

Python's memory management involves a private heap that holds all Python objects and data structures. The Python memory manager is responsible for managing this private heap internally. The Python memory manager has different components which deal with various dynamic storage management aspects, like sharing, segmentation, preallocation or caching. At the lowest level, a memory allocator ensures that there is enough space in the private heap for storing all Python-related data by interacting with the memory manager of the operating system. On top of the raw memory allocator, several object-specific allocators operate on the same heap and implement distinct memory management policies adapted to the peculiarities of every object type. For example, integer objects are managed differently within the heap than strings, tuples or dictionaries because integers imply different storage requirements and speed/space trade offs. The Python memory manager thus delegates some of the work to the object-specific allocators, but ensures that the latter operate within the bounds of the private heap. It is important to understand that the management of the Python heap is performed by the interpreter itself and that the user has no control over it, even if they regularly manipulate object pointers to memory blocks inside that heap. The allocation of heap space for Python objects and other internal buffers is performed on demand by the Python memory manager through the Python/C API functions.