

# Mini Project

## Hands-on LED Blinking using MicroPython

**Prof. Sudip Misra**

Department of Computer Science and Engineering

Indian Institute of Technology Kharagpur

Email: [smisra@sit.iitkgp.ernet.in](mailto:smisra@sit.iitkgp.ernet.in)

Website: <http://cse.iitkgp.ac.in/~smisra/>

Research Lab: [cse.iitkgp.ac.in/~smisra/swan/](http://cse.iitkgp.ac.in/~smisra/swan/)



# Contents

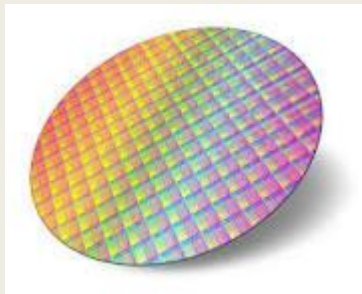


1. **Micro-controllers**
2. **Development Boards**
3. **Prototyping Essentials & Common Electronic Components**
4. **Integrated Development Environment**
5. **Micro-python**
6. **Flashing the firmware to the ESP8266**
7. **Connecting the components**
8. **The code to turn an LED on and off**
9. **Uploading the code**
10. **Common pitfalls (Debugging)**

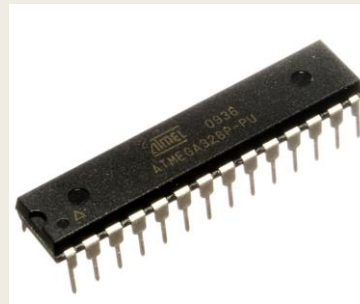


# Microcontrollers

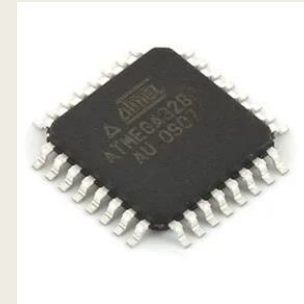
- A microcontroller is a *compact integrated circuit* designed to govern a specific operation in an embedded system. A typical microcontroller includes a *processor, memory, and input/output (I/O) peripherals* on a single chip.
- They are an *integral part* of any *IoT system*.
- They are made up of Silicon Wafers and can come in different shapes and sizes. The suitable package is selected based on our design constraints.



Silicon Wafer



DIP

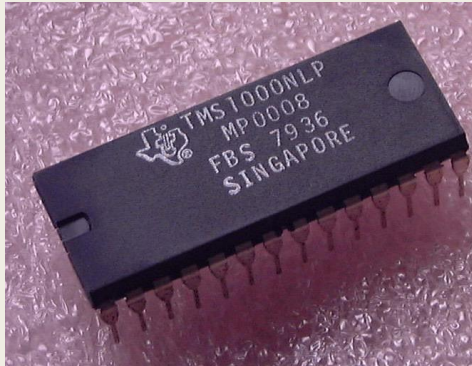


TQFP

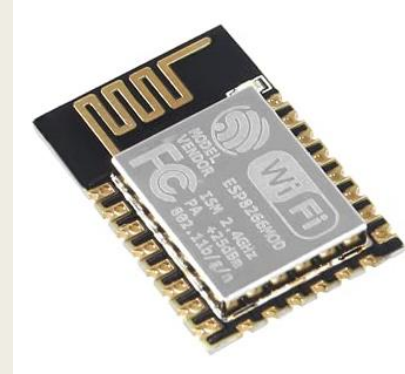


QFN

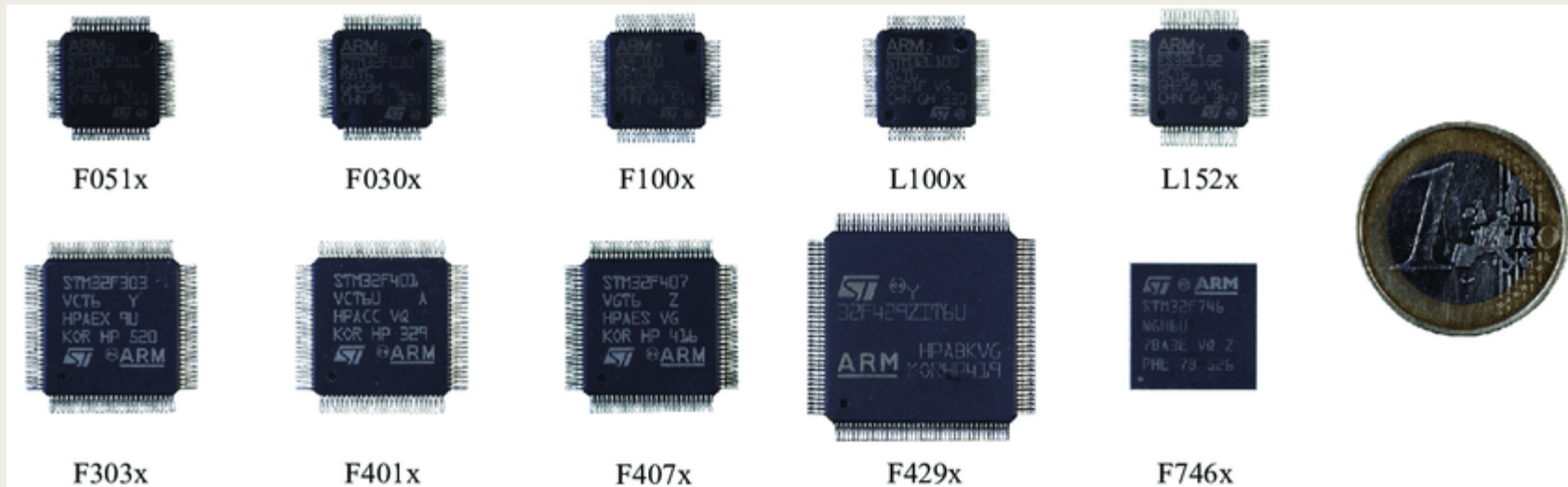
# Microcontrollers



The first microcontroller TMS 1000 in a 28-pin plastic DIP package. It was developed by Texas Instruments in 1974.



This is the ESP8266 (ESP12E). We will be working with this microcontroller package. It has built-in Wi-Fi capabilities.



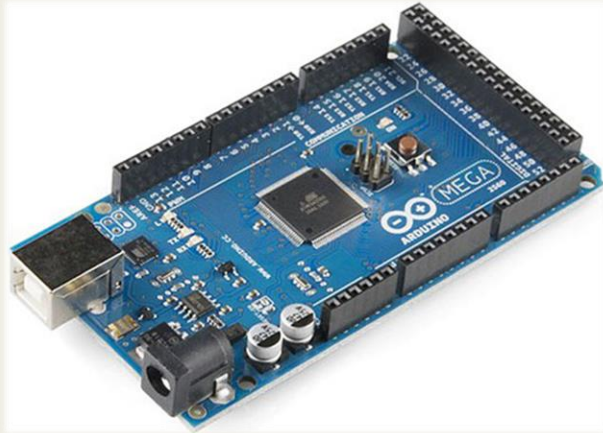
Microcontrollers today can be smaller than the size of a coin

# Development Boards

- A microcontroller *requires certain peripheral components* for its functioning.
- They require external components for *power management* and *programming*.
- As a result, they can not be used standalone. Therefore, for our ease during prototyping, we use a *Development Board*.
- *Development Boards* are printed circuit boards with a microcontroller/microprocessor mounted on them *with few other hardware components*. Development boards are meant for System Designers to *become acquainted with programming a processor onboard* and also to *develop and test projects effectively and efficiently*.



