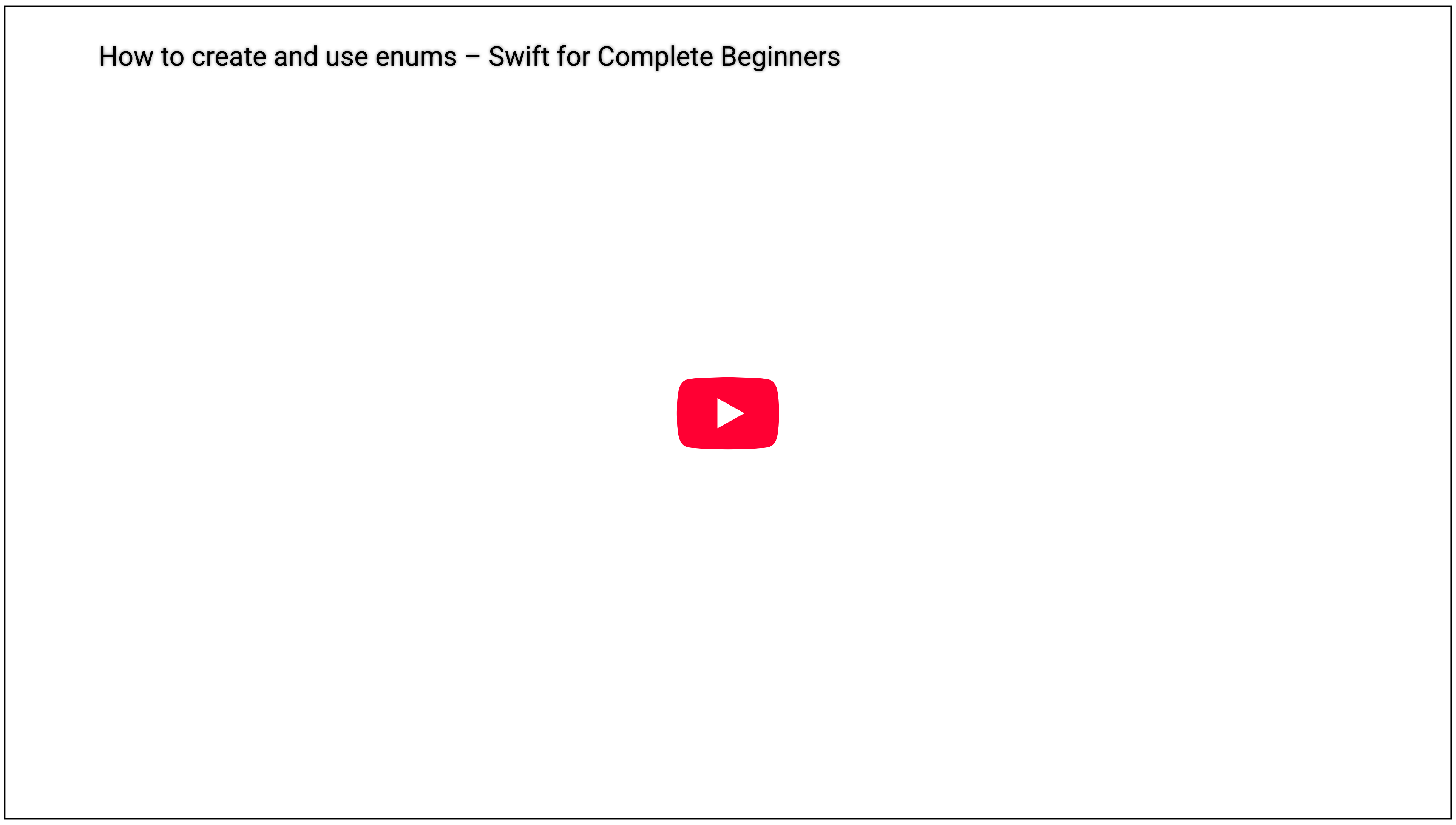
< How to use sets for fast data lookup How to use type annotations >

How to create and use enums

Paul Hudson @twostraws November 1st 2021

Updated for Xcode 16.4


How to create and use enums – Swift for Complete Beginners

An enum – short for *enumeration* – is a set of named values we can create and use in our code. They don't have any special meaning to Swift, but they are more efficient and safer, so you'll use them a lot in your code.

To demonstrate the problem, let's say you wanted to write some code to let the user select a day of the week. You might start out like this:

```
var selected = "Monday"
```

Later on in your code you change it, like so:

```
selected = "Tuesday"
```

That might work well in very simple programs, but take a look at this code:

```
selected = "January"
```

Oops! You accidentally typed in a month rather than a day – what will your code do? Well, you might be lucky enough to have a colleague spot the error as they review your code, but how about this:

```
selected = "Friday "
```

That has a space at the end of Friday, and “Friday ” with a space is different from “Friday” without a space in Swift’s eyes. Again, what would your code do?

Using strings for this kind of thing takes some very careful programming, but it’s also pretty inefficient – do we really need to store all the letters of “Friday” to track one single day?

This is where enums come in: they let us define a new data type with a handful of specific values that it can have. Think of a Boolean, that can only have true or false – you can’t set it to “maybe” or “probably”, because that isn’t in the range of values it understands. Enums are the same: we get to list up front the range of values it can have, and Swift will make sure you never make a mistake using them.

So, we could rewrite our weekdays into a new enum like this:

```
enum Weekday {
    case monday
    case tuesday
    case wednesday
    case thursday
    case friday
}
```

That calls the new enum `Weekday`, and provides five cases to handle the five weekdays.

Now rather than using strings, we would use the enum. Try this in your playground:

```
var day = Weekday.monday
day = Weekday.tuesday
day = Weekday.friday
```

With that change you can’t accidentally use “Friday ” with an extra space in there, or put a month name instead – you must always choose one of the possible days listed in the enum. You’ll even see Swift offer up all possible options when you’ve typed `Weekday.`, because it knows you’re going to select one of the cases.

Swift does two things that make enums a little easier to use. First, when you have many cases in an enum you can just write `case` once, then separate each case with a comma:

```
enum Weekday {
    case monday, tuesday, wednesday, thursday, friday
}
```

Second, remember that once you assign a value to a variable or constant, its data type becomes fixed – you can’t set a variable to a string at first, then an integer later on. Well, for enums this means you can skip the enum name after the first assignment, like this:

```
var day = Weekday.monday
day = .tuesday
day = .friday
```

Swift knows that `.tuesday` must refer to `Weekday.tuesday` because `day` must always be some kind of `Weekday`.

Although it isn't visible here, one major benefit of enums is that Swift stores them in an optimized form – when we say `Weekday.monday` Swift is likely to store that using a single integer such as 0, which is much more efficient to store and check than the letters M, o, n, d, a, y.

[< How to use sets for fast data lookup](#) [How to use type annotations >](#)

Was this page useful? Let us know!

☆☆☆☆☆

Average rating: 4.7/5

Twitter

Mastodon

Email

Sponsor the site

About Glossary Code License Privacy Policy Refund Policy Update Policy Code of Conduct

You are not logged in

Log in or create account