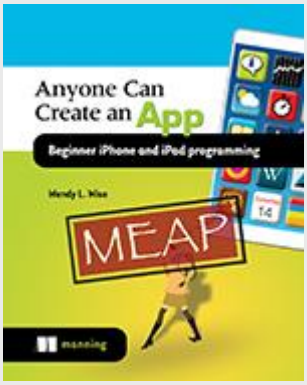

Learning to interact with others: the Swift Playground

By Wendy L. Wise

In this article, excerpted from the book Anyone Can Create an App, I'll introduce you to the Swift Playground, an Apple tool that allows developers to write code and see the results immediately – without having to run the app. We'll also cover Frameworks and variable types.

The Swift Playground is a very helpful interactive tool that provides quick feedback on the code you've written. Sometimes when you create apps, you run them several times before you are finished writing them to make sure they still work. That's because Xcode needs to compile them into a format that the iPhone can work with, and it checks for errors in your code during that process.

Playground provides you with much faster feedback because it doesn't require you to explicitly run the code and it takes no time at all to compile and give you almost immediate feedback; so it is a great place to *play* around (see what Apple did there?) and learn! Let's get started with Playground!

Open Xcode and this time instead of selecting "Create a New Xcode Project," select "Get Started With a New Playground" – the first option, as shown in the figure below.

Figure 1 Start by creating a new playground

In the next screen that opens, leave the default as "MyPlayground," and leave the platform as "iOS". Now save it; I saved mine in my dev folder. Once you save the file, the new playground will open like the one shown below.

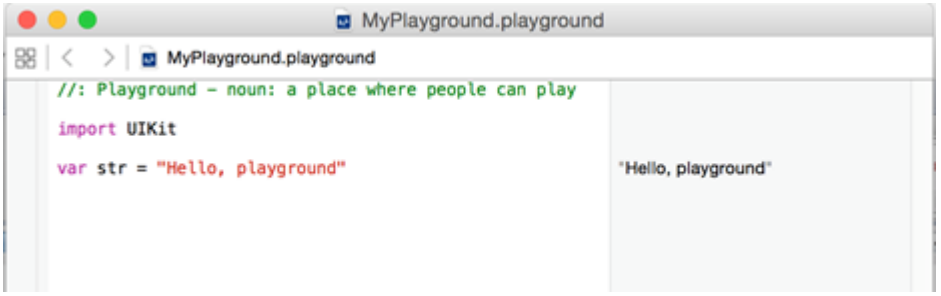

Figure 2 The new playground opens once you click save

- The top line is obviously a comment, because it starts with // on the line, and it is green which denotes a comment.
- The next line is "import UIKit." A UIKit is a framework – but what is a framework you ask…

Frameworks

Imagine trying to build a house and think of all the tools that might be required for that. You may need lots of different kinds of carpentry tools, plumbing tools, electrical tools, roofing

tools, tiling tools, painting tools, floorings tools, etc. Imagine calling up a supply store and telling them to please deliver all the tools you will need to build a house. Not only would you have an enormous stack of tools in front of your empty lot, but you probably would spend a lot of time actually looking for the tool you need when you actually need it. This simply isn't efficient and it surely isn't optimized for building a house. The same concepts apply to writing an app.

There are so many features and functionality included in one iPhone or iPad that it would be very inefficient if Apple tried to make them all available to the programmer up front. Each app that you wrote would essentially have a huge stack of tools sitting behind it and you may never even need those tools. Apple's enormous stack of tools might include photo tools, map tools, health kit tools, address book tools, sound tools, video tools, game tools, etc.

Now imagine if Xcode had to sort through all of these tools every time you wanted to compile and run your program! It would take an enormous of time and it simply wouldn't be efficient or optimized for building an app. So Apple packaged together the needed resources for these different types of tools into one package, which is called a *framework*. This framework is like ordering just the plumbing tools for your house when you need them. You can order the tools (import the framework in Xcode) and have them sitting there ready for you when you need them. Xcode doesn't actually use the tools until you ask for one - like going out in the front yard and getting a wrench from the pile of plumbing tools instead of bringing them all in at once. So a framework is a single grouping of resources that Xcode can easily access in your project. My previous examples of Apple tools, such as photo and map, etc., are actually Apple frameworks! You don't need to remember the list below - they are only shown as an example to give you an idea of what they are.

