Learning Functional Programming with Elixir|>

A Short Guide Through Functional Programming

Kenny Ballou

zData, Inc.

March 19, 2016

Who am I?

- Hacker
- Developer (read gardener)
- Mathematician
- Student

Figure: "Functional programming combines the flexibility and power of abstract mathematics with the intuitive clarity of abstract mathematics."

XKCD on Functional Programming[6]

Motivation

Why Functional Programming?

- Easy to Reason About

Motivation

Why Functional Programming?

- Easy to Reason About
- Trivial to Test

Motivation

Why Functional Programming?

- Easy to Reason About
- Trivial to Test
- Functional Composition

Motivation

Why Functional Programming?

- Easy to Reason About
- Trivial to Test
- Functional Composition
- State or Side-Effects are explicit

Anti-Motivation

Why not Functional Programming?

- Explicit state can be hard

Anti-Motivation

Why not Functional Programming?

- Explicit state can be hard
- Side-effect free programming seems cumbersome

Anti-Motivation

Why not Functional Programming?

- Explicit state can be hard
- Side-effect free programming seems cumbersome
- Performance

Anti-Motivation

Why not Functional Programming?

- Explicit state can be hard
- Side-effect free programming seems cumbersome
- Performance
- Learning Curve

What is Functional Programming?

- Functional programming is a paradigm

What is Functional Programming?

- Functional programming is a paradigm
- Prefers “mathematical” functions

What is Functional Programming?

- Functional programming is a paradigm
- Prefers “mathematical” functions
- Uses (mostly) immutable data

What is Functional Programming?

- Functional programming is a paradigm
- Prefers “mathematical” functions
- Uses (mostly) immutable data
- Everything is an expression

What is Functional Programming?

- Functional programming is a paradigm
- Prefers “mathematical” functions
- Uses (mostly) immutable data
- Everything is an expression
- Functions are 1st class

What is Functional Programming?

- Functional programming is a paradigm
- Prefers “mathematical” functions
- Uses (mostly) immutable data
- Everything is an expression
- Functions are 1st class
  - This gives higher-order functions

What is Elixir? Erlang/OTP?

- Elixir is a new functional, dynamically typed language

What is Elixir? Erlang/OTP?

- Elixir is a new functional, dynamically typed language
- Elixir's design is heavily influenced by Erlang's design

What is Elixir? Erlang/OTP?

- Elixir is a new functional, dynamically typed language
- Elixir's design is heavily influenced by Erlang's design
  - Erlang is also a functional, dynamically typed language, first appeared in 1986

What is Elixir? Erlang/OTP?

- Elixir is a new functional, dynamically typed language
- Elixir's design is heavily influenced by Erlang's design
  - Erlang is also a functional, dynamically typed language, first appeared in 1986
  - Built around concurrency, fault-tolerance, and high-availability

What is Elixir? Erlang/OTP?

- Elixir is a new functional, dynamically typed language
- Elixir's design is heavily influenced by Erlang's design
  - Erlang is also a functional, dynamically typed language, first appeared in 1986
  - Built around concurrency, fault-tolerance, and high-availability
- Elixir compiles to BEAM (Erlang) bytecode

What is Elixir? Erlang/OTP?

- Elixir is a new functional, dynamically typed language
- Elixir's design is heavily influenced by Erlang's design
  - Erlang is also a functional, dynamically typed language, first appeared in 1986
  - Built around concurrency, fault-tolerance, and high-availability
- Elixir compiles to BEAM (Erlang) bytecode
- Elixir "looks" like Ruby

Interactive Elixir

`iex`

```
Erlang/OTP 18 [erts-7.2.1] [source] [64-bit] [smp:8:8] [async-
    threads:10] [hipe] [kernel-poll:false]

Interactive Elixir (1.2.3) - press Ctrl+C to exit (type h() ENTER
     for help)

iex(1)>
```

Hello, World

```
iex> "hello, world"
"hello, world"
```

Hello, World

```
iex> IO.puts "hello, world"
hello, world
:ok
```

Elixir|> Essentials

```
iex> 42
42
iex> :ok
:ok
iex> [1, 2, 3, 4]
[1, 2, 3, 4]
iex> {:reply, 42}
{:reply, 42}
iex> 'hello, world'
'hello, world'
iex> [104, 101, 108, 108, 111]
'hello'
iex> 'こにちは、世界'
[12371, 12395, 12385, 12399, 12289, 19990, 30028]
iex> "Hello, 世界"
"Hello, 世界"
iex> <<"Hello, 世界" :: utf8>>
"Hello, 世界"
```

