Jan 8, 2025

LEC 01 - Python Memory Model

Key Terms:

Object: An instance of a data structure that contains data. An object has a unique ID (ex. ID 148), a type (ex. Str), and a value (ex. “Hi”). Everything in Python is constructed as an object

Id: A unique identifier for an object in memory, accessible using the `id()` function

Type: The kind of object, which defines the types of operations that can be performed on the object. Examples are str, int, bool, etc.

Value: The data stored in an object, such as `42`, `"Hello"`, and `False`

Refers to: Describes the relationship between a variable and the object it points to. For example, the statement `x = 10` states that the variable `x` refers to an integer object of some ID, of type int, which has a value of 10

Assignment statement: Used to make a variable refer to an object, such as `x = 10`

Alias: Occurs when 2 or more variables both refer to the same object in memory, typically with mutable objects

Immutable / Mutable: An object is immutable, its value can’t be changed after creation. Examples include int, str, bool, and tuple. A mutable object can have its value changed after creation, such as list, and dict.

Mutate: Modifying the contents of a mutable objects, such as `lst.append(3)`

Reassign: Updating the object that a variable refers to, such as `x = 10` followed by `x = 20`

Practice Problem:

What is wrong with the following code:

```
lst = [1, 2, 3]
for item in lst:
      item = item + 1
      print(lst)
```

You can't reassign list items by modifying the loop variable

Jan 8, 2025

LEC 01 - Python Memory Model

Python Memory Model - Functions:

Key Terms:

Frame: Data about a function call (including local variables)

Call Stack: A collection of the frames of the function that are currently running

How function calls are tracked:

- Each call to a function creates a *frame* that stores the needed info, such as arguments
- As one function calls another, these frames pile up in a sack
- Called the *call stack*

Function calls:

1. A new frame is added to the top of the call stack
2. Each parameter is defined in the frame, with no values yet
3. Each argument is evaluated and its value is assigned to the respective parameter
4. The function body is executed
5. The frame stores any variables made during the function run, these are local variables

- When a function is called, its parameters become aliases of the arguments passed to the function call

---

Memory Models:

- The left side shows the call stack, whose frames show the programs variables and which ids they store
- The right side shows the object space, where all objects are shown with their respective id, type, and value
- Double boxes indicate immutability, while single boxes indicate mutability

---

Example 1:

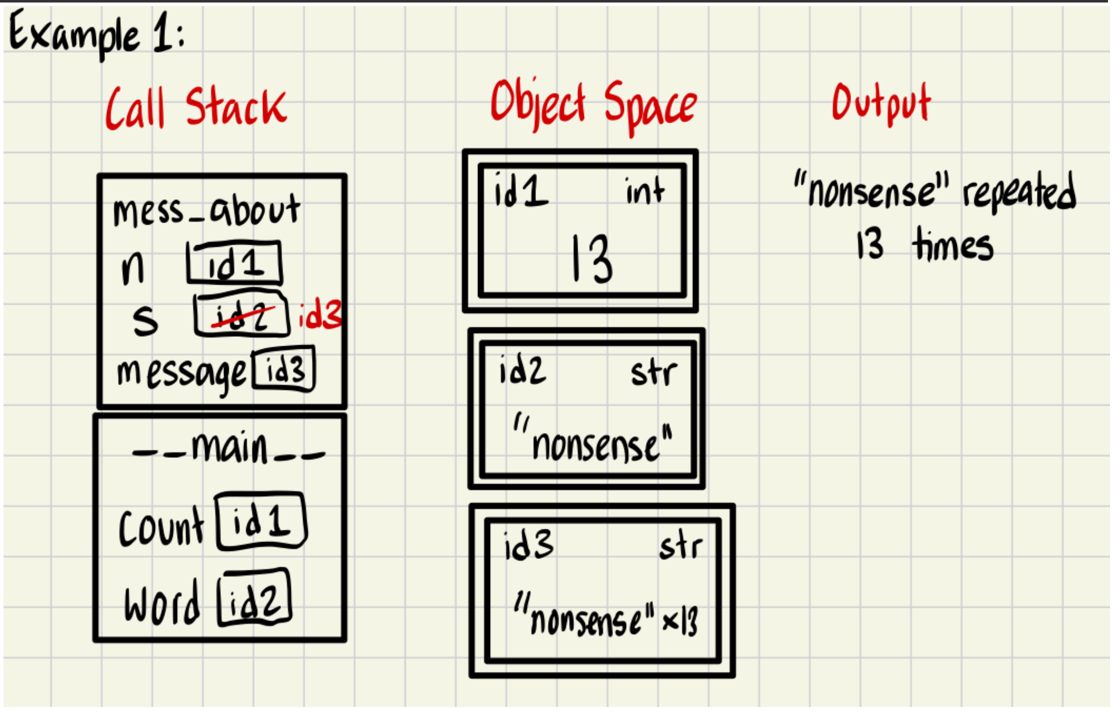
```python
def mess_about(n: int, s: str) -> None:
      message = s * n
```

Jan 8, 2025

LEC 01 - Python Memory Model

```
        s = message
        print(s)

if __name__ == '__main__':
        count = 13
        word = 'nonsense'
        mess_about(count, word)
        print(word)
```



Example 1:
Call Stack
Object Space
Output
mess_about
n id1
s ~~id2~~ id3
message id3
__main__
count id1
word id2
id1 int 13
id2 str "nonsense"
id3 str "nonsense" ×13
"nonsense" repeated 13 times