- Photos framework
- MapKit framework
- HealthKit framework
- AddressBook framework
- AVKit framework
- GameKit framework
- And many, many more!

Anytime you want to add a feature in your app that has to deal with photos for isntance, you will need to import the Photos framework. This brings us back to the "import UIKit" statement at the top of our playground file. I explained in Chapter 3 that "UI" stands for "User Interface" – which is the part of the application the user can actually interact with. So UIKit is essentially exactly what it sounds like: a kit that provides you with the tools necessary to create the user interface (or front end) of an application. When you create an app that allows users to interact with it, you must *import* the UIKit framework into your code.

Now let's move on to the final line in the playgound – var str = "Hello, Playground".

Types of Variables

Let's look at the statement `var str = "Hello, Playground"` and break it down into its different components.

- The letters `var` stand for *variable*, which is the way Swift can store a value and access it again later. It is also "variable" in that you can change it to something else later if you want to.
- The letters `str` stand for the *name* of the variable.
- The `=` denotes that the variable name should be set to something
- The `"Hello, Playground"` is what should be stored in the variable named `str`.

So in pseudocode, I would say "I would like to store the words "Hello, Playground," and I want to call it "str," or to say it more concisely "Set variable "str" equal to "Hello, Playground".

Syntax

When coding in swift, the syntax for variables is:

var (which is a keyword in swift) name_of_variable (this is the name you enter) = some_value (where you decide the value). Or:

```
var name_of_variable = your_value
```

TRY IT IN YOUR PLAYGROUND

Now go back to Xcode and look at your playground again. Start a new line and type "str" and hit enter (Xcode may try to auto-complete the line to String, but just leave it as "str" – add a space after "str" if you need to). Your playground should look like this now:

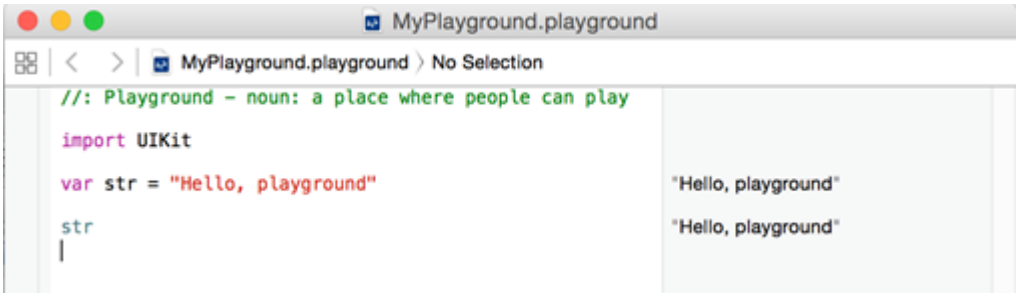

Figure 3 Your playground with another "str"

Notice on the right side of the playground there are now two lines that say "Hello, playground". This is how the playground provides you with "instant gratification" – also known as results without compiling and running your app.

- The line that starts with "var" shows the output on the right side of the screen in the gray area – Xcode evaluated the variable "str" and said it was equal to "Hello, playground".
- The next line did the same thing because the first line told Xcode to store the value "Hello, playground" in the variable named "str".
- The next line basically said "What is the value of str?".

Instant gratification! Now we've learned about variables – or ways to store values in Swift. But what kind of variables are there, you ask? I'm so glad you asked!

Not your shoe strings

The first type of variable we're going to cover is a string. A *string* can be anything from a single letter, a word, or even a sentence that is enclosed with double quotes (""). Xcode (and Playground) know that "Hello, World" is a string type variable because you surrounded the words with double quotes. Variables can be changed, so let's change the variable `str` to "This is a string". Go back to your playgound and add a new line and type `str = "This is a string"`. The right side of the playground should now say "This is a string".

Enter a new line and type `str`. The Playground again shows you that `str` is equal to "This is a string", as shown below.

