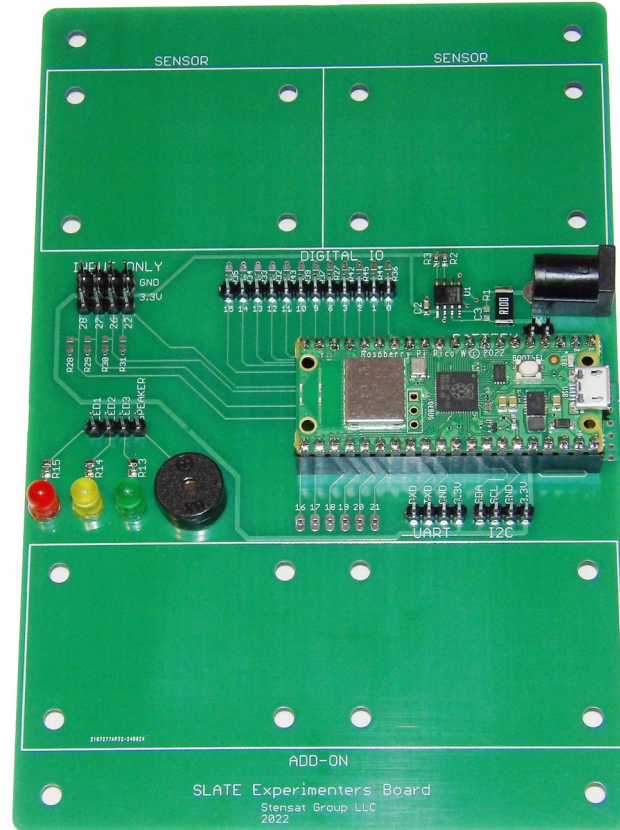


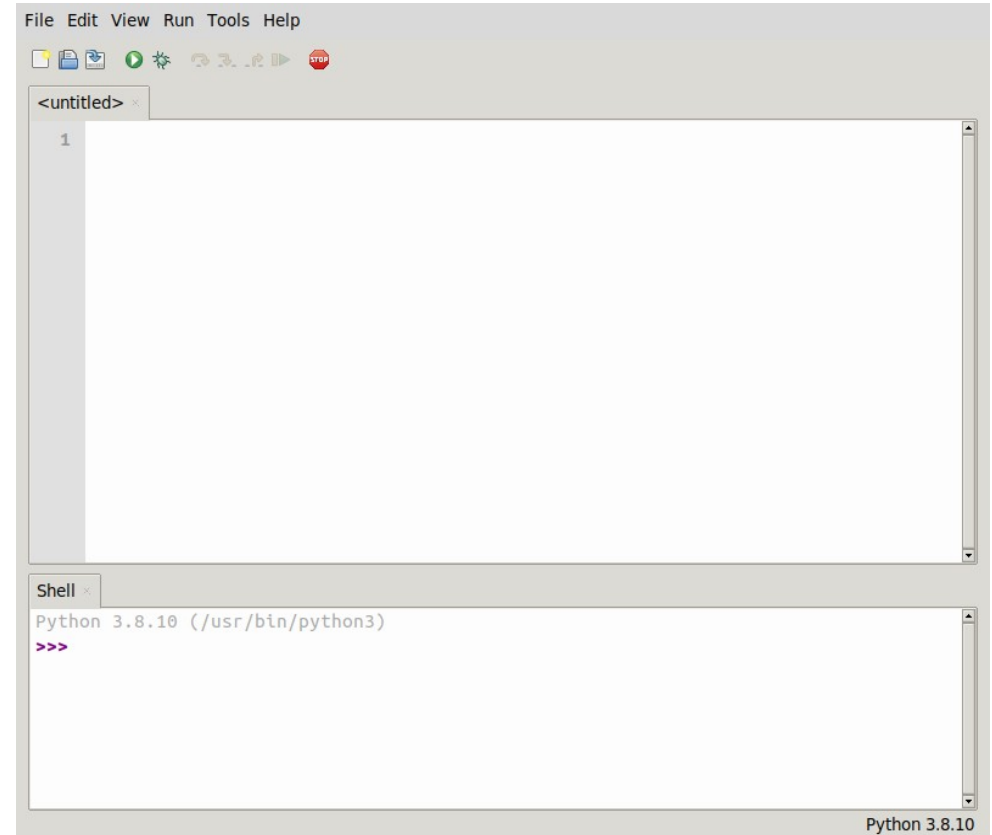
SLATE Programming



- The SLATE is preloaded with MicroPython.
- MicroPython is a reduced version of Python 3 that allows direct control of hardware and uploading a program to automatically run.
- To communicate with the board, an Integrated Development Environment (IDE) will be used called Thonny. Thonny can be downloaded from <https://thonny.org>

