2021Python面试题-持续更新

目录

Python基础

- 19.如何删除字符串中的前置空格
- 20.Python 中的 pass 语句有什么作用?
- 21.请解释 Python 中的闭包?
- 22.解释 Python 中的//，%和**运算符
- 23.Python 中有多少种运算符，解释算术运算符。
- 24.解释 Python 中的关系运算符。
- 25.解释 Python 中的位运算符
- 26.如何在 Python 使用多进制数字?
- 27.如何获取字典中的所有键?
- 28.问什么标识符不建议使用下划线开头?
- 29.什么是元组的解封装?
- 30.Python3和Python2中 int 和 long的区别?
- 31.列举 Python 中的基本数据类型?
- 32.将"hello world"转换为首字母大写"Hello World
- 33.如何检测字符串中只含有数字?
- 34.将字符串"ilovechina"进行反转
- 35.Python 交换两个变量的值
- 36.Python 里面如何实现 tuple 和 list 的转换?
- 37.Python 中的字符串格式化方式你知道哪些?
- 38.如何对list去重?
- 39.给定两个 list，A 和 B，找出相同元素和不同元素
- 40.如何打乱一个列表的元素?
- 41.字典操作中 del 和 pop 有什么区别
- 42.请合并下面两个字典 a = {“A”：1,“B”：2},b = {“C”：3,“D”：4}
- 43.如何把元组 (“a”,“b”) 和元组 (1,2)，变为字典 {“a”：1,“b”：2}
- 44.如何交换字典 {“A”：1,“B”：2}的键和值
- 45.切片Slice
- 46.什么是切片
- step = 1
- step > 1
- 47.元组的定义
- 48.字符串的三种引号
- 49.字典dict访问
- 50.字典的setdefault函数

Python面向对象

- 1.面向对象
- 2.什么是类和类变量?
- 3.实例和实例化以及实例变量
- 4.数据成员
- 5.方法和静态方法以及类方法
- 6.什么是方法重写
- 7.**__init__**
- 8.self
- 9.类的初始化：new() 和 init()
- 10.**@classmethon**
- 11.**@staticmethod**
- 12.设计的一个面向对象程序设计的完整示例。

- 13.私有属性
- 14.类的继承
- 15.多继承

常用类库

- 1.什么是时间元组?
- 2.使用datetime获取今天日期及前N天日期
- 3.获取以秒为单位的浮点时间time():
- 4.获取人可以直观理解的时间ctime():
- 5.浮点时间转化为直观时间:
- 6.获取格林尼治时间UTC（Coordinated Universal Time，协调时间）格式:
- 7.将UTC格式的时间转化为浮点值的时间:
- 8.strptime 和 strftime 函数
- 9.返回本地区当前日期时间datetime对象
- 10.返回数组：（年、第多少周、星期几）
- 11.如何用Python删除一个文件?
- 12.python如何copy一个文件?
- 13.python如何打开文件?
- 14.python如何重命名文件?
- 15.python如何创建目录?
- 16.python如何删除目录?
- 17.python如何进行文件定位?
- 18.python如何读取键盘输入?
- 19.python如何关闭文件?
- 20.python如何向文件写入数据?
- 21.python如何从文件读取数据?

Python进阶

- 1.写函数，接收两个数字参数，返回最大值
- 2.写函数，获取传入列表的所有奇数位索引对应的元素，并将其作为新列表返回。
- 3.写函数，检查传入的字符串是否含有空字符串，返回结果，包含空字符串返回True，不包含返回False
- 4.定义一个函数,实现两个数四则运算,要注意有3个参数,分别是运算符和两个运算的数字.
- 5.filter、map、reduce 的作用?
- 6.请实现一个装饰器,通过一次调用使函数重复执行5次。
- 7.如何判断一个值是函数还是方法?
- 8.可更改(mutable)与不可更改(immutable)对象
- 9.匿名函数
- 10.变量作用域
- 11.模块与包
- 12.模块的使用
- 13.包的使用
- 14.File（文件)方法 python3
- open() 方法
- 15.异常处理的定义
- 16.异常处理的意义

- 17.常见的异常
- 18.如何进行异常处理

Python数据库编程

- 1.什么是MySQLdb?
- 2.如何连接数据库?
- 3.如何创建数据库表?
- 4.如何执行数据插入?
- 5.如何执行数据库查询操作?
- 6.如何更新数据库数据?
- 7.如何删除数据库数据?
- 8.如何使用数据库事务?
- 9.python如何操作redis?
- 10.如果redis中的某个列表中的数据量非常大，如何实现循环显示每一个值?
- 11.什么是一致性哈希？Python中是否有相应模块?

PythonWeb开发

- 1.什么是Flask？有什么优点?
- 2.Django和Flask有什么区别?
- 3.Flask-WTF是什么，有什么特点?
- 4.Flask脚本的常用方式是什么?
- 5.如何在Flask中访问会话?
- 6.解释Python Flask中的数据库连接?
- 7.Flask框架有哪些依赖组件?
- 8.Flask蓝图的作用?
- 9.列举使用过的Flask第三方组件?
- 10. 简述Flask上下文管理流程?
- 11.Flask框架默认session处理机制?
- 12.django请求的生命周期?
- 13.列举django中间件的5个方法？以及django中间件的应用场景?
- 14.django rest framework框架中都有那些组件?
- 15.django rest framework如何实现的用户访问频率控制?
- 16.django中如何实现单元测试?
- 17.django-debug-toolbar的作用?
- 18.什么是wsgi?
- 19.简述什么是FBV和CBV?
- 20.django中csrf的实现机制
- 21.Django本身提供了runserver，为什么不能用来部署?（runserver与uWSGI的区别）
- 22.Django如何实现websocket?

Python爬虫

Python基础

1.为什么学习Python

因为自己本身想学习编程，在同学和朋友的推荐下，我最后选择了Python。

Python这门语言，入门比较简单，它简单易学，生态圈比较强大，涉及的地方比较多，特别是在人工智能，和数据分析这方面。在未来我觉得是往自动化，人工智能这方面发展的，所以学习了Python。

2.Python和其他语言的区别

从三个方面看Python

语言特点

简洁 优雅 省略了各种大括号和分号,还有一些关键字,类型说明

语言类型

解释型语言,运行的时候是一行一行的解释,并运行,所以调试代码很方便,开发效率高.

第三方库

python是开源的,并且python的定位时任由其发展,应用领域很多
比如Web,运维,自动化测试,爬虫,数据分析,人工智能.Python具有非常完备的第三方库

3.Python的优势

1. 简单：Python奉行简洁主义，易于读写，它使你能够专注于解决问题而不是去搞明白语言本身。
2. 免费：Python是开源软件。这意味着你不用花一分钱便能复制、阅读、改动它，这也是Python越来越优秀的原因——它是由一群希望看到一个更加优秀的Python的人创造并经常改进着的。
3. 兼容性：Python兼容众多平台，所以开发者不会遇到使用其他语言时常会遇到的困扰。
4. 面向对象：Python既支持面向过程，也支持面向对象编程。在面向过程编程中，程序员复用代码，在面向对象编程中，使用基于数据和函数的对象。
5. 丰富的库：Python标准库确实很庞大。它可以帮助你处理各种工作，包括正则表达式、文档生成、单元测试、线程、数据库、网页浏览器、CGI、FTP、电子邮件、XML、XML-RPC、HTML、WAV文件、密码系统、GUI(图形用户界面)、Tk和其他与系统有关的操作。
6. 规范的代码：Python采用强制缩进的方式使得代码具有极佳的可读性。
7. 可扩展性和可嵌入性。如果你需要你的一段关键代码运行得更快或者希望某些算法不公开，你可以把你的部分程序用C或C++编写，然后在你的Python程序中使用它们。你可以把Python嵌入你的C/C++程序，从而向你的程序用户提供脚本功能。

4.Python的解释器种类?

Python解释器主要有以下几个:

1、CPython

官方版本的解释器：CPython。这个解释器是用C语言开发的，所以叫CPython。在命令行下运行python就是启动CPython解释器。CPython是使用最广且被的Python解释器。

2、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，但是执行Python代码的功能和CPython是完全一样的。CPython用>>>作为提示符，而IPython用In [序号]:作为提示符。

3、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。

4、Jython

Jython是运行在Java平台上的Python解释器，可以直接把Python代码编译成Java字节码执行。

5、IronPython

IronPython和Jython类似，只不过IronPython是运行在微软.Net平台上的Python解释器，可以直接把Python代码编译成.Net的字节码。

在这些Python解释器中，使用广泛的是CPython 。

5.python2和python3区别

一、核心类差异

1. Python3 对 Unicode 字符的原生支持。
   Python2 中使用 ASCII 码作为默认编码方式导致 string 有两种类型 str 和 unicode，Python3 只
   支持 unicode 的 string。Python2 和 Python3 字节和字符对应关系为：
2. Python3 采用的是绝对路径的方式进行 import
   Python2 中相对路径的 import 会导致标准库导入变得困难（想象一下，同一目录下有 file.py，如
   何同时导入这个文件和标准库 file）。Python3 中这一点将被修改，如果还需要导入同一目录的文件必
   须使用绝对路径，否则只能使用相关导入的方式来进行导入。
3. Python2 中存在老式类和新式类的区别，Python3 统一采用新式类。新式类声明要求继承 object，
   必须用新式类应用多重继承。
4. Python3 使用更加严格的缩进。Python2 的缩进机制中，1 个 tab 和 8 个 space 是等价的，所
   以在缩进中可以同时允许 tab 和 space 在代码中共存。这种等价机制会导致部分 IDE 使用存在问题。
   Python3 中 1 个 tab 只能找另外一个 tab 替代，因此 tab 和 space 共存会导致报错：
   TabError:
   inconsistent use of tabs and spaces in indentation.

二、废弃类差异

1. print 语句被 Python3 废弃，统一使用 print 函数
2. exec 语句被 python3 废弃，统一使用 exec 函数
3. execfile 语句被 Python3 废弃，推荐使用 exec(open("./filename").read())
4. 不相等操作符"<>"被 Python3 废弃，统一使用"!="
5. long 整数类型被 Python3 废弃，统一使用 int
6. xrange 函数被 Python3 废弃，统一使用 range，Python3 中 range 的机制也进行修改并提高
   了大数据集生成效率
7. Python3 中这些方法再不再返回 list 对象：dictionary 关联的 keys()、values()、items()，zip()，
   map()，filter()，但是可以通过 list 强行转换：
8. mydict={"a":1,"b":2,"c":3}
9. mydict.keys() #<built-in method keys of dict object at 0x00000000040B4C8>
10. list(mydict.keys()) #['a', 'c', 'b']

11. 迭代器 iterator 的 next()函数被 Python3 废弃，统一使用 next(iterator)
12. raw_input 函数被 Python3 废弃，统一使用 input 函数
13. 字典变量的 has_key 函数被 Python 废弃，统一使用 in 关键词
14. file 函数被 Python3 废弃，统一使用 open 来处理文件，可以通过 io.IOBase 检查文件类型
15. apply 函数被 Python3 废弃
16. 异常 StandardError 被 Python3 废弃，统一使用 Exception

三 、修改类差异

1. 浮点数除法操作符“/”和“//”的区别
   “ / ”:
   Python2：若为两个整形数进行运算，结果为整形，但若两个数中有一个为浮点数，则结果为
   浮点数;
   Python3:为真除法，运算结果不再根据参加运算的数的类型。
   “//”:
   Python2：返回小于除法运算结果的最大整数；从类型上讲，与"/"运算符返回类型逻辑一致。
   Python3：和 Python2 运算结果一样。
2. 异常抛出和捕捉机制区别
   Python2
3. raise IOError, "file error" #抛出异常
4. except NameError, err: #捕捉异常
   Python3
5. raise IOError("file error") #抛出异常
6. except NameError as err: #捕捉异常
7. for 循环中变量值区别
   Python2，for 循环会修改外部相同名称变量的值
8. i = 1
9. print （'comprehension: ', [i for i in range(5)]）
10. print （'after: i =', i） #i=4
    Python3，for 循环不会修改外部相同名称变量的值
11. i = 1
12. print （'comprehension: ', [i for i in range(5)]）
13. print （'after: i =', i） #i=1
14. round 函数返回值区别
    Python2，round 函数返回 float 类型值
15. isinstance(round(15.5),int) #True
    Python3，round 函数返回 int 类型值
16. isinstance(round(15.5),float) #True
17. 比较操作符区别
    Python2 中任意两个对象都可以比较
18. 11 < 'test' #True
    Python3 中只有同一数据类型的对象可以比较
19. 11 < 'test' # TypeError: unorderable types: int() < str()

四、第三方工具包差异

五、工具安装问题

6.深拷贝和浅拷贝的区别是什么?

