Structure and Interpretation of Computer Programs

COMP200

SUMMARY

What have we learned?

- **type**: a set of values
  - every value has a type
- procedure types include .... → ...
  - number of arguments required
  - type of each argument
  - type of result of the procedure
- **types**: a mathematical theory for reasoning **efficiently** about programs
  - useful for preventing certain common types of errors
  - basis for many analysis and optimization algorithms

REMEMBER

Procedure Abstraction

```
(* 2 2)
(* 57 57)
(* k k)
```

REMEMBER

Procedure Abstraction

```
(* 2 2)
(* 57 57)
(* k k)

(lambda (x) (* x x))
```

parameter

actual pattern

REMEMBER

Procedure Abstraction

```
(* 2 2)
(* 57 57)
(* k k)

(lambda (x) (* x x))

(define square (lambda (x) (* x x)))
```

REMEMBER

Procedure Abstraction

```
(* 2 2)
(* 57 57)
(* k k)

(lambda (x) (* x x))

(define square (lambda (x) (* x x)))
```

number → number

PROCEDURE ABSTRACTION

Other Common Patterns

$$1 + 2 + \ldots + 100 = (100 * 101)/2 \qquad \sum_{k=1}^{100} k$$

$$1 + 4 + 9 + \ldots + 100^2 = (100 * 101 * 201)/6 \qquad \sum_{k=1}^{100} k^2$$

$$1 + 1/3^2 + 1/5^2 + \ldots + 1/101^2 = \pi^2/8 \qquad \sum_{k=1 \text{ odd}}^{101} k^{-2}$$

PROCEDURE ABSTRACTION

Other Common Patterns

$$\sum_{k=1}^{100} k$$

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

$$\sum_{k=1}^{100} k^2$$

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

$$\sum_{k=1 \text{ odd}}^{101} k^{-2}$$

```
(define (pi-sum a b)
  (if (> a b)
      0
      (+ (/ 1 (square a))
         (pi-sum (+ a 2) b))))
```

PROCEDURE ABSTRACTION

Other Common Patterns

$$\sum_{k=1}^{100} k$$

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

$$\sum_{k=1}^{100} k^2$$

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

$$\sum_{k=1 \text{ odd}}^{101} k^{-2}$$

```
(define (pi-sum a b)
  (if (> a b)
      0
      (+ (/ 1 (square a))
         (pi-sum (+ a 2) b))))
```

PROCEDURE ABSTRACTION

Other Common Patterns

$$\sum_{k=1}^{100} k$$

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

$$\sum_{k=1}^{100} k^2$$

```scheme
(define (sum-squares a b)
  (if (> a b)
      0
      (+ (square a)
         (sum-squares (+ 1 a) b))))
```

$$\sum_{k=1 \text{ odd}}^{101} k^{-2}$$

```scheme
(define (pi-sum a b)
  (if (> a b)
      0
      (+ (/ 1 (square a))
         (pi-sum (+ a 2) b))))
```

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

PROCEDURE ABSTRACTION

Other Common Patterns

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

PROCEDURE ABSTRACTION

Other Common Patterns

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

$$(\underbrace{\text{number} \rightarrow \text{number}}_{\text{procedure}},\ \text{number},\ \underbrace{\text{number} \rightarrow \text{number}}_{\text{procedure}},\ \text{number}) \rightarrow \text{number}$$

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

HIGHER ORDER PROCEDURES

Other Common Patterns

```scheme
(define (sum-integers1 a b)
  (sum (lambda (x) x) a (lambda (x) (+ x 1)) b))

(define (sum-squares1 a b)
  (sum square a (lambda (x) (+ x 1)) b))

(define (pi-sum1 a b)
  (sum (lambda (x) (/ 1 (square x))) a (lambda (x) (+ x 2)) b))
```

PROCEDURE ABSTRACTION

Other Common Patterns

$$\sum_{k=1}^{100} k$$

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

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

```
(define (sum-integers1 a b)
  (sum (lambda (x) x) a (lambda (x) (+ x 1)) b))
```

PROCEDURE ABSTRACTION

Other Common Patterns

$$\sum_{k=1}^{100} k^2$$

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

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

```
(define (sum-squares1 a b)
  (sum square a (lambda (x) (+ x 1)) b))
```

PROCEDURE ABSTRACTION

Other Common Patterns

$$\sum_{k=1 \text{ odd}}^{101} k^{-2}$$

```
(define (pi-sum a b)
  (if (> a b)
      0
      (+ (/ 1 (square a))
         (pi-sum (+ a 2) b))))
```

