

1、一行代码实现 1--100 之和利用 sum()函数求和

```
In [3]: sum(range(1, 101))
Out[3]: 5050
```

2、如何在一个函数内部修改全局变量函数内部 global 声明修改全局变量

```
In [2]: a = 5
In [3]: def fn():
...:     global a
...:     a = 4
...:
In [4]: fn()
In [5]: print(a)
4
```

3、列出 5 个 python 标准库

os: 提供了不少与操作系统相关联的函数

sys: 通常用于命令行参数

re: 正则匹配

math: 数学运算

datetime:处理日期时间

4、字典如何删除键和合并两个字典

```
In [12]: dic = {"name": "zs", "age": 18}
In [13]: del dic["name"]
In [14]: dic
Out[14]: {'age': 18}
In [15]: dic2 = {"name": "ls"}
In [16]: dic.update(dic2)
In [17]: dic
Out[17]: {'age': 18, 'name': 'ls'}
```

5、谈下 python 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 程序的时候会霸占 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。

多进程中因为每个进程都能被系统分配资源, 相当于每个进程有了一个 python 解释器, 所以多进程可以实现多个进程的同时运行, 缺点是进程系统资源开销大

6、python 实现列表去重的方法

先通过集合去重, 在转列表:

```
In [7]: list = [11, 12, 13, 12, 15, 16, 13]
In [8]: a = set(list)
In [9]: a
Out[9]: {11, 12, 13, 15, 16} ← 集合去重
In [10]: [x for x in a]
Out[10]: [16, 11, 12, 13, 15]
```


7、fun(*args,**kwargs)中的*args,**kwargs 什么意思?

*args 和 **kwargs主要用于函数定义。你可以将不定数量的参数传递给一个函数。这里的`不定`的意思是:预先并不知道,函数使用者会传递多少个参数给你,所以在这个场景下使用这两个关键字。*args是用来发送一个非键值对的可变数量的参数列表给一个函数.这里有个例子帮你理解这个概念:

```
In [6]: def demo(args_f,*args_v):
        print args_f
        for x in args_v:
            print x
        ...:

In [7]: demo('a','b','c','d')
a
b
c
d
```

`**kwargs`允许你将不定长度的键值对,作为参数传递给一个函数。如果你想要在一个函数里处理带名字的参数,你应该使用`**kwargs`。这里有个例子帮你理解这个概念:

 `**kwargs`允许你将不定长度的键值对,作为参数传递给一个函数。如果你想要在一个函数里处理带名字的参数,你应该使用`**kwargs`。这里有个例子帮你理解这个概念:

```
In [1]: def demo(**args_v):
...:     for k,v in args_v.items():
...:         print k,v
...:

In [2]: demo(name='njcx')
name njcx
```

8、python2 和 python3 的 range (100) 的区别

python2 返回列表, python3 返回迭代器, 节约内存

9、一句话解释什么样的语言能够用装饰器?

函数可以作为参数传递的语言, 可以使用装饰器

10、python 内建数据类型有哪些

整型--int

布尔型--bool

字符串--str

列表--list

元组--tuple

字典--dict

11、简述面向对象中__new__和__init__区别

`__init__`是初始化方法, 创建对象后, 就立刻被默认调用了, 可接收参数, 如图

```

In [1]: class Bike:
...:     def __init__(self, newWheelNum, newColor):
...:         self.wheelNum = newWheelNum
...:         self.color = newColor
...:     def move(self):
...:         print('车会跑')
...:
...:     # 创建对象
...:     BM = Bike(2, 'green')
...:     print('车的颜色为:%s' % BM.color)
...:     print('车轮子数量为:%d' % BM.wheelNum)
...:
车的颜色为:green
车轮子数量为:2

```

__init__ 方法自动被调用，可以创建对象接收参数

只打印 __init__ 方法执行的结果，move 方法未执行

- 1、__new__ 至少要有两个参数 cls，代表当前类，此参数在实例化时由 Python 解释器自动识别
- 2、__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
- 3、__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
- 4、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。

```

1 class A(object):
2     def __init__(self):
3         print("这是 init 方法", self)
4
5     def __new__(cls):
6         print("这是cls的ID", id(cls))
7         print("这是 new 方法", object.__new__(cls))
8         return object.__new__(cls)
9
10 A()
11 print("这是类A的ID", id(A))

```

这是cls的ID 1591939383448

这是 new 方法 <_main_.A object at 0x00000172A8DDC518>

这是 init 方法 <_main_.A object at 0x00000172A8DDC518>

这是类A的ID 1591939383448

[Finished in 0.1s]

init方法中的self和new方法返回值地址一样，说明返回值是对象

cls和类ID一样，说明指向同一个类，也就是cls就是创建的实例类

12、简述 with 方法打开处理文件帮我们做了什么？

```
3 f=open("./1.txt","wb")
9 try:
0     f.write("hello world")
1 except:
2     pass
3 finally:
4     f.close()
```

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 finally f.close()关闭文件，with 方法帮我们实现了 finally 中 f.close

(当然还有其他自定义功能，有兴趣可以研究 with 方法源码)

13、python 中生成随机整数、随机小数、0--1 之间小数方法

随机整数：random.randint(a,b),生成区间内的整数

随机小数：习惯用 numpy 库，利用 np.random.randn(5)生成 5 个随机小数

0-1 随机小数：random.random(),括号中不传参

```
36 import random
37 import numpy as np
38 result = random.randint(10,20)  ← 区间
39 res = np.random.randn(5)  ←
40 ret = random.random()  ← 不能传数值
41 print("正整数",result)
42 print("5个随机小数",res)
43 print("0-1随机小数",ret)
```

正整数 14
5个随机小数 [0.17721496 -0.76364375 -0.39721767 -0.23669313 0.21354755]
0-1随机小数 0.6659634699304517
[Finished in 1.9s]

14、避免转义给字符串加哪个字母表示原始字符串？

r，表示需要原始字符串，不转义特殊字符

15、<div class="nam">中国</div>，用正则匹配出标签里面的内容（“中国”），其中 class 的类名是不确定的

```
45 import re
46 str = '<div class="nam">中国</div>'
47 res=re.findall(r'<div class=".*">(.*?)</div>',str)
48 print(res)
```

['中国']
[Finished in 0.5s]

.代表可有可无
*代表任意字符
满足类名可以变化

(.*?)提取文本

16、python 中断言方法举例

assert () 方法，断言成功，则程序继续执行，断言失败，则程序报错

```
50 a=3
51 assert(a>1)
52 print("断言成功，程序继续向下执行")
53
54 b=4
55 assert(b>7)
56 print("断言失败，程序报错")
57
```

['中国']
断言成功，程序继续向下执行
Traceback (most recent call last):
File "C:\Users\ry-wu.junya\Desktop\test\gensimm.py", line 55, in <module>
assert(b>7)
AssertionError
[Finished in 0.6s]

17、python2 和 python3 区别？列举 5 个

1、Python3 使用 print 必须要以小括号包裹打印内容，比如 print('hi')

Python2 既可以使用带小括号的方式，也可以使用一个空格来分隔打印内容，比如 print 'hi'

