@oheydrew

PATTERN MATCHING & WITH STATEMENTS IN ELIXIR

`WITH` GREAT POWER

Start with something basic
(pattern matching)

Use it in something more complex
(with statements)

Pattern Matching

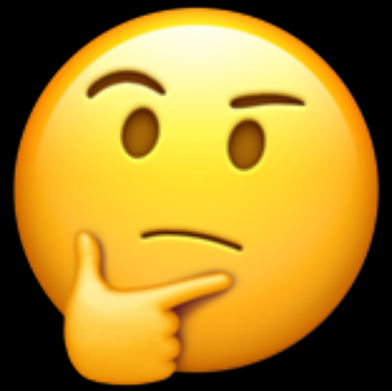
this = that

(fun fact: In elixir, = is not the “equality” operator, it’s the “match” operator)

```
%{animal: "tiger"} = %{animal: "tiger"}
```

```
%{animal: "monkey"} = %{animal: "tiger"}
```

```
%{animal: _ } = %{animal: "tiger"}
```

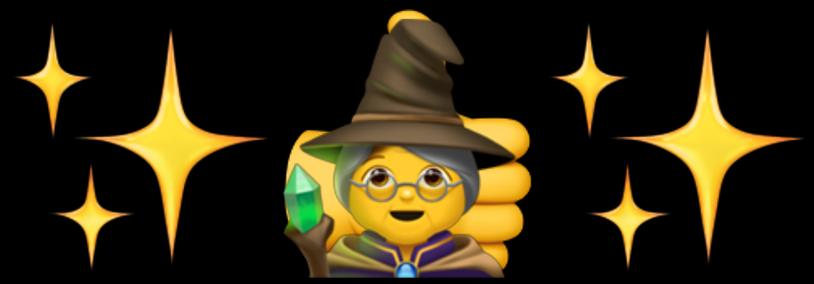


```
animal = "meerkat"

case animal do
  "meerkat" -> IO.puts("Hell yes, meerkats!")
  _         -> IO.puts(“...those aren’t meerkats. I’m going home”)
end
```

```
%{animal: my_animal} = %{animal: "tiger"}

my_animal = "tiger"
```



Pattern Matching = Powerful Wizardry

Matching

&

Assignment

```elixir
def print_child_details(%{name: nil}), do: IO.puts("This child has no name.")
def print_child_details(%{name: name}), do: IO.puts("Child's name: #{name}")
def print_child_details(_), do: IO.puts("Error: Invalid data structure.")
```

What’s with `with`?

Take some kind of data…

Do something with it…

Do something else with it…

Do another thing with it…

Pass it on to something else.

Error Handling

left <- right

```
with
  pattern_match_output  <- call_function(data),
  another_pattern_match <- call_another_function(pattern_match_output),
  final_pattern_match   <- yet_another_function(another_pattern_match)

do
  do_something_with(final_pattern_match, another_pattern_match)

else
  nil   -> handle_nil()
  error -> handle_error(error)
```

Successful with -> do

```
def update_and_activate_child(child_id, new_details) do
  with {:ok, child}         <- Children.find_by(child_id),
       {:ok, updated_child} <- Children.update_details(child, new_details),
       :ok                  <- Children.activate_enrolment(updated_child)
  do
    render(ChildrenView, "success.json", %{data: %{child: updated_child}})
  else
    {:error, error_reason}  -> {:error, :find_or_update_error, error_reason}
    error_reason            -> {:error, :generic_error, error_reason}
  end
end
```

Successful with -> do

```
def update_and_activate_child(child_id, new_details) do
  with {:ok, child}         <- Children.find_by(child_id),
       {:ok, updated_child} <- Children.update_details(child, new_details),
       :ok                  <- Children.activate_enrolment(updated_child)
  do
    render(ChildrenView, "success.json", %{data: %{child: updated_child}})
  else
    {:error, error_reason}  -> {:error, :find_or_update_error, error_reason}
    error_reason            -> {:error, :generic_error, error_reason}
  end
end
```

Successful with -> do

```
def update_and_activate_child(child_id, new_details) do
  with {:ok, child}         <- Children.find_by(child_id),
       {:ok, updated_child} <- Children.update_details(child, new_details),
       :ok                  <- Children.activate_enrolment(updated_child)
  do
    render(ChildrenView, "success.json", %{data: %{child: updated_child}})
  else
    {:error, error_reason}  -> {:error, :find_or_update_error, error_reason}
    error_reason            -> {:error, :generic_error, error_reason}
  end
end
```

Successful with -> do

```
def update_and_activate_child(child_id, new_details) do
  with {:ok, child}         <- Children.find_by(child_id),
       {:ok, updated_child} <- Children.update_details(child, new_details),
       :ok                  <- Children.activate_enrolment(updated_child)
  do
    render(ChildrenView, "success.json", %{data: %{child: updated_child}})
  else
    {:error, error_reason}  -> {:error, :find_or_update_error, error_reason}
    error_reason            -> {:error, :generic_error, error_reason}
  end
end
```

