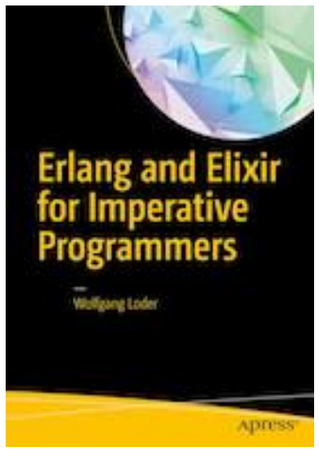
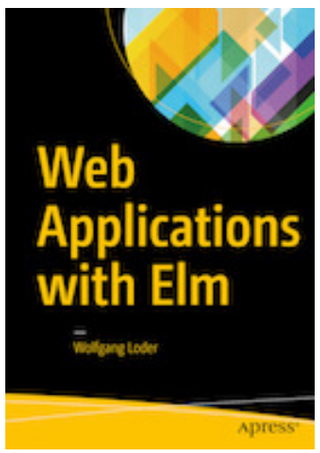
# Functional Concepts in Elixir

# Wolfgang Loder

## Software Engineer




*“Erlang and Elixir for Imperative Programmers”, 2016*




*“Web Applications with Elm”, 2018*

# Functional Concepts

**Assign code to process particular data**

**Passing code as parameters**

**Processing structured data**

**Processing application state in steps**

**Ensuring integrity of data**

# Pattern Matching

*“Assignment Operator”*

*Match Operator*

**Expression** **=** **Expression**

**“Pattern”** **“Term”**

# Pattern Matching

```
current_user_id = "eyJhbGciOiJIUzI1NiIsInR5cCI6"
```

**pattern** **term**

current_user_id **is set to** “eyJhbGciOiJIUzI1NiIsInR5cCI6”

“eyJhbGciOiJIUzI1NiIsInR5cCI6” **matches** current_user_id

current_user_id **is bound to** “eyJhbGciOiJIUzI1NiIsInR5cCI6”

# Pattern Matching

```
"eyJhbGciOiJIUzI1NiIsInR5cCI6" = current_user_id
```

current_user_id **matches** “eyJhbGciOiJIUzI1NiIsInR5cCI6”

```
current_user_id = get_session(conn, :current_user_id)
```

**Expression**

# Pattern Matching

```
1  current_user_id = "eyJhbGciOiJIUzI1NiIsInR5cCI6"
2  current_user_id = get_session(conn, :current_user_id)
```

"user@googlemail.com"

1 current_user_id **is bound to** “eyJhbGciOiJIUzI1NiIsInR5cCI6”

2 current_user_id **is bound to** “user@googlemail.com”

# Pattern Matching

*Pin operator*

^

```
1   ^current_user_id = "eyJhbGciOiJIUzI1NiIsInR5cCI6"
2   current_user_id = get_session(conn, :current_user_id)
```

** (MatchError) no match of right hand side value:
“user@googlemail.com"

# Pattern Matching

*interpolation/scope expression*

```
q = from p in Payment,
limit: ^pagesize,
offset: ^page,
select: p
```

# Pattern Matching

```elixir
{:ok, pid} = DynamicSupervisor.start_child
                                (Logic.Supervisor,
                                 Logic.Worker)
```

```elixir
%{"pagesize" => pagesize, "page" => page} = params
```

```elixir
<<"TRIPIQ_CONFIG",
    valid :: size(8),
    group :: size(8)>> = <<"TRIPIQ_CONFIG", 1, 99 >>
```

# Pattern Matching

```elixir
{ users, count} = Auth.list_users(predicate)
```

```elixir
{ users, _} = Auth.list_users(predicate)
```

```elixir
1 { users, _count} = Auth.list_users(predicate)
2 { otherusers, _count} = Auth.list_users(predicate)
```

```elixir
_ = 42  # -> 42
```

Compile Error

```elixir
_count = 42  # -> 42
```

Warning

# Pattern Matching

```
case Safiriservice.Auth.authenticate_user(email,
password) do
  {:ok, user} ->
    …
  {:error, message} ->
    …
end

[head | tail] = [1, 2, 3]
```

