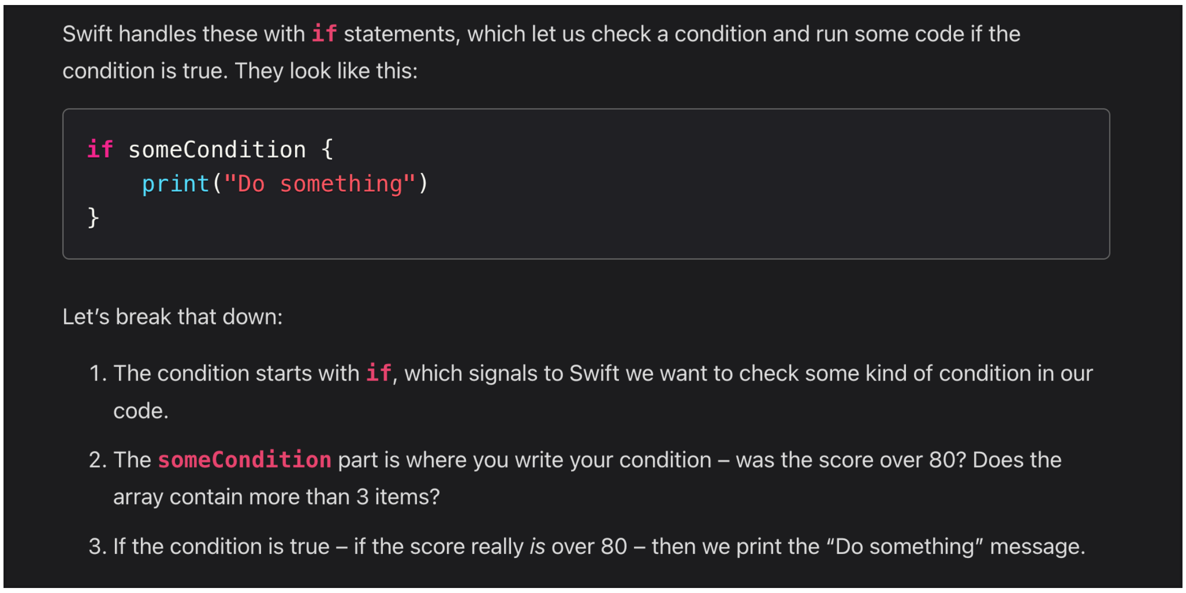
Day 5

Conditions (IF else)

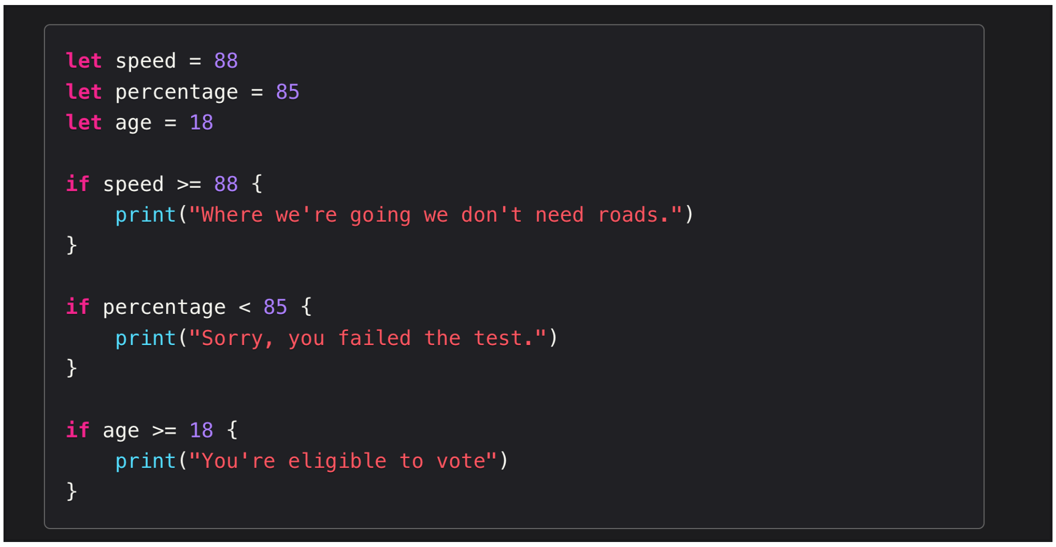


Swift handles these with `if` statements, which let us check a condition and run some code if the condition is true. They look like this:

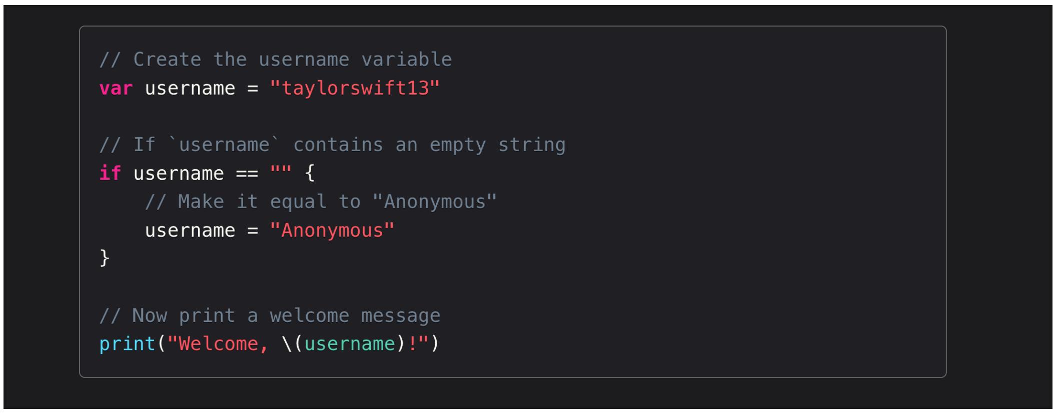
```swift
if someCondition {
    print("Do something")
}
```

Let’s break that down:

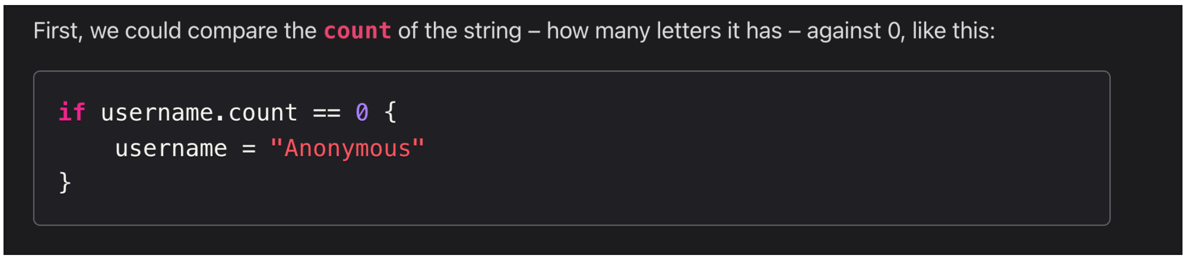
1. The condition starts with `if`, which signals to Swift we want to check some kind of condition in our code.
2. The `someCondition` part is where you write your condition – was the score over 80? Does the array contain more than 3 items?
3. If the condition is true – if the score really *is* over 80 – then we print the “Do something” message.



```swift
let speed = 88
let percentage = 85
let age = 18

if speed >= 88 {
    print("Where we're going we don't need roads.")
}

if percentage < 85 {
    print("Sorry, you failed the test.")
}

if age >= 18 {
    print("You're eligible to vote")
}
```

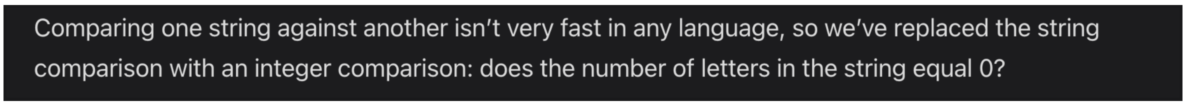


```swift
// Create the username variable
var username = "taylorswift13"

// If `username` contains an empty string
if username == "" {
    // Make it equal to "Anonymous"
    username = "Anonymous"
}

// Now print a welcome message
print("Welcome, \(username)!")
```

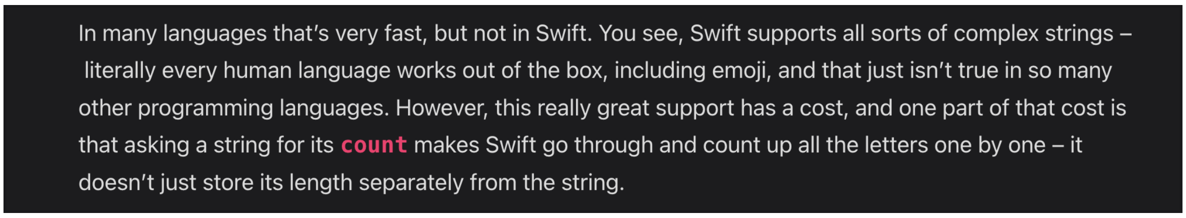


First, we could compare the `count` of the string – how many letters it has – against 0, like this:

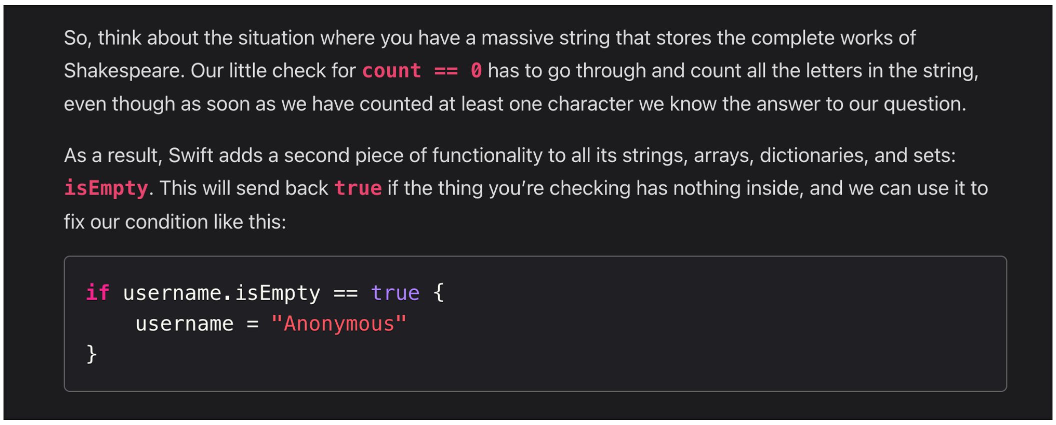
```swift
if username.count == 0 {
    username = "Anonymous"
}
```



Comparing one string against another isn't very fast in any language, so we've replaced the string comparison with an integer comparison: does the number of letters in the string equal 0?



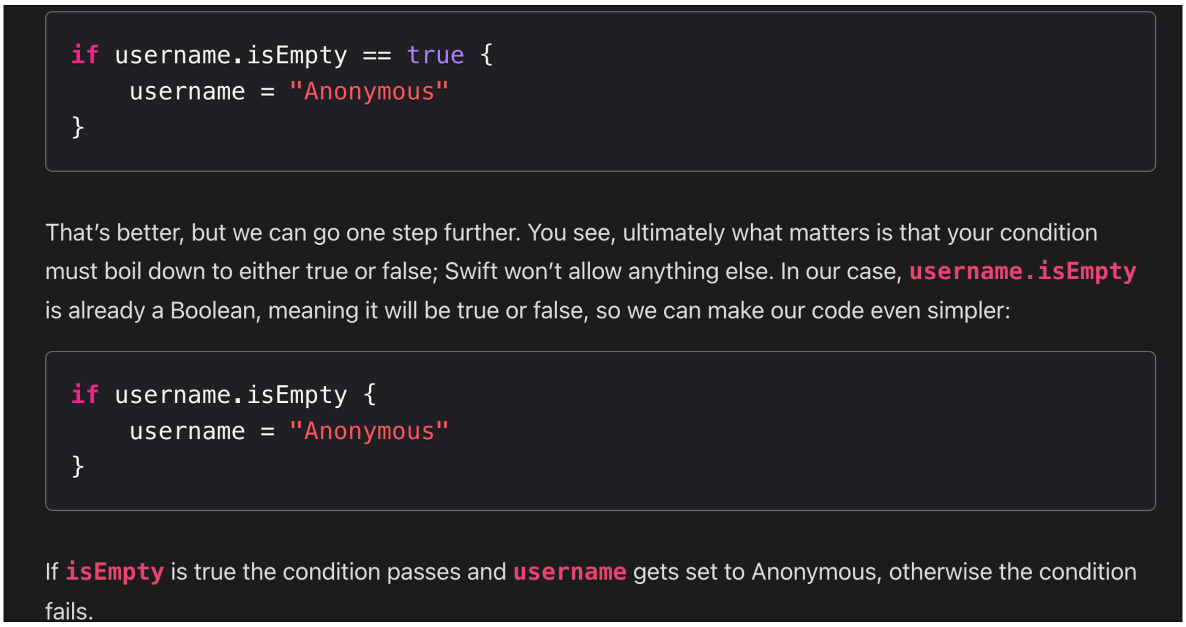
In many languages that's very fast, but not in Swift. You see, Swift supports all sorts of complex strings – literally every human language works out of the box, including emoji, and that just isn't true in so many other programming languages. However, this really great support has a cost, and one part of that cost is that asking a string for its `count` makes Swift go through and count up all the letters one by one – it doesn't just store its length separately from the string.



