

# IOS-XCODE INTRODUCCIÓN A SWIFT

---

*Carlos Andrés Parra, Ph.D.  
Pontificia Universidad Javeriana  
Bogotá Colombia*

# SWIFT

---

- Lenguaje orientado a objetos y funcional
- Reemplazo de ObjectiveC
- Se usa en el desarrollo de aplicaciones para Apple (MacOS e iOS)
- Lanzado por Apple en 2014, aún tiene elementos de compatibilidad con ObjectiveC
- Fuertemente tipado, con inferencia de tipos
- Actualmente en versión 5.5

[swift.org](https://swift.org)

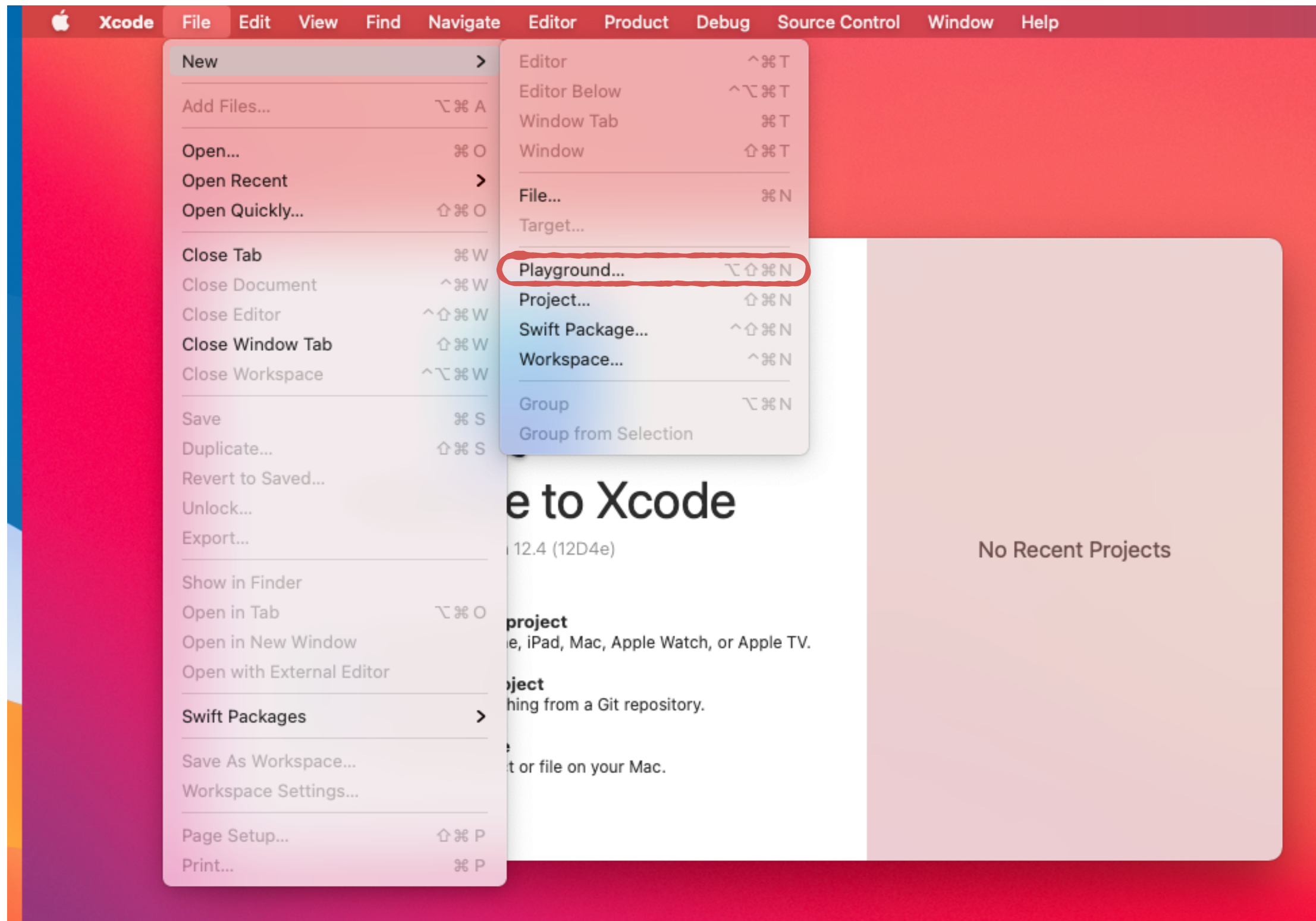
# SWIFT – HERRAMIENTAS DE DESARROLLO

---

- XCode (MacOS)
  - Entorno de desarrollo
- Swift PlayGrounds (iOS - iPad)
- Sólo el lenguaje
  - Playground Xcode
  - repl.it

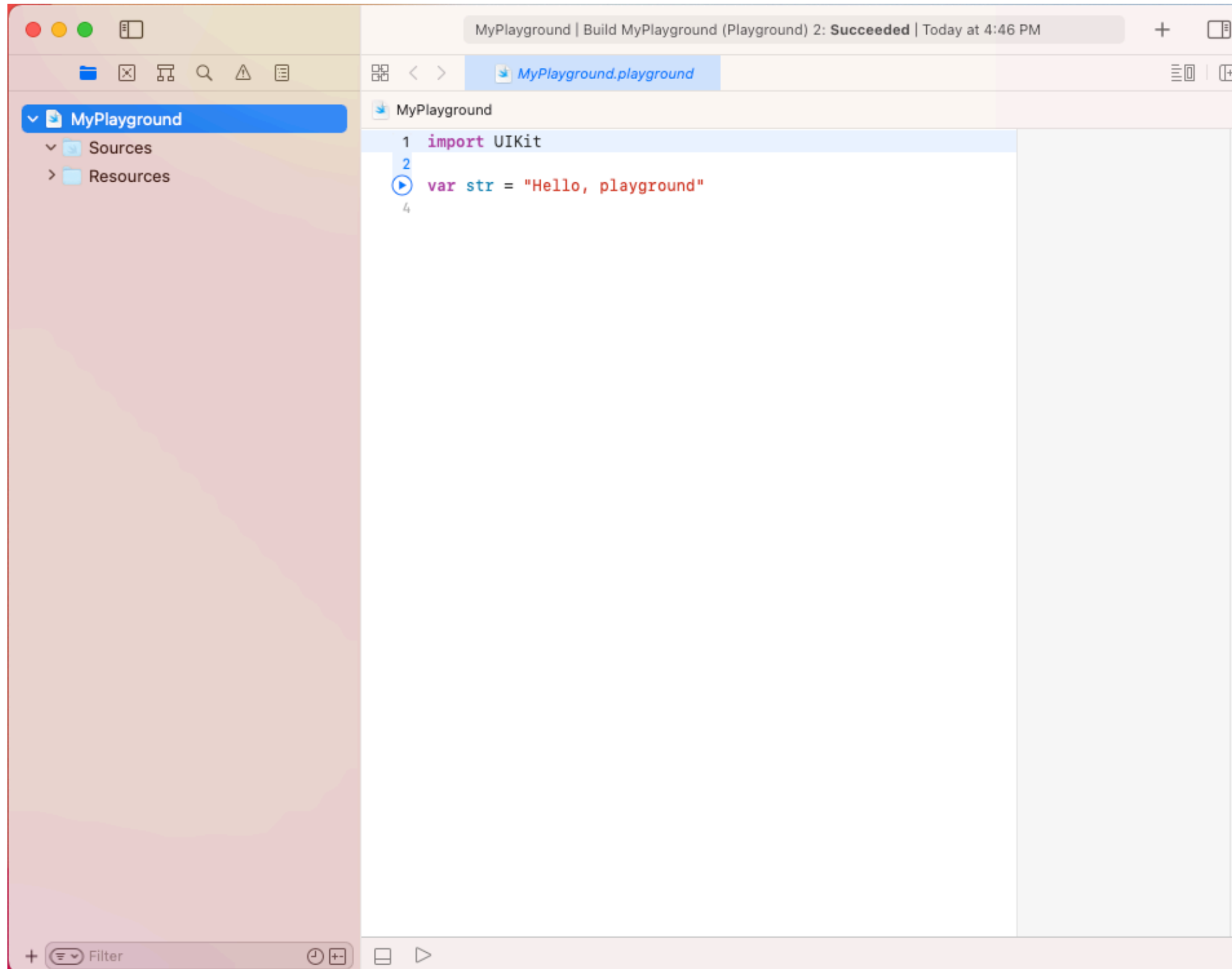
# PLAYGROUND

.....



