

**EVERYTHING IS  
reduce**

# THIS PRESENTATION IS ABOUT

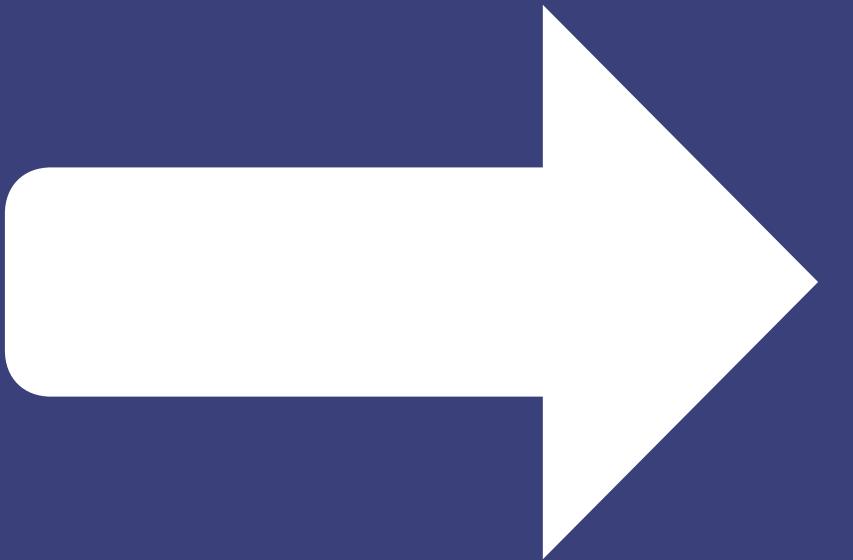
**THIS PRESENTATION IS ABOUT**

**reduce**

**TRANSFORM  
SOMETHING**

**TRANSFORM  
TO A DIFFERENT FORM**

**TRANSFORM  
TO A MORE BASIC FORM**



```
def pulp(orange) do
```



```
end
```

```
def pulp([,,]) do  
end
```



```
def reduce([,,]) do  
end
```



Final

# Lists

[ 1 , 2 , 3 ]

# Maps

```
%{apple: "🍏", banana: "🍌"}
```

So...

```
Enum.sum([1, 2, 3])  
#=>
```

```
Enum.sum([1, 2, 3])  
#=> 6
```

```
Enum.join(["Elixir", "is", "awesome!"], "")  
#=>
```

```
Enum.join(["Elixir", "is", "awesome!"], "")  
#=> "Elixir is awesome!"
```

[1, 2, 3]

# PERFORM OPERATION

# DERIVED INFORMATION

**CREATE  
NEW  
INFORMATION**



[1, 2, 3]

```
Enum.each([1, 2, 3], fn number ->  
  IO.puts("Current number is #{number}")  
end)  
#=>
```

```
Enum.each([1, 2, 3], fn number ->  
  IO.puts("Current number is #{number}")  
end)
```

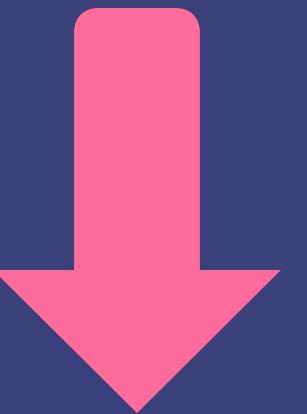
Current number is 1

Current number is 2

Current number is 3

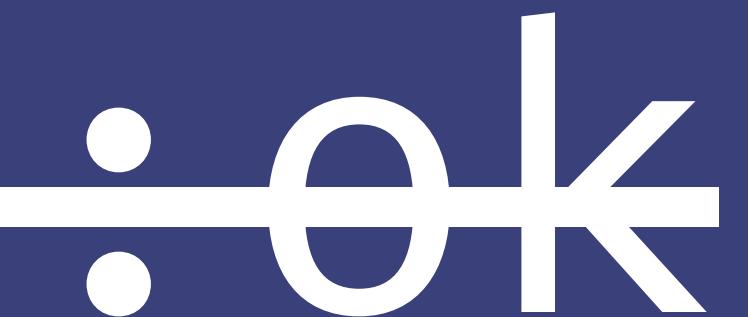
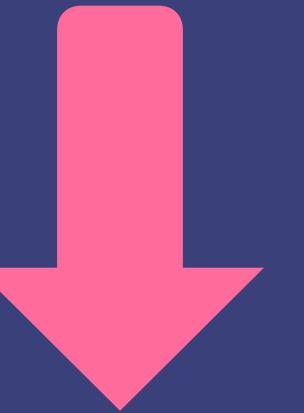
#=> :ok