2、python2 range(1,10)返回列表，python3 中返回迭代器，节约内存

3、python2 中使用 ascii 编码，python 中使用 utf-8 编码

4、python2 中 unicode 表示字符串序列，str 表示字节序列

python3 中 str 表示字符串序列，byte 表示字节序列

5、python2 中为正常显示中文，引入 coding 声明，python3 中不需要

6、python2 中是 raw_input()函数，python3 中是 input()函数

18、列出 python 中可变数据类型和不可变数据类型，并简述原理

不可变数据类型：数值型、字符串型 string 和元组 tuple

不允许变量的值发生变化，如果改变了变量的值，相当于是新建了一个对象，而对于相同的值的对象，在内存中则只有一个对象（一个地址），如下图用 id()方法可以打印对象的 id

```
In [1]: a = 3
In [2]: b = 3
In [3]: id(a)
Out[3]: 1365598496
In [4]: id(b)
Out[4]: 1365598496
In [5]: _
```

可变数据类型：列表 list 和字典 dict;

允许变量的值发生变化，即如果对变量进行 append、+=等这种操作后，只是改变了变量的值，而不会新建一个对象，变量引用的对象的地址也不会变化，不过对于相同的值的不同对象，在内存中则会存在不同的对象，即每个对象都有自己的地址，相当于内存中对于同值的对象保存了多份，这里不存在引用计数，是实实在在的对象。

```
In [5]: a = [1,2]
In [6]: b = [1,2]
In [7]: id(a)
Out[7]: 2572957427336
In [8]: id(b)
Out[8]: 2572957321544
In [9]: _
```

19、s = "ajldjlajfdljfddd", 去重并从小到大排序输出"adfjl"

set 去重, 去重转成 list, 利用 sort 方法排序, reverse=False 是从小到大排

list 是不变数据类型, s.sort 时候没有返回值, 所以注释的代码写法不正确

```
1 s = "ajldjlajfdljfddd"
2 s = set(s)
3 s = list(s)
4 s.sort(reverse=False)
5 # s = s.sort(reverse=False)
6 res = "".join(s)
7 print(res)
8
```

adfjl
[Finished in 0.1s]

20、用 lambda 函数实现两个数相乘

```
8
9 sum=lambda a,b:a*b
10 print(sum(5,4))
```

20
[Finished in 0.1s]

表达式
参数

21、字典根据键从小到大排序

dic={"name":"zs","age":18,"city":"深圳","tel":"1362626627"}

```
In [4]: dic={"name":"zs","age":18,"city":"深圳","tel":"1362626627"}
In [5]: lis=sorted(dic.items(),key=lambda i:i[0],reverse=False)
In [6]: lis
Out[6]: [('age', 18), ('city', '深圳'), ('name', 'zs'), ('tel', '1362626627')]
In [7]: dict(lis)
Out[7]: {'age': 18, 'city': '深圳', 'name': 'zs', 'tel': '1362626627'}
```

根据键来排序
dict函数

22、利用 collections 库的 Counter 方法统计字符串每个单词出现的次数

"kjaljfj;ldsjafl;hdsllfdhg;lahfbl;hl;ahlf;h"


```

20 from collections import Counter
21 a = "kjalfj;ldsjafl;hdsllfdhg;lahfbl;hl;ahlf;h"
22 res=Counter(a)
23 print(res)
24
Counter({'l': 9, ';': 6, 'h': 6, 'f': 5, 'a': 4, 'j': 3, 'd': 3, 's': 2, 'k': 1, 'g': 1, 'b': 1})
[Finished in 0.1s]

```

23、字符串 a = "not 404 found 张三 99 深圳"，每个词中间是空格，用正则过滤掉英文和数字，最终输出 "张三 深圳"

```

25 import re
26 a = "not 404 found 张三 99 深圳"
27 list = a.split(" ")
28 print(list)
29 res=re.findall('\d+|[a-zA-Z]+',a)
30 for i in res:
31     if i in list:
32         list.remove(i)
33 new_str=" ".join(list)
34 print(res)
35 print(new_str)
36
37
38
['not', '404', 'found', '张三', '99', '深圳']
['not', '404', 'found', '99']
张三 深圳
[Finished in 0.1s]

```

连接多个匹配方式
匹配数字
匹配单词

顺便贴上匹配小数的代码，虽然能匹配，但是健壮性有待进一步确认

```

24 #
25 import re
26 a = "not 404 50.56 found 张三 99 深圳"
27 list = a.split(" ")
28 print(list)
29 res=re.findall('\d+\.\d*|[a-zA-Z]+',a)
30 for i in res:
31     if i in list:
32         list.remove(i)
33 new_str=" ".join(list)
34 print(res)
35 print(new_str)
36
37
38
['not', '404', '50.56', 'found', '张三', '99', '深圳']
['not', '404', '50.56', 'found', '99']
张三 深圳
[Finished in 0.1s]

```

匹配小数

24、filter 方法求出列表所有奇数并构造新列表，a = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

filter() 函数用于过滤序列，过滤掉不符合条件的元素，返回由符合条件元素组成的新列表。该接收两个参数，第一个为函数，第二个为序列，序列的每个元素作为参数传递给函数进行判，然后返回 True 或 False，最后将返回 True 的元素放到新列表

```
38 a = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
39 def fn(a):
40     return a%2==1
41 newlist = filter(fn, a)
42 newlist=[i for i in newlist]
43 print(newlist)
44
45
[1, 3, 5, 7, 9]
[Finished in 0.1s]
```

25、列表推导式求列表所有奇数并构造新列表，a = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

```
45 res = [i for i in a if i%2==1]
46 print(res)
[1, 3, 5, 7, 9]
[Finished in 0.1s]
```

26、正则 re.compile 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。

27、a= (1,) b=(1), c=("1") 分别是什么类型的数据？

```
In [14]: type((1))
Out[14]: int

In [15]: type(("1"))
Out[15]: str

In [16]: type((1,))
Out[16]: tuple
```

28、两个列表[1,5,7,9]和[2,2,6,8]合并为[1,2,2,3,6,7,8,9]extend 可以将另一个

集合中的元素逐一添加到列表中，区别于 append 整体添加。

```
48 list1=[1,5,7,9]
49 list2=[2,2,6,8]
50
51 list1.extend(list2)
52 print(list1)
53
54 list1.sort(reverse=False)
55 print(list1)
56
```

[1, 5, 7, 9, 2, 2, 6, 8]
[1, 2, 2, 5, 6, 7, 8, 9]
[Finished in 0.1s]

合并
排序

29、log 日志中，我们需要用时间戳记录 error,warning 等的发生时间，请用 datetime 模块打印当前时间戳 “2018-04-01 11:38:54” 顺便把星期的代码也贴上了。

```
116 # datetime模块
117 import datetime
118 a=str(datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')) + ' 星期: ' + str(datetime.datetime.now().isoweekday())
119 print(a)
```

2018-04-01 11:41:55 星期: 7
[Finished in 0.5s]

日期
星期

30、写一段自定义异常代码自定义异常用 raise 抛出异常。

```
104
2 105 # raise自定义异常
106 def fn():
107     try:
108         for i in range(5):
109             if i>2:
110                 raise Exception("数字大于2了")
111     except Exception as ret:
112         print(ret)
113 fn()
```

数字大于2了
[Finished in 0.1s]

31、正则表达式匹配中, (.*?) 和 (.*?) 匹配区别?

