Chapter IV

Swift Programming Basic

IV.A Brief History of Swift

A general-purpose, multi-paradigm, compiled programming language created for iOS, OS X, watchOS, tvOS and Linux development by Apple Inc. introduced at Apple's 2014 Worldwide Developers Conference (WWDC). It underwent an upgrade to version 1.2 during 2014 and a more major upgrade to Swift 2 at WWDC 2015.version 2.2 was made open source on December 3, 2015 for Apple's platforms and Linux.Version 3.0 is out publicly September 13, 2016.

1. Swift Introduction

Swift is a new programming language developed by Apple Inc for iOS and OS X development. Swift adopts the best of C and Objective-C, without the constraints of C compatibility. Swift designers took ideas from various other popular languages like Objective-C, Rust, Haskell, Ruby, Python, C#, CLU, and far too many others to list.

Highlights of Swift Features at a glance

- Swift makes use of safe programming patterns.
- Swift provides modern programming features.
- Swift provides Objective-C like syntax.
- Swift is a fantastic way to write iOS and OS X apps.
- Swift provides seamless access to existing Cocoa frameworks.
- Swift unifies the procedural and object-oriented portions of the language.
- Swift does not need a separate library import for functionality like input/output or string handling.

In Xcode, start with a playground option and enter a name for playground and select iOS as platform. Finally you will get Playground window as follows:

```
import UIKit
var str = "Hello, playground”
```

2. Variables & Data Types

a. Identifiers

A Swift identifier is a name used to identify a variable, function, or any other user-defined item. a letter A to Z or a to z or an underscore _ followed by zero or more letters, underscores, and digits (0 to 9) and not allow punctuation characters such as @, $, and % within identifiers. Swift is a case sensitive programming language.

Example of Acceptable Characters

```
Azad      zara   abc   move_name  a_123
myname50  _temp  j     a23b9      retVal
```

The following keywords are reserved in Swift.

Keywords used in declarations

class	**deinit**	**enum**	**extension**
func	**import**	**init**	**internal**
let	**operator**	**private**	**protocol**
public	**static**	**struct**	**subscript**
typealias	**var**		

Keywords used in statements

break	**case**	**continue**	**default**
do	**else**	**fallthrough**	**for**
if	**in**	**return**	**switch**
where	**while**		

Keywords used in expressions and types

as	**dynamicType**	**false**	**is**
nil	**self**	**Self**	**super**
true	**_COLUMN_**	**_FILE__FUNCTION_**	
LINE			

Keywords used in particular contexts

associativity	**convenience**	**dynamic**	**didSet**
final	**get**	**infix**	**inout**
lazy	**left**	**mutating**	**none**
nonmutating	**optional**	**override**	**postfix**
precedence	**prefix**	**Protocol**	**required**
right	**set**	**Type**	**unowned**
weak	**willSet**		

Whitespaces

b. Swift Datatype

Swift offers the programmer a rich assortment of built-in as well as user defined data types. Following is a list of basic data types which will be used most frequently when declaring variables:

Int or UInt - This is is used for whole numbers. More specifically you can use Int32, Int64 to define 32 or 64 bit signed integer where as UInt32 or UInt64 to define 32 or 64 bit unsigned integer variables. For example, 42 and -23.

Float - This is used to represent a 32-bit floating-point number and used for numbers with smaller decimal points. For example 3.14159, 0.1, and -273.158.

Double - This is used to represent a 64-bit floating-point number and used when floating-point values must be very large. For example 3.14159, 0.1, and -273.158.

Bool - This represents a boolean value which is either true or false.

String - This is ordered collection of characters. For example, "Hello, World!"

Character - This is a single-character string literal. For example, "C"

Optional - This represents a variable that can hold either a value or no value.

Before you use variables, you must declare them using **var** keyword:

```
var variableName = <initial value>
```

c. Swift - Constants

The constants refer to fixed values that the program may not alter during its execution. Constants can be of any of the basic data types like an integer constant, a floating constant, a character constant, or a string literal.

