一、变量

变量有以下几种：数值、字符串、列表、元组、字典

1.数值和字符串

#数值： a = 100

#字符串： a = "100"

# %s 占位，用%后的变量替换

a = 100

b = "Hello World!"

print("number is:%s and words are \"%s\""%(a,b))

2.列表

#列表：用方括号[]扩起来的

c = [1,2,3,4,5,6,7]

d = ["zhangsan","lisi"]

e = [1,2,"zhangsan"]

#1.列表读取

#1.1 用 列表名[索引号]索引列表中的元素

例如：

c[0]

#1.2 用 列表名[起：止]表示切片，从列表中切出相应的元素，前闭后开

例如：

c[0:2] => [1,2]

c[:] => [1,2,3,4,5,6,7]

#1.3 用 列表名[起：止：步长] 定义带步长的切片，步长的正负代表方向，大小代表步长；起止正负代表方向，大小代表开始或终止位置

例如：

切出[5,4,3,2] print(c[4:0:-1])

切出[5,4,3,2,1]  print(c[4::-1])

切出[6,4,2] print(c[5:0:-2]) print(c[-2::-2])

#2.列表修改

#2.1 列表名[索引号] = 新值

例如： c[6] = 8 print(c)

#3.列表删除

#3.1 del 列表名[索引号]

例如：

del c[6]

#4.列表插入

#4.1 列表名.insert(插入位置索引号,新元素)

例如：

c.insert(6,7)

c.insert(1,100)

3.元组

#元组：用圆括号()括起来，特点是一旦定义不能改变

# f = (1,2,3)

# print(f)

4.字典

#字典：用大括号{}括起来 像这样：{键:值, 键:值, 键:值} 几个键值对

dic = {1:"123", "name":"zhang", "height":180}

# print(dic)

#1. 读取字典数据

#用 字典名[键]索引字典中的值

# print(dic["name"])

#2.修改

#用 字典名[键] = 新值

dic["name"] = "cheng"

# print(dic["name"])

#3.删除

#用 del 字典名[键]

del dic[1]

# print(dic)

#4.插入

#字典名[新建] = 新值

dic[1] = 124

# print(dic)

二、条件语句

#条件语句 有三种形式

#形式1

#if 条件成立 :

#  执行任务

#形式2

#if 条件1成立 :

#  执行任务1

#else:

#  执行任务2

#形式3

#if 条件1成立 :

#  执行任务1

#elif 条件2成立 :

#  执行任务2

#  ...

#elif 条件n成立 :

#  执行任务n

#else:

#  执行任务n+1

三、循环语句

#循环语句 有三种形式，终止循环用 break

#形式1：

#for 变量 in range（开始值，结束值）: （注意：区间前闭后开）

# 执行某些任务

# for i in range(0,5):

#     print("count is:%s"%i)

#形式2:

#for 变量 in 列表名:

# 执行某些任务

#形式3:

#while 条件:

# 执行某些任务

四、函数、模块、包

#函数、模块、包

#1.函数；执行某些操作的一段代码 如Input（）

#1.1 定义函数

#def 函数名 （参数表）：

#  函数体

#1.2 使用函数

#函数名（参数表）

# def hi\_name(yourname):

#     print("Hello %s!" %yourname

# hi\_name("Xiaojian Cheng")

#带返回值

# def add(a,b):

#     return a+b

# c = add(1,2)

# print(c)

#2.模块：函数的集合，先导入，在使用，用模块.函数名调用

#import time

#time.xxx（）

#3.包：包含多个模块

#from PIL import Image 从某个包导入某个模块

五、类、对象、面向对象

#类、对象、面向对象

#pass 用来占位

#类里定义函数时，语法规定第一个参数必须是self

#\_\_init\_\_函数，在新对象实例化时会自动运行，用于给新对象赋初值

#子类中调用父类函数用self

# class Animals:

#     pass #占位

# class Mammals(Animals): #子类继承父类，父类放在括号里

#     def breastfeed(self):

#         print("feeding young")

# class Cats(Mammals):

#     def \_\_init\_\_(self,spots):

#         self.spots = spots

#     def catch\_mouse(self):

#         print("cat catch mouse!")

六、文件操作

#文件操作，总共分三步，这里使用的是pickle包

#1.文件写操作

#1.1 开：文件变量 = open（“文件路径文件名”，“wb”）

#1.2 存：pickle.dump（待写入的变量，文件变量）

#1.3 关：文件变量.close（）

import pickle

game\_data = {

    "position":"N2 E4",

    "pocket":["keys","knife"],

    "money":160

}

# save\_file = open("save.csv","wb")

# pickle.dump(game\_data,save\_file)

# save\_file.close()

#2.文件写操作

#1.1 开：文件变量 = open（“文件路径文件名”，“rb”）

#1.2 取：放内容的变量 = pickle.load（文件变量）

#1.3 关：文件变量.close（）

import pickle

save\_file = open("save.txt","rb")

load\_game\_data = pickle.load(save\_file)

print(load\_game\_data)

save\_file.close()

最后再来总结一下：

    变量：

        数值、字符串：比较简单列表，方括号[]，有增删改查操作元组，圆括号()，一旦定义不能改变，只能读取字典，大括号{}，有增删改查操作条件语句：

        有三种形式循环语句：

        也有三种形式函数、模块、包：

        一个包里可以有多个模块，每个模块可以定义一些函数完成特定功能类、对象、面向对象：

        Python也可以封装对象，new出实例调用功能函数文件操作：

        借助pickle包进行读写操作

考试来袭，看看这些知识点都掌握了吗？

1. a += 1 相当于 a = a + 1。

2. a, b = 2, 5 相当于 a = 2; b = 5。

3. 10/3 结果为 3.33333...，求的是10除以3的商，是一个浮点小数。

4. 10//3 结果为 3，求的是10除以3的商整数部分，又叫地板除。

5. 17%4 结果为 1，表示取余数，17除以4商4，余数为1。

6. 4\*\*2 结果为 16，求4的平方。

7. 3\*\*3 结果为 27，求3的立方。

8. '@'\*8 的结果为 '@@@@@@@@'，表示 把@重复8次得到一个新字符串。

9. int(10.5)，把小数10.5转成整数10。

10. float(10)，把整数10转成小数10.0。

11. int("20")，把字符串"20"转成整数20。

12. str(20)，把整数20转成字符串"20"。

13. 列表 a = [ 1, 2, 3, 4, 5, 6, 7]，a[0] 为1，表示列表第一项；a[-1] 为7，表示列表最后一项；

a[x:y] 为一个新列表，表示列表索引为x的项到索引为y-1的项组成的新列表。比如：a[1:5]表示索引为1的项到索引为5-1=4的项，即2,3,4,5组成的新列表[2,3,4,5]。

上面的x如果省略不写，默认为0；y如果省略不写，默认为-1。如：a[:4]结果为[1,2,3,4]，a[2:]结果为[3,4,5,6,7]。

a[x:y:z]第三个参数z，表示间隔，默认为1。如a[::]结果跟a一样。a[1:5:2]结果为[2,4]。求列表总长度：len(a)。

14. 字符串的索引操作，跟列表操作类似。一个字符串，可以看作一个字符的列表。如"about"可认为是一个列表["a", "b", "o", "u", "t"]。

元组可以看作是不可变的列表，索引切片、求长度等操作与列表类似。

15. 在列表末尾增加一个元素：a.append(5)。

16. 删除列表末尾的元素：a.pop()。

17. 删除列表中索引值为i的元素：a.pop(i)。

18. 删除列表中值为e的元素：a.remove(e)，只会删除e第一次出现位置。

19. 判断元素e是否在列表a中：e in a，如果在就会得到True，不在False。

20. 确定元素e在列表a中的索引值：a.index(e)，如果不存在，会报错，所以使用前需要先用if e in a判断一下。

示例：

a = [1,2,3,4]

a.append(5)

#此时a为[1, 2, 3, 4, 5]

a.pop()

#a.pop()返回值5，此时a为[1, 2, 3, 4]

a.remove(2)

#此时 a为[1, 3, 4]

a.pop(2)

#返回4，此时 a 为[1, 3]

i = a.index(3)

#此时i的值为1

21. 将列表a和列表b合并：a+b，包含重复元素。

22. 要想得到列表a中独一无二的元素用：set(a)，这是一个集合，要再转成列表，需要list(set(a))强制转换。

23. 以下a, b代表集合。

集合的特点是：集合元素不会重复，元素之间的顺序不重要。

而列表的特点是：列表元素可以重复，元素的顺序很重要。a - b 计算a和b的差集，即在a中而不在b中的元素构成的集合。

24. a | b 计算a和b的并集，即在a和b中的所有元素，不重复计。

25. a & b 计算a和b的交集，即既在a中，又在b中的元素集合。

26. a ^ b 相当于 a | b - (a&b)。

示例：​​​​​​​

a = {1,4,5,10}

b = {2,5,13,20}

# a - b 的值为：{1, 10, 4}

# a | b 的值为：{1, 2, 4, 5, 10, 13, 20}

# a & b 的值为：{5}

# a ^ b 的值为：{1, 2, 4, 10, 13, 20}

27. 以下的a代表字典。

字典是key（键）和value（值）一一对应的数据类型，键是不能重复的，并且键必须是不可变对象。比如：a = {'ty': 10, 'cg': 20}。其中的'ty'是一个不可变的字符串。

28. a.keys()会返回字典所有键，但并不是一个列表类型，需要强制在外面用上list(a.keys())转换成列表。

29. a.values()会返回字典所有值，但并不是一个列表类型，需要强制在外面用上list(a.values())转换成列表。

30. 判断字典a中有没有某一个键k，用k in a。

示例：

a = {'ty': 10, 'cg': 20}

#a['ty']的值为：10

#list(a.keys())的值为：['ty', 'cg']

#list(a.values())的值为：[10, 20]

if 'cg' in a:

print("Yes")

for k,v in a.items():

print(k, v)

#输出：ty 10

#cg 20

1. 列表（list）和元组（tuple）有什么区别？

在我每一次应聘Python数据科学家的面试中，这个问题都会被问到。所以对这个问题的答案，我可以说是了如指掌。

列表是可变的。创建后可以对其进行修改。

元组是不可变的。元组一旦创建，就不能对其进行更改。

列表表示的是顺序。它们是有序序列，通常是同一类型的对象。比如说按创建日期排序的所有用户名，如["Seth", "Ema", "Eli"]。

元组表示的是结构。可以用来存储不同数据类型的元素。比如内存中的数据库记录，如(2, "Ema", "2020–04–16")（#id, 名称,创建日期）。

2. 如何进行字符串插值？

在不导入Template类的情况下，有3种方法进行字符串插值。

name = 'Chris'# 1. f stringsprint(f'Hello {name}')# 2. % operatorprint('Hey %s %s' % (name, name))# 3. formatprint( "My name is {}".format((name)))

    1

3. “is”和“==”有什么区别？

在我的Python职业生涯的早期，我认为它们是相同的，因而制造了一些bug。所以请大家听好了，“is”用来检查对象的标识(id)，而“==”用来检查两个对象是否相等。

我们将通过一个例子说明。创建一些列表并将其分配给不同的名字。请注意，下面的b指向与a相同的对象。

a = [1,2,3]b = ac = [1,2,3]

    1

下面来检查是否相等，你会注意到结果显示它们都是相等的。

print(a == b)print(a == c)#=> True#=> True

    1

但是它们具有相同的标识（id）吗？答案是不。

print(a is b)print(a is c)#=> True#=> False

    1

我们可以通过打印他们的对象标识（id）来验证这一点。

print(id(a))print(id(b))print(id(c))#=> 4369567560#=> 4369567560#=> 4369567624

    1

你可以看到：c和a和b具有不同的标识（id）。

4. 什么是装饰器（decorator）？

这是每次面试我都会被问到的另一个问题。它本身就值得写一篇文章。如果你能自己用它编写一个例子，那么说明你已经做好了准备。

装饰器允许通过将现有函数传递给装饰器，从而向现有函数添加一些额外的功能，该装饰器将执行现有函数的功能和添加的额外功能。

我们将编写一个装饰器，该装饰器会在调用另一个函数时记录日志。

编写装饰器函数logging。它接受一个函数func作为参数。它还定义了一个名为log\_function\_called的函数，它先执行打印出一些“函数func被调用”的信息（print(f’{func} called.’)），然后调用函数func。最后返回定义的函数。

def logging(func): def log\_function\_called: print(f'{func} called.') func return log\_function\_called

    1

让我们编写其他两个函数，我们最终会将装饰器添加到其中（但还没有）。

def my\_name: print('chris')def friends\_name: print('naruto')my\_namefriends\_name#=> chris#=> naruto

    1

现在将装饰器添加到上面编写的两个函数之中。

@loggingdef my\_name: print('chris')@loggingdef friends\_name: print('naruto')my\_namefriends\_name#=> <function my\_name at 0x10fca5a60> called.#=> chris#=> <function friends\_name at 0x10fca5f28> called.#=> naruto

    1

现在，你了解了如何仅仅通过在其上面添加@logging（装饰器），就能够轻松地将日志添加到我们编写的任何函数中。

5. 解释Range函数

Range函数可以用来创建一个整数列表，一般用在for循环中。它有3种使用方法。

Range函数可以接受1到3个参数，参数必须是整数。

请注意：我已经将range的每种用法包装在一个递推式构造列表（list comprehension）中，以便我们可以看到生成的值。

用法1 - range(stop)：生成从0到参数“stop”之间的整数。

[i for i in range(10)]#=> [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

    1

用法2 - range(start, stop) : 生成从参数“start”到“stop”之间的整数

[i for i in range(2,10)]#=> [2, 3, 4, 5, 6, 7, 8, 9]

    1

用法3 - range(start, stop, step)：以参数“step”为步长，生成从“start”到“stop”之间的整数。

[i for i in range(2,10,2)]#=> [2, 4, 6, 8]

    1

6. 定义一个名为car的类，它有两个属性：“color”和“speed”。然后创建一个实例并返回“speed”。

class Car : def \_\_init\_\_(self, color, speed): self.color = color self.speed = speedcar = Car('red','100mph')car.speed#=> '100mph'

    1

7. Python中的实例方法、静态方法和类方法有什么区别？

实例方法：接受self参数，并且与类的特定实例相关。

静态方法：使用装饰器 @staticmethod，与特定实例无关，并且是自包含的（不能修改类或实例的属性）。

类方法：接受cls参数，并且可以修改类本身。

我们将通过一个虚构的CoffeeShop类来说明它们之间的区别。

class CoffeeShop: specialty = 'espresso'

 def \_\_init\_\_(self, coffee\_price): self.coffee\_price = coffee\_price

 # instance method def make\_coffee(self): print(f'Making {self.specialty} for ${self.coffee\_price}')

 # static method  @staticmethod def check\_weather: print('Its sunny') # class method @classmethod def change\_specialty(cls, specialty): cls.specialty = specialty print(f'Specialty changed to {specialty}')

    1

    2

    3

    4

CoffeeShop类有一个属性specialty，默认值设为“espresso”。CoffeeShop类的每个实例初始化时都使用了coffee\_price这个属性。同时，它还有3个方法，一个实例方法，一个静态方法和一个类方法。

让我们将coffee\_price的值设为5，来初始化CoffeeShop的一个实例。然后调用实例方法make\_coffee。

coffee\_shop = CoffeeShop('5')coffee\_shop.make\_coffee#=> Making espresso for $5

    1

现在我们来调用静态方法。静态方法无法修改类或实例状态，因此通常用于工具函数，例如，把2个数字相加。我们这里用它来检查天气。天气晴朗。太好了！

coffee\_shop.check\_weather#=> Its sunny

    1

现在让我们使用类方法修改CoffeeShop的属性specialty，然后调用make\_coffee方法来制作咖啡。

coffee\_shop.change\_specialty('drip coffee')#=> Specialty changed to drip coffeecoffee\_shop.make\_coffee#=> Making drip coffee for $5

    1

注意，make\_coffee过去是用来做意式浓缩咖啡（espresso）的，但现在用来做滴滤咖啡（drip coffee）了！

8. “func”和“ func”有什么区别？

这个问题的目的是想看看你是否理解所有函数也是Python中的对象。

def func: print('Im a function')

func#=> function \_\_main\_\_.func>func #=> Im a function

    1

    2

func是表示函数的对象，它可以被分配给变量或传递给另一个函数。带圆括号的func调用该函数并返回其输出。

9. 解释map函数的工作原理。

Map函数返回一个列表，该列表由对序列中的每个元素应用一个函数时返回的值组成。

def add\_three(x): return x + 3li = [1,2,3][i for i in map(add\_three, li)]#=> [4, 5, 6]

    1

上面，我对列表中的每个元素的值加了3。

10. 解释reduce函数的工作原理。

这个问题很棘手，在你使用过它几次之前，你得努力尝试自己能够理解它。

reduce接受一个函数和一个序列，然后对序列进行迭代。在每次迭代中，当前元素和前一个元素的输出都传递给函数。最后，返回一个值。

from functools import reducedef add\_three(x,y): return x + y li = [1,2,3,5] reduce(add\_three, li)#=> 11

    1

返回11，它是1 + 2 + 3 + 5的总和。

11.解释filter函数的工作原理

Filter函数顾名思义，是用来按顺序过滤元素。

每个元素都被传递给一个函数，如果函数返回True，则在输出序列中返回该元素；如果函数返回False，则将其丢弃。

def add\_three(x): if x % 2 == 0: return True  else: return Falseli = [1,2,3,4,5,6,7,8][i for i in filter(add\_three, li)]#=> [2, 4, 6, 8]

    1

注意上面所有不能被2整除的元素如何被删除的。

12. Python是按引用调用还是按值调用？

如果你在谷歌上搜索这个问题并阅读前几页，你就要准备好进入语义的迷宫了。你最好只是了解它的工作原理。

不可变对象（如字符串、数字和元组等）是按值调用的。请注意下面的例子，当在函数内部修改时，name的值在函数外部不会发生变化。name的值已分配给内存中该函数作用域的新块。

name = 'chr'def add\_chars(s): s += 'is' print(s)add\_chars(name) print(name)#=> chris#=> chr

    1

可变对象（如列表等）是通过引用调用的。注意下面的例子中，函数外部定义的列表在函数内部的修改是如何影响到函数外部的。函数中的参数指向内存中存储li值的原始块。

li = [1,2]def add\_element(seq): seq.append(3) print(seq)add\_element(li) print(li)#=> [1, 2, 3]#=> [1, 2, 3]

    1

13. 如何使用reverse函数反转一个列表？

下面的代码对一个列表调用reverse函数，对其进行修改。该方法没有返回值，但是会对列表的元素进行反向排序。

li = ['a','b','c']print(li)li.reverseprint(li)#=> ['a', 'b', 'c']#=> ['c', 'b', 'a']

    1

14. 字符串乘法是如何工作的？

让我们看看将字符串" cat"乘以3的结果。

'cat' \* 3#=> 'catcatcat'

    1

该字符串将自身连接3次。

15. 列表乘法是如何工作的?

我们来看看将列表[1,2,3]乘以2的结果。

[1,2,3] \* 2#=> [1, 2, 3, 1, 2, 3]

    1

输出的列表包含了重复两次的列表[1,2,3]的内容。

16. 类中的“self”指的是什么？

“self”引用类本身的实例。这就是我们赋予方法访问权限并且能够更新方法所属对象的能力。

下面，将self传递给\_\_init\_\_，使我们能够在初始化时设置实例的颜色。

class Shirt: def \_\_init\_\_(self, color): self.color = color

s = Shirt('yellow')s.color#=> 'yellow'

    1

    2

17. 如何在Python中连接列表？

将2个列表相加，就是将它们连接在一起。但请注意，数组的工作方式不是这样的。

a = [1,2]b = [3,4,5]

a + b#=> [1, 2, 3, 4, 5]

    1

    2

18. 浅拷贝和深拷贝之间有什么区别？

我们将在一个可变对象（列表）的上下文中讨论这个问题，对于不可变的对象，浅拷贝和深拷贝的区别并不重要。

我们将介绍三种情况。

1、引用原始对象。这将新对象li2指向li1所指向的内存中的同一位置。因此，我们对li1所做的任何更改也会在li2中发生。

li1 = [['a'],['b'],['c']]li2 = li1li1.append(['d'])print(li2)#=> [['a'], ['b'], ['c'], ['d']]

    1

2、创建原始对象的浅拷贝副本。我们可以使用list构造函数来实现这一点。浅拷贝创建一个新对象，但是用对原始对象的引用填充它。因此，向原始列表li3中添加新对象不会传播到li4中，但是修改li3中的一个对象将传播到li4中。

li3 = [['a'],['b'],['c']]li4 = list(li3)li3.append([4])print(li4)#=> [['a'], ['b'], ['c']]li3[0][0] = ['X']print(li4)#=> [[['X']], ['b'], ['c']]

    1

3、创建一个深拷贝副本。这是用copy.deepcopy完成的。现在，这两个对象是完全独立的，并且对其中一个对象所做的更改不会对另外一个对象产生影响。

import copyli5 = [['a'],['b'],['c']]li6 = copy.deepcopy(li5)li5.append([4])li5[0][0] = ['X']print(li6)#=> [['a'], ['b'], ['c']]

    1

19. 列表和数组有什么区别？

注意：Python的标准库有一个array（数组）对象，但在这里，我特指常用的Numpy数组。

列表存在于python的标准库中。数组由Numpy定义。

列表可以在每个索引处填充不同类型的数据。数组需要同构元素。

列表上的算术运算可从列表中添加或删除元素。数组上的算术运算按照线性代数方式工作。

列表还使用更少的内存，并显著具有更多的功能。

20. 如何连接两个数组？

记住，数组不是列表。数组来自Numpy和算术函数，例如线性代数。

我们需要使用Numpy的连接函数concatenate来实现。

import numpy as npa = np.array([1,2,3])b = np.array([4,5,6])np.concatenate((a,b))#=> array([1, 2, 3, 4, 5, 6])

    1

21. 你喜欢Python的什么？

Python可读性很强，并且有一种Python方式可以处理几乎所有事情，这意味着它有一种简洁明了的首选方法。

我将Python与Ruby进行对比，Ruby通常有很多种方法来做某事，但是没有指南说哪种方法是首选。

22. 你最喜欢Python的哪个库？

在处理大量数据时，没有什么比Pandas（熊猫）更有帮助了，因为Pandas让操作和可视化数据变得轻而易举。

23. 举出几个可变和不可变对象的例子？

不可变意味着创建后不能修改状态。例如：int、float、bool、string和tuple。

可变意味着可以在创建后修改状态。例如列表（list）、字典（dict）和集合（set）。

24. 如何将一个数字四舍五入到小数点后三位？

使用round(value, decimal\_places)函数。

a = 5.12345round(a,3)#=> 5.123

    1

25. 如何分割一个列表？

分割语法使用3个参数，list[start:stop:step]，其中step是返回元素的间隔。

a = [0,1,2,3,4,5,6,7,8,9]print(a[:2])#=> [0, 1]print(a[8:])#=> [8, 9]print(a[2:8])#=> [2, 3, 4, 5, 6, 7]print(a[2:8:2])#=> [2, 4, 6]

    1

26. 什么是pickling?

