Types

- We've seen how every value has a type
  - #f - Boolean
  - 4.5 - number
  - “Hi” - String
- Allows us to analyze the form and behavior of expressions

Procedures have types too

- In Scheme, procedures have types too
- Denote types by indicating the types of each argument and the type of the return value
- Example:
  - number → number indicates a procedure that takes a number and returns a number

Type analysis

- Consider + for two numbers
- Type: number, number → number
- This says:
  - Additions takes two numbers and results in a number

Examples

- What are the types of
  - Square
  - =
  - Append
  - Null?
Procedure types

- Type of a procedure is a contract
  - The operands have the specified type, the result will have the specified type
- Otherwise behavior is undefined
  - Could be an error (bad) or arbitrary behavior (worse!)

Higher Order Procedures

- What does this procedure do?
  
  ```scheme
  (define (sumto n op)
    (if (<= n 0)
      (op n)
      (+ (op n) (sumto (- n 1) op))))
  ```

Apply the type analysis

- What does (sumto 3 square) evaluate to?
- What is the type of sumto?
  
  ```scheme
  (define (sumto n op)
    (if (<= n 0)
      (op n)
      (+ (op n) (sumto (- n 1) op))))
  ```

Higher Order Procedures

- What does (proc1 6) evaluate to?
- What is the type of make-adder?
- What does ((make-adder 10) 11) evaluate to?

```
(define make-adder
  (lambda (x)
    (lambda (y)
      (+ x y))))

(define proc1 (make-adder 5))
```
Overall Idea

- Procedures are like any other type
  - Use procedures as arguments
  - Create procedures that return procedures

map

- Define a procedure map that takes an operation and a list, and returns a list where the operation has been applied to each item in the list
- Example
  (map square (list 1 2 3 4 5))
  Value: (1 4 9 16 25)

(map (define (map op lst)
  (if (null? lst)
      null
    (cons (op (car lst)
             (map op (cdr lst)))))))

add1

- Now define a procedure add1 that adds 1 to each item in a list

(add1 (define (add1 lst)
       (map (lambda (x) (+ x 1))
             lst)))

Result and type for these expressions

(+ 1 1)
(lambda (x) (+ x 1))
((lambda (x) (+ x 1)) 1)
(define (a x) (+ (x 10) 5))
a
(a 5)
(a square)
New Special Form: and

• \((\text{and} \ arg_1 \ arg_2 \ldots \ arg_n)\)
  – Evaluates arguments in left to right order
  – Stops evaluating after one evaluates to \#f and returns \#f
  – Otherwise, returns value of last argument

New Special Form - or

• \((\text{or} \ arg_1 \ arg_2 \ldots \ arg_n)\)
  – Evaluates arguments in left to right order
  – Stops evaluating after one evaluates to something not false and returns that value
  – Otherwise, returns \#f