Elixir|> Essentials

```
iex> %{}
%{}
iex> %{a: 1, b: 2}
%{a: 1, b: 2}
iex> defmodule Person do
...>   defstruct name: nil, age: 0, height: 0
...> end
iex> %Person{name: "Kenny Ballou", age: 24, height: 177}
%Person{name: "Kenny Ballou", age: 24, height: 177}
```

Elixir|> Essentials

```
iex> f = fn(x) -> x * x end
#Function<6.54118792/1 in :erl_eval.expr/5>
iex> f.(2)
4
iex> defmodule Foobar do
...>   def foo(x), do: x * 2
...>   def bar(y) do
...>     y |> foo() |> (&*/2).(3)
...>   end
...> end
iex> Foobar.foo(5)
10
iex> Foobar.bar(2)
12
```

Dispelling Assignment

There is no spoon

- = does **not** mean *assign*
  - `x = 1` is not *assign 1 to* x
- = is a *match* operator
  - = is a constraint-solving operator

Pattern Matching

```
iex> x = 1
1
iex> 1 = x
1
iex> x = 2
2
iex> 1 = x
** (MatchError) no match of right hand side value: 2
```

Pattern Matching

```
iex> [a, b, c] = [1, 2, 3]
[1, 2, 3]
iex> a
1
iex> b
2
iex> c
3
iex> [1, _, c] = [1, 2, 3]
[1, 2, 3]
iex> [2, _, d] = [1, 2, 3]
** (MatchError) no match of right hand side value: [1, 2, 3]

iex> [h|t] = [1, 2, 3]
[1, 2, 3]
iex> h
1
iex> t
[2, 3]
```

Pattern Matching

```
iex> %{a: 1} = %{a: 1, b: 2, c: 3}
%{a: 1, b: 2, c: 3}
iex> %{} = %{a: 3}
%{a: 3}
```

Pattern Matching

```
defmodule Foobar do
  def sum_list([]), do: 0
  def sum_list([h|t]), do: h + sum_list(t)
end

iex> Foobar.sum_list [1, 2, 3, 4, 5]
15
```

Pattern Matching

Brief Introduction to IEEE-754

- 64-bit floating point (doubles) numbers are represented using IEEE-754
- 32-bit (single/`float`) and 128-bit (quadurpals) are similarly represented with varying number of bits for each component
- There are four main components
  - sign, $\pm$, 1 bit
  - exponent, 11 bits
  - fraction (mantissa), 52 bits
  - bias, built-in, typically 1023 for doubles

Pattern Matching

Brief Introduction to IEEE-754

To convert from binary bits to a "float", we can use the following formula:

$$(-1^{\text{sign}}) \cdot \left(1 + \frac{\text{mantissa}}{2^{52}}\right) \cdot (2^{\text{exponent}-\text{bias}})$$

Pattern Matching

```
iex> << sign :: size(1), exp :: size(11), mantissa :: size(52)>>
    =  <<3.14159 :: float>>
<<64, 9, 249, 240, 27, 134, 110>>
iex> sign
0
iex> exp
1024
iex> mantissa
2570632149304942
iex> :math.pow(-1, sign) *
...> (1 + mantissa / :math.pow(2, 52)) *
...> :math.pow(2, exp - 1023)
3.14159
```

The Functional Approach

To all the things!

- Less iteration

The Functional Approach

To all the things!

- Less iteration
- More (Tail) Recursion

The Functional Approach

To all the things!

- Less iteration
- More (Tail) Recursion
- Must Relearn Patterns

The Functional Approach

To all the things!

