作业1

2019年7月29日 19:10

1. 简要说明Python垃圾回收机制

序言

python的GC模块机制实现了自动内存管理，GC做的事情就是解放程序员的双手，找出内存中不用的资源并释放这块内存。主要使用了三种垃圾回收机制：引用计数，标记清楚，分代回收。

引用计数

引用计数法**Reference Counting**的原理是，每个对象都维护一个**引用计数**字段，记录这个对象被引用的次数(如果不清楚变量->引用->对象 的问题，可以查看这篇文章Python的深拷贝和浅拷贝),如果有新的引用指向对象，对象引用计数就加一，引用被销毁时，对象引用计数减一，当用户的引用计数为0时，该内存被释放。可以通过sys.getrefcount()函数查看对象被引用的个数。

这种方法主要存在两种问题：

需要去维护引用计数，存在执行效率问题

无法解决循环引用问题

所谓循环引用就是：有一组对象的引用计数不为0，但是这组对象实际上并没有被变量引用，它们之间是相互引用，而且也不会有其他的变量再去引用这组对象，最终导致如果使用 引用计数法 这些对象占用的内存永远不会被释放。

举个例子：

```
In  [15]: a = []
          b = []
          a.append(b)
          b.append(a)

In  [16]: a
Out[16]: [[[...]]]

In  [17]: b
Out[17]: [[[...]]]
```

可以看到，现在a b都出现了循环引用，此时就算使用del语句删除变量，被使用的内存也不会被回收，所以就需要第二种GC机制：

标记清除

标记清除**Mark-Sweep**是针对**循环引用问题**的回收机制，作用的对象是**容器类型**的对象(比如：list、set、dict等)。

原理是：通过根节点对象(不会被删除的对象)对有向图把所有**活动对象**打上标记，然后回收没有被标记的**非活动对象**。

分代回收

分代回收是建立在标记清除基础上的一种辅助回收容器对象的GC机制。 无论开发的程序类型如何，规模如何，都有这样的相同之处：一些比例的内存生存周期都很短，而另一些内存的生存周期比较长，可能会伴随着整个程序的开始和结束。 所以分代回收就根据系统中内存存活时间把它们划分成不同的集合：一共分成三个集合，每个集合称为一个**代**。 它们的垃圾收集频率 随 对象 存活存活时间的增大 而 减小。也就是说：对于存活时间越长的对象，就越不可能是垃圾，减少对其的收集频率。而新创建的对象都在第一代，第一代集合总数达到上限后，会触发GC机制：可以回收的对象所占的内存被释放，不能被回收的移到中年代。

2. 什么是斐波那契数列、素数、质数和猴子吃桃问题（文字说明即可）？

斐波那契数列

```
i = 1
j = 1
print (i)
print (j)
while True:
    x = i + j
    i = j
    j = x
    if x <= 100:
        print (x)
    else:
        break
```

素数

```
print("1")
for i in range(3,100000,2):
    for j in range(3,int(i**0.5)+1,2):  //除以该数开平方以内的数即可,奇数不可能整除偶数，所以步长为2
        if i % j ==0:
            break
    else:
        print (i)
```

猴子吃桃

```
patch = 1
for i in range(9):  //因为第九天已经发现就剩一个桃子了，所以只需循环9次
    patch = 2*(patch + 1)
print(patch)
```

3. 请写出列表支持的所有方法及说明（例如: append 向列表末尾添加元素）

列表list

- 一个队列，一个排列整齐的队伍
- 列表内的个体称作元素，由若干元素组成列表
- 元素可以是任意对象(数字、字符串、对象、列表等)
- 列表内元素有顺序，可以使用索引
- 线性的数据结构

- 使用【】表示
- 列表是可变的

列表、链表、queue、stack的差异

列表定义、初始化

- list() //生成一个空列表
- list(iterable) //生成一个列表
- 列表不能一开始就定大小

```
lst = list()
lst = []
lst = [2,6,9'q']
lst = list(range(5))
```

列表索引访问

- 索引，也叫下标
- 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
- 为了理解方便，可以认为列表是从左至右排列的，
- 列表通过索引访问

  list[index] //index就是索引，使用中括号访问

```
In  [4]: l[0],l[-5]
Out[4]: (1, 1)
```

- 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

```
In  [14]: l.index(2)
Out[14]: 1
```

