免費電子書

學習

Swift Language

Free unaffiliated eBook created from
Stack Overflow contributors.

#swift

1: Swift

Swift

SwiftApple 。 SwiftApplemacOSiOStvOSwatchOSObjective-CCocoa / CocoaAPI。 SwiftmacOS Linux。 AndroidWindows。

SwiftGitHub ;。

bugs.swift.org。

SwiftSwift 。

- Swift
- Swift
- Swift
- API
- Swift iBooks
- ...developer.apple.com 。

Swift	Xcode	
	-	2010-07-17
1.0	Xcode 6	201462
1.1	Xcode 6.1	20141016
1.2	Xcode 6.3	201529
2.0	Xcode 7	2015-06-08
2.1	Xcode 7.1	2015923
	-	2015123
2.2	Xcode 7.3	2016321
2.3	Xcode 8	2016913
3.0	Xcode 8	2016913
3.1	Xcode 8.3	2017327

Examples

Swift

`hello.swift`

```
print("Hello, world!")
```

- `swift`

  Linux`CTRL` + `ALT` + `T` *macOS* Launchpad。 `cd` **`directory_name`** `cd ..`

```
print("Hello, world!")
```

  。

- `swiftc`

```
print("Hello, world!")
```

`hello`。 `./` 。

```
print("Hello, world!")
```

- swift REPLRead-Eval-Print-Loop`swift`

```
print("Hello, world!")
```

  - `func greet(name: String, surname: String) { // function body }` - `namesurname` 。
  - `print("Greetings \(name) \(surname)")` - “Greetings”`name surname` 。 `\(` **`variable_name`** `)`。
  - `let myName = "Homer"let mySurname = "Simpson"` - `let myName mySurname`values `"Homer" "Simpson"` 。
  - `greet(name: myName, surname: mySurname)` - `myName mySurname`。

REPL

```
print("Hello, world!")
```

`CTRL` + `D`REPL。

Swift

。

Swift。 macOS/ Library / Developer / Toolchains。

```
export PATH=/Library/Developer/Toolchains/swift-latest.xctoolchain/usr/bin:"${PATH}"
```

Linuxclang

```
export PATH=/Library/Developer/Toolchains/swift-latest.xctoolchain/usr/bin:"${PATH}"
```

SwiftSwift

```
export PATH=/Library/Developer/Toolchains/swift-latest.xctoolchain/usr/bin:"${PATH}"
```

Swift

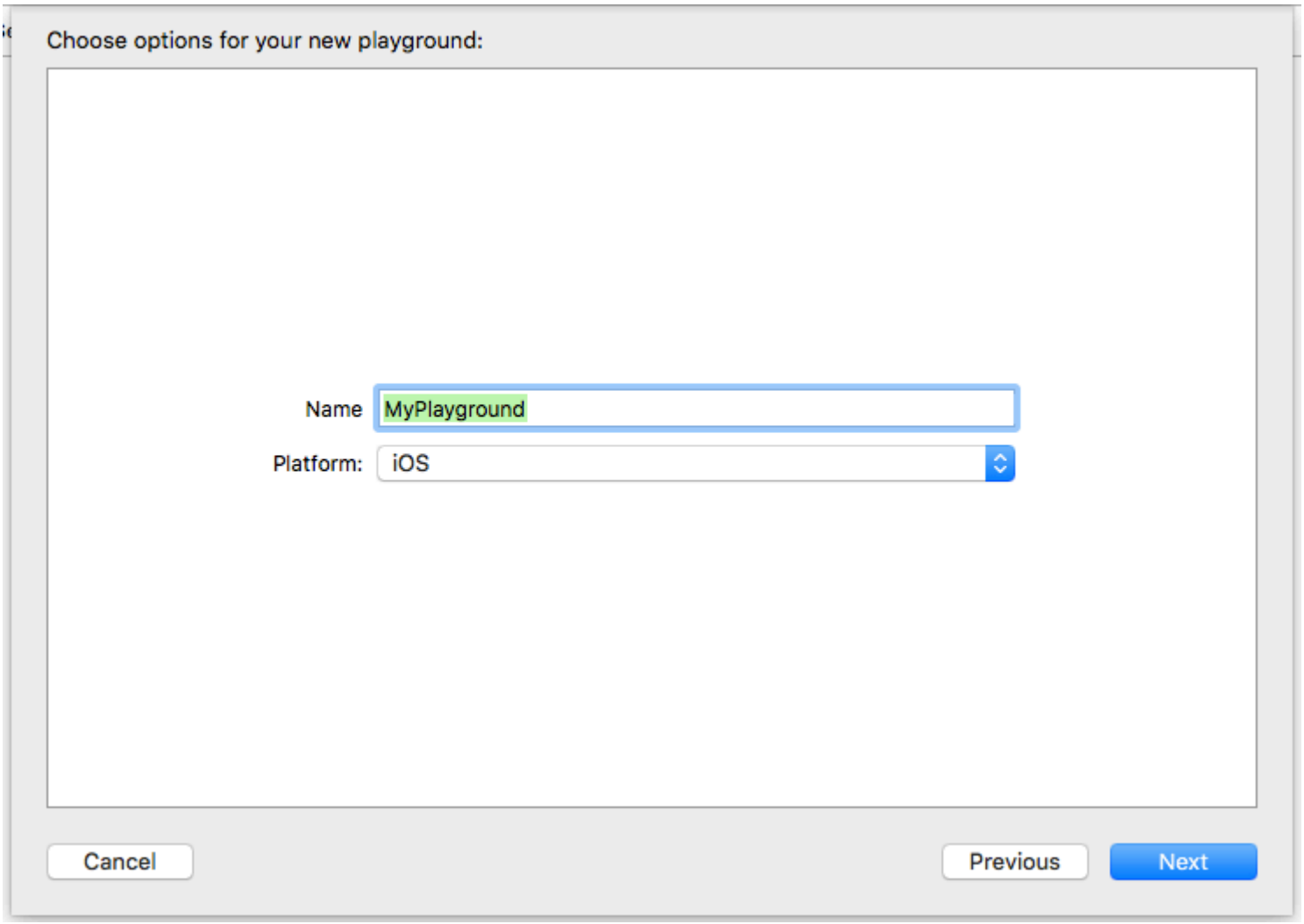
```
export PATH=/Library/Developer/Toolchains/swift-latest.xctoolchain/usr/bin:"${PATH}"
```

MacSwift

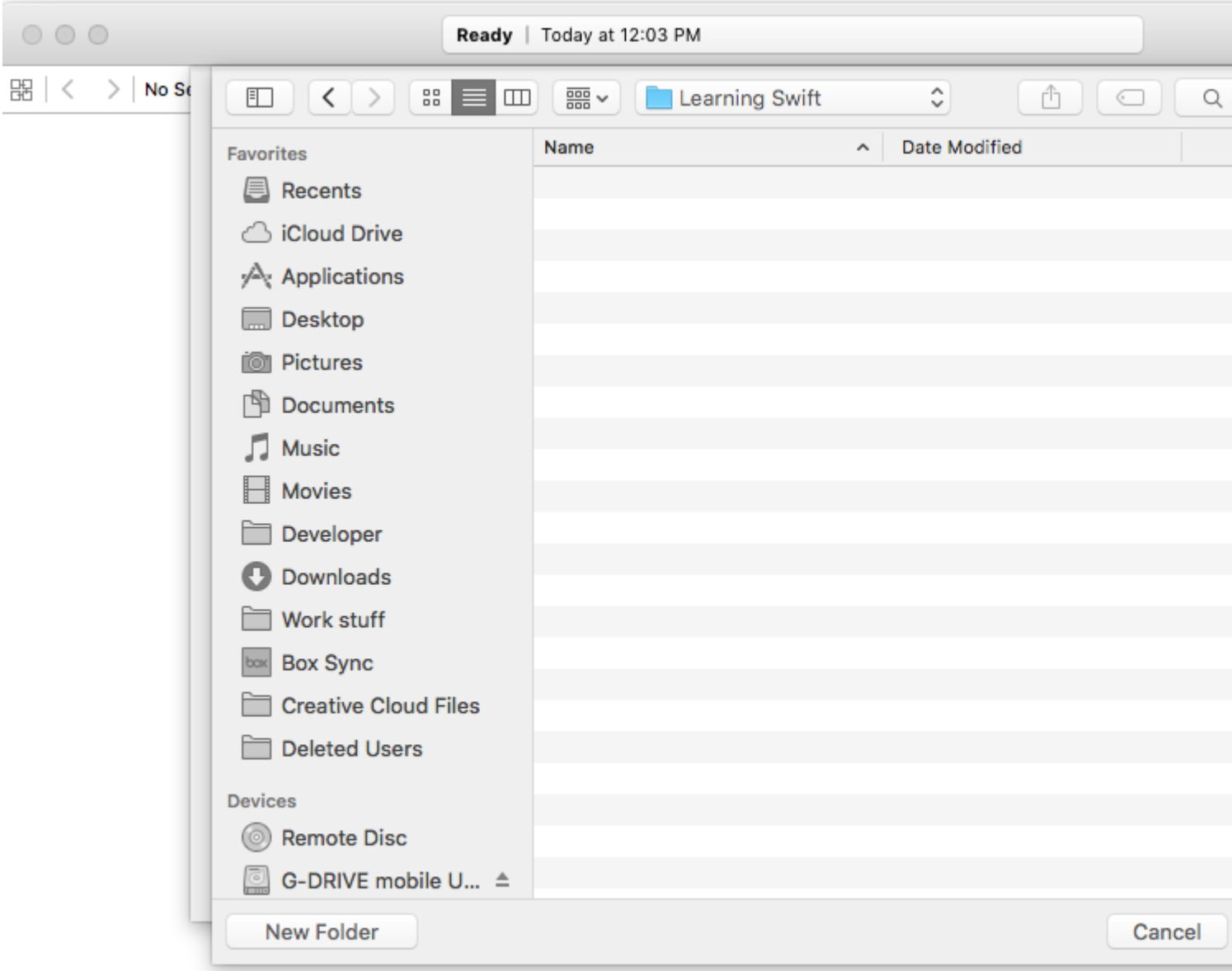
MacMac App StoreXcode。

Xcode**Playground**

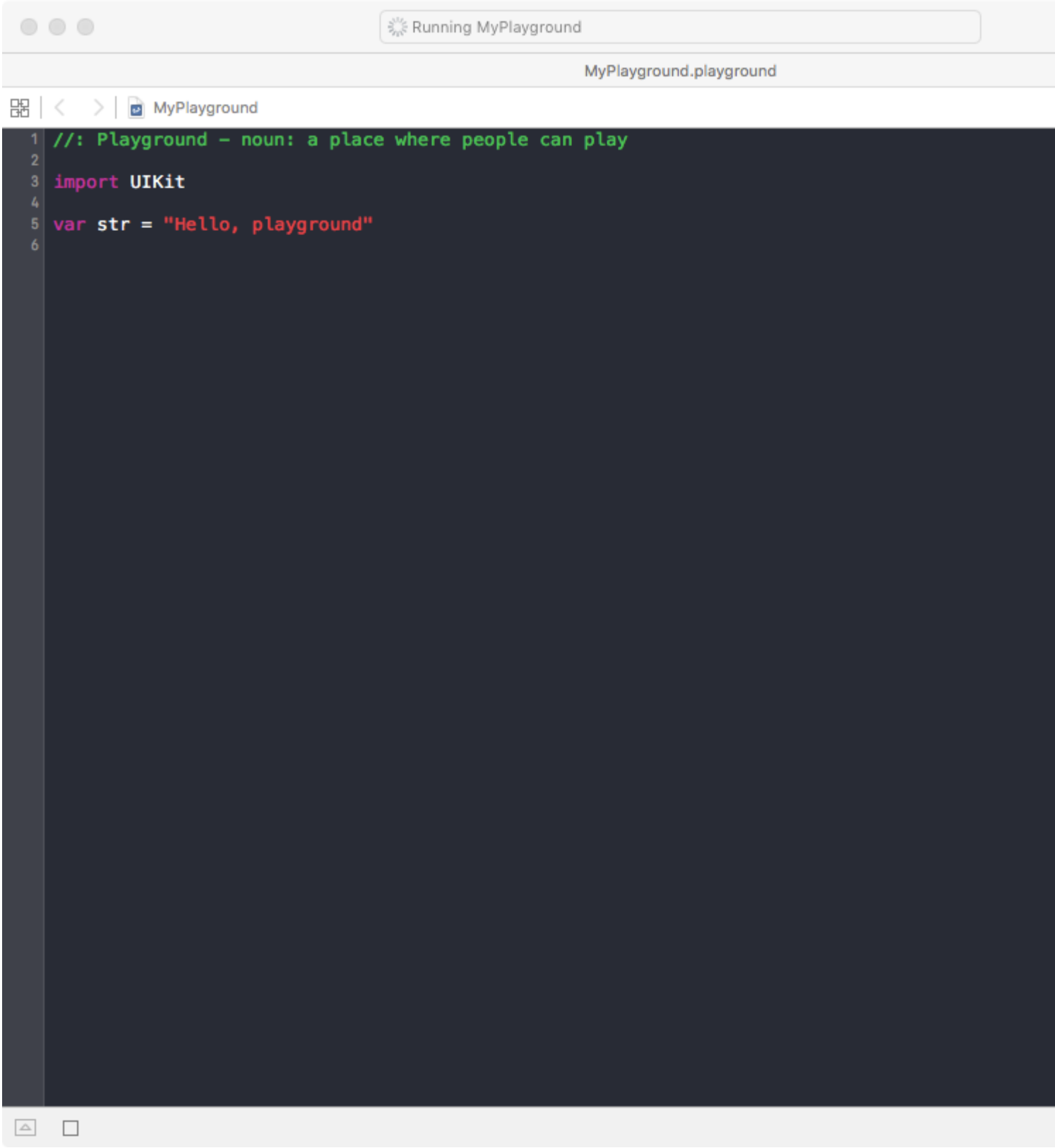
Playground`MyPlayground` “ ”



Playground**Create**


Ready | Today at 12:03 PM
Learning Swift
Favorites
Recents
iCloud Drive
Applications
Desktop
Pictures
Documents
Music
Movies
Developer
Downloads
Work stuff
Box Sync
Creative Cloud Files
Deleted Users
Devices
Remote Disc
G-DRIVE mobile U...
Name
Date Modified
New Folder
Cancel

Playground

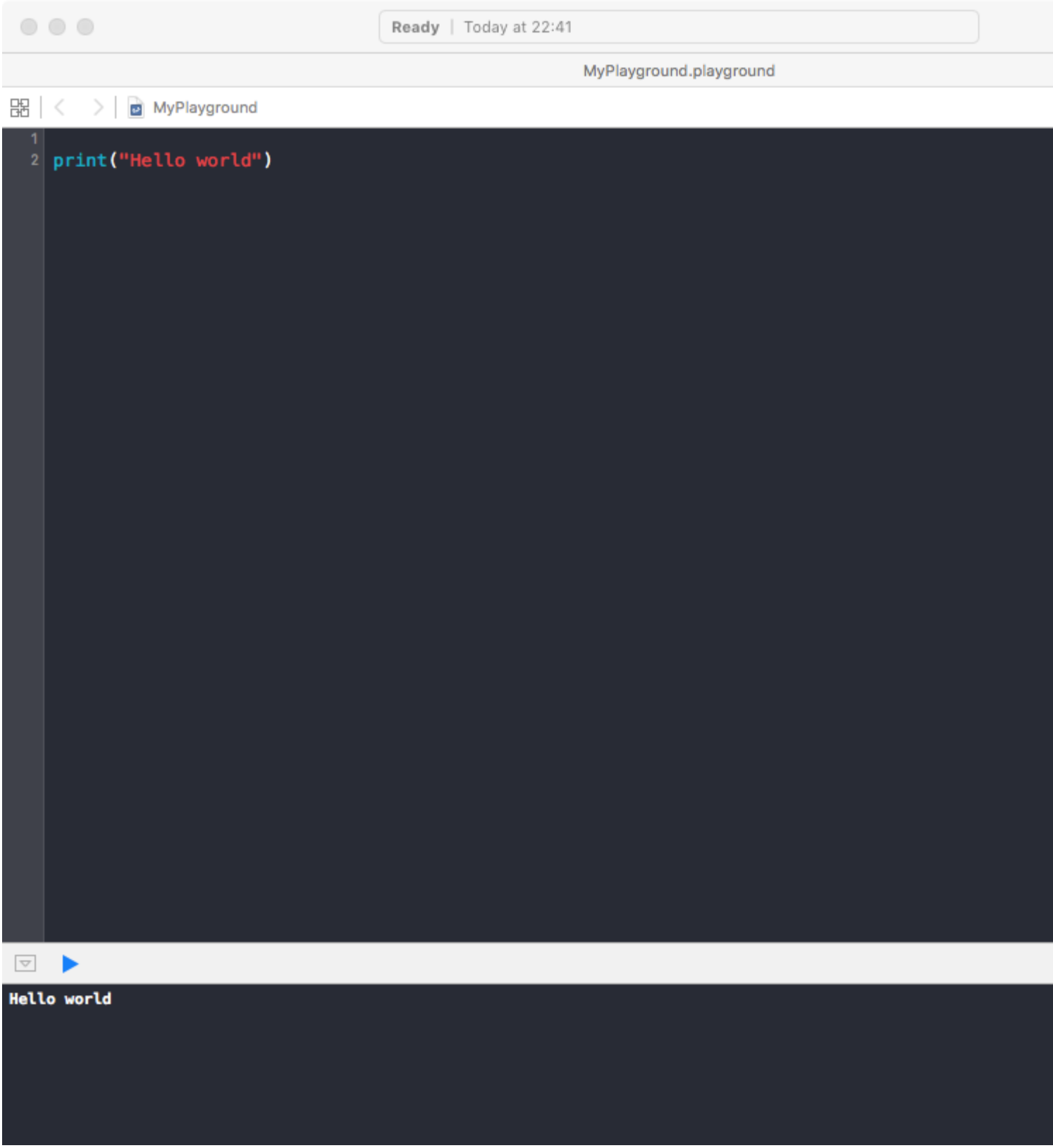


Playground + `cmd` + `Y` 。

Playground

```
print("Hello world")
```

“Hello world”“Hello world \ n”



Swift

iPadSwift Playgrounds

Swift PlaygroundsSwift。

1-App StoreiPadSwift Playgrounds 。


Mac
iPad
iTunes Preview
Swift Playgroun
By Apple
Open iTunes to buy and do
View in iTunes
Free

2.
3. “ ”+

2:

“。 。 “

Apple Inc.“SwiftCocoaObjective-CSwift 3.1。 ”iBooks。 https://itun.es/us/utTW7.l

。

BufferPointers。

- MemoryLayout 。
- Unmanaged 。
- UnsafeBufferPointer 。
- UnsafeBufferPointerIterator *UnsafeBufferPointerUnsafeMutableBufferPointer* 。
- UnsafeMutableBufferPointer 。
- UnsafeMutablePointer 。
- UnsafeMutableRawBufferPointer *nonowning*。
- UnsafeMutableRawBufferPointer.Iterator 。
- UnsafeMutableRawPointer 。
- UnsafePointer 。
- UnsafeRawBufferPointer 。
- UnsafeRawBufferPointer.Iterator 。
- UnsafeRawPointer 。

  Swiftdoc.org

Examples

UnsafeMutablePointer

```
struct UnsafeMutablePointer<Pointee>
```

。

UnsafeMutablePointer。 Pointee。 UnsafeMutablePointer。 。

。 UnsafeMutablePointer。

```
struct UnsafeMutablePointer<Pointee>
```

Swift 3.0

Swift;

```
public init?(validatingUTF8 cString: UnsafePointer<CChar>)
```

nullUTF-8。

UTF-8。 `nil` 。 `CChar` - UTF-8。

> **Source Apple Inc.Swift 3** For header accessIn PlaygroundCmd +Swift
>
> `import Swift`

```
public init?(validatingUTF8 cString: UnsafePointer<CChar>)
```

> SourceApple Inc.Swift Header File Example

https://riptutorial.com/zh-TW/swift/topic/9140/--

3: AES

Examples

IVSwift 3.0CBCAES

iv

`aesCBC128Encrypt`IV。
`aesCBC128Decrypt`IV。

。 Base64/。

128161922425632。 。

PKCS7。

Common Crypto

`#import <CommonCrypto/CommonCrypto.h>`

`Security.framework`。

。

```
enum AESError: Error {
    case KeyError((String, Int))
    case IVError((String, Int))
    case CryptorError((String, Int))
}

// The iv is prefixed to the encrypted data
func aesCBCEncrypt(data:Data, keyData:Data) throws -> Data {
    let keyLength = keyData.count
    let validKeyLengths = [kCCKeySizeAES128, kCCKeySizeAES192, kCCKeySizeAES256]
    if (validKeyLengths.contains(keyLength) == false) {
        throw AESError.KeyError(("Invalid key length", keyLength))
    }

    let ivSize = kCCBlockSizeAES128;
    let cryptLength = size_t(ivSize + data.count + kCCBlockSizeAES128)
    var cryptData = Data(count:cryptLength)

    let status = cryptData.withUnsafeMutableBytes {ivBytes in
        SecRandomCopyBytes(kSecRandomDefault, kCCBlockSizeAES128, ivBytes)
    }
    if (status != 0) {
        throw AESError.IVError(("IV generation failed", Int(status)))
    }

    var numBytesEncrypted :size_t = 0
    let options   = CCOptions(kCCOptionPKCS7Padding)
```

```
    let cryptStatus = cryptData.withUnsafeMutableBytes {cryptBytes in
        data.withUnsafeBytes {dataBytes in
            keyData.withUnsafeBytes {keyBytes in
                CCCrypt(CCOperation(kCCEncrypt),
                        CCAlgorithm(kCCAlgorithmAES),
                        options,
                        keyBytes, keyLength,
                        cryptBytes,
                        dataBytes, data.count,
                        cryptBytes+kCCBlockSizeAES128, cryptLength,
                        &numBytesEncrypted)
            }
        }
    }

    if UInt32(cryptStatus) == UInt32(kCCSuccess) {
        cryptData.count = numBytesEncrypted + ivSize
    }
    else {
        throw AESError.CryptorError(("Encryption failed", Int(cryptStatus)))
    }

    return cryptData;
}

// The iv is prefixed to the encrypted data
func aesCBCDecrypt(data:Data, keyData:Data) throws -> Data? {
    let keyLength = keyData.count
    let validKeyLengths = [kCCKeySizeAES128, kCCKeySizeAES192, kCCKeySizeAES256]
    if (validKeyLengths.contains(keyLength) == false) {
        throw AESError.KeyError(("Invalid key length", keyLength))
    }

    let ivSize = kCCBlockSizeAES128;
    let clearLength = size_t(data.count - ivSize)
    var clearData = Data(count:clearLength)

    var numBytesDecrypted :size_t = 0
    let options   = CCOptions(kCCOptionPKCS7Padding)

    let cryptStatus = clearData.withUnsafeMutableBytes {cryptBytes in
        data.withUnsafeBytes {dataBytes in
            keyData.withUnsafeBytes {keyBytes in
                CCCrypt(CCOperation(kCCDecrypt),
                        CCAlgorithm(kCCAlgorithmAES128),
                        options,
                        keyBytes, keyLength,
                        dataBytes,
                        dataBytes+kCCBlockSizeAES128, clearLength,
                        cryptBytes, clearLength,
                        &numBytesDecrypted)
            }
        }
    }

    if UInt32(cryptStatus) == UInt32(kCCSuccess) {
        clearData.count = numBytesDecrypted
    }
    else {
        throw AESError.CryptorError(("Decryption failed", Int(cryptStatus)))
    }
```

```
    return clearData;
}
```

```
enum AESError: Error {
    case KeyError((String, Int))
    case IVError((String, Int))
    case CryptorError((String, Int))
}

// The iv is prefixed to the encrypted data
func aesCBCEncrypt(data:Data, keyData:Data) throws -> Data {
    let keyLength = keyData.count
    let validKeyLengths = [kCCKeySizeAES128, kCCKeySizeAES192, kCCKeySizeAES256]
    if (validKeyLengths.contains(keyLength) == false) {
        throw AESError.KeyError(("Invalid key length", keyLength))
    }

    let ivSize = kCCBlockSizeAES128;
    let cryptLength = size_t(ivSize + data.count + kCCBlockSizeAES128)
    var cryptData = Data(count:cryptLength)

    let status = cryptData.withUnsafeMutableBytes {ivBytes in
        SecRandomCopyBytes(kSecRandomDefault, kCCBlockSizeAES128, ivBytes)
    }
    if (status != 0) {
        throw AESError.IVError(("IV generation failed", Int(status)))
    }

    var numBytesEncrypted :size_t = 0
    let options   = CCOptions(kCCOptionPKCS7Padding)

    let cryptStatus = cryptData.withUnsafeMutableBytes {cryptBytes in
        data.withUnsafeBytes {dataBytes in
            keyData.withUnsafeBytes {keyBytes in
                CCCrypt(CCOperation(kCCEncrypt),
                        CCAlgorithm(kCCAlgorithmAES),
                        options,
                        keyBytes, keyLength,
                        cryptBytes,
                        dataBytes, data.count,
                        cryptBytes+kCCBlockSizeAES128, cryptLength,
                        &numBytesEncrypted)
            }
        }
    }

    if UInt32(cryptStatus) == UInt32(kCCSuccess) {
        cryptData.count = numBytesEncrypted + ivSize
    }
    else {
        throw AESError.CryptorError(("Encryption failed", Int(cryptStatus)))
    }

    return cryptData;
}

// The iv is prefixed to the encrypted data
func aesCBCDecrypt(data:Data, keyData:Data) throws -> Data? {
    let keyLength = keyData.count
```

```
    let validKeyLengths = [kCCKeySizeAES128, kCCKeySizeAES192, kCCKeySizeAES256]
    if (validKeyLengths.contains(keyLength) == false) {
        throw AESError.KeyError(("Invalid key length", keyLength))
    }

    let ivSize = kCCBlockSizeAES128;
    let clearLength = size_t(data.count - ivSize)
    var clearData = Data(count:clearLength)

    var numBytesDecrypted :size_t = 0
    let options   = CCOptions(kCCOptionPKCS7Padding)

    let cryptStatus = clearData.withUnsafeMutableBytes {cryptBytes in
        data.withUnsafeBytes {dataBytes in
            keyData.withUnsafeBytes {keyBytes in
                CCCrypt(CCOperation(kCCDecrypt),
                        CCAlgorithm(kCCAlgorithmAES128),
                        options,
                        keyBytes, keyLength,
                        dataBytes,
                        dataBytes+kCCBlockSizeAES128, clearLength,
                        cryptBytes, clearLength,
                        &numBytesDecrypted)
            }
        }
    }

    if UInt32(cryptStatus) == UInt32(kCCSuccess) {
        clearData.count = numBytesDecrypted
    }
    else {
        throw AESError.CryptorError(("Decryption failed", Int(cryptStatus)))
    }

    return clearData;
}
```

```
enum AESError: Error {
    case KeyError((String, Int))
    case IVError((String, Int))
    case CryptorError((String, Int))
}

// The iv is prefixed to the encrypted data
func aesCBCEncrypt(data:Data, keyData:Data) throws -> Data {
    let keyLength = keyData.count
    let validKeyLengths = [kCCKeySizeAES128, kCCKeySizeAES192, kCCKeySizeAES256]
    if (validKeyLengths.contains(keyLength) == false) {
        throw AESError.KeyError(("Invalid key length", keyLength))
    }

    let ivSize = kCCBlockSizeAES128;
    let cryptLength = size_t(ivSize + data.count + kCCBlockSizeAES128)
    var cryptData = Data(count:cryptLength)

    let status = cryptData.withUnsafeMutableBytes {ivBytes in
        SecRandomCopyBytes(kSecRandomDefault, kCCBlockSizeAES128, ivBytes)
    }
    if (status != 0) {
        throw AESError.IVError(("IV generation failed", Int(status)))
```

```
    }

    var numBytesEncrypted :size_t = 0
    let options   = CCOptions(kCCOptionPKCS7Padding)

    let cryptStatus = cryptData.withUnsafeMutableBytes {cryptBytes in
        data.withUnsafeBytes {dataBytes in
            keyData.withUnsafeBytes {keyBytes in
                CCCrypt(CCOperation(kCCEncrypt),
                        CCAlgorithm(kCCAlgorithmAES),
                        options,
                        keyBytes, keyLength,
                        cryptBytes,
                        dataBytes, data.count,
                        cryptBytes+kCCBlockSizeAES128, cryptLength,
                        &numBytesEncrypted)
            }
        }
    }

    if UInt32(cryptStatus) == UInt32(kCCSuccess) {
        cryptData.count = numBytesEncrypted + ivSize
    }
    else {
        throw AESError.CryptorError(("Encryption failed", Int(cryptStatus)))
    }

    return cryptData;
}

// The iv is prefixed to the encrypted data
func aesCBCDecrypt(data:Data, keyData:Data) throws -> Data? {
    let keyLength = keyData.count
    let validKeyLengths = [kCCKeySizeAES128, kCCKeySizeAES192, kCCKeySizeAES256]
    if (validKeyLengths.contains(keyLength) == false) {
        throw AESError.KeyError(("Invalid key length", keyLength))
    }

    let ivSize = kCCBlockSizeAES128;
    let clearLength = size_t(data.count - ivSize)
    var clearData = Data(count:clearLength)

    var numBytesDecrypted :size_t = 0
    let options   = CCOptions(kCCOptionPKCS7Padding)

    let cryptStatus = clearData.withUnsafeMutableBytes {cryptBytes in
        data.withUnsafeBytes {dataBytes in
            keyData.withUnsafeBytes {keyBytes in
                CCCrypt(CCOperation(kCCDecrypt),
                        CCAlgorithm(kCCAlgorithmAES128),
                        options,
                        keyBytes, keyLength,
                        dataBytes,
                        dataBytes+kCCBlockSizeAES128, clearLength,
                        cryptBytes, clearLength,
                        &numBytesDecrypted)
            }
        }
    }

    if UInt32(cryptStatus) == UInt32(kCCSuccess) {
```

```
        clearData.count = numBytesDecrypted
    }
    else {
        throw AESError.CryptorError(("Decryption failed", Int(cryptStatus)))
    }

    return clearData;
}
```

CBCIV。 IVIV。 CBC。

。

PBKDF2。

RNCryptor 。

throw / catch。

CBCAESIVSwift 2.3

iv

aesCBC128EncryptIV。 aesCBC128DecryptIV。

。 Base64/。

12816。 Swift 3.0。

PKCS7。

Common Crypto#import <CommonCrypto / CommonCrypto.h>Security.framework。

Swift 3。

。

```
func aesCBC128Encrypt(data data:[UInt8], keyData:[UInt8]) -> [UInt8]? {
    let keyLength   = size_t(kCCKeySizeAES128)
    let ivLength    = size_t(kCCBlockSizeAES128)
    let cryptDataLength = size_t(data.count + kCCBlockSizeAES128)
    var cryptData = [UInt8](count:ivLength + cryptDataLength, repeatedValue:0)

    let status = SecRandomCopyBytes(kSecRandomDefault, Int(ivLength),
UnsafeMutablePointer<UInt8>(cryptData));
    if (status != 0) {
        print("IV Error, errno: \(status)")
        return nil
    }

    var numBytesEncrypted :size_t = 0
    let cryptStatus = CCCrypt(CCOperation(kCCEncrypt),
```

```
                            CCAlgorithm(kCCAlgorithmAES128),
                            CCOptions(kCCOptionPKCS7Padding),
                            keyData, keyLength,
                            cryptData,
                            data, data.count,
                            &cryptData + ivLength, cryptDataLength,
                            &numBytesEncrypted)

    if UInt32(cryptStatus) == UInt32(kCCSuccess) {
        cryptData.removeRange(numBytesEncrypted+ivLength..<cryptData.count)
    }
    else {
        print("Error: \(cryptStatus)")
        return nil;
    }

    return cryptData;
}

func aesCBC128Decrypt(data data:[UInt8], keyData:[UInt8]) -> [UInt8]? {
    let clearLength = size_t(data.count)
    var clearData   = [UInt8](count:clearLength, repeatedValue:0)

    let keyLength   = size_t(kCCKeySizeAES128)
    let ivLength    = size_t(kCCBlockSizeAES128)

    var numBytesDecrypted :size_t = 0
    let cryptStatus = CCCrypt(CCOperation(kCCDecrypt),
                            CCAlgorithm(kCCAlgorithmAES128),
                            CCOptions(kCCOptionPKCS7Padding),
                            keyData, keyLength,
                            data,
                            UnsafePointer<UInt8>(data) + ivLength, data.count - ivLength,
                            &clearData, clearLength,
                            &numBytesDecrypted)

    if UInt32(cryptStatus) == UInt32(kCCSuccess) {
        clearData.removeRange(numBytesDecrypted..<clearLength)

    } else {
        print("Error: \(cryptStatus)")
        return nil;
    }

    return clearData;
}
```

```
func aesCBC128Encrypt(data data:[UInt8], keyData:[UInt8]) -> [UInt8]? {
    let keyLength   = size_t(kCCKeySizeAES128)
    let ivLength    = size_t(kCCBlockSizeAES128)
    let cryptDataLength = size_t(data.count + kCCBlockSizeAES128)
    var cryptData = [UInt8](count:ivLength + cryptDataLength, repeatedValue:0)

    let status = SecRandomCopyBytes(kSecRandomDefault, Int(ivLength),
UnsafeMutablePointer<UInt8>(cryptData));
    if (status != 0) {
        print("IV Error, errno: \(status)")
        return nil
    }
```

```
    var numBytesEncrypted :size_t = 0
    let cryptStatus = CCCrypt(CCOperation(kCCEncrypt),
                              CCAlgorithm(kCCAlgorithmAES128),
                              CCOptions(kCCOptionPKCS7Padding),
                              keyData, keyLength,
                              cryptData,
                              data, data.count,
                              &cryptData + ivLength, cryptDataLength,
                              &numBytesEncrypted)

    if UInt32(cryptStatus) == UInt32(kCCSuccess) {
        cryptData.removeRange(numBytesEncrypted+ivLength..<cryptData.count)
    }
    else {
        print("Error: \(cryptStatus)")
        return nil;
    }

    return cryptData;
}

func aesCBC128Decrypt(data data:[UInt8], keyData:[UInt8]) -> [UInt8]? {
    let clearLength = size_t(data.count)
    var clearData   = [UInt8](count:clearLength, repeatedValue:0)

    let keyLength   = size_t(kCCKeySizeAES128)
    let ivLength    = size_t(kCCBlockSizeAES128)

    var numBytesDecrypted :size_t = 0
    let cryptStatus = CCCrypt(CCOperation(kCCDecrypt),
                              CCAlgorithm(kCCAlgorithmAES128),
                              CCOptions(kCCOptionPKCS7Padding),
                              keyData, keyLength,
                              data,
                              UnsafePointer<UInt8>(data) + ivLength, data.count - ivLength,
                              &clearData, clearLength,
                              &numBytesDecrypted)

    if UInt32(cryptStatus) == UInt32(kCCSuccess) {
        clearData.removeRange(numBytesDecrypted..<clearLength)

    } else {
        print("Error: \(cryptStatus)")
        return nil;
    }

    return clearData;
}
```

```
func aesCBC128Encrypt(data data:[UInt8], keyData:[UInt8]) -> [UInt8]? {
    let keyLength   = size_t(kCCKeySizeAES128)
    let ivLength    = size_t(kCCBlockSizeAES128)
    let cryptDataLength = size_t(data.count + kCCBlockSizeAES128)
    var cryptData = [UInt8](count:ivLength + cryptDataLength, repeatedValue:0)

    let status = SecRandomCopyBytes(kSecRandomDefault, Int(ivLength),
UnsafeMutablePointer<UInt8>(cryptData));
    if (status != 0) {
        print("IV Error, errno: \(status)")
        return nil
```

```
    }

    var numBytesEncrypted :size_t = 0
    let cryptStatus = CCCrypt(CCOperation(kCCEncrypt),
                              CCAlgorithm(kCCAlgorithmAES128),
                              CCOptions(kCCOptionPKCS7Padding),
                              keyData, keyLength,
                              cryptData,
                              data, data.count,
                              &cryptData + ivLength, cryptDataLength,
                              &numBytesEncrypted)

    if UInt32(cryptStatus) == UInt32(kCCSuccess) {
        cryptData.removeRange(numBytesEncrypted+ivLength..<cryptData.count)
    }
    else {
        print("Error: \(cryptStatus)")
        return nil;
    }

    return cryptData;
}

func aesCBC128Decrypt(data data:[UInt8], keyData:[UInt8]) -> [UInt8]? {
    let clearLength = size_t(data.count)
    var clearData   = [UInt8](count:clearLength, repeatedValue:0)

    let keyLength   = size_t(kCCKeySizeAES128)
    let ivLength    = size_t(kCCBlockSizeAES128)

    var numBytesDecrypted :size_t = 0
    let cryptStatus = CCCrypt(CCOperation(kCCDecrypt),
                              CCAlgorithm(kCCAlgorithmAES128),
                              CCOptions(kCCOptionPKCS7Padding),
                              keyData, keyLength,
                              data,
                              UnsafePointer<UInt8>(data) + ivLength, data.count - ivLength,
                              &clearData, clearLength,
                              &numBytesDecrypted)

    if UInt32(cryptStatus) == UInt32(kCCSuccess) {
        clearData.removeRange(numBytesDecrypted..<clearLength)

    } else {
        print("Error: \(cryptStatus)")
        return nil;
    }

    return clearData;
}
```

PKCS7ECBAES

AppleIV

ECB。

```
func AESEncryption(key: String) -> String? {
```

```
        let keyData: NSData! = (key as NSString).data(using: String.Encoding.utf8.rawValue) as
NSData!

        let data: NSData! = (self as NSString).data(using: String.Encoding.utf8.rawValue) as
NSData!

        let cryptData    = NSMutableData(length: Int(data.length) + kCCBlockSizeAES128)!

        let keyLength              = size_t(kCCKeySizeAES128)
        let operation: CCOperation = UInt32(kCCEncrypt)
        let algoritm:  CCAlgorithm = UInt32(kCCAlgorithmAES128)
        let options:   CCOptions   = UInt32(kCCOptionECBMode + kCCOptionPKCS7Padding)

        var numBytesEncrypted :size_t = 0


        let cryptStatus = CCCrypt(operation,
                                  algoritm,
                                  options,
                                  keyData.bytes, keyLength,
                                  nil,
                                  data.bytes, data.length,
                                  cryptData.mutableBytes, cryptData.length,
                                  &numBytesEncrypted)

        if UInt32(cryptStatus) == UInt32(kCCSuccess) {
            cryptData.length = Int(numBytesEncrypted)

            var bytes = [UInt8](repeating: 0, count: cryptData.length)
            cryptData.getBytes(&bytes, length: cryptData.length)

            var hexString = ""
            for byte in bytes {
                hexString += String(format:"%02x", UInt8(byte))
            }

            return hexString
        }

        return nil
    }
```

AES https://riptutorial.com/zh-TW/swift/topic/7026/aes

4: KituraSwift HTTP

`KituraKitura`

`Kitura`swiftWebWeb。 HTTP。 XCodeOS Xswift 3.0Linux。

Examples

Package.swift。 swift。 hello worldGitHub repos。 `KituraHeliumLogger` 。 Package.swift。 *kitura-helloworld*URL。

```
import PackageDescription
let package = Package(
    name: "kitura-helloworld",
        dependencies: [
            .Package(url: "https://github.com/IBM-Swift/HeliumLogger.git", majorVersion: 1,
minor: 6),
            .Package(url: "https://github.com/IBM-Swift/Kitura.git", majorVersion: 1, minor:
6) ] )
```

Sources。 main.swift。 。 。

```
import PackageDescription
let package = Package(
    name: "kitura-helloworld",
        dependencies: [
            .Package(url: "https://github.com/IBM-Swift/HeliumLogger.git", majorVersion: 1,
minor: 6),
            .Package(url: "https://github.com/IBM-Swift/Kitura.git", majorVersion: 1, minor:
6) ] )
```

RouterHTTP。 GET*Hello world* post。

```
import PackageDescription
let package = Package(
    name: "kitura-helloworld",
        dependencies: [
            .Package(url: "https://github.com/IBM-Swift/HeliumLogger.git", majorVersion: 1,
minor: 6),
            .Package(url: "https://github.com/IBM-Swift/Kitura.git", majorVersion: 1, minor:
6) ] )
```

```
import PackageDescription
let package = Package(
    name: "kitura-helloworld",
        dependencies: [
            .Package(url: "https://github.com/IBM-Swift/HeliumLogger.git", majorVersion: 1,
minor: 6),
            .Package(url: "https://github.com/IBM-Swift/Kitura.git", majorVersion: 1, minor:
6) ] )
```

HTTP

```
import PackageDescription
let package = Package(
    name: "kitura-helloworld",
        dependencies: [
            .Package(url: "https://github.com/IBM-Swift/HeliumLogger.git", majorVersion: 1,
minor: 6),
            .Package(url: "https://github.com/IBM-Swift/Kitura.git", majorVersion: 1, minor:
6) ] )
```

Package.swiftResources。 。 SwiftPackage.swiftPackagesmain.swift

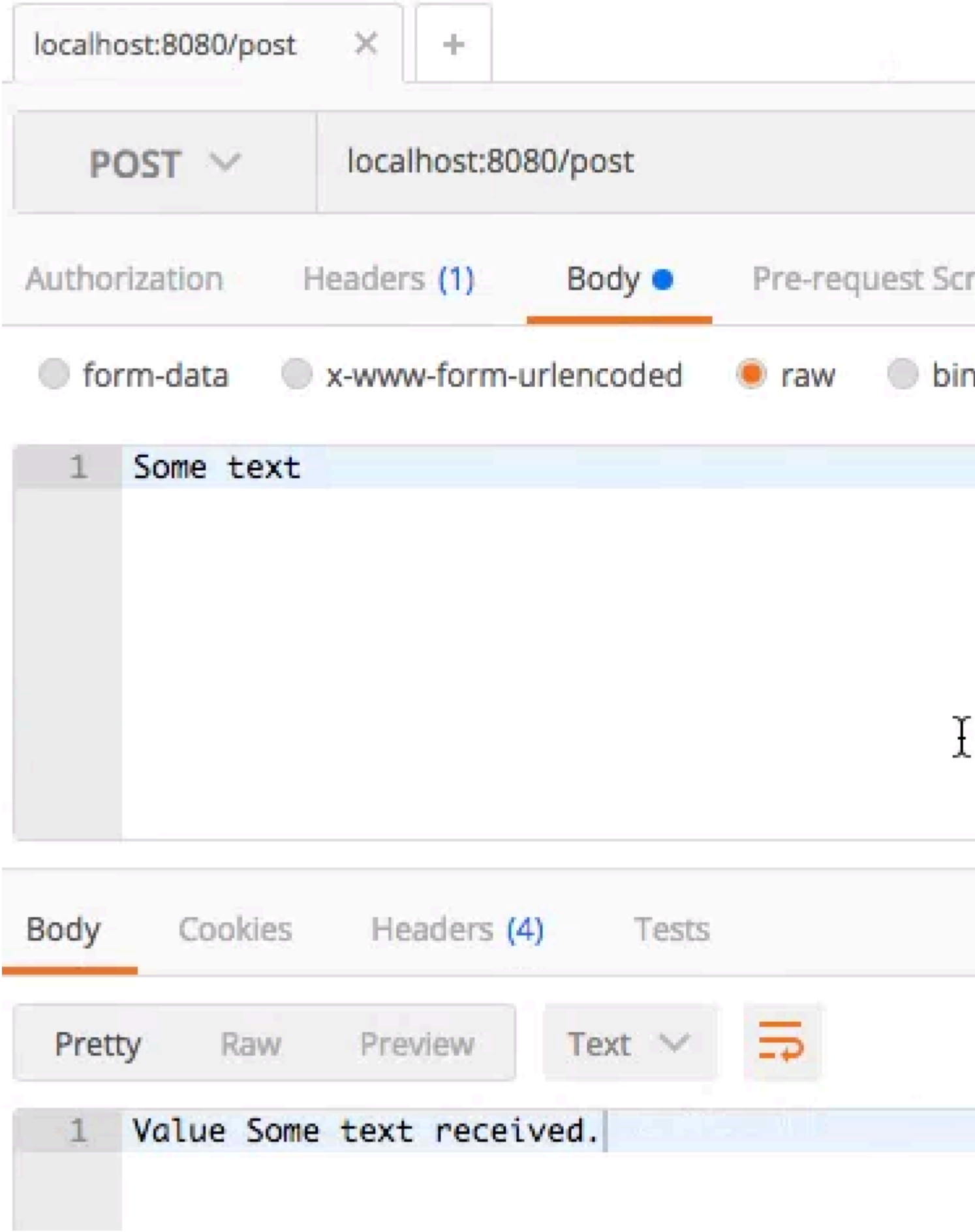
```
import PackageDescription
let package = Package(
    name: "kitura-helloworld",
        dependencies: [
            .Package(url: "https://github.com/IBM-Swift/HeliumLogger.git", majorVersion: 1,
minor: 6),
            .Package(url: "https://github.com/IBM-Swift/Kitura.git", majorVersion: 1, minor:
6) ] )
```

。 。

```
import PackageDescription
let package = Package(
    name: "kitura-helloworld",
        dependencies: [
            .Package(url: "https://github.com/IBM-Swift/HeliumLogger.git", majorVersion: 1,
minor: 6),
            .Package(url: "https://github.com/IBM-Swift/Kitura.git", majorVersion: 1, minor:
6) ] )
```

`localhost:8080/get` as urlEnter。 。

HTTP`localhost:8080/post` ◦ ◦



KituraSwift HTTP https://riptutorial.com/zh-TW/swift/topic/10690/kituraswift-http

5: OptionSet

Examples

OptionSet

OptionSetType。 /

```
struct Features : OptionSet {
  let rawValue : Int
  static let none = Features(rawValue: 0)
  static let feature0 = Features(rawValue: 1 << 0)
  static let feature1 = Features(rawValue: 1 << 1)
  static let feature2 = Features(rawValue: 1 << 2)
  static let feature3 = Features(rawValue: 1 << 3)
  static let feature4 = Features(rawValue: 1 << 4)
  static let feature5 = Features(rawValue: 1 << 5)
  static let all: Features = [feature0, feature1, feature2, feature3, feature4, feature5]
}

Features.feature1.rawValue //2
Features.all.rawValue //63

var options: Features = [.feature1, .feature2, .feature3]

options.contains(.feature1) //true
options.contains(.feature4) //false

options.insert(.feature4)
options.contains(.feature4) //true


var otherOptions : Features = [.feature1, .feature5]

options.contains(.feature5) //false

options.formUnion(otherOptions)
options.contains(.feature5) //true

options.remove(.feature5)
options.contains(.feature5) //false
```