Pickling是Python中序列化和反序列化对象的常用方法。

在下面的示例中，我们对一个字典列表进行序列化和反序列化。

import pickleobj = [ {'id':1, 'name':'Stuffy'}, {'id':2, 'name': 'Fluffy'}]with open('file.p', 'wb') as f: pickle.dump(obj, f)with open('file.p', 'rb') as f: loaded\_obj = pickle.load(f)print(loaded\_obj)#=> [{'id': 1, 'name': 'Stuffy'}, {'id': 2, 'name': 'Fluffy'}]

    1

27. 字典和JSON有什么区别？

Dict是Python的一种数据类型，是经过索引但无序的键和值的集合。

JSON只是一个遵循指定格式的字符串，用于传输数据。

28. 你在Python中使用了哪些ORM？

ORM（对象关系映射）将数据模型（通常在应用程序中）映射到数据库表，并简化了数据库事务。

SQLAlchemy通常用于Flask的上下文中，而Django拥有自己的ORM。

29. any和all如何工作？

Any接受一个序列，如果序列中的任何元素为true，则返回true。

All只有当序列中的所有元素都为true时，才返回true。

a = [False, False, False]b = [True, False, False]c = [True, True, True]print( any(a) )print( any(b) )print( any(c) )#=> False#=> True#=> Trueprint( all(a) )print( all(b) )print( all(c) )#=> False#=> False#=> True

    1

30. 字典和列表的查找速度哪个更快？

在列表中查找一个值需要O（n）时间，因为需要遍历整个列表，直到找到值为止。

在字典中查找一个值只需要O（1）时间，因为它是一个哈希表。

如果有很多值，这会造成很大的时间差异，因此通常建议使用字典来提高速度。但字典也有其他限制，比如需要唯一键。

31. 模块（module）和包（package）有什么区别？

模块是可以一起导入的文件（或文件集合）。

import sklearn

    1

包是模块的目录。

from sklearn import cross\_validation

    1

因此，包是模块，但并非所有模块都是包。

32. 如何在Python中递增和递减一个整数？

可以使用“+=”和“-=”对整数进行递增和递减。

value = 5value += 1print(value)#=> 6value -= 1value -= 1print(value)#=> 4

    1

33. 如何返回一个整数的二进制值？

使用bin函数。

bin(5)#=> '0b101'

    1

34. 如何从列表中删除重复的元素？

可以通过将一个列表先转化为集合，然后再转化回列表来完成。

a = [1,1,1,2,3]a = list(set(a))print(a)#=> [1, 2, 3]

    1

35. 如何检查一个值是不是在列表中存在？

使用“in”。

'a' in ['a','b','c']

#=> True'a' in [1,2,3]#=> False

    1

    2

36. append和extend有什么区别?

Append将一个值添加到一个列表中，而extend将另一个列表的值添加到一个列表中。

a = [1,2,3]b = [1,2,3]a.append(6)print(a)#=> [1, 2, 3, 6]b.extend([4,5])print(b)#=> [1, 2, 3, 4, 5]

    1

37. 如何取一个整数的绝对值？

这可以通过abs函数来实现。

abs(2#=> 2

abs(-2)#=> 2

    1

    2

38. 如何将两个列表组合成一个元组列表？

可以使用zip函数将列表组合成一个元组列表。这不仅仅限于使用两个列表。也适合3个或更多列表的情况。

a = ['a','b','c']b = [1,2,3]

[(k,v) for k,v in zip(a,b)]#=> [('a', 1), ('b', 2), ('c', 3)]

    1

    2

39. 如何按字母顺序对字典进行排序？

你不能对字典进行排序，因为字典没有顺序，但是你可以返回一个已排序的元组列表，其中包含字典中的键和值。

d = {'c':3, 'd':4, 'b':2, 'a':1}

sorted(d.items)#=> [('a', 1), ('b', 2), ('c', 3), ('d', 4)]

    1

    2

40. 一个类如何继承Python的另一个类？

在下面的示例中，Audi继承自Car。继承带来了父类的实例方法。

class Car: def drive(self): print('vroom')class Audi(Car): passaudi = Audiaudi.drive

    1

41. 如何删除字符串中的所有空白？

最简单的方法是使用空白拆分字符串，然后将拆分成的字符串重新连接在一起。

s = 'A string with white space'''.join(s.split)#=> 'Astringwithwhitespace'

    1

42. 在迭代序列时，为什么要使用enumerate？

enumerate允许在序列上迭代时跟踪索引。它比定义和递增一个表示索引的整数更具Python感。

li = ['a','b','c','d','e']for idx,val in enumerate(li): print(idx, val)#=> 0 a#=> 1 b#=> 2 c#=> 3 d#=> 4 e

    1

43. pass、continue和break之间有什么区别?

pass意味着什么都不做。我们之所以通常使用它，是因为Python不允许在没有代码的情况下创建类、函数或if语句。

在下面的例子中，如果在i>3中没有代码的话，就会抛出一个错误，因此我们使用pass。

a = [1,2,3,4,5]for i in a: if i > 3: pass print(i)#=> 1#=> 2#=> 3#=> 4#=> 5

Continue会继续到下一个元素并停止当前元素的执行。所以当i<3时，永远不会达到print(i)。

for i in a: if i < 3: continue print(i)#=> 3#=> 4#=> 5

    1

    2

    3

    4

break会中断循环，序列不再重复下去。所以不会被打印3以后的元素。

for i in a: if i == 3: break print(i) #=> 1#=> 2

    1

44. 如何将for循环转换为使用递推式构造列表（list comprehension）？

For循环如下：

a = [1,2,3,4,5]

a2 = for i in a: a2.append(i + 1)print(a2)#=> [2, 3, 4, 5, 6]

    1

    2

用递推式构造列表来修改这个for循环，代码如下：

a

a3 = [i+1 for i in a]

print(a3)#=> [2, 3, 4, 5, 6]

    1

    2

    3

    4

递推式构造列表通常被认为更具Python风格，同时仍易于阅读。

45. 举一个使用三元运算符的例子。

三元运算符是一个单行的if/else语句。

语法看起来像“if 条件 else b”。

x = 5y = 10'greater' if x > 6 else 'less'#=> 'less''greater' if y > 6 else 'less'#=> 'greater'

    1

46. 检查一个字符串是否仅仅包含数字？

可以使用isnumeric方法。

'123abc...'.isalnum#=> False'

123abc'.isalnum#=> True

    1

    2

47. 检查一个字符串是否仅仅包含字母？

你可以使用isalpha。

'123a'.isalpha#=> False'a'.isalpha#=> True

    1

48. 检查字符串是否只包含数字和字母？

你可以使用isalnum。

'123abc...'.isalnum#=> False'123abc'.isalnum#=> True

    1

49. 从字典返回键列表

这可以通过将字典传递给Python的list构造函数list来完成。

d = {'id':7, 'name':'Shiba', 'color':'brown', 'speed':'very slow'}list(d)#=> ['id', 'name', 'color', 'speed']

    1

50. 如何将一个字符串转化为全大写和全小写？

你可以使用upper和lower字符串方法。

small\_word = 'potatocake'big\_word = 'FISHCAKE'small\_word.upper#=> 'POTATOCAKE'big\_word.lower#=> 'fishcake'

    1

51. remove、del和pop有什么区别?

remove 删除第一个匹配的值。

li = ['a','b','c','d']

li.remove('b')li#=> ['a', 'c', 'd']

    1

    2

del按索引删除元素。

li = ['a','b','c','d']

del li[0]li#=> ['b', 'c', 'd']

    1

    2

pop 按索引删除一个元素并返回该元素。

li = ['a','b','c','d']

li.pop(2)#=> 'c'

li#=> ['a', 'b', 'd']

    1

    2

    3

52. 举一个递推式构造字典（dictionary comprehension）的例子

下面我们将创建一个字典，其中字母表中的字母作为键，并以字母索引作为值。

# creating a list of lettersimport stringlist(string.ascii\_lowercase)alphabet = list(string.ascii\_lowercase)# list comprehensiond = {val:idx for idx,val in enumerate(alphabet)}d#=> {'a': 0,#=> 'b': 1,#=> 'c': 2,#=> ...#=> 'x': 23,#=> 'y': 24,#=> 'z': 25}

    1

53. Python中的异常处理是如何进行的？

Python提供了3个关键字来处理异常，try、except和finally。

语法如下：

try:  # try to do thisexcept:  # if try block fails then do thisfinally:  # always do this

    1

在下面的简单示例中，try块失败，因为我们不能将字符串添加到整数中。except块设置val=10，然后finally块打印出“complete”。

try: val = 1 + 'A'except: val = 10finally: print('complete')

print(val)#=> complete#=> 10

    由于 Python 内部引用计数的特性，对于不可变对象，浅拷贝和深拷贝的作用是一致的，就相当于复制了一份副本，原对象内部的不可变对象的改变，不会影响到复制对象浅拷贝的拷贝。其实是拷贝了原始元素的引用（内存地址），所以当拷贝可变对象时，原对象内可变对象的对应元素的改变，会在复制对象的对应元素上，有所体现深拷贝在遇到可变对象时，又在内部做了新建了一个副本。所以，不管它内部的元素如何变化，都不会影响到原来副本的可变对象

import copy

a = "张小鸡"

b = a

c = copy.copy(a)

d = copy.deepcopy(a)

print "赋值：id(b)->>>", id(b)

print "浅拷贝：id(c)->>>", id(c)

print "深拷贝：id(d)->>>", id(c)

1.赋值: 只是复制了新对象的引用，不会开辟新的内存空间。

并不会产生一个独立的对象单独存在，只是将原有的数据块打上一个新标签，所以当其中一个标签被改变的时候，数据块就会发生变化，另一个标签也会随之改变。

2.浅拷贝: 创建新对象，其内容是原对象的引用。

浅拷贝有三种形式： 切片操作，工厂函数，copy模块中的copy函数。

如： lst = [1,2,[3,4]]

切片操作：lst1 = lst[:] 或者 lst1 = [each for each in lst]

工厂函数：lst1 = list(lst)

copy函数：lst1 = copy.copy(lst)

浅拷贝之所以称为浅拷贝，是它仅仅只拷贝了一层，拷贝了最外围的对象本身，内部的元素都只是拷贝了一个引用而已，在lst中有一个嵌套的list[3,4]，如果我们修改了它，情况就不一样了。

浅复制要分两种情况进行讨论：

1）当浅复制的值是不可变对象（字符串、元组、数值类型）时和“赋值”的情况一样，对象的id值（id()函数用于获取对象的内存地址）与浅复制原来的值相同。

2）当浅复制的值是可变对象（列表、字典、集合）时会产生一个“不是那么独立的对象”存在。有两种情况：

第一种情况：复制的对象中无复杂子对象，原来值的改变并不会影响浅复制的值，同时浅复制的值改变也并不会影响原来的值。原来值的id值与浅复制原来的值不同。

第二种情况：复制的对象中有复杂子对象（例如列表中的一个子元素是一个列表），如果不改变其中复杂子对象，浅复制的值改变并不会影响原来的值。 但是改变原来的值中的复杂子对象的值会影响浅复制的值。

3.深拷贝：和浅拷贝对应，深拷贝拷贝了对象的所有元素，包括多层嵌套的元素。深拷贝出来的对象是一个全新的对象，不再与原来的对象有任何关联。

所以改变原有被复制对象不会对已经复制出来的新对象产生影响。

只有一种形式，copy模块中的deepcopy函数

1、什么是 Python？

Python 是一种编程语言，它有对象、模块、线程、异常处理和自动内存管理，可以加入其他语言的对比。

Python 是一种解释型语言，Python 在代码运行之前不需要解释。

Python 是动态类型语言，在声明变量时，不需要说明变量的类型。

Python 适合面向对象的编程，因为它支持通过组合与继承的方式定义类。

在 Python 语言中，函数是第一类对象。

Python 代码编写快，但是运行速度比编译型语言通常要慢。

Python 用途广泛，常被用走"胶水语言"，可帮助其他语言和组件改善运行状况。

使用 Python，程序员可以专注于算法和数据结构的设计，而不用处理底层的细节。

2、赋值、浅拷贝和深拷贝的区别？

(1) 赋值

在 Python 中，对象的赋值就是简单的对象引用，这点和 C++不同，如下所示：

a = [1,2,"hello",['python', 'C++']]

b = a

    1

    2

在上述情况下，a 和 b 是一样的，他们指向同一片内存，b 不过是 a 的别名，是引用。

我们可以使用 b is a 去判断，返回 True，表明他们地址相同，内容相同，也可以使用 id() 函数来查

看两个列表的地址是否相同。

赋值操作(包括对象作为参数、返回值)不会开辟新的内存空间，它只是复制了对象的引用。也就是说除了 b 这个名字之外，没有其他的内存开销。修改了 a，也就影响了 b，同理，修改了 b，也就影响了 a。

(2) 浅拷贝

浅拷贝会创建新对象，其内容非原对象本身的引用，而是原对象内第一层对象的引用。

浅拷贝有三种形式:切片操作、工厂函数、copy 模块中的 copy 函数。

比如上述的列表 a，切片操作：b = a[:] 或者 b = [x for x in a]；

工厂函数：b = list(a)；

copy 函数：b = copy.copy(a)；

浅拷贝产生的列表 b 不再是列表 a 了，使用 is 判断可以发现他们不是同一个对象，使用 id 查看，他们也不指向同一片内存空间。但是当我们使用 id(x) for x in a 和 id(x) for x in b 来查看 a 和 b 中元素的地址时，可以看到二者包含的元素的地址是相同的。

在这种情况下，列表 a 和 b 是不同的对象，修改列表 b 理论上不会影响到列表 a。

但是要注意的是，浅拷贝之所以称之为浅拷贝，是它仅仅只拷贝了一层，在列表 a 中有一个嵌套的 list，如果我们修改了它，情况就不一样了。

比如：a[3].append(‘java’)，查看列表 b，会发现列表 b 也发生了变化，这是因为，我们修改了嵌套的 list，修改外层元素，会修改它的引用，让它们指向别的位置，修改嵌套列表中的元素，列表的地址并未发生变化，指向的都是用一个位置。

(3) 深拷贝

深拷贝只有一种形式，copy 模块中的 deepcopy() 函数。

深拷贝和浅拷贝对应，深拷贝拷贝了对象的所有元素，包括多层嵌套的元素。因此，它的时间和空间开销要高。

同样的对列表 a，如果使用 b = copy.deepcopy(a)，再修改列表 b 将不会影响到列表 a，即使嵌套的列表具有更深的层次，也不会产生任何影响，因为深拷贝拷贝出来的对象根本就是一个全新的对象，不再与原来的对象有任何的关联。

(4) 注意点

对于非容器类型，如数字、字符，以及其他的“原子”类型，没有拷贝一说，产生的都是原对象的引用。

如果元组变量值包含原子类型对象，即使采用了深拷贝，也只能得到浅拷贝。

3、init 和\_\_new\_\_的区别？

当我们使用「类名()」创建对象的时候，Python 解释器会帮我们做两件事情：第一件是为对象在内存分配空间，第二件是为对象进行初始化。「分配空间」是\_\_new\_\_ 方法，初始化是\_\_init\_\_方法。

new 方法在内部其实做了两件时期：第一件事是为「对象分配空间」，第二件事是「把对象的引用返回给 Python 解释器」。当 Python 的解释器拿到了对象的引用之后，就会把对象的引用传递给 init 的第一个参数 self，init 拿到对象的引用之后，就可以在方法的内部，针对对象来定义实例属性。

之所以要学习 new 方法，就是因为需要对分配空间的方法进行改造，改造的目的就是为了当使用「类名()」创建对象的时候，无论执行多少次，在内存中永远只会创造出一个对象的实例，这样就可以达到单例设计模式的目的。

4、Python 的变量、对象以及引用？

首先把结论抛出来：

    变量是到内存空间的一个指针，也就是拥有指向对象连接的空间；

    对象是一块内存，表示它们所代表的值；

    引用就是自动形成的从变量到对象的指针。

以下是具体解释：

在 Python 中使用变量的时候不需要提前声明变量及其类型，变量还是会正常工作。在 Python 中，这个是以一种非常流畅的方式完成，下面以 a = 1 为例我们来看一下它到底是个什么情况。

首先是怎么知道创建了变量：对于变量 a，或者说是变量名 a，当程序第一次给它赋值的时候就创建了它，其实真实情况是 Python 在代码运行之前就先去检测变量名，我们不去具体深究这些，你只需要当作是「最开始的赋值创建了变量」。

再者是怎么知道变量是什么类型：其实这个很多人都没有搞清楚，「类型」这个概念不是存在于变量中，而是存在于对象中。变量本身就是通用的，它只是恰巧在某个时间点上引用了当时的特定对象而已。就比如说在表达式中，我们用的那个变量会立马被它当时所引用的特定对象所替代。

上面这个是动态语言明显区别于静态语言的地方，其实对于刚开始来说，如果你适应将「变量」和「对象」分开，动态类型你也就可以很容易理解了。

我们还是以 a = 1 为例，其实从上面的讲述中，我们很容易的可以发现对于 a = 1 这个赋值语句 Python 是如何去执行它的：创建一个代表值 1 的对象 --> 创建一个变量 a --> 将变量 a 和对象 1 连接。 下面我用一个图来更清晰的表示一下：

由上图我们可以看出，变量 a 其实变成了对象 1 的一个引用。如果你学过指针的话，你就会发现在内部「变量其实就是到对象内存空间的一个指针」。

同样还是上图，我们还可以看出在 Python 中「引用」是从变量到对象的连接，它就是一种关系，在内存中以指针的形式实现。

另外，我也打包成 PDF 方便阅读。

在这里插入图片描述

5、创建百万级实例如何节省内存？

可以定义类的 slot 属性，用它来声明实例属性的列表，可以用来减少内存空间的目的。

具体解释：

首先，我们先定义一个普通的 User 类：

class User1:

    def \_\_init\_\_(self, id, name, sex, status):

        self.id = id

        self.name = name

        self.sex = sex

        self.status = status

    1

    2

    3

    4

    5

    6

然后再定义一个带 slot 的类：

class User2:

    \_\_slots\_\_ = ['id', 'name', 'sex', 'status']

    def \_\_init\_\_(self, id, name, sex, status):

        self.id = id

        self.name = name

        self.sex = sex

        self.status = status

    1

    2

    3

    4

    5

    6

    7

接下来创建两个类的实例：

u1 = User1('01', 'rocky', '男', 1)

u2 = User1('02', 'leey', '男', 1)

    1

    2

我们已经知道 u1 比 u2 使用的内存多，我们可以这样来想，一定是 u1 比 u2 多了某些属性，我们分别来看一下 u1 和 u2 的属性：

['\_\_class\_\_', '\_\_delattr\_\_', '\_\_dict\_\_', '\_\_dir\_\_', '\_\_doc\_\_', '\_\_eq\_\_', '\_\_format\_\_', '\_\_ge\_\_', '\_\_getattribute\_\_', '\_\_gt\_\_', '\_\_hash\_\_', '\_\_init\_\_', '\_\_init\_subclass\_\_', '\_\_le\_\_', '\_\_lt\_\_', '\_\_module\_\_', '\_\_ne\_\_', '\_\_new\_\_', '\_\_reduce\_\_', '\_\_reduce\_ex\_\_', '\_\_repr\_\_', '\_\_setattr\_\_', '\_\_sizeof\_\_', '\_\_str\_\_', '\_\_subclasshook\_\_', '\_\_weakref\_\_', 'id', 'name', 'sex', 'status']

    1

['\_\_class\_\_', '\_\_delattr\_\_', '\_\_dir\_\_', '\_\_doc\_\_', '\_\_eq\_\_', '\_\_format\_\_', '\_\_ge\_\_', '\_\_getattribute\_\_', '\_\_gt\_\_', '\_\_hash\_\_', '\_\_init\_\_', '\_\_init\_subclass\_\_', '\_\_le\_\_', '\_\_lt\_\_', '\_\_module\_\_', '\_\_ne\_\_', '\_\_new\_\_', '\_\_reduce\_\_', '\_\_reduce\_ex\_\_', '\_\_repr\_\_', '\_\_setattr\_\_', '\_\_sizeof\_\_', '\_\_slots\_\_', '\_\_str\_\_', '\_\_subclasshook\_\_', 'id', 'name', 'sex', 'status']

    1

乍一看好像差别不大，我们下面具体来看一下差别在哪：

set(dir(u1)) - set(dir(u2))

    1

通过做集合的差集，我们得到 u1 和 u2 在属性上的具体差别：

{'\_\_weakref\_\_', '\_\_dict\_\_'}

    1

在我们不使用弱引用的时候，weakref 并不占用多少内存，那最终这个锅就要 dict 来背了。

下面我们来看一下 dict:

u1.\_\_dict\_\_

    1

输出结果如下所示：

{'id': '01', 'name': 'rocky', 'sex': '男', 'status': 1}

    1

输出一个字典，在它内部我们发现了刚刚在类里定义的属性，这个字典就是为了实例动态绑定属性的一个字典，我们怎么动态绑定呢？比如我们现在没有 u1.level 这个属性，那么我们可以为它动态绑定一个 level 属性，比如 u1.level = 10，然后我们再来考察这个字典：

u1.\_\_dict\_\_

    1

现在输出的结果为：

{'id': '01', 'name': 'rocky', 'sex': '男', 'status': 1, 'level': 10}

    1

这样看到 level 进入到这个字典中。

这样一个动态绑定属性的特性，其实是以牺牲内存为代价的，因为这个 dict 它本身是占用内存的，接下来我们来验证这件事情：

import sys

sys.getsizeof(u1.\_\_dict\_\_)

    1

    2

我们用 sys 模块下的 getsizeof 方法，它可以得到一个对象使用的内存：

112

    1

我们可以看到这个字典占用了 112 的字节。反观 u2，它没有了 dict 这个属性，我们想给它添加一个属性，也是被拒绝的。

u2.level = 10

    1

显示的结果如下所示：

AttributeError: 'User2' object has no attribute 'level'

    1

6、Python 里面如何生成随机数？

在 Python 中用于生成随机数的模块是 random，在使用前需要 import. 如下例子可以酌情列举：

random.random()：生成一个 0-1 之间的随机浮点数

random.uniform(a, b)：生成[a,b]之间的浮点数

random.randint(a, b)：生成[a,b]之间的整数

random.randrange(a, b, step)：在指定的集合[a,b)中，以 step 为基数随机取一个数

random.choice(sequence)：从特定序列中随机取一个元素，这里的序列可以是字符串，列表，元组等。

7、Python 是强语言类型还是弱语言类型？

Python 是强类型的动态脚本语言。

    强类型：不允许不同类型相加。

    动态：不使用显示数据类型声明，且确定一个变量的类型是在第一次给它赋值的时候。

    脚本语言：一般也是解释型语言，运行代码只需要一个解释器，不需要编译。

8、谈一下什么是解释性语言，什么是编译性语言？

计算机不能直接理解高级语言，只能直接理解机器语言，所以必须要把高级语言翻译成机器语言，计算机才能执行高级语言编写的程序。

解释性语言在运行程序的时候才会进行翻译。

编译型语言写的程序在执行之前，需要一个专门的编译过程，把程序编译成机器语言（可执行文件）。

9、Python 中有日志吗?怎么使用？

Python 中有日志，Python 自带 logging 模块，调用 logging.basicConfig()方法，配置需要的日志等级和相应的参数，Python 解释器会按照配置的参数生成相应的日志。

补充知识：

Python 的标准日志模块

Python 标准库中提供了 logging 模块供我们使用。在最简单的使用中，默认情况下 logging 将日志打印到屏幕终端，我们可以直接导入 logging 模块，然后调用 debug，info，warn，error 和 critical 等函数来记录日志，默认日志的级别为 warning，级别比 warning 高的日志才会被显示（critical > error > warning > info > debug），「级别」是一个逻辑上的概念，用来区分日志的重要程度。

import logging

logging.debug('debug message')

logging.info("info message")

logging.warn('warn message')

logging.error("error message")

logging.critical('critical message')