# Pattern Matching

```elixir
def show(conn, %{"id" => id}) do
  pizza = Model.get_pizza!(id)
  render(conn, "show.json", pizza: pizza)
end
```

```elixir
def delete_user(%User{} = user) do
  Repo.delete(user)
end
```

```elixir
defp validate_pass(%{other: %{password: ""}}) do
  {:error, "Password required"}
end
```

# Pattern Matching

```
def render("404.html", _assigns) do  …

def render("500.html", _assigns) do  …
```

```
def render("401.json", %{message: message}) do …

def render("404.json", _assigns) do  …

def render(_, _assigns) do  …
```

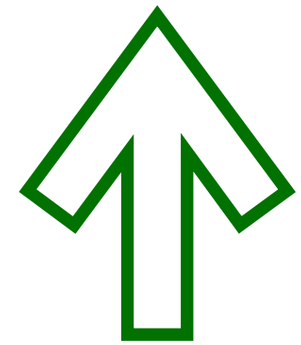
# Pattern Matching

```
def index(conn, params) when params == %{} do …
```

```
def show(conn, %{"id" => id})
    when is_integer(id) do …
```

# Higher Order Function

```
{ users, _ } = Auth.list_users(predicate)
```



**Functions as argument and/or return functions**

# Higher Order Function

```elixir
sum = fn(x, y) -> x + y end
list |> Enum.reduce(0, sum)

Enum.Filter([1, 2, 3], func)

Enum.Map([1, 2, 3], func)

Keyword.get_and_update([a: 1], :a, func)
```

# Higher Order Function

```
def create_from_record do
  fn (d) -> documentrecord(
    docid: documentrecord(d,:docid)+1,
    name: documentrecord(d,:name))
  end
end
```

# Higher Order Function

```elixir
def call_function(function, a) do
  function.(a)
end
```

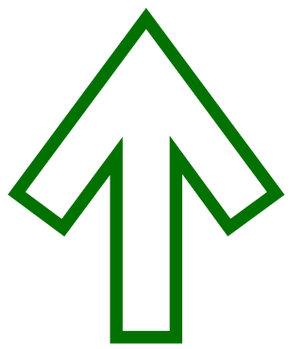
```elixir
def call_with_fn() do
  call_function(fn n -> n*n end,4)
end

def call_with_variable() do
  sqfunc = fn n -> n*n end
  call_function(sqfunc, 4)
end
```

# Higher Order Function

```
def direct_call() do
  fn n -> n*n end.(4)
end

def call_with_variable_shorthand() do
  sqfunc = &(&1*&1)
  call_function(sqfunc, 4)
end

def direct_call_shorthand() do
  (&(&1*&1)).(4)
end
```

# Higher Order Function

```
def multiply(x, y) do
  x*y
end

def double() do
  fn x -> multiply(2,x) end
end

c = double()
c.(21)
```



**Currying**

# Recursion

## Available in imperative languages

```
int recursiveCall(int data) {
   f1(data);
   return recursiveCall(data);
}
```

# Recursion

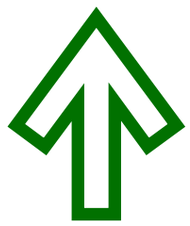
```elixir
def get_nested_map_from_list(nm, nestedlevel)
                                  when nestedlevel < 1 do
  nm
end
```

```elixir
def get_nested_map_from_list(nm, nestedlevel) do
  l = List.first(nm)
  get_nested_map_from_list(l, nestedlevel-1)
end
```