# PLAYGROUND

---



# SWIFT – VARIABLES Y CONSTANTES

---

- Una variable puede cambiar su valor a lo largo de la ejecución, una constante no. (Mutable vs Inmutable)

```
var variable = 50  
let constante = 50
```

```
variable = 100 //ok  
constante = 20 //ko
```

```
error: MyPlayground.playground:2:11: error: cannot assign to value:  
'constante' is a 'let' constant  
constante = 20 //ko  
~~~~~ ^
```

# SWIFT – INFERENCIA Y CONVERSIÓN DE TIPOS

---

➤ Swift es fuertemente *tipado*, pero ofrece inferencia de tipos

➤ Tipos implícitos:

```
let enteroImplicito = 70
let dobleImplicito = 70.0
```

➤ Tipos explícitos:

```
let enteroExplicito : Int = 70
```

➤ Para convertir un tipo, se debe hacer una instancia del nuevo tipo:

```
let etiqueta = "el precio es: " //Definición implícita de String
let valor = 90.0 //Definición implícita de Double
let todo = etiqueta + String(valor) // Cast de Double a String
print(todo)
```

el precio es: 90.0

# SWIFT – INFERENCIA Y CONVERSIÓN DE TIPOS

---

- Otra forma es usar `\()` para incluir valores en cadenas:

```
let manzanas = 5
let peras = 7
let frase = "Tengo \(manzanas) manzanas y \(peras) peras"
print(frase)
```

Tengo 5 manzanas y 7 peras

- Texto multi-línea con 3 comillas dobles

```
let textoMultiLinea = """
Este es un texto multilinea
"Tengo \(manzanas) manzanas y \(peras) peras"
"""
print(textoMultiLinea)
```

Este es un texto multilinea  
"Tengo 5 manzanas y 7 peras"



# ARREGLOS Y DICCIONARIOS

---

- Un arreglo es un conjunto de valores ordenados, un diccionario es un conjunto de parejas en cualquier orden

```
//Array
```

```
var myArray = ["Hola", "Mundo", "hello", "world"]  
print(myArray)  
myArray[0] = "Bonjour"  
print(myArray)
```

```
//Dictionary
```

```
var myDictionary = ["iMac":1500, "iPhone":1000, "AirTag":29]  
print(myDictionary)  
myDictionary["iPhone"]=1100 //Nuevos precios  
print(myDictionary)
```

```
//Sin inicialización, se deben definir los tipos
```

```
var myArrayNoData = [String]()  
var myDictionaryNoData = [String:Int]()
```

```
["Hola", "Mundo", "hello", "world"]  
["Bonjour", "Mundo", "hello", "world"]  
["iPhone": 1000, "iMac": 1500, "AirTag": 29]  
["iPhone": 1100, "iMac": 1500, "AirTag": 29]
```

# CICLOS – CONDICIONALES

---

```
let notasIndividuales = [2.5, 3.9, 2.0, 4.5, 5]
var pasaron=0
var perdieron = Int(0) //var perdieron:Int=0
for nota in notasIndividuales {
    if nota<3.0 {
        perdieron += 1
    }else {
        pasaron+=1
    }
}
print("\(pasaron) estudiantes pasaron")
print("\(perdieron) estudiantes perdieron")
```

<b>3 estudiantes pasaron</b> <b>2 estudiantes perdieron</b>
--

# EJERCICIO 1

---

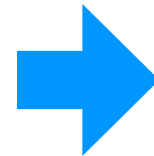
- Utilizando la herramienta PlayGround de XCode construya un diccionario que contenga los productos que se muestran a continuación:
  - Apple Watch 6: \$300
  - iPhone 12 = \$749
  - OnePlus9 = \$500
  - iMac24 = \$1400
  - PlayStation 5: \$500
  - Macbook: \$1700
  - FitBit Versa: \$200
- Construya un ciclo que itere sobre los productos y calcule el valor total de los productos que valen menos de \$1000

# LOOPS

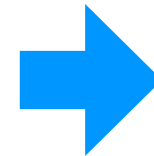


- Se puede utilizar while para iterar hasta que se cumpla una condición. También se puede dejar la condición al final para que se haga al menos una vez:

```
var n = 2
while n < 100 {
  n *= 2
}
print(n)
```



```
var m = 2
repeat {
  m *= 2
} while m < 100
print(m)
```



Se ejecuta al menos una vez

128

- For sobre rangos: se usa ... como equivalente a  $\leq$  y  $.. $<$  como equivalente a  $<$$

```
for i in 0...10{ // <=
  print(i)
}
```

0 1 2 3 4 5 6 7 8 9 10

```
for i in 0.. $<$ 10{ // <
  print(i)
}
```

0 1 2 3 4 5 6 7 8 9

- El rango puede ser de otro tipo pero hay que indicarle cuanto avanzar en cada iteración

```
for i in stride(from: 5.5, to: 10.2, by:0.4){
  print(i)
}
```

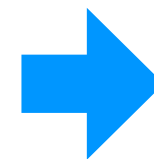
5.5  
5.9  
6.3  
6.7  
7.1  
7.5 ...

# OPTIONALS

---

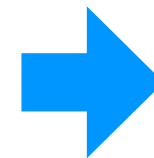
- En swift un optional es un valor que puede ser nulo (nil) o tener asociado un contenido de otro tipo. Se marcan en la definición con un interrogante:

```
var optionalString: String? = "Pika"  
print(optionalString == nil)  
print(optionalString)  
if let greeting = optionalString {  
    print("Hola \(greeting)")  
}
```



```
false  
Optional("Pika")  
Hola Pika
```

```
var optionalString: String? // = "Pika"  
print(optionalString == nil)  
print(optionalString)  
if let greeting = optionalString {  
    print("Hola \(greeting)")  
}
```



```
true  
nil
```

# OPTIONALS – GUARD

---

- Otras formas de trabajar con optionals
- Guard: similar a if let, se puede usar para detener la ejecución de un método si no se puede abrir el optional

```
var algoOpcional : String? = "Hola"
func printCadena (cadena:String?) {
    guard let miCadena = cadena else {
        print("La cadena llegó nula")
        return
    }
    print(miCadena.count)
}
printCadena(cadena: algoOpcional)
```

La cadena llegó nula

# OPTIONALS – FORCED UNWRAPPING

---

- Force unwrapped se usa para forzar a swift a tomar el valor interno del opcional, si el valor era nil, la aplicación va a fallar.
- Sólo se debe usar si se esta absolutamente seguro de que el valor no es nil. (Datos de interface builder)

```
var algoOpcional : String? //="Hola"
```

```
//Forced unwrapping  
let forced = algoOpcional!  
print(forced)
```

Hola

Fatal error: Unexpectedly found nil while unwrapping an Optional value

# OPTIONALS – CHAINING

---

- Cuando un objeto es opcional y se accede a alguna de sus propiedades, se debe usar el operador ?