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

```
(define (pi-sum1 a b)
  (sum (lambda (x) (/ 1 (square x))) a (lambda (x) (+ x 2)) b))
```

HIGHER ORDER PROCEDURES

Computing Derivatives

$$f : x \mapsto x^2$$

$$Df : x^2 \mapsto 2x$$

$$f : x \mapsto x^3$$

$$Df : x^3 \mapsto 3x^2$$

HIGHER ORDER PROCEDURES

Computing Derivatives

$$f : x \mapsto x^2$$

$$Df : x^2 \mapsto 2x$$

```
(define f
  (lambda (x) (* x x)))
```

$$f : x \mapsto x^3$$

$$Df : x^3 \mapsto 3x^2$$

```
(define f
  (lambda (x) (* x x x)))
```

HIGHER ORDER PROCEDURES

Computing Derivatives

$$f : x \mapsto x^2$$

$$Df : x^2 \mapsto 2x$$

$$f : x \mapsto x^3$$

$$Df : x^3 \mapsto 3x^2$$

$$Df \approx \frac{f(x+\epsilon) - f(x)}{\epsilon}$$

HIGHER ORDER PROCEDURES

Computing Derivatives

$$Df \approx \frac{f(x+\epsilon) - f(x)}{\epsilon}$$

```
(define deriv
  (lambda (f)
```

HIGHER ORDER PROCEDURES

Computing Derivatives

$$Df \approx \frac{f(x+\epsilon) - f(x)}{\epsilon}$$

```
(define deriv
  (lambda (f)
    (lambda (x)
      (/ (- (f (+ x epsilon)) (f x))
         epsilon))))
```

HIGHER ORDER PROCEDURES

Computing Derivatives

$$Df \approx \frac{f(x+\epsilon) - f(x)}{\epsilon}$$

```
(define deriv
  (lambda (f)
    (lambda (x)
      (/ (- (f (+ x epsilon)) (f x))
         epsilon))))
```

...

HIGHER ORDER PROCEDURES

Computing Derivatives

$$Df \approx \frac{f(x+\epsilon) - f(x)}{\epsilon}$$

```
(define deriv
  (lambda (f)
    (lambda (x)
      (/ (- (f (+ x epsilon)) (f x))
         epsilon))))
```

(number → number) → (number → number)

HIGHER ORDER PROCEDURES

Using Derivatives

```
(define deriv
  (lambda (f)
    (lambda (x)
      (/ (- (f (+ x epsilon)) (f x))
         epsilon))))
```

```
(define square
  (lambda (y) (* y y)))

(define epsilon 0.001)

((deriv square) 5)
```

HIGHER ORDER PROCEDURES

Using Derivatives

```
(define deriv
  (lambda (f)
    (lambda (x)
      (/ (- (f (+ x epsilon)) (f x))
         epsilon))))
```

```
(define square
  (lambda (y) (* y y)))

(define epsilon 0.001)

((deriv square) 5)
```

