Midterm Exam

1. If a variable is defined within a procedure, only that procedure can access that variable
    a. True
    b. False
2. How do you define a variable in Scheme Racket?
    a. (let variable-name value)
    b. (define variable-name value)
    c. (set! variable-name value)
    d. (var variable-name value)
3. Which of the following creates an empty list in Racket?
    a. '()
    b. (list)
    c. []
    d. ()
4. lambda creates a procedure in the same way as define, except the procedure has no parameters.
    a. True
    b. False
5. The following procedure is a recursive procedure that shows an iterative process.

```
(define (my-function n)
  (if (= n 0)
      1
      (* n (my-function (- n 1)))))

(my-function 3)
```

    a. True
    b. False
6. What is the output of evaluating this code:

```
(define lst '(1 2 3))
(append (cdr lst) '(4 5 6))
```

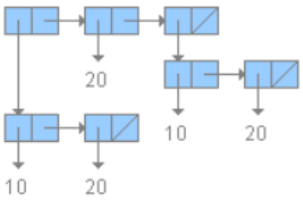
    a. '(1 4 5 6)
    b. '(2 3 4 5 6)
    c. '(1 2 3 4 5 6)
    d. '(4 5 6)
7. What is the output of the following Racket code?

```
(define (my-func x y)
  (let ((a (+ x y))
        (b (- x y)))
    (list a b)))

(my-func 5 3)
```

a. (5 3)
b. ((8 2) (5 3))
c. ‘(2 8)
d. ‘(8 2)

8. The diagram is equivalent to ‘((10 20) 20 (10 20))



a. True
b. False

9. Procedure **list-increment** takes as input a List of numbers, and produces as output a List containing each element in the input List incremented by one. For example, (list-increment 1 2 3) evaluates to (2 3 4).

Choose the correct answer to complete the missing part in list-increment procedure

```
(define (list-increment p)
  (if (null? p)
      null
      ; YOUR CODE GOES HERE
```

a. (cons (* 1 (car p)) (list-increment (cdr p)))))
b. (+ (car p) (list-increment(cdr items)))))
c. (+ 1 (length (cdr items)))))
d. (cons (+ 1 (car p)) (list-increment (cdr p)))))

10. Choose one procedure that is equivalent to the following procedure:

```
(define (sinsq x) (* (sin x) (sin x)))
(sinsq (/ pi 4))
```

a. ((lambda (* (sin x) (sin x))) (/ pi 4))
b. ((lambda (x) (* (sin x) (sin x))) (/ pi 4))
c. (define (* (sin x) (sin x))) (sinsq (/ pi 4))
d. No correct answer

11. Which of the following Racket expressions will apply the procedure (lambda (x) (* x x)) to each element in the list (0 2 3 4) and return a new list containing the results?

a. (filter (* x x) '(0 2 3 4))
b. (foldl (* x x) 1 '(0 2 3 4))
c. (map (* x x) '(0 2 3 4))
d. (map (lambda (x) (* x x)) '(0 2 3 4))

12. Given the following procedures f, g and compose. What is the expected output?

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

(define g
  (lambda (x)
    (+ (* 2 x) 1)))

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

(define fg (compose f g))
(fg 3)
```

    a. Answer = 49

13. Define a recursive procedure named add-up that takes a list of numbers. This procedure returns the sum of the numbers in the list, by means of a recursive process.

    For example,
    > (add-up '(1 2 3)) ; returns 6
    > (add-up '(1)) ; returns 1
    > (add-up '()) ; returns 0

    a.
```
(define (add-up nums)
  (if (null? nums)
      0
      (+ (car nums) (add-up (cdr nums)))))
```

14. In the evaluation of following expressions

```
(define a 5)
(define (hello_world a)
  (cond
    [(= a 3)(display "I")]
    [(and (< a 6) (> a 2))(display "can")]
    [(> a 7)(display "do")]
    [(= a 7)(display "it")]))
```

    the expression (hello_world 6) evaluates to “can”

    a. True
    b. False

15. A higher-order procedure must either A) receive a procedure as an input or B) return a procedure or C) both

    a. True
    b. False

16. How do you create a list in Scheme Racket?

    a. {1 2 3}

b. [1 2 3]
c. (1 2 3)
d. (list 1 2 3)

17. The procedure **power-iter** …

```
(define (power x n)
  (power-iter 1 0 x n ))

(define (power-iter product counter x max-count)
  (if (>= counter max-count)
      product
      (power-iter (* x product) (+ counter 1) x max-count)))
```

a. does not allow tail-call elimination
b. does not do a tail call
c. generates an iterative process
d. No correct answer

18. What is the output of executing the following code?

