

第二部分 iOS语法基础



授课教师：杨文川



语法基础（4学时）

本章是课程的Swift语言部分，本章的教学要求掌握Swift语言的最基本语法，为后续Swift程序设计打下基础，内容包括：

- 基本数据类型（整形Int、浮点型Double和Float、布尔型Bool、字符串型String、数组型Array和字典型Dictionary、元组Tuple、可选类型Optional）
- 运算符（算术运算符、关系运算符及逻辑运算符等）
- 字符串及其操作
- 集合、数组和字典
- 控制流（for-in语句、while及repeat-while语句、if条件语句、switch条件语句及控制流中的跳转语句）
- 函数（定义、调用、形参、类型和嵌套等）与闭包



基本数据类型

- 常量和变量
- 整型和浮点型
- 布尔型
- 元组型
- 可选型
- 异常处理

常量和变量

常量的值不会发生改变

- 常量的定义
 - *let constantName = value*
- 变量的值在运行过程中会变化
 - *var variableName = value*
- Swift是一个类型安全语言，自动进行类型检查，并对类型不匹配的语句报错提示。
- 在声明时，如果没有显式的指定数据类型，Swift就会根据对其的赋值来进行类型推测。

1	import UIKit	
2	/*	
3	Here we can describe the function of this	
4	file.	
5	Author information should be mentioned.	
6	*/	
7		
8	let lengthOfTelephoneNumber = 8	8
9	let lengthOfIdentityCardNumber = 18	18
10	let lengthOfZipNumber = 6	6
11		
12	var totalVisits = 2000000	2,000,000
13	var numberOfMembers = 10000	10,000
14	var currentSpeed = 120	120
15		
16	var a_b = "a and b", aPerson = "one person"	
17	let hello = "say hello"	"say hello"
18	let mul : String	
19	var numOfApple : Int	
20		
21	//hello = "say hi" need to be fixed	
22	a_b = "c and d"	"c and d"
23		
24	print(a_b)	"c and d\n"
25	print(hello)	"say hello\n"
26	print("The value of variable a_b is \(a_b)")	"The value of variable a_b is c and d\n"
27	print(aPerson)	"one person\n"
28		

代码参见iosSwift的Chapter01-1



整型和浮点型

- 多种长度的整型，如Int8、Int16、Int32、Int64分别表示位长为8、16、32、64位的有符号整型
- 不同位长的无符号整型，分别为UInt8、UInt16、UInt32、UInt64
- 常用的是整型Int
 - 64位的平台上，Int就是Int64
 - 32位的平台上，Int就是Int32
- 浮点型就是带有小数部分的数的类
 - Float为32位浮点数类型
 - Double为64位浮点数类型

1	import UIKit	
2		
3	let minValue = Int8.min	-128
4		
5	let maxOfInt8 = Int8.max	127
6		
7	let maxOfInt16 = Int16.max	32,767
8		
9	let maxOfInt32 = Int32.max	2,147,483,647
10		
11	let maxOfInt64 = Int64.max	9,223,372,036,854,775,807

代码参见iosSwift的Chapter01-2



布尔型

- 布尔型是用来描述逻辑上的真或假的类型。
- 关键字：Bool
- 它有两个值，true和false
- 主要用于判断语句中，用来控制程序根据不同的条件来执行不同的分支流程

```
1  //: Playground - noun: a place where people can play
2
3  import UIKit
4
5  var daytime : Bool
6  daytime = true
7
8  var lightSwitch = true
9
10 if daytime {
11     lightSwitch = false
12     print("It is daytime, so turn lightswitch off!")
13 } else {
14     lightSwitch = true
15     print("It is night, please turn lightswitch on!")
16 }
```

true

true

false

"It is daytime, so turn lightswitch off!\n"

代码参见iosSwift的Chapter01-3



元组型

- 元组型就是由多个类型组成的一个复合类型，其中每一个类型都可以是任意类型，并不要求是相同的类型。
- 读取方式
 - 将元组的值赋值给另一个元组
 - 通过下标来访问元组中的特定元素
 - 在定义元组的时候给每个元素命名。在读取元组的时候，就可以通过这些元素的名字来获取元素的值
- 常用作函数的返回值

```
1  //: Playground - noun: a place where
2    people can play
3
4  import UIKit
5
6  let http404Error = (404, "Not Found") (.0 404, .1 "Not Found")
7
8  let (Code, Description) = http404Error
9  print(Code)                       "404\n"
10 print(Description)                 "Not Found\n"
11
12 print(http404Error.0)              "404\n"
13 print(http404Error.1)              "Not Found\n"
14
15 let http406Error = (Code: 406,
16   Description: "Not Acceptable") (.0 406, .1 "Not Acceptable")
17
18 print(http406Error.Code)            "406\n"
19 print(http406Error.Description)    "Not Acceptable\n"
```

