**Que - 4: How memory is managed in Python?**

In simple terms, Python manages memory automatically for you, so you don't have to worry about it much. Here's how it works:

1. **Memory Allocation**:
   * When you create variables (like numbers, lists, or strings), Python automatically gives them the memory they need.
2. **Reference Counting**:
   * Python keeps track of how many times an object (e.g., a variable) is used. If no one is using it anymore, Python clears that memory to free up space.
3. **Garbage Collection**:
   * If objects are stuck in a loop (they reference each other and can't be cleared by counting), Python has a special system called **garbage collection** to clean them up.
4. **Reusing Memory**:
   * Python tries to be smart by reusing memory for small things like small numbers and short strings, instead of creating new memory every time.

**In short:**

Python automatically handles memory behind the scenes using a system that frees up unused memory, so you don't need to worry about managing memory manually.

**Que - 5: What is the purpose continue statement in python?**

The continue statement in Python is used to skip the rest of the code inside the current loop iteration and move to the next iteration immediately.

**Purpose:**

* **Skip the current iteration**: When Python encounters a continue statement inside a loop (like a for or while loop), it jumps back to the beginning of the loop for the next iteration, without executing the code that comes after continue.

**Example:**

python

Copy code

for number in range(1, 6):

if number == 3:

continue # Skip number 3 and move to the next iteration

print(number)

**Output:**

Copy code

1

2

4

5

In this example, when the loop reaches number == 3, the continue statement tells Python to skip printing 3 and move to the next number.

**In short:**

The continue statement is useful when you want to skip certain steps in a loop without breaking out of the loop entirely.

**Que – 14: What are negative indexes and why are they used?**

In Python, **negative indexes** allow you to access elements of sequences (like lists, strings, and tuples) from the **end** rather than the beginning. The last element has an index of -1, the second last has an index of -2, and so on.

**Why Negative Indexes are Used:**

1. **Easy Access to End Elements**: Negative indexing makes it easier to access elements starting from the end without needing to calculate the length of the sequence.
2. **Shorter, Cleaner Code**: It allows for more concise code when you want to reference elements near the end of a list or string.

**Example:**

python

Copy code

my\_list = [10, 20, 30, 40, 50]

# Accessing elements using positive indexes

print(my\_list[0]) # Output: 10 (first element)

print(my\_list[4]) # Output: 50 (last element)

# Accessing elements using negative indexes

print(my\_list[-1]) # Output: 50 (last element)

print(my\_list[-2]) # Output: 40 (second to last element)

**How Negative Indexes Work:**

* **Index -1**: Refers to the last element.
* **Index -2**: Refers to the second last element.
* This continues backward as you increase the negative value.

**Example with Strings:**

python

Copy code

my\_string = "Python"

# Accessing characters using negative indexing

print(my\_string[-1]) # Output: n (last character)

print(my\_string[-3]) # Output: h (third to last character)

**In short:**

Negative indexes are used to easily access elements from the end of a sequence without needing to know the exact length of the sequence.