# python-面试通关宝典

秋招、春招的小伙伴，有面Python开发方向的，看这一个repo就够啦😘 （ps:也欢迎大家贡献面试中遇到的Python相关问题，不断丰富这个repo以帮助到更多的人✿✿ヽ(°▽°)ノ✿）

# Table of Contents

* [python-面试通关宝典](#python-面试通关宝典)
	+ [一.语言特性](#一语言特性)
		- [1.谈谈对 Python 和其他语言的区别？](#1谈谈对-python-和其他语言的区别)
		- [2.简述解释型和编译型编程语言？](#2简述解释型和编译型编程语言)
		- [3.Python 的解释器种类以及相关特点？](#3python-的解释器种类以及相关特点)
		- [4.说说你知道的 Python3 和 Python2 之间的区别？](#4说说你知道的-python3-和-python2-之间的区别)
		- [5.Python3 和 Python2 中 int 和 long 的区别？](#5python3-和-python2-中-int-和-long-的区别)
		- [6.xrange 和 range 的区别？](#6xrange-和-range-的区别)
	+ [二.编码规范](#二编码规范)
		- [7.什么是 PEP8？](#7什么是-pep8)
		- [8.了解 Python 之禅吗？](#8了解-python-之禅吗)
		- [9.了解 docstring 吗？](#9了解-docstring-吗)
		- [10.了解类型注解吗？](#10了解类型注解吗)
		- [11.例举你知道 Python 对象的命名规范，例如方法或者类等。](#11例举你知道-python-对象的命名规范例如方法或者类等)
		- [12.Python 中的注释有几种？](#12python-中的注释有几种)
		- [13.如何优雅的给一个函数加注释？](#13如何优雅的给一个函数加注释)
		- [14.如何给变量加注释？](#14如何给变量加注释)
		- [15.Python 代码缩进中是否支持 Tab 键和 空格 混用？](#15python-代码缩进中是否支持-tab-键和-空格-混用)
		- [16.是否可以在一句 import 中导入多个库？](#16是否可以在一句-import-中导入多个库)
		- [17.在给 Python 文件命名的时候需要注意什么？](#17在给-python-文件命名的时候需要注意什么)
		- [18.例举几个规范 Python 代码风格的工具。](#18例举几个规范-python-代码风格的工具)
	+ [三.数据类型](#三数据类型)
		- [字符串](#字符串)
			* [19.列举 Python 中的基本数据类型。](#19列举-python-中的基本数据类型)
			* [20.如何区别 可变数据类型 和 不可变数据类型？](#20如何区别-可变数据类型-和-不可变数据类型)
			* [21.将 "hello world" 转换为首字母大写 "Hello World"。](#21将-hello-world-转换为首字母大写-hello-world)
			* [22.如何检测字符串中只含有数字？](#22如何检测字符串中只含有数字)
			* [23.将字符串 "ilovechina" 进行反转。](#23将字符串-ilovechina-进行反转)
			* [24.Python 中的字符串格式化方式你知道哪些？](#24python-中的字符串格式化方式你知道哪些)
			* [25.有一个字符串开头和末尾都有空格，比如 " adabdw "。要求写一个函数把这个字符串的前后空格都去掉。](#25有一个字符串开头和末尾都有空格比如--adabdw-要求写一个函数把这个字符串的前后空格都去掉)
			* [26.获取字符串 "123456" 最后的两个字符。](#26获取字符串-123456-最后的两个字符)
			* [27.一个编码为 GBK 的字符串 S，要将其转成 UTF-8 编码的字符串，应如何操作？](#27一个编码为-gbk-的字符串-s要将其转成-utf-8-编码的字符串应如何操作)
			* [28.字符串 s = "info:xiaoZhang 33 shandong"，用正则切分字符串输出 ['info', 'xiaoZhang', '33', 'shandong']。](#28字符串-s--infoxiaozhang-33-shandong用正则切分字符串输出-info-xiaozhang-33-shandong)
			* [29.怎样将字符串转换为小写？](#29怎样将字符串转换为小写)
			* [30.单引号、双引号和三引号的区别？](#30单引号双引号和三引号的区别)
			* [31.字符串 a = "你好 中国 "，去除多余空格只留一个空格。](#31字符串-a--你好-----中国--去除多余空格只留一个空格)
		- [列表](#列表)
			* [32.已知 AList = [1,2,3,1,2]，对 AList 列表元素去重，写出具体过程。](#32已知-alist--12312对-alist-列表元素去重写出具体过程)
			* [33.如何将 "1,2,3" 变成 ["1","2","3"]？](#33如何将-123-变成-123)
			* [34.给定两个 list，A 和 B，找出相同元素和不同元素。](#34给定两个-lista-和-b找出相同元素和不同元素)
			* [35.用一行代码展开该列表 [[1,2],[3,4],[5,6]]，得出[1,2,3,4,5,6]。](#35用一行代码展开该列表-123456得出123456)
			* [36.合并列表 [1,5,7,9] 和 [2,2,6,8]。](#36合并列表-1579-和-2268)
			* [37.如何打乱一个列表的元素？](#37如何打乱一个列表的元素)
		- [字典](#字典)
			* [38.字典操作中 del 和 pop 有什么区别？](#38字典操作中-del-和-pop-有什么区别)
			* [39.将如下字典按照年龄排序。](#39将如下字典按照年龄排序)
			* [40.请合并下面两个字典 a = {"A":1,"B":2}，b = {"C":3,"D":4}。](#40请合并下面两个字典-a--a1b2b--c3d4)
			* [41.如何使用生成式的方式生成一个字典，写一段功能代码。](#41如何使用生成式的方式生成一个字典写一段功能代码)
			* [42.如何把 元组 ("a","b") 和 元组(1,2)，变为字典{"a":1,"b":2}？](#42如何把-元组-ab-和-元组12变为字典a1b2)
		- [综合数据类型](#综合数据类型)
			* [43.Python 常用的数据结构的类型及其特性？](#43python-常用的数据结构的类型及其特性)
			* [44.如何将 元组("A","B") 和 元组(1,2) 合并成 字典{"A":1,"B":2}？](#44如何将-元组ab-和-元组12-合并成-字典a1b2)
			* [45.Python 里面如何实现 tuple 和 list 的转换？](#45python-里面如何实现-tuple-和-list-的转换)
			* [46.我们知道对于列表可以使用切片操作进行部分元素的选择，那么如何对生成器类型的对象实现相同的功能呢？](#46我们知道对于列表可以使用切片操作进行部分元素的选择那么如何对生成器类型的对象实现相同的功能呢)
			* [47.请将 [i for i in range(3)] 改成 生成器。](#47请将-i-for-i-in-range3-改成-生成器)
			* [48.将 a="hello" 和 b="你好" 编码成 bytes 类型。](#48将-ahello-和-b你好-编码成-bytes-类型)
			* [49.下面的代码输出结果是什么？](#49下面的代码输出结果是什么)
			* [50.下面的代码输出的结果是什么？](#50下面的代码输出的结果是什么)
		- [操作类题目](#操作类题目)
			* [51.在 Python 中交换两个变量的值。](#51在-python-中交换两个变量的值)
			* [52.在读文件操作的时候会使用 read、readline 或者 readlines，简述它们各自的作用。](#52在读文件操作的时候会使用-readreadline-或者-readlines简述它们各自的作用)
			* [53.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？](#53json-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)
			* [54.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？](#54json-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)
			* [55.有两个磁盘文件 A 和 B，各存放一行字母，要求把这两个文件中的信息合并（按字母顺序排列），输出到一个新文件 C 中。](#55有两个磁盘文件-a-和-b各存放一行字母要求把这两个文件中的信息合并按字母顺序排列输出到一个新文件-c-中)
			* [56.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)。](#56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)
			* [57.写一个函数，接收整数参数 n，返回一个函数。函数的功能是把函数的参数和 n 相乘并把结果返回。](#57写一个函数接收整数参数-n返回一个函数函数的功能是把函数的参数和-n-相乘并把结果返回)
			* [58.下面的代码会存在什么问题，如何改进？](#58下面的代码会存在什么问题如何改进)
			* [59.一行代码输出 1-100 之间的所有偶数。](#59一行代码输出-1-100-之间的所有偶数)
			* [60.with 语句的作用，并用它写一段代码。](#60with-语句的作用并用它写一段代码)
			* [61.Python 字典和 json 字符串相互转化方法。](#61python-字典和-json-字符串相互转化方法)
			* [62.请写一个 Python 逻辑，计算一个文件中的大写字母数量。](#62请写一个-python-逻辑计算一个文件中的大写字母数量)
		- [高级特性](#高级特性)
			* [63.函数装饰器有什么作用？请列举说明。](#63函数装饰器有什么作用请列举说明)
			* [64.简述 Python 垃圾回收机制。](#64简述-python-垃圾回收机制)
			* [65.魔法函数 \_\_call\_\_怎么使用？](#65魔法函数-__call__怎么使用)
			* [66.如何判断一个对象是函数还是方法？](#66如何判断一个对象是函数还是方法)
			* [67.简述 @classmethod 和 @staticmethod 用法和区别。](#67简述-classmethod-和-staticmethod-用法和区别)
			* [68.Python 中的接口如何实现？](#68python-中的接口如何实现)
			* [69.你了解 Python 中的反射吗？](#69你了解-python-中的反射吗)
			* [70.简述 metaclass 的作用和其应用场景。](#70简述-metaclass-的作用和其应用场景)
			* [71.对比 hasattr()，getattr() 和 setattr() 的用法。](#71对比-hasattrgetattr-和-setattr-的用法)
			* [72.请列举你知道的 Python 的魔法方法及用途。](#72请列举你知道的-python-的魔法方法及用途)
			* [73.如何知道一个 Python 对象的类型？](#73如何知道一个-python-对象的类型)
			* [74.Python 中的 元类(metaclass) 使用举例。](#74python-中的-元类metaclass-使用举例)
