**Alexandre.HE23621@student.rcl.ac.uk**

**APR24 - SWE4201 - Introduction to Software Development**

|  |  |
| --- | --- |
| **Jose Garrido HE23621** |  |

Contents

[**Introduction :** 2](#_Toc188738340)

[Portfolio Item 1: Student Registration System 3](#_Toc188738341)

[Design and Explanation 3](#_Toc188738342)

[Testing the Code 3](#_Toc188738343)

[Portfolio Item 2: Python Program to Display Patterns 4](#_Toc188738344)

[Portfolio Item 2: Structure Display 7](#_Toc188738345)

[Design and Explanation 7](#_Toc188738346)

[Testing the Code 7](#_Toc188738347)

[Portfolio Item 3: Java Student Record System 8](#_Toc188738348)

[Design and Explanation 8](#_Toc188738349)

[Testing the Code 8](#_Toc188738350)

[Critical Reflection 9](#_Toc188738351)

[References 9](#_Toc188738352)

[Appendix 10](#_Toc188738353)

# **Introduction :**

This assignment serves as a comprehensive exercise in the practical application of Python and Java programming, focusing on the development of real-world systems. The tasks outlined are designed to reinforce foundational concepts in software development while addressing core learning outcomes such as algorithm design, programming constructs, systematic testing, and object-oriented principles.

The assignment is divided into three distinct yet interconnected portfolio items. The first task involves creating a Python-based student registration system that effectively manages student records using a dictionary data structure. This system emphasizes the importance of efficient data management, error handling, and user-friendly interactions.

The second task explores the versatility of Python by demonstrating the use of fundamental data structures, including lists, dictionaries, and tuples. Through this exercise, the understanding of how these structures operate and their practical applications in solving programming problems are showcased.

The final task introduces Java programming to design and implement a robust student record system. This system integrates object-oriented principles such as encapsulation, abstraction, and modularity to manage and manipulate student data. Additionally, systematic testing methods and a user-friendly interface ensure the program meets functional and usability requirements.

Each task aligns with the learning outcomes of this module, providing an opportunity to apply theoretical knowledge in a practical context. By integrating programming constructs, systematic testing, and effective design, this assignment highlights the importance of a structured and methodical approach to software development.

# Portfolio Item 1: Student Registration System

Objective: Create a Python program for managing student registrations.  
Key Features:  
- Add students with unique IDs.  
- Retrieve student details.  
- Prevent duplicate entries and handle invalid inputs.

## Design and Explanation

Data Structure: A dictionary is used to store student details, ensuring efficient lookups and updates. Error handling is implemented to validate inputs, ensuring unique IDs and non-empty fields.

Sample Code:  
  
1. students[student\_id] = {"Name": name, "Course": course}

## Testing the Code

Test cases were designed for valid and invalid inputs. Outputs were verified for correctness. For example, adding a student with ID '101' and name 'Alice' resulted in a successful registration and correct retrieval of details.

# Portfolio Item 2: Python Program to Display Patterns

**Overview**

The goal of this task is to design a Python program that generates three distinct patterns: an increasing triangle, a decreasing triangle, and a symmetrical triangle. Each pattern highlights the use of nested loops and emphasizes structured problem-solving in Python programming.

**Design Explanation**

The program uses loops to create and display the three patterns. Each pattern is designed with specific requirements for the number of rows and alignment of the \* symbols.

**Pattern (a): Increasing Triangle**

* **Objective**:
  + Generate a right-angled triangle where the number of \* symbols increase with each row.
* **Design**:
  + Use a for loop to iterate through the rows, starting from 1 and increasing to the desired number of rows (e.g., 10).
  + For each row, the number of \* symbols equal the current row number.
* **Implementation**:
  + A single loop handles both the row count and the printing of the \* symbols.
  + Example:

1. for i in range(1, 11): # Rows from 1 to 10

2. print('\*' \* i)

3. Output:

4. \*

5. \*\*

6. \*\*\*

7. \*\*\*\*

8. \*\*\*\*\*

9. \*\*\*\*\*\*

10. \*\*\*\*\*\*\*

11. \*\*\*\*\*\*\*\*

12. \*\*\*\*\*\*\*\*\*

13. \*\*\*\*\*\*\*\*\*\*

**Pattern (b): Decreasing Triangle**

* **Objective**:
  + Generate a right-angled triangle where the number of \* symbols decreases with each row.
* **Design**:
  + Use a for loop to iterate through the rows, starting from the maximum number of rows (e.g., 10) and decreasing to 1.
  + For each row, the number of \* symbols equals the current row number.
* **Implementation**:
  + A single loop iterates over a decreasing range and prints the \* symbols for each row.
  + Example:

1. for i in range(10, 0, -1): # Rows from 10 to 1

2. print('\*' \* i)

3. Output:

4. \*\*\*\*\*\*\*\*\*\*

5. \*\*\*\*\*\*\*\*\*

6. \*\*\*\*\*\*\*\*

7. \*\*\*\*\*\*\*

8. \*\*\*\*\*\*

9. \*\*\*\*\*

10. \*\*\*\*

11. \*\*\*

12. \*\*

13. \*

**Pattern (c): Symmetrical Triangle**

* **Objective**:
  + Generate a cantered, symmetrical triangle of \* symbols with increasing rows.
* **Design**:
  + Use a combination of loops to handle alignment and star printing:
    - The first loop handles the number of rows.
    - A nested loop prints spaces to centre-align the stars.
    - Another nested loop prints the \* symbols for the current row.
* **Implementation**:
  + Calculate the number of spaces for alignment as the difference between the total rows and the current row.
  + Multiply the \* symbol for the current row to create the triangle.
  + Example:

1. rows = 6 # Number of rows

2. for i in range(rows):

3. print(' ' \* (rows - i - 1), end='') # Spaces for alignment

4. print('\* ' \* (i + 1)) # Print stars

5. Output:

6. CopiarEditar

7. \*

8. \* \*

9. \* \* \*

10. \* \* \* \*

11. \* \* \* \* \*

12. \* \* \* \* \* \*

**Testing and Results**

1. **Testing Approach**:
   * Verify each pattern individually by running the respective functions.
   * Test edge cases, such as setting the number of rows to 1 or increasing it to higher values.
2. **Results**:
   * The program correctly generates each pattern:
     + **Pattern (a)**: A right-angled triangle with increasing stars.
     + **Pattern (b)**: A right-angled triangle with decreasing stars.
     + **Pattern (c)**: A centered triangle with aligned stars.

**Challenges and Solutions**

* **Challenge**: Aligning the stars for the symmetrical triangle.
  + **Solution**: Calculate spaces dynamically based on the current row and total rows.
* **Challenge**: Ensuring reusability and clarity of code.
  + **Solution**: Encapsulate each pattern in a function to make the program modular and maintainable.

# Portfolio Item 2: Structure Display

Objective: Demonstrate Python structures like lists, dictionaries, and tuples.  
Key Features:  
- Display elements of each structure.  
- Show operations like indexing and updating.

## Design and Explanation

Static data was used to demonstrate the capabilities of lists, dictionaries, and tuples. Functions were modular to allow easy understanding and reuse.

1. Sample Code:  
  
sample\_list = ["meat", "diary", "gluten", "peanuts"]

## Testing the Code

Tested outputs for static data and verified the correctness of all operations for each structure.

# Portfolio Item 3: Java Student Record System

Objective: Create a Java program for managing student records.  
Key Features:  
- Add, retrieve, update, and display student details.  
- Use of object-oriented principles like encapsulation and abstraction.

## Design and Explanation

The program is built around four main classes, each with a clear and specific role:

1. **Student Class**:
   * This class holds the details of each student, like their ID, name, and course. By keeping these fields private and using getters and setters, it ensures the data is managed safely and follows good programming practices. The toString method provides an easy way to display a student's information.
2. **StudentRecordSystem Class**:
   * This class manages all the student records using a HashMap. It handles tasks like adding, retrieving, updating, and listing students. The use of a HashMap makes it quick to find and update student details, ensuring the system runs efficiently.
3. **ProgressBar Class**:
   * This class adds a nice touch to the user experience by showing a loading bar during operations like adding or retrieving records. It’s a small detail that makes the program feel more polished and engaging.
4. **Main Class**:
   * The Main class ties everything together. It provides a simple menu for users to interact with the system and connects user inputs to the features in StudentRecordSystem. It also ensures inputs are validated, keeping the program easy to use and reliable.

