboot your computer. Then plug in the EPS32 module. The yellow LED should be ON.

Run the Device Manager program and check under Ports (COM & LPT) tab. If the CP210x driver is successfully loaded, you should see a device under this name listed.



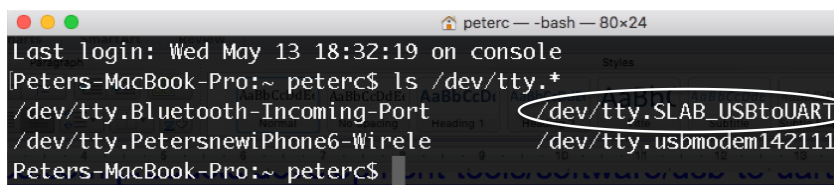
For Macbook

Run the Terminal program  **Terminal** (in Applications > Utilities folder):

 Macintosh HD >  Applications >  Utilities

Enter the following command: `ls /dev/tty.*` (“ls” is the unix command “list directory”, “/dev” is the folder that contains all device drivers used by your computer, “tty” stands for teletype, the first brand of computer terminals used decades ago, but the name stuck, “*” is just the wildcard character.)

If the driver is installed properly, you will see a file `/dev/tty.SLAB_USBtoUART` among other files in this directory.



¹ CP2102 is the chip on the ESP32 module made by Silicon Labs that links between USB and the UART interface on the ESP32 microcontroller.

Task 2 – Installing ESP tools to flash the ESP32 module

The ESP32 module in your Home Lab Kit is a **Heltec wifi 32 kit**, and it contains a microcontroller chip, the ESP32, made by **Espressif**. The same company also made its predecessor, the popular ESP8266, which can be found in many IoT devices such as smart lights and smart plugs used in homes.

The ESP32 module is preloaded with a program known as a “**bootloader**”, which allows users to download and flash their programs onto the ESP32 internal memory. Once done, power can be removed, and the program will remain. Such memory that retains its contents is called “**non-volatile**” memory. Your USB flash drive essentially is a type of non-volatile memory..

Espressif officially supports a special utility program to let users flash their ESP32 chips. This utility, “`esptool.py`”, is a Python program that allows you to erase the flash memory inside the ESP32 and over-write (or flash) your own program. Task 2 is to install this `esptool` utility on your computer.

Step 1: Check your Python installation

All subsequent steps require that you have Python installed on your laptop. You can check this by opening a Terminal window (on Mac) or a Console window (on Windows PC), and enter the command: `python3 --version` (“--” = two dashes) or `python --version`.

```
(Lab4) Peters-MacBook-Pro:Lab4 peterc$ python3 --version
Python 3.8.1
```

You need Python 3.4 or newer for the remaining of this term for Electronics 1.

Step 2: Installing the Python Install Package pip

Before you can install the `esptool` utility, you need to install a Python Install Package (`pip`) to help you to install the `esptool` and other Python packages. This may sound a bit tedious, but you will find `pip` is a utility that you cannot do without in many other occasions. Its installation is therefore well worth the effort.

The easiest way to install `pip` is to:

1. Download `get-pip.py` to a folder on your computer.
(Link: <https://bootstrap.pypa.io/get-pip.py>)
2. Open a Terminal or Console window, and navigate to the folder containing the file `get-pip.py` using the “`cd`” and “`ls`” commands

```
Peters-MacBook-Pro:~ peterc$ cd Downloads
Peters-MacBook-Pro:Downloads peterc$ ls get-pip.py
get-pip.py
```

3. Run the following command: `python3 get-pip.py` and `pip` will be automatically installed. Check the installation with command `pip --version`. (See my screen log below.)

```
Peters-MacBook-Pro:Downloads peterc$ python3 get-pip.py
Collecting pip
  Downloading pip-20.1.1-py2.py3-none-any.whl (1.5 MB)
    |#####| 1.5 MB 1.9 MB/s
Collecting wheel
  Using cached wheel-0.34.2-py2.py3-none-any.whl (26 kB)
Installing collected packages: pip, wheel
Attempting uninstall: pip
  Found existing installation: pip 19.2.3
  Uninstalling pip-19.2.3:
    Successfully uninstalled pip-19.2.3
Successfully installed pip-20.1.1 wheel-0.34.2
Peters-MacBook-Pro:Downloads peterc$ pip --version
pip 20.1.1 from /Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-packages/pip (python 3.8)
```

