Table of Content

I. iOS Development Fundamental

1. 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.

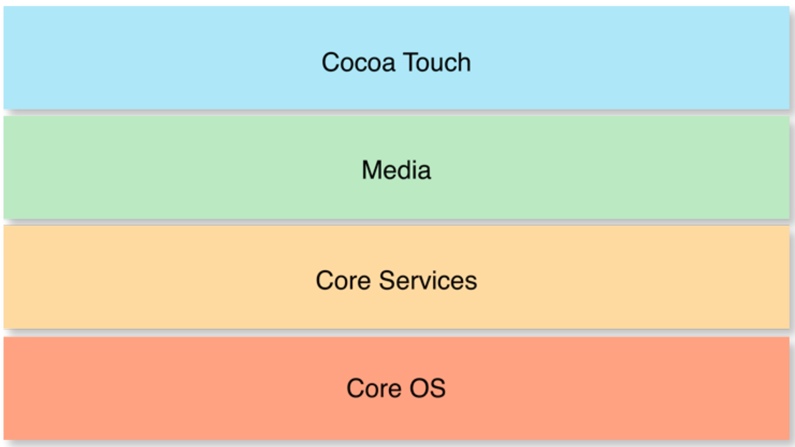
2.iOS : Operation System Apple Mobile Platform

- (See also notes on keynote)

1. iOS App Architecture

At the highest level, iOS acts as an intermediary between the underlying hardware and the apps you create. Apps do not talk to the underlying hardware directly. Instead, they communicate with the hardware through a set of well-defined system interfaces.

The implementation of iOS technologies can be viewed as a set of layers, which are shown in Figure I-1. Lower layers contain fundamental services and technologies. Higher-level layers build upon the lower layers and provide more sophisticated services and technologies.



(Fig I-1)

2. Framework

The iOS technologies are packaged as frameworks.Apple delivers most of its system interfaces in special packages called frameworks. A framework is a directory that contains a dynamic shared library and the resources (such as header files, images, and helper apps) needed to support that library. To use frameworks, you add them to your app project from Xcode. eg.Foundation.framework, UIKit.framework, MapKit.framework

3. App Store and Distribution Channel

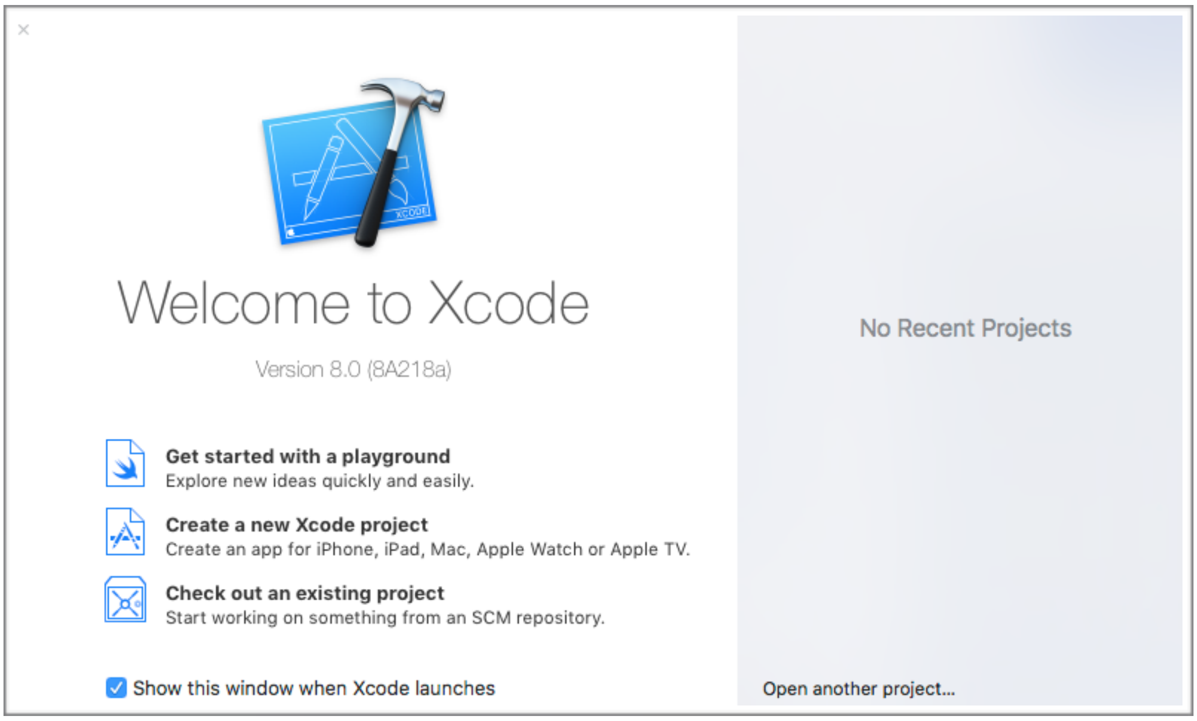
The App Store is a digital distribution platform for mobile apps on iOS.The apps can be downloaded directly to iOS devices or onto a personal computer via iTunes and offers an update service, e.g. if one application is updated, it will notify to update to existing users and allow to install.

