# Curso Elixir

## Sección 9

### Enum

El módulo Enum incluye más de 70 funciones para trabajar con colecciones.

En iex hagamos los siguiente para desplegar las funciones del módulo

```
iex> Enum.__info__(:functions) |> Enum.each(fn({function, arity}) ->
iex> IO.puts "#{function}/#{arity}"
iex> end)
```

### Funciones de Enum

ahora veamos algunas de estas funciones

- all?

  Cuando usas all?/2,y muchas de las funciones de Enum, proveemos una función anónima para aplicarla a todos los elementos de nuestra colección.

  ```
  iex> Enum.all?(["foo", "bar", "hello"], fn(s) -> String.length(s) == 3 end)
  iex> Enum.all?(["foo", "bar", "hello"], fn(s) -> String.length(s) > 1 end)
  ```

- any?

  ```
  iex> Enum.any?(["foo", "bar", "hello"], fn(s) -> String.length(s) == 5 end)
  ```

- chunk_every

  Divide tu colección en fragmentos

  ```
  iex> Enum.chunk_every([1, 2, 3, 4, 5, 6], 2)
  iex> Enum.chunk_every([1, 2, 3, 4, 5, 6], 4)
  ```

- chunk_by

  Igual que chunk_every pero basado en algo más que tamaño

  ```
  iex> Enum.chunk_by(["one", "two", "three", "four", "five"], fn(x) -> String.length(x) end)
  iex> Enum.chunk_by(["one", "two", "three", "four", "five", "six"], fn(x) -> String.length(x) end)
  ```

- map_every

  Ejecuta una función cada n elementos

```
iex> Enum.map_every([1, 2, 3, 4, 5, 6, 7, 8], 3, fn x -> x + 100 end)
```

- each

  Puede ser necesario iterar sobre una colección sin producir un nuevo valor

  ```
  iex> Enum.each(["one", "two", "three"], fn(s) -> IO.puts(s) end)
  ```

- map

  Aplica la función a cada elemento y regresa una nueva colección

  ```
  iex> Enum.map([0, 1, 2, 3], fn(x) -> x - 1 end)
  ```

- min

  min/1 Encuntra el valor mínimo en una colección

  ```
  iex> Enum.min([5, 3, 0, -1])
  ```

  min/2 realiza lo mismo, pero en caso que el enumerable sea vacío, este permite que se especifique una función que produzca el valor del mínimo

  ```
  iex> Enum.min([], fn -> :foo end)
  ```

- max

  max/1 retorna el máximo valor de la colección

  ```
  iex> Enum.max([5, 3, 0, -1])
  ```

  `

  max/2

  ```
  Enum.max([], fn -> :bar end)
  ```

- filter

  La función filter/2 nos permite filtrar una colección que incluya solamente aquellos elementos que evalúan a true utilizando la función anónima

  ```
  iex> Enum.filter([1, 2, 3, 4], fn(x) -> rem(x, 2) == 0 end)
  ```

- reduce

Con reduce/3 podemos transformar nuestra colección a un único valor. Para hacer esto aplicamos un acumulador opcional que será pasado a nuestra función; si no se provee un acumulador, el primer valor en el enumerable es usado

```
iex> Enum.reduce([1, 2, 3], 10, fn(x, acc) -> x + acc end)
iex> Enum.reduce([1, 2, 3], fn(x, acc) -> x + acc end)
iex> Enum.reduce(["a","b","c"], "1", fn(x,acc) -> x <> acc end)
```

- sort

  Ordenar nuestras colecciones

  ```
  iex> Enum.sort([5, 6, 1, 3, -1, 4])
  iex> Enum.sort([:foo, "bar", Enum, -1, 4])
  ```

  Mientras que sort/2 nos permite proveer una función de ordenación propia:

  ```
  iex> Enum.sort([%{:count => 4}, %{:count => 1}], fn(x, y) -> x[:count] > y[:count] end)
  # sin la función
  iex> Enum.sort([%{:count => 4}, %{:count => 1}])
  ```

- uniq_by

  Podemos usar uniq_by/2 para eliminar duplicados de nuestras colecciones:

  ```
  iex> Enum.uniq_by([1, 2, 3, 2, 1, 1, 1, 1, 1], fn x -> x end)
  ```