# Problem 1: Python - the basic syntax

We are going to briefly introduce you to Python in this assignment. This introduction is by no means comprehensive. I highly recommend you brush up on Python through a few tutorials:

- https://wiki.python.org/moin/BeginnersGuide
- https://www.w3schools.com/python/

Python provides an extensive amount of documentation, e.g., https://docs.python.org/3.12/reference/index.html. Googling a command or question is also quite useful.

You will now go through a basic series of tutorials. Take as long or short as you need to ensure you feel like you know what is going on for the questions below. The tutorials have a fair amount of detail, so **you may want to skim over some of the topics and take note that they exist and come back to them as you need** (e.g., Python Operators are pretty close to c++, you might just scroll through the list and call it good and then come back later as needed). Come back to these tutorials throughout the semester as you need. From https://www.w3schools.com/python/, complete the following tutorials:

- Python Intro
- Python Syntax
- Python Comments`
- Python Variables
- Python Data Types
- Python Numbers
- Python Strings
- Python Booleans
- Python Operators
- Python Functions *[Math Processing Error]* Python Arguments

Note that in the code below there is an `import` statement. That statement imports a function from an existing package that allows the variables to be visualized within a Jupyter notebook.

In [6]:
```
from IPython.display import display # Used to display variables nicely in Ju

# Modify the x, y, and z variables to have the number one in a integer, floa
x = 0 # Should be an integer
display("x = ", x)
y = 0.0 # Should be a float
display("y = ", y)
z = "zero" # Should be a string
display("z = ", z)
```

```
'x = '
0
```

```
'y = '
0.0
'z = '
'zero'
```

In [5]:
```
# Add two to x, y, and z using the "+" operator
x = 0+2 # Add number two
display("x = ", x)
y = 0.0+2.0
display("y = ", y)
z = "zero" + "two"# Add the string "two"
display("z = " , z)
```

```
'x = '
2
'y = '
2.0
'z = '
'zerotwo'
```

## Problem 2: The list

The list is just what it sounds like. It provides a list of elements of any type. Complete the following tutorial and the coding exercise below.

Tutorial: https://www.w3schools.com/python/python_lists.asp

In [7]:
```
from IPython.display import display # Used to display variables nicely in Ju

# Create a list with the elements 1, 2., "three"
x = [1, 2.0, "three"]
display("x = ", x)

# Copy the 0th element to the variable zero
zero = x[0]
display("zero = ", zero)

# Copy the final element to the variable "final". Note that the index `-1` c
final = x[-1]
display("final = ", final)

# Append the item 4. to the end of the list and display the list
x.append(4.0)
display("x = ", x)
```

```
'x = '
[1, 2.0, 'three']
'zero = '
1
'final = '
'three'
'x = '
[1, 2.0, 'three', 4.0]
```

## Problem 3: The tuple

Tuples are similar to lists, but the collection is unchangeable. Complete the following tutorial and coding exercise.

Tutorial: https://www.w3schools.com/python/python_tuples.asp

In [8]:
```
from IPython.display import display # Used to display variables nicely in Ju

# Create a tuple with the elements 5, 7., and "apple"
x = (5, 7.0, "apple")

# Display the 0th element
zero = x[0]
display("zero = ", zero)

# Display the 1 element
one = x[1]
display("one = ", one)

# Display the last element
last = x[-1]
display("last = ", last)
```

```
'zero = '
5
'one = '
7.0
'last = '
'apple'
```

## Problem 4: The dict

The dictionary provides a mapping data type that you can use to map one item to another. Complete the following tutorial and coding exercise.

Tutorial: https://www.w3schools.com/python/python_dictionaries.asp

In [9]:
```
from IPython.display import display # Used to display variables nicely in Ju

# Create a dictionary that maps "apples" to "oranges" and "1" to "one"
x = {"apples": "oranges", 1: "one"}
display("x = ", x)

# Lookup the "apples" element from within x and display it
lookup = x["apples"]
display("lookup = ", lookup)

# Add the mapping from "two" to 2. and display the resulting dictionary
x["two"] = 2.0
display("x = ", x)
```

```
'x = '
{'apples': 'oranges', 1: 'one'}
'lookup = '
'oranges'
'x = '
{'apples': 'oranges', 1: 'one', 'two': 2.0}
```

## Problem 5: Structural programming

Complete the following tutorials from w3schools and the coding exercise below.

