# 目 录

[目 录 1](#_Toc27582)

[第一阶 入门 13](#_Toc22992)

[第一周 13](#_Toc9751)

[1.1程序设计的基本方法 13](#_Toc14719)

[IPO 13](#_Toc9085)

[编程解决问题的步骤 13](#_Toc4629)

[求解计算问题的精简步骤 13](#_Toc13572)

[1.3 实例1：温度转换 13](#_Toc25277)

[1.4 python基本语法元素 13](#_Toc11764)

[第二周 13](#_Toc18268)

[2.1 深入理解Python语言 13](#_Toc24936)

[C语言 14](#_Toc6699)

[Java语言 14](#_Toc28186)

[C++语言 14](#_Toc32470)

[VB语言 14](#_Toc20857)

[Python语言 14](#_Toc3000)

[编程语言的种类 14](#_Toc17526)

[机器语言 14](#_Toc10744)

[汇编语言 14](#_Toc14252)

[高级语言 14](#_Toc19346)

[超级语言 15](#_Toc20105)

[2.3模块1：turtle库的使用 15](#_Toc10875)

[Python计算生态=标准库+ 第三方库 15](#_Toc24286)

[turtle库 15](#_Toc3325)

[2.4 turtle程序语法元素分析 15](#_Toc29946)

[库引用: 15](#_Toc10598)

[turtle画笔控制函数 16](#_Toc15532)

[turtle运动控制函数 16](#_Toc20521)

[turtle方向控制函数 16](#_Toc20725)

[循环语句与函数range()： 16](#_Toc29496)

[第三周 16](#_Toc16662)

[3.1数字类型 16](#_Toc22584)

[浮点数运算存在尾数的原因 16](#_Toc4491)

[结论： 17](#_Toc7447)

[总结 17](#_Toc30933)

[3.3 字符串类型及操作 17](#_Toc15268)

[字符串类型的表示： 17](#_Toc15758)

[字符串操作符： 17](#_Toc3747)

[字符串处理函数： 17](#_Toc30353)

[字符串处理方法： 17](#_Toc518)

[字符串类型的格式化: 17](#_Toc27149)

[3.4 time库的使用 17](#_Toc26725)

[时间获取： 17](#_Toc16019)

[时间格式化： 18](#_Toc278)

[程序计时： 18](#_Toc88)

[进阶： 18](#_Toc11365)

[产生浮点数时间： 18](#_Toc28725)

[产生struct\_time时间： 18](#_Toc15501)

[产生字符串时间： 18](#_Toc287)

[第四周 控制结构 18](#_Toc9163)

[4.1 程序的分支结构 18](#_Toc25358)

[操作符： 18](#_Toc1569)

[保留字(用于条件组合的三个)： 18](#_Toc13886)

[小结： 18](#_Toc26136)

[4.3程序的循环结构 18](#_Toc12650)

[小结： 19](#_Toc11398)

[4.4 模块3：random库的使用 19](#_Toc6895)

[需要掌握的能力 19](#_Toc2987)

[基本随机数函数： 19](#_Toc13308)

[扩展随机数函数： 19](#_Toc41)

[第五周 函数 19](#_Toc28211)

[5.1函数的定义和使用（参数：可选参数{赋初值}，可变参数、返回值：0个，1个，多个） 19](#_Toc1683)

[参数传递的两种方式: 19](#_Toc18282)

[局部变量和全局变量 19](#_Toc4782)

[小结： 20](#_Toc17524)

[5.7实例7：七段数码管绘制 20](#_Toc32162)

[理解方法思维 20](#_Toc2829)

[小结： 20](#_Toc2717)

[第六周 组合数据类型 20](#_Toc23656)

[6.1 集合类型及操作 20](#_Toc31986)

[定义： 20](#_Toc24408)

[操作： 20](#_Toc15160)

[函数和方法： 20](#_Toc5516)

[应用场景： 20](#_Toc4433)

[6.2 序列类型及操作 21](#_Toc21474)

[定义: 21](#_Toc12055)

[操作: 21](#_Toc21037)

[应用场景: 21](#_Toc12921)

[6.4 字典类型及操作 21](#_Toc15836)

[定义： 21](#_Toc15374)

[操作： 21](#_Toc12506)

[函数方法： 21](#_Toc21244)

[字典类型应用场景： 21](#_Toc16122)

[第七周 文件 21](#_Toc18438)

[7.1 文件的使用 22](#_Toc14655)

[-文件的使用方式： 22](#_Toc11409)

[打开模式 22](#_Toc17678)

[-文本文件&二进制文件，open()和close() 22](#_Toc32507)

[第二部分：Python基础语法（5周） 22](#_Toc29625)

[-文件内容的读取： 22](#_Toc26589)