- Less iteration
- More (Tail) Recursion
- Must Relearn Patterns
- Performance

Fibonacci

$$F_n = F_{n-1} + F_{n-2}$$
$$F_0 = 0$$
$$F_1 = 1$$

Fibonacci

```
1 defmodule Fib do
2   def seq(0), do: 0
3   def seq(1), do: 1
4   def seq(n) when n > 1, do: seq(n-1) + seq(n-2)
5 end
6
7 IO.puts Fib.seq(50)
```

Fibonacci

Iteratively

```
1  defmodule Fib do
2    def seq(0), do: 0
3    def seq(1), do: 1
4    def seq(n) when n > 1, do: compute_seq(n, 1, [1, 0])
5
6    defp compute_seq(n, i, acc) when n == i do
7      hd(acc)
8    end
9    defp compute_seq(n, i, acc) do
10     compute_seq(n, i+1, [hd(acc) + (acc |> tl |> hd) | acc])
11   end
12 end
13
14 IO.puts Fib.seq(50)
```

Fibonacci Performance

```
% /usr/bin/time elixir fib_rec.exs
12586269025
603.66user 0.08system 10:04.26elapsed 99%CPU (0avgtext+0avgdata
    32892maxresident)k
2856inputs+0outputs (0major+6861minor)pagefaults 0swaps

% /usr/bin/time elixir fib_itr.exs
12586269025
0.25user 0.05system 0:00.27elapsed 112%CPU (0avgtext+0avgdata
    38668maxresident)k
0inputs+0outputs (0major+7450minor)pagefaults 0swaps
```

Quicksort

- Similar to merge sort
- Sort by partitioning
- Has a nice recursive definition

Quicksort

```elixir
defmodule Quicksort do
  def sort([]), do: []
  def sort([h|t]) do
    {lower, upper} = t |> Enum.partition(&(&1 <= h))
    sort(lower) ++ [h] ++ sort(upper)
  end
end

1..10
  |> Enum.shuffle
  |> IO.inspect
  |> Quicksort.sort
  |> IO.inspect
```

Quicksort

```
% /usr/bin/time elixir qs.exs
[4, 3, 5, 6, 7, 8, 10, 9, 1, 2]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
0.28user 0.02system 0:00.27elapsed 112%CPU (0avgtext+0avgdata
    38936maxresident)k
0inputs+8outputs (0major+6654minor)pagefaults 0swaps
```

Map-Reduce

- Functional way to process collections

Map-Reduce

- Functional way to process collections
- Can be (partially) pipelined

Map-Reduce

- Functional way to process collections
- Can be (partially) pipelined
- Mapping can be lazy

Map-Reduce

- Functional way to process collections
- Can be (partially) pipelined
- Mapping can be lazy
- Map is Reduce

Map-Reduce

- Functional way to process collections
- Can be (partially) pipelined
- Mapping can be lazy
- Map is Reduce
- They are “folds”

Map-Reduce

Implementing our own Map

```
1 defmodule MyMap do
2   def map([], _), do: []
3   def map([h|t], f) do
4     [f.(h)] ++ map(t, f)
5   end
6 end
7
8 [1, 2, 3, 4, 5] |> MyMap.map(fn(x) -> x * 2 end) |> IO.inspect
```

```
% elixir my_map.exs
[2, 4, 6, 8, 10]
```

Map-Reduce

Implementing our own (simple) Reduce

```
1  defmodule MyReduce do
2    def reduce([], acc, _), do: acc
3    def reduce([h|t], acc, f) do
4      reduce(t, f.(h, acc), f)
5    end
6  end
7
8  [1, 2, 3, 4, 5] |>
9  MyReduce.reduce(0, fn(x, acc) -> x + acc end)
10 |> IO.inspect
```

```
% elixir my_red.exs
15
```

Map-Reduce

Map-Redux

```elixir
defmodule MapReduce do
  def reduce([], acc, _), do: acc
  def reduce([h|t], acc, f) do
    reduce(t, f.(h, acc), f)
  end

  def map([], _), do: []
  def map(l, f) do
    reduce(l, [], fn(x, acc) -> [f.(x) | acc] end)
    |> Enum.reverse
  end
end

[1, 2, 3, 4, 5]
|> MapReduce.map(fn(x) -> x * 2 end)
|> IO.inspect
|> MapReduce.reduce(0, fn(x, acc) -> acc + x end)
|> IO.inspect
```