			* [75.简述 any() 和 all() 方法。](#75简述-any-和-all-方法)
			* [76.用 filter 方法求出列表 a = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] 中的所有奇数，并构造新列表。](#76用-filter-方法求出列表-a---1-2-3-4-5-6-7-8-9-10-中的所有奇数并构造新列表)
			* [77.什么是猴子补丁？](#77什么是猴子补丁)
			* [78.在 Python 中是如何管理内存的？](#78在-python-中是如何管理内存的)
			* [79.当退出 Python 时是否释放所有内存分配？](#79当退出-python-时是否释放所有内存分配)
		- [正则表达式](#正则表达式)
			* [80.使用正则表达式匹配<html><h1>www.baidu.com</html>中的地址。](#80使用正则表达式匹配htmlh1wwwbaiducomhtml中的地址)
			* [81.正则表达式匹配中 (.\*) 和 (.\*\*?) 匹配区别？](#81正则表达式匹配中--和--匹配区别)
			* [82.写一段匹配邮箱的正则表达式。](#82写一段匹配邮箱的正则表达式)
		- [其他内容](#其他内容)
			* [83.解释一下 Python 中 pass 语句的作用。](#83解释一下-python-中-pass-语句的作用)
			* [84.简述你对 input() 函数的理解。](#84简述你对-input-函数的理解)
			* [85.Python 中的 is 和 == 有什么区别？](#85python-中的-is-和--有什么区别)
			* [86.Python 中的作用域有哪些？](#86python-中的作用域有哪些)
			* [87.三元运算写法和应用场景。](#87三元运算写法和应用场景)
			* [88.了解 enumerate 吗？](#88了解-enumerate-吗)
			* [89.列举 5 个 Python 中的标准模块。](#89列举-5-个-python-中的标准模块)
			* [90.如何在函数中设置一个全局变量？](#90如何在函数中设置一个全局变量)
			* [91.pathlib 的用法举例。](#91pathlib-的用法举例)
			* [92.请对 Python 中的异常处理写一个简单的应用场景。](#92请对-python-中的异常处理写一个简单的应用场景)
			* [93.Python 中递归的最大次数是多少？如何突破？](#93python-中递归的最大次数是多少如何突破)
			* [94.什么是面向对象的 MRO？](#94什么是面向对象的-mro)
			* [95.简述 isinstance 的作用以及应用场景。](#95简述-isinstance-的作用以及应用场景)
			* [96.什么是断言？并描述一个应用场景。](#96什么是断言并描述一个应用场景)
			* [97.lambda 表达式格式以及应用场景。](#97lambda-表达式格式以及应用场景)
			* [98.新式类和旧式类的区别有哪些？](#98新式类和旧式类的区别有哪些)
			* [99.dir() 是用来干什么的？](#99dir-是用来干什么的)
			* [100.一个包里有三个模块，demo1.py、demo2.py 和 demo3.py，但使用 from tools import \* 导入模块时，如何保证只有 demo1、demo3 被导入？](#100一个包里有三个模块demo1pydemo2py-和-demo3py但使用-from-tools-import--导入模块时如何保证只有-demo1demo3-被导入)
			* [101.列举 5 个 Python 中的异常类型以及其含义。](#101列举-5-个-python-中的异常类型以及其含义)
			* [102.copy 和 deepcopy 的区别是什么？](#102copy-和-deepcopy-的区别是什么)
			* [103.请阐述代码中经常遇到的 \*args, \*\*kwargs 的含义及用法。](#103请阐述代码中经常遇到的-args-kwargs-的含义及用法)
			* [104.Python 中会有函数或成员变量包含单下划线前缀和结尾，或双下划线前缀结尾，它们的区别是什么？](#104python-中会有函数或成员变量包含单下划线前缀和结尾或双下划线前缀结尾它们的区别是什么)
			* [105.简述 w、a 和 wb 文件写入模式的区别。](#105简述-wa-和-wb-文件写入模式的区别)
			* [106.举例 sort 和 sorted 的区别。](#106举例-sort-和-sorted-的区别)
			* [107.什么是负索引？](#107什么是负索引)
			* [108.pprint 模块是干什么的？](#108pprint-模块是干什么的)
			* [109.解释一下 Python 中的赋值运算符。](#109解释一下-python-中的赋值运算符)
			* [110.解释一下 Python 中的逻辑运算符。](#110解释一下-python-中的逻辑运算符)
			* [111.讲讲 Python 中的位运算符。](#111讲讲-python-中的位运算符)
			* [112.在 Python 中如何使用多进制数字？](#112在-python-中如何使用多进制数字)
			* [113.怎样声明多个变量并赋值？](#113怎样声明多个变量并赋值)
		- [算法和数据结构](#算法和数据结构)
			* [114.已知：](#114已知)
			* [(1) 从 AList 和 BSet 中 查找 4，最坏时间复杂度那个大？](#1-从-alist-和-bset-中-查找-4最坏时间复杂度那个大)
			* [(2) 从 AList 和 BSet 中 插入 4，最坏时间复杂度那个大？](#2-从-alist-和-bset-中-插入-4最坏时间复杂度那个大)
			* [115.用 Python 实现一个二分查找的函数。](#115用-python-实现一个二分查找的函数)
			* [116.Python 单例模式的实现方法。](#116python-单例模式的实现方法)
			* [117.使用 Python 实现一个斐波那契数列。](#117使用-python-实现一个斐波那契数列)
			* [118.找出列表中的重复数字。](#118找出列表中的重复数字)
			* [119.找出列表中的单个数字。](#119找出列表中的单个数字)
			* [120.写一个冒泡排序。](#120写一个冒泡排序)
			* [121.写一个快速排序。](#121写一个快速排序)
			* [122.写一个拓扑排序。](#122写一个拓扑排序)
			* [123.用 Python 实现一个二进制计算。](#123用-python-实现一个二进制计算)
			* [124.有一组 " " 和 "-" 符号，要求将 " " 排到左边，"-" 排到右边，写出具体的实现方法。](#124有一组--和---符号要求将--排到左边--排到右边写出具体的实现方法)
			* [125.单链表反转。](#125单链表反转)
			* [126.交叉链表求交点。](#126交叉链表求交点)
			* [127.用队列实现栈。](#127用队列实现栈)
			* [128.找出数据流的中位数。](#128找出数据流的中位数)
			* [129.二叉搜索树中第 K 小的元素。](#129二叉搜索树中第-k-小的元素)
		- [爬虫相关](#爬虫相关)
			* [130.在 requests 模块中，requests.content 和 requests.text 什么区别？](#130在-requests-模块中requestscontent-和-requeststext-什么区别)
			* [131.简要写一下 lxml 模块的使用方法框架。](#131简要写一下-lxml-模块的使用方法框架)
			* [132.说一说 scrapy 的工作流程。](#132说一说-scrapy-的工作流程)
			* [133.简述 scrapy 的去重原理。](#133简述-scrapy-的去重原理)
			* [134.scrapy 中间件有几种类，你用过哪些中间件？](#134scrapy-中间件有几种类你用过哪些中间件)
			* [135.你写爬虫的时候都遇到过什么反爬虫措施，你是怎么解决的？](#135你写爬虫的时候都遇到过什么反爬虫措施你是怎么解决的)
			* [136.为什么会用到代理？](#136为什么会用到代理)
			* [137.代理失效了怎么处理？](#137代理失效了怎么处理)
			* [138.列出你知道 header 的内容以及信息。](#138列出你知道-header-的内容以及信息)
			* [139.说一说如何打开浏览器访问 www.baidu.com 获取到结果，简述整个流程。](#139说一说如何打开浏览器访问-wwwbaiducom-获取到结果简述整个流程)
			* [140.爬取速度过快时，出现了验证码怎么处理？](#140爬取速度过快时出现了验证码怎么处理)
			* [141.scrapy 和 scrapy-redis 有什么区别？为什么选择 redis 数据库？](#141scrapy-和-scrapy-redis-有什么区别为什么选择-redis-数据库)
			* [142.分布式爬虫主要解决什么问题？](#142分布式爬虫主要解决什么问题)
			* [143.写爬虫是用多进程好还是多线程好？ 为什么？](#143写爬虫是用多进程好还是多线程好-为什么)
			* [144.解析网页的解析器使用最多的是哪几个？](#144解析网页的解析器使用最多的是哪几个)
			* [145.需要登录的网页，如何在不使用动态爬取的情况下解决 ip、cookie 和 session（其中有一些是动态生成的）的同时限制？](#145需要登录的网页如何在不使用动态爬取的情况下解决-ipcookie-和-session其中有一些是动态生成的的同时限制)
			* [146.验证码的解决。](#146验证码的解决)
			* [147.使用的最多的数据库（mysql，mongodb，redis 等）有哪些？并简述对它的理解？](#147使用的最多的数据库mysqlmongodbredis-等有哪些并简述对它的理解)
		- [网络编程](#网络编程)
			* [148.TCP 和 UDP 的区别有哪些？](#148tcp-和-udp-的区别有哪些)
			* [149.简要介绍三次握手和四次挥手。](#149简要介绍三次握手和四次挥手)
			* [150.什么是粘包？ socket 中造成粘包的原因是什么？ 哪些情况会发生粘包现象？](#150什么是粘包-socket-中造成粘包的原因是什么-哪些情况会发生粘包现象)
		- [并发](#并发)
			* [151.举例说明 conccurent.future 的中线程池的用法。](#151举例说明-conccurentfuture-的中线程池的用法)
			* [152.说一说多线程，多进程 和 协程 的区别。](#152说一说多线程多进程-和-协程-的区别)
			* [153.简述 GIL。](#153简述-gil)
			* [154.进程之间如何通信？](#154进程之间如何通信)
			* [155.IO 多路复用的作用？](#155io-多路复用的作用)
			* [156.select、poll 和 epoll 模型的区别。](#156selectpoll-和-epoll-模型的区别)
			* [157.什么是并发和并行？](#157什么是并发和并行)
			* [158.解释什么是异步非阻塞？](#158解释什么是异步非阻塞)
			* [159.简述 threading.local 的作用。](#159简述-threadinglocal-的作用)
		- [Git 面试题](#git-面试题)
			* [160.说说你知道的 git 命令。](#160说说你知道的-git-命令)
			* [161.git 如何查看某次提交修改的内容？](#161git-如何查看某次提交修改的内容)

