llection Types

Getting Started with Elixir

João Gonçalves

Video 2.2

Collection Types

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

- Available collection types
- Functions to manipulate collections
- Imutability
- Type composition

Lists

- An ordered collection of items
- Delimited by square brackets $\underbrace{[\quad]}$
- An empty list [ ]

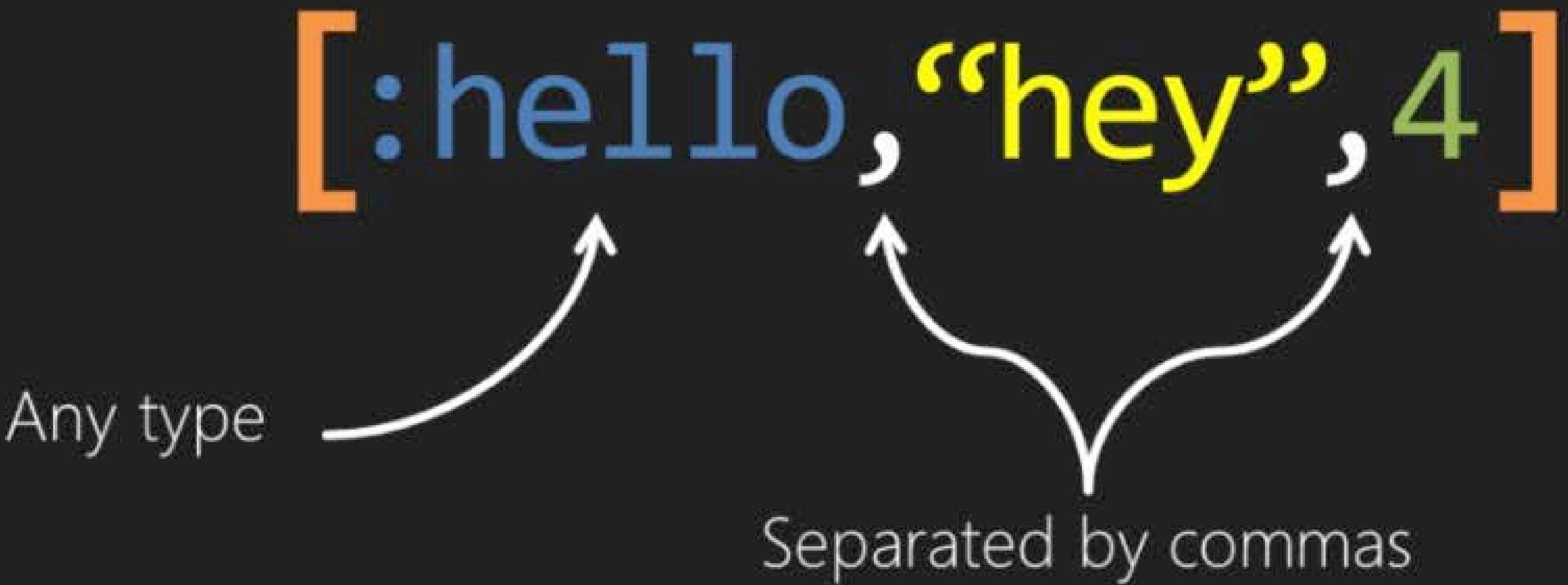
Lists

```
[:hello,"hey",4]
