Tasks – Variables vs Lists, Cutting Lists (Slicing), and Real Copies [:]

Understand: **variables vs lists** (assignment with lists shares the same list); **cutting lists** with **slicing** `[start:stop]`, `[:stop]`, `[start:]`, `[::step]`; and **real copies** with **[:]** or **list()**. Do the list labs (18, 19) first. Create each file, run it, and check the output.

Run scripts with: `python3 script_name.py`

Part 1 – Variables (numbers) vs lists

Task 1.1 – Numbers: assignment copies the value (`var_num_copy.py`)

- Create `var_num_copy.py`. Assign `a = 10` and then `b = a`. Change `a` to 20: `a = 20`. Print `a` and `b`. You should see 20 and 10. With numbers, `b = a` copies the value, so changing `a` later does not change `b`.

Expected output:

```
20
10
```

Task 1.2 – Lists: assignment does NOT copy (`list_same_reference.py`)

- Create `list_same_reference.py`. Assign `a = [1, 2, 3]` and then `b = a`. Change the first element: `a[0] = 99`. Print `a` and `b`. You should see [99, 2, 3] for **both**. With lists, `b = a` does not copy the list; `b` and `a` refer to the same list, so changing an element through one affects the other.

Expected output:

```
[99, 2, 3]
[99, 2, 3]
```

Task 1.3 – Modify through "b", see "a" change (`list_shared_modify.py`)

- Create `list_shared_modify.py`. Assign `original = [10, 20, 30]` and `other = original`. Use `other.append(40)` and then `other[0] = 0`. Print `original` and `other`. Both should show [0, 20, 30, 40]. So "other" is just another name for the same list.

Expected output:

```
[0, 20, 30, 40]
[0, 20, 30, 40]
```

Part 2 – Cutting lists (slicing)

Slicing **cuts** a list: `list[start:stop]` gives elements from index `start` up to (but not including) `stop`. Omitted start means from the beginning; omitted stop means to the end. `list[::step]` uses a step (e.g. every 2nd element).

Task 2.1 – Slice [start:stop] (`list_slice_cut.py`)

- Create `list_slice_cut.py`. Assign `nums = [10, 20, 30, 40, 50, 60]`. Create `middle = nums[2:5]` (indices 2, 3, 4). Print `middle`. Result: [30, 40, 50]. The stop index 5 is not included.

Expected output:

```
[30, 40, 50]
```

Task 2.2 – First N and last N (`list_slice_first_last.py`)

- Create `list_slice_first_last.py`. Assign `data = [1, 2, 3, 4, 5, 6, 7]`. Print `data[:3]` (first 3 elements) and `data[-3:]` (last 3 elements). Use two print statements.

Result: [1, 2, 3] and [5, 6, 7].

Expected output:

```
[1, 2, 3]
[5, 6, 7]
```

Task 2.3 – From index to end (`list_slice_from.py`)

- Create `list_slice_from.py`. Assign `items = ["a", "b", "c", "d", "e"]`. Create `tail = items[2:]` (from index 2 to the end). Print `tail`. Result: ["c", "d", "e"].

Expected output:

```
['c', 'd', 'e']
```

Task 2.4 – Step: every 2nd element (`list_slice_step.py`)

- Create `list_slice_step.py`. Assign `nums = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]`. Use `evens = nums[::2]` (start and stop omitted, step 2). Print `evens`. Result: [0, 2, 4, 6, 8]. Then use `odds = nums[1::2]` and print. Result: [1, 3, 5, 7, 9].

Expected output:

```
[0, 2, 4, 6, 8]
[1, 3, 5, 7, 9]
```

Task 2.5 – Slice is a new list (`list_slice_new_list.py`)

- Create `list_slice_new_list.py`. Assign `a = [1, 2, 3, 4, 5]`. Set `b = a[1:4]` (a slice/cut). Change `a[2] = 99`. Print `a` and `b`. Result: a is [1, 2, 99, 4, 5]; b is [2, 3, 4]. So the slice created a **new** list; changing a does not change b.

