Elixir: Scalable and Efficient Application Development

João Gonçalves

Pattern Matching Versus Assignment

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

- What is pattern matching
- How does it differ from assignment

```
list = [1,2,3,4,5]
```

Packt>

```
list = [1,2,3,4,5]
```

Assign the value of the list to the variable named `list`

```
x = 1
```

We can do this with any type

$$x = 1$$

$$1 = x$$

$$x = 1$$

$$1 = x$$

$$1 = 1$$

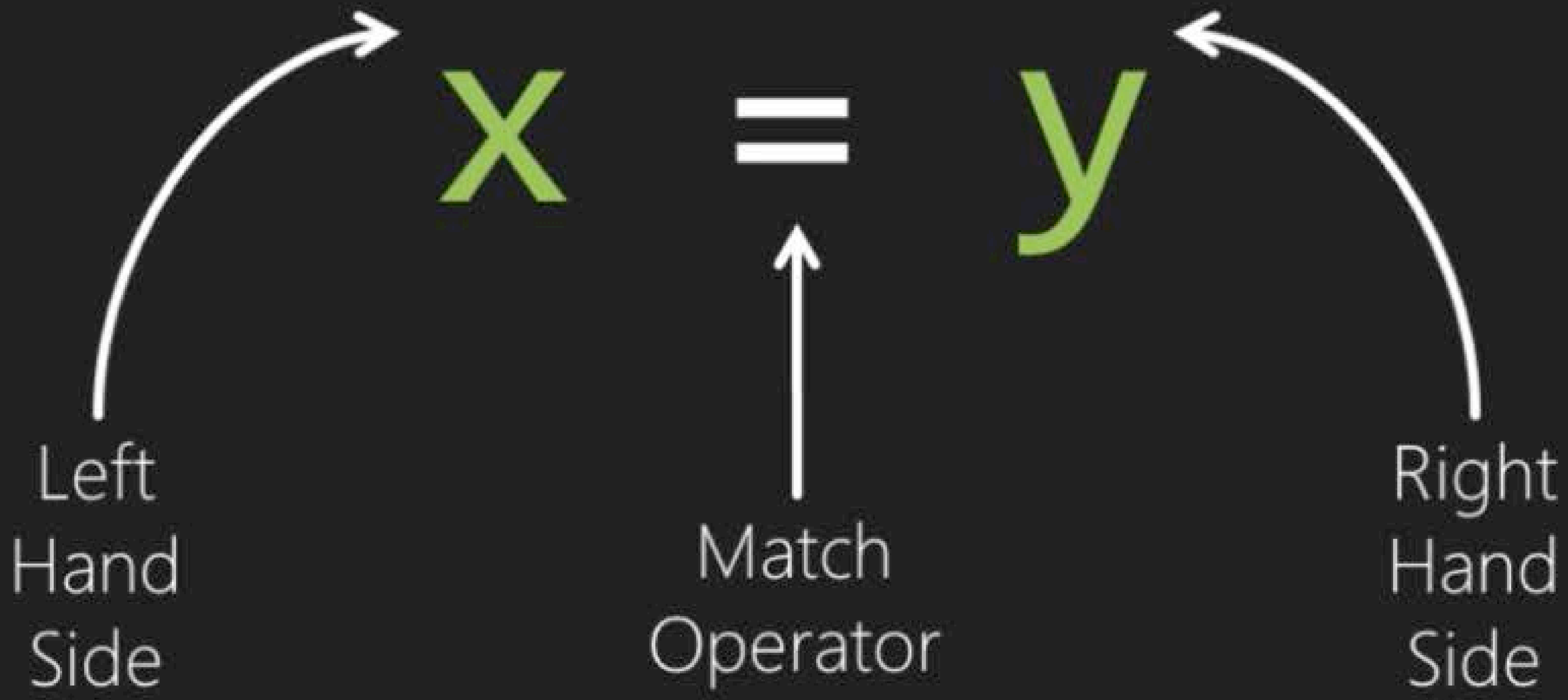
The Match Operator

$$x \quad = \quad y$$

$\uparrow$

Match
Operator

The Match Operator

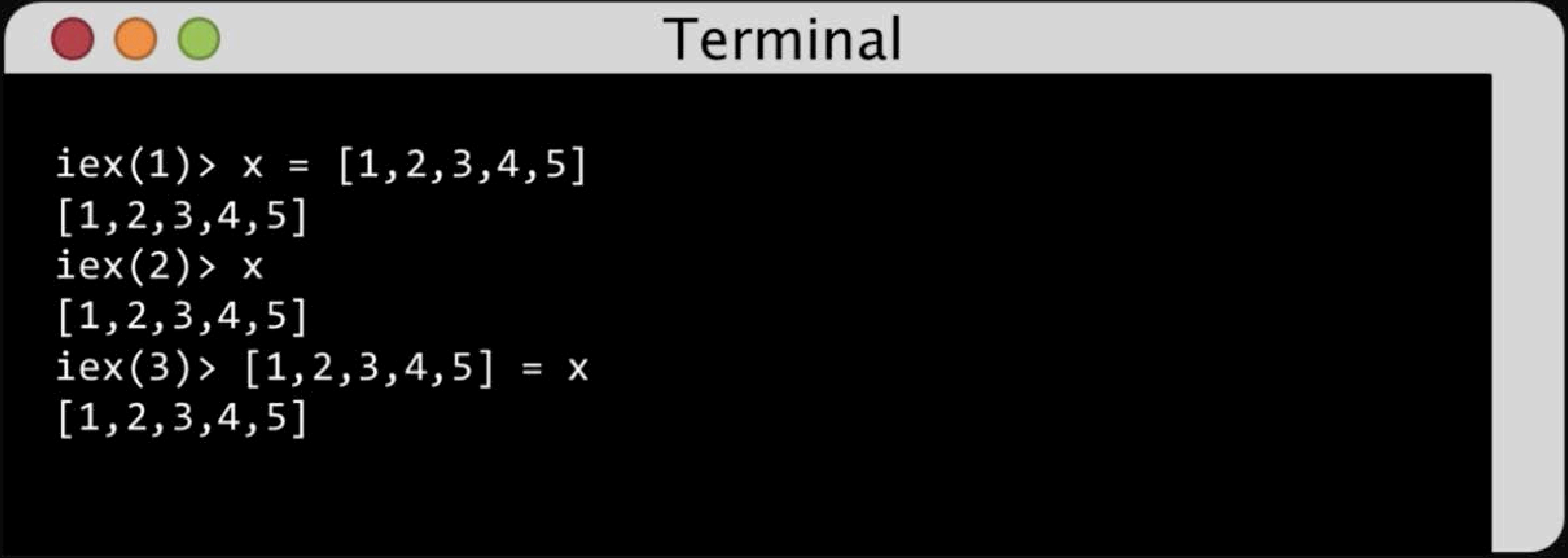


The Match Operator

$$x = y$$

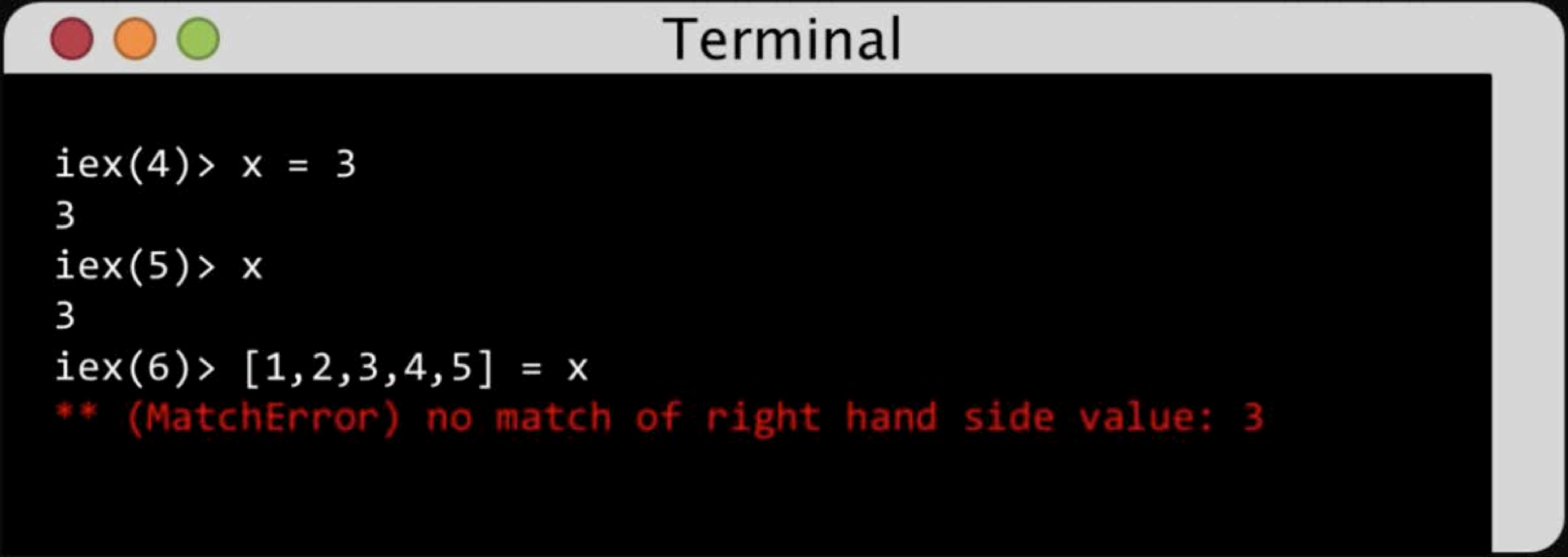
Match the left hand side
with the value on the right hand side