```

ind 5s

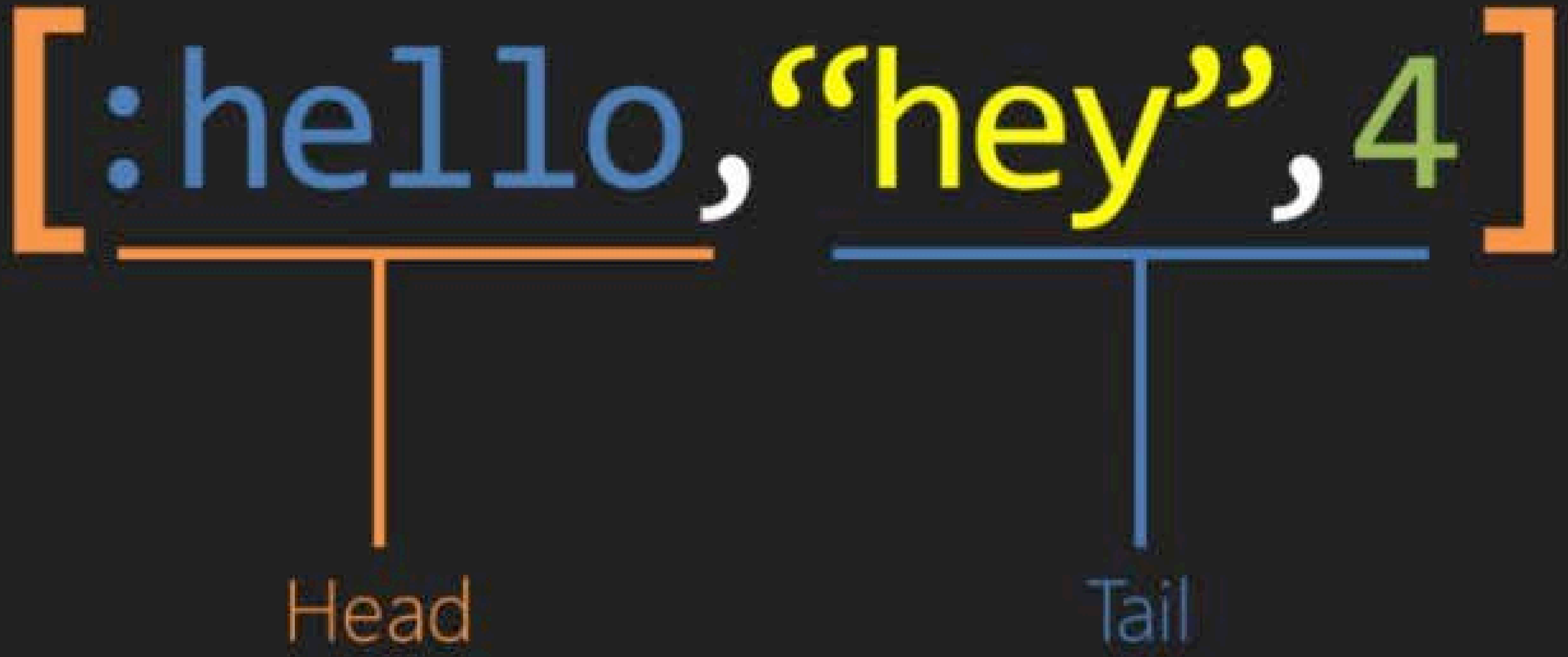
1× 1:00 / 12:31

Lists

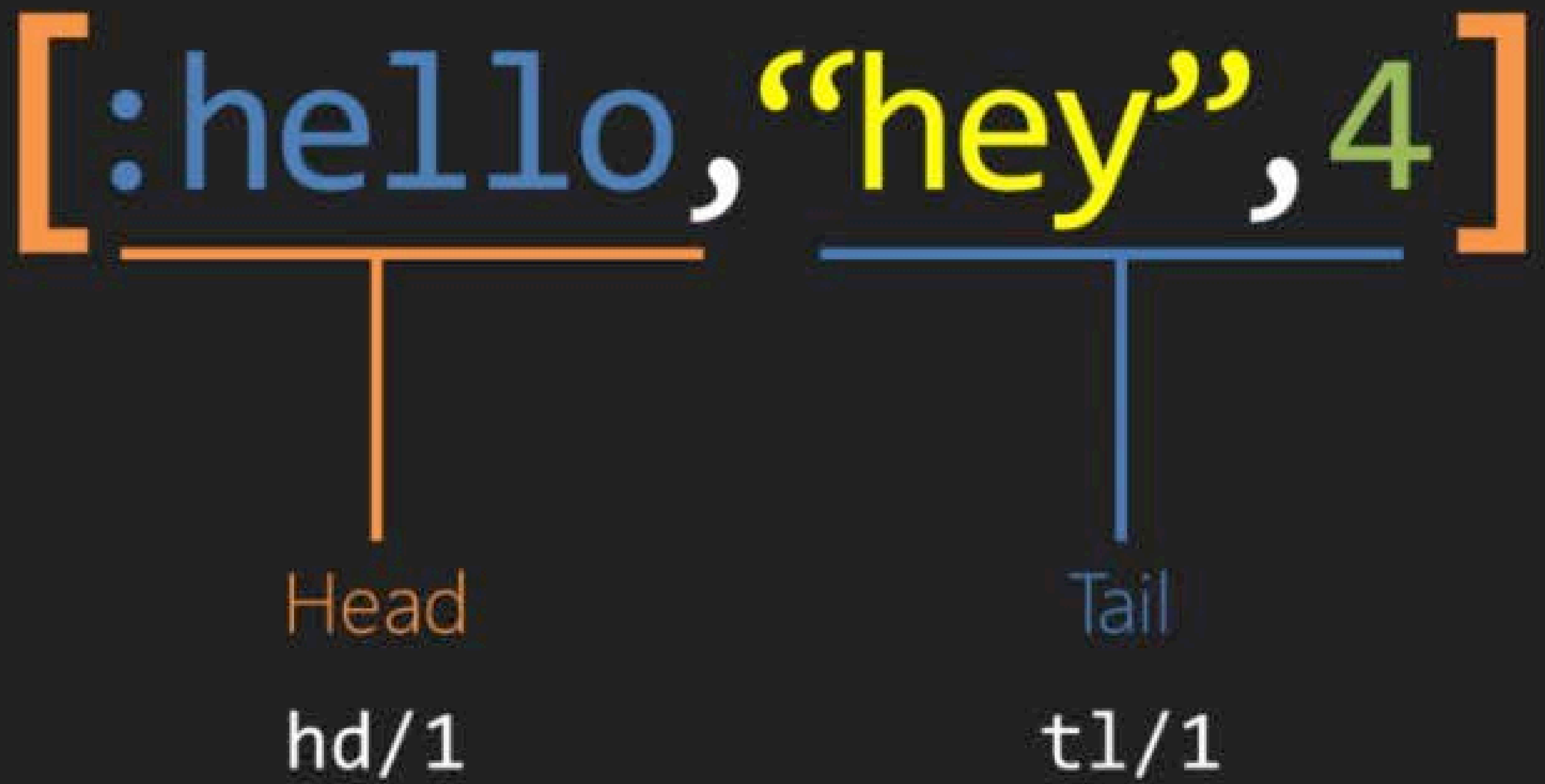


```
[:hello, "hey", 4]
```

Lists

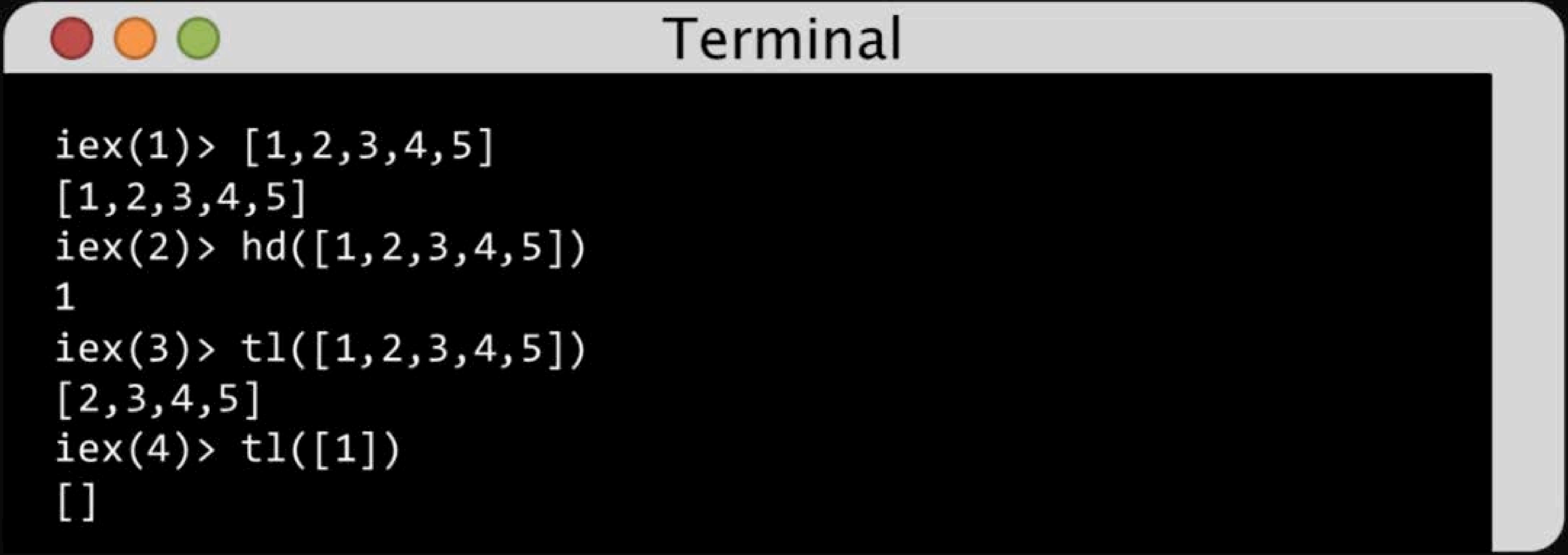


Lists



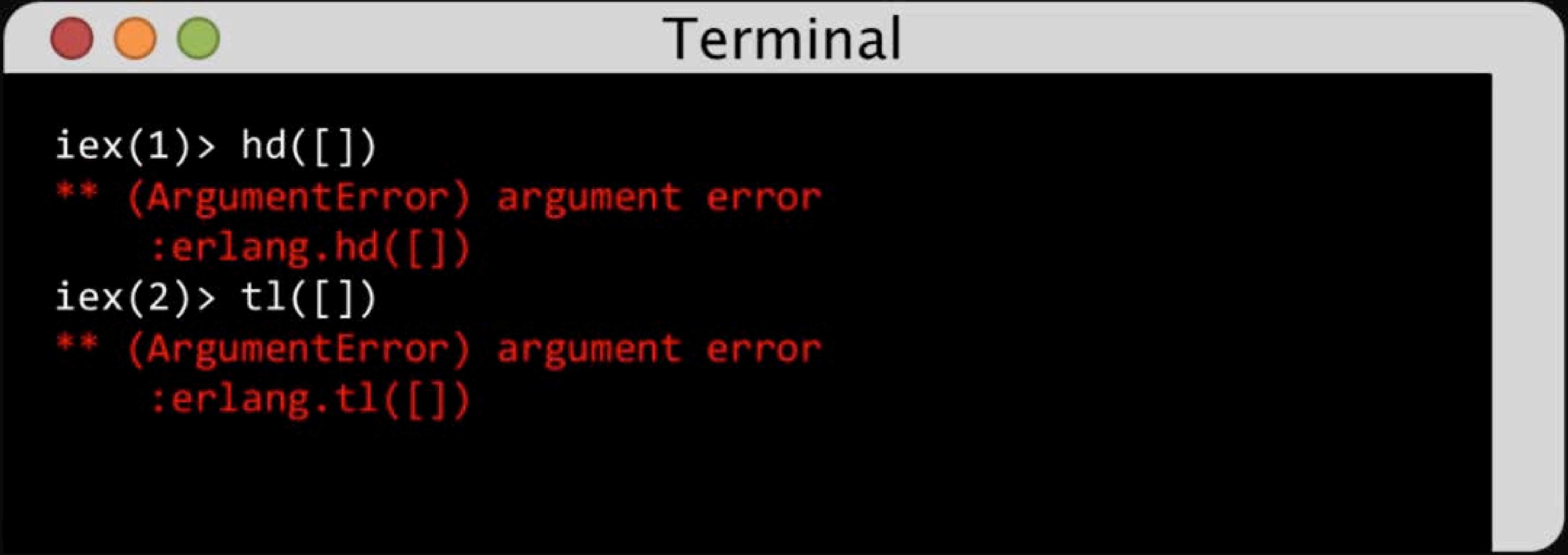
```
[:hello,“hey”,4]
```

Lists – Head and Tail


Terminal

```
iex(1)> [1,2,3,4,5]
[1,2,3,4,5]
iex(2)> hd([1,2,3,4,5])
1
iex(3)> tl([1,2,3,4,5])
[2,3,4,5]
iex(4)> tl([1])
[]
```

