Functional Programming

ACM12
Zhao Zhuoyue

What is functional programming?

- A programming paradigm.
- Models computation as the evaluation of functions.
- Avoids side effects.
- Functions are first-class objects.

Scheme

- One of the two dialects of LISP.
- Developed by Guy L. Steele and Gerald Jay Sussman in 1975.

- Classic Textbook: *Structure and Interpretation of Computer Programs*
- Racket (http://racket-lang.org/)

An example: factorial

Functional:

```
(define (fact n)
   (if (> n 0)
      (* n (fact (- n 1)))
      1))
```

Imperative:

```
int fact(int n){
        int fact = 1;
        for (int i = 1; i <= n; ++i)
                fact *= i;
}
```

Outline

- Expressions
- Naming
- Procedures
- Lexical scope vs. Dynamic scope
- Applicative order vs. Normal order
- Conditional expressions
- Pairs and lists
- Higher-order functions

Expressions

- Prefix expressions
- +, -, *, /, modulo

- Examples:

```
(+ 1 2)                 ===> 3
(* 25 4 12)             ===> 1200
(+ (* 3 5) (- 10 6))    ===> 19
```

Coding Style

- When code becomes complex, code with a good style is easy to read and debug.

```
(+ (* 3
      (+ (* 2 4)
         (+ 3 5)))
   (+ (- 10 7)
      6))
```

Naming

- Defining variables
- Binding a variable to an object rather than assigning a value.
- (define <name> <expression>)

```
(define pi 3.14)
(define radius 2)
(define area (* pi (* radius radius)))
```

Procedures

- Defines procedures (functions)
- (define (<variable> <formals>) <body>)

- Creates anonymous procedure
- (lambda (<formals>) <body>)

```
(define (square x) (* x x))
(define (sum-of-square x y)
    (+ (square x) (square y)))
```

Lexical Scope OR Dynamic Scope

- What is the result of the following code:

```
(define x 5)
(define (id) x)
(define (f x) (id))

(f 4)   ===> 5 or 4?
```

- Scheme requires implementation to be lexical scoped.

Applicative order

- Evaluates all the arguments before calling a function.

```
(sum-of-square (+ 1 2) (+ 1 3))
(sum-of-square 3 4)
(+ (square 3) (square 4))
(+ (* 3 3) (* 4 4))
(+ 9 15)
25
```

Normal order

- Apply the function first. Delay the evaluation of arguments until necessary.

```
(sum-of-square (+ 1 2) (+ 1 3))
(+ (square (+ 1 2)) (square (+ 1 3))
(+ (* (+ 1 2) (+ 1 2)) (* (+ 1 3) (+ 1 3)))
(+ (* 3 3) (* 4 4))
(+ 9 15)
25
```

Conditional expressions

- Special forms:
  - if
  - cond
  - and
  - or

If

- (if <test> <consequence>)
- (if <test> <consequence> <alternative>)

```
(define (sgn x)
    (if (> x 0)
        1
        (if (= x 0)
            0
            -1)))
```

cond

- (cond (<test1> <expr1>) …)

```
(define (sgn x)
    (cond ((> x 0) 1)
          ((= x 0) 0)
          (else -1)))
```

and/or

- (and <test1> …)
- Expressions is evaluated from left to right until #f is encountered or all has been evaluated.

- (or <test1> …)
- Expressions is evaluated from left to right until a true value is encountered or all has been evaluated.

Why special forms?

- Does the following work?

```
(define (new-if test expr alter)
    (cond (test expr)
          (else alter)))

(new-if (> 1 0) 1 (/ 1 0))
```

Why special forms?

- What about the following?

