**深 圳 大 学 实 验 报 告**

**课程名称：­ Python程序设计**

**实验项目名称： 实验3：数据结构**

**学院： 计算机与软件学院**

**专业： 软件工程**

**指导教师： 潘浩源**

**报告人： 郑彦薇 学号： 2020151022**

**实验时间： 2022/03/25~2022/4/5**

**实验报告提交时间： 2022/04/05**

**教务部制**

|  |
| --- |
| **一、实验目的：**  **熟悉对python中列表、字典、元组等数据结构的操作** |
| 1. **实验方法步骤** 2. **读题，对每个问题提出解决问题的思路** 3. **对需要编程解决的题目按照得到的思路编写源代码** |
| **三、实验过程及内容：**  **（一）列表的理解：**  **1、解决问题的思路与方法：**  测试以下三种情况，解释输出结果的原因  **（1）s = [0] \* 3**  **print(s)**  **s[0] += 1**  **print(s)**  对于上述代码，测试得到代码运行结果为：  1648196083(1)  原因解释：  s=[0]\*3创建了一个长度为3，元素全为0的列表s，对s进行输出，得到第一行结果；  s[0]+=1对已创建列表中索引为0的元素即“0”进行加一的操作得到1，再对s进行输出，得到第二行结果。  **（2）s = [''] \* 3**  **print(s)**  **s[0] += 'a'**  **print(s)**  对于上述代码，测试得到代码运行结果为：  **1648196391(1)**  原因解释：  s = [''] \* 3创建了一个长度为3，元素全为空的列表s，对s进行输出，得到第一行结果；  s[0] += 'a'将已创建列表中索引为0对应的元素进行加‘a’操作，第一个元素的值被更改为a，对s进行输出，得到第二行结果。  **（3）s = [[]] \* 3**  **print(s)**  **s[0] += [1]**  **print(s)**  对于上述代码，测试得到代码运行结果为：  1648196639(1)  原因解释：  s=[[]]\*3创建了一个长度为3，元素全为“[]”的列表s，对s进行输出，得到第一行结果；  因为s的每个元素都为一个列表，当使用\*运算符时，包含列表的列表重复创建新列表时，实质是对创建已有对象的引用，因此在对s[0]进行s[0]+=[1]的修改时，相应的引用也就是另外两个空列表“[]”也会被修改。故对s进行输出，得到第二行结果。   1. **遇到的问题和收获：**   在测试最后一段代码时，一开始对于出现的结果并不确定出于什么原因。在测试时插入s.append([])  print(s)并更改修改项的索引为2（如下图所示）：  1648197791(1)  得到的输出为：  1648197826(1)  从而确定通过\*运算符得到的前三个“[]”实质上属于同一个对象，后来插入的“[]”为另一个对象，也反过来验证原测试代码的结果的合理性。   1. **元组的理解：**   **1、解决问题的思路与方法：**  **（1）**下面代码是否正确？解释原因  **t = (1, 2, 3)**  **t.append(4)**  **t.remove(0)**  **t[0] = 1**  以上的代码是不正确的。  原因解释：t=(1,2,3)创建的是元组，而append、remove都是列表的功能，在这里并不能正确使用。另外元素0并不在元组中，也不能正常进行删除。  **（2）**下面代码是否正确？解释原因  **t1 = (1, 2, 3, 7, 1, 0, 5)**  **t2 = (1, 2, 5)**  **t1 = t2[:]**  以上代码是不正确的。  原因解释：根据t1,t2元组的元素，首先t1为“（1，2，3，7，1，0，5）”对于元组有：mytuple[:]表示取元组mytuple中的所有元素，故t2[:]的结果为“（1，2，5）”，很明显t1与t2[:]并不相等。  **（3）切片：**测试下面代码，解释输出的结果  **t = (1, 2, 2, 7, 8, -2, 5)**  **print(t[3])**  **print(t[1: 3])**  **print(t[-1:-3])**  **print(t[-1:-3:-1])**  **print(t[-1: : -3])**  **print(t[ : -1: 3])**  **print(t[3 : -1: 3])**  **print(t[3 : -1: -3])**  对于上述代码，测试得到的运行结果为：  1648199137(1)  原因解释：  **①print(t[3])：**根据列表t的内容，该语句输出索引为3的元素，即输出元素7；  **②print(t[1:3])：**对列表进行切片，start=1，end=3，输出列表中索引从1~3-1的元素，即输出片段（2，2）；  **③print(t[-1:-3])：**对列表进行切片，start=-1，end=-3，按start和end的值，目的是输出列表中索引从-1的元素依次取到索引位-3的前一位元素，在当前列表中表现为从右往左进行输出，但默认输出顺序为左到右，方向相反，故最终输出结果为()；  **④print(t[-1:-3:-1])：**在上述语句的基础上添加step=-1，表示所取片段进行从右往左的输出，与默认start到end的顺序相同。根据start=-1，end=-3，依次从索引为-1的元素即5往左取到索引为-3即8的“前一位”-2，故输出（5，-2）；  **⑤print(t[-1::-3])：**表示从索引-1开始，从右往左，每3位取一个值，即从5开始往左，依次取出7，1，最终输出（5，7，1）；  **⑥print(t[:-1:3])：**默认从列表的第一个元素开始，到索引为-1-1=-2即-2结束，每三位取一个值，依次取得1，7，故输出（1，7）；  **⑦print(t[3:-1:3])：**因为start=3，end=-1，step=3表示从索引为3的元素开始依次往后取到索引为-1的前一位即索引为-2结束，每3位取一个值，取出元素7。因为在元组中，若元组只包含一个元素，最后需多写一个逗号，故最终输出（7，）  **⑧print(t[3:-1:-3])：**因为start=3，end=-1，step=-3表示从索引为3的元素开始从右往左，每三位取一个值，直到索引为-1的元素在当前顺序下的前一位结束，根据元组t可以知道，索引为-1的元素为5，索引为3的元素为7，从7到5为左到右，而根据step规定，取值顺序为从右到左，方向相反，故最终输出().   1. **遇到的问题和收获：**   通过对第三段代码的运行测试，根据结果对切片功能有了更深刻的认识。   1. **函数+列表：** 2. **解决问题的思路与方法：**   写出每段代码的输出,解释原因  **(a)def f(n):**  **n=5**  **m = 2**  **f(m)**  **print(m)**  测试得到代码运行结果为：  **1648201314(1)**  原因解释：  传入函数f(n)的m值在函数中被改为5，但这个值并没有返回给主函数，也就是m=5这个结果只保留在f(n)函数中，对m进行输出，得到的结果还是2.  **(b)def f(L):**  **L[0] = 42**  **print(L[0])**  **L = [1,2,3]**  **f(L)**  **print(L[0])**  测试得到代码运行结果为：  **1648201581(1)**  原因解释：  传入函数f(L)的列表L与主函数中的L为同个对象，在函数中，将列表中索引为0的元素赋值为0，即列表L中元素L[0]更改为42，故在函数f(L)中以及在主函数中，对L[0]进行输出，都可以得到结果42.  **(c)def f():**  **n = 5**  **n=4**  **n=f()**  **print(n)**  测试得到代码运行结果为：  **1648201905(1)**  原因解释：  主函数中首先将n赋值为4，又令n=f()，进入函数f()后，得到一个n=5的结果，但这个结果并没有返回给主函数，而是返回一个默认的值None，而函数外的n一开始的值4又已被更改为f()即None，故最终输出结果None。  **(d)def f(L):**  **L2 = L**  **L = [1,2]**  **L[0] = 5**  **print(L)**  **L = [2,3]**  **print(L2)**  测试得到代码运行结果为：此段代码不能正常输出  原因解释：在主函数中并没有定义L2变量，不能正常运行并输出。  **2、遇到的问题和收获：**  通过对代码(c)的测试，认识了在python中，就算无return语句，函数也会执行return的逻辑，且默认返回值为None。   1. **字典的理解：**   **1、解决问题的思路与方法：**  （1）以下哪些字典创建是有效的，哪些是无效的？解释原因。  **d = {[1, 2]:1, [3, 4]:3}**  **d = {(1, 2):1, (3, 4):3}**  **d = {{1, 2}:1, {3, 4}:3}**  **d = {"12":1, "34":3}**  以上字典创建中，d = {[1, 2]:1, [3, 4]:3}、d = {{1, 2}:1, {3, 4}:3}是无效的，d = {(1, 2):1, (3, 4):3}、d = {"12":1, "34":3}是有效的。  原因解释：  创建字典时，可以通过指定每个key:value对来初始化字典，但是对于字典的key，要求key为任意不可变数据。而[1,2]、[3,4]以及{1,2}、{3,4}分别属于列表和集合或字典，都是可变数据类型，不能作为字典中的key使用。相反，(1,2)、(3,4)和"12"、 "34"分别属于元组和字符串，都是不可变数据类型，可作为字典中的key使用。因此d = {[1, 2]:1, [3, 4]:3}、d = {{1, 2}:1, {3, 4}:3}是无效的，d = {(1, 2):1, (3, 4):3}、d = {"12":1, "34":3}是有效的。   1. 基于以下代码，回答问题 2. 根据题意编写得到的代码如下：   1648805237(1)  代码运行结果展示：  1648805349(1)  原因解释：  根据字典的创建规则和访问规则，可知D.items()可以返回字典中的key-value对列表。通过for循环，在每一趟循环里对D.items()得到的键值列表进行访问，且以x进行记录；又根据键值列表可以知道，在x中索引为0对应键值对中的key值，即x[0]=key。对于字典，又可以通过key对字典进行索引获得value的值，即D[x[0]]=D[key]=value，因此通过D[x[0]]便可在每一趟循环中，依次获得字典中的value值，通过循环对所有值进行相加存储于sum中，最后对sum进行输出，即可得到结果65.   1. 根据题意编写得到的代码如下：   1648805292(1)  代码运行结果展示：  1648805320(1)  原因解释：  同理于问题（a）中所提，字典中可以通过items()获得键值对列表，通过for循环对每次获得的键值对列表存储于x中，又根据键值对列表信息可以知道，在x中索引为1对应键值对中的value值，即x[1]=value。在循环中，每次获得字典中的每对键值对，并通过索引1访问键值对中的值，进行相加，将结果存储于sum中，最后对sum进行输出，也可以获得相同的结果为65.  **2、遇到的问题和收获：**  解决问题（1）时，对可变和不可变数据类型的认识更加深刻；解决问题（2）时，通过两个子问题掌握了两种访问字典中value值的方法。   1. **列表元素的增加：** 2. **解决问题的思路与方法：**   根据题目要求，在计时开始和计时结束之间编写列表元素增加代码，统计不同方法下的运行时间差，采取步骤如下：   1. 设置多层循环，重复增加元素过程，实现对短时间的统计 2. 创建列表并赋初始值 3. 分别使用“+”和append()增加列表元素，统计时间，实验中选择重复在列表中增加元素10进行对时间的统计 4. 