1. (lambda () 5)

   Ans: #<procedure>

2.((lambda (x) (* 5 2)) 24)

   Ans:10

3.((lambda (op) (op 5 5)) *)

   Ans: 25

4.(define foo
  (let ((x 4))
    (lambda (y) (+ x y))))
  (foo 6)

  Ans: 10

5.(lambda (x) (+ x x))

   Ans: #<procedure>

6.((lambda (x) (+ x x)) 4)

   Ans: 8

(Oprtr args)

 Optr is a 'lambda procedure' with arg '4'

7.(define reverse-subtract
  (lambda (x y)
    (- y x)))
(reverse-subtract 7 10)

Ans: 3

x---7
y---10
10- 7 =3

8. (lambda (x) (* x 2)) Ans: just a procedure

9. ((lambda (y) (* (+ y 2) 8)) 10)

Ans: 96

Y gets value 10… body gets evaluated to 96

10. ((lambda (a) (a 3)) (lambda (z) (* z z)))

Ans: 9

(oprtn args)

((first lambda proc)(second lambda proc))

First lambda proc (in yellow) gets the argument second lambda proc (in green)

((lambda (a) (a 3)) second lambda proc)

second lambda proc is the argument passed as to a

(a 3)

((lambda (z) (* z z)) 3)

Now 3 is passed as argument to z

Which evaluates to 9

11. (

(lambda (b) (* 10 ((lambda (c) (* c b)) b)))

((lambda (e) (+ e 5)) 5)

)

Ans: 1000

(opr args)

Argument is also a lambda proc (e) with arg 5 ➔ which evaluates to (+ e 5) that is 10

So 10 is the argument to the first lambda proc----

((lambda (b) (* 10 ((lambda (c) (* c b)) b)) 10)

b is the argument to the lambda proc(c) ➔ (* b b)

//which is in the body of first lambda proc (b)

((lambda (b) (* 10 (* b b)) 10)

(* 10 (* 10 10))

1000

12. ((lambda (n) (+ n 10))

((lambda (m) (m ((lambda(p) (* p 5)) 7))) (lambda (q) (+ q q)))

Ans: 80

(Oprtn Args)

Green lambda (m) is the argument to pink lambda (n)

Where

Green lambda has argument lambda(q)

Lambda(p) evaluates to 35

((m 35) (lambda (q) (+ q q))

((+ q q)35) ➔ 70

((lambda (n) (+ n 10)) 70)

Which evaluates to 80

13.((lambda(x) (x x)) (lambda(y) 4))

Ans: 4

Lambda can have multiple expressions as body.. last one will be evaluated.

14. ((lambda(y z) (z y)) * (lambda(a) (a 3 5)))

Ans :15

y = * first argument for lambda proc in yellow

z= (lambda(a) (a 3 5))

once the arguments are passed

(lambda(y z) (z y))

(z y)

((lambda(a) (a 3 5)) *) Star * is the argument passed to a

(* 3 5)

15

15. ((lambda (y) 42 (* y 2)) 5)

Ans: 10

// Lambda can have multiple expressions as body.. last one will be evaluated.

((lambda (y) 42 (y + 2) (* y 2)) 5)

Ans 10

```
(define f (lambda (y) 42 (* y 2)))

f   give answer as procedure

(f 5) gives answer as → 10

Trial run variants!!!



16. (let ((x 2) (y 3))
        (let ((foo (lambda (z) (+ x y z)))
         (x 7))
    (foo 4)))

Ans: 9

17.

(let ((x 2) (y 3))
  (* x y))

Ans:6

18.
(define add3
   (lambda (x) (+ x 3)))
(add3 3)
Ans : 6

19.

(let ((x 5))
  (define foo (lambda (y) (bar x y)))
  (define bar (lambda (a b) (+ (* a b) a)))
  (foo (+ x 3)))

Ans: 45

20.

(cond ((> 3 2) 5)
      ((< 3 2) 6))
Ans: 6

21.
(cond ((> 3 3) 2)
      ((< 3 3) 1)
      (else 0))
Ans:0

22.
(and (= 2 2) (> 2 1))
(and (= 2 2) (< 2 1))
(and)

Ans:
#t
#f
#t
```

23.

```
(or (= 2 2) (> 2 1))
(or (= 2 2) (< 2 1))
(or #f #f #f)
```

Ans:

```
#t
#t
#f
```

24.

```
((if #f + *) 3 4)          ➔
(if (> 3 2) 'yes 'no)        ➔
(if (> 2 3) 'yes 'no)        ➔
(if (> 3 2)
    (- 3 2)
    (+ 3 2))              ➔
```

Ans:

```
12
'yes
'no
1
```

25.

```
(define x 2)
(+ x 1)
(+ x 1)
```

Ans:

```
3
3
```

26.

```
(define reciprocal
  (lambda (n)
    (if (= n 0)
        "oops!"
        (/ 1 n))))

(reciprocal 10)
(reciprocal 1/10)
(reciprocal 0)
(reciprocal (reciprocal 1/10))
```

Ans:

```
1/10
10
"oops!"
1/10
```

27.

Convert the following arithmetic expressions into Scheme expressions and evaluate them.

a. 1.2 × (2 - 1/3) + -8.7

b. (2/3 + 4/9) ÷ (5/11 - 4/3)

c. 1 + 1 ÷ (2 + 1 ÷ (1 + 1/2))
d. 1 × -2 × 3 × -4 × 5 × -6 × 7

(Try urself)

28.

```
(let ((x 2))
  (+ x 3))
```
Ans: 5

29.

```
(let ((y 3))
  (+ 2 y))
```
Ans:5

30.

```
(let ((x 2) (y 3))
  (+ x y))
```

Ans:5

31.

```
(let ((a (* 4 4)))
  (+ a a))
```

Ans:32

32.
```
(let ((f +))
  (f 2 3))
```
Ans:5

33.
```
(let ((f +) (x 2))
  (f x 3))
```
Ans: 5

34.
```
(let ((f +) (x 2) (y 3))
  (f x y))
```
Ans: 5

35.

```
(let ((+ *))
  (+ 2 3))
```
Ans: 6
(+ 2 3) Ans: 5
Why????
The variables bound by let are visible only within the body of the let

36.

```
(let ((a 4) (b -3))
```

```
  (let ((a-squared (* a a))
        (b-squared (* b b)))
    (+ a-squared b-squared)))
```

Ans: 25

37.

```
(let ((x 1))
  (let ((x (+ x 1)))
    (+ x x)))
```

Ans: 4
When nested let expressions bind the same variable, only the binding created by the inner let is visible within its body.
This is called "Shadowing"
This could be avoided as

```
(let ((x 1))
  (let ((new-x (+ x 1)))
    (+ new-x new-x)))
```

Ans: 4

38.
Determine the value of the following expression. Explain how you derived this value.

```
(let ((x 9))
  (* x
     (let ((x (/ x 3)))
       (+ x x))))
```

Ans: 54

39.

```
(let ((f (lambda (x) x)))
  (f 5))
```

Ans:5

40.

```
(define abs
  (lambda (n)
    (if (< n 0)
        (- 0 n)
        n)))

(abs 77)

 (abs -77)
```

Ans:
77
77

Different ways for abs.. try everything and see the results!!!!!!!!

Another way of abs as abs1

```
(define abs1
  (lambda (n)
    (if (>= n 0)
        n
```

```
        (- 0 n))))
(abs1 -77)

77
(define abs
  (lambda (n)
    (if (not (< n 0))
        n
        (- 0 n))))

(define abs
  (lambda (n)
    (if (or (> n 0) (= n 0))
        n
        (- 0 n))))

(define abs
  (lambda (n)
    (if (= n 0)
        0
        (if (< n 0)
            (- 0 n)
            n))))

(define abs
  (lambda (n)
    ((if (>= n 0) + -)
     0
     n)))

41.
(if #t 'true 'false)
(if #f 'true 'false)
(if '() 'true 'false)
(if 1 'true 'false)
(if '(a b c) 'true 'false)

Ans:
'true
'false
'true
'true
'true


42.

(not #t)
(not "false")
(not #f)

Ans:
#f
#f
#t

43.
(or)
(or #f)
(or #f #t)
(or #f 'a #f)
```

```
Ans:
#f
#f
#t
'a


44.
(define sign
  (lambda (n)
    (if (< n 0)
        -1
        (if (> n 0)
            +1
            0))))

(sign -88.3)


(sign 0)

(sign 333333333333)

(* (sign -88.3) (abs -88.3))

Ans:
-1
0
1
-88.3

45.
(define sign
  (lambda (n)
    (cond
      ((< n 0) -1)
      ((> n 0) +1)
      (else 0))))

(sign -88.3)


(sign 0)


(sign 333333333333)


(* (sign -88.3) (abs -88.3))
Ans:
-1
0
1
-88.3

46.
(define income-tax
  (lambda (income)
    (cond
      ((<= income 10000) (* income .05))
      ((<= income 20000) (+ (* (- income 10000) .08) 500.00))
```

```
      ((<= income 30000) (+ (* (- income 20000) .13) 1300.00))
      (else (+ (* (- income 30000) .21) 2600.00)))))

(income-tax 5000)

 (income-tax 15000)

 (income-tax 25000)

 (income-tax 50000)
```

Ans:

250.0

900.0

1950.0

6800.0

47.

```
(define goodbye
  (lambda ()
    (goodbye)))

(goodbye)
```

This procedure takes no arguments and simply applies itself immediately. There is no value after the <graphic> because goodbye never returns.