Enterprise Apps are in-house iOS apps developed for a certain organization and intended to use within organization

3. Using Xcode IDE

1. Xcode Functionality

Xcode IDE, center of the Apple development, is an productive environment for building apps for Mac, iPhone, iPad, Apple Watch, and Apple TV. see Figure 1.2



Welcome to Xcode

Version 8.0 (8A218a)

Get started with a playground
Explore new ideas quickly and easily.

Create a new Xcode project
Create an app for iPhone, iPad, Mac, Apple Watch or Apple TV.

Check out an existing project
Start working on something from an SCM repository.

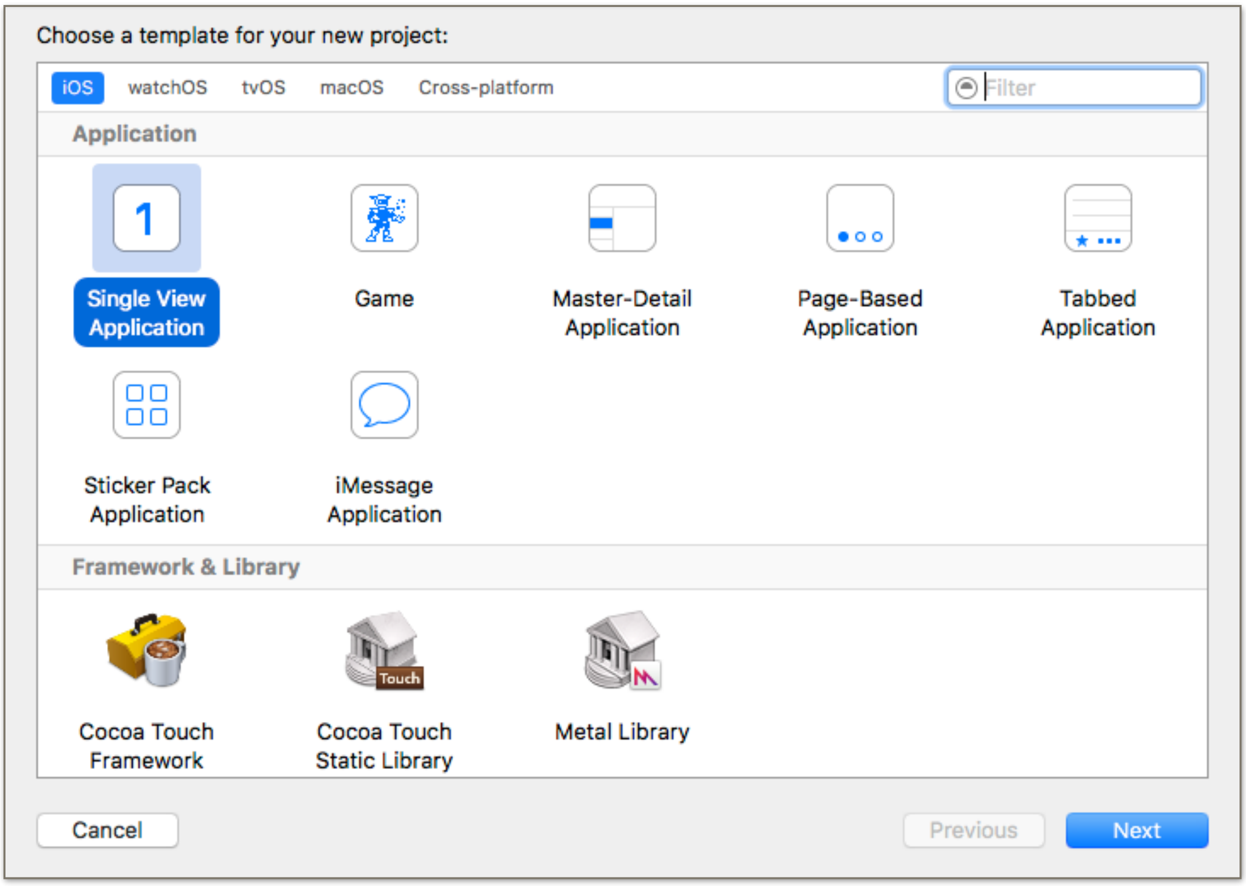
✓ Show this window when Xcode launches

No Recent Projects

Open another project...

