FSE598 前沿计算技术

模块 2 数据与数据处理
单元 2 数据结构与类型
第 2 讲 高级数据类型和结构

讲座的英文版内容基于本书：
Y. Chen 《编程语言入门：C、C++、Scheme、Prolog、C# 和 Python 编程》(Introduction to Programming Languages: Programming in C, C++, Scheme, Prolog, C#, and Python)，第 6 版，Kendall Hunt Publishing Company，2019 年。
https://www.public.asu.edu/~ychen10/book/IntroPl.html

本讲大纲

学习容器数据结构

- 列表
- 字典
- 集合
- 元组

Python 容器

- Python 有一些直接内置于语言中的聚合结构
  - 不同于 Java 和其他语言，这些通常是默认语言功能，不需要导入库程序
- 容器：
  - 字符串
  - 列表
  - 字典
  - 元组
  - 集合

Python 列表（对比数组）

- 当你听到人们在讨论 Python 提到数组一词时，他们通常是指列表类型
- 它通常被称为数组，因为它与其他语言中的数组具有相同的“特质”
  - 方括号
  - 带索引
  - 等等
- 然而，列表与我们可能常用的数组并不相同
  - 实际上，根据你使用的解释器，底层结构实际上可能会有所不同
- Python 列表是一个可变长度数组，类似于 Java 中的 ArrayList
  - 但不同于 C 和 C++ 的方式

列表

- 数组是固定大小的内存块，通过相同的名称存储多个变量
  - 每个变量的索引是通过指针算术实现的
  - 数组的每个元素必须具有相同的数据类型
- Java 中的 ArrayList 通过数组类管理
  - 每个 ArrayList 都是一个对象
  - 该对象为我们管理一个数字

列表... 数组 ... 有什么区别?

数组 Array

- 大小固定
- 由程序员管理
- 不能缩小和变大
- 内存一般由程序员管理

List

Python 列表/ Java 中的 ArrayList

- 受系统管理的数组
  - 底层固定数组
- 对象将根据需要替换基础数组
- 初始创建后程序员对大小没有发言权
- 自动变大
- 可能会浪费内存

好的……太好了……那又怎样？

- 为什么知道这一点很重要？
- 就像在 Java 中使用 ArrayLists 一样，在 Python 中使用列表有一个固有的缺陷
- 我们不得不放弃控制，也可能需要牺牲效率
- 我们获得了一种动态感觉的数据结构，它保证了一步操作的运行时间
- 由于我们*通常*不太关心 Python 中的内存效率——所以我们可以接受这种权衡

列表构造

- 要创建列表，我们只需将 [] 赋值给一个变量
- 如果我们愿意，这个 [] 可以包含初始值：
  - myList = []
    - 空列表
  - myList = [1,2,3,4];
    - 包含 1、2、3 和 4 的列表
- 我们也可以构建一份列表
  - myList = list()
    - 空列表
  - myList = list(< iterable >)
    - 将 一个可迭代结构的内容复制到新列表中

列表是对象

- 由于列表是对象，因此它们拥有大量有用的方法和运算符
- [] – index
  - myList[index] 访问特定项目
- [<start>:<finish>]
  - 获取列表的子列表
- [<start>:]
  - 从当前列表获取从start开始到结尾的子列表
- +
  - 将两个列表连接在一起

列表是对象

- append(x)
  - 将项目添加到列表末尾
- exented(<iterable>)
  - 将一个可迭代结构中的所有项目添加到列表末尾
- remove(x)
  - 从列表中查找并删除 x
  - *变更大小*
- pop(*[index]*)
  - 删除列表的最后一项并返回它
  - 如果提供了索引，则删除该索引并返回
- reverse()
  - 反转列表（不返回）
- sort()
  - 对可以排序的数据进行排序
  - 可传入一个函数作为排序标准使用

列表操作

void extend(.)	将元素列表添加到当前列表或数组的末尾。	myList.extend(["boat", "train"])
int index(.)	返回拥有指定值的第一个元素的索引。	print("1st 'car':", myList.index("car"))
void insert(..)	在指定位置添加元素。	myList.insert(2, "plane")
int len(.)	返回列表或数组中的元素的数量（长度）。	print("list length is:", len(myList))
void pop(.)	删除指定位置的元素。	myList.pop(3) # remove the 4th element
void remove(.)	删除等于指定值的第一个元素。	myList.remove("bike")
void reverse()	反转列表或数组的顺序。	myList.reverse()
void sort(.)	对数字或字符串列表进行排序。无法对混合的数字和字符串进行排序。数组排序不行。	numList = [1, 4.5, 7, 3.1, 77, 1, 5.5, 3] numList.sort() strList.sort()