深拷贝和浅拷贝最根本的区别在于是否真正获取一个对象的复制实体，而不是引用。
假设B复制了A，修改A的时候，看B是否发生变化:
如果B跟着也变了，说明是浅拷贝，拿人手短！（修改堆内存中的同一个值）
如果B没有改变，说明是深拷贝，自食其力！（修改堆内存中的不同值）

浅拷贝（shallowCopy）只是增加了一指针指向已存在的内存地址。
深拷贝（deepCopy）是增加了一个指针并且申请了一个新的内存，使这个增加的指针指向这个新的内存

使用深拷贝的情况下，释放内存的时候不会因为出现浅拷贝时释放同一个内存的错误。

浅拷贝：仅仅时指向被复制的内存地址，如果原地址发生改变，那么浅复制出来的对象也会相应的改变。
深拷贝：在计算机中开辟一块新的内存地址用于存放复制的对象。

但是在python中，对浅拷贝和深拷贝，还需要分数据类型是可变类型还是不可边类型。
可变数据类型 不可变数据类型

	可变数据类型	不可变数据类型
浅拷贝	只对可变类型的第一层对象进行拷贝，对拷贝的对象会开辟新的内存空间进行存储，子对象不进行拷贝	不会给拷贝的对象开辟新的内存空间，而只是拷贝了这个对象的引用。
深拷贝	会对该对象到最后一个可变类型的每一层对象就行拷贝，对每一层拷贝的对象都会开辟新的内存空间进行存储	如果子对象没有可变类型则不会进行拷贝，而只是拷贝了这个对象的引用，否则会对该对象到最后一个可变类型的每一层对象就行拷贝，对每一层拷贝的对象都会开辟新的内存空间进行存储

7.位和字节的关系?

位: "位(bit)"是电子计算机中最小的数据单位。每一位的状态只能是0或1。

字节: 8个二进制位构成1个"字节(Byte)"，它是存储空间的基本计量单位。1个字节可以储存1个英文字母或者半个汉字，换句话说，1个汉字占据2个字节的存储空间。

8.b、B、KB、MB、GB 的关系?

```
b 比特bit ``/` `位
B—字节       ``1` `B ``=` `8b``（``8``个bit``/` `位）一个字节（byte）等于
``8``位（bit）
KB—千比特    ``1` `kB ``=` `1024` `B (kB ``-` `kilobajt)
MB—兆比特    ``1` `MB ``=` `1024` `kB (MB ``-` `megabajt)
GB—吉比特    ``1` `GB ``=` `1024` `MB (GB ``-` `gigabajt)
```

9.python递归的最大层数?

```
import sys
sys.getrecursionlimit()     # 获取最大递归层数   默认是1000（0-999）
sys.setrecursionlimit(1200)     # 设置最大递归层数
```

10.解释 Python 中的 help() 函数和 dir() 函数。

help() 函数返回帮助文档和参数说明:

运行结果如下:

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() 函数返回对象中的所有成员 (任何类型)

11.当退出 Python 时是否释放所有内存分配?

答案是否定的。那些具有对象循环引用或者全局命名空间引用的变量，在 Python 退出是往往不会被释放

另外不会释放 C 库保留的部分内容。

12.什么是 Python 字典?

字典是我在 C++和 Java 中没有见过的数据结构，它拥有键-值对

3

字典是可变的，我们也可以用推导式的方式创建它.

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

要了解更多字典的内容请点击 Python Dictionaries（https://data-flair.training/blogs/python-dictionaries/）

13.能否解释一下 *args 和 **kwargs?

如果我们不知道将多少个参数传递给函数，比如当我们想传递一个列表或一个元组值时，就可以使用*args。

3

2

1

4

7

当我们不知道将会传入多少关键字参数时，使用**kwargs 会收集关键字参数。

a.1

b.2

c.7

使用 args 和 kwargs 作为参数名只是举例，可以任意替换。

对于 Python 的基础题任何疑问，请在评论区提问。

14.什么是负索引?

我们先创建如下列表：

与正索引不同，负索引是从右边开始检索。

6

同样可以用于列表的切片：

[3, 4, 5, 6, 7]

15. 如何随机打乱列表中元素，要求不引用额外的内存空间?

我们用 random 包中的 shuffle() 函数来实现。

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

16.解释 Python 中的 join() 和 split() 函数

join() 函数可以将指定的字符添加到字符串中。

‘1,2,3,4,5’

split() 函数可以用指定的字符分割字符串

[‘1’, ‘2’, ‘3’, ‘4’, ‘5’]

17.Python 区分大小写吗?

验证 Python 是否区分大小写的方法是测试 myname 和 Myname 在程序中是不是算同一个标识符。观察以下代码的返回结果：

Myname

NameError: name 'Myname' is not defined

如你所见，这里出现了 NameError，所以 Python 是区分大小的语言。

18.Python 中标识符的命名规则?

Python 中的标识符可以是任意长度，但必须遵循以下命名规则：

1. 只能以下划线或者 A-Z/a-z 中的字母开头。
2. 其余部分只能使用 A-Z/a-z/0-9。
3. Python 标识符区分大小写。
4. 关键字不能作为标识符。Python 有以下这些关键字：

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	

19.如何删除字符串中的前置空格

前置空格是第一个非空格字符前的所有空格，使用 lstrip() 函数来删除.

' Ayushi '

如图这个字符串既包含前置空格也包含后置空格. 调用 lstrip() 函数去除了前置空格。如果想去除后置空格，使用 rstrip() 函数。

' Ayushi'

20.Python 中的 pass 语句有什么作用?

我们在写代码时，有时可能只写了函数声明而没想好函数怎么写，但为了保证语法检查的正确必须输入一些东西。在这种情况下，我们使用 pass 语句。

类似的 break 语句可以跳出循环。

0

1

2

continue 语句可以跳到下一轮循环。

0

1

2

4

5

6

21.请解释 Python 中的闭包?

如果在一个内部函数里。对在外部作用域（但不是在全局作用域）的变量进行引用，那么内部函数就是一个闭包。

7

闭包的详细解释请点击 Closures in Python。（https://data-flair.training/blogs/python-closure/）

22.解释 Python 中的//，%和**运算符

//运算符执行地板除法，返回结果的整数部分 (向下取整)。

3

用/符号除法结果为 3.5。

符号表示取幂. ab 返回 a 的 b 次方

1024

% 是取模符号。返回除法后的余数。

6

0.5

23.Python 中有多少种运算符，解释算术运算符。

这类面试问题可以判断你的 Python 功底，可以举一些实例来回答这类问题。

在 Python 中我们有 7 中运算符:算术运算符、关系 (比较) 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。

\1. 加号 (+) 将两个对象的值相加。

15

\2. 减号 (-) 将第一个对象的值减去第二个对象的值。

-1

\3. 乘号 (*) 将两个对象的值相乘。

56

\4. 除号 (/) 将第一个对象的值除以第二个对象的值。

0.875

1.0

关于地板除法、取模和取幂，请参考上一个问题。

24.解释 Python 中的关系运算符。

关系运算符用来比较两个对象。

\1. 判断小于 (<)：如果符号左边的值比右边小则返回 True。

False

\2. 判断大于 (>)：如果符号左边的值比右边大则返回 True。

True

出现上面的错误结果是因为 Python 的浮点运算存在一些 Bug。

\3. 判断小于等于 (<=)：如果符号左边的值小于或等于右边则返回 True。

True

\4. 大判断于等于 (>=)：如果符号左边的值大于或等于右边则返回 True。

True

\5. 判断等于 (==) 如果符号两边的值相等则返回 True。

True

\6. 判断不等于 (!=) 如果符号两边的值不等则返回 True。

True

True

25.解释 Python 中的位运算符

此运算符按二进制位对值进行操作。

\1. 与 (&) 返回按位与结果

2

\2. 或 (|) 返回按位或结果

3

\3. 异或 (^) 返回按位异或结果

1

\4. 取反 (~) 返回按位取反结果

-3

\5. 左移位 (<<) 将符号左边数的二进制左移右边数位

4

1 的二级制 001 左移 2 位变成 100 也即十进制的 4

\6. 右移位 (>>)

1

想了解关于位运算符的更多内容请点击 Operators in Python（https://data-flair.training/blogs/python-operators/）

26.如何在 Python 使用多进制数字?

除十进制以外，在 Python 中还可以使用二进制、八进制、十六进制。

\1. 二进制数有 0 和 1 组成，我们使用 0b 或 0B 前缀表示二进制数

10

使用 bin() 函数可以将数字转换为二进制

‘0b1111’

\2. 八进制数由数字 0-7 组成，使用前缀 0o 或 0O 表示 8 进制数

‘0o10’

\3. 十六进数由数字 0-15 组成，使用前缀 0x 或者 0X 表示 16 进制数

‘0x10’

‘0xf’

27.如何获取字典中的所有键?

使用 keys() 来获取字典中的所有键

28.问什么标识符不建议使用下划线开头?

因为在 Python 中以下划线开头的变量为私有变量，如果你不想让变量私有，就不要使用下划线开头。

29.什么是元组的解封装?

首先我们来介绍元组封装：

(3, 4, 5)

将 3，4，5 封装到元组 mytuple 中。

现在我们要将这些值解封装到变量 x，y，z 中

12

30.Python3和Python2中 int 和 long的区别?

python2有非浮点数准备的`int`和`long`类型。`int`类型最大值不能超过sys.maxint，而且这个最大值是平台相关的。

可以通过在数字的末尾附上一个 L 来定义长整型，显然，它比`int`类型表示的数字范围更大。

python3里，只有一种整数类型`int`,大多数情况下，和python 2 中的长整型类似。

31.列举 Python 中的基本数据类型?

Python3中有6种基本数据类型，列表（list）、元组（tuble）、字典（dict）、集合（sets）、字符串（string）、数字（digit）。

32.将"hello world"转换为首字母大写"Hello World

```
“hello world”.title()
```

33.如何检测字符串中只含有数字?

```
'123bp'.isdigit()----------返回True or False
#或者使用正则：
bool(re.search(r’\d’,’qw123’))
#或者使用Unicode码：
if uchar >= u'\u0030' and uchar <= u'\u0039'
```

34.将字符串"ilovechina"进行反转

```
"ilovechina"[::-1]
```

35.Python 交换两个变量的值

a, b = b, a

36.Python 里面如何实现 tuple 和 list 的转换?

```
tuple(list)
```

37.Python 中的字符串格式化方式你知道哪些?

```
'I %s her %s'%('love','cat')
'I {a} her {b}'.format(a='love',b='cat')
f'I {a} her {b}' #【Python3.6推荐写法】
```

38.如何对list去重?