(.*) 是贪婪匹配, 会把满足正则的尽可能多的往后匹配。

(.*?) 是非贪婪匹配, 会把满足正则的尽可能少匹配。

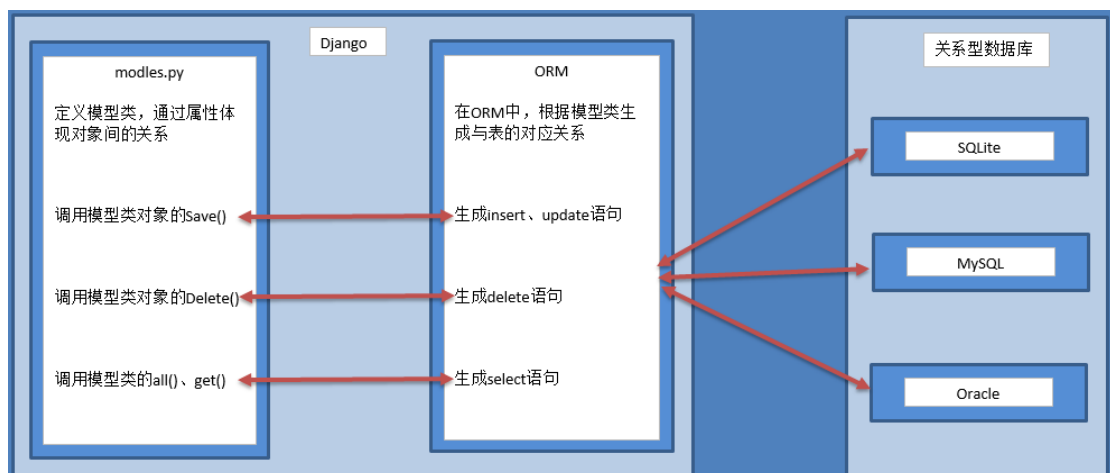
```
97 s("<a>哈哈</a><a>呵呵</a>")
98 import re
99 res1=re.findall("<a>(.*?)</a>",s)
100 print("贪婪匹配",res1)
101 res2=re.findall("<a>(.*?)</a>",s)
102 print("非贪婪匹配",res2)
```

贪婪匹配 ['哈哈<a>呵呵']
非贪婪匹配 ['哈哈', '呵呵']
[Finished in 0.1s]

32、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。



33、[[1,2],[3,4],[5,6]]一行代码展开该列表, 得出[1,2,3,4,5,6]列表推导式的骚操作。

运行过程: for i in a, 每个 i 是 [1,2], [3,4], [5,6], for j in i, 每个 j 就是 1,2,3,4,5,6,

合并后就是结果。

```
120
121 # 展开列表
122 a=[[1,2],[3,4],[5,6]]
123 x=[j for i in a for j in i]
124 print(x)

[1, 2, 3, 4, 5, 6]
[Finished in 0.1s]
```

还有更骚的方法，将列表转成 numpy 矩阵，通过 numpy 的 flatten () 方法，代码永远是只有更骚，没有最骚。

```
121 # 展开列表
122 a=[[1,2],[3,4],[5,6]]
123 x=[j for i in a for j in i]
124 print(x)
125
126
127 import numpy as np
128 b=np.array(a).flatten().tolist()
129 print(b)
130

[1, 2, 3, 4, 5, 6]
[1, 2, 3, 4, 5, 6]
[Finished in 0.6s]
```

34、x="abc",y="def",z=["d","e","f"],分别求出 x.join(y)和 x.join(z)返回的结果
join()括号里面的可迭代对象，x 插入可迭代对象中间，形成字符串，结果一致，有没有突然感觉字符串的常见操作都不会玩了。

顺便建议大家学下 os.path.join()方法，拼接路径经常用到，也用到了 join,和字符串操作中的 join 有什么区别，该问题大家可以查阅相关文档，后期会有答案。

```
86 x="abc"
87 y="def"
88 z=["d","e","f"]
89
90 m=x.join(y)
91 n=x.join(z)
92 print(m)
93 print(n)

dabceabcf
dabceabcf
[Finished in 0.1s]
```

35、举例说明异常模块中 try except else finally 的相关意义

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。

```
132 try:
133     num = 100
134     print(num)
135 except NameError as errorMsg:
136     print('产生错误了:%s'%errorMsg)
137 else:
138     print('没有捕获到异常，则执行该语句')
139
140 try:
141     num = 100
142     print(num)
143 except NameError as errorMsg:
144     print('产生错误了:%s'%errorMsg)
145 finally:
146     print('不管是否捕获到异常，都执行该句')
147

100
没有捕获到异常，则执行该语句
100
不管是否捕获到异常，都执行该句
[Finished in 0.6s]
```

36、举例说明 zip () 函数用法

zip()函数在运算时，会以一个或多个序列（可迭代对象）做为参数，返回一个元组的列表。同时将这些序列中并排的元素配对。

zip()参数可以接受任何类型的序列，同时也可以有两个以上的参数;当传入参数的长度不同时，zip 能自动以最短序列长度为准进行截取，获得元组。

```
155 #
156 a = [1,2]
157 b = [3,4]
158 res=[i for i in zip(a,b)]
159 print(res)
160
161 a = (1,2)
162 b = (3,4)
163 res=[i for i in zip(a,b)]
164 print(res)
165
166 a = "ab"
167 b = "xyz"
168 res=[i for i in zip(a,b)]
169 print(res)
```

← 列表

← 元组

← 字符串

```
[(1, 3), (2, 4)]
[(1, 3), (2, 4)]
[('a', 'x'), ('b', 'y')]
[Finished in 0.2s]
```

37、a="张明 98 分"，用 re.sub，将 98 替换为 100

```
171 import re
172 a = "张明 98分"
173 ret = re.sub(r"\d+", "100", a)
174 print(ret)
```

张明 100分

[Finished in 0.1s]

38、a="hello"和 b="你好"编码成 bytes 类型

```
175  
> 数  
176 a=b"hello"  
tes  
177 b="哈哈".encode()  
> j  
178 print(a,b)  
179 print(type(a),type(b))  
张明 100分  
b'hello' b'\xe5\x93\x88\xe5\x93\x88'  
<class 'bytes'> <class 'bytes'>  
[Finished in 0.1s]
```

39、[1,2,3]+[4,5,6]的结果是多少？

两个列表相加，等价于 extend。

```
180  
> j  
181 a = [1,2,3]  
182 b = [4,5,6]  
183 res=a+b  
184 print(res)  
[1, 2, 3, 4, 5, 6]  
[Finished in 0.1s]
```

40、提高 python 运行效率的方法

- 1、使用生成器，因为可以节约大量内存；
- 2、循环代码优化，避免过多重复代码的执行；
- 3、核心模块用 Cython PyPy 等，提高效率；
- 4、多进程、多线程、协程；
- 5、多个 if elif 条件判断，可以把最有可能先发生的条件放到前面写，这样可以减少程序判断的次数，提高效率。

41、遇到 bug 如何处理

- 1、细节上的错误，通过 print () 打印，能执行到 print () 说明一般上面的代码没有

问题，分段检测程序是否有问题，如果是js的话可以 alert 或 console.log

2、如果涉及一些第三方框架，会去查官方文档或者一些技术博客。

3、对于 bug 的管理与归类总结，一般测试将测试出的 bug 用 teambin 等 bug 管理工具进行记录，然后我们会一条一条进行修改，修改的过程也是理解业务逻辑和提高自己编程逻辑缜密性的方法，我也都会收藏做一些笔记记录。

4、导包问题、城市定位多音字造成的显示错误问题。

42、正则匹配，匹配日期 2018-03-20

url='https://sycm.taobao.com/bda/tradinganaly/overview/get_summary.json?

dateRange=2018-03-20%7C2018-03-

20&dateType=recent1&device=1&token=ff25b109b&_ =1521595613462' 仍有

同学问正则，其实匹配并不难，提取一段特征语句，用 (.*?) 匹配即可。

```
30 import re
31
32 url='https://sycm.taobao.com/bda/tradinganaly/overview/get_summary.json?dateRange=2018-03-20%7C2018-03-20&dateType=recent1&device=1&token=ff25b109b&_ =1521595613462'
33
34 result = re.findall(r'dateRange=(.*?)%7C(.*?)&', url)
35 print(result)
[('2018-03-20', '2018-03-20')]
```