列表

stack queue

- 使用这些方法，可以轻松地将列表作为堆栈或队列使用。
  - 堆栈: append 和 pop 自然会产生类似于堆栈的行为
  - 队列: 你可以使用 pop(0) 来获取队列行为

```
lst1 = [1, 1, 1, 4, 5, 1, 7]
lst2 = [1, 1, 1, 4, 5, 1, 7]
print('List 1 = ', lst1)
print('List 2 = ', lst2)
for i in range(4):
    if i == 1:
        lst1.pop()
for i in range(4):
    if i == 1:
        lst2.pop(0)
print('New List 1 = ', lst1)
print('New List 2 = ', lst2)
```

```
List 1 =  [1, 1, 1, 4, 5, 1, 7]
List 2 =  [1, 1, 1, 4, 5, 1, 7]
New List 1 =  [1, 1, 1, 4]
New List 2 =  [4, 5, 1, 7]
Press any key to continue . . .
```

列表操作示例

```
myList = [1, 2.5, "car", "bike"]
print("list length  is:", len(myList))
myList.append("boat")
print(myList)
myList.insert(2, "plane")
print(myList)
print("count of word bike in list is:",
myList.count("bike"))
myList.remove("bike")
print(myList)
myList1 = myList.copy()
print("Copied list is:", myList1)
myList.extend(["car", "train"])
print(myList)
print("the index of the first 'car' in list is:", myList.index("car"))
print("list bef. pop:", myList)
myList.pop(3)
print("list aft. pop at 3:", myList)
myList.reverse()
print(myList)
myList.clear()
print("After clear operation, myList =", myList)
numList = [1, 4.5, 7, 2, 3.1, 77, 1, 5.5, 3]
strList = ["car", "bike", "plane", "boat", "car"]
numList.sort()
strList.sort()
print(numList)
print(strList)
```

```
list length  is: 4
[1, 2.5, 'car', 'bike', 'boat']
[1, 2.5, 'plane', 'car', 'bike', 'boat']
count of word bike in list is: 1
[1, 2.5, 'plane', 'car', 'boat']
Copied list is: [1, 2.5, 'plane', 'car', 'boat']
[1, 2.5, 'plane', 'car', 'boat', 'car', 'train']
the index of the first 'car' in list is: 3
list bef. pop: [1, 2.5, 'plane', 'car', 'boat', 'car', 'train']
list aft. pop at 3: [1, 2.5, 'plane', 'boat', 'car', 'train']
['train', 'car', 'boat', 'plane', 2.5, 1]
After clear operation, myList = []
[1, 1, 2, 3, 3.1, 4.5, 5.5, 7, 77]
['bike', 'boat', 'car', 'car', 'plane']
```

字典数据结构

- 字典类似于数组。在数组中，我们使用整数作为其元素的索引。
  - 在字典中，也使用索引来访问元素。但是，它不是数字(position)索引，而是字符串（名称）索引。
- 字典类似于列表或链表。它是动态的。它提供以下动态操作：
  - Create()：创建空字典。
  - Insert(e)：向字典插入元素 e。
  - Delete(e)：如果 e 存在的话，从字典中删除元素 e。
  - Lookup(e)：查看元素 e 是否在字典中。如果是，则返回与元素关联的信息。

字典示例

```
zip = dict() # Create a dictionary structure
zip = {"Tempe": 85281, "Chandler": 85224, "Phoenix": 85003, "Mesa": 85201, "Scottsdale": 85250}
print("zip is", zip)
print("Tempe zipcode is ", zip["Tempe"])
print("Mesa zipcode is ", zip["Mesa"])
print("Mesa zipcode is ", zip.get("Mesa"))
print("mesa zipcode is ", zip.get("mesa")) # will not cause an error
print("tempe zipcode is ", zip.get("tempe")) # will not cause an error
print("Gilbert zipcode is ", zip.get("Gilbert")) # will not cause an error
print("tempe zipcode is ", zip["tempe"]) # will cause an error
```

```
zip is {'Tempe': 85281, 'Chandler': 85224, 'Phoenix': 85003, 'Mesa': 85201, 'Scottsdale': 85250}
Tempe zipcode is  85281
Mesa zipcode is  85201
Mesa zipcode is  85201
mesa zipcode is  None
tempe zipcode is  None
Gilbert zipcode is  None
```

