

- Python基础

- Python语言基本

- 1.列出 5 个常用 Python 标准库?
- 2.Python的内建数据类型有哪些?
- 3.简述 with 方法打开处理文件帮我们做了什么?
- 4.Python的可变和不可变数据类型?
- 5.Python 获取当前日期?
- 6.谈谈对 Python 的了解和其他语言的区别
- 7.说说你知道的Python3 和 Python2 之间的区别
- 8.了解 Python 之禅么?
- 9.了解 docstring 么?
- 10.了解类型注解么?
- 11.例举你知道 Python 对象的命名规范，例如方法或者类等
- 12.例举几个规范 Python 代码风格的工具
- 13.一个编码为 GBK 的字符串 S，要将其转成 UTF-8 编码的字符串，应如何操作?
- 14.用正则切分字符串去除非符号
- 15.单引号、双引号、三引号的区别?
- [16. `[[1,2],[3,4],[5,6]]`一行代码展开该列表，得出`[1,2,3,4,5,6]`(#16 `[[12][34][56]]`一行代码展开该列表得出`[123456]`)
- 17.哪些不能作为字典的键
- 18.如何交换字典 `{"A": 1,"B": 2}`的键和值?
- 19.对生成器类型的对象实现切片功能

- 20.关于list tuple copy 和 deepcopy 的区别是什么？
- 21.代码中经常遇到的*args, **kwargs 含义及用法。
- 22.Python 中会有函数或成员变量包含单下划线前缀和结尾，和双下划线前缀结尾，区别是什么？
- 23.json 序列化时，可以处理的数据类型有哪些？如何定制支持 datetime 类型？
- 24.json 序列化时，默认遇到中文会转换成 unicode，如果想要保留中文怎么办？
- 25.如果当前的日期为 20190530，要求写一个函数输出 N 天后的日期，(比如 N 为 2，则输出 20190601)
- 26.python 字典和 json 字符串相互转化方法
- 27.函数装饰器有什么作用？请列举说明？
- 28.call
- 29.如何判断一个对象是函数还是方法？
- 30.python实现接口？
- 31.Python 中的反射了解么？
- 32.metaclass or type
- 33.Python中递归的最大次数1000 ?怎么改
- 34.列举 5 个 Python 中的异常类型以及其含义
- 35.w、a+、wb 文件写入模式的区别
- 36.举例 sort 和 sorted 的区别
- 37.在 requests 模块中，requests.content 和 requests.text 什么区别
- 38.python新式类和经典类的区别
- 39.字符串的操作题目
- 40.可变类型和不可变类型

- 41.is和==有什么区别?
- 42.求出列表所有奇数并构造新列表
- 43.用一行python代码写出1+2+3+10248
- 44.Python中变量的作用域? (变量查找顺序)
- 45.字符串 "123" 转换成 123 , 不使用内置api, 例如 int()
- 46.Given an array of integers
- 47.有一个jsonline格式的文件file.txt大小约为10K
- 48.返回该文件夹中所有文件的路径
- 49.设计实现遍历目录与子目录, 抓取.pyc文件
- 50.输入日期, 判断这一天是这一年的第几天?
- 51.打乱一个排好序的list对象alist?
- 52.现有字典 d= {'a':24,'g':52,'i':12,'k':33}请按value值进行排序?
- 53.字典推导式
- 54.请反转字符串 "aStr"?
- 55.请按alist中元素的age由大到小排序
- 56.下面代码的输出结果将是什么?
- 57.写一个列表生成式, 产生一个公差为11的等差数列
- 58.给定两个列表, 怎么找出他们相同的元素和不同的元素?
- 59.统计一个文本中单词频次最高的10个单词?
- 60.给定一个任意长度数组, 实现一个函数
- 61.写一个函数找出一个整数数组中, 第二大的数
- 62.阅读一下代码他们的输出结果是什么?

- 63.统计一段字符串中字符出现的次数
- 64.super函数的具体用法和场景

- Python高级

- 元类

- 65.Python中类方法、类实例方法、静态方法有何区别？
 - 66.遍历一个object的所有属性，并print每一个属性名？
 - 67.写一个类，并让它尽可能多的支持操作符？
 - 68.介绍Cython, Pypy Cpython Numba各有什么缺点
 - 69.请描述抽象类和接口类的区别和联系
 - 70.Python中如何动态获取和设置对象的属性？

- 内存管理与垃圾回收机制

- 71.哪些操作会导致Python内存泄露，怎么处理？
 - 72.关于内存溢出和内存泄漏的区别
 - 73.Python的内存管理机制及调优手段？

- 函数

- 74.什么是Hash（散列函数）？
 - 75.编写函数的4个原则
 - 76.函数调用参数的传递方式是值传递还是引用传递？
 - 77.如何在function里面设置一个全局变量
 - 78.带参数的装饰器？
 - 79.递归函数停止的条件？

- 设计模式

- 80.对设计模式的理解，简述你了解的设计模式？
- 81.python如何实现单例模式
- 82.单例模式的应用场景有那些？
- 83.对装饰器的理解，并写出一个计时器记录方法执行性能的装饰器？
- 84.解释以下什么是闭包？
- 85.函数装饰器有什么作用？
- 86.生成器，迭代器的区别？
- 87.X是什么类型？
- 88.请用一行代码 实现将1-N 的整数列表以3为单位分组
- 89.Python中yield的用法？

- 面向对象

- 90.Python的魔法方法
- 91.谈谈你对面向对象的理解？

- 正则表达式

- 92.请写出一段代码用正则匹配出ip？
- 93.a = "abbbccc"，用正则匹配为abccc,不管有多少b，就出现一次？
- 94.Python字符串查找和替换？
- 95.用Python匹配HTML g tag的时候，<.> 和 <.*?> 有什么区别
- 96.正则表达式贪婪与非贪婪模式的区别？

- 系统编程

- 97.进程总结
- 98.谈谈你对多进程，多线程，以及协程的理解，项目是否用？
- 99.Python异步使用场景有那些？
- 100.多线程共同操作同一个数据互斥锁同步？
- 101.什么是多线程竞争？
- 102.请介绍一下Python的线程同步？
- 103.解释以下什么是锁，有哪几种锁？
- 104.什么是死锁？
- 105.多线程交互访问数据，如果访问到了就不访问了？
- 106.什么是线程安全，什么是互斥锁？
- 107.说说下面几个概念：同步，异步，阻塞，非阻塞？
- 108.什么是僵尸进程和孤儿进程？怎么避免僵尸进程？
- 109..python中进程与线程的使用场景？
- 110..线程是并发还是并行，进程是并发还是并行？
- 111.并行(parallel)和并发 (concurrency)?
- 112.IO密集型和CPU密集型区别？
- 113.python asyncio的原理？
- 网络编程
 - 114.怎么实现强行关闭客户端和服务端之间的连接？
 - 115.简述TCP和UDP的区别以及优缺点？
 - 116.简述浏览器通过WSGI请求动态资源的过程？
 - 117.描述用浏览器访问www.baidu.com的过程

- 118.Post和Get请求的区别?
- 119.列出你知道的HTTP协议的状态码，说出表示什么意思?
- 120.请简单说一下三次握手和四次挥手?
- 121.为什么客户端在TIME-WAIT状态必须等待2MSL的时间?
- 122.说说HTTP和HTTPS区别?
- 123.谈一下HTTP协议以及协议头部中表示数据类型的字段?
- 124.HTTP请求方法都有什么?
- 125.使用Socket套接字需要传入哪些参数?
- 126.HTTP常见请求头?
- 127.七层模型?
- 128.url的形式?

- Web

- Flask

- 129.对Flask蓝图(Blueprint)的理解?
- 130.Flask 和 Django 路由映射的区别?

- Django

- 131.ORM是什么，ORM的优缺点
- 132.查找 Django 项目中的性能瓶颈
- 133.什么是wsgi,uwsgi,uWSGI?
- 134.Django、Flask、Tornado的对比?
- 135.CORS 和 CSRF的区别?

