

Another way to use a map at the School of Geography

Nicolas Payette



SCHOOL OF GEOGRAPHY
AND THE ENVIRONMENT



January 24th 2019

Overview

A word about functions

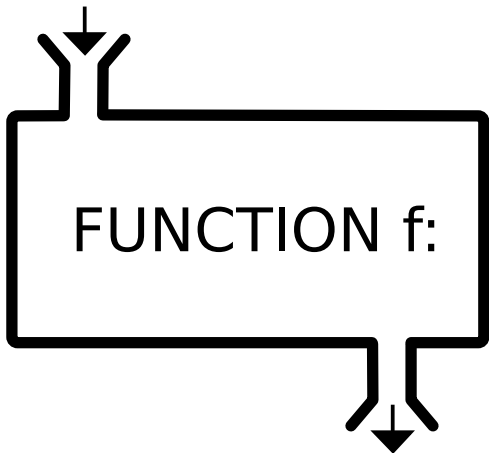
Mapping over a collection

A tiny practical example

Mapping over two collections

A word about functions

INPUT x



OUTPUT $f(x)$

A **higher-order function**:

- ▶ takes one or more functions as inputs
- ▶ and/or outputs a function.

A **higher-order function**:

- ▶ takes one or more functions as inputs
- ▶ and/or outputs a function.

map is a higher-order function that:

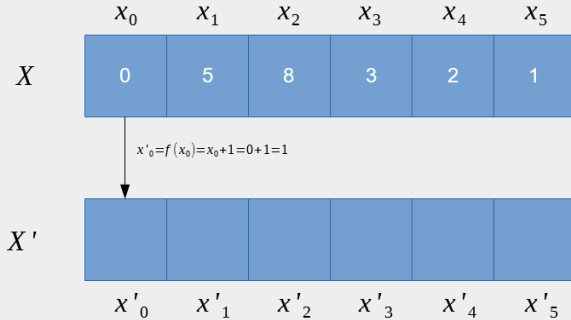
- ▶ takes a function and a collection of things as inputs
- ▶ and outputs another collection of things.

Mapping over a collection

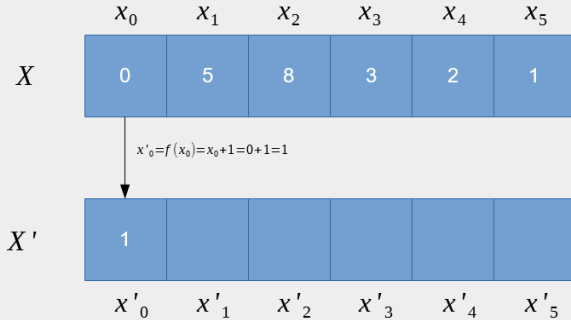
$$X' = \text{map}(X, f) \quad f(x) = x + 1$$

	x_0	x_1	x_2	x_3	x_4	x_5
X	0	5	8	3	2	1
X'						
	x'_0	x'_1	x'_2	x'_3	x'_4	x'_5

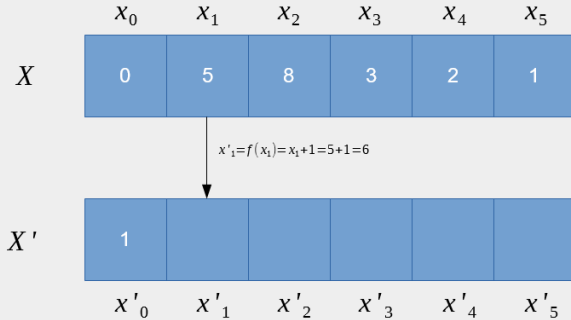
$$X' = \text{map}(X, f) \quad f(x) = x + 1$$



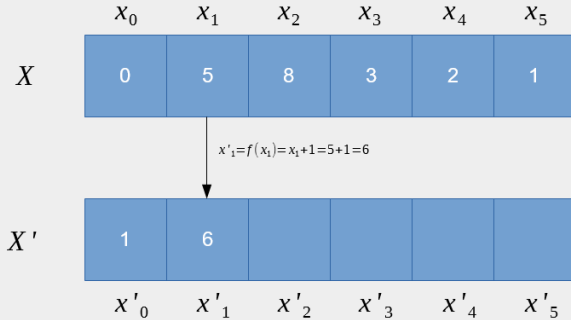
$$X' = \text{map}(X, f) \quad f(x) = x + 1$$



