

# + Introducción a Elixir sesión 4.

Santiago Posada

SofkaU

#ElDesafíoEsContigo



## Multi Clause function:

```
defmodule Geometry do
  def area({:rectangle, a, b}) do
    a * b
  end

  def area({:square, a}) do
    a * a
  end

  def area({:circle, r}) do
    r * r * 3.14159
  end
end
```



## Guards:



```
defmodule Playground do
  def test(x) when x < 0 do
    :negative
  end


  def test(0), do: :zero

  def test(x) when x > 0 do
    :positive
  end
end
```

```
### number < atom < reference < function < port < pid
### < tuple < map < list < bitstring (binary)
```







```
defmodule Playground do
  def test(x) when is_number(x) and x < 0 do
    :negative
  end

  def test(0), do: :zero

  def test(x) when is_number(x) and x > 0 do
    :positive
  end
end
```

```
## (=, ≠, ≡, ≇, >, <, ≤, ≥)
## and, or, not, !
## (+, -, *, /)
## Kernel.is_number(term) Kernel.is_atom(term) etc
```

## Guards:

```
iex(12)> double = fn x -> x*2 end  
double = fn x -> x*2 end  
#Function<6.128620087/1 in :erl_eval.expr/5>  
iex(13)> double.(3)  
double.(3)  
6
```

## Multi Clause function: con funciones anónimas, guards y verificación de tipo de dato

```
defmodule Playground do
  test = fn
    x when is_number(x) and x < 0 →
      :negative

    0 →
      :zero

    x when is_number(x) and x > 0 →
      :positive
  end
end
```



## IF y Unless:

if => Cuando la condición se cumpla haga esto o sino esto

unless => a menos que la condición se cumpla haga esto o si no haga esto otro

- Cuando la condición no se cumpla haga esto y sino haga esto otro.

```
def max(a, b) do
  if a ≥ b, do: a, else: b
end

def max(a, b) do
  unless a ≥ b, do: b, else: a
end
```

Bloque COND / DO  
VS  
Multi clause functions  
VS  
SWITCH CASE o CASE statement

```
def max(a, b) do
  cond do
    a ≥ b → a
    true → b
  end
end
```

```
def max(a, b) do
  case a ≥ b do
    true → a
    false → b
    _ → :error
  end
end
```

```
def max(a, b) when a ≥ b do
  a
end

def max(a, b) do
  b
end
```



## WITH:

```
with pattern1 <- expression1,  
    pattern2 <- expression2,  
    ...  
do  
    # Código a ejecutar si todos los patrones coinciden  
else  
    # Código a ejecutar si alguno de los patrones no coincide (opcional)  
end
```

WITH:

```
defmodule UserManager do
  def create_user(params) do
    with {:ok, user} <- insert_user_into_db(params),
         {:ok, email} <- send_welcome_email(user) do
      {:ok, user}
    else
      {:error, reason} -> {:error, reason}
    end
  end
end

defp insert_user_into_db(params) do
  # Lógica para insertar el usuario en la base de datos
  # Por ejemplo, podría devolver {:ok, user} o {:error, reason}
end

defp send_welcome_email(user) do
  # Lógica para enviar el correo electrónico de bienvenida
  # Por ejemplo, podría devolver {:ok, email} o {:error, reason}
end
end
```

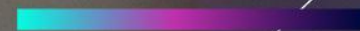





WITH:

```
defmodule Playground do
  def extract_user(user) do
    with {:ok, login} <- extract_login(user),
         {:ok, email} <- extract_email(user),
         {:ok, password} <- extract_password(user) do
      {:ok, %{login: login, email: email, password: password}}
    end
  end

  defp extract_login(%{"login" => login}), do: {:ok, login}
  defp extract_login(_), do: {:error, "login missing"}
  defp extract_email(%{"email" => email}), do: {:ok, email}
  defp extract_email(_), do: {:error, "email missing"}
  defp extract_password(%{"password" => password}), do: {:ok, password}
  defp extract_password(_), do: {:error, "password missing"}
end
```





# ¿Qué es un Struct?





&lt; / &gt;



Calle 12 # 30-80 Medellín

Calle 85 # 11 – 53 Int 6 Of. 301 Bogotá



+57 604 266 4547



info@sofka.com.co



www.sofka.com.co



Síguenos



Sofka Technologies



Sofka\_Technologies