- Python If...Else
- Python While Loops
- Python For Loops

In [11]:
```
# Create a for loop that displays all of the elements in x one at a time
x = ["four", 5., 6]
for val in x:
    display(val)

# Create a for loop that loops through and displays the keys of y one at a t
y = {"four": 4, "five": 5., "six": 6.}
for key in y.keys():
    display(key)

# Create a for loop that iterates through the values in x, checks to see if
# If it is a key within y, then display the mapping to the value. If not, th
# the particular list value is not in y
for val in x:
    if val in y:
        display(val, " maps to ", y[val])
    else:
        display(val, " is not in y")
```

```
'four'
5.0
6
'four'
'five'
'six'
'four'
' maps to '
4
5.0
' is not in y'
6
' is not in y'
```

## Problem 6: Intro to Numpy

Numpy is an essential package developed for mathematical operations in Python. We will use it for extensively for matrix and general algebraic operations. Complete the following

tutorials and coding exercise.

- Numpy for beginners
- Numpy for Matlab users
- From w3schools.com
  - Getting started
  - Creating arrays

We will work quite heavily with numpy matrices. A numpy matrix can be created in a host of ways, but the most straight forward is to use the `np.array` initializer. In this case, each row of the matrix is initialized using an array and the matrix is an array of arrays. For example, the following matrix *[Math Processing Error]* can be initialized as

```
ex_mat = np.array([ [1., 2., 3.],
                    [4., 5., 6.]])
```

where the array `[1., 2., 3.]` is the first row and the array `[4., 5., 6.]` is the second.

We would like to perform the following matrix multiplication: *[Math Processing Error]*

There are two multiplication operators that you can utilize. The first is the asterisk, `*`, and the second is the at-symbol, `@`. Be careful as they produce very different results. Perform each multiplication, display the result, and answer the following questions.

## Question: What is the difference between `*` and `@`?

Answer: the `*` operator multiplies the matrices element by element, while the `@` properly multiplies the matrices

In [12]:
```
from IPython.display import display # Used to display variables nicely in Ju
import numpy as np

# Create the matrices (I've provided one for you)
A = np.array([[1, 2],
              [3, 4]])
B = np.array([[2, 1],
              [1, 0]])

# Multiply the matrices together (use the asterisk, i.e., A*B)
res_bad = A * B
display("Bad result = ", res_bad)

# Multiply the matrices together (use the ampersand, i.e., A@B)
res_good = A @ B
display("Good result: ", res_good)
```

```
'Bad result = '
```

```
array([[2, 2],
       [3, 0]])
'Good result: '
array([[ 4,  1],
       [10,  3]])
```

Now, perform the matrix multiplication for *[Math Processing Error]*

Calculate the shape of each matrix and the result and answer the following question:

## Question: What do the elements of the .shape tuple correspond to?

Answer: (rows, columns)

In [14]:
```
from IPython.display import display # Used to display variables nicely in Ju
import numpy as np

# Create the matrices (I've provided one for you)
A = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
B = np.array([[1], [2], [3]])

# Calculate the matrix multiplication of A and B
result = A @ B
display("A times B = ", result)

# Calculate the shape of each
display("Shape of A: ", A.shape)
display("Shape of B: ", B.shape)
display("Shape of result: ", result.shape)
```

```
'A times B = '
array([[14],
       [32],
       [50]])
'Shape of A: '
(3, 3)
'Shape of B: '
(3, 1)
'Shape of result: '
(3, 1)
```

Now, let's extract elements from the matrices. There are two main ways of getting an element out of a matrix.

1. Double indexing: `A[1,2]` gets the element from row 1 and column 2. Remember zero indexing!
2. .item: `A.item(4)` gets the fourth item stored in the matrix

Complete the following exercise and answer the following.

## Question: How would you relate the .item to rows and columns?

Answer: `.item` moves through the 2d array as if it were a 1d array and the rows were laid out end to end, rows and columns is a much more intuitive way to extract a single element but much more difficult to implement an iterative process with

In [ ]:
```
# Display items 0 through 8 of A using the .item function
for k in range(9):
    display("A.item(", k, ") = ", A.item(k))

# Use double indexing to extract the number 6 from A
res = A[1, 2]
display("Result: ", res)
```

```
'A.item('
0
') = '
1
'A.item('
1
') = '
2
'A.item('
2
') = '
3
'A.item('
3
') = '
4
'A.item('
4
') = '
5
'A.item('
5
') = '
6
'A.item('
6
') = '
7
'A.item('
7
') = '
8
'A.item('
8
') = '
9
'Result: '
np.int64(6)
```

**Be careful with dimensions**. Note the following two ways to extract the middle column of *A*. Calculate the shape of each of the results and answer the following question.

