



Swift 4

Highlights

Jonathan Lehr, Founder and VP, Training

About Objects

- Reston, VA
- Full-stack consulting (NFL, Marriott, Chicos, etc.) and training
- Strong focus on iOS and middleware
- Roots in NeXT, OpenStep, WebObjects + enterprise backends

Resources

[GitHub.com/AboutObjectsTraining](https://github.com/AboutObjectsTraining)

- [Swift 4 Highlights](#)
- [Swift 3 Lowlights](#)

[Ole Begemann \(oleb.net/blog\)](http://oleb.net/blog)

- [Playground: What's new in Swift 4](#)

Swift 4 Overview

New/Enhanced in Swift 4



- Strings and One-Sided Ranges
- Collections
 - Dictionary
 - Set
- Key-Value Coding
- Codable Protocol and JSON Support

Migrating From Swift 3

- Explicit `@objc` directives required on a per-method basis to enable dynamic dispatch
- Some Swift 4 `String` APIs now return a new type, `Substring`

Strings

Strings in Swift 2 and Swift 3

- Dropped `Collection` conformance
- Added `characters` property containing collection of `Character` (extended grapheme cluster)
- Substrings referred to original string's storage
 -  Very efficient
 -  Potential memory leak

Swift 3 String API Clutziness

```
let stars = "☆☆☆☆☆☆☆☆"
let charView1 = stars.characters.dropFirst(2)
print(charView1)
// CharacterView(_core:
// Swift._StringCore(_baseAddress: Optional(0x0000000101362454),
// _countAndFlags: 9223372036854775816, _owner: nil), _coreOffset: 2)

let charView2 = charView1.dropLast(3)
// Still a CharacterView 🙄
print(String(charView2))
// ☆☆☆☆☆

// Swift 4
print(stars.dropFirst(2).dropLast(3))
// ☆☆☆☆☆
```

Swift 3 String API Clutziness

```
let stars = "★★★★★☆☆☆☆"
let charView1 = stars.characters.dropFirst(2)
print(charView1)
// CharacterView(_core:
// Swift._StringCore(_baseAddress: Optional(0x0000000101362454),
// _countAndFlags: 9223372036854775816, _owner: nil), _coreOffset: 2)

let charView2 = charView1.dropLast(3)
// Still a CharacterView 🙄
print(String(charView2))
// ★★★★★

// Swift 4
print(stars.dropFirst(2).dropLast(3))
// ★★★★★
```

Swift 3 String API Clutziness

```
let stars = "★★★★★☆☆☆☆"
let charView1 = stars.characters.dropFirst(2)
print(charView1)
// CharacterView(_core:
// Swift._StringCore(_baseAddress: Optional(0x0000000101362454),
// _countAndFlags: 9223372036854775816, _owner: nil), _coreOffset: 2)

let charView2 = charView1.dropLast(3)
// Still a CharacterView 🙄
print(String(charView2))
// ★★★★★

// Swift 4
print(stars.dropFirst(2).dropLast(3))
// ★★★★★
```

Swift 4 Strings (SE-0163)

- Adds back `Collection` conformance and deprecates `characters` property
- Adds `Substring` type
 - Prevents leaks by helping developers avoid accidental storage of `Substring` instances
 - `String` and `Substring` share API by conforming to `StringProtocol`

String Collection API Examples

```
// Looping through a string's characters:
```

```
let s = "abc"
```

```
for c in s {  
    print(c)
```

```
}
```

```
// a
```

```
// b
```

```
// c
```

```
// Inserting characters:
```

```
var name = "Fred Smith"
```

```
let index = name.index(of: " ") ?? name.endIndex
```

```
name.insert(contentsOf: " W.", at: index)
```

```
// Fred W. Smith
```

String Collection API Examples

```
// Looping through a string's characters:
```

```
let s = "abc"
```

```
for c in s {  
    print(c)
```

```
}
```

```
// a
```

```
// b
```

```
// c
```

```
// Inserting characters:
```

```
var name = "Fred Smith"
```

```
let index = name.index(of: " ") ?? name.endIndex
```

```
name.insert(contentsOf: " W.", at: index)
```

```
// Fred W. Smith
```

Substring Example

```
let name = "Fred Smith"  
let last: Substring = name.dropFirst(5)  
// type ^^^^ shown for clarity  
print(last) // "Smith"
```

```
struct Dude {  
    var name: String?  
}
```

```
var dude = Dude()
```

```
dude.name = name //  Doesn't compile
```

