Dia 1 SwiftUI

SwiftUI es un potente marco para crear aplicaciones interactivas por el usuario para iOS, macOS, tvOS e incluso watchOS. Pero no puedes crear software sin tener un lenguaje de programación, por lo que detrás de SwiftUI se encuentra Swift: un lenguaje de programación potente, flexible y moderno que usarás para todas tus aplicaciones SwiftUI.

Como dijo Mark Twain una vez, "el secreto para salir adelante es empezar". Bueno, estás empezando ahora, así que vamos a sumergirnos y aprender sobre variables, constantes y tipos de datos simples en Swift.

Hoy tienes siete tutoriales que completar. Si quieres profundizar en cada tema, hay una lectura adicional opcional, pero no necesitas leer eso a menos que quieras y tengas tiempo. En cualquier caso, hay una serie de pruebas cortas para asegurarse de que ha entendido los conceptos clave.

Lo sé, lo sé: la tentación es fuerte de seguir viendo más vídeos y hacer más pruebas más allá de las enlazadas a continuación, pero recuerda: ¡no te apresures! Es mucho mejor hacer una hora al día todos los días que hacer trozos con grandes espacios entre ellos.

Introducción: ¿Por qué Swift?:

Hay muchos lenguajes de programación por ahí, pero creo que vas a disfrutar mucho aprendiendo Swift.
Esto es en parte por razones prácticas: ¡puedes ganar mucho dinero en la App Store!, pero también hay muchas razones técnicas.

Verás, Swift es un idioma relativamente joven, ya que solo se lanzó en 2014. Esto significa que no
tiene mucho de la cruft lingüística que puede sufrir las lenguas antiguas, y por lo general significa que solo hay una forma de resolver un problema en particular.

Al mismo tiempo, ser un lenguaje de programación tan nuevo significa que Swift aprovecha todo tipo
de ideas nuevas basadas en los éxitos, y a veces, los errores, de los lenguajes más antiguos.
Por ejemplo, hace que sea difícil escribir accidentalmente código inseguro, hace que sea muy fácil escribir código que sea claro y comprensible, y es compatible con todos los idiomas del mundo,
por lo que nunca verás esos extraños errores de caracteres que plagan otros idiomas.

Swift en sí es solo el lenguaje, y no está diseñado para dibujar nada en la pantalla. Cuando se trata
de crear software con Swift, utilizarás SwiftUI: el potente marco de Apple que crea texto, botones,
imágenes, interacción del usuario y mucho más. Como su nombre indica, SwiftUI fue creado para Swift,
está literalmente diseñado para
aprovechar el poder y la seguridad que ofrece el lenguaje, lo que hace que sea muy rápido crear aplicaciones realmente potentes.

Por lo tanto, deberías aprender Swift porque puedes ganar mucho dinero con él, pero también porque
hace muchas cosas muy bien. Sin cruft, sin confusión, solo mucha energía al alcance de la mano.
¿Qué es lo que no te gusta?

Introducción: Cómo seguir:

He estado enseñando a la gente a escribir Swift desde 2014, el mismo año en que Swift se lanzó, y en este momento Hacking with Swift es el sitio más grande del mundo dedicado a enseñar Swift.

En el camino aprendí una gran cantidad sobre qué temas son los que más importan, cómo estructurar los temas en un flujo suave y consistente y, lo que es más importante, cómo ayudar a los estudiantes a recordar los temas que han aprendido. Este curso es el producto de todo ese aprendizaje.

A diferencia de mi trabajo anterior, esto no se esfuerza por enseñarte todos los aspectos de Swift, sino que pasa más tiempo en el subconjunto de características que más importan: las que usarás en cada aplicación que crees, una y otra vez. Sí, hay algunas características avanzadas del lenguaje cubiertas, pero las he seleccionado en función de la utilidad. Cuando hayas terminado el libro, es posible que quieras seguir aprendiendo algunas de las características más avanzadas, pero sospecho que preferirías estar ocupado aprendiendo a usar SwiftUI.

Cada capítulo de este libro está disponible tanto en texto como en vídeo, pero cubren exactamente el mismo material, por lo que puedes aprender la forma que más te convenga. Si estás usando los vídeos, te darás cuenta de que a veces presento temas usando diapositivas y a veces los demuestro en Xcode. Puede que se sienta repetitivo, pero es intencional: hay muchas cosas que aprender, y si vieras cada una solo una vez, ¡no se quedaría en tu memoria!