OptionSet https://riptutorial.com/zh-TW/swift/topic/1242/optionset

6: PBKDF2

Examples

2Swift 3

。

SHA1SHA256SHA512。

rounds。 100ms500ms。

Common Crypto

`#import <CommonCrypto/CommonCrypto.h>`

`Security.framework`。

```
password     password String
salt         salt Data
keyByteCount number of key bytes to generate
rounds       Iteration rounds

returns      Derived key


func pbkdf2SHA1(password: String, salt: Data, keyByteCount: Int, rounds: Int) -> Data? {
    return pbkdf2(hash:CCPBKDFAlgorithm(kCCPRFHmacAlgSHA1), password:password, salt:salt,
keyByteCount:keyByteCount, rounds:rounds)
}

func pbkdf2SHA256(password: String, salt: Data, keyByteCount: Int, rounds: Int) -> Data? {
    return pbkdf2(hash:CCPBKDFAlgorithm(kCCPRFHmacAlgSHA256), password:password, salt:salt,
keyByteCount:keyByteCount, rounds:rounds)
}

func pbkdf2SHA512(password: String, salt: Data, keyByteCount: Int, rounds: Int) -> Data? {
    return pbkdf2(hash:CCPBKDFAlgorithm(kCCPRFHmacAlgSHA512), password:password, salt:salt,
keyByteCount:keyByteCount, rounds:rounds)
}

func pbkdf2(hash :CCPBKDFAlgorithm, password: String, salt: Data, keyByteCount: Int, rounds:
Int) -> Data? {
    let passwordData = password.data(using:String.Encoding.utf8)!
    var derivedKeyData = Data(repeating:0, count:keyByteCount)

    let derivationStatus = derivedKeyData.withUnsafeMutableBytes {derivedKeyBytes in
        salt.withUnsafeBytes { saltBytes in

            CCKeyDerivationPBKDF(
                CCPBKDFAlgorithm(kCCPBKDF2),
                password, passwordData.count,
                saltBytes, salt.count,
                hash,
                UInt32(rounds),
```

```
                derivedKeyBytes, derivedKeyData.count)
        }
    }
    if (derivationStatus != 0) {
        print("Error: \(derivationStatus)")
        return nil;
    }

    return derivedKeyData
}
```

```
password     password String
salt         salt Data
keyByteCount number of key bytes to generate
rounds       Iteration rounds

returns      Derived key


func pbkdf2SHA1(password: String, salt: Data, keyByteCount: Int, rounds: Int) -> Data? {
    return pbkdf2(hash:CCPBKDFAlgorithm(kCCPRFHmacAlgSHA1), password:password, salt:salt,
keyByteCount:keyByteCount, rounds:rounds)
}

func pbkdf2SHA256(password: String, salt: Data, keyByteCount: Int, rounds: Int) -> Data? {
    return pbkdf2(hash:CCPBKDFAlgorithm(kCCPRFHmacAlgSHA256), password:password, salt:salt,
keyByteCount:keyByteCount, rounds:rounds)
}

func pbkdf2SHA512(password: String, salt: Data, keyByteCount: Int, rounds: Int) -> Data? {
    return pbkdf2(hash:CCPBKDFAlgorithm(kCCPRFHmacAlgSHA512), password:password, salt:salt,
keyByteCount:keyByteCount, rounds:rounds)
}

func pbkdf2(hash :CCPBKDFAlgorithm, password: String, salt: Data, keyByteCount: Int, rounds:
Int) -> Data? {
    let passwordData = password.data(using:String.Encoding.utf8)!
    var derivedKeyData = Data(repeating:0, count:keyByteCount)

    let derivationStatus = derivedKeyData.withUnsafeMutableBytes {derivedKeyBytes in
        salt.withUnsafeBytes { saltBytes in

            CCKeyDerivationPBKDF(
                CCPBKDFAlgorithm(kCCPBKDF2),
                password, passwordData.count,
                saltBytes, salt.count,
                hash,
                UInt32(rounds),
                derivedKeyBytes, derivedKeyData.count)
        }
    }
    if (derivationStatus != 0) {
        print("Error: \(derivationStatus)")
        return nil;
    }

    return derivedKeyData
}
```

```
password     password String
salt         salt Data
keyByteCount number of key bytes to generate
rounds       Iteration rounds

returns      Derived key


func pbkdf2SHA1(password: String, salt: Data, keyByteCount: Int, rounds: Int) -> Data? {
    return pbkdf2(hash:CCPBKDFAlgorithm(kCCPRFHmacAlgSHA1), password:password, salt:salt,
keyByteCount:keyByteCount, rounds:rounds)
}

func pbkdf2SHA256(password: String, salt: Data, keyByteCount: Int, rounds: Int) -> Data? {
    return pbkdf2(hash:CCPBKDFAlgorithm(kCCPRFHmacAlgSHA256), password:password, salt:salt,
keyByteCount:keyByteCount, rounds:rounds)
}

func pbkdf2SHA512(password: String, salt: Data, keyByteCount: Int, rounds: Int) -> Data? {
    return pbkdf2(hash:CCPBKDFAlgorithm(kCCPRFHmacAlgSHA512), password:password, salt:salt,
keyByteCount:keyByteCount, rounds:rounds)
}

func pbkdf2(hash :CCPBKDFAlgorithm, password: String, salt: Data, keyByteCount: Int, rounds:
Int) -> Data? {
    let passwordData = password.data(using:String.Encoding.utf8)!
    var derivedKeyData = Data(repeating:0, count:keyByteCount)

    let derivationStatus = derivedKeyData.withUnsafeMutableBytes {derivedKeyBytes in
        salt.withUnsafeBytes { saltBytes in

            CCKeyDerivationPBKDF(
                CCPBKDFAlgorithm(kCCPBKDF2),
                password, passwordData.count,
                saltBytes, salt.count,
                hash,
                UInt32(rounds),
                derivedKeyBytes, derivedKeyData.count)
        }
    }
    if (derivationStatus != 0) {
        print("Error: \(derivationStatus)")
        return nil;
    }

    return derivedKeyData
}
```

2Swift 2.3

Swift 3

```
func pbkdf2SHA1(password: String, salt: [UInt8], keyCount: Int, rounds: Int) -> [UInt8]? {
    return pbkdf2(CCPBKDFAlgorithm(kCCPRFHmacAlgSHA1), password:password, salt:salt,
keyCount:keyCount, rounds:UInt32(rounds))
}

func pbkdf2SHA256(password: String, salt: [UInt8], keyCount: Int, rounds: Int) -> [UInt8]? {
    return pbkdf2(CCPBKDFAlgorithm(kCCPRFHmacAlgSHA256), password:password, salt:salt,
```

```
keyCount:keyCount, rounds:UInt32(rounds))
}

func pbkdf2SHA512(password: String, salt: [UInt8], keyCount: Int, rounds: Int) -> [UInt8]? {
    return pbkdf2(CCPBKDFAlgorithm(kCCPRFHmacAlgSHA512), password:password, salt:salt,
keyCount:keyCount, rounds:UInt32(rounds))
}

func pbkdf2(hash :CCPBKDFAlgorithm, password: String, salt: [UInt8], keyCount: Int, rounds:
UInt32!) -> [UInt8]! {
    let derivedKey   = [UInt8](count:keyCount, repeatedValue:0)
    let passwordData = password.dataUsingEncoding(NSUTF8StringEncoding)!

    let derivationStatus = CCKeyDerivationPBKDF(
        CCPBKDFAlgorithm(kCCPBKDF2),
        UnsafePointer<Int8>(passwordData.bytes), passwordData.length,
        UnsafePointer<UInt8>(salt), salt.count,
        CCPseudoRandomAlgorithm(hash),
        rounds,
        UnsafeMutablePointer<UInt8>(derivedKey),
        derivedKey.count)


    if (derivationStatus != 0) {
        print("Error: \(derivationStatus)")
        return nil;
    }

    return derivedKey
}
```

```
func pbkdf2SHA1(password: String, salt: [UInt8], keyCount: Int, rounds: Int) -> [UInt8]? {
    return pbkdf2(CCPBKDFAlgorithm(kCCPRFHmacAlgSHA1), password:password, salt:salt,
keyCount:keyCount, rounds:UInt32(rounds))
}

func pbkdf2SHA256(password: String, salt: [UInt8], keyCount: Int, rounds: Int) -> [UInt8]? {
    return pbkdf2(CCPBKDFAlgorithm(kCCPRFHmacAlgSHA256), password:password, salt:salt,
keyCount:keyCount, rounds:UInt32(rounds))
}

func pbkdf2SHA512(password: String, salt: [UInt8], keyCount: Int, rounds: Int) -> [UInt8]? {
    return pbkdf2(CCPBKDFAlgorithm(kCCPRFHmacAlgSHA512), password:password, salt:salt,
keyCount:keyCount, rounds:UInt32(rounds))
}

func pbkdf2(hash :CCPBKDFAlgorithm, password: String, salt: [UInt8], keyCount: Int, rounds:
UInt32!) -> [UInt8]! {
    let derivedKey   = [UInt8](count:keyCount, repeatedValue:0)
    let passwordData = password.dataUsingEncoding(NSUTF8StringEncoding)!

    let derivationStatus = CCKeyDerivationPBKDF(
        CCPBKDFAlgorithm(kCCPBKDF2),
        UnsafePointer<Int8>(passwordData.bytes), passwordData.length,
        UnsafePointer<UInt8>(salt), salt.count,
        CCPseudoRandomAlgorithm(hash),
        rounds,
        UnsafeMutablePointer<UInt8>(derivedKey),
        derivedKey.count)
```

```
    if (derivationStatus != 0) {
        print("Error: \(derivationStatus)")
        return nil;
    }

    return derivedKey
}
```

```
func pbkdf2SHA1(password: String, salt: [UInt8], keyCount: Int, rounds: Int) -> [UInt8]? {
    return pbkdf2(CCPBKDFAlgorithm(kCCPRFHmacAlgSHA1), password:password, salt:salt,
keyCount:keyCount, rounds:UInt32(rounds))
}

func pbkdf2SHA256(password: String, salt: [UInt8], keyCount: Int, rounds: Int) -> [UInt8]? {
    return pbkdf2(CCPBKDFAlgorithm(kCCPRFHmacAlgSHA256), password:password, salt:salt,
keyCount:keyCount, rounds:UInt32(rounds))
}

func pbkdf2SHA512(password: String, salt: [UInt8], keyCount: Int, rounds: Int) -> [UInt8]? {
    return pbkdf2(CCPBKDFAlgorithm(kCCPRFHmacAlgSHA512), password:password, salt:salt,
keyCount:keyCount, rounds:UInt32(rounds))
}

func pbkdf2(hash :CCPBKDFAlgorithm, password: String, salt: [UInt8], keyCount: Int, rounds:
UInt32!) -> [UInt8]! {
    let derivedKey   = [UInt8](count:keyCount, repeatedValue:0)
    let passwordData = password.dataUsingEncoding(NSUTF8StringEncoding)!

    let derivationStatus = CCKeyDerivationPBKDF(
        CCPBKDFAlgorithm(kCCPBKDF2),
        UnsafePointer<Int8>(passwordData.bytes), passwordData.length,
        UnsafePointer<UInt8>(salt), salt.count,
        CCPseudoRandomAlgorithm(hash),
        rounds,
        UnsafeMutablePointer<UInt8>(derivedKey),
        derivedKey.count)


    if (derivationStatus != 0) {
        print("Error: \(derivationStatus)")
        return nil;
    }

    return derivedKey
}
```

Swift 2.3

Swift 3

```
func pbkdf2SHA1Calibrate(password:String, salt:[UInt8], msec:Int) -> UInt32 {
    let actualRoundCount: UInt32 = CCCalibratePBKDF(
        CCPBKDFAlgorithm(kCCPBKDF2),
        password.utf8.count,
        salt.count,
        CCPseudoRandomAlgorithm(kCCPRFHmacAlgSHA1),
        kCCKeySizeAES256,
```

```
        UInt32(msec));
    return actualRoundCount
}
```

```
func pbkdf2SHA1Calibrate(password:String, salt:[UInt8], msec:Int) -> UInt32 {
    let actualRoundCount: UInt32 = CCCalibratePBKDF(
        CCPBKDFAlgorithm(kCCPBKDF2),
        password.utf8.count,
        salt.count,
        CCPseudoRandomAlgorithm(kCCPRFHmacAlgSHA1),
        kCCKeySizeAES256,
        UInt32(msec));
    return actualRoundCount
}
```

10094339

Swift 3

PRF。

。

```
password Sample password.
salt     Sample salt.
msec     Targeted duration we want to achieve for a key derivation.

returns  The number of iterations to use for the desired processing time.


func pbkdf2SHA1Calibrate(password: String, salt: Data, msec: Int) -> UInt32 {
    let actualRoundCount: UInt32 = CCCalibratePBKDF(
        CCPBKDFAlgorithm(kCCPBKDF2),
        password.utf8.count,
        salt.count,
        CCPseudoRandomAlgorithm(kCCPRFHmacAlgSHA1),
        kCCKeySizeAES256,
        UInt32(msec));
    return actualRoundCount
}
```

```
password Sample password.
salt     Sample salt.
msec     Targeted duration we want to achieve for a key derivation.

returns  The number of iterations to use for the desired processing time.


func pbkdf2SHA1Calibrate(password: String, salt: Data, msec: Int) -> UInt32 {
    let actualRoundCount: UInt32 = CCCalibratePBKDF(
        CCPBKDFAlgorithm(kCCPBKDF2),
        password.utf8.count,
        salt.count,
        CCPseudoRandomAlgorithm(kCCPRFHmacAlgSHA1),
        kCCKeySizeAES256,
        UInt32(msec));
```

```
    return actualRoundCount
}
```

```
password Sample password.
salt     Sample salt.
msec     Targeted duration we want to achieve for a key derivation.

returns  The number of iterations to use for the desired processing time.


func pbkdf2SHA1Calibrate(password: String, salt: Data, msec: Int) -> UInt32 {
    let actualRoundCount: UInt32 = CCCalibratePBKDF(
        CCPBKDFAlgorithm(kCCPBKDF2),
        password.utf8.count,
        salt.count,
        CCPseudoRandomAlgorithm(kCCPRFHmacAlgSHA1),
        kCCKeySizeAES256,
        UInt32(msec));
    return actualRoundCount
}
```

PBKDF2 https://riptutorial.com/zh-TW/swift/topic/7053/pbkdf2

7: RxSwift

Examples

RxSwift

FRP。

Observable。 FRP Observable。

Observable - .Next .Success .Error。

```
--(1)--(2)--(3)|-->
```

Int。 .Next。

```
--(1)--(2)--(3)|-->
```

.ErrorObservable。

1. RxSwift。 。
2. RxSwift Slack。
3. RxMarbles。
4. 。

*RxSwift*Observable

```
import RxSwift

let intObservale = Observable.just(123) // Observable<Int>
let stringObservale = Observable.just("RxSwift") // Observable<String>
let doubleObservale = Observable.just(3.14) // Observable<Double>
```

。 。 。

Observable。

```
import RxSwift

let intObservale = Observable.just(123) // Observable<Int>
let stringObservale = Observable.just("RxSwift") // Observable<String>
let doubleObservale = Observable.just(3.14) // Observable<Double>
```

```
import RxSwift

let intObservale = Observable.just(123) // Observable<Int>
let stringObservale = Observable.just("RxSwift") // Observable<String>
```

```
let doubleObservale = Observable.just(3.14) // Observable<Double>
```

.NextsubscribeNext

```
import RxSwift

let intObservale = Observable.just(123) // Observable<Int>
let stringObservale = Observable.just("RxSwift") // Observable<String>
let doubleObservale = Observable.just(3.14) // Observable<Double>
```


```
import RxSwift

let intObservale = Observable.just(123) // Observable<Int>
let stringObservale = Observable.just("RxSwift") // Observable<String>
let doubleObservale = Observable.just(3.14) // Observable<Double>
```


Observable。 Observable<SomeResultType>。


```
import RxSwift

let intObservale = Observable.just(123) // Observable<Int>
let stringObservale = Observable.just("RxSwift") // Observable<String>
let doubleObservale = Observable.just(3.14) // Observable<Double>
```


。

addDisposableTodisposeBagdispose。 。

。

1. disposeBagaddDisposableTo。
2. takeUntil。

DisposeBag。

```
let bag = DisposeBag()
Observable.just(1).subscribeNext {
    print($0)
}.addDisposableTo(bag)
```

DisposeBag *RxSwift*NSObject + Rx。


```
let bag = DisposeBag()
Observable.just(1).subscribeNext {
    print($0)
}.addDisposableTo(bag)
```


selftakeUntil(rx_deallocated)

```
let bag = DisposeBag()
Observable.just(1).subscribeNext {
    print($0)
```

```
}.addDisposableTo(bag)
```

```
Observable.combineLatest(firstName.rx_text, lastName.rx_text) { $0 + " " + $1 }
.map { "Greetings, \($0)" }
.bindTo(greetingLabel.rx_text)
```

ObservablescombineLatestObservable。 UITextField"Greetings, \($0)"UILabel。

UITableViewUICollectionView

```
Observable.combineLatest(firstName.rx_text, lastName.rx_text) { $0 + " " + $1 }
.map { "Greetings, \($0)" }
.bindTo(greetingLabel.rx_text)
```

cellForRowAtIndexPathRx。 RxnumberOfRowsAtIndexPath。

RxCocoaControlEvents

RxSwift。

RxCocoa。 UIObservable 。 ControlEventObservable ControlProperties 。 Observable

- 。
- Complete。
- MainScheduler.instance 。

```
button.rx_tap.subscribeNext { _ in   // control event
    print("User tapped the button!")
}.addDisposableTo(bag)

textField.rx_text.subscribeNext { text in // control property
    print("The textfield contains: \(text)")
}.addDisposableTo(bag)
// notice that ControlProperty generates .Next event on subscription
// In this case, the log will display
// "The textfield contains: "
// at the very start of the app.
```

Rx@IBAction。 viewDidLoadUI。

。 。 -

button.rx_tap ControlEvent

```
button.rx_tap.subscribeNext { _ in   // control event
    print("User tapped the button!")
}.addDisposableTo(bag)

textField.rx_text.subscribeNext { text in // control property
    print("The textfield contains: \(text)")
}.addDisposableTo(bag)
// notice that ControlProperty generates .Next event on subscription
```

```
// In this case, the log will display
// "The textfield contains: "
// at the very start of the app.
```

◦ withLatestFromtextFieldwithLatestFrom ◦

```
button.rx_tap.subscribeNext { _ in   // control event
    print("User tapped the button!")
}.addDisposableTo(bag)

textField.rx_text.subscribeNext { text in // control property
    print("The textfield contains: \(text)")
}.addDisposableTo(bag)
// notice that ControlProperty generates .Next event on subscription
// In this case, the log will display
// "The textfield contains: "
// at the very start of the app.
```

◦

Observablemapfiltermap◦ validateEmail◦

```
button.rx_tap.subscribeNext { _ in   // control event
    print("User tapped the button!")
}.addDisposableTo(bag)

textField.rx_text.subscribeNext { text in // control property
    print("The textfield contains: \(text)")
}.addDisposableTo(bag)
// notice that ControlProperty generates .Next event on subscription
// In this case, the log will display
// "The textfield contains: "
// at the very start of the app.
```

.....Bool◦

RxSwift https://riptutorial.com/zh-TW/swift/topic/4890/rxswift

8: Swift Advance

`map` `flatMap` `filter``reduce`ArrayDictionary。 。

Examples

```
struct User {
    var name: String
    var age: Int
    var country: String?
}

//User's information
let user1 = User(name: "John", age: 24, country: "USA")
let user2 = User(name: "Chan", age: 20, country: nil)
let user3 = User(name: "Morgan", age: 30, country: nil)
let user4 = User(name: "Rachel", age: 20, country: "UK")
let user5 = User(name: "Katie", age: 23, country: "USA")
let user6 = User(name: "David", age: 35, country: "USA")
let user7 = User(name: "Bob",age: 22, country: nil)

//User's array list
let arrUser = [user1, user2, user3, user4, user5, user6, user7]
```

map。 map。

```
struct User {
    var name: String
    var age: Int
    var country: String?
}

//User's information
let user1 = User(name: "John", age: 24, country: "USA")
let user2 = User(name: "Chan", age: 20, country: nil)
let user3 = User(name: "Morgan", age: 30, country: nil)
let user4 = User(name: "Rachel", age: 20, country: "UK")
let user5 = User(name: "Katie", age: 23, country: "USA")
let user6 = User(name: "David", age: 35, country: "USA")
let user7 = User(name: "Bob",age: 22, country: nil)

//User's array list
let arrUser = [user1, user2, user3, user4, user5, user6, user7]
```

。

```
struct User {
    var name: String
    var age: Int
    var country: String?
}

//User's information
let user1 = User(name: "John", age: 24, country: "USA")
```

```
let user2 = User(name: "Chan", age: 20, country: nil)
let user3 = User(name: "Morgan", age: 30, country: nil)
let user4 = User(name: "Rachel", age: 20, country: "UK")
let user5 = User(name: "Katie", age: 23, country: "USA")
let user6 = User(name: "David", age: 35, country: "USA")
let user7 = User(name: "Bob",age: 22, country: nil)

//User's array list
let arrUser = [user1, user2, user3, user4, user5, user6, user7]
```

filterincludeArray。

```
struct User {
    var name: String
    var age: Int
    var country: String?
}

//User's information
let user1 = User(name: "John", age: 24, country: "USA")
let user2 = User(name: "Chan", age: 20, country: nil)
let user3 = User(name: "Morgan", age: 30, country: nil)
let user4 = User(name: "Rachel", age: 20, country: "UK")
let user5 = User(name: "Katie", age: 23, country: "USA")
let user6 = User(name: "David", age: 35, country: "USA")
let user7 = User(name: "Bob",age: 22, country: nil)

//User's array list
let arrUser = [user1, user2, user3, user4, user5, user6, user7]
```

reduce。

Swift 2.3 -

```
struct User {
    var name: String
    var age: Int
    var country: String?
}

//User's information
let user1 = User(name: "John", age: 24, country: "USA")
let user2 = User(name: "Chan", age: 20, country: nil)
let user3 = User(name: "Morgan", age: 30, country: nil)
let user4 = User(name: "Rachel", age: 20, country: "UK")
let user5 = User(name: "Katie", age: 23, country: "USA")
let user6 = User(name: "David", age: 35, country: "USA")
let user7 = User(name: "Bob",age: 22, country: nil)

//User's array list
let arrUser = [user1, user2, user3, user4, user5, user6, user7]
```

3 -

```
struct User {
    var name: String
    var age: Int
```

```
    var country: String?
}

//User's information
let user1 = User(name: "John", age: 24, country: "USA")
let user2 = User(name: "Chan", age: 20, country: nil)
let user3 = User(name: "Morgan", age: 30, country: nil)
let user4 = User(name: "Rachel", age: 20, country: "UK")
let user5 = User(name: "Katie", age: 23, country: "USA")
let user6 = User(name: "David", age: 35, country: "USA")
let user7 = User(name: "Bob",age: 22, country: nil)

//User's array list
let arrUser = [user1, user2, user3, user4, user5, user6, user7]
```

flatMap。nil。 -

。 flatMap -

```
let arrStateName = [["Alaska", "Iowa", "Missouri", "New Mexico"], ["New York", "Texas",
"Washington", "Maryland"], ["New Jersey", "Virginia", "Florida", "Colorado"]]
```

```
let arrStateName = [["Alaska", "Iowa", "Missouri", "New Mexico"], ["New York", "Texas",
"Washington", "Maryland"], ["New Jersey", "Virginia", "Florida", "Colorado"]]
```

flatten。

```
let arrStateName = [["Alaska", "Iowa", "Missouri", "New Mexico"], ["New York", "Texas",
"Washington", "Maryland"], ["New Jersey", "Virginia", "Florida", "Colorado"]]
```

Swift Advance https://riptutorial.com/zh-TW/swift/topic/9279/swift-advance

9: SwiftNSRegularExpression

```
*?+[(){}^$|\./
```

Examples

String

```
extension String {
    func matchesPattern(pattern: String) -> Bool {
        do {
            let regex = try NSRegularExpression(pattern: pattern,
                                                options: NSRegularExpressionOptions(rawValue:
0))
            let range: NSRange = NSMakeRange(0, self.characters.count)
            let matches = regex.matchesInString(self, options: NSMatchingOptions(), range:
range)
            return matches.count > 0
        } catch _ {
            return false
        }
    }
}

// very basic examples - check for specific strings
dump("Pinkman".matchesPattern("(White|Pinkman|Goodman|Schrader|Fring)"))

// using character groups to check for similar-sounding impressionist painters
dump("Monet".matchesPattern("(M[oa]net)"))
dump("Manet".matchesPattern("(M[oa]net)"))
dump("Money".matchesPattern("(M[oa]net)"))     // false

// check surname is in list
dump("Skyler White".matchesPattern("\\w+ (White|Pinkman|Goodman|Schrader|Fring)"))

// check if string looks like a UK stock ticker
dump("VOD.L".matchesPattern("[A-Z]{2,3}\\.L"))
dump("BP.L".matchesPattern("[A-Z]{2,3}\\.L"))

// check entire string is printable ASCII characters
dump("tab\tformatted text".matchesPattern("^[\u{0020}-\u{007e}]*$"))

// Unicode example: check if string contains a playing card suit
dump("♠".matchesPattern("[\u{2660}-\u{2667}]"))
dump("♡".matchesPattern("[\u{2660}-\u{2667}]"))
dump("".matchesPattern("[\u{2660}-\u{2667}]"))    // false

// NOTE: regex needs Unicode-escaped characters
dump("♣".matchesPattern("♣"))           // does NOT work
```

。

```
extension String {
    func matchesPattern(pattern: String) -> Bool {
```

```
        do {
            let regex = try NSRegularExpression(pattern: pattern,
                                                options: NSRegularExpressionOptions(rawValue:
0))
            let range: NSRange = NSMakeRange(0, self.characters.count)
            let matches = regex.matchesInString(self, options: NSMatchingOptions(), range:
range)
            return matches.count > 0
        } catch _ {
            return false
        }
    }
}

// very basic examples - check for specific strings
dump("Pinkman".matchesPattern("(White|Pinkman|Goodman|Schrader|Fring)"))

// using character groups to check for similar-sounding impressionist painters
dump("Monet".matchesPattern("(M[oa]net)"))
dump("Manet".matchesPattern("(M[oa]net)"))
dump("Money".matchesPattern("(M[oa]net)"))     // false

// check surname is in list
dump("Skyler White".matchesPattern("\\w+ (White|Pinkman|Goodman|Schrader|Fring)"))

// check if string looks like a UK stock ticker
dump("VOD.L".matchesPattern("[A-Z]{2,3}\\.L"))
dump("BP.L".matchesPattern("[A-Z]{2,3}\\.L"))

// check entire string is printable ASCII characters
dump("tab\tformatted text".matchesPattern("^[\u{0020}-\u{007e}]*$"))

// Unicode example: check if string contains a playing card suit
dump("♠□".matchesPattern("[\u{2660}-\u{2667}]"))
dump("♡".matchesPattern("[\u{2660}-\u{2667}]"))
dump("□".matchesPattern("[\u{2660}-\u{2667}]"))    // false

// NOTE: regex needs Unicode-escaped characters
dump("♣□".matchesPattern("♣□"))           // does NOT work
```

Swift。

```
let letters = "abcdefg"
let pattern = "[a,b,c]"
let regEx = try NSRegularExpression(pattern: pattern, options: [])
let nsString = letters as NSString
let matches = regEx.matches(in: letters, options: [], range: NSMakeRange(0, nsString.length))
let output = matches.map {nsString.substring(with: $0.range)}
//output = ["a", "b", "c"]
```

NSString。

do catch

```
let letters = "abcdefg"
let pattern = "[a,b,c]"
let regEx = try NSRegularExpression(pattern: pattern, options: [])
let nsString = letters as NSString
```

```
let matches = regEx.matches(in: letters, options: [], range: NSMakeRange(0, nsString.length))
let output = matches.map {nsString.substring(with: $0.range)}
//output = ["a", "b", "c"]
```

。

。

。

```
var money = "¢¥€£$¥€£¢"
let pattern = "¢"
do {
    let regEx = try NSRegularExpression (pattern: pattern, options: [])
    let nsString = money as NSString
    let range = NSMakeRange(0, nsString.length)
    let correct$ = regEx.stringByReplacingMatches(in: money, options: .withTransparentBounds,
range: range, withTemplate: "$")
} catch let error as NSError {
    print("Matching failed")
}
//correct$ = "$¥€£$¥€£$"
```

\. becomes \\.

```
(){}[]/\+*$>.|^?
```

```
(){}[]/\+*$>.|^?
```

。

```
var validDate = false

let numbers = "35/12/2016"
let usPattern =  "^(0[1-9]|1[012])[-/.](0[1-9]|[12][0-9]|3[01])[-/.](19|20)\\d\\d$"
let ukPattern = "^(0[1-9]|[12][0-9]|3[01])[-/](0[1-9]|1[012])[-/](19|20)\\d\\d$"
do {
    let regEx = try NSRegularExpression(pattern: ukPattern, options: [])
    let nsString = numbers as NSString
    let matches = regEx.matches(in: numbers, options: [], range: NSMakeRange(0,
nsString.length))

    if matches.count > 0 {
        validDate = true
    }

    validDate

} catch let error as NSError {
    print("Matching failed")
}
//output = false
```

NSRegularExpression

```
func isValidEmail(email: String) -> Bool {

    let emailRegEx = "[A-Z0-9a-z._%+-]+@[A-Za-z0-9.-]+\\.[A-Za-z]{2,4}"

    let emailTest = NSPredicate(format:"SELF MATCHES %@", emailRegEx)
    return emailTest.evaluate(with: email)
}
```

String

```
func isValidEmail(email: String) -> Bool {

    let emailRegEx = "[A-Z0-9a-z._%+-]+@[A-Za-z0-9.-]+\\.[A-Za-z]{2,4}"

    let emailTest = NSPredicate(format:"SELF MATCHES %@", emailRegEx)
    return emailTest.evaluate(with: email)
}
```

SwiftNSRegularExpression https://riptutorial.com/zh-TW/swift/topic/5763/swiftnsregularexpression

10: Swift

Examples

`Person`

```
struct Person {
    let name: String
    let birthYear: Int?
}
```

`Person(s)`

```
struct Person {
    let name: String
    let birthYear: Int?
}
```

Person`nameString`。

```
struct Person {
    let name: String
    let birthYear: Int?
}
```

```
let numbers = [3, 1, 4, 1, 5]
// non-functional
for (index, element) in numbers.enumerate() {
    print(index, element)
}

// functional
numbers.enumerate().map { (index, element) in
    print((index, element))
}
```

// 。

```
/// Projection
var newReleases = [
    [
        "id": 70111470,
        "title": "Die Hard",
        "boxart": "http://cdn-0.nflximg.com/images/2891/DieHard.jpg",
        "uri": "http://api.netflix.com/catalog/titles/movies/70111470",
        "rating": [4.0],
        "bookmark": []
    ],
    [
        "id": 654356453,
        "title": "Bad Boys",
        "boxart": "http://cdn-0.nflximg.com/images/2891/BadBoys.jpg",
```

```
        "uri": "http://api.netflix.com/catalog/titles/movies/70111470",
        "rating": [5.0],
        "bookmark": [[ "id": 432534, "time": 65876586 ]]
    ],
    [
        "id": 65432445,
        "title": "The Chamber",
        "boxart": "http://cdn-0.nflximg.com/images/2891/TheChamber.jpg",
        "uri": "http://api.netflix.com/catalog/titles/movies/70111470",
        "rating": [4.0],
        "bookmark": []
    ],
    [
        "id": 675465,
        "title": "Fracture",
        "boxart": "http://cdn-0.nflximg.com/images/2891/Fracture.jpg",
        "uri": "http://api.netflix.com/catalog/titles/movies/70111470",
        "rating": [5.0],
        "bookmark": [[ "id": 432534, "time": 65876586 ]]
    ]
]


var videoAndTitlePairs = [[String: AnyObject]]()
newReleases.map { e in
    videoAndTitlePairs.append(["id": e["id"] as! Int, "title": e["title"] as! String])
}

print(videoAndTitlePairs)
```

```
var newReleases = [
    [
        "id": 70111470,
        "title": "Die Hard",
        "boxart": "http://cdn-0.nflximg.com/images/2891/DieHard.jpg",
        "uri": "http://api.netflix.com/catalog/titles/movies/70111470",
        "rating": 4.0,
        "bookmark": []
    ],
    [
        "id": 654356453,
        "title": "Bad Boys",
        "boxart": "http://cdn-0.nflximg.com/images/2891/BadBoys.jpg",
        "uri": "http://api.netflix.com/catalog/titles/movies/70111470",
        "rating": 5.0,
        "bookmark": [[ "id": 432534, "time": 65876586 ]]
    ],
    [
        "id": 65432445,
        "title": "The Chamber",
        "boxart": "http://cdn-0.nflximg.com/images/2891/TheChamber.jpg",
        "uri": "http://api.netflix.com/catalog/titles/movies/70111470",
        "rating": 4.0,
        "bookmark": []
    ],
    [
        "id": 675465,
        "title": "Fracture",
        "boxart": "http://cdn-0.nflximg.com/images/2891/Fracture.jpg",
```

```
        "uri": "http://api.netflix.com/catalog/titles/movies/70111470",
        "rating": 5.0,
        "bookmark": [[ "id": 432534, "time": 65876586 ]]
    ]
]

var videos1 = [[String: AnyObject]]()
/**
 *  Filtering using map
 */
newReleases.map { e in
    if e["rating"] as! Float == 5.0 {
        videos1.append(["id": e["id"] as! Int, "title": e["title"] as! String])
    }
}

print(videos1)

var videos2 = [[String: AnyObject]]()
/**
 *  Filtering using filter and chaining
 */
newReleases
    .filter{ e in
        e["rating"] as! Float == 5.0
    }
    .map { e in
    videos2.append(["id": e["id"] as! Int, "title": e["title"] as! String])
}

print(videos2)
```

- Swift。 。

```
struct Painter {
    enum Type { case Impressionist, Expressionist, Surrealist, Abstract, Pop }
    var firstName: String
    var lastName: String
    var type: Type
}

let painters = [
    Painter(firstName: "Claude", lastName: "Monet", type: .Impressionist),
    Painter(firstName: "Edgar", lastName: "Degas", type: .Impressionist),
    Painter(firstName: "Egon", lastName: "Schiele", type: .Expressionist),
    Painter(firstName: "George", lastName: "Grosz", type: .Expressionist),
    Painter(firstName: "Mark", lastName: "Rothko", type: .Abstract),
    Painter(firstName: "Jackson", lastName: "Pollock", type: .Abstract),
    Painter(firstName: "Pablo", lastName: "Picasso", type: .Surrealist),
    Painter(firstName: "Andy", lastName: "Warhol", type: .Pop)
]

// list the expressionists
dump(painters.filter({$0.type == .Expressionist}))

// count the expressionists
dump(painters.filter({$0.type == .Expressionist}).count)
// prints "2"

// combine filter and map for more complex operations, for example listing all
```

```
// non-impressionist and non-expressionists by surname
dump(painters.filter({$0.type != .Impressionist && $0.type != .Expressionist})
    .map({$0.lastName}).joinWithSeparator(", "))
// prints "Rothko, Pollock, Picasso, Warhol"
```

Swift https://riptutorial.com/zh-TW/swift/topic/2948/swift

11: Swift

Examples

Swift

Swift

```
mkdir AwesomeProject
cd AwesomeProject
```

Git

```
mkdir AwesomeProject
cd AwesomeProject
```

- CLI。

```
mkdir AwesomeProject
cd AwesomeProject
```

- *main.swift*。

```
mkdir AwesomeProject
cd AwesomeProject
```

AwesomeProject.swift。

*Sources*Swift。

Package.swift

```
mkdir AwesomeProject
cd AwesomeProject
```

Git

```
mkdir AwesomeProject
cd AwesomeProject
```

Git。

```
mkdir AwesomeProject
cd AwesomeProject
```

.build / debug。

- “SomeOtherPackage”*Package.swift*

```
mkdir AwesomeProject
cd AwesomeProject
```

Swift Package Manager。

Swift https://riptutorial.com/zh-TW/swift/topic/5144/swift

12: Swift

。 。 。

Examples

。 。 。 On2。 。

```
extension Array where Element: Comparable {

func insertionSort() -> Array<Element> {

    //check for trivial case
    guard self.count > 1 else {
        return self
    }

    //mutated copy
    var output: Array<Element> = self

    for primaryindex in 0..<output.count {

        let key = output[primaryindex]
        var secondaryindex = primaryindex

        while secondaryindex > -1 {
            if key < output[secondaryindex] {

                //move to correct position
                output.remove(at: secondaryindex + 1)
                output.insert(key, at: secondaryindex)
            }
            secondaryindex -= 1
        }
    }

    return output
}
}
```

。 。 。 On2。

```
extension Array where Element: Comparable {

func bubbleSort() -> Array<Element> {

    //check for trivial case
    guard self.count > 1 else {
        return self
    }

    //mutated copy
    var output: Array<Element> = self

    for primaryIndex in 0..<self.count {
```

```
        let passes = (output.count - 1) - primaryIndex

        //"half-open" range operator
        for secondaryIndex in 0..<passes {
            let key = output[secondaryIndex]

            //compare / swap positions
            if (key > output[secondaryIndex + 1]) {
                swap(&output[secondaryIndex], &output[secondaryIndex + 1])
            }
        }
    }

    return output
}

}
```

。 。 。 On2。 。

```
extension Array where Element: Comparable {

func bubbleSort() -> Array<Element> {

    //check for trivial case
    guard self.count > 1 else {
        return self
    }

    //mutated copy
    var output: Array<Element> = self

    for primaryIndex in 0..<self.count {
        let passes = (output.count - 1) - primaryIndex

        //"half-open" range operator
        for secondaryIndex in 0..<passes {
            let key = output[secondaryIndex]

            //compare / swap positions
            if (key > output[secondaryIndex + 1]) {
                swap(&output[secondaryIndex], &output[secondaryIndex + 1])
            }
        }
    }

    return output
}

}
```

。 。 minmin。 。 On2 - 。

func selectionSort - > Array {//self.count> 1 else {return self}

```
extension Array where Element: Comparable {

func bubbleSort() -> Array<Element> {
```

```
    //check for trivial case
    guard self.count > 1 else {
        return self
    }

    //mutated copy
    var output: Array<Element> = self

    for primaryIndex in 0..<self.count {
        let passes = (output.count - 1) - primaryIndex

        //"half-open" range operator
        for secondaryIndex in 0..<passes {
            let key = output[secondaryIndex]

            //compare / swap positions
            if (key > output[secondaryIndex + 1]) {
                swap(&output[secondaryIndex], &output[secondaryIndex + 1])
            }
        }
    }

    return output
}

}
```

- On log n

Quicksort。 On log n。 。 Quicksort。 Quicksort。

pivot。

pivotpivotpivot。 。 。

。

mutating func quickSort - > Array {

```
extension Array where Element: Comparable {

func bubbleSort() -> Array<Element> {

    //check for trivial case
    guard self.count > 1 else {
        return self
    }

    //mutated copy
    var output: Array<Element> = self

    for primaryIndex in 0..<self.count {
        let passes = (output.count - 1) - primaryIndex

        //"half-open" range operator
        for secondaryIndex in 0..<passes {
            let key = output[secondaryIndex]
```

```
            //compare / swap positions
            if (key > output[secondaryIndex + 1]) {
                swap(&output[secondaryIndex], &output[secondaryIndex + 1])
            }
        }
    }

    return output
}

}
```

。 。 minmin。 。 On2 - 。

```
func selectionSort() -> Array<Element> {
    //check for trivial case
    guard self.count > 1 else {
        return self
    }

    //mutated copy
    var output: Array<Element> = self

    for primaryindex in 0..<output.count {
        var minimum = primaryindex
        var secondaryindex = primaryindex + 1

        while secondaryindex < output.count {
            //store lowest value as minimum
            if output[minimum] > output[secondaryindex] {
                minimum = secondaryindex
            }
            secondaryindex += 1
        }

        //swap minimum value with array iteration
        if primaryindex != minimum {
            swap(&output[primaryindex], &output[minimum])
        }
    }

    return output
}
```

。 。 。 n。 Big O Notation。

- **On** 。 。
- **On2** nn。 。 On2On3.....
- **Olog n**n。 。

- On log n

Quicksort。 On log n。 。 Quicksort。 Quicksort。

1. pivot。

2. pivotpivotpivot。 。 。

3. 。

```
mutating func quickSort() -> Array<Element> {

func qSort(start startIndex: Int, _ pivot: Int) {

    if (startIndex < pivot) {
        let iPivot = qPartition(start: startIndex, pivot)
        qSort(start: startIndex, iPivot - 1)
        qSort(start: iPivot + 1, pivot)
    }
}
qSort(start: 0, self.endIndex - 1)
return self
```

}

mutating func qPartitionstart startIndexInt_ pivotInt - > Int {

```
mutating func quickSort() -> Array<Element> {

func qSort(start startIndex: Int, _ pivot: Int) {

    if (startIndex < pivot) {
        let iPivot = qPartition(start: startIndex, pivot)
        qSort(start: startIndex, iPivot - 1)
        qSort(start: iPivot + 1, pivot)
    }
}
qSort(start: 0, self.endIndex - 1)
return self
```

```
mutating func quickSort() -> Array<Element> {

func qSort(start startIndex: Int, _ pivot: Int) {

    if (startIndex < pivot) {
        let iPivot = qPartition(start: startIndex, pivot)
        qSort(start: startIndex, iPivot - 1)
        qSort(start: iPivot + 1, pivot)
    }
}
qSort(start: 0, self.endIndex - 1)
return self
```

