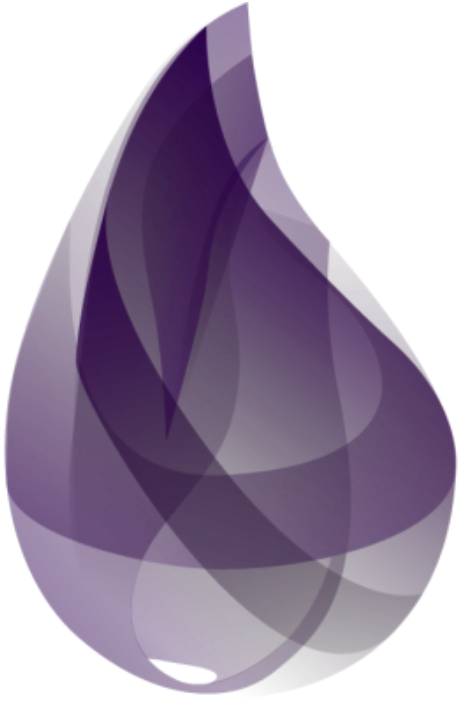


Real-time Drupal

or how I learned to stop worrying and love Elixir

Speaker notes

- Welcome to real-time Drupal
- or how I learned to stop worrying and love Elixir
- Really, why I think Elixir and Phoenix are technologies of great value
- The goal of this talk is to give you the tools to start to learn about Elixir and Phoenix

Hello My Name Is Frank

I am a Christian, Father, and Technology Enthusiast.

- *Online my name is* **frob** (IRC, d.o, github)
- *On Twitter I am* **@frobdfas**
- *My Blog is* www.frobiovox.com
- *I work for* Clarity Innovations Inc.

Speaker notes

- Hello my name is Frank Robert Anderson
- I am a Christian, Father, and Technology Enthusiast.
- Web development since high school
- Been working in Drupal professionally for just shy of 10 years.
- I have worked almost exclusively in the Education industry both higher-ed and k-12 for over 5 years.
- Online you can find me as frob
  - IRC
  - github
  - drupal.org
  - Unless there is a drupalgedon type security announcement, you will not find me on Drupal Slack. But I am squatting on the name Frob there too.
- Twitter frobdfas
- blog
- website
- I work for Clarity Innovations.


Clarity Innovations

Speaker notes

- Clarity Innovations is a professional services firm based in Portland, Oregon focused on providing K-12 and higher education technology consulting to non-profits, schools, and corporations.
- Our 25 employees combine leading edge technology and design skills with direct experience in the classroom and university.
- We inform product strategy, provides instructional design, develops training content, and engineers mobile and web applications that improves the process and practice of teaching and learning.

Disclosures

- I am not an Elixir Developer
- I do not run any Elixir Production Apps
- I am not an evangelist for any tech company or platform

Speaker notes

- @TODO put notes here .

Technology Enthusiast

Speaker notes

- I make it my job to know as much as I can about emerging technologies.
- I have many hobbies that have span many things.
  - Dnd
  - Game Mechanics
  - Used to record music
  - Still live sound (at church)
- Make things

Give back to the community

- this past year I reached my 9 year 12 month anniversary on d.o
- I have been a part of Drupal for a long time
- This is my third Drupal Con presentation.
  - last year I gave a talk on Voice UI
  - This year I am talking about Real-time web application
  - I like to give talks about things that I like
  - I like to give talks about things that I think are of value to the Drupal community
- The goal of this talk is to give you the tools to start to learn about Elixir and Phoenix

Some not-scientific-at-all™ tests

```
ab -n 1000 -c 100 http://www.a-static-site-i-hosted.com
```

vs

```
ab -n 1000 -c 100 http://127.0.0.1:4000/admin/content
```

Speaker notes

- In 2015 I wrote a blog post about Elixir performance
- Some not-scientific-at-all™ benchmarks showed that Phoenix, at that time was as almost as performant and scalable as a static site

Some not-scientific-at-all™ test results

Static

```
Requests per second:    182.15 [#/sec] (mean)
Time per request:       549.001 [ms] (mean)
Time per request:       5.490 [ms] (mean, across all concurr
```

Dynamic

```
Requests per second:    133.54 [#/sec] (mean)
Time per request:       748.844 [ms] (mean)
Time per request:       7.488 [ms] (mean, across all concurr
```

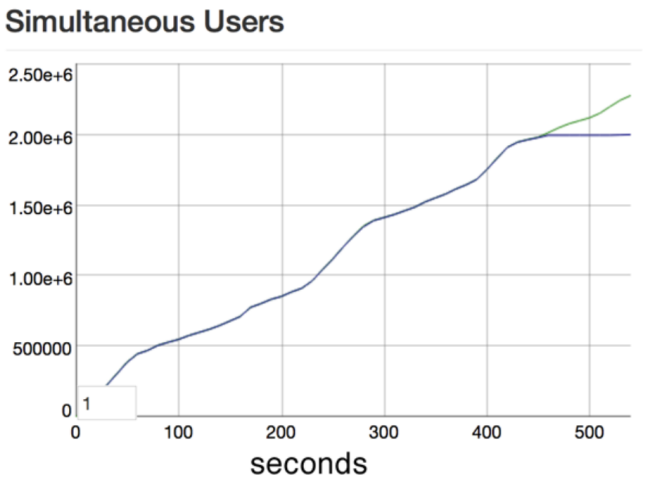
Speaker notes

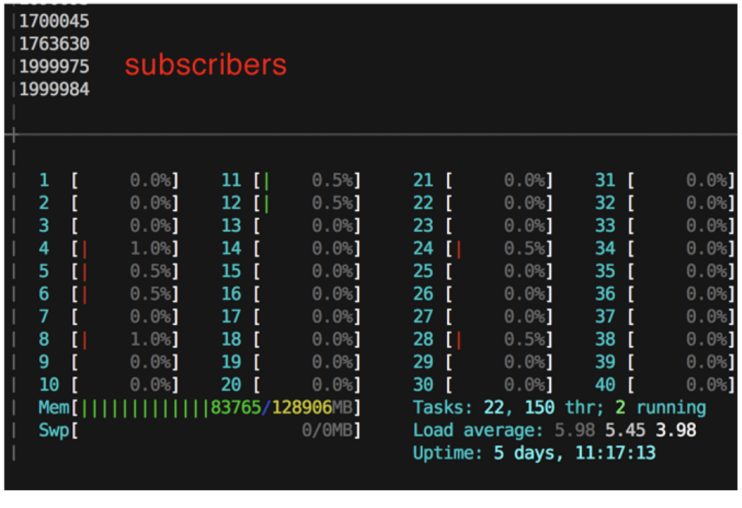
- In 2015 I wrote a blog post about Elixir performance
- Some not-scientific-at-all™ benchmarks showed that Phoenix, at that time was as almost as performant and scalable as a static site
- The next month something very interesting happened

