Episode 07: Enum and Stream

Summary

- Enum
- Capture operator
- Streams

Enum

Typical function signature: `Enum.function(Enumerable, function)`

Types which implement `Enumerable`

- Lists
- Keyword lists
- Maps (but not Structs in order to support polymorphism w/ protocols)
- Ranges
- Streams

Reminder regarding maps and `Enumerable` functions

Maps act like lists of tuples, where each entry is represented by `{:key, value}`.

Functions mentioned at LearnElixir.tv

- `Enum.at/2`
- `Enum.filter/2`
- `Enum.reduce/2`
- `Enum.into/2`
  - *Useful for turning the results of Enum functions on maps (i.e. keyword lists) back into maps*
- `Enum.take/2`

Functions mentioned at ElixirSchool

Capture Operator

This is pretty self explanatory, so I've just included some examples to jog my memory.

```
Enum.reduce([1, 2, 3], &(&1 + &2))
Enum.reduce([1, 2, 3], &+/2)

Enum.filter([1, 2, 3], &is_number/1)
Enum.filter([1, 2, 3], &is_number(&1))

Enum.map(["napoleon", "bonaparte"], &String.upcase/1)
```

Stream

When you call `Stream` functions, you are actually building a struct containing (1) the enumerable and (2) all the functions that will operate on it. This is so that Elixir can do the work all at once at a later time. For example,

```
[1, 2, 3, "string"]
|> Stream.filter(&is_number/1)
|> Stream.map(&(&1 * 2))
```

is equivalent to,

```
%Stream{
  enum: [1, 2, 3, "string"],
  funs: [
    #Function<40.129278153/1 in Stream.filter/2>,
    #Function<48.129278153/1 in Stream.map/2>
  ]
}
```

Stream is better than enum for

1. Long (or infinite!) enumerables, and/or
2. when performing multiple operations.

Functions mentioned at LearnElixir.tv

- `Stream.cycle/1`
- `Stream.iterate/2`
- `Stream.resource/3`
  - Useful for converting anything into a stream
  - e.g., paginated data, lines in a file, events on a socket
  - Resources
    - resource/3
    - Stream Paginated APIs in Elixir
    - Daniel’s updated method for paginated APIs