Part 1 — Shallow Copy vs Deep Copy (With Memory Models)

Example 1 — Shallow Copy (Flat List)

```
a = [1, 2, 3]
b = a.copy()
```

```
Memory After Copy:

a -----> [ 1 , 2 , 3 ]
b -----> [ 1 , 2 , 3 ]

Both lists are different outer objects.
The integers are shared references.
```

```
b[0] = 9
```

```
Memory After Reassignment:

a -----> [ 1 , 2 , 3 ]
b -----> [ 9 , 2 , 3 ]

We reassigned b[0].
The integer 1 was not modified.
The container slot changed to point to 9.
```

Example 2 — Shallow Copy (Nested List)

```
a = [[1,2],[3,4]]
b = a.copy()
```

```
Memory After Copy:

a -----> [ L1 , L2 ]
b -----> [ L1 , L2 ]

L1 ----> [1,2]
L2 ----> [3,4]

Outer list copied.
Inner lists shared.
```

```
b[0][0] = 99
```

```
Memory After Mutation:
```

```
a -----> [ L1 , L2 ]
b -----> [ L1 , L2 ]

L1 ----> [99,2]

We mutated L1.
Both a[0] and b[0] reference L1.
Therefore both appear changed.
```

Example 3 — Deep Copy

```
import copy

a = [[1,2],[3,4]]
b = copy.deepcopy(a)
```

```
Memory After Deep Copy:

a -----> [ L1 , L2 ]
b -----> [ L3 , L4 ]

L1 ----> [1,2]
L2 ----> [3,4]

L3 ----> [1,2]
L4 ----> [3,4]

Everything duplicated recursively.
No shared inner objects.
```

Part 2 — Default Arguments (Call-by-Call Memory Model)

```
def add_item(item, bucket=[]):
    bucket.append(item)
    return bucket
```

```
At Definition Time:

Python creates ONE list object D.

add_item.__defaults__ ----> ( D )
D ----> []


Call 1:
```

```
add_item(1)

No bucket supplied.
Python binds:

bucket ----> D

After append:

D ----> [1]


Call 2:
add_item(2)

No bucket supplied.
Python again binds:

bucket ----> D

After append:

D ----> [1,2]

Same object reused.
```

Now Contrast With Supplying a Bucket

```
add_item(1, [1,2,3])

Caller supplies a new list B.

Python binds:

bucket ----> B

Default D is NOT used.

After append:

B ----> [1,2,3,1]
D ----> []


add_item(2)

No bucket supplied.

Python binds:

bucket ----> D
```

After append:

D ----> [2]

Default used only when argument omitted.