Episode 05: Control Flow

Summary

- Control flow
- Pattern matching
- Guards
- “Static typing” with guards

Control flow

Elixir features:

- **cond**: multiple conditions
- **case**: pattern matching against an expression (esp. with tuples)
- **if**: evaluates if condition is `false` or `nil`, returns `nil`
- **unless**: opposite behaviour as `if` blocks

```
cond do
  expr -> code
  true -> default
end

case expr do
  {:ok, value} -> code
  {:error, value} -> code
  _other -> default
end

if expr do
  code
else
  code
end

unless expr do
  code
end
```

Pattern matching

```
defmodule Patterns do
  def blank?(nil),    do: true
  def blank?(false),  do: true
  def blank?(""),     do: true
  def blank?(_other), do: false

  # Alternatively,
  def blank_v2?(value) when value in [nil, false, ""], do: true
  def blank_v2?(_), do: false

  # Even better,
  def blank_v3?(value), do: value in [nil, false, ""]
end
```

Guards

Guards can be used in functions (seen above) and case statements.

```
# In general,
case expr do
  matched_value when ... -> code
  _other -> code
end

# In practice,
case response do
  {:ok, body} ->
    # Success
  {:error, status_code, body} when status_code in 400..499 ->
    # Handle 400 status codes
  {:error, status_code, body} when status_code in 500..599 ->
    # Handle 500 status codes
  _other ->
    # Default case
end
```

“Static typing” with guards

Simple example to assert type of struct:

```
def name(%User{} = user) do
  user.first_name <> " " user.last_name
end

def name(%Episode{name: name}) when is_binary(name) do
  name
end

def name(unsupported) do
  raise "name? does not support #{inspect unsupported}
end
```

The first function works because of pattern matching. To illustrate why, observe that both of the following commands produce “`no match of right hand side value`” errors:

```
%{__struct__: "SomeModule"} = %{}
%{__struct__: "SomeModule"} = %{__struct__: "OtherModule"}
```

They fail because Elixir cannot match `%{__struct__: "SomeModule"}` to look like the thing on the right. Thus, if someone gives our function the wrong type of struct (or no struct at all!) then the pattern match will fail.

The Elixir getting started docs write, “[…] a map matches as long as the keys in the pattern,” i.e., the left hand side, “exist in the given map,” i.e., the right hand side. “Therefore, an empty map matches all maps.”

Now, it is totally fine if the right hand struct has more key/value pairs than the left hand side. What matters is that, **at the minimum**, the right hand side meets the left hand side’s structure, or that the left hand side can be matched to the right.