1）先建立一个新的空列表，通过遍历原来的列表，再利用逻辑关系not in 来去重。此方法保证了列表的顺序性。

```
li=[1,2,3,4,5,1,2,3]
new_li=[]
for i in li:
    if i not in new_li:
        new_li.append(i)
print(new_li)
```

2）将列表转化为集合再转化为列表，利用集合的自动去重功能。简单快速。缺点是：使用set方法无法保证去重后的顺序。

```
li=[1,2,3,4,5,1,2,3]
new_li=list(set(li))
new_li.sort(key=li.index)
print(new_li)
```

39.给定两个 list，A 和 B，找出相同元素和不同元素

```
A、B 中相同元素：set(A)&set(B)
A、B 中不同元素：set(A)^set(B)
```

40.如何打乱一个列表的元素?

```
random.shuffle(l1)
```

41.字典操作中 del 和 pop 有什么区别

del d[key] 直接删除
d.pop(key) 返回键的值

42.请合并下面两个字典 a = {“A”：1,“B”：2},b = {“C”：3,“D”：4}

```
a.update(b)
```

43.如何把元组 (“a”,“b”) 和元组 (1,2)，变为字典 {“a”：1,“b”：2}

```
dict(zip(a,b))
```

44.如何交换字典 {“A”：1,“B”：2}的键和值

```
{value:key for key value in dict.items()}
```

45.切片Slice

切片，是一个比较生疏的名词，这是现代计算机编程语言或者说Python里的一个概念，大致意思是从一个集合里切出一块来，就像切一块豆腐，一刀下去切出两块豆腐，问题是两刀能切出几块？开个小玩笑！

先看一个函数range、返回值是列表，内容和传入range的函数有关。

```
a = range(1, 21)
print a
```

结果

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
```

明白了，range可以产生从1到20共20个元素的列表，range的第二个参数不包含这个值，但包含第一个参数的值，每个元素值相差为1。

46.什么是切片

回到正题，切片是对有序的集合而言，意思从有序集合里提取数据构成子集集合，给定提取的起点start、终点end以及方向上的步长step，**能否切出非空子集，起点start需能沿步长方向上到达终点**。

字符串和列表、元组都是有序集合，均可实现切片操作，以列表为例给出切片的语法格式如下：

```
子集名 = 有序集合名[start : end : step]
```

上边range(1, 21)和切片里的start、end是呼应一致的，含起点start对应的值而不含终点end所对应的值。

step = 1

子集的构成是从start开始每step取一个数据一至到end-1位置结束，step默认为1时，可以省略不写。

```
s = "python"
a = list(s)
print a
b = a[2:5]
print b
```


程序的结果如下：

```
['p', 'y', 't', 'h', 'o', 'n']
['t', 'h', 'o']
```

0	1	2	3	4	5
'p'	'y'	't'	'h'	'o'	'n'

从print b这条语句的打印结果['t', 'h', 'o']可知，'t'在a列表的index为2，'n'字符的index为5，那么a[2:5]的step为1，从't'开始逐个取回字符't'、'h'、'o'组成新的列表的子集b，而字符'n'的index为5，5作为取切片操作a[2:5]里的终点，其字符'n'不取回。从例子可以看出step 为1可以理解为“逐个取”。

step > 1

step可以大于1，这个时候对于step的理解可以这样认为，每step个取一个。

```
s = "python"
a = list(s)
print a
b = a[2:5:2]
print b
```

程序的结果如下：

```
['p', 'y', 't', 'h', 'o', 'n']
['t', 'o']
```

对于列表a取切片a[2 : 5 : 2]， 从index为2开始，每2个元素为一组取每组的第一个数据值，一只到index为5结束，但不取index为5的数据。

a[2 : 5 : 2]具体操作是这样的，从index为2的字符't'开始，到index为5的字符'n'之前，每2个为一组('t', 'h')、('o', 'n')取每组的第一个元素值't'、'o'即结果子集b的值。

再看一个例子：

```
a = range(12)
print a
b = a[1 : 10 : 3]
print b
```

结果是：

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
[1, 4, 7]
```

从两行输出结果的第一行可以看出range函数可以产生一个从0开始到小于12的整数的列表，共12个数据。

第二行的输出[1, 4, 7]是怎么产生的呢?这个和对a列表的切片操作有关，a[1 : 10 : 3]，这里a[1:10] = [1, 2, 3, 4, 5, 6, 7, 8, 9]，每3个一组，从每组取第一个，即 [1, 2, 3],[4, 5, 6], [7, 8, 9]这三组，取每组第一个1、4、7构成b这个子集，最后得到的b = [1, 4, 7]。

47.元组的定义

定义一个元组很简单，将一堆数据用圆括号括起来，用逗号间隔各个元素即可定义一个元组，元组里的数据是只读的不可被修改。

```
变量名 = (元素值序列)
```

48.字符串的三种引号

```
print 'hello'
print "hello"
print '''hello'''
print """hello"""
```

一般三引号可以在Python源代码里用作多行注释或定义多行的字符串，另外Python可以用井号(#)进行单行注释。

```
a = 12
b = 13
"""
求和
"""
c = a + b
```

字符串是有序不可修改的序列，可以通过索引或者for循环体访问字符串里的各个元素值。

```
s = "python"
print s[1]
for c in s:
    printc
```


49.字典dict访问

通过键值

字典的数据是由key:value对儿构成的每项数据，那么想访问某项数据的value需要可以通过[]运算来获得，其语法结构如下：

```
字典名[key]
```

字典的get函数

字典有个等价函数get可以获得这种方式的相同结果，语法结构如下：

```
字典名.get(key)
```

get函数返回值就是这个key所对应的值。

例如：

```
d = {1 : 2, "a" : 13, 12.4 : 77}
print d.get("a")
```

但是如果无key的话，get返回None而不会报错发生异常，而用[]运算即`字典名[key]`则会报错异常。

50.字典的setdefault函数

还有一个函数setdefault() 函数，它和get函数类似,返回指定键的值，如果键不在字典中，将会添加键并将值设置为一个指定值，默认为None。

get() 和 setdefault() 区别： setdefault() 返回的键如果不在字典中，会添加键（更新字典），而 get() 不会添加键。

参考链接

https://www.jianshu.com/p/373cdf20039d

https://blog.csdn.net/qdPython/article/details/101286827

https://blog.csdn.net/pangzhaowen/article/details/80650478

https://blog.csdn.net/Xidian2850/article/details/108171194

http://liao.cpython.org/

https://baijiahao.baidu.com/s?id=1607651363840614527&wfr=spider&for=pc

https://www.cnblogs.com/tianyiliang/p/7845932.html

https://blog.csdn.net/qq_40082282/article/details/102914050

Python面向对象

1.面向对象

抽象：提取现实世界中某事物的关键特性，为该事物构建模型的过程。对同一事物在不同的需求下，需要提取的特性可能不一样。得到的抽象模型中一般包含：属性（数据）和操作（行为）。这个抽象模型我们称之为类。对类进行实例化得到对象。

封装：封装可以使类具有独立性和隔离性；保证类的高内聚。只暴露给类外部或者子类必须的属性和操作。类封装的实现依赖类的修饰符（public、protected和private等）

继承：对现有类的一种复用机制。一个类如果继承现有的类，则这个类将拥有被继承类的所有非私有特性（属性和操作）。这里指的继承包含：类的继承和接口的实现。

多态：多态是在继承的基础上实现的。多态的三个要素：继承、重写和父类引用指向子类对象。父类引用指向不同的子类对象时，调用相同的方法，呈现出不同的行为；就是类多态特性。多态可以分成编译时多态和运行时多态。

2.什么是类和类变量?

用来描述具有相同属性和方法的对象的集合。它定义了该集合中每个对象所共有的属性和方法。其中的对象被称作类的实例。

类变量：类变量是所有实例公有的变量。类变量定义在类中，但在方法体之外。

3.实例和实例化以及实例变量

实例：也称对象。通过类定义的初始化方法，赋予具体的值，成为一个”有血有肉的实体”。

实例化：创建类的实例的过程或操作。

实例变量：定义在实例中的变量，只作用于当前实例。

4.数据成员

类变量、实例变量、方法、类方法、静态方法和属性等的统称。

5.方法和静态方法以及类方法

方法：类中定义的函数。

静态方法：不需要实例化就可以由类执行的方法

类方法：类方法是将类本身作为对象进行操作的方法。

6.什么是方法重写

如果从父类继承的方法不能满足子类的需求，可以对父类的方法进行改写，这个过程也称override。

7.__init__

可以成为类的实例对象的构造函数，每次通过类创建一个该类的对象是调用此函数，所以其下的以sefl.前缀的变量是每个创建好了的实例(化)对象的所独有的。换句话说，有多少个类的对象内存里就有多少份这个实例对象变量存在。就像生产了多少小汽车就有多少个方向盘似的。

8.self

代表运行时的类的实例对象本身，一般在类的内部设计时出现，在程序里使用对象编程时不用self。在实例对象的成员函数里以self.前缀的变量是实例对象的成员变量，没有self.的变量是本方法函数的局部变量。

9.类的初始化：new() 和 init()

new()方法用来实例化最终的类对象，在类创建之前被调用，它在类的主体被执行完后开始执行。

init()方法是在类被创建之后被调用，用来执行其他的一些输出化工作

当我们构造元类的时候，通常只需要定一个init()或new()方法，但不是两个都定义。但是，如果需要接受其他的关键词参数的话，这两个方法就要同时提供，并且都要提供对应的参数签名。

10.@classmethon

这个关键字是修饰器，修饰也是说下面的函数是类的方法函数而不是类的对象的方法函数。

11.@staticmethod

这个也是修饰器，说明接下来的函数是一个静态函数，和实例对象的成员函数、类函数的区别主要在第一个形参，既无self又无cls。可以被类或对象直接调用。 差不多解释完了，下面来看一个具体的类的实例程序。

12.设计的一个面向对象程序设计的完整示例。


```
# coding:utf-8
class Horse(object):
    variety = "大宛马"
    def __init__(self, name = "green", height = 0.5, length = 1.3, sex =
"male"):
        # self.name是成员变量，name是形参、局部变量
        self.name = name
        self.height = height
        self.length = length
        self.sex = sex
        print "A baby horse is born called", self.name

    def print_info(self):
        print self.name, self.height, self.length, self.sex,
Horse.variety#,Horse.address
        Horse.print_variety() # 在对象方法里通过类调用类方法，避免
        Horse().print_ci(200, 100)# 对象调用静态方法
        Horse.print_ci(200, 100) # 类调用静态方法

    @staticmethod
    def print_ci(x, y):
        print x, y

    @classmethod
    def pp(cls):
        # 类使用类变量
        print cls.variety, Horse.variety, cls.address
```

```
        #cls.print_variety()
        print Horse().name  # 对象使用对象的成员变量
    @classmethod
    def print_variety(cls):
        cls.address = "xi'an"
        print "type", type(cls.address)
        print cls.variety, Horse.variety, cls.address
        Horse.pp()# 类调用类方法
        Horse().print_ci(100, 100)# 对象调用静态方法

a = Horse("xiaoxuanfeng")
b = Horse("pilihuo", sex = "female")
a.print_info()
b.print_info()
Horse.print_variety()
print "*" * 20
Horse.pp() # 类调用类方法
Horse.print_ci(12, 23)# 类外类调用静态方法
a.print_ci(23, 31) # 类外对象调用静态方法
```