- 136.Session, Cookie, JWT的理解
- 137.简述Django请求生命周期
- 138.用的restframework完成api发送时间时区
- 139.nginx, tomcat, apach到都是什么?
- 140.请给出你熟悉关系数据库范式有哪些，有什么作用?
- 141.简述QQ登陆过程
- 142.项目中日志的作用
- 143.django中间件的使用?
- 144.谈一下你对uWSGI和nginx的理解?
- 145.Python中三大框架各自的应用场景?
- 146.Django中哪里用到了线程? 哪里用到了协程? 哪里用到了进程?
- 147.有用过Django REST framework吗?
- 148.对cookies与session的了解? 他们能单独用吗?
- 149.测试工具



欢迎添加vx: **257735** 长按扫码发送「**666**」获取10本Python精美电子书

Python基础

Python语言基本

1.列出 5 个常用 Python 标准库?

os, logging, system, time, re, math,threading

2.Python的内建数据类型有哪些?

string, int, list, tuple, dict

3.简述 with 方法打开处理文件帮我们做了什么?

with 语句适用于对资源进行访问的场合，确保不管使用过程中是否发生异常都会执行必要的“清理”操作，释放资源，比如文件使用后自动关闭、线程中锁的自动获取和释放等。

with语句即“上下文管理器”，在程序中用来表示代码执行过程中所处的前后环境 上下文管理器：含有**enter**和**exit**方法的对象就是上下文管理器。 **enter()**：在执行语句之前，首先执行该方法，通常返回一个实例对象，如果with语句有as目标，则将对象赋值给as目标。

exit()：执行语句结束后，自动调用**exit()**方法，用户释放资源，若此方法返回布尔值True，程序会忽略异常。 使用环境：文件读写、线程锁的自动释放等。

4.Python的可变和不可变数据类型?

可变 list, dict, set 不可变 int string tuple

5.Python 获取当前日期?

```
from datetime import datetime; datetime.now()
```

6.谈谈对 Python 的了解和其他语言的区别

python是典型的动态类型强类型语言

强类型语言, 不需要隐士转换

解释性，解释型语言使用解释器将源码逐行解释成机器码并立即执行，不会进行整体性的编译和链接处理，相当于把编译语言中的编译和解释混合到一起同时完成。

简洁优雅，面向对象，跨平台，

Python是动态类型语言，而Java是静态类型语言。

7.说说你知道的Python3 和 Python2 之间的区别

print, string/unicode, exception, divide, xrange,

8.了解 Python 之禅么？

```
import this
```

```
The Zen of Python, by Tim Peters
Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

优美胜于丑陋（Python 以编写优美的代码为目标）

明了胜于晦涩（优美的代码应当是明了的，命名规范，风格相似）

简洁胜于复杂（优美的代码应当是简洁的，不要有复杂的内部实现）

复杂胜于凌乱（如果复杂不可避免，那代码间也不能有难懂的关系，要保持接口简洁）

扁平胜于嵌套（优美的代码应当是扁平的，不能有太多的嵌套）

间隔胜于紧凑（优美的代码有适当的间隔，不要奢望一行代码解决问题）

可读性很重要（优美的代码是可读的）
即便假借特例的实用性之名，也不可违背这些规则（这些规则至高无上）
不要包容所有错误，除非你确定需要这样做（精准地捕获异常，不写 `except:pass` 风格的代码）
当存在多种可能，不要尝试去猜测
而是尽量找一种，最好是唯一一种明显的解决方案（如果不确定，就用穷举法）
虽然这并不容易，因为你不是 Python 之父（这里的 Dutch 是指 Guido）
做也许好过不做，但不假思索就动手还不如不做（动手之前要细思量）
如果你无法向人描述你的方案，那肯定不是一个好方案；反之亦然（方案测评标准）
命名空间是一种绝妙的理念，我们应当多加利用（倡导与号召）

9.了解 docstring 么？

文档字符串是一个重要工具，用于解释文档程序，帮助你的程序文档更加简单易懂。我们可以在函数体的第一行使用一对三个单引号 或者一对三个双引号 来定义文档字符串。你可以使用 **doc** 调用函数中的文档字符串属性

10.了解类型注解么？

```
def list_to_str (param_list:list,connect_str: str = " ") -> str:
    pass
```

python3 中注解用来给参数，返回值，变量的类型加上注解，对代码没影响 Python 提供了一个工具方便我们测试类型注解的正确性

`pip install mypy mypy demo.py` 若无错误则无输出

11.例举你知道 Python 对象的命名规范，例如方法或者类等

变量命名：字母数字下划线，不能以数字开头

`_` 受保护的

`__` 私有的

init 内置变量

函数和方法（类中叫做方法，模块中称作函数）命名：

12.例举几个规范 Python 代码风格的工具

pylint, yapf, autopep8, flake8

13.一个编码为 GBK 的字符串 S，要将其转成 UTF-8 编码的字符串，应如何操作？

```
demo_str = "demo".encode("gbk") demo=demo_str.decode('gbk').encode('utf-8')
```

14.用正则切分字符串去除非符号

```
s="info: xiaoZhang 33 shandong" ['info', 'xiaoZhang', '33', 'shandong']  
re.compile(r'\W').split(s)
```

15.单引号、双引号、三引号的区别？

在不需要转义的时候，单引号和双引号无区别

```
"abc 'wewe'we"  
'abc "wewe"we'  
'abc \'wewe\'we'
```

16. [[1,2],[3,4],[5,6]]一行代码展开该列表，得出[1,2,3,4,5,6]

```
question_list = [[1,2],[3,4],[5,6]] [a for inside in question_list for a in inside]
```

17.哪些不能作为字典的键

字典中的键是不可变类型，可变类型list和dict不能作为字典键 一个对象能不能作为字典的key，就取决于其有没有hash方法

18.如何交换字典 {'A': 1,'B': 2}的键和值？

```
result_dic = {v: k for k, v in demo_dic.items()}
```

19.对生成器类型的对象实现切片功能

```
import itertools
itertools.islice(generator, 10, 20)
```

20.关于list tuple copy 和 deepcopy 的区别是什么?

tuple:

```
a = (1, 2, 3, [4, 5, 6, 7], 8)
a[3] = 3 //TypeError
a[3][3] = 9 // a (1, 2, 3, [4, 5, 6, 9], 8)
```

列表是可变数据类型，数据的值可以修改的 这里只是修改了元祖子对象的值，而不是修改了元祖的值 修改可变类型的值不会改变内存id，因此元祖的引用还是没有发生变化 可以这么理解，只要不修改元祖中值的**内存id**，那么就可以进行“修改元祖”操作扩展，面试官可能会问到：元祖是否可以被修改？ 答：元祖是不可变数据类型，因此不能修改元祖中的值，但是如果元组中有可变数据类型，那么可以修改可变数据类型中的值，修改可变数据类型的值并不会使其内存id发生变化，所以元祖中元素中的内存id也没有改变，因此就做到了“修改元祖”操作

list:

```
a = [1,2,[3,4]]
b = a
c = a[:]
d = a.copy()
e = copy.deepcopy(a)

id(a), id(b), id(c), id(d), id(e) // 只有a b 是同一个指向
>>> (4398429512, 4398429512, 4398398664, 4398429576, 4398429192)

a.append(5) // 只有b跟着改变
a, b, c,d,e
([1, 2, [3, 4], 5], [1, 2, [3, 4], 5], [1, 2, [3, 4]], [1, 2, [3, 4]], [1, 2, [3, 4]])

a[2][1] = 7 // 除了deepcopy, 其他都跟着变了
a, b, c,d,e
([1, 2, [3, 7], 5], [1, 2, [3, 7], 5], [1, 2, [3, 7]], [1, 2, [3, 7]], [1, 2, [3, 4]])
```

copy 仅拷贝对象本身，而不拷贝对象中引用的其它对象。deepcopy 除拷贝对象本身，而且拷贝对象中引用的其它对象。（子对象）

21.代码中经常遇到的*args, **kwargs 含义及用法。

args 是 arguments 的缩写，表示位置参数 kwargs 是 keyword arguments 的缩写，表示关键字参数

22.Python 中会有函数或成员变量包含单下划线前缀和结尾，和双下划线前缀结尾，区别是什么？

下划线开头的命名方式被常用于模块中，在一个模块中以单下划线开头的变量和方法会被默认划入模块内部范围。

当使用 from my_module import * 导入时，单下划线开头的变量和方法是不会被导入的。但使用 import my_module 导入的话，仍然可以用 my_module._var 这样的形式访问属性或方法。

双下划线开头和结尾的是一些 python 的“魔术”对象

class A中定义的属性cont，这样的变量获取时需要用A._Acont

23.json 序列化时，可以处理的数据类型有哪些？如何定制支持 datetime 类型？

json序列化时，可以处理列表、字典、字符、数值、布尔和None 定制datetime类型↓

24.json 序列化时，默认遇到中文会转换成 unicode，如果想要保留中文怎么办？

```
print(json.dumps(dict_demo, ensure_ascii=False))
```

25.如果当前的日期为 20190530，要求写一个函数输出 N 天后的日期，(比如 N 为 2，则输出 20190601)