Successful with -> do

```
def update_and_activate_child(child_id, new_details) do
  with {:ok, child}         <- Children.find_by(child_id),
       {:ok, updated_child} <- Children.update_details(child, new_details),
       :ok                  <- Children.activate_enrolment(updated_child)
  do
    render(ChildrenView, "success.json", %{data: %{child: updated_child}})
  else
    {:error, error_reason}  -> {:error, :find_or_update_error, error_reason}
    error_reason            -> {:error, :generic_error, error_reason}
  end
end
```

Successful with -> do

```
def update_and_activate_child(child_id, new_details) do
  with {:ok, child}         <- Children.find_by(child_id),
       {:ok, updated_child} <- Children.update_details(child, new_details),
       :ok                  <- Children.activate_enrolment(updated_child)
  do
    render(ChildrenView, "success.json", %{data: %{child: updated_child}})
  else
    {:error, error_reason}  -> {:error, :find_or_update_error, error_reason}
    error_reason            -> {:error, :generic_error, error_reason}
  end
end
```

Successful with -> do

```
def update_and_activate_child(child_id, new_details) do
  with {:ok, child}         <- Children.find_by(child_id),
       {:ok, updated_child} <- Children.update_details(child, new_details),
       :ok                  <- Children.activate_enrolment(updated_child)
  do
    render(ChildrenView, "success.json", %{data: %{child: updated_child}})
  else
    {:error, error_reason}  -> {:error, :find_or_update_error, error_reason}
    error_reason            -> {:error, :generic_error, error_reason}
  end
end
```

Successful with -> do

```elixir
def update_and_activate_child(child_id, new_details) do
  with {:ok, child}         <- Children.find_by(child_id),
       {:ok, updated_child} <- Children.update_details(child, new_details),
       :ok                  <- Children.activate_enrolment(updated_child)
  do
    render(ChildrenView, "success.json", %{data: %{child: updated_child}})
  else
    {:error, error_reason}  -> {:error, :find_or_update_error, error_reason}
    error_reason            -> {:error, :generic_error, error_reason}
  end
end
```

Failures -> else

```
def update_and_activate_child(child_id, new_details) do
  with {:ok, child}         <- Children.find_by(child_id),
       {:ok, updated_child} <- Children.update_details(child, new_details),
       :ok                  <- Children.activate_enrolment(updated_child)
  do
    render(ChildrenView, "success.json", %{data: %{child: updated_child}})
  else
    {:error, error_reason}  -> {:error, :find_or_update_error, error_reason}
    error_reason            -> {:error, :generic_error, error_reason}
  end
end
```

Failures -> else

```
def update_and_activate_child(child_id, new_details) do
  with {:ok, child}         <- Children.find_by(child_id),
       {:ok, updated_child} <- Children.update_details(child, new_details),
       :ok                  <- Children.activate_enrolment(updated_child)
  do
    render(ChildrenView, "success.json", %{data: %{child: updated_child}})
  else
    {:error, error_reason}  -> {:error, :find_or_update_error, error_reason}
    error_reason            -> {:error, :generic_error, error_reason}
  end
end
```

Failures -> else

```
def update_and_activate_child(child_id, new_details) do
  with {:ok, child}         <- Children.find_by(child_id),
       {:ok, updated_child} <- Children.update_details(child, new_details),
       :ok                  <- Children.activate_enrolment(updated_child)
  do
    render(ChildrenView, "success.json", %{data: %{child: updated_child}})
  else
    {:error, error_reason}  -> {:error, :find_or_update_error, error_reason}
    error_reason            -> {:error, :generic_error, error_reason}
  end
end
```

Failures -> else

```elixir
def update_and_activate_child(child_id, new_details) do
  with {:ok, child}         <- Children.find_by(child_id),
       {:ok, updated_child} <- Children.update_details(child, new_details),
       :ok                  <- Children.activate_enrolment(updated_child)
  do
    render(ChildrenView, "success.json", %{data: %{child: updated_child}})
  else
    {:error, error_reason}  -> {:error, :find_or_update_error, error_reason}
    error_reason            -> {:error, :generic_error, error_reason}
  end
end
```

Failures -> else

```
def update_and_activate_child(child_id, new_details) do
  with {:ok, child}         <- Children.find_by(child_id),
       {:ok, updated_child} <- Children.update_details(child, new_details),
       :ok                  <- Children.activate_enrolment(updated_child)
  do
    render(ChildrenView, "success.json", %{data: %{child: updated_child}})
  else
    {:error, error_reason}  -> {:error, :find_or_update_error, error_reason}
    error_reason            -> {:error, :generic_error, error_reason}
  end
end
```

Failures -> else

```
def update_and_activate_child(child_id, new_details) do
  with {:ok, child}         <- Children.find_by(child_id),
       {:ok, updated_child} <- Children.update_details(child, new_details),
       :ok                  <- Children.activate_enrolment(updated_child)
  do
    render(ChildrenView, "success.json", %{data: %{child: updated_child}})
  else
    {:error, error_reason}  -> {:error, :find_or_update_error, error_reason}
    error_reason            -> {:error, :generic_error, error_reason}
  end
end
```

