EJEMPLOS

- Fibonacci
- Primos
- Divisores
- Criba de Eratostenes

Fibonacci con recursión no lineal

```
fibonacci :: Integer -> Integer
fibonacci 0 = 1
fibonacci 1 = 1
fibonacci n = fibonacci (n-1) + fibonacci (n-2)
```

```
> fibonacci 30
1346269
```

Fibonacci con recursión lineal I

```
fibonacci1 :: Integer -> Integer
fibonacci1 n = head (fibonacci1Aux n)

fibonacci1Aux :: Integer -> [Integer]
fibonacci1Aux 0 = [1]
fibonacci1Aux 1 = [1,1]
fibonacci1Aux n = (fib1 + fib2) : fib1 : fib2 : listfib
  where (fib1 : fib2 : listfib) = fibonacci1Aux (n-1)
```

```
> fibonacci1 30
1346269
```

Fibonacci con recursión lineal II

```
fibonacci2 :: Integer -> Integer
fibonacci2 n = fst (fibonacci2Aux n)

fibonacci2Aux :: Integer -> (Integer, Integer)

fibonacci2Aux n
  | n < 0    =  error "Fibonacci de numero negativo"
  | n == 0   =  (1, 0)
  | n == 1   =  (1, 1)
  | otherwise =
     let (fib1, fib2) = fibonacci2Aux (n-1) in (fib1 + fib2, fib1)
```

```
> fibonacci2 30
1346269
```

Resultados

> fibonacci 30
1346269
(1.93 secs, 519,581,880 bytes)

> fibonacci1 30
1346269
(0.00 secs, 88,664 bytes)

> fibonacci2 30
1346269
(0.00 secs, 83,368 bytes)

Números primos

```
primo :: Integer -> Bool
primo n
  | n < 2          = False
  | otherwise = divisores2M n (n-1) == []

divisores2M :: Integer -> Integer -> [Integer]
divisores2M n m
  | m < 2 || m > n = []
  | mod n m == 0  = m : divisores2M n (m-1)
  | mod n m /= 0  = divisores2M n (m-1)
```

Generalizar la función divisores en un intervalo cualquiera

Centésimo primo/primo n-ésimo

```
primo100 :: Integer
primo100 = lstprimos !! 99
```

- Generalizamos:

```
primoNesimo :: Integer -> Integer
primoNesimo n = lstprimos !! (n-1)
```

```
lstprimos :: [Integer]    lstprimos = ¿?
```

- lstprimos = (lista de los números primos) es infinita
- Podemos poner un límite
- lstNprimos n = lista de los n primeros primos

Lista de los n primeros números primos

```
lstNprimos :: Integer -> [Integer]
lstNprimos n = lstprimosAux n 1 [2]

lstprimosAux :: Integer -> Integer -> [Integer] -> [Integer]
lstprimosAux n cont (x:xs)
  |n  == cont = (x:xs)
  |otherwise = lstprimosAux n (cont+1) (p:x:xs)
        where p = sigprimo x
```

(sigprimo n = primo siguiente a n ¿?)

primoNesimo' n = head $ lstNprimos n ≈ head (lstNprimos n)

* Están ordenados en orden decreciente

Primer primo mayor que x

```
sigprimo x :: Integer -> Integer
sigprimo x =
   if primo (x+1) then x + 1 else sigprimo (x+1)


-- primer primo mayor que 10000
> sigprimo 10000
10007
```

Lista de los infinitos números primos

```
lstprimos :: [Integer]
lstprimos = primosAux  [2]
primosAux :: [Integer] -> [Integer]
primosAux (x:xs) = x:primosAux (p:xs)
        where p = sigprimo x

--n-ésimo número primo
primoNesimo n = lstprimos !! (n-1)
```

Números primos de nuevo

```
divisores :: Integer -> [Integer]
divisores n = [x | x <- [1..n], n `mod` x = 0]

primo n = divisores n == [1,n]

lstprimos = [p | p <- [2..], primo p]
lstNprimos n = take n lstprimos

esPrimo :: Integer -> Bool
esPrimo n = head (dropWhile (<n) lstprimos) == n
```

Criba de Eratostenes

```
lstprimos :: [Integer]
lstprimos = cribaEratos  [2..]

cribaEratos :: [Integer]->[Integer]
cribaEratos (p:xs) =
            p:cribaEratos [x |x <- xs, x `mod` p /= 0]
```