Csc344 (Programming Languages) Syllabus, Fall 2021

Instructor - Class Meetings - Office Hours - Email Processing Hours

- Instructor: Craig Graci, Computer Science & Cognitive Science
- Class meetings: Monday, Wednesday & Friday 9:10am-10:05am - **in-person**
- Office Hours: Tuesday 7:30am-9:00am & Wednesday 2:00pm-3:30pm - **zoom/meet**
- Email Processing Hours: Monday 2:00pm-3:00pm & Friday 2:00pm-3:00pm

Text

There will be no required textbook for this course.

That said, you should know that there are many great textbooks on programming languages. You might like to get acquainted with any number of them. Some are appropriate to the level of course that you are now taking. Some are not. They very greatly in terms of nature and scope. If I were featuring ML, Java, and Prolog in this course, I might require the following book:

- Webber, A. (2011). *Modern Programming languages: a practical introduction.* Franklin, Beedle & Associates.

Since I am not featuring two of those languages, I will merely recommend this book as a good read for someone who programs in a language or two but would like to learn more about programming languages (rather than merely learning more programming languages). I won't be directly referencing this book in the course, but if you should like to read about some of the programming language concepts and constructs that I will be presenting this semester, this text, or another like it, may serve you well.

Course Description

The course will be framed as one big story about programming languages, drawn from the endless possibilities for such a story. The "plot" will focus on finding answers to the question "What are the characteristics of a good programming language?". The setting will primarily be the US, England, and France over the hundred year period starting in 1950. (Not all stories are set completely in the past!) The "characters" will be selected programming languages and their most salient underlying concepts, together with the individuals who are credited with contributing to the formulation of the languages and concepts. These characters (languages and concepts and computer scientists) will range in significance, some playing lead roles, some supporting roles, and some bit parts. In turn, the more significant characters will generally be presented in terms of stories of their own. Why the story metaphor? It just seems fitting for a realm of knowledge that is so compelling and controversial as that of programming languages. Although I will be the one telling the story, you will be substantially responsible for constructing meaning from the story that is told. Moreover, in large measure, the meaning that you derive from this course will emerge from your authentic engagement in a number of programming activities and other assignments that will be incorporated into the course. The more enthusiastically you pursue these activities, the more "meaning" you are likely to derive from the course.

The story will begin within the realm of Racket, a dialect of Scheme which is a derivative of Lisp, the preeminent list processing language. In other words, the course will begin by considering an attractive manifestation of the most classic language in all of computer science, Lisp. Racket will afford an opportunity to do some engaging interactive programming right out of the gate. From there we will focus on elements of the language that will allow us to readily explore "Historical Lisp", classic list processing, recursive list processing, and higher order functions, among other significant topics.

From Lisp, the course will move to a study of Prolog. Terms will be introduced as the basis of the relational knowledge representation on which Prolog stands. The idea of crafting a knowledge base of facts and rules is presented, as is the notion of executing a program by querying the knowledge base. List processing is revisited in the context of Prolog and its head/tail notation. The flexibility of Prolog procedures is examined. A number of applications are presented, including a state space problem solver. The basics of resolution and unification are discussed.

The functional programming paradigm then becomes the focus of attention as Haskell takes center stage in the course for several weeks. Basic principles of functional programming are presented. The challenges and rewards associated with strong lexical typing are considered. The type inferencing system is studied in some detail. Higher order functions are reconsidered in the context of Haskell. Lazy evaluation is introduced. Simple applications from the realm of recreational mathematics are presented.

To round out the course, we will engage in a very brief investigation of the Rust programming languages. As an analytical adventure, we will compare and contrast features of the Rust programming language with those of the three programming languages that are more promently featured in the course, Lisp, Prolog, and Haskell.

As each preeminent experience unfolds, you will be required to engage in a reasonable amount of computer programming, and also in a number of activities that relate to programming languages and the art of computer programming. Although the course will principally feature the programming languages in Racket/Scheme/Lisp, Prolog, and Haskell, a rather long list of languages, constructs, and concepts will be incorporated into the course. Here is an approximation to the list:

> Algol – anonymous functions – assembly language – BNF – closures – compiler – context free languages – curried functions – dynamic scoping – dynamic typing – Fortran – functional programming – garbage collection – Haskell – heap – higher order functions – imperative programming – interpreter – Java – lazy evaluation – Lisp – lexical/static scoping – lexical/static typing – literate programming – logic programming – machine language – object-oriented programming – parser – parse tree – Racket – recursion – regular expressions – resolution – run time stack – scanner – Scheme – Smalltalk – token – type – unification

Some of these "ideas" will merely be mentioned in passing, since you most likely already possess considerable knowledge of them. Others will be considered at length.

