Contents

1 BYOGenServer - An Exploration

1.1 Concurrency Primitives

1.1.1 Processes

```
run_query = fn(number) ->
:timer.sleep(2000)
"result #{number}"
end

1..5 |> Enum.each(&run_query.(&1))
```

1.2 Spawn

1.2.1 Spawning a process

```
spawn(fn -> :timer.sleep(2000); IO.puts("Hello from process #{inspect(self)}") end )
#PID<0.91.0>
```

1.2.2 Spawning multiple processes

```
def async_query(number) do
  spawn(fn ->
    :timer.sleep(2000)
    IO.puts("Query number #{number} in process #{inspect(self)}")
  end)
end
```

1.3 Communicating with processes

1.3.1 Sending a message

```
send(self, "Hola Kumusta")
```

1.4 Receive

1.4.1 Receiving a message

```
receive do
"Hola Kumusta" ->
  IO.puts("Konichiwa")
end
```

1.4.2 Failed messages can clog up a mailbox

```
send(self, "Pardon my French")
receive do
  "Hola Kumusta" ->
    IO.puts("Konichiwa")
  _ ->
    IO.puts("I don’t know, maybe ask Siri")
end
```

1.4.3 Don’t wait forever

```
receive do
  after 5000 ->
    "I’m done waiting. Mic Drop."
end
```

1.4.4 Communication with a running process

```
def message_receiver do
  spawn(fn ->
    receive do
      "Hola Kumusta" ->
        IO.puts("Konnichiwa")
      _ ->
        IO.puts("I don’t know, maybe ask Siri.")
    after 5000
      "I’m done waiting. "
    end
  end)
end

def say_hola(pid) do
  send(pid, "Hola Kumusta")
end
```

1.4.5 Replying to a sender

```
  def message_responder do
    spawn(fn ->
      receive do
        {caller, "Hola Kumusta"} ->
          send(caller, "Konnichiwa")
      end
    end)
  end

pid = Processes.message_responder
send(pid, {self, "Hola Kumusta"})
result = receive do
  msg -> msg
end

IO.puts(result)
```

1.5 Long running processes

1.5.1 Recursion is our friend

```
defmodule BYOGenServer.Looping do
  def start do
    spawn(&loop/0)
  end

  defp loop do
    receive do
      #do stuff
    end
    loop
  end
end
```

1.5.2 Stateful processes

```
defmodule BYOGenServer.StatefulProcess do
  def start do
    spawn(fn ->
      loop("")
    end)
  end

  defp loop(state) do
    new_state = receive do
      {:get, caller} ->
        send(caller, state)
      {:new_string, str} when is_binary(str) ->
        str
      {:reverse} ->
        String.reverse(state)
      {:downcase} ->
        String.downcase(state)
    end
    loop(new_state)
  end
end
```

1.5.3 Naming a process

```
Process.register(pid, :some_name)
```