Trading Bots and Elixir

Who am I?

- Theo Bohnen
- Driven Alliance
- @tbohnenjnr

Why Elixir?

Some of the basics that I like

- String
- Atoms
- Lists
- Tuples
- Maps
- Piping
- Pattern Matching

Strings

```
iex(67)> <<60,70,90>>
"<FZ"
```

```
iex(71)> "let's " <> "join"
"let's join"
```

Strings

```
iex(72)> "let's join" <> <<0>>
<<108, 101, 116, 39, 115, 32, 106, 111, 105, 110, 0>>
```

Lists

```
iex(18)> list_example = [123,123,123]
'{{{'
```

Lists

```
iex(19)> i list_example
Term
  '{{{'
Data type
  List
Description
  This is a list of integers that is printed as a sequence of characters
  delimited by single quotes because all the integers in it represent valid
  ASCII characters. Conventionally, such lists of integers are referred to as
  "char lists" (more precisely, a char list is a list of Unicode codepoints,
  and ASCII is a subset of Unicode).
Raw representation
  [123, 123, 123]
Reference modules
  List
```

Lists

```
iex(20)> char_list_example = 'asdf'
'asdf'
iex(21)> i char_list_example
Term
  'asdf'
Data type
  List
Description
  This is a list of integers that is printed as a sequence of characters
  delimited by single quotes because all the integers in it represent valid
  ASCII characters. Conventionally, such lists of integers are referred to as
  "char lists" (more precisely, a char list is a list of Unicode codepoints,
  and ASCII is a subset of Unicode).
Raw representation
  [97, 115, 100, 102]
Reference modules
  List
```

Atoms

```
iex(89)> i :this_is_an_atom
Term
  :this_is_an_atom
Data type
  Atom
Reference modules
  Atom
```

Tuples

```
iex(17)> tuple_example = {:ok, [123]}
{:ok, '{'}
```

```
iex(38)> {:ok, val} = {:ok, 123}
{:ok, 123}
iex(39)> val
123
```

Maps

```
iex(76)> map_example = %{name: "Theo"}
%{name: "Theo"}
iex(77)> map_example.name
"Theo"
```

Maps

```
iex(78)> map_example = %{Name: "Theo"}
%{Name: "Theo"}
iex(79)> map_example.Name
** (CompileError) iex:79: invalid alias: "map_example.Name". If you wanted to define an alias, an alias must expand to an atom at compile time but it d
id not, you may use Module.concat/2 to build it at runtime. If instead you wanted to invoke a function or access a field, wrap the function or field na
me in double quotes

iex(79)> map_example."Name"
"Theo"
```

Maps

```
iex(86)> map_example = %{map_example | :name => "Theodore"}
%{name: "Theodore", surname: "Bohnen"}
```

Piping

```
iex(98)> import Enum
nil
iex(99)> [1,2,3] |> sum
6
```

Piping

```
iex(105)> [1,2,3] |> Enum.map(&(&1 * 2)) |> IO.inspect
[2, 4, 6]
```

```
iex(114)> [1,2,3] |> Enum.map(fn i -> i * 2 end) |> IO.inspect
[2, 4, 6]
```

Matching

```
iex(106)> match_example = "a"
"a"
iex(107)> "a" = match_example
"a"
iex(108)> "b" = match_example
** (MatchError) no match of right hand side value: "a"
```

Pattern Matching

```
iex(109)> defmodule PatternMatchingExample do
...(109)>   def test_pattern_matching() do
...(109)>     {:ok, "Return value"}
...(109)>   end
...(109)> end
```

```
iex(111)> {:ok, val} = PatternMatchingExample.test_pattern_matching
{:ok, "Return value"}
iex(112)> val
"Return value"
```

MIX

```
theobohnen in /tmp/hotcode λ mix help
mix                   # Runs the default task (current: "mix run")
mix app.start         # Starts all registered apps
mix archive           # Lists all archives
mix archive.build     # Archives this project into a .ez file
mix archive.install   # Installs an archive locally
mix archive.uninstall # Uninstalls archives
mix clean             # Deletes generated application files
mix cmd               # Executes the given command
mix compile           # Compiles source files
mix deps              # Lists dependencies and their status
mix deps.clean        # Deletes the given dependencies' files
mix deps.compile      # Compiles dependencies
mix deps.get          # Gets all out of date dependencies
mix deps.unlock       # Unlocks the given dependencies
mix deps.update       # Updates the given dependencies
mix do                # Executes the tasks separated by comma
mix escript.build     # Builds an escript for the project
mix help              # Prints help information for tasks
mix hex               # Prints Hex help information
mix hex.build         # Builds a new package version locally
mix hex.config        # Reads or updates Hex config
mix hex.docs          # Publishes docs for package
mix hex.info          # Prints Hex information
mix hex.key           # Hex API key tasks
mix hex.outdated      # Shows outdated Hex deps for the current project
mix hex.owner         # Hex package ownership tasks
mix hex.publish       # Publishes a new package version
mix hex.registry      # Hex registry tasks
mix hex.search        # Searches for package names
mix hex.user          # Hex user tasks
mix loadconfig        # Loads and persists the given configuration
mix local             # Lists local tasks
mix local.hex         # Installs Hex locally
mix local.public_keys # Manages public keys
mix local.rebar       # Installs rebar locally
mix new               # Creates a new Elixir project
mix profile.fprof     # Profiles the given file or expression with fprof
mix run               # Runs the given file or expression
mix test              # Runs a project's tests
iex -S mix            # Starts IEx and run the default task
```