Hay una última cosa: es posible que notes cuántos capítulos comienzan con "Cómo...", y eso es intencional: este libro está aquí para mostrarte cómo hacer las cosas de una manera práctica, en lugar de profundizar en la teoría. La teoría es importante, y te encontrarás con mucho de ella a medida que puedas seguir aprendiendo, pero aquí el enfoque es implacablemente práctico porque creo que la mejor manera de aprender algo nuevo es probarlo tú mismo.

La programación es un arte: no pases todo el tiempo afilando tu lápiz cuando deberías estar dibujando.

Cómo crear variables y constantes:

Cada vez que crees programas, vas a querer almacenar algunos datos. Tal vez sea el nombre del usuario que acaban de escribir, tal vez sean algunas noticias que descargaste de Internet, o tal vez sea el resultado de un cálculo complejo que acabas de realizar.

Swift nos da dos formas de almacenar datos, dependiendo de si quieres que los datos cambien con el tiempo.
La primera opción se utiliza automáticamente cuando se crea un nuevo patio de recreo, porque contendrá esta línea:

```
var greeting = "Hello, playground"
```

Eso crea una nueva variable llamada greeting, y debido a que es una variable, su valor puede variar, puede cambiar a medida que se ejecuta nuestro programa.

Consejo: La otra línea en un patio de recreo de macOS es la import Cocoa, que trae una enorme colección de código proporcionado por Apple para facilitar la creación de aplicaciones. Esto incluye muchas funciones importantes, así que no la elimines.

Realmente hay cuatro piezas de sintaxis ahí:

-. La palabra clave var significa "crear una nueva variable"; ahorra un poco de escritura.

-. Estamos llamando a nuestro saludo variable. Puedes llamar a tu variable como quieras, pero la mayoría de las veces querrás hacerla descriptiva.

-. El signo de igual asigna un valor a nuestra variable. No necesitas tener esos espacios a cada lado del signo de igualdad si no quieres, pero es el estilo más común.

-. El valor que estamos asignando es el texto "Hola, patio de recreo". Tenga en cuenta que el texto está escrito entre comillas dobles, para que Swift pueda ver dónde comienza el texto y dónde termina.

Si has usado otros idiomas, es posible que hayas notado que nuestro código no necesita un punto y coma al final de la línea. Swift permite puntos y comas, pero son muy raros: solo los necesitarás si quieres escribir dos piezas de código en la misma línea por alguna razón.

Cuando haces una variable, puedes cambiarla con el tiempo:

```
var name = "Ted"
name = "Rebecca"
name = "Keeley"
```

Eso crea una nueva variable llamada nombre, y le da el valor "Ted". Luego se cambia dos veces, primero a "Rebecca" y luego a "Keeley" - no volvemos a usar var porque estamos modificando una variable existente en lugar de crear una nueva. Puedes cambiar las variables tanto como necesites, y el valor antiguo se descarta cada vez.

(Eres bienvenido a poner un texto diferente en tus variables, pero soy un gran fan del programa de televisión

Ted Lasso, así que fui con Ted. Y sí, puedes esperar otras referencias de Ted Lasso y más en los siguientes capítulos.)

Si nunca quieres cambiar un valor, necesitas usar una constante en su lugar. La creación de una constante funciona
casi de manera idéntica a la creación de una variable, excepto que usamos let en lugar de var, así:

```
let character = "Daphne"
```

Ahora, cuando usamos let, hacemos una constante, que es un valor que no puede cambiar. Swift literalmente no nos
deja, y mostrará un gran error si lo intentamos.

¿No me crees? Intenta poner esto en Xcode:

```
let character = "Daphne"
character = "Eloise"
character = "Francesca"
```

Una vez más, no hay palabras clave en esas líneas segunda y tercera porque no estamos creando nuevas
Constantes, solo estamos tratando de cambiar la que ya tenemos. Sin embargo, como dije, eso no funcionará,
no puedes cambiar una constante, ¡de lo contrario no sería constante!

Si tenías curiosidad, "deja que" venga del mundo de las matemáticas, donde dicen cosas como "deja que x sea igual a 5".

