< Checkpoint 2 How to check multiple conditions >

How to check a condition is true or false

Paul Hudson @twostraws October 25th 2021

Updated for Xcode 16.4

How to check a condition is true or false – Swift for Complete Beginners

Programs very often make *choices*:

- If the student's exam score was over 80 then print a success message.
- If the user entered a name that comes after their friend's name alphabetically, put the friend's name first.
- If adding a number to an array makes it contain more than 3 items, remove the oldest one.
- If the user was asked to enter their name and typed nothing at all, give them a default name of "Anonymous".

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:

```
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.

Of course, that isn't *everything* in the code: I didn't mention the little **{** and **}** symbols. These are called *braces* – opening and closing braces, more specifically – although sometimes you'll hear them referred to as "curly braces" or "curly brackets".

These braces are used extensively in Swift to mark blocks of code: the opening brace starts the block, and the closing brace ends it. *Inside* the code block is all the code we want to run if our condition happens to be true when it's checked, which in our case is printing a message.

You can include as much code in there as you want:

```
if someCondition {
    print("Do something")
    print("Do something else")
    print("Do a third thing")
}
```

Of course, what really matters is the **someCondition** part, because that's where your checking code comes in: what condition do you actually want to check?

Well, let's try the score example: if a **score** constant is over 80, let's print a message. Here's how that would look in code:

```
let score = 85

if score > 80 {
    print("Great job!")
}
```

In that code, **score > 80** is our condition. You'll remember **>** from school meaning "is greater than", so our complete condition is "if score is greater than 80." And if it *is* greater than 80, "Great job!" will be printed – nice!

That **>** symbol is a comparison operator, because it compares two things and returns a Boolean result: is the thing on the left greater than the thing on the right? You can also use **<** for less than, **>=** for "greater than or equal", and **<=** for "less than or equal."

Let's try it out – what do you think this code will print?

```
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")
}
```

Try and run the code mentally in your head – which **print()** lines will actually be run?

Well, our first one will run if **speed** is greater than or equal to 88, and because it is *exactly* 88 the first **print()** code will be run.

The second one will run if **percentage** is less than 85, and because it is exactly 85 the second **print()** will *not* run – we used less than, not less than or equal.

The third will run if **age** is greater than or equal to 18, and because it's exactly 18 the third **print()** *will* run.

Now let's try our second example condition: if the user entered a name that comes after their friend's name alphabetically, put the friend's name first. You've seen how **<**, **>=** and others work great with numbers, but they also work equally well with strings right out of the box:

```
let ourName = "Dave Lister"
let friendName = "Arnold Rimmer"

if ourName < friendName {
    print("It's \(ourName) vs \(friendName)")
}

if ourName > friendName {
    print("It's \(friendName) vs \(ourName)")
}
```

So, if the string inside **ourName** comes before the string inside **friendName** when sorted alphabetically, it prints **ourName** first then **friendName**, exactly as we wanted.

Let's take a look at our third example condition: if adding a number to an array makes it contain more than 3 items, remove the oldest one. You've already met **append()**, **count**, and **remove(at:)**, so we can now put all three together with a condition like this:

```
// Make an array of 3 numbers
var numbers = [1, 2, 3]

// Add a 4th
numbers.append(4)

// If we have over 3 items
if numbers.count > 3 {
    // Remove the oldest number
    numbers.remove(at: 0)
}

// Display the result
print(numbers)
```

Now let's look at our fourth example condition: if the user was asked to enter their name and typed nothing at all, give them a default name of "Anonymous".

To solve this you'll first need to meet two other comparison operators you'll use a lot, both of which handle equality. The first is **==** and means "is equal to," which is used like this:

```
let country = "Canada"

if country == "Australia" {
    print("G'day!")
}
```

The second is **!=**, which means “is not equal to”, and is used like this:

```
let name = "Taylor Swift"

if name != "Anonymous" {
    print("Welcome, \(name)")
}
```

In our case, we want to check whether the username entered by the user is empty, which we could do like this:

```
// 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)!")
```

That **""** is an empty string: we start the string and end the string, with nothing in between. By comparing **username** to that, we’re checking if the user also entered an empty string for their username, which is exactly what we want.

Now, there are other ways of doing this check, and it’s important you understand what they do.

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

```
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"
}
```

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.

< Checkpoint 2

How to check multiple conditions >

Was this page useful? Let us know!

☆☆☆☆☆

Average rating: 4.6/5

You are not logged in

Log in or create account