OTP

- Open Telecommunications platform

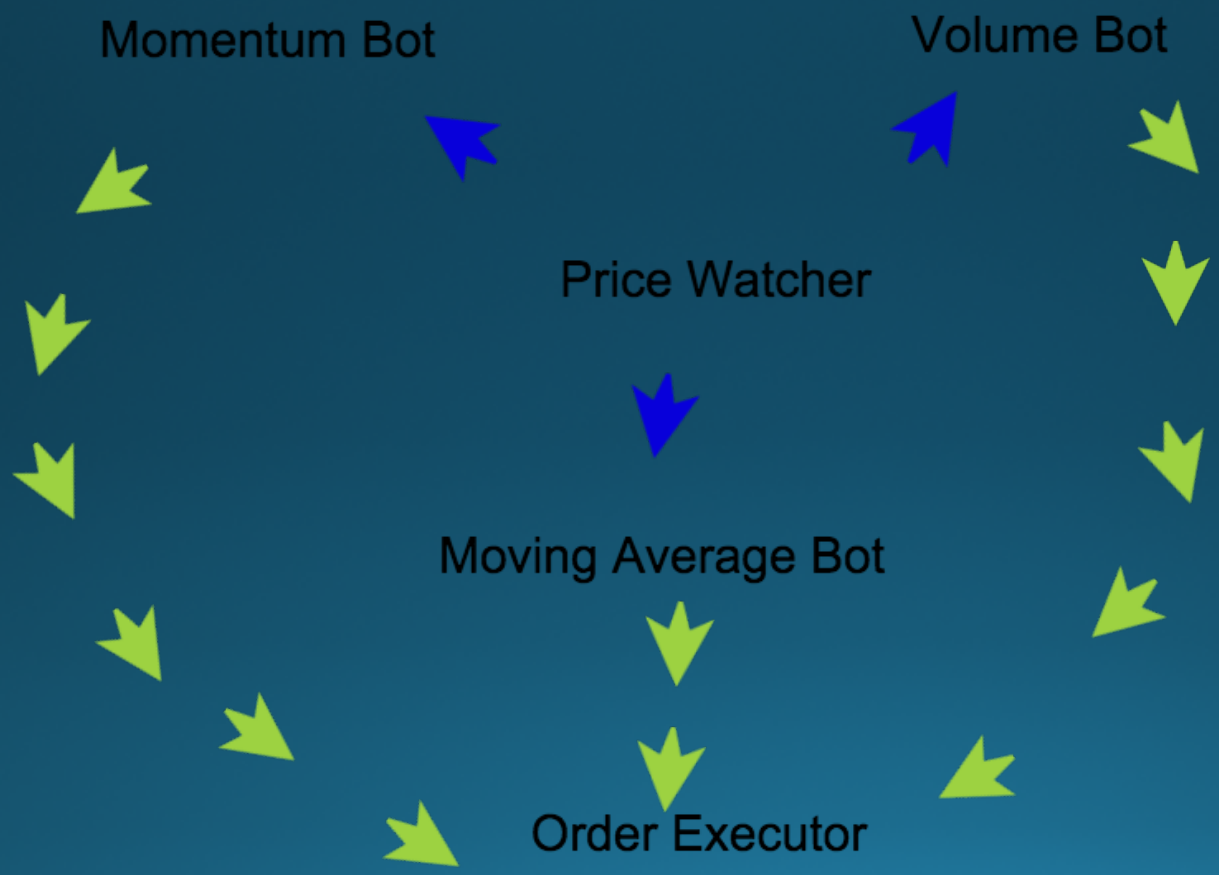
Gen_Server

Supervisor

- Failing Fast
- Processes and computation are cheap and can be restarted, but data is sacred.
- Strategies
  - :one_for_one
  - :one_for_all
  - :rest_for_one
  - :simple_one_for_one

Hot Code Swapping

Trading Bot Example



Final Thoughts

References

- https://tkowal.wordpress.com/2015/10/20/failing-fast-and-slow-in-erlang-and-elixir