

---

# **Thonny Shell & Program Window**

**ECE 476 Advanced Embedded Systems**

**Jake Glower - Lecture #2**

Please visit [Bison Academy](#) for corresponding lecture notes, homework sets, and solutions



---

## Introduction:

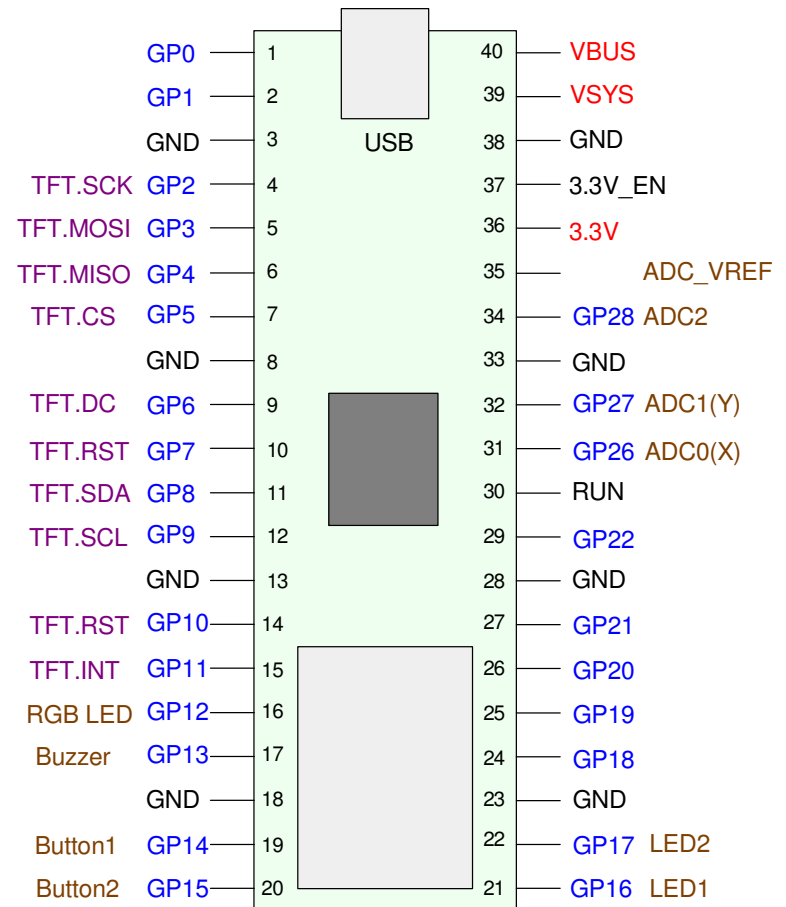
The Raspberry Pi-Pico is a microcontroller version of the Raspberry Pi

It can be programmed in

- Assembler
- C
- Python
- Other...

In ECE 476, we'll be using  
MicroPython

- A subset of Python
- Designed for microcontrollers



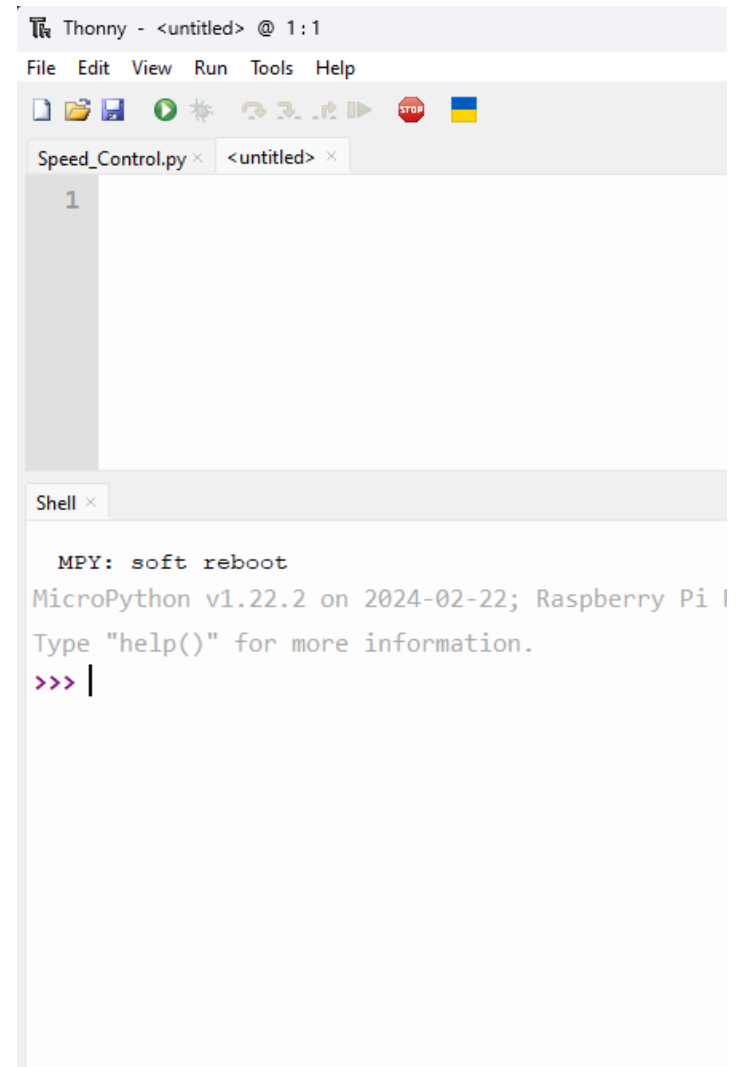
---

# Thonny, MicroPython, & Matlab

MicroPython is similar to Matlab

- Can be used as a calculator
- Works with complex numbers
- Has similar syntax
- Has windows (program, command)
- Is a programming language

