免費電子書

學習

Swift Language

Free unaffiliated eBook created from
Stack Overflow contributors.

#swift

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: swift-language

It is an unofficial and free Swift Language ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official Swift Language.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

1: 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 ..

```
$ swift hello.swift
Hello, world!
```

  。

- swiftc

```
$ swiftc hello.swift
```

hello。 ./ 。

```
$ ./hello
Hello, world!
```

- swift REPLRead-Eval-Print-Loopswift

```
func greet(name: String, surname: String) {
    print("Greetings \(name) \(surname)")
}

let myName = "Homer"
let mySurname = "Simpson"

greet(name: myName, surname: mySurname)
```

  - 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 mySurnamevalues "Homer" "Simpson" 。
  - greet(name: myName, surname: mySurname) - myName mySurname。

REPL

```
$  swift
Welcome to Apple Swift. Type :help for assistance.
  1> func greet(name: String, surname: String) {
```

```
  2.     print("Greetings \(name) \(surname)")
  3. }
  4>
  5> let myName = "Homer"
myName: String = "Homer"
  6> let mySurname = "Simpson"
mySurname: String = "Simpson"
  7> greet(name: myName, surname: mySurname)
Greetings Homer Simpson
  8> ^D
```

CTRL + DREPL。

Swift

。

Swift。 macOS/ Library / Developer / Toolchains。

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

Linuxclang

```
$ sudo apt-get install clang
```

SwiftSwift

```
$ export PATH=/path/to/Swift/usr/bin:"${PATH}"
```

Swift

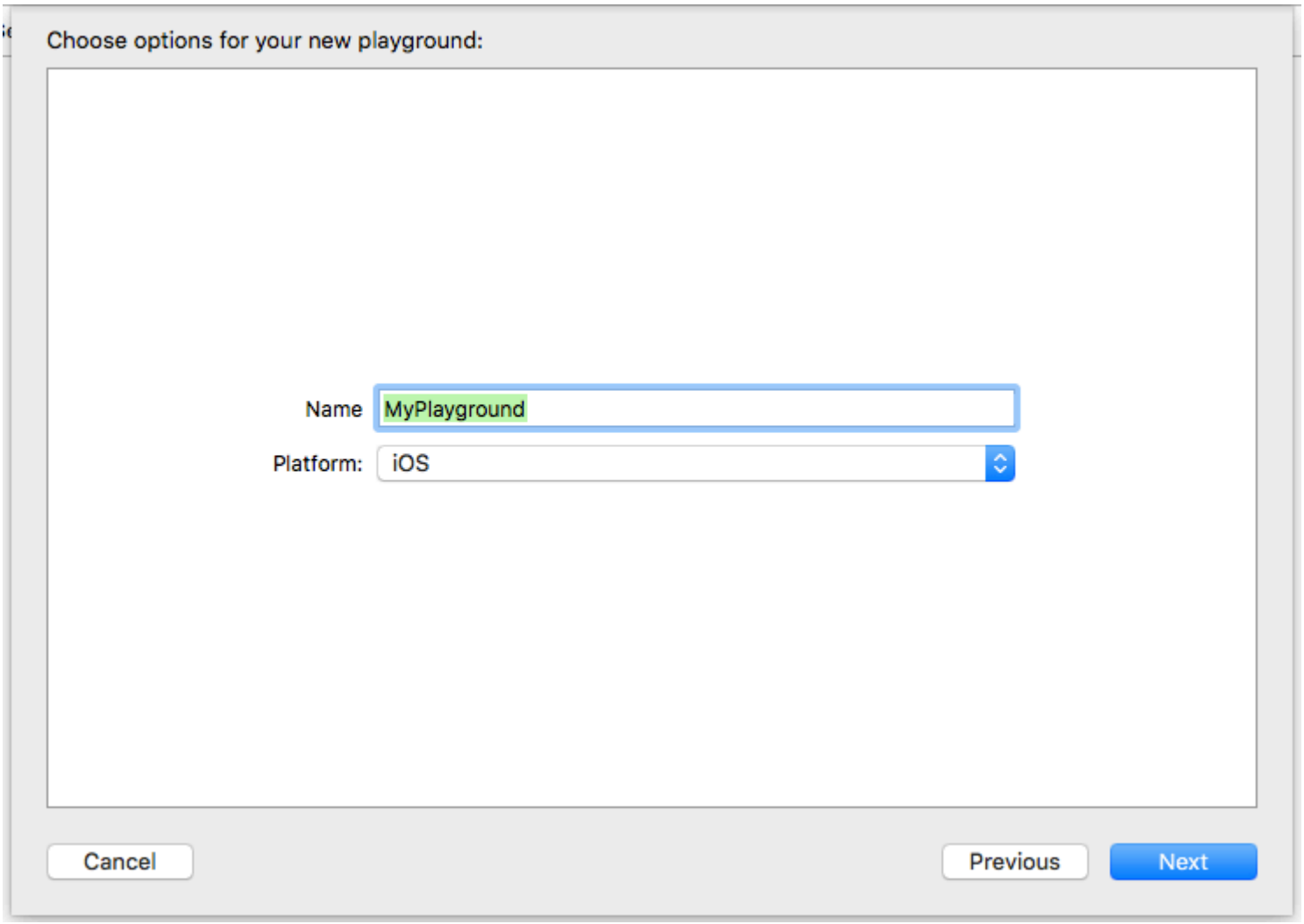
```
$ swift --version
```

MacSwift

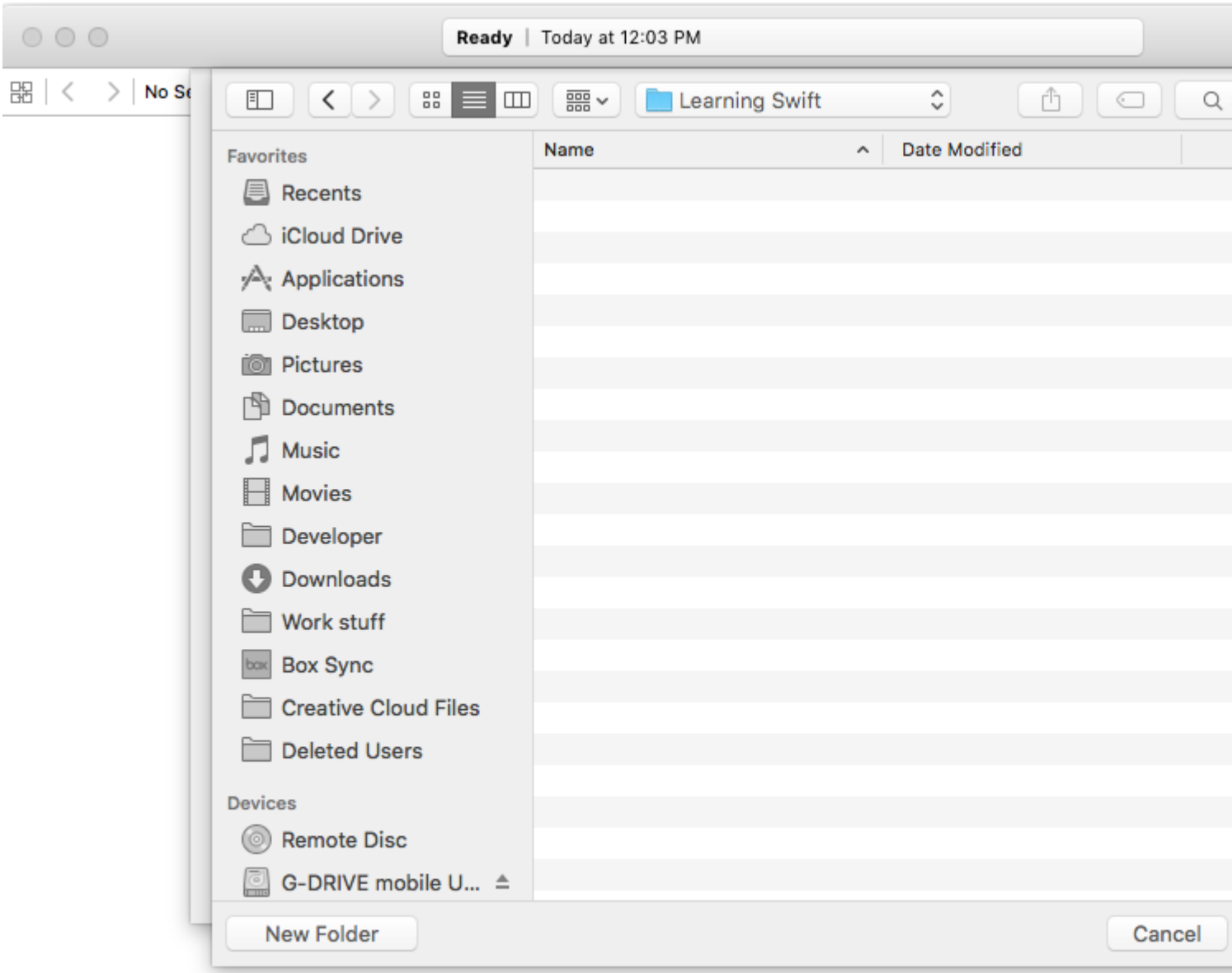
MacMac App StoreXcode。

Xcode**Playground**

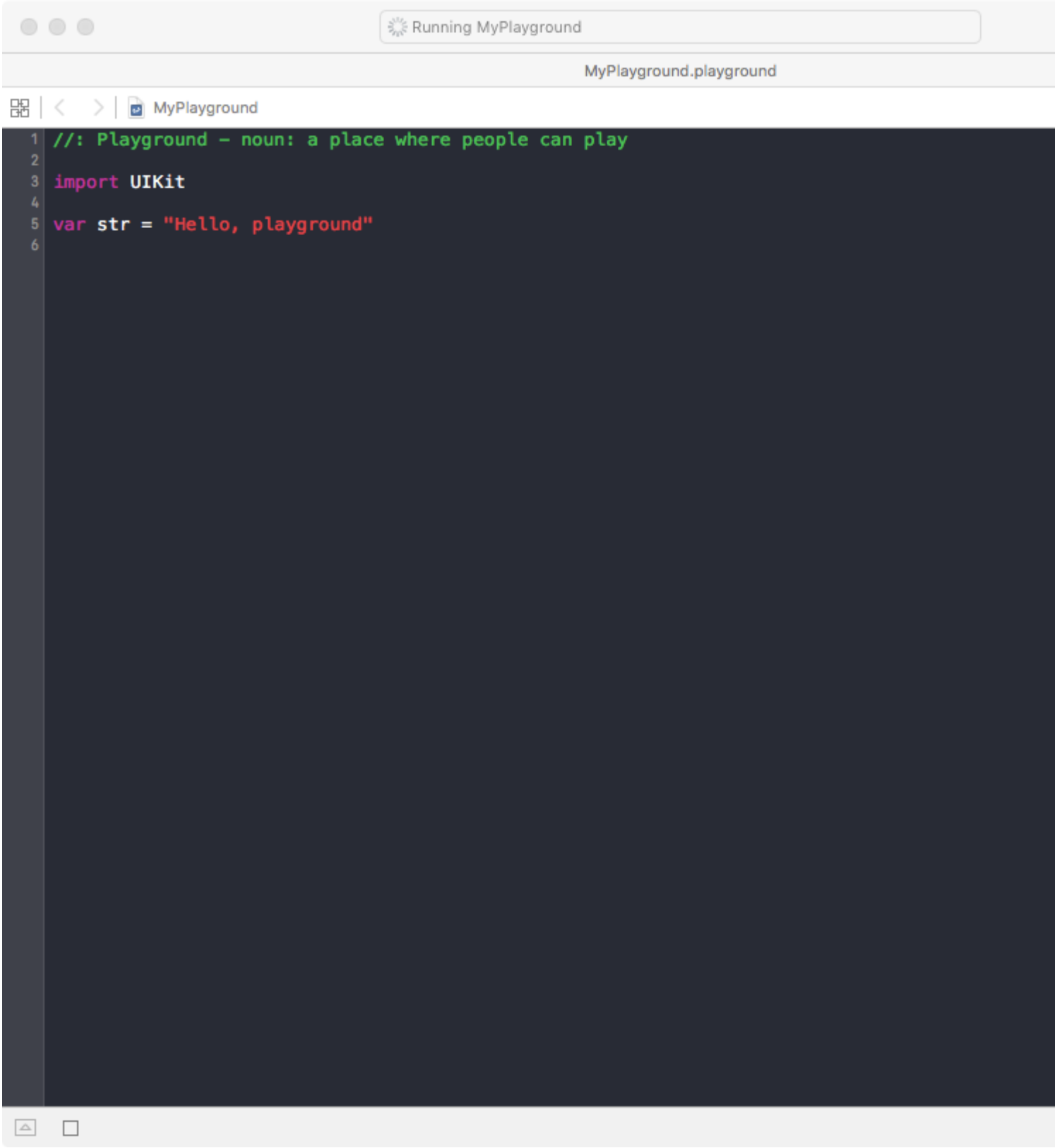
Playground`MyPlayground` “ ”



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

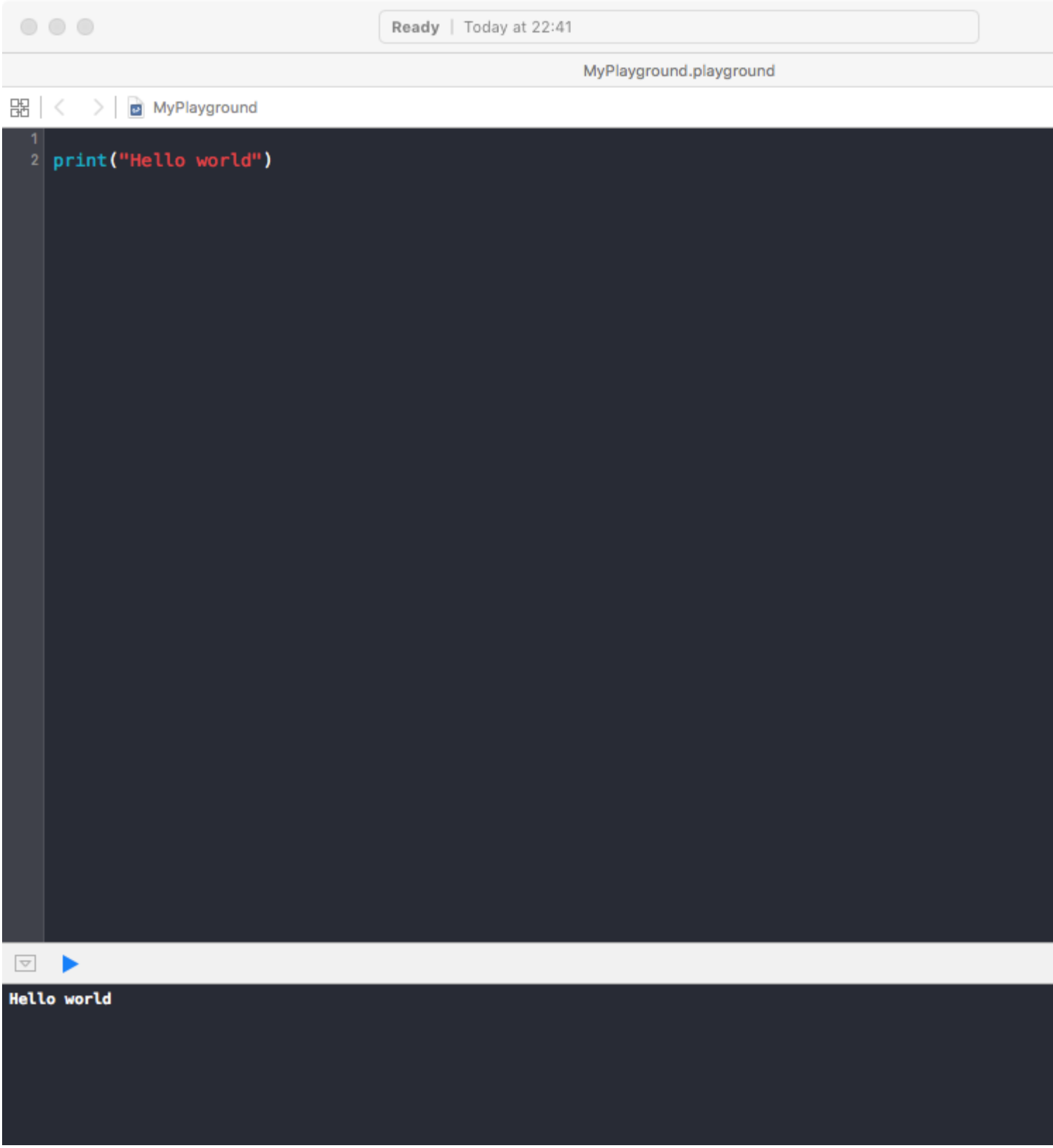


Playground + cmd + Y 。

Playground

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

“Hello world”“Hello world \ n”



Swift

iPadSwift Playgrounds

Swift PlaygroundsSwift。

1-App StoreiPadSwift Playgrounds 。


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

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。

```
import Foundation

let arr = [1,5,7,8]

let pointer = UnsafeMutablePointer<[Int]>.allocate(capacity: 4)
pointer.initialize(to: arr)

let x = pointer.pointee[3]
```

```
print(x)

pointer.deinitialize()
pointer.deallocate(capacity: 4)

class A {
  var x: String?

  convenience init (_ x: String) {
    self.init()
    self.x = x
  }

  func description() -> String {
    return x ?? ""
  }
}


let arr2 = [A("OK"), A("OK 2")]
let pointer2 = UnsafeMutablePointer<[A]>.allocate(capacity: 2)
pointer2.initialize(to: arr2)

pointer2.pointee
let y = pointer2.pointee[1]

print(y)

pointer2.deinitialize()
pointer2.deallocate(capacity: 2)
```

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`

```
let validUTF8: [CChar] = [67, 97, 102, -61, -87, 0]
     validUTF8.withUnsafeBufferPointer { ptr in
         let s = String(validatingUTF8: ptr.baseAddress!)
         print(s as Any)
     }
     // Prints "Optional(Café)"

     let invalidUTF8: [CChar] = [67, 97, 102, -61, 0]
     invalidUTF8.withUnsafeBufferPointer { ptr in
      let s = String(validatingUTF8: ptr.baseAddress!)
      print(s as Any)
```

```
    }
// Prints "nil"
```

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;
}
```

```
let clearData = "clearData0123456".data(using:String.Encoding.utf8)!
let keyData   = "keyData890123456".data(using:String.Encoding.utf8)!
print("clearData:   \(clearData as NSData)")
print("keyData:     \(keyData as NSData)")

var cryptData :Data?
do {
    cryptData = try aesCBCEncrypt(data:clearData, keyData:keyData)
    print("cryptData:   \(cryptData! as NSData)")
}
catch (let status) {
    print("Error aesCBCEncrypt: \(status)")
}

let decryptData :Data?
do {
    let decryptData = try aesCBCDecrypt(data:cryptData!, keyData:keyData)
    print("decryptData: \(decryptData! as NSData)")
}
catch (let status) {
    print("Error aesCBCDecrypt: \(status)")
}
```

```
clearData:   <636c6561 72446174 61303132 33343536>
keyData:     <6b657944 61746138 39303132 33343536>
cryptData:   <92c57393 f454d959 5a4d158f 6e1cd3e7 77986ee9 b2970f49 2bafcf1a 8ee9d51a bde49c31
d7780256 71837a61 60fa4be0>
decryptData: <636c6561 72446174 61303132 33343536>
```

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;
}
```

```
let clearData = toData("clearData0123456")
let keyData   = toData("keyData890123456")

print("clearData:   \(toHex(clearData))")
print("keyData:     \(toHex(keyData))")
let cryptData = aesCBC128Encrypt(data:clearData, keyData:keyData)!
print("cryptData:   \(toHex(cryptData))")
let decryptData = aesCBC128Decrypt(data:cryptData, keyData:keyData)!
print("decryptData: \(toHex(decryptData))")
```

```
clearData:   <636c6561 72446174 61303132 33343536>
keyData:     <6b657944 61746138 39303132 33343536>
cryptData:   <9fce4323 830e3734 93dd93bf e464f72a a653a3a5 2c40d5ea e90c1017 958750a7 ff094c53
6a81b458 b1fbd6d4 1f583298>
decryptData: <636c6561 72446174 61303132 33343536>
```

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 Kitura
import Foundation
import HeliumLogger

HeliumLogger.use()
```

RouterHTTP。GET*Hello world* post。

```
let router = Router()

router.get("/get") {
    request, response, next in
    response.send("Hello, World!")
    next()
}

router.post("/post") {
    request, response, next in
    var string: String?
    do{
        string = try request.readString()

    } catch let error {
        string = error.localizedDescription
    }
    response.send("Value \(string!) received.")
    next()
}
```

```
let port = 8080
```

HTTP

```
Kitura.addHTTPServer(onPort: port, with: router)
Kitura.run()
```

Package.swiftResources。 。 SwiftPackage.swiftPackagesmain.swift

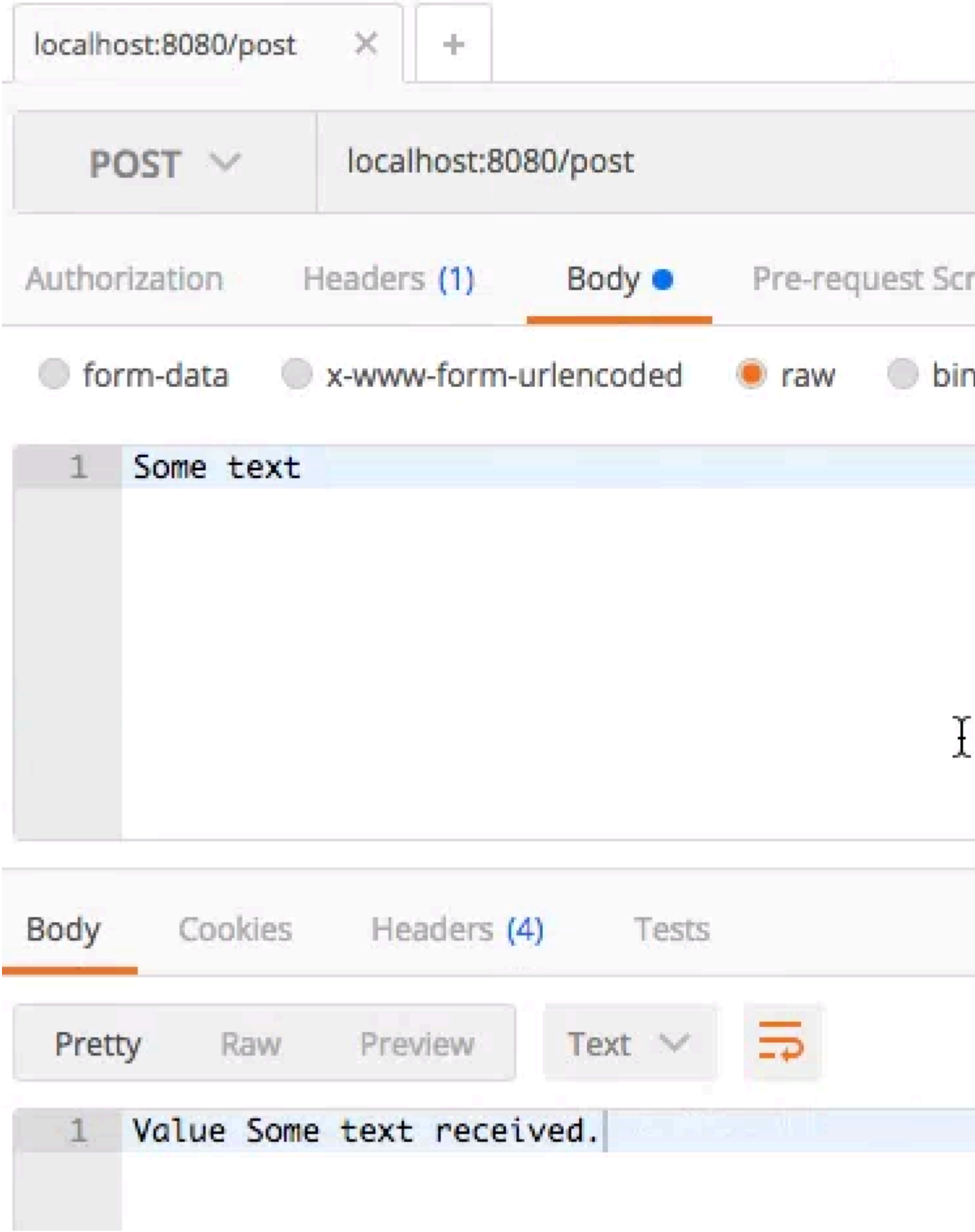
```
swift build
```

。 。

```
.build/debug/kitura-helloworld
```

`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
}
```

```
let password     = "password"
//let salt       = "saltData".data(using: String.Encoding.utf8)!
let salt         = Data(bytes: [0x73, 0x61, 0x6c, 0x74, 0x44, 0x61, 0x74, 0x61])
let keyByteCount = 16
let rounds       = 100000

let derivedKey = pbkdf2SHA1(password:password, salt:salt, keyByteCount:keyByteCount,
rounds:rounds)
print("derivedKey (SHA1): \(derivedKey! as NSData)")
```

```
derivedKey (SHA1): <6b9d4fa3 0385d128 f6d196ee 3f1d6dbf>
```

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
}
```

```
let password = "password"
// let salt  = [UInt8]("saltData".utf8)
let salt     = [UInt8]([0x73, 0x61, 0x6c, 0x74, 0x44, 0x61, 0x74, 0x61])
let rounds   = 100_000
let keyCount = 16

let derivedKey = pbkdf2SHA1(password, salt:salt, keyCount:keyCount, rounds:rounds)
print("derivedKey (SHA1):   \(NSData(bytes:derivedKey!, length:derivedKey!.count))")
```

```
derivedKey (SHA1):   <6b9d4fa3 0385d128 f6d196ee 3f1d6dbf>
```

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
}
```

```
let saltData       = [UInt8]([0x73, 0x61, 0x6c, 0x74, 0x44, 0x61, 0x74, 0x61])
let passwordString = "password"
let delayMsec      = 100

let rounds = pbkdf2SHA1Calibrate(passwordString, salt:saltData, msec:delayMsec)
print("For \(delayMsec) msec delay, rounds: \(rounds)")
```

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
}
```

```
let saltData       = Data(bytes: [0x73, 0x61, 0x6c, 0x74, 0x44, 0x61, 0x74, 0x61])
let passwordString = "password"
let delayMsec      = 100

let rounds = pbkdf2SHA1Calibrate(password:passwordString, salt:saltData, msec:delayMsec)
print("For \(delayMsec) msec delay, rounds: \(rounds)")
```

```
For 100 msec delay, rounds: 93457
```

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。

```
--X->
```

.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。

```
Observable.just(12).subscribe {
    print($0)
}
```

```
.Next(12)
.Completed()
```

.NextsubscribeNext

```
Observable.just(12).subscribeNext {
    print($0) // prints "12" now
}
```

```
Observable.of(1,2,3,4,5).subscribeNext {
    print($0)
}
// 1
// 2
// 3
// 4
// 5

// I can represent existing data types as Observables also:
[1,2,3,4,5].asObservable().subscribeNext {
    print($0)
}
// result is the same as before.
```

Observable。 Observable<SomeResultType>。

```
Observable.create { observer in    // create an Observable ...
    MyNetworkService.doSomeWorkWithCompletion { (result, error) in
        if let e = error {
            observer.onError(e)    // ..that either holds an error
        } else {
            observer.onNext(result) // ..or emits the data
            observer.onCompleted()  // ..and terminates successfully.
        }
    }
    return NopDisposable.instance // here you can manually free any resources
                                  //in case if this observable is being disposed.
}
```

。

addDisposableTodisposeBagdispose。 。

。

1. disposeBagaddDisposableTo。
2. takeUntil。

DisposeBag。

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

DisposeBag *RxSwift*NSObject + Rx。

```
Observable.just(1).subscribeNext {
    print($0)
```

```
}.addDisposableTo(rx_disposeBag)
```

selftakeUntil(rx_deallocated)

```
let _ = sequence
    .takeUntil(rx_deallocated)
    .subscribe {
        print($0)
    }
```

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

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

UITableViewUICollectionView

```
viewModel
.rows
.bindTo(resultsTableView.rx_itemsWithCellIdentifier("WikipediaSearchCell", cellType:
WikipediaSearchCell.self)) { (_, viewModel, cell) in
    cell.title = viewModel.title
    cell.url = viewModel.url
}
.addDisposableTo(disposeBag)
```

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

```
----()-----------------------()----->
```

◦ withLatestFromtextFieldwithLatestFrom ◦

```
button.rx_tap.withLatestFrom(textField.rx_text)

----("")--------------------("123")--->
//  ^ tap    ^ i wrote 123     ^ tap
```

◦

Observablemapfiltermap◦ validateEmail◦

```
button.rx_tap                                  // ControlEvent<Void>
        .withLatestFrom(textField.rx_text)     // Observable<String>
        .map(validateEmail)                    // Observable<Bool>
        .map { (isCorrect) in
            return isCorrect ? "Email is correct" : "Input the correct one, please"
        }                                      // Observable<String>
        .bindTo(label.rx_text)
        .addDisposableTo(bag)
```

.....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。

```
//Fetch all the user's name from array
let arrUserName = arrUser.map({ $0.name }) // ["John", "Chan", "Morgan", "Rachel", "Katie",
"David", "Bob"]
```

。

```
// Fetch all user country name & ignore nil value.
let arrCountry = arrUser.flatMap({ $0.country }) // ["USA", "UK", "USA", "USA"]
```

filterincludeArray。

```
// Filtering USA user from the array user list.
let arrUSAUsers = arrUser.filter({ $0.country == "USA" }) // [user1, user5, user6]

// User chaining methods to fetch user's name who live in USA
let arrUserList = arrUser.filter({ $0.country == "USA" }).map({ $0.name }) // ["John",
"Katie", "David"]
```

reduce。

Swift 2.3 -

```
//Fetch user's total age.
let arrUserAge = arrUser.map({ $0.age }).reduce(0, combine: { $0 + $1 }) //174
```

```
//Prepare all user name string with seperated by comma
let strUserName = arrUserName.reduce("", combine: { $0 == "" ? $1 : $0 + ", " + $1 }) // John,
Chan, Morgan, Rachel, Katie, David, Bob
```

3 -

```
//Fetch user's total age.
let arrUserAge = arrUser.map({ $0.age }).reduce(0, { $0 + $1 }) //174

//Prepare all user name string with seperated by comma
let strUserName = arrUserName.reduce("", { $0 == "" ? $1 : $0 + ", " + $1 }) // John, Chan,
Morgan, Rachel, Katie, David, Bob
```

flatMap。nil。 -

。flatMap -

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

```
let arrFlatStateList = arrStateName.flatMap({ $0 }) // ["Alaska", "Iowa", "Missouri", "New
Mexico", "New York", "Texas", "Washington", "Maryland", "New Jersey", "Virginia", "Florida",
"Colorado"]
```

flatten。

```
// Swift 2.3 syntax
let arrSortedStateList = arrStateName.flatMap({ $0 }).sort(<) // ["Alaska",     "Colorado",
"Florida", "Iowa", "Maryland", "Missouri", "New Jersey", "New Mexico", "New York", "Texas",
"Virginia", "Washington"]

// Swift 3 syntax
let arrSortedStateList = arrStateName.flatMap({ $0 }).sorted(by: <) // ["Alaska", "Colorado",
"Florida", "Iowa", "Maryland", "Missouri", "New Jersey", "New Mexico", "New York", "Texas",
"Virginia", "Washington"]
```

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
```

。

```
// Pattern validation for a UK postcode.
// This simply checks that the format looks like a valid UK postcode and should not fail on
```

```
false positives.
private func isPostcodeValid(postcode: String) -> Bool {
    return postcode.matchesPattern("^[A-Z]{1,2}([0-9][A-Z]|[0-9]{1,2})\\s[0-9][A-Z]{2}")
}

// valid patterns (from
https://en.wikipedia.org/wiki/Postcodes_in_the_United_Kingdom#Validation)
// will return true
dump(isPostcodeValid("EC1A 1BB"))
dump(isPostcodeValid("W1A 0AX"))
dump(isPostcodeValid("M1 1AE"))
dump(isPostcodeValid("B33 8TH"))
dump(isPostcodeValid("CR2 6XH"))
dump(isPostcodeValid("DN55 1PT"))

// some invalid patterns
// will return false
dump(isPostcodeValid("EC12A 1BB"))
dump(isPostcodeValid("CRB1 6XH"))
dump(isPostcodeValid("CR 6XH"))
```

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 numbers = "121314"
let pattern = "1[2,3]"
do {
    let regEx = try NSRegularExpression(pattern: pattern, options: [])
    let nsString = numbers as NSString
    let matches = regEx.matches(in: numbers, options: [], range: NSMakeRange(0,
nsString.length))
    let output = matches.map {nsString.substring(with: $0.range)}
    output
} catch let error as NSError {
    print("Matching failed")
}
//output = ["12", "13"]
```

。

。

。

```
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 \\.

