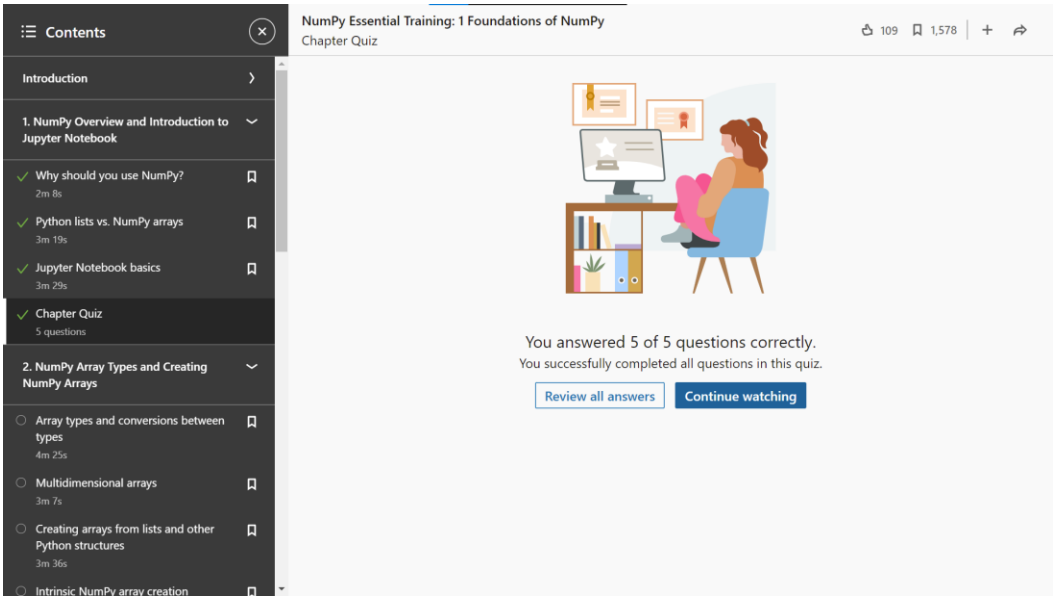
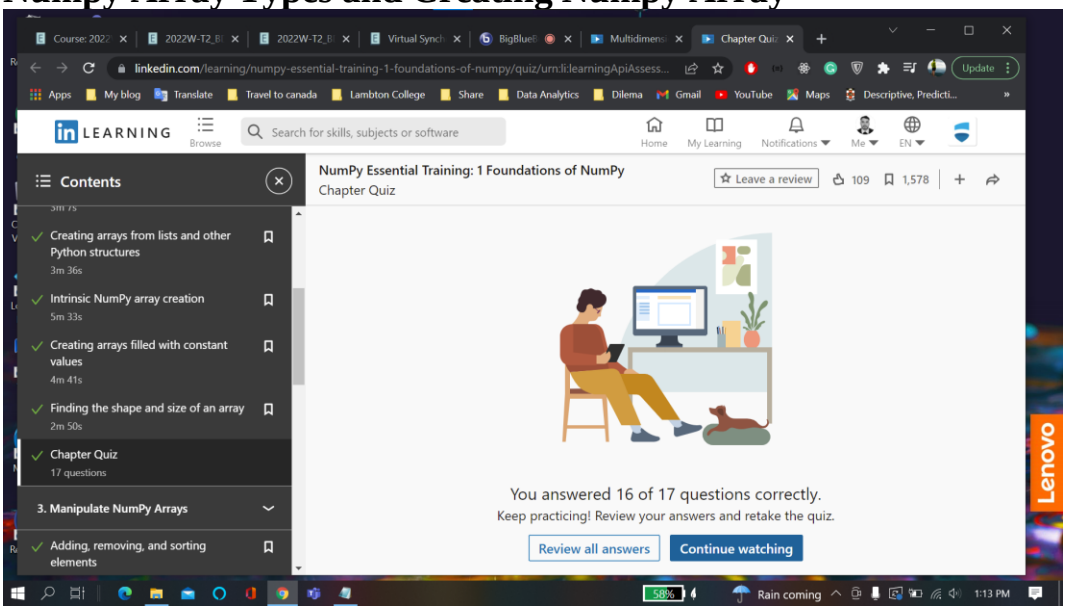
BDM 1034 - Application Design for Big Data
Week9
Submitted by: Aadarsha Chapagain
Student ID:C0825975
Submitted to: Prof. Teresa Zhu

Here I have attached the certificate I achieved from Linkedin learning for Course "***NumPy Essential Training: 1 Foundations of NumPy***" along with the Screenshot of score I achieved

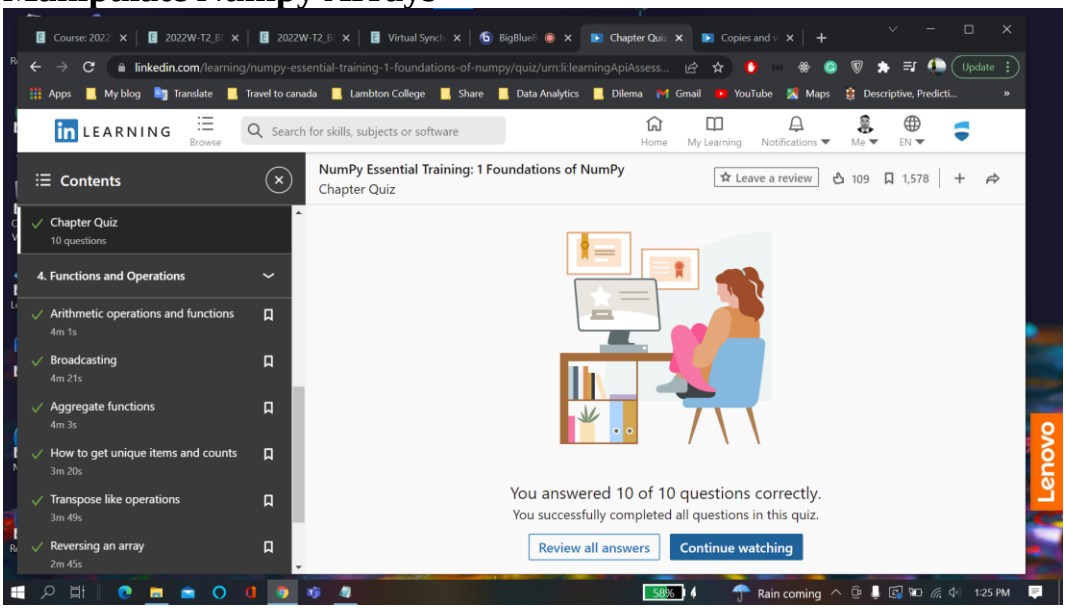
Chapter 1:
Overview



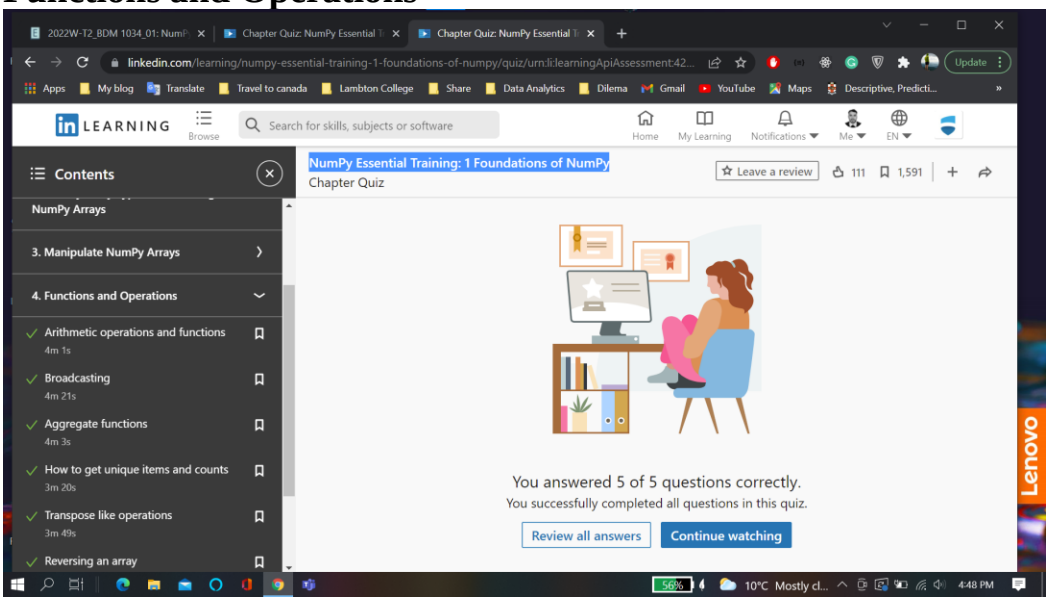
Chapter 2
Numpy Array Types and Creating Numpy Array



Chapter 3:
Manipulate Numpy Arrays



Chapter 4:
Functions and Operations



Numpy_essentials_part1

March 20, 2022

1 NumPy Essential Training: 1 Foundations of NumPy

1.1 Chapter 1 : Overview

[1]:
```
2+8*5
```

[1]:
```
42
```

1.2 Chapter 2 : Numpy array types and creating Numpy Arrays

1.2.1 Array types and conversions between types

[2]:
```
import numpy as np
```

[4]:
```
integers=np.array([10,20,30,40,50])
```

[5]:
```
integers
```

[5]:
```
array([10, 20, 30, 40, 50])
```

[6]:
```
integers[0]
```

[6]:
```
10
```

[7]:
```
integers[0]=20
integers
```

[7]:
```
array([20, 20, 30, 40, 50])
```

[8]:
```
integers[0]=21.5
integers
```

[8]:
```
array([21, 20, 30, 40, 50])
```

[9]:
```
integers.dtype
```

[9]:
```
dtype('int32')
```

[10]:
```
smallerIntegers=np.array(integers,dtype=np.int8)
```

[12]:
```
smallerIntegers
```

[12]:
```
array([21, 20, 30, 40, 50], dtype=int8)
```

[13]:
```
integers.nbytes
```

[13]:
```
20
```

[14]:
```
overflow = np.array([127,128,129], dtype = np.int8)
overflow
```

[14]:
```
array([ 127, -128, -127], dtype=int8)
```

[16]:
```
floats=np.array([1.2,2.3,3.4,5.1,8.3])
```

[17]:
```
floats
```

[17]:
```
array([1.2, 2.3, 3.4, 5.1, 8.3])
```

[18]:
```
floats.dtype
```

[18]:
```
dtype('float64')
```

1.2.2 Multidimensional array

[19]:
```
nums=np.array([[1,2,3,4,5],[6,7,8,9,10]])
nums
```

[19]:
```
array([[ 1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10]])
```

[21]:
```
nums[0,0]
```

[21]:
```
1
```

[22]:
```
nums[1,4]
```

[22]:
```
10
```

[23]:
```
nums.ndim
```

[23]:
```
2
```

[24]:
```
multi_arr=np.array([[[1,2,3],[4,5,6]],[[7,8,9],[10,11,12]]])
```

[25]:
```
multi_arr
```

[25]:
```
array([[[ 1,  2,  3],
        [ 4,  5,  6]],

       [[ 7,  8,  9],
        [10, 11, 12]]])
```

[26]:
```
multi_arr[1,0,2]
```

[26]:
```
9
```

1.2.3 Creating arrays from Lists and other Python structures

[27]:
```
first_list=[1,2,3,4,5,6,7,8,9,10]
```

[28]:
```
first_list
```

[28]:
```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

[29]:
```
first_array=np.array(first_list)
```

[30]:
```
first_array
```

[30]:
```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

[32]:
```
second_list=[1,2,3,-1.23,50,128000.56,4.56]
```

[33]:
```
second_array=np.array(second_list)
```

[34]:
```
second_array.dtype
```

[34]:
```
dtype('float64')
```

[35]:
```
third_list=['Ann',111111,'Peter',111112,'Susan',111113,'John',111114]
```

[36]:
```
third_array=np.array(third_list)
```

[37]:
```
third_array
```

[37]:
```
array(['Ann', '111111', 'Peter', '111112', 'Susan', '111113', 'John',
       '111114'], dtype='<U11')
```

[38]:
```
first_tuple =(5,10,15,20,25,30)
```

[39]:
```
array_from_tuple=np.array(first_tuple)
array_from_tuple
```

[39]:
```
array([ 5, 10, 15, 20, 25, 30])
```

[40]:
```
array_from_tuple.dtype
```

[40]:
```
dtype('int32')
```

[41]:
```
multi_dim_list=[[[0,1,2], [3,4,5]], [[6,7,8],[9,10,11]]]
```

[42]:
```
arr_from_multi_dim_list=np.array(multi_dim_list)
```

[43]:
```
arr_from_multi_dim_list
```

[43]:
```
array([[[ 0,  1,  2],
        [ 3,  4,  5]],

       [[ 6,  7,  8],
        [ 9, 10, 11]]])
```

1.2.4 Intrisic NumPy array creation

[44]:
```
integers_array=np.arange(10)
integers_array
```

[44]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

[45]:
```
integers_second_array=np.arange(100,130)
integers_second_array
```

[45]:
```
array([100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112,
       113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125,
       126, 127, 128, 129])
```

[47]:
```
integers_third_array=np.arange(100,151,2)
integers_third_array
```

[47]:
```
array([100, 102, 104, 106, 108, 110, 112, 114, 116, 118, 120, 122, 124,
       126, 128, 130, 132, 134, 136, 138, 140, 142, 144, 146, 148, 150])
```

[48]:
```
first_floats_arr=np.linspace(10,20)
first_floats_arr
```

[48]:
```
array([10.        , 10.20408163, 10.40816327, 10.6122449 , 10.81632653,
       11.02040816, 11.2244898 , 11.42857143, 11.63265306, 11.83673469,
       12.04081633, 12.24489796, 12.44897959, 12.65306122, 12.85714286,
       13.06122449, 13.26530612, 13.46938776, 13.67346939, 13.87755102,
       14.08163265, 14.28571429, 14.48979592, 14.69387755, 14.89795918,
       15.10204082, 15.30612245, 15.51020408, 15.71428571, 15.91836735,
       16.12244898, 16.32653061, 16.53061224, 16.73469388, 16.93877551,
       17.14285714, 17.34693878, 17.55102041, 17.75510204, 17.95918367,
       18.16326531, 18.36734694, 18.57142857, 18.7755102 , 18.97959184,
```

```
       19.18367347, 19.3877551 , 19.59183673, 19.79591837, 20.        ])
```