Step 3: Install esptool

Now in the command window enter the command: `pip install esptool`. (See my screen log below).

```
Peters-MacBook-Pro:Downloads peterc$ pip install esptool
Collecting esptool
  Downloading esptool-2.8.tar.gz (84 kB)
    |-----| 84 kB 1.9 MB/s with command pip -v . (See
Requirement already satisfied: pyserial>=3.0 in /Library/Frameworks/Python.Framework/Versio
ns/3.8/lib/python3.8/site-packages (from esptool) (3.4)
Collecting pyaes
  Downloading pyaes-1.6.1.tar.gz (28 kB)
    |-----| 28 kB 1.5 MB/s
Collecting ecdsa
  Downloading ecdsa-0.15-py2.py3-none-any.whl (100 kB)
    |-----| 100 kB 4.7 MB/s
Collecting six>=1.9.0
  Downloading six-1.15.0-py2.py3-none-any.whl (10 kB)
Building wheels for collected packages: esptool, pyaes
  Building wheel for esptool (setup.py) ... done
  Created wheel for esptool: filename=esptool-2.8-py3-none-any.whl size=142114 sha256=94079
6abb69409203fa9578d53fc01f83db97f9aacf952dbcc89e1b8b848b9bc
  Stored in directory: /Users/peterc/Library/Caches/pip/wheels/4a/4c/c7/aa45baeba596ab0e9d6
24a32e3f5680db5aed6773ae1ffa5ed
  Building wheel for pyaes (setup.py) ... done
  Created wheel for pyaes: filename=pyaes-1.6.1-py3-none-any.whl size=26345 sha256=e1cda6de
62adba7b1117b8076b816c1b4f222270c307374c01a393978417eda2
  Stored in directory: /Users/peterc/Library/Caches/pip/wheels/aa/ca/9c/8a3c00512585c703edc
457db81c066b9609d76758c74f72ac6
Successfully built esptool pyaes
Installing collected packages: pyaes, six, ecdsa, esptool
Successfully installed ecdsa-0.15 esptool-2.8 pyaes-1.6.1 six-1.15.0
```

Now you are ready to use the esptool utility.

Task 3: Erase and Flash MicroPython onto your ESP32

You are now ready to install the MicroPython (uPy) interpreter program onto your ESP32 module. Doing so will overwrite the SIG_GEN application which was preloaded on your ESP32. This is not a problem – you won't need the signal generator again for the rest of the term.

Step 1: Download the uPy binary file `esp32spiram.bin` from the course webpage.

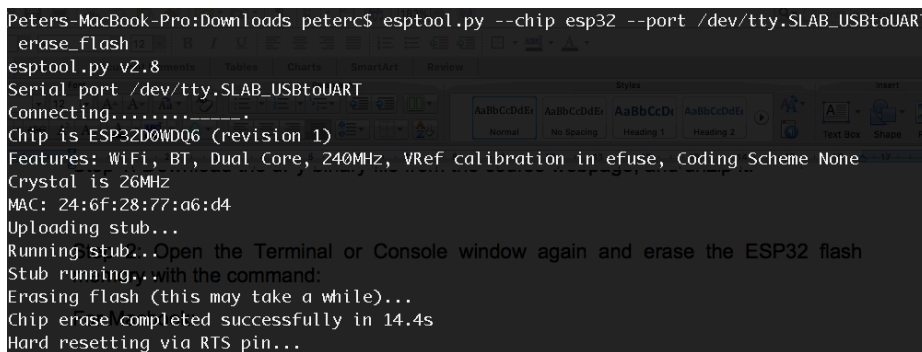
Step 2: Open the Terminal or Console window again and erase the ESP32 flash memory with the command:

For Macbook

```
esptool.py --chip esp32 --port /dev/tty.SLAB_USBtoUART erase_flash
```

