COMP 333: Concepts of Programming Languages
Fall 2025

Instructor: Kyle Dewey (kyle.dewey@csun.edu)
Course Web Page: https://kyledewey.github.io/comp333-fall25
Office: JD 4419

Course Description (From the Catalog)
Discussion of issues in the design, implementation and use of high-level programming languages through a historical framework, including how languages reflect different design philosophies and use requirements and the technical issues in the design of main abstraction constructs of programming languages. Other approaches to imperative or object-oriented programming, functional programming, logical programming and parallel programming.

Learning Objectives:
Successful students will be able to:
- Understand when to use, and write programs using:
  - Execution approaches: compilation, interpretation, and just-in-time (JIT) compilation
  - Memory management: manual, reference counting, garbage collection, ownership and borrowing
  - Types: dynamic typing, static typing, strong typing, weak typing, untyped
- Write programs utilizing class-based and prototyped-based inheritance, virtual dispatch, and higher-order functions
- Write memory diagrams representing the connections between allocated values, differentiate between stack-allocated and heap-allocated values, and understand when to use stack vs. heap allocation.

Course Motivation and Goal
Programming languages, like the tools in a typical toolbox, are built to solve problems. A toolbox may have different sizes and shapes of both hammers and screwdrivers. Similarly, different programming languages may be closely related to each other, or potentially very different from each other. The more different a language or tool, the more different the kind(s) of problem(s) it is designed to solve.

The danger of getting too close to one language or programming paradigm is that your thinking adapts to fit that language/paradigm. In keeping with the toolbox analogy, if you have a hammer, then all problems become nails. I may insist on fixing a leaky pipe with a sledgehammer, but I won't get back my security deposit.

My primary goal with this course is to expand your toolbox, and expose you to languages which behave very differently from each other. My intention is to warp your brain a bit, and force you to think in ways you're not used to. This will improve your

problem-solving skills, specifically by allowing you to look at the same problem from different angles.

A secondary goal is to give you a sense of how different languages are built, and how they work. We will focus primarily on modern language design and implementation, though many basic concepts haven't changed much in the 60+ year history of programming languages.

Textbook and Other Required Class Materials
No textbooks are required. You may wish to look at Programming Language Pragmatics (Michael Scott) as a reference, though the course does not follow that book. A computer, be it a laptop or otherwise, is required.

Grading
Your grade is based on the following components:

Assignments	14%
Midterm Exam 1	27%
Midterm Exam 2	27%
Final Exam	32%

There will be a series of coding-based assignments issued throughout the semester, which cover core parts of the different languages and paradigms you'll use. Not all of these will be weighted evenly, nor will you always be given the same amount of time for assignments. Exactly which assignments are assigned depends on how the class progresses. In general, assignments will be submitted through Canvas (https://canvas.csun.edu/). In the event that there is a problem with Canvas, you may email your assignment to me (kyle.dewey@csun.edu) as a last resort.

Plus/minus grading is used, according to the scale below:

If your score is >=...	...you will receive...
92.5	A
89.5	A-
86.5	B+
82.5	B
79.5	B-
76.5	C+
72.5	C

If your score is >=...	...you will receive...
69.5	C-
66.5	D+
62.5	D
59.5	D-
0	F

If you are not present for the final exam and you have not previously made alternative arrangements with me for the final exam, a grade of WU (unauthorized withdrawal) will be assigned.

Collaboration for Assignments
All students are required to submit their own individual work. For assignments (and **only** assignments), students may discuss among each other, as long as they don't digitally share code. That is, you **cannot** simply email your code to someone else. However, you **may** discuss your actual code with someone else, including merely viewing code. The only stipulation is that **if you do discuss with someone else, say so in your submission.** This is not for punitive reasons; this is only so I get a sense of who is working with who. My intention with this policy is to enable collaborative learning, as opposed to simply sharing a solution.

Plagiarism and Academic Honesty
While collaboration is allowed on assignments, you are responsible for all of your own work. You may **not** take code from online sources and submit it as your own. If you must take code from online, cite where you took the code from. Worst-case scenario, you'll receive a 0 for whatever you took, but no further action will be taken. In general, code taken online which solves more general things (e.g., "how do I iterate through an array in Java") is more acceptable than code which solves more specific things (e.g., "how do I implement a recursive find function over immutable linked lists in JavaScript"). General bits of code only give you pieces of a solution, whereas specific bits of code often will give you a complete copy/pastable solution. If it's not 100% clear if something is permitted to be used or not, you can always ask me beforehand.

Chegg is specifically disallowed as an online resource, as it's almost always used as a repository of complete questions with answers. That is, the questions/answers are practically always of the specific kind mentioned above.

