< How to check a condition is true or false

How to use switch statements to check multiple conditions >

How to check multiple conditions

Paul Hudson @twostraws November 24th 2021

Updated for Xcode 16.4

When we use `if` we must provide Swift some kind of condition that will either be true or false once it has been evaluated. If you want to check for several different values, you can place them one after the other like this:

```
let age = 16

if age >= 18 {
    print("You can vote in the next election.")
}

if age < 18 {
    print("Sorry, you're too young to vote.")
}
```

However, that's not very efficient if you think about it: our two conditions are mutually exclusive, because if someone is greater than or equal to 18 (the first condition) then they can't be less than 18 (the second condition), and the opposite is also true. We're making Swift do work that just isn't needed.

In this situation, Swift provides us with a more advanced condition that lets us add an `else` block to our code – some code to run if the condition is *not* true.

Using `else` we could rewrite our previous code to this:

```
let age = 16

if age >= 18 {
    print("You can vote in the next election.")
} else {
    print("Sorry, you're too young to vote.")
}
```

Now Swift only needs to check **age** once: if it's greater than or equal to 18 the first **print()** code is run, but if it's any value *less* than 18 the second **print()** code is run.

So, now our condition looks like this:

```
if someCondition {
    print("This will run if the condition is true")
} else {
    print("This will run if the condition is false")
}
```

There's an even *more* advanced condition called **else if**, which lets you run a new check if the first one fails. You can have just one of these if you want, or have multiple **else if**, and even combine **else if** with an **else** if needed. However, you can only ever have one **else**, because that means "if all the other conditions have been false."

Here's how that looks:

```
let a = false
let b = true

if a {
    print("Code to run if a is true")
} else if b {
    print("Code to run if a is false but b is true")
} else {
    print("Code to run if both a and b are false")
}
```

You can keep on adding more and more **else if** conditions if you want, but watch out that your code doesn't get too complicated!

As well as using **else** and **else if** to make more advanced conditions, you can also check more than one thing. For example, we might want to say "if today's temperature is over 20 degrees Celsius but under 30, print a message."

This has two conditions, so we could write this:

```
let temp = 25

if temp > 20 {
    if temp < 30 {
        print("It's a nice day.")
    }
}
```

Although that works well enough, Swift provides a shorter alternative: we can use **&&** to combine two conditions together, and the whole condition will only be true if the two parts inside the condition are true.

So, we could change our code to this:

```
if temp > 20 && temp < 30 {
    print("It's a nice day.")
}
```

You should read **&&** as "and", so our whole conditions reads "if temp is greater than 20 and temp is less than 30, print a message." It's called a *logical operator* because it combines Booleans to make a new Boolean.

&& has a counterpart that is two pipe symbols, **||**, which means "or". Whereas **&&** will only make a condition be true if both subconditions are true, **||** will make a condition be true if *either* subcondition is true.

For example, we could say that a user can buy a game if they are at least 18, or if they are under 18 they must have permission from a parent. We could write that using **||** like so:

```
let userAge = 14
let hasParentalConsent = true

if userAge >= 18 || hasParentalConsent == true {
    print("You can buy the game")
}
```

That will print "You can buy the game", because although the first half of our condition fails – the user is *not* at least 18 – the second half passes, because they *do* have parental consent.

Remember, using **== true** in a condition can be removed, because we're obviously already checking a Boolean. So, we could write this instead:

```
if userAge >= 18 || hasParentalConsent {
    print("You can buy the game")
}
```

To finish up with checking multiple conditions, let's try a more complex example that combines **if**, **else if**, **else**, and **||** all at the same time, and even shows off how enums fit into conditions.

In this example we're going to create an enum called **TransportOption**, which contains five cases: airplane, helicopter, bicycle, car, and scooter. We'll then assign an example value to a constant, and run some checks:

- If we are going somewhere by airplane or by helicopter, we'll print "Let's fly!"
- If we're going by bicycle, we'll print "I hope there's a bike path..."
- If we're going by car, we'll print "Time to get stuck in traffic."
- Otherwise we'll print "I'm going to hire a scooter now!"

Here's the code for that:

```
enum TransportOption {
    case airplane, helicopter, bicycle, car, scooter
}

let transport = TransportOption.airplane

if transport == .airplane || transport == .helicopter {
    print("Let's fly!")
} else if transport == .bicycle {
    print("I hope there's a bike path…")
} else if transport == .car {
    print("Time to get stuck in traffic.")
} else {
    print("I'm going to hire a scooter now!")
}
```

I'd like to pick out a few parts of that code:

1. When we set the value for `transport` we need to be explicit that we're referring to `TransportOption.airplane`. We can't just write `.airplane` because Swift doesn't understand we mean the `TransportOption` enum.
2. Once that has happened, we don't need to write `TransportOption` any more because Swift knows `transport` must be some kind of `TransportOption`. So, we can check whether it's equal to `.airplane` rather than `TransportOption.airplane`.
3. The code using `||` to check whether `transport` is equal to `.airplane` *or* equal to `.helicopter`, and if *either* of them are true then the condition is true, and "Let's fly!" is printed.
4. If the first condition fails – if the transport mode isn't `.airplane` or `.helicopter` – then the second condition is run: is the transport mode `.bicycle`? If so, "I hope there's a bike path…" is printed.
5. If we aren't going by bicycle either, then we check whether we're going by car. If we are, "Time to get stuck in traffic." is printed.
6. Finally, if all the previous conditions fail then the `else` block is run, and it means we're going by scooter.

< How to check a condition is true or false

How to use switch statements to check multiple conditions >

Was this page useful? Let us know!

☆☆☆☆☆

Average rating: 4.9/5