WHAT'S NEW IN SWIFT 5

ADAM MIKA

SWIFT 5

- Xcode 10.2
- Part of the upcoming OS updates
  - iOS 12.2
  - tvOS 12.2
  - watchOS 5.2
  - macOS 10.14.4

GETTING STARTED

How you can use Swift 5 today

GETTING STARTED

Two options:

» swift.org Snapshots

» Xcode 10.2 Beta Builds

SWIFT.ORG SNAPSHOTS

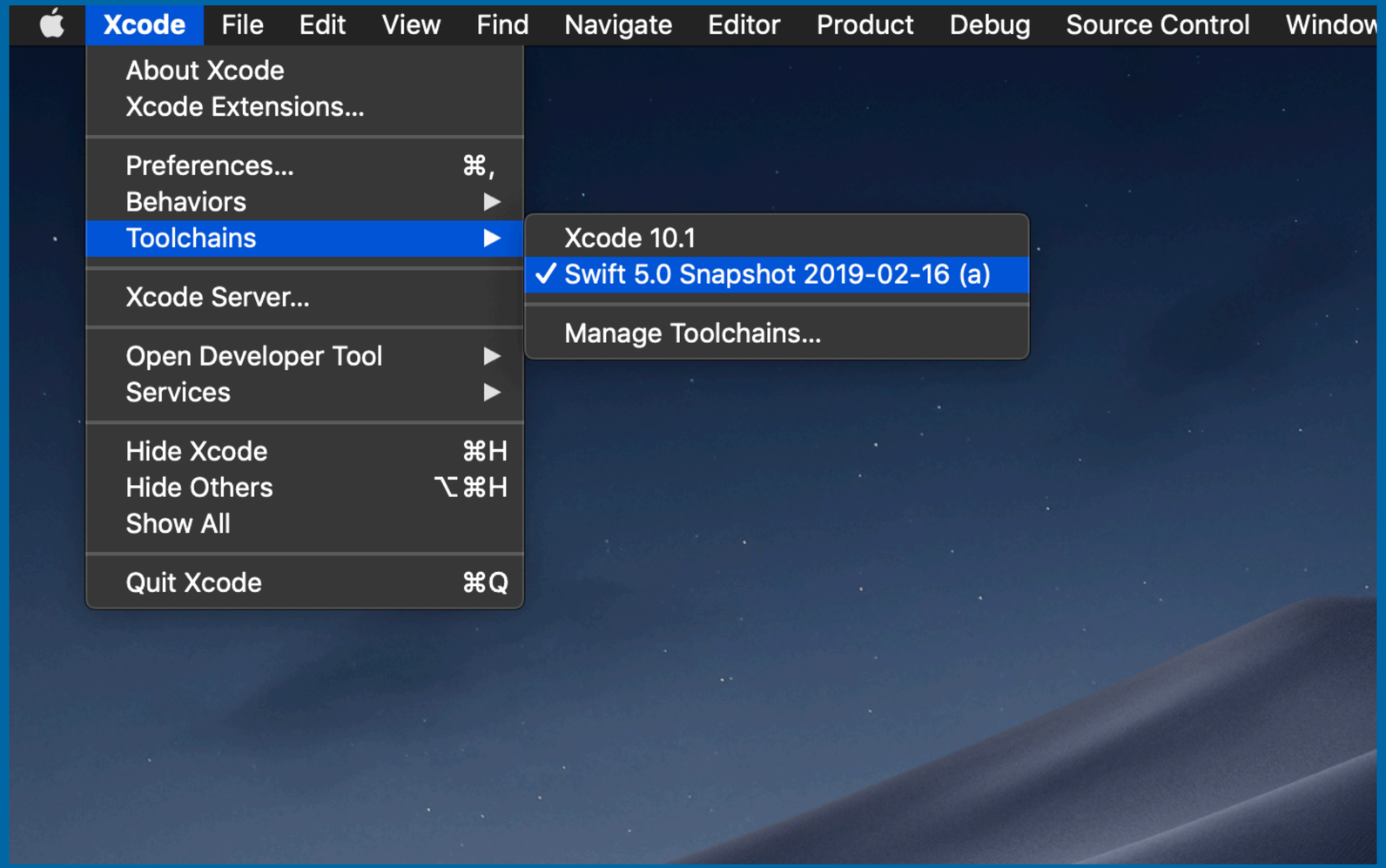
Swift 5.0 Development

Swift 5.0 Snapshots are prebuilt binaries that are automatically created from `swift-5.0-branch` branch. These snapshots are not official releases. They have gone through automated unit testing, but they have not gone through the full testing that is performed for official releases.

Download	Date
Xcode (Debugging Symbols)	February 15, 2019
Ubuntu 18.04 (Signature)	February 15, 2019
Ubuntu 16.04 (Signature)	February 15, 2019
Ubuntu 14.04 (Signature)	February 16, 2019

