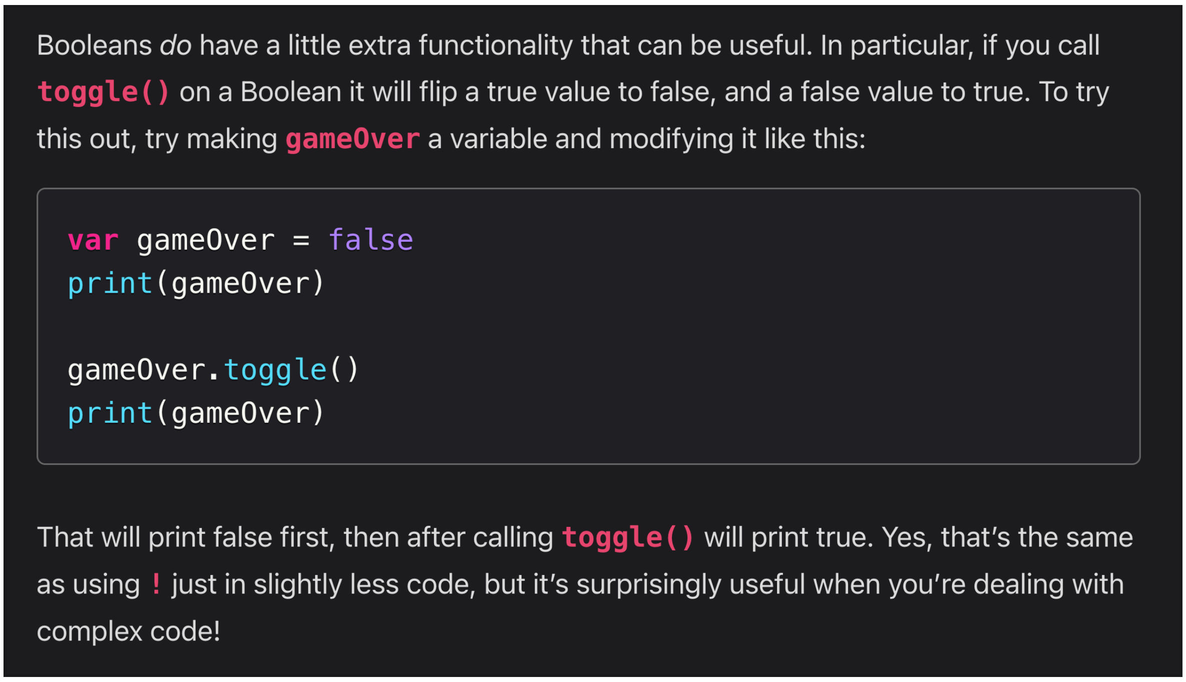
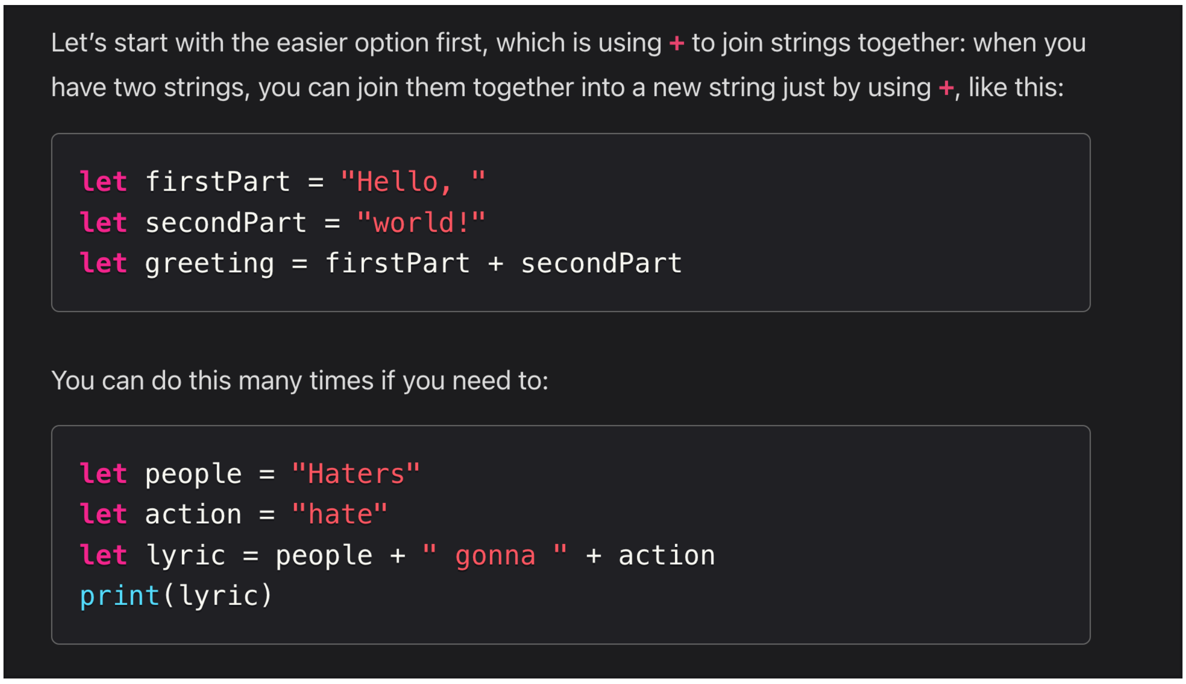
Day 2



