< How to create variables and constants

How to store whole numbers >

How to create strings

Paul Hudson @twostraws November 25th 2021

Updated for Xcode 16.4

When you assign text to a constant or variable, we call that a *string* – think of a bunch of Scrabble tiles threaded onto a string to make a word.

Swift's strings start and end with double quotes, but what you put inside those quotes is down to you. You can use short pieces of alphabetic text, like this:

```
let actor = "Denzel Washington"
```

You can use punctuation, emoji and other characters, like this:

```
let filename = "paris.jpg"
let result = "⭐ You win! ⭐"
```

And you can even use other double quotes inside your string, as long as you're careful to put a backslash before them so that Swift understands they are *inside* the string rather than *ending* the string:

```
let quote = "Then he tapped a sign saying \"Believe\" and walked away."
```

Don't worry – if you miss off the backslash, Swift will be sure to shout loudly that your code isn't quite right.

There is no realistic limit to the length of your strings, meaning that you could use a string to store something very long such as the complete works of Shakespeare. However, what you'll find is that Swift doesn't like line breaks in its strings. So, this kind of code isn't allowed:

```
let movie = "A day in
the life of an
Apple engineer"
```

That doesn't mean you *can't* create strings across multiple lines, just that Swift needs you to treat them specially: rather than one set of quotes on either side of your string, you use three, like this:

```
let movie = """
A day in
the life of an
Apple engineer
"""
```

These multi-line strings aren't used terribly often, but at least you can see how it's done: the triple quotes at the start and end are on their own line, with your string in between.

Once you've created your string, you'll find that Swift gives us some useful functionality to work with its contents. You'll learn more about this functionality over time, but I want to introduce you to three things here.

First, you can read the length of a string by writing **`.count`** after the name of the variable or constant:

```
print(actor.count)
```

Because **`actor`** has the text "Denzel Washington", that will print 17 – one for each letter in the name, plus the space in the middle.

You don't need to print the length of a string directly if you don't want to – you can assign it to another constant, like this:

```
let nameLength = actor.count
print(nameLength)
```

The second useful piece of functionality is **`uppercased()`**, which sends back the same string except every one of its letter is uppercased:

```
print(result.uppercased())
```

Yes, the open and close parentheses are needed here but *aren't* needed with **`count`**. The reason for this will become clearer as you learn, but at this early stage in your Swift learning the distinction is best explained like this: if you're asking Swift to read some data you don't need the parentheses, but if you're asking Swift to do some work you *do*. That's not wholly true as you'll learn later, but it's enough to get you moving forward for now.

The last piece of helpful string functionality is called **`hasPrefix()`**, and lets us know whether a string starts with some letters of our choosing:

```
print(movie.hasPrefix("A day"))
```

There's also a **`hasSuffix()`** counterpart, which checks whether a string ends with some text:

```
print(filename.hasSuffix(".jpg"))
```

Important: Strings are case-sensitive in Swift, which means using **`filename.hasSuffix(".JPG")`** will return false because the letters in the string are lowercase.

Strings are really powerful in Swift, and we've only really scratched the surface of what they can do!

< How to create variables and constants

How to store whole numbers >

Was this page useful? Let us know!

☆☆☆☆☆

Average rating: 4.8/5