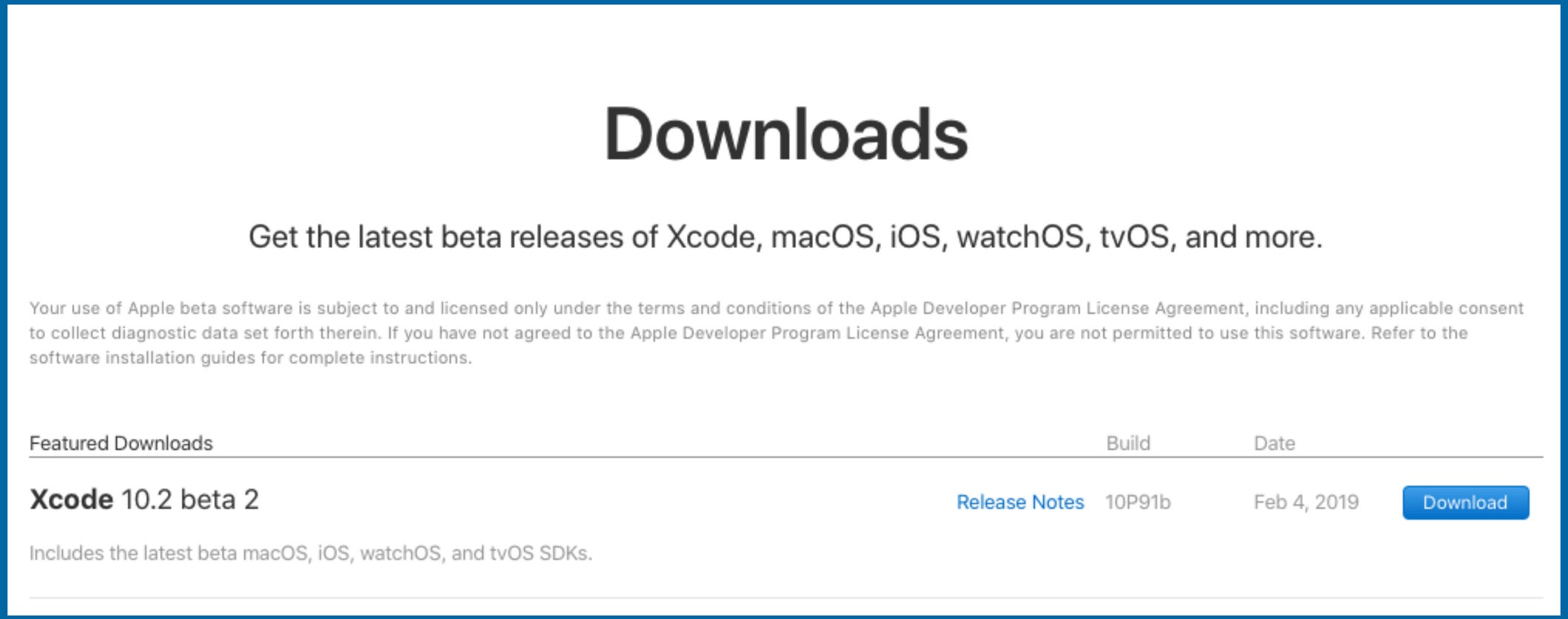
https://swift.org/download/#snapshots

SWIFT.ORG SNAPSHOTS



SWIFT.ORG SNAPSHOTS

XCODE 10.2 BETA 2



https://developer.apple.com/download/

SO WHAT'S NEW?

