João Gonçalves

Literals and Operators

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

- Observing the built-in literal types
- Exploring the basic operators

Before We Start

Terminal

```
>iex
Erlang/OTP 19 [erts-8.2]

Interactive Elixir (1.3.4) - press Ctrl+C to exit (type
h() ENTER for help)
iex(1)>
```

Numbers and Arithmetic Operators

- Integer
  - `1`
  - `-30`
- Floating point
  - `1.54932`
  - `1.5e-15`

Numbers and Arithmetic Operators

- Integer Bases

Base	Example	Decimal
Hexadecimal (16)	`0xFE`	254
Octal (8)	`0o773`	507
Binary (2)	`0b1110`	14

Numbers and Arithmetic Operators

- Integer – Separator
  - `43312209`
  - `43_312_209`

Numbers and Arithmetic Operators

- Precision

5543857348953478543953485734576423485423678452367423546723456237452367452367453267452376452376452376453267452367457623456312457623452 63

Numbers and Arithmetic Operators

- Operators

Operator	Example	Result
Addition (+)	`1.5 + 9`	`10.5`
Subtraction (-)	`1 - 2`	`-1`
Multiplication (*)	`9 * 8`	`72`
Division (/)	`10 / 4`	`2.5`

Numbers and Arithmetic Operators

- Integer Division Functions

Operator	Example	Result
Division (div)	`div(9, 7)`	1
Modulo/Remainder (rem)	`rem(9, 7)`	2

Booleans

Terminal

```
iex(1)> (1 + 2 + (3 * 12)) / div(6,3)
13.0
```

Booleans

- True
  - Everything is truthy
- False
  - nil is falsey

Booleans

and, or, not → strict

&&, ||, ! → non-strict

Booleans

Terminal

```
iex(1)> true and false
false
iex(2)> true && false
false
iex(3)> 3 and true
** (ArgumentError) argument error: 3
iex(4)> 3 && true
true
```

Booleans – Comparison Operators

>, <, >=, <=, !=, !==, ==, ===

true

false

Booleans – Comparison Operators

`>, <, >=, <=, !=, ==, !==, ===`

Non-Strict

Strict

Booleans – Comparison Operators

Terminal

```
iex(1)> 15.0 == 15
true
iex(2)> 15.0 === 15
false
```

Booleans – Comparison Operators

- Not just for numbers
  - `number < atom < reference < function < port < pid < tuple < map < list < bitstring`
- Multiple types can be compared with one another by this order

Strings

"Hello, World!"

"γειά σου κόσμ"

Strings

"Hello, World!"

"γειά σου κόσμ"

UTF-8

Strings

```
"Hello,\nWorld!"  ──────►  Hello,
                           World!
```

Strings

```
"I am #{2 * 15} years old."
```

```
I am 30 years old.
```

Strings – Some fun(ctions)

Terminal

```
iex(1)> String.reverse("Hello")
"olleH"
iex(2)> String.replace("Hello, World!", "World", "You")
"Hello, You!"
```

Atoms

- A constant whose name is its value
  - `:hello`
  - `:Hey_you`
  - `:">_<"`

Atoms

`:4_seasons` → `:"4_seasons"`

Is invalid!

Valid atom

João Gonçalves

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 [ ]
- An empty list [ ]

