1. Haskell is known for its powerful type system.
   1. Every expressions type is known at compile time. This makes code safer.
   2. Everything in Haskell has a type. This allows the compiler to reason about the program extensively.
   3. Unlike Java, Haskell has **type inference**.
2. Explicit type declaration
   1. Types are denoted with **::**
   2. The **::** operators means “has type of”
   3. Explicit type are always denoted with the first letter uppercase.
   4. Functions also have types – after all they are first class citizens.
      1. When writing functions, we can choose to give them an explicit type declaration. This is generally considered good practice.
   5. For example, **f :: [Char] -> [Char]** means that function **f** takes one String as a parameter and returns another String.
   6. For functions that take more than one parameter we write something like: **addThree :: Int -> Int -> Int -> Int**. This means that the function takes three Int parameters and also returns an Int.
      1. The return type is always last in the declaration.
      2. We separate each parameter with an arrow due to implicit currying.
3. **Type Variables**
   1. Some functions can operate on *various* types
      1. For example, the **head** function on a list returns the head element of any list, regardless of its type
   2. The type of head is  **a -> a**
      1. In this example, **a** is *not* a type. It is a **type variable** which means that **a** can be of *any type*
   3. Type variables allow functions to operate on any type in a **type-safe manner**.
   4. These are a lot like **generics** in other languages. But Haskell is much more powerful because it makes it very easy to write very general functions.
   5. Functions that use type variables are called **polymorphic functions**
4. **Type Classes**
   1. A Type class is an interface that defines a behavior.
   2. If a type is an *instance* of a type class, then is supports and implements the designated behavior.
   3. When we make a type an instance of a type class, we define what those functions mean for that particular type.
   4. A new symbol is used with type classes: **=>**
      1. Everything before the =>is a *class constraint*
      2. That just means that the type of anything before the **=>** must be an instance of the type class being declared
      3. For example, to use “==” both arguments need to be instances of the **Eq** class.
   5. Note: type classes are *not* related to the construct of classes in OOP
   6. One type can be an instance of many different type classes. One type class can have many instances.
   7. In some cases, an type must be an instance of one type class in order to be a valid instance of some other type class. For example, in order to be an instance of Ord it must be necessary to first be an instance of Eq.