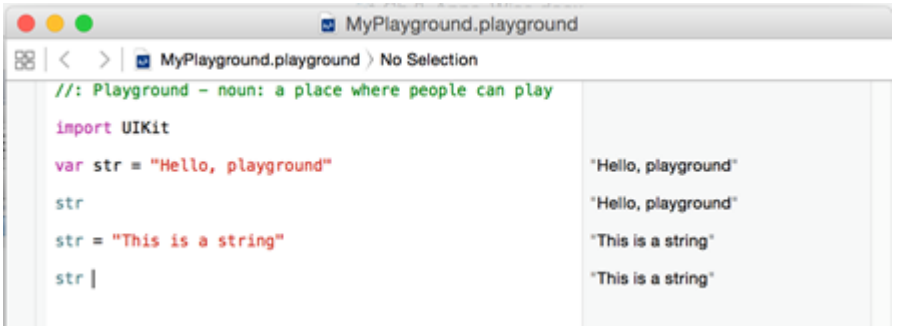

Figure 4 Setting the variable str to a new value

Xcode stored the value `"This is a string"` in place of the previous value `"Hello, playground"`. You may be wondering now how Xcode knows that the variable `str` should be a string. The Swift language *interprets* the variable type by how you use it. Swift and Xcode recognize the variable str as a String because it is surrounded by double quotes.

You can also explicity tell Xcode and Swift that this is a String by changing the statement to:

```
var myStr: String = "This is a string"
```

which says in English:

"Create a new variable name myStr of type String, and set it equal to 'This is a String'."

You don't need to explicity define the variable type often, but you should recognize the syntax when you see it.

> **CODING CONVENTIONS: USE LOWER CASE FOR VARIABLE NAMES**
>
> You may notice that the variable names start with a lowercase letter. This is a common coding convention – or the way most people conventionally name variables.
>
> It is much easier to differentiate a variable that you have created (like str or myStr) from the declaration of a type of variable or class name, which start with upper case (String, or ViewController). You can use uppercase to start the name of variables, but it does make the code harder to read.
>
> I highly recommend sticking with the coding convention of starting your variable names with lowercase letters.

If you enter a new line on your Playground and add the statement: `var myStr: String = "hello"`, Playground will show you the value of the variable `myStr` on the right side as shown below.

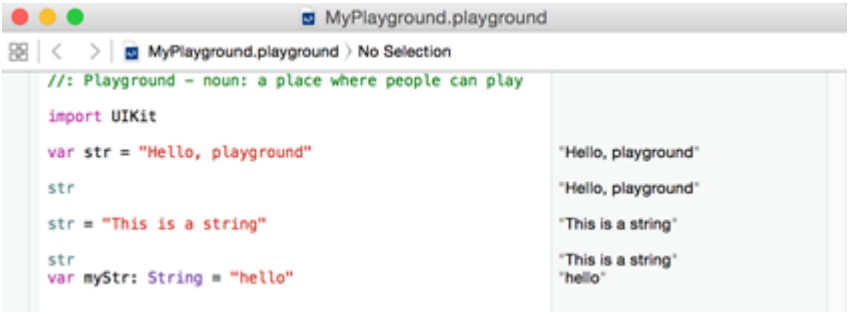

Figure 5 Explictly stating that the variable myStr is of type String

Let's continue looking at other types of variables now.

Going back to math class

Regardless of whether it's been a month or a few decades since you were in a math class, you probably remember there are different types of numbers like: whole numbers, decimals, integers, etc. Guess what? You get to use that knowledge again for creating apps, even though you probably thought at the time – I'm never going to use this!

The first type of number we're going to discuss is an integer. In Swift, we use the variable type `Int` for *integers*, which are whole numbers (numbers without decimal places). An `Int` can store big numbers like -2,147,483,648 or even 2,147,483,648. That's right – an `Int` can be a positive number, a negative number, or even zero. Let's try it in Playground to see an example. Add a new line to your playground and add: `"var myCatsAge = 15"`. Playground shows you that your cat's age is equal to 15 on the right side. Fantastic!

How old will your cat be in five years, I wonder? Let's check. Add a new line to your playground and type: `"var inFiveYears = myCatsAge + 5"`. Playground, shown below, immediately shows you the number 20! Wow! Instant gratification!

