> 公众号：**Python极客专栏**，原创整理。

基础篇

1、Python有哪些特点和优点?

作为一门编程入门语言，Python主要有以下特点和优点：

- 可解释
- 具有动态特性
- 面向对象
- 简明简单
- 开源
- 具有强大的社区支持

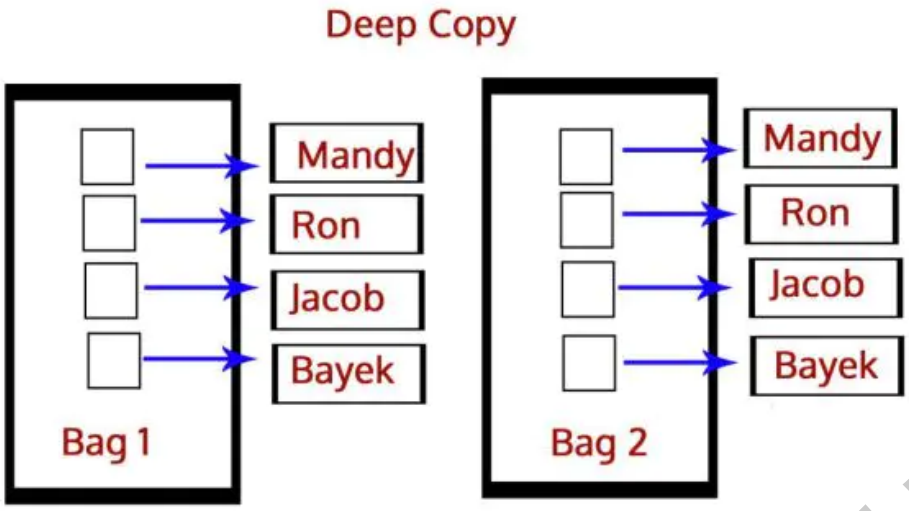
当然，实际上Python的优点远不止如此，可以阅读该文档，详细了解：

戳这里

2、深拷贝和浅拷贝之间的区别是什么?

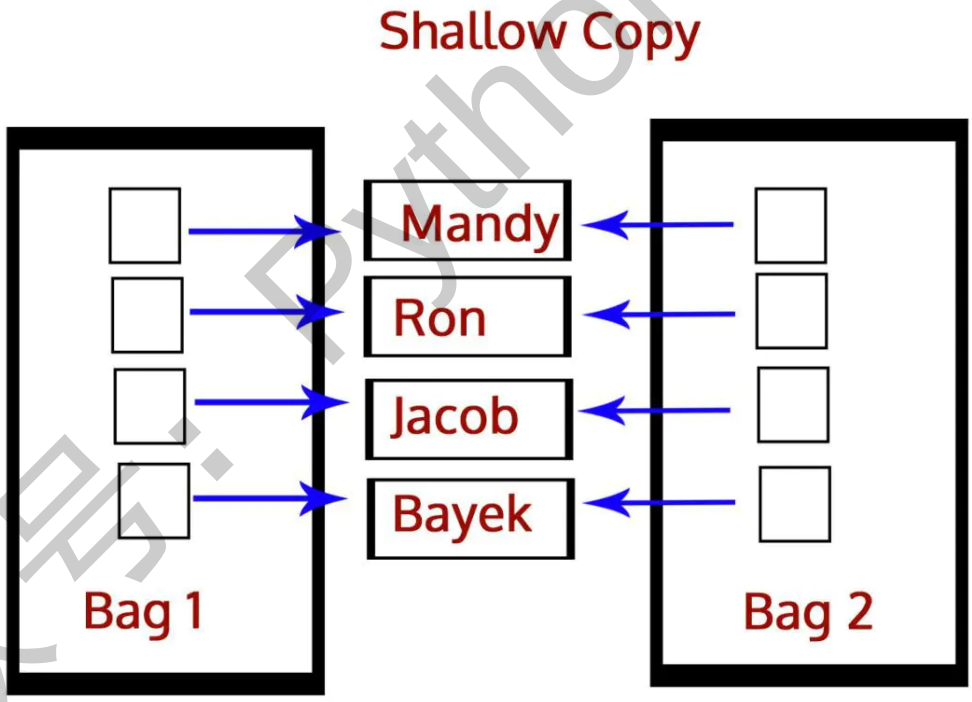
答：深拷贝就是将一个对象拷贝到另一个对象中，这意味着如果你对一个对象的拷贝做出改变时，不会影响原对象。在Python中，我们使用函数deepcopy()执行深拷贝，导入模块copy，如下所示：

```
>>> import copy
>>> b=copy.deepcopy(a)
```



而浅拷贝则是将一个对象的引用拷贝到另一个对象上，所以如果我们在拷贝中改动，会影响到原对象。
我们使用函数function()执行浅拷贝，使用如下所示：

```
>>> b=copy.copy(a)
```



3、 列表和元组之间的区别是?

答：二者的主要区别是列表是可变的，而元组是不可变的。举个例子，如下所示：

```
>>> mylist=[1,3,3]
>>> mylist[1]=2
>>> mytuple=(1,3,3)
>>> mytuple[1]=2
Traceback (most recent call last):
File "<pyshell#97>", line 1, in <module>
mytuple[1]=2
```

会出现以下报错：

```
TypeError: 'tuple' object does not support item assignment
```

关于列表和元组的更多内容，可以查看这里：

戳这里

4、 Python支持什么数据类型?

这是最基本的Python面试问题。

Python支持5种数据类型：

1、 Numbers（数字）——用于保存数值

```
>>> a=7.0
>>>
```

2、 Strings（字符串）——字符串是一个字符序列。我们用单引号或双引号来声明字符串。

```
>>> title="Ayushi's Book"
```

3、 Lists（列表）——列表就是一些值的有序集合，我们用方括号声明列表。

```
>>> colors=['red','green','blue']
>>> type(colors)
 <class 'list'>
```

4、 Tuples（元组）——元组和列表一样，也是一些值的有序集合，区别是元组是不可变的，意味着我们无法改变元组内的值。

```
>>> name=('Ayushi','Sharma')
>>> name[0]='Avery'
Traceback (most recent call last):
File "<pyshell#129>", line 1, in <module>
name[0]='Avery'
```

TypeError：'tuple' 对象不支持数据项分配

5、 Dictionary（字典）——字典是一种数据结构，含有键值对。我们用大括号声明字典。

```
>>> squares={1:1,2:4,3:9,4:16,5:25}
>>> type(squares)
<class 'dict'>
>>> type({})
<class 'dict'>
```

我们还可以使用字典引导式：

```
>>> squares={x:x**2 for x in range(1,6)}
>>> squares
{1: 1, 2: 4, 3: 9, 4: 16, 5: 25}
```

5、 在Python中如何声明一条注释?

和C++等编程语言不同，Python并没有多行注释，只有散列字符（#）。在符号#后的内容都被视作注释，解释器会自动将其忽略。

```
>>> #注释行1
>>> #注释行2
```

实际上你可以在代码中任何位置插入注释，用以解释代码。

6、在Python中怎样将字符串转换为整型变量?

如果字符串只含有数字字符，可以用函数int()将其转换为整数。

```
>>> int('227')
227
```

我们检查一下变量类型：

```
>>> type('227')
<class 'str'>
>>> type(int('227'))
<class 'int'>
```

7、单引号，双引号，三引号的区别?

- 单引号和双引号主要用来表示字符串

  ```
  比如:
  单引号:'python'
  双引号:"python"
  ```

- 三引号

三单引号:'''python '''，也可以表示字符串一般用来输入多行文本，或者用于大段的注释； 三双引号："""python"""，一般用在类里面,用来注释类,这样省的写文档,直接用类的对象`__doc__`访问获得文档。

区别

若你的字符串里面本身包含单引号,必须用双引号

```
例子:"can't find the log\n"
```

8、 在Python中怎样获取输入?

我们用函数input()从用户那里获取输入。在Python 2中，我们还有另一个函数raw_input()。比如input()将文本获取为参数值展现出来：

```
>>> a=input('Enter a number')
```

输入数字7

但是如果你多加注意，会发现它以字符串形式获取输入。

```
>>> type(a)
<class 'str'>
```

将之乘以2能得到：

```
>>> a*=2
>>> a
'77'
```

那么如果需要使用整数时呢?

我们使用int()函数。

```
>>> a=int(input('Enter a number'))
```

输入数字7.

现在当我们将之乘以2就会得到：

```
>>> a*=2
>>> a

14
```


9、 Python中的不可变集合（frozenset）是什么?

我们举例来回答此类Python面试问题。

首先，我们讨论一下什么是集合。集合就是一系列数据项的合集，不存在任何副本。另外，集合是无序的。

```
>>> myset={1,3,2,2}
>>> myset
{1, 2, 3}
```

这就意味着我们无法索引它。

```
>>> myset[0]
Traceback (most recent call last):
File "<pyshell#197>", line 1, in <module>
myset[0]
```

TypeError：‘set’不支持索引。 不过，集合是可变的。而不可变集合却不可变，这意味着我们无法改变它的值，从而也使其无法作为字典的键值。

```
>>> myset=frozenset([1,3,2,2])
>>> myset
frozenset({1, 2, 3})
>>> type(myset)
<class 'frozenset'>
```

更多关于集合的内容，查看这里。

10、 在Python中如何生成一个随机数?

要想生成随机数，我们可以从random模块中导入函数random()。

```
>>> from random import random
>>> random()
0.7931961644126482
```

这里我们调用help函数。

```
>>> help(random)
```

关于内置函数random的help运行结果：

```
random(…) method of random.Random instance
random() -> x in the interval [0, 1).
```

这意味着random()会返回一个大于等于0且小于1 的随机数。

我们还可以使用函数randint()，它会用两个参数表示一个区间，返回该区间内的一个随机整数。

```
>>> from random import randint
>>> randint(2,7)
6

>>> randint(2,7)
5

>>> randint(2,7)
7

>>> randint(2,7)
6
```

11、 怎样将字符串中第一个字母大写?

最简单的方法就是用capitalize()方法。

```
>>> 'ayushi'.capitalize()
‘Ayushi’
>>> type(str.capitalize)
<class 'method_descriptor'>
```

不过这也会让其它字母变为大写。

```
>>> '@yushi'.capitalize()
‘@yushi’
```

12、 如何检查字符串中所有的字符都为字母数字?

对于这个问题，我们可以使用isalnum()方法。

```
>>> 'Ayushi123'.isalnum()
True

>>> 'Ayushi123!'.isalnum()
False
```

我们还可以用其它一些方法：

```
>>> '123.3'.isdigit()
False

>>> '123'.isnumeric()
True

>>> 'ayushi'.islower()
True

>>> 'Ayushi'.isupper()
False
```

```
>>> 'Ayushi'.istitle()
True

>>> '   '.isspace()
True

>>> '123F'.isdecimal()
False
```

13、 什么是Python中的连接（concatenation）？

Python中的连接就是将两个序列连在一起，我们使用+运算符完成。

```
>>> '32'+'32'
‘3232’

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

>>> (2,3)+(4)
Traceback (most recent call last):
File "<pyshell#256>", line 1, in <module>
(2,3)+(4)
```

TypeError：只能将元组（不是“整数”）连接到元组。

这里4被看作一个整数，我们再来一次。

```
>>> (2,3)+(4,)
(2, 3, 4)
```

14、 请谈谈Python的不足之处。

Python有以下缺陷：

- Python的可解释特征会拖累其运行速度。
- 虽然Python在很多方面都性能良好，但在移动计算和浏览器方面表现不够好。
- 由于是动态语言，Python使用鸭子类型，即duck-typing，这会增加运行时错误。

15、 如果你困在了死循环里，怎么打破它?

出现了这种问题时，我们可以按Ctrl+C，这样可以打断执行程序。我们创建一个死循环来解释一下。

```
>>> def counterfunc(n):
    while(n==7):print(n)
>>> counterfunc(7)
7
```

```
7

7

7

7

7

7

7

Traceback (most recent call last):
 File "<pyshell#332>", line 1, in <module>
   counterfunc(7)
 File "<pyshell#331>", line 2, in counterfunc
   while(n==7):print(n)
KeyboardInterrupt

>>>
```

16、 如何在Python中创建自己的包? **

Python中创建包是比较方便的，只需要在当前目录建立一个文件夹，文件夹中包含一个**init**.py文件和若干个模块文件，其中**init**.py可以是一个空文件，但还是建议将包中所有需要导出的变量放到**all**中，这样可以确保包的接口清晰明了，易于使用。

17、 如何计算一个字符串的长度?

这个也比较简单，在我们想计算长度的字符串上调用函数len()即可。

```
>>> len('Ayushi Sharma')
13
```

18、 解释一下Python中的三元运算子

不像C++，我们在Python中没有?:，但我们有这个：

```
[on true] if [expression] else [on false]
```

如果表达式为True，就执行[on true]中的语句。否则，就执行[on false]中的语句。

下面是使用它的方法：

```
>>> a,b=2,3
>>> min=a if a<b else b
>>> min
```

运行结果：

```
2
```

```
>>> print("Hi") if a<b else print("Bye")
```

运行结果：

```
Hi
```

19、 Python中的字典是什么?

字典是C++和Java等编程语言中所没有的东西，它具有键值对。

```
>>> roots={25:5,16:4,9:3,4:2,1:1}
>>> type(roots)
<class 'dict'>
>>> roots[9]
```

运行结果为：

```
3
```

字典是不可变的，我们也能用一个推导式来创建它。

```
>>> roots={x**2:x for x in range(5,0,-1)}
>>> roots
```

运行结果：

```
{25: 5, 16: 4, 9: 3, 4: 2, 1: 1}
```

20、 请解释使用*args和**kwargs的含义

当我们不知道向函数传递多少参数时，比如我们向传递一个列表或元组，我们就使用*args。

```
>>> def func(*args):
    for i in args:
        print(i)
>>> func(3,2,1,4,7)
```

运行结果为：

```
3

2

1

4

7
```

在我们不知道该传递多少关键字参数时，使用**kwargs来收集关键字参数。

```
>>> def func(**kwargs):
    for i in kwargs:
        print(i,kwargs[i])
>>> func(a=1,b=2,c=7)
```

运行结果为：

```
a.1

b.2

c.7
```

21、 什么是负索引?

我们先创建这样一个列表：

```
>>> mylist=[0,1,2,3,4,5,6,7,8]
```

负索引和正索引不同，它是从右边开始检索。

```
>>> mylist[-3]
```

运行结果：

```
6
```

它也能用于列表中的切片：

```
>>> mylist[-6:-1]
```

结果：

```
[3, 4, 5, 6, 7]
```

22、 Python区分大小写吗?

如果能区分像myname和Myname这样的标识符，那么它就是区分大小写的。也就是说它很在乎大写和小写。我们可以用Python试一试：

```
>>> myname='Ayushi'
>>> Myname
Traceback (most recent call last):
File "<pyshell#3>", line 1, in <module>
```

运行结果：

```
Myname

NameError: name ‘Myname’ is not defined
```

可以看到，这里出现了NameError，所以Python是区分大小写的。

23、 Python中的标识符长度能有多长?

在Python中，标识符可以是任意长度。此外，我们在命名标识符时还必须遵守以下规则：

1. 只能以下划线或者 A-Z/a-z 中的字母开头
2. 其余部分可以使用 A-Z/a-z/0-9
3. 区分大小写
4. 关键字不能作为标识符，Python中共有如下关键字：

and	def	False	import	not	True
as	del	finally	in	or	try
assert	elif	for	is	pass	while
break	else	from	lambda	print	with
class	except	global	None	raise	yield
continue	exec	if	nonlocal	return	

24、怎样将字符串转换为小写?

我们使用lower()方法。

```
>>> 'AyuShi'.lower()
```

结果：

```
‘ayushi’
```

使用upper()方法可以将其转换为大写。

```
>>> 'AyuShi'.upper()
```

结果：

```
'AYUSHI'
```

另外，使用isupper()和islower()方法检查字符春是否全为大写或小写。

```
>>> 'AyuShi'.isupper()
False

>>> 'AYUSHI'.isupper()
True

>>> 'ayushi'.islower()
True

>>> '@yu$hi'.islower()
True

>>> '@YU$HI'.isupper()
True
```

那么，像@和$这样的字符既满足大写也满足小写。

Istitle()能告诉我们一个字符串是否为标题格式。

```
>>> 'The Corpse Bride'.istitle()
True
```

25、 Python中的pass语句是什么?

