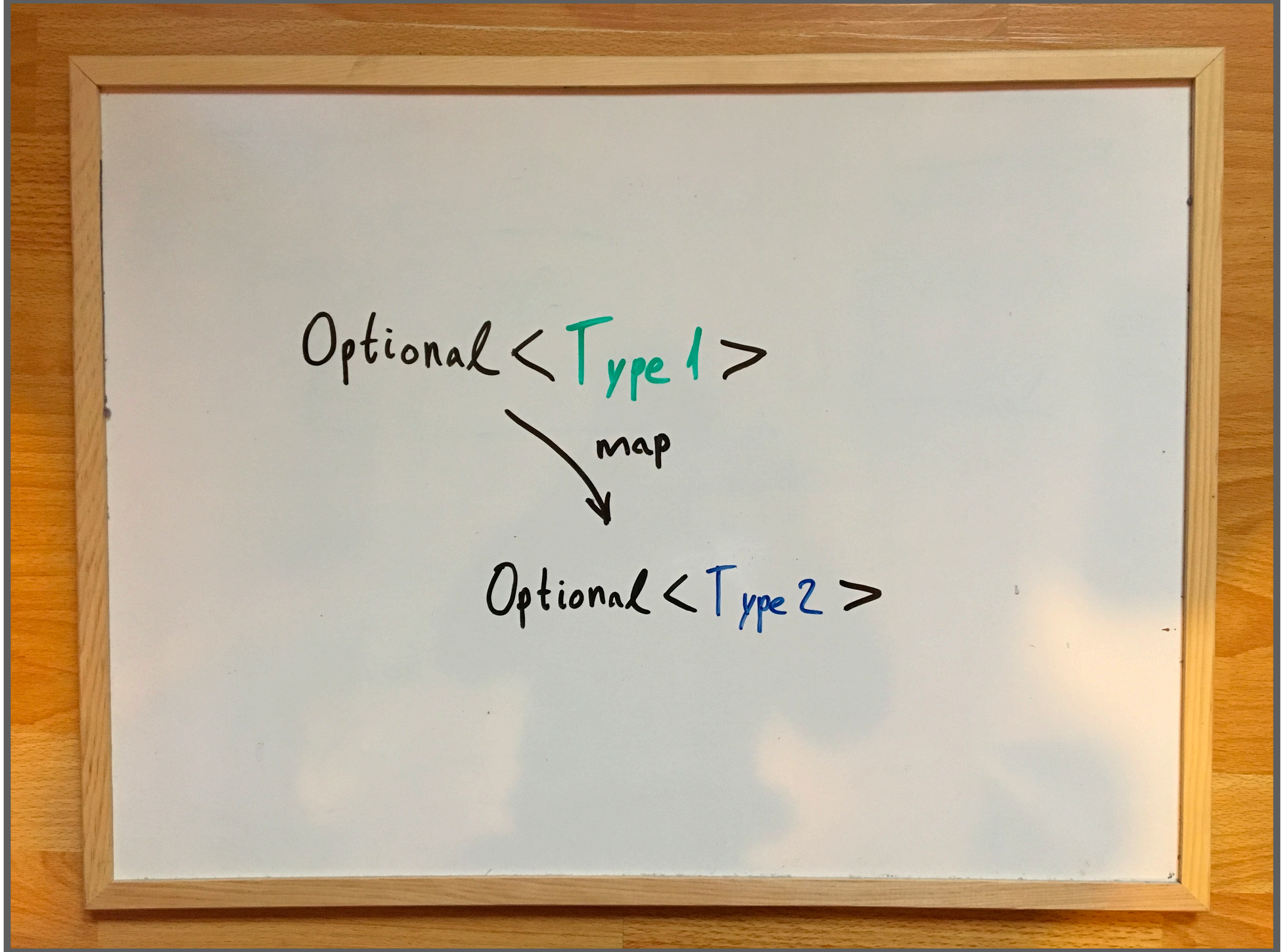
# Swift
# Higher Order Functions

Сбербанк Онлайн

Devteam


Optional<Type 1>
map
Optional<Type 2>

# Function Types

# Function Types

```
func addTwoInts(_ a: Int, _ b: Int) -> Int {
    return a + b
}
func multiplyTwoInts(_ a: Int, _ b: Int) -> Int {
    return a * b
}
```