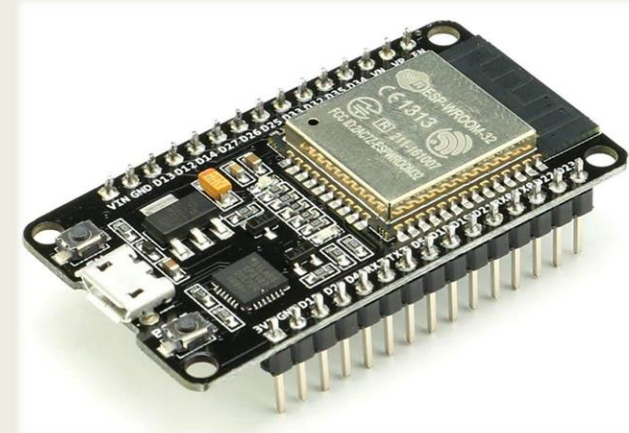
# Development Boards



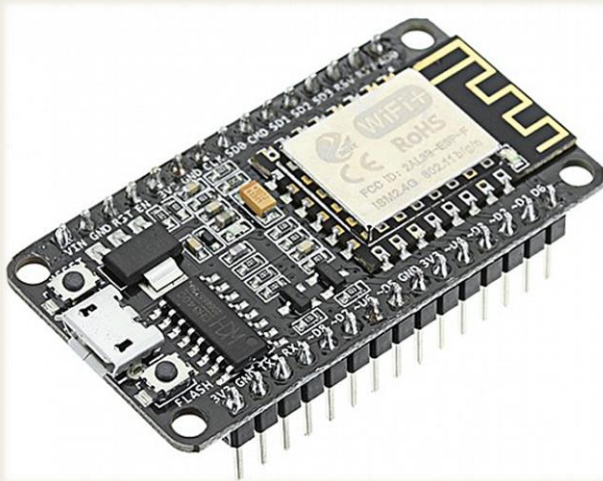
Arduino Mega based on ATMEGA2560



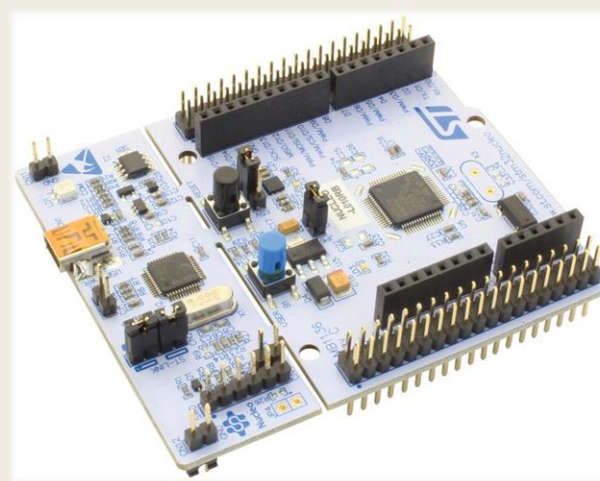
Arduino Uno based on ATMEGA328



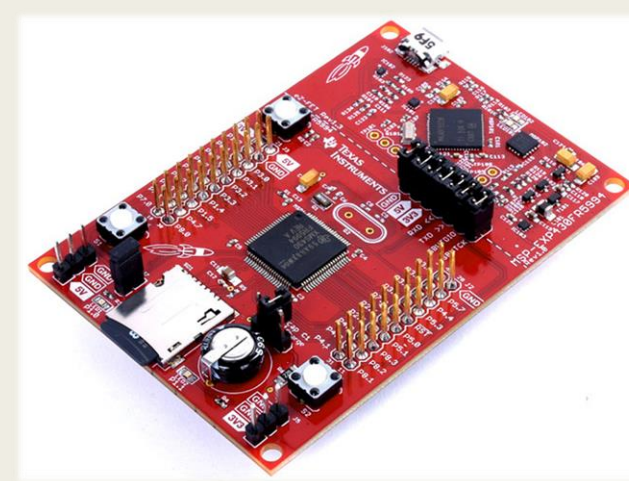
DOIT ESP32 NodeMCU based on ESP32



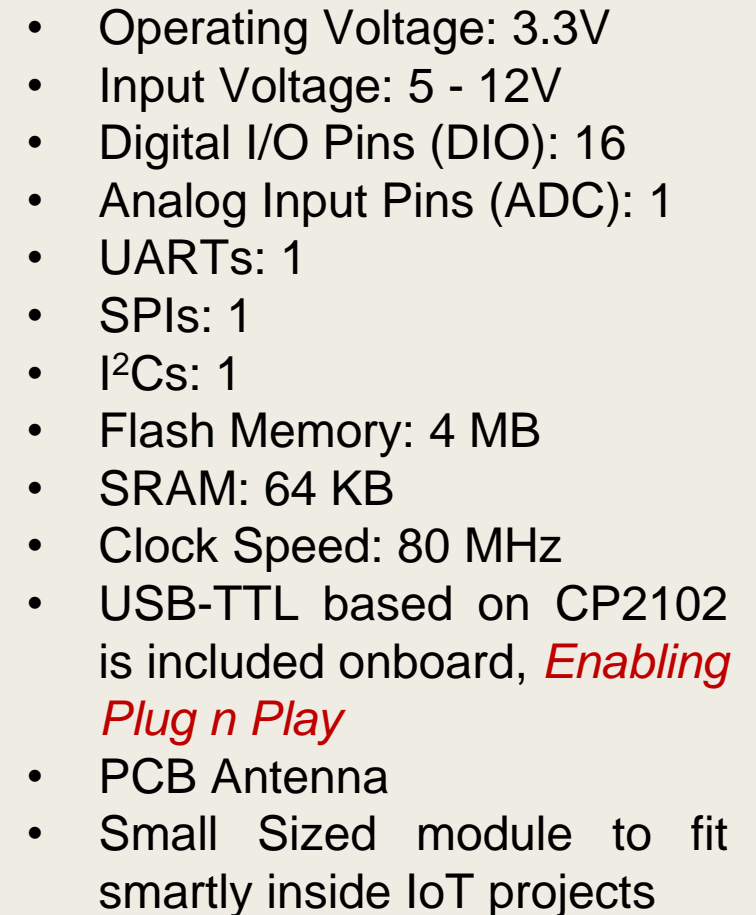
Amica NodeMCU Devkit V 1.0 based on ESP8266



NUCLEO-L010RB development board based on STM32L010RB



MSP430FR5994 LaunchPad Development Kit



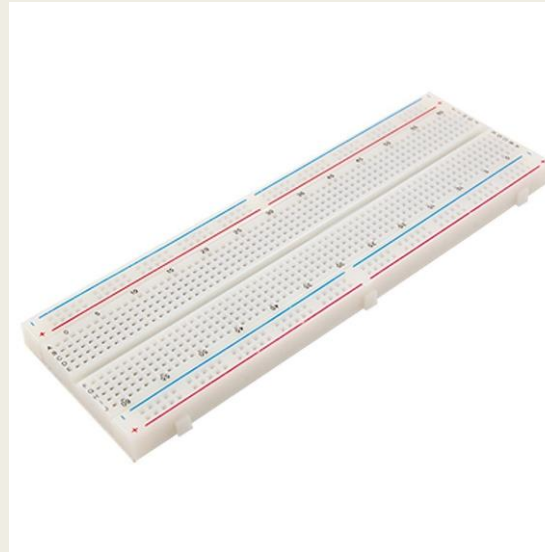
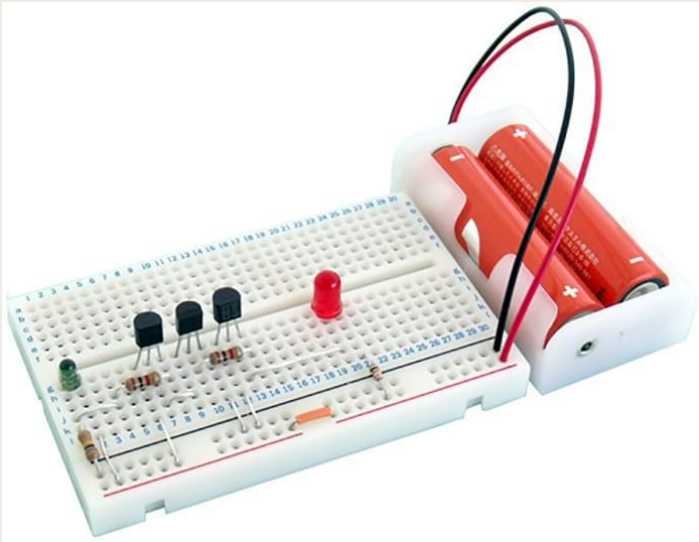
## ESP8266 Development Board – NodeMCU Devkit V1.0



# Prototyping Essentials - Breadboard

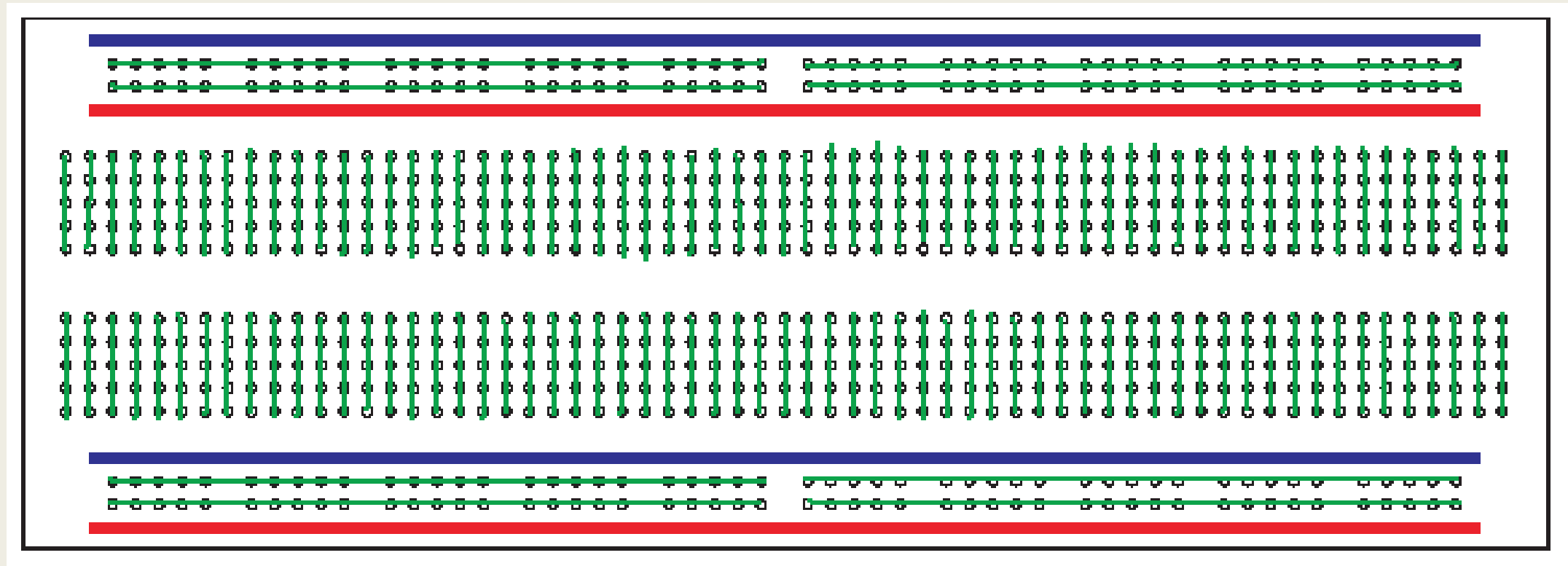


- A Breadboard is simply a *board for prototyping* or building circuits on. It allows us to place components and connections on the board to *make circuits without soldering*. The holes in the breadboard take care of the connections by physically holding onto parts or wires where we put them and *electrically connecting them inside the board*.





# Prototyping Essentials - Breadboard



Internal Connections of a Breadboard

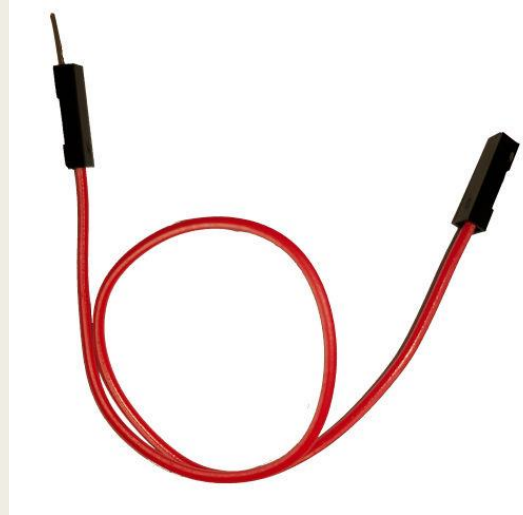
# Prototyping Essentials – Jumper Wires

- Jumper wires* are simply wires that have **connector pins at each end**, allowing them to be used to connect two points to each other **without soldering**. Jumper wires are typically used with breadboards and other prototyping tools in order to make it **easy to change a circuit as needed**.



## Male to Male

Notice the PIN-type connectors on both the ends



## Male to Female

Notice the PIN-type connector in one end and socket type on the other end



## Female to Female

Notice the socket type connectors on both the ends

# Common Electronic Components - Resistors

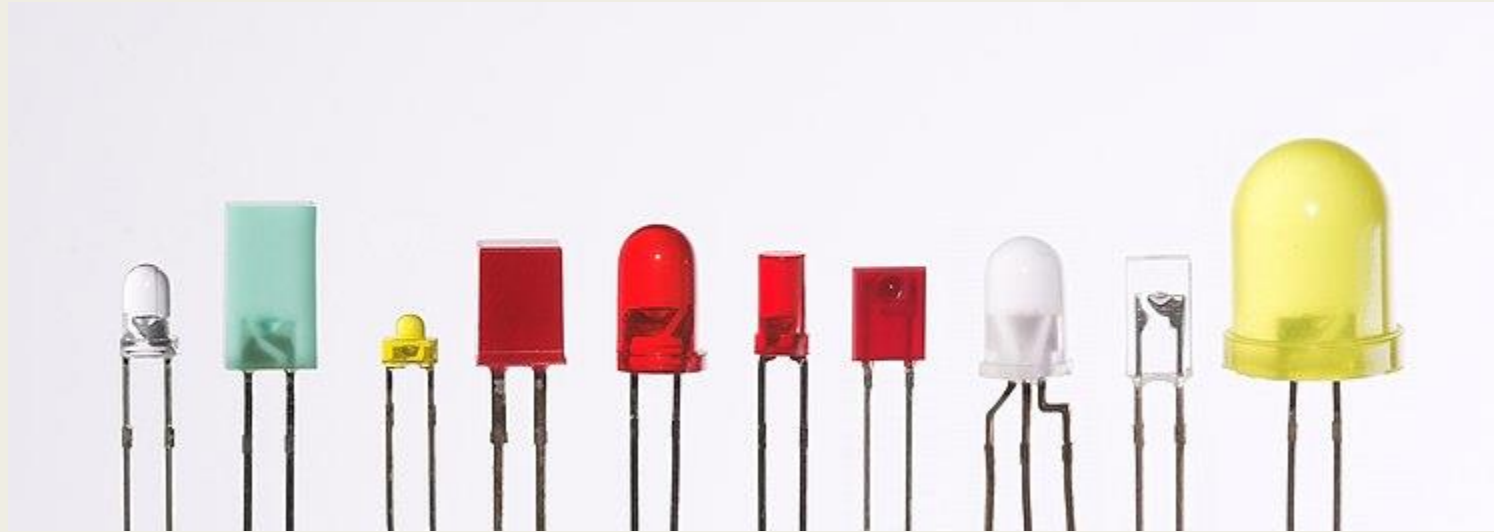
- **Resistors** – These are electrical components that limit or regulate the flow of current in an electronic circuit. They can also be used to **adjust signal levels, divide voltages, bias active elements, and terminate transmission lines, among other uses.**



Depending upon their intended application and material of construction, resistors can come in different shapes and sizes.