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

```
let specials = "(){}[]"
let pattern = "(\\(|\\{|\\[)"
do {
    let regEx = try NSRegularExpression(pattern: pattern, options: [])
    let nsString = specials as NSString
    let matches = regEx.matches(in: specials, options: [], range: NSMakeRange(0,
nsString.length))
    let output = matches.map {nsString.substring(with: $0.range)}
} catch let error as NSError {
    print("Matching failed")
}
//output = ["(", "{", "["]
```

。

```
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

```
extension String
{
    func isValidEmail() -> 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: self)
    }
}
```

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

10: Swift

Examples

`Person`

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

`Person(s)`

```
let persons = [
    Person(name: "Walter White", birthYear: 1959),
    Person(name: "Jesse Pinkman", birthYear: 1984),
    Person(name: "Skyler White", birthYear: 1970),
    Person(name: "Saul Goodman", birthYear: nil)
]
```

Person`nameString`。

```
let names = persons.map { $0.name }
// ["Walter White", "Jesse Pinkman", "Skyler White", "Saul Goodman"]
```

```
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

```
git init
```

◦ CLI◦

```
swift package init --type executable
```

◦ *main.swift*◦

```
swift package init --type library
```

AwesomeProject.swift◦

*Sources*Swift◦

Package.swift

```
import PackageDescription

let package = Package(
    name: "AwesomeProject"
)
```

Git

```
git tag '1.0.0'
```

Git◦

```
swift build
```

.build / debug◦

◦ “SomeOtherPackage”*Package.swift*

```
import PackageDescription

let package = Package(
    name: "AwesomeProject",
    targets: [],
    dependencies: [
        .Package(url: "https://github.com/someUser/SomeOtherPackage.git",
                 majorVersion: 1),
    ]
)
```

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 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
}
}
```

。 。 minmin。 。 On2 - 。

func selectionSort - > Array {//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
}
```

- On log n

Quicksort。On log n。。 Quicksort。 Quicksort。

pivot。

pivotpivotpivot。。。

。

mutating func quickSort - > Array {

```
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 qPartition(start startIndex: Int, _ pivot: Int) -> Int {

var wallIndex: Int = startIndex

//compare range with pivot
for currentIndex in wallIndex..<pivot {

    if self[currentIndex] <= self[pivot] {
        if wallIndex != currentIndex {
            swap(&self[currentIndex], &self[wallIndex])
        }
```

```
        //advance wall
        wallIndex += 1
    }
}
    //move pivot to final position
    if wallIndex != pivot {
        swap(&self[wallIndex], &self[pivot])
    }
    return wallIndex
}
```

。。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 {

```
var wallIndex: Int = startIndex

//compare range with pivot
for currentIndex in wallIndex..<pivot {

    if self[currentIndex] <= self[pivot] {
        if wallIndex != currentIndex {
            swap(&self[currentIndex], &self[wallIndex])
        }

        //advance wall
        wallIndex += 1
    }
}
```

```
    //move pivot to final position
    if wallIndex != pivot {
        swap(&self[wallIndex], &self[pivot])
    }
    return wallIndex
}
```

。。。。。。

```
//
//  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 - ◦

```
//
//  Trie.swift
//  SwiftStructures
//
//  Created by Wayne Bishop on 10/14/14.
//  Copyright (c) 2014 Arbutus Software Inc. All rights reserved.
//
import Foundation


public class Trie {

    private var root: TrieNode!


    init(){
        root = TrieNode()
    }



    //builds a tree hierarchy of dictionary content
    func append(word keyword: String) {


        //trivial case
        guard keyword.length > 0 else {
            return
        }


        var current: TrieNode = root


        while keyword.length != current.level {

            var childToUse: TrieNode!
            let searchKey = keyword.substring(to: current.level + 1)
```

```
        //print("current has \(current.children.count) children..")


        //iterate through child nodes
        for child in current.children {

            if (child.key == searchKey) {
                childToUse = child
                break
            }

        }


        //new node
        if childToUse == nil {

            childToUse = TrieNode()
            childToUse.key = searchKey
            childToUse.level = current.level + 1
            current.children.append(childToUse)
        }


        current = childToUse


    } //end while


    //final end of word check
    if (keyword.length == current.level) {
        current.isFinal = true
        print("end of word reached!")
        return
    }



} //end function




//find words based on the prefix
func search(forWord keyword: String) -> Array<String>! {


    //trivial case
    guard keyword.length > 0 else {
        return nil
    }


    var current: TrieNode = root
    var wordList = Array<String>()


    while keyword.length != current.level {

        var childToUse: TrieNode!
```

```
            let searchKey = keyword.substring(to: current.level + 1)


            //print("looking for prefix: \(searchKey)..")


            //iterate through any child nodes
            for child in current.children {

                if (child.key == searchKey) {
                    childToUse = child
                    current = childToUse
                    break
                }

            }


            if childToUse == nil {
               return nil
            }


        } //end while



        //retrieve the keyword and any descendants
        if ((current.key == keyword) && (current.isFinal)) {
            wordList.append(current.key)
        }


        //include only children that are words
        for child in current.children {

            if (child.isFinal == true) {
                wordList.append(child.key)
            }

        }


        return wordList


    } //end function


}
```

GitHub

pushpop。LIFO。。

github

```
//
//  Stack.swift
```

```
//  SwiftStructures
//
//  Created by Wayne Bishop on 8/1/14.
//  Copyright (c) 2014 Arbutus Software Inc. All rights reserved.
//
import Foundation


class Stack<T> {

    private var top: Node<T>

    init() {
        top = Node<T>()
    }


    //the number of items - O(n)
    var count: Int {


        //return trivial case
        guard top.key != nil else {
          return 0
        }


        var current = top
        var x: Int = 1


        //cycle through list
        while current.next != nil {
            current = current.next!
            x += 1
        }

        return x

    }


    //add item to the stack
    func push(withKey key: T) {


        //return trivial case
        guard top.key != nil else {
            top.key = key
            return
        }


        //create new item
        let childToUse = Node<T>()
        childToUse.key = key


        //set new created item at top
        childToUse.next = top
        top = childToUse
```

```
    }


    //remove item from the stack
    func pop() {

        if self.count > 1 {
            top = top.next
        }
        else {
            top.key = nil
        }

    }


    //retrieve the top most item
    func peek() -> T! {


        //determine instance
        if let topitem = top.key {
            return topitem
        }

        else {
            return nil
        }

    }



    //check for value
    func isEmpty() -> Bool {

        if self.count == 0 {
            return true
        }

        else {
            return false
        }

    }


}
```

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

```
func example(_ handler: SuccessHandler) {}
```

```
func example(_ handler: (NSURLSessionDataTask, AnyObject?) -> Void) {}
```

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

Void to Void。

```
var func: Handler?

func = {}
```

```
typealias Number = NSNumber
```

。

```
typealias PersonTuple = (name: String, age: Int, address: String)
```

```
func getPerson(for name: String) -> PersonTuple {
    //fetch from db, etc
    return ("name", 45, "address")
}
```

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 = dispatch_get_main_queue()
```

。Swift 3`DispatchQueue`

3.0

```
let globalConcurrentQueue = DispatchQueue.global(qos: .default)
```

```
let globalConcurrentQueue = DispatchQueue.global()
```

3.0

```
let globalConcurrentQueue = dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0)
```

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

3.0

```
let myConcurrentQueue = DispatchQueue(label: "my-concurrent-queue", qos: .userInitiated,
attributes: [.concurrent], autoreleaseFrequency: .workItem, target: nil)
let mySerialQueue = DispatchQueue(label: "my-serial-queue", qos: .background, attributes: [],
autoreleaseFrequency: .workItem, target: nil)
```

3.0

```
let myConcurrentQueue = dispatch_queue_create("my-concurrent-queue",
DISPATCH_QUEUE_CONCURRENT)
let mySerialQueue = dispatch_queue_create("my-serial-queue", DISPATCH_QUEUE_SERIAL)
```

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
```

```
queue.sync {
    // Do some task
}
// ... code here will not execute until the task is finished
```

```
queue.asyncAfter(deadline: .now() + 3) {
    //this will be executed in a background-thread after 3 seconds
}
```

```
// ... code here will execute immediately, before the task finished
```

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

2.0

```
let mainQueue = dispatch_get_main_queue()
let highQueue = dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_HIGH, 0)
let backgroundQueue = dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_BACKGROUND, 0)
```

```
dispatch_async(queue) {
    // Your code run run asynchronously. Code is queued and executed
    // at some point in the future.
}
// Code after the async block will execute immediately
```

```
dispatch_sync(queue) {
    // Your sync code
}
// Code after the sync block will wait until the sync task finished
```

`NSEC_PER_SEC`

```
dispatch_after(dispatch_time(DISPATCH_TIME_NOW, Int64(2.5 * Double(NSEC_PER_SEC))),
dispatch_get_main_queue()) {
    // Code to be performed in 2.5 seconds here
}
```

UI

```
dispatch_async(queue) {
    // Your time consuming code here
    dispatch_async(dispatch_get_main_queue()) {
        // Update the UI code
    }
}
```

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

```
DispatchQueue.concurrentPerform(iterations: iterations) { index in
```

```
    // Do something computationally expensive here
}
```

3.0

```
dispatch_apply(iterations, queue) { index in
    // 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 queue = OperationQueue()
queue.name = "My Queue"
queue.qualityOfService = .default
```

。

OperationOperationQueue

3.0

```
// An instance of some Operation subclass
let operation = BlockOperation {
    // perform task here
}

queue.addOperation(operation)
```

OperationQueue

3.0

```
myQueue.addOperation {
    // some task
}
```

OperationOperationQueue

3.0

```
let operations = [Operation]()
// Fill array with Operations

myQueue.addOperation(operations)
```

Operation

```
myQueue.maxConcurrentOperationCount = 3 // 3 operations may execute at once

// Sets number of concurrent operations based on current system conditions
myQueue.maxConcurrentOperationCount = NSOperationQueueDefaultMaxConcurrentOperationCount
```

◦ isSuspendedfalse

3.0

```
myQueue.isSuspended = true

// Re-enable execution
myQueue.isSuspended = false
```

OperationQueue◦ ◦

FoundationOperation◦ ◦

Operation◦ isReadytrue ◦

Operation

3.0

```
class MyOperation: Operation {

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

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

}
```

2.3

```
class MyOperation: NSOperation {

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

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

}
```

OperationQueueOperationQueue

1.0

```
myQueue.addOperation(operation)
```

◦

Operation◦

OperationOperation ◦

1.0

```
operation2.addDependency(operation1)

operation2.removeDependency(operation1)
```

Operation

1.0

```
   operation.start()
```

◦ start◦

◦

OperationURLSessionOperation◦ isAsynchronoustrue startmain◦

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

Operation ◦

cancelisCancelledtrue ◦ OperationmainisCancelled◦

1.0

```
operation.cancel()
```

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))
```

```
[3, 5, 7, 9, 11, 13, 10, 8, 6, 4, 102]
```

closure

```
// assigning the closure to a variable and then passing it to a function as param
let add2 = {(item: Int)-> Int in return item + 2}
print(math.performOperation(inputArray: arrayToBeProcessed, operation: add2))
```

```
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))
```

```
[3, 5, 7, 9, 11, 13, 10, 8, 6, 4, 102]
```

closure

```
// passing the closure directly to the function as param
print(math.performOperation(inputArray: arrayToBeProcessed, operation: { $0 * 2 }))
```

```
// 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

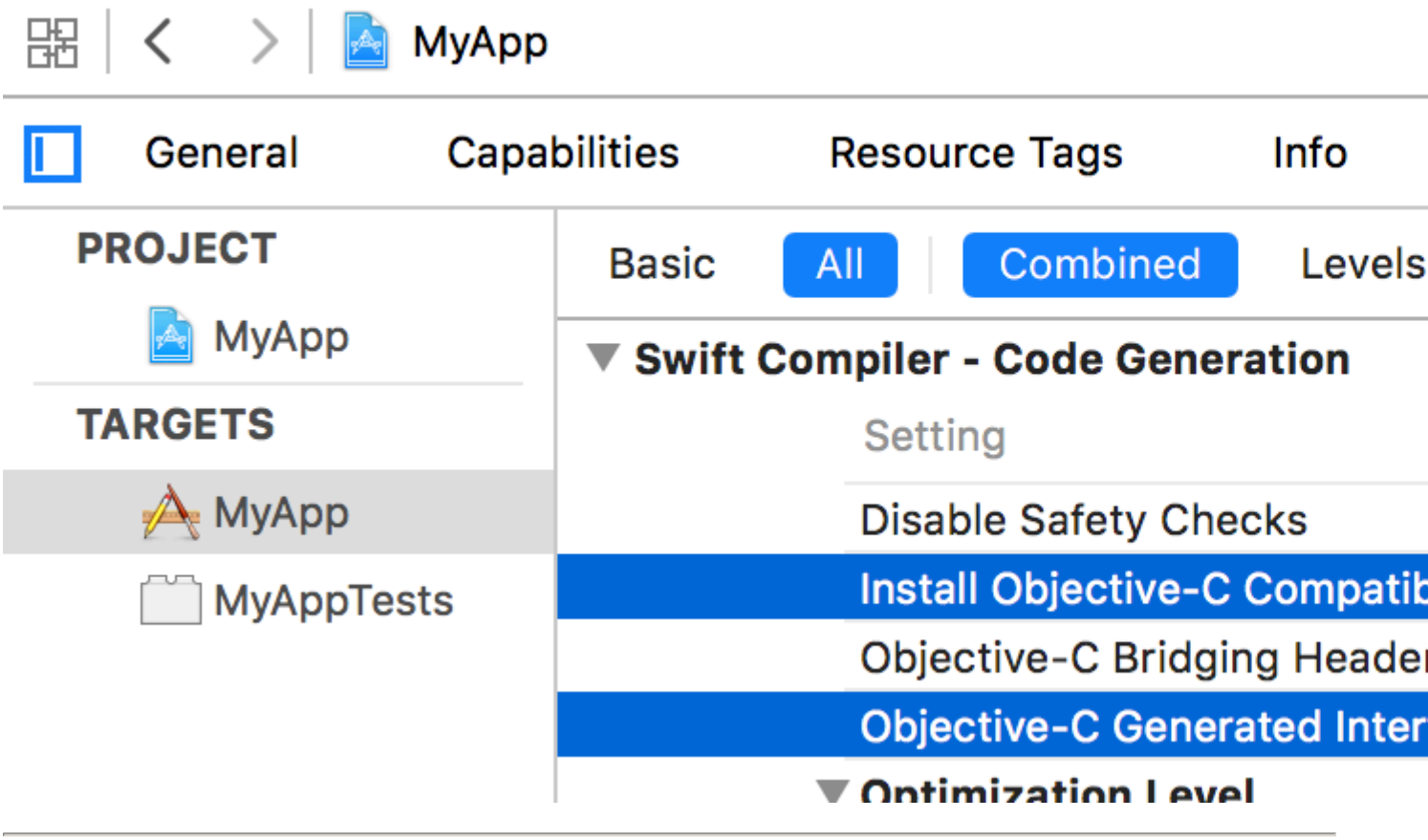
// The class must be `public` to be visible, unless this target also has a bridging header
public class MySwiftClass: NSObject {
    // ...
}
```

```
// MyViewController.m in MyApp

#import "MyViewController.h"
#import "MyApp-Swift.h"                      // import the generated interface
#import <MyFramework/MyFramework-Swift.h>  // or use angle brackets for a framework target

@implementation MyViewController
- (void)demo {
    [[MySwiftClass alloc] init];  // use the Swift class
}
@end
```

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

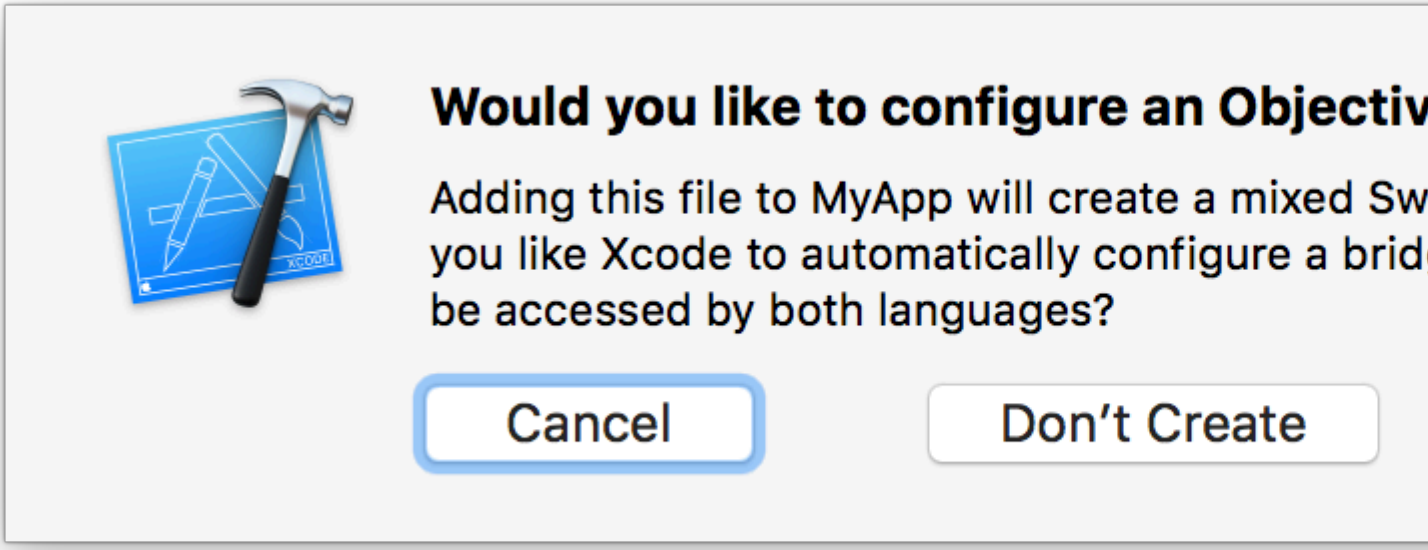


`@import MyFramework;`SwiftObj-C。

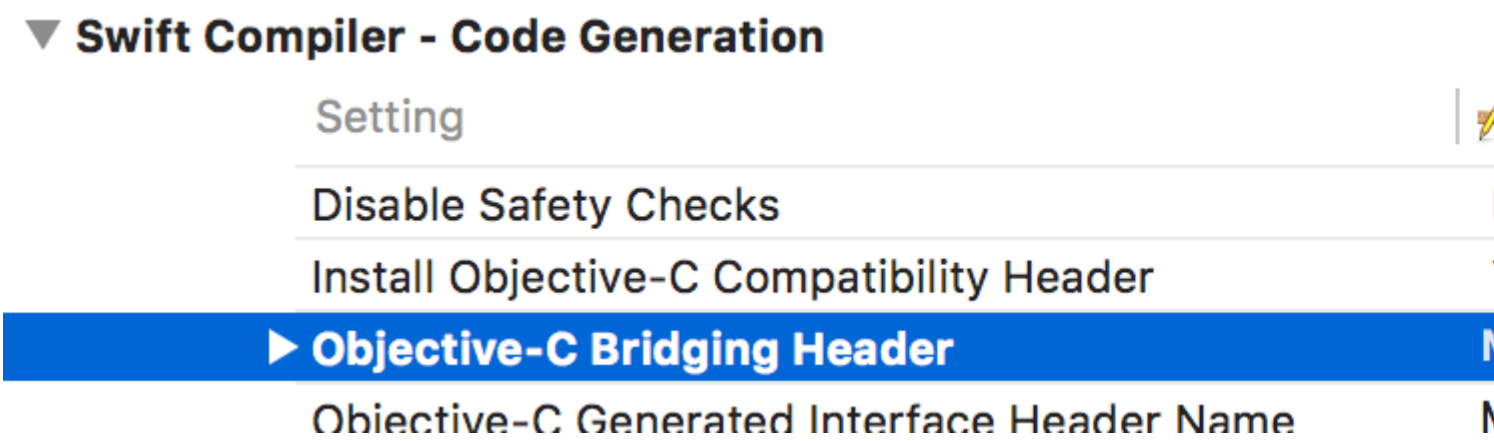
SwiftObjective-C

MyFrameworkObjective-C`import MyFramework`Swift。

SwiftObjective-CC。 Xcode

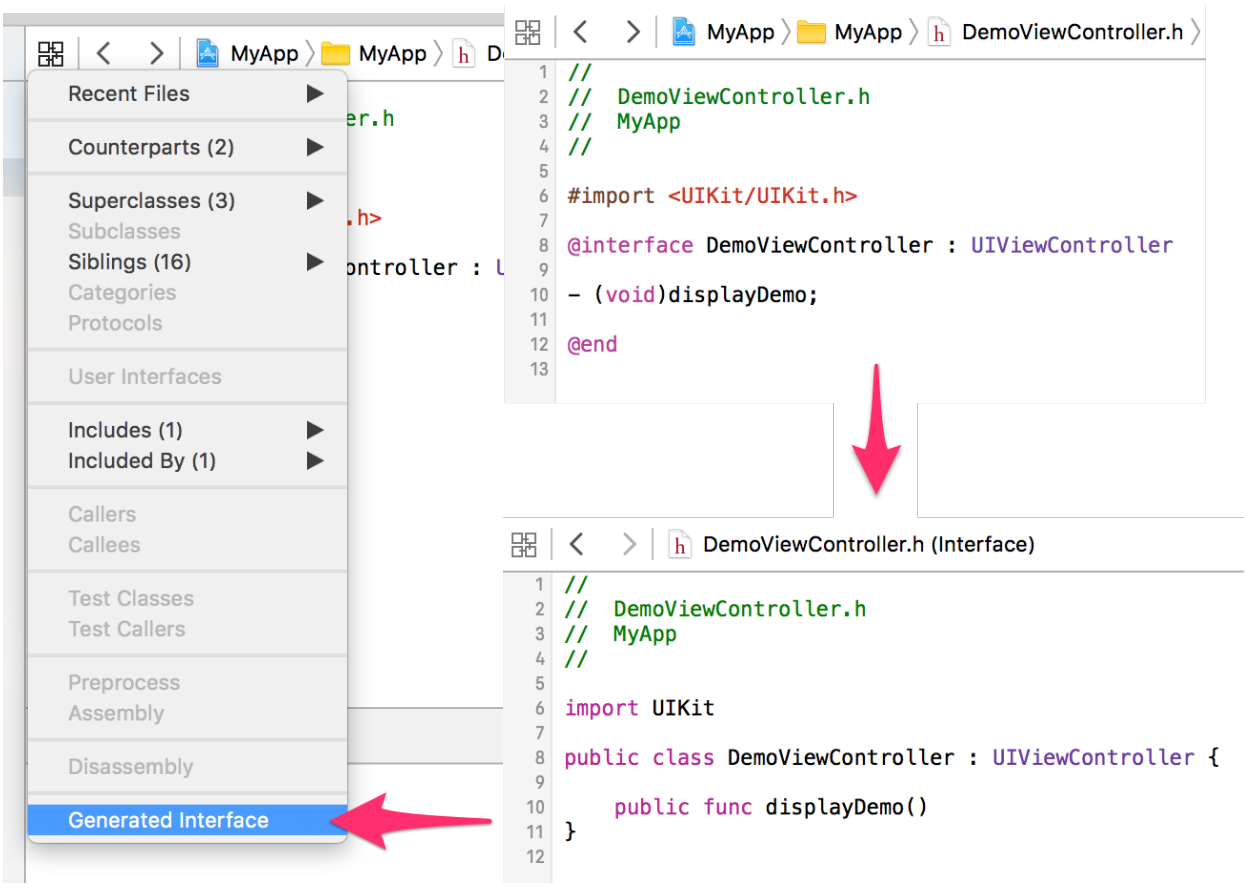


Objective-C Bridging Header




```
// 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
```

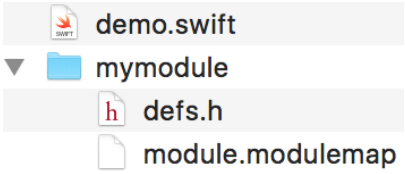
```
// demo.swift
extension Color: CustomStringConvertible {  // extension on a C struct
    public var description: String {
        return "Color(red: \(red), green: \(green), blue: \(blue))"
    }
}
print("MAX_VALUE is: \(MAX_VALUE)")  // C macro becomes a constant
let color = Color(red: 0xCA, green: 0xCA, blue: 0xD0)  // C struct initializer
print("The color is \(color)")
```

```
$ swiftc demo.swift -import-objc-header defs.h && ./demo
MAX_VALUE is: 255
The color is Color(red: 202, green: 202, blue: 208)
```

C

`import mymodule`CSwift。

`module.modulemapmymodule`




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

`import`

```
// demo.swift
import mymodule
print("Empty color: \(Color())")
```

`-I directory`swiftc

```
swiftc -I . demo.swift   # "-I ." means "search for modules in the current directory"
```

Clang 。

Objective-CSwift

`NS_REFINED_FOR_SWIFT`APISwift __

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

```
@end
```

```
public class MyClass : NSObject {
    public func __indexOfObject(obj: AnyObject) -> Int
}
```

“Swifty”**API** 。 NSNotFound

```
extension MyClass {
    // Rather than returning NSNotFound if the object doesn't exist,
    // this "refined" API returns nil.
    func indexOfObject(obj: AnyObject) -> Int? {
        let idx = __indexOfObject(obj)
        if idx == NSNotFound { return nil }
        return idx
    }
}

// Swift code, using "if let" as it should be:
let myobj = MyClass()
if let idx = myobj.indexOfObject(something) {
    // do something with idx
}
```

Objective-C`nil` 。 `_Nonnull`

```
void
doStuff(const void *const _Nonnull data, void (^_Nonnull completion)())
{
    // complex asynchronous code
}
```

Swift`nil`

```
doStuff(
    nil,  // error: nil is not compatible with expected argument type 'UnsafeRawPointer'
    nil)  // error: nil is not compatible with expected argument type '() -> Void'
```

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

```
void
callNow(__attribute__((noescape)) void (^_Nonnull f)())
{
    // f is not stored anywhere
}
```

。

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 TimelineViewController: UIViewController {
```

```
    var networkService = FirebaseNetworkService()

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

    @IBAction func logoutButtonPressed(_ sender: UIButton) {
        networkService.logutCurrentUser()
    }
}
```

FirebaseNetworkService

```
class FirebaseNetworkService {

    func loginUser(username: String, passwordHash: String) {
        // Implementation not important for this example
    }

    func logutCurrentUser() {
        // Implementation not important for this example
    }
}
```

1015FirebaseNetworkService。Firebase。CompanyNetworkServiceFirebaseNetworkService。CompanyNetworkService。

UI。

。

。 。

```
protocol NetworkService {
    func loginUser(username: String, passwordHash: String)
    func logutCurrentUser()
}
```

NetworkService

```
class FirebaseNetworkServiceImpl: NetworkService {
    func loginUser(username: String, passwordHash: String) {
        // Firebase implementation
    }

    func logutCurrentUser() {
        // Firebase implementation
    }
}
```

LoginViewControllerTimelineViewControllerNetworkServiceFirebaseNetworkService。

LoginViewController

```
class LoginViewController: UIViewController {

    // No need to initialize it here since an implementation
    // of the NetworkService protocol will be injected
    var networkService: NetworkService?