Booleans *do* have a little extra functionality that can be useful. In particular, if you call `toggle()` on a Boolean it will flip a true value to false, and a false value to true. To try this out, try making `gameOver` a variable and modifying it like this:

```
var gameOver = false
print(gameOver)

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

That will print false first, then after calling `toggle()` will print true. Yes, that's the same as using `!` just in slightly less code, but it's surprisingly useful when you're dealing with complex code!

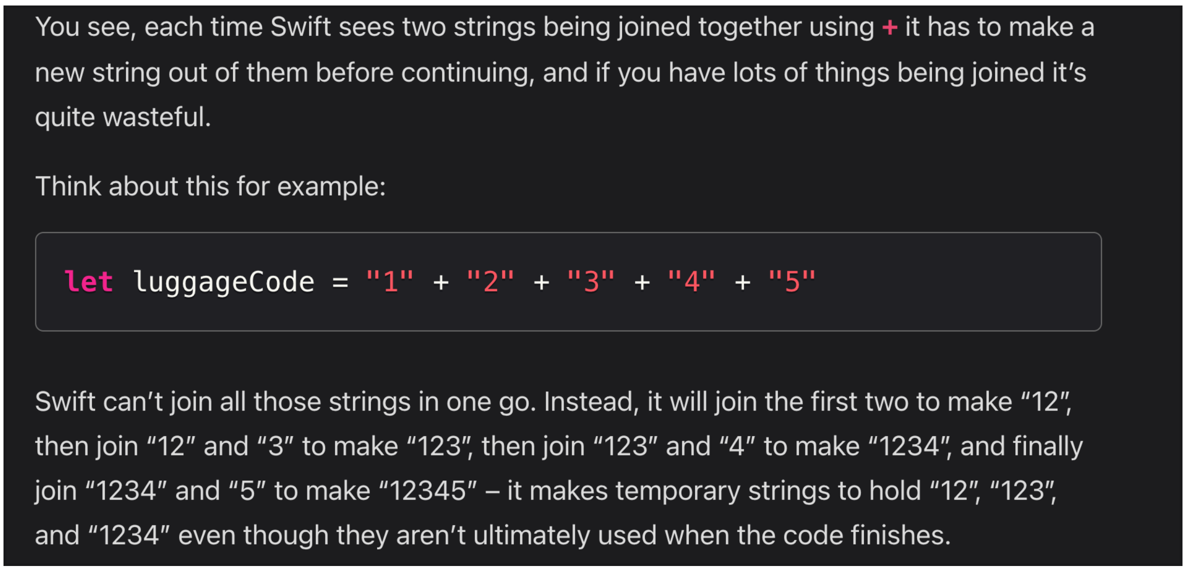


Let's start with the easier option first, which is using `+` to join strings together: when you have two strings, you can join them together into a new string just by using `+`, like this:

```
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)
```



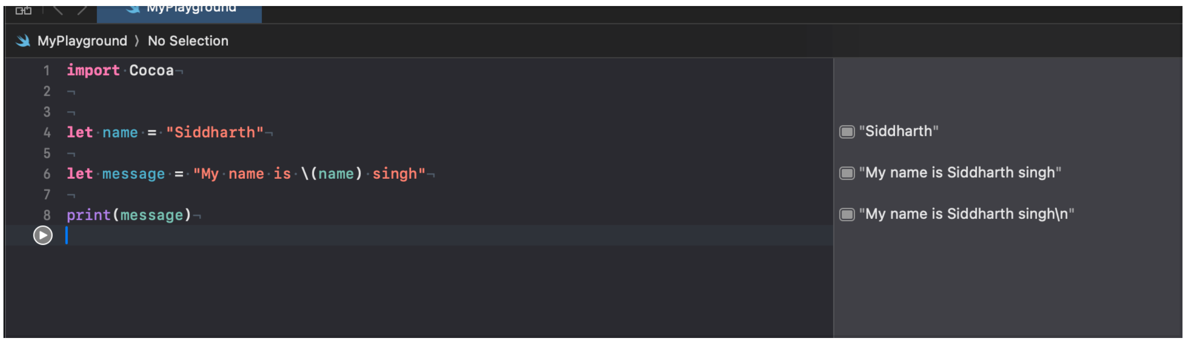
You see, each time Swift sees two strings being joined together using **+** it has to make a new string out of them before continuing, and if you have lots of things being joined it's quite wasteful.

Think about this for example:

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

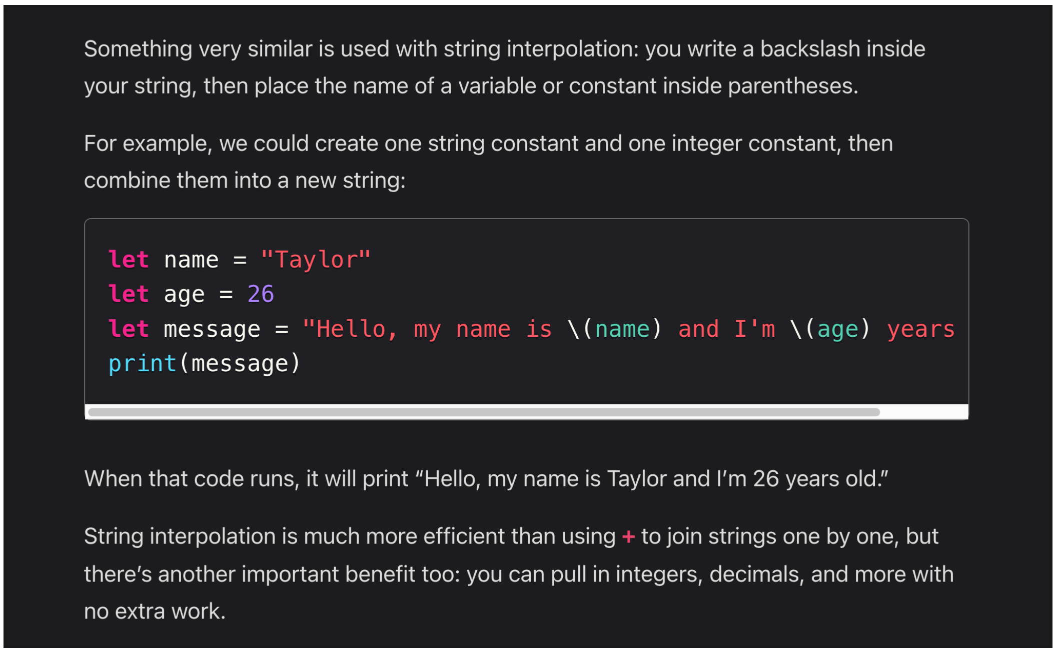
Swift can't join all those strings in one go. Instead, it will join the first two to make "12", then join "12" and "3" to make "123", then join "123" and "4" to make "1234", and finally join "1234" and "5" to make "12345" – it makes temporary strings to hold "12", "123", and "1234" even though they aren't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

String Interpolation :