[ 1 , 2 , 3 ]

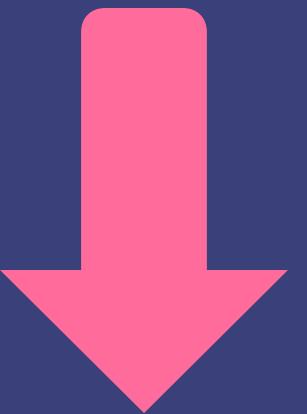


:ok

[ 1 , 2 , 3 ]



[1, 2, 3]



[2, 4, 6]

```
Enum.map([1, 2, 3], fn number ->  
  number * 2  
end)  
#=>
```

```
Enum.map([1, 2, 3], fn number ->  
  number * 2  
end)  
#=> [2, 4, 6]
```

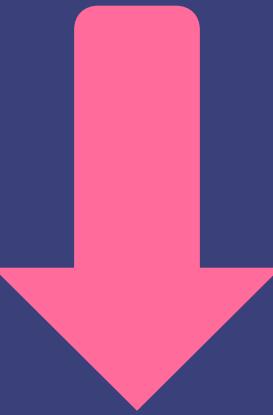


[1, 2, 3]

```
[ "a", "b", "c" ]
```

```
[ "a", "a", "a", "b", "c", "c" ]
```

```
[ "a" , "a" , "a" , "b" , "c" , "c" ]
```

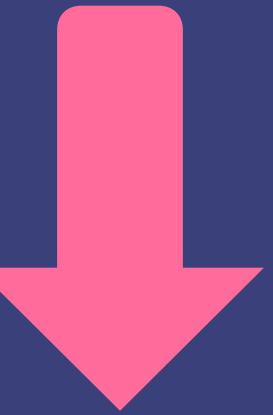


```
%{ "a" => 3 , "b" => 1 , "c" => 2 }
```

# Requirements

- Take in a list
- Return a map
- Loop over each element in list
- Add letter key to map if it doesn't exist
- If letter key exists, increment its value

```
[ "a" , "a" , "a" , "b" , "c" , "c" ]
```



```
%{ "a" => 3 , "b" => 1 , "c" => 2 }
```

reduce



Enum.reduce(enumerable, acc, fun)

```
Enum.reduce(["a", "a", "a", "b", "c", "c"], acc, fun)
```

```
list = ["a", "a", "a", "b", "c", "c"]  
Enum.reduce(list, acc, fun)
```

```
list = ["a", "a", "a", "b", "c", "c"]  
Enum.reduce(list, accumulator, fun)
```

```
list = ["a", "a", "a", "b", "c", "c"]  
Enum.reduce(list, acc, fun)
```

```
list = [ "a", "a", "a", "b", "c", "c"]
```

```
Enum.reduce(list, acc, fun)
```

```
#=> %{"a" => 3, "b" => 1, "c" => 2}
```

```
list = [ "a", "a", "a", "b", "c", "c"]
```

```
Enum.reduce(list, %{}, fun)
```

```
#=> %{"a" => 3, "b" => 1, "c" => 2}
```

```
list = [ "a", "a", "a", "b", "c", "c"]  
Enum.reduce(list, %{"a" => 10}, fun)  
#=> %{"a" => 3, "b" => 1, "c" => 2}
```

```
list = [ "a", "a", "a", "b", "c", "c"]
```

```
Enum.reduce(list, %{}, fun)
```

```
#=> %{"a" => 3, "b" => 1, "c" => 2}
```

```
list = [ "a", "a", "a", "b", "c", "c"]

Enum.reduce(list, %{}, fn char, map =>
  # ...
  # ...
  # ...
end)

#=> %{"a" => 3, "b" => 1, "c" => 2}
```

```
list = ["a", "a", "a", "b", "c", "c"]

Enum.reduce(list, %{}, fn char, map =>
  Map.update(map, char, 1, fn value =>
    value + 1
  end)
end)

#=> %{"a" => 3, "b" => 1, "c" => 2}
```

```
list = ["a", "a", "a", "b", "c", "c"]

Enum.reduce(list, %{}, fn char, map =>
  Map.update(map, char, 1, fn value =>
    value + 1
  end)
end)

#=> %{"a" => 3, "b" => 1, "c" => 2}
```

```
list = ["a", "a", "a", "b", "c", "c"]

Enum.reduce(list, %{}, fn char, map =>
  Map.update(map, char, 1, fn value =>
    value + 1
  end)
end)

#=> %{"a" => 3, "b" => 1, "c" => 2}
```

```
list = ["a", "a", "a", "b", "c", "c"]

Enum.reduce(list, %{}, fn char, map =>
  Map.update(map, char, 1, fn value =>
    value + 1
  end)
end)

#=> %{"a" => 3, "b" => 1, "c" => 2}
```

```
list = ["a", "a", "a", "b", "c", "c"]

Enum.reduce(list, %{}, fn char, map =>
  Map.update(map, char, 1, fn value =>
    value + 1
  end)
end)

#=> %{"a" => 3, "b" => 1, "c" => 2}
```

```
list = ["a", "a", "a", "b", "c", "c"]

Enum.reduce(list, %{}, fn char, map =>
  Map.update(map, char, 1, fn value =>
    value + 1
  end)
end)

#=> %{"a" => 3, "b" => 1, "c" => 2}
```

