Rob Howard @damncabbage

Traversing Error Mountain

1920 x 1080

80 Montserrat

55 Monaco.f({a:b});

43 Monaco.f({a:b});

35 Open Sans

Mont Monaco.f() Open

Mont Monaco.f() **Open**

Rob Howard @damncabbage

Traversing Error Mountain

1920 x 1080

80 Familiar Pro

55 Inconso.f({a:b});

43 Inconso.f({a:b});

35 Open Sans

Famil Inconso.f() Open

Famil Inconso.f() Open

Traversing Error Mountain

Traversing Error Mountain

A Helpful Little Function and Some Building Blocks

{:error, ...}
{:ok, ...}

An Elixir Pattern

```
{:ok, …}
{:error, …}
```

```
{:ok, …}          ✅
{:error, …}
```

```
{:ok, …}       ✅
{:error, …}    ❌
```

```
{:ok, 123}
{:error, …}
```

```
{:ok, 123}
{:error, "kaboom"}
```

```
File.read("hello.txt")
#=> {:ok, "World"}

File.read("invalid.txt")
#=> {:error, :enoent}
```

```
File.read("hello.txt")
#=> {:ok, "World"}

File.read("invalid.txt")
#=> {:error, :enoent}
```

```
File.read("hello.txt")
#=> {:ok, "World"}

File.read("invalid.txt")
#=> {:error, :enoent}
```

```
MyApp.Repo.insert(changes)
#=> {:ok, #Ecto.Changeset<>}

MyApp.Repo.insert(wtf)
#=> {:error, #Ecto.Changeset<>}
```

The Example

```
files = ["abc.json", "def.json", "ghi.json"]

contents = files
|> Enum.map(fn (file) ->
     File.read(file)
   end)


# => [
    {:ok, "..."},
    {:error, :enoent},
    {:ok, "..."}
  ]
```

The Function

🥁......

traverse/2

traverse/2

```
[...]
```

traverse/2

```
[...]

fn (…) ->
  {:ok, …}
end
```

traverse/2

```
[...]
```

```
fn (…) ->
  {:error, …}
end
```

```
files = ["abc.json", "def.json", "ghi.json"]

contents = files
|> Enum.map(fn (file) ->
     File.read(file)
   end)



# =>
  [  {:ok, "..."},
     {:error, :enoent},
     {:ok, "..."}
  ]
```

```elixir
files = ["abc.json", "def.json", "ghi.json"]

contents = files
|> Enum.map(fn (file) ->
     File.read(file)
   end)



# =>
  [  {:ok, "..."},
     {:error, :enoent},
     {:ok, "..."}
  ]
```

```
files = ["abc.json", "def.json", "ghi.json"]

contents = files
|> Result.traverse(fn (file) ->
     File.read(file)
   end)




# => {:error, :enoent}
```

```
files = ["abc.json", "def.json", "ghi.json"]

contents = files
|> Result.traverse(fn (file) ->
     File.read(file)
   end)






# => {:error, :enoent}
```

```
files = ["abc.json", "def.json", "ghi.json"]

contents = files
|> Result.traverse(fn (file) ->
     File.read(file)
   end)




# => {:error, :enoent}
```

```
files = ["abc.json", "def.json", "ghi.json"]

contents = files
|> Result.traverse(fn (file) ->
     File.read(file)
   end)




# => {:ok, [" {} ", "[123]", "{'hi':123}"] }
```

Building Blocks

Construction

```
{:ok,123}       ✅
{:error,"💥"}    ❌
```

```
{:ok,123}
{:error,"💥"}
```

```
ok(123)
error("💥")
```

```
ok(123) → {:ok, 123}
error("💥") → {:error,"💥"}
```

```
[]
[123]
[123, 456]
```

```
[]
[123 | []]
[123 | [456 | []]]
```