# Common Electronic Components - LEDs

- **LED** – A *light-emitting diode (LED)* is a semiconductor device that emits light when an electric current flows through it. They are widely used as a standard source of light in electrical equipment. It has a wide range of applications ranging from our mobile phones to large appliances like air conditioners.



Depending upon their intended application LEDs can also come in different shapes and sizes.



# Common Electronic Components - Switches

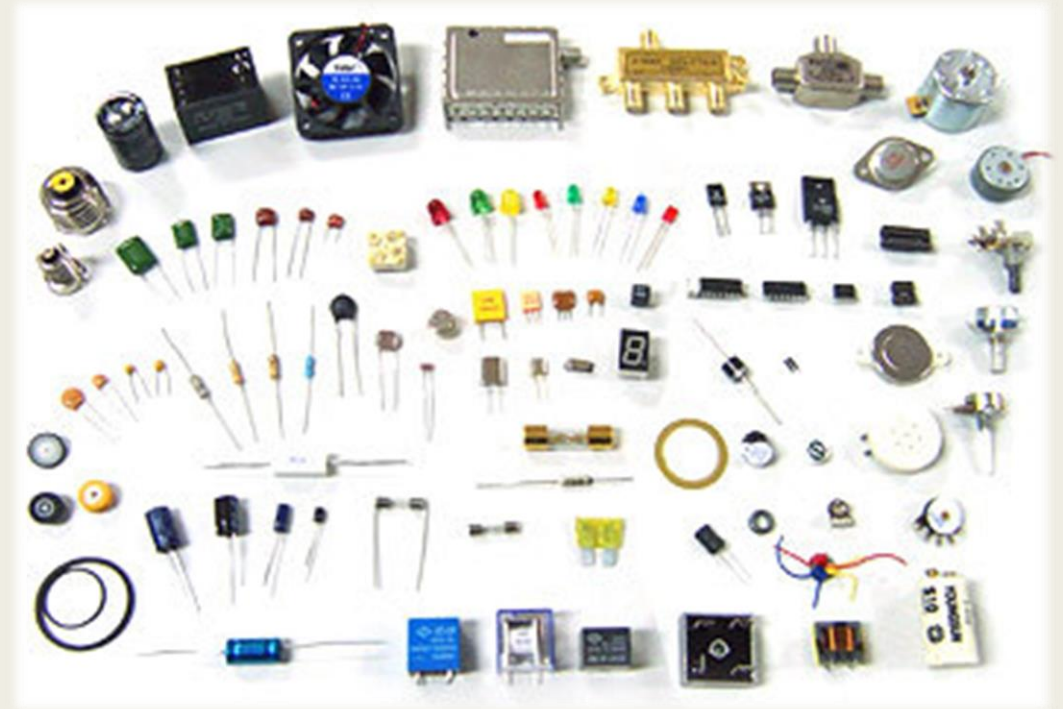
- **Switch** – An electric switch is a device – usually *electromechanical* – used to open and close an electric circuit. This disables and enables the flow of electric current, respectively.



Depending upon their intended application Switches can also come in different shapes and sizes.

# Common Electronic Components

- *There are several other components but we have discussed only the ones that we will be using in our project.*
- *Some other components – Capacitors, Transistors, MOSFETs, Triacs, Inductors, **Integrated Circuit (IC)**, Fuse, Transformer, Relay, Circuit Breaker, etc.*
- *Keep in mind that **Microcontrollers are a type of Integrated Circuits (IC).***

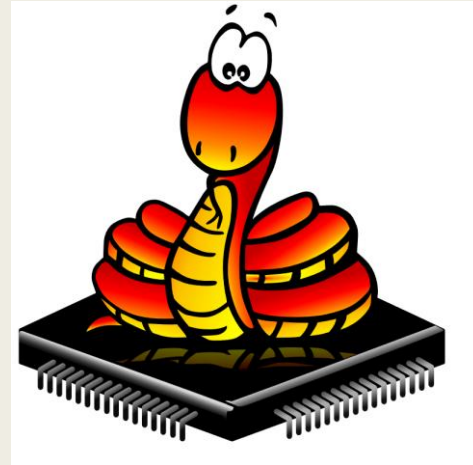


# Integrated Development Environment (IDE)

- An *integrated development environment (IDE)* is a software application that helps programmers develop software code efficiently.
- It *increases developer productivity* by combining capabilities such as *software editing, building, testing, and packaging* in an *easy-to-use application*.
- Just as *writers use text editors* and *accountants use spreadsheets*, *software developers use IDEs* to make their job easier.
- We will be using the *Thonny*. Thonny is an integrated development environment for Python that is designed for beginners.

# Micropython – Our choice of language

- **MicroPython** is a lean and efficient implementation of the **Python 3** programming language that includes a **small subset** of the Python standard library and is **optimized to run on microcontrollers** and in **constrained environments**.



Micropython Logo

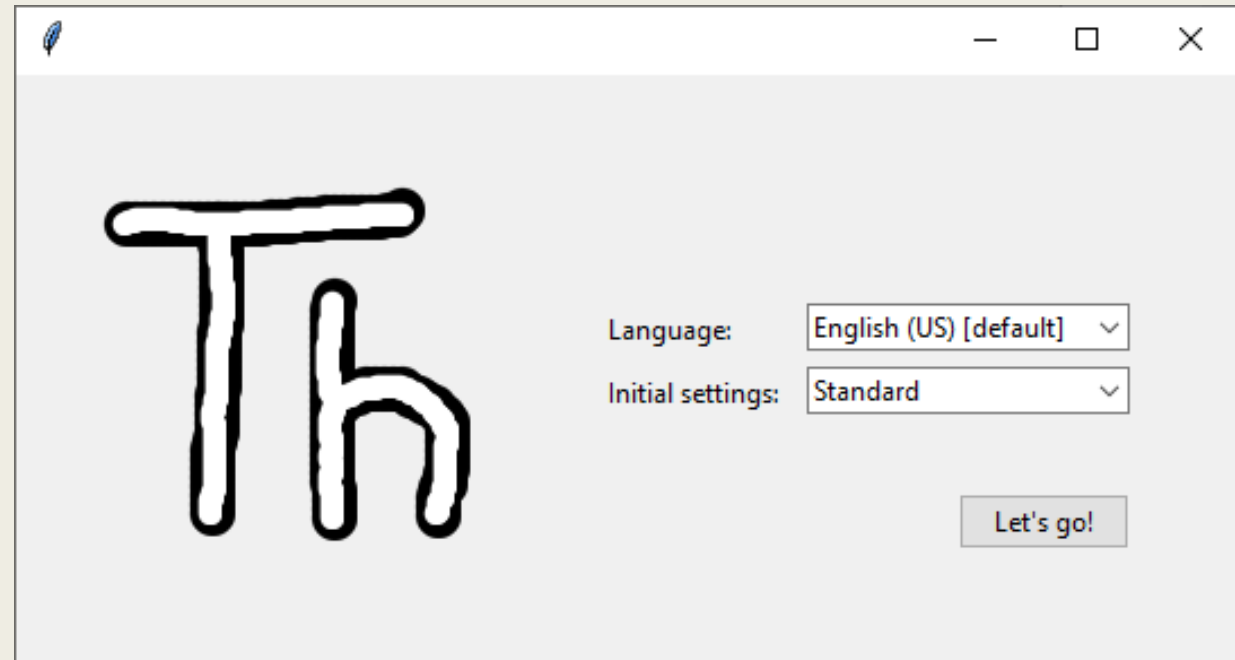
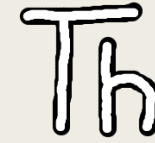


# Flashing the firmware to the ESP8266

- **Firmware** is a software that *provides basic machine instructions* that allow the hardware to *function and communicate with other software* running on a device.
- For our Micropython code to be executed in the target development board, *we must first upload the proper firmware.*
- The firmware depends on the specifications of the target board *like architecture, family, RAM, ROM etc.*

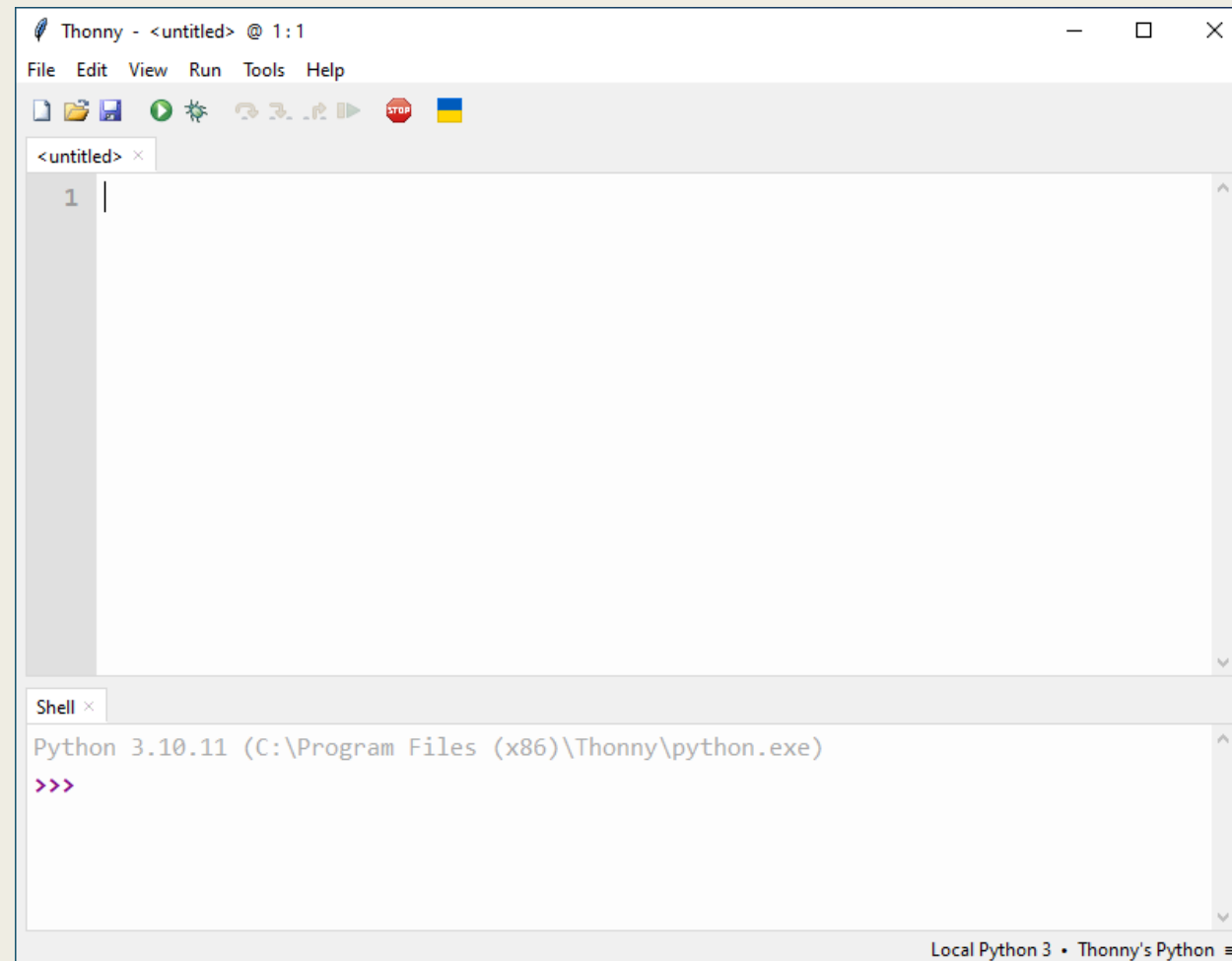
# Flashing the firmware to the ESP8266

1. Double-click on the **Thonny** icon on your desktop.
2. Select the **Language** as “English (US) [default]” and the **Initial settings** as “Standard”
3. Then click on **Let's Go!**



