

# CS-257L

## Nonimperative Programming: Scheme!

Instructor:

Joel Castellanos

e-mail: [joel@unm.edu](mailto:joel@unm.edu)

Web: <http://cs.unm.edu/~joel/>

Office: Farris Engineering  
Center (FEC) room 321

```
(define prime-factors (lambda (n)
  (if (< n 1)
      "Bad Input: Expected a positive
      whole number."
      (if (not (= n (truncate n)))
          "Bad Input: Expected a positive
          whole number."
          (try n 2 ())))))
```

# Homework – Due Friday – Feb. 8

---

- Read Chapter 3 of The Little Schemer
- Nothing to hand-in.

# Solution to HW-4: prime-factors

---

```
=====
; This function returns a list consisting of the prime
; factors of any positive whole number "n".
; This is a user-interface function, therefore, it needs
; data checking.
=====
(define prime-factors (lambda (n)
  (if (< n 1)
      "Bad Input: Expected a positive whole number."
      (if (not (= n (truncate n)))
          "Bad Input: Expected a positive whole number."
          ; call "try" with 2 as the first factor to try
          ; and an empty list of factors.
          (try n 2 ())
        )
    ) ) )
```

# Lists and Strings

---

In the examples below, in what way do the arguments of display differ?

1. (display '(Bad Input: Expected a positive whole number.))
2. (display 'Bad Input: Expected a positive whole number.)
3. (display "Bad Input: Expected a positive whole number.")

1. List
2. Reference to undefined identifier: input:
3. String  
> (string? "Hello World")  
#t

# prime-factors – with debuting

```
(define prime-factors (lambda (n)
  (display "Enter prime-factors...")
  (newline)
  (if (< n 1) "Bad Input"
      (if (not (= n (truncate n)))
          "Bad Input"
          (begin
             (display "About to call try with n=")
             (display n)
             (newline)
             (try n 2 ()))
          )
      )
  )
  (display "...Done prime-factors") (newline)
) )
```

# Compound Logic

Rewrite this to do both error checks in a single if.

```
(define prime-factors (lambda (n)
  (if (< n 1)
      "Bad Input"
      (if (not (= n (truncate n)))
          "Bad Input"
          (try n 2 ())
      ) ) ) )
```

```
(define prime-factors (lambda (n)
  (if (or (< n 1) (< (truncate n) n))
      "Bad Input"
      (try n 2 ())
  ) ) )
```

# Solution to HW-4: isFactor?

---

```
;=====
; This helper function returns true iff "divisor"
; divides evenly into "n".
; The function assumes two numeric arguments.
;=====
(define isFactor?
  (lambda (n divisor)
    (zero? (remainder n divisor))
  )
)
```

# Solution to HW-4: `try` -header only

---

```
=====
;This helper function checks to see if "testFactor" is
; a factor of "n".
;If it is, then it appends "testFactor" to "primelist"
; and calls itself recursively with "testFactor"
; factored out of "n".
;If "testFactor" is not a factor of "n", then the
; function calls itself recursively with an
; incremented value of "testFactor".
;The recursion ends when "testFactor" becomes larger
; than the current value of "n" - which is the original
; "n" with all the primes inside of "primelist" factored
; out.
=====
(define try
  (lambda (n testFactor primelist)
```

# Solution to HW-4: `try` -w/o comments

---

```
(define try
  (lambda (n testFactor primelist)
    (if (> testFactor (sqrt n)) (cons n primelist)
        (if (isFactor? n testFactor)
            (try (quotient n testFactor)
                testFactor
                (cons testFactor primelist))
            (try n
                (+ testFactor 1)
                primelist)
            )
        )
    ) ) ) )
```

# Solution to HW-4: `try` –with comments

```
(define try
  (lambda (n testFactor primelist)
    ;If this next line is true, then "n" cannot have any more
    ;factors. Thus, append n to list of primes and exit.
    (if (> testFactor (sqrt n)) (cons n primelist)
        ;If this is true, then "testFactor" is a factor of "n".
        ;Thus, append "testFactor" to the list of primes,
        ;Factor "testFactor" out of "n" and call "try" recursively.
        (if (isFactor? n testFactor)
            (try (quotient n testFactor)
                testFactor
                (cons testFactor primelist))
            ;Reached iff "testFactor" is not a factor of "n".
            ;Thus, increment "testFactor" and "try" again.
            (try n (+ testFactor 1) primelist)))
  ) ) )
```

# Solution to HW-4: Unit Test – add to definition file

---

```
(prime-factors 11)      ;(11)
(prime-factors 12)      ;(3 2 2)
(prime-factors 25)      ;(5 5)
(prime-factors 100)     ;(5 5 2 2)
(prime-factors 1024)    ;(2 2 2 2 2 2 2 2 2 2)
(prime-factors 81)      ;(3 3 3 3)
(prime-factors 96769)   ;(96769)
(prime-factors 15485863) ;(15485863)
(prime-factors 275604541) ;(275604541)
(prime-factors 1)       ;(1)
(prime-factors 0)       ;(bad input...
(prime-factors -7)      ;(bad input...
(prime-factors 5.5)     ;(bad input...
```

# Next Slide Quiz

---

- Closed Computers,
- Closed Neighbors,
- Closed Cell Phones
- Open Notes, and books

# Quiz - #2

```
(define Fibonacci
  (lambda (n)
    (display "x ")
    (if (or (< n 1) (< (truncate n) n)) "Bad Input"
        (case n
          ((1 2) 1)
          (else (+ (Fibonacci (- n 1))
                  (Fibonacci (- n 2))
                  )))
    ) ) ) ) )
```

The output of `(Fibonacci 3)` is: `x x x 2`.

What is the output of `(Fibonacci 6)`?

# The Little Schemer: Real Style

---

What does this mean?

1. "Our book is written in the true language of Computer Science."
2. "Our book is a series of Unit Tests."