在用Python写代码时，有时可能还没想好函数怎么写，只写了函数声明，但为了保证语法正确，必须输入一些东西，在这种情况下，我们会使用pass语句。

```
 >>> def func(*args):
        pass
>>>
```

同样，break语句能让我们跳出循环。

```
>>> for i in range(7):
    if i==3: break
print(i)
```

结果：

```
0

1

2
```

最后，continue语句能让我们跳到下个循环。

```
>>> for i in range(7):
    if i==3: continue
print(i)
```

结果：

```
0

1

2

4

5

6
```

26、 解释一下Python中的//， ** 运算符

//运算符执行地板除法（向下取整除），它会返回整除结果的整数部分。

```
>>> 7//2
3
```

这里整除后会返回3.5。

同样地，**执行取幂运算。a**b会返回a的b次方。

```
>>> 2**10
1024
```

最后，%执行取模运算，返回除法的余数。

```
>>> 13%7
6
>>> 3.5%1.5
0.5
```

27、 在Python中有多少种运算符？解释一下算数运算符。

在Python中，我们有7种运算符：算术运算符、关系运算符、赋值运算符、逻辑运算符、位运算符、成员运算符、身份运算符。

我们有7个算术运算符，能让我们对数值进行算术运算：

1.加号（+），将两个值相加

```
>>> 7+8
15
```

2.减号（-），将第一个值减去第二个值

```
>>> 7-8
-1
```

3.乘号（*），将两个值相乘

```
>>> 7*8
56
```

4.除号（/），用第二个值除以第一个值

```
>>> 7/8
0.875
>>> 1/1
1.0
```

5.向下取整除、取模和取幂运算，参见上个问题。

28、解释一下Python中的关系运算符

关系运算符用于比较两个值。

1.小于号（<），如果左边的值较小，则返回True。

```
>>> 'hi'<'Hi'
False
```

2.大于号（>），如果左边的值较大，则返回True。

```
>>> 1.1+2.2>3.3
True
```

3.小于等于号（<=），如果左边的值小于或等于右边的值，则返回Ture。

```
>>> 3.0<=3
True
```

4.大于等于号（>=），如果左边的值大于或等于右边的值，则返回True。

```
>>> True>=False
True
```

1. 等于号（==），如果符号两边的值相等，则返回True。

```
>>> {1,3,2,2}=={1,2,3}
True
```

1. 不等于号（!=），如果符号两边的值不相等，则返回True。

```
>>> True!=0.1
True
>>> False!=0.1
True
```

29、 解释一下Python中的赋值运算符

这在Python面试中是个重要的面试问题。

我们将所有的算术运算符和赋值符号放在一起展示：

```
>>> a=7
>>> a+=1
>>> a
8

>>> a-=1
>>> a
7

>>> a*=2
>>> a
14

>>> a/=2
>>> a
7.0

>>> a**=2
>>> a
49

>>> a//=3
>>> a
16.0

>>> a%=4
>>> a
0.0
```


30、 解释一下Python中的逻辑运算符

Python中有3个逻辑运算符：and，or，not。

```
>>> False and True
False

>>> 7<7 or True
True

>>> not 2==2
False
```

31、 解释一下Python中的成员运算符

通过成员运算符'in'和'not in'，我们可以确认一个值是否是另一个值的成员。

```
>>> 'me' in 'disappointment'
True

>>> 'us' not in 'disappointment'
True
```

32、 解释一下Python中的身份运算符

这也是一个在Python面试中常问的问题。

通过身份运算符'is'和'is not'，我们可以确认两个值是否相同。

```
>>> 10 is '10'
False

>>> True is not False
True
```

33、 讲讲Python中的位运算符

该运算符按二进制位对值进行操作。

1. 与（&），按位与运算符：参与运算的两个值,如果两个相应位都为1,则该位的结果为1,否则为0

```
>>> 0b110 & 0b010
2
```

2.或（|），按位或运算符：只要对应的二个二进位有一个为1时，结果位就为1。

```
>>> 3|2
3
```

3.异或（^），按位异或运算符：当两对应的二进位相异时，结果为1

```
>>> 3^2
1
```

4.取反（~），按位取反运算符：对数据的每个二进制位取反,即把1变为0,把0变为1

```
>>> ~2
-3
```

5.左位移（<<），运算数的各二进位全部左移若干位，由 << 右边的数字指定了移动的位数，高位丢弃，低位补0

```
>>> 1<<2
4
```

6.右位移（>>），把">>"左边的运算数的各二进位全部右移若干位，>> 右边的数字指定了移动的位数

```
>>> 4>>2
1
```

更多关于运算符的知识，参考这里：

戳这里

34、 docstring是什么？**

Docstring是一种文档字符串，用于解释构造的作用。我们在函数、类或方法中将它放在首位来描述其作用。我们用三个单引号或双引号来声明docstring。

```
>>> def sayhi():
    """
用该函数打印Hi
"""
  print("Hi")
>>> sayhi()

Hi
```

要想获取一个函数的docstring，我们使用它的*doc*属性。

```
要想获取一个函数的docstring，我们使用它的__doc__属性。
>>> sayhi.__doc__
'\n\tThis function prints Hi\n\t'
```

和注释不同，docstring在运行时会保留下来。

35. 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，但是是无法修改的。即元组是不可变的。

字典（dict）：字典是无序的集合，是由 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。

36. 简述上述数据类型的常用方法

字符串

1、切片

```
mystr='luobodazahui'
mystr[1:3]
```

output

```
'uo'
```

2、format

```
mystr2 = "welcome to luobodazahui, dear {name}"
mystr2.format(name="baby")
```

output

```
'welcome to luobodazahui, dear baby'
```

3、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。

```
mylist = ['luo', 'bo', 'da', 'za', 'hui']
mystr3 = '-'.join(mylist)
print(mystr3)
```

outout

```
'luo-bo-da-za-hui'
```

4、replace

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

```
mystr4 = 'luobodazahui-haha'
print(mystr4.replace('haha', 'good'))
```

output

```
luobodazahui-good
```

5、split

切割字符串,得到一个列表。

```
mystr5 = 'luobo,dazahui good'
# 以空格分割
print(mystr5.split())
# 以h分割
print(mystr5.split('h'))
# 以逗号分割
print(mystr5.split(','))
```

output

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

列表

1、切片

同字符串

2、append 和 extend

向列表中国添加元素

```
mylist1 = [1, 2]
mylist2 = [3, 4]
mylist3 = [1, 2]
mylist1.append(mylist2)
print(mylist1)
mylist3.extend(mylist2)
print(mylist3)
```

outout

```
[1, 2, [3, 4]]
[1, 2, 3, 4]
```

3、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)
mylist4.pop()
print(mylist4)
mylist4.remove('c')
print(mylist4)
```

output

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

4、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。

reverse：是将list逆置。

```
mylist5 = [1, 5, 2, 3, 4]
mylist5.sort()
print(mylist5)
mylist5.reverse()
print(mylist5)
```

output

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

字典

1、清空字典

dict.clear()

```
dict1 = {'key1':1, 'key2':2}
dict1.clear()
print(dict1)
```

output

```
{}
```

2、指定删除

使用 pop 方法来指定删除字典中的某一项

```
dict1 = {'key1':1, 'key2':2}
d1 = dict1.pop('key1')
print(d1)
print(dict1)
```

output

```
1
{'key2': 2}
```

3、遍历字典

```
dict2 = {'key1':1, 'key2':2}
mykey = [key for key in dict2]
print(mykey)
myvalue = [value for value in dict2.values()]
print(myvalue)
key_value = [(k, v) for k, v in dict2.items() ]
print(key_value)
```

output

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

4、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)
```

output

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

37、python中内置的数据结构有几种?

a. 整型 int、 长整型 long、浮点型 float、 复数 complex

b. 字符串 str、 列表 list、 元祖 tuple

c. 字典 dict 、 集合 set

d. Python3 中没有 long，只有无限精度的 int

进阶篇

1、在Python中如何使用多进制数字?

我们在Python中，除十进制外还可以使用二进制、八进制和十六进制。

1. 二进制数字由0和1组成，我们使用 0b 或 0B 前缀表示二进制数。

```
>>> int(0b1010)
10
```

2.使用bin()函数将一个数字转换为它的二进制形式。

```
>>> bin(0xf)
'0b1111'
```

3.八进制数由数字 0-7 组成，用前缀 0o 或 0O 表示 8 进制数。

```
>>> oct(8)
'0o10'
```

4.十六进数由数字 0-15 组成，用前缀 0x 或者 0X 表示 16 进制数。

```
>>> hex(16)
'0x10'

>>> hex(15)
'0xf'
```

2、 怎样获取字典中所有键的列表?

使用 keys() 获取字典中的所有键

```
>>> mydict={'a':1,'b':2,'c':3,'e':5}
>>> mydict.keys()
dict_keys(['a', 'b', 'c', 'e'])
```

3、 为何不建议以下划线作为标识符的开头

因为Python并没有私有变量的概念，所以约定速成以下划线为开头来声明一个变量为私有。所以如果你不想让变量私有，就不要使用下划线开头。

4、 怎样声明多个变量并赋值?

一共有两种方式:

```
>>> a,b,c=3,4,5 #This assigns 3, 4, and 5 to a, b, and c respectively
>>> a=b=c=3 #This assigns 3 to a, b, and c
```

5、 元组的解封装是什么?

首先我们来看解封装:

```
>>> mytuple=3,4,5
>>> mytuple
(3, 4, 5)
```

这将 3，4，5 封装到元组 mytuple 中。

现在我们将这些值解封装到变量 x，y，z 中：

```
>>> x,y,z=mytuple
>>> x+y+z
```

得到结果12.

6、 解释Python中的help()和dir()函数

Help()函数是一个内置函数，用于查看函数或模块用途的详细说明：

```
>>> import copy
>>> help(copy.copy)
```

运行结果为：

```
Help on function copy in module copy:

copy(x)

Shallow copy operation on arbitrary Python objects.

See the module's __doc__ string for more info.
```

Dir()函数也是Python内置函数，dir() 函数不带参数时，返回当前范围内的变量、方法和定义的类型列表；带参数时，返回参数的属性、方法列表。

以下实例展示了 dir 的使用方法：

```
>>> dir(copy.copy)
```

运行结果为：

```
['__annotations__', '__call__', '__class__', '__closure__', '__code__',
'__defaults__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__',
'__format__', '__ge__', '__get__', '__getattribute__', '__globals__', '__gt__',
'__hash__', '__init__', '__init_subclass__', '__kwdefaults__', '__le__',
'__lt__', '__module__', '__name__', '__ne__', '__new__', '__qualname__',
'__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__',
'__str__', '__subclasshook__']
```

7、 Python中的闭包是什么?

当一个嵌套函数在其外部区域引用了一个值时，该嵌套函数就是一个闭包。其意义就是会记录这个值。

```
>>> def A(x):
    def B():
        print(x)
    return B
>>> A(7)()
```

结果：

```
7
```

更多关于闭包的知识，请参看这里：

戳这里

8、 什么是猴子补丁?

在运行期间动态修改一个类或模块。

```
>>> class A:
    def func(self):
        print("Hi")
>>> def monkey(self):
print "Hi, monkey"
>>> m.A.func = monkey
>>> a = m.A()
>>> a.func()
```

运行结果为：

```
Hi, Monkey
```

9、 什么是递归?

在调用一个函数的过程中，直接或间接地调用了函数本身这个就叫递归。但为了避免出现死循环，必须要有一个结束条件，举个例子：

```
>>> def facto(n):
    if n==1: return 1
    return n*facto(n-1)
>>> facto(4)
24
```

10、 什么是生成器?

生成器会生成一系列的值用于迭代，这样看它又是一种可迭代对象。它是在for循环的过程中不断计算出下一个元素，并在适当的条件结束for循环。

我们定义一个能逐个“yield”值的函数，然后用一个for循环来迭代它。

```
>>> def squares(n):
    i=1
    while(i<=n):
        yield i**2
        i+=1
>>> for i in squares(7):
    print(i)
1

4

9

16

25

36

49
```

更多关于生成器的内容，参看这里。

11、 什么是迭代器?

迭代器是访问集合元素的一种方式。迭代器对象从集合的第一个元素开始访问，直到所有的元素被访问完结束。迭代器只能往前不会后退。我们使用inter()函数创建迭代器。

```
odds=iter([1,3,5,7,9])
```

每次想获取一个对象时，我们就调用next()函数。

```
>>> next(odds)
1

>>> next(odds)
3

>>> next(odds)
5

>>> next(odds)
7

>>> next(odds)
9
```

现在我们再次调用它，会抛出StopIteration异常。这是因为它已经抵达需要迭代的值的尾部。

```
>>> next(odds)
Traceback (most recent call last):
File "<pyshell#295>", line 1, in <module>
next(odds)
StopIteration
```

更多关乎迭代器的内容，参看这里。

12、 请说说生成器和迭代器之间的区别

- 在使用生成器时，我们创建一个函数；在使用迭代器时，我们使用内置函数iter()和next()。
- 在生成器中，我们使用关键字'yield'来每次生成/返回一个对象。
- 生成器中有多少'yield'语句，你可以自定义。
- 每次'yield'暂停循环时，生成器会保存本地变量的状态。而迭代器并不会使用局部变量，它只需要一个可迭代对象进行迭代。
- 使用类可以实现你自己的迭代器，但无法实现生成器。
- 生成器运行速度快，语法简洁，更简单。
- 迭代器更能节约内存。

关于生成器和迭代器二者的对比，更多内容查看这里。

13、 在Python中如何实现多线程?

一个线程就是一个轻量级进程，多线程能让我们一次执行多个线程。我们都知道，Python是多线程语言，其内置有多线程工具包。

Python中的GIL（全局解释器锁）确保一次执行单个线程。一个线程保存GIL并在将其传递给下个线程之前执行一些操作，这会让我们产生并行运行的错觉。但实际上，只是线程在CPU上轮流运行。当然，所有的传递会增加程序执行的内存压力。

14、 解释一下Python中的继承

当一个类继承自另一个类，它就被称为一个子类/派生类，继承自父类/基类/超类。它会继承/获取所有类成员（属性和方法）。

继承能让我们重新使用代码，也能更容易的创建和维护应用。Python支持如下种类的继承：

- 单继承：一个类继承自单个基类
- 多继承：一个类继承自多个基类
- 多级继承：一个类继承自单个基类，后者则继承自另一个基类
- 分层继承：多个类继承自单个基类
- 混合继承：两种或多种类型继承的混合 更多关于继承的内容，参见：

戳这里

15、 在Python中是如何管理内存的?

对象的引用计数机制 Python内部使用引用计数，来保持追踪内存中的对象，所有对象都有引用计数。

引用计数增加的情况：

总结一下对象会在一下情况下引用计数加1：

1. 对象被创建：x='spam'
2. 另外的别人被创建：y=x
3. 被作为参数传递给函数：foo(x)
4. 作为容器对象的一个元素：a=[1,x,'33']

引用计数减少情况

1. 一个本地引用离开了它的作用域。比如上面的foo(x)函数结束时，x指向的对象引用减1。
2. 对象的别名被显式的销毁：del x ；或者del y
3. 对象的一个别名被赋值给其他对象：x=789
4. 对象从一个窗口对象中移除：myList.remove(x)
5. 窗口对象本身被销毁：del myList，或者窗口对象本身离开了作用域。

垃圾回收

1. 当内存中有不再使用的部分时，垃圾收集器就会把他们清理掉。它会去检查那些引用计数为0的对象，然后清除其在内存的空间。当然除了引用计数为0的会被清除，还有一种情况也会被垃圾收集器清掉：当两个对象相互引用时，他们本身其他的引用已经为0了。
2. 垃圾回收机制还有一个循环垃圾回收器, 确保释放循环引用对象(a引用b, b引用a, 导致其引用计数永远不为0)。

在Python中，许多时候申请的内存都是小块的内存，这些小块内存在申请后，很快又会被释放，由于这些内存的申请并不是为了创建对象，所以并没有对象一级的内存池机制。这就意味着Python在运行期间会大量地执行malloc和free的操作，频繁地在用户态和核心态之间进行切换，这将严重影响Python的执行效率。为了加速Python的执行效率，Python引入了一个内存池机制，用于管理对小块内存的申请和释放。

内存池机制

1. Python提供了对内存的垃圾收集机制，但是它将不用的内存放到内存池而不是返回给操作系统；
2. Pymalloc机制：为了加速Python的执行效率，Python引入了一个内存池机制，用于管理对小块内存的申请和释放；
3. 对于Python对象，如整数，浮点数和List，都有其独立的私有内存池，对象间不共享他们的内存池。也就是说如果你分配又释放了大量的整数，用于缓存这些整数的内存就不能再分配给浮点数。

16、 当退出Python时，是否释放全部内存?

答案是No。循环引用其它对象或引用自全局命名空间的对象的模块，在Python退出时并非完全释放。

另外，也不会释放C库保留的内存部分。

17、 请写一个Python逻辑，计算一个文件中的大写字母数量

```
>>> import os

>>> os.chdir('C:\\Users\\lifei\\Desktop')
>>> with open('Today.txt') as today:
    count=0
    for i in today.read():
        if i.isupper():
            count+=1
print(count)
```

运行结果：

```
26
```

18、 怎么移除一个字符串中的前导空格?

字符串中的前导空格就是出现在字符串中第一个非空格字符前的空格。我们使用方法lstrip()可以将它从字符串中移除。

```
>>> '   Ayushi '.lstrip()
```

结果：

```
‘Ayushi   ’
```

可以看到，该字符串既有前导字符，也有后缀字符，调用lstrip()去除了前导空格。如果我们想去除后缀空格，就用rstrip()方法。

```
>>> '   Ayushi '.rstrip()
```

结果：

```
‘   Ayushi’
```

19、 如何以就地操作方式打乱一个列表的元素?

为了达到这个目的，我们从random模块中导入shuffle()函数。

```
>>> from random import shuffle
>>> shuffle(mylist)
>>> mylist
```

运行结果：

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

20、 解释Python中的join()和split()函数

Join()能让我们将指定字符添加至字符串中。

```
>>> ','.join('12345')
```

运行结果：

```
‘1,2,3,4,5’
```

Split()能让我们用指定字符分割字符串。

```
>>> '1,2,3,4,5'.split(',')
```

运行结果：

```
['1', '2', '3', '4', '5']
```

21、 什么是函数?

当我们想执行一系列语句时，我们可以为其赋予一个名字。我们来定义一个函数，让它取两个数返回一个更大的数。

```
>>> def greater(a,b):
返回 a if a>b else b
>>> greater(3,3.5)
3.5
```

你可以自己创建函数，也可以使用Python的很多内置函数，看这里。

22、 解释lambda表达式，什么时候会用到它?

如果我们需要一个只有单一表达式的函数，我们可以匿名定义它。拉姆达表达式通常是在需要一个函数，但是又不想费神去命名一个函数的场合下使用，也就是指匿名函数。

假如我们想将上面Q 14中的函数定义为拉姆达表达式，可以在解释器中输入如下代码：

```
>>> (lambda a,b:a if a>b else b)(3,3.5)
3.5
```

这里，a和b都是输入，a if a>b else b就是返回的输入，参数为3和3.5.

当然，也有可能没有任何输入。

```
>>> (lambda :print("Hi"))()
Hi
```

更多关于拉姆达表达式的内容，参考这里。

23、 函数zip()的是干嘛的?

Python新手可能对这个函数不是很熟悉，zip()可以返回元组的迭代器。

```
>>> list(zip(['a','b','c'],[1,2,3]))
[('a', 1), ('b', 2), ('c', 3)]
```

在这里zip()函数对两个列表中的数据项进行了配对，并用它们创建了元组。

```
>>> list(zip(('a','b','c'),(1,2,3)))
[('a', 1), ('b', 2), ('c', 3)]
```

24、 解释Python的参数传递机制

Python使用按引用传递（pass-by-reference）将参数传递到函数中。如果你改变一个函数内的参数，会影响到函数的调用。这是Python的默认操作。不过，如果我们传递字面参数，比如字符串、数字或元组，它们是按值传递，这是因为它们是不可变的。

25、 如何用Python找出你目前在哪个目录？**

我们可以使用函数/方法getcwd()，从模块os中将其导入。

```
>>> import os
>>> os.getcwd()
'C:\\Users\\lifei\\AppData\\Local\\Programs\\Python\\Python36-32'
>>> type(os.getcwd)
<class 'builtin_function_or_method'>
```

我们还可以用chdir()修改当前工作目录。

```
>>> os.chdir('C:\\Users\\lifei\\Desktop')
>>> os.getcwd()
'C:\\Users\\lifei\\Desktop'
```

26、 怎样发现字符串中与‘cake’押韵的第一个字?