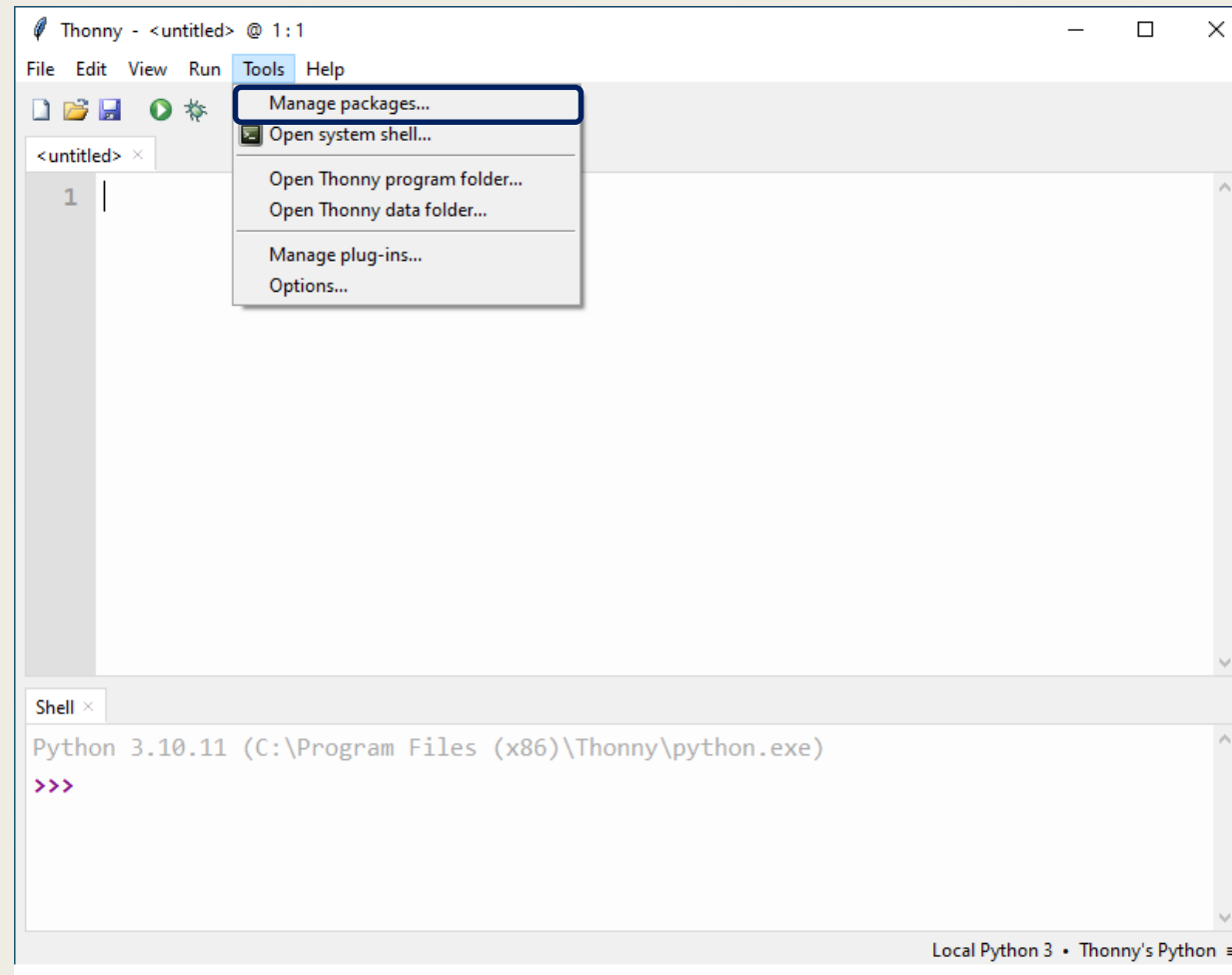
# Flashing the firmware to the ESP8266

4. *You will be displayed a window like this.*



# Flashing the firmware to the ESP8266

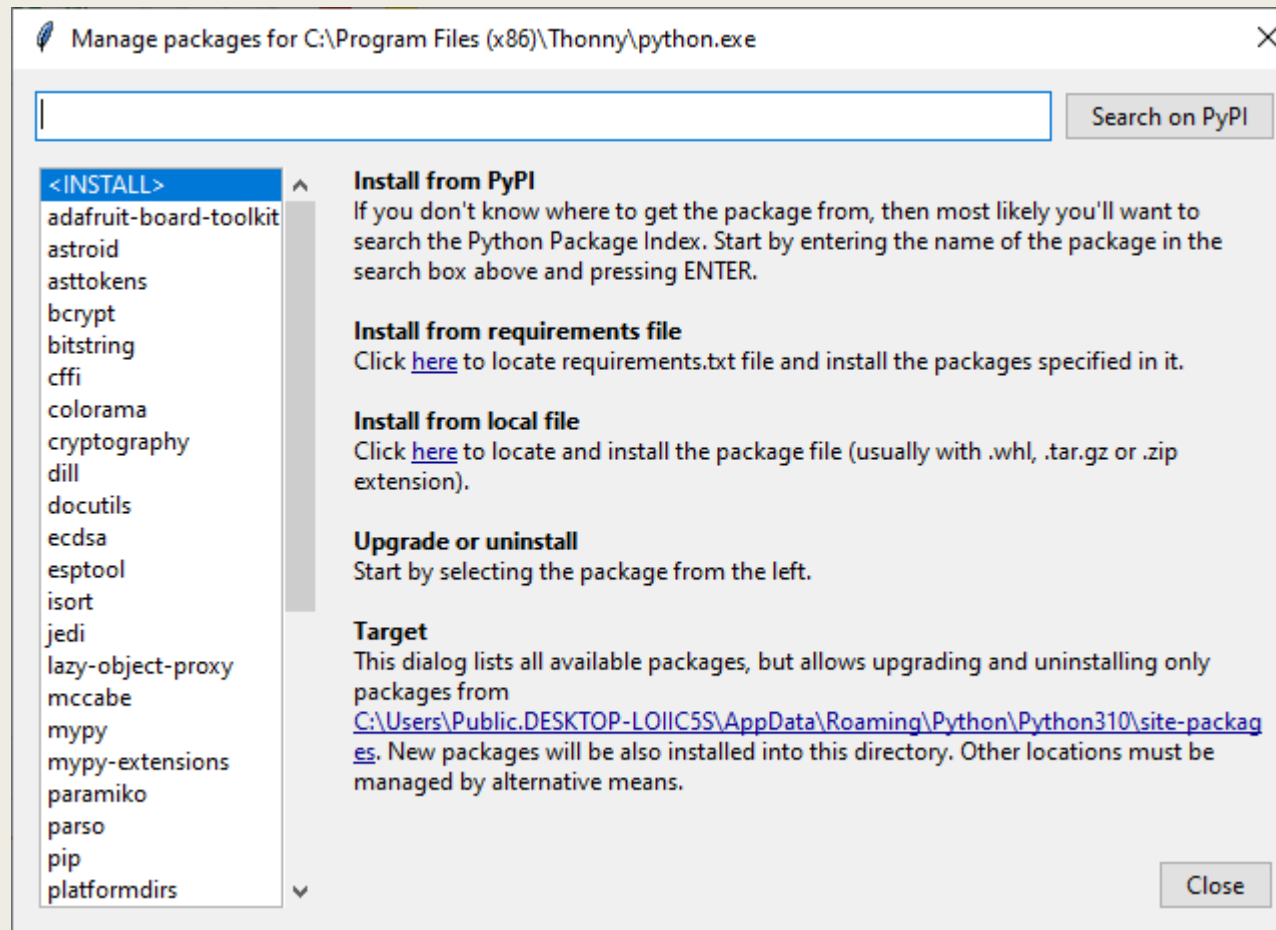
5. Next, click on **Tools** and then on **Manage packages...**





