How Much Memory Does Python Allocate For Each Data Type?

Understanding Memory Usage In CPython

Agenda Items

- Introduction-Why Memory Matters In Python
- Cpython Memory Allocation – Small VS Larg Objects
- Internal Structure Of Python Objects
- Measuring Memory – Shallow VS Deep Size
- Memory Usage Of Basic Data Types
- Memory Usage Of Collections
- Why Memory Sizes Vary
- Conclusion

Introduction

Python uses an object-based memory model, which means every value is stored as an object.

Each object contains:

Metadata (like reference count)

Type information

The actual value

This design affects how much memory each data type consumes.

How Cpython Allocates Memory

CPython uses a layered memory allocation system:

Small objects (≤ 512 bytes) are allocated using pymalloc.

Large objects are allocated using the system's malloc.

Small objects are stored in pools and arenas to improve speed and reduce fragmentation.

Internal Structure Of a Python Object

- Every Python object begins with PyObject_HEAD, which includes:

1. Reference count
2. Pointer to the object's type

- After that, the actual value of the object is stored (e.g., integer, string, references to list items).

Measuring Memory : Shallow Size

sys.getsizeof(object) returns the shallow size of the object only.

This includes the object’s structure, but not the memory of the objects it references (e.g., items in a list or dictionary).

Deep Size

The deep size of an object includes:

The object itself

All objects it references, recursively

This gives a more accurate estimate of real memory usage for structures like lists, sets, and dictionaries.

Memory Usage For Basic Data Types

- Approximate memory usage in CPython (64-bit):

❖ int → ~28 bytes

❖ float → ~24 bytes

❖ bool → similar to int

str → 49 bytes minimum + 1 byte per character

- Values can vary depending on Python version and system architecture.

Memory Usage for Collections

- Collections require additional space for storing pointers and internal structures:

❖ list → base size + 8 bytes per element

❖ tuple → smaller than list but still stores pointers

❖ dict → uses hash tables; a small dict may consume ~280 bytes

❖ set → similar hashing structure; large overhead

Why Memory Sizes Vary

- Different data structures have different implementations:

❖ list → dynamic array

❖ dict → hash table

❖ set → hashed storage

❖ str → Unicode object

- This directly affects memory consumption.

Conclusion

Python stores every value as an object with metadata.

sys.getsizeof() reports only shallow size.

Deep size includes nested objects and gives the real memory usage.

Understanding Python’s memory model helps optimize performance and reduce memory usage.

Thank You

My Linkedin
My GitHub