What’s New in Swift

Session 402

Doug Gregor, Swift Compiler Team
Bob Wilson, Swift Performance Team

Ben Cohen, Swift Standard Library Team
John McCall, Swift Compiler Team
What's new in Swift 4

By Ole Begemann • May 2017

Table of contents

1. Instructions (see below)
2. One-sided ranges
3. Strings
4. Private declarations visible in same-file extensions
5. Key paths
6. Encoding and decoding
7. Dictionary and Set enhancements
8. MutableCollection.swapAt method
9. Generic subscripts
10. NSNumber bridging
11. Class and subtype existentials

Instructions

This playground requires Swift 4. To run it in Xcode 8.3 (before Xcode 9 becomes available):

1. Download the latest Swift snapshot from swift.org.
2. Install the snapshot.
3. In Xcode, go to Xcode > Toolchains > Manage Toolchains... and select the snapshot:
4. Private declarations visible in same-file extensions
5. Key paths
6. Encoding and decoding
7. Dictionary and Set enhancements
8. MutableCollection.swapAt method
9. Generic subscripts
10. NSNumbers bridging
11. Class and subtype existentials

Instructions

This playground requires Swift 4. To run it in Xcode 8.3 (before Xcode 9 becomes available):

1. Download the latest Swift snapshot from swift.org.
2. Install the snapshot.
3. In Xcode, go to Xcode > Toolchains > Manage Toolchains... and select the snapshot:
Swift is developed on GitHub
Open evolution process
Open Source

Swift is developed on GitHub

Open evolution process

Refactoring coming soon

• Can contribute new refactorings
• Toolchains can add refactorings to Xcode
Swift Package Manager

Growing ecosystem

• 7,000+ packages on GitHub
• Popular for server-side Swift
Swift Package Manager

Growing ecosystem
• 7,000+ packages on GitHub
• Popular for server-side Swift

Significant improvements in Swift 4
• New manifest API
• Better development workflow, diagnostics, dependency resolution
• Xcode project generation
Agenda

Language refinements and additions
Source compatibility
Tools and performance
Standard library
Exclusive access to memory
// "Private" Access Control

struct Date: Equatable, Comparable {

  private let secondsSinceReferenceDate: Double

  static func ==(lhs: Date, rhs: Date) -> Bool {
    return lhs.secondsSinceReferenceDate == rhs.secondsSinceReferenceDate
  }

  static func <(lhs: Date, rhs: Date) -> Bool {
    return lhs.secondsSinceReferenceDate < rhs.secondsSinceReferenceDate
  }
}
// "Private" Access Control

struct Date {
    private let secondsSinceReferenceDate: Double
}

extension Date: Equatable {
    static func ==(lhs: Date, rhs: Date) -> Bool {
        return lhs.secondsSinceReferenceDate == rhs.secondsSinceReferenceDate
    }
}

extension Date: Comparable {
    static func <(lhs: Date, rhs: Date) -> Bool {
        return lhs.secondsSinceReferenceDate < rhs.secondsSinceReferenceDate
    }
}
// "Private" Access Control

struct Date {
    private let secondsSinceReferenceDate: Double
}

extension Date: Equatable {
    static func ==(lhs: Date, rhs: Date) -> Bool {
        return lhs.secondsSinceReferenceDate == rhs.secondsSinceReferenceDate
    }
}

extension Date: Comparable {
    static func <(lhs: Date, rhs: Date) -> Bool {
        return lhs.secondsSinceReferenceDate < rhs.secondsSinceReferenceDate
    }
}
// "Private" Access Control

struct Date {
    fileprivate let secondsSinceReferenceDate: Double
}

date extension Date: Equatable {
    static func ==(lhs: Date, rhs: Date) -> Bool {
        return lhs.secondsSinceReferenceDate == rhs.secondsSinceReferenceDate
    }
}

date extension Date: Comparable {
    static func <(lhs: Date, rhs: Date) -> Bool {
        return lhs.secondsSinceReferenceDate < rhs.secondsSinceReferenceDate
    }
}
struct Date {
    private let secondsSinceReferenceDate: Double
}

extension Date: Equatable {
    static func ==(lhs: Date, rhs: Date) -> Bool {
        return lhs.secondsSinceReferenceDate == rhs.secondsSinceReferenceDate
    }
}

extension Date: Comparable {
    static func <(lhs: Date, rhs: Date) -> Bool {
        return lhs.secondsSinceReferenceDate < rhs.secondsSinceReferenceDate
    }
}
// Composing Classes and Protocols

protocol Shakeable {
    func shake()
}

extension UIButton: Shakeable { /* … */ }
extension UISlider: Shakeable { /* … */ }
// Composing Classes and Protocols

protocol Shakeable {
    func shake()
}

extension UIButton: Shakeable { /* … */ }
extension UISlider: Shakeable { /* … */ }