```swift
import Cocoa


let name = "Siddharth"

let message = "My name is \(name) singh"

print(message)
```

```
"Siddharth"
"My name is Siddharth singh"
"My name is Siddharth singh\n"
```



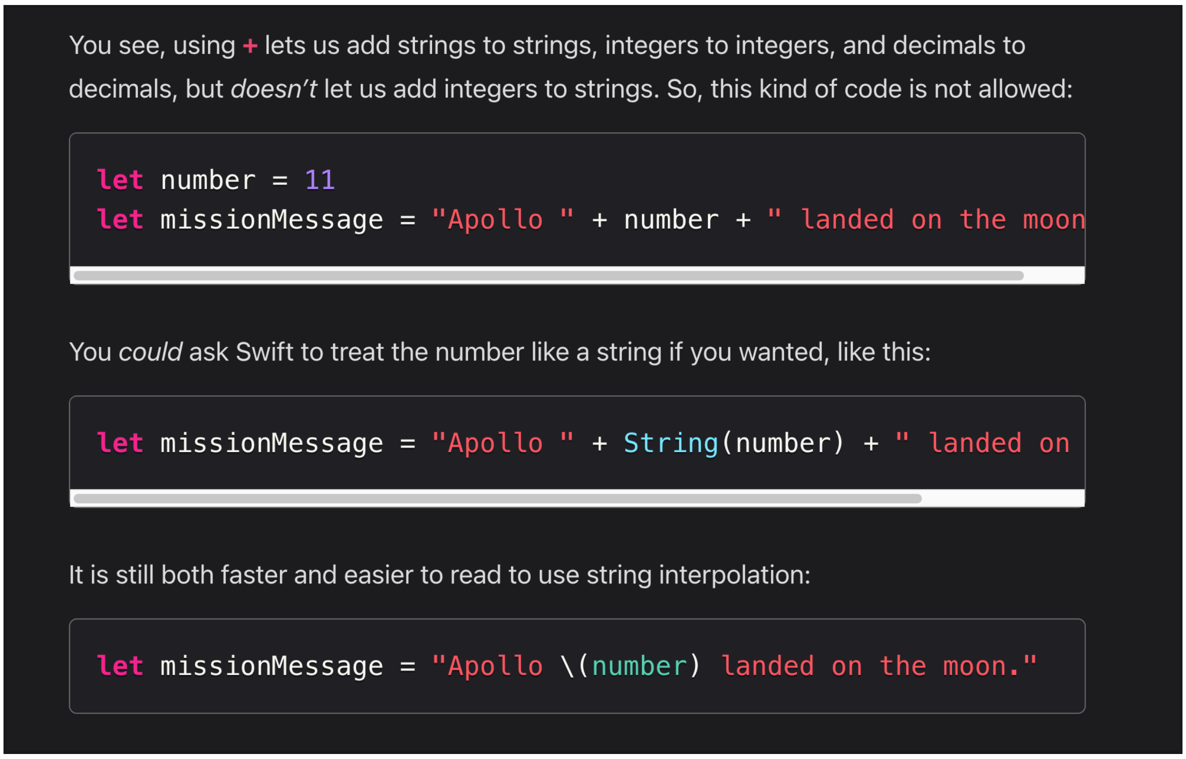
Something very similar is used with string interpolation: you write a backslash inside your string, then place the name of a variable or constant inside parentheses.

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
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:

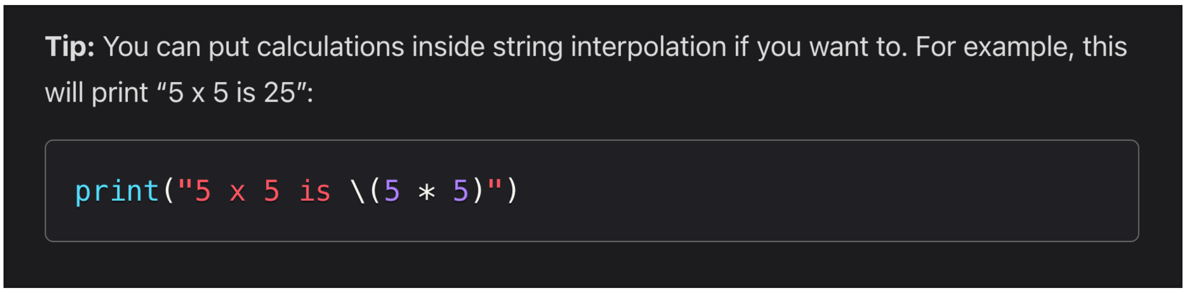
```
let number = 11
let missionMessage = "Apollo " + number + " landed on the moon
```

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

