Type Safety and Type Inference

Swift is a *type-safe* language. A type safe language encourages you to be clear about the types of values your code can work with. If part of your code requires a `String`, you can't pass it an `Int` by mistake.

Because Swift is type safe, it performs *type checks* when compiling your code and flags any mismatched types as errors. This enables you to catch and fix errors as early as possible in the development process.

Type-checking helps you avoid errors when you're working with different types of values. However, this doesn't mean that you have to specify the type of every constant and variable that you declare. If you don't specify the type of value you need, Swift uses *type inference* to work out the appropriate type. Type inference enables a compiler to deduce the type of a particular expression automatically when it compiles your code, simply by examining the values you provide.

Because of type inference, Swift requires far fewer type declarations than languages such as C or Objective-C. Constants and variables are still explicitly typed, but much of the work of specifying their type is done for you.

Type inference is particularly useful when you declare a constant or variable with an initial value. This is often done by assigning a *literal value* (or *literal*) to the constant or variable at the point that you declare it. (A literal value is a value that appears directly in your source code, such as `42` and `3.14159` in the examples below.)

For example, if you assign a literal value of `42` to a new constant without saying what type it is, Swift infers that you want the constant to be an `Int`, because you have initialized it with a number that looks like an integer:

```
let meaningOfLife = 42
// meaningOfLife is inferred to be of type Int
```

Likewise, if you don't specify a type for a floating-point literal, Swift infers that you want to create a `Double`:

```
let pi = 3.14159
// pi is inferred to be of type Double
```

Swift always chooses `Double` (rather than `Float`) when inferring the type of floating-point numbers.

If you combine integer and floating-point literals in an expression, a type of `Double` will be inferred from the context:

```
let anotherPi = 3 + 0.14159
// anotherPi is also inferred to be of type Double
```

The literal value of `3` has no explicit type in and of itself, and so an appropriate output type of `Double` is inferred from the presence of a floating-point literal as part of the addition.