

# Thonny Shell & Program Window

## Introduction:

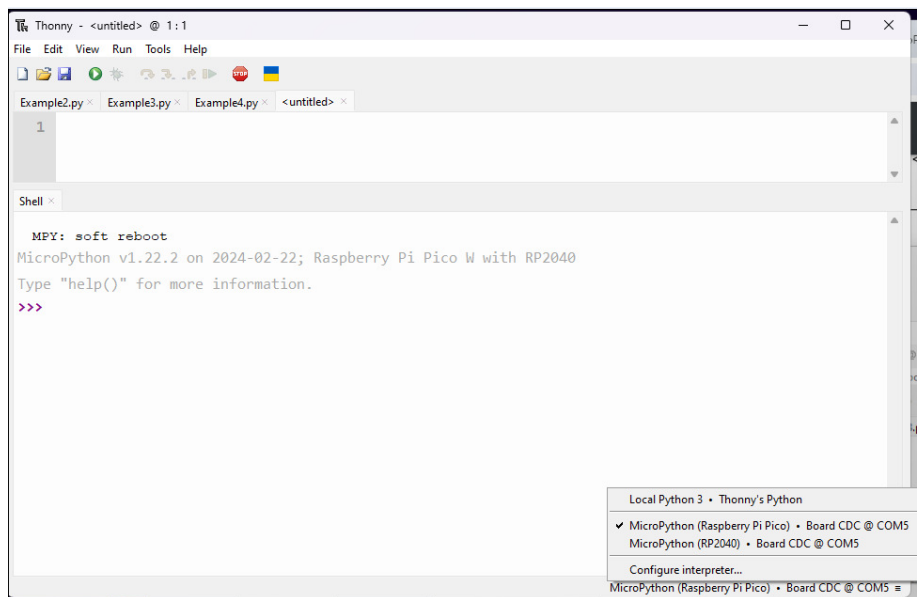
Thonny and MicroPython are very similar to Matlab

- Like Matlab, Thonny can be used like a calculator. The user can type in equations and have them evaluated as each instruction is typed.
- Like Matlab, MicroPython can work with complex numbers
- Like Matlab, Thonny allows you to create a file with multiple instructions (what Matlab calls a script). The operator can then execute those instructions as a program.

This lecture goes over using MicroPython in this fashion

## Installing Thonny

- Locate Thonny 4.1.4
- Download to PC
- Connect to Pico board
- Install Micropython



Once Thonny is installed, connect your Pico board to the computer then click on the lower-right corner to install MicroPython

## Thonny: Command Window

<https://www.fredscave.com/05-micropython-identifier-naming.html>

Once Thonny is installed and you're connected to your Pico chip, you're ready to start running Python code. Thonny looks very much like Matlab:

- There is command window (shell) where you can type in code directly and see the result
- There is a script window (top window) where you can write and run programs.

Like Matlab, Python is an interpretive language.

- Each line of code is executed one-by-one, allowing you to see the result of each instruction.
- This also means that Python is slower than C code.

As a general rule, each time you go up one level in programming, the code becomes 3-10 times larger and 3-10 times slower:

- C is typically 3-10 times larger and slower than assembler
- Python is typically 3-10 times larger and slower than C

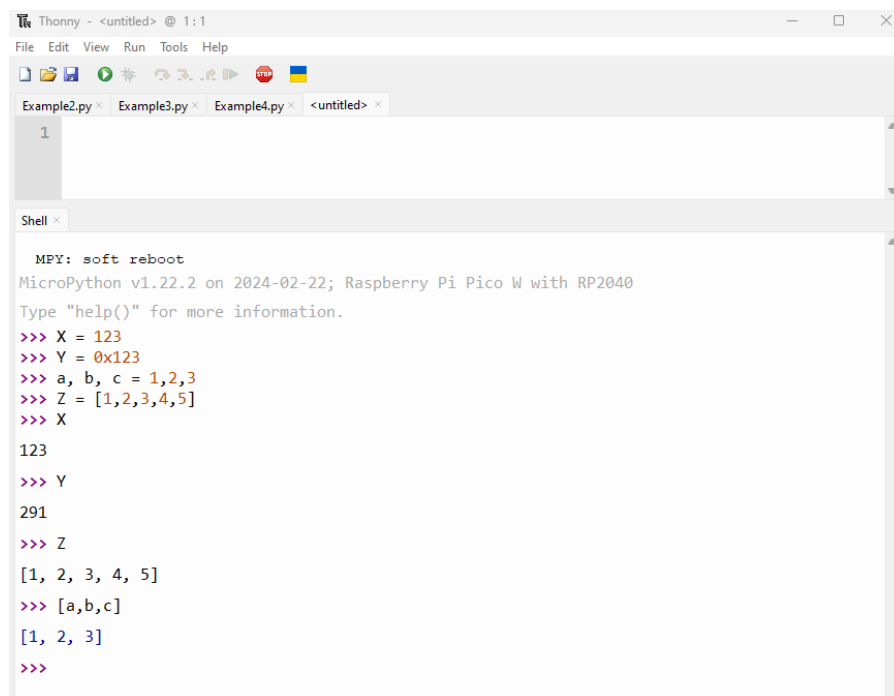
In return, you get a language which is easier to use, easier to modify, and easier to build upon.

