INF1002 Programming Fundamentals Python Projects

Objective of the project

- Practice abstracting real-world problems into logical structures and implementing them in code.
- Develop skills in writing modular, efficient, and maintainable programs.
- Apply appropriate frameworks and libraries for problem-solving.
- Coordinate teamwork and unify coding styles in collaborative development.
- Strengthen project management skills, including communication and conflict resolution.

Timeline and Deliverables

Week	Deliverables	Deadline
1	**Proposal** — Team list, project title, short description, initial task allocation	Week 2 Monday 8:00 AM
2	**System Design** — System/module breakdown, core feature list, dataset description, updated task allocation	Week 3 Monday 8:00 AM
3–4	Development — Core code implementation, working prototype. **GitHub Link**	Week 5 Monday 8:00 AM
5–6	Testing & **Final Report** — Testing report, final documentation, demo video, updated code (if needed)	Week 7 Monday 8:00 AM

Report Preparation/submission Instruction:

How to submit

- Submit your documents in xSITE Dropbox (‘Python Project xxx) in pdf format.

Video Requirements

- Presentation (Slides Optional)
    - The video must include a report on the entire system. You can present based on your report without needing to create slides, but make sure the presentation covers the key aspects of the system, such as objectives, design, functionality, results, etc.

- Participation
    - Each member must speak to practice their presentation skills. You can decide among yourselves who presents which part.
- System Demonstration
    - The video should include a demonstration of the system in action.
- Fully Functional System
    - The system you submit must be fully functional and able to run without issues.
- Code Structure Explanation
    - The video should provide an overview of the code structure, explaining the purpose of each module.
    - Walk through the main parts of the code without going into detail for every line, focusing on the overall design and functionality.
- Time Management
    - You have flexibility in managing the overall time. The video should not be overly simple, with a suggested duration of around 15 to 30 minutes.

Report format

- The final report must be between 12 to 15 pages (single column, font size 12, single spacing). Reports longer or shorter may be penalized.
- To reinforce logical reasoning and problem decomposition, your report should primarily explain the project's Key Functions. If helpful, use the following as guidelines (you don't need to follow every item strictly) for each chosen function:
    - Context & Purpose: What problem does this function solve in your project? E.g. *Best Time to Buy and Sell Stock*
    - Signature: Function name, parameters, and return type. *best_profit(prices: list[int]) -> int*
    - Inputs & Constraints: Data types, valid ranges, edge conditions. *Close prices of stock, non-negative numbers*
    - Outputs: Exact meaning and format. *Maximum profit from one buy and one sell*
    - Algorithm & Rationale: The core idea and why it works (in your own words).
    - Complexity: Big-O time and space.
    - Edge Cases: How the function handles them.
    - Example Trace: A small, concrete example showing step-by-step state changes.
    - Unit Tests (samples): 3–5 illustrative tests (including edge cases).

- The following provides a standard format for your reference.

- Title and team information
    - In the first page, you shall provide your project title, and team and members information including team ID, team member name, student ID and emails.
- Abstract
    - A summary of your project including, the problem you

investigated or the purpose, your proposed approach/method and your major findings including key quantitative results and interpretations.

- Introduction
    - Write down the detail context of your project, state the purpose of the work of your project in the form of question or problem your project investigated; Briefly explain your rationale and approach you used, and the outcome/results/conclusion of your project.
- Related works or Literature
    - In this section, you need to provide a brief related works with citations. Discuss with citations whether there are any other papers/tools/systems addressed the similar/same problem and what are the difference between yours and theirs. Any other related information etc.
- Methods
    - In this section, you can write down the detail implementation of your project. This may include the information below:
    - Dataset used: introduce the detail dataset you used
    - System diagram: A diagram includes the main component/technology of your project. Each component of the diagram can be introduced with details in the below sections.
    - Data pre-processing: introduce the details about how you pre-process your data in order to analyze them.
    - Main Tasks: you need to list all the analyses tasks that you performed in your project. And write down the very detailed algorithms or approaches that you did, the rationale of doing that and proper discussion and explanation.
- Result and Insight
    - You must write down the results, outcomes and insight by analyzing the data. You may wish to add proper Tables and Figures to show your result/insight with proper and detailed interpretations and discussion. The discussion is used to interpret your results in light of what was found from the data analysis and explain the new understanding of the problem after taking your results into consideration.
- Conclusion
    - Write down a concise conclusion to summary your whole project with potential interesting future work.
- References
    - Put down all references including the papers you cited or URLs you used in the project.
- Appendix
    - In your appendix, you can add additional information, results, screenshots, and so on, if needed. **Please note that a reflection is mandatory.**

Reflection

- Each report must include a reflection section in the appendix, and **each group member must write their own reflection**.
- Please answer the following three questions briefly:

- Time Management
  - How did you manage your time during the project? Was there anything you would do different next time?
- Technical Challenge
  - Which part of the project was the most difficult technically, and how did you solve it (or try to solve it)?
  - What are examples of good programming practices?
  - What are some bad practices or behaviors that should be avoided?
- Other Reflections
  - Is there anything else you would like to reflect on (e.g., what you learned, surprises, or teamwork)?
- There are no length requirements for this section.