Using let keyword as follows:

let constantName = <initial value>

e.g

```
let constA = 42
print(constA)
```

d. Tuple

A tuple is a group of zero or more values represented as one value.The type of a tuple is determined by the values it has.You can use tuples to initialize more than one variable on a single line:

```
var person = ("John", "Smith")


var firstName = person.0 // John

var lastName = person.1 // Smith
```

You can name the elements from a tuple and use those names to refer to them. An element name is an identifier followed by a colon(:).

```
var person = (firstName: "John", lastName: "Smith")

var firstName = person.firstName // John

var lastName = person.lastName // Smith
```

Note: Tuple are value types. When you initialize a variable tuple with another one it will actually create a copy.

e. Bound Values (Maximum and Minimum allowable value)

f. Swift Optional

Swift also introduces optionals type, which handles the absence of a value. Optionals say either

"**there is a value, and it equals x" or "there isn't a value at all**".

Optional Typed variable has two possible values, None and Some(T), where T is an associated value of the correct data type available in Swift.

```
var perhapsInt: Int?

var perhapsStr: String?
```

Above declaration is equivalent to explicitly initialising it to nil which means no value:

```
var perhapsStr: String?  = nil
```

Let's say see following example to understand how optionals is working in Swift:

```
var myString:String? = nil


if myString != nil {

print(myString)
```

```
//1.Variable
//Int, IntX, UIntX
//Double, Float
//String

var num1   = 1  // implicit  //OS Depended
var num2:Int = 2 //Explicit , Annotation
var num3 = num1 + num2 //Operation => must be of equal type

var num4 = 3.5
var num5:Double = 4.5
var num6:Float = 3.5


var num7 = num5 + Double( num6 )
print(num7)

var greetingText = "Hello"
var ver = 4.2

// "Hello Swift 4.2" is
print(greetingText + " Swift " + String(ver) )
print("\(greetingText) Swift \(ver+1.0) ") //interpolate


//Constant variables with let
let g = 9.8
let v0 = 5.0
var v1 = v0 + g * 10


//2.Operators
// + , - , / , * ,  && , || , == , != , <= , >= , < , > , &
, | ,<< , >> ,


//3.Collection - Tuple , Array , Dictionary, Set
//Tuple
var person = ("John", "Smith")

var firstName = person.0 // John

var lastName = person.1 // Smith


var aStudent = (name:"Marry",grade:5,age:10.2)
print(aStudent.name)
```

```
}else{
print("myString has nil value")
}
```

```
--------------------------[Result]---------------------------------------
myString has nil value
```

Optionals are similar to using nil with pointers in Objective-C, but they work for any type, not just classes.

1. Forced Unwrapping

If you defined a variable as optional then to get the value from this variable you will have to unwrap it. This just means putting an exclamation mark at the end of the variable.

e.g

```
import UIKit
var myString:String?
myString = "Hello, Swift!"
var theOtherString = myString!

print(theOtherString)
```

```
--------------------------[Result]---------------------------------------
Optional("Hello, Swift!")
```

2. Automatic Unwrapping

You can also declare you optional variables using exclamation mark '!' instead of a question mark '?'. Such optional variables will unwrap automatically and you do not need to use any further exclamation mark at the end of the variable to get assigned value.

e.g

```
import UIKit
var myString:String!
myString = "Hello, Swift!"
if myString != nil {
print(myString)
}else{
print("myString has nil value")
```

}

---------------------------[Result]--

Hello, Swift!

