Swift Cheat Sheet (Basics)

Declaring Constants

```
let radius = 3.45
let numOfColumns = 5
let myName = "Wei-Meng Lee"
```

Declaring Variables

```
let radius = 3.45
var myAge = 25
var circumference =
    2 * 3.14 * radius
var rate:Int = 2
```

Printing

```
print()
```

Type Alias

```
typealias CustomerIDType = UInt32
typealias CustomerNameType = String

var customerID: CustomerIDType
var customerName:
    CustomerNameType
customerID = 12345
customerName = "Chloe Lee"
```

Tuples

```
var pt1 = (7,8)
var pt2: (Int, Int)
pt2 = (7,8)

var flight = (7031, "ATL", "ORD")
let (flightno, orig, dest) =
    flight
print(flightno)  //---7031---
print(orig)      //---ATL---
print(dest)      //---ORD---

print(flight.0)  //---7031---
print(flight.1)  //---ATL---
print(flight.2)  //---ORD---
```

Optionals

```
let str = "125"
let num:Int? = Int(str)
```

Unwrapping Optionals

```
if num != nil {
    let multiply = num! * 2
    print(multiply)
}
```

Implicitly Unwrapped Optionals

```
let str = "125"
let num:Int! = Int(str)
if num != nil {
    let multiply = num * 2
    print(multiply)
}
```

Conditional Unwrapping

```
var s1:String?
print(s1?.characters.count)
    //---prints out nil---
print(s1!.characters.count)
    //---crash---
```

Optional Binding

```
func getProductCode(code:String) ->
String? {
    if code == "Diet Coke" {
        return "12345"
    } else {
        return nil
    }
}

if let productCode =
    getProductCode(
        code: "Diet Coke") {
    print(productCode)
} else {
    print("Not found")
}
```

Enumerations

```
enum BagColor {
    case Black
    case White
    case Red
    case Green
    case Yellow
}
var colorOfBag:BagColor
colorOfBag = BagColor.Yellow
// OR
colorOfBag = .Yellow
```

Enumeration Raw Values

```
enum BagColor: String {
    case Black  = "Black"
    case White  = "White"
    case Red    = "Red"
    case Green  = "Green"
    case Yellow = "Yellow"
}

var colorOfBag:BagColor
colorOfBag = BagColor.Yellow
var c = colorOfBag.rawValue
print(c)  //---"Yellow"---

var colorOfSecondBag:BagColor? =
    BagColor(rawValue:"Green")

if colorOfSecondBag ==
    BagColor.Green {
    ...
}
```

AutoIncrement for Raw Values

```
enum DayOfWeek: Int {
    case Monday = 1
    case Tuesday
    case Wednesday
    case Thursday
    case Friday
    case Saturday
    case Sunday
}

var day:DayOfWeek
day = DayOfWeek.Tuesday
print (day)         //---"Tuesday"---
print (day.rawValue)    //---2---
```

Strings

```
var str1 = "A string"
var str2:String = "A string"
var str3 = str1 + str2
var str4 = "A" + " " + "String"
```

Characters

```
var euroStr = "€"
   //---String---
var euro:Character = "€"
   //---Character---
var price = String(euro) + "2500"
   //---€2500---
```

Unicode

```
let hand:Character = "\u{270B}"
let star = "\u{2b50}"
let bouquet = "\u{1F490}"
```

Casting String as NSString

```
var str1 =
    "This is a Swift string"
print(
    (str1 as NSString).length)
```

Declaring as NSString

```
var str1:NSString =
    "This is a NSString..."
var str2 =
    "This is a NSString..." as
    NSString

print(str2.length)
print(str2.contains(
    "NSString"))
print(str2.hasPrefix("This"))
print(str2.hasSuffix("..."))
print(str2.uppercased)
print(str2.lowercased)
print(str2.capitalized)
```

Nil Coalescing Operator

```
var gender:String?
var genderOfCustomer =
  gender ?? "male" //---male---

gender = "female"
genderOfCustomer =
  gender ?? "male" //---female---
```

Range Operators

```
//---Closed Range Operator---
//---prints 5 to 9 inclusive---
for num in 5...9 {
    print(num)
}

//---Half-open Range Operator
//---prints 5 to 8---
for num in 5..<9 {
    print(num)
}
```

Functions

```
func addNums(
    num1: Int,
    num2: Int,
    num3: Int) -> Int {
    return num1 + num2 + num3
}
var sum =
addNums(num1:1, num2: 2, num3: 3)
```

Returning Tuple

```
func countNumbers(string: String)
    -> (odd:Int, even:Int) {
    var odd = 0, even = 0
    ...
    return (odd, even)
}
```

Function Parameter Name

```
func doSomething(
    num1: Int,
    secondNum num2: Int) {
    ...
}

doSomething(num1:5, secondNum:6)
```

