Objectives

- (Practice) To utilise a number of standard List operations in Scala
- (Theory) To understand the architectural principles of this recursive data structure

Resources

You should refer to the following resources accessible via Blackboard:

- Topic 5 Lecture videos 5A – 5D
- **Topic 5 folder** (zipped) – includes the notes and slides for this topic
- External website for Scala doc and other **Learning Resources** (see the folder on Blackboard). The Scala documentation (scaladoc) for collection.immutable.List is a standard resource and you should become familiar with it. The number of methods available – many of which are inherited from other linear collections – can be quite overwhelming at first. However, try to focus on the methods covered in the notes and exercises for this topic. As you gain confidence with these you will be able to appreciate the others. There is no need to look at the zip/scan/reduce/fold operators at this stage as these will be covered later.

Introduction

The **List** data structure is the *workhorse* data structure in functional programming. Indeed, the name of the first functional language, **LISP** (1958), stood for **Lis**t **P**rocessing. Scala contains several implementations of the List data structure. The first is the one you get by default in your programs. If you simply declare, say, the following without importing a specific List library:

val xs: List[Int] = List(1, 2, 3)

then the compiler uses the List definition *scala.collection.immutable.List*. Notice the package name "immutable" within this pathname. This is the default list collection – one whose values may not change. Immutable lists are the preferred lists in functional programming languages since they combine linear sequencing and the avoidance of shared mutable state. There is a parallel package for each of the major collection classes which includes, e.g., *scala.collection.mutable.List*. These are provided because Scala is a hybrid OO/FP language and, as such, provides full support for programming with mutable data structures. We will restrict ourselves to immutable lists.

Scala (since version 2.13) provides LazyList type. Instances of LazyList are immutable linked lists whose values are evaluated non-strictly. This permits us to define "infinite" lists whose values we will never enumerate completely, but whose values we can consume to any arbitrary amount as the application demands. We will look at lazy lists in a future topic. It is useful to note that Haskell's lists are lazy because non-strict evaluation is the norm in that language. By contrast, Scala is evaluated using a strict semantics except in specific circumstances such as those provided by the LazyList class.

Activities

3.1 Watch the videos

Watch the videos 5A – 5D. The accompanying slides can be found in the topic-5-folder. These videos give you the background to the list as a recursive data structure; their encoding using classes and methods in Scala; and the use of some standard first-order and higher-order methods.

3.2 Install and run the ListDemo programs

Open your Scala IDE and within the **src** folder move to the **demo.list** package (you may need to create this **list** package if this is the first time).

Copy the Scala files **ListDemo1.scala** and **ListDemo2.scala** into the **list** package. The exercises are embedded within these programs. Follow the instructions in the comments.