Lists

```
[:hello,“hey”,4]
```

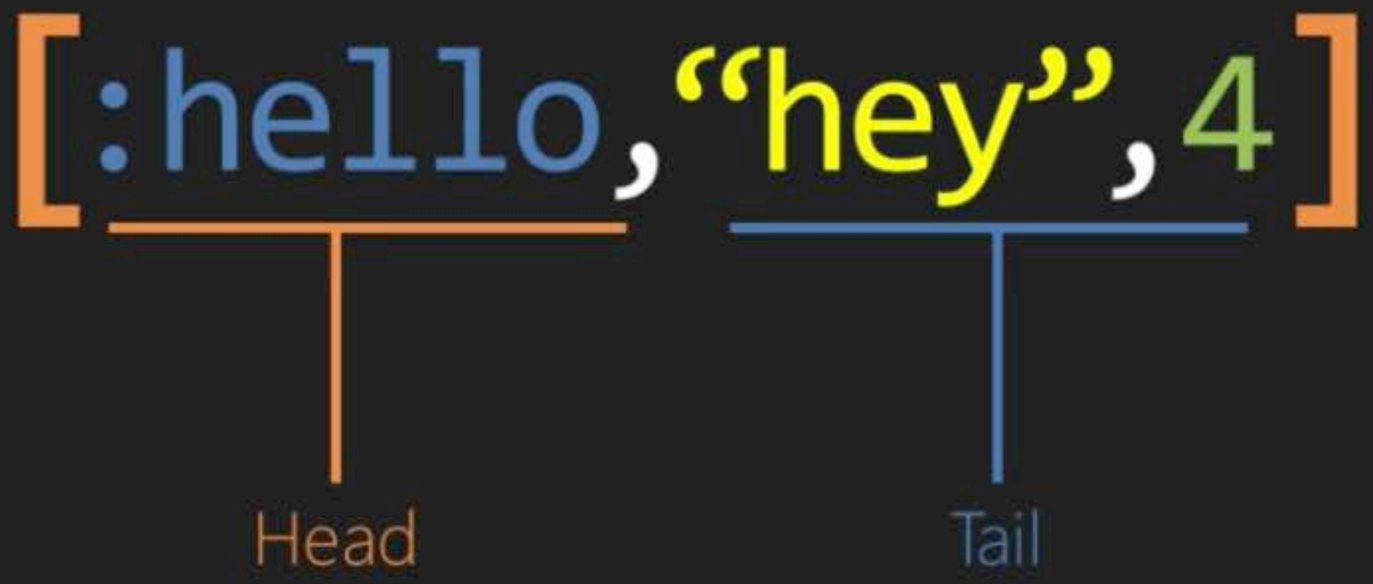
Lists

[:hello,“hey”,4]

