$\boxed{4}$ Procedure Abstraction

$\boxed{4.1}$ LAMBDA, FUNCALL, and #'

(DEFUN SQUARE (N) (* N N)) computes the function
#'(LAMBDA (N) (* N N)) and associates it
with the name SQUARE

FUNCALL calls a function
(FUNCALL #'(LAMBDA (N) (* N N)) 5) $\Longrightarrow$ 25

#'SQUARE denotes the function associated with SQUARE
by the DEFUN above: (FUNCALL #'SQUARE 3) $\Longrightarrow$ 9

#'x $\equiv$ (FUNCTION x) [equivalent inputs to LISP]

$\boxed{4.2}$ MAPCAR

E.g.:

```
(MAPCAR #'SQUARE '(5 3)) ==> '(25 9)
(MAPCAR #'SQUARE '(5)) ==> '(25)
(MAPCAR #'ODDP '(1 2 3 4 5)) ==> '(T NIL T NIL T)
```

RR:

```
(MAPCAR #'f '()) ==> '()
(MAPCAR #'f '(x . y)) ==> (CONS (FUNCALL #'f 'x)
                                (MAPCAR #'f 'y))
```

f, x, y $\in$ s-expression

$\boxed{4.3}$ SUM, SUM-SQUARE

Examples: `(SUM 1 3)` $\Longrightarrow$ `(+ 1 2 3)` $\Longrightarrow$ `6`

`(SUM-SQUARE 2 5)` $\Longrightarrow$ `(+ 4 9 16 25)` $\Longrightarrow$ `54`

RR: `(SUM m n)` $\Longrightarrow$ `0` if `m > n`

$\Longrightarrow$ `(+ m (SUM (+ m 1) n))` otherwise

`(SUM-SQUARE m n)` $\Longrightarrow$ `0` if `m > n`

$\Longrightarrow$ `(+ (SQUARE m)`

`(SUM-SQUARE (+ m 1) n))`

`m`, `n` $\in$ number otherwise

$\boxed{4.4}$ SUM-TERM

E.g.: `(SUM-TERM #'f m n s)` [s is 'step']

$$\Longrightarrow \quad f(m) + f(m+s) + f(m+2s) + \ldots + f(n)$$

imprecise

`(SUM-SQUARE m n)`

$\Longrightarrow$ `(SUM-TERM #'SQUARE m n 1)`

RR: `(SUM-TERM #'f m n s)`

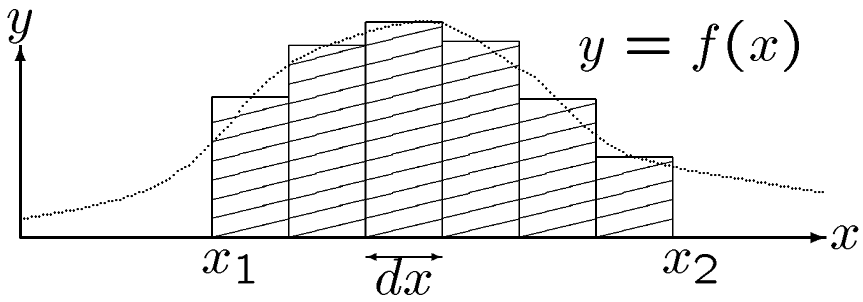
$\Longrightarrow$ `0` if `m > n`

$\Longrightarrow$ `(+ (FUNCALL #'f m)`
`(SUM-TERM #'f (+ m s) n s)` otherwise

`f` $\in$ s-expression `m, n, s` $\in$ number

$\boxed{4.5}$ Approximate Integration, INTEGRAL

Area under the curve
$y = f(x)$
from x_1 to x_2
$= \int_{x_1}^{x_2} f(x)dx$



RR: (INTEGRAL #'f x_1 x_2 dx) $\Longrightarrow$

```
(SUM-TERM #'(LAMBDA (X)
               (* dx (FUNCALL #'f
                        (+ X (* 0.5 dx)))))
          x1 x2 dx)
```

$\boxed{4.6}$ Computing pi, SUM-PI

$$\frac{\pi}{8} = \frac{1}{1 \cdot 3} + \frac{1}{5 \cdot 7} + \frac{1}{9 \cdot 11} + \cdots$$

`(SUM-PI m)` $\Longrightarrow$ [π computed using the first `m` terms]

```
(* 8 (SUM-TERM
        #'(LAMBDA (N)             ; N is number of term.
              (/ 1.0 (* N (+ N 2))))
        1 (* 4 m) 4))
```

$\boxed{4.7}$ Streams: Infinite Lists; Closures

stream := ()
| (first . cdr-function)

RR:
```
(STREAM-CAR '(x . f)) ==> 'x
(STREAM-CAR '()) ==> NIL
(STREAM-CDR '(x . f)) ==> (FUNCALL 'f)
(STREAM-CDR '()) ==> NIL
```

```
(DEFUN INTEGERS-FROM (N)
  (CONS N #'(LAMBDA () (INTEGERS-FROM (+ N 1)))))
```

Comments:

1. A `#'(LAMBDA (...) ...)` expression is called a closure. It consists of a piece of code and a set of variables bound into the closure.

   For example, in the code for `INTEGERS-FROM`, the expression

   ```
   #'(LAMBDA () (INTEGERS-FROM (+ N 1)))
   ```

   constructs a closure that binds the variable `N` from the surrounding `INTEGERS-FROM` function execution into the closure. This variable can then be read or written within the closure.

2. There is another function, `APPLY`, that is like `FUNCALL` except that the last argument to `APPLY` is a list whose elements are taken to be arguments. Thus:

   ```
   (FUNCALL  #'FOO 1 2 3 4 5)
   (APPLY  #'FOO '(1 2 3 4 5))
   (APPLY  #'FOO 1 '(2 3 4 5))
   (APPLY  #'FOO 1 2 '(3 4 5))
   (APPLY  #'FOO 1 2 3 '(4 5))
   (APPLY  #'FOO 1 2 3 4 '(5))
   (APPLY  #'FOO 1 2 3 4 5 '())
   ```

   all call `FOO` with the same arguments.