[-数据的文件写入： 22](#_Toc9863)

[7.3 一维数据的格式化和处理 22](#_Toc10539)

[小结： 22](#_Toc31136)

[7.4 二维数据的格式化和处理 22](#_Toc7734)

[小结： 22](#_Toc27399)

[第八周 程序设计方法学 23](#_Toc8074)

[8.1 体育竞技分析 23](#_Toc4364)

[8.2 python程序设计思维 23](#_Toc12183)

[提高用户体验： 23](#_Toc23090)

[小结： 23](#_Toc26971)

[8.3 PYthon第三方库的安装 23](#_Toc27107)

[小结： 23](#_Toc3444)

[第九周 23](#_Toc26035)

[9.1 从数据处理到人工智能 24](#_Toc15569)

[小结： 24](#_Toc24029)

[9.3 从web解析到网络空间 24](#_Toc30135)

[9.4 从人机交互到艺术设计 24](#_Toc12956)

[第九周+ 全课程总结与学习展望 24](#_Toc9546)

[第一部分：Python快速入门（2周） 24](#_Toc32637)

[第三部分：Python编程思维（2周） 24](#_Toc31590)

[小结： 24](#_Toc11989)

[第二阶 面向对象 25](#_Toc2808)

[一、Python 编程新思维及实践 25](#_Toc3947)

[p0 判断第三方库可用性的一些方法 25](#_Toc749)

[p13 1.2 Python常用标准库解析 25](#_Toc901)

[正则表达式： 25](#_Toc5975)

[经典正则表达式实例 25](#_Toc26181)

[re库 26](#_Toc14524)

[P18 1.4 Python常用标准库解析（中）2 26](#_Toc10307)

[os库之路径操作 26](#_Toc30427)

[1、os.path.abspath(path) 26](#_Toc32188)

[2、os.path.normpath(path) 26](#_Toc29323)

[3、os.path.relpath(path) 26](#_Toc4982)

[4、os.path.dirname((path) 26](#_Toc8098)

[5、os.path.basename(path) 26](#_Toc10120)

[6、os.path.join(path, \*paths) 27](#_Toc24326)

[7、os.path.exists(path) 27](#_Toc3498)

[8、os.path.isfile(path) 27](#_Toc31319)

[9、os.path.isdir(path) 27](#_Toc8887)

[10、os.path.getatime(path) 27](#_Toc5584)

[11、os.path.getmtime(path) 27](#_Toc4485)

[12、os.path.getctime(path) 27](#_Toc4694)

[13、os.path.getsize(path) 27](#_Toc26824)

[os库之进程管理 28](#_Toc4368)

[os.system(command) 28](#_Toc18412)

[os库之环境参数 28](#_Toc23179)

[1、os.chdir(path) 28](#_Toc21705)

[2、os.getcwd() 28](#_Toc2225)

[3、os.getlogin() 28](#_Toc6717)

[4、os.cpu\_count() 28](#_Toc24122)

[5、os.urandom(n) 28](#_Toc26389)

[小结: 28](#_Toc20993)

[os库之路径操作： 28](#_Toc13552)

[os库之进程管理： 29](#_Toc17533)

[os库之环境参数： 29](#_Toc17018)

[P19 1.4 Python常用标准库解析（中）3 29](#_Toc392)

[filecmp 库小结： 29](#_Toc7289)

[P22 1.5 Python常用标准库解析（下）2 29](#_Toc31175)

[sys库常用参数： 29](#_Toc29741)

[sys常用函数： 29](#_Toc14326)

[sys库 小结 30](#_Toc21980)

[P23 1.5 Python常用标准库解析（下）3 30](#_Toc31196)

[getopt库 30](#_Toc10359)

[处理函数： 30](#_Toc20930)

[p37 2.2Python 第三方库解析（上）5 30](#_Toc26561)

[小结： 30](#_Toc7313)

[p41 2.4Python 第三方库解析（下）2 31](#_Toc21605)

[pipenv库小结 31](#_Toc14864)

[二、Python 面向对象语法精讲 31](#_Toc29131)

[第一周内容 31](#_Toc8147)

[Python类的构建 31](#_Toc10176)

[实力1：银行ATM等待时间分析 31](#_Toc25816)

[Python类的封装 31](#_Toc20006)

[Python类的继承 31](#_Toc4843)

[Python语法实践 31](#_Toc23359)

[第二周内容 31](#_Toc13381)

[Python类的运算 32](#_Toc29023)

[Python类的对态 32](#_Toc1792)

[实例2：图像的四则运算 32](#_Toc12028)

[Python对象的引用 32](#_Toc19803)

[Python类的高级话题 32](#_Toc31865)

[Python语法实践 32](#_Toc10462)