代码参见iosSwift的Chapter01-4

可选型

- 用来表示一个变量或常量可能有值，或没有值的情况
- 声明一个可选型的常量：
 - *let constantName : Type?*
- 声明一个可选型的变量：
 - *var variableName : Type?*

```
41 var num4 : Int?  
42 var str4 : String?
```

```
nil  
nil
```

```
3 import UIKit  
4  
5 var str1 = "85"  
6 var num1 = Int.init(str1)  
7 print(num1)  
8  
9 var str2 = "86"  
10 var num2 = Int.init(str2)  
11 print(num2!)  
12  
13 var str3 = "Hello, playground"  
14 var num3 = Int.init(str3)  
15 print(num3)
```

```
"85"  
85  
"Optional(85)\n"  
  
"86"  
86  
"86\n"  
  
"Hello, playground"  
nil  
"nil\n"
```

代码参见iosSwift的Chapter01-5

可选型的if语句强制解析

- 用if语句通过比较可选型和nil的值是否不相等 (!=) 来判断可选型是否包含值
 - 如果有值，则判断结果为true
 - 如果没有值，则判断结果为false。
- 对于有值的情况，我们可以通过前面介绍的感叹号 “!” 来进行强制解析，从而获得可选型的值

```
17 if (num1 != nil) {  
18     print(num1!)  
19 } else {  
20     print("There is no value of num1")  
21 }  
22  
23 if (num3 != nil) {  
24     print(num1)  
25 } else {  
26     print("There is no value of num3")  
27 }
```

"85\n"

"There is no value of num3\n"

代码参见iosSwift的Chapter01-5

可选型的直接使用

- 在if或while的条件判断语句中，把可选型的值赋给一个临时变量（或常量），如果可选型含有值，则这条赋值语句的值为true，同时该临时变量将获取可选型含有的值；如果可选型的值为nil，则这条赋值语句的值为false，可在此分支输入相应的处理语句

- 格式如下：

```
if let constantName =  
    optionalName  
    { some statements }
```

```
29 if let number = num1 {  
30     print("The value of num1 is \  
        (number)")  
31 } else {  
32     print("num1 is nil")  
33 }  
34  
35 if let number = num3 {  
36     print("The value of num1 is \  
        (number)")  
37 } else {  
38     print("num3 is nil")  
39 }
```

"The value of num1 is 85\n"

"num3 is nil\n"

代码参见iosSwift的Chapter01-5

异常处理

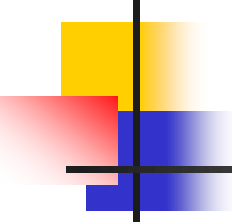
- 异常处理：解决程序运行在过程中可能会遇到各种出错的情况
- 异常处理分为两部分
 - 抛出异常：当一个函数执行中遇到某种错误，函数需要抛出一个异常
 - 捕获异常：函数的调用者处理这个异常

```
3 import UIKit
4
5 enum NameError : ErrorType {
6     case EmptyName
7 }
8
9 func canThrowError(name : String) throws {
10     if name.isEmpty {
11         throw NameError.EmptyName
12     }
13     print("There is no error")
14     print("The name is \(name)")
15 }
16
17 do {
18     try canThrowError("Tommy")
19     try canThrowError("")
20 } catch NameError.EmptyName {
21     print("There is an error")
22     print("name is empty!")
23 }
24
```

"There is no error\n"
"The name is Tommy\n"

"There is an error\n"
"name is empty!\n"

代码参见iosSwift的Chapter01-6



第二部分 语法基础

本章是课程的Swift语言部分，本章的教学要求掌握Swift语言的最基本语法，为后续Swift程序设计打下基础，内容包括：

- 基本数据类型（整形Int、浮点型Double和Float、布尔型Bool、字符串型String、数组型Array和字典型Dictionary、元组Tuple、可选类型Optional）
- 运算符（算术运算符、关系运算符及逻辑运算符等）
- 字符串及其操作
- 集合、数组和字典
- 控制流（for-in语句、while及repeat-while语句、if条件语句、switch条件语句及控制流中的跳转语句）
- 函数（定义、调用、形参、类型和嵌套等）与闭包



运算符

- 赋值运算符
- 算术运算符
- 关系运算符
- 逻辑运算符
- 其它运算符



赋值运算符

- 赋值运算符 “=” 用来初始化或者更新一个变量的值
- 元组也可以通过赋值运算符，对其中的所有元素一次性赋值

```
3 import UIKit
4
5 var str = "Hello, playground"
6
7 var newStr = str
8
9 let (a,b,c) = (1,2,3)
10
11 print(a)
12 print(b)
13 print(c)
```

"Hello, playground"

"Hello, playground"

"1\n"

"2\n"

"3\n"

代码参见iosSwift的Chapter02-1

算术运算符

加减乘除

代码参见iosSwift的Chapter02-2

```
3 import UIKit
4
5 var a : Int = 8
6 var b : Int = 2
7
8 print(a+b)
9 print(a-b)
10 print(a*b)
11 print(a/b)
12
13 var str1 : String = "Hello,"
14 var str2 : String = "world!"
15
16 print(str1+str2)
```

```
8
2
"10\n"
"6\n"
"16\n"
"4\n"
"Hello,"
"world!"
"Hello,world!\n"
```