For Windows PC (x is the COM PORT number)

```
esptool.py --chip esp32 --port COMx erase_flash
```



```
Peters-MacBook-Pro:Downloads peterc$ esptool.py --chip esp32 --port /dev/tty.SLAB_USBtoUART
erase_flash
esptool.py v2.8
Serial port /dev/tty.SLAB_USBtoUART
Connecting.....
Chip is ESP32D0WDQ6 (revision 1)
Features: WiFi, BT, Dual Core, 240MHz, VRef calibration in efuse, Coding Scheme None
Crystal is 26MHz
MAC: 24:6F:28:77:a6:d4
Uploading stub...
Running stub...Open the Terminal or Console window again and erase the ESP32 flash
Stub running..with the command:
Erasing flash (this may take a while)...
Chip erase completed successfully in 14.4s
Hard resetting via RTS pin...
```

Step 3: Flash uPy onto ESP32

Navigate to the folder containing the downloaded `esp32spiram.bin` file. Enter the command:

For Macbook

```
esptool.py --chip esp32 --port /dev/tty.SLAB_USBtoUART write_flash -z 0x1000
esp32spiram.bin
```

For Windows PC (x is the COM PORT number)

```
esptool.py --chip esp32 --port COMx write_flash -z 0x1000 esp32spiram.bin
```

(See screen log below.)

```

Peters-MacBook-Pro:~ peterc$ cd Downloads
Peters-MacBook-Pro:Downloads peterc$ esptool.py --chip esp32 --port /dev/tty.SLAB_USBtoUART
write_flash -z 0x1000 esp32spiram.bin
esptool.py v2.8
Serial port /dev/tty.SLAB_USBtoUART
Connecting.....
Chip is ESP32D0WDQ6 (revision 1)
Features: WiFi, BT, Dual Core, 240MHz, VRef calibration in efuse, Coding Scheme None
Crystal is 26MHz
MAC: 24:6f:28:77:a6:d4
Uploading stub...
Running stub...
Stub running...
Configuring flash size...
Auto-detected Flash size: 8MB
Flash params set to 0x0230
Compressed 1510800 bytes to 932839... /dev/ttyUSBx write_flash -z 0x1000 esp32spiram.bin
Wrote 1510800 bytes (932839 compressed) at 0x00001000 in 82.6 seconds (effective 146.3 kbit
/s)... (See screen log below.)
Hash of data verified.
Leaving...
Hard resetting via RTS pin...

```

Step 4: Verify that uPy is installed correctly

For Macbook

Run the Terminal application and enter the command (baudrate is 115200):

```
screen /dev/tty.SLAB_USBtoUART 115200
```

Now the Terminal screen will be connected to the ESP32 running uPy. Type ENTER a few times and you should see the familiar Python REPL >>>. If this fails, unplug and plug the USB cable and try again.

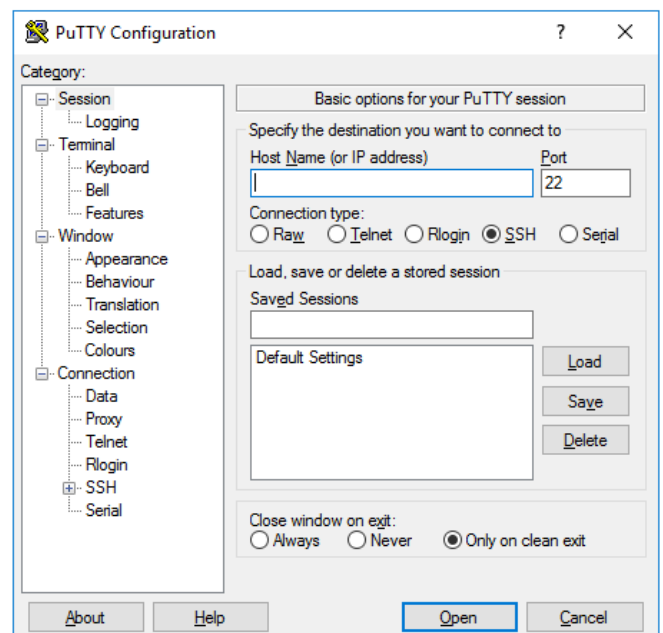
For Windows PC

Download the terminal program known as **PuTTY** from:

<https://www.ssh.com/ssh/putty/download>

Install PuTTY on your laptop and run this program. You will have to configure PuTTY by clicking the “serial” radial button and enter the COM PORT (e.g. COM4) connected to the ESP32. Also, choose the speed to be 115200. Run PuTTY.