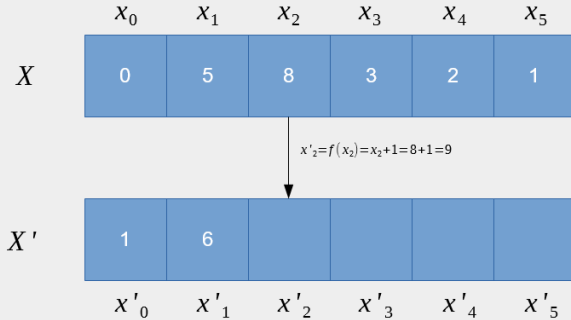
$$X' = \text{map}(X, f) \quad f(x) = x + 1$$



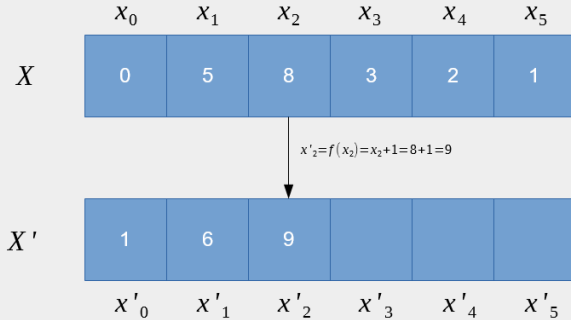
$$X' = \text{map}(X, f) \quad f(x) = x + 1$$



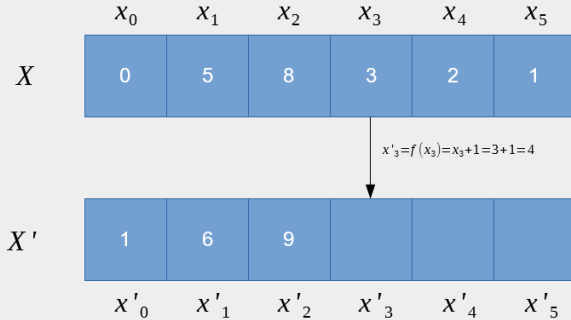
$$X' = \text{map}(X, f) \quad f(x) = x + 1$$



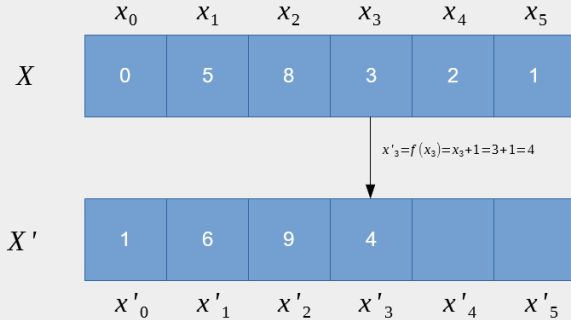
$$X' = \text{map}(X, f) \quad f(x) = x + 1$$



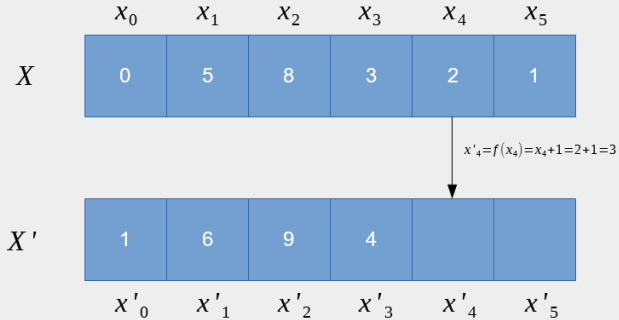
$$X' = \text{map}(X, f) \quad f(x) = x + 1$$



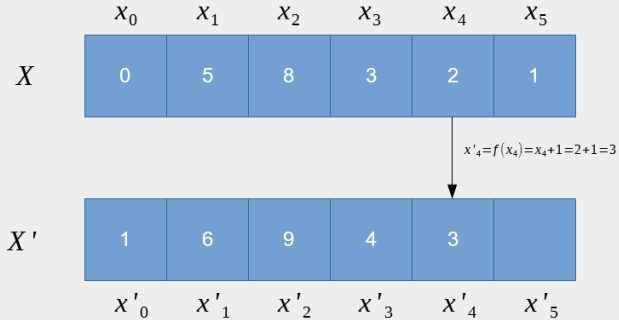
$$X' = \text{map}(X, f) \quad f(x) = x + 1$$