(Fig I-2)

It has a text editor where you'll type in your source code and it has a visual editor for designing your app's user interface and Xcode transforms the source code that you write into an executable app and launches it in the Simulator or on your iPhone. Because no app is bug-free, Xcode also has a debugger that helps you find defects in your code.



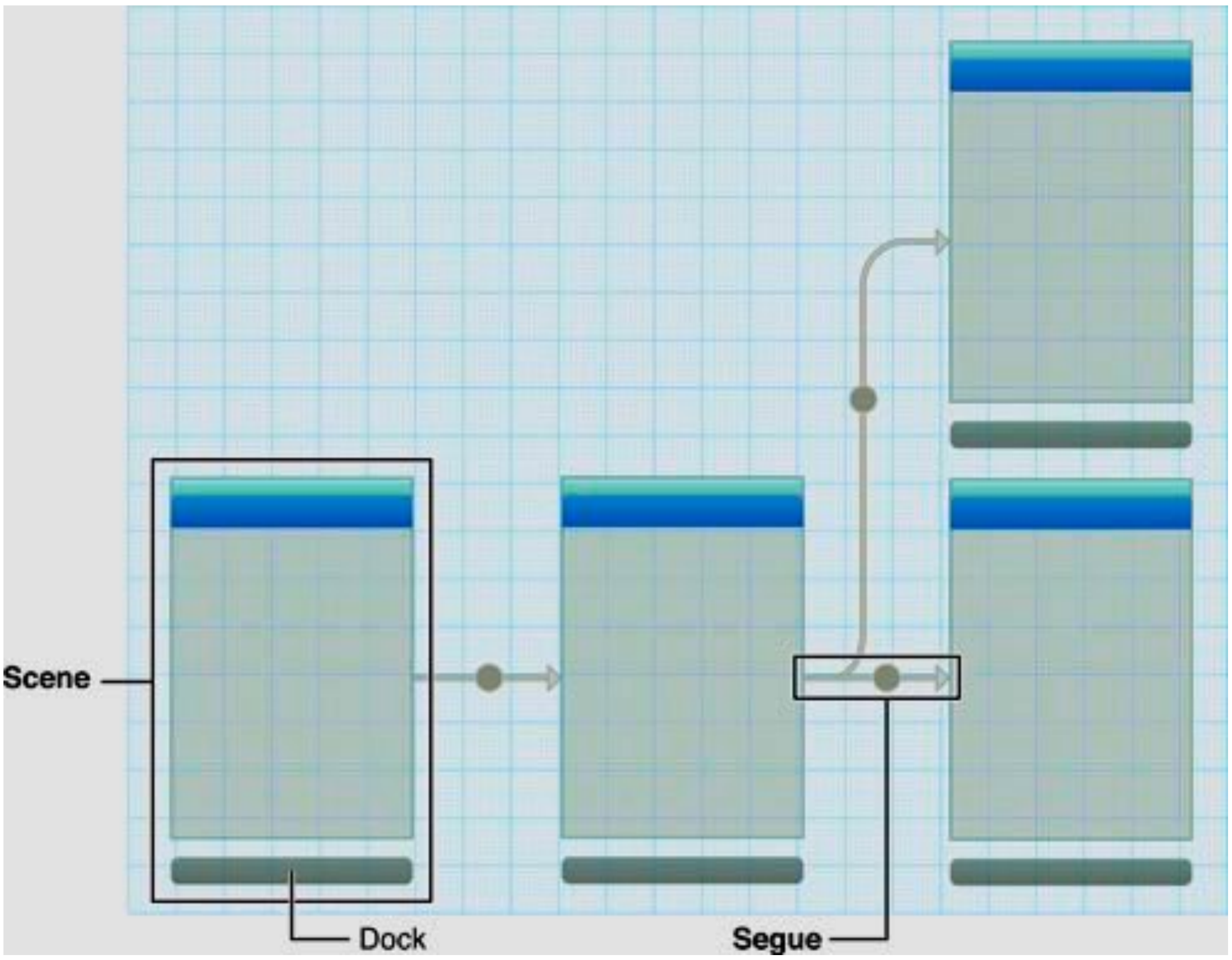
(Fig I-3) Xcode Default Templates

2. **Running Simulator**
   - **(See Demo)**
3. **Default Templates**
   - **(See Demo)**

4. Using Storyboard

A storyboard is a visual representation of the user interface of an iOS application, showing screens of content and the connections between those screens. A storyboard is composed of a sequence of scenes, each of which represents a view controller and its views; scenes are connected by segue objects, which represent a transition between two view controllers. (Fig 1-4)

Xcode provides a visual editor for storyboards, where you can lay out and design the user interface of your application by adding views such as buttons, table views, and text views onto scenes. In addition, a storyboard enables you to connect a view to its controller object, and to manage the transfer of data between view controllers. Using storyboards enable you to visualize the appearance and flow of your user interface on one canvas.