# Function Types

```
(Int, Int) -> Int
```

# Using Function Types

```
(Int, Int) -> Int
```

```
var mathFunction: (Int, Int) -> Int = addTwoInts
```

```
let anotherMathFunction = addTwoInts
// anotherMathFunction is inferred to be of type (Int, Int) -> Int
```

# Using Function Types

- as parameter types
- as return types

```
func printMath
    print("Res
}
```

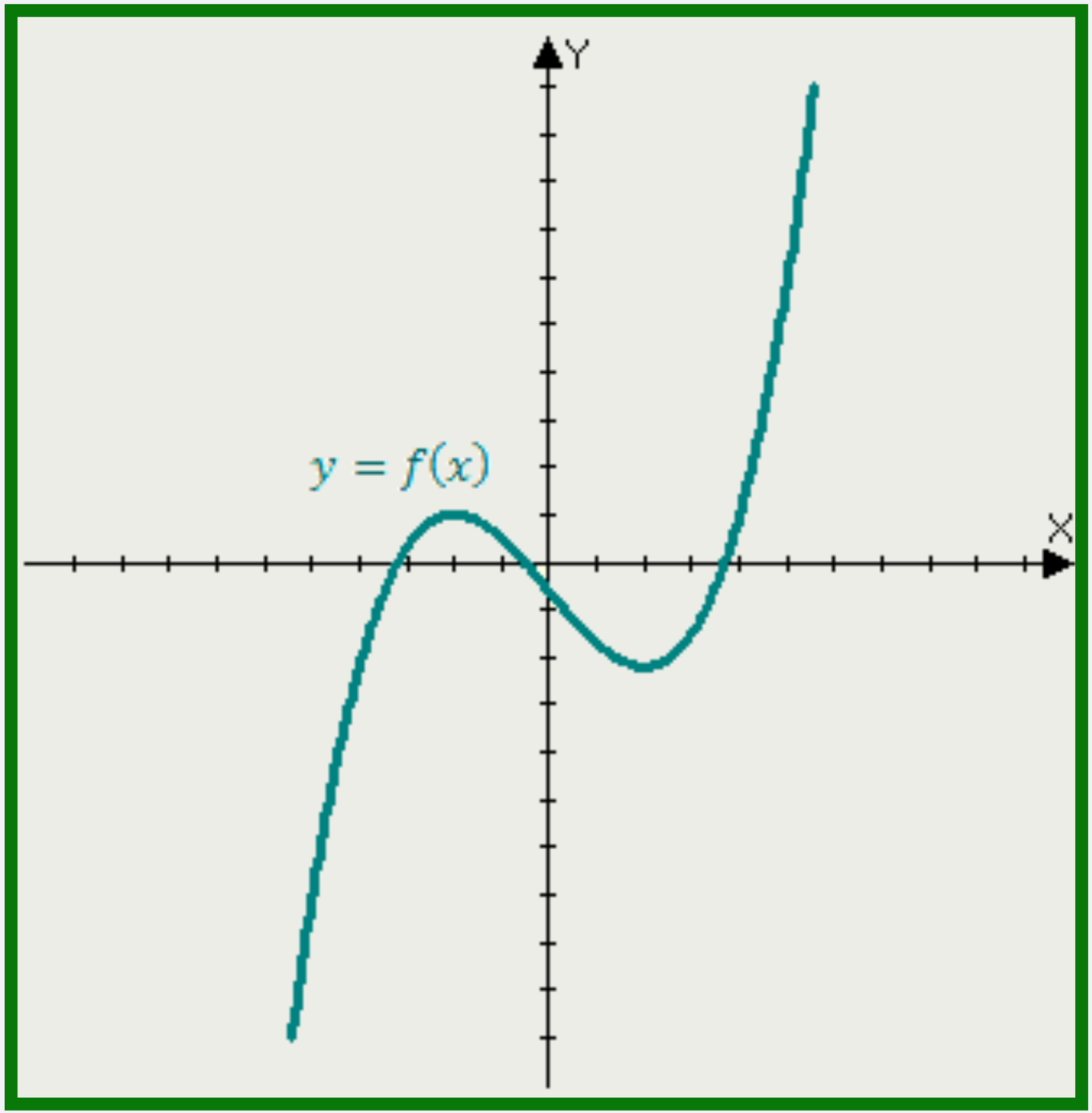
```
var currentValue = -4
let moveNearerToZero = chooseStepFunction(backward: currentValue > 0)
// moveNearerToZero now refers to the nested stepForward() function
while currentValue != 0 {
    print("\(currentValue)... ")
    currentValue = moveNearerToZero(currentValue)
}
print("zero!")
// -4...
// -3...
// -2...
// -1...
// zero!
```

```
t, _ b: Int) {
```