我们可以使用函数search()，然后用group()获取输出。

```
>>> import re
>>> rhyme=re.search('.ake','I would make a cake, but I hate to bake')
>>> rhyme.group()
'make'
```

我们知道，函数search()会在第一次匹配时停止运行，这样我们就能得到第一个与‘cake’押韵的字。

27、 如何以相反顺序展示一个文件的内容?

我们首先回到桌面，使用模块os中的chdir()函数/方法。

```
>>> import os
>>> os.chdir('C:\\Users\\lifei\\Desktop')
```

这里我们要使用的文件时Today.txt，它的内容如下：

```
OS, DBMS, DS, ADA

HTML, CSS, jQuery, JavaScript

Python, C++, Java

This sem's subjects

Debugger

itertools

Container
```

我们将内容读取为一个列表，然后在上面调用reversed()函数：

```
>>> for line in reversed(list(open('Today.txt'))):
   print(line.rstrip())
container

itertools

Debugger

This sem's subjects

Python, C++, Java

HTML, CSS, jQuery, JavaScript

OS, DBMS, DS, ADA
```

如果没有rstrip()，我们会在输出中得到空行。

28、 什么是Tkinter？

TKinter是一款很知名的Python库，用它我们可以制作图形用户界面。其支持不同的GUI工具和窗口构件，比如按钮、标签、文本框等等。这些工具和构件均有不同的属性，比如维度、颜色、字体等。

我们也能导入Tkinter模块。

```
>>> import tkinter
>>> top=tkinter.Tk()
```

这会为你创建一个新窗口，然后可以在窗口上添加各个构件。

29、 请谈谈.pyc文件和.py文件的不同之处

虽然这两种文件均保存字节代码，但.pyc文件是Python文件的编译版本，它有平台无关的字节代码，因此我们可以在任何支持.pyc格式文件的平台上执行它。Python会自动生成它以优化性能（加载时间，而非运行速度）。

30、简单介绍一下python函数式编程?

在函数式编程中，函数是基本单位，变量只是一个名称，而不是一个存储单元。

除了匿名函数外，Python还使用fliter(),map(),reduce(),apply()函数来支持函数式编程。 所以你的重点围绕fliter(),map(),reduce().apply()来介绍就可以顺利和面试官达成一致

31、python中函数装饰器有什么作用?

装饰器本质上是一个Python函数，它可以让其他函数在不需要做任何代码变动的前提下增加额外功能，装饰器的返回值也是一个函数对象。

它经常用于有切面需求的场景，比如： `插入日志`、`性能测试`、`事务处理`、`缓存`、`权限校验`等场景。

有了装饰器，就可以抽离出大量与函数功能本身无关的雷同代码并继续重用。

32、请解释一下python的线程锁Lock和Rlock的区别，以及你曾经在项目中是如何使用的?

从原理上来说：在同一线程内，对RLock进行多次acquire()操作，程序不会阻塞。 资源总是有限的，程序运行如果对同一个对象进行操作，则有可能造成资源的争用，甚至导致死锁 也可能导致读写混乱

33、字典、列表查询时的时间复杂度是怎样的?

列表是序列，可以理解为数据结构中的数组，字典可以理解为数据结构中的`hashmap`，python中list对象的存储结构采用的是线性表，因此其查询复杂度为`O(n)`。 而dict对象的存储结构采用的是散列表(hash表)，其在最优情况下查询复杂度为O(1)。 dict的占用内存稍比list大，会在1.5倍左右。

34、 python下多线程的限制以及多进程中传递参数的方式?

python多线程有个全局解释器锁（global interpreter lock），简称GIL，这个GIL并不是python的特性，他是只在Cpython解释器里引入的一个概念，而在其他的语言编写的解释器里就没有这个GIL例如：Jython。

这个锁的意思是任一时间只能有一个线程运用解释器，跟单cpu跑多个程序一个意思，我们都是轮着用的，这叫“并发”，不是“并行”。

为什么会有GIL?

多核CPU的出现，充分利用多核，采用多线程编程慢慢普及，难点就是线程之间数据的一致性和状态同步

说到GIL解释器锁，我们容易想到在多线程中共享全局变量的时候会有线程对全局变量进行的资源竞争，会对全局变量的修改产生不是我们想要的结果，而那个时候我们用到的是python中线程模块里面的互斥锁，哪样的话每次对全局变量进行操作的时候，只有一个线程能够拿到这个全局变量；看下面的代码：

```
import threading
global_num = 0
```

```
def test1():
    global global_num
    for i in range(1000000):
        global_num += 1

    print("test1", global_num)

def test2():
    global global_num
    for i in range(1000000):
        global_num += 1

    print("test2", global_num)

t1 = threading.Thread(target=test1)
t2 = threading.Thread(target=test2)
t1.start()
t2.start()
```

接下来加入互斥锁

```
import threading
import time
global_num = 0

lock = threading.Lock()

def test1():
    global global_num
    lock.acquire()
    for i in range(1000000):
        global_num += 1
    lock.release()
    print("test1", global_num)

def test2():
    global global_num
    lock.acquire()
    for i in range(1000000):
        global_num += 1
    lock.release()
    print("test2", global_num)

t1 = threading.Thread(target=test1)
t2 = threading.Thread(target=test2)
start_time = time.time()

t1.start()
t2.start()
```


哪些情况适合用多线程呢：

> 只要在进行耗时的IO操作的时候，能释放GIL，所以只要在IO密集型的代码里，用多线程就很合适

哪些情况适合用多进程呢：

> 用于计算密集型，比如计算某一个文件夹的大小

多进程间同享数据

> 多进程间同享数据，能够运用multiprocession.Value和multiprocessing.Array

35、解释一下python的and-or语法

> bool and a or b

相当于bool? a: b

```
>>> a = "first"
>>> b = "second"
>>> 1 and a or b  # 输出内容为 'first'
>>> 0 and a or b  # 输出内容为 'second'
```

上述内容你应该可以理解，但是还存在一个问题，请看下面的代码

```
>>> a = ""
>>> b = "second"
>>> 1 and a or b # 输出内容为 'second'

复制代码
```

> 因为 a 是一个空串，空串在一个布尔环境中被Python看成假值，这个表达式将“失败”，且返回 b 的值。 如果你不将它想象成象 bool ? a : b 一样的语法，而把它看成纯粹的布尔逻辑，这样的话就会得到正确的理解。 1 是真，a 是假，所以 1 and a 是假。假 or b 是b。

应该将 and-or 技巧封装成一个函数：

```
def choose(bool, a, b):
    return (bool and [a] or [b])[0]
```

因为 [a] 是一个非空列表，它永远不会为假。甚至 a 是 0 或 '' 或其它假值，列表[a]为真，因为它有一个元素。

36、请描述方法重载与方法重写?

方法重载

是在一个类里面，方法名字相同，而参数不同。返回类型可以相同也可以不同。 重载是让类以统一的方式处理不同类型数据的一种手段。

方法重写

子类不想原封不动地继承父类的方法，而是想作一定的修改，这就需要采用方法的重写。方法重写又称方法覆盖。

37、 Python 中的 os 模块常见方法?

os 属于 python内置模块，所以细节在官网有详细的说明，本道面试题考察的是基础能力了，所以把你知道的都告诉面试官吧 官网地址 `https://docs.python.org/3/library/os.html`

os模块包含了很多操作文件和目录的函数

os对象方法

函数名称	用途
os.remove()	删除文件
os.walk()	生成目录树下的所有文件名
os.chdir()	改变目录
os.getcwd()	返回当前工作目录
os.listdir(path=".")	列举指定目录中的文件名(“.”表示当前目录，“..”表示上一级目录)
os.mkdir(path)	创建建单层目录，如果该目录已存在则抛出异常
os.rename(old,new)	将文件old重命名为new
更多内容查阅官网吧	官网很容易看的

38、如何提高Python 程序的运行性能?

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

39、 Python中的不可变集合（frozenset）是什么?

集合分为两种类型：

- set —— 可变集合。集合中的元素可以动态的增加或删除。
- frozenset —— 不可变集合。集合中的元素不可改变。

首先，我们讨论一下什么是集合。集合就是一系列数据项的合集，不存在任何副本。另外，集合是无序的。

```
>>> myset={1,3,2,2}
>>> myset
{1, 2, 3}
```

这就意味着我们无法索引它

```
>>> myset={1,3,2,2}
>>> myset
{1, 2, 3}
>>> myset[0]
Traceback (most recent call last):
  File "<pyshell#11>", line 1, in <module>
    myset[0]
TypeError: 'set' object does not support indexing
>>>
```

TypeError：'set'不支持索引。 集合是可变的。而不可变集合却不可变，这意味着我们无法改变它的值，从而也使其无法作为字典的键值。

```
>>> myset=frozenset([1,3,2,2])
>>> myset
frozenset({1, 2, 3})
>>> type(myset)
<class 'frozenset'>
>>>
```

40、 print 调用 Python 中底层的什么方法?

print

print() 用 `sys.stdout.write()` 实现

```
    import sys

    print('hello')
    sys.stdout.write('hello')
    print('world')
    # 结果:
    # hello
    # helloworld
```

复制代码

> 上述代码你应该可以总结一下。

1. sys.stdout.write()结尾没有换行，而print()是自动换行的。
2. write()只接收字符串格式的参数。
3. print()能接收多个参数输出，write()只能接收一个参数输出。

input

> Python3中的input() 用 sys.stdin.readline() 实现。

```
    import sys
    a = sys.stdin.readline()
    print(a, len(a))

    b = input()
    print(b, len(b))
```

```
    # 结果:
    # hello
    # hello
    #  6
    # hello
    # hello 5
```
复制代码

41、range 和 xrange 的区别?

首先我们看看range:

- 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`。注意这里是生成一个序列。
- xrange的用法与range相同，即xrange([start,] stop[, step])根据start与stop指定的范围以及step设定的步长,它所不同的是xrange并不是生成序列，而是作为`一个生成器`。即她的数据生成一个取出一个。

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，所以相对来说，xrange比range性能优化很多，因为他不需要一下子开辟一块很大的内存，特别是数据量比较大的时候。

注意:

1. xrange和range这两个基本是使用在循环的时候。
2. 当需要输出一个列表的时候，就必须要使用range了。

42、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？介绍一下 except 的作用和用法?

会继续处理 finally 中的代码；用 raise 方法可以抛出自定义异常

- 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

43、说一说Python自省?

在python中，检查某些事物以确定它是什么、它知道什么以及它能做什么。

自省向程序员提供了极大的灵活性和控制力。

说的更简单直白一点：==自省就是面向对象的语言所写的程序在运行时，能够知道对象的类型。简单一句就是，运行时能够获知对象的类型。==

例如python, buby, object-C, c++都有自省的能力，这里面的c++的自省的能力最弱，只能够知道是什么类型，而像python可以知道是什么类型，还有什么属性。

Python中比较常见的自省（introspection）机制(函数用法)有：`dir()`，`type()`，`hasattr()`，`isinstance()`，通过这些函数，我们能够在程序运行时得知对象的类型，判断对象是否存在某个属性，访问对象的属性。

1. dir() 函数是 Python 自省机制中最著名的部分了。它返回传递给它的任何对象的属性名称经过排序的列表。如果不指定对象，则 dir() 返回当前作用域中的名称。
2. type() 函数有助于我们确定对象是字符串还是整数，或是其它类型的对象。
3. 对象拥有属性，并且 dir() 函数会返回这些属性的列表。但是，有时我们只想测试一个或多个属性是否存在。如果对象具有我们正在考虑的属性，那么通常希望只检索该属性。这个任务可以由 hasattr() 和 getattr() 函数来完成。
4. isinstance() 函数测试对象，以确定它是否是某个特定类型或定制类的实例。

其他可以参考博客：blog.csdn.net/qq_34979346...

44、Python里面如何实现tuple和list的转换?

函数`tuple(seq)`可以把所有`可迭代的(iterable)序列`转换成一个`tuple`，元素不变，排序也不变

list转为tuple：

```
temp_list = [1,2,3,4,5]
```

将temp_list进行强制转换：`tuple(temp_list)` 确定是否转换成功：`print(type(temp_list))`

函数`list(seq)`可以把所有的`序列和可迭代的对象`转换成一个list,元素不变，排序也不变

tuple 转为list：

```
temp_tuple = (1,2,3,4,5)
```

方法类似，也是进行强制转换即可：`list(temp_tuple)` 确定是否转换成功：`print(type(temp_tuple))`

45、Python里面search()和match()的区别?

它们两个都在`re模块`中

- `match()`函数是在string的开始位置匹配，如果不匹配，则返回None;
- `search()`会扫描整个string查找匹配;

match()

```
>>> import re
>>> print(re.match('hello','helloworld').span())  # 开头匹配到
(0, 5)
>>> print(re.match('hello','nicehelloworld').span()) # 开头没有匹配到
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    print(re.match('hello','nicehelloworld').span())
AttributeError: 'NoneType' object has no attribute 'span'
>>>
```

search()

```
>>> print(re.search('a','abc'))
<_sre.SRE_Match object; span=(0, 1), match='a'>
>>> print(re.search('a','bac').span())
(1, 2)
>>>
```

46、 is 和 == 的区别?

> hon中对象包含的三个基本要素，分别是：`id(身份标识)`、`type(数据类型)`和`value(值)`

id 身份标识，就是在内存中的地址

完整的举例

```
>>> a = 'hello'
>>> b = 'hello'
>>> print(a is b)
True
>>> print(a==b)
True
>>> a = 'hello world'
>>> b = 'hello world'
>>> print(a is b)
False
>>> print(a == b)
True
>>> a = [1,2,3]
>>> b = [1,2,3]
>>> print(a is b)
False
>>> print(a == b)
True
>>> a = [1,2,3]
>>> b = a
>>> print(a is b)
True
>>> print(a == b)
True
>>>
```

- `==`是python标准操作符中的`比较操作符`，用来比较判断`两个对象的value(值)`是否相等

- `is`也被叫做同一性运算符（对象标示符），这个运算符比较判断的是对象间的`唯一身份标识`，也就是`id`（内存中的地址）是否相同

我们在检查 a is b 的时候，其实相当于检查`id(a) == id(b)`。而检查`a == b`的时候，实际是调用了对象 a 的`__eq()__`方法，`a == b`相当于`a.__eq__(b)`。

这里还有一个问题，为什么 a 和 b 都是 "hello" 的时候，a is b 返回True，而 a 和 b都是 "hello world" 的时候，a is b 返回False呢?

这是因为前一种情况下Python的`字符串驻留机制`起了作用。对于较小的字符串，为了提高系统性能Python会保留其值的一个副本，当创建新的字符串的时候直接指向该副本即可。

所以 "hello" 在内存中只有一个副本，a 和 b 的 id 值相同，而 "hello world" 是长字符串，不驻留内存，Python中各自创建了对象来表示 a 和 b，所以他们的值相同但 id 值不同。

试一下当`a=247`,`b=247`时它们的id还是否会相等。事实上Python 为了优化速度，使用了小整数对象池，避免为整数频繁申请和销毁内存空间。而Python 对小整数的定义是 [-5, 257)，只有数字在-5到256之间它们的id才会相等，超过了这个范围就不行了。

```
>>> a = 247
>>> b = 247
>>> print(a is b)
True
>>> a = 258
>>> b = 258
>>> print(a is b)
False
>>>
复制代码
```

> is 是检查两个对象是否指向同一块内存空间，而 == 是检查他们的值是否相等。is比`==`更加严格

47、a=1, b=2, 不用中间变量交换a和b的值?

方法一

```
>>> a = 5
>>> b = 6
>>> a = a+b
>>> b = a-b
>>> a = a-b
```


方法二

```
>>> a = a^b
>>> b = b^a
>>> a = a^b
```

方法三

```
a,b = b,a
```

48、python中生成随机整数、随机小数、0~1之间小数方法?

python中生成随机整数

```
import random
random.randint(1,10)
复制代码
```

随机小数

看自己习惯，可以用random库，也可以用numpy库

```
import random
random.random()

# 利用np.random.randn(5)生成5个随机小数
import numpy as np
np.random.randn(5)
复制代码
```

0~1之间小数

```
random.random()
复制代码
```

49、如何避免转义，给字符串加哪个字母表示原始字符串?

这个就面试题的要点是几个特殊Python3字符串前缀u、b、r

- 无前缀 & u前缀

> 字符串默认创建即以Unicode编码存储，可以存储中文。

```
string = 'a'  等效于  string = u'a'Unicode中通常每个字符由2个字节表示u'a' 即 u'\u0061'  实际内存中为  [0000 0000] [0110 0001]复制代码
```

- b前缀

> 字符串存储为Ascll码，无法存储中文。

- r前缀

> 与上述两种不是一样的东西了。 r前缀就相当于三引号，主要解决的是 转义字符，特殊字符 的问题，其中所有字符均视为普通字符。

所以这道题的正确答案是`r`前缀

50、python中断言方法举例?

> assert 语句，在需要确保程序中的某个条件一定为真才能让程序运行的话就非常有用

下面做一些assert用法的语句供参考：

```
assert 1==1
assert 2+2==2*2
assert len(['my boy',12])<10
assert range(4)==[0,1,2,3]
复制代码
```

这里介绍几个常用断言的使用方法，可以一定程度上帮助大家对预期结果进行判断。

- assertEqual
- assertNotEqual
- assertTrue
- assertFalse
- assertIsNone
- assertIsNotNone

assertEqual 和 assertNotEqual

1. assertEqual：如两个值相等，则pass
2. assertNotEqual：如两个值不相等，则pass

使用方法:

`assertEqual(first,second,msg)` 其中 `first` 与 `second` 进行比较，如果相等则通过； msg为失败时打印的信息，选填； 断言assertNotEqual反着用就可以了。

assertTrue和assertFalse

1. assertTrue：判断bool值为True，则pass
2. assertFalse：判断bool值为False，则Pass

使用方法:

assertTrue(expr,msg)其中express输入相应表达式，如果表达式为真，则pass； msg选填； 断言assertFalse如果表达式为假，则pass

assertIsNone和assertIsNotNone

1. assertIsNone：不存在，则pass
2. assertIsNotNone：存在，则pass

使用方法： assertIsNone(obj,msg)检查某个元素是否存在

51、如何查找一个字符串中特定的字符？find和index的差异?