3. Optional Binding

```
//3.Collection - Array
var numbers1:[Int] = [10,20,30,40,50,60,70,80,90]
var numbers2 = [1,2,3,4,5,6.5]  //auto double array
var numbers3:[Float]  = [1,2,3,4,5,6.5] // float array ,
                                      //must set Explicitly
var strArray:[String] = ["A","B","C","D","E"]

var numbersArray = [Int]() //create empty interger array
var dummyArray:[Any] = [1,"One", 4.5]


print(numbers1.first)
print("First Element:==>",numbers1[0])
print("Total Elements:==>",numbers1.count)
print("last elements:==>", numbers1[ numbers1.count - 1 ])

//3.Collection - Dictionary

var dict = [ "a":"Apple"  , "b":"Banana" ,"c":"Cat",
             “d":"Dragon","e":"Elephant"]


var priceList:[String :Double] = [“Egg": 150.0,
                                  “Chicken": 2000,
                                  “Mushroon": 200,
                                  “Fish":3000 ]
print(priceList["Egg"])

//3.Collection - Set
var uniqueNumbers1:Set<Int> = [1,2,3,4,5]

uniqueNumbers1.insert(5)
//5 already exist , so still having the same element
print(uniqueNumbers1.count)

var uniqueNumbers2:Set<Int> = [2,4,8,6,8]
var un12Union = uniqueNumbers1.union(uniqueNumbers2)
var un12Intercept =
uniqueNumbers1.intersection(uniqueNumbers2)
var un12subtract =
uniqueNumbers1.subtracting(uniqueNumbers2)

print(un12Union,un12Intercept,un12subtract)
```

You use optional binding to find out whether an optional contains a value, and if so, to make that value available as a temporary constant or variable.

A optional bindings for the if statement is as follows:

```
//4.Optional

var numberOpt:Int? = 3
print(numberOpt)
numberOpt = nil
print(numberOpt)

var numberOptAuto:Int!
numberOptAuto = 6
print(numberOptAuto)

//Check whether nil or not
//as IT IS NOT SURE VALUE HAVING OR NOT?

numberOpt =  15
if numberOpt != nil {
    //to get the value from optional variable
    //apply force unwrapping, using !
    var addition =  numberOpt! +  5
    print("Added by Optional",addition)
}

if numberOptAuto != nil {
    var addition =  numberOptAuto  +  5 //note, ! is absent
print("Added by Optional Auto",addition)
}

//using Optional Binding
if let numberNOpt = numberOpt {

    var addition = numberNOpt + 5
     //numberNopt become constant as of let

    print("Added using optional binding",addition)
}
```

if let constantName = someOptional {

statements

```
}
e.g
        import UIKit
        var myString:String?
        myString = "Hello, Swift!"
        if let yourString = myString {
                print("Your string has - \(yourString)")
        }else{
                print("Your string does not have a value")
        }
---------------------------[Result]----------------------------------------
Your string has - Hello, Swift!
```

3. Swift Operators

An operator is a symbol that tells the compiler to perform specific mathematical or logical manipulations. Objective-C language is rich in built-in operators and provides following types of operators:

- Arithmetic Operators
- Comparison Operators
- Logical Operators
- Bitwise Operators
- Assignment Operators
- Range Operators
- Misc Operators

Note: Operator precedence determines the grouping of terms in an expression. This affects how an expression is evaluated.

4. Swift Arrays

Swift arrays are used to store ordered lists of values of the same type. Swift puts strict checking which does not allow you to enter a wrong type in an array even by mistake.

You can create an empty array of a certain type using following initializer syntax:

var someArray = [SomeType]()

e.g To create an array of a given size and with an initial value:

var someArray = [SomeType](count: NumberOfElements, repeatedValue: InitialValue)

e.g To create an empty array of Int type having 3 elements and initial value as zero:

var someInts = [Int](count: 3, repeatedValue: 0)

```
//5.Conditional Statement
let mark =  50
if mark >= 40 {

    if mark >= 75 {
        print("Grade A+")
    }else if mark >= 70 {
        print("Grade A")
    }else if mark >= 65 {
        print("Grade B+")
    }else if mark >= 60 {
        print("Grade B")
    }else if mark >= 55 {
        print("Grade C+")
    }else if mark >= 50 {
        print("Grade C")
    }else if mark >= 45 {
        print("Grade D+")
    }else  {
        print("Grade D")
    }
} else {
    print("Failed")
}
```

