Numbers, Data, Functions

CS 151: Introduction to Computer Science I

What is computer science?

- "Computer science is not really about computers – and it's not about computers in the same sense that physics is not really about particle accelerators, and biology is not about microscopes and Petri dishes...and geometry isn't really about using surveying instruments. ... it's very easy to confuse the essence of what you're doing with the tools that you use." - Han Abelson
- Main subject of our course: designing and writing computer programs
- Computer programs are a way to interact with data
- Java, C, Python, Ruby, Scala, **Racket**, ...
- Excel, Powerpoint (transitions), smartphone (alarm, calendar), email (autoreply, mass email templating), Google search tricks, ...

Typed Racket

- Typed Racket is variant of Racket which checks for type errors before the program is executed
- Racket is derived from Scheme which is a functional programming language
- Racket itself is a multi-paradigm programming language but we will focus on functional paradigm
- Pure functions, no side effects, iteration through recursion, referential transparency, high-order functions

Expressions

5 + 6, (-1)*9, 182/ (9 + 4)
Racket: prefix notation

```
(+ 5 6)
```

```
(* -1 9)
```

```
(/ 182 (+ 9 4))
```

Won't work:

```
+ 5 6
```

```
(5 + 6)
```

Expressions

9 * (10 - 7) + 15.5

```
(+ (* 9 (- 10 7)) 15.5)
```

Arbitrary arity: $1 + 2 + 3 + 4 + 5$

```
(+ 1 2 3 4 5) ⟹ 15
```

Expression trees capture the structure of expressions

Types: Integers

- Every piece of data in Racket is associated with a **type**
- `Integer` includes . . . , -3, -2, -1, 0, 1, 2, 3, . . .
- Some functions: `max`, `min`

```
(max 99 100) ⟹ 100
```

```
(max 5 50 15 22) ⟹ 50
```

Some other functions: `quotient`, `remainder`

```
(remainder 17 3) ⟹ 2
```

```
(quotient 17 3) ⟹ 5
```

Dividing two `Integer`s may not be an `Integer`

`(/ 5 4)` $\Longrightarrow$ $1\frac{1}{4}$

Types: Exact-Rationals

`Exact-Rational` includes numbers that can be written as fractions

$$\texttt{(+ (/ 8 3) (/ 5 2))} \Longrightarrow \texttt{(+ } 2\tfrac{2}{3}\ 2\tfrac{1}{2}\texttt{)} \Longrightarrow 5\tfrac{1}{6}$$

- `ceiling`, `floor`, `round`

$$\texttt{(ceiling } 6\tfrac{3}{5}\texttt{)} \Longrightarrow 7$$

$$\texttt{(floor } 6\tfrac{3}{5}\texttt{)} \Longrightarrow 6$$

$$\texttt{(round } 6\tfrac{3}{5}\texttt{)} \Longrightarrow 7$$

Types: Reals

- What about $\sqrt{3}$?
- `Real` includes all real numbers
- New functions: `sqrt`, `sin`, `cos`, `exp`, `log`, `expt`
- Some built-in real numbers: π, e, ϕ

```
(sqrt 3) ⟹ 1.7320508075688772
```

Note: computer stores only an **inexact approximation** because memory is finite

All together

The radius of the Earth is about 6400 km. What is its volume?

$$\frac{4\pi}{3}6400^3$$

```
(* (/ 4 3) pi 6400 6400 6400)
```

```
(* (/ 4 3) pi (expt 6400 3))
```

All of these are of type `Real` and also of type `Number`

Types: Booleans

- What about $3 < 5$?
- `Boolean` includes true and false
- Captures Yes/No, True/False, Up/Down, A/B, ...
- Result of comparisons on `Numbers`: `<`, `>`, `<=`, `>=`, `=`

```
(< 3 5) ⟹ #t
```

```
(= (/ 75 5) 15) ⟹ #t
```

Higher arity: "$3 < 4$ and $4 < 5$" equivalent to

```
(< 3 4 5)
```

Types: Booleans

Functions on booleans: `and`, `or`, `not`
"$3 < 5$ and also $3 > 0$"

```
(and (< 3 5) (> 3 0))
```

```
(and (not #t) #f)
```

```
(and (or #f #t) (and #t  #t))
```

```
(not (not (not #t)))
```

If expressions

What about “if 81 $>$ 75, then 8 otherwise 5"?

```
(if (> 81 75) 8 5) ⟹ (if #t  8 5) ⟹ 8
```

```
(if (> 71 75) 8 5) ⟹ (if #f  8 5) ⟹ 5
```

