CS6.302 - Software System Development

Assignment 3 – Python

Due: 15 Nov 2025, 05:00 PM

Total Marks: 100

NOTE: This assignment is an individual submission, not a group activity. Evaluation will be conducted based on a fixed grading rubric (syntax, logic, input and output) and the marks are divided as per prescribed weightage in respective question. Inputs/output should fit the criteria mentioned in respective questions. Unless it is specified, all input/output criteria are open to interpretation. All questions in the assignment are self-explanatory. Do not reach us for any clarifications. If you are answering a question based on a certain assumption, please feel free to mention it as part of your README file.

Submission Instructions:

Please follow the instructions below carefully. A strict zero will be given if the submission format doesn't adhere to this.

- Please submit your code in Moodle.
- Programs must be working correctly.
- Programs must be written in Python.
- Programs must be submitted with the format mentioned in each question. Programs must be saved in files with the correct file name given in each question.
- Please do not forget to include a README.md file to mention your assumptions, execution instructions or anything else in the ZIP
- Please feel free to make any **valid assumptions** wherever the question seems ambiguous. Clearly mention these assumptions in README.md file. Marks will be awarded **only if the assumptions are stated explicitly**. Any **invalid assumptions will lead to penalties**.
- Submissions that do not adhere to the file structure or naming convention **will not be graded**.
- If you are using any LLM for this task, please declare your usage with all required details here - https://forms.office.com/r/754tAUacRk If you are found not mentioning about your LLM usage despite using one, you will be awarded '0'. You will be awarded '0' if your submission is found to be plagiarized with other submissions.
- **Submission Format**:
  - You are required to submit this lab activity as <roll_number>.zip. For example, if you roll number 20162153, then your submission file should be 20162153.zip.
  - Inside this .zip, create **separate folders for each question (Q1, Q2,...Q6).** Each folder should contain the submission in the format specified for that particular question.

Q1: Code Similarity Analysis using Python — The Case of 27 MERN Implementations of *VidyaVichar* project (25 Marks)

Objective: To design and implement a Python-based framework to **quantitatively and qualitatively evaluate code similarity** across 27 independent implementations of the same MERN-stack project titled *VidyaVichar* - ***Dataset*.** Students will apply techniques ranging from syntactic (textual) comparison to semantic (structural and functional) similarity analysis and present their findings with visual analytics.

Background: *VidyaVichar* is a MERN (MongoDB, Express, React, Node.js) stack project developed by 27 different teams. Although each team solved the same problem statement, their implementations may differ in: Folder structures, Coding conventions, Logic flow and architectural design, API route naming and model structures. Understanding **how similar or diverse** these implementations are providing valuable insight into software engineering practices, plagiarism detection, and collaborative coding styles.

Assignment Tasks:

Part A — Preprocessing & Data Understanding (5 Marks)

1. Collect all 27 project folders and organize them in a common directory.
2. Preprocess the code:
    a. Remove comments, logs, and minified code.
    b. Normalize formatting (indentation, spacing).
    c. Identify comparable file types: .js, .jsx, .json, .css.
3. Summarize:
    a. File and folder count per project.
    b. Line of code (LOC), number of React components, Express routes, and Mongoose models.

Part B — Code Similarity Computation (15 Marks)

Design Python scripts to perform at least **three levels of similarity analysis**:

Level	Description	Expected Tools / Libraries
Textual Similarity	Line-level or token-level similarity	difflib, Levenshtein, TF-IDF, cosine_similarity
Structural Similarity	Abstract Syntax Tree (AST) comparison, route or schema matching	esprima, ast, json parsing
Semantic Similarity	Code embeddings, function-level meaning similarity	codeBERT, graphcodeBERT, sentence-transformers

Each student must:

- Compute pairwise similarity between all 27 projects (or a sampled subset).
- Present the results as a **similarity matrix (NxN)**.
- Interpret which projects are most and least similar.

Part C — Visualization & Reporting (5 Marks)

1. Visualize findings using Python libraries like matplotlib, seaborn, or plotly.
    a. Heatmap of similarity matrix.
    b. Network graph of project clusters based on similarity.
    c. Bar charts for average similarity by metric.
2. Write a 1 page **analytical report** explaining:
    a. Your methodology and justification of techniques used.
    b. Observations and insights about coding diversity.
    c. Any patterns of potential reuse or structural consistency.

Submission Requirements:

Submit the following files in a folder named Q1:

1. similarity_analysis.ipynb – Python notebook with documented code.
2. results/ – Folder containing:
    a. Preprocessing summaries (CSV/JSON).
    b. Similarity matrix and plots.
3. report.pdf – Analytical write-up with discussion and conclusions.
4. Optional: interactive dashboard (e.g., Streamlit) for similarity visualization.
5. README.md - mentioning an sort of assumptions or requirements needed to run your code.