2 Million Served

The Road to 2 Million Websocket Connections in Phoenix

By Gary Rennie · 2015-11-03 · v1.0.0





Speaker notes

- The phoenix team was able to simulate using 30-40 aws severs
- hitting one really beaffy server 2 million connections
- They could have pushed if further, but thought why?
- php has a problem
  - It can never do this
  - I am not one who likes to say something can never happen
  - what I am really saying is it would take a tremendouse effort to do anything remotely like this in php.
  - That is because of the fundemental way php works
  - it is designed around the request response lifecycle
  - it isn't horrible at long running things anymore, but it still isn't good.
  - Drupal (or anything based on Symphony HTTPKernel) are especially bound by this.

Foreshadowing

- How it works
- Technology
- Architecture
- Into to Elixir
- Conclusion

Speaker notes

- How it works
- Technology
- Architecture
- Into to Elixir
- Conclusion

How to get started

Speaker notes

- Phoenix

Phoenix

- MVC Framework with no "Models"
- Contextual Functionality separated into related apps

```
lib/
├── demo
│   ├── accounts
│   │   ├── accounts.ex
│   │   └── user.ex
└── demo_web
    └── user_live
        ├── edit.ex
        ├── index.ex
        ├── new.ex
        ├── presence_index.ex
        └── show.ex
```

Speaker notes

- Framework build on Elixir
- app vs app_web I seperate the frontend code from the web frontent
  - it is an MVC but not like most MVCs
    - They have removed the model (dumping ground) and streamlined the consept
- context I Dedicated modules that expose related functionallity

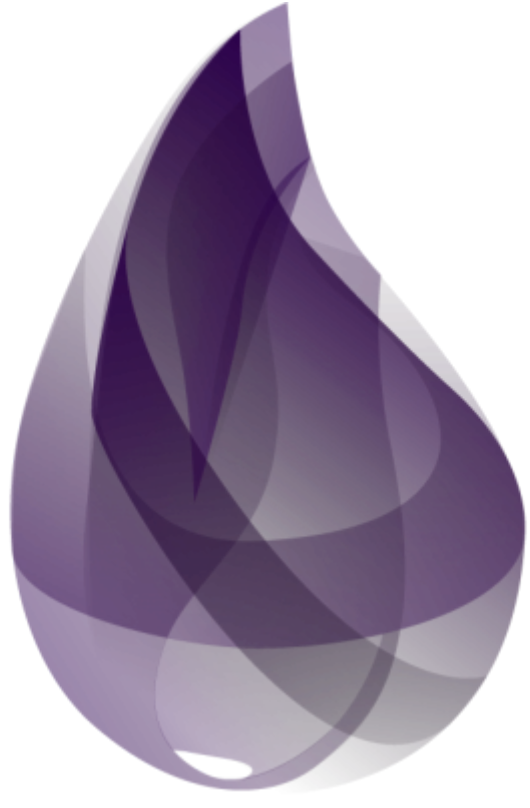
Phoenix Features

- Channels for websockets
- HTTP/2
- Useful Generators
- Live Reloading
- Front-end tool agnostic
- Built in tests

Speaker notes

- Phoenix feature walkthrough
  - Channels | websockets
    - The connection from client to server
  - HTTP/2 | push only, browsers don't support stateful connections
    - templates support pushing assets
  - generator | mix plugins
    - helps you understand what to do and where to put things
    - The generators are a type of self documentation
  - live reloading
  - built in test
    - this means that it is built with the idea that there will be tests and test drivin development
  - What is so special here? | https://youtu.be/bk3icU8iIto?t=680

Elixir



Speaker notes

- First off it has a similar logo, so it has to be good right?

BEAM

Speaker notes

- BEAM walkthrough

The ErlangVM

- Born from Erlang
- Created to solve the problem of communication at scale
- Phone networks cannot go down for maintenance

Speaker notes

- Erlang
  - 30 year old from Ericson
  - Erlang is good for communication at massive scale
    - Turns out that telephone switches are analagous to the web
    - https://youtu.be/bk3icU8iIto?t=779
  - Code can be deployed to a running server (code deltas)
    - The phones cannot be brought down for any reason

Processes and messages

Speaker notes

- Processes and messages
  - the vm works like an operating system
  - automatically distributes processes across cpu and cores
  - process is cheap (around 1k) and has own garbage collection
  - one process doesn't block another
  - If a process dies, that is okay it's isolated, you can bring it back,
    - Have you tried turning it off and on again
    - https://youtu.be/bk3icU8iIto?t=939

Concurrency

- async isn't concurrency
- WhatsApp 2 million
- Bleacher Report
  - from minutes to millisecond
  - from 150 servers to 30
  - from heavy cache to no cache

Speaker notes

- concurrency
  - async isn't concurancy
  - WhatsApp 2 million on one node less than 20 people
    - old numbers but it is hard to beat these numbers when this is already beating world wide sms traffic
  - Bleacher Report
    - from few minutes for a push
    - 150 aws instances with lots of caching
    - after 10-30ms response for push
    - 30 aws instances with no caching

Performance

- WhatsApp
- Bleacher Report
- Phoenix 2 millions concurrent connections on one beefy machine

