**Catalog Description**

*Environments, stores, scoping, functional and imperative languages, modules, classes, data encapsulation, types, and polymorphism. Implementation of these constructs in a definitional interpreter.*

**Instructor**

*Jim Royer*  (jsroyer at syr dot edu)

*Office: CST 4-185, x1028*

*Office hours: TBA & by appointment.*

**Teaching Assistant**

*TBA  (TBA at syr dot edu)*

*Office: TBA*

*Office hours: TBA & by appointment.*

**Course webpage**

http://www.cis.syr.edu/courses/cis352

**Texts**

* Basics of Compiler Design, by Torben Mogensen.*


**References**

Other notes will be distributed during the semester.

**Prerequisites**

In this course, we will be building on foundations established in CIS 252, CIS 275, and CIS 351. In CIS 252, you used the functional-programming language Haskell to learn concepts such as recursion, iteration, modularity, and data abstraction, as well as higher-order functions and pattern matching. In this course, we will be using the functional-programming language Haskell. We will spend the first few weeks of the semester getting familiar with Haskell. However, I am explicitly assuming that you've seen the major ideas of functional programming before: if you do not understand the concepts as I review them, it is your responsibility to ask for help and to get up to speed.

In CIS 275, you covered basic set theory, functions, predicate logic, and induction. In this course, we will make use of these topics in multiple ways. An important feature of programs written in functional languages is that they are typically much easier to reason about and prove correct than those written in imperative languages. We will make significant use of inductively defined data types and data structures: understanding induction is essential for understanding these beasts and for correctly implementing algorithms that operate on them.

In CIS 351, you saw many data structures—including stacks, queues, lists, and trees—as well as algorithms that operate on them. In this course, we will frequently use trees as internal representations of program code (these representations are known as “abstract syntax”) that we
then manipulate in various ways. We will make use of stacks (or related structures) to implement concepts such as program environments or stack frames.

Objectives

Whenever you learn a new programming language, you're faced not only with learning a new syntax but also with determining what features the language provides, how those features interact with one another, and how to use those features effectively. Furthermore, many programming applications ultimately require you to develop and implement your own small (or not so small) language by which the end user interacts with the application.

This course explores concepts underlying the definition, implementation, and use of programming languages. The goal is to provide you with an understanding of (and a vocabulary for) common language features, including how they are implemented, how other language-design choices affect them, and how they can be used effectively in program development.

This year's edition of the course continues an experiment in incorporating elements of computational semantics of natural languages (e.g., English) into our work on programming languages. The studies of natural and programming languages have many elements in common. We (students and teacher) will discover how well these mesh as the semester progresses.

We will be using the programming language Haskell.

Outcomes (Provisional and likely to be revised)

After completion of the course, you should be able to:

- When given an informal but fairly precise English description, write a Haskell program that accurately captures the desired behavior.
- Write data-directed Haskell programs over lists, trees, and other inductively defined data structures.
- When given a moderate-sized Haskell program and relevant input, calculate the result of that program.
- When given a BNF description of concrete syntax and a specific piece of concrete syntax, draw the corresponding abstract-syntax tree.
- When given a Haskell or $\lambda$-calculus expression, identify the free variables, identify the bound variables, and calculate the lexical addresses of the bound variables.
- When given a $\lambda$-calculus expression, identify all $\beta$-redexes and $\beta$-reduce the expression.
- When given a small piece of code, calculate its value under a variety of execution scenarios (dynamic or lexical scope, different parameter-passing mechanisms, etc).
- Implement an interpreter for a simple language incorporating many of the constructs listed in the “course topics” section below.

Measurement

Your final grade will be based on a variety of activities:

Homework assignments (45% of final grade)

There will be a homework assignment approximately every week: I will drop the lowest homework grade at the end of the semester. Occasionally, students may be asked to explain their homework to me or to the TA: in such cases, the homework grade will be based on the results of this explanation.
Exams (55% of final grade)
There will be five or six in-class quizzes during the semester. The lowest quiz score will be dropped at the end of the semester. There will also be a two-hour optional final exam: the exam portion of your final grade will be the greater of (1) your cumulative average of in-class exams, and (2) your score on the final exam.

You should also be aware of the following submission practice and policy:

On some homework and lab assignments, you may work singly or in pairs; on other assignments, you must work alone. If you work with someone else, you should submit a single solution: each persons name must appear clearly on the first page. All members of a group will receive the same grade (even if only one student is asked to explain a homework); groups of more than two will receive no credit.

Programming assignments should be submitted via Blackboard. Written homework assignments should be placed in the marked bin near CST 4-226. Assignments are due by the date and time specified on them: No late assignments will be accepted.

Topics

Other Information

Academic Integrity
All members of the Syracuse University community—faculty, staff, and students—are expected to maintain academic integrity in all situations. As a member of this community, you should also be familiar with the Universitys academic-integrity policy, which is available at:

http://academicintegrity.syr.edu

I expect all students to behave with academic integrity: do not cheat, plagiarize, or commit fraud. Fraud includes altering previously graded work; plagiarism includes using someone elses work without proper credit. If I discover any violations of academic integrity, I will give the guilty parties failing grades (XF) for the course and report the culprits to the Office of Academic Integrity. If you are unsure whether a certain action is an academic-integrity violation, assume that it is: you may ask us for clarification at any time.

Every student must read and sign a copy of the course Honor Policy, which details your obligations to behave with academic integrity. Students will receive zeroes on all coursework until this sheet is turned in.

Accommodations
Our community values diversity and seeks to promote meaningful access to educational opportunities for all students. Syracuse University and I are committed to your success and to supporting Section
504 of the Rehabilitation Act of 1973 as amended and the Americans with Disabilities Act (1990). This means that in general no individual who is otherwise qualified shall be excluded from participation in, be denied benefits of, or be subjected to discrimination under any program or activity, solely by reason of having a disability.

If you believe that you need accommodations for a disability, please contact the Office of Disability Services (ODS),

http://disabilityservices.syr.edu,

located at 804 University Avenue, Room 309, or call 315-443-4498 for an appointment to discuss your needs and the process for requesting accommodations. ODS is responsible for coordinating disability-related accommodations and will issue students with documented disabilities “Accommodation Authorization Letters,” as appropriate. Because accommodations may require early planning and generally are not provided retroactively, please contact ODS as soon as possible.

**Religious Observances**

SUs religious observance policy, found at

http://supolicies.syr.edu/empben/religiousobservance.htm,

recognizes the diversity of faiths represented among the campus community and protects the rights of students, faculty, and staff to observe religious holy days according to their tradition. Under the policy, students are provided an opportunity to make up any examination, study, or work requirements that may be missed due to a religious observance provided they notify their instructors before the end of the second week of classes. For fall and spring semesters, an online notification process is available through MySlice/Student Services/Enrollment/My Religious Observances from the first day of class until the end of the second week of class.