Figure 6 Integers can be added together as shown on the last line

You should be able to understand this line intuitively: you just created a new variable `inFiveYears`, and said that it should be equal to however old your cat is now `myCatsAge`, plus five. You can do math with integers, just like you did in math class. That means you can add (+), multiply (*), divide (/), and subtract (-) integers. Write a new line of code to find out how old your cat was 10 years ago…I'll wait here.

Did you do it? You should have added a line similar to this: `var tenYearsAgo = myCatsAge - 10`. Playground should have then instantly given you the answer of 5. You will use the `Int` data type extensively in your programming career. Let's head back to math class now and learn about other number types.

Double, double toil and trouble

The next data type that we're going to discuss is a double. A *double* is a number with decimal points or they are sometimes call fractional numbers (whereas a whole number can't have decimal points). A double can have up to 15 decimal places (15 numbers to the right of the decimal point) like 3.141592653589793 and it can be positive or negative. Doubles are very useful for calculator apps or when trying to find sales tax or a tip percentage.

Let's compute the sales tax for an item in playground now. Try this first and then check to see if you were right. You can multiply two numbers by using the `*` symbol.

- Create a variable for the price of an item, say 12.47
- Create a variable for the tax percent, say 1.07
- Multiply the two together to get the result of 13.3429

How did that work. Did you get it? Check your work by reviewing Figure 7 below.

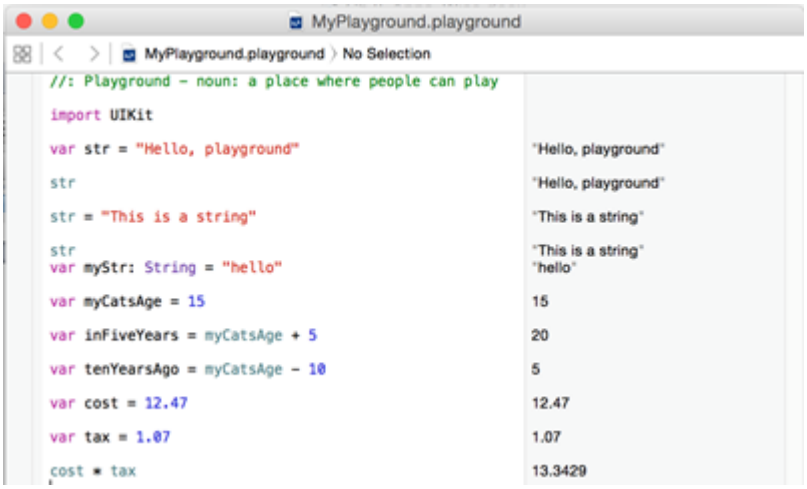

Figure 7 Calculating the total cost after tax

You'll notice in my example that I didn't create a variable to store the final answer in, meaning I can't save the total cost. I could have written it like I did: `cost * tax`, or I could store the value for additional manipulation: `var total = cost * tax`. When coding in Swift for your apps, you will want to use the second method of storing the value so you can continue to use it in your program. For instance, if I want to print out the total or display it to the user in an app, you would want to store the total in a variable so you would just need to print or show the variable instead of the calculation.

There are many more data types that we could cover here, but you won't need to use all of them.

Concepts to remember

1. The Playground is a great tool to help you learn and try new concepts. You can create new playgrounds and save your work if you want.
2. Frameworks are like groups of tools that you will need for different apps.
3. You must import a framework to have access to the tools, like the UIKit for user interfaces.
4. Variables are used to store values that you want to access later.
5. Variable names should always start with a lower case letter and can't have any spaces or special characters.
6. Data types are different kinds of data, as defined by the types of values that they can take like a string (String), a whole number (Int), or a decimal number (Double). There are many types of data types.

Summary

This article covered some of the basic data types that you'll use in your programming career. You can experiment or "play" with the different data types on the Swift Playground so you can get immediate results. Try adding numbers, dividing, subtracting, etc to get used to working with the number formats. We covered a lot in this article – including:

- The Playgound – a great place to experiment and try things without having to wait for an app to run.
- Frameworks – a grouping of tools or classes that you'll need for different kinds of apps. The primary framework we've been using so far is the UIKit framework, which let's you access the user interface features.
- Variables are ways to store information that you want to access later.
- Variables have data types – or kinds of data they can store.