Teamwork Tips – Questions to Ask Yourself

When you face difficulties in the project, try asking yourself these questions:

- What have I contributed to the team?
- Did I speak up and share my ideas during team meetings?
- Have I helped someone else in the group?
- When I encountered problems, was I open and honest in seeking help?
- Did I make an effort to understand different perspectives?
- When I noticed issues in others' work, did I give constructive suggestions?
- Was my communication style respectful and acceptable to others?
- When the team made decisions, what criteria did we use to accept or reject suggestions?
- How does our team communicate? Do we meet regularly, chat in a group?
- When I feel communication is needed, did I take the initiative to start a conversation?

Plagiarism

SIT's policy on copying does not allow you to copy software as well as your assessment solutions from another person. It is not acceptable to copy other person's work. It is the students' responsibility to guarantee that their assessment solutions are their own work. Meanwhile, you must also ensure that others don't obtain access to your work. Where such plagiarism is detected, both assessments involved will receive ZERO mark.

Rubrics

Category	Weightage	Excellent (10-8.5)	Good (8.4-7)	Average (6.9-6)	Fail to Meet Expectations (5.9-0)
Programming: Correct logic, complete functionality, efficiency	0.25	Implements all required features correctly with efficient, well-tested logic.	Most features implemented with minor logic issues or incomplete testing.	Basic functionality works but with noticeable errors or missing features.	Frequent logic errors, incomplete or non-functional program.
Programming: Modular design, reusable functions	0.25	Code is well-structured into reusable, modular functions with clear separation of concerns.	Mostly modular code with small issues in reuse or structure.	Limited modularity, repetitive code or poor function design.	No modular structure, highly repetitive or hard-to-follow code.
Programming: Code readability and comments	0.15	Code is clean, well-formatted, with clear variable names and helpful comments throughout.	Generally readable code with some missing comments or naming inconsistencies.	Readable but minimal comments and unclear variable naming.	Poor readability, lack of comments, confusing or inconsistent naming.
Reflection: Individual learning reflections	0.10	Detailed and thoughtful reflection showing deep understanding and lessons learned.	Good reflection with relevant insights but lacks depth in some areas.	Basic reflection with limited insight or vague responses.	Little to no meaningful reflection on learning experience.
On-time submission: Milestones and final delivery	0.10	Deduct 2 points per late submission			
Peer review: Contribution to team and feedback to peers	0.15	Evaluated through the peer evaluation system			

Project Idea Samples

Feel free to add extra features.

Please keep in mind that the focus is not on stacking as many features as possible, but on solving well-defined “smaller” problems with clear logic. For example, I would value

more the way you design a function to count how many times a word appears (even if that may be a very simple case), rather than just calling model_x.predict(article) to check whether the output is positive.

I will also continue to share new ideas, if I have any, over the next few weeks.

Example of how to make your project stand out:

Sliding Window Example: Stock SMA(5)

Suppose we want to calculate the **Simple Moving Average (SMA)** with window size 5 for a stock price series.

- **Naïve approach (O(n·k)):**
  For each position, we take the previous 5 numbers and compute their sum again from scratch. This repeats a lot of work, because most numbers are used again in the next window.
- **Sliding window approach (O(n)):**
  First, compute the sum of the first 5 numbers. Then, when the window slides by one step, subtract the element that goes out of the window and add the new one that comes in. This way, each element is added once and removed once.

This reduces the complexity from **O(n·k)** to **O(n)**. For large datasets (like stock prices with millions of points), the sliding window method is much more efficient.

```
def sma_5(prices):
    k = 5
    window_sum = sum(prices[:k])
    result = [window_sum / k]

    for i in range(k, len(prices)):
        window_sum += prices[i] - prices[i-k]   # update in O(1)
        result.append(window_sum / k)

    return result
```

Key point: Both methods give the same result, but the sliding window approach saves a lot of redundant computation.

And if you also handle corner cases properly (e.g., when the data has fewer than 5 points, or when missing values appear), that will make your solution even stronger.

Project 1: Stock Market Trend Analysis

Objective

The goal of this project is to analyse stock market data and practice transforming real-world problems into programming logic using Python. Students will calculate basic trading metrics, identify market trends, and visualize results. The project emphasizes algorithmic thinking, data processing, modular code design, and clear documentation of the development process.

Dataset

Input: A dataset containing daily stock prices (open, high, low, close, volume at minimum) for a period of up to three years.

