Fundamentos de Desarrollo de Aplicaciones para Dispositivos Móviles con SWIFT

C O N T E N I D O

- Aspectos generales.
- Arquitectura.
- Entorno de desarrollo.
- FrameWorks.
- Estructuras, Protocolos y Clases.
- Tipos de variables.
- Flujos de control.
- Bucles.

ASPECTO GENERALES

Es un lenguaje intuitivo.

Es seguro, obliga a declarar las variables antes de usarlas y a definir su tipo, aunque estas pueden ser inferidas. Además realiza una revisión de sus reglas al momento de compilar.

Orientado a patrones de diseño.

Rápido de implementar, ya que está pensado para los dispositivos de su plataforma.

Gestión de memoria automática.

ASPECTO GENERALES

Control de errores avanzado.

Orientado al hardware, ya que esta creado para un buen rendimiento de sus dispositivos.

La sintaxis es concisa y expresiva.

Mecanismos para implementar el patrón de diseño MVC.

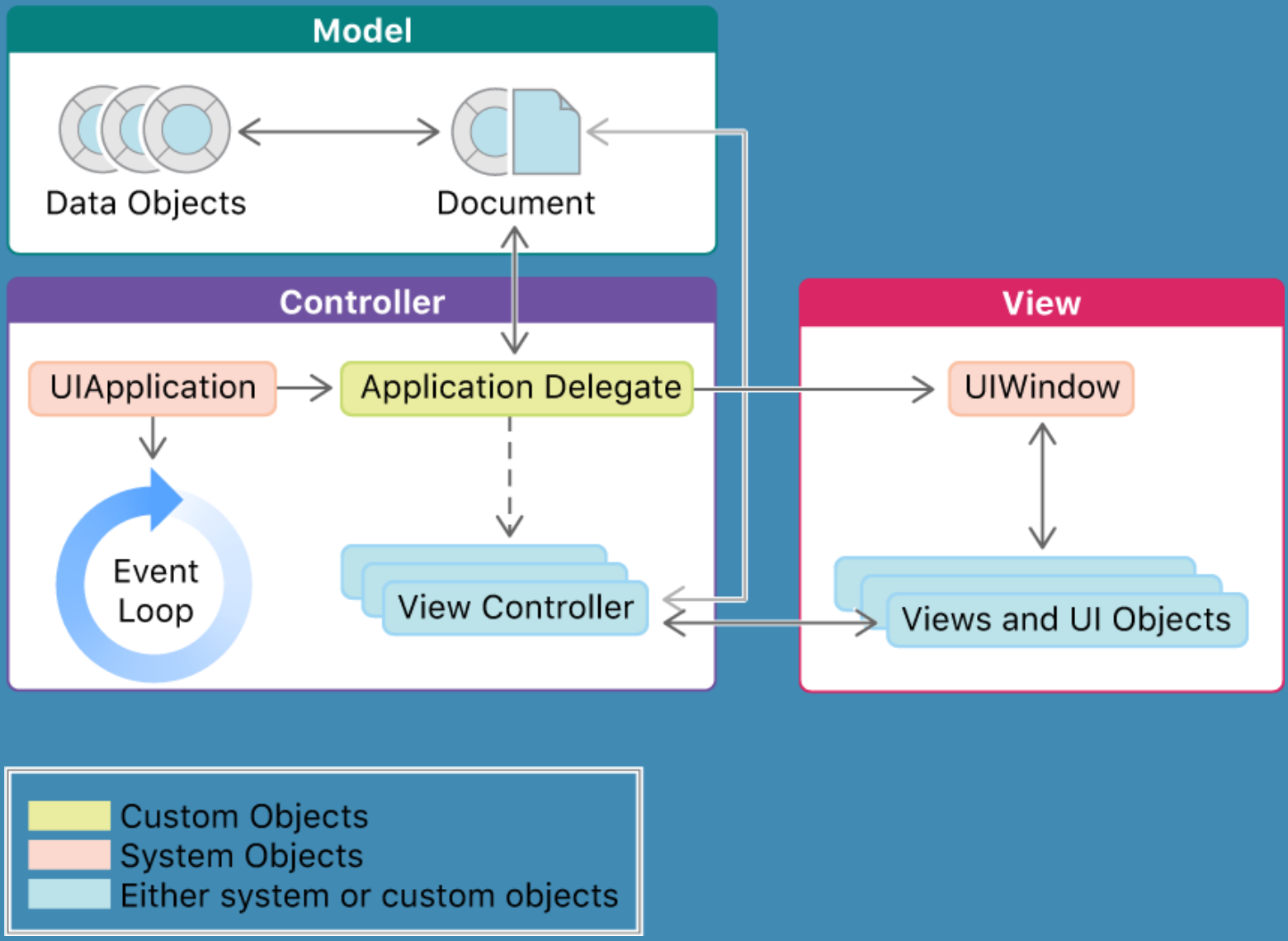
ARQUITECTURA DE LA APLICACIÓN

Aplica el Patrón de Diseño MVC, una solución reutilizable a problemas recurrentes que se encuentran en el diseño de software.

Facilita el trabajo del desarrollador al escribir aplicaciones para IOS.

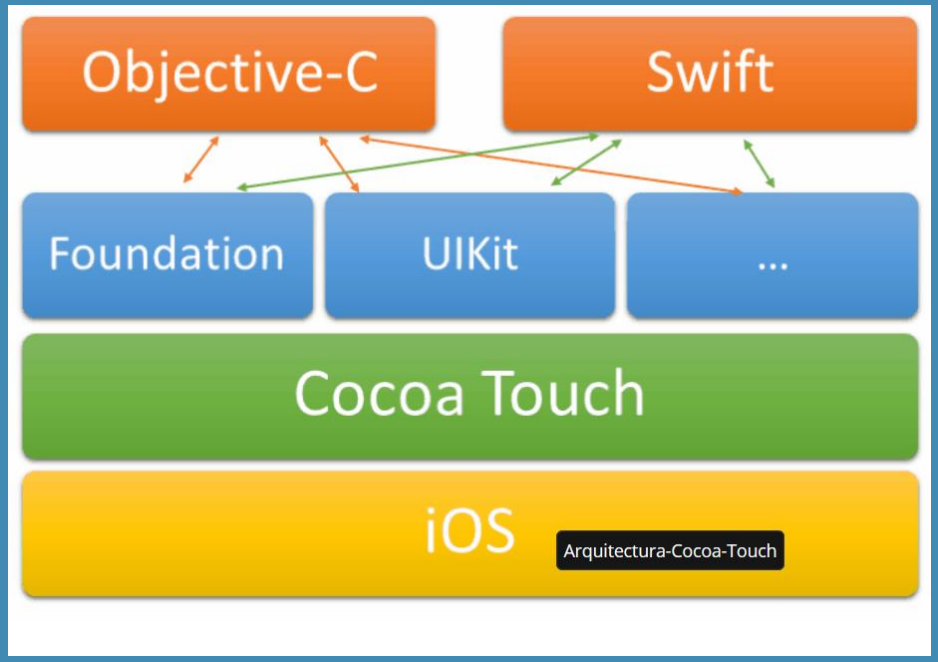
División de la aplicación en tres capas en las que cada una tiene una responsabilidad.

