Hello Swift

Copyright

Guía de Estudio

El libro está pensado para aquellos que quieren dar el primer paso en el mundo de iOS y para quienes necesitan actualizarse al nuevo lenguaje. Es un excelente complemento al curso practico que se encuentra en nuestro sitio web, https://nacioncodigo.com, esta guia te servirá para extender los conceptos y aprender cosas nuevas, podrás consultar cualquier duda rápidamente.

Imágenes

Las imágenes del libro corresponden con capturas de pantalla del curso en video como también realizadas por el instructor, para ayudarte a obtener una referencia temporaL. No son un elemento esencial para seguir la guia

Feedback o errores

Si encuentras algún error, o quieres brindarnos un feedback, puedes escribirnos a soporte@nacioncodigo.com

Agradecemos tu apoyo.

01

Introducción a Swift

Acerca de Swift.

Acerca de Swift

Swift es un lenguaje de programación moderno, seguro, rápido e interactivo que se puede usar para desarrollar software para una variedad de dispositivos, incluidos teléfonos, computadoras y servidores. Swift combina lo mejor del diseño de lenguaje moderno con los conocimientos de la ingeniería de Apple y la comunidad de código abierto. El compilador de Swift está optimizado para el rendimiento, mientras que el lenguaje está optimizado para el desarrollo. Como resultado, Swift es una excelente opción para desarrollar software rápido, seguro y elegante.

Swift ha sido diseñado para ser fácil de aprender y usar, incluso para los programadores nuevos. Es un lenguaje de programación de gran calidad, con una sintaxis clara y concisa. Swift también es un lenguaje expresivo, lo que significa que es fácil escribir código que sea claro y fácil de entender.

Los playgrounds de Swift son una excelente manera de experimentar con el código y ver los resultados de manera inmediata. Los playgrounds son una forma de ejecutar código Swift en una ventana del navegador, sin la necesidad de crear y ejecutar una aplicación. Esto hace que sea fácil probar nuevas ideas y ver los resultados de su código

Swift es un lenguaje de programación rápido y seguro que se puede usar para crear todo, desde aplicaciones pequeñas hasta sistemas operativos grandes. El código Swift se compila en código de máquina, lo que significa que se puede ejecutar directamente en el hardware, sin la necesidad de un intérprete. Esto hace que Swift sea muy rápido, especialmente en comparación con los lenguajes interpretados.

Swift se ha estado desarrollando por años y continúa evolucionando con nuevas características y capacidades. Nuestros objetivos para Swift son ambiciosos. Estamos ansiosos por ver lo que creas con él.

Compatibilidad de Versiones

Esta guía describe Swift 5.9, la versión predeterminada de Swift que se incluye en Xcode 15. Puedes usar Xcode 15 para compilar targets desarrollados en Swift 5.9, Swift 4.2, o Swift 4.

Al usar Xcode 15 para compilar código Swift 4 y Swift 4.2, la mayoría de las funcionalidades de Swift 5.9 estarán disponibles. Dicho esto, los siguientes cambios solo están disponibles para código desarrollado en Swift 5.9 o posterior:

- Las funciones que devuelven un tipo opaco requieren el runtime de Swift 5.1.
- La expresión **try?** no introduce un nivel adicional de opcionalidad en las expresiones que ya devuelven opcionales.
- Se infiere que las expresiones de inicialización de enteros literales grandes son del tipo de entero correcto. Por ejemplo, **UInt64(0xffff_ffff_ffff_ffff)** resulta en el valor correcto en lugar de desbordarse.

La concurrencia requiere Swift 5.9 o posterior, y una versión de la biblioteca estándar de Swift que proporcione los tipos de concurrencia correspondientes. En las plataformas de Apple, establece un target de implementación de al menos iOS 13, macOS 10.15, tvOS 13, o watchOS 6.

Un target escrito en Swift 5.9 puede depender de uno escrito en Swift 4.2 o Swift 4, y viceversa. Esto significa que si tienes un proyecto grande que está dividido en varios frameworks, podrás migrar tu código de Swift 4 a Swift 5.9 un framework a la vez.

Playgrounds

Antes de aprender el lenguaje de programación Swift, es importante aprender sobre una característica de Xcode llamada Playgrounds. Los Playgrounds son una función de Xcode que le permite aprender Swift y experimentar con el SDK de iOS. Los conceptos que se tratarán en este capítulo se pueden poner en práctica al experimentar con muchos de los ejemplos de código Swift de introducción contenidos en los capítulos siguientes.

Que es un PlayGround ?