Q2: Implementation and Performance Analysis of Image Blurring (15 Marks)

1. Problem Description

The objective of this question is to write a Python script that implements, benchmarks, and analyzes two distinct methods for applying a 3x3 mean blur filter to a generated high-contrast image. The primary goal is to demonstrate a practical and theoretical understanding of the performance differences between an iterative Python approach and a vectorized NumPy approach.

A **3x3 mean blur** is an image convolution operation where the value of each output pixel is the arithmetic mean of the corresponding input pixel and its eight immediate neighbors.

Note: Use a Jupyter notebook for this question.

2. Setup and Input Data Generation

As the first part of your script, you must programmatically generate the input image. Do not use an external image file.

Your script should use the **Pillow** library (PIL) to create a 400x200 pixel, 8-bit grayscale image. The image must have a black background (pixel value 0) and contain the word "**SSD TAs ARE THE BEST**" centered in a large,

white font (pixel value 255). Once generated, this Pillow image must be converted into a NumPy array to serve as the original_image for the subsequent tasks.

3. Implementation Requirements

You are required to implement two distinct functions to perform the blurring operation.

Task 1: Iterative Python Implementation

Write a function named blur_python that accepts a single argument: a NumPy array representing the image. This function must use nested for loops to iterate through the image pixels and calculate the 3x3 mean blur. To handle image borders, your iteration should exclude the outermost single-pixel boundary.

Task 2: Vectorized NumPy Implementation

Write a second function named blur_numpy that also accepts a single argument: a NumPy array representing the image. This implementation must not contain any explicit Python for loops. The logic must be fully vectorized using NumPy's array slicing, padding, and broadcasting capabilities to achieve the same blurring effect.

4. Benchmarking and Verification

After implementing both functions, your script must perform the following evaluation steps:

1. **Benchmark:** Measure the execution time (in seconds) of both blur_python and blur_numpy when processing the original_image. Print the results clearly to the console.
2. **Visualize:** Using the matplotlib library, generate a single figure that displays three subplots side-by-side:
    a. The original_image.
    b. The blurred image returned by blur_python.
    c. The blurred image returned by blur_numpy.

5. Analysis and Discussion

Conclude your assignment with a written analysis that addresses the following points based on your results:

1. **Performance Results:** Report the performance difference between the two implementations. State whether the magnitude of this difference aligns with theoretical expectations.
2. **Core Concepts:** Explain the fundamental reasons for the observed performance gap. Your explanation must correctly define and apply the following concepts to your implementations:
    a. **Vectorization:** How does this principle apply to your blur_numpy solution?
    b. **Compiled Code vs. Interpreter:** Contrast how NumPy operations are executed compared to the iterative logic in your blur_python function.

c. **Memory Layout:** Briefly explain why NumPy's contiguous memory model provides an advantage for this type of numerical task.

6. Submission Requirements

Submit the following files in a folder named Q2:

1. box_blur.ipynb - Your complete implementation with all code and analysis and discussion section. (The analysis and discussion can be added a markdown in your notebook)
2. README.md - mentioning an sort of assumptions or requirements needed to run your code.

Q3. Sorting Algorithms (10 marks):

In algorithms there are various kinds of sorting algorithms. Using them depends on various different factors, such as time, space, distributed setting, edge devices and a lot more.

Design a Python package with the following:

1. A parent abstract base class for sorting algorithms.
2. Implement bubble sort, selection sort, quick sort, and merge sort using this interface in different classes.
3. Design a test package that tests these 4 algorithms to check for the correctness of these algorithms. You may use packages such as pytest (not compulsory to use pytest) for the same.
4. A class which calls these various algorithms based on a parameter.
5. The number of elements to sort can be less than 2x1e5, and each element falls within the range of INT32.
6. The parameter for the sort of function should include the following:
    a. Ascending or descending
    b. Name of the particular algorithm to be used
    c. The size of the List
    d. The input list
7. Output should be a new list that is returned.
8. The algorithm should only work for Integer data type

Submission Requirements:

Strictly use git for the same (github not necessary). Use git tags or branches. Have a folder called 'Sorting_Package' with 3 folders called, src, test and reports. Have a main.py where you can showcase how these different function works. You should have a input file for the input, and the outputs should be redirected using the terminal.

Q4. Pylint time: (10 marks)

Go to the code written in Q3. Run pylint on it. Generate the report and attach the report. Now refactor the code and make the pylint score greater than 8.5. Make sure that you attach both the codes and reports. Before and after refactoring. Attach a report explaining what changes you made and why. Make sure your algorithm passed all the test cases written by you after refactoring. Now add another sorting algorithm for shell sort, and see how the pylint score changes.

Submission Requirements:

Add the necessary items to the 'Sorting_Package' folder. Showcase the changes using git tags and branches.