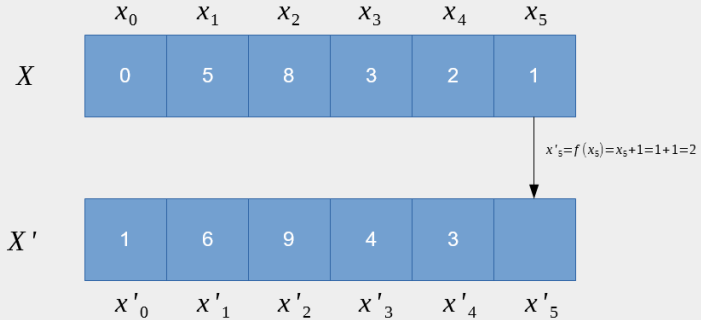
$$X' = \text{map}(X, f) \quad f(x) = x + 1$$



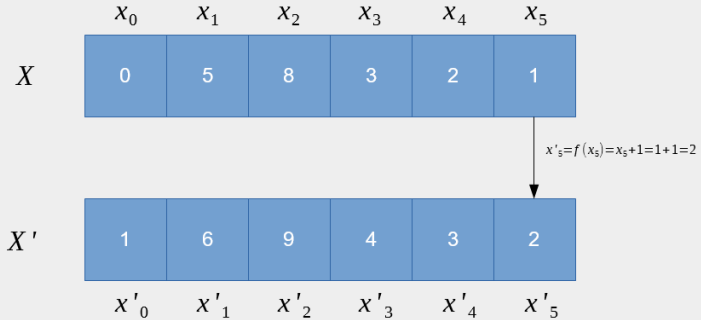
$$X' = \text{map}(X, f) \quad f(x) = x + 1$$



$$X' = \text{map}(X, f) \quad f(x) = x + 1$$



$$X' = \text{map}(X, f) \quad f(x) = x + 1$$



$$X' = \text{map}(X, f) \quad f(x) = x + 1$$

	x_0	x_1	x_2	x_3	x_4	x_5
X	0	5	8	3	2	1

	x'_0	x'_1	x'_2	x'_3	x'_4	x'_5
X'	1	6	9	4	3	2

[https://en.wikipedia.org/wiki/Map_\(higher-order_function\)](https://en.wikipedia.org/wiki/Map_(higher-order_function))

Python

```
def f(x):  
    return x + 1  
xs = [0, 5, 8, 3, 2, 1]  
  
# For loop version  
result = []  
for x in xs:  
    result.append(f(x))  
print(result)  
  
# List comprehension version  
print([f(x) for x in xs])  
  
# Map version  
print(list(map(f, xs)))
```

Julia

```
f(x) = x + 1  
xs = [0, 5, 8, 3, 2, 1]
```

```
# For loop version:  
result = []  
for x = xs  
    push!(result, f(x))  
end  
println(result)
```

```
# List comprehension version:  
[f(x) for x = xs] |> println
```

```
# Map version:  
map(f, xs) |> println
```

```
# Broadcast version:  
f.(xs) |> println
```

R

```
f <- function(x) { x + 1 }  
xs <- c(0, 5, 8, 3, 2, 1)
```

```
# For loop version  
result <- c()  
for (x in xs) {  
  result <- c(result, f(x))  
}  
print(result)
```

```
# Base version, using sapply  
print(sapply(xs, f))
```

```
# Using purrr::map  
library(purrr)  
xs %>% map_dbl(f) %>% print
```


NetLogo

```
to-report f [ x ]  
  report x + 1  
end
```

```
to demo-map
```

```
  let xs [0 5 8 3 2 1]
```

```
  ; for loop version:
```

```
  let result []  
  foreach xs [ x -> set result lput f x result ]  
  print result
```

```
  ; map version:
```

```
  print map f xs
```

```
  ; bonus `of` version
```

```
  create-turtles 10  
  print [ f who ] of turtles
```

```
end
```

Agentsets

Lists

of



map

with



filter

ask



foreach

Java

```
import java.util.Arrays;
import java.util.ArrayList;
import java.util.List;
import java.util.function.Function;
import java.util.stream.Collectors;

class DemoMap {
    public static void main(String[] args) {

        List<Integer> xs = Arrays.asList(0, 5, 8, 3, 2, 1);
        Function<Integer, Integer> f = x -> x + 1;

        // Using a for loop:
        List<Integer> result = new ArrayList<>();
        for (Integer x: xs) { result.add(f.apply(x)); }
        System.out.println(result);

        // Using map:
        System.out.println(
            xs.stream().map(f).collect(Collectors.toList()));
    }
}
```

