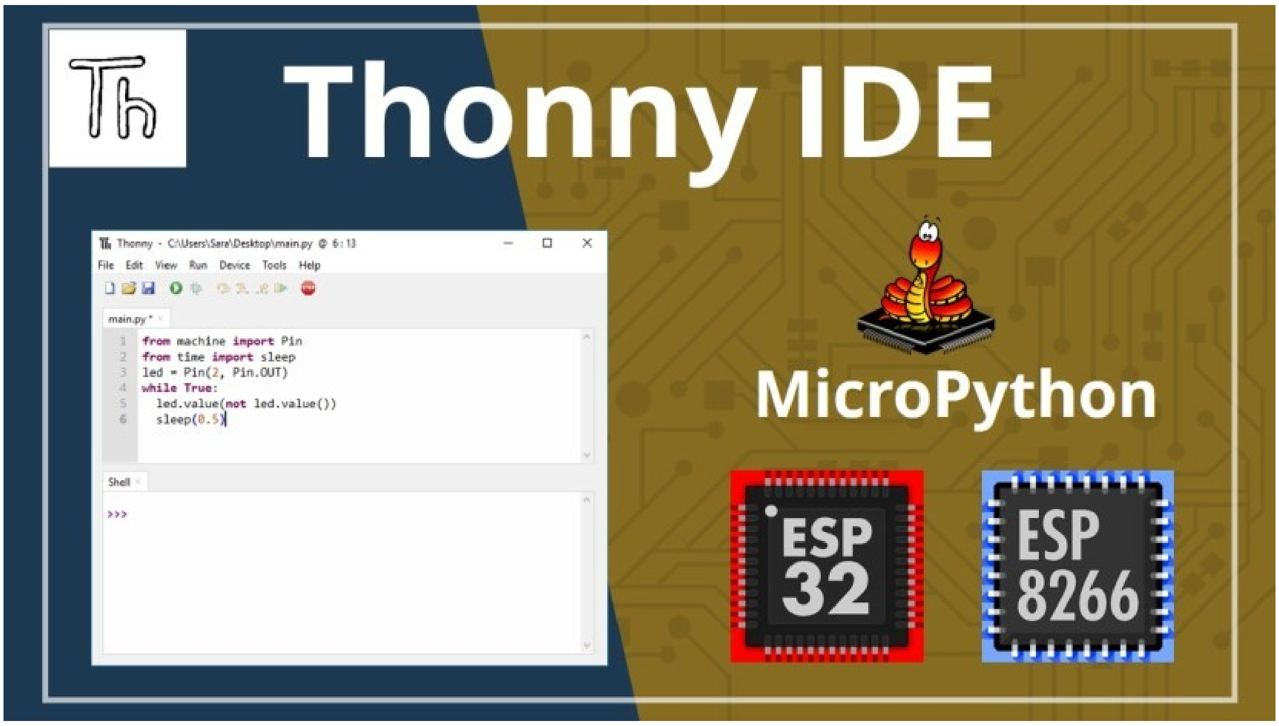
# Part 21
# -
# MicroPython using Thonny

*Version: 2021-02-02*

## MicroPython for ESP32 using Thonny

If you want to program your ESP32 with MicroPython firmware, it’s very handy to use an IDE.
In this guide, we’ll introduce you to Thonny IDE.


Thonny IDE
Thonny - C:\Users\Sara\Desktop\main.py @ 6 : 13
File Edit View Run Device Tools Help
main.py *
Shell
MicroPython
ESP 32
ESP 8266

```
from machine import Pin
from time import sleep
led = Pin(2, Pin.OUT)
while True:
  led.value(not led.value())
  sleep(0.5)
```

```
>>>
```

Thonny allows you to program your ESP32 boards with MicroPython, and it is compatible with Windows, Mac OS X, and Linux. It even comes installed by default with Raspbian OS for Raspberry Pi. Additionally, it’s easy to install, so you shouldn’t have problems with the installation process.

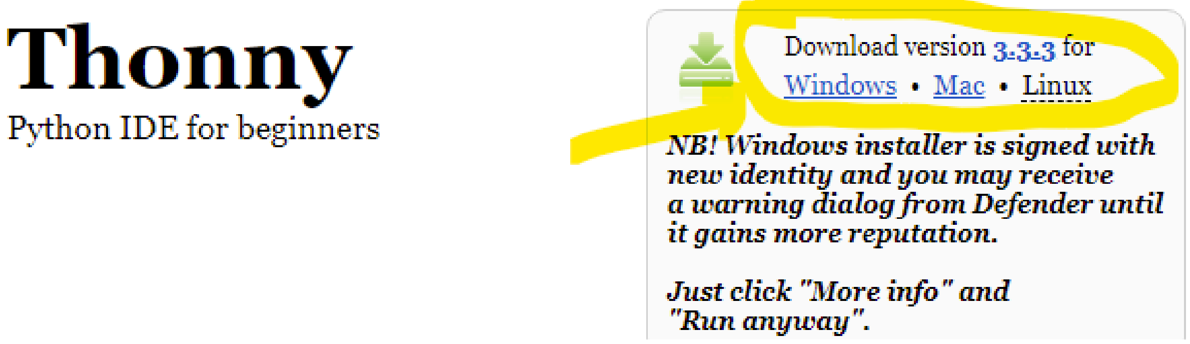
### Downloading and Flashing MicroPython Firmware

Unlike other boards, MicroPython isn’t flashed onto the ESP32 by default. That’s the first thing you need to do to start programming your boards with MicroPython: flash the firmware.
If not already done, see previous document to do so.

## Installing Thonny IDE on Windows

**Tip:** Thonny IDE comes installed by default on Raspbian OS that is used with the Raspberry Pi board.

To install Thonny on your Windows PC, follow the next instructions
Go to https://thonny.org and dwnload the version for Windows and wait a few seconds while it downloads.



Thonny

Python IDE for beginners

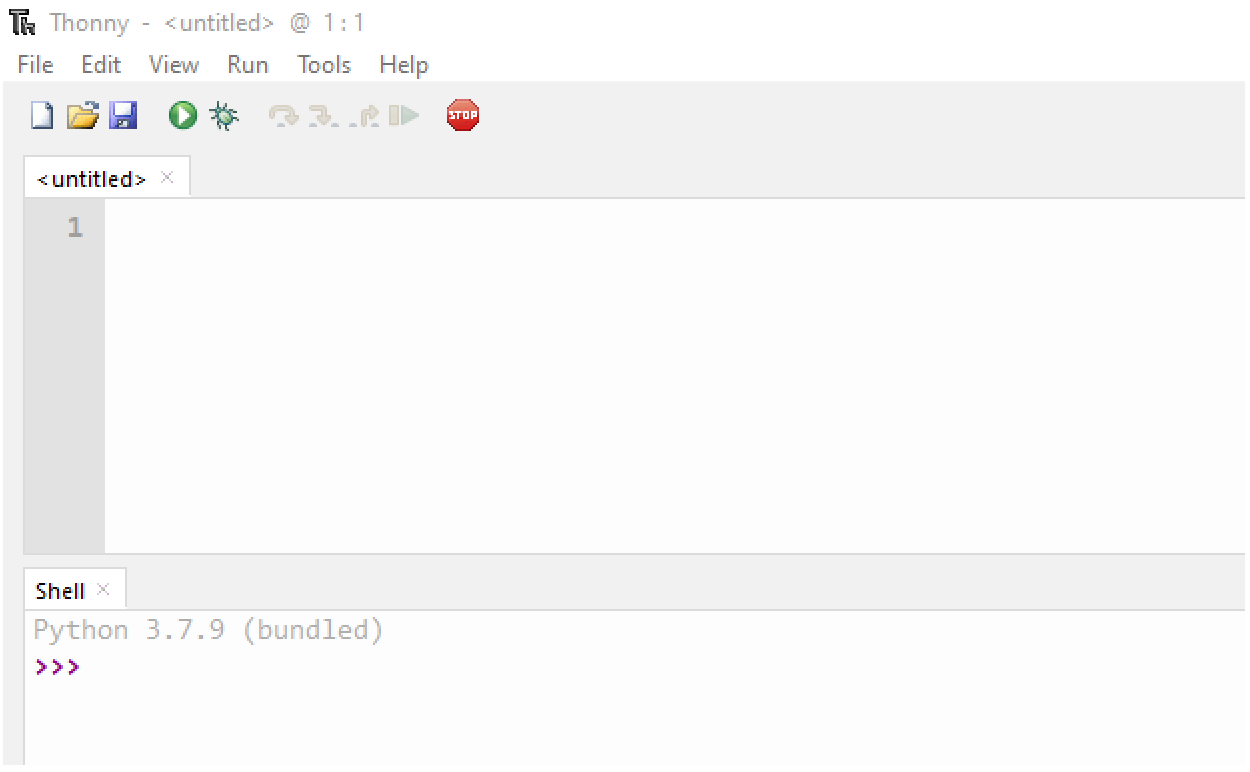
Download version 3.3.3 for
Windows • Mac • Linux

