1. **Alias:**

Alias occurs when two or more variable names refer to the same memory address in programming. This means that modifying the object through one reference also affects all other references pointing to the same memory.

Effects of Alias:

* Changes in one reference affect all other references to the same object.
* May cause unexpected behavior if not handled properly.
* Useful in some cases, such as object sharing between multiple modules.

**Example :**

frA = ["Apple", "Banana", "Cherry"] # List stored in memory area A

frB = frA # frB becomes an alias of frA, referencing memory area A

In this case:

* frA and frB are not separate objects but rather different names for the same list in memory.
* Any modification made to frA will also be reflected in frB, as they both reference the same memory location.

**Avoiding Alias by Copying Data**

To create an independent copy instead of an alias, the following methods can be used:

frC = frA.copy() # Or use frA[:], or list(frA)

frC.append("Mango")

print(frA) # ['Apple', 'Banana', 'Cherry', 'Dragonfruit']

print(frC) # ['Apple', 'Banana', 'Cherry', 'Dragonfruit', 'Mango']

**Now, frC is a completely independent copy, and any changes made to frC do not affect frA.**

1. **Garbage Collection in Python:**

Garbage Collection (GC) is a mechanism that automatically reclaims memory occupied by objects that are no longer in use. The primary purpose of GC is to prevent memory leaks and optimize program performance.

**Reference Counting Mechanism**

Each object in Python has a reference count that tracks the number of variables pointing to it. The reference count increases when a new reference is created and decreases when a reference is deleted.

**Example**

import sys

x = [1, 2, 3] # Create a list

print(sys.getrefcount(x)) # Output: 2 (x + sys.getrefcount call)

y = x # Create an alias y pointing to the same list

print(sys.getrefcount(x)) # Output: 3 (x, y, and sys.getrefcount)

del y # Delete alias y

print(sys.getrefcount(x)) # Output: 2 (x + sys.getrefcount)

**When y is deleted, the reference count of the list decreases. If the count reaches zero, Python automatically frees the memory.**

**Garbage Collector - Handling Cyclic References**

Reference counting has a significant drawback: it cannot handle cyclic references (when two or more objects reference each other). Python uses Garbage Collector (GC) to detect and reclaim memory occupied by such cyclic references.

**Example**

import gc

class Node:

def \_\_init\_\_(self, name):

self.name = name

self.ref = None # Reference variable

A = Node("A")

B = Node("B")

A.ref = B # A references B

B.ref = A # B references A

del A

del B

gc.collect() # Trigger Garbage Collector to reclaim memory

In this scenario:

* A references B, and B references A, creating a cyclic reference.
* When del A and del B are executed, both objects still exist because they reference each other.
* gc.collect() detects the cyclic reference and properly frees the memory.