Ejercicio 2

Utilice *ghci* para decidir si las expresiones *(2^29)/(2^9) y 2^20* son iguales. Recuerde que el operador de potenciación en *ghci* es infijo y se escribe “**^**”

Respuesta

En el comando de *ghci* al escribir 2^29/2^9 == 2^20 me retorna un valor lógico **True.**

Ejercicio 3

Utilizando las funciones ***head*** y ***tail***, y dada la lista “hola mundo”, obtenga el segundo elemento de la misma (la letra “o”).

Respuesta

head(tail *“hola mundo”**)***, retorna “**o**”.

Ejercicio 4

Utilizando las funciones ***head*** y ***reverse***, y dada la lista “hola mundo”, obtenga el último elemento de la misma (la letra “o”).

Respuesta

head(reverse *“hola mundo”**)***, retorna “**o**”**.**

Ejercicio 5

Utilizando la función realizada en el ejercicio anterior y la función *mod* determine si un número, representado como la lista de sus dígitos (ej: 123 = [1,2,3]) es par

Respuesta

Para determinar que un número sea par, solamente nos tenemos que fijar que en el último dígito sea par, y más dada una lista, podemos aprovechar el uso del ***head*** y ***reverse,*** dicha función está declarada de la siguiente forma:

```
esNoPar :: [Int] -> Bool
esNoPar xs = head(reverse xs) `mod` 2 == 0
```

Ejercicio 6

Utilizando la función ***sum*** la función ***mod*** y un número representado de igual manera que en el ítem 5, determine si un número es múltiplo de 3.

Respuesta

```
mulTres :: [Int] -> Bool
mulTres xs = sum xs `mod` 3 == 0
```

Ejercicio 7

Utilizando las funciones de los ítem 5, 6, determine si un número es múltiplo de 6.

Respuesta

Utilicé solamente la función del ejercicio 6 y creé un adicional para el caso de verificar que sea múltiplo de 2:

```
multDos :: [Int] -> Bool
multDos xs = sum xs `mod` 2 == 0    --adicional para ejercicio 7

multSeis :: [Int] -> Bool
multSeis xs = multDos xs && mulTres xs
```

Ejercicio 8

Escriba una función que dado un número retorne la lista de sus dígitos.

Respuesta

Pensé de manera recursiva, como caso base que si tengo un n menor a 10, entonces me devuelve esa lista, en el caso inductivo realiza una recursión de cola, donde desmenuza todos los números hasta llegar al caso base, una vez realizado se concatena con el **n `mod` 10** que es el ultimo dígito:

```
numLista :: Int -> [Int]
numLista n
    | n < 10 = [n]  --en el caso de que de un numero menor que 10,
                    --directamente lo transforme en lista
    | otherwise = numLista (n `div` 10) ++ [n `mod` 10]
```

Ejercicio 9

Investigue las definiciones de las funciones **take** y **drop**, utilizando estas funciones, implemente una funcion *cortar :: Int -> Int -> [Char] -> [Char]* que dado dos enteros i y j y un string w, devuelva el substring que se encuentra entre las posiciones i y j

Respuesta

```haskell
cortar :: Int -> Int -> [Char] -> [Char]
cortar i j w = if j>i then (take (j-(i+1))(drop i w)) else (take
(i-(j+1))(drop j w))
```