The Match Operator



```
iex(1)> x = [1,2,3,4,5]
[1,2,3,4,5]
iex(2)> x
[1,2,3,4,5]
iex(3)> [1,2,3,4,5] = x
[1,2,3,4,5]
```

The Match Operator



```
iex(4)> x = 3
3
iex(5)> x
3
iex(6)> [1,2,3,4,5] = x
** (MatchError) no match of right hand side value: 3
```

Matching Rules

$$x = 1$$

$$\uparrow$$

If the left hand side contains a variable,
the variable on the right hand side
is bound to the variable

Matching Rules

$$x = y$$
$$\uparrow$$

If the right hand side contains a name,
the left hand side is matched to the
value of the variable with that name or
a function with the same name, if it exists

Matching Rules

```
x = 1
x = 2
```

A variable can be "re-assigned" with a different value on a subsequent match

Pin Operator

Pin Operator $\longrightarrow$ `^x = y`

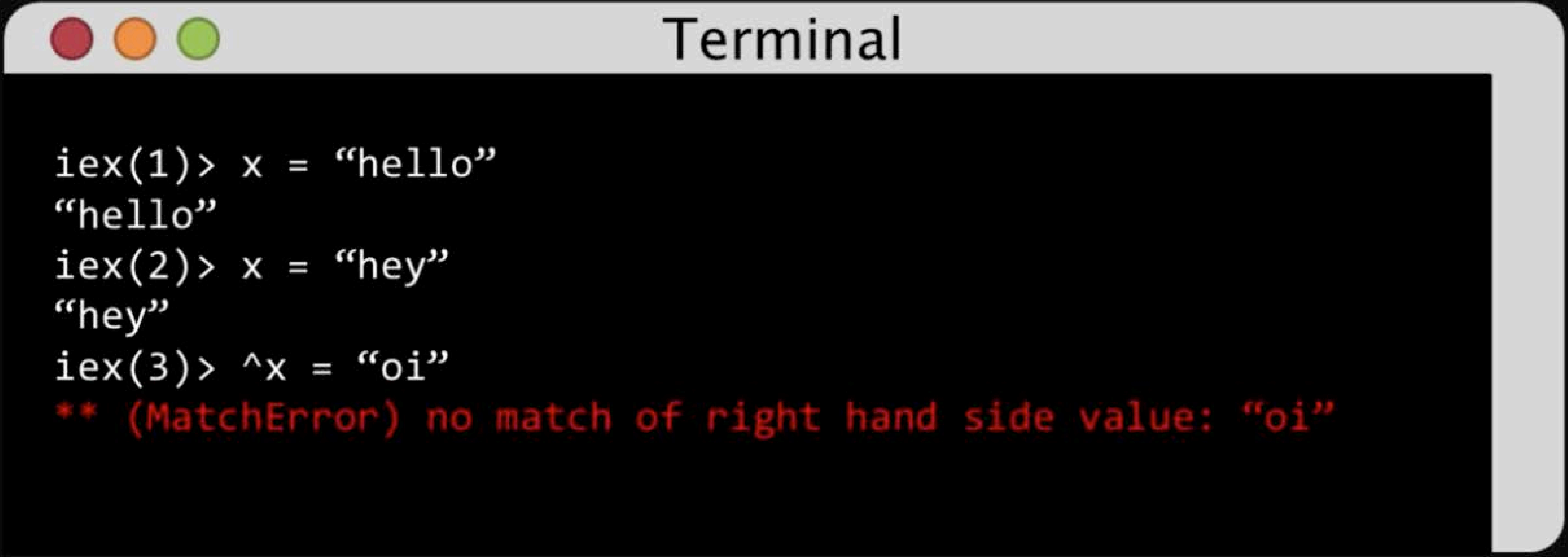
Pin Operator

$$\text{Pin Operator} \longrightarrow \hat{}x = y$$

Strict check for a match, no binding of variables

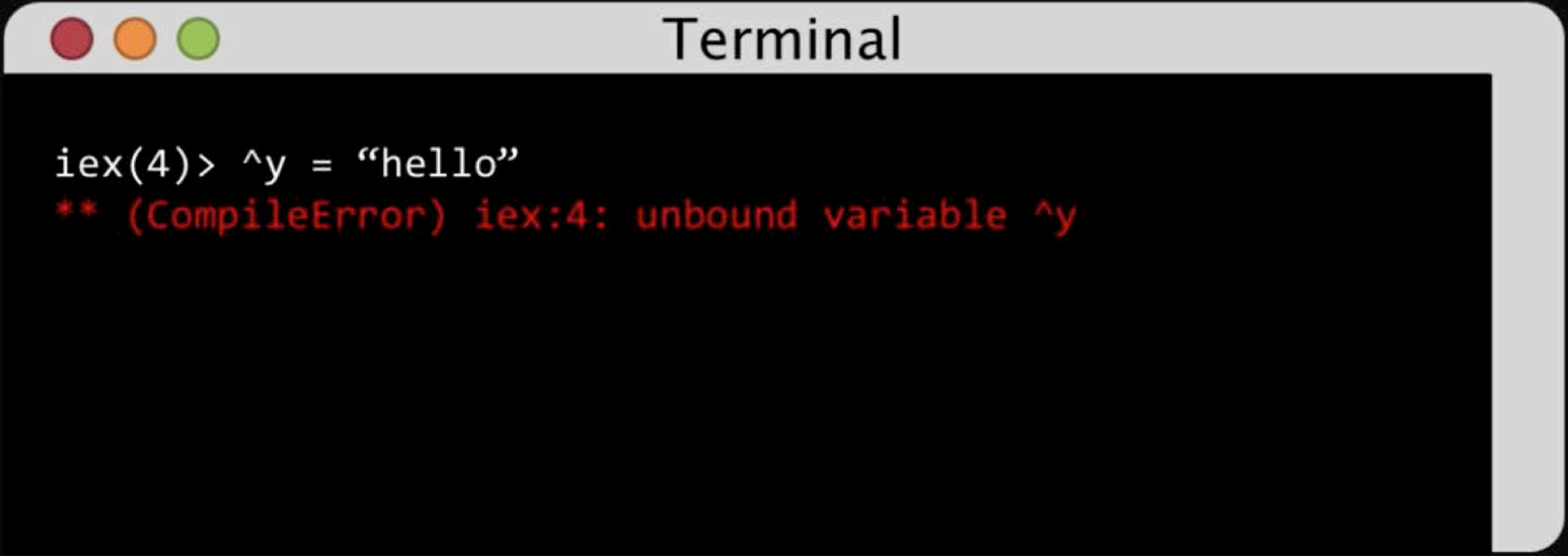
Packt>

Pin Operator



```
iex(1)> x = “hello”
“hello”
iex(2)> x = “hey”