*NB! Windows installer is signed with new identity and you may receive a warning dialog from Defender until it gains more reputation.*

*Just click "More info" and "Run anyway".*

***Note:*** as you can see, at writing this doc, the version was 3.3.3

Run the *.exe* file and **f**ollow the installation wizard to complete the installation process. You just need to click "Next".
After completing the installation, open Thonny IDE. A window as shown in the following figure should open.

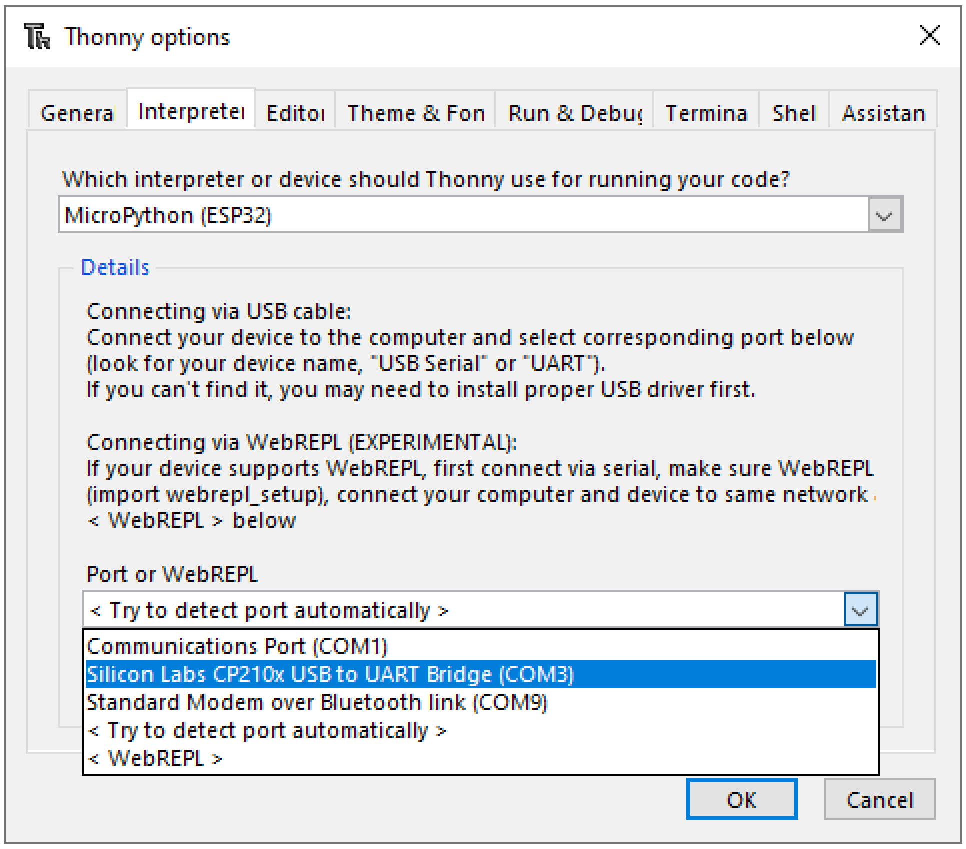


```
Python 3.7.9 (bundled)
>>>
```

**Testing the Installation**
***Important:*** before testing the installation, your ESP32 board needs to be flashed with MicroPython firmware.

Connect the board to your computer using an USB cable. To test the installation, you need to tell Thonny that you want to run MicroPython Interpreter and select the board you are using.

Go to Tools > Options and select the Interpreter tab. Choose MicroPython on a generic device and then, select your device serial port


Thonny options

Genera | Interpretei | Editoı | Theme & Fon | Run & Debuę | Termina | Shel | Assistan

Which interpreter or device should Thonny use for running your code?
MicroPython (ESP32)

Details

Connecting via USB cable:
Connect your device to the computer and select corresponding port below
(look for your device name, "USB Serial" or "UART").
If you can't find it, you may need to install proper USB driver first.

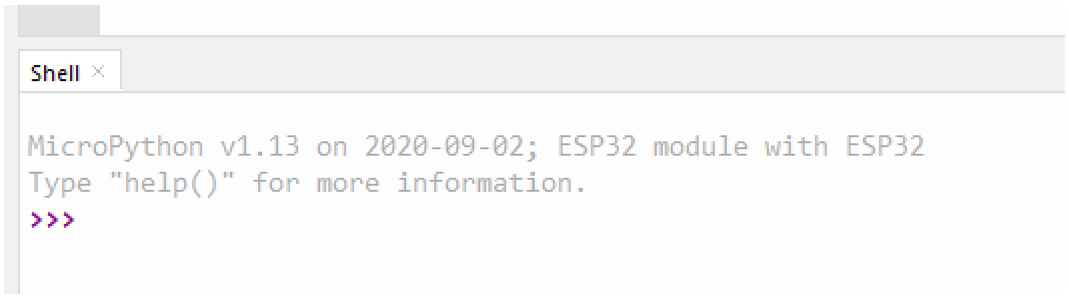
Connecting via WebREPL (EXPERIMENTAL):
If your device supports WebREPL, first connect via serial, make sure WebREPL
(import webrepl_setup), connect your computer and device to same network
< WebREPL > below

Port or WebREPL
< Try to detect port automatically >

Communications Port (COM1)
Silicon Labs CP210x USB to UART Bridge (COM3)
Standard Modem over Bluetooth link (COM9)
< Try to detect port automatically >
< WebREPL >