ABI STABILITY

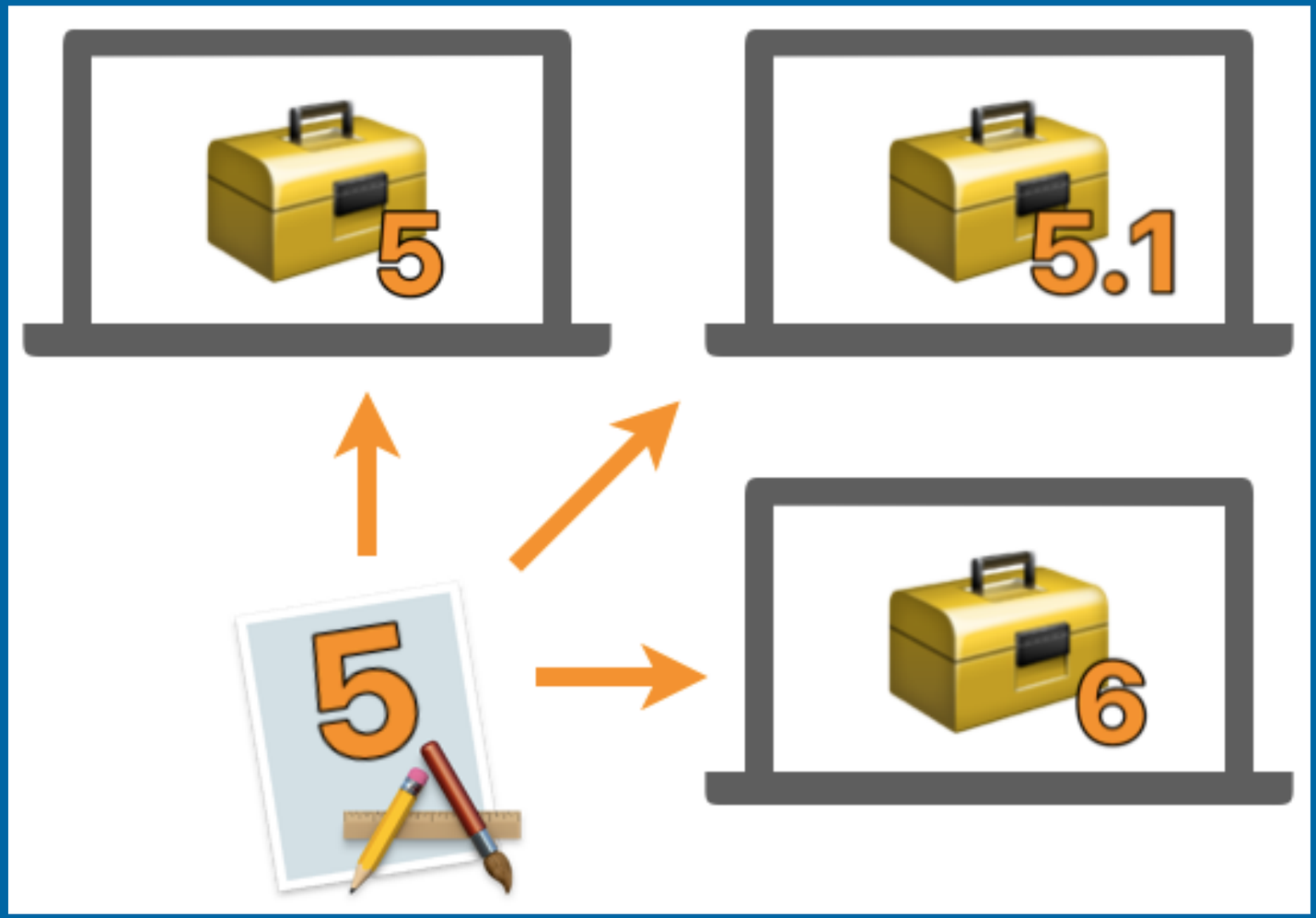
ABI STABILITY

- Binary compatability for apps and libraries on Apple platforms
  - macOS, iOS, watchOS, tvOS

ABI STABILITY

Guarantee that going forward, an app built with one version of the Swift compiler will be able to talk to a library built with another version

ABI STABILITY



ABI STABILITY

» Because of ABI stability, you no longer need to bundle the Swift libraries with your apps!

» Space savings of anywhere between 5-15 MB

RAW STRINGS

RAW STRINGS

Swift 5 adds the ability to create raw strings, where backslashes and quote marks are interpreted as those literal symbols rather than escapes characters or string terminators

RAW STRINGS

Place on or more # symobols before and after your strings

EXAMPLES

```
let quote = #"Quotation marks can be "used" within a string."#

let backslash = #"I can use \backslashes within the string."#

let value = 42
let answer = #"The answer to life, the universe, and everything is \#(value)"#

let tweet = ##"Swift 5 is "cool"#iosdeveloper"##
```

WHERE THIS CAN BE USEFUL

```
let regex1 = "\\\\[A-Z]+[A-Za-z]+\\.[a-z]+"

let regex2 = #"\\[A-Z]+[A-Za-z]+\.[a-z]+"#
```

STANDARD RESULT TYPE

STANDARD RESULT TYPE

Swift 5 adds a Result type into the standard library

