Tipos de datos complejos, parte 2

Hoy vamos a terminar nuestra mirada a los tipos de datos examinando la anotación de tipo, que es la forma de Swift de permitirnos dictar exactamente qué tipo de datos debe ser cada variable y constante. Una vez hecho eso, resumiremos lo que está cubierto y luego probaremos otro punto de control para que puedas evaluar lo que has aprendido hasta ahora.

Lo sé, probablemente estés harto de los tipos de datos en este momento, pero como dijo Eric Raymond, "las buenas estructuras de datos y el código malo funcionan mucho mejor que al revés".

Hoy aprenderás sobre la anotación de tipo, luego pasarás por el resumen y el punto de control. Hay algunas lecturas adicionales opcionales que te recomiendo que leas, junto con una breve prueba para asegurarte de que has entendido lo que se enseñó.

Hoy debería llevarte mucho menos de una hora completarlo, pero eso está bien, el día 13 probablemente tomará más de una hora, ¡así que todo se equilibra!

1. Cómo usar las anotaciones de tipo

Swift es capaz de averiguar qué tipo de datos tiene una constante o variable en función de lo que le asignamos. Sin embargo, a veces no queremos asignar un valor de inmediato, o a veces queremos anular la elección de tipo de Swift, y ahí es donde entran en anotaciones de tipo.

Hasta ahora hemos estado haciendo constantes y variables como esta:

```
let surname = "Lasso"
var score = 0
```

Esto utiliza la inferencia de tipo: Swift infiere que el apellido es una cadena porque le estamos asignando texto, y luego infiere que la puntuación es un entero porque le estamos asignando un número entero.

Las anotaciones de tipo nos permiten ser explícitos sobre qué tipos de datos queremos, y se ve así:

```
let surname: String = "Lasso"
var score: Int = 0
```

Ahora estamos siendo explícitos: el apellido debe ser una cadena y la puntuación debe ser un entero. Eso es exactamente lo que la inferencia de tipo de Swift habría hecho de todos modos, pero a veces no lo es, a veces querrás elegir un tipo diferente.

Por ejemplo, tal vez la puntuación sea un decimal porque el usuario puede obtener medio punto, así que escribirías esto:

```
var score: Double = 0
```

Sin la: Doble parte, Swift inferiría que es un entero, pero estamos anulando eso y diciendo que definitivamente es un número decimal.

Hemos analizado algunos tipos de datos hasta ahora, y es importante que conozcas sus nombres para que puedas usar la anotación de tipo correcta cuando sea necesario.

La cadena contiene el texto:

```
let playerName: String = "Roy"
```

Int tiene números enteros:

```
var luckyNumber: Int = 13
```

Double Tiene números decimales:

```
let pi: Double = 3.141
```

Bool Es verdadero o falso:

```
var isAuthenticated: Bool = true
```

Array Tiene muchos valores diferentes, todos en el orden en que los agregas. Esto debe ser especializado, como **[String]**:

```
var albums: [String] = ["Red", "Fearless"]
```

Dictionary Tiene muchos valores diferentes, donde se puede decidir cómo se debe acceder a los datos. Esto debe ser especializado, como **[String: Int]**:

```
var user: [String: String] = ["id": "@twostraws"]
```

Set Tiene muchos valores diferentes, pero los almacena en un orden que está optimizado para comprobar lo que contiene. Esto debe ser especializado, como **Set<String>**:

```
var books: Set<String> = Set(["The Bluest Eye", "Foundation", "Girl, Woman
```

Conocer todos estos tipos es importante para los momentos en los que no quieres proporcionar valores iniciales. Por ejemplo, esto crea una matriz de cadenas:

```
var soda: [String] = ["Coke", "Pepsi", "Irn-Bru"]
```

No se necesita una anotación de tipo allí, porque Swift puede ver que estás asignando una matriz de cadenas. Sin embargo, si quisieras crear una matriz vacía de cadenas, necesitarías saber el tipo:

```
var teams: [String] = [String]()
```

Una vez más, la anotación de tipo no es necesaria, pero aún así necesitas saber que una matriz de cadenas está escrita como **[String]** para que puedas hacer la cosa. Recuerde, debe agregar los paréntesis abiertos y cerrados al hacer matrices, diccionarios y conjuntos vacíos, porque es donde Swift nos permite personalizar la forma en que se crean.

Algunas personas prefieren usar la anotación de tipo y luego asignarle una matriz vacía de esta manera:

```
var cities: [String] = []
```

Prefiero usar la inferencia de tipo tanto como sea posible, así que escribiría esto:

```
var clues = [String]()
```

Además de todos esos, hay enumeraciones. Las enumeraciones son un poco diferentes de las demás porque nos permiten crear nuevos tipos propios, como una enumeración que contiene días de la semana, una enumeración que contiene el tema de la interfaz de usuario que el usuario quiere, o incluso una enumeración que contiene qué pantalla se muestra actualmente en nuestra aplicación.