func shakeEm(controls: [] ???) {
    for control in controls where control.state.isEnabled {
        control.shake()
    }
}
// Composing Classes and Protocols

protocol Shakeable {
    func shake()
}

extension UIButton: Shakeable { /* … */ }
extension UISlider: Shakeable { /* … */ }
func shakeEm(controls: [UIControl]) {
    for control in controls where control.state.isEnabled {
        control.shake()
    }
}
// Composing Classes and Protocols

protocol Shakeable {
    func shake()
}

extension UIButton: Shakeable { /* … */ }
extension UISlider: Shakeable { /* … */ }

func shakeEm(controls: [UIControl]) {
    for control in controls where control.state.isEnabled {
        control.shake()
    }
}

error: value of type 'UIControl' has no member named 'shake'
// Composing Classes and Protocols

protocol Shakeable {
    func shake()
}

extension UIButton: Shakeable { /* ... */ }
extension UISlider: Shakeable { /* ... */ }

func shakeEm(controls: [Shakeable]) {
    for control in controls where control.state.isEnabled {
        control.shake()
    }
}

// Composing Classes and Protocols

protocol Shakeable {
    func shake()
}

extension UIButton: Shakeable { /* … */ }
extension UISlider: Shakeable { /* … */ }

func shakeEm(controls: [Shakeable]) {
    for control in controls where control.state.isEnabled {
        control.shake()
    }
}

error: value of type 'Shakeable' has no member named 'state'
// SE-0156: Class and Subtype Existentials

protocol Shakeable {
    func shake()
}
extension UIButton: Shakeable { /* … */ }
extension UISlider: Shakeable { /* … */ }
func shakeEm(controls: [UIControl & Shakeable]) {
    for control in controls where control.state.isEnabled {
        control.shake()
    }
}

NEW
Class and Protocol Composition in the SDK

// Objective-C API
@interface NSCandidateListTouchBarItem<CandidateType> : NSTouchBarItem
@property (nullable, weak) NSView <NSTextInputClient> *client;
@end
// Class and Protocol Composition in the SDK

// Objective-C API
@interface NSCandidateListTouchBarItem<CandidateType> : NSTouchBarItem
@property (nullable, weak) NSView <NSTextInputClient> *client;
@end

// Swift 3
class NSCandidateListTouchBarItem<CandidateType: AnyObject> : NSTouchBarItem {
    var client: NSView?
}

// Class and Protocol Composition in the SDK

// Objective-C API
@interface NSCandidateListTouchBarItem<CandidateType> : NSTouchBarItem
@property (nullable, weak) NSView <NSTextInputClient> *client;
@end

// Swift 4
class NSCandidateListTouchBarItem<CandidateType: AnyObject> : NSTouchBarItem {
    var client: (NSView & NSTextInputClient?)
}

NEW
Swift 4—Improving Cocoa Idioms

SE-0161 Smart KeyPaths: Better Key-Value Coding for Swift
SE-0166 Swift Archival & Serialization
SE-0167 Swift Encoders
Source Compatibility
Swift 4

Swift 4 largely source-compatible with Swift 3
- Refinements
- SDK improvements

Additive features extend existing syntax
Swift 3.2

Compilation mode of the Swift 4 compiler
• Not a separate toolchain!

Emulates Swift 3
• Allows Swift 3 syntax that has changed in Swift 4
• “Rolls back” SDK changes
Swift 3.2

Compilation mode of the Swift 4 compiler

• Not a separate toolchain!

Emulates Swift 3

• Allows Swift 3 syntax that has changed in Swift 4
• “Rolls back” SDK changes

Most Swift 3 code should compile unmodified
Swift 3.2

Compilation mode of the Swift 4 compiler

• Not a separate toolchain!

Emulates Swift 3

• Allows Swift 3 syntax that has changed in Swift 4
• "Rolls back" SDK changes

Most Swift 3 code should compile unmodified

Provides most Swift 4 features and new SDKs
Migrating from Swift 3.2 to Swift 4

Select targets and playgrounds to convert:

- Kickstarter.app
- Kickstarter_Framework.framework
- Library.framework
- LiveStream.framework
- Kickstarter-iOS.playground

Options:
- Cancel
- Previous
- Next
Swift 3.2 and Swift 4 Coexistence

Swift language is set per-target

Migrate one target at a time
- No need to migrate in dependency order
- Your dependencies can migrate asynchronously
Swift package manager will pick the appropriate Swift version for each package.

```swift
let package = Package(
    name: "HTTP",
    ...
    swiftLanguageVersions: [3, 4])
```
Build Improvements

Bob Wilson, Swift Performance Team
New Build System

Xcode 9 includes a preview of a new build system

Fast: lower overhead, especially for large projects
New Build System

Xcode 9 includes a preview of a new build system

Fast: lower overhead, especially for large projects
Precompiled Bridging Headers
Bridging header for large mixed-source projects can be slow to parse
Precompiled Bridging Headers

Bridging header for large mixed-source projects can be slow to parse.
Precompiled Bridging Headers
Bridging header for large mixed-source projects can be slow to parse

Speeds up Apple’s Music app build by 40%
Shared Build for Coverage Testing
Shared Build for Coverage Testing

Xcode 8 builds separately for coverage testing
Shared Build for Coverage Testing

Xcode 8 builds separately for coverage testing
Shared Build for Coverage Testing

Xcode 8 builds separately for coverage testing
Shared Build for Coverage Testing

Xcode 8 builds separately for coverage testing.

