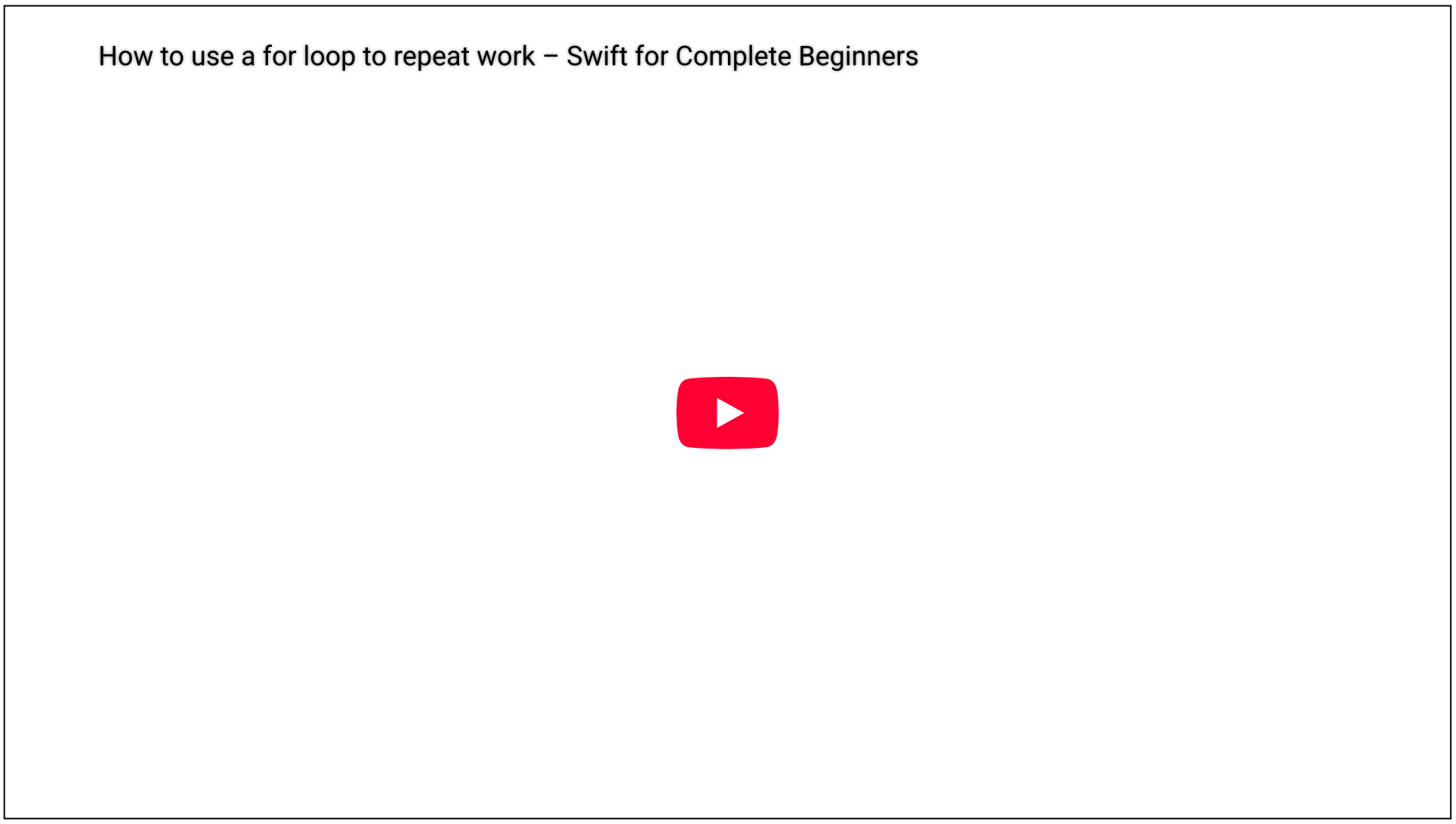
< How to use the ternary conditional operator for quick tests

How to use a while loop to repeat work >

How to use a for loop to repeat work

Paul Hudson @twostraws October 25th 2021

Updated for Xcode 16.4



Computers are really great at doing repetitive work, and Swift makes it easy to repeat some code a fixed number of times, or once for every item in an array, dictionary, or set.