13.私有属性

变量和函数

- 定义私有变量

```
class aa(object):
    def __init__(self, w, v):
        self.x = w
        self.__y = v
    def p(self):
        print "  x", self.x
        print "__y", self.__y
ai = aa(12, 13)
ai.p()
```

程序的执行结果:

```
  x 12
__y 13
```

- 定义私有函数,不能在类的外部调用。

```
class aa(object):
    def __init__(self, w, v):
        self.x = w
        self.__y = v
    def p(self):
        print "  x", self.x
```

```
        print "__y", self.__y
        self.__q()
    def __q(self):
        print "private method of class aa"
ai = aa(12, 13)
ai.p()
#aa.__q()
#ai.__q()
#ai._aa__q()
```

程序的执行结果：

```
  x 12
__y 13
private method of class aa
```

__q函数是类aa的私有函数，可以在类内部使用，但不能在类之外使用。

14.类的继承

假如已经有几个类，而类与类之间有共同的变量属性和函数属性，那就可以把这几个变量属性和函数属性提取出来作为基类的属性。而特殊的变量属性和函数属性，则在本类中定义，这样只需要继承这个基类，就可以访问基类的变量属性和函数属性。可以提高代码的可扩展性。通过继承可以快速扩展和实现函数的多样性

举例：

```
class aa(object):
    def __init__(self, v):
        self.x = v
        self.__pa = 10
    def p(self):
        print "class aa's instance method"
    def info(self):
        print "info of aa instance"
class cc(aa):
    def __init__(self, v):
        self.z = v
        self__pc = 10
    def p(self):
        print "class cc's instance method"
a = aa(10)
c = cc(30)
a.p()
a.info()
c.p()
c.info()
```

程序执行结果：

```
class aa's instance method
info of aa instance
class cc's instance method
info of aa instance
```

15.多继承

Python的类允许可以有多个父类，从左至右有顺序要求，这和后续搜索查找数据和函数有关系。

```
  class 子类(父类1，  父类2，  ...)
```

多继承时子类的父类间和子类要满足人类的伦理规则。

举例:

```
class aa(object):
    def __init__(self, v):
        self.x = v
    def px(self):
        print self.x

class bb(object):
    def __init__(self, v):
        self.y = v
    def py(self):
        print self.y

class cc(aa, bb):
    def __init__(self, v, v1 = 100):
        print "cc"
        super(cc, self).__init__(v1)
        #aa.__init__(self, v1)
        #bb.__init__(self, v1)
        print "ccx"
        self.z = v
    def pz(self):
        print self.z

a = aa(12)
a.px()
b = bb(13)
b.py()
c = cc(14)
print dir(c)
c.pz()
```


程序的执行结果:

```
12
13
cc
ccx
['__class__', ..., '__weakref__', 'px', 'py', 'pz', 'x', 'z']
14
```

参考链接

https://blog.csdn.net/weixin_39628063/article/details/111455821

http://liao.cpython.org/27classobject/

https://blog.csdn.net/weixin_35390379/article/details/112022047

常用类库

1.什么是时间元组?

很多Python函数用一个元组装起来的9组数字处理时间:

序号	字段	值
0	4位数年	2008
1	月	1 到 12
2	日	1到31
3	小时	0到23
4	分钟	0到59
5	秒	0到61 (60或61 是闰秒)

序号	字段	值
6	一周的第几日	0到6 (0是周一)
7	一年的第几日	1到366 (儒略历)
8	夏令时	-1, 0, 1, -1是决定是否为夏令时的旗帜

上述也就是struct_time元组。这种结构具有如下属性：

序号	属性	值
0	tm_year	2008
1	tm_mon	1 到 12
2	tm_mday	1 到 31
3	tm_hour	0 到 23
4	tm_min	0 到 59
5	tm_sec	0 到 61 (60或61 是闰秒)
6	tm_wday	0到6 (0是周一)
7	tm_yday	1 到 366(儒略历)
8	tm_isdst	-1, 0, 1, -1是决定是否为夏令时的旗帜

2.使用datetime获取今天日期及前N天日期

问：在Python中如何获取今天的日期？如何获取前N天的日期？
答：使用datetime模块可以完成日期获取任务。
（1）获取当天日期

```
import datetime
# 获取当天日期
# strftime()函数是将time信息输出为想要的格式，如‘2020-08-03’
date_today = datetime.datetime.now().strftime('%Y-%m-%d')
# 输出带有小时分钟的日期
data_today2 = datetime.datetime.now().strftime('%Y-%m-%d %H-%M-%S')
```

（2）获取前N天日期

```
# 通过timedelta(N)函数完成
date_3today_ago = (datetime.datetime.now() -
datetime.timedelta(3)).strftime('%Y-%m-%d')
```

3.获取以秒为单位的浮点时间time():

```
>>> import time
>>> print time.time()#获取当前时间的浮点值，单位为秒
1369031293.33
>>>
```

4.获取人可以直观理解的时间ctime():

```
print time.ctime()
Mon May 20 14:29:30 2013#获取人能理解的直观时间
```

5.浮点时间转化为直观时间:

```
>>> t = time.time()#浮点时间
>>> print t
1369034676.69
>>> print time.ctime(t)#浮点时间转化为直观时间
Mon May 20 15:24:36 2013
```

6.获取格林尼治时间UTC（Coordinated Universal Time，协调时间）格式:

```
>>> print time.gmtime()#获取UTC格式的当前时间
time.struct_time(tm_year=2013, tm_mon=5, tm_mday=20, tm_hour=6,
tm_min=37, tm_sec=45, tm_wday=0, tm_yday=140, tm_isdst=0)
```

7.将UTC格式的时间转化为浮点值的时间:

```
>>> gmt = time.gmtime()#UTC格式的时间
>>> print gmt
time.struct_time(tm_year=2013, tm_mon=5, tm_mday=20, tm_hour=6,
tm_min=48, tm_sec=13, tm_wday=0, tm_yday=140, tm_isdst=0)
>>> print time.mktime(gmt)#将UTC格式的时间转化为浮点值的时间
1369003693.0

>>> lt = time.localtime()#将UTC格式当前时区当前时间
>>> print lt
time.struct_time(tm_year=2013, tm_mon=5, tm_mday=20, tm_hour=14,
tm_min=49, tm_sec=25, tm_wday=0, tm_yday=140, tm_isdst=0)
>>> print time.mktime(lt)##将UTC格式的时间转化为浮点值的时间
1369032565.0
```

8.strptime 和 strftime 函数

```
时间.strftime(时间格式)
datetime.strptime(字符串,时间格式)

示范:
datetime.strptime(str,'%Y-%m-%d')
datetime.now().strftime("%Y-%m-%d %H:%M:%S")
```

###

9.返回本地区当前日期时间datetime对象

```
datetime.today()
# 输出 : datetime.datetime(2019, 12, 9, 13, 27, 54, 693978)
```

10.返回数组: （年、第多少周、星期几）

```
d = datetime(2019,12,6,13,30,50)
d.isocalendar()
# 输出 : (2019, 49, 5)
```

11.如何用Python删除一个文件?

使用os.remove(filename)或者os.unlink(filename)

12.python如何copy一个文件?

shutil模块有一个copyfile函数可以实现文件拷贝

13.python如何打开文件?

open(file_name)

14.python如何重命名文件?

os.rename(current_file_name, new_file_name)

15.python如何创建目录?

os.mkdir("newdir")

16.python如何删除目录?

os.rmdir('dirname')

17.python如何进行文件定位?

tell()方法告诉你文件内的当前位置, 换句话说，下一次的读写会发生在文件开头这么多字节之后。

seek（offset [,from]）方法改变当前文件的位置。Offset变量表示要移动的字节数。From变量指定开始移动字节的参考位置。

如果from被设为0，这意味着将文件的开头作为移动字节的参考位置。如果设为1，则使用当前的位置作为参考位置。如果它被设为2，那么该文件的末尾将作为参考位置。

18.python如何读取键盘输入?

raw_input函数
raw_input([prompt]) 函数从标准输入读取一个行，并返回一个字符串（去掉结尾的换行符）。

input函数
input([prompt]) 函数和 raw_input([prompt]) 函数基本类似，但是 input 可以接收一个Python表达式作为输入，并将运算结果返回。

19.python如何关闭文件?

File 对象的 close（）方法刷新缓冲区里任何还没写入的信息，并关闭该文件，这之后便不能再进行写入。

当一个文件对象的引用被重新指定给另一个文件时，Python 会关闭之前的文件。用 close（）方法关闭文件是一个很好的习惯。

20.python如何向文件写入数据?

File 对象的write()方法可将任何字符串写入一个打开的文件。需要重点注意的是，Python字符串可以是二进制数据，而不是仅仅是文字。

write()方法不会在字符串的结尾添加换行符('\n'):

21.python如何从文件读取数据?

File 对象的read())方法从一个打开的文件中读取一个字符串。需要重点注意的是，Python字符串可以是二进制数据，而不是仅仅是文字。

参考链接

https://blog.csdn.net/LI20132017/article/details/107769363

https://www.cnblogs.com/fengff/p/8674681.html

https://blog.csdn.net/weixin_33853794/article/details/86739039

https://www.runoob.com/python/python-tutorial.html

Python进阶

1.写函数，接收两个数字参数，返回最大值

```
def res_max(number1,number2):
    l1 = []
    l1.append(number1)
    l1.append(number2)
    return max(l1)
```

2.写函数，获取传入列表的所有奇数位索引对应的元素，并将其作为新列表返回。

```
def getnewlist(mylist):
    list1=[];
    for i in range(0,len(mylist)):
        if i%2!=0:
        list1.append(mylist[i])
    return list1
```

3.写函数，检查传入的字符串是否含有空字符串，返回结果，包含空字符串返回True，不包含返回False

```
def str_spack(string):
    if string.find(' '):
        return True
    else:
        return False
```

4.定义一个函数,实现两个数四则运算,要注意有3个参数,分别是运算符和两个运算的数字.

```
def arithmetic(number1, number2, symbol):

    if symbol == '+':
        s = number1 + number2
    elif symbol == '-':
        s = number1 - number2
    elif symbol == '*':
        s = number1 * number2
    elif symbol == '/':
        s = number1 / number2
    return s

方法二：
def getresult(num1,fh,num2):
    str1=str(num1)+fh+str(num2)
    return eval(str1)
print(getresult(10,'*',20))
```


5.filter、map、reduce 的作用?

1. filter---过滤条件用的
2. map--将内容里的元素 逐个处理
3. reduce--用于做累计算的

6.请实现一个装饰器,通过一次调用使函数重复执行5次。

```
import  time
def wrapper(func):
    def inner(*args,**kwargs):
        for i in range(5):
            time.sleep(0.5)
            func(*args,**kwargs)
    return inner
@wrapper
def func():
    print('a')

func()
```

7.如何判断一个值是函数还是方法?

