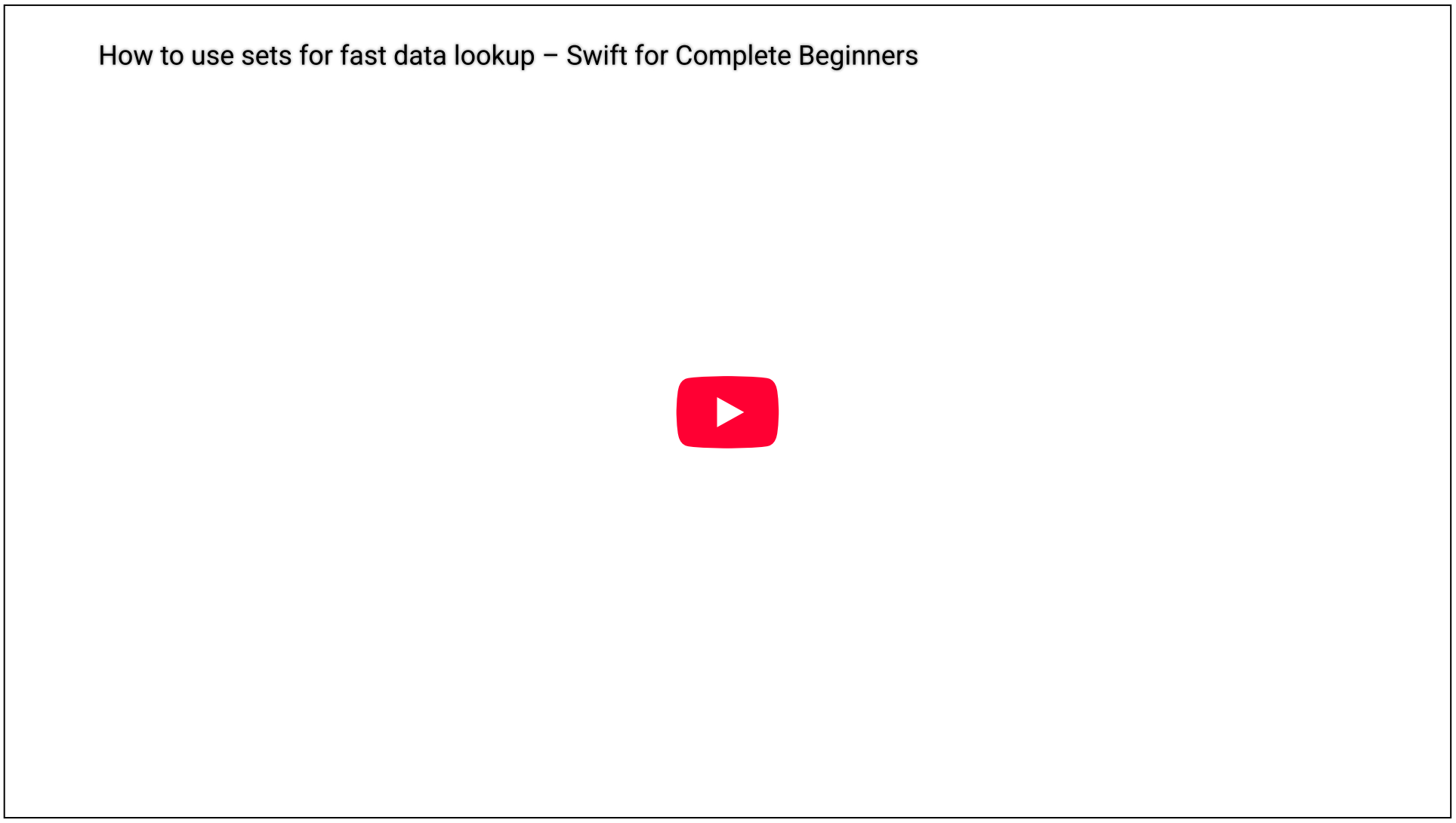
< How to store and find data in dictionaries

How to create and use enums >

How to use sets for fast data lookup

Paul Hudson @twostraws November 8th 2021

Updated for Xcode 16.4



So far you've learned about two ways of collecting data in Swift: arrays and dictionaries. There is a third very common way to group data, called a *set* – they are similar to arrays, except you can't add duplicate items, and they don't store their items in a particular order.

Creating a set works much like creating an array: tell Swift what kind of data it will store, then go ahead and add things. There are two important differences, though, and they are best demonstrated using some code.

First, here's how you would make a set of actor names:

```
let people = Set(["Denzel Washington", "Tom Cruise", "Nicolas Cage", "Samuel L Jackson"
```

Notice how that actually creates an array first, then puts that array into the set? That's intentional, and it's the standard way of creating a set from fixed data. Remember, the set will automatically remove any duplicate values, and it won't remember the exact order that was used in the array.

If you're curious how the set has ordered the data, just try printing it out:

```
print(people)
```

You *might* see the names in the original order, but you might also get a completely different order – the set just doesn't care what order its items come in.

The second important difference when adding items to a set is visible when you add items individually. Here's the code:

```
var people = Set<String>()
people.insert("Denzel Washington")
people.insert("Tom Cruise")
people.insert("Nicolas Cage")
people.insert("Samuel L Jackson")
```

Notice how we're using **`insert()`**? When we had an array of strings, we added items by calling **`append()`**, but that name doesn't make sense here – we aren't adding an item to the end of the set, because the set will store the items in whatever order it wants.

Now, you might think sets just sound like simplified arrays – after all, if you can't have duplicates and you lose the order of your items, why not just use arrays? Well, both of those restrictions actually get turned into an advantage.

First, not storing duplicates is sometimes exactly what you want. There's a reason I chose actors in the previous example: the Screen Actors Guild requires that all its members have a unique stage name to avoid confusion, which means that duplicates must never be allowed. For example, the actor Michael Keaton (Spider-Man Homecoming, Toy Story 3, Batman, and more) is actually named Michael Douglas, but because there was *already* a Michael Douglas in the guild (Avengers, Falling Down, Romancing the Stone, and more), he had to have a unique name.

Second, instead of storing your items in the exact order you specify, sets instead store them in a highly optimized order that makes it very fast to locate items. And the difference isn't small: if you have an array of 1000 movie names and use something like **`contains()`** to check whether it contains "The Dark Knight" Swift needs to go through every item until it finds one that matches – that might mean checking all 1000 movie names before returning false, because The Dark Knight wasn't in the array.

In comparison, calling **`contains()`** on a set runs so fast you'd struggle to measure it meaningfully. Heck, even if you had a million items in the set, or even 10 million items, it would still run instantly, whereas an array might take minutes or longer to do the same work.

Most of the time you'll find yourself using arrays rather than sets, but sometimes – just sometimes – you'll find that a set is exactly the right choice to solve a particular problem, and it will make otherwise slow code run in no time at all.

Tip: Alongside **`contains()`**, you'll also find **`count`** to read the number of items in a set, and **`sorted()`** to return a sorted array containing the the set's items.

< How to store and find data in dictionaries

How to create and use enums >