Level 3

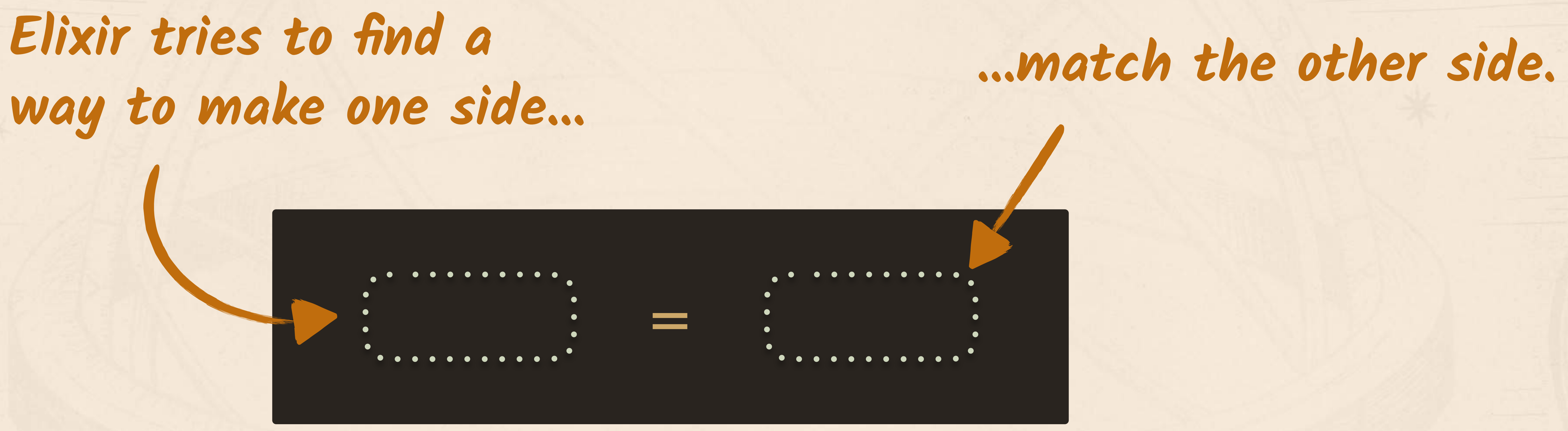
Pattern Matching

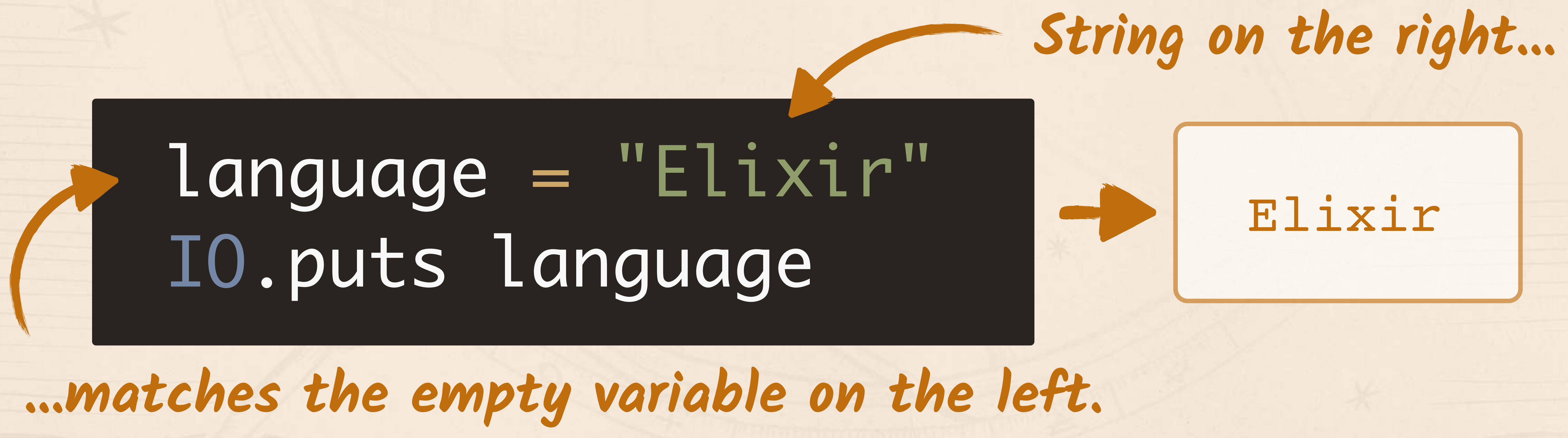
Removing if Statements From Our Program

TRY ELIXIR

The Match Operator

The `=` symbol in Elixir is called the **match operator**. It matches values on one side against corresponding structures on the other side.