```
//Optional Chaining
struct Car {
    var model:Int
    var brand:String
}
var myCar:Car?    //= Car(model: 2018, brand: "BMW")
if let model = myCar?.model {
    print(model+1)
}
```

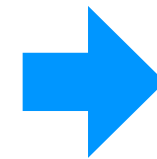


# OPTIONALS

---

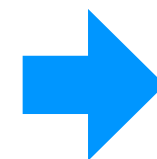
- Se puede usar el operador ?? para evaluar si el optional es diferente a nil y dar un valor alternativo

```
Var nickName:String? //="Pika"  
let fullName = "John Appleseed"  
print("Hola \(nickName ?? fullName)")
```



**Hola John Appleseed**

```
let nickName:String? = "Pika"  
let fullName = "John Appleseed"  
print("Hola \(nickName ?? fullName)")
```



**Hola Pika**

- ¿Qué pasa si en el primer caso se usa **let** para nickName?

# ITERAR SOBRE DICCIONARIOS Y OPERADOR \_

---

```
let interestingNumbers = [
    "prime": [2, 3, 5, 7, 11, 13, 17],
    "fibonacci": [1, 1, 2, 3, 5, 8, 13, 21, 34, 55],
    "square": [1, 4, 9, 16, 25, 36, 49]
]
var largest = 0
var kindLargest:String? = nil
for(kind, numbers) in interestingNumbers{
    for number in numbers{
        if (number > largest){
            kindLargest = kind
            largest = number
        }
    }
}
print("Número más largo \(largest) de tipo \(kindLargest ?? "ninguno")")
```

Número más largo 55 de tipo fibonacci

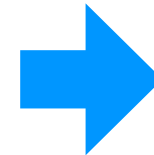
```
//Si no interesa el tipo
for(_, numbers) in interestingNumbers{
```

# SWITCHES

---

- Los switches en Swift soportan diferentes tipos de datos e incluso patrones:

```
let mascota = "gato persa"
switch(mascota) {
    case "perro beagle":
        print("🐶")
    case "perro poddle":
        print("🐩")
    case "gato":
        print("🐱")
    case let x where x.hasSuffix("persa"):
        print("🐱")
    default:
        print("🌳")
}
```



- Siempre debe haber un default. No hay necesidad de usar un break entre cases. Sí, los emojis se integran con el código y la consola

## EJERCICIO 2

---

- Construya un arreglo con 8 emojis de su preferencia.
- Con base en un número aleatorio, escoja una posición del arreglo.
- Muestre en pantalla el emoji seleccionado aleatoriamente y el nombre del emoji en texto claro en español.

```
let random = Int.random(in: 0.. $8$ )
```

# FUNCIONES Y CLOSURES

---

- Para definir una función, se usa la palabra `func`, luego se da el nombre a la función, luego entre paréntesis los parámetros separados por comas. El retorno se coloca después con el operador `->`

```
func saludo(nombre:String, dia:String) -> String {  
    return "Hola \ (nombre), hoy es: \ (dia)"  
}
```

```
//Invocacion
```

```
print(saludo(nombre: "Carlos", dia: "Jueves"))
```

Hola Carlos, hoy es: Jueves

# FUNCIONES Y CLOSURES

---

- En Swift se pueden definir etiquetas para los parámetros de una función. Las etiquetas están hechas para ser utilizadas por quien invoca la función.

```
func saludar(a nombre:String, el dia:String) -> String {  
    return "Hola \(nombre), hoy es: \(dia)"  
}  
print(saludar(a: "Carlos", el: "Jueves"))
```

- Si no se desea tener una etiqueta se usa \_

```
func saludoEtiquetaParcial(_ nombre:String, el dia:String) -> String {  
    return "Hola \(nombre), hoy es: \(dia)"  
}  
print(saludoEtiquetaParcial("Carlos", el: "Jueves"))
```

# TUPLAS

---

- Se pueden usar tuplas para devolver varios valores en una función

```
func estadisticas (entrada valores: [Int]) -> (min:Int, max:Int, sum:Int){  
    var min = valores[0]  
    var max = valores[0]  
    var sum = 0  
    for valor in valores{  
        if valor > max{  
            max = valor  
        } else if valor < min{  
            min = valor  
        }  
        sum += valor  
    }  
    return (min, max, sum)  
}  
let estadistica = estadisticas (entrada: [5, 3, 100, 3, 9])  
print(estadistica.min)  
print(estadistica.max)  
print(estadistica.sum)  
print(estadistica.2)
```

3
100
120
120

# FUNCIONES COMO TIPOS DE DATOS

---

- En Swift, las funciones también son tipos de datos, esto quiere decir que una función puede recibir como parámetro una función, o puede devolver una función como retorno

```
func hacerIncremento () -> ((Int) -> Int) {  
    let cant = 5  
    func sumar (valor:Int)->Int{  
        return valor + cant  
    }  
    return sumar  
}  
let incremento = hacerIncremento() //valor de tipo función  
print(incremento(8))
```



# FUNCIONES COMO TIPOS DE DATOS

---

- Funciones que reciben otras funciones como parámetros

```
func hasAnyMatches(list: [Int], condition : (Int)->Bool)->Bool{
    for item in list {
        if (condition(item)){
            return true;
        }
    }
    return false
}
func lessThanTwenty (value:Int) -> Bool {
    if (value<20){
        return true
    }
    return false
}
let values = [11,43, 56, 12, 5, 6, 74]
print(hasAnyMatches(list: values, condition: lessThanTwenty))
```

<b>true</b>
-------------

## EJERCICIO 3

---

- Modifique el código anterior para que ahora se cree un arreglo donde estén los números que están por debajo de 20. Pruebe el funcionamiento e imprima el resultado en la consola

# CLOSURES

---

- Un closure es un bloque de código anónimo que se puede invocar. Una función es un tipo de closure

```
let values = [11, 43, 56, 12, 5, 6, 74]
//FUNCIÓN COMO PARÁMETRO
func mySort (first:Int, second:Int) -> Bool{
    if(first>second){
        return true
    }
    return false
}
let sortedValues = values.sorted(by: mySort)
print(sortedValues)
```

**sorted** es una función de los arreglos que recibe otra función como parámetro, en donde se comparan dos enteros y se retorna un bool

```
[74, 56, 43, 12, 11, 6, 5]
```

# CLOSURES

---

- Los closures no necesitan nombre y se pueden abreviar

```
let values = [11, 43, 56, 12, 5, 6, 74]
//CLOSURE
let closureSorted =
  values.sorted(by: {(first: Int, second: Int) -> Bool in
    if (first > second) {
      return true;
    }
    return false
  })
print(closureSorted)
```

```
//CLOSURE ABREVIADO
let shortClosureSorted = values.sorted(by: {$0 > $1})
print(shortClosureSorted)
```

```
//CLOSURE AÚN MÁS ABREVIADO
let superShortClosure = values.sorted {$0 > $1}
print(superShortClosure)
```

[74, 56, 43, 12, 11, 6, 5]
----------------------------

# MAP - REDUCE - FILTER -> HIGH ORDER FUNCTIONS

```
let carros = ["renault clio": 1500, "mazda 3": 2000, .....  
             "bmw serie 3": 2500, "ford mustang": 4000]  
  
//Filtrar los carros con menos de 2000cc  
let carrosFrugales = carros.filter{ $0.value<=2000 }  
print(carrosFrugales)  
  
//Retornar un arreglo con las llaves modificadas con el sufijo turbo  
let carrosTurbo = carros.map ({ (carro) in return carro.key+" turbo"})  
print(carrosTurbo)  
  
//Retornan un diccionario con los valores incrementados en 1000  
let carrosPotenciadosDic = carros.mapValues({(value) in return value+1000})  
print(carrosPotenciadosDic)  
  
//Sumar todos los valores  
let totalCC = carros.reduce(0, {(result, pair) in return result + pair.value})  
print(totalCC)  
//Buscar el motor más grande  
let maxCC = carros.reduce (0,{(result, pair) in  
    if result<pair.value{  
        return pair.value  
    }  
    return result  
})  
print(maxCC)
```

```
["renault clio": 1500, "mazda 3": 2000]  
["bmw serie 3 turbo", "ford mustang turbo", "renault clio turbo", "mazda 3 turbo"]  
["bmw serie 3": 3500, "ford mustang": 5000, "renault clio": 2500, "mazda 3": 3000]  
10000  
4000
```

# EJERCICIO 4

---

Utilizando el mismo arreglo del ejercicio 1 desarrolle:

- Una función que retorne los artículos por debajo de mil dólares (Filter)
- Un closure que retorne el precio de los artículos en pesos colombianos (MapValues)
- Un closure que calcule el precio total de los artículos (Reduce)

# SWIFT – TIPOS DE DATOS

---

- Clases
- Estructuras
- Enumeraciones
- Protocolos
- Extensiones

# CLASS VS STRUCT

---

- Clases
  - Permiten herencia
  - Son tipos *referenciados* (referenced types)
    - Es el mismo espacio en memoria y múltiples referencias
- Estructuras
  - No permiten herencia
  - Son tipos por valor (value types)
    - siempre se hace una copia



# CLASSES

---

```
class Animal {  
    var age : Int  
    var name : String  
    let status = "Alive"  
  
    init(age: Int, name: String){  
        self.age = age  
        self.name = name  
    }  
  
    func description() -> String{  
        return "Animal \ (name) is \ (age) years old"  
    }  
}
```

*Propiedades*

*Initializer (Constructor)*

# CLASSES – HERENCIA, STORED AND COMPUTED PROPERTIES

.....