Importante: Por favor, elimine las dos líneas de código que muestran errores, ¡realmente no puede cambiar las constantes!

Cuando estés aprendiendo Swift, puedes pedirle a Xcode que imprima el valor de cualquier variable. No usarás
esto mucho en aplicaciones reales porque los usuarios no pueden ver lo que está impreso, pero es muy útil como una
forma sencilla de ver lo que hay dentro de tus datos.

Por ejemplo, podríamos imprimir el valor de una variable cada vez que se establezca. Intenta introducir esto en tu patio de recreo:

```
var playerName = "Roy"
print(playerName)

playerName = "Dani"
print(playerName)

playerName = "Sam"
print(playerName)
```

Consejo: Puedes ejecutar código en tu patio de recreo de Xcode haciendo clic en el icono de reproducción azul a la
izquierda. Si te mueves hacia arriba o hacia abajo a lo largo de esa franja azul, verás que el icono de reproducción
también se mueve - esto te permite ejecutar el código hasta cierto punto si quieres, pero la mayoría de las veces aquí
querrás correr hasta la última línea.

Es posible que te hayas dado cuenta de que llamé a mi variable playerName, y no a playername, player_name o alguna otra
alternativa. Esta es una opción: a Swift realmente no le importa cómo llames a tus constantes y variables, siempre y cuando
te refieras a ellas de la misma manera en todas partes. Por lo tanto, no puedo usar el nombre del jugador primero y luego

el nombre del jugador más tarde. Swift ve a esos dos como nombres diferentes.

Aunque a Swift no le importa cómo nombramos nuestros datos, el estilo de nomenclatura que estoy usando es el estándar entre
los desarrolladores de Swift, lo que llamamos una convención. Si tienes curiosidad, el estilo se llama "caso de camello",
porque la segunda y las palabras posteriores de un nombre comienzan con un pequeño golpe para la letra mayúscula:

```
let managerName = "Michael Scott"
let dogBreed = "Samoyed"
let meaningOfLife = "How many roads must a man walk down?"
```

Si puede, prefiera usar constantes en lugar de variables: no solo le da a Swift la oportunidad de optimizar su código un
poco mejor, sino que también le permite a Swift asegurarse de que nunca cambie el valor de una constante por accidente.

¿Por qué Swift tiene variables?

Las variables nos permiten almacenar información temporal en nuestro programa y forman una parte clave de casi todos los
programas de Swift. En última instancia, tu programa va a transformar los datos de alguna manera: tal vez dejes que el
usuario ingrese en las tareas de la lista de tareas pendientes y luego las marques, tal vez los dejes vagar por una isla
desierta trabajando para un mapache capitalista, o tal vez leas la hora del dispositivo y la muestres en un reloj.
En cualquier caso, estás tomando algún tipo de datos, transformándolos de alguna manera y mostrándolos al usuario.

Por supuesto, el "transformarlo de alguna manera" es donde entra la verdadera magia, porque esa es la parte en la que
ocurre tu brillante idea de aplicación. Pero el proceso de almacenar datos en la memoria, aferrarse a algo que el usuario
escribió o algo que descargó de Internet, es donde entran en maden las variables.

Una vez que crees una variable usando var, puedes cambiarla tantas veces como quieras sin volver a usar var. Por ejemplo:

```
var favoriteShow = "Orange is the New Black"
favoriteShow = "The Good Place"
favoriteShow = "Doctor Who"
```

Si ayuda, intenta leer var como "crear una nueva variable". Por lo tanto, la primera línea anterior podría leerse en voz
alta como "crear una nueva variable llamada favoriteShow y darle el valor Orange is the New Black". Las líneas 2 y 3 no
tienen var allí, por lo que modifican el valor existente en lugar de crear una nueva variable.

Ahora imagina que tenías var en las tres líneas: usaste var favoriteShow cada vez. Eso no tendría mucho sentido, porque
estarías diciendo "crear una nueva variable llamada favoriteShow" tres veces, y la variable claramente no es nueva
después de tu primer intento. Swift marcará esto como un error, lo que significa que no te permitirá ejecutar tu código
hasta que elijas un nombre diferente para tus variables.

Eso puede parecer un comportamiento molesto, pero créeme: ¡es útil! Swift quiere que seas claro: ¿estás tratando de
modificar una variable existente (si es así, elimina la var la segunda y las siguientes veces), o estás tratando de

crear una nueva variable (en cuyo caso, nómbrala de otra manera).

