Introducción a la Programación Funcional

- Lista por comprensión

Lista por comprensión

- Se basa en la definición formal de conjunto por ejemplo la definición de los números pares mayores a 10 :

- s = { 2 * x | x € N, 2 * x > 10 }

- Esto se lee como los números pares (2*x) que pertenecen (€) a los números naturales (N) y que sean mayores que 10. La parte anterior al separador se llama la función de salida, x es la variable, N es el conjunto de entrada y 2 * x > 10 es el predicado.

Lista por comprensión

- Esta definición se podría escribir en Haskell de la siguiente manera:

  s = [2*x | x <- [0..], x*2 > 10]

- Como vemos es muy similar a la definición matemática, solo que se define los números naturales como [0..] los dos puntos indican que esta lista es infinita. Cuando pensamos en una lista infinita, automáticamente relacionamos con un bucle que nunca termina o con un cuelgue de nuestro programa, esto es porque venimos del mundo imperativo pero cómo Haskell **ejecuta las sentencias de forma perezosa**, podemos modelar listas infinitas sin problema.

Lista por comprensión

- Podemos modelar diferentes listas sin ningún problema, por ejemplo todos los números del 50 al 100 cuyo resto al dividir por 7 fuera 3 :

  ghci> [x | x <- [50..100], x `mod` 7 == 3]

  [52,59,66,73,80,87,94]

- Por suerte las listas por compresión, no son solo una característica de Haskell sino que se encuentra en diferentes lenguajes,pero solo los lenguajes que tienen lazy evaluation pueden modelar listas infinitas.

Lista por comprensión

Ceylon

```
{ for (x in 0..100) if ( x**2 > 3) x * 2 }
```

Clojure

```
 (take 20
   (for [x (range) :when (> (* x x) 3)]
     (* 2 x)))
 ;; ⇒ (4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42)
```

CoffeeScript

```
 (x * 2 for x in [0..20] when x*x > 3)
```

Elixir

```
for x <- 0..100, x*x > 3, do: x*2
```

Lista por comprensión

Erlang

```
S = [2*X || X <- lists:seq(0,100), X*X > 3].
```

F#

```
> seq { for x in 0..100 do
        if x*x > 3 then yield 2*x } ;;
val it : seq<int> = seq [4; 6; 8; 10; ...]
```

Groovy

```
s = (1..100).grep { it ** 2 > 3 }.collect { it * 2 }
s = (1..100).grep { x -> x ** 2 > 3 }.collect { x -> x * 2 }
```

Lista por comprensión

JavaScript 1.7

```
js> [2*x for each (x in [0,1,2,3,4,5,6,7]) if (x*x<5)]
[0, 2, 4]
```

Julia

```
y = [x^2+1 for x in 1:100]
```

Perl 6

```
my @s = ($_ * 2 if $_ ** 2 > 3 for ^100);
```

Python

```
S = [2 * x for x in range(101) if x ** 2 > 3]
```

Lista por comprensión

Scala

```
val s = for (x <- Stream.from(0) if x*x > 3) yield 2*x
```

Smalltalk

```
((1 to: 100) select: [:x|x*x>3]) collect: [:x|2*x]
```

En resumen podemos decir que las listas por comprensión es una característica de algunos lenguajes con lo cual podemos definir listas de forma comprensiva.

http://en.wikipedia.org/wiki/List_comprehension