

What is a Shallow Copy?


A shallow copy creates a new outer object, but it doesn't copy the inner objects — it reuses references.

Think of it like this:

"Make a new box (list), but just point to the same items inside."

Flat List Example (no nested structures):


python

 Copy  Edit


```
a = [1, 2, 3] b = a[:] # shallow copy
```

- `a` and `b` are **different lists**.
- But since `1`, `2`, and `3` are **immutable** numbers, it doesn't matter if they're shared.
- Changing `a[0]` to a new number won't affect `b`.

python

 Copy  Edit

```
a[0] = 99 print(b) # Output: [1, 2, 3] → b is unchanged
```

 This is **safe** for flat lists with immutable values (like `int`, `str`, `bool`).

Nested List Example:

python

 Copy  Edit

```
a = [[1, 2], [3, 4]] b = a[:] # shallow copy a[0][0] = 99 print(b) # Output:  
[[99, 2], [3, 4]] !changed!
```

Here, the **outer list is new**, but the **inner lists are shared**, so modifying them affects both.

Now, Your Screenshot Example

python

 Copy  Edit

```
h1 = [1, 2, 3] h2 = h1[:] # ← shallow copy of h1 h1[0] = 77 # ← modifies h1 only
```

What Happens:

Variable	Description	Value
h1	Original list	[77, 2, 3]
h2	Shallow copy of h1	[1, 2, 3]

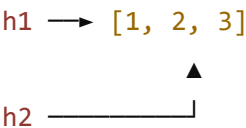
- `h2 = h1[:]` creates a **new list**.
- Since `1`, `2`, and `3` are **immutable**, they are fine to copy.
- When you do `h1[0] = 77`, it **only changes** `h1`, not `h2`.

Visualization:

Before:

CSS

 Copy  Edit



After:

CSS

 Copy  Edit

```
h1 → [77, 2, 3]
h2 → [1, 2, 3] (remains unchanged)
```

✓ Summary:

- ✓ Shallow copy: outer list is copied, inner references are shared.
- ✓ Works safely for flat lists with immutable values (like your example).
- ⚠ Be careful with nested lists — inner elements will be **shared** unless you use `deepcopy`.

Let me know if you'd like to test this with a **nested structure** example!



new outer object, but it doesn't copy the inner objects

Great! Let's focus on that key phrase:

"Shallow copy creates a new outer object, but it doesn't copy the inner objects."

Let me explain this clearly with a **nested list example** — this is where shallow copy behavior really matters.



🔄 Example:

python

Copy Edit

```
original = [[1, 2], [3, 4]] shallow = original[:] # shallow copy
```

What happened here?

-  `shallow` is a **new list** (new outer object).
-  But the **inner lists** `[1, 2]` and `[3, 4]` are **not copied**.
- Instead, the **same inner lists** are used in both `original` and `shallow`.

Now, modify an inner object:

python

 Copy  Edit

```
original[0][0] = 99
```

Since both `original[0]` and `shallow[0]` point to the **same inner list**, this change affects both:

python

 Copy  Edit

```
print(original) # [[99, 2], [3, 4]] print(shallow) # [[99, 2], [3, 4]] !
```

Visualization:

SCSS

 Copy  Edit

```
original → [ [1, 2], [3, 4] ]
           ↑   ↑
shallow  ——— (same inner lists)
```

So when you modify an inner list (like `original[0][0]`), both `original` and `shallow` see the change.

 If you want **everything** copied (outer + inner), use:

python







 Copy  Edit

```
import copy
deep = copy.deepcopy(original)
```

Now `deep` is a **fully independent** copy — changes in `original` won't affect it at all.



Summary:

Copy Type	Outer Object New?	Inner Objects New?	Changes Affect Original?
Shallow Copy	 Yes	 No	 Yes (if mutable inside)
Deep Copy	 Yes	 Yes	 No