CSE 413: Programming Languages and their Implementation

Hal Perkins
Autumn 2014
Today’s Outline

• Administrative info
• Overview of the course
• Introduction to Scheme Racket
Registration

• Please sign up on info sheet at end of class you’re still trying to get in

• We’ll see what we can do, but no promises (depends on how many requests there are, resources available, etc.)
Who, Where & When

• Instructor
  » Hal Perkins (perkins@cs.washington.edu)

• Teaching Assistants
  » James Barnes, Gurwinder Gulati, Gene Kim, Soumya Vasisht
  » Office hours & locations tba, etc.
    • Fill out the doodle on the course web

• Lectures
  » MWF 11:30-12:20, MOR 220
All info is on the CSE 413 web:

http://www.cs.washington.edu/413

Look there for schedules, contact information, assignments, links to discussion boards and mailing lists, etc.
CSE 413 E-mail List

• If you are registered for the course you are automatically added.

• E-mail list is used for posting important announcements by instructor and TAs

• You are responsible for anything sent here
  » Mail to this list is sent to your UW email address
CSE 413 Discussion Board

• Use the Catalyst GoPost message board to stay in touch outside of class
  » Staff will watch and contribute too
  » General discussion of class contents
  » Hints and ideas about assignments (but not detailed code or solutions)
  » Other topics related to the course
• To-Do: post or reply to the intro message and it will track unread postings for you! (Do it!!)
Course Computing

• All software is freely available and can be installed anywhere you want
  » Links on the course web

• Also should be installed in the College of Arts & Sciences Instructional Computing Lab
Grading: Estimated Breakdown

• Approximate Grading:
  » Homework + Projects: 55%
  » Midterm: 15% (in class, tent. 11/02)
  » Final: 25% (Wed. Dec 10, 2:30 pm)
  » Other 5%

• Assignments:
  » Weights will differ depending on difficulty
  » Assignments will be a mix of shorter written exercises and shorter/longer programming projects
Deadlines & Late Policy

• Assignments generally due Thursday evenings via the web
  » Exact times and dates given for each assignment

• Late policy: 4 late days per person
  » At most 2 on any single assignment
  » Used only in integer units
  » If we have any group work, both students must have late days to use and both are charged if used
  » Don’t burn them up early!!
Academic (Mis-)Conduct

• You are expected to do your own work
  » Exceptions (group work), if any, will be clearly announced
• Things that are academic misconduct:
  » Sharing solutions, doing work for others, or accepting work from others
  » Copying solutions on the web
  » Consulting solutions from previous offerings of this course
  » etc. Will not attempt to provide exact legislation and invite attempts to weasel around the rules
• Integrity is a fundamental principle in the academic world (and elsewhere) – we and your classmates trust you; don’t abuse that trust
Reading

- No required $$$ textbook
- Good resources on the web
- Follow “Functional Programming/Racket” link:
  - Racket documentation (*Guide* has language details)
  - *How to Design Programs* Intro textbook using Scheme
  - *Structure and Interpretation of Computer Programs*
    - Fantastic, classic intro CS book from MIT. Some good examples here that are directly useful
Tentative Course Schedule

- Week 1: Functional Programming/Racket
- Week 2: Functional Programming/Racket
- Week 3: Functional Programming/Racket
- Week 4: FP wrapup/intro to Ruby
- Weeks 5-6: Object-oriented programming and Ruby; scripting languages
- Weeks 7-9: Language implementation, compilers and interpreters
- Week 10: garbage collection; special topics
Work to do!

• Download Racket and install

• Run DrRacket and verify facts like $1+1=2$

• Post or reply on discussion board so it will track unread articles for you

• Fill out office hour doodle
Now where were we?

• Programming Languages

• Language Implementation
Why Functional Programming?

• Focus on “functional programming” because of simplicity, power, elegance

• Stretch our brains – different ways of thinking about programming and computation
  » Often a good way to think even if stuck with C/Fortran/…

• Now mainstream – lambdas/closures in Javascript, C#, Java 8; f.p. idioms in C++11; F# standard in Visual Studio, functional programming is the “secret sauce” in Google’s infrastructure; etc., etc.