Una última cosa: aunque las variables forman el núcleo de muchos programas Swift, aprenderás que a veces es mejor evitarlas.
¡Más sobre eso más adelante!

¿Por qué Swift tiene constantes y variables?

Las variables son una excelente manera de almacenar datos temporales en sus programas, pero Swift nos da una segunda
opción que es aún mejor: las constantes. Son idénticos a las variables en todos los sentidos, con una diferencia importante:
no podemos cambiar sus valores una vez que se establecen.

A Swift le encantan las constantes y, de hecho, te recomendará que uses una si creaste una variable y luego nunca cambiaste
su valor. La razón de esto se trata de evitar problemas: cualquier variable que crees puede ser cambiada por ti cuando quieras
y con la frecuencia que quieras, por lo que pierdes algo de control: esa pieza importante de los datos del usuario que
ocordaste podría ser eliminada o reemplazada en cualquier momento en el futuro.

Las constantes no nos permiten cambiar los valores una vez que se establecen, por lo que es un poco como un contrato con Swift:
estás diciendo "este valor importa, no me dejes cambiarlo sin importar lo que haga". Claro, podrías intentar hacer el mismo
contrato con una variable, pero un deslizamiento de tu teclado podría arruinar las cosas y Swift no podría ayudar.
Al usar una constante en su lugar, solo cambiando var para dejar, le estás pidiendo a Swift que se asegure de que el valor nunca
cambie, que es otra cosa de la que ya no tienes que preocuparte.

Cómo crear cadenas:

Cuando asignas texto a una constante o variable, llamamos a eso una cadena. Piensa en un montón de azulejos de Scrabble
enhesillados en una cadena para hacer una palabra.

Las cuerdas de Swift comienzan y terminan con comillas dobles, pero lo que pones dentro de esas citas es para ti.
Puedes usar piezas cortas de texto alfabético, como esta:

```
let actor = "Denzel Washington"
```

Puedes usar puntuación, emojis y otros personajes, como este:

```
let filename = "paris.jpg"
let result = "⭐ You win! ⭐"
```

e incluso puedes usar otras comillas dobles dentro de tu cuerda, siempre y cuando tengas cuidado de poner una barra de espaldas
ante ellos para que Swift entienda que están dentro de la cuerda en lugar de terminar la cuerda:

```
let quote = "Then he tapped a sign saying \"Believe\" and walked away."
```

No te preocupes si te pierdes la reacción, Swift se asegurará de gritar fuerte que tu código no es muy correcto.

No hay un límite realista a la longitud de sus cuerdas, lo que significa que podría utilizar una cuerda para almacenar algo muy
largo, como las obras completas de Shakespeare. Sin embargo, lo que encontrarás es que a Swift no le rompo la línea en sus cuerdas.
Por lo tanto, este tipo de código no está permitido:

```
let movie = "A day in
the life of an
Apple engineer"
```

Eso no significa que no puedes crear cadenas a través de múltiples líneas, sólo que Swift necesita que las trates especialmente:
en lugar de un conjunto de comillas a cada lado de tu cadena, usas tres, así:

Estas cuerdas multilínea no se usan terriblemente a menudo, pero al menos se puede ver cómo se hace: las triples citas al principio
y al final están en su propia línea, con su cadena en el medio.

Una vez que hayas creado tu cuerda, encontrarás que Swift nos da una funcionalidad útil para trabajar con su contenido. Apredará
más sobre esta funcionalidad con el tiempo, pero quiero presentarles tres cosas aquí.

Primero, puedes leer la longitud de una cadena escribiendo .countdespués del nombre de la variable o constante:

```
print(actor.count)
```

Porque actortiene el texto de "Denzel Washington", que imprimirá 17 1 por cada letra en el nombre, más el espacio en el medio.

No es necesario imprimir la longitud de una cadena directamente si no quieres asignarlo a otra constante, como esta:

```
let nameLength = actor.count
print(nameLength)
```

La segunda pieza útil de la funcionalidad es uppercased(), que envía de vuelta la misma cuerda excepto cada una de sus letras
está en la parte superior:

```
print(result.uppercased())
```

Sí, los paréntesis abiertos y cercanos son necesarios aquí, pero no se necesitan con count. La razón de esto se aclarará a medida
que aprendas, pero en esta etapa temprana de tus Aplicaciones aprender la distinción se explica mejor como esta: si le pides a Swift
que lea algunos datos no necesitas los paréntesis, pero si le pides a Swift que hagas algún trabajo.do Eso no es del todo cierto,
ya que aprenderás más tarde, pero es suficiente para hacerte avanzar por ahora.