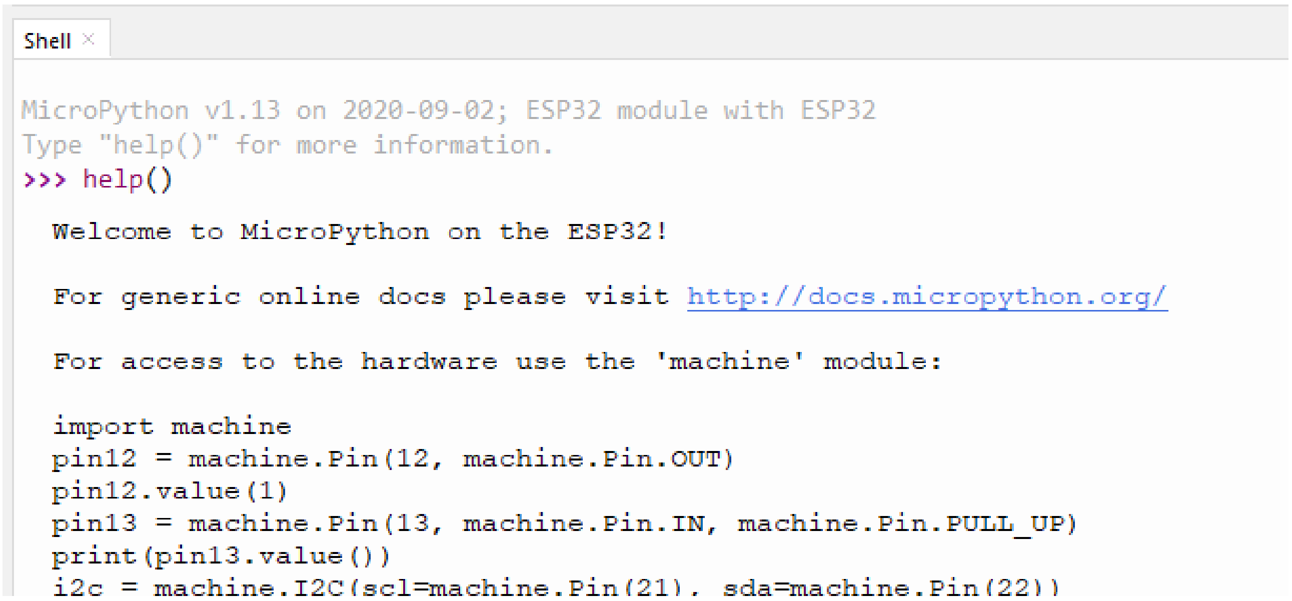
You can also select the "`Try to detect automatically`" option, but *only* if you just have one board connected to your computer at a time. Otherwise, select the specific port for the board you're using.

Thonny IDE should now be connected to your board and you should see the prompt on the Shell.



```
MicroPython v1.13 on 2020-09-02; ESP32 module with ESP32
Type "help()" for more information.
>>>
```

Type the command **help()** in the Shell and see if it responds back.



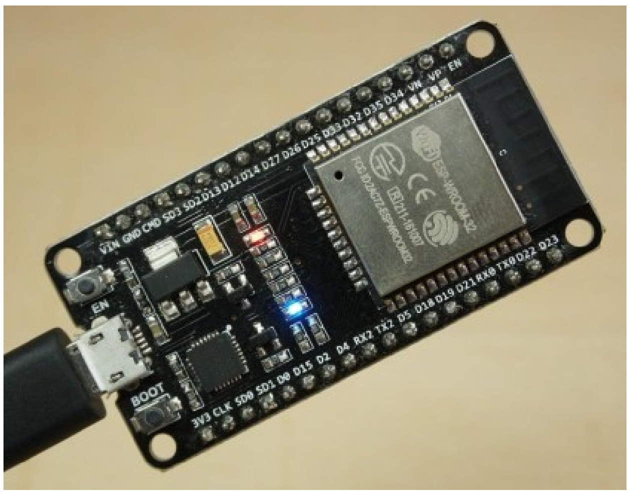
```
MicroPython v1.13 on 2020-09-02; ESP32 module with ESP32
Type "help()" for more information.
>>> help()
  Welcome to MicroPython on the ESP32!

  For generic online docs please visit http://docs.micropython.org/

  For access to the hardware use the 'machine' module:

  import machine
  pin12 = machine.Pin(12, machine.Pin.OUT)
  pin12.value(1)
  pin13 = machine.Pin(13, machine.Pin.IN, machine.Pin.PULL_UP)
  print(pin13.value())
  i2c = machine.I2C(scl=machine.Pin(21), sda=machine.Pin(22))
```

If it responded back, everything is working fine. Now, you can send a few more commands to test.

Send the following commands to light up the on-board LED

```
>>> from machine import Pin
>>> Pin(2, Pin.OUT).value(1)
```

The on-board LED should light up.

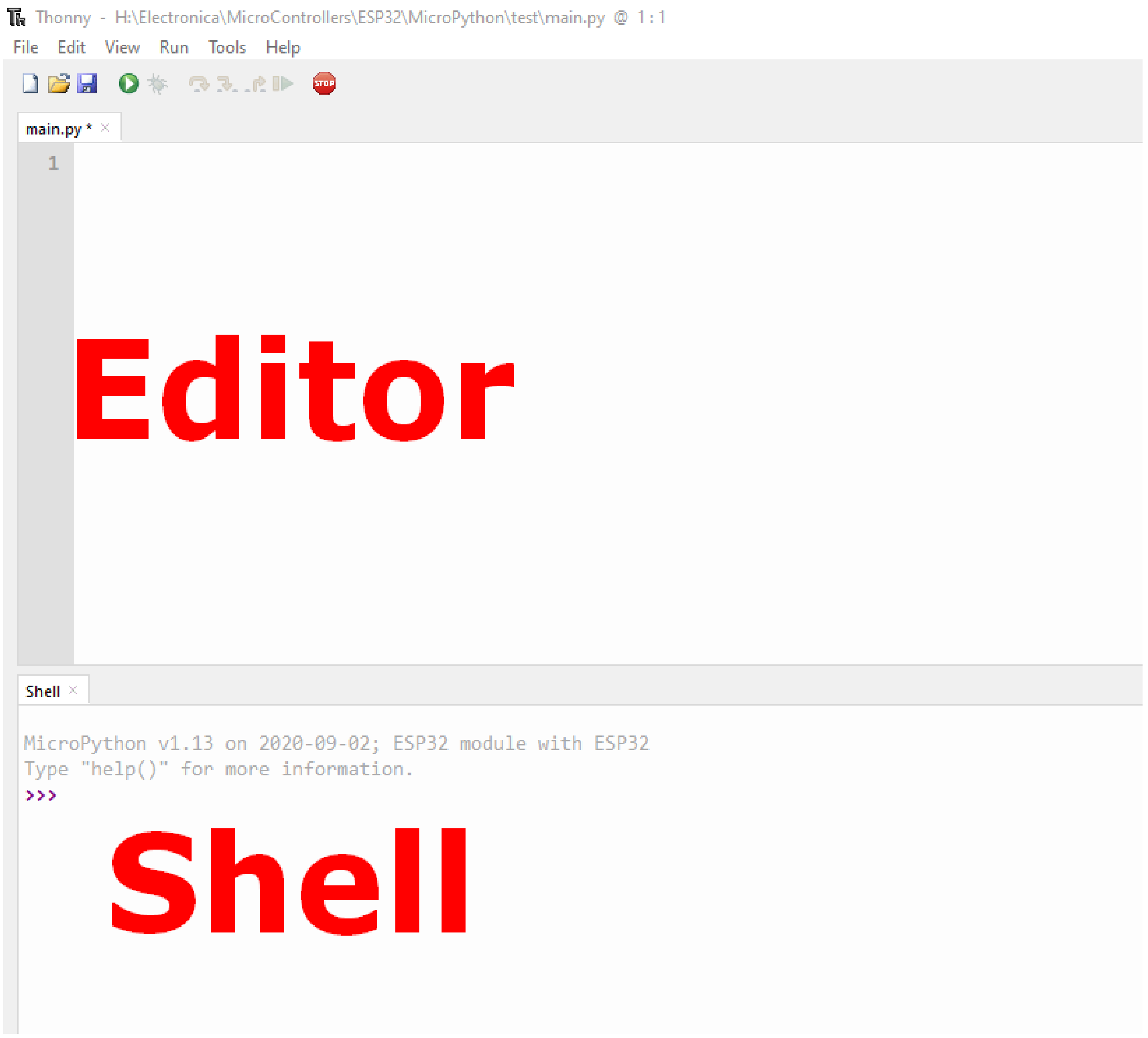


Then, turn off the led

```
>>> Pin(2, Pin.OUT).value(0)
```

## Thonny IDE Overview

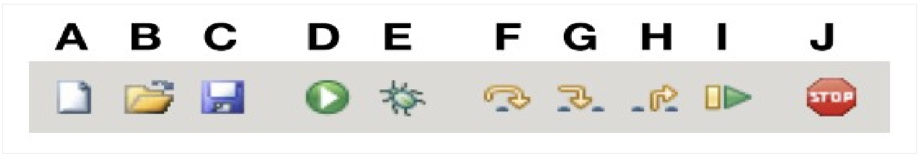
There are two different sections: the **Editor** and the **MicroPython Shell/Terminal**:


Thonny - H:\Electronica\MicroControllers\ESP32\MicroPython\test\main.py @ 1 : 1
File Edit View Run Tools Help
main.py *
Editor
Shell
Shell

```
MicroPython v1.13 on 2020-09-02; ESP32 module with ESP32
Type "help()" for more information.
>>>
```

The **Editor** section is where you write your code and edit your *.py* files. You can open more than one file, and the Editor will open a new tab for each file.
On the **Shel**l you can type commands to be executed immediately by your ESP board without the need to upload new files. The terminal also provides information about the state of an executing program, shows errors related with upload, syntax errors, prints messages, etc…

**The Icons**
Across the top you'll see several icons. You'll see an image of the icons below, with a letter above each one. We will use these letters to talk about each of the icons:



**A:** The paper icon allows you to create a new file. Typically in Python you want to separate your programs into separate files.

**B:** The open folder icon allows you to open a file that already exists on your computer. This might be useful if you come back to a program that you worked on previously.

**C:** The floppy disk icon allows you to save your code. Press this early and often. You'll use this later to save your first Thonny Python program.

**D:** The play icon allows you to run your code.

**E:** The bug icon allows you to debug your code. It's inevitable that you will encounter bugs when you're writing code. Bugs can come in many forms, sometimes appearing when you use inappropriate syntax and sometimes when your logic is incorrect. Thonny's bug button is typically used to spot and investigate bugs.

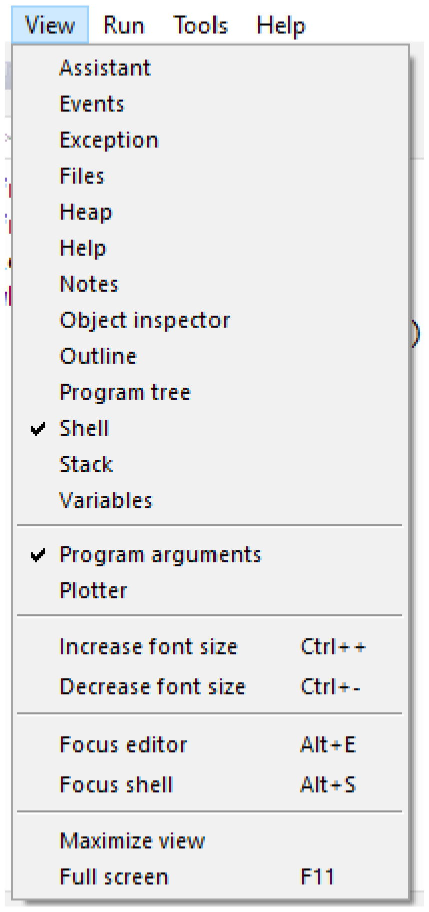
**F-H:** The arrow icons allow you to run your programs step by step. This can be very useful when you're debugging or, in other words, trying to find those nasty bugs in your code. These icons are used after you press the bug icon. You'll notice as you hit each arrow, a yellow highlighted bar will indicate which line or section Python is currently evaluating:

- The **F** arrow tells Python to take a big step, meaning jumping to the next line or block of code.
- The **G** arrow tells Python to take a small step, meaning diving deep into each component of an expression.
- The **H** arrow tells Python to exit out of the debugger.

**I:** The resume icon allows you to return to play mode from debug mode. This is useful in the instance when you no longer want to go step by step through the code, and instead want your program to finish running.

**J:** The stop icon allows you to stop running your code. This can be particularly useful if, let's say, your code runs a program that opens a new window, and you want to stop that program. You'll use the stop icon later in the tutorial.

**Other UI Features**
To see more of the other features that Thonny has to offer, navigate to the menu bar and select the *View* dropdown. You should see that *Shell* has a check mark next to it, which is why you see the Shell section in Thonny's application window:



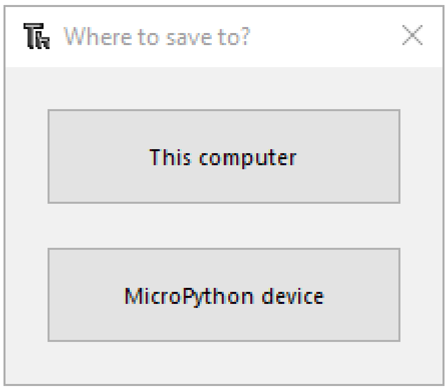
Let's explore some of the other offerings, specifically those that will be useful

- **Help:** You'll select the *Help* view if you want more information about working with Thonny.
- **Variables:** This feature can be very valuable. A variable in Python is a value that you define in code. Variables can be numbers, strings, or other complex data structures. This section allows you to see the values assigned to all of the variables in your program.
- **Assistant:** The Assistant is there to give you helpful hints when you hit Exceptions or other types of errors.

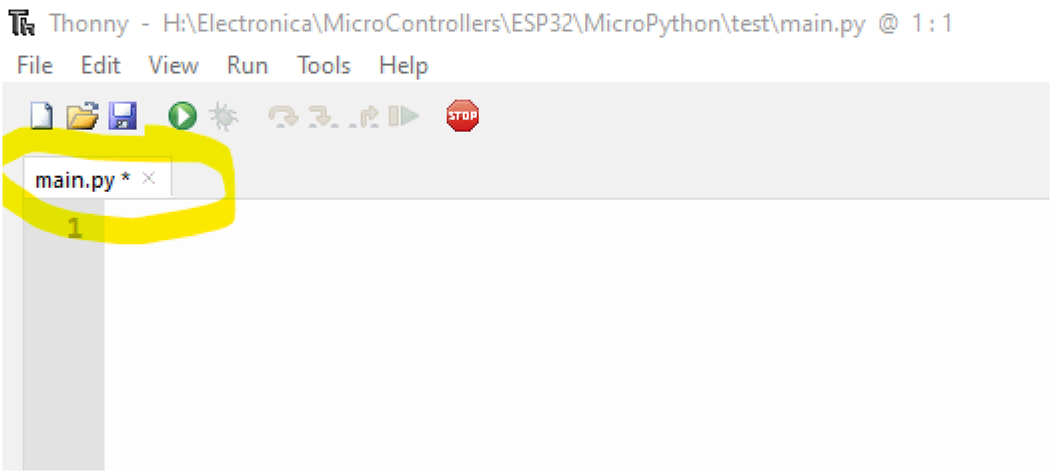
The other features will become useful as you advance your skills. Check them out once you get more comfortable with Thonny!

**Running Your First Script**
To get you familiar with the process of writing a file and executing code on your ESP32 boards, we'll upload a new script that simply blinks the on-board LED of your ESP32.

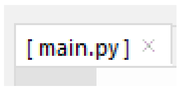
***Creating the main.py file on your board***
When you open Thonny IDE for the first time, the Editor shows an untitled file. Save that file as *main.py*. Simply, click the save icon. You will be asked to save the file on your computer or on MicroPython Device. Go for 'This computer' and save it with the name *main.py*.