取模

代码参见iosSwift的Chapter02-3

```
3 import UIKit
4
5 var a : Int = 9
6 var b : Int = 2
7
8 a % b
9
10 var c : Float = 2.5
11 var d : Float = 2.3
12
13 c % d
```

```
9
2
1
2.5
2.3
0.2
```

自增

代码参见iosSwift的Chapter02-4

```
3 import UIKit
4
5 var a = 3
6
7 let b = ++a
8 print("b = \(b) and a = \(a)")
9
10 let c = a++
11 print("c = \(c) and a = \(a)")
```

```
3
4
"b = 4 and a = 4\n"
4
"c = 4 and a = 5\n"
```



关系运算符

- 关系运算符就是用来比较两个值之间的关系的运算符
- 比较的结果为一个布尔值，即：true或者false
- 包括：等于 “==”、不等于 “!=”、大于 “>”、小于 “<”、大于等于 “>=”、小于等于 “<=”
- 非常适合用于条件语句中

```
5 var a = 3, b = 6
6
7 a == b
8 a != b
9 a > b
10 a < b
11 a >= b
12 a <= b
13
14 if a == b {
15     print("a is \(a), b is \(b),
16           they are equal.")
17 } else {
18     print("a is \(a), b is \(b),
19           they are not equal!")
20 }
```

```
false
true
false
true
false
true
```

```
"a is 3, b is 6, they are not equal!\n"
```

代码参见iosSwift的Chapter02-5

逻辑运算符

- 逻辑运算符包括：与 “&&”、或 “||”、非 “!”，操作数为布尔型
- 逻辑非运算是一元运算符
- 逻辑与和逻辑或运算符是二元运算符

```
5 var on : Bool = false
6
7 if !on {
8     print("the light is off")
9 }
```

false

"the light is off\n"

代码参见iosSwift的Chapter02-7

```
3 import UIKit
4
5 var on : Bool = false
6
7 var sunshine : Bool
8 var indoors : Bool
9
10 func lightControl(sunshine:Bool,indoors:Bool) {
11
12     on = (!sunshine)&&indoors
13
14     if !on {
15         print("turn light off")
16     } else {
17         print("turn light on")
18     }
19 }
20 sunshine = false
21 indoors = true
22 lightControl(sunshine, indoors:indoors)
```

false

true

"turn light on\n"

false
true

代码参见iosSwift的Chapter02-8



三元条件运算符

- 格式为：question ? answer1 : answer2
- 语义为：如果question为true，则返回answer1；否则返回answer2
- 等价于：if question : { answer1 } else { answer2 }

5	let contentHeight = 100	100
6	let bottomHeight = 20	20
7	var hasHeader = true	true
8	let pageHeight = contentHeight + bottomHeight + (hasHeader ? 30 : 10)	150

代码参见iosSwift的Chapter02-10



区间运算符

- 闭区间运算符`a...b`表示一个从`a`到`b`的所有值的区间（包括`a`和`b`）。
- 闭区间运算符在循环语句中使用起来非常方便。
- 半闭区间运算符`a..b`表示一个从`a`到`b`的所有值的区间（包括`a`，但不包括`b`）。

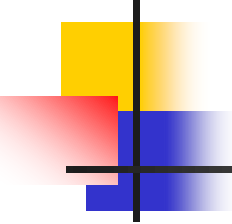
```
5  for i in 0...3 {  
6      print(i)  
7  }
```

(4 times)

```
9  for i in 0..3 {  
10     print(i)  
11 }
```

(3 times)

代码参见iosSwift的Chapter02-11



第二部分 语法基础

本章是课程的Swift语言部分，本章的教学要求掌握Swift语言的最基本语法，为后续Swift程序设计打下基础，内容包括：

- 基本数据类型（整形Int、浮点型Double和Float、布尔型Bool、字符串型String、数组型Array和字典型Dictionary、元组Tuple、可选类型Optional）
- 运算符（算术运算符、关系运算符及逻辑运算符等）
- 字符串及其操作
- 集合、数组和字典
- 控制流（for-in语句、while及repeat-while语句、if条件语句、switch条件语句及控制流中的跳转语句）
- 函数（定义、调用、形参、类型和嵌套等）与闭包



字符串

- 字符串是一组有序字符的集合，用String类型来表示，或者用Character类型的集合来表示。
- 常用操作
 - 空字符串
 - 字符串操作
 - 字符串比较



空字符串

- 在构建一个很长的字符串时，常常以一个空字符串作为初始值，再逐步增加字符串长度。
- 检查一个字符串是否被正确初始化
- 值类型
 - 在进行字符串常量或者变量的赋值操作时，复制的是字符串的值，而不是字符串的指针

```
5 var emptyStr = ""  
6 var anotherEmptyStr = String()
```

```
""  
""
```

```
8 if emptyStr.isEmpty {  
9     print("This String is empty")  
10 }
```

```
"This String is empty\n"
```

```
12 var str1 = "hello world"  
13 var str2 = str1  
14 str2 = "hello China"  
15 print(str1)
```

```
"hello world"  
"hello world"  
"hello China"  
"hello world\n"
```

代码参见iosSwift的Chapter03-1

字符串操作

- 如果要取出字符串中的字符，可以通过for-in循环来遍历字符串的characters属性，从而获得字符串中的每一个字符

```
3 import UIKit
4
5 for charInString in "Hello, world!".characters {
6     print(charInString)
7 }
```

(13 times)

H
e
l
l
o
,
w
o
r
l
d
!