[49]:
```
second_floats_arr=np.linspace(10,20,5)
second_floats_arr
```

[49]:
```
array([10. , 12.5, 15. , 17.5, 20. ])
```

[50]:
```
first_rand_arr=np.random.rand(10)
first_rand_arr
```

[50]:
```
array([0.28083474, 0.72100694, 0.49540719, 0.63129895, 0.95275235,
       0.95169323, 0.78251022, 0.69581257, 0.00405148, 0.57213105])
```

[51]:
```
second_rand_arr=np.random.rand(4,4)
second_rand_arr
```

[51]:
```
array([[0.83787471, 0.95592772, 0.54018326, 0.95119388],
       [0.02367659, 0.65404243, 0.63859117, 0.98532049],
       [0.48937791, 0.23079885, 0.70746353, 0.07578554],
       [0.87183243, 0.54870318, 0.13721802, 0.33023857]])
```

[52]:
```
third_rand_arr=np.random.randint(0,100,20)
third_rand_arr
```

[52]:
```
array([38, 56, 75, 20, 61, 28, 88,  6, 68, 17, 11, 25, 40, 15, 63, 79, 46,
       41, 17, 32])
```

1.2.5 Creating arrays filled with constant values

[53]:
```
first_z_array=np.zeros(5)
first_z_array
```

[53]:
```
array([0., 0., 0., 0., 0.])
```

[54]:
```
second_z_array=np.zeros((4,5))
second_z_array
```

[54]:
```
array([[0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.]])
```

[55]:
```
first_ones_array=np.ones(6)
first_ones_array
```

[55]:
```
array([1., 1., 1., 1., 1., 1.])
```

[56]:
```
second_ones_array=np.ones((7,8))
second_ones_array
```

[56]:
```
array([[1., 1., 1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1., 1., 1.]])
```

[57]:
```
third_ones_array=np.ones((4,5),dtype=int)
third_ones_array
```

[57]:
```
array([[1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1]])
```

[58]:
```
first_fill_array=np.empty(10,dtype=int)
first_fill_array.fill(12)
first_fill_array
```

[58]:
```
array([12, 12, 12, 12, 12, 12, 12, 12, 12, 12])
```

[59]:
```
first_full_array=np.full(5,10)
first_full_array
```

[59]:
```
array([10, 10, 10, 10, 10])
```

[60]:
```
second_full_array=np.full((4,5),8)
second_full_array
```

[60]:
```
array([[8, 8, 8, 8, 8],
       [8, 8, 8, 8, 8],
       [8, 8, 8, 8, 8],
       [8, 8, 8, 8, 8]])
```

1.2.6 Finding the shape and size of an array

[61]:
```
first_arr=np.arange(20)
first_arr
```

[61]:
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19])
```

[62]:
```
second_arr=np.linspace((1,2),(10,20),10)
second_arr
```

[62]:
```
array([[ 1.,  2.],
       [ 2.,  4.],
       [ 3.,  6.],
       [ 4.,  8.],
       [ 5., 10.],
       [ 6., 12.],
       [ 7., 14.],
       [ 8., 16.],
       [ 9., 18.],
       [10., 20.]])
```

[63]:
```
third_arr=np.full((2,2,2),10)
third_arr
```

[63]:
```
array([[[10, 10],
        [10, 10]],

       [[10, 10],
        [10, 10]]])
```

[64]:
```
np.shape(first_arr)
```

[64]:
```
(20,)
```

[65]:
```
np.shape(second_arr)
```

[65]:
```
(10, 2)
```

[66]:
```
np.shape(third_arr)
```

[66]:
```
(2, 2, 2)
```

[67]:
```
np.size(first_arr)
```

[67]:
```
20
```

[69]:
```
np.size(second_arr)
```

[69]:
```
20
```

[70]:
```
np.size(third_arr)
```

[70]:
```
8
```

1.3 Chapter 3 : Manipulate Numpy arrays

1.3.1 Adding, removing and sorting elements

[71]:
```
first_arr=np.array([1, 2, 3, 5])
first_arr
```

[71]:
```
array([1, 2, 3, 5])
```

[72]:
```
new_first_arr=np.insert(first_arr,3,4)
new_first_arr
```

[72]:
```
array([1, 2, 3, 4, 5])
```

[73]:
```
second_arr=np.array([1,2,3,4])
second_arr
```

[73]:
```
array([1, 2, 3, 4])
```

[74]:
```
new_second_arr=np.append(second_arr,5)
new_second_arr
```

[74]:
```
array([1, 2, 3, 4, 5])
```

[75]:
```
third_arr=np.array([1,2,3,4,5])
third_arr
```

[75]:
```
array([1, 2, 3, 4, 5])
```

[76]:
```
del_arr=np.delete(third_arr,4)
del_arr
```

[76]:
```
array([1, 2, 3, 4])
```

[77]:
```
integers_arr=np.random.randint(0,20,20)
integers_arr
```

[77]:
```
array([15,  0, 19, 12, 16,  6,  6,  9,  7, 11,  2, 12,  4,  2,  0, 14, 16,
        3, 17, 15])
```

[78]:
```
print(np.sort(integers_arr))
```

```
[ 0  0  2  2  3  4  6  6  7  9 11 12 12 14 15 15 16 16 17 19]
```

[79]:
```
integers_2dim_arr=np.array([[3, 2, 5,7, 4], [5, 0, 8,3, 1]])
integers_2dim_arr
```

[79]:
```
array([[3, 2, 5, 7, 4],
       [5, 0, 8, 3, 1]])
```

[80]:
```
print(np.sort(integers_2dim_arr))
```

```
[[2 3 4 5 7]
 [0 1 3 5 8]]
```

[81]:
```
colors=np.array(['orange','green','yellow','white','black','pink','blue','red'])
colors
```

[81]:
```
array(['orange', 'green', 'yellow', 'white', 'black', 'pink', 'blue',
       'red'], dtype='<U6')
```

[82]:
```
print(np.sort(colors))
```

```
['black' 'blue' 'green' 'orange' 'pink' 'red' 'white' 'yellow']
```

1.3.2 Copies and views

[83]:
```
students_ids_number=np.array([1111,1212,1313,1414,1515,1616,1717,1818])
students_ids_number
```

[83]:
```
array([1111, 1212, 1313, 1414, 1515, 1616, 1717, 1818])
```

[84]:
```
students_ids_number_reg=students_ids_number
print("id of students_ids_number",id(students_ids_number))
print("id of students_ids_number_reg",id(students_ids_number_reg))
```

```
id of students_ids_number 1717033159632
id of students_ids_number_reg 1717033159632
```

[85]:
```
students_ids_number_reg[1]=2222
print(students_ids_number)
print(students_ids_number_reg)
```

```
[1111 2222 1313 1414 1515 1616 1717 1818]
[1111 2222 1313 1414 1515 1616 1717 1818]
```

[87]:
```
students_ids_number_cp=students_ids_number.copy()
print(students_ids_number_cp)
```

```
[1111 2222 1313 1414 1515 1616 1717 1818]
```

[88]:
```
print(students_ids_number_cp==students_ids_number)
```

```
[ True  True  True  True  True  True  True  True]
```

[89]:
```
print ("id of students_ids_number",id(students_ids_number))
print("id of students_ids_number_cp",id(students_ids_number_cp))
```

```
id of students_ids_number 1717033159632
id of students_ids_number_cp 1717033159440
```

[90]:
```
students_ids_number[0]=1000
print ("original: ", students_ids_number)
print("copy: ",students_ids_number_cp)
```

```
original:  [1000 2222 1313 1414 1515 1616 1717 1818]
copy:  [1111 2222 1313 1414 1515 1616 1717 1818]
```

[92]:
```
students_ids_number_v=students_ids_number.view()
```

[94]:
```
students_ids_number_v[0]=2000
print("original: ", students_ids_number)
print("view:",students_ids_number_v)
```

```
original:  [2000 2222 1313 1414 1515 1616 1717 1818]
view: [2000 2222 1313 1414 1515 1616 1717 1818]
```

[95]:
```
print(students_ids_number_cp.base)
print(students_ids_number_v.base)
```

```
None
[2000 2222 1313 1414 1515 1616 1717 1818]
```

1.3.3 Reshaping Arrays

[96]:
```
first_arr=np.arange(1,13)
first_arr
```

[96]:
```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12])
```

[97]:
```
second_arr=np.reshape(first_arr,(3,4))
second_arr
```

[97]:
```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12]])
```

[98]:
```
third_arr=np.reshape(first_arr,(6,2))
third_arr
```

[98]:
```
array([[ 1,  2],
       [ 3,  4],
       [ 5,  6],
       [ 7,  8],
       [ 9, 10],
       [11, 12]])
```

[99]:
```
fourth_arr=np.reshape(first_arr,(4,4))
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Input In [99], in <module>
----> 1 fourth_arr=np.reshape(first_arr,(4,4))

File <__array_function__ internals>:180, in reshape(*args, **kwargs)

File E:
 ↪\Anaconda\envs\ApplicationDesignBDM1034\lib\site-packages\numpy\core\fromnume ic.
 ↪py:298, in reshape(a, newshape, order)
    198 @array_function_dispatch(_reshape_dispatcher)
    199 def reshape(a, newshape, order='C'):
    200     """
    201     Gives a new shape to an array without changing its data.
    202
   (…)
    296             [5, 6]])
    297     """
--> 298     return _wrapfunc(a, 'reshape', newshape, order=order)

File E:
 ↪\Anaconda\envs\ApplicationDesignBDM1034\lib\site-packages\numpy\core\fromnume ic.
 ↪py:57, in _wrapfunc(obj, method, *args, **kwds)
     54     return _wrapit(obj, method, *args, **kwds)
     56 try:
---> 57     return bound(*args, **kwds)
     58 except TypeError:
     59     # A TypeError occurs if the object does have such a method in its
     60     # class, but its signature is not identical to that of NumPy's. Thi
   (…)
     64     # Call _wrapit from within the except clause to ensure a potential
     65     # exception has a traceback chain.
     66     return _wrapit(obj, method, *args, **kwds)

ValueError: cannot reshape array of size 12 into shape (4,4)
```

[100]:
```
fifth_arr=np.reshape(first_arr,(3,2,2))
print(fifth_arr)
print("Dimensions of fifth_arr is ",fifth_arr.ndim)
```

```
[[[ 1  2]
  [ 3  4]]

 [[ 5  6]
  [ 7  8]]

 [[ 9 10]
```

```
  [11 12]]]
Dimensions of fifth_arr is  3
```

[102]:
```
sixth_arr=np.array([[1,2],[3,4],[5,6]])
sixth_arr
```

[102]:
```
array([[1, 2],
       [3, 4],
       [5, 6]])
```

[103]:
```
seventh_arr_flat=np.reshape(sixth_arr,-1)
seventh_arr_flat
```

[103]:
```
array([1, 2, 3, 4, 5, 6])
```

[104]:
```
eighth_arr_flat=sixth_arr.flatten()
print("eighth_arr_flat:",eighth_arr_flat)
ninth_arr_rav=sixth_arr.ravel()
print("ninth_arr_rav:",ninth_arr_rav)
```

```
eighth_arr_flat: [1 2 3 4 5 6]
ninth_arr_rav: [1 2 3 4 5 6]
```

[106]:
```
eighth_arr_flat[0]=100
```

[108]:
```
ninth_arr_rav[0]=200
```

[109]:
```
print("eighth_arr_flat:",eighth_arr_flat)
print("ninth_arr_rav:",ninth_arr_rav)
print("sixth_arr:",sixth_arr)
```

```
eighth_arr_flat: [100   2   3   4   5   6]
ninth_arr_rav: [200   2   3   4   5   6]
sixth_arr: [[200   2]
 [  3   4]
 [  5   6]]
```

[110]:
```
twodim_arr=np.reshape(np.arange(12),(3,4))
twodim_arr
```

[110]:
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

[111]:
```
twodim_arr[1,1]
```

[111]:
```
5
```

[112]:
```
twodim_arr[1]
```

[112]:
```
array([4, 5, 6, 7])
```

[113]:
```
threedim_arr=np.reshape(np.arange(3*4*5),(3,4,5))
threedim_arr
```

[113]:
```
array([[[ 0,  1,  2,  3,  4],
        [ 5,  6,  7,  8,  9],
        [10, 11, 12, 13, 14],
        [15, 16, 17, 18, 19]],

       [[20, 21, 22, 23, 24],
        [25, 26, 27, 28, 29],
        [30, 31, 32, 33, 34],
        [35, 36, 37, 38, 39]],

       [[40, 41, 42, 43, 44],
        [45, 46, 47, 48, 49],
        [50, 51, 52, 53, 54],
        [55, 56, 57, 58, 59]]])
```

[114]:
```
threedim_arr[0,2,3]
```

[114]:
```
13
```

[115]:
```
threedim_arr[2,-1,-1]
```

[115]:
```
59
```

[116]:
```
onedim_arr=np.arange(10)
onedim_arr
```

[116]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

[117]:
```
onedim_arr[2:6]
```

[117]:
```
array([2, 3, 4, 5])
```

[118]:
```
onedim_arr[:5]
```

[118]:
```
array([0, 1, 2, 3, 4])
```

[119]:
```
onedim_arr[-3:]
```

[119]:
```
array([7, 8, 9])
```

[121]:
```
onedim_arr[::2]
```

[121]:
```
array([0, 2, 4, 6, 8])
```

[131]:
```
twodim_arr
```

[131]:
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

[126]:
```
twodim_arr[1:,1:]
```

[126]:
```
array([[ 5,  6,  7],
       [ 9, 10, 11]])
```

[127]:
```
twodim_arr[1,:]
```

[127]:
```
array([4, 5, 6, 7])
```

[128]:
```
twodim_arr[:,2]
```

[128]:
```
array([ 2,  6, 10])
```

1.3.4 Joining and splitting ararys

[132]:
```
first_arr=np.arange(1,11)
second_arr=np.arange(11,21)
print("first_arr",first_arr)
print("second_arr",second_arr)
```