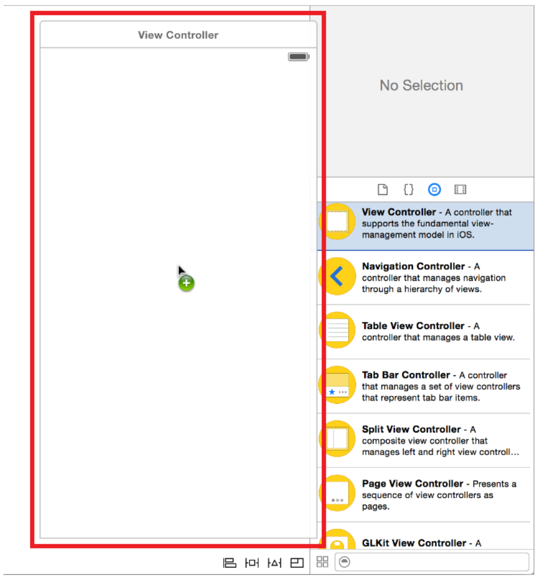
(Fig 1-4)

Note: A Scene Corresponds to a Single View Controller and Its Views .A Segue Manages the Transition Between Two Scenes.

1. **Creating Scene in Storyboard**

   Creating Scenes in storyboard is actually creating a visual representation of a scene in which view controller and its view are defined as connected. Type "UIViewController" in search box of utility windows and drag the View Controller Visual Component to the Main.Storyboard (Fig1-5)

(Fig 1-5)



View Controller - A controller that supports the fundamental view-management model in iOS.

Navigation Controller - A controller that manages navigation through a hierarchy of views.

Table View Controller - A controller that manages a table view.

Tab Bar Controller - A controller that manages a set of view controllers that represent tab bar items.

Split View Controller - A composite view controller that manages left and right view controll...

Page View Controller - Presents a sequence of view controllers as pages.

GLKit View Controller - A

2. Storyboard In Action

Using segue, storyboard can be illustrated from which view controller is connected to which and how program will work.To connect one view controller to another view controller, press and hold control key and link from one view controller to another view controller using segue.Name the segue identifier as you desired.

- (See Demo)

5. iOS Basic UI Elements

1.Containers: Windows & View

A window handles the overall presentation of your app's user interface. Windows work with views and a window is just a blank container for one or more views to sit in.windows do not have title bars, close boxes, or any other visual adornments in iOS. Users don't see, close, or move the window of an iOS app. And instead of opening another window to display new content and instead an iOS app changes the views inside its window.It has several responsibilities:

- It contains your application's visible content.
- It plays a key role in the delivery of touch events to your views and other application objects.
- It works with your application's view controllers to facilitate orientation changes.

When you base a new iOS app project on one of the Xcode templates and use storyboards to design your user interface, you don't have to explicitly create, configure, or load your app's window.

When you create a main storyboard file for your app—and identify it as the main storyboard in your information property list file—iOS performs several setup tasks for you. Specifically, at launch time iOS:

- Instantiates a window.
- Loads the main storyboard and instantiates its initial view controller.
- Assigns the new view controller to the window's rootViewController property and then makes the window visible.

Before the initial view controller is displayed, your app delegate is called to give you a chance to configure the view controller.

Views can include others user interface elements and responsible to deliver touch event to its containers and others related tasks. and its can be identify by tag for futures reference.

- (See Demo)

2. Using Basic UI Elements

Note:type and filter in Utility Pane's search box and observe following each user interface elements.

1. Label

A label displays static text and Displays any amount of static text
Doesn't allow user interaction except, potentially, to copy the text
Use a label to name or describe parts of your UI or to provide short messages to the user. A label is best suited for displaying a relatively small amount of text.

2. Buttons (System Button)

A system button performs an app-specific action.

3. Switch

A switch presents two mutually exclusive choices or states.
and it indicates the binary state of an item and it is used specifically in table views only

4. Stepper

A stepper increases or decreases a value by a constant amount.
and supports custom images.But use a stepper when users might need to make small adjustments to a value and avoid using a stepper when users are likely to make large changes to a value.