Los valores de una enumeración tienen el mismo tipo que la enumeración en sí, por lo que podríamos escribir algo como esto:

```
enum UIStyle {
    case light, dark, system
}

var style = UIStyle.light
```

Esto es lo que permite a Swift eliminar el nombre de enumeración para futuras tareas, para que podamos escribir **style = .dark** - sabe que cualquier nuevo valor para **style** debe ser algún tipo de **UIStyle**

Ahora, hay una muy buena probabilidad de que te preguntes cuándo deberías usar anotaciones de tipo, por lo que podría ser útil que sepas que prefiero usar la inferencia de tipo tanto como sea posible, lo que significa que asigno un valor a una constante o variable y Swift elige el tipo correcto automáticamente. A veces esto significa usar algo como la puntuación **var score = 0,0** para obtener un **Double**.
La excepción más común a esto es con las constantes para las que todavía no tengo valor. Verás, Swift es realmente inteligente: puedes crear una constante que aún no tiene un valor, más tarde proporciona ese valor, y Swift se asegurará de que no la usemos accidentalmente hasta que haya un valor presente. También garantizará que solo establezca el valor una vez, para que permanezca constante.

Por ejemplo:

```
let username: String
// lots of complex logic
username = "@twostraws"
// lots more complex logic
print(username)
```

Ese código es legal: estamos diciendo que el **username** contendrá una cadena en algún momento, y proporcionamos un valor antes de usarlo. Si faltaba la línea de asignación - **username = "@twostraws"** -, entonces Swift se negaría a construir nuestro código porque el **username** no tendría un valor, y de manera similar, si intentáramos establecer un valor para el **username** por segunda vez, Swift también se quejaría.

Este tipo de código requiere una anotación de tipo, porque sin que se le asigne un valor inicial, Swift no sabe qué tipo de **username** de datos contendrá.

Independientemente de si usa inferencia de tipo o anotación de tipo, hay una regla de oro: Swift debe saber en todo momento qué tipos de datos contienen sus constantes y variables. Esto está en el centro de ser un lenguaje seguro para el tipo, y nos impide hacer cosas sin sentido como **5 + true** o similares.

Importante: Aunque la anotación de tipo puede permitirnos anular la inferencia de tipo de Swift hasta cierto punto, nuestro código terminado aún debe ser posible. Por ejemplo, esto no está permitido:

```
let score: Int = "Zero"
```

Swift simplemente no puede convertir "Cero" en un entero para nosotros, incluso con una anotación de tipo que lo solicita, por lo que el código simplemente no se compilará.

¿Por que Swift tiene anotaciones de tipo?

Una pregunta común que la gente hace al aprender Swift es "¿por qué Swift tiene anotaciones de tipo?", que generalmente va seguida de "¿cuándo debo usar anotaciones de tipo en Swift?"

La respuesta a la primera pregunta es principalmente una de tres razones:

1. Swift no puede averiguar qué tipo se debe usar.

2. Quieres que Swift use un tipo diferente al predeterminado.

3. Todavía no quieres asignar un valor.

El primero de ellos suele ocurrir solo en un código más avanzado. Por ejemplo, si estabas cargando algunos datos de Internet que sabes que son el nombre de tu político local, Swift no puede saberlo con anticipación, así que tendrás que decírselo.

El segundo escenario es bastante común a medida que aprendes más en Swift, pero en este momento un ejemplo simple está tratando de crear una variable doble sin tener que escribir constantemente ".0" en todas partes:

```
var percentage: Double = 99
```

Eso hace que el **percentage** sea el doble con un valor de 99,0. Sí, le hemos asignado un número entero, pero nuestra anotación de tipo deja claro que el tipo de datos real que queremos es el doble.

La tercera opción ocurre cuando quieres decirle a Swift que una variable va a existir, pero no quieres establecer su valor todavía. Esto sucede en muchos lugares de Swift, y se ve así:

```
var name: String
```

Luego puede asignar una cadena para nombrar más tarde, pero no puede asignar un tipo diferente porque Swift sabe que no sería válido.

Por supuesto, la segunda pregunta aquí es "¿cuándo debo usar las anotaciones de tipo en Swift?" Esto es mucho más subjetivo, porque la respuesta generalmente depende de tu estilo personal.

En mi propio código, prefiero usar la inferencia de tipo tanto como sea posible. Eso significa que dejo de lado las anotaciones de tipo y dejo que Swift averigüe el tipo de cosas en función de los datos que almaceno en ellas. Mis razones para esto son:

Hace que mi código sea más corto y más fácil de leer.

Me permite cambiar el tipo de algo simplemente cambiando lo que sea su valor inicial.

Algunas otras personas prefieren usar siempre una anotación de tipo explícita, y eso también funciona bien, realmente es una cuestión de estilo.

¿Por qué querrías crear una colección vacía?