- 计算字符数量

```
5 var str = "Hello, world!"
6 for charInString in str.characters {
7     print(charInString)
8 }
9
10 let countString = str.characters.count
```

"Hello, world!"
(13 times)
13

代码参见iosSwift的Chapter03-2

字符串操作

- 字符串和字符都可以通过加法运算符 “+” 来进行连接
- 字符串中插值
- 保存一个字符串的大写和小写版本

```
12 var str1 = "Hello"
13 var str2 = "world"
14 var character1 = ","
15 var character2 = "!"
16
17 let newString = str1 + character1 + str2 + character2
```

```
"Hello"
"world"
","
"!"
"Hello,world!"
```

```
19 let insertedNum = 888
20
21 let insertString = "The number \(insertedNum)
    is inserted into this string!"
```

```
888
"The number 888 is inserted into this string!"
```

```
29 let theString = "It's my world!"
30 print("the uppercaseString is \(theString.
    uppercaseString)")
31 print("the lowercaseString is \(theString.
    lowercaseString)")
```

```
"It's my world!"
"the uppercaseString is IT'S MY WORLD!\n"
"the lowercaseString is it's my world!\n"
```

代码参见iosSwift的Chapter03-2



字符串比较

- 字符串相等

```
23 var compareStr1 = "It is compare string."  
24 var compareStr2 = "It is compare string."  
25 if compareStr1 == compareStr2 {  
26     print("they are equal!")  
27 }
```

```
"It is compare string."  
"It is compare string."  
  
"they are equal!\n"
```

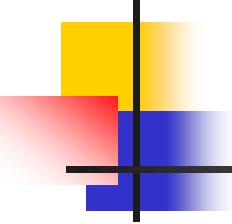
- 字符串前缀/后缀相等：判断一个字符串是否包含特定的前缀字符串

- *theString.hasPrefix(thePrefix)*
- *theString.hasSuffix(theSuffix)*

```
30 let bookInfo1 = "History book: world history"  
31 let bookInfo2 = "History book: China history"  
32 if bookInfo1.hasPrefix("History book") {  
33     print("book1 is a history book.")  
34 }  
35 if bookInfo2.hasSuffix("history") {  
36     print("book2 is a history book.")  
37 }
```

```
"History book: world history"  
"History book: China history"  
  
"book1 is a history book.\n"  
  
"book2 is a history book.\n"
```

代码参见iosSwift的Chapter03-2



第二部分 语法基础

本章是课程的Swift语言部分，本章的教学要求掌握Swift语言的最基本语法，为后续Swift程序设计打下基础，内容包括：

- 基本数据类型（整形Int、浮点型Double和Float、布尔型Bool、字符串型String、数组型Array和字典型Dictionary、元组Tuple、可选类型Optional）
- 运算符（算术运算符、关系运算符及逻辑运算符等）
- 字符串及其操作
- 集合、数组和字典
- 控制流（for-in语句、while及repeat-while语句、if条件语句、switch条件语句及控制流中的跳转语句）
- 函数（定义、调用、形参、类型和嵌套等）与闭包



数组

- 数组是一种按照顺序来存储相同类型数据的集合，相同的值可以多次出现在一个数组中的不同位置。
- 数组是类型安全的，数组中包含的数据类型必须是明确的。
- 数组的声明格式为： *Array[DataType]* 或 *[DataType]*



两个有用的函数

- isEmpty : 用来判断数组是否为空
- append : 用来向数组的末端添加一个元素
- 实例
 - 创建了一个空的字符串数组，然后通过isEmpty来判断数组是否为空，再通过append来添加新的元素到数组中。

```
var animalArray = [String]()  
if animalArray.isEmpty {  
    print("animalArray is empty!")  
}  
animalArray.append("tiger")  
animalArray.append("lion")
```

```
[]  
"animalArray is empty!\n"  
["tiger"]  
["tiger", "lion"]
```

代码参见iosSwift的Chapter04-1

数组初始化

- 在声明数组的时候直接初始化

- 整型数组的直接赋值
- 字符串型数组的直接赋值

```
var oneBitNumberArray : Array<Int> = [0,1,2,3,4,5,6,7,8,9]
var botanyArray1 : [String] = ["rosemary","parsley","sage","thyme"]
var botanyArray2 = ["rosemary","parsley","sage","thyme"]
print("There are \n(botanyArray1.count) kinds of botany.")
```