Learning Outcomes

Upon successful completion of this course it is expected that you will be able to:

- Write relatively simple programs in Racket (Racket/Scheme/Lisp).
- Perform basic list processing of the sort that tends to be featured in classical Lisp programming.
- Discuss "Historical Lisp" and its contributions to programming languages.
- Compose recursive list processing functions.
- Implement higher order functions and program with higher order functions.
- Define and implement a very modest language in Racket.
- Meaningfully answer the question "Why Lisp?".
- Construct relational knowledge bases in Prolog.

- Write programs in Prolog to query and modify knowledge bases.
- Discuss the flexibility inherent in Prolog's pattern matching mechanism.
- Define recursive list processing functions in Prolog.
- Implement a state space problem solver in Prolog.
- Discuss the operation of Prolog, at a fairly high level of abstraction, in terms of resolution and unification.
- Describe the concepts of function application, currying, and partial function application.
- Represent knowledge in Haskell using lists and tuples.
- Write Haskell programs that exploit higher order functions.
- Infer types in the manner of the Haskell type system.
- Write Haskell programs that incorporate lazy evaluation.
- Deconstruct Haskell implementations of recreational mathematics problem solvers.
- Implement an interpreter for a very simple language in Haskell.
- Make appropriate use of various language description mechanisms (e.g., BNF, EBNF, context free grammars, regular expressions, and syntax diagrams).
- Define lexical typing and dynamic typing, clearly distinguish between them, and say something about their appearance in real programming languages.
- Define lexical scoping and dynamic scoping, clearly distinguish between them, and say something about their appearance in real programming languages.
- Describe essential ideas pertaining to memory allocation and deallocation, including the idea of garbage collection.
- Describe scanners and parsers, interpreters and compilers.
- Characterize programming paradigms, including: imperative programming, functional programming, logic programming, and object-oriented programming.
- Discuss various sorts of programming, including structured programming and literate programming.

Teaching Model

In order to effectively learn something, it is helpful to rely on a suitable model of learning to guide your progress. In college, your professors generally establish some constraints on your engagement with course material which will, in turn, constrain whatever model of learning you might establish for yourself. For the present course, these are the principle constraints of engagement:

1. **Classroom Presence**: Come to class prepared (1) to share material pertaining to the various learning activities in which you are expected to engage, and (2) to contribute something meaningful when asked for a contribution of some sort, perhaps an idea pertaining to language description, perhaps a fragment of code, perhaps a "transcript" of a program demo, perhaps a thought comparing/contrasting one language with another, or perhaps something of an altogether different sort.
2. **Programming Challenges**: You will be asked to engage in a number of programming activities. Some will call on you to mimic an interactive session that I will share with you, perhaps with a bit of modification. Some will call on you to simply write a collection of short programs (e.g., Lisp functions, Prolog predicates, Haskell functions), according to specification. Some will be a bit more ambitious, calling on you to build a problem solver or a modest interpreter. Sometimes there will be conceptual or theoretical elements to the programming assignments, in addition to the writing and demoing of programs.
3. **Problem Sets**: You will be asked to do a small number of problem sets which are computationally oriented but do not call on you to write or demo computer programs.

4. **Web Site**: You will be asked to build a web site, subject to a number of constraints, on which to place your work for this course. The web site will provide you with an opportunity to demonstrate that you can craft an enviable on-line portfolio of work. For programming assignments, I will look to the site in order to see that your work is in order. (That said, I will occasionally ask that you actually demo your programs for me.) With respect to the few problem sets, I will look to your Web site to see that you did the work. I will regularly be calling upon one or another of you to present work by walking us through the items that you are expected to archive on your work site.
5. **Exam 1**: There will be a 1 hour closed-book in-class exam during one of the class days within week 6 or week 7 of the semester. The exam will be formally announced one week prior to the exam date.
6. **Exam 2**: There will be a 1 hour closed-book in-class exam during one of the class days within week 12 or week 13 of the semester. The exam will be formally announced one week prior to the exam date.
7. **Final Exam**: There will be a 2 hour open-book/open-note in-class exam during the officially scheduled final exam period, which is Wednesday, December 8, from 8:00am to 10:00am.