[123]

```
List.wrap(123)
List.wrap(123) → [123]
```

Deconstruction

```
case value do
  {:ok, x} ->
    x + 1

  {:error, e} ->
    "Problem: #{e}"
end
```

```
????(value,
  fun(x) ->
    x + 1
  end,
  fun(e) ->
    "Problem: #{e}"
  end
)
```

```
case  [1,2,3] do
  [] ->
    0
  [head] ->
    head + 1
  [head | tail] ->
    ???
end
```

```
foldr([1,2,3],

  0,
  fun(x, acc) ->
    x + acc
  end

)
```

```
????(value,
  fun(x) ->
    x + 1
  end,
  fun(e) ->
    "Problem: #{e}"
  end
)
```

```
Result.fold(value,
  fun(x) ->
    x + 1
  end,
  fun(e) ->
    "Problem: #{e}"
  end
)
```

```
Result.match(value,
  fun(x) ->
    x + 1
  end,
  fun(e) ->
    "Problem: #{e}"
  end
)
```

BACK to

The Example

```
files = ["abc.json", "def.json", "ghi.json"]

contents = files
|> Result.traverse(fn (file) ->
     File.read(file)
   end)









# => {:error, :enoent}
```

```
files = ["abc.json", "def.json", "ghi.json"]

contents = files
|> Result.traverse(fn (file) ->
     file
     |> File.read()
   end)
```

```
# => {:error, :enoent}
```

```
files = ["abc.json", "def.json", "ghi.json"]

contents = files
|> Result.traverse(fn (file) ->
     file
     |> File.read()
     |> Result.map(fn (x) -> String.trim(x) end)
   end)



# => {:error, :enoent}
```

map()???

map()

```
Enum.map([1,2,3],
  fun(x) ->
    x * 2
  end
)

# => [2,4,6]
```

```
Enum.map([1,2,3],
  fun(x) ->
    x * 2
  end
)

# => [2,4,6]
```

```elixir
Enum.map([1,2,3],
  fun(x) ->
    x * 2
  end
)

# => [2,4,6]
```

