Collection names are pointers?

August 23, 2021

1 General explanation

The identifier of a collection data type is a pointer to the collection. I don't know the underlying structure of collection data types in Python, so I cannot exactly pinpoint what exactly do the identifiers point to. The first element? The hash table for the collection?

This is not immediately apparent in Python as it is in C, for example, since the name of a collection gets automatically presented as the contents of the collection. However, the following demontrations with mutable collection data types indicate this property…

The demonstrations show that even if you pass a mutable collection's name as an argument to a function or assign the collection's name's value to another identifier, any changes you make in the other identifier (function argument or duplicate) will reflect in the original collection itself, since each identifier points to the same memory address.

2 Passing collection name as argument

[3]:
```python
def addItem(lst, item):
    lst.append(item)
l = [1, 2, 3]
addItem(l, "new")
print(l)
```

```
[1, 2, 3, 'new']
```

Here, by passing l, you are duplicating not the original list for the argument lst, but rather the memory address of that list. Hence, while l and lst by themselves are stored in different memory addresses (as they store the pointer and not the actual list and its values), they point to the same address

3 Working with collection name's duplicate

[8]:
```python
l1 = [10, 9, 8]
l2 = l1
l2.append(2)
print(l1)
```

```
[10, 9, 8, 2]
```

Here, by assigning l2 to l1, you are duplicating not the original list, but rather the memory address of that list. Hence, while l1 and l2 by themselves are stored in different memory addresses (as they store the pointer and not the actual list and its values), they point to the same address.

4 Notes

4.1

I have only demonstrated for lists, but the above can be demonstrated for any mutable collection data type, including data types such sets, dictionaries and tuples...

[16]:
```
s1 = {10, 9, 8}
s2 = s1
s2.remove(10)
print(s1)
```

```
{8, 9}
```

[19]:
```
d1 = {"a":1, "b":2}
d2 = d1
d2.update({"x":-1, "y":-2})
print(d1)
```

```
{'a': 1, 'b': 2, 'x': -1, 'y': -2}
```

4.2

I have used only mutable collection data types for the demostrations, but I think I can safely assume that the fact that the name of a collection is in fact the pointer to the collection applies to all collection data types, mutable or not, and ordered or not.