Coverage instrumentation is very low overhead.
Shared Build for Coverage Testing

Xcode 8 builds separately for coverage testing

Coverage instrumentation is very low overhead

Xcode 9 shares the same build → avoid building twice
Indexing While Building
Indexing While Building
Indexing While Building

Background indexing often duplicates effort
Build process now updates the index
More accurate results
Predictable Performance
// Unpredictable Performance in Swift 3

protocol Ordered {
    func precedes(_ other: Ordered) -> Bool
}

func testSort(_ values: inout [Ordered]) {
    values.sort { $0.precedes($1) }
}
Unpredictable Performance in Swift 3
Unpredictable Performance in Swift 3
Unpredictable Performance in Swift 3
Unpredictable Performance in Swift 3
Unpredictable Performance in Swift 3

Sort Time vs Struct Size

- Sort Time:
  - Y-axis values: 0, 5, 10, 15, 20

- Struct Size:
  - X-axis values: 1, 2, 3, 4

The graph shows an unexpected increase in sort time as the struct size increases from 3 to 4.
Existential Containers

Understanding Swift Performance

WWDC 2016
Existential Containers

Implementation of a value of unknown type
Existential Containers

Implementation of a value of unknown type

Inline value buffer: currently three words
Existential Containers

Implementation of a value of unknown type

Inline value buffer: currently three words

Large values stored on the heap
Existential Containers

Implementation of a value of unknown type

Inline value buffer: currently three words

Large values stored on the heap

Heap allocation is slow
Why am I here?
COW Existential Buffers

Swift now uses copy-on-write (COW) reference-counted existential buffers.

Copied only when modified while not uniquely referenced.

Avoids expensive heap allocations.
COW Existential Buffers

Graph showing the relationship between sort time and struct size.
Faster Generic Code

Specialization: compiler generates code for specific types

Not always possible: unspecialized generic code is also important

Stack allocation of generic buffers
Smaller Binaries
// Unused Conformance Removal

struct Date {
    private let secondsSinceReferenceDate: Double
}
extension Date: Equatable {
    static func ==(lhs: Date, rhs: Date) -> Bool {
        return lhs.secondsSinceReferenceDate == rhs.secondsSinceReferenceDate
    }
}
extension Date: Comparable {
    static func <(lhs: Date, rhs: Date) -> Bool {
        return lhs.secondsSinceReferenceDate < rhs.secondsSinceReferenceDate
    }
}
// Unused Conformance Removal

struct Date {
    private let secondsSinceReferenceDate: Double
}

extension Date: Equatable {
    static func ==(lhs: Date, rhs: Date) -> Bool {
        return lhs.secondsSinceReferenceDate == rhs.secondsSinceReferenceDate
    }
}
class MyClass: NSObject {
    func print() { ... }
    func show() { print() ... }
}

Unused @objc Thunks
class MyClass: NSObject {
    func print() { ... }
    func show() { print() ... }
}
Unused @objc Thunks

class MyClass: NSObject {
    @objc func print() { ... }
    @objc func show() { print() ... }
}

Swift 3 automatically infers @objc
class MyClass: NSObject {
    @objc func print() { ... }
    @objc func show() { print() ... }
}

Swift 3 automatically infers @objc

print() ←-[MyClass print]
show() ←-[MyClass show]
Unused @objc Thunks

Swift 3 automatically infers @objc

class MyClass: NSObject {
    @objc func print() {
        ...
    }
    @objc func show() {
        print()
        ...
    }
}

print() - [MyClass print]
show() - [MyClass show]
Unused @objc Thunks

Swift 3 automatically infers @objc

Objective-C thunks are often unused
Swift 4 only infers @objc when it is needed
• Overriding an Objective-C method
• Conforming to an Objective-C protocol

Reduced size of Apple’s Music app by 5.7%
SE-0160: Limited @objc Inference

Use @objc on extension with a group of functions

```swift
@objc extension MyClass {
    func f(_: String?) { ... }
    func g(_: Int?) { ... }
}
```

Compiler will report errors for anything not expressible in Objective-C
SE-0160: Limited @objc Inference

Use @objc on extension with a group of functions

```swift
@objc extension MyClass {
    func f(_: String?) { ... }
    func g(_: Int?) { ... }
}
```

error: method cannot be in an @objc extension of a class (without @nonobjc) because the type of the parameter cannot be represented in Objective-C

