Python for Data Science Assignment-1

● MCQs

1. What is the output of the following Python code snippet?
   x = [1, 2, 3]
   y = x
   y.append(4)
   print(x)

   a) [1, 2, 3]
   b) [1, 2, 3, 4]
   c) NameError
   d) TypeError

   Ans: b) [1,2,3,4]

2. Which of the following data types is an ordered, immutable sequence of elements?

   a) List

   b) Set

   c) Tuple

   d) Dictionary

   Answer: c) Tuple

● Descriptive

1. Explain the fundamental difference between **mutable** and **immutable** data types in Python with a clear example.

Answer:

The key difference between mutable and immutable data types in Python lies in whether their state can be changed after creation.

- A **mutable** object can be modified in-place after it has been created. When you assign a new variable to an existing mutable object (e.g., `list_b = list_a`), both variables refer to the exact same object in memory. Any changes made through one variable will be reflected in the other. Examples include **lists**, **dictionaries**, and **sets**.

  Example:

  list_a = [1, 2, 3]

  list_b = list_a

  list_b.append(4)

  print(list_a)

  # Output: [1, 2, 3, 4]

- An **immutable** object cannot be changed after it's created. When you try to "modify" an immutable object, you are actually creating a completely new object in memory and assigning the new value to the variable. Examples include **integers**, **strings**, and **tuples**. Example:

  str_a = "hello"

  str_b = str_a

  str_b += " world"

  print(str_a)

  # Output: hello

  # str_a remains unchanged because `str_b += " world"` created a new string object.

2. For data science and numerical computing, NumPy arrays are preferred over standard Python lists. Explain two major advantages of using NumPy arrays that make them more efficient for these tasks.

Answer:

NumPy arrays are the cornerstone of numerical computing in Python primarily due to their superior efficiency and functionality.

1. Efficiency (Speed and Memory): NumPy arrays are significantly faster and more memory-efficient than Python lists.
    - Speed: NumPy operations are implemented in C and Fortran, which are highly optimized languages. This allows NumPy to perform calculations much faster than the equivalent loops in native Python lists. This process is known as vectorization, where operations are applied to entire arrays at once, eliminating the need for slow, explicit Python loops.
    - Memory: NumPy arrays store elements of the same data type in a contiguous block of memory. In contrast, Python lists store pointers to objects that can be scattered throughout memory. This contiguous storage of typed data saves a substantial amount of memory and allows for more efficient access.
2. Functionality (Vectorized Operations): NumPy provides a vast collection of powerful, built-in mathematical functions that can be applied to entire arrays without writing a loop. This leads to cleaner, more readable, and concise code.