**🌀 Python Variable Life Cycle – Explained for Beginners**

**🔹 1. Variable Declaration (Creation)**

x = 10

📌 You just **created a variable** named x and assigned it the value 10.

* Python creates an object 10 in memory.
* The name x is linked (i.e., *references*) to that object.

🧠 Think of it like this:  
A label x is stuck to a jar that contains 10.

**🔠 2. Rules for Naming Variables**

| **Rule** | **Example** |
| --- | --- |
| Must start with a letter or \_ | ✅ name, \_score |
| Cannot start with a number | ❌ 1value |
| Only letters, digits, underscore | ✅ user\_1, ❌ user-name |
| Cannot be a Python keyword | ❌ if = 5 |

**💾 3. Value Stored in Memory**

* Each value is stored at a specific location in memory.
* A variable name points to that location.

x = [1, 2, 3]

Here, x points to a list object stored somewhere in memory.

Use id() to check the memory address:

print(id(x)) # e.g., 140556798238016

**🔗 4. Reference and Reference Counting**

🧠 A **reference** is simply a link from a variable to the value it holds.

🧮 Python automatically counts how many variables point to a value.

**🔍 You can check it using:**

import sys

x = [1, 2, 3]

print(sys.getrefcount(x)) # Might print 2

👀 Why 2?

* One from x
* One temporary from the getrefcount() function call

**♻️ 5. Interning (Value Reuse for Small Immutables)**

🧠 Python automatically **reuses** some immutable values, such as:

* Integers from **-5 to 256**
* Common short strings (like "hello")

**✅ Examples:**

a = 100

b = 100

print(a is b) # True

**❌ Non-interned:**

x = 300

y = 300

print(x is y) # False

**✅ Interned strings:**

s1 = "hello"

s2 = "hello"

print(s1 is s2) # True

🔍 Use is to compare if two variables point to the **same object**.

**🧱 6. Memory Pooling (Optimization)**

🧠 **Pooling** refers to Python keeping frequently used values in memory for reuse.

* Interning is a part of memory pooling.
* Mostly applies to:
  + Integers from -5 to 256

x = 256

y = 256

print(x is y) # ✅ True

x = 257

y = 257

print(x is y) # ❌ False

**🔁 7. Reassignment or Scope Exit**

When a variable is reassigned or its scope ends:

x = 10

x = 20 # Now 10 is no longer referenced

Or after a function ends:

def temp():

a = [1, 2]

# After function ends, `a` disappears

**🗑️ 8. Garbage Collection**

When a value has **zero references**, Python’s garbage collector removes it to free memory.

import gc

gc.collect() # Manually trigger garbage collection

🧹 Happens automatically in background when needed.

**✅ Summary Table**

| **Step** | **Description** |
| --- | --- |
| 1️⃣ Declaration | Create variable and assign value |
| 2️⃣ Naming Rules | Follow valid variable name syntax |
| 3️⃣ Stored in Memory | Python creates object and links name to it |
| 4️⃣ Reference Counting | Python tracks how many names point to the object |
| 5️⃣ Interning | Reuse of small immutable values (e.g., -5 to 256) |
| 6️⃣ Memory Pooling | Common values are pre-allocated for reuse |
| 7️⃣ Reassignment/Scope End | Old references are removed |
| 8️⃣ Garbage Collection | Objects with no references are deleted from memory |