You should now see the Python REPL >>> inside the PuTTY terminal window.



For both Mac and PC, you are running a terminal program to directly communicate with uPy running on the ESP32. You can enter any valid uPy program code and these will be executed immediately. For example, try:

```
print("Hello world!")
```

Task 4: Install PyCharm IDE for MicroPython

If you opened Terminal on MacBook or PuTTY on PC, you must close these application before you proceed to Task 4.

You can control the ESP32 using uPy in the interactive mode. However, this is not practical except for testing a few simple uPy scripts. For an substantive program, you need an Integrated Development Environment (IDE) with an Editor, a terminal program, and an easy way to flash new files onto the ESP32.

Step 1: Sign up and install PyCharm

The best platform to use is PyCharm and as a student, you can sign up to use their software for free. Go and visit:

<https://www.jetbrains.com/community/education/#students>

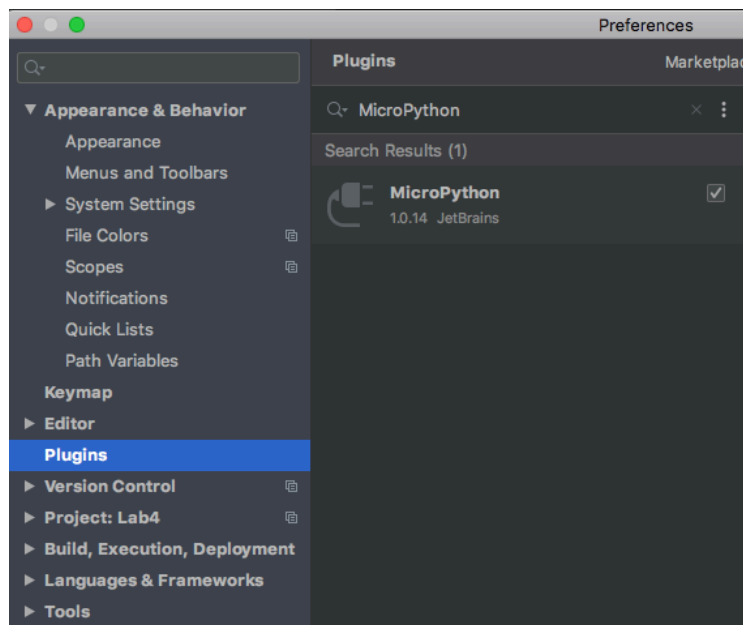
Sign up for a free student account. Download the full professional version. Alternatively, you can also just download PyCharm Community Edition (not full version). Install and run PyCharm.

Step 2: Create Project folder

Create a new project in a new folder, and call this “HomeLab4” or something suitable. You can leave all other settings as default. PyCharm will set up the new directory at your specified locaton with all the files that it needs.

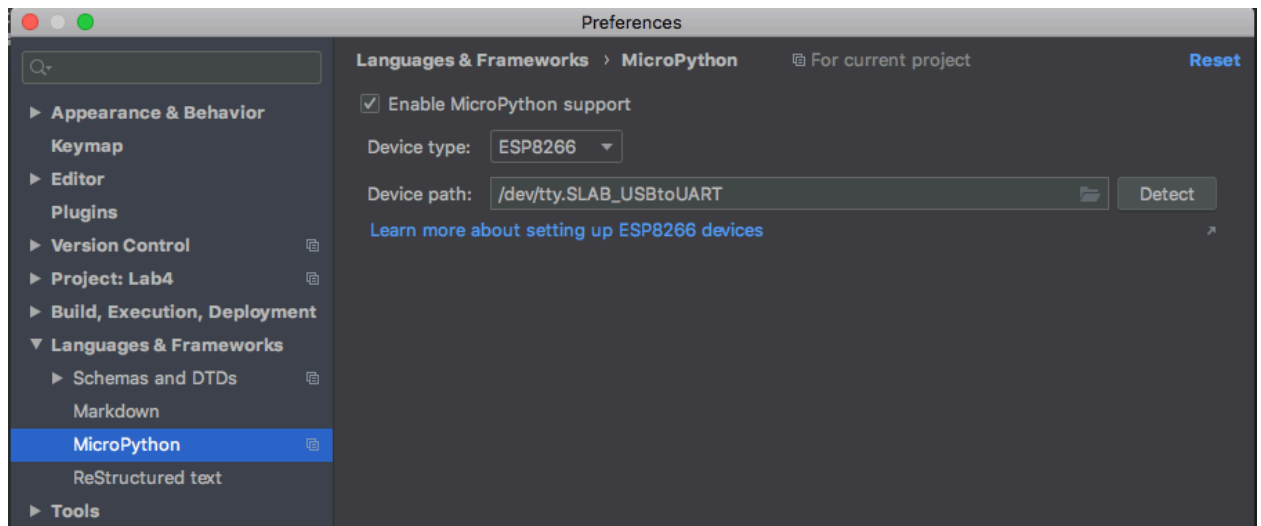
Step 3: Set up MicroPython plugin in PyCharm