ARQUITECTURA DE LA APLICACIÓN



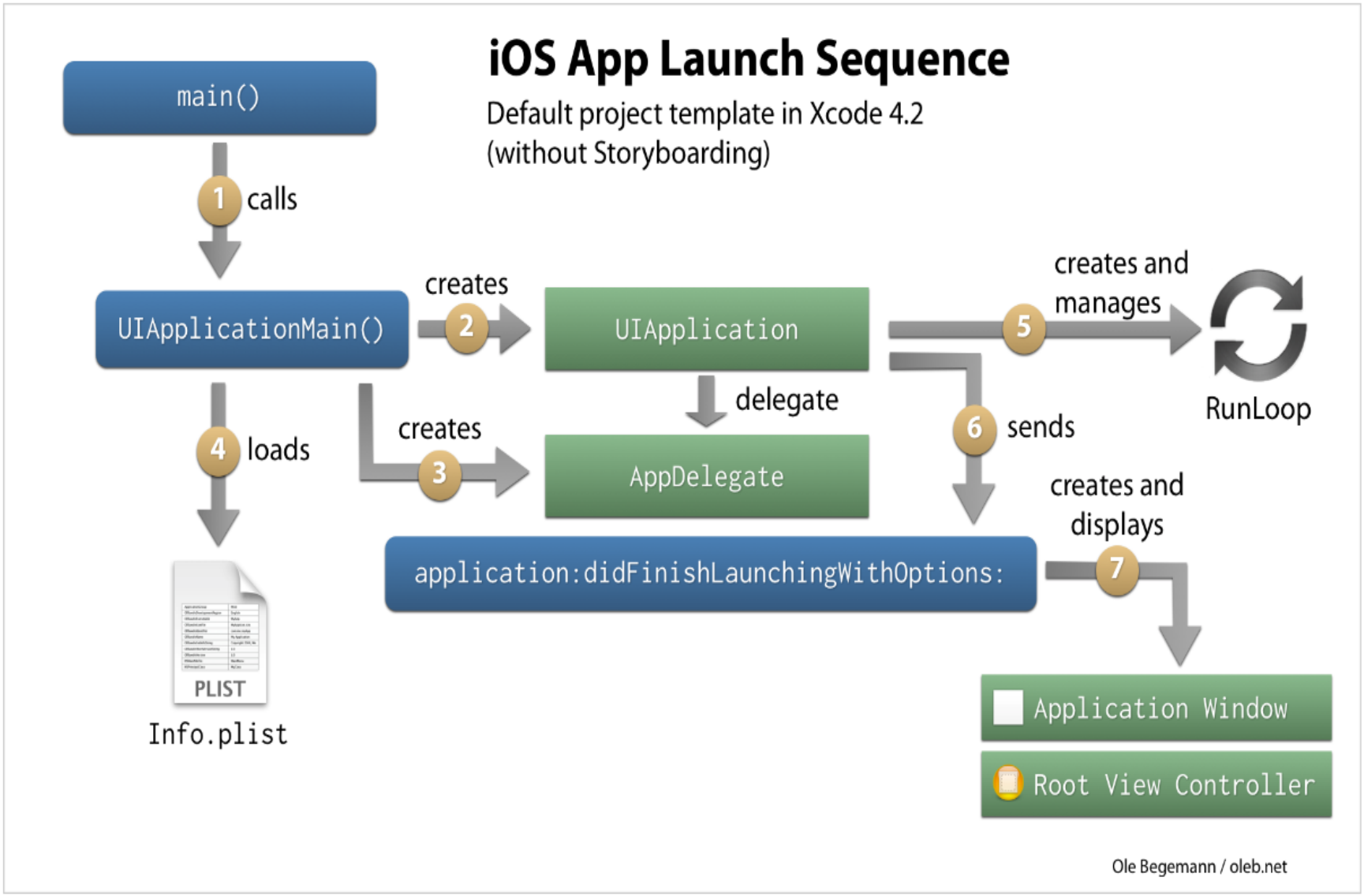
https://developer.apple.com/documentation/uikit/about_app_development_with_uikit

ARQUITECTURA DE LA APLICACIÓN



https://www.campusmvp.es/recursos/post/Objective-C-o-Swift-Que-lenguaje-aprender-para-programar-en-iOS.aspx

ARQUITECTURA DE LA APLICACIÓN



Flowchart of the default app launch sequence in iOS as of Xcode 4.2 for a non-storyboarded app. Feel free to share this image under a Creative Commons Attribution license (CC-BY). In a storyboarded app, `UIApplicationMain()` would additionally initiate the loading of

https://oleb.net/blog/2012/02/app-launch-sequence-ios-revisited/

ARQUITECTURA DE LA APLICACIÓN

La capa **Model** es en donde se almacenan los **datos** de la aplicación. Las clases que gestionen la **persistencia** de datos, los objetos Model. Las clases de comunicación de Cocoa Touch.

La capa Controller es la encargada de sincronizar la comunicación entre la capa View y la capa Model.

La capa View representa la interfaz con el usuario, despliega los datos de la capa modelo a través de la capa Controller.

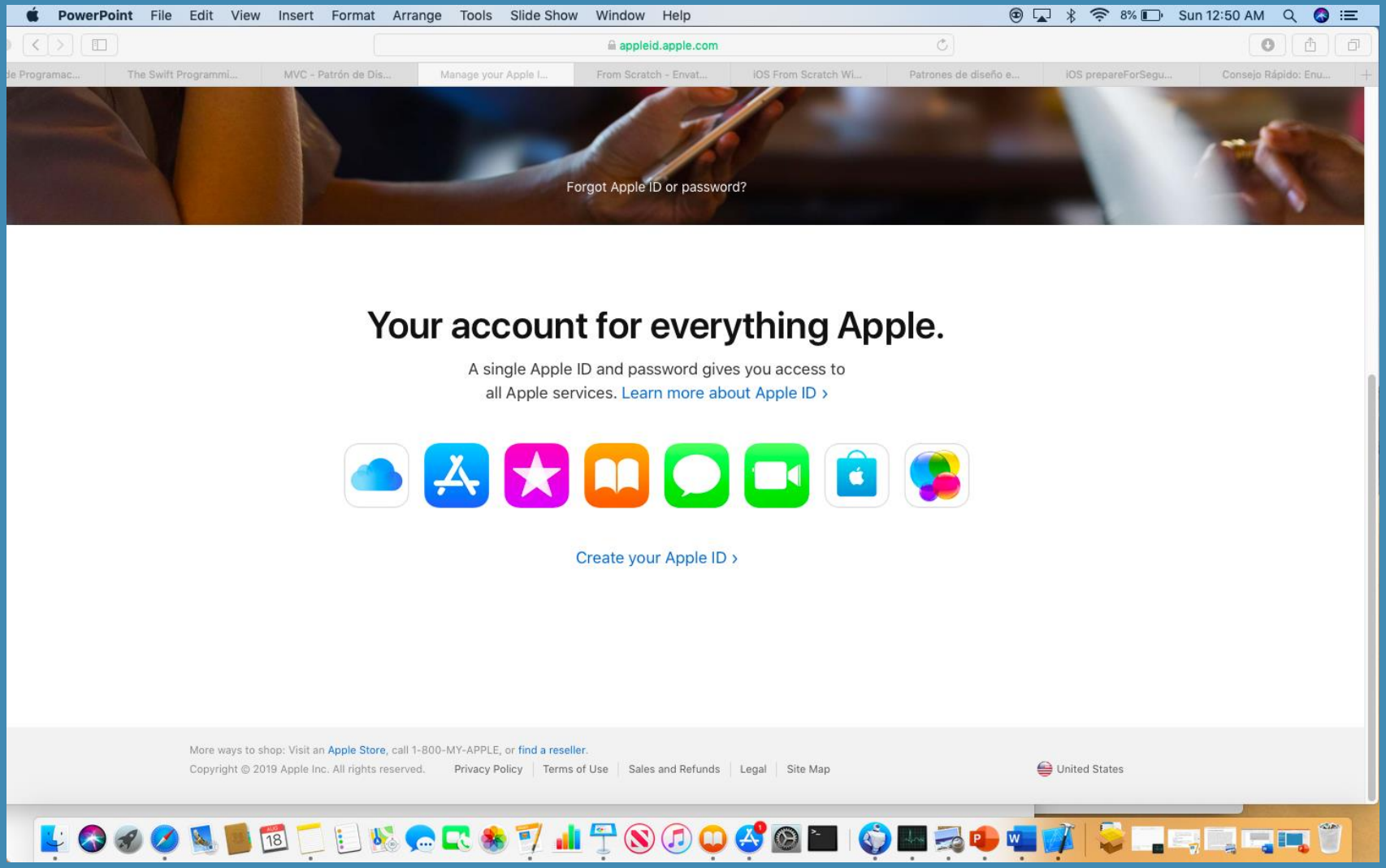
.

ENTORNO DE DESARROLLO

El entorno de Desarrollo básicamente está constituido por la herramienta XCODE, aunque para iniciar una aplicación, es necesario contar con ID de APPLE y si se require enviar aplicaciones a la tienda APP STORE es necesario inscribirse pagando una cuota anual.