Compiler will report errors for anything not expressible in Objective-C
Migration for Limited `@objc` Inference

Swift 4 `@objc` Inference:
- Minimize Inference (recommended)
- Match Swift 3 Behavior

Built product will have reduced binary size.
Migration for Minimal Inference

Migrator cannot identify all the functions that need `@objc`

Inferred Objective-C thunks marked as deprecated to help you find them

- Build warnings about deprecated methods
- Console messages when running deprecated thunks
Manually add @objc to fix build warnings

```swift
[vc showStatus];
}
warning: Swift method 'ViewController.showStatus' uses '@objc' inference deprecated in Swift 4; add '@objc' to provide an Objective-C entrypoint
```
Build Warnings

Manually add @objc to fix build warnings

```swift
[vc showStatus];
}

```warning: Swift method 'ViewController.showStatus' uses '@objc' inference deprecated in Swift 4; add '@objc' to provide an Objective-C entrypoint```

```swift
func showStatus() {
    print("ViewController status:")
    if let name = title {
        print(" \(name)"")
    }
```
Manually add `@objc` to fix build warnings

```swift
[vc showStatus];
}

warning: Swift method 'ViewController.showStatus' uses '@objc' inference deprecated in Swift 4; add '@objc' to provide an Objective-C entrypoint

@objc func showStatus() {
    print("ViewController status:")
    if let name = title {
        print(" \(name)"")
    }
```
Run your code, including all your tests, and fix issues logged to the console.

2017-05-26 10:00:01.531842-0700 MyApp[48356:20427473] *** /Users/bwilson/Desktop/MyApp/MyApp/ViewController.swift:26:5: implicit Objective-C entrypoint -[MyApp.ViewController showStatus] is deprecated and will be removed in Swift 4; add explicit '@objc' to the declaration to emit the Objective-C entrypoint in Swift 4 and suppress this message.
Run your code, including all your tests, and fix issues logged to the console.

2017-05-26 10:00:01.531842-0700 MyApp[48356:20427473] *** /Users/bwilson/Desktop/MyApp/MyApp/ViewController.swift:26:5: implicit Objective-C entrypoint -[MyApp.ViewController showStatus] is deprecated and will be removed in Swift 4; add explicit '@objc' to the declaration to emit the Objective-C entrypoint in Swift 4 and suppress this message.
Finish Migration

Change build setting to Default

Apple’s Music app migration: only a handful of manual changes required
Symbol Size

Swift symbols take up a lot of space

Example: macOS libswiftCore library
Symbol Size

Swift symbols take up a lot of space

Example: macOS libswiftCore library
Symbol Size

Swift symbols take up a lot of space

Example: macOS libswiftCore library
Symbol Stripping

Linkers use a separate trie structure to find symbols.

Swift symbols are rarely needed in the symbol table.

New build setting enabled by default.

<table>
<thead>
<tr>
<th>Strip Style</th>
<th>All Symbols</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strip Swift Symbols</td>
<td>Yes</td>
</tr>
<tr>
<td>Targeted Device Family</td>
<td>1,2</td>
</tr>
</tbody>
</table>

View symbols with "xcrun dyldinfo -export" instead of nm.
Symbol Stripping

Swift standard libraries are stripped during App Thinning.

New option when exporting project archive.

- Strip Swift symbols
  Reduce app size by stripping symbols from Swift standard libraries.
Swift Strings
Faster, easier character processing

Ben Cohen, Swift Standard Library Team
public typealias CChar = Int8
$e + \prime$
# Ruby

```ruby
one = "\u{E9}"

two = "\u{65}\u{301}"
```

é

é
# Ruby

one = "\u{E9}"

two = "\u{65}\u{301}"

one.length

two.length

one == two
Unicode Correctness by Default

In Swift, a Character is a grapheme
Unicode Correctness by Default

In Swift, a Character is a grapheme

twoCodeUnits.count
oneCodeUnit == twoCodeUnits

1
true
// Grapheme Breaking
// Grapheme Breaking

var family = "👩"
family += "\u{200D}👩"
family += "\u{200D}👧"
family += "\u{200D}👦"

print(family)

family.count
Faster Character Processing
Benchmark for Latin-derived characters, Han ideographs, and Kana
// Graphemes can be of arbitrary length

var family = "👩"
family += "\u{200D}👩"
family += "\u{200D}👧"
family += "\u{200D}👦"

print(family)
family.count
// Swift 3 strings had a collection of characters

let values = "one,two,three..."

var i = values.characters.startIndex
while let comma = values.characters[i..<values.characters.endIndex].index(of: ",") {
    if values.characters[i..<comma] == "two" {
        print("found it!")
    }
    i = values.characters.index(after: comma)
}
let values = "one,two,three..."

var i = values.characters.startIndex
while let comma = values.characters[i..<values.characters.endIndex].index(of: "," ) {
    if values.characters[i..<comma] == "two" {
        print("found it!")
    }
    i = values.characters.index(after: comma)
}
// Swift 3 strings had a collection of characters

let values = "one,two,three...

