COMP2511

Tutorial 1

Introduction

- Hello! I'm James (he/him)
- Very confused 3rd year Computer Science/Maths student, briefly did Psychology initially
- Skill issued, dropped COMP4128 a couple hours into doing the first problem set
- First time tutoring/lab assisting this course! Sorry in advance for any hiccups :(
- Probably playing video games whenever I'm not doing uni work...
- Please feel free to stop me whenever you'd like me to clarify something during the tutorial, or if I'm speaking too quickly!

Icebreakers

- If everyone is down, please introduce yourselves, your year/degree and any fun facts you would like to share!

Welcome to COMP2511!

- In previous courses, you learnt how to become more proficient as **programmers**.
  - COMP1531: Working on large-scale projects as a team, web-based programming
  - COMP2521: Exploring various data structures and solving a range of algorithmic problems
- In this course, the focus is on developing your ability as **designers**, in the *context* of programming.

Design Principles

- Design is pretty subjective, so what necessarily facilitates ***good*** code?
- What are some things that we can all agree are desirable in code?
  - At the text level, code that adheres to widely used **conventions** and stylistic **patterns** for readability
  - Logic that is correct, yet **simple** enough to understand
  - **Abstraction** - focusing on how things operate at a high level, rather than being too fussed or dependent on concrete details
  - **Modularisation** - separating responsibilities and logic into different parts, rather than shoving all program logic into one component
  - **Scalability** and **extensibility** whenever requirements evolve
- These are all ideas that you'll run into throughout the course!

Assessments

- **Labs (15%)** - 7 labs, each *manually marked* out of 10. Your overall lab mark is out of 70, leaving a buffer for *one* lab. *No late submissions* are accepted - we take whatever is on your main branch at the time of submission
- **Assignment 1 (15%)** - *individual* assignment
- **Assignment 2 (20%)** - *pair* assignment
- **Assignment 3 (optional, 8% bonus)** - to make up any marks lost in the previous assignments
- **Final Exam (50%)** - *40% hurdle*, at least 50% of the exam's content will be derived straight from tutes/labs

Imperative Programming

- You may be used to programming around defining **procedures** and **functions** to modify and use data in various ways to solve problems.
- These instructions would have a very low level of abstraction, and importance is placed on the specific operations performed on the data, rather than the data itself.
- This is characteristic of the **imperative** programming paradigm, which is likely the paradigm you are used to programming in.

Object-Oriented Programming

- **Object-oriented programming** is a paradigm that revolves around user-defined types called **classes**.
- Classes hold their own data (like structs in C) and methods (i.e. functions) to manipulate that data and potentially interact with other classes.
- An instance of a class is referred to as an **object**.
- The data and methods of objects, and the interactions between different objects then form the logic of a program.

Object-Oriented Programming

- The bulk of this course will be around exploring this paradigm, and how we can use it to design larger-scale systems **effectively**.
- However, that's not to say that OOP is the be-all end-all solution to system design!
- Can you think of some pros of OOP, based on the earlier discussion?
  - Allows users to focus on high-level functionality (**abstraction**) by hiding away concrete details and implementation (**encapsulation**)
  - **Modularisation** by separating different responsibilities into different classes
- How about some cons of OOP?
  - The overhead and memory cost of managing objects could get very large
  - Misuse of the paradigm could make programs even **more** complicated unnecessarily (more on this when we learn about design patterns!)

Java and Gradle

- We will use **Java**, a well-known object-oriented language, for all code in this course.
- Java is a friendly entry-point for programmers getting started with OOP.
- Java shares some similar syntax with C in its static typing and variable declarations.
- All code in Java has to exist within a class.
- Unlike C, Java has automatic **memory management**!
  - This means that you won't have to deal with things like memory leaks, for the most part.
- **Gradle** is used for dependency and build management.

Coding! (and Git revision)

- HelloWorld.java
  - git add, git commit, git push
- Sum.java
  - Inside a new file Sum.java, write a program that uses the Scanner class which reads in a line of numbers separated by spaces, and sums them.
- Shouter.java
  - Inside a new file Shouter.java, write a program that stores a message and has methods for getting the message, updating the message and printing it out in all caps. Write a main() method for testing this class.