```
from datetime import datetime, timedelta
now_date = "20190530"
now_date = datetime.strptime(now_date, "%Y%m%d").date()
offset = timedelta(days=2)
(now_date + offset).strftime("%Y%m%d")
```

26.python 字典和 json 字符串相互转化方法

```
#导包
import json

#json字符串转换成字典
json.loads(json_str)

#字典转换成json字符串
json.dumps(dict)
```

27.函数装饰器有什么作用？请列举说明？

1，引入日志 2，函数执行时间统计3，执行函数前预备处理4，执行函数后清理功能5，权限校验等场景6，缓存7，事务处理

28.call

可以调用的对象: 一个特殊的魔术方法可以让类的实例的行为表现的像函数一样


```

class Entity:
    '''调用实体来改变实体的位置。'''

    def __init__(self, size, x, y):
        self.x, self.y = x, y
        self.size = size

    def __call__(self, x, y):
        '''改变实体的位置'''
        self.x, self.y = x, y

e = Entity(1, 2, 3) // 创建实例
e(4, 5) //实例可以象函数那样执行，并传入x y值，修改对象的x y

```

https://www.jianshu.com/p/e1d95c4e1697?utm_source=oschina-app

29.如何判断一个对象是函数还是方法？

在类外声明def为函数

类中声明def：使用类调用为函数，使用实例化对象调用为方法

可以使用isinstance()判断

```

class Work(object):
    def show(self):
        print("执行show方法")

work = Work()
print(Work.show)
print(work.show)

结果：
<function Work.show at 0x000001CC55BC5268>
<bound method Work.show of <__main__.Work object at 0x000001CC55C2F240>>

from types import MethodType,FunctionType
print(isinstance(Work.show,FunctionType))
print(isinstance(work.show,MethodType))

结果：
True
True

```

30.python实现接口？

接口只是定义了一些方法，而没有去实现，多用于程序设计时，只是设计需要有什么样的功能，但是并没有实现任何功能，这些功能需要被另一个类（B）继承后，由类B去实现其中的某个功能或全部功能。

遵循：开放封闭原则，依赖导致原则，接口隔离原则，继承多态。

编程思想：为子类做规范；归一化设计：几个类都实现了相同的方法 抽象类：最好单继承，且可以简单的实现功能，接口类：可以多继承，且最好不实现具体功能

在python中接口由抽象类和抽象方法去实现，接口是不能被实例化的，只能被别的类继承去实现相应的功能。

个人觉得接口在python中并没有那么重要，因为如果要继承接口，需要把其中的每个方法全部实现，否则会报编译错误，还不如直接定义一个class，其中的方法实现全部为pass，让子类重写这些函数。

方法一：用抽象类和抽象函数实现方法（适用于单继承） 方法二：用普通类定义接口（推荐）

31.Python 中的反射了解么？

在Python中，能够通过一个对象，找出其type、class、attribute或method的能力，称为反射或自省 具有反射能力的函数有type(),isinstance(),callable().dir().getattr()等

32.metaclass or type

<https://www.liaoxuefeng.com/wiki/897692888725344/923030550637312>

33.Python中递归的最大次数1000 ?怎么改

```
import sys sys.setrecursionlimit(1500) # set the maximum depth as 1500
```

34.列举 5 个 Python 中的异常类型以及其含义

```
BaseException
+-- SystemExit
+-- KeyboardInterrupt
+-- GeneratorExit
+-- Exception
    +-- StopIteration
    +-- StopAsyncIteration
    +-- ArithmeticError
    |   +-- FloatingPointError
    |   +-- OverflowError
    |   +-- ZeroDivisionError
    +-- AssertionError
    +-- AttributeError
    +-- BufferError
    +-- EOFError
    +-- ImportError
    |   +-- ModuleNotFoundError
    +-- LookupError
    |   +-- IndexError
    |   +-- KeyError
    +-- MemoryError
    +-- NameError
    |   +-- UnboundLocalError
    +-- OSError
    |   +-- BlockingIOError
    |   +-- ChildProcessError
    |   +-- ConnectionError
    |   |   +-- BrokenPipeError
    |   |   +-- ConnectionAbortedError
    |   |   +-- ConnectionRefusedError
    |   |   +-- ConnectionResetError
    |   +-- FileExistsError
    |   +-- FileNotFoundError
    |   +-- InterruptedError
    |   +-- IsADirectoryError
    |   +-- NotADirectoryError
    |   +-- PermissionError
    |   +-- ProcessLookupError
    |   +-- TimeoutError
    +-- ReferenceError
    +-- RuntimeError
    |   +-- NotImplementedError
    |   +-- RecursionError
    +-- SyntaxError
```

```
|    +-- IndentationError
|        +-- TabError
+-- SystemError
+-- TypeError
+-- ValueError
|    +-- UnicodeError
|        +-- UnicodeDecodeError
|        +-- UnicodeEncodeError
|        +-- UnicodeTranslateError
+-- Warning
    +-- DeprecationWarning
    +-- PendingDeprecationWarning
    +-- RuntimeWarning
    +-- SyntaxWarning
    +-- UserWarning
    +-- FutureWarning
    +-- ImportWarning
    +-- UnicodeWarning
    +-- BytesWarning
    +-- ResourceWarning
```

35.w、a+、wb 文件写入模式的区别

r: 读取文件, 若文件不存在则会报错

w: 写入文件, 若文件不存在则会先创建再写入, 会覆盖原文件

a: 写入文件, 若文件不存在则会先创建再写入, 但不会覆盖原文件, 而是追加在文件末尾

rb,wb: 分别于r,w类似, 用于读写二进制文件

r+: 可读、可写, 文件不存在也会报错, 写操作时会覆盖

w+: 可读、可写, 文件不存在先创建, 会覆盖

a+: 可读、可写, 文件不存在先创建, 不会覆盖, 追加在末尾

36.举例 sort 和 sorted 的区别

使用sort()方法对list排序会修改list本身,不会返回新list, sort()不能对dict字典进行排序;

sorted方法对可迭代的序列排序生成新的序列, 对dict排序默认会按照dict的key值进行排序, 最后返回的结果是一个对key值排序好的list;

sorted对tuple, dict依然有效, 而sort不行

37.在 requests 模块中, requests.content 和 requests.text 什么区别

.content中间存的是字节码 .text存的是.content编码后的字符串

操作方式就是, 如果想取得文本就用.text, 如果想获取图片, 就用.content

38.python新式类和经典类的区别

这2篇文章很好的介绍了新式类的特性:

新式类多继承搜索顺序(广度优先): 先在水平方向查找, 然后再向上查找

经典类多继承搜索顺序(深度优先): 先深入继承树左侧查找, 然后再返回, 开始查找右侧

<http://stackoverflow.com/questions/54867/what-is-the-difference-between-old-style-and-new-style-classes-in-python> <http://www.cnblogs.com/btchenguang/archive/2012/09/17/2689146.html>

39.字符串的操作题目

全字母短句 PANGRAM 是包含所有英文字母的句子, 比如: A QUICK BROWN FOX JUMPS OVER THE LAZY DOG. 定义并实现一个方法 get_missing_letter, 传入一个字符串采纳数, 返回参数字符串变成一个 PANGRAM 中所缺失的字符。应该忽略传入字符串参数中的大小写, 返回应该都是小写字符并按字母顺序排序 (请忽略所有非 ACSII 字符)

下面示例是用来解释, 双引号不需要考虑:

(0)输入: "A quick brown for jumps over the lazy dog"

返回: ""

(1)输入: "A slow yellow fox crawls under the proactive dog"

返回: "bjkmaqz"

(2)输入: "Lions, and tigers, and bears, oh my!"

返回: "cfjqpquvwxyz"

(3)输入: ""

返回: "abcdefghijklmnopqrstuvwxy"

```
def get_missing_letter(a):  
    s1 = set("abcdefghijklmnopqrstuvwxy")  
    s2 = set(a)  
    ret = "".join(sorted(s1-s2))  
    return ret  
  
print(get_missing_letter("python"))
```

40.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。

41.is和==有什么区别?

a = 1024 b = 1024 a is b > False a == b > True is: 比较的是两个对象的id值是否相等, 也就是比较俩对象是否为同一个实例对象。是否指向同一个内存地址

== : 比较的两个对象的内容/值是否相等, 默认会调用对象的eq()方法



欢迎添加vx: [257735](#) 长按扫码发送「[666](#)」获取10本Python精美电子书

42. 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]
res = [ i for i in a if i%2==1]
print(res)
```

43.用一行python代码写出1+2+3+10248

```
from functools import reduce
#1.使用sum内置求和函数
num = sum([1,2,3,10248])
print(num)
#2.reduce 函数
num1 = reduce(lambda x,y :x+y,[1,2,3,10248])
print(num1)
```