ENTORNO DE DESARROLLO

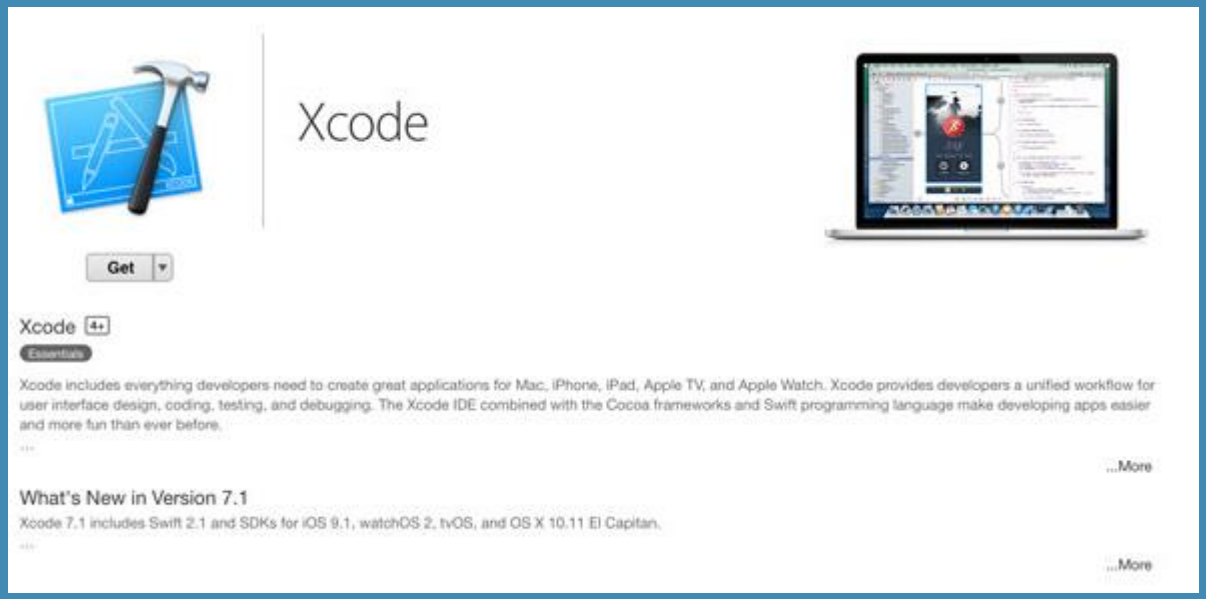
Creación de un APPLE ID



ENTORNO DE DESARROLLO

Descarga e Instalación

Descarga e instalación de la última version de XCODE. Se puede descargar Xcode de dos maneras: a través del sitio web de desarrolladores de Apple o a través de la App Store OS X (tu Mac).

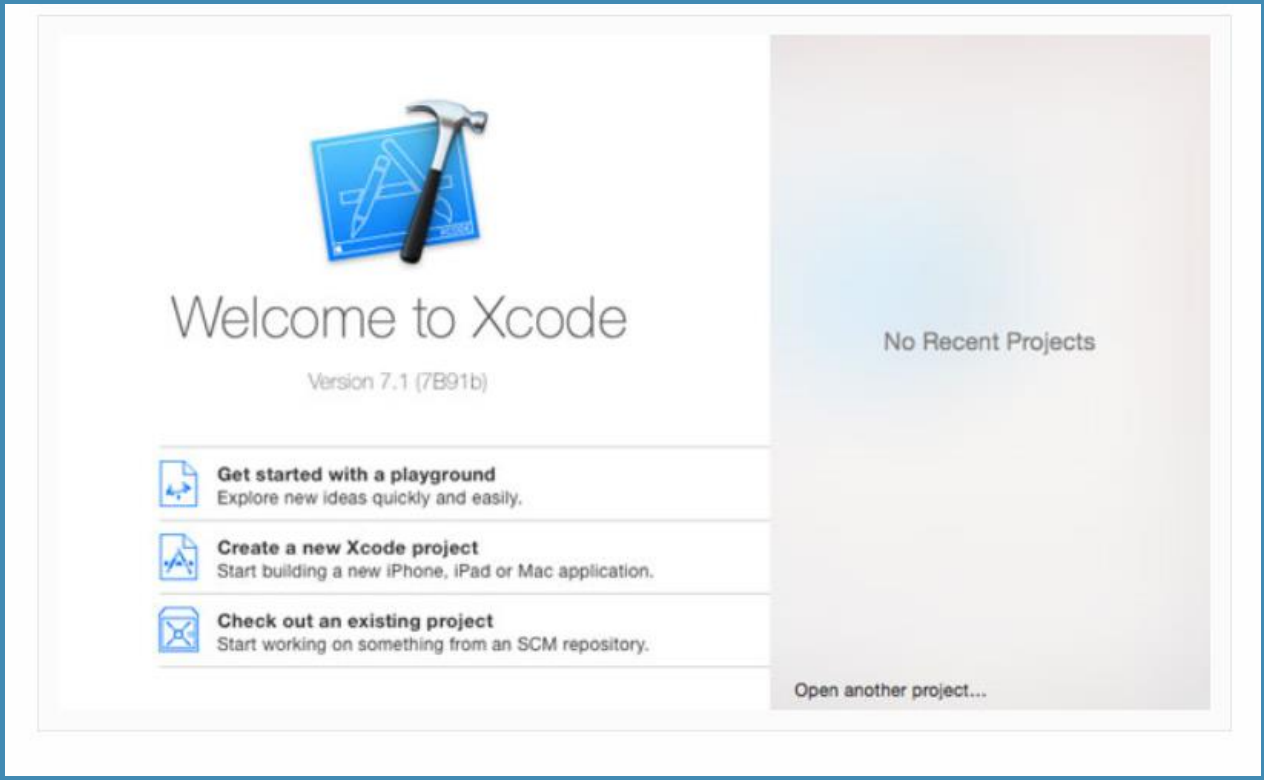


https://developer.apple.com/xcode/

ENTORNO DE DESARROLLO

Ejecución de XCODE

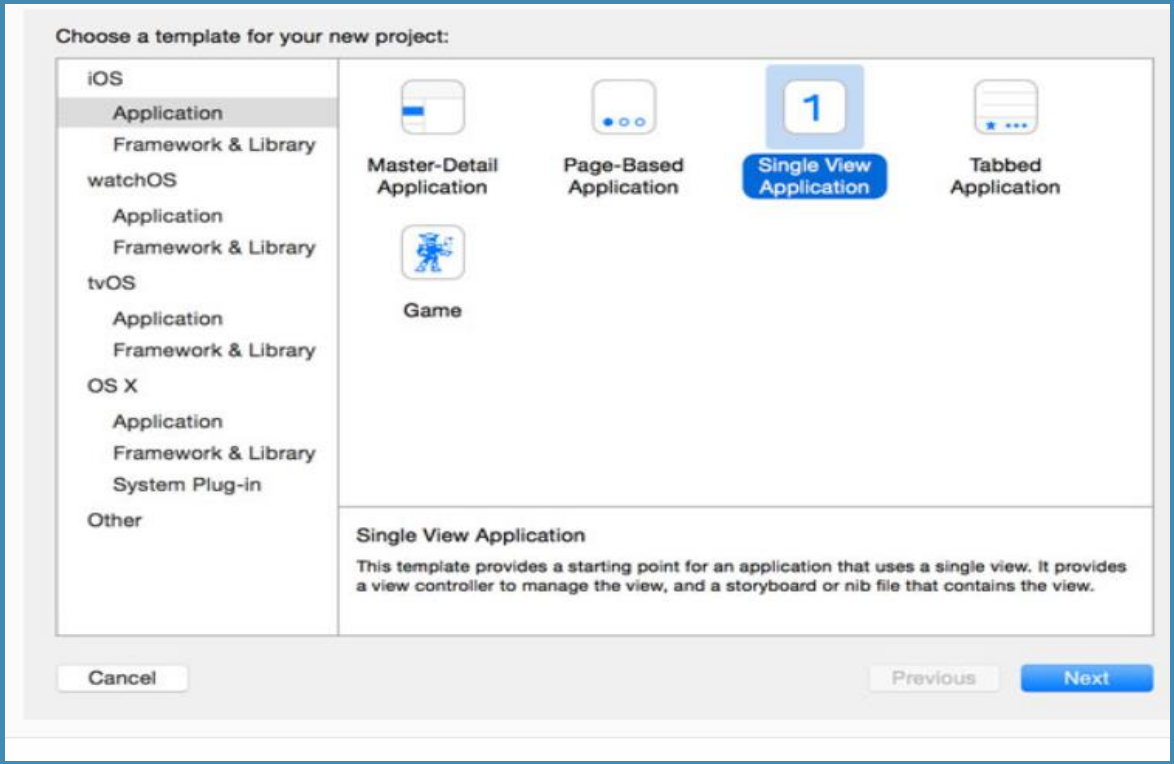
Ejecutar XCODE desde el icono de lanzamiento. Seleccionar alguna de las opciones. Un proyecto es un paquete que contiene todos los recursos para gestionar una aplicación.