- 使用find和index方法查找

1、find()方法：查找子字符串，若找到返回从0开始的下标值，若找不到返回-1
2、index()方法：python 的index方法是在字符串里查找子串第一次出现的位置，类似字符串的find方法，不过比find方法更好的是，如果查找不到子串，会抛出异常，而不是返回-1

52、如何在函数中设置一个全局变量?

- 在函数中使用global关键字定义变量

53、可变类型和不可变类型

1,可变类型有list,dict.不可变类型有string，number,tuple.

2,当进行修改操作时，可变类型传递的是内存中的地址，也就是说，直接修改内存中的值，并没有开辟新的内存。

3,不可变类型被改变时，并没有改变原内存地址中的值，而是开辟一块新的内存，将原地址中的值复制过去，对这块新开辟的内存中的值进行操作。

54、Python中变量的作用域？（变量查找顺序)

函数作用域的LEGB顺序

1.什么是LEGB?

L： local 函数内部作用域

E: enclosing 函数内部与内嵌函数之间

G: global 全局作用域

B： build-in 内置作用

python在函数里面的查找分为4种，称之为LEGB，也正是按照这是顺序来查找的

55、请描述抽象类和接口类的区别和联系

1.抽象类： 规定了一系列的方法，并规定了必须由继承类实现的方法。由于有抽象方法的存在，所以抽象类不能实例化。可以将抽象类理解为毛坯房，门窗，墙面的样式由你自己来定，所以抽象类与作为基类的普通类的区别在于约束性更强

2.接口类：与抽象类很相似，表现在接口中定义的方法，必须由引用类实现，但他与抽象类的根本区别在于用途：与不同个体间沟通的规则，你要进宿舍需要有钥匙，这个钥匙就是你与宿舍的接口，你的舍友也有这个接口，所以他也能进入宿舍，你用手机通话，那么手机就是你与他人交流的接口

3.区别和关联：

1.接口是抽象类的变体，接口中所有的方法都是抽象的，而抽象类中可以有非抽象方法，抽象类是声明方法的存在而不去实现它的类

2.接口可以继承，抽象类不行

3.接口定义方法，没有实现的代码，而抽象类可以实现部分方法

4.接口中基本数据类型为static而抽象类不是

56、Python的内存管理机制及调优手段?

内存管理机制: 引用计数、垃圾回收、内存池

引用计数：引用计数是一种非常高效的内存管理手段，当一个Python对象被引用时其引用计数增加1，

当其不再被一个变量引用时则计数减1,当引用计数等于0时对象被删除。弱引用不会增加引用计数

垃圾回收：

1.引用计数

引用计数也是一种垃圾收集机制，而且也是一种最直观、最简单的垃圾收集技术。当Python的某个对象的引用计数降为0时，说明没有任何引用指向该对象，该对象就成为要被回收的垃圾了。比如某个新建对象，它被分配给某个引用，对象的引用计数变为1，如果引用被删除，对象的引用计数为0,那么该对象就可以被垃圾回收。不过如果出现循环引用的话，引用计数机制就不再起有效的作用了。

2.标记清除

https://foofish.net/python-gc.html

调优手段

1.手动垃圾回收

2.调高垃圾回收阈值

3.避免循环引用

57、内存泄露是什么？如何避免？

内存泄漏指由于疏忽或错误造成程序未能释放已经不再使用的内存。内存泄漏并非指内存在物理上的消失，而是应用程序分配某段内存后，由于设计错误，导致在释放该段内存之前就失去了对该段内存的控制，从而造成了内存的浪费。

有 __del__() 函数的对象间的循环引用是导致内存泄露的主凶。不使用一个对象时使用: del object 来删除一个对象的引用计数就可以有效防止内存泄露问题。

通过Python扩展模块gc 来查看不能回收的对象的详细信息。

可以通过 sys.getrefcount(obj) 来获取对象的引用计数，并根据返回值是否为0来判断是否内存泄露

58、编写函数的4个原则

1.函数设计要尽量短小

2.函数声明要做到合理、简单、易于使用

3.函数参数设计应该考虑向下兼容

4.一个函数只做一件事情，尽量保证函数语句粒度的一致性

59、函数调用参数的传递方式是值传递还是引用传递？

Python的参数传递有：位置参数、默认参数、可变参数、关键字参数。

函数的传值到底是值传递还是引用传递、要分情况：

不可变参数用值传递：像整数和字符串这样的不可变对象，是通过拷贝进行传递的，因为你无论如何都不可能在原处改变不可变对象。

可变参数是引用传递：比如像列表，字典这样的对象是通过引用传递、和C语言里面的用指针传递数组很相似，可变对象能在函数内部改变。

60、对缺省参数的理解？

缺省参数指在调用函数的时候没有传入参数的情况下，调用默认的参数，在调用函数的同时赋值时，所传入的参数会替代默认参数。

*args是不定长参数，它可以表示输入参数是不确定的，可以是任意多个。

**kwargs是关键字参数，赋值的时候是以键值对的方式，参数可以是任意多对在定义函数的时候

不确定会有多少参数会传入时，就可以使用两个参数

61、hasattr() getattr() setattr() 函数使用详解?

hasattr(object,name)函数:

判断一个对象里面是否有name属性或者name方法，返回bool值，有name属性（方法）返回True，否则返回False。

```
class function_demo(object):
    name = 'demo'
    def run(self):
        return "hello function"
functiondemo = function_demo()
res = hasattr(functiondemo, "name") # 判断对象是否有name属性，True
res = hasattr(functiondemo, "run") # 判断对象是否有run方法，True
res = hasattr(functiondemo, "age") # 判断对象是否有age属性，False
print(res)
```

getattr(object, name[,default])函数:

获取对象object的属性或者方法，如果存在则打印出来，如果不存在，打印默认值，默认值可选。注意：如果返回的是对象的方法，则打印结果是：方法的内存地址，如果需要运行这个方法，可以在后面添加括号().

```
functiondemo = function_demo()
getattr(functiondemo, "name")# 获取name属性，存在就打印出来 --- demo
getattr(functiondemo, "run") # 获取run 方法，存在打印出方法的内存地址
getattr(functiondemo, "age") # 获取不存在的属性，报错
getattr(functiondemo, "age", 18)# 获取不存在的属性，返回一个默认值
```

setattr(object, name, values)函数:

给对象的属性赋值，若属性不存在，先创建再赋值

```
class function_demo(object):
    name = "demo"
    def run(self):
        return "hello function"
functiondemo = function_demo()
res = hasattr(functiondemo, "age") # 判断age属性是否存在，False
print(res)
setattr(functiondemo, "age", 18) # 对age属性进行赋值，无返回值
res1 = hasattr(functiondemo, "age") # 再次判断属性是否存在，True
```

综合使用

```
class function_demo(object):
    name = "demo"
    def run(self):
        return "hello function"
functiondemo = function_demo()
res = hasattr(functiondemo, "addr") # 先判断是否存在
if res:
    addr = getattr(functiondemo, "addr")
    print(addr)
else:
    addr = getattr(functiondemo, "addr", setattr(functiondemo, "addr", "北京首都"))
    print(addr)
```

###

练习题

1、1行代码实现1到100的和?

分析：这题考察的是对Python内置函数的了解程度

Python常见的内置函数有

Built-in Functions				
abs()	delattr()	hash()	memoryview()	set()
all()	dict()	help()	min()	setattr()
any()	dir()	hex()	next()	slice()
ascii()	divmod()	id()	object()	sorted()
bin()	enumerate()	input()	oct()	staticmethod()
bool()	eval()	int()	open()	str()
breakpoint()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()
complex()	hasattr()	max()	round()	

官方查询手册如下 docs.python.org/3/library/f...

图片中我框选的是比较常用的一些，你可能见过，这题考察的是sum也就是求和 具体的使用

sum(iterable[, start])

1. iterable -- 可迭代对象，如：列表、元组、集合。
2. start -- 指定相加的参数，如果没有设置这个值，默认为0。

例如

```
sum([1,2,3]) # 结果为6
sum([1,2,3],5) # 结果为11
```

python一行代码如何实现1~100的和

还要用到第二个内置函数 range()

range(start, stop[, step])

1. start: 计数从 start 开始。默认是从 0 开始。例如range（5）等价于range（0， 5）；
2. stop: 计数到 stop 结束，但不包括 stop。例如：range（0， 5） 是[0, 1, 2, 3, 4]没有5
3. step：步长，默认为1。例如：range（0， 5） 等价于 range(0, 5, 1)

解答：

```
sum(range(1,101))
```

2、列出几个python标准库

你先明确的是什么是Python标准库

> Python标准库(standard library)。 标准库会随着Python解释器，一起安装在你的电脑中的。它是Python的一个组成部分。 这些标准库是Python为你准备好的利器，可以让编程事半功倍。

文档手册可以查阅 > docs.python.org/zh-cn/3.7/l...

了解这个内容，这道题回答起来就非常简单了

1. os模块
2. re模块
3. pickle 模块
4. datetime模块
5. time模块
6. math模块

3、下面Python代码的运行结果是?

这种题目，考察的是代码默读能力

```
def f(x,l=[]):
    for i in range(x):
        l.append(i*i)
    print(l)
f(2)
f(3,[3,2,1])
f(3)
```

- f(2)

```
def f(2,l=[]):
    for i in range(2): # i=0,1
        l.append(i*i) # [0,1]
    print(l)
```

- f(3,[3,2,1])

```
def f(3,l=[3,2,1]):
    for i in range(3): # i=0,1,2
        l.append(i*i) # [3,2,1,0,1,4]
    print(l)
```

- f(3)

```
def f(3,l=[]):
    for i in range(3): # i=0,1,2
        l.append(i*i) # [0,1,4] ???对吗?
    print(l)
```

这个地方，你需要避免踩坑，一定要注意列表是可变的，如果单独的写没有任何问题，但是函数调用的三行代码放在一起就有点意思了

`f(3,[3,2,1])` 将l进行了重新赋值。但是第三次调用函数使用的依旧是第一次的l,所以避免踩坑哦~~~~

f(3)运行的正确结果是 `[0,1,0,1,4]`

```
In [10]: def f(x,l=[]):
             for i in range(x):
                 l.append(i*i)
         print(l)
         f(2)
         f(3,[3,2,1])
         f(3)
```

```
[0, 1]
[3, 2, 1, 0, 1, 4]
[0, 1, 0, 1, 4]
```

4、python实现列表去重的方法?

简单直接的办法，集合里面的元素不可以重复

```
my_list = [1,1,2,2,3,3,5,6,7,88]
my_set = set(my_list)
my_list = [x for x in my_set]
my_list
```

循环判断去重

```
ids = [1,1,2,2,3,3,5,6,7,88]
news_ids = []
for id in ids:
    if id not in news_ids:
        news_ids.append(id)
print(news_ids)
```

字典的fromkeys方法实现

```
my_list=[1,1,2,2,3,3,5,6,7,88]
d = {}.fromkeys(my_list)
print(d.keys())
```

5、写一个函数，将两个dict（key是数字，value是string）进行合并，函数返回合并后的dict。

> 规则如下：如果一个key仅仅存在于其中一个dict中，则直接加入合并后的dict； 如果一个key在两个dict中都存在，那么给定一个choice值，choice可以是任何string，如果choice是任一个dict中的value，则写入，否则不写入。

代码如下

```
def function(lefdict,rightdict,choice):
    samelist=lefdict.keys() & rightdict.keys()  # dict.keys()返回的是一个可迭代对象，取两个dict的keys的交集
    diflist = lefdict.keys() ^ rightdict.keys() # 取两个dict的keys的不同集
    newdict={}
    for key,value in lefdict.items():
        if key in diflist:
            newdict[key]=value
        elif key in samelist:
            if value==choice:
                newdict[key]=value
    for key,value in rightdict.items():
        if key in diflist:
            newdict[key]=value
        elif key in samelist:
            if value==choice:
                newdict[key]=value
    print(samelist,diflist)
    print(newdict)
    return newdict
```

```
function({1:'a',2:'b',3:'c'},{4:'f',2:'b',3:'d'},'b')
```

6、把一个字符串的尾字母移到首位，比如‘abcde’->‘eabcd’

> 把一个字符串的尾字母移到首位，比如‘abcde’->‘eabcd’，称为一次字符串的旋转。如果字符串1的任何一次旋转可以包含字符串2，则返回true，否则返回false，请写一个函数实现。

代码如下

```
def function(str1, str2):
    str1 = str1[-1] + str1[:-1]
    print(str1)
    if str1.find(str2) != -1:
        return True
    else:
        return False

print(function("abcde", "cde"))
```

7、 修改以下Python代码，使得下面的代码调用类A的show方法?

原始代码

```

class A(object):
    def run(self):
        print("基础 run 方法")

class B(A):
    def run(self):
        print("衍生 run 方法 ")

obj = B()
obj.run()
```

面试要点：

> 类继承，只要通过`__class__`方法指定类对象就可以了。

修改代码

```
class A(object):
    def run(self):
        print("基础 run 方法")

class B(A):
    def run(self):
        print("衍生 run 方法 ")

obj = B()
obj.__class__ = A
obj.run()
```

8、修改以下Python代码，使得代码能够运行

原始代码

```
class A(object):
    def __init__(self,a,b):
        self.__a = a
        self.__b = b

    def show(self):
        print("a=",self.__a,"b=",self.__b)

a = A(5,10)
a.show()
a(20)
```

面试要点：

> 是方法对象，为了能让对象实例能被直接调用，需要实现`__call__`方法

修改代码

```
class A(object):
    def __init__(self,a,b):
        self.__a = a
        self.__b = b

    def show(self):
        print("a=",self.__a,"b=",self.__b)

    def __call__(self, num):
        print("call:",num + self.__a)

a = A(5,10)
a.show()
a(20)
```

9、如何添加代码，使得没有定义的方法都调用myfunc方法?

原始代码

```
class A(object):
    def __init__(self,a,b):
        self.a1 = a
        self.b1 = b
        print("初始化方法")

    def myfunc(self):
        print("myfunc")

a1 = A(10,20)
a1.fn1()
a1.fn2()
a1.fn3()
```

修改代码

```
class A(object):
    def __init__(self,a,b):
        self.a1 = a
        self.b1 = b
        print("初始化方法")

    def myfunc(self):
        print("myfunc")

    def __getattr__(self, item):
        return self.myfunc

a1 = A(10,20)
a1.fn1()
a1.fn2()
a1.fn3()
```

> 考点 python的默认方法， 只有当没有定义的方法调用时，才会调用方法 `__getattr__`。 当 fn1 方法传入参数时，我们可以给 myfunc方法增加一个 *args 不定参数来兼容。

10、 如何用 Python 来发送邮件?

smtplib 标准库

可以参考菜鸟教程:www.runoob.com/python/pyth...

11、 4G 内存怎么读取一个 5G 的数据?

方法一

可以通过生成器，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。

```
def get_lines():  # 生成器
    with open('big.data', 'r') as f:
        while True:
            data = f.readlines(100)
            if data:
                yield data
            else:
                break

f = get_lines()  # 迭代器对象
print(next(f))
print(next(f))
print(next(f))
```

复制代码

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。

11、在Python中输入某年某月某日，判断这一天是这一年的第几天？(可以用 Python 标准库)

方法一

```
year = int(input('请输入4位数字的年份：'))  #获取年份
month = int(input('请输入月份：'))  #获取月份
day = int(input('请输入是哪一天：'))  #获取日
if month == 1:
    count = day
elif month == 2:
    count = 31 + day
elif (month >= 3) and ((year % 4 == 0 and year % 100 != 0) or year % 400 == 0):
    if month == 3:
        count = 31 + 29 + day
    if month == 4:
        count = 31 + 29 + 31 + day
    if month == 5:
        count = 31 + 29 + 31 + 30 + day
    if month == 6:
        count = 31 + 29 + 31 + 30 + 31 + day
    if month == 7:
        count = 31 + 29 + 31 + 30 + 31 + 30 + day
    if month == 8:
        count = 31 + 29 + 31 + 30 + 31 + 30 + 31 + day
    if month == 9:
        count = 31 + 29 + 31 + 30 + 31 + 30 + 31 + 31 + day
    if month == 10:
        count = 31 + 29 + 31 + 30 + 31 + 30 + 31 + 31 + 30 + day
```

```
    if month == 11:
        count = 31 + 29 + 31 + 30 + 31 + 30 + 31 + 31 + 30 + 31 + day
    if month == 12:
        count = 31 + 29 + 31 + 30 + 31 + 30 + 31 + 31 + 30 + 31 + 30 + day

else:
    if month == 3:
        count = 31 + 28 + day
    if month == 4:
        count = 31 + 28 + 31 + day
    if month == 5:
        count = 31 + 28 + 31 + 30 + day
    if month == 6:
        count = 31 + 28 + 31 + 30 + 31 + day
    if month == 7:
        count = 31 + 28 + 31 + 30 + 31 + 30 + day
    if month == 8:
        count = 31 + 28 + 31 + 30 + 31 + 30 + 31 + day
    if month == 9:
        count = 31 + 28 + 31 + 30 + 31 + 30 + 31 + 31 + day
    if month == 10:
        count = 31 + 28 + 31 + 30 + 31 + 30 + 31 + 31 + 30 + day
    if month == 11:
        count = 31 + 28 + 31 + 30 + 31 + 30 + 31 + 31 + 30 + 31 + day
    if month == 12:
        count = 31 + 28 + 31 + 30 + 31 + 30 + 31 + 31 + 30 + 31 + 30 + day

print('第' + str(count) + '天')
```

互联网找到了几个解法

方法二

```
import datetime

y = int(input('请输入4位数字的年份：'))  # 获取年份
m = int(input('请输入月份：'))  # 获取月份
d = int(input('请输入是哪一天：'))  # 获取“日”

targetDay = datetime.date(y, m, d)  # 将输入的日期格式化成标准的日期
dayCount = targetDay - datetime.date(targetDay.year - 1, 12, 31)  # 减去上一年最后一天
print('%s是%s年的第%s天。' % (targetDay, y, dayCount.days))
```

方法三

```
import datetime
dtime = input("请输入求天数的日期(20191111)：")
tnum = datetime.datetime.strptime(dtime,'%Y%m%d').strftime("%j")
print(dtime + "在一年中的天数是: " + tnum + "天。")
```

13、列表[1,2,3,4,5],请使用map()函数输出[1,4,9,16,25]，并使用列表推导式提取出大于10的数，最终输出[16,25]。

map是python高阶用法，字面意义是映射，它的作用就是把一个数据结构映射成另外一种数据结构。

map用法比较绕，最好是对基础数据结构很熟悉了再使用，比如列表，字典，序列化这些。

map的基本语法如下：

```
map(函数, 序列1, 序列2, ...)
复制代码
```

> Python 2.x 返回列表。 Python 3.x 返回迭代器。

```
list = [1,2,3,4,5]
def fn(x):
    return x ** 2

res = map(fn,list)
res = [i for i in res]
print(res)

res = [i for i in res if i > 10]
print(res)
```

14、设计一个函数返回给定文件名的后缀?

