Day 4

Swift is able to figure out what type of data a constant or variable holds based on what we assign to it. However, sometimes we don't want to assign a value immediately, or sometimes we want to override Swift's choice of type, and that's where type annotations come in.

So far we've been making constants and variables like this:

```
let surname = "Lasso"
var score = 0
```

This uses *type inference*: Swift *infers* that `surname` is a string because we're assigning text to it, and then infers that `score` is an integer because we're assigning a whole number to it.

Type annotations let us be explicit about what data types we want, and look like this:

```
let surname: String = "Lasso"
var score: Int = 0
```

`String` holds text:

```
let playerName: String = "Roy"
```

`Int` holds whole numbers:

```
var luckyNumber: Int = 13
```

`Double` holds decimal numbers:

```
let pi: Double = 3.141
```

`Bool` holds either true or false:

```
var isAuthenticated: Bool = true
```

`Array` holds lots of different values, all in the order you add them. This must be specialized, such as `[String]`:

```
var albums: [String] = ["Red", "Fearless"]
```

Dictionary holds lots of different values, where you get to decide how data should be accessed. This must be specialized, such as **[String: Int]**:

```
var user: [String: String] = ["id": "@twostraws"]
```

Set holds lots of different values, but stores them in an order that's optimized for checking what it contains. This must be specialized, such as **Set<String>**:

```
var books: Set<String> = Set(["The Bluest Eye", "Foundation", "Girl, Woman
```

Knowing all these types is important for times when you don't want to provide initial values. For example, this creates an array of strings:

```
var soda: [String] = ["Coke", "Pepsi", "Irn-Bru"]
```

Type annotation isn't needed there, because Swift can see you're assigning an array of strings. However, if you wanted to create an *empty* array of strings, you'd need to know the type:

```
var teams: [String] = [String]()
```

Now, there's a very good chance you'll be asking *when* you should use type annotations, so it might be helpful for you to know that I prefer to use type inference as much as possible, meaning that I assign a value to a constant or variable and Swift chooses the correct type automatically. Sometimes this means using something like **`var score = 0.0`** so that I get a **Double**.

The most common exception to this is with constants I don't have a value for yet. You see, Swift is really clever: you can create a constant that doesn't have a value just yet, later on *provide* that value, and Swift will ensure we don't accidentally use it until a value is present. It will also ensure that you only ever set the value once, so that it remains constant.

For example:

```
let username: String
// lots of complex logic
username = "@twostraws"
// lots more complex logic
print(username)
```

Updated for Xcode 16

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:

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.