  匹配不到，抛出异常valueError

```
In  [15]: l.index(9)
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-15-1bd4ae027d3a> in <module>
----> 1 l.index(9)

ValueError: 9 is not in list
```

- count(value)

  返回列表中匹配value的次数

```
In  [17]: l.count(2)
Out[17]: 2
```

- 时间复杂度

  index和count方法都是O(n) //编历所有元素

  随着列表数据规模的增大，而效率下降
- 如何返回列表元素的个数?

  len()

如何查帮助

- 官方帮助文档

  搜索关键字
- IPython中

  help(keyword)

  keyword可以是变量，对象，类型，函数名，方法名

列表元素修改

- 索引访问修改

  list[index] = value

  索引不要超界

```
In  [25]: l[1] = 9

In  [26]: l
Out[26]: [1, (9, [...]), 3, 6, 8, 2, '-a']
```

列表增加、插入元素

- append(object) // 返回值为none

  列表尾部追加元素，返回None

```
In  [27]: l.append('k')

In  [28]: l
Out[28]: [1, (9, [...]), 3, 6, 8, 2, '-a', 'k']
```

  返回None就意味着没有新的列表产生，就地修改

  时间复杂度是O(1)
- extend(iteratable) //None

  将可迭代对象的元素追加进来，返回None

  就地修改

```
28]: l
28]: [1, (9, [...]), 3, 6, 8, 2, '-a', 'k']

30]: l.extend([1,2])

31]: l
31]: [1, (9, [...]), 3, 6, 8, 2, '-a', 'k', 1, 2]
```

- append与extend的区别

  append只会讲括号里的内容原封不动地追加到列表里，而extend会将其转换成可迭代对象的元素追

加进列表

```
[32]: l.append(range(0,5))

[33]: l
t[33]: [1, (9, [...]), 3, 6, 8, 2, '-a', 'k', 1, 2, range(0, 5)]

[34]: l.extend(range(0,5))

[35]: l
t[35]: [1, (9, [...]), 3, 6, 8, 2, '-a', 'k', 1, 2, range(0, 5), 0, 1, 2, 3, 4]
```

- \+ //列表连接

  连接操作，将两个列表连接起来

  产生新的列表，原列表不动

  本质上调用是__add__()方法

```
In [35]: l
Out[35]: [1, (9, [...]), 3, 6, 8, 2, '-a', 'k', 1, 2, range(0, 5), 0, 1, 2, 3, 4]

In [36]: i=['sfeng']

In [37]: l + i
Out[37]: [1,
          (9, [1, (...), 3, 6, 8, 2, '-a', 'k', 1, 2, range(0, 5), 0, 1, 2, 3, 4]),
          3,
          6,
          8,
          2,
          '-a',
          'k',
          1,
          2,
          range(0, 5),
          0,
          1,
          2,
          3,
          4,
          'sfeng']
```

- * //重复操作，将列表元素重复n次，返回新的列表

```
[41]: print(x)
      [[1, 2, 3], [1, 2, 3], [1, 2, 3]]

[42]: x[0][1]=20

[43]: print(x)
      [[1, 20, 3], [1, 20, 3], [1, 20, 3]]
```

列表删除元素

- remove(value)

  从左至右查找第一个匹配value的值，移除该元素，返回None

  就地修改

  编历修改

- pop([index])

  不指定索引index，就从列表尾部弹出一个元素

  指定索引index，就从索引处弹出一个元素，索引超界抛出IndexError错误

- clear()

  清除列表所有元素，剩下一个空列表

列表其他操作

- reverse()

  将列元素反转，返回None

  就地修改
- sort(key=None,reverse=False)

  对列表元素进行排序，就地修改，默认升序

  reverse为True，反转，降序

  key一个函数，指定key如何排序

  lst.sort(key=functionname)
- in //判段元素是否在列表内

4. 实现一个简易的计算器，效果如下：

（1）. 运行后提示让用户输入一个数字

（2）. 提示输入操作符（+ - * /）

（3）. 再次提示输入一个数字

（4）. 打印计算结果

（5）. 在不退出程序的前提下，可以允许用户继续输入新一组数据计

```
while True:
    num1 = int(input('please input a num'))
    ops = input('please input a ops in "+ - * /"')
    num2 = int(input('please input a num'))
    if ops == "+":
        print (num1 + num2)
    elif ops == "-":
        print (num1 - num2)
    elif ops == "*":
        print (num1 * num2)
    else:
        print (num1 / num2)
```