Csc344 Course Syllabus, Spring 2023

Csc344 is the **programming languages** course at Oswego.

Instructor - Class Meetings - Office Hours - Email Processing Hours

Classes for the Spring 2023 edition of Csc344 will be offered in **face-to-face / in-person mode**, rather than any sort of remote or hybrid mode, so long as the college is not required to go remote, once again, due to the pandemic or some other unforeseen circumstance.

- Instructor: Craig Graci, Computer Science & Cognitive Science
- Class meetings: Tuesday & Thursday 11:10am-12:30pm - **face-to-face / in-person**
- Office Hours: Tuesday 8:30am-10:00am, Thursday 7:00am-8:30am - **Google Meet, by appointment only**
- 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, please be mindful of the fact 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 would 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 of the story is 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 story 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.

Embedded within the treatment of Lisp/Scheme/Racket will be a rather elaborate example of a program which amounts to the implementation of a domain specific language for the creation of educational microworlds. In a course on programming languages it is generally considered good form to consider the design and implementation of programming languages from a perspective that will afford you an opportunity to grasp the more saliet aspects of design and implementation without overwhelming you with the time consuming technical details that generally accompany the creation of real languages. The program that we will consider in considerable detail, the GSpace program, was crafted with this in mind.

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. Prolog's powerful pattern matching mechansim is considered. 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 briefly 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 examined.

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. Most significantly, we will consider Rust's ideas surrounding **ownership**, which governs memory management in the language. Unlike Lisp, Prolog, and Haskell, Rust does not incorporate a garbage collector.

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 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 – higher order functions – imperative programming – interpreter – Java – lazy evaluation – Lisp – lexical/static scoping – lexical/static typing – literate programming – logic programming – machine language – macro expansion – object-oriented programming – ownership – parser – parse tree – pattern matching – Prolog – Racket – recursion – regular expressions – resolution – run time heap – run time stack – Rust – 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 - Abstract Version

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

1. Appriciate the value of Lisp to the programming language community, write programs in Lisp that feature its most salient elements, and understand Lisp as a language oriented language, in that it serves nicely as a medium in which to craft domain specific languages.
2. Write programs in Prolog, articulate the nature and significance of logic programming, and discuss the inferencing engine which undergirds the language.
3. Write programs in Haskel, including those that make extensive use of recursion, higher order functions, and list comprehensions. Discuss the virtues of functional programming in a compelling manner, and analyze a number of programming languages with respect to their support for functional programming.
4. Demonstrate a working knowledge of some of the more salient ideas surrounding programming languages. Make good use of language design mechanisms, including BNF and phrase structure grammars. Distinguish between lexical typing and dynamic typing, and distinguish between lexical scoping and dynamic scoping. Discuss extreme models of memory management, and also the Rust model of memory management, with an eye towards fitting languages to applications.

Learning Outcomes - Detailed Version

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.
- 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.

- Write some very basic programs in Rust.
- Discuss the basics of Cargo and the infrastructural support that it provides for Rust programming.
- Describe key aspects of ownership in Rust.
- Make appropriate use of various language description mechanisms (e.g., BNF, 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 basic ideas associated with scanners and parsers, interpreters and compilers.
- Characterize programming paradigms, including: procedural programming, functional programming, logic programming, object oriented programming, and language oriented 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 Construction**: 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 admirable on-line portfolio of work. By taking seriously the task of presenting solutions your assignments in a clear manner for all to see, you are likely to improve your skills with respect to computer programming and computational problem solving to a greater degree than if just marginally make an effort to complete the assignments. I will occasionally be calling upon one or another of you to present work by walking us through some of the items that you are expected to archive on your work site.
5. **Exam 1**: There will be a 1 hour closed-book exam during one of the class periods 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 exam during one of the class periods 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 exam during the officially scheduled final exam period for this course.

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.

Suggested Process of Engagement for Students

Considering the design of this course, the following informal procedure describes how I would engage in it, were I taking the course.