    1

    2

    3

    4

    5

    6

    7

上述代码的执行结果如下所示：

WARNING:root:warn message

ERROR:root:error message

CRITICAL:root:critical message

    1

    2

    3

我在上面说过，用 print 的话会产生大量的信息，从而很难从中找到真正有用的信息。而 logging 中将日志分成不同的级别以后，我们在大多数时间只保存级别比较高的日志信息，从而提高了日志的性能和分析速度，这样我们就可以很快速的从一个很大的日志文件里找到错误的信息。

配置日志格式

我们在用 logging 来记录日志之前，先来进行一些简单的配置：

import logging

logging.basicConfig(filename= 'test.log', level= logging.INFO)

logging.debug('debug message')

logging.info("info message")

logging.warn('warn message')

logging.error("error message")

logging.critical('critical message')

    1

    2

    3

    4

    5

    6

    7

    8

    9

运行上面的代码以后，会在当前的目录下新建一个 test.log 的文件，这个文件中存储 info 以及 info 以上级别的日志记录。运行一次的结果如下所示：

INFO:root:info message

WARNING:root:warn message

ERROR:root:error message

CRITICAL:root:critical message

    1

    2

    3

    4

上面的例子中，我是用 basicConfig 对日志进行了简单的配置，其实我们还可以进行更为复杂些的配置，在此之前，我们先来了解一下 logging 中的几个概念：

Logger：日志记录器，是应用程序中可以直接使用的接口。

Handler：日志处理器，用以表明将日志保存到什么地方以及保存多久。

Formatter：格式化，用以配置日志的输出格式。

    1

    2

    3

上述三者的关系是：一个 Logger 使用一个 Handler，一个 Handler 使用一个 Formatter。那么概念我们知道了，该如何去使用它们呢？我们的 logging 中有很多种方式来配置文件，简单的就用上面所说的 basicConfig，对于比较复杂的我们可以将日志的配置保存在一个配置文件中，然后在主程序中使用 fileConfig 读取配置文件。

基本的知识我们知道了，下面我们来做一个小的题目：日志文件保存所有 debug 及其以上级别的日志，每条日志中要有打印日志的时间，日志的级别和日志的内容。请先自己尝试着思考一下，如果你已经思考完毕请继续向下看：

import logging

logging.basicConfig(

   level= logging.DEBUG,

   format = '%(asctime)s : %(levelname)s : %(message)s',

   filename= "test.log"

)

logging.debug('debug message')

logging.info("info message")

logging.warn('warn message')

logging.error("error message")

logging.critical('critical message')

    1

    2

    3

    4

    5

    6

    7

    8

    9

    10

    11

    12

    13

上述代码的一次运行结果如下：

2018-10-19 22:50:35,225 : DEBUG : debug message

2018-10-19 22:50:35,225 : INFO : info message

2018-10-19 22:50:35,225 : WARNING : warn message

2018-10-19 22:50:35,225 : ERROR : error message

2018-10-19 22:50:35,225 : CRITICAL : critical message

    1

    2

    3

    4

    5

我刚刚在上面说过，对于比较复杂的我们可以将日志的配置保存在一个配置文件中，然后在主程序中使用 fileConfig 读取配置文件。下面我们就来看一个典型的日志配置文件（配置文件名为 logging.conf）：

[loggers]

keys = root

[handlers]

keys = logfile

[formatters]

keys = generic

[logger\_root]

handlers = logfile

[handler\_logfile]

class = handlers.TimedRotatingFileHandler

args = ('test.log', 'midnight', 1, 10)

level = DEBUG

formatter = generic

[formatter\_generic]

format = %(asctime)s %(levelname)-5.5s [%(name)s:%(lineno)s] %(message)s

    1

    2

    3

    4

    5

    6

    7

    8

    9

    10

    11

    12

    13

    14

    15

    16

    17

    18

    19

    20

在上述的日志配置文件中，首先我们在 [loggers] 中声明了一个叫做 root 的日志记录器（logger），在 [handlers] 中声明了一个叫 logfile 的日志处理器（handler），在 [formatters] 中声明了一个名为 generic 的格式化（formatter）。之后在 [logger\_root] 中定义 root 这个日志处理器（logger） 所使用的日志处理器（handler） 是哪个，在 [handler\_logfile] 中定义了日志处理器（handler） 输出日志的方式、日志文件的切换时间等。最后在 [formatter\_generic] 中定义了日志的格式，包括日志的产生时间，级别、文件名以及行号等信息。

有了上述的配置文件以后，我们就可以在主代码中使用 logging.conf 模块的 fileConfig 函数加载日志配置：

import logging

import logging.config

logging.config.fileConfig('logging.conf')

logging.debug('debug message')

logging.info("info message")

logging.warn('warn message')

logging.error("error message")

logging.critical('critical message')

    1

    2

    3

    4

    5

    6

    7

    8

    9

    10

上述代码的运行一次的结果如下所示：

2018-10-19 23:00:02,809 WARNI [root:8] warn message

2018-10-19 23:00:02,809 ERROR [root:9] error message

2018-10-19 23:00:02,809 CRITI [root:10] critical message

    1

    2

    3

10、Python 是如何进行类型转换的？

内建函数封装了各种转换函数，可以使用目标类型关键字强制类型转换，进制之间的转换可以用 int(‘str’，base=‘n’)将特定进制的字符串转换为十进制，再用相应的进制转换函数将十进制转换为目标进制。

可以使用内置函数直接转换的有：

list---->tuple tuple(list)

tuple---->list list(tuple)

    1

    2

另外，我也打包成 PDF 方便阅读。

在这里插入图片描述

11、Python 中的作用域？

Python 中，一个变量的作用域总是由在代码中被赋值的地方所决定。当 Python 遇到一个变量的话它会按照这的顺序进行搜索：

本地作用域(Local)—>当前作用域被嵌入的本地作用域(Enclosing locals)—>全局/模块作用域

(Global)—>内置作用域(Built-in)。

12、什么是 Python 自省？

Python 自省是 Python 具有的一种能力，使程序员面向对象的语言所写的程序在运行时,能够获得对象的类 Python 型。

Python 是一种解释型语言，为程序员提供了极大的灵活性和控制力。

13、什么是 Python 的命名空间？

命名空间，又名 namesapce，是在很多的编程语言中都会出现的术语，趁着这个题顺便给大家仔细介绍一下。

全局变量 & 局部变量

全局变量和局部变量是我们理解命名空间的开始，我们先来看一段代码：

x = 2

def func():

   x = 3

   print('func x ---> ',x)

func()

print('out of func x ---> ',x)

    1

    2

    3

    4

    5

    6

    7

这段代码输出的结果如下：

func x ---> 3

out of func x ---> 2

    1

    2

从上述的结果中可以看出，运行 func()，输出的是 func() 里面的变量 x 所引用的对象 3，之后执行的是代码中的最后一行。这里要区分清楚，前一个 x 输出的是函数内部的变量 x，后一个 x 输出的是函数外的变量 x，两个变量互相不影响，在各自的作用域中起作用。

那个只在函数内起作用的变量就叫 “局部变量”，有了 “局部” 就有相应的 “全部”，但是后者听起来有歧义，所以就叫了 “全局”。

x = 2

def func():

   global x = 3 #注意此处

   print('func x ---> ',x)

func()

print('out of func x ---> ',x)

    1

    2

    3

    4

    5

    6

    7

这段代码中比上段代码多加了一个 global x，这句话的意思是在声明 x 是全局变量，通俗点说就是这个 x 和 函数外的 x 是同一个了，所以结果就成了下面这样：

func x ---> 3

out of func x ---> 3

    1

    2

这样乍一看好像全局变量好强，可以管着函数内外，但是我们还是要注意，全局变量还是谨慎使用的好，因为毕竟内外有别，不要带来混乱。

作用域

作用域，用比较直白的方式来说，就是程序中变量与对象存在关联的那段程序，比如我在上面说的， x = 2 和 x = 3 是在两个不同的作用域中。

通常的，作用域是被分为静态作用域和动态作用域，虽然我们说 Python 是动态语言，但是它的作用域属于静态作用域，即 Python 中的变量的作用域是由该变量所在程序中的位置所决定的。

在 Python 中作用域被划分成四个层级，分别是：local（局部作用域），enclosing（嵌套作用域），global（全局作用域）和 built - in（内建作用域）。对于一个变量，Python 也是按照之前四个层级依次在不用的作用域中查找，我们在上一段代码中，对于变量 x，首先搜索的是函数体内的局部作用域，然后是函数体外的全局作用域，至于这段话具体怎么来理解，请看下面的例子：

def out\_func():

   x = 2

   def in\_func():

       x = 3

       print('in\_func x ---> ',x)

   in\_func()

   print('out\_func x ---> ',x)

x = 4

out\_func()

print('x == ',x)

    1

    2

    3

    4

    5

    6

    7

    8

    9

    10

    11

上述代码运行的结果是：

in\_func x ---> 3

out\_func x ---> 2

x == 4

    1

    2

    3

仔细观察一下上面的代码和运行的结果，你就会发现变量在不同的范围内进行搜索的规律，是不是感觉这些都是以前被你忽略的呢？

命名空间

《维基百科》中说 “命名空间是对作用域的一种特殊的抽象”，在这里我用一个比方来具体说明一下：

比如张三在公司 A，他的工号是 111，李四在公司 B，他的工号也是 111，因为两个人在不同的公司，他们俩的工号可以相同但是不会引起混乱，这里的公司就表示一个独立的命名空间，如果两个人在一个公司的话，他们的工号就不能相同，否则光看工号也不知道到底是谁。

其实上面举的这个例子的特点就是我们使用命名空间的理由，在大型的计算机程序中，往往会出现成百上千的标识符，命名空间提供隐藏区域标识符的机制。通过将逻辑上相关的标识符构成响应的命名空间，可以使整个系统更加的模块化。

我在开头引用的《维基百科》的那句话说 “命名空间是对作用域的一种特殊的抽象”，它其实包含了处于该作用域内的标识符，且它本身也用一个标识符来表示。在 Python 中，命名空间本身的标识符也属于更外层的一个命名空间，所以命名空间也是可以嵌套的，它们共同生活在 “全局命名空间” 下。

简言之，不同的命名空间可以同时存在，但是彼此独立，互不干扰。当然了，命名空间因为其对象的不同也有所区别，可以分为以下几种：

    1.本地命名空间：模块中有函数或者类的时候，每个函数或者类所定义的命名空间即是本地命名空间，当函数返回结果或者抛出异常的时候，本地命名空间也就结束了。

    2.全局命名空间：每个模块创建了自己所拥有的全局命名空间，不同模块的全局命名空间彼此独立，不同模块中相同名称的命名空间也会因为模块的不同而不相互干扰。

    3.内置命名空间：当 Python 运行起来的时候，它们就存在了，内置函数的命名空间都属于内置命名空间，所以我们可以在任何程序中直接运行它们。

程序查询命名空间的时候也有一套顺序，依次按照本地命名空间 ，全局命名空间，内置命名空间。

def fun(like):

   name = 'rocky'

   print(locals())

fun('python')

    1

    2

    3

    4

    5

访问本地命名空间使用 locals 完成，我们来看一下结果：

{'name': 'rocky', 'like': 'python'}

    1

从上面的结果中可以看出，命名空间中的数据存储的结构和字典是一样的。可能你已经猜到了，当我们要访问全局命名空间的时候，可以使用 globals。

关于命名空间还有一个生命周期的问题，就是一个命名空间什么时候出现，什么时候消失，这个很好理解，就是哪部分被读入内存，哪部分的命名空间就存在了，比如我们在上面说的，Python 启动，内置命名空间就建立。

14、你所遵循的代码规范是什么？

PEP 8 编码风格

Python 代码从第一眼看上去，给人的感觉就是简洁优美，可读性强，也就是我们日常所说的「高颜值」。一方面是因为 Python 自身的优秀设计，比如统一的锁进，没有多余的符号从而让代码变的更加简洁；另一方面就是因为它有着一套较为统一的编码风格，当然它本身只是编码风格方面的建议而不是强制，相应的在编写 Python 代码的编辑器自动提供 PFP 8 检查，当你编写的代码违反了 PEP 8 规范的时候，会给出警告信息和修正的建议。与此同时，还有专门的检查工具对 Python 的代码风格进行检查。

由上，还是建议在编写 Python 代码的时候都遵循 PEP 8 编码规范，毕竟你以后不可能是只一个人写代码，未来不论是在公司或者某些开源项目中，作为其中的一份子，肯定还是要在风格上向大众看齐的。

PEP 8 编码规范详细的给出了 Python 编码的指导，包括什么对齐啦，包的导入顺序啦，空格和注释啦还有命名习惯等方方面面，并且还有详细的事例。

下面我以「包」的导入为例，看一下 PEP 8 给出的具体编程指导。在 Python 中， import 应该一次只导入一个模块，不同的模块应该独立一行：

import pandas

import numpy

    1

    2

反面例子：

import pandas,numpy

    1

如果想要从一个模块里面导入多个，也可以像下面这样：

from subprocess import Popen, PIPE

    1

import 语句应该处于源码文件的顶部，位于模块注释和文档字符串之后，全局变量和常量之前。在导入不同的库的时候，应该按照以下的顺序分组，各个分组之间以空行分隔：

    导入标准库模块

    导入相关第三方库模块

    导入当前应用程序/库模块

具体事例如下所示：

import os

import time

import psutil

from test import u\_test,my\_test

    1

    2

    3

    4

    5

    6

Python 中还支持相对导入和绝对导入，在这里还是强推绝对导入。因为绝对导入的可读性更好一些，也不容易出错，即使出错了也会给出更加详细的错误信息。具体如下所示：

from sub\_package import tools

from sub\_package.tools  import msg

    1

    2

当然除了上述以外还有更多对于包的规范的描述，PEP 8 的编码风格指导比较长，并且写的非常详细，所以我就不在这一一介绍了，详细的可以参考 Python 官网上的资料。

pycodestyle 检查代码规范

我在上面说过 PEP 8 只是官方给出的 Python 编码规范，并没有强制要求大家都遵守，但是又由于大家都在用，所以它也就变成了事实上的 Python 代码风格标准，既然都是标准了，那么就应该有工具来检查这个标准，这样可以帮助 Python 小白规范自己的代码，也可以帮助大家在开源或者工作中形成统一的代码风格。

为了达成上述的目的，官方提供了同名的命令行工具来检查 Python 代码是否违反了 PEP 8 规范，并且对违反规范的地方给出了相应的提示信息。

pip install pep8

    1

规范的名字是 PEP 8 ，这个检查代码风格的命令行工具叫 pep8，这个很容易引起大家的困惑，因此 Python 之父建议将 pep8 重新命名为 pycodestyle，下面我们来看一下 pycodestyle 的用法。

首先通过 pip 安装一下：

pip install pycodestyle

    1

对一个或者多个文件运行 pycodestyle，打印检查报告：

在这里插入图片描述

通过 --show-source 显示不符合规范的源码，以便程序员进行修改，具体如下所示：

在这里插入图片描述

autopep8 格式化代码

autopep8 能够将 Python 代码自动格式化为 PEP 8 风格，它使用 pycodestyle 工具来决定代码中的哪部分需要被格式化，这能够修复大部分 pycodestyle 工具中报告的排版问题。autopep8 本身也是一个用 Python 写的工具，所以我们还是可以用 pip 直接安装：

pip install autopep8

    1

它的使用方式也很简单，具体如下所示：

autopep8 --in-place test\_search.py

    1

上述代码如果不带 --in-place 的话，会将 autopep8 格式化以后的代码直接输出到控制台。我们可以用这种方式检查 autopep8 的修改，使用 --in-place 则会直接将结果保存到源文件中。在这我继续用上面的例子中用到的 py 文件，具体如下所示：

在这里插入图片描述

上面的例子中，autopep8 顺利的修复了所有的问题，但是如果你这个时候查看源文件的话，你会发现源文件的内容还是和原来一样，并没有被修改。这个时候我们就要用到 --in-place，加上这个选项将不会有任何输出， autopep8 会直接修改源文件。

autopep8 --in-place test\_search.py

    1

15、关于 Python 程序的运行方面，有什么手段能提升性能？

1、使用多进程，充分利用机器的多核性能

2、对于性能影响较大的部分代码，可以使用 C 或 C++ 编写

3、对于 IO 阻塞造成的性能影响，可以使用 IO 多路复用来解决

4、尽量使用 Python 的内建函数

5、尽量使用局部变量

另外，我也打包成 PDF 方便阅读。

在这里插入图片描述

16、dict 的 items() 方法与 iteritems() 方法的不同？

items方法将所有的字典以列表方式返回，其中项在返回时没有特殊的顺序

iteritems方法有相似的作用，但是返回一个迭代器对象

17、os.path和sys.path的区别？

os.path是module，包含了各种处理长文件名(路径名)的函数。

sys.path是由目录名构成的列表，Python 从中查找扩展模块( Python 源模块, 编译模块,或者二进制扩展). 启动 Python 时,这个列表从根据内建规则,PYTHONPATH 环境变量的内容, 以及注册表( Windows 系统)等进行初始化。

18、4G 内存怎么读取一个 5G 的数据？

方法一：

通过生成器，分多次读取，每次读取数量相对少的数据（比如 500MB）进行处理，处理结束后

在读取后面的 500MB 的数据。

方法二：

可以通过 linux 命令 split 切割成小文件，然后再对数据进行处理，此方法效率比较高。可以按照行

数切割，可以按照文件大小切割。

    在Linux下用split进行文件分割：

    模式一：指定分割后文件行数

    对与txt文本文件，可以通过指定分割后文件的行数来进行文件分割。

    命令：split -l 300 large\_file.txt new\_file\_prefix

    模式二：指定分割后文件大小

    split -b 10m server.log waynelog

###　19、输入某年某月某日，判断这一天是这一年的第几天？

使用 Python 标准库 datetime

import datetime

def dayofyear():

    year = input("请输入年份：")

    month = input("请输入月份：")

    day = input("请输入天：")

    date1 = datetime.date(year=int(year)，month=int(month)，day=int(day))

    date2 = datetime.date(year=int(year)，month=1，day=1)

    return (date1-date2+1).days

    1

    2

    3

    4

    5

    6

    7

    8

    9

    10

    11

20、说明一下 os.path 和 sys.path 分别代表什么？

os.path 主要是用于对系统路径文件的操作。

sys.path 主要是对 Python 解释器的系统环境参数的操作（动态的改变 Python 解释器搜索路径）。

21、Python 中的 os 模块常见方法？

os.remove() 删除文件

os.rename() 重命名文件

os.walk() 生成目录树下的所有文件

os.chdir() 改变目录

os.mkdir/makedirs 创建目录/多层目录

os.rmdir/removedirs 删除目录/多层目录

os.listdir() 列出指定目录的文件

os.getcwd() 取得当前工作目录

os.chmod() 改变目录权限

os.path.basename() 去掉目录路径，返回文件名

os.path.dirname() 去掉文件名，返回目录路径

os.path.join() 将分离的各部分组合成一个路径名

os.path.split() 返回（dirname(),basename())元组

os.path.splitext() 返回(filename,extension)元组

os.path.getatime\ctime\mtime 分别返回最近访问、创建、修改时间

os.path.getsize() 返回文件大小

os.path.exists() 是否存在

os.path.isabs() 是否为绝对路径

os.path.isdir() 是否为目录

os.path.isfile() 是否为文件

22、说一下字典和 json 的区别？

字典是一种数据结构，json 是一种数据的表现形式，字典的 key 值只要是能 hash 的就行，json 的必须是字符串。

23、什么是可变、不可变类型？

可变不可变指的是内存中的值是否可以被改变，不可变类型指的是对象所在内存块里面的值不可以改变，有数值、字符串、元组；可变类型则是可以改变，主要有列表、字典。

24、存入字典里的数据有没有先后排序？

存入的数据不会自动排序，可以使用 sort 函数对字典进行排序。

25、lambda 表达式格式以及应用场景？

lambda函数就是可以接受任意多个参数（包括可选参数）并且返回单个表达式值得函数。

语法：lambda [arg1 [,arg2,…argn]]:expression

def calc(x,y):

    return x\*y

    1

    2

将上述一般函数改写为匿名函数：

lambda x,y:x\*y

    1

应用

(1) lambda函数比较轻便，即用即仍，适合完成只在一处使用的简单功能。

(2) 匿名函数，一般用来给filter，map这样的函数式编程服务

(3) 作为回调函数，传递给某些应用，比如消息处理。

26、如何理解 Python 中字符串中的\字符？

1、转义字符

2、路径名中用来连接路径名

3、编写太长代码手动软换行

27、常用的 Python 标准库都有哪些？

os 操作系统、time 时间、random 随机、pymysql 连接数据库、threading 线程、multiprocessing

进程、queue 队列

第三方库：

django、flask、requests、virtualenv、selenium、scrapy、xadmin、celery、re、hashlib、md5

常用的科学计算库：Numpy，Pandas、matplotlib

28、如何在Python中管理内存？

python中的内存管理由Python私有堆空间管理。所有Python对象和数据结构都位于私有堆中。程序员无权访问此私有堆。python解释器负责处理这个问题。

Python对象的堆空间分配由Python的内存管理器完成。核心API提供了一些程序员编写代码的工具。

Python还有一个内置的垃圾收集器，它可以回收所有未使用的内存，并使其可用于堆空间。

29、介绍一下 except 的作用和用法？

except: 捕获所有异常

except:<异常名>: 捕获指定异常

except:<异常名 1, 异常名 2>: 捕获异常 1 或者异常 2

except:<异常名>,<数据>: 捕获指定异常及其附加的数据

except:<异常名 1,异常名 2>:<数据>: 捕获异常名 1 或者异常名 2,及附加的数据

30、在 except 中 return 后还会不会执行 finally 中的代码？怎么抛出自定义异常？

会继续处理 finally 中的代码；

用 raise 方法可以抛出自定义异常。

31、read、readline 和 readlines 的区别？

read:读取整个文件。

readline：读取下一行，使用生成器方法。

readlines：读取整个文件到一个迭代器以供我们遍历。

另外，我也打包成 PDF 方便阅读。

在这里插入图片描述

32、range 和 xrange 的区别？

两者用法相同，不同的是 range 返回的结果是一个列表，而 xrange 的结果是一个生成器，前者是直接开辟一块内存空间来保存列表，后者是边循环边使用，只有使用时才会开辟内存空间，所以当列表

很长时，使用 xrange 性能要比 range 好。

33、请简述你对 input()函数的理解？

在 Python3 中，input()获取用户输入，不论用户输入的是什么，获取到的都是字符串类型的。

在 Python2 中有 raw\_input()和 input(), raw\_input()和 Python3 中的 input()作用是一样的，

input()输入的是什么数据类型的，获取到的就是什么数据类型的。

34、代码中要修改不可变数据会出现什么问题？抛出什么异常？

代码不会正常运行，抛出 TypeError 异常。

35、print 调用 Python 中底层的什么方法？

print 方法默认调用 sys.stdout.write 方法，即往控制台打印字符串。

36、Python 的 sys 模块常用方法

sys.argv 命令行参数 List，第一个元素是程序本身路径

sys.modules.keys() 返回所有已经导入的模块列表

sys.exc\_info() 获取当前正在处理的异常类,exc\_type、exc\_value、exc\_traceback 当前处理的

异常详细信息

sys.exit(n) 退出程序，正常退出时 exit(0)  sys.hexversion 获取 Python 解释程序的版本值，16 进制格式如：0x020403F0

sys.version 获取 Python 解释程序的版本信息

sys.maxint 最大的 Int 值

sys.maxunicode 最大的 Unicode 值

sys.modules 返回系统导入的模块字段，key 是模块名，value 是模块

sys.path 返回模块的搜索路径，初始化时使用 PYTHONPATH 环境变量的值

sys.platform 返回操作系统平台名称

sys.stdout 标准输出

sys.stdin 标准输入

sys.stderr 错误输出

sys.exc\_clear() 用来清除当前线程所出现的当前的或最近的错误信息

sys.exec\_prefix 返回平台独立的 python 文件安装的位置

sys.byteorder 本地字节规则的指示器，big-endian 平台的值是’big’,little-endian 平台的值是

‘little’  sys.copyright 记录 python 版权相关的东西

sys.api\_version 解释器的 C 的 API 版本

sys.version\_info 元组则提供一个更简单的方法来使你的程序具备 Python 版本要求功能

37、unittest 是什么？

在 Python 中，unittest 是 Python 中的单元测试框架。它拥有支持共享搭建、自动测试、在测试

中暂停代码、将不同测试迭代成一组等的功能。

38、模块和包是什么？

在 Python 中，模块是搭建程序的一种方式。每一个 Python 代码文件都是一个模块，并可以引用

其他的模块，比如对象和属性。

一个包含许多 Python 代码的文件夹是一个包。一个包可以包含模块和子文件夹。

39、什么是正则的贪婪匹配？

>>>re.search('ab\*c', 'abcaxc')