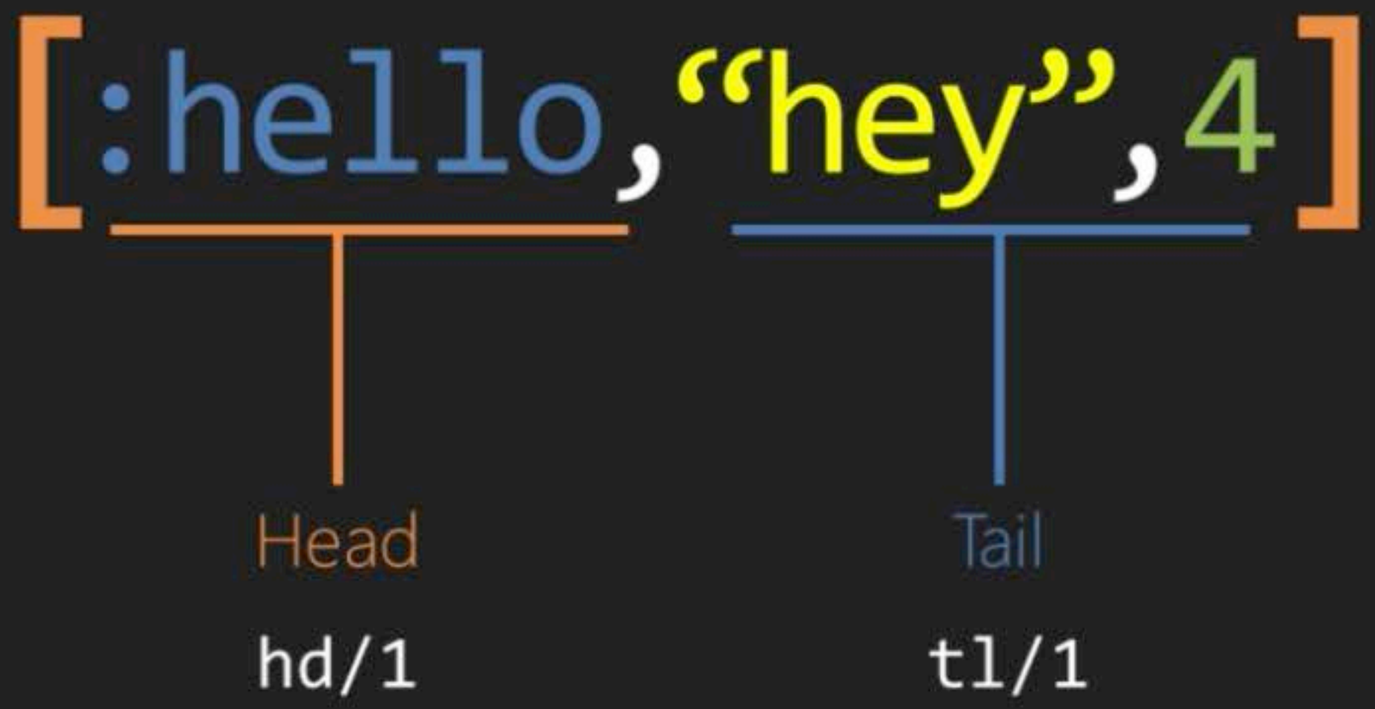
Any type

Separated by commas

Lists



Lists



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 – The Cons Cell

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

Two elements separated by a "Pipe"

"Cons" Cell

Lists – The Cons Cell

Lists – The Cons Cell

Lists – The Cons Cell


```
[:hello]
```

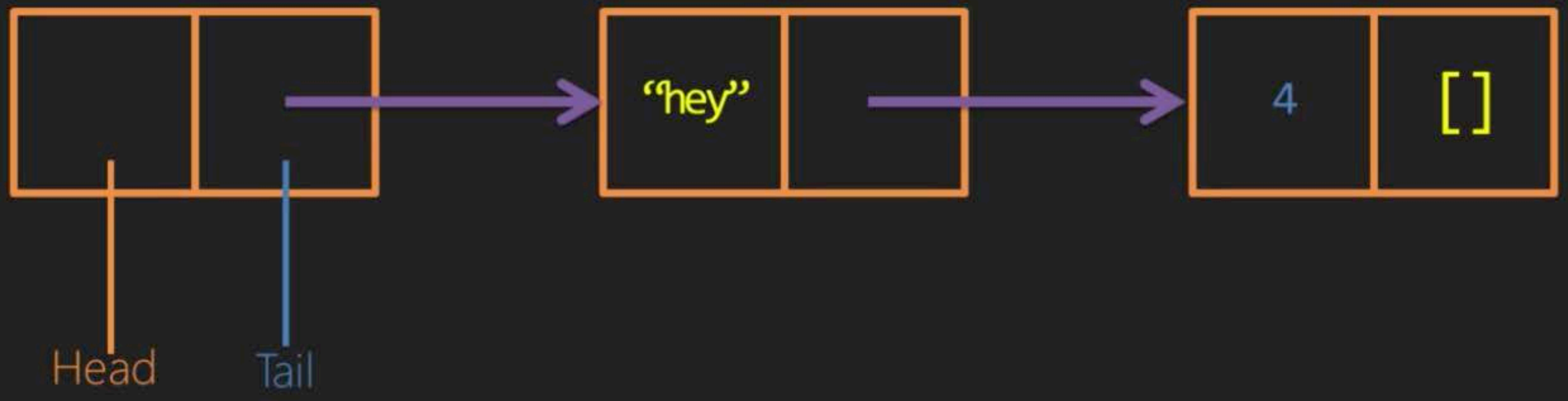
Lists – The Cons Cell

[:hello|["hey"|[4|[]]]]

[:hello,"hey",4]