。 。 。 。 。

```
//
//  GraphFactory.swift
//  SwiftStructures
//
//  Created by Wayne Bishop on 6/7/14.
//  Copyright (c) 2014 Arbutus Software Inc. All rights reserved.
//
```

```
import Foundation


public class SwiftGraph {


    //declare a default directed graph canvas
    private var canvas: Array<Vertex>
    public var isDirected: Bool


    init() {
        canvas = Array<Vertex>()
        isDirected = true
    }


    //create a new vertex
    func addVertex(key: String) -> Vertex {


        //set the key
        let childVertex: Vertex = Vertex()
        childVertex.key = key


        //add the vertex to the graph canvas
        canvas.append(childVertex)


        return childVertex
    }



    //add edge to source vertex
    func addEdge(source: Vertex, neighbor: Vertex, weight: Int) {


        //create a new edge
        let newEdge = Edge()


        //establish the default properties
        newEdge.neighbor = neighbor
        newEdge.weight = weight
        source.neighbors.append(newEdge)


        print("The neighbor of vertex: \(source.key as String!) is \(neighbor.key as
String!)..")


        //check condition for an undirected graph
        if isDirected == false {


           //create a new reversed edge
           let reverseEdge = Edge()
```

```
            //establish the reversed properties
            reverseEdge.neighbor = source
            reverseEdge.weight = weight
            neighbor.neighbors.append(reverseEdge)

            print("The neighbor of vertex: \(neighbor.key as String!) is \(source.key as
String!)..")

        }


    }





    /* reverse the sequence of paths given the shortest path.
       process analagous to reversing a linked list. */

    func reversePath(_ head: Path!, source: Vertex) -> Path! {


        guard head != nil else {
            return head
        }

        //mutated copy
        var output = head


        var current: Path! = output
        var prev: Path!
        var next: Path!


        while(current != nil) {
            next = current.previous
            current.previous = prev
            prev = current
            current = next
        }


        //append the source path to the sequence
        let sourcePath: Path = Path()

        sourcePath.destination = source
        sourcePath.previous = prev
        sourcePath.total = nil

        output = sourcePath


        return output

    }
```

```
//process Dijkstra's shortest path algorthim
func processDijkstra(_ source: Vertex, destination: Vertex) -> Path? {


    var frontier: Array<Path> = Array<Path>()
    var finalPaths: Array<Path> = Array<Path>()


    //use source edges to create the frontier
    for e in source.neighbors {

        let newPath: Path = Path()


        newPath.destination = e.neighbor
        newPath.previous = nil
        newPath.total = e.weight


        //add the new path to the frontier
        frontier.append(newPath)

    }


    //construct the best path
    var bestPath: Path = Path()


    while frontier.count != 0 {

        //support path changes using the greedy approach
        bestPath = Path()
        var pathIndex: Int = 0


        for x in 0..<frontier.count {

            let itemPath: Path = frontier[x]

            if  (bestPath.total == nil) || (itemPath.total < bestPath.total) {
                bestPath = itemPath
                pathIndex = x
            }

        }



        //enumerate the bestPath edges
        for e in bestPath.destination.neighbors {

            let newPath: Path = Path()

            newPath.destination = e.neighbor
            newPath.previous = bestPath
            newPath.total = bestPath.total + e.weight


            //add the new path to the frontier
            frontier.append(newPath)
```

```
        }


        //preserve the bestPath
        finalPaths.append(bestPath)


        //remove the bestPath from the frontier
        //frontier.removeAtIndex(pathIndex) - Swift2
        frontier.remove(at: pathIndex)



    } //end while



    //establish the shortest path as an optional
    var shortestPath: Path! = Path()


    for itemPath in finalPaths {

        if (itemPath.destination.key == destination.key) {

            if  (shortestPath.total == nil) || (itemPath.total < shortestPath.total) {
                shortestPath = itemPath
            }

        }

    }


    return shortestPath

}



///an optimized version of Dijkstra's shortest path algorthim
func processDijkstraWithHeap(_ source: Vertex, destination: Vertex) -> Path! {


    let frontier: PathHeap = PathHeap()
    let finalPaths: PathHeap = PathHeap()


    //use source edges to create the frontier
    for e in source.neighbors {

        let newPath: Path = Path()


        newPath.destination = e.neighbor
        newPath.previous = nil
        newPath.total = e.weight


        //add the new path to the frontier
```

```
            frontier.enQueue(newPath)

        }


        //construct the best path
        var bestPath: Path = Path()


        while frontier.count != 0 {

            //use the greedy approach to obtain the best path
            bestPath = Path()
            bestPath = frontier.peek()


            //enumerate the bestPath edges
            for e in bestPath.destination.neighbors {

                let newPath: Path = Path()

                newPath.destination = e.neighbor
                newPath.previous = bestPath
                newPath.total = bestPath.total + e.weight


                //add the new path to the frontier
                frontier.enQueue(newPath)

            }


            //preserve the bestPaths that match destination
            if (bestPath.destination.key == destination.key) {
                finalPaths.enQueue(bestPath)
            }


            //remove the bestPath from the frontier
            frontier.deQueue()


        } //end while



        //obtain the shortest path from the heap
        var shortestPath: Path! = Path()
        shortestPath = finalPaths.peek()


        return shortestPath

    }


    //MARK: traversal algorithms


    //bfs traversal with inout closure function
    func traverse(_ startingv: Vertex, formula: (_ node: inout Vertex) -> ()) {
```

```
    //establish a new queue
    let graphQueue: Queue<Vertex> = Queue<Vertex>()


    //queue a starting vertex
    graphQueue.enQueue(startingv)


    while !graphQueue.isEmpty() {

        //traverse the next queued vertex
        var vitem: Vertex = graphQueue.deQueue() as Vertex!


        //add unvisited vertices to the queue
        for e in vitem.neighbors {
            if e.neighbor.visited == false {
                print("adding vertex: \(e.neighbor.key!) to queue..")
                graphQueue.enQueue(e.neighbor)
            }
        }


        /*
        notes: this demonstrates how to invoke a closure with an inout parameter.
        By passing by reference no return value is required.
        */

        //invoke formula
        formula(&vitem)


    } //end while


    print("graph traversal complete..")


}




//breadth first search
func traverse(_ startingv: Vertex) {


    //establish a new queue
    let graphQueue: Queue<Vertex> = Queue<Vertex>()


    //queue a starting vertex
    graphQueue.enQueue(startingv)


    while !graphQueue.isEmpty() {

        //traverse the next queued vertex
        let vitem = graphQueue.deQueue() as Vertex!
```

```
        guard vitem != nil else {
            return
        }

        //add unvisited vertices to the queue
        for e in vitem!.neighbors {
            if e.neighbor.visited == false {
                print("adding vertex: \(e.neighbor.key!) to queue..")
                graphQueue.enQueue(e.neighbor)
            }
        }


        vitem!.visited = true
        print("traversed vertex: \(vitem!.key!)..")


    } //end while


    print("graph traversal complete..")


} //end function



//use bfs with trailing closure to update all values
func update(startingv: Vertex, formula:((Vertex) -> Bool)) {


    //establish a new queue
    let graphQueue: Queue<Vertex> = Queue<Vertex>()


    //queue a starting vertex
    graphQueue.enQueue(startingv)


    while !graphQueue.isEmpty() {

        //traverse the next queued vertex
        let vitem = graphQueue.deQueue() as Vertex!

        guard vitem != nil else {
            return
        }

        //add unvisited vertices to the queue
        for e in vitem!.neighbors {
            if e.neighbor.visited == false {
                print("adding vertex: \(e.neighbor.key!) to queue..")
                graphQueue.enQueue(e.neighbor)
            }
        }


        //apply formula..
        if formula(vitem!) == false {
            print("formula unable to update: \(vitem!.key)")
```

```
            }
            else {
                print("traversed vertex: \(vitem!.key!)..")
            }

            vitem!.visited = true


        } //end while


        print("graph traversal complete..")


    }





}
```

trie - ◦

```
//
//  GraphFactory.swift
//  SwiftStructures
//
//  Created by Wayne Bishop on 6/7/14.
//  Copyright (c) 2014 Arbutus Software Inc. All rights reserved.
//
import Foundation


public class SwiftGraph {


    //declare a default directed graph canvas
    private var canvas: Array<Vertex>
    public var isDirected: Bool


    init() {
        canvas = Array<Vertex>()
        isDirected = true
    }


    //create a new vertex
    func addVertex(key: String) -> Vertex {


        //set the key
        let childVertex: Vertex = Vertex()
        childVertex.key = key


        //add the vertex to the graph canvas
        canvas.append(childVertex)
```

```
        return childVertex
    }



    //add edge to source vertex
    func addEdge(source: Vertex, neighbor: Vertex, weight: Int) {


        //create a new edge
        let newEdge = Edge()


        //establish the default properties
        newEdge.neighbor = neighbor
        newEdge.weight = weight
        source.neighbors.append(newEdge)


        print("The neighbor of vertex: \(source.key as String!) is \(neighbor.key as
String!)..")


        //check condition for an undirected graph
        if isDirected == false {


           //create a new reversed edge
           let reverseEdge = Edge()


           //establish the reversed properties
           reverseEdge.neighbor = source
           reverseEdge.weight = weight
           neighbor.neighbors.append(reverseEdge)

           print("The neighbor of vertex: \(neighbor.key as String!) is \(source.key as
String!)..")

        }


    }





    /* reverse the sequence of paths given the shortest path.
       process analagous to reversing a linked list. */

    func reversePath(_ head: Path!, source: Vertex) -> Path! {


        guard head != nil else {
            return head
        }

        //mutated copy
        var output = head
```

```
    var current: Path! = output
    var prev: Path!
    var next: Path!


    while(current != nil) {
        next = current.previous
        current.previous = prev
        prev = current
        current = next
    }


    //append the source path to the sequence
    let sourcePath: Path = Path()

    sourcePath.destination = source
    sourcePath.previous = prev
    sourcePath.total = nil

    output = sourcePath


    return output

}




//process Dijkstra's shortest path algorthim
func processDijkstra(_ source: Vertex, destination: Vertex) -> Path? {


    var frontier: Array<Path> = Array<Path>()
    var finalPaths: Array<Path> = Array<Path>()


    //use source edges to create the frontier
    for e in source.neighbors {

        let newPath: Path = Path()


        newPath.destination = e.neighbor
        newPath.previous = nil
        newPath.total = e.weight


        //add the new path to the frontier
        frontier.append(newPath)

    }


    //construct the best path
    var bestPath: Path = Path()
```

```
        while frontier.count != 0 {

            //support path changes using the greedy approach
            bestPath = Path()
            var pathIndex: Int = 0


            for x in 0..<frontier.count {

                let itemPath: Path = frontier[x]

                if  (bestPath.total == nil) || (itemPath.total < bestPath.total) {
                    bestPath = itemPath
                    pathIndex = x
                }

            }



            //enumerate the bestPath edges
            for e in bestPath.destination.neighbors {

                let newPath: Path = Path()

                newPath.destination = e.neighbor
                newPath.previous = bestPath
                newPath.total = bestPath.total + e.weight


                //add the new path to the frontier
                frontier.append(newPath)

            }


            //preserve the bestPath
            finalPaths.append(bestPath)


            //remove the bestPath from the frontier
            //frontier.removeAtIndex(pathIndex) - Swift2
            frontier.remove(at: pathIndex)



        } //end while



        //establish the shortest path as an optional
        var shortestPath: Path! = Path()


        for itemPath in finalPaths {

            if (itemPath.destination.key == destination.key) {

                if  (shortestPath.total == nil) || (itemPath.total < shortestPath.total) {
                    shortestPath = itemPath
                }
```

```
        }

    }


    return shortestPath

}



///an optimized version of Dijkstra's shortest path algorthim
func processDijkstraWithHeap(_ source: Vertex, destination: Vertex) -> Path! {


    let frontier: PathHeap = PathHeap()
    let finalPaths: PathHeap = PathHeap()


    //use source edges to create the frontier
    for e in source.neighbors {

        let newPath: Path = Path()


        newPath.destination = e.neighbor
        newPath.previous = nil
        newPath.total = e.weight


        //add the new path to the frontier
        frontier.enQueue(newPath)

    }


    //construct the best path
    var bestPath: Path = Path()


    while frontier.count != 0 {

        //use the greedy approach to obtain the best path
        bestPath = Path()
        bestPath = frontier.peek()


        //enumerate the bestPath edges
        for e in bestPath.destination.neighbors {

            let newPath: Path = Path()

            newPath.destination = e.neighbor
            newPath.previous = bestPath
            newPath.total = bestPath.total + e.weight


            //add the new path to the frontier
            frontier.enQueue(newPath)
```

```
        }


        //preserve the bestPaths that match destination
        if (bestPath.destination.key == destination.key) {
            finalPaths.enQueue(bestPath)
        }


        //remove the bestPath from the frontier
        frontier.deQueue()


    } //end while



    //obtain the shortest path from the heap
    var shortestPath: Path! = Path()
    shortestPath = finalPaths.peek()


    return shortestPath

}


//MARK: traversal algorithms


//bfs traversal with inout closure function
func traverse(_ startingv: Vertex, formula: (_ node: inout Vertex) -> ()) {


    //establish a new queue
    let graphQueue: Queue<Vertex> = Queue<Vertex>()


    //queue a starting vertex
    graphQueue.enQueue(startingv)


    while !graphQueue.isEmpty() {

        //traverse the next queued vertex
        var vitem: Vertex = graphQueue.deQueue() as Vertex!


        //add unvisited vertices to the queue
        for e in vitem.neighbors {
            if e.neighbor.visited == false {
                print("adding vertex: \(e.neighbor.key!) to queue..")
                graphQueue.enQueue(e.neighbor)
            }
        }


        /*
        notes: this demonstrates how to invoke a closure with an inout parameter.
        By passing by reference no return value is required.
        */
```

```
        //invoke formula
        formula(&vitem)


    } //end while


    print("graph traversal complete..")


}




//breadth first search
func traverse(_ startingv: Vertex) {


    //establish a new queue
    let graphQueue: Queue<Vertex> = Queue<Vertex>()


    //queue a starting vertex
    graphQueue.enQueue(startingv)


    while !graphQueue.isEmpty() {

        //traverse the next queued vertex
        let vitem = graphQueue.deQueue() as Vertex!

        guard vitem != nil else {
            return
        }

        //add unvisited vertices to the queue
        for e in vitem!.neighbors {
            if e.neighbor.visited == false {
                print("adding vertex: \(e.neighbor.key!) to queue..")
                graphQueue.enQueue(e.neighbor)
            }
        }


        vitem!.visited = true
        print("traversed vertex: \(vitem!.key!)..")


    } //end while


    print("graph traversal complete..")


} //end function



//use bfs with trailing closure to update all values
```

```
    func update(startingv: Vertex, formula:((Vertex) -> Bool)) {


        //establish a new queue
        let graphQueue: Queue<Vertex> = Queue<Vertex>()


        //queue a starting vertex
        graphQueue.enQueue(startingv)


        while !graphQueue.isEmpty() {

            //traverse the next queued vertex
            let vitem = graphQueue.deQueue() as Vertex!

            guard vitem != nil else {
                return
            }

            //add unvisited vertices to the queue
            for e in vitem!.neighbors {
                if e.neighbor.visited == false {
                    print("adding vertex: \(e.neighbor.key!) to queue..")
                    graphQueue.enQueue(e.neighbor)
                }
            }


            //apply formula..
            if formula(vitem!) == false {
                print("formula unable to update: \(vitem!.key)")
            }
            else {
                print("traversed vertex: \(vitem!.key!)..")
            }

            vitem!.visited = true


        } //end while


        print("graph traversal complete..")


    }




}
```

GitHub

pushpop。LIFO。。

github

```
//
//  GraphFactory.swift
//  SwiftStructures
//
//  Created by Wayne Bishop on 6/7/14.
//  Copyright (c) 2014 Arbutus Software Inc. All rights reserved.
//
import Foundation


public class SwiftGraph {


    //declare a default directed graph canvas
    private var canvas: Array<Vertex>
    public var isDirected: Bool


    init() {
        canvas = Array<Vertex>()
        isDirected = true
    }


    //create a new vertex
    func addVertex(key: String) -> Vertex {


        //set the key
        let childVertex: Vertex = Vertex()
        childVertex.key = key


        //add the vertex to the graph canvas
        canvas.append(childVertex)


        return childVertex
    }



    //add edge to source vertex
    func addEdge(source: Vertex, neighbor: Vertex, weight: Int) {


        //create a new edge
        let newEdge = Edge()


        //establish the default properties
        newEdge.neighbor = neighbor
        newEdge.weight = weight
        source.neighbors.append(newEdge)


        print("The neighbor of vertex: \(source.key as String!) is \(neighbor.key as
String!)..")


        //check condition for an undirected graph
```

```
        if isDirected == false {


           //create a new reversed edge
           let reverseEdge = Edge()


           //establish the reversed properties
           reverseEdge.neighbor = source
           reverseEdge.weight = weight
           neighbor.neighbors.append(reverseEdge)

           print("The neighbor of vertex: \(neighbor.key as String!) is \(source.key as
String!)..")

        }


    }





    /* reverse the sequence of paths given the shortest path.
       process analagous to reversing a linked list. */

    func reversePath(_ head: Path!, source: Vertex) -> Path! {


        guard head != nil else {
            return head
        }

        //mutated copy
        var output = head


        var current: Path! = output
        var prev: Path!
        var next: Path!


        while(current != nil) {
            next = current.previous
            current.previous = prev
            prev = current
            current = next
        }


        //append the source path to the sequence
        let sourcePath: Path = Path()

        sourcePath.destination = source
        sourcePath.previous = prev
        sourcePath.total = nil

        output = sourcePath
```

```
        return output

    }




    //process Dijkstra's shortest path algorthim
    func processDijkstra(_ source: Vertex, destination: Vertex) -> Path? {


        var frontier: Array<Path> = Array<Path>()
        var finalPaths: Array<Path> = Array<Path>()


        //use source edges to create the frontier
        for e in source.neighbors {

            let newPath: Path = Path()


            newPath.destination = e.neighbor
            newPath.previous = nil
            newPath.total = e.weight


            //add the new path to the frontier
            frontier.append(newPath)

        }


        //construct the best path
        var bestPath: Path = Path()


        while frontier.count != 0 {

            //support path changes using the greedy approach
            bestPath = Path()
            var pathIndex: Int = 0


            for x in 0..<frontier.count {

                let itemPath: Path = frontier[x]

                if  (bestPath.total == nil) || (itemPath.total < bestPath.total) {
                    bestPath = itemPath
                    pathIndex = x
                }

            }



            //enumerate the bestPath edges
            for e in bestPath.destination.neighbors {

                let newPath: Path = Path()
```

```
            newPath.destination = e.neighbor
            newPath.previous = bestPath
            newPath.total = bestPath.total + e.weight


            //add the new path to the frontier
            frontier.append(newPath)

        }


        //preserve the bestPath
        finalPaths.append(bestPath)


        //remove the bestPath from the frontier
        //frontier.removeAtIndex(pathIndex) - Swift2
        frontier.remove(at: pathIndex)



    } //end while



    //establish the shortest path as an optional
    var shortestPath: Path! = Path()


    for itemPath in finalPaths {

        if (itemPath.destination.key == destination.key) {

            if  (shortestPath.total == nil) || (itemPath.total < shortestPath.total) {
                shortestPath = itemPath
            }

        }

    }


    return shortestPath

}



///an optimized version of Dijkstra's shortest path algorthim
func processDijkstraWithHeap(_ source: Vertex, destination: Vertex) -> Path! {


    let frontier: PathHeap = PathHeap()
    let finalPaths: PathHeap = PathHeap()


    //use source edges to create the frontier
    for e in source.neighbors {

        let newPath: Path = Path()
```

```
        newPath.destination = e.neighbor
        newPath.previous = nil
        newPath.total = e.weight


        //add the new path to the frontier
        frontier.enQueue(newPath)

    }


    //construct the best path
    var bestPath: Path = Path()


    while frontier.count != 0 {

        //use the greedy approach to obtain the best path
        bestPath = Path()
        bestPath = frontier.peek()


        //enumerate the bestPath edges
        for e in bestPath.destination.neighbors {

            let newPath: Path = Path()

            newPath.destination = e.neighbor
            newPath.previous = bestPath
            newPath.total = bestPath.total + e.weight


            //add the new path to the frontier
            frontier.enQueue(newPath)

        }


        //preserve the bestPaths that match destination
        if (bestPath.destination.key == destination.key) {
            finalPaths.enQueue(bestPath)
        }


        //remove the bestPath from the frontier
        frontier.deQueue()


    } //end while



    //obtain the shortest path from the heap
    var shortestPath: Path! = Path()
    shortestPath = finalPaths.peek()


    return shortestPath

}
```

```
//MARK: traversal algorithms


//bfs traversal with inout closure function
func traverse(_ startingv: Vertex, formula: (_ node: inout Vertex) -> ()) {


    //establish a new queue
    let graphQueue: Queue<Vertex> = Queue<Vertex>()


    //queue a starting vertex
    graphQueue.enQueue(startingv)


    while !graphQueue.isEmpty() {

        //traverse the next queued vertex
        var vitem: Vertex = graphQueue.deQueue() as Vertex!


        //add unvisited vertices to the queue
        for e in vitem.neighbors {
            if e.neighbor.visited == false {
                print("adding vertex: \(e.neighbor.key!) to queue..")
                graphQueue.enQueue(e.neighbor)
            }
        }


        /*
        notes: this demonstrates how to invoke a closure with an inout parameter.
        By passing by reference no return value is required.
        */

        //invoke formula
        formula(&vitem)


    } //end while


    print("graph traversal complete..")


}




//breadth first search
func traverse(_ startingv: Vertex) {


    //establish a new queue
    let graphQueue: Queue<Vertex> = Queue<Vertex>()


    //queue a starting vertex
```

```
        graphQueue.enQueue(startingv)


        while !graphQueue.isEmpty() {

            //traverse the next queued vertex
            let vitem = graphQueue.deQueue() as Vertex!

            guard vitem != nil else {
                return
            }

            //add unvisited vertices to the queue
            for e in vitem!.neighbors {
                if e.neighbor.visited == false {
                    print("adding vertex: \(e.neighbor.key!) to queue..")
                    graphQueue.enQueue(e.neighbor)
                }
            }


            vitem!.visited = true
            print("traversed vertex: \(vitem!.key!)..")


        } //end while


        print("graph traversal complete..")


    } //end function



    //use bfs with trailing closure to update all values
    func update(startingv: Vertex, formula:((Vertex) -> Bool)) {


        //establish a new queue
        let graphQueue: Queue<Vertex> = Queue<Vertex>()


        //queue a starting vertex
        graphQueue.enQueue(startingv)


        while !graphQueue.isEmpty() {

            //traverse the next queued vertex
            let vitem = graphQueue.deQueue() as Vertex!

            guard vitem != nil else {
                return
            }

            //add unvisited vertices to the queue
            for e in vitem!.neighbors {
                if e.neighbor.visited == false {
                    print("adding vertex: \(e.neighbor.key!) to queue..")
                    graphQueue.enQueue(e.neighbor)
```

```
                }
            }


            //apply formula..
            if formula(vitem!) == false {
                print("formula unable to update: \(vitem!.key)")
            }
            else {
                print("traversed vertex: \(vitem!.key!)..")
            }

            vitem!.visited = true


        } //end while


        print("graph traversal complete..")


    }




}
```

MIT

c2015Wayne BishopArbutus Software Inc.

“”/

。

“”。 。 。

Swift https://riptutorial.com/zh-TW/swift/topic/9116/swift

13: Typealias

Examples

```
typealias SuccessHandler = (NSURLSessionDataTask, AnyObject?) -> Void
```

```
SuccessHandlervar string = ""string。

SuccessHandler
```

```
typealias SuccessHandler = (NSURLSessionDataTask, AnyObject?) -> Void
```

```
typealias SuccessHandler = (NSURLSessionDataTask, AnyObject?) -> Void
```

```
typealias Handler = () -> Void
typealias Handler = () -> ()
```

Void to Void。

```
typealias Handler = () -> Void
typealias Handler = () -> ()
```

```
typealias Number = NSNumber
```

```
。

typealias Number = NSNumber
```

```
typealias Number = NSNumber
```

Typealias https://riptutorial.com/zh-TW/swift/topic/7552/typealias

14:

- Swift 3.0
- DispatchQueue.main //
- DispatchQueue“my-serial-queue”[。 serial.qosBackground]//
- DispatchQueue.globalattributes[。 qosDefault]//
- DispatchQueue.main.async {...} //
- DispatchQueue.main.sync {...} //
- DispatchQueue.main.asyncAfter.now+ 3{...} //x
- <3.0
- dispatch_get_main_queue//
- dispatch_get_global_queuedispatch_queue_priority_t0//dispatch_queue_priority_t
- dispatch_asyncdispatch_queue_t{ - > Void in ...} //dispatch_queue_t
- dispatch_syncdispatch_queue_t{ - > Void in ...} //dispatch_queue_t
- dispatch_afterdispatch_timeDISPATCH_TIME_NOWInt64dispatch_queue_t{...}; // dispatch_queue_t

Examples

Grand Central DispatchGCD

Grand Central Dispatch“”。 。

- **Serial Dispatch Queues** 。 。
- 。
- **Main Dispatch Queue**。

3.0

```
let mainQueue = DispatchQueue.main
```

3.0

```
let mainQueue = DispatchQueue.main
```

。 Swift 3`DispatchQueue`

3.0

```
let mainQueue = DispatchQueue.main
```

```
let mainQueue = DispatchQueue.main
```

3.0

```
let mainQueue = DispatchQueue.main
```

iOS 8.`userInteractive` `.userInitiated` `.default` `.utility`.`background`。 `DISPATCH_QUEUE_PRIORITY_`。

3.0

```
let mainQueue = DispatchQueue.main
```

3.0

```
let mainQueue = DispatchQueue.main
```

Swift 3.`workItem`。 `.never` `.inherit`。 `.never`。

Grand Central DispatchGCD

3.0

`sync` `asyncafter`。

```
let queue = DispatchQueue(label: "myQueueName")

queue.async {
    //do something

    DispatchQueue.main.async {
        //this will be called in main thread
        //any UI updates should be placed here
    }
}
// ... code here will execute immediately, before the task finished
```

```
let queue = DispatchQueue(label: "myQueueName")

queue.async {
    //do something

    DispatchQueue.main.async {
        //this will be called in main thread
        //any UI updates should be placed here
    }
}
// ... code here will execute immediately, before the task finished
```

```
let queue = DispatchQueue(label: "myQueueName")
```

```
queue.async {
    //do something

    DispatchQueue.main.async {
        //this will be called in main thread
        //any UI updates should be placed here
    }
}
// ... code here will execute immediately, before the task finished
```

UI`DispatchQueue.main.async { ... }`

2.0

```
let queue = DispatchQueue(label: "myQueueName")

queue.async {
    //do something

    DispatchQueue.main.async {
        //this will be called in main thread
        //any UI updates should be placed here
    }
}
// ... code here will execute immediately, before the task finished
```

```
let queue = DispatchQueue(label: "myQueueName")

queue.async {
    //do something

    DispatchQueue.main.async {
        //this will be called in main thread
        //any UI updates should be placed here
    }
}
// ... code here will execute immediately, before the task finished
```

```
let queue = DispatchQueue(label: "myQueueName")

queue.async {
    //do something

    DispatchQueue.main.async {
        //this will be called in main thread
        //any UI updates should be placed here
    }
}
// ... code here will execute immediately, before the task finished
```

`NSEC_PER_SEC`

```
let queue = DispatchQueue(label: "myQueueName")

queue.async {
```

```
    //do something

    DispatchQueue.main.async {
        //this will be called in main thread
        //any UI updates should be placed here
    }
}
// ... code here will execute immediately, before the task finished
```

UI

```
let queue = DispatchQueue(label: "myQueueName")

queue.async {
    //do something

    DispatchQueue.main.async {
        //this will be called in main thread
        //any UI updates should be placed here
    }
}
// ... code here will execute immediately, before the task finished
```

UI`dispatch_async(dispatch_get_main_queue()) { ... }`

GCD。 。

```
for index in 0 ..< iterations {
    // Do something computationally expensive here
}
```

`concurrentPerform` Swift 3`dispatch_apply` Swift 2 `concurrentPerform`

3.0

```
for index in 0 ..< iterations {
    // Do something computationally expensive here
}
```

3.0

```
for index in 0 ..< iterations {
    // Do something computationally expensive here
}
```

`0iterationsindex。 。 。`

- `concurrentPerform` / `dispatch_apply`。 。
- 。 。
- 。 。

。“”。10010010,000。。

OperationQueue

OperationQueue。GCDFIFO。。

OperationQueue

3.0

```
let mainQueue = OperationQueue.main
```

OperationQueue

3.0

```
let mainQueue = OperationQueue.main
```

。

OperationOperationQueue

3.0

```
let mainQueue = OperationQueue.main
```

OperationQueue

3.0

```
let mainQueue = OperationQueue.main
```

OperationOperationQueue

3.0

```
let mainQueue = OperationQueue.main
```

Operation

```
let mainQueue = OperationQueue.main
```

。 isSuspendedfalse

3.0

```
let mainQueue = OperationQueue.main
```

OperationQueue。。

Foundation`Operation`。 。

`Operation`。 `isReady``true`。

`Operation`

3.0

```
class MyOperation: Operation {

    init(<parameters>) {
        // Do any setup work here
    }

    override func main() {
        // Perform the task
    }

}
```

2.3

```
class MyOperation: Operation {

    init(<parameters>) {
        // Do any setup work here
    }

    override func main() {
        // Perform the task
    }

}
```

`OperationQueue``OperationQueue`

1.0

```
class MyOperation: Operation {

    init(<parameters>) {
        // Do any setup work here
    }

    override func main() {
        // Perform the task
    }

}
```

。

`Operation`。

`Operation``Operation`。

1.0

```
class MyOperation: Operation {

    init(<parameters>) {
        // Do any setup work here
    }

    override func main() {
        // Perform the task
    }

}
```

Operation

1.0

```
class MyOperation: Operation {

    init(<parameters>) {
        // Do any setup work here
    }

    override func main() {
        // Perform the task
    }

}
```

◦ start◦

◦

OperationURLSessionOperation◦ isAsynchronoustrue startmain◦

OperationisExecuting isFinishedKVO◦ isExecutingtrue ◦ Operation isExecutingfalse isFinishedtrue ◦ isCancelledisFinishedtrue ◦ ◦

Operation ◦

cancelisCancelledtrue ◦ OperationmainisCancelled◦

1.0

```
class MyOperation: Operation {

    init(<parameters>) {
        // Do any setup work here
    }

    override func main() {
        // Perform the task
    }

}
```

https://riptutorial.com/zh-TW/swift/topic/1649/

15: Swift

。 。

Examples

```
struct Mathematics
{
    internal func performOperation(inputArray: [Int], operation: (Int)-> Int)-> [Int]
    {
        var processedArray = [Int]()

        for item in inputArray
        {
            processedArray.append(operation(item))
        }

        return processedArray
    }


    internal func performComplexOperation(valueOne: Int)-> ((Int)-> Int)
    {
        return
            ({
                 return valueOne + $0
            })
    }

}


let arrayToBeProcessed = [1,3,5,7,9,11,8,6,4,2,100]

let math = Mathematics()

func add2(item: Int)-> Int
{
    return (item + 2)
}

// assigning the function to a variable and then passing it to a function as param
let add2ToMe = add2
print(math.performOperation(inputArray: arrayToBeProcessed, operation: add2ToMe))
```

```
struct Mathematics
{
    internal func performOperation(inputArray: [Int], operation: (Int)-> Int)-> [Int]
    {
        var processedArray = [Int]()

        for item in inputArray
        {
            processedArray.append(operation(item))
        }
```

```
        return processedArray
    }


    internal func performComplexOperation(valueOne: Int)-> ((Int)-> Int)
    {
        return
            ({
                 return valueOne + $0
            })
    }

}


let arrayToBeProcessed = [1,3,5,7,9,11,8,6,4,2,100]

let math = Mathematics()

func add2(item: Int)-> Int
{
    return (item + 2)
}

// assigning the function to a variable and then passing it to a function as param
let add2ToMe = add2
print(math.performOperation(inputArray: arrayToBeProcessed, operation: add2ToMe))
```

closure

```
struct Mathematics
{
    internal func performOperation(inputArray: [Int], operation: (Int)-> Int)-> [Int]
    {
        var processedArray = [Int]()

        for item in inputArray
        {
            processedArray.append(operation(item))
        }

        return processedArray
    }


    internal func performComplexOperation(valueOne: Int)-> ((Int)-> Int)
    {
        return
            ({
                 return valueOne + $0
            })
    }

}


let arrayToBeProcessed = [1,3,5,7,9,11,8,6,4,2,100]

let math = Mathematics()
```

```
func add2(item: Int)-> Int
{
    return (item + 2)
}

// assigning the function to a variable and then passing it to a function as param
let add2ToMe = add2
print(math.performOperation(inputArray: arrayToBeProcessed, operation: add2ToMe))
```

```
func multiply2(item: Int)-> Int
{
    return (item + 2)
}


let multiply2ToMe = multiply2

// passing the function directly to the function as param
print(math.performOperation(inputArray: arrayToBeProcessed, operation: multiply2ToMe))
```

```
func multiply2(item: Int)-> Int
{
    return (item + 2)
}


let multiply2ToMe = multiply2

// passing the function directly to the function as param
print(math.performOperation(inputArray: arrayToBeProcessed, operation: multiply2ToMe))
```

closure

```
func multiply2(item: Int)-> Int
{
    return (item + 2)
}


let multiply2ToMe = multiply2

// passing the function directly to the function as param
print(math.performOperation(inputArray: arrayToBeProcessed, operation: multiply2ToMe))
```

```
// function as return type
print(math.performComplexOperation(valueOne: 4)(5))
```

9

Swift https://riptutorial.com/zh-TW/swift/topic/8618/swift

16: CObjective-C

AppleSwiftCocoaObjective-C。

Examples

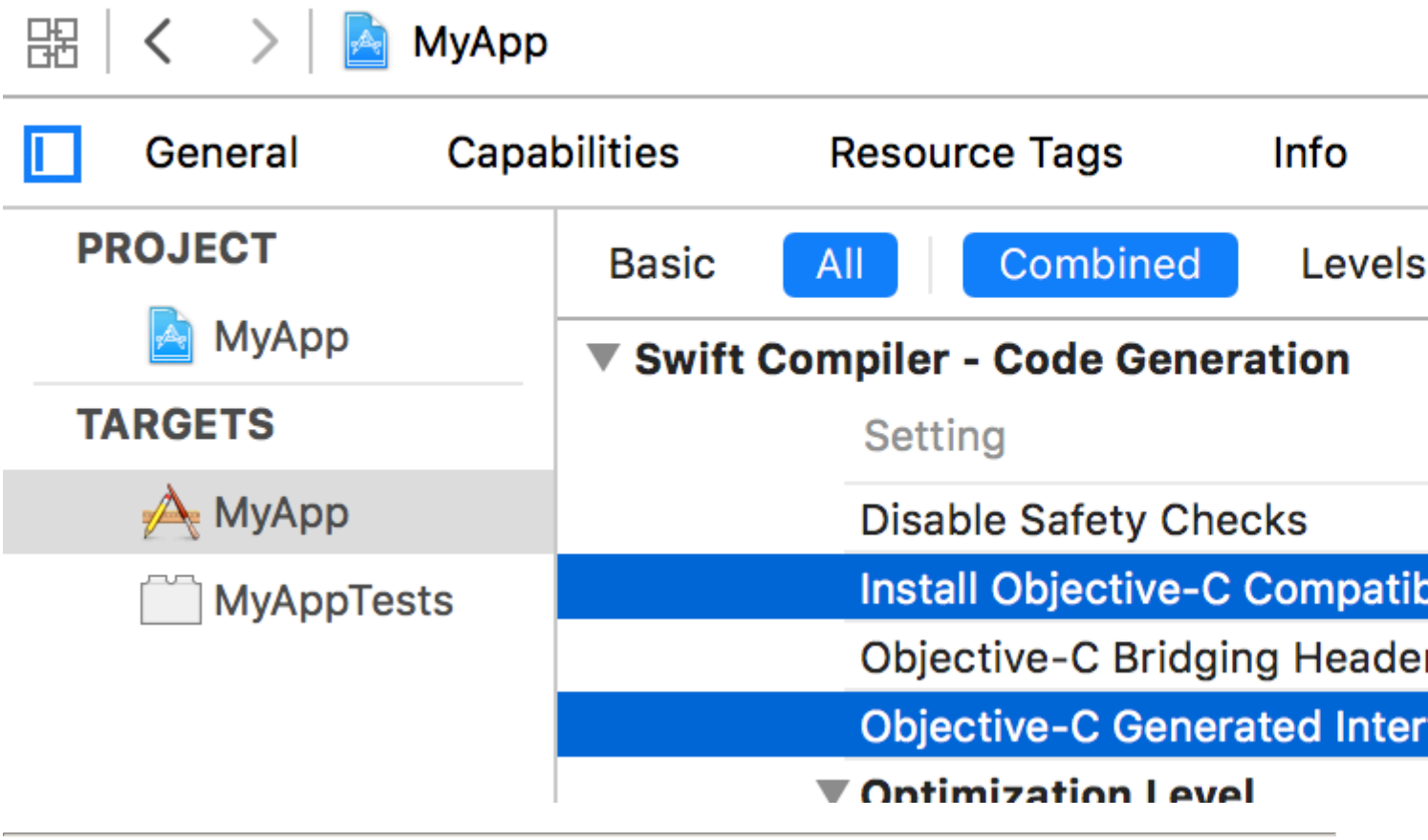
Objective-CSwift

“ **MyModule** ”Xcode`MyModule-Swift.h`SwiftObjective-C。 Swift

```
// MySwiftClass.swift in MyApp
import Foundation

// The class must be `public` to be visible, unless this target also has a bridging header
public class MySwiftClass: NSObject {
    // ...
}
```


```
// MySwiftClass.swift in MyApp
import Foundation

// The class must be `public` to be visible, unless this target also has a bridging header
public class MySwiftClass: NSObject {
    // ...
}
```


- **Objective-C** Obj-C。
- **Objective-C** -Swift.h。

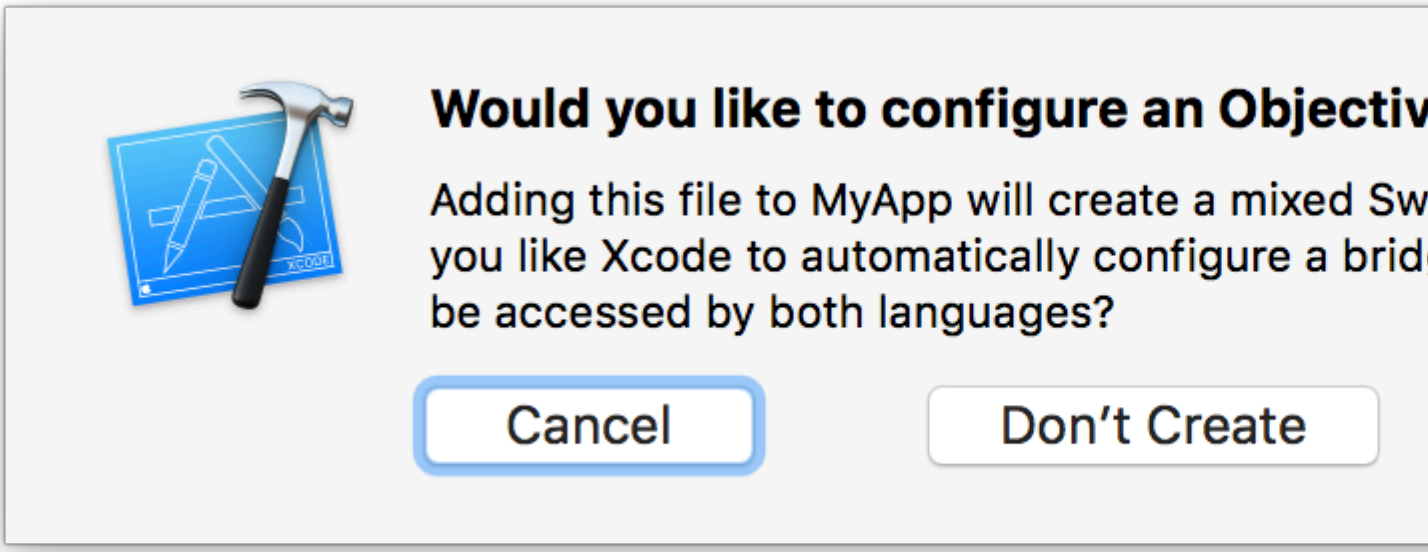


`@import MyFramework;`SwiftObj-C。

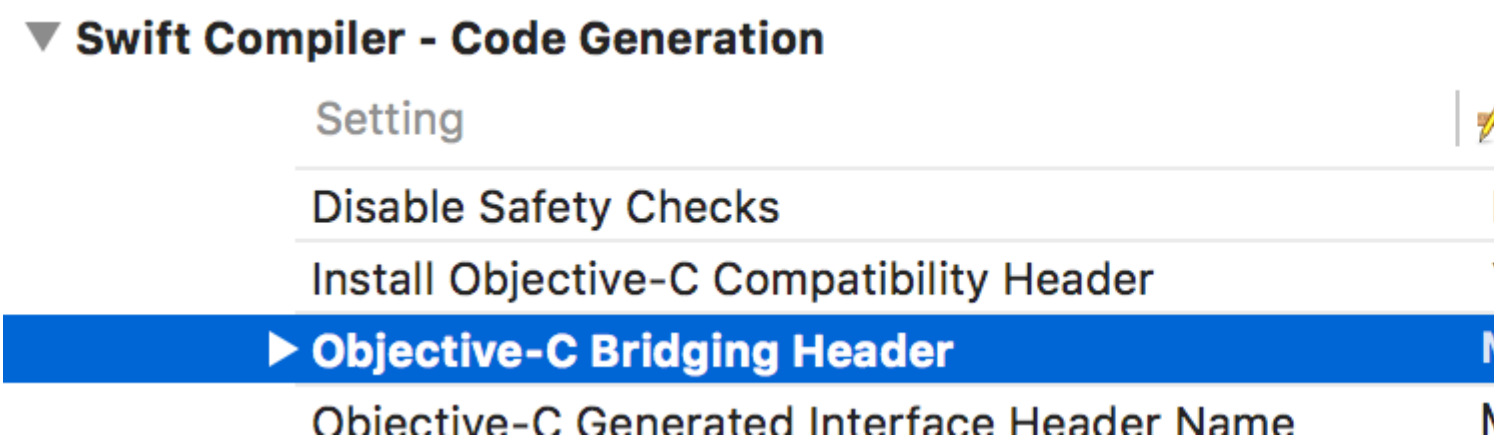
SwiftObjective-C

MyFrameworkObjective-C`import MyFramework`Swift。

SwiftObjective-CC。 Xcode

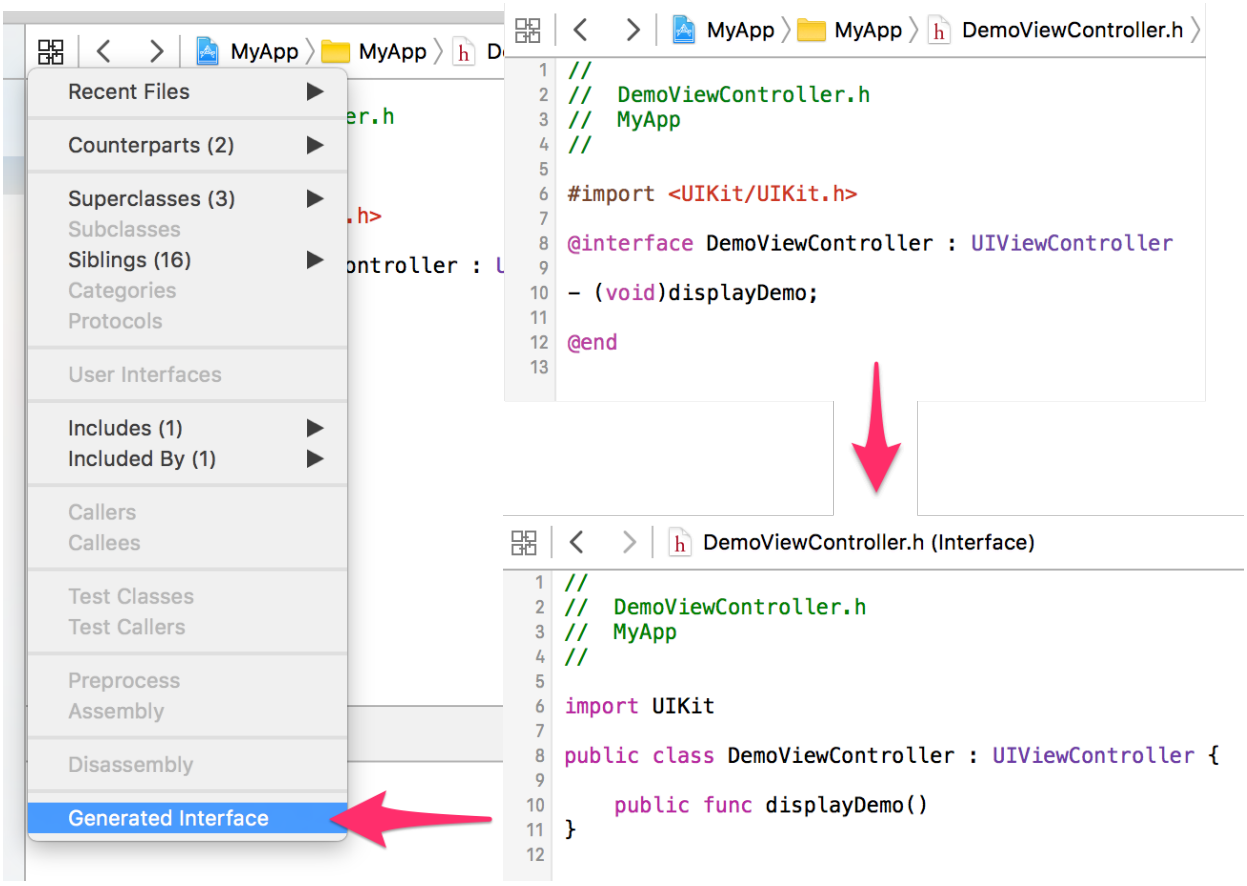


Objective-C Bridging Header




```
// MyApp-Bridging-Header.h
#import "MyClass.h"  // allows code in this module to use MyClass
```

Related Items^ 1**Generated Interface**Objective-CSwift。



swiftc

```
-import-objc-headerswiftc
```

```
// defs.h
struct Color {
    int red, green, blue;
};
```

```
#define MAX_VALUE 255
```

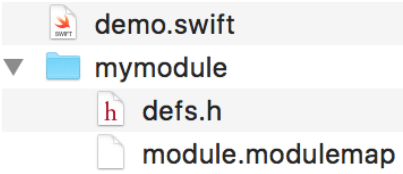
```
// defs.h
struct Color {
    int red, green, blue;
};

#define MAX_VALUE 255
```

```
// defs.h
struct Color {
    int red, green, blue;
};

#define MAX_VALUE 255
```

C

`import mymodule`CSwift。

`module.modulemapmymodule`




```
// mymodule/module.modulemap
module mymodule {
    header "defs.h"
}
```

`import`

```
// mymodule/module.modulemap
module mymodule {
    header "defs.h"
}
```

`-I directoryswiftc`

```
// mymodule/module.modulemap
module mymodule {
    header "defs.h"
}
```

Clang 。

Objective-CSwift

`NS_REFINED_FOR_SWIFT`APISwift __

```
@interface MyClass : NSObject
- (NSInteger)indexOfObject:(id)obj NS_REFINED_FOR_SWIFT;
@end
```

```
@interface MyClass : NSObject
- (NSInteger)indexOfObject:(id)obj NS_REFINED_FOR_SWIFT;
@end
```

“Swifty”**API** 。 NSNotFound

```
@interface MyClass : NSObject
- (NSInteger)indexOfObject:(id)obj NS_REFINED_FOR_SWIFT;
@end
```

Objective-C`nil` 。 `_Nonnull`

```
@interface MyClass : NSObject
- (NSInteger)indexOfObject:(id)obj NS_REFINED_FOR_SWIFT;
@end
```

Swift`nil`

```
@interface MyClass : NSObject
- (NSInteger)indexOfObject:(id)obj NS_REFINED_FOR_SWIFT;
@end
```

`_Nonnull` `_Nullable` `nil`。 `_Nullable`;。

```
@interface MyClass : NSObject
- (NSInteger)indexOfObject:(id)obj NS_REFINED_FOR_SWIFT;
@end
```

。

C

SwiftCC。

LinuxC`Glibc`;Apple`Darwin` 。

```
#if os(macOS) || os(iOS) || os(tvOS) || os(watchOS)
import Darwin
#elseif os(Linux)
import Glibc
#endif

// use open(), read(), and other libc features
```

CObjective-C https://riptutorial.com/zh-TW/swift/topic/421/cobjective-c

17:

Examples

Dependenct

。 。 。

。 。 。

- 。

- UI
- -
-

DI

1.
2.
3. DISwinjectCleanseDipTyphoon

DIInversion of Control。

DIView Controllers - iOS。

DI

LoginViewControllerTimelineViewController 。 LoginViewControllerTimelineViewController。 **FirebaseNetworkService** 。

LoginViewController

```
class LoginViewController: UIViewController {

    var networkService = FirebaseNetworkService()

    override func viewDidLoad() {
        super.viewDidLoad()
    }
}
```

TimelineViewController

```
class LoginViewController: UIViewController {
```

```
    var networkService = FirebaseNetworkService()

    override func viewDidLoad() {
        super.viewDidLoad()
    }
}
```

FirebaseNetworkService

```
class LoginViewController: UIViewController {

    var networkService = FirebaseNetworkService()

    override func viewDidLoad() {
        super.viewDidLoad()
    }
}
```

1015FirebaseNetworkService。 Firebase。 CompanyNetworkServiceFirebaseNetworkService。 CompanyNetworkService。

UI。

。

。 。

```
class LoginViewController: UIViewController {

    var networkService = FirebaseNetworkService()

    override func viewDidLoad() {
        super.viewDidLoad()
    }
}
```

NetworkService

```
class LoginViewController: UIViewController {

    var networkService = FirebaseNetworkService()

    override func viewDidLoad() {
        super.viewDidLoad()
    }
}
```

LoginViewControllerTimelineViewControllerNetworkServiceFirebaseNetworkService。

LoginViewController

```
class LoginViewController: UIViewController {
```

```
    var networkService = FirebaseNetworkService()

    override func viewDidLoad() {
        super.viewDidLoad()
    }
}
```

TimelineViewController

```
class LoginViewController: UIViewController {

    var networkService = FirebaseNetworkService()

    override func viewDidLoad() {
        super.viewDidLoad()
    }
}
```

LoginViewControllerTimelineViewControllerNetworkService

LoginViewController**AppDelegate**。

```
class LoginViewController: UIViewController {

    var networkService = FirebaseNetworkService()

    override func viewDidLoad() {
        super.viewDidLoad()
    }
}
```

AppDelegateLoginViewControllerNetworkService。

TimelineViewControllerNetworkService。 LoginViewControllerTimlineViewController。

LoginViewControllerprepareForSegue。

LoginViewController

```
class LoginViewController: UIViewController {

    var networkService = FirebaseNetworkService()

    override func viewDidLoad() {
        super.viewDidLoad()
    }
}
```

。

NetworkServiceFirebase。

NetworkService

```
class LoginViewController: UIViewController {

    var networkService = FirebaseNetworkService()

    override func viewDidLoad() {
        super.viewDidLoad()
    }
}
```

AppDelegateFirebaseNetworkServiceImpl

```
class LoginViewController: UIViewController {

    var networkService = FirebaseNetworkService()

    override func viewDidLoad() {
        super.viewDidLoad()
    }
}
```

LoginViewControllerTimelineViewController。

DI。

Swift **DI**

1. Swift
2.
3.

DI

```
    protocol Engine {
        func startEngine()
        func stopEngine()
    }

    class TrainEngine: Engine {
        func startEngine() {
            print("Engine started")
        }

        func stopEngine() {
            print("Engine stopped")
        }
    }

    protocol TrainCar {
    var numberOfSeats: Int { get }
    func attachCar(attach: Bool)
}

class RestaurantCar: TrainCar {
    var numberOfSeats: Int {
        get {
            return 30
```

```
        }
    }
    func attachCar(attach: Bool) {
        print("Attach car")
    }
}

class PassengerCar: TrainCar {
    var numberOfSeats: Int {
        get {
            return 50
        }
    }
    func attachCar(attach: Bool) {
        print("Attach car")
    }
}

class Train {
    let engine: Engine?
    var mainCar: TrainCar?
}
```

- Train

```
    protocol Engine {
        func startEngine()
        func stopEngine()
    }

    class TrainEngine: Engine {
        func startEngine() {
            print("Engine started")
        }

        func stopEngine() {
            print("Engine stopped")
        }
    }

    protocol TrainCar {
    var numberOfSeats: Int { get }
    func attachCar(attach: Bool)
}

class RestaurantCar: TrainCar {
    var numberOfSeats: Int {
        get {
            return 30
        }
    }
    func attachCar(attach: Bool) {
        print("Attach car")
    }
}

class PassengerCar: TrainCar {
    var numberOfSeats: Int {
```

```
        get {
            return 50
        }
    }
    func attachCar(attach: Bool) {
        print("Attach car")
    }
}

class Train {
    let engine: Engine?
    var mainCar: TrainCar?
}
```