## **一.语言特性**

#### 1.谈谈对 Python 和其他语言的区别？

Python属于解释型语言，当程序运行时，是一行一行的解释，并运行，所以调式代码很方便，开发效率高，
还有龟叔给Python定位是任其自由发展、优雅、明确、简单，所以在每个领域都有建树，所有它有着非常强大的第三方库，
特点：
语法简洁优美，功能强大，标准库与第三方库都非常强大，而且应用领域也非常广
可移植性，可扩展性，可嵌入性
缺点：
　　运行速度慢，

- 解释型
 - python/php
- 编译型
 - c/java/c#

#### 2.简述解释型和编译型编程语言？

解释型：就是边解释边执行（Python，php）
编译型：编译后再执行（c、java、c#）

#### 3.Python 的解释器种类以及相关特点？

CPython

是官方版本的解释器：CPython。是使用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，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。
PyPy

由Python写的解释器，它的执行速度是最快。PyPy采用JIT技术，对Python代码进行动态编译（注意不是解释），
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。

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。

IronPython
IronPython和Jython类似，只不过IronPython是运行在.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。

小结：
　　Python的解释器很多，但使用最广泛的还是CPython。如果要和Java或.Net平台交互，最好的办法不是用Jython或IronPython，而是通过网络调用来交互，确保各程序之间的独立性。

#### 4.说说你知道的 Python3 和 Python2 之间的区别？

1：打印时，py2需要可以不需要加括号，py3 需要
python 2 ：print ('lili') , print 'lili'
python 3 : print ('lili')
python3 必须加括号

exec语句被python3废弃，统一使用exec函数

2：内涵
Python2：1，臃肿，源码的重复量很多。
 2，语法不清晰，掺杂着C，php，Java，的一些陋习。
 3, 2020年1月1日开始停止维护
Python3：几乎是重构后的源码，规范，清晰，优美。

3、输出中文的区别
python2：要输出中文 需加 # -\*- encoding:utf-8 -\*-
Python3 ： 直接搞

4：input不同
python2 ：raw\_input
python3 ：input 统一使用input函数

5：指定字节
python2在编译安装时，可以通过参数-----enable-unicode=ucs2 或-----enable-unicode=ucs4分别用于指定使用2个字节、4个字节表示一个unicode；
python3无法进行选择，默认使用 ucs4
查看当前python中表示unicode字符串时占用的空间：

impor sys
print（sys.maxunicode）
#如果值是65535，则表示使用usc2标准，即：2个字节表示
#如果值是1114111，则表示使用usc4标准，即：4个字节表示

6：
py2：xrange
　　　　range
py3：range 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：在包的知识点里
包：一群模块文件的集合 + \_\_init\_\_
区别：py2 ： 必须有\_\_init\_\_
　　　py3：不是必须的了

8：不相等操作符"<>"被Python3废弃，统一使用"!="

9：long整数类型被Python3废弃，统一使用int

10：迭代器iterator的next()函数被Python3废弃，统一使用next(iterator)

11：异常StandardError 被Python3废弃，统一使用Exception

12：字典变量的has\_key函数被Python废弃，统一使用in关键词

13：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

#### 5.Python3 和 Python2 中 int 和 long 的区别？

在python3里，只有一种整数类型int,大多数情况下，和python２中的长整型类似。

#### 6.xrange 和 range 的区别？

xrange用法与range完全相同，所不同的是生成的不是一个数组，而是一个生成器。
要生成很大的数字序列的时候，用xrange会比range性能优很多，因为不需要一上来就开辟一块很大的内存空间，这两个基本上都是在循环的时候用。

在 Python 3 中，range() 是像 xrange() 那样实现，xrange()被抛弃。

## **二.编码规范**

#### 7.什么是 PEP8？

PEP是 Python Enhancement Proposal 的缩写，翻译过来就是 Python增强建议书
简单说就是一种编码规范，是为了让代码“更好看”，更容易被阅读
具体可参考：
https://www.python.org/dev/peps/pep-0008/

#### 8.了解 Python 之禅吗？

import this

#### 9.了解 docstring 吗？

Python有一个很奇妙的特性，称为 文档字符串 ，它通常被简称为 docstrings 。DocStrings是一个重要的工具，由于它帮助你的程序文档更加简单易懂，你应该尽量使用它。你甚至可以在程序运行的时候，从函数恢复文档字符串。
使用魔法方法'\_\_doc\_\_'可以打印docstring的内容

#### 10.了解类型注解吗？

def add(x:int, y:int) -> int:
 return x + y
用 : 类型 的形式指定函数的参数类型，用 -> 类型 的形式指定函数的返回值类型

#### 11.例举你知道 Python 对象的命名规范，例如方法或者类等。

类名都使用首字母大写开头(Pascal命名风格)的规范；
全局变量全用大写字母，单词之间用 \_分割；
普通变量用小写字母，单词之间用 \_分割；
普通函数和普通变量一样；
私有函数以 \_\_ 开头（2个下划线），其他和普通函数一样；

#### 12.Python 中的注释有几种？

单行注释，多行注释，docstring注释

#### 13.如何优雅的给一个函数加注释？

在函数逻辑行的首行使用""" xxx """给函数添加注释，注释中可包含函数参数的说明，返回值说明等
def foo(bar):
 """
 This is an example.
 :param bar: explain param bar
 """
 return bar

#### 14.如何给变量加注释？

参数注释：以冒号（:）标记
返回值注释：以 -> 标记
示例：
def add(x:int, y:int) -> int:
 return x + y

#### 15.Python 代码缩进中是否支持 Tab 键和 空格 混用？

支持，Python 并没有强制要求你用Tab缩进或者用空格缩进，但在 PEP8中，建议使用4个空格来缩进

#### 16.是否可以在一句 import 中导入多个库？

可以的
import json,random,requests

#### 17.在给 Python 文件命名的时候需要注意什么？

全小写，单词之间使用下划线分隔

#### 18.例举几个规范 Python 代码风格的工具。

pylint,black,pycharm也带有pep8的代码规范工具

## **三.数据类型**

### **字符串**

#### 19.列举 Python 中的基本数据类型。

Python3 中有六个标准的数据类型：

Number（数字）
String（字符串）
List（列表）
Tuple（元组）
Set（集合）
Dictionary（字典）
Python3 的六个标准数据类型中：

不可变数据（3 个）：Number（数字）、String（字符串）、Tuple（元组）；
可变数据（3 个）：List（列表）、Dictionary（字典）、Set（集合）。

#### 20.如何区别 可变数据类型 和 不可变数据类型？

Python3 的六个标准数据类型中：

不可变数据（3 个）：Number（数字）、String（字符串）、Tuple（元组）；
可变数据（3 个）：List（列表）、Dictionary（字典）、Set（集合）。

#### 21.将 "hello world" 转换为首字母大写 "Hello World"。

z = 'hello world'
z.title()

#### 22.如何检测字符串中只含有数字？

# 分为两种情况
# 1.不包含正负号 +-
a = '32323'
a.isdigit()
# 2.含有正负号
import re
re.match(r'[+-]?\d+$',a)

#### 23.将字符串 "ilovechina" 进行反转。

s = 'ilovechina'
s = s[::-1]

#### 24.Python 中的字符串格式化方式你知道哪些？

# Python3.6之后的版本提供了三种字符串格式化的方式
# 1. %s占位符
def foo(name):
 return 'hello %s' % name
# 2. format()
def foo(name):
 return 'hello {}'.format(name)
# f-string
def foo(name):
 return f'hello {name}'

#### 25.有一个字符串开头和末尾都有空格，比如 " adabdw "。要求写一个函数把这个字符串的前后空格都去掉。

s = " adabdw "
s.strip()

#### 26.获取字符串 "123456" 最后的两个字符。

s = '123456'

print(s[-2:])

