Discussion 11

Scheme, Scheme Lists

Aditya Balasubramanian
aditbala [at] berkeley [dot] edu

Slides available at `teaching.aditbala.com`

Announcements

- Homework 8 due Thursday 4/13.
- Midterm 2 regrade requests due Friday 4/21.

Scheme

Scheme

- What is Scheme?
  - Another programming language!
  - A dialect of Lisp (**LIS**t **P**rocessor)
- Allows us to bring together all of our previous knowledge
- Recursion based
  - No iterative loops!
  - Only recursion and tree recursion :)

Scheme Primitives

- What is a primitive?
  - Expressions that are simplified or cannot be divided up further
- What are some primitives
  - Numbers -> Floats, Integers
  - Booleans -> Truth-y values, False-y values
- NOTE
  - Everything other than `#f` will evaluate to `True` in Scheme

Scheme Primitives (Example)

```
scm> 1
1
scm> 2
2
scm> #t
True
scm> #f
False
```

Defining Variables in Scheme

- How do we define variables in Scheme?
  - Use `define`
- `(define <variable name> <value>)`
  - `(define adit 10)`
  - `adit -> 10`
  - Evaluates `<value>` and binds the value to `<variable name>` in the current environment.
- `'`
  - Accesses the `<variable name>` but not the value
  - Useful for when you don't want to modify the or evaluate the `<variable name>`

WWSD (Primitives and Defining Variables)

Scheme Call Expressions

- What are Call Expressions?
  - How we invoke functions
- `(<operator> <operand>)`
  - `<operator>` comes first (different than Python)

    (+ 1 2)
- How do we evaluate?
  - Same as Python
  - Evaluate `<operator>`, `<operand>`, and then apply `<operator>` to `<operands>`

WWSD (Call Expressions)

Scheme Special Forms

- What are Special Forms?
  - Look like Call Expressions, but behave slightly differently
- What do they look like?
  - `define`, `if`, `cond`, `and`, `or`, `lambda`, `begin`, `else`

Scheme `if`

- `(if <predicate> <if-true> [if-false])`
  - Evaluate `<predicate>`
  - if `<predicate>` is truth-y
    - evaluate `<if-true>`
  - if `<predicate>` is false-y
    - evaluate `<if-false>`
- `(if (< 4 5) 1 2)`
- `1`

Scheme `cond`

- `(cond (<pred1> <if-pred1>) ... (<predn> <if-predn>) [(else <else-expression>)])`
  1. Evaluate the predicates `<pred1>, <pred2>, ..., <predn>` in order until you reach one that evaluates to a truth-y value, then return corresponding `<if-predn>`.
  2. If none of the predicates are truth-y and there is an else clause, evaluate and return `<else-expression>`.
- `(cond ((< 4 5) 1) (else 2) )`
- `1`

Scheme Booleans

- `and`, `or`, `not`
  - Similar to Python (short-circuits)
- Equivalence
  - `=` -> numbers
  - `eq?` -> check if same object (`is`)
  - `equal?` -> check if contents are the same (`==`)

Scheme Lambdas

- All functions are Lambdas in Scheme!

```
scm> (define square (lambda (x) (* x x)))
square
scm> (define (square x) (* x x))          ; Same as above
square
scm> square
(lambda (x) (* x x))
scm> (square 4)
16
```

Q1: Virahanka-Fibonacci

```
def virfib(n):
    if n == 0 or n == 1:
        return n
    return virfib(n - 1) + virfib(n - 2)
```

Scheme Lists

- What are Scheme lists?
  - Linked Lists!!!
- Syntax difference
  - `car <any value>` / `Link.first`
  - `cdr <nil or another Scheme list>` / `Link.rest`

```
scm> nil
()
scm> (define lst (cons 1 (cons 2 (cons 3 nil))))
lst
scm> lst
(1 2 3)
scm> (car lst)
1
scm> (cdr lst)
(2 3)
```

Scheme Lists (Creation)

- `(cons <first> <rest>)`
  - `<rest>` must be another list or `nil`
- `(list <item1> ... <itemn>)`
  - Returns a list with the `<item1> ... <itemn>` in order as its elements.
- `` `(<expr>) `` or `(quote <expr>)`
  - Returns the list exactly as typed, without evaluating any of the individual elements (different than `list` and `cons`)

`=, eq?, equal?`

- `(= <a> <b>)`
  - returns if `a` equals `b`, both must be numbers
- `(eq? <a> <b>)`
  - returns if `a` and `b` are equivalent primitive values, or are the same object, like `is` in Python
- `(equal? <a> <b>)`
  - returns if `a` and `b` are *pairs/lists* that have the same contents, like `==` in Python, if `a` and not `b`, behaves like `eq?`

Worksheet

Thank you!!!

Anon Feedback -> https://tinyurl.com/adit-anon