As far as which language is best, it depends.

- If you really need speed and size, use assembler.
- If speed, number crunching, and reusable code is needed, use C
- If incorporating features such as graphics displays, touch-screens, web interfaces, etc. are needed, use Python.

In short, programming languages are tools. If the tool helps you do your job, use it. If the tool makes your life hard, don't use it.

Anyway, getting back to the Thonny command window (Matlab-speak) or shell window (Thonny-speak). Once Thonny is running, you can type commands in the shell.

The image shows a screenshot of the Thonny IDE. The top window is titled 'Thonny - <untitled> @ 1:1' and contains a code editor with a single line of code: '1'. Below the code editor is a 'Shell' window. The shell window displays the following text: 'MPY: soft reboot', 'MicroPython v1.22.2 on 2024-02-22; Raspberry Pi Pico W with RP2040', 'Type "help()" for more information.', and a series of Python commands and their outputs: '>>> X = 123', '>>> Y = 0x123', '>>> a, b, c = 1,2,3', '>>> Z = [1,2,3,4,5]', '>>> X', '123', '>>> Y', '291', '>>> Z', '[1, 2, 3, 4, 5]', '>>> [a,b,c]', '[1, 2, 3]', and '>>>'.

In the shell window, you can use Thonny as a calculator using MicroPython syntax

Here, Python act like a calculator, very much like Matlab. At the command line you can do calculations such as

```
>>> 3 + 5
8
```

You can create variables with names - as long as you use a valid variable name. To be valid

- The first character must be a letter or an underscore (`_`)
- The following characters can *only* be letters, underscores, or digits

```
>>> Pi.Value = 3.14159      invalid    decimal points are not allowed
>>> Pi Value = 3.14159    invalid    spaces are not allowed
>>> Pi_Value = 3.14159   valid variable name
```

MicroPython is case sensitive. The following code creates two different variables

```
>>> Name = 'Jake'
>>> name = 'Bill'
```

- You *cannot* use the following words as variable names  
and, as, assert, break, class, continue, def  
del, elif, else, except, finally, for, from, global  
if, import, in, is, lambda, nonlocal, not, or, pass,  
raise, return, try, while, with, yield, False, None, True

## MicroPython Syntax

Assigning values to variables:

```
X = 123          decimal 123
X = 0x123       hex 123
x, y, z = 1, 2, 3
X = [1,2,3,4,5] matrix or array
X = range(1,6)  same matrix
X = [[1,2],[3,4]] 2x2 matrix
```

### Operations

```
+      add
-      subtract
*      multiply
/      divide (result is usually a float)
//     divide and round down (result is integer)
%      modulus (remainder)
**     raise to the power
```

```
X.append(6)    append 6 to the end of array X
```

### Logic Operations

```
&      logical AND (bitwise)
|      logical OR (bitwise)
^      logical XOR (bitwise)
>>    shift right
<<    shift left
```

**# comment statement**

```
# this is a comment statement
```

**Conditionals:**

```
X > Y
X < Y
X >= Y
X == Y
X != Y
```

**Converting variable types:**

```
int(X)    convert to an integer, round down
round(X)  round to nearest integer
float(X)  convert to a floating point number
```

note: Python automatically adjusts variable types - you don't need to declare them like you do in C. For example:

```
>>> X = 3          X is automatically treated like an integer
>>> Y = 4          Y is automatically treated like an integer
>>> Z = X/Y        Z becomes a float (0.75)
>>> Z = X//Y       Z is an integer (0)
```