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

s.encode('utf-8')

#### 28.字符串 s = "info:xiaoZhang 33 shandong"，用正则切分字符串输出 ['info', 'xiaoZhang', '33', 'shandong']。

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

#### 29.怎样将字符串转换为小写？

b = 'HHH'
b.lower()

#### 30.单引号、双引号和三引号的区别？

s = 'hello'
s= "hello"
双引号可以包含单引号字符，单引号无法包含双引号字符
三引号可以用来加注释，所加注释可以使用\_\_doc\_\_查看

#### 31.字符串 a = "你好 中国 "，去除多余空格只留一个空格。

a = "你好 中国 "
s = ' '.join(a.strip().split())

### **列表**

#### 32.已知 AList = [1,2,3,1,2]，对 AList 列表元素去重，写出具体过程。

a\_list = [1,2,3,1,2]
ss = set(a\_list)

#### 33.如何将 "1,2,3" 变成 ["1","2","3"]？

s = "1,2,3"
s.split(',')

#### 34.给定两个 list，A 和 B，找出相同元素和不同元素。

# 最直接的方法
list\_a = [1,2,3,4,5,6]
list\_b = [2,3,6]
same\_l = []
not\_same = []
for i in list\_a:
 if i not in list\_b:
 not\_same.append(i)
for j in list\_b:
 if j not in list\_a:
 not\_same.append(j)
for x in list\_a:
 if x in list\_b:
 same\_l.append(x)
# 奇技淫巧
list\_a = [1,2,3,4,5,6]
list\_b = [2,3,6]
set1 = set(list\_a)
set2 = set(list\_b)
# 相同元素
print(set1&set2)
# 不同元素
print(set1^set2)

#### 35.用一行代码展开该列表 [[1,2],[3,4],[5,6]]，得出[1,2,3,4,5,6]。

mm = [[1,2],[3,4],[5,6]]
[j for a in mm for j in a]

#### 36.合并列表 [1,5,7,9] 和 [2,2,6,8]。

a = [1,5,7,9]
b = [2,2,6,8]
# 方法1
a.extend(b)
# 方法2
a[0:0] = b
# 方法3
a += b

#### 37.如何打乱一个列表的元素？

import random
a = [1,5,7,9]
random.shuffle(a)

### **字典**

#### 38.字典操作中 del 和 pop 有什么区别？

del 操作删除键值对，不返回值；
pop 操作删除键值对的同时，返回键所对应的值。

#### 39.将如下字典按照年龄排序。

d1 = [
 {'name':'alice', 'age':38},
 {'name':'bob', 'age':18},
 {'name':'Carl', 'age':28},
]

sorted(d1,key=lambda x:x['age'])

#### 40.请合并下面两个字典 a = {"A":1,"B":2}，b = {"C":3,"D":4}。

# python3合并字典有三种方式
# 1.
a = {'a':1,'b':2}
b = {'c':3,'d':4}
c = {}
c.update(a)
c.update(b)
# 2.
c = dict(a,\*\*b)
# 3.
c = {\*\*a,\*\*b} # 官方推荐这种方式

#### 41.如何使用生成式的方式生成一个字典，写一段功能代码。

{x:x\*x for x in range(6)}

#### 42.如何把 元组 ("a","b") 和 元组(1,2)，变为字典{"a":1,"b":2}？

a = ('a','b')
b = (1,2)
z=zip(a,b)
c = dict(z)

### **综合数据类型**

#### 43.Python 常用的数据结构的类型及其特性？

List,tuple,dict,set是比较常用的数据结构，queue,heap,deque,ProrityQueue，multiprocessing.Queue等进阶的数据结构类型。特性就去查查吧，写在这里太长了。

#### 44.如何将 元组("A","B") 和 元组(1,2) 合并成 字典{"A":1,"B":2}？

a = ('A','B')
b = (1,2)
z=zip(a,b)
c = dict(z)

#### 45.Python 里面如何实现 tuple 和 list 的转换？

tuple(list) # tuple转list
list(tuple) # list 转tuple

#### 46.我们知道对于列表可以使用切片操作进行部分元素的选择，那么如何对生成器类型的对象实现相同的功能呢？

使用自带的itertools库进行实现，具体实现方式 itertools.islice(生成器对象，起始位置，结束位置)，即可实现切片功能。

#### 47.请将 [i for i in range(3)] 改成 生成器。

(i for i in range(3))

#### 48.将 a="hello" 和 b="你好" 编码成 bytes 类型。

a.encode()
b.encode()

#### 49.下面的代码输出结果是什么？

a = (1,2,3,[4,5,6,7],8)
a[2] = 2
报错，元组是不可变对象，不支持修改

#### 50.下面的代码输出的结果是什么？

a = (1,2,3,[4,5,6,7],8)
a[5] = 2
报错，元组是不可变对象，下标越界

### **操作类题目**

#### 51.在 Python 中交换两个变量的值。

a,b = b,a

#### 52.在读文件操作的时候会使用 read、readline 或者 readlines，简述它们各自的作用。

read将整个文本都读取为一个字符串，占用内存大，readline读取为一个生成器，支持遍历和迭代，占用空间小。readlines将文本读取为列表，占用空间大

#### 53.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？

字符串、数字（整数和浮点数）、字典、列表、布尔值、None。使用strftime将datetime格式化为标准字符串类型即可。

#### 54.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？

使用json.dumps函数时，添加参数ensure\_ascii=False，如果想显示的更美观，可以添加indent=2参数，会在每个key值前添加两个空格。

#### 55.有两个磁盘文件 A 和 B，各存放一行字母，要求把这两个文件中的信息合并（按字母顺序排列），输出到一个新文件 C 中。

with open('A.txt','r') as f:
 a = f.readlines()[0]
with open('B.txt','r') as f:
 b = f.readlines()[0]
 a.extend(b).sort()
with open('C.txt','w') as f:
 for i in a:
 f.write(i)

#### 56.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)。

import datetime
def getday(n):
 y,m,d = 2019,5,30
 the\_date = datetime.datetime(y,m,d)
 result\_date = the\_date + datetime.timedelta(days=n)
 target\_date = result\_date.strftime('%Y%m%d')
 return target\_date

#### 57.写一个函数，接收整数参数 n，返回一个函数。函数的功能是把函数的参数和 n 相乘并把结果返回。

def mul(n):
 def wrapper(m):
 return n\*m
 return wrapper

 # 闭包的基本操作

#### 58.下面的代码会存在什么问题，如何改进？

def strappend(num):
 str='first'
 for i in range(num):
 str+=str(i)
 return str
 # 将str(i)改为str[i]

#### 59.一行代码输出 1-100 之间的所有偶数。

[x for x in range(101) if x %2 ==0]

#### 60.with 语句的作用，并用它写一段代码。

# with语句用来管理资源，及时关闭文件等操作，避免资源的泄漏
with open('a.txt','r') as f:
 f.read()

#### 61.Python 字典和 json 字符串相互转化方法。

json.dumps() 将Python中的对象转换为JSON中的字符串对象
json.loads() 将JSON中的字符串对象转换为Python中的对象

#### 62.请写一个 Python 逻辑，计算一个文件中的大写字母数量。

import re
with open('a.txt','r') as f:
 f\_content = f.read()
 len\_cap = len(re.compile(r'[A-Z]').findall(f\_content))

### **高级特性**

#### 63.函数装饰器有什么作用？请列举说明。

函数装饰器可以在不修改原函数的条件下，为原函数添加额外的功能，例如记录日志，运行性能，缓存等
以记录函数运行时间为例，实现一个装饰器
import time
def time\_it(func):
 def wrapper(func):
 start\_time = time.time()
 res = func()
 end\_time = time.time()
 return start\_time - end\_time
 return wrapper

#### 64.简述 Python 垃圾回收机制。

引用计数机制：
python里每一个东西都是对象，它们的核心就是一个结构体：PyObject
 typedef struct\_object {
 int ob\_refcnt;
 struct\_typeobject \*ob\_type;
} 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
#define Py\_INCREF(op) ((op)->ob\_refcnt++) //增加计数
#define Py\_DECREF(op) \ //减少计数
 if (--(op)->ob\_refcnt != 0) \
 ; \
 else \
 \_\_Py\_Dealloc((PyObject \*)(op))

当引用计数为0时，该对象生命就结束了。
引用计数机制的优点：

简单
实时性：一旦没有引用，内存就直接释放了。不用像其他机制等到特定时机。实时性还带来一个好处：处理回收内存的时间分摊到了平时

#### 65.魔法函数 \_\_call\_\_怎么使用？

class Bar:
 def \_\_call\_\_(self, \*args, \*\*kwargs):
 print('i am instance method')

b = Bar() # 实例化
b() # 实例对象b 可以作为函数调用 等同于b.\_\_call\_\_ 使用

# OUT: i am instance method

# 带参数的类装饰器
class Bar:

 def \_\_init\_\_(self, p1):
 self.p1 = p1

 def \_\_call\_\_(self, func):
 def wrapper():
 print("Starting", func.\_\_name\_\_)
 print("p1=", self.p1)
 func()
 print("Ending", func.\_\_name\_\_)
 return wrapper

@Bar("foo bar")
def hello():
 print("Hello")

#### 66.如何判断一个对象是函数还是方法？

判断对象是函数或方法应该使用type(obj)