Expected output:

```
[1, 2, 99, 4, 5]
[2, 3, 4]
```

Part 3 – Real copy with [:]

Task 3.1 – Copy with [:] (`list_slice_copy.py`)

- Create `list_slice_copy.py`. Assign `a = [1, 2, 3]`. Create a real copy: `b = a[:]`. The slice `a[:]` means "all elements" and creates a **new** list. Change `a[0] = 99`. Print `a` and `b`. You should see [99, 2, 3] and [1, 2, 3]. So `b` is unchanged.

Expected output:

```
[99, 2, 3]
[1, 2, 3]
```

Task 3.2 – Copy with [:], then modify the copy (`list_slice_modify_copy.py`)

- Create `list_slice_modify_copy.py`. Assign `original = [5, 10, 15]`. Set `copy = original[:]`. Append 20 to `copy` and change `copy[0] = 0`. Print `original` and `copy`. Original should still be [5, 10, 15]; copy should be [0, 10, 15, 20].

Expected output:

```
[5, 10, 15]
[0, 10, 15, 20]
```

Task 3.3 – Sort the copy, keep original order (`list_copy_then_sort.py`)

- Create `list_copy_then_sort.py`. Assign `nums = [30, 10, 20]`. Create a copy with `sorted_list = nums[:]`. Call `sorted_list.sort()`. Print `nums` and `sorted_list`.

Original should still be [30, 10, 20]; sorted_list should be [10, 20, 30]. So you can sort a copy without changing the original.

Expected output:

```
[30, 10, 20]
[10, 20, 30]
```

Part 4 – Real copy with list()

Task 4.1 – Copy with list() (`list_copy_list_func.py`)

- Create `list_copy_list_func.py` . Assign `a = [1, 2, 3]` . Create a copy: `b = list(a)` . Change `a[0] = 99` . Print `a` and `b` . You should see [99, 2, 3] and [1, 2, 3]. So `list(a)` also creates a new list with the same elements.

Expected output:

```
[99, 2, 3]
[1, 2, 3]
```

Task 4.2 – [:] vs list() both work (`list_copy_both.py`)

- Create `list_copy_both.py` . Assign `data = [7, 8, 9]` . Set `copy1 = data[:]` and `copy2 = list(data)` . Change `data[1] = 0` . Print `data` , `copy1` , and `copy2` . data should be [7, 0, 9]; copy1 and copy2 should both still be [7, 8, 9]. So both methods make a real copy.

Expected output:

```
[7, 0, 9]
[7, 8, 9]
[7, 8, 9]
```

Part 5 – When to use a copy

Task 5.1 – Reverse a copy, keep original (`list_reverse_copy.py`)

- Create `list_reverse_copy.py`. Assign `items = [1, 2, 3, 4, 5]`. Create `reversed_copy = items[:]`, then call `reversed_copy.reverse()`. Print "Original:" and items, then "Reversed copy:" and reversed_copy. Original stays [1, 2, 3, 4, 5]; reversed_copy is [5, 4, 3, 2, 1].

Expected output:

```
Original: [1, 2, 3, 4, 5]
Reversed copy: [5, 4, 3, 2, 1]
```

Task 5.2 – No copy: "backup" that isn't real (`list_fake_backup.py`)

- Create `list_fake_backup.py`. Assign `data = [10, 20, 30]` and `backup = data` (no copy). Then do `data.append(40)` and `data[0] = 0`. Print "data:" and data, "backup:" and backup. Both show [0, 20, 30, 40]. So "backup" was not a real backup. Then in a comment or a second run idea: to make a real backup you would use `backup = data[:]` before changing data.

Expected output:

```
data: [0, 20, 30, 40]
backup: [0, 20, 30, 40]
```

Task 5.3 – Real backup with [:] (`list_real_backup.py`)

- Create `list_real_backup.py`. Assign `data = [10, 20, 30]` and `backup = data[:]` (real copy). Then do `data.append(40)` and `data[0] = 0`. Print "data:" and data, "backup:" and backup. data should be [0, 20, 30, 40]; backup should still be [10, 20, 30]. So backup is unchanged.

Expected output:

```
data: [0, 20, 30, 40]
backup: [10, 20, 30]
```

Done

You've seen: **variables** vs **lists** (assignment shares the same list); **cutting lists** with **slicing** `[start:stop]`, `[:n]`, `[n:]`, `[::step]` (slices create new lists); **real copy** with `[:]` or `list()`. Use a copy when you want to modify or sort one list without changing the other.