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

["rosemary", "parsley", "sage", "thyme"]

["rosemary", "parsley", "sage", "thyme"]

"There are 4 kinds of botany.\n"

- 一次性赋值

```
var twoBitNumberArray = [Int](count: 6, repeatedValue: 0)
var threeBitNumberArray = [Int](count: 3, repeatedValue: 11)
var theAddedNumberArray = twoBitNumberArray + threeBitNumberArray
```

[0, 0, 0, 0, 0, 0]

[11, 11, 11]

[0, 0, 0, 0, 0, 0, 11, 11, 11]

代码参见iosSwift的Chapter04-1

数组的操作

- 累加操作
- 下标操作
- 插入和删除
- 遍历：for-in

```
animalArray += ["hawk"]  
animalArray += ["sheep", "horse"]
```

```
["tiger", "lion", "hawk"]  
["tiger", "lion", "hawk", "sheep", "horse"]
```

```
var theFirstAnimal = animalArray[0]  
animalArray[0] = "hen"  
  
animalArray[2...4] =  
    ["goat", "rat", "cock", "rabbit"]
```

```
"tiger"  
"hen"  
  
["goat", "rat", "cock", "rabbit"]
```

```
animalArray.insert("snake", atIndex:  
    3)  
  
animalArray.removeAtIndex(2)  
  
animalArray.removeFirst()  
animalArray.removeLast()
```

```
["hen", "lion", "goat", "snake", "rat", "cock", "rabbit"]  
  
"goat"  
  
"hen"  
"rabbit"
```

```
for animal in animalArray {  
    print(animal)  
}  
  
for (index, animal) in animalArray.  
    enumerate() {  
    print("No.\(index) animal is \  
        (animal)")  
}
```

(4 times)

(4 times)

代码参见iosSwift的Chapter04-1



集合

- 集合中的元素是相同数据类型的，并且元素值是唯一的。
- 集合中的元素是无序的。
- 声明格式为： *Set<DataType>*



集合的初始化

- 从创建一个空的集合开始
 - 需要显式的指出集合中元素的类型
- 通过直接赋值
 - 通过被赋的初始值推断集合中元素的类型

```
var weatherOfSanya = Set<  
    String>()  
weatherOfSanya =  
    ["rainy", "sunny", "stormy"]  
  
var weatherOfBj : Set =  
    ["dry", "windy", "froggy"]
```

```
[]  
  
{"stormy", "rainy", "sunny"}  
  
{"windy", "dry", "froggy"}
```

代码参见iosSwift的Chapter04-2



集合的操作

- 删除操作

```
weatherOfSanya.remove("stormy")  
weatherOfSanya.remove("dry")
```

"stormy"
nil

- 包含操作

```
if weatherOfSanya.contains("sunny")  
{  
    print("Sanya is sunny  
        sometimes.")  
}
```

"Sanya is sunny sometimes.\n"

- 遍历

```
for weather in weatherOfSanya {  
    print("\(weather)")  
}
```

(3 times)

代码参见iosSwift的Chapter04-2



字典

- 通过键值对的形式存储数据，每一个值都对应一个唯一的键。
- 数据的组织是无序的。
- 对字典中值的操作，一般都是基于值所对应的键。
- 声明格式为： *Dictionary<KeyType, ValueType>* 或 *[KeyType : ValueType]*
 - *KeyType*为键的数据类型
 - *ValueType*为值的数据类型

字典的初始化

- 声明并创建一个空字典

```
var ascIIDictChar = Dictionary<Int, Character>() [:]  
var ascIIDictNum = [Int: Int]() [:]
```

- 声明字典变量的时候直接初始化为具体的值

```
var ascIIDictChar = [97:"a", 98:"b",  
99:"c", 100:"d", 101:"e", 102:"f"]
```

```
[100: "d", 101: "e", 99: "c", 97: "a", 98: "b", 102: "f"]
```

```
var ascIIDictNum = [32:0, 33:1, 34:2, 35:3,  
36:4, 37:5, 38:6]
```

```
[32: 0, 34: 2, 36: 4, 38: 6, 35: 3, 33: 1, 37: 5]
```

- 下标语法

```
ascIIDictChar[103] = "g"  
print(ascIIDictChar)  
ascIIDictChar[97] = "A"  
print(ascIIDictChar)
```

```
"g"  
"[103: \"g\", 101: \"e\", 100: \"d\", 99: \"c\", 97: \"a\", 98: \"b\", 102: \"f\"]\n"  
"A"  
"[103: \"g\", 101: \"e\", 100: \"d\", 99: \"c\", 97: \"A\", 98: \"b\", 102: \"f\"]\n"
```

