Meet the Swift Algorithm and Collections Package

- Algorithms

Algorithms

1. Map
2. CompactMap
3. FlatMap
4. Lazy
5. Windows
6. AdjacentPairs
7. Chunks
8. Chunked

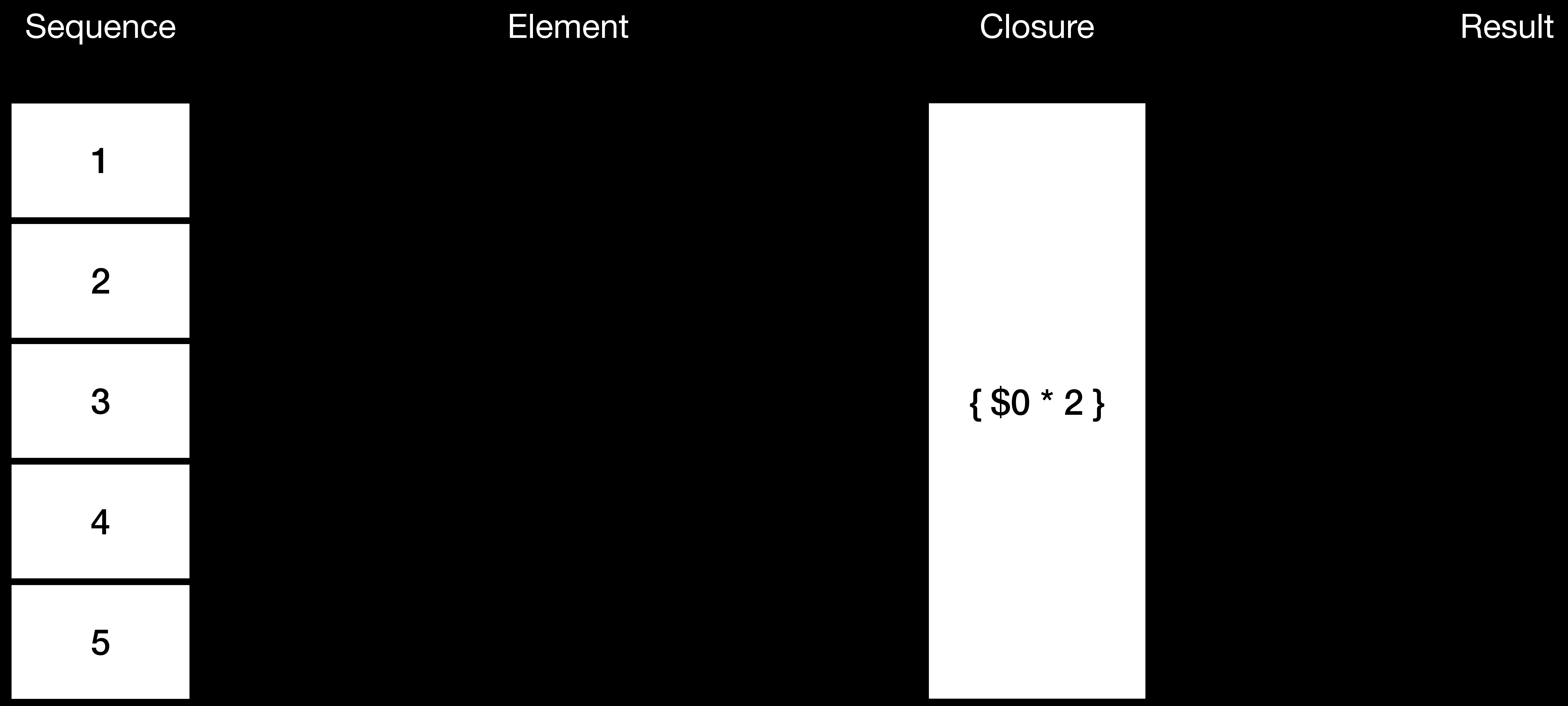
Map

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

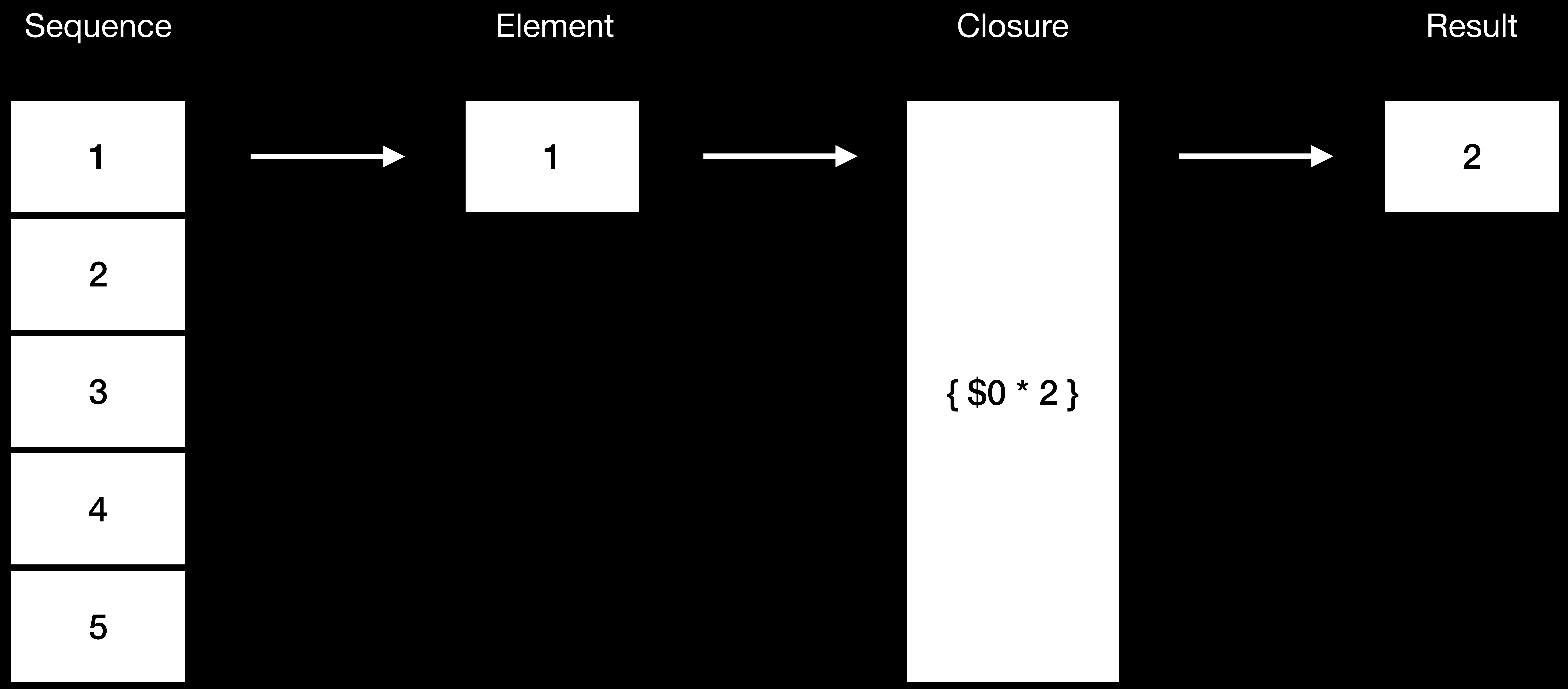
Returns an array containing the results of mapping the given closure over the sequence’s elements.