Lists – The Cons Cell

- Linked List



Lists – Operations

- Concatenation
  - ++
- Subtraction
  - --

Lists – Operations

Terminal

```
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]
```

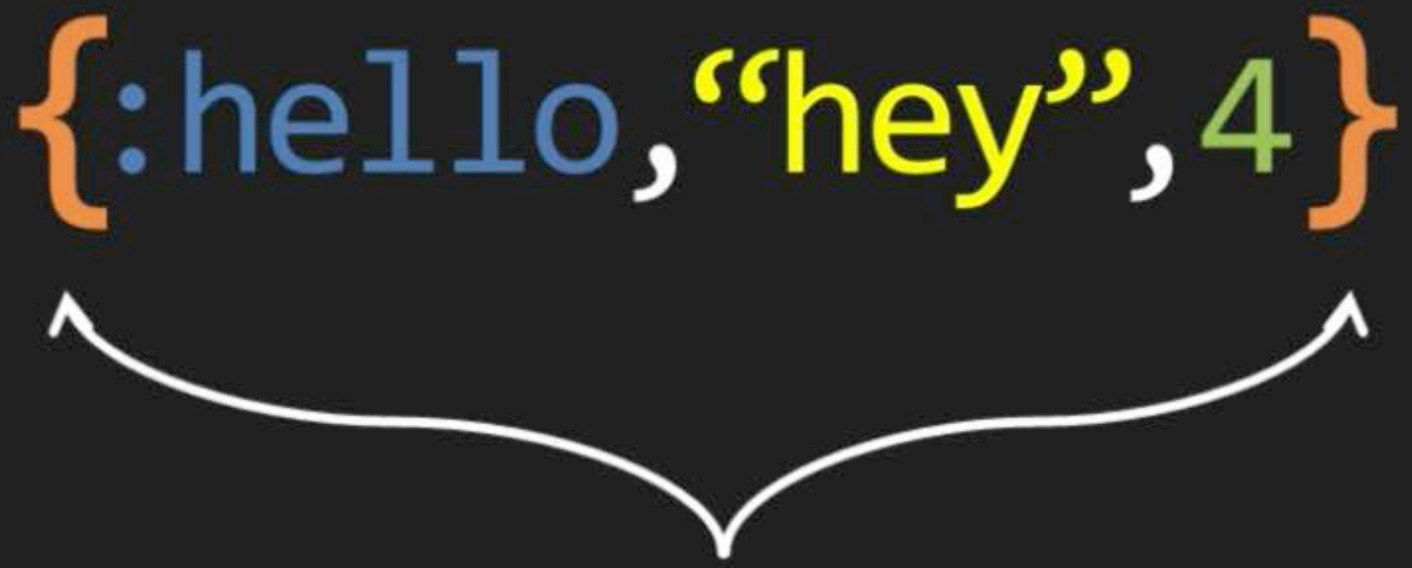
Tuple

- An ordered collection of items

Tuple

```
{:hello,“hey”,4}
```

Tuple



Tuples - Functions