[面向过程和面向对象编程对比 32](#_Toc11557)

[OOP三个重要特征· 33](#_Toc8157)

[封装Encapsulation：属性和方法的抽象· 33](#_Toc13629)

[继承 Inheritance：代码复用的高级抽象 33](#_Toc10230)

[多态Polymorphism：仅针对方法，方法灵活性的抽象 33](#_Toc1268)

[常用术语概述： 33](#_Toc9582)

[类Class和对象Object 33](#_Toc22737)

[面向对象术语概述· 33](#_Toc3749)

[P11 1.2 python类的构建2 34](#_Toc4577)

[小结： 34](#_Toc7158)

[P12 1.2 python类的构建3 34](#_Toc31884)

[类的构造函数 34](#_Toc28601)

[P13 1.2 python类的构建4 34](#_Toc25366)

[类的属性 35](#_Toc16199)

[类属性： 35](#_Toc28937)

[实例属性： 35](#_Toc3377)

[定义：属性是类内部定义的变量 35](#_Toc22307)

[P14 1.2 python类的构建5 35](#_Toc17921)

[类的方法 35](#_Toc23418)

[实例方法： 35](#_Toc23620)

[类方法： 35](#_Toc8179)

[自由方法： 35](#_Toc9650)

[静态方法： 35](#_Toc31039)

[保留方法： 35](#_Toc2725)

[P15 1.2 python类的构建6 35](#_Toc17150)

[类的析构函数 35](#_Toc25154)

[定义： 36](#_Toc26886)

[\_\_del\_\_()的使用说明： 36](#_Toc27073)

[sys.getrefcount(<对象名>）获得对象的引用次数: 36](#_Toc27791)

[小结： 36](#_Toc23127)

[P20 1.4 Python类的封装2 36](#_Toc24283)

[私有类属性： 37](#_Toc5739)

[定义： 37](#_Toc21588)

[区别：私有类属性名开始需要有两个下划线(\_\_)，如\_count 37](#_Toc4345)

[私有实例属性： 37](#_Toc19296)

[定义： 37](#_Toc4314)

[区别：私有实例属性名开始需要有两个下划线（\_\_），如\_\_name 37](#_Toc20815)

[私有方法： 37](#_Toc20598)

[定义： 37](#_Toc30685)

[P23 1.4 Python类的封装5 37](#_Toc8133)

[类的保留属性 37](#_Toc26531)

[仅用<类名>访问的保留属性： 37](#_Toc17270)

[可以用<类名>和<对象名>访问的保留属性： 38](#_Toc32552)

[P24 1.4 Python类的封装6 38](#_Toc25147)

[类的保留方法 38](#_Toc11131)

[小结： 38](#_Toc6535)

[P25 1.4 Python类的封装7 39](#_Toc20076)

[单元小结： 39](#_Toc23884)

[P26 1.5 Python类的继承1 39](#_Toc26608)

[继承 Inheritance：代码复用的高级抽象· 39](#_Toc8813)

[P28 1.5Python类的继承3 39](#_Toc9002)

[类继承的构建 39](#_Toc4614)

[定义： 39](#_Toc11565)

[基类名可以带有路径：ModuleName.BaseClassName 39](#_Toc13241)

[派生类的约束： 40](#_Toc10877)

[P29 1.5 Python类的继承4 40](#_Toc12195)

[Python最基础类 40](#_Toc2512)

[object类是Python所有类的基类： 40](#_Toc3230)

[Python对象的三个要素：标识、类型和值· 40](#_Toc9634)

[2个与基础类有关的Python内置功能： 40](#_Toc13824)

[P30 1.5 Python类的继承5 40](#_Toc29588)

[类的属性重载 40](#_Toc11263)

[重载：派生类对基类属性或方法的再定义： 40](#_Toc14263)

[最近覆盖原则：重载无需特殊标记· 40](#_Toc7249)

[P31 1.5 Python类的继承6 40](#_Toc9005)

[类的方法重载 40](#_Toc19660)

[定义： 41](#_Toc16579)

[完全重载： 41](#_Toc20500)

[增量重载： 41](#_Toc26484)

[P32 1.5 Python类的继承7 41](#_Toc31887)

[多继承的构建： 41](#_Toc27127)

[类多继承的使用说明: 41](#_Toc8259)

[P33 1.5 Python类的继承8 41](#_Toc13252)

[单元小结： 41](#_Toc66)

[P36 2.1 Python类的运算1 41](#_Toc18228)

[1，运算的理解 42](#_Toc26830)

[2，算术运算的重载 42](#_Toc28068)

[3，比较运算的重载 42](#_Toc31588)

[4，其他运算的重载 42](#_Toc4680)