Lists – Head and Tail



```
iex(1)> hd([])
** (ArgumentError) argument error
    :erlang.hd([])
iex(2)> tl([])
** (ArgumentError) argument error
    :erlang.tl([])
```

Lists – The Cons Cell

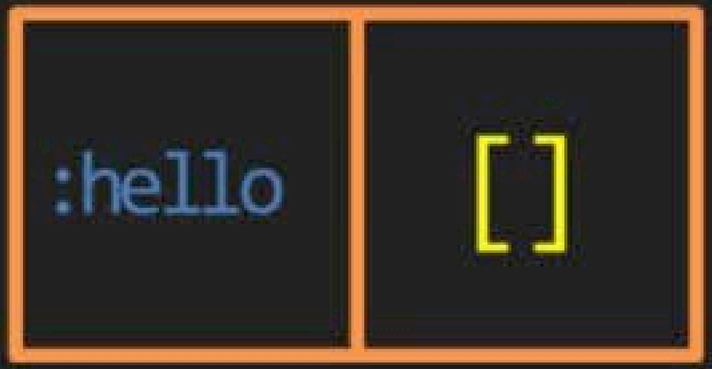
```
[:hello|[]]
```

Two elements separated by a "Pipe"

"Cons" Cell

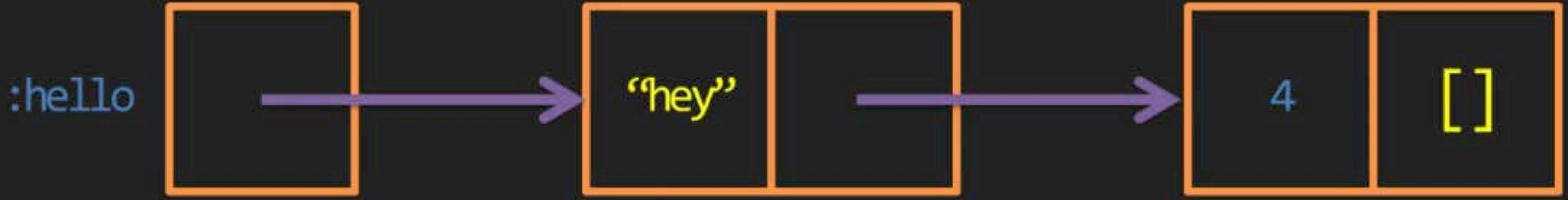
Lists – The Cons Cell

Lists – The Cons Cell



```
[:hello]
```

Lists – The Cons Cell

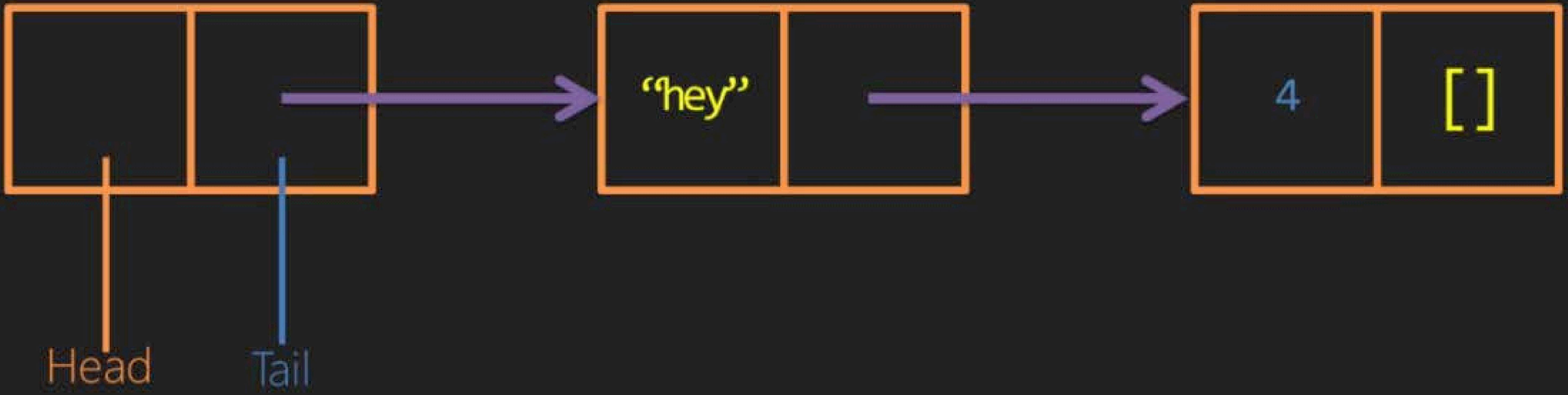


```
[:hello|[“hey”|[4|[]]]]
```

```
[:hello,“hey”,4]
```

Lists – The Cons Cell

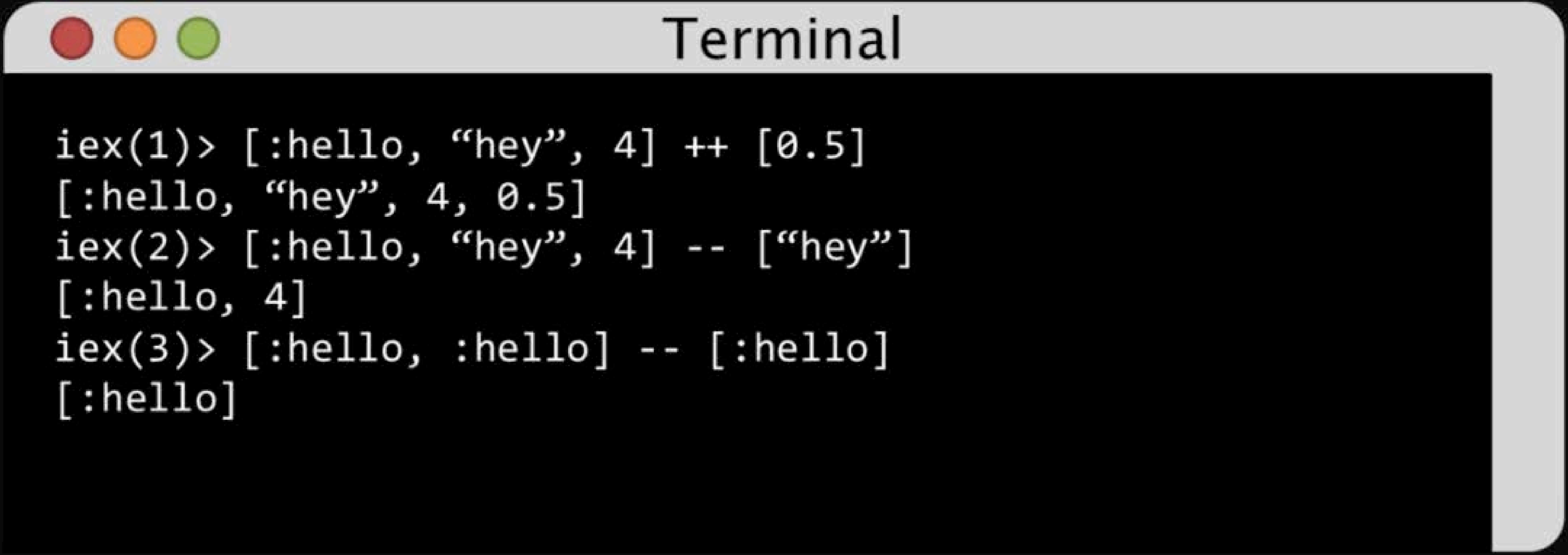
- Linked List



Lists – Operations

- Concatenation
    - ++
- Subtraction
    - --

Lists – Operations



```
iex(1)> [:hello, “hey”, 4] ++ [0.5]
[:hello, “hey”, 4, 0.5]
iex(2)> [:hello, “hey”, 4] -- [“hey”]
[:hello, 4]
iex(3)> [:hello, :hello] -- [:hello]
[:hello]
```