- Under the pulldown menu PyCharm, open `Preferences`. A window will pop up. From the menu list on the left, select `Plugins`. Search from MicroPython and click install. (See screen log below.)



- Click on `Languages & Frameworks` in the left menu list, and select MicroPython.

- Select “Enable MicroPython Support”
- Under device type, select ESP8266 (there is no formal support for ESP32 and ESP8266, which is the predecessor to ESP32, is close enough).
- Enter the location of the device driver under Device Path. For Macbook, this would be `/dev/tty.SLAB_USBtoUART`. For PC, this would be COMx. Then click OK.

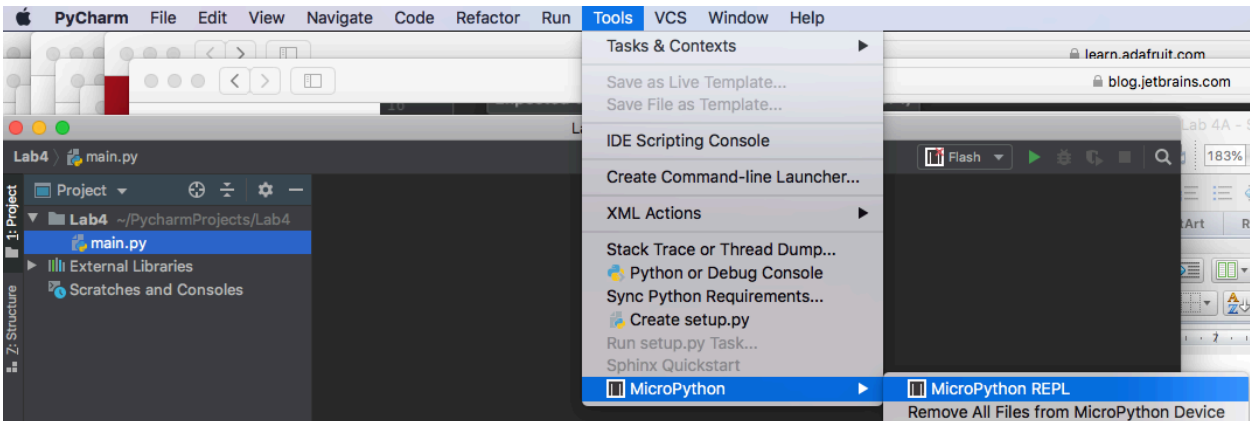


You are now ready to use PyCharm for uPy development.

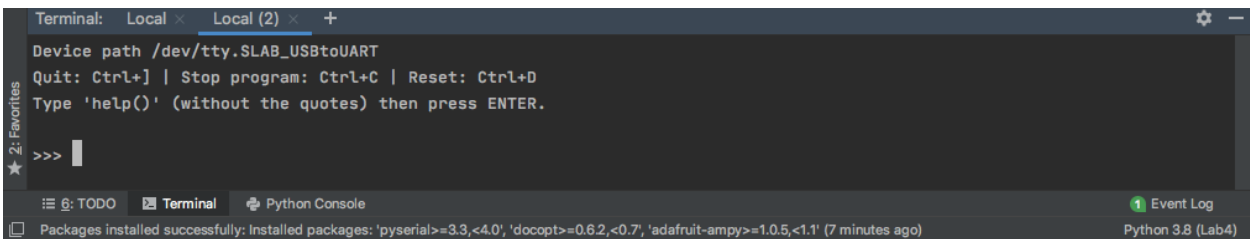
Task 5: “Hello world!” inside PyCharm environment

Create a new file under HomeLab4 workspace, and name the new file “main.py”. The first time you do this, there will be a message at the top of the window to say that esp8266 support is required but not available. On the right, there is a link labelled “install esp8266 support”. Click this link.