The Editor should now have a tab called *main.py*.


Thonny - H:\Electronica\MicroControllers\ESP32\MicroPython\test\main.py @ 1 : 1
File
Edit
View
Run
Tools
Help
main.py *

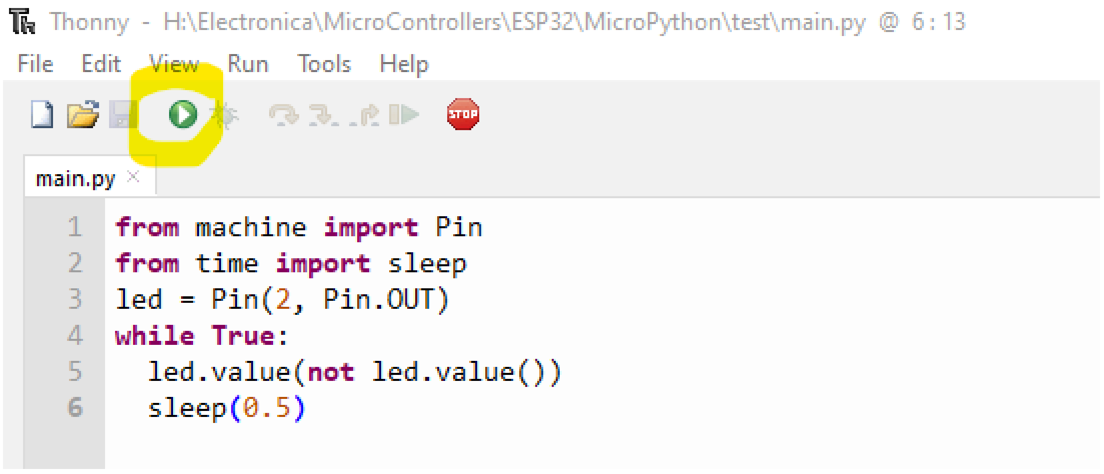
***Note:*** when save on your computer the file tab looks like above. If saved on the micropython device it looks like this. Notice the square brackets



Copy the following code to the *main.py* file:

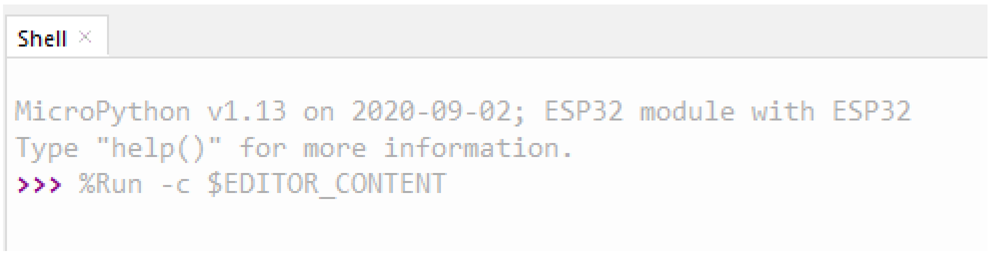
```
from machine import Pin
from time import sleep
led = Pin(2, Pin.OUT)
while True:
  led.value(not led.value())
  sleep(0.5)
```

***Running the Script***
Select the "Run" command


Thonny - H:\Electronica\MicroControllers\ESP32\MicroPython\test\main.py @ 6 : 13
File Edit View Run Tools Help
STOP
main.py

```python
from machine import Pin
from time import sleep
led = Pin(2, Pin.OUT)
while True:
  led.value(not led.value())
  sleep(0.5)
```

It should write the following on the Shell.

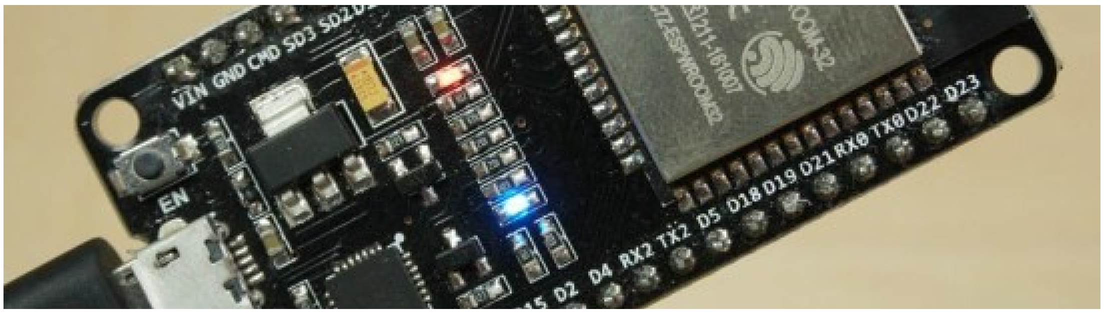


```
MicroPython v1.13 on 2020-09-02; ESP32 module with ESP32
Type "help()" for more information.
>>> %Run -c $EDITOR_CONTENT
```

**Note:** uploading the code as main script will save the current file with the name *main.py* on the ESP, even if you have saved it in your computer with a different name. The same happens for the *boot.py* file.

***Important:*** when the ESP restarts, first it runs the *boot.py* file and afterwards the *main.py*.

The ESP on-board LED should be blinking.

