CSSE1001 — Introduction to Software Engineering

Table of contents

Overview . . . 1
What the course means by its own title . . . 1
Staff . . . 2
Course structure (by week) . . . 3
Learning outcomes . . . 4
Assessment . . . 4
Academic integrity . . . 6
Course resources . . . 6
Setting expectations . . . 7
Lectures . . . 7

Overview

Introduction to Software Engineering through programming, with a focus on the fundamentals of computing and programming using an exploratory, problem-based approach: building abstractions with procedures, data and objects; data modelling; designing, coding, and debugging programs of increasing complexity. Taught in Python. The GUI module has been removed from the scope of this course to leave more time for fundamentals.

No prior programming knowledge is assumed. Incompatible with COMP1502, CSSE7030, ENGG1001.

What the course means by its own title

The very first hour of the course defines its terms, and the definitions chain into one another. Each is a tentative working definition, not a formal claim. Knowing the chain is what makes

the shape of the semester legible: the course is *named* after software engineering, but it ends on computer science.

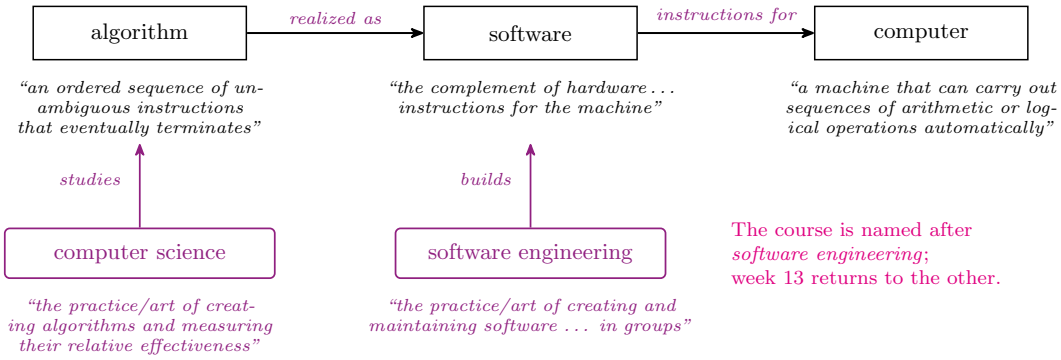

Figure 1: The five definitions and how they hang together. Software is realized algorithms, and software is instructions for the machine. Computer science *studies* algorithms; software engineering *builds* software.

Two things follow, and both show up in the assessment table below. **Software engineering** is the *in groups* discipline, which is why the course is built around programs you write, submit and maintain rather than theory you prove. And **computer science** is the *measuring their relative effectiveness* discipline, which is exactly the brief of week 13 — sorting-algorithms[1] for the creating, algorithmic-complexity[2] for the measuring. That is why the two carve-outs from assessment are the first hour and the last week: they are the two ends of the same sentence, and everything examinable sits between them.

The full treatment, with the history around the definitions, is history-of-computing[3]; the lecture note is 2025-07-29-a-brief-history-of-computing[4].

Staff

- **Dr Paul Vrbik** — course coordinator & lecturer — p.vrbik@uq.edu.au / CSSE1001@eecs.uq.edu.au — 78-204C — office hours by appointment. PhD Computer Science (Western Ontario, computational algebraic geometry); at UQ since 2020. Lectures weeks 1-6 and 12-13.
- **Dr Kasra Khosoussi** — coordinator & lecturer — k.khosoussi@uq.edu.au — 78-348 — office hours by appointment. PhD Robotics (UTS); previously CSIRO Data61 and MIT. Lectures weeks 7-11.

[1] courses/csse1001/sorting-algorithms.pdf
[2] courses/csse1001/algorithmic-complexity.pdf
[3] courses/csse1001/history-of-computing.pdf
[4] courses/csse1001/2025-07-29-a-brief-history-of-computing.pdf

Course structure (by week)

The semester runs in three phases. Each lecture slot is two hours, one topic per hour; the "covers" column says which assessment each week feeds.

Imperative programming

Week	Starting	Hour 1	Hour 2	Covers
1	28 Jul	History	Arithmetic	A1, MT, EX (excluding hour 1)
2	4 Aug	Memory model (variables)	Data (primitive)	A1, MT, EX
3	11 Aug	Functions	Control structures (if-statements)	A1, MT, EX
4	18 Aug	Control structures (while-loops)	Data (non-primitive)	A1, MT, EX
5	25 Aug	For-loops	Comprehensions	A1, MT, EX
6	1 Sep	IO & string methods	Testing & debugging	A1, EX

Object-oriented programming

Week	Starting	Hour 1	Hour 2	Covers
7	8 Sep	Exceptions, scope	Intro to OOP	A2, EX
8	15 Sep	Dunder methods	Representation invariants	A2, EX
9	22 Sep	Composition	Inheritance 1	A2, EX
Break	29 Sep	*(break)*	*(break)*	—
10	6 Oct	Unified Modelling Language	Inheritance 2	A2, EX
11	13 Oct	Design patterns, MVC	Sudoku GUI	A2, EX

Advanced

Week	Starting	Hour 1	Hour 2	Covers
12	20 Oct	Recursion	Recursion	EX
13	27 Oct	Functional programming	Complexity	**None (not testable)**
14	3 Nov	Revision week	Revision week	—

There are two carve-outs: **week 1 hour 1 (history) is excluded from assessment**, and **all of week 13 is explicitly not testable**. Both are still worth reading — week 13 is where the course's opening definition of computer science finally pays off — but they shouldn't compete with revision for exam marks.