代码参见iosSwift的Chapter04-4

字典的操作

■ 更新元素

```
print(ascIIDictChar)
if let originValue =
    ascIIDictChar.updateValue("a",
    forKey: 97) {
    print("The origin value is \
        (originValue)")
}
print(ascIIDictChar)
```

```
"[103: "g", 101: "e", 100: "d", 99: "c", 97: "A", 98: "b", 102: "f"]\n"
"The origin value is A\n"
"[103: "g", 101: "e", 100: "d", 99: "c", 97: "a", 98: "b", 102: "f"]\n"
```

■ 删除元素

```
print(ascIIDictChar)
ascIIDictChar[97] = nil
print(ascIIDictChar)

if let removedValue =
    ascIIDictChar.
    removeValueForKey(98) {
    print("Value \ \(removedValue)
        is removed.")
}
```

```
"[103: "g", 101: "e", 100: "d", 99: "c", 97: "a", 98: "b", 102: "f"]\n"
nil
"[101: "e", 100: "d", 99: "c", 102: "f", 98: "b", 103: "g"]\n"

"Value b is removed.\n"
```

■ 遍历字典

```
for (ascIICode, char) in
    ascIIDictChar {
    print("ascII code \ \(ascIICode)
        express char \ \(char) ")
}
```

(5 times)

```
ascII code 101 express char e
ascII code 100 express char d
ascII code 99 express char c
ascII code 102 express char f
ascII code 103 express char g
```

代码参见iosSwift的Chapter04-4



keys和values属性

- keys : 键的集合

```
for ascIICode in ascIIDictChar.keys  
{  
    print("keys:\(ascIICode);")  
}
```

(5 times)

```
keys:101;  
keys:100;  
keys:99;  
keys:102;  
keys:103;
```

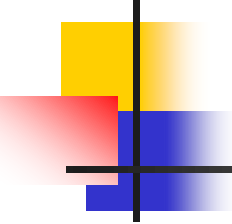
- values : 值的集合

```
for char in ascIIDictChar.values {  
    print("chars:\(char);")  
}
```

(5 times)

```
chars:e;  
chars:d;  
chars:c;  
chars:f;  
chars:g;
```

代码参见iosSwift的Chapter04-4



第二部分 语法基础

本章是课程的Swift语言部分，本章的教学要求掌握Swift语言的最基本语法，为后续Swift程序设计打下基础，内容包括：

- 基本数据类型（整形Int、浮点型Double和Float、布尔型Bool、字符串型String、数组型Array和字典型Dictionary、元组Tuple、可选类型Optional）
- 运算符（算术运算符、关系运算符及逻辑运算符等）
- 字符串及其操作
- 集合、数组和字典
- 控制流（for-in语句、while及repeat-while语句、if条件语句、switch条件语句及控制流中的跳转语句）
- 函数（定义、调用、形参、类型和嵌套等）与闭包



控制流

- for-in循环
- while和repeat循环
- if条件语句
- switch条件语句
- break和continue控制转移语句



for-in循环

- for循环是指按照指定次数，重复执行一系列语句的操作。
- for循环有两种形式，即：for-in循环和for条件递增循环。
- for-in循环主要用来遍历一个特定范围内的所有元素。

```
for i in 1...6 {  
    print("No.\(i);")  
}
```

No.1;
No.2;
No.3;
No.4;
No.5;
No.6;

```
var i : Int  
for i = 0; i < 5; ++i {  
    print("i = \(i)")  
}
```

i = 0
i = 1
i = 2
i = 3
i = 4

代码参见iosSwift的Chapter05-1

while和repeat循环

- while循环，即：在循环执行一系列语句前，先进行条件语句的判断，为false则结束循环，为true则继续执行循环体内语句。
- while循环的格式为：
while condition {
 statements
}
- repeat-while循环，即：执行循环体内的一系列语句，然后进行条件语句的判断，为false则结束循环，为true则继续执行循环体内语句。
- repeat-while循环的格式为：
repeat {
 statements
} while condition

<pre>i = 0 while i < 5 { print("i = \(i)") ++i } i = 0 repeat { print("i = \(i)") ++i } while i < 5</pre>	<pre>0 (5 times) (5 times) 0 (5 times) (5 times)</pre>
---	--

代码参见iosSwift的Chapter05-1



if 条件语句

- if 条件语句的格式为：

```
if condition {  
    statements  
} else {  
    statements  
}
```
- else 语句还可以继续嵌套新的 if-else 语句，嵌套的数量没有限制。

```
if condition1 {  
    statements  
} else if condition2 {  
    statements  
} else {  
    statements  
}
```

```
var weather : String  
weather = "autumn"  
  
if weather == "spring" {  
    print("All trees turn green.")  
} else if weather == "summer" {  
    print("It's too hot.")  
} else if weather == "autumn" {  
    print("Leaves are falling.")  
} else {  
    print("Snow will come!")  
}
```

"Leaves are falling.\n"