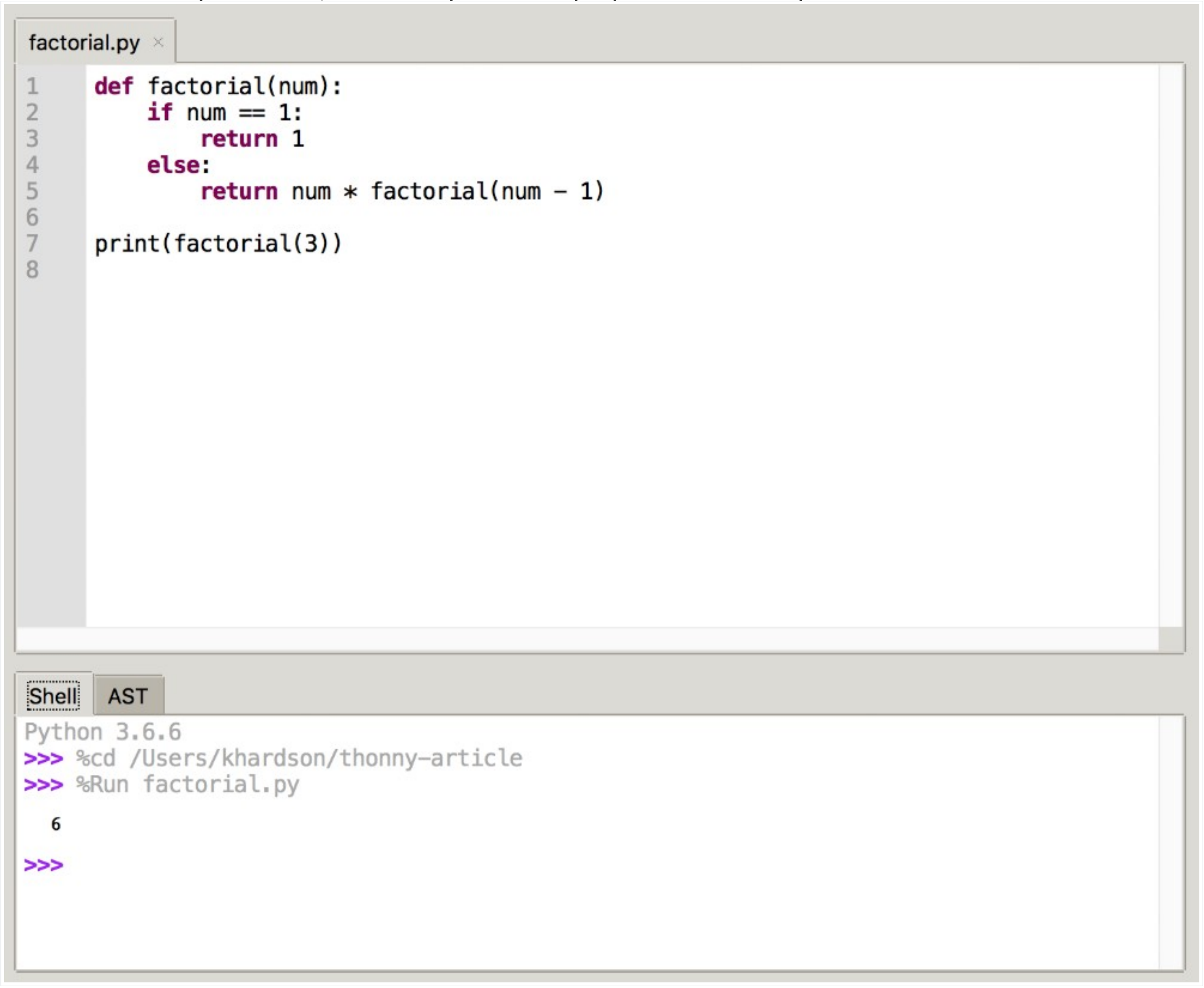


**Write Some More Code**
Create a new file and add the following function:

```
def factorial(num):
    if num == 1:
        return 1
    else:
        return num * factorial(num - 1)

print(factorial(3))
```

***Save Your Code***
Before we move on, let's save your program. You can also do this by clicking the blue floppy disk icon or by going to the menu bar and selecting *File* > *Save*. Let's call the program `factorial.py`. Again, you will be asked to save the file on your computer or on the MicroPython device.

***Run Your Code***
In order to run your code, find and press the play icon. The output should look like this:

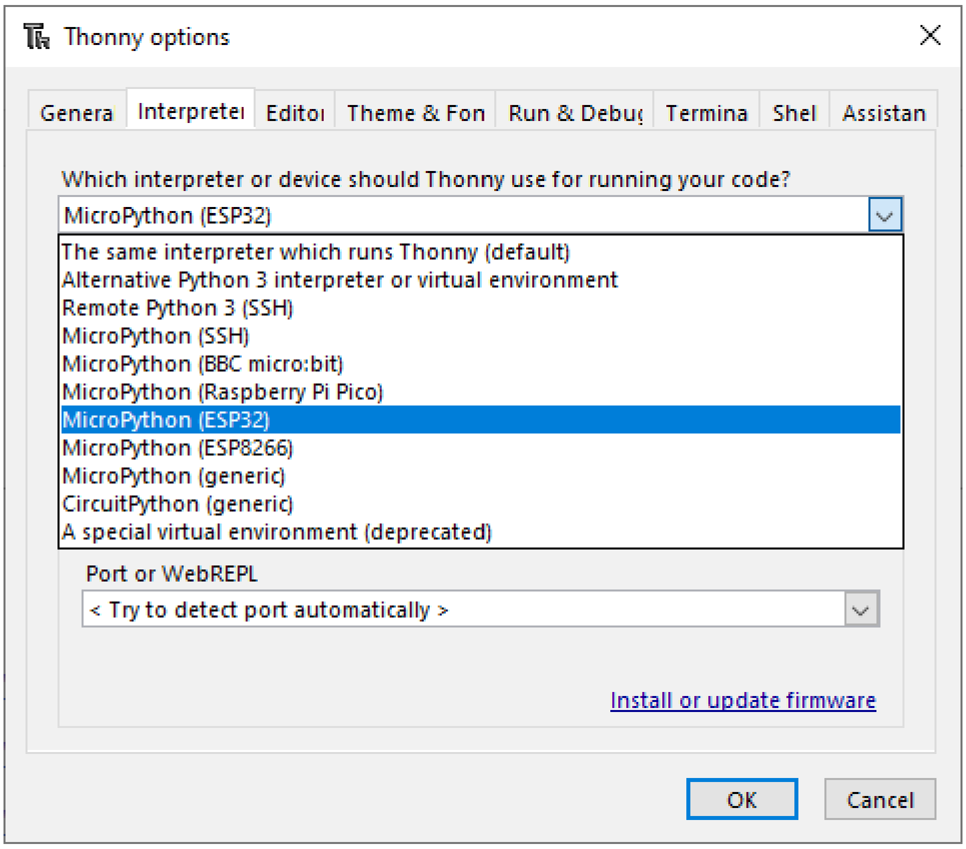


```
def factorial(num):
    if num == 1:
        return 1
    else:
        return num * factorial(num - 1)

print(factorial(3))
```

```
Python 3.6.6
>>> %cd /Users/khardson/thonny-article
>>> %Run factorial.py

  6

>>>
```

***Debug Your Code***

Debugger is only available when you have selected ‘`The same interpreter which runs Thonny (default)`’ in `Tools → Options → Interpreter`. If you select ‘`MicroPython (ESP32)`’, debugging is not available


Thonny options

Which interpreter or device should Thonny use for running your code?

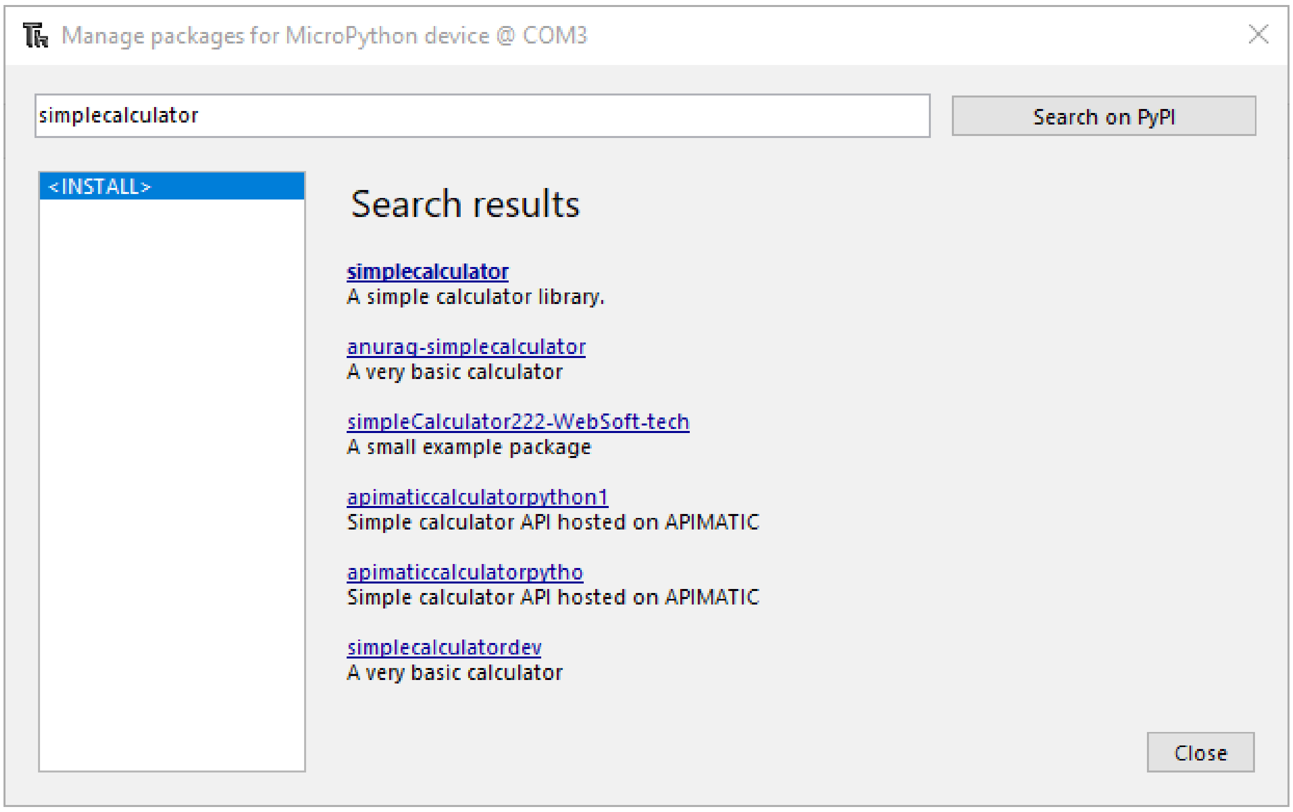
MicroPython (ESP32)