考察字符串操作

1. rfind() # 右侧字符出现的位置
2. 注意下面的0<pos<2 用法
3. if ... else用法

```
def get_suffix(filename, has_dot=False):
    """
    获取文件名的后缀名

    :param filename: 文件名
    :param has_dot: 返回的后缀名是否需要带点

    :return: 文件的后缀名
    """
    pos = filename.rfind('.')
    if 0 < pos < len(filename) - 1:
        index = pos if has_dot else pos + 1
        return filename[index:]
    else:
        return ''
```

15、 这两个参数是什么意思：*args，**kwargs？我们为什么要使用它们?

1. 如果我们不确定要往函数中传入多少个参数，或者我们想往函数中以`列表`和`元组`的形式传参数时，那就使要用*args；

2. 如果我们不知道要往函数中传入多少个关键词参数，或者想传入`字典`的值作为关键词参数时，那就要使用**kwargs。
3. args和kwargs这两个标识符是约定俗成的用法，你当然还可以用*tom和**jarry，但是这样显的不专业。

下面是具体的示例：`案例来源互联网搜索，都书写一遍即可掌握`

```
def f(*args,**kwargs):
    print(args, kwargs)

l = [1,2,3]
t = (4,5,6)
d = {'a':7,'b':8,'c':9}

f()
f(1,2,3)                    # (1, 2, 3) {}
f(1,2,3,"groovy")           # (1, 2, 3, 'groovy') {}
f(a=1,b=2,c=3)              # () {'a': 1, 'c': 3, 'b': 2}
f(a=1,b=2,c=3,zzz="hi")     # () {'a': 1, 'c': 3, 'b': 2, 'zzz': 'hi'}
f(1,2,3,a=1,b=2,c=3)        # (1, 2, 3) {'a': 1, 'c': 3, 'b': 2}

f(*l,**d)                   # (1, 2, 3) {'a': 7, 'c': 9, 'b': 8}
f(*t,**d)                   # (4, 5, 6) {'a': 7, 'c': 9, 'b': 8}
f(1,2,*t)                   # (1, 2, 4, 5, 6) {}
f(q="winning",**d)          # () {'a': 7, 'q': 'winning', 'c': 9, 'b': 8}
f(1,2,*t,q="winning",**d)   # (1, 2, 4, 5, 6) {'a': 7, 'q': 'winning', 'c': 9,
'b': 8}

def f2(arg1,arg2,*args,**kwargs):
    print(arg1,arg2, args, kwargs)

f2(1,2,3)                   # 1 2 (3,) {}
f2(1,2,3,"groovy")          # 1 2 (3, 'groovy') {}
f2(arg1=1,arg2=2,c=3)       # 1 2 () {'c': 3}
f2(arg1=1,arg2=2,c=3,zzz="hi")  # 1 2 () {'c': 3, 'zzz': 'hi'}
f2(1,2,3,a=1,b=2,c=3)       # 1 2 (3,) {'a': 1, 'c': 3, 'b': 2}

f2(*l,**d)                  # 1 2 (3,) {'a': 7, 'c': 9, 'b': 8}
f2(*t,**d)                  # 4 5 (6,) {'a': 7, 'c': 9, 'b': 8}
f2(1,2,*t)                  # 1 2 (4, 5, 6) {}
f2(1,1,q="winning",**d)     # 1 1 () {'a': 7, 'q': 'winning', 'c': 9, 'b': 8}
f2(1,2,*t,q="winning",**d)  # 1 2 (4, 5, 6) {'a': 7, 'q': 'winning', 'c': 9,
'b': 8}
```

16、 求出`0~n`的所有正整数中数字k（0~9）出现的次数。

举例

> 例如：k=1，n=12，那么 1 在 [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]一共出现5次[1,10,11,12] 输入：k=1，n=12 输出：5

解答思路：

> 统计数字 1 在 [1,10,11,12]出现的次数这非常像Python中统计字符串a在字符串b中出现的次数：b.count(a) 所以我们将把数字转为字符串来做统计。

```
def digit_count(k,n):
    listn = []
    count = 0
    for i in range(0,n+1):
        count += str(i).count(str(k))
        if str(k) in str(i):
            listn.append(str(i))

    return count,listn

c,ls = digit_count(1,12)
print(c,ls)
```

17、阅读下面的代码，默读出A0，A1至An的最终值。

```
A0 = dict(zip(('a','b','c','d','e'),(1,2,3,4,5)))
A1 = range(10)
A2 = [i for i in A1 if i in A0]
A3 = [A0[s] for s in A0]
A4 = [i for i in A1 if i in A3]
A5 = {i:i*i for i in A1}
A6 = [[i,i*i] for i in A1]
```

默读代码类的题目，相对来说是比较简单的。重点去研究列表解析，之后你就可以轻松的回答这些问题喽~

```
A0 = {'a': 1, 'c': 3, 'b': 2, 'e': 5, 'd': 4}
A1 = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
A2 = []
A3 = [1, 3, 2, 5, 4]
A4 = [1, 2, 3, 4, 5]
A5 = {0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81}
A6 = [[0, 0], [1, 1], [2, 4], [3, 9], [4, 16], [5, 25], [6, 36], [7, 49], [8,
64], [9, 81]]
```

18、 list = ['a','a','a',1,2,3,4,5,'A','B','C']提取出”12345”?

这个考点考了python的解压赋值的知识点，即 a,b,c,*middle,d,e,f = list, *middle = [1,2,3,4,5]。

注意，解压赋值提取出来的是列表

```
list = ['a','a','a',1,2,3,4,5,'A','B','C']
a,b,c,*middle,d,e,f = list
print(middle)
print(type(middle))
```

19、如何用Python删除一个文件?

> os模块的使用

`os.remove(path)`

删除文件 path，删除时候如果path是一个目录， 抛出 OSError错误。如果要删除目录，请使用 rmdir()。

remove() 同 unlink() 的功能是一样的

```
os.remove('a.txt')
```

`os.removedirs(path)`

递归地删除目录。类似于`rmdir()`, 如果子目录被成功删除， `removedirs()` 将会删除父目录；但子目录没有成功删除，将抛出错误。

例如， `os.removedirs("a/b/c")` 将首先删除 `c` 目录，然后再删除 `b` 和 `a`, 如果他们是空的话，则子目录不能成功删除，将抛出 `OSError异常`

`os.rmdir(path)` 删除目录 path，要求path必须是个`空目录`，否则抛出OSError错误

20、写一个函数, 输入一个字符串, 返回倒序排列的结果?

使用字符串本身的翻转

```
def order_by(str):
    return str[::-1]

print(order_by('123456'))

输出：654321
```

把字符串变为列表，用列表的reverse函数

```
def reverse2(text='abcdef'):
    new_text=list(text)
    new_text.reverse()
    return ''.join(new_text)

reverse2('abcdef')
```

新建一个列表，从后往前取

```
def reverse3(text='abcdef'):
    new_text=[]
    for i in range(1,len(text)+1):
        new_text.append(text[-i])
    return ''.join(new_text)

reverse3('abcdef')
```

利用双向列表deque中的extendleft函数

```
from collections import deque
def reverse4(text='abcdef'):
    d = deque()
    d.extendleft(text)
    return ''.join(d)

reverse4('abcdef')
```

21、在Python中读取大文件

- 利用生成器generator

```
def read_in_block(file_path):
    BLOCK_SIZE = 1024
    with open(file_path, "r") as f:
        while True:
            block = f.read(BLOCK_SIZE)  # 每次读取固定长度到内存缓冲区
            if block:
                yield block
            else:
                return  # 如果读取到文件末尾，则退出

def test3():
    file_path = "/tmp/test.log"
    for block in read_in_block(file_path):
        print block
```

- 迭代器进行迭代遍历：for line in file

```
def test4():
    with open("/tmp/test.log") as f:
        for line in f:
            print line
```

for line in f 这种用法是把文件`对象f`当作迭代对象， 系统将自动处理IO缓冲和内存管理， 这种方法是更加`pythonic`的方法。 比较简洁。

> Pythonic追求的是对Python语法的充分发挥，写出的代码带Python味儿，而不是看着向C或JAVA

22、 元组的解封装是什么?

首先我们来看解封装：

```
>>> mytuple=3,4,5
>>> mytuple
(3, 4, 5)
```

这将 3，4，5 封装到元组 mytuple 中。

现在我们将这些值解封装到变量 x，y，z 中：

```
>>> x,y,z=mytuple
>>> x+y+z
```

得到结果12.

23、把a='aaabbcccdddde'这种形式的字符串，压缩成a3b2c3d4e1这种形式。

```
a='aaabbcccddddde'
aa=''
for i in sorted(list(set(a)),key=a.index):
aa=aa+i+str(a.count(i))
print(aa)
```

24、一个数如果恰好等于它的因子之和，这个数就称为‘完数’，比如6=1+2+3，编程找出1000以内的所有的完数。

```
wanshu=[]
for i in range(1,1001):
s=0
for j in range(1,i//2+1):
if i % j ==0:
s+=j
else:
if i==s:
wanshu.append(i)
print(wanshu)
```

25、输入一个字符串，输出该字符串的字符的所有组合。如输入'abc',输出a,b,c,ab,ac,bc,abc.

```
def getC(s):
if not s:
return
len_s=len(s)
ss=[]
for i in range(len_s):
combination(s,0,i,ss)
aaa=[]
def combination(s,index,num,ss):
global aaa
if num==-1:
return
if index==len(s):
```

```
return
ss.append(s[index])
aaa.append(''.join(ss))
combination(s,index+1,num-1,ss)
ss.pop()
combination(s,index+1,num,ss)

getC('123')
print(aaa)
print(sorted(set(aaa),key=lambda x:len(str(x))))
```

26、给定一个非空的字符串，判断它是否可以由它的一个子串重复多次构成。给定的字符串只含有小写英文字母，并且长度不超过10000。例如：'ababab',返回True，'ababa'，返回False

```
def solution(s):
ll=len(s)
for i in range(1,ll//2+1):
if ll % i == 0:
j=0
while s[:i]==s[j:j+i] and j<ll:
    j=j+i
if j==ll:
    return True
return False

print(solution('abababa'))
```

27、filter、map、reduce的作用。

1、filter() 相当于过滤器的作用

```
s=[1,2,3,5,6,8,9,10,25,12,30]
# 筛选出3的倍数
# 第一个参数为一个返回True或者False的函数，第二个参数为可迭代对象
# 该函数把可迭代对象依次传入第一个函数，如果为True，则筛选
d=filter(lambda x:True if x % 3 == 0 else False,s)
print(list(d))
```

2、map()函数，

```
# 第一个参数为函数，依次将后面的参数传给第一个函数，并执行函数
# 如果有多个参数则，依次将后面的对应传给参数
s=map(lambda x,y:x+y,range(10),range(10))
print(list(s))
ss=map(lambda x:x*x,range(10))
print(list(ss))
```

3、reduce()函数

```
from functools import reduce
# 开始的时候将可迭代对象的第一个数和第二个数当成x和y
# 然后将第一次函数的执行结果当成x，然后再传入一个数当成y
# 再执行函数
s=reduce(lambda x,y:x+y,range(101))
print(s) # 相当于0+1+2+……+99+100
```

###

28、使用生成器编写一个函数实现生成指定个数的斐波那契数列

```
def fib2(imax):
t,a,b=0,0,1
while t<imax:
yield b
a,b=b,a+b
t+=1

for i in fib2(10):
print(i)
```

29、一行代码通过filter和lambda函数输出alist=[1,22,2,33,23,32]中索引为奇数的值

```
alist=[1,22,2,33,23,32]
ss=[x[1] for x in filter(lambda x:x[0]%2==1,enumerate(alist))]
print(ss)
```

30、编写一个函数实现十进制转62进制，分别用0-9A-Za-z,表示62位字母

```
import string
print(string.ascii_lowercase) # 小写字母
print(string.ascii_uppercase) # 大写字母
print(string.digits) # 0-9

s=string.digits+string.ascii_uppercase+string.ascii_lowercase
def _10_to_62(num):
ss=''
while True:
ss=s[num%62]+ss
if num//62==0:
break
num=num//62
return ss
print(_10_to_62(65))
```

31、实现一个装饰器，限制该函数被调用的频率，如10秒一次

```
import time
from functools import wraps
def dec(func):
key=func.__name__
cache={key:None}
@wraps(func)
def inner(*args,**kwargs):
result=None
if cache.get(key) is None:
cache[key]=time.time()
result=func(*args,**kwargs)
print('执行函数中')
else:
now=time.time()
if now-cache[key]>10:
    cache[key]=now
    result=func(*args,**kwargs)
    print('执行函数中')
else:
    print('函数执行受限')
return result
return inner

@dec
def add(x,y):
print(x+y)

add(1,2)
add(1,3)
time.sleep(10)
add(3,4)
```

32、实现一个装饰器，通过一次调用，使函数重复执行5次

```
from functools import wraps
def dec(func):
@wraps(func)
def inner(*args,**kwargs):
result=[func(*args,**kwargs) for i in range(5)]
return result
return inner

@dec
def add(x,y):
return x+y
print(add(1,2))
```

33、编写一个函数，找出数组中没有重复的值的和

```
def func(lis):
lis1=[]
del_lis=[]
for i in lis:
if i not in lis1:
if i not in del_lis:
    lis1.append(i)
else:
del_lis.append(i)
lis1.remove(i)
return sum(lis1)

def func2(lis):
return sum([i for i in set(lis) if lis.count(i)==1])

print(func2([3,4,1,2,5,6,6,5,4,3,3]))
```

19.设计实现遍历目录与子目录，抓取.pyc文件

第一种方法:

```
import os

def get_files(dir,suffix):
    res = []
    for root,dirs,files in os.walk(dir):
        for filename in files:
            name,suf = os.path.splitext(filename)
            if suf == suffix:
                res.append(os.path.join(root,filename))

    print(res)

get_files("./",'.pyc')
```

第二种方法:

```
import os

def pick(obj):
    if obj.endswith(".pyc"):
        print(obj)

def scan_path(ph):
    file_list = os.listdir(ph)
    for obj in file_list:
        if os.path.isfile(obj):
            pick(obj)
        elif os.path.isdir(obj):
            scan_path(obj)

if __name__=='__main__':
    path = input('输入目录')
    scan_path(path)
```

第三种方法

```
from glob import iglob


def func(fp, postfix):
    for i in iglob(f"{fp}/**/*{postfix}", recursive=True):
        print(i)

if __name__ == "__main__":
    postfix = ".pyc"
    func("K:\Python_script", postfix)
```

20、字符串 "123" 转换成 123，不使用内置api，例如 int()

方法一： 利用 str 函数

```
def atoi(s):
    num = 0
    for v in s:
        for j in range(10):
            if v == str(j):
                num = num * 10 + j
    return num
```

方法二： 利用 ord 函数

```
def atoi(s):
    num = 0
    for v in s:
        num = num * 10 + ord(v) - ord('0')
    return num
```

方法三: 利用 eval 函数

```
def atoi(s):
    num = 0
    for v in s:
        t = "%s * 1" % v
        n = eval(t)
        num = num * 10 + n
    return num
```

方法四: 结合方法二，使用 reduce，一行解决

```
from functools import reduce
def atoi(s):
    return reduce(lambda num, v: num * 10 + ord(v) - ord('0'), s, 0)
```

21、python代码实现删除一个list里面的重复元素

```
def distFunc1(a):
    """使用集合去重"""
```

```
    a = list(set(a))
    print(a)

def distFunc2(a):
    """将一个列表的数据取出放到另一个列表中，中间作判断"""
    list = []
    for i in a:
        if i not in list:
            list.append(i)
    #如果需要排序的话用sort
    list.sort()
    print(list)

def distFunc3(a):
    """使用字典"""
    b = {}
    b = b.fromkeys(a)
    c = list(b.keys())
    print(c)

if __name__ == "__main__":
    a = [1,2,4,2,4,5,7,10,5,5,7,8,9,0,3]
    distFunc1(a)
    distFunc2(a)
    distFunc3(a)
```

###

22、统计一个文本中单词频次最高的10个单词?

```
import re

# 方法一
def test(filepath):

    distone = {}

    with open(filepath) as f:
        for line in f:
            line = re.sub("\W+", " ", line)
            lineone = line.split()
            for keyone in lineone:
                if not distone.get(keyone):
                    distone[keyone] = 1
                else:
                    distone[keyone] += 1
    num_ten = sorted(distone.items(), key=lambda x:x[1], reverse=True)[:10]
    num_ten =[x[0] for x in num_ten]
    return num_ten


# 方法二
# 使用 built-in 的 Counter 里面的 most_common
import re
from collections import Counter
```

```
def test2(filepath):
    with open(filepath) as f:
        return list(map(lambda c: c[0], Counter(re.sub("\W+", " ",
f.read()).split()).most_common(10)))
```

23、请写出一个函数满足以下条件

该函数的输入是一个仅包含数字的list,输出一个新的list，其中每一个元素要满足以下条件：

1、该元素是偶数

2、该元素在原list中是在偶数的位置(index是偶数)

```
def num_list(num):
    return [i for i in num if i %2 ==0 and num.index(i)%2==0]

num = [0,1,2,3,4,5,6,7,8,9,10]
result = num_list(num)
print(result)
```