Сбербанк Онлайн

Devteam

# Functional programming



# Pure function

- A function must always take an argument
- A function must always return a value
- A function must not depend on anything that changes
- A function must not change anything by itself

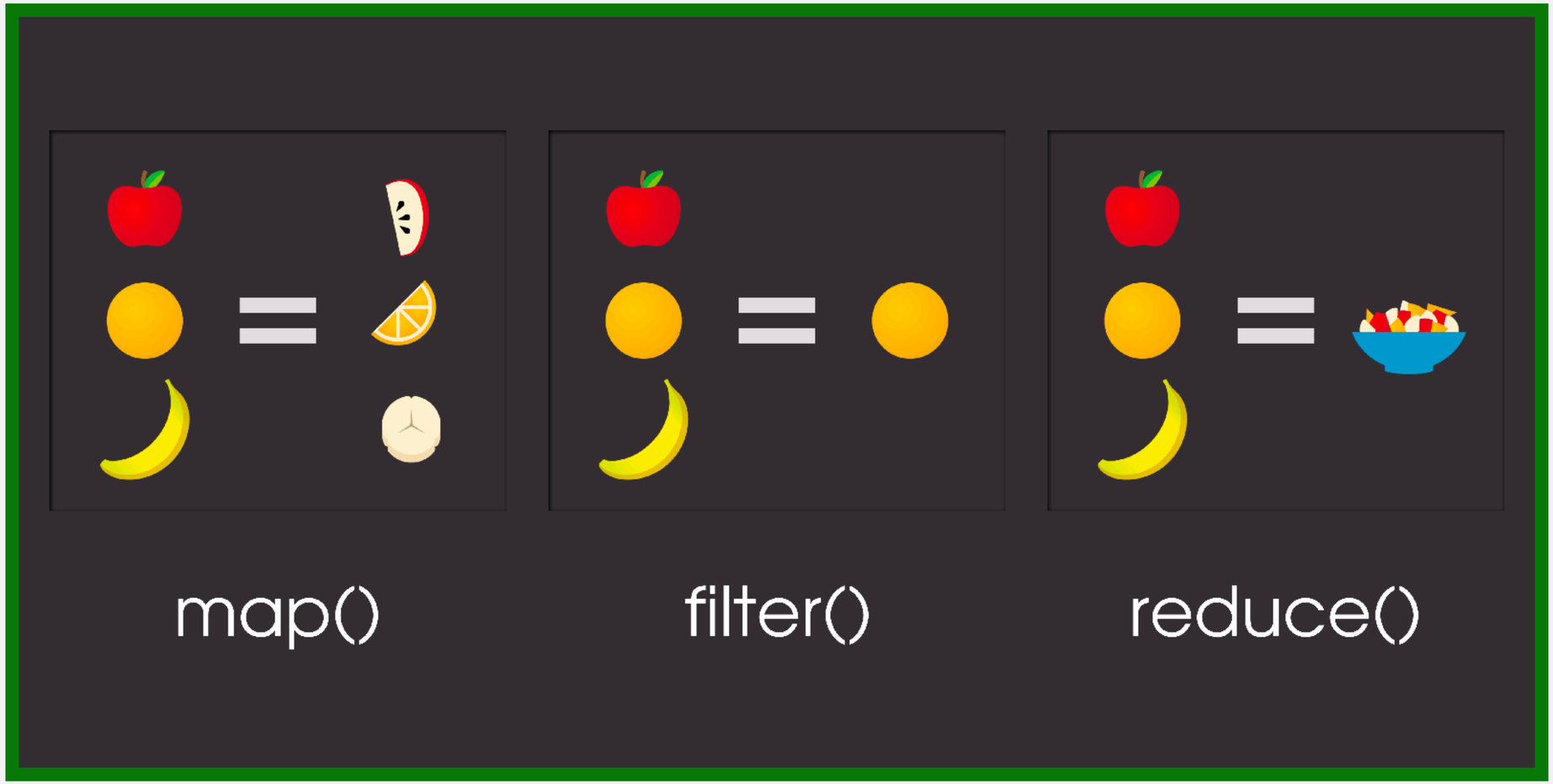
# Pure function

```
func add(x: Int, y: Int) -> Int { x + y }
```