5. Segmented control

A segmented control is a linear set of segments, each of which functions as a button that can display a different view.Consists of two or more segments whose widths are proportional, based on the total number of segments and can display text or images
Use a segmented control to offer choices that are closely related

6. Slider Control

A slider allows users to make adjustments to a value or process throughout a range of allowed values. Use a slider to give users fine-grained control over values they can choose or over the operation of the current process.

7. Progress View

A progress view shows the progress of a task or process that has a known duration. Use a progress view to give feedback on a task that has a well-defined duration, especially when it’s important to indicate approximately how long the task will take.

8. Text Field

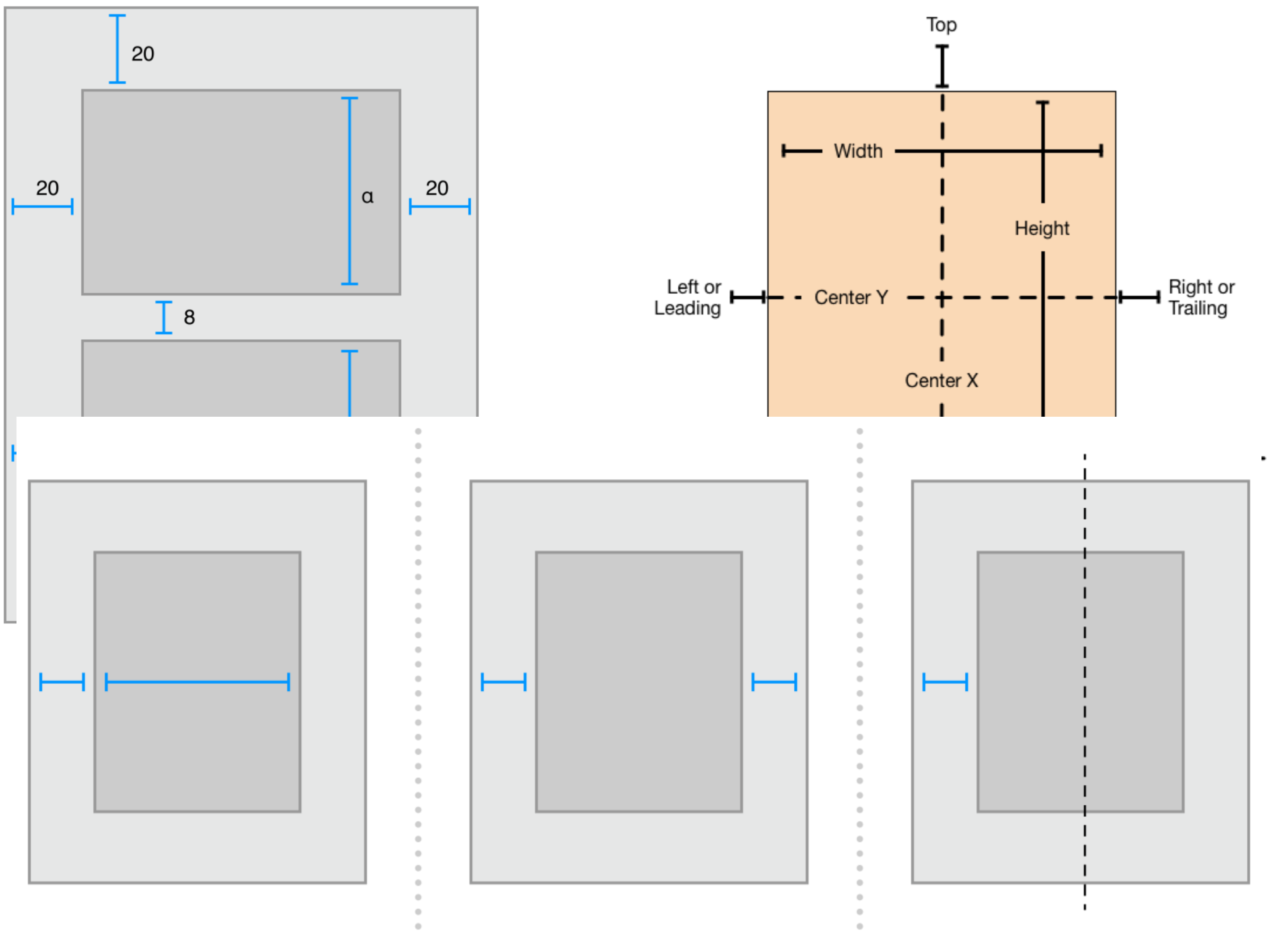
A text field accepts a single line of user input

