**Guía 01 Paradigmas de Programación**

1. ¿Cuál de las siguientes expresiones no es una lista válida en Haskell?

**R: b) [ (1,2), (3,4,5) ]**

1. Es una función que tiene variables de tipo en su definición.

**R: Función polimórfica**

1. Es una función que se llama a sí misma en su definición.

**R: Función recursiva**

1. Es la función que toma una o más funciones como entrada y/o devuelven una función como salida.

**R: Función de orden superior**

1. Es una función o subrutina definida que no está enlazada a un identificador.

**R: Expresiones Lambda o función Lambda**

1. En Haskell una cadena es \_\_\_\_\_\_\_\_\_\_\_ .

**R: Una lista de caracteres**

1. En Haskell, ¿a cuál de las listas de abajo es igual [2, 4..20]?

**R: [2, 4, 6, 8, 10, 12, 14, 16, 18, 20]**

1. La función head’ definida abajo:

head’ :: [a] -> a

head’ [ ] = error “lista vacía!”

head’ (x : \_) = x

es: \_\_\_\_\_\_\_\_\_\_\_\_\_\_ .

**R: Polimórfica**

1. En Haskell dado el patrón (x:xs) la xs concuerda con:

**R:La cola de la lista**

1. En Haskell, de los patrones de abajo ¿cuál concuerda con listas de 3 elementos?

**R: (x : y : z : [ ])**

1. En Haskell el resultado de [x\*2 | x <- [1..10], x\*2 >= 14] es:

**R: [14, 16, 18, 20]**

1. ¿Cuál es el resultado de zipWith’ max [6, 3, 2, 1] [7, 3, 1, 5] ?

**R: [7, 3, 2, 5]**

1. ¿Qué hace la siguiente función?

f :: [a] -> [a]

f [ ] = [ ]

f (x : xs) = x:f xs

**R: Devuelve una lista que es una copia de la lista que recibe como parámetro**

1. ¿Cuál de las siguientes expresiones es igual a [1, 9, 25]?

**R: map (^2) ( filter (not even) [1..6] )**

1. ¿Cuál de las siguientes expresiones es igual a [1, 2, 3, 4 ]?

**R: d) take 4 (iterate (+1) 1)**

1. El tipo de la función uncurry f (x, y) = f x y es:

**R: a) (a->b->c) -> (a, b) -> c**

**Problemas**

1. Codificar una función que reciba un número y cuyo tipo de retorno sea una cadena que diga si el número es:

* Menor que cero
* Igual que cero
* Mayor que cero

module Test where

comp :: Int -> String

comp x

  | x < 0 = "Menor que cero"

  | x == 0 = "Igual a cero"

  | x > 0 = "Mayor que cero"

1. Codificar una función que reciba 2 números y cuyo tipo de retorno sea una cadena. Que la función regrese una cadena que diga si el 1er número es:

* Menor que el 2do número
* Igual que el 2do número
* Mayor que el 2do número

module Test where

comp :: (Int,Int) -> String

comp (x,y)

  | x < y = "Menor que el segundo"

  | x == y = "Igual al segundo"

  | x > y = "Mayor que el segundo"

1. Codificar una función que reciba 2 números y cuyo tipo de retorno sea una cadena. Que la función regrese una cadena que diga si los 2 números son coordenadas que están:

* En el 1er cuadrante x positivo y positivo
* En el 2do cuadrante x negativo y positivo
* En el 3er cuadrante x negativo y negativo
* En el 4to cuadrante x positivo y negativo

module Test where

coord :: (Int, Int) -> String

coord (x, y)

  | x > 0 && y > 0 = "Primer cuadrante"

  | x < 0 && y > 0 = "Segundo cuadrante"

  | x < 0 && y < 0 = "Tercer cuadrante"

  | x > 0 && y < 0 = "Cuarto cuadrante"

1. Codificar una función que reciba 3 números y cuyo tipo de retorno sea una cadena. Los 3 números son las longitudes de los lados de un triángulo. Que la función regrese una cadena que diga si el triángulo es equilátero o no lo es.

module Test where

triangulo :: (Int, Int, Int) -> String

triangulo (x, y, z)

  | x == y && y == z && x == z = "Es equilatero"

  | otherwise = "No es equilatero"

1. Codificar una función que reciba 3 números y cuyo tipo de retorno sea una cadena. Los 3 números son las longitudes de los lados de un triángulo. Que la función regrese una cadena que diga si el triángulo es isósceles o no lo es.

module Test where

triangulo :: (Int, Int, Int) -> String

triangulo (a, b, c)

  | a == b || b == c || a == c = "Isosceles"

  | otherwise = "No es equilatero"

**Tuplas**

1. Escriba una función “repeDupla” que reciba un valor y devuelva un par (tupla) con ese valor repetido.
2. Codificar una función que reciba 2 tuplas de 3 y cuyo tipo de retorno sea un número. Se debe calcular el producto escalar de 2 tuplas de 3.
3. Escriba una función recursiva que obtenga todos los primeros elementos de una lista de tuplas.

**Listas**

1. Codificar una función que reciba 2 listas y cuyo tipo de retorno sea un número. Debe calcular el producto escalar de 2 listas usando recursividad.
2. Codificar una función recursiva que reciba un valor y devuelva una lista infinita que solo tiene el cuadrado de ese valor.
3. Escriba una función recursiva “listaHasta::Int->[Int]” que crea una lista desde 1 hasta el número pasado. Por ejemplo, listaHasta 3 es [1, 2, 3].
4. Escriba una función recursiva “cuenta x xs” que devuelve el número de veces que x ocurre en la lista xs.
5. Escriba una función recursiva “PrimeraPosicion x xs” que devuelve la primera posición en la que aparece x en la lista xs.

**Lambdas**

1. Escriba una función lambda que multiplique un número por 2.
2. Escriba una función lambda que eleve un número al cuadrado.
3. Escriba una función lambda que sume dos números.
4. Escriba una función lambda que calcule el promedio de dos números.