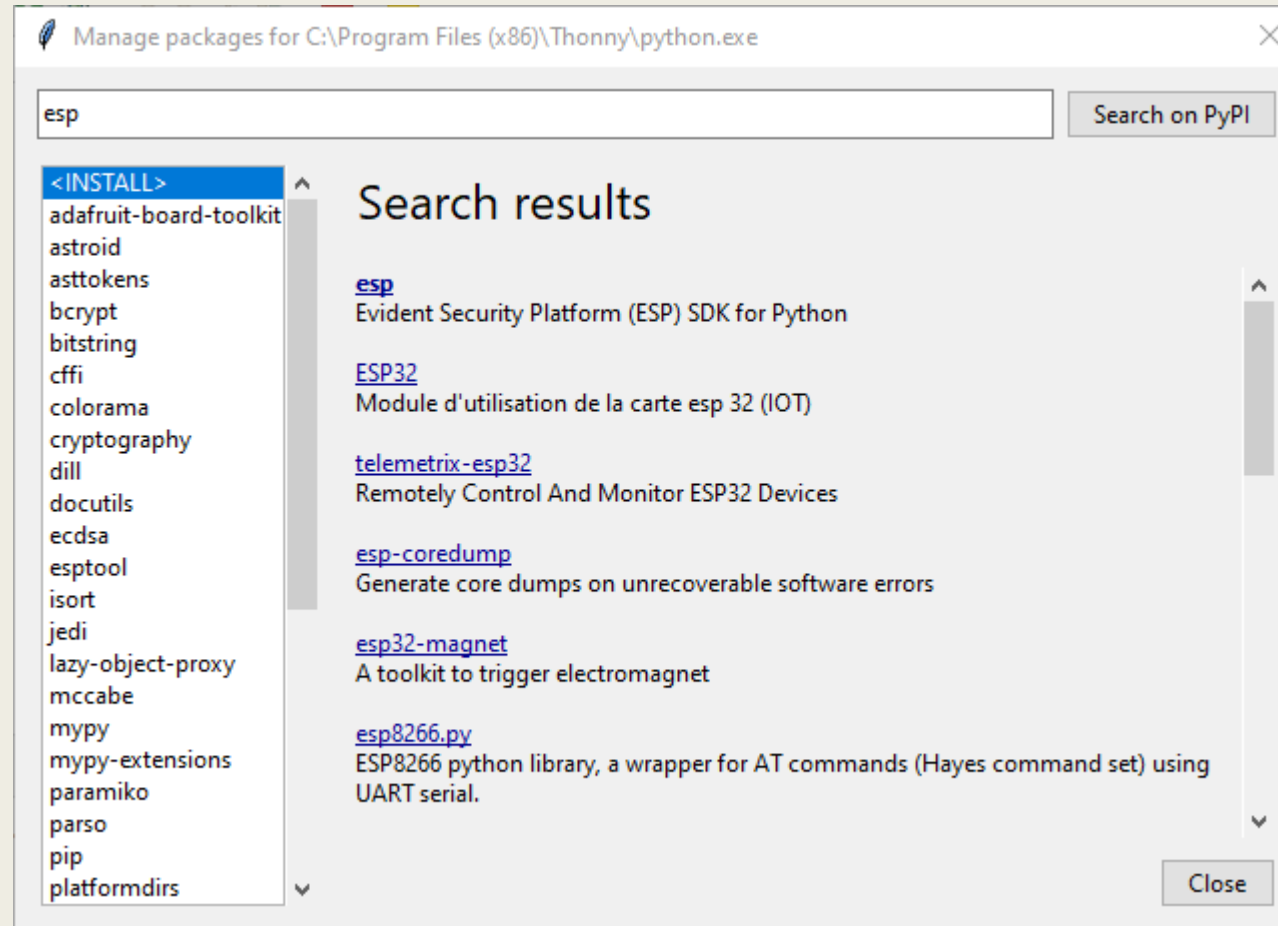
# Flashing the firmware to the ESP8266

6. You will then see a window like this. Search for the **esp** package.



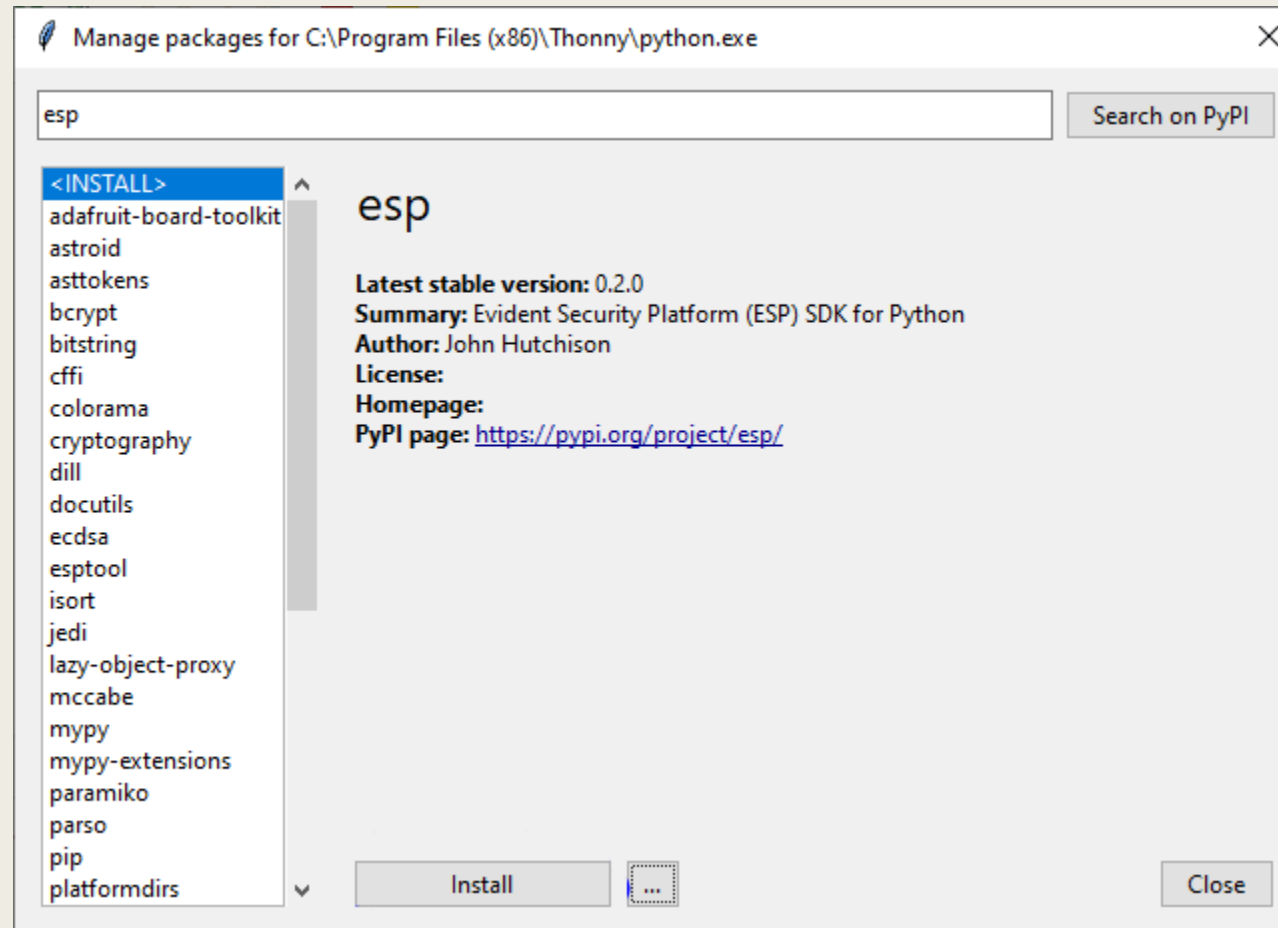
# Flashing the firmware to the ESP8266

7. *The next window should look like this. Now click on **esp**.*



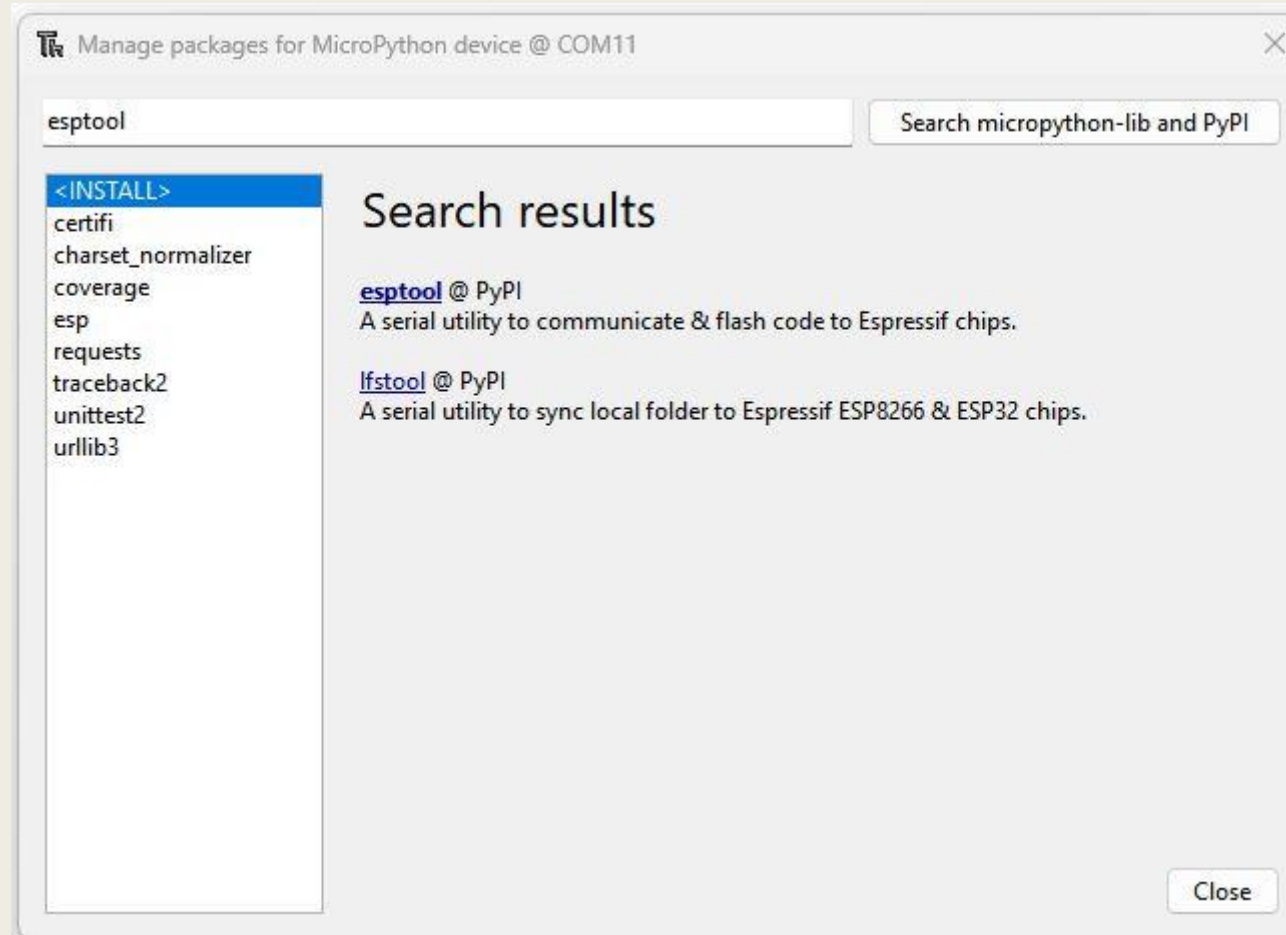
# Flashing the firmware to the ESP8266

8. *The next window should look like this. Click on **Install** and wait for the program to complete installation. It will take some time.*



# Flashing the firmware to the ESP8266

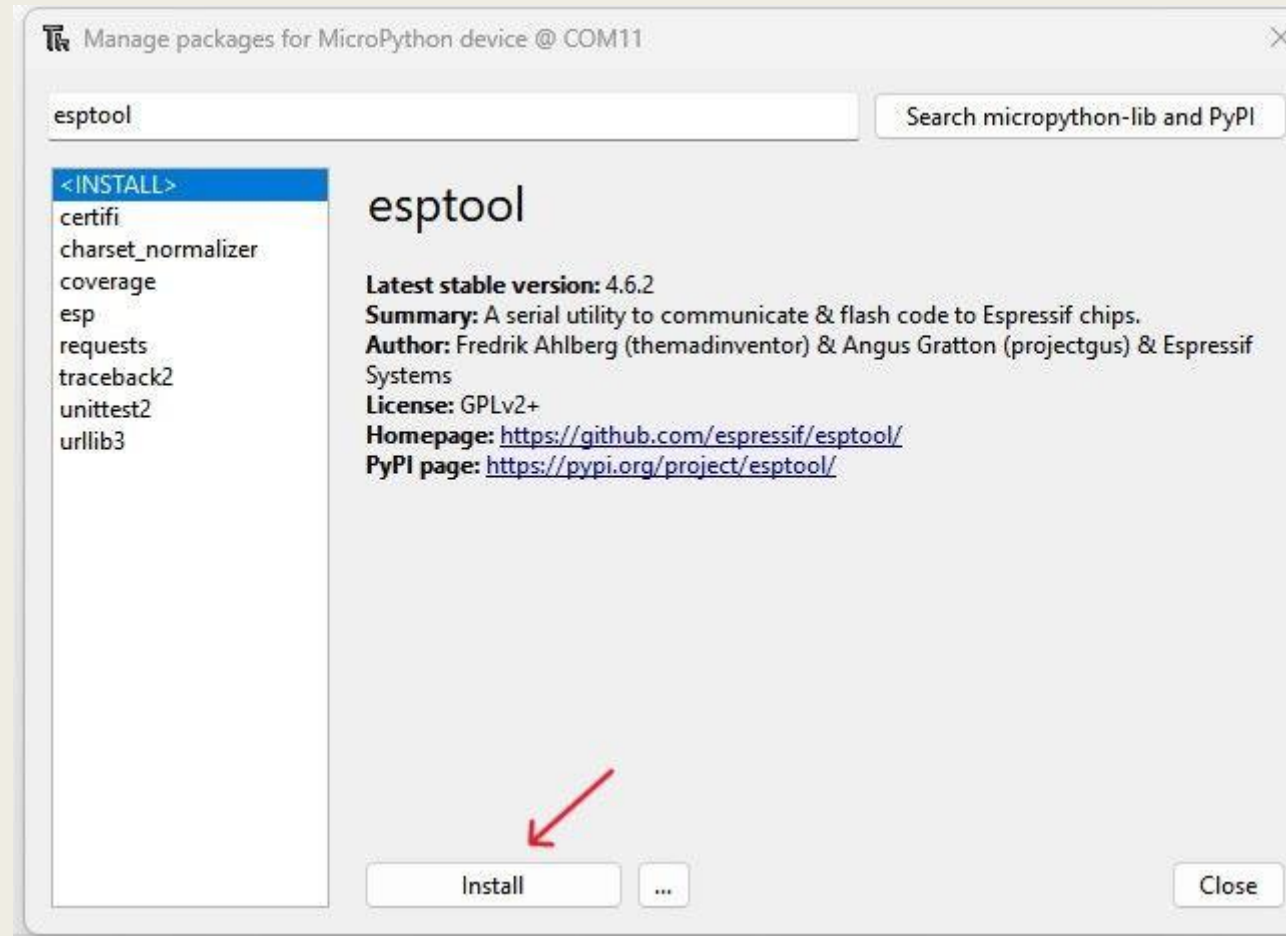
9. Now we need to install the **esptool** package. The process is same.