Substring Example

```
let name = "Fred Smith"
let last: Substring = name.dropFirst(5)
// type ^^^^^^^^^ shown for clarity
print(last) // "Smith"
```

```
struct Dude {
    var name: String?
}
```

```
var dude = Dude()
```

```
dude.name = name //  Doesn't compile
```


Multi-Line String Literals (SE-168)

- Enclosed in triple-quotes
- Whitespace up to trailing quotes ignored

```
let year = 2017
```

```
let numPages = 240
```

```
let jsonText = """  
  {
```

```
    "title": "War of the Worlds",
```

```
    "author": "H. G. Wells",
```

```
    "publication_year": \ (year),
```

```
    "number_of_pages": \ (numPages)
```

```
  }
```

```
  """
```

One-Sided Ranges (SE-172)

- Ranges can be expressed without explicit starting or ending values

```
let s = "Hello 🌍!"  
// Compute an index relative to start of string.  
let index = s.index(s.startIndex, offsetBy: 6)  
  
let head = s[..<index]  
print(head) // "Hello "  
  
let tail = s[index...]  
print(tail) // "🌍!"
```

Collections

Dictionary Keys and Values (SE-154)

- Adds type-specific collections for keys and values
 - Faster key lookups
 - More efficient value mutation

```
let books = ["Emma": 11.95, "Henry V": 14.99,  
            "1984": 14.99, "Utopia": 11.95]
```

```
guard let index = books.index(forKey: "Emma") else { return }  
print(books.values[index])  
// 11.95
```

Dictionary & Set Enhancements(SE-165)

- Dictionary-specific `map` and `filter`
- Grouping sequence elements
- Default values for subscripts
- Merging dictionaries

Dictionary-Specific Filter

```
let books = ["Emma": 11.95, "Henry V": 14.99,  
            "1984": 14.99, "Utopia": 11.95]  
  
// In Swift 3, Dictionary's `filter` method returned an  
// array of key-value tuples instead of a dictionary.  
let cheapBooks = books.filter { $0.value < 12.00 }  
// [(key: "Utopia", value: 11.95), (key: "Emma", value: 11.95)]  
  
// If you need a Dictionary result, you have to produce one manually  
let cheapBooksDict = cheapBooks.reduce([:]) {  
    var dict = $0  
    dict[$1.key] = $1.value  
    return dict  
}  
// ["Utopia": 11.95, "Emma": 11.95]
```

Dictionary-Specific Filter

```
let books = ["Emma": 11.95, "Henry V": 14.99,
             "1984": 14.99, "Utopia": 11.95]

// In Swift 3, Dictionary's `filter` method returned an
// array of key-value tuples instead of a dictionary.
let cheapBooks = books.filter { $0.value < 12.00 }
// ["Utopia": 11.95, "Emma": 11.95]

// If you need a Dictionary result, you have to produce one manually
let cheapBooksDict = cheapBooks.reduce([:]) {
    var dict = $0
    dict[$1.key] = $1.value
    return dict
}
// ["Utopia": 11.95, "Emma": 11.95]
```

Dictionary-Specific Map

```
let books = ["Emma": 11.95, "Henry V": 14.99,  
            "1984": 14.99, "Utopia": 11.95]
```

```
// Similarly, Dictionary's `map` method returns an array of values  
let discount = 0.10  
let discountedPrices = books.map { $0.value * (1 - discount) }  
// [10.75, 13.49, 10.75, 13.49]
```

```
// That's fine if you simply want to sum the values, but suppose  
// you want to produce a list of discounted prices?
```

```
// Swift 4 adds `mapValues`, which returns a Dictionary  
let discount = 0.10  
let discountedBooks = books.mapValues { $0 * (1 - discount) }  
// ["Utopia": 10.75, "1984": 13.49, "Emma": 10.75, "Henry V": 13.49]
```


Dictionary-Specific Map

```
let books = ["Emma": 11.95, "Henry V": 14.99,  
            "1984": 14.99, "Utopia": 11.95]
```

```
// Similarly, Dictionary's `map` method returns an array of values  
let discount = 0.10  
let discountedPrices = books.map { $0.value * (1 - discount) }  
// [10.75, 13.49, 10.75, 13.49]
```

```
// That's fine if you simply want to sum the values, but suppose  
// you want to produce a list of discounted prices?
```

```
// Swift 4 adds `mapValues`, which returns a Dictionary  
let discount = 0.10  
let discountedBooks = books.mapValues { $0 * (1 - discount) }  
// ["Utopia": 10.75, "1984": 13.49, "Emma": 10.75, "Henry V": 13.49]
```

