Funkcionalno programiranje u Elixiru

Funkcionalno programiranje

- Deklarativno (fokus na što a ne kako)
- Lakše čitanje
- Kraći programi
- Jednostavnije održavanje

```
def fact(1), do: 1
def fact(n) when n > 1, do: n * fact(n - 1)
```

```
int fact(int n)
{
  int i, fact = 1;

  if (n < 0) {
    return 0;
  }

  for (i = 1; i <= n; i++) {
    fact = fact * i;
  }

  return fact;
}
```

Erlang

Erlang/OTP

- Razvijen u Ericssonu, 1986. Godine
- Primjena: telefonski switchevi
- OTP - Open Telecom Platform
    - Concurrent
    - Distributed
    - Fault-tolerant
    - Hot swapping
- Defanzivno programiranje je antipattern u Erlangu (i Elixiru)


Elixir

Elixir

- Spaja Erlang VM sa modernom funkcionalnom sintaksom
- Dynamic typing
- Metaprogramiranje
- IEx (Elixir's interactive shell)
- mix
- mix format
- Hex (Erlang/Elixir package manager)

Elixir - pattern matching

```elixir
x = 1

{x, y} = {1, 2}
```

```elixir
case {1, 2} do
  {2, 3} ->
    "not match"

  {1, 2} ->
    "match"
end
```

```elixir
case {:ok, 2} do
  {:ok, result} when result > 5 ->
    "match with access to #{result} and result > 5"

  {:ok, result} ->
    "match with access to #{result} and result not > 5"

  {:error, message} ->
    IO.puts(message)
end
```

Elixir - pattern matching

```elixir
def zero?(x) do
  if !is_integer(x) do
    raise "NaN"
  end

  if x == 0 do
    true
  else
    false
  end
end
```

```elixir
def zero?(0), do: true
def zero?(x) when is_integer(x), do: false
```

Elixir - rekurzija

```elixir
def sum_list([head | tail], accumulator) do
  sum_list(tail, head + accumulator)
end

def sum_list([], accumulator), do: accumulator
```

```elixir
Enum.reduce([1, 2, 3], 0, fn x, acc -> x + acc end)
```

Elixir - moduli

```
defmodule Math do
  def sum_list([head | tail], accumulator) do
    sum_list(tail, head + accumulator)
  end

  def sum_list([], accumulator), do: accumulator
end

IO.puts(Math.sum_list([1, 2, 3], 0))
```

```
is_integer(x)
```

Elixir - pipe

```elixir
IO.puts(Math.sum_list([1, 2, 3], 0))

[1, 2, 3]
|> Math.sum_list(0)
|> IO.puts
```

```elixir
list =
  [1, [2], 3]
  |> List.flatten()
  |> Enum.map(fn x -> x * 2 end)
```

```elixir
def get_resp(path) do
  path
  |> get_api_url()
  |> HTTPoison.get!()
  |> Map.get(:body)
  |> IO.inspect()
  |> Jason.decode!()
end
```

Elixir - pipe

```elixir
defmodule Tools do
  def grep(file_path, regex) do
    file_path
    |> File.read!()
    |> String.split("\n", trim: true)
    |> Enum.filter(fn line -> Regex.match?(regex, line) end)
  end
end
```

```
iex(2)> Tools.grep("materijali/primjeri/5-pipe.ex", ~r/(E|e)num/)
["  |> Enum.map(fn x -> x * 2 end)",
 "    |> Enum.filter(fn line -> Regex.match?(regex, line) end)"]
```

Phoenix

Phoenix

- Model View Controller (MVC) arhitektura
- Rails style generacija koda
- Nije monolitičan kao Rails
    - Plug (web server, router)
    - Ecto (pristup bazi, migracije)
    - LiveView

```
$ mix phx.new demo
$ mix phx.gen.html Catalog Product products title:string price:decimal nabavanet_id:integer
```

Ecto

```
from(m in Movie, where: m.id > 1000, select: m.title)
|> Repo.all()
```

```
movies = from(m in Movie, where: [stars: 5])

query =
  from(c in Character,
    join: ^movies,
    on: m.id == c.movie_id,
    where: c.name == "Vito Corleone",
    select: {m.title, c.name}
  )

Repo.all(query)
```

LiveView

- “Rich, real-time user experiences with server-rendered HTML”
- Pametno praćenje promjena (šalje samo promjenu klijentu)
- Klijent se automatski ažurira nakon promjene na serveru
- Nije potreban REST API i JS glue kod

Demo

https://github.com/bdeak4/elixir-demo