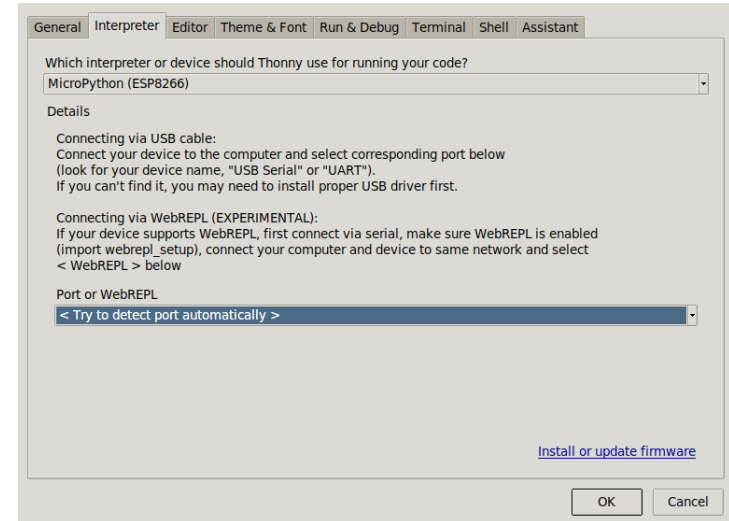
Micropython Interface

- Thonny includes a text editor and a command interface. The top text area is where programs can be entered.
- The bottom shell area is the command interface where you can type python instructions and get immediate results. It is also the space where any text printed from the program is displayed and error messages.