44.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，也正是按照这个顺序来查找的

45.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)
```

46. Given an array of integers

给定一个整数数组和一个目标值, 找出数组中和为目标值的两个数。你可以假设每个输入只对应一种答案, 且同样的元素不能被重复利用。示例: 给定 `nums = [2,7,11,15]`, `target=9` 因为 `nums[0]+nums[1] = 2+7 =9`, 所以返回 `[0,1]`

```
def two_sum(nums, target):
    """
    :type nums: List[int]
    :type target: int
    :rtype: List[int]
    """
    for i in nums:
        if target - i in nums and i is not target - i:
            return [nums.index(i), nums.index(target - i)]

nums_all = [2, 7, 11, 15]
target = 9
nums = two_sum(nums_all, target)
print(nums)
```

47.有一个jsonline格式的文件file.txt大小约为10K

```
def get_lines():
    with open('file.txt','rb') as f:
        return f.readlines()

if __name__ == '__main__':
    for e in get_lines():
        process(e) # 处理每一行数据
```

现在要处理一个大小为10G的文件，但是内存只有4G，如果在只修改get_lines 函数而其他代码保持不变的情况下，应该如何实现？需要考虑的问题都有哪些？

```
def get_lines():
    with open('file.txt','rb') as f:
        for i in f:
            yield i
```

Pandaaaa906提供的方法

```
from mmap import mmap

def get_lines(fp):
    with open(fp,"r+") as f:
        m = mmap(f.fileno(), 0)
        tmp = 0
        for i, char in enumerate(m):
            if char==b"\n":
                yield m[tmp:i+1].decode()
                tmp = i+1

if __name__=="__main__":
    for i in get_lines("fp_some_huge_file"):
        print(i)
```

要考虑的问题有：内存只有4G无法一次性读入10G文件，需要分批读入分批读入数据要记录每次读入数据的位置。分批每次读取数据的大小，太小会在读取操作花费过多时间。 <https://stackoverflow.com/questions/30294146/python-fastest-way-to-process-large-file>

48.返回该文件夹中所有文件的路径

```
def print_directory_contents(sPath):  
    """  
    这个函数接收文件夹的名称作为输入参数  
    返回该文件夹中文件的路径  
    以及其包含文件夹中文件的路径  
    """  
    import os  
    for s_child in os.listdir(s_path):  
        s_child_path = os.path.join(s_path, s_child)  
        if os.path.isdir(s_child_path):  
            print_directory_contents(s_child_path)  
        else:  
            print(s_child_path)
```

49.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)

```

50.输入日期， 判断这一天是这一年的第几天？

```

import datetime
def dayofyear():
    year = input("请输入年份: ")
    month = input("请输入月份: ")
    day = input("请输入天: ")
    date1 = datetime.date(year=int(year),month=int(month),day=int(day))
    date2 = datetime.date(year=int(year),month=1,day=1)
    return (date1-date2).days+1

```

51.打乱一个排好序的list对象alist?

```
import random
alist = [1,2,3,4,5]
random.shuffle(alist)
print(alist)

# 下面这段代码是shuffle的实现
items = list(alist)
for i in xrange(len(alist)):
    alist[i] = items.pop(self.randrange(len(items)))
```

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

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

53.字典推导式

```
# 将字符串 "k:1 |k1:2|k2:3|k3:4", 处理成字典 {k:1,k1:2,...}
# d = {key:value for (key,value) in iterable}
d = {k:int(v) for t in str1.split("|") for k, v in (t.split(":"), )}
```

54.请反转字符串 "aStr"?

```
print("aStr"[::-1])
```

55.请按alist中元素的age由大到小排序

```
alist = [{'name':'a','age':20},{'name':'b','age':30},{'name':'c','age':25}]
def sort_by_age(list1):
    return sorted(alist, key=lambda x:x['age'], reverse=True)
```

56.下面代码的输出结果将是什么?

```
list = ['a','b','c','d','e']
print(list[10:])
```

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

57.写一个列表生成式,产生一个公差为11的等差数列

```
print([x*11 for x in range(10)])
```

58.给定两个列表,怎么找出他们相同的元素和不同的元素?

```
list1 = [1,2,3]
list2 = [3,4,5]
set1 = set(list1)
set2 = set(list2)
print(set1 & set2)
print(set1 ^ set2)
```

59.统计一个文本中单词频次最高的10个单词?

```
import re

def test(filepath):

    distone = {}
    numTen = []

    with open(filepath,"r",encoding="utf-8") 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
    numTen = sorted(distone.items(),key=lambda x:x[1],reverse=True)[:10]
    numTen =[x[0]for x in numTen]
    return numTen
```

60.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'

```
def func1(l):
    if isinstance(l,str):
        l = list(l)
        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))
```

61.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
    """
    #直接排序，输出倒数第二个数即可
    tmp_list = sorted(num_list)
    print ("Second_large_num is :",tmp_list[-2])
    #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]
        else:
            pass
    print("Second_large_num is :",two)
if __name__ == '__main__':
    num_list = [34,11,23,56,78,0,9,12,3,7,5]
    find_Second_large_num(num_list)
```

62.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()])
```

提示: 闭包，作用域 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] 这篇讲述的很详细了 <https://www.cnblogs.com/shiqi17/p/9608195.html>

63.统计一段字符串中字符出现的次数

```
def count_str(str_data):
    """定义一个字符出现次数的函数"""
    dict_str = {}
    for i in str_data:
        dict_str[i] = dict_str.get(i,0)+1
    return dict_str
dict_str = count_str("AAABBCCAC")
str_count_data = ""
for k,v in dict_str.items():
    str_count_data += k +str(v)
print(str_count_data)
```

64.super函数的具体用法和场景

https://python3-cookbook.readthedocs.io/zh_CN/latest/c08/p07_calling_method_on_parent_class.html

Python高级

元类

65. Python中类方法、类实例方法、静态方法有何区别？

类方法: 是类对象的方法，在定义时需要在上方使用 @classmethod 进行装饰,形参为cls，表示类对象，类对象和实例对象都可调用

类实例方法: 是类实例化对象的方法,只有实例对象可以调用，形参为self,指代对象本身;

静态方法: 是一个任意函数，在其上方使用 @staticmethod 进行装饰，可以用对象直接调用，静态方法实际上跟该类没有太大关系

66. 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
```

67. 写一个类，并让它尽可能多的支持操作符？

```
class Array:
    __list = []

    def __init__(self):
        print "constructor"

    def __del__(self):
        print "destruct"
```

```

def __str__(self):
    return "this self-defined array class"

def __getitem__(self, key):
    return self.__list[key]

def __len__(self):
    return len(self.__list)

def Add(self, value):
    self.__list.append(value)

def Remove(self, index):
    del self.__list[index]

def DisplayItems(self):
    print "show all items---"
    for item in self.__list:
        print item

```

68.介绍Cython, Pypy Cpython Numba各有什么缺点

CPython CPython is Guido van Rossum's reference version of the Python computing language. It's most often called simply "Python"; speakers say "CPython" generally to distinguish it explicitly from other implementations. CPython是使用最广的Python解释器。教程的所有代码也都在CPython下执行

IPython IPython是基于CPython之上的一个交互式解释器，也就是说，IPython只是在交互方式上有所增强

PyPy PyPy是另一个Python解释器，它的目标是执行速度。PyPy采用JIT技术，对Python代码进行动态编译（注意不是解释），所以可以显著提高Python代码的执行速度。绝大部分Python代码都可以在PyPy下运行，但是PyPy和CPython有一些是不同的，这就导致相同的Python代码在两种解释器下执行可能会有不同的结果。如果你的代码要放到PyPy下执行，就需要了解PyPy和CPython的不同点

Jython Jython是将Python code在JVM上面跑和调用java code的解释器。

69.请描述抽象类和接口类的区别和联系

70.Python中如何动态获取和设置对象的属性？

hasattr(), getattr(), setattr()

内存管理与垃圾回收机制

71.哪些操作会导致Python内存泄露，怎么处理？

在使用KafkaProducer 进行消息读写的时候， 错误的配置了buffer_memory参数的值， 导致每次进行 类调用， 都导致客户端不断的重复写内存， 并且因为最开始没有使用单例模式， 导致内存不断上升， 最后溢出。 解决办法就是修正错误的配置了buffer_memory参数的值 和使用单例

