Tecnológico de Monterrey

Implementación de métodos computacionales

Gpo. 641.

Actividad 2.2

Programación funcional, parte 2

Alejandro Pozos Aguirre [A01656017]
Carlos Damián Suárez Bernal [A01656277]
Diego Jacobo Martínez [A01656583]

17/Marzo/2023

Ejercicios 1-6

```
"1. Right (insert 5 '(1 3 6 7 9 16))"'(1 3 5 6 7 9 16)
"1. Wrong (insert \"a\" '(1 3 6 7 9 16))""not a numeric list or number"
" 2. Right (insertion-sort '(4 3 6 8 3 0 9 1 7))"'(0 1 3 3 4 6 7 8 9)
" 2. Wrong (insertion-sort '(4 \"a\" 6 8 3 0 9 1 7))""not a list of numbers"
"3. Right (rotate-left -8 '(a b c d e f g))"'(g a b c d e f)
"3. Wrong (rotate-left \"a\" '(a b c d e f g))""not numeric"
" 4. Right (prime-factors 666)"'(2 3 3 37)
" 4. Wrong (rotate-left \"a\" '(a b c d e f g))""not a number"
"5. Right (gcd 6307 1995)"7
"5. Wrong (gcd 6307 \"a\")""not number(s)"
" 6. Right (deep-reverse '(a (b (c (d (e (f (g (h i j)))))))))"'(((((((j i h) g) f) e) d) c) b) a)
" 6. Wrong (deep-reverse 12)""not a list"
```

Ejercicio 7

```
712.rkt> (insert-everywhere 1 '(a b c))
'((1 a b c) (a 1 b c) (a b 1 c) (a b c 1))
712.rkt> (insert-everywhere 2 2)
"No es una lista"
712.rkt>
```

Ejercicio 8

```
712.rkt> (pack '(a a a a b c c a a d e e e e))
'((a a a a) (b) (c c) (a a) (d) (e e e e))
712.rkt> (pack 25)
"No es una lista"
712.rkt>
```

Ejercicio 9

```
712.rkt> (compress '(a a a a b c c a a d e e e e))
'(a b c a d e)
712.rkt> (compress 4)
"No es una lista"
712.rkt>
```

Ejercicio 10

```
712.rkt> (encode 4)
"No es una lista"
712.rkt> (encode '(a a a a b c c a a d e e e e))
'((4 a) (1 b) (2 c) (2 a) (1 d) (4 e))
```

Ejercicio 11

```
712.rkt> (encode-modified '(a a a a b c c a a d e e e e))
'((4 a) b (2 c) (2 a) d (4 e))
712.rkt> (encode-modified 45)
"No es una lista"
```

Ejercicio 12

```
712.rkt> (decode '((4 a) b (2 c) (2 a) d (4 e)))
'(a a a a b c c a a d e e e e)
712.rkt> (decode '())
'()
```

Ejercicio 13

```
> ((args-swap list) 1 2)
'(2 1)
> ((args-swap /) 8 2)
1/4
> ((args-swap cons) '(1 2 3) '(4 5 6))
'((4 5 6) 1 2 3)
> ((args-swap map) '(-1 1 2 5 10) /)
'(-1 1 1/2 1/5 1/10)
> 
```

Ejercicio 14

```
> (there-exists-one? positive? '())
#f
> (there-exists-one? positive? '(-1 -10 4 -5 -2 -1))
#t
> (there-exists-one? negative? '(-1))
#t
> (there-exists-one? symbol? '(4 8 15 16 23 42))
#f
> (there-exists-one? symbol? '(4 8 15 sixteen 23 42))
#t
> 
```

```
> (there-exists-one? number? "lst")
"lst no es una lista"
```

Ejercicio 15

```
> (linear-search '() 5 =)
#f
> (linear-search '(48 77 30 31 5 20 91 92 69 97 28 32 17 18 96) 5 =)
4
> (linear-search '("red" "blue" "green" "black" "white") "black" string=?)
3
> (linear-search '(a b c d e f g h) 'h equal?)
7
>
```

```
> (linear-search '(a b c d e f g h) 'h "equal?")
"eq-fun no es una función"
```

Ejercicio 16

```
> (define f (lambda (x) (* x x x)))
(define df (deriv f 0.001))
(define ddf (deriv df 0.001))
(define dddf (deriv ddf 0.001))
> (df 5)
75.01500100002545
> (ddf 5)
30.006000002913424
> (dddf 5)
5.999993391014868
>
```

```
> (deriv "f" 0.001)
"f no es una funcion"
```

Ejercicio 17

```
> (newton (lambda (x) (- x 10)) 1)
10.000000000023306
> (newton (lambda (x) (+ (* 4 x) 2)) 1)
-0.5000000000000551
> (newton (lambda (x) (+ (* x x x) 1)) 50)
-0.9999999980685114
> (newton (lambda (x) (+ (cos x) (* 0.5 x))) 5)
-1.029866529322135
>
```

```
> (newton "string" 12)
"f no es una funcion"
```

Ejercicio 18

```
;-------------------------------------------------------------------------------------------------------------
#|
18. La función integral toma a y b como limites inferior y superior, n como la cantidad de veces
que se ejecutara el método de Simpson y f como una función dependiente de x. Devuelve el resultado de la
integral definida, usando la regla de Simpson.
|#
;;Función deriv
(define (integral a b n f)
  (if (procedure? f)
      (if (number? a)
          (if (number? b)
              (if (> b a)
                  (if (integer? n)
                      (if (> n 0)
                          (integralSumaRecursive 0 n a (/ (- b a) n) f 0) ;;Inicializa Recursión
                       "n no es menor a 0")
                   "n no es un numero entero")
               "b no es mayor que a")
           "b no es un numero")
       "a no es un numero")
   "f no es una funcion")
)
;;Función integralSumaRecursive que resuelve Simpson de manera Recursiva
(define (integralSumaRecursive i n a h f total)
  (if (= i (+ n 1))
      (* total (/ h 3))
      (if (or (= i 0) (= i n))
          (integralSumaRecursive (+ i 1) n a h f (+ total (f (+ a (* i h)))))
          (if (odd? i)
              (integralSumaRecursive (+ i 1) n a h f (+ total (* 4 (f (+ a (* i h))))))
              (integralSumaRecursive (+ i 1) n a h f (+ total (* 2 (f (+ a (* i h))))))
          ))))
```

```
> (integral 0 1 10 (lambda (x) (* x x x)))
1/4
```

```
> (integral 1 2 10
(lambda (x)
(integral 3 4 10
(lambda (y)
(* x y)))))
5 1/4
```

```
> (integral 0 1 10 "string")
"f no es una funcion"
```