**Tail-Call Optimization**

# Recursion

```
nontailrecursiveloop(N)  ->
   io:format("~w~n", [N]),
   2 * nontailrecursiveloop(N).
```



## Body-Recursive

# Recursion

**Iteration through data structures**

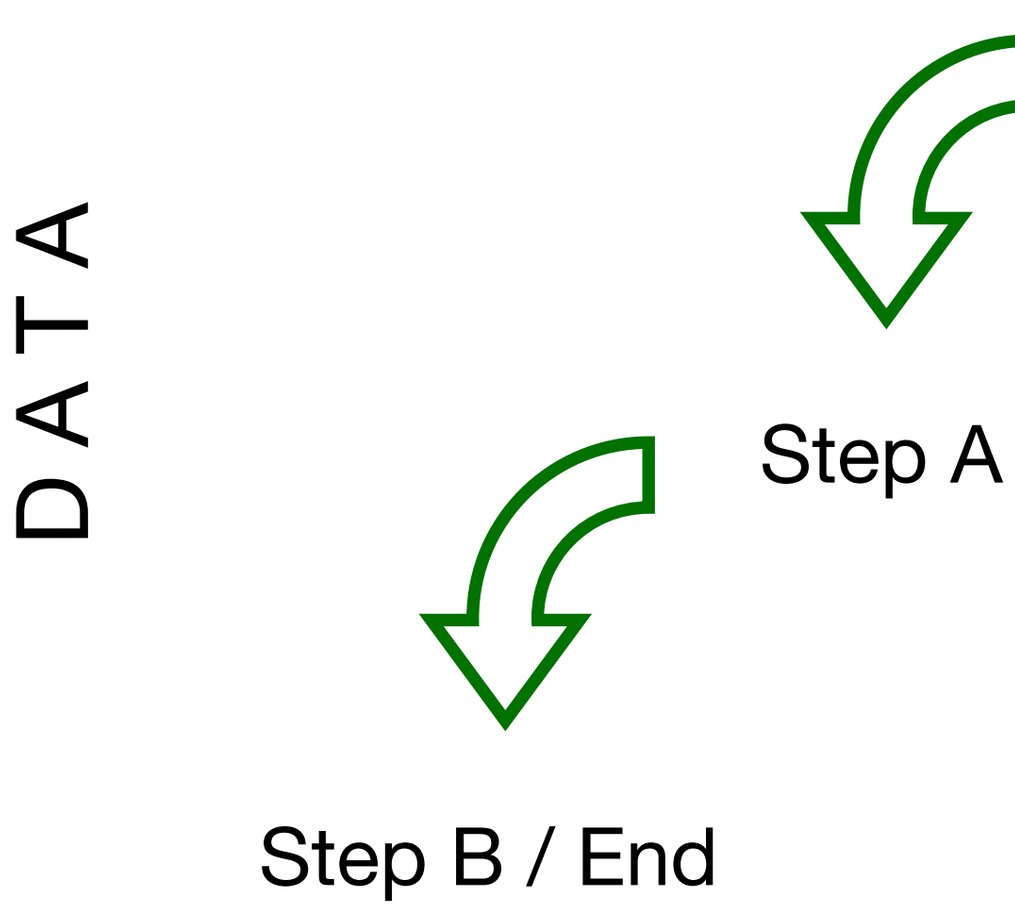
**No for loops**

```
def sum([]), do: 0

def sum([head | tail]), do: head + sum(tail)
```

**Enum, Map, Stream**

# Continuation Passing

## Workflow



# Continuation Passing

```elixir
def initiate(data) do
  case data do
    {:ok,1} -> continue({:ok,2},&StepA/1)
    _ -> :error
  end
end

defp continue(data,f) do
  f.(data)
end
```

# Continuation Passing

```elixir
defp stepA(data) do
  case data do
    {:ok,2} -> continue({:stop},&stepB/1)
    _ -> :error
  end
end

defp stepB(data) do
  case data do
    {:stop} -> {:stopped}
    _ -> :error
  end
end
```

# Continuation Passing

## Pro

Breaking down of (long-running) processing of data

## Alternatives

Recursion

Supervisor / GenServer

Async /Await

# Referential Transparency

```elixir
def return_number(n) when is_number(n) do
  n
end
```

```elixir
return_number(42) # -> 42
```

**always**
**same result**

# Referential Transparency

```
def multiply(x, y) when is_number(x) and is_number(y)
 do
   x*y
end
```

```
def double() do
   fn x when is_number(x) -> multiply(2,x) end
end
```

```
c = double
c.("a")
```

```
** (FunctionClauseError) no function clause matching
```

# Referential Transparency

**Why important?**

Code Optimization / Memoization

**How to achieve?**

Pure Functions

Immutable Data

Deterministic Functions

# Functional Concepts

Assign code to process particular data

-> Pattern Matching

Passing code as parameters

-> Higher Order Function

Processing structured data

-> Recursion

# Functional Concepts

Processing application state in steps

-> Continuation Passing

Ensuring integrity of data

-> Referential Transparency

# The End

**@wolfgang_loder**

**kujua**

elixir forum