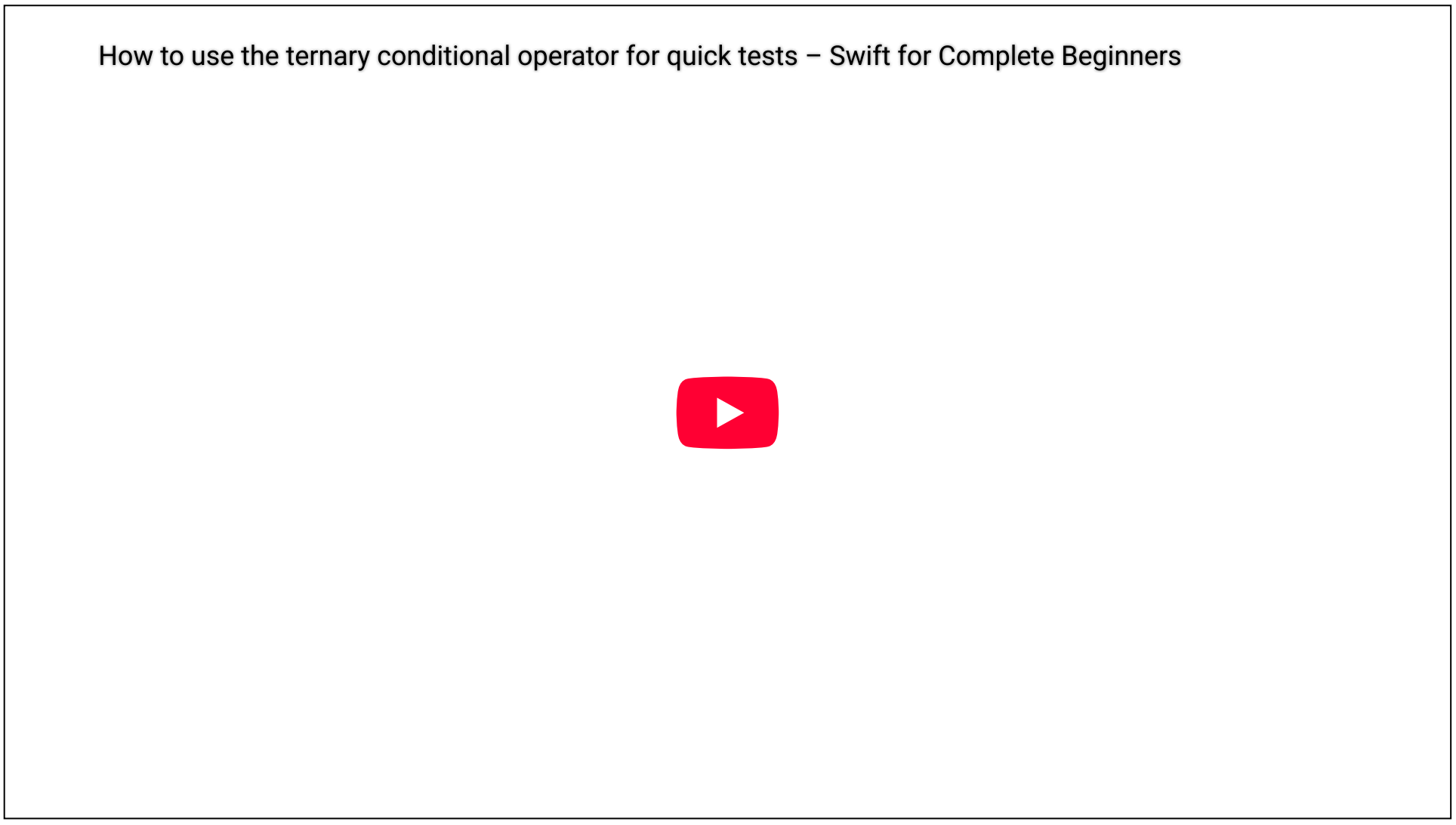
< How to use switch statements to check multiple conditions

How to use a for loop to repeat work >

How to use the ternary conditional operator for quick tests

Paul Hudson @twostraws October 4th 2021

Updated for Xcode 16.4



There's one last way to check conditions in Swift, and when you'll see it chances are you'll wonder when it's useful. To be fair, for a long time I very rarely used this approach, but as you'll see later it's really important with SwiftUI.

This option is called the *ternary conditional operator*. To understand why it has that name, you first need to know that `+`, `-`, `==`, and so on are all called *binary* operators because they work with two pieces of input: `2 + 5`, for example, works with 2 and 5.

Ternary operators work with *three* pieces of input, and in fact because the ternary conditional operator is the only ternary operator in Swift, you'll often hear it called just "the ternary operator."

Anyway, enough about names: what does this actually do? Well, the ternary operator lets us check a condition and return one of two values: something if the condition is true, and something if it's false.

For example, we could create a constant called `age` that stores someone's age, then create a second constant called `canVote` that will store whether that person is able to vote or not:

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

When that code runs, `canVote` will be set to "Yes" because `age` is set to 18.

As you can see, the ternary operator is split into three parts: a check (**`age >= 18`**), something for when the condition is true ("Yes"), and something for when the condition is false ("No"). That makes it exactly like a regular **`if`** and **`else`** block, in the same order.

If it helps, Scott Michaud suggested a helpful mnemonic: WTF. It stands for "what, true, false", and matches the order of our code:

- What is our condition? Well, it's **`age >= 18`**.
- What to do when the condition is true? Send back "Yes", so it can be stored in **`canVote`**.
- And if the condition is false? Send back "No".

Let's look at some other examples, start with an easy one that reads an hour in 24-hour format and prints one of two messages:

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

Notice how that doesn't assign the result anywhere – either the true or false case just gets printed depending on the value of **`hour`**.

Or here's one that reads the **`count`** of an array as part of its condition, then sends back one of two strings:

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

It gets a *little* hard to read when your condition use **`==`** to check for equality, as you can see here:

```
enum Theme {
    case light, dark
}

let theme = Theme.dark

let background = theme == .dark ? "black" : "white"
print(background)
```

The **`= theme ==`** part is usually the bit folks find hard to read, but remember to break it down:

- What? **`theme == .dark`**
- True: "black"
- False: "white"

So if theme is equal to **`.dark`** return "Black", otherwise return "White", then assign that to **`background`**.

Now, you might be wondering why the ternary operator is useful, particularly when we have regular **`if`**/**`else`** conditions available to us. I realize it's not a great answer, but you'll have to trust me on this: there are some times, particularly with SwiftUI, when we have no choice and *must* use a ternary.

You can see roughly what the problem is with our code to check hours:

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

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!

< How to use switch statements to check multiple conditions

How to use a for loop to repeat work >

Was this page useful? Let us know!

☆☆☆☆☆

Average rating: 4.8/5