代码参见iosSwift的Chapter05-2



switch条件语句

- switch条件语句将一个值与若干个可能匹配的模式进行比较。执行第一个匹配成功的模式所对应的代码。
- 格式如下：
switch someValue {
 case value1:
statementsFor1
 case value2 , 3:
statementsFor23
 default:
statementsForDefault
}

```
enum month {  
    case January  
    case February  
    case March  
    case April  
    case May  
    case June  
    case July  
    case August  
    case September  
    case October  
    case November  
    case December  
}  
var curMonth : month  
  
curMonth = month.February  
  
switch curMonth {  
case month.January : print("the first  
month of a year")  
case month.February : print("There is  
the most important festival in  
this month")  
case  
month.April,month.May,month.June  
: print("month of the second  
season")  
default : print("\(curMonth) doesn't  
match any case.")  
}
```

```
"There is the most important festival in this month\n"
```

代码参见iosSwift的Chapter05-3

break和continue控制转移语句

- 控制转移语句就是改变原有的代码执行顺序，实现代码的跳转。
- break语句可以用于循环语句中，也可以用于其它的控制流语句中。当执行break语句时，直接终止当前控制流，并跳到控制流以外的后续语句处继续执行。
- continue语句在循环语句中使用，当执行continue语句时，本次循环结束，继续下一次循环的执行。
- 两者差别
 - break语句终止全部后续的循环语句
 - continue只是结束当次循环语句的执行。

```
for i in 1...6 {  
    if i == 4 {  
        continue  
    }  
    print("No.\(i)")  
}
```

(5 times)

No.1
No.2
No.3
No.5
No.6

```
for i in 1...6 {  
    if i == 4 {  
        break  
    }  
    print("No.\(i)")  
}
```

(3 times)

No.1
No.2
No.3

代码参见iosSwift的Chapter05-8



第二部分 语法基础

本章是课程的Swift语言部分，本章的教学要求掌握Swift语言的最基本语法，为后续Swift程序设计打下基础，内容包括：

- 基本数据类型（整形Int、浮点型Double和Float、布尔型Bool、字符串型String、数组型Array和字典型Dictionary、元组Tuple、可选类型Optional）
- 运算符（算术运算符、关系运算符及逻辑运算符等）
- 字符串及其操作
- 集合、数组和字典
- 控制流（for-in语句、while及repeat-while语句、if条件语句、switch条件语句及控制流中的跳转语句）
- 函数（定义、调用、形参、类型和嵌套等）与闭包



函数

- 定义和调用
- 函数形参
- 函数类型
- 嵌套函数



定义和调用

- 函数名是用来标识函数的，可以理解为一个函数的代号。
- 要能够比较清楚而简略的描述该函数所完成的任务，从而大大提高函数的可读性
- 要定义函数的传入值的类型，即：形参的类型。
- 要定义函数执行完毕后返回值的类型。
- 要定义函数体，即：一系列执行语句，用来完成特定的任务
- 函数调用：可以通过函数名，并根据形参的类型传入值。当函数有返回值的时候，还要注意接收函数返回值的变量的类型要和返回值类型一致。

```
func mulAdd(mul1:Int,mul2:Int,add:Int) -> Int {  
    let result = mul1*mul2 + add  
    return result  
}
```

```
var result : Int
```

```
result = mulAdd(3, mul2: 5, add: 3)
```

```
print("The result of mulAdd is \(result)")
```

```
18  
18
```

```
18
```

```
"The result of mulAdd is 18\n"
```

代码参见iosSwift的Chapter06-1



函数形参

- 可以有一个形参或多个形参，也可以没有形参
- 不带形参的函数在调用的时候，需要在函数名后面跟着一对空的括号

```
func mulAdd() -> Int {  
    let mul1,mul2,add : Int  
    mul1 = 3  
    mul2 = 5  
    add = 3  
    return mul1*mul2 + add  
}
```

```
var result = mulAdd()
```

```
print("The result of mulAdd is \(result)")
```

3
5
3
18

18

"The result of mulAdd is 18\n"

代码参见iosSwift的Chapter06-2

- 函数也可以没有返回类型

```
func mulAdd(mul1:Int,mul2:Int,add:Int){  
    print("The result of mulAdd is \  
        (mul1*mul2 + add)")  
}
```

```
mulAdd(3,mul2:5,add:3)
```

"The result of mulAdd is 18\n"

代码参见iosSwift的Chapter06-3

函数形参

- 如果函数的返回值为多个，那么就需要用元组来作为返回值类型。

```
func climate(city:String)-
>(averageTemperature:Int,weather:String,
wind:String) {
var averageTemperature : Int
var weather,wind : String
switch city {
case "beijing": averageTemperature = 25;
weather = "dry";wind = "strong"
case "shanghai": averageTemperature = 15;
weather = "wet";wind = "weak"
default : averageTemperature = 10; weather
= "sunny"; wind = "normal"
}

return (averageTemperature,weather,wind)
}

var climateTemp : (Int,String,String)

climateTemp = climate("beijing")
```

(3 times)

(.0 25, .1 "dry", .2 "strong")

(.0 25, .1 "dry", .2 "strong")