External Parameter Names Shorthand

```
func doSomething(
    _ num1: Int, num2: Int) {
}
doSomething(5, num2:6)

func doSomething(
    _ num1: Int, _ num2: Int) {
}
doSomething(5, 6)
```

Default Parameter Value

```
func join(firstName:String,
              lastName:String,
              joiner:String = " ")
    -> String {
    ...
}

var fullName = join(
    firstName: "Wei-Meng",
lastName:"Lee",
    joiner:",")

fullName = join(
    firstName: "Wei-Meng",
lastName:"Lee")
```

Variadic Parameters

```
func average(nums: Int...) ->
    Float {
    var sum: Float = 0
    for num in nums {
        sum += Float(num)
    }
    return sum/Float(nums.count)
}
var avg = average(nums:1,2,3,4,5,6)
```

In-Out Parameters

```
func fullName(
    name:inout String,
          withTitle title:String)  {
    ...
}
var myName = "Wei-Meng Lee"
fullName(name: &myName,
         withTitle:"Mr.")
```

Arrays

```
var names = [String]()
var addresses:[String] =
    [String]()
names.append("Lee")
addresses.append("Singapore")

var OSes:[String] = ["iOS",
    "Android", "Windows Phone"]

var numbers:[Int] =
    [0,1,2,3,4,5,6,7,8,9]

var item1 = OSes[0] // "iOS"
var item2 = OSes[1] // "Android"
var item3 = OSes[2] // "Windows
                    // Phone"
var count = OSes.count // 3
```

Dictionaries

```
var platforms1:
    Dictionary<String, String> = [
       "Apple": "iOS",
       "Google" : "Android",
       "Microsoft" : "Windows Phone"
]

var platforms2 = [
    "Apple": "iOS",
    "Google": "Android",
    "Microsoft": "Windows Phone"
]
print(platforms1["Apple"]!)
//---"iOS"---

var count = platforms1.count
let companies = platforms1.keys
let oses = platforms1.values

var months =
    Dictionary<Int, String>()
months[1] = "January"
months = [:] // empty again
```

Switch Statement

```
var grade: Character
grade = "A"
switch grade {
    case "A", "B", "C", "D":
        print("Passed")
    case "F":
        print("Failed")
    default:
        print("Undefined")
}
```

Explicit Fallthrough

```
var grade: Character
grade = "A"
switch grade {
    case "A":
        fallthrough
    case "B":
        fallthrough
    case "C":
        fallthrough
    case "D":
        print("Passed")
    case "F":
        print("Failed")
    default:
        print("Undefined")
}
```

Matching Range

```
var percentage = 85
switch percentage {
    case 0...20:
        print("Group 1")
    case 21...40:
        print("Group 2")
    case 41...60:
        print("Group 3")
    case 61...80:
        print("Group 4")
    case 81...100:
        print("Group 5")
    default:
        print(
        "Invalid percentage")
}
```

Matching Tuples

```
//---(math, science)---
var scores = (70,40)
switch scores {
    case (0,0):
        print("Not good!")
    case (100,100):
        print("Perfect!")
    case (50...100, _):
        print("Math passed!")
    case (_, 50...100):
        print(
            "Science passed!")
    default:
        print("Both failed!")
}
```

Labeled Statement

```
var i = 0
outerLoop: while i<3 {
    i += 1
    var j = 0
    while j<3 {
        j += 1
        print("(\(i),\(j))")
        break outerLoop
        //---exit the outer While
        // loop---
    }
}
```

Structures

```
struct Go {
    var row = 0
    var column = 0
}

var stone1 = Go()
print(stone1.row)    //---0---
print(stone1.column) //---0---
stone1.row = 12
stone1.column = 16
```

Memberwise Initializer

```
struct Go {
    var row:Int
    var column:Int
}
var stone1 =
    Go(row:12, column:16)

var stone2 = Go() //---error---
```

Looping

```
// prints 0 to 4
var count = 0
while count < 5 {
    print (count)
    count += 1
}

// prints 0 to 4
count = 0
repeat {
    print (count)
    count += 1
} while count < 5

// prints 0 to 4
for i in 0 ..< 5 {
    print (i)
}

// prints 4 to 0
for i in (0..<5).reversed() {
    print (i)
}

// prints all even nums from 0 to 9
for i in (0..<10) where i % 2 == 0 {
    print (i)
}

// prints 1 to 5
for i in stride(from:1, through: 5,
by: 1) {
   print (i)
}

// prints 1,3,5
for i in stride(from:1, through: 5,
by: 2) {
    print (i)
}

// prints 5 to 1
for i in stride(from:5, through: 1,
by: -1) {
    print (i)
}
```