var i = values.startIndex
while let comma = values[i..<values.endIndex].index(of: ",") {
    if values[i..<comma] == "two" {
        print("found it!")
    }
    i = values.index(after: comma)
}
// Swift 4 strings are a collection of characters

let values = "one,two,three...

var i = values.startIndex
while let comma = values[i..<values.endIndex].index(of: ",") {
    if values[i..<comma] == "two" {
        print("found it!")
    }
    i = values.index(after: comma)
}
// SE-0172: Simpler One-Sided Slicing Syntax

let values = "one,two,three..."

var i = values.startIndex
while let comma = values[i..<values.endIndex].index(of: ",") {
    if values[i..<comma] == "two" {
        print("found it!")
    }
    i = values.index(after: comma)
}
let values = "one,two,three..."

var i = values.startIndex
while let comma = values[i...].index(of: ",") {
    if values[i..<comma] == "two" {
        print("found it!")
    }
    i = values.index(after: comma)
}
// Using String as a Collection

let asciiTable = zip(65..., "ABCDEFGHIJKLMNOPQRSTUVWXYZ")
for (code, character) in asciiTable {
    print(code, character, separator: "::")
}

// Using String as a Collection

"Good luck 🇯🇵 in the game tonight!"
// Using String as a Collection

"Good luck 🇯 in the game tonight!"

extension Unicode.Scalar {
    var isRegionalIndicator: Bool {
        return ("🇦"..."🇿").contains(self)
    }
}

// Using String as a Collection

"Good luck 🇯🇵 in the game tonight!"

extension Character {
    var isFlag: Bool {
        let scalars = self.unicodeScalars
        return scalars.count == 2 && scalars.first!.isRegionalIndicator && scalars.last!.isRegionalIndicator
    }
}
// Using String as a Collection

let message = "Looking forward to 🇧🇷 vs 🇺🇸 game!"

message.contains { $0.isFlag }  // true

let flags = message.filter { $0.isFlag }

flags.count  // 2
let s = "one,two,three"

s.split(separator: ",")
// SE-0163: String Slicing

let s = "one,two,three"

s.split(separator: ",")

["one","two","three"]: [String]
let s = "one,two,three"
s.split(separator: ",")

["one","two","three"]: [Substring]
Swift Strings Have Three Properties

"Hello, world!"

- Start
- Count = 13
- Owner

Owning class: Capacity, reference count

3 words
Owner Reference Count Drops to Zero
Owner is freed, frees the string buffer

"Hello, world!"

Start
Count = 13
Owner

Owning class
Capacity, reference count
Owner Reference Count Drops to Zero
Owner is freed, frees the string buffer

Hello, world!
Owner Reference Count Drops to Zero
Owner is freed, frees the string buffer
Creating a Substring

"Hello, world!"

Start
Count = 13
Owner

Owning class
Capacity, reference count
Creating a Substring

"Hello, world!"

- **Start**
- **Count = 13**
- **Owner**

Owning class
Capacity, reference count
Creating a Substring

Owning class
Capacity, reference count
Start
Count = 13
Owner

"Hello, world!"
Start
Count = 5
Owner

"world"
Original String Goes Out of Scope
Owner still referenced, buffer remains

"Hello, world!"
Start
Count = 13
Owner

Hello, world!

"world"
Start
Count = 5
Owner

Owning class
Capacity, reference count
Original String Goes Out of Scope
Owner still referenced, buffer remains

Hello, world!

Start
Count = 5
Owner

Owning class
Capacity, reference count

"world"
// Substrings can waste memory

let big = downloadHugeString()
let small = extractTinyString(from: big)

mainView.titleLabel.text = small
// Substrings can waste memory

let big = downloadHugeString()
let small = extractTinyString(from: big)

mainView.titleLabel.text = small
// Substrings can waste memory

let big = downloadHugeString()
let small = extractTinyString(from: big)

mainView.titleLabel.text = small

error: cannot assign value of type 'Substring' to type 'String'
// String(_:Substring) Copies the Buffer

let big = downloadHugeString()
let small = extractTinyString(from: big)

mainView.titleLabel.text = small
// String(_:Subsequence) copies the buffer

let big = downloadHugeString()
let small = extractTinyString(from: big)

mainView.titleLabel.text = String(small)
// Substring and type inference

let keyAndValue = setting.split(":")

if keyAndValue.first == "animation", let value = keyAndValue.last
    view.animate = value == "on" ? true : false
}
func tellJoke(name: String, character: Character) {
    let punchline = name.filter { $0 != character }
    let n = name.count - punchline.count

    let joke = "Q: Why does \(name) have \(n) \(character)'s in their name?\nA: I don't know, why does \(name) have \(n) \(character)'s in their name?\nQ: Because otherwise they'd be called \(punchline)."
    print(joke)
}

