Great! Let's break down and **explain your entire code step by step**, focusing on key concepts like:

* sys.getrefcount()
* variable reassignment
* mutability and immutability
* object references
* shallow vs deep copies
* == vs is

**🔧 1. sys.getrefcount() – Reference Count in Python**

python

CopyEdit

import sys

sys.getrefcount(24601) # Output: 3

* sys.getrefcount(obj) returns **how many references** exist to that object in memory.
* The output 3 means the object 24601 is currently referenced 3 times (including the temporary one passed to the function).
* Strings like 'Hari' showing 4294967295 is an **unrealistic number**, possibly due to platform-specific issues or object internals (e.g., string interning on Windows).

**🔄 2. Reassigning Variables**

python

CopyEdit

a = 3

a = 'Hello Earth'

a = 3.14

a = 5

* Variable a is repeatedly reassigned.
* Each time, Python **creates a new object** and a **points** to the new one.
* Old values are garbage collected if no other variable refers to them.

python

CopyEdit

a = a + 2 # 5 + 2 = 7

✅ a becomes 7.

**🧠 3. Mutable vs Immutable Behavior**

python

CopyEdit

myListOne = [1,2,3]

myListTwo = myListOne

✅ Both myListOne and myListTwo **refer to the same list** in memory.

python

CopyEdit

myListOne = 'Ironman'

✅ Now, myListOne points to a **new string object**, but myListTwo still refers to the **original list** [1, 2, 3].

**🧪 4. Changing List Elements**

python

CopyEdit

myListOne = [1,2,3]

myListOne[0] = 55

✅ myListOne becomes [55, 2, 3].  
🔁 myListTwo is unchanged because it was no longer pointing to the same object after 'Ironman'.

**🔗 5. Shared References**

python

CopyEdit

l1 = [1,2,3]

l2 = l1

l1[0] = 77

✅ Now **both l1 and l2 are [77, 2, 3]** because they refer to the **same object**.

**🔂 6. Rebinding a Reference**

python

CopyEdit

p1 = [1,2,3]

p2 = p1

p2 = [1,2,3]

p1[0] = 99

* Initially, p1 and p2 refer to the same list.
* Then p2 is assigned a **new list**, so p1 and p2 are **separate** now.

Result:

python

CopyEdit

p1 = [99, 2, 3]

p2 = [1, 2, 3]

**🪞 7. List Slicing (Shallow Copy)**

python

CopyEdit

h1 = [1,2,3]

h2 = h1[:]

✅ Creates a **shallow copy**. They are **two different objects**, but contain the same elements.

python

CopyEdit

h1[0] = 55

✅ Changes only h1, not h2.

**🧬 8. Deep Copy**

python

CopyEdit

import copy

h2 = copy.deepcopy(h1)

✅ Even nested structures are copied recursively.

**⚖️ 9. == vs is in Python**

**== checks value equality**

**is checks object identity (same memory address)**

python

CopyEdit

m = [1,2,3]

n = [1,2,3]

m == n # ✅ True (values are equal)

m is n # ❌ False (different objects)

python

CopyEdit

m = n

m is n # ✅ True (same object now)

**✅ Conclusion:**

| **Concept** | **Explanation** |
| --- | --- |
| getrefcount() | Shows how many references point to an object |
| Reassignment | Creates a new object, doesn't change the old |
| == vs is | == compares values, is compares memory |
| Mutable types (lists) | Can be changed; changes reflect in all references |
| Immutable types (ints, strings) | Cannot be changed in place |
| Shallow copy ([:]) | Creates a new list, same inner values |
| Deep copy (copy.deepcopy) | Creates an entirely new copy recursively |