e.g To create an array of three elements and assign three values to that array:

```
//6.Looping
var array = [ 10,20,30,40,50]

//While
//1. initialise
//2. Loop exit condition
//3. Condition must be met, inc or dec to the index or x

var i = 0 //1
while (i < array.count) { //2
    print(i, array[i])
    i += 2 //3
}
print("\\\\\\\\\\\\")

for e in array {         print(e)
}
print("----------")

for i in 0..<array.count { // 0..<5  , 0...5
    print(array[i])
}

//Try stride(from: 0, to: array.count, by:1)
var j = 10.     //use to execute one time at least
repeat {

    print("=>",j)
    j += 1
}
while j <= 20
```

var someInts:[Int] = [10, 20, 30]

Note: Array index starts from 0 and you can retrieve and modify a value of the array by using subscript syntax as somearray[indexNumber].

You can add additional element to array using **append** method and count the number of elements by **count** properties. Also, **isEmpty** let you determine the array has no elements or not

5. Swift Dictionaries

Swift dictionaries are used to store unordered lists of values of the same type. Swift puts strict checking which does not allow you to enter a wrong type in a dictionary even by mistake.

Swift dictionaries use unique identifier known as a key to store a value which later can be referenced and looked up through the same key. Unlike items in an array, items in a dictionary do not have a specified order. You can use dictionary when you need to look up values based on their identifiers.

A dictionary key can be either integer or string without a restriction, but it should be unique with-in a dictionary.

If you assign a created dictionary to a variable then its always mutable which means you can change it by adding, removing, or changing its items but if you assign a dictionary to a constant then that dictionary is immutable, and its size and contents cannot be changed.

You can create a empty dictionary of a certain type using following initializer syntax:

var someDict = [KeyType: ValueType]()

e.g

var someDict = [Int: String]()

```
//6.Looping with control statement

print("Using continue to skip, without 30")
for e in array {      // stride(from: 0, to: array.count,
by:
    if e == 30 {
        continue
    }
    print(e)
}
print(“----------")

print("Using break to end loop, after 30")
for e in array {      // stride(from: 0, to: array.count,
by:

    print(e)
    if e == 30 {
        break
    }
}
print("----------")
```

var someDict:[Int:String] = [1:"One", 2:"Two", 3:"Three"]

You can retrieve and modify a value from a dictionary by using subscript syntax, passing the key of the value you want to retrieve within square brackets immediately after the name of the dictionary as follows:

var someVar = someDict[key]

Note: you can use removeValueForKey() method to remove a key-value pair from a dictionary.This method removes the key-value pair if it exists and returns the removed value, or returns nil if no value existed

6. Control Flow: Decision Making

Decision making structures require that the programmer specify one or more conditions to be evaluated or tested by the program, along with a statement or statements to be executed if the condition is determined to be true, and optionally, other statements to be executed if the condition is determined to be false

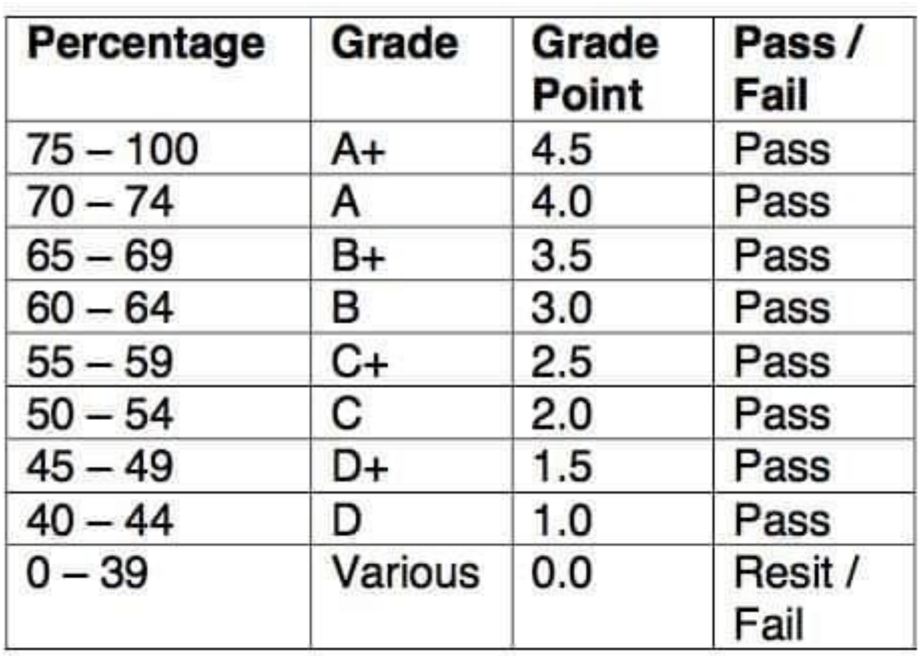
Statement	**Description**
if	An if statement consists of a boolean expression followed by one or more statements.
if...else	An if statement can be followed by an optional else statement, which executes when the boolean expression is false.
if...else if...else	An if statement can be followed by an optional else if...else statement, which is very useful to test various conditions using single if...else if statement.
nested if	You can use one if or else if statement inside another if or else if statement(s).