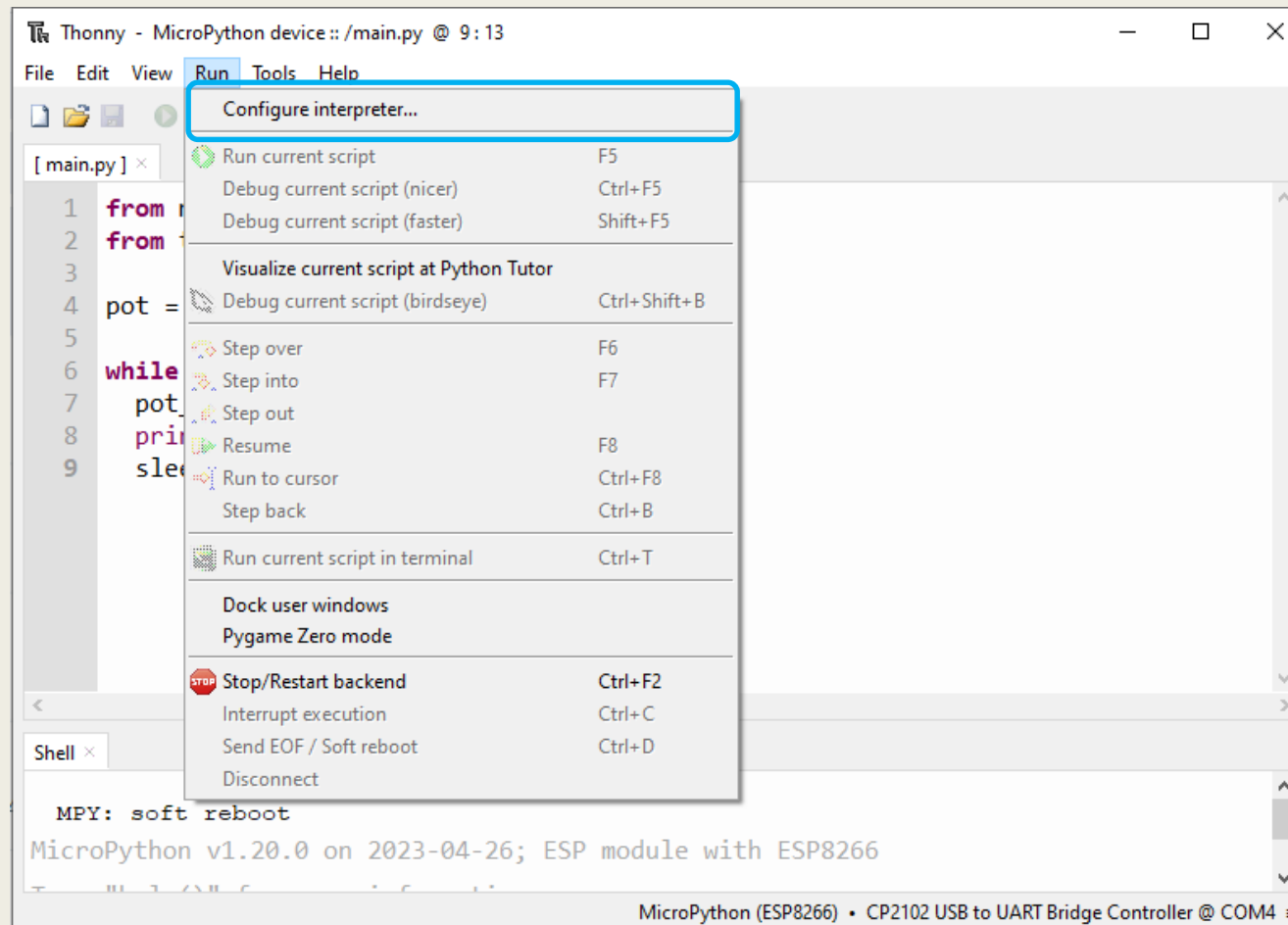
# Flashing the firmware to the ESP8266

9. Now we need to install the **esptool** package. The process is same.



# Flashing the firmware to the ESP8266

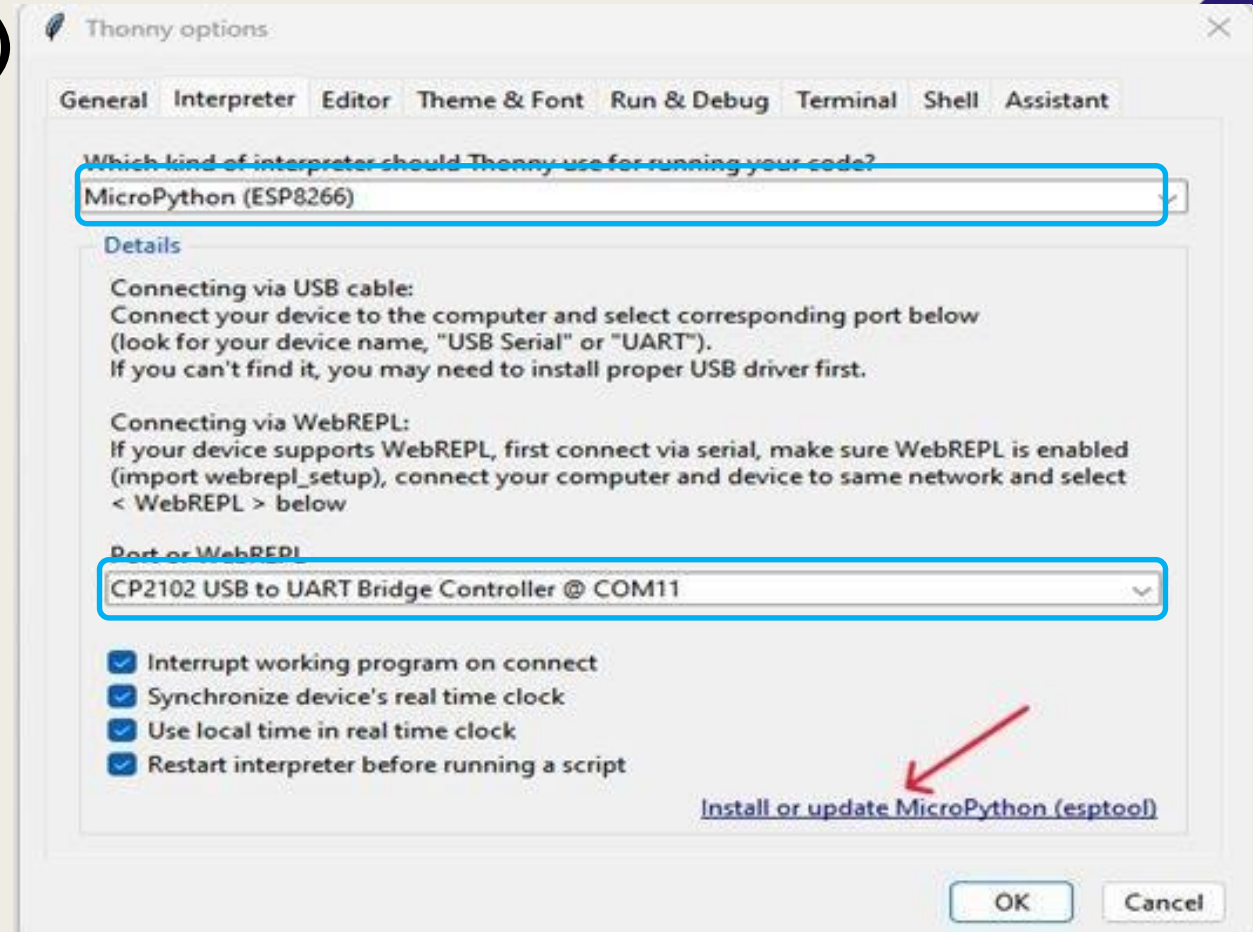
10. Now we need to configure the interpreter. Click on **Run** and then click on **Configure interpreter...**



# Flashing the firmware to the ESP8266

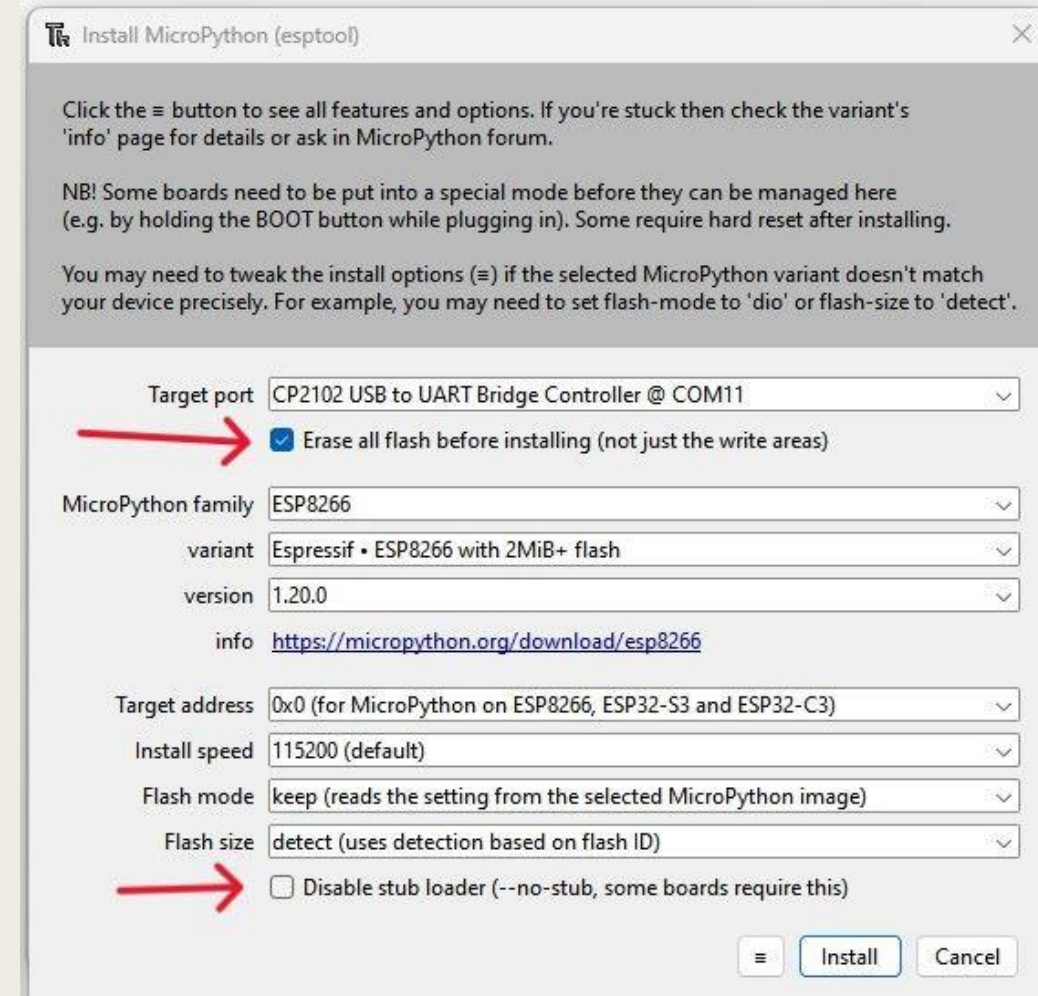
11. Select **MicroPython (ESP8266)** and **CP2102 USB to UART Bridge Controller @ COMXX** from the dropdown menu. The last digits of your COM port might vary.

12. Now, click on **Install or update MicroPython (esptool)**.



# Flashing the firmware to the ESP8266

13. Make sure the **Erase all flash before installing** option is checked.
14. Select other options as shown in the image.
15. Also, uncheck the **Disable stub Loader** option.



Install MicroPython (esptool)

Click the ≡ button to see all features and options. If you're stuck then check the variant's 'info' page for details or ask in MicroPython forum.