Gives us a simpler, clearer way of handling errors in complex code such as asynchronous APIs.

WHAT IS RESULT TYPE?

```
enum Result<ResultType, ErrorType: Error > {
    case success(ResultType)
    case failure(ErrorType)
}
```

EXAMPLE

```
func fetchUnreadCount(from urlString: String,
                      completionHandler: @escaping (Result<Int, NetworkError>) -> Void)  {
    guard urlString.hasPrefix("https://") else {
        completionHandler(.failure(.unsecureURL))
        return
    }

    print("Fetching \(urlString)...")

    // complicated networking code here

    completionHandler(.success(5))
}
```

EXAMPLE (CONT)

```
fetchUnreadCount(from: "https://www.google.com") { result in
    switch result {
    case .success(let count):
        print("\(count) unread messages.")
    case .failure(let error):
        print(error.localizedDescription)
    }
}
```

EXTRA FEATURES

» get() function

» Initializer that accepts throwing closure

get() FUNCTION

```
fetchUnreadCount(from: "https://www.google.com") { result in
    if let count = try? result.get() {
        print("\(count) unread messages.")
    }
}
```

INITIALIZER

```
let result = Result { try String(contentsOfFile: someFile) }
```

FUTURE ENUM CASES

FUTURE ENUM CASES

Swift requires switch statements to be exhaustive

What if the framework you are using adds a new enum case?