"if 22 $\geq 7\pi$, then 100 else 200"

```
(if (>= 22 (* 7 pi)) 100 200)
```

If expressions

The price of a ticket to "Hamilton" is $50 if the day of the month ends in a 0 (otherwise it's $100)
What's the price today?

```
(if (= (remainder 18 10) 0)
    50
    100)
```

```
(- 100 (if (not (= (remainder 18 10) 0) 50 0)))
```

If expressions

At a Cubs baseball game, the stadium makes $20 off of every ticket sale. They also pay $100,000 to the staff during the game. However, if more than 30,000 people come, they will also buy everyone a hotdog, which costs $2 per person.

The number of people who went to the game was 42,991. How much money did the stadium make?

```
(- (* 20 42991)
    (if (> 42991 30000) (* 2 42991) 0)
    100000)
```

Variables

- Give names to data that has meaning or is reused
- **define** binds an *identifier* with an expression, function or values..

```
(:  attendees :  Integer)
(define attendees 42991)
(- (* 20 attendees)
    (if (> attendees 30000) (* 2 attendees) 0)
    100000)
```

```
(:  day-of-month :  Integer)
(define day-of-month 18)
(if (= (remainder day-of-month 10) 0)
    50
    100)
```

Variables: Counting Sheep

```
(: number-of-sheep : Integer)
(define number-of-sheep 2)
(: number-of-humans : Integer)
(define number-of-humans 1)
```

Lots of new things we can compute:

```
(: number-of-legs : Integer)
(define number-of-legs
  (+
   (* 2 number-of-humans)
   (* 4 number-of-sheep)))
```

Variables: Counting Sheep

```
(: average-legs : Exact-Rational)
(define average-legs
  (/
   number-of-legs
   (+ number-of-sheep number-of-humans)))
```

```
(: sheep-dominance? : Boolean)
(define sheep-dominance?
  (> number-of-sheep number-of-humans))
```

```
(: scared? : Boolean)
(define scared?
  (and
   sheep-dominance?
   (> number-of-sheep 50)))
```

Functions

Recall our expression for the volume of the Earth:

```
(* (/ 4 3) pi 6400 6400 6400)
```

Function version:

```
(:  planet-volume :  Real -> Real)
(define (planet-volume radius)
  (* (/ 4 3) pi radius radius radius))
```

Another version

```
(:  planet-volume :  Real -> Real)
(define planet-volume   (lambda (radius)   (* (/ 4
3) pi radius radius radius)))
```

Functions

$$f(x) = x^2 + 1$$

```
(:  f :  Real -> Real)
(define (f x)
  (+ (* x x) 1))
```

$$g(x, y, z) = xyz$$

```
(:  g :  Real Real Real -> Real )
(define (g x y z)
  (* x y z))
```

Evaluate:

(f -3) $\Longrightarrow$ 10

(g 1 2 3) $\Longrightarrow$ 6

Functions: Counting Sheep

```
(:  num-legs :  Integer Integer -> Integer)
(define (num-legs num-humans num-sheep)
  (+
   (* 2 num-humans)
   (* 4 num-sheep)))
```

```
(num-legs 35 0) ⟹ 70
(num-legs 5 6) ⟹ 34
(num-legs 1 -1) ⟹ -2
```

```
(:  sheep-dom?  :  Integer Integer -> Boolean)
(define (sheep-dom?  num-humans num-sheep)
  (> num-sheep num-humans))
```

Function Contracts

Every function you write in this course will have four parts:

1. The type annotation (also called the “contract”)
2. A purpose (also include any assumptions about the input not captured by the type)
3. The definition
4. Tests

Note: you get to choose how to name the functions. Choose something meaningful

Function Contracts

```
(:  planet-volume :  Real -> Real)
;; compute the volume of a planet
;; with the given radius
;; radius should be positive
(define (planet-volume radius)
  (* (/ 4 3) pi radius radius radius))
```

```
(:  num-legs :  Integer Integer -> Integer)
;; output the total number of legs
;; num-humans and num-sheep should be positive
(define (num-legs num-humans num-sheep)
  (+
   (* 2 num-humans)
   (* 4 num-sheep)))
```

Tests

```
(check-expect (num-legs 1 1) 6)
(check-expect (num-legs 0 0) 0)

(check-within (sqrt 2) 1.414 0.001)

;; test the volume of Earth
(check-within (planet-volume 6400)
  1098066219443.5236 0.01)
;; test the volume of Mercury
(check-within (planet-volume 2440)
  60840000000 10000000)

(test)
```

