SHORTER FEEDBACK LOOPS WITH LIVEBOOK

Linus De Meyere

github.com/linusdm/livebook_fosdem
WHO KNOWS LIVEBOOK?
GOALS

- Introduction to Livebook
- How to get Livebook
- Livebook stories from the trenches
- Starting in the middle with Livebook
Welcome to Livebook
Get to know Livebook, see how it works, and learn its features.

Distributed portals with Elixir
A fast-paced introduction to Elixir by building distributed data-transfer portals.

Elixir and Livebook
Learn how to use some of their unique features together.

RUNNING SESSIONS (1)

My first Livebook
/Users/linus/dev/livebook_fosdem/start.liveemd
0 MB  Created 3 minutes ago
My first Livebook

```elixir
Mix.install[
  [:kino, "@ 0.8.1"]
]
```

Some title

```elixir
IO.puts("Hello from Livebook!")
```

```
Hello from Livebook!
```

```
:ok
```

Any markdown goes here
Learn

Check out a number of examples showcasing various parts of the Elixir ecosystem. Click on any notebook you like and start playing around with it!

Welcome to Livebook
Get to know Livebook, see how it works, and learn its features.

- Distributed portals with Elixir
  A fast-paced introduction to Elixir by building distributed data-transfer portals.

- Elixir and Livebook
  Learn how to use some of their unique features together.

- Introduction to Kino
  Make your notebooks interactive with inputs, controls, and more.

- Plotting with VegaLite
  Learn how to quickly create numerous plots for your data.

- Maps with MapLibre
  Seamlessly plot maps using geospatial and tabular data.

Deep dive into Kino
5 notebooks
IT'S JUST MARKDOWN
Welcome to Livebook

Basic usage

Livebook is a tool for crafting interactive and collaborative code notebooks.

Each notebook consists of a number of cells, which serve as primary building blocks. There are Markdown cells (such as this one) that allow you to describe your work and Code cells to run your Elixir code!

To insert a new cell move your cursor between cells and click one of the revealed buttons.

```elixir
# This is a Code cell - as the name suggests that's where the code goes.
# To evaluate this cell, you can either press the "Evaluate" button above
# or use `Ctrl + Enter` (or `Cmd + Enter` on a Mac).
message = "hey, grab yourself a cup of 🌡️"

String.replace(message, "☕️", "🍵")
```

Subsequent cells have access to the bindings you've defined:

```elixir
String.replace(message, "☕️", "🍵")
```

Note however that bindings are not global, so each cell sees only stuff that goes above itself. This approach helps to keep the notebook clean and predictable as you keep working on it!

Sections
# Welcome to Livebook

## Basic usage

Livebook is a tool for crafting **interactive** and **collaborative** code notebooks.

Each notebook consists of a number of cells, which serve as primary building blocks.

There are **Markdown** cells (such as this one) that allow you to describe your work and **Code** cells to run your Elixir code!

To insert a new cell move your cursor between cells and click one of the revealed buttons. ↓

```
```elixir
# This is a Code cell – as the name suggests that's where the code goes.
# To evaluate this cell, you can either press the "Evaluate" button above
# or use 'Ctrl + Enter' (or Cmd + Enter on a Mac)!
message = "hey, grab yourself a cup of ☕"
```

Subsequent cells have access to the bindings you've defined:

```
```elixir
String.replace(message, "☕", "🫐")
```

Note however that bindings are not global, so each cell sees only stuff that goes above itself. This approach helps to keep the notebook clean and predictable as you keep working on it!

## Sections

You can leverage so called **sections** to nicely group related cells together.