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

TimelineViewController

```
class TimelineViewController: UIViewController {

    var networkService: NetworkService?

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

    @IBAction func logoutButtonPressed(_ sender: UIButton) {
        networkService?.logutCurrentUser()
    }
}
```

LoginViewControllerTimelineViewControllerNetworkService

LoginViewController**AppDelegate**。

```
func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions:
[UIApplicationLaunchOptionsKey: Any]?) -> Bool {
    // This logic will be different based on your project's structure or whether
    // you have a navigation controller or tab bar controller for your starting view
controller
    if let loginVC = window?.rootViewController as? LoginViewController {
        loginVC.networkService = FirebaseNetworkServiceImpl()
    }
    return true
}
```

AppDelegateLoginViewControllerNetworkService。

TimelineViewControllerNetworkService。 LoginViewControllerTimlineViewController。

LoginViewControllerprepareForSegue。

LoginViewController

```
class LoginViewController: UIViewController {
    // No need to initialize it here since an implementation
    // of the NetworkService protocol will be injected
    var networkService: NetworkService?

    override func viewDidLoad() {
```

```
        super.viewDidLoad()
    }

    override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
        if segue.identifier == "TimelineViewController" {
            if let timelineVC = segue.destination as? TimelineViewController {
                // Injecting the NetworkService implementation
                timelineVC.networkService = networkService
            }
        }
    }
}
```

。

NetworkServiceFirebase。

NetworkService

```
class CompanyNetworkServiceImpl: NetworkService {
    func loginUser(username: String, passwordHash: String) {
        // Company API implementation
    }

    func logutCurrentUser() {
        // Company API implementation
    }
}
```

AppDelegateFirebaseNetworkServiceImpl

```
func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions:
[UIApplicationLaunchOptionsKey: Any]?) -> Bool {
        // This logic will be different based on your project's structure or whether
        // you have a navigation controller or tab bar controller for your starting view
controller
        if let loginVC = window?.rootViewController as? LoginViewController {
            loginVC.networkService = CompanyNetworkServiceImpl()
        }
        return true
    }
```

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◦

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

    init(engine: Engine) {
        self.engine = engine
    }
}
```

TrainEngine

```
let train = Train(engine: TrainEngine())
```

let◦ ◦

DI◦ DIPassengerCar

```
train.mainCar = PassengerCar()
```

◦ mainCarPassengerCar◦

◦ ◦ Train

```
func reparkCar(trainCar: TrainCar) {
    trainCar.attachCar(attach: true)
    engine?.startEngine()
    engine?.stopEngine()
    trainCar.attachCar(attach: false)
}
```

Train`TrainCar` ◦

```
train.reparkCar(trainCar: RestaurantCar())
```

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

18:

。

。

:)。

-

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

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 namedTuple = (first: 1, middle: "dos", last: 3)

// Values can be read with the named property
print(namedTuple.first)  // 1
print(namedTuple.middle) // dos

// And still with the index number
print(namedTuple.2)      // 3
```

```
var numbers: (optionalFirst: Int?, middle: String, last: Int)?

//Later On
numbers = (nil, "dos", 3)

print(numbers.optionalFirst)// nil
print(numbers.middle)//"dos"
print(numbers.last)//3
```

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

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

```
let unnamedTuple = ("uno", "dos")
let (one, two) = unnamedTuple
```

```
print(one) // "uno"
print(two) // "dos"
```

_

```
let longTuple = ("ichi", "ni", "san")
let (_, _, third) = longTuple
print(third) // "san"
```

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

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

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

let myTuple = tupleReturner()
print(myTuple.anInteger) // 3
print(myTuple.aString) // "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
typealias Circle = (center: (x: CGFloat, y: CGFloat), radius: CGFloat)

let unitCircle: Circle = ((0.0, 0.0), 1)

func doubleRadius(ofCircle circle: Circle) -> Circle {
    // Aliased tuples also have access to value labels in the original tuple type.
    return (circle.center, circle.radius * 2.0)
}
```

struct 。

2。

2

2

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

```
(a, b) = (b, a)
```

```
print(a) // "Emmett Brown"
print(b) // "Marty McFly"
```

4

```
var a = 0
var b = 1
var c = 2
var d = 3

(a, b, c, d) = (d, c, b, a)

print(a, b, c, d) // 3, 2, 1, 0
```

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 = (UIUserInterfaceSizeClass.Compact, UIUserInterfaceSizeClass.Regular)

    switch switchTuple {
    case (.Regular, .Compact):
        //statement
    case (.Regular, .Regular):
        //statement
    case (.Compact, .Regular):
        //statement
    case (.Compact, .Compact):
        //statement
    }
```

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。

。

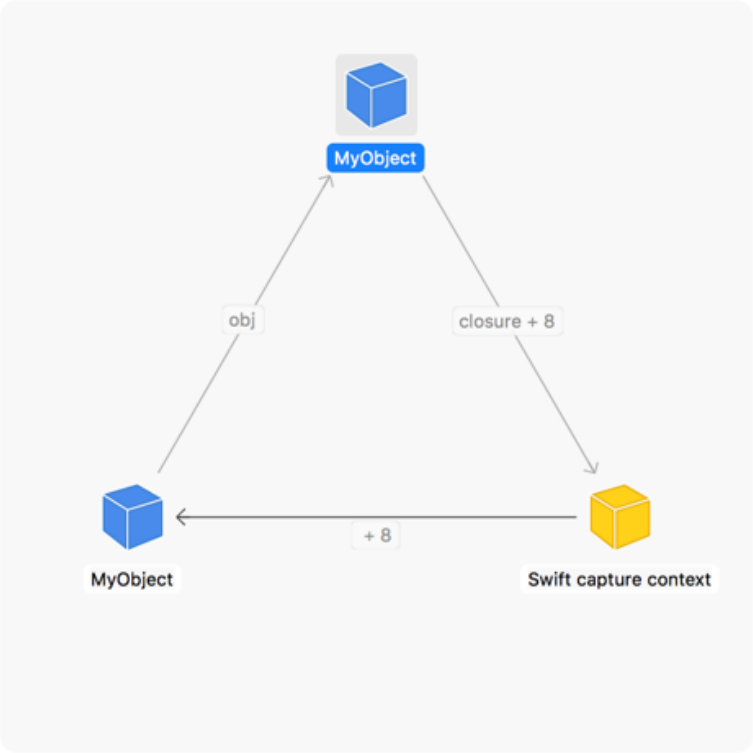
```
class A : CLLocationManagerDelegate
{
    init()
    {
        let locationManager = CLLocationManager()
        locationManager.delegate = self
        locationManager.startLocationUpdates()
    }
}
```

。

```
class A : CLLocationManagerDelegate
{
    let locationManager:CLLocationManager

    init()
    {
        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 B { weak var a: A? = nil }
```

◦ ◦ **nil** ◦

```
a.b = b  // a retains b
b.a = a  // b holds a weak reference to a -- not 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

```
func preferredFilenameExtension(for uti: String) -> String! {
  let result = UTTypeCopyPreferredTagWithClass(uti, kUTTagClassFilenameExtension)
  guard result != nil else { return nil }

  return result!.takeRetainedValue() as String
}
```

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 = 42 // the type 'Int' is inferred here
}
```

◦ downvotes

```
struct Example {
    var upvotes: Int
    var downvotes: Int
    init() {
         upvotes = 0
    } // error: Return from initializer without initializing all stored properties
}
```

```
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
```

```
let myBadDistance = MetricDistance(42) // error: argument labels do not match any available
overloads
```

_

```
struct MetricDistance {
    var distanceInMeters: Double
    init(_ meters: Double) {
        distanceInMeters = meters
```

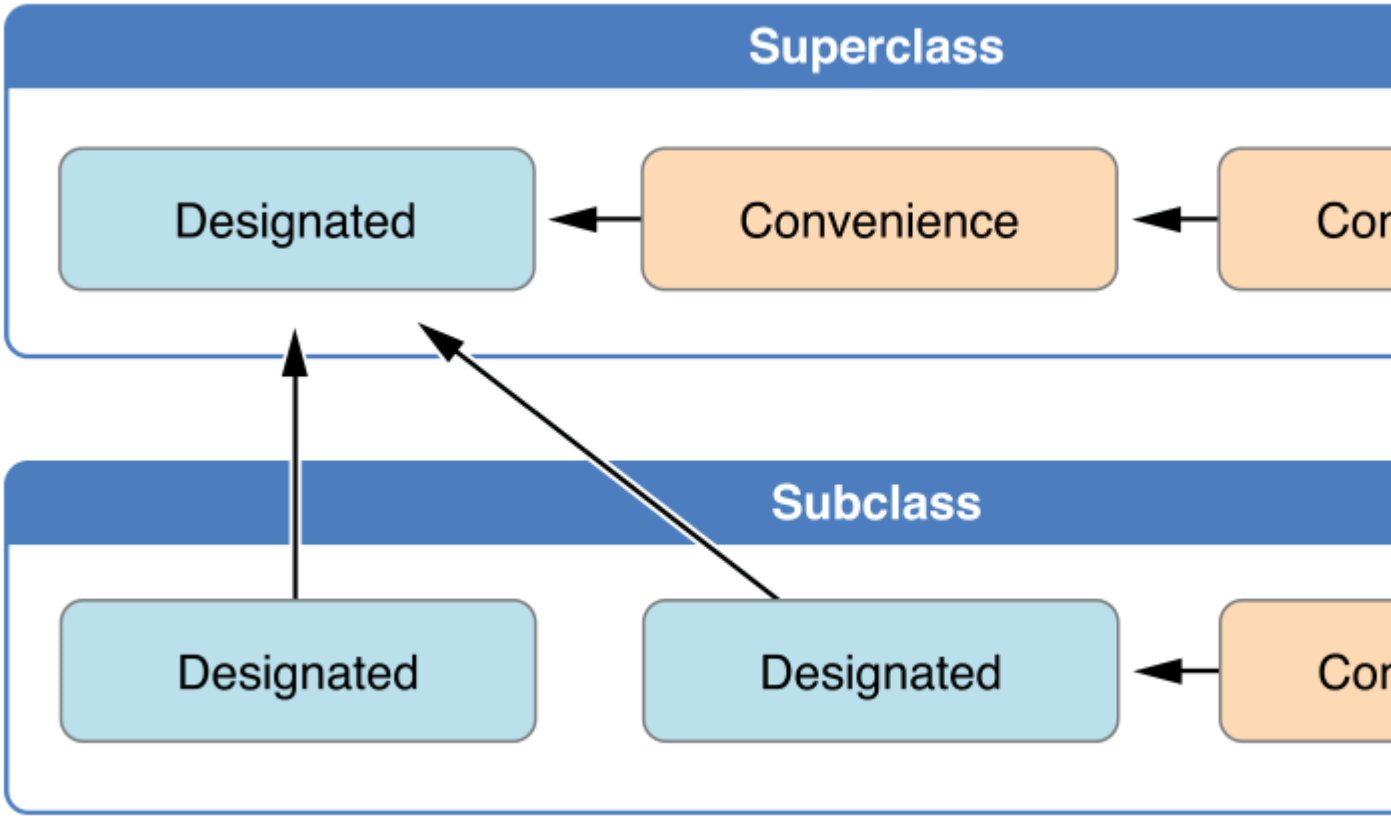
```
    }
}
let myDistance = MetricDistance(42) // distanceInMeters = 42
```

`self`

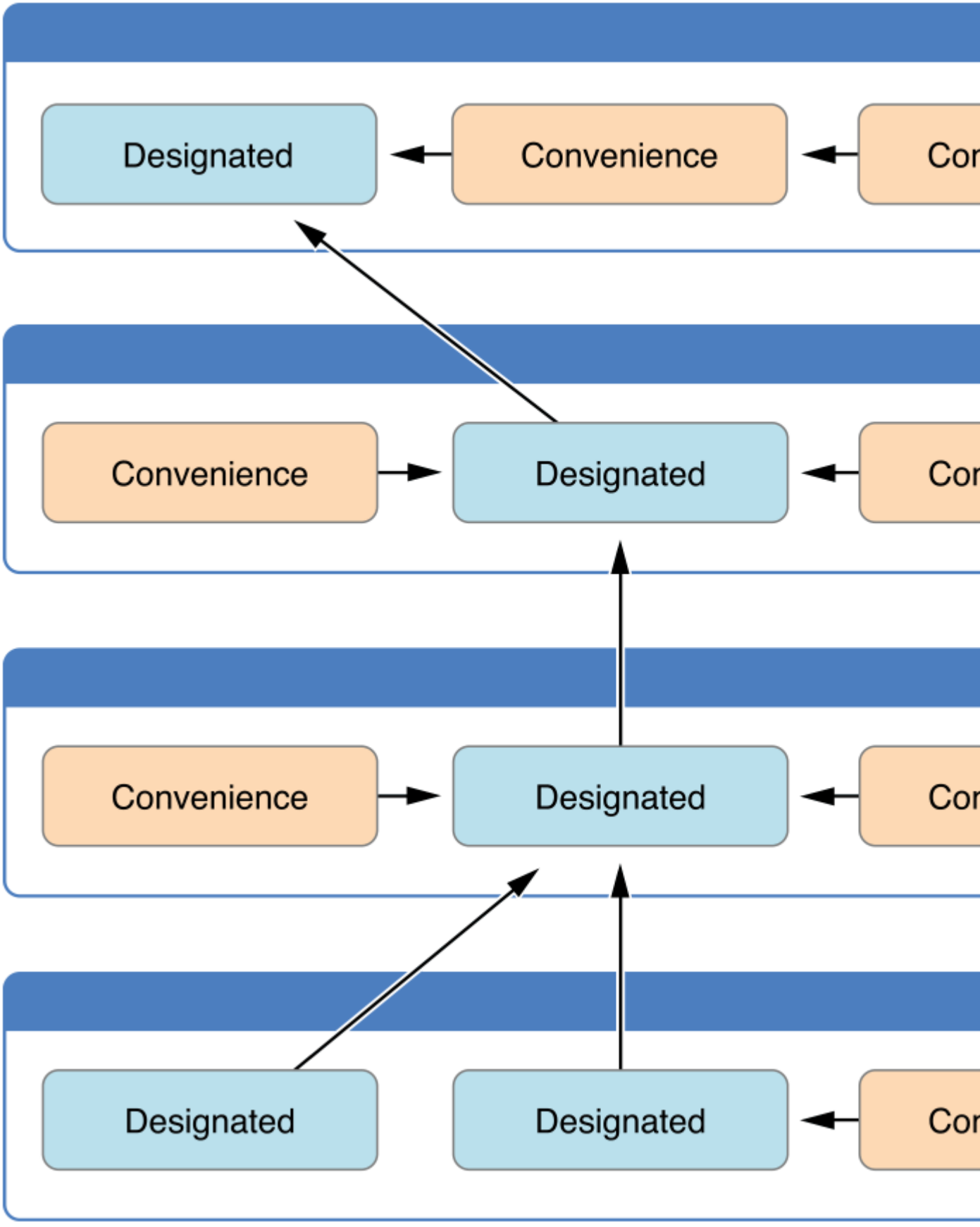
```
struct Color {
    var red, green, blue: Double
    init(red: Double, green: Double, blue: Double) {
        self.red = red
        self.green = green
        self.blue = blue
    }
}
```

Swift。Apple3

1. 。



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)
    }
}
```

```
let c = Foo(someString: "Some string", someValue: 10, someBool: true)
```

init

```
let a = Foo()
```

initotherStringString

```
let b = Foo(otherString: "Some string")
```

```
let d = Baz(someFloat: 3)
```

init

```
let e = Baz()
```

Swift

Struct

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

Struct

```
struct User {
    let name: String

    init(name: String?) throws {

        guard let name = name else {
           ValidationError.invalid
        }

        self.name = name
    }
}
```

throwable

```
do {
   let user = try User(name: "Sample name")

   // success
}
catch ValidationError.invalid {
     // handle error
}
```

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

21:

Examples

◦ hello ◦

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

◦

```
hello()
//output: "Hello World"
```

◦ ◦

```
func magicNumber(number1: Int)
{
    print("\(number1) Is the magic number")
}
```

\(number1) String Interpolation String◦

◦

```
magicNumber(5)
//output: "5 Is the magic number
let example: Int = 10
magicNumber(example)
//output: "10 Is the magic number"
```

Int◦

```
func magicNumber(number1: Int, number2: Int)
{
    print("\(number1 + number2) Is the magic number")
}
```

◦

```
let ten: Int = 10
let five: Int = 5
magicNumber(ten,number2: five)
//output: "15 Is the magic number"
```

◦

```
func magicNumber(one number1: Int, two number2: Int)
{
    print("\(number1 + number2) Is the magic number")
}

let ten: Int = 10
let five: Int = 5
magicNumber(one: ten, two: five)
```

。

```
func magicNumber(one number1: Int = 5, two number2: Int = 10)
{
    print("\(number1 + number2) Is the magic number")
}

magicNumber()
//output: "15 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 maths(number: Int) -> (times2: Int, times3: Int)
{
    let two = number * 2
    let three = number * 3
    return (two, three)
}
let resultTuple = maths(5)
//resultTuple = (10, 15)
```

throws

```
func errorThrower()throws -> String {}
```

throw

```
func errorThrower()throws -> String {
  if true {
    return "True"
  } else {
    // Throwing an error
    throw Error.error
  }
}
```

dotry

```
do {
  try errorThrower()
}
```

Swift

Swift ◦ ◦

func◦

```
class Counter {
    var count = 0
    func increment() {
        count += 1
    }
}
```

Counterincrement()

```
let counter = Counter()  // create an instance of Counter class
counter.increment()      // call the instance method on this instance
```

static func◦ class func◦

```
class SomeClass {
    class func someTypeMethod() {
        // type method implementation goes here
    }
}
```

```
SomeClass.someTypeMethod()  // type method is called on the SomeClass type itself
```

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

```
loadData("123") { result in
    print(result)
}
```

+ - ??◦

- **-** x
- x **+** y
- x **++**

Swift◦

◦ sum

```
func sum(_ a: Int, _ b: Int) -> Int {
    return a + b
}
```

```
func sum(_ a: Int, _ b: Int, _ c: Int) -> Int {
    return a + b + c
}
```

◦ Swift ...

```
func sum(_ numbers: Int...) -> Int {
    return numbers.reduce(0, combine: +)
}
```

numbers[Int]Array◦ T...[T]◦

```
let a = sum(1, 2) // a == 3
let b = sum(3, 4, 5, 6, 7) // b == 25
```

Swift◦

◦ sum◦ variadic◦ sum

```
func sum(_ n1: Int, _ n2: Int, _ numbers: Int...) -> Int {
    return numbers.reduce(n1 + n2, combine: +)
}

sum(1, 2) // ok
sum(3, 4, 5, 6, 7) // ok
sum(1) // not ok
```

```
sum() // not ok
```

◦

```
struct DaysOfWeek {

  var days = ["Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat"]

  subscript(index: Int) -> String {
    get {
      return days[index]
    }
    set {
      days[index] = newValue
    }
  }
}
```

```
var week = DaysOfWeek()
//you access an element of an array at index by array[index].
debugPrint(week[1])
debugPrint(week[0])
week[0] = "Sunday"
debugPrint(week[0])
```

◦ ◦ ◦

```
struct Food {

  enum MealTime {
    case Breakfast, Lunch, Dinner
  }

  var meals: [MealTime: String] = [:]

  subscript (type: MealTime) -> String? {
    get {
      return meals[type]
    }
    set {
      meals[type] = newValue
    }
  }
}
```

```
var diet = Food()
diet[.Breakfast] = "Scrambled Eggs"
diet[.Lunch] = "Rice"

debugPrint("I had \(diet[.Breakfast]) for breakfast")
```

◦ Void ◦

```
func closedFunc(block: (()->Void)? = nil) {
```

```
    print("Just beginning")

    if let block = block {
        block()
    }
}
```

```
closedFunc() { Void in
    print("Over already")
}
```

print closedFunc()◦

```
class ViewController: UIViewController {

    override func viewDidLoad() {
        let _  = A.init(){Void in self.action(2)}
    }

    func action(i: Int) {
        print(i)
    }
}


class A: NSObject {
    var closure : ()?

    init(closure: (()->Void)? = nil) {
        // Notice how this is executed before the  closure
        print("1")
        // Make sure closure isn't nil
        self.closure = closure?()
    }
}
```

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 }
```

```
(Int, Int) -> (Int)
```

。

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。

```
struct MyStruct : MyProtocol {
    // Implement the protocol's requirements here
}
class MyClass : MyProtocol {
    // Implement the protocol's requirements here
}
enum MyEnum : MyProtocol {
    case caseA, caseB, caseC
    // Implement the protocol's requirements here
}
```

```
extension MyProtocol {

    // default implementation of doSomething() -> Bool
    // conforming types will use this implementation if they don't define their own
    func doSomething() -> Bool {
        print("do something!")
        return true
    }
}
```

associatedtype

```
func doStuff(object: MyProtocol) {
    // All of MyProtocol's requirements are available on the object
    print(object.message)
    print(object.doSomething())
}

let items : [MyProtocol] = [MyStruct(), MyClass(), MyEnum.caseA]
```

3.0

Swift 3 &

```
func doStuff(object: MyProtocol & AnotherProtocol) {
    // ...
}

let items : [MyProtocol & AnotherProtocol] = [MyStruct(), MyClass(), MyEnum.caseA]
```

3.0

protocol<...> <>◦

```
protocol AnotherProtocol {
    func doSomethingElse()
}

func doStuff(object: protocol<MyProtocol, AnotherProtocol>) {

    // All of MyProtocol & AnotherProtocol's requirements are available on the object
    print(object.message)
    object.doSomethingElse()
}

// MyStruct, MyClass & MyEnum must now conform to both MyProtocol & AnotherProtocol
let items : [protocol<MyProtocol, AnotherProtocol>] = [MyStruct(), MyClass(), MyEnum.caseA]
```

```
extension String : MyProtocol {
    // Implement any requirements which String doesn't already satisfy
}
```

associatedtype

```
protocol Container {
    associatedtype Element
    var count: Int { get }
    subscript(index: Int) -> Element { get set }
}
```

```
// These are NOT allowed, because Container has associated type requirements:
func displayValues(container: Container) { ... }
class MyClass { let container: Container }
// > error: protocol 'Container' can only be used as a generic constraint
// > because it has Self or associated type requirements

// These are allowed:
func displayValues<T: Container>(container: T) { ... }
class MyClass<T: Container> { let container: T }
```

associatedtypeassociatedtype

```
struct ContainerOfOne<T>: Container {
    let count = 1          // satisfy the count requirement
    var value: T

    // satisfy the subscript associatedtype requirement implicitly,
    // by defining the subscript assignment/return type as T
    // therefore Swift will infer that T == Element
    subscript(index: Int) -> T {
        get {
            precondition(index == 0)
            return value
        }
        set {
            precondition(index == 0)
            value = newValue
        }
    }
}

let container = ContainerOfOne(value: "Hello")
```

T - Element associatedtype Element。 Elementassociatedtype Element。

typealiasassociatedtype typealias

```
struct ContainerOfOne<T>: Container {

    typealias Element = T
    subscript(index: Int) -> Element { ... }

    // ...
}
```

```
// Expose an 8-bit integer as a collection of boolean values (one for each bit).
extension UInt8: Container {

    // as noted above, this typealias can be inferred
    typealias Element = Bool
```

```
    var count: Int { return 8 }
    subscript(index: Int) -> Bool {
        get {
            precondition(0 <= index && index < 8)
            return self & 1 << UInt8(index) != 0
        }
        set {
            precondition(0 <= index && index < 8)
            if newValue {
                self |= 1 << UInt8(index)
            } else {
                self &= ~(1 << UInt8(index))
            }
        }
    }
}
```

```
extension Array: Container {}  // Array satisfies all requirements, including Element
```

CocoaCocoaTouch。 。

。 Objectsprotocol。 。

。

。

```
class Parent { }
class Child { }
```

。

Swiftprotocoldelegateprotocol 。 delegateparent。

```
protocol ChildDelegate: class {
    func childDidSomething()
}
```

```
class Child {
    weak var delegate: ChildDelegate?
}
```

delegateChildDelegate delegateweak。 delegate。 。

weakclassChildDelegate。

```
delegate?.childDidSomething()
```

。

ChildDelegate。

```
class Parent: ChildDelegate {
    ...

    func childDidSomething() {
        print("Yay!")
    }
}
```

```
extension Parent: ChildDelegate {
    func childDidSomething() {
        print("Yay!")
    }
}
```

```
// In the parent
let child = Child()
child.delegate = self
```

Swift protocol。 @objcoptional。

UITableViewiOSUITableViewDelegateUITableViewDataSource。

```
@objc public protocol UITableViewDelegate : NSObjectProtocol, UIScrollViewDelegate {

        // Display customization
        optional public func tableView(tableView: UITableView, willDisplayCell cell:
UITableViewCell, forRowAtIndexPath indexPath: NSIndexPath)
        optional public func tableView(tableView: UITableView, willDisplayHeaderView view:
UIView, forSection section: Int)
        optional public func tableView(tableView: UITableView, willDisplayFooterView view:
UIView, forSection section: Int)
        optional public func tableView(tableView: UITableView, didEndDisplayingCell cell:
UITableViewCell, forRowAtIndexPath indexPath: NSIndexPath)
        ...
}
```

```
class MyViewController : UIViewController, UITableViewDelegate
```

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")
}
```

```
extension NotificationName {
    static let documentationLaunched = NotificationNames(rawValue:
"DocumentationLaunchedNotification")
}
```

RawRepresentable

```
func post(notification notification: NotificationName) -> Void {
    // use notification.rawValue
}
```

NotificationName

```
post(notification: .dataFinished)
```

RawRepresentable

```
// RawRepresentable has an associate type, so the
// method that wants to accept any type conforming to
// RawRepresentable needs to be generic
func observe<T: RawRepresentable>(object: T) -> Void {
    // object.rawValue
}
```

class◦ ◦

```
protocol ClassOnlyProtocol: class, SomeOtherProtocol {
    // Protocol requirements
}
```

ClassOnlyProtocol ◦

```
struct MyStruct: ClassOnlyProtocol {
    // error: Non-class type 'MyStruct' cannot conform to class protocol 'ClassOnlyProtocol'
```

```
}
```

ClassOnlyProtocol ◦

```
protocol MyProtocol: ClassOnlyProtocol {
    // ClassOnlyProtocol Requirements
    // MyProtocol Requirements
}

class MySecondClass: MyProtocol {
    // ClassOnlyProtocol Requirements
    // MyProtocol Requirements
}
```

◦

```
protocol Foo : class {
    var bar : String { get set }
}

func takesAFoo(foo:Foo) {

    // this assignment requires reference semantics,
    // as foo is a let constant in this scope.
    foo.bar = "new value"
}
```

Foobarfoo◦

Foo - var◦

```
protocol Foo {
    var bar : String { get set }
}

func takesAFoo(foo:Foo) {
    foo.bar = "new value" // error: Cannot assign to property: 'foo' is a 'let' constant
}
```

```
func takesAFoo(foo:Foo) {
    var foo = foo // mutable copy of foo
    foo.bar = "new value" // no error - satisfies both reference and value semantics
}
```

weakweak◦

```
weak var weakReference : ClassOnlyProtocol?
```

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.

MirrorSwiftstruct ◦ ◦ Core Data◦ structNSManagedObject ◦

2.

MirrorchildrenMirror◦ 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◦

```
let sampleProject = Project()
sampleProject.title = "MirrorMirror"
sampleProject.id = 199
sampleProject.platform = "iOS"
sampleProject.version = 2
sampleProject.info = "test app for Reflection"
```

Mirror◦ childrenAnyForwardCollection<Child> ChildsubjecttypealiasChildlabel: Stringvalue: Any ◦

```
let projectMirror = Mirror(reflecting: sampleProject)
let properties = projectMirror.children

print(properties.count)        //5
print(properties.first?.label) //Optional("title")
print(properties.first!.value) //MirrorMirror
print()

for property in properties {
```

```
    print("\(property.label!):\(property.value)")
}
```

XcodePlaygroundConsolefor for。

```
title:MirrorMirror
id:199
platform:iOS
version:2
info:Optional("test app for Reflection")
```

Xcode 8 beta 2Playground

`value.dynamicType` Swift 3 `type(of: value)` Swift 2 `value.dynamicType` Swift`Mirror` 。

`NSObject` `class_copyPropertyListproperty_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

```
let primitiveDataTypes: Dictionary<String, Any> = [
    "c" : Int8.self,
    "s" : Int16.self,
    "i" : Int32.self,
    "q" : Int.self, //also: Int64, NSInteger, only true on 64 bit platforms
    "S" : UInt16.self,
```

```
    "I" : UInt32.self,
    "Q" : UInt.self, //also UInt64, only true on 64 bit platforms
    "B" : Bool.self,
    "d" : Double.self,
    "f" : Float.self,
    "{" : Decimal.self
]

   func getNameOf(property: objc_property_t) -> String? {
        guard let name: NSString = NSString(utf8String: property_getName(property)) else {
return nil }
        return name as String
    }
```

NSObject.TypeNSObjectNSDate Swift3 Date NSString Swift3 String NSNumber AnyDictionary。 IntInt32 Bool value types。 NSObject .selfInt- Int.self.selfNSObject.TypeAny。 Dictionary<String, Any>? Dictionary<String, NSObject.Type>? 。

```
class Book: NSObject {
    let title: String
    let author: String?
    let numberOfPages: Int
    let released: Date
    let isPocket: Bool

    init(title: String, author: String?, numberOfPages: Int, released: Date, isPocket: Bool) {
        self.title = title
        self.author = author
        self.numberOfPages = numberOfPages
        self.released = released
        self.isPocket = isPocket
    }
}

guard let types = getTypesOfProperties(in: Book.self) else { return }
for (name, type) in types {
    print("'\(name)' has type '\(type)'")
}
// Prints:
// 'title' has type 'NSString'
// 'numberOfPages' has type 'Int'
// 'author' has type 'NSString'
// 'released' has type 'NSDate'
// 'isPocket' has type 'Bool'
```

AnyNSObject.Type NSObjectstandard ==

```
func checkPropertiesOfBook() {
    guard let types = getTypesOfProperties(in: Book.self) else { return }
    for (name, type) in types {
        if let objectType = type as? NSObject.Type {
            if objectType == NSDate.self {
                print("Property named '\(name)' has type 'NSDate'")
            } else if objectType == NSString.self {
                print("Property named '\(name)' has type 'NSString'")
            }
        }
    }
```

```
}
```

==

```
func ==(rhs: Any, lhs: Any) -> Bool {
    let rhsType: String = "\(rhs)"
    let lhsType: String = "\(lhs)"
    let same = rhsType == lhsType
    return same
}
```

value types

```
func checkPropertiesOfBook() {
    guard let types = getTypesOfProperties(in: Book.self) else { return }
    for (name, type) in types {
        if type == Int.self {
            print("Property named '\(name)' has type 'Int'")
        } else if type == Bool.self {
            print("Property named '\(name)' has type 'Bool'")
        }
    }
}
```

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 = [Int: String]()
// or
var books: [Int: String] = [:]
```

。 。

```
var books: [Int: String] = [1: "Book 1", 2: "Book 2"]
//books = [2: "Book 2", 1: "Book 1"]
var otherBooks = [3: "Book 3", 4: "Book 4"]
//otherBooks = [3: "Book 3", 4: "Book 4"]
```

Dictionary

```
var books = [Int: String]()
//books = [:]
books[5] = "Book 5"
//books = [5: "Book 5"]
books.updateValue("Book 6", forKey: 5)
//[5: "Book 6"]
```

updateValue。

```
let previousValue = books.updateValue("Book 7", forKey: 5)
//books = [5: "Book 7"]
//previousValue = "Book 6"
```

```
books[5] = nil
//books [:]
books[6] = "Deleting from Dictionaries"
//books = [6: "Deleting from Dictionaries"]
let removedBook = books.removeValueForKey(6)
//books = [:]
//removedValue = "Deleting from Dictionaries"
```