This lecture goes over using MicroPython in this fashion



The screenshot shows the Thonny IDE interface. The top window is titled "Thonny - <untitled> @ 1:1" and contains a menu bar (File, Edit, View, Run, Tools, Help) and a toolbar with icons for file operations, execution, and a stop button. Below the toolbar, there are two tabs: "Speed\_Control.py" and "<untitled>". The main editor area shows a single line of code: "1". Below the editor is a "Shell" window. The shell output reads: "MPY: soft reboot", "MicroPython v1.22.2 on 2024-02-22; Raspberry Pi 1", "Type 'help()' for more information.", and a prompt ">>> |".

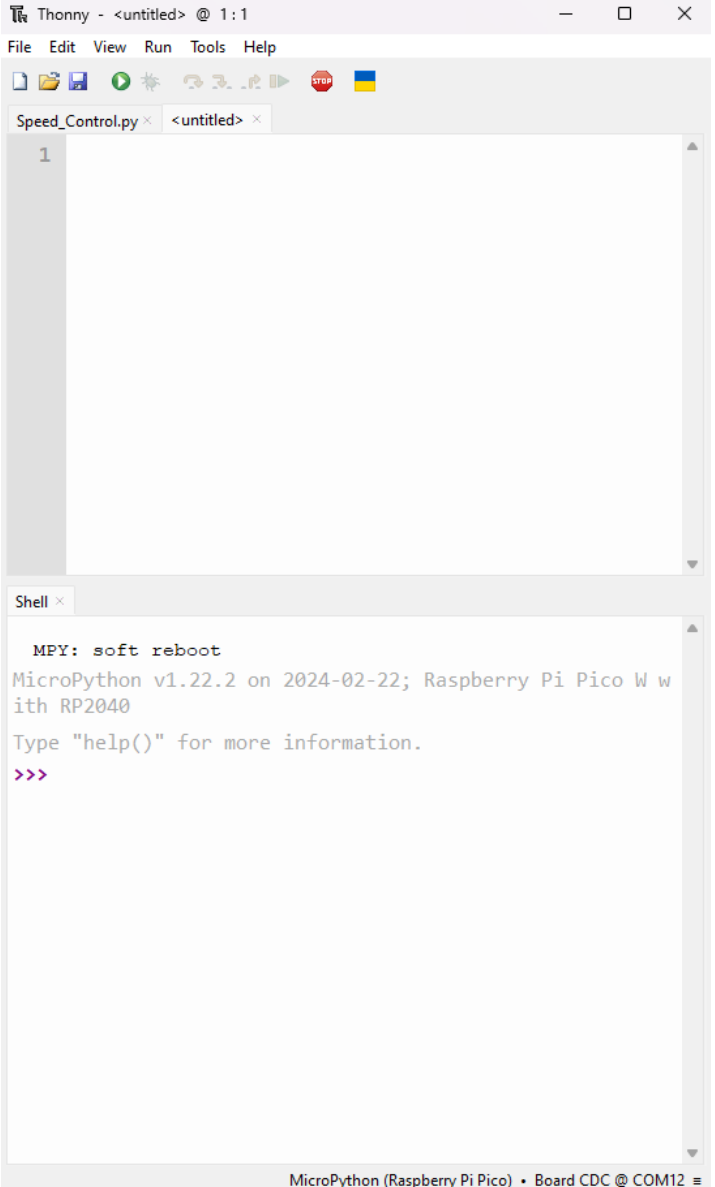
---

# Installing Thonny

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

Click on the lower-right corner

- Select your Pi-Pico chip
- It will prompt you to install MicroPython if this is the first time using your chip