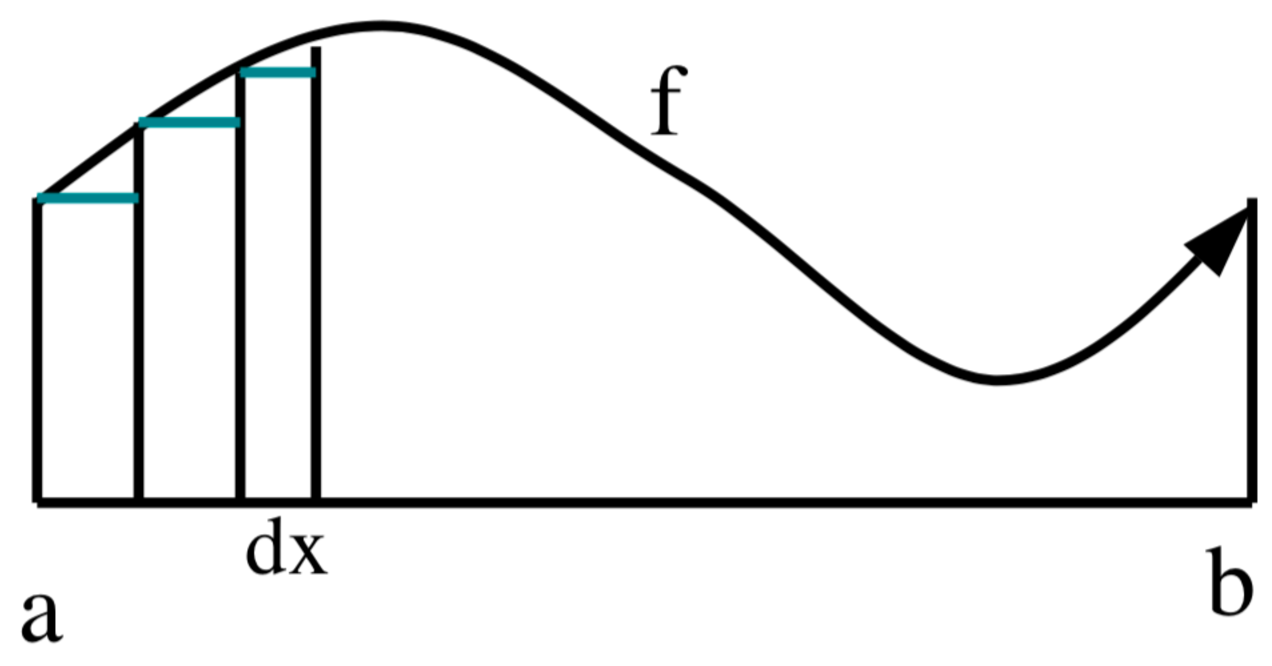
```
((lambda (x)
   (/ (- ((lambda (y) (* y y))(+ x epsilon))
         ((lambda (y) (* y y)) x)))
   epsilon) 5)

(lambda (x)
   (/ (- ((lambda (y) (* y y))(+ 5 epsilon))
         ((lambda (y) (* y y)) 5)))
   epsilon)
```

HIGHER ORDER PROCEDURES

Why?

