Onboarding New Elixir Members

Do not assume they are Elixir enthusiasts.

- Do not ask people to study during nights and weekends just because you did.
- Do not assume they are thrilled to have the opportunity to work with Elixir just becaue you are.

Help them see the benefits.

- Concurrent tests.
- Server response times.
- Making a web request inside of a web request.
- Having most of Redis and a tiny bit of Kafka at your fingertips without any external dependencies.
- Processes have their own GC.
- One process trying to use 100% CPU does not really slow other things down.
- Productivity

Relate concepts to languages they are already familiar with

- A module serves the same code organization use-case as a class.
- An OO `User` object that had a bunch of methods would ideally be replaced by a `%User{}` struct with some functions for building the struct from params or transforming it into a different format
- A `GenServer` is not a code organization tool. It can hold state and behavior just like an Object but you should only use it when you can not achieve something with pure functions.

Acknowledge gotchas

- Elixir config solves config in a really robust and unique way, but it is different from how it was done in other languages, so people will be confused by it. Provide help and review.
- If this is your first compiled language, you are going to get some new benefits but also some new problems. If you put config in the wrong place, you could read an environment variable on the machine that compiles the code and hard-code that env var into the binary 😬.
- Charlists are weird and confusing, put double quotes on strings.

Things to set up before they even start.

- Ideally, let them work on a small part of something that is already in production.
- If it is a new thing, get it deployed before they start or pair with them until it is deployed.
- If any higher level OTP tools need to be used GenServers, ETS, Agents, Phoenix PubSub, etc, make those choices for them in the beginning.
- Help choose libraries in the beginning, what do you look for on hex.pm?

Things to set up before they even start.

- Document how to get into a production console.
- Things to run in CI:
  - `mix test`
  - `mix format --check-formatted`
  - `mix credo --strict`
  - `mix hex.audit`

Things they can ignore in the beginning.

- GenServers
- Processes
- Recursive Functions
- Concurrency
- ETS
- OTP
- Agents
- Supervisors
- Behaviors
- Protocols
- Macros

Things they need to start right away.

- Pattern Matching values out of a Map
- Read all the errors carefully, with the attitude that (I am going to be great at understanding these) understanding the errors in any language is a huge help.
- How to create a Module with naming conventions
- Named functions
- Anonymous functions

Things they need to start right away.

- How you call named functions, always on the module, you don't do `"dewet".upcase` but instead `String.upcase("dewet")`. Coming from OO, it is as if everything is a class method.
- How to use the `List`, `Map`, `String` and `Enum` modules.
- Pattern match on function heads instead of using if statements. This is not a good candidate for a group decision with newcomers.
- How to create Structs, and pattern match on them.
- Writing tests with ExUnit.

Coding style with newcomers in mind.

- Break the work into smaller chunks than what is normally comfortable or efficient.
- Create clean interfaces to the things they don't need to learn in the beginning. Things like GenServers.
- Follow your own testing best practices very strictly.
- Don't use recursion if a pipeline of `Enum` functions will do.

Coding style with newcomers in mind.

```
defmodule Doubler do
  def double(collection) do
    collection
    |> Enum.filter(&is_integer/1)
    |> Enum.map(&(&1 * 2))
  end
end

Doubler.double([1, 2, :kaboom, 3])

[2, 4, 6]
```

Coding style with newcomers in mind.

```
defmodule Doubler do
  def double([]), do: []

  def double([head | tail]) when is_integer(head) do
    [head * 2 | double(tail)]
  end

  def double([_ | tail]) do
    double(tail)
  end
end

Doubler.double([1, 2, :kaboom, 3])

[2, 4, 6]
```

Learning Resources not to start with

- Programming Elixir - Dave Thomas
- Elixir in Action - Saša Jurić

Learning Resourcs to start with

- https://elixir-lang.org/getting-started **(First 15 sections)**
- **Learn Functional Programming with Elixir - Ulisses Almeida**
- **Pragmatic Studio Elixir & OTP**
- https://learn-elixir.dev **(Free Preview)**

Go onboard people with excellence