Click on the "Book" icon (<i class="ri-livebook-sections"></i>) in the sidebar
- Automatically save to filesystem
- Autocompletion
- Inline documentation
- Code formatting
REPRODUCIBILITY

- No global mutable state
- Sequential model for running code cells
- Efficient change tracking - stale cells
- Branching sections
- Package management in the notebook itself
ERLANG VM PROCESSES AND DISTRIBUTION EVERYWHERE

Clients

Browser #1

Browser #2

WebSockets

Livebook

Erlang distribution

Runtime

Code

Smart cell

Output #1

Output #2

WebSockets

Erlang distribution

Erlang distribution

Erlang distribution

Erlang distribution
INSTALLATION

- **Desktop application** (Windows and Mac)
- Escript
- Docker image
- In the cloud - somewhere remote

```
mix escript.install hex livebook

docker run -p 8080:8080 -p 8081:8081 --pull always livebook/liveb```


run the desktop app, or

livebook server
livebook server new
livebook server path/to/directory/
livebook server path/to/some.livemd
livebook server https://example.com/some_public.livemd
livebook server --help

^ lots of startup/deployment options here
“Start with the riskiest parts of your development”

Every project manager
BENEFITS

• Start in the middle
• Increase transparency
• Document the process
• Livebooks as shareable deliverables
• Lower the barriers to entry (also for non-coders)
CONTEXT

• Small software shop doing custom development
• Many projects at the same time
• Small teams (teams of two)
• Important to have good DX
• Good documentation really helps
• Communication with clients is key
CASE #1

EXPLORING AN UNDOCUMENTED LEGACY API

- Low level TCP protocol
- Use `gen_tcp` to send and receive messages
- Stub out the server for end-to-end scenarios
- Great for documentation purposes (no meta info available)
- Collaborate and create a shared understanding of the system
Documenting the various messages

Login

Log the user in and get a reference that is used to authenticate subsequent requests.

**input** [client code, username, password]

Successful login

```
1  {:ok, ["10", _message | _]} = Client.send_request_and_receive_response(54, credentials, endpoint)
```

```
{:ok, 
 [ 
   "10",
   "Login OK",
],
```

^ pattern matching is awesome 💜
CASE #1

EXPLORING AN UNDOCUMENTED LEGACY API

- Reproduce bugs (can be referenced in github issues)
- Facilitates discussions on the right level of abstraction
- Verify bugfixes, without having to integrate in a real application
- idea: record test fixtures (thanks Adam Lancaster)
- idea: generate template for documentation
Mix project integration

leans on Mix.install/2
TYPICAL LIFECYCLE WITH LIVEBOOK

1. Experiment with code in Livebook
2. Maybe add tests
3. Add $path$ dependency on local mix project
4. Promote reusable code to local mix project
Manipulating stub server responses
Managing secrets

^ don't keep sensitive data in your notebooks
CONCURRENT ETL PIPELINE

- CSV → Postgres
- Concurrent data processing in Livebook
- The power of processes is available
- Fun with Flow
- Using Ecto from Livebook
1. `ImportFlow.import_all_flow(model, update_fn: update_fn)`
2. `Flow.stream(link: false)`
3. `Stream.run()`
Case #3

Connecting to an online environment

- Remember, it's all erlang distribution behind the scenes
- You need your node's name/sname and cookie
- Great for one-off tasks
- Setup your first admin user
- Poke around your live system
- Implement features without a UI yet
- Remember, it's all live!
Change your runtime settings to "attached node"

Runtime settings

- Elixir standalone
- Attached node

Connect the session to an already running node and evaluate code in the context of that node. Thanks to this approach you can work with an arbitrary Elixir runtime. Make sure to give the node a name and a cookie, for example:

```
iex --sname test --cookie mycookie
```

Then enter the connection information below:

Name
my-app@45a0:b366:8af7:9dd1:3241:3e00:0e0d:539b

Cookie
mz5z5R2h1U6pDhMwkj4azrbwCf64FXpwHmSXSC1ov2ZOZfhEl+V6gChSRnr5nBX4

Connect
TESTING IN LIVEBOOK

- Doctests are executed automatically
- You can write regular ExUnit test cases
DBG() IN LIVEBOOK

- dbg( ) was recently added (elixir v1.14)
- Manipulate your pipeline live
OTHER RESOURCES

- **The DockYard Academy**: open source curriculum to help students learn Elixir
- **Project Bumblebee**: Neural Networks in Livebook (GPT2, Stable Diffusion, ... on your computer!)
NO NUMBATS HERE!