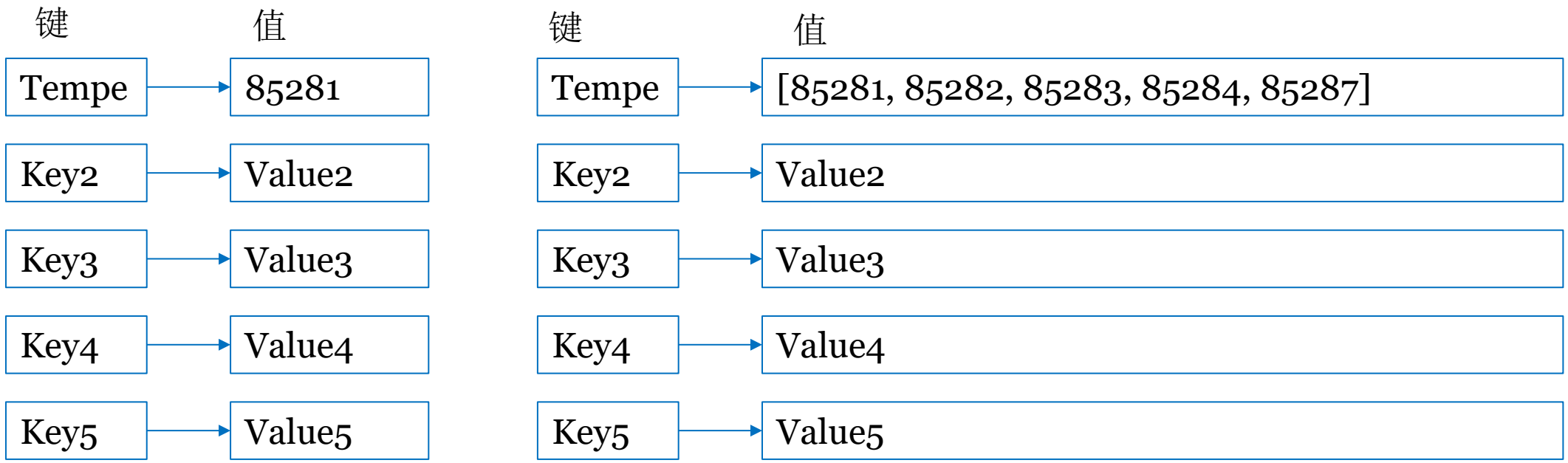
字典示例

```
zipcode = dict() # Create a dictionary structure
zipcode["Tempe"] = [85281, 85282, 85283, 85284, 85287] # insert
zipcode["Chandler"] = [85224, 85225, 85226, 85226] # insert
zipcode["Phoenix"] = [85003, 85004, 85006, 85007, 85008, 85009] # insert
zipcode["Mesa"] = [85201, 85202, 85203, 85204, 85205, 85206, 85207] # insert
zipcode["Scottsdale"] = [85250, 85251, 85253, 85254, 85255] # insert
cities = ["Tempe", "Mesa", "Gilbert", "Phoenix", "Scottsdale", "Chandler"]
for city in cities:
   if city in zipcode:
      print("The zip code of", city, "is", zipcode[city]) # lookup
   else:
      print("The city", city, "is not in my list")
city = input("Please enter a city name:")
found = False
for c in cities:
    c1 = c
    city1 = city
    if c1.lower() == city1.lower():
        print("The zip code of", c, "is", zipcode[c]) # lookup
        found = True
if found == False:
    print("The city", city, "is not in my list")
```

```
The zip code of Tempe is [85281, 85282, 85283, 85284, 85287]
The zip code of Mesa is [85201, 85202, 85203, 85204, 85205, 85206, 85207]
The city Gilbert is not in my list
The zip code of Phoenix is [85003, 85004, 85006, 85007, 85008, 85009]
The zip code of Scottsdale is [85250, 85251, 85253, 85254, 85255]
The zip code of Chandler is [85224, 85225, 85226, 85226]
Please enter a city name:Mesa
The zip code of Mesa is [85201, 85202, 85203, 85204, 85205, 85206, 85207]
Press any key to continue . . .
```

字典结构

字典由键-值(key-value)对组成。



集合

- Python 中的**集合**指数学中的集合。
- 集合中元素的顺序并不重要，重复的元素将被忽略。
- Python 集合用大括号引用。通过集合类型还可以在相同的集合变量中包含不同的值类型。

集合操作

pop會移除最前面的元素

```
set1 = {1, 2, 3.5, "a", "bike"}
set2 = {"a", "bike", 1, 3.5, 2}
set3 = {3.5, "a", 1, "bike", 1, 2, "bike"}
if set1 == set2 and set2 == set3:
    print("set1, set2, and set3 are equal")
else:
    print("set1, set2, and set3 are NOT equal")
print(set1)
print(set2)
for x in set3:
    print(x, end = ' ') # print without newline
print("\n") # print a newline
print(set1)
print(set2)
print(set3)
```

```
set1.pop()
set2.pop()
set3.pop()
print(set1)
print(set2)
print(set3)
set1.remove("bike")
print(set1)
set1.add("car")
print(set1)
set1.add(1.4)
print(set1)
set4=set1.intersection(set2)
set5=set1.union(set2)
print("Intersection of set1 and set2:", set4)
print("Union of set1 and set2:", set5)
```

```
set1, set2, and set3 are equal
{1, 2, 3.5, 'bike', 'a'}
{'bike', 2, 3.5, 1, 'a'}
1 bike 3.5 2 a

{1, 2, 3.5, 'bike', 'a'}
{'bike', 2, 3.5, 1, 'a'}
{1, 'bike', 3.5, 2, 'a'}
{2, 3.5, 'bike', 'a'}
{2, 3.5, 1, 'a'}
{'bike', 3.5, 2, 'a'}
{2, 3.5, 'a'}
{2, 3.5, 'a', 'car'}
{2, 3.5, 1.4, 'a', 'car'}
Intersection of set1 and set2: {'a', 2, 3.5}
Union of set1 and set2: {1.4, 2, 3.5, 1, 'a', 'car'}
Press any key to continue . . .
```