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

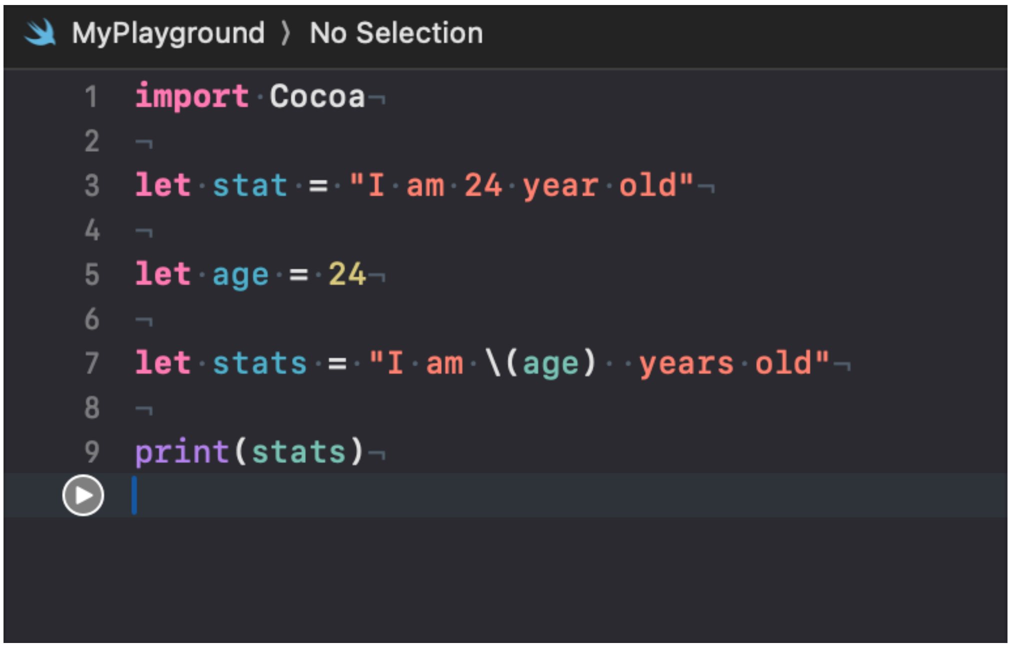
It is still both faster and easier to read to use string interpolation:

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



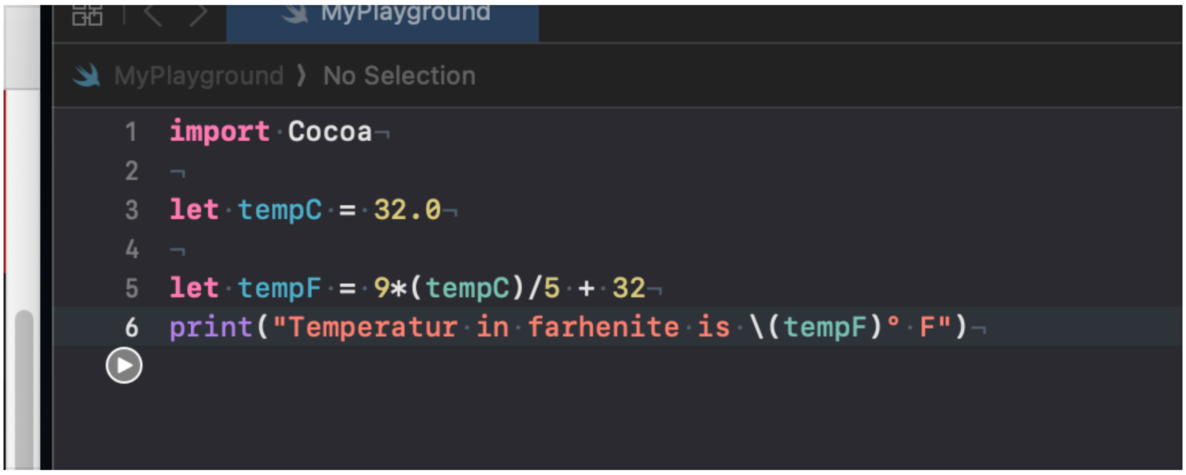
Tip: You can put calculations inside string interpolation if you want to. For example, this will print “5 x 5 is 25”:

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



```swift
import Cocoa

let stat = "I am 24 year old"

let age = 24

let stats = "I am \(age)  years old"

print(stats)
```

Celcius to Farhenite conversion ->



```swift
import Cocoa

let tempC = 32.0

let tempF = 9*(tempC)/5 + 32
print("Temperatur in farhenite is \(tempF)° F")
```