My role as teacher will be to (1) orchestrate these learning activities, and (2) choreograph classes in a manner that integrates various aspects of these learning activities with the introduction and elaboration of material which is of central programming language concern.

Requirements

You are required to regularly attend class. You are required to participate appropriately in classroom activities, including regular discussions and occasional student presentations (generally based on selected elements from the various programming assignments, problem sets, and exams). You are required to complete all of the programming assignments. You are required to complete all of the problem sets. You are required to take the two regular season exams and the final exam.

Grading

Your grade will be determined on the basis of:

- Classroom Presense (10%)
- Web site (10%)
- Programming assignments (25%)
- Problem sets (5%)
- Exam 1 (15%)
- Exam 2 (15%)
- Final Exam (20%)

Furthermore, I will adhere to the typical process for allocating grades. Thus, with respect to overall percentages, 90 or above will map to A, 80s will map to B, 70s will map to C, 60s will map to D, and other numbers will map to E.

Important Notes

1. This is an in-person, face-to-face class. I intend to teach the course accordingly, and to adhere to the admonition of my dean:
   - CLAS Dean (August 4, 2021): Faculty should continue to plan to teach their courses as posted in the schedule. **Please don't make individual exceptions to allow students to participate remotely in face-to-face courses. We know how disruptive that is to both faculty and students.**
2. Statement precluding the student use of cell phones or laptops or other electronic communication devices in the classroom: **Students will not be permitted to use cell phones or laptops or other communication devices while class is in session.** If you should need to check your phone for extraordinary reasons, please just quietly remove yourself from the classroom and check your cell for communications in the hallway. In the case of an emergency or other unexpected exigency, tend to your emergency or unexpected exigency. Otherwise, please simply quietly return to class immediately after checking your phone.
3. In consideration of lingering COVID consequences, I will generally distribute assignments via email, and I will generally place other key documents on a course site that I will maintain.
4. In the event that the college should need to go remote, as it did starting in March, 2020, I plan to use a teaching model that I developed for use at that time and that served me and my students well for the 2+ semesters that in-person teaching was so severely limited. If the need should arise, I will send you an email with a description of the model. With luck, that unfortunate situation will not recur this semester.
5. The course web page is located at:
   `http://www.cs.oswego.edu/~blue/course_pages/2021/Fall/ProgrammingLanguages/`
6. Generally speaking, I will be processing student email this semester twice each week, on Monday from 2:00pm-3:00pm and on Friday from 2:00pm-3:00pm. Please expect my responses to any email that you should send me to be timed accordingly.
7. Generally speaking, I will holding my office hours twice each week, on Tuesday from 7:30am-9:00am and on Wednesday from 2:00pm-3:30pm. I plan to conduct my office hours solely via Zoom this semester. If you would like to meet with me during an office hour within a given week, please send me an email prior to 2pm on Monday of that week, and I will be happy to schedule a Zoom meeting with you. If you have a preferred time to meet during my office hours, I will do my best to accommodate your preference. Otherwise, I will schedule on a first come - first served basis, allocating the next office hour "slot" that is available.
8. Requests to make up exams will rarely be considered unless accompanied by a written medical excuse for your absence.
9. It is intended that you complete your work by yourself. You are, of course, welcome to ask specific technical questions of others and converse over conceptual issues, but you should be doing your own work. Compelling evidence that someone other than you contributed conspicuously to the completion of required work will result in a "maximum negative" grade for that assignment, failure in the course, or worse.
10. College Intellectual Integrity Statement: SUNY Oswego is committed to Intellectual Integrity. Any form of intellectual dishonesty is a serious concern and therefore prohibited. You can find the full policy online at `http://www.oswego.edu/integrity`.
11. College Disability Statement: "If you have a disabling condition, which may interfere with your ability to successfully complete this course, please contact the Office of Accessibility Services."
12. Clery Act/Title IX Reporting Statement: SUNY Oswego is committed to enhancing the safety and security of the campus for all its members. In support of this, faculty may be required to report their knowledge of certain crimes or harassment. Reportable incidents include harassment on the basis of sex or gender prohibited by Title IX and crimes covered by the Clery Act. For more information about Title IX protections, go to https://www.oswego.edu/title-ix/ or contact the Title IX Coordinator, 405 Culkin Hall, 315-312-5604, titleix@oswego.edu. For more information about the Clery Act and campus reporting, go to the University Police annual report: `https://www.oswego.edu/police/annual-report.`