IS1111 Tutorial Assignment 3 – Lists, Functions & Comprehensions

Overview

This assignment gives you more practice with lists and functions, and introduces slightly more advanced list comprehension patterns. You will also work with lists of lists (2D data) and use f-strings for formatting.

- Lists recap + deeper indexing/slicing patterns
- Copy vs alias (why changes sometimes "mysteriously" spread)
- Functions that process lists (cleaning, searching)
- List comprehensions (transform, filter, and simple if/else)
- Lists of lists (2D) with variable-length inner lists
- String formatting (f-strings, including optional : .2f)

---

1. Indexing & slicing patterns (warehouse packing)

A warehouse has a packing list of items (in order):

```
items = ["Keyboard", "Mouse", "Monitor", "Webcam", "Headset", "Dock", "Charger", "USB
Cable"]
```

(a) Print the FIRST two items using slicing.

(b) Print the LAST two items using slicing.

(c) Create a list called even_index_items containing items at indices 0, 2, 4, 6, ...

(d) Create a list called odd_index_items containing items at indices 1, 3, 5, 7, ...

(e) Split the list into TWO halves using slicing and len(...):

- first_half should contain the first half
- second_half should contain the second half

Hint: mid = len(items) // 2

(f) Rotate the list LEFT by 2 positions using slicing (no loops).

Example: [1, 2, 3, 4, 5] rotated left by 2 -> [3, 4, 5, 1, 2]

(g) Alias vs copy:

- Create alias_items = items
- Create copy_items = items[:]

- Change alias_items[0] to "ALIAS_CHANGED"
- Print items, alias_items, copy_items

You should see that items and alias_items changed together, but copy_items did not.

(h) Make a copy called items_edit, then replace the MIDDLE TWO items with ["Gift Card", "Sticker"] using slice assignment (no loops).

---

2. Functions + list methods + f-strings (product codes)

A technician is entering product codes, but the data is messy: extra spaces, inconsistent casing, blanks, and duplicates.

```
raw_codes = ["  kb-01 ", "KB-01", "ms-02", "", "  mon-10", "web-03 ", "MON-10", "   ",
"dock-07"]
```

(a) Write a function normalise_codes(raw_codes) that returns a NEW list that:

- strips spaces (strip)
- skips blanks (after strip, ignore "")
- uppercases the codes (upper)
- removes duplicates (case-insensitive)
- keeps the FIRST occurrence (preserve order)
- does NOT modify raw_codes
- do NOT use set(...)

Example output:

```
["KB-01", "MS-02", "MON-10", "WEB-03", "DOCK-07"]
```

(b) Write a function make_pick_list(codes) that returns ONE formatted string:

```
Pick List
---------
1) KB-01
2) MS-02
...
Total: 5
```

Rules:

- Use f-strings
- Do NOT use enumerate
- Use a normal counter variable instead (start at 1)
- Build a list of lines, then return "\n".join(lines)

(c) Write a function code_exists(codes, target) that:

- returns True if target exists in codes (case-insensitive)
- returns False if it does not exist
- do NOT return an index
- do NOT return -1

Example:
code_exists(["KB-01", "MS-02"], "ms-02") -> True
code_exists(["KB-01", "MS-02"], "mon-10") -> False

(d) Write a function remove_code_once(codes, target) that returns a NEW list
with the first matching target removed (case-insensitive).
If not found, return a copy of the original list.

Example:
remove_code_once(["KB-01", "MS-02", "MS-02"], "ms-02") -> ["KB-01", "MS-02"]

---

3. List comprehensions

Write the following functions using list comprehensions.
Reminder: [expression for item in items if condition]
You can also use a simple if/else inside the expression: [value_if_true if condition else value_if_false for item in items]

(a) clean_tags(tags)

- tags is a list of strings
- return a NEW list where each tag is stripped and lowercased, and blank strings are removed

Example: clean_tags([" Urgent ", " TODO", " ", "ReView"]) -> ["urgent", "todo", "review"]

(b) double_even(nums)

- nums is a list of ints
- return a NEW list where ONLY the even numbers are doubled

Example: double_even([1, 2, 3, 4, 5, 6]) -> [4, 8, 12]

(c) label_temperature(temps)

- temps is a list of ints (Celsius)
- return a NEW list where each temp becomes:
  "cold" if temp < 10
  "mild" if 10 <= temp < 20
  "warm" if temp >= 20

Hint: nested if/else is allowed, but keep it readable.
Example: label_temperature([6, 12, 19, 20, 25]) -> ["cold", "mild", "mild", "warm", "warm"]

(d) safe_to_float(values)

- values is a list of strings like [" 12.5", "x", " 7 ", "9.0", ""]
- return a NEW list of floats for the values that look like numbers

Rules: strip spaces, allow ONE dot

Hint: s.strip(); s.replace(".", "", 1).isdigit()

Example: safe_to_float([" 12.5", "x", " 7 ", "9.0", ""]) -> [12.5, 7.0, 9.0]

Challenge:

(e) extract_extensions(files)

- files is a list of strings like ["report.pdf", "photo.JPG", "README", "data.csv"]
- return a NEW list of file extensions (lowercased) for ONLY the filenames that contain a dot

Hint: name.split(".")[-1]

Example: extract_extensions(["report.pdf", "photo.JPG", "README", "data.csv"]) -> ["pdf", "jpg", "csv"]

4. Lists of lists + functions + formatting (daily sales)

We store shop sales as a list inside each record. Each shop can have a different number of days recorded (variable length!).

```
shops = [
    ["Cork",    [1200, 980, 1100]],
    ["Dublin",  [2100, 1990, 2500, 2300]],
    ["Galway",  [800, 760]],
    ["Limerick", [1500, 1400, 1550]],
]
```

(a) Print Dublin's FIRST day sale using indexing.

(b) Print Galway's LAST day sale using negative indexing.

(c) Add a new day sale 820 to Galway (update her sales list).

(d) Add a new day sale 2600 to Dublin (update his sales list).

(e) Add a new shop record: ["Waterford", [900, 920, 880]]

(f) Write a function average_sales(record)

- record is ONE shop record like ["Cork", [1200, 980, 1100]]
- return the average of the sales numbers
- do NOT hard-code the number of days
- assume there is at least 1 number

(g) Write a function classify_sales(avg)

Return:

- "Low" if avg < 1000
- "Medium" if 1000 <= avg < 1800
- "High" if avg >= 1800

(h) Write a function format_sales_report(shops)

- return ONE multi-line string like:

```
Sales Report
-----------
Cork: sales=[1200, 980, 1100] avg=1093.33 class=Medium
Dublin: sales=[...] avg=... class=High
...
```

Rules:

- Use f-strings
- Build a list of lines then join with "\n"
- Optional: format avg to 2 decimal places using {avg:.2f}

(i) Write a function top_shop(shops)

- return the NAME of the shop with the highest average
- if there's a tie, return the first one

(j) shops_with_min_days(shops, n)

- return a list of shop names who have AT LEAST n sales entries
- use a list comprehension