FUTURE ENUM CASES

```
enum PasswordError: Error {
    case short
    case obvious
    case simple
}
```

FUTURE ENUM CASES

```
func showOld(error: PasswordError) {
    switch error {
    case .short:
        print("Your password was too short.")
    case .obvious:
        print("Your password was too obvious.")
    case .simple:
        print("Your password was too simple.")
    }
}
```

FUTURE ENUM CASES

```
enum PasswordError: Error {
    case short
    case obvious
    case simple
    case old
}
```

FUTURE ENUM CASES

```
func showOld(error: PasswordError) {
    switch error {
    case .short:
        print("Your password was too short.")
    case .obvious:
        print("Your password was too obvious.")
    case .simple:
        print("Your password was too simple.")
    default:
        print("Your password is not suitable")
    }
}
```

FUTURE ENUM CASES

```
func showNew(error: PasswordError) {
    switch error {
    case .short:
        print("Your password was too short.")
    case .obvious:
        print("Your password was too obvious.")
    case .simple:
        print("Your password was too simple.")
    @unknown default:
        print("Your password is not suitable")
    }
}
```

FLATTENING NESTED OPTIONALS

FLATTENING NESTED OPTIONALS

Swift 5 modifies the way try? works so that nested optionals are flattened to become regular optionals

This makes it work the same way as optional chaining and conditional typecasts

FLATTENING NESTED OPTIONALS

```
struct User {
    var id: Int

    init?(id: Int) {
        if id < 1 {
            return nil
        }

        self.id = id
    }

    func getMessages() throws -> String {
        // complicated code here
        return "No messages"
    }
}
```

FLATTENING NESTED OPTIONALS

```
let user = User(id: 1)
let messages = try? user?.getMessages()
```

FLATTENING NESTED OPTIONALS

```
let user = User(id: 1)
let messages = try? user?.getMessages()
```

Swift 4

messages is a String??

Swift 5

messages is a String?

CHECKING FOR INTEGER MULTIPLES

CHECKING FOR INTEGER MULTIPLES

Swift 5 adds an isMultiple(of:) function to Integers

CHECKING FOR INTEGER MULTIPLES

```
let rowNumber = 4

if rowNumber.isMultiple(of: 2) {
    print("Even")
} else {
    print("Odd")
}
```

COUNT MATCHING

COUNT MATCHING

Swift 5 introduces a new count(where:) function that performs the equivalent of a filter() and count in a single pass

COUNT MATCHING - SWIFT 4

```
let scores = [100, 80, 85]
let filtered = scores.filter { $0 >= 85 }
let passCount = filtered.count    // 2
```

COUNT MATCHING - SWIFT 5

```
let scores = [100, 80, 85]
let passCount = scores.count { $0 >= 85 } // 2
```

COUNT MATCHING

```
let pythons = [
    "Eric Idle",
    "Graham Chapman",
    "John Cleese",
    "Michael Palin",
    "Terry Gilliam",
    "Terry Jones"
    ]
let terryCount = pythons.count { $0.hasPrefix("Terry") }  // 2
```

COUNT MATCHING

This function is available to all types that conform to Sequence

COMPACT MAP VALUES

COMPACT MAP VALUES

New compactMapValues() function available on Dictionaries

COMPACT MAP VALUES

» compactMap() functionality of arrays

  » Transform my values, unwrap the results, then discard anything that's nil

» mapValues() functionality of dictionaries

  » Leave my keys intact but transform my values

COMPACTMAP()

```
let times = [
    "38",
    "42",
    "35",
    "DNF"
]

let compactedTimes = times.compactMap { Int($0) }

// [38, 42, 35]
```

MAPVALUES()

```
let times = [
    "Hudson": "38",
    "Clarke": "42",
    "Robinson": "35",
    "Hartis": "DNF"
]

let mappedTimes = times.mapValues { Int($0) }

/**
    ["Clarke": Optional(42),
    "Robinson": Optional(35),
    "Hartis": nil,
    "Hudson": Optional(38)]
*/
```

COMPACTMAPVALUES()

