Structure and Interpretation of Computer Programs

COMP200

QUIZ

1. Write a procedure to compute the area of a triangle.
2. Write a procedure to double a number n times.
3. Write two procedures for multiplication
   one with recursive and one with iterative process.
   a) Specify space and time complexity for each.
   b) (bonus) Can you make it faster?

MULTIPLICATION

Recursive vs. Iterative

```
(define (mult a b)
  (if (= b 1) a
      (+ a (mult a (- b 1)))))
```

- **time:**
- **space:**

```
(define (imult-helper a b ans)
      (if (= b 0)
          ans
          (imult-helper a (- b 1) (+ a ans))))

(define (imult a b)
  (imult-helper a b 0))
```

- **time:**
- **space:**

MULTIPLICATION

Fast Recursive

```
(define (mult a b)
  (if (= b 1) a
      (+ a (mult a (- b 1)))))
```

- **time:**
- **space:**

```
(define (mult-fast a b)
  (if (= b 1) a
      (if (odd? b) (+ a (mult-fast a (- b 1)))
          (mult-fast (+ a a) (/ b 2)))))
```

- **time:**
- **space:**

EXAMPLE

Towers of Hanoi

- Only one disk can be moved at a time.
- Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack, i.e. a disk can only be moved if it is the uppermost disk on a stack.
- No disk may be placed on top of a smaller disk.

EXAMPLE

Towers of Hanoi

- Wishful thinking

  For a tower of 3 disks:

  1. move the top 2 to spare
  2. move the bottom to the destination
  3. move the tower of 2 to the destination from spare.

EXAMPLE

Towers of Hanoi

```
(define (move n from to spare)
  (cond ((= n 0) "done")
        (else
          (move (- n 1) from spare to)
          (print-move from to)
          (move (- n 1) spare to from))))
```

```
(define (print-move from to)
   (display "Move top disk from ")
   (display from)
   (display " to ")
   (display to)
   (newline))
```

```
> (move 3 1 2 3)
Move top disk from 1 to 2
Move top disk from 1 to 3
Move top disk from 2 to 3
Move top disk from 1 to 2
Move top disk from 3 to 1
Move top disk from 3 to 2
Move top disk from 1 to 2
"done"
```

SUMMARY

What have we learned?

- Why do these orders of growth matter?

		2	10	100
Constant	$\Theta(1)$	1	1	1
Logarithmic	$\Theta(\log\ n)$	1	3.33	6.66
Linear	$\Theta(n)$	2	10	100
Quadratic	$\Theta(n^2)$	4	100	10,000
Exponential	$\Theta(2^n)$	4	1024	~1.26 x 10^30

SUMMARY

What have we learned?

- Why do these orders of growth matter?
- Main concern: general order of growth.
  - **Exponential** is very expensive as the problem size grows.
  - Clever thinking can turn an **inefficient** approach to a more **efficient** one.
- Actual performance vs. order of growth.

TODAY

Outline

- Types of objects and procedures
- Procedural abstractions
- Capturing patterns across procedures
  Higher Order Procedures

TYPES

Example

```
> (+ 5 10)
15
```

TYPES

Example

```
> (+ 5 10)
15
> (+ 5 "hi")
```

TYPES

Example

```
> (+ 5 10)
15
> (+ 5 "hi")
+: contract violation
  expected: number?
  given: "hi"
  argument position: 2nd
  other arguments...:
```

TYPES

Simple Data

A taxonomy of expression types:

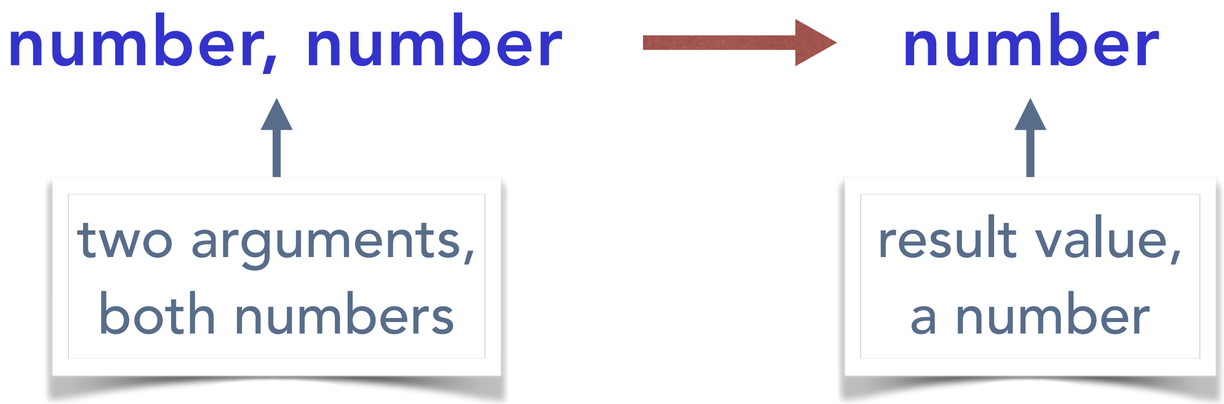
- Simple Data
- Number
  integer, real, rational
- String
- Boolean
- Names (Symbols)

TYPES

Procedures

TYPES

Procedures: +



TYPES

Examples

```
15
"hi"
```

number

TYPES

Examples

```
15
"hi"
square
```

number

string

TYPES

Examples

```
15
"hi"
square
>
```

number

string

number → number

TYPES

Examples

```
15
"hi"
square
>
(> 5 4)
```

number

string

number → number

number, number → boolean

TYPES

Examples

`15`	number
`"hi"`	string
`square`	number → number
`>`	number, number → boolean
`(> 5 4)`	#false

TYPES

Procedures

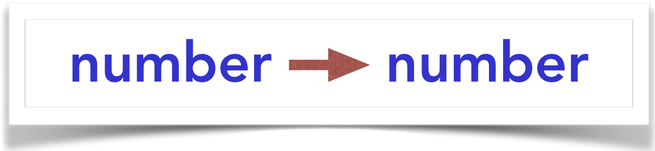
The type of a procedure is a **contract**:

- If the operands have the specified types,
  the procedure will result in a value of the specified type.
- Otherwise, its behavior is undefined.
  maybe an error, maybe random behavior

TYPES

Precisely

- A type describes a set of scheme values



describes the set of all procedures

- whose result is a number,
- which require one argument that must be a number

TYPES

Precisely

- Every scheme value has a type
  - Some values can be described by multiple types
  - If so, choose the type which describes the largest set

TYPES

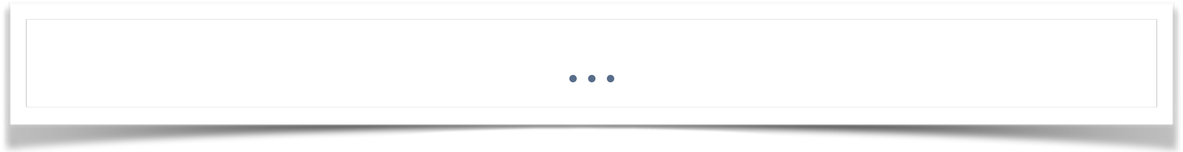
Precisely

- Every scheme value has a type
  - Some values can be described by multiple types
  - If so, choose the type which describes the largest set
- Special form keywords like **`define`** do not name values therefore special form keywords have no type.

