目录

[Python中常用的自省函数 1](#_Toc29381)

[例 1](#_Toc12984)

[Python各种排序算法 2](#_Toc16048)

[冒泡排序(Bubble Sort) 2](#_Toc15872)

[选择排序(Selection Sort) 3](#_Toc593)

[闭包 3](#_Toc10955)

[Python 的内存机制 3](#_Toc24850)

[魔法方法 4](#_Toc15424)

[静态方法，类实例方法和类方法 4](#_Toc19782)

[进程、线程、协程 4](#_Toc5938)

[uuid 5](#_Toc21610)

[加密 5](#_Toc22011)

[按字典中的 Value 值进行排序 5](#_Toc18189)

[一行代码展开一个二维列表 5](#_Toc28221)

# Python中常用的自省函数

**在计算机编程中，自省是指：检查某些事物以确定它是什么、它知道什么以及它能做什么。自省向程序员提供了极大的灵活性和控制力。**

dir()：返回与该对象关联的方法和属性的列表。

help()：启动内置的帮助系统，查看很多 Python 自带的帮助文档信息。

id()：返回对象的 “标识值”。

type()：返回对象的类型。

hasattr() 和 getattr()：分别判断对象是否有某个属性及获得某个属性值。

callable()：如果实参是可调用的，返回 True，否则返回 False。

isinstance()：可以确认某个对象是属于某种类型。

例：

使用 getattr() 去获取某个对象属性。一般情况下，我们想访问某个对象的属性，一般用点属性运算符进行操作，比如一个类实例对象 obj，它有一个 name 属性，我们可以这样访问这个属性：obj.name。

现在我们想要构造一个方法对同一类型的对象按照某个属性进行排序，但是具体根据那个属性由用户定义。但是我们不可能给每个属性都设计一个排序方法，这样工作量会非常大

class Person(object):  
 def \_\_init\_\_(self, name, age):  
 self.name = name  
 self.age = age  
 def \_\_repr\_\_(self):  
 # 用于显示Person对象  
 return "Person name:{}, age: {}".format(self.name, self.age)  
def persons\_sort\_by(persons, key):  
 # persons是一个Person对象列表  
 return sorted(persons, key=lambda p: getattr(p, key))  
>>> p1 = Person(name='John', age=20)  
>>> p2 = Person(name='Alex', age=24)  
>>> p3 = Person(name='Young', age=15)  
>>> persons\_sort\_by([p1, p2, p3], key='name')  
[Person name:Alex, age: 24,  
 Person name:John, age: 20,  
 Person name:Young, age: 15]

# Python各种排序算法

冒泡排序(Bubble Sort)，选择排序(Selection Sort)，插入排序(Insertion Sort，

希尔排序(Shell Sort)，归并排序(Merge Sort)，快速排序(Quick Sort)，堆排序(Heap Sort)

## 冒泡排序(Bubble Sort)

基本思想：两两比较相邻记录的关键字，如果反序则交换。

nums=[7,2,5,3,1]  
for i in range(len(nums) - 1): # 遍历 len(nums)-1 次  
 for j in range(len(nums) - i - 1): # 已排好序的部分不用再次遍历  
 if nums[j] > nums[j+1]:  
 nums[j], nums[j+1] = nums[j+1], nums[j]

## 选择排序(Selection Sort)

基本思想：选择排序不受输入数据的影响，即在任何情况下时间复杂度不变。选择排序每次选出最小的元素，因此需要遍历 n-1 次。

nums=[7,2,5,3,1]

for i in range(len(nums)-1):  
 for j in range(i+1,len(nums)):  
 if nums[i]>nums[j]:  
 nums[i],nums[j]=nums[j],nums[i]

# 闭包

在一个内部函数中，对外部作用域的变量进行引用，(并且一般外部函数的返回值为内部函数)，那么内部函数就被认为是闭包。简而言之，闭包指延伸作用域的函数。

闭包有两个要素：

1.嵌套在另一个函数中的函数

2.自由变量的引用

def plus\_one():  
 v = 1  
 def add(new\_value):  
 return v + new\_value  
 return add # 这里返回一个函数对象

如果实想引用外部变量并且要给它赋值？答案是nonlocal。

*nonlocal声明的变量不是局部变量,也不是全局变量,而是外部嵌套函数内的变量。*

def plus\_one(n):  
 def add(new\_value):  
 nonlocal n  
 return print(n + new\_value)  
 return add

