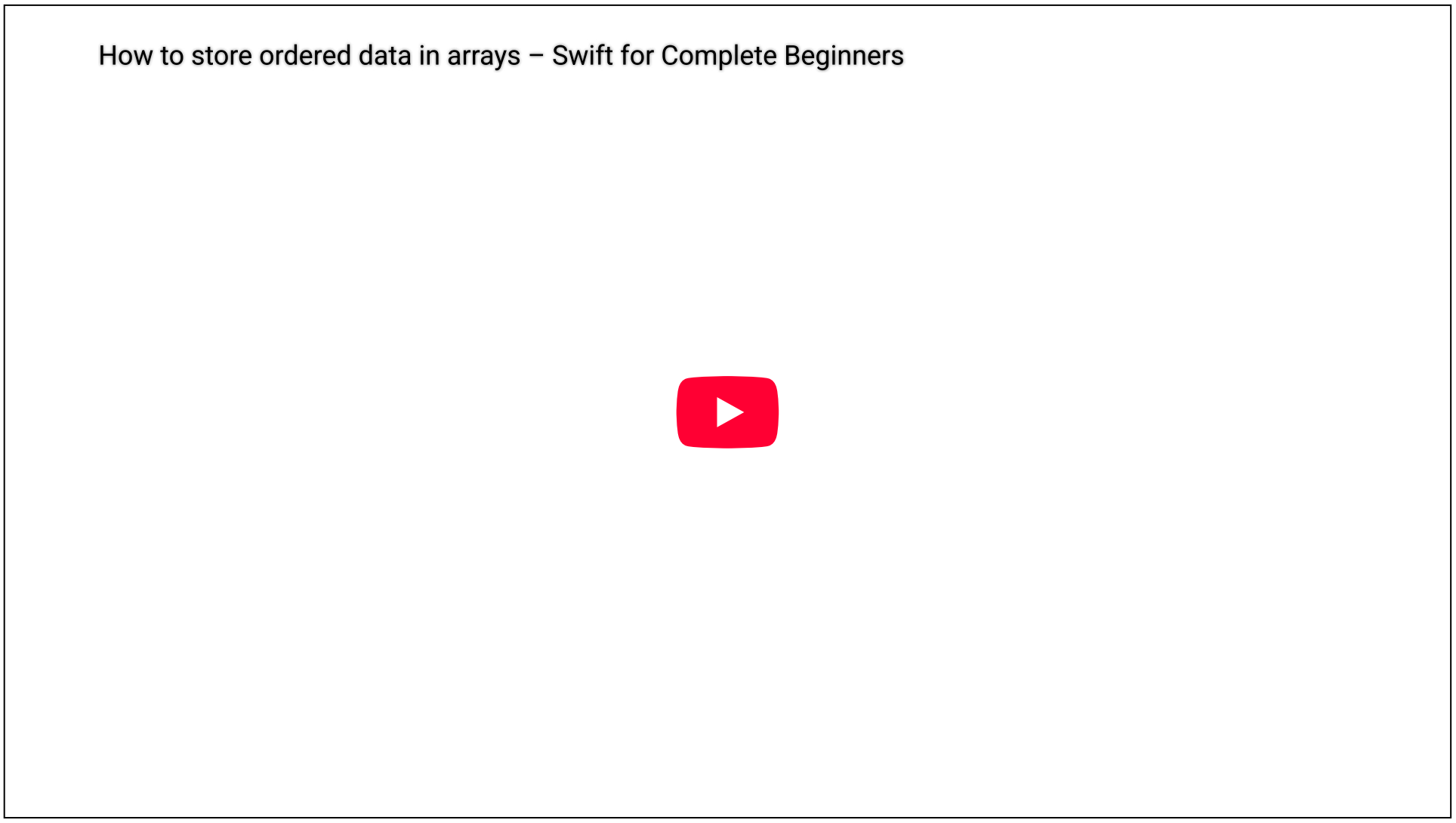
< Checkpoint 1 How to store and find data in dictionaries >

How to store ordered data in arrays

Paul Hudson @twostraws October 25th 2021

Updated for Xcode 16.4



It's extremely common to want to have lots of data in a single place, whether that's the days of the week, a list of students in a class, a city's population for the last 100 years, or any of countless other examples.

In Swift, we do this grouping using an *array*. Arrays are their own data type just like `String`, `Int`, and `Double`, but rather than hold just one string they can hold zero strings, one string, two strings, three, fifty, fifty million, or even more strings – they can automatically adapt to hold as many as you need, and always hold data in the order you add it.

Let's start with some simple examples of creating arrays:

```
var beatles = ["John", "Paul", "George", "Ringo"]
let numbers = [4, 8, 15, 16, 23, 42]
var temperatures = [25.3, 28.2, 26.4]
```

That creates three different arrays: one holding strings of people's names, one holding integers of important numbers, and one holding decimals of temperatures in Celsius. Notice how we start and end arrays using square brackets, with commas between every item.

