**1**

Find the last element of a list.  
Eg., l = [1, 2, 3, 4]  
last\_element(l) -> 4

def last\_element(l):

"""

Returns last element of list l

"""

# YOUR CODE HERE

print (l[-1])

last\_element([1])

last\_element([“Hello”, “World”])

"""Testing code for last\_element"""

assert last\_element([1]) == 1

assert last\_element(["Hello", "World"]) == "World"

**2**

Find the number of elements in a list.  
Eg., l = [1, 2, 3, 4]  
num\_elements(l) -> 4

def num\_elements(l):

"""

Returns number of elements in the list l

"""

# YOUR CODE HERE

print len(l)

num\_elements([2, 3, 4])

"""Testing code for num\_elements"""

assert num\_elements([2, 3, 4])==3

**3**

Reverse a list and return it.  
Eg., l = [1, 2, 3, 4]  
reverse\_list(l) -> [4, 3, 2, 1]

def reverse\_list(l):

"""

Returns the reverse of a list l

"""

# YOUR CODE HERE

l.reverse()

print(l)

reverse\_list([2, 3, 4])

"""Testing code for reverse\_list"""

assert reverse\_list([2, 3, 4])== [4, 3, 2]

**4**

Find whether a list a palindrome -> a sequence that reads the same forwards and backwards  
is\_palindrome([1, 2, 1]) -> True  
is\_palindrome([1, 2, 3, 2, 1] -> True  
is\_palindrome([1, 2, 3, 1]) -> False

def is\_palindrome(l):

"""

Returns True if l is a palindrome, False otherwise

"""

# YOUR CODE HERE

lr=l.copy()

l.reverse()

if list(l)==list(lr):

print(“True”)

else:

print(“False”)

is\_palindrome([1, 2, 1])

is\_palindrome([1, 2, 3, 2, 1])

is\_palindrome([1, 2, 3, 4])

"""Testing code for is\_palindrome"""

assert(is\_palindrome([1, 2, 1])) == True

assert(is\_palindrome([1, 2, 3, 2, 1])) == True

assert(is\_palindrome([1, 2, 3, 4])) == False

**5**

Replace consecutive duplicate elements of list with single element.  
Eg., l = [a, a, a, b, b, c, a, a, d, d, d, x, x]  
compress(l) -> [a, b, c, a, d, x]

def compress(l):

"""

Returns a list with consecutive duplicate elements replaced by a single element

"""

# YOUR CODE HERE

i = 0

while i < len(l)-1:

if l[i] == l[i+1]:

del list[i]

else:

i = i+1

print(l)

compress([1, 2, 2])

compress([1, 2, 2, 2, 1, 1, 3, 'x', 'x', 'x'])

"""Testing code for compress"""

assert(compress([1, 2, 2])) == [1, 2]

assert(compress([1, 2, 2, 2, 1, 1, 3, 'x', 'x', 'x'])) == [1, 2, 1, 3, 'x']

**6**

Pack consecutive duplicates of list elements into sublists.  
Eg., l = [1, 1, 1, 2, 2, 3, 3, 4] pack(l) -> [ [1,1,1], [2,2], [3,3], [4] ]

def pack(l):

"""

Returns a list with consecutive duplicate elements packed into sublists

"""

# YOUR CODE HERE

"""Testing code for pack"""

assert(pack([1, 1, 1, 2]) == [[1, 1, 1], [2]])

assert(pack([1, 1, 1, 2, 1, 1, 3, 3, 3])) == [[1, 1, 1], [2], [1, 1], [3, 3, 3]]

**7**

Given two indices, i and k, the slice is the list containing the elements between the i'th and k'th element of the original list (left limit included). Start counting the elements with 0.

Eg., l = [1, 3, 9, 8, 7]; slice(l, 1, 3) --> [3, 9]

def slice(l, i, k):

"""

Returns a list containing the elements between i'th and k'th elements of original list l.

