Parcial

1.
    a. Convertir la siguiente expresión en una **función Lisp**:

$$\left(1 - \frac{1}{2} \cdot m^{-5}\right) \cdot \left(\frac{1 + \frac{i}{m}}{1 - \frac{i}{m}}\right)^{mt} \cdot K$$

    b. A partir de la expresión obtenida en el punto a, determinar la **cantidad total de átomos y listas** que contiene la función, y detallarlos explícitamente.

    c. A partir de la expresión obtenida en el punto a, extraer el átomo `t` utilizando las funciones que usted crea conveniente.

2. Se desea diseñar un sistema que gestione compras para una obra, donde se necesita adquirir **arena, piedra y hierro** según un monto determinado de dinero.

    a. Desarrollar una función que permita el ingreso por parte del operador de:

    - El valor por m³ de arena
    - El valor por m³ de piedra
    - El valor por kg de hierro
    - El monto disponible

    Esta función deberá invocar internamente a las funciones de los puntos b, c, d **y también**:

    - Validar que ningún valor ingresado sea negativo o nulo.
    - Mostrar un mensaje distinto si el dinero es menor a todos los valores unitarios.

    b. Definir una función que, a partir del dinero disponible y del valor del m³ de **arena**, devuelva:

    - `"monto insuficiente"`: si no alcanza para comprar ni 1 m³.
    - `"monto justo para 2m³"`: si el monto alcanza justo para comprar 2 m³.

- `"monto suficiente para más de 2m³"`: si el monto supera el valor de 2 m³.

c. Definir una función **predicado** que verifique si con **la mitad del dinero disponible** se puede comprar **al menos**:

- 4 m³ de arena
- 1.5 m³ de piedra
- 10 kg de hierro

Todos los valores deben ser ingresados como parámetros.

d. Definir una función `resumen-compra-materiales` que:

- Tome como parámetros el dinero disponible, y los valores de arena, piedra y hierro.
- Devuelva una **lista de tres sublistas**, cada una con:
  1. El nombre del material (como string)
  2. El monto total disponible
  3. El precio unitario
  4. La cantidad que se puede comprar con ese dinero
  5. Un mensaje adicional, por ejemplo:
     - `"compra mínima cubierta"` si se puede comprar al menos 1 unidad
     - `"insuficiente"` si no alcanza ni para 1 unidad
     - `"excedente de compra"` si se puede comprar más de 10 unidades

3.

a. Analice las siguientes expresiones Lisp. Indique qué valor devuelve cada una, y explique **paso a paso cómo llega al resultado**.

Debe identificar claramente si cada `cond` es verdadero o falso, qué cláusula se ejecuta y qué devuelve.

```
(defun evalua1 (X Y)
  (cond
    ((and (listp X) (not (numberp Y))) (append X Y))
    ((and (numberp X) (symbolp Y)) (list X Y))
    ((or (and (integerp X) (not (consp Y)))
         (and (listp Y) (listp X)))
```

```
     (cons X (reverse Y)))
    (t 'caso-default)))
```

Evalúe las siguientes expresiones:

`(evalua1 '(4 2) '(1 3 5))`

`(evalua1 3 'Z)`

`(evalua1 '(a b) 'c)`

`(evalua1 9 '(a b c))`

b. Dada la siguiente función:

```
(defun analiza (X Y Z)
  (cond
    ((and (listp X) (listp Y) (symbolp Z)) (list Z X Y))
    ((or (null X) (numberp Y)) (cons Y X))
    ((and (stringp Z) (not (listp Y))) (list (length Z)))
    ((not (equal X Y)) (append Y Z))
    (t 'nada)
  )
)
```

Evalúe las siguientes expresiones:

`(analiza '(1 2) '(3 4) 'R)`

`(analiza nil 5 "hola")`

`(analiza '(1 2) '(1 2) '(3 4))`

`(analiza 2 '(3 4) 'Z)`

c. Dada la siguiente función:

```
(defun calculo (A B C)
    (if (> (+ (* A 2) (- C B)) 0)
        (list 'positivo (+ (* A 2) (- C B)))
        (list 'negativo (+ (* A 2) (- C B)))
    )
)
```

Evalúe las siguientes llamadas:

`(calculo 3 2 10)`

`(calculo 1 8 5)`

`(calculo 0 0 0)`

- Modifique la función anterior para que, en lugar de usar `if`, use `cond`, y agregue una cláusula adicional que devuelva `'cero` si el resultado de la operación es 0.