Scala

```
import collection.mutable.ListBuffer
```

```
val xs = List(0, 5, 8, 3, 2, 1)
```

```
val f = (x: Int) => x + 1
```

```
// Using a for loop:
```

```
val result = new ListBuffer[Int]()
```

```
for (x <- xs) result += f(x)
```

```
println(result)
```

```
// Using map:
```

```
println(xs.map(f))
```

```
// Using for/yield
```

```
println(for (x <- xs) yield f(x))
```

A tiny practical example

Read all CSV files in a folder

R:

```
library(tidyverse)
list.files(pattern = "*.csv$") %>% map(read_csv) %>% bind_rows()
```

Julia:

```
using DataFrames, CSV
vcat([CSV.read(f) for f = readdir() if endswith(f, ".csv")]....)
```

Mapping over two collections

Python

```
xs = [0, 5, 8, 3, 2, 1]  
ys = [9, 4, 1, 6, 7, 8]
```

```
# For loop version  
result = []  
for x, y in zip(xs, ys):  
    result.append(x + y)  
print(result)
```

```
# List comprehension version  
print([x + y for x, y in zip(xs, ys)])
```

```
# Map version  
import operator  
print(list(map(operator.add, xs, ys)))
```


Julia

```
xs = [0, 5, 8, 3, 2, 1]  
ys = [9, 4, 1, 6, 7, 8]
```

```
# For loop version:
```

```
result = []  
for (x, y) = zip(xs, ys)  
    push!(result, x + y)  
end  
println(result)
```

```
# List comprehension version:
```

```
[x + y for (x, y) = zip(xs, ys)] |> println
```

```
# Map version:
```

```
map(+, xs, ys) |> println
```

```
# Broadcast version:
```

```
xs .+ ys |> println
```

R

```
xs <- c(0, 5, 8, 3, 2, 1)
ys <- c(9, 4, 1, 6, 7, 8)
```

```
# For loop version
```

```
result <- c()
for (i in seq_along(xs)) {
  result <- c(result, xs[i] + ys[i])
}
print(result)
```

```
# Base version, using mapply
```

```
print(mapply(`+`, xs, ys))
```

```
# Using purrr::map
```

```
library(purrr)
xs %>% map2_dbl(ys, `+`) %>% print
```

```
# Using dplyr:
```

```
library(dplyr)
tibble(x = xs, y = ys) %>% transmute(x + y) %>% .[[1]]
```

NetLogo

```
to demo-map2
```

```
  let xs [0 5 8 3 2 1]
```

```
  let ys [9 4 1 6 7 8]
```

```
  ; for loop version:
```

```
  let result []
```

```
  foreach range length xs [ i ->
```

```
    set result lput (item i xs + item i ys) result
```

```
]
```

```
  print result
```

```
  ; map version:
```

```
  print (map + xs ys)
```

```
end
```

Java

```
import java.util.Arrays;
import java.util.ArrayList;
import java.util.List;
import java.util.stream.Collectors;
import java.util.stream.IntStream;

class DemoMap {
    public static void main(String[] args) {

        List<Integer> xs = Arrays.asList(0, 5, 8, 3, 2, 1);
        List<Integer> ys = Arrays.asList(9, 4, 1, 6, 7, 8);

        // Using a for loop:
        List<Integer> result = new ArrayList<>();
        for (int i = 0; i < xs.size(); i++) {
            result.add(xs.get(i) + ys.get(i));
        }
        System.out.println(result);

        // Using map:
        System.out.println(IntStream.range(0, xs.size())
            .mapToObj(i -> xs.get(i) + ys.get(i))
            .collect(Collectors.toList()));
    }
}
```

Scala

```
import collection.mutable.ListBuffer

val xs = List(0, 5, 8, 3, 2, 1)
val ys = List(9, 4, 1, 6, 7, 8)

// Using a for loop:
val result = new ListBuffer[Int]()
for ((x, y) <- xs.zip(ys)) result += x + y
println(result)

// Using map:
println(xs.zip(ys).map { case (x, y) => x + y })

// Using for/yield
println(for ((x, y) <- xs.zip(ys)) yield x + y)
```

An iceberg floating in a blue ocean under a blue sky with wispy clouds. The tip of the iceberg is above the water line, while the much larger, more complex structure is submerged below. The word 'MAP' is written in white, bold, sans-serif font with a black outline on the visible tip. The words 'FUNCTIONAL PROGRAMMING' are written in the same style at the bottom of the image.

MAP

**FUNCTIONAL
PROGRAMMING**

Code and slides:

<https://github.com/nicolaspayette/demo-map>