"""

# YOUR CODE HERE

print(l[i:k])

slice([1, 3, 8, 9, 7], 1, 3)

slice([1, 4, 6, 'x', 9, 0], 2, 10)

"""Testing code for slice"""

assert(slice([1, 3, 8, 9, 7], 1, 3)) == [3,8]

assert(slice([1, 4, 6, 'x', 9, 0], 2, 10)) == [6, 'x', 9, 0]

**8**

Given a list l, index i and element elem, return a new list with elem at index i  
Eg., l = [1, 3, 9, 8, 7];  
insert\_element(l, 1, 3) --> [1, 3, 9, 1, 8, 7]

def insert\_element(l, i, elem):

"""

Returns a new list containing elem at index i. If i > len (l), insert element at the end of the list

"""

# YOUR CODE HERE

l.insert(i, elem)

print(l)

insert\_element([1, 2, 3, 4], 2, 5)

insert\_element([1, 5], 3, 5)

"""Testing code for insert\_element"""

assert(insert\_element([1, 2, 3, 4,], 2, 5)[2]) == 5

assert(insert\_element([1, 5, ], 3, 5)) == [1, 5, 5]

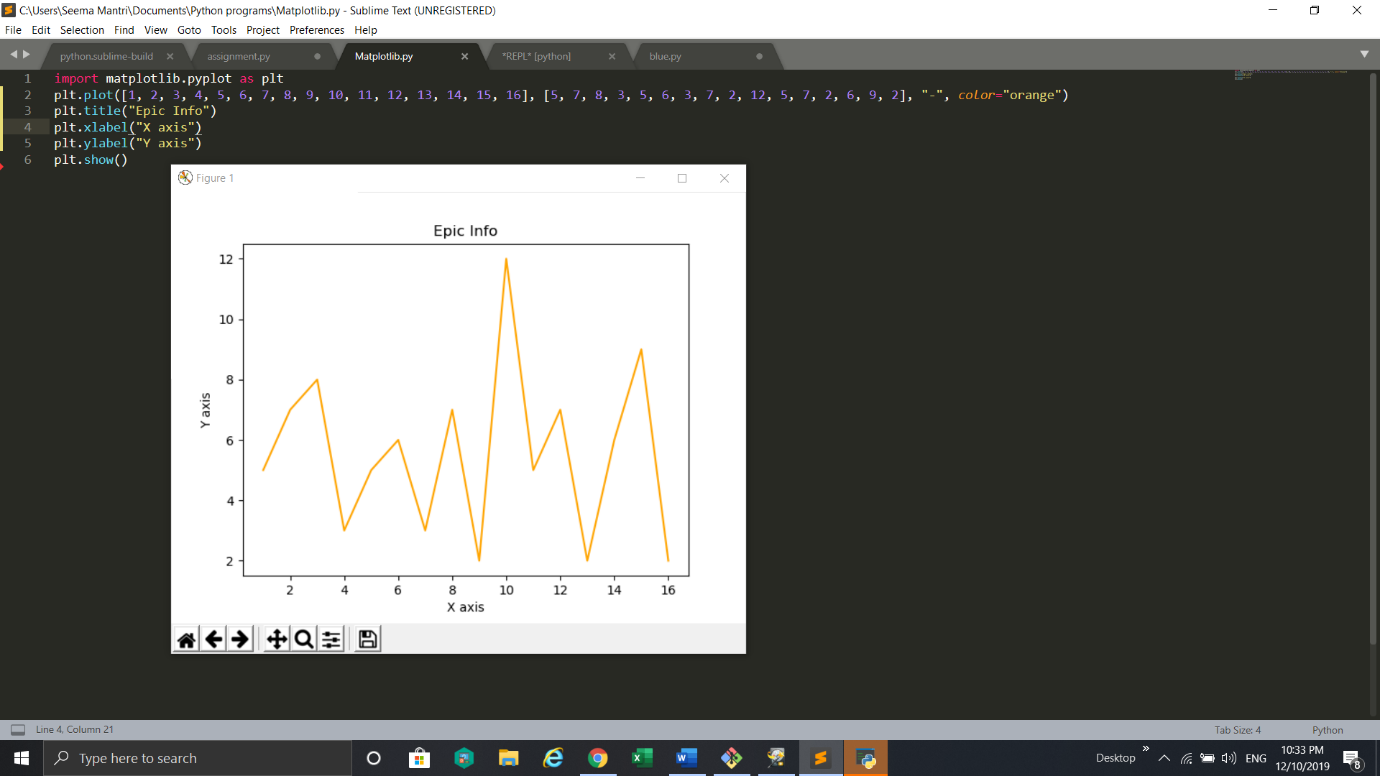
import matplotlib.pyplot as plt

plt.plot([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16], [5, 7, 8, 3, 5, 6, 3, 7, 2, 12, 5, 7, 2, 6, 9, 2], "-", color="orange")

plt.title("Epic Info")

plt.xlabel("X axis")

plt.ylabel("Y axis")

plt.show()

#How do I get the grey background?

### Windows and Strides

Given a 1d array, and parameters window\_len and stride\_len, do the following:

* Divide the 1d array into smaller arrays of size window\_len starting at index 0
* move the starting index by step size stride\_len each time
* Return all windows as a 2d matrix

Eg., Input:  
arr = [1, 3, 7, 1, 2, 6, 0, 1]  
stride\_len = 2 , window\_len = 4

* First window (starting at index 0) should be of len window\_len (4) --> [1, 3, 7, 1]
* Move by step size stride\_len(2) and take next window --> [7, 1, 2, 6]  
  and so on..

Output matrix  
[[1, 3, 7, 1],  
[7, 1, 2, 6],  
[2, 6, 0, 1]]

def gen\_strides(a, stride\_len, window\_len):

'''

Input:

a: Numpy array of 1 dimension

stride\_len: int, stride length

window\_len : int, window length

Output:

Numpy array of 2 dimensions containing windowed strides as explained above

'''

# YOUR CODE HERE

"""Test for strides"""

assert (np.all(gen\_strides(np.array([1, 3, 7, 1, 2, 6, 0, 1]),2,4) == np.array([[1, 3, 7, 1], [7, 1, 2, 6], [2, 6, 0, 1]])))

print("Sample Tests passed", '\U0001F44D')

### Shuffle

Given a numpy array of arbitrary dimensions (> 1), shuffle its rows randomly.  
Hint - You need to shuffle along axis 0

def shuf(arr):

'''

Input:

arr: Numpy array of arbitrary number of dimensions (>1)

Output:

numpy array of same shape as arr but with rows shuffled

'''

# YOUR CODE HERE

"""Test for shuf"""

arr=np.array([[1, 2, 3],[4, 5, 6],[7, 8, 9]])

assert np.any(shuf(np.array([[1, 2, 3],[4, 5, 6],[7, 8, 9]])) != np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]]))

assert shuf(np.array([[1, 2, 3],[4, 5, 6],[7, 8, 9]])).shape == np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]]).shape

print("Sample Tests passed", '\U0001F44D')

### Match

Get the positions where corresponding elements (same indices) of array a and array b match

Eg. Input <br>

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

b = np.array([7,2,10,2,7,4,9,4,9,8])

Desired Output:

[1, 3, 5, 7])

def match(a,b):

'''

Inputs:

a, b: numpy arrays of same shape of 1 dimension

Outputs:

list containing indices where both arrays have same elements

'''

# YOUR CODE HERE

  list(a) = set(a)

    list(b) = set(b)

    if len(a\_set.intersection(b\_set)) > 0:

         print(a\_set.intersection(b\_set))

    else:

        return("no common elements")

match([1, 2, 3, 2, 3, 4, 3, 4, 5, 6], [7, 2, 10, 2, 7, 4, 9, 4, 9, 8])

"""Test for match"""

assert(match(np.array([1,2,3,2,3,4,3,4,5,6]),np.array([7,2,10,2,7,4,9,4,9,8])) == [1,3,5,7])

print("Sample Tests passed", '\U0001F44D')

### Inverse of an array

**Hint:** Search numpy library for inverse function

import numpy as np

def inv(arr):

"""

Given an array arr (square matrix), find its inverse

"""

