Lists

Here is a list of Ints, and a list of Strings.

```
val xs: List<Int> = listOf(1, 2, 3, 4, 5, 6, 7)
val strings: List<String> = listOf("quick", "brown", "fox")
```

Higher Order Functions

We can pass a function as an argument to another function.
There are three ways to do this.

Passing the function using a 'method reference'

```
fun square(x: Int): Int = x * x
xs.map(::square)
```

Using an Anonymous function assigned to a value

```
val square2: (Int) -> Int = fun(x: Int): Int = x * x
xs.map(square2)
```

This is a value with a function type. On the right hand side we omit the function name.

We can call this type of function just like a regular function:

```
square2(7)
```

Using a Lambda function

The lambda syntax uses braces and an arrow.

```
val squareFunc: (Int) -> Int = { x -> x * x }
```

Or, we can pass the lambda function directly to `map`:

```
xs.map({ y -> y + 1 })
```

However, when the HOF takes only a single argument, omit the parentheses.

```
xs.map { y -> y + 1 }
```

We can chain calls together:

```
xs.map { y -> y + 1 }.map { x -> x * x }
```

We can write a function that takes a function as an argument, and call it in the body:

```
fun applyFuncTo(
    x: Int,
    f: (Int) -> Int,
): Int = f(x)
```

Accessing Elements

```
val strs = listOf("quick", "brown", "fox")
val first = strs[0]
val second = strs[1]
```

Adding Elements

Adding elements to a list does not modify the original list.

```
val moreStrings = strs.plus("ran")
```

Read/Write Details

Regular lists are read only. Once they are created, we can't change their contents.

```
val readOnlyList: List<String> = listOf("quick", "brown", "fox")
```

If we want to change the contents of a list, we need a different type: `MutableList`.

```
val listA: MutableList<String> = mutableListOf("quick", "brown",
"fox", "dog")
```

`MutableList` Operations

```
listA.add("lion")
listA.add("tiger")
listA.remove("lion")
```

We can still call `plus()`, but that creates a new list. It doesn't modify the original.

```
val listB = listA.plus("elephant")
```

The `forEach` function

We can use the `forEach` function to process each element.

```
mappedList.forEach { s -> println(s) }
mappedList.forEach { println(it) }
mappedList.forEach(::println)
```

For Loops

```
for (s in listA) {
    println(s)
}
for (i in 0 .. 3) {
```

```
    println(listA[i])
}
```