43、list=[2,3,5,4,9,6]，从小到大排序，不许用 sort，输出[2,3,4,5,6,9]

利用 min()方法求出最小值，原列表删除最小值，新列表加入最小值，递归调用获取最小值的函数，反复操作。

```
list_
1.ht
FOLDE
> 数
tes
76 list=[2,3,5,4,9,6]
77 new_list=[]
78 def get_min(list):
79     # 获取列表最小值
80     a=min(list)
81     # 删除最小值
82     list.remove(a)
83     # 将最小值加入新列表
84     new_list.append(a)
85     # 保证最后列里面有值，递归调用获取最小值
86     # 直到所有值获取完，并加入新列表返回
87     if len(list)>0:
88         get_min(list)
89     return new_list
90
91 new_list=get_min(list)
92 print(new_list)
[2, 3, 4, 5, 6, 9]
```

44、写一个单列模式

因为创建对象时 `__new__` 方法执行，并且必须 `return` 返回实例化出来的对象所
`cls.__instance` 是否存在，不存在的话就创建对象，存在的话就返回该对象，来保证只
有一个实例对象存在（单列），打印 ID，值一样，说明对象同一个。

```
37 # 实例化一个单例
38 class Singleton(object):
39     __instance = None
40
41     def __new__(cls, age, name):
42         #如果类属性__instance的值为None.
43         #那么就创建一个对象，并且赋值为这个对象的引用，保证下次调用这个方法时
44         #能够知道之前已经创建过对象了，这样就保证了只有1个对象
45         if not cls.__instance:
46             cls.__instance = object.__new__(cls)
47         return cls.__instance
48
49 a = Singleton(18, "dongGe")
50 b = Singleton(8, "dongGe")
51
52 print(id(a))
53 print(id(b))
54
55 a.age = 19 #给a指向的对象添加一个属性
56 print(b.age)#获取b指向的对象的age属性
57
2223334542920
2223334542920
19
```

45、保留两位小数

题目本身只有 `a="%.03f"%1.3335`,让计算 `a` 的结果，为了扩充保留小数的思路，提供
`round` 方法（数值，保留位数）。

```
59 # 保留2位小数
60 a = "%.03f"%1.3335
61 print(a,type(a))
62 b=round(float(a),1)
63 print(b)
64 b=round(float(a),2)
65 print(b)
66
67 A = zip(("a","b","c","d","e"),(1,2,3,4,5))
68 A0 = dict(A)
69 # print(A0)

1.333 <class 'str'>
1.3
1.33
```

46、求三个方法打印结果

`fn("one",1)` 直接将键值对传给字典。

`fn("two",2)`因为字典在内存中是可变数据类型，所以指向同一个地址，传了新的参数后，会相当于给字典增加键值对。

`fn("three",3,{})`因为传了一个新字典，所以不再是原先默认参数的字典。

```
113 def fn(k,v,dic={}):
114     dic[k]=v
115     print(dic)
116 fn("one",1)
117 fn("two",2)
118 fn("three",3,{})
119
```

```
{'one': 1}
{'one': 1, 'two': 2}
{'three': 3}
[Finished in 0.3s]
```

47、分别从前端、后端、数据库阐述 web 项目的性能优化

前端优化：

- 1、减少 http 请求、例如制作精灵图；
- 2、html 和 CSS 放在页面上部，javascript 放在页面下面，因为 js 加载比 HTML 和 Css 加载慢，所以要优先加载 html 和 css,以防页面显示不全，性能差，也影响用户体验差。

后端优化：

- 1、缓存存储读写次数高，变化少的数据，比如网站首页的信息、商品的信息等。应用程序读取数据时，一般是先从缓存中读取，如果读取不到或数据已失效，再访问磁盘数据库，并将数据再次写入缓存；
- 2、异步方式，如果有耗时操作，可以采用异步，比如 celery；
- 3、代码优化，避免循环和判断次数太多，如果多个 if else 判断，优先判断最有可能先

发生的情况。

数据库优化：

- 1、如有条件，数据可以存放于 redis，读取速度快；
- 2、建立索引、外键等。

48、使用 pop 和 del 删除字典中的"name"字段，dic={"name":"zs","age":18}

```
158
159 dic = {"name":"zs","age":18}
160 dic.pop("name")
161 print(dic)
162 dic = {"name":"zs","age":18}
163 del dic["name"]
164 print(dic)

{'age': 18}
{'age': 18}
[Finished in 0.3s]
```

49、计算代码运行结果，zip 函数历史文章已经说了，得出[("a",1),("b",2),
("c",3),("d",4),("e",5)]

```
72 A = zip(("a","b","c","d","e"),(1,2,3,4,5))
73 A0 = dict(A)
74 A1=range(10)
75 A2=[i for i in A1 if i in A0]
76 A3=[A0[s] for s in A0]
77 print("A0",A0)
78 print(list(zip(("a","b","c","d","e"),(1,2,3,4,5))))
79 print(A2)
80 print(A3)

A0 {'a': 1, 'b': 2, 'c': 3, 'd': 4, 'e': 5}
[('a', 1), ('b', 2), ('c', 3), ('d', 4), ('e', 5)]
[]
[1, 2, 3, 4, 5]
```

dict()创建字典新方法。

```
165
166 s = dict([["name","zs"],["age",18]])
167 print(s)
168
169 s = dict(("name","zs"),("age",18))
170 print(s)
```

列表

元组

```
A0 {'a': 1, 'b': 2, 'c': 3, 'd': 4, 'e': 5}
[('a', 1), ('b', 2), ('c', 3), ('d', 4), ('e', 5)]
[]
[1, 2, 3, 4, 5]
{'age': 18}
{'age': 18}
{'name': 'zs', 'age': 18}
{'name': 'zs', 'age': 18}
[Finished in 0.6s]
```

50、简述同源策略

同源策略需要同时满足以下三点要求：

- 1) 协议相同
- 2) 域名相同
- 3) 端口相同

http:www.test.com 与 https:www.test.com 不同源——协议不同

http:www.test.com 与 http:www.admin.com 不同源——域名不同

http:www.test.com 与 http:www.test.com:8081 不同源——端口不同

只要不满足其中任意一个要求，就不符合同源策略，就会出现“跨域”。

51、简述 cookie 和 session 的区别

- 1, session 在服务器端，cookie 在客户端（浏览器）；
- 2、session 的运行依赖 session id，而 session id 是存在 cookie 中的，也就是说，如果浏览器禁用了 cookie，同时 session 也会失效，存储 Session 时，键与 Cookie 中的 sessionid 相同，值是开发人员设置的键值对信息，进行了 base64 编码，过期时

间由开发人员设置;

3、cookie 安全性比 session 差。

52、简述多线程、多进程

进程:

- 1、操作系统进行资源分配和调度的基本单位, 多个进程之间相互独立;
- 2、稳定性好, 如果一个进程崩溃, 不影响其他进程, 但是进程消耗资源大, 开启的进程数量有限制。

线程:

- 1、CPU 进行资源分配和调度的基本单位, 线程是进程的一部分, 是比进程更小的能独立运行的基本单位, 一个进程下的多个线程可以共享该进程的所有资源;
- 2、如果 IO 操作密集, 则可以多线程运行效率高, 缺点是如果一个线程崩溃, 都会造成进程的崩溃。

应用:

- 1、IO 密集的用多线程, 在用户输入, sleep 时候, 可以切换到其他线程执行, 减少等待的时间;
- 2、CPU 密集的用多进程, 因为假如 IO 操作少, 用多线程的话, 因为线程共享一个全局解释器锁, 当前运行的线程会霸占 GIL, 其他线程没有 GIL, 就不能充分利用多核 CPU 的优势。

53、简述 any()和 all()方法

any():只要迭代器中有一个元素为真就为真。

all():迭代器中所有的判断项返回都是真, 结果才为真。

python 中什么元素为假?