# YOUR CODE HERE

iarr=np.linalg.inv(arr)

print(iarr)

inv([[6, 1, 1], [4, -2, 5], [2, 8, 7]])

if(arr.shape[0]==arr.shape[1]):

i=np.eye(arr.shape[0])

return np.linalg.inv(arr)

print(inv(np.array([[6, 1, 1], [4, -2, 5], [2, 8, 7]])))

[[ 0.17647059 -0.00326797 -0.02287582]

[ 0.05882353 -0.13071895 0.08496732]

[-0.11764706 0.1503268 0.05228758]]

"""Test for inv"""

assert np.all(np.isclose(inv(np.array([[6, 1, 1], [4, -2, 5], [2, 8, 7]])).tolist(), np.array([[0.17647058823529413, -0.0032679738562091526, -0.02287581699346405],[0.05882352941176469, -0.130718954248366, 0.0849673202614379],[-0.1176470588235294, 0.1503267973856209, 0.0522875816993464]])))

print("Sample Tests passed", '\U0001F44D')

### Linear Equations

**Hint**: x = inverse\_of\_a \* b.

def lin\_eqn(a,b):

'''

Solve the system of linear equations

of the form ax = b

Eg.

Solve the system of linear equation

x + 2\*y = 8

3\*x + 4\*y = 18

Given inputs a and b represent coefficients and constant of linear equation respectively

coefficients:

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

constants:

b = np.array([8, 18])

Desired Output: [2,3]

'''

# YOUR CODE HERE

c= np.linalg.inv(arr)

list(sol)=c\*b

print(sol)

lin\_eqn([[1, 2], [3, 4]], [8, 18])

x=lin\_eqn(np.array([[1.0, 2.0], [3, 4]]),np.array([8.0, 18.0]))

print(x)

[2. 3.]

"""Test for lin\_eqn"""

assert np.any(lin\_eqn(np.array([[1, 2], [3, 4]]),np.array([8, 18])) == np.array([2., 3.]))

print("Sample Tests passed", '\U0001F44D')

### rankArray

Rank the items in a multidimensional array 'arr'.  
The rank of an item is its index in the sorted list of all items in 'arr' (starting from 0).  
Eg., consider arr = [0, 6, 14, 12, 11]  
arr\_sorted = [0, 6, 11, 12, 14]  
So rank of 0 -> 0; 6->1; 11->2, 12->3, 14->4  
rankArray returns a list where each element is replaced by its rank  
rankArray(arr) -> [0, 1, 4, 3, 2]

==========================================  
Another example,  
Eg:

arr = [ [ 9 4 15 0 17], [16 17 8 9 0] ] ... (2X5 array)

Desired output:  
[ [4 2 6 0 8], [7 9 3 5 1] ] ... (2X5 array)

Here minimum value in arr is 0 at (0,3), so rank the index of 0 in output matrix as 0  
i.e. out[0][3] = 0  
Next minimum is also 0 at index (1,4), so rank the index (1,4) in output matrix as 1 and so on...

============================================= If two elements repeat; for eg., both a1 = arr[0, 1] and a2 = arr[0, 3] are equal to 1, then rank of the a1 is lower than the rank of a2 because at the first index at which they differ, the index of a1 is lower than that of a2 (similar to lexicographical ordering)

import numpy as np

def rankArray(arr):

'''

Input:

arr: Numpy array of arbitrary dimensions

Output:

numpy array of same shape as arr but with elements replaced by their ranks

'''

sarr=sorted(arr)

rarr=[sarr.index(i) for i in arr]

print(raar)

rankArray([[9, 4, 15, 0, 17], [16,17,8,9,0]])

# YOUR CODE HERE

print(rankArray(np.array([[9, 4, 15, 0, 17], [16,17,8,9,0]])).tolist())

[[4, 2, 6, 0, 8], [7, 9, 3, 5, 1]]

"""Test for rankArray"""

assert np.all(rankArray(np.array([[9, 4, 15, 0, 17], [16,17,8,9,0]])) == np.array([[4,2, 6, 0, 8], [7, 9, 3, 5, 1]]).tolist())

print("Sample Tests passed", '\U0001F44D')