Level 2

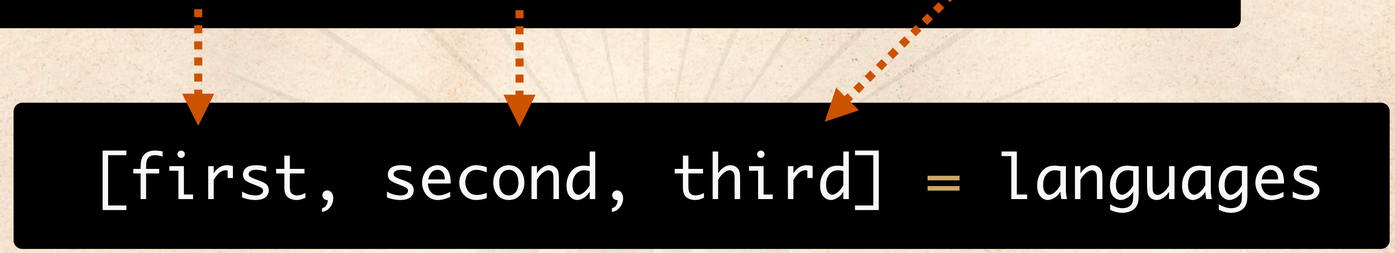
The End Is the Beginning

Lists & Recursion

Reading Elements From a List

We can use pattern matching on lists to read individual elements.

```
languages = ["Elixir", "JavaScript", "Ruby"]
```



```
[first, second, third] = languages
```

However, this does **not** scale well as the list grows...

```
languages = ["Elixir", "JavaScript", "Ruby", "Go"]
[first, second, third, fourth] = languages
```

Splitting a List With the cons Operator

The cons operator `|` is used to split a list into head (first element) and tail (remaining elements).

```
languages = ["Elixir", "JavaScript", "Ruby"]
[head | tail] = languages
```

"Elixir" ["JavaScript", "Ruby"]

Pick the first...

```
languages = ["Elixir", "JavaScript", "Ruby"]
[head | _] = languages
```

...and ignore the rest with no compiler warnings.

Using cons in Function Pattern Matching

The cons operator can be used in function pattern matching to split lists into head and tail.

```
defmodule Language do
  def print_list([head | tail]) do
    IO.puts "Head: #{head}"
    IO.puts "Tail: #{tail}"
  end
end
```

Split single list argument into head and tail

```
Language.print_list(["Elixir", "JavaScript", "Ruby"])
```

```
Head: Elixir
Tail: JavaScriptRuby
```

* MIXING IT UP * with *ELIXIR*

No for Loops

There are no for loops in Elixir. How can we iterate through a list without using a for loop?

```
defmodule Language do
  def print_list([head | tail]) do
    ?????
  end
end
```

Cannot use a loop here

```
Language.print_list(["Elixir", "JavaScript", "Ruby"])
```

```
Head: Elixir
Tail: JavaScriptRuby
```