```
buffer_memory (int): The total bytes of memory the producer should use
to buffer records waiting to be sent to the server. If records are
sent faster than they can be delivered to the server the producer
will block up to max_block_ms, raising an exception on timeout.
In the current implementation, this setting is an approximation.
Default: 33554432 (32MB)
```

72.关于内存溢出和内存泄漏的区别

内存溢出：（Out Of Memory---OOM）

系统已经不能再分配出你所需要的空间，比如你需要100M的空间，系统只剩90M了，这就叫内存溢出

内存泄漏：（Memory Leak）

强引用所指向的对象不会被回收，可能导致内存泄漏，虚拟机宁愿抛出OOM也不会去回收他指向的对象

73.Python的内存管理机制及调优手段？

gc模块 <https://docs.python.org/3.7/library/gc.html>

```
gc.disable() # 暂停自动垃圾回收。
gc.collect() # 执行一次完整的垃圾回收，返回垃圾回收所找到无法到达的对象的数量。
gc.set_threshold() # 设置Python垃圾回收的阈值。
gc.set_debug() # 设置垃圾回收的调试标记。调试信息会被写入std.err。
```

Python有两种共存的内存管理机制: 引用计数和垃圾回收

垃圾回收机制:

1. 引用计数 PyObject

python里每一个东西都是对象，它们的核心就是一个结构体：PyObject

PyObject是每个对象必有的内容，其中ob_refcnt就是做为引用计数。当一个对象有新的引用时，它的ob_refcnt就会增加，当引用它的对象被删除，它的ob_refcnt就会减少

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

标记-清除机制，顾名思义，首先标记对象（垃圾检测），然后清除垃圾（垃圾回收）。

首先初始所有对象标记为白色，并确定根节点对象（这些对象是不会被删除），标记它们为黑色（表示对象有效）。

将有效对象引用的对象标记为灰色（表示对象可达，但它们所引用的对象还没检查），检查完灰色对象引用的对象后，将灰色标记为黑色。

重复直到不存在灰色节点为止。最后白色结点都是需要清除的对象。

3. 分代回收

从前面“标记-清除”这样的垃圾收集机制来看，这种垃圾收集机制所带来的额外操作实际上与系统中总的内存块的数量是相关的，当需要回收的内存块越多时，垃圾检测带来的额外操作就越多，而垃圾回收带来的额外操作就越少；反之，当需回收的内存块越少时，垃圾检测就将比垃圾回收带来更少的额外操作

- 1、新创建的对象做为0代
- 2、每执行一个【标记-删除】，存活的对象代数就+1
- 3、代数越高的对象（存活越持久的对象），进行【标记-删除】的时间间隔就越长。这个间隔，江湖人称阈值。

4. 三种情况触发垃圾回收

- 1、调用gc.collect()
- 2、GC达到阈值时
- 3、程序退出时

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

函数

74.什么是Hash（散列函数）？

75.编写函数的4个原则

- 1、函数设计要尽量短小，嵌套层次不宜过深。避免过长函数，嵌套最好能控制在3层之内
- 2、函数申明应该合理，简单，易于使用。除函数名能够正确反映其大体功能外，参数的设计也应该简洁明了，参数个数不宜太多
- 3、函数参数设计应该考虑向下兼容。可以通过加入默认参数来避免退化
- 4、一个函数只做一件事，就要尽量保证抽象层级的一致性，所有语句尽量在一个粒度上。若在一个函数中处理多件事，不利于代码的重用

76.函数调用参数的传递方式是值传递还是引用传递？

python不允许程序员选择采用传值还是传引用。Python参数传递采用的肯定是“传对象引用”的方式。这种方式相当于传值和传引用的一种综合。如果函数收到的是一个可变对象（比如字典或者列表）的引用，就能修改对象的原始值——相当于通过“传引用”来传递对象。如果函数收到的是一个不可变对象（比如数字、字符或者元组）的引用，就不能直接修改原始对象——相当于通过“传值”来传递对象。

77.如何在function里面设置一个全局变量

global x

78.带参数的装饰器？

```
def decorator(*dargs, **dkwargs):  
  
    def params_wrapper(func):  
        @wraps(func)  
        def wrapper(*args, **kwargs):  
            func(*args, **kwargs)  
        return wrapper  
    return params_wrapper
```

类装饰器

```
class decorator(object):  
    def __init__(self, func):  
        self.func = func  
  
    def __call__(self, *args, **kwargs):  
        print('before.....')  
        res = self.func(*args, **kwargs)  
        print('after.....')  
        return res
```

79.递归函数停止的条件？

递归的终止条件一般定义在递归函数内部，在递归调用前要做一个条件判断，根据判断的结果选择是继续调用自身，还是return；返回终止递归。

终止的条件：

- 1、判断递归的次数是否达到某一限定值
- 2、判断运算的结果是否达到某个范围等，根据设计的目的来选择

设计模式

80.对设计模式的理解，简述你了解的设计模式？

设计模式是经过总结，优化的，对我们经常会碰到的一些编程问题的可重用解决方案。一个设计模式并不像一个类或一个库那样能够直接作用于我们的代码，反之，设计模式更为高级，它是一种必须在特定情形下实现的一种方法模板。常见的是工厂模式和单例模式

81.python如何实现单例模式

第一种方法:使用装饰器

```
def singleton(cls):
    instances = {}
    def wrapper(*args, **kwargs):
        if cls not in instances:
            instances[cls] = cls(*args, **kwargs)
        return instances[cls]
    return wrapper

@singleton
class Foo(object):
    pass

foo1 = Foo()
foo2 = Foo()
print foo1 is foo2 #True
```

第二种方法：使用基类 New 是真正创建实例对象的方法，所以重写基类的new 方法，以此保证创建对象的时候只生成一个实例

```

class Singleton(object):
    def __new__(cls,*args,**kwargs):
        if not hasattr(cls,'_instance'):
            cls._instance = super(Singleton,cls).__new__(cls,*args,**kwargs)
        return cls._instance

class Foo(Singleton):
    pass

foo1 = Foo()
foo2 = Foo()

print foo1 is foo2 #True

```

82.单例模式的应用场景有那些？

单例模式应用的场景一般发现在以下条件下： 资源共享的情况下，避免由于资源操作时导致的性能或损耗等，如日志文件，应用配置。控制资源的情况下，方便资源之间的互相通信。如线程池等，

- 1.网站的计数器
- 2.应用配置
- 3.多线程池
- 4.数据库配置 数据库连接池
- 5.应用程序的日志应用...

83.对装饰器的理解，并写出一个计时器记录方法执行性能的装饰器？

装饰器本质上是一个callable object，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。

```

import time
from functools import wraps

def timeit(func):
    @wraps(func)
    def wrapper(*args, **kwargs):

```



```

        start = time.clock()
        ret = func(*args, **kwargs)
        end = time.clock()
        print('used:', end-start)
        return ret

    return wrapper

@timeit
def foo():
    print('in foo()'foo())

```

84.解释以下什么是闭包?

在函数内部再定义一个函数，并且这个函数用到了外边函数的变量，那么将这个函数以及用到的一些变量称之为闭包。

```

import functools
def decorator(func):
    @functools.wraps(func)
    def wrapper(*args, **kwargs):
        print "start"
        return func(*args, **kwargs)
    return wrapper

```

85.函数装饰器有什么作用?

装饰器本质上是一个callable object，它可以在让其他函数在不需要做任何代码的变动的前提下增加额外的功能。装饰器的返回值也是一个函数的对象，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。权限的校验等场景，有了装饰器就可以抽离出大量的与函数功能本身无关的雷同代码并发并继续使用。详细参考：<https://manjusaka.itscoder.com/2018/02/23/something-about-decorator/>

86.生成器，迭代器的区别?

迭代器是遵循迭代协议的对象。用户可以使用 `iter()` 以从任何序列得到迭代器（如 `list`, `tuple`, `dictionary`, `set` 等）。另一个方法则是创建一个另一种形式的迭代器——`generator`。要获取下一个元素，则使用成员函数 `next()`（Python 2）或函数 `next()` function（Python 3）。当没有元素时，则引发 `StopIteration` 此例外。若要实现自己的迭代器，则只要实现 `next()`（Python 2）或 `__next__()`（Python 3）

生成器（Generator），只是在需要返回数据的时候使用`yield`语句。每次`next()`被调用时，生成器会返回它脱离的位置（它记忆语句最后一次执行的位置和所有的数据值）

区别：生成器能做到迭代器能做的所有事，而且因为自动创建`iter()`和`next()`方法，生成器显得特别简洁，而且生成器也是高效的，使用生成器表达式取代列表解析可以同时节省内存。除了创建和保存程序状态的自动方法，当发生器终结时，还会自动抛出`StopIteration`异常。

官方介绍：<https://docs.python.org/3/tutorial/classes.html#iterators>

87.X是什么类型?