Applied classes (from week 1) and practicals (from week 2) run alongside lectures, giving guided/self-directed practice with demonstrator feedback. All formative, no grades. 11-12 formative Ed Lessons tasks (immediate feedback, optional) and Gradescope (immediate feedback on assignment submissions) support self-study.

Cohort: ~428 students, 14 teaching assistants, 2 instructors, 2 coordinators.

Learning outcomes

After successfully completing this course you should be able to:

1. Apply program constructs such as variables, selection, iteration, and sub-routines.
2. Apply basic object-oriented concepts such as classes, instances, and methods.
3. Read and analyse code written by others.
4. Analyse a problem and design an algorithmic solution to it.
5. Read and analyse a design and translate it into a working program.
6. Apply techniques for testing and debugging.

Assessment

Assessment	Weight	Due	Covers
Assignment 1	15%	19/09/2025, 3:00pm	Weeks 1-4 (functions, lists, tuples, dictionaries, loops, if-statements)
In-semester exam (Saturday)	25%	Sat 13 Sep 2025, 7:15pm, 90 min	Up to week 5
Assignment 2	20%	24/10/2025, 3:00pm	Weeks 1-11
End-of-semester exam	40%	End-of-semester exam period (8/11/2025 - 22/11/2025), 120 min	Entire course. **Hurdle** — see the grade cutoffs below.

Both exams are closed-book, identity-verified, in-person, and paper-based (calculator: Casio FX82 series or UQ-approved/labelled calculator only).

Some lecture questions are real past-exam questions. Lecture 1B has two, both multiple-choice on operator precedence — `2 ** 3 % 5 - 1` and `x = 35 * 2 % 5 ** 2`. On the second, **60% of the class answered correctly**. Treat them as calibration: see python-operator-precedence-and-associativity[5] for how they are worked.

Grade cutoffs and the exam hurdle

The final exam is a **hurdle that escalates with the grade** — it is not a single pass mark. Writing M for the computed mark and E for the final exam percentage:

Grade	Mark	Final exam hurdle
7 (High Distinction)	85-100	$E \geq 75\%$
6 (Distinction)	75-84	$E \geq 65\%$
5 (Credit)	65-74	$E \geq 55\%$
4 (Pass)	50-64	$E \geq 47\%$
3 (Marginal Fail)	47-49	$E \geq 47\%$
2 (Fail)	20-46	—
1 (Low Fail)	1-19	—

So a mark of 80 with a 60% exam is a **5**, not a 6. Final grades and marks may be scaled *upwards*, but not downwards.

Grading formula

With A_1, A_2, EM, EF as percentages for Assignment 1, Assignment 2, the in-semester exam, and the end-of-semester exam respectively:

$$\text{MARK} = \max(M_1, M_2)$$

where

$$M_1 = 0.15A_1 + 0.20A_2 + 0.25EM + 0.40EF, \qquad M_2 = 0.15A_1 + 0.20A_2 + 0.10EM + 0.55EF$$

So a poor in-semester exam result can be mitigated by a strong final exam. The mark is computed to two digits of precision, then rounded **up**.

[5] courses/csse1001/python-operator-precedence-and-associativity.pdf

Previous cohort results (grade distribution)

Term	7	6	5	4	3	≥ 3
2024 S1	12%	21%	28%	12%	7%	80%
2023 S2	9%	12%	17%	28%	4%	70%
2022 S2	12%	16%	23%	30%	4%	85%
2021 S2	10%	12%	17%	25%	6%	70%

Assignments are submitted online via Gradescope. Extensions: up to 7 days maximum (in multiples of 24 hours), handled by the school (not the lecturer) — a medical certificate is required, with no exceptions. Work must be *received* by the deadline, not just transmitted; there is an undisclosed small grace-period buffer so work only a few minutes late isn't penalised. Beyond that, late assignment submissions receive 0%.

Academic integrity

- Don't cheat — penalties are harsh, and detection tools are sophisticated.
- Students who maximise assessment scores by using ChatGPT, abusing staff support, or cheating typically fail the exam.
- You may discuss solutions to problems with others, but must not share code — including for assignments.