Python 的内存机制

内存机制主要包括垃圾收集和内存管理两部分。Python 主要使用基于引用计数的垃圾回收机制，用于回收一些不再被使用的内存，以及内存池，用于管理小块内存的申请和释放。

sys.getrefcount()函数来查看对象的引用计数

# 魔法方法

也称之为特殊方法或双下划线方法

\_\_init\_\_构造器，当一个实例被创建的时候初始化的方法；

\_\_new\_\_实例化对象调用的第一个方法，它只取下 cls 参数，并把其他参数传给\_\_init\_\_；

\_\_call\_\_允许一个类的实例像函数一样被调用；

\_\_getitem\_\_定义获取容器中指定元素的行为，相当于 self[key]；

\_\_getattr\_\_定义当用户试图访问一个不存在属性的时候的行为；

\_\_setattr\_\_定义当一个属性被设置的时候的行为；

\_\_getattribute\_\_定义当一个属性被访问的时候的行为。

# 静态方法，类实例方法和类方法

* ****类方法****  
  是类对象的方法，在定义时需要在上方使用@classmethod进行装饰，形参为cls，表示类对象，类对象和实例对象都可调用。
* ****类实例方法****  
  是类实例化对象的方法，只有实例对象可以调用，形参为self，指代对象本身。
* ****静态方法****  
  是一个任意函数，在其上方使用@staticmethod进行装饰，可以用对象直接调用。

# 进程、线程、协程

* ****进程****：一个运行的程序（代码）就是一个进程，没有运行的代码叫程序，进程是****系统资源分配****的最小单位，进程拥有自己独立的内存空间，所有进程间数据不共享，开销大；
* ****线程****: ****cpu 调度****执行的最小单位，也叫执行路径，不能独立存在，依赖进程存在，一个进程至少有一个线程，叫主线程，而多个线程共享内存（数据共享，共享全局变量)，从而极大地提高了程序的运行效率。线程的调度是****由操作系统负责****；
* ****协程****: 是一种用户态的轻量级线程，协程的调度完全****由用户控制****。协程拥有自己的寄存器上下文和栈。协程调度时，将寄存器上下文和栈保存到其他地方，在切回来的时候，****恢复先前保存的寄存器上下文和栈****，直接操中栈则基本没有内核切换的开销，可以不加锁的访问全局变量，所以上下文的切换非常快。

# **uuid**

1.  uuid.uuid1([node[, clock\_seq]])  : 基于时间戳

使用主机ID, 序列号, 和当前时间来生成UUID, 可保证全球范围的唯一性. 但由于使用该方法生成的UUID 中包含有 主机的网络地址, 因此可能危及隐私. 该函数有两个参数, 如果 node 参数未指定, 系统将会 自动调用 getnode() 函 数 来获取主机的硬件地址. 如果 clock\_seq  参数未指定系统会使用一个随机产 生的14位序列号来代替.

>>> uuid.uuid1()

UUID('a8098c1a-f86e-11da-bd1a-00112444be1e')

2.  uuid.uuid3(namespace, name) : 基于名字的MD5散列值

通过计算命名空间和名字的MD5散列值来生成UUID, 可以保证同一命名空间中不同名字的唯一性和不同 命名空间的 唯一性, 但同一命名空间的同一名字生成的UUID相同.

uuid.uuid3(uuid.NAMESPACE\_DNS, 'python.org')

UUID('6fa459ea-ee8a-3ca4-894e-db77e160355e')

4.  uuid.uuid4() : 基于随机数

通过随机数来生成UUID. 使用的是伪随机数有一定的重复概率.

uuid.uuid4()

UUID('16fd2706-8baf-433b-82eb-8c7fada847da')

5.  uuid.uuid5(namespace, name) : 基于名字的SHA-1散列值

通过计算命名空间和名字的SHA-1散列值来生成UUID, 算法与 uuid.uuid3() 相同.

uuid.uuid5(uuid.NAMESPACE\_DNS, 'python.org')

UUID('886313e1-3b8a-5372-9b90-0c9aee199e5d')

去掉UID中间的'-'：uid = str(uuid.uuid4()) , suid = ''.join(uid.split('-'))

# 加密

加密：hashlib.sha1(value.encode('utf-8')).hexdigest()

# 按字典中的 Value 值进行排序

dict(sorted(d.items(), key=lambda x: x[1], reverse=True)) # 升序

# 一行代码展开一个二维列表

from functools import reduce

reduce(lambda x, y: x + y, [[1,2],[3,4],[5,6]])