Map-Reduce

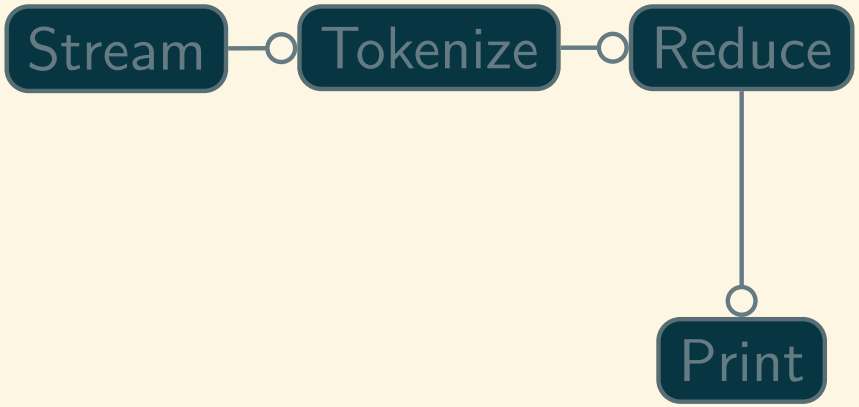
Map-Redux

```
% elixir my_map_red.exs
[2, 4, 6, 8, 10]
30
```

Word Counting

- Stream lines out of a file
- Tokenize into words
- Reduce words and print results

Word Counting



Word Counting

Streaming Data

```
1  defp stream_file(filename) do
2    File.stream!(filename)
3  end
```

Word Counting

Tokenizing Words

```
1  defp tokenize_words(line) do
2    line |> Stream.flat_map(&String.split/1)
3  end
```

Word Counting

Reducing Words

```
1  defp reduce_words(words) when is_list(words) do
2    Enum.reduce(words, %{}, &update_count/2)
3  end
4
5  defp update_count(word, acc) do
6    Map.update(acc, word, 1, &(&1 + 1))
7  end
```

Word Counting

Counting Words

```
1  def count_words(filename) do
2    stream_file(filename)
3    |> tokenize_words
4    |> Enum.to_list
5    |> reduce_words
6  end
```

Word Counting

Results

```
% elixir wc.exs
%{"Kometevskyite?\"" => 1, "silent" => 1, "cease" => 1, "along,"
    => 1,
  "Refund\"" => 1, "prepare)" => 1, "black" => 4, "shrunken" =>
    1, "Man" => 1,
  "payments" => 3, "Redistribution" => 1, "Gutenberg:" => 1, "
    SEND" => 1,
  "first" => 3, "accepted" => 1, "whole" => 9, "happened.\"" =>
    1,
  "casually," => 1, "forward." => 2, "thoroughly" => 1, ...}
```

Improve Word Counting

```
1  def count_words(filename) do
2    stream_file(filename)
3    |> tokenize_words
4    |> Stream.map(&String.downcase/1)
5    |> Enum.to_list
6    |> reduce_words
7  end
```

Parallel Map

- Spawn a process for each element in the list
- Evaluate the provided function for the element
- Gather and Return results

Parallel Map

```elixir
defmodule MyMap do
  def pmap(collection, f) do
    collection |>
    Enum.map(&(Task.async(fn -> f.(&1) end))) |>
    Enum.map(&Task.await/1)
  end
end

MyMap.pmap(1..10_000, &(&1 * &1)) |> IO.inspect
```

Parallel Map

```
% /usr/bin/time elixir pmap.exs
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144, 169, 196, 225,
    256, 289, 324,
 361, 400, 441, 484, 529, 576, 625, 676, 729, 784, 841, 900, 961,
     1024, 1089,
 1156, 1225, 1296, 1369, 1444, 1521, 1600, 1681, 1764, 1849,
    1936, 2025, 2116,
 2209, 2304, 2401, 2500, ...]
0.43user 0.06system 0:00.39elapsed 127%CPU (0avgtext+0avgdata
    53092maxresident)k
6672inputs+8outputs (1major+12587minor)pagefaults 0swaps
```

Echo Server

Echo Server

- Handle connections from clients (`netcat`)

Echo Server

- Handle connections from clients (`netcat`)
- Echo back contents of clients messages