Course resources

- **IDLE** — the IDE officially supported by tutors.
- **Ed Discussion** — the course discussion board.
- **Ed Lessons** — the formative programming tasks with immediate feedback.
- **Gradescope** — assignment submission, with immediate correctness feedback.
- **Open Help Center** (a.k.a. "ITLC"/"EECSLC") — 78-217 — Mon, Tue, Wed, Fri 10am-4pm.
- Individual tech help — csse1001@helpdesk.eait.uq.edu.au

The **Electronic Course Profile is the source of truth** if it ever disagrees with the lecture material.

Setting expectations

- Lectures are only a **first exposure** to a topic — you're expected to do most of your learning by completing assigned tasks, and to revise both before and after lectures.
- Secondary school vs. university: 20 weeks / 5 hrs-per-week / 100 hrs instruction / 30 classmates (secondary) vs. 12 weeks / 2 hrs-per-week / 24 hrs instruction / 1000 classmates (this course). Teachers *teach* you; instructors *guide* you.
- Course staff are human — they will sometimes make errors, be terse, and are often sleep-deprived, despite trying hard to provide a good learning experience.
- **How to succeed**: (1) practice makes perfect, (2) learn to debug, (3) do well on the exam.

Lectures

- 2025-07-29-a-brief-history-of-computing[6] — Lecture 1A
- 2025-07-29-python-as-a-calculator[7] — Lecture 1B
- 2025-08-04-python-memory-model[8] — Lecture 2A
- 2025-08-04-primitive-data[9] — Lecture 2B
- 2025-08-11-functions[10] — Lecture 3A
- 2025-08-11-sequence-selection-and-iteration[11] — Lecture 3B
- 2025-08-18-while-loops[12] — Lecture 4A
- 2025-08-18-non-primitive-data[13] — Lecture 4B
- 2025-08-25-for-loops[14] — Lecture 5A
- 2025-08-25-comprehensions[15] — Lecture 5B
- For Loop Practice[16] — Lecture 5C
- 2025-09-01-string-methods[17] — Lecture 6A
- 2025-09-01-file-io[18] — Lecture 6B
- 2025-09-01-testing[19] — Lecture 6C
- 2025-09-09-exceptions[20] — Lecture 7A

[6] courses/csse1001/2025-07-29-a-brief-history-of-computing.pdf
[7] courses/csse1001/2025-07-29-python-as-a-calculator.pdf
[8] courses/csse1001/2025-08-04-python-memory-model.pdf
[9] courses/csse1001/2025-08-04-primitive-data.pdf
[10] courses/csse1001/2025-08-11-functions.pdf
[11] courses/csse1001/2025-08-11-sequence-selection-and-iteration.pdf
[12] courses/csse1001/2025-08-18-while-loops.pdf
[13] courses/csse1001/2025-08-18-non-primitive-data.pdf
[14] courses/csse1001/2025-08-25-for-loops.pdf
[15] courses/csse1001/2025-08-25-comprehensions.pdf
[16] courses/csse1001/week-five-exercises.pdf
[17] courses/csse1001/2025-09-01-string-methods.pdf
[18] courses/csse1001/2025-09-01-file-io.pdf
[19] courses/csse1001/2025-09-01-testing.pdf
[20] courses/csse1001/2025-09-09-exceptions.pdf

- 2025-09-09-scope[21] — Lecture 7B
- 2025-09-09-object-oriented-programming[22] — Lecture 7C
- 2025-09-16-dunder-methods[23] — Lecture 8A
- 2025-09-16-representation-invariants[24] — Lecture 8B
- 2025-09-23-composition[25] — Lecture 9A
- 2025-09-23-inheritance[26] — Lecture 9B
- 2025-10-07-uml[27] — Lecture 10A
- 2025-10-07-advanced-inheritance[28] — Lecture 10B
- 2025-10-14-model-view-controller[29] — Lecture 11A
- 2025-10-20-recursion[30] — Lecture 12A
- Towers of Hanoi[31] — supplementary, alongside Lecture 12A
- 2025-10-28-complexity-and-sorting[32] — Lecture 12B (week 13)

Lecture 0 (Welcome) is not a separate note — its content is the logistics on this page. Week 13's functional programming lecture has no note yet.

[21] courses/csse1001/2025-09-09-scope.pdf
[22] courses/csse1001/2025-09-09-object-oriented-programming.pdf
[23] courses/csse1001/2025-09-16-dunder-methods.pdf
[24] courses/csse1001/2025-09-16-representation-invariants.pdf
[25] courses/csse1001/2025-09-23-composition.pdf
[26] courses/csse1001/2025-09-23-inheritance.pdf
[27] courses/csse1001/2025-10-07-uml.pdf
[28] courses/csse1001/2025-10-07-advanced-inheritance.pdf
[29] courses/csse1001/2025-10-14-model-view-controller.pdf
[30] courses/csse1001/2025-10-20-recursion.pdf
[31] courses/csse1001/towers-of-hanoi.pdf
[32] courses/csse1001/2025-10-28-complexity-and-sorting.pdf