[5，成员运算的重载 42](#_Toc15782)

[P37 2.1 Python类的运算2 42](#_Toc3404)

[运算重载的限制： 42](#_Toc16525)

[P38 2.1 Python类的运算3 42](#_Toc14816)

[算术运算符的重载 42](#_Toc26511)

[算术运算的种类· 42](#_Toc22716)

[算术运算符的重载：一元算术运算符: 42](#_Toc252)

[算术运算符的重载：二元算术运算符: 42](#_Toc5524)

[P39 2.1 Python类的运算4 43](#_Toc28087)

[比较运算符的重载 43](#_Toc21982)

[比较运算： 43](#_Toc26785)

[比较运算符的重载: 43](#_Toc947)

[P40 2.1 Python类的运算5 43](#_Toc11674)

[成员运算的重载 43](#_Toc7933)

[成员运算的种类· 43](#_Toc7230)

[成员运算符的重载：成员获取 43](#_Toc5170)

[P41 2.1 Python类的运算6 43](#_Toc29024)

[其他运算的重载 44](#_Toc3624)

[小结： 44](#_Toc20617)

[P42 2.1 Python类的运算7 44](#_Toc24101)

[小结： 44](#_Toc6725)

[P43 2.2 Python类的多态1 44](#_Toc19836)

[1，对态的理解 44](#_Toc31170)

[2，参数类型的多态 44](#_Toc27939)

[3，参数形式的多态 44](#_Toc3224)

[P44 2.2 Python类的多态2 44](#_Toc23266)

[多态的理解 44](#_Toc23991)

[P45 2.2 Python类的多态3 44](#_Toc28939)

[参数类型的多态 44](#_Toc29895)

[P46 2.2 Python类的多态4 45](#_Toc15577)

[参数形式的多态 45](#_Toc14571)

[P47 2.2 Python类的多态5 45](#_Toc26302)

[单元小结： 45](#_Toc29814)

[P50 2.4 Python对象的引用1 45](#_Toc30571)

[1，引用的理解 45](#_Toc21792)

[2，浅拷贝和深拷贝 45](#_Toc1903)

[3，类的方法引用 45](#_Toc20648)

[P51 2.4 Python对象的引用2 45](#_Toc19770)

[引用的理解 45](#_Toc9407)

[引用Reference：对象的指针： 45](#_Toc29874)

[不可变对象的引用：字符串 45](#_Toc1729)

[不可变对象的引用：字符串 46](#_Toc31372)

[可变对象引用：列表 46](#_Toc21044)

[导致引用+1的情况： 46](#_Toc24400)

[导致引用-1的情况· 46](#_Toc13362)

[小结： 46](#_Toc22426)

[P52 2.4 Python对象的引用3 46](#_Toc809)

[浅拷贝和深拷贝 46](#_Toc2906)

[深拷贝： 46](#_Toc11670)

[小结： 46](#_Toc17443)

[P53 2.4 Python对象的引用4 47](#_Toc16298)

[类的方法引用 47](#_Toc10944)

[再看类的实例方法： 47](#_Toc16115)

[P54 2.4 Python对象的引用5 47](#_Toc30378)

[单元小结： 47](#_Toc21638)

[P55 2.5 Python类的高级话题1 47](#_Toc17950)

[P56 2.5 Python类的高级话题2 47](#_Toc16700)

[命名空间的理解 47](#_Toc19422)

[命名空间Namespace：从名字到对象的一种映射： 47](#_Toc13560)

[命名空间Namespace：从名字到对象的一种映射: 48](#_Toc10522)

[P57 2.5 Python类的高级话题3 48](#_Toc10842)

[类的特性装饰器 48](#_Toc29419)

[@property:类的特性装饰器: 48](#_Toc20110)

[P58 2.5 Python类的高级话题4 48](#_Toc2602)

[自定义的异常类型 48](#_Toc8955)

[异常Exception 也是一种Python类： 48](#_Toc4108)

[P59 2.5 Python类的高级话题5 48](#_Toc4861)

[类的名称修饰 48](#_Toc6576)

[名称修饰 Name Mangling：类中名称的变换约定: 48](#_Toc19761)

[1，\_X:单下划线开头的名称修饰: 48](#_Toc14675)

[2，X\_：单下划线结尾的名称修饰· 48](#_Toc2218)

[3，\_\_X:双下划线开头的名称修饰: 49](#_Toc10335)

[4，\_\_X\_\_:双下划线开头和结尾的名称修饰 49](#_Toc1352)

[5，\_:单下划线 49](#_Toc19168)

[小结： 49](#_Toc29644)

[P60 2.5 Python类的高级话题6 49](#_Toc8155)

[Python最小空类 49](#_Toc11109)