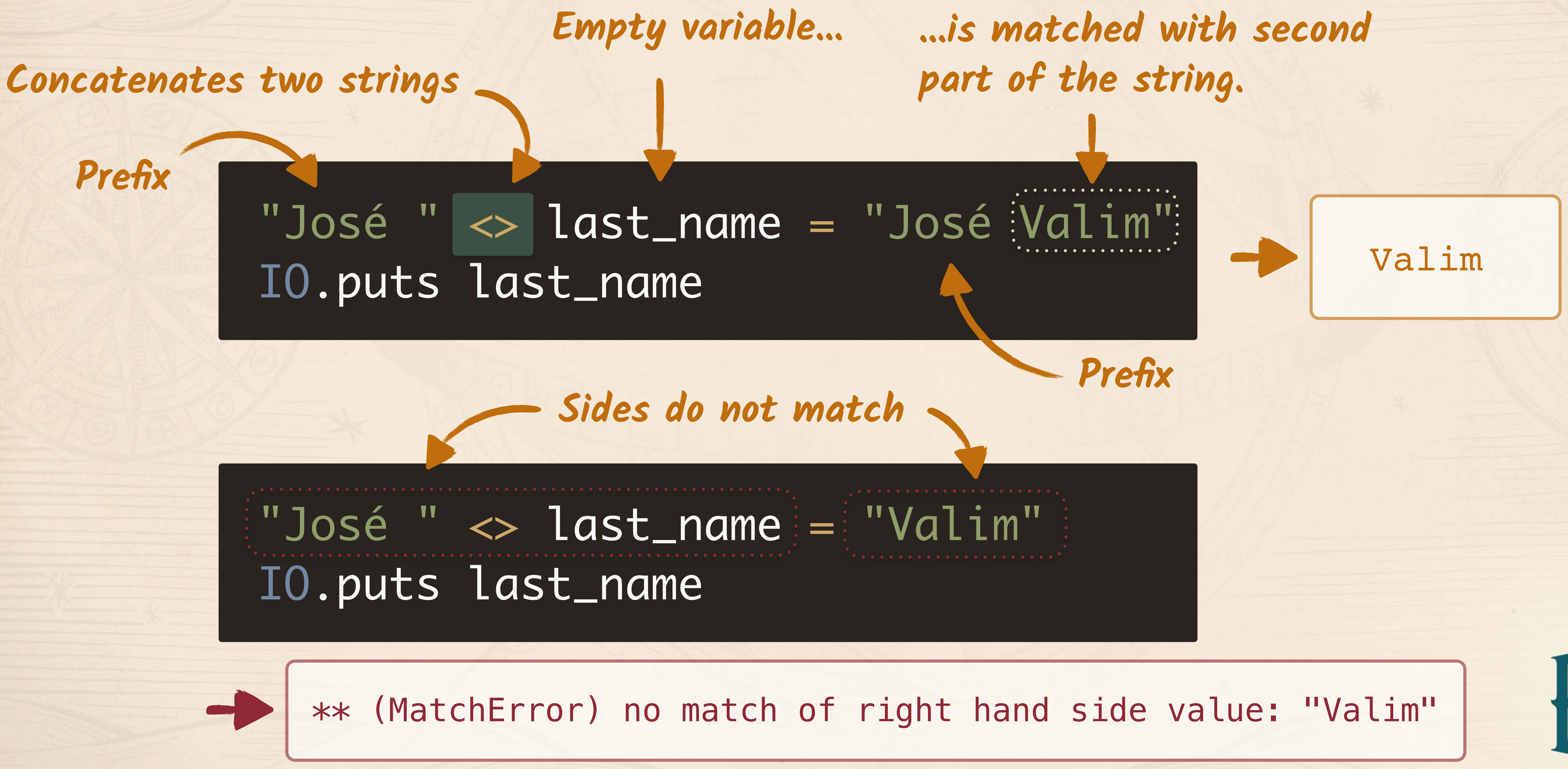


```elixir
language = "Elixir"
IO.puts language
```

```
Elixir
```

Pattern Matching Strings

In this example, we use pattern matching together with the `<>` operator for **string concatenation** to extract a string starting with a given prefix.



```elixir
"José " <> last_name = "José Valim"
IO.puts last_name
```

```
Valim
```

```elixir
"José " <> last_name = "Valim"
IO.puts last_name
```

```
** (MatchError) no match of right hand side value: "Valim"
```

Pattern Matching and Lists

Elixir uses square brackets `[]` to specify a **list** and allows us to use pattern matching to read elements from it.

A new list with two elements

```
data = ["Elixir", "Valim"]
IO.puts data
```

```
ElixirValim
```

Elements from the list on the right...

```
[lang, author] = data

IO.puts "#{lang}, #{author}"
```

```
Elixir, Valim
```

...are a perfect match against those from the list on the left.

TRY ELIXIR

Refactoring Conditionals That Use Arguments

In functional languages like Elixir, the use of if statements is less common than in other languages.

```
defmodule Account do
  def run_transaction(balance, amount, type) do
    if type == :deposit do
      balance + amount
    else
      balance - amount
    end
  end
end
```

Argument being used on conditional

✖

Using a function argument, like type, on an if statement indicates a good opportunity for refactoring

These are atoms, which are similar to strings but more memory efficient

```
Account.run_transaction(1000, 50, :deposit)
```

```
1050
```

```
Account.run_transaction(1050, 30, :withdrawal)
```

```
1020
```

Replacing if Statements With Pattern Matching

Using pattern matching in **function arguments**, we can split functions with if statements into multiple **clauses.**

```elixir
defmodule Account do
  def run_transaction(balance, amount, :deposit) do
    balance + amount
  end

  def run_transaction(balance, amount, :withdrawal) do
    balance - amount
  end
end
```

✓

First clause

Matches first clause

Second clause

Matches second clause

```elixir
Account.run_transaction(1000, 50, :deposit)
```

```
1050
```

```elixir
Account.run_transaction(1050, 30, :withdrawal)
```

```
1020
```

Pattern Matching and the Pipe Operator

Pattern matching and the pipe operator are widely used in Elixir. It's common to use them together, like this:

```elixir
defmodule Account do
  def run_transaction(balance, amount, :deposit) do ...
  def run_transaction(balance, amount, :withdrawal) do
end
```

Receives 1000 as first argument

```elixir
1000
|> Account.run_transaction(50, :deposit)
|> Account.run_transaction(30, :withdrawal)
```

Receives result from previous function call as first argument

```
1020
```