###

24、两个有序列表，l1,l2，对这两个列表进行合并不可使用extend

```
def loop_merge_sort(l1,l2):
    tmp = []
    while len(l1)>0 and len(l2)>0:
        if l1[0] <l2[0]:
            tmp.append(l1[0])
            del l1[0]
        else:
            tmp.append(l2[0])
            del l2[0]
    while len(l1)>0:
        tmp.append(l1[0])
        del l1[0]
    while len(l2)>0:
        tmp.append(l2[0])
        del l2[0]
    return tmp
```


25、给定一个任意长度数组，实现一个函数

让所有奇数都在偶数前面，而且奇数升序排列，偶数降序排序，如字符串'1982376455',变成'1355798642'

```
# 方法一
def func1(l):
    if isinstance(l, str):
        l = [int(i) for i in l]
    l.sort(reverse=True)
    for i in range(len(l)):
        if l[i] % 2 > 0:
            l.insert(0, l.pop(i))
    print(''.join(str(e) for e in l))

# 方法二
def func2(l):
    print("".join(sorted(l, key=lambda x: int(x) % 2 == 0 and 20 - int(x) or
int(x))))
```

26、写一个函数找出一个整数数组中，第二大的数

```
def find_second_large_num(num_list):
    """
    找出数组第2大的数字
    """
    # 方法一
    # 直接排序，输出倒数第二个数即可
    tmp_list = sorted(num_list)
    print("方法一\nSecond_large_num is :", tmp_list[-2])

    # 方法二
    # 设置两个标志位一个存储最大数一个存储次大数
    # two 存储次大值，one 存储最大值，遍历一次数组即可，先判断是否大于 one，若大于将 one 的
值给 two，将 num_list[i] 的值给 one，否则比较是否大于two，若大于直接将 num_list[i] 的值给
two，否则pass
    one = num_list[0]
    two = num_list[0]
    for i in range(1, len(num_list)):
        if num_list[i] > one:
            two = one
            one = num_list[i]
        elif num_list[i] > two:
            two = num_list[i]
    print("方法二\nSecond_large_num is :", two)

    # 方法三
    # 用 reduce 与逻辑符号 (and, or)
    # 基本思路与方法二一样，但是不需要用 if 进行判断。
    from functools import reduce
    num = reduce(lambda ot, x: ot[1] < x and (ot[1], x) or ot[0] < x and (x,
ot[1]) or ot, num_list, (0, 0))[0]
    print("方法三\nSecond_large_num is :", num)


if __name__ == '__main__':
    num_list = [34, 11, 23, 56, 78, 0, 9, 12, 3, 7, 5]
    find_second_large_num(num_list)
```

27、阅读一下代码他们的输出结果是什么?

```
def multi():
    return [lambda x : i*x for i in range(4)]
print([m(3) for m in multi()])
```

正确答案是[9,9,9,9]，而不是[0,3,6,9]产生的原因是Python的闭包的后期绑定导致的，这意味着在闭包中的变量是在内部函数被调用的时候被查找的，因为，最后函数被调用的时候，for循环已经完成, i 的值最后是3,因此每一个返回值的i都是3,所以最后的结果是[9,9,9,9]

28、遍历一个object的所有属性，并print每一个属性名?

```
class Car:
    def __init__(self,name,loss): # loss [价格，油耗，公里数]
        self.name = name
        self.loss = loss

    def getName(self):
        return self.name

    def getPrice(self):
        # 获取汽车价格
        return self.loss[0]

    def getLoss(self):
        # 获取汽车损耗值
        return self.loss[1] * self.loss[2]

Bmw = Car("宝马",[60,9,500]) # 实例化一个宝马车对象
print(getattr(Bmw,"name")) # 使用getattr()传入对象名字,属性值。
print(dir(Bmw)) # 获Bmw所有的属性和方法
```

29、手写一个判断时间的装饰器

```
import datetime


class TimeException(Exception):
    def __init__(self, exception_info):
        super().__init__()
        self.info = exception_info

    def __str__(self):
        return self.info


def timecheck(func):
    def wrapper(*args, **kwargs):
        if datetime.datetime.now().year == 2019:
            func(*args, **kwargs)
        else:
```

```
            raise TimeException("函数已过时")

    return wrapper


@timecheck
def test(name):
    print("Hello {}, 2019 Happy".format(name))


if __name__ == "__main__":
    test("backbp")
```

爬虫篇

1、 HTTPS和HTTP的区别：

- https协议要申请证书到ca，需要一定经济成本
- http是明文传输，https是加密的安全传输
- 连接的端口不一样，http是80，https是443
- http连接很简单，没有状态；https是ssl加密的传输，身份认证的网络协议，相对http传输比较安全。

还有很多，自己去整理一下吧

2、 如何解决验证码的问题，用什么模块，听过哪些人工打码平台?

PIL、pytesser、tesseract模块

平台的话有：（打码平台特殊，不保证时效性）

- 云打码
- 挣码
- 斐斐打码
- 若快打码
- 超级鹰

3、 ip 被封了怎么解决，自己做过 ip 池么?

关于 ip 可以通过 ip 代理池来解决问题 ip 代理池相关的可以在 github 上搜索 ip proxy 自己选一个 去说 github.com/awolfly9/IP... 提供大体思路：

1. 获取器 通过 requests 的爬虫爬取免费的 IP 代理网址获取 IP。
2. 过滤器通过获取器获取的代理请求网页数据有数据返回的保存进 Redis。
3. 定时检测器定时拿出一部分 Proxy 重新的用过滤器进行检测剔除不能用的代理。
4. 利用 Flask web 服务器提供 API 方便提取 IP

4、 Python如何爬取 HTTPS 网站?

这类问题属于简单类问题

- 在使用 requests 前加入：requests.packages.urllib3.disable_warnings()。
- 为 requests 添加 verify=False 参数
- 导入ssl模块

```
import ssl
ssl._create_default_https_context = ssl._create_unverified_context
复制代码
```

5、python 爬虫有哪些常用框架?

序号	框架名称	描述	官网
1	Scrapy	Scrapy是一个为了爬取网站数据，提取结构性数据而编写的应用框架。 可以应用在包括数据挖掘，信息处理或存储历史数据等一系列的程序中。用这个框架可以轻松爬下来如亚马逊商品信息之类的数据。	scrapy.org/

序号	框架名称	描述	官网
2	PySpider	pyspider 是一个用python实现的功能强大的网络爬虫系统，能在浏览器界面上进行脚本的编写，功能的调度和爬取结果的实时查看，后端使用常用的数据库进行爬取结果的存储，还能定时设置任务与任务优先级等。	github.com/binux/pyspi...
3	Crawley	Crawley可以高速爬取对应网站的内容，支持关系和非关系数据库，数据可以导出为JSON、XML等。	project.crawley-cloud.com/
4	Portia	Portia是一个开源可视化爬虫工具，可让您在不需要任何编程知识的情况下爬取网站！简单地注释您感兴趣的页面，Portia将创建一个蜘蛛来从类似的页面提取数据。	github.com/scrapinghub...
5	Newspaper	Newspaper可以用来提取新闻、文章和内容分析。使用多线程，支持10多种语言等。	github.com/codelucas/n...
6	Beautiful Soup	Beautiful Soup 是一个可以从HTML或XML文件中提取数据的Python库.它能够通过你喜欢的转换器实现惯用的文档导航,查找,修改文档的方式.Beautiful Soup会帮你节省数小时甚至数天的工作时间	www.crummy.com/software/Be...
7	Grab	Grab是一个用于构建Web刮板的Python框架。借助Grab，您可以构建各种复杂的网页抓取工具，从简单的5行脚本到处理数百万个网页的复杂异步网站抓取工具。Grab提供一个API用于执行网络请求和处理接收到的内容，例如与HTML文档的DOM树进行交互。	docs.grablib.org/en/latest/#...
8	Cola	Cola是一个分布式的爬虫框架，对于用户来说，只需编写几个特定的函数，而无需关注分布式运行的细节。任务会自动分配到多台机器上，整个过程对用户是透明的。	没找着~
9	很多	看自己积累	多百度

6、 Scrapy 的优缺点?

优点：scrapy 是异步的

采取可读性更强的 xpath 代替正则强大的统计和 log 系统，同时在不同的 url 上爬行支持 shell 方式，方便独立调试写 middleware,方便写一些统一的过滤器，通过管道的方式存入数据库。

缺点：基于 python 的爬虫框架，扩展性比较差

基于 twisted 框架，运行中的 exception 是不会干掉 reactor，并且异步框架出错后是不会停掉其他任务的，数据出错后难以察觉。

7、 scrapy 和 request?

- scrapy 是封装起来的框架，他包含了下载器，解析器，日志及异常处理，基于多线程， twisted 的方式处理，对于固定单个网站的爬取开发，有优势，但是对于多网站爬取，并发及分布式处理方面，不够灵活，不便调整与括展。
- request 是一个 HTTP 库， 它只是用来，进行请求，对于 HTTP 请求，他是一个强大的库，下载，解析全部自己处理，灵活性更高，高并发与分布式部署也非常灵活，对于功能可以更好实现。

8、描述下 scrapy 框架运行的机制?

1. 从 start_urls 里获取第一批 url 并发送请求，请求由引擎交给调度器入请求队列，获取完毕后，调度器将请求队列里的请求交给下载器去获取请求对应的响应资源，并将响应交给自己编写的解析方法做提取处理，如果提取出需要的数据，则交给管道文件处理；
2. 如果提取出 url，则继续执行之前的步骤（发送 url 请求，并由引擎将请求交给调度器入队列...），直到请求队列里没有请求，程序结束。

9、 实现模拟登录的方式有哪些?

- 使用一个具有登录状态的 cookie，结合请求报头一起发送，可以直接发送 get 请求，访问登录后才能访问的页面。
- 先发送登录界面的 get 请求，在登录页面 HTML 里获取登录需要的数据（如果需要的话），然后结合账户密码，再发送 post 请求，即可登录成功。然后根据获取的 cookie信息，继续访问之后的页面。

10、 你遇到过的反爬虫的策略？怎么解决?

1. BAN IP
2. BAN USERAGENT
3. BAN COOKIES
4. 验证码验证
5. javascript渲染
6. ajax异步传输
7. 等.....

- 通过headers反爬虫：解决策略，伪造headers
- 基于用户行为反爬虫：动态变化去爬取数据，模拟普通用户的行为
- 通过动态更改代理ip来反爬虫
- 基于动态页面的反爬虫：跟踪服务器发送的ajax请求，模拟ajax请求,selnium 和phtamjs

正则表达式篇

1、请写出一段代码用正则匹配出ip?

```
import re
ip ='192.168.1.1'
trueIp =re.search(r'(([01]{0,1}\d{0,1}\d|2[0-4]\d|25[0-5])\.){3}([01]
{0,1}\d{0,1}\d|2[0-4]\d|25[0-5])',ip)
print(trueIp)
```

2、a = “abbbccc”，用正则匹配为abccc,不管有多少b，就出现一次?

```
思路：不管有多少个b替换成一个

re.sub(r'b+', 'b', a)
```

3、Python字符串查找和替换?

a、str.find()：正序字符串查找函数
函数原型：
str.find(substr [,pos_start [,pos_end]])
返回str中第一次出现的substr的第一个字母的标号，如果str中没有substr则返回-1，也就是说从左边算起的第一次出现的substr的首字母标号。

参数说明：
str：代表原字符串
substr：代表要查找的字符串
pos_start：代表查找的开始位置，默认是从下标0开始查找
pos_end：代表查找的结束位置

例子：
'aabbcc.find('bb')' # 2

b、str.index()：正序字符串查找函数
index()函数类似于find()函数，在Python中也是在字符串中查找子串第一次出现的位置，跟find()不同的是，未找到则抛出异常。

函数原型：
str.index(substr [, pos_start, [pos_end]])

参数说明：
str：代表原字符串
substr：代表要查找的字符串
pos_start：代表查找的开始位置，默认是从下标0开始查找
pos_end：代表查找的结束位置

例子：
'acdd 11 23'.index(' ') # 4

c、str.rfind()：倒序字符串查找函数

函数原型：

```
str.rfind( substr [, pos_start [,pos_ end ] ])
```
返回str中最后出现的substr的第一个字母的标号，如果str中没有substr则返回-1，也就是说从右边算起的第一次出现的substr的首字母标号。

参数说明：
str：代表原字符串
substr：代表要查找的字符串
pos_start：代表查找的开始位置，默认是从下标0开始查找
pos_end：代表查找的结束位置

例子：
```
'adsfddf'.rfind('d') # 5
```

d、str.rindex()：倒序字符串查找函数
rindex()函数类似于rfind()函数，在Python中也是在字符串中倒序查找子串最后一次出现的位置，跟rfind()不同的是，未找到则抛出异常。

函数原型：
```
str.rindex(substr [, pos_start, [ pos_end ] ] )
```

参数说明：
str：代表原字符串
substr：代表要查找的字符串
pos_start：代表查找的开始位置，默认是从下标0开始查找
pos_end：代表查找的结束位置

例子：
```
 'adsfddf'.rindex('d') # 5
```

e、使用re模块进行查找和替换：

函数	说明
re.match(pat, s)	只从字符串s的头开始匹配，比如('123', '12345')匹配上了，而('123','01234')就是没有匹配上，没有匹配上返回None，匹配上返回matchobject
re.search(pat, s)	从字符串s的任意位置都进行匹配，比如('123','01234')就是匹配上了，只要s只能存在符合pat的连续字符串就算匹配上了，没有匹配上返回None，匹配上返回matchobject
re.sub(pat,newpat,s)	re.sub(pat,newpat,s) 对字符串中s的包含的所有符合pat的连续字符串进行替换，如果newpat为str,那么就是替换为newpat,如果newpat是函数，那么就按照函数返回值替换。sub函数两个有默认值的参数分别是count表示最多只处理前几个匹配的字符串，默认为0表示全部处理；最后一个是flags，默认为0

f、使用replace()进行替换：
基本用法：对象.replace(rgExp,replaceText,max)

其中，rgExp和replaceText是必须要有的，max是可选的参数，可以不加。
rgExp是指正则表达式模式或可用标志的正则表达式对象，也可以是 String 对象或文字；
replaceText是一个String 对象或字符串文字；
max是一个数字。
对于一个对象，在对象的每个rgExp都替换成replaceText，从左到右最多max次。

```
s1='hello world'
s1.replace('world','liming')
```

4、用Python匹配HTML tag的时候，<.> *和* <.?> 有什么区别

第一个代表贪心匹配，第二个代表非贪心；
?在一般正则表达式里的语法是指的"零次或一次匹配左边的字符或表达式"相当于{0,1}
而当?后缀于*,+,?,{n},{n,},{n,m}之后，则代表非贪心匹配模式，也就是说，尽可能少的匹配左边的字符或表达式，这里是尽可能少的匹配.(任意字符)

所以：第一种写法是，尽可能多的匹配，就是匹配到的字符串尽量长，第二中写法是尽可能少的匹配，就是匹配到的字符串尽量短。
比如<tag>tag>tag>end，第一个会匹配<tag>tag>tag>,第二个会匹配<tag>。

5、正则表达式贪婪与非贪婪模式的区别?

贪婪模式：
定义：正则表达式去匹配时，会尽量多的匹配符合条件的内容
标识符：+，?，*，{n}，{n,}，{n,m}
匹配时，如果遇到上述标识符，代表是贪婪匹配，会尽可能多的去匹配内容

非贪婪模式：
定义：正则表达式去匹配时，会尽量少的匹配符合条件的内容 也就是说，一旦发现匹配符合要求，立马就匹配成功，而不会继续匹配下去(除非有g，开启下一组匹配)
标识符：+?，??，*?，{n}?，{n,}?，{n,m}?
可以看到，非贪婪模式的标识符很有规律，就是贪婪模式的标识符后面加上一个?

参考文章：https://dailc.github.io/2017/07/06/regularExpressionGreedyAndLazy.html

6、写出开头匹配字母和下划线，末尾是数字的正则表达式?

```
s1='_aai0efe00'
res=re.findall('^[a-zA-Z_]?[a-zA-Z0-9_]{1,}\d$',s1)
print(res)
```

7、怎么过滤评论中的表情?

思路：主要是匹配表情包的范围，将表情包的范围用空替换掉

```
import re
pattern = re.compile(u'[\uD800-\uDBFF][\uDC00-\uDFFF]')
pattern.sub('',text)
```

8、简述Python里面search和match的区别

match()函数只检测字符串开头位置是否匹配，匹配成功才会返回结果，否则返回None；
search()函数会在整个字符串内查找模式匹配,只到找到第一个匹配然后返回一个包含匹配信息的对象,该对象可以通过调用group()方法得到匹配的字符串,如果字符串没有匹配，则返回None。

网络编程

1、怎么实现强行关闭客户端和服务器之间的连接?

在socket通信过程中不算循环检测一个全局变量（开关标记变量），一旦标记变量变为关闭，则调用socket的close方法，循环结束，从而达到关闭连接的目的。

推荐阅读： https://www.cnblogs.com/yang950718/p/10794019.html

2、简述TCP和UDP的区别以及优缺点?

https://blog.csdn.net/xiaobangkuaipao/article/details/76793702

3、简述浏览器通过WSGI请求动态资源的过程?

浏览器发送的请求被Nginx监听到，Nginx根据请求的URL的PATH或者后缀把请求静态资源的分发到静态资源的目录，别的请求根据配置好的转发到相应端口。
实现了WSGI的程序会监听某个端口，监听到Nginx转发过来的请求接收后(一般用socket的recv来接收HTTP的报文)以后把请求的报文封装成 `environ` 的字典对象，然后再提供一个 `start_response` 的方法。把这两个对象当成参数传入某个方法比如 `wsgi_app(environ, start_response)` 或者实现了 `__call__(self, environ, start_response)` 方法的某个实例。这个实例再调用 `start_response` 返回给实现了WSGI的中间件，再由中间件返回给Nginx。

4、描述用浏览器访问www.baidu.com的过程

1.根据域名到DNS中找到IP

2.根据IP建立TCP连接(三次握手)

3.连接建立成功发起http请求

4.服务器响应http请求

5.浏览器解析HTML代码并请求html中的静态资源（js,css）

6.关闭TCP连接（四次挥手）

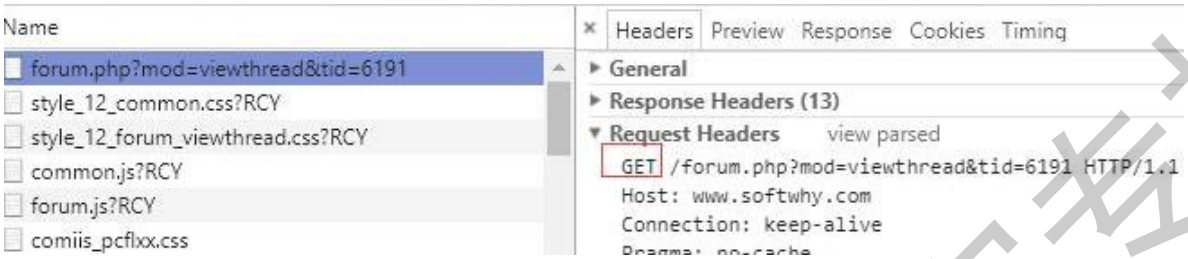
7.浏览器渲染页面

推荐阅读： https://blog.csdn.net/weixin_38497513/article/details/80918425

5、Post和Get请求的区别?