# First-Class Functions

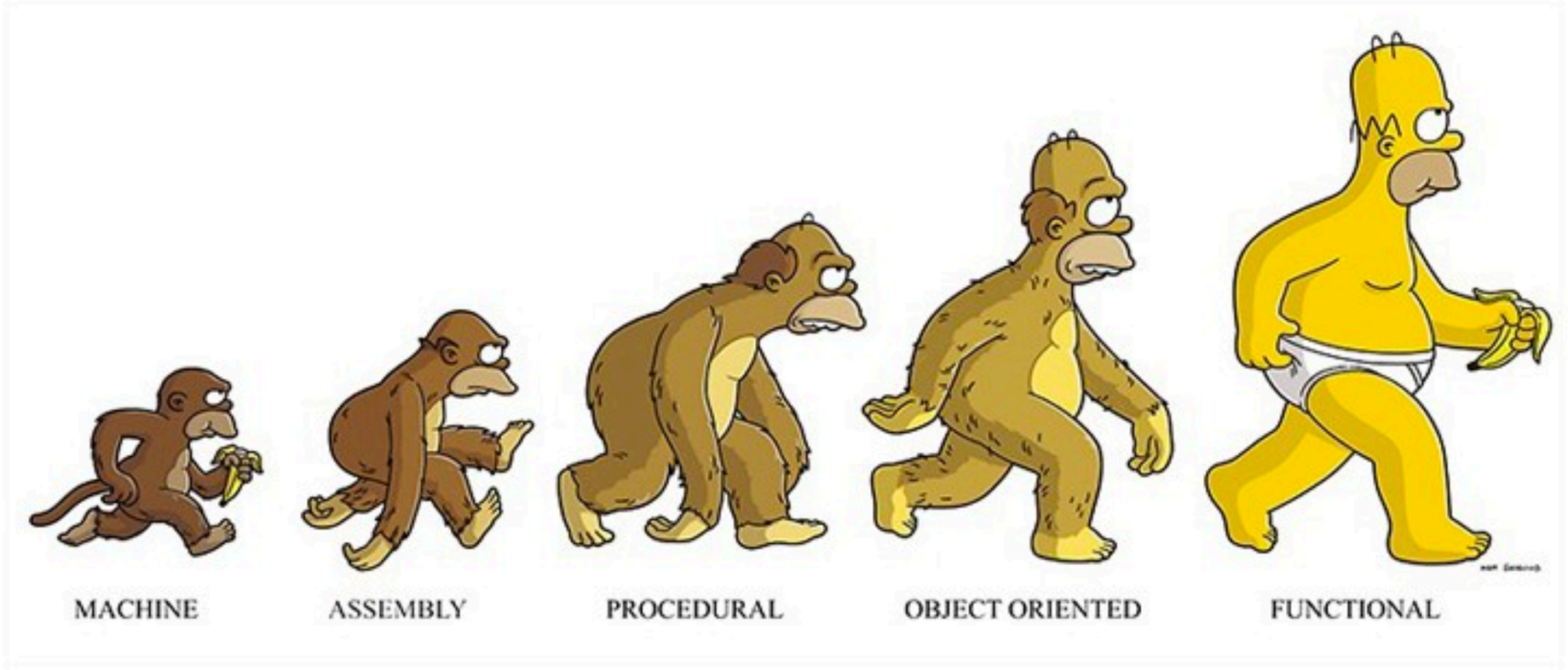
```
let multiply = { (x: Int, y: Int) -> Int in x * y }

multiply(2, 3) // 6
```

# Higher-Order Functions



# Currying and Partial Application

```
func add(_ x: Int) -> (_ y: Int) -> Int {
    { y in return x + y }
}

let addTwo = add(2)
let oneTwoThree = [1, 2, 3]
let threeFourFive = oneTwoThree.map(addTwo) // [3, 4, 5]
```