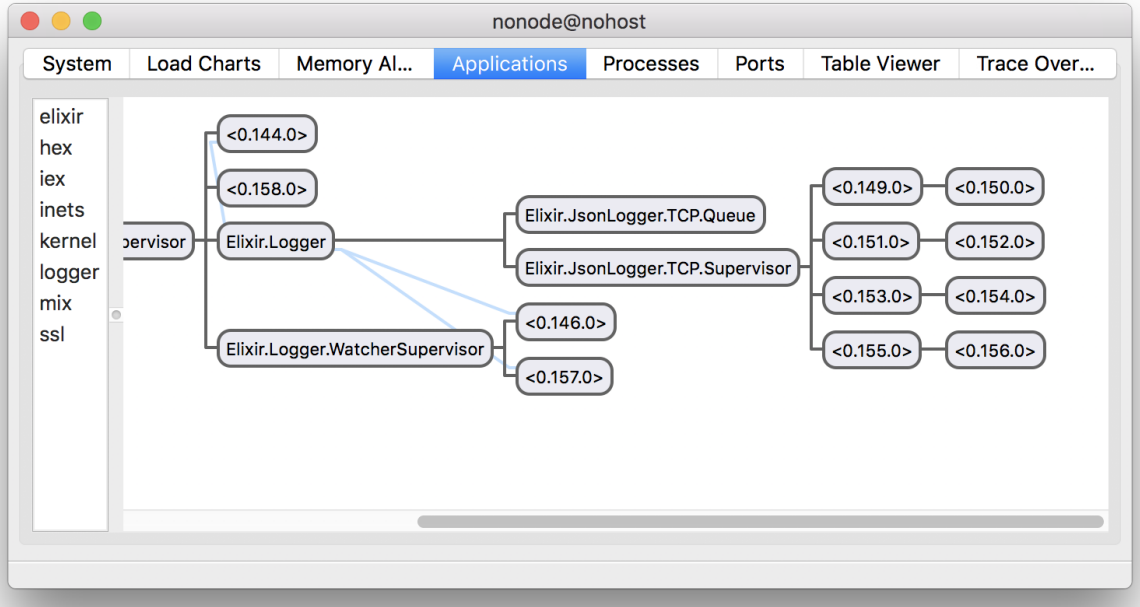
Speaker notes

- Performance
  - WhatsApp 2 mill was Erlang
  - Bleacher Report was Pure Elixir
  - Phoenix 2 mill on a heafty server 40 cores 128GB ram
    - As benchmarked with 40 machines opening connectsions at 40/s to send wikipedia articles to eachother

Other Cool Stuff

`remote_console`

`observer`



Speaker notes

- remote_console
- observer

What is Elixir good at

long running
soft real-time
high concurrency

Speaker notes

- What BEAM is good at
  - long running
  - soft real-time
  - high concurrency
- What BEAM is not good at
  - short running
  - true real-time

What is Elixir not good at

short running
true real-time

Speaker notes

- What BEAM is good at
  - long running
  - soft real-time
  - high concurrency
- What BEAM is not good at
  - short running
  - true real-time

Architecture for Drupal Integration

Speaker notes

- We need to bring Drupal and Elixir together
- pools for requests
  - you can easily exhaust your system resources if you do not limit the maximum number of concurrent processes that your program can spawn. Poolboy is a widely used lightweight, generic pooling library for Erlang that addresses this issue.
  - http://learningelixir.joekain.com/streaming-through-a-pool-in-elixir/
  - We can try using a pool of processes to handle URL unshortening in Domain Scraper. Previously, I used a queue of asynchronous tasks for this purpose. Using a pool differs from using a queue in that the queue can get blocked if one URL takes a very long time to unshorten.

Easy Integration

Speaker notes

- Easy - Drupal as Drupal, Phoenix as webserver/ESI
  - Drupal, PHP, Maria, Nginx/Apache for content management
  - Phoenix, Cowboy for application serving HTTP/S and WS

Easiest Integration

Speaker notes

- Easiest - Drupal as Drupal and Phoenix as Phoenix, include phoenix.js
  - Drupal, PHP, Maria, Nginx/Apache for HTTP/S
  - Phoenix, Postgres, Cowboy for WS

Hard Integration

Speaker notes

- Hard - Drupal as CGI, Phoenix as webserver/ESI
  - Drupal, PHP, Maria for content management com via FastCGI
  - Phoenix, Cowboy for app serving

Tips on how to speak Elixir

Introduction

Syntax

Features

Speaker notes

- Elixir is a language
- It is a language with a unique syntax
- It also has its own vocabulary
  - First will be a intro to the ecosystem
  - Then we will talk about the unique bits of syntax
  - Lastly we will discuss some of the more unique lanugadge features it has

How to be a functional programmer

- Grow beard
- long hair
- wear plaid

Speaker notes

- How to be a functional programmer
  - Grow Beard
  - Long hair, preferably in a bun
  - wear plaid

How to be a functional programmer

Speaker notes

- How to be a functional programmer
  - Grow Beard
  - Long hair, preferably in a bun
  - wear plaid
  - you need an advanced degree in mathematics or physics
- While I was looking for a Creative Commons picture of a hipster I found an image of a core Erlang dev.

Functional PHP

https://www.youtube.com/watch?v=M3_xnTK6-pA

Speaker notes

- Truth is functional programming is just programming
- True there are no objects in Elixir but that isn't what makes functional programming functional
- in all honesty you can program in php today in a functional paradime and it would just be considered good programming
- There is a really good talk on this subject from DrupalCon Austin

Introduction to the Elixir Ecosystem

Speaker notes

- These are some of the nicities of the elixir developer ecosystem

Elixir forum

https://elixirforum.com/

Speaker notes

- I will start with a link to the Elixir Forum

IEX

Speaker notes

- iex

Mix

```
mix --help
Mix is a build tool for Elixir

Usage: mix [task]

Examples:

    mix               - Invokes the default task (mix run) in
    mix new PATH      - Creates a new Elixir project at the gi
    mix help          - Lists all available tasks
    mix help TASK     - Prints documentation for a given task

The --help and --version flags can be given instead of a tas
```

- mix
  - https://elixir-lang.org/getting-started/mix-otp/introduction-to-mix.html
  - This is the built in build tool for elixir, liken to npm or composer
  - But with other generators built in
    - Think Drupal console and composer mixed together and fully supported as native parts of the language

Rebar

Speaker notes

- rebar
  - Kind of like mix, but for Erlang.
  - Technically Rebar3

Unit Tests

```
mix test
```

Speaker notes

- unit tests
  - ExUnit - built in unit test framework
  - Notice how simple this command is. Its because it is just another mix task

Hex Packages & Docs

Speaker notes

- Hex Packages
  - documentation
    - https://hexdocs.pm/
    - Hexdocs is a place for hex packages to host their documentation
    - This is a part of the language, it isn't a solution that was bolted on afterward with three other competing solutions.