Together, these classes make the program organized, easy to work with, and user-friendly.

1. Sample Code:  
  
students.put(student.getId(), student);

## Testing the Code

All functionalities were tested:  
- Adding students.  
- Retrieving existing and non-existing IDs.  
- Updating student details.  
- Displaying all records.  
Test results confirmed the program's correctness and robustness.

# Critical Reflection

# This assignment significantly enhanced my understanding of Python and Java programming, particularly in designing and implementing real-world systems using object-oriented principles and systematic testing. Through the Python tasks, I gained a deeper appreciation for the versatility of Python data structures, such as lists, dictionaries, and tuples, and their practical applications in managing and manipulating data efficiently. The task of generating patterns using nested loops further solidified my ability to use control structures effectively, allowing me to create visually structured outputs.

# The Java component provided invaluable insights into object-oriented programming (OOP), emphasizing core concepts like encapsulation, abstraction, and modularity. By designing a student record system, I learned how to break down complex problems into manageable components, such as creating dedicated classes for Student, StudentRecordSystem, and ProgressBar. This modular approach not only improved code organization but also demonstrated how to reuse and extend functionalities efficiently.

# One of the key strengths I identified during this assignment was my ability to implement robust error-handling mechanisms. For example, in both Python and Java, I designed the programs to validate user inputs, prevent duplicate entries, and provide clear feedback for invalid operations. Additionally, my focus on modular design ensured that each function or class was dedicated to a single responsibility, improving code readability and maintainability.

# However, I also recognized areas for improvement. One such area is optimizing data structures for scalability, particularly in the Java program, where the choice of a HashMap was suitable for the current scope but might require re-evaluation for larger datasets or more complex queries. Additionally, while I implemented systematic testing for both Python and Java programs, I realized that incorporating automated testing frameworks, such as unittest in Python or JUnit in Java, could further enhance the reliability and robustness of my code.

# Overall, this assignment was a rewarding learning experience that strengthened my technical skills, problem-solving abilities, and appreciation for structured programming. It also highlighted the importance of balancing functionality with scalability and maintainability, which are critical aspects of software development in both academic and professional contexts. Moving forward, I aim to refine my understanding of data structures and incorporate more advanced testing methodologies to further enhance the quality of my work.

# Overall was an amazing experience even for me that I am comfortable with both languages, I learn a lot with this assignment and with the classes, the professor was amazing explaining everything and when we had questions they were answered, I came across with some problems but they were solved easily with the help of my colleagues and professor.

# References

1. Refereed Academic Journal:

*Weber, R., Sambasivam, S., Rague, B., & Wolthuis, S. (2017). Java vs. Python Coverage of Introductory Programming Concepts. Information Systems Education Journal, 15(3), 42-54.*

[ERIC](https://files.eric.ed.gov/fulltext/EJ1140879.pdf?utm_source=chatgpt.com)

*This study compares Java and Python in terms of their coverage of introductory programming concepts, providing insights into their effectiveness in teaching object-oriented principles.*

2. Academic Book 1:

*Meyer, B. (1997). Object-Oriented Software Construction (2nd ed.). Prentice Hall.*

*A comprehensive guide to object-oriented programming, discussing design principles, modularity, and systematic testing.*

3. Academic Book 2:

*Litvin, M., & Litvin, G. (2011). Java Methods: Object-Oriented Programming and Data Structures. Skylight Publishing.*

[Amazon](https://www.amazon.com/Java-Methods-Object-Oriented-Programming-Structures/dp/0982477562?utm_source=chatgpt.com)

*This book offers an in-depth introduction to object-oriented programming in Java, covering essential data structures and algorithms.*

4. Academic Book 3:

*Goodrich, M. T., Tamassia, R., & Goldwasser, M. H. (2013). Data Structures and Algorithms in Python. Wiley.*

*An extensive resource on data structures and algorithms, emphasizing object-oriented design patterns in Python.*

5. Additional Reference:

*Marrero, W., & Settle, A. (2005). Testing first: Emphasizing testing in early programming courses. ACM SIGCSE Bulletin, 37(3), 4-8.*

*This paper discusses the importance of systematic testing in programming education, highlighting methodologies that can be applied in both Python and Java.*

# Appendix

Python Codes: [Link to ZIP file]  
Java Codes: [Link to ZIP file]