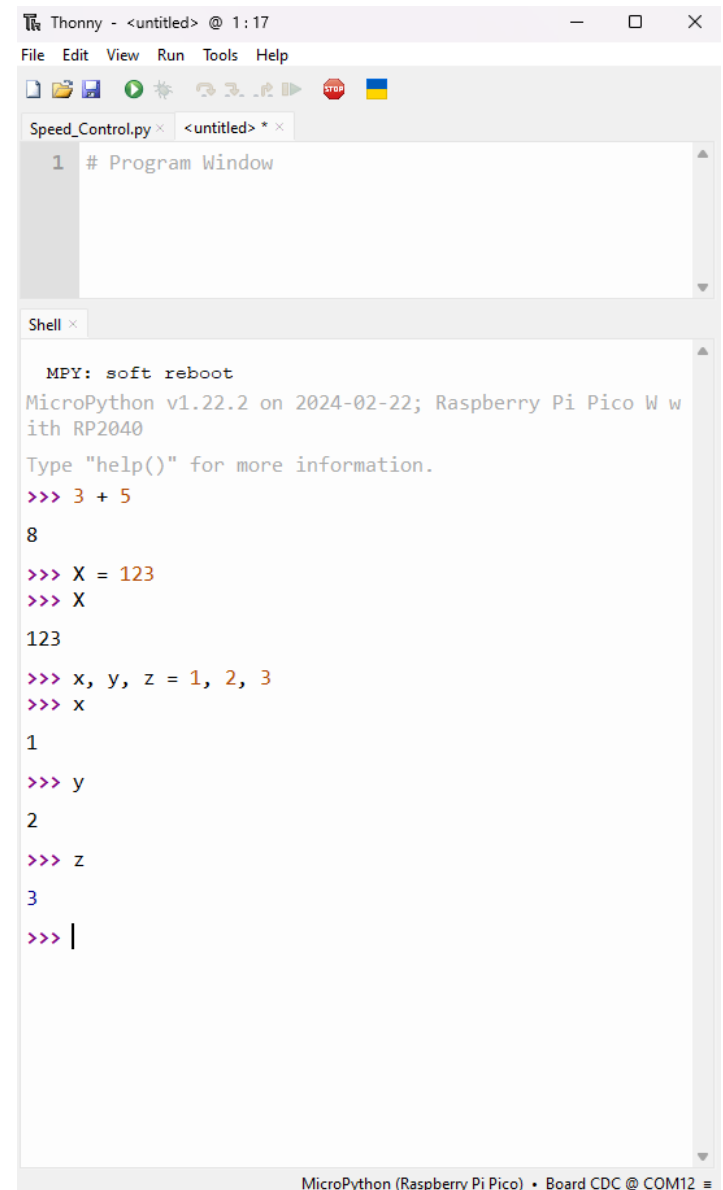
The screenshot shows the Thonny IDE window. The title bar reads "Thonny - <untitled> @ 1:1". The menu bar includes "File", "Edit", "View", "Run", "Tools", and "Help". The toolbar contains icons for file operations and execution. The editor area shows a file named "Speed\_Control.py" with a single line of code: "1". Below the editor is a "Shell" window displaying 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 prompt ">>>". The status bar at the bottom indicates "MicroPython (Raspberry Pi Pico) • Board CDC @ COM12".

---

# Thonny: Command Window

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



The screenshot shows the Thonny IDE interface. The top window is a script editor titled 'Speed\_Control.py' containing a single line of code: `1 # Program Window`. Below it is a shell window titled 'Shell' which displays the output of a MicroPython session. The output starts with 'MPY: soft reboot' and 'MicroPython v1.22.2 on 2024-02-22; Raspberry Pi Pico W with RP2040'. It then shows several interactive commands and their results: `>>> 3 + 5` returns `8`; `>>> X = 123` and `>>> X` returns `123`; `>>> x, y, z = 1, 2, 3` and `>>> x` returns `1`; `>>> y` returns `2`; `>>> z` returns `3`. The shell prompt is currently `>>> |`. The status bar at the bottom indicates 'MicroPython (Raspberry Pi Pico) • Board CDC @ COM12'.

---

Python is an interpretive language.

- Similar to Matlab
- Each line of code is executed as you type
- Allows you to see each result

Python is slower than C code.

- C is 3-10x slower than assembler
- Python is 3-10x slower than C

In return, you get a language which is

- Easier to use,
- Easier to modify, and
- Easier to build upon.

```
Thonny - <untitled> @ 1:17
File Edit View Run Tools Help
Speed_Control.py x <untitled> * x
1 # Program Window

Shell x
MPY: soft reboot
MicroPython v1.22.2 on 2024-02-22; Raspberry Pi Pico W with RP2040
Type "help()" for more information.
>>> 3 + 5
8
>>> X = 123
>>> X
123
>>> x, y, z = 1, 2, 3
>>> x
1
>>> y
2
>>> z
3
>>> |

MicroPython (Raspberry Pi Pico) • Board CDC @ COM12
```

---

## What 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.
-

---

## Thonny & Shell Window

- Command Window in Matlab-Speak

You can type commands in the shell.

Python act like a calculator, very much like Matlab

At the command line you can do calculations such as

```
>>> 3 + 5
8

>>> (2 + 3) * (7 + 8)
75

>>> 1 / (1/50 + 1/60 + 1/70)
19.6262
```



---

# Variable Names in Python

Variables are valid in Python

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
```

---

## Case Sensitivity

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

---

---

## Declaring Variables:

Python allows you to create new variables on the fly

- You don't have to declare all of your variables at the start of a program

Python automatically adjusts variable types

```
>>> 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
```



The screenshot shows the Thonny IDE interface. The top window is titled "Speed\_Control.py" and contains the following code:

```
1 # Program Window
```

The bottom window is titled "Shell" and displays the output of the code execution:

```
MPY: soft reboot  
MicroPython v1.22.2 on 2024-02-22; Raspberry  
with RP2040  
Type "help()" for more information.  
>>> print('Hello World')  
Hello World  
>>> X = 2**0.5  
>>> print('X = ',X)  
X = 1.414214  
>>>
```

---

## X = input()

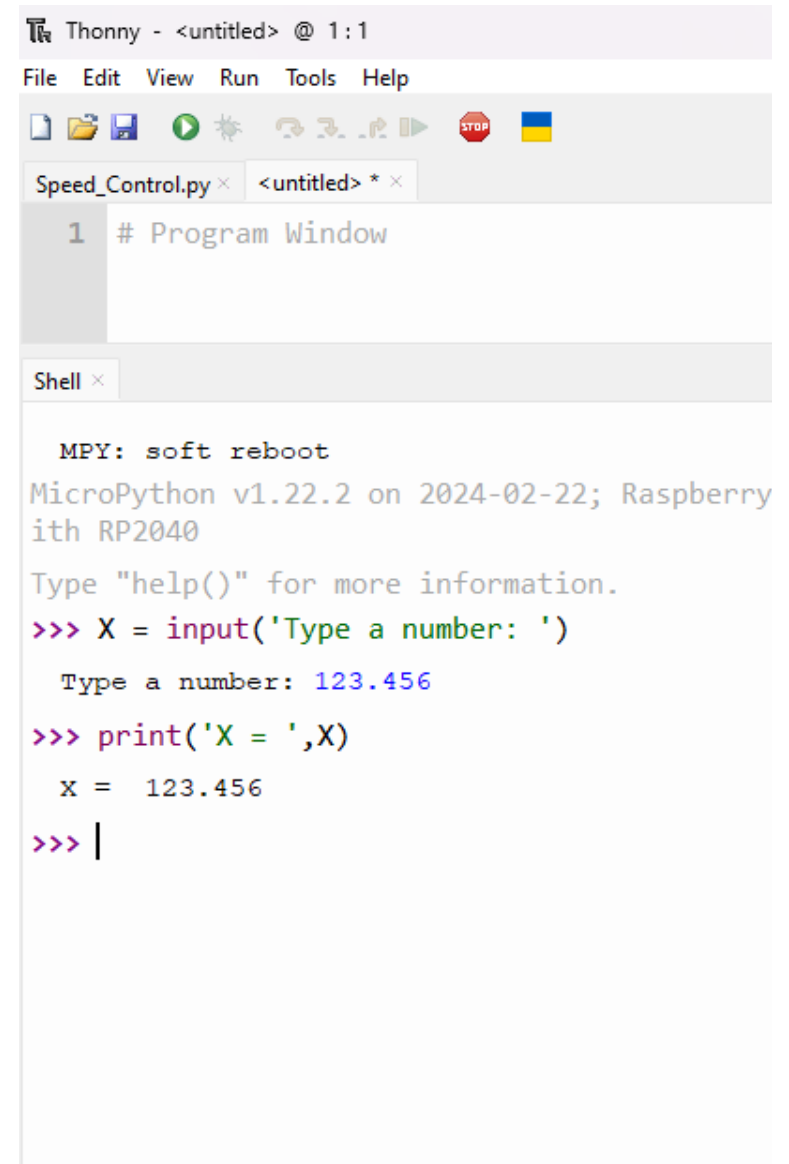
- Prompts user for an input

Python interprets this as a string

- Convert to an integer or float

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

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



The screenshot shows the Thonny Python IDE interface. The top window, titled "Speed\_Control.py", contains a single line of code: `1 # Program Window`. Below it is the "Shell" window, which displays the following output:

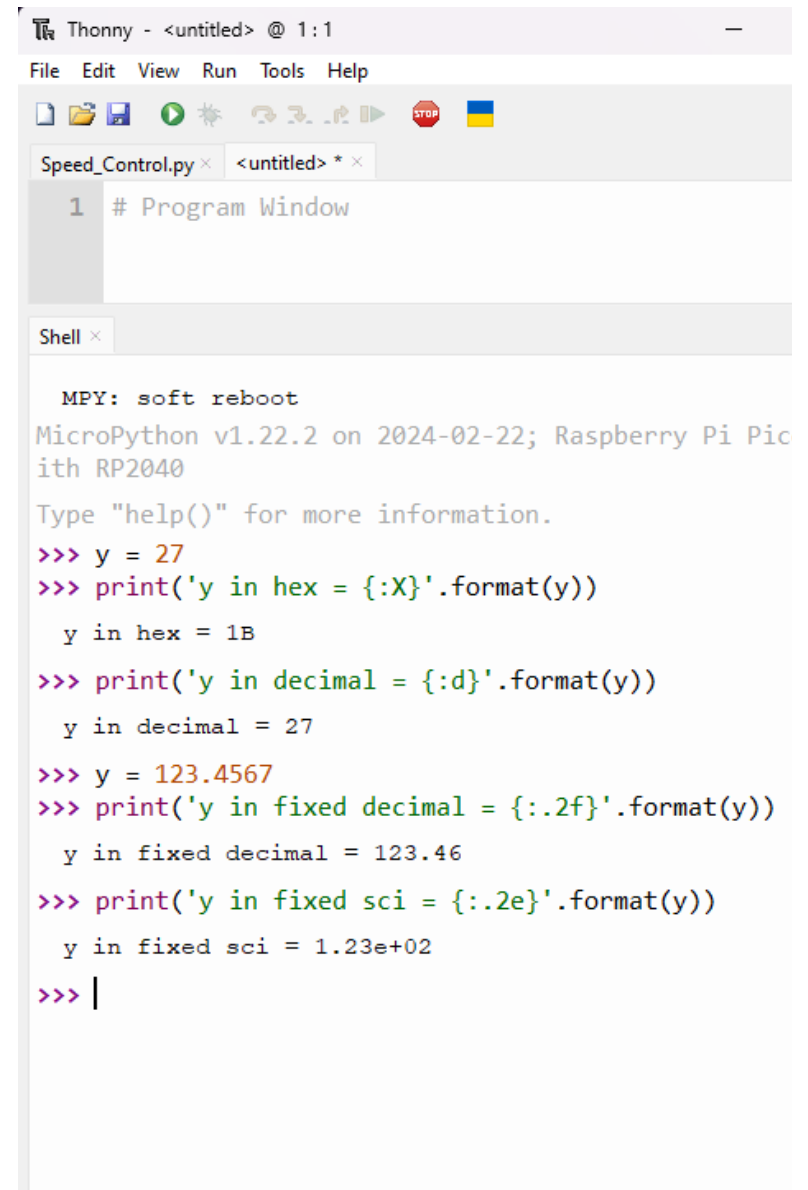
```
MPY: soft reboot
MicroPython v1.22.2 on 2024-02-22; Raspberry
ith RP2040
Type "help()" for more information.
>>> X = input('Type a number: ')
Type a number: 123.456
>>> print('X = ',X)
X = 123.456
>>> |
```

---

# Formatting Output

Makes output prettier

- X hexadecimal
- d decimal
- 2f two fixed decimal places
- 2e two decimal in scientific notation