```
first_arr [ 1  2  3  4  5  6  7  8  9 10]
second_arr [11 12 13 14 15 16 17 18 19 20]
```

[133]:
```
con_arr=np.concatenate((first_arr,second_arr))
con_arr
```

[133]:
```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17,
       18, 19, 20])
```

[134]:
```
third_2darr=np.array([[1,2,3,4,5], [6,7,8,9,10]])
third_2darr
```

[134]:
```
array([[ 1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10]])
```

[135]:
```
fourth_2darr=np.array([[11,12,13,14,15], [16,17,18,19,20]])
fourth_2darr
```

[135]:
```
array([[11, 12, 13, 14, 15],
       [16, 17, 18, 19, 20]])
```

[136]:
```
con2d_arr = np.concatenate((third_2darr,fourth_2darr),axis=1)
con2d_arr
```

[136]:
```
array([[ 1,  2,  3,  4,  5, 11, 12, 13, 14, 15],
       [ 6,  7,  8,  9, 10, 16, 17, 18, 19, 20]])
```

[137]:
```
st_arr = np.stack((first_arr,second_arr))
st_arr
```

[137]:
```
array([[ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10],
       [11, 12, 13, 14, 15, 16, 17, 18, 19, 20]])
```

[138]:
```
hst_arr=np.hstack((first_arr,second_arr))
hst_arr
```

[138]:
```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17,
       18, 19, 20])
```

[139]:
```
vst_arr=np.vstack((first_arr,second_arr))
vst_arr
```

[139]:
```
array([[ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10],
       [11, 12, 13, 14, 15, 16, 17, 18, 19, 20]])
```

[141]:
```
fifth_arr=np.arange(1,13)
fifth_arr
```

[141]:
```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12])
```

[142]:
```
sp_arr=np.array_split(fifth_arr,4)
sp_arr
```

[142]:
```
[array([1, 2, 3]), array([4, 5, 6]), array([7, 8, 9]), array([10, 11, 12])]
```

[143]:
```
print(sp_arr[1])
```

```
[4 5 6]
```

[144]:
```
sp_arr=np.array_split(fifth_arr,8)
sp_arr
```

[144]:
```
[array([1, 2]),
 array([3, 4]),
 array([5, 6]),
 array([7, 8]),
 array([9]),
 array([10]),
 array([11]),
 array([12])]
```

[147]:
```
np.hsplit(third_2darr,5)
```

[147]:
```
[array([[1],
        [6]]),
 array([[2],
        [7]]),
 array([[3],
        [8]]),
 array([[4],
        [9]]),
 array([[ 5],
        [10]])]
```

[148]:
```
vs_arr=np.vsplit(third_2darr,2)
vs_arr
```

[148]:
```
[array([[1, 2, 3, 4, 5]]), array([[ 6,  7,  8,  9, 10]])]
```

1.4 Chapter 4 : Functions and Operations

1.4.1 Arithmetic operations

[149]:
```
a=np.arange(1,11)
b=np.arange(21,31)
print("a",a)
print("b",b)
```

```
a [ 1  2  3  4  5  6  7  8  9 10]
b [21 22 23 24 25 26 27 28 29 30]
```

[150]:
```
a+b
```

[150]:
```
array([22, 24, 26, 28, 30, 32, 34, 36, 38, 40])
```

[151]:
```
b-a
```

[151]:
```
array([20, 20, 20, 20, 20, 20, 20, 20, 20, 20])
```

[152]:
```
a*b
```

[152]:
```
array([ 21,  44,  69,  96, 125, 156, 189, 224, 261, 300])
```

[153]:
```
b/a
```

[153]:
```
array([21.        , 11.        ,  7.66666667,  6.        ,  5.        ,
        4.33333333,  3.85714286,  3.5       ,  3.22222222,  3.        ])
```

[154]:
```
a**b
```

[154]:
```
array([          1,     4194304,  -346101685,           0,   167814181,
       -1543503872,    17998615,           0, -1635065239,  1073741824])
```

[155]:
```
a*2
```

[155]:
```
array([ 2,  4,  6,  8, 10, 12, 14, 16, 18, 20])
```

[156]:
```
np.add(a,b)
```

[156]:
```
array([22, 24, 26, 28, 30, 32, 34, 36, 38, 40])
```

[157]:
```
np.subtract(b,a)
```

[157]:
```
array([20, 20, 20, 20, 20, 20, 20, 20, 20, 20])
```

[158]:
```
np.multiply(a,b)
```

[158]:
```
array([ 21,  44,  69,  96, 125, 156, 189, 224, 261, 300])
```

[159]:
```
np.divide(b,a)
```

[159]:
```
array([21.        , 11.        ,  7.66666667,  6.        ,  5.        ,
        4.33333333,  3.85714286,  3.5       ,  3.22222222,  3.        ])
```

[160]:
```
np.mod(b,a)
```

[160]:
```
array([0, 0, 2, 0, 0, 2, 6, 4, 2, 0])
```

[161]:
```
np.power(a,b)
```

[161]:
```
array([          1,     4194304,  -346101685,           0,   167814181,
       -1543503872,    17998615,           0, -1635065239,  1073741824])
```

[162]:
```
np.sqrt(a)
```

[162]:
```
array([1.        , 1.41421356, 1.73205081, 2.        , 2.23606798,
       2.44948974, 2.64575131, 2.82842712, 3.        , 3.16227766])
```

1.4.2 Broadcasting

[163]:
```
a=np.arange(1,10).reshape(3,3)
a
```

[163]:
```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

[164]:
```
b=np.arange(1,4)
b
```

[164]:
```
array([1, 2, 3])
```

[165]:
```
a+b
```

[165]:
```
array([[ 2,  4,  6],
       [ 5,  7,  9],
       [ 8, 10, 12]])
```

[166]:
```
c=np.arange(1,3)
c
```

[166]:
```
array([1, 2])
```

[167]:
```
a+c
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Input In [167], in <module>
----> 1 a+c

ValueError: operands could not be broadcast together with shapes (3,3) (2,)
```

[168]:
```
d=np.arange(24).reshape(2,3,4)
d
```

[168]:
```
array([[[ 0,  1,  2,  3],
        [ 4,  5,  6,  7],
        [ 8,  9, 10, 11]],

       [[12, 13, 14, 15],
        [16, 17, 18, 19],
        [20, 21, 22, 23]]])
```

[169]:
```
e=np.arange(4)
e
```

[169]:
```
array([0, 1, 2, 3])
```

[170]:
```
d-e
```

[170]:
```
array([[[ 0,  0,  0,  0],
        [ 4,  4,  4,  4],
        [ 8,  8,  8,  8]],

       [[12, 12, 12, 12],
        [16, 16, 16, 16],
        [20, 20, 20, 20]]])
```

1.4.3 Aggregate functions

[171]:
```
first_arr=np.arange(10,110,10)
first_arr
```

[171]:
```
array([ 10,  20,  30,  40,  50,  60,  70,  80,  90, 100])
```

[172]:
```
second_arr=np.arange(10,100,10).reshape(3,3)
second_arr
```

[172]:
```
array([[10, 20, 30],
       [40, 50, 60],
       [70, 80, 90]])
```

[173]:
```
third_arr=np.arange(10,110,10).reshape(2,5)
third_arr
```

[173]:
```
array([[ 10,  20,  30,  40,  50],
       [ 60,  70,  80,  90, 100]])
```

[174]:
```
first_arr.sum()
```

[174]:
```
550
```

[175]:
```
second_arr.sum()
```

[175]:
```
450
```

[177]:
```
third_arr.sum()
```

[177]:
```
550
```

[178]:
```
second_arr.sum(axis=0)
```

[178]:
```
array([120, 150, 180])
```

[179]:
```
second_arr.sum(axis=1)
```

[179]:
```
array([ 60, 150, 240])
```

[180]:
```
first_arr.prod()
```

[180]:
```
1704722432
```

[181]:
```
second_arr.prod()
```

[181]:
```
-1786839040
```

[182]:
```
third_arr.prod()
```

[182]:
```
1704722432
```

[183]:
```
third_arr.prod(axis=0)
```

[183]:
```
array([ 600, 1400, 2400, 3600, 5000])
```

[184]:
```
np.average(first_arr)
```

[184]:
```
55.0
```

[185]:
```
np.average(second_arr)
```

[185]:
```
50.0
```

[186]:
```
np.average(third_arr)
```

[186]:
```
55.0
```

[187]:
```
np.min(first_arr)
```

[187]:
```
10
```

[188]:
```
np.max(first_arr)
```

[188]:
```
100
```

[189]:
```
np.mean(first_arr)
```

[189]:
```
55.0
```

[190]:
```
np.std(first_arr)
```

[190]:
```
28.722813232690143
```

1.4.4 How to get unique items and counts

[191]:
```
first_arr=np.array([1,2,3,4,5,6,1,2,7,2,1,10,7,8])
```

[192]:
```
np.unique(first_arr)
```

[192]:
```
array([ 1,  2,  3,  4,  5,  6,  7,  8, 10])
```

[193]:
```
second_arr=np.array([[1, 1, 2,1] ,[ 3, 1, 2,1] , [1, 1, 2, 1], [ 7, 1, 1, 1]])
second_arr
```

[193]:
```
array([[1, 1, 2, 1],
       [3, 1, 2, 1],
       [1, 1, 2, 1],
       [7, 1, 1, 1]])
```

[194]:
```
np.unique(second_arr)
```

[194]:
```
array([1, 2, 3, 7])
```

[195]:
```
np.unique(second_arr,axis=0)
```

[195]:
```
array([[1, 1, 2, 1],
       [3, 1, 2, 1],
       [7, 1, 1, 1]])
```

[196]:
```
np.unique(second_arr,axis=1)
```

[196]:
```
array([[1, 1, 2],
       [1, 3, 2],
       [1, 1, 2],
       [1, 7, 1]])
```

[197]:
```
np.unique(first_arr,return_index= True)
```

[197]:
```
(array([ 1,  2,  3,  4,  5,  6,  7,  8, 10]),
 array([ 0,  1,  2,  3,  4,  5,  8, 13, 11], dtype=int64))
```

[198]:
```
np.unique(second_arr,return_counts=True)
```

[198]:
```
(array([1, 2, 3, 7]), array([11,  3,  1,  1], dtype=int64))
```

1.4.5 Transpose like operations

[199]:
```
first_2dimarr=np.arange(12).reshape((3,4))
first_2dimarr
```

[199]:
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

[200]:
```
np.transpose(first_2dimarr)
```

[200]:
```
array([[ 0,  4,  8],
       [ 1,  5,  9],
       [ 2,  6, 10],
       [ 3,  7, 11]])
```

[201]:
```
second_2dimarr=np.arange(6).reshape(3,2)
second_2dimarr
```

[201]:
```
array([[0, 1],
       [2, 3],
       [4, 5]])
```

[202]:
```
np.transpose(second_2dimarr,(1,0))
```

[202]:
```
array([[0, 2, 4],
       [1, 3, 5]])
```

[203]:
```
first_3dimarr=np.arange(24).reshape(2,3,4)
first_3dimarr
```

[203]:
```
array([[[ 0,  1,  2,  3],
        [ 4,  5,  6,  7],
        [ 8,  9, 10, 11]],

       [[12, 13, 14, 15],
        [16, 17, 18, 19],
        [20, 21, 22, 23]]])
```

[204]:
```
np.moveaxis(first_3dimarr,0,-1)
```

[204]:
```
array([[[ 0, 12],
        [ 1, 13],
        [ 2, 14],
        [ 3, 15]],

       [[ 4, 16],
        [ 5, 17],
        [ 6, 18],
        [ 7, 19]],

       [[ 8, 20],
        [ 9, 21],
        [10, 22],
        [11, 23]]])
```

[205]:
```
np.swapaxes(first_3dimarr,0,2)
```

[205]:
```
array([[[ 0, 12],
        [ 4, 16],
        [ 8, 20]],

       [[ 1, 13],
        [ 5, 17],
        [ 9, 21]],

       [[ 2, 14],
        [ 6, 18],
        [10, 22]],

       [[ 3, 15],
```

```
        [ 7, 19],
        [11, 23]]])
```

1.4.6 Reversing an array

[206]:
```
arr_1dim=[10,1,9,2,8,3,7,4,6,5]
arr_1dim
```

[206]:
```
[10, 1, 9, 2, 8, 3, 7, 4, 6, 5]
```

[207]:
```
arr_1dim[::-1]
```

[207]:
```
[5, 6, 4, 7, 3, 8, 2, 9, 1, 10]
```

[208]:
```
np.flip(arr_1dim)
```

[208]:
```
array([ 5,  6,  4,  7,  3,  8,  2,  9,  1, 10])
```

[209]:
```
arr_2dim=np.arange(9).reshape(3,3)
arr_2dim
```

[209]:
```
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])
```

[210]:
```
np.flip(arr_2dim)
```

[210]:
```
array([[8, 7, 6],
       [5, 4, 3],
       [2, 1, 0]])
```

[211]:
```
np.flip(arr_2dim,1)
```

[211]:
```
array([[2, 1, 0],
       [5, 4, 3],
       [8, 7, 6]])
```

[212]:
```
arr_3dim=np.arange(24).reshape(2,3,4)
arr_3dim
```

[212]:
```
array([[[ 0,  1,  2,  3],
        [ 4,  5,  6,  7],
        [ 8,  9, 10, 11]],

       [[12, 13, 14, 15],
        [16, 17, 18, 19],
        [20, 21, 22, 23]]])
```

[213]:
```
np.flip(arr_3dim,1)
```

[213]:
```
array([[[ 8,  9, 10, 11],
        [ 4,  5,  6,  7],
        [ 0,  1,  2,  3]],

       [[20, 21, 22, 23],
        [16, 17, 18, 19],
        [12, 13, 14, 15]]])
```