Notes:

The ? : Operator can be used to replace if...else statements. It has the following general form:

Exp1 ? Exp2 : Exp3; Where Exp1, Exp2, and Exp3 are expressions. Notice the use and placement of the colon.

Ex. Consider the following marking scheme and express with if else if statement to determine , fail , pass and grading



Percentage	Grade	Grade Point	Pass / Fail
75 – 100	A+	4.5	Pass
70 – 74	A	4.0	Pass
65 – 69	B+	3.5	Pass
60 – 64	B	3.0	Pass
55 – 59	C+	2.5	Pass
50 – 54	C	2.0	Pass
45 – 49	D+	1.5	Pass
40 – 44	D	1.0	Pass
0 – 39	Various	0.0	Resit / Fail

7. Swift Loops

Looping are used when you need to execute a block of code several number of times.

Swift programming language provides following kinds of loop to handle looping requirements

Loop Type	Description
for-in	This loop performs a set of statements for each item in a range, sequence, collection, or progression.
while loop	Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body.
repeat .... while	Like a while statement, except that it tests the condition at the end of the loop body.

1. Loop Control Statements

Loop control statements change execution from its normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed.Swift supports the following control statements

Statement	Description
continue	This statement tells a loop to stop what it is doing and start again at the beginning of the next iteration through the loop.
break	Terminates the loop statement and transfers execution to the statement immediately following the loop.
fallthrough	The fallthrough statement simulates the behavior of swift switch to C-style switch.

8. Swift Functions

A function is a set of statements organized together to perform a specific task. Swift functions contain parameter type and its return types.

Syntax:

```
func funcname(Parameters) -> returntype
{
Statement1
Statement2
---
Statement N
return parameters
}
```

1. **Functions Local Vs External Parameter Names**

Local parameter names are accessed inside the function alone whereas External parameter names allow us to name a function parameters to make their purpose more clear.

e.g

```
func pow(firstArg a: Int, secondArg b: Int) -> Int {
var res = a
for _ in 1..<b {
res = res * a
}
print(res)
return res
}
pow(firstArg:5, secondArg:3)
-------------------------[Results]---------------------------
125
```

2. **Variadic Parameters**

Parameters can be specified as variadic by (...) after the parameter name.

```
func NameofFunction(members: N...){
        for i in members {
            print(i)
            }
        }
        vari(4,3,5)
```

3. **Constant, Variable and I/O Parameters**

Functions by default consider the parameters as 'constant' where as the user can declare the arguments to the functions as variables also.

I/O parameters in Swift provide functionality to retain the parameter values even though its values are modified after the function call. At the beginning of function parameter definition 'inout' keyword is declared to retain the member values.

It derives the keyword 'inout' since its values are passed 'in' to the function and its values are accessed and modified by its function body and it is returned back 'out' of the function to modify the original argument.