• Let go of Java/C/… for now
  » Easier to approach functional prog. on its own terms
  » We’ll make connections to other languages as we go
Scheme / Racket

• Scheme: *The classic functional language*
  » Enormously influential in education, research

• Racket
  » Modern Scheme dialect with some changes/extras
  » DrRacket programming environment (was DrScheme for many years)

• Expect your instructor to say “Scheme” a bunch
Functional Programming

• Programming consists of defining and evaluating functions

• No side effects (assignment)
  » An expression will always yield the same value when evaluated (referential transparency)

• No loops (use recursion instead)

• Racket/Scheme/Lisp include assignment and loops but they are not needed and we won’t use
  » i.e., you will “lose points”
Primitive Expressions

• constants
  » Integer
  » rational
  » real
  » boolean

• variable names (symbols)
  » Names can contain almost any character except white space and parentheses
  » Stick with simple names like value, x, iter, ...
Compound Expressions

• Either a combination or a special form
• 1. Combination: (operator operand operand …)
  » there are quite a few pre-defined operators
  » We can define our own operators

• 2. Special form
  » keywords in the language
  » eg, define, if, cond
Combinations

(operator operand operand …)

• this is prefix notation, the operator comes first
• a combination always denotes a procedure application
• the operator is a symbol or an expression, the applied procedure is the associated value
  » +, -, abs, my-function
  » characters like * and + are not special; if they do not stand alone then they are part of some name
Evaluating Combinations

• To evaluate a combination
  » Evaluate the subexpressions of the combination
  » Apply the procedure that is the value of the leftmost subexpression (the operator) to the arguments that are the values of the other subexpressions (the operands)

• Examples (demo)
Evaluating Special Forms

• Special forms have unique evaluation rules
• `(define x 3)` is an example of a special form; it is not a combination
  » the evaluation rule for a simple define is "associate the given name with the given value"
  » All special forms do something different from simple evaluation of a value from (evaluated) operands
• There are a few more special forms, but there are surprisingly few compared to other languages
Procedures
Recall the define special form

• Special forms have unique evaluation rules
• (define x 3) is an example of a special form; it is not a combination
  » the evaluation rule for a simple define is "associate the given name with the given value"
Define and name a variable

\[(\text{define } \langle \text{name} \rangle \langle \text{expr} \rangle)\]

» define - special form

» name - name that the value of expr is bound to

» expr - expression that is evaluated to give the value for name

• define is valid only at the top level of a \text{<program>} and at the beginning of a \text{<body>
Define and name a procedure

\[
\text{(define (\langle \text{name} \rangle \langle \text{formal params} \rangle) \langle \text{body} \rangle)}
\]

- define - special form
- name - the name that the procedure is bound to
- formal parameters - names used within the body of procedure, bound when procedure is called
- body - expression (or sequence of expressions) that will be evaluated when the procedure is called.
- The result of the last expression in the body will be returned as the result of the procedure call
Example definitions

(define pi 3.1415926535)

(define (area-of-disk r)
  (* pi (* r r)))

(define (area-of-ring outer inner)
  (- (area-of-disk outer)
     (area-of-disk inner)))
Defined procedures are "first class"

• Compound procedures that we define are used exactly the same way the primitive procedures provided in Scheme are used
  » names of built-in procedures are not special; they are simply names that have been pre-defined
  » you can't tell whether a name stands for a primitive (built-in) procedure or a compound (defined) procedure by looking at the name or how it is used
  » [Disclaimer: This is not always strictly true in Racket.]
Booleans

• Recall that one type of data object is boolean
  » #t (true) or #f (false)
• We can use these explicitly or by calculating them in expressions that yield boolean values
• An expression that yields a true or false value is called a predicate
  » #t =>
  » (< 5 5) =>
  » (> pi 0) =>
Conditional expressions

• As in all languages, we need to be able to make decisions based on values and do something depending on the result
Special form: cond

(cond ⟨clause1⟩ ⟨clause2⟩ ... ⟨clausen⟩)

• each clause is of the form
  » (⟨predicate⟩ ⟨expression⟩)

• the last clause can be of the form
  » (else ⟨expression⟩)
Example: sign.scm

; return the sign of x as -1, 0, or 1

(define (sign x)
  (cond
   ((< x 0) -1)
   ((= x 0) 0)
   ((> x 0) +1))))
Special form: if

(if ⟨predicate⟩ ⟨consequent⟩ ⟨alternate⟩)

(if ⟨predicate⟩ ⟨consequent⟩ )
Logical composition

(and \langle e_1 \rangle \langle e_2 \rangle \ldots \langle e_n \rangle)
(or \langle e_1 \rangle \langle e_2 \rangle \ldots \langle e_n \rangle)
(not \langle e \rangle)

• Scheme interprets the expressions ei one at a time in left-to-right order until it determines the correct value
in-range.scm

; true if val is lo <= val <= hi

(define (in-range lo val hi)
  (and (<= lo val)
       (<= val hi)))