La última pieza de funcionalidad de cuerda útil se llama hasPrefix(), y nos permite saber si una cuerda comienza con algunas letras
de nuestra elección:

```
print(movie.hasPrefix("A day"))
```

Ahí también un hasSuffix()contraparte, que comprueba si una cadena termina con algún texto:

```
print(filename.hasSuffix(".jpg"))
```

Importante: Las cuerdas son sensibles a los casos en Swift, que significa el uso filename.hasSuffix(".JPG")volverán falsa porque
las letras de la cuerda son minúsculas.

Las cuerdas son realmente poderosas en Swift, y sólo hemos arañado realmente la superficie de lo que pueden hacer.

¿Por qué Swift necesita cadenas de varias líneas?

Las cadenas estándar de Swift comienzan y terminan con comillas, pero nunca deben contener ningún salto de línea. Por ejemplo,
esta es una cadena estándar:

```
var quote = "Change the world by being yourself"
```

Eso funciona bien para trozos pequeños de texto, pero se vuelve feo en el código fuente si tienes mucho texto que quieres almacenar.
Ahí es donde entran en las cadenas de varias líneas: si usas comillas triples, puedes escribir tus cadenas en tantas líneas como necesites,
lo que significa que el texto sigue siendo fácil de leer en tu código:

```
var burns = """
The best laid schemes
O’ mice and men
Gang aft agley
"""
```

Swift ve esos saltos de línea en su cadena como parte del texto en sí, por lo que esa cadena en realidad contiene tres líneas.

Consejo: A veces querrás tener largas cadenas de texto en tu código sin usar varias líneas, pero esto es bastante raro. Específicamente,
esto es más comúnmente importante si planea compartir su código con otros: si ven un mensaje de error en su programa, es posible que
quieran buscarlo en su código, y si lo ha dividido en varias líneas, su búsqueda podría fallar.

Cómo almacenar números enteros:

Cuando trabajas con números enteros como 3, 5, 50 o 5 millones, estás trabajando con lo que Swift llama enteros, o Int para abreviar -
"entero" es originalmente una palabra latina que significa "entero", si tenías curiosidad.

Hacer un nuevo entero funciona como hacer una cadena: usa let o var dependiendo de si quieres una constante o una variable, proporciona
un nombre y luego dale un valor. Por ejemplo, podríamos crear una constante de puntuación como esta:

```
let score = 10
```

Los números enteros pueden ser muy grandes: más allá de los miles de millones, los más allá de los billones, los cuatrillones pasados
y más allá de los quintillones, pero también pueden ser muy pequeños, pueden contener números negativos hasta los quintillones.

Cuando escribes números a mano, puede ser difícil ver lo que está pasando. Por ejemplo, ¿qué número es este?

```
let reallyBig = 100000000
```

Si estuviéramos escribiendo eso a mano, probablemente escribiríamos "100.000.000", momento en el que está claro que el número es de
100 millones. Swift tiene algo similar: puedes usar guiones bajos, _, para dividir los números como quieras.

Por lo tanto, podríamos cambiar nuestro código anterior a esto:

```
let reallyBig = 100_000_000
```

A Swift en realidad no le importan los guiones bajos, así que si quisieras, podrías escribir esto en su lugar:

```
let reallyBig = 1_00__00___00____00
```

El resultado final es el mismo: reallyBig se establece en un entero con el valor de 100.000.000.

Por supuesto, también puedes crear enteros a partir de otros números enteros, utilizando los tipos de operadores aritméticos que
aprendiste en la escuela: + para la suma, - para la resta, * para la multiplicación y / para la división.

Por ejemplo:

```
let lowerScore = score - 2
let higherScore = score + 10
let doubledScore = score * 2
let squaredScore = score * score
let halvedScore = score / 2
print(score)
```

En lugar de hacer nuevas constantes cada vez, Swift tiene algunas operaciones especiales que ajustan un entero de alguna manera y
asignan el resultado al número original.