"Structure Preserving"

```
Result.map({:ok, 111}
  fun(x) ->
    x * 2
  end
)

# => {:ok, 222}
```

```elixir
Result.map({:ok, 111}
  fun(x) ->
    x * 2
  end
)

# => {:ok, 222}
```

```
Result.map({:error, "nope"},
  fun(x) ->
    x * 2
  end
)

# => {:error, "nope"}
```

```
def map(x, func) do
  Result.match(x,
    fn (val) ->
      func.(val) |> Result.ok()
    end,
    fn (err) ->
      e |> Result.error()
    end
  )
end
```

```
def map(x, func) do
  case x do
    {:ok, val} ->
      func.(val) |> Result.ok()

    {:error, err} ->
      e |> Result.error()

  end
end
```

```
files = ["abc.json", "def.json", "ghi.json"]

contents = files
|> Result.traverse(fn (file) ->
     file
     |> File.read()
     |> Result.map(fn (x) -> String.trim(x) end)
   end)




# => {:error, :enoent}
```

```
files = ["abc.json", "def.json", "ghi.json"]

contents = files
|> Result.traverse(fn (file) ->
     file
     |> File.read()
     |> Result.map(fn (x) -> String.trim(x) end)
   end)




# => {:error, :enoent}
```

```
files = ["abc.json", "def.json", "ghi.json"]

contents = files
|> Result.traverse(fn (file) ->
     file
     |> File.read()
     |> Result.map(fn (x) -> String.trim(x) end)
   end)




# => {:error, :enoent}
```

```
files = ["abc.json", "def.json", "ghi.json"]

contents = files
|> Result.traverse(fn (file) ->
     file
     |> File.read()
     |> Result.map(fn (x) -> String.trim(x) end)
   end)



# => {:ok, [" {} ", "[123]", "{'hi':123}"] }
```

```
files = ["abc.json", "def.json", "ghi.json"]

contents = files
|> Result.traverse(fn (file) ->
     file
     |> File.read()
     |> Result.map(fn (x) -> String.trim(x) end)
   end)



# => {:ok, ["{}",   "[123]", "{'hi':123}"] }
```

```
files = ["abc.json", "def.json", "ghi.json"]

contents = files
|> Result.traverse(fn (file) ->
     file
     |> File.read()
     |> Result.map(fn (x) -> String.trim(x) end)
   end)




# => {:error, :enoent}
```

```
files = ["abc.json", "def.json", "ghi.json"]

contents = files
|> Result.traverse(fn (file) ->
     file
     |> File.read()
     |> Result.map(&String.trim/1)
   end)




# => {:error, :enoent}
```

```
files = ["abc.json", "def.json", "ghi.json"]

contents = files
|> Result.traverse(fn (file) ->
     file
     |> File.read()
     |> Result.map(&String.trim/1)
     |> Result.map_error(fn (e) ->
          {file, e}
        end)
   end)

# => {:error, {"def.json", :enoent}}
```

```
files = ["abc.json", "def.json", "ghi.json"]

contents = files
|> Result.traverse(fn (file) ->
     file
     |> File.read()
     |> Result.map(&String.trim/1)
     |> Result.map_error(fn (e) ->
          {file, e}
        end)
   end)

# => {:error, {"def.json", :enoent}}
```

```
files = ["abc.json", "def.json", "ghi.json"]

contents = files
|> Result.traverse(fn (file) ->
     file
     |> File.read()
     |> Result.map(&String.trim/1)
     |> Result.map_error(fn (e) ->
          {file, e}
        end)
   end)

# => {:error, {"def.json", :enoent}}
```

```elixir
files = ["abc.json", "def.json", "ghi.json"]

contents = files
|> Result.traverse(fn (file) ->
     file
     |> File.read()
     |> Result.map(&String.trim/1)
     |> Result.map_error(fn (e) ->
          {file, e}
        end)
   end)

# => {:error, {"def.json", :enoent}}
```

```
files = ["abc.json", "def.json", "ghi.json"]

contents = files
|> Result.traverse(fn (file) ->
     file
     |> File.read()
     |> Result.map(&String.trim/1)
     |> Result.annotate(fn (e) ->
          {file, e}
        end)
   end)

# => {:error, {"def.json", :enoent}}
```

```
files = ["abc.json", "def.json", "ghi.json"]

contents = files
|> Result.traverse(fn (file) ->
     file
     |> File.read()
     |> Result.map(&String.trim/1)
     |> Result.map_error(fn (e) ->
          {file, e}
        end)
   end)

# => {:error, {"def.json", :enoent}}
```

```
files = ["abc.json", "def.json", "ghi.json"]

contents = files
|> Result.traverse(fn (file) ->
    file
    |> File.read()
    |> Result.map(&String.trim/1)
    |> Result.map_error(fn (e) -> {file, e} end)
  end)



# => {:error, {"def.json", :enoent}}
```

```
files = ["abc.json", "def.json", "ghi.json"]

contents = files
|> Result.traverse(fn (file) ->
     file
     |> File.read()




     |> Result.map_error(fn (e) -> {file, e} end)
   end)

# => {:error, {"def.json", :enoent}}
```

```
files = ["abc.json", "def.json", "ghi.json"]

contents = files
|> Result.traverse(fn (file) ->
     file
     |> File.read()
     |> Result.and_then(fn (contents) ->
          Poison.decode(contents)
        end)
     |> Result.map_error(fn (e) -> {file, e} end)
   end)

# => {:error, {"def.json", :enoent}}
```

```elixir
files = ["abc.json", "def.json", "ghi.json"]

contents = files
|> Result.traverse(fn (file) ->
     file
     |> File.read()
     |> Result.and_then(fn (contents) ->
          Poison.decode(contents)
        end)
     |> Result.map_error(fn (e) -> {file, e} end)
   end)

# => {:error, {"def.json", :enoent}}
```