Un playground es un entorno interactivo en el que puedes introducir y ejecutar código Swift. Los resultados de tu código se muestran en tiempo real, lo que te permite aprender la sintaxis de Swift y los aspectos visuales del desarrollo de aplicaciones iOS sin necesidad de compilar y ejecutar tu código continuamente. Los playgrounds también admiten comentarios de texto enriquecido, lo que los convierte en una buena manera de documentar tu código para futuras referencias o como herramienta de formación.

Crear un PlayGround

Para crear un nuevo Playground, inicie Xcode y seleccione la opción de menú Archivo -> Nuevo -> Playground.... Elige la opción iOS en el panel resultante y selecciona la plantilla Blank.

La plantilla Blank es útil para probar la codificación Swift. La plantilla Single View, por su parte, proporciona un entorno de controlador de vista para probar código que requiere un diseño de interfaz de usuario.

En la siguiente pantalla, asigne un nombre al playground en nuestro ejemplo le pondremos DemoSwiftPL y elija una ubicación adecuada del sistema de archivos en la que se guardará la zona de juegos antes de hacer clic en el botón Crear.

Una vez creada el playground, aparecerá la siguiente pantalla lista para introducir el código Swift:

3-1 PlayGround en Xcode

El panel situado a la izquierda de la ventana (marcado con una A en la Figura 3-1) es el panel Navegador, que permite acceder a las carpetas y archivos que componen el área de juego. Para ocultar y mostrar este panel, pulse el botón indicado por la flecha más a la izquierda. El panel central (B) es el editor del playground donde se introducen las líneas de código Swift. El panel de la derecha (C) se denomina panel de resultados y es donde se muestran los resultados de cada expresión Swift introducida en el panel del editor del playground. La barra de pestañas (D) contendrá una pestaña para cada archivo actualmente abierto en el editor del playground. Para cambiar a un archivo diferente, simplemente seleccione la pestaña correspondiente. Para cerrar un archivo abierto, sitúe el puntero del ratón sobre la pestaña y haga clic en el botón "X" cuando aparezca a la izquierda del nombre del archivo.

Un ejemplo de Swift Playground

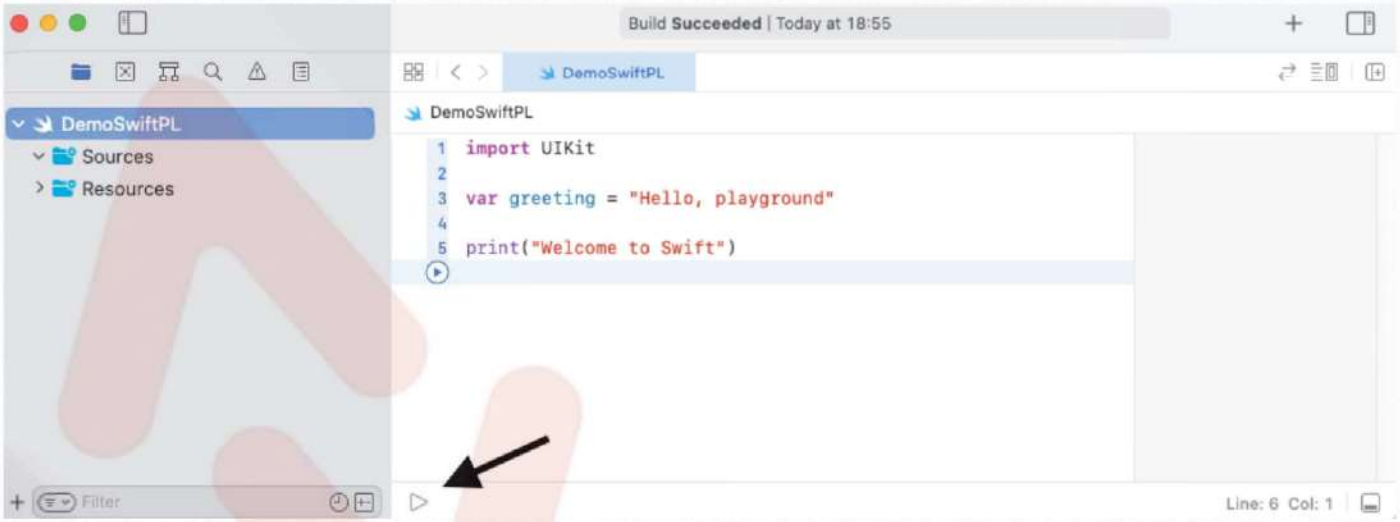
Tal vez el más simple de los ejemplos en cualquier lenguaje de programación (que al menos hace algo tangible) es escribir algo de código para dar salida a una sola línea de texto. Swift no es una excepción a esta regla así que, dentro de la ventana del playground, empieza a añadir otra línea de código Swift para que se lea como sigue:

```
import UIKit

var greeting = "Hello, playground"

print("Welcome to Swift")
```

Cuando se pulsa, este botón ejecutará todo el código de la página actual del playground desde la primera línea de código hasta la última.



