nctions and Modules

Getting Started with Elixir

João Gonçalves

Section 4

Functions

www.packtpub.com

Packt>

In this Section, we are going to take a look at...

- Definition of functions and modules
- Pattern matching in functions
- Anonymous functions

João Gonçalves

Video 4.1

Functions and Modules

In this video, we are going to take a look at...

- What is a function
- Defining functions in Elixir
- How to call functions
- Chaining function calls
- Modules – Containers of functions

What is a Function?

$$f(x) \rightarrow y$$

What is a Function?

$$\underbrace{f}_{\text{“Name”}}(\underbrace{x}_{\text{Arguments}}) \to \underbrace{y}_{\text{Return value}}$$

Why Functions?

- Reuse computations
- Combine to express more powerful computations

A Function in Elixir

```
def say_hello() do
  :hello
end
```

A Function in Elixir

"Name"

Arguments

```
def say_hello() do
  :hello
end
```

Body & Return value

Calling a Function

$$\begin{tikzcd}
\texttt{say\_hello()} \arrow[d] \\
\texttt{:hello}
\end{tikzcd}$$

Function Notation

- Elixir allows the definition of functions with the same name but with different arity

$\texttt{say\_hello/0} \longleftarrow$ Arity

Default Arguments

```
def say_hello(name) do
  “Hello #{name}”
end
```

Default Arguments

```
def say_hello(name\\ "you") do
  "Hello #{name}"
end
```

Chaining Function Calls

```
def person do
  %{first_name: “Joe”, last_name: “Smith”}
end

def full_name(person) do
  “#{person.first_name} #{person.last_name}”
end

def say_hello(name, from) do
  “#{from} says: Hello #{name}!”
end
     say_hello(full_name(person), “Jeff”)
```

Chaining Function Calls

```
def person do
  %{first_name: "Joe", last_name: "Smith"}
end

def full_name(person) do
  "#{person.first_name} #{person.last_name}"
end

def say_hello(name, from) do
  "#{from} says: Hello #{name}!"
end
```

```
person |> full_name |> say_hello("Jeff")
```

Chaining Function Calls

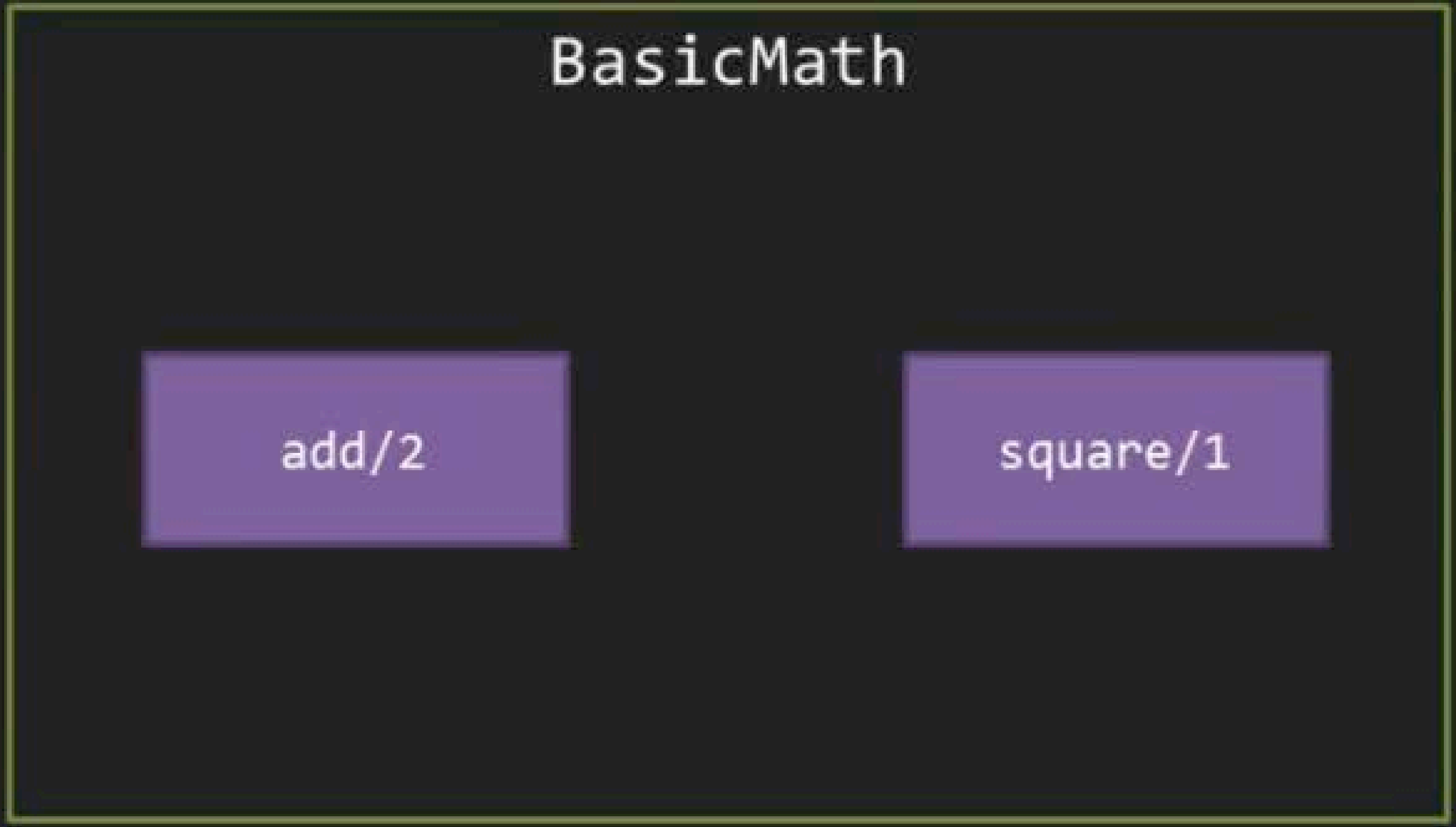
- Injects the value on the left as the first argument of the function on the right

"Pipe" operator

```
person |> full_name |> say_hello("Jeff")
```

Modules

- A group of closely related functions



Modules in Elixir

```
defmodule BasicMath do
  def add(x,y), do: x + y
  def square(x) do
    x * x
  end
end
```

Calling a Function from a Module

```
BasicMath.add(2,5)
```

$\downarrow$

```
7
```

Composing Modules

```
defmodule ComplexMath do

  def cube(x) do
    BasicMath.square(x) * x
  end
end
```

Composing Modules

```
defmodule ComplexMath do
  alias BasicMath, as: Math
  def cube(x) do
    Math.square(x) * x
  end
end
```

Composing Modules

```
defmodule ComplexMath do
  import BasicMath
  def cube(x) do
    square(x) * x
  end
end
```

Composing Modules

```
defmodule ComplexMath do
  import BasicMath, only: [square: 1]
  def cube(x) do
    square(x) * x
  end
end
```

Composing Modules

`alias`

Reference a module by a different name

`import`

Include the functions of a module

Private Functions

```
defmodule Example do
  def hello, do: say_hello
  def say_hello do
    :hello
  end
end
```

Private Functions

Only visible by functions in the module →

```
defmodule Example do
  def hello, do: say_hello
  defp say_hello do
    :hello
  end
end
```

Constants

```
defmodule Example do
  @hello = :hello
  def hello do
    @hello
  end
end
```

Constants

Constant

```
defmodule Example do
  @hello = :hello
  def hello do
    @hello
  end
end
```

Value is bound at compile-time