- The same interpreter which runs Thonny (default)
- Alternative Python 3 interpreter or virtual environment
- Remote Python 3 (SSH)
- MicroPython (SSH)
- MicroPython (BBC micro:bit)
- MicroPython (Raspberry Pi Pico)
- MicroPython (ESP32)
- MicroPython (ESP8266)
- MicroPython (generic)
- CircuitPython (generic)
- A special virtual environment (deprecated)

Port or WebREPL

< Try to detect port automatically >

**The Package Manager**
As you continue to learn Python, it can be quite useful to download a Python package to use inside of your code. This allows you to use code that someone else has written inside of your program.
Consider an example where you want to do some calculations in your code. Instead of writing your own calculator, you might want to use a third-party package called `simplecalculator`. In order to do this, you'll use Thonny's package manager.
The package manager will allow you to install packages that you will need to use with your program. Specifically, it allows you to add more tools to your toolbox. Thonny has the built-in benefit of handling any conflicts with other Python interpreters.
To access the package manager, go to the menu bar and select `Tools > Manage Packages…`
This should pop open a new window with a search field. Type `simplecalculator` into that field and click the `Search on PyPi` button.
The output should look similar to this:



Search results

simplecalculator
A simple calculator library.

anurag-simplecalculator
A very basic calculator

simpleCalculator222-WebSoft-tech
A small example package

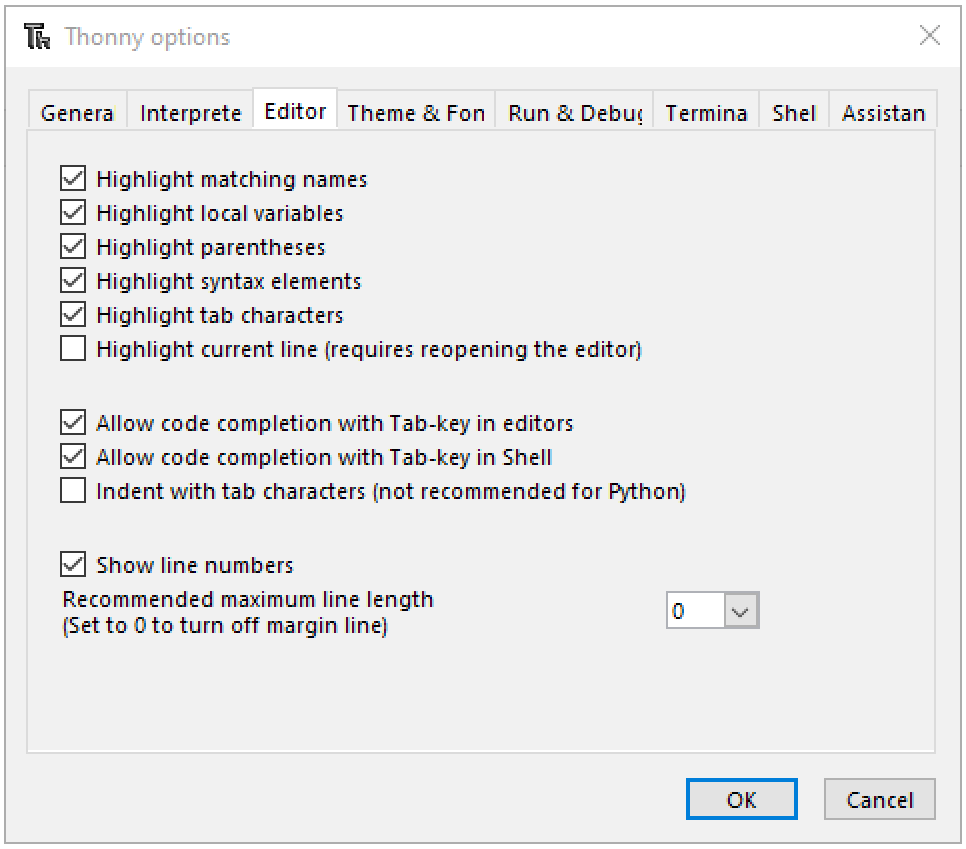
apimaticcalculatorpython1
Simple calculator API hosted on APIMATIC

apimaticcalculatorpytho
Simple calculator API hosted on APIMATIC

simplecalculatordev
A very basic calculator

Select the right one (which is the first) and go ahead to click `Install` to install this package.
You will see a small window pop up showing the system's logs while it installs the package.
Once it completes, you are ready to use `simplecalculator` in your code.

**Variable Scope Highlighting**
Thonny offers variable highlighting to remind you that the same name doesn't always mean the same variable. In order for this feature to work, on the menu bar, go to `Tools → Options → Editor` and ensure that `Highlight matching names` is checked.


Thonny options

Genera | Interprete | Editor | Theme & Fon | Run & Debu | Termina | Shel | Assistan

- ☑ Highlight matching names
- ☑ Highlight local variables
- ☑ Highlight parentheses
- ☑ Highlight syntax elements
- ☑ Highlight tab characters
- ☐ Highlight current line (requires reopening the editor)

- ☑ Allow code completion with Tab-key in editors
- ☑ Allow code completion with Tab-key in Shell
- ☐ Indent with tab characters (not recommended for Python)

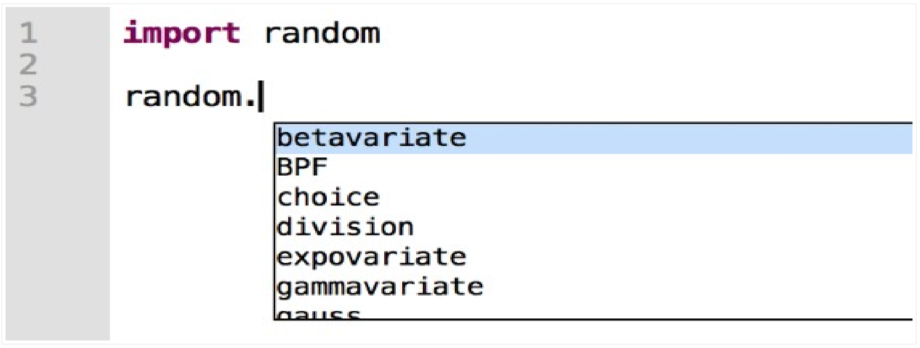
- ☑ Show line numbers

Recommended maximum line length
(Set to 0 to turn off margin line) 0

OK | Cancel

This feature can really help you avoid typos and understand the scope of your variables.

**Code Completion**
Thonny also offers code completion for APIs. Notice in the snapshot below how pressing the Tab key shows the methods available from the random library:


```
import random

random.
```
```
betavariate
BPF
choice
division
expovariate
gammavariate
gauss
```

This can be very useful when you're working with libraries and don't want to look at the documentation to find a method or attribute name.

