CSSE2002/7023

Semester 2, 2021

Programming in the Large

Week 1.1: Course Overview

Programming in the Small

For those of you coming straight from CSSE1001/7030, code you have written up to now has likely been:

1. small
2. written solely by you
3. relatively easy to debug
4. for a limited purpose
5. to exist for a limited time

Programming in the Small

For those of you coming straight from CSSE1001/7030, code you have written up to now has likely been:

1. small
2. written solely by you
3. relatively easy to debug
4. for a limited purpose
5. to exist for a limited time

... but large sofware projects are often none of these.

CSSE2002/7023 — Programming in the Large

CSSE2002/7023 is about learning some practices that will make it easier for your code to scale up in size without becoming a broken, unmaintainable mess (while getting some practice with Java).

CSSE2002/7023 — Course Coordinator

Ahmad Abdel-Hafez (a.abdelhafez@uq.edu.au)

Consultation: Email for appointment.

CSSE2002/7023 — Course Staff

The following are your tutors for this semester:

- Maxwell Miller
- Ella De Lore
- Jason Storey
- Ashine Dissanayake
- Mike Pham
- Huansong Zeng
- Matthew Burton
- Rasmus Jepsen
- Savita Datta
- Anna Truffet
- Connor Mulville
- Yiwen Jiang
- Alexandra Belonogov
- Caleb Wishart
- Tom Cranitch

CSSE2002/7023 — Other Contributions

Lectures, tutorials, and practicals:

- Developed by Richard Thomas, Joel Fenwick, Jessica Korte, and Scott Heath, in combination with Brae Webb and Emily Bennett, Andrew Valantine, and other course staff.
- Derived from materials developed by Larissa Meinecke and Graeme Smith.

CSSE2002/7023 — Learning

The purpose of a university course is that by the end of the course, you (as an individual) have acquired certain skills and knowledge.

Assessment is used to measure skills and knowledge as accurately as possible, but it is not perfect.

We encourage you to learn as much as possible from this course, in addition to the skills and knowledge that are directly assessed. Whilst assessment counts towards a grade for this course, additional skills and knowledge will count towards an entire career.

How to Learn

- Experiment with coding (learning by doing)
  - Try compiling and running examples
  - Try changing parts and see if the effects are what you expect
- Ask questions
  - You can ask questions in the lectures, discussion forums, or consult tutors directly in the practical and tutorial sessions.
- Use Internet resources
  - `stackoverflow.com` is one of the richest programming resources
  - Often `stackoverflow.com` pages are listed at the top of search engine results
  - You can also ask general questions on these sites (but not how to do specific pieces of assessment)

Good Practices

- Keep a question log
  - Make efficient use of tutor time.
  - Don't need to remember it.
- *"Start"* assignments as soon as you get them:
  - Get the spec
  - Read it
  - Note any immediate questions
- Leave time for problems to arise
- Regular study/review
- Read the discussion forum regularly
  - So you know what is there when you need it later.
  - So you know what sort of questions are being asked.
- Be mentally active during lectures — be careful of electronics.
- Talk to student services if you have an ongoing diagnosed issue impacting your studies and you want an SAPD.

CSSE2002/7023 — You

We assume that you have passed CSSE1001/7030 or equivalent, so you can use the following in Python:

CSSE2002/7023 — You

We assume that you have passed CSSE1001/7030 or equivalent, so you can use the following in Python:

- variables
- control flow (loops, ifs)
- functions
- lists, strings, dictionaries
- objects, classes

CSSE2002/7023 — You

We assume that you have passed CSSE1001/7030 or equivalent, so you can use the following in Python:

- variables
- control flow (loops, ifs)
- functions
- lists, strings, dictionaries
- objects, classes

We do not assume:

- that you know Java (at first)

What CSSE2002/7023 isn't

This is not just a Java course — We will be teaching you *some* Java, but knowing everything (or even most things) about Java is not the only goal.

Programming in the Small (again)

For those of you coming straight from CSSE1001/7030, code you have written up to now has likely been:

1. small ($< 10,000$ lines of code)
2. writen solely by you
3. probably relatively easy to debug
4. for a limited purpose (i.e. receive marks)
5. to exist for a limited time

Programming in the Small (again)

For those of you coming straight from CSSE1001/7030, code you have written up to now has likely been:

1. small ($< 10,000$ lines of code)
2. writen solely by you
3. probably relatively easy to debug
4. for a limited purpose (i.e. receive marks)
5. to exist for a limited time

The goal of programming up until now has been to write small amounts of code that works until it is marked.

