Today’s topics
• Who should take CS17
• CS17-related activities/workload
• The design of CS17/18
• The place of CS
• A first encounter with a computer programming language
CSCI0170 = “CS17”
Prof. John Hughes = “Spike”
jfh@cs.brown.edu
Who should take CS17?
Who should take CS17? (2)

• What if I can’t program?
  • Then you should take CS17: it's an introduction to CS
  • About half of the TAs had no prior programming experience before CS17

• What if I *can* program?
  • Then you should take CS17 (probably)
  • About half the TAs had some or lots of experience, and still found it valuable.

• What if I don’t like to work?
  • *...then* maybe CS17 is the wrong course for you
CS017 activities

• Class, 3 hours per week, required; in class activities, occasional quizzes
• 2-hour lab, once per week
• Homework every week, 1-8 hours
• Three larger projects, spaced throughout semester
• Take-home final due near end of reading period
• Optional workshops to reinforce ideas from class

• Details: see syllabus on website (Google: CS17)
What makes CS17 function well?

• 4 head teaching assistants (HTAs)
  • Amelia O’Halloran
  • Rahul Mani
  • Giselle Garcia
  • Maddy Adams

• 24 more teaching assistants (UTAs)
  • Hold “TA hours”
  • Run labs
  • Take class notes
Respect/consideration

• Treat others in the class with respect
• Treat staff with respect
• Have *self-respect*: get things done when they are due.
  • HW takes about one long night’s work per week; you have 7 days to do it. Plan ahead, and don’t tell me that your hockey game or any other one-or-two-day obligation makes it impossible for you to do it on time.
  • If you’re sick for 5 days, let me know (*not* the TAs)
• No phones/computers during class unless directed
Course features

• Multiple programming languages (Racket, ReasonML, then Java, Scala in CS18)
• “Analysis” of programs early
• “No magic” (or at least minimal magic)
  • You’ll be able to perfectly predict what any program does, step by step
• CS17/18 sequence integrates big ideas from computer science with learning to program
How’s it different from CS15?

• Integrated approach
• No magic
• No skits
• Fewer hours per week (some years)
• More working together
• Equally good preparation for subsequent courses in CS
Course Goals

• Learn strategies for finding efficient and elegant solutions to computational problems
• Learn multiple languages, so that you see what’s language-independent and what’s a bigger-picture idea
• Learn about other areas of CS through project experience:
  • Bootstrapping ("Bignum")
  • Program interpretation ("Rackette")
  • Artificial intelligence ("Game")
Power, Simplicity, Elegance

• By the end of CS15, you’ll write 2000-line programs
  • For some of us, that’s a failure rather than an achievement
• We aim for simple (hence completely understandable), powerful tools that let us write elegant programs that are provably correct
Pair programming

• On many assignments, you’ll work in pairs, in a very structured way.
• Leads to
  • faster success on tasks
  • more exchanges of skills
  • better articulation of thoughts
  • better preparation for the workplace
Grading

• Absolute scale: your grade is *independent of other students’ grades*
• Learn the material well, you’ll get an A
• Details on course syllabus
How to get a B in CS17

• Get a solid A for the first 9 weeks, then slack off.

• Getting a B for the first 5 weeks leaves you unprepared for the next 7.
Course Mechanics

• Banner registration is *essential*
• First lab is *this weekend*
• “cs accounts” set up from Banner registration list
  • No registration, no account, no lab, ...
How important is computer science?

- It makes your phone work
- It’ll help you get a job
- ...

Activity

• Introduce yourself to both your neighbors
• Tell them something (interesting) about yourself that you expect makes you different from them
• 30 seconds!
To speak a second language is to have a second soul.
--Charlemagne
Racket

• Our first programming language
• You’ll only learn/use a small part of Racket
  • We’ll call it “CS17 Racket”
  • It’s enough to write essentially any program in the world (take CS1010 to see why!)
• Programs consist of a sequence of zero or more definitions, followed by a zero or one expressions
• *Colored italic text* indicates new terms that have special meanings different from the ones you’re used to.
  • Think of them as words you don’t know yet, in a new foreign language
Racket Demo
Post-demo cooldown

• “Programs consist of a sequence of zero or more definitions, followed by a zero or one expressions”
• That was a no-definitions, one-expression program
• We’ll be writing somewhat more interesting programs soon
• You’ll be using a DrRacket to write and run programs. You could download it right now.
• It allows you to use several versions of the language. We’ll tell you which one to use as we go.
Racket operation

• Racket “runs” a program by reading a piece at a time, processing it, often through a process called evaluation and sometimes printing something, and then moving on to the next program piece.
  • NB: There might be errors, in which case everything stops.