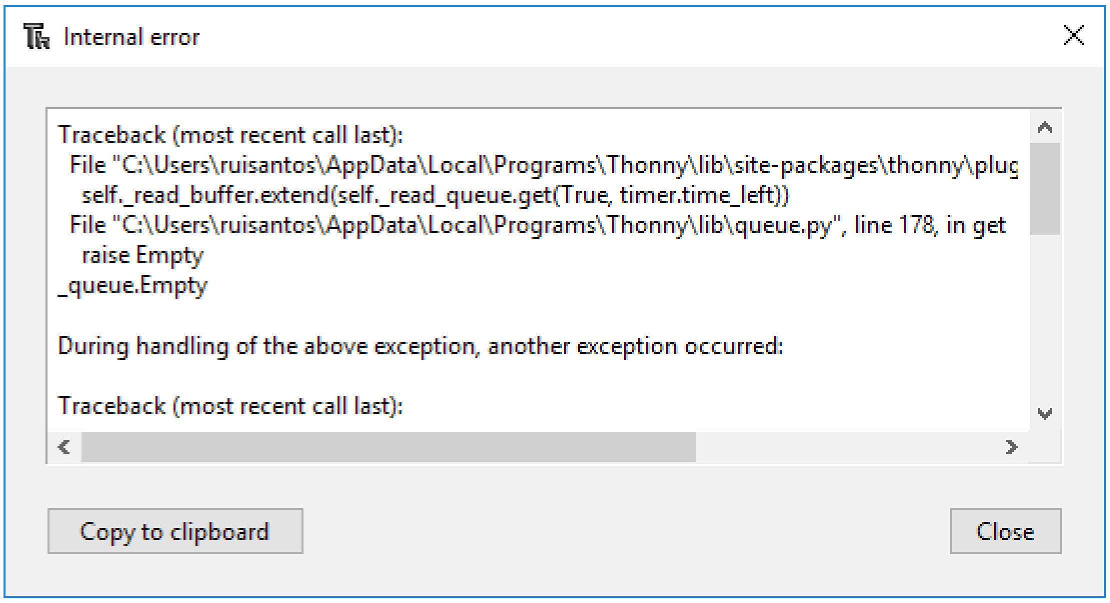
**Troubleshooting Tips for Thonny IDE**
We've discovered some common problems and error messages that occur with Thonny IDE. Usually restarting your ESP with the on-board EN/RST button fixes your problem. Or pressing the Thonny IDE *"Stop/Restart backend"* button and repeating your desired action. In case it doesn't work for you, read these next common errors and discover how to solve them.

```
Error #1: You get one of the following messages:
========================= RESTART =========================
Unable to connect to COM4
Error: could not open port 'COM4': FileNotFoundError(2, 'The system cannot
find the file specified.', None, 2)

Check the configuration, select Run → Stop/Restart or press Ctrl+F2 to try
again. (On some occasions it helps to wait before trying again.)
Or:
========================= RESTART =========================
Could not connect to REPL.
Make sure your device has suitable firmware and is not in bootloader mode!
Disconnecting.
Or:
========================= RESTART =========================
Lost connection to the device (EOF).
```

Unplug, and plug back your ESP board. Then, double-check that you've selected the right serial port in the *Tools > Options > Interpreter > Port*. Then, click the *"Stop/Restart backend"* button to establish a serial communication. You should now be able to upload a new script or re-run new code.
These errors might also mean that you have your serial port being used in another program (like a serial terminal or in the Arduino IDE). Double-check that you've closed all the programs that might be establishing a serial communication with your ESP board. Then, unplug and plug back your ESP board. Finally, restart Thonny IDE.

```
Error #2: Thonny IDE fails to respond or gives an Internal Error
```

When this happens, you can usually close that window and it will continue to work. If it keeps crashing, we recommend restarting the Thonny IDE software.



```
Traceback (most recent call last):
  File "C:\Users\ruisantos\AppData\Local\Programs\Thonny\lib\site-packages\thonny\plug
    self._read_buffer.extend(self._read_queue.get(True, timer.time_left))
  File "C:\Users\ruisantos\AppData\Local\Programs\Thonny\lib\queue.py", line 178, in get
    raise Empty
_queue.Empty

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
```

**Error #3**: Thonny IDE hangs when pressing "**Stop/Restart backend**" button

When you press the "Stop/Restart backend" button, you need to wait a few seconds. The ESP needs time to restart and establish the serial communication with Thonny IDE. If you press the "Stop" button multiple times or if you press the button very quickly, the ESP will not have enough time to restart properly and it's very likely to crash Thonny IDE.

**Error #4**: Problem restarting your ESP board, running a new script or opening the serial port

```
Brownout detector was triggered
Or if the ESP keeps restarting and printing the ESP boot information:
ets Jun 8 2016 00:22:57

rst:0x1 (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 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:4732
load:0x40078000,len:7496
load:0x40080400,len:5512
```

The "Brownout detector was triggered" error message or constant reboots  means that there's some sort of hardware problem. It's often related to one of the following issues

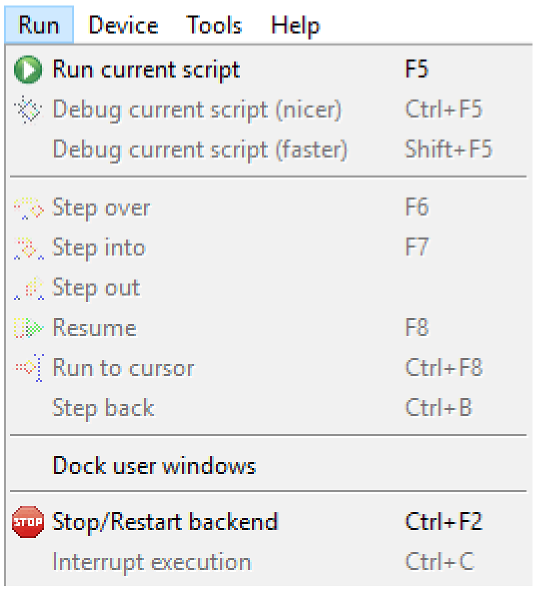
- Poor quality USB cable;
- USB cable is too long;
- Board with some defect (bad solder joints);
- Bad computer USB port;
- Or not enough power provided by the computer USB port.

Try a different shorter USB cable (with data wires), try a different computer USB port or use a USB hub with an external power supply.
***Important:*** if you keep having constant problems or weird error messages, we recommend re-flashing your ESP board with the latest version of MicroPython firmware: Flash/Upload MicroPython Firmware to ESP32.

**Error #5**: When I try to establish a serial communication with the ESP32 in Thonny IDE, I can't get it to connect.

We think this is what's happening: when you're running a script in your board, sometimes it's busy running that script and performing the tasks.
So, you need to try start the connection by pressing *"Stop/Restart backend"* button multiple times or restart the ESP to catch available to establish the serial communication.
***Warning:*** don't press the *"Stop/Restart backend"* button multiple times very quickly. After pressing that button, you need to be patient and wait a few seconds for the command to run.
If you're running a script that uses Wi-Fi, deep sleep, or it's doing multiple tasks, I recommend trying 3 or 4 times to establish the communication. If you can't, I recommend re-flash the ESP with MicroPython firmware.

**Error #6:** debug tools are grayed out:


Run
Device
Tools
Help
Run current script F5
Debug current script (nicer) Ctrl+F5
Debug current script (faster) Shift+F5
Step over F6
Step into F7
Step out
Resume F8
Run to cursor Ctrl+F8
Step back Ctrl+B
Dock user windows
Stop/Restart backend Ctrl+F2
Interrupt execution Ctrl+C