#### 67.简述 @classmethod 和 @staticmethod 用法和区别。

一般来说，要使用某个类的方法，需要先实例化一个对象再调用方法。

而使用@staticmethod或@classmethod，就可以不需要实例化，直接类名.方法名()来调用。

这有利于组织代码，把某些应该属于某个类的函数给放到那个类里去，同时有利于命名空间的整洁。

既然@staticmethod和@classmethod都可以直接类名.方法名()来调用，那他们有什么区别呢

从它们的使用上来看,
@staticmethod不需要表示自身对象的self和自身类的cls参数，就跟使用函数一样。
@classmethod也不需要self参数，但第一个参数需要是表示自身类的cls参数。
如果在@staticmethod中要调用到这个类的一些属性方法，只能直接类名.属性名或类名.方法名。

而@classmethod因为持有cls参数，可以来调用类的属性，类的方法，实例化对象等，避免硬编码
class A(object):
 bar = 1
 def foo(self):
 print 'foo'

 @staticmethod
 def static\_foo():
 print 'static\_foo'
 print A.bar

 @classmethod
 def class\_foo(cls):
 print 'class\_foo'
 print cls.bar
 cls().foo()

A.static\_foo()
A.class\_foo()

#### 68.Python 中的接口如何实现？

#抽象类加抽象方法就等于面向对象编程中的接口
from abc import ABCMeta,abstractmethod

class interface(object):
 \_\_metaclass\_\_ = ABCMeta #指定这是一个抽象类
 @abstractmethod #抽象方法
 def Lee(self):
 pass

 def Marlon(self):
 pass

class RelalizeInterfaceLee(interface):#必须实现interface中的所有函数，否则会编译错误
 def \_\_init\_\_(self):
 print '这是接口interface的实现'
 def Lee(self):
 print '实现Lee功能'
 def Marlon(self):
 pass

#### 69.你了解 Python 中的反射吗？

通过字符串映射object对象的方法或者属性
hasattr(obj,name\_str): 判断objec是否有name\_str这个方法或者属性
getattr(obj,name\_str): 获取object对象中与name\_str同名的方法或者函数
setattr(obj,name\_str,value): 为object对象设置一个以name\_str为名的value方法或者属性
delattr(obj,name\_str): 删除object对象中的name\_str方法或者属性

举个栗子
import requests

class Http(object):

 def get(self,url):
 res = requests.get(url)
 response = res.text
 return response

 def post(self,url):
 res = requests.post(url)
 response = res.text
 return response

# 使用反射后
url = "https://www.jianshu.com/u/14140bf8f6c7"
method = input("请求方法>>>:")
h = Http()

if hasattr(h,method):
 func = getattr(h,method)
 res = func(url)
 print(res)
else:
 print("你的请求方式有误...")

#### 70.简述 metaclass 的作用和其应用场景。

metaclass我没怎么用过，不能乱说误人子弟，可以看下这篇博文https://www.cnblogs.com/xybaby/p/7927407.html

#### 71.对比 hasattr()，getattr() 和 setattr() 的用法。

hasattr(obj,name\_str): 判断objec是否有name\_str这个方法或者属性
getattr(obj,name\_str): 获取object对象中与name\_str同名的方法或者函数
setattr(obj,name\_str,value): 为object对象设置一个以name\_str为名的value方法或者属性
delattr(obj,name\_str): 删除object对象中的name\_str方法或者属性

#### 72.请列举你知道的 Python 的魔法方法及用途。

1. \_\_call\_\_:允许一个类的实例像函数一样被调用。实质上说，这意味着 x() 与 x.\_call\_() 是相同的
2.\_\_init\_\_:显示初始化属性
3.\_\_str\_\_,\_\_repr\_\_,定义类的时候，重写这两个方法可以让类更清晰
再就是\_\_setattr\_\_,\_\_getattr\_\_,\_\_delattr\_\_等等

#### 73.如何知道一个 Python 对象的类型？

type(obj)

81.**Python 的传参是传值还是传址？**

说传值或者传引用都不准确。非要安一个确切的叫法的话，叫传对象（call by object）
具体可以参考这篇文章：https://foofish.net/python-function-args.html

#### 74.Python 中的 元类(metaclass) 使用举例。

参考77.

#### 75.简述 any() 和 all() 方法。

#any(x)判断x对象是否为空对象，如果都为空、0、false，则返回false，如果不都为空、0、false，则返回true

#all(x)如果all(x)参数x对象的所有元素不为0、''、False或者x为空对象，则返回True，否则返回False
>>> any('123')
True
>>> any([0,1])
True
>>> any([0,'0',''])
True
>>> any([0,''])
False
>>> any([0,'','false'])
True
>>> any([0,'',bool('false')])
True
>>> any([0,'',False])
False
>>> any(('a','b','c'))
True
>>> any(('a','b',''))
True
>>> any((0,False,''))
False
>>> any([])
False
>>> any(())
False
>>> all(['a', 'b', 'c', 'd']) #列表list，
True
>>> all(['a', 'b', 'c', 'd']) #列表list，元素都不为空或0
True
>>> all(['a', 'b', '', 'd']) #列表list，存在一个为空的元素
False
>>> all([0, 1,2, 3]) #列表list，存在一个为0的元素
False
>>> all(('a', 'b', 'c', 'd')) #元组tuple，元素都不为空或0
True
>>> all(('a', 'b', '', 'd')) #元组tuple，存在一个为空的元素
False
>>> all((0, 1,2, 3)) #元组tuple，存在一个为0的元素
False
>>> all([]) # 空列表
True
>>> all(()) # 空元组
True
>>> #注意：空元组、空列表返回值为True，这里要特别注意

#### 76.用 filter 方法求出列表 a = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] 中的所有奇数，并构造新列表。

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

#### 77.什么是猴子补丁？

猴子补丁是一个概念，不是python中发明的，其他动态语言也有这么个概念。 《松本行弘的程序世界》这本书，里面专门有一章讲了猴子补丁的设计，所谓的猴子补丁的含义是指在动态语言中，不去改变源码而对功能进行追加和变更

#### 78.在 Python 中是如何管理内存的？

参考71.

#### 79.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。

### **正则表达式**

#### 80.使用正则表达式匹配<html><h1>www.baidu.com</html>中的地址。

import re
s = '<html><h1>www.baidu.com</html>'
p = re.compile(r'<html><h1>(.\*?)</html>')
result = re.findall(p,s)[0]

a="张明 98 分"，用 re.sub，将 98 替换为 100

import re
a="张明 98 分"
pa = re.compile(r'\d+')
re.sub(pa,'100',a)

#### 81.正则表达式匹配中 (.\*) 和 (.\*\*?) 匹配区别？

加？会将贪婪模式改成懒惰模式,如果有问号的话，则表示匹配0个或1个问号前面的表达式

#### 82.写一段匹配邮箱的正则表达式。

r'[0-9a-zA-Z\_]\*@.+\.(com|cn|net)$'

### **其他内容**

#### 83.解释一下 Python 中 pass 语句的作用。

Python pass 是空语句，是为了保持程序结构的完整性。

pass 不做任何事情，一般用做占位语句。

#### 84.简述你对 input() 函数的理解。

在python3中，input()获取用户输入，不论用户输入的是什么，获取到的都是字符串类型的。

#### 85.Python 中的 is 和 == 有什么区别？

is比较的是两个对象的id值是否相等，也就是比较两个对象是否为同一个实例对象，是否指向同一个内存地址。

==比较的是两个对象的内容是否相等，默认会调用对象的\_\_eq\_\_()方法。

#### 86.Python 中的作用域有哪些？

L （Local） 局部作用域
E （Enclosing） 闭包函数外的函数中
G （Global） 全局作用域
B （Built-in） 内建作用域

#### 87.三元运算写法和应用场景。

条件语句比较简单时可以使用三元运算符，最常见的就是
res = 'test True' if expression is True else 'test False'

#### 88.了解 enumerate 吗？

# 遍历列表时候，携带索引index
a = ['a','b','c']
for index,item in enumerate(a):
 print(index,item)

#### 89.列举 5 个 Python 中的标准模块。

json,re,random,datetime,codecs

#### 90.如何在函数中设置一个全局变量？

使用global关键字

#### 91.pathlib 的用法举例。

from pathlib import Path

data\_folder = Path("source\_data/text\_files/")

file\_to\_open = data\_folder / "raw\_data.txt"

f = open(file\_to\_open)

print(f.read())

#### 92.请对 Python 中的异常处理写一个简单的应用场景。

class NameTooShortError(ValueError):
 pass
def validate(name):
 if len(name) < 10:
 raise NameTooShortError(name)

# 使用自定义异常类，使得发生异常时的错误更容易排查

#### 93.Python 中递归的最大次数是多少？如何突破？

python解释器限制最大的递归深度是999，可以通过
import sys
sys.setrecursionlimit(10000) # set the maximum depth as 10000
重新设置最大递归深度

#### 94.什么是面向对象的 MRO？

对于支持继承的编程语言来说，其方法（属性）可能定义在当前类，也可能来自于基类，所以在方法调用时就需要对当前类和基类进行搜索以确定方法所在的位置。而搜索的顺序就是所谓的「方法解析顺序」（Method Resolution Order，或MRO）。对于只支持单继承的语言来说，MRO 一般比较简单；而对于 Python 这种支持多继承的语言来说，MRO 就复杂很多。