```
X= (i for i in range(10))  
X是 generator类型
```

88.请用一行代码 实现将1-N 的整数列表以3为单位分组

```
N =100  
print ([[x for x in range(1,100)] [i:i+3] for i in range(0,100,3)])
```

89.Python中yield的用法?

`yield`就是保存当前程序执行状态。你用`for`循环的时候，每次取一个元素的时候就会计算一次。用`yield`的函数叫`generator`，和`iterator`一样，它的好处是不用一次计算所有元素，而是用一次算一次，可以节省很多空间，`generator`每次计算需要上一次计算结果，所以用`yield`，否则一`return`，上次计算结果就没了

90.Python的魔法方法

<https://segmentfault.com/a/1190000007256392>

91.谈谈你对面向对象的理解?

在我理解,面向对象是向现实世界模型的自然延伸,这是一种“万物皆对象”的编程思想。在现实生活中的任何物体都可以归为一类事物,而每一个个体都是一类事物的实例。面向对象的编程是以对象为中心,以消息为驱动,所以程序=对象+消息。

面向对象有三大特性,封装、继承和多态。

封装就是将一类事物的属性和行为抽象成一个类,使其属性私有化,行为公开化,提高了数据的隐秘性的同时,使代码模块化。这样做使得代码的复用性更高。

继承则是进一步将一类事物共有的属性和行为抽象成一个父类,而每一个子类是一个特殊的父类--有父类的行为和属性,也有自己特有的行为和属性。这样做扩展了已存在的代码块,进一步提高了代码的复用性。

如果说封装和继承是为了使代码重用,那么多态则是为了实现接口重用。多态的一大作用就是为了解耦--为了解除父子类继承的耦合度。如果说继承中父子类的关系是IS-A的关系,那么接口和实现类之间的关系是HAS-A。简单来说,多态就是允许父类引用(或接口)指向子类(或实现类)对象。很多的设计模式都是基于面向对象的多态性设计的。

总结一下,如果说封装和继承是面向对象的基础,那么多态则是面向对象最精髓的理论。掌握多态必先了解接口,只有充分理解接口才能更好的应用多态

正则表达式

92.请写出一段代码用正则匹配出ip?

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

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

94. Python字符串查找和替换？

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)` 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flags，默认为0

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

96.正则表达式贪婪与非贪婪模式的区别？

如：String str="abcaxc"; Patter p="ab*c";

贪婪匹配:正则表达式一般趋向于最大长度匹配，也就是所谓的贪婪匹配。如上面使用模式p匹配字符串str，结果就是匹配到：abcaxc(abc)。正则引擎默认是贪婪的，当出现""时,它会尽量去匹配尽可能长的字符串

非贪婪匹配：就是匹配到结果就好，就少的匹配字符。如上面使用模式p匹配字符串str，结果就是匹配到：abc(ab*c)。

系统编程

97.进程总结

进程：程序运行在操作系统上的一个实例，就称之为进程。进程需要相应的系统资源：内存、时间片、pid。创建进程：首先要导入multiprocessing中的Process：创建一个Process对象; 创建Process对象时，可以传递参数;

```
p = Process(target=XXX,args=(tuple,),kwargs={key:value})
target = XXX 指定的任务函数，不用加(),
args=(tuple,)kwargs={key:value}给任务函数传递的参数
```

使用start()启动进程 结束进程 给子进程指定函数传递参数Demo

```
import os
from multiprocessing import Process
import time

def pro_func(name,age,**kwargs):
    for i in range(5):
        print("子进程正在运行中, name=%s,age=%d,pid=%d"%(name,age,os.getpid()))
        print(kwargs)
        time.sleep(0.2)
if __name__ == "__main__":
    #创建Process对象
    p = Process(target=pro_func,args=('小明',18),kwargs={'m':20})
    #启动进程
    p.start()
    time.sleep(1)
    #1秒钟之后，立刻结束子进程
    p.terminate()
    p.join()
```

注意：进程间不共享全局变量

进程之间的通信-Queue

在初始化Queue()对象时（例如q=Queue()），若在括号中没有指定最大可接受的消息数量，获数量为负值时，那么就代表可接受的消息数量没有上限一直到内存尽头）

Queue.qsize():返回当前队列包含的消息数量

Queue.empty():如果队列为空，返回True，反之False

Queue.full():如果队列满了，返回True,反之False

Queue.get([block[,timeout]]):获取队列中的一条消息，然后将其从队列中移除，

block默认值为True。

如果block使用默认值，且没有设置timeout（单位秒），消息队列如果为空，此时程序将被阻塞（停在读中状态），直到消息队列读到消息为止，如果设置了timeout，则会等待timeout秒，若还没读取到任何消息，则抛出“Queue.Empty”异常：

Queue.get_nowait()相当于Queue.get(False)

Queue.put(item,[block[,timeout]]):将item消息写入队列，block默认值为True; 如果block使用默认值，且没有设置timeout（单位秒），消息队列如果已经没有空间可写入，此时程序将被阻塞（停在写入状态），直到从消息队列腾出空间为止，如果设置了timeout，则会等待timeout秒，若还没空间，则抛出“Queue.Full”异常 如果block值为False，消息队列如果没有空间可写入，则会立刻抛出“Queue.Full”异常; Queue.put_nowait(item):相当Queue.put(item,False)

进程间通信Demo:

```
from multiprocessing import Process, Queue
import os, time, random
#写数据进程执行的代码:
def write(q):
    for value in ['A', 'B', 'C']:
        print("Put %s to queue..." % value)
        q.put(value)
        time.sleep(random.random())
#读数据进程执行的代码
def read(q):
    while True:
        if not q.empty():
            value = q.get(True)
            print("Get %s from queue." % value)
            time.sleep(random.random())
        else:
            break
if __name__ == '__main__':
    #父进程创建Queue，并传给各个子进程
    q = Queue()
    pw = Process(target=write, args=(q,))
    pr = Process(target=read, args=(q,))
    #启动子进程pw，写入:
    pw.start()
    #等待pw结束
    pw.join()
    #启动子进程pr，读取:
    pr.start()
    pr.join()
```

```
#pr 进程里是死循环，无法等待其结束，只能强行终止：
print('')
print('所有数据都写入并且读完')
```

进程池Pool

```
#coding:utf-8
from multiprocessing import Pool
import os,time,random

def worker(msg):
    t_start = time.time()
    print("%s 开始执行，进程号为%d"%(msg,os.getpid()))
    # random.random()随机生成0-1之间的浮点数
    time.sleep(random.random()*2)
    t_stop = time.time()
    print(msg,"执行完毕，耗时%.2f"%(t_stop-t_start))

po = Pool(3)#定义一个进程池，最大进程数3
for i in range(0,10):
    po.apply_async(worker,(i,))
print("----start----")
po.close()
po.join()
print("-----end-----")
```

进程池中使用Queue

如果要使用Pool创建进程，就需要使用multiprocessing.Manager()中的Queue(),而不是multiprocessing.Queue(),否则会得到如下的错误信息：

RuntimeError: Queue objects should only be shared between processs through inheritance

```
from multiprocessing import Manager,Pool
import os,time,random
def reader(q):
    print("reader 启动(%s),父进程为 (%s)"%(os.getpid(),os.getpid()))
    for i in range(q.qsize()):
        print("reader 从Queue获取到消息:%s"%q.get(True))

def writer(q):
    print("writer 启动 (%s),父进程为(%s)"%(os.getpid(),os.getpid()))
    for i in range(10):
        q.put(i)

if __name__ == "__main__":
```

```
print("(%s)start"%os.getpid())
q = Manager().Queue()#使用Manager中的Queue
po = Pool()
po.apply_async(wrtier,(q,))
time.sleep(1)
po.apply_async(reader,(q,))
po.close()
po.join()
print("(%s)End"%os.getpid())
```

98.谈谈你对多进程，多线程，以及协程的理解，项目是否用？

这个问题被问的概念相当之大， 进程：一个运行的程序（代码）就是一个进程，没有运行的代码叫程序，进程是系统资源分配的最小单位，进程拥有自己独立的内存空间，所有进程间数据不共享，开销大。

线程: cpu调度执行的最小单位，也叫执行路径，不能独立存在，依赖进程存在，一个进程至少有一个线程，叫主线程，而多个线程共享内存（数据共享，共享全局变量),从而极大地提高了程序的运行效率。

协程: 是一种用户态的轻量级线程，协程的调度完全由用户控制。协程拥有自己的寄存器上下文和栈。协程调度时，将寄存器上下文和栈保存到其他地方，在切回来的时候，恢复先前保存的寄存器上下文和栈，直接操中栈则基本没有内核切换的开销，可以不加锁的访问全局变量，所以上下文的切换非常快。

<https://www.jb51.net/article/142080.htm>



欢迎添加vx: [257735](#) 长按扫码发送「[666](#)」获取10本Python精美电子书

99.Python异步使用场景有那些？

异步的使用场景:

- 1、 不涉及共享资源，获对共享资源只读，即非互斥操作
- 2、 没有时序上的严格关系

- 3、不需要原子操作，或可以通过其他方式控制原子性
- 4、常用于IO操作等耗时操作，因为比较影响客户体验和使用性能
- 5、不影响主线程逻辑

100.多线程共同操作同一个数据互斥锁同步？