Variables are only passed as an argument for in-out parameter since its values alone are modified inside and outside the function. Hence no need to declare strings and literals as in-out parameters. '&' before a variable name refers that we are passing the argument to the in-out parameter.

e.g

```
func temp(a1: inout Int, b1: inout Int) {
let t = a1
a1 = b1
b1 = t
}
var no = 2
var co = 10
temp(&no, &co)
print("Swapped values are \(no), \(co)")
-------------------------[Results]---------------------------

Swapped values are 10, 2
```

4. **Function Types & Its Usage**

Using Function Types

Functions are first passed with integer, float or string type arguments and then it is passed as constants or variables to the function as mentioned below.

```
var addition: (Int, Int) -> Int = sum
```

Here sum is a function name having 'a' and 'b' integer variables which is now declared as a variable to the function name addition. Hereafter both addition and sum function both have same number of arguments declared as integer datatype and also return integer values as references.

```
func sum(a: Int, b: Int) -> Int {
return a + b
}
var addition: (Int, Int) -> Int = sum
print("Result: \(addition(40, 89))")
-------------------------[Results]---------------------------

Result: 129
```

5. **Function Types as Parameter Types & Return Types**

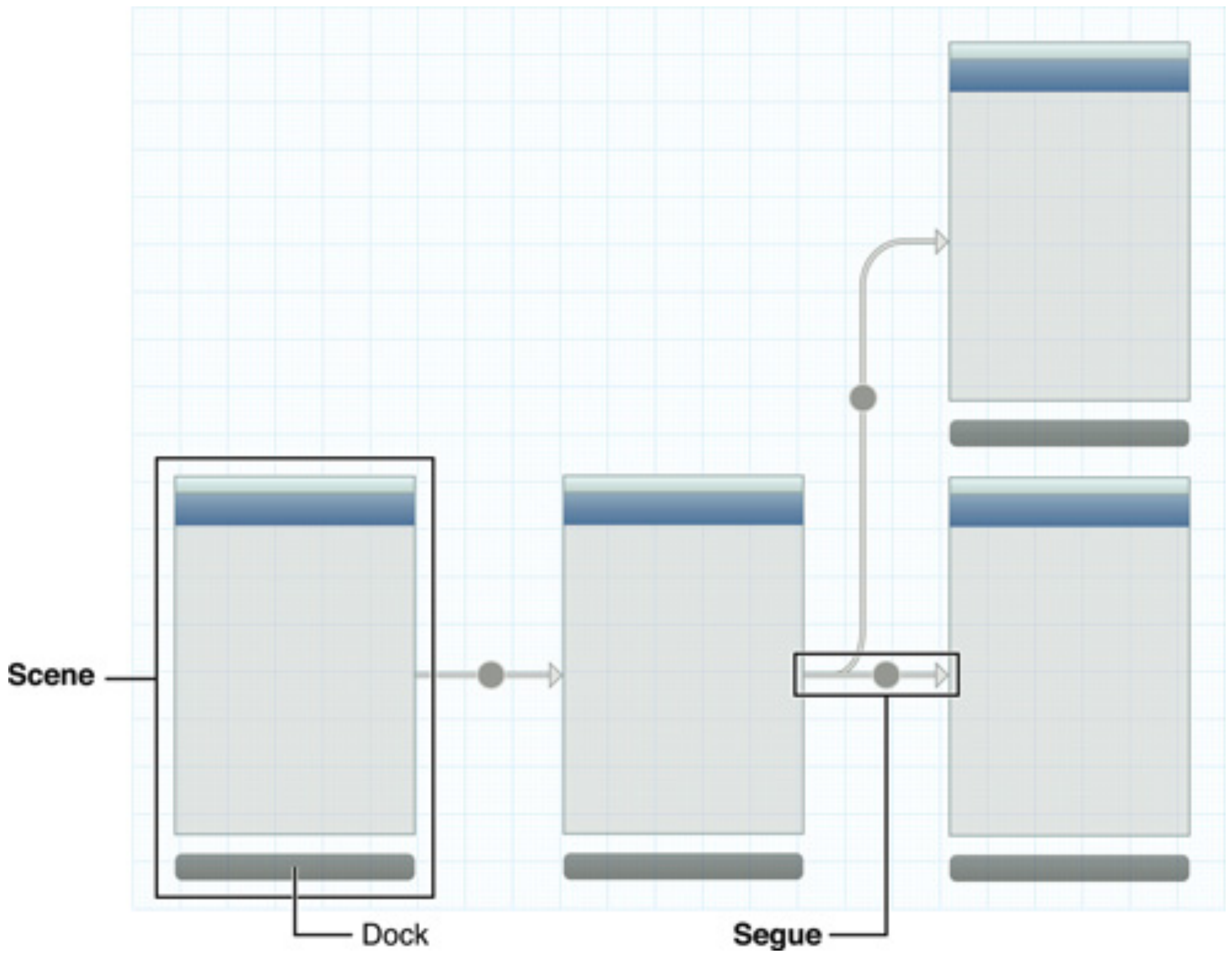
We can also pass the function itself as parameter types to another function.

```
func sum(a: Int, b: Int) -> Int {
        return a + b
}
var addition: (Int, Int) -> Int = sum
print("Result: \(addition(40, 89))")
        func another(addition: (Int, Int) -> Int, a: Int, b: Int) {
        print("Result: \(addition(a, b))")
}
another(sum, 10, 20)
-------------------------[Results]---------------------------

Result: 129
Result: 30
```