Dictionary-Specific Map

```
let books = ["Emma": 11.95, "Henry V": 14.99,  
            "1984": 14.99, "Utopia": 11.95]  
  
// Similarly, Dictionary's `map` method returns an array of values  
let discount = 0.10  
let discountedPrices = books.map { $0.value * (1 - discount) }  
// [10.75, 13.49, 10.75, 13.49]  
  
// That's fine if you simply want to sum the values, but suppose  
// you want to produce a list of discounted prices?  
  
// Swift 4 adds `mapValues`, which returns a Dictionary  
let discount = 0.10  
let discountedBooks = books.mapValues { $0 * (1 - discount) }  
// ["Utopia": 10.75, "1984": 13.49, "Emma": 10.75, "Henry V": 13.49]
```

Grouping Sequence Elements

- Swift 4 adds a new initializer for grouping sequences of values. 🔥

```
let books = ["Emma": 11.95, "Henry V": 14.99,  
            "1984": 14.99, "Utopia": 11.95]
```

```
let booksByPrice = Dictionary(grouping: books, by: { $0.value })
```

```
// [11.95: [(key: "Utopia", value: 11.95),  
//         (key: "Emma", value: 11.95)],  
// 14.99: [(key: "1984", value: 14.99),  
//         (key: "Henry V", value: 14.99)]]
```

Default Values for Subscripts

```
// Access with default value may not seem like a huge win
```

```
let books = ["Emma": 11.95, "Henry V": 14.99,  
             "1984": 14.99, "Utopia": 11.95]
```

```
// Swift 3:
```

```
let price = books["Foo"] ?? 0
```

```
// Swift 4:
```

```
let price2 = books["Foo", default: 0]
```

```
// ...but mutation with a default value is 😎
```

```
var discountedBooks = books
```

```
let keys = ["Emma", "1984", "Foo"]
```

```
for key in keys {
```

```
    discountedBooks[key, default: 0] *= 0.9
```

```
}
```

```
// ["Utopia": 11.95, "1984": 13.49, "Foo": 0.0, "Emma": 10.75, "Henry V": 14.99]
```

Default Values for Subscripts

```
// Access with default value may not seem like a huge win

let books = ["Emma": 11.95, "Henry V": 14.99,
             "1984": 14.99, "Utopia": 11.95]

// Swift 3:
let price = books["Foo"] ?? 0
// Swift 4:
let price2 = books["Foo", default: 0]

// ...but mutation with a default value is 😎
var discountedBooks = books
let keys = ["Emma", "1984", "Foo"]
for key in keys {
    discountedBooks[key, default: 0] *= 0.9
}
// ["Utopia": 11.95, "1984": 13.49, "Foo": 0.0, "Emma": 10.75, "Henry V": 14.99]
```

Merging Dictionaries

```
let personal = ["home": "703-333-4567", "cell": "202-444-1234"]
let work = ["main": "571-222-9876", "cell": "703-987-5678"]

// If keys match, replaces the current value with the newer value
var phones1 = personal
phones1.merge(work) { _, new in new }
["main": "571-222-9876", "cell": "703-987-5678", "home": "703-333-4567"]

// If keys match, replaces the current value with a tuple of both values
var phones2: [String: Any] = personal
phones2.merge(work) { (personal: $0, work: $1) }
["main": "571-222-9876",
 "cell": (personal: "202-444-1234", work: "703-987-5678"),
 "home": "703-333-4567"]
```

Merging Dictionaries

```
let personal = ["home": "703-333-4567", "cell": "202-444-1234"]
let work = ["main": "571-222-9876", "cell": "703-987-5678"]

// If keys match, replaces the current value with the newer value
var phones1 = personal
phones1.merge(work) { _, new in new }
["main": "571-222-9876", "cell": "703-987-5678", "home": "703-333-4567"]

// If keys match, replaces the current value with a tuple of both values
var phones2: [String: Any] = personal
phones2.merge(work) { (personal: $0, work: $1) }
["main": "571-222-9876",
 "cell": (personal: "202-444-1234", work: "703-987-5678"),
 "home": "703-333-4567"]
```

Key-Value Coding

Smart KeyPaths (SE-161)

- Allows key paths to be used with non-objc types
- New expression syntax for key paths
 - Similar to property reference, but prefixed with \ for example, \Book.rating
 - Expression result is an instance of KeyPath

Smart KeyPaths Example (1)

```
struct Person {  
    var name: String  
    var address: Address  
}
```

```
struct Address: CustomStringConvertible {  
    var street: String  
    var city: String  
}
```

```
let address = Address(street: "21 Elm", city: "Reston")  
let person = Person(name: "Jo", address: address)
```

```
let name = person[keyPath: \Person.name]  
// "Jo"  
let city = person[keyPath: \Person.address.city]  
// "Reston"
```