Por ejemplo, esto crea una variable de contador igual a 10, y luego le añade 5 más:

```
var counter = 10
counter = counter + 5
```

En lugar de escribir counter = counter + 5, puede usar el operador abreviado +=, que agrega un número directamente al
número entero en cuestión:

```
counter += 5
print(counter)
```

Eso hace exactamente lo mismo, solo que con menos escritura. Llamamos a estos operadores de asignación de compuestos,
y vienen en otras formas:

```
counter *= 2
print(counter)
counter -= 10
print(counter)
counter /= 2
print(counter)
```

Antes de que terminemos con los números enteros, quiero mencionar una última cosa: al igual que las cadenas, los enteros tienen
alguna funcionalidad útil adjunta. Por ejemplo, puede llamar a isMultiple(of:) en un entero para averiguar
si es un múltiplo de otro entero.

Por lo tanto, podríamos preguntarnos si 120 es un múltiplo de tres como este:

```
let number = 120
print(number.isMultiple(of: 3))
```

Estoy llamando a isMultiple(of:) en una constante allí, pero puedes usar el número directamente si quieres:

```
print(120.isMultiple(of: 3))
```

Cómo almacenar números decimales:

Cuando trabajas con números decimales como 3.1, 5.56 o 3.141592654, estás trabajando con lo que Swift llama números de coma flotante.
El nombre proviene de la forma sorprendentemente compleja en que su computadora almacena los números: intenta almacenar números muy
grandes, como 123,456,789, en la misma cantidad de espacio que los números muy pequeños, como 0,0000000001, y la única forma en que
puede hacerlo es moviendo el punto decimal en función del tamaño del número.

Este método de almacenamiento hace que los números decimales sean notoriamente problemáticos para los programadores, y puedes probar
esto con solo dos líneas de código Swift:

```
let number = 0.1 + 0.2
print(number)
```

Cuando eso se ejecuta, no imprimirá 0.3. En su lugar, imprimirá 0.300000000000000 - ese 0,3, luego 15 ceros, luego un 4 porque... bueno,
como dije, es complejo.

Explicaré más por qué es complejo en un momento, pero primero centrémonos en lo que importa.

En primer lugar, cuando creas un número de coma flotante, Swift lo considera un Doble. Esa es la abreviatura de "número de coma flotante
de doble precisión", que me doy cuenta de que es un nombre bastante extraño: la forma en que hemos manejado los números de coma flotante
ha cambiado mucho a lo largo de los años, y aunque Swift hace un buen trabajo al simplificar esto, a veces puede encontrar algún código
más antiguo que es más complejo. En este caso, significa que Swift asigna el doble de la cantidad de almacenamiento que algunos idiomas
más antiguos, lo que significa que un doble puede almacenar números absolutamente masivos.

En segundo lugar, Swift considera que los decimales son un tipo de datos completamente diferente a los enteros, lo que significa que no
puedes mezclarlos. Después de todo, los números enteros siempre son 100 % precisos, mientras que los decimales no lo son, por lo que Swift
no te deja juntar los dos a menos que lo pidas específicamente.

En la práctica, esto significa que no puedes hacer cosas como agregar un entero a un decimal, por lo que este tipo de código producirá un error:

```
let a = 1
let b = 2.0
let c = a + b
```

Sí, podemos ver que b es realmente solo el entero 2 disfrazado de decimal, pero Swift todavía no permitirá que se ejecute ese código.
Esto se llama seguridad de tipo: Swift no nos permite mezclar diferentes tipos de datos por accidente.

Si quieres que eso suceda, tienes que decirle explícitamente a Swift que debe tratar el Doble dentro de b como un Int:

```
let c = a + Int(b)
```

O trata el Int dentro de a como un Doble:

En tercer lugar, Swift decide si quieres crear un Doble o un Int en función del número que proporciones; si hay un punto allí, tienes un
Doble, de lo contrario es un Int. Sí, incluso si los números después del dos son 0.

Así que:

```
let double1 = 3.1
let double2 = 3131.3131
let double3 = 3.0
let int1 = 3
```

Combinado con la seguridad del tipo, esto significa que una vez que Swift ha decidido qué tipo de datos tiene una constante o variable,
siempre debe contener ese mismo tipo de datos. Eso significa que este código está bien:

```
var name = "Nicolas Cage"
name = "John Travolta"
```

Pero este tipo de código no es:

```
var name = "Nicolas Cage"
name = 57
```

Eso dice que el nombre de Swift almacenará una cadena, pero luego intenta poner un entero allí en su lugar.