Cuando estás aprendiendo Swift, es común ver ejemplos como este:

```
let names = ["Eleanor", "Chidi", "Tahani", "Jianyu", "Michael", "Janet"]
```

Esa es una matriz constante de seis cadenas, y debido a que es constante, significa que no podemos agregar más cosas a la matriz: conocemos todos nuestros elementos cuando se crea la matriz, por lo que el resto de nuestro programa solo está usando esos datos fijos.

Pero a veces no conoces todos tus datos por adelantado, y en esos casos es más común crear una colección vacía y luego añadir cosas a medida que calculas tus datos.

Por ejemplo, tenemos nuestra matriz de nombres fijos arriba, y si quisiéramos averiguar qué nombres comenzaron con J, entonces lo haríamos:

Crear una matriz de cadenas vacía llamada algo así como jNames

Revisa cada nombre de la matriz de nombres originales y comprueba si comienza con "J"

Si lo hace, añádelo a la matriz jNames.

Cuando hayamos repasado todos los nombres, terminaremos con dos cadenas en jNames: Jianyu y Janet. Por supuesto, si nuestra comprobación fuera qué nombres comenzaron con "X", entonces terminaríamos sin nombres en la matriz, y eso está bien. Comenzó como vacío y terminó como vacío.

Más adelante en este libro veremos el código Swift necesario para que todo esto funcione de verdad.

2. Resumen: Datos complejos

Ahora hemos ido más allá de los tipos de datos simples, y hemos empezado a buscar formas de agruparlos e incluso crear los nuestros propios con enumeraciones. Así que, vamos a recapitular:

- Las matrices nos permiten almacenar muchos valores en un solo lugar y luego leerlos usando índices enteros. Las matrices siempre deben estar especializadas para que contengan un tipo específico y tengan una funcionalidad útil como count, append() y contains().

- Los diccionarios también nos permiten almacenar muchos valores en un solo lugar, pero vamos a leerlos usando las claves que especificamos. Deben estar especializados para

tener un tipo específico para la clave y otro para el valor, y tener una funcionalidad similar a las matrices, como contains() y count.

- Los conjuntos son una tercera forma de almacenar muchos valores en un solo lugar, pero no podemos elegir el orden en el que almacenan esos artículos. Los conjuntos son muy eficientes para saber si contienen un artículo específico.

- Las enumeraciones nos permiten crear nuestros propios tipos simples en Swift para que podamos especificar un rango de valores aceptables, como una lista de acciones que el usuario puede realizar, los tipos de archivos que podemos escribir o los tipos de notificación para enviar al usuario.

- Swift siempre debe conocer el tipo de datos dentro de una constante o variable, y sobre todo utiliza la inferencia de tipo para averiguarlo en función de los datos que asignamos. Sin embargo, también es posible usar la anotación de tipo para forzar un tipo en particular.

De las matrices, diccionarios y conjuntos, es seguro decir que usarás las matrices con mucho más. Después de eso vienen los diccionarios, y los conjuntos vienen un tercio lejano. Eso no significa que los conjuntos no sean útiles, ¡pero sabrás cuándo los necesitas!

3. Punto de control 2

Ahora que has conocido matrices, diccionarios y conjuntos, quiero hacer una pausa por un momento para darte la oportunidad de resolver un pequeño desafío de codificación. No está diseñado para hacerte tropezar, sino para animarte a detenerte por un tiempo y pensar en lo que has aprendido.

Esta vez, el desafío es crear una matriz de cadenas, luego escribir algún código que imprima el número de elementos en la matriz y también el número de elementos únicos en la matriz.

Proporcionaré algunos consejos a continuación, pero por favor, tómese el tiempo para pensar en una solución antes de leerlos. Confía en mí: ¡olvidar lo que has aprendido y luego volver a aprenderlo, en realidad hace que se hunda más profundamente!

¿Sigues aquí? Vale, aquí hay algunas pistas:

- Deberías empezar creando una matriz de cuerdas, usando algo como **let albums = ["Red", "Fearless"]**

- Puedes leer el número de elementos de tu matriz usando **albums.count**.

- El recuento también existe para los conjuntos.

- Los conjuntos se pueden hacer a partir de matrices usando **Set(someArray)**

- Los conjuntos nunca incluyen duplicados.

Consejo: Aunque el punto de control no pide mucho, no te sorprendas si pasas algún tiempo mirando la pantalla preguntándote qué hacer. Eso no es una mala señal, de hecho, diría que es una buena señal. Creo firmemente que no hay aprendizaje sin lucha, ¡así que no tengas miedo de luchar por ello!

¡Comparte tu progreso!

Si usas Twitter, el botón de abajo preparará un tuit que diga que has completado hoy, junto con un gráfico de celebración, la URL de esta página y el hashtag del desafío. No te preocupes, ¡no se enviará hasta que lo confirmes en Twitter!