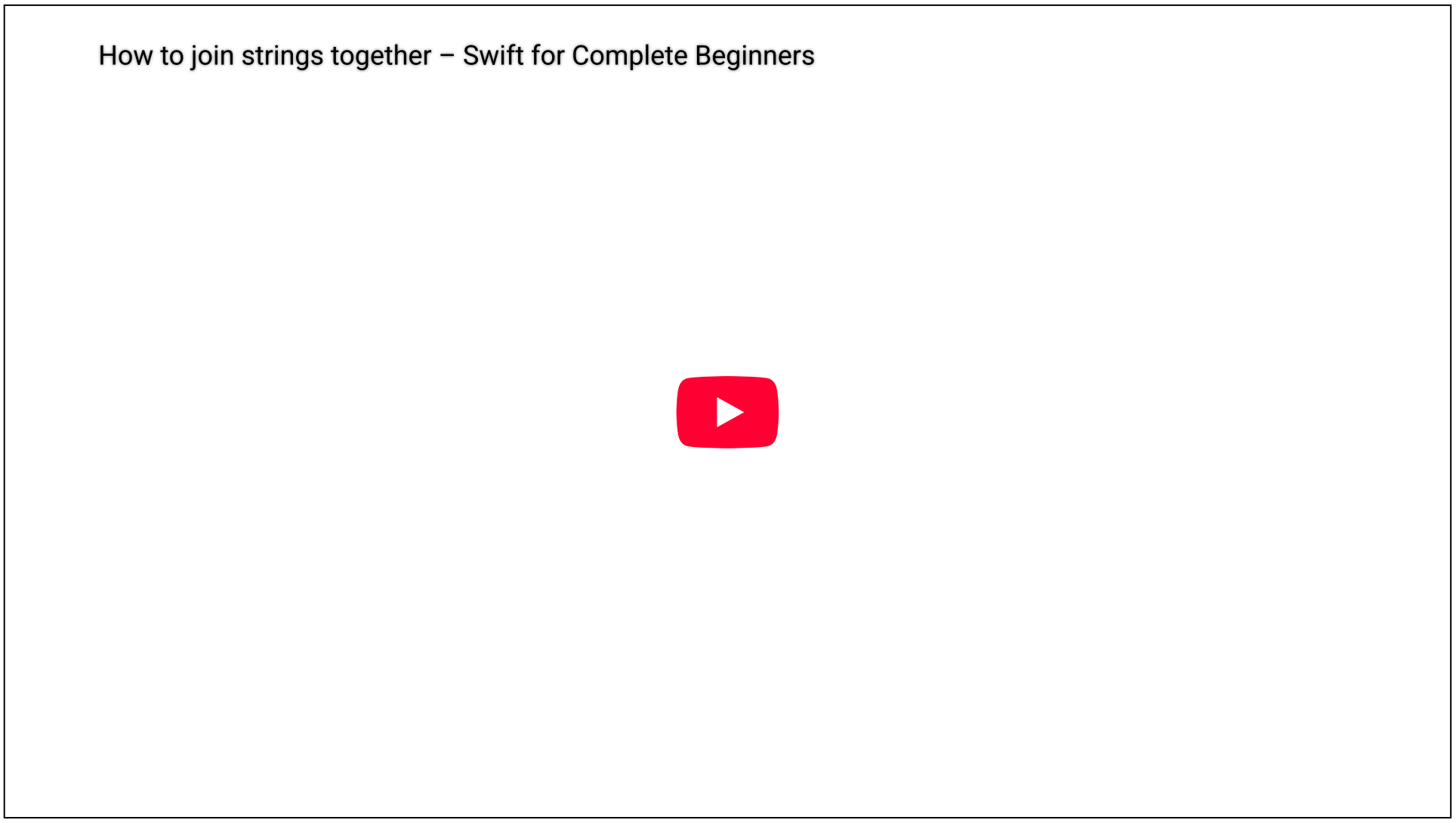
< How to store truth with Booleans

Summary: Simple data >

How to join strings together

Paul Hudson @twostraws October 25th 2021

Updated for Xcode 16.4



Swift gives us two ways to combine strings together: joining them using **+**, and a special technique called *string* interpolation that can place variables of any type directly inside strings.

Let's start with the easier option first, which is using **+** to join strings together: when you have two strings, you can join them together into a new string just by using **+**, like this:

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

You can do this many times if you need to:

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

When that runs it will print "Haters gonna hate" – yes, I'm a big fan of Taylor Swift, and I think her lyrics make a natural fit for a tutorial about Swift programming!

Notice how we're using **+** to join two strings, but when we used **`Int`** and **`Double`** it added numbers together? This is called *operator overloading* – the ability for one operator such as **+** to mean different things depending on how it's used. For strings, it also applies to **+=**, which adds one string directly to another.

This technique works great for small things, but you wouldn't want to do it too much. You see, each time Swift sees two strings being joined together using **+** it has to make a new string out of them before continuing, and if you have lots of things being joined it's quite wasteful.

Think about this for example:

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

Swift can't join all those strings in one go. Instead, it will join the first two to make "12", then join "12" and "3" to make "123", then join "123" and "4" to make "1234", and finally join "1234" and "5" to make "12345" – it makes temporary strings to hold "12", "123", and "1234" even though they aren't ultimately used when the code finishes.

Swift has a better solution called *string interpolation*, and it lets us efficiently create strings from other strings, but also from integers, decimal numbers, and more.

If you remember, earlier I said that you can include double quotes inside strings as long as they have a backslash before them so Swift knows to treat them specially:

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

Something very similar is used with string interpolation: you write a backslash inside your string, then place the name of a variable or constant inside parentheses.

For example, we could create one string constant and one integer constant, then combine them into a new string:

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

When that code runs, it will print "Hello, my name is Taylor and I'm 26 years old."

String interpolation is much more efficient than using **+** to join strings one by one, but there's another important benefit too: you can pull in integers, decimals, and more with no extra work.

You see, using **+** lets us add strings to strings, integers to integers, and decimals to decimals, but *doesn't* let us add integers to strings. So, this kind of code is not allowed:

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

You *could* ask Swift to treat the number like a string if you wanted, like this:

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

It is still both faster and easier to read to use string interpolation:

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

Tip: You can put calculations inside string interpolation if you want to. For example, this will print “5 x 5 is 25”:

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


< How to store truth with Booleans

Summary: Simple data >

Was this page useful? Let us know!

☆☆☆☆☆

Average rating: 4.9/5