输出时间统计结果，进行对比   **（1）使用“+”增加元素：**  首先根据题意，编写使用“+”增加元素的代码如下：  1648958602(1)  得到元素为10的列表list1，加到myList中  重复列表增加  元素过程t次  运行结果如下：  1648958652(1)  **（2）使用append()增加元素：**  首先根据题意，编写使用append()增加元素的代码如下：  1648957988(1)  增加元素10  重复列表增加元素过程t次  运行结果如下：  1648957452(1)   1. **遇到的问题和收获：** 2. **实验发现：**通过上述运行结果，可以知道，使用“+”增加列表元素时，重复t=100000次的实际运行时间约为15.35，而使用append()函数增加列表元素时，重复t=10000000次的时间也仅仅约为1.29，可以知道使用append()增加列表元素的效率时远远高于使用“+”增加列表元素的效率。 3. **实验收获：**通过该实验学习python环境中对程序运行时间的统计，同时也认识了不同的增加元素的方法的运行效率差异，可以更好的选择，增加元素的方式。使用“+”增加列表元素时，需要创建新列表，这一过程也会增加内存的消耗，综合效率与内存，在增加列表元素时，应尽可能的选择append()方法。 4. **合并两个排序的列表：** 5. **解决问题的思路与方法：** 6. **不使用sort()或sorted()：**   Merge函数思路和方法：   1. 创建一个新的空列表为传入的两个列表相加，获取传入的两个列表的长度 2. 通过下标索引比较当前较小值，因为传入函数的两个列表已经有序，因此只需按照列表顺序依次对当前索引所指数据进行比较 3. 索引从0开始，进行比较，将较小值存入list，并对该值所在列表的索引+1，进行下一个值的比较，直到已经完成list1或list2最后一个数的比较 4. 比较结束后，list1或list2可能仍有未比较的数，设置循环将这些数依次存入list即可得到新的有序序列   具体编程如下：  1648974970(1)  初始化List为传入的两个列表相加  对传入函数的list1和list2的元素进行比较，将当前的较小值存入list中，并将对应的下标加1  1648975085(1)  将被默认为字符串的输入转换为列表，使程序能够正常运行  运行结果如下：  输入：[2，4，7]，[1，5，6]  1648975248(1)   1. **使用sort()或sorted()：**   思路和方法：   1. 输入两个列表元素，并将元素传入函数中 2. 创建新列表为传入的两个列表相加 3. 调用sort函数进行排序   具体编程如下：  1648974455(1)  将被默认为字符串的输入转换为列表，且元素为整型  调用sort()进行排序  运行结果如下：  输入：[2，4，7]，[1，5，6]  1648975248(1)   1. **遇到的问题与收获：**   以使用sort()函数排序得到的结果举例，在一开始进行输入时，只是简单的将输入转换为列表，得到的结果如下所示：  1648975500(1)  这是因为输入的“2 4 7”中，空格和数字都被默认为字符串，在输出时也是以字符形式进行输出，且空格会被输出。因此在具体编程中应先将输入转换为整型数组，再将其转换为列表进行操作，从而得到正确结果。   1. **子列表：** 2. **解决问题的思路和方法：** 3. 根据传入的列表获得列表的长度 4. 按照长度设置循环，首先判断list2的第一个元素是否在list1中：从list1第一个元素开始，依次往后判断list1中是否存在与list2第一个元素相同的值 5. 若不在，说明list2一定不是list1的子序列，返回False 6. 若在，则按照子序列的特点，从list2的第一位及其在list1中的位置开始，依次往后判断list1和list2的元素是否一一对应，若不对应，返回False；若list2元素已经全部遍历，则返回True。   具体编程如下：  1649004546(1)  对输入的强制转换  若list2的所有元素已经遍历，且不存在上述不成立的情况，则list2为list1的子列表，返回True  对第一个元素之后的元素依次进行判断，若不能够一一对应，说明list2不为list1的子列表，返回False  首先判断list2的第一个元素是否在list1中，若存在，则进入进一步判断的循环  运行结果如下：  List1统一设定为[4，10，2，3，50，100]  输入list2为[3，2，50]：  1649004903(1)  输入list2为[2，3，50]：  1649004942(1)  输入list2为[2，3，40]：  1649004981(1)   1. **遇到的问题与收获：**   实验进行的是子列表的判断，需注意子列表除了满足每一个元素都在主列表中，还要注意顺序以及连续性。   