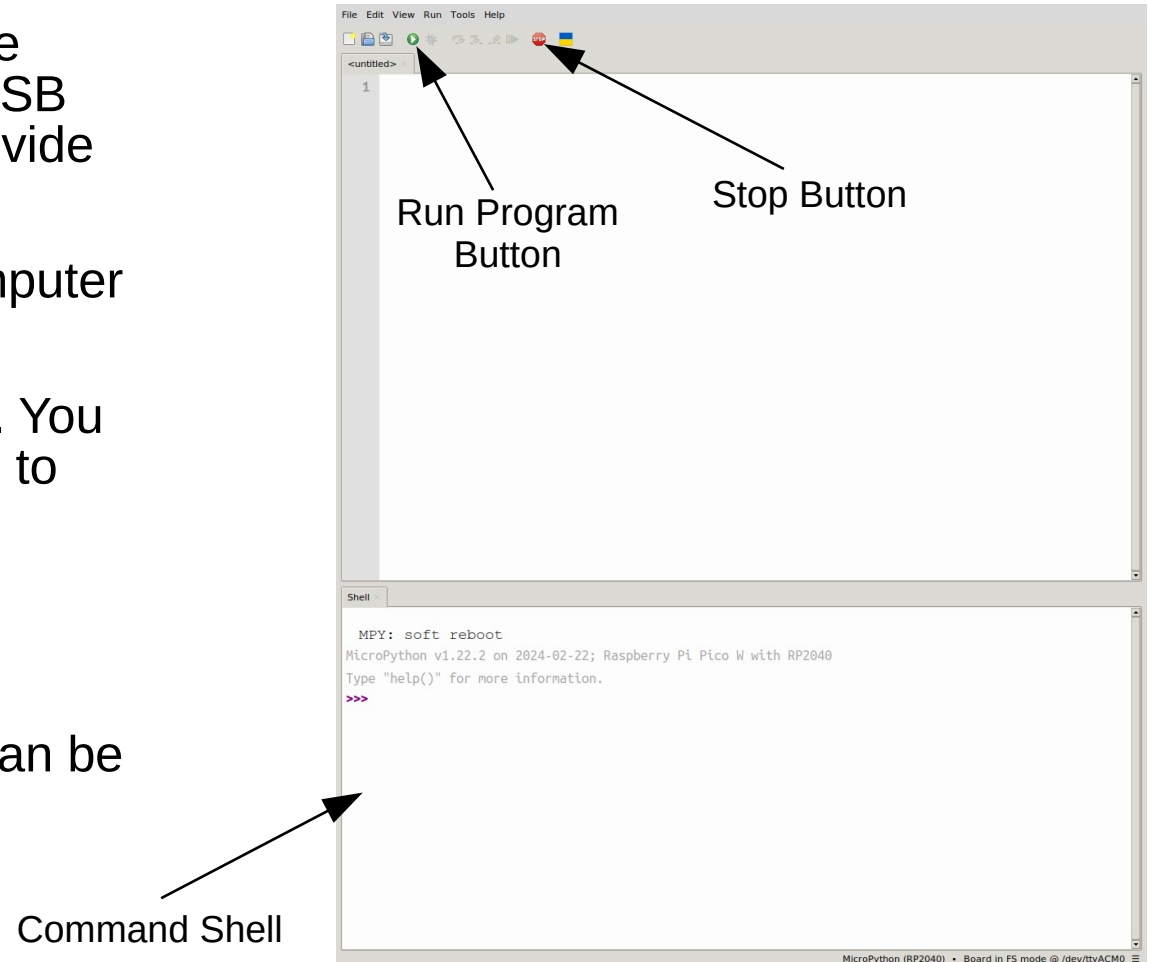
Micropython Interface

- To configure Thonny to communicate with the board, select the **Tools** menu and select **Options...**
- Select the **Interpreter** tab and select the interpreter **MicroPython (Rp2040)**.
- Below, select Port to be **<Try to detect port automatically>**.
- Click on OK.
- A prompt **>>>** should appear in the command shell.



Micropython Interface

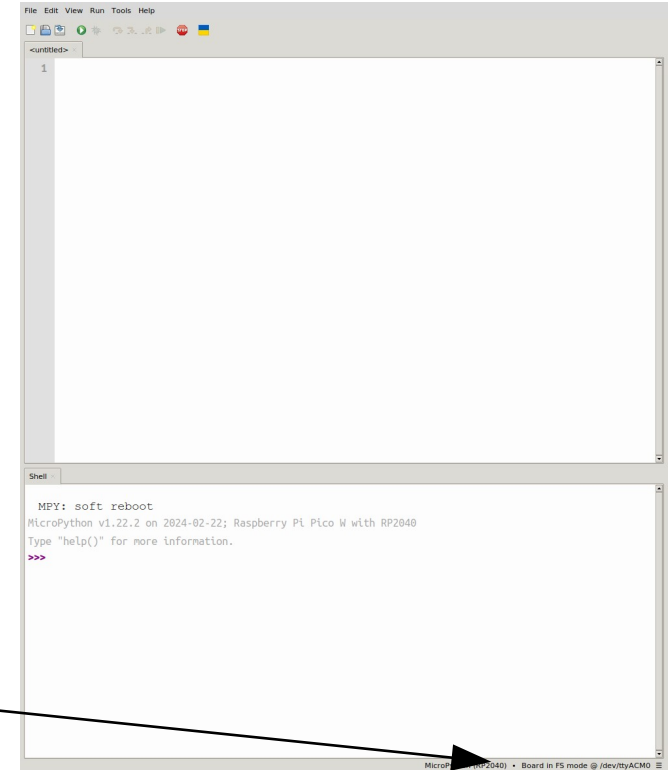
- You do not need to turn on the board when connecting the USB cable. The USB cable will provide power.
- Connect the board to the computer with a USB cable.
- Thonny should try to connect. You can click on the **STOP** Button to make the connection.
- In the shell a prompt `>>>` will appear. That indicates communications has been established and commands can be entered.



Micropython Interface

- If a prompt does not appear, Click on the bottom of the window where shown. It should pop up a list of devices to connect. Select the one that looks similar in the picture.

