# COMP 333: Concepts of Programming Languages
# Fall 2021

**Instructor:** Kyle Dewey (kyle.dewey@csun.edu)
**Course Web Page:** https://kyledewey.github.io/comp333-fall21
**Office**: JD 4419 (I will not physically be there); Zoom link for office hours posted on Canvas and available via email.

**Special COVID-19 Message**
The course is being run as a synchronous online course. We will never meet in person, though we will meet virtually via Zoom regularly each week at the assigned time (see SOLAR for meeting time details). Assuming you're enrolled in the course or on the waitlist, I have emailed the Zoom link to you. If somehow you do not have the Zoom link, email me at kyle.dewey@csun.edu. For exams, if you cannot be on Zoom during the scheduled exam time, let me know ahead of time for alternative arrangements.

The synchronous lectures will be recorded and made available through Canvas, in case students want to view them asynchronously. These lectures will only be accessible to CSUN students either enrolled or waitlisted for the course. However, **by enrolling in, or waitlisting this course, you consent to having any voice or webcam recorded.** That said, **I will never require you to use your webcam or speak**; only what you voluntarily send will be recorded. Questions can either be asked verbally or textually through the Zoom chat. I will verbally repeat any questions in the chat before answering them, but I will not identify who said the question. Even so, it's possible that your name will end up in the meeting recording, identifying you as a participant (Zoom sometimes unavoidably shows participant names in the recording).

**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: garbage collection, reference counting
  - Types: dynamic typing, static typing, strong typing, weak typing, untyped
  - Parameter passing: call-by-value, call-by-name
- Read context-free grammars, construct abstract syntax trees, and parse programs
- Write object-oriented programs using:

  - Inheritance: class-based, prototype-based
  - Virtual dispatch
- Write functional programs using:
  - Higher-order functions
  - Algebraic data types and pattern matching
  - Generics and parametric polymorphism
- Write logic programs using:
  - Unification
  - Nondeterminism
- Write parallel programs using Java's parallel streams

**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 to close to one language or programming paradigm is that your thinking adapts to fit that language/paradigm. In keeping with the toolbox analogy, you have a hammer, and 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 | 36% |
| Midterm Exam 1 | 17% |
| Midterm Exam 2 | 17% |
| Final Exam | 30% |

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 |
| 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 Swift"). 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.

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.

**Attendance**
In the first week of class, I will take attendance. If you miss both sessions in the first week and have not made alternative arrangements with me, you must drop the class, as per University policy (http://catalog.csun.edu/policies/attendance-class-attendance/). After the first week I will not take attendance, and attendance is not mandatory, though you are strongly encouraged to attend. I enforce this policy in order to help students on the waitlist get into the course.

**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 | Tuesday | Thursday |
|---|---|---|
| 1 | 8/31: Introduction, motivation | 9/2: OOP introduction with Java, using inheritance to avoid code duplication |
| 2 | 9/7: Class-based inheritance, subtyping, virtual dispatch | 9/9: Functional/immutable lists: representation and operations |
| 3 | 9/14: Class-based OOP spillover | 9/16: JavaScript introduction, types introduction (static vs. dynamic, strong vs. weak, untyped) |
| 4 | 9/21: Higher-order functions: use and high-level representation | 9/23: Prototype-based inheritance |
| 5 | 9/28: Prototype-based inheritance | 9/30: JavaScript spillover |

| Week | Tuesday | Thursday |
|---|---|---|
| 6 | 10/5: JavaScript spillover, midterm 1 review | 10/7: **Midterm Exam 1** |
| 7 | 10/12: Midterm 1 retrospective, interpretation, compilation, just-in-time compilation, garbage collection and reference counting | 10/14: Garbage collection and reference counting, introduction to functional programming in Swift |
| 8 | 10/19: First-order and higher-order functions in Swift | 10/21: First-order and higher-order functions in Swift |
| 9 | 10/26: Algebraic data types, pattern matching | 10/28: Algebraic data types, pattern matching |
| 10 | 11/2: Generics, parametric polymorphism | 11/4: Generics, parametric polymorphism |
| 11 | 11/9: BNF grammars, abstract syntax trees | ~~11/11~~: Veteran's day (no class) |
| 12 | 11/16: Swift spillover, midterm 2 review | 11/18: **Midterm Exam 2** |
| 13 | 11/23: Midterm 2 retrospective, Introduction to logic programming in Prolog | ~~11/25~~: Thanksgiving (no class) |
| 14 | 11/30: Nondeterminism, backtracking | 12/2: Nondeterminism, backtracking |
| 15 | 12/7: Structures, unification | 12/9: Structures, unification |

Final Exams:

- Section 1 (11:00 AM - 12:15 PM): 12/14, 10:15 AM - 12:15 PM
- Section 2 (12:30 PM - 1:45 PM): 12/16, 12:45 PM - 2:45 PM
- Section 3 (2:00 PM - 3:15 PM): 12/16, 3:00 PM - 5:00 PM