Paradigmas – Introducción a Racket

1 - Representar las siguientes operaciones:

1. **12 + (5-3) \* ((16 /4) + 5)**

(+(\*(+(/ 16 4) 5) (- 5 3)) 12)

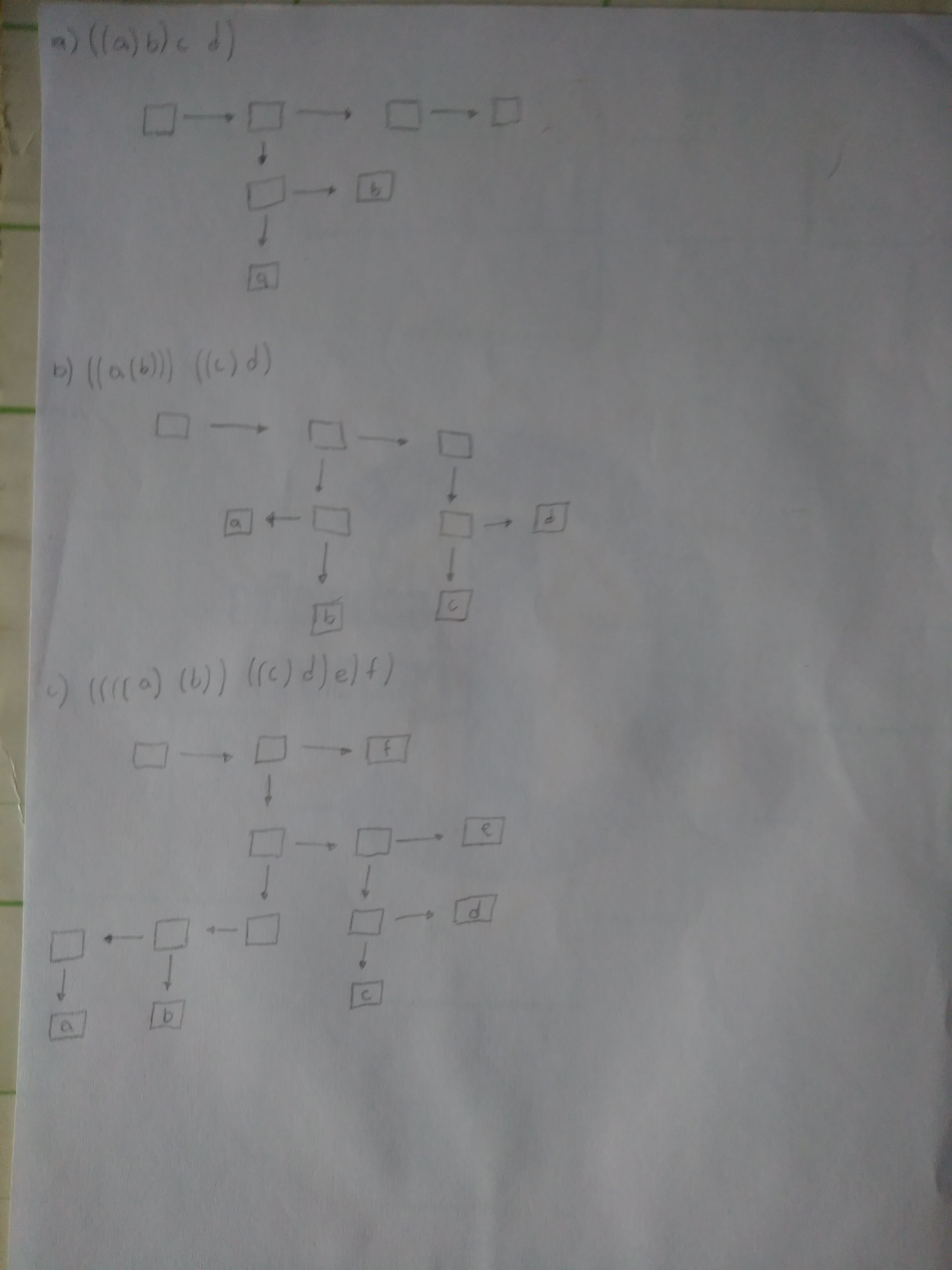
1. **(((11 \* 3) / 10) - 10) + ((5 \* 5) - (3^2)) + 8**

(+ (+ (- (/ (\* 11 3) 10) 10) (- (\* 5 5) (\* 3 3))) 8)

1. **(((5^3)/5)^2) + ((6 / 3) + 8 - 7))**

(+ (expt (/(expt 5 3) 5) 2) (- (+ (/ 6 3) 8) 7) )

2 - Grafique los siguientes casos:



3 - Utilizando if, represente:

1. (if (> 5 3) (printf "5") (printf "3"))
2. (if (= 3 5) (printf "Son iguales") (if (< 3 5) (printf "3 es menor") (printf "5 es menor")))

4 - Definir una secuencia de operaciones CAR/CDR, FIRST/REST y la versión corta CADR para extraer el atomo "pera" de las siguientes listas: (Obs: se deben tener 3 versiones por cada ejercicio).

1. (car (car (cdr (cdr '(manzana naranja pera uva)))))

(first(rest (rest '(manzana naranja pera uva))))

(caaddr '(manzana naranja pera uva))

1. (car(car(cdr '((manzana naranja) (pera uva)))))

(first(first(rest '((manzana naranja) (pera uva)))))

(caadr '((manzana naranja) (pera uva)))

1. (car (car (cdr (cdr (car '(((manzana) (naranja) (pera) (uva))))))))

(first (first (rest (rest (first '(((manzana) (naranja) (pera) (uva))))))))

(car(caddar '(((manzana) (naranja) (pera) (uva)))))

(car (car (cdr (car (cdr '(manzana ((naranja) (pera)) (((uva)))))))))

(first (first (rest (first (rest '(manzana ((naranja) (pera)) (((uva)))))))))

(car(cadadr '(manzana ((naranja) (pera)) (((uva))))))

1. (car (car (cdr (cdr '((((manzana))) ((naranja)) (pera) uva)))))

(first (first (rest (rest '((((manzana))) ((naranja)) (pera) uva)))))

(caaddr '((((manzana))) ((naranja)) (pera) uva))

5 - Defina la función suma que suma dos números pasados como parámetros.

(define (suma a b) (+ a b))

6 - Defina la función cuadrado, que retorna el número pasado elevado al cuadrado.

(define (cuadrado n) (\* n n))

7 - Defina la función área-circulo, que recibe como parámetro el radio y retorna el área del círculo.

(define (area-circulo r) (\* 3.14 (\* r r)))

8 - Defina la función mayor, que recibe 3 número y retorna el mayor de ellos (utilizando cond).

(define (mayor a b c) (cond

[(and (> a b) (> a c)) a]

[(and (> b a) (> b c)) b]

[(and (> c a) (> c b)) c]))