Dictionary

```
var books: [Int: String] = [1: "Book 1", 2: "Book 2"]
```

```
let bookName = books[1]
//bookName = "Book 1"
```

valuesvalues

```
for book in books.values {
    print("Book Title: \(book)")
}
//output: Book Title: Book 2
//output: Book Title: Book 1
```

keyskeys

```
for bookNumbers in books.keys {
    print("Book number: \(bookNumber)")
}
// outputs:
// Book number: 1
// Book number: 2
```

keyvalue

```
for (book,bookNumbers)in books{
print("\(book)  \(bookNumbers)")
}
// outputs:
// 2  Book 2
// 1  Book 1
```

Array Dictionary - ◦

Dictionary◦

```
// Create a multilevel dictionary.
var myDictionary: [String:[Int:String]]! =
["Toys":[1:"Car",2:"Truck"],"Interests":[1:"Science",2:"Math"]]

print(myDictionary["Toys"][2]) // Outputs "Truck"
print(myDictionary["Interests"][1]) // Outputs "Science"
```

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 chr: Character = "H" // valid
let chr2: Character = "" // valid
let chr3: Character = "abc" // invalid - multiple grapheme clusters
```

。 。

\(value) 。 。

```
let number = 5
let interpolatedNumber = "\(number)"  // string is "5"
let fortyTwo = "\(6 * 7)"             // string is "42"

let example = "This post has \(number) view\(number == 1 ? "" : "s")"
// It will output "This post has 5 views" for the above example.
// If the variable number had the value 1, it would output "This post has 1 view" instead.
```

"\(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 message = "Then he said, \"I \u{1F496} you!\""

print(message) // Then he said, "I 💖 you!"
```

+

```
let name = "John"
let surname = "Appleseed"
let fullName = name + " " + surname  // fullName is "John Appleseed"
```

+=

```
let str2 = "there"
var instruction = "look over"
instruction += " " + str2  // instruction is now "look over there"

var instruction = "look over"
instruction.append(" " + str2)  // instruction is now "look over there"
```

```
var greeting: String = "Hello"
let exclamationMark: Character = "!"
greeting.append(exclamationMark)
// produces a modified String (greeting) = "Hello!"
```

```
var alphabet:String = "my ABCs: "
alphabet.append(contentsOf: (0x61...0x7A).map(UnicodeScalar.init)
                                         .map(Character.init) )
```

```
// produces a modified string (alphabet) = "my ABCs: abcdefghijklmnopqrstuvwxyz"
```

3.0

`appendContentsOf(_:)` `append(_:)` ◦

`joinWithSeparator(_:)`

```
let words = ["apple", "orange", "banana"]
let str = words.joinWithSeparator(" & ")

print(str)   // "apple & orange & banana"
```

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

```
"abc" == "def"          // false
"abc" == "ABC"          // false
"abc" == "abc"          // true

// "LATIN SMALL LETTER A WITH ACUTE" == "LATIN SMALL LETTER A" + "COMBINING ACUTE ACCENT"
"\u{e1}" == "a\u{301}"  // true
```

/

```
"fortitude".hasPrefix("fort")      // true
"Swift Language".hasSuffix("age")  // true
```

Swift StringUnicode。 ◦

```
let str = "ที่①!"
```

`characters`Unicode

```
Array(str.characters)  // ["ที่", "", "①", "!"]
```

`unicodeScalars`Unicode ที่3--3607,3637,3656 - `characters`

```
str.unicodeScalars.map{ $0.value }  // [3607, 3637, 3656, 128076, 9312, 33]
```

UTF-8 `UInt8`UTF-16 `UInt16`

```
Array(str.utf8)   // [224, 184, 151, 224, 184, 181, 224, 185, 136, 240, 159, 145, 140, 226,
145, 160, 33]
Array(str.utf16)  // [3607, 3637, 3656, 55357, 56396, 9312, 33]
```

`characters` `unicodeScalars` `utf8`,`utf16`Collection `count`

```
// NOTE: These operations are NOT necessarily fast/cheap!

str.characters.count     // 4
str.unicodeScalars.count // 6
str.utf8.count           // 17
str.utf16.count          // 7
```

```
for c in str.characters { // ...
for u in str.unicodeScalars { // ...
for byte in str.utf8 { // ...
for byte in str.utf16 { // ...
```

Unicode

```
var str: String = "I want to visit □□, Москва, मुंबई, القاهرة, and □□□. □"
var character: Character = "□"
```

```
var str: String = "\u{61}\u{5927}\u{1F34E}\u{3C0}" // a□□π
var character: Character = "\u{65}\u{301}" // é = "e" + accent mark
```

Swift `Character`Unicode。 。

- >

```
// Accesses views of different Unicode encodings of `str`
str.utf8
str.utf16
str.unicodeScalars // UTF-32
```

- >

```
let value0: UInt8 = 0x61
let value1: UInt16 = 0x5927
let value2: UInt32 = 0x1F34E

let string0 = String(UnicodeScalar(value0)) // a
let string1 = String(UnicodeScalar(value1)) // □
```

```
let string2 = String(UnicodeScalar(value2)) // 

// convert hex array to String
let myHexArray = [0x43, 0x61, 0x74, 0x203C, 0x1F431] // an Int array
var myString = ""
for hexValue in myHexArray {
    myString.append(UnicodeScalar(hexValue))
}
print(myString) // Cat!!
```

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 reversedCharacters = aString.characters.reversed()
let reversedString = String(reversedCharacters)
```

2.2

```
let text = "AaBbCc"
let uppercase = text.uppercaseString // "AABBCC"
let lowercase = text.lowercaseString // "aabbcc"
```

3.0

```
let text = "AaBbCc"
let uppercase = text.uppercased() // "AABBCC"
let lowercase = text.lowercased() // "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 = NSCharacterSet.letterCharacterSet()

let phrase = "Test case"
let range = phrase.rangeOfCharacterFromSet(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 phrase = "Test case"
let charset = CharacterSet(charactersIn: "t")

if let _ = phrase.rangeOfCharacter(from: charset, options: .caseInsensitive) {
   print("yes")
}
else {
   print("no")
}
```

2.2

```
let charset = NSCharacterSet(charactersInString: "t")

if let _ = phrase.rangeOfCharacterFromSet(charset, options: .CaseInsensitiveSearch, range:
nil) {
   print("yes")
}
else {
    print("no")
}
```

3.0

```
let phrase = "Test case"
let charset = CharacterSet(charactersIn: "t")
```

```
if let _ = phrase.rangeOfCharacter(from: charset, options: .caseInsensitive, range:
phrase.startIndex..<phrase.endIndex)) {
   print("yes")
}
else {
   print("no")
}
```

StringCharacter

```
let text = "Hello World"
let char: Character = "o"
```

CharacterString

```
let sensitiveCount = text.characters.filter { $0 == char }.count // case-sensitive
let insensitiveCount = text.lowercaseString.characters.filter { $0 ==
Character(String(char).lowercaseString) } // case-insensitive
```

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: text, set: chars)
```

```
let number: Int = 7
let str1 = String(format: "%03d", number) // 007
let str2 = String(format: "%05d", number) // 00007
```

```
let number: Float = 3.14159
let str1 = String(format: "%.2f", number) // 3.14
let str2 = String(format: "%.4f", number) // 3.1416 (rounded)
```

```
let number: Int = 13627
let str1 = String(format: "%2X", number) // 353B
```

```
let str2 = String(format: "%2x", number) // 353b (notice the lowercase b)
```

```
let number: Int = 13627
let str1 = String(number, radix: 16, uppercase: true) //353B
let str2 = String(number, radix: 16) // 353b
```

```
let number: Int = 13627
let str1 = String(number, radix: 36) // aij
```

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()
}
```

`endIndex` `string[string.endIndex]` `string[string.startIndex]` ◦ `""` `string.startIndex == string.endIndex` `true` ◦ `startIndex.successor()` ◦

3.0

Swift 3String`successor()` `predecessor()` `advancedBy(_:)` `advancedBy(_:limit:)` `distanceTo(_:)` ◦

◦

`.index(after:)` ,. `.index(before:)` `.index(_:, offsetBy:)` ◦

```
let string = "My fantastic string"
var currentIndex = string.startIndex

while currentIndex != string.endIndex {
    print(string[currentIndex])
    currentIndex = string.index(after: currentIndex)
}
```

currentIndex.index。

3.0

```
var index:String.Index? = string.endIndex.predecessor()

while index != nil {
    print(string[index!])
    if index != string.startIndex {
        index = index.predecessor()
    }
    else {
        index = nil
    }
}
```

3.0

```
var currentIndex: String.Index? = string.index(before: string.endIndex)

while currentIndex != nil {
    print(string[currentIndex!])
    if currentIndex != string.startIndex {
        currentIndex = string.index(before: currentIndex!)
    }
    else {
        currentIndex = nil
    }
}
```

IndexInt 。

```
let string = "My string"
string[2] // can't do this
string.characters[2] // and also can't do this
```

3.0

```
index = string.startIndex.advanceBy(2)
```

3.0

```
currentIndex = string.index(string.startIndex, offsetBy: 2)
```

3.0

```
index = string.endIndex.advancedBy(-2)
```

3.0

```
currentIndex = string.index(string.endIndex, offsetBy: -2)
```

3.0

```
index = string.startIndex.advanceBy(20, limit: string.endIndex)
```

3.0

```
currentIndex = string.index(string.startIndex, offsetBy: 20, limitedBy: string.endIndex)
```

```
for c in string.characters {
    print(c)
}
```

WhiteSpaceNewLine

3.0

```
let someString = "  Swift Language  \n"
let trimmedString =
someString.stringByTrimmingCharactersInSet(NSCharacterSet.whitespaceAndNewlineCharacterSet())
// "Swift Language"
```

`stringByTrimmingCharactersInSet`String。

。

```
let trimmedWhiteSpace =
someString.stringByTrimmingCharactersInSet(NSCharacterSet.whitespaceCharacterSet())
// "Swift Language  \n"
```

```
let trimmedNewLine =
someString.stringByTrimmingCharactersInSet(NSCharacterSet.newlineCharacterSet())
// "  Swift Language  "
```

3.0

```
let someString = "  Swift Language  \n"

let trimmedString = someString.trimmingCharacters(in: .whitespacesAndNewlines)
// "Swift Language"

let trimmedWhiteSpace = someString.trimmingCharacters(in: .whitespaces)
// "Swift Language  \n"

let trimmedNewLine = someString.trimmingCharacters(in: .newlines)
// "  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.dataUsingEncoding(NSUTF8StringEncoding)
```

`Data` / `NSData` to `String`

3.0

```
let string = String(data: data, encoding: .utf8)
```

2.2

```
let string = String(data: data, encoding: NSUTF8StringEncoding)
```

SwiftStringString

3.0

```
let startDate = "23:51"

let startDateAsArray = startDate.components(separatedBy: ":") // ["23", "51"]`
```

2.2

```
let startDate = "23:51"

let startArray = startDate.componentsSeparatedByString(":") // ["23", "51"]`
```

3.0

```
let myText = "MyText"

let myTextArray = myText.components(separatedBy: " ") // myTextArray is ["MyText"]
```

2.2

```
let myText = "MyText"

let myTextArray = myText.componentsSeparatedByString(" ") // myTextArray is ["MyText"]
```

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
```

```
func hash(name:String, data:Data) -> Data? {
    let algos = ["MD2":    (CC_MD2,    CC_MD2_DIGEST_LENGTH),
                 "MD4":    (CC_MD4,    CC_MD4_DIGEST_LENGTH),
                 "MD5":    (CC_MD5,    CC_MD5_DIGEST_LENGTH),
                 "SHA1":   (CC_SHA1,   CC_SHA1_DIGEST_LENGTH),
                 "SHA224": (CC_SHA224, CC_SHA224_DIGEST_LENGTH),
                 "SHA256": (CC_SHA256, CC_SHA256_DIGEST_LENGTH),
                 "SHA384": (CC_SHA384, CC_SHA384_DIGEST_LENGTH),
                 "SHA512": (CC_SHA512, CC_SHA512_DIGEST_LENGTH)]
    guard let (hashAlgorithm, length) = algos[name]  else { return nil }
    var hashData = Data(count: Int(length))

    _ = hashData.withUnsafeMutableBytes {digestBytes in
        data.withUnsafeBytes {messageBytes in
            hashAlgorithm(messageBytes, CC_LONG(data.count), digestBytes)
        }
    }
    return hashData
}
```

String

```
name: A name of a hash function as a String
string: The String to be hashed
returns: the hashed result as Data
```

```
func hash(name:String, string:String) -> Data? {
    let data = string.data(using:.utf8)!
    return hash(name:name, data:data)
```

```
}
```

```
let clearString = "clearData0123456"
let clearData   = clearString.data(using:.utf8)!
print("clearString: \(clearString)")
print("clearData: \(clearData as NSData)")

let hashSHA256 = hash(name:"SHA256", string:clearString)
print("hashSHA256: \(hashSHA256! as NSData)")

let hashMD5 = hash(name:"MD5", data:clearData)
print("hashMD5: \(hashMD5! as NSData)")
```

```
clearString: clearData0123456
clearData: <636c6561 72446174 61303132 33343536>

hashSHA256: <aabc766b 6b357564 e41f4f91 2d494bcc bfa16924 b574abbd ba9e3e9d a0c8920a>
hashMD5: <4df665f7 b94aea69 695b0e7b baf9e9d6>
```

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
```

```
func hmac(hashName:String, message:Data, key:Data) -> Data? {
    let algos = ["SHA1":   (kCCHmacAlgSHA1,   CC_SHA1_DIGEST_LENGTH),
                 "MD5":    (kCCHmacAlgMD5,    CC_MD5_DIGEST_LENGTH),
                 "SHA224": (kCCHmacAlgSHA224, CC_SHA224_DIGEST_LENGTH),
                 "SHA256": (kCCHmacAlgSHA256, CC_SHA256_DIGEST_LENGTH),
                 "SHA384": (kCCHmacAlgSHA384, CC_SHA384_DIGEST_LENGTH),
                 "SHA512": (kCCHmacAlgSHA512, CC_SHA512_DIGEST_LENGTH)]
    guard let (hashAlgorithm, length) = algos[hashName]  else { return nil }
    var macData = Data(count: Int(length))

    macData.withUnsafeMutableBytes {macBytes in
        message.withUnsafeBytes {messageBytes in
            key.withUnsafeBytes {keyBytes in
                CCHmac(CCHmacAlgorithm(hashAlgorithm),
                       keyBytes,     key.count,
                       messageBytes, message.count,
                       macBytes)
            }
        }
```

```
    }
    return macData
}
```

```
hashName: name of a hash function as String
message:  message as String
key:      key as String
returns:  digest as Data
```

```
func hmac(hashName:String, message:String, key:String) -> Data? {
    let messageData = message.data(using:.utf8)!
    let keyData = key.data(using:.utf8)!
    return hmac(hashName:hashName, message:messageData, key:keyData)
}
```

```
hashName: name of a hash function as String
message:  message as String
key:      key as Data
returns:  digest as Data
```

```
func hmac(hashName:String, message:String, key:Data) -> Data? {
    let messageData = message.data(using:.utf8)!
    return hmac(hashName:hashName, message:messageData, key:key)
}
```

//

```
let clearString = "clearData0123456"
let keyString   = "keyData8901234562"
let clearData   = clearString.data(using:.utf8)!
let keyData     = keyString.data(using:.utf8)!
print("clearString: \(clearString)")
print("keyString:   \(keyString)")
print("clearData: \(clearData as NSData)")
print("keyData:   \(keyData as NSData)")

let hmacData1 = hmac(hashName:"SHA1", message:clearData, key:keyData)
print("hmacData1: \(hmacData1! as NSData)")

let hmacData2 = hmac(hashName:"SHA1", message:clearString, key:keyString)
print("hmacData2: \(hmacData2! as NSData)")

let hmacData3 = hmac(hashName:"SHA1", message:clearString, key:keyData)
print("hmacData3: \(hmacData3! as NSData)")
```

```
clearString: clearData0123456
keyString:   keyData8901234562
clearData: <636c6561 72446174 61303132 33343536>
keyData:   <6b657944 61746138 39303132 33343536 32>

hmacData1: <bb358f41 79b68c08 8e93191a da7dabbc 138f2ae6>
hmacData2: <bb358f41 79b68c08 8e93191a da7dabbc 138f2ae6>
hmacData3: <bb358f41 79b68c08 8e93191a da7dabbc 138f2ae6>
```

https://riptutorial.com/zh-TW/swift/topic/7885/

29:

Examples

Bool

Bool `truefalse`。

```
let aTrueBool = true
let aFalseBool = false
```

Bools。 `if`

```
func test(_ someBoolean: Bool) {
    if someBoolean {
        print("IT'S TRUE!")
    }
    else {
        print("IT'S FALSE!")
    }
}
test(aTrueBool)  // prints "IT'S TRUE!"
```

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。

```
if question {
    answerIfTrue
} else {
    answerIfFalse
}
```

```
func isTurtle(_ value: Bool) {
    let color = value ? "green" : "red"
    print("The animal is \(color)")
}

isTurtle(true) // outputs 'The animal is green'
isTurtle(false) // outputs 'The animal is red'
```

```
func actionDark() {
    print("Welcome to the dark side")
}

func actionJedi() {
    print("Welcome to the Jedi order")
}

func welcome(_ isJedi: Bool) {
    isJedi ? actionJedi() : actionDark()
}

welcome(true) // outputs 'Welcome to the Jedi order'
welcome(false) // outputs 'Welcome to the dark side'
```

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 write(data: UnsafePointer<UInt8>, dataLength: Int) throws {
    var stream:NSOutputStream = getOutputStream()
    defer {
        stream.close()
    }

    let written = stream.write(data, maxLength: dataLength)
    guard written >= 0 else {
        throw stream.streamError! /* 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
```

```
let names = ["James", "Emily", "Miles"]

for name in names {
   print(name)
}

// James
// Emily
// Miles
```

2.1 2.2

`SequenceTypeenumerate()`。

```
for (index, name) in names.enumerate() {
   print("The index of \(name) is \(index).")
}
```

```
// The index of James is 0.
// The index of Emily is 1.
// The index of Miles is 2.
```

`enumerate()` `Int` 0。 - 。

3.0

Swift 3 `enumerate()` `enumerated()`

```
for (index, name) in names.enumerated() {
   print("The index of \(name) is \(index).")
}
```

```
let ages = ["James": 29, "Emily": 24]

for (name, age) in ages {
    print(name, "is", age, "years old.")
}

// Emily is 24 years old.
// James is 29 years old.
```

2.1 2.2

`SequenceType` `reverse()`

```
for i in (0..<3).reverse() {
    print(i)
}

for i in (0...2).reverse() {
    print(i)
}

// Both print:
// 2
// 1
// 0

let names = ["James", "Emily", "Miles"]

for name in names.reverse() {
    print(name)
}

// Miles
// Emily
// James
```

3.0

Swift 3 `reverse()` `reversed()`

```
for i in (0..<3).reversed() {
```

```
    print(i)
}
```

2.1 2.2

`Strideablestride(_:_:)`

```
for i in 4.stride(to: 0, by: -2) {
    print(i)
}

// 4
// 2

for i in 4.stride(through: 0, by: -2) {
    print(i)
}

// 4
// 2
// 0
```

1.2 3.0

Swift 3 `Stridablestride(_:_:)``stride(_:_:_:)`

```
for i in stride(from: 4, to: 0, by: -2) {
    print(i)
}

for i in stride(from: 4, through: 0, by: -2) {
    print(i)
}
```

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 { item in
    print(item)
}
```

*。returncontinue。。

```
let arr = [1,2,3,4]

arr.forEach {

    // blocks for 3 and 4 will still be called
    if $0 == 2 {
        return
    }
}
```

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**

```
let points = [(5, 0), (31, 0), (5, 31)]
for case (_, 0) in points {
    print("point on x-axis")
}

//point on x-axis
//point on x-axis
```

?

```
let optionalNumbers = [31, 5, nil]
for case let number? in optionalNumbers {
    print(number)
}

//31
//5
```

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。 。 。 。

```
struct MyStruct {

    let myProperty: Int

}
```

。 /。

```
struct MyStruct {

    let myProperty: MyClass

}
```

。 struct。

。

https://riptutorial.com/zh-TW/swift/topic/4067/

34:

Swift。

Examples

/get。

```
extension ExtensionOf {
    //new functions and get-variables
}
```

`self`

`String.length()`

```
extension String {
    func length() -> Int {
        return self.characters.count
    }
}

"Hello, World!".length() // 13
```

get。 `.length`

```
extension String {
    var length: Int {
        get {
            return self.characters.count
        }
    }
}

"Hello, World!".length // 13
```

。 `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。

```
let val1: Int = 10

val1.factorial  // returns 3628800
```

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
    }
}
```

```
let array = ["a","b","c"]
let resultado = array.toUpperCase()
```

。 。

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(offset: Int) -> Character {
        let newIndex = self.index(self.startIndex, offsetBy: offset)
        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 decimal = 10               // ten
let decimal = -1000            // negative one thousand
let decimal = -1_000           // equivalent to -1000
let decimal = 42_42_42         // equivalent to 424242
let decimal = 0755             // equivalent to 755, NOT 493 as in some other languages
let decimal = 0123456789

let hexadecimal = 0x10         // equivalent to 16
let hexadecimal = 0x7FFFFFFF
let hexadecimal = 0xBadFace
let hexadecimal = 0x0123_4567_89ab_cdef

let octal = 0o10               // equivalent to 8
let octal = 0o755              // equivalent to 493
let octal = -0o0123_4567

let binary = -0b101010         // equivalent to -42
let binary = 0b111_101_101     // equivalent to 0o755
let binary = 0b1011_1010_1101  // equivalent to 0xB_A_D
```

```
let decimal = 0.0
let decimal = -42.0123456789
let decimal = 1_000.234_567_89

let decimal = 4.567e5               // equivalent to 4.567×10⁵, or 456_700.0
let decimal = -2E-4                 // equivalent to -2×10⁻⁴, or -0.0002
let decimal = 1e+0                  // equivalent to 1×10⁰, or 1.0

let hexadecimal = 0x1p0             // equivalent to 1×2⁰, or 1.0
let hexadecimal = 0x1p-2            // equivalent to 1×2⁻², or 0.25
let hexadecimal = 0xFEEDp+3         // equivalent to 65261×2³, or 522088.0
let hexadecimal = 0x1234.5P4        // equivalent to 0x12345, or 74565.0
```

```
let hexadecimal = 0x123.45P8        // equivalent to 0x12345, or 74565.0
let hexadecimal = 0x12.345P12       // equivalent to 0x12345, or 74565.0
let hexadecimal = 0x1.2345P16       // equivalent to 0x12345, or 74565.0
let hexadecimal = 0x0.12345P20      // equivalent to 0x12345, or 74565.0
```

```
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
```

```
Int8(-129.0) // fatal error: floating point value cannot be converted to Int8 because it is
less than Int8.min
Int8(-129)   // crash: EXC_BAD_INSTRUCTION / SIGILL
Int8(-128)   // ok
Int8(-2)     // ok
Int8(17)     // ok
Int8(127)    // ok
Int8(128)    // crash: EXC_BAD_INSTRUCTION / SIGILL
Int8(128.0)  // fatal error: floating point value cannot be converted to Int8 because it is
greater than Int8.max
```

```
Int(-2.2)  // -2
Int(-1.9)  // -1
Int(-0.1)  //  0
Int(1.0)   //  1
Int(1.2)   //  1
Int(1.9)   //  1
Int(2.0)   //  2
```

```
Int(Float(1_000_000_000_000_000_000))  // 999999984306749440
```

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"
```

```
let x = 42, y = 9001
"Between \(x) and \(y)"  // equivalent to "Between 42 and 9001"
```

```
if let num = Int("42") { /* ... */ }               // num is 42
if let num = Int("Z2cF") { /* ... */ }             // returns nil (not a number)
if let num = Int("z2cf", radix: 36) { /* ... */ }  // num is 1635999
if let num = Int("Z2cF", radix: 36) { /* ... */ }  // num is 1635999
if let num = Int8("Z2cF", radix: 36) { /* ... */ }  // returns nil (too large for Int8)
```

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
```

。

```
ceil(3.000) // 3
ceil(3.001) // 4
ceil(3.999) // 4

ceil(-3.000) // -3
ceil(-3.001) // -3
ceil(-3.999) // -3
```

。

```
floor(3.000) // 3
floor(3.001) // 3
floor(3.999) // 3

floor(-3.000) // -3
floor(-3.001) // -4
floor(-3.999) // -4
```

DoubleInt 。

```
Int(3.000) // 3
Int(3.001) // 3
Int(3.999) // 3

Int(-3.000) // -3
Int(-3.001) // -3
Int(-3.999) // -3
```

- round ceilfloor6432。

```
arc4random_uniform(someNumber: UInt32) -> UInt32
```

0someNumber - 1◦

UInt324,294,967,2952^32 - 1 ◦

- ```
  let flip = arc4random_uniform(2) // 0 or 1
  ```
- ```
  let roll = arc4random_uniform(6) + 1 // 1...6
  ```
- ```
  let day = arc4random_uniform(31) + 1 // 1...31
  ```
- 2090

  ```
  let year = 1990 + arc4random_uniform(10)
  ```

```
let number = min + arc4random_uniform(max - min + 1)
```

number maxminUInt32 ◦

- arc4randomarc4random_uniform◦
- UInt32Int◦

Swiftpow()Double

```
pow(BASE, EXPONENT)
```

base52

```
let number = pow(5.0, 2.0) // Equals 25
```

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
```

```
// Create an immutable array of type [Int] containing 2, 4, and 7
let arrayOfInts = [2, 4, 7]
```

```
let arrayOfUInt8s: [UInt8] = [2, 4, 7]   // type annotation on the variable
```

```
let arrayOfUInt8s = [2, 4, 7] as [UInt8] // type annotation on the initializer expression
let arrayOfUInt8s = [2 as UInt8, 4, 7]   // explicit for one element, inferred for the others
```

```
// An immutable array of type [String], containing ["Example", "Example", "Example"]
let arrayOfStrings = Array(repeating: "Example",count: 3)
```

```
let dictionary = ["foo" : 4, "bar" : 6]

// An immutable array of type [(String, Int)], containing [("bar", 6), ("foo", 4)]
let arrayOfKeyValuePairs = Array(dictionary)
```

Swift `Int[[Int]]` `Array<Array<Int>>` ◦

```
let array2x3 = [
    [1, 2, 3],
    [4, 5, 6]
]
// array2x3[0][1] is 2, and array2x3[1][2] is 6.
```

```
var array3x4x5 = Array(repeating: Array(repeating: Array(repeating: 0,count: 5),count:
4),count: 3)
```

```
var exampleArray:[Int] = [1,2,3,4,5]
//exampleArray = [1, 2, 3, 4, 5]
```

```
let exampleOne = exampleArray[2]
//exampleOne = 3
```

2`Array` ◦ `Array` `Array`0◦

```
let value0 = exampleArray[0]
let value1 = exampleArray[1]
let value2 = exampleArray[2]
let value3 = exampleArray[3]
let value4 = exampleArray[4]
//value0 = 1
//value1 = 2
//value2 = 3
//value3 = 4
//value4 = 5
```

`Array`

```
var filteredArray = exampleArray.filter({ $0 < 4 })
//filteredArray = [1, 2, 3]
```

```
var evenArray = exampleArray.filter({ $0 % 2 == 0 })
//evenArray = [2, 4]
```

nil ◦

```
exampleArray.indexOf(3) // Optional(2)
```

Array◦ Arraynil ◦

```
exampleArray.first // Optional(1)
exampleArray.last // Optional(5)
exampleArray.maxElement() // Optional(5)
exampleArray.minElement() // Optional(1)
```

```
var exampleArray = [1,2,3,4,5]
exampleArray.isEmpty //false
exampleArray.count //5
```

◦

```
exampleArray = exampleArray.reverse()
//exampleArray = [9, 8, 7, 6, 5, 3, 2]
```

```
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]
```

```
exampleArray.removeAtIndex(3)
//exampleArray = [1, 2, 3, 5, 6, 7, 8, 9, 10]
exampleArray.removeLast()
//exampleArray = [1, 2, 3, 5, 6, 7, 8, 9]
exampleArray.removeFirst()
//exampleArray = [2, 3, 5, 6, 7, 8, 9]
```

```
var array = [3, 2, 1]
```

ArraySequenceType ◦

2.1 2.2

Swift 2sort()◦

```
let sorted = array.sort()  // [1, 2, 3]
```

3.0

Swift 3sorted() ◦

```
let sorted = array.sorted()  // [1, 2, 3]
```

ArrayMutableCollectionType ◦

2.1 2.2

Swift 2sortInPlace()◦

```
array.sortInPlace() // [1, 2, 3]
```

3.0

Swift 3sort() ◦

```
array.sort() // [1, 2, 3]
```

Comparable◦

- Comparable◦ LandmarkComparable - ◦

```
struct Landmark {
    let name : String
    let metersTall : Int
}

var landmarks = [Landmark(name: "Empire State Building", metersTall: 443),
                 Landmark(name: "Eifell Tower", metersTall: 300),
                 Landmark(name: "The Shard", metersTall: 310)]
```

2.1 2.2

```
// sort landmarks by height (ascending)
landmarks.sortInPlace {$0.metersTall < $1.metersTall}

print(landmarks) // [Landmark(name: "Eifell Tower", metersTall: 300), Landmark(name: "The
Shard", metersTall: 310), Landmark(name: "Empire State Building", metersTall: 443)]

// create new array of landmarks sorted by name
let alphabeticalLandmarks = landmarks.sort {$0.name < $1.name}

print(alphabeticalLandmarks) // [Landmark(name: "Eifell Tower", metersTall: 300),
Landmark(name: "Empire State Building", metersTall: 443), Landmark(name: "The Shard",
metersTall: 310)]
```

3.0

```
// sort landmarks by height (ascending)
landmarks.sort {$0.metersTall < $1.metersTall}

// create new array of landmarks sorted by name
let alphabeticalLandmarks = landmarks.sorted {$0.name < $1.name}
```

◦

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 words = numbers.map(String.init)
```

map(_:) transform - Int2

```
let numbers = [1, 2, 3, 4, 5]
let numbersTimes2 = numbers.map {$0 * 2}
print(numbersTimes2) // [2, 4, 6, 8, 10]
```

flatMap_ :)Array

things Array Any◦

```
let things: [Any] = [1, "Hello", 2, true, false, "World", 3]
```

◦ Int(s)Int Array◦

```
let numbers = things.flatMap { $0 as? Int }
```

numbers[Int] ◦ flatMapnil

```
[1, 2, 3]
```

SequenceTypefilter(_:)◦

[Int]

```
let numbers = [22, 41, 23, 30]

let evenNumbers = numbers.filter { $0 % 2 == 0 }

print(evenNumbers)  // [22, 30]
```

30[Person]