```
import threading
import time
class MyThread(threading.Thread):
    def run(self):
        global num
        time.sleep(1)

        if mutex.acquire(1):
            num +=1
            msg = self.name + 'set num to ' +str(num)
            print msg
            mutex.release()

num = 0
mutex = threading.Lock()
def test():
    for i in range(5):
        t = MyThread()
        t.start()
if __name__=="__main__":
    test()
```

101.什么是多线程竞争？

线程是非独立的，同一个进程里线程是数据共享的，当各个线程访问数据资源时会出现竞争状态即：数据几乎同步会被多个线程占用，造成数据混乱，即所谓的线程不安全

那么怎么解决多线程竞争问题？---锁

锁的好处： 确保了某段关键代码（共享数据资源）只能由一个线程从头到尾完整地执行能解决多线程资源竞争下的原子操作问题。

锁的坏处： 阻止了多线程并发执行，包含锁的某段代码实际上只能以单线程模式执行，效率就大大地下降了

锁的致命问题: 死锁

102.请介绍一下Python的线程同步?

一、 setDaemon(False) 当一个进程启动之后，会默认产生一个主线程，因为线程是程序执行的最小单位，当设置多线程时，主线程会创建多个子线程，在Python中，默认情况下就是 setDaemon(False),主线程执行完自己的任务以后，就退出了，此时子线程会继续执行自己的任务，直到自己的任务结束。

例子

```
import threading
import time

def thread():
    time.sleep(2)
    print('---子线程结束---')

def main():
    t1 = threading.Thread(target=thread)
    t1.start()
    print('---主线程--结束')

if __name__ == '__main__':
    main()
#执行结果
---主线程--结束
---子线程结束---
```

二、 setDaemon (True) 当我们使用setDaemon(True)时，这是子线程为守护线程，主线程一旦执行结束，则全部子线程被强制终止

例子

```
import threading
import time
def thread():
    time.sleep(2)
    print('---子线程结束---')
def main():
    t1 = threading.Thread(target=thread)
    t1.setDaemon(True)#设置子线程守护主线程
```

```

t1.start()
print('---主线程结束---')

if __name__ == '__main__':
    main()
#执行结果
---主线程结束--- #只有主线程结束，子线程来不及执行就被强制结束

```

三、join（线程同步）join 所完成的工作就是线程同步，即主线程任务结束以后，进入堵塞状态，一直等待所有的子线程结束以后，主线程再终止。

当设置守护线程时，含义是主线程对于子线程等待timeout的时间将会杀死该子线程，最后退出程序，所以说，如果有10个子线程，全部的等待时间就是每个timeout的累加和，简单的来说，就是给每个子线程一个timeout的时间，让他去执行，时间一到，不管任务有没有完成，直接杀死。

没有设置守护线程时，主线程将会等待timeout的累加和这样的一段时间，时间一到，主线程结束，但是并没有杀死子线程，子线程依然可以继续执行，直到子线程全部结束，程序退出。

例子

```

import threading
import time

def thread():
    time.sleep(2)
    print('---子线程结束---')

def main():
    t1 = threading.Thread(target=thread)
    t1.setDaemon(True)
    t1.start()
    t1.join(timeout=1) #1 线程同步，主线程堵塞1s 然后主线程结束，子线程继续执行
                    #2 如果不设置timeout参数就等子线程结束主线程再结束
                    #3 如果设置了setDaemon=True和timeout=1主线程等待1s后会强制杀死子线程
    #然后主线程结束
    print('---主线程结束---')