We are now in a position to run the simple “Hello World!” script inside PyCharm. To do that, go to the top “Tools” menu and select `MicroPython -> MicroPython REPL`.



A “Local” terminal window will appear in the lower part of PyCharm, and a Python REPL should appear as shown:



If this does not work, simply unplug and plug the USB cable on the ESP32 and try again.

The REPL you see here is NOT from your laptop. PyCharm is now communicating with the ESP32, much like what you did at the end of Task 3. The difference is that you do this within the integrated environment of PyCharm.

You can now type the uPy script:

```
print("Hello world!")
```

You should see the result in the terminal window.

Task 6: Explore what are included in MicroPython

Under the uPy REPL, enter:

```
>>> help('modules')
```

You will see the list of included library modules within uPy. These are all stored on the flash memory on the ESP32 chip.

You can also find out what is inside any of these modules. For example, you might want to explore one of the most important module – “machine”.

```
>>> import machine
>>> help(machine)
```

```
>>> help('modules')
__main__      gc                uasyncio/stream  upip_utarfile
_boot        inisetup         ubinascii        upysh
_owewire     machine          ubluetooth       urandom
_thread     math             ucollections     ure
_uasyncio   micropython     ucryptolib       urequests
_webrepl    neopixel        uctypes          uselect
apa106     network         uerrno           usocket
btree      ntptime         uhashlib         ussl
builtins   onewire         uhashlib         ustruct
cmath      sys             uheapq           utime
dht        uarray          uio               utimeq
ds18x20    uasyncio/___init___  ujson            uwebsocket
esp        uasyncio/core   umqtt/robust     uzlib
esp32     uasyncio/event  umqtt/simple     webrepl
flashbdev uasyncio/funcs  uos              webrepl_setup
framebuf  uasyncio/lock  upip             websocket_helper
Plus any modules on the filesystem
```

This will print a very long list of functions, classes and attributes defined in the “machine” module. One such item inside machine is the PWM Class.

You can explore the methods defined within the PWM Class by doing this:

```
>>> from machine import PWM
>>> help(PWM)
```

```
ADC -- <class 'ADC'>
DAC -- <class 'DAC'>
I2C -- <class 'I2C'>
PWM -- <class 'PWM'>
RTC -- <class 'RTC'>
SPI -- <class 'SoftSPI'>
UART -- <class 'UART'>
```

```
>>> from machine import PWM
>>> help(PWM)
object <class 'PWM'> is of type type
  init -- <function>
  deinit -- <function>
  freq -- <function>
  duty -- <function>
```

This immediately tells you that PWM has four methods (or functions), namely, `PWM.init()`, `PWM.deinit()`, `PWM.freq()` and `PWM.duty()`.

Description of MicroPython library modules can be found online under:

<https://docs.micropython.org/en/latest/library/index.html#>

However, the method shown above helps you to explore what are available and what the methods are called quickly.

Now explore uPy for yourself.

Task 7: Flashing the “Hello world!” program onto the ESP32

Having to type the uPy script each time you run a program is obviously not practical. It is far better if you store your program “permanently” on the ESP32 onchip flash memory. (Permanently here means the program will stay even if you switch power OFF, but it is in fact not permanent because you can erase and flash another program in its place later.)

What happens on boot up?

To do this you need to understand what happens when you switch the power ON or press the RESET switch (lower switch under the USB socket). This process is known as “bootstrapping” or “boot up” for short.