```
struct Person {
```

```
    var age : Int
}

let people = [Person(age: 22), Person(age: 41), Person(age: 23), Person(age: 30)]

let peopleYoungerThan30 = people.filter { $0.age < 30 }

print(peopleYoungerThan30) // [Person(age: 22), Person(age: 23)]
```

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]

```
let strings = ["1", "foo", "3", "4", "bar", "6"]

let numbersThatCanBeConverted = strings.flatMap { Int($0) }

print(numbersThatCanBeConverted) // [1, 3, 4, 6]
```

flatMap(_:)nil

```
let optionalNumbers : [Int?] = [nil, 1, nil, 2, nil, 3]

let numbers = optionalNumbers.flatMap { $0 }

print(numbers) // [1, 2, 3]
```

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 result = Array(slice)
```

```
let words = ["Hey", "Hello", "Bonjour", "Welcome", "Hi", "Hola"]
let selectedWords = Array(words[2...4]) // ["Bonjour", "Welcome", "Hi"]
```

```
struct Box {
    let name: String
    let thingsInside: Int
}
```

Box(es)

```
let boxes = [
    Box(name: "Box 0", thingsInside: 1),
    Box(name: "Box 1", thingsInside: 2),
    Box(name: "Box 2", thingsInside: 3),
    Box(name: "Box 3", thingsInside: 1),
    Box(name: "Box 4", thingsInside: 2),
    Box(name: "Box 5", thingsInside: 3),
    Box(name: "Box 6", thingsInside: 1)
]
```

thingsInsidethingsInside Dictionary key◦

```
let grouped = boxes.reduce([Int:[Box]]()) { (res, box) -> [Int:[Box]] in
    var res = res
    res[box.thingsInside] = (res[box.thingsInside] ?? []) + [box]
    return res
}
```

[Int:[Box]]

```
[
    2: [Box(name: "Box 1", thingsInside: 2), Box(name: "Box 4", thingsInside: 2)],
    3: [Box(name: "Box 2", thingsInside: 3), Box(name: "Box 5", thingsInside: 3)],
    1: [Box(name: "Box 0", thingsInside: 1), Box(name: "Box 3", thingsInside: 1), Box(name:
"Box 6", thingsInside: 1)]
]
```

flatMap_ :)

nil flatMap(_:)S

```
extension SequenceType {
    public func flatMap<S : SequenceType>(transform: (Self.Generator.Element) throws -> S)
rethrows -> [S.Generator.Element]
}
```

- [S.Generator.Element] ◦

```
let primes = ["2", "3", "5", "7", "11", "13", "17", "19"]
let allCharacters = primes.flatMap { $0.characters }
// => "["2", "3", "5", "7", "1", "1", "1", "3", "1", "7", "1", "9"]"
```

1. primes[String] flatMap(_:) ◦
2.

primesString Array<String>.Generator.Element ◦

3. String.CharacterView◦
4. - [String.CharacterView.Generator.Element] ◦

flatMap(_:) - 2D1D3D2D◦

$0

```
// A 2D array of type [[Int]]
let array2D = [[1, 3], [4], [6, 8, 10], [11]]

// A 1D array of type [Int]
let flattenedArray = array2D.flatMap { $0 }

print(flattenedArray) // [1, 3, 4, 6, 8, 10, 11]
```

3.0

sorted()

```
let words = ["Hello", "Bonjour", "Salute", "Ahola"]
let sortedWords = words.sorted()
print(sortedWords) // ["Ahola", "Bonjour", "Hello", "Salute"]
```

sort()

```
var mutableWords = ["Hello", "Bonjour", "Salute", "Ahola"]
mutableWords.sort()
print(mutableWords) // ["Ahola", "Bonjour", "Hello", "Salute"]
```

```
let words = ["Hello", "Bonjour", "Salute", "Ahola"]
let sortedWords = words.sorted(isOrderedBefore: { $0 > $1 })
print(sortedWords) // ["Salute", "Hello", "Bonjour", "Ahola"]
```

```
let words = ["Hello", "Bonjour", "Salute", "Ahola"]
let sortedWords = words.sorted() { $0 > $1 }
print(sortedWords) // ["Salute", "Hello", "Bonjour", "Ahola"]
```

```
let words = ["Hello", "bonjour", "Salute", "ahola"]
let unexpected = words.sorted()
print(unexpected) // ["Hello", "Salute", "ahola", "bonjour"]
```

```
let words = ["Hello", "bonjour", "Salute", "ahola"]
let sortedWords = words.sorted { $0.lowercased() < $1.lowercased() }
print(sortedWords) // ["ahola", "bonjour", "Hello", "Salute"]
```

import FoundationNSStringcaseInsensitiveCompare

```
let words = ["Hello", "bonjour", "Salute", "ahola"]
let sortedWords = words.sorted { $0.caseInsensitiveCompare($1) == .orderedAscending }
print(sortedWords) // ["ahola", "bonjour", "Hello", "Salute"]
```

localizedCaseInsensitiveCompare。

.numericcompare

```
let files = ["File-42.txt", "File-01.txt", "File-5.txt", "File-007.txt", "File-10.txt"]
let sortedFiles = files.sorted() { $0.compare($1, options: .numeric) == .orderedAscending }
print(sortedFiles) // ["File-01.txt", "File-5.txt", "File-007.txt", "File-10.txt", "File-
42.txt"]
```

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 sum = numbers.reduce(0, combine: +)
```

remove(at:)。

Swift3

```
extension Array where Element: Equatable {

    mutating func remove(_ element: Element) {
```

```
        _ = index(of: element).flatMap {
            self.remove(at: $0)
        }
    }
}
```

```
    var array = ["abc", "lmn", "pqr", "stu", "xyz"]
    array.remove("lmn")
    print("\(array)")    //["abc", "pqr", "stu", "xyz"]

    array.remove("nonexistent")
    print("\(array)")    //["abc", "pqr", "stu", "xyz"]
    //if provided element value is not present, then it will do nothing!
```

```
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 minimumNumber = numbers.min() // Optional(1)
let maximumNumber = numbers.max() // Optional(25)
```

- `nil`。

`Comparable`。

```
Comparable。
```

```
struct Vector2 {
    let dx : Double
    let dy : Double

    var magnitude : Double {return sqrt(dx*dx+dy*dy)}
}

let vectors = [Vector2(dx: 3, dy: 2), Vector2(dx: 1, dy: 1), Vector2(dx: 2, dy: 2)]
```

2.1 2.2

```
// Vector2(dx: 1.0, dy: 1.0)
let lowestMagnitudeVec2 = vectors.minElement { $0.magnitude < $1.magnitude }
```

```
// Vector2(dx: 3.0, dy: 2.0)
let highestMagnitudeVec2 = vectors.maxElement { $0.magnitude < $1.magnitude }
```

3.0

```
let lowestMagnitudeVec2 = vectors.min { $0.magnitude < $1.magnitude }
let highestMagnitudeVec2 = vectors.max { $0.magnitude < $1.magnitude }
```

。

```
extension Array {
    subscript (safe index: Int) -> Element? {
        return indices ~= index ? self[index] : nil
    }
}
```

```
if let thirdValue = array[safe: 2] {
    print(thirdValue)
}
```

2

zipSequenceType2Zip2Sequence。

```
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。

2Int(s)

```
let list0 = [0, 2, 4]
let list1 = [0, 4, 8]
```

list1list0list0。

```
let list1HasDoubleOfList0 = !zip(list0, list1).filter { $0 != (2 * $1)}.isEmpty
```

https://riptutorial.com/zh-TW/swift/topic/284/

37:

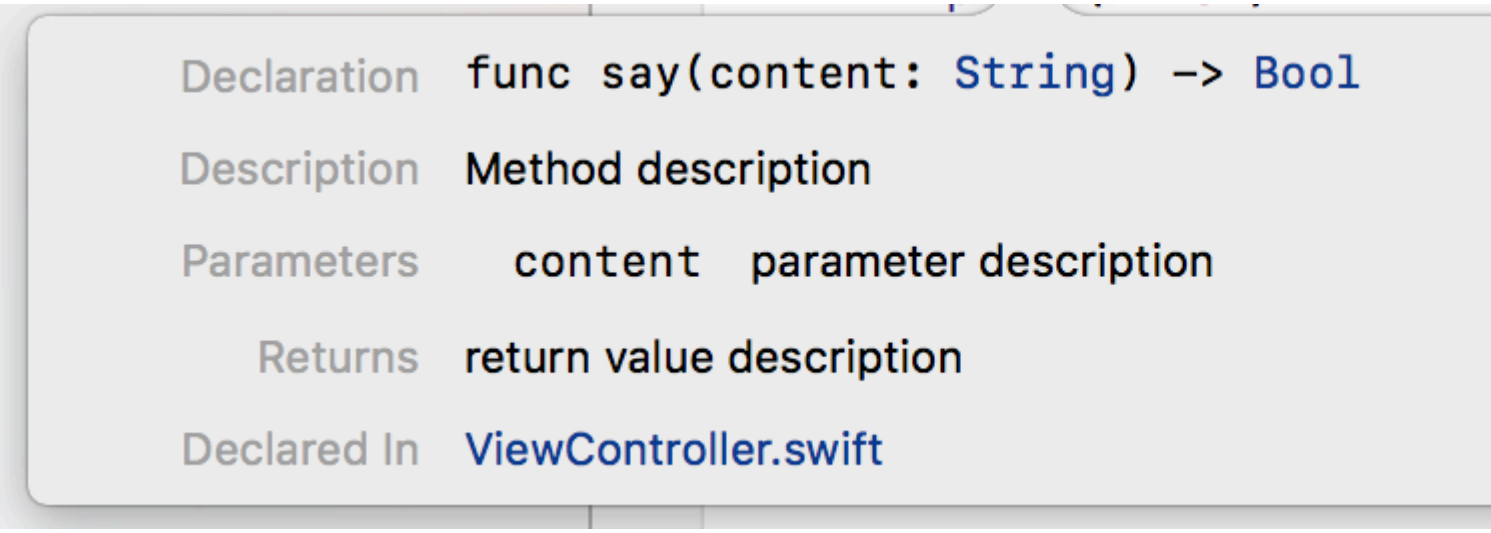
Examples

```
/// 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 □□ + □□ + /。

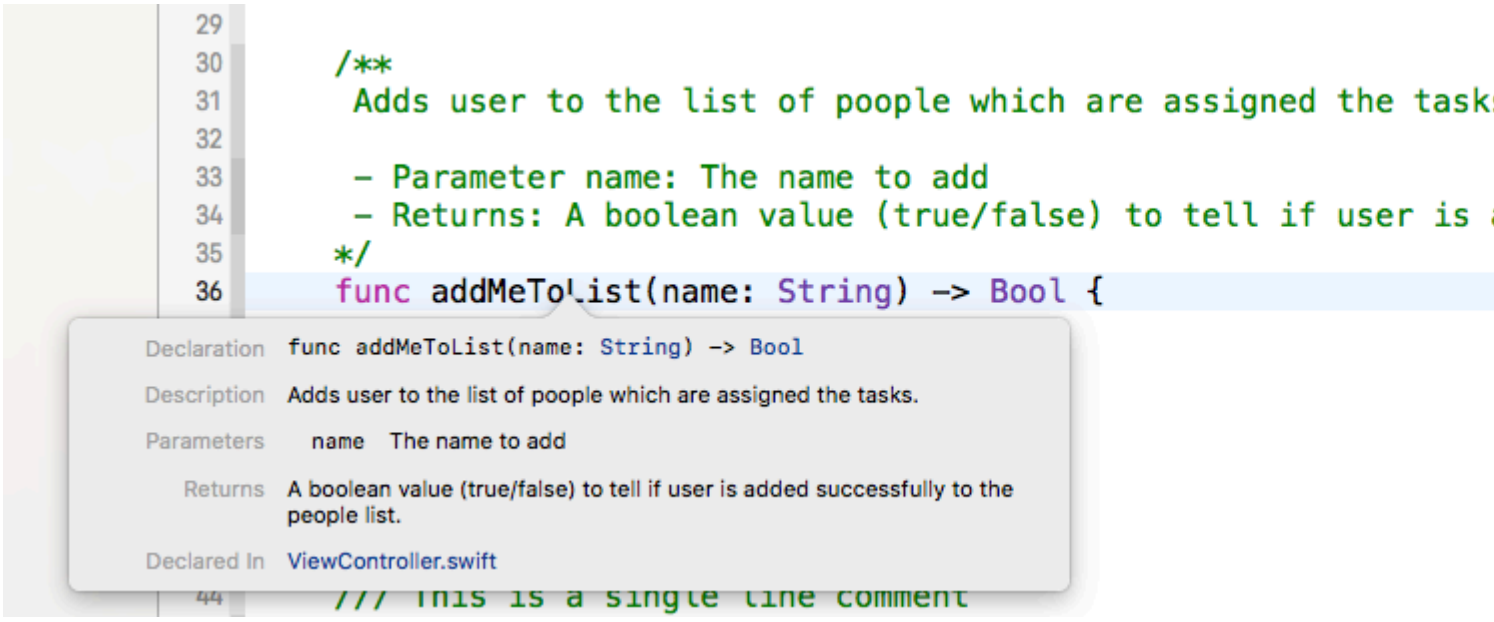



```
/**
 Adds user to the list of poople which are assigned the tasks.

 - 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
}
```




```
/// This is a single line comment
func singleLineComment() {

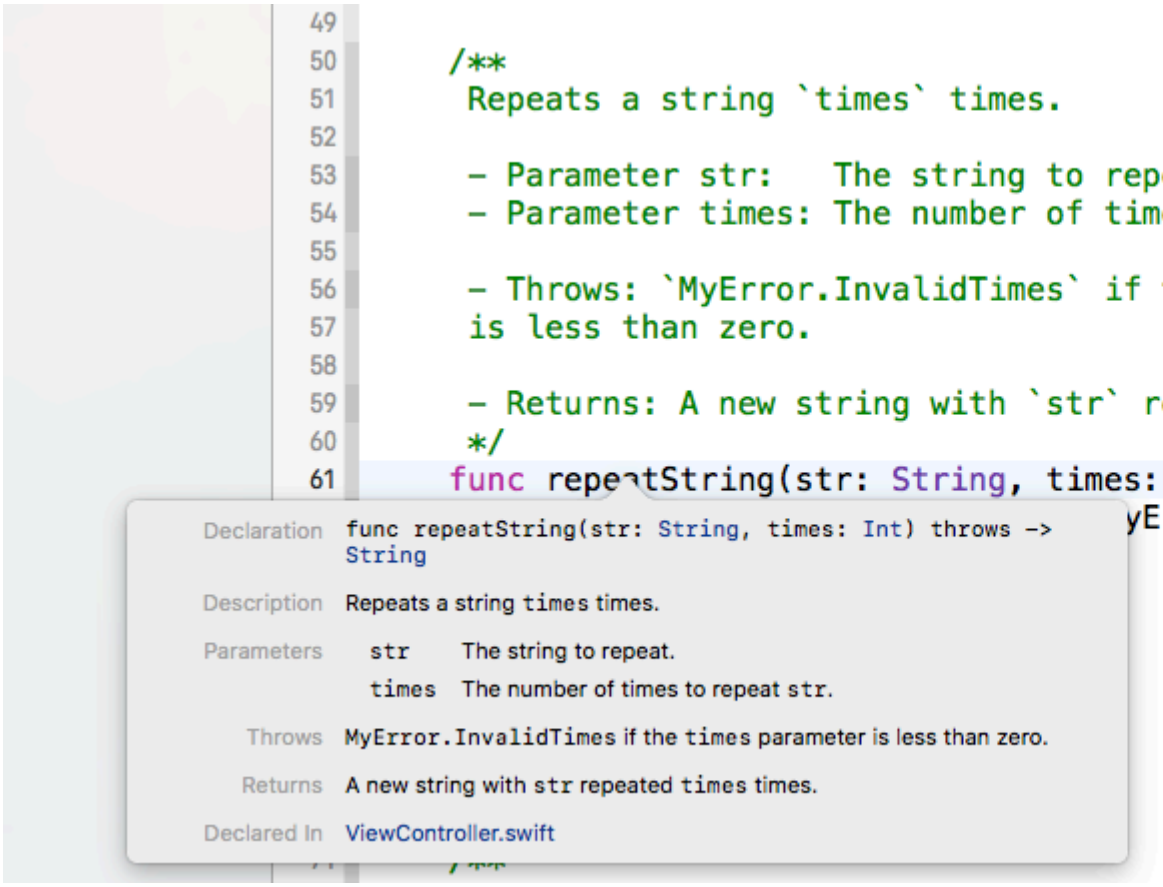
}
```


```
/**
 Repeats a string `times` times.

 - Parameter str:   The string to repeat.
 - Parameter times: The number of times to repeat `str`.

 - Throws: `MyError.InvalidTimes` if the `times` parameter
 is less than zero.

 - Returns: A new string with `str` repeated `times` times.
*/
func repeatString(str: String, times: Int) throws -> String {
    guard times >= 0 else { throw MyError.invalidTimes }
    return "Hello, world"
}
```



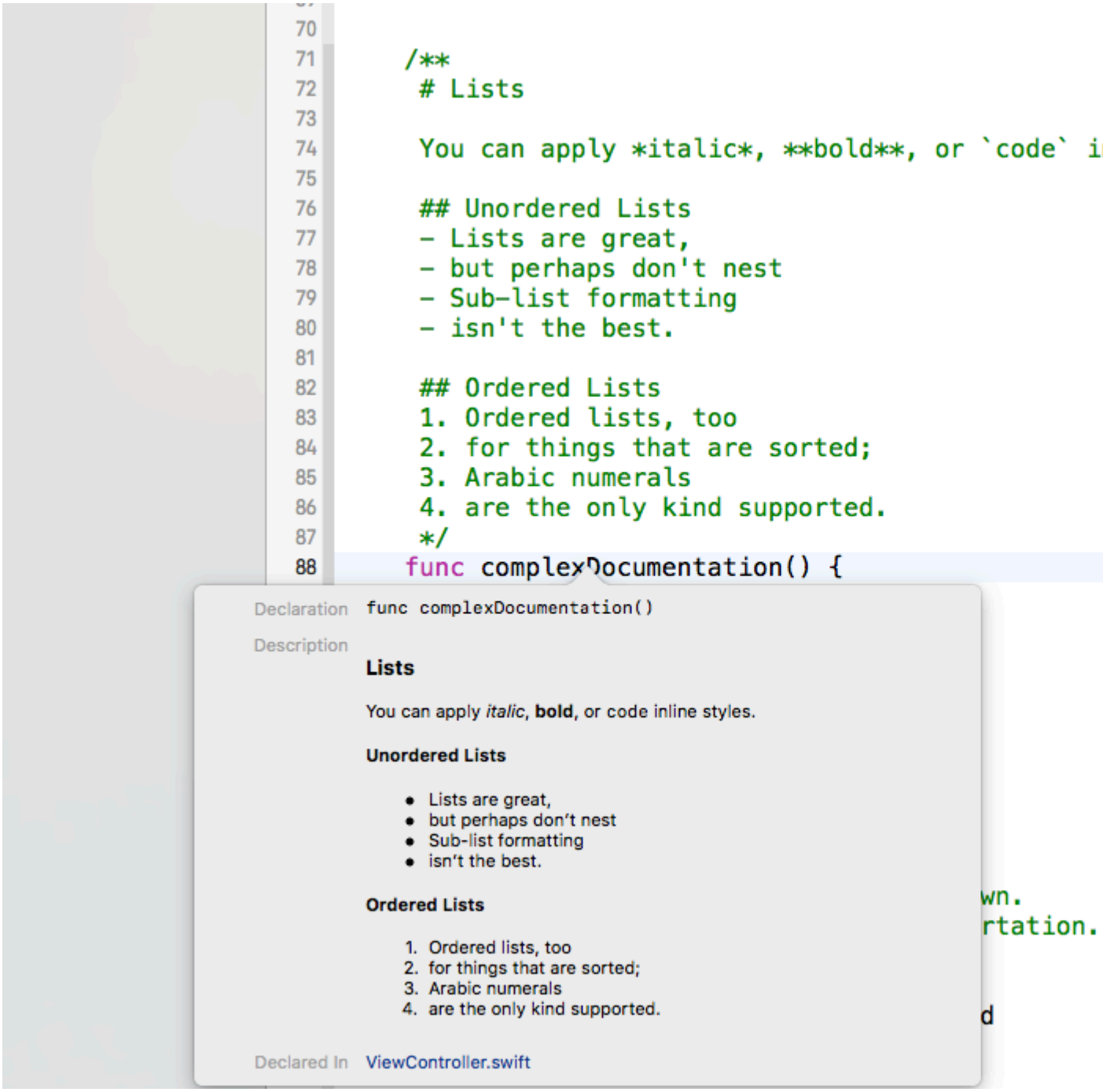

```
/**
 # Lists

 You can apply *italic*, **bold**, or `code` inline styles.

 ## Unordered Lists
 - Lists are great,
 - but perhaps don't nest
 - Sub-list formatting
 - isn't the best.

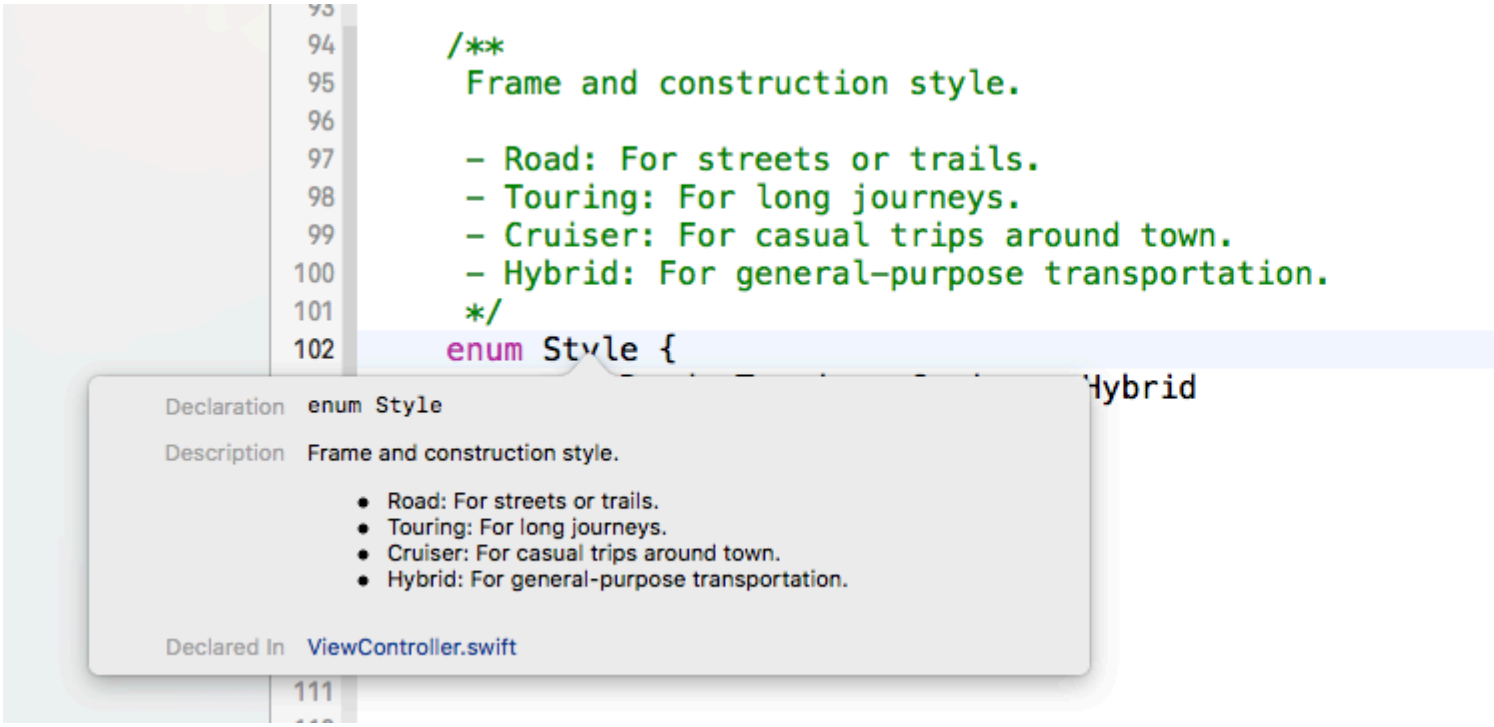
 ## Ordered Lists
 1. Ordered lists, too
 2. for things that are sorted;
 3. Arabic numerals
 4. are the only kind supported.
*/
func complexDocumentation() {

}
```




```
/**
 Frame and construction style.

 - Road: For streets or trails.
 - Touring: For long journeys.
 - Cruiser: For casual trips around town.
 - Hybrid: For general-purpose transportation.
*/
enum Style {
    case Road, Touring, Cruiser, Hybrid
}
```



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 }
```

```
let dir = Direction.up
let dir: Direction = Direction.up
let dir: Direction = .up

// func move(dir: Direction)...
move(Direction.up)
move(.up)

obj.dir = Direction.up
obj.dir = .up
```

/switch

```
switch dir {
case .up:
    // handle the up case
case .down:
    // handle the down case
case .left:
    // handle the left case
case .right:
    // handle the right case
}
```

Hashable Equatable

```
if dir == .down { ... }

let dirs: Set<Direction> = [.right, .left]

print(Direction.up)  // prints "up"
debugPrint(Direction.up)  // prints "Direction.up"
```

```
enum Action {
    case jump
```

```
    case kick
    case move(distance: Float)  // The "move" case has an associated distance
}
```

```
performAction(.jump)
performAction(.kick)
performAction(.move(distance: 3.3))
performAction(.move(distance: 0.5))
```

switch

```
switch action {
case .jump:
    ...
case .kick:
    ...
case .move(let distance):  // or case let .move(distance):
    print("Moving: \(distance)")
}
```

if case

```
if case .move(let distance) = action {
    print("Moving: \(distance)")
}
```

guard case

```
guard case .move(let distance) = action else {
    print("Action is not move")
    return
}
```

Equatable ◦ ==

```
extension Action: Equatable { }

func ==(lhs: Action, rhs: Action) -> Bool {
    switch lhs {
    case .jump: if case .jump = rhs { return true }
    case .kick: if case .kick = rhs { return true }
    case .move(let lhsDistance): if case .move (let rhsDistance) = rhs { return lhsDistance ==
rhsDistance }
    }
    return false
}
```

```
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)
    indirect case branch(Tree<T>, Tree<T>)
}

let tree = Tree.branch(.leaf(1), .branch(.leaf(2), .leaf(3)))
```

`indirect`

```
indirect enum Tree<T> {
    case leaf(T)
    case branch(Tree<T>, Tree<T>)
}
```

```
enum Rotation: Int {
    case up = 0
    case left = 90
    case upsideDown = 180
    case right = 270
}
```

rawValue

```
enum Rotation {
    case up
    case right
    case down
    case left
}

let foo = Rotation.up
foo.rawValue //error
```

0

```
enum MetasyntacticVariable: Int {
    case foo  // rawValue is automatically 0
    case bar  // rawValue is automatically 1
    case baz = 7
    case quux  // rawValue is automatically 8
}
```

```
enum MarsMoon: String {
    case phobos  // rawValue is automatically "phobos"
    case deimos  // rawValue is automatically "deimos"
}
```

RawRepresentable ◦ `.rawValue`

```
func rotate(rotation: Rotation) {
    let degrees = rotation.rawValue
    ...
}
```

`init?(rawValue:)`

```
let rotation = Rotation(rawValue: 0)  // returns Rotation.Up
let otherRotation = Rotation(rawValue: 45)  // returns nil (there is no Rotation with rawValue
45)

if let moon = MarsMoon(rawValue: str) {
    print("Mars has a moon named \(str)")
} else {
    print("Mars doesn't have a moon named \(str)")
}
```

hashValue。

```
let quux = MetasyntacticVariable(rawValue: 8)// rawValue is 8
quux?.hashValue //hashValue is 3
```

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
        }
    }
}
```

```
var direction = CompassDirection(degrees: 0) // Returns CompassDirection.north
```

```
direction = CompassDirection(degrees: 90) // Returns CompassDirection.east
print(direction.value) //prints 90
direction = CompassDirection(degrees: 500) // Returns nil
```

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
    }
}
```

```
var dir = Direction(oppositeTo: .down) // Direction.up

dir.changeDirection() // Direction.left

let opposite = dir.opposite // Direction.right
```

。

```
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
    }
}
```

```
let instrment1 = Orchestra.Strings.viola
let instrment2 = Orchestra.Keyboards.piano
```

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 printOptionalNum(num: Int?) {
    guard let unwrappedNum = num else {
        print("num does not exist")
        return
    }
    print(unwrappedNum)
}
```

Guard`where`

```
func printOptionalNum(num: Int?) {
guard let unwrappedNum = num, unwrappedNum == 10 else {
    print("num does not exist or is not 10")
    return
}
print(unwrappedNum)
}
```

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 {  // Execute the following code if the first condition is true.
    print("num is less than 10")
} else if num == 10 {  // Or, if not, check the next condition...
    print("num is 10")
} else {  // If all else fails...
    print("all other conditions were false, so num is greater than 10")
}
```

&&||

AND

```
let num = 10
let str = "Hi"
if num == 10 && str == "Hi" {
    print("num is 10, AND str is \"Hi\"")
}
```

*num == 10*false。 。

OR

```
if num == 10 || str == "Hi" {
    print("num is 10, or str is \"Hi\")
}
```

*num == 10*true。

NOT

```
if !str.isEmpty {
    print("str is not empty")
}
```

“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")
}
```

```
// num originally has type Int?
if let num = num {
    // num has type Int inside this block
}
```

1.2 3.0

,

```
if let unwrappedNum = num, let unwrappedStr = str {
    // Do something with unwrappedNum & unwrappedStr
} else if let unwrappedNum = num {
    // Do something with unwrappedNum
} else {
    // num was nil
}
```

where

```
if let unwrappedNum = num where unwrappedNum % 2 == 0 {
    print("num is non-nil, and it's an even number")
}
```

where

```
if let num = num                            // num must be non-nil
    where num % 2 == 1,                     // num must be odd
    let str = str,                          // str must be non-nil
    let firstChar = str.characters.first    // str must also be non-empty
    where firstChar != "x"                  // the first character must not be "x"
{
    // all bindings & conditions succeeded!
}
```

3.0

3 where SE-0099 ,◦

```
if let unwrappedNum = num, unwrappedNum % 2 == 0 {
    print("num is non-nil, and it's an even number")
}

if let num = num,                           // num must be non-nil
    num % 2 == 1,                           // num must be odd
```

```
    let str = str,                           // str must be non-nil
    let firstChar = str.characters.first,   // str must also be non-empty
    firstChar != "x"                         // the first character must not be "x"
{
    // all bindings & conditions succeeded!
}
```

□□□□□

□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□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
}
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□true□□false。 □□□□□□□□□□□□□□□□□