用type()来判断，如果是method为方法，如果是function则是函数。

括号中写入变量名,,不要有括号什么别的符号之类的

8.可更改(mutable)与不可更改(immutable)对象

在 python 中，strings, tuples, 和 numbers 是不可更改的对象，而 list,dict 等则是可以修改的对象。

- 不可变类型：变量赋值 **a=5** 后再赋值 **a=10**，这里实际是新生成一个 int 值对象 10，再让 a 指向它，而 5 被丢弃，不是改变a的值，相当于新生成了a。
- 可变类型：变量赋值 **la=[1,2,3,4]** 后再赋值 **la[2]=5** 则是将 list la 的第三个元素值更改，本身la没有动，只是其内部的一部分值被修改了。

python 函数的参数传递：

- 不可变类型：类似 c++ 的值传递，如 整数、字符串、元组。如fun（a），传递的只是a的值，没有影响a对象本身。比如在 fun（a）内部修改 a 的值，只是修改另一个复制的对象，不会影响 a 本身。
- 可变类型：类似 c++ 的引用传递，如 列表，字典。如 fun（la），则是将 la 真正的传过去，修改后fun外部的la也会受影响

python 中一切都是对象，严格意义我们不能说值传递还是引用传递，我们应该说传不可变对象和传可变对象。

9.匿名函数

python 使用 lambda 来创建匿名函数。

- lambda只是一个表达式，函数体比def简单很多。
- lambda的主体是一个表达式，而不是一个代码块。仅仅能在lambda表达式中封装有限的逻辑进去。
- lambda函数拥有自己的命名空间，且不能访问自有参数列表之外或全局命名空间里的参数。

- 虽然lambda函数看起来只能写一行，却不等同于C或C++的内联函数，后者的目的是调用小函数时不占用栈内存从而增加运行效率。

10.变量作用域

一个程序的所有的变量并不是在哪个位置都可以访问的。访问权限决定于这个变量是在哪里赋值的。

变量的作用域决定了在哪一部分程序你可以访问哪个特定的变量名称。两种最基本的变量作用域如下：

- 全局变量
- 局部变量

11.模块与包

模块首先是一个含有源代码的文件在Python里以.py结尾，文件里可以有函数的定义、变量的定义或者对象(类的实例化)的定义等等内容。如果一个项目的代码量较大，函数较多，最好把一个文件分为多个文件来管理，这样总程序脉络清晰易于维护和团队分工协作，这就是Python里存在模块的意义所在。模块名就是文件名(不含.py)，例如假设有一个模块：xopowo.py那么模块名为xopowo。

但当一个项目的模块文件不断的增多，为了更好地管理项目，通常将功能相近相关的模块放在同一个目录下，这就是包，故包从物理结构上看对应于一个目录，一个特殊要求，包目录下必有一个空的**init**.py文件，这是包区分普通目录的标签或者说是标志。包下可以又有包称为子包，子包也是一个目录，子包的目录下也得有一个空的**init**.py文件。这样就构成了分级或者说分层的项目管理体系。

12.模块的使用

模块，可是Python自带的、而外安装的或者开发者自己写的，在一个文件里使用模块很简单用import即可，import有点像C语言的include。

以Python2的内建模块datetime为例，讲解一下模块的基本使用。

在新程序里使用datetime模块可以有两种方式：方式一是把模块引入，而模块里的函数的使用需要用点运算的方式来来使用。

```
import datetime
birthday = datetime.date(2011,7,23)
print birthday
```

而文件引用模块里某函数还有另外一种方式就是用from import来直接引入某模块里的某函数，即方式二。

```
from datetime import date,time
birthday = date(2011,7,23)
print birthday
```

使用方式二文件只能用import后列出的函数，而模块datetime里的其他函数无法在本文件里使用，所以一种特殊的写法如下：

```
from datetime import *
```

也就是说datetime里的所有函数在本程序里均可使用。

13.包的使用

包，实际是更大规模的以目录形式存在的模块集合，包可以含子包，包区别于目录是包的目录下有一个空的**init**.py文件。包和模块一样有Python自带的包，也可以通过工具安装一些包，例如numpy就是数据科学领域比较常用的一个包，需额外安装，当然也可以自己开发一些包。

以Python2自带的包multiprocessing为例，其下还有子包dummy。

```
liao@liao:/usr/lib/python2.7/multiprocessing$ ls
connection.py   forking.py   heap.pyc      managers.py   pool.pyc
queues.py     reduction.pyc     synchronize.py
connection.pyc  forking.pyc  __init__.py   managers.pyc  process.py
queues.pyc    sharedctypes.py   synchronize.pyc
dummy           heap.py      __init__.pyc  pool.py       process.pyc
reduction.py  sharedctypes.pyc  util.py
liao@liao:/usr/lib/python2.7/multiprocessing$ ls dummy/
connection.py  connection.pyc  __init__.py  __init__.pyc
liao@liao:/usr/lib/python2.7/multiprocessing$
```

multiprocess包下有很多的模块，例如process模块，那么可以在一个示例程序里使用包multiprocess里的process模块。

```
#coding:utf-8
from multiprocessing import Process
import os
def test(name):
    print "Process ID:  %s" % (os.getpid())
    print "Parent Process ID:  %s" % (os.getppid())
if __name__ == "__main__":
    proc = Process(target=test, args=('nmask',))
    proc.start()
    proc.join()
```

需要解释的是`from multiprocessing import Process`是从包multiprocess里引入Process，但Process类定义在process.py文件里，包含Process类的process.py文件是在multiprocessing目录下的，故是multiprocessing包里的一个模块。通过Python交互环境可以查明这一点。

```
>>> from multiprocessing import Process
>>> help(Process)
Help on class Process in module multiprocessing.process:

class Process(__builtin__.object)
```

代码`from multiprocessing import Process`也可以这样去写`from multiprocessing.process import Process`这样写既写了包名又写了模块名即包.模块，其实在Python里一般还是直接用包名(偷懒)，而少有既写包又写模块的。

14.File（文件)方法 python3

open() 方法

Python open() 方法用于打开一个文件，并返回文件对象，在对文件进行处理过程都需要使用到这个函数，如果该文件无法被打开，会抛出 OSError。

注意：使用 open() 方法一定要保证关闭文件对象，即调用 close() 方法。

open() 函数常用形式是接收两个参数：文件名(file)和模式(mode)。

```
open(file, mode='r')
```

完整的语法格式为：

```
open(file, mode='r', buffering=-1, encoding=None, errors=None,
newline=None, closefd=True, opener=None)
```

参数说明：

- file: 必需，文件路径（相对或者绝对路径）。
- mode: 可选，文件打开模式
- buffering: 设置缓冲
- encoding: 一般使用utf8
- errors: 报错级别
- newline: 区分换行符
- closefd: 传入的file参数类型
- opener: 设置自定义开启器，开启器的返回值必须是一个打开的文件描述符。

15.异常处理的定义

python解释器检测到错误，触发异常（也允许程序员自己触发异常）

程序员编写特定的代码，专门用来捕捉这个异常（这段代码与程序逻辑无关，与异常处理有关）

如果捕捉成功则进入另外一个处理分支，执行你为其定制的逻辑，使程序不会崩溃，这就是异常处理

16.异常处理的意义

python解析器去执行程序，检测到了一个错误时，触发异常，异常触发后且没被处理的情况下，程序就在当前异常处终止，后面的代码不会运行，所以你必须提供一种异常处理机制来增强你程序的健壮性与容错性

17.常见的异常

```
AttributeError 试图访问一个对象没有的属性，比如foo.x，但是foo没有属性x
IOError 输入/输出异常；基本上是无法打开文件
ImportError 无法引入模块或包；基本上是路径问题或名称错误
IndentationError 语法错误（的子类） ；代码没有正确对齐
IndexError 下标索引超出序列边界，比如当x只有三个元素，却试图访问x[5]
KeyError 试图访问字典里不存在的键
KeyboardInterrupt Ctrl+C被按下
NameError 尝试访问一个没有申明的变量
SyntaxError Python代码非法，代码不能编译(个人认为这是语法错误，写错了）
TypeError 传入对象类型与要求的不符合
UnboundLocalError 试图访问一个还未被设置的局部变量，基本上是由于另有一个同名的全局变量，导致你以为正在访问它
ValueError 传入一个调用者不期望的值，即使值的类型是正确的
```

18.如何进行异常处理

- 使用if判断式


```
num1=input('>>: ') #输入一个字符串试试
if num1.isdigit():
    int(num1) #我们的正统程序放到了这里,其余的都属于异常处理范畴
elif num1.isspace():
    print('输入的是空格,就执行我这里的逻辑')
elif len(num1) == 0:
    print('输入的是空,就执行我这里的逻辑')
else:
    print('其他情情况,执行我这里的逻辑')

#第二段代码
# num2=input('>>: ') #输入一个字符串试试
# int(num2)

#第三段代码
# num3=input('>>: ') #输入一个字符串试试
# int(num3)
```

问题一：
使用if的方式我们只为第一段代码加上了异常处理，针对第二段代码，你得重新写一堆if，elif等
而这些if，跟你的代码逻辑并无关系,可读性差

问题二：
第一段代码和第二段代码实际上是同一种异常，都是ValueError,相同的错误按理说只处理一次就可以了，而用if，由于这二者if的条件不同，这只能逼着你重新写一个新的if来处理第二段代码的异常
第三段也一样

- try...except

语法：

```
try:
<语句>          #运行别的代码
except <异常类型>:
<语句>          #如果在try部份引发了'name'异常
except <异常类型> as <数据>:
<语句>          #如果引发了'name'异常，获得附加的数据
else:
<语句>          #如果没有异常发生
```

注：

python2 和 3 处理 except 子句的语法有点不同，需要注意；

Python2

```
try:
    print (1/0)
except ZeroDivisionError, err:           # , 加原因参数名称
    print ('Exception: ', err)
```

Python3

```
try:
    print (1/0)
except ZeroDivisionError as err:         # as 加原因参数名称
    print ('Exception: ', err)
```

例

```
try:
    fh = open("testfile", "w")
    fh.write("这是一个测试文件，用于测试异常!!")
except IOError:
    print("Error: 没有找到文件或读取文件失败")
else:
    print("内容写入文件成功")
    fh.close()
```

输出

```
内容写入文件成功
```

注:

异常类只能用来处理指定的异常情况，如果非指定异常则无法处理。（异常是由程序的错误引起的，语法上的错误跟异常处理无关，必须在程序运行前就修正）

```
# 未捕获到异常，程序直接报错

s1 = 'hello'
try:
    int(s1)
except IndexError as e:
    print e
```

输出

```
  File "/Users/hexin/PycharmProjects/py3/day9/1.py", line 11
    print e
          ^
SyntaxError: Missing parentheses in call to 'print'
```

- 多分支

```
try:
    msg=input('>>:')
    int(msg) #ValueError
    #
    # print(x) #NameError
    # #
    # # l=[1,2]
    # # l[10] #IndexError
    #
    # 1+'asdfsadfasdf' #TypeError
```

```
except ValueError as e:
    print(e)
except NameError:
    print('NameError')
except KeyError as e:
    print(e)
```

```
>>:gg
invalid literal for int() with base 10: 'gg'
```

- 万能异常

在python的异常中，有一个万能异常：Exception，他可以捕获任意异常

```
s1 = 'hello'
try:
    int(s1)
except Exception as e:
    '丢弃或者执行其他逻辑'
    print(e)
```

输出

```
invalid literal for int() with base 10: 'hello'
```


- try-finally 语句

try-finally 语句无论是否发生异常都将执行最后的代码。

```
s1 = 'hello'
try:
    int(s1)
except IndexError as e:
    print(e)
except KeyError as e:
    print(e)
except ValueError as e:
    print(e)
#except Exception as e:
#    print(e)
else:
```

```
    print('try内代码块没有异常则执行我')
finally:
    print('无论异常与否,都会执行该模块,通常是进行清理工作')
```

输出

```
invalid literal for int() with base 10: 'hello'
无论异常与否,都会执行该模块,通常是进行清理工作
```

- raise主动触发异常

我们可以使用raise语句自己触发异常

raise语法格式如下:

```
raise [Exception [, args [, traceback]]]
```

语句中Exception是异常的类型（例如，NameError）参数是一个异常参数值。该参数是可选的，如果不提供，异常的参数是"None"。

最后一个参数是可选的（在实践中很少使用），如果存在，是跟踪异常对象。

```
try:
    raise TypeError('类型错误')
except Exception as e:
    print(e)
```

输出

```
类型错误
```

- 自定义异常

```
class hexinException(BaseException):
    def __init__(self,msg):
        self.msg=msg
    def __str__(self):
        return self.msg

try:
    raise hexinException('类型错误')
except hexinException as e:
    print(e)
```

输出

```
类型错误
```

参考链接

https://www.cnblogs.com/puti306/p/12080526.html

https://www.cnblogs.com/jiazeng/articles/11299438.html

https://www.runoob.com/python/python-functions.html

https://www.runoob.com/python/python-files-io.html

https://www.cnblogs.com/zhangyingai/p/7097920.html

Python数据库编程

1.什么是MySQLdb?