3-2 Ejecución Ejemplo

Como resultado obtendremos el mensaje "Welcome to Swift" tanto en el panel de resultados como en el área de depuración

02

Fundamentos

Trabaja con tipos comunes de datos y escribe sintaxis básicas.

Tipos de colecciones

Swift proporciona tres tipos de colecciones primarias, conocidas como arrays, sets y diccionarios, para almacenar colecciones de valores. Las matrices son colecciones ordenadas de valores. Los conjuntos son colecciones desordenadas de valores únicos. Los diccionarios son colecciones desordenadas de asociaciones clave-valor.

Mutabilidad de las colecciones

La mutabilidad de las colecciones en Swift se refiere a la capacidad de modificar o alterar los elementos almacenados en una colección después de su creación. En Swift, las colecciones pueden ser mutables o inmutables, lo que determina si se pueden modificar después de su creación.

- **Colecciones Mutables**: Son aquellas que permiten cambios en sus elementos después de su inicialización. Por ejemplo, los arrays y diccionarios mutables (Array y Dictionary) pueden tener elementos agregados, eliminados o modificados después de su creación. Esto brinda flexibilidad pero requiere un manejo cuidadoso para mantener la consistencia de los datos.

- **Colecciones Inmutables**: En contraste, las colecciones inmutables no permiten cambios en sus elementos una vez que se han creado. Los arrays y diccionarios inmutables (NSArray y NSDictionary) garantizan que sus elementos no cambien después de la creación. Esto puede ser útil en situaciones donde la invariabilidad de

Arrays

Un array es una estructura de datos que almacena una colección ordenada de elementos del mismo tipo en una secuencia indexada. En Swift, los arrays permiten acceder a sus elementos mediante índices numéricos y pueden contener cualquier tipo de datos. Esto facilita la organización y manipulación de conjuntos de información de manera eficiente.

Sintaxis abreviada de tipo de matriz

La sintaxis abreviada del tipo de array en Swift permite definir y declarar arrays de manera más concisa y legible. Se utiliza mediante un par de corchetes [] con la especificación del tipo de elementos entre ellos. Por ejemplo:

```swift
// Definición de un array de enteros utilizando la sintaxis abreviada
var numbers: [Int] = [1, 2, 3, 4, 5]
```

En este caso, **[Int]** indica que numbers es un array que contiene elementos de tipo **Int**. La sintaxis abreviada es útil para declarar rápidamente arrays con tipos específicos, sin la necesidad de utilizar la palabra clave **Array** seguida de la especificación del tipo entre paréntesis angulares **< >**.

Creación de una matriz vacía

La creación de un array vacío en Swift es el proceso de inicializar un array sin ningún elemento almacenado en él. Esto proporciona una estructura vacía que puede llenarse más tarde con elementos según sea necesario.

```swift
// Crear un array vacío de cadenas
var emptyArray: [String] = []
```

En este ejemplo, **emptyArray** se declara como un array vacío de cadenas **(String).** La inicialización se realiza utilizando corchetes **[]** con la especificación del tipo **([String])** para indicar que es un array que contendrá elementos de tipo cadena. Este array está listo para recibir y almacenar elementos más adelante en el código.

09

Propiedades y Metodos

Accede a los valoes almacenados y calculados
tambien define y llama funciones

Propiedades

Las propiedades vinculan valores a una clase específica, estructura o enumeración. Las propiedades almacenadas retienen valores constantes y variables como parte integral de una instancia, mientras que las propiedades calculadas evalúan (en lugar de retener) un valor. Las clases, estructuras y enumeraciones ofrecen propiedades calculadas, mientras que las propiedades almacenadas son exclusivas de las clases y estructuras.

Las propiedades almacenadas y calculadas generalmente se vinculan a instancias de un tipo específico. No obstante, también es posible asociar propiedades al propio tipo, las cuales se conocen como propiedades de tipo. Adicionalmente, es factible establecer observadores de propiedades para supervisar los cambios en el valor de una propiedad y responder a ellos con acciones personalizadas. Los observadores de propiedades pueden ser añadidos tanto a propiedades almacenadas definidas por el usuario como a aquellas que una subclase herede de su superclase.