[214]:
```
np.flip(arr_3dim,2)
```

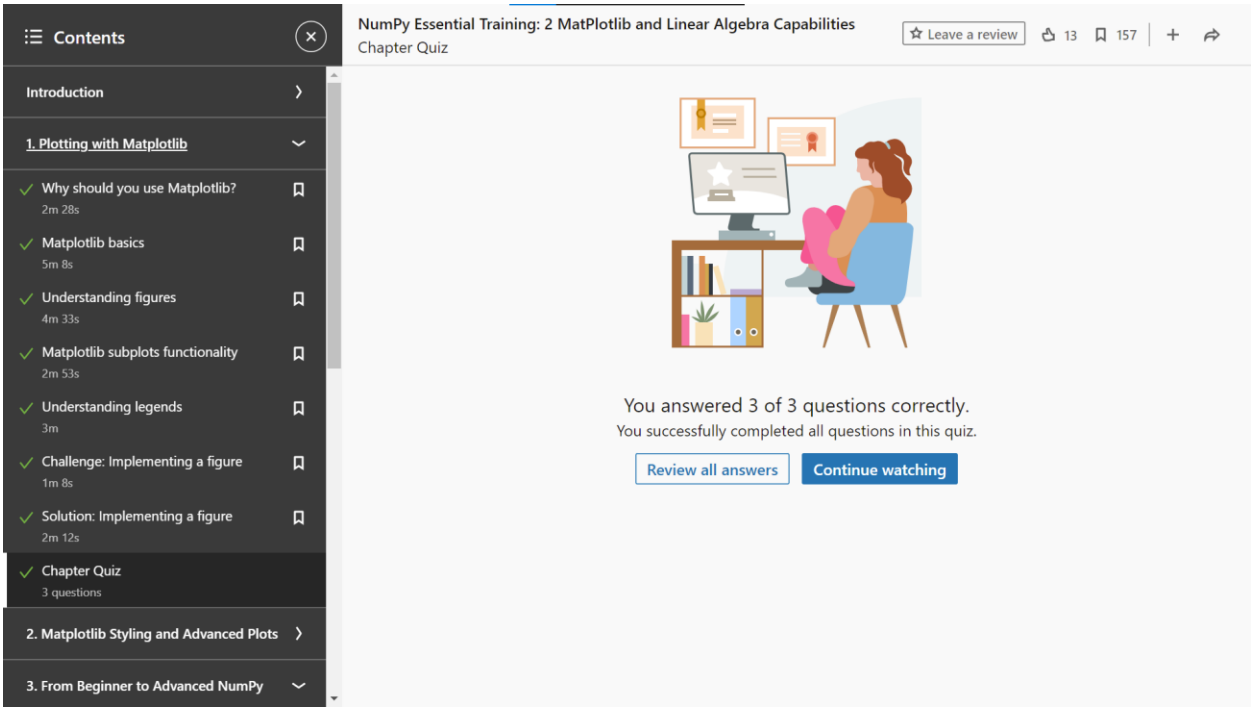
[214]:
```
array([[[ 3,  2,  1,  0],
        [ 7,  6,  5,  4],
        [11, 10,  9,  8]],

       [[15, 14, 13, 12],
        [19, 18, 17, 16],
        [23, 22, 21, 20]]])
```

[ ]:
```

```

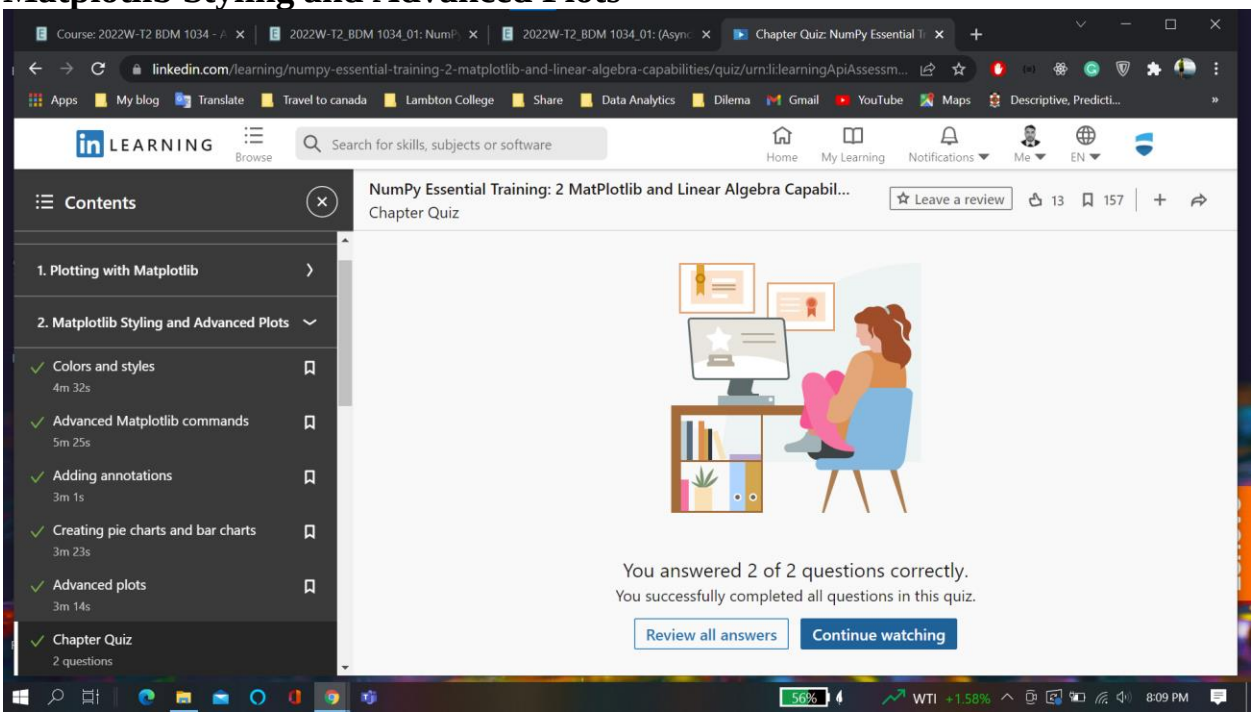
BDM 1034 - Application Design for Big Data
Week9
Submitted by: Aadarsha Chapagain
Student ID:C0825975
Submitted to: Prof. Teresa Zhu

Here I have attached the certificate I achieved from Linkedin learning for Course "***NumPy Essential Training: 2 Foundations of NumPy***" along with the Screenshot of score I achieved

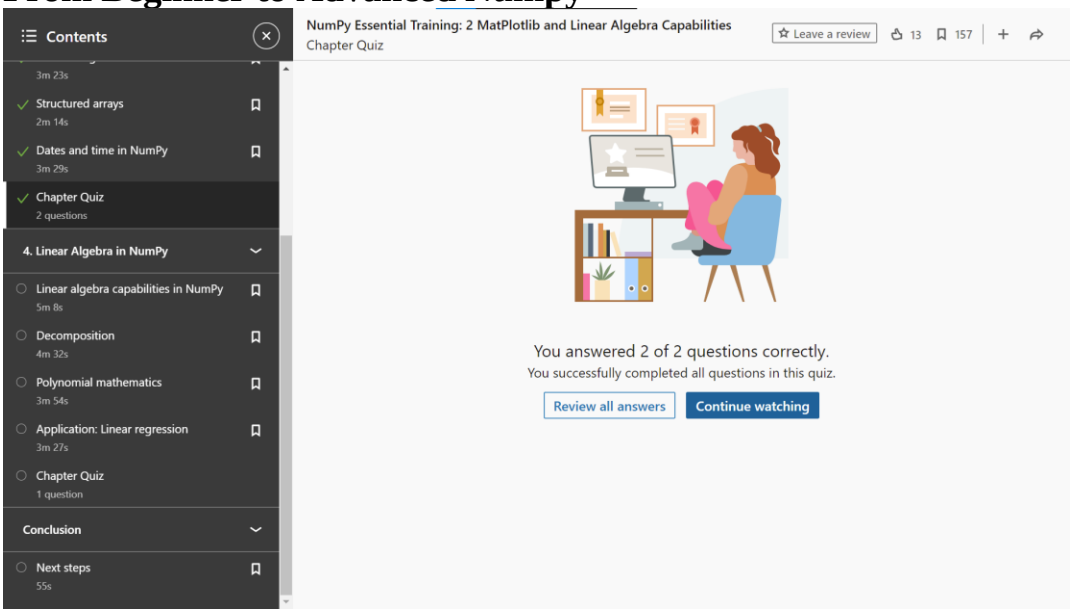
Chapter 1: Plotting with Matplotlib
Overview



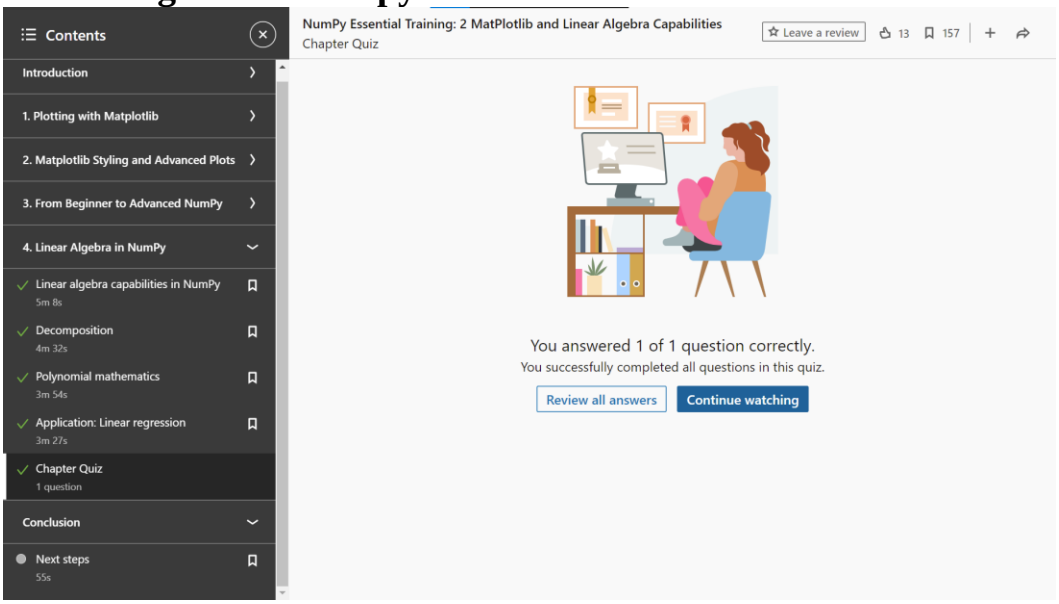
Chapter 2
Matplotlib Styling and Advanced Plots



Chapter 3:
From Beginner to Advanced Numpy



Chapter 4:
Linear Algebra in Numpy



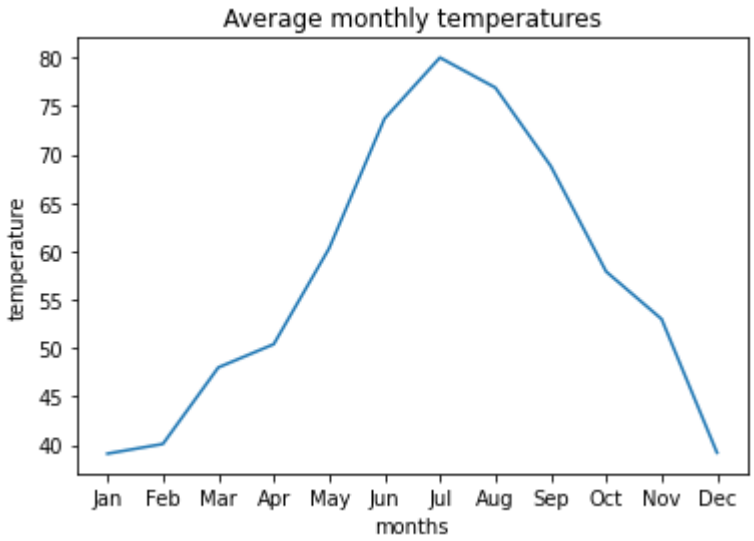
NumPy Essential Training: 2 MatPlotlib and Linear Algebra Capabilities

Chapter 1: Plotting with Matplotlib

In [1]:
```
%matplotlib notebook
%matplotlib inline
```

In [2]:
```
import numpy as np
import matplotlib.pyplot as plt
```

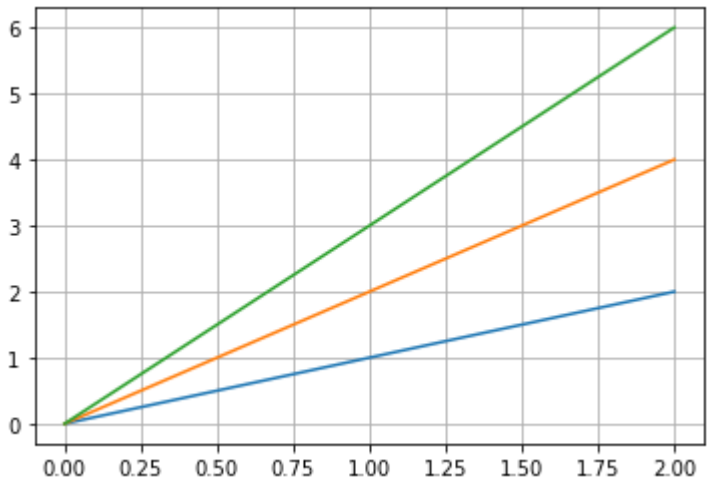
In [3]:
```
average_monthly_temperatures = [39.1, 40.1, 48.0, 50.4, 60.3, 73.7, 80.0, 76.9, 68.8, 5
months=['Jan','Feb','Mar','Apr','May','Jun','Jul','Aug','Sep','Oct','Nov','Dec']
fig = plt.figure()
plt.plot(months,average_monthly_temperatures)
plt.title("Average monthly temperatures")
plt.xlabel("months")
plt.ylabel("temperature")
plt.show()
```