#### 95.简述 isinstance 的作用以及应用场景。

来判断一个对象是否是一个已知的类型
举个栗子：
 p= 'sfa'
 isinstance(p,str)
 True

#### 96.什么是断言？并描述一个应用场景。

Python 的断言语句是一种调试工具，用来测试某个断言条件。如果断言条件 为真，则程序将继续正常执行;但如果条件为假，则会引发 AssertionError 异常并显示相关 的错误消息。

#### 97.lambda 表达式格式以及应用场景。

d = {'a':2,'b':1,'c':3,'d':'4'}
sorted(d,key=lambda x :x[1])
# 将字典d按照值排序

#### 98.新式类和旧式类的区别有哪些？

Python 有两种类：经典类（classic class）和新式类（new-style class）。两者的不同之处在于新式类继承自 object。在 Python 2.1 以前，经典类是唯一可用的形式；Python 2.2 引入了新式类，使得类和内置类型更加统一；在 Python 3 中，新式类是唯一支持的类。

#### 99.dir() 是用来干什么的？

dir() 函数不带参数时，返回当前范围内的变量、方法和定义的类型列表；带参数时，返回参数的属性、方法列表。如果参数包含方法\_\_dir\_\_()，该方法将被调用。如果参数不包含\_\_dir\_\_()，该方法将最大限度地收集参数信息。

#### 100.一个包里有三个模块，demo1.py、demo2.py 和 demo3.py，但使用 from tools import \* 导入模块时，如何保证只有 demo1、demo3 被导入？

但若想使用from pacakge\_1 import \*这种形式的写法，需在  init.py中加上：   all = [‘file\_a’, ‘file\_b’] #package\_1下有file\_a.py和file\_b.py，在导入时init.py文件将被执行。
但不建议在 init.py中写模块，以保证该文件简单。不过可在init.py导入我们需要的模块，以便避免一个个导入、方便使用。

#### 101.列举 5 个 Python 中的异常类型以及其含义。

1. ArithmeticError 此基类用于派生针对各种算术类错误而引发的内置异常: OverflowError, ZeroDivisionError, FloatingPointError
2. BufferError 当与 缓冲区 相关的操作无法执行时将被引发。
3. LookupError 此基类用于派生当映射或序列所使用的键或索引无效时引发的异常: IndexError, KeyError。 这可以通过 codecs.lookup() 来直接引发
4. ImportError 当 import 语句尝试加载模块遇到麻烦时将被引发。 并且当 from ... import 中的 "from list" 存在无法找到的名称时也会被引发
5. IndexError 当序列抽取超出范围时将被引发。 （切片索引会被静默截短到允许的范围；如果指定索引不是整数则 TypeError 会被引发

#### 102.copy 和 deepcopy 的区别是什么？

copy 即所谓的浅拷贝，赋值的时候非递归地复制子对象的引用；
deepcopy 即所谓的深拷贝，赋值的时候递归复制子对象。
举个栗子，
xs = [1,2,[2,3,4],3]
ys = xs # 浅拷贝
zs = deepcopy(xs) # 深拷贝
xs[2][0] = 5
print(ys)
[1,2,[2,3,4],3]
print(xs)
[1,2,[5,3,4],3]

print(zs)
[1,2,[5,3,4],3] # 由于深拷贝已经递归复制了子对象，所以内部的List也发生改变

#### 103.请阐述代码中经常遇到的 \*args, \*\*kwargs 的含义及用法。

这两个是python中的可变参数。\*args表示任何多个位置参数，它是一个tuple；\*\*kwargs表示关键字参数，它是一个dict。并且同时使用\*args和\*\*kwargs时，必须\*args参数列要在\*\*kwargs前

#### 104.Python 中会有函数或成员变量包含单下划线前缀和结尾，或双下划线前缀结尾，它们的区别是什么？

前置单下划线\_var:命名约定，用来表示该名称仅在内部使用。一般对 Python 解释器没 有特殊含义(通配符导入除外)，只能作为对程序员的提示。
后置单下划线 var\_:命名约定，用于避免与 Python 关键字发生命名冲突。
前置双下划线\_\_var:在类环境中使用时会触发名称改写，对 Python 解释器有特殊含义。
前后双下划线\_\_var\_\_:表示由 Python 语言定义的特殊方法。在自定义的属性中要避免
使用这种命名方式。

#### 105.简述 w、a+ 和 wb 文件写入模式的区别。

w:写入时会覆盖上一次的写入
a+:追加写入
wb:以二进制文件形式写入

#### 106.举例 sort 和 sorted 的区别。

sort()与sorted()的不同在于，sort是在原位重新排列列表，而sorted()是产生一个新的列表

#### 107.什么是负索引？

负索引和正索引不同，它是从右边开始检索。例如：使用负索引取出列表的最后一个数
lis[-1] # 取出列表的最后一个元素
lis[-2] # 取出列表的倒数第二个元素

#### 108.pprint 模块是干什么的？

print()和pprint()都是python的打印模块，功能基本一样，唯一的区别就是pprint()模块打印出来的数据结构更加完整，每行为一个数据结构，更加方便阅读打印输出结果。特别是对于特别长的数据打印，print()输出结果都在一行，不方便查看，而pprint()采用分行打印输出，所以对于数据结构比较复杂、数据长度较长的数据，适合采用pprint()打印方式

#### 109.解释一下 Python 中的赋值运算符。

在python中，使用 = 可以给变量赋值。
在算术运算时，为了简化代码的编写，Python还提供了一系列与算术运算符对应的赋值运算符
例如，c += a 等效于 c=c+a

#### 110.解释一下 Python 中的逻辑运算符。

and, or, not

#### 111.讲讲 Python 中的位运算符。

& 按位与运算符：参与运算的两个值,如果两个相应位都为1,则该位的结果为1,否则为0 (a & b) 输出结果 12 ，二进制解释： 0000 1100
| 按位或运算符：只要对应的二个二进位有一个为1时，结果位就为1 (a | b) 输出结果 61 ，二进制解释： 0011 1101
^ 按位异或运算符：当两对应的二进位相异时，结果为1 (a ^ b) 输出结果 49 ，二进制解释： 0011 0001
~ 按位取反运算符：对数据的每个二进制位取反,即把1变为0,把0变为1 (~a ) 输出结果 -61 ，二进制解释： 1100 0011， 在一个有符号二进制数的补码形式。
<< 左移动运算符：运算数的各二进位全部左移若干位，由”<<”右边的数指定移动的位数，高位丢弃，低位补0 a << 2 输出结果 240 ，二进制解释： 1111 0000
>> 右移动运算符：把”>>”左边的运算数的各二进位全部右移若干位，”>>”右边的数指定移动的位数 a >> 2 输出结果 15 ，二进制解释： 0000 1111

#### 112.**在 Python 中如何使用多进制数字？**

1、二进制数字由0和1组成，我们使用0b或0B前缀表示二进制数

print(int(0b1010))#10
2、使用bin()函数将一个数字转换为它的二进制形式

print(bin(0xf))#0b1111
3、八进制数由数字0-7组成，用前缀0o或0O表示8进制数

print(oct(8))#0o10
4、十六进数由数字0-15组成，用前缀0x或者0X表示16进制数

print(hex(16))#0x10
print(hex(15))#0xf

#### 113.怎样声明多个变量并赋值？

a,b = 1,2

### **算法和数据结构**

#### 114.已知：

AList = [1,2,3]
BSet = {1,2,3}

##### (1) 从 AList 和 BSet 中 查找 4，最坏时间复杂度那个大？

查找列表最坏时间复杂度是O(n),查找字典是O(1)，因为字典的数据结构是散列表

##### (2) 从 AList 和 BSet 中 插入 4，最坏时间复杂度那个大？

插入的操作都是O(1)

#### 115.用 Python 实现一个二分查找的函数。

def binary\_search(item,arr):
 start = 0
 end = len(arr) - 1
 while start <= end:
 mid = (start + end) // 2
 if item == arr[mid]:
 return True
 elif item < arr[mid]:
 end = mid - 1
 else:
 start = mid + 1
array = [1,2,3,4,5,6]
print(binary\_search(2,array))

#### 116.Python 单例模式的实现方法。

单例模式（Singleton Pattern）是一种常用的软件设计模式，该模式的主要目的是确保某一个类只有一个实例存在。当你希望在整个系统中，某个类只能出现一个实例时，单例对象就能派上用场。

比如，某个服务器程序的配置信息存放在一个文件中，客户端通过一个 AppConfig 的类来读取配置文件的信息。如果在程序运行期间，有很多地方都需要使用配置文件的内容，也就是说，很多地方都需要创建 AppConfig 对象的实例，这就导致系统中存在多个 AppConfig 的实例对象，而这样会严重浪费内存资源，尤其是在配置文件内容很多的情况下。事实上，类似 AppConfig 这样的类，我们希望在程序运行期间只存在一个实例对象
1.使用装饰器实现单例模式
def singleton(cls):
 \_instance = {}
 def \_singleton(\*args,\*\*kargs):
 if cls not in \_instance:
 \_instance[cls] = cls(\*args,\*\*args)
 return \_instance[cls]
 return \_singleton

2.使用类实现单例模式
import threading
class Singleton(object):
 \_instance\_lock = threading.Lock()

 def \_\_init\_\_(self):
 pass

 def \_\_new\_\_(cls, \*args, \*\*kwargs):
 if not hasattr(Singleton, "\_instance"):
 with Singleton.\_instance\_lock:
 if not hasattr(Singleton, "\_instance"):
 Singleton.\_instance = object.\_\_new\_\_(cls)
 return Singleton.\_instanc

