3.1 引言

::: details INFO
译者：Tr4cck，Mancuoj

来源：3.1 Introduction

对应：Disc 10、Lab 10
:::

第一章和第二章中描述了编程的两个基本要素：函数和数据之间的密切联系。我们知道了如何使用高阶函数将函数作为数据进行操作，如何使用消息传递和对象系统为数据定义行为。我们还研究了组织大型程序的技术，如函数抽象、数据抽象、类继承和泛型函数，这些核心概念为我们编写模块化、可维护和可扩展的程序打下了坚实的基础。

本章重点讨论编程的第三个基本要素：程序本身。Python 程序只是文本的集合，只有通过解释过程，我们才可以基于该文本执行有意义的计算。像 Python 这样的编程语言之所以很实用，是因为我们可以定义一个解释器，一个用于求解和执行 Python 程序的程序。毫不夸张地说，这就是编程中最基本的思想：解释器决定了编程语言中表达式的含义，但它只是另一个程序。

要理解这一点，就必须转变我们作为程序员的固有形象。我们要把自己视为语言的设计者，而不仅仅是别人设计的语言的使用者。

3.1.1 编程语言

各种程序设计语言在其语法结构、功能和应用领域方面有很大的不同。在通用编程语言中，函数定义和函数应用的结构是普遍存在的。但是，也有一些强大的语言不包括对象系统、高阶函数、赋值，甚至不包括控制结构，如 `while` 和 `for` 语句。我们将介绍 Scheme 编程语言，它是一门具有最小功能集的强大语言。本文介绍的 Scheme 子集不允许出现可变值。

在本章中，我们将研究解释器的设计以及它们在执行程序时产生的计算过程。为通用编程语言设计一个解释器可能令人望而生畏，毕竟，解释器是可以根据它们的输入执行任何可能的计算的程序。然而，许多解释器都有一个优雅的结构，即两个互递归函数：第一个函数求解环境中的表达式，第二个函数将函数应用于参数。

这些函数是递归的，因为它们是相互定义的：调用一个函数需要求解其函数体中的表达式，而求解一个表达式可能涉及调用一个或多个函数。