<\_sre.SRE\_Match object; span=(0, 3), match='abc'>

>>>re.search('ab\D+c', 'abcaxc')

<\_sre.SRE\_Match object; span=(0, 6), match='abcaxc'>

    1

    2

    3

    4

    5

贪婪匹配：正则表达式一般趋向于最大长度匹配，也就是所谓的贪婪匹配。

非贪婪匹配：就是匹配到结果就好，就少的匹配字符。

另外，我也打包成 PDF 方便阅读。

在这里插入图片描述

40、常用字符串格式化哪几种？

% 格式化字符串操作符

print 'hello %s and %s' % ('df', 'another df')

    1

字典形式的字符串格式化方法

print 'hello %(first)s and %(second)s' % {'first': 'df', 'second': 'another df'}

    1

字符串格式化（format）

(1) 使用位置参数

位置参数不受顺序约束，且可以为{}，参数索引从0开始，format里填写{}对应的参数值。

>>> msg = "my name is {}, and age is {}"

>>> msg.format("hqs",22)

'my name is hqs, and age is 22'

    1

    2

    3

(2) 使用关键字参数

关键字参数值要对得上，可用字典当关键字参数传入值，字典前加\*\*即可

>>> hash = {'name':'john' , 'age': 23}

>>> msg = 'my name is {name}, and age is {age}'

>>> msg.format(\*\*hash)

'my name is john,and age is 23'

    1

    2

    3

    4

(3) 填充与格式化

:[填充字符][对齐方式 <^>][宽度]

>>> '{0:\*<10}'.format(10)      # 左对齐

'10\*\*\*\*\*\*\*\*'

    1

    2

41、面向对象深度优先和广度优先是什么？

在子类继承多个父类时，属性查找方式分深度优先和广度优先两种。

当类是经典类时，多继承情况下，在要查找属性不存在时，会按照深度优先方式查找下去。

当类是新式类时，多继承情况下，在要查找属性不存在时，会按照广度优先方式查找下去。

42、“一行代码实现 xx”类题目

(1) 一行代码实现 1 - 100 的和

可以利用 sum() 函数。

在这里插入图片描述

(2) 一行代码实现数值交换

不用二话，直接换。

在这里插入图片描述

(3) 一行代码求奇偶数

使用列表推导式。

在这里插入图片描述

(4) 一行代码展开列表

使用列表推导式，稍微复杂一点，注意顺序。

在这里插入图片描述

(5) 一行代码打乱列表

用到 random 的 shuffle。

在这里插入图片描述

(6) 一行代码反转字符串

使用切片。

在这里插入图片描述

(7) 一行代码查看目录下所有文件

使用 os 的 listdir。

在这里插入图片描述

(8) 一行代码去除字符串间的空格

法 1 replace 函数。

在这里插入图片描述

法 2 join & split 函数。

在这里插入图片描述

(9) 一行代码实现字符串整数列表变成整数列表

使用 list & map & lambda。

在这里插入图片描述

(10) 一行代码删除列表中重复的值

使用 list & set。

在这里插入图片描述

\*\*(11) 一行代码实现 9 \* 9 乘法表

稍稍复杂的列表推导式，耐心点就行，一点点的搞…

在这里插入图片描述

(12) 一行代码找出两个列表中相同的元素

使用 set 和 &。

在这里插入图片描述

(13) 一行代码找出两个列表中不同的元素

使用 set 和 ^。

在这里插入图片描述

(14)一行代码合并两个字典

使用 Update 函数。

在这里插入图片描述

(15) 一行代码实现字典键从小到大排序

使用 sort 函数。

1. 解释型和编译型语言的区别

    编译型语言：把做好的源程序全部编译成二进制的可运行程序。然后，可直接运行这个程序。如：C，C++ ；

    解释型语言：把做好的源程序翻译一句，然后执行一句，直至结束！如：Python。

    注意：Java 有些特殊，java程序也需要编译，但是没有直接编译成为机器语言，而是编译称为字节码，然后用解释方式执行字节码。

\quad

\quad

2. 简述下 Python 中的字符串、列表、元组和字典

    字符串（str）：字符串是用引号括起来的任意文本，是编程语言中最常用的数据类型。

    列表（list）：列表是有序的集合，可以向其中添加或删除元素。

    元组（tuple）：元组也是有序集合，元组中的数无法修改。即元组是不可变的。

    字典（dict）：字典是无序的集合，是由键值对（key-value）组成的。

    集合（set）：是一组 key 的集合，每个元素都是唯一，不重复且无序的。

\quad

\quad

3. 简述上述数据类型的常用方法

字符串：

    切片：'luobodazahui'[1:3]

    format："welcome to luobodazahui, dear {name}"format(name="baby")

    join：可以用来连接字符串，将字符串、元组、列表中的元素以指定的字符(分隔符)连接生成一个新的字符串。'-'.join(['luo', 'bo', 'da', 'za', 'hui'])

    String.replace(old,new,count)：将字符串中的 old字符替换为 New字符，count为替换的个数 'luobodazahui-haha'.replace('haha', 'good')

    split：切割字符串，得到一个列表

>>> mystr5 = 'luobo,dazahui good'

>>> print(mystr5.split())  # 默认以空格分割

['luobo,dazahui', 'good']

>>> print(mystr5.split('h'))  # 以h分割

['luobo,daza', 'ui good']

>>> print(mystr5.split(','))  # 以逗号分割

['luobo', 'dazahui good']

    1

    2

    3

    4

    5

    6

    7

    8

    9

    10

列表：

    切片，同字符串

    append和 extend向列表中添加元素

>>> mylist1 = [1, 2]

>>> mylist2 = [3, 4]

>>> mylist3 = [1, 2]

>>> mylist1.append(mylist2)

>>> print(mylist1)

[1, 2, [3, 4]]

>>> mylist3.extend(mylist2)

>>> print(mylist3)

[1, 2, 3, 4]

    1

    2

    3

    4

    5

    6

    7

    8

    9

    10

    11

    删除元素 del：

        根据下标进行删除pop：删除最后一个元素

        remove：根据元素的值进行删除

>>> mylist4 = ['a', 'b', 'c', 'd']

>>> del mylist4[0]

>>> print(mylist4)

['b', 'c', 'd']

>>> mylist4.pop()

>>> print(mylist4)

['b', 'c']

>>> mylist4.remove('c')

>>> print(mylist4)

['b']

    1

    2

    3

    4

    5

    6

    7

    8

    9

    10

    11

    12

    13

    元素排序 sort：是将list按特定顺序重新排列，默认为由小到大，参数 reverse=True可改为倒序，由大到小。

>>> mylist5 = [1, 5, 2, 3, 4]

>>> mylist5.sort()

>>> print(mylist5)

[1, 2, 3, 4, 5]

>>> mylist5.reverse()

>>> print(mylist5)

[5, 4, 3, 2, 1]

    1

    2

    3

    4

    5

    6

    7

    reverse：是将list逆置。

字典：

    清空字典 dict.clear()

>>> dict1 = {'key1':1, 'key2':2}

>>> dict1.clear()

>>> dict1

{}

    1

    2

    3

    4

    指定删除：使用 pop方法来指定删除字典中的某一项（随机的）。

>>> dict1 = {'key1':1, 'key2':2}

>>> d1 = dict1.pop('key1')

>>> dict1

{'key2': 2}

>>> d1

1

    1

    2

    3

    4

    5

    6

    遍历字典

>>> dict2 = {'key1':1, 'key2':2}

>>> mykey = [key for key in dict2]  # ['key1', 'key2']

>>> mykey

['key1', 'key2']

>>> myvalue = [value for value in dict2.values()]

>>> myvalue

[1, 2]

>>> key\_value = [(k, v) for k, v in dict2.items()]

>>> key\_value

[('key1', 1), ('key2', 2)]

    1

    2

    3

    4

    5

    6

    7

    8

    9

    10

    fromkeys用于创建一个新字典，以序列中元素做字典的键，value为字典所有键对应的初始值。

>>> keys = ['zhangfei', 'guanyu', 'liubei', 'zhaoyun']

>>> dict.fromkeys(keys, 0)

{'zhangfei': 0, 'guanyu': 0, 'liubei': 0, 'zhaoyun': 0}

    1

    2

    3

\quad

\quad

4. 简述 Python 中的字符串编码

计算机在最初的设计中，采用了8个比特（bit）作为一个字节（byte）的方式。一个字节能表示的最大的整数就是255，如果要表示更大的整数，就必须用更多的字节。最早，计算机只有 ASCII 编码，即只包含大小写英文字母、数字和一些符号，这些对于其他语言，如中文，日文显然是不够用的。后来又发明了Unicode，Unicode把所有语言都统一到一套编码里，这样就不会再有乱码问题了。当需要保存到硬盘或者需要传输的时候，就转换为UTF-8编码。UTF-8 是隶属于 Unicode 的可变长的编码方式。

在 Python 中，以 Unicode 方式编码的字符串，可以使用 encode() 方法来编码成指定的 bytes，也可以通过 decode()方法来把 bytes编码成字符串。

>>> "你好".encode('utf-8')

b'\xe4\xbd\xa0\xe5\xa5\xbd'

>>> b'\xe4\xb8\xad\xe6\x96\x87'.decode('utf-8')

"你好"

    1

    2

    3

    4

\quad

\quad

5. 一行代码实现数值交换

>>> a, b = 1, 2

>>> a, b = b, a

>>> print(a, b)

    1

    2

    3

\quad

\quad

6. is 和 == 的区别

==是比较操作符，只是判断对象的值（value）是否一致，而 is 则判断的是对象之间的身份（内存地址）是否一致。对象的身份，可以通过 id() 方法来查看。

>>> c = d = [1, 2]

>>> e = [1, 2]

>>> print(c is d)

True

>>> print(c == d)

True

>>> print(c is e)

False

>>> print(c == e)

True

    1

    2

    3

    4

    5

    6

    7

    8

    9

    10

    11

    12

    13

    14

只有 id一致时，is比较才会返回 True，而当 value一致时，== 比较就会返回 True。

\quad

\quad

7. Python 函数中的参数类型

位置参数，默认参数，可变参数，关键字参数。

\quad

\quad

8. \*arg 和 \*\*kwarg 作用

允许我们在调用函数的时候传入多个实参

>>> def test(\*arg, \*\*kwarg):

...     if arg:

...         print("arg:", arg)

...     if kwarg:

...         print("kearg:", kwarg)

...

>>> test('ni', 'hao', key='world')

arg: ('ni', 'hao')

kearg: {'key': 'world'}

    1

    2

    3

    4

    5

    6

    7

    8

    9

可以看出，\*arg 会把位置参数转化为 tuple，\*\*kwarg 会把关键字参数转化为 dict。

\quad

\quad

9. 获取当前时间

>>> import time

>>> import datetime

>>> print(datetime.datetime.now())

2022-09-12 19:51:24.314335

>>> print(time.strftime('%Y-%m-%d %H:%M:%S'))

2022-09-12 19:51:24

    1

    2

    3

    4

    5

    6

\quad

\quad

10. PEP8 规范

简单列举10条：

    尽量以免单独使用小写字母’l’，大写字母’O’，以及大写字母’I’等容易混淆的字母。

    函数命名使用全部小写的方式，可以使用下划线。

    常量命名使用全部大写的方式，可以使用下划线。

    使用 has或 is前缀命名布尔元素，如: is\_connect = True; has\_member = False。

    不要在行尾加分号，也不要用分号将两条命令放在同一行。

    不要使用反斜杠连接行。

    方法定义之间空1行，顶级定义之间空两行。

    如果一个类不继承自其它类，就显式的从object继承。

    内部使用的类、方法或变量前，需加前缀\_表明此为内部使用的。

    要用断言来实现静态类型检测。

\quad

\quad

11. Python 的深浅拷贝（🧡🧡）

    浅拷贝

>>> import copy

>>> list1 = [1, 2, 3, [1, 2]]

>>> list2 = copy.copy(list1)

>>> list2.append('a')

>>> list2[3].append('a')

>>> list1

[1, 2, 3, [1, 2, 'a']]

>>> list2

[1, 2, 3, [1, 2, 'a'], 'a']

    1

    2

    3

    4

    5

    6

    7

    8

    9

浅拷贝只成功”独立“拷贝了列表的外层，而列表的内层列表，还是共享的。（划重点！！！）

    深拷贝

>>> import copy

>>> list1 = [1, 2, 3, [1, 2]]

>>> list3 = copy.deepcopy(list1)

>>> list3.append('a')

>>> list3[3].append('a')

>>> list1

[1, 2, 3, [1, 2]]

>>> list3

[1, 2, 3, [1, 2, 'a'], 'a']

    1

    2

    3

    4

    5

    6

    7

    8

    9

深拷贝使得两个列表完全独立开来，每一个列表的操作，都不会影响到另一个。

\quad

\quad

12. [lambda x:i\*x for i in range(4)]🧡🧡

这是一道非常经典的题目了。

>>> def num():

...     return [lambda x:i\*x for i in range(4)]

...

>>> [m(1) for m in num()]

[3, 3, 3, 3]

    1

    2

    3

    4

    5

why?

详细看链接：《Python面试题目：[lambda x: x\*i for i in range(4)]》

\quad

\quad

13. 打印九九乘法表

>>> for i in range(1, 10):

...     for j in range(1, i + 1):

...         print(f"{i}\*{j}={i \* j}", end=" ")

...     print()

...

1\*1=1

2\*1=2 2\*2=4

3\*1=3 3\*2=6 3\*3=9

4\*1=4 4\*2=8 4\*3=12 4\*4=16

5\*1=5 5\*2=10 5\*3=15 5\*4=20 5\*5=25

6\*1=6 6\*2=12 6\*3=18 6\*4=24 6\*5=30 6\*6=36

7\*1=7 7\*2=14 7\*3=21 7\*4=28 7\*5=35 7\*6=42 7\*7=49

8\*1=8 8\*2=16 8\*3=24 8\*4=32 8\*5=40 8\*6=48 8\*7=56 8\*8=64

9\*1=9 9\*2=18 9\*3=27 9\*4=36 9\*5=45 9\*6=54 9\*7=63 9\*8=72 9\*9=81

    1

    2

    3

    4

    5

    6

    7

    8

    9

    10

    11

    12

    13

    14

知识点：print函数默认是会换行的，其有一个默认参数 end。

\quad

\quad

14. filter、map、reduce 的作用

    filter 函数用于过滤序列，它接收一个函数和一个序列，把函数作用在序列的每个元素上，然后根据返回值是True还是False决定保留还是丢弃该元素。

>>> mylist = list(range(10))

>>> list(filter(lambda x: x % 2 == 1, mylist))

[1, 3, 5, 7, 9]

    1

    2

    3

    map函数传入一个函数和一个序列，并把函数作用到序列的每个元素上，返回一个可迭代对象。

>>> list(map(lambda x: x % 2, mylist))

[0, 1, 0, 1, 0, 1, 0, 1, 0, 1]

>>> list(map(lambda x: x \* 2, mylist))

[2, 4, 6, 8, 10, 12, 14, 16, 18]

    1

    2

    3

    4

    reduce函数用于递归计算，同样需要传入一个函数和一个序列，并把函数和序列元素的计算结果与下一个元素进行计算。

    reduce()将一个数据集合（链表，元组等）中的所有数据进行下列操作：用传给reduce 中的 function函数 （有两个参数）先对集合中的第 1、2 个元素进行操作，得到的结果再与第三个数据用 function 函数运算，最后得到一个结果。

>>> from functools import reduce

>>> reduce(lambda x, y: x + y, range(101))

5050

    1

    2

    3

reduce()的应用：

    一行代码搞定计算所有元素的积：

reduce(lambda x, y: x \* y, [1, 2, 3, 4])

    1

    计算列表中所有元素的最大值：

reduce(lambda x, y: x if x > y else y, [1, 2, 3, 4])

    1

    有一个列表：[3, 5, 8, 1]对应的是3581的每一个数字，要从这个列表计算出原来的数，可以这样做：

reduce(lambda x, y: x \* 10 + y, [3, 5, 8, 1])

    1

可以看出，上面的三个函数与匿名函数相结合使用，可以写出强大简洁的代码。

\quad

\quad

15. 为什么不建议函数的默认参数传入可变对象

例如：

>>> def test(L=[]):

...     L.append('test')

...     print(L)

...

>>> test()

['test']

>>> test()

['test', 'test']

    1

    2

    3

    4

    5

    6

    7

    8

默认参数是一个列表，是可变对象[]，Python在函数定义的时候，默认参数 L 的值就被计算出来了，是[]，每次调用函数，如果 L 的值变了，那么下次调用时，默认参数的值就已经不再是[]了。

\quad

\quad

16. 面向对象中\_\_new\_\_ 和 \_\_init\_\_ 区别（🧡🧡）

    \_\_new\_\_是在实例创建之前被调用的，因为它的任务就是创建实例然后返回该实例对象，是个静态方法。

    \_\_init\_\_是当实例对象创建完成后被调用的，然后设置对象属性的一些初始值，通常用在初始化一个类实例的时候，是一个实例方法。

    \_\_new\_\_至少要有一个参数cls，代表当前类，此参数在实例化时由 Python 解释器自动识别。

    \_\_new\_\_必须要有返回值，返回实例化出来的实例，这点在自己实现\_\_new\_\_时要特别注意，可以 return 父类（通过 super(当前类名, cls)）\_\_new\_\_出来的实例，或者直接是 object的\_\_new\_\_出来的实例。

    \_\_init\_\_有一个参数 self，就是这个\_\_new\_\_返回的实例，\_\_init\_\_在\_\_new\_\_的基础上可以完成一些其它初始化的动作，\_\_init\_\_不需要返回值。

    如果\_\_new\_\_创建的是当前类的实例，会自动调用\_\_init\_\_函数，通过 return 语句里面调用的\_\_new\_\_函数的第一个参数是 \*\*cls\*\*\*\* 来保证是当前类实例\*\*，如果是其他类的类名，那么实际创建返回的就是其他类的实例，其实就不会调用当前类的\_\_init\_\_函数，也不会调用其他类的\_\_init\_\_函数。

\quad

\quad

17. 三元运算规则

看下面的例子：如果 a>b 成立 就输出 a-b 否则a+b。

>>> >>> a, b = 1, 2

>>> h = a - b if a > b else a + b

>>> h

3

    1

    2

    3

    4

\quad

\quad

18. 生成随机数

>>> import random

>>> random.random()

0.7571910055209727

>>> random.randint(1, 100)

23

>>> random.uniform(1, 5)

3.0640732831151687

    1

    2

    3

    4

    5

    6

    7

    用np.random也可以！

\quad

\quad

19. zip 函数用法

zip() 函数将可迭代的对象作为参数，将对象中对应的元素打包成一个元组，然后返回由这些元组组成的列表。

>>> list1 = ['zhangfei', 'guanyu', 'liubei', 'zhaoyun']

>>> list2 = [0, 3, 2, 4]

>>> list(zip(list1, list2))

[('zhangfei', 0), ('guanyu', 3), ('liubei', 2), ('zhaoyun', 4)]

    1

    2

    3

    4

\quad

\quad

20. range和xrange的区别

只有Python2中才有xrange()和range()，Python3中的range()其实就是Python2中xrange()。

    range([start,] stop[, step])，根据start与stop指定的范围以及step设定的步长，生成一个序列；

    xrange() 生成一个生成器，可以很大的节约内存。

\quad

\quad

21. with方法打开文件的作用

打开文件在进行读写的时候可能会出现一些异常状况，如果按照常规的f.open()写法，我们需要 try，except，finally，做异常判断，并且文件最终不管遇到什么情况，都要执行f.close() 关闭文件，with方法帮我们实现了finally 中 f.close()。

with open("hello.txt", "a") as f:

    f.write("hello world!")

    1

    2

\quad

\quad

22. 字符串转列表

>>> s = "1,2,3,4,5,6,7,8,9"

>>> s.split(",")

['1', '2', '3', '4', '5', '6', '7', '8', '9']

    1

    2

    3

\quad

\quad

23. 字符串转整数

>>> s = "1,2,3,4,5,6,7,8,9"

>>> list(map(lambda x: int(x), s.split(",")))

[1, 2, 3, 4, 5, 6, 7, 8, 9]

    1

    2

    3

\quad

\quad

24. 删除列表中的重复值

利用集合去重：

>>> mylist = [1, 2, 3, 4, 5, 5, 6, 7, 4, 3]

>>> list(set(mylist))

[1, 2, 3, 4, 5, 6, 7]

    1

    2

    3

\quad

\quad

25. 字符串单词统计

统计字符串中字母个数：

>>> from collections import Counter

>>> mystr = 'sdfsfsfsdfsd,were,hrhrgege.sdfwe!sfsdfs'

>>> Counter(mystr)

Counter({'s': 9,

         'd': 5,

         'f': 7,

         ',': 2,

         'w': 2,

         'e': 5,

         'r': 3,

         'h': 2,

         'g': 2,

         '.': 1,

         '!': 1})

    1

    2

    3

    4

    5

    6

    7

    8

    9

    10

    11

    12

    13

    14

统计字符串中单词个数

>>> mystr2 = "hello, Nice to meet you!"

