



CS1 Recitation

Week 1

Admin

- READ YOUR CS ACCOUNT E-MAIL!!!
 - Important announcements, like when the cluster will be unavailable, or when you need to reset your password.
- If you want to forward your e-mail:
 - Make a `.forward` file in your home directory
 - Put your preferred e-mail address in there
 - E-mail will be forwarded (no copy will be kept on CS cluster)

Converting Math to Scheme

- $4 + 9 * 3 = ???$
 - 31, not 39.
 - The $*$ is applied first, then the $+$
 - $*$ has *higher precedence* than $+$
- Scheme doesn't have precedence rules
 - All combinations are explicitly wrapped in parentheses
- When converting math expressions to Scheme, do highest precedence operations first
 - $(* 9 3)$
 - $(+ 4 (* 9 3))$

More Complicated Example

- $3 + 15 / 4 + 6 * 9$
- $/$ and $*$ have the highest precedence...
 - $(+ 3 (/ 15 4) (* 6 9))$
- Could also write: $(+ (+ 3 (/ 15 4)) (* 6 9))$
- Arithmetic operators can take more than two arguments
 - Simplify expressions, if possible.
 - Make sure simplified expression still accurately represents original (i.e. don't flip signs, do reciprocals, etc.)



Evaluating Scheme Combinations

Evaluating combinations is very simple.

1. Evaluate operands
 - Scheme doesn't specify an order for operands.
 - Left-to-right is simple and clear, so do that.
2. Evaluate the operator
 - Don't forget this step!
 - In CS1, we do this *after* evaluating operands
3. Apply operator to operands
4. State result of computation

Simple Example

- $(+ 4 (* 9 3))$
 - Evaluate 4 \rightarrow 4
 - Evaluate $(* 9 3)$
 - Evaluate 9 \rightarrow 9
 - Evaluate 3 \rightarrow 3
 - Evaluate * \rightarrow [primitive procedure *]
 - Apply * to 9, 3 \rightarrow 27
 - Result: 27
 - Evaluate + \rightarrow [primitive procedure +]
 - Apply + to 4, 27 \rightarrow 31
 - **Result: 31**

Numbers in Scheme

- What do you get when you run `(/ 3 9)` ?
 - $1/3$, *not* $0.333\dots$
 - $1/3$ is an exact answer, $0.333\dots$ is not
 - Scheme preserves exactness, if it can
- Math will result in fractions by default, unless you use decimal in the operands.
 - `(/ 3 9)` = $1/3$
 - `(/ 3.0 9)` = $0.333\dots$
 - `(/ 3 9.0)` = $0.333\dots$
 - `(/ 3.0 9.0)` = $0.333\dots$

Definitions in Scheme

- (define x 53)
- Remember – define is a special form! Skip first operand, evaluate second operand.
 - $53 \rightarrow 53$
 - Associate 53 with x (can also say “Bind 53 to x”)
- Don't forget to evaluate define statements
 - State that it's a special form, so that it's clear why second operand is evaluated first.

Lambda Expressions

- `(lambda (x) (* x x))`
 - lambda creates a procedure in Scheme
- Evaluates to itself:
 - `(lambda (x) (* x x)) → (lambda (x) (* x x))`
 - Or: `(lambda (x) (* x x)) → itself`
- Can be combined with define:
 - `(define sq (lambda (x) (* x x)))`
 - Now can use sq as a function: `(sq 5)` produces 25

Evaluating Defines with Lambdas

- Evaluate (define sq (lambda (x) (* x x)))
 - define is a special form. Skip first operand. Evaluate second operand.
 - Evaluate (lambda (x) (* x x)) → itself
 - Associate the new function with the name “sq”

Simplified Function Definitions

- Can also say: `(define (sq x) (* x x))`
- This is *syntactic sugar* for:
 - `(define sq (lambda (x) (* x x)))`
- When evaluating the “sugared” form, you must explicitly state the desugaring step.
- Evaluate `(define (sq x) (* x x))`
 - Desugar to: `(define sq (lambda (x) (* x x)))`
 - Define is a special form. Evaluate second operand.
 - *Continue the evaluation of define as usual...*

Using Our New Function

- Evaluate (sq 5)
 - Evaluate 5 \rightarrow 5
 - Evaluate sq \rightarrow (lambda (x) (* x x))
 - Apply (lambda (x) (* x x)) to 5
 - Substitute 5 for x in (* x x) \rightarrow (* 5 5)
 - Evaluate (* 5 5)
 - Evaluate 5 \rightarrow 5
 - Evaluate 5 \rightarrow 5
 - Evaluate * \rightarrow [primitive procedure *]
 - Apply * to 5, 5 \rightarrow 25
 - Result: 25
 - Result: 25
 - **Result: 25**

What's Wrong Here?

- You type: 15 - 6
- Is this valid Scheme?
 - No! Needs to be a combination surrounded with parentheses
 - This is a syntax error – it doesn't have the right form.

What's Wrong Here?

- Now you type (15 - 6)
- Is this valid Scheme?
- What is the problem?
 - Operator needs to be first in Scheme, then operands
- What does the Scheme interpreter say?
 - "15 isn't a procedure"
- Interpreter feedback is very limited.
 - It can't tell you what you meant, only what it expects.

What's Wrong Here?

- You type `(- * 6 3 4)`
- Is this valid Scheme?
 - It's a valid combination...
 - Functions *can* be operands in Scheme.
 - However, evaluation *does* result in an error.
- Can't subtract a number from a function
 - This is a *semantic* error, not a syntax error.
 - Expression has correct form, but doesn't make sense.

Writing Useful Procedures

- Want to write a Scheme procedure to convert miles to kilometers.
 - 1.6 km in a mile – easy conversion!
- Give your procedures understandable names
 - Bad names: m2k, convert, f, answer5b
 - Good name: miles-to-km
- Document your procedures with comments!
 - **:: Converts a distance in miles, to kilometers.**
 - (define miles-to-km ...)

Converting Miles to Kilometers

- Can write it the “sugar-free” way
 - (define miles-to-km (lambda (miles) (* 1.6 miles)))
- Or, the sugared way
 - (define (miles-to-km miles) (* 1.6 miles))
- Use meaningful variable names too.
 - Really important for maintainable programs
- Test your code!
 - Can use google.com to check your answers
 - “15 miles in km” → 15 miles = 24.14016 kilometers

Reuse Your Work!

- “The best programmers are fundamentally lazy.”
 - Don’t do unnecessary work. (*Constructive laziness.*)
- Now you want to convert furlongs to kilometers
 - 8 furlongs = 1 mile
 - (define (furlongs-to-km f) (miles-to-km (/ f 8)))
- Better yet, write a separate furlongs-to-miles fn.
 - Then furlongs-to-km can just compose the two
 - (define (furlongs-to-miles f) (/ f 8))
 - (define (furlongs-to-km f)
 (miles-to-km (furlongs-to-miles f)))