1. **分饼干** 2. **解决问题的思路和方法：** 3. 首先对贪心算法的思路进行简述：贪心算法是从问题的初始状态出发，通过若干次贪心选择得到最优解，即对当前的情况做出最好的选择以得到最优解或较好的解，在此题的应用为：用小的饼干满足小的胃口 4. 根据贪心算法，首先对获得的贪吃指数和饼干尺寸列表进行从小到大的排序 5. 从两个列表的第一个元素开始，按照饼干尺寸的顺序，判断当前饼干尺寸能否满足当前胃口，若能，则得到满足的孩子数量加一，并以下一块饼干对下一个小孩的胃口进行比较和判断；若不能，则以下一块饼干的尺寸判断能否满足当前胃口 6. 循环上述过程直到遍历所有饼干尺寸或遍历所有小孩的贪吃指数，最后返回满足的孩子数量   具体编程如下：  1649064305(1)  运行结果如下：  输入贪吃指数：1，2，3；饼干尺寸：1，1：  1649064363(1)  输入贪吃指数：1，2；饼干尺寸：1，2，3：  1649064341(1)   1. **遇到的问题和收获：**   了解了如何利用贪心算法解决问题。   1. **“几乎对称”列表** 2. **解决问题的思路和方法：** 3. 首先判断输入列表是否已经为对称列表，若不是，才进一步进行判断 4. 根据输入的列表获得列表的长度，根据长度设置双层循环 5. 在循环中，对元素进行两两互换，并判断当前互换所得列表是否为对称列表 6. 若互换后为对称列表，则说明满足几乎对称的条件，返回True 7. 若循环结束仍不存在这样的两个元素交换后使得列表对称，说明输入的列表不是几乎对称列表，返回False   具体编程如下：  1649140205(1)  需注意：temp=list并不能实现复制而是指向同一个列表，因此同样利用切片操作进行复制  利用切片功能实现列表的倒序，从而进行判断  运行结果如下：  输入[1，2，1，2]:  1649069623(1)  说明：交换lst[0]和lst[1]得到[2，1，1，2]为对称列表  输入[1，2，3，4，5，1，4，3，2，5]:  1649069693(1)  说明：交换lst[5]和lst[9]得到[1，2，3，4，5，5，4，3，2，1]为对称列表  输入[1，2，3，3，2，0]：  1649141283(1)  **2、遇到的问题和收获：**  在进行列表复制时，应注意temp=list只是让temp指针指向list所拥有的内容，并不能达到复制列表用于解决问题这一目的。   1. **矩阵** 2. **解决问题的思路和方法：**   对于矩阵的输入：输入矩阵的行数n，设置循环，输入每一行并转换为列表，添加为列表的新元素。   1. **Matrix\_dim(M):** 2. 传入函数的列表的特点：可知列表中的每一个元素都为其所表示的矩阵中的每一行 3. 根据上述特点，统计列表的元素个数即为矩阵的行数，每个元素（元素也为列表）的具体个数为矩阵的列数 4. 统计行数列数，创建另一个列表，将行、列数添加到新列表中，对新列表进行返回即可得到列表所表示矩阵的维度   具体编程如下：  1649151082(1)  将每一行输入转换为列表，作为新元素添加到列表M中  获取行数和列数  运行结果如下：  **1649151305(1)**   1. **Mult\_M\_v(M,v):** 2. 根据要求创建列表v且列表长度为m与矩阵M的列数相同，列表v表示向量，并将向量和矩阵传入函数中 3. 按照矩阵与向量相乘的运算法则，访问矩阵中的行，将每一行中的元素与向量对应每行元素进行相乘相加 4. 根据矩阵和向量相乘的结果的特点，将每一行相乘结果作为新元素添加到新创建的列表lst中，且每一次得到的结果应先转换为列表，表示得到的结果为n\*1矩阵   具体编程如下：  1649151914(1)  每一个结果转换为列表，添加为lst的新元素  访问矩阵每一行每一个元素，与向量相乘相加  运行结果如下：  **1649152036(1)**   1. **Transpose(M):** 2. 