Stored Properties

En Swift, las "Stored Properties" (Propiedades Almacenadas) son variables o constantes que se almacenan directamente en una instancia de una clase o estructura. Estas propiedades pueden tener valores asignados que persisten a lo largo del ciclo de vida de la instancia.

```swift
struct Person {
    // Propiedades almacenadas
    var name: String
    var age: Int
}

// Crear una instancia de Person
var person1 = Person(name: "Juan", age: 25)

// Acceder y modificar propiedades almacenadas
print(person1.name)  // Imprime: Juan
print(person1.age)   // Imprime: 25

person1.age = 26     // Modificar la propiedad age
print(person1.age)   // Imprime: 26
```

Computed Properties

En Swift, las "Computed Properties" (Propiedades Calculadas) son propiedades cuyo valor no se almacena directamente, sino que se calcula mediante un bloque de código. Estos bloques de código se ejecutan cada vez que se accede a la propiedad, permitiendo realizar cálculos dinámicos en tiempo real.

```swift
struct Circle {
    // Propiedades almacenadas
    var radius: Double

    // Propiedad calculada para el área
    var area: Double {
        return Double.pi * radius * radius
    }
}

// Crear una instancia de Circle
var myCircle = Circle(radius: 3.0)

// Acceder a la propiedad calculada
print(myCircle.area)  // Imprime: 28.274333882308138

// Modificar la propiedad radius
myCircle.radius = 5.0

// La propiedad calculada se recalcula automáticamente al acceder
print(myCircle.area)  // Imprime: 78.53981633974483
```

En este ejemplo, la estructura **Circle** tiene una propiedad almacenada **radius** que representa el radio del círculo. La propiedad calculada **area** devuelve el área del círculo, calculada dinámicamente en función del radio actual. Cada vez que accedemos a la propiedad **area**, se ejecuta el bloque de código que realiza el cálculo y devuelve el resultado. Las propiedades calculadas son útiles para encapsular lógica de cálculo y proporcionar valores basados en otras propiedades.

12

Optional Chaining y Error Handling

Accediendo a los mienbros de un valor opcional y responder a los errores y recuperarse de ellos

Optional Chaining

En Swift, Optional Chaining es una característica que permite realizar una secuencia de operaciones que involucran propiedades, métodos y subíndices en una cadena de llamadas, sin necesidad de verificar explícitamente si cada paso de la cadena devuelve un valor nulo (nil).

Cuando se utiliza Optional Chaining, si alguna parte de la cadena devuelve un valor nulo, la cadena completa simplemente produce un resultado nulo sin causar un error de tiempo de ejecución. Esto facilita la escritura de código más conciso y legible al trabajar con estructuras de datos que pueden contener valores opcionales.

Esta característica es particularmente útil cuando tratamos con tipos opcionales, como aquellos que se declaran con el tipo **Optional<T>** o con el uso del símbolo "?" al definir propiedades y métodos. Optional Chaining elimina la necesidad de verificar manualmente cada nivel de la cadena para evitar la propagación de valores nulos.

En resumen, Optional Chaining en Swift proporciona una manera segura y conveniente de acceder y manipular propiedades, métodos y subíndices en una cadena de llamadas, sin necesidad de verificar explícitamente si cada paso devuelve un valor nulo. Esto mejora la legibilidad y reduce la necesidad de escribir código redundante al tratar con valores opcionales.

Defining Model Classes for Optional Chaining

Cuando trabajamos con Optional Chaining y definimos clases de modelo, estamos creando estructuras que contienen propiedades opcionales, métodos y subíndices que pueden tener valores nulos. Esto es especialmente útil cuando trabajamos con datos que pueden no estar presentes o inicializados en ciertos momentos. Al definir clases de modelo con propiedades opcionales, podemos utilizar Optional Chaining para acceder a estas propiedades de manera segura sin preocuparte por manejar manualmente los casos en los que las propiedades pueden ser nulas.

Supongamos que estamos creando una aplicación de redes sociales y queremos definir una clase **Usuario** que tenga una propiedad opcional llamada **dirección** para representar la ubicación de un usuario. La **dirección** puede no estar disponible para todos los usuarios, por lo que la definimos como opcional.

```
class Usuario {
    var nombre: String
    var dirección: Dirección?

    init(nombre: String, dirección: Dirección? = nil) {
        self.nombre = nombre
        self.dirección = dirección
    }
}

class Dirección {
    var ciudad: String
    var país: String

    init(ciudad: String, país: String) {
        self.ciudad = ciudad
        self.país = país
    }
}
```

En este ejemplo, la clase **Usuario** tiene una propiedad opcional **dirección** que representa la ubicación del usuario. La clase **Dirección** tiene propiedades **ciudad** y **país**. Al definir estas clases con propiedades opcionales, podemos utilizar Optional Chaining para acceder a la dirección de un usuario de manera segura, incluso si la dirección no está presente:

Hello Swift

nacion CODIGO