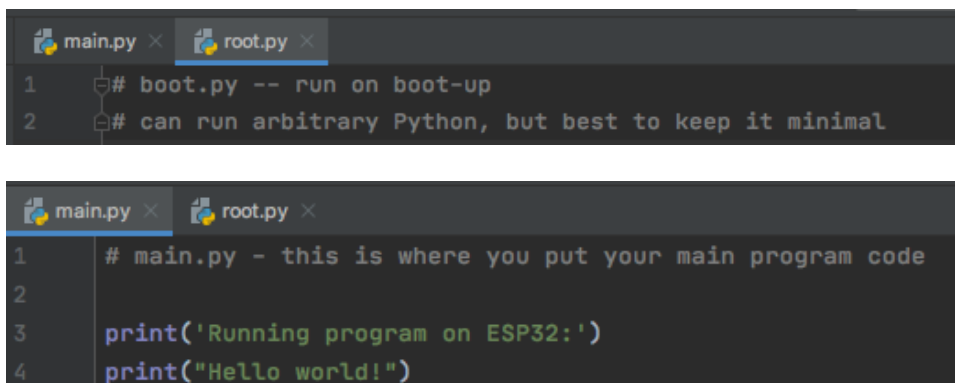
On powerup or reset, the system looks for a uPy script under the name “boot.py”. This normally contains some configuration scripts and import statements. It can empty (but must be exists).

After boot.py, ESP32 runs the uPy script “main.py”. This is where you should put your main program code.

To run uPy script automatically, you need to create boot.py and main.py using the editor in PyCharm, stored them locally on your computer disk (in the folder that you specify when creating the new project), and then flash these two files to ESP32.

Step 1: Create your uPy code files

Use the editor, create `boot.py` and `main.py` as shown below and save them in your HomeLab4 folder.

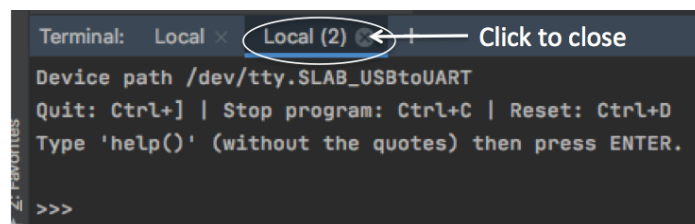


```
main.py x root.py x
1 # boot.py -- run on boot-up
2 # can run arbitrary Python, but best to keep it minimal

main.py x root.py x
1 # main.py - this is where you put your main program code
2
3 print('Running program on ESP32:')
4 print("Hello world!")
```

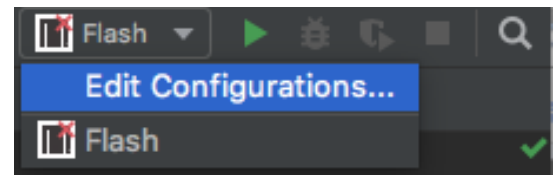
Step 2: Exit MicroPython Terminal window

If you had been communicating with ESP32 via the uPy REPL (as in Task 6), you MUST terminate this by closing that terminal window. This is because you need to use the USB connection, not for terminal communication, but to flash your program. There is only one link (USB to UART) and the same link is used for two purposes, but only one at a time. (See below.)

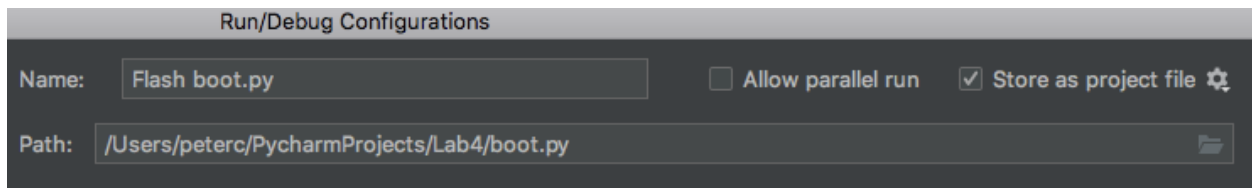


Step 3: Edit Configuration

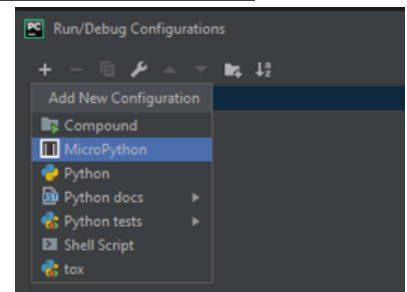
To prepare flashing `boot.py` and `main.py` to the ESP32, you need to click the “Add Configuration” button and select `MicroPython > Flash` to set up flashing the ESP32. There after, you can edit the configuration by selecting **ICON: Edit Configurations ...**:




A window will pop up, select the program file you want to flash and click OK.