Tests

Writing good tests is an important part of being a successful programmer

- Do you cover **all** normal use cases of the function?
- Did you catch any edge cases?
- Make each test look for exactly one bug

Some Broken Code...

```
(:  positive?  :  Real -> Boolean)
;; return true if x is positive and false if
negative
(define (positive?  x)
  (if (> x 0) #t #f))
```

```
(:  f :  Real -> Real)
;; compute f(x) = x^2 + 1
(define (f x)
  (* (+ x x) 1)
```

```
(:  abs :  Integer -> Integer)
;; compute absolute value of x
(define (abs x)
  (sqrt (* x x)))
```

Some Broken Code...

```
(:  avg-legs :  Integer Integer -> Exact-Rational)
(define (avg-legs num-humans num-sheep)
  (/
   (num-legs num-humans num-sheep)
   (+ num-sheep num-humans)))
```

Types: Strings

`String` includes any phrase of keyboard characters:
“Bob”, “what do you want for dinner”, “b398af3”

```
"Hello world!"
```

```
(:  is-it-5 :  Integer -> String)
;; tells you if the input is 5
(define (is-it-5 n)
  (if (= n 5)
      "It’s 5!!"
      "Not 5..."))
```

Types: Strings

Compare `Strings` with `string=?`

```
(:  whats-for-breakfast :  String -> String)
(define (whats-for-breakfast desire)
;; respond to a breakfast request
  (if (string=?  desire "yogurt")
      "Coming right up!"
      "We don't have that..."))
```

Stick together `Strings` with `string-append`

```
(string-append "John&" "Paul&" "Ringo&" "George.")
⟹ "John&Paul&Ringo&George."
```

Types: Symbols

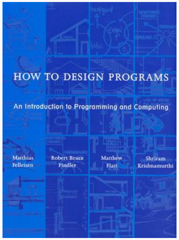
Symbol includes one-word phrases of keyboard characters:
'red, 'Chicago, 'computer-science, 'asleep

```
(:  hunger :  Real -> Symbol)
(define (hunger hours-since-eating)
  (if (> hours-since-eating 4.5)
      'very-hungry
      'kinda-hungry))
```

```
(:  num-to-color :  Integer -> Symbol)
(define (num-to-color n)
;; color the integers R, G, B, R, G, B, ...
  (if (= (remainder n 3) 0)
      'red
      (if (= (remainder n 3) 1)
          'green
          'blue)))
```

Book

- There is a book. It's available here: `www.htdp.org` You're not required to buy it.
- You will need DrRacket: `racket-lang.org`
- Questions on Racket syntax: `people.cs.uchicago.edu/~adamshaw/cmsc15100-2017/typed-racket-guide/index.html`
- ...or `docs.racket-lang.org`
- ...or ask on Piazza



Canvas+Piazza

Canvas:

- Lecture slides, homework, syllabus
- Information regarding exams etc
- Submit homework, receive feedback, see grades

Piazza:

- Ask/answer questions

Collaboration Policy

Pertinent summary of UChicago's academic honesty guidelines: `http://college.uchicago.edu/advising/academic-integrity-student-conduct`

- Never copy work from any other source and submit it as your own.
- Never allow your work to be copied.
- Never submit work identical to another student's.
- Document all collaboration.
- Cite your sources.
- Never submit work which you can't explain to the teaching staff

Collaboration Policy

- Nancy has been working dilligently on Question 2 for an hour but feels stuck. Nancy says to her friend Ivan, “I’m completely stuck on Question 2. Can you give me a hint?".
- When Nancy turns in her homework, she should acknowledge on her homework that she got a hint on Question 2 from Ivan.

Collaboration Policy

- It's 1pm on the day of the lecture, and Salem is just starting his homework. He doesn't know how to solve most of it, so he asks Kylie to send him her code with the intention of paraphrasing.
- In this instance, Salem should submit what work he has done on the assignment. There are many homework assignments in this course, and a few lower scores are ok, provided he starts his next assignment a little earlier.

Collaboration Policy

- Peter is confused about the subject of recursion. He looks on the internet for some code examples with explanations, and it happens that one of the examples is one of the homework problems.
- Peter should explain this situation. He should be sure to note at the top of his homework which website the problem was found on.

What to know

- Converting math to code: integers, rationals, reals
- Booleans and if
- Function syntax
- Strings and symbols
- Collaboration policy