Control Flow, Patterns, and Error Handling

Allen Holub

http://holub.com | Allen Holub | @allenholub

```
let list = [0, 1, 2, 3]


for  element in list  {
    print( "\(element)" )
}
```

```
let list = [0, 1, 2, 3]


for  i in 0 ..< list.count  {
    print( "\(list[i])" )
}
```

```
let list = [0, 1, 2, 3]


for  var i=0; i < list.count; ++i {
    print( "\(list[i])" )
}
```

```
let list = [0, 1, 2, 3]


for _ in 1...5 {
    print( "hello" )
}
```

```
let list = [0, 1, 2, 3]


for ;; {
    print( "hello" )
}
```

```
let airports = ["SFO":"San Francisco",
                "LHR":"London"]

for (code, place) in airports  {
    print( "\(code) in \(place)" )
}
```

```
while condition {
    /*…*/
}
```

```
repeat {
    /*…*/
} while condition
```

```
repeat {
    /*…*/
} while condition

if condition {
    /*…*/
} else if condition {
    /*…*/
} else {
    /*…*/
}
```

```
whileLabel: while someCondition {
    if otherCondition     { continue whileLabel }
    if someOtherCondition { break    whileLabel }
}


outerLabel: if someCondition {
    innerLabel: if otherCondition {
        if someOtherCondition   { break outerLabel }
        if yetAnotherCondition  { break innerLabel }
    }
}
```

```
let someItem: Character = "e"

switch someItem {
  case "a":
      print("vowel")

  default:
      print("consonant")
}
```

```
let someItem: Character = "e"

switch someItem {
  case "a", "e", "i", "o", "u" :
      print("vowel")

  case "a"..."z":
      fallthrough

  default:
      print("consonant")
}
```

```
let someItem = "wildebeest"

switch someItem {
   case "aardvark"..."antelope"
        /*...*/
   case "baboon"..."bushbuck"
        /*...*/
   case "caiman"..."curlew"
        /*...*/
   default:
        print("some other animal")
}
```

```
let aPoint = (1.0, 2.0)

switch aPoint {
   case (let x, 2.0):   print("\(x)")
   case (1.0, let y):   print("\(y)")
   case let(x,y) where x>0.0 && y>0.0:
                        print("\(y),\(x)")
   case let(x,y):       print("\(y),\(x)")
}
```

```
let aPoint  = (x:1.0, y: 2.0)

switch aPoint {
   case (let x, 2.0):   print("\(x)")

   case (1.0, let y):   print("\(y)")

   case let(x,y) where x>0.0 && y>0.0:
                        print("\(y),\(x)")

   default: print("\(aPoint.x) \(aPoint.y)")

}
```

```
let rgbaColor = ( r:1.0, g:1.0, b:1.0, a:1.0 )

switch rgbaColor {
    case (1.0, 1.0, 1.0, 1.0):
        print("white")
    case let (r, g, b, 1.0) where r==g && g==b:
        print("gray")
    case (0.0, 0.5...1.0, let b, _ ):
        print("blue is \(b)")
    default: break
}
```

```swift
let  r=1.0, g=1.0, b=1.0, a=1.0

switch (r,g,b,a) {
    case (1.0, 1.0, 1.0, 1.0):
        print("white")
    case let (r, g, b, 1.0) where r==g && g==b:
        print("gray")
    case (0.0, 0.5...1.0, let b, _ ):
        print("blue is \(b)")
    default: break
}
```

```
enum Status {
        case Okay(status:Int)
        case Error(code: Int, message:String)
        case NA
}
let myStatus = Status.Okay(status:0)

switch myStatus {
    case .NA:                    /*…*/
    case .Okay(0):               /*…*/
    case .Error(0, ""):          /*…*/
}
```

```
enum Status {
        case Okay(status:Int)
        case Error(code: Int, message:String)
        case NA
}
let myStatus = Status.Okay(status:0)

switch myStatus {
    case .NA:                       /*…*/
    case .Okay(0):                  /*…*/
    case .Error(0, _ ):             /*…*/
    case .Error(1..<100, _): /*…*/
    case .Error(let code, let msg):
         print("ERROR \(code): \(msg)")
}
```

```
enum ConnectionStatus { case Down(Status)
                        case Up(Status)
                      }
let connection = ConnectionStatus.Down(Status.NA)

switch connection {
   case .Up  (.Okay):                    /*…*/
   case .Down(.Error):                   /*…*/
   case .Down(.Error(0,_)):              /*…*/
   case .Down(.Error(1..<100,_)):        /*…*/
   case .Down(.Error(let status,_))
            where status==1 || status > 5:   /*…*/
   case .Down:                                /*…*/
   default:                              /*…*/
}
```

