**Fundamentals Of Python**

**Que-1**: **How memory is managed in Python?**

**Ans:** Memory management in Python is primarily managed by the Python runtime environment using a combination of strategies like reference counting, garbage collection, and memory pooling. Here's an overview of how memory is managed in Python:

**1. Reference Counting:** Python uses a reference counting mechanism to keep track of the number of references to each object. When an object's reference count drops to zero, meaning it's no longer being used, the memory occupied by that object can be freed immediately. This helps manage memory efficiently and reclaim memory as soon as it's no longer needed.

**2. Garbage Collection:** While reference counting can efficiently reclaim memory for objects with simple reference structures, it has limitations when dealing with cyclic references (objects referencing each other in a loop). Python's cyclic garbage collector detects and clears cyclic references. It identifies objects that are unreachable and can't be accessed through the reference chain from the root objects, marking them as eligible for garbage collection.

**3. Memory Pooling:** Python uses memory pools to manage memory allocation for small objects. It helps reduce the overhead of frequently allocating and deallocating small chunks of memory. Python divides memory into fixed-size blocks and manages these blocks in pools. Objects of the same size are allocated from the same pool, reducing fragmentation and improving memory utilization.

**4. Automatic Memory Management:** Python automatically handles memory allocation and deallocation for most objects. When objects are no longer referenced, they are eventually reclaimed by the garbage collector, which frees up the associated memory.

**5. Memory Optimizations:** Python employs various memory optimization techniques, such as interning small immutable objects (like small strings) to reuse memory, which reduces the overhead of creating duplicate objects.

**6. Memory Views:** Python provides memory views that allow efficient sharing of memory between data structures without copying the underlying data. This is useful for scenarios where memory efficiency is crucial, such as working with large arrays or binary data.

**7. Memory Profiling and Monitoring:** Python provides tools and libraries for memory profiling and monitoring, such as the **`memory\_profiler`** package, which can help identify memory leaks or inefficient memory usage patterns.

**Que-2: What is the purpose continue statement in python?**

**Ans:** In Python, the `continue` statement is used inside loops (such as `for` and `while` loops) to control the flow of execution. When the `continue` statement is encountered within a loop, it immediately stops the current iteration and moves on to the next iteration, bypassing the remaining code within the loop's block for the current iteration.

The primary purpose of the `continue` statement is to skip specific iterations of a loop under certain conditions, without prematurely exiting the entire loop. It is particularly useful when you want to skip some part of the loop's block based on a condition, but you still want to continue looping through the remaining items.

**Que-3:** **What are negative indexes and why are they used?**

**Ans:** Negative indexes in Python are used to access elements in a sequence (like strings, lists, tuples, etc.) from the end of the sequence rather than from the beginning. The last element of the sequence has an index of **-1**, the second-to-last element has an index of **-2**, and so on. Negative indexes provide a convenient way to access elements from the end without needing to know the exact length of the sequence.