Level 4–1

Control Flow

The case Statement

Listing Content From a File

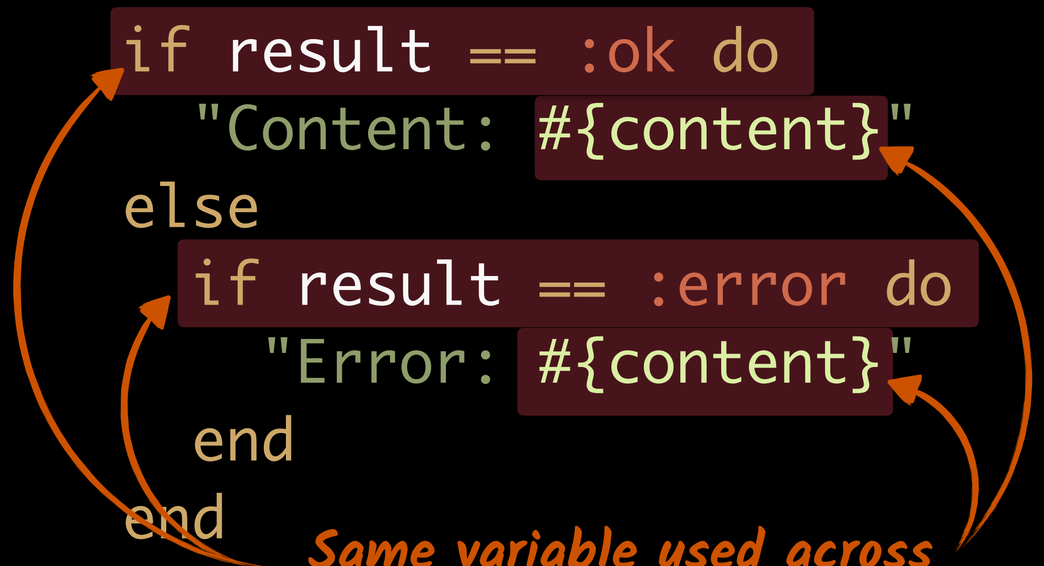
The function Account.list_transactions() **takes a file name as argument and lists its contents.**

```
defmodule Account do
  def list_transactions(filename) do
    { result, content } = File.read(filename)

    if result == :ok do
      "Content: #{content}"
    else
      if result == :error do
        "Error: #{content}"
      end
    end
  end
end
```

Nested if Statements Are Hard to Read

Repeating variables (result, content**) in nested** if **statements illustrate a common code smell.**

```
defmodule Account do
  def list_transactions(filename) do
    { result, content } = File.read(filename)

    if result == :ok do
      "Content: #{content}"
    else
      if result == :error do
        "Error: #{content}"
      end
    end
  end
end
```



Using case to Test Values Against Patterns

The case statement tests a **value** against a set of **patterns**.

```
defmodule Account do
  def list_transactions(filename) do
    { result, content } = File.read(filename)

    case result do
      :ok -> "Content: #{content}"
      :error -> "Error: #{content}"
    end
  end
end
```

Value to be tested...

Return values from successful matches

...patterns to test against

Misleading Variable Names

Using result **as the test value for the** case **statement is leading to the use of the same variable name (**content**) for the content of the file (when** result **is** :ok**) or for the error (when** result **is** :error**).**

```
defmodule Account do
  def list_transactions(filename) do
    { result, content } = File.read(filename)

    case result do
      :ok -> "Content: #{content}"
      :error -> "Error: #{content}"
    end
  end
end
```

Let's use something else here...

This is an error type and NOT the content...

Better Variable Names With case

The case statement accepts tuples for the test values as well as for the patterns to be tested against. This gives us **more flexibility for naming variables**.

```
defmodule Account do
  def list_transactions(filename) do

    case File.read(filename) do
      { :ok, content } -> "Content: #{content}"
      { :error, type } -> "Error: #{type}"
    end
  end
end
```

Test value is a tuple!

Tuples can be used as patterns too!

More meaningful variable name

No Code Smell & Works as Expected

```elixir
defmodule Account do
  def list_transactions(filename) do
    case File.read(filename) do
      { :ok, content } -> "Content: #{content}"
      { :error, type } -> "Error: #{type}"
    end
  end
end
```

```elixir
Account.list_transactions("transactions.csv")
```

```
Content: 01/12/2016,deposit,1000.00
01/12/2016,withdrawal,10.00
01/13/2016,withdrawal,25.00,
...
```

```elixir
Account.list_transactions("does-not-exist")
```

```
Error: enoent
```

Using case with Guard Clauses

The case statement allows extra conditions to be specified with a **guard clause**.

```
defmodule Account do
  def list_transactions(filename) do
    case File.read(filename) do
      { :ok, content }
        when byte_size(content) > 10 -> "Content: (...)"
      { :ok, content } -> "Content: #{content}"
      { :error, type } -> "Error: #{type}"
    end
  end
end
```

built-in function

does not list transactions

returns true when file content is greater than 10 characters.

```
Account.list_transactions("loooong-list.csv")
```

→ `Content: (...)`