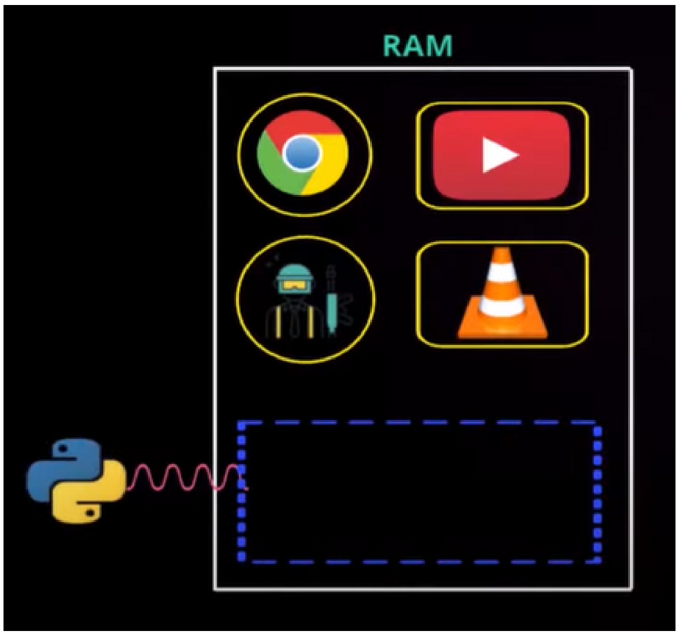
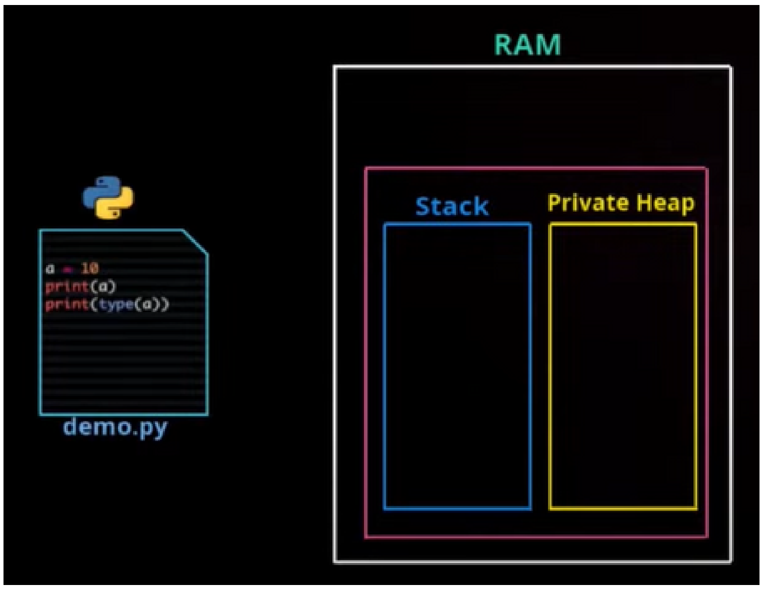
Memory Management in Python

Python is an object-oriented programming language that uses memory pointers under the hood for object reference. In Python, a pointer is a variable that stores the memory address of another variable or object. When you create a variable in Python, you are actually creating a pointer to an object in memory.

Python programs execute within a dedicated portion of the computer's memory, which is specifically allocated to the Python interpreter. This means that Python does not use the entire available memory of the computer, but rather a specific portion of it that is reserved for the interpreter to execute Python code.





```
a = 10
print(a)
print(type(a))
```

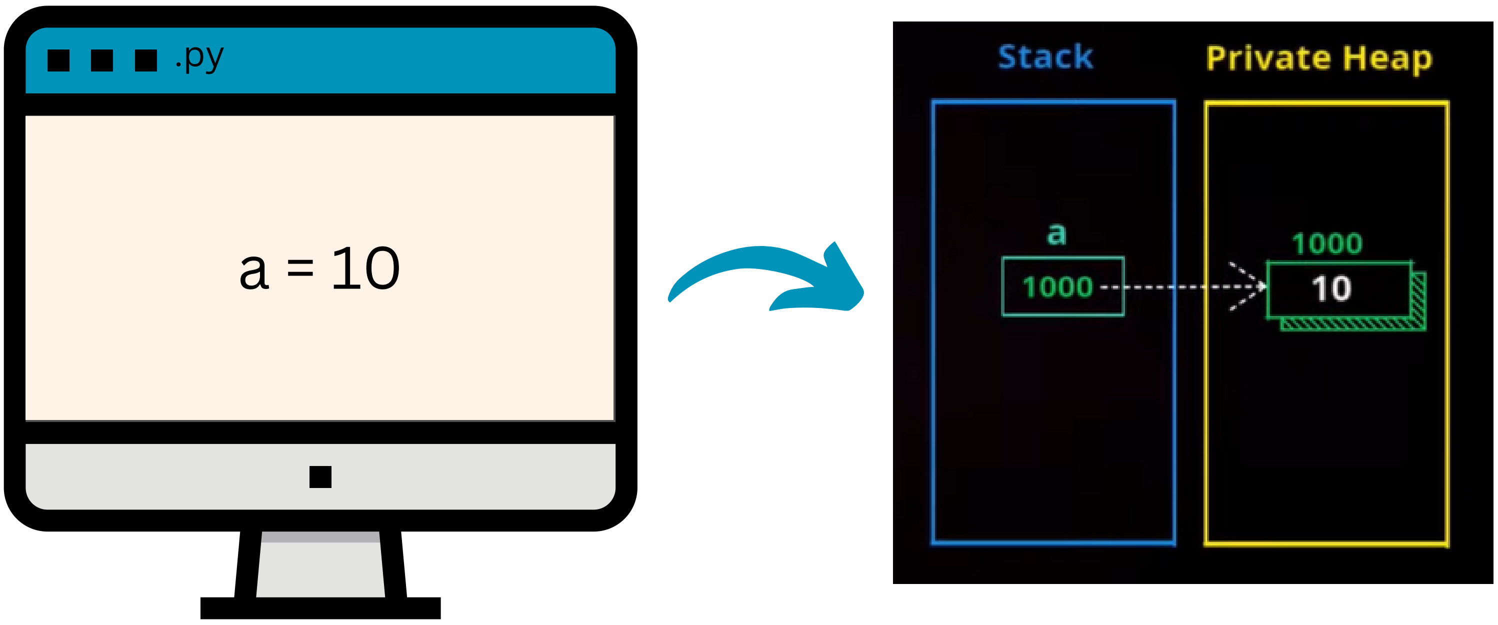
The dedicated portion of memory allocated for a running Python program is divided into two distinct regions:

- **THE STACK**
- **THE PRIVATE HEAP.**

- **Stack Memory:** The stack is a region of memory used to store *local variables and function calls*. It is managed by the interpreter and is limited in size. When a function is called, a new stack frame is created to store local variables and function arguments. When the function returns, the stack frame is destroyed, and the memory is released.

- **Heap Memory:** The heap is a region of memory used to store *objects and data structures*. It is managed by the Python interpreter's memory manager. When you create an object in Python, it is allocated on the heap. The memory used by the object is automatically managed by the interpreter's memory manager.

For Example -



```
a = 10
```

In Python, objects such as the integer 10 are stored in the private heap region of memory, with a unique address such as 1000. References or names in Python, on the other hand, are always stored in the stack region of memory. These references contain the memory address of the object they point to, allowing them to access and manipulate the object's data stored in the private heap.