**EVERY**  
Enum **FUNCTION IS**  
**reduce**

```
Enum.map([1, 2, 3], fn number ->
  number * 2
end)
#=> [2, 4, 6]
```

```
Enum.reduce([1, 2, 3], [], fn number, list ->
  [number * 2 | list]
end)
#=>
```

```
Enum.reduce([1, 2, 3], [], fn number, list ->
  [number * 2 | list]
end)
#=> [6, 4, 2]
```

```
Enum.reduce([1, 2, 3], [], fn number, list ->
  [number * 2 | list]
end)
|> Enum.reverse()
#=> [2, 4, 6]
```

```
Enum.map([1, 2, 3], fn number ->  
  number * 2  
end)  
#=> [2, 4, 6]
```

```
Enum.reduce([1, 2, 3], [], fn number, list ->  
  [number * 2 | list]  
end)  
|> Enum.reverse()  
#=> [2, 4, 6]
```

**SAYAMAP**

```
Enum.each([1, 2, 3], fn number ->  
  IO.puts("Current number is #{number}")  
end)
```

Current number is 1

Current number is 2

Current number is 3

#=> :ok

```
Enum.reduce([1, 2, 3], nil, fn number, _acc  ->
  IO.puts("Current number is #{number}")
end)
```

Current number is 1

Current number is 2

Current number is 3

#=> :ok

```
Enum.reduce([1, 2, 3], nil, fn number, _acc ->
  IO.puts("Current number is #{number}")
end)
```

```
Current number is 1
```

```
Current number is 2
```

```
Current number is 3
```

```
#=> :ok
```

```
Enum.reduce([1, 2, 3], nil, fn number, _acc  ->
  IO.puts("Current number is #{number}")
end)
```

```
Current number is 1
```

```
Current number is 2
```

```
Current number is 3
```

```
#=> :ok
```

```
Enum.reduce([1, 2, 3], nil, fn number, _acc  ->
  IO.puts("Current number is #{number}")
end)
```

Current number is 1

Current number is 2

Current number is 3

#=> :ok

```
Enum.each([1, 2, 3], fn number ->
  IO.puts("Current number is #{number}")
end)
Current number is 1
Current number is 2
Current number is 3
#=> :ok
```

```
Enum.reduce([1, 2, 3], nil, fn number, _acc ->
  IO.puts("Current number is #{number}")
end)
Current number is 1
Current number is 2
Current number is 3
#=> :ok
```

**SAYAMAP**

ALL ARE FOR



reduce

< >  elixir/enum.ex at v1.8.2 · elixir-lang/elixir · GitHub

```
773     def each(enumerable, fun) do
774       reduce(enumerable, nil, fn entry, _ ->
775         fun.(entry)
776         nil
777       end)
778
779       :ok
780     end
```

< >  elixir/enum.ex at v1.8.2 · elixir-lang/elixir · GitHub

```
1330   def map(enumerable, fun) do
1331     reduce(enumerable, [], R.map(fun)) |> :lists.reverse()
1332   end
1333
```





# Thanks!

@paulfioravanti