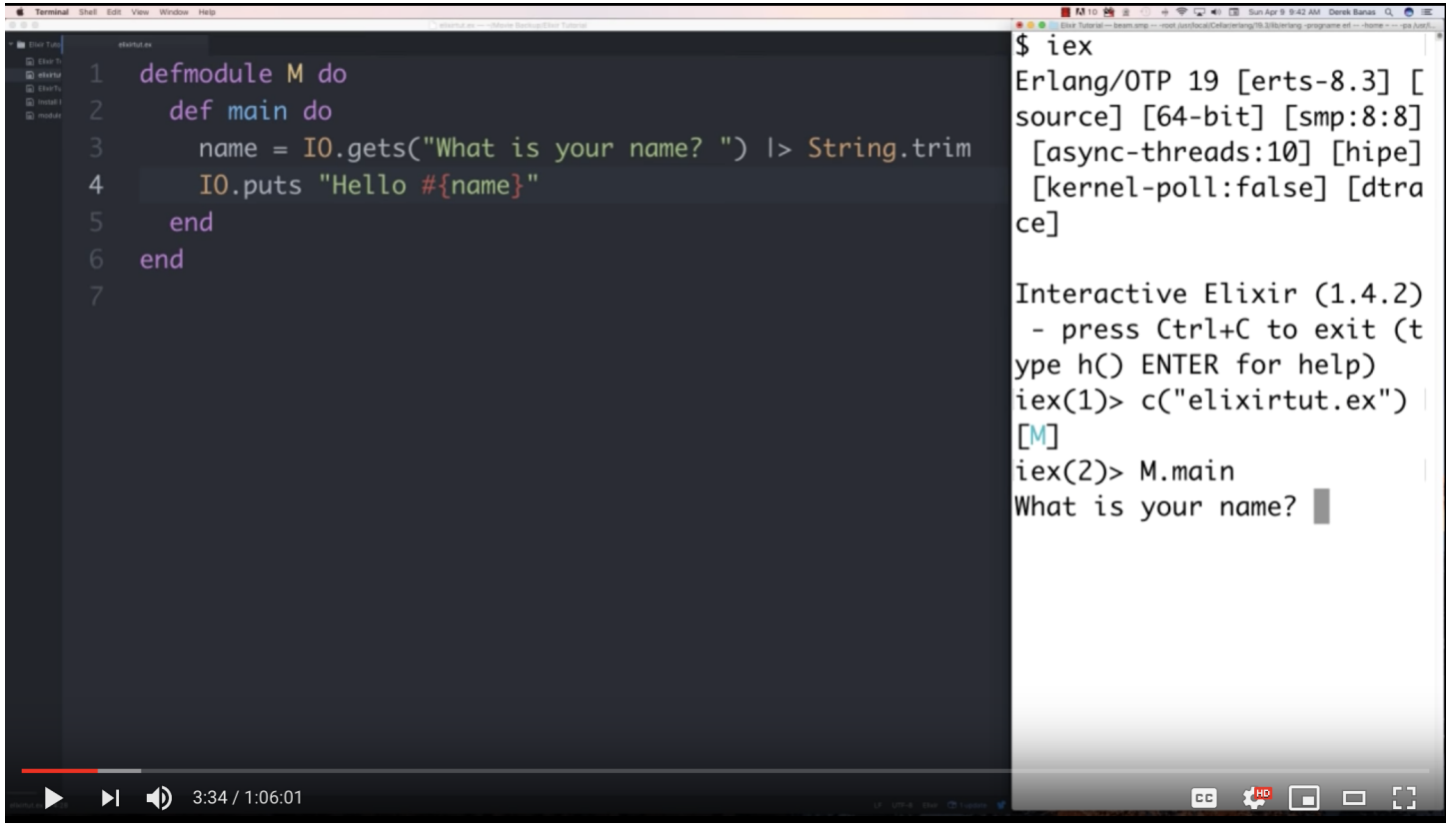
Syntax Overview

Speaker notes

- Now lets go into some of the more interesting sytax of elixir
- This is so you will be more familiar with it

This is what we will go over

Branching, Types, List, Tuples, Maps, Lists, Pattern Matching, Modules, Enumeration, Concurrency, Pipes



https://www.youtube.com/watch?v=pBNOavRoNL0

Speaker notes

- Branching, Types, List, Tuples, Maps, Lists, Pattern Matching, Modules, Enumeration, Concurrency, Pipes
- http://www.newthinktank.com/2017/04/learn-elixir-one-video/

Decision Making

Speaker notes

- 16:06 Decision Making

Comparison

Speaker notes

- 13:30 Comparison

Data is typed

```
7 == 7.0 => true
```

```
7 === 7.0 => false
```

Speaker notes

- 13:30 Comparison
  - Data is typed
    - int vs float
      - `7 == 7.0` => true
      - `7 === 7.0` => false

if/then and unless

```
if age >= 18 do
  "can vote"
else
  "cannot vote"
end
```

```
unless age !== 18 do
  "You are 18"
end
```

Speaker notes

- and the rest
  - "> < <= >="
  - and or NOT && ||
- if age >= 18 do
  "can vote"
  else "cannot vote" end
- unless age === 18 do "You are 18" else end

Case

```
case some_variable do
  1 -> "something"
  2 -> "something else"
  3 -> "something completely different"
end
```

Speaker notes

- switch like statements
  - case some_variable do 1 -> "something" 2 -> "something else" 3 -> "something completely different" end

Cond statement

```
cond do
  age >= 18 -> "can vote"
  age >= 16 -> "can drive"
  age >= 14 -> "cannot drive"
  true -> "default"
end
```

Speaker notes

- Cond is useful for checking across many different statments
- cond do age >= 18 -> "can vote" age >= 16 -> "can drive" age >= 14 -> "cannot drive" true -> "default" end

Ternary

```
message = if age > 18, do: "Can Vote", else: "Can't Vote"
```

Speaker notes

- Ternatry
  - No actual ternary ? : operator
  - http://learningwithjb.com/posts/elixir's-version-of-the-ternary-operator
  - Just an if/else on one line

Types

Speaker notes

- Talk briefly about the different native data types in elixir and how they are used.

