Objective-C Data Types

Objective-C inherits all of the primitive types of C, and defines a few of its own, too. But, applications also require higher-level tools like strings, dictionaries, and dates. The Foundation Framework defines several classes that provide the standard, object-oriented data structures found in typical high-level programming languages.

The Primitives module introduces Objective-C's native types, and the rest of the above modules cover the fundamental Foundation classes. Below, you'll find general information about interacting with these classes.

Creating Objects

There are two ways to instantiate objects in Objective-C. First, you have the standard `alloc/init` pattern introduced in the Classes module. For example, you can create a new `NSNumber` object with the following:

```
NSNumber *twentySeven = [[NSNumber alloc] initWithInt:27];
```

But, most of the Foundation Framework classes also provide corresponding factory methods, like so:

```
NSNumber *twentySeven = [NSNumber numberWithInt:27];
```

This allocates a new `NSNumber` object for you, autoreleases it, and returns it. Before ARC, this was a convenient way to create objects, but in modern programs, there is no practical difference between these two instantiation patterns.

In addition, numbers, strings, arrays, and dictionaries can be created as literals (e.g., `@"Bill"`). You'll see all three notations throughout the Objective-C literature, as well as in this tutorial.

Comparing Objects

Comparing objects is one of the biggest pitfalls for Objective-C beginners. There are two distinct types of equality comparisons in Objective-C:

- **Pointer comparison** uses the == operator to see if two pointers refer to the same memory address (i.e., the same object). It's not possible for different objects to compare equal with this kind of comparison.
- **Value comparison** uses methods like `isEqualToNumber:` to see if two objects represent the same value. It *is* possible for different objects to compare equal with this kind of comparison.

Generally, you'll want to use the second option for more robust comparisons. The relevant methods are introduced alongside each data type (e.g., `NSNumber`'s Comparing Numbers section).

Continue to *Primitives* ›