Programming Languages — Theory & Application

This is the site for “CSE 320: Programming Languages” as taught by Andrew Brinker in the Fall quarter of 2017 at CSU, San Bernardino.

Expectations ↓

1. Attend all class and lab sessions. If you are unable to attend, reach out to me promptly so we can work together to keep you up to speed.
2. Participate in class discussions. This class will be collaborative and explorational, and that is best facilitated when everyone participates. You don't need to do a lot, but share your ideas, and ask questions when you have them.
3. Do the reading before each class. Do the optional reading if you can. You’ll get a lot more out of the class with the optional reading.
4. If you have a problem, reach out to me. I am here as a resource to you, and my number one goal is making sure you learn the material and succeed in the course. The sooner you reach out, the better we will be able to address the problem.

Meeting Times ↓

Classes are held every Tuesday and Thursday, lectures are from 4:00pm to 5:15pm, labs are from 5:25pm to 6:20pm.

Materials ↓

This course requires the second edition of "Programming Languages: Application and Interpretation" by Shriram Krishnamurthi. This book is available for free online in both HTML and PDF formats, and need not be purchased.

Each pre-course reading will include optional materials. Questions from these optional materials will be included for extra points in the regular quizzes.

Some pre-course reading will include links or suggestions for further exploration into the topic at hand. These are entirely for your own personal edification, and will not be required for use in the course.

Structure ↓

The class component will serve to enable discussion and exploration of the topics covered in reading. To do well in this class, *you must do the assigned reading*. Students who arrive to class without having done the reading will likely find themselves struggling.

The lab component will serve to introduce you to a variety of programming languages, to provide practical knowledge about different languages, their underlying paradigms, and how the different decisions the language designers

make influence the resulting language.

Grading ↓

There are four ways to get points: quizzes, labs, projects, and exams. There are a total of 1000 points available, and your final grade is based solely on how many points you have.

Type	Count	Points Per Assignment	Total
Quizzes	10	10	100
Labs	10	20	200
Projects	3	100	300
Exams	2	200	400

Late work is not generally accepted. If special circumstances prevent you from doing an assignment, please contact me promptly so we can discuss potential solutions.

Reading ↓

Read Before Class On	Reading

09/26	Interpretation
09/28	Functions 1
10/03	Functions 2
10/05	Mutation
10/10	Recursion
10/12	Objects 1
10/17	Objects 2
10/19	Memory 1
10/24	Memory 2
10/26	Representation 1
10/31	Representation 2
11/02	Control Flow 1
11/07	Control Flow 2
11/09	Types 1

11/14	Types 2
11/16	Types 3
11/21	Types Review

Interpretation

The focus in this reading should be on the following concepts:

- What is a parser?
- What is an interpreter?
- What is desugaring?
- What are reasons to have a core language that all extra language constructs desugar into?
- What does it mean for recursion to be generative?

1. **Required:** Chapters 1, 2, 3, and 4 of PLAI. (19 pages). It is highly recommend to do the exercises in the book.
2. **Required:** How to Talk About Programming Languages by Lionel Barrow.
3. **Required:** "7 lines of code, 3 minutes: implement a programming language from scratch" by Matthew Might.
4. **Optional:** Chapters 4, 5, 6 of "Crafting Interpreters", by Robert Nystrom.
5. **Advanced:** "Monadic Parsing in Haskell" by Graham Hutton and Erik Meijer.
6. **Advanced:** "LL and LR Parsing Demystified" by Josh Haberman.

7. **Advanced:** "LL and LR in Context: Why Parsing Tools Are Hard" by Josh Haberman.
8. **Advanced:** "Parsing with Derivatives" by Matthew Might.
9. **Advanced:** "Packrat Parsing: Simple, Powerful, Lazy, Linear Time" by Bryan Ford.

Functions 1

The focus in this reading should be on the following concepts:

- What are function application and function definition?
- What are formal parameters and actual parameters?
- What is substitution?
- What are free identifiers and bound identifiers?
- What is the environment?
- What is static scope? What is dynamic scope?