Numbers

```
is_int(1) => true
```

```
is_float(1.0) => true
```

Speaker notes

- is_int
  - no maximum size
- is_float

Atoms

```
is_atom(:an_atom)
```

```
{is_atom(:Drupal)} => true
{is_atom(:"Drupal dot Org")} => true
```

Speaker notes

- is_atom
  - `Atom \#{is_atom(:Drupal)}`
  - `Atom \#{is_atom(:"Drupal dot Org")}`
- I don't expect the value of atoms to be apperent at first glance
- once I talk about pattern matching you will start to see how they are used
- We use atoms in php with the same syntax for placeholders

Ranges

```
one_to_ten = 1..10
```

Speaker notes

- ranges `one_to_ten 1..10`

Strings

```
string = "Hello World!"
```

```
String.length("Hello World!") => 12
```

```
"Hello" <> " " <> "World!"" => "Hello World!"
```

```
String.contains?("foobar", "foo") => true
```

Speaker notes

- 06:38 Strings
  - double quotes `string = "Hello World!"`
  - String Length `String.length("Hello World!")` => 12
  - String concatination `"Hello" <> " " <> "World!""`
  - String search `String.contains?("foobar", "foo")` => true
    - notice in this contains function there is a question mark
    - This is common when the expectation is a boolean return
    - kind of like using "is" like is_present
  - String first char `String.first("foobar")` => "f"
  - String char at index `String.at("foobar", 3)` => "b"
  - String substring `String.slice("foobar", 3, 5)` => "bar"
  - String explode `String.split("foobar", "b")` => ["foo", "oo"]
  - String revers `String.revers("foobar", "b")` => ["foo", "oo"]
  - String upcase `String.upcase("foobar", "b")` => ["foo", "oo"]

- String downcase `String.downcase("foobar", "b")` => ["foo", "oo"]
- String capitalization `String.capitalize("foobar", "b")` => ["foo", "oo"]

Math

```
div(5, 4) => 1
rem(5, 4) => 1
```

Speaker notes

-
  -
    -
      - /
- integer division
  - `div(5, 4)` => 1
  - `rem(5, 4)` => 1

Tuples & Maps

Speaker notes

- 2 or more values
- not for enumerating, cannot use as a list
- `is_tuple({1, 3.14, :Drupal})` => true
- append `new_tuple = Tuple.append(old_tuple, 66)`
- get element `el = elem(some_tuple, 2)`
- delete element `new_tuple = Tuple.delete_at(old_tuple, 2)`
- insert element `new_tuple = Tuple.insert_at(old_tuple, 2, "foobar")`
- get size `new_tuple = tuple_size(old_tuple)` => 3
- bunch of elements `eleven_nines = Tuple.duplicate(9, 11)`
- `{name, quest, color} = {"Robin", "To find the holy grail", "red, no blue"}`
- `IO.puts("favorite \#{color}")` => "red, no blue"

Tuples

```
is_tuple({1, 3.14, :Drupal})  => true
```

append

```
new_tuple = Tuple.append(old_tuple, 66)
```

get element

```
el = elem(some_tuple, 2)
```

get size

```
new_tuple = tuple_size(old_tuple) => 3
```

Speaker notes

- 2 or more values
- not for enumerating, cannot use as a list
- `is_tuple({1, 3.14, :Drupal})` => true
- append `new_tuple = Tuple.append(old_tuple, 66)`
- get element `el = elem(some_tuple, 2)`
- delete element `new_tuple = Tuple.delete_at(old_tuple, 2)`
- insert element `new_tuple = Tuple.insert_at(old_tuple, 2, "foobar")`
- get size `new_tuple = tuple_size(old_tuple)` => 3
- bunch of elements `eleven_nines = Tuple.duplicate(9, 11)`
- `{name, quest, color} = {"Robin", "To find the holy grail", "red, no blue"}`
- `IO.puts("favorite \#{color}")` => "red, no blue"

Maps

```
map = %{"key" => "value", "key2" => "value2"}
```

Speaker notes

- Like dictionaries
- `map = %{"key" => "value", "key2" => "value2"}`
- or an associaded array in php

Lists

- List.insert_at
- List.delete_at
- List.first
- List.last

Speaker notes

- a list is not an array
  - arrays do not exist in erlang or elixir
  - an array's speed comes from the way the memory is handled and that isn't a thing in the VM
- List.insert_at
- List.delete_at
- List.first
- List.last

List Concatniation and Subtraction

```
[1,2,3] ++ [4,5,6]
list3 = list1 -- list2
```

Speaker notes

- list concatination `[1,2,3] ++ [4,5,6]`
- list subtraction `list3 = list1 -- list2`

List with key value pair

```
list = [name: "Robin", quest: "To find the holy grail"]
```

Speaker notes

- search
  - if 4 in list3 do end
- A list can also be a key value pair with tuples.
  - `list = [{name:} "Robin", {quest:} "To find the holy grail"]`
  - remove the curly braces

List operations

```
list1 = [1,2,3]
[the_head | the_rest] = list1 => 1, [2, 3]
```

Speaker notes

- `[1,2,3]`
- first and everything lese
  - `[head | tail] = list1` => 1, [2, 3]
- this is super important when we do recursion

Enumeration

```
Enum.each ["words", "in", "list"], fn word ->
  IO.puts word
end
```