元组（Tuples）

- Python 中的**元组**类似于列表和集合：
  - 它们可以有不同类型的元素，并且
  - 它们都是可迭代的，我们可以使用 for 循环和其他迭代操作遍历它们的元素。
- Python 中的**元组**又不同于列表和集合：
  - 元组是不可变的，不能修改，而列表和集合可以修改。
  - 元组允许冗余元素，元素的顺序很重要。
  - 在语法层面上，列表使用方括号，集合使用大括号，元组使用圆括号。

元组

- 元组是不可变的集合
- 一旦构建，就无法改变
  - 不会缩小也不会变大
  - 无法改变其中包含的数据
- 元组语法与列表语法几乎相同
- 不同之处在于元组使用 () 说明语法
  - myTuple = (1, 2, 3, 4, 5)
- 但元组也使用 [] 表示索引
  - print(myTuple[1]) → 2

元组与列表具有的相似的特征

- 元组具有相似的特征
- [] – index
  - myTuple[index] 访问特定项目
- myTuple[<start>:<finish>]
  - 获取元组的子元组
- myTuple[<start>:]
  - 从当前元组获取从start开始到结尾的子元组
- +
  - 将两个元组连接在一起作为第三个元组

元组操作

```
myTup1 = (0, 1, 20, 3, 51, 7, 9, 20) # define a tuple of int
myTup2 = ("b", "c", "b", "p", "t") # define a tuple of char
myTup3 = (1, 2, 3, "bike", "car", "train") # define a tuple of mixed types
print(myTup1[0], "- print the 1st element in integer tuple")
print(myTup1[2:6], "- print the 3rd to 6th elements")
print(myTup1[:-3], "- beginning to the 4th element counting from back")
print(myTup1[5:], "- 6th to end")
print(myTup1[:], "- beginning to end: all elements")
print(myTup2[1], "- 2nd element in float tuple")
print(myTup2[2:5], "- 3rd to 5th elements")
print(myTup2[:-5], "- beginning to the 6th element counting from back")
print(myTup2[5:], "- 6th to end")
print(myTup2[:], "- beginning to end")
print(myTup2[0], "- 3rd element in Unicode character tuple")
print(myTup3[2:4], "- 3rd to 4th")
print(myTup3[:-3], "- beginning to the 4th element counting from back")
print(myTup3[3:], "- 4th to end")
print(myTup3[:], "- beginning to end")
for x in myTup1: print(x, end = ' ')
for x in myTup2: print(x, end = ' ')
for x in myTup3: print(x, end = ' ')
print("\n")
```

```
0 - print the 1st element in integer tuple
(20, 3, 51, 7) - print the 3rd to 6th elements
(0, 1, 20, 3, 51) - beginning to the 4th element counting from back
(7, 9, 20) - 6th to end
(0, 1, 20, 3, 51, 7, 9, 20) - beginning to end: all elements
c - 2nd element in float tuple
('b', 'p', 't') - 3rd to 5th elements
() - beginning to the 6th element counting from back
() - 6th to end
('b', 'c', 'b', 'p', 't') - beginning to end
b - 3rd element in Unicode character tuple
(3, 'bike') - 3rd to 4th
(1, 2, 3) - beginning to the 4th element counting from back
('bike', 'car', 'train') - 4th to end
(1, 2, 3, 'bike', 'car', 'train') - beginning to end
0 1 20 3 51 7 9 20 b c b p t 1 2 3 bike car train
```

为什么我们需要元组，而不仅仅是列表？

- 像列表一样，元组可以包含不同数据类型的元素，但是
  - 元组元素本质上是不可变的。
  - 元组是由逗号分隔的 Python 对象的集合。
  - 由于本质上是静态的，元组比列表快（因为它们不能被修改）。
- 通常，元组用于表示“记录”
  - 例如，SQL 查询结果通常以元组或元组列表的形式返回
- 我们还可以使用元组作为从函数返回多个值的一种方式
  - 函数只返回一个结果，但返回的可以是元组！
  - 特别是你希望将它们作为常量的情况下。