```
(<CONDITION>) ? <TRUE VALUE> : <FALSE VALUE>
```

□□□□□□□□□□□□□□□□□□□□□□□□□

```
let a = 5
let b = 10
let min = a < b ? a : b
let max = a > b ? a : b
```

□□□□□□□□□□□a <b。 □□□□□□□□□□min□□□□□a;□□□□□□□□□□□b□□。

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□Swift□□□□□□□□□□□□□□□□max□min。

Nil-Coalescing□□□

□□□□□□<OPTIONAL> ?? <DEFAULT VALUE>□□<OPTIONAL>□□□□□□□<OPTIONAL>□□□n□□□□<DEFAULT VALUE>。 <OPTIONAL>□□□□□□□。 <DEFAULT VALUE>□□□<OPTIONAL>□□□□□□□。

nil-coalescing□□□□□□□□□□□□□□□□□□□□

```
a != nil ? a! : b
```

□□□□□□□□□□□□

```
(a ?? b) == (a != nil ? a! : b) // ouputs true
```

□□□□

```
let defaultSpeed:String = "Slow"
var userEnteredSpeed:String? = nil

print(userEnteredSpeed ?? defaultSpeed) // ouputs "Slow"

userEnteredSpeed = "Fast"
print(userEnteredSpeed ?? defaultSpeed) // ouputs "Fast"
```

□□□□□□□□□□ https://riptutorial.com/zh-TW/swift/topic/475/□□□□

[illegible] 41: [illegible]

[illegible]

> [illegible]。[illegible]。
>
> [illegible]Swift[illegible]Swift[illegible]。[illegible]Swift[illegible]Array[illegible]Dictionary[illegible]。[illegible]Int[illegible]String[illegible]Swift[illegible]。[illegible]。
>
> [illegible] Apple[illegible]Swift[illegible]

Examples

[illegible]

[illegible] [illegible] 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
    }
}
```

[illegible]MyGenericClass [illegible]type[illegible]Equatable[illegible]==[illegible]type[illegible]

```
let myFloatGeneric = MyGenericClass<Double>(value: 2.71828) // valid
let myStringGeneric = MyGenericClass<String>(value: "My String") // valid

// "Type [Int] does not conform to protocol 'Equatable'"
let myInvalidGeneric = MyGenericClass<[Int]>(value: [2])

let myIntGeneric = MyGenericClass<Int>(value: 72)
print(myIntGeneric.valueEquals(72)) // true
print(myIntGeneric.valueEquals(-274)) // false

// "Cannot convert value of type 'String' to expected argument type 'Int'"
print(myIntGeneric.valueEquals("My String"))
```

[illegible]

[illegible]。[illegible]Any[illegible]。

[illegible]<>[illegible]。

[illegible]

□□□□ □□□□□□□□□□□□□□

```
/// 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
}
```

□□□□□□□□□□□□□T□□□□□□□□□Swift□□□□□□T□□□□□□□□□□□□□□□□□□□□。

```
let randomOutput = pickRandom(5, 7) // returns an Int (that's either 5 or 7)
```

□□□□□□□□□□□□□□。□□Swift□□T == Int - □□□□□□□□□□(Int, Int) -> Int。

□□□□□□□□□□□□□□□ - □□□□□□□□□□□□□□□□。□□□□□□□□□□□

```
struct Foo {}

let foo = Foo()

let randomOutput = pickRandom(foo, 5) // error: cannot convert value of type 'Int' to expected argument type 'Foo'
```

□□□□

□□□□□□□ □ □□□□□□□□□ □□□□□□□□□□□□□□□□。

```
class Bar<T> {
    var baz : T

    init(baz:T) {
        self.baz = baz
    }
}
```

□□□□□Bar□□□□□□□□□□□□□□□。□□□□□□□□□□□□□init(baz:T)□□□。

```
let bar = Bar(baz: "a string") // bar's type is Bar<String>
```

□□□□□□□□T□□□□String□□□□□□□Bar<String>□□。□□□□□□□□□□□

```
let bar = Bar<String>(baz: "a string")
```

□□。□□□□□□□□□baz □□□□□□□□□□□□□□String□□。

```
let str = bar.baz // of type String
```

□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□。□□□□□□□□□□

```
func takeABarInt(bar:Bar<Int>) {
    ...
}
```

□□□□□□Bar<Int>。□□□□□□□□□□□□□Int□Bar□□□□□□□□□□。

□□□□□□□

□□□□□□□□□□□□□□□□□。□□□。□□□Swift□Array□□□□□Element□□□□□□□□□□□□□□□Array□□□□□□□。

```
public struct Array<Element> : RandomAccessCollection, MutableCollection {
    ...
}
```

□□□□□

□□□□□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
    }
}
```

□□□□□□□□□□□□□□□□□□□□

```
let myStringGeneric = MyGenericClass<String>(value: "My String Value")
let myIntGeneric = MyGenericClass<Int>(value: 42)

print(myStringGeneric.getValue()) // "My String Value"
print(myIntGeneric.getValue()) // 42

myStringGeneric.setValue("Another String")
myIntGeneric.setValue(1024)

print(myStringGeneric.getValue()) // "Another String"
print(myIntGeneric.getValue()) // 1024
```

□□□□□□□□□□□□□□□□□

```
class AnotherGenericClass<TypeOne, TypeTwo, TypeThree>{

    var value1: TypeOne
    var value2: TypeTwo
    var value3: TypeThree
    init(value1: TypeOne, value2: TypeTwo, value3: TypeThree){
        self.value1 = value1
        self.value2 = value2
        self.value3 = value3
    }

    func getValueOne() -> TypeOne{return self.value1}
    func getValueTwo() -> TypeTwo{return self.value2}
    func getValueThree() -> TypeThree{return self.value3}
}
```

□□□□□□□□□□

```
let myGeneric = AnotherGenericClass<String, Int, Double>(value1: "Value of pi", value2: 3,
value3: 3.14159)

print(myGeneric.getValueOne() is String) // true
print(myGeneric.getValueTwo() is Int) // true
print(myGeneric.getValueThree() is Double) // true
print(myGeneric.getValueTwo() is String) // false

print(myGeneric.getValueOne()) // "Value of pi"
print(myGeneric.getValueTwo()) // 3
print(myGeneric.getValueThree()) // 3.14159
```

□□□□□

□□□□□□□□□

```
// 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")
```

```
        }
    }
}
```

□□

```
var myArray = [1,2,3]
print(myArray)

// Prints [1,2,3]
```

□□□□□□□□□□□□□□□□。□□□□□□□□□□□□□□□。

```
myArray.remove(2)
print(myArray)

// Prints [1,3]
```

□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
protocol JSONDecodable {
    static func from(_ json: [String: Any]) -> Any?
}
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□。

```
let myTestObject = TestObject.from(myJson) as? TestObject
```

□□□□□□□□□□□□□TestObject □□□□□□□□□□□Any□□□□。

□□□

```
protocol JSONDecodable {
    associatedtype Element
    static func from(_ json: [String: Any]) -> Element?
}

struct TestObject: JSONDecodable {
    static func from(_ json: [String: Any]) -> TestObject? {
    }
}

let testObject = TestObject.from(myJson) // testObject is now automatically `TestObject?`
```

□□□□□□□

□□□□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>(first: T, second: T) where T: Comparable, T: Hashable {
    // Access hashable function
    guard first.hashValue == second.hashValue else {
        return
    }
    // Access comparable function
    if first == second {
        print("\(first) and \(second) are equal.")
    }
}
```

□□□□□□□□□□□□□□□□。□□□□□□□□□□□□□□□□□□□

```
// "Element" is the generics type defined by "Array". For this example, we
// want to add a function that requires that "Element" can be compared, that
// is: it needs to adhere to the Equatable protocol.
public extension Array where Element: Equatable {
    /// Removes the given object from the array.
    mutating func remove(_ element: Element) {
        // We could also use "self.index(of: element)" here, as "index(of:_)"
        // is also defined in an extension with "where Element: Equatable".
        // For the sake of this example, explicitly make use of the Equatable.
        if let index = self.index(where: { $0 == element }) {
            self.remove(at: index)
        } else {
            fatalError("Removal error, no such element:\"\(element)\" in array.\n")
        }
    }
}
```

□□□□□□□□ 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)
```

□□

```
>>> Hello
>>> "Hello"
>>> [foo: 1, bar: 2]
>>> ["foo": 1, "bar": 2]
```

□□□□□□□□□□□□□□□□

```
let wordArray = ["foo", "bar", "food, bars"]

print(wordArray)
debugPrint(wordArray)
```

□□

```
>>> [foo, bar, food, bars]
>>> ["foo", "bar", "food, bars"]
```

□□□□□□□□□□□□□□□□□□□□□□□□4□□□□□□□□3□□□。□□□□□□□□□□□□□□□□□□debugPrint

□□□□□□□□□

```
struct Foo: Printable, DebugPrintable {
    var description: String {return "Clear description of the object"}
    var debugDescription: String {return "Helpful message for debugging"}
```

```
}

var foo = Foo()

print(foo)
debugPrint(foo)

>>> Clear description of the object
>>> Helpful message for debugging
```

□□

dump□□□□□□□□□□□□□□□□□□□□。

□□□□□□□□□

```
let names = ["Joe", "Jane", "Jim", "Joyce"]
dump(names)
```

□□□

```
▿4□□□
- [0]□□
- [1]□□
- [2]□□□
- [3]□□□□
```

□□□□□

```
let attributes = ["foo": 10, "bar": 33, "baz": 42]
dump(attributes)
```

□□□

```
▿3□□/□□
▿[0] : ( 2□□□□
- .0□□
- .1□33
▿[1] : ( 2□□□□
- .0□baz
- .1□42
▿[2] : ( 2□□□□
- .0□foo
- .1□10
```

dump□□□□dump(_:name:indent:maxDepth:maxItems:)。

□□□□□□□□□。

□□□□□□□□□□□□name □□□□□□□□□□□□□□□□

```
dump(attributes, name: "mirroring")
```

□□□

```
▿□□□3□□/□□
▿[0] : ( 2□□□□
- .0□□
- .1□33
▿[1] : ( 2□□□□
- .0□baz
```

```
      - .1： 42
      ▿[2] :( 2□□□□
      - .0： foo
      - .1： 10
```

□□□□□□□□□□□□□□□□□□□□□□maxItems:□□□□□□□□□□□□□maxDepth:□□□□□□□□□□□□□indent:。

print□□vs dump□□

□□□□□□□□□□□□□□□□□print()□□。□□□□□□□□□□□□□

```
class Abc {
    let a = "aa"
    let b = "bb"
}
```

□□□□□Abc□□□

```
let abc = Abc()
```

□□□□□□□□□print()□□□□□

```
App.Abc
```

□dump()□□

```
App.Abc #0
- a: "aa"
- b: "bb"
```

□□□□□ dump()□□□□□□□□□□□□□print()□□□□□。

□□□ dump()□UI□□□□□□

```
let view = UIView(frame: CGRect(x: 0, y: 0, width: 100, height: 100))
```

□□dump(view)□□□□□

```
 - <UIView: 0x108a0cde0; frame = (0 0; 100 100); layer = <CALayer: 0x159340cb0>> #0
    - super: UIResponder
      - NSObject
```

print(view)□□□□□

```
<UIView: 0x108a0cde0; frame = (0 0; 100 100); layer = <CALayer: 0x159340cb0>>
```

□□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 13□14□38.582 ProjetName [2286□7473287] [1,2,3,4,5]
```

□□□□□**ProjectName**□□□□□。

2 Only String□ NSLog()□□String□□□□□□print()□□□□□□□□□□□□□□□□□□□。
□□

```
let array = [1, 2, 3, 4, 5]
print(array) //prints [1, 2, 3, 4, 5]
NSLog(array) //error: Cannot convert value of type [Int] to expected argument type 'String'
```

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 self□UIViewController”□□□□□□□□□□□□□□□□。

□□□□□□□□□“p”□

```
import Foundation
import UIKit
import ObjectiveC          // don't forget this

var _Handle: UInt8 = 42    // it can be any value

protocol Able: class {
    var click:UIView? { get set }
    var x:CGFloat? { get set }
    // note that you >> do not << declare p here
}

extension Able where Self:UIViewController {

    var p:YourType { // YourType might be, say, an Enum
      get {
          return objc_getAssociatedObject(self, & _Handle) as! YourType
          // HOWEVER, SEE BELOW
          }
      set {
          objc_setAssociatedObject(self, & _Handle, newValue, .OBJC_ASSOCIATION_RETAIN)
          // often, you'll want to run some sort of "setter" here...
          __setter()
          }
    }

    func __setter() { something = p.blah() }

    func someOtherExtensionFunction() { p.blah() }
```

```
    // it's ok to use "p" inside other extension functions,
    // and you can use p anywhere in the conforming class
}
```

□□□□□□□□□□□□□□□□“□□”□□□“p”□

□□□□□□□□□□□□□□□□□□□□□“p”。 □□

```
class Clock:UIViewController, Able {
    var u:Int = 0

    func blah() {
      u = ...
      ... = u
      // use "p" as you would any normal property
      p = ...
      ... = p
    }

    override func viewDidLoad() {
      super.viewDidLoad()
      pm = .none // "p" MUST be "initialized" somewhere in Clock
    }
}
```

□□。 □□□□□□□□□。

Xcode □□□□□□□□□□□□□□□□“p”。

□□□“p”□□□□□□□□□□□□viewDidLoad□。

□□□□□□□p□□□□□□□□□□□□ 。 p□□□□□□□□□□□□□□□□。 □□□□□□□□p“□□”□□□□□□□□□□□□□“□p□□□□□□”。 □□□□□Xcode□□□□“□□□p” □□□□□□。

□□□□□□□□□□□□□□□□□□“□□□□□p□□□□□□□□□□□□”。 □□□□□□□□□□□□viewDidLoad□□□。 □

□□□□□□□□。

□□□□□□□□□“p”□□□□□□getter□□□□□ 。

□□□□□□□□□□□□□□□□

```
    get {
        let g = objc_getAssociatedObject(self, &_Handle)
        if (g == nil) {
            objc_setAssociatedObject(self, &_Handle, _default initial value_,
.OBJC_ASSOCIATION)
            return _default initial value_
        }
        return objc_getAssociatedObject(self, &_Handle) as! YourType
        }
```

□□。 Xcode □□□□□□□□□□□□□□□□p。 □□□p□□□□□□□□□□□□viewDidLoad□。

□□□□□□......

□□□□□□□□□□□□□□□

```
func _aoGet(_ ss: Any!, _ handlePointer: UnsafeRawPointer!, _ safeValue: Any!)->Any! {
    let g = objc_getAssociatedObject(ss, handlePointer)
    if (g == nil) {
        objc_setAssociatedObject(ss, handlePointer, safeValue, .OBJC_ASSOCIATION_RETAIN)
```

```
        return safeValue
    }
    return objc_getAssociatedObject(ss, handlePointer)
}

func _aoSet(_ ss: Any!, _ handlePointer: UnsafeRawPointer!, _ val: Any!) {
    objc_setAssociatedObject(ss, handlePointer, val, .OBJC_ASSOCIATION_RETAIN)
}
```

□□。 □□□□□□□□□□□□□□□□□□□□□。□

□□□□□□□□□□

```
protocol PMable: class {
    var click:UILabel? { get set } // ordinary properties here
}

var _pHandle: UInt8 = 321

extension PMable where Self:UIViewController {

    var p:P {
        get {
            return _aoGet(self, &_pHandle, P() ) as! P
        }
        set {
            _aoSet(self, &_pHandle, newValue)
            __pmSetter()
        }
    }

    func __pmSetter() {
        click!.text = String(p)
    }

    func someFunction() {
        p.blah()
    }
}
```

□□_aoGet□□□□□□P□□□□□□□□□□P□□□□□□□□□"”□0□□□□□□□。□

□□□□□□□□□□ https://riptutorial.com/zh-TW/swift/topic/1085/□□□□

第 44 章：

Examples


```
struct Repository {
    let identifier: Int
    let name: String
    var description: String?
}
```

Repository identifier name description。 identifier name let -keyword。。。。

。。

Repository description nil 。

```
let newRepository = Repository(identifier: 0, name: "New Repository", description: "Brand New
Repository")
```



```
first = "Hello"
second = first
first += " World!"
// first == "Hello World!"
// second == "Hello"
```

String。

```
window0 === window1 // works because a window is a class instance
"hello" === "hello" // error: binary operator '===' cannot be applied to two 'String' operands
```

。

。

struct struct mutating

```
struct Counter {
    private var value = 0

    mutating func next() {
        value += 1
    }
}
```


mutating□□□□□□□□□□□□struct□。

```
var counter = Counter()
counter.next()
```

□□□□□□□□□□□□

□□□□□ mutating□□□□□□□□□□□struct□

```
let counter = Counter()
counter.next()
//  error: cannot use mutating member on immutable value: 'counter' is a 'let' constant
```

□□□□□□□

□□□□□□□□□□□□□□□

```
class MyView: NSView { }  // works

struct MyInt: Int { } // error: inheritance from non-protocol type 'Int'
```

□□□□□□□□□□□□□□□

```
struct Vector: Hashable { ... }  // works
```

□□**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)
```

□□□□□□□□□□□□□□□□□□

```
    print(pizzaRange.range) // 200
```

□□□□□□□□□□□□□□□□□□□□□□

```
 print(pizzaRange.center.latitude) // 44.9871
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□。

```
pizzaRange.range = 250
```

□□□□□□□□ https://riptutorial.com/zh-TW/swift/topic/255/□□

第 45 章：

URLSession FileManager

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□□
- private□set□var area = 0

□□

1. □□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□。 □□□□□□□□□□□□□□□□□。

□□□□□□□□□□□□□□□□□□□。 □□□□□□□□□□□□□□□□□□□□□□□□□□。

□□□□□□□□□□□□□□□□□□□。

2. □□□□□□

□□□□□□□□□□□□□□□□□□。

3. ***Getter*□*Setter*□□□**

□□property□setter□private□□getter□internal□□□□□□□。 □□□□□getter□setter□□□□□□。 □□□□□□□□□□

4. □□□□□

□□□□□□□□□Initializers□Protocols□Extensions□Generics□Type 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□□□□□□

```
let myCar = Car("Apple", model: "iCar")
```

Car.make□□□□

```
print(myCar.make)
```

□□□□□□□□□□□□□□□□□□□□□□Car□□。

Car.model□□□□

```
print(myCar.model)
```

□□□□□Car□□□□□□□□□□□□。

Car.otherName□ fileprivate□

```
print(myCar.otherName)
```

□□□□□□Car □□□□□□□□□□。

Car.fullName□□□□

```
print(myCar.fullName)
```

□□Swift 3□□□□□。 private□□□□□□□□struct / class□□。

```
public struct Car {

    public let make: String       //public
    let model: String             //internal
    private let fullName: String! //private

    public init(_ make: String, model model: String) {
        self.make = make
        self.model = model
        self.fullName = "\(make)\(model)"
    }
}
```

□□□□□□□□□□□□□□□□□□□□Swift□□□□□□□□□□□□。□□□□□□□□□□□□□□□□□□□□□□□□□□□□。

□□□□□

```
public class SuperClass {
    private func secretMethod() {}
}

internal class SubClass: SuperClass {
    override internal func secretMethod() {
        super.secretMethod()
    }
}
```

Getters□ Setters□□

```
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

□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□。

□□□□□□□□□□□□□□□□□□Foo□□□□static□□。□□□□ static□□□□□□□□□□□□□□□□□□□□□□□□□□□。

```
public class Foo
{
    static let shared = Foo()

    // Used for preventing the class from being instantiated directly
    private init() {}

    func doSomething()
    {
        print("Do something")
    }
}
```

□□□

```
Foo.shared.doSomething()
```

□□□□□private□□□□□□

> □□。□□□□□□□□□□Swift□□□□□□□□□□□□□□□□□□□□□init□□□□□□□
> 。 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
```

□□□□□□□□□□□□□□□。 □□□□□□□□□□□□□□□□...

```
notifCentre.addObserver(self, selector: #selector(self.myFunc), name: "myNotification", object: nil)
```

□□□□□□□“readForMyFunc”□□□□。 □□□□□□□□□□□□□myFunc□□。 □□□□□□□□□□□□□□

```
func myFunc(){
    print("The notification has been received")
}
```

> □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□。

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
notifCentre.post(name: "myNotification", object: nil)
```

□□□□□□□□□□□□□□□□

```
let myInfo = "pass this on"
notifCentre.post(name: "myNotification", object: ["moreInfo":myInfo])
```

□□□□□□□□□□□□□□□□□

```
func myFunc(_ notification: Notification){
    let userInfo = (notification as NSNotification).userInfo as! [String: AnyObject]
    let passedInfo = userInfo["moreInfo"]
    print("The notification \(moreInfo) has been received")
    //prints - The notification pass this on has been received
}
```

□□□

□□□□□□□□□□□□□□□□□□command□□□□□□□□processing□□□□□□□□□□。□□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□□。

```
struct PurchaseRequest {
  var amount : Float
  var purpose : String
}
```

□□□□□□□□□□□□□□。

```
class ManagerPower : PurchasePower {
  var allowable: Float = 20
  var role : String = "Manager"
  var successor: PurchasePower?
}

class DirectorPower : PurchasePower {
  var allowable: Float = 100
  var role = "Director"
  var successor: PurchasePower?
```

```
}

class PresidentPower : PurchasePower {
  var allowable: Float = 5000
  var role = "President"
  var successor: PurchasePower?
}
```

□□□□□□□□□□□

```
let manager = ManagerPower()
let director = DirectorPower()
let president = PresidentPower()

manager.successor = director
director.successor = president
```

□□□□□□□□□□□□□□

□□□□□□□□□

```
manager.process(request: PurchaseRequest(amount: 2, purpose: "buying a pen"))  // Manager will
approve $ 2.0 for buying a pen
manager.process(request: PurchaseRequest(amount: 90, purpose: "buying a printer")) // Director
will approve $ 90.0 for buying a printer

manager.process(request: PurchaseRequest(amount: 2000, purpose: "invest in stock")) //
President will approve $ 2000.0 for invest in stock
```

□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□。 □□□□

```
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)
  }
```

```
}
```

□□□□

```
let ninjaTurtles = Turtles(turtles: [Turtle(name: "Leo"),
                                     Turtle(name: "Mickey"),
                                     Turtle(name: "Raph"),
                                     Turtle(name: "Doney")])
print("Splinter and")
for turtle in ninjaTurtles {
  print("The great: \(turtle)")
}
```

□□□□□

> □□□□□□□□□□□□□□□□ 。 □□□。 □□□□□□□□□□□□□□□□
> □□□□□□□□□□□□□□□□□□□□□□。 □□。
>
> *-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

    }
}
```

□□□□□□□

```
let aCar = Car(color: UIColor.black,
               numberOfSeats: 4,
               numberOfWheels: 4,
               type: .saloon,
               gearType: .automatic,
               motor: Motor(id: "101", name: "Super Motor",
                            model: "c4", numberOfCylinders: 6),
               shouldHasAirbags: true)

print(aCar)

/* Printing
 color: UIExtendedGrayColorSpace 0 1
 Number of seats: 4
 Number of Wheels: 4
 Type: automatic
 Motor: Motor(id: "101", name: "Super Motor", model: "c4", numberOfCylinders: 6)
 Airbag Availability: true
*/
```

□□□□□□□□□□□□□□□□□□□□□□□□□。

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□。

CarBuilder□□

```
class CarBuilder {
    var color: UIColor = UIColor.black
    var numberOfSeats: UInt8 = 5
```

```
    var numberOfWheels: UInt8 = 4
    var type: CarType = .saloon
    var gearType: GearType = .automatic
    var motor: Motor = Motor(id: "111", name: "Default Motor",
                             model: "T9", numberOfCylinders: 4)
    var shouldHasAirbags: Bool = false

    func buildCar() -> Car {
        return Car(color: color, numberOfSeats: numberOfSeats, numberOfWheels: numberOfWheels,
type: type, gearType: gearType, motor: motor, shouldHasAirbags: shouldHasAirbags)
    }
}
```

CarBuilder□□□□□□□□□□□□□□□□□□car□□□□。

□□□□□CarBuilder□□□□□

```
var builder = CarBuilder()
// currently, the builder creates cars with default configuration.

let defaultCar = builder.buildCar()
//print(defaultCar.description)
/* prints
 color: UIExtendedGrayColorSpace 0 1
 Number of seats: 5
 Number of Wheels: 4
 Type: automatic
 Motor: Motor(id: "111", name: "Default Motor", model: "T9", numberOfCylinders: 4)
 Airbag Availability: false
*/

builder.shouldHasAirbags = true
// now, the builder creates cars with default configuration,
// but with a small edit on making the airbags available

let safeCar = builder.buildCar()
print(safeCar.description)
/* prints
 color: UIExtendedGrayColorSpace 0 1
 Number of seats: 5
 Number of Wheels: 4
 Type: automatic
 Motor: Motor(id: "111", name: "Default Motor", model: "T9", numberOfCylinders: 4)
 Airbag Availability: true
 */

builder.color = UIColor.purple
// now, the builder creates cars with default configuration
// with some extra features: the airbags are available and the color is purple

let femaleCar = builder.buildCar()
print(femaleCar)
/* prints
 color: UIExtendedSRGBColorSpace 0.5 0 0.5 1
 Number of seats: 5
 Number of Wheels: 4
 Type: automatic
 Motor: Motor(id: "111", name: "Default Motor", model: "T9", numberOfCylinders: 4)
 Airbag Availability: true
*/
```

□□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
}

protocol CarBluePrint {
    var color: UIColor { get set }
    var numberOfSeats: UInt8 { get set }
    var numberOfWheels: UInt8 { get set }
    var type: CarType { get set }
    var gearType: GearType { get set }
    var motor: Motor { get set }
    var shouldHasAirbags: Bool { get set }
}

class Car: CustomStringConvertible, CarBluePrint {
    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

    }
}

class CarBuilder: CarBluePrint {
    var color: UIColor = UIColor.black
    var numberOfSeats: UInt8 = 5
    var numberOfWheels: UInt8 = 4
    var type: CarType = .saloon
    var gearType: GearType = .automatic
    var motor: Motor = Motor(id: "111", name: "Default Motor",
                             model: "T9", numberOfCylinders: 4)
    var shouldHasAirbags: Bool = false

    func buildCar() -> Car {
        return Car(color: color, numberOfSeats: numberOfSeats, numberOfWheels: numberOfWheels,
type: type, gearType: gearType, motor: motor, shouldHasAirbags: shouldHasAirbags)
    }
}
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□;□□□□□□□□□□□□□□□□□□□□/□□。

□□□□□□□□□□□□□□□□“□□□□”□□□□□□□□

```
protocol CarBluePrint {
    var color: UIColor { get set }
    var numberOfSeats: UInt8 { get set }
    var numberOfWheels: UInt8 { get set }
    var type: CarType { get set }
    var gearType: GearType { get set }
    var motor: Motor { get set }
    var shouldHasAirbags: Bool { get set }

    // adding the new property
    var batteryName: String { get set }
}
```

□□□□□□□□□□□□□□□□□□□□CarBluePrint□□□□□□'batteryName'□□。 □□□□CarBuilder□□□□□□batteryName□□□□□□
。

□batteryName new□□□□□CarBluePrint□□□□ Car□CarBuilder□□□□□□□□

```
class Car: CustomStringConvertible, CarBluePrint {
    var color: UIColor
    var numberOfSeats: UInt8
    var numberOfWheels: UInt8
    var type: CarType
    var gearType: GearType
    var motor: Motor
    var shouldHasAirbags: Bool
    var batteryName: String

    var description: String {
        return "color: \(color)\nNumber of seats: \(numberOfSeats)\nNumber of Wheels:
\(numberOfWheels)\nType: \(gearType)\nMotor: \(motor)\nAirbag Availability:
\(shouldHasAirbags)\nBattery Name: \(batteryName)"
    }
```

```
    init(color: UIColor, numberOfSeats: UInt8, numberOfWheels: UInt8, type: CarType, gearType:
GearType, motor: Motor, shouldHasAirbags: Bool, batteryName: String) {

        self.color = color
        self.numberOfSeats = numberOfSeats
        self.numberOfWheels = numberOfWheels
        self.type = type
        self.gearType = gearType
        self.motor = motor
        self.shouldHasAirbags = shouldHasAirbags
        self.batteryName = batteryName
    }
}

class CarBuilder: CarBluePrint {
    var color: UIColor = UIColor.red
    var numberOfSeats: UInt8 = 5
    var numberOfWheels: UInt8 = 4
    var type: CarType = .saloon
    var gearType: GearType = .automatic
    var motor: Motor = Motor(id: "111", name: "Default Motor",
                             model: "T9", numberOfCylinders: 4)
    var shouldHasAirbags: Bool = false
    var batteryName: String = "Default Battery Name"

    func buildCar() -> Car {
        return Car(color: color, numberOfSeats: numberOfSeats, numberOfWheels: numberOfWheels,
type: type, gearType: gearType, motor: motor, shouldHasAirbags: shouldHasAirbags, batteryName:
batteryName)
    }
}
```

□□□□□□□□□CarBuilder□□□□□

```
var builder = CarBuilder()

let defaultCar = builder.buildCar()
print(defaultCar)
/* prints
 color: UIExtendedSRGBColorSpace 1 0 0 1
 Number of seats: 5
 Number of Wheels: 4
 Type: automatic
 Motor: Motor(id: "111", name: "Default Motor", model: "T9", numberOfCylinders: 4)
 Airbag Availability: false
 Battery Name: Default Battery Name
*/

builder.batteryName = "New Battery Name"

let editedBatteryCar = builder.buildCar()
print(editedBatteryCar)
/*
 color: UIExtendedSRGBColorSpace 1 0 0 1
 Number of seats: 5
 Number of Wheels: 4
 Type: automatic
 Motor: Motor(id: "111", name: "Default Motor", model: "T9", numberOfCylinders: 4)
 Airbag Availability: false
 Battery Name: New Battery Name
```

```
*/
```

□□□□□□□□□ - □□□□ https://riptutorial.com/zh-TW/swift/topic/4941/□□□□---□□□

□□ 48: □□□□ – □□

□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□。□□□。

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□。

Examples

□□□

□□□□□□□□□□□□□□□□□□□□□□□**Adaptee** □□□□□□□□□□□□□**Target**□□□。□□□。

□Swift□□□□□□□□□□□□□□□□□□□□□□□□□。□□□□□□□□□□□□Target□□□□□□□□□□□□□□□□□□□□□□□□Adaptee□□□□。

```
// 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□□□□□□□□□□□□□□□□□□□□□□。□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□。

□□□□□□□UserDefaults□□□□□□□□□□□□□Facade□□□。