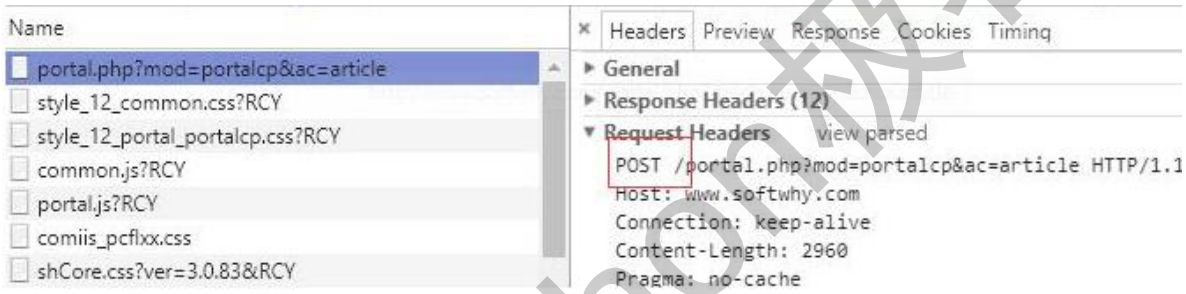
POST和GET都是HTTP请求的方法，当然并不只有这两个方法，只是这两个最常用。

图示如下：



过谷歌浏览器开发者工具的Network选项卡，可以捕获HTTP请求或者响应的相关信息。

上面就是一个GET请求，再来看一个POST请求演示截图：



POST与GET请求区别总结：

（1）.请求数据的传输方式不同:

GET请求数据是通过URI链接传输，看如下请求：

```
[HTML] 纯文本查看 复制代码http://www.softwhy.com/forum.php?mod=viewthread&tid=6191
```

上面链接问号（?）后面就是要向HTTP服务器发送的数据，如果有多个数据，用&连接。

POST请求则是将数据放在HTTP请求体中，截图如下：

```
POST http://www.softwhy.com/portal.php?mod=portalcp&ac=article HTTP/1.1
Host: www.softwhy.com
Connection: keep-alive
Content-Length: 5164
Cache-Control: max-age=0
Origin: http://www.softwhy.com
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gec
Content-Type: multipart/form-data; boundary=----WebKitFormBoundaryLVyFPx4xQgkhFrBy
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,
Referer: http://www.softwhy.com/portal.php?mod=portalcp&ac=article&op=edit&aid=7888
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.8
Cookie: hHd4_2132_saltkey=LEZQ7d8q; hHd4_2132_lastvisit=1507544444; hHd4_2132_pc_siz

------WebKitFormBoundaryLVyFPx4xQgkhFrBy
Content-Disposition: form-data; name="title"
```

红框中就是POST请求向HTTP发送的部分数据，GET和HEAD请求不具有此部分。

（2）.传输数据的大小不同:

虽然HTTP协议没有对传输的数据大小进行限制，也没有对URI长度进行限制，但是实际应用中，不同的浏览器和服务器对此都有所限制。GET方式所能发送数据的大小非常有限，不同的浏览器有所不同。POST不通过URL传值，理论上数据大小不受限，不过各个HTTP服务器会规定对POST提交数据大小的限制，Apache、IIS6都有各自的配置，传输数据的大小要远大于GET方式。

（3）.安全性：

GET是通过URI发送数据，甚至直接在浏览器地址栏就能观察到，安全性非常的低。

POST在各个方面都要比GET更为安全。

6、cookie 和session 的区别?

1、存储位置不同

> cookie的数据信息存放在客户端浏览器上。
>
> session的数据信息存放在服务器上。

2、存储容量不同

> 单个cookie保存的数据<=4KB，一个站点最多保存20个Cookie。
>
> 对于session来说并没有上限，但出于对服务器端的性能考虑，session内不要存放过多的东西，并且设置session删除机制。

3、存储方式不同

> cookie中只能保管ASCII字符串，并需要通过编码方式存储为Unicode字符或者二进制数据。
>
> session中能够存储任何类型的数据，包括且不限于string，integer，list，map等。

4、隐私策略不同

> cookie对客户端是可见的，别有用心的人可以分析存放在本地的cookie并进行cookie欺骗，所以它是不安全的。
>
> session存储在服务器上，对客户端是透明对，不存在敏感信息泄漏的风险。

5、有效期上不同

> 开发可以通过设置cookie的属性，达到使cookie长期有效的效果。
>
> session依赖于名为JSESSIONID的cookie，而cookie JSESSIONID的过期时间默认为-1，只需关闭窗口该session就会失效，因而session不能达到长期有效的效果。

6、服务器压力不同

> cookie保管在客户端，不占用服务器资源。对于并发用户十分多的网站，cookie是很好的选择。
>
> session是保管在服务器端的，每个用户都会产生一个session。假如并发访问的用户十分多，会产生十分多的session，耗费大量的内存。

7、浏览器支持不同

假如客户端浏览器不支持cookie：

> cookie是需要客户端浏览器支持的，假如客户端禁用了cookie，或者不支持cookie，则会话跟踪会失效。关于WAP上的应用，常规的cookie就派不上用场了。
>
> 运用session需要使用URL地址重写的方式。一切用到session程序的URL都要进行URL地址重写，否则session会话跟踪还会失效。

假如客户端支持cookie：

> cookie既能够设为本浏览器窗口以及子窗口内有效，也能够设为一切窗口内有效。

> session只能在本窗口以及子窗口内有效。

8、跨域支持上不同

> cookie支持跨域名访问。

> session不支持跨域名访问。

7、列出你知道的HTTP协议的状态码，说出表示什么意思?

1xx（临时响应）

表示临时响应并需要请求者继续执行操作的状态码。

100（继续）	**请求者应当继续提出请求。服务器返回此代码表示已收到请求的第一部分，正在等待其余部分。**
101（切换协议）	请求者已要求服务器切换协议，服务器已确认并准备切换。

2xx（成功） 表示成功处理了请求的状态码。

200（成功）	**服务器已成功处理了请求。通常，这表示服务器提供了请求的网页。如果是对您的 robots.txt 文件显示此状态码，则表示 Googlebot 已成功检索到该文件。**
201（已创建）	请求成功并且服务器创建了新的资源。
202（已接受）	服务器已接受请求，但尚未处理。
203（非授权信息）	服务器已成功处理了请求，但返回的信息可能来自另一来源。
204（无内容）	服务器成功处理了请求，但没有返回任何内容。
205（重置内容）	服务器成功处理了请求，但没有返回任何内容。与 204 响应不同，此响应要求请求者重置文档视图（例如，清除表单内容以输入新内容）。
206（部分内容）	服务器成功处理了部分 GET 请求。

3xx（重定向）

要完成请求，需要进一步操作。通常，这些状态码用来重定向。Google 建议您在每次请求中使用重定向不要超过 5 次。您可以使用网站管理员工具查看一下 Googlebot 在抓取重定向网页时是否遇到问题。诊断下的网络抓取页列出了由于重定向错误导致 Googlebot 无法抓取的网址。

300（多种选择）	**针对请求，服务器可执行多种操作。服务器可根据请求者 (user agent) 选择一项操作，或提供操作列表供请求者选择。**
301（永久移动）	请求的网页已永久移动到新位置。服务器返回此响应（对 GET 或 HEAD 请求的响应）时，会自动将请求者转到新位置。您应使用此代码告诉 Googlebot 某个网页或网站已永久移动到新位置。
302、（临时移动）	服务器目前从不同位置的网页响应请求，但请求者应继续使用原有位置来响应以后的请求。此代码与响应 GET 和 HEAD 请求的 301 代码类似，会自动将请求者转到不同的位置，但您不应使用此代码来告诉 Googlebot 某个网页或网站已经移动，因为 Googlebot 会继续抓取原有位置并编制索引。
303（查看其他位置）	请求者应当对不同的位置使用单独的 GET 请求来检索响应时，服务器返回此代码。对于除 HEAD 之外的所有请求，服务器会自动转到其他位置。
304（未修改）	自从上次请求后，请求的网页未修改过。服务器返回此响应时，不会返回网页内容。如果网页自请求者上次请求后再也没有更改过，您应将服务器配置为返回此响应（称为 If-Modified-Since HTTP 标头）。服务器可以告诉 Googlebot 自从上次抓取后网页没有变更，进而节省带宽和开销。 .
305（使用代理）	请求者只能使用代理访问请求的网页。如果服务器返回此响应，还表示请求者应使用代理。
307（临时重定向）	服务器目前从不同位置的网页响应请求，但请求者应继续使用原有位置来响应以后的请求。此代码与响应 GET 和 HEAD 请求的 <a href=answer.py?answer=>301 代码类似，会自动将请求者转到不同的位置，但您不应使用此代码来告诉 Googlebot 某个页面或网站已经移动，因为 Googlebot 会继续抓取原有位置并编制索引。

4xx（请求错误） 这些状态码表示请求可能出错，妨碍了服务器的处理。

400（错误请求）	**服务器不理解请求的语法。**
401（未授权）	请求要求身份验证。对于登录后请求的网页，服务器可能返回此响应。
403（禁止）	服务器拒绝请求。如果您在 Googlebot 尝试抓取您网站上的有效网页时看到此状态码（您可以在 Google 网站管理员工具诊断下的网络抓取页面上看到此信息），可能是您的服务器或主机拒绝了 Googlebot 访问。
404（未找到）	服务器找不到请求的网页。例如，对于服务器上不存在的网页经常会返回此代码。 如果您的网站上没有 robots.txt 文件，而您在 Google 网站管理员工具"诊断"标签的 robots.txt 页上看到此状态码，则这是正确的状态码。但是，如果您有 robots.txt 文件而又看到此状态码，则说明您的 robots.txt 文件可能命名错误或位于错误的位置（该文件应当位于顶级域，名为 robots.txt）。 如果对于 Googlebot 抓取的网址看到此状态码（在"诊断"标签的 HTTP 错误页面上），则表示 Googlebot 跟随的可能是另一个页面的无效链接（是旧链接或输入有误的链接）。
405（方法禁用）	禁用请求中指定的方法。
406（不接受）	无法使用请求的内容特性响应请求的网页。
407（需要代理授权）	此状态码与 <a href=answer.py?answer=35128>401（未授权）类似，但指定请求者应当授权使用代理。如果服务器返回此响应，还表示请求者应当使用代理。
408（请求超时）	服务器等候请求时发生超时。
409（冲突）	服务器在完成请求时发生冲突。服务器必须在响应中包含有关冲突的信息。服务器在响应与前一个请求相冲突的 PUT 请求时可能会返回此代码，以及两个请求的差异列表。
410（已删除）	如果请求的资源已永久删除，服务器就会返回此响应。该代码与 404（未找到）代码类似，但在资源以前存在而现在不存在的情况下，有时会用来替代 404 代码。如果资源已永久移动，您应使用 301 指定资源的新位置。
411（需要有效长度）	服务器不接受不含有效内容长度标头字段的请求。
412（未满足前提条件）	服务器未满足请求者在请求中设置的其中一个前提条件。
413（请求实体过大）	服务器无法处理请求，因为请求实体过大，超出服务器的处理能力。

400（错误请求）	**服务器不理解请求的语法。**
414（请求的 URI 过长）	请求的 URI（通常为网址）过长，服务器无法处理。
415（不支持的媒体类型）	请求的格式不受请求页面的支持。
416（请求范围不符合要求）	如果页面无法提供请求的范围，则服务器会返回此状态码。
417（未满足期望值）	服务器未满足"期望"请求标头字段的要求。

5xx（服务器错误） 这些状态码表示服务器在处理请求时发生内部错误。这些错误可能是服务器本身的错误，而不是请求出错。

500（服务器内部错误）	**服务器遇到错误，无法完成请求。**
501（尚未实施）	服务器不具备完成请求的功能。例如，服务器无法识别请求方法时可能会返回此代码。
502（错误网关）	服务器作为网关或代理，从上游服务器收到无效响应。
503（服务不可用）	服务器目前无法使用（由于超载或停机维护）。通常，这只是暂时状态。
504（网关超时）	服务器作为网关或代理，但是没有及时从上游服务器收到请求。
505（HTTP 版本不受支持）	服务器不支持请求中所用的 HTTP 协议版本。

8、请简单说一下三次握手和四次挥手?

9、说一下什么是tcp的2MSL?

10、为什么客户端在TIME-WAIT状态必须等待2MSL的时间?

11、说说HTTP和HTTPS区别?

12、谈一下HTTP协议以及协议头部中表示数据类型的字段?

13、HTTP请求方法都有什么?

14、使用Socket套接字需要传入哪些参数？

15、HTTP常见请求头?

16、七层模型?

17、url的形式?

Flask篇

1、 什么是Flask?

Flask是Python编写的一款轻量级Web应用框架。其 WSGI 工具箱采用 Werkzeug ，模板引擎则使用 Jinja2。Flask使用 BSD 授权。其中两个环境依赖是Werkzeug和jinja2，这意味着它不需要依赖外部库。正因如此，我们将其称为轻量级框架。

Flask会话使用签名cookie让用户查看和修改会话内容。它会记录从一个请求到另一个请求的信息。不过，要想修改会话，用户必须有密钥Flask.secret_key。

2、 Flask中的请求上下文和应用上下文是什么?

- 在Flask中处理请求时，就会产生一个 “请求上下文” 对象，整个请求的处理过程，都会在这个上下文对象中进行。 这保证了请求的处理过程不被干扰。 包含了和请求处理相关的信息，同时Flask还根据werkzeug.local模块中实现的一种数据结构LocalStack用来存储“请求上下文”对象。
- “应用上下文” 也是一个上下文对象，可以使用with语句构造一个上下文环境，它也实现了push、pop等方法。 “应用上下文” 的构造函数也和 “请求上下文” 类似，都有app、url_adapter等属性。 “应用上下文” 存在的一个主要功能就是确定请求所在的应用。

3、对Flask蓝图(Blueprint)的理解?

蓝图的定义

蓝图 /Blueprint 是Flask应用程序组件化的方法，可以在一个应用内或跨越多个项目共用蓝图。使用蓝图可以极大简化大型应用的开发难度，也为Flask扩展提供了一种在应用中注册服务的集中式机制。

蓝图的应用场景：

把一个应用分解为一个蓝图的集合。这对大型应用是理想的。一个项目可以实例化一个应用对象，初始化几个扩展，并注册一集合的蓝图。

以URL前缀和/或子域名，在应用上注册一个蓝图。URL前缀/子域名中的参数即成为这个蓝图下的所有视图函数的共同的视图参数（默认情况下）
在一个应用中用不同的URL规则多次注册一个蓝图。

通过蓝图提供模板过滤器、静态文件、模板和其他功能。一个蓝图不一定要实现应用或视图函数。

初始化一个Flask扩展时，在这些情况中注册一个蓝图。

蓝图的缺点：

不能在应用创建后撤销注册一个蓝图而不销毁整个应用对象。

使用蓝图的三个步骤

1.创建一个蓝图对象

```
blue = Blueprint("blue",__name__)
```

2.在这个蓝图对象上进行操作，例如注册路由、指定静态文件夹、注册模板过滤器...

```
@blue.route('/')
def blue_index():
    return "Welcome to my blueprint"
```

3.在应用对象上注册这个蓝图对象

```
app.register_blueprint(blue,url_prefix="/blue")
```

4、Flask 和 Django 路由映射的区别?

在django中，路由是浏览器访问服务器时，先访问的项目中的url，再由项目中的url找到应用中url，这些url是放在一个列表里，遵从从前往后匹配的规则。在flask中，路由是通过装饰器给每个视图函数提供的，而且根据请求方式的不同可以一个url用于不同的作用。

Django篇

1、如何理解 Django 被称为 MTV 模式?

这个题就是面向对象设计和设计模式的开始。 你可能比较熟悉的模式叫做: MVC。说是 Model View Controller，而在 Django 中因为 Template 来处理视图展现，所以称为: MTV。 接下里会问到的就是分层的概念，有句话叫：“没有什么问题是不能通过增加一层解决的，如果有，那就再加一层。”当然还会有设计模式的一些原则等着你，比如开-闭原则、单一职责原则等。

2、简述Django的orm

ORM，全拼Object-Relation Mapping，意为对象-关系映射

实现了数据模型与数据库的解耦，通过简单的配置就可以轻松更换数据库，而不需要修改代码只需要面向对象编程 ORM操作本质上会根据对接的数据库引擎，翻译成对应的sql语句, 所有使用Django开发的项目无需关心程序底层使用的是MySql、Oracle、SQLite....，如果数据库迁移，只需要更换Django的数据库引擎即可。

3、解释下什么是 ORM 以及它的优缺点是什么?

ORM：Object Relational Mapping(对象关系映射)，它做的事就是帮我们封装一下对数据库的操作，避免我们来写不太好维护的 SQL 代码。

- 优点就是让我们写的代码更容易维护，因为里面不用夹杂着各种 SQL 代码。
- 缺点是失去了 SQL 的灵活，并且越是通用的 ORM 框架，性能损耗会越大。

说到性能损耗，可以接着聊的是 Django 中的 raw sql，也就是说 Model.objects.raw 这个方法的使用，它的作用、原理、性能提升等。还可以继续聊另外一个老生常谈的问题：N+1 的问题。

4、Django 系统中如何配置数据库的长连接?