[定义： 49](#_Toc3350)

[Python最小空类的作用: 49](#_Toc201)

[P61 2.5 Python类的高级话题7 49](#_Toc29479)

[单元小结： 49](#_Toc3812)

[三、Python 计算生态构建 50](#_Toc17762)

[P1 1.课程基本情况 50](#_Toc32438)

[课程目标 50](#_Toc3711)

[第1周内容结构 50](#_Toc30025)

[1，深入理解Python库· 50](#_Toc25131)

[2，模块和包的构建· 50](#_Toc31426)

[3，实例1：矩阵乘法模块的构建· 50](#_Toc25623)

[4，深入理解import系统· 50](#_Toc22591)

[5，Python第三方库的发布· 50](#_Toc2881)

[6，Python语法实践 51](#_Toc9212)

[第2周内容 51](#_Toc25591)

[1，C语言精简概述 51](#_Toc26417)

[2，Python与C的交互方法 51](#_Toc12019)

[3，Python扩展的CFFI方法 51](#_Toc21128)

[4，实例2：矩阵乘法的C语言加速 51](#_Toc27053)

[5，Python语法实践 51](#_Toc1862)

[P2 1.0 第1周内容导学 51](#_Toc2897)

[第一周学习目标 51](#_Toc5853)

[P5 1.1 深入理解Python库·33 51](#_Toc25245)

[模块的命名空间 51](#_Toc21473)

[模块是一个命名空间： 51](#_Toc10155)

[模块是一个命名空间： 52](#_Toc27278)

[P6 1.1 深入理解Python库4· 52](#_Toc31294)

[包的命名空间 52](#_Toc28176)

[包是一个有层次的命名空间： 52](#_Toc26151)

[P7 1.1 深入理解Python库5· 52](#_Toc3759)

[模块的名称属性 52](#_Toc99)

[名称属性：表达模块名称的预定义变量： 52](#_Toc20558)

[if \_\_name\_\_=='\_\_main\_\_: 52](#_Toc31906)

[P8 1.1 深入理解Python库6 52](#_Toc16855)

[单元小结： 52](#_Toc30804)

[P9 1.2 模块和包的构建1 52](#_Toc21627)

[1，模块的构建 53](#_Toc19297)

[2，常规包的构建· 53](#_Toc26129)

[3，命名空间包的构建 53](#_Toc29575)

[P10 1.2 模块和包的构建2 53](#_Toc24451)

[模块的构建 53](#_Toc6057)

[模块的构建原则：如何编写好一个.py文件？ 53](#_Toc3928)

[模块的构建原则· 53](#_Toc19095)

[P11 1.2 模块和包的构建3 53](#_Toc27106)

[常规包的构建· 53](#_Toc12569)

[包有两种：常规包和命名空间包: 53](#_Toc25503)

[常规包 Regular Packages·连续目录空间表达的、有层次的命名空间: 53](#_Toc3388)

[\_\_all\_\_属性： 53](#_Toc32708)

[常规包的构建流程： 53](#_Toc317)

[常规包的导入流程（import）： 54](#_Toc15381)

[命名空间包的构建 54](#_Toc11899)

[命名空间包：表达命名空间层次结构的一种逻辑包形式： 54](#_Toc3179)

[sys.path:指定搜索路径的字符串列表： 54](#_Toc14687)

[\_\_path\_\_属性： 54](#_Toc27558)

[P13 1.2 模块和包的构建5 54](#_Toc16666)

[单元小结: 54](#_Toc29906)

[P16 1.4 深入理解import系统1 55](#_Toc4768)

[单元开篇: 55](#_Toc20165)

[P17 1.4 深入理解import系统2 55](#_Toc22615)

[import系统· 55](#_Toc17808)

[import系统：扩展命名空间及功能的方法： 55](#_Toc21576)

[import系统：扩展命名空间及功能的方法: 55](#_Toc2153)

[模块对象 55](#_Toc24794)

[定义： 55](#_Toc30718)

[P18 1.4 深入理解import系统3 55](#_Toc5211)

[模块查找 55](#_Toc4181)

[查找的开始· 55](#_Toc1511)

[查找路径：1 56](#_Toc4764)

[查找策略：2-4 56](#_Toc26221)

[P19 1.4 深入理解import系统4 56](#_Toc30365)

[模块的加载· 56](#_Toc30424)

[建立模块对象的过程： 56](#_Toc10417)

[建立模块对象的过程： 56](#_Toc27830)

[模块对象的命名空间: 56](#_Toc102)

[P20 1.4 深入理解import系统5 56](#_Toc8395)

[import使用方式 56](#_Toc32050)

[import的三种使用方式： 57](#_Toc32617)

[import <模块名>： 57](#_Toc493)