TrainEngine

```
    protocol Engine {
        func startEngine()
        func stopEngine()
    }

    class TrainEngine: Engine {
        func startEngine() {
            print("Engine started")
        }

        func stopEngine() {
            print("Engine stopped")
        }
    }

    protocol TrainCar {
    var numberOfSeats: Int { get }
    func attachCar(attach: Bool)
}

class RestaurantCar: TrainCar {
    var numberOfSeats: Int {
        get {
            return 30
        }
    }
    func attachCar(attach: Bool) {
        print("Attach car")
    }
}

class PassengerCar: TrainCar {
    var numberOfSeats: Int {
        get {
            return 50
        }
    }
    func attachCar(attach: Bool) {
        print("Attach car")
    }
}

class Train {
    let engine: Engine?
```

```
    var mainCar: TrainCar?
}
```

let。 。

DI。 DIPassengerCar

```
    protocol Engine {
        func startEngine()
        func stopEngine()
    }

    class TrainEngine: Engine {
        func startEngine() {
            print("Engine started")
        }

        func stopEngine() {
            print("Engine stopped")
        }
    }

    protocol TrainCar {
    var numberOfSeats: Int { get }
    func attachCar(attach: Bool)
}

class RestaurantCar: TrainCar {
    var numberOfSeats: Int {
        get {
            return 30
        }
    }
    func attachCar(attach: Bool) {
        print("Attach car")
    }
}

class PassengerCar: TrainCar {
    var numberOfSeats: Int {
        get {
            return 50
        }
    }
    func attachCar(attach: Bool) {
        print("Attach car")
    }
}

class Train {
    let engine: Engine?
    var mainCar: TrainCar?
}
```

。 mainCarPassengerCar。

◦ ◦ Train

```
    protocol Engine {
        func startEngine()
        func stopEngine()
    }

    class TrainEngine: Engine {
        func startEngine() {
            print("Engine started")
        }

        func stopEngine() {
            print("Engine stopped")
        }
    }

    protocol TrainCar {
    var numberOfSeats: Int { get }
    func attachCar(attach: Bool)
}

class RestaurantCar: TrainCar {
    var numberOfSeats: Int {
        get {
            return 30
        }
    }
    func attachCar(attach: Bool) {
        print("Attach car")
    }
}

class PassengerCar: TrainCar {
    var numberOfSeats: Int {
        get {
            return 50
        }
    }
    func attachCar(attach: Bool) {
        print("Attach car")
    }
}

class Train {
    let engine: Engine?
    var mainCar: TrainCar?
}
```

Train`TrainCar` ◦

```
    protocol Engine {
        func startEngine()
        func stopEngine()
    }

    class TrainEngine: Engine {
        func startEngine() {
            print("Engine started")
```

```
        }

        func stopEngine() {
            print("Engine stopped")
        }
    }

    protocol TrainCar {
    var numberOfSeats: Int { get }
    func attachCar(attach: Bool)
}

class RestaurantCar: TrainCar {
    var numberOfSeats: Int {
        get {
            return 30
        }
    }
    func attachCar(attach: Bool) {
        print("Attach car")
    }
}

class PassengerCar: TrainCar {
    var numberOfSeats: Int {
        get {
            return 50
        }
    }
    func attachCar(attach: Bool) {
        print("Attach car")
    }
}

class Train {
    let engine: Engine?
    var mainCar: TrainCar?
}
```

https://riptutorial.com/zh-TW/swift/topic/8198/

18:

。

。

:)。

-

。
developer.apple.com/library/content/documentation/Swift/Conceptual/Swift_Programming_Language/Thel

Examples

。 。

```
let tuple = ("one", 2, "three")

// Values are read using index numbers starting at zero
print(tuple.0) // one
print(tuple.1) // 2
print(tuple.2) // three
```


```
let tuple = ("one", 2, "three")

// Values are read using index numbers starting at zero
print(tuple.0) // one
print(tuple.1) // 2
print(tuple.2) // three
```

```
let tuple = ("one", 2, "three")

// Values are read using index numbers starting at zero
print(tuple.0) // one
print(tuple.1) // 2
print(tuple.2) // three
```


```
let myTuple = (name: "Some Name", age: 26)
let (first, second) = myTuple

print(first) // "Some Name"
print(second)  // 26
```


```
let myTuple = (name: "Some Name", age: 26)
let (first, second) = myTuple

print(first) // "Some Name"
print(second)  // 26
```

–

```
let myTuple = (name: "Some Name", age: 26)
let (first, second) = myTuple

print(first) // "Some Name"
print(second)  // 26
```

```
func tupleReturner() -> (Int, String) {
    return (3, "Hello")
}

let myTuple = tupleReturner()
print(myTuple.0) // 3
print(myTuple.1) // "Hello"
```

```
func tupleReturner() -> (Int, String) {
    return (3, "Hello")
}

let myTuple = tupleReturner()
print(myTuple.0) // 3
print(myTuple.1) // "Hello"
```

typealias

。

```
// Define a circle tuple by its center point and radius
let unitCircle: (center: (x: CGFloat, y: CGFloat), radius: CGFloat) = ((0.0, 0.0), 1.0)

func doubleRadius(ofCircle circle: (center: (x: CGFloat, y: CGFloat), radius: CGFloat)) ->
(center: (x: CGFloat, y: CGFloat), radius: CGFloat) {
    return (circle.center, circle.radius * 2.0)
}
```

`typealias`。


```
// Define a circle tuple by its center point and radius
let unitCircle: (center: (x: CGFloat, y: CGFloat), radius: CGFloat) = ((0.0, 0.0), 1.0)

func doubleRadius(ofCircle circle: (center: (x: CGFloat, y: CGFloat), radius: CGFloat)) ->
(center: (x: CGFloat, y: CGFloat), radius: CGFloat) {
    return (circle.center, circle.radius * 2.0)
}
```


`struct` 。

2。

2

2

```
var a = "Marty McFly"
var b = "Emmett Brown"
```

```
var a = "Marty McFly"
var b = "Emmett Brown"
```

```
var a = "Marty McFly"
var b = "Emmett Brown"
```

4

```
var a = "Marty McFly"
var b = "Emmett Brown"
```

Switch

```
let switchTuple = (firstCase: true, secondCase: false)

switch switchTuple {
 case (true, false):
    // do something
 case (true, true):
    // do something
 case (false, true):
    // do something
 case (false, false):
    // do something
}
```

EnumSize Classes

```
let switchTuple = (firstCase: true, secondCase: false)

switch switchTuple {
 case (true, false):
    // do something
 case (true, true):
    // do something
 case (false, true):
    // do something
 case (false, false):
    // do something
}
```

https://riptutorial.com/zh-TW/swift/topic/574/

19:

Swift。 。 。 [`Unmanaged`] [1]。 [1]https//developer.apple.com/reference/swift/unmanaged

weak-keyword

`weak`。

unowned-keyword

`unowned` -keyword。

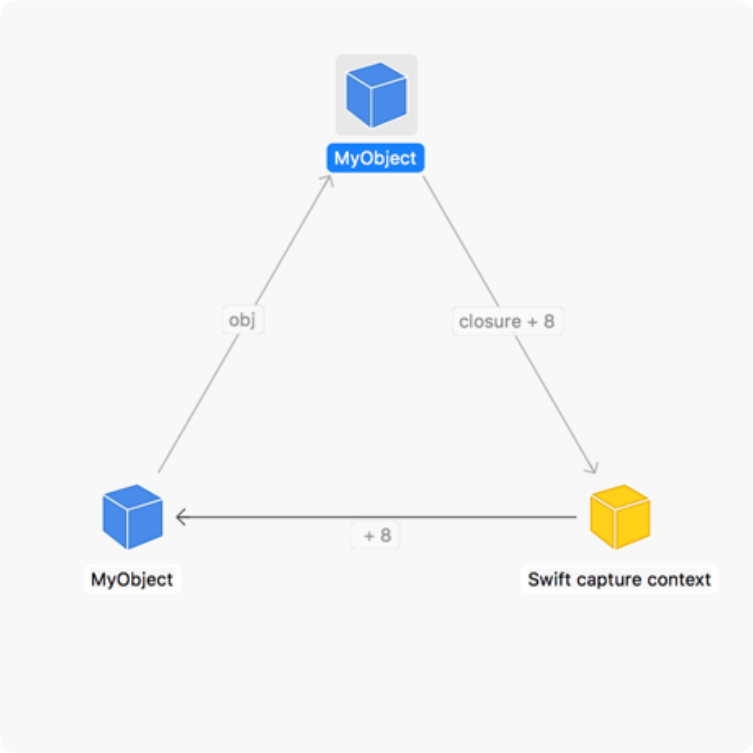
。

```
class A : CLLocationManagerDelegate
{
    init()
    {
        let locationManager = CLLocationManager()
        locationManager.delegate = self
        locationManager.startLocationUpdates()
    }
}
```

。

```
class A : CLLocationManagerDelegate
{
    init()
    {
        let locationManager = CLLocationManager()
        locationManager.delegate = self
        locationManager.startLocationUpdates()
    }
}
```

Examples



。 。

```
class A { var b: B? = nil }
class B { var a: A? = nil }

let a = A()
let b = B()

a.b = b  // a retains b
b.a = a  // b retains a -- a reference cycle
```

。 。

---

weakunowned 。

```
class A { var b: B? = nil }
class B { var a: A? = nil }

let a = A()
let b = B()

a.b = b  // a retains b
b.a = a  // b retains a -- a reference cycle
```

。 。 **nil** 。

```
class A { var b: B? = nil }
class B { var a: A? = nil }

let a = A()
let b = B()
```

```
a.b = b  // a retains b
b.a = a  // b retains a -- a reference cycle
```

`weakunowned` 。

C APISwift。 。

punnedC`toOpaqueUnmanagedfromOpaque`

```
setupDisplayLink() {
  let pointerToSelf: UnsafeRawPointer = Unmanaged.passUnretained(self).toOpaque()
  CVDisplayLinkSetOutputCallback(self.displayLink, self.redraw, pointerToSelf)
}

func redraw(pointerToSelf: UnsafeRawPointer, /* args omitted */) {
  let recoveredSelf = Unmanaged<Self>.fromOpaque(pointerToSelf).takeUnretainedValue()
  recoveredSelf.doRedraw()
}
```

`passUnretainedunowned`。

Objective-C API。 `Unmanagedretainrelease` 。 `passRetainedtakeRetainedValue`

```
setupDisplayLink() {
  let pointerToSelf: UnsafeRawPointer = Unmanaged.passUnretained(self).toOpaque()
  CVDisplayLinkSetOutputCallback(self.displayLink, self.redraw, pointerToSelf)
}

func redraw(pointerToSelf: UnsafeRawPointer, /* args omitted */) {
  let recoveredSelf = Unmanaged<Self>.fromOpaque(pointerToSelf).takeUnretainedValue()
  recoveredSelf.doRedraw()
}
```

API。

https://riptutorial.com/zh-TW/swift/topic/745/

20:

Examples

```
struct Example {
    var upvotes: Int
    init() {
        upvotes = 42
    }
}
let myExample = Example() // call the initializer
print(myExample.upvotes) // prints: 42
```

```
struct Example {
    var upvotes: Int
    init() {
        upvotes = 42
    }
}
let myExample = Example() // call the initializer
print(myExample.upvotes) // prints: 42
```

- downvotes

```
struct Example {
    var upvotes: Int
    init() {
        upvotes = 42
    }
}
let myExample = Example() // call the initializer
print(myExample.upvotes) // prints: 42
```

```
struct MetricDistance {
    var distanceInMeters: Double

    init(fromCentimeters centimeters: Double) {
        distanceInMeters = centimeters / 100
    }
    init(fromKilometers kilos: Double) {
        distanceInMeters = kilos * 1000
    }
}

let myDistance = MetricDistance(fromCentimeters: 42)
// myDistance.distanceInMeters is 0.42
let myOtherDistance = MetricDistance(fromKilometers: 42)
// myOtherDistance.distanceInMeters is 42000
```

```
struct MetricDistance {
    var distanceInMeters: Double

    init(fromCentimeters centimeters: Double) {
```

```
        distanceInMeters = centimeters / 100
    }
    init(fromKilometers kilos: Double) {
        distanceInMeters = kilos * 1000
    }
}

let myDistance = MetricDistance(fromCentimeters: 42)
// myDistance.distanceInMeters is 0.42
let myOtherDistance = MetricDistance(fromKilometers: 42)
// myOtherDistance.distanceInMeters is 42000
```

_

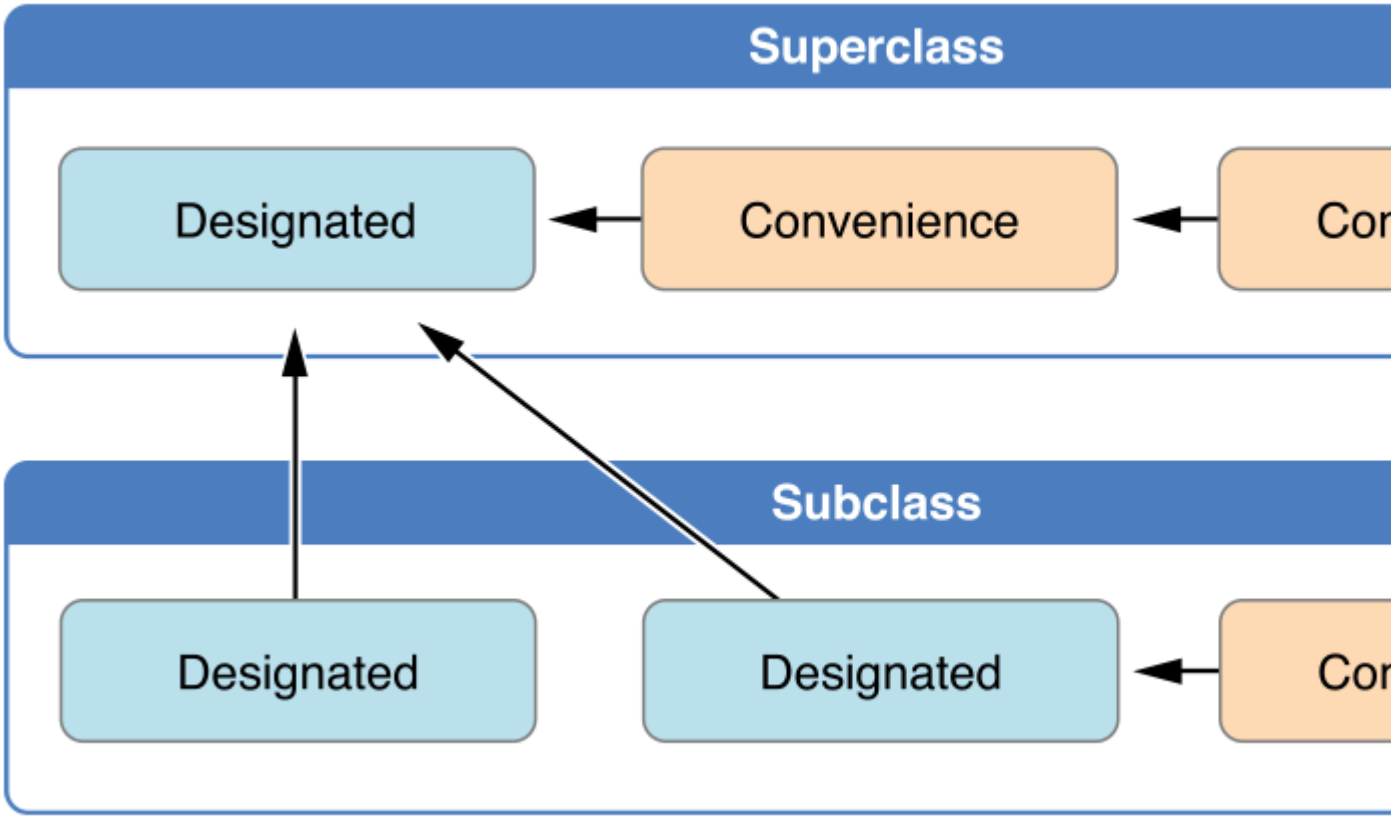
```
struct MetricDistance {
    var distanceInMeters: Double

    init(fromCentimeters centimeters: Double) {
        distanceInMeters = centimeters / 100
    }
    init(fromKilometers kilos: Double) {
        distanceInMeters = kilos * 1000
    }
}

let myDistance = MetricDistance(fromCentimeters: 42)
// myDistance.distanceInMeters is 0.42
let myOtherDistance = MetricDistance(fromKilometers: 42)
// myOtherDistance.distanceInMeters is 42000
```

`self`

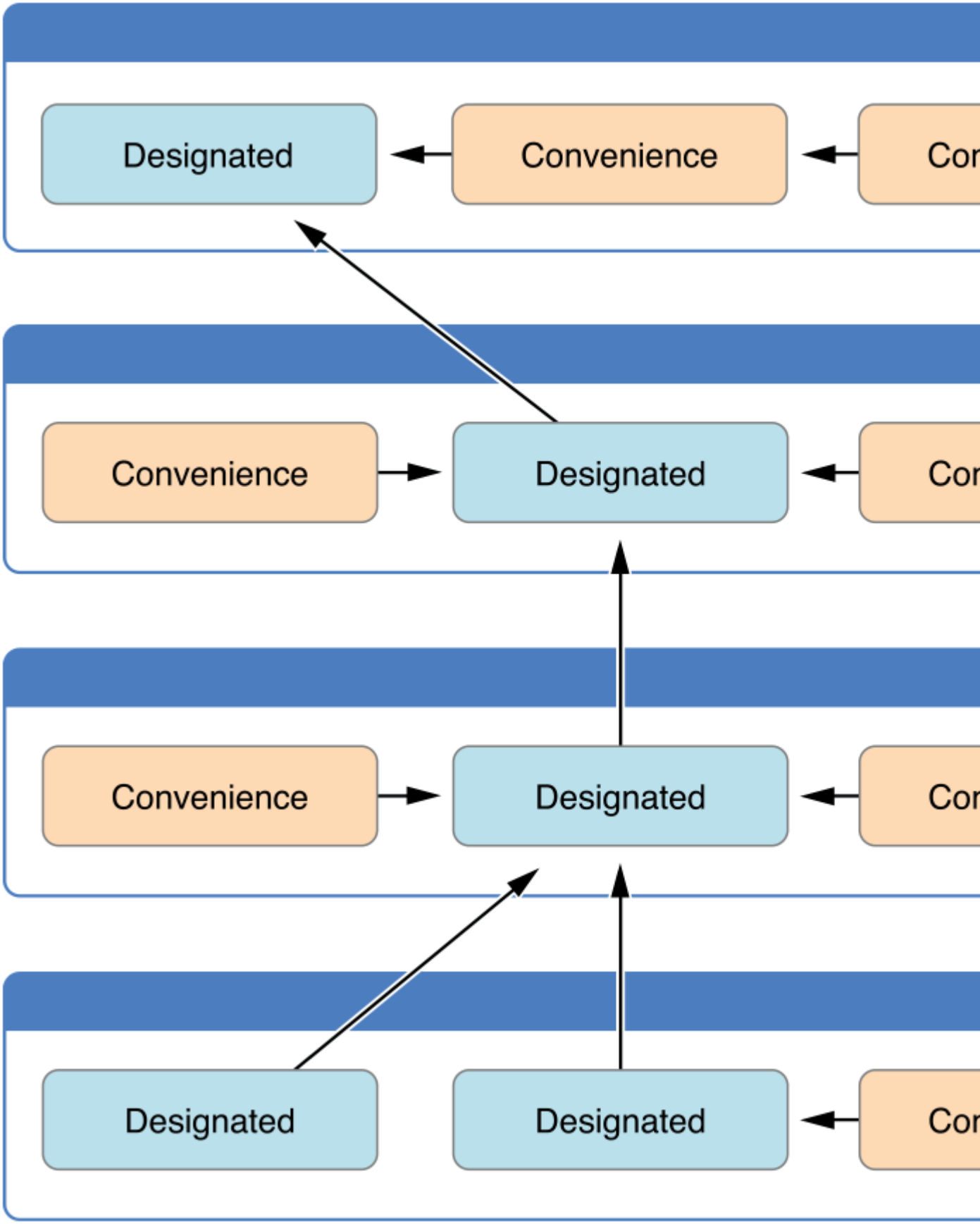
```
struct MetricDistance {
    var distanceInMeters: Double

    init(fromCentimeters centimeters: Double) {
        distanceInMeters = centimeters / 100
    }
    init(fromKilometers kilos: Double) {
        distanceInMeters = kilos * 1000
    }
}

let myDistance = MetricDistance(fromCentimeters: 42)
// myDistance.distanceInMeters is 0.42
let myOtherDistance = MetricDistance(fromKilometers: 42)
// myOtherDistance.distanceInMeters is 42000
```

Swift。Apple3

1. 。


Superclass
Designated
Convenience
Subclass
Designated
Designated

2. ◦
3. ◦




```
class Foo {

    var someString: String
    var someValue: Int
    var someBool: Bool
```

```
    // Designated Initializer
    init(someString: String, someValue: Int, someBool: Bool)
    {
        self.someString = someString
        self.someValue = someValue
        self.someBool = someBool
    }

    // A convenience initializer must call another initializer from the same class.
    convenience init()
    {
        self.init(otherString: "")
    }

    // A convenience initializer must ultimately call a designated initializer.
convenience init(otherString: String)
    {
        self.init(someString: otherString, someValue:  0, someBool: false)
    }
}


class Baz: Foo
{
    var someFloat: Float

    // Designed initializer
    init(someFloat: Float)
    {
        self.someFloat = someFloat

        // A designated initializer must call a designated initializer from its immediate
superclass.
        super.init(someString: "", someValue: 0, someBool: false)
    }

    // A convenience initializer must call another initializer from the same class.
    convenience init()
    {
        self.init(someFloat: 0)
    }
}
```

```
class Foo {

    var someString: String
    var someValue: Int
    var someBool: Bool

    // Designated Initializer
    init(someString: String, someValue: Int, someBool: Bool)
    {
        self.someString = someString
        self.someValue = someValue
        self.someBool = someBool
    }

    // A convenience initializer must call another initializer from the same class.
    convenience init()
```

```
    {
        self.init(otherString: "")
    }

    // A convenience initializer must ultimately call a designated initializer.
convenience init(otherString: String)
    {
        self.init(someString: otherString, someValue:  0, someBool: false)
    }
}


class Baz: Foo
{
    var someFloat: Float

    // Designed initializer
    init(someFloat: Float)
    {
        self.someFloat = someFloat

        // A designated initializer must call a designated initializer from its immediate
superclass.
        super.init(someString: "", someValue: 0, someBool: false)
    }

    // A convenience initializer must call another initializer from the same class.
    convenience init()
    {
        self.init(someFloat: 0)
    }
}
```

init

```
class Foo {

    var someString: String
    var someValue: Int
    var someBool: Bool

    // Designated Initializer
    init(someString: String, someValue: Int, someBool: Bool)
    {
        self.someString = someString
        self.someValue = someValue
        self.someBool = someBool
    }

    // A convenience initializer must call another initializer from the same class.
    convenience init()
    {
        self.init(otherString: "")
    }

    // A convenience initializer must ultimately call a designated initializer.
convenience init(otherString: String)
    {
        self.init(someString: otherString, someValue:  0, someBool: false)
```

```
    }
}


class Baz: Foo
{
    var someFloat: Float

    // Designed initializer
    init(someFloat: Float)
    {
        self.someFloat = someFloat

        // A designated initializer must call a designated initializer from its immediate
superclass.
        super.init(someString: "", someValue: 0, someBool: false)
    }

    // A convenience initializer must call another initializer from the same class.
    convenience init()
    {
        self.init(someFloat: 0)
    }
}
```

initotherStringString

```
class Foo {

    var someString: String
    var someValue: Int
    var someBool: Bool

    // Designated Initializer
    init(someString: String, someValue: Int, someBool: Bool)
    {
        self.someString = someString
        self.someValue = someValue
        self.someBool = someBool
    }

    // A convenience initializer must call another initializer from the same class.
    convenience init()
    {
        self.init(otherString: "")
    }

    // A convenience initializer must ultimately call a designated initializer.
convenience init(otherString: String)
    {
        self.init(someString: otherString, someValue:  0, someBool: false)
    }
}


class Baz: Foo
{
    var someFloat: Float
```

```
    // Designed initializer
    init(someFloat: Float)
    {
        self.someFloat = someFloat

        // A designated initializer must call a designated initializer from its immediate
superclass.
        super.init(someString: "", someValue: 0, someBool: false)
    }

    // A convenience initializer must call another initializer from the same class.
    convenience init()
    {
        self.init(someFloat: 0)
    }
}
```

```
class Foo {

    var someString: String
    var someValue: Int
    var someBool: Bool

    // Designated Initializer
    init(someString: String, someValue: Int, someBool: Bool)
    {
        self.someString = someString
        self.someValue = someValue
        self.someBool = someBool
    }

    // A convenience initializer must call another initializer from the same class.
    convenience init()
    {
        self.init(otherString: "")
    }

    // A convenience initializer must ultimately call a designated initializer.
convenience init(otherString: String)
    {
        self.init(someString: otherString, someValue:  0, someBool: false)
    }
}


class Baz: Foo
{
    var someFloat: Float

    // Designed initializer
    init(someFloat: Float)
    {
        self.someFloat = someFloat

        // A designated initializer must call a designated initializer from its immediate
superclass.
        super.init(someString: "", someValue: 0, someBool: false)
    }

    // A convenience initializer must call another initializer from the same class.
```

```
    convenience init()
    {
        self.init(someFloat: 0)
    }
}
```

init

```
class Foo {

    var someString: String
    var someValue: Int
    var someBool: Bool

    // Designated Initializer
    init(someString: String, someValue: Int, someBool: Bool)
    {
        self.someString = someString
        self.someValue = someValue
        self.someBool = someBool
    }

    // A convenience initializer must call another initializer from the same class.
    convenience init()
    {
        self.init(otherString: "")
    }

    // A convenience initializer must ultimately call a designated initializer.
convenience init(otherString: String)
    {
        self.init(someString: otherString, someValue:  0, someBool: false)
    }
}


class Baz: Foo
{
    var someFloat: Float

    // Designed initializer
    init(someFloat: Float)
    {
        self.someFloat = someFloat

        // A designated initializer must call a designated initializer from its immediate
superclass.
        super.init(someString: "", someValue: 0, someBool: false)
    }

    // A convenience initializer must call another initializer from the same class.
    convenience init()
    {
        self.init(someFloat: 0)
    }
}
```

Swift

Struct

```
enum ValidationError: Error {
    case invalid
}
```

Struct

```
enum ValidationError: Error {
    case invalid
}
```

throwable

```
enum ValidationError: Error {
    case invalid
}
```

https://riptutorial.com/zh-TW/swift/topic/1778/

21:

Examples

◦ hello ◦

```
func hello()
{
    print("Hello World")
}
```

◦

```
func hello()
{
    print("Hello World")
}
```

◦ ◦

```
func magicNumber(number1: Int)
{
    print("\(number1) Is the magic number")
}
```

`\(number1)` String Interpolation String◦

◦

```
func magicNumber(number1: Int)
{
    print("\(number1) Is the magic number")
}
```

Int◦

```
func magicNumber(number1: Int)
{
    print("\(number1) Is the magic number")
}
```

◦

```
func magicNumber(number1: Int)
{
    print("\(number1) Is the magic number")
}
```

。

```
func magicNumber(number1: Int)
{
    print("\(number1) Is the magic number")
}
```

。

```
func magicNumber(number1: Int)
{
    print("\(number1) Is the magic number")
}
```

。

```
func findHypotenuse(a: Double, b: Double) -> Double
{
    return sqrt((a * a) + (b * b))
}

let c = findHypotenuse(3, b: 5)
//c = 5.830951894845301
```

。

```
func findHypotenuse(a: Double, b: Double) -> Double
{
    return sqrt((a * a) + (b * b))
}

let c = findHypotenuse(3, b: 5)
//c = 5.830951894845301
```

throws

```
func errorThrower()throws -> String {}
```

throw

```
func errorThrower()throws -> String {}
```

dotry

```
func errorThrower()throws -> String {}
```

Swift

Swift 。 。

func。

```
class Counter {
    var count = 0
    func increment() {
        count += 1
    }
}
```

Counterincrement()

```
class Counter {
    var count = 0
    func increment() {
        count += 1
    }
}
```

static func。 class func。

```
class Counter {
    var count = 0
    func increment() {
        count += 1
    }
}
```

```
class Counter {
    var count = 0
    func increment() {
        count += 1
    }
}
```

Inout

inout。 inout& 。

```
func updateFruit(fruit: inout Int) {
    fruit -= 1
}

var apples = 30 // Prints "There's 30 apples"
print("There's \(apples) apples")

updateFruit(fruit: &apples)

print("There's now \(apples) apples") // Prints "There's 29 apples".
```

。

```
func loadData(id: String, completion:(result: String) -> ()) {
    // ...
    completion(result:"This is the result data")
}
```

Trailing Closure

```
func loadData(id: String, completion:(result: String) -> ()) {
    // ...
    completion(result:"This is the result data")
}
```

`+ - ??`。

- **-** x
- x **+** y
- x **++**

Swift。

。 `sum`

```
func sum(_ a: Int, _ b: Int) -> Int {
    return a + b
}
```

```
func sum(_ a: Int, _ b: Int) -> Int {
    return a + b
}
```

。 Swift `...`

```
func sum(_ a: Int, _ b: Int) -> Int {
    return a + b
}
```

`numbers[Int]Array`。 `T...[T]`。

```
func sum(_ a: Int, _ b: Int) -> Int {
    return a + b
}
```

Swift。

。 `sum`。 variadic。 `sum`

```
func sum(_ a: Int, _ b: Int) -> Int {
    return a + b
}
```

。

```
struct DaysOfWeek {

  var days = ["Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat"]
```

```
  subscript(index: Int) -> String {
    get {
      return days[index]
    }
    set {
      days[index] = newValue
    }
  }
}
```

```
struct DaysOfWeek {

  var days = ["Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat"]

  subscript(index: Int) -> String {
    get {
      return days[index]
    }
    set {
      days[index] = newValue
    }
  }
}
```

◦ ◦ ◦

```
struct DaysOfWeek {

  var days = ["Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat"]

  subscript(index: Int) -> String {
    get {
      return days[index]
    }
    set {
      days[index] = newValue
    }
  }
}
```

```
struct DaysOfWeek {

  var days = ["Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat"]

  subscript(index: Int) -> String {
    get {
      return days[index]
    }
    set {
      days[index] = newValue
    }
  }
}
```

◦ Void ◦

```
func closedFunc(block: (()->Void)? = nil) {
    print("Just beginning")

    if let block = block {
        block()
    }
}
```


```
func closedFunc(block: (()->Void)? = nil) {
    print("Just beginning")

    if let block = block {
        block()
    }
}
```


print closedFunc()。

```
func closedFunc(block: (()->Void)? = nil) {
    print("Just beginning")

    if let block = block {
        block()
    }
}
```


1

2

```
func jediTrainer () -> ((String, Int) -> String) {
  func train(name: String, times: Int) -> (String) {
    return "\(name) has been trained in the Force \(times) times"
  }
  return train
}

let train = jediTrainer()
train("Obi Wan", 3)
```

。

```
func sum(x: Int, y: Int) -> (result: Int) { return x + y }
```


```
func sum(x: Int, y: Int) -> (result: Int) { return x + y }
```


。

https://riptutorial.com/zh-TW/swift/topic/432/

22:

。 。

Swift。 。

。 gettersetter。

Swift 。

Objective-CSwift。

Java。

Examples

Swift。 “”。

Swift。

Swift/。 SwiftExtensions。 Protocol Extensions。

。 Protocol ExtensionsProtocolsSwift。

Swift。 Swift。 。 OOP。

```
protocol MyProtocol {
    init(value: Int)                        // required initializer
    func doSomething() -> Bool              // instance method
    var message: String { get }             // instance read-only property
    var value: Int { get set }              // read-write instance property
    subscript(index: Int) -> Int { get }    // instance subscript
    static func instructions() -> String    // static method
    static var max: Int { get }             // static read-only property
    static var total: Int { get set }       // read-write static property
}
```

`{ get }``{ get set }`。 `{ get }`gettable。 `{ get set }`gettable。

```
protocol MyProtocol {
    init(value: Int)                        // required initializer
    func doSomething() -> Bool              // instance method
    var message: String { get }             // instance read-only property
    var value: Int { get set }              // read-write instance property
    subscript(index: Int) -> Int { get }    // instance subscript
    static func instructions() -> String    // static method
    static var max: Int { get }             // static read-only property
    static var total: Int { get set }       // read-write static property
}
```

```
protocol MyProtocol {
    init(value: Int)                        // required initializer
    func doSomething() -> Bool              // instance method
    var message: String { get }             // instance read-only property
    var value: Int { get set }              // read-write instance property
    subscript(index: Int) -> Int { get }    // instance subscript
    static func instructions() -> String    // static method
    static var max: Int { get }             // static read-only property
    static var total: Int { get set }       // read-write static property
}
```

associatedtype

```
protocol MyProtocol {
    init(value: Int)                        // required initializer
    func doSomething() -> Bool              // instance method
    var message: String { get }             // instance read-only property
    var value: Int { get set }              // read-write instance property
    subscript(index: Int) -> Int { get }    // instance subscript
    static func instructions() -> String    // static method
    static var max: Int { get }             // static read-only property
    static var total: Int { get set }       // read-write static property
}
```

3.0

Swift 3 &

```
protocol MyProtocol {
    init(value: Int)                        // required initializer
    func doSomething() -> Bool              // instance method
    var message: String { get }             // instance read-only property
    var value: Int { get set }              // read-write instance property
    subscript(index: Int) -> Int { get }    // instance subscript
    static func instructions() -> String    // static method
    static var max: Int { get }             // static read-only property
    static var total: Int { get set }       // read-write static property
}
```

3.0

protocol<...> <>∘

```
protocol MyProtocol {
    init(value: Int)                        // required initializer
    func doSomething() -> Bool              // instance method
    var message: String { get }             // instance read-only property
    var value: Int { get set }              // read-write instance property
    subscript(index: Int) -> Int { get }    // instance subscript
    static func instructions() -> String    // static method
    static var max: Int { get }             // static read-only property
    static var total: Int { get set }       // read-write static property
}
```

```
protocol MyProtocol {
    init(value: Int)                        // required initializer
```

```
    func doSomething() -> Bool            // instance method
    var message: String { get }           // instance read-only property
    var value: Int { get set }            // read-write instance property
    subscript(index: Int) -> Int { get }  // instance subscript
    static func instructions() -> String  // static method
    static var max: Int { get }           // static read-only property
    static var total: Int { get set }     // read-write static property
}
```

associatedtype

```
protocol Container {
    associatedtype Element
    var count: Int { get }
    subscript(index: Int) -> Element { get set }
}
```

```
protocol Container {
    associatedtype Element
    var count: Int { get }
    subscript(index: Int) -> Element { get set }
}
```

associatedtypeassociatedtype

```
protocol Container {
    associatedtype Element
    var count: Int { get }
    subscript(index: Int) -> Element { get set }
}
```

T - Element associatedtype Element。 Elementassociatedtype Element。

typealiasassociatedtype typealias

```
protocol Container {
    associatedtype Element
    var count: Int { get }
    subscript(index: Int) -> Element { get set }
}
```

```
protocol Container {
    associatedtype Element
    var count: Int { get }
    subscript(index: Int) -> Element { get set }
}
```

```
protocol Container {
    associatedtype Element
    var count: Int { get }
    subscript(index: Int) -> Element { get set }
}
```

CocoaCocoaTouch。 。

。 Objectsprotocol。 。

。

。

```
class Parent { }
class Child { }
```

。

Swiftprotocoldelegateprotocol 。 delegateparent。

```
class Parent { }
class Child { }
```

```
class Parent { }
class Child { }
```

delegateChildDelegate delegateweak。 delegate。 。

weakclassChildDelegate。

```
class Parent { }
class Child { }
```

。

ChildDelegate。

```
class Parent { }
class Child { }
```

```
class Parent { }
class Child { }
```

```
class Parent { }
class Child { }
```

Swift protocol。 @objcoptional。

UITableViewiOSUITableViewDelegateUITableViewDataSource。

```
class Parent { }
class Child { }
```

```
class Parent { }
class Child { }
```

optionalUITableViewDelegate。

。

```
protocol MyProtocol {
    func doSomething()
}

extension MyProtocol where Self: UIViewController {
    func doSomething() {
        print("UIViewController default protocol implementation")
    }
}

class MyViewController: UIViewController, MyProtocol { }

let vc = MyViewController()
vc.doSomething() // Prints "UIViewController default protocol implementation"
```

RawRepresentable

```
// RawRepresentable has an associatedType RawValue.
// For this struct, we will make the compiler infer the type
// by implementing the rawValue variable with a type of String
//
// Compiler infers RawValue = String without needing typealias
//
struct NotificationName: RawRepresentable {
    let rawValue: String

    static let dataFinished = NotificationNames(rawValue: "DataFinishedNotification")
}
```


```
// RawRepresentable has an associatedType RawValue.
// For this struct, we will make the compiler infer the type
// by implementing the rawValue variable with a type of String
//
// Compiler infers RawValue = String without needing typealias
//
struct NotificationName: RawRepresentable {
    let rawValue: String

    static let dataFinished = NotificationNames(rawValue: "DataFinishedNotification")
}
```


RawRepresentable

```
// RawRepresentable has an associatedType RawValue.
// For this struct, we will make the compiler infer the type
// by implementing the rawValue variable with a type of String
//
// Compiler infers RawValue = String without needing typealias
```

```
//
struct NotificationName: RawRepresentable {
    let rawValue: String

    static let dataFinished = NotificationNames(rawValue: "DataFinishedNotification")
}
```

NotificationName

```
// RawRepresentable has an associatedType RawValue.
// For this struct, we will make the compiler infer the type
// by implementing the rawValue variable with a type of String
//
// Compiler infers RawValue = String without needing typealias
//
struct NotificationName: RawRepresentable {
    let rawValue: String

    static let dataFinished = NotificationNames(rawValue: "DataFinishedNotification")
}
```

RawRepresentable

```
// RawRepresentable has an associatedType RawValue.
// For this struct, we will make the compiler infer the type
// by implementing the rawValue variable with a type of String
//
// Compiler infers RawValue = String without needing typealias
//
struct NotificationName: RawRepresentable {
    let rawValue: String

    static let dataFinished = NotificationNames(rawValue: "DataFinishedNotification")
}
```

class◦ ◦

```
protocol ClassOnlyProtocol: class, SomeOtherProtocol {
    // Protocol requirements
}
```

ClassOnlyProtocol ◦

```
protocol ClassOnlyProtocol: class, SomeOtherProtocol {
    // Protocol requirements
}
```

ClassOnlyProtocol ◦

```
protocol ClassOnlyProtocol: class, SomeOtherProtocol {
    // Protocol requirements
}
```

。

```
protocol ClassOnlyProtocol: class, SomeOtherProtocol {
    // Protocol requirements
}
```

Foobarfoo。

Foo - var。

```
protocol ClassOnlyProtocol: class, SomeOtherProtocol {
    // Protocol requirements
}
```

```
protocol ClassOnlyProtocol: class, SomeOtherProtocol {
    // Protocol requirements
}
```

weakweak。

```
protocol ClassOnlyProtocol: class, SomeOtherProtocol {
    // Protocol requirements
}
```

Hashable

SetsDictionaries(key)HashableEquatable。

Hashable

- hashValue
- ==!= 。

structHashable

```
struct Cell {
    var row: Int
    var col: Int

    init(_ row: Int, _ col: Int) {
        self.row = row
        self.col = col
    }
}

extension Cell: Hashable {

    // Satisfy Hashable requirement
    var hashValue: Int {
        get {
            return row.hashValue^col.hashValue
        }
    }
```

```
    // Satisfy Equatable requirement
    static func ==(lhs: Cell, rhs: Cell) -> Bool {
        return lhs.col == rhs.col && lhs.row == rhs.row
    }

}

// Now we can make Cell as key of dictonary
var dict = [Cell : String]()

dict[Cell(0, 0)] = "0, 0"
dict[Cell(1, 0)] = "1, 0"
dict[Cell(0, 1)] = "0, 1"

// Also we can create Set of Cells
var set = Set<Cell>()

set.insert(Cell(0, 0))
set.insert(Cell(1, 0))
```

。 。

https://riptutorial.com/zh-TW/swift/topic/241/

23:

- //
- mirror.displayStyle //Xcode
- mirror.description //CustomStringConvertible
- mirror.subjectType //
- mirror.superclassMirror //

1.

`Mirror`Swift`struct` ◦ ◦ `Core Data`◦ `structNSManagedObject` ◦

2.