```
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!
    }
```

```
}
```

□□□□□□□□。

```
Defaults.set("Beyond all recognition.", forKey:"fooBar")
Defaults.object(forKey: "fooBar")
```

□□□□□□□□□UserDefaults□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□。

□□□□□□□□ - □□□ https://riptutorial.com/zh-TW/swift/topic/9497/□□□□---□□

□□ 49: □□□□□

□□

□□ □□□□□□□

□□ □□□□□□

□□□□□□□□□□Swift□□□□iBook。

Examples

□□□□

□□var□□□□□□□□□□□□□□□□□□□□

```
var num: Int = 10
```

□□□□□□□□□

```
num = 20 // num now equals 20
```

□□□□□let□□□

```
let num: Int = 10 // num cannot change
```

Swift□□□□□□□□□□□□□□□□□□□□□□

```
let ten = 10 // num is an Int
let pi = 3.14 // pi is a Double
let floatPi: Float = 3.14 // floatPi is a Float
```

□□□□□□□□□□□□ - □□□□□□□□□□□□unicode□□□□□□□□□□

> □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□Unicode□□□□□□□□□□□。 □□□□□□□□□□
>
> □□ developer.apple.com

```
var π: Double = 3.14159
var □□: String = "Apples"
```

□□□□

□□□□□□□□□□□□□ □□□□□□□□ □□□□“□□□□”□□□。 □□□□□□□/□□□□□□□□□□

```
class Dog {
    var name = ""
}
```

□□□□□□□□□Dog□□□□□□□□name□□□String。 □□□Dog□□□□□□□□□□□□□□

```
let myDog = Dog()
myDog.name = "Doggy" // myDog's name is now "Doggy"
```

□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□□。

□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□。□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□。□□lazy□□□□□□□□□□□□

```
lazy var veryExpensiveVariable = expensiveMethod()
```

□□□□□□□□□□□□□□□□□

```
lazy var veryExpensiveString = { () -> String in
    var str = expensiveStrFetch()
    str.expensiveManipulation(integer: arc4random_uniform(5))
    return str
}()
```

□□□□var□□□□□□□□□□□。

□□□□

□□□□□□□□□□ □□□□□□□□□getter□setter□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□。□□□□□□□□□□□□□□□

```
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 circumference: Double {
    get {
        return pi * radius * 2
    }
}
```

□□□□□□□□□□□□□□□□get □

```
var circumference: Double {
    return pi * radius * 2
}
```

□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□

```
func printSomething() {
    let localString = "I'm local!"
    print(localString)
}

func printSomethingAgain() {
    print(localString) // error
}
```

[illegible]。[illegible]

```
let globalString = "I'm global!"
print(globalString)

func useGlobalString() {
    print(globalString) // works!
}

for i in 0..<2 {
    print(globalString) // works!
}

class GlobalStringUser {
    var computeGlobalString {
        return globalString // works!
    }
}
```

[illegible]"[illegible]"[illegible]。

[illegible]

[illegible]。[illegible]。[illegible]static[illegible]type[illegible]

```
struct Dog {
    static var noise = "Bark!"
}

print(Dog.noise) // Prints "Bark!"
```

[illegible]class[illegible]static[illegible]。[illegible]

```
class Animal {
    class var noise: String {
        return "Animal noise!"
    }
}
class Pig: Animal {
    override class var noise: String {
        return "Oink oink!"
    }
}
```

[illegible] 。

[illegible]

[illegible]。

```
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。

  □□□□□□□□□□□□□□□didSet□willSet □

  - □□□□□
  - □□□□didSet□willSet□□□□

- □□□□□didSet□willSet oldValue□newValue□□□□□□□□□□□□

```
var myFontSize = 10 {
    willSet(newFontSize) {
        print("Will set font to \(newFontSize), it was \(myFontSize)")
    }
    didSet(oldFontSize) {
        print("Did set font to \(myFontSize), it was \(oldFontSize)")
    }
}
```

  □□□□□□□□□□setter□□□□□□□□□□□□□□□□□□□□

  - willSet(oldValue)□didSet(newValue)□□□□□□□□□□□□□□□□□□□□□□□□□□□□。

□□□□□□□□□□□ https://riptutorial.com/zh-TW/swift/topic/536/□□□□□

□□ 50: □□

□□

“□□□□□□□□□nil□□□□□□”

□□□□Apple Inc.“The Swift Programming Language□Swift 3.1 Edition□。”iBooks。
https://itun.es/us/k5SW7.l

□□□□□□□□□□□□□□let□□□□□□□□□□□□if-let□□□□□□guard-let□□□□□□□□□□□□□□□switch□□□□□□□□case-let□ □□□□coalesce□□□□??□□□□□nil

□□

- var optionalName□optionalType□ //□□□□□□□□□□□□□nil
- var optionalName□optionalType□ = value //□□□□□□□□□
- var optionalName□optionalType□ //□□□□□□□□□□□□
- □□□□ //□□□□□□□□□

□□

□□□□□□□□□□□□□Swift□□□□ 。

Examples

□□□□□□

Optionals□□□□□□□□□□□□□□□。 □□□□□□□□□□□□□□□□□□□□□□□□□nil □□□□□□。

□□□□□Swift□□□□□□□□□□□□□□□□□□□□□□□□□Apple□□□□□□□。 Apple□□□□□□□□□□□□□nil□□□□□□□□□□□□Objective-C□□□□□□□API□□□□□□□□。

□Swift□□□□□□nil□□□□□□□□□□□□□□。 □□□□□□a□□□□□□!□□□□?□□□□□。 □□□□□Int□□□□□□□□□

```
var numberOne: Int! = nil
var numberTwo: Int? = nil
```

?□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□。 □□□□□□□□□Int □□□□□□Int? □□□□□□□□□□□□□□□□□□nil

```
let str1 = "42"
let num1: Int? = Int(str1) // 42

let str2 = "Hello, World!"
let num2: Int? = Int(str2) // nil
```

!□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□。 □□□□□UIButton!□viewDidLoad()□□□□□□

```
//myButton will not be accessed until viewDidLoad is called,
//so a ! optional can be used here
var myButton: UIButton!

override func viewDidLoad(){
    self.myButton = UIButton(frame: self.view.frame)
    self.myButton.backgroundColor = UIColor.redColor()
    self.view.addSubview(self.myButton)
}
```

□□□□□

□□□□Optional□□□□□□□□□□。

□□□□□□□□□□□□□□□ Optional□□□□! □□□□ Optional !□□□。

□□□□□□□□□□□“□□□□□□□□□□”□□□□□□□“□□□□□□□□”。

□□□□□□□□□nil□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□*nil* □□□□□□□□□□□□□□!□□□□。

```
var text: String? = nil
var unwrapped: String = text! //crashes with "unexpectedly found nil while unwrapping an
Optional value"
```

□□□□□□□□□□□□if-let□□□□□□□□□□nil □□□□□□□□□□□□□

```
var number: Int?
if let unwrappedNumber = number {       // Has `number` been assigned a value?
    print("number: \(unwrappedNumber)") // Will not enter this line
} else {
    print("number was not assigned a value")
}
```

□□□ □□□□□□ □

```
var number: Int?
guard let unwrappedNumber = number else {
    return
}
print("number: \(unwrappedNumber)")
```

□□□□ unwrappedNumber□□□□□□□if-let□□□□□□□guard□□□。

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
var firstName:String?
var lastName:String?

if let fn = firstName, let ln = lastName {
    print("\(fn) + \(ln)")//pay attention that the condition will be true only if both
optionals are not nil.
}
```

□□。

□□□□□□□□□□□□□□□□□□□□□。 □□□□□□□□□if - else□□□

```
var firstName:String? = "Bob"
var myBool:Bool? = false

if let fn = firstName, fn == "Bob", let bool = myBool, !bool {
    print("firstName is bob and myBool was false!")
}
```

□□□□□□

□□□□□□□□□□□□*nil coalescing*□□□□□□□□□□□□□□□□□

```
func fallbackIfNil(str: String?) -> String {
    return str ?? "Fallback String"
}
print(fallbackIfNil("Hi")) // Prints "Hi"
print(fallbackIfNil(nil)) // Prints "Fallback String"
```

□□□□□□□□ □□□□□□□□□□□□□nil□□□□□□□□□□□

```
func someExpensiveComputation() -> String { ... }

var foo : String? = "a string"
let str = foo ?? someExpensiveComputation()
```

□□□□□□□□foo□□nil□□□□□□□someExpensiveComputation()。

□□□□□□□nil□□□□□□□□□□

```
var foo : String?
var bar : String?

let baz = foo ?? bar ?? "fallback string"
```

□□□□□□□ baz□□□□foo□□□□□□□□□□□□□□□□□□□□□□□□□□□□□bar□□□□□□□□□□□□□□□。

□□□□

□□□□□Optional Chaining□□□□□ □□□□□□□□□□□。 □□□□□□□□?□□□□□□□□□□□□□□□□□□□□□□□□□。

```
struct Foo {
    func doSomething() {
        print("Hello World!")
    }
}

var foo : Foo? = Foo()

foo?.doSomething() // prints "Hello World!" as foo is non-nil
```

□□foo□□□□□□□□□□□□doSomething()。 □□foo□nil □□□□□□□□□□ – □□□□□□□□□□□□□。

```
var foo : Foo? = nil

foo?.doSomething() // will not be called as foo is nil
```

□□□□Objective-C□□□□□□□nil□□□□□

“□□□□□”□□□□□□□□“□□□□”□□□□□□/□□□□□□□□。 □□。

```
struct Foo {
    var bar : Int
    func doSomething() { ... }
}

let foo : Foo? = Foo(bar: 5)
print(foo?.bar) // Optional(5)
```

□□foo?.bar□□□□□□Int?□□bar□□□□□□□□foo□□□□□□。

□□□□□□□□□□Void□□□□□Void?□□□□□□□□□。 □□□□□□□□□□□□□□□□□□□□□□□□。

```
let foo : Foo? = Foo()

if foo?.doSomething() != nil {
    print("foo is non-nil, and doSomething() was called")
} else {
    print("foo is nil, therefore doSomething() wasn't called")
}
```

□□□□□□□Void?□□□□nil□□□□□□□□□□□□□□foo□□□□nil□。

□□ – □□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□。 □□□□□□□□□C□□□□□□□□□null□□□□□□□□□□□□□□。 □□□□□□□□□□□□□□□□。 □□□□□□□□□□-1□□□□□□□□□□□。 □□□□□□□□□□□“□□”□。

□□□□□□□□□Swift□□□□□□□□□□□□。 □□□□□□□□ □□□□□□□□□□□□□ □

□□□

```
var possiblyInt: Int?
```

□□□□□□□□□□□□□□□□□□。

□□□nil□□□□□□□□□□□□□。

```
possiblyInt = 5      // PossiblyInt is now 5
possiblyInt = nil    // PossiblyInt is now unassigned
```

nil□□□□□□□□□□

```
if possiblyInt != nil {
    print("possiblyInt has the value \(possiblyInt!)")
}
```

□□□□!□print□□□□□□□□。

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□;□□□□□□□□□□□□□□□□□□□。

□□□□Int□□。

```
var someInt
someInt = parseInt("not an integer") // How would this function indicate failure?
```

□□□□Swift□□□□□□□□□□□□□□□□□ Int。 □□□□□□□□nil。

```
var someInt?
someInt = parseInt("not an integer")  // This function returns nil if parsing fails
if someInt == nil {
    print("That isn't a valid integer")
}
```

□□□□□□□ 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

```
enum NetworkError: ErrorType {
    case Offline
    case ServerError(String)
}
```

3.0

```
enum NetworkError: Error {
    // Swift 3 dictates that enum cases should be `lowerCamelCase`
    case offline
    case serverError(String)
}
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□do / catch □ throw□try。

```
func fetchResource(resource: NSURL) throws -> String {
    if let (statusCode, responseString) = /* ...from elsewhere...*/ {
        if case 500..<600 = statusCode {
            throw NetworkError.serverError(responseString)
        } else {
            return responseString
        }
    } else {
        throw NetworkError.offline
    }
}
```

□□□□do / catch□□□

```
do {
    let response = try fetchResource(resURL)
    // If fetchResource() didn't throw an error, execution continues here:
    print("Got response: \(response)")
    ...
} catch {
    // If an error is thrown, we can handle it here.
```

```
    print("Whoops, couldn't fetch resource: \(error)")
}
```

□□□□□□□□□□□□□□□□□□□□try□□□ try? □□□try! □

```
// error: call can throw but is not marked with 'try'
let response = fetchResource(resURL)

// "try" works within do/catch, or within another throwing function:
do {
    let response = try fetchResource(resURL)
} catch {
    // Handle the error
}

func foo() throws {
    // If an error is thrown, continue passing it up to the caller.
    let response = try fetchResource(resURL)
}

// "try?" wraps the function's return value in an Optional (nil if an error was thrown).
if let response = try? fetchResource(resURL) {
    // no error was thrown
}

// "try!" crashes the program at runtime if an error occurs.
let response = try! fetchResource(resURL)
```

□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□。

2.2

```
enum CustomError: ErrorType {
    case SomeError
    case AnotherError
}

func throwing() throws {
    throw CustomError.SomeError
}
```

3.0

```
enum CustomError: Error {
    case someError
    case anotherError
}

func throwing() throws {
    throw CustomError.someError
}
```

Do-Catch□□□□□□□□□□□□□□□□□□□□catch□□□□□□□□□□□□error □

```
do {
    try throwing()
```

```
} catch {
    print(error)
}
```

□□□□□□□□□□□□

```
do {
    try throwing()
} catch let oops {
    print(oops)
}
```

□□□□□□□□□catch□□。□□□□□Do□□□□□□□□□□□□□□□□□□□□。

□□Do-Catch□□□□□□□□□□□□□CustomError □□□□□□□□□□□□□□□□NSError。

2.2

```
do {
    try somethingMayThrow()
} catch let custom as CustomError {
    print(custom)
} catch let error as NSError {
    print(error)
}
```

3.0

□Swift 3□□□□□□□□□□□□NSError。

```
do {
    try somethingMayThrow()
} catch let custom as CustomError {
    print(custom)
} catch {
    print(error)
}
```

□□□□□□□□□□□□□□□□□□□

```
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
        }
    }
```

```
}
```

□□□□□□□

```
let airforceOne = Plane()
do {
    try airforceOne.fly()
} catch let emergency as Plane.Emergency {
    switch emergency {
    case .NoFuel:
        // call nearest airport for emergency landing
    case .EngineFailure(let reason):
        print(reason) // let the mechanic know the reason
    case .DamagedWing:
        // Assess the damage and determine if the president can make it
    }
}
```

□□□□□□

Swift□□□□□□□□□□□□□□□□□□□□□□□□□□。□□□□□□□□□□□□□□□□□□□□□□□□□try□□□□□。 try□□□□□□□。□□□□□□□□□□□□□□□□□□□□□□□。

□□□□□□□□□loadImage□atPath :)□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□。□□□。

```
let photo = try! loadImage(atPath: "./Resources/John Appleseed.jpg")
```

□□□□□□□□□□□□□□

□□□□□□□□enum

```
enum RegistrationError: Error {
    case invalidEmail
    case invalidPassword
    case invalidPhoneNumber
}
```

□□RegistrationError extension□□□□□□□□。

```
extension RegistrationError: LocalizedError {
    public var errorDescription: String? {
        switch self {
        case .invalidEmail:
            return NSLocalizedString("Description of invalid email address", comment: "Invalid
Email")
        case .invalidPassword:
            return NSLocalizedString("Description of invalid password", comment: "Invalid
Password")
        case .invalidPhoneNumber:
            return NSLocalizedString("Description of invalid phoneNumber", comment: "Invalid
Phone Number")
        }
    }
}
```

□□□□□

```
let error: Error = RegistrationError.invalidEmail
print(error.localizedDescription)
```

□□□□□□□□□□ 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 string = "Dog"
switch string {
case "Cat", "Dog":
  print("Animal is a house pet.")
default:
  print("Animal is not a house pet.")
}
```

□□□□□□□□□□

```
Animal is a house pet.
```

□□□□□□

switch□□□□□□□case□□□□□□□□。

```
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")
}
```

□□□□

switch□□□□□□case□□□□□□□□。

```
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**□□

where□□□□□switch 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 = 0□z = 0□x□□□□。□□□□□□□□x□□。
3. □□x = 0□z = 0□y□□□□。□□□□□□□□□□。
4. □□x = 0□y = 0□z□□□□。□□□□□□□□z□□。
5. □□□□□□。

□□□□□□□□□□□□□□□□。

□□□□□□□□□□□□□□switch□□□□□□□□□□□□

```
let coordinates: (x: Int, y: Int, z: Int) = (3, 2, 5)

switch (coordinates) {
case (0, 0, 0):
  print("Origin")
case (let x, 0, 0):
  print("On the x-axis at x = \(x)")
case (0, let y, 0):
  print("On the y-axis at y = \(y)")
case (0, 0, let z):
  print("On the z-axis at z = \(z)")
case (let x, let y, let z):
  print("Somewhere in space at x = \(x), y = \(y), z = \(z)")
}
```

□□□□□□□□let□□□□□□□□□□。□□□□□□□□□□□□□□□□□□□□。

□□□□switch□□□□□□□□□。□□□□□□□□□□□□□□□□ - □□□□□□□□□□□□□□□□□□□□□□。□□switch□□□□□□□□□□□□□□□□□□□□□□□。

□□□□□□□let-where□□□□□□□□□□□。□□□

```
let coordinates: (x: Int, y: Int, z: Int) = (3, 2, 5)

switch (coordinates) {
case (let x, let y, _) where y == x:
  print("Along the y = x line.")
case (let x, let y, _) where y == x * x:
  print("Along the y = x^2 line.")
default:
break
}
```

□□□□□□□“y□□x”□“y□□x□□”□□。

□□□□

□□□□□□□swift□□□□□□□□□□□□□□□□□□□□case□□□□□□□□□□□□□□□□。□□□□□□□□□□□□□□□□□□□□□□□□□□□□fallthrough□□。

```
switch(value) {
case 'one':
    // do operation one
    fallthrough
case 'two':
    // do this either independant, or in conjunction with first case
default:
    // default operation
}
```

□□□□□□□□□□□□□□。

□□□□□

Switch□□□Enum□□□□□

```
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□□□□□□□□□□□□□。□□□□□□□□ segues□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□segue□□□□□□□□□□□。

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 3 sytax

```
override func prepare(for 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.JSONObjectWithData□jsonData□options□NSJSONReadingOptions□//□jsonData□□□□Object。□□□□□□□。
- NSJSONSerialization.dataWithJSONObject□jsonObject□options□NSJSONWritingOptions□//□JSON□□□□NSData。□□NSJSONWritingOptions.PrettyPrinted□□□□□□□□□□□□□。

Examples

□□Apple Foundation□Swift□□□□□JSON□□□□□□□□□

JSONSerialization□□□□Apple□Foundation□□□。

2.2

□□JSON

JSONObjectWithData□□□□NSData □□□□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: .AllowFragments□□□options: []□□□□□□□□□□□□□□□□□□JSON。

□JSON

□□dataWithJSONObject□□JSON□□□□□□□□□□□□□□□□NSNull□□□□□□□□□□□□□□□UTF-8□□□NSData。

```
do {
    // Convert object to JSON as NSData
    let jsonData = try NSJSONSerialization.dataWithJSONObject(jsonObject, options: [])
    print("JSON data: \(jsonData)")

    // Convert NSData to String
    let jsonString = String(data: jsonData, encoding: NSUTF8StringEncoding)!
    print("JSON string: \(jsonString)")
} catch {
    print("error writing JSON: \(error)")
}
```

□□□□□options: .PrettyPrinted□□□options: []□□□□□□。

3.0

Swift 3□□□□□□□□□□□。

```
do {
    guard let jsonData = "[\"Hello\", \"JSON\"]".data(using: String.Encoding.utf8) else {
        fatalError("couldn't encode string as UTF-8")
    }

    // Convert JSON from NSData to AnyObject
    let jsonObject = try JSONSerialization.jsonObject(with: jsonData, options: [])

    // Try to convert AnyObject to array of strings
    if let stringArray = jsonObject as? [String] {
        print("Got array of strings: \(stringArray.joined(separator: ", "))")
    }
} catch {
    print("error reading JSON: \(error)")
}

do {
    // Convert object to JSON as NSData
    let jsonData = try JSONSerialization.data(withJSONObject: jsonObject, options: [])
    print("JSON data: \(jsonData)")

    // Convert NSData to String
    let jsonString = String(data: jsonData, encoding: .utf8)!
    print("JSON string: \(jsonString)")
} catch {
    print("error writing JSON: \(error)")
}
```

□□□□□□□□□□□□□**Swift 4.0**□□□□□。

□Swift 4.0□□□Swift□□□□□Encodable□Decodable□□□□□□□□□□□□□□□□□□□□□□。□□□□□□□□Encoder□Decoder□□□□□□□□□□□□□□□□□□□□□□□□□□□JSON□□□□□□□□□。□Codable□□□□□□□□□Encodable□Decodable□□。□□□□□□□□□□□JSON□□□□□。

□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□Codable□□。□□□□□□□□□□□□□□String □ Int□Double ;□□□□□□□Date □ Data□URL。□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□Codable。

□□□□□□□□□□Book□□□□Codable。

```
struct Book: Codable {
    let title: String
    let authors: [String]
    let publicationDate: Date
}
```

□□□□□□□□□□□□□Array□Dictionary□□Codable□□□□□□□□□□。

□□□□Codable □ Book□□□□□□□□Apple Foundation□JSONEncoder□JSONDecoder□□□JSON□□JSON□□□□□Book□□□□□□□□□JSON□□□。□□□□□□Encoder□Decoder□□□□□□□□□□□□□□□□□□。

□□□JSON□□

```
// Create an instance of Book called book
let encoder = JSONEncoder()
let data = try! encoder.encode(book) // Do not use try! in production code
print(data)
```

□□encoder.outputFormatting = .prettyPrinted□□□□□。 ##□JSON□□□□

□JSON□□□□

```
// Retrieve JSON string from some source
let jsonData = jsonString.data(encoding: .utf8)!
let decoder = JSONDecoder()
let book = try! decoder.decode(Book.self, for: jsonData) // Do not use try! in production code
print(book)
```

□□□□□□□□ Book.self□□□□□□□□□□JSON□□□。

□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□API□□JSON□□□□□□□□□□□□□□JSON□□□□□API□。

□□□□□□□JSON□□□□□□□□□□□□□Encodable。

```
struct Book: Encodable {
    let title: String
    let authors: [String]
    let publicationDate: Date
}
```

□□□□□□□JSON□□□□□□□□□□□□Decodable。

```
struct Book: Decodable {
    let title: String
    let authors: [String]
    let publicationDate: Date
}
```

□□□□□□□□

API□□□□□□Swift□□□□□□□□□□□□□□□□□□□□□□□□。□□□JSON□□□□□□□□□□□□□□□□□□□□□□□□□□□JSON□□□□□□□□□□□□□□□□□□□□□□□□。□□□□□□□□□□□□□□CodingKey□□□□□□□□□□□□□□□□□□。

```
struct Book: Codable {
    // ...
    enum CodingKeys: String, CodingKey {
        case title
        case authors
        case publicationDate = "publication_date"
    }
}
```

CodingKeys□□□□□□□□□□□□□□Codable□□□publicationDate□□□□□□□publication_date □□□□□□□API□□。

SwiftyJSON

SwiftyJSON□□□Swift□□□□□□□□□□□JSON□□□□□□□□□□□□□。

□□□□□□□□□ https □ //github.com/SwiftyJSON/SwiftyJSON

□□□□SwiftyJSON□□□□□□□□□□□□□□□□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
}
```

□SwiftyJSON□□□□□□□□

```
let json = JSON(data: data)
if let bookName = json[0]["book"]["name"].string {
    //We can now use the book name
}
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□nil。

□□□SwiftyJSON□□□Git□□□□□□□□□□ - Swift 3□□□□□。□□□“SwiftyJSON.swift”□□□□□□□□□□□□□□

```
import SwiftyJSON
```

□□□□□□□□□□□□□□□□□JSON□□□

```
let jsonObject = JSON(data: dataObject)
```

□□

```
let jsonObject = JSON(jsonObject) //This could be a string in a JSON format for example
```

□□□□□□□□□□□□□□

```
let firstObjectInAnArray = jsonObject[0]
let nameOfFirstObject = jsonObject[0]["name"]
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
let nameOfFirstObject = jsonObject[0]["name"].string //This will return the name as a string

let nameOfFirstObject = jsonObject[0]["name"].double //This will return null
```

□□□□□□□□□□swift□□□

```
let convolutedPath = jsonObject[0]["name"][2]["lastName"]["firstLetter"].string
```

□□□□□

```
let convolutedPath = jsonObject[0, "name", 2, "lastName", "firstLetter"].string
```

SwiftyJSON□□□□□□□□□□□□□

```
if let name = json[1337].string {
    //You can use the value - it is valid
} else {
    print(json[1337].error) // "Array[1337] is out of bounds" - You cant use the value
}
```

□□□□□□JSON□□□□□□□□□□□□

```
var originalJSON:JSON = ["name": "Jack", "age": 18]
originalJSON["age"] = 25 //This changes the age to 25
originalJSON["surname"] = "Smith" //This creates a new field called "surname" and adds the
```

```
value to it
```

□□□□□JSON□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
if let string = json.rawString() { //This is a String object
    //Write the string to a file if you like
}

if let data = json.rawData() { //This is an NSData object
    //Send the data to your server if you like
}
```

□□□

Freddy□□Big Nerd Ranch□□□JSON□□□。□□□□□□□□□

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
    ]
  }
}
```

```
let jsonString = "{\"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]}}"
let jsonData = jsonString.dataUsingEncoding(NSUTF8StringEncoding)!
```

□□□□□□□□

□□□□□□□□□□□□□□□JSON□□□□□□□□□□□□□。

```
do {
    let json = try JSON(data: jsonData)
    let success = try json.bool("success")
} catch {
    // do something with the error
}
```

□□□□□try □□□□□json□□□□□"success"□□□□□ - □□□□□□□□□□□□□□□□□。

□□□□□□□□□□□□JSON□□□□□□□□□。 □□□□□□□□□□□□□□□□□□□□□□□□□□□□□。

```
do {
    let json = try JSON(data: jsonData)
    let georgiaZipCodes = try json.array("states", "Georgia")
    let firstPersonName = try json.string("people", 0, "name")
} catch {
    // do something with the error
}
```

□□□□□□□□

□□□JSON□□□□□□□JSONDecodable□□□□□□。

```
public struct Person {
    public let name: String
    public let age: Int
    public let spouse: Bool
}

extension Person: JSONDecodable {
    public init(json: JSON) throws {
        name = try json.string("name")
        age = try json.int("age")
        spouse = try json.bool("spouse")
    }
}

do {
    let json = try JSON(data: jsonData)
    let people = try json.arrayOf("people", type: Person.self)
} catch {
    // do something with the error
}
```

□□□□□□□

□□JSON□□□□□□□□□□□NSData。

```
let success = JSON.Bool(false)
let data: NSData = try success.serialize()
```

□□□□□□□□

□□□□JSONEncodable□□□□□□□□□□□□□□□□□□□NSData。

```
extension Person: JSONEncodable {
    public func toJSON() -> JSON {
        return .Dictionary([
            "name": .String(name),
            "age": .Int(age),
            "spouse": .Bool(spouse)
        ])
    }
}

let newPerson = Person(name: "Glenn", age: 23, spouse: true)
let data: NSData = try newPerson.toJSON().serialize()
```

□□

Arrow□Swift□□□□JSON□□□。

□□□□□JSON□□□<--□□□□□□□□□□□□□□□□□

```
identifier <-- json["id"]
name <-- json["name"]
stats <-- json["stats"]
```

□□

Swift□□

```
struct Profile {
    var identifier = 0
    var name = ""
    var link: NSURL?
    var weekday: WeekDay = .Monday
    var stats = Stats()
    var phoneNumbers = [PhoneNumber]()
}
```

JSON□□

```
{
    "id": 15678,
    "name": "John Doe",
    "link": "https://apple.com/steve",
    "weekdayInt" : 3,
    "stats": {
        "numberOfFriends": 163,
        "numberOfFans": 10987
    },
    "phoneNumbers": [{
                     "label": "house",
                     "number": "9809876545"
                     }, {
                     "label": "cell",
                     "number": "0908070656"
                     }, {
                     "label": "work",
```

```
                    "number": "0916570656"
    }]
}
```

□□

```
extension Profile: ArrowParsable {
    mutating func deserialize(json: JSON) {
        identifier <-- json["id"]
        link <-- json["link"]
        name <-- json["name"]
        weekday <-- json["weekdayInt"]
        stats <- json["stats"]
        phoneNumbers <-- json["phoneNumbers"]
    }
}
```

□□

```
let profile = Profile()
profile.deserialize(json)
```

□□□

□□□

```
github "s4cha/Arrow"
```

□*CocoaPods*

```
pod 'Arrow'
use_frameworks!
```

□□

□□□Xcode□□□□□□□□□Arrow.swift□□

https://github.com/s4cha/Arrow

□□□□□□□

□GitHub□□□□□Arrow□□□□□□□□□□□□Framework□□。□□□□□□□□□□。

□□□**JSON**□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□JSON□□□□□□□□□□□□□□□□□□□□Todo as

```
struct Todo {
    let comment: String
}
```

□□□□□□JSON□□□□□□□□NSJSONSerialization□□□□□□□NSData □□□□□□□□□□。

□□□□□□□□JSONDecodable□□

```
typealias JSONDictionary = [String:AnyObject]
protocol JSONDecodable {
    associatedtype Element
    static func from(json json: JSONDictionary) -> Element?
```

```
}
```

Todo JSONDecodable

```
extension Todo: JSONDecodable {
    static func from(json json: JSONDictionary) -> Todo? {
        guard let comment = json["comment"] as? String else { return nil }
        return Todo(comment: comment)
    }
}
```

json

```
{
    "todos": [
        {
            "comment" : "The todo comment"
        }
    ]
}
```

API AnyObject。 JSONDictionary

```
guard let jsonDictionary = dictionary as? JSONDictionary else { return }
```

JSON Todo todos

```
guard let todosDictionary = jsonDictionary["todos"] as? [JSONDictionary] else { return }
```

flatMap Todo nil

```
let todos: [Todo] = todosDictionary.flatMap { Todo.from(json: $0) }
```

JSON Swift 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□□

```
   func jsonParsingMethod() {
        //get the file
        let filePath = Bundle.main.path(forResource: "animals", ofType: "json")
        let content = try! String(contentsOfFile: filePath!)

        let data: Data = content.data(using: String.Encoding.utf8)!
        let json: NSDictionary =  try! JSONSerialization.jsonObject(with: data as Data,
options:.mutableContainers) as! NSDictionary

//Call which part of the file you'd like to pare
        if let results = json["mammals"] as? [[String: AnyObject]] {

            for res in results {
                //this will print out the names of the mammals from out file.
                if let rates = res["name"] as? String {
                    print(rates)
                }
           }
       }
    }
```