In [5]:
```
fig.savefig('average_monthly_temperatures.png')
```

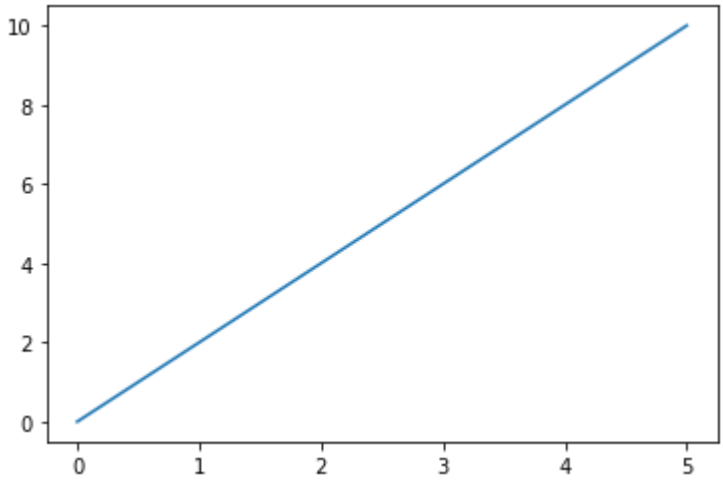
In [6]:
```
fig.savefig('average_monthly_temperatures.pdf')
```

In [8]:
```
x = np.arange(3)
plt.plot(x,x)
plt.plot(x,2*x)
plt.plot(x,3*x)
plt.grid(True)
plt.show()
```

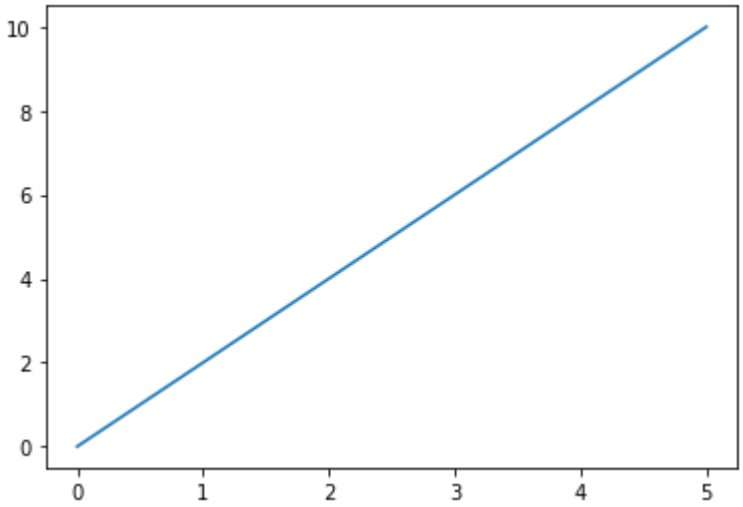


In [9]:
```
x = np.linspace(0,5,5)
y=2*x
plt.plot(x,y)
#plt.show()
```

Out[9]:
```
[<matplotlib.lines.Line2D at 0x2077c9c8610>]
```

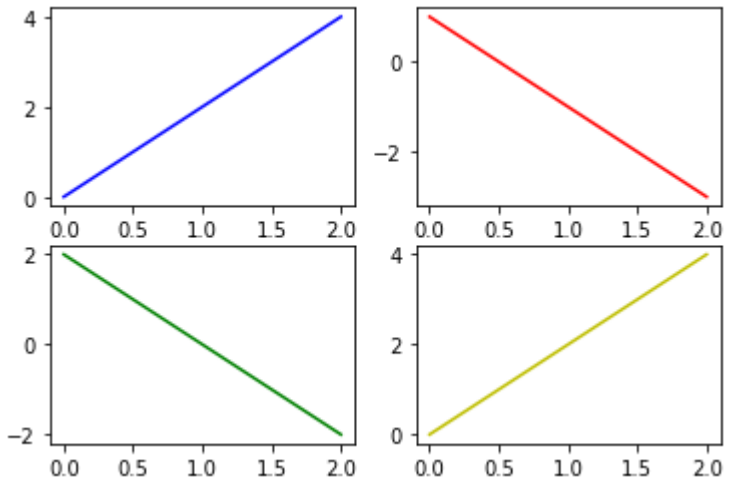


In [10]:
```
fig = plt.figure()
axes = fig.add_axes([0.1,0.1,0.8,0.8])
axes.plot(x,y)
plt.show()
```



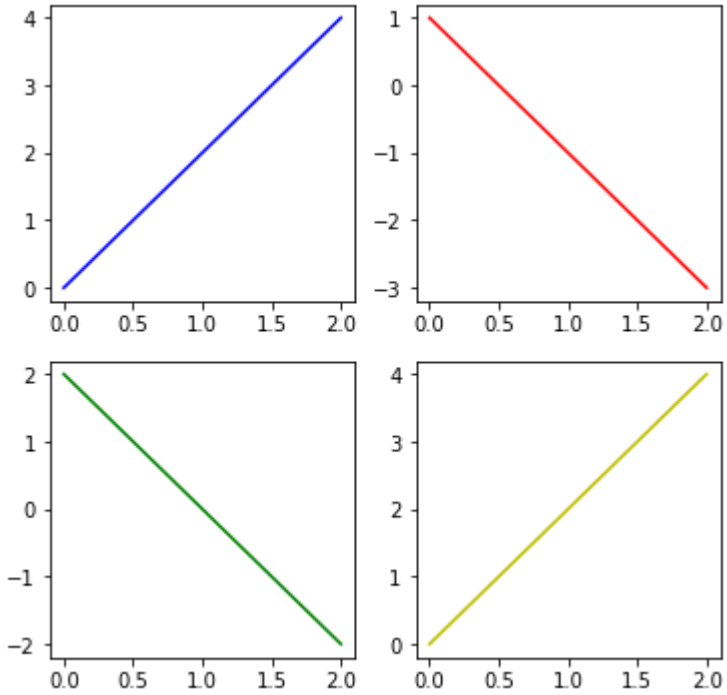
In [11]:

```
fig=plt.figure()
x=np.arange(3)
y=2*x
plt.subplot(2,2,1)
plt.plot(x,y,'b')

plt.subplot(2,2,2)
plt.plot(x,1-y,'r')

plt.subplot(2,2,3)
plt.plot(x,2-y,'g')

plt.subplot(2,2,4)
plt.plot(x,y,'y')

plt.show()
```



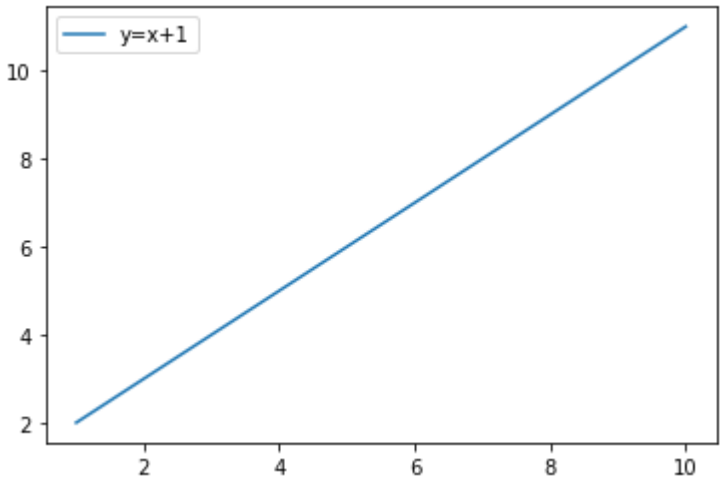
In [12]:

```
fig, axs = plt.subplots(2, 2, figsize=(6,6))
axs[0, 0].plot(x, y, 'b')
axs[0, 1].plot(x, 1-y, 'r')
axs[1, 0].plot(x, 2-y, 'g')
axs[1, 1].plot(x, y, 'y')
plt.show()
```



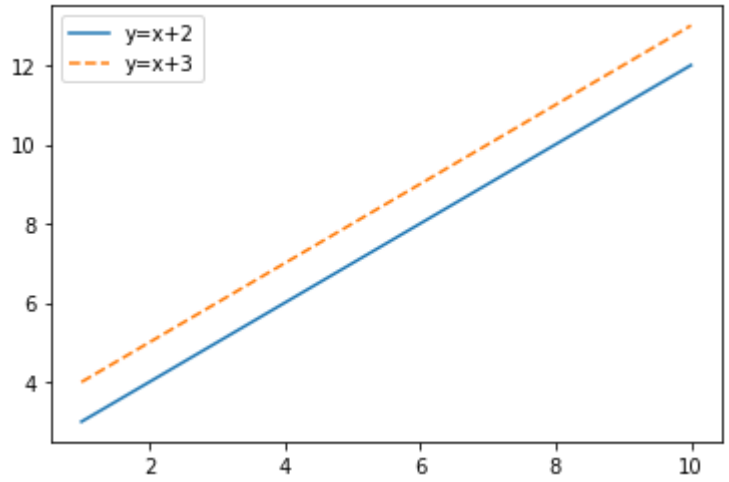
In [13]:
```
x = np.linspace(1,10)
first_line = plt.plot(x, x+1, label= 'y=x+1')
plt.legend()
```

Out[13]:
```
<matplotlib.legend.Legend at 0x2077c8ff4c0>
```

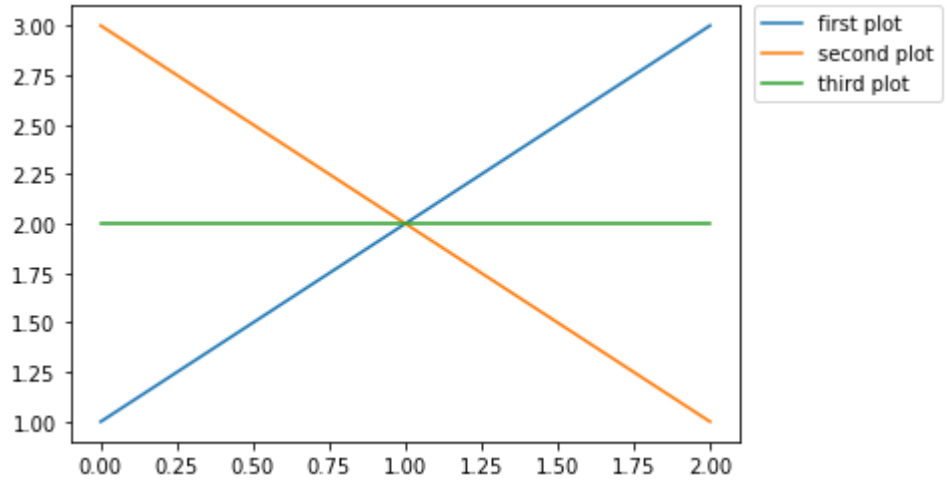


In [14]:
```
second_line, = plt.plot(x,x+2,linestyle='solid')
second_line.set_label('y=x+2')
third_line, = plt.plot(x,x+3,linestyle='dashed')
third_line.set_label('y=x+3')
plt.legend()
```

Out[14]:
```
<matplotlib.legend.Legend at 0x2077cb41100>
```



In [15]:
```
first_plot,=plt.plot([1,2,3],label='first plot')
second_plot,=plt.plot([3,2,1],label='second plot')
third_plot,=plt.plot([2,2,2],label='third plot')
plt.legend(bbox_to_anchor=(1.02, 1.0), borderaxespad=0);
```

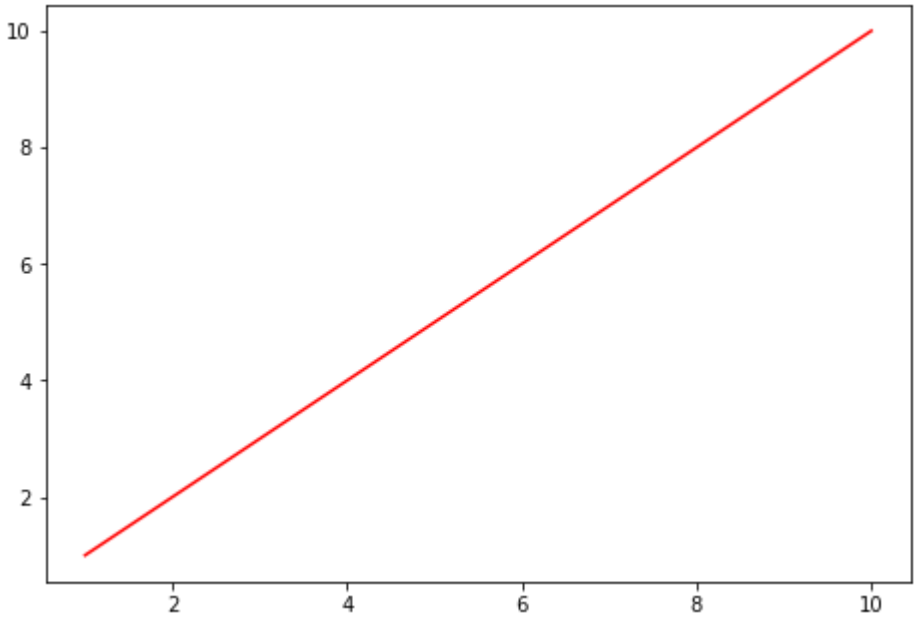


Chapter 2: Matplot lib Styling and Advanced Plots

Color and styles

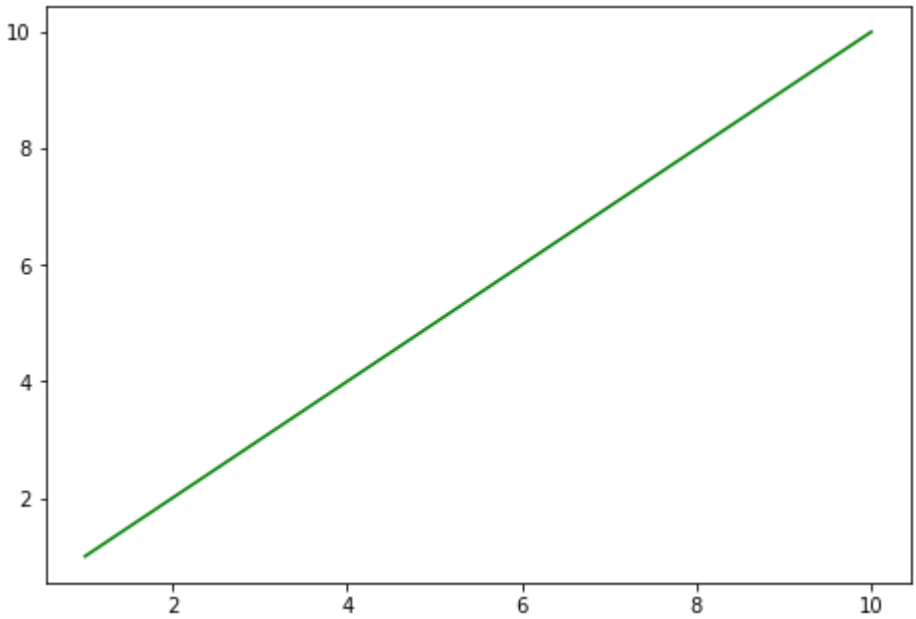
In [16]:
```
first_figure = plt.figure()
x = np.linspace(1, 10)
y = np.linspace(1, 10)
ax=first_figure.add_axes([0,0,1,1])
ax.plot(x,y, color='red')
```

