CHAPTER 7: MORE FUNCTIONAL PATTERNS

A value that can be used as an argument to a function is a *first-class* value.

Some ERROR cases!!!

The first one, not in scope because y was *inside* the `let` expression.

```
bindExp :: Integer -> String
bindExp x = let z = y + x in -- Here y is not in scope!!!
            let y = 5 in "the integer was: "
            ++ show x ++ " and y was: "
            ++ show y ++ " and z was: " ++ show z
```

This time, `x = 10` shadows the `x` argument.

```
bindExp :: Integer -> String
bindExp x = let x = 10; y = 5 in
            "the integer was: " ++ show x
            ++ " and y was: " ++ show y
```

The reason is that Haskell has **lexical/static scoping** (depends on the location in the code and the lexical context - like `let` or `where` clauses).

Anonymous functions

```
Prelude> (\x -> x * 3) 5
15
```

Incomplete pattern matches applied to data they don't handle will return *bottom*, a non-value used to denote that the program cannot return a value or result.

1 Pattern matching against data constructors

Some data constructors have parameters, and pattern matching can let us expose and make use of the data in their arguments.

newtype

newtype is a special case of **data** declaration. It permits only 1 constructor and only one field.

2 Higher-order function

Higher-order function

Higher-order functions are functions that accept functions as arguments. Functions are values.

Guards always evaluate sequentially, so it should be ordered from the case that is most restricive to the one that is least restrictive.

Function composition and "pointfree" style

Currying

Currying is the process of transforming a function that takes multiple arguments into a series of functions which each take 1 argument and return one result.

And don't use error.