Unit 4f: Common Types of Function

1 Introduction

The general term function refers to any function that can take arguments (parameters) and return any type. The term predicate is commonly used to refer to a specific type of function that expresses some kind of condition and returns a Boolean. In English, the verb *predicate* means to require something as a condition of something else.

The following terms are sometimes used to describe specific types of function:

- **Function** – can return any type
- **Consumer or Action** – function that doesn't return anything
- **Supplier** – function that takes no parameters and returns a value
- **Predicate** – function that returns Boolean

These terms are used formally as function types in some languages (not Scala, though), e.g. C# has **Func**, **Action** and **Predicate** types; Java 8 has **Function**, **Consumer**, **Supplier**, **Predicate** interfaces.

2 Composition with HO functions

The following call composes our *filter* and *predicate* functions:

```
scala> filter(List(1,2,3,4,5,6), mypredicate)
res0: List[Int] = List(2, 4, 6)
```

Composition gives great flexibility; let's say we want to solve a different filtering problem, to get only the values in the list that are > 3. We can use the same filter function but compose it with a different predicate.

This example defines the predicate using a lambda expression rather than declaring a named function:

```
scala> filter(List(1,2,3,4,5,6), (x:Int) => x > 3)
res1: List[Int] = List(4, 5, 6)
```

Note that Scala also allows a very compact syntax for lambdas with one parameter – another use for the underscore symbol:

```
filter(List(1,2,3,4,5,6), _ > 3)
```

3 Scala collection functional combinators

The filtering example you have seen works, but you don't actually need to write your own filter function for a Scala collections. As you saw briefly in unit 3, collections have many HO functions "built-in" that can be composed with your own functions to provide ways to transform those collections – functional combinators. You saw the examples using the *exists* and *filter* functions.

The predicate defined here (or any other predicate defined as a named function, a function variable or a lambda) can be used to filter a **List** like this:

```
scala> List(1,2,3,4,5,6).filter(mypredicate)
res2: List[Int] = List(2, 4, 6)
```