ENTORNO DE DESARROLLO

Ejecución de XCODE

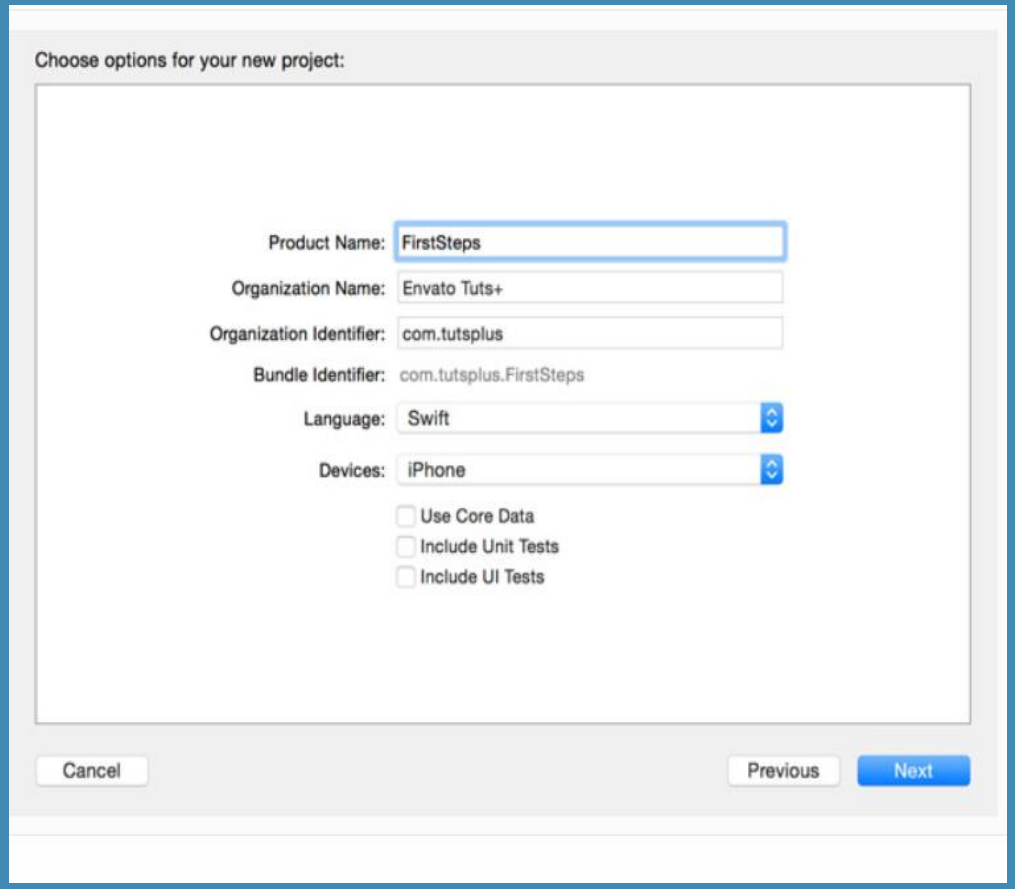
Existen varias opciones para iniciar una aplicación, aunque una fundamental es la plantilla de aplicación **Single View**. Seleccionar esa opción de la lista de **iOS > Application Templates** y oprimir clic en **Next**



ENTORNO DE DESARROLLO

Ejecución de XCODE

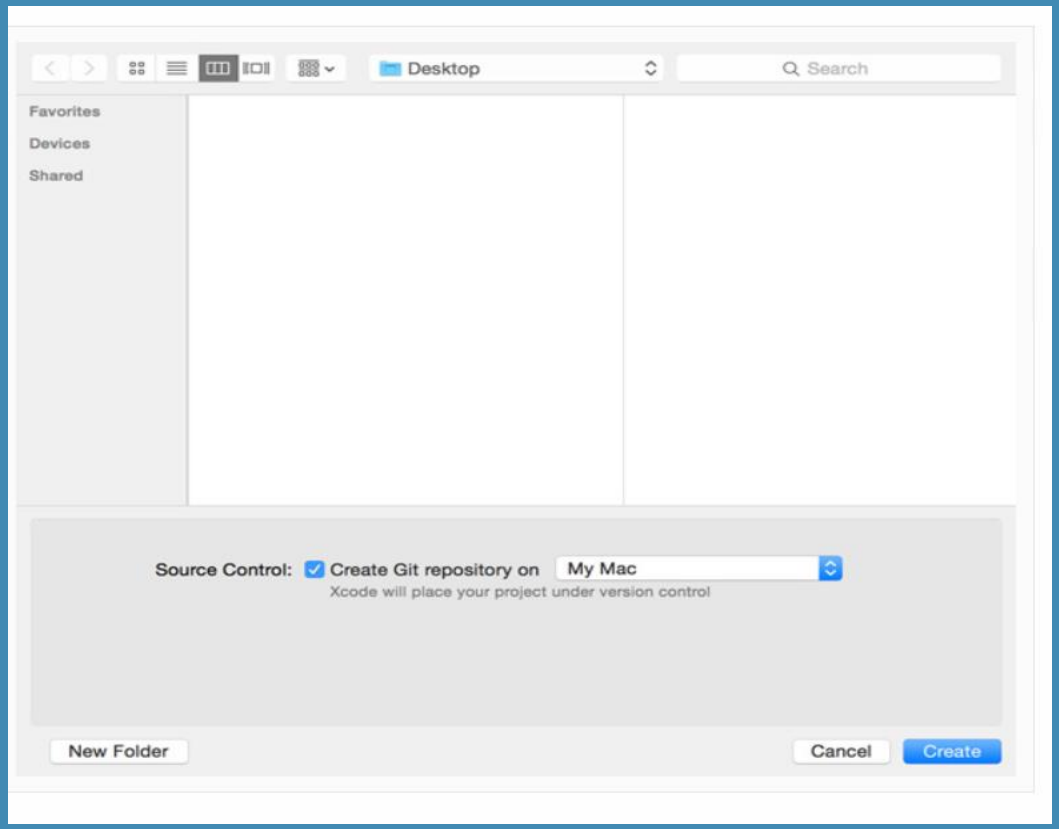
Después de oprimir el botón NEXT a continuación se deben llenar los datos de configuración del proyecto.