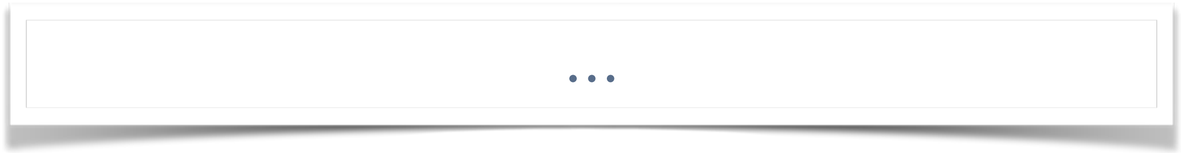
TYPES

Examples

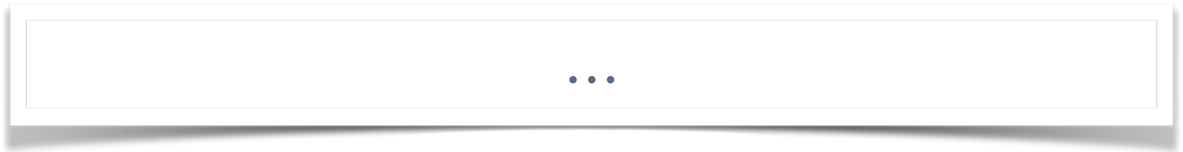
```
(lambda (a b c) (if (> a 0) (+ b c) (- b c)))
```




```
(lambda (p) (if p "hi" "bye"))
```




```
(* 3.14 (* 2 5))
```



TYPES

Examples

```
(lambda (a b c) (if (> a 0) (+ b c) (- b c)))
```

number, number, number → number

```
(lambda (p) (if p "hi" "bye"))
```

boolean → string

```
(* 3.14 (* 2 5))
```

number

SUMMARY

What have we learned?

- **type**: a set of values
  - every value has a type

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

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

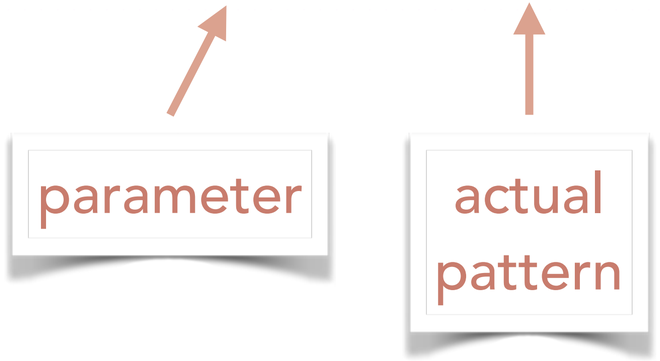
```
(* 2 2)
(* 57 57)
(* k k)
```

REMEMBER

Procedure Abstraction

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

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



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

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

```
(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))))
```

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

PROCEDURE ABSTRACTION

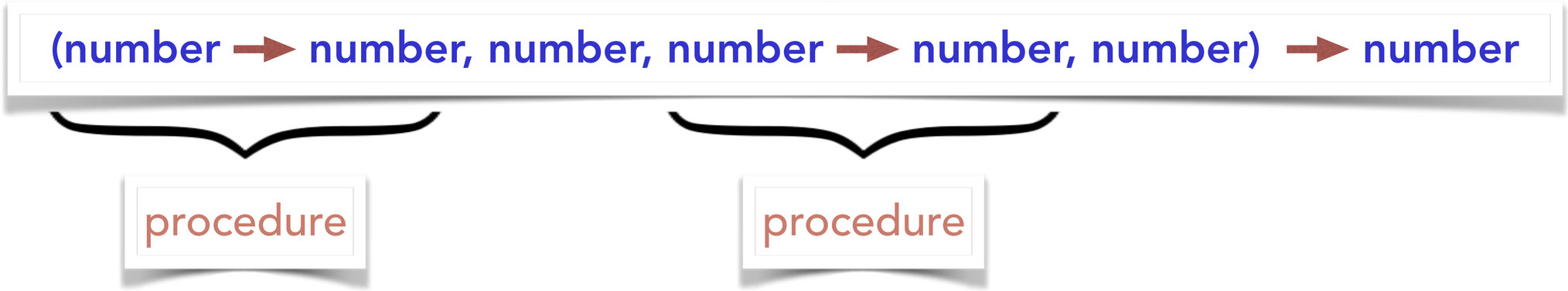
Other Common Patterns

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

PROCEDURE ABSTRACTION

Other Common Patterns

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



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

HIGHER ORDER PROCEDURES

Other Common Patterns

```
(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))
```