# Python内置数据结构

讲师: Wayne

从业十余载,漫漫求知路

## 元组tuple

- □ 一个有序的元素组成的集合
- □ 使用小括号()表示
- □ 元组是不可变对象

#### 元组的定义 初始化

- □定义
  - □ tuple() -> empty tuple
  - □ tuple(iterable) -> tuple initialized from iterable's items

```
t = tuple() # 工厂方法
t = ()
t = tuple(range(1,7,2)) # iterable
t = (2,4,6,3,4,2)
t = (1,) # 一个元素元组的定义,注意有个逗号
t = (1,)*5
t = (1,2,3) * 6
```

#### 元组元素的访问

- □ 支持索引(下标)
- □ 正索引:从左至右,从0开始,为列表中每一个元素编号
- □ 负索引:从右至左,从-1开始
- □ 正负索引不可以超界,否则引发异常IndexError
- □ 元组通过索引访问
  - □ tuple[index] , index就是索引 , 使用中括号访问

t[1]

t[-2]

t[1] = 5

#### 元组查询

- index(value,[start,[stop]])
  - □ 通过值value , 从指定区间查找列表内的元素是否匹配
  - □ 匹配第一个就立即返回索引
  - □ 匹配不到,抛出异常ValueError
- □ count(value)
  - □ 返回列表中匹配value的次数
- □时间复杂度
  - □ index和count方法都是O(n)
  - □ 随着列表数据规模的增大,而效率下降
- □ len(tuple)
  - □ 返回元素的个数

## 元组其它操作

□ 元组是只读的,所以增、改、删方法都没有

## 命名元组namedtuple

tom = Student('tom', 20)

jerry = Student('jerry', 18)

tom.name

□ 帮助文档中, 查阅namedtuple, 有使用例程 □ namedtuple(typename, field\_names, verbose=False, rename=False) □ 命名元组,返回一个元组的子类,并定义了字段 □ field\_names可以是空白符或逗号分割的字段的字符串,可以是字段的列表 from collections import namedtuple Point = namedtuple('\_Point',['x','y']) # Point为返回的类 p = Point(11, 22)Student = namedtuple('Student', 'name age')

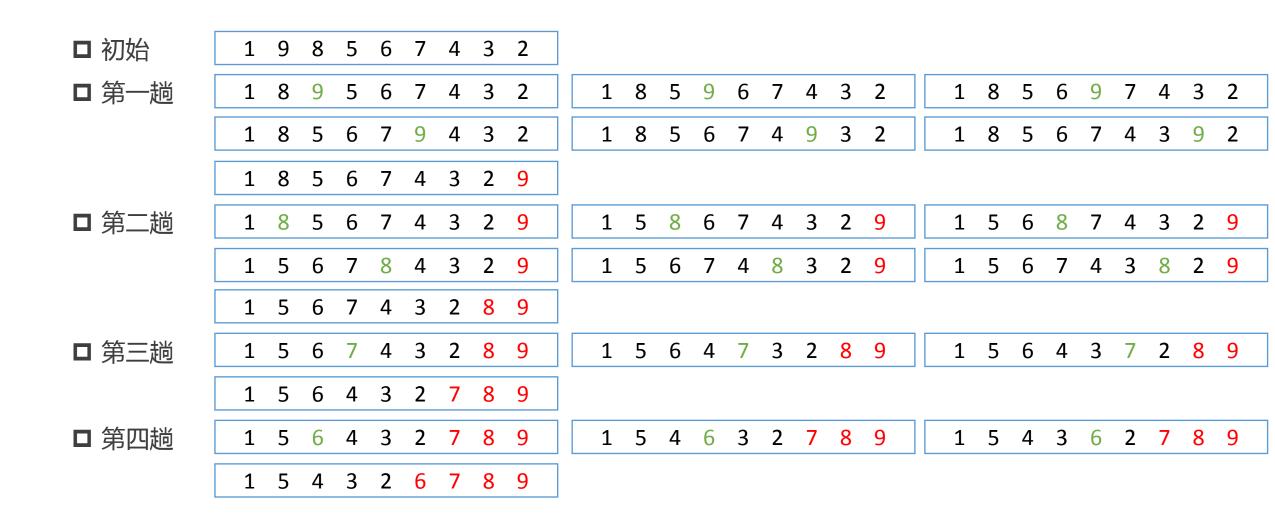
#### 练习

- □ 依次接收用户输入的3个数,排序后打印
  - 1. 转换int后,判断大小排序。使用分支结构完成
  - 2. 使用max函数
  - 3. 使用列表的sort方法
  - 4. 冒泡法