ENTORNO DE DESARROLLO

Ejecución de XCODE

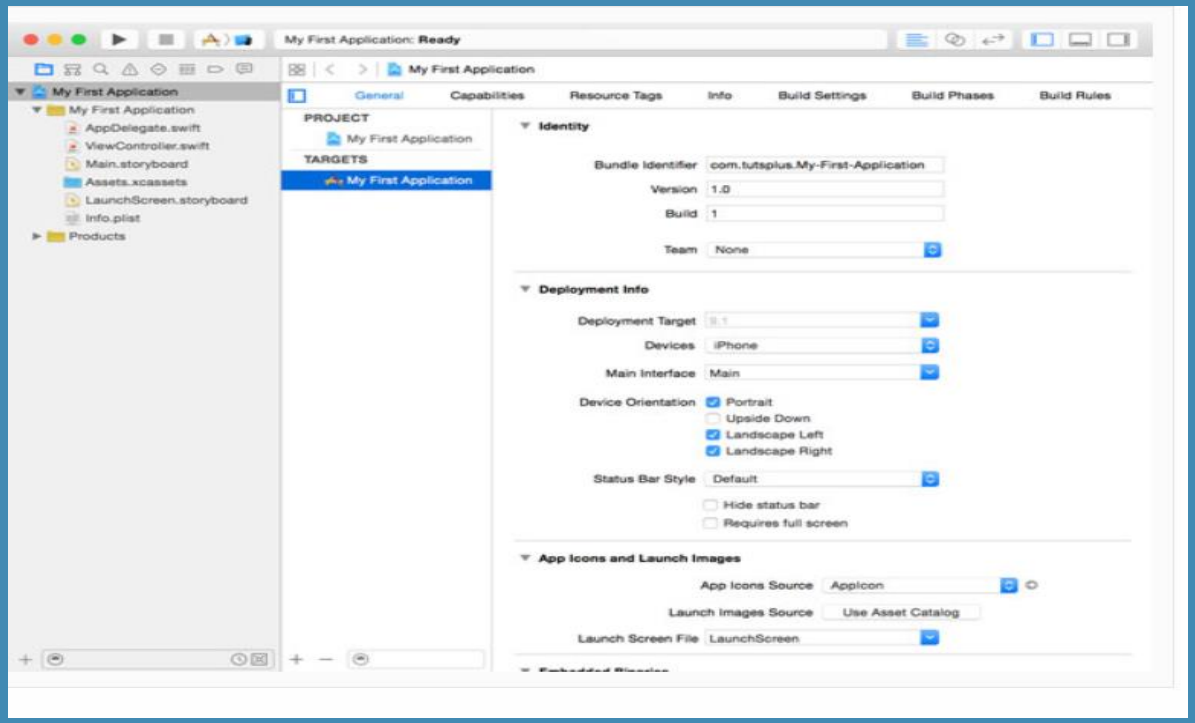
Finalmente XCODE solicita un lugar donde guardar el proyecto, incluyendo además la posibilidad de un repositorio local GIT.



ENTORNO DE DESARROLLO

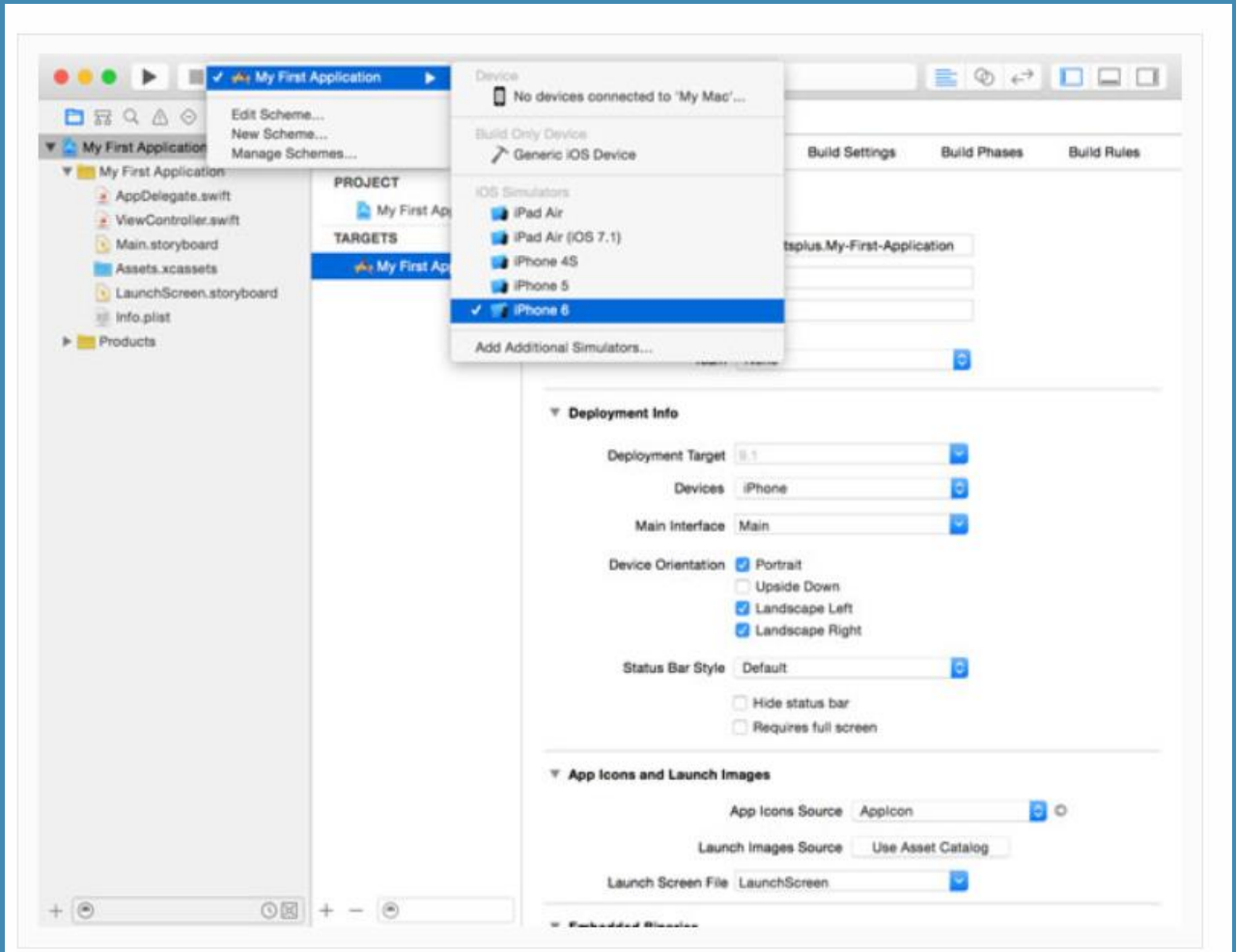
Desarrollo y Ejecución de una aplicación

Para reproducir una aplicación se debe oprimir el botón que está en la parte superior izquierda en forma de triangulo recostado, pero antes se debe verificar que el esquema seleccionado es del simulador del dispositivo requerido de algún simulador activo.



ENTORNO DE DESARROLLO

Desarrollo y Ejecución de una aplicación



ENTORNO DE DESARROLLO

Desarrollo y Ejecución de una aplicación

El simulador no tiene una cámara o un acelerómetro, sus capacidades GPS se limitan a una lista de rutas y ubicaciones predefinidas y la interacción con el usuario se limita a gestos que requieren uno o dos dedos.

Es conveniente siempre probar las aplicaciones en un dispositivo físico antes de subirse a la App Store.

