**Q1. If you have any, what are your choices for increasing the comparison between different figures on the same graph**?

**HOW TO CHOOSE WHICH TYPE OF GRAPH TO USE?**

**. . . a Line graph.**

Line graphs are used to track changes over short and long periods of time. When smaller changes exist, line graphs are better to use than bar graphs. Line graphs can also be used to compare changes over the same period of time for more than one group.

**. . . a Pie Chart.**

Pie charts are best to use when you are trying to compare parts of a whole. They do not show changes over time.

**. . . a Bar Graph.**

Bar graphs are used to compare things between different groups or to track changes over time. However, when trying to measure change over time, bar graphs are best when the changes are larger.

**. . . an Area Graph.**

Area graphs are very similar to line graphs. They can be used to track changes over time for one or more groups. Area graphs are good to use when you are tracking the changes in two or more related groups that make up one whole category (for example public and private groups).

**. . . an X-Y Plot.**

X-Y plots are used to determine relationships between the two different things. The x-axis is used to measure one event (or variable) and the y-axis is used to measure the other. If both variables increase at the same time, they have a positive relationship. If one variable decreases while the other increases, they have a negative relationship. Sometimes the variables don't follow any pattern and have no relationship.

**Q2. Can you explain the benefit of compound interest over a higher rate of interest that does not compound after reading this chapter?**

Compound interest **causes your wealth to grow faster**. It makes a sum of money grow at a faster rate than simple interest because you will earn returns on the money you invest, as well as on returns at the end of every compounding period. This means that you don't have to put away as much money to reach your goals!

Let us discuss the formula for compound interest. The formula to calculate compound interest annually is given by:

A = P(1 + R/100)t

Compound Interest = A – P

Where,   
A is amount   
P is the principal amount   
R is the rate and   
T is the time span

|  |
| --- |
| # Python3 program to find compound  # interest for given values.      def compound\_interest(principle, rate, time):        # Calculates compound interest      Amount = principle \* (pow((1 + rate / 100), time))      CI = Amount - principle      print("Compound interest is", CI)      # Driver Code  compound\_interest(10000, 10.25, 5) |

**Output**

Compound interest is 6288.946267774416

**Time Complexity: O(1)**since no loop is used the algorithm takes up constant time to perform the operations  
**Auxiliary Space: O(1)**since no extra array is used so the space taken by the algorithm is constant

**Q3. What is a histogram, exactly? Name a numpy method for creating such a graph.**

Numpy has a built-in numpy. histogram() function which represents the frequency of data distribution in the graphical form. The rectangles having equal horizontal size corresponds to class interval called bin and variable height corresponding to the frequency.

*numpy.histogram(data, bins=10, range=None, normed=None, weights=None, density=None)* Import libraries

**import** numpy as np

# Creating dataset

a = np.random.randint(100, size =(50))

# Creating histogram

np.histogram(a, bins = [0, 10, 20, 30, 40,

                        50, 60, 70, 80, 90,

                        100])

hist, bins = np.histogram(a, bins = [0, 10,

                                     20, 30,

                                     40, 50,

                                     60, 70,

                                     80, 90,

                                     100])

# printing histogram

print()

print (hist)

print (bins)

print()

**Q4. If necessary, how do you change the aspect ratios between the X and Y axes?**

Setting the aspect ratio of the Matplotlib plot in Python

The method set\_aspect() is used to set the aspect ratio. The parameter of this method is a number which is a division of the X-axis with respect to the Y-axis.

We are taking a line plot using plot()

1] With no fixed aspect ratio

import matplotlib.pyplot as plt

import numpy as np

#Setting the axes.

x = np.arange(1,5,0.2)

y = np.sin(7\*x)

#Setting the Dimentions of the Graph

plt.figure(figsize = (5,5))

plt.plot(x,y)

plt.title('plot with undefined aspect ratio')

plt.show()

After making changes and setting the absolute aspect ratio. Even the figure size is the same!

import matplotlib.pyplot as plt

import numpy as np

#Setting the axes.

x = np.arange(1,5,0.2)

y = np.sin(7\*x)

#Setting the Dimentions of the Graph

plt.figure(figsize = (5,5))

#selecting the current axis

ax = plt.gca()

#sets the ratio to 5

ax.set\_aspect(5)

plt.plot(x,y)

plt.title('plot with defined aspect ratio')

plt.show()

**Q5. Compare and contrast the three types of array multiplication between two numpy arrays: dot product, outer product, and regular multiplication of two numpy arrays.**

In Python if we have two numpy arrays which are often referred as a vector. The ‘\*’ operator and numpy.dot() work differently on them. It’s important to know especially when you are dealing with data science or competitive programming problem.