So, think about the situation where you have a massive string that stores the complete works of Shakespeare. Our little check for `count == 0` has to go through and count all the letters in the string, even though as soon as we have counted at least one character we know the answer to our question.

As a result, Swift adds a second piece of functionality to all its strings, arrays, dictionaries, and sets: `isEmpty`. This will send back `true` if the thing you're checking has nothing inside, and we can use it to fix our condition like this:

```
if username.isEmpty == true {
    username = "Anonymous"
}
```

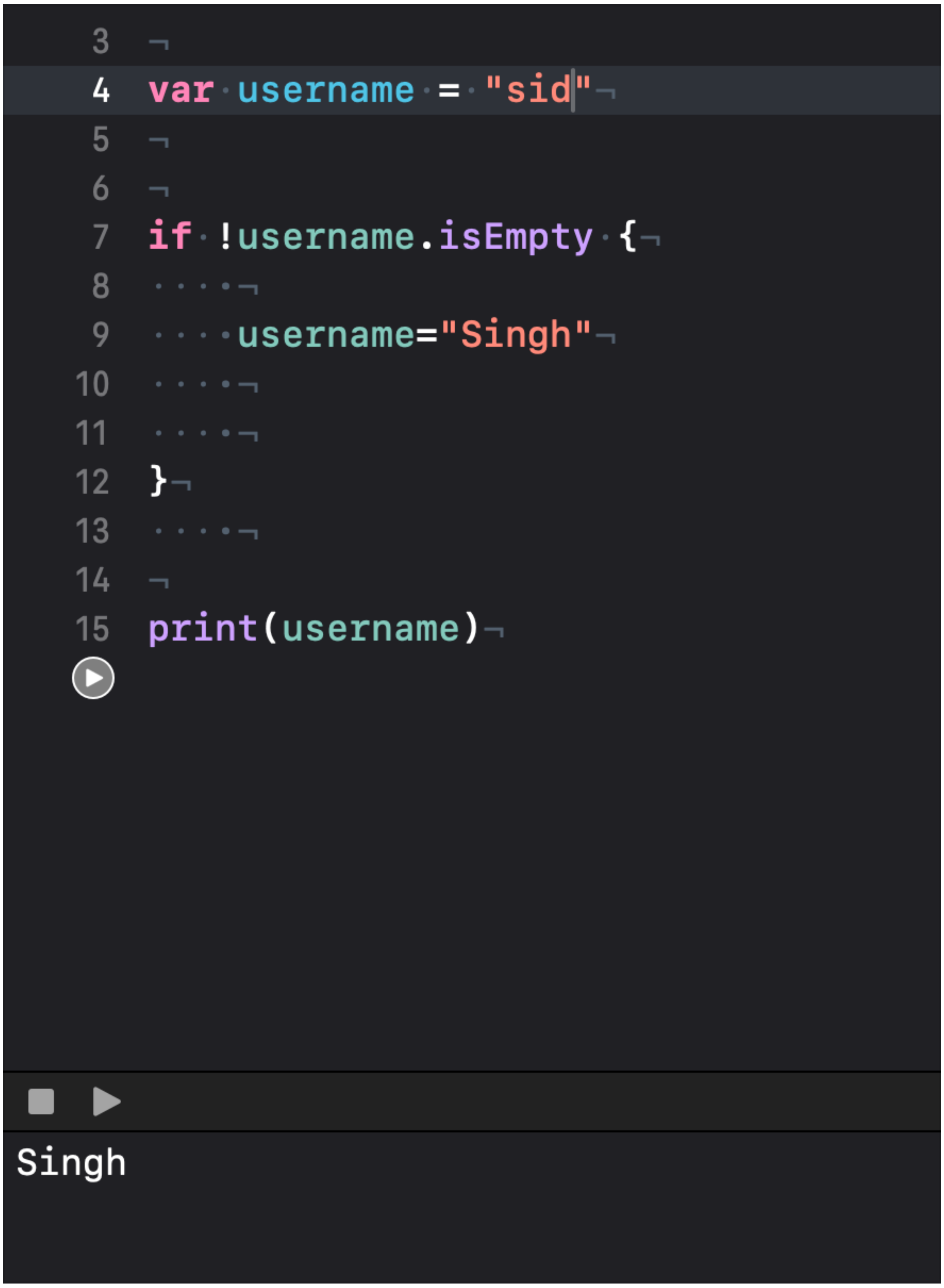


```
if username.isEmpty == true {
    username = "Anonymous"
}
```

That's better, but we can go one step further. You see, ultimately what matters is that your condition must boil down to either true or false; Swift won't allow anything else. In our case, `username.isEmpty` is already a Boolean, meaning it will be true or false, so we can make our code even simpler:

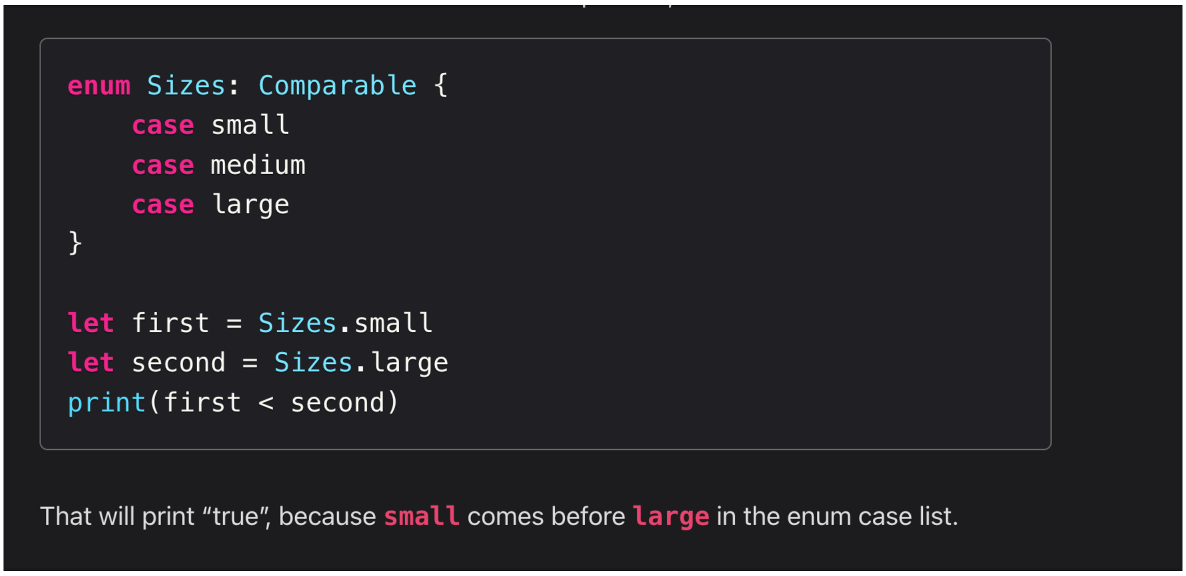
```
if username.isEmpty {
    username = "Anonymous"
}
```

If `isEmpty` is true the condition passes and `username` gets set to Anonymous, otherwise the condition fails.



