swift 4 SL review   LIJIA XU

```
print ("The value is \(myVariable)")
// this is comment
/* comment also */
var a=42       } Let x=0.0, y=0.0, z=0.0
Let one=1      } Int  Double and Float
```

Type Annotations: } Bool string

```
var Msg: String { Msg = "Hello"
```

Operator: Unary, Binary, Tenary

(-, !,), (+, -, *, /, %) (a? b: c)

string concatenation: "Hello, " + "world"

Compond Operators var a=1

a+=2

Comparison Operators ==, !=, >, <, >=, <=

Tenary Operator (question? answer1 : answer2)

```
gender ==0 ? print("male") : print("Female")
```

Range Operators closed range operator ...

1...3 // 1, 2, 3   (a...b)

half-open range operator (a..<b)

1..<3 // 1, 2 { optionial can't contain any-thing else than declared

Logical Operators !a, a&&b, a||b

Optionals: var myCode: Int? = 404

nil ⇒ myCode = nil

Switch with Where

```
Let myPoint (1, -1)
switch myPoint(x, y){
 case Let (x,y) where x==y:
    print(....)
 case Let(x,y) where x==-y:
    print(....)
 Case Let(x,y):
    print("(\(x), \(y))")
}
```

continue: stops loop, restars begining next cycle

```
while a>b{
   print(...)
}
repeat{
 a+=b
}while a>0
for x in 1...5{
   print("\(x)")
}
```

break: stop whole loop

Fallthrough: swift default have break

strings: Let char = "..." check if char.isempty{...} } empty ones: var emptyString = " " var eString = String()

Arrays: Array<T> ⇒ [T] init: var ints = [Int]()

with Default var array = [Double](repeating: 0.0, count: 4)

Literal ⇒ var array: [String] = ["A", "B"]

⇓

type inference var array = ["A", "B"]

check: array.isempty

Moddify: array.append("c") { for x in arrays{ }

range: array[0...1] = ["D", "D"] // ["D", "D", "C"]

array.insert("F", at: 0)

Let removed = array.remove(at: 0)

or ⇒ array.removeLast()

```
for(index, value) in array.enumerated(){
print("Item \(index+1): \(value)")}
```

Sets: Unique elements

Var set = Set<T>()

```
var names: Set<String> = ["A", "B", "C"]
```

{ Infer

```
var names: Set = ["A", "B", "C"]
```

names.insert("D")   if names.contains("A") {...}else{ }

```
for name in names{
 ....
}
```

or

```
For name in names.sorted(){
 ....}
```

setA.union(setB) sorted

Dictionaries: Dictionary<key, value> ⇒ [key: value]

```
var dics = [Int: String]()
```

remove: dics["A"] = nil

```
* let oldValue = dics.updateValue("A", forKey "B")
if let removed = dics.removeValue(forKey: "B"){
   print(....)}else{
   print(....)}
for( key, value) in dics{
....}
```

Tuples <

```
func fun(x: Int, y: Int)->(z: Int, k: Int){... return(z,k)}
func someFunction(P1: Int=12){}
someFunction(P1: 6)
someFunction()
```

Inout funcs

```
swapInts(a: &x, b: &y)
func swapInts(a: inout Int,
b: inout Int){
Let tempA=a
 a=b
 b=tempA}
```

Function Types as Parameter Types

```
func printResult(mathFunc: (Int,
Int) -> Int, a: Int, b: Int){ }
printResult(addInts, 2, 3}
```

closures:

global and nested functions are speial kind of

{(parameters) ⇒ return type in statements}

The sorted method: `var reversed = names.sorted(by: {(s1: String, s2: String) -> Bool in return s1>s2})`

Inteering Type ⇒ `var reversed = names.sorted(by: {(s1, s2) in return s1>s2})` single line omit

`var reversed = names.sorted({(s1, s2) in s1>s2})`

shorthad argument ⇒ `var reversed = names.sorted(by: {$0 > $1})` * `reversed = names.sorted(by: >)`

Tuples ⇒ Let error = (404, "Not Found")

Let (statusCode, statusMessage) = error

can be broken down into separate constant or vars

or use index numbers // can name individal names

let error = (statusCode: 404, statusMsg: "Not found")

print("\(statusCode)") or print("\(error.0)")

Enumerations: common type for a group of values

*Unlike C and object-C, swift Enum no defaultval

```
enum Compass{
 case North
 case South
 case East
 case West
}
```

(or) enum Compass{case N, S, E, W}

var direction = Compass.east

instances of Class or Structure

Structures are value types

all basic types in Swift (Int, Bool ...) are Value types

Classes are Reference Types; check if reference the same instance ⇒ === or !==

*String, Array and Dictionary are structures

A lazy stored property's initial value is not calculated until the first time is used

lazy must be variable

a computed property provides a getter and, optionally, a setter

computed property with a getter but no setter is called read-only computed property

Property Observers: called every time a value set

```
Var totalSteps: Int = 0{ willSet(newStep){
print(" to \(newStep)")}didSet{ if totalSteps
>oldValue{print("added \(totalSteps-oldValue)")}}}
```

Type properties class SomeClass{

static var someString = "Something!"}

print(SomeClass.someString)

modifying Value Types ⇒ mutating keyword

Type methods: static infront of func

Subscripts: access values from an instance

```
struct TimeTable{ let multi: Int
 Subscript(index: Int) -> Int{
 return multi * index}}
Let threeTimes = TimeTable(multi: 3)
print(threeTimes[5]) // prints 15
```

Do not write override when override required designed initlizer

※ all stored properties of a class, must be assigned an initial value ~~with~~ init

deinit{ } ⇒ only for classes