#### 117.使用 Python 实现一个斐波那契数列。

# 很多人上来就写下面这种解法
def fibnaqie(n):
 if n < 2:
 return n
 return fibnaqie(n - 1) + fibnaqie(n - 2)
# 这种解法时间复杂度高，而且一般不是面试官最想要的答案

# 下面这种优化的版本
def fib2(n):
 fib\_n = 0
 fib\_one = 1
 fib\_two = 0
 res = [0, 1]
 if n < 2:
 return res[n]
 for i in range(2, n + 1):
 fib\_n = fib\_one + fib\_two
 fib\_two = fib\_one
 fib\_one = fib\_n
 return fib\_n
# 所以适当地选择递归还是迭代，要看具体情况，处理树，或者图的遍历的时候，递归还是比迭代优先考虑的

#### 118.找出列表中的重复数字。

def find\_duplicate(arr):
 not\_dup = set()
 dup = set()
 for x in arr:
 if x not in not\_dup:
 not\_dup.add(x)
 else:
 dup.add(x)
 return dup

if \_\_name\_\_ == '\_\_main\_\_':
 array = [1, 2, 3, 4, 4, 4, 4]
 print(find\_duplicate(array))

#### 119.找出列表中的单个数字。

a = ['a', 1, 2, 'b']
for x in a:
 if str(x).isdigit():
 print(x)

#### 120.写一个冒泡排序。

def bubble\_sort(arr):
 n = len(arr)
 if len(arr) < 2:
 return arr
 for i in range(n - 1):
 count = 0
 for j in range(n - i - 1):
 if arr[j] > arr[j + 1]:
 arr[j], arr[j + 1] = arr[j + 1], arr[j]
 count += 1
 if count == 0:
 return arr
 return arr

if \_\_name\_\_ == '\_\_main\_\_':
 arr = [2, 1, 5, 3, 6]
 arr1 = [1, 2, 3, 4, 5]
 print(bubble\_sort(arr1))

#### 121.写一个快速排序。

def quick\_sort(arr):
 if len(arr) < 2:
 return arr
 else:
 pivot = arr[0]
 less = [i for i in arr[1:] if i <= pivot]
 more = [i for i in arr[1:] if i > pivot]
 return quick\_sort(less) + [pivot] + quick\_sort(more)

if \_\_name\_\_ == '\_\_main\_\_':
 array = [2, 1, 6, 3, 4, 5]
 print(quick\_sort(array))

#### 122.写一个拓扑排序。

# 使用循环进行拓扑排序
def topoSort(G):
 # 初始化计算被指向节点的字典
 cnt = dict((u, 0) for u in G.keys())
 import pdb;pdb.set\_trace()
 # 若某节点被其他节点指向，该节点计算量+1
 for u in G:
 for v in G[u]:
 cnt[v] += 1
 # 收集被指向数为0的节点,此时Q只有一个节点，即起始节点a
 Q = [u for u in cnt.keys() if cnt[u] == 0]
 # 记录结果
 seq = []
 while Q:
 s = Q.pop()
 seq.append(s)
 for u in G[s]:
 cnt[u] -= 1
 if cnt[u] == 0:
 Q.append(u)
 return seq

# 有向无环图的邻接字典
G = {
 'a': {'b', 'f'},
 'b': {'c', 'd', 'f'},
 'c': {'d'},
 'd': {'e', 'f'},
 'e': {'f'},
 'f': {}
}

res = topoSort(G)
print(res)

#### 123.用 Python 实现一个二进制计算。

'''
以实现二进制加法为例
给定两个二进制字符串，返回他们的和（用二进制表示）。
输入为非空字符串且只包含数字 1 和 0。
输入: a = “11”, b = “1”
输出: “100”
'''

def binary\_plus(a, b):
 a, b = int('0b' + a, 2), int('0b'+b, 2)
 return bin(a + b)[2:]

if \_\_name\_\_ == '\_\_main\_\_':
 a = '11'
 b = '1'
 print(binary\_plus(a, b))

# 解释一下，class int(x, base)
x–字符串或数字
base–进制数，默认十进制。
bin()函数返回一个整型int或者长整数long int的二进制表示，bin()运算返回的是二进制。所以前两位是二进制的标志，需要[2:]去除

#### 124.有一组 "+" 和 "-" 符号，要求将 "+" 排到左边，"-" 排到右边，写出具体的实现方法。

def some\_sort(arr):
 list\_a = []
 list\_b = []
 for x in arr:
 if x == '+':
 list\_a.append(x)
 else:
 list\_b.append(x)
 list\_a.extend(list\_b)
 print(list\_a)
 return list\_a

#### 125.单链表反转。

class ListNode:
 def \_\_init\_\_(self, val):
 self.val = val
 self.next = None

def reverse\_node(head):
 p = head
 q = head.next
 head.next = None
 while q:
 r = q.next
 q.next = p
 p = q
 q = r
 head = p
 return head

if \_\_name\_\_ == '\_\_main\_\_':
 l1 = ListNode(3)
 l1.next = ListNode(2)
 l1.next.next = ListNode(1)

 l = reverse\_node(l1)
 print(l.val, l.next.val, l.next.next.val)
# 1,2,3

#### 126.交叉链表求交点。

'''先遍历两个链表，获知两个链表各自的长度，往后调整较长链表的指针，使之与较短链表的起始指针对齐，然后两个链表同时往后扫描，直至二者的节点相同，证明该节点是相交节点，否则返回 None，表明两个链表不相交，时间复杂度o(n),空间复杂度0(1)'''
class ListNode:
 def \_\_init\_\_(self, val):
 self.val = val
 self.next = None

def solution(head\_a, head\_b):
 def get\_list\_length(head):
 len = 0
 while head:
 len += 1
 head = head.next
 return head

 def forward\_long\_list(long\_len, short\_len, head):
 delta = long\_len - short\_len
 while delta and head:
 head = head.next
 delta -= 1
 return head

 def main\_func(head\_a, head\_b):
 head\_a\_len = get\_list\_length(head\_a)
 head\_b\_len = get\_list\_length(head\_b)
 if head\_a\_len > head\_b\_len:
 head\_a = forward\_long\_list(head\_a\_len, head\_b\_len, head\_a)
 else:
 head\_b = forward\_long\_list(head\_a\_len, head\_b\_len, head\_b)
 while head\_a and head\_b:
 if head\_a == head\_b:
 return head\_a
 head\_a = head\_a.next
 head\_b = head\_b.next
 return None

 return main\_func(head\_a, head\_b)

#### 127.用队列实现栈。

class Stack:
 def \_\_init\_\_(self):
 self.queue\_a = []
 self.queue\_b = []

 def push(self, val):
 self.queue\_a.append(val)

 def pop(self):
 if len(self.queue\_a) == 0:
 return None
 # 核心思想就是留一个元素在队列a中，最后交换队列a,b，再从b中取栈顶元素
 while len(self.queue\_a) != 1:
 self.queue\_b.append(self.queue\_a.pop(0))
 self.queue\_a, self.queue\_b = self.queue\_b, self.queue\_a
 return self.queue\_b.pop()

test\_stack = Stack()
for i in range(5):
 test\_stack.push(i)
for i in range(5):
 print(test\_stack.pop())

#### 128.找出数据流的中位数。

# 中位数是有序列表中间的数。如果列表长度是偶数，中位数则是中间两个数的平均值
# 如果是奇数，则中位数就是中间的数字
def find\_zhongwei(arr):
 arr.sort()
 if len(ar) == 0:
 return
 if len(ar) == 1:
 return ar[0]
 mid = len(arr) // 2
 if len(arr) % 2 == 0:
 return (arr[mid - 1] + arr[len(arr) / 2]) / 2
 else:
 return arr[mid]

if \_\_name\_\_ == '\_\_main\_\_':
 ar = [2, 1, 5, 3, 4]
 print(find\_zhongwei(ar))

#### 129.二叉搜索树中第 K 小的元素。

# 问题本质：对二叉树进行中序遍历，中序排序后，返回第K-1个值
class Solution(object):
 def kthSmallest(self, root, k):
 """
 :type root: TreeNode
 :type k: int
 :rtype: int
 """
 def inorderTraversal(root):
 if root is None:
 return []
 res = []
 res.extend(inorderTraversal(root.left))
 res.append(root.val)
 res.extend(inorderTraversal(root.right))
 return res
 return inorderTraversal(root)[k - 1]

### **爬虫相关**

#### 130.在 requests 模块中，requests.content 和 requests.text 什么区别？

resp.text返回的是Unicode型的数据。

resp.content返回的是bytes型也就是二进制的数据。
也就是说，如果你想取文本，可以通过r.text。

如果想取图片，文件，则可以通过r.content。

（resp.json()返回的是json格式数据）

#### 131.简要写一下 lxml 模块的使用方法框架。

Python 标准库中自带了 xml 模块，但是性能不够好，而且缺乏一些人性化的 API，相比之下，第三方库 lxml 是用 Cython 实现的，而且增加了很多实用的功能，可谓爬虫处理网页数据的一件利器。lxml 大部分功能都存在 lxml.etree中.
可以看一下官方文档，https://lxml.de/index.html