Tuple

- An ordered collection of items

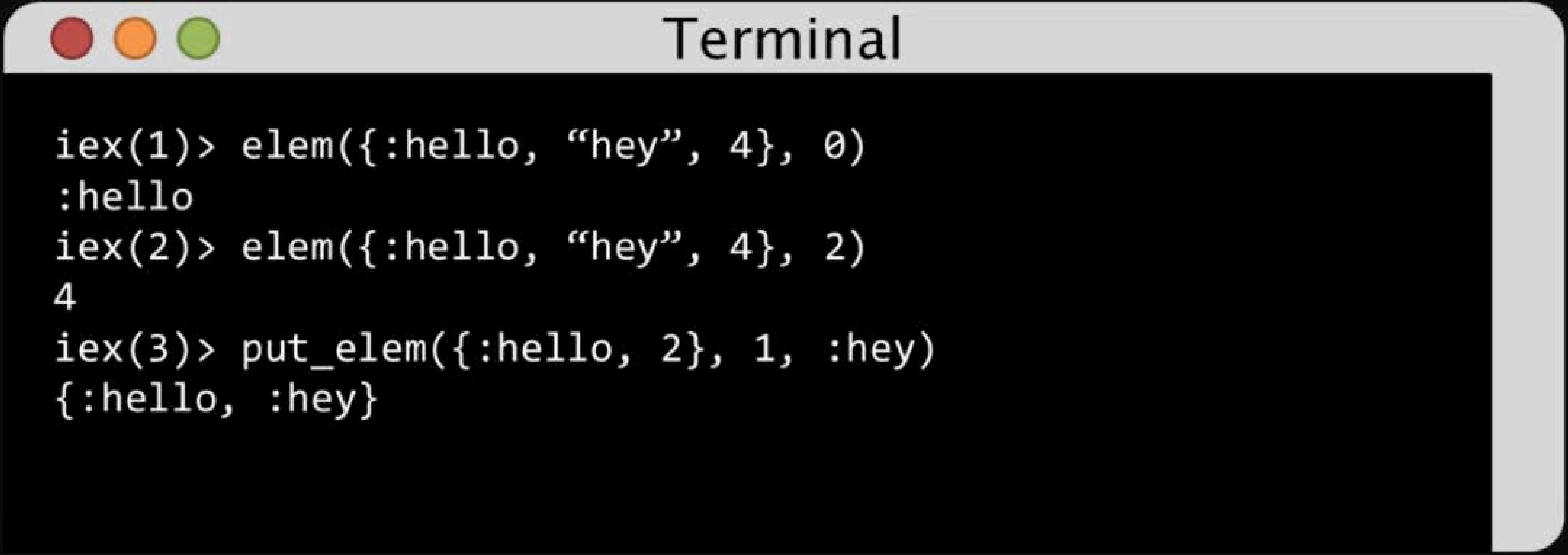
Tuple

```
{:hello, "hey", 4}
```

Tuples - Functions

- Index
  - `elem/2`
- Size
  - `tuple_size/1`
- Replacement
  - `put_elem/3`

Tuples - Functions



```
iex(1)> elem({:hello, “hey”, 4}, 0)
:hello
iex(2)> elem({:hello, “hey”, 4}, 2)
4
iex(3)> put_elem({:hello, 2}, 1, :hey)
{:hello, :hey}
```

Lists versus Tuples

List

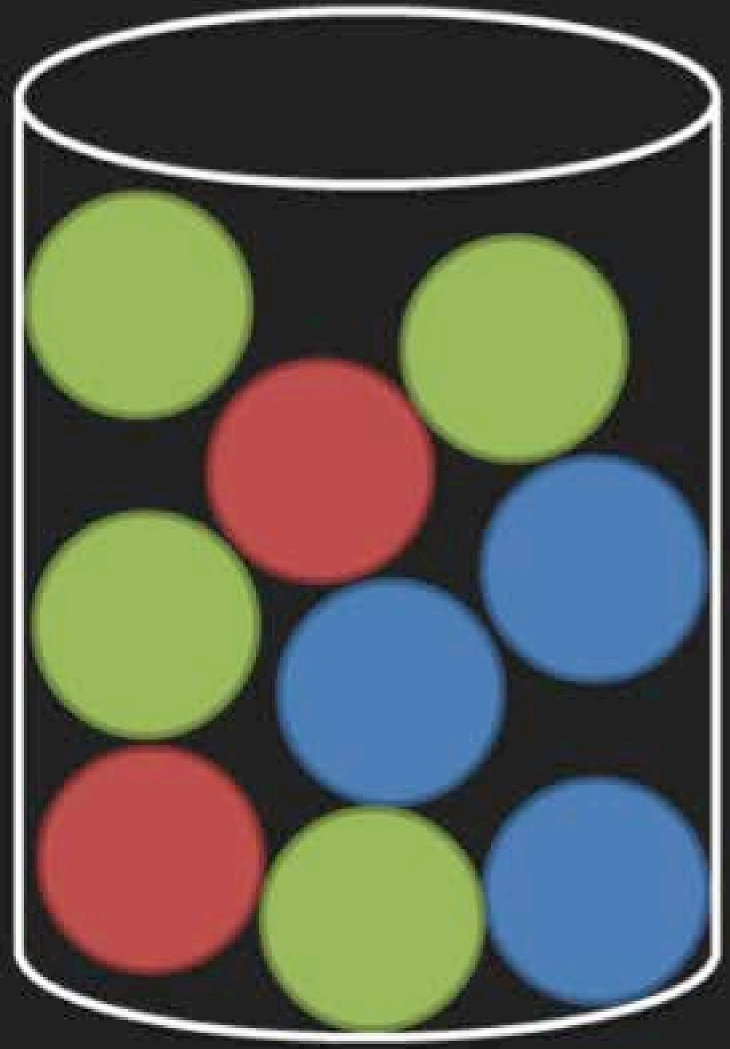
?

