Table of Contents

In Python, a **shallow copy** is a new object that is a **copy of the original object**, but it **does not create copies of nested objects** (objects inside objects). Instead, it only copies references to those inner objects.

---

1. ☑ What is a Shallow Copy?

- It copies the outer object.
- Inner elements (like lists or dictionaries inside a list) are **not copied**, only their **references** are copied.
- Changes to nested objects in the copied object **affect the original**.

---

2. 📌 How to Create a Shallow Copy

1. **Using `copy()` method (for lists, dicts, etc.):**

```
original = [[1, 2], [3, 4]]
shallow = original.copy()
```

2. **Using the `copy` module:**

```
import copy
shallow = copy.copy(original)
```

---

3. 🔍 Example:

```
import copy

original = [[1, 2], [3, 4]]
shallow = copy.copy(original)

shallow[0][0] = 100

print("Original:", original)  # [[100, 2], [3, 4]]
print("Shallow:", shallow)    # [[100, 2], [3, 4]]
```

☑ The outer list is copied. ❌ The inner lists are **shared**, so modifying `shallow[0][0]` changes `original[0][0]`.

Here's a line-by-line explanation of the code:

◇ Code:

```
import copy
```

☑ This imports Python's built-in `copy` module, which provides functions to perform shallow and deep copies.

```
original = [[1, 2], [3, 4]]
```

☑ A list named `original` is created. It contains **two inner lists**, making it a nested list (a list of lists).

```
shallow = copy.copy(original)
```

☑ This creates a **shallow copy** of `original` and stores it in `shallow`.

- A **new outer list** is created.
- The **inner lists are not copied**; only their **references** are copied.

So now:

- `original` and `shallow` are two different outer lists.
- But `original[0]` and `shallow[0]` point to the **same inner list** `[1, 2]`.

```
shallow[0][0] = 100
```

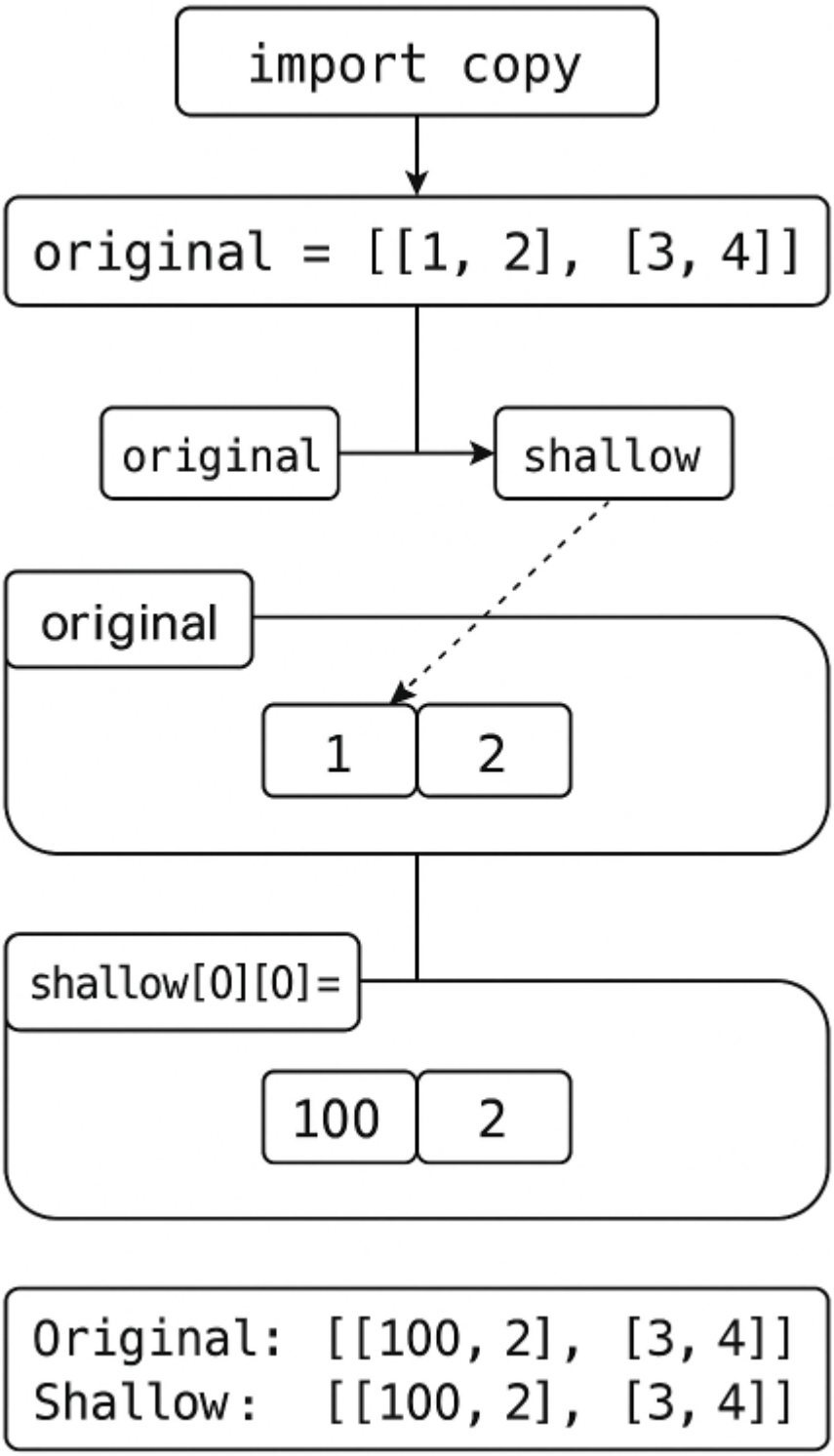
☑ This changes the first element of the first inner list **through the shallow copy**.

Since the inner list `[1, 2]` is **shared** between `original` and `shallow`, the change affects both.

```
print("Original:", original)  # [[100, 2], [3, 4]]
print("Shallow:", shallow)    # [[100, 2], [3, 4]]
```

☑ Output shows that **both `original` and `shallow` are affected**:

```
Original: [[100, 2], [3, 4]]
Shallow:  [[100, 2], [3, 4]]
```



🔍 Summary:

- `copy.copy()` only copies the outer list.
- The inner lists are **shared** between `original` and `shallow`.
- Changes to inner lists in the copy will **affect the original**.

---

4. ☑ When to Use Shallow Copy

Use shallow copy when:

- You want a new outer container.
- You don't plan to modify nested objects separately.

📚 Related Topics

- **What is a Deep Copy in Python?** – A deep copy creates a completely independent copy of an object and all nested objects inside it. 👉 Learn more