Chapter 5. Array-Based Sequences

Contents:

1. Python序列类型

在本章节，我们主要探讨Python中的几种“序列”，分别为list(列表)，tuple(元组)以及str(字符串)。这几种数据结构有很多共同点：都支持如seq[k]这种形式的索引查询；都使用一个更底层的概念-“数组”，来表示序列。当然，这几种数据结构“类”的抽象化中也有很多不同之处，这些不同之处更多地隐藏在python内部类的实例之中。这几种数据结构类在Python之中被广泛使用，是我们今后构造复杂数据结构的基础，所以我们必须清楚地理解这些类的外部调用方式以及内部工作机制。

外部调用 – Public Behaviors

对于一个类的对外语义的恰当理解是一个合格的程序员必须要做到的。虽然在Python中对于列表、字符串以及元组的基本使用简单直接，但是对于这几个类的使用还是有几个重要的细微之处，我们必须要了解这些，比如对于序列的复制在底层是如何处理的，以及对于序列的切片操作是如何处理的。如果对于这些机制的了解只是浮于表面，那么在实际使用中，很容易造成代码上的错误。因此，在此书中，我们建立了精确而形象的模型来解释这些类。这些图形会更好的帮助我们掌握对于这些类的更高级的使用，比如说使用嵌套列表来构造多维数组。

实现细节 – Implementation Details

对于这些类的内部实现机制关注过多看起来有违面向对象的原则。在2.1.2章节中，我们着重强调了“封装”(encapsulation)这个概念，也就是作为类的使用者不必了解这个类的内部实现细节。如果我们只是要编写正确的代码，那么我们只需要了解一个类的外部接口则足以；但是如果我们要提高程序的效率，我们就必须深入了解这个类的内部细节。

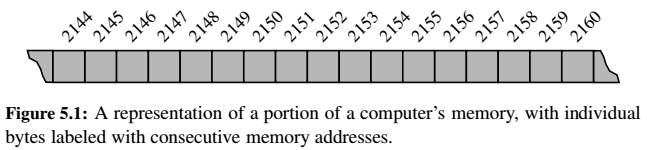
渐进分析和实验分析 – Asymptotic and Experimental Analysis

我们将使用在第3章中提到的渐进分析来描述Python中各种序列数据类型中各种操作的运行效率。同样，我们会进行实验来证明理论上的渐进分析的正确性。

1. 底层数组

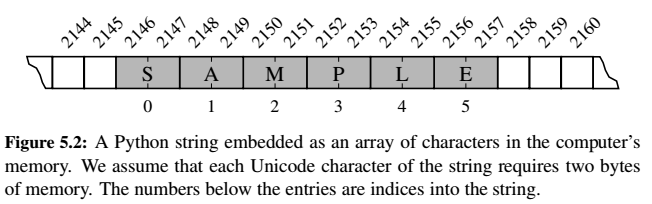
为了准确地描述Python中的序列数据类型，我们首先需要了解底层的计算机体系结构。计算机的内存是由含有信息的比特位组成的，而在不同的计算机体系结构下，比特又被分为更大的存储单元。通常来讲由8个比特组成一个字节（bytes）。

一个计算机系统的内存通常由大量的字节来组成，对于这种情况，计算机用一个“内存地址”的抽象概念来读取内存每个字节中的内容。实际上，每个字节的内存都有一个唯一的地址，比如下图中的“byte 2150”以及“byte 2157”.

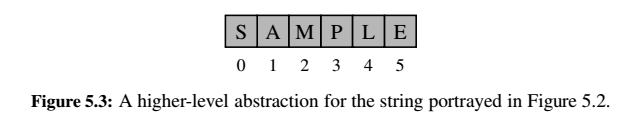


计算机的物理内存是顺序分布的，但计算机系统仍然可以通过RAM（random access memory）这种内存方式在线性时间内（O(1)）读取任意一个字节中的内容。

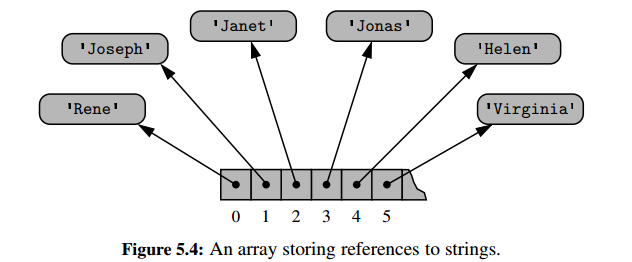
数组定义：在计算机科学中，数组数据结构（array data structure），简称数组（Array），是由相同类型的元素（element）的集合所组成的数据结构，分配一块连续的内存来存储。利用元素的索引（index）可以计算出该元素对应的储存地址。如下图为一个由6个字符组成的数组，在内存中分配了连续的12个字节来存储。数组中的每个单元所分配的内存大小应当相同，只有这样才能用index在线性时间内对任意一个单元进行寻址。



通常情况下，我们只需要关注对于数组结构的高层抽象即可，如下图：



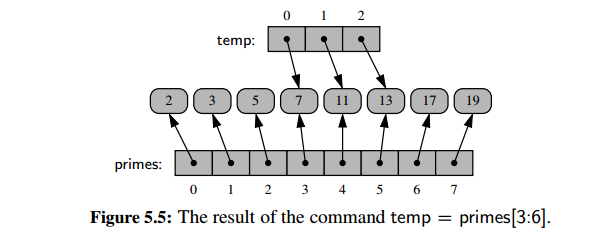
* 1. 引用数组



我们在前面章节中说过，为了满足线性寻址的要求Python必须使数组中每个单元所占的内存大小相同。但在实际中，每个单元中的元素有可能需要不同的内存空间，虽然我们可以为了每个单元都保留出可能的最大的空间，但这样显然对于内存空间是一种浪费。

所以，Python采用了一种“引用”（references）的内部机制来实现list和tuple：底层的连续的一段内存中的数组中存储的是一组地址，通过每个元素的地址可以refer to到每一个元素。

在python中，list和tuple都是引用型数据结构。



* 1. Python中的compact array

1. 动态数组
   1. 实现一个动态数组
   2. 动态数组“折旧”分析
   3. Python中的列表类
2. Python序列数据结构的效率
   1. Python中的列表和元组
   2. Python中的字符串
3. 运用数组序列
4. 多维数据集
5. 练习