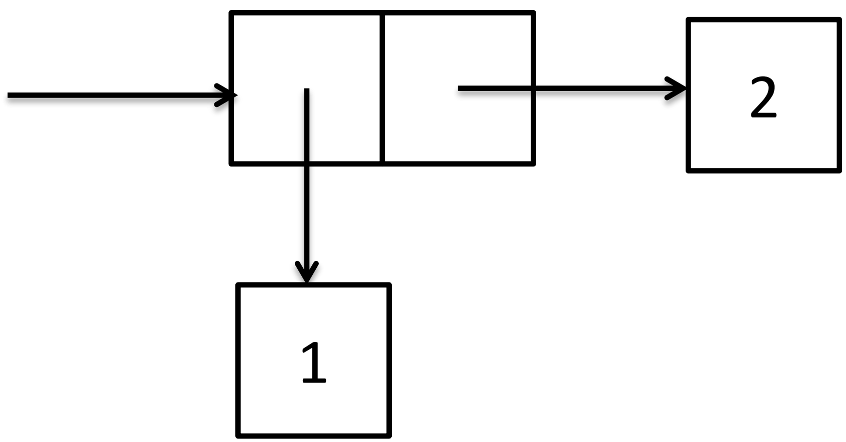
```
(define (new-and expr1 expr2)
    (if expr1
        (if expr2 expr2 #f)
        #f))
```

- The same reason.

Pairs and lists

- Ordered pairs <x, y>

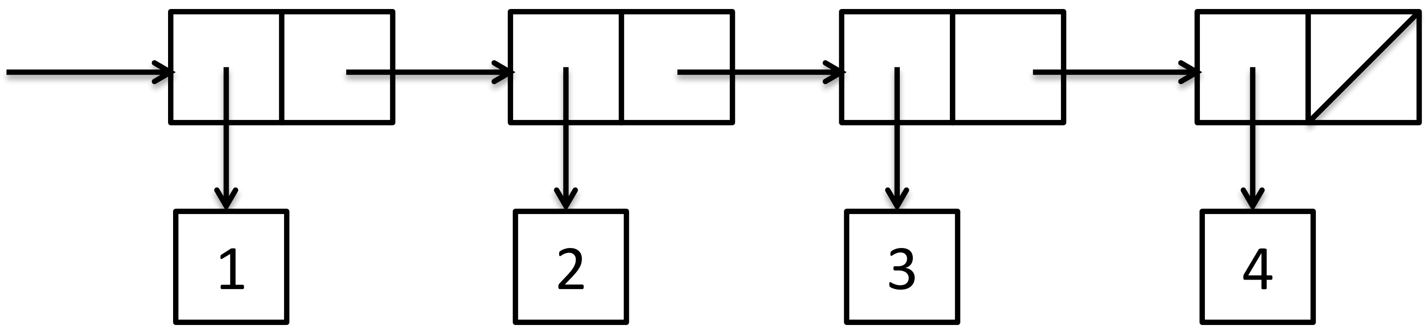
```
(define p (cons 1 2))
(car p)        ===> 1
(cdr p)        ===> 2
```



Lists

- Constructing lists from ordered pairs

```
(cons 1 (cons 2 (cons 3 (cons4 '()))
(list 1 2 3 4)
```



Lists

```
(length '(1 2 3 4)) ===> 4
(car '(1 2 3 4)) ===> 1
(cdr '(1 2 3 4)) ===> (2 3 4)
(caddr '(1 2 3 4)) ===> 3
```

An example: square root

- Calculates the square root of a number y.
- Newton's method

$$x_{k+1} = \frac{1}{2}(x_k + \frac{y}{x_k})$$

- Iterates until converge.

```
(define (iter guess x)
    (if (good-enough? guess x)
        guess
        (iter (improve guess x) x)
```

An example: square root

- Define good-enough? and improve

```
(define (good-enough? guess x)
    (< (abs (- (square guess) x)) 0.001))

(define (improve guess x)
    (average guess (/ x guess)))
```

- square, average

An example: square root

- Put them together (DIY)

- How to generalize the function to implement newton’s method to solve any equation?

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

Higher-order functions

- Functions that takes functions as input or outputs a function

```
(define (square x) (* x x))
(define (cube x) (* x (square x)))
(apply + (map square '(1 2 3)))    ===> 14
(apply + (map cube '(1 2 3)))      ===> 36
```

Higher order functions

- Compose two functions

```
(define (comp f g)
  (lambda args
    (f (apply g args))))

((comp sqrt
       (lambda args
         (apply + (map square args)))) 3 4)
==> 5
```

Example: Newton's method

```
(define (improve f x)
    (- x (/ (f x)
            ((derivative f) x))))
```

- What about the derivative? Approximate it!

```
(define (derivative f)
    (lambda (x)
      (/ (- (f (+ x 0.0001)) (f x))
             0.0001)))
```

- Complete the function yourself.

Let’s call it a day.