Shallow and Deep Copying Report

Mahmoud Fathy Mohamed

1 Difference Between Shallow and Deep Copying

In Python, assignment statements create references to the same object rather than copying it. Python provides the copy module to create actual copies which offer functions for shallow (copy.copy()) and deep (copy.deepcopy()) copies.

1.1 Shallow Copy

A shallow copy only copies the outer structure and references the nested objects. So, when a list is copied, any change on its nested objects also affects the original list but a change on its outer structure does not, as seen in the example.

Shallow Copy Example

```
import copy

a = [[1, 2, 3], [4, 5, 6]]

# Creating a shallow copy of the nested list 'original'
b = copy.copy(a)

# Modifying an element in the shallow-copied list
b[0][0] = 99
b.append([7, 8, 9])

# Printing the original and shallow-copied lists
print(a)
print(b)
```

Output

```
[[99, 2, 3], [4, 5, 6]]
[[99, 2, 3], [4, 5, 6], [7, 8, 9]]
```

1.2 Deep Copy

Deep copying constructs a new collection object and then recursively populates it with copies of the child objects found in the original. This means that any changes made to a copy of the object do not reflect in the original object.
Nested elements of the original list a are recursively duplicated to ensure that even deeply nested objects are entirely independent in the copied list.

Deep Copy Example

Here is an example of a deep copy. Changes to the nested list in the copy do not affect the original.

```
import copy

a = [[1, 2, 3], [4, 5, 6]]

# Creating a deep copy of the nested list 'a'
b = copy.deepcopy(a)

# Modifying an element in the deep-copied list
b[0][0] = 99

print(a)
print(b)
```

Output

```
[[1, 2, 3], [4, 5, 6]]
[[99, 2, 3], [4, 5, 6]]
```