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

Tuples - Functions

Terminal

```
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

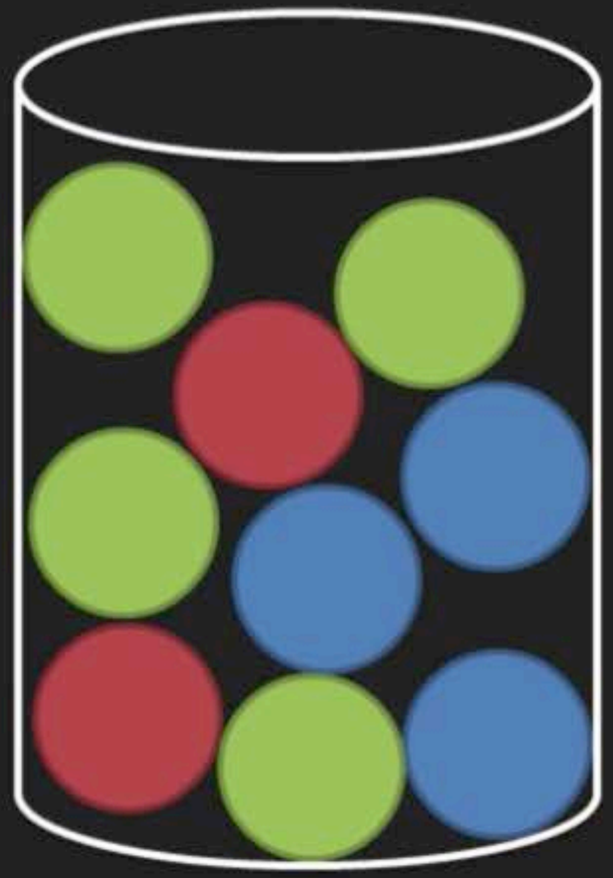
List

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

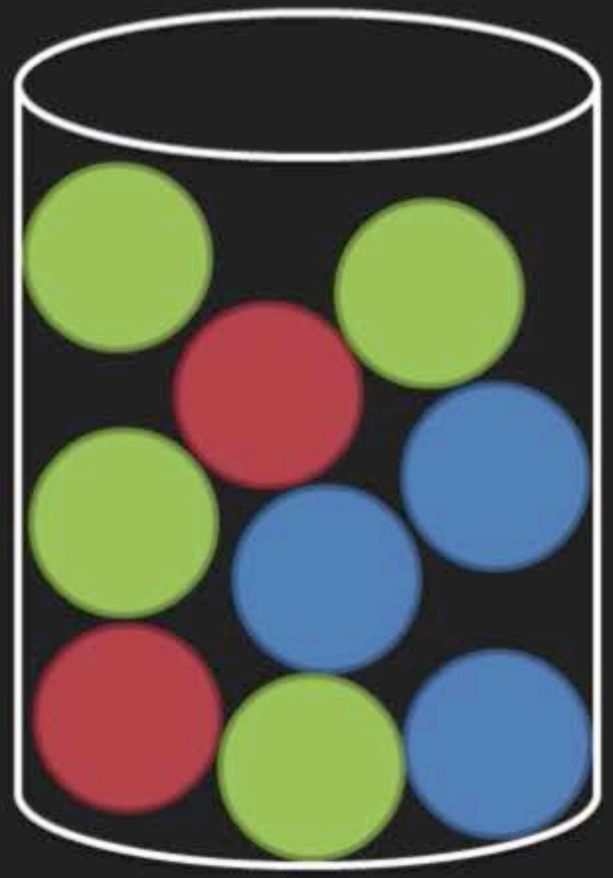
Keyword Lists



Keyword Lists

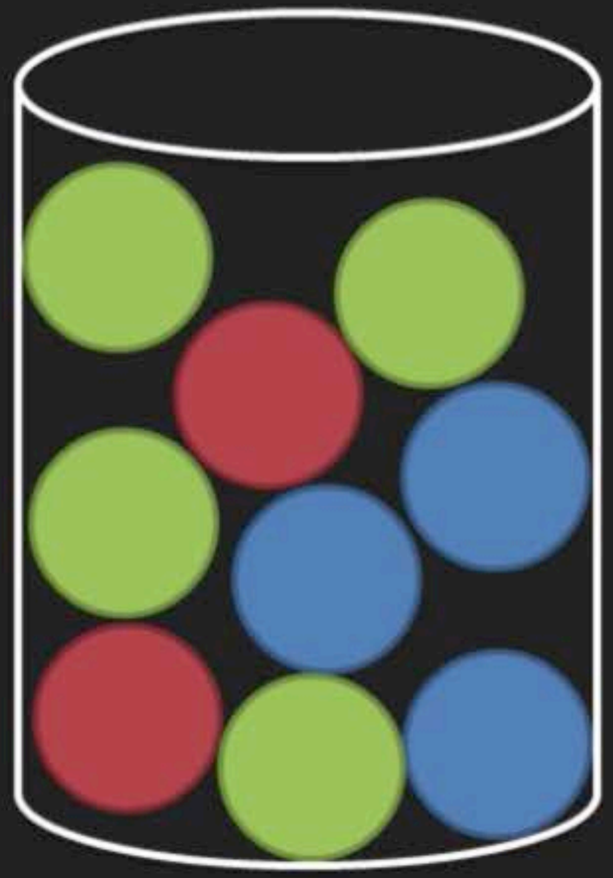
Count the Circles