Tuple

Lists versus Tuples

	List	Tuple
Structure	Linked list	Contiguous memory
Insertion	Fast (prepending)	Expensive
Size	Slow	Fast
Fetch by index	Slow	Fast
Fetch first	Fast	Fast

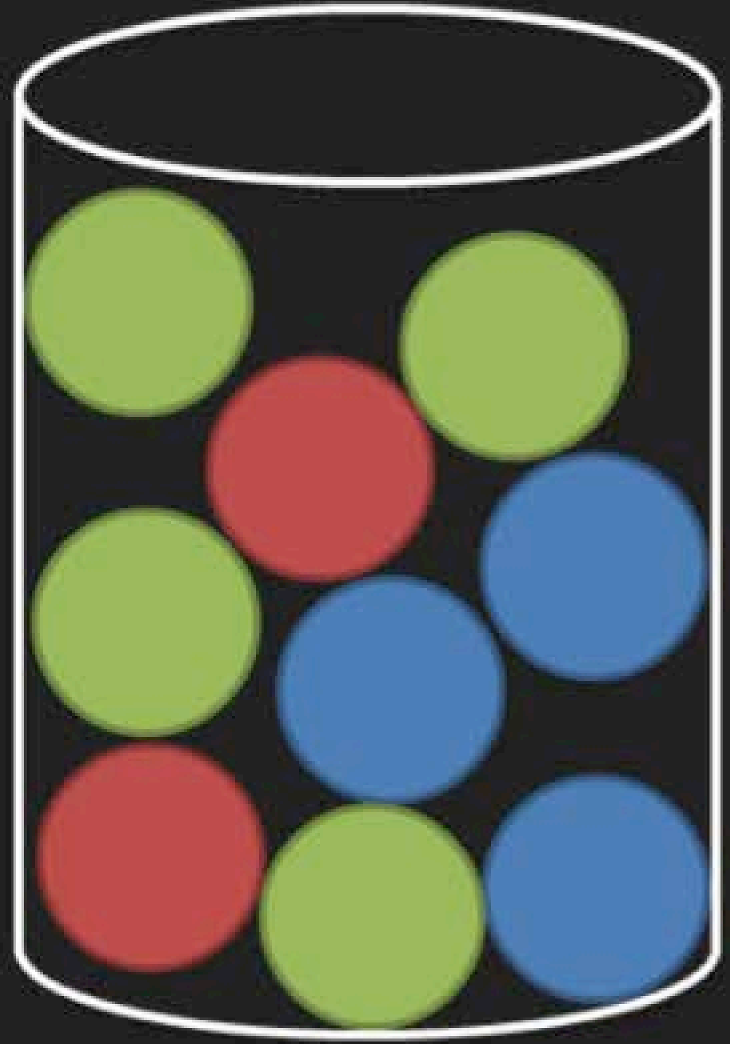
Keyword Lists



Keyword Lists

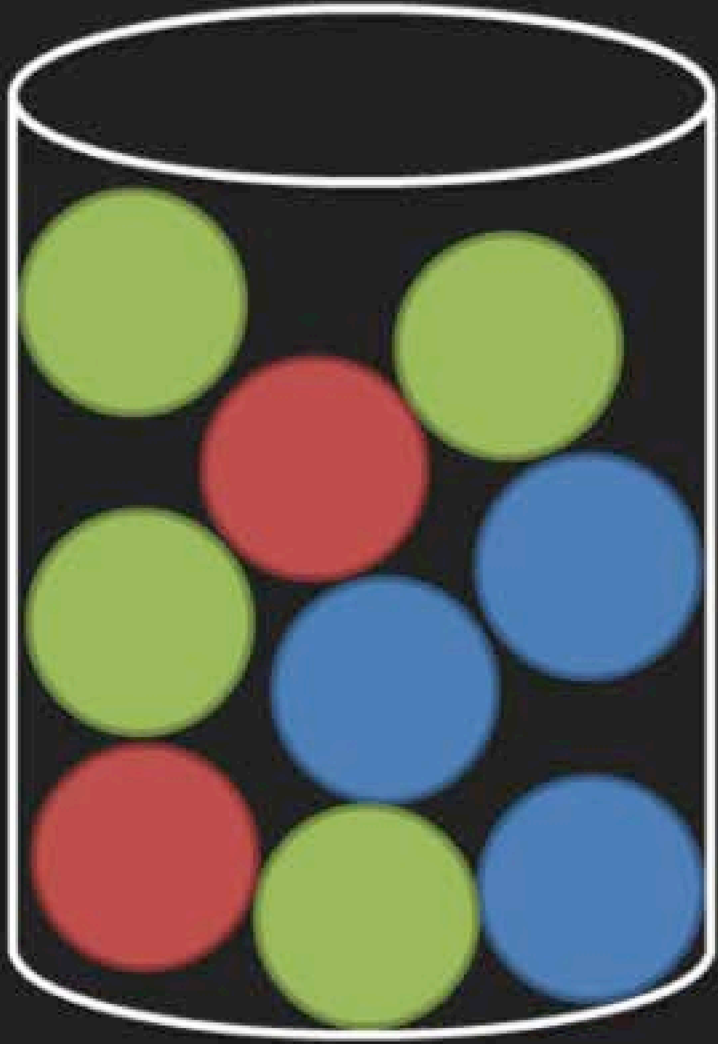
Count the Circles

Red	=	2
Green	=	4
Blue	=	3



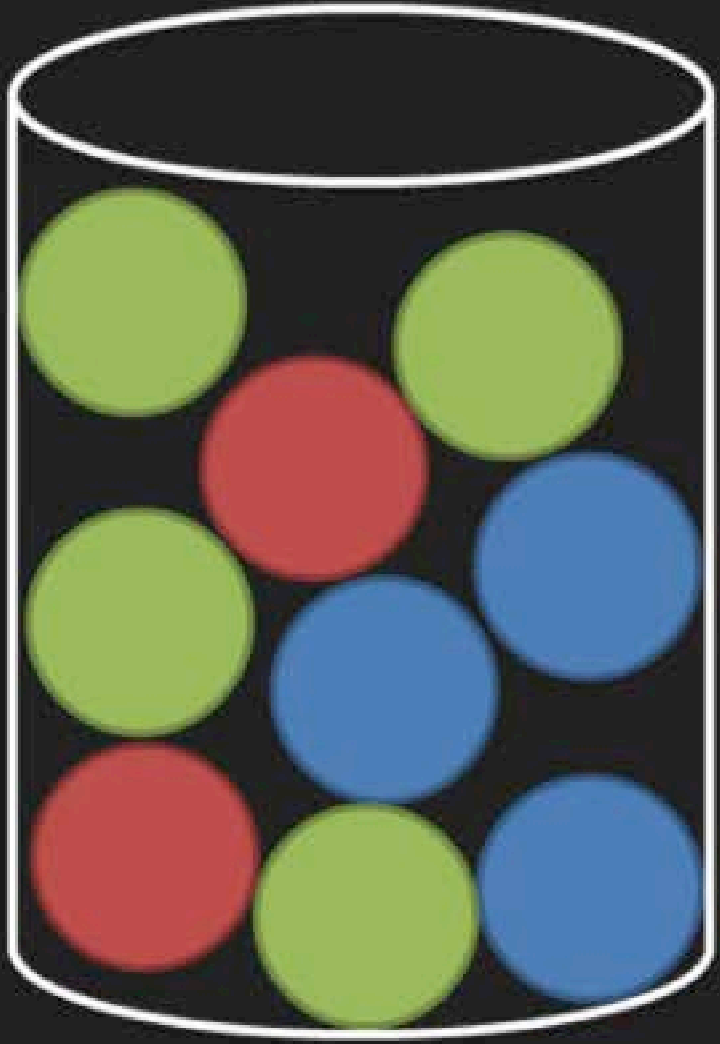
Keyword Lists

[2,4,3]