```
class Dog: Animal{
```

```
    let communicate = "brawls"  
    var breed: String
```

**Stored Properties**  
Siempre en Memoria

```
    var ageInDogYears: Int{  
        get{  
            return self.age*7  
        }  
        set{  
            //self.age = newValue*7  
        }  
    }  
}
```

**Computed Properties**  
Se evalúan cada vez que se leen  
**newValue** es el nombre por defecto

```
    init(age: Int, name: String, breed:String){  
        self.breed = breed  
        super.init(age: age, name: name)  
    }
```

Llamado al init del padre

```
    override func description() -> String {  
        return "\(name) is a \(breed),  
            it is \(age) years old which is \(ageInDogYears)  
            in dog years"  
    }  
}
```

# CLASSES

---

```
let animal = Animal(age: 5, name: "Nemo")
var dog = Dog(age: 6, name: "Marley", breed: "Labrador")
print(animal.description())
print(dog.description())

dog.age = 8
print(dog.description())
```

**Animal Nemo is 5 years old**

**Marley is a Labrador, it is 6 years old which is 42 in dog years**

**Marley is a Labrador, it is 8 years old which is 56 in dog years**

# CLASSES – LISTENERS FOR PROPERTIES

---

```
class Dog: Animal{  
    var health= "OK"
```

```
    override var age: Int{  
        willSet{  
            if newValue > 12 {  
                health="Check with Vet"  
            }  
        }  
    }  
}
```

Sólo esta en la edad del perro, no del animal  
willSet -> antes del cambio, newValue existe  
didSet -> después del cambio, se usa age

```
    func healthStatus() -> String {  
        return "Health status \ \(health)"  
    }  
}
```

```
let dog = Dog(age: 6, name: "Marley", breed: "Labrador")  
print(dog.description())  
dog.age = 8  
print(dog.description())
```

```
dog.age = 13  
print(dog.healthStatus())
```

**Marley is a Labrador, it is 8 years old which is 56 in dog years.  
Health status OK  
Health status Check with Vet**

# STRUCTS VS CLASSES

---

- **Struct** -> Soportan casi las mismas características que las clases pero no tienen herencia y son value types, se hacen copias en cada utilización

```
class ComputerC{  
    var brand = "apple"  
    var ram = 8  
    func description() -> String {  
        return "Class: brand \(brand)  
            and ram \(ram)"  
    }  
}
```

```
var computerC = ComputerC()  
print(computerC.description())  
var newComputerC = computerC  
newComputerC.ram=16  
newComputerC.brand = "Dell"  
print(computerC.description())
```

```
Class: brand apple and ram 8  
Class: brand Dell and ram 16
```

```
struct ComputerS{  
    var brand = "apple"  
    var ram = 8  
    func description() -> String {  
        return "Struct: brand \(brand)  
            and ram \(ram)"  
    }  
}
```

```
var computerS = ComputerS()  
print(computerS.description())  
var newComputerS = computerS  
newComputerS.ram=16  
newComputerS.brand = "Dell"  
print(computerS.description())
```

```
Struct: brand apple and ram 8  
Struct: brand apple and ram 8
```

# ENUMERATIONS

---

- Una enumeración sirve para almacenar un conjunto de valores, también puede tener métodos asociados

```
enum Food : String {  
    case Concentrado, Leche, Atun  
}
```

```
var aFood = Food.Atun
```

```
print("Value: \ (aFood)")  
print("Raw Value: \ (aFood.rawValue)")
```

<b>Value: Atun</b> <b>Raw Value: Atun</b>
--

# ENUMS

---

```
enum Rank: Int {  
    case ace = 1 // Start at one not at zero, the rest follow after  
    case two, three, four, five, six, seven, eight, nine, ten  
    case jack, queen, king  
    func simpleDescription() -> String {  
        switch(self){  
            case .ace:  
                return "ace"  
            case .jack:  
                return "jack"  
            case .queen:  
                return "queen"  
            case .king:  
                return "king"  
            default:  
                return String(self.rawValue)  
        }  
    }  
}  
  
let fourth = Rank.four  
print(fourth)  
let fourthRaw = fourth.rawValue  
print(fourthRaw)
```

<b>four</b> <b>4</b>
-------------------------

# ENUMS

---

- Se puede acceder al método de inicialización por defecto

- `init?(rawValue:)`

```
//invocación a init?(rawValue:)  
//Es un optional!  
if let newCard = Rank(rawValue: 11){  
    print(newCard.simpleDescription())  
}
```

**jack**

- El optional sería nil en caso de que el número estuviera por fuera de los valores de la enumeración (p.ej. 15). En se caso, con el if let, el bloque de código de la aplicación no se ejecuta.



# PROTOCOLOS

---

- Se usa para definir un comportamiento que se debe respetar por otros tipos como structs y classes

```
protocol ExampleProtocol {  
    var simpleDescription: String { get }  
    mutating func adjust()  
}
```

Define un protocolo. Todos los tipos que lo implementen deben tener una propiedad y un método `adjust()`

```
extension Int: ExampleProtocol {  
    var simpleDescription: String {  
        return "The number \(self)"  
    }  
    mutating func adjust() {  
        self += 42  
    }  
}
```

En Swift, los métodos de las estructuras no pueden cambiar los valores de las propiedades por defecto, si se quiere hacer, hay que definir al método como **mutating**

```
var number: Int = 7  
print(number.simpleDescription)  
number.adjust()  
print(number.simpleDescription)
```

<b>The number 7</b> <b>The number 49</b>
---

# EJERCICIO 1

.....

1. Construya una clase Gato que herede de Animal
2. Modifique el método description() para que represente mejor la información del gato. Pruebe e imprima el resultado en la consola.
3. Construya una propiedad computable para manejar un emoji asociado al gato. Si el nombre del gato es de menos de 5 letras, el emoji que se debe usar es: 🐱 , sino se debe usar: 🐈 . Pruebe e imprima el resultado en la consola
4. Cree una propiedad que se llame alimento, que corresponde a una enumeración que puede tomar los valores Concentrado, Leche, Atún.
5. Modifique su clase para que cuando se haga un cambio sobre la propiedad edad, se modifique de la siguiente manera, si es menor a 10, el alimento debe ser concentrado, si esta entre 10 y 15 el alimento debe ser leche y finalmente si es mayor a 15 el alimento debe ser Atún. Pruebe e imprima el resultado en la consola.
6. Defina un protocolo nuevo que se llame AnimalNadador. Este protocolo agrega una propiedad distancia, que representa la cantidad de metros que puede nadar un animal y un método nadar.
7. Extienda su gato con el nuevo protocolo, el método nadar debe retornar una frase que indique la cantidad de metros que puede nadar un gato (p.ej. 30). Pruebe e imprima el resultado en la consola

# CONTROL DE ACCESO

---

- En Swift el control de acceso está basado en **módulos** y **archivos**
- **Modulo** es un conjunto de archivos, por ejemplo una app o una librería
- Un **archivo** es individual y pertenece a un módulo
- Se tienen 5 posibles valores de acceso:
  - **Open** access and **public** access enable entities to be used within any source file from their defining module, and also in a source file from another module that imports the defining module.
  - **Internal** access enables entities to be used within any source file from their defining module
  - **File-private** access restricts the use of an entity to its own defining source file.
  - **Private** access restricts the use of an entity to the enclosing declaration

<https://docs.swift.org/swift-book/LanguageGuide/AccessControl.html>

# CONTROL DE ACCESO

---

## ➤ Definición