What does a "large" software project look like?

`Linux kernel, Version 4.1`

- $\approx$ 19.5M lines of code (2015)
- Contributions from more than $13,500$ developers (2015)
- $\approx$ 24 years of development (as at 2015)
- Estimated at $3 Billion cost to redevelop (2011)
- The Linux kernel powers mobile phone OS Android, which according to Google has over 2 Billion monthly users (2017)

(`https://www.phoronix.com/scan.php?page=news_item&px=Linux-19.5M-Stats`)

(`http://linuxcost.blogspot.com/2011/03/cost-of-linux.html`)

(`https://twitter.com/Google/status/864890655906070529`)

You may end up working on software which is:

1. large — 100K lines of code and up
2. writen by many people
3. impossible for one person to understand all at once
4. the purpose and features of which develops over time
5. needs to work for years

Tasks Programmers Do

Professionally, you will need to:

- Document your code
- Debug it
- Test it

Tasks Programmers Do

Professionally, you will need to:

- Document your code — because someone who didn't write it (or doesn't remember it) may need to work on it.
- Debug it
- Test it

Tasks Programmers Do

Professionally, you will need to:

- Document your code — because someone who didn't write it (or doesn't remember it) may need to work on it.
- Debug it — the more code you write the more opportunity to make mistakes.
- Test it

Tasks Programmers Do

Professionally, you will need to:

- Document your code — because someone who didn't write it (or doesn't remember it) may need to work on it.
- Debug it — the more code you write the more opportunity to make mistakes.
- Test it — it is **far** better for you to find bugs than for someone else to find them. You want to have a reason for confidence in your work beyond "I hope it is okay".

Tasks Programmers Do

Professionally, you will need to:

- Document your code — because someone who didn't write it (or doesn't remember it) may need to work on it.
- Debug it — the more code you write the more opportunity to make mistakes.
- Test it — it is **far** better for you to find bugs than for someone else to find them. You want to have a reason for confidence in your work beyond "I hope it is okay".

CSEE2002/7023 teaches practices that support documenting, debugging, and testing of code to allow programming to scale to large software projects.

Cheating and Collusion

All code which you submit for assessment must be, either

- supplied by teaching staff **for this run** of the course.
  OR
- written entirely, individually by you.

All code you submit will be checked for collusion and plagiarism (including but not limited to using MoSS). If similarity is found, a school investigation may be started (these are not fun).

Seek help from course staff in plenty of time if you get into trouble.

Be very careful when seeking outside help — outside tutors may facilitate cheating (and get you caught) rather than your learning.

Misconduct — Things *Not* to Do

1. Do not show your code to other students.
2. Do not look at other student's code.
3. Do not use code given to you in other semesters or courses.
4. Do not **try** to get other people to write code for you.
5. Do not store your code in any online repository which **could** be accessible to others. (This will be deemed as if you have broken Item 1).
   - e.g. GitHub, BitBucket, ...
6. Do not start a few days before the deadline and then claim that cheating was the only way to get it done in time.

Contact Hours

Each week:

Activity	Hours	Content
Lectures	2	Theory and programming demonstration
Tutorial	1	Theory exercises
Practicals	2	Programming exercises

Tutorials and Pracs start in **Week 2**.

Discussion in Pracs

Pracs involve:

- Exercises
- Assignment Help

You can discuss exercises, but the aim is to help you learn for the assignments. Your group knowing how to do the exercises is not a substitute for individual competance.

Assignments **must** be solo – no discussions in pracs or elsewhere.

Assessment

CSSE2002/7023	
MyJavaTutor Problems	10%
Assignment 1	20%
Assignment 2	20%
Exam (Final)	50%

Read the ECP for dates and rules.

Course Tools

If you'd like to get a head start, feel free to download these onto your personal machines:

- IntelliJ IDEA
- Java Development Kit (Open JDK) 11. Java 11 is the current long-term supported version of Java (until 2026).