```
Thonny - <untitled> @ 1:1
File Edit View Run Tools Help
Speed_Control.py x <untitled> * x
1 # Program Window

Shell x
MPY: soft reboot
MicroPython v1.22.2 on 2024-02-22; Raspberry Pi Pic
ith RP2040
Type "help()" for more information.
>>> y = 27
>>> print('y in hex = {:X}'.format(y))
y in hex = 1B
>>> print('y in decimal = {:d}'.format(y))
y in decimal = 27
>>> y = 123.4567
>>> print('y in fixed decimal = {:.2f}'.format(y))
y in fixed decimal = 123.46
>>> print('y in fixed sci = {:.2e}'.format(y))
y in fixed sci = 1.23e+02
>>> |
```



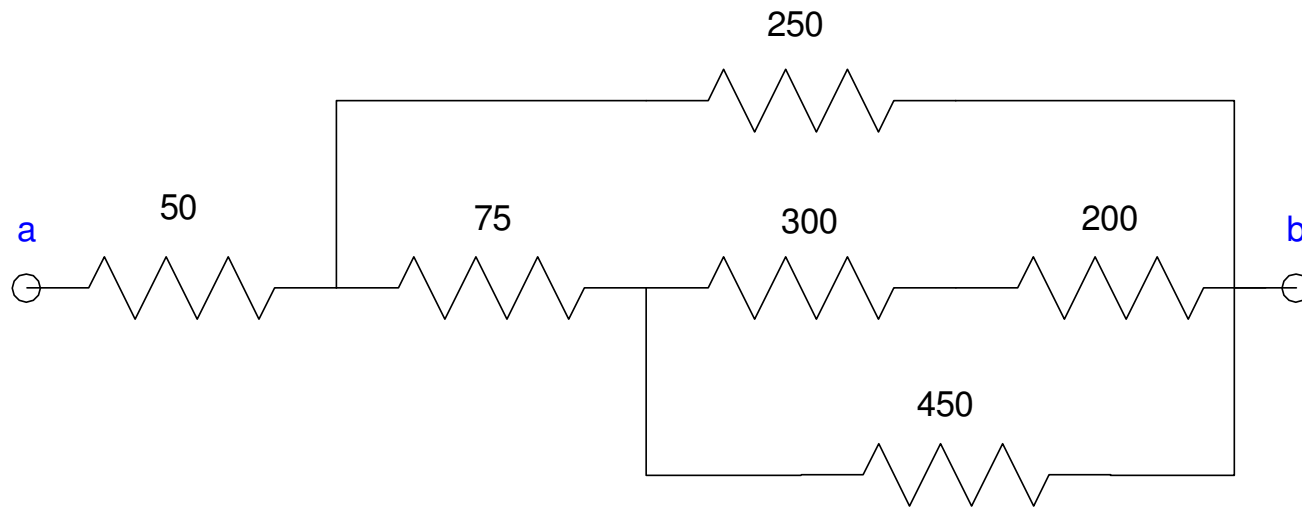
---

## Command Window & Scripts Examples

Like Matlab, MicoPython can be used like a calculator.

Example 1: Find the resistance  $R_{ab}$ :

- Resistors add in series as  $R_s = R_1 + R_2$
- Resistors in parallel as  $R_p = \left(\frac{1}{R_1} + \frac{1}{R_2}\right)^{-1}$



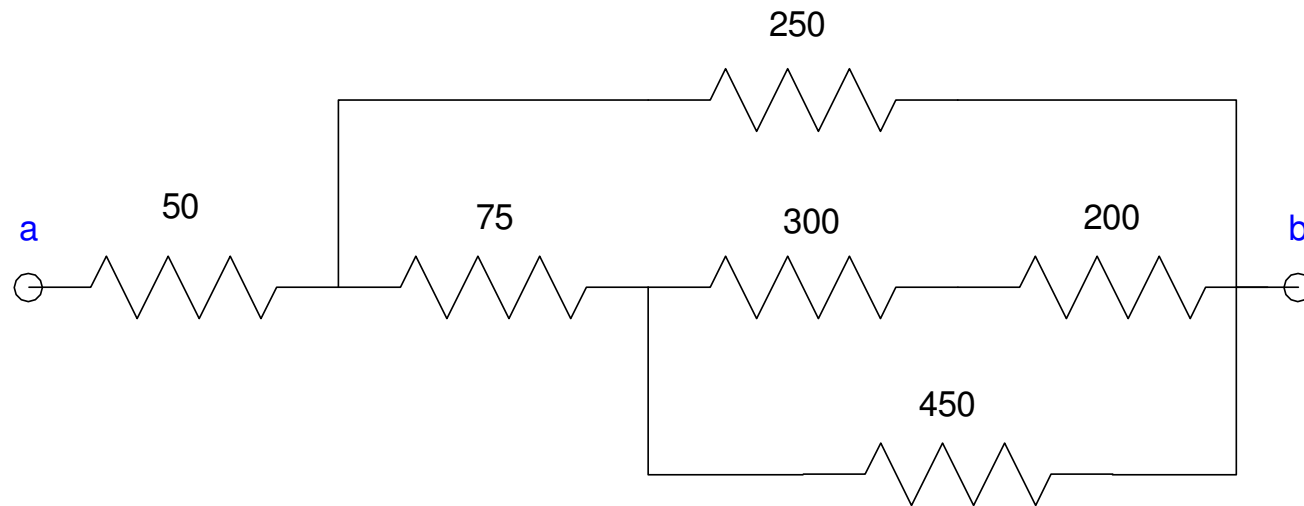
---

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:

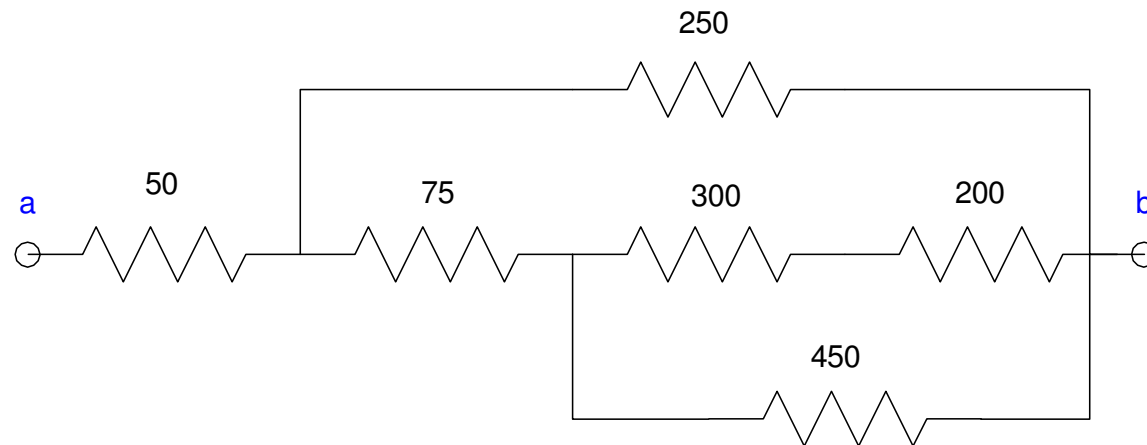


