Curso Elixir

Sección 6

if y unless

Es probable que hayas visto if/2 antes, unless funciona de manera inversa a if. En Elixir ambos están definidos como macros, no son construcciones propias del lenguaje.

Debería tenerse en cuenta que en Elixir, los únicos valores falsos son nil y el booleano false.

```
iex> if String.valid?("Hello") do
...>   "Valid string!"
...> else
...>   "Invalid string."
...> end
"Valid string!"
```

```
iex> unless is_integer("hello") do
...>   "Not an Int"
...> end
"Not an Int"
```

case

Si es necesario buscar una coincidencia en múltiples patrones podemos usar case:

```
iex> case {:ok, "Hello World"} do
...>   {:ok, result} -> result
...>   {:error} -> "Uh oh!"
...>   _ -> "Catch all"
...> end
"Hello World"
```

La variable _ es una inclusión importante en la declaración case. Sin esto, cuando no se encuentre una coincidencia, se lanzará un error:

```
iex> case :even do
...>   :odd -> "Odd"
...> end
** (CaseClauseError) no case clause matching: :even

iex> case :even do
...>   :odd -> "Odd"
...>   _ -> "Not Odd"
...> end
"Not Odd"
```

Si intentas coincidir con variables existentes debes usar el operador pin ^:

```
iex> pie = 3.14
 3.14
```

```
iex> case "cherry pie" do
...>   ^pie -> "Not so tasty"
...>   pie -> "I bet #{pie} is tasty"
...> end
"I bet cherry pie is tasty"
```

Otra característica interesante de case es que soporta cláusulas de guardia

```
iex> case {1, 2, 3} do
...>   {1, x, 3} when x > 0 ->
...>     "Will match"
...>   _ ->
...>     "Won't match"
...> end
"Will match"
```

Las funciones anónimas también pueden tener cláusulas de guardia

```
iex> f = fn
...>   x, y when x > 0 -> x + y
...>   x, y -> x * y
...> end
#Function<13.126501267/2 in :erl_eval.expr/5>
iex> f.(1, 3)
4
iex> f.(-1, 3)
-3
```

cond

Cuando necesitamos coincidencias con condiciones, y no con valores, podemos usar cond; esto es parecido a else if o elsif

```
iex> x = 10
10
iex> cond do
...> x + 2 == 11 ->
...> "falso"
...> x *2 < 10 ->
...> "falso"
...> x == 10 ->
...> "true"
...> end
"true"
```

al igual que case, cond regresará un error en caso de no coincidir, para esto podemos usar true

```
iex> cond do
...(4)> 2 * 2 == 2 ->
...> "Error"
...> 3 * 3 == 3 ->
...> "Error"
...> true -> "Todo mal"
...> end
```

with

Podemos usar este operador cuando tenemos case anidados

```
iex> usuario = %{nombre: "miguel", apellido: "iñiguez"}
%{apellido: "iñiguez", nombre: "miguel"}
iex> with {:ok, nombre} <- Map.fetch(usuario, :nombre),
...> {:ok, apellido} <- Map.fetch(usuario, apellido),
...> do: apellido <> " " <> nombre
"iñiguez miguel"
```

En caso de que la expresión falle, regresará :error

```
iex> usuario = %{nombre: "miguel"}
%{nombre: "miguel"}
iex> with {:ok, nombre} <- Map.fetch(usuario, :nombre),
...> {:ok, apellido} <- Map.fetch(usuario, :apellido),
...> do: apellido <> "," <> nombre
:error
```

También es posible utilizar else dentro de with

```
iex> user = %{password: "admin123", username: "admin"}
iex> with {:ok, username} <- Map.fetch(user, :username),
...> true <- String.length(username) > 8 do
...> {:ok, "valid username"}
...> else
...> _ -> {:error, "invalid username"}
...> end
```

Actividad 3

Usando with refactorizar el siguiente código

```
iex> case Map.fetch(user, :username) do
...> {:ok, username} ->
...> case Map.fetch(user, :password) do
...> {:ok, password} ->
...> IO.inspect("User: "<>username<>" Pass: "<> password)
...> :error -> :error
...> end
...> :error -> :error
...> end
```