“hey”
iex(3)> ^x = “oi”
** (MatchError) no match of right hand side value: “oi”
```

Pin Operator



```
iex(4)> ^y = “hello”
** (CompileError) iex:4: unbound variable ^y
```

Forms of pattern matching
Section 3, Lecture 7

Go to Dashboard

Elixir: Scalable and Efficient Application Development

João Gonçalves

Forms of Pattern Matching

0:01 /6:00

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

- Types of pattern matching
- Binary types
- Leveraging pattern matching

```
x = 1
```

Pattern Matching

$x = 1$

Pattern Matching

Pattern Matching Revisited

```
[1,2,3] = [1,2,3]
```

Pattern Matching Revisited

```
[x,y,z] = [1,2,3]
```

$$\begin{tikzcd} x \arrow[d, bend right] & y \arrow[d] & z \arrow[d, bend left] \\ 1 & 2 & 3 \end{tikzcd}$$

Pattern Matching Revisited

```
[x,2,3] = [1,2,3]
```

Pattern Matching Revisited

```
[x|[2,3]] = [1,2,3]
```

Pattern Matching Revisited

$$\begin{tikzcd}[column sep=tiny] {[} & x \arrow[dl, bend right] & \mid & y \arrow[dr, bend left] & {]} & = & {[1,2,3]} \\ 1 & & & & {[2,3]} \end{tikzcd}$$