(Refer: Basic UI Elements' Properties & Action Table)

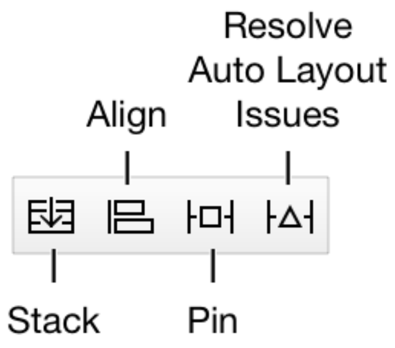
3. Autolayout & Constrain

Auto Layout dynamically calculates the size and position of all the views in your view hierarchy, based on constraints placed on those views.
This constraint-based approach to design allows you to build user interfaces that dynamically respond to both internal and external changes.

Auto Layout defines your user interface using a series of constraints. Constraints typically represent a relationship between two views. Auto Layout then calculates the size and location of each view based on these constraints. This produces layouts that dynamically respond to both internal and external changes.



(Fig-1-6 A,B & C)



(Fig-1-7)

There are two basic types of attributes. Size and Location

Size attributes (for example, Height and Width) and location attributes (for example, Leading, Left, and Top). Size attributes are used to specify how large an item is, without any indication of its location. Location attributes are used to specify the location of an item relative to something else. However, they carry no indication of the item’s size.

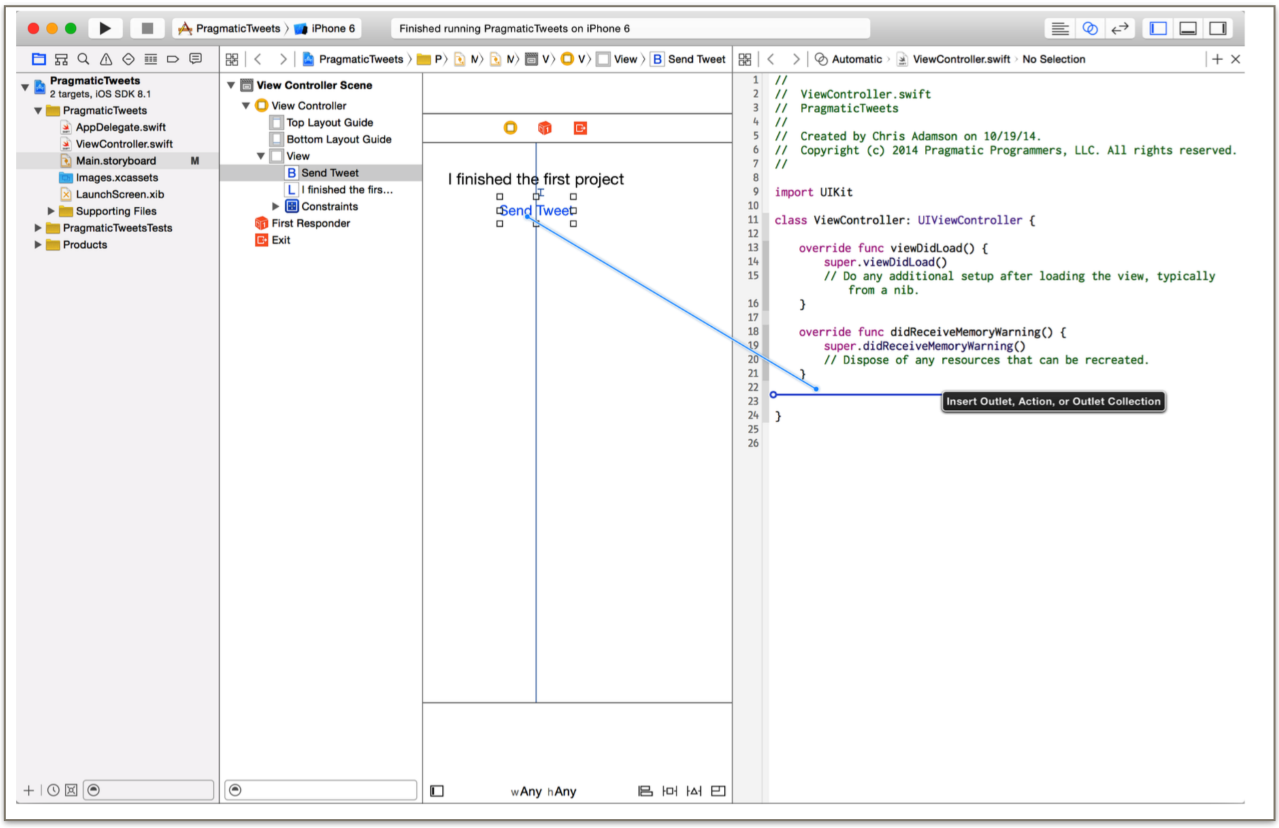
In general, the constraints must define both the size and the position of each view. satisfiable layout requires two constraints per view per dimension (Fig 1-6 C)