**print()** Information can be sent to the shell window using a *print()* statement

```
>>> print('Hello World')
Hello World

>>> X = 2**0.5
>>> print('X = ',X)
X = 1.414214
```

**X = input()** Information can be passed to your program using the *input()* statement. For example, prompt the user to input a number for X:

```
>>> X = input('Type in a number')
```

This will result in X being a string (typing in *Hello World* is valid). If you want to receive the input as a number, convert the result as:

```
>>> X = int( input('Type in a number') )
>>> X = float( input('Type in a number') )
```

When writing to the shell, numbers can be formatted if desired. Examples follow:

```
>>> msg = '27 in binary = {:b}'.format(27)
>>> msg
'27 in binary = 11011'

>>> msg = '27 in binary = {:b}'.format(27)
>>> msg
'27 in binary = 11011'
```

```

>>> msg = '27 in hex = {:X}'.format(27)
>>> msg
'27 in hex = 1B'

>>> msg = '0x2134 in decimal = {:d}'.format(0x1234)
>>> msg
'0x2134 in decimal = 4660'

>>> msg = '123.4567 rounded to 2 decimal = {:.2f}'.format(123.4567)
>>> msg
'123.4567 rounded to 2 decimal = 123.46'

>>> msg = '123.4567 rounded to 2 decimal = {:.2e}'.format(123.4567)
>>> msg
'123.4567 rounded to 2 decimal = 1.23e+02'

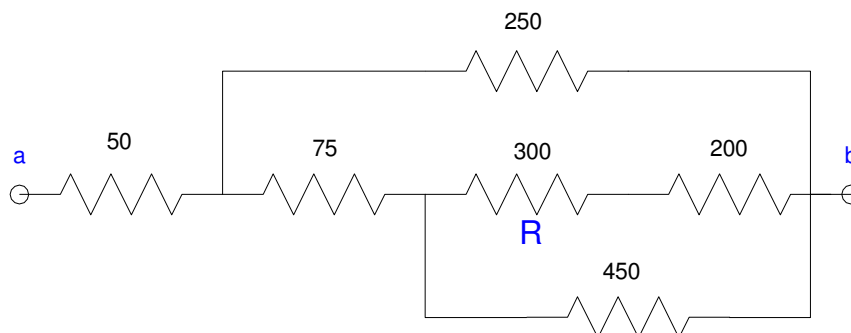
>>> msg = '79/255 = {:.2%}'.format(79/255)
>>> msg
'79/255 = 30.98%'

```

## Command Window & Scripts Examples

Like Matlab, MicoPython can be used like a calculator.

Example 1: Find the resistance Rab:



To simplify, recall that resistors add in series and parallel as:

$$R_s = R_1 + R_2$$

$$R_p = \left( \frac{1}{R_1} + \frac{1}{R_2} \right)^{-1}$$

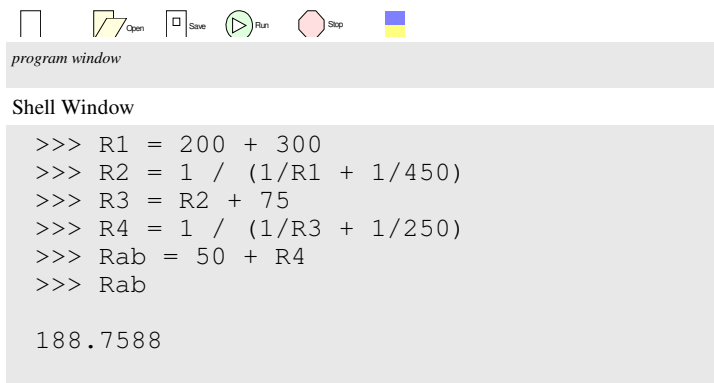
For example, the 200 Ohm and 300 Ohm are in series

$$R_{net} = 200 + 300 = 500\Omega$$

This is in parallel with 450 Ohms

$$450 || 500 = \left( \frac{1}{450} + \frac{1}{500} \right)^{-1} = 236.84\Omega$$