Speaker notes

- How to go through the lists
- Enumerate through list
  - ```
          IO.puts word
        end
    ```
    ```

Pattern Matching

```
foo = "bar"
```

Speaker notes

- 39:33 Pattern Matching
  - This might look like assignment but it isn't
  - In Elixir the = isn't about assignment event though that seems to happen

For Example

```
{a, b, c} = {:hello, "world", 42}
a == :hello => true
```

Speaker notes

- Assignment is about assigning a memory location to a value and that isn't what is happening
- the = is actually the match operator and it allows us to define matching rules
- For example
  - `foo = "bar"`
  - `{a, b, c} = {:hello, "world", 42}`
  - `a == :hello` => true
  - `{a, b, c} = {:hello, "world"}`
    - Causes a match error, The Tuples are of different lengths

Must actually match

```
{a, b, c} = {:hello, "world"}
```

Speaker notes

- `{a, b, c} = {:hello, "world"}`
  - Causes a match error, The Tuples are of different lengths

Notice anything?

```
list1 = [1,2,3]
[the_head | the_rest] = list1 => 1, [2, 3]
```

Speaker notes

- Do you notice anythin interesting about this from my last example?
- head and tail are pattern maching in list 1
  - head is matched against the first element
  - tial no matches the rest of the elements

Mod.fun/arity

Speaker notes

- Mod.fun/arity
  - module
    - no classes only module namespaces
  - 41:01 Anonymous Functions
    - normal syntax, should be familiar
      - `add_things = fn (a, b) -> a + b end`
    - short syntax
      - `add_things = &(&1 + &2)`
    - one function many definitions
      - ```
                {a, b} -> a + b
                {a, b, c} -> a + b + c
        end```
      -

```
defmodule Example do
  def main do
    IO.puts "Hello World!"
  end
end
```

Speaker notes

- Here is your hello world example module.

Mod.fun/arity

```
No function defined foo/2
```

This means there is no function defined with the name **foo** that takes two arguments.

Speaker notes

- functions are in the defined in the namespace with def
- arity is the number of arguments a function has
  - You will see error messages that say things like
    - No function defined foo/2
    - Meaning that somewhere foo(a, b) is getting called and it hasn't been defined
- this is really important, as we will soon see with recursion.

Same name different arguments

```
defmodule Greeter do
  def hello(name, place) do
    "Hello #{name}, from #{place}"
  end

  def hello(name) do
    "Hello #{name}"
  end
end
```

Speaker notes

- We can have two function definitions with two differnt numbers of argument
- Becuase the Arity is differnt the function's signature is differant and elixir knows what to execute based on the signature.

Multi-Clause Functions

```
defmodule Greeter do
  def hello(:jane), do: "Hello Jane"

  def hello(name) do
    "Hello #{name}"
  end
end
```

Speaker notes

- functions
  - https://blog.carbonfive.com/2017/10/19/pattern-matching-in-elixir-five-things-to-remember/
- Multi-Clause Functions
  - The function that is executed is the one where the arguments that are passed in match the pattern of the arguments in that function's signature.
  - How many people here are familiar with Drupal 7 and hook based development?
    - remember how we would do this?
      - `function example_login_form_form_alter($form, $form_state)`
    - Pattern matching makes this a part of the language in an interesting way.
      - https://youtu.be/gom6nEvtl3U?t=1159
      - you don't need to use switch or if as much for branching
    - So (hook alter example) becomes
      - `def hook_alter(form_id = 'login_form', form, _form_state)`
  - underscores in front of arguments are ignored by compiler

- default values with double backslash \

```
        IO.puts foo
      end```
```

APIs like this become possible

Speaker notes

- How many people here are familiar with Drupal 7 and hook based development?
  - remember how we would do this?
    - `function example_login_form_form_alter($form, $form_state)`
  - Pattern matching makes this a part of the language in an interesting way.
    - https://youtu.be/gom6nEvtl3U?t=1159
    - you don't need to use switch or if as much for branching
  - So (hook alter example) becomes
    - `def hook_alter(form_id = 'login_form', form, _form_state)`
- underscores in front of arguments are ignored by compiler
- default values with double backslash \

  - ```
          IO.puts foo
        end```

APIs like this become possible

Drupal Form Alter

```
function example_login_form_form_alter($form, $form_state) {
  ...
}
```

A fake equivalent in Elixir

```
def alter(:login_form, form, _form_state) :do nil
```

Speaker notes

- How many people here are familiar with Drupal 7 and hook based development?
  - remember how we would do this?
    - `function example_login_form_form_alter($form, $form_state)`
  - Pattern matching makes this a part of the language in an interesting way.
    - https://youtu.be/gom6nEvtl3U?t=1159
    - you don't need to use switch or if as much for branching
  - So (hook alter example) becomes
    - `def hook_alter(form_id = 'login_form', form, _form_state)`
- underscores in front of arguments are ignored by compiler
- default values with double backslash \

  - ```
    IO.puts foo
    end```

Loops & Enumeration

Speaker notes

- There is no looping
  - state is immutable
  - reassigning a counter
- You will need to use recursion or Enumerables

Enumerable types

- List
- Map
- Range

Speaker notes

- A Type must implement the Enumerable protocol
- List, Map, and Range

Enumeration

- Enum.each
- Enum.map
- Enum.reduce([], fn)
- Enum.uniq([])

Speaker notes

- Enum.each performs action on each item
- Enum.map returns array after performing some action
- Enum.reduce([], fn) returns a single value
- Enum.uniq([]) returns a list with all duplicates removed

Recursion

```
def print_list([element|list]) do
  IO.puts element
  print_list(list)
end

def print_list([]), do: nil
```

Speaker notes

- 45:52 Recursion

```
    IO.puts element
    print_list(list)
  end
    def print_list([]), {do:} nil
    remove the curly brackets
```

Recursion

```elixir
def print_list([element|list]) do
  IO.puts element
  print_list(list)
end

def print_list([]), do: nil
```

```php
function print_list($array) {
  if (empty($array)) {
    return;
  }

  $first = array_shift($array);
  echo $first;
  print_list($array);
}
```

Speaker notes

- 45:52 Recursion