and_then()

```
Result.and_then(
  {:ok, "[1,2,3]"},
  fun(x) ->
    Poison.decode(x)
  end
)

# => {:ok, [1,2,3]}
```

```
Result.and_then(
  {:error, :yeah_nah},
  fun(x) ->
    Poison.decode(x)
  end
)

# => {:error, :yeah_nah}
```

```
Result.and_then(
  {:ok, "[1,2,3]"},
  fun(x) ->
    Poison.decode(x)
  end
)

# => {:ok, [1,2,3]}
```

```
Result.and_then(
  {:ok, "haha bad json"},
  fun(x) ->
    Poison.decode(x)
  end
)

# => {:error, {:invalid, ...}
```

```
{:ok, "foo.json"}
|> Result.and_then(
     &File.read/1
   )
|> Result.and_then(
     &Poison.decode/1
   )
|> ...
```

```elixir
def and_then(x, func) do
  match(x,
    func,
    &Result.error/1
  )
end
```

```
def and_then(x, func) do
  case x do
    {:ok, val} -> func.(val)
    {:error, e} -> Result.error(e)
  end
end
```

```
files = ["abc.json", "def.json", "ghi.json"]

contents = files
|> Result.traverse(fn (file) ->
     file
     |> File.read()
     |> Result.and_then(fn (contents) ->
          Poison.decode(contents)
        end)
     |> Result.map_error(fn (e) -> {file, e} end)
   end)

# => {:error, {"def.json", :enoent}}
```

```
files = ["abc.json", "def.json", "ghi.json"]

contents = files
|> Result.traverse(fn (file) ->
     file
     |> File.read()
     |> Result.and_then(fn (contents) ->
          Poison.decode(contents)
        end)
     |> Result.map_error(fn (e) -> {file, e} end)
   end)

# => {:error, {"def.json", :enoent}}
```

```elixir
files = ["abc.json", "def.json", "ghi.json"]

contents = files
|> Result.traverse(fn (file) ->
     file
     |> File.read()
     |> Result.and_then(fn (contents) ->
          Poison.decode(contents)
        end)
     |> Result.map_error(fn (e) -> {file, e} end)
   end)

# => {:error, {"def.json", :enoent}}
```

```
files = ["abc.json", "def.json", "ghi.json"]

contents = files
|> Result.traverse(fn (file) ->
     file
     |> File.read()
     |> Result.and_then(fn (contents) ->
          Poison.decode(contents)
        end)
     |> Result.map_error(fn (e) -> {file, e} end)
   end)

# => {:error, {"ghi.json", {:invalid, "{", 23}}}
```

```elixir
files = ["abc.json", "def.json", "ghi.json"]

contents = files
|> Result.traverse(fn (file) ->
     file
     |> File.read()
     |> Result.and_then(fn (contents) ->
          Poison.decode(contents)
        end)
     |> Result.map_error(fn (e) -> {file, e} end)
   end)

# => {:ok, [%{"hello" => "world"}, [1,2,3], %{} ]}
```

traverse()

```
def traverse(results, func) do
  Enum.reduce_while(results, {:ok, []},
    fn(element, {:ok, okays}) ->
      Result.match(func.(element),
        fn (v) ->
           {:cont, {:ok, [v | okays]}} end
        fn (e) ->
           {:halt, {:error, e}} end
      )
    end
  )
  |> Result.map(&Enum.reverse/1)
end
```

```elixir
def traverse(results, func) do
  Enum.reduce_while(results, {:ok, []},
    fn(element, {:ok, okays}) ->
      case func.(element) do
        {:ok, v} ->
           {:cont, {:ok, [v | okays]}}
        {:error, e} ->
           {:halt, {:error, e}}
      end
    end
  )
  |> Result.map(&Enum.reverse/1)
end
```

```elixir
def traverse(results, func) do
  Enum.reduce_while(results, {:ok, []},
    fn(element, {:ok, okays}) ->
      case func.(element) do
        {:ok, v} ->
           {:cont, {:ok, okays ++ [v]}}
        {:error, e} ->
           {:halt, {:error, e}}
      end
    end
  )
end
```

```elixir
def traverse(results, func) do
  Enum.reduce_while(results, {:ok, []},
    fn(element, {:ok, okays}) ->
      case func.(element) do
        {:ok, v} ->
           {:cont, {:ok, [v | okays]}}
        {:error, e} ->
           {:halt, {:error, e}}
      end
    end
  )
  |> Result.map(&Enum.reverse/1)
end
```

So What?

Composition 💖

~~Combinators~~

Helper Functions

Composition 💖

Cheat Codes

```
    match() -> Fold
      map() -> Functor
 and_then() -> Monad
```

```
     match()  -> Fold
       map()  -> Functor
  and_then()  -> Monad
```

https://gist.github.com/damncabbage/b3de0cd72a345fab7ba63c2898e00b63

https://github.com/CrowdHailer/OK

https://github.com/expede/witchcraft

"Railway-Oriented Programming"
https://fsharpforfunandprofit.com/rop/

Traversing Error Mountain

A Helpful Little Function and Some Building Blocks

Rob Howard
@damncabbage
http://robhoward.id.au