答案: (0, 空字符串, 空列表、空字典、空元组、None, False)

```
In [3]: bool(0)
Out[3]: False

In [4]: bool("")
Out[4]: False

In [5]: bool([])
Out[5]: False

In [6]: bool(())
Out[6]: False

In [7]: bool({})
Out[7]: False

In [8]: bool(None)
Out[8]: False

In [9]:
```

测试 all()和 any()方法。

```
In [9]: a = [True, False]

In [10]: any(a)
Out[10]: True

In [11]: all(a)
Out[11]: False

In [12]: a = ""

In [13]: any(a)
Out[13]: False

In [14]: b=['good', 'good', 'good', 'bad']

In [15]: all(b)
Out[15]: True

In [16]: b=['good', 'good', 'good', '']

In [17]: all("b")
Out[17]: True

In [18]: any("b")
Out[18]: True

In [19]: b
Out[19]: ['good', 'good', 'good', '']

In [20]: b=['good', 'good', 'good', ""]

In [21]: b
Out[21]: ['good', 'good', 'good', '']

In [22]: all(b)
Out[22]: False
```

这里注意是"b"

54、IOError、AttributeError、ImportError、IndentationError、IndexError、KeyError、SyntaxError、NameError 分别代表什么异常

IOError: 输入输出异常。

AttributeError: 试图访问一个对象没有的属性。

ImportError: 无法引入模块或包，基本是路径问题。

IndentationError: 语法错误，代码没有正确的对齐。

IndexError: 下标索引超出序列边界。

KeyError: 试图访问你字典里不存在的键。

SyntaxError: Python 代码逻辑语法出错，不能执行。

NameError: 使用一个还未赋予对象的变量。

55、python 中 copy 和 deepcopy 区别

1、复制不可变数据类型，不管 copy 还是 deepcopy,都是同一个地址当浅复制的值是不可变对象（数值，字符串，元组）时和=“赋值”的情况一样，对象的 id 值与浅复制原来的值相同。

```
272
list_ 273 a = "哈哈"
1.ht 274 b = a
FOLDE 275 c = copy.copy(a)
> 数 276 d = copy.deepcopy(a)
tes 277 print(a,id(a))
> i 278 print(b,id(b))
279 print(c,id(c))
280 print(d,id(d))

哈哈 2348789079624
哈哈 2348789079624
哈哈 2348789079624
哈哈 2348789079624
[Finished in 0.5s]
```

2、复制的值是可变对象（列表和字典）

浅拷贝 copy 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），改变原来的值 中的复杂子对象的值 ，会影响浅复制的值。

深拷贝 deepcopy：完全复制独立，包括内层列表和字典。

```
# copy和deepcopy深浅复制
# list = [1,2,{"name":"zs"}]
list = [1,2,[3,4]]
a=copy.copy(list)
b=copy.deepcopy(list)
# 外层列表: [1,2,[3,4]]
# 内层列表: [3,4]
# 复杂子对象: [3,4],我们认为包含嵌套结构的内层列表(字典)为复杂子对象
# 简单子对象: 1,2
print("测试原始数据和copy和deepcopy后的结果及ID地址")
print("结果表明对于外层列表来说，三者是独立的对象")
print("原始数据和id",list,id(list))
print("原始数据和id",a,id(a))
print("原始数据和id",b,id(b))
print("*****")
print("测试修改外层列表的简单子对象，也就是修改1或者2")
print("结果表明修改了原始list之后,a和b并没有随之改变，符合我们的正常逻辑，因为是三个不同的对象")
list[0]=10
print("将1改成10结果",list)
print("将1改成10结果",a)
print("将1改成10结果",b)
print("*****")
print("测试内层列表的值的修改，也就是测试复杂子对象的值的修改")
print("结果表明copy浅拷贝并没有真正将内层列表(字典)独立拷贝出来，导致修改了list内层列表(字典)后a的内层列表(字典)值也变了")
print("结果表明deepcopy深拷贝可以将内层列表和(字典)独立拷贝出来,所以b的内层列表(字典)值不变")
list[2][0]=5
print("将3改成5结果",list)
print("将3改成5结果",a)
print("将3改成5结果",b)
```

```
测试原始数据和copy和deepcopy后的结果及ID地址
结果表明对于外层列表来说，三者是独立的对象
原始数据和id [1, 2, [3, 4]] 2465779174664
原始数据和id [1, 2, [3, 4]] 2465784158664
原始数据和id [1, 2, [3, 4]] 2465779174856
*****
测试修改外层列表的简单子对象，也就是修改1或者2
结果表明修改了原始list之后,a和b并没有随之改变，符合我们的正常逻辑，因为是三个不同的对象
将1改成10结果 [10, 2, [3, 4]]
将1改成10结果 [1, 2, [3, 4]]
将1改成10结果 [1, 2, [3, 4]]
*****
测试内层列表的值的修改，也就是测试复杂子对象的值的修改
结果表明copy浅拷贝并没有真正将内层列表(字典)独立拷贝出来，导致修改了list内层列表(字典)后a的内层列表(字典)值也变了
结果表明deepcopy深拷贝可以将内层列表和(字典)独立拷贝出来,所以b的内层列表(字典)值不变
将3改成5结果 [10, 2, [5, 4]]
将3改成5结果 [1, 2, [5, 4]]
将3改成5结果 [1, 2, [3, 4]]
```

56、列出几种魔法方法并简要介绍用途

__init__:对象初始化方法

__new__:创建对象时候执行的方法，单列模式会用到

__str__:当使用 print 输出对象的时候，只要自己定义了__str__(self)方法，那么就会打印从在这个方法中 return 的数据

__del__:删除对象执行的方法

57、C:\Users\ry-wu.junya\Desktop>python 1.py 22 33 命令行启动程序并传参，print(sys.argv)会输出什么数据？

文件名和参数构成的列表。

```
C:\Users\ry-wu.junya\Desktop>python 1.py 22 33  
[ '1.py', '22', '33' ]
```

58、请将[i for i in range(3)]改成生成器

生成器是特殊的迭代器：

- 1、列表表达式的【】改为（）即可变成生成器；
- 2、函数在返回值得时候出现 yield 就变成生成器，而不是函数了。

中括号换成小括号即可

```
In [1]: a = (i for i in range(3))  
In [2]: type(a)  
Out[2]: generator  
In [3]: _
```

59、a = " hehheh ",去除收尾空格

```
In [3]: a = " hehheh "  
In [4]: a.strip()  
Out[4]: 'hehheh'  
In [5]: _
```

60、举例 sort 和 sorted 对列表排序，list=[0,-1,3,-10,5,9]

```
187
188 # 列表排序
189 list = [0,-1,3,-10,5,9]
190 list.sort(reverse=False)
191 print("list.sort在list基础上修改，无返回值",list)
192
193 list = [0,-1,3,-10,5,9]
194 res = sorted(list,reverse=False)
195 print("sorted有返回值是新的list",list)
196 print("返回值",res)
```