Requirements

Core Functionalities

1. Simple Moving Average (SMA): Compute SMA for a given window size (e.g., 5 days).
2. Upward and Downward Runs: Count the number and total occurrences of consecutive upward and downward days (based on close-to-close changes), and identify the longest streaks for each direction.
3. Daily Returns: Compute simple daily returns: $r_t = \frac{P_t - P_{t-1}}{P_{t-1}}$
4. Max Profit Calculation: Implement the solution for Best Time to Buy and Sell Stock II (https://leetcode.com/problems/best-time-to-buy-and-sell-stock-ii/description/) to find maximum profit with multiple transactions allowed.

Visualization

1. Plot daily closing price vs. SMA on the same chart.
2. Highlight upward and downward runs on the price chart.

Validation results

1. Include comparisons with trusted or manual calculations for at least 5 test cases to ensure your results are correct.

- The requirement of “at least 5 test cases” is just a general guideline. Here a test case means one situation where you compare your implementation against a trusted source or manual calculation. For example: Suppose you want to calculate the Simple Moving Average (SMA) for stock prices. You can implement your own function to compute it. Then, as a validation step, you compare your result with a trusted calculation method. For example, you can compare against the result produced by pandas’ built-in .rolling().mean() function, or you can manually calculate the SMA for a few days using Excel. Each such comparison counts as one test case.
- You can also design corner cases, such as when the data series is shorter than the SMA window, or when one day’s data is missing.

- So in total, you should have at least 5 such validations to demonstrate correctness. But this is a flexible requirement, you can do more if useful, or fewer if it really doesn't add value.

Project 2: Sentiment Analysis System

Objective

The goal of this project is to design and implement a sentiment analysis system based on a predefined sentiment dictionary. Given a text document containing multiple paragraphs and sentences, the system should process the text and compute sentiment scores. Students will practice decomposing a real-world problem into programming logic and algorithms, collaborate using GitHub, and produce a final system with both backend logic and a simple frontend visualization.

Dataset

For a sentiment analysis system, you need a dataset of sufficient size to test and demonstrate its effectiveness. If you choose to use a machine learning approach, you must have a training set (to build the model) and a test set (to evaluate it). The training set and test set must not overlap to ensure a fair and reliable evaluation.

Example datasets:

- IMDb Movie Reviews — 50,000 English movie reviews, labelled positive or negative (balanced dataset). Source

Requirements

1. Calculate the sentiment score of each sentence using the provided dictionary.
2. Identify the most positive and most negative sentences in the entire text.
3. ~~Apply a sliding window over paragraphs (e.g., 3 sentences per window) to determine the most positive and most negative paragraph segments.~~
4. Apply a sliding window over paragraphs (e.g., 3 sentences per window) to determine the most positive and most negative paragraph segments.
   Follow-up: If you also want to know which exact sentences form these segments, how would you modify the algorithm?
5. Without fixing the window size, find the most positive and most negative continuous paragraph segments of arbitrary length.
   Follow-up: If you also want to know which exact sentences form these segments, how would you modify the algorithm?
6. Suppose you have an English dictionary. During data processing, all spaces in a sentence were accidentally removed (e.g., "thisisapen"). Re-insert spaces to find a possible valid segmentation.
   Follow-up: If multiple valid segmentations exist, how would you return all possible combinations?

The dataset (text samples) and sentiment dictionary (https://github.com/fnielsen/afinn/blob/master/afinn/data/AFINN-en-165.txt) will be provided.

Project 3: Phishing Email Detection

Objective

The objective of this project is to design and implement a rule-based system to detect phishing emails. Students will write Python programs to analyze emails using various string processing and logical techniques, practicing problem decomposition and programming fundamentals.

Dataset

For phishing email detection, you need a dataset containing labelled examples of legitimate (ham) and phishing/spam emails to test your approach. The dataset should include enough diversity in email topics, writing styles, and formats. Both rule-based and machine learning methods are acceptable, but you must use a separate test set to evaluate performance.

Example datasets (you may choose your own):

1. Enron Email Dataset: https://www.cs.cmu.edu/~enron/
2. SpamAssassin Public Corpus: https://www.kaggle.com/datasets/beatoa/spamassassin-public-corpus

Requirements

1. Whitelist Check: Verify if the sender's email domain is on a predefined safe list.
2. Keyword Detection: Scan email subject and body for suspicious keywords (e.g., 'urgent', 'verify', 'account').
3. Keyword Position Scoring: Assign higher risk scores for suspicious keywords appearing in subject lines or early in the message.
4. Edit Distance Check: Compare email domains and sender names against known legitimate domains to detect visually similar fakes.
5. Suspicious URL Detection: Identify links that do not match the claimed domain or contain IP addresses instead of domains.
6. Final Risk Scoring: Combine results from all rules to classify emails as Safe or Phishing.