Curso Elixir

Sección 10

Comprehensions

Las comprensiones algunas veces pueden ser usadas para descomponer enumerables

```
iex> list = [1, 2, 3, 4, 5]
[1, 2, 3, 4, 5]
iex> for x <- list do
...> x * x
...> end
[1, 4, 9, 16, 25]
```

Ejemplo en otros enumerables

```
# Key list
iex> klist = [x: 10, y: 20, z: 5]
[x: 10, y: 20, z: 5]

iex> for {key, value} <- klist do
...> {key, value * 2}
...> end
[x: 20, y: 40, z: 10]

# Maps
iex> map = %{:x => 10, :y => 20, :z => 5}
%{x: 10, y: 20, z: 5}

iex> for {key, value} <- map do
...> {key, value *2}
...> end
[x: 20, y: 40, z: 10]


# Binarios
iex(10)> for <<x <- str>>, do: <<x>>
["E", "l", "i", "x", "i", "r"]
```

Además de descomponer una enumerable, también se puede usar el patrón de coincidencia

```
iex> klist = [x: 10, y: 20, x: 5]
iex> for {:x, value} <- klist do
...> value
...> end
[10, 5]
```

Es posible usar múltiples generadores como iteraciones anidadas:

```
iex(16)> for n <- list, i <- 1..n, do: IO.puts "#{n} - #{i}"
1 - 1
2 - 1
2 - 2
3 - 1
3 - 2
```

```
3 - 3
[:ok, :ok, :ok, :ok, :ok, :ok]
```

Filtros

Los filtros son guardias en las listas e comprensión

```
iex> import Integer
Integer
iex> for x <- 1..10, is_odd(x), do: x
[1, 3, 5, 7, 9]
```

Es posible realizar más de un filtro

```
import Integer
iex>for x <- 1..10, is_odd(x), x > 3, do: x
[5, 7, 9]
```

Into

Como ya vimos las compresiones siempre regresan una lista. Para poder especificar el tipo de resultado usamos :into

```
iex> klist = [x: 10, y: 20, x: 5]
[x: 10, y: 20, x: 5]
iex> for {key, value} <- klist, into: %{} do
...> {key, value}
...> end
%{x: 5, y: 20}
```