#WWDC18

Using Collections Effectively

Session 229

Swift에서 컬렉션을 효율적으로 사용하는 법

Michael LeHew, Foundation

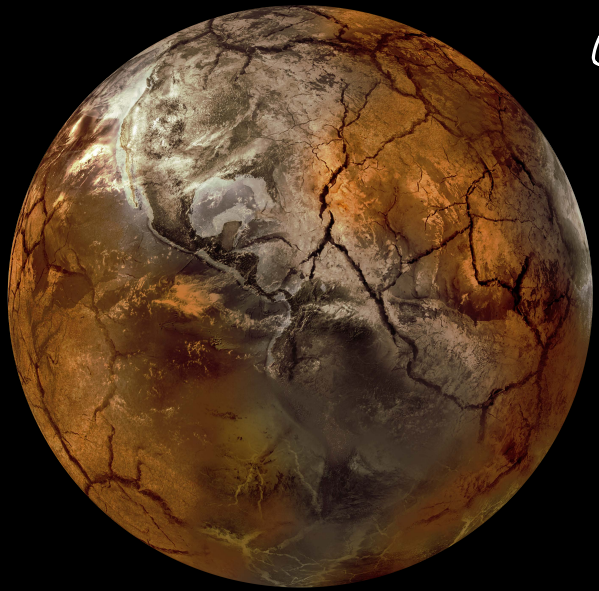
Fundamentals

Indices and Slices

Lazy

Mutability and Multithreading

Foundation and Bridging



Collection이 없는 세상

// A World Without Arrays 배열이 없다면

```
// A World Without Arrays

let bear1 = "Grizzly"
```

```
// A World Without Arrays

let bear1 = "Grizzly"
let bear2 = "Panda"
let bear3 = "Polar"
let bear4 = "Spectacled"
```

```swift
// A World Without Arrays

let bear1 = "Grizzly"
let bear2 = "Panda"
let bear3 = "Polar"
let bear4 = "Spectacled"

print("\(bear1) bear") // Grizzly bear
print("\(bear2) bear") // Panda bear
print("\(bear3) bear") // Polar bear
print("\(bear4) bear") // Spectacled bear
```

```swift
// A World Without Dictionaries

func habitat(for bear: String) -> String? {









}
```

```
// A World Without Dictionaries

func habitat(for bear: String) -> String? {
    if bear == "Polar" {
        return "Arctic"
    } else if bear == "Grizzly" {
        return "Forest"
    } else if bear == "Brown" {
        return "Forest"
    } else if /* all the other bears */
    ...
    return nil
}
```

```
// A World Without Dictionaries

func habitat(for bear: String) -> String? {
   if bear == "Polar" {
      return "Arctic"
   } else if bear == "Grizzly" {
      return "Forest"
   } else if bear == "Brown" {
      return "Forest"
   } else if /* all the other cases */
   ...
   return nil
}

let bear = "Polar"
print("\(bear) bears live in the \(habitat(for: bear).")
```

dictionary

```
// Our World Has Collections
```

다행

```
let bear = ["Grizzly", "Panda", "Polar", "Spectacled"]
let habitats = ["Grizzly": "Forest", "Polar": "Arctic"]
```

```
// Our World Has Collections

let bear = ["Grizzly", "Panda", "Polar", "Spectacled"]
let habitats = ["Grizzly": "Forest", "Polar": "Arctic"]

for bear in bears {
  print("\(bear) Bear")
}
```

```swift
// Our World Has Collections

let bear = ["Grizzly", "Panda", "Polar", "Spectacled"]
let habitats = ["Grizzly": "Forest", "Polar": "Arctic"]

for bear in bears {
  print("\(bear) Bear")
}

let bear = bears[2]
let habitat = habitats[bear] ?? ""
print("\(bear) bears live in the \(habitat)")
```

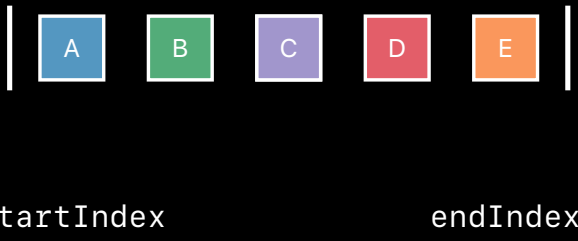
protocol Collection

Collections Store Elements

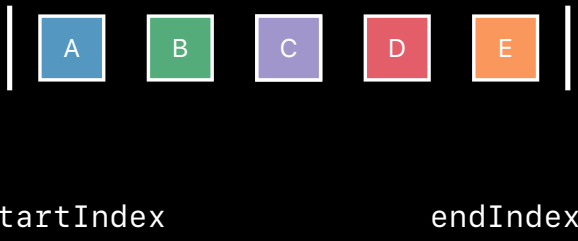
Collections Store Elements

Array
tree
:
무엇이든 동일하게 startIndex 접근 가능

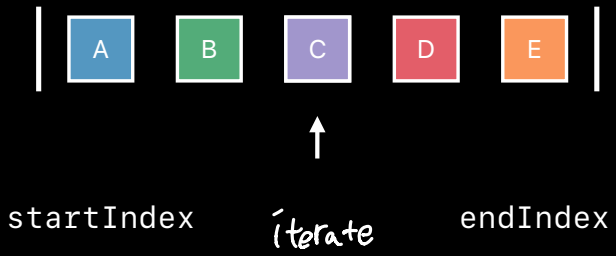
Collections Store Elements



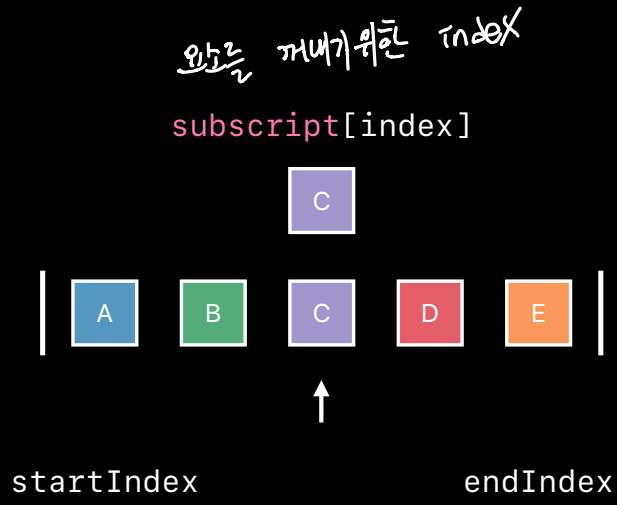
Collections Store Elements



Collections Store Elements



Collections Store Elements



```swift
// Declaration of Collection

protocol Collection : Sequence {
   associatedtype Element
   associatedtype Index : Comparable

   subscript(position: Index) -> Element { get }

   var startIndex: Index { get }
   var endIndex: Index { get }

   func index(after i: Index) -> Index
}
```

Collection은 Sequence를 상속

```swift
// Declaration of Collection

protocol Collection : Sequence {
    associatedtype Element
    associatedtype Index : Comparable

    subscript(position: Index) -> Element { get }

    var startIndex: Index { get }
    var endIndex: Index { get }

    func index(after i: Index) -> Index
}
```

element , index 의 타입이 필요

```swift
// Declaration of Collection

protocol Collection : Sequence {
    associatedtype Element
    associatedtype Index : Comparable

    subscript(position: Index) -> Element { get }

    var startIndex: Index { get }
    var endIndex: Index { get }

    func index(after i: Index) -> Index
}
```

비교가능 해야 함

```swift
// Declaration of Collection

protocol Collection : Sequence {
    associatedtype Element
    associatedtype Index : Comparable

    subscript(position: Index) -> Element { get }

    var startIndex: Index { get }
    var endIndex: Index { get }

    func index(after i: Index) -> Index
}
```

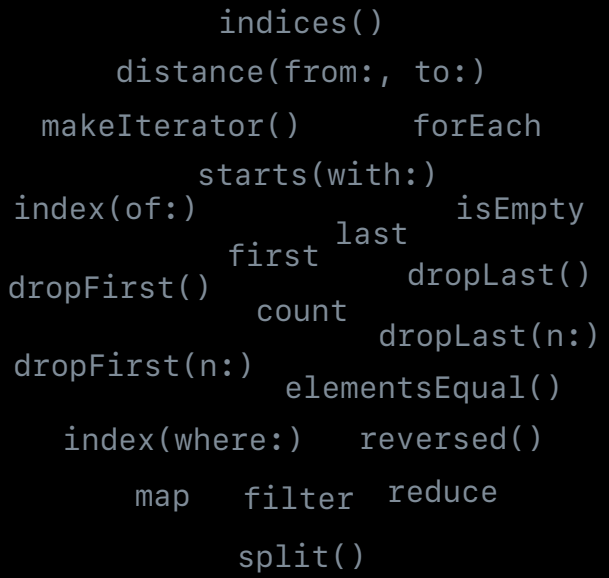
요소 하나를 획득

```swift
// Declaration of Collection

protocol Collection : Sequence {
   associatedtype Element
   associatedtype Index : Comparable

   subscript(position: Index) -> Element { get }

   var startIndex: Index { get }
   var endIndex: Index { get }

   func index(after i: Index) -> Index
}
```

```swift
// Declaration of Collection

protocol Collection : Sequence {
   associatedtype Element
   associatedtype Index : Comparable

   subscript(position: Index) -> Element { get }

   var startIndex: Index { get }
   var endIndex: Index { get }

   func index(after i: Index) -> Index
}
```

after

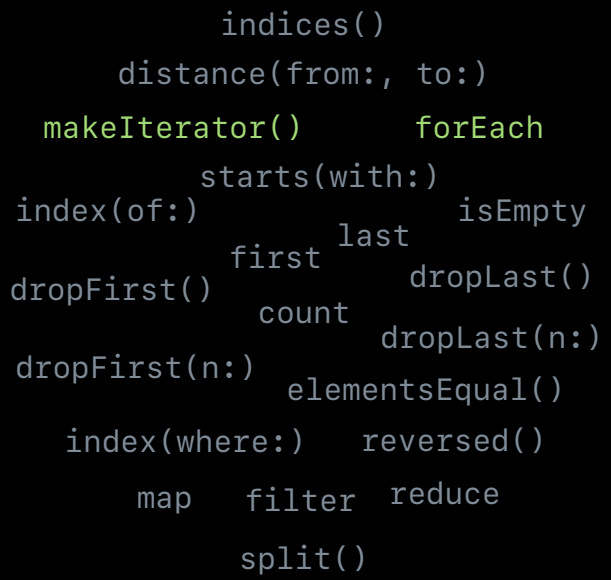
Protocol Extensions



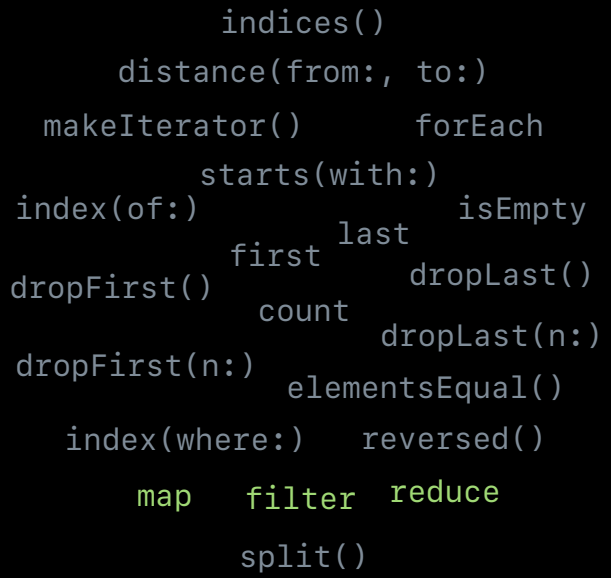
Protocol Extensions

첫번째, 마지막 ...

Protocol Extensions



Protocol Extensions

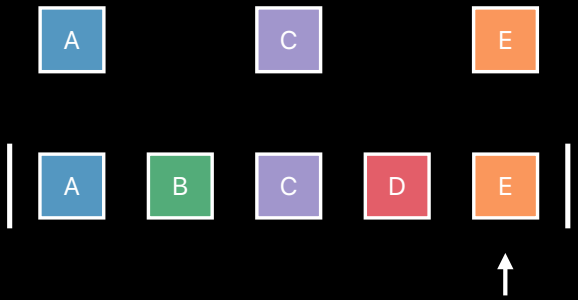


Every Other Element

이미 모든 element를 순회하는 기능은 있다.

Every Other Element

하나씩 건너뛰고 순회하고 싶다면?



```
extension Collection {
    func everyOther(_ body: (Element) -> Void) {
        let start = self.startIndex
        let end = self.endIndex

        var iter = start
        while iter != end {
            body(self[iter])
            let next = index(after: iter)
            if next == end { break }
            iter = index(after: next)
        }
    }
}

(1...10).everyOther { print($0) }
```

```
extension Collection {
   func everyOther(_ body: (Element) -> Void) {
      let start = self.startIndex
      let end = self.endIndex

      var iter = start
      while iter != end {
         body(self[iter])
         let next = index(after: iter)
         if next == end { break }
         iter = index(after: next)
      }
   }
}

(1...10).everyOther { print($0) }
```

```
extension Collection {
   func everyOther(_ body: (Element) -> Void) {
      let start = self.startIndex
      let end = self.endIndex

      var iter = start
      while iter != end {
         body(self[iter])
         let next = index(after: iter)
         if next == end { break }
         iter = index(after: next)
      }
   }
}

(1...10).everyOther { print($0) }
```

```
extension Collection {
   func everyOther(_ body: (Element) -> Void) {
      let start = self.startIndex
      let end = self.endIndex

      var iter = start
      while iter != end {
         body(self[iter])
         let next = index(after: iter)
         if next == end { break }
         iter = index(after: next)
      }
   }
}

(1...10).everyOther { print($0) }
```

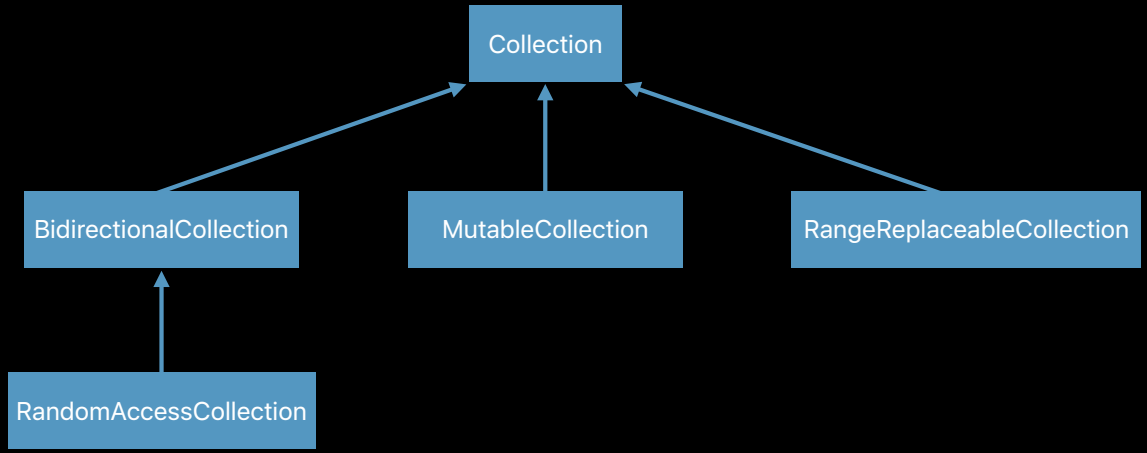
```
extension Collection {
   func everyOther(_ body: (Element) -> Void) {
      let start = self.startIndex
      let end = self.endIndex

      var iter = start
      while iter != end {
         body(self[iter])
         let next = index(after: iter)
         if next == end { break }
         iter = index(after: next)
      }
   }
}

(1...10).everyOther { print($0) }
```

```
extension Collection {
   func everyOther(_ body: (Element) -> Void) {
      let start = self.startIndex
      let end = self.endIndex

      var iter = start
      while iter != end {
         body(self[iter])
         let next = index(after: iter)
         if next == end { break }
         iter = index(after: next)
      }
   }
}

(1...10).everyOther { print($0) }
```

```
extension Collection {
   func everyOther(_ body: (Element) -> Void) {
      let start = self.startIndex
      let end = self.endIndex

      var iter = start
      while iter != end {
         body(self[iter])
         let next = index(after: iter)
         if next == end { break }
         iter = index(after: next)
      }
   }
}

(1...10).everyOther { print($0) } // 1, 3, 5, 7, 9
```

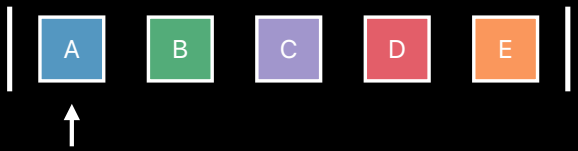
Collections Protocol Hierarchy

Collections Protocol Hierarchy 콜렉션 계층



Collections

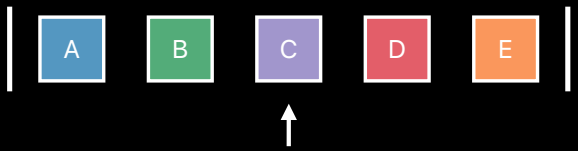
```
func index(after: Self.Index) -> Self.Index
```



Collections

순방향

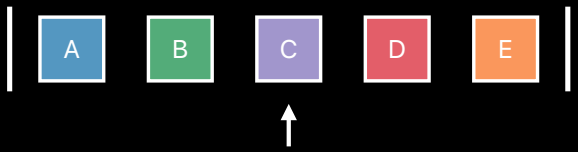
```
func index(after: Self.Index) -> Self.Index
```



Bidirectional Collections

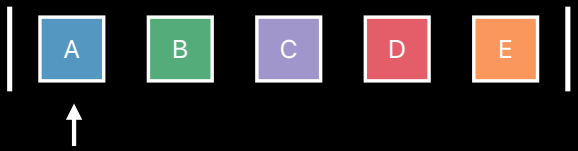
역방향도 가능

```
func index(before: Self.Index) -> Self.Index
```



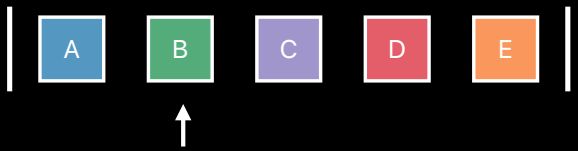
Bidirectional Collections

```
func index(before: Self.Index) -> Self.Index
```



Bidirectional Collections

```
func index(before: Self.Index) -> Self.Index
```



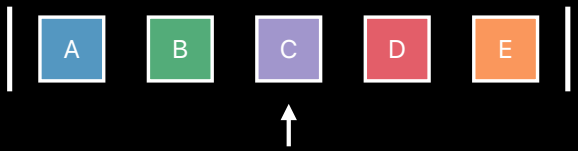
Random Access Collections

어느위치에 접근하든 일정 시간 소요

```
// constant time
func index(_ idx: Index, offsetBy n: Int) -> Index
func distance(from start: Index, to end: Index) -> Int
```

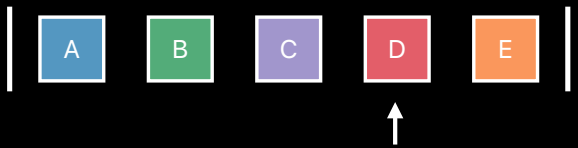
Random Access Collections

```
// constant time
func index(_ idx: Index, offsetBy n: Int) -> Index
func distance(from start: Index, to end: Index) -> Int
```



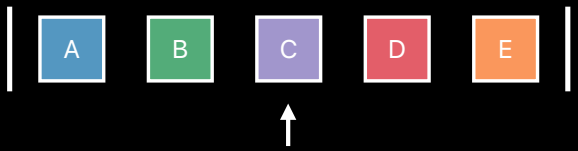
Random Access Collections

```
// constant time
func index(_ idx: Index, offsetBy n: Int) -> Index
func distance(from start: Index, to end: Index) -> Int
```



Random Access Collections

```
// constant time
func index(_ idx: Index, offsetBy n: Int) -> Index
func distance(from start: Index, to end: Index) -> Int
```



Collections In Swift

`Array` `Set` `Dictionary`

Collections In Swift

Array Set Dictionary

Data Range String

Collections In Swift

`Array` `Set` `Dictionary`

`IndexPath` `IndexSet`

`Data` `Range` `String`

Collections In Swift

Indices

Each collection defines its own index 각 Collection은 고유의 index 사용

Must be `Comparable` index는 Comparable

Think of these as opaque Int라고 생각하면 말것

What is the first element of an Array?

array는 위에 Int로 first와 접근 가능

```
array[0]
```

What is the first element of a Set?

```
set[0]
```

가능한가?

컴파일 에러

```
set[0]
```

Cannot subscript a value of type 'Set' with an index of type 'Int'

```
set[set.startIndex]
```

가능

```
set[set.startIndex]
array[array.startIndex]
```

모든 Collection에 있음

```
set[set.startIndex]
array[array.startIndex]
```

Fatal error: Index out of range

empty collection old error

first로 접근하면 optional

```
array.first
set.first
```

What is the second element of a Collection?

Find the Second Element of a `Collection`

```
extension Collection {
   var second: Element? {

   }
}
```

Find the Second Element of a Collection

```
extension Collection {
   var second: Element? {
      return self[1]
   }
}
```

Find the Second Element of a Collection

왜 Int로 접근은 안됨

```
extension Collection {
   var second: Element? {
      return self[1]
   }
}
```

Cannot subscript a value of type 'Collection' with an index of type 'Int'

Find the Second Element of a Collection

```
extension Collection {
   var second: Element? {
      return self[self.startIndex + 1]
   }
}
```

Find the Second Element of a Collection

```
extension Collection {
    var second: Element? {
        return self[self.startIndex + 1]
    }
}
```

```
Binary operator '+' cannot be applied to
    operands of type 'Index' and 'Int'
```

+1도 불가능

Find the Second Element of a Collection

```
extension Collection {
   var second: Element? {
      // Is the collection empty?

      // Get the second index

      // Is that index valid?

      // Return the second element

   }
}
```

Find the Second Element of a Collection

```
extension Collection {
   var second: Element? {
      // Is the collection empty?

      // Get the second index

      // Is that index valid?

      // Return the second element

   }
}
```

startIndex
endIndex

Find the Second Element of a Collection

```
extension Collection {
    var second: Element? {
        // Is the collection empty?
        guard self.startIndex != self.endIndex else { return nil }
        // Get the second index

        // Is that index valid?

        // Return the second element

    }
}
```

가장 먼저 empty 체크

Find the Second Element of a Collection

```
extension Collection {
    var second: Element? {
        // Is the collection empty?
        guard self.startIndex != self.endIndex else { return nil }
        // Get the second index

        // Is that index valid?

        // Return the second element

    }
}
```

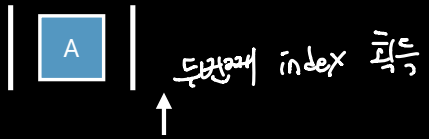
최소 하나는 있다고 추측 가능

Find the Second Element of a Collection

```
extension Collection {
   var second: Element? {
      // Is the collection empty?
      guard self.startIndex != self.endIndex else { return nil }
      // Get the second index
      let index = self.index(after: self.startIndex)
      // Is that index valid?

      // Return the second element

   }
}
```

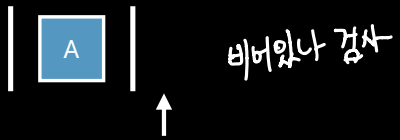
Find the Second Element of a Collection

```
extension Collection {
   var second: Element? {
      // Is the collection empty?
      guard self.startIndex != self.endIndex else { return nil }
      // Get the second index
      let index = self.index(after: self.startIndex)
      // Is that index valid?

      // Return the second element

   }
}
```



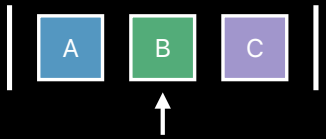
Find the Second Element of a Collection

```
extension Collection {
    var second: Element? {
        // Is the collection empty?
        guard self.startIndex != self.endIndex else { return nil }
        // Get the second index
        let index = self.index(after: self.startIndex)
        // Is that index valid?
        guard index != self.endIndex else { return nil }
        // Return the second element

    }
}
```



Find the Second Element of a Collection

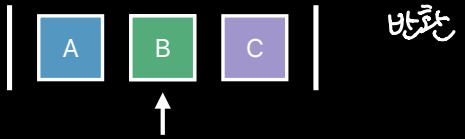
```
extension Collection {
    var second: Element? {
        // Is the collection empty?
        guard self.startIndex != self.endIndex else { return nil }
        // Get the second index
        let index = self.index(after: self.startIndex)
        // Is that index valid?
        guard index != self.endIndex else { return nil }
        // Return the second element

    }
}
```



두번째도 있다면

Find the Second Element of a Collection

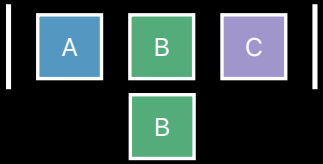
```
extension Collection {
    var second: Element? {
        // Is the collection empty?
        guard self.startIndex != self.endIndex else { return nil }
        // Get the second index
        let index = self.index(after: self.startIndex)
        // Is that index valid?
        guard index != self.endIndex else { return nil }
        // Return the second element
        return self[index]
    }
}
```



반환

Find the Second Element of a Collection

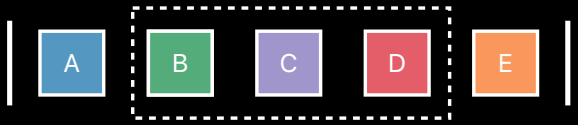
```
extension Collection {
    var second: Element? {
        // Is the collection empty?
        guard self.startIndex != self.endIndex else { return nil }
        // Get the second index
        let index = self.index(after: self.startIndex)
        // Is that index valid?
        guard index != self.endIndex else { return nil }
        // Return the second element
        return self[index]
    }
}
```



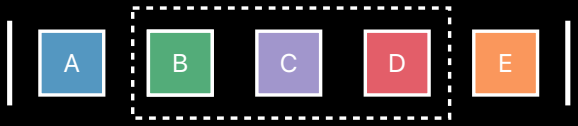
Forming a Slice

Slice는 Collection의 일부

Forming a Slice

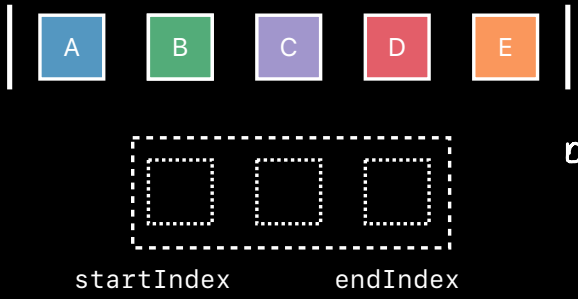


Forming a Slice



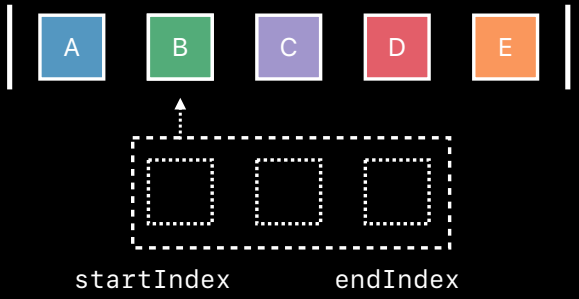
`startIndex` `endIndex`

Forming a Slice

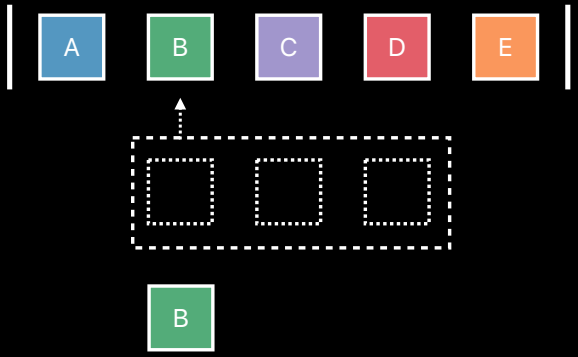


original과 별개로 존재

Forming a Slice



Forming a Slice



원본에 접근하는 것이다.
index는 공유 가능하다.

```
subscript[index]
```

```swift
// Slice Share Indices with Original Collection

let array = [1, 2, 3, 4, 5]

let subarray = array.dropFirst()
let secondIndex = array.index(after: array.startIndex)

print(secondIndex == subarray.startIndex)
```

```swift
// Slice Share Indices with Original Collection

let array = [1, 2, 3, 4, 5]

let subarray = array.dropFirst()
let secondIndex = array.index(after: array.startIndex)

print(secondIndex == subarray.startIndex)
```

```swift
// Slice Share Indices with Original Collection

let array = [1, 2, 3, 4, 5]

let subarray = array.dropFirst()
let secondIndex = array.index(after: array.startIndex)

print(secondIndex == subarray.startIndex)
```

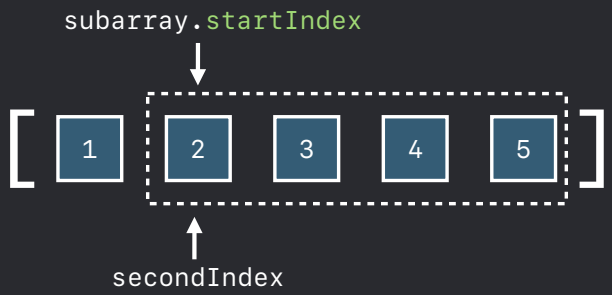
첫번째 값만 제외

```swift
// Slice Share Indices with Original Collection

let array = [1, 2, 3, 4, 5]

let subarray = array.dropFirst()
let secondIndex = array.index(after: array.startIndex)

print(secondIndex == subarray.startIndex)
```

```swift
// Slice Share Indices with Original Collection

let array = [1, 2, 3, 4, 5]

let subarray = array.dropFirst()
let secondIndex = array.index(after: array.startIndex)

print(secondIndex == subarray.startIndex) // true
```

Index 는 공유 된다. 확인가능

덕분에 간단하게 두번째 Index 접근



Find the Second Element of a Collection

아까 만든 두번째 index 코드

```
var second: Element? {
   // Is the collection empty?
   guard self.startIndex != self.endIndex else { return nil }
   // Get the second index
   let index = self.index(after: self.startIndex)
   // Is the second index valid?
   guard index != self.endIndex else { return nil }
   // Return the second element
   return self[index]
}
```

Find the Second Element of a Collection

```
var second: Element? {
   return self.dropFirst().first
}
```

한줄 가능

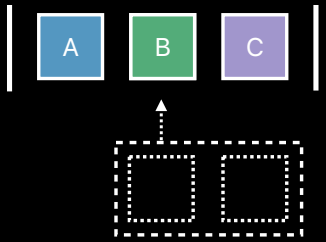
Find the Second Element of a Collection

```
var second: Element? {
   return self.dropFirst().first
}
```

그림처럼 동작함

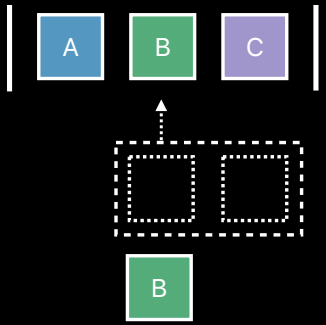
Find the Second Element of a Collection

```
var second: Element? {
   return self.dropFirst().first
}
```



Find the Second Element of a Collection

```
var second: Element? {
   return self.dropFirst().first
}
```



Slices

Produce `Collection`-like peers of original collection

Slices

Produce `Collection`-like peers of original collection

`Array` ⇢ `ArraySlice`

Slices

Produce `Collection`-like peers of original collection

`Array` ⇢ `ArraySlice`

`String` ⇢ `Substring`

Slices

Produce `Collection`-like peers of original collection

`Array`	⟶	`ArraySlice`
`String`	⟶	`Substring`
`Set`	⟶	`Slice<Set>`

Slices

Produce `Collection`-like peers of original collection

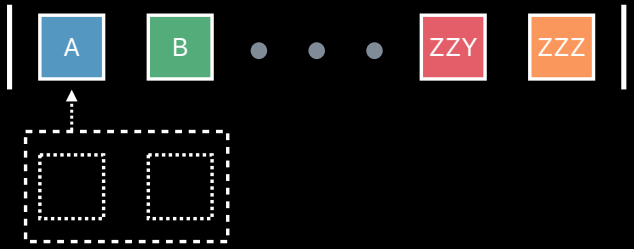
Slice 타입들

`Array`	-->	`ArraySlice`
`String`	-->	`Substring`
`Set`	-->	`Slice<Set>`
`Data`	-->	`Data`
`Range`	-->	`Range`

Slices Keep Underlying Storage Alive

엄청 큰 콜렉션에서
일부에 접근 할 수 있음

Slices Keep Underlying Storage Alive



```
// Slicing Keeps Underlying Storage

extension Array {
    var firstHalf: ArraySlice<Element> {
        return self.dropLast(self.count / 2)
    }
}
```

```
// Slicing Keeps Underlying Storage

extension Array {
   var firstHalf: ArraySlice<Element> {
      return self.dropLast(self.count / 2)
   }
}

var array = [1, 2, 3, 4, 5, 6, 7, 8]
var firstHalf = array.firstHalf
array = []
```

```swift
// Slicing Keeps Underlying Storage

extension Array {
   var firstHalf: ArraySlice<Element> {
      return self.dropLast(self.count / 2)
   }
}

var array = [1, 2, 3, 4, 5, 6, 7, 8]
var firstHalf = array.firstHalf // [1, 2, 3, 4]
array = []
```

(게으른) 복사본

```
// Slicing Keeps Underlying Storage

extension Array {
   var firstHalf: ArraySlice<Element> {
      return self.dropLast(self.count / 2)
   }
}

var array = [1, 2, 3, 4, 5, 6, 7, 8]
var firstHalf = array.firstHalf // [1, 2, 3, 4]
array = []
```

```swift
// Slicing Keeps Underlying Storage

extension Array {
   var firstHalf: ArraySlice<Element> {
      return self.dropLast(self.count / 2)
   }
}

var array = [1, 2, 3, 4, 5, 6, 7, 8]
var firstHalf = array.firstHalf // [1, 2, 3, 4]
array = []

print(firstHalf.first!) // 1
```

Slice로 접근가능

```swift
// Slicing Keeps Underlying Storage

extension Array {
   var firstHalf: ArraySlice<Element> {
      return self.dropLast(self.count / 2)
   }
}

var array = [1, 2, 3, 4, 5, 6, 7, 8]
var firstHalf = array.firstHalf // [1, 2, 3, 4]
array = []

print(firstHalf.first!) // 1

let copy = Array(firstHalf) // [1, 2, 3, 4]
firstHalf = []
print(copy.first!)
```

복사본

```swift
// Slicing Keeps Underlying Storage

extension Array {
   var firstHalf: ArraySlice<Element> {
      return self.dropLast(self.count / 2)
   }
}

var array = [1, 2, 3, 4, 5, 6, 7, 8]
var firstHalf = array.firstHalf // [1, 2, 3, 4]
array = []

print(firstHalf.first!) // 1

let copy = Array(firstHalf) // [1, 2, 3, 4]
firstHalf = []
print(copy.first!)
```

```swift
// Slicing Keeps Underlying Storage

extension Array {
   var firstHalf: ArraySlice<Element> {
      return self.dropLast(self.count / 2)
   }
}

var array = [1, 2, 3, 4, 5, 6, 7, 8]
var firstHalf = array.firstHalf // [1, 2, 3, 4]
array = []

print(firstHalf.first!) // 1

let copy = Array(firstHalf) // [1, 2, 3, 4]
firstHalf = []
print(copy.first!) // 1
```

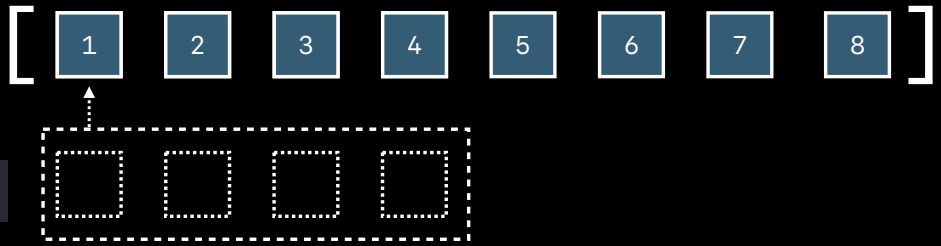
Copying a Slice

```
var array = [...]
```

Copying a Slice

```
var array = [...]
```

```
var firstHalf = array.firstHalf
```



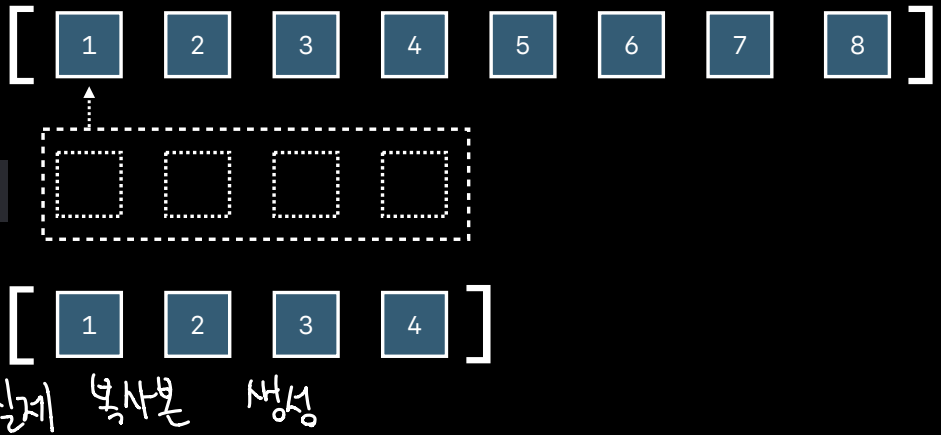
가리키기만 함

Copying a Slice

```
var array = [...]
```

```
var firstHalf = array.firstHalf
```

```
let copy = Array(firstHalf)
```



Copying a Slice

```
var array = []
```

비었거만

원본유지


```
var firstHalf = array.firstHalf
```

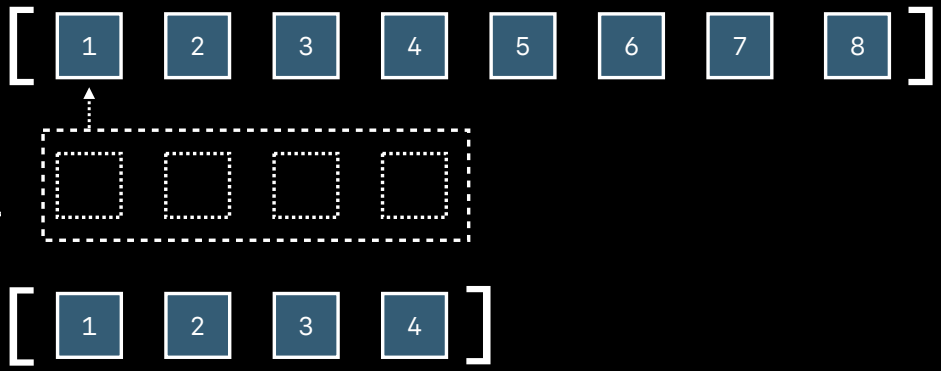
```
let copy = Array(firstHalf)
```

Copying a Slice

```
var array = []
```

```
var firstHalf = []
```

```
let copy = Array(firstHalf)
```



Copying a Slice

```
var array = []
```

```
var firstHalf = []
```

```
let copy = Array(firstHalf)
```

Slice를 비운 후에야 원본은 제거

깊은 복사와 차이가 있음

Eager Functions

Swift의 function은 기본적으로 eager copy

```
let items = (1...4000).map { $0 * 2 }.filter { $0 < 10 }
```

Eager Functions

```
let items = (1...4000).map { $0 * 2 }.filter { $0 < 10 }
```

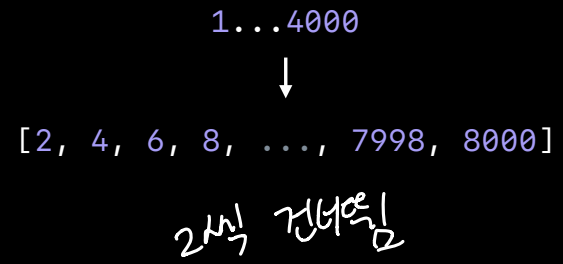
1...4000

시작과 끝으로

범위 나타냄

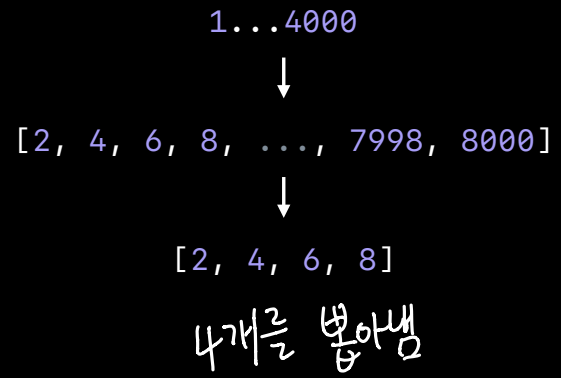
Eager Functions

```
let items = (1...4000).map { $0 * 2 }.filter { $0 < 10 }
```



Eager Functions

```
let items = (1...4000).map { $0 * 2 }.filter { $0 < 10 }
```



Eager Functions

```
let items = (1...4000).map { $0 * 2 }.filter { $0 < 10 }
```

마지막 4개만 있으면 되는데

뒤가 낭비스럽다.

(중간 계산)

```
[2, 4, 6, 8]
```

Lazy Functions

case와 대신 lazy가 있다

```
let items = (1...4000).lazy.map { $0 * 2 }.filter { $0 < 10 }
```

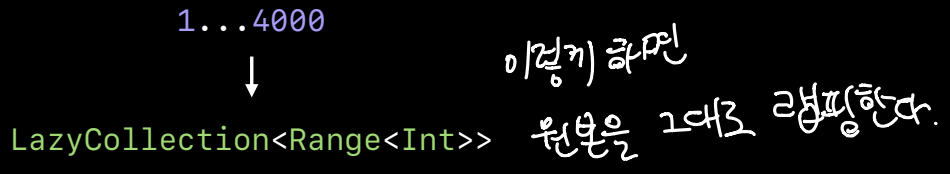
Lazy Functions

```
let items = (1...4000).lazy.map { $0 * 2 }.filter { $0 < 10 }
```

1...4000

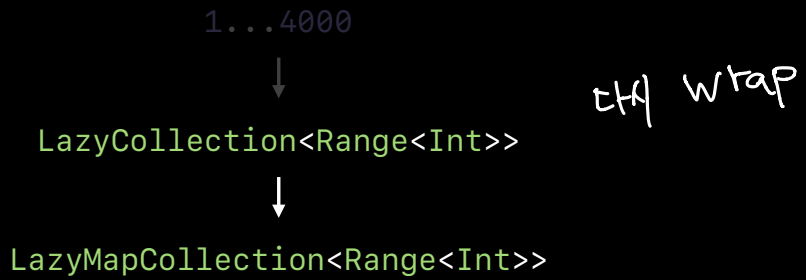
Lazy Functions

```
let items = (1...4000).lazy.map { $0 * 2 }.filter { $0 < 10 }
```



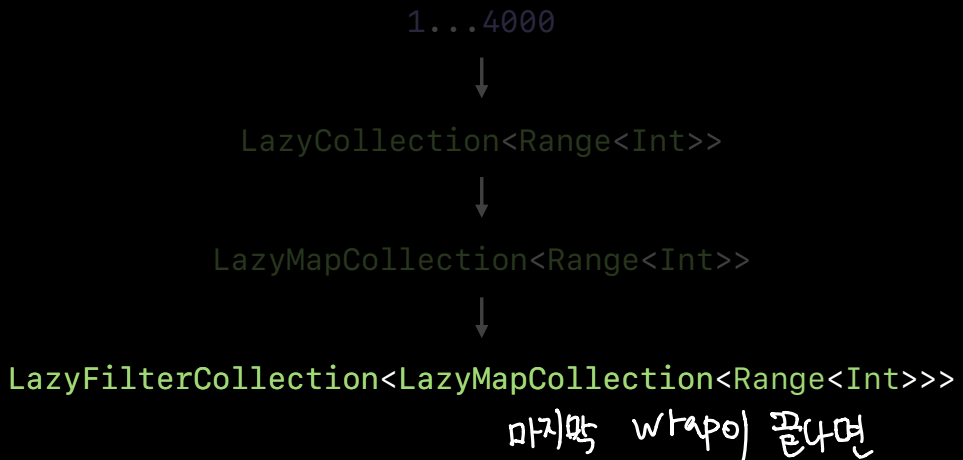
Lazy Functions

```
let items = (1...4000).lazy.map { $0 * 2 }.filter { $0 < 10 }
```



Lazy Functions

```
let items = (1...4000).lazy.map { $0 * 2 }.filter { $0 < 10 }
```



Lazy Functions

```
let items = (1...4000).lazy.map { $0 * 2 }.filter { $0 < 10 }
```

LazyFilterCollection LazyMapCollection 1...4000

Lazy Functions

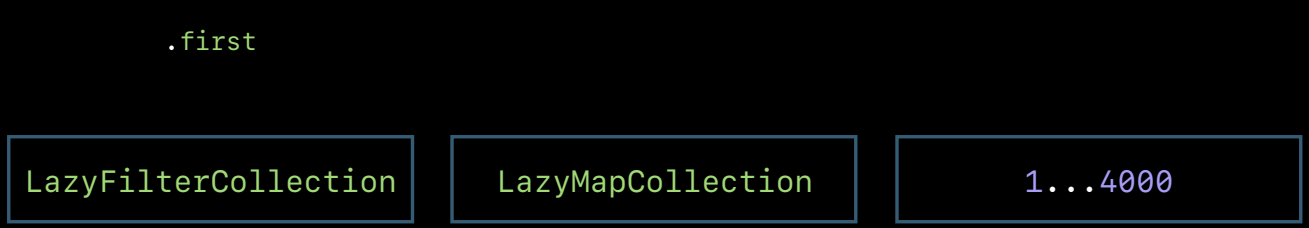
```
let items = (1...4000).lazy.map { $0 * 2 }.filter { $0 < 10 }
```

Lazy Functions

```
let items = (1...4000).lazy.map { $0 * 2 }.filter { $0 < 10 }
items.first
```

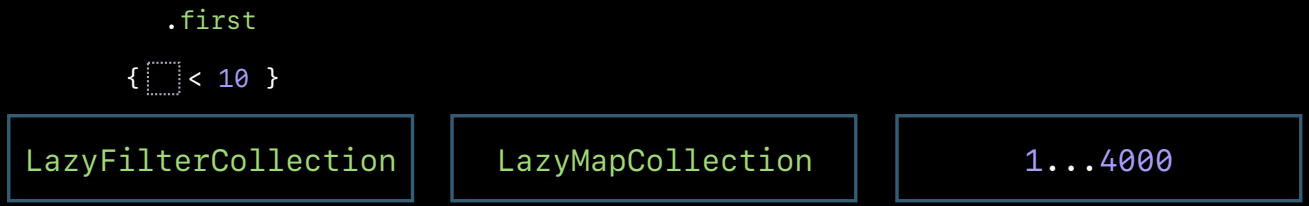
Lazy Functions

```
let items = (1...4000).lazy.map { $0 * 2 }.filter { $0 < 10 }
items.first
```



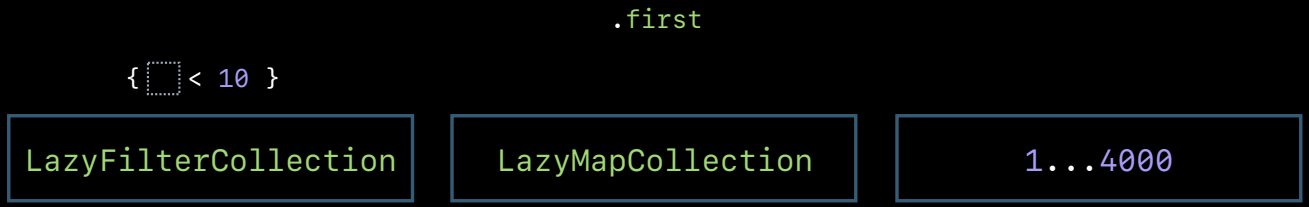
Lazy Functions

```
let items = (1...4000).lazy.map { $0 * 2 }.filter { $0 < 10 }
items.first
```



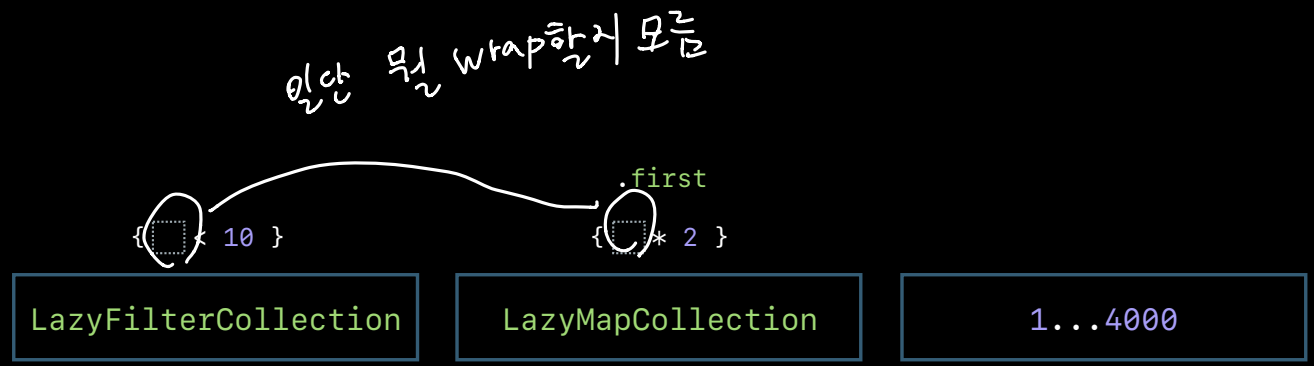
Lazy Functions

```
let items = (1...4000).lazy.map { $0 * 2 }.filter { $0 < 10 }
items.first
```



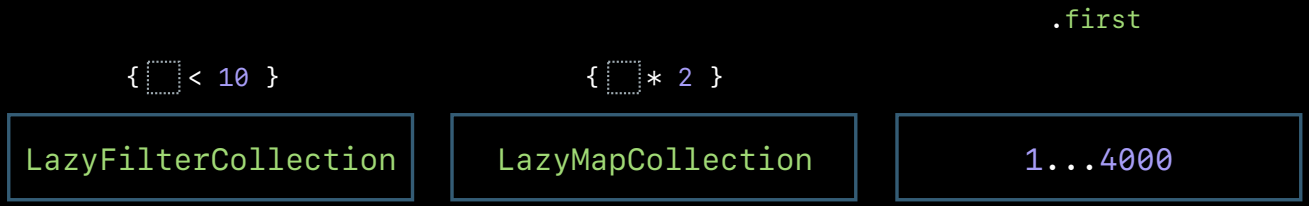
Lazy Functions

```
let items = (1...4000).lazy.map { $0 * 2 }.filter { $0 < 10 }
items.first
```



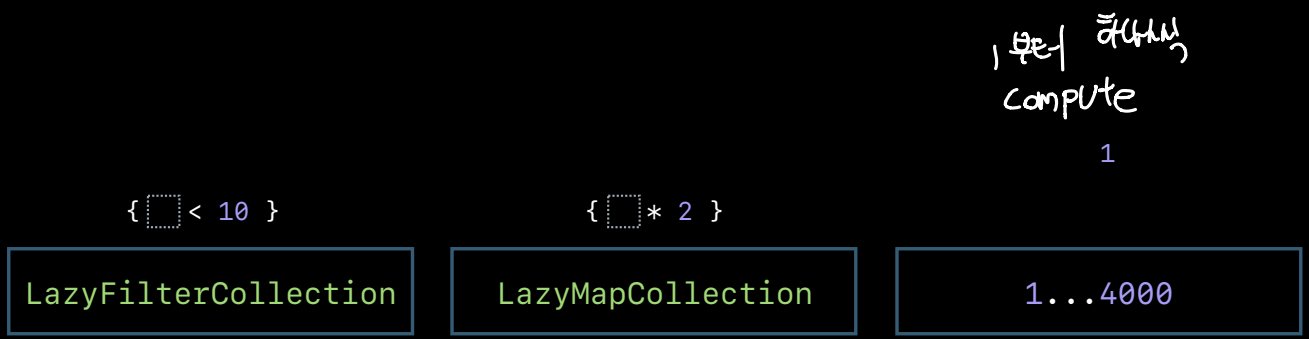
Lazy Functions

```
let items = (1...4000).lazy.map { $0 * 2 }.filter { $0 < 10 }
items.first
```



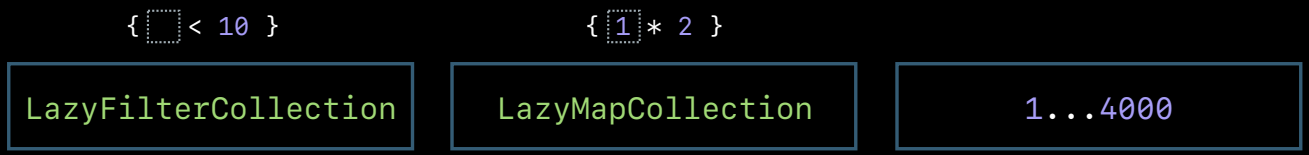
Lazy Functions

```
let items = (1...4000).lazy.map { $0 * 2 }.filter { $0 < 10 }
items.first
```



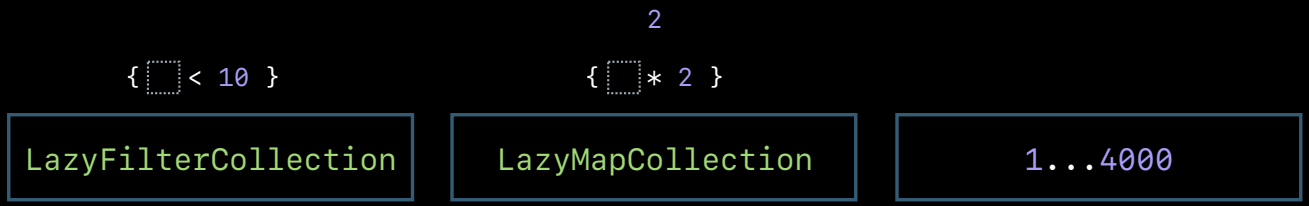
Lazy Functions

```
let items = (1...4000).lazy.map { $0 * 2 }.filter { $0 < 10 }
items.first
```



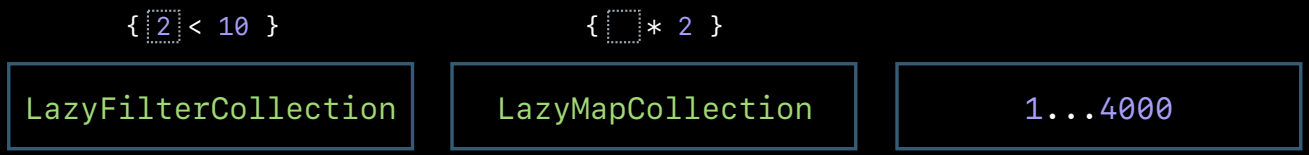
Lazy Functions

```
let items = (1...4000).lazy.map { $0 * 2 }.filter { $0 < 10 }
items.first
```



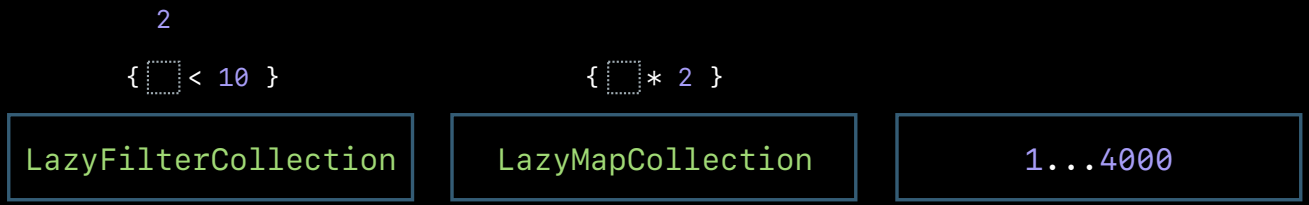
Lazy Functions

```
let items = (1...4000).lazy.map { $0 * 2 }.filter { $0 < 10 }
items.first
```



Lazy Functions

```
let items = (1...4000).lazy.map { $0 * 2 }.filter { $0 < 10 }
items.first
```



Lazy Functions

```
let items = (1...4000).lazy.map { $0 * 2 }.filter { $0 < 10 }
items.first    // 2
```

// Lazy Defers Computation

문제점 상황

```
let bears = ["Grizzly", "Panda", "Spectacled", "Gummy Bears", "Chicago"]
```

```
// Lazy Defers Computation

let bears = ["Grizzly", "Panda", "Spectacled", "Gummy Bears", "Chicago"]
```

```swift
// Lazy Defers Computation

let bears = ["Grizzly", "Panda", "Spectacled", "Gummy Bears", "Chicago"]

let redundantBears = bears.lazy.filter {
   print("Checking '\($0)'")
   return $0.contains("Bear")
}

print(redundantBears.first!)
```

lazy .filter 추가

```
// Lazy Defers Computation

let bears = ["Grizzly", "Panda", "Spectacled", "Gummy Bears", "Chicago"]

let redundantBears = bears.lazy.filter {
   print("Checking '\($0)'")
   return $0.contains("Bear")
}

print(redundantBears.first!)
```

```
// Lazy Defers Computation

let bears = ["Grizzly", "Panda", "Spectacled", "Gummy Bears", "Chicago"]

let redundantBears = bears.lazy.filter { // LazyFilterCollection<Array<String>>
   print("Checking '\($0)'")
   return $0.contains("Bear")
}

print(redundantBears.first!)
```

```swift
// Lazy Defers Computation

let bears = ["Grizzly", "Panda", "Spectacled", "Gummy Bears", "Chicago"]

let redundantBears = bears.lazy.filter {
   print("Checking '\($0)'")
   return $0.contains("Bear")
}

print(redundantBears.first!)
```

```
// Lazy Defers Computation

let bears = ["Grizzly", "Panda", "Spectacled", "Gummy Bears", "Chicago"]

let redundantBears = bears.lazy.filter {
   print("Checking '\($0)'")
   return $0.contains("Bear")
}

print(redundantBears.first!)
```

```
// Lazy Defers Computation
```

```
let bears = ["Grizzly", "Panda", "Spectacled", "Gummy Bears", "Chicago"]

let redundantBears = bears.lazy.filter {
    print("Checking '\($0)'") // Checking 'Grizzly'
    return $0.contains("Bear")
}

print(redundantBears.first!)
```

```
// Lazy Defers Computation
                ↓
let bears = ["Grizzly", "Panda", "Spectacled", "Gummy Bears", "Chicago"]

let redundantBears = bears.lazy.filter {
   print("Checking '\($0)'") // Checking 'Grizzly'
   return $0.contains("Bear") // false
}

print(redundantBears.first!)
```

false 확인되면 바로 다음으로 간다

```
// Lazy Defers Computation
```

```
let bears = ["Grizzly", "Panda", "Spectacled", "Gummy Bears", "Chicago"]

let redundantBears = bears.lazy.filter {
   print("Checking '\($0)'")
   return $0.contains("Bear")
}

print(redundantBears.first!)
```

```swift
// Lazy Defers Computation
```

```swift
let bears = ["Grizzly", "Panda", "Spectacled", "Gummy Bears", "Chicago"]

let redundantBears = bears.lazy.filter {
    print("Checking '\($0)'") // Checking 'Panda'
    return $0.contains("Bear")
}

print(redundantBears.first!)
```

```
// Lazy Defers Computation
```

```
let bears = ["Grizzly", "Panda", "Spectacled", "Gummy Bears", "Chicago"]

let redundantBears = bears.lazy.filter {
   print("Checking '\($0)'") // Checking 'Panda'
   return $0.contains("Bear") // false
}

print(redundantBears.first!)
```

```
// Lazy Defers Computation
```

```
let bears = ["Grizzly", "Panda", "Spectacled", "Gummy Bears", "Chicago"]

let redundantBears = bears.lazy.filter {
   print("Checking '\($0)'")
   return $0.contains("Bear")
}

print(redundantBears.first!)
```

```
// Lazy Defers Computation

let bears = ["Grizzly", "Panda", "Spectacled", "Gummy Bears", "Chicago"]

let redundantBears = bears.lazy.filter {
    print("Checking '\($0)'") // Checking 'Spectacled'
    return $0.contains("Bear")
}

print(redundantBears.first!)
```

```
// Lazy Defers Computation

let bears = ["Grizzly", "Panda", "Spectacled", "Gummy Bears", "Chicago"]

let redundantBears = bears.lazy.filter {
   print("Checking '\($0)'") // Checking 'Spectacled'
   return $0.contains("Bear") // false
}

print(redundantBears.first!)
```

```
// Lazy Defers Computation

let bears = ["Grizzly", "Panda", "Spectacled", "Gummy Bears", "Chicago"]

let redundantBears = bears.lazy.filter {
   print("Checking '\($0)'")
   return $0.contains("Bear")
}

print(redundantBears.first!)
```


// Lazy Defers Computation
let bears = ["Grizzly", "Panda", "Spectacled", "Gummy Bears", "Chicago"]
let redundantBears = bears.lazy.filter {
print("Checking '\($0)'") // Checking 'Gummy Bears'
return $0.contains("Bear")
}
print(redundantBears.first!)

```
// Lazy Defers Computation

let bears = ["Grizzly", "Panda", "Spectacled", "Gummy Bears", "Chicago"]

let redundantBears = bears.lazy.filter {
   print("Checking '\($0)'") // Checking 'Gummy Bears'
   return $0.contains("Bear") // true
}

print(redundantBears.first!)
```

```swift
// Lazy Defers Computation

let bears = ["Grizzly", "Panda", "Spectacled", "Gummy Bears", "Chicago"]

let redundantBears = bears.lazy.filter {
   print("Checking '\($0)'")
   return $0.contains("Bear")
}

print(redundantBears.first!) // Gummy Bears
```

찾았다

```swift
// Lazy Defers Computation

let bears = ["Grizzly", "Panda", "Spectacled", "Gummy Bears", "Chicago"]

let redundantBears = bears.lazy.filter {
   print("Checking '\($0)'")
   return $0.contains("Bear")
}

print(redundantBears.first!) // Gummy Bears
print(redundantBears.first!)
```

같은 작업을 여러번 할 필요는 없다

```
// Lazy Defers Computation

let bears = ["Grizzly", "Panda", "Spectacled", "Gummy Bears", "Chicago"]

let redundantBears = bears.lazy.filter {
   print("Checking '\($0)'")
   return $0.contains("Bear")
}

print(redundantBears.first!) // Gummy Bears
print(redundantBears.first!)
```

```
// Lazy Defers Computation

let bears = ["Grizzly", "Panda", "Spectacled", "Gummy Bears", "Chicago"]

let redundantBears = bears.lazy.filter {
   print("Checking '\($0)'")
   return $0.contains("Bear")
}

print(redundantBears.first!) // Gummy Bears
print(redundantBears.first!)
```

```swift
// Lazy Defers Computation

let bears = ["Grizzly", "Panda", "Spectacled", "Gummy Bears", "Chicago"]

let redundantBears = bears.lazy.filter {
   print("Checking '\($0)'")
   return $0.contains("Bear")
}

print(redundantBears.first!) // Gummy Bears
print(redundantBears.first!) // Gummy Bears
```

```
// Be Lazy, Exactly Once

let bears = ["Grizzly", "Panda", "Spectacled", "Gummy Bears", "Chicago"]

let redundantBears = bears.lazy.filter {
   print("Checking '\($0)'")
   return $0.contains("Bear")
}

let filteredBears = Array(redundantBears)
print(filteredBears.first!)
```

copy

```
// Be Lazy, Exactly Once

let bears = ["Grizzly", "Panda", "Spectacled", "Gummy Bears", "Chicago"]

let redundantBears = bears.lazy.filter {
   print("Checking '\($0)'")
   return $0.contains("Bear")
}

let filteredBears = Array(redundantBears)
print(filteredBears.first!)
```

```swift
// Be Lazy, Exactly Once

let bears = ["Grizzly", "Panda", "Spectacled", "Gummy Bears", "Chicago"]

let redundantBears = bears.lazy.filter {
   print("Checking '\($0)'")
   return $0.contains("Bear")
}

let filteredBears = Array(redundantBears) // ["Gummy Bears"]
print(filteredBears.first!)
```

```swift
// Be Lazy, Exactly Once

let bears = ["Grizzly", "Panda", "Spectacled", "Gummy Bears", "Chicago"]

let redundantBears = bears.lazy.filter {
   print("Checking '\($0)'")
   return $0.contains("Bear")
}

let filteredBears = Array(redundantBears) // ["Gummy Bears"]
print(filteredBears.first!) // Gummy Bears
```

Advice: When to Be Lazy?

lazy가 필요한 시점은?

Chained computation

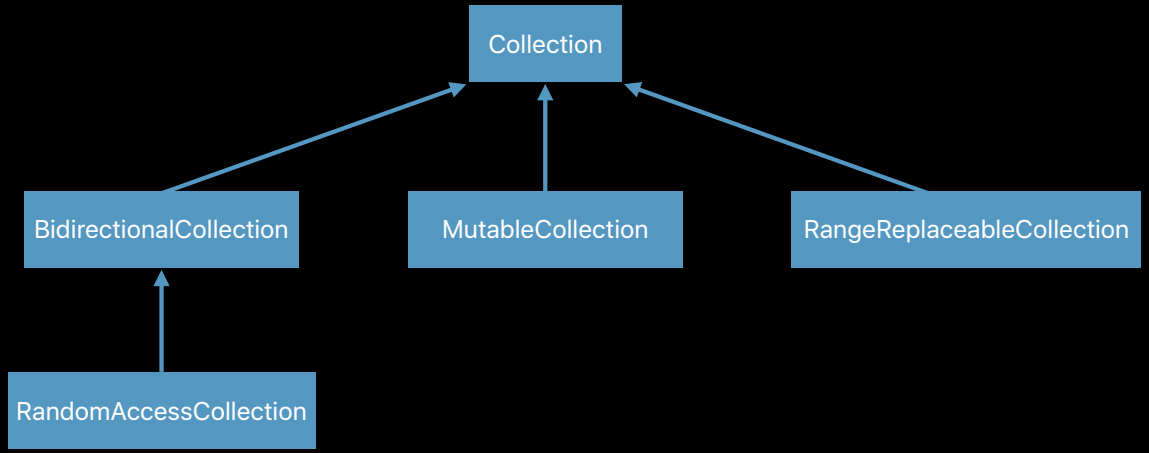
map, filter chain 특히 일부를 필요로 할 때

Only need part of a result

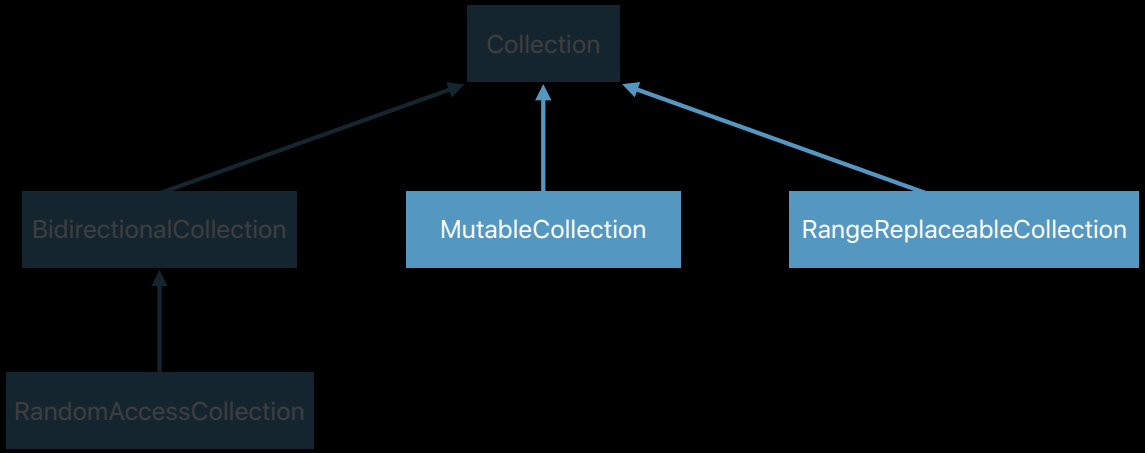
No side effects

Avoid API boundaries

Collections Protocol Hierarchy



Collections Protocol Hierarchy



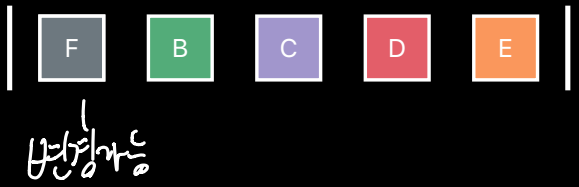
Mutable Collection

setter가 추가된다

```
// constant time
subscript(_: Self.Index) -> Element { get set }
```

Mutable Collection

```
// constant time
subscript(_: Self.Index) -> Element { get set }
```



Range Replaceable Collections

임의 만큼 변경

```
replaceSubrange(_:, with:)
```

Range Replaceable Collections

```
replaceSubrange(_:, with:)
```

Range Replaceable Collections

```
replaceSubrange(_:, with:)
```

Range Replaceable Collections

```
replaceSubrange(_:, with:)
```

Why did this collection code crash?

Collection 에서 crash 나는경우

Follow-Up Questions

Are you mutating your collection? Collection을 변경하려 하는가?

Are your collections accessed from multiple threads?

멀티스레드 접근 하는가?

Crashing Collection Code

```
var array = ["A", "B", "C", "D", "E"]
let index = array.firstIndex(of: "E")!
array.remove(at: array.startIndex)
print(array[index])
```

Crashing Collection Code

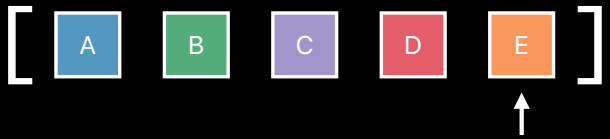
우리는 배열을 가지고 있다.

```
var array = ["A", "B", "C", "D", "E"]
let index = array.firstIndex(of: "E")!
array.remove(at: array.startIndex)
print(array[index])
```

Crashing Collection Code

```
var array = ["A", "B", "C", "D", "E"]
let index = array.firstIndex(of: "E")!
array.remove(at: array.startIndex)
print(array[index])
```

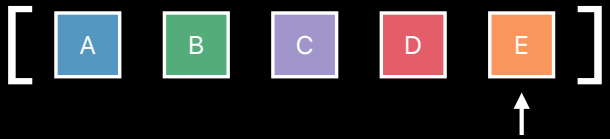
E index 를 가져왔다.



Crashing Collection Code

```
var array = ["A", "B", "C", "D", "E"]
let index = array.firstIndex(of: "E")!
array.remove(at: array.startIndex)
print(array[index])
```

그리고 변이는 일으킨다.



Crashing Collection Code

```
var array = ["A", "B", "C", "D", "E"]
let index = array.firstIndex(of: "E")!
array.remove(at: array.startIndex)
print(array[index])
```

Crashing Collection Code

```
var array = ["A", "B", "C", "D", "E"]
let index = array.firstIndex(of: "E")!
array.remove(at: array.startIndex)
print(array[index])
```

Crashing Collection Code

```
var array = ["A", "B", "C", "D", "E"]
let index = array.firstIndex(of: "E")!
array.remove(at: array.startIndex)
print(array[index])
```

```
Fatal Error: Index out of range.
```

crash

Avoid Index Invalidation

변이를 먼저 일으켜야 한다.

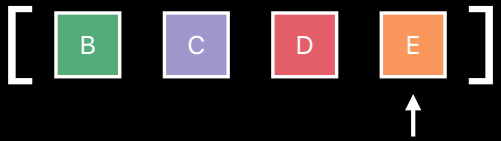
```
var array = ["A", "B", "C", "D", "E"]
array.remove(at: array.startIndex)
if let idx = array.firstIndex(of: "E") {
   print(array[idx])
}
```

Avoid Index Invalidation

```
var array = ["A", "B", "C", "D", "E"]
array.remove(at: array.startIndex)
if let idx = array.firstIndex(of: "E") {
   print(array[idx])
}
```

Avoid Index Invalidation

```
var array = ["A", "B", "C", "D", "E"]
array.remove(at: array.startIndex)
if let idx = array.firstIndex(of: "E") {
   print(array[idx])
}
```



변이는 index를 항상 무효화 시킨다

```
// Reusing Invalid Dictionary Indices

var favorites: [String : String] = [
   "dessert" : "honey ice cream",
   "sleep"   : "hibernation",
   "food"    : "salmon"
]

let foodIndex = favorites.index(forKey: "food")!
print(favorites[foodIndex])

favorites["accessory"] = "tie"
favorites["hobby"] = "stealing picnic supplies"

print(favorites[foodIndex])
```

```
// Reusing Invalid Dictionary Indices

var favorites: [String : String] = [
   "dessert" : "honey ice cream",
   "sleep"   : "hibernation",
   "food"    : "salmon"
]

let foodIndex = favorites.index(forKey: "food")!
print(favorites[foodIndex])

favorites["accessory"] = "tie"
favorites["hobby"] = "stealing picnic supplies"

print(favorites[foodIndex])
```

```swift
// Reusing Invalid Dictionary Indices

var favorites: [String : String] = [
   "dessert" : "honey ice cream",
   "sleep"   : "hibernation",
   "food"    : "salmon"
]

let foodIndex = favorites.index(forKey: "food")!
print(favorites[foodIndex]) // (key: "food", value: "salmon")

favorites["accessory"] = "tie"
favorites["hobby"] = "stealing picnic supplies"

print(favorites[foodIndex])
```

```
// Reusing Invalid Dictionary Indices

var favorites: [String : String] = [
   "dessert" : "honey ice cream",
   "sleep"   : "hibernation",
   "food"    : "salmon"
]

let foodIndex = favorites.index(forKey: "food")!
print(favorites[foodIndex]) // (key: "food", value: "salmon")

favorites["accessory"] = "tie"
favorites["hobby"] = "stealing picnic supplies"

print(favorites[foodIndex])
```

```swift
// Reusing Invalid Dictionary Indices

var favorites: [String : String] = [
   "dessert" : "honey ice cream",
   "sleep"   : "hibernation",
   "food"    : "salmon"
]

let foodIndex = favorites.index(forKey: "food")!
print(favorites[foodIndex]) // (key: "food", value: "salmon")

favorites["accessory"] = "tie"
favorites["hobby"] = "stealing picnic supplies"

print(favorites[foodIndex]) // (key: "sleep", value: "hibernation")
```

```swift
// Reusing Invalid Dictionary Indices

var favorites: [String : String] = [
   "dessert" : "honey ice cream",
   "sleep"   : "hibernation",
   "food"    : "salmon"
]

let foodIndex = favorites.index(forKey: "food")!
print(favorites[foodIndex]) // (key: "food", value: "salmon")

favorites["accessory"] = "tie"
favorites["hobby"] = "stealing picnic supplies"

print(favorites[foodIndex]) // (key: "sleep", value: "hibernation")
```

```
Fatal error: Attempting to access Dictionary
elements using an invalid Index
```

mutation 후에 index는 invalid 하다.

```swift
// Always Work With Up-To-Date Indices

var favorites: [String : String] = [
   "dessert" : "honey ice cream",
   "sleep"   : "hibernation",
   "food"    : "salmon"
]

let foodIndex = favorites.index(forKey: "food")!
print(favorites[foodIndex]) // (key: "food", value: "salmon")

favorites["accessory"] = "tie"
favorites["hobby"] = "stealing picnic supplies"

if let foodIndex = favorites.index(forKey: "food") {
   print(favorites[foodIndex])
}
```

```swift
// Always Work With Up-To-Date Indices

var favorites: [String : String] = [
   "dessert" : "honey ice cream",
   "sleep"   : "hibernation",
   "food"    : "salmon"
]

let foodIndex = favorites.index(forKey: "food")!
print(favorites[foodIndex]) // (key: "food", value: "salmon")

favorites["accessory"] = "tie"
favorites["hobby"] = "stealing picnic supplies"

if let foodIndex = favorites.index(forKey: "food") {
   print(favorites[foodIndex]) // (key: "food", value: "salmon")
}
```

mutation 후에 index를 획득한 것

하지만
index를 자주 구하는 건
비용이 클수도 있다

Advice: Indices and Slices

Use caution when keeping indices/slices 인덱스나 slice 보관은 주의

Mutation invalidates

Calculate only as needed

index를 매번구하는건
linear하게 비용이 드는 collections의 경우.

Are your collections reachable from multiple threads?

Multithreaded Mutable Collections

Our collections optimized for single-threaded access

This is a Good Thing™

Undefined behavior without mutual exclusion

```
// Example of Thread-Unsafe Practices

var sleepingBears = [String]()
let queue = DispatchQueue.global()

queue.async { sleepingBears.append("Grandpa") }
queue.async { sleepingBears.append("Cub") }
```

```
// Example of Thread-Unsafe Practices

var sleepingBears = [String]()
let queue = DispatchQueue.global()

queue.async { sleepingBears.append("Grandpa") }
queue.async { sleepingBears.append("Cub") }
```

```
// Example of Thread-Unsafe Practices

var sleepingBears = [String]()
let queue = DispatchQueue.global()

queue.async { sleepingBears.append("Grandpa") }
queue.async { sleepingBears.append("Cub") }
```

```
// Example of Thread-Unsafe Practices

var sleepingBears = [String]()
let queue = DispatchQueue.global()

queue.async { sleepingBears.append("Grandpa") }
queue.async { sleepingBears.append("Cub") }
```

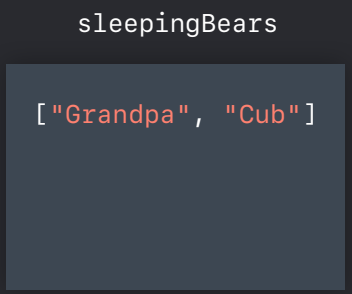
```
// Example of Thread-Unsafe Practices

var sleepingBears = [String]()
let queue = DispatchQueue.global()

queue.async { sleepingBears.append("Grandpa") }   queue.async { sleepingBears.append("Cub") }
```

동시에 접근

```
// Example of Thread-Unsafe Practices

var sleepingBears = [String]()
let queue = DispatchQueue.global()

queue.async { sleepingBears.append("Grandpa") }   queue.async { sleepingBears.append("Cub") }
```

sleepingBears

```swift
// Example of Thread-Unsafe Practices

var sleepingBears = [String]()
let queue = DispatchQueue.global()

queue.async { sleepingBears.append("Grandpa") }   queue.async { sleepingBears.append("Cub") }
```



```swift
// Example of Thread-Unsafe Practices

var sleepingBears = [String]()
let queue = DispatchQueue.global()

queue.async { sleepingBears.append("Grandpa") }   queue.async { sleepingBears.append("Cub") }
```

sleepingBears

```
["Grandpa", "Cub"]
["Cub", "Grandpa"]
```

```swift
// Example of Thread-Unsafe Practices

var sleepingBears = [String]()
let queue = DispatchQueue.global()

queue.async { sleepingBears.append("Grandpa") }   queue.async { sleepingBears.append("Cub") }
```

sleepingBears

```
["Grandpa", "Cub"]
["Cub", "Grandpa"]
   ["Grandpa"]
```

```
// Example of Thread-Unsafe Practices

var sleepingBears = [String]()
let queue = DispatchQueue.global()

queue.async { sleepingBears.append("Grandpa") }   queue.async { sleepingBears.append("Cub") }
```

sleepingBears

```
["Grandpa", "Cub"]
["Cub", "Grandpa"]
   ["Grandpa"]
     ["Cub"]
```

```swift
// Example of Thread-Unsafe Practices

var sleepingBears = [String]()
let queue = DispatchQueue.global()

queue.async { sleepingBears.append("Grandpa") }   queue.async { sleepingBears.append("Cub") }
```

sleepingBears

```
["Grandpa", "Cub"]
["Cub", "Grandpa"]
   ["Grandpa"]
     ["Cub"]
```

```
malloc: *** error
for object
0x100586238: pointer
being freed was not
allocated
```

이상하게나 Crash

```
// Example of Thread-Unsafe Practices

var sleepingBears = [String]()
let queue = DispatchQueue.global()

queue.async { sleepingBears.append("Grandpa") }
queue.async { sleepingBears.append("Cub") }
```

Thread 1

Thread 2

Thread 3

Thread 4

sleepingBears

```
["Grandpa", "Cub"]
["Cub", "Grandpa"]
["Grandpa"]
["Cub"]
```

```
malloc: *** error
for object
0x100586238: pointer
being freed was not
allocated
```

TSAN이 잡아줌.

```
WARNING: ThreadSanitizer: Swift access race
  Modifying access of Swift variable at 0x7b0800023cf0 by thread Thread 3:
    ...

  Previous modifying access of Swift variable at 0x7b0800023cf0 by thread Thread 2:
    ...

  Location is heap block of size 24 at 0x7b0800023ce0 allocated by main thread:
    ...

SUMMARY: ThreadSanitizer: Swift access race main.swift:515 in closure #1 in gotoSleep()
```

q class race

```
WARNING: ThreadSanitizer: Swift access race
  Modifying access of Swift variable at 0x7b0800023cf0 by thread Thread 3:
    ...

  Previous modifying access of Swift variable at 0x7b0800023cf0 by thread Thread 2:
    ...

  Location is heap block of size 24 at 0x7b0800023ce0 allocated by main thread:
    ...

SUMMARY: ThreadSanitizer: Swift access race main.swift:515 in closure #1 in gotoSleep()
```

```
WARNING: ThreadSanitizer: Swift access race
  Modifying access of Swift variable at 0x7b0800023cf0 by thread Thread 3:
    ...

  Previous modifying access of Swift variable at 0x7b0800023cf0 by thread Thread 2:
    ...

  Location is heap block of size 24 at 0x7b0800023ce0 allocated by main thread:
    ...

SUMMARY: ThreadSanitizer: Swift access race main.swift:515 in closure #1 in gotoSleep()
```

```
// Avoid concurrent mutation

var sleepingBears = [String]()
let queue = DispatchQueue(label: "Bear-Cave")

queue.async { sleepingBears.append("Grandpa") }   queue.async { sleepingBears.append("Cub") }
```

```
// Avoid concurrent mutation

var sleepingBears = [String]()
let queue = DispatchQueue(label: "Bear-Cave")

queue.async { sleepingBears.append("Grandpa") }   queue.async { sleepingBears.append("Cub") }
```

```
// Avoid concurrent mutation

var sleepingBears = [String]()
let queue = DispatchQueue(label: "Bear-Cave")

queue.async { sleepingBears.append("Grandpa") }
queue.async { sleepingBears.append("Cub") }
```

```swift
// Avoid concurrent mutation

var sleepingBears = [String]()
let queue = DispatchQueue(label: "Bear-Cave")

queue.async { sleepingBears.append("Grandpa") }
queue.async { sleepingBears.append("Cub") }
queue.async { print(sleepingBears) }
```

Serial Q

```swift
// Avoid concurrent mutation

var sleepingBears = [String]()
let queue = DispatchQueue(label: "Bear-Cave")

queue.async { sleepingBears.append("Grandpa") }
queue.async { sleepingBears.append("Cub") }
queue.async { print(sleepingBears) }
```

```
["Grandpa", "Cub"]
```

Advice: Multithreading

Prefer state accessible from a single thread

When this is not possible:

- Ensure mutual exclusion
- Use TSAN

Advice: Prefer Immutable Collections

Easier to reason about data that can't change

Less surface area for bugs

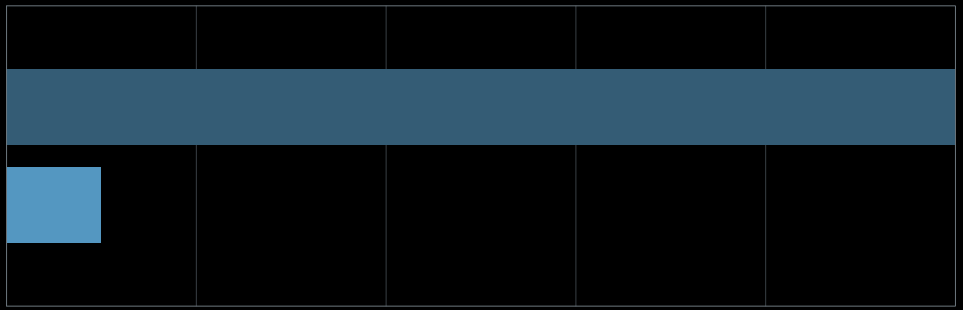
Emulate mutation with slices and lazy

The compiler will help you

Advice: Forming New collections

Use capacity hints if possible

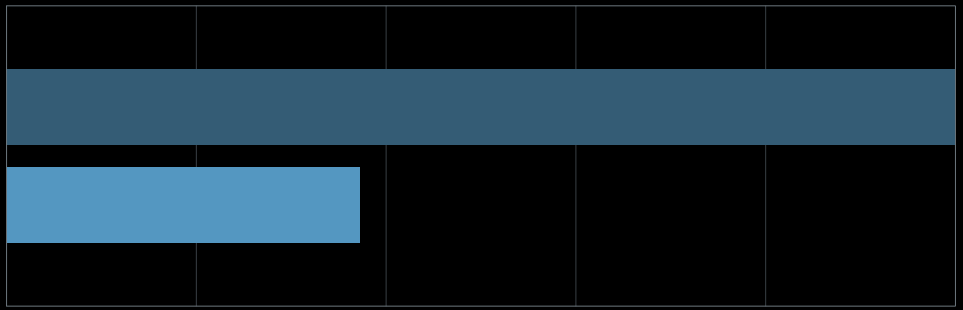
```
Array.reserveCapacity(_:)
Set(minimumCapacity:)
Dictionary(minimumCapacity:)
```



Advice: Forming New collections

Use capacity hints if possible

```
Array.reserveCapacity(_:)
Set(minimumCapacity:)
Dictionary(minimumCapacity:)
```



Advice: Forming New collections

Use capacity hints if possible

```
Array.reserveCapacity(_:)
Set(minimumCapacity:)
Dictionary(minimumCapacity:)
```

Advice: Forming New collections

Use capacity hints if possible

```
Array.reserveCapacity(_:)
Set(minimumCapacity:)
Dictionary(minimumCapacity:)
```

Advice: Forming New collections

Use capacity hints if possible

```
Array.reserveCapacity(_:)
Set(minimumCapacity:)
Dictionary(minimumCapacity:)
```

Foundation Collections

Objc 개발자들이 사랑한

Reference Type Collections

NSArray
NSMutableArray
NSPointerArray
NSData

NSSet
NSMutableSet
NSCountedSet
NSMutableOrderedSet
NSHashTable
NSIndexSet
NSCharacterSet

NSDictionary
NSMutableDictionary
NSMapTable

Value and Reference Collections

Value and Reference Collections

```
// Value
```

```
// Reference
```

Value and Reference Collections

```
// Value
var x: [String] = []
```

```
// Reference
let x = NSMutableArray()
```

Value and Reference Collections

```swift
// Value
var x: [String] = []
```


```swift
// Reference
let x = NSMutableArray()
```

Value and Reference Collections

```
// Value
var x: [String] = []
```


```
// Reference
let x = NSMutableArray()
```

Value and Reference Collections

```swift
// Value
var x: [String] = []
x.append("🐻")
```


```swift
// Reference
let x = NSMutableArray()
x.add("🐻")
```

Value and Reference Collections

```swift
// Value
var x: [String] = []
x.append("🐻")
```


```swift
// Reference
let x = NSMutableArray()
x.add("🐻")
```

Value and Reference Collections

```swift
// Value
var x: [String] = []
x.append("🐻")
```


```swift
// Reference
let x = NSMutableArray()
x.add("🐻")
```

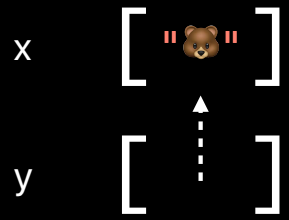
Value and Reference Collections

```swift
// Value
var x: [String] = []
x.append("🐻")
var y = x
```


```swift
// Reference
let x = NSMutableArray()
x.add("🐻")
let y = x
```

Value and Reference Collections

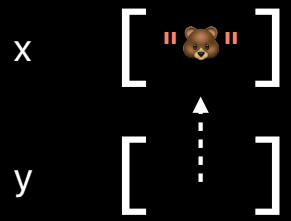
```swift
// Value
var x: [String] = []
x.append("🐻")
var y = x
```



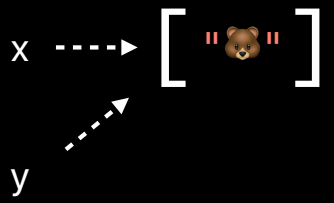

```swift
// Reference
let x = NSMutableArray()
x.add("🐻")
let y = x
```

Value and Reference Collections

```swift
// Value
var x: [String] = []
x.append("🐻")
var y = x
```

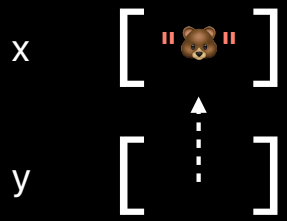



```swift
// Reference
let x = NSMutableArray()
x.add("🐻")
let y = x
```

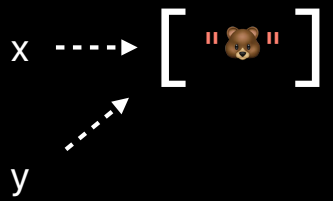


Value and Reference Collections

```swift
// Value
var x: [String] = []
x.append("🐻")
var y = x
y.append("🐼")
```

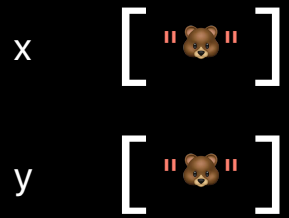



```swift
// Reference
let x = NSMutableArray()
x.add("🐻")
let y = x
y.add("🐼")
```

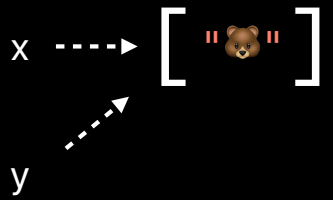


Value and Reference Collections

```swift
// Value
var x: [String] = []
x.append("🐻")
var y = x
y.append("🐼")
```

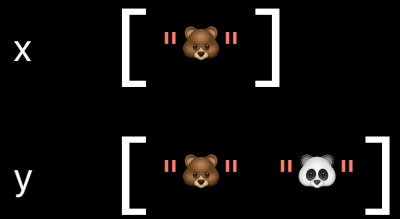



```swift
// Reference
let x = NSMutableArray()
x.add("🐻")
let y = x
y.add("🐼")
```

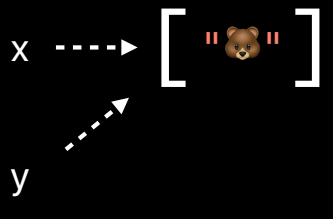


Value and Reference Collections

```swift
// Value
var x: [String] = []
x.append("🐻")
var y = x
y.append("🐼")
```

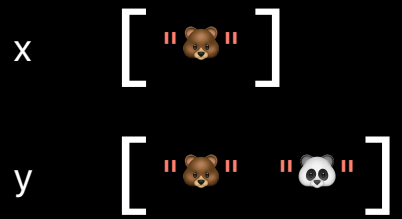



```swift
// Reference
let x = NSMutableArray()
x.add("🐻")
let y = x
y.add("🐼")
```

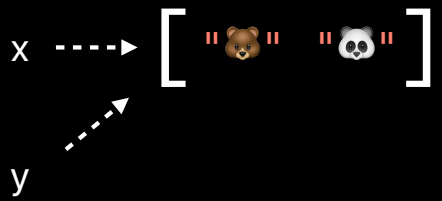


Value and Reference Collections

```swift
// Value
var x: [String] = []
x.append("🐻")
var y = x
y.append("🐼")
```




```swift
// Reference
let x = NSMutableArray()
x.add("🐻")
let y = x
y.add("🐼")
```

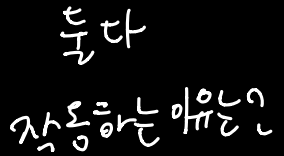


Objective-C APIs in Swift

```
// Objective-C
@interface NSView
@property NSArray<NSView *> *subviews;
@end
```

Objective-C APIs in Swift

```
// Objective-C
@interface NSView
@property NSArray<NSView *> *subviews;
@end
```




```
// Swift
class NSView {
   var subviews: [NSView]
}
```

Bridging

Converts between runtime types

두 언어 사이의 다리

Bidirectional

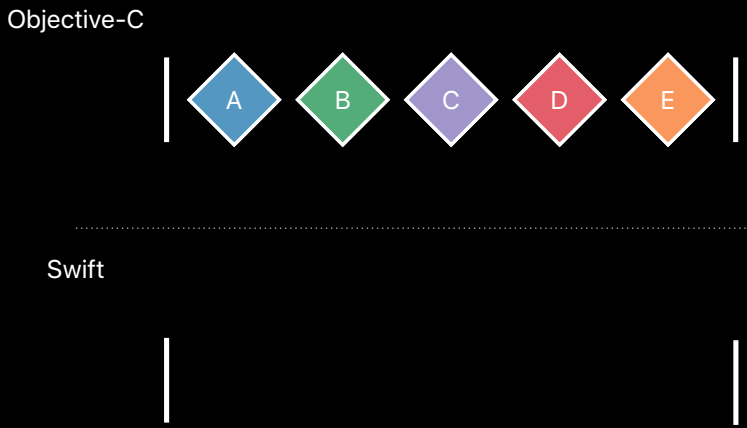
Bridging of collections

- Is necessary
- Can be cheap, but is never free

How Bridging Works

Objective-C

How Bridging Works



How Bridging Works

Objective-C

Swift

Two Kinds of Bridging

Eager when element types are bridged

Otherwise lazy

- Bridged on first use

Bridging Examples

Bridging Examples

`NSArray<NSData *> *`	`[Data]`	Eager

Bridging Examples

NSData – Data

`NSArray<NSData *> *`	`[Data]`	Eager
`NSArray<NSView *> *`	`[NSView]`	Lazy

NSView는 브릿지 되어있지 않다
여전히 참조타입

Bridging Examples

`NSArray<NSData *> *`	`[Data]`	Eager
`NSArray<NSView *> *`	`[NSView]`	Lazy
`NSDictionary<NSString *, id> *`	`[String : Any]`	Eager

Identifying Bridging Problems 브릿징 문제 식별

Measure your performance with Instruments

Especially inside loops at language boundaries

Look for hotspots like:

- `_unconditionallyBridgeFromObjectiveC`
- `bridgeEverything`

```
// A Story About Bridging

let story = NSString(string: """
   Once upon time there lived a family of Brown Bears. They had long brown hair.
   ...
   They were happy with their new hair cuts. The end.
   """)
```

```
// A Story About Bridging

let story = NSString(string: """
   Once upon time there lived a family of Brown Bears. They had long brown hair.
   ...
   They were happy with their new hair cuts. The end.
   """)

let text = NSMutableAttributedString(string: story)


let range = text.string.range(of: "Brown")!
let nsrange = NSRange(range, in: text.string)

text.addAttribute(.foregroundColor, value: NSColor.brown, range: nsrange)
```

```
// A Story About Bridging

let story = NSString(string: """
   Once upon time there lived a family of Brown Bears. They had long brown hair.
   ...
   They were happy with their new hair cuts. The end.
   """)

let text = NSMutableAttributedString(string: story)


let range = text.string.range(of: "Brown")!   // Range<String.Index>
let nsrange = NSRange(range, in: text.string)

text.addAttribute(.foregroundColor, value: NSColor.brown, range: nsrange)
```

```
// A Story About Bridging

let story = NSString(string: """
   Once upon time there lived a family of Brown Bears. They had long brown hair.
   ...
   They were happy with their new hair cuts. The end.
   """)

let text = NSMutableAttributedString(string: story)


let range = text.string.range(of: "Brown")!
let nsrange = NSRange(range, in: text.string)    // NSRange

text.addAttribute(.foregroundColor, value: NSColor.brown, range: nsrange)
```

```swift
// A Story About Bridging

let story = NSString(string: """
   Once upon time there lived a family of Brown Bears. They had long brown hair.
   ...
   They were happy with their new hair cuts. The end.
   """)

let text = NSMutableAttributedString(string: story)


let range = text.string.range(of: "Brown")!
let nsrange = NSRange(range, in: text.string)

text.addAttribute(.foregroundColor, value: NSColor.brown, range: nsrange)
```

이게 느리게 동작한다...

```
// A Story About Bridging

let story = NSString(string: """
   Once upon time there lived a family of Brown Bears. They had long brown hair.
   ...
   They were happy with their new hair cuts. The end.
   """)

let text = NSMutableAttributedString(string: story)


let range = text.string.range(of: "Brown")!
let nsrange = NSRange(range, in: text.string)

text.addAttribute(.foregroundColor, value: NSColor.brown, range: nsrange)
```

```
// A Story About Bridging

let story = NSString(string: """
   Once upon time there lived a family of Brown Bears. They had long brown hair.
   ...
   They were happy with their new hair cuts. The end.
   """)

let text = NSMutableAttributedString(string: story)

let range = text.string.range(of: "Brown")!
let nsrange = NSRange(range, in: text.string)

text.addAttribute(.foregroundColor, value: NSColor.brown, range: nsrange)
```

10 ms

브릿지를 여러번 하고있다

400 ms

400 ms

25 ms

```
// A Story About Bridging

let story = NSString(string: """
   Once upon time there lived a family of Brown Bears. They had long brown hair.
   ...
   They were happy with their new hair cuts. The end.
   """)

let text = NSMutableAttributedString(string: story)

let range = text.string.range(of: "Brown")!
let nsrange = NSRange(range, in: text.string)

text.addAttribute(.foregroundColor, value: NSColor.brown, range: nsrange)
```

10 ms

400 ms

400 ms

25 ms

```
// A Story About Bridging

let story = NSString(string: """
   Once upon time there lived a family of Brown Bears. They had long brown hair.
   ...
   They were happy with their new hair cuts. The end.
   """)

let text = NSMutableAttributedString(string: story)

let range = text.string.range(of: "Brown")!
let nsrange = NSRange(range, in: text.string)

text.addAttribute(.foregroundColor, value: NSColor.brown, range: nsrange)
```

10 ms

400 ms

400 ms

25 ms

When Bridging Happens

```
text.string
```

When Bridging Happens

```
text.string
```

```
class NSMutableAttributedString : ... {
   var string: String
}
```

When Bridging Happens

```
text.string
```

```
class NSMutableAttributedString : ... {
   var string: String
}
```

```
@interface NSMutableAttributedString : ...
@property NSString *string;
@end
```

When Bridging Happens

```
text.string
```

```
class NSMutableAttributedString : ... {
   var string: String
}
```

```
@interface NSMutableAttributedString : ...
@property NSString *string;
@end
```

"Once upon time ... The end."

NSString

When Bridging Happens

```
text.string
```

```
class NSMutableAttributedString : ... {
   var string: String
}
```

```
@interface NSMutableAttributedString : ...
@property NSString *string;
@end
```

"Once upon time ... The end."

String

"Once upon time ... The end."

NSString

When Bridging Happens

```
@interface NSMutableAttributedString : ...
@property NSString *string;
@end
```

```
class NSMutableAttributedString : ... {
   var string: String
}
```

```
@interface NSString : ...
- (NSRange)rangeOfString:(NSString *)string;
@end
```

```
struct String {
  func range(of: StringPrococol) -> Range
}
```

When Bridging Happens

```objc
@interface NSMutableAttributedString : ...
@property NSString *string;
@end
```

```swift
class NSMutableAttributedString : ... {
   var string: String
}
```

```objc
@interface NSString : ...
- (NSRange)rangeOfString:(NSString *)string;
@end
```

```swift
struct String {
  func range(of: StringPrococol) -> Range
}
```

When Bridging Happens

```
@interface NSMutableAttributedString : ...
@property NSString *string;
@end
```

```
class NSMutableAttributedString : ... {
   var string: String
}
```

```
@interface NSString : ...
- (NSRange)rangeOfString:(NSString *)string;
@end
```

```
struct String {
  func range(of: StringPrococol) -> Range
}
```

더 좋게 바꿔보자

```
// A Story About Bridging

let story = NSString(string: """
   Once upon time there lived a family of Brown Bears. They had long brown hair.
   ...
   They were happy with their new hair cuts. The end.
   """)

let text = NSMutableAttributedString(string: story)

let range = text.string.range(of: "Brown")!
let nsrange = NSRange(range, in: text.string)

text.addAttribute(.foregroundColor, value: NSColor.brown, range: nsrange)
```

10 ms

400 ms

400 ms

25 ms

```swift
// A Story About Bridging

let story = NSString(string: """
   Once upon time there lived a family of Brown Bears. They had long brown hair.
   ...
   They were happy with their new hair cuts. The end.
   """)

let text = NSMutableAttributedString(string: story)
let string = text.string

let range = string.range(of: "Brown")!
let nsrange = NSRange(range, in: string)

text.addAttribute(.foregroundColor, value: NSColor.brown, range: nsrange)
```

```
// A Story About Bridging

let story = NSString(string: """
   Once upon time there lived a family of Brown Bears. They had long brown hair.
   ...
   They were happy with their new hair cuts. The end.
   """)

let text = NSMutableAttributedString(string: story)
let string = text.string

let range = string.range(of: "Brown")!
let nsrange = NSRange(range, in: string)

text.addAttribute(.foregroundColor, value: NSColor.brown, range: nsrange)
```

10 ms

400 ms

3 ms

25 ms

```
// A Story About Bridging

let story = NSString(string: """
   Once upon time there lived a family of Brown Bears. They had long brown hair.
   ...
   They were happy with their new hair cuts. The end.
   """)

let text = NSMutableAttributedString(string: story)
let string = text.string

let range = string.range(of: "Brown")!
let nsrange = NSRange(range, in: string)

text.addAttribute(.foregroundColor, value: NSColor.brown, range: nsrange)
```

10 ms

400 ms

3 ms

25 ms

```
// A Story About Bridging

let story = NSString(string: """
   Once upon time there lived a family of Brown Bears. They had long brown hair.
   ...
   They were happy with their new hair cuts. The end.
   """)

let text = NSMutableAttributedString(string: story)
let string = text.string

let range = string.range(of: "Brown")!
let nsrange = NSRange(range, in: string)

text.addAttribute(.foregroundColor, value: NSColor.brown, range: nsrange)
```

10 ms

400 ms

3 ms

25 ms

```swift
// A Story About Bridging

let story = NSString(string: """
   Once upon time there lived a family of Brown Bears. They had long brown hair.
   ...
   They were happy with their new hair cuts. The end.
   """)

let text = NSMutableAttributedString(string: story)
let string = text.string as NSString    // NSString

let nsrange = string.range(of: "Brown")

text.addAttribute(.foregroundColor, value: NSColor.brown, range: nsrange)
```

NEW

브릿지가 없음

```
// A Story About Bridging

let story = NSString(string: """
   Once upon time there lived a family of Brown Bears. They had long brown hair.
   ...
   They were happy with their new hair cuts. The end.
   """)

let text = NSMutableAttributedString(string: story)
let string = text.string as NSString    // NSString


let nsrange = string.range(of: "Brown")      // NSRange

text.addAttribute(.foregroundColor, value: NSColor.brown, range: nsrange)
```

NEW

```
// A Story About Bridging

let story = NSString(string: """
   Once upon time there lived a family of Brown Bears. They had long brown hair.
   ...
   They were happy with their new hair cuts. The end.
   """)

let text = NSMutableAttributedString(string: story)
let string = text.string as NSString    // NSString

let nsrange = string.range(of: "Brown")      // NSRange

text.addAttribute(.foregroundColor, value: NSColor.brown, range: nsrange)
```

NEW

10 ms

1 ms

3 ms

25 ms

```
// A Story About Bridging

let story = NSString(string: """
   Once upon time there lived a family of Brown Bears. They had long brown hair.
   ...
   They were happy with their new hair cuts. The end.
   """)

let text = NSMutableAttributedString(string: story)
let string = text.string as NSString    // NSString

let nsrange = string.range(of: "Brown")   // NSRange

text.addAttribute(.foregroundColor, value: NSColor.brown, range: nsrange)
```

NEW

10 ms

1 ms

3 ms

25 ms

좋은 브릿지가 마련되어 있다

"Brown" is String

```
// A Story About Bridging

let story = NSString(string: """
   Once upon time there lived a family of Brown Bears. They had long brown hair.
   ...
   They were happy with their new hair cuts. The end.
   """)

let text = NSMutableAttributedString(string: story)
let string = text.string as NSString    // NSString

let nsrange = string.range(of: "Brown") // NSRange

text.addAttribute(.foregroundColor, value: NSColor.brown, range: nsrange)
```

NEW

10 ms

1 ms

3 ms

25 ms

옵션을 많이 주면 이것도 복잡

Advice: When to Use Foundation Collections

You need reference semantics

You are working with known proxies

- `NSAttributedString.string`
- Core Data Managed Objects

You've measured and identified bridging costs

Now It’s Your Turn

Explore your existing collections

Measure your code

Audit your mutable state

Gain mastery in Playgrounds

More Information

https://developer.apple.com/wwdc18/229

Cocoa Lab	Technology Lab 7	Friday 11:00AM
Swift Open Hours	Technology Lab 10	Friday 3:00PM


WWDC18