When it comes to reading values *out* from an array, we ask for values by the position they appear in the array. The position of an item in an array is commonly called its *index*.

This confuses beginners a bit, but Swift actually counts an item's index from zero rather than one – `beatles[0]` is the first element, and `beatles[1]` is the second, for example.

So, we could read some values out from our arrays like this:

```
print(beatles[0])
print(numbers[1])
print(temperatures[2])
```

Tip: Make sure an item exists at the index you're asking for, otherwise your code will crash – your app will just stop working.

If your array is variable, you can modify it after creating it. For example, you can use **`append()`** to add new items:

```
beatles.append("Adrian")
```

And there's nothing stopping you from adding items more than once:

```
beatles.append("Allen")
beatles.append("Adrian")
beatles.append("Novall")
beatles.append("Vivian")
```

However, Swift does watch the *kind* of data you're trying to add, and will make sure your array only ever contains one type of data at a time. So, this kind of code isn't allowed:

```
temperatures.append("Chris")
```

This also applies to reading data out of the array – Swift knows that the **`beatles`** array contains strings, so when you read one value out you'll always get a string. If you try to do the same with **`numbers`**, you'll always get an integer. Swift won't let you mix these two different types together, so this kind of code isn't allowed:

```
let firstBeatle = beatles[0]
let firstNumber = numbers[0]
let notAllowed = firstBeatle + firstNumber
```

This is type safety, just like how Swift won't let us mix integers and decimals, except it's taken to a deeper level. Yes, all **`beatles`** and **`numbers`** are both arrays, but they are specialized types of arrays: one is an array of strings, and one is an array of integers.

You can see this more clearly when you want to start with an empty array and add items to it one by one. This is done with very precise syntax:

```
var scores = Array<Int>()
scores.append(100)
scores.append(80)
scores.append(85)
print(scores[1])
```

We've covered the last four lines already, but that first line shows how we have a specialized array type – this isn't just any array, it's an array that holds integers. This is what allows Swift to know for sure that **`beatles[0]`** must always be a string, and also what stops us from adding integers to a string array.

The open and closing parentheses after **Array<Int>** are there because it's possible to customize the way the array is created if you need to. For example, you might want to fill the array with lots of temporary data before adding the real stuff later on.

You can make other kinds of array by specializing it in different ways, like this:

```
var albums = Array<String>()
albums.append("Folklore")
albums.append("Fearless")
albums.append("Red")
```

Again, we've said that must always contain strings, so we can't try to put an integer in there.

Arrays are so common in Swift that there's a special way to create them: rather than writing **Array<String>**, you can instead write **[String]**. So, this kind of code is exactly the same as before:

```
var albums = [String]()
albums.append("Folklore")
albums.append("Fearless")
albums.append("Red")
```

Swift's type safety means that it must always know what type of data an array is storing. That might mean being explicit by saying **albums** is an **Array<String>**, but if you provide some initial values Swift can figure it out for itself:

```
var albums = ["Folklore"]
albums.append("Fearless")
albums.append("Red")
```

Before we're done, I want to mention some useful functionality that comes with arrays.

First, you can use **.count** to read how many items are in an array, just like you did with strings:

```
print(albums.count)
```

Second, you can remove items from an array by using either **remove(at:)** to remove one item at a specific index, or **removeAll()** to remove everything:

```
var characters = ["Lana", "Pam", "Ray", "Sterling"]
print(characters.count)

characters.remove(at: 2)
print(characters.count)

characters.removeAll()
print(characters.count)
```

That will print 4, then 3, then 0 as characters are removed.

Third, you can check whether an array contains a particular item by using **contains()**, like this:

```
let bondMovies = ["Casino Royale", "Spectre", "No Time To Die"]
print(bondMovies.contains("Frozen"))
```

Fourth, you can sort an array using **`sorted()`**, like this:

```
let cities = ["London", "Tokyo", "Rome", "Budapest"]
print(cities.sorted())
```

That returns a new array with its items sorted in ascending order, which means alphabetically for strings but numerically for numbers – the original array remains unchanged.

Finally, you can reverse an array by calling **`reversed()`** on it:

```
let presidents = ["Bush", "Obama", "Trump", "Biden"]
let reversedPresidents = presidents.reversed()
print(reversedPresidents)
```

Tip: When you reverse an array, Swift is very clever – it doesn't actually do the work of rearranging all the items, but instead just remembers to itself that you want the items to be reversed. So, when you print out **`reversedPresidents`**, don't be surprised to see it's not just a simple array any more!

Arrays are extremely common in Swift, and you'll have lots of opportunity to learn more about them as you progress. Even better **`sorted()`**, **`reversed()`**, and lots of other array functionality also exists for strings – using **`sorted()`** there puts the string's letters in alphabetical order, making something like "swift" into "fistw".

< Checkpoint 1

How to store and find data in dictionaries >

Was this page useful? Let us know!

☆☆☆☆☆

Average rating: 4.7/5

You are not logged in

Log in or create account