>>> len(mystr2.split(" "))

5

    1

    2

    3

\quad

\quad

26. 列表推导，求奇偶数

>>> [x for x in range(20) if x % 2 == 1]

[1, 3, 5, 7, 9, 11, 13, 15, 17, 19]

    1

    2

\quad

\quad

27. 一行代码展开列表

>>> list1 = [[1, 2], [3, 4], [5, 6]]

>>> [j for i in list1 for j in i]

[1, 2, 3, 4, 5, 6]

    1

    2

    3

\quad

\quad

28. 实现二分法查找函数

二分查找算法也称折半查找，基本思想就是折半，对比大小后再折半查找，必须是有序序列才可以使用二分查找。

非递归算法：

def binary\_search(data, item):

    left = 0

    right = len(data) - 1

    while left <= right:

        mid = (left + right) // 2

        if data[mid] == item:

            return True

        elif data[mid] < item:

            left = mid + 1

        else:

            right = mid - 1

    return False

    1

    2

    3

    4

    5

    6

    7

    8

    9

    10

    11

    12

递归算法：

def binary\_search(data, item):

    n = len(data)

    if n > 0:

        mid = n // 2

        if data[mid] == item:

            return True

        elif data[mid] > item:

            return binary\_search(data[:mid], item)

        else:

            return binary\_search(data[mid + 1:], item)

    return False

    1

    2

    3

    4

    5

    6

    7

    8

    9

    10

    11

\quad

\quad

29. 合并两个元组到字典

如：("zhangfei", "guanyu")，(66, 80) -> {'zhangfei': 66, 'guanyu': 80}。

>>> a = ("zhangfei", "guanyu")

>>> b = (66, 80)

>>> dict(zip(a, b))

{'zhangfei': 66, 'guanyu': 80}

    1

    2

    3

    4

\quad

\quad

30. 给出如下代码的输入，并简单解释

例子1：

>>> a = (1, 2, 3, [4, 5, 6, 7], 8)

>>> a[3] = 2

    1

    2

报错：TypeError: ‘tuple’ object does not support item assignment。

原因：元组不能修改！tuple 是不可变类型，不能改变 tuple 里的元素。

例子2：

>>> a = (1, 2, 3, [4, 5, 6, 7], 8)

>>> a[3][2] = 2

>>> a

(1, 2, 3, [4, 5, 2, 7], 8)

    1

    2

    3

    4

list 是可变类型，改变其元素是允许的。

\quad

\quad

31. 字典和 json 转换

dict1 = {'zhangfei':1, "liubei":2, "guanyu": 4, "zhaoyun":3}

myjson = json.dumps(dict1)    # 字典转JSON

mydict = json.loads(myjson)   # JSON转字典

    1

    2

    3

\quad

\quad

32. 列表推导式、字典推导式和生成器

列表推导式：返回一个列表。

>>> td\_list = [i for i in range(10)]

>>> td\_list

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

>>> type(td\_list)

list

    1

    2

    3

    4

    5

生成器：

>>> ge\_list = (i for i in range(20))

>>> ge\_list

<generator object <genexpr> at 0x000001C3C127AAC8>

>>> type(ge\_list)

generator

    1

    2

    3

    4

    5

字典推导式：

>>> dic = {k: 2 for k in ["a", "b", "c", "d"]}

>>> dic

{'a': 2, 'b': 2, 'c': 2, 'd': 2}

>>> type(dic)

dict

    1

    2

    3

    4

    5

\quad

\quad

33. 简述 read、readline、readlines 的区别

    read() 读取整个文件

    readline() 读取下一行,使用生成器方法

    readlines() 读取整个文件到一个迭代器以供我们遍历

\quad

\quad

34. 打乱一个列表

>>> import random

>>> list = list(range(1, 10))

>>> random.shuffle(list)

>>> list

[3, 9, 1, 4, 6, 2, 8, 7, 5]

    1

    2

    3

    4

    5

\quad

\quad

35. 反转字符串

>>> 'luobodazahui'[::-1]

'iuhazadoboul'

    1

    2

\quad

\quad

36. 单下划线和双下划线的作用

    \_\_foo\_\_：一种约定，Python内部的名字，用来区别其他用户自定义的命名，以防冲突，就是例如\_\_init\_\_()，\_\_del\_\_()，\_\_call\_\_()些特殊方法。

    \_foo：一种约定，用来指定变量私有。不能用from module import \* 导入，其他方面和公有变量一样访问。

    \_\_foo：这个有真正的意义：解析器用\_classname\_\_foo来代替这个名字，以区别和其他类相同的命名，它无法直接像公有成员一样随便访问，通过对象名.\_类名\_\_xxx这样的方式可以访问。

\quad

\quad

37. 新式类和旧式类

    在 python 里凡是继承了 object的类，都是新式类

    Python3 里只有新式类

    Python2 里面继承 object的是新式类，没有写父类的是经典类

    经典类目前在 Python 里基本没有应用

\quad

\quad

38. Python 面向对象中的继承有什么特点

    同时支持单继承与多继承，当只有一个父类时为单继承，当存在多个父类时为多继承；

    子类会继承父类所有的属性和方法，子类也可以覆盖父类同名的变量和方法；

    在继承中基类的构造（\_\_init\_\_()）方法不会被自动调用，它需要在其派生类的构造中专门调用；

    在调用基类的方法时，需要加上基类的类名前缀，且需要带上 self 参数变量。区别于在类中调用普通函数时并不需要带上 self 参数。

\quad

\quad

39. super 函数的作用

super()函数是用于调用父类(超类)的一个方法

class A():

    def funcA(self):

        print("this is func A")

class B(A):

    def funcA\_in\_B(self):

        super(B, self).funcA()

    def funcC(self):

        print("this is func C")

>>> ins = B()

>>> ins.funcA\_in\_B()

this is func A

>>> ins.funcC()

this is func C

    1

    2

    3

    4

    5

    6

    7

    8

    9

    10

    11

    12

    13

    14

    15

    16

\quad

\quad

40. 类中的各种函数

主要分为实例方法、类方法和静态方法。

    实例方法

定义：第一个参数必须是实例对象，该参数名一般约定为“self”，通过它来传递实例的属性和方法（也可以传类的属性和方法）。

调用：只能由实例对象调用。

    类方法

定义：使用装饰器@classmethod。第一个参数必须是当前类对象，该参数名一般约定为“cls”，通过它来传递类的属性和方法（不能传实例的属性和方法）。

调用：实例对象和类对象都可以调用。

    静态方法

定义：使用装饰器@staticmethod。参数随意，没有“self”和“cls”参数，但是方法体中不能使用类或实例的任何属性和方法。

调用：实例对象和类对象都可以调用。

静态方法是类中的函数，不需要实例。静态方法主要是用来存放逻辑性的代码，主要是一些逻辑属于类，但是和类本身没有交互。即在静态方法中，不会涉及到类中的方法和属性的操作。可以理解为将静态方法存在此类的名称空间中。

类方法是将类本身作为对象进行操作的方法。他和静态方法的区别在于：不管这个方式是从实例调用还是从类调用，它都用第一个参数把类传递过来。

\quad

\quad

41. 如何判断是函数还是方法

    与类和实例无绑定关系的 function都属于函数（function）

    与类和实例有绑定关系的 function都属于方法（method）

普通函数：

def func1():

    pass

>>> print(func1)

<function func1 at 0x000001C3C12A4438>

    1

    2

    3

    4

类中的函数：

class People(object):

    def func2(self):

        pass

    @staticmethod

    def func3():

        pass

    @classmethod

    def func4(cls):

        pass

>>> people = People()

>>> print(people.func2)

<bound method People.func2 of <\_\_main\_\_.People object at 0x000001C3C12A6A48>>

>>> print(people.func3)

<function People.func3 at 0x000001C3C12A4A68>

>>> print(people.func4)

<bound method People.func4 of <class '\_\_main\_\_.People'>>

    1

    2

    3

    4

    5

    6

    7

    8

    9

    10

    11

    12

    13

    14

    15

    16

    17

    18

    19

    20

    21

    22

\quad

\quad

42. isinstance 的作用以及与type()的区别（🧡🧡）

isinstance()函数来判断一个对象是否是一个已知的类型，类似type()。

区别：

    type()不会认为子类是一种父类类型，不考虑继承关系；

    isinstance()会认为子类是一种父类类型，考虑继承关系。

class A(object):

    pass

class B(A):

    pass

>>> a = A()

>>> b = B()

>>> print(isinstance(a, A))

True

>>> print(type(a) == A)

True

>>> print(isinstance(b, A))

True

>>> print(type(b) == A)

False

    1

    2

    3

    4

    5

    6

    7

    8

    9

    10

    11

    12

    13

    14

    15

    16

    17

    18

\quad

\quad

43. 单例模式与工厂模式

    单例模式：主要目的是确保某一个类只有一个实例存在；

    工厂模式：包涵一个超类，这个超类提供一个抽象化的接口来创建一个特定类型的对象，而不是决定哪个对象可以被创建。

单例模式：

这种模式涉及到一个单一的类，该类负责创建自己的对象，同时确保只有单个对象被创建。这个类提供了一种访问其唯一的对象的方式，可以直接访问，不需要实例化该类的对象。

意图：保证一个类仅有一个实例，并提供一个访问它的全局访问点。

主要解决：一个全局使用的类频繁地创建与销毁。

举例：操作一个文件，应该用一个唯一的实例去操作。

工厂模式：

工厂类模式设计的核心是：让“生产”和“产品”解耦。

工厂模式的主要解决的问题：将原来分布在各个地方的对象创建过程单独抽离出来，交给工厂类负责创建。其他地方想要使用对象直接找工厂（即调用工厂的方法）获取对象。

优点：

    一个调用者想创建一个对象，只要知道其名称就可以了。

    扩展性高，如果想增加一个产品，只要扩展一个工厂类就可以。

    屏蔽产品的具体实现，调用者只关心产品的接口。

\quad

\quad

44. 查看目录下的所有文件

>>> import os

>>> print(os.listdir('.'))

    1

    2

\quad

\quad

45. 计算由1到5组成的互不重复的三位数

for i in range(1, 6):

    for j in range(1, 6):

        for k in range(1, 6):

            if (i != j) and i != k and j != k:

                print(f"{i}{j}{k}")

    1

    2

    3

    4

    5

\quad

\quad

46. 去除字符串首尾空格

>>> "   hello world    ".strip()

'hello world'

    1

    2

\quad

\quad

47. 去除字符串中间的空格

方法一：

>>> "hello you are good".replace(" ", "")

'helloyouaregood'

    1

    2

方法二：

>>> "".join("hello you are good".split(" "))

'helloyouaregood'

    1

    2

\quad

\quad

48. 字符串格式化方式

方法一：使用 % 操作符

print("This is for %s" % "Python")

print("This is for %s, and %s" %("Python", "You"))

    1

    2

方法二：str.format（在 Python3 中，引入了这个新的字符串格式化方法）

print("This is my {}".format("chat"))

print("This is {name}, hope you can {do}".format(name="zhouluob", do="like"))

    1

    2

方法三：f-strings（在 Python3-6 中，引入了这个新的字符串格式化方法）

name = "luobodazahui"

print(f"hello {name}")

    1

    2

一个复杂些的例子：

def mytest(name, age):

    return f"hello {name}, you are {age} years old!"

>>> people = mytest("luobo", 20)

>>> print(people)

hello luobo, you are 20 years old!

    1

    2

    3

    4

    5

    6

\quad

\quad

49. 将"hello world"转换为首字母大写"Hello World"

title() 函数：

>>> str1 = "hello world"

>>> str1.title()

'Hello World'

    1

    2

    3

不使用 title() 函数

>>> str1 = "hello world"

>>> " ".join(list(map(lambda x: x.capitalize(), str1.split(" "))))

'Hello World'

>>> " ".join(list(map(lambda x: x[0].upper() + x[1:], str1.split(" "))))

'Hello World'

    1

    2

    3

    4

    5

    6

\quad

\quad

50. 一行代码转换列表中的整数为字符串

如：[1, 2, 3] -> [“1”, “2”, “3”]

>>> list1 = [1, 2, 3]

>>> list(map(lambda x: str(x), list1))

['1', '2', '3']

    1

    2

    3

\quad

\quad

51. Python 中的反射（🧡🧡）

反射就是通过字符串的形式，导入模块；通过字符串的形式，去模块寻找指定函数，并执行。利用字符串的形式去对象（模块）中操作（查找/获取/删除/添加）成员，一种基于字符串的事件驱动！

简单理解就是用来判断某个字符串是什么，是变量还是方法。

例子：先定义一个类：

class NewClass(object):

    def \_\_init\_\_(self, name, male):

        self.name = name

        self.male = male

    def myname(self):

        print(f'My name is {self.name}')

    def mymale(self):

        print(f'I am a {self.male}')

    1

    2

    3

    4

    5

    6

    7

    8

    9

    10

>>> people = NewClass('luobo', 'boy')

>>> print(hasattr(people, 'name'))

True

>>> print(getattr(people, 'name'))

luobo

>>> setattr(people, 'male', 'girl')

>>> print(getattr(people, 'male'))

girl

    1

    2

    3

    4

    5

    6

    7

    8

getattr，hasattr，setattr，delattr 对模块的修改都在内存中进行，并不会影响文件中真实内容。

详情请看博客：《Python进阶系列》十八：详解Python中的反射——通过字符串的形式操作对象

\quad

\quad

52. metaclass 元类（🧡🧡）

类与实例：首先定义类以后，就可以根据这个类创建出实例，所以：先定义类，然后创建实例；

类与元类：先定义元类， 根据 metaclass 创建出类，所以：先定义 metaclass，然后创建类。

class MyMetaclass(type):

    def \_\_new\_\_(cls, class\_name, class\_parents, class\_attr):

        class\_attr['print'] = "this is my metaclass's subclass %s" %class\_name

        return type.\_\_new\_\_(cls, class\_name, class\_parents, class\_attr)

    class MyNewclass(object, metaclass=MyMetaclass):

    pass

>>> myinstance = MyNewclass()

>>> myinstance.print

"this is my metaclass's subclass MyNewclass"

    1

    2

    3

    4

    5

    6

    7

    8

    9

    10

    11

详细内容见博客：《Python进阶系列》十五：详解Python中的元类（metaclass）

\quad

\quad

53. sort 和 sorted 的区别

sort()是可变对象列表（list）的方法，无参数，无返回值。sort()会改变可变对象。

>>> list1 = [2, 1, 3]

>>> print(list1.sort())

None

>>> list1

[1, 2, 3]

>>> dict1.sort()

Traceback (most recent call last):

  File "<stdin>", line 1, in <module>

AttributeError: 'dict' object has no attribute 'sort'

    1

    2

    3

    4

    5

    6

    7

    8

    9

sorted()是产生一个新的对象。sorted(L)返回一个排序后的L，不改变原始的L，sorted()适用于任何可迭代容器。

>>> dict1 = {'test1':1, 'test2':2}

>>> list1 = [2, 1, 3]

>>> print(sorted(dict1))

['test1', 'test2']

>>> print(sorted(list1))

[1, 2, 3]

    1

    2

    3

    4

    5

    6

\quad

\quad

54. Python 中的 GIL

GIL 是 Python 的全局解释器锁，同一进程中假如有多个线程运行，一个线程在运行 Python 程序的时候会占用 Python 解释器（加了一把锁即 GIL），使该进程内的其他线程无法运行，等该线程运行完后其他线程才能运行。如果线程运行过程中遇到耗时操作，则解释器锁解开，使其他线程运行。所以在多线程中，线程的运行仍是有先后顺序的，并不是同时进行。

\quad

\quad

55. 产生8位随机密码（🧡🧡）

string 模块用法：

1、将大写的ASCII字符列表和数字组合起来

>>> string.ascii\_uppercase + string.digits

'ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789'

    1

    2

2、string.printable输出被视为可打印符号的 ASCII 字符组成的字符串。

>>> string.printable

'0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ!"#$%&\'()\*+,-./:;<=>?@[\\]^\_`{|}~ \t\n\r\x0b\x0c'

    1

    2

string.printable[:-7]去除被视为空白符号的 ASCII 字符组成的字符串。

使用python的random模块，使用其中的choice方法，从给定的字符序列中随机选择字符组合。

>>> import random

>>> import string

>>> "".join(random.choice(string.printable[:-7]) for i in range(8))

    1

    2

    3

这就产生了8位随机密码。

\quad

\quad

56. 输出原始字符

python提供了输出原始字符串“r”的方法。

>>> print('hello\nworld')

hello

world

>>> print(b'hello\nworld')

b'hello\nworld'

>>> print(r'hello\nworld')

hello\nworld

    1

    2

    3

    4

    5

    6

    7

\quad

\quad

57. 简述 any() 和 all() 方法

all：如果存在 0，Null，False 返回 False，否则返回 True；

any：如果都是 0，None，False时，返回 False。

例子：

>>> all([1, 2, 3, 0])

False

>>> all([1, 2, 3])

True

>>> any([1, 2, 3])

True

>>> any([0, None, False])

False

    1

    2

    3

    4

    5

    6

    7

    8

\quad

\quad

58. 反转整数

首先判断是否是整数，再判断是否是一位数字，最后再判断是不是负数。

 def reverse\_int(x):

     if not isinstance(x, int):

         return False

     if -10 < x < 10:

         return x

     tmp = str(x)

     if tmp[0] != '-':

         tmp = tmp[::-1]

         return int(tmp)

    else:

        tmp = tmp[1:][::-1]

        x = int(tmp)

        return -x

    1

    2

    3

    4

    5

    6

    7

    8

    9

    10

    11

    12

    13

>>> reverse\_int(-23837)

-73832

    1

    2

\quad

\quad

59. 函数式编程（🧡🧡）

函数式编程是一种抽象程度很高的编程范式，纯粹的函数式编程语言编写的函数没有变量，因此，任意一个函数，只要输入是确定的，输出就是确定的，这种纯函数称之为没有副作用。而允许使用变量的程序设计语言，由于函数内部的变量状态不确定，同样的输入，可能得到不同的输出，因此，这种函数是有副作用的。由于 Python 允许使用变量，因此，Python 不是纯函数式编程语言。

函数式编程的一个特点就是，允许把函数本身作为参数传入另一个函数，还允许返回一个函数！

函数作为返回值例子：

 def sum(\*args):

     def inner\_sum():

         tmp = 0

         for i in args:

             tmp += i

         return tmp

     return inner\_sum

>>> mysum = sum(2, 4, 6)

>>> print(type(mysum))

<class 'function'>

>>> mysum()

12

    1

    2

    3

    4

    5

    6

    7

    8

    9

    10

    11

    12

    13

\quad

\quad

60. 简述闭包（🧡🧡）

如果在一个内部函数里，对在外部作用域（但不是在全局作用域）的变量进行引用，那么内部函数就被认为是闭包（closure）。

在这里插入图片描述

闭包特点：

    必须有一个内嵌函数；

    内嵌函数必须引用外部函数中的变量；

    外部函数的返回值必须是内嵌函数。

详细看链接：《Python面试题目：[lambda x: x\*i for i in range(4)]》

\quad

\quad

61. 简述装饰器（🧡🧡）

装饰器是一种特殊的闭包，就是在闭包的基础上传递了一个函数，然后覆盖原来函数的执行入口，以后调用这个函数的时候，就可以额外实现一些功能了。

一个打印 log 的例子：

 import time

 def log(func):

     def inner\_log(\*args, \*\*kw):

         print("Call: {}".format(func.\_\_name\_\_))

         return func(\*args, \*\*kw)

     return inner\_log

@log

def timer():

    print(time.time())

timer()

# Call: timer

# 1560171403.5128365

    1

    2

    3

    4

    5

    6

    7

    8

    9

    10

    11

    12

    13

    14

    15

本质上，decorator就是一个返回函数的高阶函数。

详细看链接：《Python进阶系列》八：装饰器的用法

\quad

\quad

62. 协程的优点

协程的优点：

    最大优势就是协程极高的执行效率。因为子程序切换不是线程切换，而是由程序自身控制，因此，没有线程切换的开销，和多线程比，线程数量越多，协程的性能优势就越明显。

    不需要多线程的锁机制，因为只有一个线程，也不存在同时写变量冲突，在协程中控制共享资源不加锁，只需要判断状态就好了，所以执行效率比多线程高很多。

因为协程是一个线程执行，那怎么利用多核CPU呢？最简单的方法是多进程+协程，既充分利用多核，又充分发挥协程的高效率，可获得极高的性能。

其他一些重要的点：

    协程并没有增加线程数量，只是在线程的基础之上通过分时复用的方式运行多个协程，而且协程的切换在用户态完成，切换的代价比线程从用户态到内核态的代价小很多。

    因此在协程调用阻塞IO操作的时候，操作系统会让线程进入阻塞状态，当前的协程和其它绑定在该线程之上的协程都会陷入阻塞而得不到调度，这往往是不能接受的。

    因此在协程中不能调用导致线程阻塞的操作。也就是说，协程只有和异步IO结合起来，才能发挥最大的威力。

    协程对计算密集型的任务没有太大的好处，计算密集型的任务本身不需要大量的线程切换，因为协程主要解决以往线程或者进程上下文切换的开销问题，所以协程主要对那些I/O密集型应用更好。

协程只有和异步IO结合起来才能发挥出最大的威力。

\quad

\quad

63. 实现斐波那契数列

斐波那契数列：又称黄金分割数列，指的是这样一个数列：1、1、2、3、5、8、13、21、34、……在数学上，斐波纳契数列以如下被以递归的方法定义：

F ( 1 ) = 1 , F ( 2 ) = 1 , F ( n ) = F ( n − 1 ) + F ( n − 2 ) ( n ≥ 2 , n ∈ N ∗ ) F(1)=1,F(2)=1, F(n)=F(n-1)+F(n-2)(n\geq2,n\in N\*) F(1)=1,F(2)=1,F(n)=F(n−1)+F(n−2)(n≥2,n∈N∗)

生成器法：

def fib(n):

    if n == 0:

        return False

    if not isinstance(n, int) or (abs(n) != n):

        return False

    a, b = 0, 1

    while n:

        a, b = b, a + b

        n -= 1

        yield a

>>> fib(10)

<generator object fib at 0x000001AF2F2395C8>

>>> [i for i in fib(10)]

[1, 1, 2, 3, 5, 8, 13, 21, 34, 55]

    1

    2

    3

    4

    5

    6

    7

    8

    9

    10

    11

    12

    13

    14

    15

递归法：

def fib(n):

    if n == 0:

        return False

    if not isinstance(n, int) or (n < 0):

        return False

    if n <= 1:

        return 1

    return fib(n - 1) + fib(n - 2)

>>> fib(10)

55

>>> [fib(i) for i in range(1, 11)]

[1, 1, 2, 3, 5, 8, 13, 21, 34, 55]

    1

    2

    3

    4

    5

    6

    7

    8

    9

    10

    11

    12

    13

\quad

\quad

64. 正则切分字符串

>>> import re

>>> str1 = "hello world:luobo dazahui"

>>> re.split(r":| ", str1)

['hello', 'world', 'luobo', 'dazahui']

    1

    2

    3

    4

\quad

\quad

65. yield 用法

yield 是用来生成迭代器的语法，在函数中，如果包含了 yield，那么这个函数就是一个迭代器。当代码执行至 yield 时，就会中断代码执行，直到程序调用 next() 函数时，才会在上次 yield 的地方继续执行。

详情见：《Python中的迭代器和生成器》 。

\quad

\quad

67. 冒泡排序

def Bubble(data, reversed):

    for i in range(len(data) - 1):

        for j in range(len(data) - i - 1):

            if data[j]  > data[j + 1]:

                data[j], data[j + 1] = data[j + 1], data[j]

    if reversed:

        data.reverse()

    return data

>>> Bubble(list1, True)

[11, 9, 8, 5, 3, 2]

>>> Bubble(list1, False)

[2, 3, 5, 8, 9, 11]

    1

    2

    3

    4

    5

    6

    7

    8

    9

    10

    11

    12

    13

\quad

\quad

68. 快速排序

    思想：首先任意选取一个数据（通常选用数组的第一个数）作为关键数据，然后将所有比它小的数都放到它前面，所有比它大的数都放到它后面，这个过程称为一趟快速排序，之后再递归排序两边的数据。

    挑选基准值：从数列中挑出一个元素，称为"基准"（pivot）。

    分割：重新排序数列，所有比基准值小的元素摆放在基准前面，所有比基准值大的元素摆在基准后面（与基准值相等的数可以到任何一边）。在这个分割结束之后，对基准值的排序就已经完成。

    递归排序子序列：递归地将小于基准值元素的子序列和大于基准值元素的子序列排序。

def partition(arr, low, high):

    pivot = arr[low]

    while low < high:

        while low < high and arr[high] >= pivot:

            high -= 1

        arr[low], arr[high] = arr[high], arr[low]

        while low < high and arr[low] <= pivot:

            low += 1

        arr[low], arr[high] = arr[high], arr[low]

    return low

def quickSort(arr, low, high):

    if low < high:

        pi = partition(arr, low, high)

        quickSort(arr, low, pi - 1)

        quickSort(arr, pi + 1, high)

