DDG University

Informal Goals

1. Learn new things related to technology.
2. Learn from each other.
3. Foster inter-team building.
4. To become better engineers.

Search for DDG University in Asana.

Structure and Interpretation of Computer Programs *(SICP)*

by Harold Abelson and Gerald Jay Sussman

1.3 Formulating Abstractions with Higher-Order Procedures

1. ***Using functions as data***

First Class Functions

- They may be named by variables.
- They may be passed as arguments to procedures.
- They may be returned as the results of procedures.
- They may be included in data structures.

Passing Procedures as Arguments

1. Find the sum of all integers between a and b:

```
(define (sum-integers a b)
  (if (> a b)
      0
      (+ a (sum-integers (+ a 1) b))))
```

2. Find the sum of all cubes between a and b:

```
(define (sum-cubes a b)
 (if (> a b)
     0
     (+ (cube a)
        (sum-cubes (+ a 1) b))))
```

3. Compute a Leibniz series:

```
(define (pi-sum a b)
 (if (> a b)
     0
     (+ (/ 1.0 (* a (+ a 2)))
        (pi-sum (+ a 4) b))))
```

$$\frac{1}{1\cdot 3} + \frac{1}{5\cdot 7} + \frac{1}{9\cdot 11} + \ldots$$

A Common Pattern

They all fit into this "template":

```
(define (⟨name⟩ a b)
  (if (> a b)
      0
      (+ (⟨term⟩ a)
         (⟨name⟩ (⟨next⟩ a) b))))
```

Which can be expressed as:

```
(define (sum term a next b)
  (if (> a b)
      0
      (+ (term a)
         (sum term (next a) next b))))
```

Sum Cubes

Find the sum of all cubes between a and b:

```
(define (sum-cubes a b)
  (if (> a b)
      0
      (+ (cube a)
         (sum-cubes (+ a 1) b))))
```

with the help of:

```
(define (sum term a next b)
  (if (> a b)
      0
      (+ (term a)
         (sum term (next a) next b))))
```

becomes:

```
(define (inc n) (+ n 1))

(define (sum-cubes a b)
  (sum cube a inc b))
```

```
(sum-cubes 1 10)
;3025
```

Sum Integers

Find the sum of all integers between a and b:

```
(define (sum-integers a b)
  (if (> a b)
      0
      (+ a (sum-integers (+ a 1) b))))
```

with the help of:

```
(define (sum term a next b)
  (if (> a b)
      0
      (+ (term a)
         (sum term (next a) next b))))
```

becomes:

```
(define (inc n) (+ n 1))

(define (identity x) x)

(define (sum-integers a b)
  (sum identity a inc b))
```

```
(sum-integers 1 10)
;55
```

Leibniz Series

```
          1     1     1
        --- + --- + ---- + ...
        1*3   5*7   9*11
```

```
(define (pi-sum a b)
  (if (> a b)
      0
      (+ (/ 1.0 (* a (+ a 2)))
         (pi-sum (+ a 4) b))))
```

with the help of:

```
(define (sum term a next b)
  (if (> a b)
      0
      (+ (term a)
         (sum term (next a) next b))))
```

becomes:

```
(define (pi-sum a b)
  (define (pi-term x)
    (/ 1.0 (* x (+ x 2))))
  (define (pi-next x)
    (+ x 4))
  (sum pi-term a pi-next b))
```

```
(* 8 (pi-sum 1 1000))
;3.139592655589783
```

A Little More Lisp

- *lambda*
- *let*

lambda

Lisp:

```
(lambda (x) (+ x 4))

(f 3)
> 7
```

Perl:

```
sub {
    my ($x) = @_;
    return $x + 4;
};

$f->(3);
> 7
```

Special form:

```
(lambda (⟨formal-parameters⟩) ⟨body⟩)
```

let

$$(x+1)^2 + (y+1)^2$$

```
(define (f x y)
 (define (f-helper a b)
   (+ (square a)
      (square b)))
 (f-helper (+ 1 x)
           (+ 1 y)))

(f 2 3)
;25
```

```
(define (f x y)
  ((lambda (a b)
    (+ (square a)
       (square b)))
  (+ 1 x)
  (+ 1 y)))
```

```
(define (f x y)
  (let ((a (+ 1 x))
        (b (+ 1 y)))
    (+ (square a)
       (square b))))
```

```
(let ((⟨var1⟩ ⟨exp1⟩)
      (⟨var2⟩ ⟨exp2⟩)
      …
      (⟨varN⟩ ⟨expN⟩))
  ⟨body⟩)
```

Procedures as Return Values

```
(define (average-damp f)
  (lambda (x)
    (average x (f x))))
```

```
((average-damp square) 10)
;55
```

Wrapping-up

- They may be named by variables.

  (define, let)
- They may be passed as arguments to procedures.

  (sum term (next a) next b))
- They may be returned as the results of procedures.

  (average-damp, lambda)
- They may be included in data structures.

  (???)

That's all for section 1.3.

Thanks!