```
public class SomePublicClass {}  
internal class SomeInternalClass {}  
fileprivate class SomeFilePrivateClass {}  
private class SomePrivateClass {}  
  
public var somePublicVariable = 0  
internal let someInternalConstant = 0  
fileprivate func someFilePrivateFunction() {}  
private func somePrivateFunction() {}
```

## ➤ Valores por defecto

```
class SomeInternalClass {}           // implicitly internal  
let someInternalConstant = 0         // implicitly internal
```

<https://docs.swift.org/swift-book/LanguageGuide/AccessControl.html>

# STATIC, CLASS FUNC

---

- Al igual que en java, se pueden definir propiedades o métodos que están asociados a la Clase y no al Objeto
- Para esto hay dos opciones
  - static
    - para funciones de structs y enums
    - propiedades
  - class func
    - para funciones de clases y protocolos
    - se pueden sobre-escribir en la herencia
    - un struct o enum que implemente un protocolo cambia la palabra class por static

# SWIFT – MANEJO DE MEMORIA

---

- Manejo de memoria basado en ARC - Automatic Reference Counting
- Se hace una limpieza basada en referencias
- Dependiendo de quién referencia un valor, se borra del heap o no.
- Cada atributo puede tener los siguientes indicativos
  - *Strong* no se borra del heap mientras alguien tenga una referencia. Si hay un deadlock ninguna se borra
  - *Weak* referencia débil que evita deadlocks
  - *Unowned* como weak pero no puede llegar a ser nil

# DIRECTIVAS – OTROS

---

- Principales directivas que se usan para el desarrollo con iOS
  - @IBOutlet -> Interface Builder Outlet
  - @IBAction -> Interface Builder Action
- NSObject, NS\*: provienen de ObjectiveC. NSObject es equivalente a Object en Java. Todo lo que empieza con NS corresponde a tipos de datos de ObjectiveC, se usa para compatibilidad con APIs escritas en ese lenguaje.

# CONVERSIÓN DE TIPOS

---

- Se utiliza la palabra reservada `as` para realizar un cast entre diferentes tipos