# Functional Programming

- functions
- effects
- data

Сбербанк Онлайн

Devteam

# Functional Programming

```swift
// Imperative Programming
var total = 0
for i in 1...10 {
    total += i
}
print("Imperative: total: ", total); // prints 55
```

```swift
// Functional Programming
func sum(start: Int, end: Int, total: Int) -> Int {
    if (start > end) {
        return total;
    }
    return sum(start: start + 1, end: end, total: total + start)
}
print("Functional: total: ", sum(start: 1, end: 10, total: 0)); // prints 55
```

# Swift functional programming

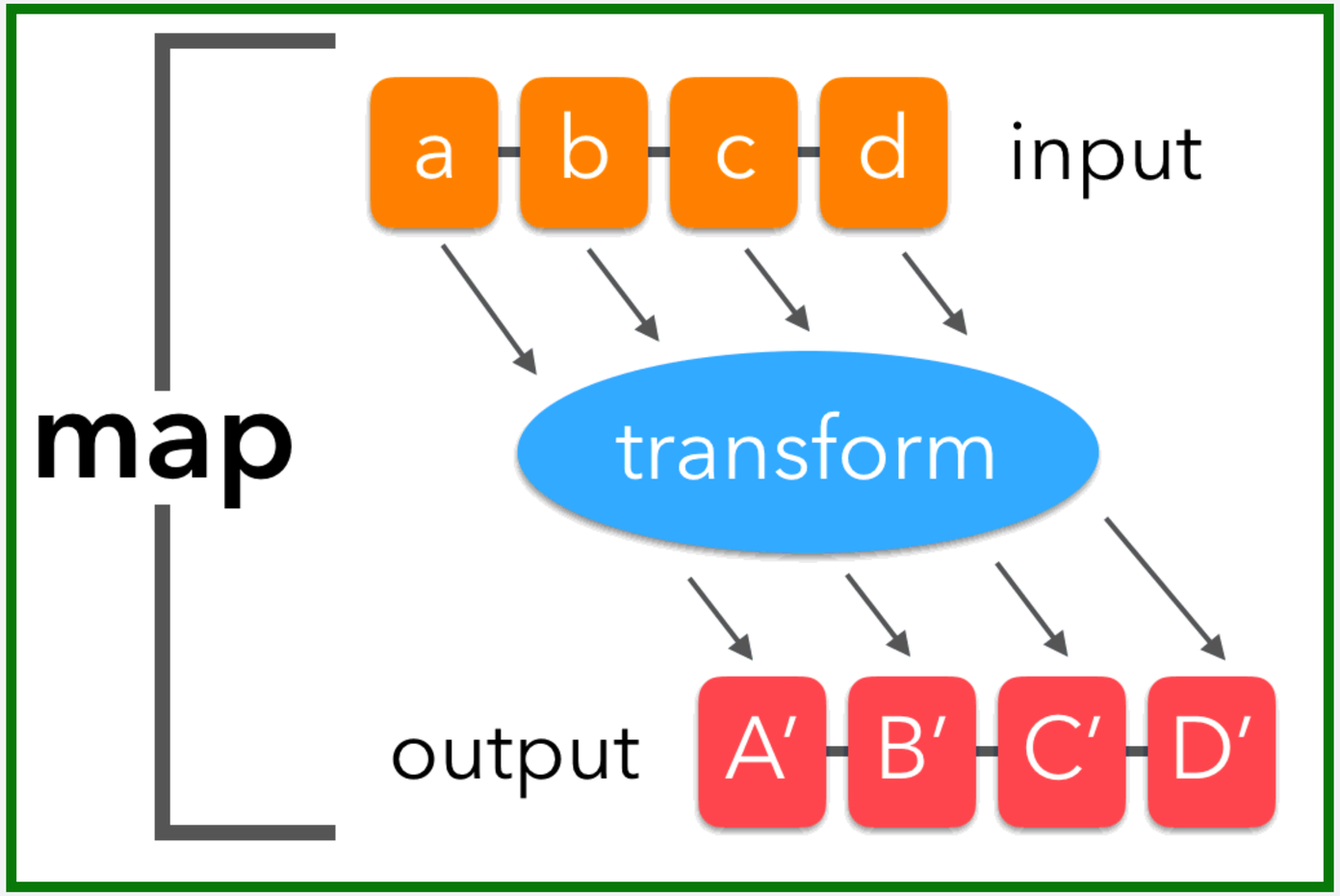

Devteam

- map
- compactMap
- flatMap
- filter
- reduce
- forEach
- contains
- removeAll
- sorted
- split


Сбербанк Онлайн

Devteam

# Map



# map(_:)

Returns an array containing the results of mapping the given closure over the sequence's elements.