Keyword Lists

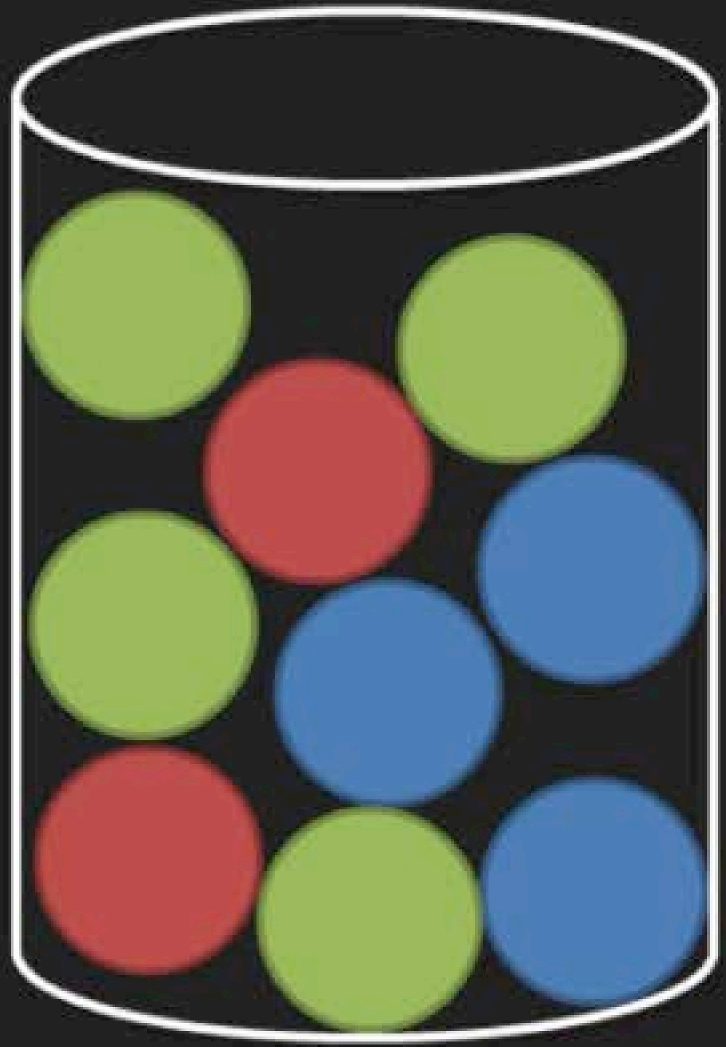
```
[{:red,2},{:green,4},{:blue,3}]
```



Keyword Lists

```
[{:red,2},{:green,4},{:blue,3}]
```

A list of tuples with 2 elements, the first being an atom



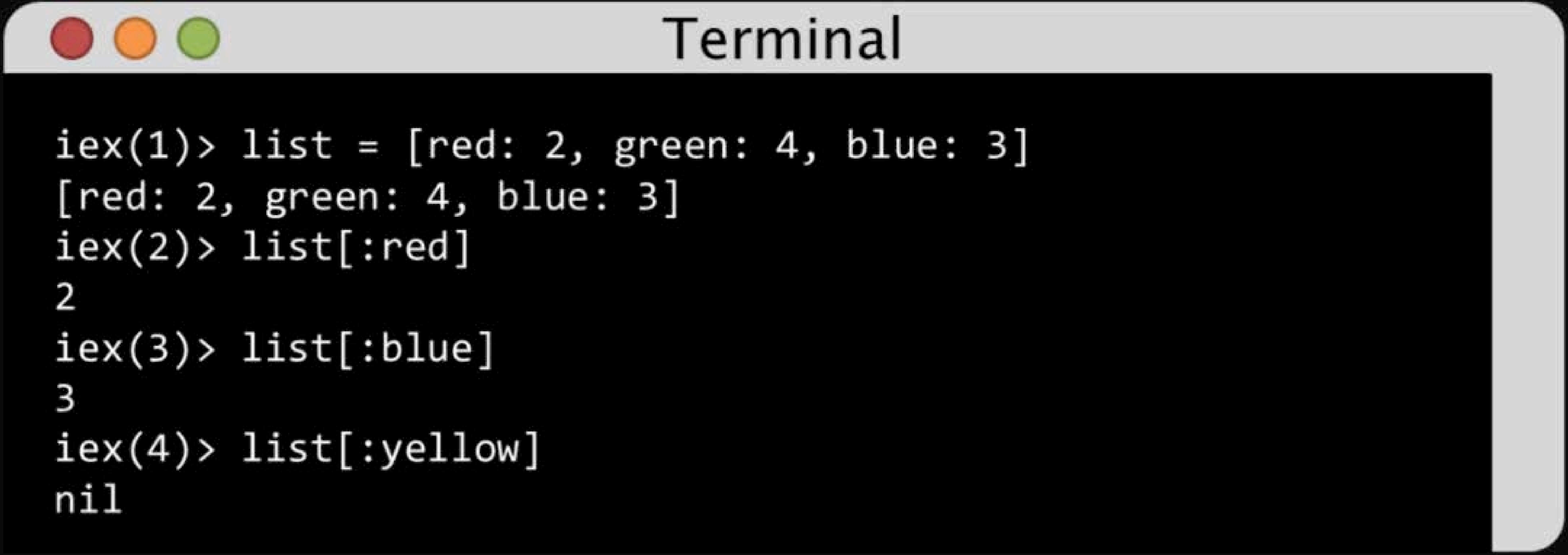
Keyword Lists

```
[{:red,2}, {:green,4}, {:blue,3}]
```

=

```
[red: 2, green: 4, blue: 3]
```

Keyword Lists - Indexing



```
iex(1)> list = [red: 2, green: 4, blue: 3]
[red: 2, green: 4, blue: 3]
iex(2)> list[:red]
2
iex(3)> list[:blue]
3
iex(4)> list[:yellow]
nil
```