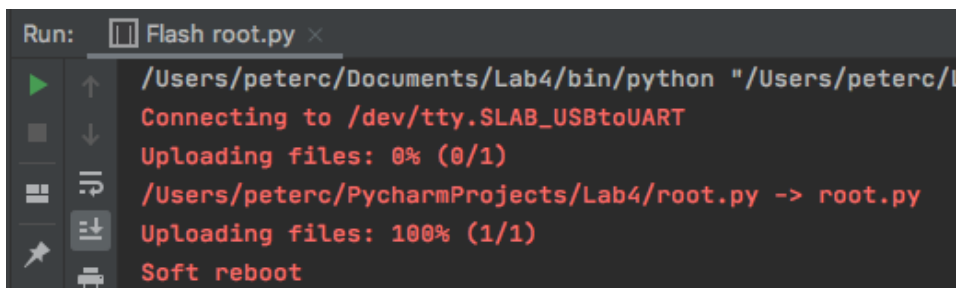


For Windows 10 users, you must use the '+' sign in the top left corner of the dialogue box (See screenshot below.)



Step 4: Flash the program

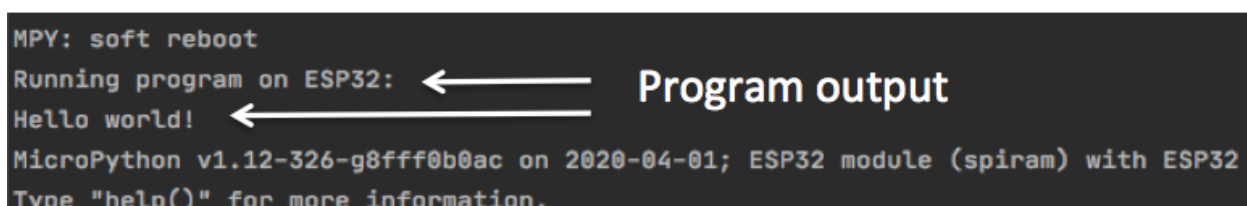
Now click the run icon (green arrow) . You will see a RUN message window appears and report that uploading is complete.



Now do the same for `main.py`.

Step 5: Run the “Hello world!” program

Go back to `Tools > MicroPython > MicroPython REPL` menu, and you will start communicating with the ESP32 via the USB cable again. **Type CTRL+D to run your program** (this is called soft reboot).



Task 8: “Hello world!” on the OLED display

This final task is to show you how to display the “Hello world!” message on the builtin OLED display.

Step 1: Download and flash the OLED driver and font files

Go to the course webpage and down two files: `oled.py.zip` and `font.py.zip` to your project folder. Unzip them.

Use `Edit Configurations ...`, flash these two files to the ESP32 non-volatile memory.

These two files provide an easy method to display text and draw diagrams on the OLED display.

The file `oled.py` is known as **driver** because shield you from needing to know about details of the OLED hardware.

Step 2: Create modify `main.py` and create a new uPy script `hello.py` as shown below.

```
main.py x hello.py x
1 # main.py - this is where you put your main program code
2
3 execfile('hello.py')

13 from oled import OLED # load oled module from flash memory
14 import time, random # uPy libraries
15
16 oled = OLED() # Instantiate an OLED object
17 oled.poweron()
18 oled.init_display() # initialise the OLED display
19
20 # Simple Hello world message
21 oled.draw_text(0,0,'Hello World!') # each character is 6x8 pixels
22 oled.display() # flush display
```

Step 3: Flash them to the ESP32

Step 4: Now press the RESET button the ESP32 module (button below the USB socket) or CTRL-D in the uPy REPL terminal. You should see this on the OLED display:



Now unplug and re-plug the USB cable. Press the RESET button and you will see that the program will run again without flashing or interacting with PyCharm. You are now ready to do Lab 4 Part B.

