Práctica 0

Semestre 2025-1

Lenguajes de Programación

Profesor: L. en C.C. Enrique Francisco Soto Astorga
Ayudante: Lázaro Eduardo Rodriguez Belmonte
Ayudante: José Manuel Madrigal Ramirez

Fecha de entrega: 30 de agosto

1. Instrucciones

Eres libre de utilizar cualquier IDE o cualquier editor de texto. Está prohibido utilizar funciones primitivas del lenguaje que resuelvan directamente los ejercicióon. **La entrega de esta práctica es en parejas**. Dentro de la documentación de cada función proporcionar una explicación del problema, describir el funcionamiento y el proceso mental para resolverlo.

2. Entregables

Deberás enviar mediante la plataforma **Google Classroom** el link a tu repositorio de github classroom.

3. Ejemplo

Cada ejercicio de tu práctica deberia lucir de la siguiente forma:

- Un predicado que reciba un número natural n, que devuelva cerdadero si n es par y falso en otro caso.

```
(defn espar? [n]
    "Toma un entero n y devuelve si es par o no. Podemos verificar si un entero es
       positivo con pos? y luego usar rem? para obtener el residuo con 2 y decidir si es
        par.

    Params:
    - n: Un entero.

    Returns:
    - Si n es par."
  (and (pos? n) (= 0 (rem n 2))))
```

4. Ejercicios

Los siguientes son ejercicios clásicos de programación y no servirán para empezar a trabajar con Clojure. Resuelvelos teniendo en cuenta que serán revisados con pruebas unitaras, así que **respeta el nombre especificado en las intrucciones**.

1. **(3 puntos)** Programa una función **get-two** que dada una lista de números enteros l y un entero objetivo o, regresa una lista de dos índices de l tal que la suma de los elements en esos índices es igual a o. Los indices de la lista resultante deben ser diferentes, es decir, si hacemos:

   ```
   > (get-two '(3 2) 6)
   ```

   '$(0, 0)$ no puede ser una respuesta valida.

   Ejemplos:

   ```
   > (get-two '(1 0 7 4 2) 8)
   > '(0 2)
   > (get-two '(9 9) 18)
   > '(0 1)
   ```

   Hint:

   Usa **diccionarios** para resolverlo.

   ```
   (hash-map) ; Crea un diccionario.
   (contains? t k) ; Regresa si en el diccionario t hay un elemento con llave k.
   (get t k) ; Regresa el valor de la llave k.
   (assoc t k v) ; Agrega en la tabla t la llave con k con valor v.
   ```

2. (2 puntos) Programa una función **fizz-buzz** que dado un entero n regresa una lista de tamaño n donde en el indice i si $i+1$ es divisible entre 3 estará almacenado F, si $i+1$ es divisible entre 5 estará almacenado B, si es divisible entre 3 y 5 estará almacenado FB y en otro caso estará almacenado $i+1$.

   Ejemplos:

   ```
   > (fizz-buzz 5)
   > '(1 2 "F" 4 "B")
   > fizz-buzz 15)
   > '(1 2 "F" 4 "B" "F" 7 8 "F" "B" 11 "F" 13 14 "FB")
   ```

3. **(2 puntos) La conjetura de Collatz plantea lo siguiente:**

   Sea n un número entero positivo, si se le aplican los dos siguientes pasos repetidamente, eventualmente llegará a 1:

   Si n es par divídelo entre dos, en otro caso multiplicalo por 3 y sumale 1.

   Define una función **lista-c** que recibe un número entero positivo y regresa una lista con todos los números por los que pasa n antes de llegar a 1.

   Ejemplos:

   ```
   > (lista-c 5)
   '(5 16 8 4 2 1)
   ```

4. **(2 puntos)** Define una función **suma** que recibe dos números enteros y regresa la suma de estos. **Para este ejercicio no puedes usar ninguna de las siguientes funciones +, -, (inc), (dec), (range), (reduce), (eval), (map).**

5. **(1 punto)** Define el predicado **pares?** que dada una lista de números, regresa si todos los elementos de esta son pares.

   ```
   > (pares? '(2 4 6 8 10))
   true
   > (pares? '(1 2 3 4))
   false
   ```

5. Extras

Realiza las siguientes funciones y obten hasta 2.5 puntos en la práctica.

6. **1 punto**

```
(my-map f l)
f: funcion
l: lista
return: l despues de aplicar a cada elemento f
```

7. **1 punto**

```
(my-foldr f i l)
f: funcion (f a b) -> b
l: lista de tipo a
i: de tipo b
return: Aplicar a cada elemento de l f con i y acumular el resultado en i.
```

8. **0.5 puntos**

```
(my-filter p l)
p: predicado
l: lista
return: Todos los elementos de l que cumplen p
```