Q5. Build a kaooa board game in Python using the following resources. You may use Python libraries like Turtle etc. Following are a few references for this board game. (20 marks) -

a. https://www.whatdowedoallday.com/kaooa/
b. https://www.youtube.com/watch?v=Jzeug1XTRQM

Submission Requirements

Submit the following files in a folder named Q5:

1. kaooa.py - The code can be divided into modules, but your submission must be executable from this file.
2. README.md - Any relevant documentation

Q6. Build a calculator in Python that evaluates mathematical expressions using octal number system (base-8). Your calculator must accept octal inputs, perform operations in octal, support user-defined recursive functions, and return results in octal format. (**20 Marks**)

Component-wise Implementation Requirements

1. **Octal Arithmetic Operations with Variables**

   The system must implement the standard arithmetic operators: addition (+), subtraction (−), multiplication (×), division (÷), modulo (%), and exponentiation (∧), while strictly adhering to the PEMDAS order of operations and supporting arbitrarily nested parentheses. It must incorporate a variable binding mechanism using the syntax `LET <variable> = <octal_value> IN <expression>`, where variables are scoped locally to the `IN <expression>`, which enables nested LET bindings. The final result of the top-level expression must also be presented in octal format.

   Expected behavior:

   "10 + 7" → "17"

   "LET x = 10 IN x + 7" → "17"

Note: Division should return integer values in octal (no fractions).

2. **User-Defined Recursive Functions**

The task is to extend the system to support user-defined recursive functions. This requires implementing a `DEF` syntax to permanently store function definitions: `DEF <name>(<param1>, <param2>, ...) = <expression>`. Functions must be callable using their name and octal arguments: `<name>(<octal_arg1>, <octal_arg2>, ...)`. The system must ensure that function parameters are local to the function's body and that the number of arguments provided matches the number of parameters defined. The evaluator must fully support recursive function calls, where both parameters and the function's return value are in octal format. To maintain stability, the system must track the recursion depth and prevent stack overflow by imposing a hard limit of 1000 nested calls.

Expected behavior:

"DEF square(x) = x * x" → (stores function)

"square(5)" → "31"

3. **Conditional Expressions**

The task is to introduce a conditional expression with the syntax `IF <condition> THEN <expression1> ELSE <expression2>`. The system must evaluate the `<condition>` and execute only the chosen branch (<expression1> or <expression2>). The condition supports standard comparison operators: equal to (==), not equal to (!=), less than (<), greater than (>), less than or equal to (<=), and greater than or equal to (>=), all operating on octal values. This conditional structure must be fully nestable and allow the use of both variables and function calls within the condition itself and within the subsequent expression branches.

Expected behavior:

"IF 10 > 7 THEN 5 ELSE 3" → "5"

Feature Interaction:

Component	Can Use
Arithmetic & Variables	Operators, parentheses, LET bindings (all in octal)
Functions	Arithmetic, variables (parameters), conditionals, recursion (all in octal)
Conditionals	Arithmetic, variables, function calls, nested conditionals (all in octal)

Example of everything together:

“DEF sum_squares(n) = IF n <= 0 THEN 0 ELSE n * n + sum_squares(n - 1)"

"sum_squares(5)" → "107"

Assertions Requirements

Include meaningful assertions throughout your code that validate pre-conditions, invariants, and post-conditions at critical points where bugs could occur

Exception Handling Requirements

Design and implement a custom exception hierarchy with informative error messages that handles all possible error scenarios gracefully

Testing Requirements

Provide comprehensive test cases that demonstrate all features work correctly, cover edge cases, and verify error handling. Include tests specifically for octal conversions and arithmetic.

Documentation Requirements

Provide clear documentation explaining:

- Your parsing approach and algorithm
- How you handle octal to decimal and decimal to octal conversions
- How you handle recursion safety and depth tracking
- Your variable scope management strategy
- Your exception hierarchy design and rationale
- Your assertion strategy (what you're protecting and why)
- Any design decisions or assumptions made

Technical Constraints

- No eval() or exec() - must parse manually
- No external libraries except re if needed for parsing
- No external mathematical or numerical libraries are permitted (specifically: SymPy and NumPy are prohibited).
- No built-in octal conversion functions like oct() or int(x, 8) - implement conversion yourself
- Recursion limit: Maximum 1000 calls
- Keywords (uppercase): LET, IN, DEF, IF, THEN, ELSE
- All inputs are octal strings, all outputs are octal strings

Submission Requirements

Submit the following files in a folder named Q6:

1. octal_calculator.py - Your complete implementation with all code

2. exceptions.py - Custom exception class definitions
3. test_cases.py - All the test cases
4. Report.pdf - Documentation
5. README.md - mentioning an sort of assumptions or requirements needed to run your code.

_(-_-)_/ Happy -_- Programming!