Sequence의 Element를 Closure로 연산한 결과를 배열로 반환

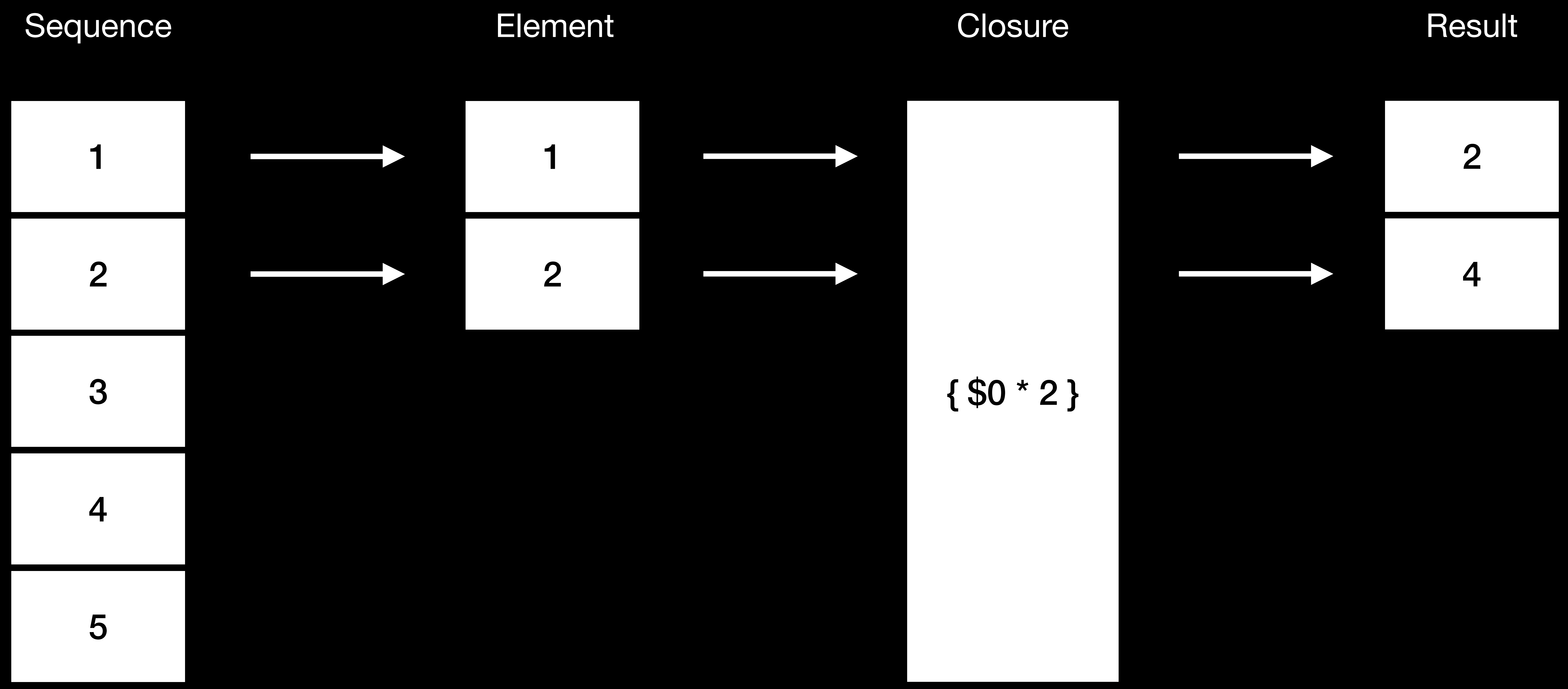
Map



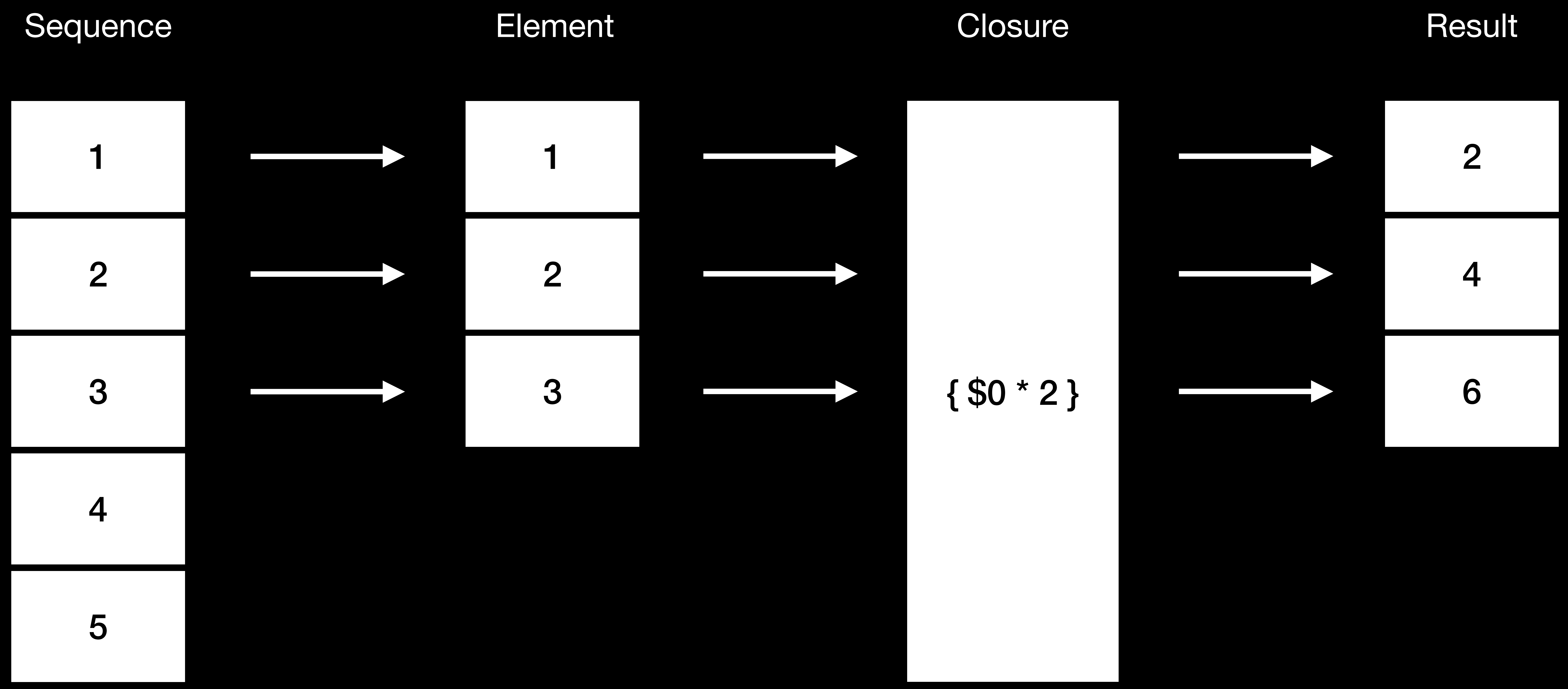
Map



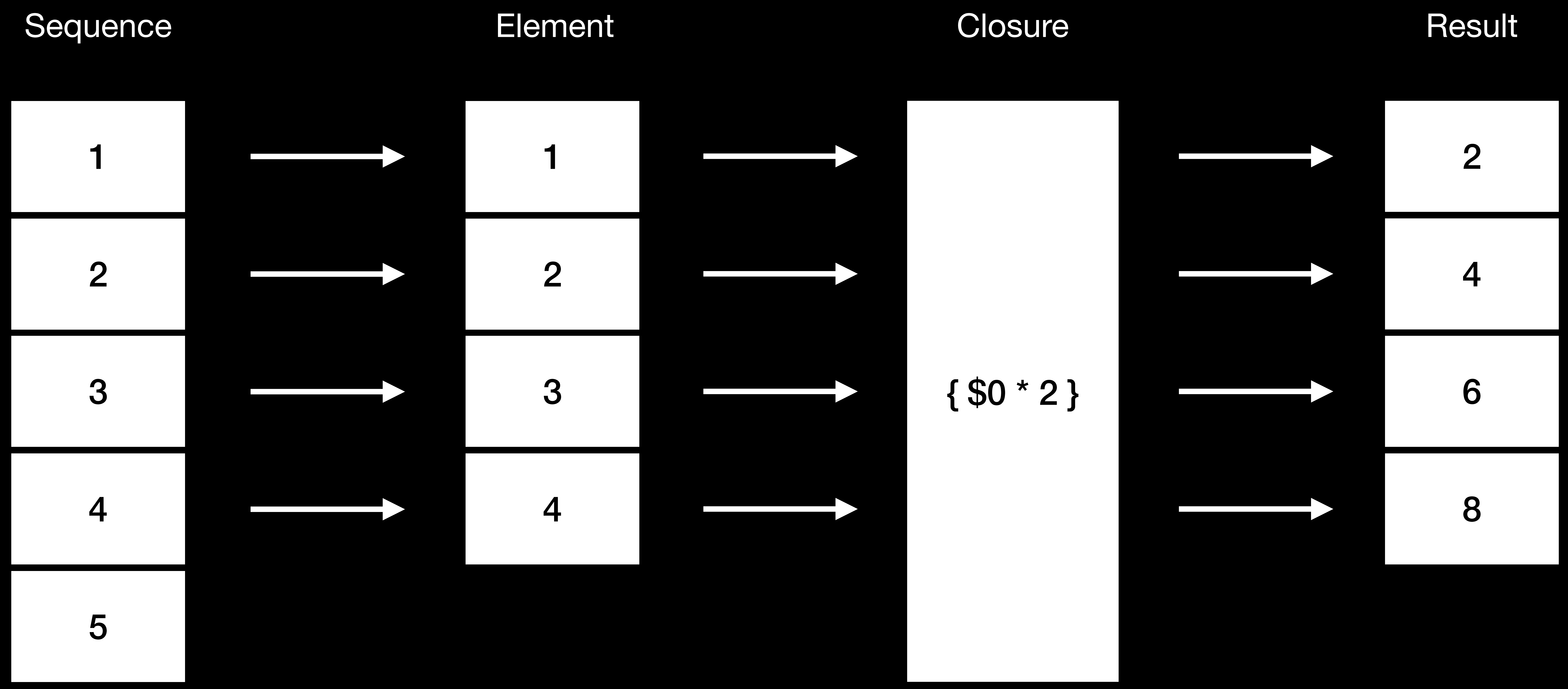
Map



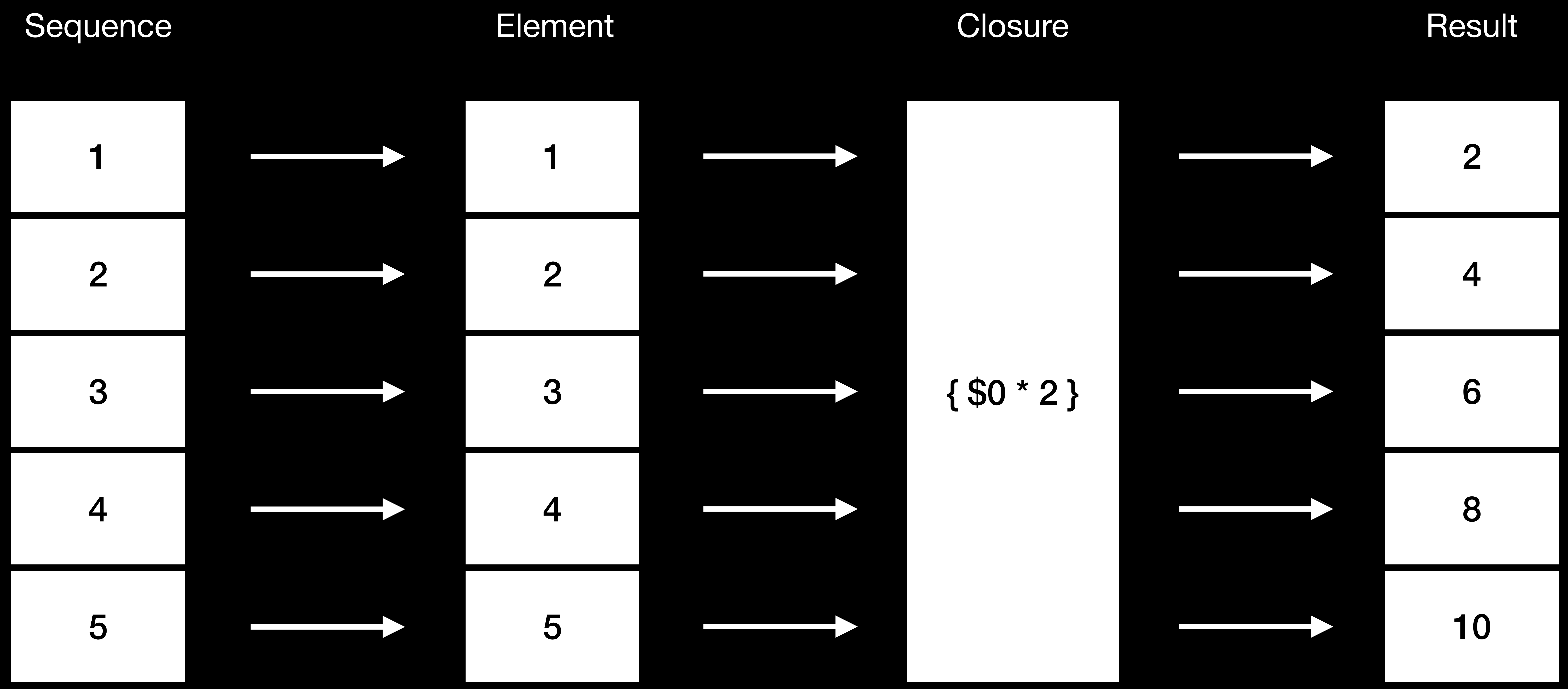
Map



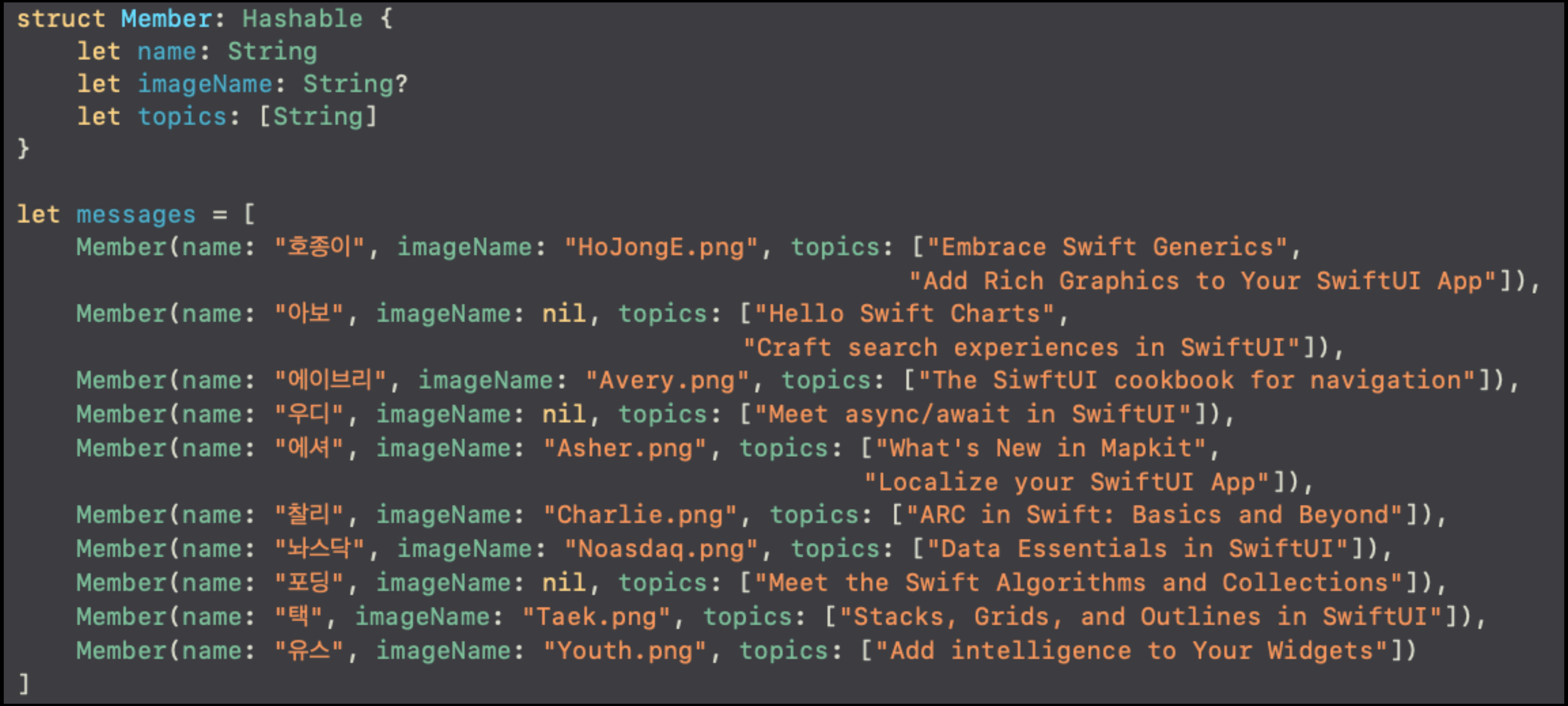
Map



Map



Sample Data

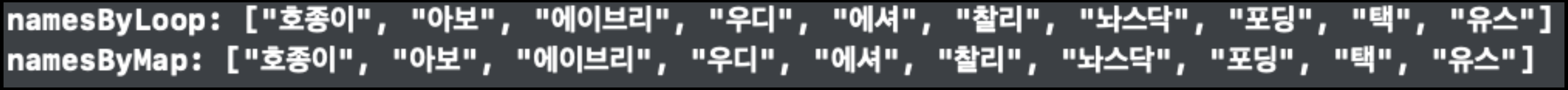


```swift
struct Member: Hashable {
    let name: String
    let imageName: String?
    let topics: [String]
}

let messages = [
    Member(name: "호종이", imageName: "HoJongE.png", topics: ["Embrace Swift Generics",
                                                           "Add Rich Graphics to Your SwiftUI App"]),
    Member(name: "아보", imageName: nil, topics: ["Hello Swift Charts",
                                                "Craft search experiences in SwiftUI"]),
    Member(name: "에이브리", imageName: "Avery.png", topics: ["The SiwftUI cookbook for navigation"]),
    Member(name: "우디", imageName: nil, topics: ["Meet async/await in SwiftUI"]),