In the shell window, you can solve for Rab:



```

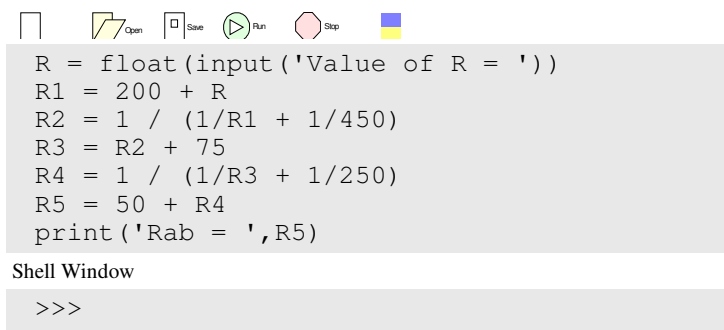
program window
Shell Window
>>> R1 = 200 + 300
>>> R2 = 1 / (1/R1 + 1/450)
>>> R3 = R2 + 75
>>> R4 = 1 / (1/R3 + 1/250)
>>> Rab = 50 + R4
>>> Rab

188.7588

```

Like Matlab, you can automate this. Suppose for example you want to find Rab if the 300 Ohm resistor varies. A program which does this is as follows: (placed in the top screen - what Matlab calls the script window)

You can also run this as a program



```

program window
Shell Window
>>>

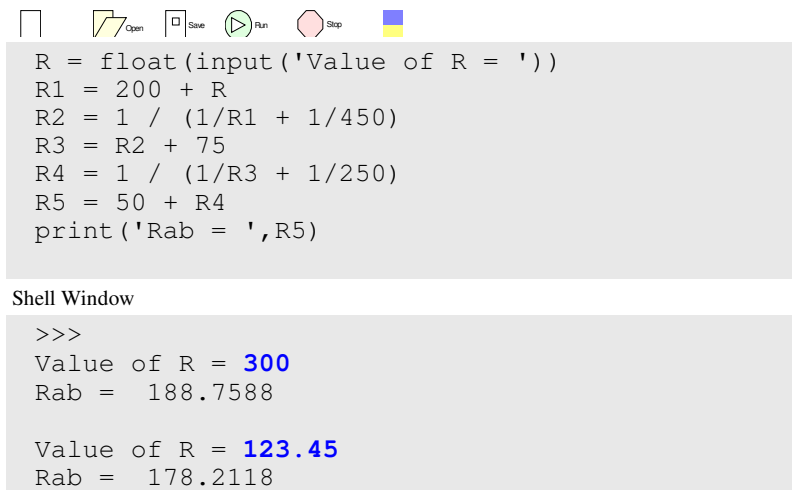
```

```

R = float(input('Value of R = '))
R1 = 200 + R
R2 = 1 / (1/R1 + 1/450)
R3 = R2 + 75
R4 = 1 / (1/R3 + 1/250)
R5 = 50 + R4
print('Rab = ',R5)

```

When you hit the *run* icon, the program starts to execute line by line. In the shell window, it will prompt you for the value of R and then it computes and displays Rab. When you type in *300*, the program computes Rab as 188.7588 ohms. When you type in *123.45*, the program computes Rab as 178.2188 ohms



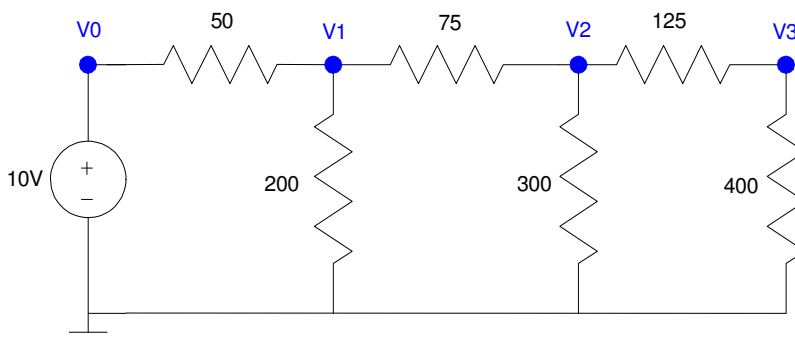
```

program window
Shell Window
>>>
Value of R = 300
Rab = 188.7588

Value of R = 123.45
Rab = 178.2118

```

Example 2: As a second example, compute the voltages  $\{V_1, V_2, V_3\}$  using voltage division:



Recall from Circuit I and voltage division, the voltages will be:

$$V_3 = \left( \frac{400}{400+125} \right) V_2$$

$$V_2 = \left( \frac{R_{20}}{R_{20}+75} \right) V_1$$

$$V_1 = \left( \frac{R_{10}}{R_{10}+50} \right) V_0$$

where  $R_{20}$  is the resistance at node 2 to ground looking right

$$R_{20} = 300 \parallel (400 + 125)$$

$R_{10}$  is the resistance at node 1 to ground looking right

$$R_{10} = 200 \parallel (75 + R_{20})$$

From the Shell (lower window):



Shell Window

```
>>> # Finding voltages using voltage division
>>> R20 = 1 / (1/300 + 1/125)
>>> R10 = 1 / (1/200 + 1/(75+R20))
>>> V1 = R10 / (R10 + 50) * 10
>>> V2 = R20 / (R20 + 75) * V1
>>> V3 = 400 / (400 + 125) * V2
>>> print(V1, V2, V3)

6.953938 4.992571 3.803864
```

If the 400 Ohm resistor varies, you could write a similar program to find these voltages by writing a program (placing the code in the program window) then hit *run*



```
R30 = int(input('Value of R30 = '))
R20 = 1 / (1/300 + 1/(125 + R30))
R10 = 1 / (1/200 + 1/(75+R20))
V1 = R10 / (R10 + 50)*10
V2 = R20 / (R20 + 75) * V1
V3 = 400 / (400 + 125) * V2
print('V1 = ', V1)
print(' V2 = ', V2)
print(' V3 = ', V3)
```

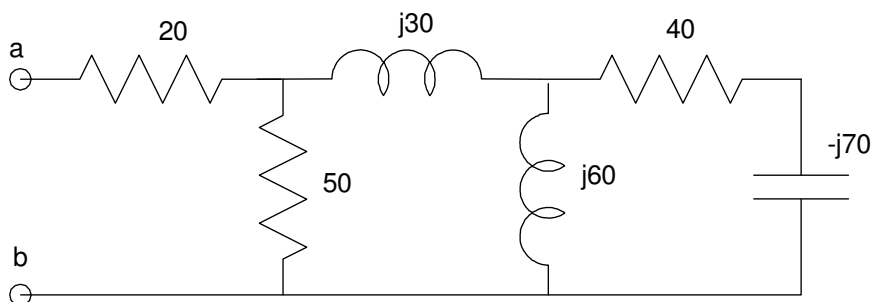
Shell Window

```
>>>
Value of R30 = 400
V1 = 6.9539
V2 = 4.9926
V3 = 3.8039

Value of R30 = 123.45
V1 = 6.7246
V2 = 4.3332
V3 = 3.3015
```



**Example 3: Complex Numbers Impedance.** Thonny can also handle complex numbers similar to Matlab. For example, find the impedance  $Z_{ab}$ :



In the Shell window:



Shell Window

```
>>> j = (-1) ** 0.5
>>> Z3 = - j*70
>>> Z2 = 1 / ( 1/(j*60) + 1/(40 + Z3))
>>> Z1 = 1 / ( 1/50 + 1 / (j*30 + Z2))
>>> Z0 = 20 + Z1
>>> print('Zab = ',Z0)

Zab = (58.96067+9.111071j)
```

If instead, you place this in the script window, when you run the program, it prompts you for the impedance of C:



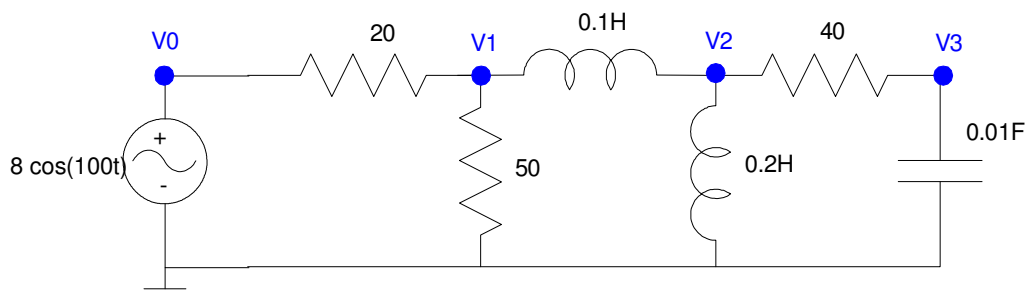
```
j = (-1) ** 0.5
X = float(input('Impedance of C3: -j'))
Z3 = -j*X
Z2 = 1 / ( 1/(j*60) + 1/(40 + Z3))
Z1 = 1 / ( 1/50 + 1 / (j*30 + Z2))
Z0 = 20 + Z1
print('Zab = ',Z0)
```