1. **Required:** Chapters 5 and 6 of PLAI. It is highly recommend to do the exercises in the book.
2. **Required:** This series of slides about static vs. dynamic scoping.
3. **Optional:** "(How to Write a (Lisp) Interpreter (in Python))" by Peter Norvig.
4. **Advanced:** "Beyond Static and Dynamic Scope" by Eric Tanter.

Functions 2

The focus in this reading should be on the following concepts:

- What are closures? How are they implemented?
- How do functions and closures relate? Are functions closures? Are closures functions?
- What is the "top-level" of a program?
- What is *capture-free substitution*? Why is it necessary?
- What is `let`? How is it defined?

1. **Required:** Chapter 7 of PLAI. It is highly recommend to do the exercises in the book.
2. **Optional:** Section 2.2.1 of "The Implementation of Functional Programming Languages" by Simon Peyton Jones.
3. **Advanced:** "A lambda is not (necessarily) a closure" by Andy Wingo.

Mutation

The focus in this reading should be on the following concepts:

- What is a box? What is it used for?
- Why does adding sequencing require the interpreter to take and return the environment?
- What is the "store," and why is it needed? How does adding the store change the environment?
- How does adding state change the semantics of existing operations, like addition and multiplication?
- What is the difference between identifiers and variables?
- What are "call by value" and "call by reference?" What is the difference between them?

1. **Required:** Chapter 8 of PLAI. It is highly recommend to do the exercises in the book.
2. **Optional:** "The Essence of Functional Programming" by Philip Wadler.

Recursion

The focus in this reading should be on the following concepts:

- What is a recursive data structure?
- What is a cyclic data structure?
- What does it mean for a computation to *diverge*?
- Why does creating a cyclic datum require a box?
- Why can't you write recursive functions without boxing?

1. **Required:** Chapter 9 of PLAI. It is highly recommend to do the exercises in the book.
2. **Optional:** Daniel P. Friedman and Matthias Felleisen explaining how to get recursion without explicit state, in this chapter from their book *The Little Schemer*. Note that you will probably need to read this at least twice to understand what they're describing. This is some trippy stuff.
3. **Advanced:** "Functional Programming with Bananas, Lenses, Envelopes and Barbed Wire" by Erik Meijer (major contributor to C#, Haskell, lead developer of LINQ, one of the guys from the video on the first day of class), Maarten Fokkinga, and Ross Paterson.
4. **Advanced:** If the above is a bit much, try "An Introduction to Recursion Schemes" by Patrick Thomson, which summarizes the above paper in a more approachable way.

Objects 1

The focus in this reading should be on the following concepts:

- The relationship between functions and objects.
- How objects require mutation and recursion, to enable access to `self`.
- How languages can choose to treat names of objects.

1. **Required:** Sections 10.1 and 10.2 and chapter 10 of PLAI. It is highly recommend to do the exercises in the book.
2. **Optional:** Chapters 1, 2, and 3 of "Object-Oriented Programming Languages: Application and Interpretation" by Eric Tanter.

Objects 2

The focus in this reading should be on the following concepts:

- What is the difference between classes and prototypes?
- Why is multiple inheritance considered a bad idea?
- What is the difference between *replacing* and *refining* when considering inheritance?
- What are mixins? How do they differ from inheritance?
- How do mixins differ from traits?

1. **Required:** Section 10.3 of chapter 10 of PLAI. It is highly recommend to do the exercises in the book.
2. **Optional:** Chapters 4, 5, 6, and 7 of "Object-Oriented Programming Languages: Application and Interpretation" by Eric Tanter.

Memory 1

The focus in this reading should be on the following concepts:

- What does it mean for memory management to be complete and sound?
- What is fragmentation?
- What is a free-list?
- What does it mean to trade space for time?
- What is padding?
- What is reference counting?
- What happens if a reference counting mechanism does not track and break references cycles?

1. **Required:** Sections 11.1, 11.2, and 11.3 and chapter 11 of PLAI. It is highly recommend to do the exercises in the book.
2. **Optional:** "What is reference counting?" by Edaqa Mortoray.
3. **Advanced:** "A Semantic Model of Reference Counting and Its Abstraction" by Paul Hudak.