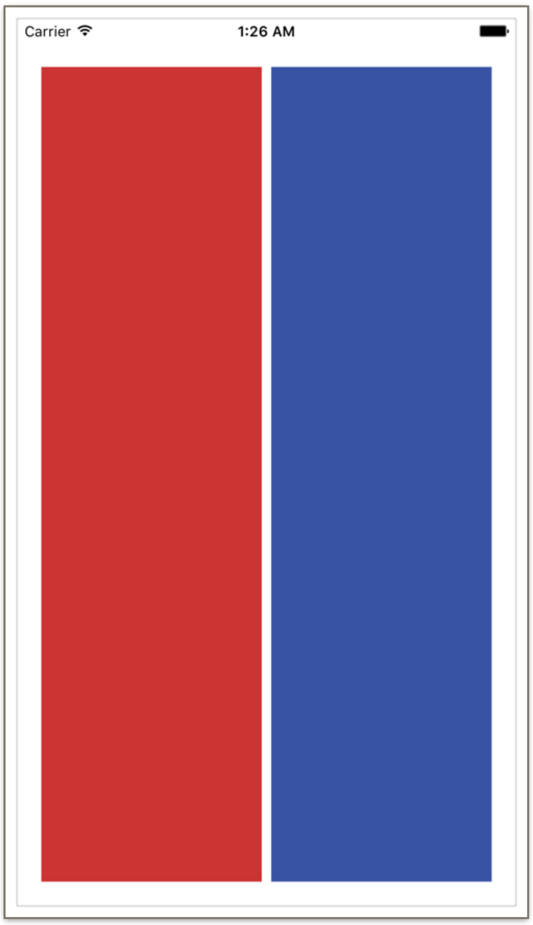
7.Exercise

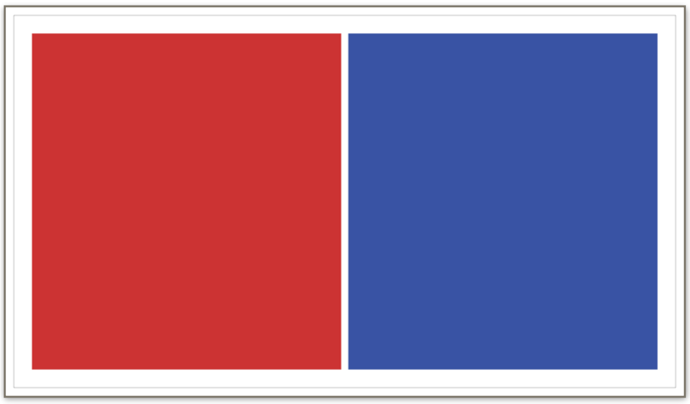
1. Create New iOS App Project using Single View Application Template and run it in iOS 10: iPhone 6, iPhone 6s, iPad Air 2 Simulator.
2. Create New iOS App Project using Master Detail View Application Template and run it as in exercise 1
3. Test the Xcode debugging features with Break point and using print function.
4. Create new iOS App using Single View Application Template and add all basic UI Elements stated in Article 1.5.2
5. Create new iOS App layout as shown.



