Tipos de datos simples, parte 2

La ley de Lynch dice que "cuando las cosas se ponen difíciles, todo el mundo se va". Cualquiera puede completar el primer día de un curso, pero se necesita diligencia para volver para el día 2, ¡bien hecho por seguir con él!

Ayer empezamos a buscar tipos de datos simples, cosas que tienen un valor, como un solo número o una sola cadena de letras. Hoy continuamos con eso mientras exploramos el almacenamiento de la verdad con booleanos y la construcción de cadenas con la interpolación. En varios puntos podrías pensar "¿realmente necesito saber esto?" Y la respuesta es sí: si está aquí en este curso, ¡es absolutamente necesario saberlo!

Pero hoy sucede algo importante, porque también vas a tener tu primer punto de control. Aquí es donde vamos a hacer una pausa para que puedas escribir tu propio código, para asegurarte de que has entendido completamente lo que se presentó. Comenzar con un lienzo en blanco va a ser difícil al principio, pero deberías tener mucho tiempo y yo también te daré pistas.

Hoy tienes dos nuevos tutoriales a seguir, además de un resumen y un punto de control que completar. Como antes, si quieres profundizar en cada tema, hay alguna lectura adicional opcional, pero no necesitas leer eso a menos que quieras.

1. Cómo almacenar la verdad con booleanos

Hasta ahora hemos visto cadenas, enteros y decimales, pero hay un cuarto tipo de datos que se colan al mismo tiempo: un tipo muy simple llamado Booleano, que almacena verdadero o falso. Si tenías curiosidad, los booleanos llevan el nombre de George Boole, un matemático inglés que pasó mucho tiempo investigando y escribiendo sobre lógica.

Digo que los booleanos se colaron porque ya los has visto un par de veces

```
let filename = "paris.jpg"
print(filename.hasSuffix(".jpg"))

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

Tanto hasSuffix() como isMultiple(of:) devuelven un nuevo valor basado en su comprobación: o la cadena tiene el sufijo o no, y 120 es un múltiplo de 3 o no lo es. En ambos lugares siempre hay una simple respuesta verdadera o falsa, que es donde entran en tran los booleanos: almacenan solo eso, y nada más.

Hacer un booleano es como hacer los otros tipos de datos, excepto que debe asignar un valor inicial de verdadero o falso, como este:

```
let goodDogs = true
let gameOver = false
```

También puede asignar el valor inicial de un booleano de algún otro código, siempre y cuando en última instancia sea verdadero o falso:

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

A diferencia de los otros tipos de datos, los booleanos no tienen operadores aritméticos como + y - - después de todo, ¿qué sería verdadero + verdadero igual? Sin embargo, los booleanos tienen un operador especial, !, que significa "no". Esto cambia el valor de un booleano de verdadero a falso, o falso a verdadero.

Por ejemplo, podríamos voltear el valor de un booleano de esta manera:

```
var isAuthenticated = false
isAuthenticated = !isAuthenticated
print(isAuthenticated)
isAuthenticated = !isAuthenticated
print(isAuthenticated)
```

Eso imprimirá "verdadero" y luego "falso" cuando se ejecute, porque isAuthenticated comenzó como falso, y lo establecemos en no falso, lo cual es cierto, luego lo volteamos de nuevo para que vuelva a ser falso.

Los booleanos tienen un poco de funcionalidad adicional que puede ser útil. En particular, si llamas a toggle() en un booleano, cambiará un valor verdadero a falso y un valor falso a verdadero. Para probar esto, intenta hacer de gameOver una variable y modificarla de esta manera:

```
var gameOver = false
print(gameOver)

gameOver.toggle()
print(gameOver)
```

Eso imprimirá falso primero, luego después de llamar a toggle() se imprimirá verdadero. ¡Sí, eso es lo mismo que usar! Solo en un poco menos de código, ¡pero es sorprendentemente útil cuando se trata de código complejo!

2. Cómo unir las cuerdas

Swift nos da dos formas de combinar cuerdas: unirlas usando +, y una técnica especial llamada interpolación de cuerdas que puede colocar variables de cualquier tipo directamente dentro de las cuerdas.

Comencemos primero con la opción más fácil, que es usar + para unir cadenas: cuando tienes dos cuerdas, puedes unirlas en una nueva cadena con solo usar +, así:

```
let firstPart = "Hello, "
let secondPart = "world!"
let greeting = firstPart + secondPart
```

Puedes hacer esto muchas veces si lo necesitas:

```
let people = "Haters"
let action = "hate"
let lyric = people + " gonna " + action
print(lyric)
```

Cuando eso se ejecute, imprimirá "Haters gonna hate" - sí, soy un gran fan de Taylor Swift, ¡y creo que sus letras son un ajuste natural para un tutorial sobre programación de Swift!

¿Notas cómo estamos usando + para unir dos cadenas, pero cuando usamos Int y Double añadimos números juntos? Esto se llama sobrecarga del operador: la capacidad de un operador, como +, de significar cosas diferentes dependiendo de cómo se use. Para las cadenas, también se aplica a +=, que añade una cadena directamente a otra.

Esta técnica funciona muy bien para cosas pequeñas, pero no te gustaría hacerlo demasiado. Verás, cada vez que Swift ve que se unen dos cuerdas usando +, tiene que hacer una nueva cadena de ellas antes de continuar, y si tienes muchas cosas unidas, es bastante derrochador.

Piensa en esto, por ejemplo:

```
let luggageCode = "1" + "2" + "3" + "4" + "5"
```

Swift no puede unir todas esas cuerdas de una sola vez. En su lugar, se unirá a los dos primeros para hacer "12", luego se unirá a "12" y "3" para hacer "123", luego se unirá a "123" y "4" para hacer "1234", y finalmente se unirá a "1234" y "5" para hacer "12345" - hace cadenas temporales para sostener "12", "123" y "1234" a pesar de que no se utilizan en última instancia cuando el código termina.

Swift tiene una mejor solución llamada interpolación de cadenas, y nos permite crear cadenas de manera eficiente a partir de otras cadenas, pero también a partir de enteros, números decimales y más.

Si te acuerdas, antes dije que puedes incluir comillas dobles dentro de las cadenas, siempre y cuando tengan una barra invertida delante de ellas para que Swift sepa tratarlas especialmente:

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

Se utiliza algo muy similar con la interpolación de cadenas: escribes una barra invertida dentro de tu cadena, luego colocas el nombre de una variable o constante dentro de los paréntesis.

Por ejemplo, podríamos crear una constante de cadena y una constante entera, y luego combinarlas en una nueva cadena:

```
let name = "Taylor"
let age = 26
let message = "Hello, my name is \(name) and I'm \(age) years old."
print(message)
```

Cuando se ejecute ese código, se imprimirá "Hola, me llamo Taylor y tengo 26 años".

La interpolación de cadenas es mucho más eficiente que usar + para unir cadenas una por una, pero también hay otro beneficio importante: puedes extraer enteros, decimales y más sin trabajo adicional.

Verás, el uso de + nos permite agregar cadenas a cadenas, enteros a enteros y decimales a decimales, pero no nos permite agregar enteros a cadenas. Por lo tanto, este tipo de código no está permitido:

```
let number = 11
let missionMessage = "Apollo " + number + " landed on the moon."
```

Podrías pedirle a Swift que trate el número como una cadena si quisieras, así:

```
let missionMessage = "Apollo " + String(number) + " landed on the moon."
```

Todavía es más rápido y más fácil de leer usar la interpolación de cadenas:

```
let missionMessage = "Apollo \(number) landed on the moon."
```

Consejo: Puedes poner cálculos dentro de la interpolación de cadenas si quieres. Por ejemplo, esto imprimirá "5 x 5 es 25":

```
print("5 x 5 is \(5 * 5)")
```

¿Por qué Swift tiene interpolación de cuerdas?

Actualizado para Xcode 15

Cuando llegue el momento de mostrar información a su usuario, ya sean mensajes que se impriman, texto en los botones o lo que sea que se ajuste a la idea de su aplicación, por lo general dependerá en gran medida de las cadenas.

Por supuesto, no solo queremos cadenas estáticas, porque queremos mostrar al usuario algún tipo de datos relevantes que pueda usar. Por lo tanto, Swift nos da la interpolación de cadenas como una forma de inyectar datos personalizados en cadenas en tiempo de ejecución: reemplaza una o más partes de una cadena con datos proporcionados por nosotros.

Por ejemplo:

```
var city = "Cardiff"
var message = "Welcome to \(city)!"
```

Por supuesto, en ese ejemplo trivial podríamos haber escrito el nombre de nuestra ciudad directamente en la cadena - "¡Bienvenido a Cardiff!" - pero en aplicaciones reales tener esto insertado dinámicamente es importante porque nos permite mostrar datos de usuario del mundo real en lugar de cosas que escribimos nosotros mismos.

Swift es capaz de colocar cualquier tipo de datos dentro de la interpolación de cadenas. Puede que el resultado no siempre sea útil, pero para todos los tipos básicos de Swift (cadenas, enteros, booleanos, etc.) los resultados son excelentes.

Consejo: La interpolación de cadenas es extremadamente poderosa en Swift. Si quieres ver lo que puede hacer, echa un vistazo a esta publicación de blog más avanzada de mi parte: https://www.hackingwithswift.com/articles/178/super-powered-string-interpolation-in-swift-5-0

3. Resumen: Datos simples

Hemos cubierto mucho sobre los conceptos básicos de los datos en los capítulos anteriores, así que recapitulemos:

Swift nos permite crear constantes usando let y variables usando var.

Si no tienes la intención de cambiar un valor, asegúrate de usar let para que Swift pueda ayudarte a evitar errores.

Las cadenas de Swift contienen texto, desde cuerdas cortas hasta novelas enteras. Funcionan muy bien con emojis y cualquier idioma del mundo, y tienen una funcionalidad útil como count y uppercased().

Creas cadenas usando comillas dobles al principio y al final, pero si quieres que tu cadena supere varias líneas, necesitas usar tres comillas dobles al principio y al final.

Swift llama a sus números enteros enteros, y pueden ser positivos o negativos. También tienen una funcionalidad útil, como isMultiple(of:).

En Swift, los números decimales se llaman doble, abreviatura de número de coma flotante de doble longitud. Eso significa que pueden contener números muy grandes si es necesario, pero tampoco son 100 % precisos; no debe usarlos cuando se requiere un 100 % de precisión, como cuando se trata de dinero.

Hay muchos operadores aritméticos incorporados, como +, -, * y /, junto con los operadores especiales de asignación de compuestos como += que modifican las variables directamente.

Puedes representar un estado simple verdadero o falso usando un booleano, que se puede voltear usando el ! Operador o llamando a toggle().

La interpolación de cadenas nos permite colocar constantes y variables en nuestras cadenas de una manera simplificada y eficiente.

Es mucho, ¿verdad? Y eso está bien, usarás todo lo de esa lista una y otra vez a medida que creas aplicaciones, hasta que finalmente lo entenderás todo sin necesidad de volver aquí.

Punto de control 1

Ya sabes lo suficiente como para empezar a escribir tu primer código útil, aunque bastante simple: vamos a convertir las temperaturas de Celsius a Fahrenheit.

Tu objetivo es escribir un patio de recreo de Swift que:

Crea una constante que mantiene cualquier temperatura en Celsius.

Lo convierte en Fahrenheit multiplicando por 9, dividiendo por 5 y luego sumando 32.

Imprime el resultado para el usuario, mostrando tanto los valores Celsius como Fahrenheit.

Ya sabes todo lo que necesitas para resolver ese problema, pero si quieres algunas pistas, añadiré algunas a continuación.

Nota: Realmente te animo a que intentes construir este patio de recreo antes de leer cualquier pista o probar mi solución. Sé que puede parecer simple, pero el curso comienza a ser más difícil pronto y es importante estar seguro de que has aprendido todos los fundamentos.

Por favor, adelante e intenta construir el patio de recreo ahora.

¿Sigues aquí? Vale, aquí hay algunas pistas:

1. Usa let para hacer tu constante. Puedes llamarlo como quieras, pero creo que Celsius sería un nombre apropiado.

2. Celsius se almacena comúnmente como un decimal, así que asegúrate de crearlo como uno. Esto podría significar agregar ".0" al final, usando 25.0 en lugar de 25, por ejemplo.

3. Usamos * para la multiplicación y/o para la división.

4. Usa \(someVariable) para activar la interpolación de cadenas.

5. Si quieres ponerte elegante con print(), puedes usar Opción+Mayús+8 para obtener el símbolo de grados: °. Esto significa que puedes escribir algo así como 25 °F.