Memory 2

The focus in this reading should be on the following concepts:

- What is a *root set*?
- What does it mean for a variable to *live*?
- What is the difference between *truth* and *provability* computationally?
- What do completeness and soundness mean in the context of garbage

collection?
- Why is sound, efficient garbage collection so difficult in C and C++?
- What is conservative garbage collection?

1. **Required:** Sections 11.4, 11.5, and 11.6 in chapter 11 of PLAI. It is highly recommend to do the exercises in the book.
2. **Optional:** "An Introduction to Garbage Collection, Part 2" by Richard Gillam.
3. **Advanced:** "The Garbage Collection Handbook: The Art of Automatic Memory Management" by Richard Jones, Anthony Hosking, and Eliot Moss.

Representation 1

The focus in this reading should be on the following concepts:

- What is "representation"?
- What happens when a language's features are mapped directly to equivalent host language features?
- What are the pros and cons of using existing host language features to represent your language features?
- What are two ways the book gives to represent closures?
- What are two ways the book gives to represent the environment?

1. **Required:** Chapter 12 of PLAI. It is highly recommend to do the exercises in the book.
2. **Optional:** "A lambda is not (necessarily) a closure" by Andy Wingo. This is an interesting example of a representation decision in a real

programming language. Astute observers will also notice that this had previously been assigned as an advanced reading a few weeks ago. Hopefully it's more approachable now!

Representation 2

The focus in this reading should be on the following concepts:

- What is a macro?
- What is macro expansion?
- What is the type of a macro? (Put another way, a macro is a function from what to what?)
- What is `#'` in a Racket program?
- What are guards, and why are they useful?
- Why should you not copy code?
- What is macro hygiene, and why is it good?

1. **Required:** Chapter 13 of PLAI. It is highly recommend to do the exercises in the book.
2. **Optional:** Chapter 3 of “Let Over Lambda” by Doug Hoyte.

Control Flow 1

The focus in this reading should be on the following concepts:

- What does it mean that the HTTP protocol is "stateless"?
- What are continuations?
- What is continuation-passing style?

- What is the difference between static and dynamic continuations?

1. **Required:** Chapter 14 of PLAI. It is highly recommend to do the exercises in the book.
2. **Optional:** "By example: Continuation-passing style in JavaScript" by Matthew Might.

Control Flow 2

The focus in this reading should be on the following concepts:

- What is a generator?
- What is the relationship between continuations and the program stack?
- What are tail calls?
- How do continuations relate to exceptions?
- What are cooperative and preemptive multitasking?

1. **Required:** Read chapter 14 of PLAI again. It is highly recommend to do the exercises in the book.
2. **Optional:** Reread "By example: Continuation-passing style in JavaScript" by Matthew Might.

Types 1

The focus in this reading should be on the following concepts:

- What is static type checking?
- What is the type environment?

- Does introducing types change the semantics of a language?
- What is strong normalization?
- How do desugaring/macros work with type checking?
- What does it mean for a type to be invariant across mutation?
- What is a type system?
- What does it mean for a type system to be sound?
- What are progress and preservation?

1. **Required:** Chapter 15 of PLAI from the start to 15.3. It is highly recommend to do the exercises in the book.
2. **Advanced:** "Types and Programming Languages" by Benjamin Pierce. A really excellent book, and the best introductio to this subject.

Types 2

The focus in this reading should be on the following concepts:

- What are type variables?
- What is parametric polymorphism?
- What is predicative polymorphism?
- What is impredicative polymorphism?
- What is relational parametricity?
- What is type inference?
- What is constraint generation and what is it for?
- What is unification?
- What does it mean for a program's types to be under-constrained and over-constrained?
- What are the principal types of an expression?

- What is `let`-polymorphism?

1. Chapter 15 of PLAI from the start of 15.3 to the end of 15.3.2.3 (stop at the start of 15.3.3). It is highly recommend to do the exercises in the book.