MySQLdb 是用于Python链接Mysql数据库的接口，它实现了 Python 数据库 API 规范 V2.0，基于 MySQL C API 上建立的。

2.如何连接数据库?

```
#!/usr/bin/python
# -*- coding: UTF-8 -*-

import MySQLdb

# 打开数据库连接
db = MySQLdb.connect("localhost", "testuser", "test123", "TESTDB",
charset='utf8' )

# 使用cursor()方法获取操作游标
cursor = db.cursor()

# 使用execute方法执行SQL语句
```

```
cursor.execute("SELECT VERSION()")

# 使用 fetchone() 方法获取一条数据
data = cursor.fetchone()

print "Database version : %s " % data

# 关闭数据库连接
db.close()
```

3.如何创建数据库表?

```
#!/usr/bin/python
# -*- coding: UTF-8 -*-

import MySQLdb

# 打开数据库连接
db = MySQLdb.connect("localhost", "testuser", "test123", "TESTDB",
charset='utf8' )

# 使用cursor()方法获取操作游标
cursor = db.cursor()

# 如果数据表已经存在使用 execute() 方法删除表。
cursor.execute("DROP TABLE IF EXISTS EMPLOYEE")

# 创建数据表SQL语句
sql = """CREATE TABLE EMPLOYEE (
         FIRST_NAME  CHAR(20) NOT NULL,
         LAST_NAME  CHAR(20),
         AGE INT,
         SEX CHAR(1),
         INCOME FLOAT )"""

cursor.execute(sql)

# 关闭数据库连接
db.close()
```


4.如何执行数据插入?

```
#!/usr/bin/python
# -*- coding: UTF-8 -*-

import MySQLdb

# 打开数据库连接
```

```
db = MySQLdb.connect("localhost", "testuser", "test123", "TESTDB",
charset='utf8' )

# 使用cursor()方法获取操作游标
cursor = db.cursor()

# SQL 插入语句
sql = """INSERT INTO EMPLOYEE(FIRST_NAME,
         LAST_NAME, AGE, SEX, INCOME)
         VALUES ('Mac', 'Mohan', 20, 'M', 2000)"""
try:
   # 执行sql语句
   cursor.execute(sql)
   # 提交到数据库执行
   db.commit()
except:
   # Rollback in case there is any error
   db.rollback()

# 关闭数据库连接
db.close()
```

5.如何执行数据库查询操作?

Python查询Mysql使用 fetchone() 方法获取单条数据, 使用fetchall() 方法获取多条数据。

fetchone(): 该方法获取下一个查询结果集。结果集是一个对象
fetchall():接收全部的返回结果行.
rowcount: 这是一个只读属性，并返回执行execute()方法后影响的行数。

6.如何更新数据库数据?

```
#!/usr/bin/python
# -*- coding: UTF-8 -*-

import MySQLdb

# 打开数据库连接
db = MySQLdb.connect("localhost", "testuser", "test123", "TESTDB",
charset='utf8' )

# 使用cursor()方法获取操作游标
cursor = db.cursor()

# SQL 更新语句
sql = "UPDATE EMPLOYEE SET AGE = AGE + 1 WHERE SEX = '%c'" % ('M')
try:
   # 执行SQL语句
   cursor.execute(sql)
```

```
    # 提交到数据库执行
    db.commit()
except:
    # 发生错误时回滚
    db.rollback()

# 关闭数据库连接
db.close()
```

7.如何删除数据库数据?

```
#!/usr/bin/python
# -*- coding: UTF-8 -*-

import MySQLdb

# 打开数据库连接
db = MySQLdb.connect("localhost", "testuser", "test123", "TESTDB",
charset='utf8' )

# 使用cursor()方法获取操作游标
cursor = db.cursor()

# SQL 删除语句
sql = "DELETE FROM EMPLOYEE WHERE AGE > %s" % (20)
try:
    # 执行SQL语句
    cursor.execute(sql)
    # 提交修改
    db.commit()
except:
    # 发生错误时回滚
    db.rollback()

# 关闭连接
db.close()
```


8.如何使用数据库事务?

```
# SQL删除记录语句
sql = "DELETE FROM EMPLOYEE WHERE AGE > %s" % (20)
try:
   # 执行SQL语句
   cursor.execute(sql)
   # 向数据库提交
   db.commit()
except:
   # 发生错误时回滚
   db.rollback()
```

9.python如何操作redis?

```
连接
- 直接连接：
    import redis
    r = redis.Redis(host='10.211.55.4', port=6379)
    r.set('foo', 'Bar')       # 这里的方法与Redis命令类似
    print r.get('foo')
- 连接池：
    import redis
    pool = redis.ConnectionPool(host='10.211.55.4', port=6379)
    r = redis.Redis(connection_pool=pool)
    r.set('foo', 'Bar')
    print r.get('foo')
```


10.如果redis中的某个列表中的数据量非常大，如何实现循环显示每一个值?

```
        def list_iter(key, count=3):
            start = 0
            while True:
                result = conn.lrange(key, start, start+count-1)
                start += count
                if not result:
                    break
                for item in result:
                    yield item
        # 调用
         for val in list_iter('num_list'):
            print(val)
```

11.什么是一致性哈希？Python中是否有相应模块？

一致性hash算法（DHT）可以通过减少影响范围的方式，解决增减服务器导致的数据散列问题，从而解决了分布式环境下负载均衡问题；
如果存在热点数据，可以通过增添节点的方式，对热点区间进行划分，将压力分配至其他服务器，重新达到负载均衡的状态。

Python模块--hash_ring，即Python中的一致性hash。

参考资料

https://www.runoob.com/python/python-mysql.html

https://www.cnblogs.com/wcwnina/p/10304641.html

PythonWeb开发

1.什么是Flask？有什么优点？

Flask是一个Web框架，就是提供一个工具，库和技术来允许你构建一个Web应用程序。这个Web应用程序可以是一些Web页面，博客，wiki，基于Web的日里应用或商业网站。

Flask属于微框架（micro-framework)这一类别，微架构通常是很小的不依赖外部库的框架。其优点如下：

- 框架很轻量
- 更新时依赖小
- 专注于安全方面的bug

2.Django和Flask有什么区别？

Flask
轻量级web框架，默认依赖两个外部库：jinja2和Werkzeug WSGI工具
适用于做小型网站以及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

Django
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(Object-Relational Mapping 对象关系映射)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

3.Flask-WTF是什么，有什么特点?

Flask-wtf是一个用于表单处理、校验并提供CSRF验证的功能的扩展库
Flask-wtf能把正表单免受CSRF<跨站请求伪造>的攻击

4.Flask脚本的常用方式是什么?

在shell中运行脚本文件
在python编译器中run

5.如何在Flask中访问会话?

会话（seesion）会话数据存储在服务器上。 会话是客户端登录到服务器并注销的时间间隔。需要在此会话中进行的数据存储在服务器上的临时目录中。
from flask import session导入会话对象
session['name'] = 'admin'给会话添加变量
session.pop('username', None)删除会话的变量

6.解释Python Flask中的数据库连接?

python中的数据库连接有2种方式
1）在脚本中以用第三方库正常连接，用sql语句正常操作数据库，如mysql关系型数据库的pymsql库
2）用ORM来进行数据库连接，flask中典型的flask_sqlalchemy，已面向对象的方式进行数据库的连接与操作

7.Flask框架有哪些依赖组件?

Route(路由)
templates(模板)
Models(orm模型)
blueprint(蓝图)
Jinja2模板引擎

8.Flask蓝图的作用?

蓝图Blueprint实现模块化的应用

- book_bp = Blueprint('book', **name**）创建蓝图对象
- 蓝图中使用路由@book_bp.route('url')
- 在另一.py文件里导入和注册蓝图from book import book_bp
  app.register_blueprint(book_bp)

作用
将不同的功能模块化
构建大型应用
优化项目结构
增强可读性,易于维护（跟Django的view功能相似）

9.列举使用过的Flask第三方组件?

flask_bootstrap
flask-WTF
flask_sqlalchemy

10. 简述Flask上下文管理流程?

每次有请求过来的时候，flask 会先创建当前线程或者进程需要处理的两个重要上下文对象，把它们保存到隔离的栈里面，这样视图函数进行处理的时候就能直接从栈上获取这些信息。

11.Flask框架默认session处理机制?

Flask的默认session利用了Werkzeug的SecureCookie，把信息做序列化(pickle)后编码(base64)，放到cookie里了。

过期时间是通过cookie的过期时间实现的。

为了防止cookie内容被篡改，session会自动打上一个叫session的hash串，这个串是经过session内容、SECRET_KEY计算出来的，看得出，这种设计虽然不能保证session里的内容不泄露，但至少防止了不被篡改。

12.django请求的生命周期?

1.wsgi,请求封装后交给web框架
2.中间件，对请求进行校验或者在请求对象中添加其他相关数据，
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
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

13.列举django中间件的5个方法？以及django中间件的应用场景?

1.process_request
接收到客户端信息后立即执行，视图函数之前
2.process_response
返回到客户端信息前最后执行，视图函数之后
3.process_view
拿到视图函数的名称，参数，执行process_view()方法
4.process_exception
视图函数出错时执行
5.process_template_response
在视图函数执行完后立即执行，前提是视图返回的对象中有一个render()方法

14.django rest framework框架中都有那些组件?

1.序列化组件:serializers 对queryset序列化以及对请求数据格式校验
2.路由组件routers 进行路由分发
3.视图组件ModelViewSet 帮助开发者提供了一些类，并在类中提供了多个方法
4.认证组件 写一个类并注册到认证类(authentication_classes)，在类的的authticate方法中编写认证逻
5.权限组件 写一个类并注册到权限类(permission_classes)，在类的的has_permission方法中编写认证逻辑。
6.频率限制 写一个类并注册到频率类(throttle_classes)，在类的的allow_request/wait 方法中编写认证逻辑
7.解析器 选择对数据解析的类，在解析器类中注册(parser_classes)
8.渲染器 定义数据如何渲染到到页面上,在渲染器类中注册(renderer_classes)
9.分页 对获取到的数据进行分页处理, pagination_class
10.版本 版本控制用来在不同的客户端使用不同的行为
在url中设置version参数，用户请求时候传入参数。在request.version中获取版本，根据版本不同 做不同处理

15.django rest framework如何实现的用户访问频率控制?

from rest_framework.throttling import SimpleRateThrottle

这里使用的节流类是继承了SimplePateThrottle类，而这个类利用了django内置的缓存来存储访问记录。通过全局节流设置，所有的视图类默认是使用UserThrottle类进行节流，如果不想使用默认的类就自定义给throttle_classes属性变量赋值，如：“throttle_classes = [VisitThrottle,]”。

16.django中如何实现单元测试?

django的单元测试使用python的unittest模块，这个模块使用基于类的方法来定义测试。

17.django-debug-toolbar的作用?