`Mirrorchildren`Mirror◦ `childlabelvalue` ◦ `titleGame of Thrones: A Song of Ice and FireGame of Thrones: A Song of Ice and Fire`◦

Examples

Mirror

```
class Project {
    var title: String = ""
    var id: Int = 0
    var platform: String = ""
    var version: Int = 0
    var info: String?
}
```

◦ Project◦

```
class Project {
    var title: String = ""
    var id: Int = 0
    var platform: String = ""
    var version: Int = 0
    var info: String?
}
```

Mirror◦ children`AnyForwardCollection<Child>` `Child`subjecttypealias◦ `Childlabel: Stringvalue: Any` ◦

```
class Project {
    var title: String = ""
    var id: Int = 0
    var platform: String = ""
    var version: Int = 0
    var info: String?
}
```

XcodePlaygroundConsolefor for。

```
class Project {
    var title: String = ""
    var id: Int = 0
    var platform: String = ""
    var version: Int = 0
    var info: String?
}
```

Xcode 8 beta 2Playground

`value.dynamicType` Swift 3 `type(of: value)` Swift 2 `value.dynamicType` Swift`Mirror` 。

`NSObject` `class_copyPropertyList` `property_getAttributes` - 。 Github

```
func getTypesOfProperties(in clazz: NSObject.Type) -> Dictionary<String, Any>? {
    var count = UInt32()
    guard let properties = class_copyPropertyList(clazz, &count) else { return nil }
    var types: Dictionary<String, Any> = [:]
    for i in 0..<Int(count) {
        guard let property: objc_property_t = properties[i], let name = getNameOf(property:
property) else { continue }
        let type = getTypeOf(property: property)
        types[name] = type
    }
    free(properties)
    return types
}

func getTypeOf(property: objc_property_t) -> Any {
    guard let attributesAsNSString: NSString = NSString(utf8String:
property_getAttributes(property)) else { return Any.self }
    let attributes = attributesAsNSString as String
    let slices = attributes.components(separatedBy: "\"")
    guard slices.count > 1 else { return getPrimitiveDataType(withAttributes: attributes) }
    let objectClassName = slices[1]
    let objectClass = NSClassFromString(objectClassName) as! NSObject.Type
    return objectClass
}

   func getPrimitiveDataType(withAttributes attributes: String) -> Any {
        guard let letter = attributes.substring(from: 1, to: 2), let type =
primitiveDataTypes[letter] else { return Any.self }
        return type
    }
```

`primitiveDataTypes`Dictionary

```
func getTypesOfProperties(in clazz: NSObject.Type) -> Dictionary<String, Any>? {
    var count = UInt32()
    guard let properties = class_copyPropertyList(clazz, &count) else { return nil }
    var types: Dictionary<String, Any> = [:]
    for i in 0..<Int(count) {
        guard let property: objc_property_t = properties[i], let name = getNameOf(property:
property) else { continue }
        let type = getTypeOf(property: property)
```

```
        types[name] = type
    }
    free(properties)
    return types
}

func getTypeOf(property: objc_property_t) -> Any {
    guard let attributesAsNSString: NSString = NSString(utf8String:
property_getAttributes(property)) else { return Any.self }
    let attributes = attributesAsNSString as String
    let slices = attributes.components(separatedBy: "\"")
    guard slices.count > 1 else { return getPrimitiveDataType(withAttributes: attributes) }
    let objectClassName = slices[1]
    let objectClass = NSClassFromString(objectClassName) as! NSObject.Type
    return objectClass
}

   func getPrimitiveDataType(withAttributes attributes: String) -> Any {
        guard let letter = attributes.substring(from: 1, to: 2), let type =
primitiveDataTypes[letter] else { return Any.self }
        return type
    }
```

`NSObject.TypeNSObjectNSDate` Swift3 `Date` `NSString` Swift3 `String` `NSNumber` `Any`Dictionary。 IntInt32 Bool `value types`。 NSObject `.self`Int- `Int.self.self`NSObject.Type`Any`。 `Dictionary<String, Any>?` `Dictionary<String, NSObject.Type>?` 。

```
func getTypesOfProperties(in clazz: NSObject.Type) -> Dictionary<String, Any>? {
    var count = UInt32()
    guard let properties = class_copyPropertyList(clazz, &count) else { return nil }
    var types: Dictionary<String, Any> = [:]
    for i in 0..<Int(count) {
        guard let property: objc_property_t = properties[i], let name = getNameOf(property:
property) else { continue }
        let type = getTypeOf(property: property)
        types[name] = type
    }
    free(properties)
    return types
}

func getTypeOf(property: objc_property_t) -> Any {
    guard let attributesAsNSString: NSString = NSString(utf8String:
property_getAttributes(property)) else { return Any.self }
    let attributes = attributesAsNSString as String
    let slices = attributes.components(separatedBy: "\"")
    guard slices.count > 1 else { return getPrimitiveDataType(withAttributes: attributes) }
    let objectClassName = slices[1]
    let objectClass = NSClassFromString(objectClassName) as! NSObject.Type
    return objectClass
}

   func getPrimitiveDataType(withAttributes attributes: String) -> Any {
        guard let letter = attributes.substring(from: 1, to: 2), let type =
primitiveDataTypes[letter] else { return Any.self }
        return type
    }
```

`AnyNSObject.Type`

`NSObject`standard ==

```
func getTypesOfProperties(in clazz: NSObject.Type) -> Dictionary<String, Any>? {
    var count = UInt32()
    guard let properties = class_copyPropertyList(clazz, &count) else { return nil }
    var types: Dictionary<String, Any> = [:]
    for i in 0..<Int(count) {
        guard let property: objc_property_t = properties[i], let name = getNameOf(property:
property) else { continue }
        let type = getTypeOf(property: property)
        types[name] = type
    }
    free(properties)
    return types
}

func getTypeOf(property: objc_property_t) -> Any {
    guard let attributesAsNSString: NSString = NSString(utf8String:
property_getAttributes(property)) else { return Any.self }
    let attributes = attributesAsNSString as String
    let slices = attributes.components(separatedBy: "\"")
    guard slices.count > 1 else { return getPrimitiveDataType(withAttributes: attributes) }
    let objectClassName = slices[1]
    let objectClass = NSClassFromString(objectClassName) as! NSObject.Type
    return objectClass
}

   func getPrimitiveDataType(withAttributes attributes: String) -> Any {
        guard let letter = attributes.substring(from: 1, to: 2), let type =
primitiveDataTypes[letter] else { return Any.self }
        return type
    }
```

==

```
func getTypesOfProperties(in clazz: NSObject.Type) -> Dictionary<String, Any>? {
    var count = UInt32()
    guard let properties = class_copyPropertyList(clazz, &count) else { return nil }
    var types: Dictionary<String, Any> = [:]
    for i in 0..<Int(count) {
        guard let property: objc_property_t = properties[i], let name = getNameOf(property:
property) else { continue }
        let type = getTypeOf(property: property)
        types[name] = type
    }
    free(properties)
    return types
}

func getTypeOf(property: objc_property_t) -> Any {
    guard let attributesAsNSString: NSString = NSString(utf8String:
property_getAttributes(property)) else { return Any.self }
    let attributes = attributesAsNSString as String
    let slices = attributes.components(separatedBy: "\"")
    guard slices.count > 1 else { return getPrimitiveDataType(withAttributes: attributes) }
    let objectClassName = slices[1]
    let objectClass = NSClassFromString(objectClassName) as! NSObject.Type
    return objectClass
}
```

```
   func getPrimitiveDataType(withAttributes attributes: String) -> Any {
        guard let letter = attributes.substring(from: 1, to: 2), let type =
primitiveDataTypes[letter] else { return Any.self }
        return type
    }
```

value types

```
func getTypesOfProperties(in clazz: NSObject.Type) -> Dictionary<String, Any>? {
    var count = UInt32()
    guard let properties = class_copyPropertyList(clazz, &count) else { return nil }
    var types: Dictionary<String, Any> = [:]
    for i in 0..<Int(count) {
        guard let property: objc_property_t = properties[i], let name = getNameOf(property:
property) else { continue }
        let type = getTypeOf(property: property)
        types[name] = type
    }
    free(properties)
    return types
}

func getTypeOf(property: objc_property_t) -> Any {
    guard let attributesAsNSString: NSString = NSString(utf8String:
property_getAttributes(property)) else { return Any.self }
    let attributes = attributesAsNSString as String
    let slices = attributes.components(separatedBy: "\"")
    guard slices.count > 1 else { return getPrimitiveDataType(withAttributes: attributes) }
    let objectClassName = slices[1]
    let objectClass = NSClassFromString(objectClassName) as! NSObject.Type
    return objectClass
}

   func getPrimitiveDataType(withAttributes attributes: String) -> Any {
        guard let letter = attributes.substring(from: 1, to: 2), let type =
primitiveDataTypes[letter] else { return Any.self }
        return type
    }
```

value types。 NSObject var myOptionalInt: Int? class_copyPropertyList。

https://riptutorial.com/zh-TW/swift/topic/1201/

24:

Swift Documentarion

- @escaping。

Examples

Swift 12。 @noescape。

Swift 3。 @escaping。

```
class ClassOne {
  // @noescape is applied here as default
  func methodOne(completion: () -> Void) {
    //
  }
}

class ClassTwo {
  let obj = ClassOne()
  var greeting = "Hello, World!"

  func methodTwo() {
    obj.methodOne() {
      // self.greeting is required
      print(greeting)
    }
  }
}
```

Swift Documentarion

@escaping

- - self。 -

```
class ClassThree {

    var closure: (() -> ())?

    func doSomething(completion: @escaping () -> ()) {
        closure = finishBlock
    }
}
```

- @escaping。

https://riptutorial.com/zh-TW/swift/topic/8623/

25:

。

Examples

。 。

Dictionary

```
var books : Dictionary<Int, String> = Dictionary<Int, String>()
```

```
var books : Dictionary<Int, String> = Dictionary<Int, String>()
```

。 。

```
var books : Dictionary<Int, String> = Dictionary<Int, String>()
```

Dictionary

```
var books = [Int: String]()
//books = [:]
books[5] = "Book 5"
//books = [5: "Book 5"]
books.updateValue("Book 6", forKey: 5)
//[5: "Book 6"]
```

updateValue。

```
var books = [Int: String]()
//books = [:]
books[5] = "Book 5"
//books = [5: "Book 5"]
books.updateValue("Book 6", forKey: 5)
//[5: "Book 6"]
```

```
var books = [Int: String]()
//books = [:]
books[5] = "Book 5"
//books = [5: "Book 5"]
books.updateValue("Book 6", forKey: 5)
//[5: "Book 6"]
```

Dictionary

```
var books: [Int: String] = [1: "Book 1", 2: "Book 2"]
let bookName = books[1]
//bookName = "Book 1"
```

valuesvalues

```
var books: [Int: String] = [1: "Book 1", 2: "Book 2"]
let bookName = books[1]
//bookName = "Book 1"
```

keyskeys

```
var books: [Int: String] = [1: "Book 1", 2: "Book 2"]
let bookName = books[1]
//bookName = "Book 1"
```

keyvalue

```
var books: [Int: String] = [1: "Book 1", 2: "Book 2"]
let bookName = books[1]
//bookName = "Book 1"
```

Array Dictionary - 。

Dictionary。

```
var books: [Int: String] = [1: "Book 1", 2: "Book 2"]
let bookName = books[1]
//bookName = "Book 1"
```

Key

```
var dict = ["name": "John", "surname": "Doe"]
// Set the element with key: 'name' to 'Jane'
dict["name"] = "Jane"
print(dict)
```

```
let myAllKeys = ["name" : "Kirit" , "surname" : "Modi"]
let allKeys = Array(myAllKeys.keys)
print(allKeys)
```

```
extension Dictionary {
    func merge(dict: Dictionary<Key,Value>) -> Dictionary<Key,Value> {
        var mutableCopy = self
        for (key, value) in dict {
            // If both dictionaries have a value for same key, the value of the other
dictionary is used.
            mutableCopy[key] = value
        }
        return mutableCopy
    }
}
```

https://riptutorial.com/zh-TW/swift/topic/310/

26:

- String.characters //String
- String.characters.count //
- String.utf8 // String.UTF8ViewStringUTF-8
- String.utf16 // String.UTF16ViewStringUTF-16
- String.unicodeScalars // String.UnicodeScalarViewStringUTF-32
- String.isEmpty //Stringtrue
- String.hasPrefixString//Stringtrue
- String.hasSuffixString//Stringtrue
- String.startIndex //
- String.endIndex //
- String.componentsseparatedByString//
- String.appendCharacter//String

Swift`String`Unicode。 Swift StringsUnicodeUnicodeemojis。

String。

Swift 。

“Swift String Design”

Examples

Swift "

```
let greeting = "Hello!"  // greeting's type is String
```


```
let greeting = "Hello!"  // greeting's type is String
```

。 。

`\(value)` 。 。


```
let greeting = "Hello!"  // greeting's type is String
```


`"\(myobj)"String(myobj) print(myobj)。 CustomStringConvertible。`

3.0

Swift 3SE-0089 `String.init<T>(_:)String.init<T>(describing:)` 。

```
"\(myobj)"String.init<T: LosslessStringConvertible>(_:)LosslessStringConvertible
init<T>(describing:)
```

。

\0	
\\	\
\t	
\v	
\r	
\n	“”
\"	"
\'	'
\u{n}	Unicode*n*

```
let greeting = "Hello!"  // greeting's type is String
```

+

```
let name = "John"
let surname = "Appleseed"
let fullName = name + " " + surname  // fullName is "John Appleseed"
```

+=

```
let name = "John"
let surname = "Appleseed"
let fullName = name + " " + surname  // fullName is "John Appleseed"
```

```
let name = "John"
let surname = "Appleseed"
let fullName = name + " " + surname  // fullName is "John Appleseed"
```

```
let name = "John"
let surname = "Appleseed"
let fullName = name + " " + surname  // fullName is "John Appleseed"
```

3.0

appendContentsOf(_:)append(_:) 。

joinWithSeparator(_:)

```
let name = "John"
let surname = "Appleseed"
```

```
let fullName = name + " " + surname  // fullName is "John Appleseed"
```

3.0

`joinWithSeparator(_:)``joinWithSeparator(_:)` `joined(separator:)` 。

`separator["a", "b", "c"].joined() == "abc"` 。

```
if str.isEmpty {
    // do something if the string is empty
}

// If the string is empty, replace it with a fallback:
let result = str.isEmpty ? "fallback string" : str
```

Unicode

```
if str.isEmpty {
    // do something if the string is empty
}

// If the string is empty, replace it with a fallback:
let result = str.isEmpty ? "fallback string" : str
```

/

```
if str.isEmpty {
    // do something if the string is empty
}

// If the string is empty, replace it with a fallback:
let result = str.isEmpty ? "fallback string" : str
```

Swift StringUnicode。 。

```
let str = "ที่□①!"
```

`characters`Unicode

```
let str = "ที่□①!"
```

`unicodeScalars`Unicode ที่3--3607,3637,3656 - `characters`

```
let str = "ที่□①!"
```

UTF-8 `UInt8`UTF-16 `UInt16`

```
let str = "ที่□①!"
```

---

characters unicodeScalars utf8utf16Collection count

```
let str = "ที่□①!"
```

```
let str = "ที่□①!"
```

Unicode

```
var str: String = "I want to visit □□, Москва, मुंबई, القاهرة, and □□□. □"
var character: Character = "□"
```

```
var str: String = "I want to visit □□, Москва, मुंबई, القاهرة, and □□□. □"
var character: Character = "□"
```

Swift Characterunicode。 。

- >

```
var str: String = "I want to visit □□, Москва, मुंबई, القاهرة, and □□□. □"
var character: Character = "□"
```

- >

```
var str: String = "I want to visit □□, Москва, मुंबई, القاهرة, and □□□. □"
var character: Character = "□"
```

UTF-8UTF-16UTF-16。 。

2.2

```
let aString = "This is a test string."

// first, reverse the String's characters
let reversedCharacters = aString.characters.reverse()

// then convert back to a String with the String() initializer
let reversedString = String(reversedCharacters)

print(reversedString) // ".gnirts tset a si sihT"
```

3.0

```
let aString = "This is a test string."

// first, reverse the String's characters
let reversedCharacters = aString.characters.reverse()

// then convert back to a String with the String() initializer
```

```
let reversedString = String(reversedCharacters)

print(reversedString) // ".gnirts tset a si sihT"
```

2.2

```
let text = "AaBbCc"
let uppercase = text.uppercaseString // "AABBCC"
let lowercase = text.lowercaseString // "aabbcc"
```

3.0

```
let text = "AaBbCc"
let uppercase = text.uppercaseString // "AABBCC"
let lowercase = text.lowercaseString // "aabbcc"
```

String

3.0

```
let letters = CharacterSet.letters

let phrase = "Test case"
let range = phrase.rangeOfCharacter(from: letters)

// range will be nil if no letters is found
if let test = range {
   print("letters found")
}
else {
   print("letters not found")
}
```

2.2

```
let letters = CharacterSet.letters

let phrase = "Test case"
let range = phrase.rangeOfCharacter(from: letters)

// range will be nil if no letters is found
if let test = range {
   print("letters found")
}
else {
   print("letters not found")
}
```

Objective-C `NSCharacterSetCharacterSet`

- `decimalDigits`
- `capitalizedLetters`
- `alphanumerics`
- `controlCharacters`
- `illegalCharacters`
-

NSCharacterSet。

3.0

```
let letters = CharacterSet.letters

let phrase = "Test case"
let range = phrase.rangeOfCharacter(from: letters)

// range will be nil if no letters is found
if let test = range {
   print("letters found")
}
else {
   print("letters not found")
}
```

2.2

```
let letters = CharacterSet.letters

let phrase = "Test case"
let range = phrase.rangeOfCharacter(from: letters)

// range will be nil if no letters is found
if let test = range {
   print("letters found")
}
else {
   print("letters not found")
}
```

3.0

```
let letters = CharacterSet.letters

let phrase = "Test case"
let range = phrase.rangeOfCharacter(from: letters)

// range will be nil if no letters is found
if let test = range {
   print("letters found")
}
else {
   print("letters not found")
}
```

`StringCharacter`

```
let text = "Hello World"
let char: Character = "o"
```

`CharacterString`

```
let text = "Hello World"
let char: Character = "o"
```

Set

2.2

```
func removeCharactersNotInSetFromText(text: String, set: Set<Character>) -> String {
   return String(text.characters.filter { set.contains( $0) })
}

let text = "Swift 3.0 Come Out"
var chars =
Set([Character]("abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLKMNOPQRSTUVWXYZ".characters))
let newText = removeCharactersNotInSetFromText(text, set: chars) // "SwiftComeOut"
```

3.0

```
func removeCharactersNotInSetFromText(text: String, set: Set<Character>) -> String {
   return String(text.characters.filter { set.contains( $0) })
}

let text = "Swift 3.0 Come Out"
var chars =
Set([Character]("abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLKMNOPQRSTUVWXYZ".characters))
let newText = removeCharactersNotInSetFromText(text, set: chars) // "SwiftComeOut"
```

```
let number: Int = 7
let str1 = String(format: "%03d", number) // 007
let str2 = String(format: "%05d", number) // 00007
```


```
let number: Int = 7
let str1 = String(format: "%03d", number) // 007
let str2 = String(format: "%05d", number) // 00007
```

```
let number: Int = 7
let str1 = String(format: "%03d", number) // 007
let str2 = String(format: "%05d", number) // 00007
```

```
let number: Int = 7
let str1 = String(format: "%03d", number) // 007
let str2 = String(format: "%05d", number) // 00007
```

```
let number: Int = 7
let str1 = String(format: "%03d", number) // 007
let str2 = String(format: "%05d", number) // 00007
```


Radix `[2, 36] Int ◦`

Swift

```
Int("123") // Returns 123 of Int type
Int("abcd") // Returns nil
Int("10") // Returns 10 of Int type
```

```
Int("10", radix: 2) // Returns 2 of Int type
Double("1.5") // Returns 1.5 of Double type
Double("abcd") // Returns nil
```

Optional 。

3.0

```
let string = "My fantastic string"
var index = string.startIndex

while index != string.endIndex {
    print(string[index])
    index = index.successor()
}
```

endIndexstring[string.endIndex]string[string.startIndex] 。 "" string.startIndex == string.endIndextrue 。 startIndex.successor() 。

3.0

Swift 3Stringsuccessor() predecessor() advancedBy(_:) advancedBy(_:limit:)distanceTo(_:) 。

。

.index(after:) ,. .index(before:).index(_:, offsetBy:) 。

```
let string = "My fantastic string"
var index = string.startIndex

while index != string.endIndex {
    print(string[index])
    index = index.successor()
}
```

currentIndex.index。

3.0

```
let string = "My fantastic string"
var index = string.startIndex

while index != string.endIndex {
    print(string[index])
    index = index.successor()
}
```

3.0

```
let string = "My fantastic string"
var index = string.startIndex

while index != string.endIndex {
```

```
    print(string[index])
    index = index.successor()
}
```

`IndexInt` 。

```
let string = "My fantastic string"
var index = string.startIndex

while index != string.endIndex {
    print(string[index])
    index = index.successor()
}
```

3.0

```
let string = "My fantastic string"
var index = string.startIndex

while index != string.endIndex {
    print(string[index])
    index = index.successor()
}
```

3.0

```
let string = "My fantastic string"
var index = string.startIndex

while index != string.endIndex {
    print(string[index])
    index = index.successor()
}
```

3.0

```
let string = "My fantastic string"
var index = string.startIndex

while index != string.endIndex {
    print(string[index])
    index = index.successor()
}
```

3.0

```
let string = "My fantastic string"
var index = string.startIndex

while index != string.endIndex {
    print(string[index])
    index = index.successor()
}
```

3.0

```
let string = "My fantastic string"
var index = string.startIndex

while index != string.endIndex {
    print(string[index])
    index = index.successor()
}
```

3.0

```
let string = "My fantastic string"
var index = string.startIndex

while index != string.endIndex {
    print(string[index])
    index = index.successor()
}
```

```
let string = "My fantastic string"
var index = string.startIndex

while index != string.endIndex {
    print(string[index])
    index = index.successor()
}
```

WhiteSpaceNewLine

3.0

```
let someString = "  Swift Language  \n"
let trimmedString =
someString.stringByTrimmingCharactersInSet(NSCharacterSet.whitespaceAndNewlineCharacterSet())
// "Swift Language"
```

`stringByTrimmingCharactersInSet`String。

。

```
let someString = "  Swift Language  \n"
let trimmedString =
someString.stringByTrimmingCharactersInSet(NSCharacterSet.whitespaceAndNewlineCharacterSet())
// "Swift Language"
```

```
let someString = "  Swift Language  \n"
let trimmedString =
someString.stringByTrimmingCharactersInSet(NSCharacterSet.whitespaceAndNewlineCharacterSet())
// "Swift Language"
```

3.0

```
let someString = "  Swift Language  \n"
let trimmedString =
someString.stringByTrimmingCharactersInSet(NSCharacterSet.whitespaceAndNewlineCharacterSet())
```

```
// "Swift Language"
```

Foundation ◦ *CocoaUIKit* *import Foundationimport Foundation* ◦

Data / NSData

StringData / NSDataData / NSDataString◦ `UTF-8` Unicode8ASCII◦ `String Encodings`

`StringData` / `NSData`

3.0

```
let data = string.data(using: .utf8)
```

2.2

```
let data = string.data(using: .utf8)
```

`Data` / `NSData` to `String`

3.0

```
let data = string.data(using: .utf8)
```

2.2

```
let data = string.data(using: .utf8)
```

SwiftStringString

3.0

```
let startDate = "23:51"

let startDateAsArray = startDate.components(separatedBy: ":") // ["23", "51"]`
```

2.2

```
let startDate = "23:51"

let startDateAsArray = startDate.components(separatedBy: ":") // ["23", "51"]`
```

3.0

```
let startDate = "23:51"

let startDateAsArray = startDate.components(separatedBy: ":") // ["23", "51"]`
```

2.2

```
let startDate = "23:51"
```

```
let startDateAsArray = startDate.components(separatedBy: ":") // ["23", "51"]`
```

https://riptutorial.com/zh-TW/swift/topic/320/

27:

。

Examples

```
func sampleWithCompletion(completion:@escaping (()-> ())){
    let delayInSeconds = 1.0
    DispatchQueue.main.asyncAfter(deadline: DispatchTime.now() + delayInSeconds) {

        completion()

    }
}

//Call the function
sampleWithCompletion {
    print("after one second")
}
```

```
enum ReadResult{
    case Successful
    case Failed
    case Pending
}

struct OutpuData {
    var data = Data()
    var result: ReadResult
    var error: Error?
}

func readData(from url: String, completion: @escaping (OutpuData) ->  Void) {
    var _data = OutpuData(data: Data(), result: .Pending, error: nil)
    DispatchQueue.global().async {
        let url=URL(string: url)
        do {
            let rawData = try Data(contentsOf: url!)
            _data.result = .Successful
            _data.data = rawData
            completion(_data)
        }
        catch let error {
            _data.result = .Failed
            _data.error = error
            completion(_data)
        }

    }
}

readData(from: "https://raw.githubusercontent.com/trev/bearcal/master/sample-data-large.json")
{ (output) in
    switch output.result {
    case .Successful:
        break
```

```
    case .Failed:
        break
    case .Pending:
        break

    }
}
```

https://riptutorial.com/zh-TW/swift/topic/9378/

28:

Examples

MD2MD4MD5SHA1SHA224SHA256SHA384SHA512Swift 3

StringData。

nameString
MD2MD4MD5SHA1SHA224SHA256SHA384SHA512

Common Crypto

```
#import <CommonCrypto/CommonCrypto.h>
```

Security.framework。

```
name: A name of a hash function as a String
data: The Data to be hashed
returns: the hashed result as Data
```

```
name: A name of a hash function as a String
data: The Data to be hashed
returns: the hashed result as Data
```

String

```
name: A name of a hash function as a String
data: The Data to be hashed
returns: the hashed result as Data
```

```
name: A name of a hash function as a String
data: The Data to be hashed
returns: the hashed result as Data
```

```
name: A name of a hash function as a String
data: The Data to be hashed
returns: the hashed result as Data
```

```
name: A name of a hash function as a String
data: The Data to be hashed
returns: the hashed result as Data
```

HMACMD5SHA1SHA224SHA256SHA384SHA512Swift 3

StringData。

nameMD5SHA1SHA224SHA256SHA384SHA512

Common Crypto

```
#import <CommonCrypto/CommonCrypto.h>
```

Security.framework。

```
hashName: name of a hash function as String
message:  message as Data
key:      key as Data
returns:  digest as Data
```

```
hashName: name of a hash function as String
message:  message as Data
key:      key as Data
returns:  digest as Data
```

```
hashName: name of a hash function as String
message:  message as Data
key:      key as Data
returns:  digest as Data
```

```
hashName: name of a hash function as String
message:  message as Data
key:      key as Data
returns:  digest as Data
```

```
hashName: name of a hash function as String
message:  message as Data
key:      key as Data
returns:  digest as Data
```

```
hashName: name of a hash function as String
message:  message as Data
key:      key as Data
returns:  digest as Data
```

//

```
hashName: name of a hash function as String
message:  message as Data
key:      key as Data
returns:  digest as Data
```

```
hashName: name of a hash function as String
message:  message as Data
key:      key as Data
returns:  digest as Data
```

https://riptutorial.com/zh-TW/swift/topic/7885/

29:

Examples

Bool

Bool `truefalse`。

```
let aTrueBool = true
let aFalseBool = false
```

Bools。 `if`

```
let aTrueBool = true
let aFalseBool = false
```

Bool

`!` operator。 `!truefalse` `!falsetrue`。

```
print(!true)  // prints "false"
print(!false) // prints "true"

func test(_ someBoolean: Bool) {
    if !someBoolean {
        print("someBoolean is false")
    }
}
```

trueOR||truefalse。 trueORtrue

```
if (10 < 20) || (20 < 10) {
    print("Expression is true")
}
```

trueAND&&true。 falsetrue

```
if (10 < 20) || (20 < 10) {
    print("Expression is true")
}
```

trueXOR^true。 truetrue

```
if (10 < 20) || (20 < 10) {
    print("Expression is true")
}
```

a bcSwift 。

3

```
question ? answerIfTrue : answerIfFalse
```

questionanswerIfTruefalseanswerIfFalse。

```
question ? answerIfTrue : answerIfFalse
```

```
question ? answerIfTrue : answerIfFalse
```

```
question ? answerIfTrue : answerIfFalse
```

https://riptutorial.com/zh-TW/swift/topic/735/

30:

Examples

defer。

guard。 defer。

。

```
func doSomething() {
    let data = UnsafeMutablePointer<UInt8>(allocatingCapacity: 42)
    // this pointer would not be released when the function returns
    // so we add a defer-statement
    defer {
        data.deallocateCapacity(42)
    }
    // it will be executed when the function returns.

    guard condition else {
        return /* will execute defer-block */
    }

}   // The defer-block will also be executed on the end of the function.
```


```
func doSomething() {
    let data = UnsafeMutablePointer<UInt8>(allocatingCapacity: 42)
    // this pointer would not be released when the function returns
    // so we add a defer-statement
    defer {
        data.deallocateCapacity(42)
    }
    // it will be executed when the function returns.

    guard condition else {
        return /* will execute defer-block */
    }

}   // The defer-block will also be executed on the end of the function.
```


defer。 defer。

```
postfix func ++ (inout value: Int) -> Int {
    defer { value += 1 } // do NOT do this!
    return value
}
```

https://riptutorial.com/zh-TW/swift/topic/4932/

31: StringUIImage

UIImage。 。

Examples

InitialsImageFactory

```
class InitialsImageFactory: NSObject {

class func imageWith(name: String?) -> UIImage? {

let frame = CGRect(x: 0, y: 0, width: 50, height: 50)
let nameLabel = UILabel(frame: frame)
nameLabel.textAlignment = .center
nameLabel.backgroundColor = .lightGray
nameLabel.textColor = .white
nameLabel.font = UIFont.boldSystemFont(ofSize: 20)
var initials = ""

if let initialsArray = name?.components(separatedBy: " ") {

  if let firstWord = initialsArray.first {
    if let firstLetter = firstWord.characters.first {
      initials += String(firstLetter).capitalized
    }

  }
  if initialsArray.count > 1, let lastWord = initialsArray.last {
    if let lastLetter = lastWord.characters.first {
      initials += String(lastLetter).capitalized
    }

  }
} else {
  return nil
}

nameLabel.text = initials
UIGraphicsBeginImageContext(frame.size)
if let currentContext = UIGraphicsGetCurrentContext() {
  nameLabel.layer.render(in: currentContext)
  let nameImage = UIGraphicsGetImageFromCurrentImageContext()
  return nameImage
}
return nil
}

}
```

StringUIImage https://riptutorial.com/zh-TW/swift/topic/10915/stringuiimage

32:

- {statements}
- for condition in condition where condition {statements}
- for var{statements}
- for _ in sequence {statements}
- for case{statements}
- for casecondition {statements}
- case var{statements}
- {}
- {statements}
- sequence.forEachbodyElementthrows - > Void

Examples

For-in

for-in。

```
for i in 0..<3 {
    print(i)
}

for i in 0...2 {
    print(i)
}

// Both print:
// 0
// 1
// 2
```

```
for i in 0..<3 {
    print(i)
}

for i in 0...2 {
    print(i)
}

// Both print:
// 0
// 1
// 2
```

2.1 2.2

`SequenceTypeenumerate()`。

```
for i in 0..<3 {
```

```
    print(i)
}

for i in 0...2 {
    print(i)
}

// Both print:
// 0
// 1
// 2
```

`enumerate()` `Int` 0。 - 。

3.0

Swift 3 `enumerate()` `enumerated()`

```
for i in 0..<3 {
    print(i)
}

for i in 0...2 {
    print(i)
}

// Both print:
// 0
// 1
// 2
```

```
for i in 0..<3 {
    print(i)
}

for i in 0...2 {
    print(i)
}

// Both print:
// 0
// 1
// 2
```

2.1 2.2

`SequenceType` `reverse()`

```
for i in 0..<3 {
    print(i)
}

for i in 0...2 {
    print(i)
}
```

```
// Both print:
// 0
// 1
// 2
```

3.0

Swift 3 `reverse()` `reversed()`

```
for i in 0..<3 {
    print(i)
}

for i in 0...2 {
    print(i)
}

// Both print:
// 0
// 1
// 2
```

2.1 2.2

`Strideablestride(_:_:)`

```
for i in 0..<3 {
    print(i)
}

for i in 0...2 {
    print(i)
}

// Both print:
// 0
// 1
// 2
```

1.2 3.0

Swift 3 `Stridablestride(_:_:)` `stride(_:_:_:)`

```
for i in 0..<3 {
    print(i)
}

for i in 0...2 {
    print(i)
}

// Both print:
// 0
// 1
// 2
```

while。 。

```
var i: Int = 0

repeat {
   print(i)
   i += 1
} while i < 3

// 0
// 1
// 2
```

while。

```
var count = 1

while count < 10 {
    print("This is the \(count) run of the loop")
    count += 1
}
```

SequenceType

```
collection.forEach { print($0) }
```

```
collection.forEach { print($0) }
```

*。 returncontinue。 。

```
collection.forEach { print($0) }
```

For-in

1. **where**

where。

```
for i in 0..<5 where i % 2 == 0 {
    print(i)
}

// 0
// 2
// 4


let names = ["James", "Emily", "Miles"]

for name in names where name.characters.contains("s") {
    print(name)
}
```

```
// James
// Miles
```

2. **case**

```
for i in 0..<5 where i % 2 == 0 {
    print(i)
}

// 0
// 2
// 4


let names = ["James", "Emily", "Miles"]

for name in names where name.characters.contains("s") {
    print(name)
}

// James
// Miles
```

?

```
for i in 0..<5 where i % 2 == 0 {
    print(i)
}

// 0
// 2
// 4


let names = ["James", "Emily", "Miles"]

for name in names where name.characters.contains("s") {
    print(name)
}

// James
// Miles
```

break◦

```
var peopleArray = ["John", "Nicole", "Thomas", "Richard", "Brian", "Novak", "Vick", "Amanda",
"Sonya"]
var positionOfNovak = 0

for person in peopleArray {
    if person == "Novak" { break }
    positionOfNovak += 1
}

print("Novak is the element located on position [\(positionOfNovak)] in peopleArray.")
//prints out: Novak is the element located on position 5 in peopleArray. (which is true)
```

https://riptutorial.com/zh-TW/swift/topic/1186/

33:

Examples

Swift。 Swift。 。 。 。

。 。

```
class MyClass {

    let myProperty: String

}
```

Swift。 。 。 。

```
class MyClass {

    let myProperty: String

}
```

。 /。

```
class MyClass {

    let myProperty: String

}
```

。 struct。

。

https://riptutorial.com/zh-TW/swift/topic/4067/

34:

Swift。

Examples

/get。

```
extension ExtensionOf {
    //new functions and get-variables
}
```

self

String.length()

```
extension ExtensionOf {
    //new functions and get-variables
}
```

get。 .length

```
extension ExtensionOf {
    //new functions and get-variables
}
```

。 IntNSString

```
extension Int {
    init?(_ string: NSString) {
        self.init(string as String)  // delegate to the existing Int.init(String) initializer
    }
}

let str1: NSString = "42"
Int(str1) // 42

let str2: NSString = "abc"
Int(str2) // nil
```

Swift。 。 。

Int 。

```
extension Int {
    var factorial: Int {
        return (1..<self+1).reduce(1, combine: *)
    }
}
```

Int API。

```
extension Int {
    var factorial: Int {
        return (1..<self+1).reduce(1, combine: *)
    }
}
```

Swift 2.2。

。

```
protocol FooProtocol {
    func doSomething()
}

extension FooProtocol {
    func doSomething() {
        print("Hi")
    }
}

class Foo: FooProtocol {
    func myMethod() {
        doSomething() // By just implementing the protocol this method is available
    }
}
```

。

where。

```
extension Array where Element: StringLiteralConvertible {
  func toUpperCase() -> [String] {
      var result = [String]()
      for value in self {
          result.append(String(value).uppercaseString)
      }
      return result
    }
}
```

```
extension Array where Element: StringLiteralConvertible {
  func toUpperCase() -> [String] {
      var result = [String]()
      for value in self {
          result.append(String(value).uppercaseString)
      }
      return result
    }
}
```

。 。

Swift

-
-
-
-
-
-

Swift Extensions

- Swift
- UIKit / Foundation
-
- //

-

```
extension Bool {
    public mutating func toggle() -> Bool {
        self = !self
        return self
    }
}

var myBool: Bool = true
print(myBool.toggle()) // false
```

。

String

2.2

```
extension String {
    subscript(index: Int) -> Character {
        let newIndex = startIndex.advancedBy(index)
        return self[newIndex]
    }
}

var myString = "StackOverFlow"
print(myString[2]) // a
print(myString[3]) // c
```

3.0

```
extension String {
    subscript(index: Int) -> Character {
        let newIndex = startIndex.advancedBy(index)
        return self[newIndex]
    }
}

var myString = "StackOverFlow"
print(myString[2]) // a
```

```
print(myString[3]) // c
```

https://riptutorial.com/zh-TW/swift/topic/324/

35:

Examples

Swift

- Intunsigned UInt 。
- Int8 Int16 Int32 Int64UInt8 UInt16 UInt32 UInt64 。
- Float32 / Float Float64 / DoubleFloat80 x86。

```
let x = 42    // x is Int by default
let y = 42.0  // y is Double by default

let z: UInt = 42     // z is UInt
let w: Float = -1    // w is Float
let q = 100 as Int8  // q is Int8
```

_ 。 。

«significand» **e** *«exponent»*; **0x** *«significand»* **p** *«exponent»*。


```
let x = 42    // x is Int by default
let y = 42.0  // y is Double by default

let z: UInt = 42     // z is UInt
let w: Float = -1    // w is Float
let q = 100 as Int8  // q is Int8
```

```
let x = 42    // x is Int by default
let y = 42.0  // y is Double by default

let z: UInt = 42     // z is UInt
let w: Float = -1    // w is Float
let q = 100 as Int8  // q is Int8
```


```
func doSomething1(value: Double) { /* ... */ }
func doSomething2(value: UInt) { /* ... */ }

let x = 42                // x is an Int
doSomething1(Double(x))   // convert x to a Double
doSomething2(UInt(x))     // convert x to a UInt
```


```
func doSomething1(value: Double) { /* ... */ }
func doSomething2(value: UInt) { /* ... */ }

let x = 42                // x is an Int
doSomething1(Double(x))   // convert x to a Double
doSomething2(UInt(x))     // convert x to a UInt
```

```
func doSomething1(value: Double) { /* ... */ }
func doSomething2(value: UInt) { /* ... */ }

let x = 42                // x is an Int
doSomething1(Double(x))  // convert x to a Double
doSomething2(UInt(x))    // convert x to a UInt
```


```
func doSomething1(value: Double) { /* ... */ }
func doSomething2(value: UInt) { /* ... */ }

let x = 42                // x is an Int
doSomething1(Double(x))  // convert x to a Double
doSomething2(UInt(x))    // convert x to a UInt
```


String

```
String(1635999)                              // returns "1635999"
String(1635999, radix: 10)                   // returns "1635999"
String(1635999, radix: 2)                    // returns "110001111011010011111"
String(1635999, radix: 16)                   // returns "18f69f"
String(1635999, radix: 16, uppercase: true)  // returns "18F69F"
String(1635999, radix: 17)                   // returns "129gf4"
String(1635999, radix: 36)                   // returns "z2cf"
```


```
String(1635999)                              // returns "1635999"
String(1635999, radix: 10)                   // returns "1635999"
String(1635999, radix: 2)                    // returns "110001111011010011111"
String(1635999, radix: 16)                   // returns "18f69f"
String(1635999, radix: 16, uppercase: true)  // returns "18F69F"
String(1635999, radix: 17)                   // returns "129gf4"
String(1635999, radix: 36)                   // returns "z2cf"
```



```
String(1635999)                              // returns "1635999"
String(1635999, radix: 10)                   // returns "1635999"
String(1635999, radix: 2)                    // returns "110001111011010011111"
String(1635999, radix: 16)                   // returns "18f69f"
String(1635999, radix: 16, uppercase: true)  // returns "18F69F"
String(1635999, radix: 17)                   // returns "129gf4"
String(1635999, radix: 36)                   // returns "z2cf"
```


---

x.5-x.5。

```
round(3.000) // 3
round(3.001) // 3
round(3.499) // 3
round(3.500) // 4
round(3.999) // 4

round(-3.000) // -3
round(-3.001) // -3
round(-3.499) // -3
round(-3.500) // -4  *** careful here ***
round(-3.999) // -4
```

。

```
round(3.000) // 3
round(3.001) // 3
round(3.499) // 3
round(3.500) // 4
round(3.999) // 4

round(-3.000) // -3
round(-3.001) // -3
round(-3.499) // -3
round(-3.500) // -4  *** careful here ***
round(-3.999) // -4
```

。

```
round(3.000) // 3
round(3.001) // 3
round(3.499) // 3
round(3.500) // 4
round(3.999) // 4

round(-3.000) // -3
round(-3.001) // -3
round(-3.499) // -3
round(-3.500) // -4  *** careful here ***
round(-3.999) // -4
```

DoubleInt 。

```
round(3.000) // 3
round(3.001) // 3
round(3.499) // 3
round(3.500) // 4
round(3.999) // 4

round(-3.000) // -3
round(-3.001) // -3
round(-3.499) // -3
round(-3.500) // -4  *** careful here ***
round(-3.999) // -4
```

- round ceilfloor6432。

arc4random_uniform(someNumber: UInt32) -> UInt32

0someNumber - 1。

UInt32 4,294,967,295 2^32 - 1 。

-

```
let flip = arc4random_uniform(2) // 0 or 1
```

- ```
  let flip = arc4random_uniform(2) // 0 or 1
  ```
- ```
  let flip = arc4random_uniform(2) // 0 or 1
  ```
- 2090

  ```
  let flip = arc4random_uniform(2) // 0 or 1
  ```

```
let flip = arc4random_uniform(2) // 0 or 1
```

`number maxminUInt32 。`

- `arc4randomarc4random_uniform。`
- `UInt32Int。`

Swift`pow()Double`

```
pow(BASE, EXPONENT)
```

base52

```
pow(BASE, EXPONENT)
```

https://riptutorial.com/zh-TW/swift/topic/454/

36:

Array。 。 ArrayElement。 - 。

- Array <Element> //Element
- [Element] //Element
- [element0element1element2... elementN] //
- [Element] //[Element]
- ArraycountrepeatedValue :) //`countrepeatedValue`
- Array_ :) //

。 。

Examples

。

。

```
var originalArray = ["Swift", "is", "great!"]
var newArray = originalArray
newArray[2] = "awesome!"
//originalArray = ["Swift", "is", "great!"]
//newArray = ["Swift", "is", "awesome!"]
```

。 。

`Array`Swift。 O1。 Array 。

`Array var let` 。

`[Int] IntArray<T>` 。

Swift。

```
// A mutable array of Strings, initially empty.

var arrayOfStrings: [String] = []       // type annotation + array literal
var arrayOfStrings = [String]()         // invoking the [String] initializer
var arrayOfStrings = Array<String>()    // without syntactic sugar
```

```
// A mutable array of Strings, initially empty.

var arrayOfStrings: [String] = []       // type annotation + array literal
var arrayOfStrings = [String]()         // invoking the [String] initializer
var arrayOfStrings = Array<String>()    // without syntactic sugar
```

```
// A mutable array of Strings, initially empty.

var arrayOfStrings: [String] = []      // type annotation + array literal
var arrayOfStrings = [String]()        // invoking the [String] initializer
var arrayOfStrings = Array<String>()   // without syntactic sugar
```

```
// A mutable array of Strings, initially empty.

var arrayOfStrings: [String] = []      // type annotation + array literal
var arrayOfStrings = [String]()        // invoking the [String] initializer
var arrayOfStrings = Array<String>()   // without syntactic sugar
```

```
// A mutable array of Strings, initially empty.

var arrayOfStrings: [String] = []      // type annotation + array literal
var arrayOfStrings = [String]()        // invoking the [String] initializer
var arrayOfStrings = Array<String>()   // without syntactic sugar
```

Swift `Int[[Int]]` `Array<Array<Int>>` 。


```
// A mutable array of Strings, initially empty.

var arrayOfStrings: [String] = []      // type annotation + array literal
var arrayOfStrings = [String]()        // invoking the [String] initializer
var arrayOfStrings = Array<String>()   // without syntactic sugar
```



```
// A mutable array of Strings, initially empty.

var arrayOfStrings: [String] = []      // type annotation + array literal
var arrayOfStrings = [String]()        // invoking the [String] initializer
var arrayOfStrings = Array<String>()   // without syntactic sugar
```


```
var exampleArray:[Int] = [1,2,3,4,5]
//exampleArray = [1, 2, 3, 4, 5]
```


```
var exampleArray:[Int] = [1,2,3,4,5]
//exampleArray = [1, 2, 3, 4, 5]
```


2`Array` 。 `Array` `Array`0。


```
var exampleArray:[Int] = [1,2,3,4,5]
//exampleArray = [1, 2, 3, 4, 5]
```


`Array`


```
var exampleArray:[Int] = [1,2,3,4,5]
//exampleArray = [1, 2, 3, 4, 5]
```

```
var exampleArray:[Int] = [1,2,3,4,5]
//exampleArray = [1, 2, 3, 4, 5]
```

nil。

```
var exampleArray:[Int] = [1,2,3,4,5]
//exampleArray = [1, 2, 3, 4, 5]
```

Array。 Arraynil。

```
var exampleArray:[Int] = [1,2,3,4,5]
//exampleArray = [1, 2, 3, 4, 5]
```

```
var exampleArray = [1,2,3,4,5]
exampleArray.isEmpty //false
exampleArray.count //5
```

。

```
var exampleArray = [1,2,3,4,5]
exampleArray.isEmpty //false
exampleArray.count //5
```

```
var exampleArray = [1,2,3,4,5]
exampleArray.append(6)
//exampleArray = [1, 2, 3, 4, 5, 6]
var sixOnwards = [7,8,9,10]
exampleArray += sixOnwards
//exampleArray = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

```
var exampleArray = [1,2,3,4,5]
exampleArray.append(6)
//exampleArray = [1, 2, 3, 4, 5, 6]
var sixOnwards = [7,8,9,10]
exampleArray += sixOnwards
//exampleArray = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

```
var array = [3, 2, 1]
```

`ArraySequenceType`。

2.1 2.2

Swift 2`sort()`。

```
var array = [3, 2, 1]
```

3.0

Swift 3sorted() 。

```
var array = [3, 2, 1]
```

ArrayMutableCollectionType 。

2.1 2.2

Swift 2sortInPlace()。

```
var array = [3, 2, 1]
```

3.0

Swift 3sort() 。

```
var array = [3, 2, 1]
```

Comparable。

- Comparable。 LandmarkComparable - 。

```
var array = [3, 2, 1]
```

2.1 2.2

```
var array = [3, 2, 1]
```

3.0

```
var array = [3, 2, 1]
```

。

map_ :)

ArraySequenceType map(_:)AB(A) throws -> B 。

IntString

```
let numbers = [1, 2, 3, 4, 5]
let words = numbers.map { String($0) }
print(words) // ["1", "2", "3", "4", "5"]
```

map(_:)。 。

StringInt

```
let numbers = [1, 2, 3, 4, 5]
```

```
let words = numbers.map { String($0) }
print(words) // ["1", "2", "3", "4", "5"]
```

`map(_:)` transform - `Int`2

```
let numbers = [1, 2, 3, 4, 5]
let words = numbers.map { String($0) }
print(words) // ["1", "2", "3", "4", "5"]
```

flatMap_ :)Array

`things` Array`Any`。

```
let things: [Any] = [1, "Hello", 2, true, false, "World", 3]
```

。 `Int(s)Int` Array。

```
let things: [Any] = [1, "Hello", 2, true, false, "World", 3]
```

`numbers[Int]` 。 `flatMapnil`

```
let things: [Any] = [1, "Hello", 2, true, false, "World", 3]
```

`SequenceTypefilter(_:)`。

`[Int]`

```
let numbers = [22, 41, 23, 30]

let evenNumbers = numbers.filter { $0 % 2 == 0 }

print(evenNumbers)  // [22, 30]
```

30`[Person]`

```
let numbers = [22, 41, 23, 30]

let evenNumbers = numbers.filter { $0 % 2 == 0 }

print(evenNumbers)  // [22, 30]
```

flatMap_ :)nil

`flatMap(_:)map(_:)` 。

```
extension SequenceType {
    public func flatMap<T>(@noescape transform: (Self.Generator.Element) throws -> T?)
rethrows -> [T]
}
```

flatMap(_:)OptionalT?。 nil - [T]。

IntString[String][Int]

```
extension SequenceType {
    public func flatMap<T>(@noescape transform: (Self.Generator.Element) throws -> T?)
rethrows -> [T]
}
```

flatMap(_:)nil

```
extension SequenceType {
    public func flatMap<T>(@noescape transform: (Self.Generator.Element) throws -> T?)
rethrows -> [T]
}
```

RangeArray。

```
let words = ["Hey", "Hello", "Bonjour", "Welcome", "Hi", "Hola"]
let range = 2...4
let slice = words[range] // ["Bonjour", "Welcome", "Hi"]
```

RangeArraySlice 。 Array。

ArraySlice<String> 。

ArraySliceCollectionTypesort filter。

Array()

```
let words = ["Hey", "Hello", "Bonjour", "Welcome", "Hi", "Hola"]
let range = 2...4
let slice = words[range] // ["Bonjour", "Welcome", "Hi"]
```

```
let words = ["Hey", "Hello", "Bonjour", "Welcome", "Hi", "Hola"]
let range = 2...4
let slice = words[range] // ["Bonjour", "Welcome", "Hi"]
```

```
struct Box {
    let name: String
    let thingsInside: Int
}
```

Box(es)

```
struct Box {
    let name: String
    let thingsInside: Int
}
```

thingsInsidethingsInside Dictionary key。

```
struct Box {
    let name: String
    let thingsInside: Int
}
```

`[Int:[Box]]`

```
struct Box {
    let name: String
    let thingsInside: Int
}
```

flatMap_ :)

`nil flatMap(_:)S`

```
extension SequenceType {
    public func flatMap<S : SequenceType>(transform: (Self.Generator.Element) throws -> S)
rethrows -> [S.Generator.Element]
}
```

- `[S.Generator.Element]` ◦

```
extension SequenceType {
    public func flatMap<S : SequenceType>(transform: (Self.Generator.Element) throws -> S)
rethrows -> [S.Generator.Element]
}
```

1. `primes[String] flatMap(_:)` ◦
2. `primesString Array<String>.Generator.Element` ◦
3. `String.CharacterView`◦
4. - `[String.CharacterView.Generator.Element]` ◦

`flatMap(_:)` - 2D1D3D2D◦

`$0`

```
extension SequenceType {
    public func flatMap<S : SequenceType>(transform: (Self.Generator.Element) throws -> S)
rethrows -> [S.Generator.Element]
}
```

3.0

`sorted()`

```
let words = ["Hello", "Bonjour", "Salute", "Ahola"]
let sortedWords = words.sorted()
print(sortedWords) // ["Ahola", "Bonjour", "Hello", "Salute"]
```

`sort()`

```
let words = ["Hello", "Bonjour", "Salute", "Ahola"]
let sortedWords = words.sorted()
print(sortedWords) // ["Ahola", "Bonjour", "Hello", "Salute"]
```


```
let words = ["Hello", "Bonjour", "Salute", "Ahola"]
let sortedWords = words.sorted()
print(sortedWords) // ["Ahola", "Bonjour", "Hello", "Salute"]
```

```
let words = ["Hello", "Bonjour", "Salute", "Ahola"]
let sortedWords = words.sorted()
print(sortedWords) // ["Ahola", "Bonjour", "Hello", "Salute"]
```

```
let words = ["Hello", "Bonjour", "Salute", "Ahola"]
let sortedWords = words.sorted()
print(sortedWords) // ["Ahola", "Bonjour", "Hello", "Salute"]
```

```
let words = ["Hello", "Bonjour", "Salute", "Ahola"]
let sortedWords = words.sorted()
print(sortedWords) // ["Ahola", "Bonjour", "Hello", "Salute"]
```


import Foundation NSString caseInsensitiveCompare


```
let words = ["Hello", "Bonjour", "Salute", "Ahola"]
let sortedWords = words.sorted()
print(sortedWords) // ["Ahola", "Bonjour", "Hello", "Salute"]
```


localizedCaseInsensitiveCompare ◦

.numericcompare


```
let words = ["Hello", "Bonjour", "Salute", "Ahola"]
let sortedWords = words.sorted()
print(sortedWords) // ["Ahola", "Bonjour", "Hello", "Salute"]
```


flatten

flatten() ◦

2D1D

```
// A 2D array of type [[Int]]
let array2D = [[1, 3], [4], [6, 8, 10], [11]]

// A FlattenBidirectionalCollection<[[Int]]>
let lazilyFlattenedArray = array2D.flatten()

print(lazilyFlattenedArray.contains(4)) // true
```

flatten() FlattenBidirectionalCollection ◦ contains(_:) array2Darray2D - ◦

Arrayreduce_combine :)

reduce(_:combine:)。 - 。

```
let numbers = [2, 5, 7, 8, 10, 4]

let sum = numbers.reduce(0) {accumulator, element in
    return accumulator + element
}

print(sum) // 36
```