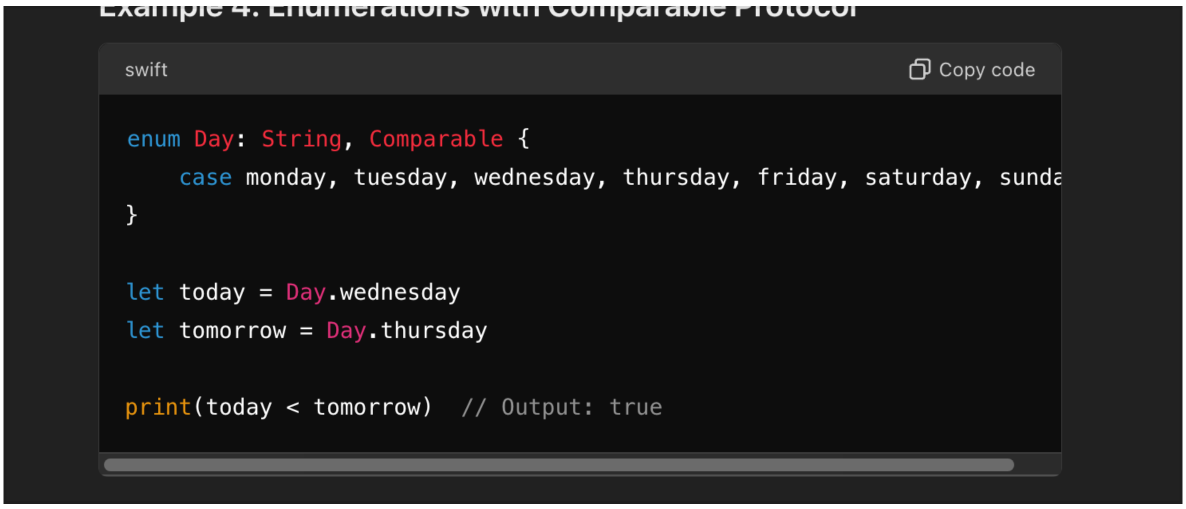
```swift

var username = "sid"


if !username.isEmpty {
    
    username="Singh"
    
    
}
    

print(username)
```

```
Singh
```



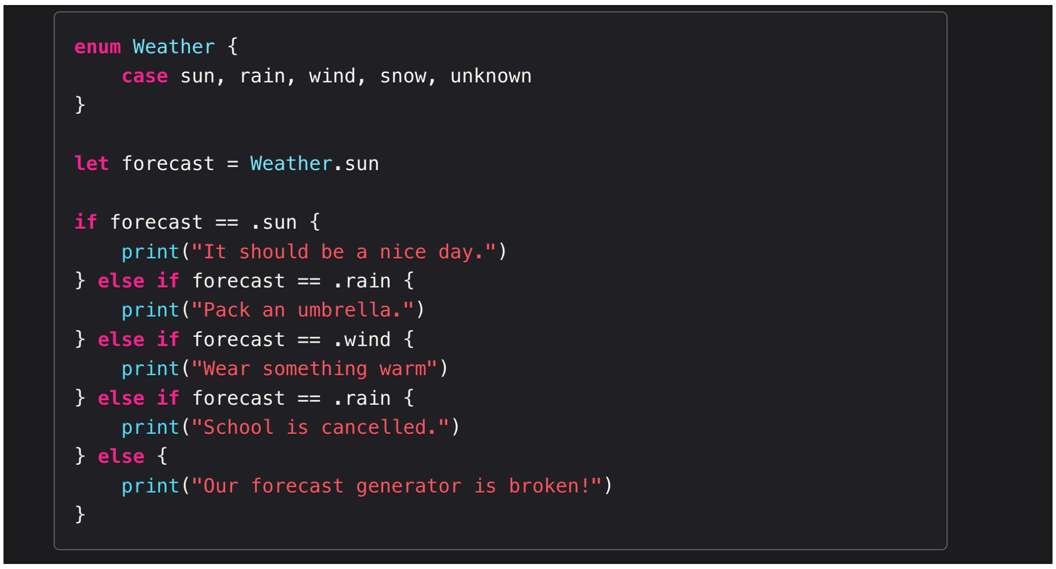
```swift
enum Sizes: Comparable {
    case small
    case medium
    case large
}

let first = Sizes.small
let second = Sizes.large
print(first < second)
```

That will print “true”, because `small` comes before `large` in the enum case list.

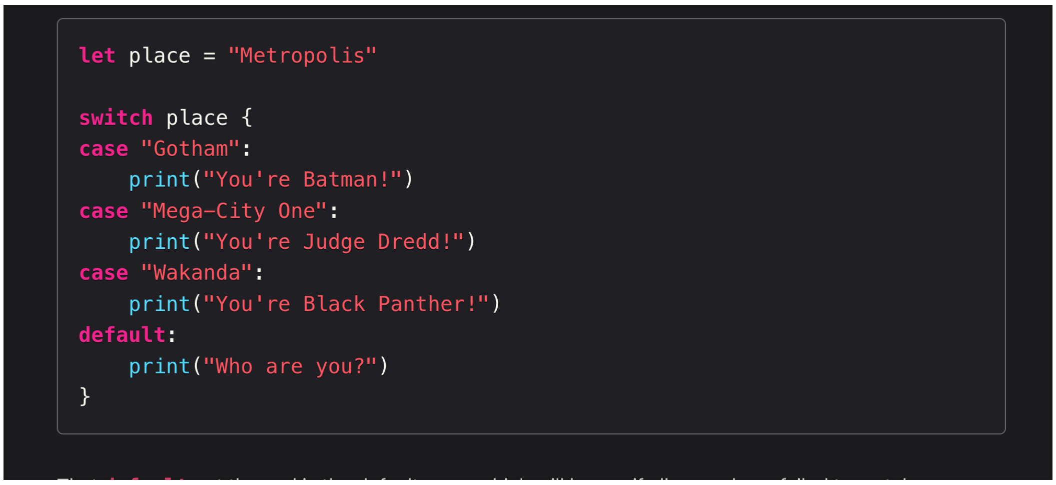


Example 4: Enumerations with Comparable Protocol

```swift
enum Day: String, Comparable {
    case monday, tuesday, wednesday, thursday, friday, saturday, sunda
}

let today = Day.wednesday
let tomorrow = Day.thursday

print(today < tomorrow)  // Output: true
```



