**Classes:**

Syntax:

*class <name\_of\_class>(<inherited base class>):*

* Class sets up a new blueprint for class. It specifies the internal structure of these new types of objects, including what methods and operations they support. NOTE IT DOES NOT CREATE ANY OBJECT OF THAT TYPE (i.e. the class statement doesn’t create any instances of that type)
* When an object of a particular type I created, that object is sometimes called an instance of that type.

*class MyList(list):*

*def remove\_min(self):*

*self.remove(min(self))*

# functions defined inside a class are known as “instance methods" because they operate on an instance of the class. By convention the name of the class instance is called “self” and is always passed as they first argument.

*def remove\_max(self):*

*self.remove(max(self))*

*x=[1,2,2,4,5,3,1]*

*y=MyList(x)*

**NumPy:**

NumPy is a Python module designed for scientific computation. NumPy has several very useful features. Here are some examples. NumPy arrays are n-dimensional array objects, and they are a core component of scientific and numerical computation in Python. NumPy also provides tools for integrating your code with existing C, C++, and Fortran code. NumPy also provides many useful tools to help you perform linear algebra, generate random numbers, and much, much more. You can learn more about NumPy from the website numpy.org

Numpy arrays are used to representing vectors and matrices. Unlike dynamically growing lists, it has a fixed size which is defined when constructed. The datatype of all elements must be same. By default the datatype Is float. REMEMBER, to create a 2 or above dimensional array, a tuple should be the argument.

*import numpy as np*

*zero\_vector=np.zero(5)* #creates a vector with all elements zero and length 5 elements (one-dimensional array)

*zero\_matrix=np.zero((2,3))* #creates a matrix with 2 rows and 3 column (two-dimensional array)

You can similarly construct arrays with ones using similar syntax.

You can also use np.empty() which allocates the requested space for the array, but does not initialize it. This means the contents could be anything, whatever happens to be at the location where the array was set up. This could be useful when dealing with very large arrays and you are sure to edit all elements of the array as this COULD SAVE SOME COMPUTATIONAL TIME as python doesn’t need to initialize. However, care must be taken when handling them as beginners to NumPy.

We can also create our own arrays in the following manner:

*a = np.array([1,2,3])*

*A = np.array([[1,2,3],[4,5,6]])*

We can only use functions like transpose to the matrix:

*A.transpose()*

**Indexing and Slicing in NumPy:**

List type indexing also applies to NumPy arrays

1. Adding two arrays of same length (if different returns error) adds each element like a matrix
2. *A[:,1]* returns the 1 indexed column ( i.e. the second column), the return type is a new NumPy array
3. Remember comparing lists and NumPy arrays following is True- A[1, :]==A[1]
4. Arrays can be used to index and refer to a certain group of elements. Again result is a NumPy array

*A=np.array([1,3,5,7]) ; ind=[0,2]* *🡨* can also be NumPy array

*A[ind] 🡪 array( [ 1 , 5 ] )*

1. Doing a comparison or conditional expression to a NumPy array, it does the checking for each element and returns a NumPy array or Boolean values depicting the results of each element.

*Z1=np.array(1,2,3,4,5)*

*Z1>3 🡪 array( [ False , False , False , True , True ] , dtype=bool)*

1. The above logic array can used to index NumPy arrays as well:

*Z1[ Z1 > 3] 🡪 array( [ 4 , 5 ] )*

Also:

*z1=np.array( [1,2,3,4,5,6] )*

*z2=np.array( [5,2,4,5,1,0] )*

*ind = z1 >3*

*z2[ind] 🡪 array( [ 5, 1, 0] )*

We see above the condition is checked for the array z1, but the elements returned are from z2, thus the positions are matched. Remember if the lengths are not same it will result in error.

REMEMBER, slicing an array or list shows you part of the original object. If you edit it, it will update the original list/array. While if you assign the return of a index of an array, it creates new copy. Example:

*Z1=np.array( [ 1 , 2 , 3 , 4] )*

*X = Z1[0:2]*

*X[0]=10000*

*Z1 🡪 array( [ 10000 , 2 , 3 , 4 ] )* #this doesn’t happen if we use indexing rather

**np.linspace():**

This function can be used to construct an equally spaced *linear* array of n characters.

Syntax:

*np.linspace( <starting\_point\_inculded> , <ending\_point\_inculded> , <number\_of\_characters>)*

Example:

*np.linspace(0,100,10) 🡪 array([ 0. , 11.11111111, 22.22222222, 33.33333333, 44.44444444,*

*55.55555556, 66.66666667, 77.77777778, 88.88888889, 100. ])*

**np.logspace():**

This function can be used to construct an equally spaced *logarithmic* array of n characters. The syntax is similar to that of the linspace(). However, REMEMBER, the starting point and ending point must be the log10 of the required starting/ending points.

Example:

*np.logspace(1,2,10) #also np.logspace(np.log10(10) , nplog10(100) , 10)*

*🡪 array([ 10. , 12.91549665, 16.68100537, 21.5443469 ,*

*27.82559402, 35.93813664, 46.41588834, 59.94842503,*

*77.42636827, 100. ])*

**Size and Shape:**

Attributes that can accessed to assess the array. REMEMBER, these are attributes not methods.

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

X.shape 🡪 (2, 3)

X.size 🡪 6

**Some more methods:**

1. **np.any(<condition>) 🡨** checks if any of the elements in the array satisfy condition, returns bool value. Internally it checks for any True in the array produced by condition.
2. **np.all(<condition>) 🡨** checks if all of the elements in the array satisfy condition returns bool value. Internally it checks is all are True in the array produced by condition.
3. **np.random.random(<length\_of\_array>) 🡨** NumPy has its own random function. Random.random returns any value in the interval (0,1). And n number of then will be generated and formed into a NumPy array.

Ex:

*x=np.random.random(10)*

*np.any(x>0.9) 🡪 True*

*np.all(x>=0.1) 🡪 True*

*x 🡪 array([0.45646105, 0.91693989, 0.76692942, 0.78663869, 0.40875667, 0.42883111, 0.45244747, 0.18175371, 0.88602548, 0.57623867])*

Example:

**Program checking if number is prime:**

*x = 20*

*not np.any([x%i == 0 for i in range(2, x)])*