Let's combine both of those

COMPACT MAP VALUES

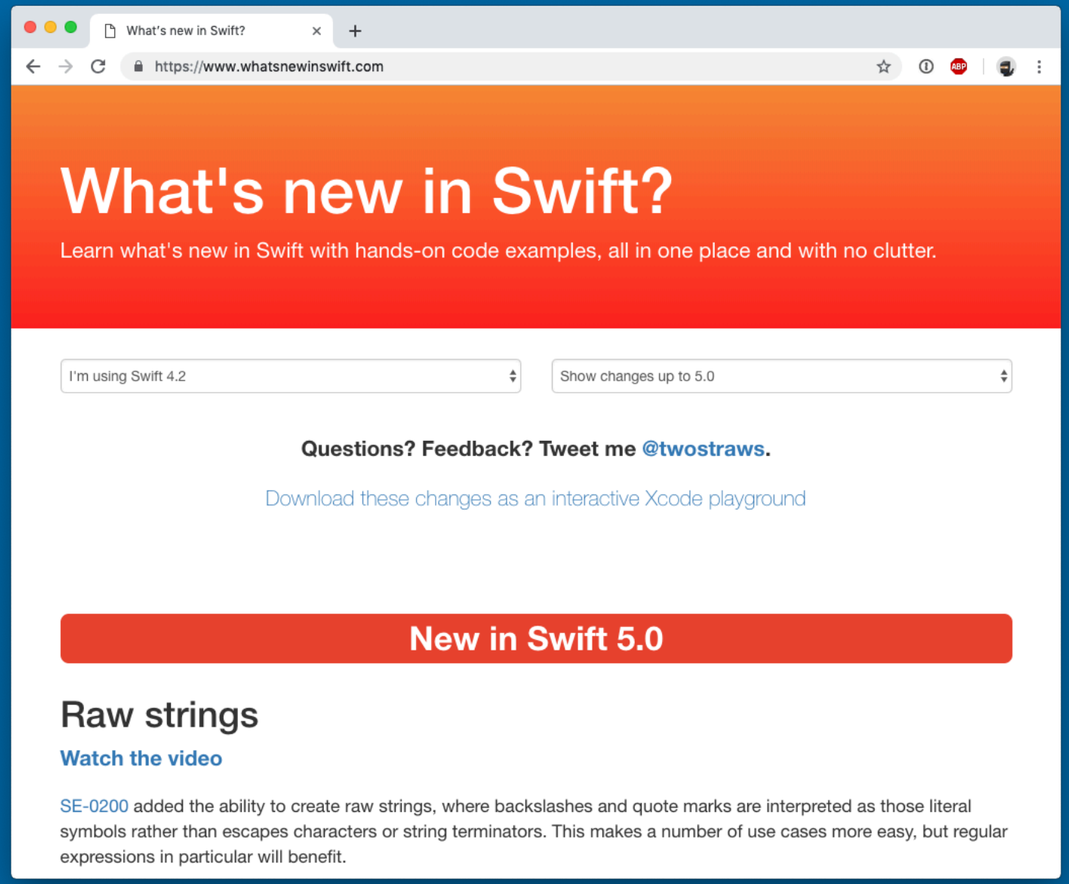
```
let times = [
    "Hudson": "38",
    "Clarke": "42",
    "Robinson": "35",
    "Hartis": "DNF"
]

let finishers1 = times.compactMapValues { Int($0) }

// ["Clarke": 42, "Hudson": 38, "Robinson": 35]
```