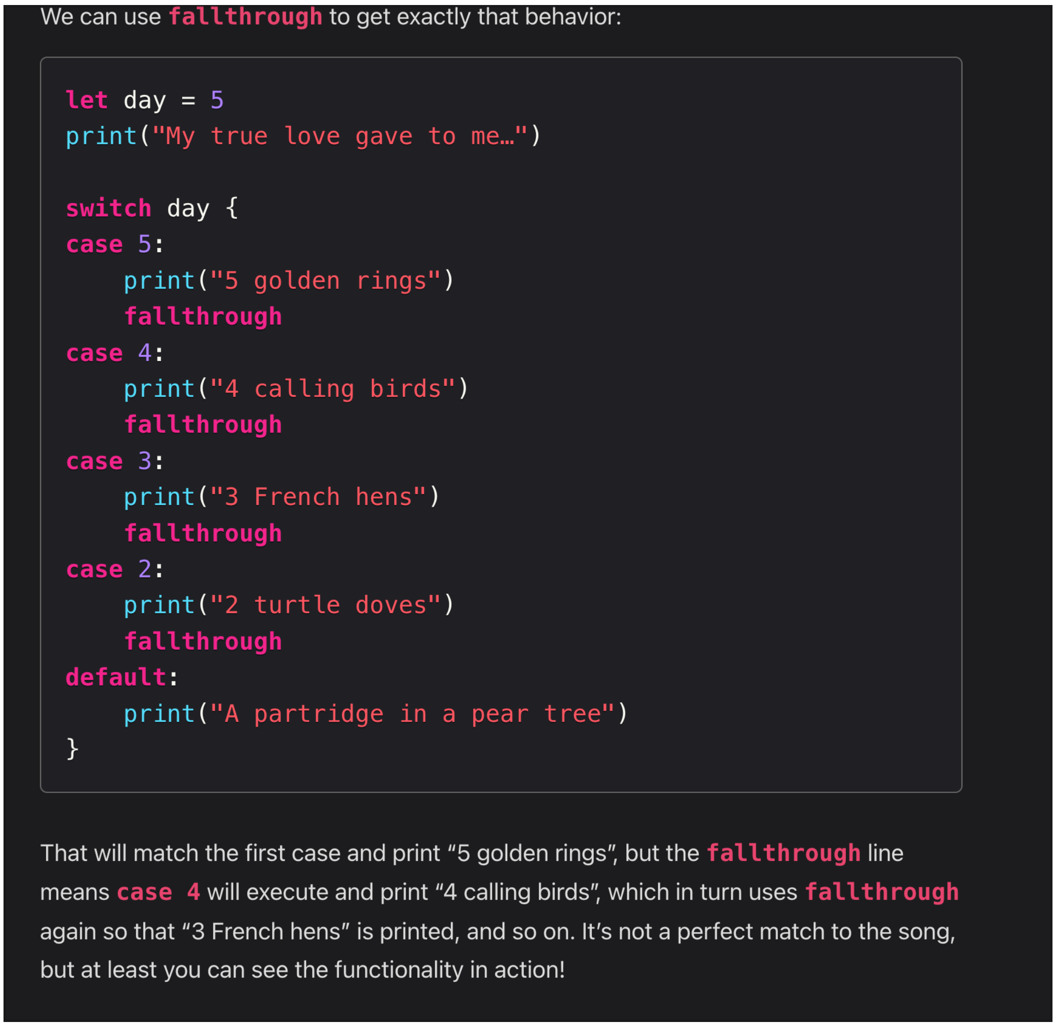
```swift
enum Weather {
    case sun, rain, wind, snow, unknown
}

let forecast = Weather.sun

if forecast == .sun {
    print("It should be a nice day.")
} else if forecast == .rain {
    print("Pack an umbrella.")
} else if forecast == .wind {
    print("Wear something warm")
} else if forecast == .rain {
    print("School is cancelled.")
} else {
    print("Our forecast generator is broken!")
}
```



```swift
let place = "Metropolis"

switch place {
case "Gotham":
    print("You're Batman!")
case "Mega-City One":
    print("You're Judge Dredd!")
case "Wakanda":
    print("You're Black Panther!")
default:
    print("Who are you?")
}
```

Default will always be at the last line .

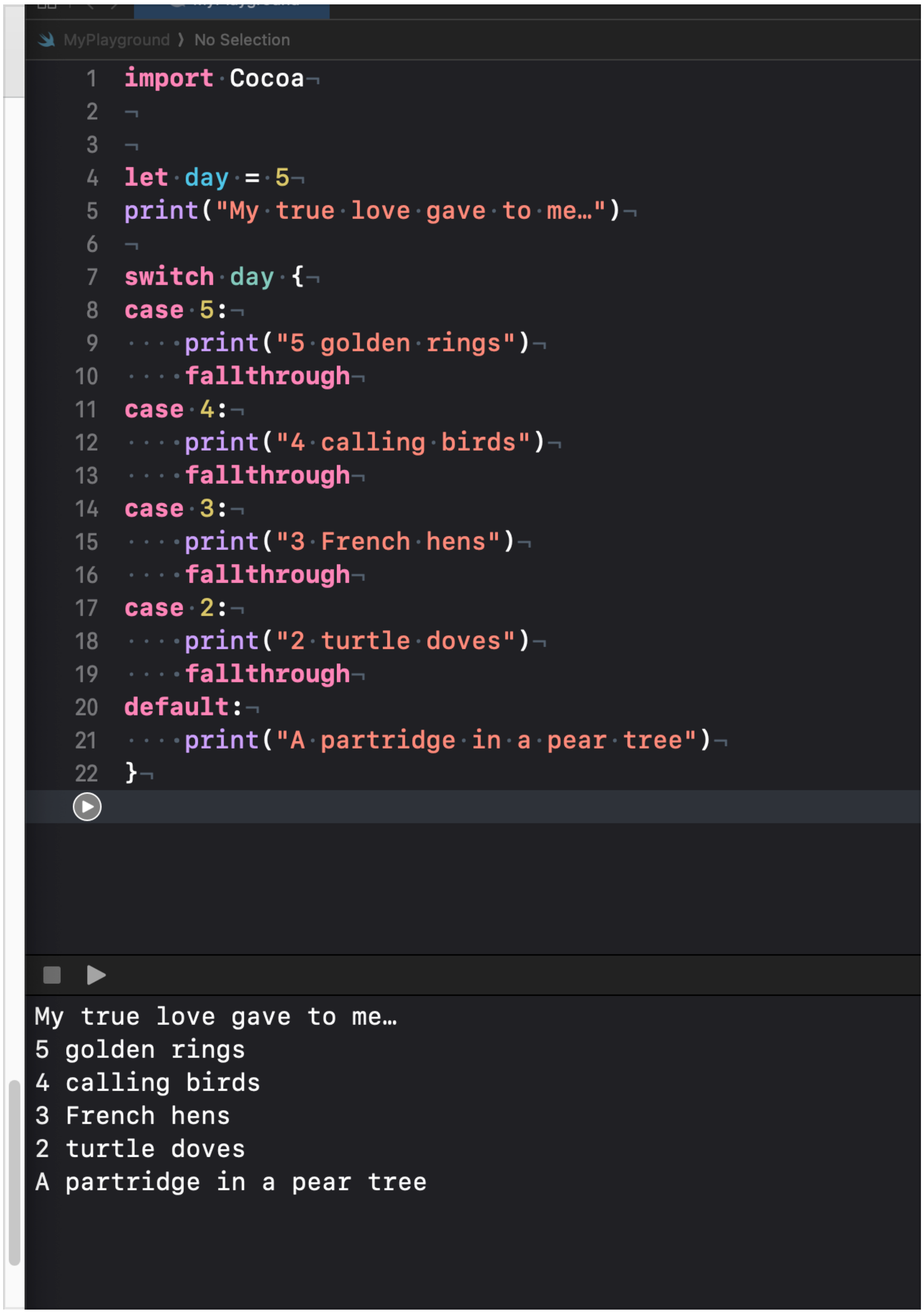
Remember: Swift checks its cases in order and runs the first one that matches. If you place `default` before any other case, that case is useless because it will never be matched and Swift will refuse to build your code.



