1. **What is memory management in python.**

Ans. Memory management in Python involves the allocation and deallocation of memory for objects used in a Python program. Python handles memory management automatically, utilizing techniques like garbage collection to reclaim unused memory and memory pools for efficient allocation.

Key aspects include:

1. **Reference Counting: Python keeps track of the number of references to each object. When an object's reference count drops to zero, it's no longer accessible and is marked for garbage collection.**
2. **Garbage Collection: Python's garbage collector identifies and reclaims memory from unreachable objects, especially in cases of circular references that reference counting alone can't handle.**
3. **Memory Pooling: Python uses memory pools, especially for smaller objects, to reduce overhead in memory allocation and improve performance.**

2. What is garbage collection.

Garbage collection (GC) is the process of automatically identifying and freeing up memory that is no longer in use by the program, helping to prevent memory leaks and optimize available memory. In Python, the garbage collector reclaims memory by deallocating objects that are no longer accessible.

Python’s garbage collection system uses two primary techniques:

1. **Reference Counting: Each object has a counter tracking how many references point to it. When this count reaches zero, the object is no longer accessible and can be freed. However, reference counting alone can’t handle circular references, where two or more objects reference each other.**
2. **Cycle Detection: To address circular references, Python has a cyclic garbage collector that identifies reference cycles and breaks them, allowing for memory to be reclaimed. This collector periodically runs in the background and inspects objects that might be involved in cycles.**

3. What is garbage collector.

A garbage collector is a system within Python that automatically manages memory by tracking and disposing of objects that are no longer in use, freeing up space for new objects.

In Python, the garbage collector is part of the `gc` module, and it handles two main tasks:

1. **Reference Counting: Python keeps track of the number of references to each object. When the reference count of an object drops to zero, the garbage collector deallocates its memory immediately, as it's no longer needed.**
2. **Cycle Detection: For cases where objects reference each other in cycles (creating reference loops), the garbage collector performs periodic checks to detect and break these cycles. This ensures that objects involved in reference cycles can still be collected even if their reference counts are non-zero.**

The garbage collector works automatically, but Python also allows developers to manually manage garbage collection through the `gc` module (e.g., triggering collections or tuning collection frequency).

4. What is difference between list and tuple in python.

Ans. The primary differences between a list and a tuple in Python are:

1. **Mutability:**
   - **List: Mutable, meaning elements can be modified, added, or removed after the list is created.**
   - **Tuple: Immutable, so elements cannot be changed, added, or removed after creation.**
2. **Syntax:**
   - **List: Created using square brackets, e.g., `my_list = [1, 2, 3]`.**
   - **Tuple: Created using parentheses, e.g., `my_tuple = (1, 2, 3)`.**
3. **Performance:**
   - **List: Generally slower than tuples for iteration and operations because of the overhead associated with mutability.**
   - **Tuple: Faster due to immutability, making them ideal for fixed collections of items.**
4. **Use Cases:**
   - **List: Used when you need a collection of items that may change during the program's execution.**
   - **Tuple: Preferred for fixed data collections, where immutability is desired, such as coordinates or items that should remain constant.**
5. **Memory Efficiency:**
   - **Tuple: Consumes slightly less memory than lists of the same size due to its immutable nature.**

5. What is the difference between set and tuple.

Ans. The primary differences between a set and a tuple in Python are:

1. **Mutability:**
   - **Set: Mutable, so you can add or remove items after creating it.**
   - **Tuple: Immutable, meaning its contents cannot be changed once created.**

2. **Order:**
    - **Set: Unordered, meaning elements have no specific order and cannot be indexed or sliced.**
    - **Tuple: Ordered, with elements in a fixed sequence, allowing indexing and slicing.**
3. **Duplicates:**
    - **Set: Does not allow duplicates; each element must be unique.**
    - **Tuple: Allows duplicates; elements can appear more than once.**
4. **Syntax:**
    - **Set: Defined with curly braces, e.g., `my_set = {1, 2, 3}`.**
    - **Tuple: Defined with parentheses, e.g., `my_tuple = (1, 2, 3)`.**
5. **Use Cases:**
    - **Set: Useful for storing unique items, performing set operations (like unions, intersections), and checking membership efficiently.**
    - **Tuple: Ideal for fixed, ordered collections of elements, often used to group related data.**

6. What is the difference between append and extend.

Ans. The primary differences between `append` and `extend` methods in Python lists are:

1. **Functionality:**
    - **append(): Adds its argument as a single element to the end of the list. If you pass a list to `append`, it will add the entire list as one item (i.e., a nested list).**
    - **extend(): Iterates over its argument, adding each element to the list individually. If you pass a list to `extend`, each item in that list will be added as a separate element.**
2. **Syntax Example:**

```
my_list = [1, 2, 3]

# Using append()

my_list.append([4, 5])

# Result: [1, 2, 3, [4, 5]]

# Using extend()

my_list.extend([4, 5])

# Result: [1, 2, 3, 4, 5]
```

3. Use Case:

- **append(): Used when you want to add a single element or object (even if it's a list).**
- **extend(): Used when you want to add multiple elements individually from an iterable, such as a list or tuple.**

7.