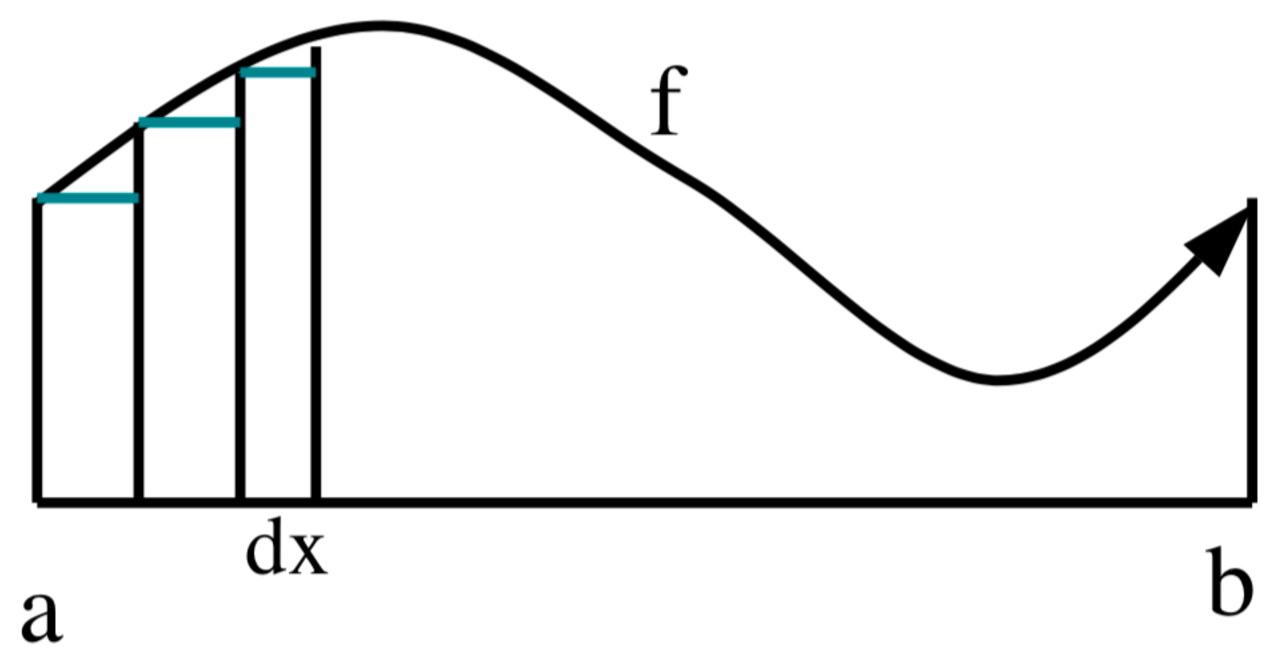
HIGHER ORDER PROCEDURES

Integration



HIGHER ORDER PROCEDURES

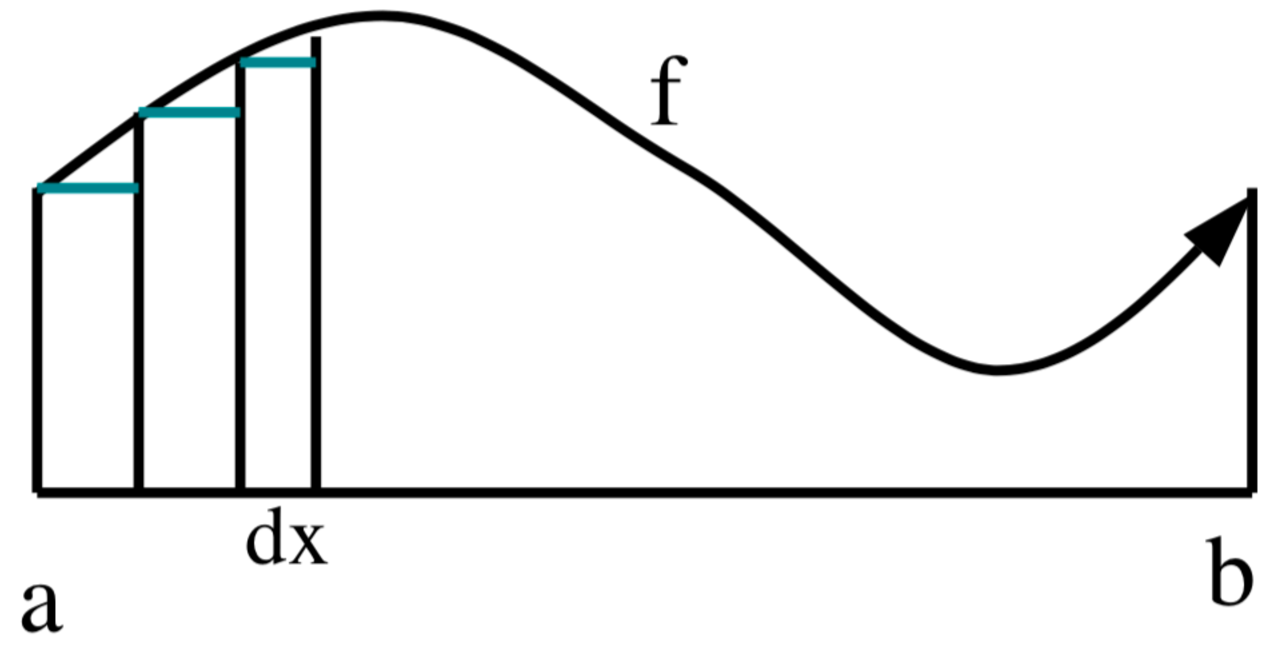
Integration



$$dx\ (f(a) + f(a + dx) + f(a + 2dx) + \cdots + f(b))$$

HIGHER ORDER PROCEDURES

Integration




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

(define (integral f a b n)
  (let ((delta (/ (- b a) n)))
    (* (sum f a (lambda (x) (+ x delta)) b) delta)))
```

$$dx\ (f(a) + f(a + dx) + f(a + 2dx) + \cdots + f(b))$$

HIGHER ORDER PROCEDURES

Integration

```
(define my-atan
  (lambda (a)
    (integral (lambda (x) (/ 1 (+ 1 (square x)))) 0 a 1000)))

(my-atan 2.0)
(my-atan 3.0)
```

HIGHER ORDER PROCEDURES

Fixed Points

$$\sqrt{x} = x/\sqrt{x}$$

$$f : y \mapsto x/y$$

HIGHER ORDER PROCEDURES

Fixed Points

$$\sqrt{x} = x/\sqrt{x}$$

$$f : y \mapsto x/y$$

if we can find a $y = \sqrt{x}$

then $f(y) = y$

such a y is called a fixed point of f

HIGHER ORDER PROCEDURES

Fixed Points

- Given a guess x_1, let new guess by $f(x_1)$
- Keep computing f of last guess, until close enough

HIGHER ORDER PROCEDURES

Fixed Points

- Given a guess x_1, let new guess by $f(x)$
- Keep computing f of last guess, until close enough

```
(define (close? u v)
  (< (abs (- u v)) 0.0001))

(define (fixed-point f i-guess)
  (define (try g)
    (if (close? (f g) g)
        (f g)
        (try (f g))))
  (try i-guess))
```

HIGHER ORDER PROCEDURES

Fixed Points