4. IBOut and IBAction

In iOS, we use Interface Builder connections to tie the user interface to our code. Using Xcode, we can create two kinds of connections:

•IBOutLet : An outlet connects a variable or property in code to an object in a story- board. This lets us read and write the object's properties, like reading the value of a slider or setting the initial contents of a text field.

•IBAction: An action connects an event generated by a storyboard object to a method in our code. This lets us respond to a button being tapped or a slider's value changing.
(See Fig-1-8)



```
//
//  ViewController.swift
//  PragmaticTweets
//
//  Created by Chris Adamson on 10/19/14.
//  Copyright (c) 2014 Pragmatic Programmers, LLC. All rights reserved.
//

import UIKit

class ViewController: UIViewController {

    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view, typically
           from a nib.
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be recreated.
    }


}
```

(Fig 1-8)

Useful Properties and Method to be used in **IBOutlet** & **IBAction** Connection

UI Types	Properties (Get/Set)	Method	Remark
Label	`text`		
	`textColor`		
	`backgroundColor`	x	
	`textAlignment`		
Button	`titleLabel`	TouchUpInside()	
textField	`text`	ValueChange	
		EditingDidEnd	
		EditingChanged	
		textFieldShouldReturn	
		resignFirstResponder	
Switch	`on`	ValueChange	
Progress View	`progress`	x	
Segment Control	`selectedSegmentIndex`	ValueChange	

6. Swift Programming (Part I)

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)
}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

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:

```
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)

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

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]()

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.

Note:

nested if	You can use one if or else if statement inside another if or else if statement(s).

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.

switch statement

A switch statement allows a variable to be tested for equality against a list of values.

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 vari<N>(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

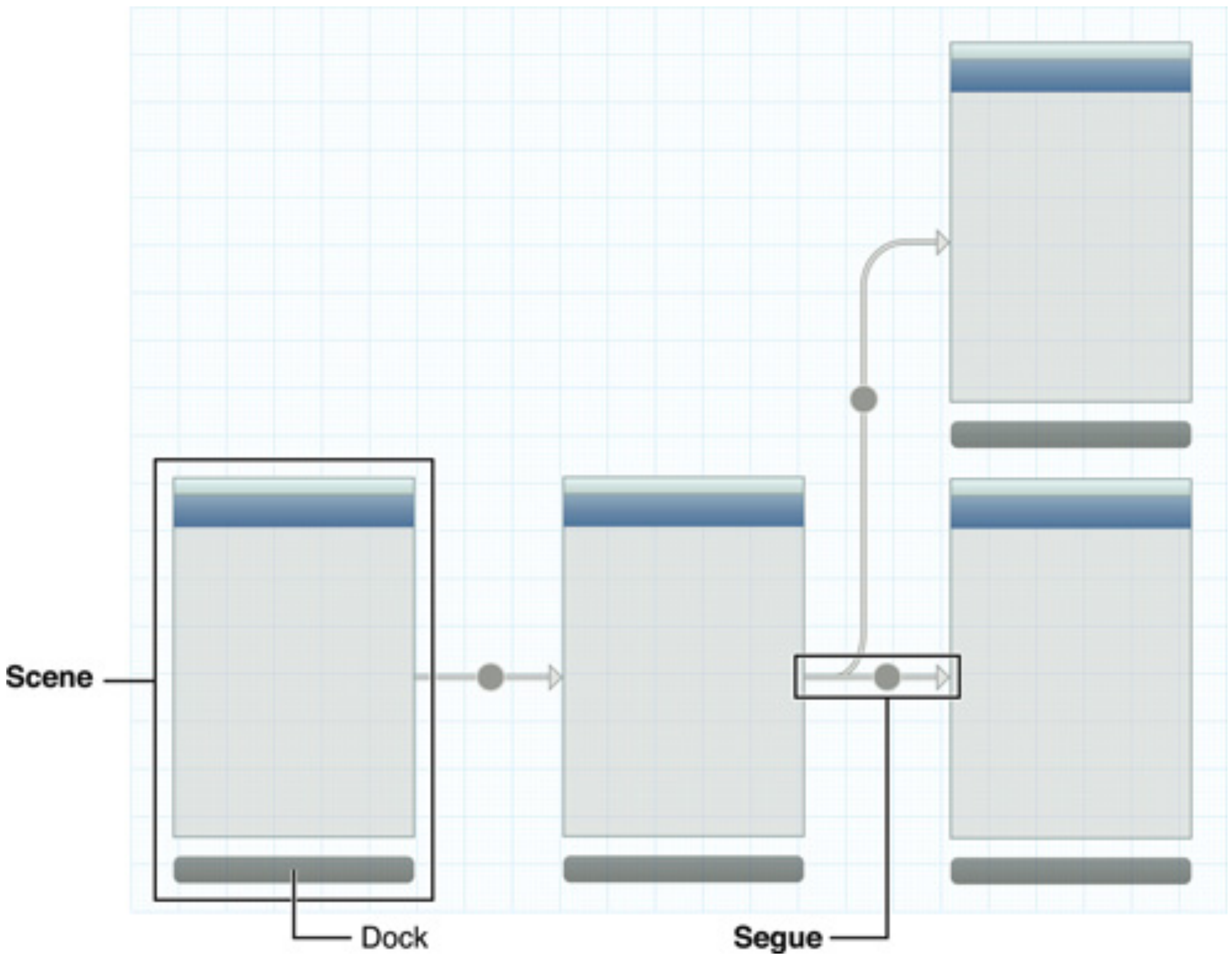
We can also pass the function itself as parameter types to another function.