We can use `fallthrough` to get exactly that behavior:

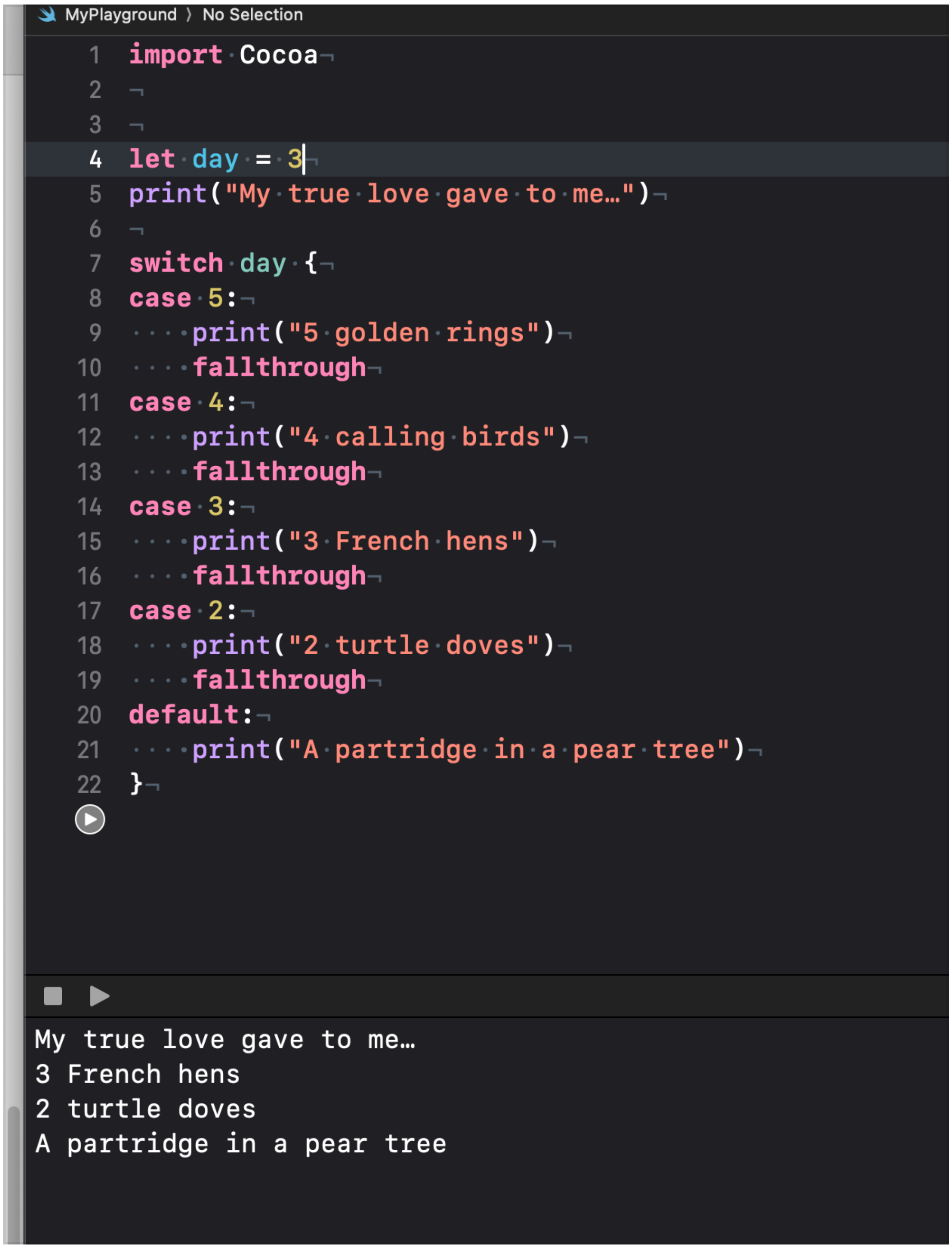
```swift
let day = 5
print("My true love gave to me…")

switch day {
case 5:
    print("5 golden rings")
    fallthrough
case 4:
    print("4 calling birds")
    fallthrough
case 3:
    print("3 French hens")
    fallthrough
case 2:
    print("2 turtle doves")
    fallthrough
default:
    print("A partridge in a pear tree")
}
```

That will match the first case and print “5 golden rings”, but the `fallthrough` line means `case 4` will execute and print “4 calling birds”, which in turn uses `fallthrough` again so that “3 French hens” is printed, and so on. It’s not a perfect match to the song, but at least you can see the functionality in action!



```swift
import Cocoa


let day = 5
print("My true love gave to me…")

switch day {
case 5:
    print("5 golden rings")
    fallthrough
case 4:
    print("4 calling birds")
    fallthrough
case 3:
    print("3 French hens")
    fallthrough
case 2:
    print("2 turtle doves")
    fallthrough
default:
    print("A partridge in a pear tree")
}
```

