Report 1: Shallow Copy vs Deep Copy in Python

Introduction

In Python programming, copying data structures is a common task when working with lists, dictionaries, objects, or other collections. Copying can be performed in two main ways: shallow copy and deep copy. Understanding the difference between them is crucial to avoid unexpected behavior in programs, especially when working with nested or mutable objects.

Shallow Copy

A shallow copy creates a new container object but does not recursively copy the elements within it. Instead, it stores references to the same objects found in the original container. As a result, if the original or the copy is modified at the nested level (for example, an element inside a list of lists), the changes will be reflected in both objects. In Python, a shallow copy can be created using the copy.copy() function from the built-in copy module, or by using certain built-in methods such as list slicing (list[:]) in some cases.

Deep Copy

A deep copy creates a completely independent duplicate of the original object and all the objects it contains, recursively. This means that nested mutable elements are also duplicated rather than referenced. Any modifications made to the deep copy will not affect the original object. A deep copy is created using the copy.deepcopy() function in Python.

Conclusion

Shallow copying is faster but may cause unintended side effects when working with mutable nested objects, as both copies share the same inner references. Deep copying eliminates this issue by fully duplicating all elements, ensuring independence between the original and the copy. Choosing between them depends on whether shared references are acceptable for the given application.