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

To increase the comparison between different figures on the same graph, you can:

1. **Use multiple subplots**: Use plt.subplot() in Matplotlib to arrange multiple graphs in a single figure. This makes it easier to compare data side by side.
2. import matplotlib.pyplot as plt
3. plt.subplot(1, 2, 1) # First subplot
4. plt.plot(data1)
5. plt.subplot(1, 2, 2) # Second subplot
6. plt.plot(data2)
7. plt.show()
8. **Plot different data with different colors or markers**: Different colors, line styles, or markers can be used to differentiate multiple datasets in the same plot for easy comparison.
9. plt.plot(x, data1, label="Data 1", color='blue')
10. plt.plot(x, data2, label="Data 2", color='red')
11. plt.legend() # Show the legend
12. **Use transparency (alpha)**: You can adjust the transparency of lines to avoid overlapping and allow better visibility of all the data.
13. plt.plot(x, data1, alpha=0.5) # Make the plot semi-transparent
14. plt.plot(x, data2, alpha=0.5)
15. **Adjust axis limits (xlim/ylim)**: Ensure both plots are on the same scale by adjusting the axis limits.
16. plt.xlim(0, 10)
17. plt.ylim(0, 100)

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

The key benefit of **compound interest** over simple interest (which doesn't compound) is that compound interest allows you to earn interest on both the initial principal and the accumulated interest over time. This means your money grows at an accelerating rate.

In contrast, with **simple interest**, you only earn interest on the principal, which means your investment grows linearly rather than exponentially.

Example:

* **Compound interest**: $100 invested at 5% annually would grow to $105 in the first year. The next year, it earns interest on the new total ($105), not just the original $100.
* **Simple interest**: The same $100 at 5% annually would only earn $5 each year, regardless of the previous year's growth.

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

A **histogram** is a graphical representation of the distribution of a dataset. It divides the data into **bins** (intervals) and counts how many data points fall into each bin. Histograms are useful for understanding the distribution, skewness, and spread of data.

In **NumPy**, you can create a histogram using the np.histogram() method, which returns the frequency counts for each bin, and you can visualize it with **Matplotlib**.

Example:

import numpy as np

import matplotlib.pyplot as plt

data = np.random.randn(1000) # Generate random data

plt.hist(data, bins=30) # Create a histogram with 30 bins

plt.show()

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

You can change the aspect ratio between the **X** and **Y** axes using **Matplotlib**'s set\_aspect() function. This allows you to control the relative scaling of the axes.

Example:

import matplotlib.pyplot as plt

x = [0, 1, 2, 3, 4]

y = [0, 1, 4, 9, 16]

plt.plot(x, y)

# Change aspect ratio to be equal

plt.gca().set\_aspect('equal', adjustable='box') # 'equal' gives equal scaling of both axes

plt.show()

You can also specify numeric values (e.g., set\_aspect(1) for equal scaling, set\_aspect(2) for twice as wide on the X-axis).

**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.**

* **Regular multiplication** (\*):
  + This is **element-wise multiplication**, meaning each element of one array is multiplied by the corresponding element in the other array.
  + Requires the arrays to have the same shape (or be broadcastable).
* A = np.array([1, 2, 3])
* B = np.array([4, 5, 6])
* result = A \* B # Output: [4 10 18]
* **Dot product** (np.dot() or @):
  + This is the **sum of the products of corresponding elements** in the arrays, usually used for **vector multiplication** or **matrix multiplication**.
  + For 1D arrays, it computes the inner product (a scalar). For 2D arrays, it computes matrix multiplication.
* A = np.array([1, 2])
* B = np.array([3, 4])
* result = np.dot(A, B) # Output: 11 (1\*3 + 2\*4)
* **Outer product** (np.outer()):
  + This computes the **outer product** of two vectors, resulting in a matrix where each element is the product of the corresponding elements of the two arrays.
* A = np.array([1, 2])
* B = np.array([3, 4])
* result = np.outer(A, B) # Output: [[3 4] [6 8]]

**Q6. Before you buy a home, which numpy function will you use to measure your monthly mortgage payment?**

To calculate your monthly mortgage payment, you can use the **np.pmt()** function, which calculates the payment for a loan based on constant payments and a constant interest rate.

Example:

import numpy as np

rate = 0.05 / 12 # Monthly interest rate (annual rate / 12)

nper = 30 \* 12 # Number of payments (e.g., for 30 years)

pv = 300000 # Loan amount

payment = np.pmt(rate, nper, -pv)

print(payment)

This will calculate the monthly payment for a mortgage loan of $300,000 with a 5% annual interest rate and a 30-year term.

**Q7. Can string data be stored in numpy arrays? If so, list at least one restriction that applies to this data.**

Yes, **string data** can be stored in **NumPy arrays**. However, there are some restrictions:

* **Fixed-length strings**: When creating an array of strings, you must specify a fixed length for the strings (e.g., dtype='S10' for strings up to 10 characters long).
* If you assign a string longer than the fixed length, it will be truncated to fit the specified length.

Example:

import numpy as np

arr = np.array(['hello', 'world'], dtype='S5') # Fixed-length strings (max 5 characters)

print(arr) # Output: [b'hello' b'world']

In this example, any string longer than 5 characters will be truncated.