tellJoke(name: "Edward Woodward", character: "d")
func tellJoke(name: String, character: Character) {
    let punchline = name.filter { $0 != character }
    let n = name.count - punchline.count

    let joke = """"""""""""""
        Q: Why does \(name) have \(n) \(character)'s in their name?
        A: I don't know, why does \(name) have \(n) \(character)'s in their name?
        Q: Because otherwise they'd be called \(punchline).
    """"""

    print(joke)
}

tellJoke(name: "Edward Woodward", character: "d")
// SE-0168: Multi-line String Literals

func tellJoke(name: String, character: Character) {
    let punchline = name.filter { $0 != character }
    let n = name.count - punchline.count

    let joke = """Q: Why does \(name) have \(n) \(character)'s in their name?
A: I don't know, why does \(name) have \(n) \(character)'s in their name?
Q: Because otherwise they'd be called \(punchline)."
    
    print(joke)
}

tellJoke(name: "Edward Woodward", character: "d")
func tellJoke(name: String, character: Character) {
    let punchline = name.filter { $0 != character }
    let n = name.count - punchline.count

    let joke = ""
    
    Q: Why does (name) have (n) (character)'s in their name?
    A: I don't know, why does (name) have (n) (character)'s in their name?
    Q: Because otherwise they'd be called (punchline).
    ""

    print(joke)
}

tellJoke(name: "Edward Woodward", character: "d")
Q: Why does \(name) have \(n) \(character)'s in their name?
A: I don't know, why does \(name) have \(n) \(character)'s in their name?
Q: Because otherwise they'd be called \(punchline).
Q: Why does \(\text{name}\) have \(\text{n}\) \(\text{character}\)'s in their name?
A: I don't know, why does \(\text{name}\) have \(\text{n}\) \(\text{character}\)'s in their name?
Q: Because otherwise they'd be called \(\text{punchline}\).
New Generics Features
Extending Sequence
// Extending Sequence

!contains { $0 != value }
extension Sequence
where Iterator.Element: Equatable {
    func containsOnly(_ value: (Iterator.Element)->Bool) -> Bool {
        return !contains { $0 != value }
    }
}

mySequence.containsOnly(5)
// Extending Sequence

extension Sequence
where Iterator.Element: Equatable {
    func containsOnly(_ value: (Iterator.Element)->Bool) -> Bool {
        return !contains { $0 != value }
    }
}

mySequence.containsOnly(5)
// Extending Sequence

extension Sequence
where Element: Equatable {
    func containsOnly(_ value: (Element) -> Bool) -> Bool {
        return
    }
}
// Swift 3 Sequence

protocol Sequence {
    associatedtype Iterator: IteratorProtocol

    func makeIterator() -> Iterator
}

protocol IteratorProtocol {
    associatedtype Element

    mutating func next() -> Element?
}
/ Swift 3 Sequence

protocol Sequence {

    associatedtype Iterator: IteratorProtocol

    func makeIterator() -> Iterator

}

protocol IteratorProtocol {

    associatedtype Element

    mutating func next() -> Element?

}
// SE-0142: Swift 4 Sequence

protocol Sequence {
    associatedtype Element
    associatedtype Iterator: IteratorProtocol where Iterator.Element == Element

    func makeIterator() -> Iterator
}

protocol IteratorProtocol {
    associatedtype Element

    mutating func next() -> Element?
}
/ SE-0142: Swift 4 Sequence

protocol Sequence {
    associatedtype Element
    associatedtype Iterator: IteratorProtocol where Iterator.Element == Element

    func makeIterator() -> Iterator
}

protocol IteratorProtocol {
    associatedtype Element

    mutating func next() -> Element?
}
// SE-0142: Swift 4 Sequence

protocol Sequence {
    associatedtype Element
    associatedtype Iterator: IteratorProtocol where Iterator.Element == Element

    func makeIterator() -> Iterator
}

protocol IteratorProtocol {
    associatedtype Element

    mutating func next() -> Element?
}
// Other important constraints

protocol Sequence {
    associatedtype SubSequence: Sequence
}
// Other important constraints

protocol Sequence {
    associatedtype SubSequence: Sequence
    where SubSequence.SubSequence == SubSequence,
        SubSequence.Element == Element
}
// Redundant constraints

extension Collection

where Element: Equatable,
    SubSequence: Collection,
    SubSequence.SubSequence == SubSequence,
    SubSequence.Element == Element {

    func containsOnly(_ x: Element) -> Bool {
        return isEmpty
        || (first == x && dropFirst().containsOnly(x))
    }

}
// Redundant constraints

extension Collection
where Element: Equatable,
    SubSequence: Collection,
    SubSequence.SubSequence == SubSequence,  
    SubSequence.Element == Element {
    warning: redundant same-type constraint 'Self.Element' == 'Self.SubSequence.Element'
    return isEmpty 
        || (first == x && dropFirst().containsOnly(x)) 
    }
}
// Redundant constraints

extension Collection
where Element: Equatable,
    SubSequence: Collection {

    func containsOnly(_ x: Element) -> Bool {
        return isEmpty || (first == x && dropFirst().containsOnly(x))
    }
}

let values = "one,two,three..."