>>> list1 = [8, 5, 1, 3, 2, 10, 11, 4, 12, 20]

>>> quickSort(list1, 0, len(list1) - 1)

>>> list1

[1, 2, 3, 4, 5, 8, 10, 11, 12, 20]

    1

    2

    3

    4

    5

    6

    7

    8

    9

    10

    11

    12

    13

    14

    15

    16

    17

    18

    19

    20

    21

    22

    23

\quad

\quad

70. requests 简介

该库是发起 HTTP 请求的强大类库，调用简单，功能强大。

>>> import requests

>>> url = "http://www.baidu.com"

>>> response = requests.get(url)  # 获得请求

>>> response.encoding = "utf-8"  # 改变其编码

>>> html = response.text  # 获得网页内容

>>> binary\_content = response.content  # 获得二进制数据

>>> raw = requests.get(url, stream=True)  # 获得原始响应内容

>>> headers = {'user-agent': 'my-test/0.1.1'}  # 定制请求头

>>> r = requests.get(url, headers=headers)

>>> cookies = {"cookie": "# your cookie"}  # cookie的使用

>>> r = requests.get(url, cookies=cookies)

    1

    2

    3

    4

    5

    6

    7

    8

    9

    10

    11

\quad

\quad

71. 比较两个 json 数据是否相等

方法一：

    json数据转换成字典；

    将两个字典按key排好序，然后使用zip()函数将两个字典对应的元素打包成元组。比较对应的元素的value是否相等；

    zip()函数用于将可迭代的对象作为参数，将对象中对应的元素打包成一个个元组。

>>> file\_text2 = '{"name":"john","age":22,"sex":"woman","address":"USA"}'

>>> file\_text1 = '{"name":"john","age":22,"sex":"man","address":"USA"}'

>>> dict1 = json.loads(file\_text1)

>>> dict2 = json.loads(file\_text2)

>>> for s1, s2 in zip(sorted(dict1), sorted(dict2)):

        if str(dict1[s1]) != str(dict2[s2]):

            print(s1 + ":" + dict1[s1] + "!=" + s2 + ":" + dict2[s2])

sex:man!=sex:woman

    1

    2

    3

    4

    5

    6

    7

    8

方法二：引入第三方模块jsonpatch。

>>> import jsonpatch

>>> src = {'numbers': [1, 3, 4, 8], 'foo': 'bar'}

>>> dst = {'foo': 'bar', 'numbers': [1, 3, 8]}

>>> patch = jsonpatch.JsonPatch.from\_diff(src, dst)

>>> print(patch)

[{"op": "remove", "path": "/numbers/2"}]

    1

    2

    3

    4

    5

    6

说明：两个json对象，src要变成dst，需要移除numbers下索引是2的元素。

>>> src = {'numbers': [1, 3, 4, 8], 'foo': 'bar'}

>>> dst = {'foo': 'bar', 'numbers': [1, 3, 4, 8]}

>>> patch = jsonpatch.JsonPatch.from\_diff(src, dst)

>>> print(patch)

[]

    1

    2

    3

    4

    5

若相等的话直接返回空列表。

    有了这个模块，判断json串是否相等不用遍历内部属性，直接判断patch长度即可。

\quad

\quad

72. 读取键盘输入

def forinput():

    input\_text = input()

    print("your input text is: ", input\_text)

>>> forinput()

>? 2

your input text is:  2

    1

    2

    3

    4

    5

    6

    7

\quad

\quad

73. enumerate()的用法

enumerate()是 Python 内置函数。enumerate()函数用于将一个可遍历的数据对象（如列表、元组或字符串）组合为一个索引序列，同时列出数据和数据下标，一般用在for循环当中。

它允许我们遍历数据并自动计数，用法如下：

for counter, value in enumerate(some\_list):

    print(counter, value)

    1

    2

不只如此，enumerate() 也接受一些可选参数，这使它更有用。

>>> my\_list = ['apple', 'banana', 'grapes', 'pear']

>>> for c, value in enumerate(my\_list, 1):

...     print(c, value)

...

1 apple

2 banana

3 grapes

4 pear

    1

    2

    3

    4

    5

    6

    7

    8

上面这个可选参数允许我们定制从哪个数字开始枚举。

此外，还可以用来创建包含索引的元组列表， 例如：

>>> my\_list = ['apple', 'banana', 'grapes', 'pear']

>>> counter\_list = list(enumerate(my\_list, 1))

>>> print(counter\_list)

[(1, 'apple'), (2, 'banana'), (3, 'grapes'), (4, 'pear')]

    1

    2

    3

    4

\quad

\quad

74. pass 语句

pass是空语句，是为了保持程序结构的完整性。pass不做任何事情，一般用做占位语句。

def forpass(n):

    if n == 1:

        pass

    else:

        print('not 1')

>>> forpass(1)

    1

    2

    3

    4

    5

    6

    7

\quad

\quad

75. 正则匹配邮箱

>>> import re

>>> email\_list= ["test01@163.com","test02@163.123", ".test03g@qq.com", "test04@gmail.com" ]

>>> for email in email\_list:

        ret = re.match("[\w]{4,20}@(.\*)\.com$", email)

        if ret:

            print("%s 是符合规定的邮件地址，匹配后结果是:%s" % (email, ret.group()))

        else:

            print("%s 不符合要求" % email)

test01@163.com 是符合规定的邮件地址，匹配后结果是:test01@163.com

test02@163.123 不符合要求

.test03g@qq.com 不符合要求

test04@gmail.com 是符合规定的邮件地址，匹配后结果是:test04@gmail.com

    1

    2

    3

    4

    5

    6

    7

    8

    9

    10

    11

    12

    13

\quad

\quad

76. 统计字符串中大写字母的数量

>>> str2 = 'werrQWSDdiWuW'

>>> counter = 0

>>> for i in str2:

        if i.isupper():

            counter += 1

>>> counter

6

    1

    2

    3

    4

    5

    6

    7

\quad

\quad

77. json 序列化时保留中文

普通序列化：

>>> import json

>>> dict1 = {'name': '萝卜', 'age': 18}

>>> dict1\_new = json.dumps(dict1)

>>> print(dict1\_new)

{"name": "\u841d\u535c", "age": 18}

    1

    2

    3

    4

    5

保留中文：

>>> import json

>>> dict1 = {'name': '萝卜', 'age': 18}

>>> dict1\_new = json.dumps(dict1, ensure\_ascii=False)

>>> print(dict1\_new)

{"name": "萝卜", "age": 18}

    1

    2

    3

    4

    5

\quad

\quad

78. 简述继承

一个类继承自另一个类，也可以说是一个孩子类/派生类/子类，继承自父类/基类/超类，同时获取所有的类成员（属性和方法）。继承使我们可以重用代码，并且还可以更方便地创建和维护代码。

Python 支持以下类型的继承：

    单继承：一个子类类继承自单个基类

    多重继承：一个子类继承自多个基类

    多级继承：一个子类继承自一个基类，而基类继承自另一个基类

    分层继承：多个子类继承自同一个基类

    混合继承：两种或两种以上继承类型的组合

\quad

\quad

79. 什么是猴子补丁（🧡🧡）

猴子补丁是指在运行时动态修改类和模块。

猴子补丁主要有以下几个用处：

    在运行时替换方法、属性等；

    在不修改第三方代码的情况下增加原来不支持的功能；

    在运行时为内存中的对象增加 patch 而不是在磁盘的源代码中增加。

详细看链接：《Python进阶系列》十七：详解Python中的猴子补丁——允许在运行时更改对象的行为

\quad

\quad

80. help() 函数和 dir() 函数

help()函数返回帮助文档和参数说明

>>> help(dict)

Help on class dict in module builtins:

class dict(object)

 |  dict() -> new empty dictionary

 |  dict(mapping) -> new dictionary initialized from a mapping object's

 |      (key, value) pairs

 |  dict(iterable) -> new dictionary initialized as if via:

 |      d = {}

 |      for k, v in iterable:

 |          d[k] = v

 |  dict(\*\*kwargs) -> new dictionary initialized with the name=value pairs

 |      in the keyword argument list.  For example:  dict(one=1, two=2)

 |

 |  Methods defined here:

 ......

    1

    2

    3

    4

    5

    6

    7

    8

    9

    10

    11

    12

    13

    14

    15

dir()函数返回对象中的所有成员 (任何类型)

>>> dir(dict)

['\_\_class\_\_',

 '\_\_contains\_\_',

 '\_\_delattr\_\_',

 '\_\_delitem\_\_',

 '\_\_dir\_\_',

 '\_\_doc\_\_',

 '\_\_eq\_\_',

 '\_\_format\_\_',

 '\_\_ge\_\_',

 '\_\_getattribute\_\_',

 ......

    1

    2

    3

    4

    5

    6

    7

    8

    9

    10

    11

    12

\quad

\quad

81. 解释 Python 中的//，％和\*\*运算符

    // 运算符执行地板除法，返回结果的整数部分 (向下取整)。

    % 是取模符号，返回除法后的余数。

    \*\* 符号表示取幂。 a\*\*b 返回 a 的b 次方。

\quad

\quad

82. 主动抛出异常

使用raise：

def test\_raise(n):

    if not isinstance(n, int):

        raise Exception("not a int type")

    else:

        print("good")

>>> test\_raise(8.9)

Traceback (most recent call last):

  File "D:\Anaconda3\lib\site-packages\IPython\core\interactiveshell.py", line 3331, in run\_code

    exec(code\_obj, self.user\_global\_ns, self.user\_ns)

  File "<ipython-input-106-27962090d274>", line 1, in <module>

    test\_raise(8.9)

  File "<ipython-input-105-8095f403c7e2>", line 3, in test\_raise

    raise Exception("not a int type")

Exception: not a int type

    1

    2

    3

    4

    5

    6

    7

    8

    9

    10

    11

    12

    13

    14

    15

\quad

\quad

83. tuple 和 list 转换

>>> tuple1 = (1, 2, 3, 4)

>>> list1 = list(tuple1)

>>> print(list1)

>>> tuple2 = tuple(list1)

>>> print(tuple2)

    1

    2

    3

    4

    5

\quad

\quad

84. 简述断言

Python 的断言就是检测一个条件，如果条件为真，它什么都不做；反之它触发一个带可选错误信息的 AssertionError。

def testassert(n):

    assert n == 2, "n is not 2"

    print("n is 2")

>>> testassert(2)

n is 2

>>> testassert(3)

Traceback (most recent call last):

  File "D:\Anaconda3\lib\site-packages\IPython\core\interactiveshell.py", line 3331, in run\_code

    exec(code\_obj, self.user\_global\_ns, self.user\_ns)

  File "<ipython-input-109-9b7ba30779d0>", line 1, in <module>

    testassert(3)

  File "<ipython-input-107-ecbc45f8f4b4>", line 2, in testassert

    assert n == 2, "n is not 2"

AssertionError: n is not 2

    1

    2

    3

    4

    5

    6

    7

    8

    9

    10

    11

    12

    13

    14

    15

\quad

\quad

85. 什么是异步非阻塞

同步异步指的是调用者与被调用者之间的关系。

    所谓同步，就是在发出一个功能调用时，在没有得到结果之前，该调用就不会返回，一旦调用返回，就得到了返回值；

    异步的概念和同步相对，调用在发出之后，这个调用就直接返回了，所以没有返回结果。当该异步功能完成后，被调用者可以通过状态、通知或回调来通知调用者。

阻塞非阻塞是线程或进程之间的关系。

    阻塞调用是指调用结果返回之前，当前线程会被挂起（如遇到io操作）。调用线程只有在得到结果之后才会返回。函数只有在得到结果之后才会将阻塞的线程激活

    非阻塞和阻塞的概念相对应，非阻塞调用指在不能立刻得到结果之前也会立刻返回，同时该函数不会阻塞当前线程

\quad

\quad

86. 什么是负索引

Python 中的序列是有索引的，它由正数和负数组成。正的数字使用’0’作为第一个索引，‘1’作为第二个索引，以此类推。负数的索引从’-1’开始，表示序列中的最后一个索引，’ - 2’作为倒数第二个索引，依次类推。

\quad

\quad

87. 退出 Python 后，内存是否全部释放

不是的，那些具有对象循环引用或者全局命名空间引用的变量，在 Python 退出时往往不会被释放，

另外不会释放 C 库保留的部分内容。

\quad

\quad

88. Flask 和 Django 的异同

    Flask 是 “microframework”，主要用来编写小型应用程序，不过随着 Python 的普及，很多大型程序也在使用 Flask。同时，在 Flask 中，我们必须使用外部库。

    Django 适用于大型应用程序。它提供了灵活性，以及完整的程序框架和快速的项目生成方法。可以选择不同的数据库，URL结构，模板样式等。

\quad

\quad

89. 创建删除操作系统上的文件

>>> f = open('test.txt', 'w')

>>> f.close()

>>> os.listdir()

['.idea',

 'test.txt',

 '\_\_pycache\_\_']

>>> os.remove('test.txt')

>>> os.listdir()

['.idea',

 '\_\_pycache\_\_']

    1

    2

    3

    4

    5

    6

    7

    8

    9

    10

    11

\quad

\quad

90. 简述 logging 模块

logging 模块是 Python 内置的标准模块，主要用于输出运行日志，可以设置输出日志的等级、日志保存路径、日志文件回滚等；相比 print，具备如下优点：

    可以通过设置不同的日志等级，在 release 版本中只输出重要信息，而不必显示大量的调试信息

    print 将所有信息都输出到标准输出中，严重影响开发者从标准输出中查看其它数据；logging 则可以由开发者决定将信息输出到什么地方，以及怎么输出。

简单配置：

>>> import logging

>>> logging.debug("debug log")

>>> logging.info("info log")

>>> logging.warning("warning log")

>>> logging.error("error log")

>>> logging.critical("critica log")

    1

    2

    3

    4

    5

    6

默认情况下，只显示了大于等于WARNING级别的日志。logging.basicConfig()函数调整日志级别、输出格式等。

详细看链接：《Python进阶系列》九：别再用print来打印了，试试logging模块

\quad

\quad

91. 统计字符串中字符出现次数

>>> from collections import Counter

>>> str1 = "nihsasehndciswemeotpxc"

>>> print(Counter(str1))

Counter({'s': 3, 'e': 3, 'n': 2, 'i': 2, 'h': 2, 'c': 2, 'a': 1, 'd': 1, 'w': 1, 'm': 1, 'o': 1, 't': 1, 'p': 1, 'x': 1})

    1

    2

    3

    4

\quad

\quad

92. 正则 re.complie 的作用

re.compile 是将正则表达式编译成一个对象，加快速度，并重复使用。

\quad

\quad

93. try except else finally 的意义

    try..except..else 没有捕获到异常，执行 else 语句

    try..except..finally 不管是否捕获到异常，都执行 finally 语句

\quad

\quad

94. 反转列表

第一种方法：使用切片

>>> list1 = list(range(10))

>>> list1[::-1]

[9, 8, 7, 6, 5, 4, 3, 2, 1, 0]

    1

    2

    3

第二种方法：使用reverse()

>>> list1 = list(range(10))

>>> list1.reverse()

>>> list1

[9, 8, 7, 6, 5, 4, 3, 2, 1, 0]

    1

    2

    3

    4

需要注意的是：这两种方法都可以反转列表，但内置函数 reverse() 会更改原始列表，而切片方法会创建一个新列表。内置函数 reverse() 比列表切片方法更快！

\quad

\quad

95. 字符串中数字替换

使用 re 正则替换：

>>> import re

>>> str1 = '我是周萝卜，今年18岁'

>>> re.sub(r"\d+", "20", str1)

'我是周萝卜，今年20岁'

    1

    2

    3

    4

\quad

\quad

96. 读取大文件（🧡🧡）

现要处理一个大小为10G的文件，但是内存只有4G，如果在只修改get\_lines 函数而其他代码保持不

变的情况下，应该如何实现？需要考虑的问题都有那些？

def get\_lines():

    with open('file.txt','rb') as f:

        return f.readlines()

if name == ' main ':

    for e in get\_lines():

        process(e) # 处理每一行数据

    1

    2

    3

    4

    5

    6

    7

方法一：readlines()函数在文件过大时并不适用，应添加参数，限制读取的字节数，并使用生成器。

def get\_lines():

    l = []

    with open('file.txt','rb') as f:

        data = f.readlines(60000)

    l.append(data)

    yield l

    1

    2

    3

    4

    5

    6

方法二：使用mmap

from mmap import mmap

def get\_lines(fp):

    with open(fp, "r+") as f:

        m = mmap(f.fileno(), 0)

        tmp = 0

        for i, char in enumerate(m):

            if char==b"\n":

                yield m[tmp:i + 1].decode()

                tmp = i + 1

if name ==" main ":

    for i in get\_lines("fp\_some\_huge\_file"):

        print(i)

    1

    2

    3

    4

    5

    6

    7

    8

    9

    10

    11

    12

    13

    14

详细看链接：《Python进阶系列》二十八：mmap模块（处理大文本）

\quad

\quad

97. 输入日期， 判断这一天是这一年的第几天

import datetime

def dayofyear():

    year = input("请输入年份: ")

    month = input("请输入月份: ")

    day = input("请输入天: ")

    date1 = datetime.date(year=int(year), month=int(month), day=int(day))

    date2 = datetime.date(year=int(year), month=1, day=1)

    return (date1 - date2).days + 1

>>> dayofyear()

请输入年份: >? 2022

请输入月份: >? 5

请输入天: >? 23

143

    1

    2

    3

    4

    5

    6

    7

    8

    9

    10

    11

    12

    13

    14

    15

\quad

\quad

98. 排序

根据字典按值排序

现有字典d= {'a':24,'g':52,'i':12,'k':33}请按value值进行排序?

>>> d = {'a': 24, 'g': 52, 'i': 12, 'k': 33}

>>> d.items()

dict\_items([('a', 24), ('g', 52), ('i', 12), ('k', 33)])

>>> sorted(d.items(), key=lambda x: x[1])

[('i', 12), ('a', 24), ('k', 33), ('g', 52)]

    1

    2

    3

    4

    5

    6

    x[0]代表用key进行排序；x[1]代表用value进行排序。

请按alist中元素的age由大到小排序

>>> alist = [{'name': 'a', 'age': 20}, {'name': 'b', 'age': 30}, {'name': 'c', 'age': 25}]

>>> sorted(alist, key=lambda x:x['age'], reverse=True)

[{'name': 'b', 'age': 30}, {'name': 'c', 'age': 25}, {'name': 'a', 'age': 20}]

    1

    2

    3

列表内，字典按照 value 大小排序

>>> list1 = [{'name': 'guanyu', 'age':29},

...         {'name': 'zhangfei', 'age': 28},

...         {'name': 'liubei', 'age':31}]

>>> sorted(list1, key=lambda x:x['age'])

[{'name': 'zhangfei', 'age': 28}, {'name': 'guanyu', 'age': 29}, {'name': 'liubei', 'age': 31}]

>>> sorted(list1, key=lambda x:x['name'])

[{'name': 'guanyu', 'age': 29}, {'name': 'liubei', 'age': 31}, {'name': 'zhangfei', 'age': 28}]

    1

    2

    3

    4

    5

    6

    7

\quad

\quad

99. 将字符串处理成字典

>>> k = "k:1|k1:2|k2:3|k3:4"

>>> dict1 = {}

>>> for items in k.split("|"):

...    key, value = items.split(":")

...    dict1[key] = value

    1

    2

    3

    4

    5

字典推导式：

>>> {k:v for items in k.split("|") for k, v in (items.split(":"), )}

{'k': '1', 'k1': '2', 'k2': '3', 'k3': '4'}

    1

    2

\quad

\quad

100. 下面代码的输出结果将是什么？

list = ['a','b','c','d','e']

print(list[10:])

    1

    2

代码将输出[]，不会产生IndexError错误，就像所期望的那样，尝试用超出成员的个数的index来获取某个列表的成员。例如，尝试获取list[10]和之后的成员，会导致IndexError。然而，尝试获取列表的切片，开始的index超过了成员个数不会产生IndexError，而是仅仅返回一个空列表。这成为特别让人恶心的疑难杂症，因为运行的时候没有错误产生，导致Bug很难被追踪到。

文章知识点与官方知识档案匹配，可进一步学习相关知识

1、列表、元组、字典的区别？

这个问题应该经常被问到，我们在这里详细做个解释。

列表(List)

Python的列表实际上是一个动态数组，存储在一个连续的内存区块中，随机存取的复杂度是O(1)，插入和删除元素时会造成内存块的移动，时间复杂度是O(n)。

同时它是一个可变对象，当我们对列表进行赋值时获取到的只是它的内存地址，如果需要将列表里的内容全部复制给另一个变量，需要用到copy（浅拷贝）和deepcopy（深度拷贝）。

元组(Tuple)

Python的元组是一个不可变的数据结构，它本质上也是一个数组。因为是不可变对象，所以Tuple的长度在创建时就是恒定的，所以我们也无法对它进行添加和删除元素的操作。不过在Tuple内所包含的对象是可变的。

当我们把包含元组的变量赋值给另一个变量时，实际上是在内存中重新申请了一块内存空间用于新建了一个元组。

字典（Dict)

Python的字典是一个哈希表，根据键值对(Key,Value)直接访问的数据结构。关于哈希函数在这里不多做解析，大家可以自行了解。

如果字典在产生哈希冲突时，也就是Key遇到重复的时候。Python会通过开放定址法来计算下一个候选位置，反复测试最终保证生成的哈希值不会产生冲突。

字典跟列表一样，也是可变对象，复制内容同样需要用到copy（浅拷贝）和deepcopy（深度拷贝）。

2、如何将一个字符串或者数字倒序输出

字符串倒序，我们可以利用Python的切片对字符串进行倒序，参考如下代码：

str1 = "长风几万里，吹度玉门关"

print(str1[::-1])

切片的参数格式: [start\_index: stop\_index: step]

如果我们不填写切片起止位置参数，那么默认是取字符串全部内容，当step参数(步长)为负数时，字符串会自动从右往左取值，-1就是依次取值，那么就自然是倒序了。

我们同样可以通过切片的原理对数字进行倒序输出。

# 正整数的情况

number = 10002

new\_number = int(str(number)[::-1])

# 负整数的情况

number = -10002

new\_number = int('-{0}'.format(str(abs(number))[::-1]))

3、谈一下Python的内存管理机制

Python内建了垃圾回收处理机制，引用计数是这个机制的一部分。

在Pyhton源码中，实际上是用 Py\_INCREF(op) 和 Py\_DECREF(op) 这两个宏来增加和减少引用计数。

当一个对象被创建、被赋值、被参数传递，函数返回之前的时候，它的引用计数值(ob\_refcnt)都可能会被加1(INC)，一直进行累加。

当对象变量失去作用域的时候，引用计数的值会减1(DEC)。

当一个对象的引用计数减少到0之后(ob\_refcnt为0)，Py\_DECREF(op)会调用该对象 "析构函数"(del) 将其从内存中释放。

4、什么是Session、Cookie、Token？

Session是一个概念，信息存储在服务端。

Cookie是对Session的一种实现，并信息存储在客户端（浏览器）。

因为HTTP协议无状态的特性，以至于我们需要在浏览器和服务端之间建立一个用于识别用户身份和详细信息的凭证，这个凭证可以是Cookie、Token任意一种。

当用户登陆成功时，我们可以将其的身份凭证在服务端生成一个Session信息，保存在文件，数据库或者内存里，通常Session会有一个Session id。

因为访问服务端Session 信息需要用到Session id，所以通常情况下，我们将Session id存在Cookie里。

Cookie其实信息在用户登陆产生Session信息之后再将Session id或者别的附加信息返回给客户端，由客户端存储在本地文件里。

当浏览器向服务端发起请求时会带着Cookie里的Session id访问服务端，服务端根据Session id找到存储好的Session信息，如果信息能找到并且内容无误，即视为访问有效。