Pop up to select device



Testing out Python

- In the shell, enter the command below to the right of >>>.

```
print("hello world")
```

- The text should be displayed under the command.
- Now enter the following:

```
while 1:  
    print("Hello world")
```

- The text in between the quotes is called a string.
- You will notice that the command console automatically indented the cursor.
- After entering the print command, the cursor is indented still.
- All code indented after the **while 1:** will be executed repeatedly.
- Press the **Backspace** key and then **Enter**. You should see the print statement running repeatedly.
- Click on the Stop button to stop the program.

Variables and Numbers

- Python supports math operations and variables.
- Variables are objects that store information such as numbers and text. Variables can be made up of any combination of letters and numbers with the one rule that the variable name must start with a letter.
- Try the following:

```
a = 5  
b = 12  
c = a + b  
print(c)
```
- Try other math operations such as `-` for subtraction, `*` for multiply and `/` for divide.

Conditional Programming

- Conditional programming is using the if statement with a comparison. With the variables from the last page, try the following:

 if a > b:
 print("A is greater than B")

 else:
 print("B is greater than A")
- The indented lines of code are executed if the comparison statement is true. If the comparison is not true, the code after the **else** statement is executed.
- Remember to press the **Backspace** key and the **Enter** key to get that code to execute.

Conditional Programming

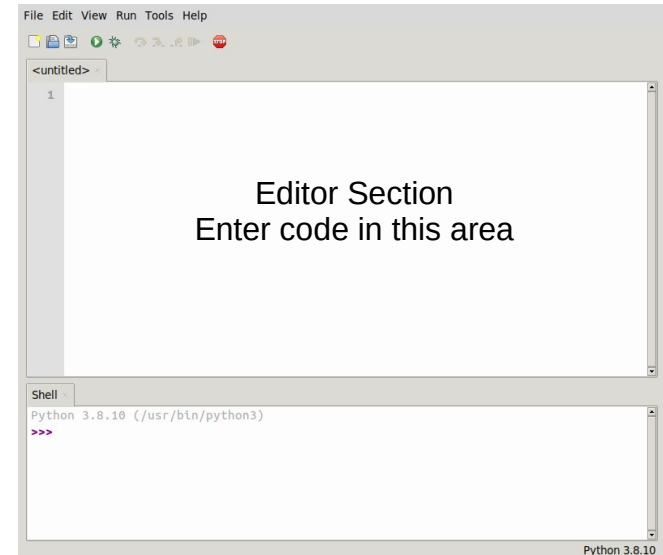
- These are all the conditional comparisons available
 - $a > b$ a is greater than b
 - $a \geq b$ a is greater than or equal to b
 - $a < b$ a is less than b
 - $a \leq b$ a is less than or equal to b
 - $a == b$ a is equal to b
 - $a != b$ a is not equal to b

Saving and Uploading Files

- The board does have some storage for program files.
- In the editor, enter the program to the right.
- Save the file by using the **File** menu and select **Save**. A pop up window will appear asking if you want to save on the **computer** or **Raspberry PI PICO**. Select **Raspberry PI PICO**. This will save the program on the board.
- Enter the file name **test1.py** and click on the **OK** button.
- Click on the green **run** button.

```
import time

while 1:
    print("This is a test")
    time.sleep(0.5)
```



Saving and Uploading Files

- In a couple seconds, the program will start running and you should see the message being displayed every half second.
- To stop the program, select the shell area and click on the **STOP** button at the top of the window.

```
import time

while 1:
    print("This is a test")
    time.sleep(0.5)
```

Importing Functions

- The first line of the program loads a module called time. This module adds functions to the python language.
- Many modules can be imported into the program to add more functions.
- To access the function in a module, you need to use the module name and the function separated with a period.
- The program shown shows the **sleep()** function being used from the time module.
- **sleep()** stops the program for the specified number of seconds. Fractional seconds can be used.

```
import time

while 1:
    print("This is a test")
    time.sleep(0.5)
```

Importing Functions

- You can also specify a specific function from a module to be imported.
- The program to the right shows importing the **sleep()** function from the time module. Notice in the code, you do not need to include the time module name when using the **sleep()** function. This can save on typing.
- The only potential problem is if you import multiple modules this way, it is possible that more than one module uses the same name for a function. This will cause problems.

```
from time import sleep

while 1:
    print("This is a test")
    sleep(0.5)
```

List Variables

- Variables can include a list of values. In other languages, the variable would be called an array.