2. Create new iOS App using Single View Application Template and add Two more Storyboards and then create transition between each view controller with segue. (Hint: use button to trigger segue to jump to another screen)

3. Create and set the following two views' constrain to make looking good in portrait and landscape as shown





4. Create a new playground file and define variables named a,b,c and d. Assign the following numbers 10, 20.0, true, "Swift", respectively at the same time.Observe the type of variable a,b c and d. Then define the required type by annotation, explicitly.

5. 10. Perform the following mathematical operation on above variables and store the result in constant name r1, r2, r3,r4,r5 and r6 (All Constant Types required to be Double Type)

a + b
a * b
a / b
a % b

6. Output above all operation as a string as shown in example (Use both methods: string interpolation and mixing variable types)

eg. 10 + 20.0 is 30.0

7.Increase variable a and b by one using += operator and using successor( ) or advanceBy( ) methods

8.Decrease variable a and b by one using -= operator and using predecessor( ) or advanceBy( ) methods

9..Observe the maximum and minimum allowable value to store in variable a by using Int.max , Int.min.

10.Type the Following Arrays in Swift Playground File. Using **For in Loop** , print all the array elements of arrayA & arrayB

```
var arrayA = [ 10 , 20 , 30 , 40 , 50 ]
var arrayB = ["Apple", "Google","LG"]
var arrayC = [String]( )
```

11.Perform the same operation on the arrayA in exercise 15 using **while loop**

12.Perform the following operation on the arrays in exercise 15 using for in loop, while

- Increment all elements of arrayA by 1

(note: indexed item is assigned by let automatically in for in loop)

- Store elements of arrayA to arrayC as a string element
- Remove the third element of arrayA, observe arrayA again
- Remove the last element of arrayB observe arrayB again
- Print the number of elements of arrayC

13.Type the Following Dictionary in Swift Playground File. Using **For in Loop** , print all the key and value pairs   of dictA & dictB

```
var dictA = [ "YGN":"Yangon","MDY":"Mandalay","MGY":"Magway"]
var dictB = [1:"Apple",2:"Bird",3:"Cat"]
```

and perform the following operation to dictA

- remove the key-valued pair MDY
- insert new key-value

14.Create two new arrays (one for keys and another for values) by converting dictionary dictA as in ex.18 using .keys and .values properties of dictionary

15.Create four functions computing given two parameters ,type double, to perform the following operation and return as double type

- Addition, subtraction , multiplication, dividen

16.Create one function computing given three parameters as typed Double, Double and String to perform the following operation and return as double type

- Addition, subtraction , multiplication, division

Third parameter is determine whether it is 'add', 'subtract' , 'multiply' or 'division' to perform the respective operation

17.Modify the function as in ex.21 to perform addition operation only if third parameter is omitted using **function overloading**

18.Modify the function as in ex.21 to perform addition operation only if third parameter is omitted using **default parameter on third parameter**

19.Modify the function as in ex.21 to perform addition operation only if third parameter is omitted using **default parameter on third parameter but return the all four operation using Tuple.**

20.Create a function with a parameter , type array of String , printing out all elements

21.Create a function with a variadic parameter , printing out all elements when parameter of any type of array is passing by.

22.Create the Following three Functions

- add (a: Double, b:Double)
- subtract (a: Double, b:Double)
- multiply (a: Double, b:Double)

and create the fourth function that taking function as a parameter to pass by any of above function.