HACKING WITH SWIFT

UNDERSTANDING SWIFT

COMPLETE REFERENCE GUIDE

Answers to the most common questions

FREE SAMPLE

Paul Hudson

Chapter 1
Simple types

Why does Swift have variables?

Variables allow us to store temporary information in our program, and form a key part of almost every Swift program. Ultimately, your program is going to transform data somehow: maybe you let the user enter in todo list tasks then check them off, maybe you let them roam around a deserted Island working for a capitalist raccoon, or maybe you read the device time and display it in a clock. Regardless, you're taking some sort of data, transforming it somehow, and showing it to the user.

Of course, the "transforming it somehow" is where the real magic comes in, because that's the part where your brilliant app idea happens. But the process of storing data in memory – holding on to something the user typed, or something you downloaded from the internet – is where variables come in.

Once you create a variable using **var**, you can change it as often as you want without using **var** again. For example:

```
var favoriteShow = "Orange is the New Black"
favoriteShow = "The Good Place"
favoriteShow = "Doctor Who"
```

If it helps, try reading **var** as "create a new variable". So, the first line above might be read out loud as "create a new variable called **favoriteShow** and give it the value Orange is the New Black." Lines 2 and 3 don't have **var** in there, so they modify the existing value rather than creating a new variable.

Now imagine you had **var** on all three lines – you used **var favoriteShow** each time. That wouldn't make much sense, because you'd be saying "create a new variable called **favoriteShow**" three times over, and the variable is clearly not new after your first attempt. Swift will flag this as an error, which means it won't let you run your code until you pick a different name for your variables.

That might seem like annoying behavior, but trust me: it's helpful! Swift wants you to be clear:

are you trying to modify an existing variable (if so, remove the **var** the second and subsequent times), or are you trying to create a new variable (in which case, name it something else.)

One last thing: although variables form the core of many Swift programs, you will learn that sometimes they are best avoided. More on that later!

Why is Swift a type-safe language?

Swift lets us create variables as strings and integers, but also many other types of data too. When you create a variable Swift can figure out what type the variable is based on what kind of data you assign to it, and from then on that variable will always have that one specific type.

For example, this creates a new variable called **meaningOfLife** equal to 42:

```
var meaningOfLife = 42
```

Because we assigned 42 as the initial value of **meaningOfLife**, Swift will assign it the type *integer* – a whole number. It's a variable, which means we can change its value as often as we need to, but we *can't* change its value: it will always be an integer.

This is *extremely* helpful when building apps, because it means Swift will make sure we don't make mistakes with our data. For example, we can't write this:

```
meaningOfLife = "Forty two"
```

That tries to assign a string to a variable that is an integer, which isn't allowed. Although we rarely make such an obvious mistake, you *will* find that this type safety helps out every single day you're writing code with Swift.

Think about it: we just created one variable then tried to change its type, which will obviously fail. But as your programs grow in size and complexity, it becomes impossible to keep the types of your variables in your head at all times, so we're effectively shifting that work on to Swift instead.

Why does Swift have multi-line strings

Swift's standard strings start and end with quotes, but must never contain any line breaks. For example, this is a standard string:

```
var quote = "Change the world by being yourself"
```

That works fine for small pieces of text, but becomes ugly in source code if you have lots of text you want to store. That's where multi-line strings come in: if you use triple quotes you can write your strings across as many lines as you need, which means the text remains easy to read in your code:

```
var burns = """
The best laid schemes
O' mice and men
Gang aft agley
"""
```

Swift sees those line breaks in your string as being part of the text itself, so that string actually contains three lines.

Tip: Sometimes you will want to have long strings of text in your code without using multiple lines, but this is quite rare. Specifically, this is most commonly important if you plan to share your code with others – if they see an error message in your program they might want to search your code for it, and if you've split it across multiple lines their search might fail.

Why does Swift need both Doubles and Integers?

Swift gives us several different ways of storing numbers in our code, and they are designed to solve different problems. Swift doesn't let us mix them together because doing so will (as in, 100% guaranteed) lead to problems.

The two main types of numbers you'll use are called integers and doubles. Integers hold whole numbers, such as 0, 1, -100, and 65 million, whereas doubles hold decimal numbers, such as 0.1, -1.001, and 3.141592654.