COMPACT MAP VALUES

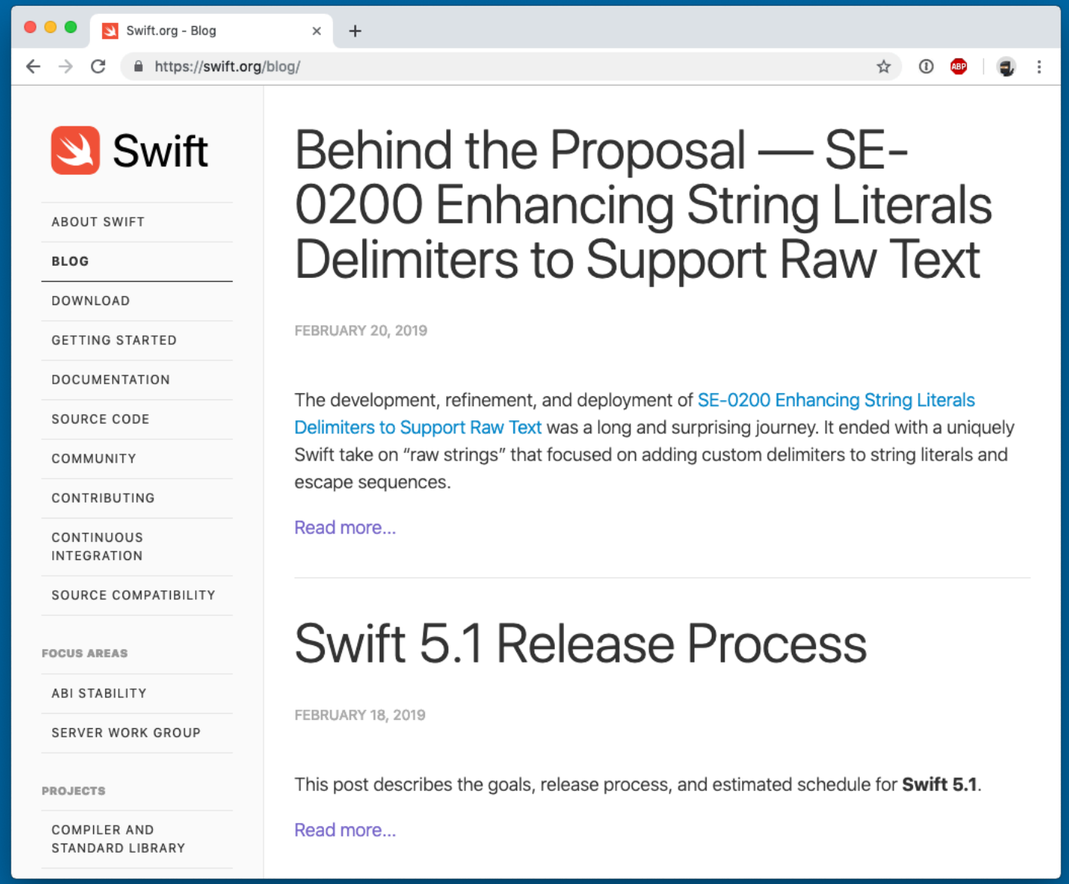
```
let times = [
    "Hudson": "38",
    "Clarke": "42",
    "Robinson": "35",
    "Hartis": "DNF"
]

let finishers2 = times.compactMapValues(Int.init)

// ["Clarke": 42, "Hudson": 38, "Robinson": 35]
```

WHERE YOU CAN LEARN MORE

WHATSNEWINSWIFT.COM



SWIFT.ORG


Swift.org - Blog
https://swift.org/blog/
Swift
ABOUT SWIFT
BLOG
DOWNLOAD
GETTING STARTED
DOCUMENTATION
SOURCE CODE
COMMUNITY
CONTRIBUTING
CONTINUOUS INTEGRATION
SOURCE COMPATIBILITY
FOCUS AREAS
ABI STABILITY
SERVER WORK GROUP
PROJECTS
COMPILER AND STANDARD LIBRARY
Behind the Proposal — SE-0200 Enhancing String Literals Delimiters to Support Raw Text
FEBRUARY 20, 2019
The development, refinement, and deployment of SE-0200 Enhancing String Literals Delimiters to Support Raw Text was a long and surprising journey. It ended with a uniquely Swift take on “raw strings” that focused on adding custom delimiters to string literals and escape sequences.
Read more…
Swift 5.1 Release Process
FEBRUARY 18, 2019
This post describes the goals, release process, and estimated schedule for Swift 5.1.
Read more…

THANK YOU!

» Questions?

» swift.mn

» info@swift.mn