Keyword Lists

- Still lists...
  - Indexing is slow
  - Ordered

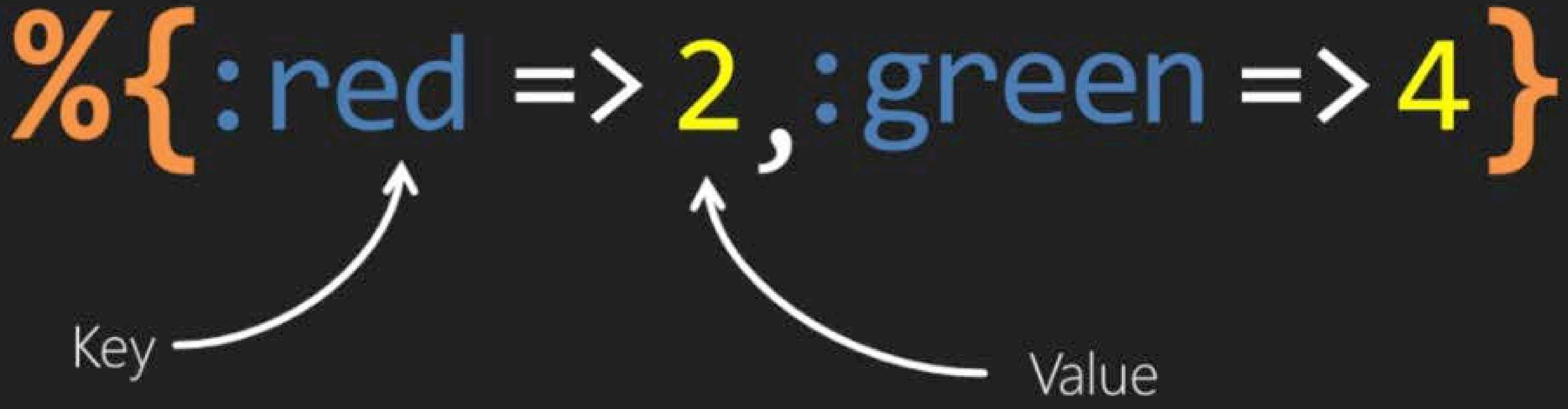
Maps

- A unordered collection of values indexed by keys

Maps

```
%{:red => 2,:green => 4}
```

Maps



```
%{:red => 2, :green => 4}
```

Maps

```
%{:red => 2, :green => 4}
=
%{red: 2, green: 4}
```

If the keys are atoms

Maps - Indexing

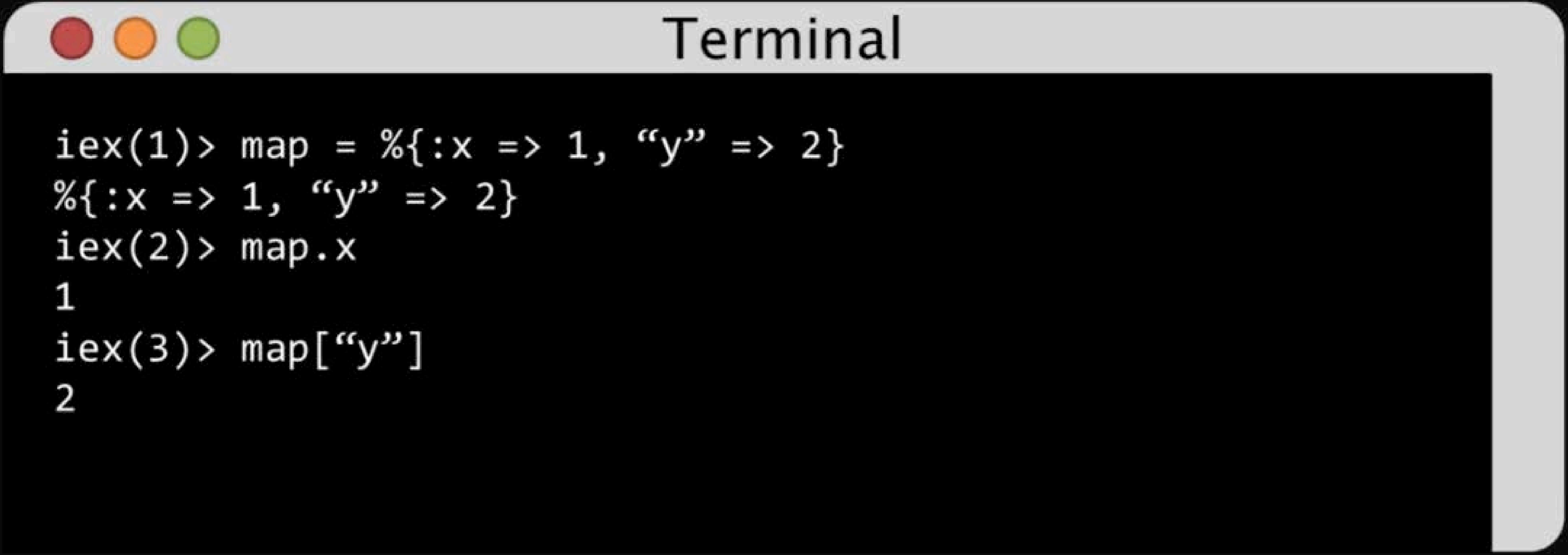
`map[key]`

Works with any type of key

`map.key`

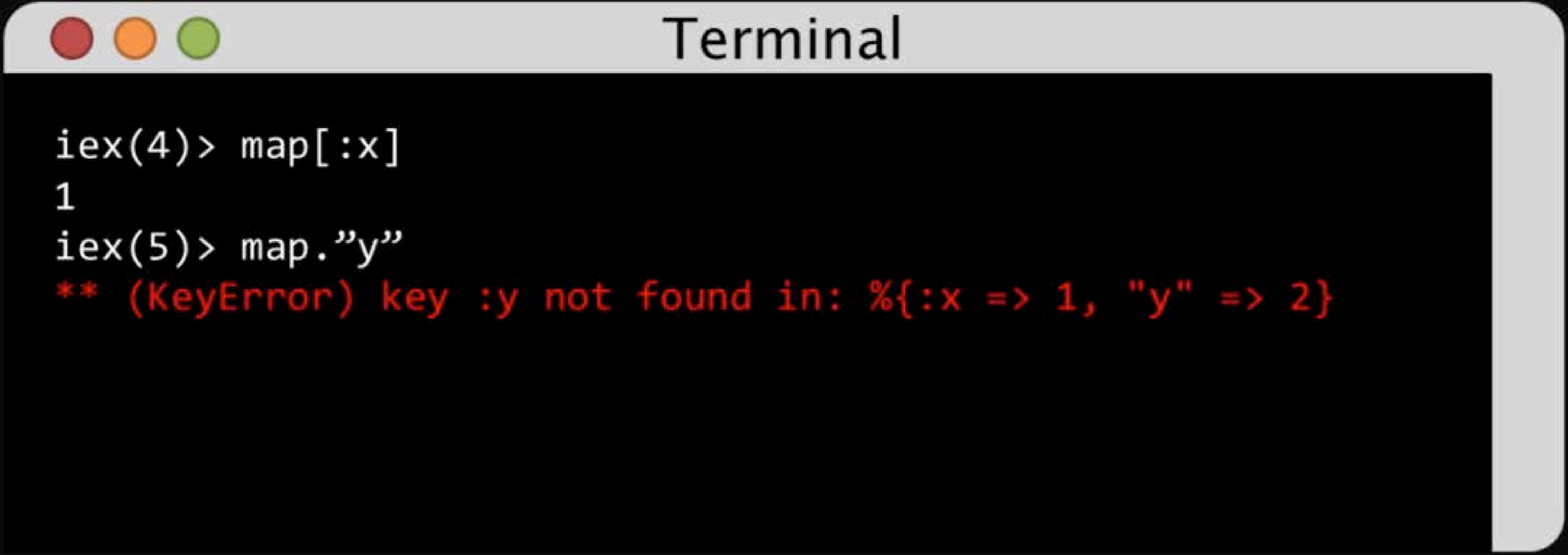
Works only on keys that are atoms

Maps - Indexing



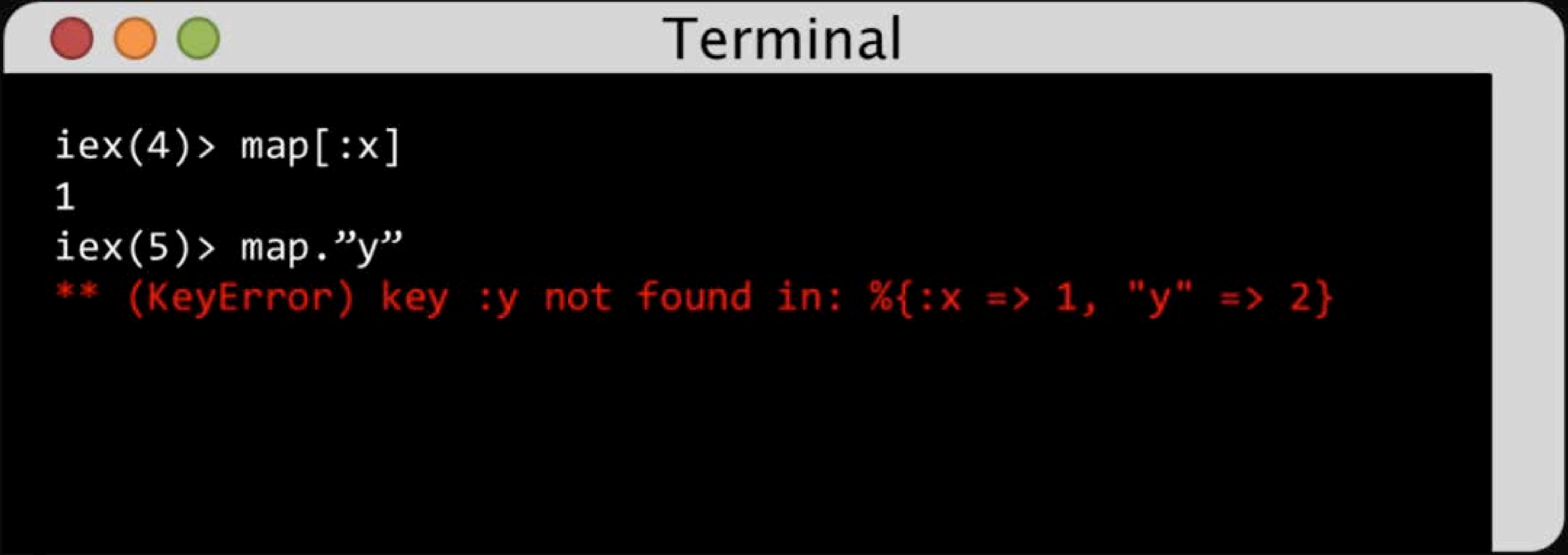
```
iex(1)> map = %{:x => 1, “y” => 2}
%{:x => 1, “y” => 2}
iex(2)> map.x
1
iex(3)> map[“y”]
2
```