□□□□□□□□□□□□□□□□□□□□□□□□□□NSObject□□□□□□□。

□□□□□□ParsingObject□□swift□□□□□□□□□□□。

□□□□□□□□JSON□□□□

。□□□□□□□□□□□□□□□□□□□□swift□□□□name□question □□□□□□

```
var name: String?
var question: String?
```

□□□□□□□□□NSObject□□□□ViewController.swift var array = ParsingObject□□□□□□□□□□□□□□□□□□□□□□□□□□□。

```
     func jsonParsingMethod() {
            //get the file
            let filePath = Bundle.main.path(forResource: "animals", ofType: "json")
            let content = try! String(contentsOfFile: filePath!)

            let data: Data = content.data(using: String.Encoding.utf8)!
            let json: NSDictionary =  try! JSONSerialization.jsonObject(with: data as Data,
 options:.mutableContainers) as! NSDictionary

 //This time let's get Sea Animals
            let results = json["Sea Animals"] as? [[String: AnyObject]]

 //Get all the stuff using a for-loop
            for i in 0 ..< results!.count {

 //get the value
                let dict = results?[i]
                let resultsArray = ParsingObject()

 //append the value to our NSObject file
                resultsArray.setValuesForKeys(dict!)
                array.append(resultsArray)

            }

        }
```

□□□□□□□□□□□□□□□tableview□□□□□

```
    func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
            return array.count
        }

        func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) ->
 UITableViewCell {
            let cell = tableView.dequeueReusableCell(withIdentifier: "Cell", for: indexPath)
 //This is where our values are stored
            let object = array[indexPath.row]
            cell.textLabel?.text = object.name
            cell.detailTextLabel?.text = object.question
            return cell
        }
```

□□□□□□□□□□JSON□ https://riptutorial.com/zh-TW/swift/topic/223/□□□□□json

第 54 章：閉包

語法

- var closureVar：<parameters> -> <returnType> //閉包變量聲明
- typealias ClosureType = <parameters> -> <returnType>
- <statement>：{[<captureList>] <parameters><throws-ness> -> <returnType>} //閉包表達式

備註

Swift閉包與其他編程語言中的Apple塊類似。

Examples

基本閉包

閉包是自包含的**lambdas**，可以作為參數傳遞給函數和方法。

```
let sayHi = { print("Hello") }
// The type of sayHi is "() -> ()", aka "() -> Void"

sayHi()  // prints "Hello"
```

閉包可以使用其範圍內可用的常量和變量。

```
let addInts = { (x: Int, y: Int) -> Int in
    return x + y
}
// The type of addInts is "(Int, Int) -> Int"

let result = addInts(1, 2)  // result is 3

let divideInts = { (x: Int, y: Int) throws -> Int in
    if y == 0 {
        throw MyErrors.DivisionByZero
    }
    return x / y
}
// The type of divideInts is "(Int, Int) throws -> Int"
```

閉包通過引用捕獲值

```
// This function returns another function which returns an integer
func makeProducer(x: Int) -> (() -> Int) {
    let closure = { x }  // x is captured by the closure
    return closure
}

// These two function calls use the exact same code,
// but each closure has captured different values.
let three = makeProducer(3)
let four = makeProducer(4)
three()  // returns 3
four()  // returns 4
```

閉包通過引用捕獲值

```
let squares = (1...10).map({ $0 * $0 })  // returns [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
let squares = (1...10).map { $0 * $0 }

NSURLSession.sharedSession().dataTaskWithURL(myURL,
    completionHandler: { (data: NSData?, response: NSURLResponse?, error: NSError?) in
        if let data = data {
            print("Request succeeded, data: \(data)")
        } else {
            print("Request failed: \(error)")
        }
    }).resume()
```

□□□□

□□□□□□□□

{ [*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。

```
// The closure's type is unknown, so we have to specify the type of x and y.
// The output type is inferred to be Int, because the + operator for Ints returns Int.
let addInts = { (x: Int, y: Int) in x + y }

// The closure's type is specified, so we can omit the parameters' type annotations.
let addInts: (Int, Int) -> Int = { x, y in x + y }
let addInts: (Int, Int) -> Int = { $0 + $1 }
```

□□□□□□

□□□□□□□□□□□□□□□□

```
func foo(value: Double, block: () -> Void) { ... }
func foo(value: Double, block: Int -> Int) { ... }
func foo(value: Double, block: (Int, Int) -> String) { ... }
```

□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□{ / } □

```
foo(3.5, block: { print("Hello") })

foo(3.5) { print("Hello") }

dispatch_async(dispatch_get_main_queue(), {
    print("Hello from the main queue")
})

dispatch_async(dispatch_get_main_queue()) {
    print("Hello from the main queue")
}
```

□□() □

```
func bar(block: () -> Void) { ... }
```

```
bar() { print("Hello") }

bar { print("Hello") }
```

@noescape□□

□□□@noescape□□□□□□□□□□□□□□□□□□□□□□□□□□self.□□□□□□□□□

```
func executeNow(@noescape block: () -> Void) {
    // Since `block` is @noescape, it's illegal to store it to an external variable.
    // We can only call it right here.
    block()
}

func executeLater(block: () -> Void) {
    dispatch_async(dispatch_get_main_queue()) {
        // Some time in the future...
        block()
    }
}
```

```
class MyClass {
    var x = 0
    func showExamples() {
        // error: reference to property 'x' in closure requires explicit 'self.' to make
capture semantics explicit
        executeLater { x = 1 }

        executeLater { self.x = 2 }  // ok, the closure explicitly captures self

        // Here "self." is not required, because executeNow() takes a @noescape block.
        executeNow { x = 3 }

        // Again, self. is not required, because map() uses @noescape.
        [1, 2, 3].map { $0 + x }
    }
}
```

□□□□3□□

□□□□□Swift 3□□□□□□□□□□@noescape。□□□□□□□□□□□□□。□Swift 3□□□□□□□□□□□□□□□□□□□“@escaping”□□□□□□□□□□□□□□□。

---

throws□ rethrows

□□□□□□□□□□□□□□□□□□□□ □

```
func executeNowOrIgnoreError(block: () throws -> Void) {
    do {
        try block()
    } catch {
        print("error: \(error)")
    }
}
```

□□□□□□□□□□□□□□□□□□□□□□□□

```
func executeNowOrThrow(block: () throws -> Void) throws {
    try block()
}
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□throw□□□

```
// It's annoying that this requires "try", because "print()" can't throw!
try executeNowOrThrow { print("Just printing, no errors here!") }
```

□□□□□□**rethrows** □□□□□□□□□□□□□□□□□□□□□□□□□ □

```
func executeNowOrRethrow(block: () throws -> Void) rethrows {
    try block()
}

// "try" is not required here, because the block can't throw an error.
executeNowOrRethrow { print("No errors are thrown from this closure") }

// This block can throw an error, so "try" is required.
try executeNowOrRethrow { throw MyError.Example }
```

□□□□□□□□□□□□rethrows □□□map() □ filter()□indexOf()。

□□□□□/□□□□□□□□□□

```
class MyClass {
    func sayHi() { print("Hello") }
    deinit { print("Goodbye") }
}
```

□□□

```
let closure: () -> Void
do {
    let obj = MyClass()
    // Captures a strong reference to `obj`: the object will be kept alive
    // as long as the closure itself is alive.
    closure = { obj.sayHi() }
    closure()  // The object is still alive; prints "Hello"
} // obj goes out of scope
closure()  // The object is still alive; prints "Hello"
```

□□□□□□□□□□□□□□□□□□□□□□□

```
let closure: () -> Void
do {
    let obj = MyClass()
    // Captures a weak reference to `obj`: the closure will not keep the object alive;
    // the object becomes optional inside the closure.
    closure = { [weak obj] in obj?.sayHi() }
    closure()  // The object is still alive; prints "Hello"
} // obj goes out of scope and is deallocated; prints "Goodbye"
closure()  // `obj` is nil from inside the closure; this does not print anything.
```

```
let closure: () -> Void
do {
    let obj = MyClass()
    // Captures an unowned reference to `obj`: the closure will not keep the object alive;
    // the object is always assumed to be accessible while the closure is alive.
    closure = { [unowned obj] in obj.sayHi() }
    closure()  // The object is still alive; prints "Hello"
} // obj goes out of scope and is deallocated; prints "Goodbye"
closure()  // crash! obj is being accessed after it's deallocated.
```

□□□□□□□□□□“ □□□□”□□□“Swift□□□□”□“ □□□□□□”□□。

□□□□

□□ 。 □□□。

```
class Game {
    var score = 0
    let controller: GCController
    init(controller: GCController) {
        self.controller = controller

        // BAD: the block captures self strongly, but self holds the controller
        // (and thus the block) strongly, which is a cycle.
        self.controller.controllerPausedHandler = {
            let curScore = self.score
            print("Pause button pressed; current score: \(curScore)")
        }

        // SOLUTION: use `weak self` to break the cycle.
        self.controller.controllerPausedHandler = { [weak self] in
            guard let strongSelf = self else { return }
            let curScore = strongSelf.score
            print("Pause button pressed; current score: \(curScore)")
        }
    }
}
```

□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□。

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

```
print("1. Going to call getData")

getData("http://www.example.com") {(result: NSData?) -> Void in

    // Called when the data from http://www.example.com has been fetched.
    print("2. Fetched data")
}

print("3. Called getData")
```

□□□□□□□□□□□□□□□□□□□□□□

```
"1. Going to call getData"
"3. Called getData"
"2. Fetched data"
```

□□□□□URL□□□□□□□□□□□□□□□□□□□print("2. Fetched data")。

□□□□□□□□

□□□□typealias□□□□。 □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□。 □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□“□□”。

```
public typealias ClosureType = (x: Int, y: Int) -> Int
```

□□□□□□□□typealias□□□□□

```
public func closureFunction(closure: ClosureType) {
    let z = closure(1, 2)
}

closureFunction() { (x: Int, y: Int) -> Int in return x + y }
```

□□□□□□□ https://riptutorial.com/zh-TW/swift/topic/262/□□

□□ 55: □

Examples

□□□

□□□□□□□□□□。□□□□□□□□□□□。

```
var colors = Set<String>()
```

□□□□□□□□□□□□□□□□□□□□。

```
var favoriteColors: Set<String> = ["Red", "Blue", "Green", "Blue"]
// {"Blue", "Green", "Red"}
```

□□□□□□□

```
var favoriteColors: Set = ["Red", "Blue", "Green"]
//favoriteColors = {"Blue", "Green", "Red"}
```

□□□□□insert(_:)□□□□□□□□□□□。

```
favoriteColors.insert("Orange")
//favoriteColors = {"Red", "Green", "Orange", "Blue"}
```

□□□□□remove(_:)□□□□□□□□□□。□□□□□□□□□□□□□□□□□□□□□□□□□□nil。

```
let removedColor = favoriteColors.remove("Red")
//favoriteColors = {"Green", "Orange", "Blue"}
// removedColor = Optional("Red")

let anotherRemovedColor = favoriteColors.remove("Black")
// anotherRemovedColor = nil
```

□□□□□□□□□

```
var favoriteColors: Set = ["Red", "Blue", "Green"]
//favoriteColors = {"Blue", "Green", "Red"}
```

□□□□□contains(_:)□□□□□□□□□□□。□□□□□□□□□□□□□true。

```
if favoriteColors.contains("Blue") {
    print("Who doesn't like blue!")
}
// Prints "Who doesn't like blue!"
```

□□□□□□□

□□□□□□□

□□□□□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 union = favoriteColors.union(newColors) // a OR b
// union = {"Red", "Purple", "Green", "Orange", "Blue"}
```

□□□ "□□ "□□□□□□□□□。

□□□□□□□□□□□

□□□□□exclusiveOr(_:)□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□。

```
let exclusiveOr = favoriteColors.exclusiveOr(newColors) // a XOR b
// exclusiveOr = {"Red", "Purple", "Orange", "Blue"}
```

□□□ "*Green*"□□□□□□□□□□□□□□□□□□□□□□□□。

□□□□□□□

□□□□□subtract(_:)□□□□□□□□□□□□□□□□□□。

```
let subtract = favoriteColors.subtract(newColors) // a - (a AND b)
// subtract = {"Blue", "Red"}
```

□□□ "□□ "□□□□□□□□□□□□□□□□□□□□□□□□□□。

□□□□□□□□□□□□□**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)

```
let ships : Set<Starship> = [Starship(name:"Enterprise D"), Starship(name:"Voyager"),
Starship(name:"Defiant") ]
```

CountedSet

3.0

Swift 3□□□CountedSet□□□□NSCountedSet Objective-C□□Swift□□□。

□□□□□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: □□□□□□□□

□□

□□□□□□□□□□□□□□Swift□□WWDC 2015□□□□□□□□ 。

□□□□□□□□□□□□ □Swift 2□□□□□□□□□□ 。

Examples

□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□。

□□□□□□□□□□□□□ViewModel□□UIViewController。

□□□□□□□□□□□

1. □□□□□□ViewModel□□□□□□□□□□□□UIViewController□□□□□□□□□□。
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. □□□

```
class FakeViewModel : ViewModelType {
   let title : String = "FakeTitle"

   var didConfirm = false
   func confirm() {
       didConfirm = true
   }
}

class ViewControllerTest : XCTestCase {
    var sut : ViewController!
    var viewModel : FakeViewModel!

    override func setUp() {
        super.setUp()

        viewModel = FakeViewModel()
        sut = // ... initialization for view controller
        sut.viewModel = viewModel

        XCTAssertNotNil(self.sut.view) // Needed to trigger view loading
    }

    func testTitleLabel() {
        XCTAssertEqual(self.sut.titleLabel.text, "FakeTitle")
    }

    func testTapOnButton() {
        sut.didTapOnButton(UIButton())
        XCTAssertTrue(self.viewModel.didConfirm)
    }
}
```

□□□□□□□□□□

□□□□□□□□□□□□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□□□□□□□□□□□□□□□□□□□□□□□□□。

```
extension GetAPIItemDataOperation {

    static func get(url: NSURL, completed: ([ItemData]) -> Void) {

        let date = NSDate(
        timeIntervalSinceNow: NSDate().timeIntervalSince1970
            + 5000)

        // get data from url
        let urlData: [String: AnyObject] = [
            "title": "Red Camaro",
            "desc": "A fast red car.",
            "thumb":"http://cars.images.com/red-camaro.png",
            "created": NSDate(), "updated": date]

        // in this example forced unwrapping is used
        // forced unwrapping should never be used in practice
        // instead conditional unwrapping should be used (guard or if/let)
        let item = Item(
            title: urlData["title"] as! String,
            description: urlData["desc"] as! String,
            thumbnailURL: NSURL(string: urlData["thumb"] as! String)!,
            created: urlData["created"] as! NSDate,
            updated: urlData["updated"] as! NSDate)

        completed([item])

    }
}

struct ItemOperation: GetAPIItemDataOperation { }
```

□□ItemData□□□□□□□□□□□□□□□□□□□□。

```
struct Item: ItemData {

    let title: String
    let description: String
    let thumbnailURL: NSURL
    let created: NSDate
    let updated: NSDate

}
```

□□□□□item□□□□□□□□。

```
extension Item: DisplayItem {

    func hasBeenUpdated() -> Bool {
        return updated.timeIntervalSince1970 >
```

```
            created.timeIntervalSince1970
    }

    func getFormattedTitle() -> String {
        return title.stringByTrimmingCharactersInSet(
            .whitespaceAndNewlineCharacterSet())
    }

    func getFormattedDescription() -> String {
        return description.stringByTrimmingCharactersInSet(
            .whitespaceAndNewlineCharacterSet())
    }
}
```

□□□□get□□□□□□□□□。

```
ItemOperation.get(NSURL()) { (itemData) in

    // perhaps inform a view of new data
    // or parse the data for user requested info, etc.
    dispatch_async(dispatch_get_main_queue(), {

        // self.items = itemData
    })

}
```

□□□□□□□□□□□□。 □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□。 □□□□□□API□□□□□□□□□□□Core Data□□。

```
// the default core data created classes + extension
class LocalItem: NSManagedObject { }

extension LocalItem {

    @NSManaged var title: String
    @NSManaged var itemDescription: String
    @NSManaged var thumbnailURLStr: String
    @NSManaged var createdAt: NSDate
    @NSManaged var updatedAt: NSDate
}
```

□□Core Data□□□□□□□□DisplayItem□□。

```
extension LocalItem: DisplayItem {

    func hasBeenUpdated() -> Bool {
        return updatedAt.timeIntervalSince1970 >
            createdAt.timeIntervalSince1970
    }

    func getFormattedTitle() -> String {
        return title.stringByTrimmingCharactersInSet(
            .whitespaceAndNewlineCharacterSet())
    }

    func getFormattedDescription() -> String {
        return itemDescription.stringByTrimmingCharactersInSet(
            .whitespaceAndNewlineCharacterSet())
    }
}
```

```
// In use, the core data results can be
// conditionally casts as a protocol
class MyController: UIViewController {

    override func viewDidLoad() {

        let fr: NSFetchRequest = NSFetchRequest(
        entityName: "Items")

        let context = NSManagedObjectContext(
        concurrencyType: .MainQueueConcurrencyType)

        do {

            let items: AnyObject = try context.executeFetchRequest(fr)
            if let displayItems = items as? [DisplayItem] {

                print(displayItems)
            }

        } catch let error as NSError {
            print(error.localizedDescription)
        }

    }
}
```

□□□□□□□□□□□□□□□ https://riptutorial.com/zh-TW/swift/topic/2502/□□□□□□□□□□

□□ 57: □

□□

init()□□□□□□□□□□□□□□□□□□□□□□□。 □□□□□□□□□□□□ □□□□

Examples

□□□□□

□□□□□□□□□□

```
class Dog {}
```

□□□□□□□□□□□□□

```
class Animal {}
class Dog: Animal {}
```

□□□□□□□ Animal□□□□Dog□□□□□ 。

□□□□

□□□□□□ □□□□□□□□□□□□□□□□□□□。

```
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: String // name is a variable property.
    let age: Int // age is a constant property.
    init(name: String, age: Int) {
        self.name = name
        self.age = age
    }
}

let constantDog = Dog(name: "Rover", age: 5)// This instance is a constant.
var variableDog = Dog(name: "Spot", age 7)// This instance is a variable.

constantDog.name = "Fido" // Not an error because name is a variable property.
constantDog.age = 6 // Error because age is a constant property.
constantDog = Dog(name: "Fido", age: 6)
/* The last one is an error because you are changing the actual reference, not
just what the reference points to. */
```

```
variableDog.name = "Ace" // Not an error because name is a variable property.
variableDog.age = 8 // Error because age is a constant property.
variableDog = Dog(name: "Ace", age: 8)
/* The last one is not an error because variableDog is a variable instance and
therefore the actual reference can be changed. */
```

□□===□□□□□□□□□□□ □□□□□□□□□□□□□□□

```
class Dog: Equatable {
    let name: String
    init(name: String) { self.name = name }
}

// Consider two dogs equal if their names are equal.
func ==(lhs: Dog, rhs: Dog) -> Bool {
    return lhs.name == rhs.name
}

// Create two Dog instances which have the same name.
let spot1 = Dog(name: "Spot")
let spot2 = Dog(name: "Spot")

spot1 == spot2   // true, because the dogs are equal
spot1 != spot2   // false

spot1 === spot2  // false, because the dogs are different instances
spot1 !== spot2  // true
```

□□□□□

□□□□□□□□□□□□□□□□□□□。□□□□□□□□ Dog□□□□□□□ name□dogYearAge □

```
class Dog {
    var name = ""
    var dogYearAge = 0
}
```

□□□□□□□□□□□□□□□

```
let dog = Dog()
print(dog.name)
print(dog.dogYearAge)
```

□□□□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□ □□□□□□□

```
class Dog {
    func bark() {
        print("Ruff!")
    }
}
```

□□□□□□□□□□□□□

```
dog.bark()
```

□□□□□□□

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 = "Paul"

let name = value as! String
print(name) // prints "Paul"

let age = value as! Double // crash: "Could not cast value…"
```

□□□□□□□□□□□□□□□□□□□□□□□□

```
let value: Any = "George"

if let name = value as? String {
    print(name) // prints "George"
}

if let age = value as? Double {
    print(age) // Not executed
}
```

□□□□□

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**□□□□□□

[illegible]

[illegible]。

Swift[illegible]is[illegible]as[illegible]。[illegible]。

[illegible]

[illegible]。[illegible]as[illegible]。

[illegible]。[illegible]。[illegible] – [illegible]。

[illegible]。[illegible]nil。[illegible]。

[illegible]。[illegible]。[illegible]。

[illegible]Int[illegible]Float[illegible] –

```
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
```

[illegible]

```
    let numbers = 888.00
    let floatValue = String(numbers)
    print(floatValue) // Output : 888.0

    // Get Float value at particular decimal point
    let numbers = 888.00
    let floatValue = String(format: "%.2f", numbers) // Here %.2f will give 2 numbers after
decimal points we can use as per our need
    print(floatValue) // Output : "888.00"
```

[illegible]

```
let numbers = 888
let intValue = String(numbers)
print(intValue) // Output : "888"
```

[illegible]

```
let numbers = 888.00
let floatValue = String(numbers)
print(floatValue)
```

String□□□□□□

```
    let numbers: Any = 888.00
    let floatValue = String(describing: numbers)
    print(floatValue) // Output : 888.0
```

□□String□Int□

```
    let hitCount = "100"
    let data :AnyObject = hitCount
    let score = Int(data as? String ?? "") ?? 0
    print(score)
```

□JSON□□□□□

```
    let json = ["name" : "john", "subjects": ["Maths", "Science", "English", "C Language"]] as
[String : Any]
    let name = json["name"] as? String ?? ""
    print(name) // Output : john
    let subjects = json["subjects"] as? [String] ?? []
    print(subjects) // Output : ["Maths", "Science", "English", "C Language"]
```

□Optional JSON□□□

```
    let response: Any = ["name" : "john", "subjects": ["Maths", "Science", "English", "C
Language"]]
    let json = response as? [String: Any] ?? [:]
    let name = json["name"] as? String ?? ""
    print(name) // Output : john
    let subjects = json["subjects"] as? [String] ?? []
    print(subjects) // Output : ["Maths", "Science", "English", "C Language"]
```

□□□□□□JSON□□

```
    let response: Any = ["name" : "john", "subjects": ["Maths", "Science", "English", "C
Language"]] //Optional Response

    guard let json = response as? [String: Any] else {
        // Handle here nil value
        print("Empty Dictionary")
        // Do something here
        return
    }
    let name = json["name"] as? String ?? ""
    print(name) // Output : john
    let subjects = json["subjects"] as? [String] ?? []
    print(subjects) // Output : ["Maths", "Science", "English", "C Language"]
```

□□□□□□□□

```
    let response: Any? = nil
    guard let json = response as? [String: Any] else {
```

```
        // Handle here nil value
        print("Empty Dictionary")
        // Do something here
        return
    }
    let name = json["name"] as? String ?? ""
    print(name)
    let subjects = json["subjects"] as? [String] ?? []
    print(subjects)
```

□□□ **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
    }
}
```

□□□□□□□□□□□□□□□

```
list.remove(at: 42)
```

□□□□□□□□。□□□□□□□□list.remove(42) □□□□□□□□□□□□□□42□Element□□□□□□□□□42□□Element。

□□□□

□□□□□□□□□□□□。

□□□□□□□□□

```
extension List {
    public mutating func removeElement(element: Element) -> Element? {
        // implementation
    }
}
```

□□□□□□□□□□□□list.removeElement(someObject)。□□someObject□□□□□□□□□□。□□□□□□□□□□□

```
extension List {
    public mutating func remove(_ member: Element) -> Element? {
        // implementation
    }
}
```

□□□□□□□□□□□□ list.remove(someObject)。

□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□。

□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□Iterator□□□□□□□□□□“Type”□□。 □□□IteratorType□

□□□□□□□□□□□□□□□□

□□□。
□□

```
func addObserver(_ observer: NSObject, forKeyPath path: String)
```

□□□□□□`object.addObserver□self□forKeyPath□path□

□□

```
func add(_ observer: NSObject, for keyPath: String)
```

□□□□□□object.add(self, for: path)

□□□□□

□□□□□□

□□□□□□□□□□□□。

□□

```
list.insert(element, at: index)
```

□□

```
list.insert(element, position: index)
```

□□□□□□

□□□□□□□□`make`□□。

□□

```
factory.makeObject()
```

□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□。

□□

```
factory.makeObject(key: value)
```

□□□

```
factory.makeObject(havingProperty: value)
```

□□□□□□□□

- □□□□□□□□□□□□□□□□□□□□□□□□□□form-□□□□□□□□□□□□。
- □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□-ing□-ed□□□□□□□□□□。

□□□□□□□□□

```
print(value)
array.sort()                  // in place sorting
list.add(value)               // mutates list
set.formUnion(anotherSet)     // set is now the union of set and anotherSet
```

□□□□□□□

```
let sortedArray = array.sorted()      // out of place sorting
let union = set.union(anotherSet)     // union is now the union of set and another set
```

□□□□□□□□

□□□□□□□□□□□□□□□□□□。

□□

```
set.isEmpty
line.intersects(anotherLine)
```

□□□□□

- □□□□□□□□□□□□□□□□□□□□□□□□□□□。
- □□□□□□□□□□□□□□□-able □ -ible□-ing□□□□。

□□

```
Collection        // describes that something is a collection
ProgressReporting // describes that something has the capability of reporting progress
Equatable         // describes that something has the capability of being equal to something
```

□□□□□□

□□□□□□□□□□□□□□□□□□□。

□□

```
let factory = ...
let list = [1, 2, 3, 4]
```

□□

□□□□□

□□□□□□□□□□□□□□□□□□□□□。

□□

```
protocol Collection {}
struct String {}
class UIView {}
struct Int {}
enum Color {}
```

□□□□......

□□□□□□□□□□□□□□□□□□□□□。

□□

```
let greeting = "Hello"
let height = 42.0

enum Color {
    case red
    case green
    case blue
}

func print(_ string: String) {
    ...
}
```

□□□□□□

□□□□□□□□□□□□□□□□。 □□□□/□□□□□□□□□□□□□□□□□□□□□□□□□□。

□□□□□

```
protocol IteratorType { ... }
```

□□□□□□□□

```
let inputView = ...
```

□□□

□□□□□□□URL□ ID□□□□□□□□□□□。 □□□□□□□□□□□□□□□□□□□□□。

□□

```
let userID: UserID = ...
let urlString: URLString = ...
```

□□□□□□□□□ https://riptutorial.com/zh-TW/swift/topic/3031/□□□□

□□ 60: □□□□□

Examples

□□□□□□

Swift□□□□□□□□□□。□□operator□□□□□□□□□□□□□operator。

□□□□□□□□□□□□□□□□□□□□□□□□□□。

1. prefix □□infix□postfix□□□□□□□□□□□□□□□。 prefix□postfix□□□□□□□□□□□□□□□□□□□□□□□□□。□□□□□□ urnary□□8□3++ ** □□□□□□□□□□□□□□□。□infix□□□□□□□□□□□□□□□□□□□□□□□□□□□2+3。
2. □□□□□□□□□□□□□□。□□□□□□□□□□□□□□? ... : Swift 2.x□□□□100□。□□□□□□□□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**4□□□19 □□□**□□□□□。

3.0

```
import Foundation

infix operator **: BitwiseShiftPrecedence

func ** (num: Double, power: Double) -> Double {
    return pow(num, power)
}
```

□Swift 3.0□□□□□□□□□□□□□□□□□BitwiseShiftPrecedence□□□□□□□□□□<< □ >>□□□□□□□□□□□□□□□□□□□□□。

**□□□□□□□□□□□□□□□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□□□□□□□□□□□。

```
func +=<K, V>(lhs: inout [K : V], rhs: [K : V]) { ... }
```

□□□□□

```
let firstDict = ["hello" : "world"]
let secondDict = ["world" : "hello"]
var thirdDict = firstDict + secondDict // ["hello": "world", "world": "hello"]

thirdDict += ["hello":"bar", "baz":"qux"] // ["hello": "bar", "baz": "qux", "world": "hello"]
```

□□□□□

□□□□□□□□□□□□□□□□□□□□CGSize

```
func *(lhs: CGFloat, rhs: CGSize) -> CGSize{
    let height = lhs*rhs.height
    let width = lhs*rhs.width
    return CGSize(width: width, height: height)
}
```

□□□□□□□

```
let sizeA = CGSize(height:100, width:200)
let sizeB = 1.1 * sizeA         //=> (height: 110, width: 220)
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
let sizeC = sizeB * 20          // ERROR
```

□□□□□□□□□□□□□

```
func *(lhs: CGSize, rhs: CGFloat) -> CGSize{
    return rhs*lhs
}
```

□□□□□□□□□□□□。

```
let sizeA = CGSize(height:100, width:200)
let sizeB = sizeA * 1.1              //=> (height: 110, width: 220)
```

□□□□□

Swift Bitwise□□□□□□□□□□□□□□□□□□□□□□□□□。 □□□□□□□□□□0b□□□□□□□□□□□□□□□□0b110□□□□□□□110□□□□□6□。 □□1□0□□□□□□□□□。

□□NOT ~ □

```
var number: UInt8 = 0b01101100
let newNumber = ~number
// newNumber is equal to 0b01101100
```

□□□□□□□□□□□□□□□□□□。 □□□□□□□□□□□UInt8□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□8□。 □□0b01101100□□□□□□UInt□□□□□□0□□□□1□□□□□□□□□□

```
var number: UInt16 = 0b01101100
// number equals 0b0000000001101100
// the 0s are not significant
let newNumber = ~number
// newNumber equals 0b1111111110010011
// the 1s are now significant
```

- 0 － > 1
- 1 － > 0

□□AND & □

```
var number = 0b0110
let newNumber = number & 0b1010
// newNumber is equal to 0b0010
```

□□□□□□&□□□□□□□□□□□□□□□□□□□□□1□□□□□□□1。

- 0□0 － > 0
- 0□1 － > 0
- 1□1 － > 1

□□OR | □

```
var number = 0b0110
let newNumber = number | 0b1000
// newNumber is equal to 0b1110
```

□□□□□□□□|□□□□□□□□□□□□□□□□□□□□□1□□□□□□□□□□□1。

- 0 | 0 － > 0
- 0 | 1 － > 1
- 1 | 1 － > 1

□□□□□□□□ ^ □

```
var number = 0b0110
let newNumber = number ^ 0b1010
// newNumber is equal to 0b1100
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□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

- DefaultPrecedence□□□□□□□TernaryPrecedence □□□□□□□□□□□。□□□□□□□□□□□□□□□□。

- □□□□□□□Apple□API□□□□□
- □□□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	Kitura□Swift 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	□□C□Objective-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	□String□□□□□□UIImage	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