这涉及到 Django 如何处理数据库连接细节的问题。默认情况下对于每一个请求 Django 都会建立一个新的数据库连接。这意味着当请求量过大时就会出现数据库(MySQL)的 Too many connection 的问题，对于这个问题，在其他的语言框架中有连接池这样的东西来减少数据库的连接数，来提升连接的使用效率。而在 Django中，为了处理这一问题，增加了一个配置:

`CONN_MAX_AGE`，在 settings 的 DATABASES 配置中。配置了该选项后，Django 会跟数据库保持链接（时长取决于 CONN_MAX_AGE 设定的值），不再会针对每个请求都创建新的连接了。

但是需要注意的是，这跟数据库连接池的概念还不太一样。

5、什么是wsgi,uwsgi,uWSGI?

WSGI:

web服务器网关接口，是一套协议。用于接收用户请求并将请求进行初次封装，然后将请求交给web框架。

实现wsgi协议的模块：wsgiref,本质上就是编写一socket服务端，用于接收用户请求（django)

werkzeug,本质上就是编写一个socket服务端，用于接收用户请求(flask)

uwsgi:

与WSGI一样是一种通信协议，它是uWSGI服务器的独占协议，用于定义传输信息的类型。
uWSGI:

是一个web服务器，实现了WSGI的协议，uWSGI协议，http协议

6、Django、Flask、Tornado的对比?

1、 Django走的大而全的方向，开发效率高。它的MTV框架，自带的ORM,admin后台管理,自带的sqlite数据库和开发测试用的服务器，给开发者提高了超高的开发效率。
重量级web框架，功能齐全，提供一站式解决的思路，能让开发者不用在选择上花费大量时间。

自带ORM和模板引擎，支持jinja等非官方模板引擎。

自带ORM使Django和关系型数据库耦合度高，如果要使用非关系型数据库，需要使用第三方库

自带数据库管理app

成熟，稳定，开发效率高，相对于Flask，Django的整体封闭性比较好，适合做企业级网站的开发。
python web框架的先驱，第三方库丰富

2、 Flask 是轻量级的框架，自由，灵活，可扩展性强，核心基于Werkzeug WSGI工具 和jinja2 模板引擎

适用于做小网站以及web服务的API,开发大型网站无压力，但架构需要自己设计

与关系型数据库的结合不弱于Django，而与非关系型数据库的结合远远优于Django

3、 Tornado走的是少而精的方向，性能优越，它最出名的异步非阻塞的设计方式

Tornado的两大核心模块：

iostraem:对非阻塞的socket进行简单的封装

ioloop: 对I/O 多路复用的封装,它实现一个单例

7、CORS 和 CSRF的区别?

什么是CORS?

CORS是一个W3C标准,全称是"跨域资源共享"(Cross-origin resoure sharing).
它允许浏览器向跨源服务器，发出XMLHttpRequest请求，从而客服了AJAX只能同源使用的限制。

什么是CSRF?

CSRF主流防御方式是在后端生成表单的时候生成一串随机token,内置到表单里成为一个字段，同时，将此串token置入session中。每次表单提交到后端时都会检查这两个值是否一致，以此来判断此次表单提交是否是可信的，提交过一次之后，如果这个页面没有生成CSRF token,那么token将会被清空,如果有新的需求，那么token会被更新。
攻击者可以伪造POST表单提交，但是他没有后端生成的内置于表单的token，session中没有token都无济于事。

8、Session,Cookie,JWT的理解

为什么要使用会话管理

众所周知，HTTP协议是一个无状态的协议，也就是说每个请求都是一个独立的请求，请求与请求之间并无关系。但在实际的应用场景，这种方式并不能满足我们的需求。举个大家都喜欢用的例子，把商品加入购物车，单独考虑这个请求，服务端并不知道这个商品是谁的，应该加入谁的购物车？因此这个请求的上下文环境实际上应该包含用户的相关信息，在每次用户发出请求时把这一小部分额外信息，也做为请求的一部分，这样服务端就可以根据上下文中的信息，针对具体的用户进行操作。所以这几种技术的出现都是对HTTP协议的一个补充，使得我们可以用HTTP协议+状态管理构建一个的面向用户的WEB应用。

Session 和Cookie的区别

这里我想先谈谈session与cookies,因为这两个技术是做为开发最为常见的。那么session与cookies的区别是什么？个人认为session与cookies最核心区别在于额外信息由谁来维护。利用cookies来实现会话管理时，用户的相关信息或者其他我们想要保持在每个请求中的信息，都是放在cookies中,而cookies是由客户端来保存，每当客户端发出新请求时，就会稍带上cookies,服务端会根据其中的信息进行操作。
当利用session来进行会话管理时，客户端实际上只存了一个由服务端发送的session_id,而由这个session_id,可以在服务端还原出所需要的所有状态信息，从这里可以看出这部分信息是由服务端来维护的。

除此以外，session与cookies都有一些自己的缺点：

cookies的安全性不好，攻击者可以通过获取本地cookies进行欺骗或者利用cookies进行CSRF攻击。使用cookies时,在多个域名下，会存在跨域问题。
session 在一定的时间里，需要存放在服务端，因此当拥有大量用户时，也会大幅度降低服务端的性能，当有多台机器时，如何共享session也会是一个问题.(redis集群)也就是说，用户第一个访问的时候

是服务器A，而第二个请求被转发给了服务器B，那服务器B如何得知其状态。实际上，session与cookies是有联系的，比如我们可以把session_id存放在cookies中的。

JWT是如何工作的

首先用户发出登录请求，服务端根据用户的登录请求进行匹配，如果匹配成功，将相关的信息放入payload中，利用算法，加上服务端的密钥生成token，这里需要注意的是secret_key很重要，如果这个泄露的话，客户端就可以随机篡改发送的额外信息，它是信息完整性的保证。生成token后服务端将其返回给客户端，客户端可以在下次请求时，将token一起交给服务端，一般是说我们可以将其放在Authorization首部中，这样也就可以避免跨域问题。

9、简述Django请求生命周期

一般是用户通过浏览器向我们的服务器发起一个请求(request),这个请求会去访问视图函数，如果不涉及到数据调用，那么这个时候视图函数返回一个模板也就是一个网页给用户)
视图函数调用模型毛模型去数据库查找数据，然后逐级返回，视图函数把返回的数据填充到模板中空格中，最后返回网页给用户。

1.wsgi ,请求封装后交给web框架（Flask，Django)

2.中间件，对请求进行校验或在请求对象中添加其他相关数据，例如：csrf,request.session

3.路由匹配 根据浏览器发送的不同url去匹配不同的视图函数

4.视图函数，在视图函数中进行业务逻辑的处理，可能涉及到：orm，templates

5.中间件，对响应的数据进行处理

6.wsgi，将响应的内容发送给浏览器

10、用的restframework完成api发送时间时区

当前的问题是用django的rest framework模块做一个get请求的发送时间以及时区信息的api

```
class getCurrenttime(APIView):
    def get(self,request):
        local_time = time.localtime()
        time_zone =settings.TIME_ZONE
        temp = {'localtime':local_time,'timezone':time_zone}
        return Response(temp)
```

11、nginx,tomcat,apach到都是什么?

Nginx（engine x)是一个高性能的HTTP和反向代理服务器，也是 一个IMAP/POP3/SMTP服务器，工作在OSI七层，负载的实现方式：轮询，IP_HASH,fair,session_sticky.
Apache HTTP Server是一个模块化的服务器，源于NCSAhttpd服务器
Tomcat 服务器是一个免费的开放源代码的Web应用服务器，属于轻量级应用服务器，是开发和调试JSP程序的首选。

12、请给出你熟悉关系数据库范式有哪些，有什么作用?

在进行数据库的设计时，所遵循的一些规范，只要按照设计规范进行设计，就能设计出没有数据冗余和数据维护异常的数据库结构。

数据库的设计的规范有很多，通常来说我们在设是数据库时只要达到其中一些规范就可以了，这些规范又称之为数据库的三范式，一共有三条，也存在着其他范式，我们只要做到满足前三个范式的要求，就能设陈出符合我们的数据库了，我们也不能全部来按照范式的要求来做，还要考虑实际的业务使用情况，所以有时候也需要做一些违反范式的要求。
1.数据库设计的第一范式(最基本)，基本上所有数据库的范式都是符合第一范式的，符合第一范式的表具有以下几个特点：

数据库表中的所有字段都只具有单一属性，单一属性的列是由基本的数据类型（整型，浮点型，字符型等）所构成的设计出来的表都是简单的二比表

2.数据库设计的第二范式(是在第一范式的基础上设计的)，要求一个表中只具有一个业务主键，也就是说符合第二范式的表中不能存在非主键列对只对部分主键的依赖关系

3.数据库设计的第三范式，指每一个非主属性既不部分依赖与也不传递依赖于业务主键，也就是第二范式的基础上消除了非主属性对主键的传递依赖

13、简述QQ登陆过程

qq登录，在我们的项目中分为了三个接口，

第一个接口是请求qq服务器返回一个qq登录的界面;

第二个接口是通过扫码或账号登陆进行验证，qq服务器返回给浏览器一个code和state,利用这个code通过本地服务器去向qq服务器获取access_token覆返回给本地服务器，凭借access_token再向qq服务器获取用户的openid(openid用户的唯一标识)

第三个接口是判断用户是否是第一次qq登录，如果不是的话直接登录返回的jwt-token给用户，对没有绑定过本网站的用户，对openid进行加密生成token进行绑定

14、项目中日志的作用

一、日志相关概念

1.日志是一种可以追踪某些软件运行时所发生事件的方法

2.软件开发人员可以向他们的代码中调用日志记录相关的方法来表明发生了某些事情

3.一个事件可以用一个包含可选变量数据的消息来描述

4.此外，事件也有重要性的概念，这个重要性也可以被成为严重性级别(level)

二、日志的作用

1.通过log的分析，可以方便用户了解系统或软件、应用的运行情况;

2.如果你的应用log足够丰富，可以分析以往用户的操作行为、类型喜好，地域分布或其他更多信息;

3.如果一个应用的log同时也分了多个级别，那么可以很轻易地分析得到该应用的健康状况，及时发现问题并快速定位、解决问题，补救损失。

4.简单来讲就是我们通过记录和分析日志可以了解一个系统或软件程序运行情况是否正常，也可以在应用程序出现故障时快速定位问题。不仅在开发中，在运维中日志也很重要，日志的作用也可以简单。总结为以下几点：

1.程序调试

2.了解软件程序运行情况，是否正常

3,软件程序运行故障分析与问题定位

4,如果应用的日志信息足够详细和丰富，还可以用来做用户行为分析

15、django中间件的使用?

Django在中间件中预置了六个方法，这六个方法的区别在于不同的阶段执行，对输入或输出进行干预，方法如下：

1.初始化：无需任何参数，服务器响应第一个请求的时候调用一次，用于确定是否启用当前中间件

```
def __init__():
    pass
```

2.处理请求前：在每个请求上调用，返回None或HttpResponse对象。

```
def process_request(request):
    pass
```

3.处理视图前:在每个请求上调用，返回None或HttpResponse对象。

```
def process_view(request,view_func,view_args,view_kwargs):
    pass
```

4.处理模板响应前：在每个请求上调用，返回实现了render方法的响应对象。

```
def process_template_response(request,response):
    pass
```

5.处理响应后：所有响应返回浏览器之前被调用，在每个请求上调用，返回HttpResponse对象。

```
def process_response(request,response):
    pass
```

6.异常处理：当视图抛出异常时调用，在每个请求上调用，返回一个HttpResponse对象。

```
def process_exception(request,exception):
    pass
```

16、谈一下你对uWSGI和nginx的理解?

1.uWSGI是一个Web服务器，它实现了WSGI协议、uwsgi、http等协议。Nginx中HttpUwsgiModule的作用是与uWSGI服务器进行交换。WSGI是一种Web服务器网关接口。它是一个Web服务器（如nginx，uWSGI等服务器）与web应用（如用Flask框架写的程序）通信的一种规范。

要注意WSGI/uwsgi/uWSGI这三个概念的区分。

WSGI是一种通信协议。

uwsgi是一种线路协议而不是通信协议，在此常用于在uWSGI服务器与其他网络服务器的数据通信。

uWSGI是实现了uwsgi和WSGI两种协议的Web服务器。

nginx 是一个开源的高性能的HTTP服务器和反向代理：

1.作为web服务器，它处理静态文件和索引文件效果非常高

2.它的设计非常注重效率，最大支持5万个并发连接，但只占用很少的内存空间

3.稳定性高，配置简洁。

4.强大的反向代理和负载均衡功能，平衡集群中各个服务器的负载压力应用

17、Python中三大框架各自的应用场景?

django:主要是用来搞快速开发的，他的亮点就是快速开发，节约成本，,如果要实现高并发的话，就要对django进行二次开发，比如把整个笨重的框架给拆掉自己写socket实现http的通信,底层用纯c,c++写提升效率，ORM框架给干掉，自己编写封装与数据库交互的框架,ORM虽然面向对象来操作数据库，但是它的效率很低，使用外键来联系表与表之间的查询;
flask: 轻量级，主要是用来写接口的一个框架，实现前后端分离，提考开发效率，Flask本身相当于一个内核，其他几乎所有的功能都要用到扩展(邮件扩展Flask-Mail，用户认证Flask-Login),都需要用第三方的扩展来实现。比如可以用Flask-extension加入ORM、文件上传、身份验证等。Flask没有默认使用的数据库，你可以选择MySQL，也可以用NoSQL。

其WSGI工具箱用Werkzeug(路由模块)，模板引擎则使用Jinja2,这两个也是Flask框架的核心。

Tornado： Tornado是一种Web服务器软件的开源版本。Tornado和现在的主流Web服务器框架（包括大多数Python的框架）有着明显的区别：它是非阻塞式服务器，而且速度相当快。得利于其非阻塞的方式和对epoll的运用，Tornado每秒可以处理数以千计的连接因此Tornado是实时Web服务的一个理想框架

18、Django中哪里用到了线程？哪里用到了协程？哪里用到了进程?

1.Django中耗时的任务用一个进程或者线程来执行，比如发邮件，使用celery.

2.部署django项目是时候，配置文件中设置了进程和协程的相关配置。

19、有用过Django REST framework吗?

Django REST framework是一个强大而灵活的Web API工具。使用RESTframework的理由有：

Web browsable API对开发者有极大的好处

包括OAuth1a和OAuth2的认证策略

支持ORM和非ORM数据资源的序列化

全程自定义开发--如果不想使用更加强大的功能，可仅仅使用常规的function-based views额外的文档和强大的社区支持

20、对cookies与session的了解？他们能单独用吗?

Session采用的是在服务器端保持状态的方案，而Cookie采用的是在客户端保持状态的方案。但是禁用Cookie就不能得到Session。因为Session是用Session ID来确定当前对话所对应的服务器Session，而Session ID是通过Cookie来传递的，禁用Cookie相当于SessionID,也就得不到Session。

21、 django开发中数据做过什么优化?

1. 设计表时，尽量少使用外键，因为外键约束会影响插入和删除性能
2. 使用缓存，减少对数据库的访问
3. orm框架下设置表时，能使用varchar确定字段长度时，就别用text
4. 可以给搜索频率搞得字段属性，在定义时创建索引
5. django orm 框架下的Querysets 本来就有缓存的
6. 如果一个页面需要多次链接数据库，最好一次性去除所有需要的数据，减少数据库的查询次数
7. 若页面只需要数据库里面的某一两个字段时，可以用QuerySet.values()
8. 在模板标签里使用with标签可以缓存Qset查询结果

22、解释一下 Django 和 Tornado 的关系、差别?

Django

Django源自一个在线新闻 Web站点，于 2005 年以开源的形式被释放出来。

Django 框架的核心组件有：

用于创建模型的对象关系映射为最终用户设计的完美管理界面一流的 URL 设计设计者友好的模板语言缓存系统等等

它鼓励快速开发,并遵循MVC设计。

Django遵守 BSD版权，最新发行版本是Django1.4，于2012年03月23日发布.Django的主要目的是简便、快速的开发数据库驱动的网站。它强调代码复用,多个组件可以很方便的以“插件”形式服务于整个框架，Django有许多功能强大的第三方插件，你甚至可以很方便的开发出自己的工具包。这使得Django具有很强的可扩展性。它还强调快速开发和DRY(Do Not RepeatYourself)原则。

Tornado

Tornado是 FriendFeed使用的可扩展的非阻塞式 web 服务器及其相关工具的开源版本。这个 Web 框架看起来有些像 web.py 或者 Google 的 webapp，不过为了能有效利用非阻塞式服务器环境，这个 Web 框架还包含了一些相关的有用工具和优化。

Tornado 和现在的主流 Web 服务器框架（包括大多数Python 的框架）有着明显的区别：它是非阻塞式服务器，而且速度相当快。得利于其 非阻塞的方式和对epoll的运用，Tornado 每秒可以处理数以千计的连接，这意味着对于实时 Web服务来说，Tornado 是一个理想的 Web 框架。我们开发这个 Web 服务器的主要目的就是为了处理 FriendFeed 的实时功能 ——在 FriendFeed 的应用里每一个活动用户都会保持着一个服务器连接。（关于如何扩容 服务器，以处理数以千计的客户端的连接的问题。

23、什么是restful API，谈谈你的理解?

- REST：Representational State Transfer 的缩写，翻译：“具象状态传输”。一般解释为“表现层状态转换”。
- REST 是设计风格而不是标准。是指客户端和服务器的交互形式。我们需要关注的重点是如何设计
- REST 风格的网络接口。

REST 的特点

1. 具象的。一般指表现层，要表现的对象就是资源。比如，客户端访问服务器，获取的数据就是资源。比如文字、图片、音视频等。
2. 表现：资源的表现形式。txt 格式、html 格式、json 格式、jpg 格式等。浏览器通过URL 确定资源的位置，但是需要在HTTP 请求头中，用Accept 和Content-Type 字段指定，这两个字段是对资源表现的描述。
3. 状态转换：客户端和服务器交互的过程。在这个过程中，一定会有数据和状态的转化，这种转化叫做状态转换。其中，GET 表示获取资源，POST 表示新建资源，PUT 表示更新资源，DELETE 表示删除资源。HTTP 协议中最常用的就是这四种操作方式。

RESTful 架构

1. 每个URL 代表一种资源；
2. 客户端和服务器之间，传递这种资源的某种表现层；
3. 客户端通过四个http 动词，对服务器资源进行操作，实现表现层状态转换。