Smart KeyPaths Example (2)

- Instances of KeyPath can be stored

```
let address = Address(street: "21 Elm", city: "Reston")
let person = Person(name: "Jo", address: address)
```

```
// Initialize an array of KeyPaths
let keyPaths = [\Person.name,
                \Person.address.city,
                \Person.address.street]
```

```
// Map KeyPaths to an array of property values
let values = keyPaths.map { person[keyPath: $0] }
```

```
// ["Jo", "Reston", "21 Elm"]
```

Smart KeyPaths Example (3)

- You can use KeyPaths to mutate properties of non-ObjC types

```
// KeyPaths allow you to mutate properties of Swift types
```

```
let address = Address(street: "21 Elm", city: "Reston")
var mutablePerson = Person(name: "Jo", address: address)
```

```
mutablePerson[keyPath: \Person.name] = "Kay"
mutablePerson[keyPath: \Person.address.city] = "Herndon"
```

```
// Person(name: "Kay", address:
//      Address(street: "21 Elm", city: "Herndon"))
```

Codable

Swift Archival and Serialization (SE-166)

- Adds protocols for
 - Encoders and decoders
 - Encodable and decodable types
 - Property keys
 - User info keys

Codable Protocols

- Compiler can synthesize default implementations

```
/// A type that can encode values into a native format  
/// for external representation.
```

```
public protocol Encodable {  
    public func encode(to encoder: Encoder) throws  
}
```

```
/// A type that can decode itself from an external representation.
```

```
public protocol Decodable {  
    public init(from decoder: Decoder) throws  
}
```

```
public typealias Codable = Decodable & Encodable
```

Standard Library Codable Types

- Optional
- Array, Dictionary
- String, Int, Double
- Date, Data, URL

Declaring Codable Types

```
// Declare Person and Dog structs conforming to Codable
struct Person: Codable {
    var name: String
    var age: Int
    var dog: Dog
}

struct Dog: Codable {
    var name: String
    var breed: Breed

    // Codable has built-in support for enums with raw values.
    enum Breed: String, Codable {
        case collie = "Collie"
        case beagle = "Beagle"
        case greatDane = "Great Dane"
    }
}
```

Swift Encoders (SE-167)

- Foundation framework classes are bridged across as Swift types

Swift Standard Library

Foundation

JSONEncoder

NSJSONSerialization

JSONDecoder

NSJSONSerialization

PropertyListEncoder

NSPropertyListSerialization

PropertyListDecoder

NSPropertyListSerialization

Encoding to JSON

```
let encoder = JSONEncoder()
encoder.outputFormatting = .prettyPrinted

let fred = Person(name: "Fred", age: 30, dog:
    Dog(name: "Spot", breed: .beagle))

let data = try! encoder.encode(fred)

{
    "name" : "Fred",
    "age" : 30,
    "dog" : {
        "name" : "Spot",
        "breed" : "Beagle"
    }
}
```

Encoding to JSON

```
let encoder = JSONEncoder()
encoder.outputFormatting = .prettyPrinted

let fred = Person(name: "Fred", age: 30, dog:
    Dog(name: "Spot", breed: .beagle))

let data = try! encoder.encode(fred)

{
  "name" : "Fred",
  "age" : 30,
  "dog" : {
    "name" : "Spot",
    "breed" : "Beagle"
  }
}
```

Encoding to JSON

```
let encoder = JSONEncoder()
encoder.outputFormatting = .prettyPrinted

let fred = Person(name: "Fred", age: 30, dog:
    Dog(name: "Spot", breed: .beagle))

let data = try! encoder.encode(fred)

{
  "name" : "Fred",
  "age" : 30,
  "dog" : {
    "name" : "Spot",
    "breed" : "Beagle"
  }
}
```

Decoding from JSON

```
let decoder = JSONDecoder()

let fredsClone = try! decoder.decode(Person.self, from: data)

// Person(name: "Fred",
//          age: 30,
//          dog: Dog(name: "Spot",
//                  breed: Dog.Breed.beagle))
```

Decoding from JSON

```
let decoder = JSONDecoder()  
  
let fredClone = try! decoder.decode(Person.self, from: data)  
  
// Person(name: "Fred",  
//         age: 30,  
//         dog: Dog(name: "Spot",  
//                 breed: Dog.Breed.beagle))
```



Codable Demo



Upcoming Classes

View online: [Public schedule](#)

Date	Title
Mar 12 – 14	Transitioning to Swift
Apr 14 – Apr 20	iOS Development in Swift: Comprehensive
Apr 30 – May 4	Advanced iOS Development

Q & A