var i = values.startIndex
while let comma = values[i...].index(of: ",") {
    if values[i..<comma] == "two" {
        print("found it!")
    }
    i = values.index(after: comma)
}
let values = "one,two,three..."

var i = values.startIndex
while let comma = values[i...].index(of: ",") {
    if values[i..<comma] == "two" {
        print("found it!")
    }
    i = values.index(after: comma)
}
// RangeExpression

struct PartialRangeFrom<Bound: Comparable> {
    let lowerBound: Bound
}

// RangeExpression

protocol RangeExpression {
    func relative<C: Collection>(to collection: C) -> Range<Bound>
    where C.Index == Bound
}
// RangeExpression

extension PartialRangeFrom: RangeExpression {  
    func relative<C: Collection>(to collection: C) -> Range<Bound>  
    where C.Index == Bound {  
        return lowerBound..<collection.endIndex
    }
}
extension PartialRangeFrom: RangeExpression {
    func relative<C: Collection>(to collection: C) -> Range<Bound>
    where C.Index == Bound {
        return lowerBound..<collection.endIndex
    }
}
extension String {
  subscript<R: RangeExpression>(range: R) -> Substring where R.Bound == Index {
    return self[range.relative(to: self)]
  }
}
// SE-0148: Generic Subscripts

extension Collection {
    subscript<\texttt{R: RangeExpression}>(\texttt{range: R}) \to \texttt{SubSequence} \texttt{where} \ R.\texttt{Bound} == \texttt{Index} { 
        \texttt{return self[range.relative(to: self)]} 
    }
}

More Standard Library Features

SE-0104 Protocol-oriented integers
SE-0153 Dictionary & Set enhancements
SE-0163 Improved String C Interop and Transcoding
SE-0170 NSNumber bridging and Numeric types
SE-0173 Add MutableCollection.swapAt(_::_)
SE-0174 Change filter to return Self for RangeReplaceableCollection
More Standard Library Features

SE-0104 Protocol-oriented integers

SE-0153 Dictionary & Set enhancements

SE-0163 Improved String C Interop and Transcoding

SE-0170 NSNumber bridging and Numeric types

SE-0173 Add MutableCollection.swapAt(_::_)

SE-0174 Change filter to return Self for RangeReplaceableCollection
Exclusive Access to Memory

John McCall, Swift Compiler Team
Ownership

Make it easier to reason about local variables
Enable better programmer optimization
Enable better compiler optimization
Enable powerful new language features
var numbers = [1, 2, 3]
for index in numbers.indices {
    numbers[index] *= 2
}
// numbers == [2, 4, 6]
extension MutableCollection {
    mutating func modifyEach(_ body : (inout Element) -> ()) {
        for index in self.indices {
            body(&self[index])
        }
    }
}

var numbers = [1, 2, 3]
números.modifyEach { element in
    element *= 2
}
// numbers == [2, 4, 6]
extension MutableCollection {
    mutating func modifyEach(_ body : (inout Element) -> ()) {
        for index in self.indices {
            body(&self[index])
        }
    }
}
extension MutableCollection {
    mutating func modifyEach(_ body: (inout Element) -> ()) {
        for index in self.indices {
            body(&self[index])
        }
    }
}
extension MutableCollection {
    mutating func modifyEach(_ body : (inout Element) -> ()) {
        for index in self.indices {
            body(&self[index])
        }
    }
}
```swift
var numbers = [1, 2, 3]
numbers.modifyEach { element in
    element *= 2
}
// numbers == [2, 4, 6]
```
```swift
var numbers = [1, 2, 3]
numbers.modifyEach { element in
    element *= 2
    numbers.removeLast()
}
// numbers == ???
```
extension MutableCollection {
    mutating func modifyEach(_ body : (inout Element) -> ()) {
        for index in self.indices {
            body(&self[index])
        }
    }
}
extension MutableCollection {
    mutating func modifyEach(_ body : (inout Element) -> ()) {
        var index = self.beginIndex
        while index != self.endIndex {
            body(&self[index])
            self.formIndex(after: &index)
        }
    }
}
```swift
var numbers = [1, 2, 3]
numbers.modifyEach { element in
    element *= 2
    numbers.removeLast()
}
// numbers == ???
```
var numbers = [1, 2, 3]

numbers.modifyEach { element in
    numbers = []
    element *= 2
}
var numbers = [1, 2, 3]
numbers.modifyEach { element in
    numbers = []
    element *= 2
}
```swift
var numbers = [1, 2, 3]
numbers.modifyEach { element in
    numbers = []
    element *= 2
}
```
```swift
var numbers = [1, 2, 3]
numbers.modifyEach { element in
    numbers = []
    element *= 2
}
```
```swift
var numbers = [1, 2, 3]
numbers.modifyEach { element in
    numbers = []
    element *= 2
}
```
```swift
var numbers = [1, 2, 3]
numbers.modifyEach { element in
    numbers = []
    element *= 2
}
```
var numbers = [1, 2, 3]

numbers.modifyEach { element in
    numbers = []
    element *= 2
}
Non-Exclusive Access to Memory