[from<模块名>import<类/函数/\*>： 57](#_Toc12440)

[import<模块名>as<别名>： 57](#_Toc19685)

[P21 1.4 深入理解import系统6 57](#_Toc24130)

[单元小结 57](#_Toc6603)

[P22 1.5Python第三方库的发布1 57](#_Toc28934)

[单元开篇 57](#_Toc966)

[P23 1.5Python第三方库的发布2 57](#_Toc8286)

[发布前的准备 58](#_Toc12663)

[Python第三方库发布: 58](#_Toc22846)

[一些基本概念· 58](#_Toc25002)

[P24 1.5Python第三方库的发布3 58](#_Toc26380)

[第三方库的发布流程 58](#_Toc24447)

[第一步：整理目录结构 58](#_Toc21026)

[第二步：创建其他相关文件 58](#_Toc4721)

[第三步：执行打包命令 58](#_Toc25620)

[第四步:执行发布命令 58](#_Toc31650)

[整体步骤： 59](#_Toc19319)

[P25 1.5Python第三方库的发布4 59](#_Toc26341)

[第三方库发布概念 59](#_Toc18563)

[与发布Distribution相关的概念： 59](#_Toc18915)

[与发布Distribution相关的概念逻辑： 59](#_Toc19786)

[与打包格式相关的概念： 59](#_Toc32244)

[P26 1.5Python第三方库的发布5 59](#_Toc1800)

[单元小结： 59](#_Toc30597)

[四、Python精进路线展望 59](#_Toc11283)

[课程目标 59](#_Toc16584)

[第1周内容 60](#_Toc24129)

[精进的Python语法： 60](#_Toc23289)

[精进的Python语言： 60](#_Toc28053)

[精进的Python计算生态： 60](#_Toc17128)

[实例1：基于OpenCV的人脸识别： 60](#_Toc25899)

[Python学习路线展望： 60](#_Toc31390)

[P3 1.1精进的Python语法1 60](#_Toc23744)

[1，上下文管理器 60](#_Toc18775)

[2，迭代器类型 60](#_Toc11306)

[3，生成器设计 60](#_Toc15235)

[4，异步协程函数 60](#_Toc1302)

[5，装饰器与语法糖 60](#_Toc18739)

[P4 1.1精进的Python语法2 60](#_Toc17228)

[上下文管理器 60](#_Toc18147)

[上下文Context：程序执行中某个状态： 60](#_Toc22251)

[上下文管理器：一个可以在程序中加载独立上下文的对象 61](#_Toc17694)

[定义： 61](#_Toc5008)

[如何使用上下文管理器 ： 61](#_Toc16714)

[上下文管理器：一个可以在程序中加载独立上下文的对象： 61](#_Toc97)

[P5 1.1精进的Python语法3 61](#_Toc28666)

[迭代器类型 61](#_Toc18818)

[迭代器类型 lterator Types：实现容器中元素的迭代： 61](#_Toc30497)

[迭代器类型 lterator Types：实现容器中元素的迭代: 62](#_Toc26128)

[迭代器类型Iterator Types:实现容器中元素的迭代: 62](#_Toc32400)

[P6 1.1精进的Python语法4 62](#_Toc24342)

[生成器设计 62](#_Toc29088)

[生成器Generator：快速实现迭代器类型的方式： 62](#_Toc25341)

[生成器函数Generator Function：实现迭代逻辑的函数： 62](#_Toc12103)

[生成器函数的设计原则： 62](#_Toc19429)

[P7 1.1精进的Python语法5 62](#_Toc3590)

[装饰器与语法糖 62](#_Toc27235)

[装饰器 Decorator：以函数为参数实现其额外功能更新： 62](#_Toc27903)

[装饰器Decorator:以函数为参数实现器额外功能更新： 63](#_Toc24132)

[语法糖Syntactic Sugar:方便使用但无实质功能的语法： 63](#_Toc7832)

[装饰器设计原理： 63](#_Toc25310)

[装饰器使用场景： 63](#_Toc311)

[P8 1.1精进的Python语法6 63](#_Toc13279)

[异步协程函数 63](#_Toc17730)

[协程Coroutine：在程序层面的多段代码异步执行： 63](#_Toc23994)

[async的用法： 64](#_Toc5215)

[await的用法： 64](#_Toc11149)

[P9 1.1精进的Python语法7 64](#_Toc25059)

[单元小结： 64](#_Toc18176)

[P10 1.2精进的Python语言1 64](#_Toc31872)

[1，Python保留字 64](#_Toc30186)

[2，Python特殊字符 64](#_Toc8860)

[3，Python内置函数 64](#_Toc23968)

[P11 1.2精进的Python语言2 64](#_Toc1402)