Out[16]:
```
[<matplotlib.lines.Line2D at 0x2077bcad0d0>]
```



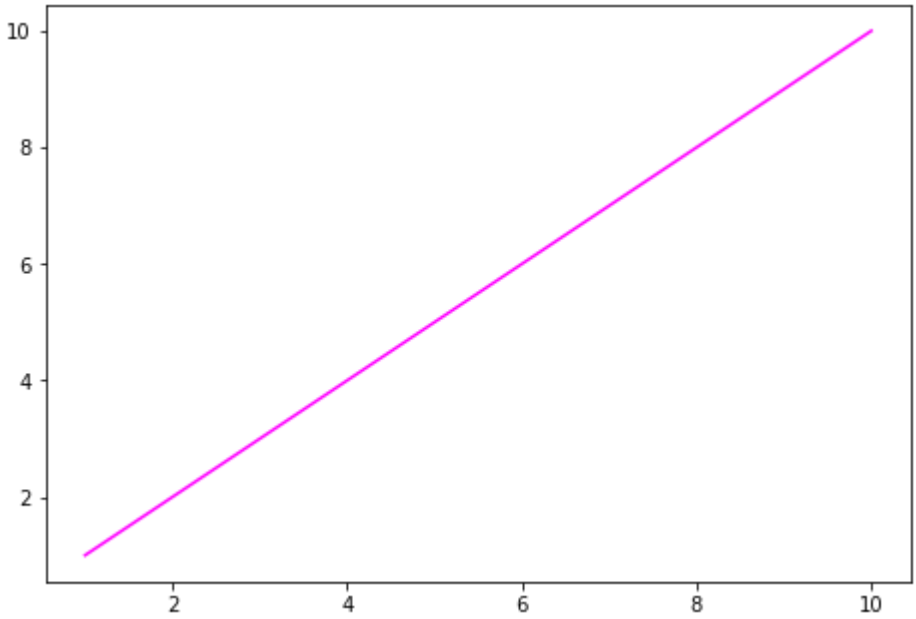
In [17]:
```
second_figure = plt.figure()
ax=second_figure.add_axes([0,0,1,1])
ax.plot(x,y, color='g')
```

Out[17]: [<matplotlib.lines.Line2D at 0x2077c95b850>]



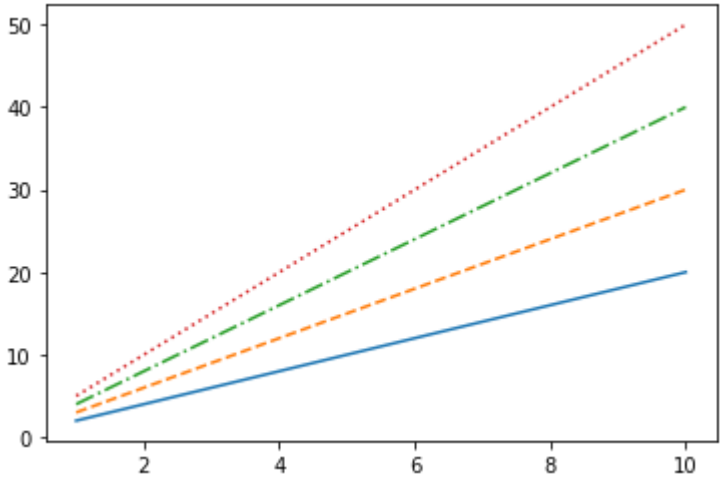
In [18]:
```
third_figure = plt.figure()
ax=third_figure.add_axes([0,0,1,1])
ax.plot(x,y, color='#FF00FF')
```

Out[18]: [<matplotlib.lines.Line2D at 0x2077dd79c40>]



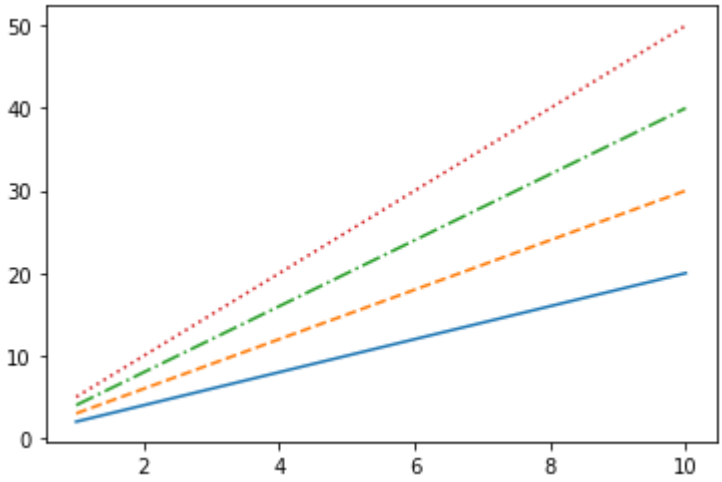
In [19]:
```
plt.plot(x,2*x,linestyle='solid')
plt.plot(x,3*x,linestyle='dashed')
plt.plot(x,4*x,linestyle='dashdot')
plt.plot(x,5*x,linestyle='dotted')
```

Out[19]:
```
[<matplotlib.lines.Line2D at 0x2077ca9fb50>]
```

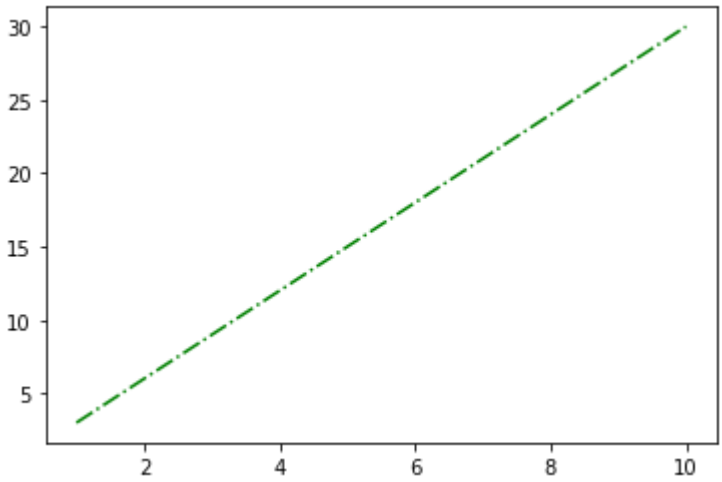


In [20]:
```
plt.plot(x,2*x,linestyle='-')
plt.plot(x,3*x,linestyle='--')
plt.plot(x,4*x,linestyle='-.')
plt.plot(x,5*x,linestyle=':')
```

Out[20]:
```
[<matplotlib.lines.Line2D at 0x2077ddb2e80>]
```

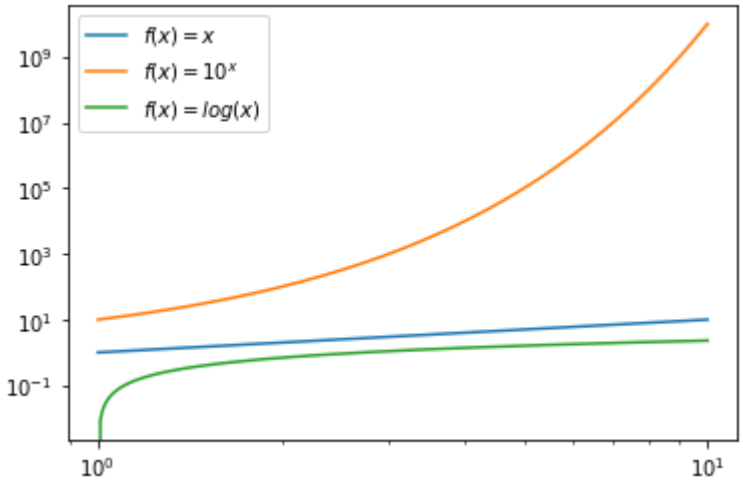


In [21]:
```
plt.plot(x, 3*x ,'-.g');
```



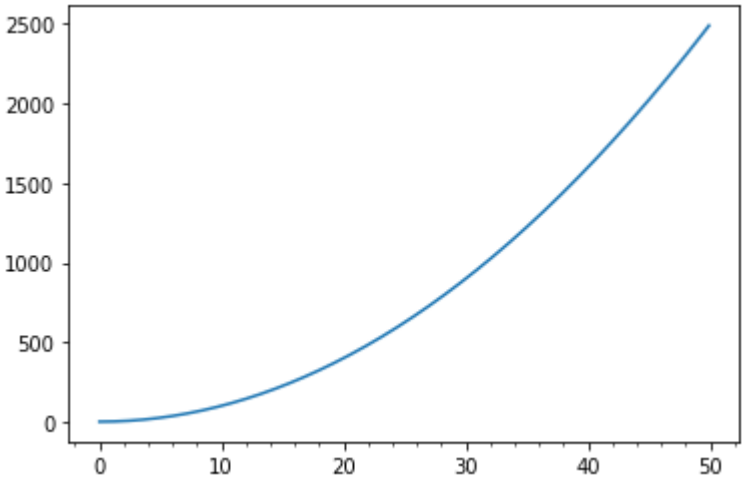
Advanced matplotlib commands

In [22]:
```
x = np.linspace(1, 10, 1024)
plt.xscale('log')
plt.yscale('log')

plt.plot(x, x, label ='$f(x)=x$')
plt.plot(x, 10**x, label ='$f(x)=10^x$')
plt.plot(x, np.log(x),label ='$f(x)=log(x)$')

plt.legend()
plt.show()
```



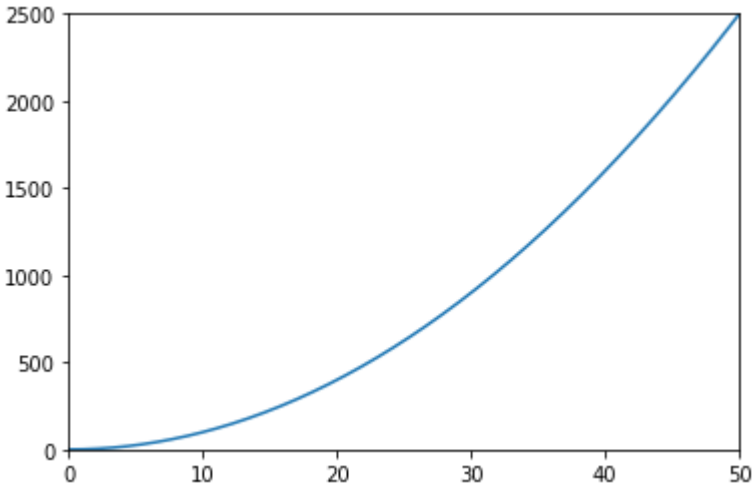
In [23]:

```
from matplotlib.ticker import (MultipleLocator, AutoMinorLocator)

x = np.arange(0.0, 50.0, 0.1)
y = x**2

fig, ax = plt.subplots()
ax.plot(x,y)

ax.xaxis.set_major_locator(MultipleLocator(10))
ax.xaxis.set_major_formatter('{x:.0f}')

ax.xaxis.set_minor_locator(MultipleLocator(2))

plt.show()
```



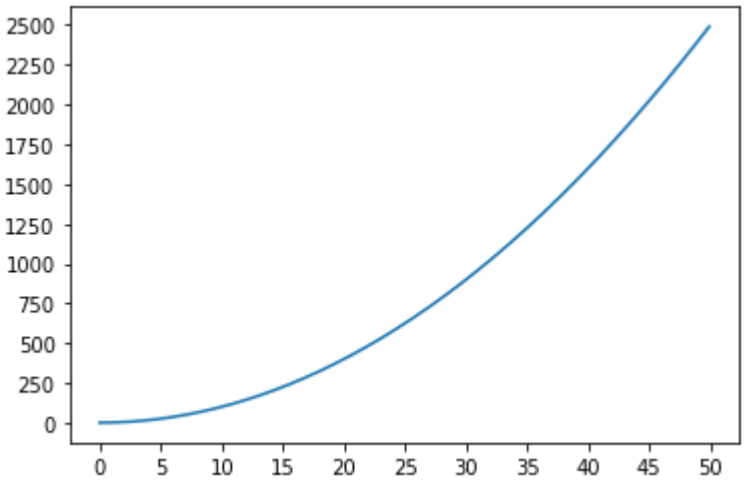
In [24]:

```
x = np.arange(0.0, 50.0, 0.1)
y = x**2
fig, ax = plt.subplots()
ax.plot(x,y)

ax.set_xlim([0, 50])
ax.set_ylim([0, 2500])

plt.show()
```



In [25]:

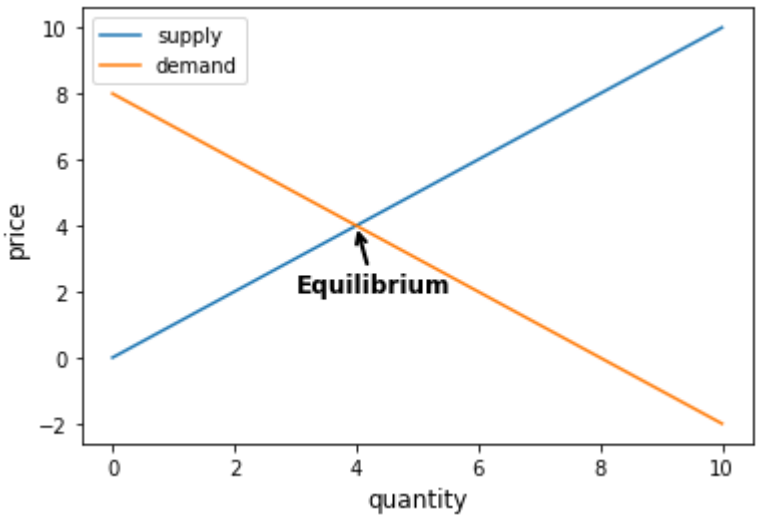
```
x = np.arange(0.0, 50.0, 0.1)
y = x**2

fig, ax = plt.subplots()
ax.plot(x,y)

ax.set_xticks([0,5,10,15,20,25,30,35,40,45,50])
ax.set_yticks([0,250,500,750,1000,1250,1500,1750,2000,2250,2500])

plt.show()
```



In [26]:

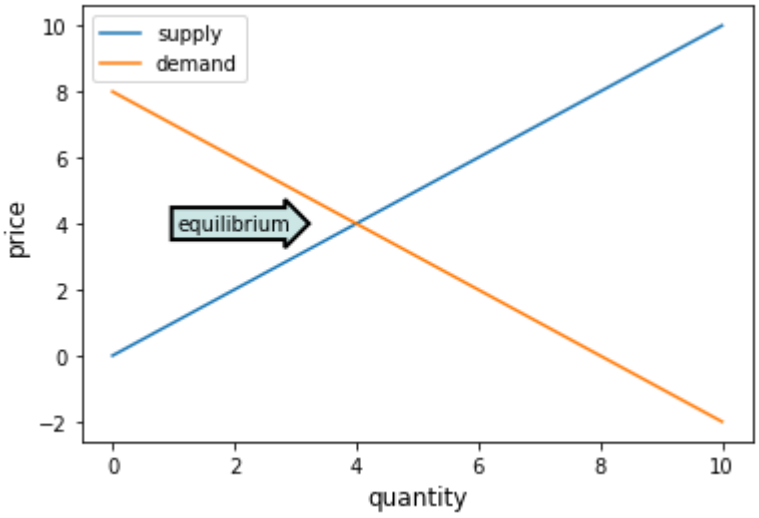
```
x = np.linspace(0, 10)

y1 = x
y2 = 8-x


fig, ax = plt.subplots()
plt.plot(x,y1,label='supply')
plt.plot(x,y2,label='demand')

ax.annotate("Equilibrium", xy=(4,4), xytext=(3,2), \
            fontsize=12, fontweight='semibold',\
            arrowprops=dict(linewidth=2, arrowstyle="->"))
```

```
plt.xlabel('quantity',fontsize=12)
plt.ylabel('price',fontsize=12)


plt.legend()
plt.show()
```



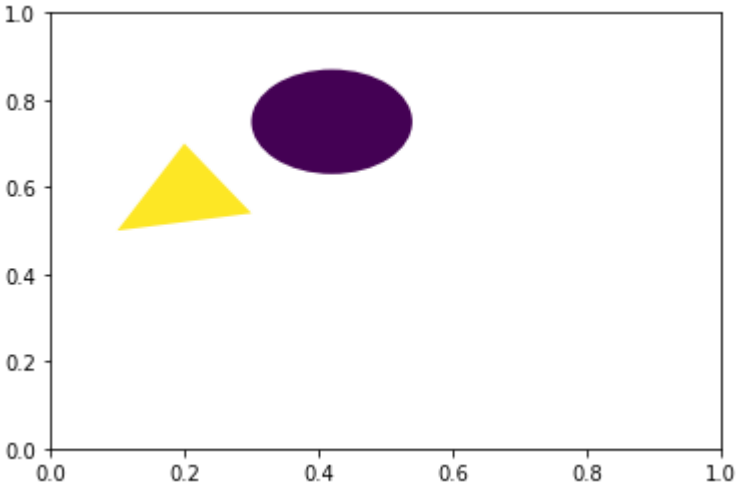
In [27]:

```
x = np.linspace(0, 10)
y1 = x
y2 = 8-x

# Plot the data
fig, ax = plt.subplots()
plt.plot(x,y1,label='supply')
plt.plot(x,y2,label='demand')

# Annotate the equilibrium point with arrow and text
bbox_props = dict(boxstyle="rarrow", fc=(0.8, 0.9, 0.9), lw=2)
t = ax.text(2,4, "equilibrium", ha="center", va="center", rotation=0,
            size=10,bbox=bbox_props)

# Label the axes
plt.xlabel('quantity',fontsize=12)
plt.ylabel('price',fontsize=12)


plt.legend()
plt.show()
```



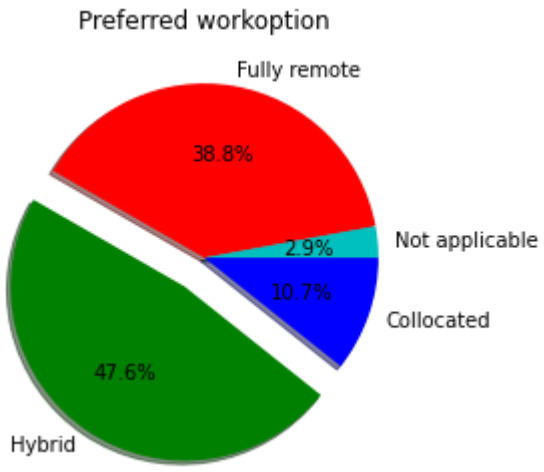
In [28]:

```
from matplotlib.patches import Circle, Polygon
from matplotlib.collections import PatchCollection


fig, ax = plt.subplots()
patches = []

# draw circle and triangle
circle = Circle((.42,.75),0.12)
triangle = Polygon([[.1,.5],[.2,.7],[.3,.54]], True)

patches += [circle,triangle]

# Draw the patches
colors = 100*np.random.rand(len(patches)) # set random colors
p = PatchCollection(patches)
p.set_array(np.array(colors))
ax.add_collection(p)

# Show the figure
plt.show()
```

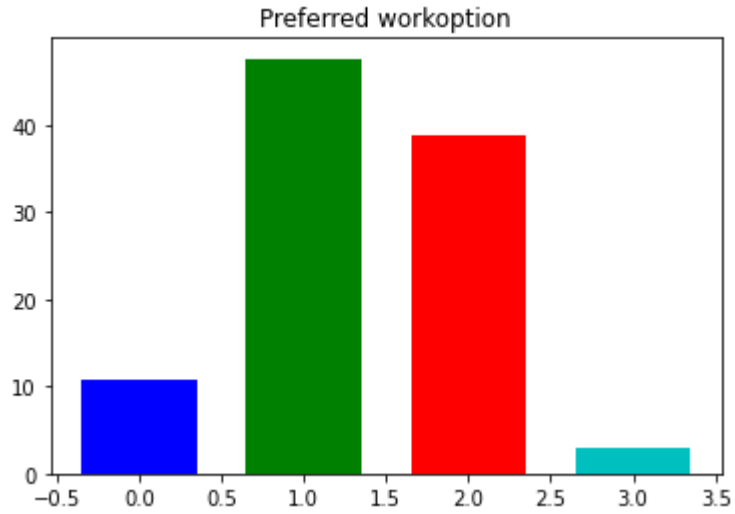


Creating pie charts and bar charts

In [29]:
```
preferred_workoption = [10.7, 47.6, 38.8, 2.9]

colors = ['b', 'g', 'r', 'c']

labels = ['Collocated', 'Hybrid', 'Fully remote', 'Not applicable']

explode = (0, 0.2, 0, 0)


plt.pie(preferred_workoption, colors=colors, labels=labels,

        explode=explode, autopct='%1.1f%%',

        counterclock=False, shadow=True)

plt.title('Preferred workoption')


plt.show()
```



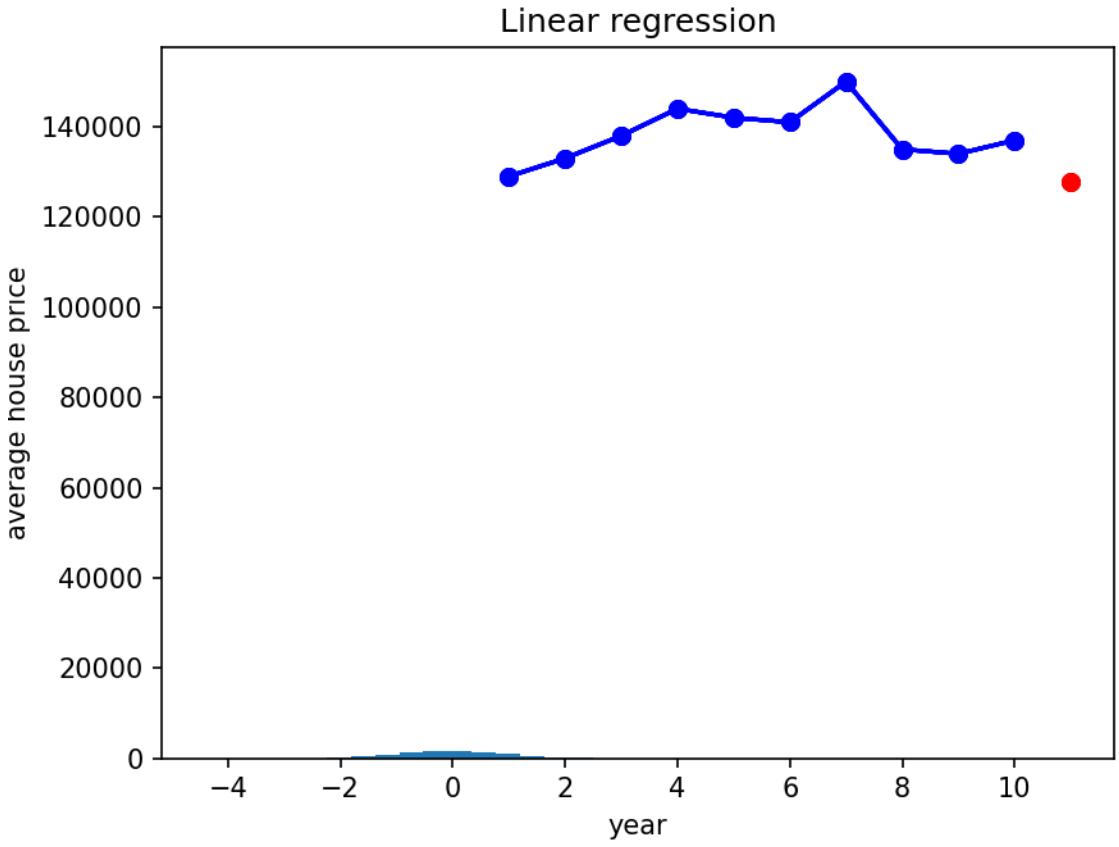
In [30]:
```
preferred_workoption = [10.7, 47.6, 38.8, 2.9]

colors = ['b', 'g', 'r', 'c']

labels = ['Collocated', 'Hybrid', 'Fully remote', 'Not applicable']

widths= [0.7, 0.7, 0.7, 0.7]

plt.bar(range(0, 4), preferred_workoption, width=widths, color=colors, align='center')

plt.title('Preferred workoption')


plt.show()
```



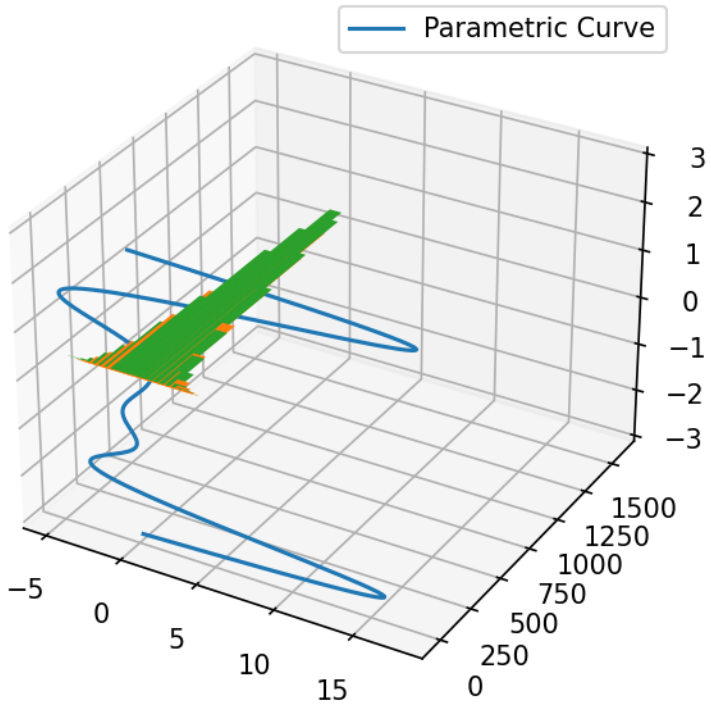
Advanced Plots

In [41]:
```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
%matplotlib notebook
```

In [42]:
```
X = np.random.randn(10000)
plt.hist(X, bins = 20)
plt.show()
```



In [38]:
```
fig = plt.figure()
ax = fig.add_subplot(projection='3d')
theta = np.linspace(-3 * np.pi, 3 * np.pi, 200)
z = np.linspace(-3, 3, 200)
r = z**3 + 1
x = r * np.sin(theta)
y = r * np.cos(theta)

ax.plot(x, y, z, label='Parametric Curve')
ax.legend()
plt.show()
```