`mix`: Ladle of Elixir|>

- Creates Skeleton Projects

`mix`: Ladle of Elixir|>

- Creates Skeleton Projects
- Compiles Code

`mix`: Ladle of Elixir|>

- Creates Skeleton Projects
- Compiles Code
- Runs Test Suites

`mix`: Ladle of Elixir|>

- Creates Skeleton Projects
- Compiles Code
- Runs Test Suites
- Creates Releases

`mix`: Ladle of Elixir|>

- Creates Skeleton Projects
- Compiles Code
- Runs Test Suites
- Creates Releases
- Anything you want

mix Output

```
% mix new echo_server
...
% find ./echo_server -type f
./echo_server/mix.exs
./echo_server/config/config.exs
./echo_server/.gitignore
./echo_server/lib/echo_server.ex
./echo_server/test/echo_server_test.exs
./echo_server/test/test_helper.exs
./echo_server/README.md
```

Elixir|> Applications

```elixir
def application do
  [mod: {EchoServer, []},
   applications: [:logger]]
end
```

mix.exs

Elixir|> Applications

```elixir
defmodule EchoServer do
  use Application

  def start(_, _) do
    EchoServer.Supervisor.start_link
  end

end
```

lib/echo_server.ex

Elixir|> Supervisors

```elixir
defmodule EchoServer.Supervisor do
  use Supervisor

  def start_link do
    Supervisor.start_link(__MODULE__, [], name: __MODULE__)
  end

  def init(_) do
    children = [
      supervisor(Task.Supervisor, [[name: EchoServer.TaskSupervisor]]),
      worker(Task, [EchoServer.Echo, :accept, [1337]])
    ]

    opts = [strategy: :one_for_one]
    supervise(children, opts)
  end

end
```

lib/echo_server/supervisor.ex

Echo Server: Echo

```elixir
def accept(port) do
  {:ok, socket} = :gen_tcp.listen(
    port,
    [:binary, packet: :line, active: false, reuseaddr: true])
  loop_acceptor(socket)
end

defp loop_acceptor(socket) do
  {:ok, client} = :gen_tcp.accept(socket)
  {:ok, pid} = Task.Supervisor.start_child(
    EchoServer.TaskSupervisor, fn -> serve(client) end)
  :ok = :gen_tcp.controlling_process(client, pid)
  loop_acceptor(socket)
end
```

lib/echo_server/echo.ex

Echo Server: Echo

```elixir
defp serve(socket) do
  socket
    |> read_line
    |> (fn(x) -> "> " <> x end).()
    |> write_line(socket)

  serve(socket)
end

defp read_line(socket) do
  {:ok, data} = :gen_tcp.recv(socket, 0)
  data
end

defp write_line(line, socket) do
  :gen_tcp.send(socket, line)
end
```

lib/echo_server/echo.ex

Echo Server

```
% iex -S mix
...
iex(1)> hd Application.
    started_applications
{:echo_server,
 'echo_server',
 '0.0.1'}
```

```
% nc localhost 1337
this is a test
> this is a test
I'm testing the echo-ability
> I'm testing the echo-ability
```

Elixir|> Resources

Elixir|> Resources

- http://elixir-lang.org/
  - http://elixir-lang.org/getting-started/introduction.html

Elixir|> Resources

- `http://elixir-lang.org/`
  - `http://elixir-lang.org/getting-started/introduction.html`
- Learn X in Y minutes [2]

Elixir|> Resources

- `http://elixir-lang.org/`
  - `http://elixir-lang.org/getting-started/introduction.html`
- Learn X in Y minutes [2]
- Programming Elixir [7]
- Metaprogramming Elixir [4]

References I


Elixir project.
`http://elixir-lang.org/`.


Learn x in y minutes, where $x =$ elixir.
`https://learnxinyminutes.com/docs/elixir/`.


Ten reasons not to use a functional programming language.
`http://tinyurl.com/c32k2jl`.


Chris McChord.
Metaprogramming Elixir.
Pragmatic Bookshelf, 2015.


Chris McChord, Bruce Tate, and José Valim.
Programming Elixir.
Pragmatic Bookshelf, 2016.

References II


Randall Munroe.
Functional.
https://xkcd.com/1270.


Dave Thomas.
Programming Elixir.
Pragmatic Bookshelf, 2014.

Learning Functional Programming with Elixir|>

A Short Guide Through Functional Programming

Kenny Ballou

zData, Inc.

March 19, 2016