- Enter the following:

```
x = [12,5]  
print(x[0])  
print(x[1])
```

- Variable **x** is a list with two items in the list which happen to be numbers.
- You access each number in the list by specifying the name of the variable and a number in between the square brackets. The number between the square brackets is called an index and specifies which item in the list to access. The first item is at position 0.

Dictionary Object

- Another way of organizing data is in an object called a dictionary.
- it is a data structure where data is stored in key-value pairs. This lets you reference data by name.
- You can create an empty dictionary
- You can add to it by specifying the name in the bracket and assigning it a value. The value can be any type and you can mix different types of values.
- Enter the commands in the Thonny console.

```
dict = {}  
  
dict['var1'] = 437  
dict['var2'] = "hello"  
  
print(dict)  
print(dict['var2'])
```

Dictionary Object

- There are a few ways to access the values in the dictionary.
- The first is to assign it to a variable by as shown in the first line. It is similar as accessing a list element.
- What happens if a key that doesn't exist is referenced. Try out line 2. You should get an error.
- Another way is to use the get method. If the key exists, the value is displayed. Try line 3.
- If the key does not exist, the second argument is displayed. Try line 4.

```
dict['var2']
```

```
dict['var3']
```

```
dict.get('var2')
```

```
dict.get('var3', 'bad')
```

- Strings are a collection of letters also called text. A string is created by having a bunch of letters and numbers between quotes. The quotes can be double quotes or single quotes.

- Try the following:

```
'I am a single quote'
```

```
"I am a double quote"
```

- When entering both, both get displayed the same.
- A string can be assigned to a variable.

```
t = "this is a text string"
```

```
print(t)
```

More Coding Examples

- Try the program to the right.
- This shows how to implement math equations and display text and the value of a variable.
- The **while** statement is different here. There is a comparison. If the result of the comparison is true, the code indented under the **while** loop will execute. The code will execute until the comparison is false.
- Save the program on the board as **test2.py** and run the program. See what is the final value of variable **c**.
- Once **c** became larger than 25, the program while loop stopped.

```
import time

a = 12
b = 8
c = 0
while c < 25:
    c = c + a - b
    print("Result
is:",c)
    time.sleep(0.5)

print("Over 25: ",c)
```

More Coding Examples

- **print()** and **sleep()** are functions.
- All functions are formatted **as func()** with the parenthesis after the name of the function.
- Values and variables can be inserted between parenthesis. They are called arguments.
- The **sleep()** function takes one argument and that is a number stating the number of seconds to stop the program execution.
- **print()** takes multiple arguments separated by commas. Notice in the code that print has an equation as an argument. This is allowed.

```
import time

a = 12
b = 8
c = 0
while c < 25:
    c = c + a - b
    print("Result is:",c+b)
    time.sleep(0.5)

print("Over 25: ",c)
```

For Loop

- The **for** loop is another type of loop that sequences through a range of numbers or through the elements of a list.
- The first program specifies a range of values from 0 to 19. 20 is never reached. That is just the way **range()** works.
- Try it out in the shell. The variable **cnt** will start with zero and increment to 19 before the for loop exits.
- If you want **cnt** to increment by more than one, add one more value to **range()**. The second example shows the variable increments by 2.

```
for cnt in range(0,20):  
    print(cnt)  
print('end of loop')
```

For loop with cnt increment by 1

```
for cnt in range(0,20,2):  
    print(cnt)  
print('end of loop')
```

For loop with cnt increment by 2

For Loop

- The **for** loop can also sequence through a list type variable.
- Try this program to see how it works. Save it to the board as **test3.py**.
- The **for** loop will step through the list assign the next element from the list to the variable **b**.

```
a = [1,5,3,4,7,12,2,6,99]
for b in a:
    print(b)
print('end of list')
```