ENTORNO DE DESARROLLO

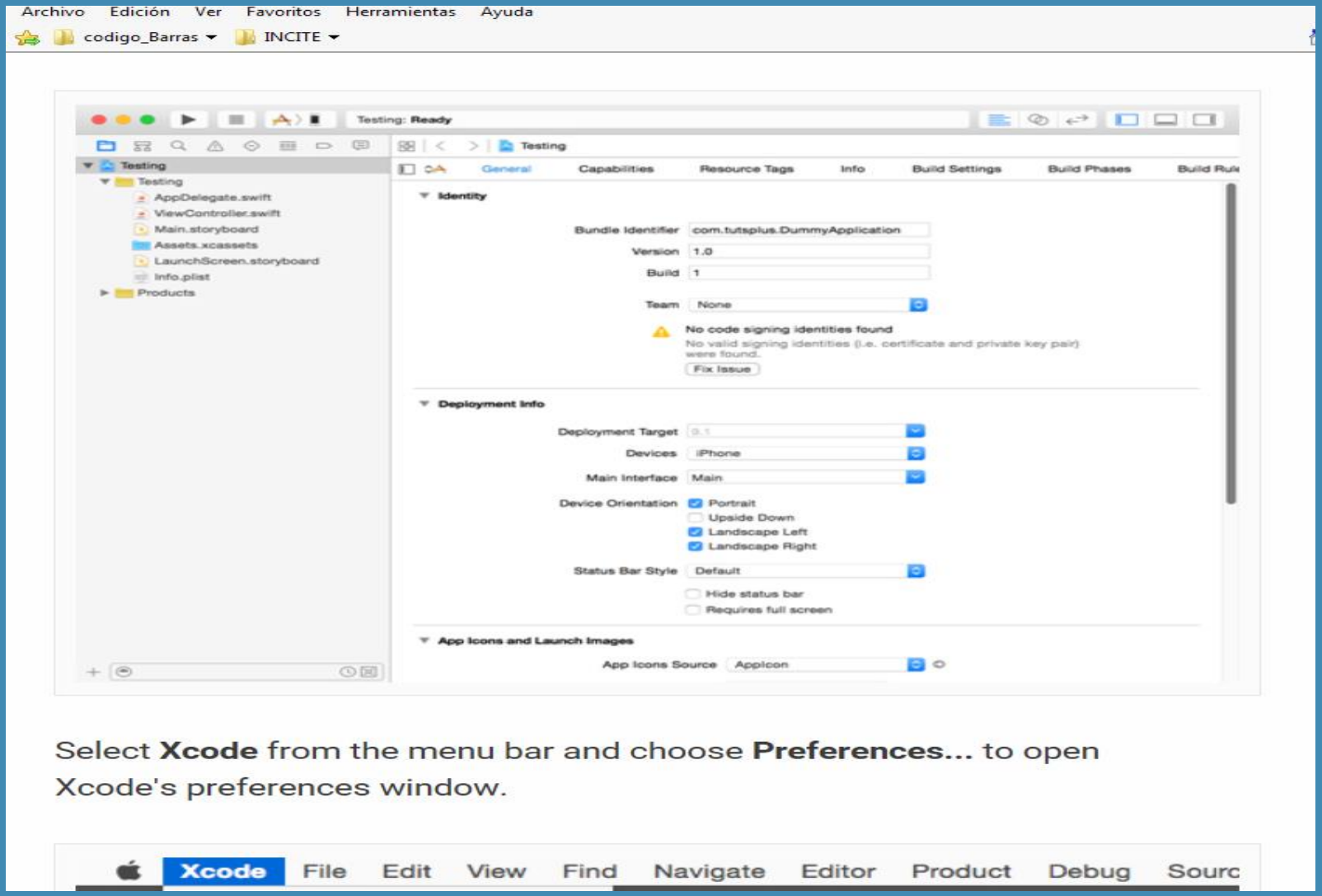
Prueba de Aplicación en un Dispositivo

Para probar una aplicación es necesario contar con una APPLE ID. Posteriormente hay que asignar el ID mencionado en la configuración del proyecto de la siguiente manera:

En la pestaña principal del proyecto y en la sección de identificación del mismo, muestra un aviso de que no se ha firmado en el campo TEAM.

ENTORNO DE DESARROLLO

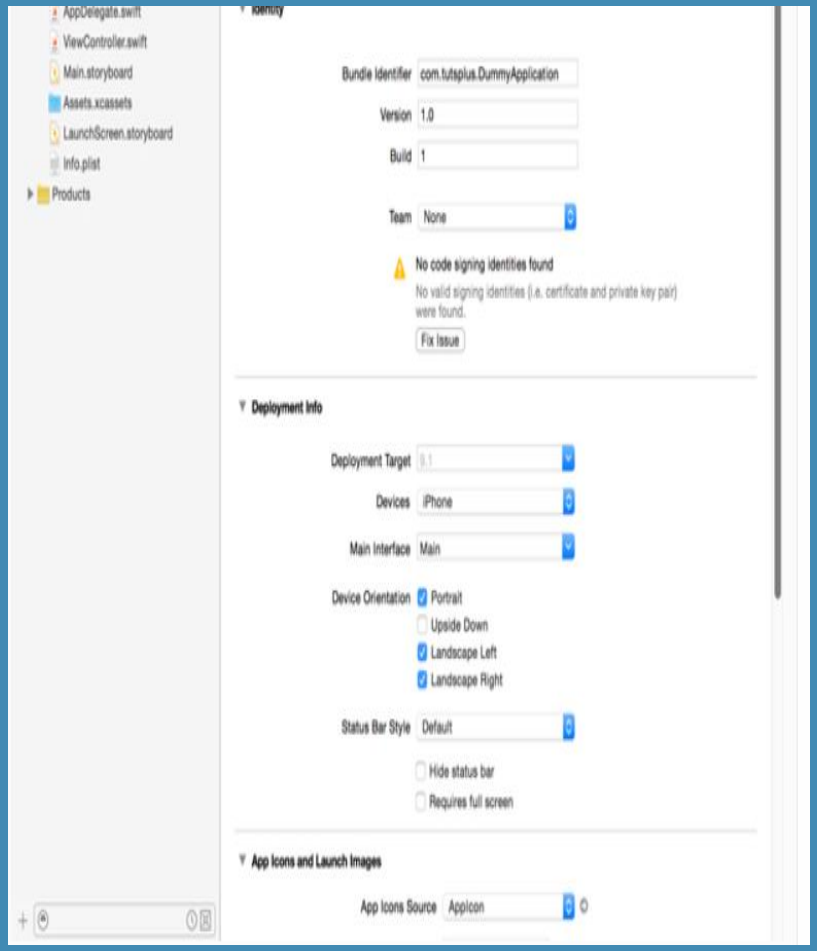
Prueba de Aplicación en un Dispositivo

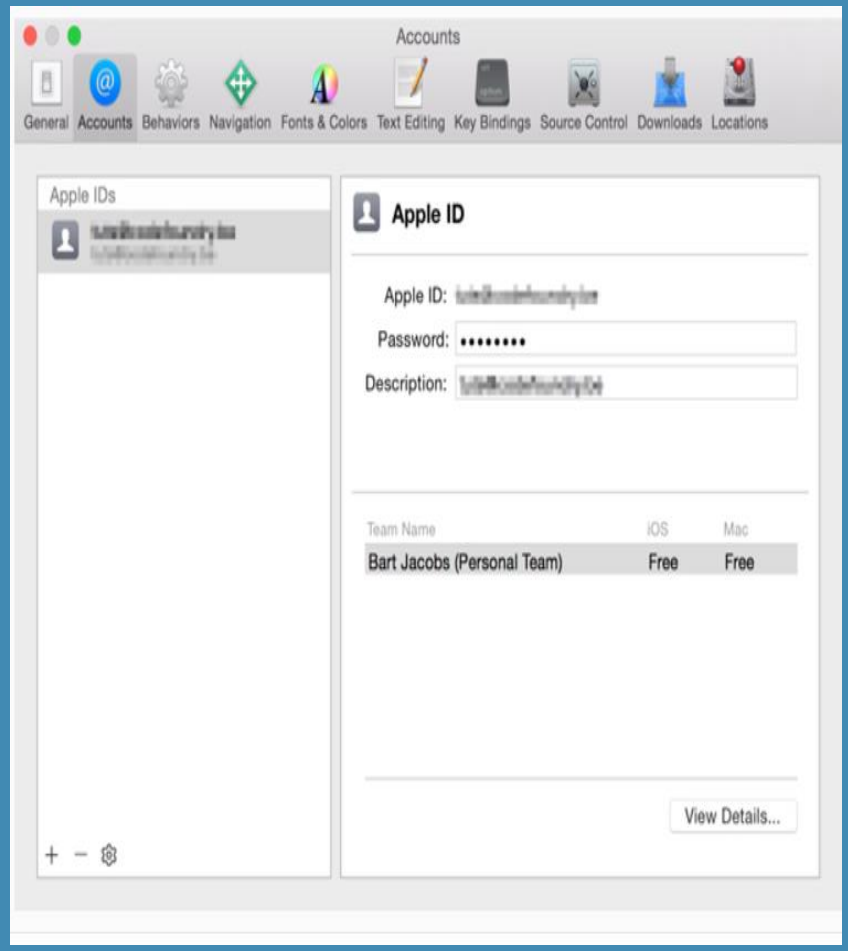


Select **Xcode** from the menu bar and choose **Preferences...** to open Xcode's preferences window.