数字大于2了
list.sort在list基础上修改，无返回值 [-10, -1, 0, 3, 5, 9]
sorted有返回值是新的list [0, -1, 3, -10, 5, 9]
返回值 [-10, -1, 0, 3, 5, 9]

61、对 list 排序 foo = [-5,8,0,4,9,-4,-20,-2,8,2,-4],使用 lambda 函数从小到大排序

```
199
200 foo = [-5,8,0,4,9,-4,-20,-2,8,2,-4]
201 a=sorted(foo,key=lambda x:x)
202 # foo.sort()
203 print(a)
204
```

数字大于2了
[-20, -5, -4, -4, -2, 0, 2, 4, 8, 8, 9]

62、使用 lambda 函数对 list 排序 foo = [-5,8,0,4,9,-4,-20,-2,8,2,-4]，输出结果为[0,2,4,8,8,9,-2,-4,-4,-5,-20]，正数从小到大，负数从大到小

(传两个条件， $x < 0$ 和 $\text{abs}(x)$)

```
199
200 foo = [-5,8,0,4,9,-4,-20,-2,8,2,-4]
201 a=sorted(foo,key=lambda x:(x<0,abs(x)))
202 # foo.sort()
203 print(a)
204
```

数字大于2了
[0, 2, 4, 8, 8, 9, -2, -4, -4, -5, -20]

63、列表嵌套字典的排序，分别根据年龄和姓名排序

```
foo = [{"name":"zs","age":19}, {"name":"ll","age":54},  
       {"name":"wa","age":17}, {"name":"df","age":23}]
```

```

204
205 foo = [{"name": "zs", "age": 19}, {"name": "ll", "age": 54},
206         {"name": "wa", "age": 17}, {"name": "df", "age": 23}]
207 a=sorted(foo, key=Lambda x:x["age"], reverse=True)
208 print(a)
209 a=sorted(foo, key=Lambda x:x["name"])
210 print(a)
211
数字大于2了
[8, 0, 4, 9, 8, 2, -5, -4, -20, -2, -4]
[{'name': 'll', 'age': 54}, {'name': 'df', 'age': 23}, {'name': 'zs', 'age': 19}, {'name': 'wa', 'age': 17}]
[{'name': 'df', 'age': 23}, {'name': 'll', 'age': 54}, {'name': 'wa', 'age': 17}, {'name': 'zs', 'age': 19}]
[Finished in 0.1s]

```

年龄从大到小

姓名从小到大

64、列表嵌套元组，分别按字母和数字排序

```

213 foo = [("zs", 19), ("ll", 54),
214         ("wa", 17), ("df", 23)]
215 a=sorted(foo, key=Lambda x:x[1], reverse=True)
216 print(a)
217 a=sorted(foo, key=Lambda x:x[0])
218 print(a)
数字大于2了
[8, 0, 4, 9, 8, 2, -5, -4, -20, -2, -4]
[('ll', 54), ('df', 23), ('zs', 19), ('wa', 17)]
[('df', 23), ('ll', 54), ('wa', 17), ('zs', 19)]
[Finished in 0.2s]

```

65、列表嵌套列表排序，年龄数字相同怎么办？

```

213 foo = [{"zs", 19}, {"ll", 54},
214         ["wa", 23], ["df", 23], ["xf", 23]]
215 a=sorted(foo, key=Lambda x:(x[1], x[0]))
216 print(a)
217 a=sorted(foo, key=Lambda x:x[0])
218 print(a)
数字大于2了
[8, 0, 4, 9, 8, 2, -5, -4, -20, -2, -4]
[['zs', 19], ['df', 23], ['wa', 23], ['xf', 23], ['ll', 54]]
[['df', 23], ['ll', 54], ['wa', 23], ['xf', 23], ['zs', 19]]

```

年龄相同怎么办？
添加参数，按字母排序

66、根据键对字典排序（方法一，zip 函数）

```

220
221 dic = {"name": "zs", "sex": "man", "city": "bj"}
222 foo=zip(dic.keys(), dic.values())
223 foo = [i for i in foo]
224 print("字典转成列表嵌套元组", foo)
225 b=sorted(foo, key=Lambda x:x[0])
226 print("根据键排序", b)
227 new_dic = {i[0]:i[1] for i in b}
228 print("字典推导式构造新字典", new_dic)
229
230
数字大于2了
字典转成列表嵌套元组 [('name', 'zs'), ('sex', 'man'), ('city', 'bj')]
根据键排序 [('city', 'bj'), ('name', 'zs'), ('sex', 'man')]
字典推导式构造新字典 {'city': 'bj', 'name': 'zs', 'sex': 'man'}

```

字典转列表嵌套元组

字典嵌套元组排序

排序完构造新字典

67、根据键对字典排序（方法二,不用 zip)

有没有发现 `dic.items` 和 `zip(dic.keys(),dic.values())`都是为了构造列表嵌套字典的结构，方便后面用 `sorted()`构造排序规则。

```
> 数 231 dic = {"name":"zs","sex":"man","city":"bj"}
    232 print("字典转成列表嵌套元组",foo)
    233 print(dic.items())
    234 b=sorted(dic.items(),key=lambda x:x[0])
    235 print("根据键排序",b)
    236 new_dic = {i[0]:i[1] for i in b}
    237 print("字典推导式构造新字典",new_dic)
```

数字大于2了
字典转成列表嵌套元组 [('name', 'zs'), ('sex', 'man'), ('city', 'bj')]
dict_items([('name', 'zs'), ('sex', 'man'), ('city', 'bj')])
根据键排序 [('city', 'bj'), ('name', 'zs'), ('sex', 'man')]
字典推导式构造新字典 {'city': 'bj', 'name': 'zs', 'sex': 'man'}
[Finished in 0.1s]

68、列表推导式、字典推导式、生成器

```
FOLDE 239 import random
> 数 240 td_list=[i for i in range(10)]
    241 print("列表推导式",td_list,type(td_list))
    242
    243 ge_list=(i for i in range(10))
    244 print("生成器",ge_list)
    245
    246 dic={k:random.randint(4,9) for k in ["a","b","c","d"]}
    247 print("字典推导式",dic,type(dic))
```

数字大于2了
列表推导式 [0, 1, 2, 3, 4, 5, 6, 7, 8, 9] <class 'list'>
生成器 <generator object <genexpr> at 0x0000018F61A0AF68>
字典推导式 {'a': 7, 'b': 6, 'c': 9, 'd': 8} <class 'dict'>
[Finished in 0.2s]

69、根据字符串长度排序，看排序是否灵活运用

```
> 数 249
    250 s = ["ab","abc","a","djkj"]
    251 b = sorted(s,key=lambda x:len(x))
    252 print(b,s)
    253 s.sort(key=len)
    254 print(s)
    255
```

sorted有返回值，不改变s本身
sort无返回值，在s自身修改

数字大于2了
['a', 'ab', 'abc', 'djkj'] ['ab', 'abc', 'a', 'djkj']
['a', 'ab', 'abc', 'djkj']
[Finished in 0.1s]

70、`s="info:xiaoZhang 33 shandong"`,用正则切分字符串输出['info', 'xiaoZhang', '33', 'shandong']

| 表示或，根据冒号或者空格切分。

```
269 s = "info:xiaoZhang 33 shandong"
270 res=re.split(r":| ",s)
271 print(res)
```

数字大于2了

```
['info', 'xiaoZhang', '33', 'shandong']
[Finished in 0.1s]
```

71、正则匹配以 163.com 结尾的邮箱

```
321
> 数 322 email_list = ["xiaoWang@163.com", "xiaoWang@163.comheihei", ".com.xiaowang@qq.com"]
323
> tes 324 for email in email_list:
325     ret = re.match("[\w]{4,20}@163\.com$", email)
326     if ret:
327         print("%s 是符合规定的邮件地址,匹配后的结果是:%s" % (email, ret.group()))
328     else:
329         print("%s 不符合要求" % email)
```