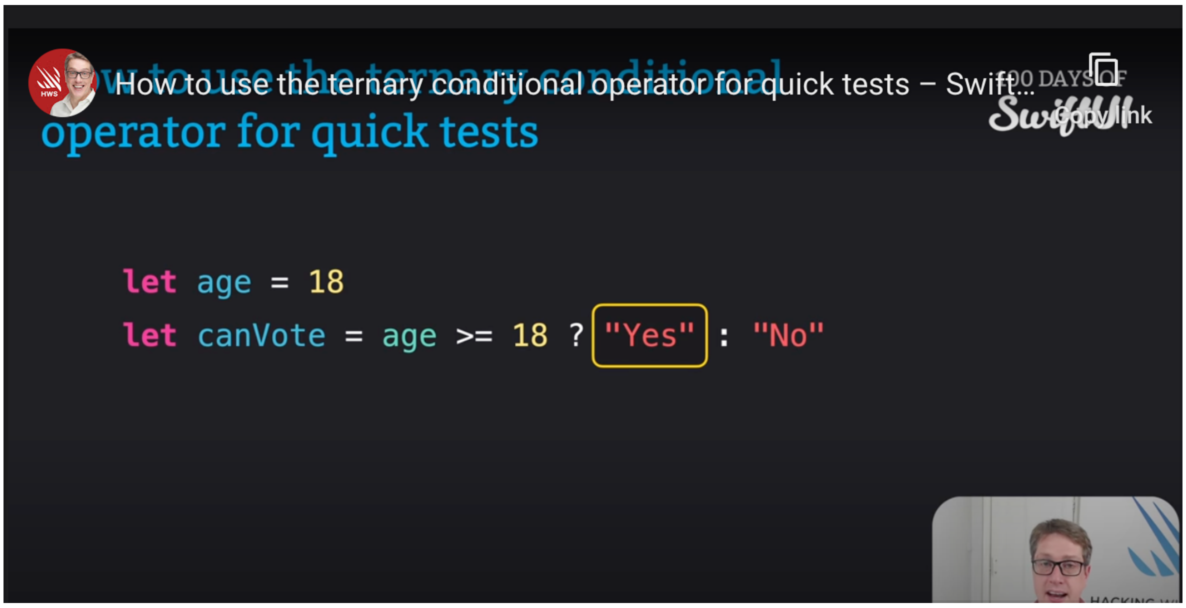
```
My true love gave to me…
5 golden rings
4 calling birds
3 French hens
2 turtle doves
A partridge in a pear tree
```



```swift
import Cocoa


let day = 3
print("My true love gave to me…")

switch day {
case 5:
    print("5 golden rings")
    fallthrough
case 4:
    print("4 calling birds")
    fallthrough
case 3:
    print("3 French hens")
    fallthrough
case 2:
    print("2 turtle doves")
    fallthrough
default:
    print("A partridge in a pear tree")
}
```

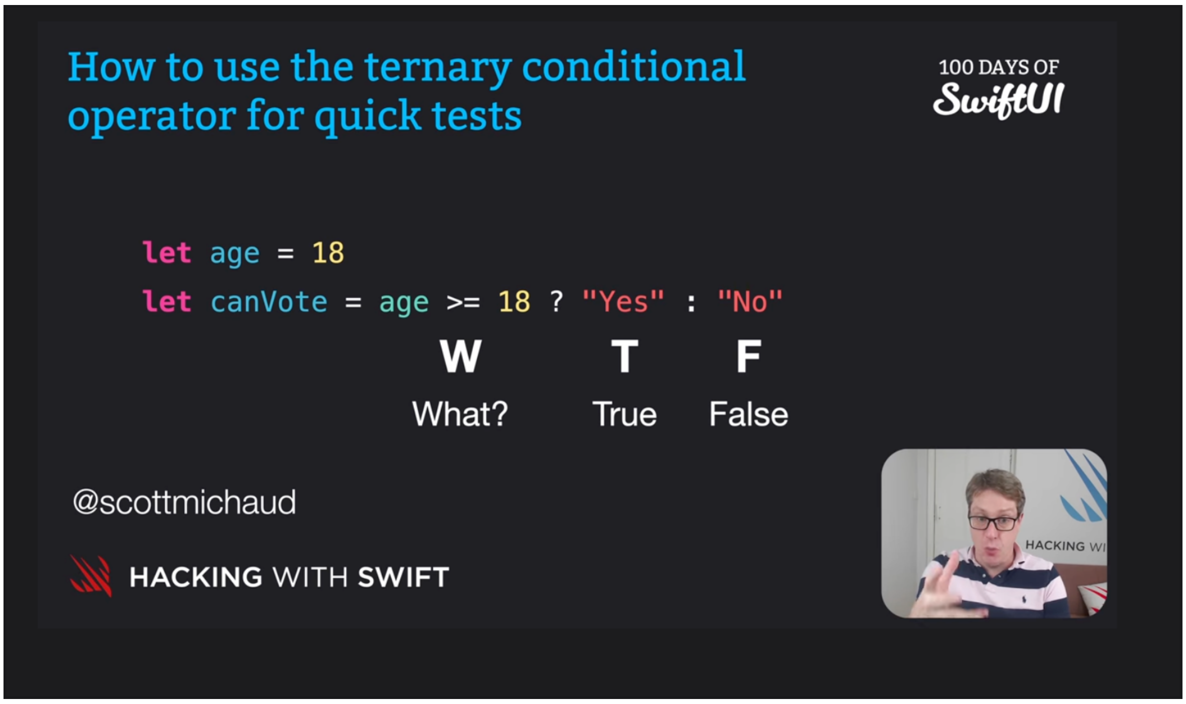
```
My true love gave to me…
3 French hens
2 turtle doves
A partridge in a pear tree
```

How to use the ternary conditional operator for quick tests


How to use the ternary conditional operator for quick tests – Swift...
Copy link
100 DAYS OF SwiftUI

How to use the ternary conditional operator for quick tests

```
let age = 18
let canVote = age >= 18 ? "Yes" : "No"
```