For each day of the semester:

- If it is a class day, I would be certain to attend class. Furthermore, if I am aware that a particular activity will be on the class agenda for the day, I would prepare to participate appropriately in the activity.
- I would scan my to do list for assignments and/or classroom exercises that are outstanding, and I would I would work in an appropriate manner towards completing them in a timely fashion. Specifically:
  - If the due date of an assignment is fast approaching (if the assignment is due in the next day or two), I would complete it, and post my work on the assignment to my web work site in the prescribed manner.
  - If an exercise was assigned during the previous class period, I would spend some time working on the exercise, endeavoring to reach a satisfactory state of completion on the exercise prior to the next class period.

Furthermore:

- When the first exam date is announced, and study materials are presented, I would work steadily, with intensity, to prepare for the exam. And, naturally, when the day of the first exam arrives, I would do my best to arrive at the exam in a relaxed, well-prepared state.
- When the second exam date is announced, and study materials are presented, I would work steadily, with intensity, to prepare for the exam. And, naturally, when the day of the second exam arrives, I would do my best to arrive at the exam in a relaxed, well-prepared state.
- When the last week of class rolls around, and study materials are presented, I would commence serious study for the final exam. And, naturally, when the day of the final exam arrives, I would do my best to arrive at the exam in a relaxed, well-prepared state.

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 build a web work site on which to present your work on the programming assignments and problem sets.

You are required to take two 1-hour regular season exams and a 2-hour final exam. These will be in-class exams.

Grading

Your grade will be determined on the basis of your performance on the following nonexclusive activities:

1. The class attendance (15 percent)
2. The web work site (15 percent)
3. The first exam (20 percent)
4. The second exam (20 percent)
5. The final exam (30 percent)

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.

On the Programming Assignments and Program Related Assignments

The programming assignments, and the programming related assignments, afford you learning opportunities that have the potential to position you to do well on the exams. Although you do not directly received points for completing the programming assignments and the programming related assignments, you indirectly receive points by virtue of the direct relationship of assignment related knowledge to the composition of the exams. Generally speaking, if you authentically engage in doing the programming assignments, and the programming related assignments, you will obtain significant "ownership" of the knowledge related to these assignments, and your efforts will likely be rewarded by enhanced performance on the exams.

What does it mean to authentically engage in doing an assignment? To authentically engage in a programming assignment means to do your best to do it on your own, using your problem solving skills and your knowledge of the programming language used to accomplish what needs to be done. If you run into a dead-end, will want to rely on "the three Rs of learning" to find a way forward. The three Rs of learning? Reflection, resourcefulness, and resilience. With respect to resourcefulness, you might look for a clue online, either at the work posted by other students in the class, or at related programs that you find on the web. Caution: Strive to take no more than the least amount of information that you need from these resources in order to continue working on your own. The more information that you take, the more you compromise the authenticity of your engagement with the assignment! As a rule, you should be mindful of the fact that there is a huge difference with respect to your relationship to knowledge obtained (1) by authentically constructing a solution to a programming problem or a program related problem, and (2) merely reading/copying a solution. This difference with respect to your relationship to the knowledge typically means the difference between passing the exams and failing them.

Learning requires feedback. What forms of feedback should you seek with respect to the assignments in this course? With respect to the programming assignments, you should find the following three sorts of feedback to be beneficial:

1. Syntax
    - Description: The compiler for a particular programming language will check the syntax of your programs. If your program compiles, you know that you have adhered to the syntax of the language.
    - Guidance: Some compilers provide informative error messages with respect to syntax. Rust compilers do just that. Some compilers provide hardly any information with respect to syntax errors, beyond the fact that there is something wrong. Prolog is an example of this sort of compiler. When you need additional help in sorting out compiler errors, you might consult the language specification or various online sites that relate to the language.

2. Correctness

    - Description: If your program does not run according to specification on an appropriate set of test data, you know that there is something quite wrong with it. This is feedback! If it seems to work on a reasonable range of test data, you should feel moderately confident that it is reasonably correct.
    - Guidance: If your program does not run according to specification in some instance, you might want to arrange for it to produce intermediate output either by means of explicit print statements or by means of some trace facility associated with the language. Judicious use of intermediate output is the primary way that one goes about debugging the behavior of a program. The intermediate output is an important form of feedback! Generally speaking, to thoroughly understand the behavior of a program requires that you appreciate the flow of the program, either implicitly (mentally) or explicitly (computationally). If the mental effort doesn't achieve the desired understanding, move on to carefully generating explicit output in a way that makes the flow of execution abundantly clear.