    Member(name: "에셔", imageName: "Asher.png", topics: ["What's New in Mapkit",
                                                        "Localize your SwiftUI App"]),
    Member(name: "찰리", imageName: "Charlie.png", topics: ["ARC in Swift: Basics and Beyond"]),
    Member(name: "놔스닥", imageName: "Noasdaq.png", topics: ["Data Essentials in SwiftUI"]),
    Member(name: "포딩", imageName: nil, topics: ["Meet the Swift Algorithms and Collections"]),
    Member(name: "택", imageName: "Taek.png", topics: ["Stacks, Grids, and Outlines in SwiftUI"]),
    Member(name: "유스", imageName: "Youth.png", topics: ["Add intelligence to Your Widgets"])
]
```

Map

```
//  Raw Loop
var namesByLoop: [String]  = []
for member in members {
    namesByLoop.append(member.name)
}

//  Map
let namesByMap = members.map { $0.name }

print("namesByLoop: \(namesByLoop)")
print("namesByMap: \(namesByMap)")
```

Map



```
namesByLoop: ["호종이", "아보", "에이브리", "우디", "에셔", "찰리", "놔스닥", "포딩", "택", "유스"]
namesByMap: ["호종이", "아보", "에이브리", "우디", "에셔", "찰리", "놔스닥", "포딩", "택", "유스"]