用来调试请求的接口

18.什么是wsgi?

WSGI是Python在处理HTTP请求时，规定的一种处理方式。如一个HTTP Request过来了，那么就有一个相应的处理函数来进行处理和返回结果。WSGI就是规定这个处理函数的参数长啥样的，它的返回结果是长啥样的？至于该处理函数的名子和处理逻辑是啥样的，那无所谓。简单而言，WSGI就是规定了处理函数的输入和输出格式。

19.简述什么是FBV和CBV?

FBV和CBV本质是一样的，基于函数的视图叫做FBV，基于类的视图叫做CBV。
在python中使用CBV的优点：
提高了代码的复用性，可以使用面向对象的技术，比如Mixin（多继承）。
可以用不同的函数针对不同的HTTP方法处理，而不是通过很多if判断，提高代码可读性。

20.django中csrf的实现机制

第一步：django第一次响应来自某个客户端的请求时,后端随机产生一个token值，把这个token保存在SESSION状态中;同时,后端把这个token放到cookie中交给前端页面;
第二步：下次前端需要发起请求（比如发帖）的时候把这个token值加入到请求数据或者头信息中,一起传给后端；Cookies:{csrftoken:xxxxx}
第三步：后端校验前端请求带过来的token和SESSION里的token是否一致。

21.Django本身提供了runserver，为什么不能用来部署？(runserver与uWSGI的区别)

1）runserver方法是调试 Django 时经常用到的运行方式，它使用Django自带的WSGI Server 运行，主要在测试和开发中使用，并且 runserver 开启的方式也是单进程 。
2）uWSGI是一个Web服务器，它实现了WSGI协议、uwsgi、http 等协议。注意uwsgi是一种通信协议，而uWSGI是实现uwsgi协议和WSGI协议的 Web 服务器。uWSGI具有超快的性能、低内存占用和多app管理等优点，并且搭配着Nginx就是一个生产环境了，能够将用户访问请求与应用 app 隔离开，实现真正的部署 。相比来讲，支持的并发量更高，方便管理多进程，发挥多核的优势，提升性能。

22.Django如何实现websocket?

django实现websocket官方推荐大家使用channels。channels通过升级http协议来升级到websocket协议。保证实时通讯。也就是说，我们完全可以用channels实现我们的即时通讯。而不是使用长轮询和计时器方式来保证伪实时通讯。他通过改造django框架，使django既支持http协议又支持websocket协议。

参考资料

https://blog.csdn.net/wl_python/article/details/81131873

https://www.cnblogs.com/bk770466199/p/12696103.html

https://www.jianshu.com/p/724233387ba3

Python爬虫

1.scrapy框架有哪几个组件/模块?

Scrapy Engine: 这是引擎，负责Spiders、ItemPipeline、Downloader、Scheduler中间的通讯，信号、数据传递等等！（像不像人的身体？）

Scheduler(调度器): 它负责接受引擎发送过来的requests请求，并按照一定的方式进行整理排列，入队、并等待Scrapy Engine(引擎)来请求时，交给引擎。

Downloader（下载器）：负责下载Scrapy Engine(引擎)发送的所有Requests请求，并将其获取到的Responses交还给Scrapy Engine(引擎)，由引擎交给Spiders来处理，

Spiders：它负责处理所有Responses,从中分析提取数据，获取Item字段需要的数据，并将需要跟进的URL提交给引擎，再次进入Scheduler(调度器)，

Item Pipeline：它负责处理Spiders中获取到的Item，并进行处理，比如去重，持久化存储（存数据库，写入文件，总之就是保存数据用的）

Downloader Middlewares（下载中间件）：你可以当作是一个可以自定义扩展下载功能的组件

2.简单说一下scrapy工作流程。

数据在整个Scrapy的流向：

程序运行的时候，

引擎：Hi！Spider, 你要处理哪一个网站?

Spiders：我要处理23wx.com

引擎：你把第一个需要的处理的URL给我吧。

Spiders：给你第一个URL是XXXXXXX.com

引擎：Hi！调度器，我这有request你帮我排序入队一下。

调度器：好的，正在处理你等一下。

引擎：Hi！调度器，把你处理好的request给我，

调度器：给你，这是我处理好的request

引擎：Hi！下载器，你按照下载中间件的设置帮我下载一下这个request

下载器：好的！给你，这是下载好的东西。（如果失败：不好意思，这个request下载失败，然后引擎告诉调度器，这个request下载失败了，你记录一下，我们待会儿再下载。）

引擎：Hi！Spiders，这是下载好的东西，并且已经按照Spider中间件处理过了，你处理一下（注意！这儿responses默认是交给def parse这个函数处理的）

Spiders：（处理完毕数据之后对于需要跟进的URL），Hi！引擎，这是我需要跟进的URL，将它的responses交给函数 def xxxx(self, responses)处理。还有这是我获取到的Item。

引擎：Hi！ Item Pipeline 我这儿有个item你帮我处理一下！调度器！这是我需要的URL你帮我处理下。然后从第四步开始循环，直到获取到你需要的信息，

注意！只有当调度器中不存在任何request了，整个程序才会停止，（也就是说，对于下载失败的ＵＲＬ，Scrapy会重新下载。）

3.scrapy指纹去重原理和scrappy-redis的去重原理?

scrapy的去重原理流程：利用hash值和集合去重。首先创建fingerprint = set()结合，然后将request对象利用sha1对象进行信息摘要，摘要完成之后， 判断hash值是否在集合中，如果在，返回true,如果不在，就add到集合。

scrapy-redis 的去重原理基本是一样的，只不过持久化存储到redis共享数据库中，当请求数据达到10亿级以上，这个时候就会非常消耗内存，一个sha1 40个字节，就会占40G的内存，这个存储绝大部分的数据库无法承受，这个时候就要使用布隆过滤器。

4.请简要介绍下scrapy框架。

scrapy 是一个快速(fast)、高层次(high-level)的基于 python 的 web 爬虫构架，用于抓取web站点并从页面中提取结构化的数据。scrapy 使用了 Twisted异步网络库来处理网络通讯。

5.为什么要使用scrapy框架？scrapy框架有哪些优点?

它更容易构建大规模的抓取项目
它异步处理请求，速度非常快
它可以使用自动调节机制自动调整爬行速度

6.scrapy如何实现分布式抓取?

可以借助scrapy_redis类库来实现。

在分布式爬取时，会有master机器和slave机器，其中，master为核心服务器，slave为具体的爬虫服务器。

我们在master服务器上搭建一个redis数据库，并将要抓取的url存放到redis数据库中，所有的slave爬虫服务器在抓取的时候从redis数据库中去链接，由于scrapy_redis自身的队列机制，slave获取的url不会相互冲突，然后抓取的结果最后都存储到数据库中。master的redis数据库中还会将抓取过的url的指纹存储起来，用来去重。相关代码在dupefilter.py文件中的request_seen()方法中可以找到。

去重问题：
dupefilter.py 里面的源码：
def request_seen(self, request):
fp = request_fingerprint(request)
added = self.server.sadd(self.key, fp)

return not added
去重是把 request 的 fingerprint 存在 redis 上，来实现的。

7.scrapy和requests的使用情况?

requests 是 polling 方式的，会被网络阻塞，不适合爬取大量数据

scapy 底层是异步框架 twisted ，并发是最大优势

8.爬虫使用多线程好？还是多进程好？为什么?

对于IO密集型代码（文件处理，网络爬虫），多线程能够有效提升效率（单线程下有IO操作会进行IO等待，会造成不必要的时间等待，而开启多线程后，A线程等待时，会自动切换到线程B，可以不浪费CPU的资源，从而提升程序执行效率）。

在实际的采集过程中，既考虑网速和相应的问题，也需要考虑自身机器硬件的情况，来设置多进程或者多线程。

9.了解哪些基于爬虫相关的模块?

网络请求：urllib，requests，aiohttp
数据解析：re，xpath，bs4，pyquery
selenium
js逆向：pyexcJs

10.列举在爬虫过程中遇到的哪些比较难的反爬机制?

动态加载的数据
动态变化的请求参数
js加密
代理
cookie

11.简述如何抓取动态加载数据?

基于抓包工具进行全局搜索
如果动态加载的数据是密文，则全局搜索是搜索不到

12.移动端数据如何抓取?

fiddler，appnium，网络配置

13.如何实现全站数据爬取?

基于手动请求发送+递归解析
基于CrwalSpider（LinkExtractor，Rule）

14.如何提升爬取数据的效率?

使用框架
线程池，多任务的异步协程
分布式

15.列举你接触的反爬机制?

从功能上来讲，爬虫一般分为数据采集，处理，储存三个部分。这里我们只讨论数据采集部分。

一般网站从三个方面反爬虫：用户请求的Headers，用户行为，网站目录和数据加载方式。前两种比较容易遇到，大多数网站都从这些角度来反爬虫。第三种一些应用ajax的网站会采用，这样增大了爬取的难度。

1）通过Headers反爬虫

从用户请求的Headers反爬虫是最常见的反爬虫策略。很多网站都会对Headers的User-Agent进行检测，还有一部分网站会对Referer进行检测（一些资源网站的防盗链就是检测Referer）。如果遇到了这类反爬虫机制，可以直接在爬虫中添加Headers，将浏览器的User-Agent复制到爬虫的Headers中；或者将Referer值修改为目标网站域名。对于检测Headers的反爬虫，在爬虫中修改或者添加Headers就能很好的绕过。

2）基于用户行为反爬虫

还有一部分网站是通过检测用户行为，例如同一IP短时间内多次访问同一页面，或者同一账户短时间内多次进行相同操作。

大多数网站都是前一种情况，对于这种情况，使用IP代理就可以解决。可以专门写一个爬虫，爬取网上公开的代理ip，检测后全部保存起来。这样的代理ip爬虫经常会用到，最好自己准备一个。有了大量代理ip后可以每请求几次更换一个ip，这在requests或者urllib2中很容易做到，这样就能很容易的绕过第一种反爬虫。

对于第二种情况，可以在每次请求后随机间隔几秒再进行下一次请求。有些有逻辑漏洞的网站，可以通过请求几次，退出登录，重新登录，继续请求来绕过同一账号短时间内不能多次进行相同请求的限制。

3）动态页面的反爬虫

上述的几种情况大多都是出现在静态页面，还有一部分网站，我们需要爬取的数据是通过ajax请求得到，或者通过JavaScript生成的。首先用Firebug或者HttpFox对网络请求进行分析。如果能够找到ajax请求，也能分析出具体的参数和响应的具体含义，我们就能采用上面的方法，直接利用requests或者urllib2模拟ajax请求，对响应的json进行分析得到需要的数据。

能够直接模拟ajax请求获取数据固然是极好的，但是有些网站把ajax请求的所有参数全部加密了。我们根本没办法构造自己所需要的数据的请求。我这几天爬的那个网站就是这样，除了加密ajax参数，它还把一些基本的功能都封装了，全部都是在调用自己的接口，而接口参数都是加密的。遇到这样的网站，我们就不能用上面的方法了，我用的是selenium+phantomJS框架，调用浏览器内核，并利用phantomJS执行js来模拟人为操作以及触发页面中的js脚本。从

填写表单到点击按钮再到滚动页面，全部都可以模拟，不考虑具体的请求和响应过程，只是完完整整的把人浏览页面获取数据的过程模拟一遍。

用这套框架几乎能绕过大多数的反爬虫，因为它不是在伪装成浏览器来获取数据（上述的通过添加 Headers一定程度上就是为了伪装成浏览器），它本身就是浏览器，phantomJS就是一个没有界面的浏览器，只是操控这个浏览器的不是人。利用 selenium+phantomJS能干很多事情，例如识别点触式（12306）或者滑动式的验证码，对页面表单进行暴力破解等等。它在自动化渗透中还 会大展身手，以后还会提到这个。

16.什么是深度优先和广度优先（优劣）

默认情况下scrapy是深度优先。
深度优先：占用空间大，但是运行速度快。
广度优先：占用空间少，运行速度慢

17.是否了解谷歌的无头浏览器?