Por último, los números decimales tienen el mismo rango de operadores y operadores de asignación compuesta que los enteros:

```
var rating = 5.0
rating *= 2
```

Muchas API más antiguas utilizan una forma ligeramente diferente de almacenar números decimales, llamada CGFloat. Afortunadamente,
Swift nos permite usar números dobles regulares en todas partes donde se espera un CGFloat, por lo que aunque verás que CGFloat
aparece de vez en cuando, puedes ignorarlo.

En caso de que tuvieras curiosidad, la razón por la que los números de coma flotante son complejos es porque los ordenadores están
tratando de usar el binario para almacenar números complicados. Por ejemplo, si divides 1 por 3, sabemos que obtienes 1/3, pero eso
no se puede almacenar en binario, por lo que el sistema está diseñado para crear aproximaciones muy cercanas.
Es extremadamente eficiente, y el error es tan pequeño que suele ser irrelevante, ¡pero al menos sabes por qué Swift no nos deja
mezclar Int y Double por accidente!

¿Por qué Swift necesita tanto dobles como enteros?:

Swift nos da varias formas diferentes de almacenar números en nuestro código, y están diseñadas para resolver diferentes problemas.
Swift no nos deja mezclarlos porque hacerlo (como en, 100% garantizado) conducirá a problemas.

Los dos tipos principales de números que usarás se llaman enteros y dobles. Los números enteros tienen números enteros, como 0, 1,
-100 y 65 millones, mientras que los dobles tienen números decimales, como 0,1, -1.001 y 3.141592654.

Al crear una variable numérica, Swift decide si la considera un número entero o un doble en función de si se incluye un punto decimal.
Por ejemplo:

```
var myInt = 1
var myDouble = 1.0
```

Como puedes ver, ambos contienen el número 1, pero el primero es un número entero y el segundo un doble.

Ahora, si ambos contienen el número 1, podrías preguntarte por qué no podemos sumarlos juntos, ¿por qué no podemos escribir
var total = myInt + myDouble? La respuesta es que Swift está jugando a lo seguro: ambos podemos ver que 1 más 1.0 será 2, pero tu doble
es una variable, por lo que podría modificarse para que sea 1.1 o 3.5 o algo más. ¿Cómo puede Swift asegurarse de que es seguro agregar
un entero a un doble? ¿Cómo puede estar seguro de que no perderá el 0,1 o el 0,5?

La respuesta es que no puede ser seguro, por lo que no está permitido. Esto te molestará al principio, pero confía en mí: es útil.

¿Por qué Swift es un lenguaje seguro para escribir?:

Swift nos permite crear variables como cadenas y enteros, pero también muchos otros tipos de datos. Cuando creas una variable,
Swift puede averiguar qué tipo es la variable en función del tipo de datos que le asignas, y a partir de entonces esa variable
siempre tendrá ese tipo específico.

Por ejemplo, esto crea una nueva variable llamada meaningOfLife igual a 42:

```
var meaningOfLife = 42
```

Debido a que asignamos 42 como el valor inicial de meaningOfLife, Swift le asignará el tipo entero: un número entero. Es una variable,
lo que significa que podemos cambiar su valor tan a menudo como sea necesario, pero no podemos cambiar su tipo: siempre será un entero.

Esto es extremadamente útil al crear aplicaciones, porque significa que Swift se asegurará de que no cometamos errores con nuestros datos.
Por ejemplo, no podemos escribir esto:

```
meaningOfLife = "Forty two"
```

Eso intenta asignar una cadena a una variable que es un entero, lo cual no está permitido. Aunque rara vez cometemos un error tan obvio,
encontrarás que este tipo de seguridad ayuda todos los días que escribes código con Swift.

Piénsalo: acabamos de crear una variable y luego intentamos cambiar su tipo, lo que obviamente fallará. Pero a medida que sus programas
crecen en tamaño y complejidad, se vuelve imposible mantener los tipos de sus variables en su cabeza en todo momento, por lo que estamos
cambiando efectivamente ese trabajo a Swift en su lugar.