```

Map

코드가 간단하다

배열을 let으로 선언할 수 있다

배열 크기를 미리 할당해서 성능이 좋다

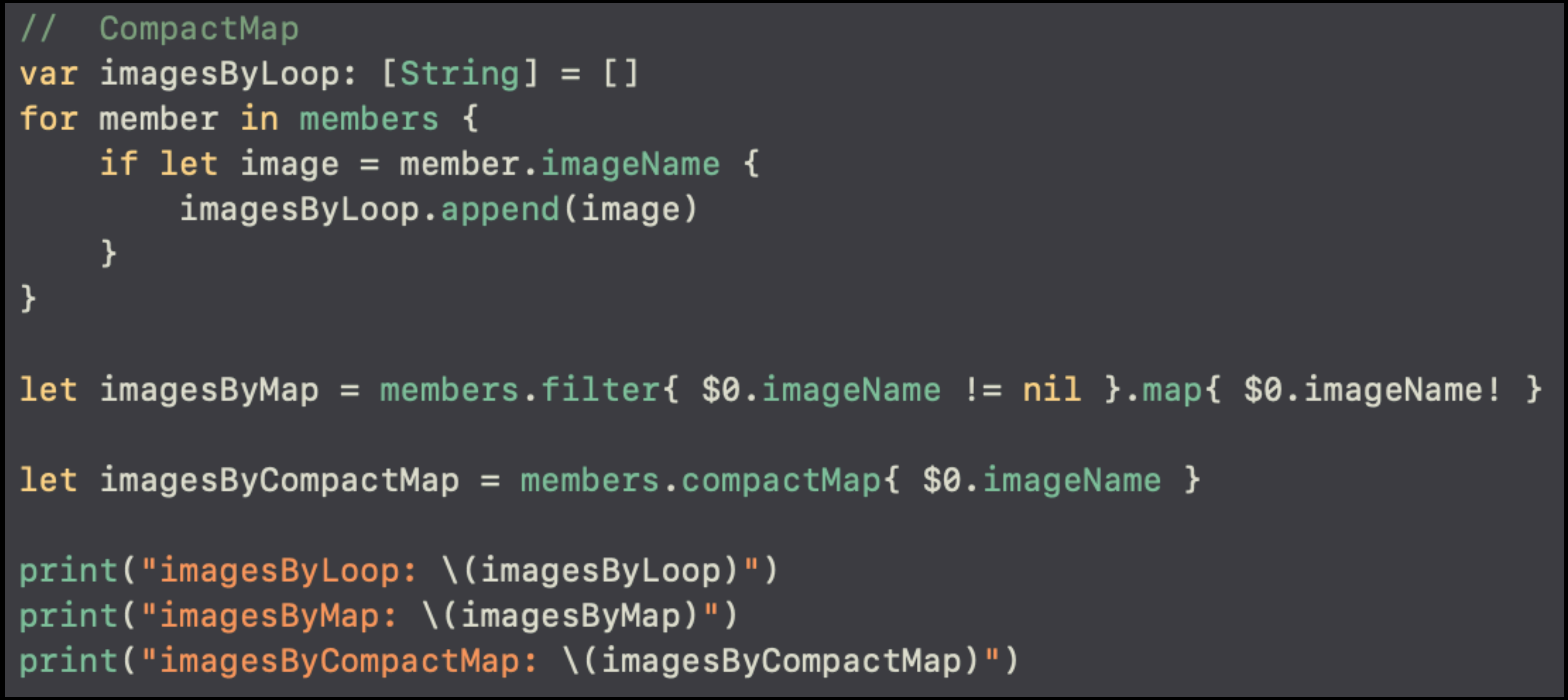
CompactMap

```
func compactMap<ElementOfResult>(_ transform: (Self.Element) throws -> ElementOf
Result?) rethrows -> [ElementOfResult]
```

Returns an array containing the non-`nil` results of calling the given transformation with each element of this sequence.

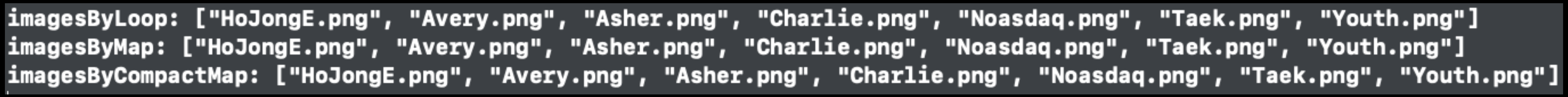
Sequence 각각의 Element를 Closure에 연산한 결과 중 nil이 아닌 값만 반환
Optional Binding + Map

CompactMap



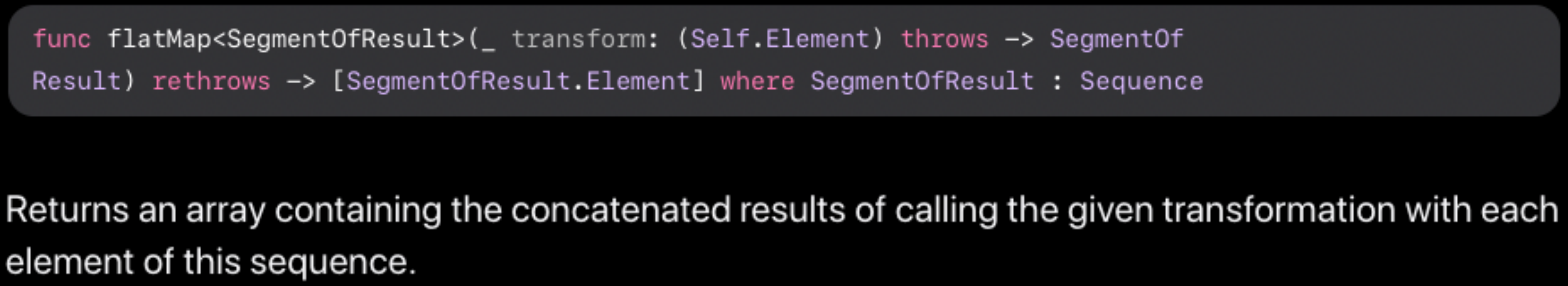
```swift
//  CompactMap
var imagesByLoop: [String] = []
for member in members {
    if let image = member.imageName {
        imagesByLoop.append(image)
    }
}

let imagesByMap = members.filter{ $0.imageName != nil }.map{ $0.imageName! }

let imagesByCompactMap = members.compactMap{ $0.imageName }

print("imagesByLoop: \(imagesByLoop)")
print("imagesByMap: \(imagesByMap)")
print("imagesByCompactMap: \(imagesByCompactMap)")
```

CompactMap



```
imagesByLoop: ["HoJongE.png", "Avery.png", "Asher.png", "Charlie.png", "Noasdaq.png", "Taek.png", "Youth.png"]
imagesByMap: ["HoJongE.png", "Avery.png", "Asher.png", "Charlie.png", "Noasdaq.png", "Taek.png", "Youth.png"]
imagesByCompactMap: ["HoJongE.png", "Avery.png", "Asher.png", "Charlie.png", "Noasdaq.png", "Taek.png", "Youth.png"]
```

FlatMap

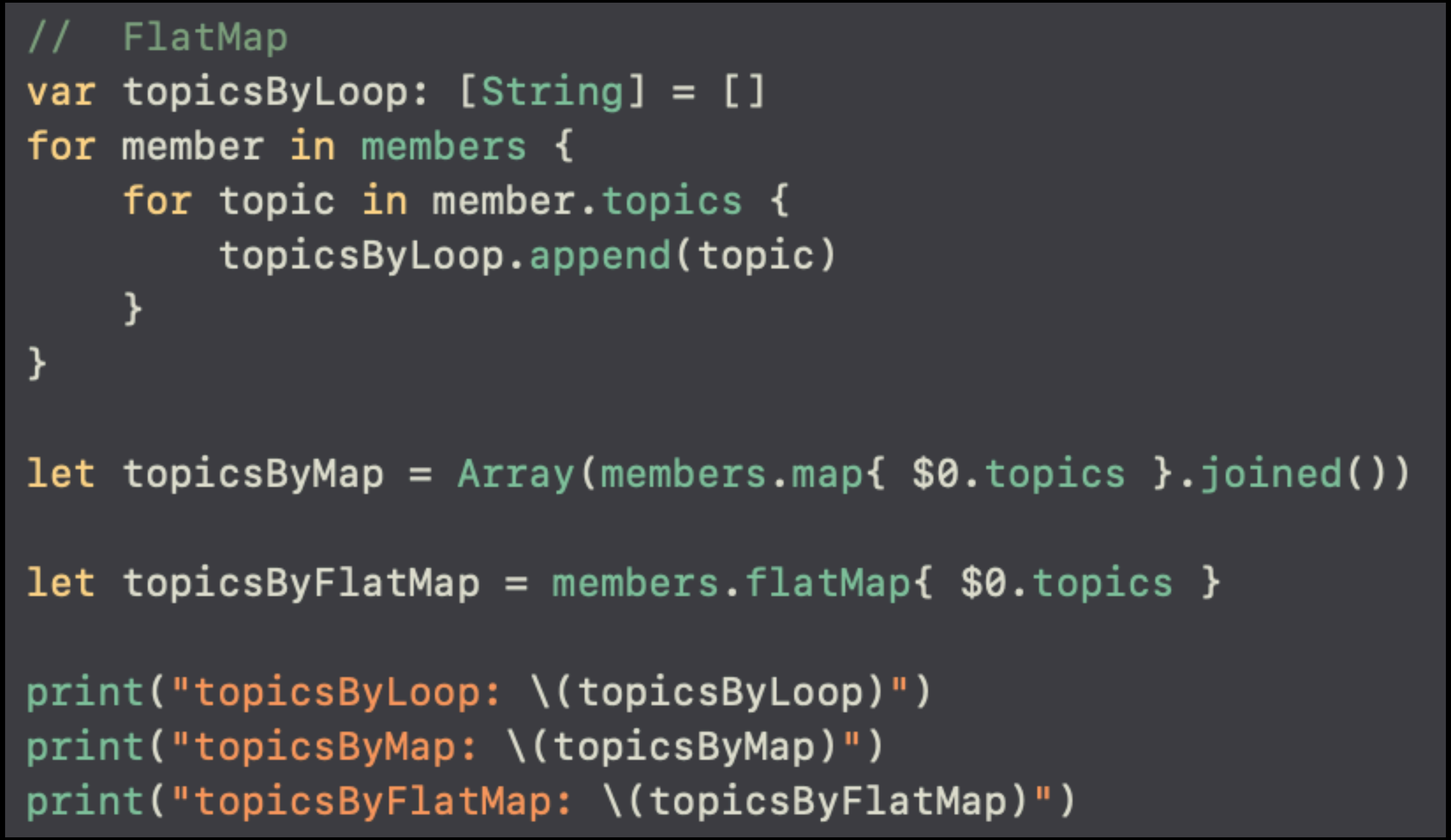


```
func flatMap<SegmentOfResult>(_ transform: (Self.Element) throws -> SegmentOf
Result) rethrows -> [SegmentOfResult.Element] where SegmentOfResult : Sequence
```

Returns an array containing the concatenated results of calling the given transformation with each element of this sequence.

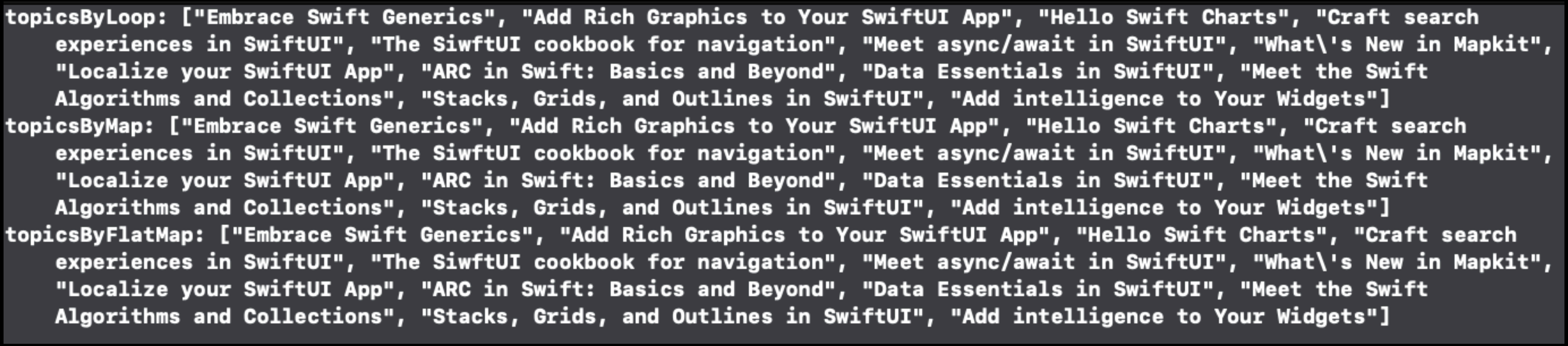
Sequence의 Element에 Closure를 연산한 결과를 배열로 반환
2차원 배열을 1차원으로 변환해서 반환

FlatMap



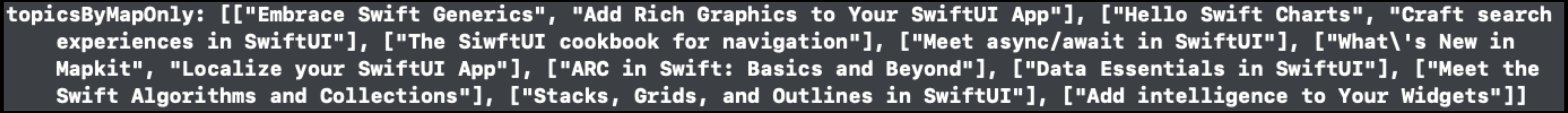
```swift
//  FlatMap
var topicsByLoop: [String] = []
for member in members {
    for topic in member.topics {
        topicsByLoop.append(topic)
    }
}

let topicsByMap = Array(members.map{ $0.topics }.joined())

let topicsByFlatMap = members.flatMap{ $0.topics }

print("topicsByLoop: \(topicsByLoop)")
print("topicsByMap: \(topicsByMap)")
print("topicsByFlatMap: \(topicsByFlatMap)")
```

FlatMap



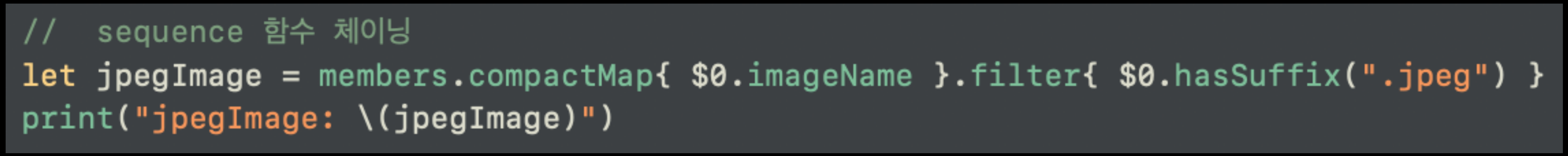
```
topicsByLoop: ["Embrace Swift Generics", "Add Rich Graphics to Your SwiftUI App", "Hello Swift Charts", "Craft search
    experiences in SwiftUI", "The SiwftUI cookbook for navigation", "Meet async/await in SwiftUI", "What\'s New in Mapkit",
    "Localize your SwiftUI App", "ARC in Swift: Basics and Beyond", "Data Essentials in SwiftUI", "Meet the Swift
    Algorithms and Collections", "Stacks, Grids, and Outlines in SwiftUI", "Add intelligence to Your Widgets"]
topicsByMap: ["Embrace Swift Generics", "Add Rich Graphics to Your SwiftUI App", "Hello Swift Charts", "Craft search
    experiences in SwiftUI", "The SiwftUI cookbook for navigation", "Meet async/await in SwiftUI", "What\'s New in Mapkit",
    "Localize your SwiftUI App", "ARC in Swift: Basics and Beyond", "Data Essentials in SwiftUI", "Meet the Swift
    Algorithms and Collections", "Stacks, Grids, and Outlines in SwiftUI", "Add intelligence to Your Widgets"]
topicsByFlatMap: ["Embrace Swift Generics", "Add Rich Graphics to Your SwiftUI App", "Hello Swift Charts", "Craft search
    experiences in SwiftUI", "The SiwftUI cookbook for navigation", "Meet async/await in SwiftUI", "What\'s New in Mapkit",
    "Localize your SwiftUI App", "ARC in Swift: Basics and Beyond", "Data Essentials in SwiftUI", "Meet the Swift
    Algorithms and Collections", "Stacks, Grids, and Outlines in SwiftUI", "Add intelligence to Your Widgets"]
```

FlatMap - Map만 사용했을 때



```
topicsByMapOnly: [["Embrace Swift Generics", "Add Rich Graphics to Your SwiftUI App"], ["Hello Swift Charts", "Craft search
    experiences in SwiftUI"], ["The SiwftUI cookbook for navigation"], ["Meet async/await in SwiftUI"], ["What\'s New in
    Mapkit", "Localize your SwiftUI App"], ["ARC in Swift: Basics and Beyond"], ["Data Essentials in SwiftUI"], ["Meet the
    Swift Algorithms and Collections"], ["Stacks, Grids, and Outlines in SwiftUI"], ["Add intelligence to Your Widgets"]]
```

Sequence Chaining


```
//  sequence 함수 체이닝
let jpegImage = members.compactMap{ $0.imageName }.filter{ $0.hasSuffix(".jpeg") }
print("jpegImage: \(jpegImage)")
```

문제점

compactMap 함수의 결과 배열을 한번 생성하고,
그 배열에 filter 함수를 다시 이용해서 다시 배열을 생성한다
-> 배열이 2번 선언된다

Lazy

```
var lazy: LazySequence<Self> { get }
```

A sequence containing the same elements as this sequence, but on which some operations, such as `map` and `filter`, are implemented lazily.

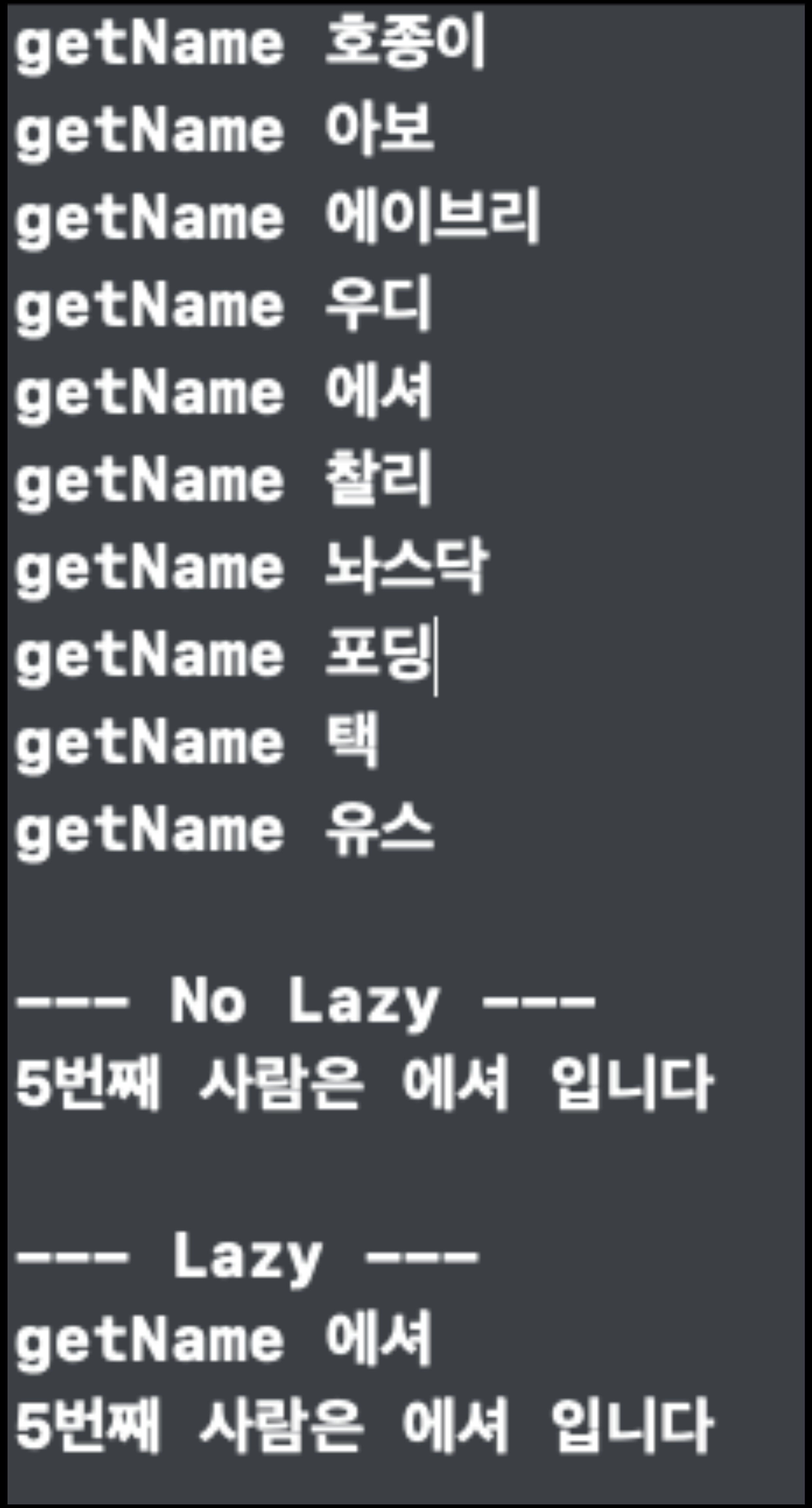
일반 시퀀스와 같은 원소를 가지지만, map과 filter와 같은 연산을 할때 lazily하게 처리

Lazy

```swift
func getName(_ member: Member) -> String {
    print("getName \(member.name)")
    return member.name
}
```

```swift
//  lazy
let names = members.map{ getName($0) }

let namesLazy = members.lazy.map{ getName($0) }

print("\n--- No Lazy ---")
print("5번째 사람은 \(names[4]) 입니다")
print("\n--- Lazy ---")
print("5번째 사람은 \(namesLazy[4]) 입니다")
```

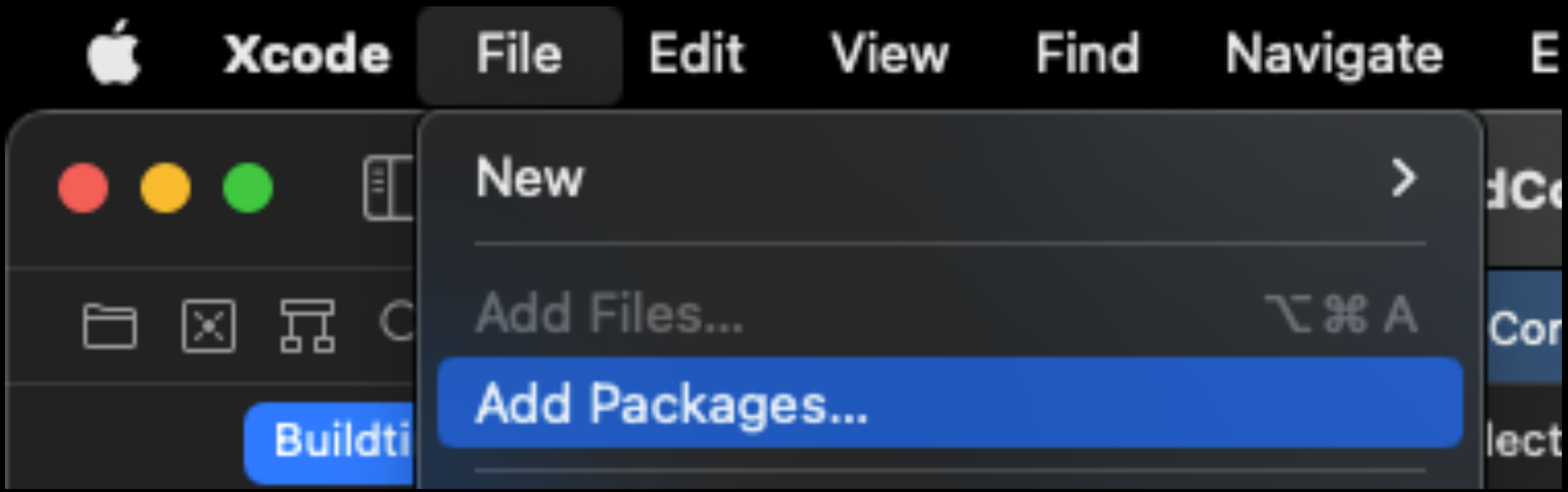
Lazy



```
getName 호종이
getName 아보
getName 에이브리
getName 우디
getName 에셔
getName 찰리
getName 놔스닥
getName 포딩
getName 택
getName 유스

--- No Lazy ---
5번째 사람은 에셔 입니다

--- Lazy ---
getName 에셔
5번째 사람은 에셔 입니다
```

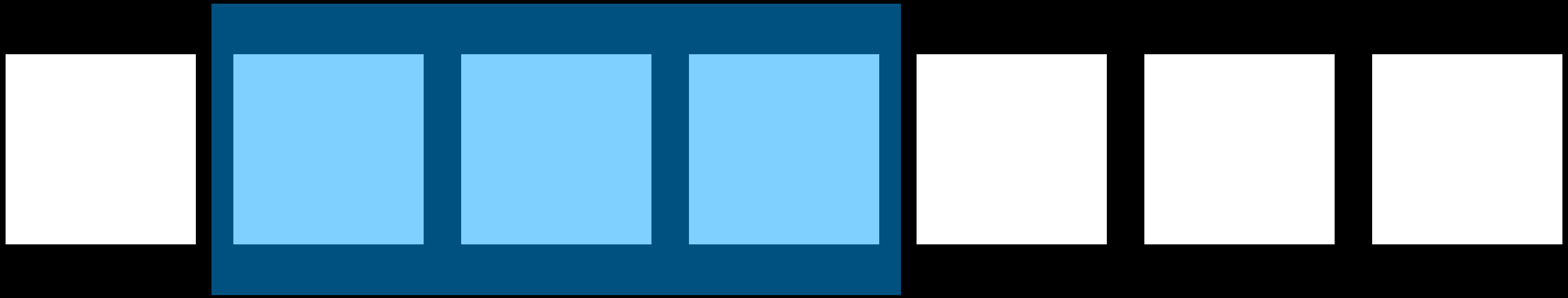
Sequence에서 필요한 값에만 접근해서 연산

Swift Algorithms Package



```
import Algorithms
```

Windows(ofCount: Int)



Windows

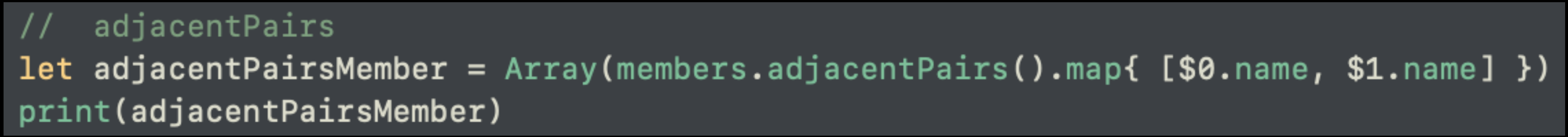
```swift
//  windows
let windowMember = Array(members.windows(ofCount: 3).map{ $0.map{ $0.name } })
print(windowMember)
```

```
[["호종이", "아보", "에이브리"], ["아보", "에이브리", "우디"], ["에이브리", "우디", "에셔"], ["우디", "에셔", "찰리"], ["에셔", "찰리",
    "놔스닥"], ["찰리", "놔스닥", "포딩"], ["놔스닥", "포딩", "택"], ["포딩", "택", "유스"]]
```

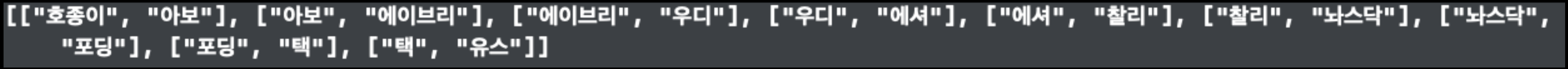
AdjacentPairs()

adjacentPairs() == windows(ofCount: 2)

AdjacentPairs()

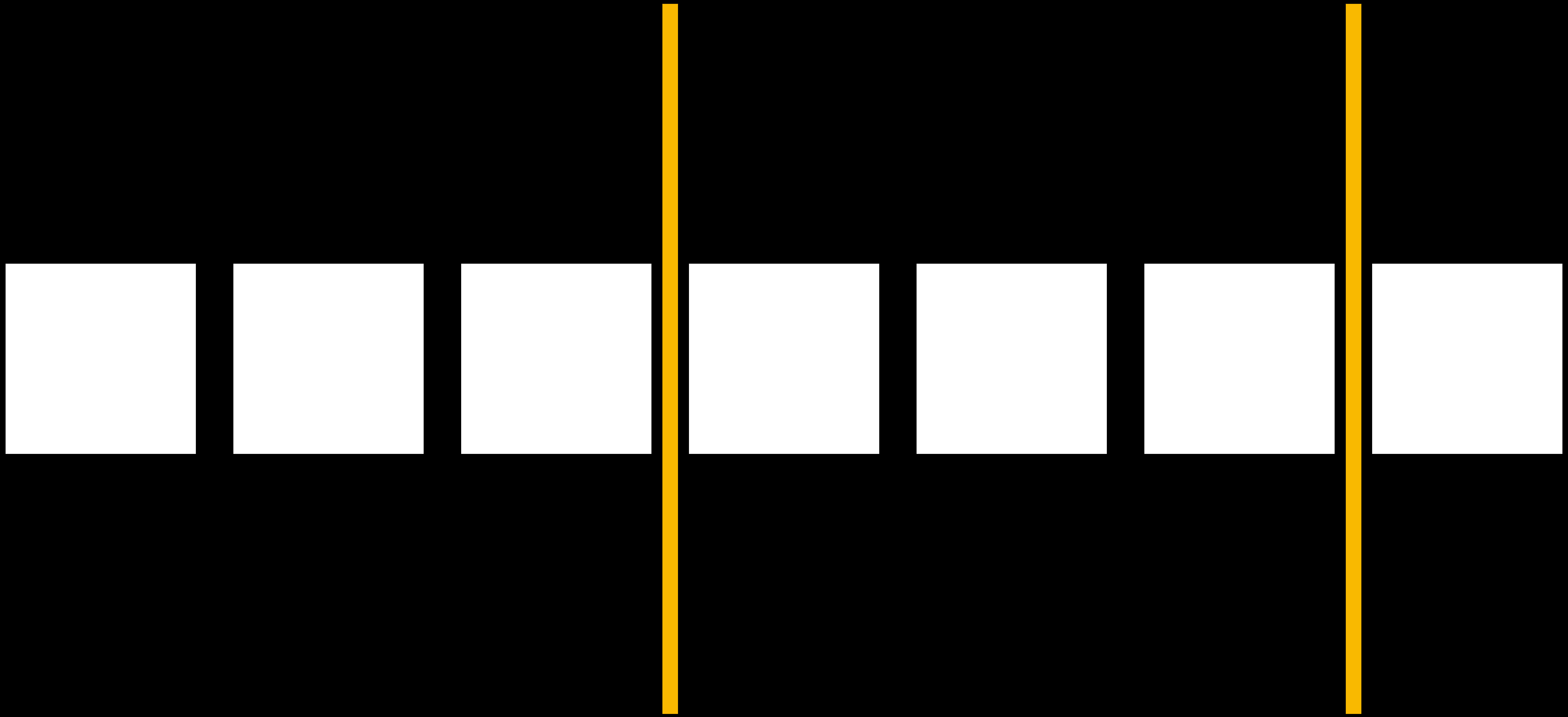


```swift
//  adjacentPairs
let adjacentPairsMember = Array(members.adjacentPairs().map{ [$0.name, $1.name] })
print(adjacentPairsMember)
```

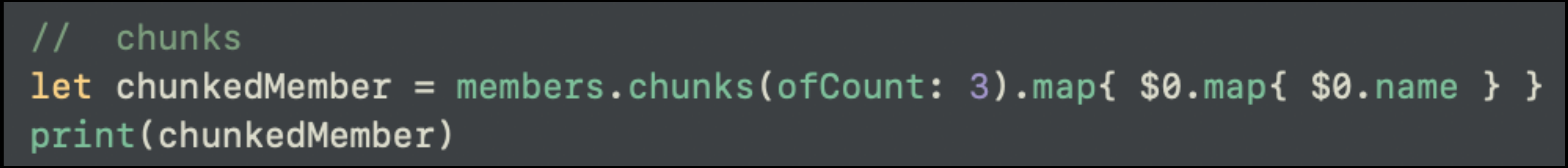


```
[["호종이", "아보"], ["아보", "에이브리"], ["에이브리", "우디"], ["우디", "에셔"], ["에셔", "찰리"], ["찰리", "놔스닥"], ["놔스닥",
    "포딩"], ["포딩", "택"], ["택", "유스"]]
```

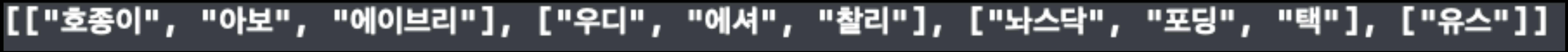
Chunks(ofCount: Int)



Chunks(ofCount: Int)

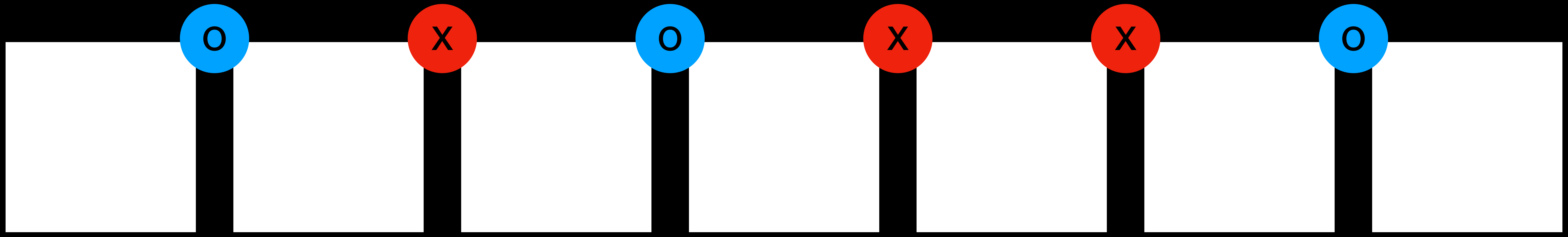


```swift
//  chunks
let chunkedMember = members.chunks(ofCount: 3).map{ $0.map{ $0.name } }
print(chunkedMember)
```

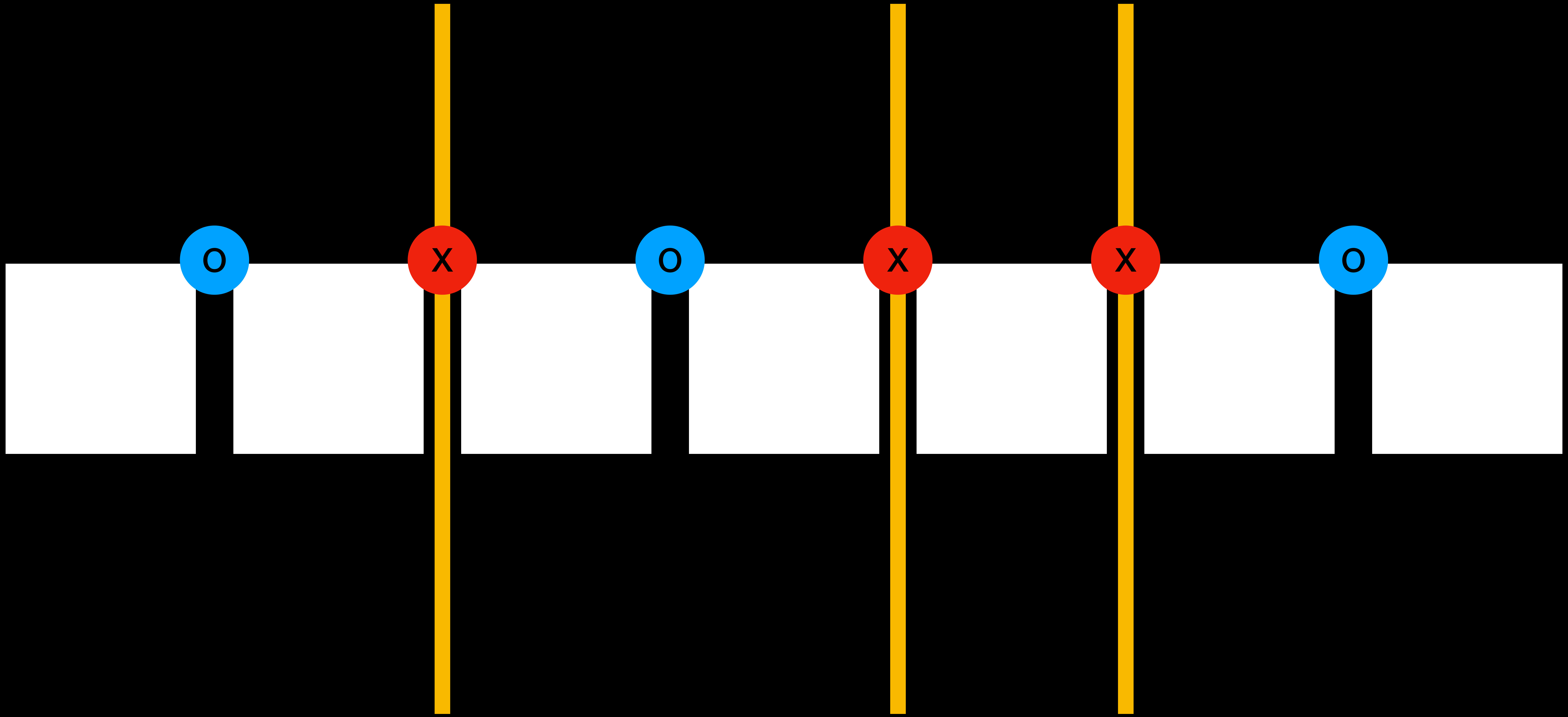


```
[["호종이", "아보", "에이브리"], ["우디", "에셔", "찰리"], ["놔스닥", "포딩", "택"], ["유스"]]
```

Chunked



Chunked



Chunked

```swift
//  chunked(on:)
let chunkedOnMember = members.chunked{ $0.name < $1.name }.map{ $0.map{ $0.name } }
print(chunkedOnMember)
```

```
[["호종이"], ["아보", "에이브리", "우디"], ["에셔", "찰리"], ["놔스닥", "포딩"], ["택"], ["유스"]]
```

Swift Algorithms / adjacentPairs() / chain(_:_:) / chunked(by:) / chunked(on:) chunks(ofCount:) / combinations() / combinations(ofCount:) / compacted() cycled() / cycled(times:) / firstNonNil(_:) / interspersed(with:) / joined(by:) max(count:) / max(count:sortedBy:) / min(count:) / min(count:sortedBy:) minAndMax() / minAndMax(by:) / partitioningIndex(where:) / permutations() permutations(ofCount:) / product(_:_:) / randomSample(count:) randomStableSample(count:) / reductions(_:) / reductions(_:_:) reductions(into:_:) / rotate(toStartAt:) / rotate(subrange:toStartAt:) split(whereSeparator:) / split(separator:) / stablePartition(by:) stablePartition(subrange:by:) / striding(by:) / suffix(while:) / trimming(while:) trimmingPrefix(while:) / trimmingSuffix(while:) / uniquePermutations() uniquePermutations(ofCount:) / uniqued() / uniqued(on:) / windows(ofCount:)