无头浏览器即headless browser，是一种没有界面的浏览器。既然是浏览器那么浏览器该有的东西它都应该有，只是看不到界面而已。

Python中selenium模块中的PhantomJS即为无界面浏览器（无头浏览器）:是基于QtWebkit的无头浏览器。

18.说下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。

19.需要登录的网页，如何解决同时限制ip，cookie，session?

解决限制IP可以使用代理IP地址池、服务器；不适用动态爬取的情况下可以使用反编译JS文件获取相应的文件，或者换用其它平台（比如手机端）看看是否可以获取相应的json文件。

20.验证码的解决?

1.输入式验证码
解决思路：这种是最简单的一种，只要识别出里面的内容，然后填入到输入框中即可。这种识别技术叫OCR，这里我们推荐使用Python的第三方库，tesserocr。对于没有什么背影影响的验证码，直接通过这个库来识别就可以。但是对于有嘈杂的背景的验证码这种，直接识别识别率会很低，遇到这种我们就得需要先处理一下图片，先对图片进行灰度化，然后再进行二值化，再去识别，这样识别率会大大提高。

验证码识别大概步骤:
转化成灰度图
去背景噪声
图片分割

2.滑动式验证码
解决思路：对于这种验证码就比较复杂一点，但也是有相应的办法。我们直接想到的就是模拟人去拖动验证码的行为，点击按钮，然后看到了缺口 的位置，最后把拼图拖到缺口位置处完成验证。

第一步：点击按钮。然后我们发现，在你没有点击按钮的时候那个缺口和拼图是没有出现的，点击后才出现，这为我们找到缺口的位置提供了灵感。

第二步：拖到缺口位置。我们知道拼图应该拖到缺口处，但是这个距离如果用数值来表示？通过我们第一步观察到的现象，我们可以找到缺口的位置。这里我们可以比较两张图的像素，设置一个基准值，如果某个位置的差值超过了基准值，那我们就找到了这两张图片不一样的位置，当然我们是从那块拼图的右侧开始并且从左到右，找到第一个不一样的位置时就结束，这是的位置应该是缺口的left，所以我们使用selenium拖到这个位置即可。这里还有个疑问就是如何能自动的保存这两张图？这里我们可以先找到这个标签，然后获取它的location和size，然后 top，bottom，left，right = location['y'] ,location['y']+size['height']+ location['x'] + size['width']，然后截图，最后抠图填入这四个位置就行。具体的使用可以查看selenium文档，点击按钮前抠张图，点击后再抠张图。最后拖动的时候要需要模拟人的行为，先加速然后减速。因为这种验证码有行为特征检测，人是不可能做到一直匀速的，否则它就判定为是机器在拖动，这样就无法通过验证了。

3.点击式的图文验证 和 图标选择

图文验证：通过文字提醒用户点击图中相同字的位置进行验证。

图标选择： 给出一组图片，按要求点击其中一张或者多张。借用万物识别的难度阻挡机器。

这两种原理相似，只不过是一个是给出文字，点击图片中的文字，一个是给出图片，点出内容相同的图片。

这两种没有特别好的方法，只能借助第三方识别接口来识别出相同的内容，推荐一个超级鹰，把验证码发过去，会返回相应的点击坐标。

然后再使用selenium模拟点击即可。具体怎么获取图片和上面方法一样。

21.滑动验证码如何破解?

破解核心思路:
1、如何确定滑块滑动的距离?
滑块滑动的距离，需要检测验证码图片的缺口位置
滑动距离 = 终点坐标 - 起点坐标
然后问题转化为我们需要屏幕截图，根据selenium中的position方法并进行一些坐标计算，获取我们需要的位置

2、坐标我们如何获取?
起点坐标:
每次运行程序，位置固定不变，滑块左边界离验证码图片左边界有6px的距离
终点坐标:
每次运行程序，位置会变，我们需要计算每次缺口的位置
怎么计算终点也就是缺口的位置?
先举个例子，比如我下面两个图片都是120x60的图片，一个是纯色的图片，一个是有一个蓝色线条的图片（蓝色线条位置我事先设定的是60px位置），我现在让你通过程序确定蓝色线条的位置，你怎么确定?

答案:
遍历所有像素点色值，找出色值不一样的点的位置来确定蓝色线条的位置
这句话该怎么理解?大家点开我下面的图片，是不是发现图片都是由一个一个像素点组成的，120×60的图片，对应的像素就是横轴有120个像素点，纵轴有60个像素点，我们需要遍历两个图片的坐标并对比色值，从（0,0）（0,1）.....一直到(120,60），开始对比两个图片的色值，遇到色值不一样的，我们return返回该位置即可

22.爬下来的数据是怎么存储?

以json格式存储到文本文件
这是最简单，最方便，最使用的存储方式，json格式保证你在打开文件时，可以直观的检查所存储的数据，一条数据存储一行，这种方式适用于爬取数据量比较小的情况，后续的读取分析也是很方便的。

存储到excel
如果爬取的数据很容易被整理成表格的形式，那么存储到excel是一个比较不错的选择，打开excel后，对数据的观察更加方便，excel也可以做一些简单的操作，写excel可以使用xlwt这个库，读取excel可以使用xlrd，同方法1一样，存储到excel里的数据不宜过多，此外，如果你是多线程爬取，不可能用多线程去写excel，这是一个限制。

存储到sqlite
sqlite无需安装，是零配置数据库，这一点相比于mysql要轻便太多了，语法方面，只要你会mysql，操作sqlite就没有问题。当爬虫数据量很大时，需要持久化存储，而你又懒得安装mysql时，sqlite绝对是最佳选择，不多呢，它不支持多进程读写，因此不适合多进程爬虫。

存储到mysql数据库
mysql可以远程访问，而sqlite不可以，这意味着你可以将数据存储到远程服务器主机上，当数据量非常大时，自然要选择mysql而不是sqlite，但不论是mysql还是sqlite，存储数据前都要先建表，根据要抓取的数据结构和内容，定义字段，这是一个需要耐心和精力的事情。

存储到mongodb
我最喜欢no sql 数据库的一个原因就在于不需要像关系型数据库那样去定义表结构，因为定义表结构很麻烦啊，要确定字段的类型，varchar 类型数据还要定义长度，你定义的小了，数据太长就会截断。
mongodb 以文档方式存储数据，你使用pymongo这个库，可以直接将数据以json格式写入mongodb, 即便是同一个collection，对数据的格式也是没有要求的，实在是太灵活了。
刚刚抓下来的数据，通常需要二次清洗才能使用，如果你用关系型数据库存储数据，第一次就需要定义好表结构，清洗以后，恐怕还需要定义个表结构，将清洗后的数据重新存储，这样过于繁琐，使用mongodb，免去了反复定义表结构的过程。

23.cookie过期的处理问题?

这时候就需要cookie自动的更新了。通常怎样自动更新cookie呢？这里会用到selenium。
步骤1、 采用selenium自动登录获取cookie，保存到文件;
步骤2、 读取cookie，比较cookie的有效期，若过期则再次执行步骤1;
步骤3、 在请求其他网页时，填入cookie，实现登录状态的保持。

24.谈一谈你对Selenium和PhantomJS了解

selenium
Selenium是一个用于Web应用程序测试的工具。Selenium测试直接运行在浏览器中，就像真正的用户在操作一样。支持的浏览器包括IE（7, 8, 9, 10, 11），Mozilla Firefox，Safari，Google Chrome，Opera等主流浏览器。这个工具的主要功能包括：测试与浏览器的兼容性——测试你的应用程序看是否能够很好得工作在不同浏览器和操作系统之上。

它的功能有:
框架底层使用JavaScript模拟真实用户对浏览器进行操作。测试脚本执行时，浏览器自动按照脚本代码做出点击，输入，打开，验证等操作，就像真实用户所做的一样，从终端用户的角度测试应用程序。
使浏览器兼容性测试自动化成为可能，尽管在不同的浏览器上依然有细微的差别。
使用简单，可使用Java，Python等多种语言编写用例脚本
也就是说，它可以根据指令，做出像真实的人在访问浏览器一样的动作，比如打开网页，截图等功能。

phantomjs
(新版本的selenium已经开始弃用phantomjs， 不过有时候我们可以单独用它做一些事情)

是一个基于Webkit的无界面浏览器，可以把网站内容加载到内存中并执行页面上的各种脚本（比如js）。

25.怎么判断网站是否更新?

1、304页面http状态码

当第二次请求页面访问的时候，该页面如果未更新，则会反馈一个304代码，而搜索引擎也会利用这个304http状态码来进行判断页面是否更新。

首先第一次肯定是要爬取网页的，假设是A.html，这个网页存储在磁盘上，相应地有个修改时间（也即是更新这个文件的时间）。

那么第二次爬取的时候，如果发现这个网页本地已经有了，例如A.html，这个时候，你只需要向服务器发送一个If-Modified-Since的请求，把A.html的修改时间带上去。

如果这段时间内，A.html更新了，也就是A.html过期了，服务器就会HTTP状态码200，并且把新的文件发送过来，这时候只要更新A.html即可。

如果这段时间内，A.html的内容没有变，服务器就会返返回HTTP状态码304（不返回文件内容），这个时候就不需要更新文件。

2、Last-Modified文件最后修改时间

这是http头部信息中的一个属性，主要是记录页面最后一次的修改时间，往往我们会发现，一些权重很高的网站，及时页面内容不更新，但是快照却还是能够每日更新，这其中就有Last-Modified的作用。通产情况下，下载网页我们使用HTTP协议，向服务器发送HEAD请求，可以得到页面的最后修改时间LastModifed,或者标签ETag。将这两个变量和上次下载记录的值的比较就可以知道一个网页是否跟新。这个策略对于静态网页是有效的。是对于绝大多数动态网页如ASP，JSP来说，LastModifed就是服务器发送Response的时间，并非网页的最后跟新时间，而Etag通常为空值。所以对于动态网页使用LastModifed和Etag来判断是不合适的，因此Last-Modified只是蜘蛛判断页面是否更新的一个参考值，而不是条件。

参考资料

https://blog.csdn.net/weixin_45387317/article/details/101375974

https://www.cnblogs.com/tianyiliang/p/10219034.html

https://www.sohu.com/a/340282079_120123190

https://www.cnblogs.com/linglichong/p/12425560.html

http://blog.itpub.net/69923331/viewspace-2654286/

https://www.cnblogs.com/cherish-cxh/p/12778718.html