elixir

# Ruby + Erlang = shiny

Ben Tyler, AmsterdamX.pm

# Elixir

- github.com/kanatohodets/intro-to-elixir-talk
- the language in brief
- on pre-emptive schedulers
- convention, configuration, concurrency
- turning it off and back on again
- ???

# The language

- Erlang VM
- functional programming for dummies
- dynamic, strong typing
- normal-enough syntax
- pattern matching

# compliments-cli

# compliments-cli

```
defmodule SayNiceThings do
  def start do
    {:ok, blob_of_nice} = File.read("compliments.dat")
    list_of_nice = String.split(blob_of_nice, "\n")
    say_nice_things(list_of_nice)
  end
  # ... (say_nice_things definition)
end
SayNiceThings.start
```

# compliments-cli

```elixir
defp say_nice_things(list_of_nice) do
  # just like: status = String.strip(IO.gets("..."))
  status =
    IO.gets("Are you feeling better? (y/n) ") |> String.strip
  case status do
    "y" ->
      IO.puts "hooray!"
    "n" ->
      IO.puts IO.ANSI.green() <> Enum.random(list_of_nice) <> IO.ANSI.reset()
      say_nice_things(list_of_nice)
    _ ->
      IO.puts "Not sure what that means."
      say_nice_things(list_of_nice)
  end
end
```

# compliments-api

a shrewdness of schedulers

cooperative and preemptive

# a shrewdness of schedulers

## cooperative

- event loops - JavaScript, IO::Async, AnyEvent, Mojolicious, POE
- coroutines - Coro, Python generators
- don’t hog the loop! (blocking operations)

# a shrewdness of schedulers

## preemptive

- operating systems (except Mac pre-OSX, Windows before 95)
- threads (libpthread, etc.)
- green threads (Go, Perl 6, Akka, Erlang/Elixir*)
- what is happening?!

# a shrewdness of schedulers

preemptive

- green threads (Go, Perl 6, Akka, Erlang/Elixir*)
- * "green processes" (no shared memory)

# compliments-api

(for netcat users)

# compliments-api

module + initialization

```elixir
defmodule TelnetNiceThings do
  def listen(port) do
    {:ok, blob_of_nice} = File.read("compliments.dat")
    list_of_nice = String.split(blob_of_nice, "\n")
    # spawn a process to hold the list_of_nice state
    Agent.start_link(fn -> list_of_nice end, name: :nice_things_db)

    tcp_options = [:binary, active: false, packet: :line, reuseaddr: true]
    {:ok, socket} = :gen_tcp.listen(port, tcp_options)
    IO.puts("bringing joy on #{port}!")
    accept_and_serve(socket)
  end
  # ... accept_and_serve definition
end
TelnetNiceThings.listen(4242);
```

# compliments-api

accepting a client

```
# ... listen definition
defp accept_and_serve(listen_socket) do
  {:ok, client_socket} = :gen_tcp.accept(listen_socket)
  # spawn a process to serve this client (think goroutine)
  spawn(fn() -> say_nice_things(client_socket) end)
  accept_and_serve(listen_socket)
end
# ... say_nice_things definition
```

# compliments-api

```
defp say_nice_things(socket) do
  prompt = "are you feeling better?> "
  :gen_tcp.send(socket, prompt)
  {:ok, status_raw} = :gen_tcp.recv(socket, 0)
  status = String.rstrip(to_string(status_raw))
  case status do
    "y" ->
      :gen_tcp.send(socket, "hooray!\n")
      :gen_tcp.close(socket)
    "n" ->
      # query the :nice_things_db process for some state
      nice_thing = Agent.get(:nice_things_db, fn(list_of_nice) ->
          Enum.random(list_of_nice)
      end)
      msg = IO.ANSI.green() <> nice_thing <> IO.ANSI.reset()
      :gen_tcp.send(socket, "#{msg}\n")
      say_nice_things(socket)
    _ ->
      :gen_tcp.send(socket, "Not sure what that means.\n")
      say_nice_things(socket)
  end
end
```

# concurrency!

# structured concurrency: OTP

(open telecom platform)

# OTP?

- patterns for writing good processes
- message APIs `{:reply, :life_is_good …}`
- synchronous/asynchronous calling conventions
- supervisors (turn it off and back on again)

# compliments-otp

(terminal time)

# other stuff!

- slick web framework: Phoenix
- sigils (and custom sigils!)
- macros
- nice tools

{:exit, :talk_over}