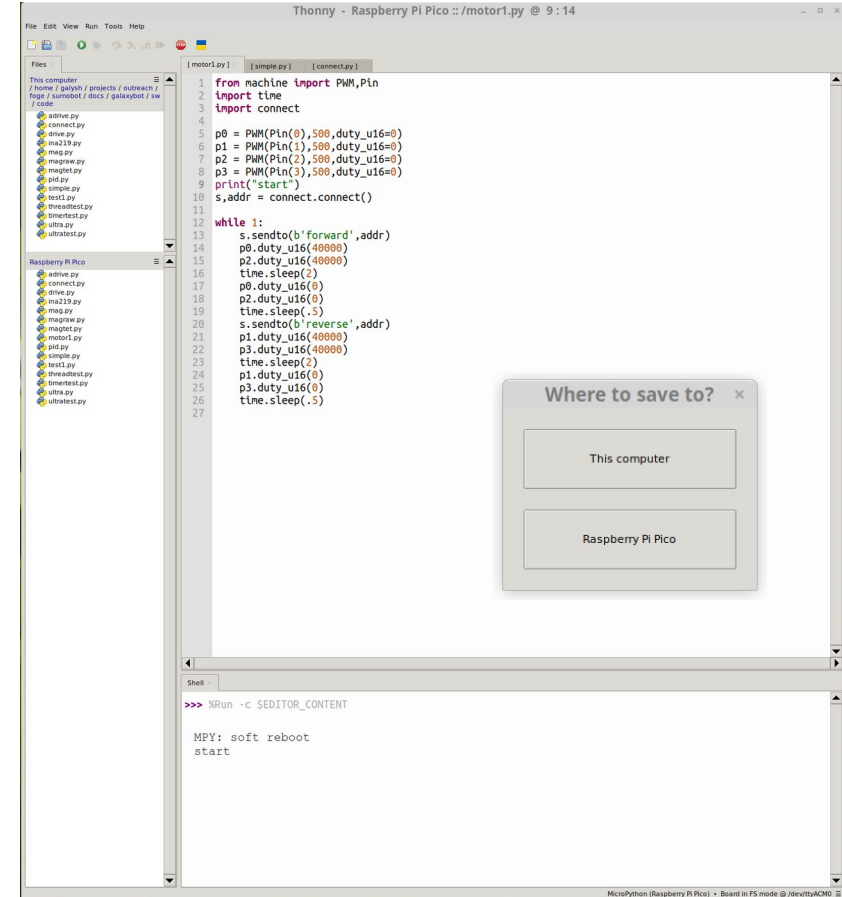
Functions

- Functions are program instructions grouped together that can be used repeatedly without having to write it over and over again.
- Functions help organize programs and simplify programming.
- Try the program to the right. Save it as **test4.py**.
- The function adds two numbers passed to it and then returns the result.
- In the **for** loop, the function is called and passes variable **d** and a constant value **5**. The returned value is stored in variable **e**.
- **d** will increment from 0 to 9. The function add 5 to the value of **d**.

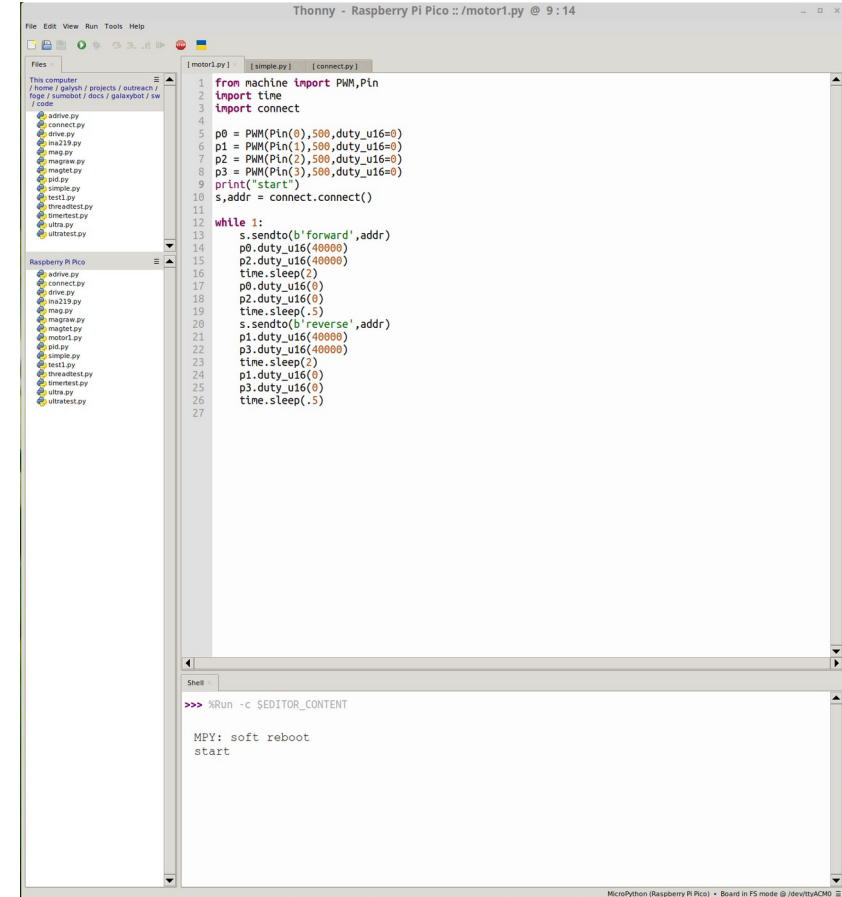
```
def add_two(a, b):  
    c = a + b  
    return c  
  
for d in range(0,10):  
    e = add_two(d,5)  
    print(e)
```