if __name__ == '__main__':
    main()

```

103.解释以下什么是锁，有哪几种锁？

锁(Lock)是python提供的对线程控制的对象。有互斥锁，可重入锁，死锁。

104.什么是死锁？

若干子线程在系统资源竞争时，都在等待对方对某部分资源解除占用状态，结果是谁也不愿先解锁，互相干等着，程序无法执行下去，这就是死锁。

GIL锁 全局解释器锁（只在cpython里才有）

作用： 限制多线程同时执行，保证同一时间只有一个线程执行，所以cpython里的多线程其实是伪多线程！

所以python里常常使用协程技术来代替多线程，协程是一种更轻量级的线程。

进程和线程的切换时由系统决定，而协程由我们程序员自己决定，而模块gevent下切换是遇到了耗时操作时才会切换

三者的关系：进程里有线程，线程里有协程。

105.多线程交互访问数据，如果访问到了就不访问了？

怎么避免重读？

创建一个已访问数据列表，用于存储已经访问过的数据，并加上互斥锁，在多线程访问数据的时候先查看数据是否在已访问的列表中，若已存在就直接跳过。

106.什么是线程安全，什么是互斥锁？

每个对象都对应于一个可称为‘互斥锁’的标记，这个标记用来保证在任一时刻，只能有一个线程访问该对象。

同一进程中的多线程之间是共享系统资源的，多个线程同时对一个对象进行操作，一个线程操作尚未结束，另一线程已经对其进行操作，导致最终结果出现错误，此时需要对被操作对象添加互斥锁，保证每个线程对该对象的操作都得到正确的结果。

107.说说下面几个概念：同步，异步，阻塞，非阻塞？

同步：多个任务之间有先后顺序执行，一个执行完下个才能执行。

异步：多个任务之间没有先后顺序，可以同时执行，有时候一个任务可能要在必要的时候获取另一个同时执行的任务的结果，这个就叫回调！

阻塞：如果卡住了调用者，调用者不能继续往下执行，就是说调用者阻塞了。

非阻塞：如果不会卡住，可以继续执行，就是说非阻塞的。

同步异步相对于多任务而言，阻塞非阻塞相对于代码执行而言。

108.什么是僵尸进程和孤儿进程？怎么避免僵尸进程？

孤儿进程：父进程退出，子进程还在运行的这些子进程都是孤儿进程，孤儿进程将被init 进程（进程号为1）所收养，并由init 进程对他们完成状态收集工作。

僵尸进程：进程使用fork 创建子进程，如果子进程退出，而父进程并没有调用wait 获waitpid 获取子进程的状态信息，那么子进程的进程描述符仍然保存在系统中的这些进程是僵尸进程。

避免僵尸进程的方法：

- 1.fork 两次用孙子进程去完成子进程的任务
- 2.用wait()函数使父进程阻塞
- 3.使用信号量，在signal handler 中调用waitpid,这样父进程不用阻塞

109..python中进程与线程的使用场景？

多进程适合在CPU密集操作（cpu操作指令比较多，如位多的的浮点运算）。

多线程适合在IO密性型操作（读写数据操作比多的的，比如爬虫）

110..线程是并发还是并行，进程是并发还是并行？

线程是并发，进程是并行；

进程之间互相独立，是系统分配资源的最小单位，同一个线程中的所有线程共享资源。

111.并行(parallel)和并发 (concurrency)?

并行： 同一时刻多个任务同时在运行

不会在同一时刻同时运行，存在交替执行的情况。

实现并行的库有： multiprocessing

实现并发的库有: threading

程序需要执行较多的读写、请求和回复任务的需要大量的IO操作，IO密集型操作使用并发更好。

CPU运算量大的程序，使用并行会更好

concurrent.futures 中ThreadPoolExecutor ProcessPoolExecutor 封装multiprocessing, threading

112.IO密集型和CPU密集型区别？

IO密集型： 系统运行，大部分的状况是CPU在等 I/O（硬盘/内存）的读/写

CPU密集型： 大部分时间用来做计算，逻辑判断等CPU动作的程序称之CPU密集型。

113.python asyncio的原理？

asyncio这个库就是使用python的yield这个可以打断保存当前函数的上下文的机制，封装好了selector 摆脱掉了复杂的回调关系

114.怎么实现强行关闭客户端和服务端之间的连接?

115.简述TCP和UDP的区别以及优缺点?

UDP是面向无连接的通讯协议，UDP数据包括目的端口号和源端口号信息。

优点：UDP速度快、操作简单、要求系统资源较少，由于通讯不需要连接，可以实现广播发送

缺点：UDP传送数据前并不与对方建立连接，对接收到的数据也不发送确认信号，发送端不知道数据是否会正确接收，也不重复发送，不可靠。

TCP是面向连接的通讯协议，通过三次握手建立连接，通讯完成时四次挥手

优点：TCP在数据传递时，有确认、窗口、重传、阻塞等控制机制，能保证数据正确性，较为可靠。

缺点：TCP相对于UDP速度慢一点，要求系统资源较多

116.简述浏览器通过WSGI请求动态资源的过程?

- 1、浏览器发送请求给web服务器；
- 2、web服务器接收到动态请求后通过wsgi协议调用框架；
- 3、框架根据请求信息向数据库获取动态数据；
- 4、框架将获取的动态数据插入模板文件，构成响应体；
- 5、框架将响应体数据、响应状态码和说明、响应头信息返回给web服务器；
- 6、web服务器接收到框架提供的数据后将数据按照响应报文的格式编码发送给浏览器；
- 7、浏览器接收到相应数据后，通过解码并按照http协议格式显示在界面上。

117.描述用浏览器访问www.baidu.com的过程

浏览器访问百度

1、先要解析出baidu.com对应的地址

- 1.1 先通过arp获取默认网关（交换机）的mac地址（mac地址指的是物理地址）（UDP广播）
- 1.2 组织数据发送给默认网关(ip是dns服务器的ip, mac是默认网关的mac地址)
- 1.3 默认网关（交换机）拥有转发数据的能力，把数据转发给路由器
- 1.4 路由器根据自己的路由协议，选择一个合适的较快的路径转发数据给目的网关
- 1.5 目的网关(dns服务器所在的网关)把数据转发给dns服务器
- 1.6 dns服务器查询解析出baidu.com对应的IP地址，并将它原路返回给请求这个域名的client

2、得到了baidu.com对应的ip地址后，会发送tcp三次握手进行连接

3、使用http协议发送请求数据给web服务器

4、web服务器收到数据请求之后，通过查询自己的服务器得到相应的结果，原路返回给浏览器

5、浏览器接收到数据后，通过浏览器自己的渲染功能来显示这个网页

6、浏览器关闭连接，即四次挥手

118.Post和Get请求的区别？

1.GET是从服务器上获取数据，POST是向服务器传送数据

2.在客户端，GET方式在通过URL提交数据，数据在URL中可以看到，POST方式，数据放置在HTML——HEADER内提交

3.对于GET方式，服务器端用Request.QueryString获取变量的值，对于POST方式，服务器端用Request.Form获取提交的数据

MORE ?

浏览器的GET和POST

接口中的GET和POST

关于安全性

关于编码

浏览器的POST需要发两个请求吗?

关于URL的长度

119.列出你知道的HTTP协议的状态码，说出表示什么意思?

120.请简单说一下三次握手和四次挥手?

https://blog.csdn.net/qq_38950316/article/details/81087809

121.为什么客户端在TIME-WAIT状态必须等待2MSL的时间?

1) 为了保证客户端发送的最后一个ACK报文段能够达到服务器。这个ACK报文段可能丢失，因而使处在LAST-ACK状态的服务器收不到确认。服务器会超时重传FIN+ACK报文段，客户端就能在2MSL时间内收到这个重传的FIN+ACK报文段，接着客户端重传一次确认，重启计时器。最好，客户端和服务器都正常进入到CLOSED状态。如果客户端在TIME-WAIT状态不等待一段时间，而是再发送完ACK报文后立即释放连接，那么就无法收到服务器重传的FIN+ACK报文段，因而也不会再发送一次确认报文。这样，服务器就无法按照正常步骤进入CLOSED状态。2) 防止已失效的连接请求报文段出现在本连接中。客户端在发送完最后一个ACK确认报文段后，再经过时间2MSL，就可以使本连接持续的时间内所产生的所有报文段都从网络中消失。这样就可以使下一个新的连接中不会出现这种旧的连接请求报文段。

122.说说HTTP和HTTPS区别?

<https://www.cnblogs.com/jesse131/p/9080925.html>

123.谈一下HTTP协议以及协议头部中表示数据类型的字段？

124.HTTP请求方法都有什么？

125.使用Socket套接字需要传入哪些参数？

126.HTTP常见请求头？

HTTP消息头是在，客户端请求（Request）或服务器响应（Response）时传递的，位请求或响应的第一行，HTTP消息体（请求或响应的内容）是其后传输。HTTP消息头，以明文的字符串格式传送，是以冒号分隔的键/值对，如：Accept-Charset: utf-8，每一个消息头最后以回车符(CR)和换行符(LF)结尾。HTTP消息头结束后，会用一个空白的字段来标识，这样就会出现两个连续的CR-LF。

HTTP消息头由IANA（The Internet Assigned Numbers Authority，互联网数字分配机构）来整理和维护。其标准最早来源于RFC 4229。IANA将其整理到了消息头文档，文档中还包括了一些新提出的信息头。

HTTP消息头支持自定义，自定义的专用消息头一般会添加'X-'前缀。

常用的HTTP请求头：

```
Accept  可接受的响应内容类型（Content-Types）。 Accept: text/plain
Authorization  用于表示HTTP协议中需要认证资源的认证信息 Authorization: Basic
OSdjJGRpbjpvGVuIANlc2SdDE==
Cache-Control  用来指定当前的请求/回复中的，是否使用缓存机制。 Cache-Control: no-cache
Cookie  由之前服务器通过Set-Cookie（见下文）设置的一个HTTP协议Cookie Cookie: $Version=1;
Skin=new;
Referer  表示浏览器所访问的前一个页面，可以认为是之前访问页面的链接将浏览器带到了当前页面。
Referer其实是Referrer这个单词，但RFC制作标准时给拼错了，后来也就将错就错使用Referer了。
Referer: http://itbilu.com/nodejs
User-Agent  浏览器的身份标识字符串 User-Agent: Mozilla/.....
```

常用的HTTP响应头

```
Status  通用网关接口的响应头字段，用来说明当前HTTP连接的响应状态。 Status: 200 OK
Set-Cookie  设置HTTP cookie Set-Cookie: UserID=itbilu; Max-Age=3600; Version=1
Server  服务器的名称 Server: nginx/1.6.3
Expires  指定一个日期/时间，超过该时间则认为此回应已经过期 Expires: Thu, 01 Dec 1994 16:00:00
GMT
```

127.七层模型?

128.url的形式?

Web

Flask

129.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")
```

130.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

131.ORM是什么，ORM的优缺点

ORM的全称是：Object Relational Mapping（对象 关系 映射）简单的说，orm是通过使用描述对象和数据之间映射的元数据，将程序中的对象自动持久化到关系数据库中。

ORM需要解决的问题是，能否把对象的数据直接保存到数据库中，又能否直接从数据库中拿到一个对象？要想做到上面两点，则必须要有映射关系。

ORM的优缺点 优点： orm的技术特点，提高了开发效率。可以自动对实体Entity对象与数据库中的Table进行字段与属性的映射；不用直接SQL编码，能够像操作对象一样从数据库中获取数据 缺点： orm会牺牲程序的执行效率和会固定思维模式，在从系统结构上来看，采用orm的系统多是多层系统的，系统的层次太多，效率就会降低，orm是一种完全面向对象的做法，所以面向对象的做法也会对性能产生一定的影响。

132.查找 Django 项目中的性能瓶颈

django-debug-toolbar or Django-silk 该工具能检测出查询操作都来自何处。从而能检测到以下瓶颈： 页面中的重复查询 ORM 调用引起的查询操作次数比预想的多 查询很慢

针对查询操作较多的页面进行优化提速 推荐看 Django 官方的[数据库优化文档](#)

使用 Memcached 或 Redis 对查询进行缓存 压缩 HTML、CSS 和 JavaScript

133.什么是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

134.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

135.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。

136.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。

137.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

138.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)
```

139.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。

140.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，通常来说我们在设计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，就能设计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维表

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于主键也不传递依赖于主键，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

141.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

142.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

143.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
```

144.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

145.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，提高开发效率，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。得益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

146.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。

147.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

148.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。

149.测试工具

unittest:一个通用的测试框架

doctest:一个更简单的模块,是为检查文档而设计的,同时非常适合用来编写单元测试。

locust: 性能测试工具

mock: (Python3 标准库) mock和patch。

factoryboy, Python测试fixtures(setup和teardown)替代库。 faker 生成多种伪数据。



欢迎添加vx: [257735](#) 长按扫码发送「[666](#)」获取10本Python精美电子书