### Question: What is the difference between the two methods?

Answer: method 1 extracts the column as an array and outputs a list regardless of the original dimensions, method 2 results in a vertical vector that follows the original orientation

In [ ]:

In [17]:
```
# Note that you'll need to run the previous two cells before running this ce

# Method 1 for extracting the middle column:
mid_col_1 = A[:, 1]
display("Method 1: ", mid_col_1)

# Method 2 for extracting the middle column:
mid_col_2 = A[:, [1]]
display("Method 2: ", mid_col_2)

# Calculate the shape of each of the results
display("Method 1 shape: ", mid_col_1.shape)
display("Method 2 shape: ", mid_col_2.shape)
```

```
'Method 1: '
array([2, 5, 8])
'Method 2: '
array([[2],
       [5],
       [8]])
'Method 1 shape: '
(3,)
'Method 2 shape: '
(3, 1)
```

## Problem 7: Vector products

Assume that *[Math Processing Error]* represents a cross-product, *[Math Processing Error]* represents a dot product, and *[Math Processing Error]* represents matrix or scalar multiplication. The notation *[Math Processing Error]* is used to represent the transpose of *[Math Processing Error]*. Use the following vectors for this problem: *[Math Processing Error]*

Do the following

- Evaluate *[Math Processing Error]* using the function `np.dot`
- Evaluate *[Math Processing Error]* using the function `v1.transpose()` and matrix multiplication
- Evaluate *[Math Processing Error]* using the function `np.cross`

Answer the following question:

### Question: What is the difference between the result returned

from np.dot vs matrix multiplication for the dot product?

Answer: `np.dot` returned an int while matrix multiplication returned an array with a single element

In [18]:
```
from IPython.display import display # Used to display variables nicely in Ju
import numpy as np

# Define the vectors (ensure you define them as column vectors)
v1 = np.array([[1],
               [2],
               [3]])
v2 = np.array([[4],
               [5],
               [6]])

# Evaluate $v_1 \circ v_2$ using the function `np.dot`
# Note that the np.dot() function requires each input to
# be a vector and not a matrix. You can reshape the column
# vector into a numpy vector using np.reshape(v1, (3,))
dot_product = np.dot(np.reshape(v1, (3,)), np.reshape(v2, (3,)))
display("Dot product: ", dot_product)


# Evaluate $v_1^T \cdot v_2$ using the function `v1.transpose()` and matrix
transpose_product = v1.transpose() @ v2
display("Transpose product: ", transpose_product)

# Evaluate $v_1 \times v_2$ using the function `np.cross`
# Note that the np.dot() function requires each input to
# be a vector and not a matrix. You can reshape the column
# vector into a numpy vector using np.reshape(v1, (3,))
cross_product = np.cross(np.reshape(v1, (3,)), np.reshape(v2, (3,)))
display("Cross product: ", cross_product)
```

```
'Dot product: '
np.int64(32)
'Transpose product: '
array([[32]])
'Cross product: '
array([-3,  6, -3])
```

# Problem 8: Function Arguments

Python allows you to call functions using either positional arguments (same syntax as c++) or using keywords. The latter is similar to specifying a key-value pair for a dictionary. Using keywords will help you avoid a lot of bugs, especially as the number of the function arguments increases.

**I highly recommend that you almost always use keywords when calling a function.** It is so easy to swap the ordering of input arguments, but very difficult to see it when you are debugging.

Complete the following tutorial: Python Arguments and the following exercise.

In [19]:
```python
# Here is a function that will calculate the solution to the quadratic equat
def quadratic_solver(a: float, b: float, c: float) -> tuple[complex, complex
    """Solves the quadratic equation ax^2 + bx + c = 0

    Args:
        a: Coefficient of x^2
        b: Coefficient of x
        c: Constant term

    Returns:
        A tuple containing the two solutions (which may be complex)
    """
    discriminant = (b**2 - 4*a*c)**0.5
    root1 = complex((-b + discriminant) / (2*a))
    root2 = complex((-b - discriminant) / (2*a))
    return (root1, root2)


# Call the function using keyword arguments with a=1, b=2, c=-3 and display
solution_1, solution_2 = quadratic_solver(a=1, b=2, c=-3)

display("Solution 1: ", solution_1)
display("Solution 2: ", solution_2)

# Call the function using keyword arguments with a=2, b=1, c=4 and display t
solution_1, solution_2 = quadratic_solver(a=2, b=1, c=4)

display("Solution 1: ", solution_1)
display("Solution 2: ", solution_2)
```

```
'Solution 1: '
(1+0j)
'Solution 2: '
(-3+0j)
'Solution 1: '
(-0.24999999999999992+1.3919410907075054j)
'Solution 2: '
(-0.2500000000000001-1.3919410907075054j)
```

# Problem 9: Classes / Objects

Complete the following tutorials from w3schools and the coding exercise.

- Python Classes/Objects
- Python **init** method
- Python self Parameter
- Python Class Properties
- Python Class Methods

**Answer the following question truthfully:** Did you go complete the tutorials? Yes