Pattern Matching Revisited

```
{:ok, result} = {:ok, 10}
```

Pattern Matching Revisited

```
{name, age} = {“Francis”, 30}
```

Pattern Matching Revisited

{name, ~~age~~} = {“Francis”, 30}

Pattern Matching Revisited

```
{name, _} = {“Francis”, 30}
```

The underscore matches anything
and is an unreadable variable

Pattern Matching Revisited

```
%{name: name} = %{name: “Francis”, age: 30}
```

Pattern Matching Revisited

```
%{name: name} = %{name: “Francis”, age: 30}
```

↓

Matches as long as the key is present on the right hand side

Packt›

A New Type

- Binary list

```
<<1,2,3>>
```

A New Type

```
<<1,2,3>>
```

1 byte

A New Type

```
<<1,2,3>> <> <<4>>
```

$\uparrow$

concatenation

A New Type

```
<<1,2,3,4>>
```

A New Type

```
<<1::size(16)>>
```

Binaries can be tagged with a size atribute

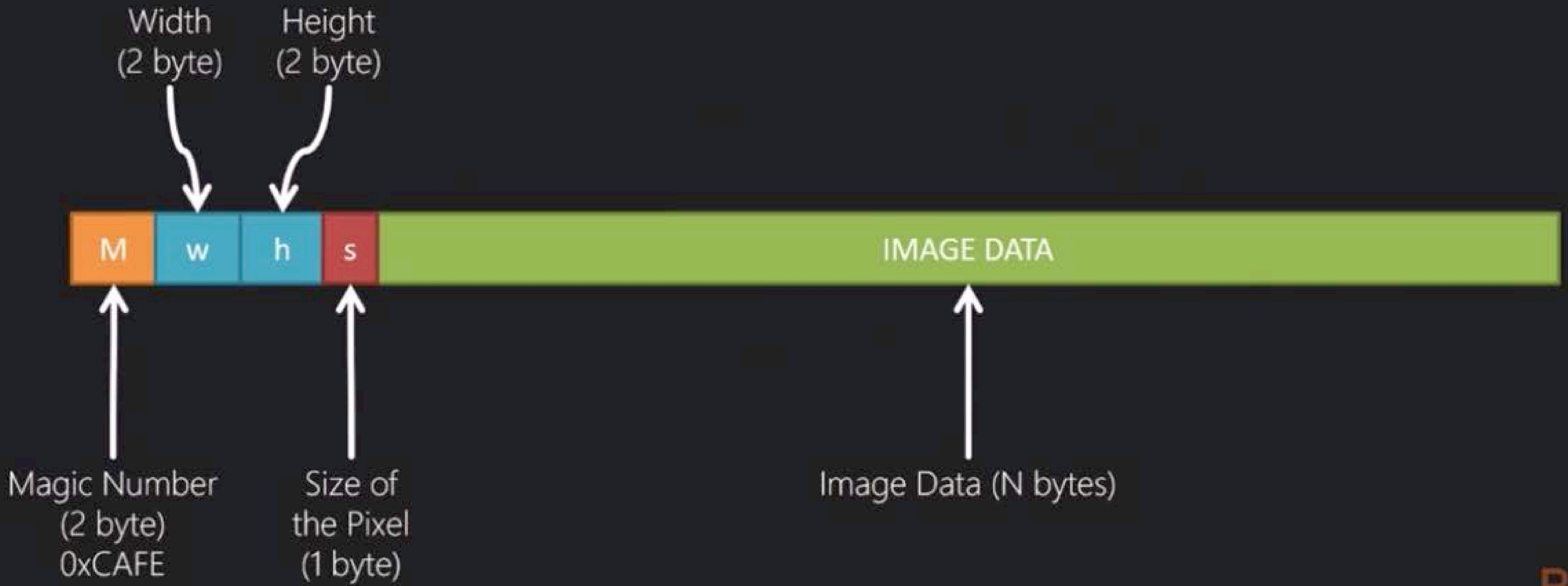
A New Type

```
<<65,"broad">>
```

```
"Abroad"
```

A New Problem

FIF (Fictitious Image Format)



A New Problem

FIF (Fictitious Image Format)

```
<<
  0xCAFE::16,
  width::16,
  height::16,
  pixel_size,
  image_data::binary
>>
```

Packt>

A New Problem

FIF (Fictitious Image Format)

```
<<
  0xCAFE::16,
  width::16,
  height::16,
  pixel_size,
  image_data::binary
>>
```

Can only be used at the end of the pattern

A New Problem

FIF (Fictitious Image Format)

```
<<
  0xCAFE::16,
  width::16,
  height::16,
  pixel_size,
  image_data::binary
>> = << ... >>
```

Quiz Time!

Quiz 3 | 2 Questions

Start Quiz Skip Quiz ›

Question 1:

Which of the following are the uses of pattern matching?

○ It is used to calculate the size of pixel

○ It is used to calculate index numbers

○ It is used to concatenate the binary numbers

○ It is used to retrieve specific information from complex data structures

Question 2:

How is the match operator represented?

- ○ `:==`
- ○ `==`
- ○ `=`
- ○ `=:`