```
# program window
```

### 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
```



---

## Python Programs:

- Place the code in the program window (top)
- Allows you to run the program over and over



```
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)
```

Shell Window

```
>>>
```

---

---

## Hit the *run* icon to execute the program

- Prompts you for an input in the shell window
- The program computes and displays Rab
- Pressing *run* repeats the process



```
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)
```

### Shell Window

```
>>>
Value of R = 300
Rab = 188.7588

Value of R = 123.45
Rab = 178.2118
```

---

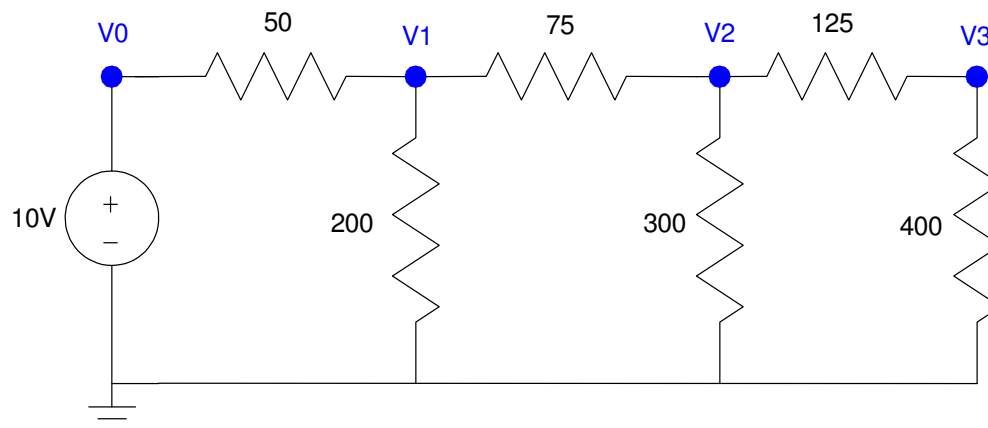
Example 2: As a second example, compute the voltages  $\{V_1, V_2, V_3\}$  using voltage division:

From Circuits I

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

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

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



$R_{20}$  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

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

---

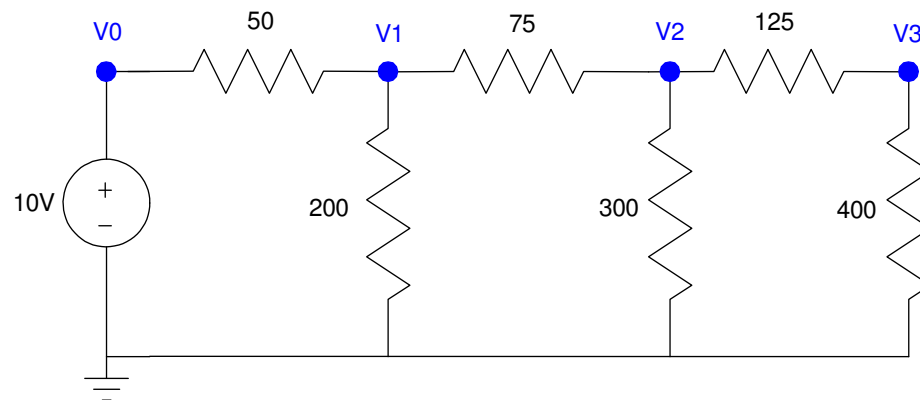
## From the Shell (lower window):