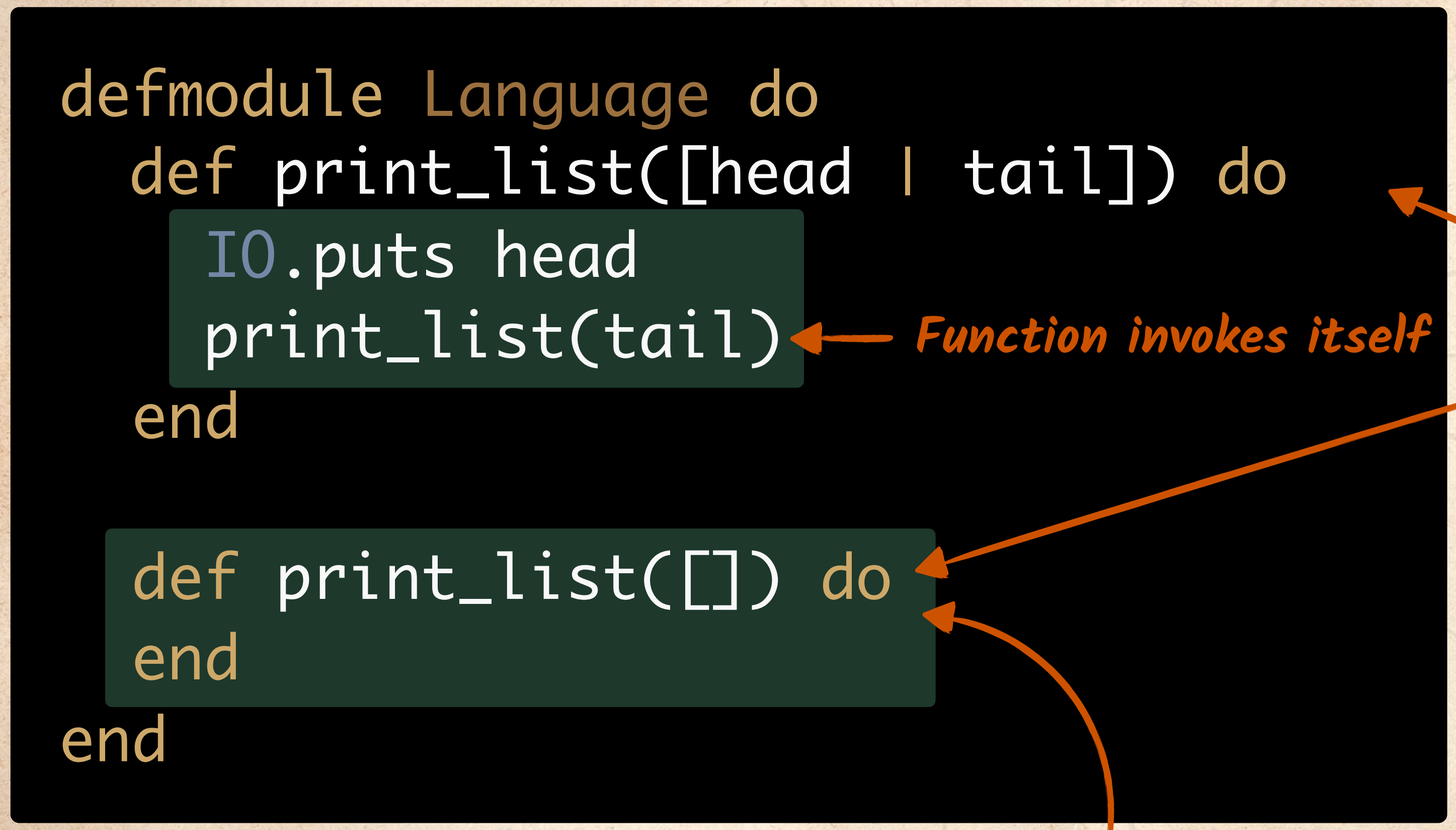
👀 *We see this now...*

...but we want this.

```
Elixir
JavaScript
Ruby
```

Understanding Recursion

Recursive functions are functions that perform operations and then **invoke themselves.**



Two Cases for Recursion

All recursive functions involve the following two cases (or two **clauses**):

1. The base case, also called **terminating scenario**, where the function does NOT invoke itself.

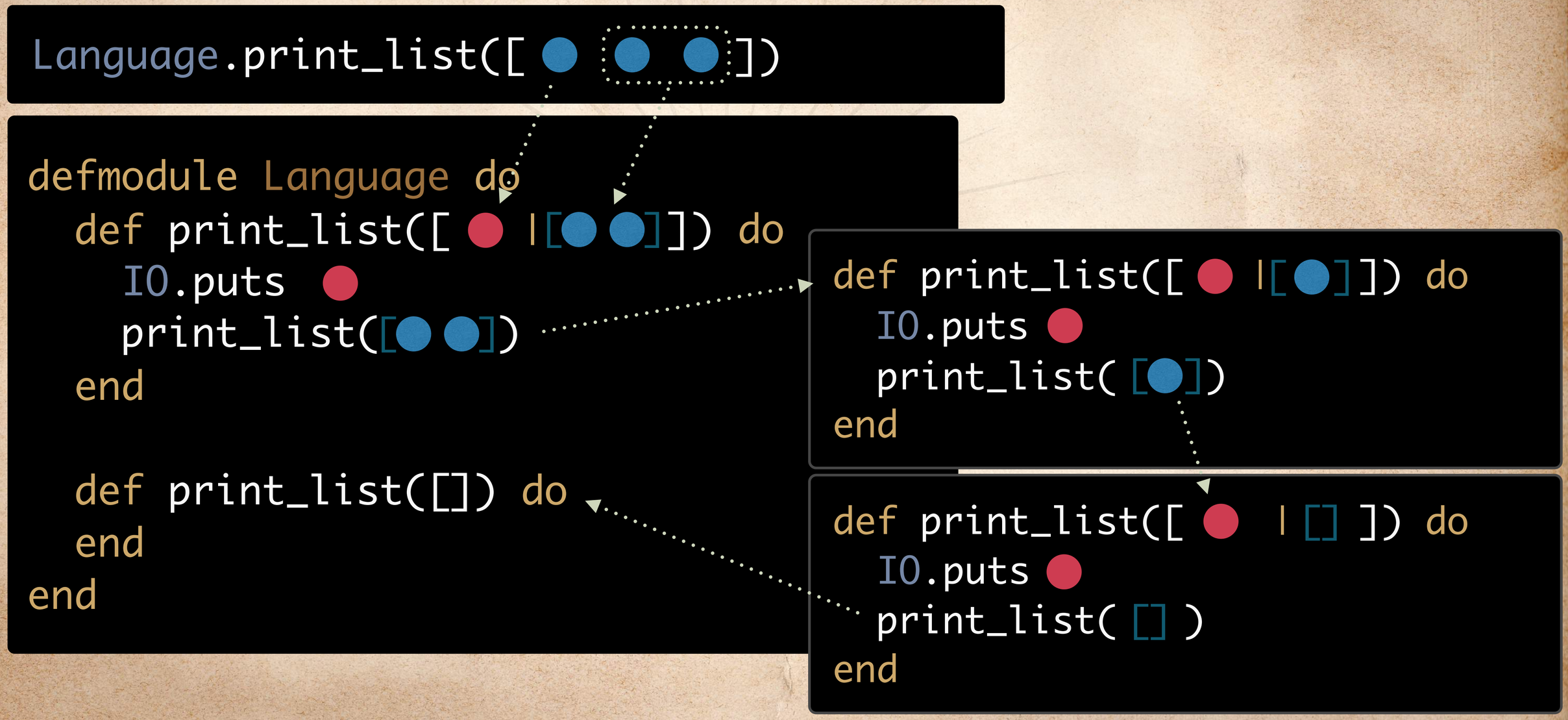
```
def print_list([]) do
end
```

2. The **recursive case**, where computation happens and the function invokes itself.

```
def print_list([head | tail]) do
  IO.puts head
  print_list(tail)
end
```

Loops With Recursion

splitting lists with the cons **operator + pattern matching + recursion = loop**



The Complete Recursive Code

Using recursion, we can now iterate through elements from a list!

```
defmodule Language do
  def print_list([head | tail]) do
    IO.puts head
    print_list(tail)
  end

  def print_list([]) do
  end
end
```

```
Elixir
JavaScript
Ruby
```

```
Language.print_list(["Elixir", "JavaScript", "Ruby"])
```