#### 132.说一说 scrapy 的工作流程。



#### 133.简述 scrapy 的去重原理。

1.需要将dont\_filter设置为False开启去重，默认是False，开启去重；

2.对于每一个url的请求，调度器都会根据请求得相关信息加密得到一个指纹信息，并且将指纹信息和set()集合中的指纹信息进行比对，如果set()集合中已经存在这个数据，就不在将这个Request放入队列中。如果set()集合中没有存在这个加密后的数据，就将这个Request对象放入队列中，等待被调度

#### 134.scrapy 中间件有几种类，你用过哪些中间件？

scrapy自带两种中间件，下载器中间件（DownloaderMiddleware）,爬虫中间件（spiderMiddleware）
DownloaderMiddleware的作用是在request之前或者response之后，对spider进行配置处理，例如动态更换ip，更换user-agent,更换cookie等
Spider中间件是介入到Scrapy的spider处理机制的钩子框架，您可以添加代码来处理发送给 Spiders 的response及spider产生的item和request。

#### 135.你写爬虫的时候都遇到过什么反爬虫措施，你是怎么解决的？

可以看看这篇博文https://blog.csdn.net/weixin\_33768481/article/details/87273454

#### 136.为什么会用到代理？

单一ip频繁重复请求同一个网站会被封掉

#### 137.代理失效了怎么处理？

构建代理池，动态更换ip

#### 138.列出你知道 header 的内容以及信息。

user-agent
referer
content-type
content-length
.....
详情可以看这篇https://kb.cnblogs.com/page/92320/

#### 139.说一说如何打开浏览器访问 www.baidu.com 获取到结果，简述整个流程。

可参考这篇博文https://www.jianshu.com/p/d616d887953a

#### 140.爬取速度过快时，出现了验证码怎么处理？

比较简单的验证码，可以用Python的PIL库（from PIL import Image）,tesserocr模块；
比较复杂的话可以引入机器学习模型，但是成本会比较高，最好还是使用高质量的代理ip，避免触发验证码。

#### 141.scrapy 和 scrapy-redis 有什么区别？为什么选择 redis 数据库？

1) scrapy是一个Python爬虫框架，爬取效率极高，具有高度定制性，但是不支持分布式。而scrapy-redis一套基于redis数据库、运行在scrapy框架之上的组件，可以让scrapy支持分布式策略，Slaver端共享Master端redis数据库里的item队列、请求队列和请求指纹集合。

2) 为什么选择redis数据库，因为redis支持主从同步，而且数据都是缓存在内存中的，所以基于redis的分布式爬虫，对请求和数据的高频读取效率非常高。

#### 142.分布式爬虫主要解决什么问题？

1)ip

2)带宽

3）cpu

4）io

#### 143.写爬虫是用多进程好还是多线程好？ 为什么？

IO密集型代码(文件处理、网络爬虫等)，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，造成不必要的时间浪费，而开启多线程能在线程A等待时，自动切换到线程B，可以不浪费CPU的资源，从而能提升程序执行效率)。在实际的数据采集过程中，既考虑网速和响应的问题，也需要考虑自身机器的硬件情况，来设置多进程或多线程

#### 144.解析网页的解析器使用最多的是哪几个？

xpath,css-selector,beautifulSoup

#### 145.需要登录的网页，如何在不使用动态爬取的情况下解决 ip、cookie 和 session（其中有一些是动态生成的）的同时限制？

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，或者换用其他平台（比如手机端）看看是否可以获取相应的json文件

#### 146.验证码的解决。

图形验证码：干扰、杂色不是特别多的图片可以使用开源库Tesseract进行识别，太过复杂的需要借助第三方打码平台

点击和拖动滑块验证码可以借助selenium、无图形界面浏览器（chromedirver或者phantomjs）和pillow包来模拟人的点击和滑动操作，pillow可以根据色差识别需要滑动的位置

#### 147.使用的最多的数据库（mysql，mongodb，redis 等）有哪些？并简述对它的理解？

这个自由发挥了，多看下MySQL和MongoDB的使用，了解Redis的基本数据结构

### **网络编程**

#### 148.TCP 和 UDP 的区别有哪些？

TCP协议和UDP协议特性区别总结：
 1. TCP协议在传送数据段的时候要给段标号；UDP协议不
 2. TCP协议可靠；UDP协议不可靠
 3. TCP协议是面向连接；UDP协议采用无连接
 4. TCP协议负载较高，采用虚电路；UDP采用无连接
 5. TCP协议的发送方要确认接收方是否收到数据段（3次握手协议）
 6. TCP协议采用窗口技术和流控制

#### 149.简要介绍三次握手和四次挥手。

https://mp.weixin.qq.com/s/jLkhjM7wOpZuWgJdAXis1A
这篇博文讲的不错，可以参考一下

另外，time\_wait状态的作用在面试中也是常问的，可以参考下面这篇，https://www.iteblog.com/archives/169.html

#### 150.什么是粘包？ socket 中造成粘包的原因是什么？ 哪些情况会发生粘包现象？

基于TCP的socket编程中，发送端为了将多个发往接收端的包，更有效的发到对方，使用了优化方法（Nagle算法），将多次间隔较小、数据量小的数据包，合并成一个大的数据包发送(把发送端的缓冲区填满一次性发送)

造成粘包的原因：
1 发送端需要等缓冲区满才发送出去，造成粘包
2 接收方不及时接收缓冲区的包，造成多个包接收

### **并发**

#### 151.举例说明 conccurent.future 的中线程池的用法。

concurrent.futures模块的基础是Exectuor，Executor是一个抽象类，它不能被直接使用。但是它提供的两个子类ThreadPoolExecutor和ProcessPoolExecutor却是非常有用，顾名思义两者分别被用来创建线程池和进程池的代码。我们可以将相应的tasks直接放入线程池/进程池，不需要维护Queue来操心死锁的问题，线程池/进程池会自动帮我们调度。
example1.py

from
concurrent.futures
import
ThreadPoolExecutor

import
time

def
return\_future\_result(message):

 time.sleep(2)

 return
message

pool
=
ThreadPoolExecutor(max\_workers=2) #
 创建一个最大可容纳2个task的线程池

future1
=
pool.submit(return\_future\_result,
("hello")) #
 往线程池里面加入一个task

future2
=
pool.submit(return\_future\_result,
("world")) #
 往线程池里面加入一个task

print(future1.done()) #
 判断task1是否结束

time.sleep(3)

print(future2.done()) #
 判断task2是否结束

print(future1.result()) #
 查看task1返回的结果

print(future2.result()) #
 查看task2返回的结果

#### 152.说一说多线程，多进程 和 协程 的区别。

这个问题虽然被问烂了，但是还是可能会问，不过网上都是答案，搜搜看下就行

#### 153.简述 GIL。

在CPython解释器中，全局解释锁GIL是在于执行Python字节码时为了保护访问Python对象而阻止多个线程执行的一把互斥锁。这把锁的存在在主要是因为CPython解释器的内存管理不是线程安全的。然而直到今天GIL依旧存在，现在的很多功能已经习惯于依赖它作为执行的保证
由于GIL的存在，当线程被操作系统唤醒后，必须拿到GIL锁后才能执行代码，也就是说同一时刻永远只有一个线程在执行，这就导致如果我们的程序是CPU密集运算型的任务，那么使用Python多线程是不能提高效率的

#### 154.进程之间如何通信？

共享内存，信号量，互斥锁，消息队列等

#### 155.IO 多路复用的作用？

I/O多路复用就通过一种机制，可以监视多个描述符，一旦某个描述符就绪（一般是读就绪或者写就绪），能够通知程序进行相应的读写操作。但select，poll，epoll本质上都是同步I/O，因为他们都需要在读写事件就绪后自己负责进行读写，也就是说这个读写过程是阻塞的，而异步I/O则无需自己负责进行读写，异步I/O的实现会负责把数据从内核拷贝到用户空间

#### 156.select、poll 和 epoll 模型的区别。

可参考这篇博文，https://www.cnblogs.com/Anker/p/3265058.html

#### 157.什么是并发和并行？

并发的实质是一个物理CPU(也可以多个物理CPU) 在若干道程序之间多路复用，并发性是对有限物理资源强制行使多用户共享以提高效率
并行”指两个或两个以上事件或活动在同一时刻发生。在多道程序环境下，并行性使多个程序同一时刻可在不同CPU上同时执行

#### 158.解释什么是异步非阻塞？

当一个异步过程调用发出后，调用者不会立刻得到结果。
实际处理这个调用的部件是在调用发出后，
通过状态、通知来通知调用者，或通过回调函数处理这个调用
非阻塞的意思是，不能立刻得到结果之前，该函数不会阻塞当前线程，而会立刻返回

#### 159.简述 threading.local 的作用。

Python提供了 threading.local 类，将这个类实例化得到一个全局对象，但是不同的线程使用这个对象存储的数据其它线程不可见(本质上就是不同的线程使用这个对象时为其创建一个独立的字典)

### **Git 面试题**

#### 160.说说你知道的 git 命令。

git clone;
git push
git status;
git commit;
...

#### 161.git 如何查看某次提交修改的内容？

知道commit id的情况下:
1. 获取commit id
 git log

2. 查看commit内容
 git show commit\_id

查看最近n次提交的修改
 git log -p -n
指定n为1则可以查看最近一次修改的内容