```
Red     = 2
Green   = 4
Blue    = 3
```

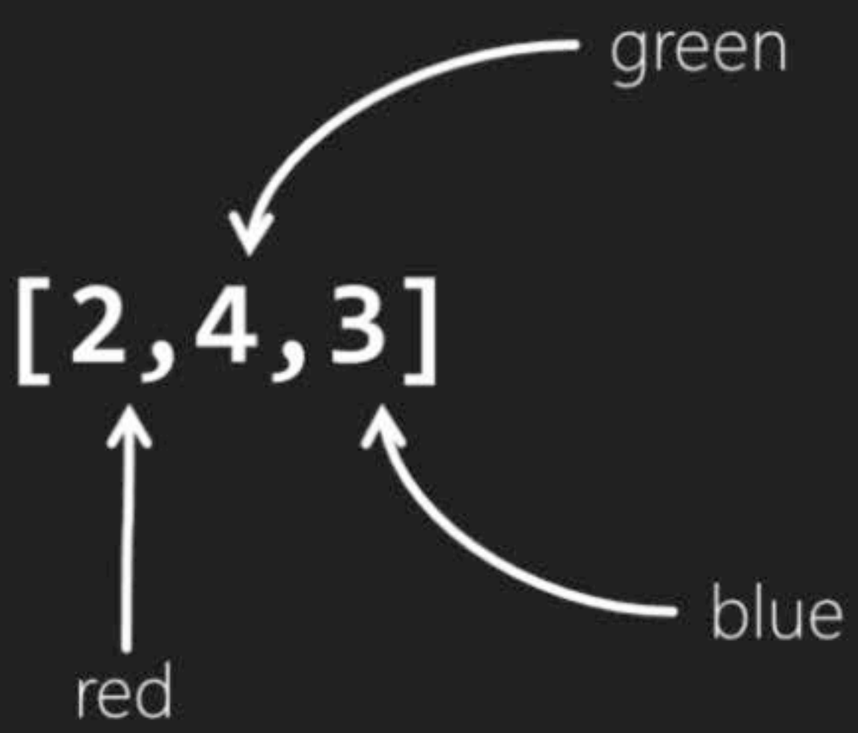


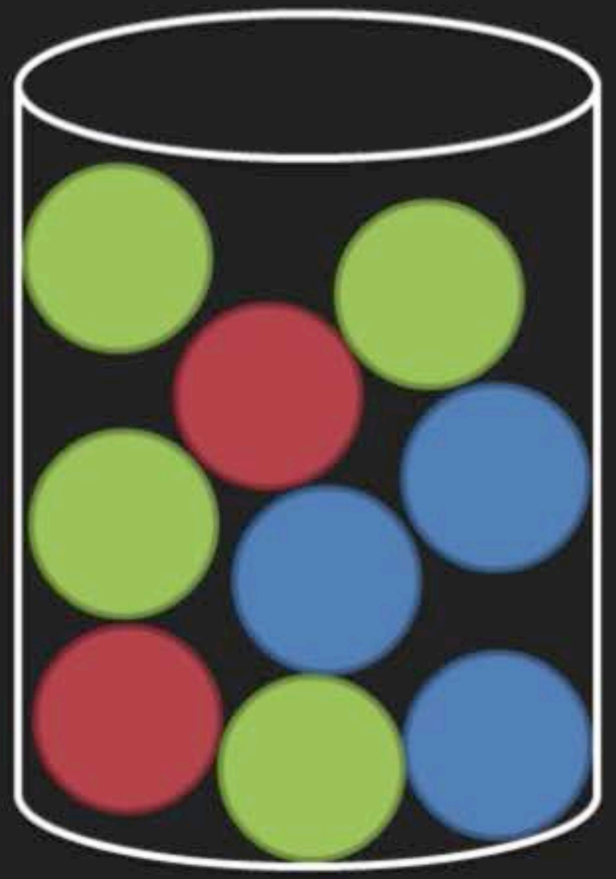
Keyword Lists

[2,4,3]



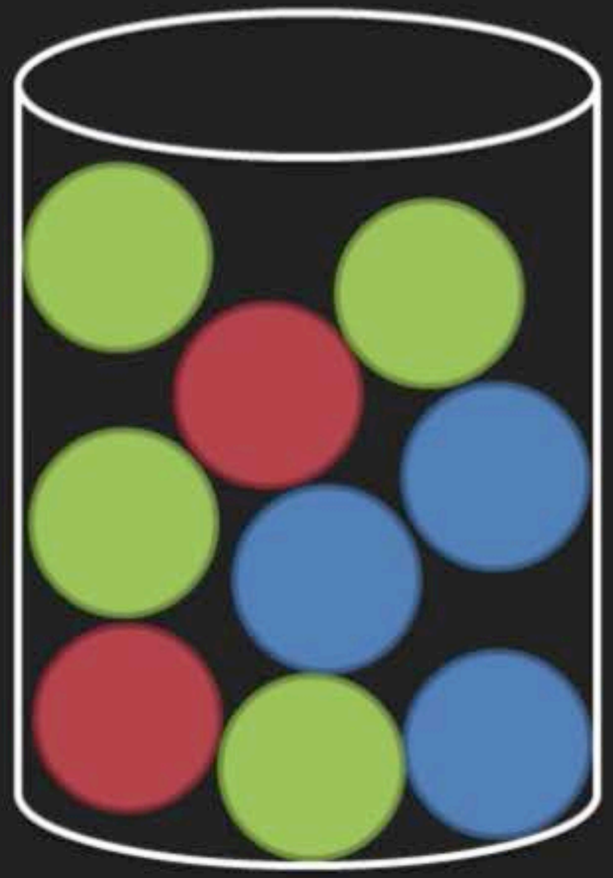
Keyword Lists





Keyword Lists

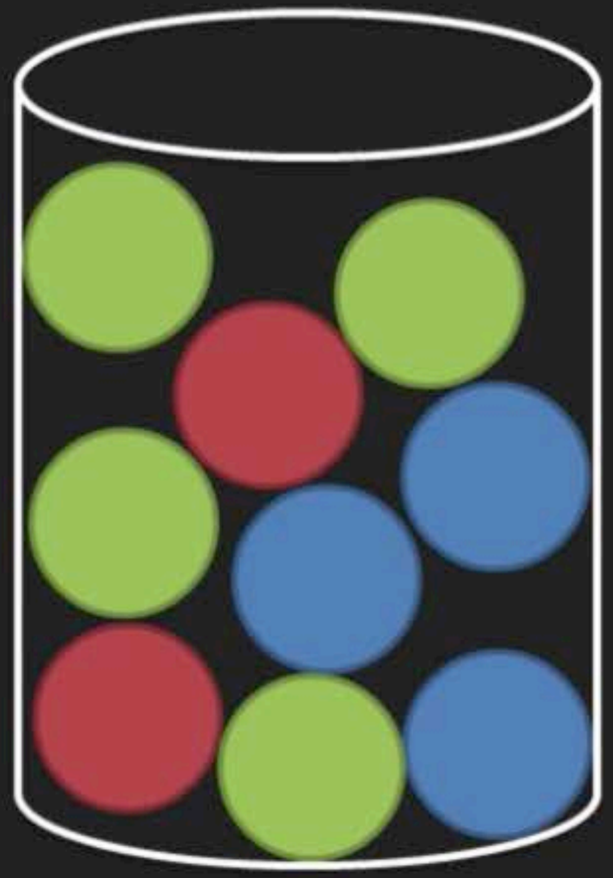
```
[{:red,2},{:green,4},{:blue,3}]
```



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

Terminal