Let's start with something simple: if we have an array of strings, we can print each string out like this:

```
let platforms = ["iOS", "macOS", "tvOS", "watchOS"]

for os in platforms {
    print("Swift works great on \(os).")
}
```

That loops over all the items in **`platforms`**, putting them one by one into **`os`**. We haven't created **`os`** elsewhere; it's created for us as part of the loop and made available only inside the opening and closing braces.

Inside the braces is the code we want to run for each item in the array, so the code above will print four lines – one for each loop item. First it puts "iOS" in there then calls **`print()`**, then it puts "macOS" in there and calls **`print()`**, then "tvOS", then "watchOS".

To make things easier to understand, we give these things common names:

- We call the code inside the braces the *loop body*
- We call one cycle through the loop body a *loop iteration*.

- We call **os** the *loop variable*. This exists only inside the loop body, and will change to a new value in the next loop iteration.

I should say that the name **os** isn't special – we could have written this instead:

```
for name in platforms {
    print("Swift works great on \(name).")
}
```

Or even this:

```
for rubberChicken in platforms {
    print("Swift works great on \(rubberChicken).")
}
```

The code will still work exactly the same.

In fact, Xcode is really smart here: if you write **for plat** it will recognize that there's an array called **platforms**, and offer to autocomplete all of **for platform in platforms** – it recognizes that **platforms** is plural and suggests the singular name for the loop variable. When you see Xcode's suggestion appear, press Return to select it.

Rather than looping over an array (or set, or dictionary – the syntax is the same!), you can also loop over a fixed range of numbers. For example, we could print out the 5 times table from 1 through 12 like this:

```
for i in 1...12 {
    print("5 x \(i) is \(5 * i)")
}
```

A couple of things are new there, so let's pause and examine them:

- I used the loop variable **i**, which is a common coding convention for "number you're counting with". If you're counting a second number you would use **j**, and if you're counting a third you would use **k**, but if you're counting a fourth maybe you should pick better variable names.
- The **1...12** part is a *range*, and means "all integer numbers between 1 and 12, as well as 1 and 12 themselves." Ranges are their own unique data type in Swift.

So, when that loop first runs **i** will be 1, then it will be 2, then 3, etc, all the way up to 12, after which the loop finishes.

You can also put loops inside loops, called *nested loops*, like this:

```
for i in 1...12 {
    print("The \(i) times table:")

    for j in 1...12 {
        print("  \(j) x \(i) is \(j * i)")
    }

    print()
}
```

That shows off a couple of other new things, so again let's pause and look closer:

- There's now a nested loop: we count from 1 through 12, and for each number inside there we count 1 through 12 again.
- Using `print()` by itself, with no text or value being passed in, will just start a new line. This helps break up our output so it looks nicer on the screen.

So, when you see `x...y` you know it creates a range that starts at whatever number `x` is, and counts up to and including whatever number `y` is.

Swift has a similar-but-different type of range that counts up to but *excluding* the final number: `..<`. This is best seen in code:

```
for i in 1...5 {
    print("Counting from 1 through 5: \(i)")
}

print()

for i in 1..<5 {
    print("Counting 1 up to 5: \(i)")
}
```

When that runs, it will print for numbers 1, 2, 3, 4, 5 in the first loop, but only numbers 1, 2, 3, and 4 in the second. I pronounce `1...5` as "one through five", and `1..<5` as "one up to five," and you'll see similar wording elsewhere in Swift.

Tip: `..<` is really helpful for working with arrays, where we count from 0 and often want to count up to but excluding the number of items in the array.

Before we're done with `for` loops, there's one more thing I want to mention: sometimes you want to run some code a certain number of times using a range, but you don't actually want the loop variable – you don't want the `i` or `j`, because you don't use it.

In this situation, you can replace the loop variable with an underscore, like this:

```
var lyric = "Haters gonna"

for _ in 1...5 {
    lyric += " hate"
}

print(lyric)
```

(Yes, that's a Taylor Swift lyric from Shake It Off, written in Swift.)

< How to use the ternary conditional operator for quick tests

How to use a while loop to repeat work >

Was this page useful? Let us know!

☆☆☆☆☆

Average rating: 4.7/5

Twitter

Mastodon

Email

Sponsor the site

You are not logged in

Log in or create account