数字大于2了

```
xiaoWang@163.com 是符合规定的邮件地址,匹配后的结果是:xiaoWang@163.com
xiaoWang@163.comheihei 不符合要求
.com.xiaowang@qq.com 不符合要求
```

72、递归求和

```
111 273 # 递归完成1+2+3+...+10的和
1.ht 274 def get_sum(num):
X list_ 275     if num>=1:
FOLDE 276         res=num+get_sum(num-1)
> 数 277     else:
tes 278         res = 0
> .i 279
280     return res
281 res=get_sum(10)
282 print(res)
```

数字大于2了

```
['info', 'xiaoZhang', '33', 'shandong']
55
[Finished in 1.9s]
```

73、python 字典和json 字符串相互转化方法

json.dumps()字典转 json 字符串， json.loads()json 转字典。

```
288 import json
289 dic = {"name": "zs"}
290 res = json.dumps(dic)
291 print(res, type(res))
292 ret = json.loads(res)
293 print(ret, type(ret))
294
```

数字大于2了

```
{"name": "zs"} <class 'str'>
{'name': 'zs'} <class 'dict'>
[Finished in 0.2s]
```

74、统计字符串中某字符出现次数

```
294
295 str = "张三 美国 张三 哈哈 张 三"
296 res = str.count("张三")
297 print(res)
```

数字大于2了

2

[Finished in 0.5s]

75、字符串转化大小写

```
298
299 str = "HHHuuu"
300 print(str.upper())
301 print(str.lower())
```

数字大于2了

2

HHHUUU

hhhUUU

[Finished in 1.9s]

76、用两种方法去空格

```
303 str = "hello world ha ha"
304 res=str.replace(" ","")
305 print(res)
306
307 list=str.split(" ")
308 res="".join(list)
309 print(res)
```

数字大于2了
helloworldhaha
helloworldhaha

77、正则匹配不是以 4 和 7 结尾的手机号

```
313 tels = ["13100001234", "18912344321", "10086", "18800007777"]
314 for tel in tels:
315     ret = re.match("1\d{9}[0-3,5-6,8-9]", tel)
316     if ret:
317         print("想要的结果",ret.group())
318     else:
319         print("%s 不是想要的手机号" % tel)
```

数字大于2了
13100001234 不是想要的手机号 ← 4结果
想要的结果 18912344321
10086 不是想要的手机号 ← 不是手机号
18800007777 不是想要的手机号 ← 7结尾
[Finished in 1.9s]

78、python 正则中 search 和 match

```
54 import re
55 s = "小明年龄18岁 工资10000"
56 res=re.search(r"\d+",s).group()
57 print("search结果",res)
58
59 res=re.findall(r"\d+",s)
60 print("findall结果",res)
61
62 res=re.match("小明",s).group()
63 print("match结果",res)
64
65 res=re.match(r"\d+",s)
66 print("试错，不加group为none,匹配不到",res)
67
68 res=re.match("工资",s).group()
69 print("match结果",res)
```

search匹配到第一个匹配到的数据
满足正则，都匹配，不用加group
匹配以"小明"开头的字符串，并匹配出小明
不加group是不对的
工资不是字符串开头，匹配不到，报错

search结果 18
findall结果 ['18', '10000']
match结果 小明
试错，不加group为none,匹配不到 None
Traceback (most recent call last):
File "C:\Users\ry-wu.junya\Desktop\面试题.py", line 68, in <module>
res=re.match("工资",s).group()
AttributeError: 'NoneType' object has no attribute 'group'
[Finished in 0.2s with exit code 1]
[cmd: ['C:\python36\python.exe', '-u', 'C:\Users\ry-wu.junya\Desktop\面试题.py']]

79、正则表达式匹配第一个 URL

findall 结果无需加 group(),search 需要加 group()提取。


```
354 #
355 #
356 s = ''
357 res=re.findall(r"https://.*?\.",s)[0]
358 print(res)
359 res=re.search(r"https://.*?\.",s)
360 print(res.group())
```

数字大于2了
https://rpic.douyucdn.cn/appCovers/2016/11/13/1213973_201611131917_small.jpg
https://rpic.douyucdn.cn/appCovers/2016/11/13/1213973_201611131917_small.jpg
[Finished in 0.2s]

80、正则匹配中文

```
364
365 title = '你好, hello, 世界'
366 pattern = re.compile(r'[\u4e00-\u9fa5]+')
367 result = pattern.findall(title)
368
369 print (result)
```

数字大于2了
['你好', '世界']
[Finished in 1.9s]

81、r、r+、rb、rb+文件打开模式区别

模式较多，比较下背背记记即可。

访问模式	说明
r	以只读方式打开文件。文件的指针将会放在文件的开头。这是默认模式。
w	打开一个文件只用于写入。如果该文件已存在则将其覆盖。如果该文件不存在，创建新文件。
a	打开一个文件用于追加。如果该文件已存在，文件指针将会放在文件的结尾。也就是说，新的内容将会被写入到已有内容之后。如果该文件不存在，创建新文件进行写入。
rb	以二进制格式打开一个文件用于只读。文件指针将会放在文件的开头。这是默认模式。
wb	以二进制格式打开一个文件只用于写入。如果该文件已存在则将其覆盖。如果该文件不存在，创建新文件。
ab	以二进制格式打开一个文件用于追加。如果该文件已存在，文件指针将会放在文件的结尾。也就是说，新的内容将会被写入到已有内容之后。如果该文件不存在，创建新文件进行写入。
r+	打开一个文件用于读写。文件指针将会放在文件的开头。
w+	打开一个文件用于读写。如果该文件已存在则将其覆盖。如果该文件不存在，创建新文件。
a+	打开一个文件用于读写。如果该文件已存在，文件指针将会放在文件的结尾。文件打开时会是追加模式。如果该文件不存在，创建新文件用于读写。
rb+	以二进制格式打开一个文件用于读写。文件指针将会放在文件的开头。
wb+	以二进制格式打开一个文件用于读写。如果该文件已存在则将其覆盖。如果该文件不存在，创建新文件。
ab+	以二进制格式打开一个文件用于追加。如果该文件已存在，文件指针将会放在文件的结尾。如果该文件不存在，创建新文件用于读写。

82、正则表达式匹配出<html><h1>www.itcast.cn</h1></html>

前面的<>和后面的<>是对应的，可以用此方法。

```
372 |
373 labels = ["<html><h1>www.itcast.cn</h1></html>", "<html><h1>www.itcast.cn</h2></html>"]
374 for label in labels:
375     ret = re.match(r"<(\w*)><(\w*)>.*?</\2></\1>", label)
376     if ret:
377         print("%s 是符合要求的标签" % ret.group())
378     else:
379         print("%s 不符合要求" % label)
```

数字大于2了
<html><h1>www.itcast.cn</h1></html> 是符合要求的标签
<html><h1>www.itcast.cn</h2></html> 不符合要求
[Finished in 0.2s]

83、python 传参数是传值还是传址？

Python 中函数参数是引用传递 (注意不是值传递)。对于不可变类型 (数值型、字符串、元组)，因变量不能修改，所以运算不会影响到变量自身；而对于可变类型 (列表字典) 来说，函数体运算可能会更改传入的参数变量。

```
381
382 def selfAdd(a):
383     a+=a
384
385 a_int = 1
386 print(a_int)
387 selfAdd(a_int)
388 print(a_int)
389
390 a_list = [1, 2]
391 print(a_list)
392 selfAdd(a_list)
393 print(a_list)
394
```