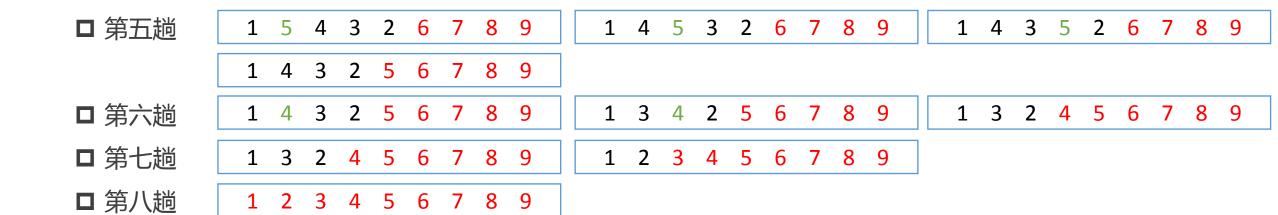
#### 冒泡法

- □冒泡法
  - □ 属于交换排序
  - □ 两两比较大小,交换位置。如同水泡咕嘟咕嘟往上冒
  - □ 结果分为升序和降序排列
- □升序
  - □ n个数从左至右,编号从0开始到n-1,索引0和1的值比较,如果索引0大,则交换两者位置,如果索引1大,则不交换。继续比较索引1和2的值,将大值放在右侧。直至n-2和n-1比较完,第一轮比较完成。第二轮从索引0比较到n-2,因为最右侧n-1位置上已经是最大值了。依次类推,每一轮都会减少最右侧的不参与比较,直至剩下最后2个数比较。
- □ 降序
  - □ 和升序相反

#### 冒泡法



## 冒泡法



#### 冒泡法代码实现(一)

□ 简单冒泡实现

```
num list = \Gamma
   [1, 9, 8, 5, 6, 7, 4, 3, 2],
    [1, 2, 3, 4, 5, 6, 7, 8, 9]
nums = num list[1]
print(nums)
length = len(nums)
count swap = 0
count = 0
# bubble sort
for i in range(length):
    for j in range(length-i-1):
        count += 1
        if nums[j] > nums[j+1]:
            tmp = nums[j]
            nums[j] = nums[j+1]
            nums[j+1] = tmp
            count swap += 1
print(nums, count swap, count)
```

#### 冒泡法代码实现(二)

□ 优化实现 左边有问题 右边正确

```
num list = [
   [1, 9, 8, 5, 6, 7, 4, 3, 2],
    [1, 2, 3, 4, 5, 6, 7, 8, 9]
nums = num list[0]
print (nums)
length = len(nums)
flag = False
count swap = 0
count = 0
# bubble sort
for i in range(length):
    for j in range(length-i-1):
        count += 1
        if nums[j] > nums[j+1]:
            tmp = nums[j]
            nums[j] = nums[j+1]
            nums[j+1] = tmp
            flag = True # swapped
            count swap += 1
    if not flag:
       break
print(nums, count swap, count)
```

```
num list =
    [1, 9, 8, 5, 6, 7, 4, 3, 2],
    [1, 2, 3, 4, 5, 6, 7, 8, 9],
    [1, 2, 3, 4, 5, 6, 7, 9, 8]
nums = num list[2]
print(nums)
length = len(nums)
count swap = 0
count = 0
# bubble sort
for i in range (length):
    flag = False
    for j in range(length-i-1):
        count += 1
        if nums[j] > nums[j+1]:
            tmp = nums[j]
            nums[j] = nums[j+1]
            nums[j+1] = tmp
            flag = True # swapped
            count swap += 1
    if not flag:
        break
print(nums, count swap, count)
```

#### 冒泡法总结

- □ 冒泡法需要数据一轮轮比较
- □ 可以设定一个标记判断此轮是否有数据交换发生,如果没有发生交换,可以结束排序,如果发生交换,继续下一轮排序
- □ 最差的排序情况是,初始顺序与目标顺序完全相反,遍历次数1,...,n-1之和n(n-1)/2
- □ 最好的排序情况是,初始顺序与目标顺序完全相同,遍历次数n-1
- □ 时间复杂度O(n²)

## 谢谢

咨询热线 400-080-6560