```
    IO.puts element
    print_list(list)
  end
    def print_list([]), {do:} nil
    remove the curly brackets
```

List Comprehension

Speaker notes

- I didn't know where else to put this.
- List comprehension is like a map, but the sytax is backwards.
- https://stackoverflow.com/a/47478020/1054656

Example

```
double_list = for el <- [1,2,3,4], do : el * 2
```

```
even_list = for el <- [1,2,3,4], do : rem(el, 2) == 0
```

Concurrency

Speaker notes

- In Elixir concurrency is based on the actor model
- That means you can spawn long running processes and then communicate with them by sending messages
- The BEAMVM handles sceduling for us and we don't have to worry about locking threads or anything like that. -if you developed with python or ruby you maybe know they have something called GIL (global interpreter lock). In short, the GIL ensures that only one thread shared memory.
- the GIL ensures thread-safe code, but it also makes really difficult to write code that can scale out running in parallel on multiple cores
- It is possible to write multi-threaded php in cli but not cgi
- you can just leave it to the webserver to spawn a new php thread to handle the request, but this doesn't give you multi-thread in a single application request
- https://www.poeticoding.com/spawning-processes-in-elixir-a-gentle-introduction-to-concurrency/

Concurrency

```
spawn(fn() -> do_stuff("foobar") end)

send(self(), {:greeting, "Robin"})
recieve do
  {:greeting, name} -> IO.puts "Hello \#{name}"
  {:farewell, name} -> IO.puts "Goodbye \#{name}"
after
  500 -> IO.puts "I didn't want to talk to you anyway!"
end
```

Speaker notes

- 1:01:21 Concurrency
  - spawn(fn() -> do_stuff("foobar") end)
  - send(self(), {:greeting, "Robin"})
  - recieve do {:greeting, name} -> IO.puts "Hello #{name}" {:farewell, name} -> IO.puts "Goodbye #{name}" after 500 -> IO.puts "I didn't want to talk to you anyway!" end
  - So we can spawn long running processes and have that process build a queue or something
  - Then we can have shorted processes receive messages from the long running process and do things for it.

Pipes

```
conn
  |> put_status(:created)
  |> put_resp_header("location", Routes._path(conn, :show, _
  |> render("show.json", _: _)
```

Speaker notes

- pipes
  - https://youtu.be/gom6nEvtl3U?t=1308
  - wraps the paramaters to wrapping functions
  - so `$array = reset(array_keys($old_array))`
    - would become `$old_array |> array_keys, reset`

Pipes

PHP

```
$array = array_sort(exploode(" ", $text))
```

Elixir

```
list1
  |> List.sort
  |> List.split(" ", list1)
```

Speaker notes

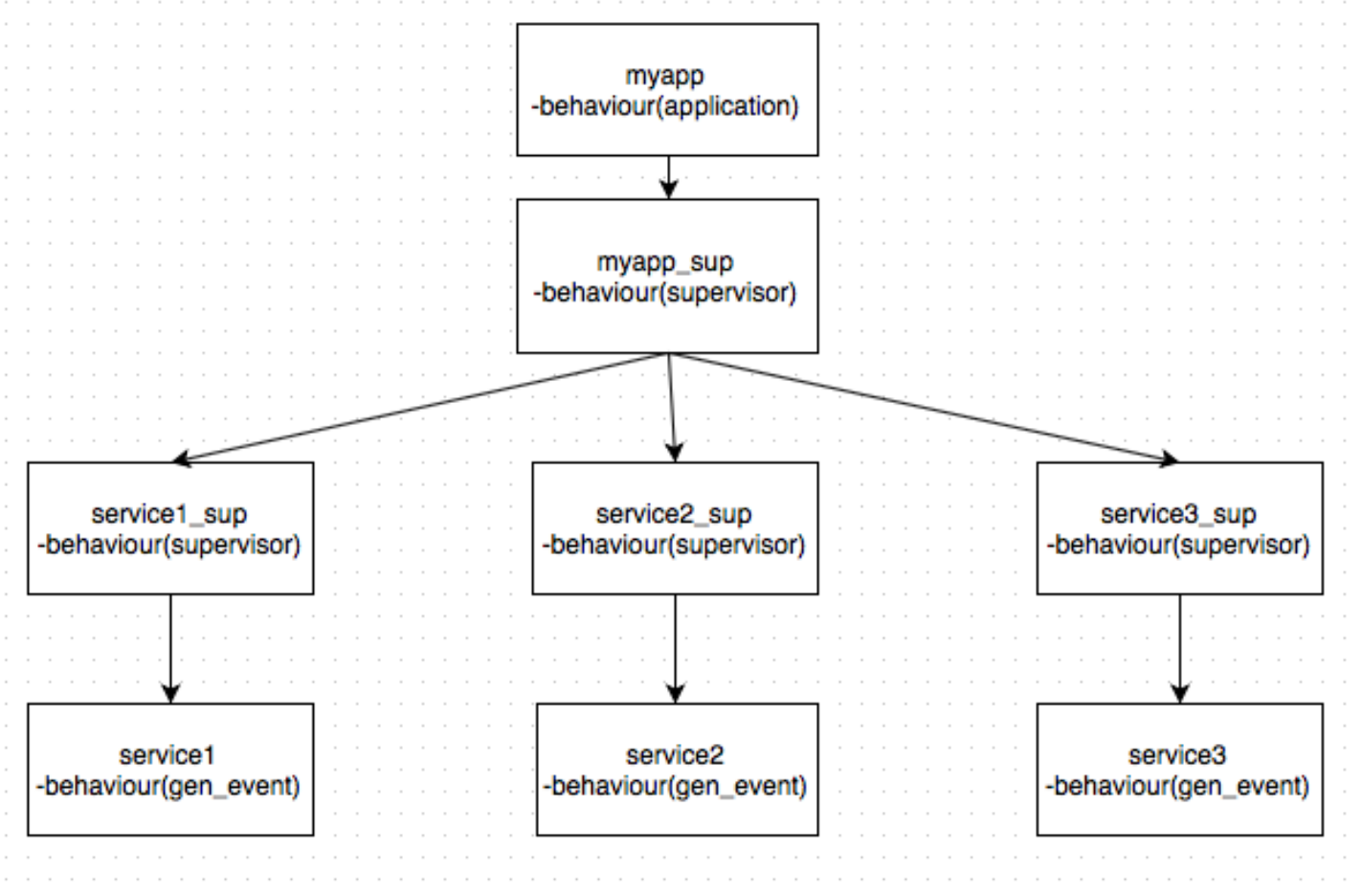
- example

Language Features

Speaker notes

- Now I'll go over some of the unique and cool language features.

OTP


myapp
-behaviour(application)
myapp_sup
-behaviour(supervisor)
service1_sup
-behaviour(supervisor)
service2_sup
-behaviour(supervisor)
service3_sup
-behaviour(supervisor)
service1
-behaviour(gen_event)
service2
-behaviour(gen_event)
service3
-behaviour(gen_event)

Speaker notes

- Open Telecom Platform
- gives us supervision trees, event managers, and other goodness.
- Basically all of the Erlang Platform.
- Concurrency
  - BeamVM will automatically use as many resources as you give it
  - That isn't limited to a single machine
  - once a another server is brought on line in only needs to link to the other servers and the load will be evenly spread across it

GenServer

Speaker notes

- PID
- cast
- each process has a mailbox
- think of it like microservives
- Let it die
  - don't worry about error handling
  - BEAM cannot segfault

Supervisors

Speaker notes

- The supervisors keep the lazy processes busy
  - Really it replaces the processes when they die
- reboots the worker processes
- processes started by supervisores aren't called by pid, but insteadh machinname
- Agents can be used to save state

Meta-Programming

