```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;; MIT 6.001 Spring/2003                                                ;;;;;
;;;;; Rob Miller - Recitation 2: Expressions                               ;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;; Student: Abrantes Araújo Silva Filho                                 ;;;;;
;;;;; Date: 2019-02-10                                                     ;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;; Write down what Scheme will print for each of the following
;;; expressions (evaluated in sequence).
;;; You can use u for unspecified, p for a procedure object, and e for
;;; an error.

(define four 4)
; u

four
; 4

(define six (lambda () 6))
; u. Created the "six" procedure, with no arguments, wich returns 6
; when applied

six
; p. Does not return 6 because the procedure was not applied.

(+ four 1)
; 5

(+ six 1)
; e. We are passing the procedure six (not the value of the procedure)
; to add with 1, and this is an error

(+ (four) 1)
; e. "four" is a name bound to value 4, and is not a procedure that
; could be evaluated do some other value, i. e., is an error to try to
; evaluate the object 4: (four) -> (4)

(+ (six) 1)
; 7

(define f-add
  (lambda (x y)
    (+ (x) (y))))
; u

(f-add six (lambda () four))
; 10. The f-add procedure takes 2 arguments, and uses a combination to
; apply the value of these: (x) and (y). So, as we pass "six", f-add
; evaluate it as (six) -> 6; and as we pass another procedure without
; an explicity name (lambda () four), f-add evaluate it as ((lambda ()
; four)) -> 4. So we can add 6 and 4 -> 10.



;;; Write a procedure circle-area that takes the radius of a circle as
;;; its argument and returns the area of the circle.  Assume pi is
;;; already defined as below.
(define pi 3.14159)
```

```
(define circle-area
  (lambda (radius) (* pi (* radius radius))))

(circle-area 5)
```

```
;;; Write a procedure sign that takes a number as its
;;; argument and returns -1 if it is negative, 1 if
;;; it is positive and 0 if the argument is zero.
(define sign
  (lambda (n)
    (cond ((< n 0) -1)
          ((= n 0)  0)
          ((> n 0)  1))))

(sign -29)
(sign 0)
(sign 34)
```

```
;;; Write some test cases for sign (i.e., expressions that use it)
;;; and show what Scheme would print for each one.
(define add-or-multiply
  (lambda (n x y)
    ((if (< n 0) + *) x y)))

(add-or-multiply -1 2 3)
; 5

(add-or-multiply 100 2 3)
; 6
```