数字大于2了
1
1
[1, 2]
[1, 2, 1, 2]
[Finished in 0.2s]

84、求两个列表的交集、差集、并集

```
1 # 获取两个列表的交集、并集、差集
2 a = [1,2,3,4]
3 b = [4,3,5,6]
4 jj1=[i for i in a if i in b] # 在a中的i, 并且在b中, 就是交集
5 jj2=List(set(a).intersection(set(b)))
6
7 bj1=List(set(a).union(set(b))) # 用union方法
8
9 cj1= List(set(b).difference(set(a))) # b中有而a中没有的 非常高效!
10 cj2= List(set(a).difference(set(b))) # a中有而b中没有的 非常高效!
11 print("交集",jj1)
12 print("交集",jj2)
13
14 print("并集",bj1)
15 # print("并集",bj2)
16 print("差集",cj1)
17 print("差集",cj2)
18
```

交集 [3, 4]
交集 [3, 4]
并集 [1, 2, 3, 4, 5, 6]
差集 [5, 6]
差集 [1, 2]
[Finished in 0.2s]

85、生成 0-100 的随机数

random.random()生成 0-1 之间的随机小数，所以乘以 100。

```
> 18
> 19 import random
> 20 res1=100*random.random()
> 21 res2=random.choice(range(0,101))
> 22 res3=random.randint(1,100)
> 23 print(res1)
> 24 print(res2)
> 25 print(res3)
```

20.575902314280647
44
64

随机小数
随机整数

86、lambda 匿名函数好处

精简代码，lambda 省去了定义函数，map 省去了写 for 循环过程。

```
> 29 # lambda好处
> 30 a = ["苏州","中国","哈哈","","","日本","","","德国"]
> 31 res=list(map(lambda x:"填充值" if x==" " else x,a))
> 32 print(res)
```

['苏州', '中国', '哈哈', '填充值', '填充值', '日本', '填充值', '填充值', '德国']
[Finished in 0.2s]

87、单引号、双引号、三引号用法

1、单引号和双引号没有什么区别，不过单引号不用按 shift，打字稍微快一点。表示字符串的时候，单引号里面可以用双引号，而不用转义字符,反之亦然。

'She said:"Yes." ' or "She said: 'Yes.' "

2、但是如果直接用单引号扩住单引号，则需要转义，像这样：

'She said:\'Yes.\' '

3、三引号可以直接书写多行，通常用于大段，大篇幅的字符串。

```
"""
hello
world
"""
```

88、python 垃圾回收机制

python 垃圾回收主要以引用计数为主，标记-清除和分代清除为辅的机制，其中标记-清除和分代回收主要是为了处理循环引用的难题。

引用计数算法

当有 1 个变量保存了对象的引用时，此对象的引用计数就会加 1；

当使用 del 删除变量指向的对象时，如果对象的引用计数不为 1，比如 3，那么此时只会让这个引用计数减 1，即变为 2，当再次调用 del 时，变为 1，如果再调用 1 次 del，此时会真的把对象进行删除

```
80     print('__init__方法被调用')
81     self.__name = name
82
83     # 析构方法
84     # 当对象被删除时，会自动被调用
85     def __del__(self):
86         print("__del__方法被调用")
87         print("%s对象马上被干掉了..."%self.__name)
88
89     cat = Animal("波斯猫")
90     cat2 = cat
91     cat3 = cat
92     print(id(cat),id(cat2),id(cat3))
93     print("----马上 删除cat对象")
94     del cat
95
96     print(id(cat2),id(cat3))
97     print("----马上 删除cat2对象")
98     del cat2
99
100    print(id(cat3))
101    print("----马上 删除cat3对象")
102    del cat3
```

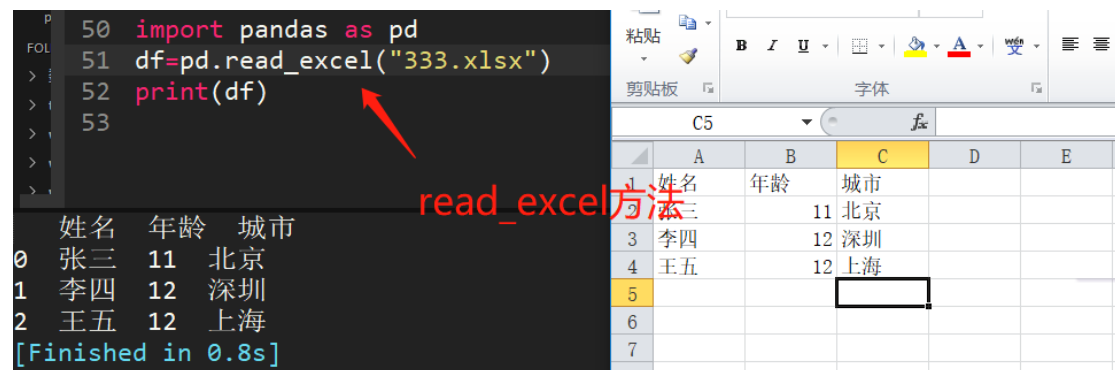
cat cat2和cat3，三者ID一样，说明是一个对象

__init__方法被调用
1438429206680 1438429206680 1438429206680
----马上 删除cat对象
1438429206680 1438429206680
----马上 删除cat2对象
1438429206680
----马上 删除cat3对象
__del__方法被调用
波斯猫对象马上被干掉了...

删除cat后，cat2和cat3仍然指向同一地址，说明对象的引用还有2个
继续删除cat2后，cat3仍然引用对象地址，引用计数还剩1个
删除cat3后，cat，cat2，cat3的引用都被删除，调用del方法，所以打印了del方法被调用这句话，而上面还有引用的时候，就不会打印这句话，

89、python 中读取 Excel 文件的方法

应用数据分析库 pandas。



The screenshot displays a Python code editor on the left and a Microsoft Excel spreadsheet on the right. In the code editor, the following code is shown:

```
50 import pandas as pd
51 df=pd.read_excel("333.xlsx")
52 print(df)
53
```

A red arrow points from the text **read_excel方法** to the `read_excel` function in the code. Below the code, the output of the `print(df)` statement is shown as a table:

	姓名	年龄	城市
0	张三	11	北京
1	李四	12	深圳
2	王五	12	上海

The Excel spreadsheet on the right shows the same data in a grid format with columns labeled A, B, and C, and rows numbered 1, 2, and 3.

90、HTTP 请求中 get 和 post 区别

1、GET 请求是通过 URL 直接请求数据，数据信息可以在 URL 中直接看到，比如浏览器访问；而 POST 请求是放在请求头中的，我们是无法直接看到的；

2、GET 提交有数据大小的限制，一般是不超过 1024 个字节，而这种说法也不完全准确，HTTP 协议并没有设定 URL 字节长度的上限，而是浏览器做了些处理，所以长度依据浏览器的不同有所不同；POST 请求在 HTTP 协议中也没有做说明，一般来说是没有设置限制的，但是实际上浏览器也有默认值。总体来说，少量的数据使用 GET，大量的数据使用 POST。

3、GET 请求因为数据参数是暴露在 URL 中的，所以安全性比较低，比如密码是不能暴露的，就不能使用 GET 请求；POST 请求中，请求参数信息是放在请求头的，所以安全性较高，可以使用。在实际中，涉及到登录操作的时候，尽量使用 HTTPS 请求，安全性更好。