```
override fun prepare(for segue: UIStoryboardSegue, sender: Any?) {  
    let tvc = segue.destination as? TargetViewController  
    tvc?.texto = campoTexto.text!  
}
```



# HERRAMIENTAS

---

## ► Xcode

×



## Welcome to Xcode

Version 9.4.1 (9F2000)



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

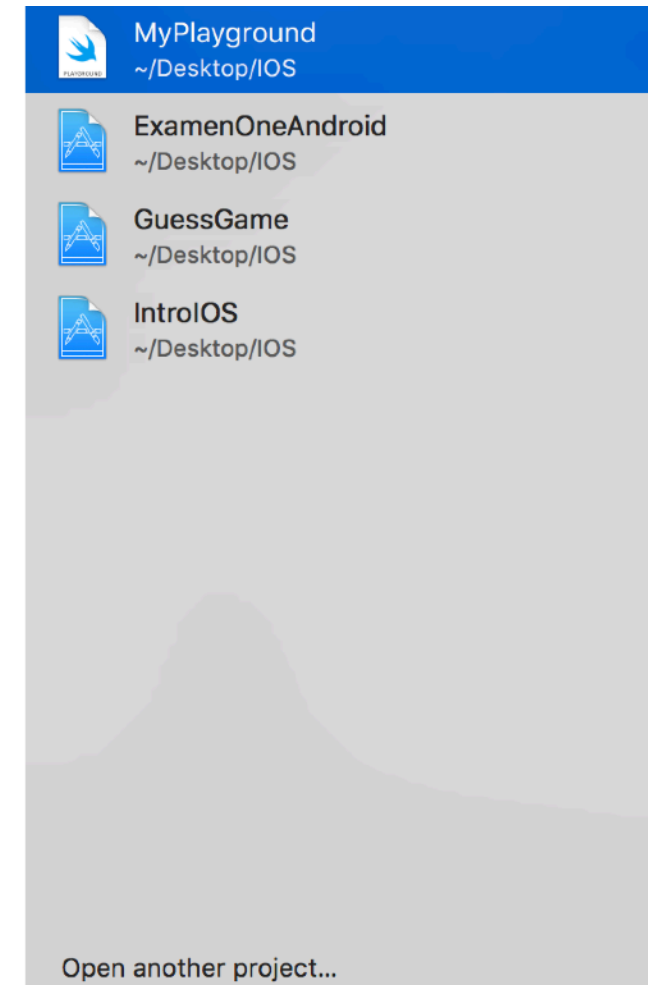


**Create a new Xcode project**  
Create an app for iPhone, iPad, Mac, Apple Watch or Apple TV.



**Clone an existing project**  
Start working on something from an SCM repository.

☒ Show this window when Xcode launches



# NUEVO PROYECTO

Choose a template for your new project:

iOS

watchOS

tvOS

macOS

Cross-platform

Filter

## Application

1

Single View App



Game



Augmented  
Reality App



Document  
Based App



Master-Detail App



Page-Based App



Tabbed App



Sticker Pack App



iMessage App

## Framework & Library



Cocoa Touch  
Framework



Cocoa Touch  
Static Library



Metal Library

Cancel

Previous

Next

# NOMBRE DE APP Y DATOS DE DESARROLLADOR

.....

Choose options for your new project:

Product Name: Clase2

Team: Carlos P. (Personal Team) ←

Organization Name: Andres Parra

Organization Identifier: co.edu.javeriana

Bundle Identifier: co.edu.javeriana.Clase2

Language: Swift

☐ Use Core Data

☒ Include Unit Tests

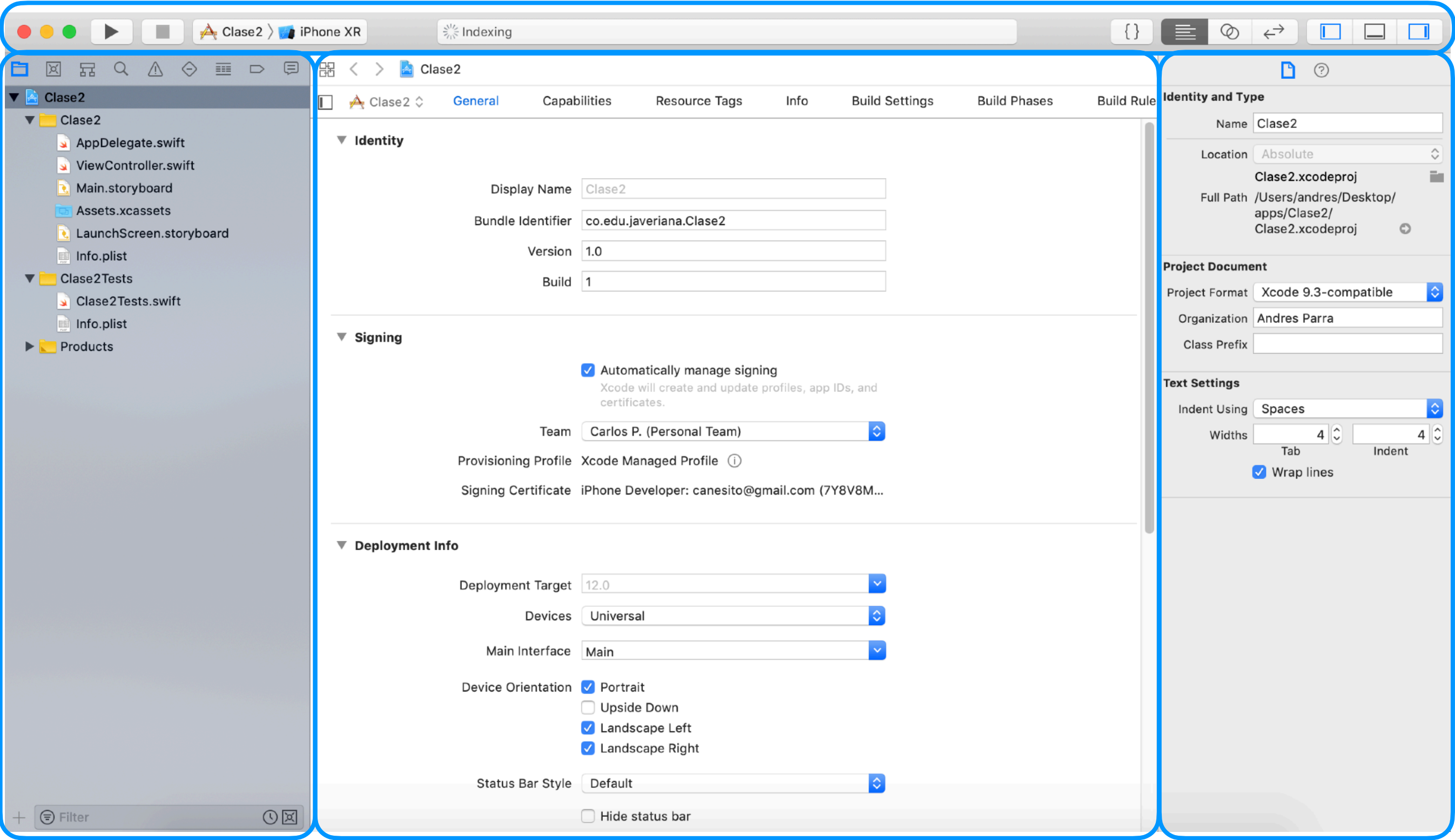
☐ Include UI Tests

Cancel Previous Next

Necesario para  
ejecutar el proyecto  
en un dispositivo  
propio

# PANTALLA PRINCIPAL

## Toolbar



Navigator

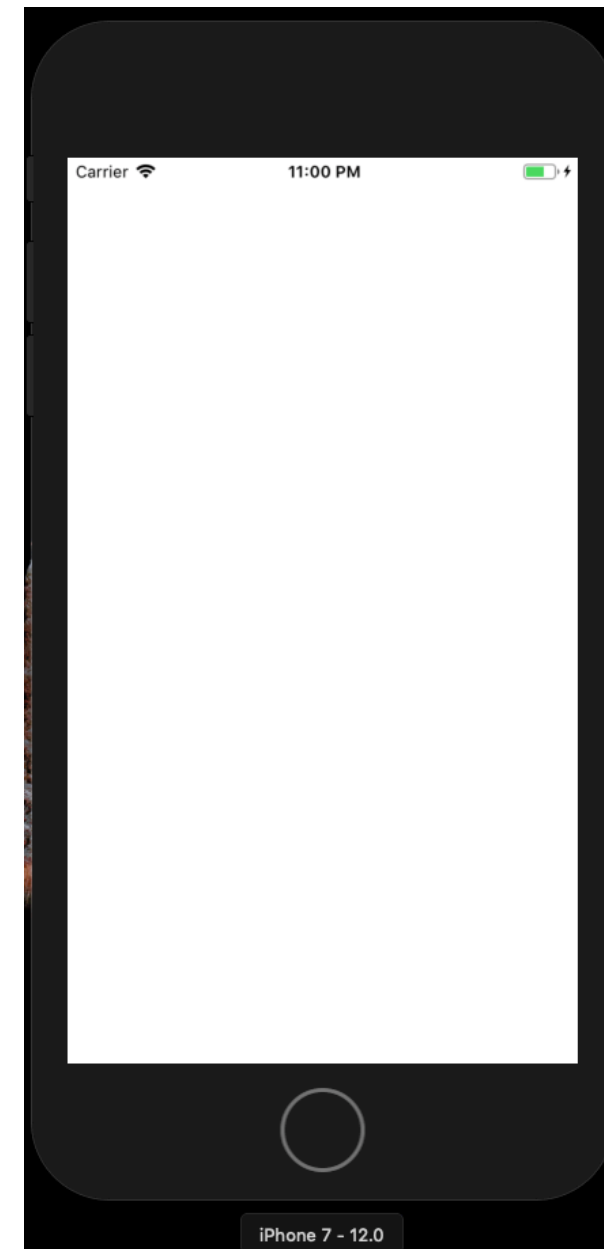
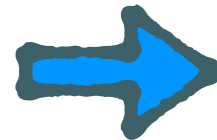
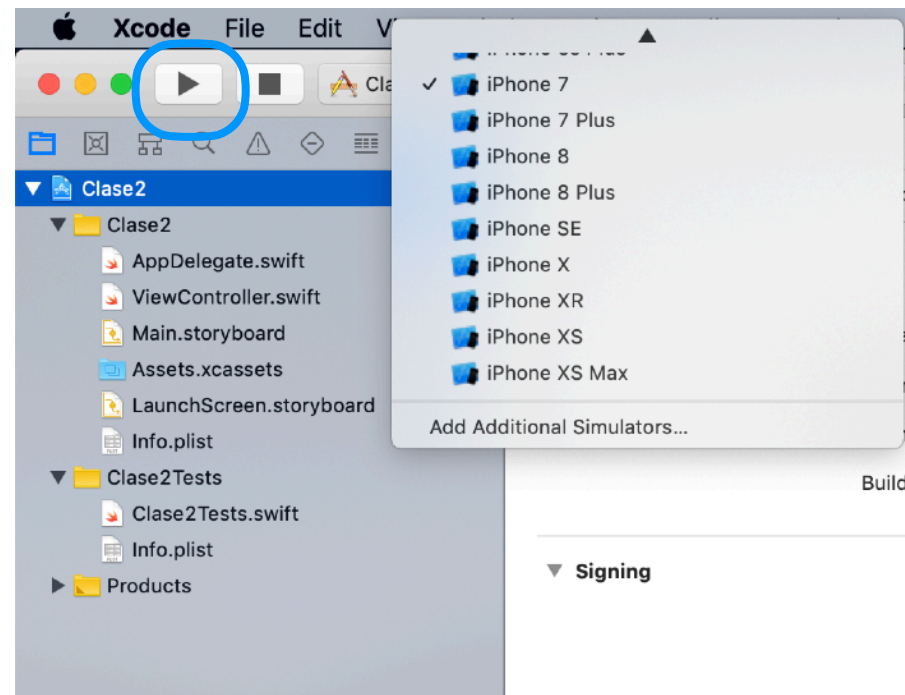
Editor Area

Inspector Area

# EMULADOR

---

- Desde el toolbar, seleccionar un dispositivo y click en Play



# CORRER APLICACIÓN EN EL DISPOSITIVO

---

- Para correr la sección en un dispositivo propio, es necesario asignar una cuenta de Apple a la aplicación. Desde el menu general:

---

## ▼ Signing

☒ Automatically manage signing

Xcode will create and update profiles, app IDs, and certificates.

Team Carlos P. (Personal Team) ▾

Provisioning Profile Xcode Managed Profile ⓘ

Signing Certificate iPhone Developer:

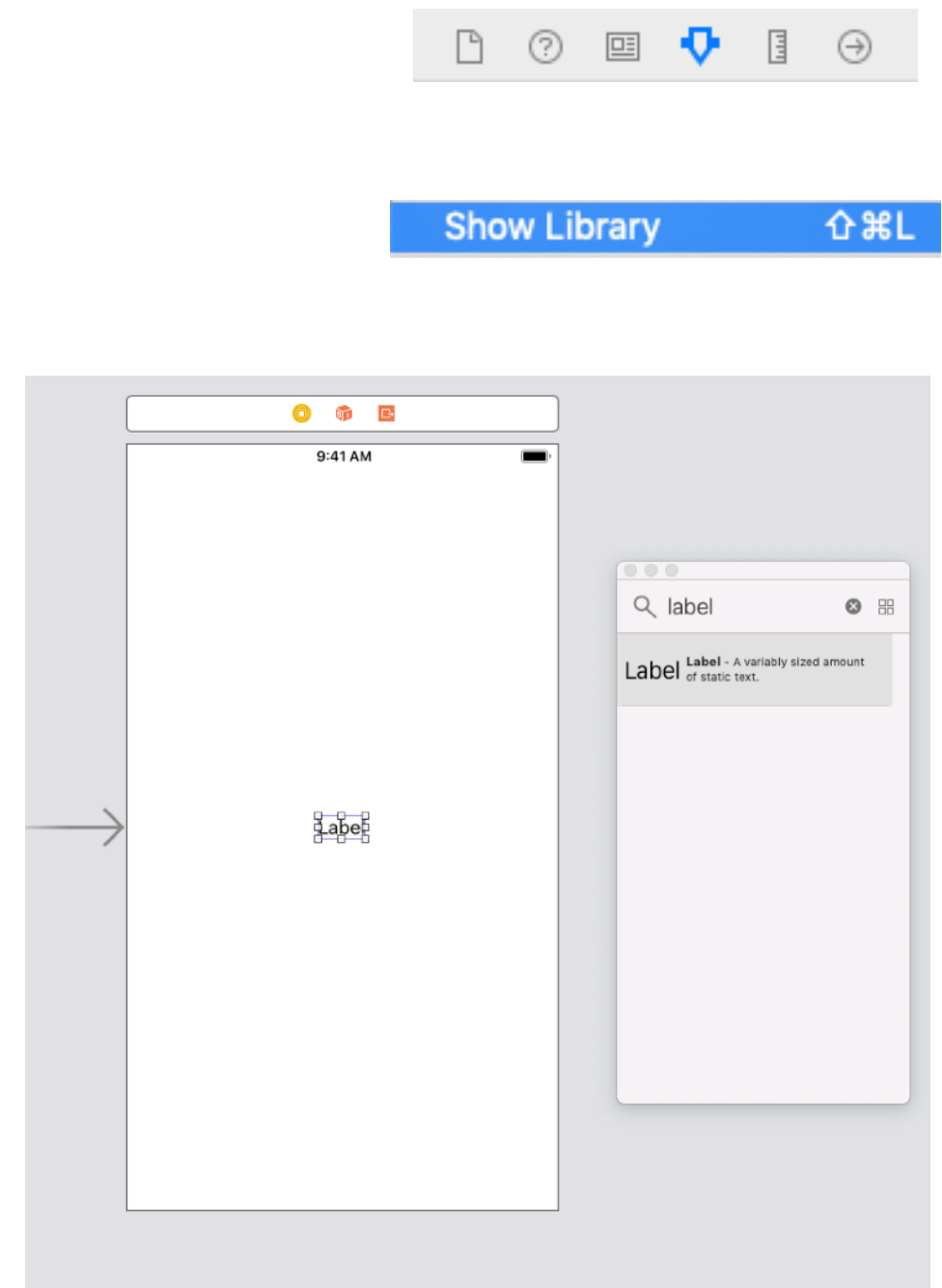
---

- Luego se debe conectar el dispositivo y aceptar los mensajes de tipo “*trust device*”

# HOLA MUNDO!

---

- Abrir el StoryBoard
- Abrir la librería de objetos.
  - En Xcode 9, inspector de atributos
  - En Xcode 10 y posteriores
    - View->Libraries->Show Library
- Buscar un label en la librería
- Drag and Drop
- Cambiar las propiedades
- Probar en el emulador



# VIEWS

---

## iOS

- TextField
- Label
- Button
- UIImageView
- Segue
- ViewController

## Android

- EditText
- TextView
- Button
- ImageView
- Intent?
- Activity?




# OUTLETS & ACTIONS

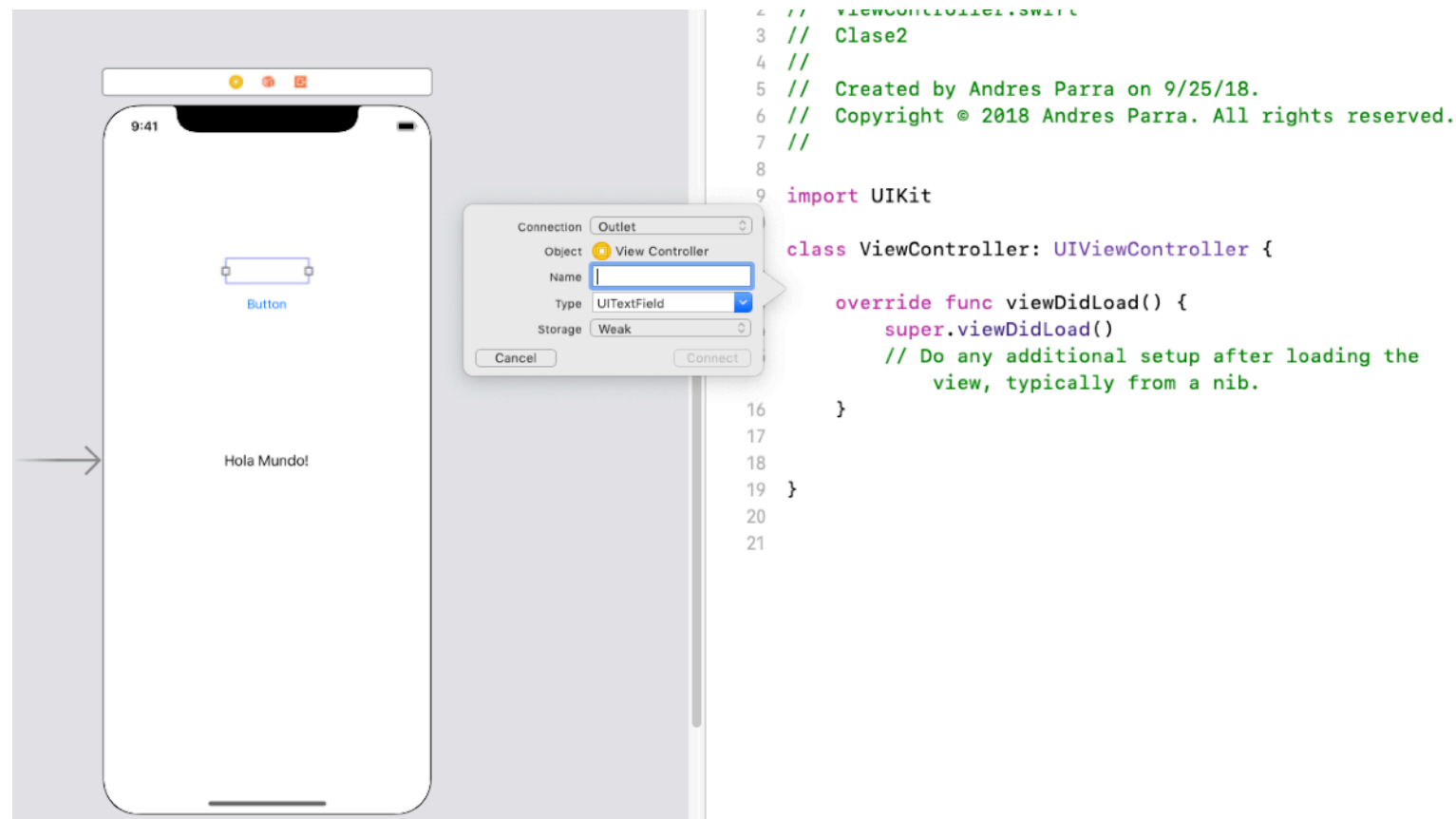
---

- Outlet:
  - Propiedad en swift que representa un elemento de Interface Builder
  - Por ejemplo, una propiedad que representa el contenido de un TextField
- Action
  - Acción que sucede cuando se interactúa con un elemento de Interface Builder
  - Por ejemplo, una función que se invoca como respuesta a oprimir un botón

# CREACIÓN DE OUTLETS

---

- Utilizar el editor combinado (Assistant Editor) 
- Ctrl+Drag and Drop desde el StoryBoard, hacia el archivo del controlador (\*.swift)
- En el diálogo que aparece, escoger outlet o action según sea el caso, dar un nombre, y personalizar el sender



# CREACIÓN DE OUTLETS

---

- Como resultado del diálogo de creación del Outlet, Xcode agrega la siguiente línea de código al controlador:

```
@IBOutlet weak var campoTexto: UITextField!
```

- @IBOutlet -> Conexión con InterfaceBuilder
- weak -> referencia débil para el manejo de memoria
- var -> propiedad que puede variar su contenido
- campoTexto -> nombre de la propiedad en el diálogo
- UITextField! -> Implicitly unwrapped de tipo UITextField

# CREACIÓN DE ACTIONS

---

- Las acciones se crean de la misma manera que los outlets (Ctrl+Drag desde IB hacia el controlador), pero se selecciona el tipo de conexión en el dialogo:



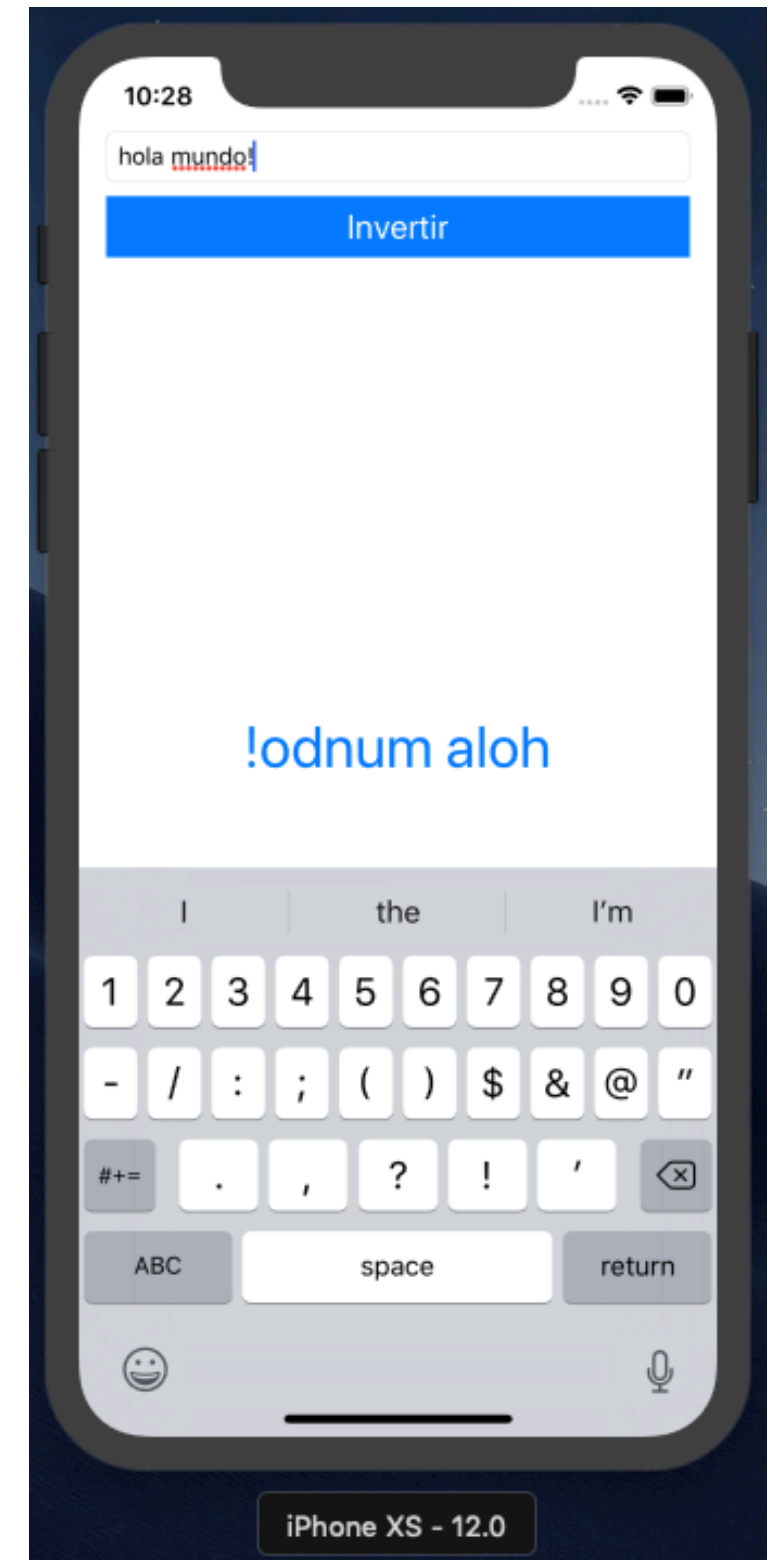
- En ese caso Xcode genera la siguiente función:

```
@IBAction func saludo(_ sender: Any) {  
}
```
- @IBAction -> Conexión con InterfaceBuilder
- función saludo con un parámetro sin etiqueta externa de tipo Any. Se ejecuta cuando se oprima el botón.

# EJERCICIO 1

---

- Construya una interfaz simple como la de la figura con un campo de texto y un botón.
  - Cuando se oprima el botón, se debe mostrar la cadena ingresada invertida. Por ejemplo si la entrada es hola mundo!, se debe mostrar un texto con la cadena !odnum aloh.
- ¿Qué pasa con el teclado?



# FIRST RESPONDER Y TECLADO

---

- En swift, existe la noción del “First Responder”. Corresponde a la vista o contenido que esta en interacción directa con el usuario.
- En el ejemplo anterior, cuando el usuario toca el campo para ingresar el texto, este se convierte en el first responder, y nunca renuncia a esa posición, es por esto que el teclado nunca desaparece.
- Para renunciar a ser el “First Responder” hay que invocar a la función `resignFirstResponder()`
- Por ejemplo, en el ejemplo anterior, cuando se oprime el botón invertir, se debe invocar:
  - `campoTexto.resignFirstResponder()`
- También se puede ocultar el teclado sobrescribiendo la función `touchesBegan` para quitar el teclado si el usuario toca en cualquier parte de la pantalla:

```
override func touchesBegan(_ touches: Set<UITouch>, with event: UIEvent?) {  
    //campoTexto.resignFirstResponder() sólo un campo de texto  
    self.view.endEditing(true)  
}
```

## EJERCICIO 2

---

- Mueva la etiqueta de respuesta a la parte inferior de la pantalla, de tal forma que el teclado la cubra cuando se despliegue.
- Modifique su aplicación para que el teclado desaparezca cuándo se oprima el botón, o cuándo se toque una parte diferente de la pantalla.

# EJERCICIO 3 – GUESSGAME

---

- Construya una aplicación con las siguientes características:
- Defina una escena en la que el usuario debe adivinar un número entre 0 y 50 generado de forma aleatoria.
- Por cada intento, la aplicación debe informar al usuario si el numero ingresado es menor o mayor al número.
- Se deben contabilizar todos los intentos realizados por el usuario. Si el usuario repite un número, este no cuenta como un nuevo intento.
- Finalmente si el usuario ingresa el número correcto, se debe mostrar un texto que indique la finalización del juego
- Recuerde manejar el FirstResponder para obtener un comportamiento adecuado del teclado

10:38



Adivina un número entre 0 y 50

Verificar

Mensaje: el número 15 es mayor

Intentos: 2



# TAREA

---

Leer la guía completa de Swift 5.5

[https://docs.swift.org/swift-book/GuidedTour/  
GuidedTour.html](https://docs.swift.org/swift-book/GuidedTour/GuidedTour.html)

Completar los ejercicios usando PlayGround de Xcode o

<https://replit.com/~>