For assignments, LLM-based tools like ChatGPT are discouraged, though not outright disallowed. It can be easy to overly on such tools without realizing it, to the point where the tool does all the work, with zero understanding of what it produces. If you do use such tools, you should be aware of the following trap I've seen a lot of students fall into over the past year:

- The student gets a solution from an LLM
- The student tests the solution, and observes that all tests pass
- The student looks at the code a bit, and decides that it "feels" right. This feeling is usually based on the relative length, complexity, formatting, and variable naming of the code, and not on its actual behavior.
- The student bombs a high-scoring, closely-related exam question, which asked them to write similar code for a similar problem.
- The student realizes they did not actually understand what the LLM produced, and then needs additional help to hopefully catch up.

In general, for this course, my opinion is that if an LLM is to be used, it's best to use prompts that are general (e.g., how does virtual dispatch in Java work?) as opposed to specific (e.g., write a Java program using virtual dispatch that solves the specific problem I've been asked to solve). A course-specific pitfall is that LLMs have been trained over a lot of sources which are just plain wrong, leading to confidently incorrect LLM answers.

No discussion whatsoever is allowed during exams, except with the instructor. Any violations can result in a failing grade for the assignment/exam, or potentially failing the course for egregious cases. A report will also be made to the Dean of Academic Affairs. Students who repeatedly violate this policy across multiple courses may be suspended or even expelled.

Communication
In general, any questions should be made through Canvas. You can also email me, though I'm usually much faster to respond to Canvas than my general email.

Late Policy / Exam Scheduling
Late assignments will be accepted without penalty if prior arrangements have been made or there is some sort of legitimate emergency (at my discretion). If you must be absent from an exam, contact me ASAP to see if alternative accommodations can be made.

If an assignment is otherwise submitted late, it will be penalized according to the following scale:

If your assignment is late by <= this many days...	...it will be deducted by...
1	10%
2	30%
3	60%
4+	100%

To be clear, assignments which are submitted four or more days beyond the deadline will not receive credit. The reason for such a harsh late policy is that we will generally discuss solutions in class shortly after the deadline, and this late policy discourages people from simply pulling a solution from an in-class discussion.

Class Feedback

I am open to any questions / comments / concerns / complaints you have about the class. If there is something relevant you want covered, I can push to make this happen. I operate off of your feedback, and no feedback tells me "everything is ok".

Class Schedule (Subject to Change):

Week	Monday	Wednesday
1	8/25: Introduction, motivation	8/27: OOP introduction with Java, using inheritance to avoid code duplication
2	~~9/1: Labor day (campus closed)~~	9/3: Using inheritance to avoid code duplication, class-based inheritance, subtyping, virtual dispatch
3	9/8: class-based inheritance, subtyping, virtual dispatch	9/10: Functional/immutable lists: representation and operations
4	9/15: Class-based OOP spillover	9/17: JavaScript introduction, types introduction (static vs. dynamic, strong vs. weak, untyped)
5	9/22: Types; Higher-order functions: use and high-level representation	9/24: Higher-order functions: use and high-level representation
6	9/29: Higher-order functions: use and high-level representation	10/1: Higher-order functions: use and high-level representation
7	10/6: Midterm Exam 1 Review	10/8: **Midterm Exam 1**
8	10/13: Midterm Exam 1 Retrospective; Objects and prototype-based inheritance	10/15: Objects and prototype-based inheritance
9	10/20: Objects and prototype-based inheritance	10/22: Objects and prototype-based inheritance
10	10/27: Objects and prototype-based inheritance; interpretation vs. compilation	10/29: interpretation vs. compilation, Just-in-time compilation; stack vs. heap-allocation

Week	Monday	Wednesday
11	11/3: Memory reclamation (manual, reference counting, garbage collection, ownership and borrowing)	11/5: Memory reclamation spillover
12	11/10: Introduction to Rust; Ownership and borrowing in Rust	11/12: Ownership and borrowing in Rust
13	11/17: Midterm 2 Review	11/19: **Midterm Exam 2**
14	11/24: Midterm Exam 2 Retrospective	11/26: Ownership and borrowing in Rust
15	12/1: Enums and pattern matching in Rust	12/3: Enums and pattern matching in Rust
16	12/8: Rust Spillover	12/10: Final Exam Review

Final Exams:

- Section 16308 (11:30 AM - 12:45 PM): 12/17, 10:15 AM - 12:15 PM, in Noski Auditorium 101
- Section 16886 (4:00 PM - 5:15 PM): 12/15, 5:30 PM - 7:30 PM, in JD 3510