Maps - Indexing



```
iex(4)> map[:x]
1
iex(5)> map.”y”
** (KeyError) key :y not found in: %{:x => 1, "y" => 2}
```

Maps - Indexing



```
iex(4)> map[:x]
1
iex(5)> map.”y”
** (KeyError) key :y not found in: %{:x => 1, "y" => 2}
```

Maps - Indexing

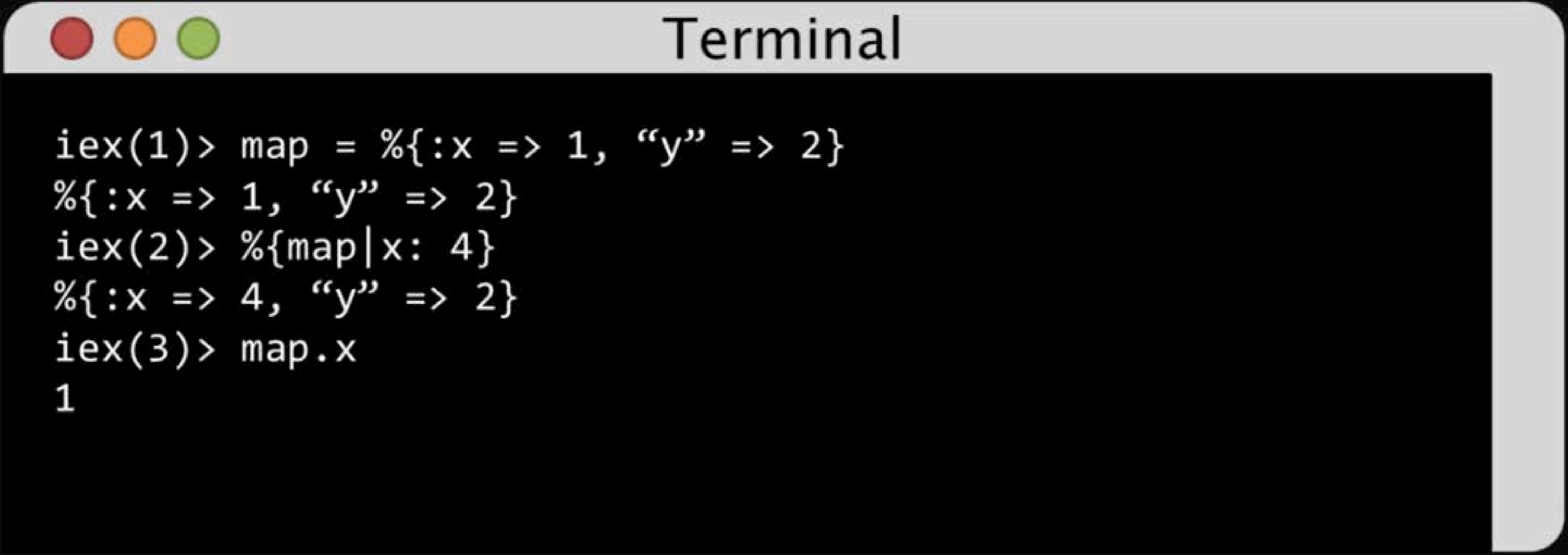
```
%{map|key=>value}
```

Works with any type of key

```
%{map|key: value}
```

Works only on keys that are atoms

Maps - Updating



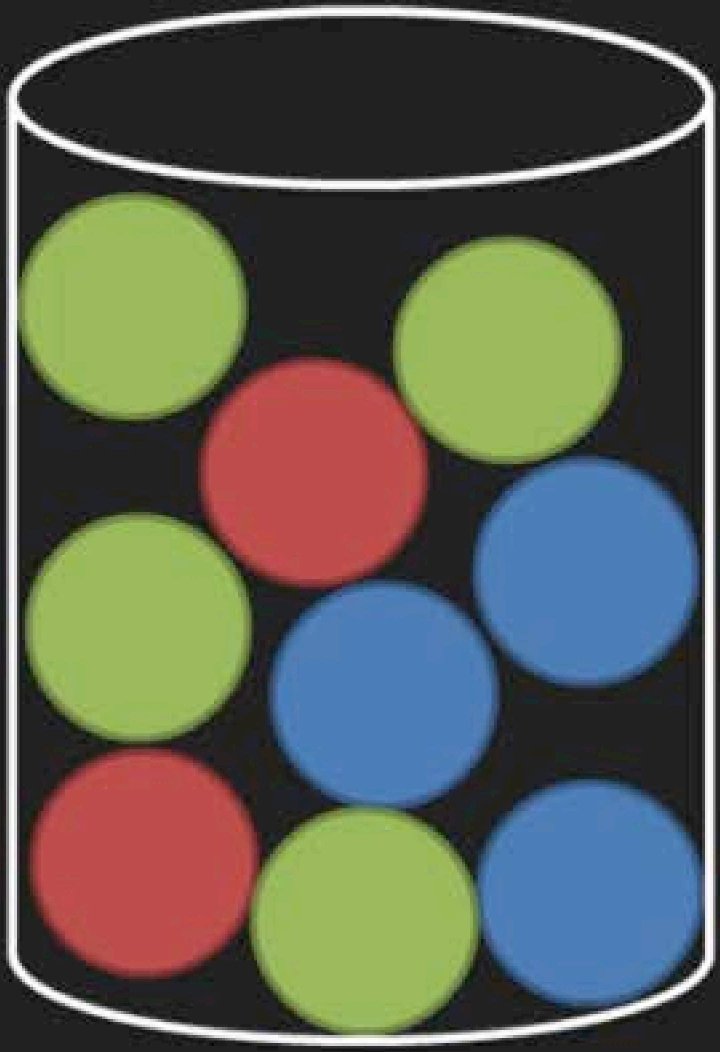
Terminal

```
iex(1)> map = %{:x => 1, “y” => 2}
%{:x => 1, “y” => 2}
iex(2)> %{map|x: 4}
%{:x => 4, “y” => 2}
iex(3)> map.x
1
```

Imutability

- Collections are Imutable
    - Any modification on a collection returns a new collection

Composition



Composition

- Counting exercise given to different people
  - John
  - Mary
  - Jeff
  - Paul
- Any person can do more than one counting exercise

```
%{red: 2, green: 4
```

Composition

```
%{
  “John” => [
    %{red: 2, green: 4}
  ],
  “Mary” => [
    %{red: 2, green: 4}, %{yellow: 5}, %{red: 7, blue: 2}
  ],
  “Jeff” => [
    %{violet: 40, blue: 2}
  ],
  “Paul” => [
    %{red: 4, blue: 3, yellow: 7}, %{blue: 5, cyan: 3}
  ]
}
```

Summary

- Literal types in Elixir
  - Numbers
  - Strings
  - Atoms
- Collection types in Elixir
  - Lists
  - Tuples
  - Maps

- Functions to manipulate these types
- Immutability of collections
- How to compose types to make more complex ones