So, what is an optional, really?

```swift
enum anOptional {
    case None
    case Some( value: Int )

    mutating func setNil() { self = None }
    mutating func setValue( value: Int ) {
        self = Some(value: value)
    }
    func getValue() -> Int {
        switch self {
        case .None: fatalError("Nil optional")
        case .Some( let value ): return value;
        }
    }
}
```

```swift
enum anOptional {
    case None
    case Some( value: Int )

    mutating func setNil() { self = None }
    mutating func setValue( value: Int ) {
        self = Some(value: value)
    }
    func getValue() -> Int {
        switch self {
        case .None: fatalError("Nil optional")
        case .Some( let value ): return value;
        }
    }
}
```

```swift
var opt : anOptional = anOptional.Some( value: 10)

switch opt {
  case .Some(let val) : print("\(val)")
  case .None :          print("n is nil")
}
```

```swift
enum anOptional {
    case None
    case Some( value: Int )

    mutating func setNil() { self = None }
    mutating func setValue( value: Int ) {
        self = Some(value: value)
    }
    func getValue() -> Int {
        switch self {
        case .None: fatalError("Nil optional")
        case .Some( let value ): return value;
        }
    }
}
```

```swift
var opt : anOptional = anOptional.Some( value: 10)

switch opt {
  case .Some(let val) where val > 0: print("\(val)")
  case .None :                       print("n is nil")
}
```

```swift
enum anOptional {
    case None
    case Some( value: Int )

    mutating func setNil() { self = None }
    mutating func setValue( value: Int ) {
        self = Some(value: value)
    }
    func getValue() -> Int {
        switch self {
        case .None: fatalError("Nil optional")
        case .Some( let value ): return value;
        }
    }
}
```

```swift
var opt : Int? = 10

switch opt {
  case let val?        where val > 0: print("\(val)")
  case  nil  :                        print("n is nil")
}
```

```
enum PostalCode {
    case UK(String)
    case US(Int,Int)
}

let buckingham = PostalCode.UK ("SW1A 1AA")

switch  buckingham {
   case let .UK(val): print("\(val)")
   default: break
}
```

```
enum PostalCode {
    case UK(String)
    case US(Int,Int)
}

let buckingham = PostalCode.UK ("SW1A 1AA")

if case let .UK(val) = buckingham {print("\(val)")}
```

```swift
var arrayOfOptionals:[Int?] = [1, nil, 3]

for case let n? in arrayOfOptionals {
   print(n)
}
```

```
var arrayOfOptionals:[Int?] = [1, nil, 3]

for case let n? in arrayOfOptionals where n < 3 {
    print(n)
}
```

	-Onone	-O
`assert(condition, "Message")`	✔	✘
`assertionFailure("Message")`	✔	✘
`precondition(condition, "Message")`	✔	✔
`preconditionFailure("Message")`	✔	✔
`fatalError("Message")`	✔	✔

Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help

TestAssert 〉 My Mac | Finished running TestAssert : TestAssert

- TestAssert
  - TestAssert
    - main.swift M
  - Products

TestAssert | Resource Tags | Build Settings | Build Phases | Build Rules

Basic | All | Combined | Levels | + | optimiz

Apple LLVM 7.0 - Code Generation

Setting	TestAssert
Enforce Strict Aliasing	Yes
Link-Time Optimization	No
▼ Optimization Level	<Multiple values>
Debug	None [-O0]
Release	Fastest, Smallest [-Os]
Optimization Profile File	/Users/allen/Desktop/TestAssert/OptimizationProfiles/TestAssert.profdata
Relax IEEE Compliance	No
Use Optimization Profile	No