Shell Window

```
>>>
Impedance of C3: -j70
Zab = (58.96067+9.111071j)

Impedance of C3: -j45.678
Zab = (54.26275+7.450337j)
```

**Example 4:** Finally, this also works with voltage division. Find {V1, V2, and V3} using MicroPython:



Recall from Circuits I, the AC impedance of inductors, resistors, and capacitors are:

$$R \rightarrow R$$

$$L \rightarrow j\omega L$$

$$C \rightarrow \frac{1}{j\omega C}$$

and voltages convert as

$$V = a \cos(\omega t) + b \sin(\omega t) \rightarrow a - jb$$

In the Shell window:



Shell Window

```
>>> w = 100
>>> Z30 = 1 / (j*w*0.01)
>>> Z20 = j*w*0.2
>>> Z12 = j*w*0.1
>>> Z3 = Z30
>>> Z2 = 1 / (1/Z20 + 1/(40 + Z30))
>>> Z1 = 1 / (1/50 + 1/(Z12 + Z20))
>>> V0 = 8 + j*0
>>> V1 = Z1 / (20 + Z1)*V0
>>> V2 = Z2 / (Z12 + Z2)*V1
>>> V3 = Z3 / (40 + Z3)*V2

>>> print('V0 = ',V0)
V0 = (8+0j)

>>> print('V1 = ',V1)
V1 = (4.658041+2.218115j)

>>> print('V2 = ',V2)
V2 = (3.27509+0.937138j)

>>> print('V3 = ',V3)
V3 = (0.02545947-0.08124077j)
```

You can also place this in the program window and run it as a program:



```
C3 = float(input('Value of C3 (F) = '))
w = 100
Z30 = 1 / (j*w*C3)
Z20 = j*w*0.2
Z12 = j*w*0.1
Z3 = Z30
Z2 = 1 / (1/Z20 + 1/(40 + Z30))
Z1 = 1 / (1/50 + 1/(Z12 + Z20))
V0 = 8 + j*0
V1 = Z1 / (20 + Z1)*V0
V2 = Z2 / (Z12 + Z2)*V1
V3 = Z3 / (40 + Z3)*V2
print('V0 = ',V0)
print('V1 = ',V1)
print('V2 = ',V2)
print('V3 = ',V3)
```

Shell Window

```
>>>
Value of C3 (F) = 0.01
V0 = (8+0j)
V1 = (4.658041+2.218115j)
V2 = (3.27509+0.937138j)
V3 = (0.02545947-0.08124077j)

Value of C3 (F) = 0.001234
V0 = (8+0j)
V1 = (4.6580 + 2.2181j)
V2 = (3.3701 + 0.9707j)
V3 = (0.3218 - 0.6176j)
```

## Differences Between Matlab and MicroPython

While Matlab and MicroPython are similar, there are differences. A *big* difference is

- Matlab was written for engineers and scientists
- MicroPython was written for the general public

More specifically:

- Matlab is a matrix language
- MicroPython is not

For example, in Matlab, variables are treated as matrices. In Matlab, when you multiply variables, it does it assuming they are matrices

Matlab Command Window

```
>> A = [1, 2; 3, 4]
A =
     1     2
     3     4

>> B = 2 * A
B =
     2     4
     6     8

>> C = A*A
C =
     7    10
    15    22
```

In contrast, Thonny treats variables as strings

Thonny Shell (Python)

```
>>> A = 'Hello'
>>> A
'Hello'

>>> B = 2*A
'HelloHello'

>>> A = [1, 2]
>>> A
[1, 2]

>>> B = 2*A
>>> B
[1, 2, 1, 2]

>>> C = A*A
Error - unsupported file type
```

This is a major shortcoming of Python - which will probably be fixed in future upgrades. As for now, however, Python does not work well with matrices.

## Summary:

Thonny is very similar to Matlab

- You can use the Shell window like the command window in Matlab. It behaves like a calculator, typing in code and seeing the result after each instruction.
- You can use the program window like the script window in Matlab. Once code is written, you can execute the program