NB! Some boards need to be put into a special mode before they can be managed here (e.g. by holding the BOOT button while plugging in). Some require hard reset after installing.

You may need to tweak the install options (≡) if the selected MicroPython variant doesn't match your device precisely. For example, you may need to set flash-mode to 'dio' or flash-size to 'detect'.

Target port: CP2102 USB to UART Bridge Controller @ COM11

☒ Erase all flash before installing (not just the write areas)

MicroPython family: ESP8266

variant: Espressif • ESP8266 with 2MiB+ flash

version: 1.20.0

info: <https://micropython.org/download/esp8266>

Target address: 0x0 (for MicroPython on ESP8266, ESP32-S3 and ESP32-C3)

Install speed: 115200 (default)

Flash mode: keep (reads the setting from the selected MicroPython image)

Flash size: detect (uses detection based on flash ID)

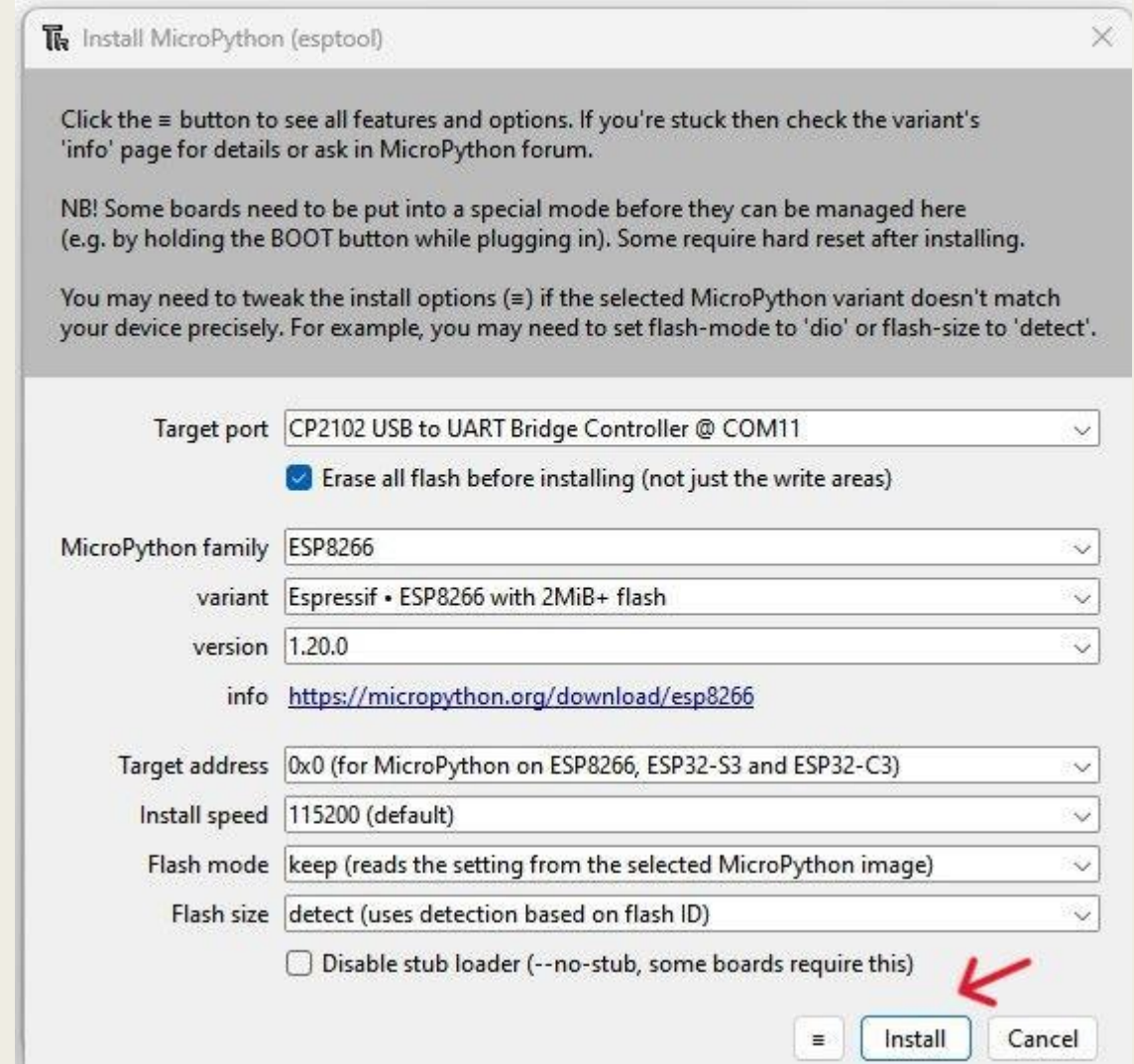
☐ Disable stub loader (--no-stub, some boards require this)

≡ Install Cancel

# Flashing the firmware to the ESP8266

16. Click on the **Install** button and your installation should proceed.

17. After successful installation close all pop-up windows.



Install MicroPython (esptool)

Click the ≡ button to see all features and options. If you're stuck then check the variant's 'info' page for details or ask in MicroPython forum.

NB! Some boards need to be put into a special mode before they can be managed here (e.g. by holding the BOOT button while plugging in). Some require hard reset after installing.

You may need to tweak the install options (≡) if the selected MicroPython variant doesn't match your device precisely. For example, you may need to set flash-mode to 'dio' or flash-size to 'detect'.

Target port: CP2102 USB to UART Bridge Controller @ COM11

☒ Erase all flash before installing (not just the write areas)

MicroPython family: ESP8266

variant: Espressif • ESP8266 with 2MiB+ flash

version: 1.20.0

info: <https://micropython.org/download/esp8266>

Target address: 0x0 (for MicroPython on ESP8266, ESP32-S3 and ESP32-C3)

Install speed: 115200 (default)

Flash mode: keep (reads the setting from the selected MicroPython image)

Flash size: detect (uses detection based on flash ID)

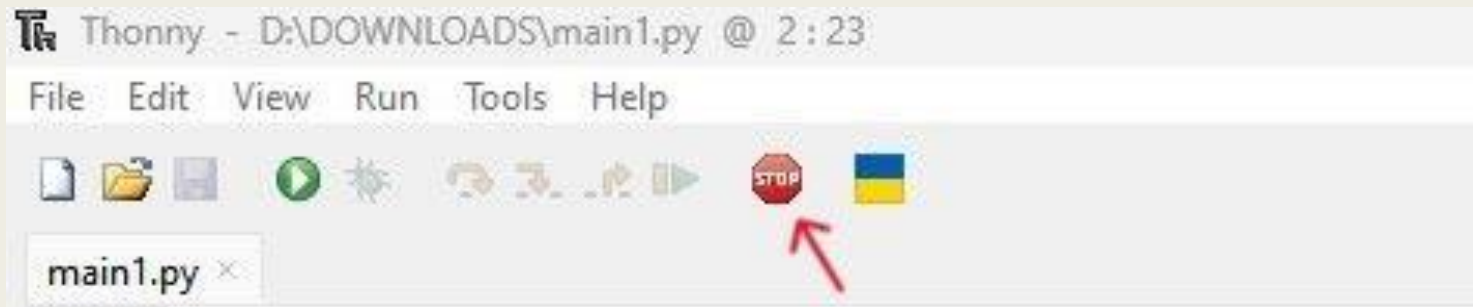
☐ Disable stub loader (--no-stub, some boards require this)

≡ Install Cancel



# Flashing the firmware to the ESP8266

18. Click the **STOP** button and then disconnect your **NodeMCU**.



# Flashing the firmware to the ESP8266

19. Connect the **NodeMCU** again, and then import **ESP** library in shell window.

20. Now, type “**esp.check\_fw()**” command in next line and press enter.

```
Shell x
KeyboardInterrupt.

MicroPython v1.20.0 on 2023-04-26; ESP module with ESP8266
Type "help()" for more information.
>>>
>>>

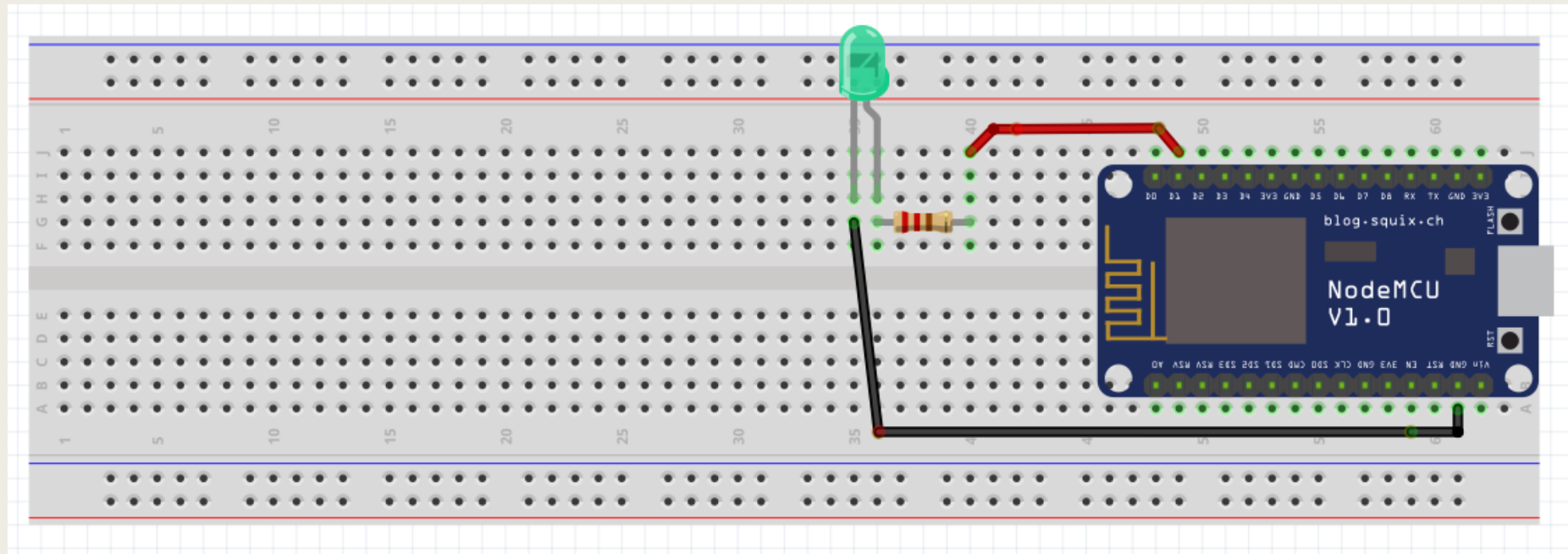
MPY: soft reboot
MicroPython v1.20.0 on 2023-04-26; ESP module with ESP8266
Type "help()" for more information.
>>> import esp
>>> esp.check_fw()

size: 634000
md5: 4796b17c5b672462934dd33d0e4a1ce7
True
>>>
```

# Connecting the components

1. *Connect the components like the diagram below.*

**Attention: Please disconnect the USB Cable from your NodeMCU before making the connections on the breadboard.**



# The code to turn an LED on and off

1. *First we import the required modules.*

```
1 from machine import Pin
2 from time import sleep
3
4 led_pin = Pin(5, Pin.OUT)
5
6 while True:
7     led_pin(1)
8     sleep(1)
9     led_pin(0)
10    sleep(1)
```

*More information on Micropython libraries can be found in the link below:*  
<https://docs.micropython.org/en/latest/library/index.html>

# The code to turn an LED on and off

2. Then we create an instance of the **Pin** class with pin number 5 set as an output pin. It is then assigned to the variable **led\_pin**.

```
1 from machine import Pin
2 from time import sleep
3
4 led_pin = Pin(5, Pin.OUT)
5
6 while True:
7     led_pin(1)
8     sleep(1)
9     led_pin(0)
10    sleep(1)
```