Chapter 3: From Beginner to advance Numpy

In [43]:
```
from __future__ import print_function
import numpy as np
```

In [44]:
```
numbers=np.arange(1,11)
numbers
```

Out[44]:
```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

In [45]:
```
np.sin(numbers)
```

Out[45]:
```
array([ 0.84147098,  0.90929743,  0.14112001, -0.7568025 , -0.95892427,
       -0.2794155 ,  0.6569866 ,  0.98935825,  0.41211849, -0.54402111])
```

In [46]:
```
np.log(numbers)
```

Out[46]:
```
array([0.        , 0.69314718, 1.09861229, 1.38629436, 1.60943791,
       1.79175947, 1.94591015, 2.07944154, 2.19722458, 2.30258509])
```

In [47]:
```
# creating numpy array
integers = np.arange(1, 101)
print("integers     :", *integers)

# creating own function
def modulo(val):
  return (val % 10)

# adding into numpy
mod_10=np.frompyfunc(modulo, 1, 1)

# using function over numpy array
mod_integers=mod_10(integers)
print("mod_integers :", *mod_integers)
```

```
integers     : 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 2
8 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57
58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 8
7 88 89 90 91 92 93 94 95 96 97 98 99 100
mod_integers : 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7
8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0
```

Introducing Strides

In [48]:
```
numbers = np.arange(10, dtype = np.int8)
numbers
```

Out[48]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9], dtype=int8)
```

In [49]:
```
numbers.strides
```

Out[49]:
```
(1,)
```

In [50]:
```
numbers.shape = 2,5
numbers
```

Out[50]:
```
array([[0, 1, 2, 3, 4],
       [5, 6, 7, 8, 9]], dtype=int8)
```

In [51]:
```
numbers.strides
```

Out[51]:
```
(5, 1)
```

In [52]:

```
first_array = np.zeros((100000,))
first_array
```

Out[52]:
```
array([0., 0., 0., ..., 0., 0., 0.])
```

In [53]:
```
second_array = np.zeros((100000 * 100, ))[::100]
second_array
```

Out[53]:
```
array([0., 0., 0., ..., 0., 0., 0.])
```

In [54]:
```
first_array.shape
```

Out[54]:
```
(100000,)
```

In [55]:
```
second_array.shape
```

Out[55]:
```
(100000,)
```

In [56]:
```
first_array.strides
```

Out[56]:
```
(8,)
```

In [57]:
```
second_array.strides
```

Out[57]:
```
(800,)
```

In [58]:
```
%timeit first_array.sum()
```

```
56.9 µs ± 5.16 µs per loop (mean ± std. dev. of 7 runs, 10,000 loops each)
```

In [59]:
```
%timeit second_array.sum()
```

```
534 µs ± 44.3 µs per loop (mean ± std. dev. of 7 runs, 1,000 loops each)
```

Structures Arrays

In [61]:
```
student_records = np.array([('Lazaro','Oneal', '0526993', 2009, 2.33), ('Dorie','Salina
       dtype=[('name', (np.str_, 10)),('surname', (np.str_, 10)), ('id', (np.str_,7)),(
student_records
```

Out[61]:
```
array([('Lazaro', 'Oneal', '0526993', 2009, 2.33),
       ('Dorie', 'Salinas', '0710325', 2006, 2.26),
       ('Mathilde', 'Hooper', '0496813', 2000, 2.56),
       ('Nell', 'Gomez', '0740631', 2003, 2.22),
       ('Lachelle', 'Jordan', '0490888', 2003, 2.13),
       ('Claud', 'Waller', '0922492', 2004, 3.6 ),
       ('Bob', 'Steele', '0264843', 2002, 2.79),
       ('Zelma', 'Welch', '0885463', 2007, 3.69)],
```

```
      dtype=[('name', '<U10'), ('surname', '<U10'), ('id', '<U7'), ('graduation_year',
'<i4'), ('gpa', '<f8')])
```

In [62]:
```
student_records[['id','graduation_year']]
```

Out[62]:
```
array([('0526993', 2009), ('0710325', 2006), ('0496813', 2000),
       ('0740631', 2003), ('0490888', 2003), ('0922492', 2004),
       ('0264843', 2002), ('0885463', 2007)],
      dtype={'names': ['id', 'graduation_year'], 'formats': ['<U7', '<i4'], 'offsets':
[80, 108], 'itemsize': 120})
```

In [63]:
```
students_sorted_by_surname = np.sort(student_records, order='surname')
print('Students sorted according to the surname :\n', students_sorted_by_surname)
```

```
Students sorted according to the surname :
 [('Nell', 'Gomez', '0740631', 2003, 2.22)
 ('Mathilde', 'Hooper', '0496813', 2000, 2.56)
 ('Lachelle', 'Jordan', '0490888', 2003, 2.13)
 ('Lazaro', 'Oneal', '0526993', 2009, 2.33)
 ('Dorie', 'Salinas', '0710325', 2006, 2.26)
 ('Bob', 'Steele', '0264843', 2002, 2.79)
 ('Claud', 'Waller', '0922492', 2004, 3.6 )
 ('Zelma', 'Welch', '0885463', 2007, 3.69)]
```

In [64]:
```
students_sorted_by_grad_year = np.sort(student_records, order='graduation_year')
print('Students sorted according to the graduation year :\n', students_sorted_by_grad_y
```

```
Students sorted according to the graduation year :
 [('Mathilde', 'Hooper', '0496813', 2000, 2.56)
 ('Bob', 'Steele', '0264843', 2002, 2.79)
 ('Lachelle', 'Jordan', '0490888', 2003, 2.13)
 ('Nell', 'Gomez', '0740631', 2003, 2.22)
 ('Claud', 'Waller', '0922492', 2004, 3.6 )
 ('Dorie', 'Salinas', '0710325', 2006, 2.26)
 ('Zelma', 'Welch', '0885463', 2007, 3.69)
 ('Lazaro', 'Oneal', '0526993', 2009, 2.33)]
```

Date and time in Numpy

In [65]:
```
np.datetime64('2022-03-01')
```

Out[65]:
```
numpy.datetime64('2022-03-01')
```

In [ ]:
```
np.datetime64('2022-03')
```

In [66]:
```
print('Number of weekdays in 2022:')
print(np.busday_count('2022','2023'))
```

```
Number of weekdays in 2022:
260
```

In [67]:
```
print('Number of weekdays in June 2022:')
np.busday_count('2022-06', '2022-07')
```

```
Number of weekdays in June 2022:
22
```

Out[67]:

In [68]:
```
np.is_busday(np.datetime64('2022-06-05'))
```

Out[68]:
```
False
```

Chapter 4 : Linear Algebra in Numpy

Linear algebra capabilities in NumPy

In [69]:
```
first_array = np.arange(16).reshape(4,4)
first_array
```

Out[69]:
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
       [12, 13, 14, 15]])
```

In [70]:
```
first_matrix = np.matrix(first_array)
first_matrix
```

Out[70]:
```
matrix([[ 0,  1,  2,  3],
        [ 4,  5,  6,  7],
        [ 8,  9, 10, 11],
        [12, 13, 14, 15]])
```

In [71]:
```
second_matrix = np.matrix(np.identity(4))
second_matrix
```

Out[71]:
```
matrix([[1., 0., 0., 0.],
        [0., 1., 0., 0.],
        [0., 0., 1., 0.],
        [0., 0., 0., 1.]])
```

In [72]:
```
matrix_a=np.random.randint(5,size=(2,3))
matrix_a
```

Out[72]:
```
array([[4, 0, 4],
       [0, 4, 3]])
```

In [73]:
```
matrix_b=np.random.randint(5,size=(3,2))
matrix_b
```

Out[73]:
```
array([[1, 4],
       [0, 0],
       [1, 3]])
```

In [74]:
```
np.matmul(matrix_a,matrix_b)
```

Out[74]:
```
array([[ 8, 28],
       [ 3,  9]])
```

In [75]:
```
matrix_c=np.matrix("0 1 2;1 0 3;4 -3 8")
matrix_c
```

Out[75]:
```
matrix([[ 0,  1,  2],
        [ 1,  0,  3],
        [ 4, -3,  8]])
```

In [76]:
```
inverse = np.linalg.inv(matrix_c)
inverse
```

Out[76]:
```
matrix([[-4.5,  7. , -1.5],
        [-2. ,  4. , -1. ],
        [ 1.5, -2. ,  0.5]])
```

In [78]:
```
print(matrix_c*inverse)
```

```
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
```

In [79]:
```
A =np.mat("1 -2 1;0 2 -8;-4 5 9")
A
```

Out[79]:
```
matrix([[ 1, -2,  1],
        [ 0,  2, -8],
        [-4,  5,  9]])
```

In [80]:
```
b = np.array([0, 16, -18])
b
```

Out[80]:
```
array([  0,  16, -18])
```

In [81]:
```
x = np.linalg.solve(A, b)
print("Solution", x)
```

```
Solution [58. 32.  6.]
```

Decomposition

In [82]:
```
first_matrix=np.matrix([[4,8],[10,14]])
print("Matrix:\n",first_matrix)
```

```
Matrix:
 [[ 4  8]
 [10 14]]
```

In [83]:
```
eigenvalues, eigenvectors = np.linalg.eig(first_matrix)
print("Eigenvalues:", eigenvalues)
print("Eigenvectors:", eigenvectors)
```

```
Eigenvalues: [-1.24695077 19.24695077]
Eigenvectors: [[-0.83619408 -0.46462222]
```

```
 [ 0.54843365 -0.885509  ]]
```

In [84]:
```
eigenvalues= np.linalg.eigvals(first_matrix)
print("Eigenvalues:", eigenvalues)
```

```
Eigenvalues: [-1.24695077 19.24695077]
```

In [85]:
```
A = np.mat("3 1 4;1 5 9;2 6 5")
print("A\n", A)

U, Sigma, V = np.linalg.svd(A, full_matrices=False)

print("U: ",U)
print("Sigma : ",Sigma)
print("V : ", V)
```

```
A
 [[3 1 4]
 [1 5 9]
 [2 6 5]]
U:  [[-0.32463251  0.79898436  0.50619929]
 [-0.75307473  0.1054674  -0.64942672]
 [-0.57226932 -0.59203093  0.56745679]]
Sigma :  [13.58235799  2.84547726  2.32869289]
V :  [[-0.21141476 -0.55392606 -0.80527617]
 [ 0.46331722 -0.78224635  0.41644663]
 [ 0.86060499  0.28505536 -0.42202191]]
```

In [86]:
```
print("Product\n", U * np.diag(Sigma) * V)
```

```
Product
 [[3. 1. 4.]
 [1. 5. 9.]
 [2. 6. 5.]]
```

M=Q*R

In [87]:
```
A
```

Out[87]:
```
matrix([[3, 1, 4],
        [1, 5, 9],
        [2, 6, 5]])
```

In [88]:
```
b = np.array([1,2,3]).reshape(3,1)
q, r = np.linalg.qr(A)
x = np.dot(np.linalg.inv(r), np.dot(q.T, b))
x
```

Out[88]:
```
matrix([[ 0.26666667],
        [ 0.46666667],
        [-0.06666667]])
```

In [89]:
```
np.linalg.solve(A,b)
```

Out[89]:
```
array([[ 0.26666667],
       [ 0.46666667],
       [-0.06666667]])
```

Polynomial mathematics

In [90]:
```
import numpy as np
from numpy.polynomial import polynomial
```

In [91]:
```
first_polynomial = np.polynomial.Polynomial([2, -3, 1])
first_polynomial
```

Out[91]: $x \mapsto 2.0 - 3.0\,x + 1.0\,x^2$

In [92]:
```
second_polynomial = np.polynomial.Polynomial.fromroots([1, 2])
second_polynomial
```

Out[92]: $x \mapsto 2.0 - 3.0\,x + 1.0\,x^2$

In [93]:
```
first_polynomial.roots()
```

Out[93]:
```
array([1., 2.])
```

In [94]:
```
second_polynomial.roots()
```

Out[94]:
```
array([1., 2.])
```

$y=x^4+2x^3+3x^2+4x+5$, x=1

y=?

In [95]:
```
np.polyval([5,4,3,2,1], 1)
```

Out[95]:
```
15
```

In [96]:
```
third_polynomial = np.polynomial.Polynomial([1,2,3,4,5])
third_polynomial
```

Out[96]: $x \mapsto 1.0 + 2.0\,x + 3.0\,x^2 + 4.0\,x^3 + 5.0\,x^4$

In [97]:
```
integral=third_polynomial.integ()
integral
```

Out[97]: $x \mapsto 0.0 + 1.0\,x + 1.0\,x^2 + 1.0\,x^3 + 1.0\,x^4 + 1.0\,x^5$

In [98]:
```
integral.deriv()
```

Out[98]: $x \mapsto 1.0 + 2.0\,x + 3.0\,x^2 + 4.0\,x^3 + 5.0\,x^4$

In [99]:
```
derivative=third_polynomial.deriv()
derivative
```

Out[99]: $x \mapsto 2.0 + 6.0\,x + 12.0\,x^2 + 20.0\,x^3$

Application Linear Regression

House market

- Input: Price data from 2012 - 2021
- Output: Avarage house market 2022?
- Asume Relationship - squared
- $y=ax^2+bx+c$

In [111…
```
import numpy as np
import matplotlib.pyplot as plt
```

In [112…
```
year = np.arange(1,11)
price = np.array([129000, 133000, 138000, 144000, 142000, 141000, 150000, 135000, 13400
year
```

Out[112…
```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

In [113…
```
a, b, c = np.polyfit(year, price, 2)
print ("a:",a)
print ("b:",b)
print ("c:",c)
```

```
a: -594.6969696969702
b: 7032.575757575754
c: 122516.66666666669
```

In [114…
```
print("Estimated price for 2022:",a*11**2 + b*11 + c )
```

```
Estimated price for 2022: 127916.66666666658
```

In [118…
```
plt.plot(year,price, color = 'blue')
plt.scatter(year,price, color = 'blue')
plt.scatter(11, a*11**2 + b*11 + c ,color='red')
plt.title('Linear regression')
plt.xlabel('year')
plt.ylabel('average house price')
```

Out[118… Text(17.583333333333336, 0.5, 'average house price')

In [ ]: