Taller Programación Funcional

Objetivo

- Que el estudiante aprenda a utilizar los elementos básicos de Scheme para procesar listas y usar funciones de orden superior

Reglas

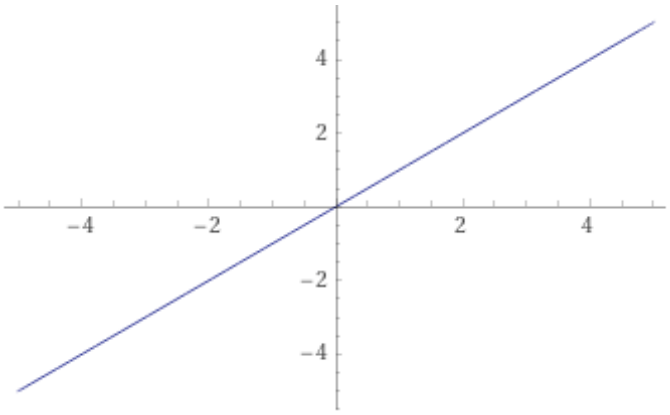
- El taller es en parejas
- Está permitido:
  - Utilizar el siguiente material de apoyo: diapositivas, libros, apuntes y la información que está publicada en Uvirtual.
  - También está permitido buscar información utilizando buscadores en Internet.
- Está prohibido:
  - Copiar total o parcialmente alguna solución encontrada en cualquier fuente.

Enunciado

Con el taller se busca crear funciones de orden superior que permitan evaluar otras funciones arbitrarias y determinar si son funciones monótonas crecientes o decrecientes, mediante algunas muestras de ejemplo.

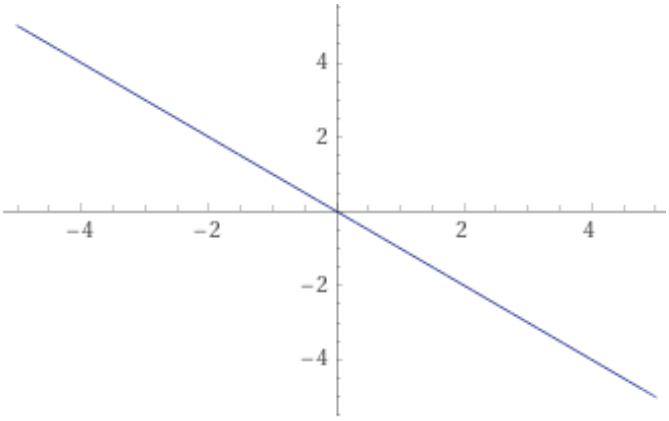
Ejemplo:

Suponga la función identidad f(x)=x



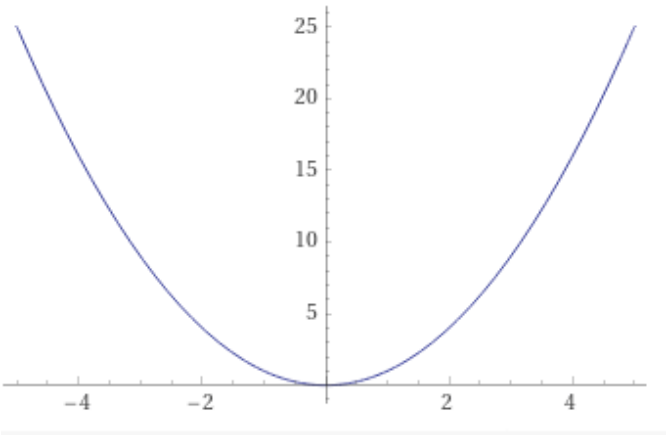
La cual es una función monótona creciente, ya que en cualquier punto si el valor x aumenta, el valor de f(x) aumenta.

Otra función monótona es f(x)=-x=(-1)x



En este caso la función es monótona decreciente.

Un caso de una función que no es monótona es f(x)=x2, ya que si x es menor que 0 es decreciente y si x>0 es creciente



1. Cree las funciones en el archivo base que le permitirán probar sus funciones y le facilitarán algunas partes
2. [10%] Cree una función llamada *evaluar-puntos* cuyos argumentos son una función y una lista y retorna una lista de listas en la cual cada elemento es una pareja (lista de 2 elementos) con el valor de x y de f(x).

   Ejemplos

```
 : (evaluar-puntos id (list -5 -1 0 10 15))
=> ((-5 -5) (-1 -1) (0 0) (10 10) (15 15))
 : (evaluar-puntos neg (list -5 -1 0 10 15))
=> ((-5 5) (-1 1) (0 0) (10 -10) (15 -15))
 : (evaluar-puntos cuadrado (list -5 -1 0 10 15))
=> ((-5 25) (-1 1) (0 0) (10 100) (15 225))
 : (evaluar-puntos cubo (list -5 -1 0 10 15))
=> ((-5 -125) (-1 -1) (0 0) (10 1000) (15 3375))
 : (evaluar-puntos (lambda (x) (sin x)) (list -5 -1 0 10 15))
=> ((-5 0.9589242746631385) (-1 -0.8414709848078965) (0 0) (10
-0.5440211108893699) (15 0.6502878401571169))
```

3. [20%] Cree una función *es-creciente-en-punto?*, la cual recibe una función y un número *x* y

retorna verdadero si la función es creciente en el punto *x*. Para esto compare los valores de f(x), f(x-1) y f(x+1). Si f(x-1)<=f(x)<=f(x+1), la función es creciente en el punto x.

Ejemplos:

```
> (es-creciente-en-punto? id 0)
=> #t
> (es-creciente-en-punto? id 1)
=> #t
> (es-creciente-en-punto? neg 1)
=> #f
> (es-creciente-en-punto? cuadrado -5)
=> #f
> (es-creciente-en-punto? cuadrado 0)
=> #f
> (es-creciente-en-punto? cuadrado 1)
=> #t
```

4. [20%] Cree una función *es-decreciente-en-punto?* Análoga a la anterior. Es decir verdadero si f(x-1)>=f(x)>=f(x+1).

   Ejemplos:

```
> (es-decreciente-en-punto? cuadrado 1)
=> #f
> (es-decreciente-en-punto? cuadrado 5)
=> #f
> (es-decreciente-en-punto? cuadrado 0)
=> #f
> (es-decreciente-en-punto? cuadrado -5)
=> #t
```

5. [20%] Cree una función *es-creciente-en-muestra?*, la cual tiene como argumento una función f y una lista de valores, y retorna verdadero si f(x) es creciente en TODOS los valores de la lista.

```
> (es-creciente-en-muestra? id (list -5 -3 0 8))
=> #t
> (es-creciente-en-muestra? neg (list -5 -3 0 8))
=> #f
> (es-creciente-en-muestra? cuadrado (list -5 -3 0 8))
=> #f
> (es-creciente-en-muestra? cuadrado (list 1 4 6))
=> #t
```

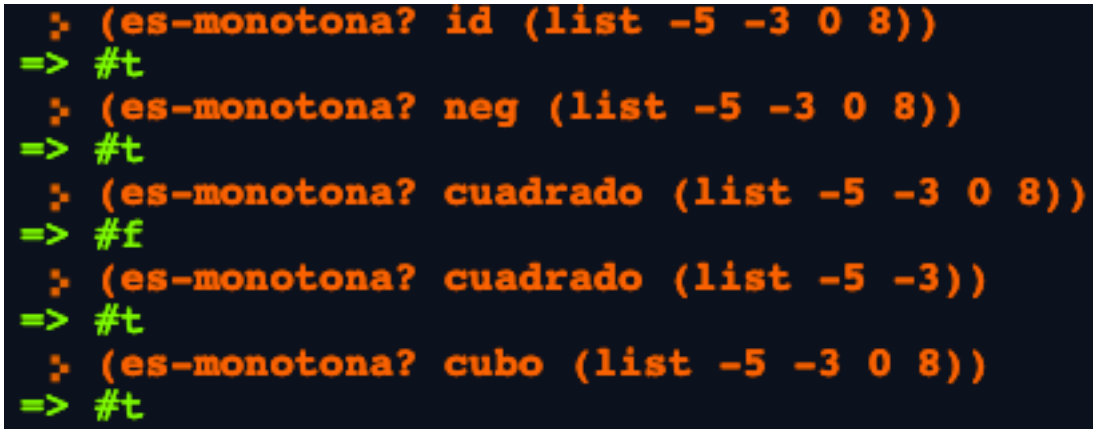
6. [20%] Cree una función *es-decreciente-en-muestra?*, la cual tiene como argumento una función f y una lista de valores, y retorna verdadero si f(x) es decreciente en TODOS los valores de la lista.

   Ejemplo:

```
> (es-decreciente-en-muestra? id (list -5 -3 0 8))
=> #f
> (es-decreciente-en-muestra? neg (list -5 -3 0 8))
=> #t
> (es-decreciente-en-muestra? cuadrado (list -5 -3 0 8))
=> #f
> (es-decreciente-en-muestra? cuadrado (list -5 -3 -1))
=> #t
```

7. [10%] Cree una función *es-monotona?,* la cual tiene como argumento una función f y una lista de valores, y retorna verdadero si f(x) es creciente o decreciente en TODOS los valores de la lista

   Ejemplos:



Nota: La función que reciben todas las funciones puede ser cualquier función con un solo argumento numérico que retorna un número, no solo las que están de ejemplo

Entregables

Subir a la actividad de Uvirtual un archivo con extensión .scm con el código fuente de la solución al problema.

Evaluación

El archivo entregado debe cumplir con lo siguiente:

1. El archivo completo debe ser interpretado por BiwaScheme sin errores. Este intérprete se puede acceder en https://repl.it/languages/scheme.
2. Utilizar la función `define` para asignarle un nombre a cada una de las funciones que Ud. incluya en el código fuente. Asegúrese que los nombres de las funciones sean idénticos a los que se solicitan en este enunciado y que reciban los tipos de parámetros exactos que se solicitan.
3. Las funciones definidas no deben depender de datos externos a la función, pero sí pueden utilizar otras funciones previamente definidas. Se permite crear cualquier función auxiliar adicional a las solicitadas.
4. Para que una función sea evaluada con nota distinta de cero, debe ejecutarse sin errores y cumplir con lo que se solicita en el enunciado. Esto quiere decir que todas las funciones de las cuales dependa también deben ejecutarse correctamente.

El incumplimiento de 1 puede significar una reducción en la nota.

Cualquier función que incumpla 2, 3 y/o 4 será evaluada con cero (0.0)