# Intro to **Elixir**

## Functional Programming Language

Mark Shelton | 17 November 2017

# Prerequisites

**Install elixir (& erlang)**

elixir-lang.github.io/install

**Install git client**

git-scm.com/downloads

# Setup

```
git clone https://github.com/fp-uwa/intro-elixir.git
cd intro-elixir
elixir problems/<filename>.exs
```

# This Seminar

Introduction to Elixir

History of Elixir & Erlang

Comparisons to other languages

Basics: Functions, Data Structures

Piping, Patterns, Captures, Streams

# Intro to Elixir

Functional

Dynamic & strongly typed

Built on top of the Erlang Virtual Machine (BEAM)

Scalable, fault-tolerant

# History of Elixir

Erlang: Ericsson in 1986

- Everything is a process
- Processes are isolated

- Elixir: 2011, modern Erlang (nicer syntax, tooling etc.)

# Comparisons

Why **Elixir**? Concurrency.

- vs. **Haskell** – Static Typing
- vs. **Clojure** – Lisp, Java
- vs. **Go** – Imperative, Google

# Basics: Execution

**Compilation Mode:**

**elixirc** <filename>**.ex** #compile file

→ Elixir.<module>**.beam** #erlang

**iex>** <module>.<func>(..) #invoke

**Script Mode:**

**elixir** <filename>**.exs** #run script

# Basics: Modules & Functions

```
#math.exs
defmodule Math do
      #Named function
      def sum(a, b) do a + b end
end
#Anonymous function
sum = fn a,b -> a + b end #invoked as sum.(a,b)
```

# Basics: Data Structures

**Atoms (aka symbols):**

```
:hello => :hello
```

**Strings vs. charlists**

```
"hełło" => "hełło“
'hełło' => [104, 101, 322, 322, 111]
```

# Basics: Data Structures cont.

**(Linked) Lists:** `list = [1,2]`

```
[0] ++ list => [0,1,2] #fast
list ++ [3] => [1,2,3] #slow
```

**Tuples:** `tuple = {5,6}`

```
elem(tuple, 1) => 6 #fast
put_elem(tuple,2,7) => {5,6,7} #slow
```

# Basics: Data Structures cont.

**Keyword Lists:**

```
list = [{:a, 1},{:b, 2}, {:a, 3}]
```

Used for options or when you need to preserve use ordering… For everything else, there are maps.

**Maps:** `map = %{:a => 1,2 => b}`

# Interesting features

- Piping & Captures
- Pattern matching
- Streams & Lazy Evaluation

We'll learn these through the following examples.

# **Example:** nucleotide_count

Given a DNA string, compute how many times each nucleotide occurs in the string. DNA is represented by an alphabet of the following nucleotides: 'A', 'C', 'G','T'.

```
open ./problems/nucleotide_count.exs

elixir problems/nucleotide_count.exs

elixir tests/nucleotide_count_test.exs
```

# **Example:** secret_handshake

Given a decimal number, convert it to binary and then to the appropriate sequence of events for a secret handshake.

```
open ./problems/secret_handshake.exs

elixir problems/secret_handshake.exs

elixir tests/secret_handshake_test.exs
```

# Example: batsmen

Read in batsmen from a file and convert them into a list of objects with initials, surnames, runs, and averages.

Round averages to the nearest integer, sort batsmen in desc order by total runs and filter for surnames that start with C.

```
open ./problems/batsmen.exs
elixir problems/batsmen.exs
```