0。 NsumN + 36 。 reduce。 accumulator。 element。

(Int, Int) -> Intreduce - +Swift

```
let numbers = [2, 5, 7, 8, 10, 4]

let sum = numbers.reduce(0) {accumulator, element in
    return accumulator + element
}

print(sum) // 36
```

remove(at:)。

Swift3

```
extension Array where Element: Equatable {

    mutating func remove(_ element: Element) {
        _ = index(of: element).flatMap {
            self.remove(at: $0)
        }
    }
}
```

```
extension Array where Element: Equatable {

    mutating func remove(_ element: Element) {
        _ = index(of: element).flatMap {
            self.remove(at: $0)
        }
    }
}
```

array.remove(25) -
cannot convert value to expected argument type

2.1 2.2

minElement()maxElement()。

```
let numbers = [2, 6, 1, 25, 13, 7, 9]

let minimumNumber = numbers.minElement() // Optional(1)
let maximumNumber = numbers.maxElement() // Optional(25)
```

3.0

Swift 3`min()` `max()`

```
let numbers = [2, 6, 1, 25, 13, 7, 9]

let minimumNumber = numbers.minElement() // Optional(1)
let maximumNumber = numbers.maxElement() // Optional(25)
```

- `nil` 。

  `Comparable`。

`Comparable`。

```
let numbers = [2, 6, 1, 25, 13, 7, 9]

let minimumNumber = numbers.minElement() // Optional(1)
let maximumNumber = numbers.maxElement() // Optional(25)
```

2.1 2.2

```
let numbers = [2, 6, 1, 25, 13, 7, 9]

let minimumNumber = numbers.minElement() // Optional(1)
let maximumNumber = numbers.maxElement() // Optional(25)
```

3.0

```
let numbers = [2, 6, 1, 25, 13, 7, 9]

let minimumNumber = numbers.minElement() // Optional(1)
let maximumNumber = numbers.maxElement() // Optional(25)
```

。

```
extension Array {
    subscript (safe index: Int) -> Element? {
        return indices ~= index ? self[index] : nil
    }
}
```

```
extension Array {
    subscript (safe index: Int) -> Element? {
        return indices ~= index ? self[index] : nil
    }
}
```

2

`zipSequenceType`2`Zip2Sequence` 。

```
let nums = [1, 2, 3]
let animals = ["Dog", "Cat", "Tiger"]
let numsAndAnimals = zip(nums, animals)
```

nomsAndAnimals

1	1
1	"Dog"
2	"Cat"
3	"Tiger"

n。

2`Int(s)`

```
let nums = [1, 2, 3]
let animals = ["Dog", "Cat", "Tiger"]
let numsAndAnimals = zip(nums, animals)
```

`list1list0list0` 。

```
let nums = [1, 2, 3]
let animals = ["Dog", "Cat", "Tiger"]
let numsAndAnimals = zip(nums, animals)
```

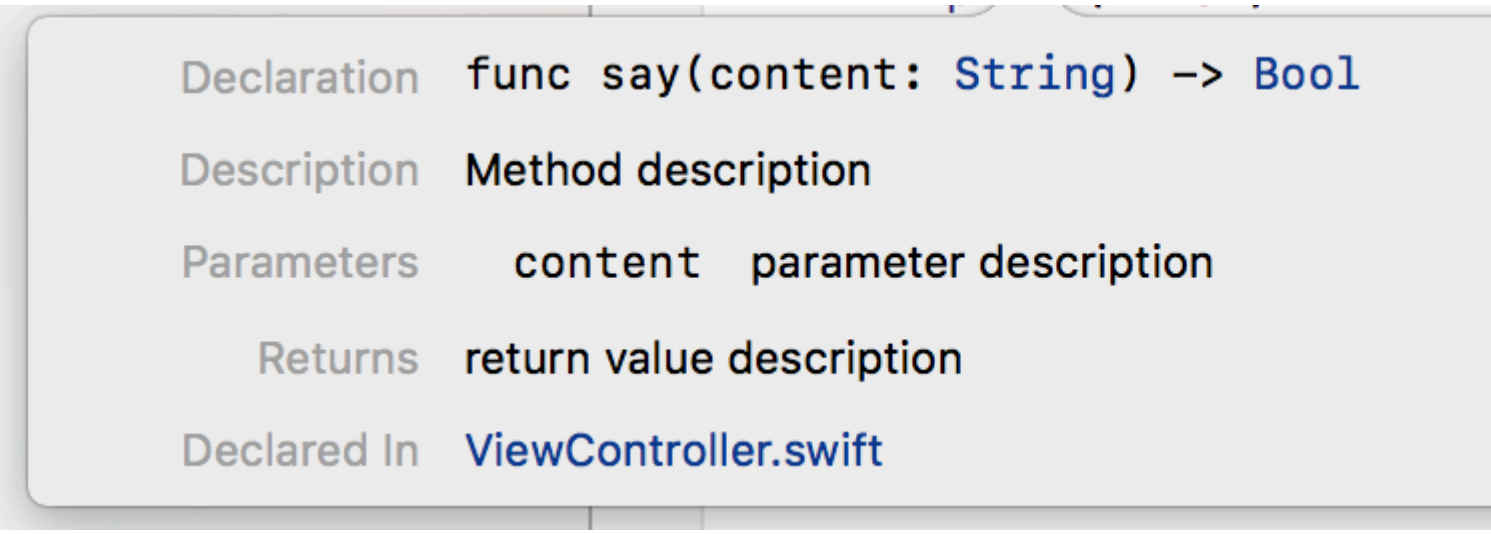
https://riptutorial.com/zh-TW/swift/topic/284/

37:

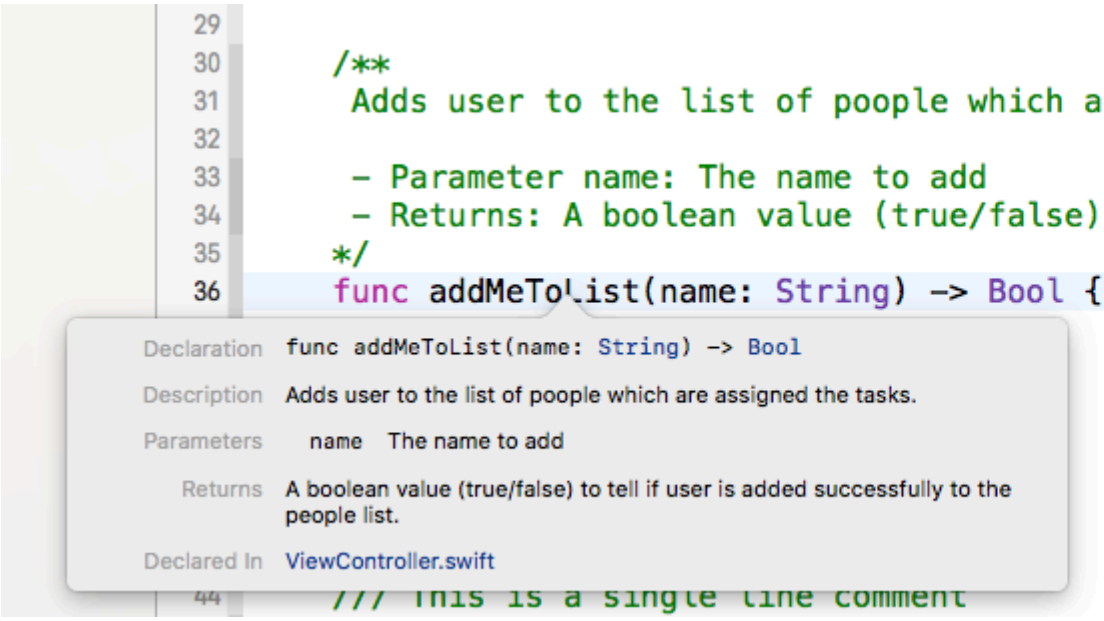
Examples

```
/// Class description
class Student {

    // Member description
    var name: String

    /// Method description
    ///
    /// - parameter content:   parameter description
    ///
    /// - returns: return value description
    func say(content: String) -> Bool {
        print("\(self.name) say \(content)")
        return true
    }
}
```

Xcode 8 ▯▯ + ▯▯ + /。

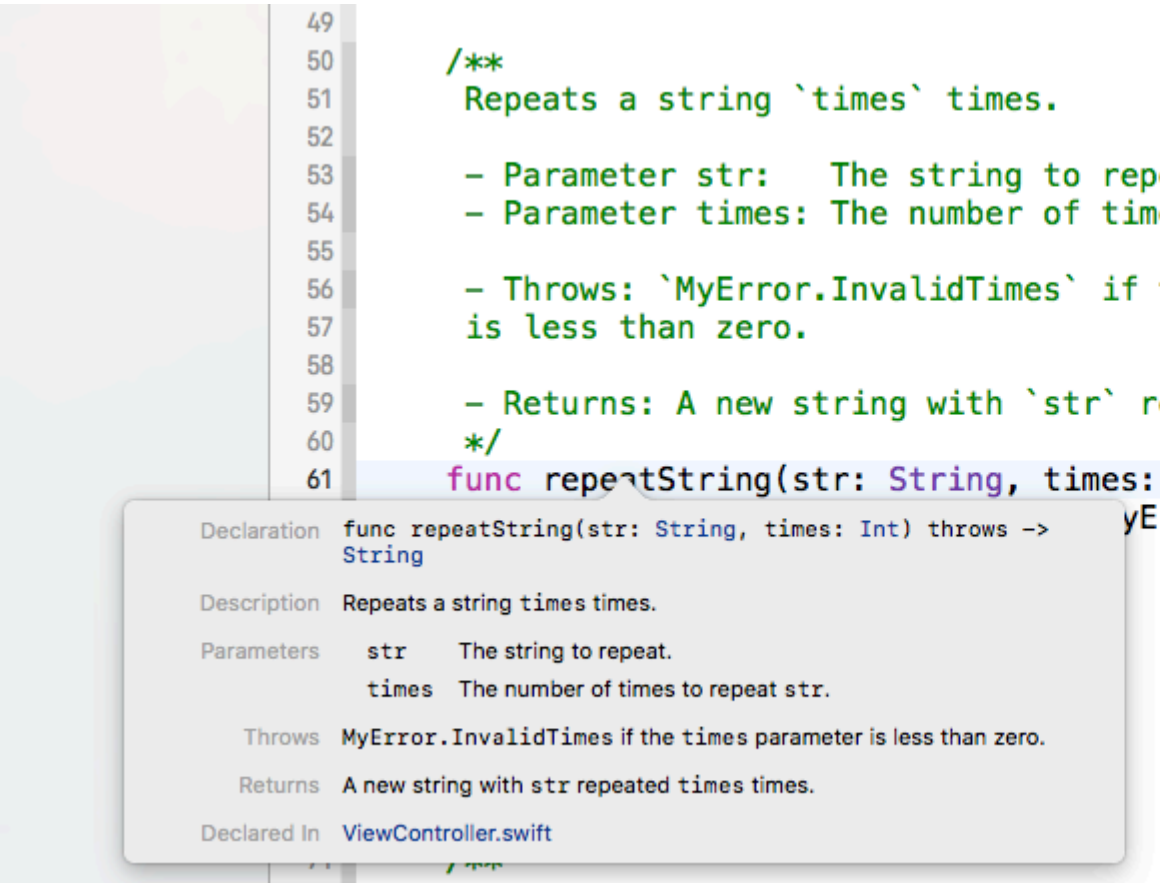



```
/**
 Adds user to the list of poople which are assigned the tasks.

 - Parameter name: The name to add
 - Returns: A boolean value (true/false) to tell if user is added successfully to the people
list.
*/
func addMeToList(name: String) -> Bool {

    // Do something....


    return true
}
```

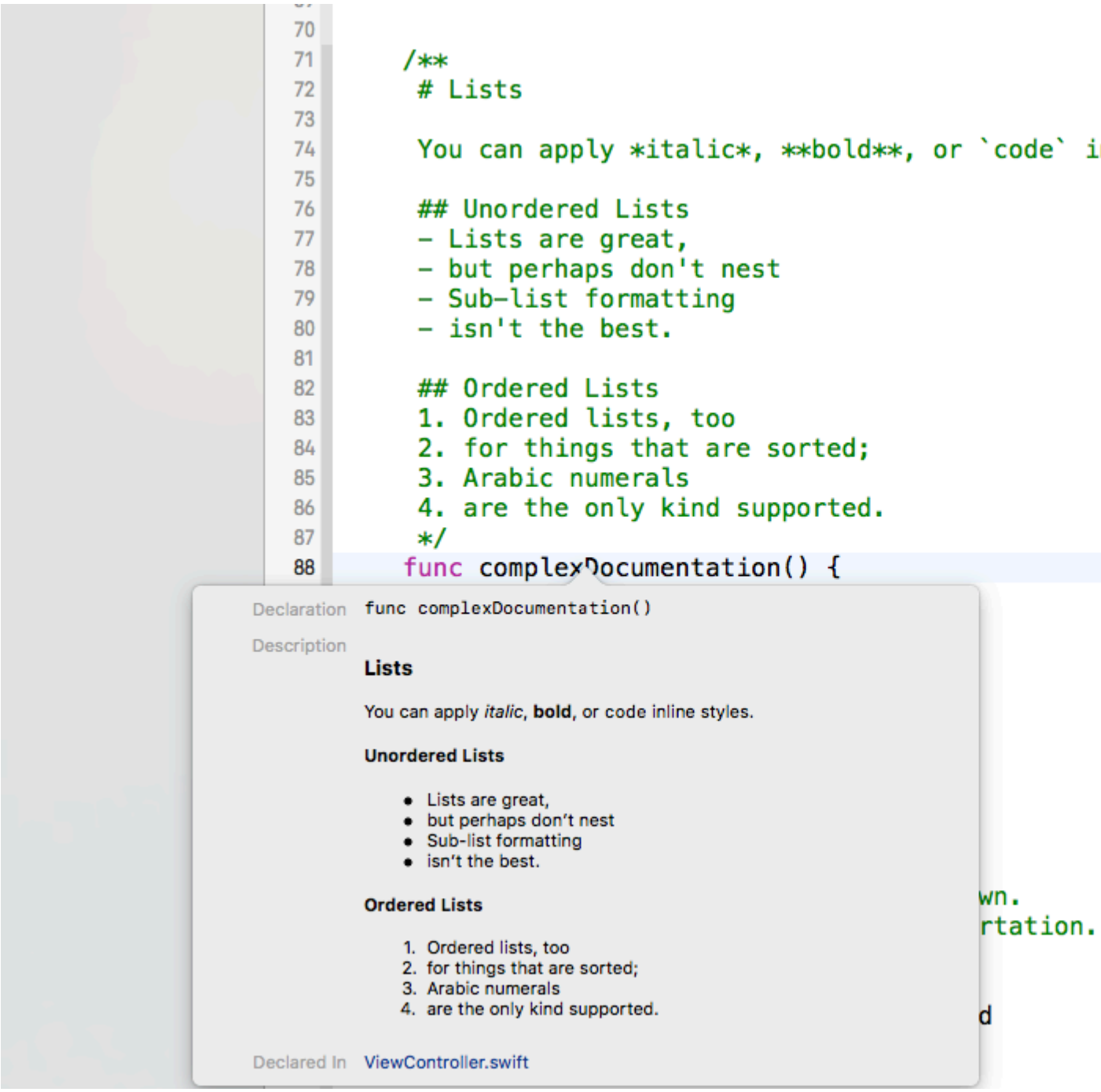



```
/**
 Adds user to the list of poople which are assigned the tasks.

 - Parameter name: The name to add
 - Returns: A boolean value (true/false) to tell if user is added successfully to the people
list.
*/
func addMeToList(name: String) -> Bool {

    // Do something....


    return true
}
```

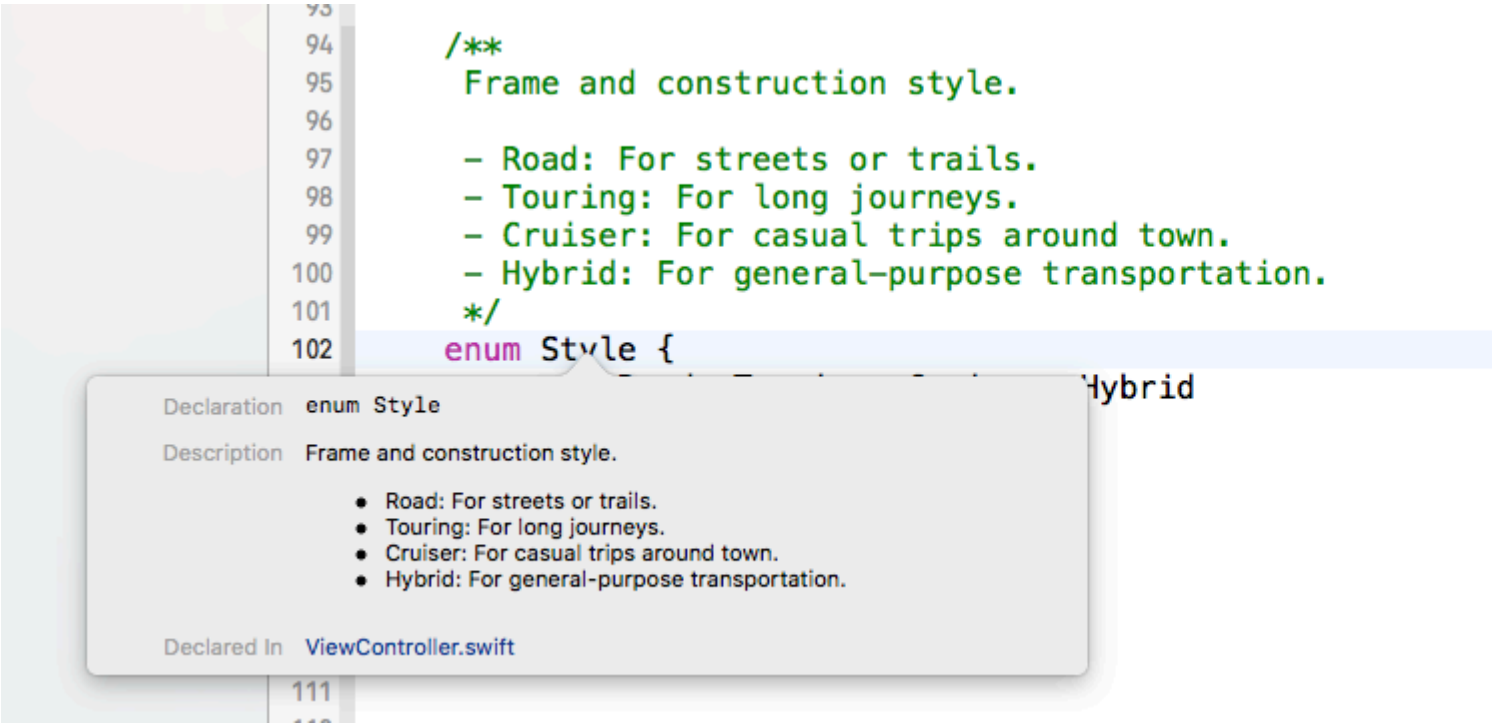

```
/**
 Adds user to the list of poople which are assigned the tasks.

 - Parameter name: The name to add
 - Returns: A boolean value (true/false) to tell if user is added successfully to the people
list.
*/
func addMeToList(name: String) -> Bool {

    // Do something....


    return true
}
```




```
/**
 Adds user to the list of poople which are assigned the tasks.

 - Parameter name: The name to add
 - Returns: A boolean value (true/false) to tell if user is added successfully to the people
list.
*/
func addMeToList(name: String) -> Bool {

    // Do something....


    return true
}
```




```
/**
 Adds user to the list of poople which are assigned the tasks.

 - Parameter name: The name to add
 - Returns: A boolean value (true/false) to tell if user is added successfully to the people
list.
*/
func addMeToList(name: String) -> Bool {

    // Do something....


    return true
}
```



https://riptutorial.com/zh-TW/swift/topic/6937/

38:

Swift/

- `NSObject`
- `dynamic`

SwiftCocoaObjective-C

`@objc`Swift APIObjective-C。 *SwiftObjective-C* 。 `dynamic`。 Objective-C`dynamic``@objc`。

。 **API`dynamic`** 。 Objective-C`method_exchangeImplementations`。 Swift 。

Objective-C

NSHipster

Examples

UIViewControllerSwizzling viewDidLoad

Objective-C。 。

Objective-CPure Swift`NSObject`。

`UIViewController`swizzle `viewDidLoad`

```
extension UIViewController {

    // We cannot override load like we could in Objective-C, so override initialize instead
    public override static func initialize() {

        // Make a static struct for our dispatch token so only one exists in memory
        struct Static {
            static var token: dispatch_once_t = 0
        }

        // Wrap this in a dispatch_once block so it is only run once
        dispatch_once(&Static.token) {
            // Get the original selectors and method implementations, and swap them with our
new method
            let originalSelector = #selector(UIViewController.viewDidLoad)
            let swizzledSelector = #selector(UIViewController.myViewDidLoad)

            let originalMethod = class_getInstanceMethod(self, originalSelector)
            let swizzledMethod = class_getInstanceMethod(self, swizzledSelector)

            let didAddMethod = class_addMethod(self, originalSelector,
method_getImplementation(swizzledMethod), method_getTypeEncoding(swizzledMethod))

            // class_addMethod can fail if used incorrectly or with invalid pointers, so check
to make sure we were able to add the method to the lookup table successfully
            if didAddMethod {
```

```
                class_replaceMethod(self, swizzledSelector,
method_getImplementation(originalMethod), method_getTypeEncoding(originalMethod))
            } else {
                method_exchangeImplementations(originalMethod, swizzledMethod);
            }
        }
    }

    // Our new viewDidLoad function
    // In this example, we are just logging the name of the function, but this can be used to
run any custom code
    func myViewDidLoad() {
        // This is not recursive since we swapped the Selectors in initialize().
        // We cannot call super in an extension.
        self.myViewDidLoad()
        print(#function) // logs myViewDidLoad()
    }
}
```

Swift Swizzling

`methodOne()methodTwo()TestSwizzling`

```
class TestSwizzling : NSObject {
    dynamic func methodOne()->Int{
        return 1
    }
}

extension TestSwizzling {

    //In Objective-C you'd perform the swizzling in load(),
    //but this method is not permitted in Swift
    override class func initialize()
    {

        struct Inner {
            static let i: () = {

                let originalSelector = #selector(TestSwizzling.methodOne)
                let swizzledSelector = #selector(TestSwizzling.methodTwo)
                let originalMethod = class_getInstanceMethod(TestSwizzling.self,
originalSelector);
                let swizzledMethod = class_getInstanceMethod(TestSwizzling.self,
swizzledSelector)
                method_exchangeImplementations(originalMethod, swizzledMethod)
            }
        }
        let _ = Inner.i
    }

    func methodTwo()->Int{
        // It will not be a recursive call anymore after the swizzling
        return methodTwo()+1
    }
}

var c = TestSwizzling()
print(c.methodOne())
```

```
print(c.methodTwo())
```

Swizzling - Objective-C

- UIView`initWithFrame:`

```
static IMP original_initWithFrame;

+ (void)swizzleMethods {
    static BOOL swizzled = NO;
    if (!swizzled) {
        swizzled = YES;

        Method initWithFrameMethod =
            class_getInstanceMethod([UIView class], @selector(initWithFrame:));
        original_initWithFrame = method_setImplementation(
            initWithFrameMethod, (IMP)replacement_initWithFrame);
    }
}

static id replacement_initWithFrame(id self, SEL _cmd, CGRect rect) {

    // This will be called instead of the original initWithFrame method on UIView
    // Do here whatever you need...

    // Bonus: This is how you would call the original initWithFrame method
    UIView *view =
        ((id (*)(id, SEL, CGRect))original_initWithFrame)(self, _cmd, rect);

    return view;
}
```

https://riptutorial.com/zh-TW/swift/topic/1436/

39:

。

Swift 。

Examples

```
enum Direction {
    case up
    case down
    case left
    case right
}

enum Direction { case up, down, left, right }
```

```
enum Direction {
    case up
    case down
    case left
    case right
}

enum Direction { case up, down, left, right }
```

/`switch`

```
enum Direction {
    case up
    case down
    case left
    case right
}

enum Direction { case up, down, left, right }
```

`Hashable` `Equatable`

```
enum Direction {
    case up
    case down
    case left
    case right
}

enum Direction { case up, down, left, right }
```

```
enum Action {
    case jump
    case kick
    case move(distance: Float)  // The "move" case has an associated distance
```

```
}
```

```
enum Action {
    case jump
    case kick
    case move(distance: Float)  // The "move" case has an associated distance
}
```

switch

```
enum Action {
    case jump
    case kick
    case move(distance: Float)  // The "move" case has an associated distance
}
```

if case

```
enum Action {
    case jump
    case kick
    case move(distance: Float)  // The "move" case has an associated distance
}
```

guard case

```
enum Action {
    case jump
    case kick
    case move(distance: Float)  // The "move" case has an associated distance
}
```

Equatable ◦ ==

```
enum Action {
    case jump
    case kick
    case move(distance: Float)  // The "move" case has an associated distance
}
```

```
enum Tree<T> {
    case leaf(T)
    case branch(Tree<T>, Tree<T>)  // error: recursive enum 'Tree<T>' is not marked 'indirect'
}
```

indirect◦

```
enum Tree<T> {
    case leaf(T)
    case branch(Tree<T>, Tree<T>)  // error: recursive enum 'Tree<T>' is not marked 'indirect'
}
```

indirect

```
enum Tree<T> {
    case leaf(T)
    case branch(Tree<T>, Tree<T>)  // error: recursive enum 'Tree<T>' is not marked 'indirect'
}
```

```
enum Rotation: Int {
    case up = 0
    case left = 90
    case upsideDown = 180
    case right = 270
}
```

rawValue

```
enum Rotation: Int {
    case up = 0
    case left = 90
    case upsideDown = 180
    case right = 270
}
```

0

```
enum Rotation: Int {
    case up = 0
    case left = 90
    case upsideDown = 180
    case right = 270
}
```

```
enum Rotation: Int {
    case up = 0
    case left = 90
    case upsideDown = 180
    case right = 270
}
```

RawRepresentable ◦ `.rawValue`

```
enum Rotation: Int {
    case up = 0
    case left = 90
    case upsideDown = 180
    case right = 270
}
```

`init?(rawValue:)`

```
enum Rotation: Int {
    case up = 0
    case left = 90
    case upsideDown = 180
    case right = 270
}
```

hashValue。

```
enum Rotation: Int {
    case up = 0
    case left = 90
    case upsideDown = 180
    case right = 270
}
```

init`init?(rawValue:)init?(rawValue:)` 。 。 。

```
enum CompassDirection {
    case north(Int)
    case south(Int)
    case east(Int)
    case west(Int)

    init?(degrees: Int) {
        switch degrees {
        case 0...45:
            self = .north(degrees)
        case 46...135:
            self = .east(degrees)
        case 136...225:
            self = .south(degrees)
        case 226...315:
            self = .west(degrees)
        case 316...360:
            self = .north(degrees)
        default:
            return nil
        }
    }

    var value: Int = {
        switch self {
            case north(let degrees):
                return degrees
            case south(let degrees):
                return degrees
            case east(let degrees):
                return degrees
            case west(let degrees):
                return degrees
        }
    }
}
```

```
enum CompassDirection {
    case north(Int)
    case south(Int)
    case east(Int)
    case west(Int)

    init?(degrees: Int) {
        switch degrees {
        case 0...45:
            self = .north(degrees)
```

```
        case 46...135:
            self = .east(degrees)
        case 136...225:
            self = .south(degrees)
        case 226...315:
            self = .west(degrees)
        case 316...360:
            self = .north(degrees)
        default:
            return nil
        }
    }

    var value: Int = {
        switch self {
            case north(let degrees):
                return degrees
            case south(let degrees):
                return degrees
            case east(let degrees):
                return degrees
            case west(let degrees):
                return degrees
        }
    }
}
```

SwiftC。 。

```
protocol ChangesDirection {
    mutating func changeDirection()
}

enum Direction {

    // enumeration cases
    case up, down, left, right

    // initialise the enum instance with a case
    // that's in the opposite direction to another
    init(oppositeTo otherDirection: Direction) {
        self = otherDirection.opposite
    }

    // computed property that returns the opposite direction
    var opposite: Direction {
        switch self {
        case .up:
            return .down
        case .down:
            return .up
        case .left:
            return .right
        case .right:
            return .left
        }
    }
}

// extension to Direction that adds conformance to the ChangesDirection protocol
```

```
extension Direction: ChangesDirection {
    mutating func changeDirection() {
        self = .left
    }
}
```

```
protocol ChangesDirection {
    mutating func changeDirection()
}

enum Direction {

    // enumeration cases
    case up, down, left, right

    // initialise the enum instance with a case
    // that's in the opposite direction to another
    init(oppositeTo otherDirection: Direction) {
        self = otherDirection.opposite
    }

    // computed property that returns the opposite direction
    var opposite: Direction {
        switch self {
        case .up:
            return .down
        case .down:
            return .up
        case .left:
            return .right
        case .right:
            return .left
        }
    }
}

// extension to Direction that adds conformance to the ChangesDirection protocol
extension Direction: ChangesDirection {
    mutating func changeDirection() {
        self = .left
    }
}
```

。

```
enum Orchestra {
    enum Strings {
        case violin
        case viola
        case cello
        case doubleBasse
    }

    enum Keyboards {
        case piano
        case celesta
        case harp
    }
```

```
    enum Woodwinds {
        case flute
        case oboe
        case clarinet
        case bassoon
        case contrabassoon
    }
}
```

```
enum Orchestra {
    enum Strings {
        case violin
        case viola
        case cello
        case doubleBasse
    }

    enum Keyboards {
        case piano
        case celesta
        case harp
    }

    enum Woodwinds {
        case flute
        case oboe
        case clarinet
        case bassoon
        case contrabassoon
    }
}
```

https://riptutorial.com/zh-TW/swift/topic/224/

40:

ifelse ifelseSwiftTrueFalse。 Swift。

Swift 。

Examples

Guard

2.0

Guardfalse。 `return breakcontinue`;。 `guard ifif`。

。

```
func printNum(num: Int) {
    guard num == 10 else {
        print("num is not 10")
        return
    }
    print("num is 10")
}
```

Guard

```
func printNum(num: Int) {
    guard num == 10 else {
        print("num is not 10")
        return
    }
    print("num is 10")
}
```

Guard`where`

```
func printNum(num: Int) {
    guard num == 10 else {
        print("num is not 10")
        return
    }
    print("num is 10")
}
```

if

`if`Bool`true`

```
let num = 10
```

```
if num == 10 {
    // Code inside this block only executes if the condition was true.
    print("num is 10")
}

let condition = num == 10   // condition's type is Bool
if condition {
    print("num is 10")
}
```

`if`**`else if`**`else`

```
let num = 10

if num == 10 {
    // Code inside this block only executes if the condition was true.
    print("num is 10")
}

let condition = num == 10   // condition's type is Bool
if condition {
    print("num is 10")
}
```

`&&` `||`

AND

```
let num = 10

if num == 10 {
    // Code inside this block only executes if the condition was true.
    print("num is 10")
}

let condition = num == 10   // condition's type is Bool
if condition {
    print("num is 10")
}
```

`num == 10`false。 。

OR

```
let num = 10

if num == 10 {
    // Code inside this block only executes if the condition was true.
    print("num is 10")
}

let condition = num == 10   // condition's type is Bool
if condition {
    print("num is 10")
```

```
}
```

`num == 10`true。

NOT

```
let num = 10

if num == 10 {
    // Code inside this block only executes if the condition was true.
    print("num is 10")
}

let condition = num == 10   // condition's type is Bool
if condition {
    print("num is 10")
}
```

“where”

- `if let` **`nil`**

```
let num: Int? = 10 // or: let num: Int? = nil

if let unwrappedNum = num {
    // num has type Int?; unwrappedNum has type Int
    print("num was not nil: \(unwrappedNum + 1)")
} else {
    print("num was nil")
}
```

```
let num: Int? = 10 // or: let num: Int? = nil

if let unwrappedNum = num {
    // num has type Int?; unwrappedNum has type Int
    print("num was not nil: \(unwrappedNum + 1)")
} else {
    print("num was nil")
}
```

1.2 3.0

,

```
let num: Int? = 10 // or: let num: Int? = nil

if let unwrappedNum = num {
    // num has type Int?; unwrappedNum has type Int
    print("num was not nil: \(unwrappedNum + 1)")
} else {
    print("num was nil")
}
```

where

```
let num: Int? = 10 // or: let num: Int? = nil

if let unwrappedNum = num {
    // num has type Int?; unwrappedNum has type Int
    print("num was not nil: \(unwrappedNum + 1)")
} else {
    print("num was nil")
}
```

where

```
let num: Int? = 10 // or: let num: Int? = nil

if let unwrappedNum = num {
    // num has type Int?; unwrappedNum has type Int
    print("num was not nil: \(unwrappedNum + 1)")
} else {
    print("num was nil")
}
```

3.0

3 `where` SE-0099 ,。

```
let num: Int? = 10 // or: let num: Int? = nil

if let unwrappedNum = num {
    // num has type Int?; unwrappedNum has type Int
    print("num was not nil: \(unwrappedNum + 1)")
} else {
    print("num was nil")
}
```

if

```
let a = 5
let b = 10
let min: Int

if a < b {
    min = a
} else {
    min = b
}

let max: Int

if a > b {
    max = a
} else {
    max = b
}
```

truefalse。

```
let a = 5
let b = 10
let min: Int

if a < b {
    min = a
} else {
    min = b
}

let max: Int

if a > b {
    max = a
} else {
    max = b
}
```

```
let a = 5
let b = 10
let min: Int

if a < b {
    min = a
} else {
    min = b
}

let max: Int

if a > b {
    max = a
} else {
    max = b
}
```

a <b。 mina;b。

Swiftmaxmin。

Nil-Coalescing

<OPTIONAL> ?? <DEFAULT VALUE><OPTIONAL><OPTIONAL>n<DEFAULT VALUE> 。 <OPTIONAL>。 <DEFAULT VALUE> <OPTIONAL>。

nil-coalescing

```
a != nil ? a! : b
```

```
a != nil ? a! : b
```

```
a != nil ? a! : b
```

https://riptutorial.com/zh-TW/swift/topic/475/

41:

。 。

SwiftSwift。 SwiftArrayDictionary。 IntStringSwift。 。

AppleSwift

Examples

Equatable

```
class MyGenericClass<Type: Equatable>{

    var value: Type
    init(value: Type){
        self.value = value
    }

    func getValue() -> Type{
        return self.value
    }

    func valueEquals(anotherValue: Type) -> Bool{
        return self.value == anotherValue
    }
}
```

MyGenericClass typeEquatable==type

```
class MyGenericClass<Type: Equatable>{

    var value: Type
    init(value: Type){
        self.value = value
    }

    func getValue() -> Type{
        return self.value
    }

    func valueEquals(anotherValue: Type) -> Bool{
        return self.value == anotherValue
    }
}
```

。 Any。

<>。

```
/// Picks one of the inputs at random, and returns it
func pickRandom<T>(_ a:T, _ b:T) -> T {
    return arc4random_uniform(2) == 0 ? a : b
```

```
}
```

TSwiftT。

```
/// Picks one of the inputs at random, and returns it
func pickRandom<T>(_ a:T, _ b:T) -> T {
    return arc4random_uniform(2) == 0 ? a : b
}
```

。 SwiftT == Int - (Int, Int) -> Int 。

- 。

```
/// Picks one of the inputs at random, and returns it
func pickRandom<T>(_ a:T, _ b:T) -> T {
    return arc4random_uniform(2) == 0 ? a : b
}
```

。

```
/// Picks one of the inputs at random, and returns it
func pickRandom<T>(_ a:T, _ b:T) -> T {
    return arc4random_uniform(2) == 0 ? a : b
}
```

Bar。 init(baz:T)。

```
/// Picks one of the inputs at random, and returns it
func pickRandom<T>(_ a:T, _ b:T) -> T {
    return arc4random_uniform(2) == 0 ? a : b
}
```

TStringBar<String>。

```
/// Picks one of the inputs at random, and returns it
func pickRandom<T>(_ a:T, _ b:T) -> T {
    return arc4random_uniform(2) == 0 ? a : b
}
```

。 baz String。

```
/// Picks one of the inputs at random, and returns it
func pickRandom<T>(_ a:T, _ b:T) -> T {
    return arc4random_uniform(2) == 0 ? a : b
}
```

。

```
/// Picks one of the inputs at random, and returns it
func pickRandom<T>(_ a:T, _ b:T) -> T {
```

```
    return arc4random_uniform(2) == 0 ? a : b
}
```

Bar<Int> ◦ IntBar◦

◦ ◦ SwiftArrayElementArray◦

```
/// Picks one of the inputs at random, and returns it
func pickRandom<T>(_ a:T, _ b:T) -> T {
    return arc4random_uniform(2) == 0 ? a : b
}
```

Type

```
class MyGenericClass<Type>{

    var value: Type
    init(value: Type){
        self.value = value
    }

    func getValue() -> Type{
        return self.value
    }

    func setValue(value: Type){
        self.value = value
    }
}
```

```
class MyGenericClass<Type>{

    var value: Type
    init(value: Type){
        self.value = value
    }

    func getValue() -> Type{
        return self.value
    }

    func setValue(value: Type){
        self.value = value
    }
}
```

```
class MyGenericClass<Type>{

    var value: Type
    init(value: Type){
        self.value = value
    }

    func getValue() -> Type{
        return self.value
    }
```

```
    func setValue(value: Type){
        self.value = value
    }
}
```

```
class MyGenericClass<Type>{

    var value: Type
    init(value: Type){
        self.value = value
    }

    func getValue() -> Type{
        return self.value
    }

    func setValue(value: Type){
        self.value = value
    }
}
```

```
// Models
class MyFirstModel {
}

class MySecondModel: MyFirstModel {
}

// Generic classes
class MyFirstGenericClass<T: MyFirstModel> {

    func doSomethingWithModel(model: T) {
        // Do something here
    }

}

class MySecondGenericClass<T: MySecondModel>: MyFirstGenericClass<T> {

    override func doSomethingWithModel(model: T) {
        super.doSomethingWithModel(model)

        // Do more things here
    }

}
```

。

```
// Need to restrict the extension to elements that can be compared.
// The `Element` is the generics name defined by Array for its item types.
// This restriction also gives us access to `index(of:_)` which is also
// defined in an Array extension with `where Element: Equatable`.
public extension Array where Element: Equatable {
    /// Removes the given object from the array.
    mutating func remove(_ element: Element) {
        if let index = self.index(of: element ) {
```

```
            self.remove(at: index)
        } else {
            fatalError("Removal error, no such element:\"\(element)\" in array.\n")
        }
    }
}
```

```
// Need to restrict the extension to elements that can be compared.
// The `Element` is the generics name defined by Array for its item types.
// This restriction also gives us access to `index(of:_)` which is also
// defined in an Array extension with `where Element: Equatable`.
public extension Array where Element: Equatable {
    /// Removes the given object from the array.
    mutating func remove(_ element: Element) {
        if let index = self.index(of: element ) {
            self.remove(at: index)
        } else {
            fatalError("Removal error, no such element:\"\(element)\" in array.\n")
        }
    }
}
```

。 。


```
// Need to restrict the extension to elements that can be compared.
// The `Element` is the generics name defined by Array for its item types.
// This restriction also gives us access to `index(of:_)` which is also
// defined in an Array extension with `where Element: Equatable`.
public extension Array where Element: Equatable {
    /// Removes the given object from the array.
    mutating func remove(_ element: Element) {
        if let index = self.index(of: element ) {
            self.remove(at: index)
        } else {
            fatalError("Removal error, no such element:\"\(element)\" in array.\n")
        }
    }
}
```


```
protocol JSONDecodable {
    static func from(_ json: [String: Any]) -> Any?
}
```

。


```
protocol JSONDecodable {
    static func from(_ json: [String: Any]) -> Any?
}
```


TestObject Any。


```
protocol JSONDecodable {
    static func from(_ json: [String: Any]) -> Any?
}
```

where

```
func doSomething<T where T: Comparable, T: Hashable>(first: T, second: T) {
    // Access hashable function
    guard first.hashValue == second.hashValue else {
        return
    }
    // Access comparable function
    if first == second {
        print("\(first) and \(second) are equal.")
    }
}
```

where

```
func doSomething<T where T: Comparable, T: Hashable>(first: T, second: T) {
    // Access hashable function
    guard first.hashValue == second.hashValue else {
        return
    }
    // Access comparable function
    if first == second {
        print("\(first) and \(second) are equal.")
    }
}
```

。

```
func doSomething<T where T: Comparable, T: Hashable>(first: T, second: T) {
    // Access hashable function
    guard first.hashValue == second.hashValue else {
        return
    }
    // Access comparable function
    if first == second {
        print("\(first) and \(second) are equal.")
    }
}
```

https://riptutorial.com/zh-TW/swift/topic/774/

42: Swift

`println``debugPrintln`Swift 2.0。

https://developer.apple.com/library/content/technotes/tn2347/_index.html
http://ericasadun.com/2015/05/22/swift-logging/

http://www.dotnetperls.com/print-swift

Examples

Debug Print。

```
print("Hello")
debugPrint("Hello")

let dict = ["foo": 1, "bar": 2]

print(dict)
debugPrint(dict)
```

```
print("Hello")
debugPrint("Hello")

let dict = ["foo": 1, "bar": 2]

print(dict)
debugPrint(dict)
```

```
print("Hello")
debugPrint("Hello")

let dict = ["foo": 1, "bar": 2]

print(dict)
debugPrint(dict)
```

```
print("Hello")
debugPrint("Hello")

let dict = ["foo": 1, "bar": 2]

print(dict)
debugPrint(dict)
```

43。 `debugPrint`

```
print("Hello")
debugPrint("Hello")

let dict = ["foo": 1, "bar": 2]

print(dict)
debugPrint(dict)
```

dump◦

```
let names = ["Joe", "Jane", "Jim", "Joyce"]
dump(names)
```

▿4
- [0]
- [1]
- [2]
- [3]

```
let names = ["Joe", "Jane", "Jim", "Joyce"]
dump(names)
```

▿3/
▿[0] :(2
- .0
- .133
▿[1] :(2
- .0baz
- .142
▿[2] :(2
- .0foo
- .110

dumpdump(_:name:indent:maxDepth:maxItems:) ◦

◦

name

```
let names = ["Joe", "Jane", "Jim", "Joyce"]
dump(names)
```

▿3/
▿[0] :(2
- .0
- .133
▿[1] :(2
- .0baz

- .142
▿[2] :(2
- .0foo
- .110

maxItems:maxDepth:indent: ◦

printvs dump

print()◦

```
class Abc {
    let a = "aa"
    let b = "bb"
}
```

Abc

```
class Abc {
    let a = "aa"
    let b = "bb"
}
```

print()

```
class Abc {
    let a = "aa"
    let b = "bb"
}
```

dump()

```
class Abc {
    let a = "aa"
    let b = "bb"
}
```

dump()print()◦

dump()UI

```
class Abc {
    let a = "aa"
    let b = "bb"
}
```

dump(view)

```
class Abc {
    let a = "aa"
    let b = "bb"
}
```

`print(view)`

```
class Abc {
    let a = "aa"
    let b = "bb"
}
```

`dump()`。

vs NSLog

swift`print()` `NSLog()` Xcode。

`print()` `NSLog()`

1 TimeStamp `NSLog()` `print()`。

```
let array = [1, 2, 3, 4, 5]
print(array)
NSLog(array.description)
```

> [1,2,3,4,5]
> 2017-05-31 131438.582 ProjetName [22867473287] [1,2,3,4,5]

ProjectName。

2 Only String `NSLog()` String`print()`。

```
let array = [1, 2, 3, 4, 5]
print(array)
NSLog(array.description)
```

3`print()` `NSLog()` 。

4 `NSLog()`。 `print()`。

5 `NSLog()` Xcode。 `print()`。

Swift https://riptutorial.com/zh-TW/swift/topic/3966/swift

43:

Examples

。

Swift。

“”。 “”。

“”

objective-c“objc_getAssociatedObject”_set。

```
get {
   return objc_getAssociatedObject(self, & _Handle) as! YourType
   }
set {
   objc_setAssociatedObject(self, & _Handle, newValue, .OBJC_ASSOCIATION_RETAIN)
    }
```

。

1. “class”。
2. “where selfUIViewController”。

“p”

```
get {
   return objc_getAssociatedObject(self, & _Handle) as! YourType
   }
set {
   objc_setAssociatedObject(self, & _Handle, newValue, .OBJC_ASSOCIATION_RETAIN)
    }
```

“”“p”

“p”。

```
get {
   return objc_getAssociatedObject(self, & _Handle) as! YourType
   }
set {
   objc_setAssociatedObject(self, & _Handle, newValue, .OBJC_ASSOCIATION_RETAIN)
    }
```

。 。

Xcode “p”。

“p”viewDidLoad。

p 。 p。 p“”“p”。 Xcode“p”。

“p”。 viewDidLoad。

。

“p”getter 。

```
get {
   return objc_getAssociatedObject(self, & _Handle) as! YourType
   }
set {
   objc_setAssociatedObject(self, & _Handle, newValue, .OBJC_ASSOCIATION_RETAIN)
    }
```

。 Xcode p。 pviewDidLoad。

.....

```
get {
   return objc_getAssociatedObject(self, & _Handle) as! YourType
   }
set {
   objc_setAssociatedObject(self, & _Handle, newValue, .OBJC_ASSOCIATION_RETAIN)
    }
```

。 。

```
get {
   return objc_getAssociatedObject(self, & _Handle) as! YourType
   }
set {
   objc_setAssociatedObject(self, & _Handle, newValue, .OBJC_ASSOCIATION_RETAIN)
    }
```

_aoGetPP“”0。

https://riptutorial.com/zh-TW/swift/topic/1085/

44:

Examples

```
struct Repository {
    let identifier: Int
    let name: String
    var description: String?
}
```

`Repositoryidentifier namedescription` ◦ `identifiernamelet` -keyword◦ ◦ ◦ ◦

◦ ◦

`Repositorydescription nil` ◦

```
struct Repository {
    let identifier: Int
    let name: String
    var description: String?
}
```

```
first = "Hello"
second = first
first += " World!"
// first == "Hello World!"
// second == "Hello"
```

String◦

```
first = "Hello"
second = first
first += " World!"
// first == "Hello World!"
// second == "Hello"
```

◦

◦

structstruct`mutating`

```
struct Counter {
    private var value = 0

    mutating func next() {
        value += 1
    }
}
```

`mutating`struct。

```
struct Counter {
    private var value = 0

    mutating func next() {
        value += 1
    }
}
```

`mutating`struct

```
struct Counter {
    private var value = 0

    mutating func next() {
        value += 1
    }
}
```

```
class MyView: NSView { }  // works

struct MyInt: Int { } // error: inheritance from non-protocol type 'Int'
```


```
class MyView: NSView { }  // works

struct MyInt: Int { } // error: inheritance from non-protocol type 'Int'
```


struct

Swift“”。

```
struct DeliveryRange {
  var range: Double
  let center: Location
}
let storeLocation = Location(latitude: 44.9871,
                             longitude: -93.2758)
var pizzaRange = DeliveryRange(range: 200,
                               center: storeLocation)
```


```
struct DeliveryRange {
  var range: Double
  let center: Location
}
let storeLocation = Location(latitude: 44.9871,
                             longitude: -93.2758)
var pizzaRange = DeliveryRange(range: 200,
                               center: storeLocation)
```


```
struct DeliveryRange {
  var range: Double
```

```
  let center: Location
}
let storeLocation = Location(latitude: 44.9871,
                             longitude: -93.2758)
var pizzaRange = DeliveryRange(range: 200,
                               center: storeLocation)
```

。

```
struct DeliveryRange {
  var range: Double
  let center: Location
}
let storeLocation = Location(latitude: 44.9871,
                             longitude: -93.2758)
var pizzaRange = DeliveryRange(range: 200,
                               center: storeLocation)
```

https://riptutorial.com/zh-TW/swift/topic/255/

45:

URLSessionFileManager

Examples

```
let url = "https://path-to-media"
let request = URLRequest(url: url)
let downloadTask = URLSession.shared.downloadTask(with: request) { (location, response, error)
in
  guard let location = location,
        let response = response,
        let documentsPath = NSSearchPathForDirectoriesInDomains(.documentDirectory,
.userDomainMask, true).first else {
    return
  }
  let documentsDirectoryUrl = URL(fileURLWithPath: documentsPath)
  let documentUrl = documentsDirectoryUrl.appendingPathComponent(response.suggestedFilename)
  let _ = try? FileManager.default.moveItem(at: location, to: documentUrl)

  // documentUrl is the local URL which we just downloaded and saved to the FileManager
}.resume()
```

```
let url = "https://path-to-media"
guard let documentsUrl = FileManager.default.urls(for: .documentDirectory, in:
.userDomainMask).first,
      let searchQuery = url.absoluteString.components(separatedBy: "/").last else {
  return nil
}

do {
  let directoryContents = try FileManager.default.contentsOfDirectory(at: documentsUrl,
includingPropertiesForKeys: nil, options: [])
  let cachedFiles = directoryContents.filter { $0.absoluteString.contains(searchQuery) }

  // do something with the files found by the url
} catch {
  // Could not find any files
}
```

https://riptutorial.com/zh-TW/swift/topic/8902/

46:

-
- car = Car“Ford”“Escape”//
- public enum
- private func calculateMarketCap
- setupView
- privatesetvar area = 0

1.