Types 3

The focus in this reading should be on the following concepts:

- What are tagged union types?
- What are untagged union types?
- What is soft typing?
- What does it mean for a type system to be nominal?
- What does it mean for a type system to be structural?
- What are intersection types?
- Why do recursive type signatures require a special constructor?
- What is subtyping?
- What is width subtyping?
- What is depth subtyping?

1. **Required:** Chapter 15 of PLAI from the start of 15.3.3 to the end of chapter 15. It is highly recommend to do the exercises in the book.

Types Review

This is a review of the last three readings. Focus on the materials outlined in those readings.

1. **Required:** Chapter 15 of PLAI. It is highly recommend to do the exercises in the book.

Quizzes ↓

Every Thursday class will begin with a quiz covering the reading assigned during the prior week. These quizzes will be quick, and should be easy for anyone keeping up with the reading. They will also include a bonus question or two based on the optional reading assigned at the end of the previous class.

Labs ↓

Date	Language	Topic
9/21 & 9/26	Racket	Expressions
9/28 & 10/3	Racket	Functions
10/5 & 10/10	Racket	Macros
10/12 & 10/17	Rust	Ownership, Lifetimes, Borrowing
10/19 & 10/24	Rust	Concurrency & Parallelism
10/31	Rust	Safety & Security

11/2 & 11/7	Java	Objects & Classes
11/9 & 11/14	Java	Object–Oriented Design
11/16 & 11/21	Prolog	Unification & Backtracking
11/28 & 11/30	Prolog	Logic & Programming [1][2]

One of the goals of this course is to combat language monoculture. If you as a programmer only use one language, you are only exposed to one way of thinking. The languages we use (whether speaking or programming) shape the ideas we consider and express. Learning more than one programming language is akin to learning more than one spoken language. In doing so, you help to broaden yourself with new perspectives and ways of thinking.

It is not expected that you become a strong programmer in each of these languages. Rather, the hope is that the exposure you receive to these languages will pay dividends in improving your understanding of and facility with programming overall. If you happen to like one or more of these languages, and want to explore them further on your own time, do.

Lab time is a time for experimentation, discovery, and question-asking (both of yourself and of me). The best way to discover the workings of a programming language is to write in it, and so I highly recommend a sense of playfulness during lab sessions. I will be available for questions, and if more than one or

two students has the same question I am happy to answer those questions for the edification of the rest of the students.

Project ↓

The course will include a four-phase project to be undertaken in teams of 8 students. Each member of a team will receive the same grade, so make sure to pick team members on whom you can rely.

In phase one, you must submit a report on an existing programming language that is not in the Top 20 of the TIOBE programming language index at the time the class starts. This means the following languages may not be used for the report:

- Java
- C
- C++
- C♯
- Python
- PHP
- JavaScript
- Visual Basic .NET
- Delphi / Object Pascal
- Perl
- Ruby
- Swift
- Assembly Language
- Go
- R
- Visual Basic
- MATLAB
- PL / SQL
- Objective-C
- Scratch

The phase 1 report is worth 100 points. If a report on any of the languages listed above is turned in, it will receive 0 points. It should detail the language's syntax, semantics, paradigm, design philosophy, and major interesting features. Each group member must contribute 2 pages to the report, meaning an 8-member group should produce a report of a minimum length of 16 pages.

In phase two, you must submit a report on an original programming language concept developed by the group. This report is a draft report, and will not be graded, but failure to submit a report in phase two will result in the group receiving no points for any submission in phase three. Like in phase one, the report should detail the language's syntax, semantics, paradigm, design philosophy, and major interesting features.

In phase three, you must submit the final draft of the report your group submitted in phase 2. This report is worth 100 points.

The final part of the project is for the group to present their finished programming language concept. This presentation is worth 100 points.

Exams ↓

The midterm exam will be administered on October 26th, during lecture. More details will be provided at a later date.

The final exam will be administered on December 5th, from 4:00pm to 5:50pm. More details will be provided at a later date.

Here is a study guide for the final exam.