设置临时变量temp，根据矩阵转置的特点，从矩阵的第一列开始，将列中的每一个元素存储到列表temp中 3. 将每一列即temp作为新元素添加到lst中 4. 以矩阵M的列数设置循环，每一趟执行上述步骤，直到循环结束，返回lst   具体编程如下：  1649152274(1)  Temp获取矩阵每一列元素  运行结果如下：  **1649152238(1)**   1. **Largest\_col\_sum(M):** 2. 根据矩阵的行数和列数设置循环，对每一列依次访问每一个元素，进行相加并将结果存储到sum中，sum在获取新的一列元素和之前，应先重新赋值为0 3. Sum与max\_sum进行比较，若比max\_sum大，则更新max\_sum为sum 4. 返回最大列元素和max\_sum   具体编程如下：  1649152763(1)  传入函数的列表元素实质为str类型，在进行相加时需先转换为整型  运行结果如下：  1649152629(1)   1. **Switch\_columns(M,i,j):** 2. 首先将传入函数的矩阵M的每一个元素转换为整型 3. 根据i，j，设置循环从第一行到最后一行，访问第i列和第j列的元素，进行交换 4. 循环结束，返回矩阵M   具体编程如下：  1649152997(1)  运行结果如下：  1649153021(1)   1. **Matrix.py:** 2. 将上述a~e所得函数编写在同一个文件中并命名为matrix.py 3. 在另一个文件中进行调用，对同一个矩阵进行以上5种操作，输出相应的结果   具体编程如下：  matrix.py文件中每个函数的编写与上述具体编程所示相同，这里不做重复的展示。  test\_matrix.py编程为：  1649153880(1)  运行结果如下：  **1649153898(1)**  **2、遇到的问题和收获：**  在python中，并不能实现矩阵的直接输入，需要通过列表的特点进行表示和转换。由于输入被默认为字符串，故在输入时要注意去除符号避免将空格或者逗号也被转换为列表元素。   1. **统计关键字**   **1、解决问题的思路和方法：**   1. 首先将nARQ.py文件转换为txt文件，在python中打开，并将其存储在统计关键字程序test3\_11.py所在文件夹中 2. 将txt文本文档中的内容进行处理：首先将所有的符号转换为空格，使用split()函数将单词以空格和换行为界进行划分，得到列表wordTwo 3. 创建字典dictword，其中所有关键字为python关键字，每一个关键字对应的值为关键字在文本中出现的次数，初始化为0 4. 根据words设置循环，从第一个单词开始，若单词出现在字典中，则对应单词出现次数加一，即value值加一 5. 按照关键字顺序对各自在nARQ文本中出现的次数进行输出，并计算总次数   具体编程如下：  1649168863(1)  1649168887(1)  1649168915(1)  1649168935(1)  运行结果如下：  **1649169059(1)**  **1649169090(1)**  **2、遇到的问题和收获：**  学会了如何在python中对某个文本中单词数量的统计，在开始统计前如何对文本进行去标点符号、转换为一维列表等，如何利用字典元素特点统计关键字出现次数。  在进行文本的访问时，将文本与.py文件存放在同一个文件夹中可以直接调用。 |

|  |
| --- |
| **四、实验总结：**   1. **该实验是利用python中列表、元组、字典等数据结构解决实际问题** 2. **在进行应用时，应注意不同数据结构不同的定义方式以及它们具备的不同功能** |
| **指导教师批阅意见：**  **成绩评定：**  **指导教师签字：**  **年 月 日** |
| **备注：** |

**注：1、报告内的项目或内容设置，可根据实际情况加以调整和补充。**

**2、教师批改学生实验报告时间应在学生提交实验报告时间后10日内**。