---

## Declaration

```
func map<T>(_ transform: (Element) throws -> T) rethrows -> [T]
```

## Parameters

`transform`
: A mapping closure. `transform` accepts an element of this sequence as its parameter and returns a transformed value of the same or of a different type.

## Return Value

An array containing the transformed elements of this sequence.

# Map

```
["hello", "world"].map { $0.uppercased() } // output: ["HELLO", "WORLD"]

[1, 2, 3].map { $0 * $0 } // output: [1, 4, 9]

[15.2, 2.5, 10.0].map(Int.init) // same output: [15, 2, 10]
```

# Map

```
class Tester {
    var name: String
    var age: Int

    init(name: String, age: Int) {
        self.name = name
        self.age = age
    }
}

let testers = [Tester(name: "John", age: 23),
               Tester(name: "Lucy", age: 25),
               Tester(name: "Tom", age: 32),
               Tester(name: "Mike", age: 29),
               Tester(name: "Hellen", age: 19),
               Tester(name: "Jim", age: 35)]

let ages = testers.map { $0.age }
print(ages)
// [23, 25, 32, 29, 19, 35]
```

# mapValues(_:)

Returns a new dictionary containing the keys of this dictionary with the values transformed by the given closure.

---

## Declaration

```
func mapValues<T>(_ transform: (Value) throws -> T) rethrows -> Dictionary<Key,
    T>
```

## Parameters

`transform`
: A closure that transforms a value. `transform` accepts each value of the dictionary as its parameter and returns a transformed value of the same or of a different type.

## Return Value

A dictionary containing the keys and transformed values of this dictionary.

# MapValues

```swift
var info = [String: String]()
info["name"] = "andrew"
info["city"] = "berlin"
info["job"] = "developer"
info["hobby"] = "computer games"

let updatedInfo = info.mapValues { $0.capitalized }
print(updatedInfo)
// ["name": "Andrew", "hobby": "Computer Games", "job": "Developer", "city": "Berlin"]
```

# Map

```
extension Array {
  // T is the output type
  func myMap<T>(_ transform: (Element) -> T) -> [T] {
    var result: [T] = []

    for item in self {
      result.append(transform(item))
    }

    return result
  }
}
```