### Shell Window

```
>>> # Finding voltages using voltage division
>>> R20 = 1 / (1/300 + 1/525)
>>> R10 = 1 / (1/200 + 1/(75+R23))
>>> 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
```



---

## Place the instructions in the program window

- Lets you run over and over again



```
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
```

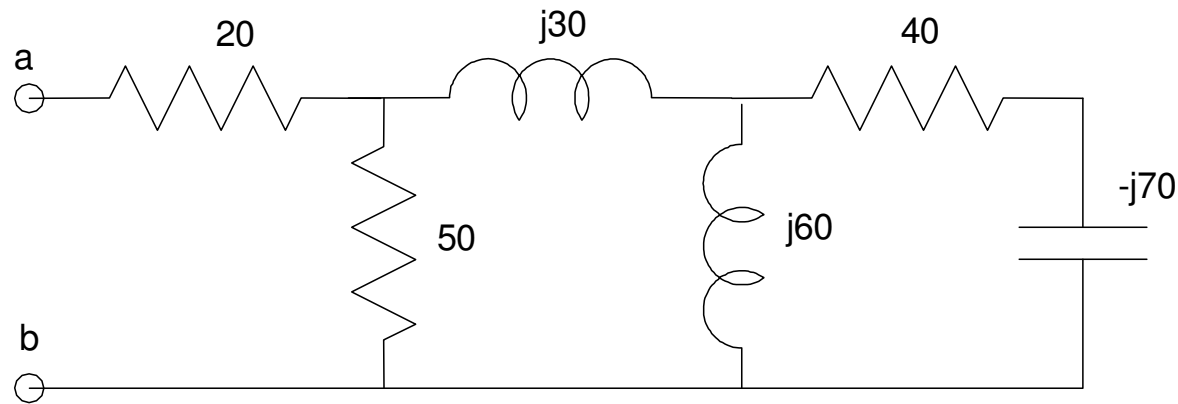


---

### Example 3: Complex Numbers Impedance.

Thonny can also handle complex numbers similar to Matlab.

For example, find the impedance  $Z_{ab}$ :



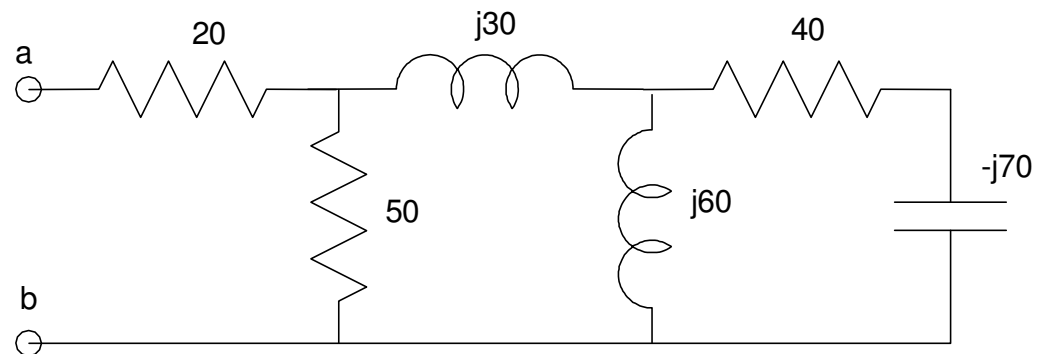
## In the Shell window:



### 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)
```



---

## In the Program Window:



```
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

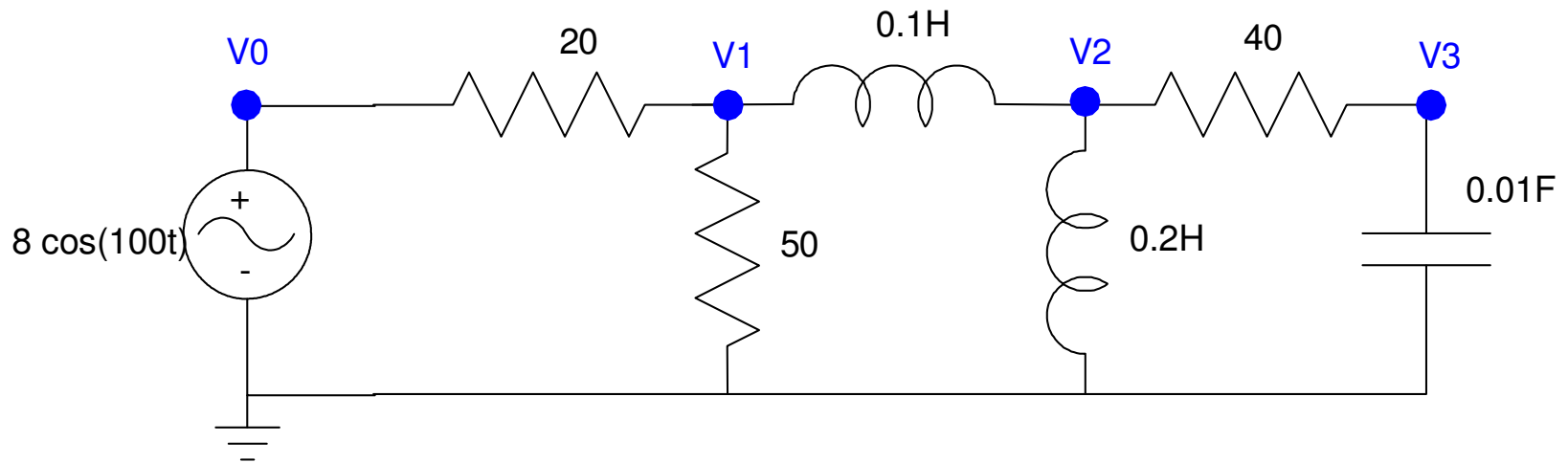
```
>>>
Impedance of C3: -j70
Zab = (58.96067+9.111071j)

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:

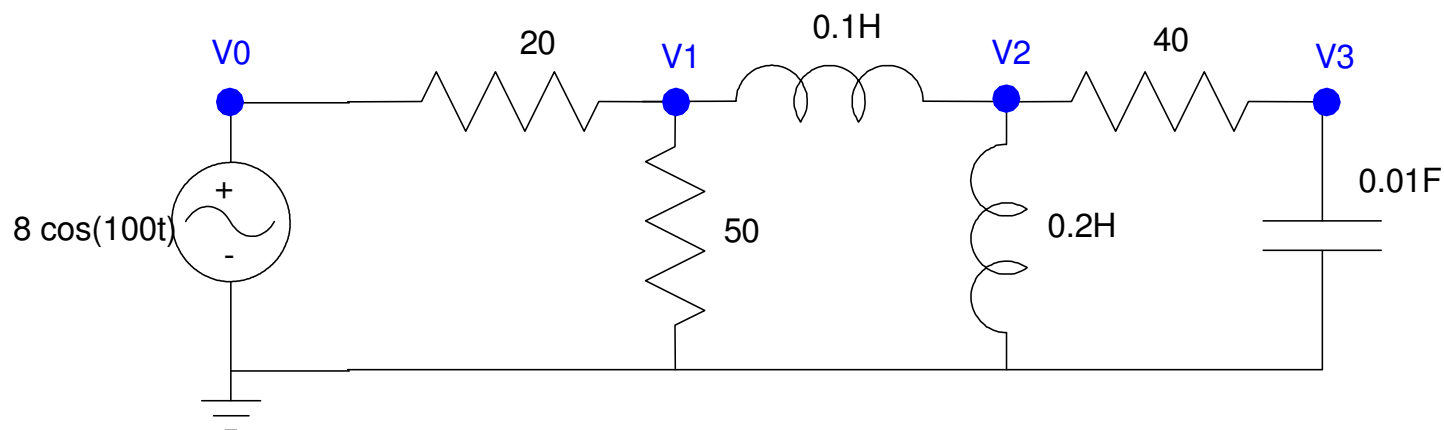
$$R \rightarrow R$$

$$L \rightarrow j\omega L$$

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

and voltages convert as

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



## In the Shell window:

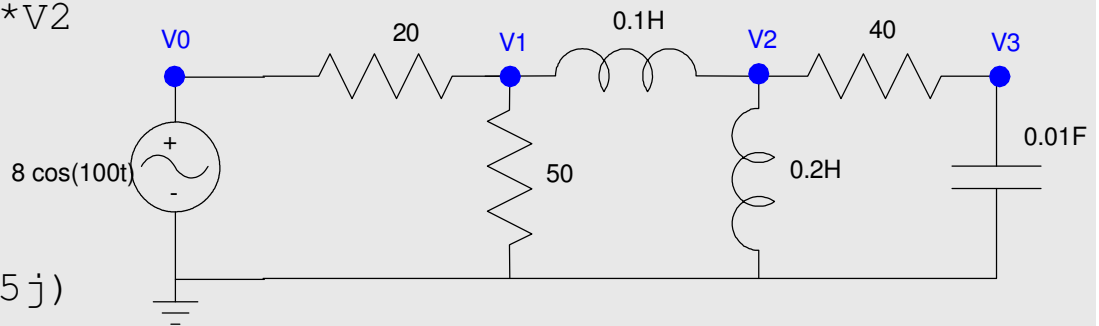
```
>>> w = 100
>>> Z30 = 1 / (j*w*0.01)
>>> Z20 = j*w*0.2
>>> Z12 = j*w*0.1
>>> Z3 = Z30
>>> Z2 = 1 / (1/Z20 + 1/(40 + Z30))
>>> Z1 = 1 / (1/50 + 1/(Z12 + Z20))
>>> V0 = 8 + j*0
>>> V1 = Z1 / (20 + Z1)*V0
>>> V2 = Z2 / (Z12 + Z2)*V1
>>> V3 = Z3 / (40 + Z3)*V2
```

```
>>> 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)
```

---

---

## Differences Between Matlab and MicroPython

Matlab and MicroPython are different

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

More specifically:

- Matlab is a matrix language
  - MicroPython is not
-



---

## Example:

- Matlab treats variables as matrices
- Python treats them as strings

### 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
```

### Thonny Shell

```
>>> 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
```

---

## 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
-