• This is called a “read-eval-print loop”, or REPL.
  • Sometimes the “eval” step isn’t actually evaluation
  • Sometimes there’s no printing
  • Still, it’s called a REPL
Read-eval-print

• In our little program,
  • Racket “read” the text 17, which it recognized as a number expression
  • Racket “evaluated” that to produce a number value
  • Racket printed out a printed representation of that number value, namely: 17
Undefined so far

- Definition
- Expression
  - Number expression
- Reading
- Processing
- Evaluation
- Printing
  - Printed representation
- Value
  - Number value

Soon the number of undefined words/terms will begin to decrease!
Definitions

• Names *simplify* things
  • “Spike” rather than “that person standing up on the stage”

• Names *preserve* things
  • Go north two miles, past the old dairy farm; take a right and drive until you pass a grove of oak trees, and a quarter mile later, there’s a dirt road on the left. You go down there and pass by four white houses and a grey one with a bright red door, and then, set way back from the road, there’s a house painted a kind of faded yellow. That’s the pharmacy.
  • Now you can just say “the pharmacy” and not have to carry out all those instructions again.
Names in Racket

- Names, in Racket, serve to name things called values.
- There are five kinds of values in the CS17 version of Racket.
  - One kind of value is “number”
  - We’ll get to the others later
- To associate a name to a value, we type something like
  \texttt{(define height 37)}
- Intuition: the name “height” now refers to the number 37
- Details as we go along
- \texttt{define} is a keyword
  - special sequence of letters that we cannot use for anything else
- Text in brown monospaced font is part of a computer program.
(define height 37)

**define** is a *keyword*
- A special sequence of letters that we cannot use for anything else
- There are about twelve of these we’ll use, and a few others beyond those

**height** is a *name*
- Racket rules:
  - “a sequence of non-whitespace, non-special characters that is not a keyword, and cannot be interpreted as a number”
- CS17 Racket rules (the ones that matter to you!):
  - “a sequence of letters and digits and hyphens, starting with a letter”
Activity!

• Work with your neighbor
• Take 60 seconds
• Racket name: “any sequence of non-whitespace, non-special non-special characters that is not a keyword, and cannot be interpreted as a number”
• CS17 Racket name: a Racket name that’s “a sequence of letters and digits and hyphens, starting with a letter”
• Which of these are names in CS17 Racket? Why or why not?
  (a) Idt (b) syn-text-30 (c) 30-rock (d) my_name (e) f30-10

• Go!
Demo (of definitions)

- (define height 37)
- Nothing much seems to happen
- I promise that something *did* happen
  - The name “height” got associated to the value “37”
- What about REPL?
  - Read happened
  - A different kind of processing (*not* evaluation) happened [because of the keyword `define`]
  - Nothing got printed
Structure of a definition (for now)

• Example: (define height 37)

• structure: (define <name> <num>)
• You don’t yet know the rules for “num”s, but any list of digits works.
• The green things in pointy brackets are descriptions of what has to
go in those places, not what you actually type.
• The parentheses and the keyword define are “literals” – things you
must type (almost) exactly as shown
  • You can add blanks before/after parens...but please don’t
  • (Approximate) You must have a blank (or other “whitespace”) after define
Activity

• Working from the example
  \texttt{(define height 37)}
  try to generalize and write a definition that associates the name \texttt{width} with the value \texttt{11}

• Go!
Using definitions

• We said that \textit{(define width 11)} “associates the name \texttt{width} with the value \texttt{11}”

• We now extend our notion of “evaluation” (which we’re defining, bit by bit) to say
  • When we evaluate a name, the result is the associated value (if any)
  • If there’s no associated value, it’s an error
Demo!
Where we are

• You know how to write definitions
• You know at least one way to *use* “defined” things
• You know one kind of “expression”, namely, “things that look like numbers”
Let’s do something more

• Arithmetic in Racket
• Demo!
How did I know I could write those things?

• What are the rules of “sentence construction” in this new language?
• How do we write down such rules?
• How do we know what the “sentences” mean when we write them?
• What do “errors” mean/do?

• ...next class.
(let
  ([alon1 (list 1 2)]
   [alon2 (map (lambda (x) (/ x 4.0)) (list 2 14))]
   (map + alon1 alon2))

  • Assumption: we can break this into “tokens”: little pieces that constitute the “words” of our language
    • (}
    • let
    • [}
    • alon1
    • /
    • +
    • 4.0
    • 14
    • )
    • ]
From tokens to language

• To “define” a programming language, specify which token-sequences are allowed
• The rules of what’s allowed are called “syntax”
  • I’ll gradually disclose these over the next few lectures
• We’ll also assign meaning to each allowable token-sequence; that’s called “semantics”
• For now, the allowable token-sequences (“programs”) are pretty limited: individual numbers, things like (+ 3 5)