。 。

。 。

。

2.

。

3. ***GetterSetter***

propertysetterprivategetterinternal。 gettersetter。

4.

InitializersProtocolsExtensionsGenericsType Aliases

Examples

Struct

3.0

Swift 3。 `open`

```
public struct Car {

    public let make: String
    let model: String //Optional keyword: will automatically be "internal"
    private let fullName: String
    fileprivate var otherName: String

    public init(_ make: String, model: String) {
        self.make = make
        self.model = model
        self.fullName = "\(make)\(model)"
        self.otherName = "\(model) - \(make)"
    }
```

```
}
```

myCar

```
public struct Car {

    public let make: String
    let model: String //Optional keyword: will automatically be "internal"
    private let fullName: String
    fileprivate var otherName: String

    public init(_ make: String, model: String) {
        self.make = make
        self.model = model
        self.fullName = "\(make)\(model)"
        self.otherName = "\(model) - \(make)"
    }
}
```

Car.make

```
public struct Car {

    public let make: String
    let model: String //Optional keyword: will automatically be "internal"
    private let fullName: String
    fileprivate var otherName: String

    public init(_ make: String, model: String) {
        self.make = make
        self.model = model
        self.fullName = "\(make)\(model)"
        self.otherName = "\(model) - \(make)"
    }
}
```

Car。

Car.model

```
public struct Car {

    public let make: String
    let model: String //Optional keyword: will automatically be "internal"
    private let fullName: String
    fileprivate var otherName: String

    public init(_ make: String, model: String) {
        self.make = make
        self.model = model
        self.fullName = "\(make)\(model)"
        self.otherName = "\(model) - \(make)"
    }
}
```

`Car`。

Car.otherNamefileprivate

```
public struct Car {

    public let make: String
    let model: String //Optional keyword: will automatically be "internal"
    private let fullName: String
    fileprivate var otherName: String

    public init(_ make: String, model: String) {
        self.make = make
        self.model = model
        self.fullName = "\(make)\(model)"
        self.otherName = "\(model) - \(make)"
    }
}
```

`Car` 。

Car.fullName

```
public struct Car {

    public let make: String
    let model: String //Optional keyword: will automatically be "internal"
    private let fullName: String
    fileprivate var otherName: String

    public init(_ make: String, model: String) {
        self.make = make
        self.model = model
        self.fullName = "\(make)\(model)"
        self.otherName = "\(model) - \(make)"
    }
}
```

Swift 3。 `privatestruct` / `class`。

```
public struct Car {

    public let make: String
    let model: String //Optional keyword: will automatically be "internal"
    private let fullName: String
    fileprivate var otherName: String

    public init(_ make: String, model: String) {
        self.make = make
        self.model = model
        self.fullName = "\(make)\(model)"
        self.otherName = "\(model) - \(make)"
    }
}
```

Swift。 。

```
public class SuperClass {
    private func secretMethod() {}
}

internal class SubClass: SuperClass {
    override internal func secretMethod() {
        super.secretMethod()
    }
}
```

GettersSetters

```
struct Square {
    private(set) var area = 0

    var side: Int = 0 {
        didSet {
            area = side*side
        }
    }
}

public struct Square {
    public private(set) var area = 0
    public var side: Int = 0 {
        didSet {
            area = side*side
        }
    }
    public init() {}
}
```

https://riptutorial.com/zh-TW/swift/topic/1075/

47: -

。 。

。

Examples

。

Foostatic。 static。

```
public class Foo
{
    static let shared = Foo()

    // Used for preventing the class from being instantiated directly
    private init() {}

    func doSomething()
    {
        print("Do something")
    }
}
```

```
public class Foo
{
    static let shared = Foo()

    // Used for preventing the class from being instantiated directly
    private init() {}

    func doSomething()
    {
        print("Do something")
    }
}
```

private

- Swiftinit。 KrakenDev

  。

```
protocol SenderProtocol
{
    func send(package: AnyObject)
}

class Fedex: SenderProtocol
{
    func send(package: AnyObject)
    {
```

```
        print("Fedex deliver")
    }
}

class RegularPriorityMail: SenderProtocol
{
    func send(package: AnyObject)
    {
        print("Regular Priority Mail deliver")
    }
}

// This is our Factory
class DeliverFactory
{
    // It will be responsable for returning the proper instance that will handle the task
    static func makeSender(isLate isLate: Bool) -> SenderProtocol
    {
        return isLate ? Fedex() : RegularPriorityMail()
    }
}

// Usage:
let package = ["Item 1", "Item 2"]

// Fedex class will handle the delivery
DeliverFactory.makeSender(isLate:true).send(package)

// Regular Priority Mail class will handle the delivery
DeliverFactory.makeSender(isLate:false).send(package)
```

sender()。

send()。 。

。

。 。 Observer - - MVC。

。

Notification Center

```
let notifCentre = NotificationCenter.default
```

。 ...

```
let notifCentre = NotificationCenter.default
```

“readForMyFunc”。 myFunc。

```
let notifCentre = NotificationCenter.default
```

。

```
let notifCentre = NotificationCenter.default
```

```
let notifCentre = NotificationCenter.default
```

```
let notifCentre = NotificationCenter.default
```

commandprocessing。 processing;processing。 processing。

。

processing。

```
protocol PurchasePower {
var allowable : Float { get }
  var role : String { get }
  var successor : PurchasePower? { get set }
}

extension PurchasePower {
  func process(request : PurchaseRequest){
    if request.amount < self.allowable {
      print(self.role + " will approve $ \(request.amount) for \(request.purpose)")
    } else if successor != nil {
      successor?.process(request: request)
    }
  }
}
```

command。

```
protocol PurchasePower {
var allowable : Float { get }
  var role : String { get }
  var successor : PurchasePower? { get set }
}

extension PurchasePower {
  func process(request : PurchaseRequest){
    if request.amount < self.allowable {
      print(self.role + " will approve $ \(request.amount) for \(request.purpose)")
    } else if successor != nil {
      successor?.process(request: request)
    }
  }
}
```

。

```
protocol PurchasePower {
var allowable : Float { get }
  var role : String { get }
  var successor : PurchasePower? { get set }
}
```

```
extension PurchasePower {
  func process(request : PurchaseRequest){
    if request.amount < self.allowable {
      print(self.role + " will approve $ \(request.amount) for \(request.purpose)")
    } else if successor != nil {
      successor?.process(request: request)
    }
  }
}
```

```
protocol PurchasePower {
var allowable : Float { get }
  var role : String { get }
  var successor : PurchasePower? { get set }
}

extension PurchasePower {
  func process(request : PurchaseRequest){
    if request.amount < self.allowable {
      print(self.role + " will approve $ \(request.amount) for \(request.purpose)")
    } else if successor != nil {
      successor?.process(request: request)
    }
  }
}
```

```
protocol PurchasePower {
var allowable : Float { get }
  var role : String { get }
  var successor : PurchasePower? { get set }
}

extension PurchasePower {
  func process(request : PurchaseRequest){
    if request.amount < self.allowable {
      print(self.role + " will approve $ \(request.amount) for \(request.purpose)")
    } else if successor != nil {
      successor?.process(request: request)
    }
  }
}
```

-

```
struct Turtle {
  let name: String
}

struct Turtles {
  let turtles: [Turtle]
}

struct TurtlesIterator: IteratorProtocol {
  private var current = 0
  private let turtles: [Turtle]

  init(turtles: [Turtle]) {
    self.turtles = turtles
```

```
  }

  mutating func next() -> Turtle? {
    defer { current += 1 }
    return turtles.count > current ? turtles[current] : nil
  }
}

extension Turtles: Sequence {
  func makeIterator() -> TurtlesIterator {
    return TurtlesIterator(turtles: turtles)
  }
}
```

```
struct Turtle {
  let name: String
}

struct Turtles {
  let turtles: [Turtle]
}

struct TurtlesIterator: IteratorProtocol {
  private var current = 0
  private let turtles: [Turtle]

  init(turtles: [Turtle]) {
    self.turtles = turtles
  }

  mutating func next() -> Turtle? {
    defer { current += 1 }
    return turtles.count > current ? turtles[current] : nil
  }
}

extension Turtles: Sequence {
  func makeIterator() -> TurtlesIterator {
    return TurtlesIterator(turtles: turtles)
  }
}
```

。 。 。 。

-Wikipedia

。 。

Car Builder。

Car

- 。
- 。
- 。
- 。
-

    -
-
-

```
import UIKit

enum CarType {
    case

    sportage,
    saloon
}

enum GearType {
    case

    manual,
    automatic
}

struct Motor {
    var id: String
    var name: String
    var model: String
    var numberOfCylinders: UInt8
}

class Car: CustomStringConvertible {
    var color: UIColor
    var numberOfSeats: UInt8
    var numberOfWheels: UInt8
    var type: CarType
    var gearType: GearType
    var motor: Motor
    var shouldHasAirbags: Bool

    var description: String {
        return "color: \(color)\nNumber of seats: \(numberOfSeats)\nNumber of Wheels:
\(numberOfWheels)\n Type: \(gearType)\nMotor: \(motor)\nAirbag Availability:
\(shouldHasAirbags)"
    }

    init(color: UIColor, numberOfSeats: UInt8, numberOfWheels: UInt8, type: CarType, gearType:
GearType, motor: Motor, shouldHasAirbags: Bool) {

        self.color = color
        self.numberOfSeats = numberOfSeats
        self.numberOfWheels = numberOfWheels
        self.type = type
        self.gearType = gearType
        self.motor = motor
        self.shouldHasAirbags = shouldHasAirbags

    }
}
```

```
import UIKit

enum CarType {
```

```
    case

    sportage,
    saloon
}

enum GearType {
    case

    manual,
    automatic
}

struct Motor {
    var id: String
    var name: String
    var model: String
    var numberOfCylinders: UInt8
}

class Car: CustomStringConvertible {
    var color: UIColor
    var numberOfSeats: UInt8
    var numberOfWheels: UInt8
    var type: CarType
    var gearType: GearType
    var motor: Motor
    var shouldHasAirbags: Bool

    var description: String {
        return "color: \(color)\nNumber of seats: \(numberOfSeats)\nNumber of Wheels:
\(numberOfWheels)\n Type: \(gearType)\nMotor: \(motor)\nAirbag Availability:
\(shouldHasAirbags)"
    }

    init(color: UIColor, numberOfSeats: UInt8, numberOfWheels: UInt8, type: CarType, gearType:
GearType, motor: Motor, shouldHasAirbags: Bool) {

        self.color = color
        self.numberOfSeats = numberOfSeats
        self.numberOfWheels = numberOfWheels
        self.type = type
        self.gearType = gearType
        self.motor = motor
        self.shouldHasAirbags = shouldHasAirbags

    }
}
```

。

。

CarBuilder

```
import UIKit

enum CarType {
    case
```

```
    sportage,
    saloon
}

enum GearType {
    case

    manual,
    automatic
}

struct Motor {
    var id: String
    var name: String
    var model: String
    var numberOfCylinders: UInt8
}

class Car: CustomStringConvertible {
    var color: UIColor
    var numberOfSeats: UInt8
    var numberOfWheels: UInt8
    var type: CarType
    var gearType: GearType
    var motor: Motor
    var shouldHasAirbags: Bool

    var description: String {
        return "color: \(color)\nNumber of seats: \(numberOfSeats)\nNumber of Wheels:
\(numberOfWheels)\n Type: \(gearType)\nMotor: \(motor)\nAirbag Availability:
\(shouldHasAirbags)"
    }

    init(color: UIColor, numberOfSeats: UInt8, numberOfWheels: UInt8, type: CarType, gearType:
GearType, motor: Motor, shouldHasAirbags: Bool) {

        self.color = color
        self.numberOfSeats = numberOfSeats
        self.numberOfWheels = numberOfWheels
        self.type = type
        self.gearType = gearType
        self.motor = motor
        self.shouldHasAirbags = shouldHasAirbags

    }
}
```

`CarBuilder`car。

`CarBuilder`

```
import UIKit

enum CarType {
    case

    sportage,
    saloon
}
```

```
enum GearType {
    case

    manual,
    automatic
}

struct Motor {
    var id: String
    var name: String
    var model: String
    var numberOfCylinders: UInt8
}

class Car: CustomStringConvertible {
    var color: UIColor
    var numberOfSeats: UInt8
    var numberOfWheels: UInt8
    var type: CarType
    var gearType: GearType
    var motor: Motor
    var shouldHasAirbags: Bool

    var description: String {
        return "color: \(color)\nNumber of seats: \(numberOfSeats)\nNumber of Wheels:
\(numberOfWheels)\n Type: \(gearType)\nMotor: \(motor)\nAirbag Availability:
\(shouldHasAirbags)"
    }

    init(color: UIColor, numberOfSeats: UInt8, numberOfWheels: UInt8, type: CarType, gearType:
GearType, motor: Motor, shouldHasAirbags: Bool) {

        self.color = color
        self.numberOfSeats = numberOfSeats
        self.numberOfWheels = numberOfWheels
        self.type = type
        self.gearType = gearType
        self.motor = motor
        self.shouldHasAirbags = shouldHasAirbags

    }
}
```

Builder Pattern。

。

CarBluePrint

```
import UIKit

enum CarType {
    case

    sportage,
    saloon
}
```

```
enum GearType {
    case

    manual,
    automatic
}

struct Motor {
    var id: String
    var name: String
    var model: String
    var numberOfCylinders: UInt8
}

class Car: CustomStringConvertible {
    var color: UIColor
    var numberOfSeats: UInt8
    var numberOfWheels: UInt8
    var type: CarType
    var gearType: GearType
    var motor: Motor
    var shouldHasAirbags: Bool

    var description: String {
        return "color: \(color)\nNumber of seats: \(numberOfSeats)\nNumber of Wheels:
\(numberOfWheels)\n Type: \(gearType)\nMotor: \(motor)\nAirbag Availability:
\(shouldHasAirbags)"
    }

    init(color: UIColor, numberOfSeats: UInt8, numberOfWheels: UInt8, type: CarType, gearType:
GearType, motor: Motor, shouldHasAirbags: Bool) {

        self.color = color
        self.numberOfSeats = numberOfSeats
        self.numberOfWheels = numberOfWheels
        self.type = type
        self.gearType = gearType
        self.motor = motor
        self.shouldHasAirbags = shouldHasAirbags

    }
}
```

;/。

“”

```
import UIKit

enum CarType {
    case

    sportage,
    saloon
}

enum GearType {
    case

    manual,
```

```
    automatic
}

struct Motor {
    var id: String
    var name: String
    var model: String
    var numberOfCylinders: UInt8
}

class Car: CustomStringConvertible {
    var color: UIColor
    var numberOfSeats: UInt8
    var numberOfWheels: UInt8
    var type: CarType
    var gearType: GearType
    var motor: Motor
    var shouldHasAirbags: Bool

    var description: String {
        return "color: \(color)\nNumber of seats: \(numberOfSeats)\nNumber of Wheels:
\(numberOfWheels)\n Type: \(gearType)\nMotor: \(motor)\nAirbag Availability:
\(shouldHasAirbags)"
    }

    init(color: UIColor, numberOfSeats: UInt8, numberOfWheels: UInt8, type: CarType, gearType:
GearType, motor: Motor, shouldHasAirbags: Bool) {

        self.color = color
        self.numberOfSeats = numberOfSeats
        self.numberOfWheels = numberOfWheels
        self.type = type
        self.gearType = gearType
        self.motor = motor
        self.shouldHasAirbags = shouldHasAirbags

    }
}
```

`CarBluePrint`'batteryName'。 `CarBuilderbatteryName`。

`batteryName` new`CarBluePrint` `CarCarBuilder`

```
import UIKit

enum CarType {
    case

    sportage,
    saloon
}

enum GearType {
    case

    manual,
    automatic
}

struct Motor {
```

```
    var id: String
    var name: String
    var model: String
    var numberOfCylinders: UInt8
}

class Car: CustomStringConvertible {
    var color: UIColor
    var numberOfSeats: UInt8
    var numberOfWheels: UInt8
    var type: CarType
    var gearType: GearType
    var motor: Motor
    var shouldHasAirbags: Bool

    var description: String {
        return "color: \(color)\nNumber of seats: \(numberOfSeats)\nNumber of Wheels:
\(numberOfWheels)\n Type: \(gearType)\nMotor: \(motor)\nAirbag Availability:
\(shouldHasAirbags)"
    }

    init(color: UIColor, numberOfSeats: UInt8, numberOfWheels: UInt8, type: CarType, gearType:
GearType, motor: Motor, shouldHasAirbags: Bool) {

        self.color = color
        self.numberOfSeats = numberOfSeats
        self.numberOfWheels = numberOfWheels
        self.type = type
        self.gearType = gearType
        self.motor = motor
        self.shouldHasAirbags = shouldHasAirbags

    }
}
```

CarBuilder

```
import UIKit

enum CarType {
    case

    sportage,
    saloon
}

enum GearType {
    case

    manual,
    automatic
}

struct Motor {
    var id: String
    var name: String
    var model: String
    var numberOfCylinders: UInt8
}
```

```
class Car: CustomStringConvertible {
    var color: UIColor
    var numberOfSeats: UInt8
    var numberOfWheels: UInt8
    var type: CarType
    var gearType: GearType
    var motor: Motor
    var shouldHasAirbags: Bool

    var description: String {
        return "color: \(color)\nNumber of seats: \(numberOfSeats)\nNumber of Wheels:
\(numberOfWheels)\n Type: \(gearType)\nMotor: \(motor)\nAirbag Availability:
\(shouldHasAirbags)"
    }

    init(color: UIColor, numberOfSeats: UInt8, numberOfWheels: UInt8, type: CarType, gearType:
GearType, motor: Motor, shouldHasAirbags: Bool) {

        self.color = color
        self.numberOfSeats = numberOfSeats
        self.numberOfWheels = numberOfWheels
        self.type = type
        self.gearType = gearType
        self.motor = motor
        self.shouldHasAirbags = shouldHasAirbags

    }
}
```

- https://riptutorial.com/zh-TW/swift/topic/4941/---

48: -

。 。

。

Examples

Adaptee Target。 。

Swift。 TargetAdaptee。

```
// The functionality to which a Client has no direct access
class Adaptee {
    func foo() {
        // ...
    }
}

// Target's functionality, to which a Client does have direct access
protocol TargetFunctionality {
    func fooBar() {}
}

// Adapter used to pass the request on the Target to a request on the Adaptee
extension Adaptee: TargetFunctionality {
    func fooBar() {
        foo()
    }
}
```

```
Client -> Target -> Adapter -> Adaptee
```

。 。

Facade。 。

UserDefaultsFacade。

```
enum Defaults {

    static func set(_ object: Any, forKey defaultName: String) {
        let defaults: UserDefaults = UserDefaults.standard
        defaults.set(object, forKey:defaultName)
        defaults.synchronize()
    }

    static func object(forKey key: String) -> AnyObject! {
        let defaults: UserDefaults = UserDefaults.standard
        return defaults.object(forKey: key) as AnyObject!
    }

}
```

。

```
enum Defaults {

    static func set(_ object: Any, forKey defaultName: String) {
        let defaults: UserDefaults = UserDefaults.standard
        defaults.set(object, forKey:defaultName)
        defaults.synchronize()
    }

    static func object(forKey key: String) -> AnyObject! {
        let defaults: UserDefaults = UserDefaults.standard
        return defaults.object(forKey: key) as AnyObject!
    }

}
```

UserDefaults。

- https://riptutorial.com/zh-TW/swift/topic/9497/---

49:

SwiftiBook 。

Examples

var

```
var num: Int = 10
```

```
var num: Int = 10
```

let

```
var num: Int = 10
```

Swift

```
var num: Int = 10
```

- unicode

  Unicode。

  developer.apple.com

```
var num: Int = 10
```

“”。 /

```
class Dog {
    var name = ""
}
```

DognameString 。 Dog

```
class Dog {
    var name = ""
}
```

。

。 。 lazy

```
lazy var veryExpensiveVariable = expensiveMethod()
```

```
lazy var veryExpensiveVariable = expensiveMethod()
```

var。

gettersetter。

```
var pi = 3.14

class Circle {
    var radius = 0.0
    var circumference: Double {
        get {
            return pi * radius * 2
        }
        set {
            radius = newValue / pi / 2
        }
    }
}

let circle = Circle()
circle.radius = 1
print(circle.circumference) // Prints "6.28"
circle.circumference = 14
print(circle.radius) // Prints "2.229..."
```

var

```
var pi = 3.14

class Circle {
    var radius = 0.0
    var circumference: Double {
        get {
            return pi * radius * 2
        }
        set {
            radius = newValue / pi / 2
        }
    }
}

let circle = Circle()
circle.radius = 1
print(circle.circumference) // Prints "6.28"
circle.circumference = 14
print(circle.radius) // Prints "2.229..."
```

get

```
var pi = 3.14

class Circle {
    var radius = 0.0
    var circumference: Double {
        get {
            return pi * radius * 2
```

```
        }
        set {
            radius = newValue / pi / 2
        }
    }
}

let circle = Circle()
circle.radius = 1
print(circle.circumference) // Prints "6.28"
circle.circumference = 14
print(circle.radius) // Prints "2.229..."
```

```
func printSomething() {
    let localString = "I'm local!"
    print(localString)
}

func printSomethingAgain() {
    print(localString) // error
}
```

。

```
func printSomething() {
    let localString = "I'm local!"
    print(localString)
}

func printSomethingAgain() {
    print(localString) // error
}
```

“”。

。 。 `static`type

```
struct Dog {
    static var noise = "Bark!"
}

print(Dog.noise) // Prints "Bark!"
```

`class``static`。

```
struct Dog {
    static var noise = "Bark!"
}

print(Dog.noise) // Prints "Bark!"
```

。

。

```
var myProperty = 5 {
    willSet {
        print("Will set to \(newValue). It was previously \(myProperty)")
    }
    didSet {
        print("Did set to \(myProperty). It was previously \(oldValue)")
    }
}
myProperty = 6
// prints: Will set to 6, It was previously 5
// prints: Did set to 6. It was previously 5
```

- myProperty willSet。 newValue myProperty。
- myProperty didSet。 oldValue myProperty。

  didSetwillSet

  -
  - didSetwillSet

- didSetwillSet oldValuenewValue

```
var myProperty = 5 {
    willSet {
        print("Will set to \(newValue). It was previously \(myProperty)")
    }
    didSet {
        print("Did set to \(myProperty). It was previously \(oldValue)")
    }
}
myProperty = 6
// prints: Will set to 6, It was previously 5
// prints: Did set to 6. It was previously 5
```

setter

- willSet(oldValue)didSet(newValue)。

https://riptutorial.com/zh-TW/swift/topic/536/

50:

“nil”

Apple Inc.“The Swift Programming LanguageSwift 3.1 Edition。”iBooks。
https://itun.es/us/k5SW7.l

letif-letguard-letswitchcase-let coalesce??nil

- var optionalNameoptionalType //nil
- var optionalNameoptionalType = value //
- var optionalNameoptionalType //
- //

Swift 。

Examples

Optionals。 `nil` 。

SwiftApple。 Apple`nil`Objective-CAPI。

Swift`nil`。 a`!?`。 `Int`

```
var numberOne: Int! = nil
var numberTwo: Int? = nil
```

`?`。 `Int` `Int?` nil

```
var numberOne: Int! = nil
var numberTwo: Int? = nil
```

`!`。 `UIButton!viewDidLoad()`

```
var numberOne: Int! = nil
var numberTwo: Int? = nil
```

Optional。

Optional`!` Optional `!`。

“”“”。

`nil`*nil* `!`。

```
var text: String? = nil
```

```
var unwrapped: String = text! //crashes with "unexpectedly found nil while unwrapping an
Optional value"
```

`if-letnil`

```
var text: String? = nil
var unwrapped: String = text! //crashes with "unexpectedly found nil while unwrapping an
Optional value"
```

```
var text: String? = nil
var unwrapped: String = text! //crashes with "unexpectedly found nil while unwrapping an
Optional value"
```

`unwrappedNumberif-letguard`◦

```
var text: String? = nil
var unwrapped: String = text! //crashes with "unexpectedly found nil while unwrapping an
Optional value"
```

◦

◦ if - else

```
var text: String? = nil
var unwrapped: String = text! //crashes with "unexpectedly found nil while unwrapping an
Optional value"
```

nil coalescing

```
func fallbackIfNil(str: String?) -> String {
    return str ?? "Fallback String"
}
print(fallbackIfNil("Hi")) // Prints "Hi"
print(fallbackIfNil(nil)) // Prints "Fallback String"
```

nil

```
func fallbackIfNil(str: String?) -> String {
    return str ?? "Fallback String"
}
print(fallbackIfNil("Hi")) // Prints "Hi"
print(fallbackIfNil(nil)) // Prints "Fallback String"
```

`foo`nil`someExpensiveComputation()` ◦

nil

```
func fallbackIfNil(str: String?) -> String {
    return str ?? "Fallback String"
}
print(fallbackIfNil("Hi")) // Prints "Hi"
```

```
print(fallbackIfNil(nil)) // Prints "Fallback String"
```

bazfoobar。

Optional Chaining 。 ?。

```
struct Foo {
    func doSomething() {
        print("Hello World!")
    }
}

var foo : Foo? = Foo()

foo?.doSomething() // prints "Hello World!" as foo is non-nil
```

foodoSomething() 。 foonil - 。

```
struct Foo {
    func doSomething() {
        print("Hello World!")
    }
}

var foo : Foo? = Foo()

foo?.doSomething() // prints "Hello World!" as foo is non-nil
```

Objective-Cnil

“”“”/。 。

```
struct Foo {
    func doSomething() {
        print("Hello World!")
    }
}

var foo : Foo? = Foo()

foo?.doSomething() // prints "Hello World!" as foo is non-nil
```

foo?.barInt?barfoo。

VoidVoid?。 。

```
struct Foo {
    func doSomething() {
        print("Hello World!")
    }
}

var foo : Foo? = Foo()
```

```
foo?.doSomething() // prints "Hello World!" as foo is non-nil
```

Void?nilfoonil。

-

。 Cnull。 。 -1。 “”。

Swift。

```
var possiblyInt: Int?
```

。

nil。

```
var possiblyInt: Int?
```

nil

```
var possiblyInt: Int?
```

!print。

;。

Int。

```
var possiblyInt: Int?
```

Swift Int。 nil 。

```
var possiblyInt: Int?
```

https://riptutorial.com/zh-TW/swift/topic/247/

51:

Swift 。

Examples

Swift

```
func reticulateSplines()                // no return value and no error
func reticulateSplines() -> Int         // always returns a value
func reticulateSplines() throws         // no return value, but may throw an error
func reticulateSplines() throws -> Int  // may either return a value or throw an error
```

ErrorType NSError。

2.0 2.2

```
func reticulateSplines()                // no return value and no error
func reticulateSplines() -> Int         // always returns a value
func reticulateSplines() throws         // no return value, but may throw an error
func reticulateSplines() throws -> Int  // may either return a value or throw an error
```

3.0

```
func reticulateSplines()                // no return value and no error
func reticulateSplines() -> Int         // always returns a value
func reticulateSplines() throws         // no return value, but may throw an error
func reticulateSplines() throws -> Int  // may either return a value or throw an error
```

`do` / `catch` `throw` `try` 。

```
func reticulateSplines()                // no return value and no error
func reticulateSplines() -> Int         // always returns a value
func reticulateSplines() throws         // no return value, but may throw an error
func reticulateSplines() throws -> Int  // may either return a value or throw an error
```

`do` / `catch`

```
func reticulateSplines()                // no return value and no error
func reticulateSplines() -> Int         // always returns a value
func reticulateSplines() throws         // no return value, but may throw an error
func reticulateSplines() throws -> Int  // may either return a value or throw an error
```

`try` `try?` `try!`

```
func reticulateSplines()                // no return value and no error
func reticulateSplines() -> Int         // always returns a value
func reticulateSplines() throws         // no return value, but may throw an error
func reticulateSplines() throws -> Int  // may either return a value or throw an error
```

。

2.2

```
enum CustomError: ErrorType {
    case SomeError
    case AnotherError
}

func throwing() throws {
    throw CustomError.SomeError
}
```

3.0

```
enum CustomError: ErrorType {
    case SomeError
    case AnotherError
}

func throwing() throws {
    throw CustomError.SomeError
}
```

Do-Catch`catcherror`

```
enum CustomError: ErrorType {
    case SomeError
    case AnotherError
}

func throwing() throws {
    throw CustomError.SomeError
}
```

```
enum CustomError: ErrorType {
    case SomeError
    case AnotherError
}

func throwing() throws {
    throw CustomError.SomeError
}
```

`catch`。 Do。

Do-Catch`CustomError` `NSError` 。

2.2

```
enum CustomError: ErrorType {
    case SomeError
    case AnotherError
}
```

```
func throwing() throws {
    throw CustomError.SomeError
}
```

3.0

Swift 3NSError。

```
enum CustomError: ErrorType {
    case SomeError
    case AnotherError
}

func throwing() throws {
    throw CustomError.SomeError
}
```

```
class Plane {

    enum Emergency: ErrorType {
        case NoFuel
        case EngineFailure(reason: String)
        case DamagedWing
    }

    var fuelInKilograms: Int

    //... init and other methods not shown

    func fly() throws {
        // ...
        if fuelInKilograms <= 0 {
            // uh oh...
            throw Emergency.NoFuel
        }
    }

}
```

```
class Plane {

    enum Emergency: ErrorType {
        case NoFuel
        case EngineFailure(reason: String)
        case DamagedWing
    }

    var fuelInKilograms: Int

    //... init and other methods not shown

    func fly() throws {
        // ...
        if fuelInKilograms <= 0 {
            // uh oh...
            throw Emergency.NoFuel
        }
    }
```

```
}
```

Swift。try。 try。 。

loadImageatPath :)。 。

```
let photo = try! loadImage(atPath: "./Resources/John Appleseed.jpg")
```

enum

```
enum RegistrationError: Error {
    case invalidEmail
    case invalidPassword
    case invalidPhoneNumber
}
```

RegistrationError extension。

```
enum RegistrationError: Error {
    case invalidEmail
    case invalidPassword
    case invalidPhoneNumber
}
```

```
enum RegistrationError: Error {
    case invalidEmail
    case invalidPassword
    case invalidPhoneNumber
}
```

https://riptutorial.com/zh-TW/swift/topic/283/

52:

◦ default case◦ ◦

Swift◦

Examples

```
let number = 3
switch number {
case 1:
    print("One!")
case 2:
    print("Two!")
case 3:
    print("Three!")
default:
    print("Not One, Two or Three")
}
```

switch◦ ◦

```
let number = 3
switch number {
case 1:
    print("One!")
case 2:
    print("Two!")
case 3:
    print("Three!")
default:
    print("Not One, Two or Three")
}
```

```
let number = 3
switch number {
case 1:
    print("One!")
case 2:
    print("Two!")
case 3:
    print("Three!")
default:
    print("Not One, Two or Three")
}
```

switchcase◦

```
let number = 3
```

```
switch number {
case 1, 2:
    print("One or Two!")
case 3:
    print("Three!")
case 4, 5, 6:
    print("Four, Five or Six!")
default:
    print("Not One, Two, Three, Four, Five or Six")
}
```

switchcase。

```
let number = 20
switch number {
case 0:
    print("Zero")
case 1..<10:
    print("Between One and Ten")
case 10..<20:
    print("Between Ten and Twenty")
case 20..<30:
    print("Between Twenty and Thirty")
default:
    print("Greater than Thirty or less than Zero")
}
```

where

whereswitch case。

```
switch (temperature) {
      case 0...49 where temperature % 2 == 0:
        print("Cold and even")

      case 50...79 where temperature % 2 == 0:
        print("Warm and even")

      case 80...110 where temperature % 2 == 0:
        print("Hot and even")

      default:
        print("Temperature out of range or odd")
}
```

```
var str: String? = "hi"
var x: Int? = 5

switch (str, x) {
case (.Some,.Some):
    print("Both have values")
case (.Some, nil):
    print("String has a value")
case (nil, .Some):
    print("Int has a value")
case (nil, nil):
```

```
    print("Neither have values")
}
```

Switch。

```
let coordinates: (x: Int, y: Int, z: Int) = (3, 2, 5)

switch (coordinates) {
case (0, 0, 0): // 1
  print("Origin")
case (_, 0, 0): // 2
  print("On the x-axis.")
case (0, _, 0): // 3
  print("On the y-axis.")
case (0, 0, _): // 4
  print("On the z-axis.")
default:        // 5
  print("Somewhere in space")
}
```

1. 0,0,0。 3D。
2. y = 0z = 0x。 x。
3. x = 0z = 0y。 。
4. x = 0y = 0z。 z。
5. 。

。

switch

```
let coordinates: (x: Int, y: Int, z: Int) = (3, 2, 5)

switch (coordinates) {
case (0, 0, 0): // 1
  print("Origin")
case (_, 0, 0): // 2
  print("On the x-axis.")
case (0, _, 0): // 3
  print("On the y-axis.")
case (0, 0, _): // 4
  print("On the z-axis.")
default:        // 5
  print("Somewhere in space")
}
```

let。 。

switch。 - 。 switch。

let-where。

```
let coordinates: (x: Int, y: Int, z: Int) = (3, 2, 5)
```

```
switch (coordinates) {
case (0, 0, 0): // 1
  print("Origin")
case (_, 0, 0): // 2
  print("On the x-axis.")
case (0, _, 0): // 3
  print("On the y-axis.")
case (0, 0, _): // 4
  print("On the z-axis.")
default:        // 5
  print("Somewhere in space")
}
```

“yx”“yx”。

swiftcase。 `fallthrough`。

```
switch(value) {
case 'one':
    // do operation one
    fallthrough
case 'two':
    // do this either independant, or in conjunction with first case
default:
    // default operation
}
```

。

SwitchEnum

```
enum CarModel {
    case Standard, Fast, VeryFast
}

let car = CarModel.Standard

switch car {
case .Standard: print("Standard")
case .Fast: print("Fast")
case .VeryFast: print("VeryFast")
}
```

`default`。

。

```
var result: AnyObject? = someMethod()

switch result {
case nil:
    print("result is nothing")
case is String:
    print("result is a String")
case _ as Double:
```

```
    print("result is not nil, any value that is a Double")
case let myInt as Int where myInt > 0:
    print("\(myInt) value is not nil but an int and greater than 0")
case let a?:
    print("\(a) - value is unwrapped")
}
```

```
public typealias mdyTuple = (month: Int, day: Int, year: Int)

let fredsBirthday =   (month: 4, day: 3, year: 1973)




switch theMDY
{
//You can match on a literal tuple:
case (fredsBirthday):
  message = "\(date) \(prefix) the day Fred was born"

//You can match on some of the terms, and ignore others:
case (3, 15, _):
  message = "Beware the Ides of March"

//You can match on parts of a literal tuple, and copy other elements
//into a constant that you use in the body of the case:
case (bobsBirthday.month, bobsBirthday.day, let year) where year > bobsBirthday.year:
  message = "\(date) \(prefix) Bob's \(possessiveNumber(year - bobsBirthday.year))" +
    "birthday"

//You can copy one or more elements of the tuple into a constant and then
//add a where clause that further qualifies the case:
case (susansBirthday.month, susansBirthday.day, let year)
  where year > susansBirthday.year:
  message = "\(date) \(prefix) Susan's " +
    "\(possessiveNumber(year - susansBirthday.year)) birthday"

//You can match some elements to ranges:.
case (5, 1...15, let year):
  message = "\(date) \(prefix) in the first half of May, \(year)"
}
```

- prepareForSegue

switch。

`prepareForSegue` 。 segue。 segue。 seguessegue。

Swift。

Swift `case let var as Class`

3.0

```
override func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {
  switch segue.destinationViewController {
    case let fooViewController as FooViewController:
```

```
      fooViewController.delegate = self

    case let barViewController as BarViewController:
      barViewController.data = data

    default:
      break
  }
}
```

3.0

Swift 3sytax

```
override func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {
  switch segue.destinationViewController {
    case let fooViewController as FooViewController:
      fooViewController.delegate = self

    case let barViewController as BarViewController:
      barViewController.data = data

    default:
      break
  }
}
```

https://riptutorial.com/zh-TW/swift/topic/207/

53: JSON

- NSJSONSerialization.JSONObjectWithDatajsonDataoptionsNSJSONReadingOptions//jsonDataObject。 。
- NSJSONSerialization.dataWithJSONObjectjsonObjectoptionsNSJSONWritingOptions//JSONNSData。 NSJSONWritingOptions.PrettyPrinted。

Examples

Apple FoundationSwiftJSON

JSONSerializationAppleFoundation。

2.2

JSON

`JSONObjectWithDataNSData AnyObject 。 as?。`

```
do {
    guard let jsonData = "[\"Hello\", \"JSON\"]".dataUsingEncoding(NSUTF8StringEncoding) else
{
        fatalError("couldn't encode string as UTF-8")
    }

    // Convert JSON from NSData to AnyObject
    let jsonObject = try NSJSONSerialization.JSONObjectWithData(jsonData, options: [])

    // Try to convert AnyObject to array of strings
    if let stringArray = jsonObject as? [String] {
        print("Got array of strings: \(stringArray.joinWithSeparator(", "))")
    }
} catch {
    print("error reading JSON: \(error)")
}
```

`options: .AllowFragmentsoptions: []`JSON。

JSON

`dataWithJSONObject`JSON`NSNull`UTF-8`NSData 。`

```
do {
    guard let jsonData = "[\"Hello\", \"JSON\"]".dataUsingEncoding(NSUTF8StringEncoding) else
{
        fatalError("couldn't encode string as UTF-8")
    }

    // Convert JSON from NSData to AnyObject
    let jsonObject = try NSJSONSerialization.JSONObjectWithData(jsonData, options: [])
```

```
    // Try to convert AnyObject to array of strings
    if let stringArray = jsonObject as? [String] {
        print("Got array of strings: \(stringArray.joinWithSeparator(", "))")
    }
} catch {
    print("error reading JSON: \(error)")
}
```

options: .PrettyPrintedoptions: []。

3.0

Swift 3。

```
do {
    guard let jsonData = "[\"Hello\", \"JSON\"]".dataUsingEncoding(NSUTF8StringEncoding) else
{
        fatalError("couldn't encode string as UTF-8")
    }

    // Convert JSON from NSData to AnyObject
    let jsonObject = try NSJSONSerialization.JSONObjectWithData(jsonData, options: [])

    // Try to convert AnyObject to array of strings
    if let stringArray = jsonObject as? [String] {
        print("Got array of strings: \(stringArray.joinWithSeparator(", "))")
    }
} catch {
    print("error reading JSON: \(error)")
}
```

Swift 4.0。

Swift 4.0Swift`EncodableDecodable`。 `EncoderDecoder`JSON。 `Codable``EncodableDecodable`。 JSON。

`Codable`。 `String` `IntDouble`;`Date` `DataURL`。 `Codable`。

`BookCodable`。

```
do {
    guard let jsonData = "[\"Hello\", \"JSON\"]".dataUsingEncoding(NSUTF8StringEncoding) else
{
        fatalError("couldn't encode string as UTF-8")
    }

    // Convert JSON from NSData to AnyObject
    let jsonObject = try NSJSONSerialization.JSONObjectWithData(jsonData, options: [])

    // Try to convert AnyObject to array of strings
    if let stringArray = jsonObject as? [String] {
        print("Got array of strings: \(stringArray.joinWithSeparator(", "))")
    }
} catch {
    print("error reading JSON: \(error)")
}
```

ArrayDictionaryCodable。

Codable BookApple FoundationJSONEncoderJSONDecoderJSONJSONBookJSON。 EncoderDecoder。

JSON

```
do {
    guard let jsonData = "[\"Hello\", \"JSON\"]".dataUsingEncoding(NSUTF8StringEncoding) else
{
        fatalError("couldn't encode string as UTF-8")
    }

    // Convert JSON from NSData to AnyObject
    let jsonObject = try NSJSONSerialization.JSONObjectWithData(jsonData, options: [])

    // Try to convert AnyObject to array of strings
    if let stringArray = jsonObject as? [String] {
        print("Got array of strings: \(stringArray.joinWithSeparator(", "))")
    }
} catch {
    print("error reading JSON: \(error)")
}
```

encoder.outputFormatting = .prettyPrinted。 ##JSON

JSON

```
do {
    guard let jsonData = "[\"Hello\", \"JSON\"]".dataUsingEncoding(NSUTF8StringEncoding) else
{
        fatalError("couldn't encode string as UTF-8")
    }

    // Convert JSON from NSData to AnyObject
    let jsonObject = try NSJSONSerialization.JSONObjectWithData(jsonData, options: [])

    // Try to convert AnyObject to array of strings
    if let stringArray = jsonObject as? [String] {
        print("Got array of strings: \(stringArray.joinWithSeparator(", "))")
    }
} catch {
    print("error reading JSON: \(error)")
}
```

Book.selfJSON。

APIJSONJSONAPI。

JSONEncodable 。

```
do {
    guard let jsonData = "[\"Hello\", \"JSON\"]".dataUsingEncoding(NSUTF8StringEncoding) else
{
```

```
        fatalError("couldn't encode string as UTF-8")
    }

    // Convert JSON from NSData to AnyObject
    let jsonObject = try NSJSONSerialization.JSONObjectWithData(jsonData, options: [])

    // Try to convert AnyObject to array of strings
    if let stringArray = jsonObject as? [String] {
        print("Got array of strings: \(stringArray.joinWithSeparator(", "))")
    }
} catch {
    print("error reading JSON: \(error)")
}
```

JSON`Decodable`。

```
do {
    guard let jsonData = "[\"Hello\", \"JSON\"]".dataUsingEncoding(NSUTF8StringEncoding) else
{
        fatalError("couldn't encode string as UTF-8")
    }

    // Convert JSON from NSData to AnyObject
    let jsonObject = try NSJSONSerialization.JSONObjectWithData(jsonData, options: [])

    // Try to convert AnyObject to array of strings
    if let stringArray = jsonObject as? [String] {
        print("Got array of strings: \(stringArray.joinWithSeparator(", "))")
    }
} catch {
    print("error reading JSON: \(error)")
}
```

APISwift。 JSONJSON。 `CodingKey`。

```
do {
    guard let jsonData = "[\"Hello\", \"JSON\"]".dataUsingEncoding(NSUTF8StringEncoding) else
{
        fatalError("couldn't encode string as UTF-8")
    }

    // Convert JSON from NSData to AnyObject
    let jsonObject = try NSJSONSerialization.JSONObjectWithData(jsonData, options: [])

    // Try to convert AnyObject to array of strings
    if let stringArray = jsonObject as? [String] {
        print("Got array of strings: \(stringArray.joinWithSeparator(", "))")
    }
} catch {
    print("error reading JSON: \(error)")
}
```

`CodingKeysCodablepublicationDatepublication_date` API。

SwiftyJSON

SwiftyJSONSwiftJSON。

https //github.com/SwiftyJSON/SwiftyJSON

SwiftyJSONJSON

```
if let jsonObject = try NSJSONSerialization.JSONObjectWithData(data, options: .AllowFragments)
as? [[String: AnyObject]],
let bookName = (jsonObject[0]["book"] as? [String: AnyObject])?["name"] as? String {
    //We can now use the book name
}
```

SwiftyJSON

```
if let jsonObject = try NSJSONSerialization.JSONObjectWithData(data, options: .AllowFragments)
as? [[String: AnyObject]],
let bookName = (jsonObject[0]["book"] as? [String: AnyObject])?["name"] as? String {
    //We can now use the book name
}
```

nil。

SwiftyJSONGit - Swift 3。 “SwiftyJSON.swift”

```
if let jsonObject = try NSJSONSerialization.JSONObjectWithData(data, options: .AllowFragments)
as? [[String: AnyObject]],
let bookName = (jsonObject[0]["book"] as? [String: AnyObject])?["name"] as? String {
    //We can now use the book name
}
```

JSON

```
if let jsonObject = try NSJSONSerialization.JSONObjectWithData(data, options: .AllowFragments)
as? [[String: AnyObject]],
let bookName = (jsonObject[0]["book"] as? [String: AnyObject])?["name"] as? String {
    //We can now use the book name
}
```

```
if let jsonObject = try NSJSONSerialization.JSONObjectWithData(data, options: .AllowFragments)
as? [[String: AnyObject]],
let bookName = (jsonObject[0]["book"] as? [String: AnyObject])?["name"] as? String {
    //We can now use the book name
}
```

```
if let jsonObject = try NSJSONSerialization.JSONObjectWithData(data, options: .AllowFragments)
as? [[String: AnyObject]],
let bookName = (jsonObject[0]["book"] as? [String: AnyObject])?["name"] as? String {
    //We can now use the book name
}
```

```
if let jsonObject = try NSJSONSerialization.JSONObjectWithData(data, options: .AllowFragments)
as? [[String: AnyObject]],
```

```
let bookName = (jsonObject[0]["book"] as? [String: AnyObject])?["name"] as? String {
    //We can now use the book name
}
```

swift

```
if let jsonObject = try NSJSONSerialization.JSONObjectWithData(data, options: .AllowFragments)
as? [[String: AnyObject]],
let bookName = (jsonObject[0]["book"] as? [String: AnyObject])?["name"] as? String {
    //We can now use the book name
}
```

```
if let jsonObject = try NSJSONSerialization.JSONObjectWithData(data, options: .AllowFragments)
as? [[String: AnyObject]],
let bookName = (jsonObject[0]["book"] as? [String: AnyObject])?["name"] as? String {
    //We can now use the book name
}
```

SwiftyJSON

```
if let jsonObject = try NSJSONSerialization.JSONObjectWithData(data, options: .AllowFragments)
as? [[String: AnyObject]],
let bookName = (jsonObject[0]["book"] as? [String: AnyObject])?["name"] as? String {
    //We can now use the book name
}
```