ENTORNO DE DESARROLLO

Prueba de Aplicación en un Dispositivo





ENTORNO DE DESARROLLO

Prueba de Aplicación en un Dispositivo

Una vez dado de alta el APPLE ID, se debe regresar a la página general de la aplicación y en la sección Identificación y el campo Team se debe agregar la cuenta creada anteriormente.

Finalmente se debe utilizar la aplicación Setting y en la pestaña General Profile, se debe seleccionar el APPLE ID y en el cuadro de dialogo escoger TRUST.

ENTORNO DE DESARROLLO

El contenido de la carpeta del proyecto que se localiza del lado izquierdo contiene los siguientes elementos:

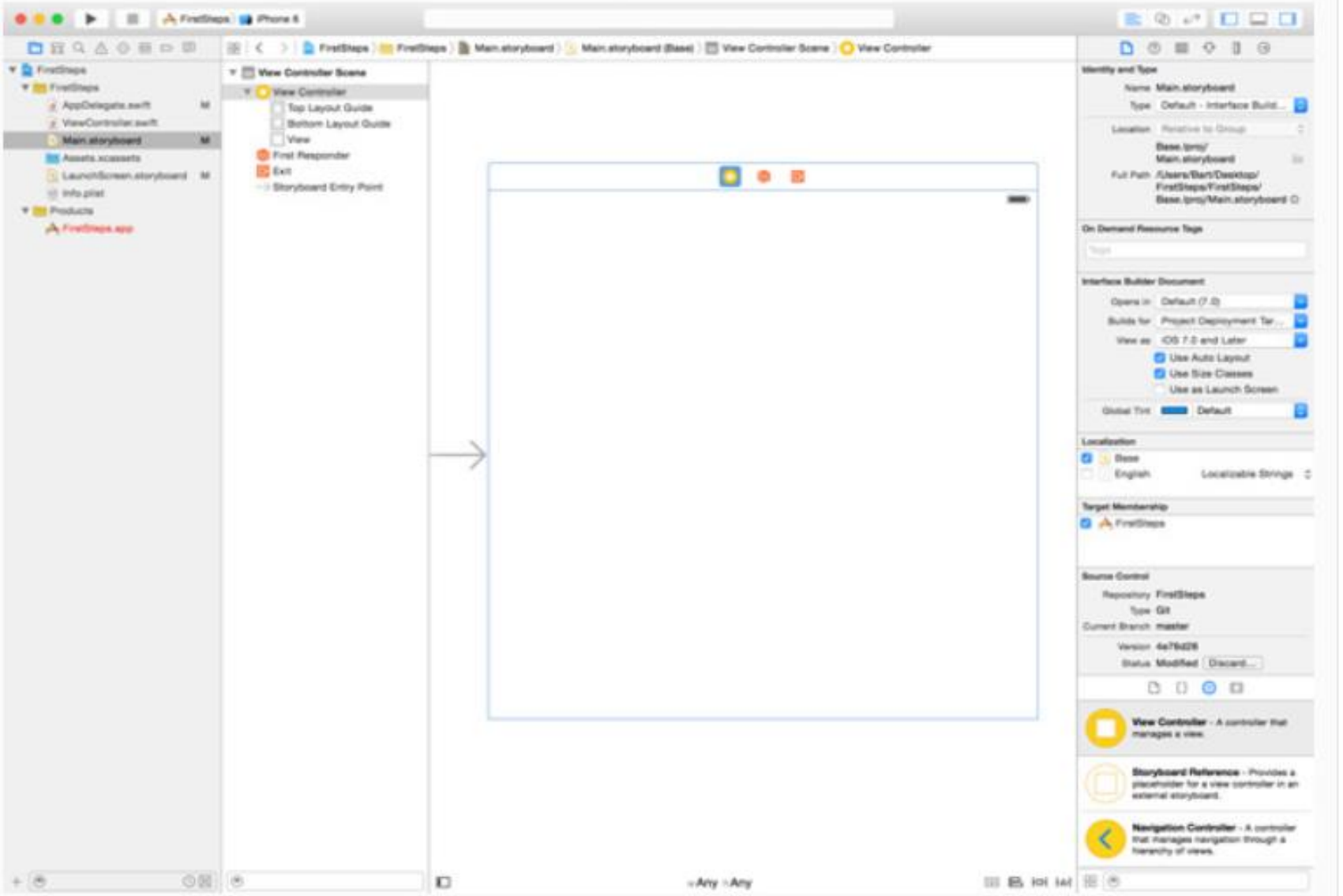
Los primeros dos archivos, AppDelegate.swift y ViewController.swift, son archivos fuente. Estos archivos contienen el código fuente de la aplicación.

Main.storyboard contiene la interfaz de usuario de la aplicación.

Info.plist, es una lista de propiedades que contiene varias configuraciones.

Assets.xcassets es un tipo especial de carpeta para almacenar los activos de su proyecto, como las imágenes.Componentes de la aplicación.

ENTORNO DE DESARROLLO



https://code.tutsplus.com/es/tutorials/ios-from-scratch-with-swift-first-steps-with-uikit--cms-25461

FRAMEWORKS

UIKIT

Las clases, protocolos y estructuras del framework UIKit, permiten la gestión entre la interface, el usuario, el sistema y la aplicación a través de una infraestructura de clases.

Gestiona el ciclo de vida de la aplicación.

Inicializa la estructura de datos de la aplicación y responde a cualquier llamado del sistema.

Centraliza la ejecución de la aplicación (clase UIApplication).

Grupo de métodos para gestionar el comportamiento de la aplicación (protocolo UIApplicationDelegate)

FRAMEWORKS
UIKIT

Clase para representar el dispositivo en turno.

Administra datos y recursos de la aplicación.

Gestiona la vista y controles que se presentan al usuario.

Gestiona la navegación entre vistas.

Gestión de los gestos.

Gestión de textos y fonts.

Gestión de Menú.

Clases, Protocolos y Estructuras

Clase, es una plantilla para la creación de objetos de datos según un modelo predefinido. Las clases se utilizan para representar entidades o conceptos, como los sustantivos en el lenguaje. Cada clase es un modelo que define un conjunto de variables o atributos -el estado, y métodos apropiados para operar con dichos datos -el comportamiento. Cada objeto creado a partir de la clase se denomina instancia de la clase.

Protocolos, son plantillas de especificación que se definen para crear reglas determinadas que se deben cumplir en una implementación que se pueden usar en varias clases o *structs*. Definen requisitos a implementar **utilizando solamente las cabeceras de las propiedades o métodos.**

Clases, Protocolos y Estructuras

Los protocolos

- No obligan a la instanciación ni a la inicialización previa.
- No soportan herencia.
- Tipos de datos por valor.

https://applecoding.com/analisis/programacion-orientada-protocolos-evolucion-swift-2

Clases, Protocolos y Estructuras

```
protocol Mortal {
 var muerto: Bool { get }
 func muere()
}

class Personaje {
 var nombre: String var vida: Int
        init(nombre:String, vida:Int) {
         self.nombre = nombre self.vida = vida
        }
 }

class Heroe: Personaje, Mortal {
     var muerto:Bool {
     return vida <= 0
     }
     func muere() {
         print("Animación de muerte")
     }
 }
```

https://applecoding.com/analisis/programacion-orientada-protocolos-evolucion-swift-2

Clases, Protocolos y Estructuras

Las estructuras o struct son muy similares a las clases, salvo las siguientes diferencias:

No soportan herencia.