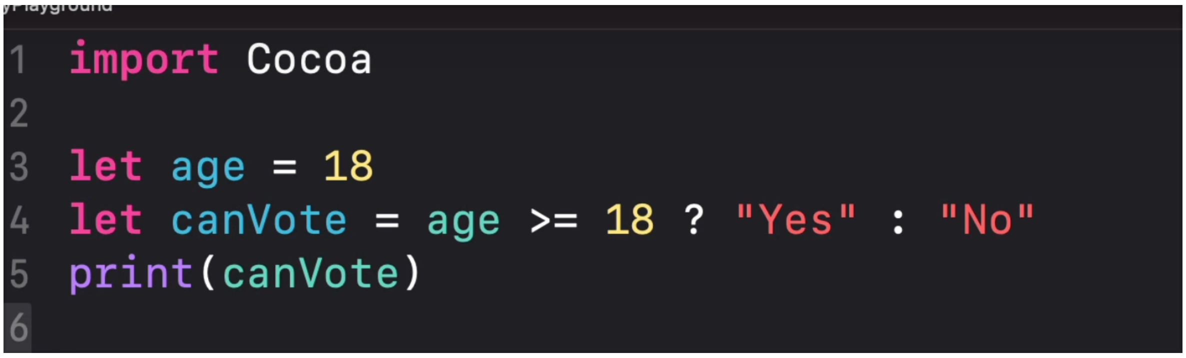


How to use the ternary conditional operator for quick tests

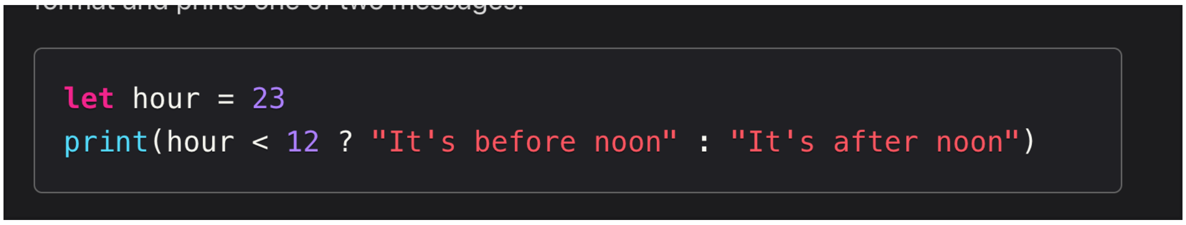
```
let age = 18
let canVote = age >= 18 ? "Yes" : "No"
```

W	T	F
What?	True	False

@scottmichaud

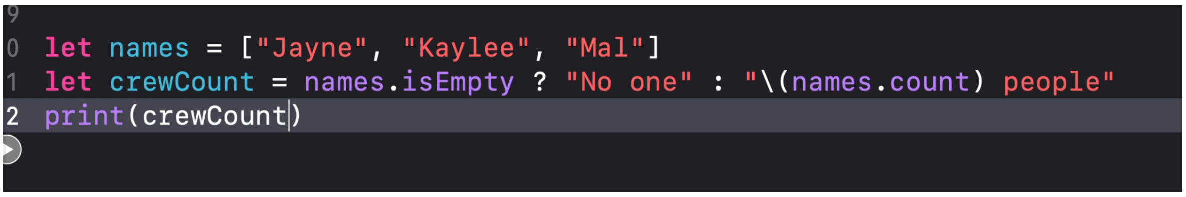


```swift
import Cocoa

let age = 18
let canVote = age >= 18 ? "Yes" : "No"
print(canVote)

```

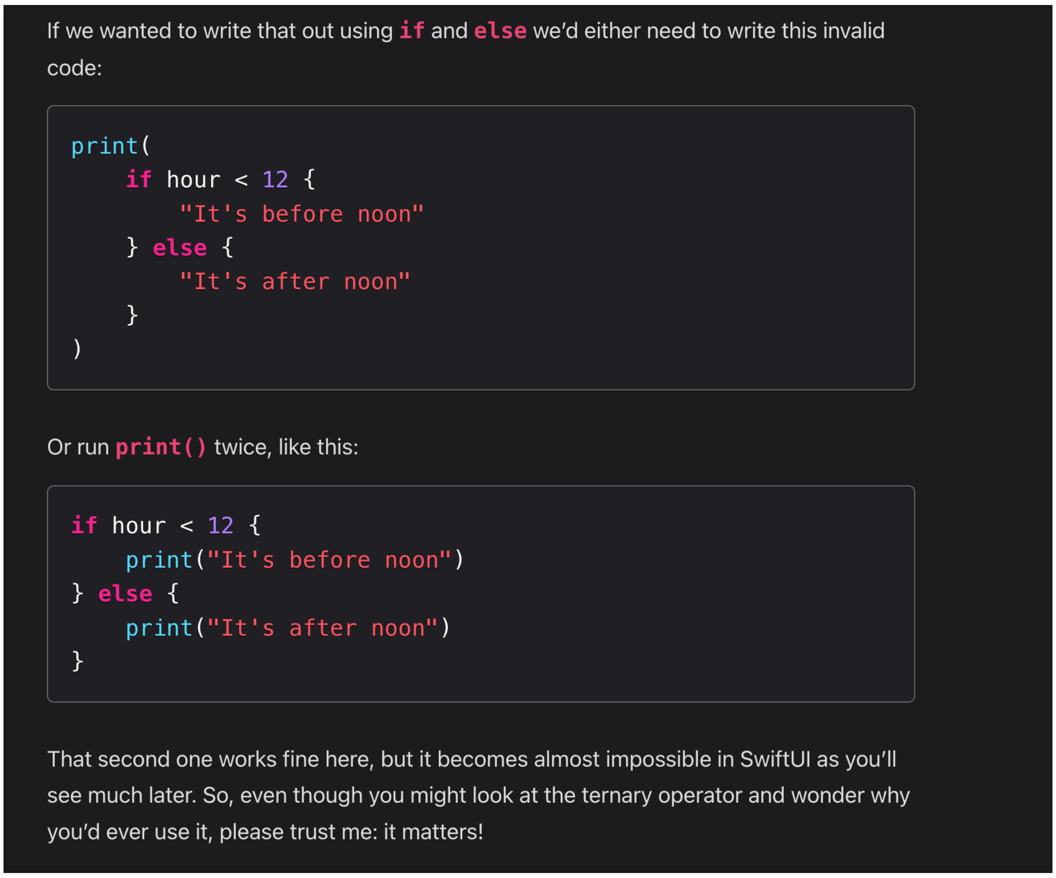


format and prints one of two messages:

```swift
let hour = 23
print(hour < 12 ? "It's before noon" : "It's after noon")
```



```swift
let names = ["Jayne", "Kaylee", "Mal"]
let crewCount = names.isEmpty ? "No one" : "\(names.count) people"
print(crewCount)
```


If we wanted to write that out using `if` and `else` we'd either need to write this invalid code:

```
print(
    if hour < 12 {
        "It's before noon"
    } else {
        "It's after noon"
    }
)
```

Or run `print()` twice, like this:

```
if hour < 12 {
    print("It's before noon")
} else {
    print("It's after noon")
}
```

That second one works fine here, but it becomes almost impossible in SwiftUI as you'll see much later. So, even though you might look at the ternary operator and wonder why you'd ever use it, please trust me: it matters!

The Second one works but the first one doesn't work