[Python保留字： 64](#_Toc28335)

[P12 1.2精进的Python语言3 67](#_Toc3700)

[Python特殊符号 67](#_Toc25574)

[特殊符号：表达Python语法的三种逻辑： 67](#_Toc12415)

[操作符· 67](#_Toc25181)

[分界符: 67](#_Toc19062)

[特殊含义 67](#_Toc2814)

[Python语法不使用的符号 68](#_Toc18713)

[P13 1.2精进的Python语言4 68](#_Toc4388)

[Python内置函数：69个 68](#_Toc28065)

[P14 1.2精进的Python语言5 68](#_Toc18239)

[单元小结： 68](#_Toc1623)

[P15 1.3精进的Python计算生态1 68](#_Toc22953)

[1，Python计算生态概述 68](#_Toc4248)

[2，数据分析到人工智能 68](#_Toc14367)

[3，Web解析到网络空间 68](#_Toc262)

[4，人机交互到艺术设计 68](#_Toc1434)

[P23 1.5 Python学习路线展望1 68](#_Toc27895)

[1，Python知识体系 69](#_Toc24262)

[2，编程小白的学习路线 69](#_Toc15378)

[3，专业求职的学习路线 69](#_Toc19635)

[4，专业择业的学习路线 69](#_Toc7980)

[5，专业小牛的精进路线 69](#_Toc24798)

# 第一阶 入门

# 第一周

## 1.1程序设计的基本方法

### IPO

-I Input输入，程序的输入

-P Process处理，程序的主要逻辑

-O Output输出，程序的输出

### 编程解决问题的步骤

-分析问题：分析问题的计算部分，想清楚

-划分边界：划分问题的功能边界，规划IPO

-设计算法：设计问题的求解算法，关注算法

-编写程序；编写问题的计算程序，编程序

-调试测试：调试程序使正确运行，运行调试

-升级维护：适应问题的升级维护，更新完善

### 求解计算问题的精简步骤

-确定IPO：明确计算部分及功能边界

-编写程序：将计算求解的设计变成现实

-调试程序：确定程序按照正确逻辑能够正确运行

## 1.3 实例1：温度转换

问题理解：

理解1：直接将温度值进行转换

理解2：将温度信息发布的声音和图像进行理解和转换

理解3：监控温度信息发布渠道，实时获取并转换温度值

## 1.4 python基本语法元素

-缩进、注释、命名、变量、保留字

-数据类型、字符串、整数、浮点数、列表:[1,2]

-赋值语句、分支语句、函数

-input()、print()、eval()、print()格式化

# 第二周

## 2.1 深入理解Python语言

### C语言

C语言更关心计算机系统结构

学习内容：指针、内存、数据类型

语言本质：理解计算机系统结构

解决问题：性能

### Java语言

一台计算机编写的程序能否在其他计算机上运行，所以Java语言诞生了

学习内容：对象、跨平台、运行时

语言本质：理解主客体关系

解决问题：跨平台

### C++语言

学习内容：对象、多态、继承

语言本质：理解主客体关系

解决问题：大规模程序

### VB语言

学习内容：对象、按钮、文本框

语言本质：理解交互逻辑（不是程序之间的交互，而是人机之间的交互）

解决问题：桌面应用

### Python语言

学习内容：编程逻辑、第三方库

语言本质：理解问题求解

解决问题：各类问题

### 编程语言的种类

### 机器语言

一种二进制语言，直接使用二进制代码表达指令

计算机硬件（cpu）可以直接执行，与具体CPU型号有关（CPU的参考手册）

### 汇编语言

一种将二进制代码直接对应助记符的编程语言

汇编语言由于与二进制代码（机器语言）一一对应的，所以它也与具体的CPU型号有关

汇编语言与CPU信号有关，程序不通用，需要汇编器转换

### 高级语言

高级语言代码与具体的CPU型号无关，编译后运行

### 超级语言

Python 唯一的超级语言

## 2.3模块1：turtle库的使用

### Python计算生态=标准库+ 第三方库

-标准库：随解释器直接安装到操作系统中的功能模块

-第三方库；需要经过安装才能使用的功能模块

-库Library、包Package、模块Module,统称模块

### turtle库

空间坐标体系：

1、绝对坐标：中间位置坐标为（0,0）：

goto()

2、海龟坐标：海龟的视角:

turtle.fd()

turtle.bk()

turtle.circle()

角度坐标体系:

1、绝对角度坐标:

turtle.seth()

2、海龟角度坐标

turtle.left(angle)

turtle.right(angle)