Plagiarism & Cheating ↓

Plagiarism and cheating are violations of the Student Conduct Code and may be dealt with by both the instructor and the Judicial Affairs Officer. Definition and procedures for addressing cheating and plagiarism are found below.

Questions about academic dishonesty and the policy should be addressed to the Office of the Vice President, Student Services. Plagiarism is the act of presenting the ideas and writings of another as one's own. Cheating is the act of obtaining or attempting to obtain credit for academic work through the use of any dishonest, deceptive, or fraudulent means. Cheating includes but is not limited to:

1. Copying, in part or in whole, from another's test, software, or other evaluation instrument.
2. Submitting work previously graded in another course unless this has been approved by the course instructor or by departmental policy.
3. Submitting work simultaneously presented in two courses, unless this has been approved by both course instructors or by the department policies of both departments.
4. Using or consulting during an examination sources or materials not authorized by the instructor.
5. Altering or interfering with grading or grading instructions.
6. Sitting for an examination by a surrogate, or as a surrogate.
7. Any other act committed by a student in the course of his or her academic work, which defrauds or misrepresents, including aiding or abetting in any of the actions defined above.

Plagiarism is academically dishonest and makes the offending student liable to penalties up to and including expulsion. Students must make appropriate acknowledgements of the original source where material written or compiled by another is used. Procedure. Allegations of academic dishonesty may be handled directly by the instructor or may be referred by the instructor to the Judicial Affairs Officer. If handled by the instructor, the instructor has the

following responsibilities:

1. To preserve the evidence in support of the allegation;
2. To notify the student of the allegation and of the evidence on which it is based;
3. To provide the student a reasonable opportunity to challenge or rebut the allegation;
4. To notify the student of the action being taken.

The instructor may employ any of the following sanctions:

1. Verbal or written reprimand;
2. Assignment or appropriate task or examination;
3. Change of grade, including assigning a punitive grade to work involving dishonesty, or for the course, project, thesis, or any other summary evaluation of the student's academic work.

If the student does not wish to accept the sanction proposed by the instructor, the student may request and require that the allegation be referred to the Judicial Affairs Officer. In that event, the procedures specified under Executive Order 970 (Student Disciplinary Procedures of the California State University) shall be observed. The instructor shall not impose any sanction other than the sanction(s) imposed through the disciplinary procedure.

Students with Disabilities ↓

If you are in need of an accommodation for a disability in order to participate in this class, please let me know and also contact Services to Students with

Disabilities at UH-183, (909) 537-5238.

Accreditation ↓

The following are learning outcomes for the Computer Science program, accredited by ABET This course supports these outcomes.

1. An ability to apply knowledge of computing and mathematics appropriate to the discipline.
2. An ability to analyze a problem, and identify and define the computing requirements appropriate to its solution.
3. An ability to design, implement, and evaluate a computer-based system, process, component, or program to meet desired needs.
4. An ability to function effectively on teams to accomplish a common goal.
5. An understanding of professional, ethical, legal, security and social issues and responsibilities.
6. An ability to communicate effectively with a range of audiences.
7. An ability to analyze the local and global impact of computing on individuals, organizations, and society.
8. Recognition of the need to and an ability to engage in continuing professional development.
9. An ability to use current techniques, skills, and tools necessary for computing practice.
10. An ability to apply mathematical foundations, algorithmic principles, and computer science theory in the modeling and design of computer-based systems in a way that demonstrates comprehension of the tradeoffs involved in design choices.

11. An ability to apply design and development principles in the construction of software systems of varying complexity.

Student Organizations ↓

It is highly recommended that you become a member of the CSUSB Computer Science and Engineering Club. The club provides a number of support and enrichment activities, including peer tutoring and talks on current topics in computing. Joining the CSE Club is an excellent way to expand your skills and opportunities as a computer scientist or engineer.

Contact Information ↓

You may contact me via email at andrew.brinker@csusb.edu with questions and comments related to the class. Make sure to preface all email subjects with "`[CSE 320]`", to ensure that I will see and respond to your email message.