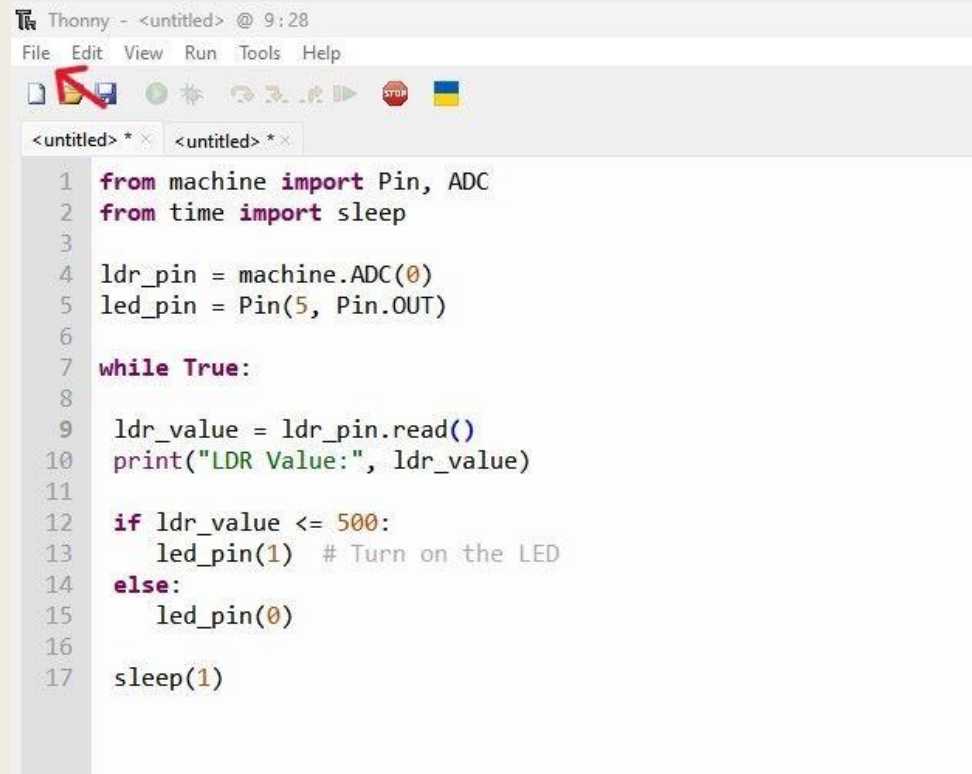
# The code to turn an LED on and off

3. *Then we create an infinite loop and inside it, we toggle the output state of the LED with a delay of 1 second. This should turn on and off the LED continuously.*

```
1 from machine import Pin
2 from time import sleep
3
4 led_pin = Pin(5, Pin.OUT)
5
6 while True:
7     led_pin(1)
8     sleep(1)
9     led_pin(0)
10    sleep(1)
```

# Uploading the Code

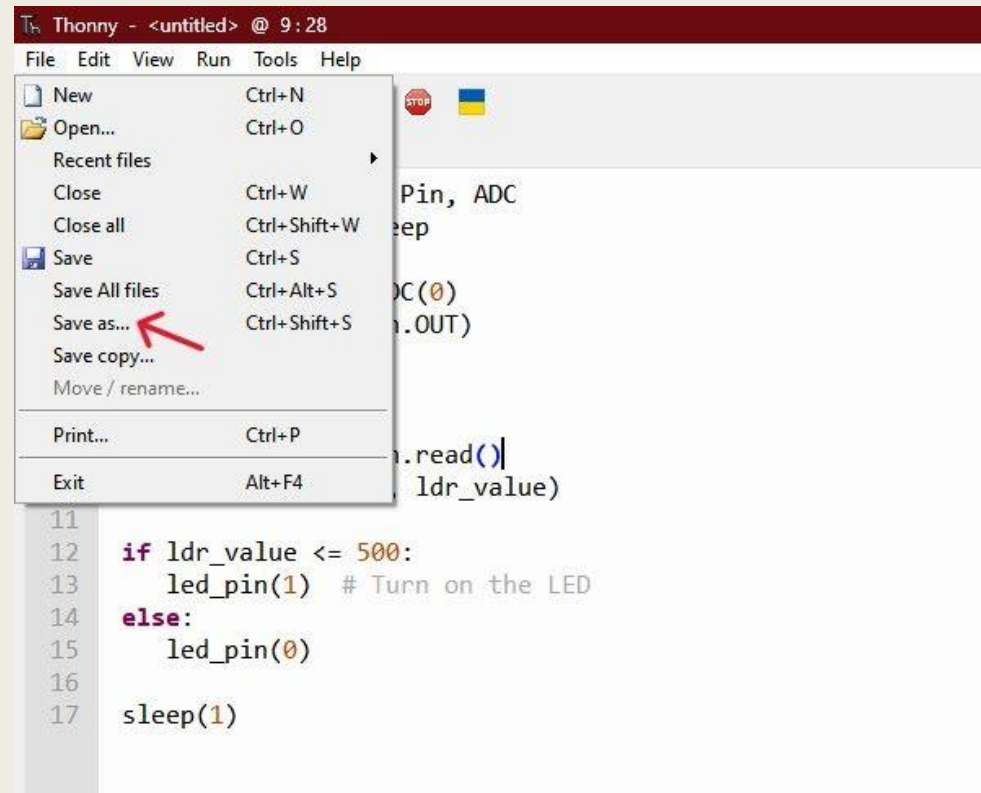
1. Click on the **FILE** menu as shown.



```
Thonny - <untitled> @ 9:28
File Edit View Run Tools Help
<untitled> *x <untitled> *x
1 from machine import Pin, ADC
2 from time import sleep
3
4 ldr_pin = machine.ADC(0)
5 led_pin = Pin(5, Pin.OUT)
6
7 while True:
8
9     ldr_value = ldr_pin.read()
10    print("LDR Value:", ldr_value)
11
12    if ldr_value <= 500:
13        led_pin(1) # Turn on the LED
14    else:
15        led_pin(0)
16
17    sleep(1)
```

# Uploading the Code

2. Click on “**Save as**” from a list of options in the file menu.



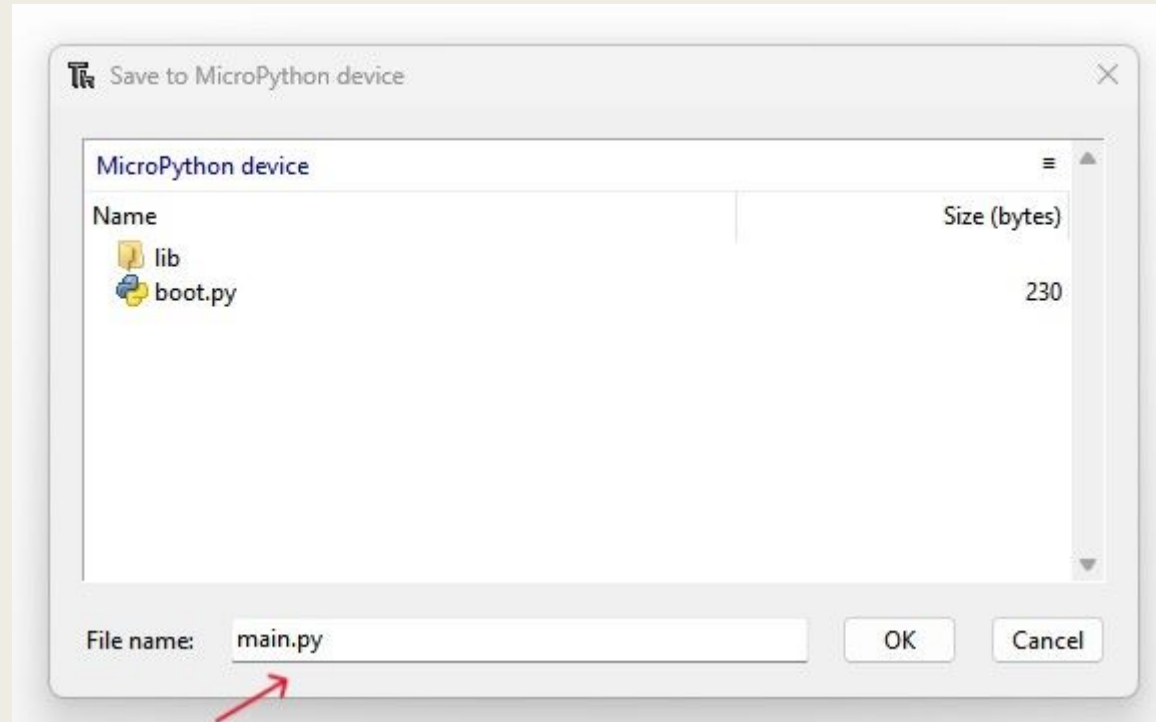
# Uploading the Code

3. *Select the location as “MicroPython device”.*



# Uploading the Code

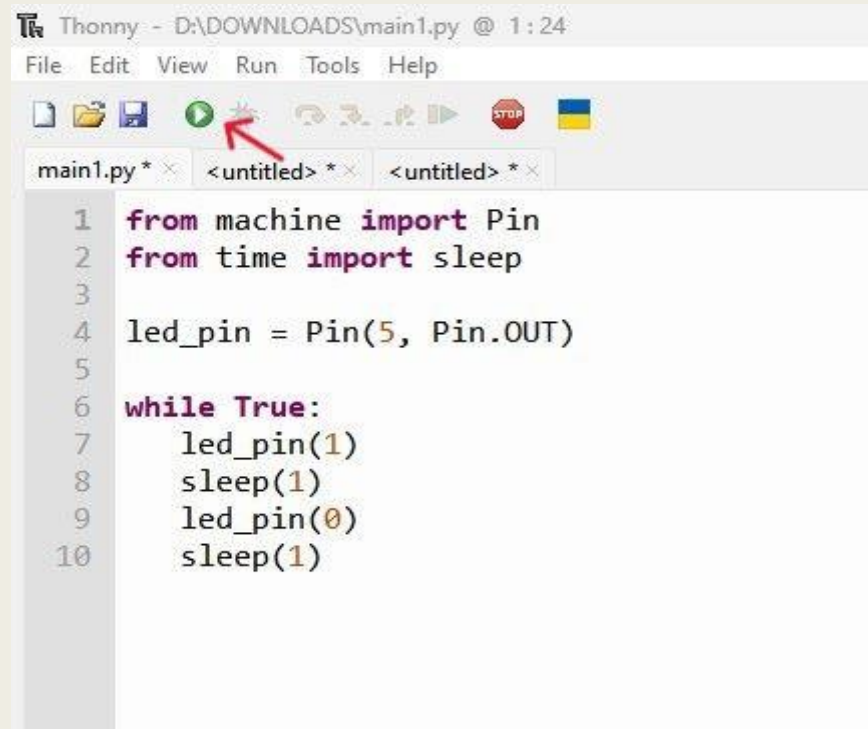
4. Give the name “**main.py**” to the file and then click on the **OK** button.





# Uploading the Code

5. Then click on the **RUN** button as shown. Your code will get uploaded to **NodeMCU**.



```
Thonny - D:\DOWNLOADS\main1.py @ 1:24
File Edit View Run Tools Help
[Icons] [Run] [Stop] [Flag]
main1.py * <untitled> * <untitled> *
1 from machine import Pin
2 from time import sleep
3
4 led_pin = Pin(5, Pin.OUT)
5
6 while True:
7     led_pin(1)
8     sleep(1)
9     led_pin(0)
10    sleep(1)
```

# Common pitfalls (Debugging)

LED not blinking??

## 1. Double-check the connections.

- Connections might be done on a wrong pin
- A wire might be broken from inside
- There might be some loose connections

## 2. Verify the code

- Check if the pin assignment is done correctly
- Check the indentations
- Check if the filename is **main.py**

## 3. Verify whether the code is uploaded

**Thank You!!!**