```
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
```

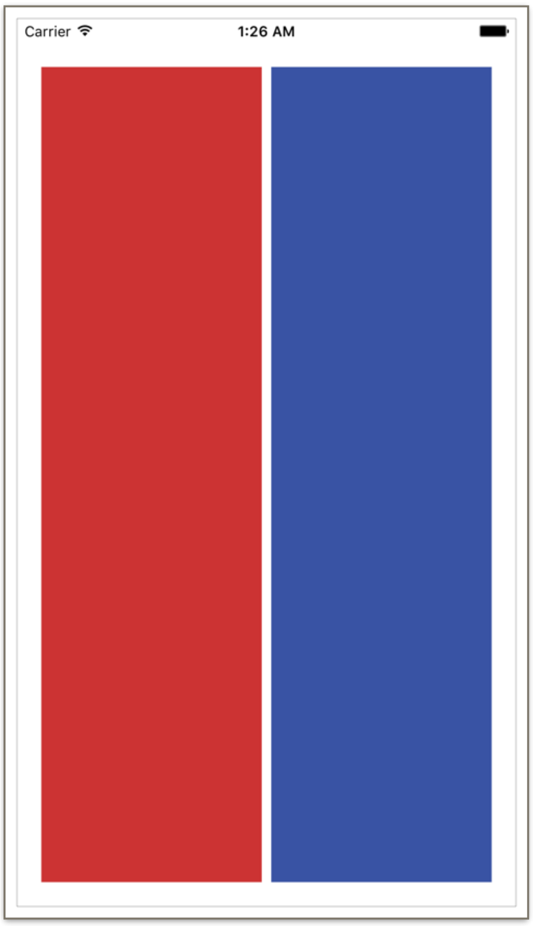
7.Exercise

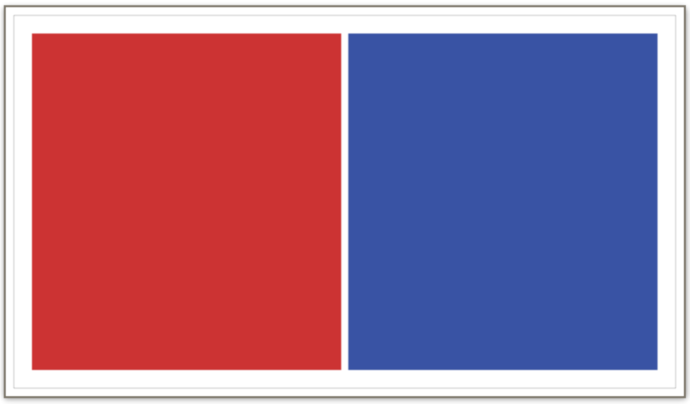
1. Create New iOS App Project using Single View Application Template and run it in iOS 10: iPhone 6, iPhone 6s, iPad Air 2 Simulator.
2. Create New iOS App Project using Master Detail View Application Template and run it as in exercise 1
3. Test the Xcode debugging features with Break point and using 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.



7. 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)

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





9. 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.

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

11.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

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

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

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

15.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]( )
```

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

17.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

18.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

19.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

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

- Addition, subtraction , multiplication, dividen

21.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

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

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

24.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.**

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

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

27.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.