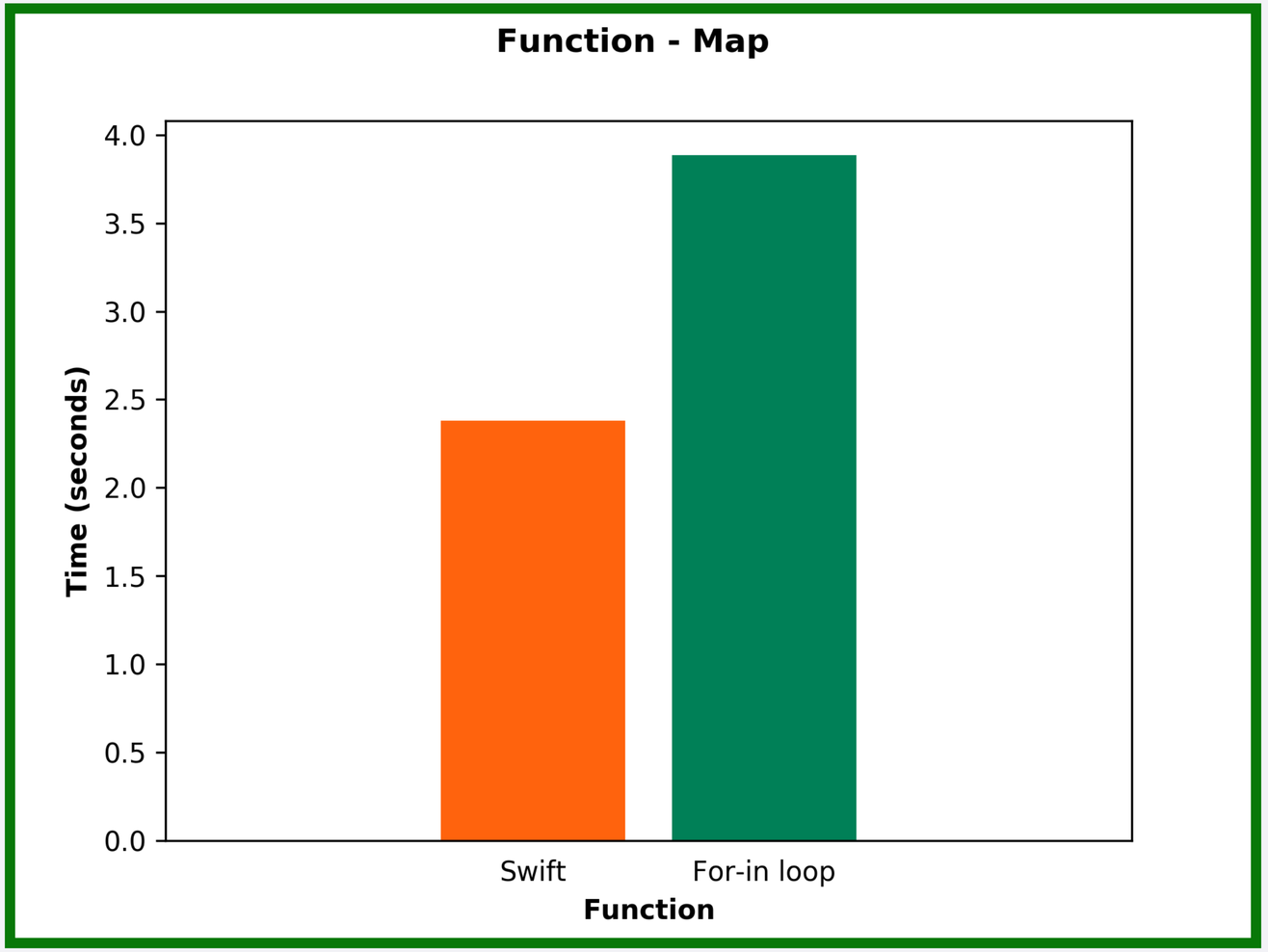
# Map. Performance

```
let elementsCount = 10_000_000
let fahrenheit = Array(repeating: 0.0, count: elementsCount).map { _ -> Double in
    return Double.random(in: 32.0 ... 113.0)
}

var fahrenheits = [[Double]]()
for _ in 0 ..< 10_000_000 {
    fahrenheits.append(Array(repeating: 0.0, count: 1).map { _ -> Double in
        return Double.random(in: 32.0 ... 113.0)
    })
}
```

# Map. Performance

```swift
// Swift
let celsius = fahrenheit.map { (degreesFahrenheit) -> Double in
    return (degreesFahrenheit - 32.0) / 1.8
}

// For-in loop
var celsius = [Double]()
for degreesFahrenheit in fahrenheit {
    celsius.append((degreesFahrenheit - 32.0) / 1.8)
}
```

# Map. Performance




Сбербанк Онлайн

Devteam

# Работа в группах

# Задача

```
let result = [1, 10, 5, -5, 2, 7, 12, 20, 0, -10]
            .filter { $0 < 10 }
            .map { String($0) }
            .reduce("") { (result, num) -> String in
                return result + num
            }
print(result)
```

**15-5270-10**

# Read

- https://www.vadimbulavin.com/swift-functional-programming-fundamentals/
- https://www.vadimbulavin.com/pure-functions-higher-order-functions-and-first-class-functions-in-swift/
- https://www.appcoda.com/higher-order-functions-swift/
- https://www.raywenderlich.com/9222-an-introduction-to-functional-programming-in-swift
- https://www.skoumal.com/en/performance-of-built-in-higher-order-functions-map-filter-reduce-and-flatmap-vs-for-in-loop-in-swift/
- https://habr.com/ru/post/440722/
- https://www.swiftindepth.com/2019-01-27/stop-filtering

- https://habr.com/ru/company/oleg-bunin/blog/462505/
- https://swiftunboxed.com/open-source/map/
- https://developer.apple.com/videos/play/wwdc2018/223/

# Домашняя работа

Есть **три** основные функции высшего порядка

- map
- filter
- reduce

**Две** из них можно выразить через **третью**. Какую? Как это можно сделать?

**Срок: до 7 июля**

Сбербанк Онлайн

Devteam