chaper one : what is fiunction programming

Read without doing you will find yourself lost

List 和 Array的区别：

Lists are quite similar to arrays, but there are two important differences. First, lists are immutable. That is, elements of a list cannot be changed by assignment. Second, lists have a recursive structure (*i.e.*, a *linked list*),1 whereas arrays are flat.

List are homogeneous

The list type in Scala is *covariant*. This means that for each pair of types S and T, if S is a subtype of T, then List[S] is a subtype of List[T]. For instance, List[String] is a subtype of List[Object]. This is natural because every list of strings can also be seen as a list of objects.2

在scala 中 Nothing 类是所有类的子类

Nil represents the empty list.

FP编程的基石:程序是由纯函数所组成

FP proceeds from a startling premise—that we construct programs using only pure functions, or functions that avoid *side effects* like writing to a database or reading from a file

We construct our programs using only *pure functions*

***pure function definition***

This is all it means for an expression to be referentially transparent—in any program, the expression can be replaced by its result without changing the meaning of the program. And we say that a function is *pure* if its body is RT, assuming RT inputs.

学习编程的方法：

A good way to understand something is to change it slightly and see how the change affects the outcome.

**Referential transparency and purity**

An expression e is *referentially transparent* if for all programs p, all occurrences of e in p can be replaced by the result of evaluating e, without affecting the observable behavior of p. A function f is *pure* if the expression f(x) is referentially transparent for all referentially transparent x.1

As we'll learn, *any* function with side effects can be split into a pure function at the "core" and possibly a pair of functions with side effects; one on the input side, and one on the output side. This is what we did when we separated the declaration of the winner from our pure function winner. This transformation can be repeated to push side effects to the "outer layers" of the program. Functional programmers often speak of implementing programs with a pure core and a thin layer on the outside that handles effects. We will return to this principle again and again throughout the book.

we can use first-class functions to factor out duplication

*function composition*, which feeds the output of one function in as the input to another function

REPL (Read Eval Print Loop)

charpter3. functional data structure

It may not change data in place or perform other side effects. *Therefore, functional data structures are immutable*