Module – 2 (Fundamentals of python)

4.) How memory is managed in Python?

- **According to the Python memory management documentation, Python has a private heap that stores our program's objects and data structures. Python memory manager takes care of the bulk of the memory management work and allows us to concentrate on our code.**
- **There are two types of memory allocation in Python, static and dynamic**.
  - **Static memory**

    The stack data structure provides **static memory allocation**, meaning the variables are in the stack memory. Statically assigned variables, as the name implies, are permanent; this means that they must be allocated in advance and persist for the duration of the program. Another point to remember is that we cannot reuse the memory allocated in the stack memory. Therefore, memory reusability is not possible.

  - **Dynamic memory**

    The dynamic memory allocation **uses heap data structures in its implementation, implying that variables are in the heap memory. As the name suggests, dynamically allocated variables are not permanent and can be changed while a program is running. Additionally, allotted memory can be relinquished and reused. However, it takes longer to complete because dynamic memory allocation occurs during program execution. Furthermore, after utilizing the allocated memory, we must release it. Otherwise, problems such as memory leaks might arise.**

- **Memory management in Python involves a private heap containing all Python objects and data structures. The management of this private heap is ensured internally by the Python memory**

manager. The Python memory manager has different components which deal with various dynamic storage management aspects, like sharing, segmentation, preallocation or caching.

5.) What is the purpose continue statement in python?

- In Python, the `continue` statement is used inside loops (such as `for` and `while` loops) to skip the rest of the current iteration and move on to the next iteration. When the `continue` statement is encountered, the remaining code inside the loop for the current iteration is skipped, and the loop proceeds to the next iteration, if any.
- The purpose of the `continue` statement is to control the flow of the loop and skip certain iterations based on a specific condition. It is often used when you want to skip over certain iterations when a certain condition is met, but you don't want to exit the loop entirely.
- for x in range(9):

```
    if x == 3:
        continue
    print(x, end = " ")
```

14.) What are negative indexes and why are they used?

- **Python is a powerful and versatile programming language that has many features and capabilities. One of these features is negative indexing, which allows you to access elements of a sequence (such as a list, a string, or a tuple) from the end, using negative numbers as indexes.**
- **Negative Indexing is used to in Python to begin slicing from the end of the string i.e. the last. Slicing in Python gets a sub-string from a string. The slicing range is set as parameters i.e. start, stop and step.**

- **Negative indexing is used in Python to manipulate sequence objects such as lists, arrays, strings, etc. Negative indexing retrieves elements from the end by providing negative numbers as sequence indexes.**

- **Python adds the length of the sequence to the negative index to get the corresponding positive index. For example, if you have a list with four elements and you use an index of -1, Python will add 4 (the length of the list) to -1 (the index) and get 3 (the positive index). Then it will return the element at position 3 in the list.**

- **If you have a sequence with n elements and you use an index of -i (where i is a positive integer), Python will add n (the length of the sequence) to -i (the index) and get n-i (the positive index). Then it will return the element at position n-i in the sequence.**