Cookie除了存储Session id外也可以存储其他非敏感的信息（例如用户昵称，头像等），提供给浏览器直接使用，而不用每次都从服务端去拿。

关于Token，实际上用在基于RESTAPI相关的服务里比较多。

它的认证机制是当用户登陆后服务端算出一个Token信息存储在服务端并返回给客户端，内容通常包含 用户id，当前时间戳，签名和其他信息。

Token在客户端一般存放于localStorage、cookie、或sessionStorage中。在服务器一般存于数据库中。

当客户端再次请求服务端时，会从本地拿到Token信息，并放在headers中，服务端收到请求，会自动去headers里拿到Token进行解析以用于识别用户身份。

5、GET和POST的区别和作用？

GET和POST在本质上没有区别，HTTP协议并没有规定GET和POST传输数据长度的限制。

唯一的限制可能存在于服务端的服务程序和浏览器。

通常在 Nginx或者各种WebServer服务程序里会有定义GET和POST传输最大长度的限制。

而GET提交的数据长度限制通常取决于浏览器，每种浏览器的限制不一样。

在HTTP协议中，使用什么样的Method和数据如何传输其实没有相互的关系，在绝大多数的WebServer里。GET和POST提交的数据其实都在BODY区域内，我们既可以通过GET来传输文件，也可以通过POST来传输文件。

之所以通常定义GET用来获取数据，POST用来提交数据是因为GET请求是幂等的，POST请求不是。

幂等性是指一次和多次请求某一个资源应该具有同样的副作用。简单来说意味着对同一URL的多个请求应该返回同样的结果。

基于幂等的原则，我们用GET进行数据的添加修改删除时会有副作用，因为在网络情况不好的时候GET会自动尝试重试，增加了重复操作数据的风险。而用它获取数据就不会存在这样的风险，因为我们哪怕对一个资源请求100万次，它还是不会改变。

这个问题看面试官的技术水准，如果面试官愿意跟你聊得比较深入，那你可以这么回答。如果面试官自己心里也是一些标准答案的话，那建议只回答GET用于获取数据，POST用于提交数据。

另外DELETE方法其实也是幂等的，哪怕你删除100万次，数据其实也只会被删除一次。

6、什么是Python的装饰器

Python的装饰器本质上也是一个函数，它的用处是可以让其他函数不做太多代码变动的情况下增加装饰器函数相关的功能。

先来看看装饰器的实现和应用。

以上我们实现了一个极简的Python装饰器，在fun1函数运行开始和结束的时候打印日志，通过这个例子我们可以发现，实际上Python的装饰器帮我们简化了调用代码，如果不用装饰器，那以上代码可以这样写。

以上代码会显式的通过log函数去调用func1函数，不便于理解，也破坏了代码调用的顺序，不够优雅。如果我需要给fun1前面加上10多个装饰器的话，那得把这个调用关系写上十多次，在阅读理解上来看基本就是灾难。

Python的装饰器通常用于但不限于以下场合。

    日志打印 性能测试 数据库事务处理 权限校验 缓存处理

总结起来Pyhton装饰器的用处是为了让我们更好的复用代码，在实现相同功能代码的情况下，采用更符合人类理解的代码结构。

其实Python的装饰器就是JAVA语言中的AOP编程（面向切片）。

7、Python的全局锁、进程、线程、协程的概念

Python的全局锁是为了保证线程安全而做的妥协，简单的说就是不管是几个核的CPU也只能在同时间运行一个线程。IO密集型运算，多线程还勉强能用，如果是CPU密集型的，那Python的多线程几乎没什么卵用。

如果要利用CPU多核的优势可以用到进程，在Pyhton中的multiprocessing模块来处理这个问题，每个计算任务都会单独起一个Python进程，让任务运行在独立的解释器环境中，提高运算效率。

而协程的概念是在于线程和进程在CPU用户态和内核态之间进行切换时会产生大量的性能开销。

在这种情况下，协程或者叫微线程的概念应运而生。

在Python3.5中之后用async/await关键字来处理协程。

它的优点是：

没有线程之间切换切换的开销。

没有线程锁的机制，因为只有一个线程，不存在共享变量加锁的情况，进一步提升了效率。

8、列表推导和字典推导

# 列表推导

list1 = [i for i in range(5)]

# 字典推导

dict1 = {k:v for k,v in dict.items()}

# 集合推导

set1 = {x\*2 for x in [1,2]}

列表推导、字典推导、集合推导都是Python的特性，字典推导在Python2.7之后才有。

这三种推导方式的作用都是可以通过已有的数据序列构建另一个新的数据序列的结构体，从代码可读性和收敛的角度来讲是很有必要的，避免了繁复琐碎的代码写法。

9、生成器

# 构造生成器

x = (i for i in range(10))

# 输出x

<generator object <genexpr> at 0x10e9d40c0>

我们可以通过以上代码看到生成器的格式和列表推导格式很像，唯一区别是把方括号换成了圆括号，通过生成器生成出来的对象不再是一个列表，而是一个生成器对象。

生成器generator的作用是当我们要产生一个很长的序列，不一定马上用到它，那么我们就没必要再生成它的时候就开辟很大一块内存空间，而是先声明一个对象，告诉它我需要一块很大的内存空间，但是等到我用的时候再挨个去产生就行了，这个在其他语言里面有个词叫做“惰性求值”，大家望文思意即可。

我们可以直接用 for循环去遍历生成器对象，依次取出值。

1 heappush(heap,item)建立大、小根堆

heapq.heappush()是往堆中添加新值，此时自动建立了小根堆，代码如下

import heapq

a = []   #创建一个空堆

heapq.heappush(a,18)

heapq.heappush(a,1)

heapq.heappush(a,20)

heapq.heappush(a,10)

heapq.heappush(a,5)

heapq.heappush(a,200)

print(a)

    1

    2

    3

    4

    5

    6

    7

    8

    9

输出

[1, 5, 20, 18, 10, 200]

    1

但heapq里面没有直接提供建立大根堆的方法，可以采取如下方法：每次push时给元素加一个负号（即取相反数），此时最小值变最大值，反之亦然，那么实际上的最大值就可以处于堆顶了，返回时再取负即可。

a = []

for i in [1, 5, 20, 18, 10, 200]:

    heapq.heappush(a,-i)

print(list(map(lambda x:-x,a)))

    1

    2

    3

    4

输出

[200, 18, 20, 1, 10, 5]

    1

2 heapify(heap)建立大、小根堆

heapq.heapfy()是以线性时间讲一个列表转化为小根堆

a = [1, 5, 20, 18, 10, 200]

heapq.heapify(a)

print(a)

    1

    2

    3

输出

[1, 5, 20, 18, 10, 200]

    1

用第一节中同样的方法建立大根堆：

a = [1, 5, 20, 18, 10, 200]

a = list(map(lambda x:-x,a))

heapq.heapify(a)

print([-x for x in a])

    1

    2

    3

    4

输出

200, 18, 20, 5, 10, 1]

    1

与第一节所得大根堆相比，虽然二叉树的第三层元素顺序不同，但都符合大根堆的定义

3 heappop(heap)

heapq.heappop()是从堆中弹出并返回最小的值

3.1 利用heappop进行堆排序

堆排序当然也要利用到heappush或者heapify，可参考排序算法总结中的堆排序，这里只贴代码

import heapq

def heap\_sort(arr):

    if not arr:

        return []

    h = []  #建立空堆

    for i in arr:

        heapq.heappush(h,i) #heappush自动建立小根堆

    return [heapq.heappop(h) for i in range(len(h))]  #heappop每次删除并返回列表中最小的值

    1

    2

    3

    4

    5

    6

    7

    8

若是从大到小排列，有两种方法：

1）先建立小根堆，然后每次heappop()，此时得到从小大的排列，再reverse

2）利用相反数建立大根堆，然后heappop(-元素)。即push(-元素)，pop(-元素)

3.2 普通list的heapop()

普通list（即并没有进行heapify等操作的list），对他进行heappop操作并不会弹出list中最小的值，而是弹出第一个值

>>> a=[3,6,1]

>>> heapify(a)                  #将a变成堆之后，可以对其操作

>>> heappop(a)

1

>>> b=[4,2,5]                   #b不是堆，如果对其进行操作，显示结果如下

>>> heappop(b)                  #按照顺序，删除第一个数值并返回,不会从中挑选出最小的

4

>>> heapify(b)                  #变成堆之后，再操作

>>> heappop(b)

2

    1

    2

    3

    4

    5

    6

    7

    8

    9

    10

4 heappushpop(heap,item)

heapq.heappushpop()是heappush和haeppop的结合，同时完成两者的功能，先进行heappush()，再进行heappop()

>>>h =  [1, 2, 9, 5]

>>> heappop(h)

1

>>> heappushpop(h,4)            #增加4同时删除最小值2并返回该最小值，与下列操作等同：

2

>>> h

[4, 5, 9]

    1

    2

    3

    4

    5

    6

    7

5 heapreplace(heap.item)弹出并返回 heap 中最小的一项，同时推入新的 item

heapq.heapreplace()与heapq.heappushpop()相反，先进行heappop()，再进行heappush()

堆的大小不变。 如果堆为空则引发 IndexError。这个单步骤操作比依次执行heappop() + heappush() 更高效，并且在使用固定大小的堆时更为适宜。 pop/push 组合总是会从堆中返回一个元素并将其替换为 item。返回的值可能会比添加的 item 更大。 如果不希望如此，可考虑改用 heappushpop()。 它的 push/pop 组合会返回两个值中较小的一个，将较大的值留在堆中。

>>> a=[]

>>> heapreplace(a,3)            #如果list空，则报错

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

IndexError: index out of range

>>> heappush(a,3)

>>> a

[3]

>>> heapreplace(a,2)            #先执行删除（heappop(a)->3),再执行加入（heappush(a,2))

3

>>> a

[2]

>>> heappush(a,5)

>>> heappush(a,9)

>>> heappush(a,4)

>>> a

[2, 4, 9, 5]

>>> heapreplace(a,6)            #先从堆a中找出最小值并返回，然后加入6

2

>>> a

[4, 5, 9, 6]

>>> heapreplace(a,1)            #1是后来加入的，在1加入之前，a中的最小值是4

4

>>> a

[1, 5, 9, 6]

    1

    2

    3

    4

    5

    6

    7

    8

    9

    10

    11

    12

    13

    14

    15

    16

    17

    18

    19

    20

    21

    22

    23

    24

    25

6 merge(\*iterables)

heapq.merge()合并多个堆然后输出

输入的list无序，merge后无序，若输入的list有序，merge后也有序

在这里插入图片描述

heapq.merge()的迭代性质意味着它对所有提供的序列都不会做一次性读取。这意味着可以利用它处理非常长的序列，而开销却非常小

7 nlargest(n , iterbale, key=None) 和nsmallest(n , iterbale, key=None)

获取列表中最大、最小的几个值，key的作用和sorted( )方法里面的key类似

>>>a = [0, 1, 2, 3, 4, 5, 5, 7, 8, 10, 15, 20, 25]

>>>heapq.nlargest(5,a)

[25, 20, 15, 10, 8]

>>>b = [('a',1),('b',2),('c',3),('d',4),('e',5)]

>>>heapq.nlargest(1,b,key=lambda x:x[1])

[('e', 5)]

    1

    2

    3

    4

    5

    6

    7

8 复杂度分析

8.1 各方法复杂度

1）heapq.heapify(x): O(n)

2）heapq.heappush(heap, item): O(logn)

3）heapq.heappop(heap): O(logn)

即插入或删除元素时，所有节点自动调整，保证堆的结构的复杂度为O(log n)

4）heapq.nlargest(k,iterable)和nsmallest(k,iterable)：O(n \* log(t))

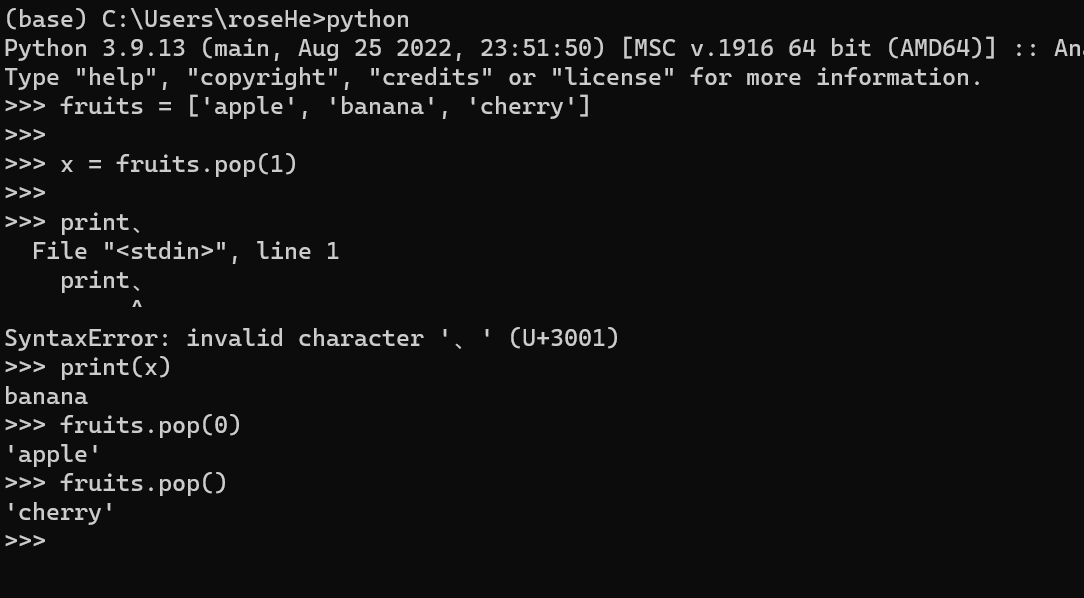
8.2 关于排序和取TopN时各方法的快慢比较

在关于排序和取Top N值时，到底使用什么方法最快，python3 cookbook给出了非常好的建议：

1）当要查找的元素个数相对比较小的时候，函数nlargest() 和 nsmallest()。

2）仅仅想查找唯一的最小或最大(N=1)的元素的话，那么使用min()和max()函数。

3）如果N的大小和集合大小接近的时候，通常先排序这个集合然后再使用切片操作会更快点 (sorted(items)[:N] 或者是 sorted(items)[-N:])。



Python sum() 函数

Python 内置函数 Python 内置函数

描述

sum() 方法对序列进行求和计算。

语法

以下是 sum() 方法的语法:

sum(iterable[, start])

参数

    iterable -- 可迭代对象，如：列表、元组、集合。

    start -- 指定相加的参数，如果没有设置这个值，默认为0。

返回值

返回计算结果。

实例

以下展示了使用 sum 函数的实例：

>>>sum([0,1,2])

3

>>> sum((2, 3, 4), 1)        # 元组计算总和后再加 1

10

>>> sum([0,1,2,3,4], 2)      # 列表计算总和后再加 2

12

1<<i 是将1左移i位，即第i位为1，其余位为0；

例如1<<2 则0001->0100

n&(1<<i)是将左移i位的1与n进行按位与，即为保留n的第i位，其余位置零

如果n第i位为0，则n&(1<<i)的值为0

否则不为0

常用if(n&(1<<i)==0)用于判断n的第i位是否为0

/ (常规除)

如：

5 / 2 = 2.5

解释：平常除法是什么结果就是什么结果。

//（地板除）

如:

5 // 2 = 2 （5 ÷ 2 = 2.5）

5 // 3 = 1 (5 ÷ 3 = 1.6666666666666667)

解释：地板除，只去除完之后的整数部分。

% （取余数）

如：

5 % 2 = 1 （5 - 2\*2 = 1）

4 % 2 = 0 （4 - 2\*2 = 0）

7 % 3 = 1 （7 - 3\*2 = 1）

13 % 5 = 3 （13 - 5\*2 = 3）

解释： 就是一个取余数的操作，除开被除数的倍数，余下几就得几。上面红色的数字带表被除数的倍数。

get() 方法 Vs dict[key] 访问元素区别  get(key) 方法在 key（键）不在字典中时，可以返回默认值 None 或者设置的默认值。

sorted() 函数对所有可迭代的对象进行排序操作。

    sort 与 sorted 区别：

    sort 是应用在 list 上的方法，sorted 可以对所有可迭代的对象进行排序操作。

    list 的 sort 方法返回的是对已经存在的列表进行操作，无返回值，而内建函数 sorted 方法返回的是一个新的 list，而不是在原来的基础上进行的操作。

语法

sorted 语法：

sorted(iterable, cmp=None, key=None, reverse=False)

参数说明：

    iterable -- 可迭代对象。

    cmp -- 比较的函数，这个具有两个参数，参数的值都是从可迭代对象中取出，此函数必须遵守的规则为，大于则返回1，小于则返回-1，等于则返回0。

    key -- 主要是用来进行比较的元素，只有一个参数，具体的函数的参数就是取自于可迭代对象中，指定可迭代对象中的一个元素来进行排序。

    reverse -- 排序规则，reverse = True 降序 ， reverse = False 升序（默认）。

返回值

返回重新排序的列表。

实例

以下实例展示了 sorted 的使用方法：

>>>a = [5,7,6,3,4,1,2]

>>> b = sorted(a)       # 保留原列表

>>> a

[5, 7, 6, 3, 4, 1, 2]

>>> b

[1, 2, 3, 4, 5, 6, 7]

>>> L=[('b',2),('a',1),('c',3),('d',4)]

>>> sorted(L, cmp=lambda x,y:cmp(x[1],y[1]))   # 利用cmp函数

[('a', 1), ('b', 2), ('c', 3), ('d', 4)]

>>> sorted(L, key=lambda x:x[1])               # 利用key

[('a', 1), ('b', 2), ('c', 3), ('d', 4)]

Python collections模块之deque详解

    deque()

        append()

        appendleft()

        extend()

        extendleft()

        pop()

        popleft()

        count()

        insert(index,obj)

        rotate(n)

        clear()

        remove()

        maxlen

    collections模块 ==> Python标准库，数据结构常用的模块；collections包含了一些特殊的容器，针对Python内置的容器，例如list、dict、set和tuple，提供了另一种选择。

collections模块常用类型有：

    双向队列（deque）

    类似于list的容器，可以快速的在队列头部和尾部添加、删除元素

    请点击deque

    计数器（Counter）

    dict的子类，计算可hash的对象

    请点击Counter

    默认字典（defaultdict）

    dict的子类，可以调用提供默认值的函数

    请点击defaultdict

    有序字典（OrderedDict）

    dict的子类，可以记住元素的添加顺序

    可命名元组（namedtuple）

    可以创建包含名称的tuple

deque()

deque是栈和队列的一种广义实现，deque是"double-end queue"的简称；deque支持线程安全、有效内存地以近似O(1)的性能在deque的两端插入和删除元素，尽管list也支持相似的操作，但是它主要在固定长度操作上的优化，从而在pop(0)和insert(0,v)（会改变数据的位置和大小）上有O(n)的时间复杂度。

常用方法：

append()

从右端添加元素（与list同）

st = "abcd"

list1 = [0, 1, 2, 3]

dst = deque(st)

dlist1 = deque(list1)

dst.append(4)

dlist1.append("k")

print(dst)

print(dlist1)

#结果：

#deque(['a', 'b', 'c', 'd', 4])

#deque([0, 1, 2, 3, 'k'])

    1

    2

    3

    4

    5

    6

    7

    8

    9

    10

    11

appendleft()

从左端添加元素

st = "abcd"

list1 = [0, 1, 2, 3]

dst = deque(st)

dlist1 = deque(list1)

dst.appendleft(4)

dlist1.appendleft("k")

print(dst)

print(dlist1)

#结果：

#deque([4, 'a', 'b', 'c', 'd'])

#deque(['k', 0, 1, 2, 3])

    1

    2

    3

    4

    5

    6

    7

    8

    9

    10

    11

extend()

从右端逐个添加可迭代对象（与list同）

Python中的可迭代对象有：列表、元组、字典、字符串

from collections import deque

ex = (1, "h", 3)

st = "abcd"

list1 = [0, 1, 2, 3]

dst = deque(st)

dlist1 = deque(list1)

dst.extend(ex)

dlist1.extend(ex)

print(dst)

print(dlist1)

#结果：

#deque(['a', 'b', 'c', 'd', 1, 'h', 3])

#deque([0, 1, 2, 3, 1, 'h', 3])

    1

    2

    3

    4

    5

    6

    7

    8

    9

    10

    11

    12

    13

extendleft()

从左端逐个添加可迭代对象

Python中的可迭代对象有：列表、元组、字典、字符串

from collections import deque

ex = [("a", 1), 3]

st = "abcd"

list1 = [0, 1, 2, 3]

dst = deque(st)

dlist1 = deque(list1)

dst.extend(ex)

dlist1.extend(ex)

print(dst)

print(dlist1)

#结果：

#deque(['a', 'b', 'c', 'd', ('a', 1), 3])

#deque([0, 1, 2, 3, ('a', 1), 3])

    1

    2

    3

    4

    5

    6

    7

    8

    9

    10

    11

    12

    13

    14

pop()

移除列表中的一个元素（默认最右端的一个元素），并且返回该元素的值（与list同），如果没有元素，将会报出IndexError

from collections import deque

st = "abcd"

list1 = [0, 1, 2, 3]

dst = deque(st)

dlist1 = deque(list1)

p = dst.pop()

p1 = dlist1.pop()

print(p)

print(p1)

print(dst)

print(dlist1)

#结果:

#d

#3

#deque(['a', 'b', 'c'])

#deque([0, 1, 2])

    1

    2

    3

    4

    5

    6

    7

    8

    9

    10

    11

    12

    13

    14

    15

    16

popleft()

移除列表中的一个元素（默认最左端的一个元素），并且返回该元素的值，如果没有元素，将会报出IndexError

from collections import deque

st = "abcd"

list1 = [0, 1, 2, 3]

dst = deque(st)

dlist1 = deque(list1)

p = dst.popleft()

p1 = dlist1.popleft()

print(p)

print(p1)

print(dst)

print(dlist1)

#结果:

#a

#0

#deque(['b', 'c', 'd'])

#deque([1, 2, 3])

    1

    2

    3

    4

    5

    6

    7

    8

    9

    10

    11

    12

    13

    14

    15

    16

count()

统计队列中的元素个数（与list同）

from collections import deque

st = "abbcd"

dst = deque(st)

p = dst.count("b")

print(dst)

print(p)

#结果:

#deque(['a', 'b', 'b', 'c', 'd'])

#2

    1

    2

    3

    4

    5

    6

    7

    8

    9

    10

insert(index,obj)

在指定位置插入元素（与list同）

from collections import deque

st = "abbcd"

dst = deque(st)

dst.insert(0, "chl")

print(dst)

#结果:

#deque(['chl', 'a', 'b', 'b', 'c', 'd'])

    1

    2

    3

    4

    5

    6

    7

    8

rotate(n)

rotate(n)， 从右侧反转n步，如果n为负数，则从左侧反转。

d.rotate(1) 等于 d.appendleft(d.pop())

from collections import deque

st = "abbcd"

dst = deque(st)

dst.rotate(1)

print(dst)

#结果:

#deque(['d', 'a', 'b', 'b', 'c'])

    1

    2

    3

    4

    5

    6

    7

    8

clear()

将deque中的元素全部删除，最后长度为0

from collections import deque

st = "abbcd"

dst = deque(st)

dst.clear()

print(dst)

#结果:

#deque([])

    1

    2

    3

    4

    5

    6

    7

    8

remove()

移除第一次出现的元素，如果没有找到，报出ValueError

from collections import deque

st = "abbcd"

dst = deque(st)

dst.remove("a")

print(dst)

dst.remove("f")

#结果:

#deque(['b', 'b', 'c', 'd'])

#ValueError: deque.remove(x): x not in deque

    1

    2

    3

    4

    5

    6

    7

    8

    9

    10

maxlen

只读的属性，deque限定的最大长度，如果无，就返回None。

当限制长度的deque增加超过限制数的项时, 另一边的项会自动删除。

from collections import deque

dst = deque(maxlen=2)

dst.append(1)

dst.append(2)

print(dst)

dst.append(3)

print(dst)

print(dst.maxlen)

#结果:

#deque([1, 2], maxlen=2)

#deque([2, 3], maxlen=2)

#2

    1

    2

    3

    4

    5

    6

    7

    8

    9

    10

    11

    12

    13

    此外,deque还支持迭代、序列化、len(d)、reversed(d)、copy.copy(d)、copy.deepcopy(d)，通过in操作符进行成员测试和下标索引。

表示正负无穷：

运行：

# 正无穷

print(float("inf"))

print(float("inf")+1)

# 负无穷

print(float("-inf"))

print(float("-inf")+1)

    1

    2

    3

    4

    5

    6

    7

输出：

inf

inf

-inf

-inf

    1

    2

    3

    4

    5

if float(1/3)>float("inf"):

    print(0)

else:

    print(1)

if float(1/3)>float("-inf"):

    print(0)

else:

    print(1)

    1

    2

    3

pop() 函数用于移除列表中的一个元素（默认最后一个元素），并且返回该元素的值。

语法

pop()方法语法：

list.pop([index=-1])

参数

    obj -- 可选参数，要移除列表元素的索引值，不能超过列表总长度，默认为 index=-1，删除最后一个列表值。

返回值

该方法返回从列表中移除的元素对象。

实例

以下实例展示了 pop()函数的使用方法：

实例

#!/usr/bin/python3

#coding=utf-8

list1 = ['Google', 'Runoob', 'Taobao']

list\_pop=list1.pop(1)

print "删除的项为 :", list\_pop

print "列表现在为 : ", list1

reverse()：

是python中列表的一个内置方法（也就是说，在字典，字符串或者元组中，是没有这个内置方法的），用于列表中数据的反转；

exp：

lista = [1, 2, 3, 4]

lista.reverse()

print(lista)

    1

    2

    3

打印结果：

    [4, 3, 2, 1]

其实，lista.reverse() 这一步操作的返回值是一个None，其作用的结果，需要通过打印被作用的列表才可以查看出具体的效果。

reversed()：

而reversed()是python自带的一个方法，准确说，应该是一个类；

关于reversed()官方解释：

    reversed(sequence) -> reverse iterator over values of the sequence

    Return a reverse iterator

translate it :

reverse（sequence） - >反转迭代器的序列值

返回反向迭代器

也就是说，在经过reversed()的作用之后，返回的是一个把序列值经过反转之后的迭代器，所以，需要通过遍历，或者List,或者next()等方法，获取作用后的值；

下面通过几个案例进行说明：

1.列表的反转：

bb = [1,3,5,7]

print(list(reversed(bb)))

    1

    2

打印结果：

    [7, 5, 3, 1]

2.元组的反转：

aa = (1, 2, 3)

print(tuple(reversed(aa)))

    1

    2

打印结果：

    (3, 2, 1)

3.字符串的反转

ss = "qwer1234"

print(''.join(reversed(ss)))

zip() 函数用于将可迭代的对象作为参数，将对象中对应的元素打包成一个个元组，然后返回由这些元组组成的列表。

如果各个迭代器的元素个数不一致，则返回列表长度与最短的对象相同，利用 \* 号操作符，可以将元组解压为列表。

>>> a =[1,2,3]

>>> b =[4,5,6]

>>> c =[4,5,6,7,8]

>>> zipped =zip(a,b)# 打包为元组的列表

[(1,4),(2,5),(3,6)]

>>>zip(a,c)# 元素个数与最短的列表一致

[(1,4),(2,5),(3,6)]

>>>zip(\*zipped)# 与 zip 相反，\*zipped 可理解为解压，返回二维矩阵式

[(1,2,3),(4,5,6)]

>>> a = [1,2,3]

>>> b = [4,5,6]

>>> c = [4,5,6,7,8]

>>> zipped = zip(a,b)     # 返回一个对象

>>> zipped

<zip object at 0x103abc288>

>>> list(zipped)  # list() 转换为列表

[(1, 4), (2, 5), (3, 6)]

>>> list(zip(a,c))              # 元素个数与最短的列表一致

[(1, 4), (2, 5), (3, 6)]

>>> a1, a2 = zip(\*zip(a,b))          # 与 zip 相反，zip(\*) 可理解为解压，返回二维矩阵式

>>> list(a1)

[1, 2, 3]

>>> list(a2)

[4, 5, 6]

>>>

map() 会根据提供的函数对指定序列做映射。

第一个参数 function 以参数序列中的每一个元素调用 function 函数，返回包含每次 function 函数返回值的新列表。

>>> def square(x) :            # 计算平方数

...     return x \*\* 2

...

>>> map(square, [1,2,3,4,5])   # 计算列表各个元素的平方

[1, 4, 9, 16, 25]

>>> map(lambda x: x \*\* 2, [1, 2, 3, 4, 5])  # 使用 lambda 匿名函数

[1, 4, 9, 16, 25]

# 提供了两个列表，对相同位置的列表数据进行相加

>>> map(lambda x, y: x + y, [1, 3, 5, 7, 9], [2, 4, 6, 8, 10])

[3, 7, 11, 15, 19]

>>> def square(x) :         # 计算平方数

...     return x \*\* 2

...

>>> map(square, [1,2,3,4,5])    # 计算列表各个元素的平方

<map object at 0x100d3d550>     # 返回迭代器

>>> list(map(square, [1,2,3,4,5]))   # 使用 list() 转换为列表

[1, 4, 9, 16, 25]

>>> list(map(lambda x: x \*\* 2, [1, 2, 3, 4, 5]))   # 使用 lambda 匿名函数

[1, 4, 9, 16, 25]

>>>

s = "abcde"

list的[]中有三个参数，用冒号分割

list[param1:param2:param3]

    param1，相当于start\_index，可以为空，默认是0

    param2，相当于end\_index，可以为空，默认是list.size

    param3，步长，默认为1。步长为-1时，返回倒序原序列

举例说明

param1 = -1，只有一个参数，作用是通过下标访问数据，-1为倒数第一个

print(s[-1])

e

param1 = 0，param2 = -1，作用是返回从start\_index = 0到end\_index = -1的一串数据

这里的[0,list.size]为左闭右开

print(s[:-1])

abcd

param1 = 0，param2 = list.size，param3 = -1，作用是返回倒序的原list

print(s[::-1])

edcba

param1 = 1，param2 = list.size，param3 = -1，这个返回的不是从1到size的倒序，而是第0、1的倒序。

一下三种格式，效果等同。

二式为，先倒序，再访问倒数两个数据

三式为，先获取前两个数据，再倒序

print(s[1::-1])

ba

print(s[::-1][-2:])

ba

print(s[0:2][::-1])

ba

param1 = 0，param2 = list.size，

param3为步长，步长大于0时，返回序列为原顺序；步长小于0时，返回序列为倒序。

步长我说不清楚，请根据示例结果，自行理解。

print(s[::1])

abcde

print(s[::2])

ace

print(s[::-1])

edcba

print(s[::-2])

eca

描述：

函数功能为取传入的多个参数中的最小值，或者传入的可迭代对象元素中的最小值。

语法：

    min(iterable, \*[, key, default])

    min(arg1, arg2, \*args[, key])

参数介绍：

默认数值型参数，取值小者；

字符型参数，取字母表排序靠前者。

key---可做为一个函数，用来指定取最小值的方法。

default---用来指定最小值不存在时返回的默认值。

arg1---字符型参数/数值型参数，默认数值型

返回值：

返回参数的最小值

python中groupby函数主要的作用是进行数据的分组以及分组后地组内运算！

对于数据的分组和分组运算主要是指groupby函数的应用，具体函数的规则如下：

df[](指输出数据的结果属性名称).groupby([df[属性],df[属性])(指分类的属性，数据的限定定语，可以有多个).mean()(对于数据的计算方式——函数名称)

举例如下：

print(df["评分"].groupby([df["地区"],df["类型"]]).mean())

#上面语句的功能是输出表格所有数据中不同地区不同类型的评分数据平均值

一、Iterable(可迭代对象)：

1、什么叫可迭代对象？

    能够进行迭代操作的对象，可迭代对象（能够使用for循环遍历的都是可迭代对象）

2、python中哪些是可迭代对象？

    序列类型（元组、列表、字符串，range）

    字典、集合

    文件对象(open(xx))

    实现了迭代协议(\_\_iter\_\_方法)的对象

1. python 的数据类型？

    数值类型、字符串、元组、列表、字典、集合（不常用）

2.可变数据类型 和 不可变数据类型 都有哪些？

    可变数据类型：list（列表）、dict（字典）、set（集合，不常用）

    不可变数据类型：数值类型（int、float、bool）、string（字符串）、tuple（元组）

3.专业名词解释

    可变数据类型：当该数据类型对应的变量的值发生了变化时，如果它对应的内存地址不发生改变，那么这个数据类型就是 可变数据类型。

    不可变数据类型：当该数据类型对应的变量的值发生了变化时，如果它对应的内存地址发生了改变，那么这个数据类型就是 不可变数据类型。

    总结：可变数据类型更改值后，内存地址不发生改变。不可变数据类型更改值后，内存地址发生改变。

操作符 & ， | ， ^， &&， ||

1.位操作符：& ， |， ^

（1）.& —————— /\* 按bit位与\*/

    运算方式:按照所给数二进制位进行计算。既对应位进行与运算，然后将结果转换成十进制结果输出

    例如：int a=1; int b=2;求a&b的值

    a=1 既//0000 0001

    b=2 既 //0000 0010

    结果为 //0000 0000

    则 a & b 的结果为0

特点 :相同为1，相异为0。

#include<stdio.h>

 int main()

 {

     int a=5;      //0101

     int b=6;      //0110

     int c=a & b;  //0100---> 4

     printf("%d\n",c);

     return 0;

 }

（2）. | —————— /\* 按位或\*/

    运算方式:按照所给数二进制位进行计算。既用对应位进行或运算，然后将结果转换成十进制结果输出

特点：一真则真。

#include<stdio.h>

 int main()

 {

     int a=5;      //0101

     int b=6;      //0110

     int c=a | b;  //0111---> 7

     printf("%d\n",c);

     return 0;

 }

（3). ^ —————— /\* 按位异或\*/

特点:相同为0，相异为1；

#include<stdio.h>

 int main()

 {

     int a=5;      //0101

     int b=6;      //0110

     int c=a ^ b;  //0011---> 3

     printf("%d\n",c);

     return 0;

 }

2.逻辑操作符:&& ，||

（1）&& —————— /\* 逻辑与\*/

特点：两个操作数同时为真才为真。

#include<stdio.h>

 int main()

 {

     int a=0;

     int b=1;

     int c=2;

     int d=a && b&&c;  //结果为假

     /\* 此时a为假，不用再计算后面的表达式，此表达式逻辑为假\*/

     int e=b&&c;//为真

     printf("%d\n",d);

     printf("%d\n",e);

     return 0;

 }

（2) || —————— /\* 逻辑或\*/

特点：一真则真，两个操作数同时为假才为假。

#include<stdio.h>

 int main()

 {

     int a=1;

     int b=2;

     int c=a || b;   //结果为真

      /\* 此时a为真，所以不会再计算 || 后面的语句，此表达式逻辑为真\*/

     printf("%d\n",c);

     return 0;

 }

Python replace() 方法把字符串中的 old（旧字符串） 替换成 new(新字符串)，如果指定第三个参数max，则替换不超过 max 次。

语法

replace()方法语法：

str.replace(old, new[, max])

参数

    old -- 将被替换的子字符串。

    new -- 新字符串，用于替换old子字符串。

    max -- 可选字符串, 替换不超过 max 次

返回值

返回字符串中的 old（旧字符串） 替换成 new(新字符串)后生成的新字符串，如果指定第三个参数max，则替换不超过 max 次。

实例

以下实例展示了replace()函数的使用方法：

实例

#!/usr/bin/python

str = "this is string example....wow!!! this is really string";

print str.replace("is", "was");

print str.replace("is", "was", 3);

以上实例输出结果如下：

thwas was string example....wow!!! thwas was really string

thwas was string example....wow!!! thwas is really string

关于 string 的 replace 方法，需要注意 replace 不会改变原 string 的内容。

实例：

temp\_str = 'this is a test'

print(temp\_str.replace('is','IS'))

print(temp\_str)

结果为：

thIS IS a test

this is a test

#function1

for key in dict:

    print (key)

---------------结果---------------------

小明

小兰

小红

#function2

for key in dict.keys():

    print (key)

---------------结果---------------------

小明

小兰

小红

#function1

for value in dict.values():

    print (value)

------------------------

129

148

89

#function2

for value in dict.itervalues():

    print (value)

for key,value in dict.items():

    print ('key: ',key,'value: ',value)

------------------------

key:  小明 value:  129

key:  小兰 value:  148

key:  小红 value:  89

#function2

for (key,value) in dict.items():

    print ('key: ',key,'value: ',value)

------------------------

key:  小明 value:  129

key:  小兰 value:  148

key:  小红 value:  89

collections模块 ==> Python标准库，数据结构常用的模块；collections包含了一些特殊的容器，针对Python内置的容器，例如list、dict、set和tuple，提供了另一种选择。

collections模块常用类型有：

    计数器（Counter）

    dict的子类，计算可hash的对象

    请点击Counter

    双向队列（deque）

    类似于list的容器，可以快速的在队列头部和尾部添加、删除元素

    请点击deque

    默认字典（defaultdict）

    dict的子类，可以调用提供默认值的函数

    请点击defaultdict

    有序字典（OrderedDict）

    dict的子类，可以记住元素的添加顺序

    可命名元组（namedtuple）

    可以创建包含名称的tuple

Python count() 方法用于统计字符串里某个字符或子字符串出现的次数。可选参数为在字符串搜索的开始与结束位置。

认识defaultdict：

当使用普通的字典时，用法一般是dict={},添加元素的只需要dict[element] =value即，调用的时候也是如此，dict[element] = xxx,但前提是element字典里，如果不在字典里就会报错,

这时defaultdict就能排上用场了，defaultdict的作用是在于，当字典里的key不存在但被查找时，返回的不是keyError而是一个默认值

Python 代码使用 collections.defaultdict() 类，可指定所有 key 对应的默认 value

ASCII

美国(国家)信息交换标准(代)码，一种使用7个或8个二进制位进行编码的方案，最多可以给256个字符

(包括字母、数字、标点符号、控制字符及其他符号)分配(或指定)数值

Unicode（万国码）

如果有一种编码，将世界上所有的符号都纳入其中，无论是英文、日文、还是中文等，大家都使用这个编码表，就不会出现编码不匹配现象。每个符号对应一个唯一的编码，乱码问题就不存在了。这就是Unicode编码

简单来说：Unicode把所有语言都统一到一套编码里，这样就不会再有乱码问题了。

区别：

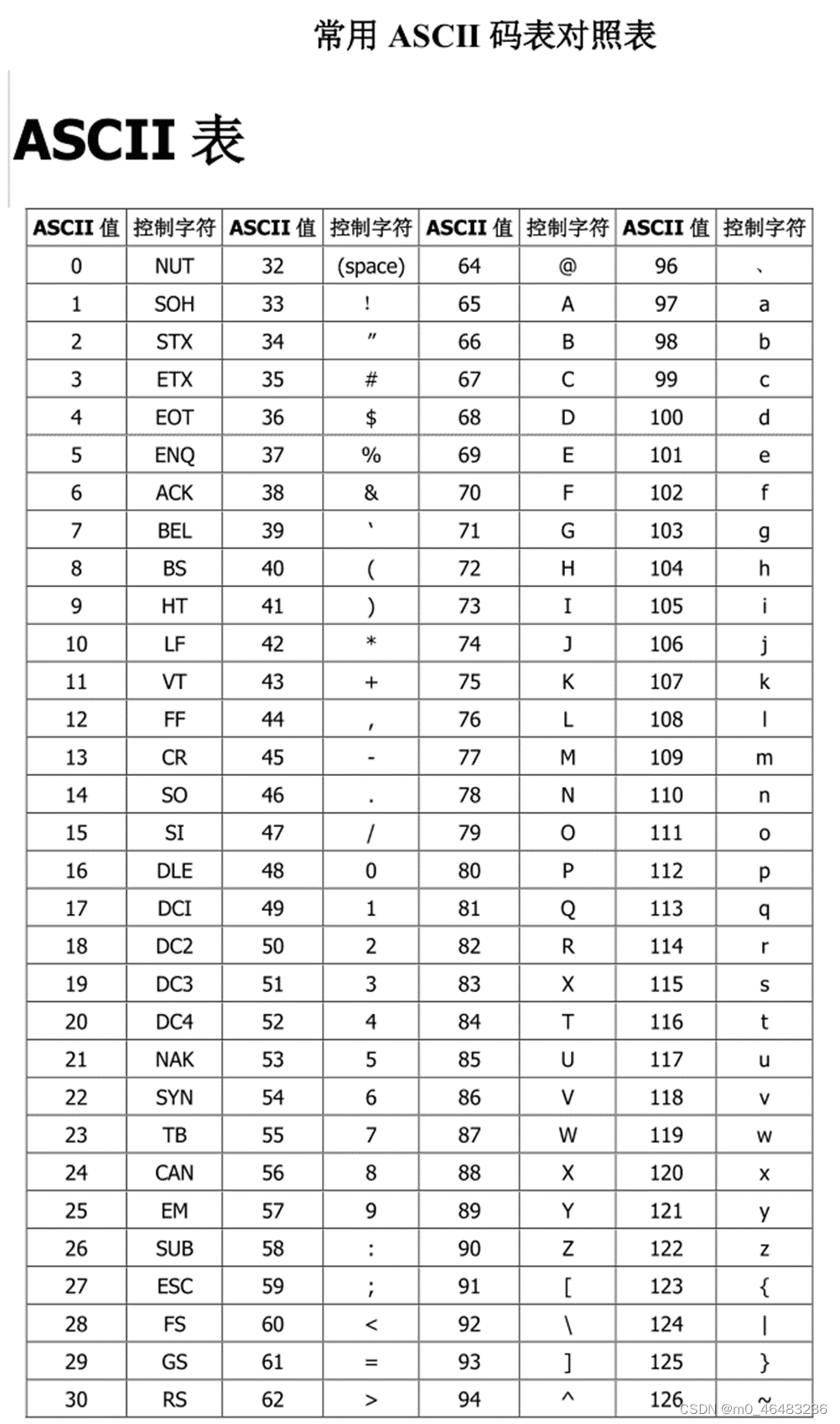
ASCII:不支持中文，1个英文占1个字节

Unicode(万国码，支持所有国家的文字显示)：支持中文，但是每个英文和中文都占2个字节

那么为什么会出现有的说ASCII有的说Unicode码的呢？

划重点

因为Python的诞生比Unicode标准发布的时间还要早，所以最早的Python只支持ASCII编码，普通的字符串’ABC’在Python内部都是ASCII编码的。，但是在Unicode标准发布以后 新版的Python，就开始逐渐基于Unicode码了

()函数介绍：

 ord() 函数是 chr() 函数（对于 8 位的 ASCII 字符串）的配对函数，它以一个字符串（Unicode 字符）作为参数，返回对应的 ASCII 数值，或者 Unicode 数值。

    >>> ord('0')

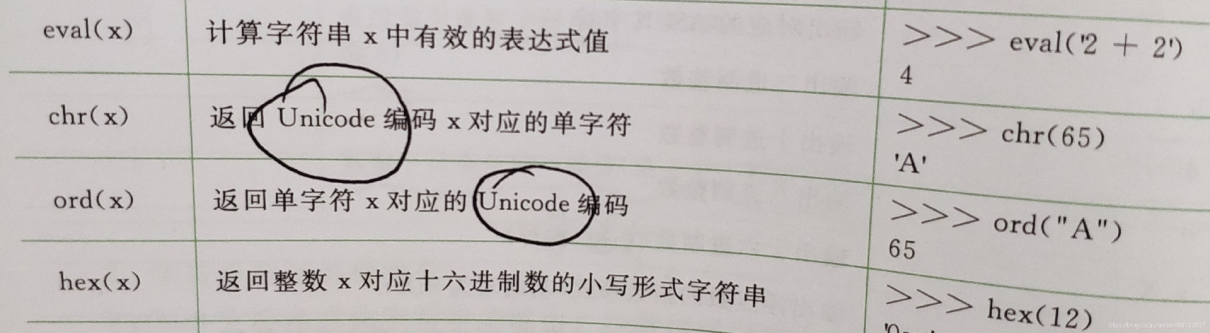
    48

    >>> ord('A')

    65

    >>> ord('a')

    97



set() 函数创建一个无序不重复元素集，可进行关系测试，删除重复数据，还可以计算交集、差集、并集等。

>>>x = set('runoob')

>>> y = set('google')

>>> x, y

(set(['b', 'r', 'u', 'o', 'n']), set(['e', 'o', 'g', 'l']))   # 重复的被删除

>>> x & y         # 交集

set(['o'])

>>> x | y         # 并集

set(['b', 'e', 'g', 'l', 'o', 'n', 'r', 'u'])

>>> x - y         # 差集

set(['r', 'b', 'u', 'n'])

>>>

>>> x = set('eleven')

>>> y = set('twelve')

>>> x,y

({'l', 'e', 'n', 'v'}, {'e', 'v', 'l', 't', 'w'})

>>> x & y  #交集

{'l', 'e', 'v'}

>>> x | y  #并集

{'e', 'v', 'n', 'l', 't', 'w'}

>>> x - y  #差集

{'n'}

>>> y -x   #差集

{'t', 'w'}

>>> x ^ y  #补集

{'t', 'n', 'w'}

>>> y ^ x  #补集

{'w', 'n', 't'}

>>>

交集 & : x&y，返回一个新的集合，包括同时在集合 x 和y中的共同元素。

并集 | : x|y，返回一个新的集合，包括集合 x 和 y 中所有元素。

差集 - : x-y，返回一个新的集合,包括在集合 x 中但不在集合 y 中的元素。

补集 ^ : x^y，返回一个新的集合，包括集合 x 和 y 的非共同元素。

大括号或 set() 函数可以用来创建集合。

set集合类需要的参数必须是迭代器类型的，如：序列、字典等，然后转换成无序不重复的元素集。由于集合是不重复的，所以可以对字符串、列表、元组进行去重操作。

创建空集合

>>> s=set()

>>> s

set()

>>> s1=set([])　＃列表

>>> s1

set()

>>> s2=set(())　＃元组

>>> s2

set()

>>> s3=set({})　＃字典

>>> s3

set()

注意：想要创建空集合，你必须使用 set() 而不是 {}。后者用于创建空字典，我们在后面介绍的一种数据结构。

创建非空集合

　即列表，元组，字典不在是空值，举两个例子

>>> s1=set([1,2,3,4])

>>> s1

{1, 2, 3, 4}

>>> s3=set({'a':2,'b':3,'c':4})

>>> s3

{'c', 'a', 'b'}

注：字典转set集合，需要注意的是，只取了字典的key，相当于将字典中的dict.keys()列表转成set集合。