

Programming Techniques 2026 – Exam Simulation

Exam part I – A

14 Ottobre 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 – Portfolio Daily Returns Summary

[6 points]

A portfolio manager records the daily percentage return of a simplified investment portfolio.

Input structure

The input is a one-dimensional list of numerical values, where each value represents the percentage return of the portfolio on one day. The list may contain positive, negative, or zero values, and the same value may occur multiple times. Assume it as given in a variable named `returns`.

Reference test case

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

```
returns = [1.2, -0.8, 2.1, 0.0, 1.5, 2.1, -1.3, 0.6]
```

Tasks

1. Create a **new list** containing only the **strictly positive** returns.
2. Compute the **total**, **average**, and **maximum** value of the original list.
3. Determine how many **distinct return** values appear in the original list.
4. Classify every return using the following rules:
 - "loss" if the return is lower than 0
 - "stable" if the return is between 0 (included) and 1 (excluded)
 - "strong" if the return is greater than or equal to 1
5. Count how many returns belong to each category.

Required output

Your program must print, in a clear format:

- the filtered list of positive returns
- total, average, and maximum return
- the number of distinct values
- the number of **loss**, **stable**, and **strong** days

Example of output structure:

```
Positive returns: [...]  
Total return: ...  
Average return: ...  
Maximum return: ...  
Distinct values: ...  
  
loss: ...  
stable: ...  
strong: ...
```

Exercise 2 – Loan Application Decision

[7 points]

A bank uses a simplified rule to determine whether a loan request is compatible with a customer's income and risk class.

The program must ask the user for:

- the requested loan amount
- the customer's monthly income
- the risk class: **low**, **medium**, or **high**

Use the following simplified policy:

- the base loan limit is four times the monthly income
- for **low** risk, the base limit is unchanged
- for **medium** risk, the base limit is reduced by 20%
- for **high** risk, the base limit is reduced by 40%
- the request is accepted if the requested amount is lower than or equal to the adjusted limit

Input validation

- requested amount must be greater than 0
- monthly income must be greater than 0
- risk class must be one of the three valid strings

Required functions

Implement at least the 3 following functions:

```
def base_limit(income):  
    ...  
  
def adjusted_limit(income, risk_class):  
    ...
```

```
def evaluate_application(amount, income, risk_class):  
    ...
```

The last function must return whether the application is accepted or rejected. You may use default arguments to simplify the function calls.

Required output

After valid input has been obtained, print:

```
Base limit: ...  
Adjusted limit: ...  
Decision: Accepted
```

or:

```
Decision: Rejected
```

The base and adjusted limits must be visible in the output even when the request is rejected.

Exercise 3 – Recursive Transaction Status Analyzer [7 points]

A transaction log is represented by a string of single-character status codes:

- G: gain
- L: loss
- R: neutral/review

The program must ask the user for a valid target status code and then analyze the log to count how many times the target appears and to create a new string in which all occurrences of the target have been removed.

Input structure

The input is a string containing zero or more single-character status codes. Its length is not fixed and any of the valid status codes may occur any number of times. Assume it as given in a variable named `status`.

Reference test case

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

```
status = "GGLRGL"
```

Implement the following two functions using **recursion**:

```
def count_status(text, target):  
    ...  
  
def remove_status(text, target):  
    ...
```

Tasks

1. `count_status(text, target)` must return how many times `target` appears in `text`.
2. `remove_status(text, target)` must return a new string containing all characters except `target`.
3. Test both functions using the provided string.

4. In the main program, ask the user for a valid target status code and then print the number of occurrences and the new string without the target.

Constraints

- both functions must use recursion
- do not use `for` or `while` loops inside these functions
- do not use `str.count()` or `str.replace()`
- the original string must not be modified (suggestion: print it to verify it actually remains unchanged)

Required tests and output

Use at least:

```
status = "GGLRGL"  
target = "G"
```

and print:

```
Occurrences of G: ...  
Without G: ...
```

Your functions must also work correctly when the input string is empty.