```
(define (f z)
  (let ((x 4)
        (y (+ z 1)))
    (* x y)))
```

a. 7
b. 20
c. 8
d. 12

19. What is the output of executing the following procedure:

```
(define numbers '(1 2 3 4 5))
(map (lambda (x) (* x 2)) numbers)
```

a. '(2 4 6 8 10)
b. '()
c. '(2 4)
d. '(1 3 5)

20. In Racket, procedures are _____ datatypes
a. Third class
b. First class
c. No correct answer
d. Second class

21. Consider the following procedures below:

```
(define (f a)
  (sum (+ a 1)(* a 2)))

(define (sum x y)
  (+ (square) (square y)))

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

Order the steps below to apply the substitution model illustrating the process generated by this procedure application: **(f 5)**

1. (sum (+ 5 1) (* 5 2))
2. (sum 6 10)
3. (+ (square 6) (square 10))
4. (+ (* 6 6) (* 10 10))
5. (+ 36 100)

22. Which of the following functions can be used to remove the first element from a list in Racket?
    a. cdr
    b. cons
    c. car
    d. append
23. What is the output of the following code:

```
(length '(a b c d))
```

    a. 0
    b. 3
    c. 5
    d. 4
24. Given the following procedures f, g what is the output ?

```
(define (f a b) (+ (g a) b))
(define (g x) (* 3 x))

(f (+ 2 3) (-15 6))
```

    a. Answer = 24
25. What is the result of running the following code in Racket?

```
(define x (cons 1 (cons 2 (cons 3 '()))))

(car x)
```

    a. (cons 1 (cons 2 (cons 3 '())))
    b. (2 3)
    c. (1 2 3)
    d. 1
26. Considering the following expressions, match each expression with its correct choice:

```
(define x 5)
(define (riddle)
  (define double (* 2 x))
  (define (triple x) (* 3 x))
  (+ x (triple double)))
```

    a. An example of Primitive Procedure: +

b. An example of Local Procedure: triple
c. The expression (riddle) evaluates to: 35
d. The expression (double) evaluates to: undefined

27. Write the values that Racket displays after it evaluates the following expression

```
((lambda (a b)
   ((if (< b a)
        +
        *)
    b a))
 4 6)
```

a. Answer = 24

28. Choose the correct definition of procedure **product** that takes a list of numbers and returns the product of the numbers in the list, by means of a **recursive process**.
For example:
> (product '(2 3 4)) ; returns 24
> (product '(3)) ; returns 3
> (product '()) ; returns 1

a.
```
(define (product nums)
  (if (list? nums)
      1
      (* (car nums) (product (cdr nums)))))
```

b.
```
(define (product nums)
  (if (empty? nums)
      0
      (* (car nums) (product (cdr nums)))))
```

c.
```
(define (product nums)
  (if (empty? nums)
      1
      (* (car nums) (product (cdr nums)))))
```

d.
```
(define (product nums)
  (if (empty? nums)
      1
      (* (car nums) (product (car nums)))))
```

29. What is the output of the following procedure?

```
(define (my-function n)
  (if (= n 0)
      0
      (+ n (my-function (- n 1)))))
(my-function 5)
```

a. 15
b. 10
c. 20
d. 5

30. What is the output after executing the following expression?

```
(define (adder x)
  (lambda (y) (+ x y)))

((adder 5) 10)
```

    a. 15
    b. 10
    c. Error message
    d. 5

31. We can use a lambda expression as an argument to a higher-order procedure
    a. True
    b. False

32. In Racket, tail call elimination involves transforming a tail recursive function into an iterative process that doesn't consume additional stack space.
    a. True
    b. False