```
(fixed-point (lambda (x) (+ 1 (/ 1 x))) 1)
```

1.6180

HIGHER ORDER PROCEDURES

Fixed Points

```
(fixed-point (lambda (x) (+ 1 (/ 1 x))) 1)
```

1.6180

$x = 1 + (1/x)$ when $x = (1 + \sqrt{5})/2 = 1.6180$

```scheme
(define (fixed-point f i-guess)
  (define (try g)
    (if (close? (f g) g)
        (f g)
        (try (f g))))
  (try i-guess))
```

HIGHER ORDER PROCEDURES

Fixed Points

```scheme
(define (my-sqrt x)
  (fixed-point
   (lambda (y) (/ x y)) 1))
```

```
(define (fixed-point f i-guess)
  (define (try g)
    (if (close? (f g) g)
        (f g)
        (try (f g))))
  (try i-guess))
```

HIGHER ORDER PROCEDURES

Fixed Points

```
(define (my-sqrt x)
  (fixed-point
   (lambda (y) (/ x y)) 1))
```

What happens if we try: `(my-sqrt 2)`

HIGHER ORDER PROCEDURES

Fixed Points

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

HIGHER ORDER PROCEDURES

Fixed Points

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

(number → number) → (number → number)

HIGHER ORDER PROCEDURES

Fixed Points

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

```
((average-damp square) 10)
((lambda (x) (average x (square x))) 10)
(average 10 (square 10))
```

HIGHER ORDER PROCEDURES

Fixed Points

```
(define (fixed-point f i-guess)
  (define (try g)
    (if (close? (f g) g)
        (f g)
        (try (f g))))
  (try i-guess))
```

```
(define (my-sqrt x)
  (fixed-point
   (average-damp
    (lambda (y) (/ x y)))
   1))
```

HIGHER ORDER PROCEDURES

Fixed Points

```
(define (my-sqrt x)
  (fixed-point
   (average-damp
    (lambda (y) (/ x y)))
   1))
```

```
(define (cbrt x)
  (fixed-point
   (average-damp
    (lambda (y) (/ x (square y))))
   1))
```

HIGHER ORDER PROCEDURES

Example

```
(define hop1
  (lambda (f x)
    (+ 2 (f (+ x 1)))))

(hop1 square 3)
```

HIGHER ORDER PROCEDURES

Example

```
(define hop1
  (lambda (f x)
    (+ 2 (f (+ x 1)))))

(hop1 square 3)
(+ 2 (square (+ 3 1)))
(+ 2 (square 4))
(+ 2 (* 4 4))
(+ 2 16)
18

(hop1 (lambda (x) (* x x)) 3)
```

HIGHER ORDER PROCEDURES

Example

```
(define hop1
  (lambda (f x)
    (+ 2 (f (+ x 1)))))
```

(number → number), number → number

↑ procedure   ↑ number   ↑ number

HIGHER ORDER PROCEDURES

Example

```
(define compose
  (lambda (f g x)
    (f (g x))))

(compose square double 3)
```

HIGHER ORDER PROCEDURES

Example

```
(define compose
  (lambda (f g x)
    (f (g x))))

(compose square double 3)
(square (double 3))
(square (* 3 2))
(square 6)
(* 6 6)
36
```

HIGHER ORDER PROCEDURES

Example

```
(define compose
  (lambda (f g x)
    (f (g x))))
```

...

HIGHER ORDER PROCEDURES

Example

```
(define compose
  (lambda (f g x)
    (f (g x))))
```

(number → number), (number → number), number → number

HIGHER ORDER PROCEDURES

Example

```scheme
(define compose
  (lambda (f g x)
    (f (g x))))

(compose
 (lambda (p)
   (if p "hi" "bye"))
 (lambda (x)
   (> x 0))
 -5)
```

boolean → string

number → boolean

number

result: string

HIGHER ORDER PROCEDURES

Example

```
(define compose
  (lambda (f g x)
    (f (g x))))
```

```
(compose < square 5)
(compose square double "hi")
```

HIGHER ORDER PROCEDURES

Example

```
(define compose
  (lambda (f g x)
    (f (g x))))
```

$(A \rightarrow B), (C \rightarrow A), C \rightarrow B$

- Meaning of **type variables**
  All places where a given type variable appears must match when you fill in the actual operand types.
- The constraints are:
  - F and G must be functions of one argument
  - the argument type of G matches the type of X
  - the argument type of F matches the result type of G
  - the result type of compose is the result type of F