Shallow copy and deep copy

- When we copy lists in Python, it may look simple, but the way Python stores that data can completely change the result.
- Especially when our lists contain other lists. That's where shallow copy and deep copy come in.

Once we understand the difference, we can avoid unexpected changes in our data and write Python code with much more confidence.

Shallow Copy:

If the list contains another list inside it, that inner list is shared, not duplicated(it means Python does **not** make a new inner list for the copy).
So changing nested data in the copy will also change the original list.

For example:

```
"""
In this example:

Outer list  → [1, 2, [2, 3]]
Inner list  → [2, 3]

A shallow copy only creates a new OUTER list,
but the INNER list is still shared between 'a' and 'b'.

So if we change the inner list in 'b',
the same change will appear in 'a' because both point
to the same inner list.
"""
a = [1,2,[2,3]]
b = a.copy()

# We change the inner list in 'b'
# Because shallow copy shares the inner list,
# this change will also appear in the original list 'a'.

b[2][1] = 7
print(a)
print(b)
```

Output:

```
[1, 2, [2, 7]]
[1, 2, [2, 7]]
```

- The inner list is shared between `a` and `b`, so changing it in `b` also updates it in `a`.
- That's why both lists show the modified value `[2, 7]` in the inner list.
- Both lists show [2, 7] because they share the same inner list."

Deep copy:

A deep copy creates a completely new list, including new copies of all inner lists.
Nothing is shared with the original.

So if you change something in the deep-copied list, the original list stays the same.

For Example:

```python
from copy import deepcopy

# In this list:
# a = [1, 2, [2, 3]]
# Outer list  → [1, 2, [...]]
# Inner list  → [2, 3]
#
# Deep copy creates a NEW outer list AND a NEW inner list.
# Nothing is shared between 'a' and 'c'.
#
# So any change inside c's inner list will NOT affect a.

a = [1, 2, [2, 3]]
c = deepcopy(a)

c[2][1] = 7   # Changing the inner list in 'c' DOES NOT change 'a'

print(a)
print(c)
```

Output:

```
[1, 2, [2, 3]]
[1, 2, [2, 7]]
```

✓ **Deep copy creates a completely separate inner list**, so changes made inside `c` do NOT affect `a`.

✓ That's why `a` stays `[1, 2, [2, 3]]` while `c` becomes `[1, 2, [2, 7]]` after the update.

Why do we need shallow copy and deep copy?

When we're working with Python lists, especially lists that contain other lists, copying isn't always as simple as it looks. If we don't choose the right type of copy, we may accidentally change our original data without realising it.

- A **shallow copy** may still share inner lists with the original, which can lead to unexpected changes.
- A **deep copy** avoids this by creating a completely separate copy, ensuring the original data stays safe.

What happens if we don't use shallow copy and deep copy correctly?

- You might change something in the copied list… and the original list changes with it.
- Your data becomes "linked" without you realising it, causing unexpected behaviour.
- Small changes can create big bugs that are hard to find.
- Your program may behave unpredictably because two lists are linked behind the scenes.
- It becomes confusing to work with your variables; you update one, and another changes, too.