Apple LLVM 7.0 - Language - C++

Setting	TestAssert
C++ Standard Library	libc++ (LLVM C++ standard library with C++11 support)

Swift Compiler - Code Generation

Setting	TestAssert
Disable Safety Checks	No
▼ Optimization Level	<Multiple values>
Debug	None [-Onone]
Release	Fast [-O]

-- INSERT --

```
Hello, World!
assertion failed: Something's Wrong!: file /Users/
allen/Desktop/TestAssert/TestAssert/main.swift, line 13
Program ended with exit code: 9
```

Auto | All Output

Identity and Type

Name: TestAssert

Location: Absolute

TestAssert.xcodeproj

Full Path: /Users/allen/Desktop/TestAssert/TestAssert.xcodeproj

On Demand Resource Tags

Not applicable

Project Document

Project Format: Xcode 3.2-compatible

Organization: Allen Holub

Class Prefix:

Text Settings

Indent Using: Spaces

Widths: 4 | 4

Cocoa Touch Class - A Cocoa Touch class

Test Case Class - A class implementing a unit test

UI Test Case Class - A class implementing a unit test

Xcode File Edit View Find Navigate Editor Product Debug Source Control Window Help

TestAssert 〉 My Mac | Finished running TestAssert : TestAssert

TestAssert
- TestAssert
  - main.swift M
- Products

TestAssert | Resource Tags | Build Settings | Build Phases | Build Rules

Basic | All | Combined | Levels | + | optimiz

Apple LLVM 7.0 - Code Generation

Setting	TestAssert
Enforce Strict Aliasing	Yes
Link-Time Optimization	No
Optimization Level	<Multiple values>
Debug	None [-O0]
Release	Fastest, Smallest [-Os]
Optimization Profile File	/Users/allen/Desktop/TestAssert/OptimizationProfiles/TestAssert.profdata
Relax IEEE Compliance	No
Use Optimization Profile	No

Apple LLVM 7.0 - Language - C++

Setting	TestAssert
C++ Standard Library	libc++ (LLVM C++ standard library with C++11 support)

Swift Compiler - Code Generation

Setting	TestAssert
Disable Safety Checks	No
Optimization Level	**Fast [-O]**
Debug	**Fast [-O]**
Release	Fa[-O]

-- INSERT --

```
Hello, World!
assertion failed: Something's Wrong!: file /Users/
allen/Desktop/TestAssert/TestAssert/main.swift, line 13
Program ended with exit code: 9
```

Auto | All Output

Identity and Type

Name: TestAssert

Location: Absolute

TestAssert.xcodeproj

Full Path: /Users/allen/Desktop/TestAssert/TestAssert.xcodeproj

On Demand Resource Tags

Not applicable

Project Document

Project Format: Xcode 3.2-compatible

Organization: Allen Holub

Class Prefix:

Text Settings

Indent Using: Spaces

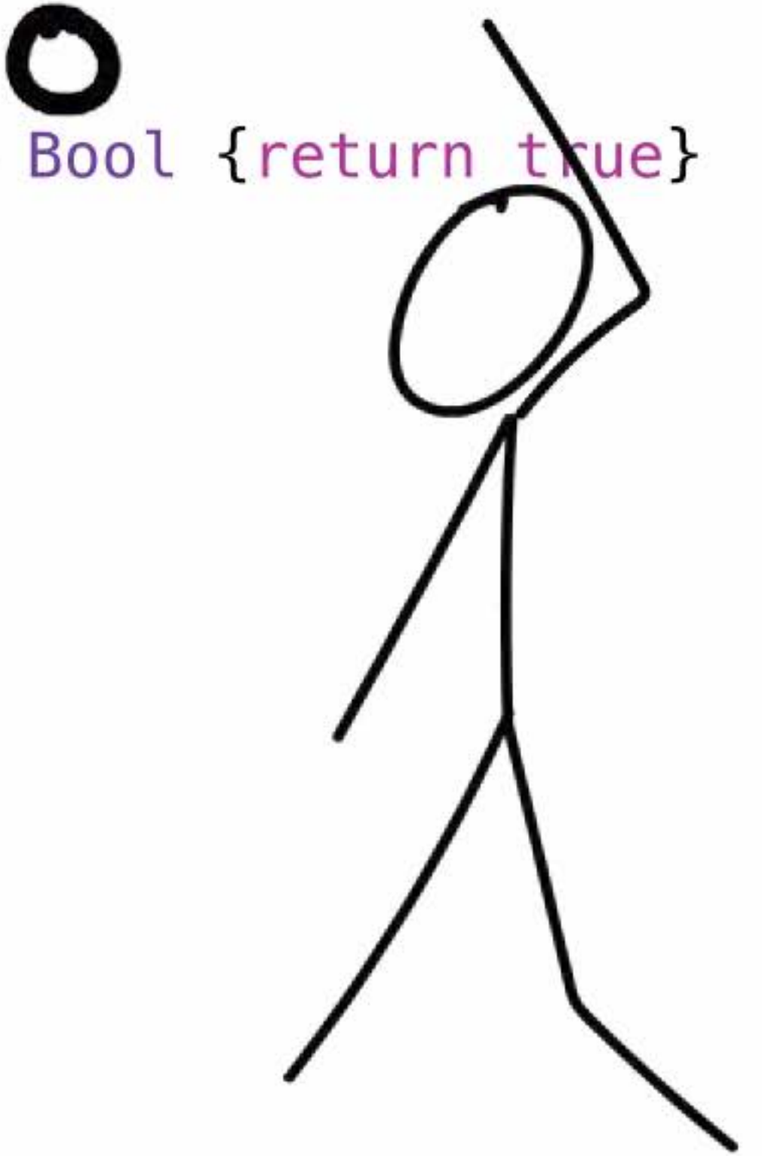
Widths: 4 | 4

Cocoa Touch Class - A Cocoa Touch class

Test Case Class - A class implementing a unit test

UI Test Case Class - A class implementing a unit test

```
class MyFile {
    var isOpen = false
    class func exists(path:String) -> Bool {return true}
    func close()        {/*…*/}
    func length()->Int {return 0}
}
```



```swift
class MyFile {
    enum Error: ErrorType {case NoSuchFile(String), NotOpen}

    init( _ path:String ) throws {
        if !MyFile.exists(path) {
            throw Error.NoSuchFile(path)
        }
    }

    func readLine() throws -> String {
        if !isOpen { throw Error.NotOpen }
        return "A Line"
    }

    var isOpen = false
    class func exists(path:String) -> Bool {return true}
    func close()        {/*…*/}
    func length()->Int {return 0}
}
```

```
func doSomething(x: Int?) {
   assert( MyFile.exists("configuration") )
   let aFile = try! MyFile("configuration")
   defer{ aFile.close() }
   do {
      let theFile = try MyFile("configuration")
      defer{ theFile.close() }
      try theFile.readLine()
   }
   catch let MyFile.Error.NoSuchFile(path) where path=="/x"{
      print("\(path)")
   }
   catch {/*…*/}
}
```

```
var optX :Int? = 10, optY :Int? = nil


if let x=optX {
   if let max=optY   {
      if  0 < x && x < max {
         let optionalReturn = doSomething(x)
         if let o = optionalReturn {
            doSomething(o)
         }
      }
   }
}
```

```
var optX :Int? = 10, optY :Int? = nil

if let x=optX, max=optY where 0 < x && x < max {

   let optionalReturn = doSomething(x)
   if let o = optionalReturn
      doSomething(o)
   }
}
```

```
var optX :Int? = 10, optY :Int? = nil


guard let x=optX, max=optY where 0 < x && x < max
                                        else { return }
let optionalReturn = doSomething(x)
guard let o = optionalReturn else { return }
doSomething(o)
```

```
guard o < 10 else { return }
```

```
guard          else
if  !( o < 10 ) { return }
```

```
if  !( o < 10 ) { return }

if let anInt = optionalInt {
    print("\(anInt)")
}
print("\(anInt)")
```

```
if  !( o < 10 ) { return }

if let anInt = optionalInt {
    print("\(anInt)")
}
else {
    doSomething(anInt)
}
print("\(anInt)")
```

```
guard let anInt = optionalInt
                  else {return}
print("\(anInt)")

doSomething(anInt)

print("\(anInt)")
```