JSON

```
if let jsonObject = try NSJSONSerialization.JSONObjectWithData(data, options: .AllowFragments)
as? [[String: AnyObject]],
let bookName = (jsonObject[0]["book"] as? [String: AnyObject])?["name"] as? String {
    //We can now use the book name
}
```

JSON

```
if let jsonObject = try NSJSONSerialization.JSONObjectWithData(data, options: .AllowFragments)
as? [[String: AnyObject]],
let bookName = (jsonObject[0]["book"] as? [String: AnyObject])?["name"] as? String {
    //We can now use the book name
}
```

FreddyBig Nerd RanchJSON。

1. JSON。
2. Swift。
3. JSON。

JSON

JSON。

```
{
  "success": true,
  "people": [
    {
      "name": "Matt Mathias",
      "age": 32,
      "spouse": true
    },
    {
      "name": "Sergeant Pepper",
      "age": 25,
      "spouse": false
    }
  ],
  "jobs": [
    "teacher",
    "judge"
  ],
  "states": {
    "Georgia": [
      30301,
      30302,
      30303
    ],
    "Wisconsin": [
      53000,
      53001
    ]
  }
}
```


```
{
  "success": true,
  "people": [
    {
      "name": "Matt Mathias",
      "age": 32,
      "spouse": true
    },
    {
      "name": "Sergeant Pepper",
      "age": 25,
      "spouse": false
    }
  ],
  "jobs": [
    "teacher",
    "judge"
  ],
  "states": {
    "Georgia": [
      30301,
      30302,
      30303
    ],
    "Wisconsin": [
      53000,
      53001
```

```
    ]
  }
}
```

JSON。

```
{
  "success": true,
  "people": [
    {
      "name": "Matt Mathias",
      "age": 32,
      "spouse": true
    },
    {
      "name": "Sergeant Pepper",
      "age": 25,
      "spouse": false
    }
  ],
  "jobs": [
    "teacher",
    "judge"
  ],
  "states": {
    "Georgia": [
      30301,
      30302,
      30303
    ],
    "Wisconsin": [
      53000,
      53001
    ]
  }
}
```

try json"success" - 。

JSON。 。

```
{
  "success": true,
  "people": [
    {
      "name": "Matt Mathias",
      "age": 32,
      "spouse": true
    },
    {
      "name": "Sergeant Pepper",
      "age": 25,
      "spouse": false
    }
  ],
  "jobs": [
    "teacher",
    "judge"
```

```
  ],
  "states": {
    "Georgia": [
      30301,
      30302,
      30303
    ],
    "Wisconsin": [
      53000,
      53001
    ]
  }
}
```

JSONJSONDecodable。

```
{
  "success": true,
  "people": [
    {
      "name": "Matt Mathias",
      "age": 32,
      "spouse": true
    },
    {
      "name": "Sergeant Pepper",
      "age": 25,
      "spouse": false
    }
  ],
  "jobs": [
    "teacher",
    "judge"
  ],
  "states": {
    "Georgia": [
      30301,
      30302,
      30303
    ],
    "Wisconsin": [
      53000,
      53001
    ]
  }
}
```

JSONNSData 。

```
{
  "success": true,
  "people": [
    {
      "name": "Matt Mathias",
      "age": 32,
      "spouse": true
    },
    {
```

```
      "name": "Sergeant Pepper",
      "age": 25,
      "spouse": false
    }
  ],
  "jobs": [
    "teacher",
    "judge"
  ],
  "states": {
    "Georgia": [
      30301,
      30302,
      30303
    ],
    "Wisconsin": [
      53000,
      53001
    ]
  }
}
```

JSONEncodableNSData。

```
{
  "success": true,
  "people": [
    {
      "name": "Matt Mathias",
      "age": 32,
      "spouse": true
    },
    {
      "name": "Sergeant Pepper",
      "age": 25,
      "spouse": false
    }
  ],
  "jobs": [
    "teacher",
    "judge"
  ],
  "states": {
    "Georgia": [
      30301,
      30302,
      30303
    ],
    "Wisconsin": [
      53000,
      53001
    ]
  }
}
```

ArrowSwiftJSON。

JSON<--

```
identifier <-- json["id"]
name <-- json["name"]
stats <-- json["stats"]
```

Swift

```
identifier <-- json["id"]
name <-- json["name"]
stats <-- json["stats"]
```

JSON

```
identifier <-- json["id"]
name <-- json["name"]
stats <-- json["stats"]
```

```
identifier <-- json["id"]
name <-- json["name"]
stats <-- json["stats"]
```

```
identifier <-- json["id"]
name <-- json["name"]
stats <-- json["stats"]
```

```
identifier <-- json["id"]
name <-- json["name"]
stats <-- json["stats"]
```

CocoaPods

```
identifier <-- json["id"]
name <-- json["name"]
stats <-- json["stats"]
```

XcodeArrow.swift

https://github.com/s4cha/Arrow

GitHubArrowFramework。 。

JSON

JSON`Todo` as

```
struct Todo {
    let comment: String
}
```

JSON`NSJSONSerializationNSData`。

`JSONDecodable`

```
struct Todo {
    let comment: String
}
```

`TodoJSONDecodable`

```
struct Todo {
    let comment: String
}
```

json

```
struct Todo {
    let comment: String
}
```

API`AnyObject` `JSONDictionary`

```
struct Todo {
    let comment: String
}
```

JSON`Todo`todos

```
struct Todo {
    let comment: String
}
```

`flatMap`Todonil

```
struct Todo {
    let comment: String
}
```

JSONSwift 3

`animals.json`JSON

```
{
    "Sea Animals": [
    {
     "name": "Fish",
     "question": "How many species of fish are there?"    },
                {
                "name": "Sharks",
                "question": "How long do sharks live?"
                },
                {
                "name": "Squid",
                "question": "Do squids have brains?"
                },
```

```
                {
                "name": "Octopus",
                "question": "How big do octopus get?"
                },
                {
                "name": "Star Fish",
                "question": "How long do star fish live?"
                }
                ],
    "mammals": [
                {
      "name": "Dog",
      "question": "How long do dogs live?"
    },
    {
      "name": "Elephant",
      "question": "How much do baby elephants weigh?"
    },
    {
      "name": "Cats",
      "question": "Do cats really have 9 lives?"
    },
    {
      "name": "Tigers",
      "question": "Where do tigers live?"
    },
    {
      "name": "Pandas",
      "question": "WHat do pandas eat?"
    }   ] }
```

JSON

JSON

```
{
    "Sea Animals": [
    {
     "name": "Fish",
     "question": "How many species of fish are there?"    },
                {
                "name": "Sharks",
                "question": "How long do sharks live?"
                },
                {
                "name": "Squid",
                "question": "Do squids have brains?"
                },
                {
                "name": "Octopus",
                "question": "How big do octopus get?"
                },
                {
                "name": "Star Fish",
                "question": "How long do star fish live?"
                }
                ],
    "mammals": [
                {
      "name": "Dog",
```

```
      "question": "How long do dogs live?"
    },
    {
      "name": "Elephant",
      "question": "How much do baby elephants weigh?"
    },
    {
      "name": "Cats",
      "question": "Do cats really have 9 lives?"
    },
    {
      "name": "Tigers",
      "question": "Where do tigers live?"
    },
    {
      "name": "Pandas",
      "question": "WHat do pandas eat?"
    }   ] }
```

NSObject。

`ParsingObject`swift。

JSON

- swift`namequestion`

```
{
    "Sea Animals": [
    {
     "name": "Fish",
     "question": "How many species of fish are there?"    },
                {
                "name": "Sharks",
                "question": "How long do sharks live?"
                },
                {
                "name": "Squid",
                "question": "Do squids have brains?"
                },
                {
                "name": "Octopus",
                "question": "How big do octopus get?"
                },
                {
                "name": "Star Fish",
                "question": "How long do star fish live?"
                }
                ],
    "mammals": [
                {
      "name": "Dog",
      "question": "How long do dogs live?"
    },
    {
      "name": "Elephant",
      "question": "How much do baby elephants weigh?"
    },
    {
```

```
      "name": "Cats",
      "question": "Do cats really have 9 lives?"
    },
    {
      "name": "Tigers",
      "question": "Where do tigers live?"
    },
    {
      "name": "Pandas",
      "question": "WHat do pandas eat?"
    }   ] }
```

NSObjectViewController.swift var array = ParsingObject。

```
{
    "Sea Animals": [
    {
     "name": "Fish",
     "question": "How many species of fish are there?"    },
                {
                "name": "Sharks",
                "question": "How long do sharks live?"
                },
                {
                "name": "Squid",
                "question": "Do squids have brains?"
                },
                {
                "name": "Octopus",
                "question": "How big do octopus get?"
                },
                {
                "name": "Star Fish",
                "question": "How long do star fish live?"
                }
                ],
    "mammals": [
                {
      "name": "Dog",
      "question": "How long do dogs live?"
    },
    {
      "name": "Elephant",
      "question": "How much do baby elephants weigh?"
    },
    {
      "name": "Cats",
      "question": "Do cats really have 9 lives?"
    },
    {
      "name": "Tigers",
      "question": "Where do tigers live?"
    },
    {
      "name": "Pandas",
      "question": "WHat do pandas eat?"
    }   ] }
```

tableview

```
{
    "Sea Animals": [
    {
     "name": "Fish",
     "question": "How many species of fish are there?"    },
                {
                "name": "Sharks",
                "question": "How long do sharks live?"
                },
                {
                "name": "Squid",
                "question": "Do squids have brains?"
                },
                {
                "name": "Octopus",
                "question": "How big do octopus get?"
                },
                {
                "name": "Star Fish",
                "question": "How long do star fish live?"
                }
                ],
    "mammals": [
                {
      "name": "Dog",
      "question": "How long do dogs live?"
    },
    {
      "name": "Elephant",
      "question": "How much do baby elephants weigh?"
    },
    {
      "name": "Cats",
      "question": "Do cats really have 9 lives?"
    },
    {
      "name": "Tigers",
      "question": "Where do tigers live?"
    },
    {
      "name": "Pandas",
      "question": "WHat do pandas eat?"
    }   ] }
```

JSON https://riptutorial.com/zh-TW/swift/topic/223/json

54:

- var closureVar<parameters> - ><returnType>//
- typealias ClosureType =<parameters> - ><returnType>
- <statement{[<captureList>]<parameters><throws-ness> - > <returnType>} //

SwiftApple 。

Examples

lambdas 。

```
let sayHi = { print("Hello") }
// The type of sayHi is "() -> ()", aka "() -> Void"

sayHi()  // prints "Hello"
```

```
let sayHi = { print("Hello") }
// The type of sayHi is "() -> ()", aka "() -> Void"

sayHi()  // prints "Hello"
```

```
let sayHi = { print("Hello") }
// The type of sayHi is "() -> ()", aka "() -> Void"

sayHi()  // prints "Hello"
```

```
let sayHi = { print("Hello") }
// The type of sayHi is "() -> ()", aka "() -> Void"

sayHi()  // prints "Hello"
```

`{` `[` *capture list* `]` `(` `)` *throws-ness* `->` *body* `in` *body* `}` 。

```
let addOne = { [] (x: Int) -> Int in return x + 1 }
let addOne = { [] (x: Int) -> Int in x + 1 }
let addOne = { (x: Int) -> Int in x + 1 }
let addOne = { x -> Int in x + 1 }
let addOne = { x in x + 1 }
let addOne = { $0 + 1 }

let addOneOrThrow = { [] (x: Int) throws -> Int in return x + 1 }
let addOneOrThrow = { [] (x: Int) throws -> Int in x + 1 }
let addOneOrThrow = { (x: Int) throws -> Int in x + 1 }
let addOneOrThrow = { x throws -> Int in x + 1 }
let addOneOrThrow = { x throws in x + 1 }
```

- 。
- 。
-

。

- ;$0 $1 $2。
- return。
- throws 。

```
let addOne = { [] (x: Int) -> Int in return x + 1 }
let addOne = { [] (x: Int) -> Int in x + 1 }
let addOne = { (x: Int) -> Int in x + 1 }
let addOne = { x -> Int in x + 1 }
let addOne = { x in x + 1 }
let addOne = { $0 + 1 }

let addOneOrThrow = { [] (x: Int) throws -> Int in return x + 1 }
let addOneOrThrow = { [] (x: Int) throws -> Int in x + 1 }
let addOneOrThrow = { (x: Int) throws -> Int in x + 1 }
let addOneOrThrow = { x throws -> Int in x + 1 }
let addOneOrThrow = { x throws in x + 1 }
```

```
func foo(value: Double, block: () -> Void) { ... }
func foo(value: Double, block: Int -> Int) { ... }
func foo(value: Double, block: (Int, Int) -> String) { ... }
```

{ / }

```
func foo(value: Double, block: () -> Void) { ... }
func foo(value: Double, block: Int -> Int) { ... }
func foo(value: Double, block: (Int, Int) -> String) { ... }
```

()

```
func foo(value: Double, block: () -> Void) { ... }
func foo(value: Double, block: Int -> Int) { ... }
func foo(value: Double, block: (Int, Int) -> String) { ... }
```

```
func foo(value: Double, block: () -> Void) { ... }
func foo(value: Double, block: Int -> Int) { ... }
func foo(value: Double, block: (Int, Int) -> String) { ... }
```

@noescape

@noescapeself.

```
func foo(value: Double, block: () -> Void) { ... }
func foo(value: Double, block: Int -> Int) { ... }
func foo(value: Double, block: (Int, Int) -> String) { ... }
```

```
func foo(value: Double, block: () -> Void) { ... }
func foo(value: Double, block: Int -> Int) { ... }
func foo(value: Double, block: (Int, Int) -> String) { ... }
```

3

Swift 3@noescape。 。 Swift 3“@escaping”。

---

throwsrethrows

```
func foo(value: Double, block: () -> Void) { ... }
func foo(value: Double, block: Int -> Int) { ... }
func foo(value: Double, block: (Int, Int) -> String) { ... }
```

```
func foo(value: Double, block: () -> Void) { ... }
func foo(value: Double, block: Int -> Int) { ... }
func foo(value: Double, block: (Int, Int) -> String) { ... }
```

throw

```
func foo(value: Double, block: () -> Void) { ... }
func foo(value: Double, block: Int -> Int) { ... }
func foo(value: Double, block: (Int, Int) -> String) { ... }
```

rethrows

```
func foo(value: Double, block: () -> Void) { ... }
func foo(value: Double, block: Int -> Int) { ... }
func foo(value: Double, block: (Int, Int) -> String) { ... }
```

`rethrows` `map()` `filter()``indexOf()` 。

/

```
class MyClass {
    func sayHi() { print("Hello") }
    deinit { print("Goodbye") }
}
```

```
class MyClass {
    func sayHi() { print("Hello") }
    deinit { print("Goodbye") }
}
```

```
class MyClass {
    func sayHi() { print("Hello") }
    deinit { print("Goodbye") }
}
```

```
class MyClass {
    func sayHi() { print("Hello") }
    deinit { print("Goodbye") }
}
```

“ ”“Swift”“ ”。

。 。

```
class MyClass {
    func sayHi() { print("Hello") }
    deinit { print("Goodbye") }
}
```

。

3.0

```
func getData(urlString: String, callback: (result: NSData?) -> Void) {

    // Turn the URL string into an NSURLRequest.
    guard let url = NSURL(string: urlString) else { return }
    let request = NSURLRequest(URL: url)

    // Asynchronously fetch data from the given URL.
    let task = NSURLSession.sharedSession().dataTaskWithRequest(request) {(data: NSData?,
response: NSURLResponse?, error: NSError?) in

        // We now have the NSData response from the website.
        // We can get it "out" of the function by using the callback
        // that was passed to this function as a parameter.

        callback(result: data)
    }

    task.resume()
}
```

GUI。

3.0

```
func getData(urlString: String, callback: (result: NSData?) -> Void) {

    // Turn the URL string into an NSURLRequest.
    guard let url = NSURL(string: urlString) else { return }
    let request = NSURLRequest(URL: url)

    // Asynchronously fetch data from the given URL.
    let task = NSURLSession.sharedSession().dataTaskWithRequest(request) {(data: NSData?,
response: NSURLResponse?, error: NSError?) in

        // We now have the NSData response from the website.
        // We can get it "out" of the function by using the callback
        // that was passed to this function as a parameter.

        callback(result: data)
    }

    task.resume()
```

```
}
```

```
func getData(urlString: String, callback: (result: NSData?) -> Void) {

    // Turn the URL string into an NSURLRequest.
    guard let url = NSURL(string: urlString) else { return }
    let request = NSURLRequest(URL: url)

    // Asynchronously fetch data from the given URL.
    let task = NSURLSession.sharedSession().dataTaskWithRequest(request) {(data: NSData?,
response: NSURLResponse?, error: NSError?) in

        // We now have the NSData response from the website.
        // We can get it "out" of the function by using the callback
        // that was passed to this function as a parameter.

        callback(result: data)
    }

    task.resume()
}
```

URL`print("2. Fetched data")` 。

`typealias`。 。 “”。

```
public typealias ClosureType = (x: Int, y: Int) -> Int
```

typealias

```
public typealias ClosureType = (x: Int, y: Int) -> Int
```

https://riptutorial.com/zh-TW/swift/topic/262/

55:

Examples

。 。

```
var colors = Set<String>()
```

。

```
var colors = Set<String>()
```

```
var favoriteColors: Set = ["Red", "Blue", "Green"]
//favoriteColors = {"Blue", "Green", "Red"}
```

`insert(_:)`。

```
var favoriteColors: Set = ["Red", "Blue", "Green"]
//favoriteColors = {"Blue", "Green", "Red"}
```

`remove(_:)`。 nil。

```
var favoriteColors: Set = ["Red", "Blue", "Green"]
//favoriteColors = {"Blue", "Green", "Red"}
```

```
var favoriteColors: Set = ["Red", "Blue", "Green"]
//favoriteColors = {"Blue", "Green", "Red"}
```

`contains(_:)`。 true。

```
var favoriteColors: Set = ["Red", "Blue", "Green"]
//favoriteColors = {"Blue", "Green", "Red"}
```

`intersect(_:)`。

```
let favoriteColors: Set = ["Red", "Blue", "Green"]
let newColors: Set = ["Purple", "Orange", "Green"]

let intersect = favoriteColors.intersect(newColors) // a AND b
// intersect = {"Green"}
```

`union(_:)`。

```
let favoriteColors: Set = ["Red", "Blue", "Green"]
let newColors: Set = ["Purple", "Orange", "Green"]
```

```
let intersect = favoriteColors.intersect(newColors) // a AND b
// intersect = {"Green"}
```

“”。

`exclusiveOr(_:)`。

```
let favoriteColors: Set = ["Red", "Blue", "Green"]
let newColors: Set = ["Purple", "Orange", "Green"]

let intersect = favoriteColors.intersect(newColors) // a AND b
// intersect = {"Green"}
```

“Green”。

`subtract(_:)`。

```
let favoriteColors: Set = ["Red", "Blue", "Green"]
let newColors: Set = ["Purple", "Orange", "Green"]

let intersect = favoriteColors.intersect(newColors) // a AND b
// intersect = {"Green"}
```

“”。

Set

`Set Hashable`

```
struct Starship: Hashable {
    let name: String
    var hashValue: Int { return name.hashValue }
}

func ==(left:Starship, right: Starship) -> Bool {
    return left.name == right.name
}
```

`Set Starship(s)`

```
struct Starship: Hashable {
    let name: String
    var hashValue: Int { return name.hashValue }
}

func ==(left:Starship, right: Starship) -> Bool {
    return left.name == right.name
}
```

CountedSet

3.0

Swift 3`CountedSet``NSCountedSet` Objective-CSwift。

CountedSet。

```
let countedSet = CountedSet()
countedSet.add(1)
countedSet.add(1)
countedSet.add(1)
countedSet.add(2)

countedSet.count(for: 1) // 3
countedSet.count(for: 2) // 1
```

https://riptutorial.com/zh-TW/swift/topic/371/

56:

SwiftWWDC 2015 。

Swift 2 。

Examples

。

ViewModelUIViewController。

1. ViewModelUIViewController。
2. ViewModel。
3. 。

```
protocol ViewModelType {
   var title : String {get}
   func confirm()
}

class ViewModel : ViewModelType {
   let title : String

   init(title: String) {
       self.title = title
   }
   func confirm() { ... }
}

class ViewController : UIViewController {
   // We declare the viewModel property as an object conforming to the protocol
   // so we can swap the implementations without any friction.
   var viewModel : ViewModelType!
   @IBOutlet var titleLabel : UILabel!

   override func viewDidLoad() {
       super.viewDidLoad()
       titleLabel.text = viewModel.title
   }

   @IBAction func didTapOnButton(sender: UIButton) {
       viewModel.confirm()
   }
}

// With DI we setup the view controller and assign the view model.
// The view controller doesn't know the concrete class of the view model,
// but just relies on the declared interface on the protocol.
let viewController = //... Instantiate view controller
viewController.viewModel = ViewModel(title: "MyTitle")
```

1. ViewModel
2.

UIViewController。

3.

```
protocol ViewModelType {
   var title : String {get}
   func confirm()
}

class ViewModel : ViewModelType {
   let title : String

   init(title: String) {
       self.title = title
   }
   func confirm() { ... }
}

class ViewController : UIViewController {
   // We declare the viewModel property as an object conforming to the protocol
   // so we can swap the implementations without any friction.
   var viewModel : ViewModelType!
   @IBOutlet var titleLabel : UILabel!

   override func viewDidLoad() {
       super.viewDidLoad()
       titleLabel.text = viewModel.title
   }

   @IBAction func didTapOnButton(sender: UIButton) {
       viewModel.confirm()
   }
}

// With DI we setup the view controller and assign the view model.
// The view controller doesn't know the concrete class of the view model,
// but just relies on the declared interface on the protocol.
let viewController = //... Instantiate view controller
viewController.viewModel = ViewModel(title: "MyTitle")
```

Swift。

。

/。

```
protocol ItemData {

    var title: String { get }
    var description: String { get }
    var thumbnailURL: NSURL { get }
    var created: NSDate { get }
    var updated: NSDate { get }

}

protocol DisplayItem {

    func hasBeenUpdated() -> Bool
```

```
    func getFormattedTitle() -> String
    func getFormattedDescription() -> String

}

protocol GetAPIItemDataOperation {

    static func get(url: NSURL, completed: ([ItemData]) -> Void)
}
```

get。

```
protocol ItemData {

    var title: String { get }
    var description: String { get }
    var thumbnailURL: NSURL { get }
    var created: NSDate { get }
    var updated: NSDate { get }

}

protocol DisplayItem {

    func hasBeenUpdated() -> Bool
    func getFormattedTitle() -> String
    func getFormattedDescription() -> String

}

protocol GetAPIItemDataOperation {

    static func get(url: NSURL, completed: ([ItemData]) -> Void)
}
```

ItemData。

```
protocol ItemData {

    var title: String { get }
    var description: String { get }
    var thumbnailURL: NSURL { get }
    var created: NSDate { get }
    var updated: NSDate { get }

}

protocol DisplayItem {

    func hasBeenUpdated() -> Bool
    func getFormattedTitle() -> String
    func getFormattedDescription() -> String

}

protocol GetAPIItemDataOperation {

    static func get(url: NSURL, completed: ([ItemData]) -> Void)
```

```
}
```

item。

```
protocol ItemData {

    var title: String { get }
    var description: String { get }
    var thumbnailURL: NSURL { get }
    var created: NSDate { get }
    var updated: NSDate { get }

}

protocol DisplayItem {

    func hasBeenUpdated() -> Bool
    func getFormattedTitle() -> String
    func getFormattedDescription() -> String

}

protocol GetAPIItemDataOperation {

    static func get(url: NSURL, completed: ([ItemData]) -> Void)
}
```

get。

```
protocol ItemData {

    var title: String { get }
    var description: String { get }
    var thumbnailURL: NSURL { get }
    var created: NSDate { get }
    var updated: NSDate { get }

}

protocol DisplayItem {

    func hasBeenUpdated() -> Bool
    func getFormattedTitle() -> String
    func getFormattedDescription() -> String

}

protocol GetAPIItemDataOperation {

    static func get(url: NSURL, completed: ([ItemData]) -> Void)
}
```

。 。 APICore Data。

```
protocol ItemData {

    var title: String { get }
```

```
    var description: String { get }
    var thumbnailURL: NSURL { get }
    var created: NSDate { get }
    var updated: NSDate { get }

}

protocol DisplayItem {

    func hasBeenUpdated() -> Bool
    func getFormattedTitle() -> String
    func getFormattedDescription() -> String

}

protocol GetAPIItemDataOperation {

    static func get(url: NSURL, completed: ([ItemData]) -> Void)
}
```

Core DataDisplayItem。

```
protocol ItemData {

    var title: String { get }
    var description: String { get }
    var thumbnailURL: NSURL { get }
    var created: NSDate { get }
    var updated: NSDate { get }

}

protocol DisplayItem {

    func hasBeenUpdated() -> Bool
    func getFormattedTitle() -> String
    func getFormattedDescription() -> String

}

protocol GetAPIItemDataOperation {

    static func get(url: NSURL, completed: ([ItemData]) -> Void)
}
```

https://riptutorial.com/zh-TW/swift/topic/2502/

57:

init()。

Examples

```
class Dog {}
```

```
class Dog {}
```

AnimalDog。

。

```
class Dog {
    var name = ""
}

let firstDog = Dog()
firstDog.name = "Fido"

let otherDog = firstDog  // otherDog points to the same Dog instance
otherDog.name = "Rover"  // modifying otherDog also modifies firstDog

print(firstDog.name)  // prints "Rover"
```

。

```
class Dog {
    var name = ""
}

let firstDog = Dog()
firstDog.name = "Fido"

let otherDog = firstDog  // otherDog points to the same Dog instance
otherDog.name = "Rover"  // modifying otherDog also modifies firstDog

print(firstDog.name)  // prints "Rover"
```

===

```
class Dog {
    var name = ""
}

let firstDog = Dog()
firstDog.name = "Fido"

let otherDog = firstDog  // otherDog points to the same Dog instance
otherDog.name = "Rover"  // modifying otherDog also modifies firstDog
```

```
print(firstDog.name)  // prints "Rover"
```

- Dog namedogYearAge

```
class Dog {
    var name = ""
    var dogYearAge = 0
}
```

```
class Dog {
    var name = ""
    var dogYearAge = 0
}
```

```
class Dog {
    var name = ""
    var dogYearAge = 0
}
```

```
class Dog {
    var name = ""
    var dogYearAge = 0
}
```

Swift。 。

```
class Animal { ... }
class Pet { ... }

class Dog: Animal, Pet { ... } // This will result in a compiler error.
```

。 。

DEINIT

```
class ClassA {

    var timer: NSTimer!

    init() {
        // initialize timer
    }

    deinit {
        // code
        timer.invalidate()
    }
}
```

https://riptutorial.com/zh-TW/swift/topic/459/

58:

- `let name = json["name"] as? String ?? ""` //
- `let name = json["name"] as? String // Output: Optional("john")`
- `let name = rank as? Int // Output: Optional(1)`
- `let name = rank as? Int ?? 0 // Output: 1`
- `let name = dictionary as? [String: Any] ?? [:] // Output: ["name" : "john", "subjects": ["Maths", "Science", "English", "C Language"]]`

Examples

as? as! ◦

as?◦

◦

```
let value: Any = "John"

let name = value as? String
print(name) // prints Optional("John")

let age = value as? Double
print(age) // prints nil
```

as!◦

◦

```
let value: Any = "John"

let name = value as? String
print(name) // prints Optional("John")

let age = value as? Double
print(age) // prints nil
```

```
let value: Any = "John"

let name = value as? String
print(name) // prints Optional("John")

let age = value as? Double
print(age) // prints nil
```

switch

```
func checkType(_ value: Any) -> String {
    switch value {
```

```
    // The `is` operator can be used to check a type
    case is Double:
        return "value is a Double"

    // The `as` operator will cast. You do not need to use `as?` in a `switch`.
    case let string as String:
        return "value is the string: \(string)"

    default:
        return "value is something else"
    }

}

checkType("Cadena")  // "value is the string: Cadena"
checkType(6.28)      // "value is a Double"
checkType(UILabel()) // "value is something else"
```

as。 。

```
let name = "Ringo"
let value = string as Any  // `value` is of type `Any` now
```

。

```
class Rat {
    var color = "white"
}

class PetRat: Rat {
    var name = "Spot"
}

func nameOfRat(□: Rat) -> String {
    guard let petRat = (□ as? PetRat) else {
        return "No name"
    }

    return petRat.name
}

let noName = Rat()
let spot = PetRat()

print(nameOfRat(noName))
print(nameOfRat(spot))
```

Swift

。

Swiftisas。 。

- as
- - -
- nil
- -

IntFloat -

```
let numbers = "888.00"
let intValue = NSString(string: numbers).integerValue
print(intValue) // Output - 888


let numbers = "888.00"
let floatValue = NSString(string: numbers).floatValue
print(floatValue) // Output : 888.0
```

```
let numbers = "888.00"
let intValue = NSString(string: numbers).integerValue
print(intValue) // Output - 888


let numbers = "888.00"
let floatValue = NSString(string: numbers).floatValue
print(floatValue) // Output : 888.0
```

```
let numbers = "888.00"
let intValue = NSString(string: numbers).integerValue
print(intValue) // Output - 888


let numbers = "888.00"
let floatValue = NSString(string: numbers).floatValue
print(floatValue) // Output : 888.0
```

```
let numbers = "888.00"
let intValue = NSString(string: numbers).integerValue
print(intValue) // Output - 888


let numbers = "888.00"
let floatValue = NSString(string: numbers).floatValue
```

```
print(floatValue) // Output : 888.0
```

String

```
let numbers = "888.00"
let intValue = NSString(string: numbers).integerValue
print(intValue) // Output - 888


let numbers = "888.00"
let floatValue = NSString(string: numbers).floatValue
print(floatValue) // Output : 888.0
```

StringInt

```
let numbers = "888.00"
let intValue = NSString(string: numbers).integerValue
print(intValue) // Output - 888


let numbers = "888.00"
let floatValue = NSString(string: numbers).floatValue
print(floatValue) // Output : 888.0
```

JSON

```
let numbers = "888.00"
let intValue = NSString(string: numbers).integerValue
print(intValue) // Output - 888


let numbers = "888.00"
let floatValue = NSString(string: numbers).floatValue
print(floatValue) // Output : 888.0
```

Optional JSON

```
let numbers = "888.00"
let intValue = NSString(string: numbers).integerValue
print(intValue) // Output - 888


let numbers = "888.00"
let floatValue = NSString(string: numbers).floatValue
print(floatValue) // Output : 888.0
```

JSON

```
    let numbers = "888.00"
    let intValue = NSString(string: numbers).integerValue
    print(intValue) // Output - 888


    let numbers = "888.00"
    let floatValue = NSString(string: numbers).floatValue
    print(floatValue) // Output : 888.0
```

```
    let numbers = "888.00"
    let intValue = NSString(string: numbers).integerValue
    print(intValue) // Output - 888


    let numbers = "888.00"
    let floatValue = NSString(string: numbers).floatValue
    print(floatValue) // Output : 888.0
```

Empty Dictionary

https://riptutorial.com/zh-TW/swift/topic/3082/

59:

Swift Swift.org API 。 The Official raywenderlich.com Swift Style Guide。

Examples

。

```
extension List {
    public mutating func remove(at position: Index) -> Element {
        // implementation
    }
}
```

```
extension List {
    public mutating func remove(at position: Index) -> Element {
        // implementation
    }
}
```

。 `list.remove(42)` 42Element42Element。

。

```
extension List {
    public mutating func remove(at position: Index) -> Element {
        // implementation
    }
}
```

`list.removeElement(someObject)` 。 `someObject`。

```
extension List {
    public mutating func remove(at position: Index) -> Element {
        // implementation
    }
}
```

`list.remove(someObject)` 。

。

Iterator“Type”。 IteratorType

。

```
extension List {
    public mutating func remove(at position: Index) -> Element {
        // implementation
    }
}
```

`object.addObserverselfforKeyPathpath

```
extension List {
    public mutating func remove(at position: Index) -> Element {
        // implementation
    }
}
```

```
object.add(self, for: path)
```

。

```
list.insert(element, at: index)
```

```
list.insert(element, at: index)
```

`make`。

```
list.insert(element, at: index)
```

。

```
list.insert(element, at: index)
```

```
list.insert(element, at: index)
```

- form-。
- -ing-ed。

```
list.insert(element, at: index)
```

```
list.insert(element, at: index)
```

。

```
list.insert(element, at: index)
```

- 。
- -able -ible-ing。

```
list.insert(element, at: index)
```

。

```
list.insert(element, at: index)
```

。

```
protocol Collection {}
struct String {}
class UIView {}
struct Int {}
enum Color {}
```

。

```
protocol Collection {}
struct String {}
class UIView {}
struct Int {}
enum Color {}
```

。 /。

```
protocol Collection {}
struct String {}
class UIView {}
```

```
struct Int {}
enum Color {}
```

```
protocol Collection {}
struct String {}
class UIView {}
struct Int {}
enum Color {}
```

URLID。 。

```
protocol Collection {}
struct String {}
class UIView {}
struct Int {}
enum Color {}
```

https://riptutorial.com/zh-TW/swift/topic/3031/

60:

Examples

Swift。 `operatoroperator` 。

。

1. `prefix` `infixpostfix`。 `prefixpostfix`。 urnary`83++` ** 。 `infix2+3` 。
2. 。 `?` ... `:` Swift 2.x100。 Swift。
3. 。 。

3.0

Swift 3.0。 。

。 `return`。

Swift。

```
import Foundation

infix operator ** { associativity left precedence 170 }

func ** (num: Double, power: Double) -> Double{
    return pow(num, power)
}
```

`infix**9**2` 。 `3**3**2(3**3)**2` 。 `170`Swift`3+2**419 **`。

3.0

```
import Foundation

infix operator ** { associativity left precedence 170 }

func ** (num: Double, power: Double) -> Double{
    return pow(num, power)
}
```

Swift 3.0BitwiseShiftPrecedence`<<` `>>`。

**Swift 3。

+

Swift `++=`。

```
// Combines two dictionaries together. If both dictionaries contain
// the same key, the value of the right hand side dictionary is used.
func +<K, V>(lhs: [K : V], rhs: [K : V]) -> [K : V] {
    var combined = lhs
    for (key, value) in rhs {
        combined[key] = value
    }
    return combined
}

// The mutable variant of the + overload, allowing a dictionary
// to be appended to 'in-place'.
func +=<K, V>(inout lhs: [K : V], rhs: [K : V]) {
    for (key, value) in rhs {
        lhs[key] = value
    }
}
```

3.0

Swift 3 `inout`。

```
// Combines two dictionaries together. If both dictionaries contain
// the same key, the value of the right hand side dictionary is used.
func +<K, V>(lhs: [K : V], rhs: [K : V]) -> [K : V] {
    var combined = lhs
    for (key, value) in rhs {
        combined[key] = value
    }
    return combined
}

// The mutable variant of the + overload, allowing a dictionary
// to be appended to 'in-place'.
func +=<K, V>(inout lhs: [K : V], rhs: [K : V]) {
    for (key, value) in rhs {
        lhs[key] = value
    }
}
```

```
// Combines two dictionaries together. If both dictionaries contain
// the same key, the value of the right hand side dictionary is used.
func +<K, V>(lhs: [K : V], rhs: [K : V]) -> [K : V] {
    var combined = lhs
    for (key, value) in rhs {
        combined[key] = value
    }
    return combined
}

// The mutable variant of the + overload, allowing a dictionary
// to be appended to 'in-place'.
func +=<K, V>(inout lhs: [K : V], rhs: [K : V]) {
    for (key, value) in rhs {
        lhs[key] = value
    }
}
```

CGSize

```
func *(lhs: CGFloat, rhs: CGSize) -> CGSize{
    let height = lhs*rhs.height
    let width = lhs*rhs.width
    return CGSize(width: width, height: height)
}
```

```
func *(lhs: CGFloat, rhs: CGSize) -> CGSize{
    let height = lhs*rhs.height
    let width = lhs*rhs.width
    return CGSize(width: width, height: height)
}
```

```
func *(lhs: CGFloat, rhs: CGSize) -> CGSize{
    let height = lhs*rhs.height
    let width = lhs*rhs.width
    return CGSize(width: width, height: height)
}
```

```
func *(lhs: CGFloat, rhs: CGSize) -> CGSize{
    let height = lhs*rhs.height
    let width = lhs*rhs.width
    return CGSize(width: width, height: height)
}
```

。

```
func *(lhs: CGFloat, rhs: CGSize) -> CGSize{
    let height = lhs*rhs.height
    let width = lhs*rhs.width
    return CGSize(width: width, height: height)
}
```

Swift Bitwise。 `0b0b110`1106。 10。

NOT `~`

```
var number: UInt8 = 0b01101100
let newNumber = ~number
// newNumber is equal to 0b01101100
```

。 `UInt8`8。 `0b01101100`UInt01

```
var number: UInt8 = 0b01101100
let newNumber = ~number
// newNumber is equal to 0b01101100
```

- 0 - > 1
- 1 - > 0

AND `&`

```
var number: UInt8 = 0b01101100
let newNumber = ~number
// newNumber is equal to 0b01101100
```

&11。

- 00 - > 0
- 01 - > 0
- 11 - > 1

OR |

```
var number: UInt8 = 0b01101100
let newNumber = ~number
// newNumber is equal to 0b01101100
```

|11。

- 0 | 0 - > 0
- 0 | 1 - > 1
- 1 | 1 - > 1

^

```
var number: UInt8 = 0b01101100
let newNumber = ~number
// newNumber is equal to 0b01101100
```

1。

- 0 ^ 0 - > 0
- 0 ^ 1 - > 1
- 1 ^ 1 - > 0

。

。

。 。

Swift+ -*。 &+ &-&*。

```
var almostTooLarge = Int.max
almostTooLarge + 1 // not allowed
almostTooLarge &+ 1 // allowed, but result will be the value of Int.min
```

Swift

。

	≥3.0		
`.`		∞	
`? ! ++ -- [] () {}`			
`! ~ + - ++ --`			
~> swift≤2.3		255	
`<< >>`	BitwiseShiftPrecedence	160	
`* / % & &*`	MultiplicationPrecedence	150	
`+ - | ^ &+ &-`	AdditionPrecedence	140	
`... ..<`	RangeFormationPrecedence	135	
`is as as? as!`	CastingPrecedence	132	
`??`	NilCoalescingPrecedence	131	
`< <= > >= == != === !== ~=`	ComparisonPrecedence	130	
`&&`	LogicalConjunctionPrecedence	120	
`||`	LogicalDisjunctionPrecedence	110	
	DefaultPrecedence *		
`? ..... :`	TernaryPrecedence	100	
`= += -= *= /= %= <<= >>= &= |= ^=`	AssignmentPrecedence	90	
`->`	FunctionArrowPrecedence		

3.0

- `DefaultPrecedenceTernaryPrecedence` ◦ ◦

- AppleAPI
- GitHub

https://riptutorial.com/zh-TW/swift/topic/1048/

S. No		Contributors
1	Swift	Ahmad F, Anas, andy, Cailean Wilkinson, Claw, Community, esthepiking, Ferenc Kiss, Jim, jtbandes, Luca Angeletti, Luca Angioloni, Moritz, nmnsud, Seyyed Parsa Neshaei, sudo, Sunil Prajapati, Tanner, user3581248
2		Tommie C.
3	AES	Matt, Stephen Leppik, zaph
4	KituraSwift HTTP	Fangming Ning
5	OptionSet	4444, Alessandro
6	PBKDF2	BUZZE, zaph
7	RxSwift	Alexander Olferuk, FelixSFD, imagngames, Moritz, Victor Sigler
8	Swift Advance	DarkDust, Sagar Thummar
9	Swift NSRegularExpression	Echelon, Hady Nourallah, ThrowingSpoon
10	Swift	Echelon, Luca Angeletti, Luke, Matthew Seaman, Shijing Lv
11	Swift	Moritz
12	Swift	Austin Conlon, Bohdan Savych, Hady Nourallah, SteBra, Stephen Leppik, Tommie C.
13	Typealias	Bartłomiej Semańczyk, Caleb Kleveter, D4ttatraya, Moritz
14		Adda_25, Ahmad F, FelixSFD, JAL, LukeSideWalker, M_G, Matthew Seaman, Palle, Rob, Santa Claus
15	Swift	Kumar Vivek Mitra
16	CObjective-C	4444, Accepted Answer, jtbandes, Mark
17		Bear with me, JPetric
18		Accepted Answer, BaSha, Caleb Kleveter, JAL, Jason Sturges, Jojodmo, kabiroberai, LopSae, Luca Angeletti, Moritz, Nathan Kellert, Rick Pasveer, Ronald Martin, tktsubota
19		Accepted Answer, Daniel Firsht, jtbandes, Marc Gravell, Moritz,

		Palle, Tricertops
20		Brduca, FelixSFD, rashfmnb, Santa Claus, Vinupriya Arivazhagan
21		Ajith R Nayak, Andy Ibanez, Caleb Kleveter, jtbandes, Kote, Luca Angeletti, Matt Le Fleur, Nikita Kurtin, noor, ntoonio, Saagar Jha, SKOOP, Stephen Schaub, ThrowingSpoon, tktsubota, ZGski
22		Accepted Answer, Ash Furrow, Cory Wilhite, Dalija Prasnikar, esthepiking, Hamish, iBelieve, Igor Bidiniuc, Jason Sturges, Jojodmo, jtbandes, Luca D'Alberti, Matt, matt.baranowski, Matthew Seaman, Oleg Danu, Rahul, SeanRobinson159, SKOOP, Tim Vermeulen, tktsubota, Undo, Victor Sigler
23		Asdrubal, LopSae, Sajjon
24		Matt
25		dasdom, Diogo Antunes, egor.zhdan, iOSDevCenter, Jason Bourne, Kirit Modi, Koushik, Magisch, Moritz, RamenChef, Saagar Jha, sasquatch, Suneet Tipirneni, That lazy iOS Guy , ThrowingSpoon
26		Akshit Soota, Andrea Antonioni, antonio081014, AstroCB, Caleb Kleveter, Carpsen90, egor.zhdan, Feldur, Franck Dernoncourt, Govind Rai, Greg, Guilherme Torres Castro, Hamish, HariKrishnan.P, HeMet, JAL, Jason Sturges, Jojodmo, jtbandes, kabiroberai, Kirit Modi, Kyle KIM, Lope, LopSae, Luca Angeletti, LukeSideWalker, Magisch, Mahmoud Adam, Matt, Matthew Seaman, Max Desiatov, maxkonovalov, Moritz, Nate Cook, Nikolai Ruhe, Panda, Patrick, pixatlazaki, QoP, sdasdadas, Shanmugaraja G, shim, solidcell, Sunil Sharma, Suragch, taylor swift, The_Curry_Man, ThrowingSpoon, user3480295, Victor Sigler, Vinupriya Arivazhagan, WMios
27		Maysam, Moritz
28		zaph
29		Andreas, jtbandes, Kevin, pableiros
30		Palle
31	StringUIImage	RubberDucky4444
32		Caleb Kleveter, D31, Efraim Weiss, Fred Faust, Hamish, Idan, Irfan, Jeff Lewis, Luca Angeletti, Moritz, Mr. Xcoder, Saagar Jha, Santa Claus, WMios, xoudini

33		Matthew Seaman
34		Brduca, David, Esqarrouth, Jojodmo, jtbandes, Luca Angeletti, Moritz, rigdonmr
35		Arsen, jtbandes, Suragch, WMios, ZGski
36		BaSha, Ben Trengrove, D4ttatraya, DarkDust, Hamish, jtbandes, Kevin, Luca Angeletti, Moritz, Moriya, nathan, pableiros, Palle, Saagar Jha, Stephen Leppik, ThrowingSpoon, tomahh, toofani, vacawama, Vladimir Nul
37		Abdul Yasin, Martin Delille, Moritz, Rashwan L
38		JAL, Noam, Umberto Raimondi
39		Alex Popov, Anh Pham, Avi, Caleb Kleveter, Diogo Antunes, Fantattitude, fredpi, Hamish, Jason Sturges, Jojodmo, jtbandes , juanjo, Justin Whitney, Matt, Matthew Seaman, Nathan Kellert , Nick Podratz, Nikolai Ruhe, SeanRobinson159, shannoga, user3480295
40		AK1, atxe, Brduca, Community, Dalija Prasnikar, DarkDust, Hamish, jtbandes, ThaNerd, Thomas Gerot, tktsubota, toofani, torinpitchers
41		Andrey Gordeev, DarkDust, FelixSFD, Glenn R. Fisher, Hamish, Jojodmo, Kent Liau, Luca D'Alberti, Suneet Tipirneni, Ven, xoudini
42	Swift	Adam Bardon, D4ttatraya, DanHabib, jglasse, Moritz, paper1111, RamenChef
43		Fattie, JAL
44		Accepted Answer, AK1, Diogo Antunes, fredpi, Josh Brown, Kevin, Luca Angeletti, Marcus Rossel, Moritz, pbush25, Rob Napier, SamG
45		Viktor Gardart
46		4444, Asdrubal, FelixSFD
47	-	Ahmad F, AMAN77, Brduca, Dalija Prasnikar, Ian Rahman, Moritz, SeanRobinson159, SimpleBeat, Sơn Đỗ Đình Thy, Stephen Leppik, Thorax, Tommie C.
48	-	Ian Rahman
49		Christopher Oezbek, FelixSFD, Jojodmo, Luke, Santa Claus,

		tktsubota
50		Anand Nimje, Andrey Gordeev, Arnaud, Caleb Kleveter, Hamish, Ian Rahman, iwillnot, Jason Sturges, Jojodmo, juanjo, Kevin, Michaël Azevedo, Moritz, Nathan Kellert, Paulw11, shannoga, SKOOP, Tanner, tktsubota, Tommie C.
51		Anil Varghese, cpimhoff, egor.zhdan, Jason Bourne, jtbandes, Mehul Sojitra, Moritz, Tom Magnusson
52		Ajwhiteway, AK1, Duncan C, elprl, Harshal Bhavsar, joan, Josh Brown, Luca Angeletti, Moritz, Santa Claus, ThrowingSpoon
53	JSON	Cyril Ivar Garcia, Ethan Kay, Glenn R. Fisher, Ian Rahman, infl3x, Jack C, Jason Sturges, jtbandes, Leo Dabus, lostAtSeaJoshua, Luca D'Alberti, maxkonovalov, Moritz, nstefan, Steffen D. Sommer, Stephen Leppik, toofani
54		ctietze, Duncan C, Hamish, Jojodmo, jtbandes, LopSae, Matthew Seaman, Moritz, Timothy Rascher, Tom Magnusson
55		Community, Dalija Prasnikar, Luca Angeletti, Moritz, Steve Moser
56		Alessandro Orrù, Fred Faust, kabiroberai, Krzysztof Romanowski
57		Dalija Prasnikar, esthepiking, FelixSFD, jtbandes, Luca Angeletti, Matt, Ryan H., tktsubota, Tommie C., Zack
58		Anand Nimje, andyvn22, godisgood4, LopSae, Nick Podratz
59		Grimxn, Moritz, Palle, Ryan H.
60		avismara, egor.zhdan, Fluidity, Hamish, Intentss, JAL, jtbandes , kennytm, Matthew Seaman, orccrusher99, tharkay