

Programming Techniques 2026 – Exam

Exam part I – A

19 October 2026

Duration: 75 minutes Total: 20 points

General instructions

- Complete all three exercises by writing clear and readable Python code, and make sure it is **well commented**.
 - You may use Python built-in functions, methods, and standard constructs covered in the course; do not use external libraries such as NumPy or Pandas not covered during the first part of the course.
 - Unless explicitly stated otherwise, you may create additional helper functions.
 - Your output must contain the intermediate and final results explicitly requested in each exercise.
 - **General requirement on input data:** Your solution must be general with respect to the input structures described in each exercise. The provided reference cases must be included in the submitted program to make results comparable, but your solution must not rely on their specific values or sizes. You are encouraged to test additional valid inputs and relevant edge cases.
-

Exercise 1 – Financial News Sentiment Monitor

[7 points]

A financial analyst wants a very simple text-based indicator of the tone of a set of headlines extracted from different digital newspapers.

Input structure

The input consists of a list of strings, where each string represents one financial headline, a list of positive keywords, and a list of negative keywords. The number of headlines and keywords is not fixed. A keyword may occur in multiple headlines, and a headline may contain more than one keyword. Assume these are given as hardcoded values in variables named `headlines`, `positive_words`, and `negative_words` (see the example below).

Reference test case

Use the following instance to demonstrate your program in the submitted solution:

```
headlines = [  
    "market rise after strong earnings",  
    "bank reports profit growth",  
    "weak demand increases risk",  
    "shares fall after loss warning",  
    "profit outlook remains strong"  
]  
  
positive_words = ["rise", "strong", "profit", "growth"]  
negative_words = ["fall", "weak", "risk", "loss"]
```

Tasks

1. Count how many headlines contain each positive and negative keyword.
2. Store the individual keyword counts in a dictionary.
3. Print the keyword statistics in alphabetical order by keyword.
4. Compute the total number of positive keyword matches and the total number of negative keyword matches.
5. Classify the overall trend using:
 - "Positive" if positive matches > negative matches
 - "Negative" if negative matches > positive matches
 - "Neutral" otherwise

6. Create a new list containing only the headlines that contain the keyword "profit".

Suggested dictionary structure and hints

Start with an empty dictionary that will store the keyword counts:

```
keyword_counts = {}
```

Suggested steps:

1. Consider every keyword in the positive and negative keyword lists.
2. For each keyword, inspect all headlines using a loop, and count how many contain it. You may use `in` to check for a match.
3. Add an entry to `keyword_counts` with the keyword as its key and the computed count as its value. Do not hardcode the keywords or their counts: the solution must work for different input lists.
4. Once the dictionary has been populated, use its counts to compute the positive and negative totals, and print the statistics in alphabetical order.

Each headline contributes at most once to the count of a particular keyword, but may contain several different keywords.

For the reference data, *after counting*, the dictionary should contain the following key–value pairs (the order of the entries is not important):

```
keyword_counts = {  
    "rise": 1,  
    "strong": 2,  
    "profit": 2,  
    "growth": 1,  
    "fall": 1,  
    "weak": 1,  
    "risk": 1,  
    "loss": 1  
}
```

Note: No word tokenization or advanced text processing is required.

Required output

Print at least:

```
Keyword statistics:  
fall: ...  
growth: ...  
loss: ...  
profit: ...  
rise: ...  
risk: ...  
strong: ...  
weak: ...  
  
Positive matches: ...  
Negative matches: ...  
Overall trend: ...  
  
Headlines containing 'profit':  
[...]
```

Exercise 2 – Investment Performance Analyzer

[7 points]

A manager wants to summarize a set of simplified investment returns expressed as percentages.

Input structure

The input is a one-dimensional list of numerical percentage returns. The number of values is not fixed and the list may contain both valid and invalid observations according to the validation rule below.

Reference test case

Use the following instance to demonstrate your program in the submitted solution:

```
returns = [4.2, -1.0, 7.5, 12.0, 3.1, 25.0, -8.0, 145.0, -130.0]
```

For this exercise, consider a return valid only if it is between -100 and 100 , both included.

Implement at least the following 3 functions:

```
def valid_returns(values):  
    ...  
  
def average(values):  
    ...  
  
def classify(value):  
    ...
```

Use the following categories for valid values:

- "loss" if $\text{value} < 0$
- "low" if $0 \leq \text{value} < 5$
- "medium" if $5 \leq \text{value} < 10$
- "high" if $\text{value} \geq 10$

Tasks

1. Create a new list containing only valid returns.
2. Compute average, minimum, and maximum of the valid returns.
3. Count how many valid returns belong to each category.
4. Produce a new list containing the valid returns sorted from highest to lowest.

Required output

Print:

```
Valid returns: [...]  
Average: ...  
Minimum: ...  
Maximum: ...  
  
loss: ...  
low: ...  
medium: ...  
high: ...  
  
Sorted returns: [...]
```

Do not modify the original list.

Exercise 3 – Nested Investment Fund Structure

[6 points]

A simplified investment structure is represented by nested lists.

Input structure

Each element can be a string, representing an individual investment product, or another list, representing a sub-portfolio that follows the same structure. There is no fixed nesting depth, and a list may contain any number

of products or sub-portfolios. The program counts the total number of individual products and checks whether a specified product exists anywhere in the nested structure.

Reference test case

Use the following instance to demonstrate your functions in the submitted solution:

```
portfolio = [
    "Bond Fund",
    [
        "European Equity",
        "US Equity",
        [
            "Technology Fund",
            "Healthcare Fund"
        ]
    ]
]
```

Implement:

```
def count_products(portfolio):
    ...

def contains_product(portfolio, target):
    ...
```

Tasks

1. `count_products()` must return the total number of individual products at every nesting level.
2. `contains_product()` must return `True` if the target product exists anywhere in the structure, otherwise `False`.
3. Test the functions with the provided structure, including a product present in a nested list and one that is absent. No interactive input is required.

Hints for the recursive functions

- Each element is either a string (one product) or a list (a nested group).
- Use a `for` loop to inspect the elements of the current list.
- You can distinguish the two cases using `type(element) == str` and `type(element) == list`.
- For a string, process the product directly; for a nested list, call the same function recursively on that sub-list.
- Think about what each function should return for an empty list: the count of products and the result of searching for a product that is not present.
- Combine the results from the current level and from the recursive calls to obtain the final answer.

Constraints

- Both functions must use recursion to process nested lists.
- Strings count as investment products; lists represent groups and must not be counted.
- Loops are allowed to inspect the elements of the current list.
- The functions must work at arbitrary nesting depths, including empty lists.

Required output

Use at least the following tests:

```
print("Total products:", count_products(portfolio))
print("Technology Fund found:",
      contains_product(portfolio, "Technology Fund"))
print("Real Estate Fund found:",
      contains_product(portfolio, "Real Estate Fund"))
```