

Python 程序设计 实验 3: 数据结构

注意事项:

- (1) 实验报告提交**截止日期: 2022.04.07, 23:59pm**, 迟交扣 20%, 缺交 0 分。
- (2) 实验报告内容包括: 解决问题的思路与方法 (如代码的解释)、遇到的问题以及收获 (简单描述即可)、代码运行结果的展示。
- (3) 实验报告提交方法: **blackboard**。
- (4) 提交要求: 实验报告+源代码, 打包上传, 命名: 学号_姓名_实验报告_3。
- (5) **禁止抄袭, 一经发现 0 分处理 (包括抄袭者和提供代码或实验报告者)!**

1. 列表的理解: 测试以下三种情况, 解释输出结果的原因。

```
s = [0] * 3
print(s)
s[0] += 1
print(s)
```

```
s = [""] * 3
print(s)
s[0] += 'a'
print(s)
```

```
s = [[]] * 3
print(s)
s[0] += [1]
print(s)
```

2. 元组的理解: 测试并回答以下问题

(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
```

(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[:]
```

(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])

```

3. 函数+列表：写出每段代码的输出, 解释原因。请独立地对待每段代码（即一个子问题中的代码与另一个子问题中的代码无关）。

(a)

```

def f(n):
    n=5

```

```

m = 2
f(m)
print(m)

```

(c)

```

def f():
    n = 5

```

```

n=4
n=f()
print(n)

```

(b)

```

def f(L):
    L[0] = 42
    print(L[0])

```

```

L = [1,2,3]
f(L)
print(L[0])

```

(d)

```

def f(L):
    L2 = L
    L = [1,2]
    L[0] = 5
    print(L)

```

```

L = [2,3]
print(L2)

```

4. 字典的理解：回答以下问题

（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}

```

（2）基于以下代码，回答问题：

```

D={"what":22, "are":11, "you":14, "doing":5, "next":9, "Saturday?":4}

```

```
sum = 0
for x in D.items():
    sum = 【代码】
```

如果【代码】分别是

(a) `sum = sum + D[x[0]]`

(b) `sum = sum + x[1]`

sum 的结果是什么？

5. 列表元素的增加：参考以下代码，比较列表中使用“+”和 `append()`这两种方法增加元素在运行时间上差异，记录测量的时间和实验发现。

```
import time
start = time.time()
### your codes
print(time.time()-start)
```

6. 合并两个排序的列表：编写函数 `merge(list1, list2)`，将两个排序好的整数列表合并到一个新的排序列表中，返回这个新的列表。

使用两种方法实现 `merge` 函数：

(1) 不使用 `sort()` 或 `sorted()`；

(2) 使用 `sort()` 或 `sorted()`。

例子：

```
merge([2, 4, 7], [1,5,6])    # return [1, 2, 4, 5, 6, 7]
```

7. 子列表：编写函数 `match_pattern(list1, list2)`，仅当 `list2` 是 `list1` 的子列表时返回 `True`。

例子：

```
list1 = [4, 10, 2, 3, 50, 100]
list2 = [3, 2, 50]
list3 = [2, 3, 50]
list4 = [2, 3, 40]
match_pattern(list1, list2) # return False
match_pattern(list1, list3) # return True
match_pattern(list1, list4) # return False
```

8. 分饼干：每个饼干都有一个尺寸，同时每个孩子都有一个贪吃指数，表示满足最小尺寸的饼干。给每个孩子至多分一块饼干，如果饼干尺寸大于孩子的贪吃指数，那么就可以将饼干分给该孩子使他得到满足。。目标是使最多的孩子得到满足，输出能够满足孩子数的最大值。

例如，输入孩子的贪吃指数为[1, 2, 3]，输入饼干的尺寸为[1, 1]，则输出 1。因为 3 个孩子的贪吃指数为 1、2、3，两块饼干的尺寸均为 1，那么只有一个孩子得到满足。（提示：使用贪心算法）。

9. “几乎对称”列表：如果在一个非对称列表中，我们交换任意两个元素之后，列表是对称的，则把这个列表称为“几乎对称”列表。例如，列表 `lst=[1, 2, 1, 2]` 是几乎对称的，因为交换 `lst[2]`和 `lst[3]`之后，得到对称的列表[1, 2, 2, 1]。编写函数 `is_almost_symmetric(lst)`，仅当列表 `lst` 是几乎对称列表时，返回 `True`。

10. 矩阵：在 Python 中，我们可以使用列表的列表 (a list of lists) 来存储矩阵，每个内部列表代表矩阵一行。例如，我们可以用

$$M = \begin{bmatrix} 5 & 6 & 7 \\ 0 & -3 & 5 \end{bmatrix}$$

来存储矩阵

$$\begin{pmatrix} 5 & 6 & 7 \\ 0 & -3 & 5 \end{pmatrix}$$

我们可以使用 `M[1]`来访问矩阵的第二行（即[0, -3, 4]），也可以使用 `M[1][2]`来访问矩阵的第二行的第三项（即 5）。

- (a) 编写函数 `matrix_dim(M)`，该函数输入上面格式的矩阵 `M`，返回矩阵 `M` 的维度。例如, `matrix_dim([[1,2],[3,4],[5,6]])` 返回 `[3, 2]`。
- (b) 编写函数 `mult_M_v(M, v)`，返回 $n \times m$ 矩阵 `M` 和 $m \times 1$ 向量 `v` 的乘积。
- (c) 编写函数 `transpose(M)`，返回矩阵的转置。
- (d) 编写函数 `largest_col_sum(M)`，寻找矩阵 `M` 中元素总和最大的列，如上面矩阵中第三列元素的总和最大，则返回 12（7+5=12）。
- (e) 编写函数 `switch_columns(M, i, j)`，交换矩阵 `M` 的第 `i` 列和第 `j` 列，返回新的矩阵 `M`。
- (f) 把以上函数（a-e）写进模块 `matrix.py`。编写 `test_matrix.py` 调用模块 `matrix` 并测试函数（a-e）。

- 11. 统计关键字：**统计文本 nARQ.py 中 Python 关键字出现的次数。关键字包括
- "and", "as", "assert", "break", "class",
 - "continue", "def", "del", "elif", "else",
 - "except", "False", "finally", "for", "from",
 - "global", "if", "import", "in", "is", "lambda",
 - "None", "nonlocal", "not", "or", "pass", "raise",
 - "return", "True", "try", "while", "with", "yield"

小提示：

可使用以下函数删除文本中的标点符号，即把标点符号变成" "，

```
import string
def removePunctuations(word):
    for ch in word:
        if ch in string.punctuation:
            word = word.replace(ch, " ")
    return word
```

- 12. 堆的理解：**描述课件中函数 heapreplace 弹出堆中最小元素，同时插入新元素时，堆的变化过程（可手写拍照）。