```
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}
```

Key

Value

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

Terminal

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

Maps - Indexing

Terminal

```
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

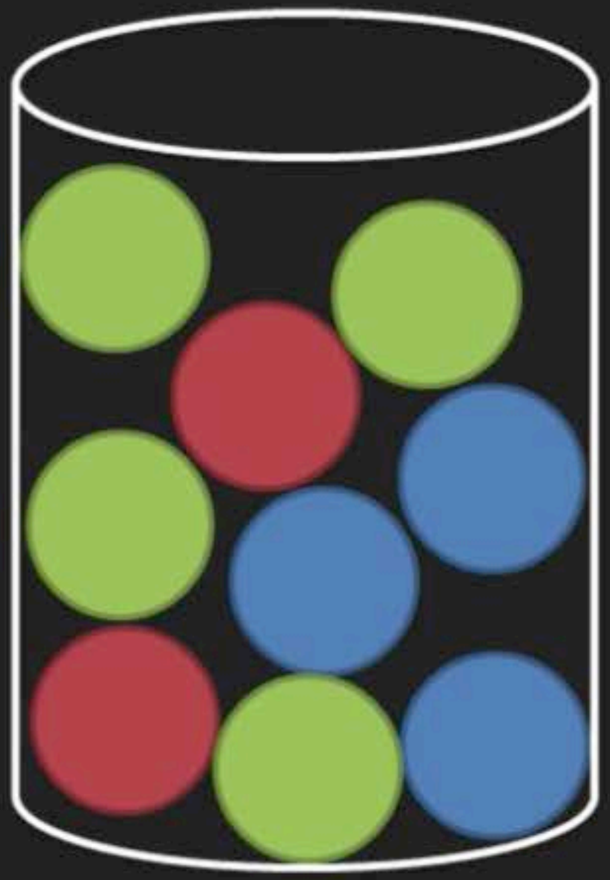
Terminal

```
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" =>

  "Mary" =>

  "Jeff" =>

  "Paul" =>
}
```

Composition

```
%{
  "John" => [

  ],
  "Mary" => [

  ],
  "Jeff" => [

  ],
  "Paul" => [

  ]
}
```

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}
  ]
}
```

Quiz Time!

Quiz 2 | 2 Questions

Start Quiz

Skip Quiz >

Question 1:

Identify the integer base from the following options:

- Binary
- Multiplication
- Modulo
- Remainder

Question 2:

Which of the following options are ordered sequential collection types?

- Lists
- Tuples
- Indexes
- Maps