

[Skip to content](#)

## Chat history

You said:

python internal working in string full details

ChatGPT said:

Python में string एक बहुत ही important और commonly used data type है। इसका internal working यानी Python string memory management, immutability, object reference, interning mechanism आदि को समझना बहुत जरूरी है। नीचे मैं सब कुछ detail में Hindi और technical clarity के साथ explain कर रहा हूँ:

---

### ◆ Python String: Basic

Python में string को " " या ' ' के बीच लिखा जाता है:

```
s = "Ankit"
```

यह एक immutable sequence of Unicode characters है।

---

### ◆ Internal Working of String in Python

#### ✓ 1. Immutable Nature

Python में strings immutable होते हैं — मतलब:

```
a = "hello" a[0] = 'H' # ❌ Error
```

आप एक बार बने string को change नहीं कर सकते। अगर आप change करते हैं, तो नया object बनता है:

```
a = "hello" a = "world" # नया object बन गया
```

🔍 Internal Explanation:

- जब आप `a = "hello"` लिखते हैं, तो Python memory में एक string object बनाता है और `a` उस object को reference करता है।

- जब आप `"world"` assign करते हैं, तो एक नया object बनता है और `a` अब नए object को point करता है।
- 

## ✅ 2. String Interning (Memory Optimization)

Python कुछ strings को memory optimization के लिए intern करता है, मतलब एक ही immutable string के लिए वही memory address use करता है।

```
a = "hello" b = "hello" print(id(a) == id(b)) # True, because of
interning
```

लेकिन यह सिर्फ कुछ cases में होता है, जैसे:

- Short strings
- Strings with only alphanumeric and underscore characters

```
x = "hello_world" y = "hello_world" print(x is y) # True
```

⚠️ Complex strings या dynamically generated strings के लिए intern जरूरी नहीं है।

```
a = "hello world!" b = "hello world!" print(a is b) # False in most
cases
```

✅ You can force interning using `sys.intern()`:

```
import sys x = sys.intern("hello world!") y = sys.intern("hello
world!") print(x is y) # True
```

---

## ✅ 3. Memory Allocation and Object ID

हर string एक PyObject होता है जिसमें:

- reference count
- type pointer
- actual data (characters)
- length stored होती है।

Python internally string को एक C structure में represent करता है जिसे CPython में `PyUnicodeObject` कहते हैं।

```
s = "hello" print(id(s)) # Gives memory address
```

---

## ✓ 4. Garbage Collection in String

जब कोई string variable reference lose कर देता है, तो garbage collector उसे memory से हटा देता है।

```
a = "hello" a = None # अब "hello" unreferenced हो गया, eligible for GC
```

---

## ✓ 5. String Operations and Copying

क्योंकि strings immutable होते हैं, हर बार जब आप string को modify करते हैं, तो नया object बनता है:

```
s = "Hello" s += " World" # नया object create होता है
```

✦ Optimization: इसीलिए heavy string operations में performance degrade हो सकता है। बेहतर है:

```
# Bad (slow) s = "" for i in range(1000): s += str(i) # Good (fast) s = ''.join([str(i) for i in range(1000)])
```

---

## ◆ Summary Table

Feature	Description
Type	<code>str</code> (immutable sequence of Unicode characters)
Mutability	Immutable
Interning	Yes, for short and simple strings
Memory Structure	PyUnicodeObject

---

Copy Behavior	New object created when modified
Reference Count	Handled by CPython
Garbage Collection	Yes, using reference counting and cyclic GC
Encoding	Internally stored as Unicode

## ◆ Diagram: String Reference in Memory

```
a = "hello"
```

```
+-----+          +-----+
| Variable a| ----> | PyUnicodeObject |
+-----+          | value: "hello"   |
                    | id: 0x7f123456   |
                    +-----+
```

When `a = "world"` is assigned:

```
a = "world"
```

```
+-----+          +-----+
| Variable a| ----> | PyUnicodeObject |
+-----+          | value: "world"   |
                    +-----+
```

## ◆ Bonus: How to Explore Internals Yourself

You can use these modules to inspect:

```
import sys
s = "hello"
print(sys.getrefcount(s)) # Reference count
print(sys.intern(s)) # Interning forcefully
```