- This covers the basics of Python programming.
- You learned to import modules to add more functions to python
- You learned about variables and how to do math with variables.
- You learned about while loops and conditional statements.
- You learned how to make a function.
- One thing to remember is that to get lines of code to execute in the while loop, function or with the conditional statements is to **indent**. Python looks for indentation and it has to be consistent. You have to always use the same number of spaces for indentation.

- When writing programs for the board, always save to **Raspberry PI Pico**. The window asking where to save will appear. After the file has been saved, you can modify it and save it and it will automatically save to the board.
- You can open a file viewer by selecting the **View** menu and select **Files**. This opens up a list of files on the left. The top list is for the files on the laptop. The bottom list is all the files stored on the board. You can double click on a file to open it in the editor.



- To move program files from the laptop, you can navigate to the location of the file on the laptop and right click on the file. Select the **Upload** in the pop-up menu and the file will get copied to the board.
- You can also download from the board to the laptop by doing the same thing.



```
Thonny - Raspberry Pi Pico :: /motor1.py @ 9:14

File Edit View Run Tools Help

Files
This computer
/home / galaxy / projects / outreach /
fuge / sumobot / docs / galaxybot / sw
/ code
  ✓ adrive.py
  ✓ connect.py
  ✓ drive.py
  ✓ ma219.py
  ✓ mag.py
  ✓ magraw.py
  ✓ magnet.py
  ✓ pid.py
  ✓ simple.py
  ✓ test1.py
  ✓ threadtest.py
  ✓ timertest.py
  ✓ ultra.py
  ✓ ultratest.py
Raspberry Pi Pico
  ✓ adrive.py
  ✓ connect.py
  ✓ drive.py
  ✓ ma219.py
  ✓ mag.py
  ✓ magraw.py
  ✓ magnet.py
  ✓ motor1.py
  ✓ simple.py
  ✓ test1.py
  ✓ threadtest.py
  ✓ timertest.py
  ✓ ultra.py
  ✓ ultratest.py

[motor1.py] [simple.py] [connect.py]

1 from machine import PWM, Pin
2 import time
3 import connect
4
5 p0 = PWM(Pin(0), 500, duty_u16=0)
6 p1 = PWM(Pin(1), 500, duty_u16=0)
7 p2 = PWM(Pin(2), 500, duty_u16=0)
8 p3 = PWM(Pin(3), 500, duty_u16=0)
9 print("start")
10 s, addr = connect.connect()
11
12 while 1:
13     s.sendto(b'forward', addr)
14     p0.duty_u16(40000)
15     p2.duty_u16(40000)
16     time.sleep(2)
17     p0.duty_u16(0)
18     p2.duty_u16(0)
19     time.sleep(.5)
20     s.sendto(b'reverse', addr)
21     p1.duty_u16(40000)
22     p3.duty_u16(40000)
23     time.sleep(2)
24     p1.duty_u16(0)
25     p3.duty_u16(0)
26     time.sleep(.5)
27

Shell
>>> %Run -c $EDITOR_CONTENT

MPY: soft reboot
start

MicroPython (Raspberry Pi Pico) • Board in FS mode @ /dev/ttyACM0
```