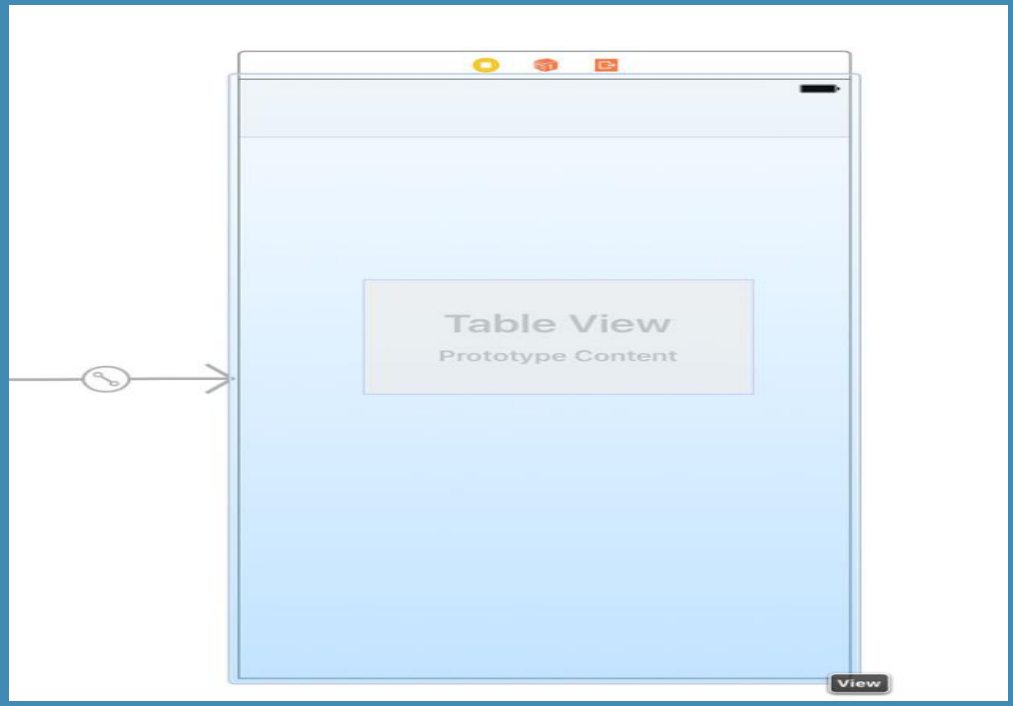
Las estructuras son tipo valor y las clases son tipo referencia.

Clases, Protocolos y Estructuras

En Swift se utilizan mucho los protocolos, ya que en casi todas las clases del modelo de vista y controlador son utilizados. Un ejemplo común es la clase ViewController que puede estar asociada a una interface gráfica que esta de conformidad a los protocolos UITableViewDataSource y UITableViewDelegate.

class ViewController: UIViewController, UITableViewDataSource, UITableViewDelegate

Clases, Protocolos y Estructuras




```
class ViewController: UIViewController, UITableViewDataSource,
UITableViewDelegate {
    @IBOutlet var tableView: UITableView!

    var items: [String] = []

    ...
}
```

Clases, Protocolos y Estructuras

```
func tableView(_ tableView: UITableView,
numberOfRowsInSection section: Int) -> Int
{
    return items.count
}

func tableView(_ tableView: UITableView, cellForRowAt indexPath:
IndexPath) -> UITableViewCell {
    // Fetch Item
    let item = items[indexPath.row]

    // Dequeue Cell
    let cell = tableView.dequeueReusableCell(withIdentifier:
"TableViewCell", for: indexPath)

    // Configure Cell
    cell.textLabel?.text = item

    return cell
}
```

TIPOS DE VARIABLES

Variables y Constantes.

var autos = 99
var nombre = ”Salvador”
let caballos = 44
let apellido = ”Sánchez”

autos = 5 // CORRECTO
caballos = 3 // ERROR
apellido = ”López" // ERROR
nombre = ”Luis" // CORRECTO

var autos = 99, nombre = “Salvador”
Let caballos = 44, apellido = “Sánchez”

TIPOS DE VARIABLES

Tipos de datos Básicos.

Int. número entero de 32 bits con un rango de entre -2.147.483.648 y 2,147,483,647.

Double, número entero de 32 bits con un rango de entre -2.147.483.648 y 2,147,483,647.

Float, número flotante de 32 bits con hasta 6 decimales de precisión.

Bool, número *booleano* que puede ser 0 (false) o 1 (true).

String, cadena de caracteres, tratada internamente con una matriz (*array*) de ellos.

TIPOS DE VARIABLES

Tipos de datos Básicos.

La inferencia de datos, se realiza a través de la propiedad que tiene el compilador de determinar por nosotros cuál es el tipo de dato a usar en una variable o constante en función de su contenido, aunque es conveniente definer el tipo de dato del que se trate ejemplos:

```
var mensaje = ”Mensaje nuevo”
Let veces = 5
```

```
var mensaje: String = ”Mensaje nuevo”
Let veces: Int = 5
```

TIPOS DE VARIABLES

Tipos de datos Complejos: colecciones.

Arreglos o matrices (array). Son una serie de elementos ya sea números o cadenas de carácteres, aunque en Swif se pueden definer indistintamente y el compilador hace la inferencia del tipo, ejemplos:

```
var matrizCadenas = ["Uno","Dos","Tres","Cuatro"]
var matrizNumeros = [1,2,3,4]
let matrizMixta = ["Uno",2,"Tres",4,true]
let matrizCadenaTipificada:[String] =
["Cinco","Seis","Siete"]
```

TIPOS DE VARIABLES

Tipos de datos Complejos: colecciones.

Arreglos o matrices (array). Para acceder a un elemento en particular, únicamente indicamos entre corchetes el número de posición del elemento dentro del arreglo iniciando con la posición cero ó elemento cero.

```
print ("Valor en pos 2 \(matrizCadenas[2])")
// Extraemos el valor 2 (posición 3) del array y lo colocamos en valor
let valor = matrizMixta[2]
var valorCadena = matrizCadenaTipificada[2]
```

TIPOS DE VARIABLES

Tipos de datos Complejos: colecciones.

Arreglos o matrices (array). Agregar elementos.

matrizCadenas += ["Cinco"]

Sustitur rangos.

matrizCadenas [2..3]= [“Tres nuevo”,”Cuatro nuevo”]

TIPOS DE VARIABLES

Tipos de datos Complejos: colecciones.

Diccionarios. La forma de definir un diccionario es a través de clave y dato de la siguiente manera:

var frutas = [”Manzanas": 3, ”Peras": 2, ”Uvas": 2, “Platanos": 2] y el compilador realizaría la inferencia siguiente:

var frutas: [String:Int] = [”Manzanas": 3, ”Peras": 2, ”Uvas": 2, “Platanos": 2].

Para asignar un valor a un dato debemos indicar su clave.
frutas[“Peras”] = 8

FLUJOS DE CONTROL

La manera de condicionar la ejecución de diversos cursos de acción se realiza a través de dos formas:

if y switch. La primera se utiliza para condicionar dos formas de flujo y la última se utiliza cuando tenemos diversos flujos de acción. A continuación se presentan ejemplos de cada una:

FLUJOS DE CONTROL

```
let matrizCadenas = ["Uno","Dos","Tres","Cuatro"]


if matrizCadenas[1] == “Dos"
{ print("Sí, el valor de la cadena es \(matrizCadenas[1])")
} if  matrizCadenas.count > 5
{ print("Hay más de cinco cadenas") }
else
{ print(“Existen cuatro o menos cadenas")
}
```

FLUJOS DE CONTROL

```
let indice = 3
let cadena  = matrizCadenas[indice]

switch cadena {
      case “Uno": print (“Es cadena Uno")
      case “Dos": print (“Es cadena Dos")
       case “Tres“: print (“Es cadena Tres")
      case “Cuatro": print (“Es cadena cuatro")
      default: print (“Cadena desconocida")
 }
```

BUCLES

Un bucle es la repetición de un pedazo de código o una iteración dentro de un arreglo o diccionario hasta que se cumpla una condición, ejemplos:

```
for x in matrizCadenas
{ print (x)
}
```

```
for num in 1...10
{
 print("Número \(num)")
}
```

BUCLES

```
var valor:Int?
let acierto = Int(arc4random()) % 20
print ("Buscando el número \(acierto)")
  repeat { valor = Int(arc4random()) % 20
      if acierto != valor { print ("No he acertado. Ha salido
            el número \(valor!)")
      }
    }
 while acierto != valor
      print("Enhorabuena. Salió el \(valor!)")
```

REFERENCIAS

https://applecoding.com/

https://developer.apple.com/documentation/uikit/

https://developer.apple.com/documentation/uikit/about_app_development_with_uikit

https://code.tutsplus.com/es/tutorials/ios-from-scratch-with-swift-first-steps-with-uikit--cms-25461

https://oleb.net/blog/2012/02/app-launch-sequence-ios-revisited/