```
def program(_) do
  IO.puts("Write Program")
end
```

Speaker notes

- This is what many of the features of Elixir that I just talked about is built on. It is a way to add syntactical sugar without changing the whole language.

Deployments

Speaker notes

- https://hexdocs.pm/distillery/home.html
- Deployments
  - Remember that this is a compiled language
  - OTP Has the ability to do no downtime releases
  - Elixir is built on OTP
  - Deployments code diff
    - Elixir has a build tool called distillery
    - A distillery build is a tarbal that includes everything needed to deploy
      - The build
      - Tools for running in these modes
        - console, foreground, and daemonized
      - and connecting to the running application
        - remote_console
        - you can open an interactive shell in the running application

No Downtime Deployments

Speaker notes

- Remember that this is a compiled language
- OTP Has the ability to do no downtime releases
- Elixir is built on OTP
- Deployments code diff

Where to deploy elixir

- AWS, Linode, Digital Ocean
- Heroku
- gigalixir
- Serverless

Speaker notes

- Where can I deploy my code?
- Self hosted
  - AWS, Linode, Digital Ocean
- Heroku
  - Elixir has roots in Ruby and so Heroku is obviously a target
  - I have heard mixed things about using Heroku
- gigalixir
  - This is another managed Elixir hosting, free for one instance with one database
- Serverless
  - I have mixed feeling about this.
  - It is easy managed and fire/forget
  - It is also more about request/response

Bring it home

You can add Phoenix to your site to add some real-time functionality.

Speaker notes

- Consider the easiest deployment
- You could add phoenix.js to your existing drupal site
  - enable the Drupal rest apis and start integrating
- Real-time doesn't always mean multi-user

What we covered

- How it works
- Technology
- Architecture
- Into to Elixir
- Conclusion

Speaker notes

- How it works
- Technology
- Architecture
- Into to Elixir
- Conclusion

Speaker notes

- There has been little actual gain with sharing code between front and back-end
- Why I gave the talk
  - The goal
  - The why, because there is so much more out there.
    - Drupal 8 got us off the island by addopting composer
    - Drupal is fine, it exists well within the php landscape
      - We need to get off our php planet
      - If Drupal is going to continue to get good we need to know what else is going on
- Live View
  - Last year, I wasn't planning on giving a talk at DrupalCon this year
  - I figured you all had heard enough from me two years in a row
  - Then I went to ElixirCon and I saw this
    - Demo some Phoenix LiveView
  - This has the potential to change everything about how a web app is expected to work
    - This got me excited
    - This is events in the browser causing things to happen on the server without a full http request and without isomorphic front/backend javascript
    - This is like the first time getting Drupal's ajax form api to work as expected

    - This is running on my machine right now
    - **play snake**
  - HTTP/2 and WS isn't the best sollution for everything
    - a website can just be a website
    - But Drupal's market position isn't in just a website
    - Drupal is for interesting web-apps

Join us for contribution sprints?

Friday, April 12, 2019

Mentored Core Sprints	First Time Contributor Workshops	General Sprints
9:00-18:00	9:00-12:00	9:00-18:00
Room: 602	Room: 606	Room: 6A

#DrupalContributions

Clarity Innovations | clarity-innovations.com

Speaker notes

- When are sprints
- Friday the 12th
- Mentored core sprints in room 602
- First Time springers in room 606
- General sprints in room 6A
- I will probably either be in the core sprint room or in the general sprint room.

What did you think?

Give us some feedback on this presentation!

Real-time Drupal (https://events.drupal.org/node/22347)

Take the DrupalCon Survey

www.surveymonkey.com/r/DrupalConSeattle

Links

Elixir Syntax Tutorial (https://www.youtube.com/watch?v=pBNOavRoNL0) This session (https://events.drupal.org/node/22347) Feedback (https://www.surveymonkey.com/r/DrupalConSeattle) Elixir and Phoenix Introduction (https://youtu.be/bk3icU8iIto) DrupalCon Austin 2014: Functional PHP (https://www.youtube.com/watch?v=M3_xnTK6-pA) Distillery Docs (https://hexdocs.pm/distillery/home.html) Into to Concurrency (https://www.poeticoding.com/spawning-processes-in-elixir-a-gentle-introduction-to-concurrency/) Into to Mix (https://elixir-lang.org/getting-started/mix-otp/introduction-to-mix.html) List Comprehension (https://stackoverflow.com/a/47478020/1054656) Streaming through a pool (http://learningelixir.joekain.com/streaming-through-a-pool-in-elixir/)