代码参见iosSwift的Chapter06-4

- 外部形参：可以明确知道每一个参数的目的，大大减少赋值错误的可能性。

```
func mulAdd(mul no1:Int,mul no2:Int,add
no3:Int){
print("The result of mulAdd is \(no1*
no2 + no3)")
}

mulAdd(mul: 3, mul: 5, add: 3)
```

"The result of mulAdd is 18\n"

代码参见iosSwift的Chapter06-5

函数类型

- 每一个函数都有特定的函数类型。
- 函数类型由形参类型和返回值类型组成。
- 函数的类型可以像其它数据类型一样使用。
 - 实例：直接给变量operation赋值为函数名，可以看到左侧显示，系统将其类型推断为函数类型(Int,Int)->Int

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

```
func helloWorld() {  
    print("Hello world!")  
}
```

<pre>var mathOperation : (Int,Int)->Int = add</pre>	(Int, Int) -> Int
<pre>var sayOperation : ()->() = helloWorld</pre>	() -> ()
<pre>mathOperation(5,6) sayOperation()</pre>	11
<pre>var operation = add operation(6,b:7)</pre>	(Int, Int) -> Int 13

代码参见iosSwift的Chapter06-8



嵌套函数

- 在函数体内定义新的函数，称为嵌套函数。
- 嵌套函数只能在函数体内使用，一般来说对于函数体外是不可见的。

```
func printResult(a:Int, b:Int) {  
    func add(a: Int,b: Int) ->Int{  
        return a + b  
    }  
  
    func sub(a: Int,b: Int) ->Int{  
        return a - b  
    }  
  
    let result : Int  
    if a>b {  
        result = sub(a,b: b)  
    } else {  
        result = add(b,b: a)  
    }  
    print("the result is \(result)")  
}  
  
printResult(6,b: 3)|
```

3

3

"the result is 3\n"

代码参见iosSwift的Chapter06-9



闭包

- 闭包是一种功能性自包含模块，可以捕获和存储上下文中任意常量和变量的引用。
- 闭包的三种形式
 - 全局函数：有名字但不会捕获任何值的闭包。
 - 嵌套函数：有名字并可以捕获其封闭函数域内值的闭包。
 - 闭包表达式：没有名字但可以捕获上下文中变量和常量值的闭包。
- 嵌套函数与闭包表达式
 - 嵌套函数是一种在复杂函数中命名和定义自包含代码块的简洁方式。
 - 闭包表达式则是一种利用内联闭包的方式实现的更为简洁的方式。

闭包表达式

- 闭包表达式的格式为：

```
{ ( parameters ) ->
returnType in
statements
}
```

函数

```
func exchange(s1:String, s2:String)->Bool {
    return s1 > s2
}

let cityArray =
    ["Beijing", "Shanghai", "Guangzhou", "Hangzhou", "Suzhou"]

var descendingArray = cityArray.sort
    (exchange)
```

(10 times)

["Beijing", "Shanghai", "Guangzhou", "Hangzhou", "Suzhou"]

["Suzhou", "Shanghai", "Hangzhou", "Guangzhou", "Beijing"]

- 闭包表达式的参数可以为常量和变量，也可以使用inout类型，但不能提供默认值。
- 在参数列表的最后可以使用可变参数。
- 可以使用元组作为参数和返回值。

闭包表达式

```
cityArray
var descendingArrayByClosures =
    cityArray.sort({(s1:String,
    s2:String)->Bool in return s1 > s2})
descendingArrayByClosures
```

["Beijing", "Shanghai", "Guangzhou", "Hangzhou", "Suzhou"]
(11 times)

["Suzhou", "Shanghai", "Hangzhou", "Guangzhou", "Beijing"]

代码参见iosSwift的Chapter07-1

尾随闭包

- 如果闭包表达式是函数的最后一个参数，那么可以使用尾随闭包的方式来增加代码的可读性。
- 尾随闭包是将整个闭包表达式从函数的参数括号里移到括号外面的一种书写方式。

```
func mathCompute(opr:String,n1:Int,n2:Int,
compute:(Int,Int)->Int) {
    switch opr {
    case "+": print("\(n1)+\(n2) = \(compute
        (n1,n2))")
    case "-": print("\(n1)-\(n2) = \(compute
        (n1,n2))")
    case "*": print("\(n1)*\(n2) = \(compute
        (n1,n2))")
    case "/": print("\(n1)/\(n2) = \(compute
        (n1,n2))")
    default : print("not support this
        operator")
    }
}
```

```
mathCompute("+", n1: 9, n2: 3, compute: {(n1:
    Int,n2:Int)->Int in n1 + n2})
mathCompute("*", n1: 9, n2: 3)
    {(n1:Int,n2:Int)->Int in n1 * n2}
```

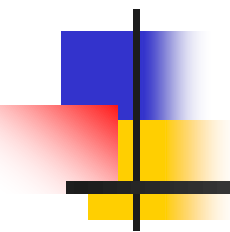
"9+3 = 12\n"

"9*3 = 27\n"

12

27

代码参见iosSwift的Chapter07-2



谢 谢