Failures -> else

```
def update_and_activate_child(child_id, new_details) do
  with {:ok, child}         <- Children.find_by(child_id),
       {:ok, updated_child} <- Children.update_details(child, new_details),
       :ok                  <- Children.activate_enrolment(updated_child)
  do
    render(ChildrenView, "success.json", %{data: %{child: updated_child}})
  else
    {:error, error_reason}  -> {:error, :find_or_update_error, error_reason}
    error_reason            -> {:error, :generic_error, error_reason}
  end
end
```

Failures -> else

```
def update_and_activate_child(child_id, new_details) do
  with {:ok, child}         <- Children.find_by(child_id),
       {:ok, updated_child} <- Children.update_details(child, new_details),
       :ok                  <- Children.activate_enrolment(updated_child)
  do
    render(ChildrenView, "success.json", %{data: %{child: updated_child}})
  else
    {:error, error_reason}  -> {:error, :find_or_update_error, error_reason}
    error_reason            -> {:error, :generic_error, error_reason}
  end
end
```

```
def update_iss_credentials(%{"service_id" => service_id} = params) do

  with {:rh, {:ok, reg_holder}} <- {:rh, RegistrationHolders.for_id(service_id)},
       {:creds, {:ok, creds}}   <- {:creds, create_creds(param, reg_holder.id)},
       {:read, {:ok, _}}        <- {:read, read_iss_case_claims(creds)},
       {:upsert, iss_creds}     <- {:upsert, Credentials.upsert(reg_holder.id, params)}
  do
    render(OnboardingView, "iss_creds.json", %{data: %{iss_creds: iss_creds}})
  else
    {:rh, _}     -> {:error, :unprocessable, "Service is not registered with Kickback"}
    {:creds, _}  -> {:error, :auth, "Could not generate credentials”}
    {:read, _}   -> {:error, :auth, "Supplied CCMS Credentials are invalid"}
    {_, error}   -> {:error, :unprocessable, error}
  end

end
```

```
def update_iss_credentials(%{"service_id" => service_id} = params) do

  with {:rh, {:ok, reg_holder}} <- {:rh, RegistrationHolders.for_id(service_id)},
       {:creds, {:ok, creds}}   <- {:creds, create_creds(param, reg_holder.id)},
       {:read, {:ok, _}}        <- {:read, read_iss_case_claims(creds)},
       {:upsert, iss_creds}     <- {:upsert, Credentials.upsert(reg_holder.id, params)}
  do
    render(OnboardingView, "iss_creds.json", %{data: %{iss_creds: iss_creds}})
  else
    {:rh, _}     -> {:error, :unprocessable, "Service is not registered with Kickback"}
    {:creds, _}  -> {:error, :auth, "Could not generate credentials”}
    {:read, _}   -> {:error, :auth, "Supplied CCMS Credentials are invalid"}
    {_, error}   -> {:error, :unprocessable, error}
  end

end
```

```
def update_iss_credentials(%{"service_id" => service_id} = params) do

  with {:rh, {:ok, reg_holder}} <- {:rh, RegistrationHolders.for_id(service_id)},
       {:creds, {:ok, creds}}   <- {:creds, create_creds(param, reg_holder.id)},
       {:read, {:ok, _}}        <- {:read, read_iss_case_claims(creds)},
       {:upsert, iss_creds}     <- {:upsert, Credentials.upsert(reg_holder.id, params)}
  do
    render(OnboardingView, "iss_creds.json", %{data: %{iss_creds: iss_creds}})
  else
    {:rh, _}     -> {:error, :unprocessable, "Service is not registered with Kickback"}
    {:creds, _}  -> {:error, :auth, "Could not generate credentials”}
    {:read, _}   -> {:error, :auth, "Supplied CCMS Credentials are invalid"}
    {_, error}   -> {:error, :unprocessable, error}
  end

end
```

```
def update_iss_credentials(%{"service_id" => service_id} = params) do

  with {:rh, {:ok, reg_holder}} <- {:rh, RegistrationHolders.for_id(service_id)},
       {:creds, {:ok, creds}}   <- {:creds, create_creds(param, reg_holder.id)},
       {:read, {:ok, _}}        <- {:read, read_iss_case_claims(creds)},
       {:upsert, iss_creds}     <- {:upsert, Credentials.upsert(reg_holder.id, params)}
  do
    render(OnboardingView, "iss_creds.json", %{data: %{iss_creds: iss_creds}})
  else
    {:rh, _}     -> {:error, :unprocessable, "Service is not registered with Kickback"}
    {:creds, _}  -> {:error, :auth, "Could not generate credentials”}
    {:read, _}   -> {:error, :auth, "Supplied CCMS Credentials are invalid"}
    {_, error}   -> {:error, :unprocessable, error}
  end

end
```

Thankyou!


```
def contact_drew(handle: "@oheydrew",
                 address: "www.oheydrew.me")
```