Episode 03: Operators

Summary

- Logical operators have interesting behaviour (`or` vs. `||`, truthiness, etc.)
  - `||` can be used to pick a "default" (see the section for an example)
- List operator `in`
- Binary concatenation `<>`
- Binary pattern matching with `=~` and regular expressions

Operators

- Operators are just functions
- Variable names can be **rebound**
  - Changes pointer, not value

Match Operator

- "_" represents a value to be ignored in a pattern
- Use "^" to match a variable's value and not rebind it
- Each variable can only be bound once in a match (unless each occurrence binds to the same value)

Use the match operator to make assertions or extract values.

```
# Extracting values
{animal, age} = {"cat", 5}
%{name: name} = %{name: "Ash", age: 32}
[first|rest] = [1, 2, 3, 4]
"/pages/" <> page_name = "/pages/home"

# Assertions
{:ok, contents} = File.read("file.txt")
%Author{} = map_of_unknown_type
```

Arithmetic

Operator	Meaning
`+`	Add
`-`	Subtract
`*`	Multiply
`/`	Divide
`div/2`	Integer division
`rem/2`	Remainder

Equality Operators

Operator	Meaning
`==`	Equal
`===`	Strictly equal (types)
`!=`	Not equal
`!==`	Not Strictly equal
<, <=, >, `>=`	Inequalities

Sorting order: `number < atom < reference < function < port < pid < tuple < map < list < bitstring`

Logical Operators

Short circuit operators: and, `or`

- Left hand side MUST be `true` or `false`
- Executes right side only if left side is not enough to determine result
    - `or`: if left side is `true`, return `true`; else return right side
    - and: if left side is `false`, return `false`; else return right side

Accepts arguments of any type: `||`, &&, `!`

- All values except `false` and `nil` evaluate to true

- If both arguments are falsey, return second
  - `||`: return first truthy arg, else return second
  - &&: return first falsey arg, else return second

Logical OR for defaults If `user.name` is equal to `nil`, then the name is set to `"John Smith"`. Otherwise, name is set to `user.name`.

```
user = %{name: nil}
name = user.name || "John Smith"
```

List Operators

The `in` operator asserts whether an element is present in a list.

```
"Name" in ["Some", "Names"] # => false
"Peyton" in ["Peyton"] # => true
101 in 'Hello' # => true
```

Combine two lists with `++` (appending is slow).

```
[1, 2, 3] ++ [4] # => [1, 2, 3, 4]
```

Remove members from a list with `--`.

```
[1, 2, 3] -- [1, 3] # => [2]
```

Prepend to a list with |. Combine | with = for complex matches.

```
[0 | [1, 2, 3]] # => [0, 1, 2, 3]

[a, b, c | tail] = [1, 2, 3, 4]
a # => 1
b # => 2
c # => 3
tail # => [4]
```

Binary Operators

Concatenate two binaries with `<>`.

```
"Hello" <> " " <> "World!" # => "Hello World!"
```

Interpolate values in binary with `#{}`.

```
"You found #{div(126, 4)} gold coins."
# => "You found 31 gold coins."

name = "Peyton"
"Hello, #{name}."
```

Compare a binary to a pattern with `=~`.

- RH-side can be a regular expression or a binary
- Return true if LH-side contains or matches RH-side pattern

```
"Goodbye" =~ ~r/Good/ # => true
"Goodbye" =~ "Good" # => true
"Test" =~ "" # => true
```