When create a numeric variable, Swift decides whether to consider it an integer or a double based on whether you include a decimal point. For example:

```
var myInt = 1
var myDouble = 1.0
```

As you can see, they both contain the number 1, but the former is an integer and the latter a double.

Now, if they both contain the number 1, you might wonder why we can't add them together – why can't we write **var total = myInt + myDouble**? The answer is that Swift is playing it safe: we can both see that 1 plus 1.0 will be 2, but your double is a variable so it could be modified to be 1.1 or 3.5 or something else. How can Swift be sure it's safe to add an integer to a double – how can it be sure you won't lose the 0.1 or 0.5?

The answer is that it *can't* be safe, which is why it isn't allowed. This will annoy you at first, but trust me: it's helpful.

Why does Swift have string interpolation?

When it comes time to showing information to your user – whether that's messages being printed out, text on buttons, or whatever fits your app idea – you will usually rely heavily on strings.

Of course, we don't just want static strings, because we want to show the user some sort of relevant data they can use. So, Swift gives us string interpolation as a way of injecting custom data into strings at runtime: it replaces one or more parts of a string with data provided by us.

For example:

```
var city = "Cardiff"
var message = "Welcome to \(city)!"
```

Of course, in that trivial example we could have just written our city name directly into the string – "Welcome to Cardiff!" – but in real apps having this inserted dynamically is important because it lets us show real-world user data rather than things we typed ourselves.

Swift is capable of placing any kind of data inside string interpolation. The result might not always be useful, but for all of Swift's basic types – strings, integers, Booleans, etc – the results are great.

Tip: String interpolation is extremely powerful in Swift. If you're keen to see just what it can do, check out this more advanced blog post from me: **https://www.hackingwithswift.com/articles/178/super-powered-string-interpolation-in-swift-5-0**

Why does Swift have constants as well as variables?

Variables are a great way to store temporary data in your programs, but Swift gives us a second option that's even better: constants. They are identical to variables in every way, with one important difference: we can't change their values once they are set.

Swift really loves constants, and in fact will recommend you use one if you created a variable then never changed its value. The reason for this is about avoiding problems: any variable you create can be changed by you whenever you want and as often as you want, so you lose some control – that important piece of user data you stashed away might be removed or replaced at any point in the future.

Constants don't let us change values once they are set, so it's a bit like a contract with Swift: you're saying "this value matters, don't let me change it no matter what I do." Sure, you could try to make the same contract with a variable, but one slip of your keyboard could screw things up and Swift wouldn't be able to help. By using a constant instead – just by changing **var** to **let** – you're asking Swift to make sure the value never changes, which is another thing you no longer need to worry about.

Why does Swift have type annotations?

A common question folks ask when learning Swift is “why does Swift have type annotations?”, which is usually followed by “when should I use type annotations in Swift?”

The answer to the first question is primarily one of three reasons:

1. Swift can’t figure out what type should be used.
2. You want Swift to use a different type from its default.
3. You don’t want to assign a value just yet.

The first of those will usually happen only in more advanced code. For example, if you were loading some data from the internet that you know happens to be the name of your local politician, Swift can’t know that ahead of time so you’ll need to tell it.

The second scenario is quite common as you learn more in Swift, but right now a simple example is trying to create a double variable without having to constantly write “.0” everywhere:

```
var percentage: Double = 99
```

That makes **percentage** a double with the value of 99.0. Yes, we have assigned an integer to it, but our type annotation makes it clear that the actual data type we want is double.

The third option happens when you want to tell Swift that a variable is going to exist, but you don’t want to set its value just yet. This happens in lots of places in Swift, and looks like this:

```
var name: String
```

You can then assign a string to **name** later on, but you can’t assign a different type because Swift knows it would be invalid.

Of course, the second question here is “when should I use type annotations in Swift?” This is much more subjective, because the answer usually depends on your personal style.

In my own code, I prefer to use type inference as much as possible. That means I leave off the type annotations, and let Swift figure out the type of things based on what data I store in them. My reasons for this are:

1. It makes my code shorter and easier to read.
2. It allows me to change the type of something just by changing whatever is its initial value.

Some other folks prefer to always use explicit type annotation, and that works fine too – it really is a question of style.