### Working of ‘\*’ operator

‘\*’ operation caries out element-wise multiplication on array elements. The element at a[i][j] is multiplied with b[i][j] .This happens for all elements of array.  
**Example:** 

Let the two 2D array are v1 and v2:-

v1 = [[1, 2], [3, 4]]

v2 = [[1, 2], [3, 4]]

Output:

[[1, 4]

[9, 16]]

From below picture it would be clear.

**Working of numpy.dot()**

It carries of normal matrix multiplication . Where the condition of number of columns of first array should be equal to number of rows of second array is checked than only numpy.dot() function take place else it shows an error.   
**Example:** 

Let the two 2D array are v1 and v2:-

v1=[[1, 2], [3, 4]]

v2=[[1, 2], [3, 4]]

Than numpy.dot(v1, v2) gives output of :-

[[ 7 10]

[15 22]]

**Examples 1:** 

* Python3

|  |  |
| --- | --- |
| import numpy as np      # vector v1 of dimension (2, 2)  v1 = np.array([[1, 2], [1, 2]])    # vector v2 of dimension (2, 2)  v2 = np.array([[1, 2], [1, 2]])    print("vector multiplication")  print(np.dot(v1, v2))    print("\nElementwise multiplication of two vector")  print(v1 \* v2)  **Q6. Before you buy a home, which numpy function will you use to measure your monthly mortgage payment?**  numpy function .pmt  In order to calculate the monthly mortgage payment, you will use the numpy function . pmt(rate, nper, pv) where: rate = The periodic (monthly) interest rate. nper = The number of payment periods (months) in the lifespan of the mortgage loan.  In order to make sure you can afford the home, you will have to calculate the monthly mortgage payment you will have to make on a loan that size.  Now, since you will be paying a monthly mortgage, you will have to convert each of the parameters into their monthly equivalents. Be careful when adjusting the interest rate, which is compounding!  In order to calculate the monthly mortgage payment, you will use the numpy function .pmt(rate, nper, pv) where:   * rate = The periodic (monthly) interest rate * nper = The number of payment periods (months) in the lifespan of the mortgage loan * pv = The total value of the mortgage loan   You have been given a 30-year mortgage loan quote for your desired amount at 3.75%. The value of the mortgage loan is available as mortgage\_loan.  The annual mortgage rate is available as mortgage\_rate  import numpy as np  # Derive the equivalent monthly mortgage rate from the annual rate  mortgage\_rate\_periodic = \_\_\_\_  # How many monthly payment periods will there be over 30 years?  mortgage\_payment\_periods = \_\_\_\_  # Calculate the monthly mortgage payment (multiply by -1 to keep it positive)  periodic\_mortgage\_payment = -1\*np.pmt(\_\_\_\_, \_\_\_\_, \_\_\_\_)  print("Monthly Mortgage Payment: " + str(round(periodic\_mortgage\_payment, 2)))  **Q7. Can string data be stored in numpy arrays? If so, list at least one restriction that applies to this data.**  The dtype of any numpy array containing string values is the maximum length of any string present in the array. Once set, it will only be able to store new string having length not more than the maximum length at the time of the creation.  NumPy builds on (and is a successor to) the successful Numeric array object. Its goal is to create the corner-stone for a useful environment for scientific computing. NumPy provides two fundamental objects: an N-dimensional array object (ndarray) and a universal function object (ufunc).  The dtype of any numpy array containing string values is the maximum length of any string present in the array. Once set, it will only be able to store new string having length not more than the maximum length at the time of the creation. If we try to reassign some another string value having length greater than the maximum length of the existing elements, it simply discards all the values beyond the maximum length.  In this post we are going to discuss ways in which we can overcome this problem and create a numpy array of **arbitrary length**.  Let’s first visualize the problem with creating an arbitrary length numpy array of string type.  # importing numpy as np  import numpy as np    # Create the numpy array  country = np.array(['USA', 'Japan', 'UK', '', 'India', 'China'])    # Print the array  print(country)  Create a numpy array of arbitrary length.  **Solution :** While creating the array assign the ‘object’ dtype to it. This lets you have all the behaviors of the python string.   |  | | --- | | # importing the numpy library as np  import numpy as np    # Create a numpy array  # set the dtype to object  country = np.array(['USA', 'Japan', 'UK', '', 'India', 'China'], dtype = 'object')    # Print the array  print(country) | |