Hard to reason about
Creates corner cases
Performance problems for libraries
Performance problems for the compiler
SE-0176: Enforcing Exclusive Access to Memory

- Read + Read: Allowed
- Read + Write: Denied
- Write + Write: Denied
var numbers = [1, 2, 3]
numbers.modifyEach { element in
    element *= 2
    numbers.removeLast()
}
// numbers == ???
```swift
var numbers = [1, 2, 3]

numbers.modifyEach { element in
    element *= 2
    numbers.removeLast()
}

// numbers == ???
```
var numbers = [1, 2, 3]

numbers.modifyEach { element in
  element *= 2
  numbers.removeLast()
}

// numbers == ???

error: simultaneous accesses, but initialization requires exclusive access
Run-time Enforcement

Global variables

Properties of classes

Local variables captured in escaping closures
var numbers = [1, 2, 3]
numbers.modifyEach { element in
    element *= 2
    numbers.removeLast()
}
var numbers = [1, 2, 3]

    numbers.modifyEach { element in
        element *= 2
        numbers.removeLast()
    }
class MyNumbers {
    var numbers = [1, 2, 3]
    func double(other: MyNumbers) {
        other.numbers.modifyEach { element in
            element *= 2
        }
        self.numbers.removeLast()
    }
}
class MyNumbers {
    var numbers = [1, 2, 3]
    func double(other: MyNumbers) {
        other.numbers.modifyEach { element in
            element *= 2
            self.numbers.removeLast()
        }
    }
}
class MyNumbers {
    var numbers = [1, 2, 3]
    func double(other: MyNumbers) {
        other.numbers.modifyEach { element in
            element *= 2
        }
        self.numbers.removeLast()
    }
}
class MyNumbers {
    var numbers = [1, 2, 3]
    func double(other: MyNumbers) {
        other.numbers.modifyEach { element in
            element *= 2
        }
        self.numbers.removeLast()
    }
}

Simultaneous accesses to 0x1105ac070, but modifications require exclusive access. Fatal access conflict detected.
Multi-threaded Enforcement

Default enforcement only catches single-threaded bugs

Thread Sanitizer catches multi-threaded bugs
Swift 3 Compatibility

Just a warning in Swift 3.2

Will be an error in later releases
Taking Advantage of Exclusive Access

More reliable performance

Lots of optimization

• In libraries
• In the compiler
• In your code

New language opportunities

https://github.com/apple/swift/blob/master/docs/OwnershipManifesto.md
Enforcement in the Developer Preview

Read the release notes

Compile-time enforcement:
• Enabled by default

Run-time enforcement:
• Disabled by default
• Off in optimized builds
What's New in Swift

- Language refinements and additions
- Source compatibility
- Tools and performance
- Standard library
- Exclusive access to memory
## Related Sessions

<table>
<thead>
<tr>
<th>Session</th>
<th>Location</th>
<th>Date/Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>What's New In Foundation</td>
<td>Hall 2</td>
<td>Wednesday 11:00AM</td>
</tr>
<tr>
<td>Finding Bugs Using Xcode Runtime Tools</td>
<td>Executive Ballroom</td>
<td>Wednesday 5:10PM</td>
</tr>
<tr>
<td>What's New in Swift Playgrounds</td>
<td>Hall 3</td>
<td>Thursday 10:00AM</td>
</tr>
<tr>
<td>Understanding Undefined Behavior</td>
<td>Grand Ballroom B</td>
<td>Thursday 9:00AM</td>
</tr>
<tr>
<td>What's New in LLVM</td>
<td>Hall 2</td>
<td>Thursday 4:10PM</td>
</tr>
<tr>
<td>Efficient Interactions with Frameworks</td>
<td>Hall 2</td>
<td>Friday 1:50PM</td>
</tr>
<tr>
<td>Understanding Swift Performance</td>
<td></td>
<td>WWDC 2016</td>
</tr>
<tr>
<td>Labs</td>
<td>Technology Lab E</td>
<td>Swift Open Hours</td>
</tr>
<tr>
<td>--------------------------</td>
<td>------------------</td>
<td>------------------</td>
</tr>
<tr>
<td>Swift Open Hours</td>
<td>Technology Lab E</td>
<td>Swift Open Hours</td>
</tr>
<tr>
<td>Swift Open Hours</td>
<td>Technology Lab D</td>
<td>Swift Open Hours</td>
</tr>
</tbody>
</table>