3. Expression

    - Description: A program should be sound with respect to principles and practices of programming. After taking at least CS1 and CS2, you should have a reasonable idea of what it means for a program to be sound with respect to principles and practices of programming. You might like to compare it with other solutions to the problem. You can do this by sampling a relatively small number of solutions to the programming problem from the course web site. Study the better ones, with an eye towards appreciating the significant differences between your solution and the others. Ask yourself pertinent questions with respect to the quality of your program and those in the sample. What do you like more about your program than that of person P? What do you like more about person P's program than yours? What do you like more about person Q's program than person P's program? The answers to these questions is a valuable form of feedback. Moreover, by regularly reading and critiquing programs you will hone your eye for what constitutes sound programming practices and the principles on which they are grounded.
    - Guidance: After you have completed your programming assignment in the most authentic manner that you can, perform a parallel terraced scan of some number of your classmates programs, with the goal of identifying a handful of sound programs. Add your own program to the mix. Then, start reading the half dozen programs, and thinking about them, with an eye towards identifying the good, and the not so good, in each – particularly your own! In doing so, you will be generating a wealth of feedback!

With respect to the non-programming assignments, you will have to enter "mechanical mode" to check form (syntax) and function (correctness) with respect to your solutions. Beyond that, you should compare your work to that of your classmates, by browsing the course web site, in order to gain insights into what constitutes good work.

On Real Points and Fake Points

Real points are authentically earned points. Fake points are inauthentically earned points. With respect to attendance and exams, all of the points that you earn in this course will be real points. With respect to the building your web site (and fashioning the assignments that you post to your web site) you have the option of working hard to authentically earn your points, or hardly working to inauthentically earn your points. In a limited, superficial sense, both kinds of points are equal in terms of the final grade. But in a much more significant, vital sense, real points earned on building a web site that represents quality programs that you authored in an authentic fashion, tend to predict success on the exams, and hence the course. Fake points garnered on these activities tend to predict poor performance on the exams, and hence in the course.

Just as fake calories and fake fun can leave you with illusory feelings of satisfaction, fake points in the course can leave you will an illusory notion that you are on track to succeed in the course. Beware! The few additional points that you inauthentically accrue on assignments or the web building activity, above what you might earn through your own hard work, will not compensate for the learning that you miss out on by copying and pasting work that you

have not made your own. Furthermore, the false notion of understanding that you may derive from your activities will likely spell doom for you on the exams, which are what really count when it comes right down to your final grade. On the other hand, students who authentically get a substantial majority of the website building points by referencing solutions to assignments that they authentically produced tend to do pretty well on the exams, and in the course.

The assignment and the web site components of the course have been carefully designed to afford you powerful learning opportunities. It is up to you to determine whether or not you will avail yourself of these learning opportunities.

On a somewhat related note, you might enjoy taking a look at the following short text:
`https://emeryberger.medium.com/coping-with-copilot-b2b59671e516`

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 Kindles or iPads or Surfaces or other electronic communication devices while class is in session.**

   (a) 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.

   (b) If you are someone who likes to keep your class notes on one of your machines, simply take the best notes that you can by hand, on paper, and then "copy" them after class to your machine, enhancing them in whatever ways you see fit. This activity, in itself, is a great learning activity, one which affords you opportunities to clarify, expand, and reinforce your knowledge.

3. In consideration of lingering COVID consequences, I will generally distribute significant documents, including assignments, by posting them to the course web site.

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/2023/Spring/Csc344/`

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 response to any email that you may send my way to be timed according to these "student email processing" hours.

7. Generally speaking, I will be holding three office hours each week, on Tuesday from 8:30am-10:00am, and on Thursday from 7:00am to 8:30am. I plan to conduct my office hours solely via Google Meet this semester. If you would like to meet with me during an office hour, please send me an email. I will process requests for office hour appointments during my email processing hours. For the most part, I will do my best to schedule on a "first-come/first-served" basis, allocating the next office hour "slot" (15 minutes per slot) that is available. If you have a preferred time to meet during my office hours, I will do my best to accommodate your preference.

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.`