## 2.4 turtle程序语法元素分析

### 库引用:

第一种

import <库名>

<库名>.<函数名>（函数参数）

第二种

from <库名> import <函数名>

from <库名> import \*

<函数名>（函数参数）

-第一种方法不会出现函数重名

-第二种方法会出现

import <库名> as <库别名>

<库别名>.<函数名>（函数参数）

### turtle画笔控制函数

penup()

pendown()

pensize(width)

pencolor(colorstring)

pencolor(r,g,b)

pencolor((r,g,b))

### turtle运动控制函数

fd(d)

circle(r,extent=None)

### turtle方向控制函数

seth()

left()

right()

### 循环语句与函数range()：

for i in range(N):

range(N)

range(M,N)

# 第三周

## 3.1数字类型

1）整数

2）浮点数

3）复数

### 浮点数运算存在尾数的原因

十进制 0.1

转换（截取53位）

二进制 0.00011001180118011001100110011001100110011001100110011010（无限接近0.1）

转换

十进制 0.1000000000000000055511151231257827021181583404541015625

### 结论：

由十进制转换二进制转换十进制过程中产生了尾数

0.1+0.2

结果无限接近0.3，但可能存在尾数

### 总结

1、整数类型的无限范围及4种进制表示

2、浮点数类型的近似无限范围、小尾数及科学计数法

3、+、-、\*、/、//、%、\*\*、二元增强赋值操作符

4、计算函数：abs()、divmod()、pow()、round()、max()、min()

5、数字类型转换：int()、float()、complex()

## 3.3 字符串类型及操作

pdf：PYE3.3-字符串类型及操作v2.5

### 字符串类型的表示：

"", '',''' ''',""" """

### 字符串处理方法：

str.lower(),str.upper(),str.split(),str.count(),str.replace(old,new),str.center(width,fillchar),str.strip(chars),str.join()

### 字符串操作符：

+ , \* , in

### 字符串处理函数：

eval(x),str(),len(x),hex(x)或oct(x),chr(x),ord(x)

### 字符串类型的格式化:

"{ <参数序号>:<格式控制标记>}".format("")

## 3.4 time库的使用

### 时间获取：

time(),ctime(),gmtime()

### 时间格式化：

s1=time.strftime("%Y-%m-%d %H:%M:%S",time.localtime()),s2=time.strptime(s1,"%Y-%m-%d %H:%M:%S")

### 程序计时：

perf\_counter(),sleep()

### 进阶：

### 产生浮点数时间：

time(),mktime()

### 产生struct\_time时间：

gmtime() ps：UTC时间,localtime()

### 产生字符串时间：

ctime(<t>) t的类型为浮点数时间, asctime(<s>) s的类型是strtuct\_time时间

# 第四周 控制结构

## 4.1 程序的分支结构

### 操作符：

< , <= ,>= , > ,== , !=

### 保留字(用于条件组合的三个)：

x and y , x or y ,not x ;ps:除0之外，其他判断都是True

### 小结：

-单分支if 二分支if-else及紧凑形式:<表达式1> if <条件> else <表达式2>,ps：("对" if guess==99 else "错")

-多分支if-elif-else 及条件之间的关系

-not and or > >= == <= < !=

-异常处理 try-except-else-finally:

finally对应语句块4一定执行 :不管程序有不有错误，执行完try中的语句都要去执行finally

else对应语句块3在不发生异常时执行:如果程序在try中并没有错误，那么try执行完后就去执行else语句

## 4.3程序的循环结构

for 循环变量 in 遍历结构

语句块

-计数循环range(N)

-计数循环range(M,N,K)

-字符串遍历循环

-列表遍历循环

-文件遍历循环

循环else的高级用法：出现break打断循环时，不执行else,没有break时，执行else语句

for i in range(0,10):  
 print(i)  
 break  
else:  
 print("exit")

### 小结：

for in 遍历循环：计数、字符串、列表、文件

while无限循环

continue 和break保留字：退出当前循环层次

循环else的高级用法：与break有关

## 4.4 模块3：random库的使用

### 需要掌握的能力

-能够利用随机数种子产生“确定”伪随机数

-能够产生随机整数

-能够对序列类型进行随机操作

### 基本随机数函数：

seed(),random()

### 扩展随机数函数：

randint(10.100),getrandbits(16),uniform(M,N),randrange(M,N,K),choice([1,2,3,4,5,6,7]),shuffle([1,2,3,4,5,6,7])

# 第五周 函数

## 5.1函数的定义和使用（参数：可选参数{赋初值}，可变参数、返回值：0个，1个，多个）

### 参数传递的两种方式:

def fact(n,m=1):

<函数体>

return

方式一：位置传递：fact(10,5)

方式二：名称传递：fact(m=5,n=10)

补充：\*args,\*\*kwargs

不定长参数传递：接收任意多的位置参数

def fun2(\*a):  
 print(a) #元组  
fun2(1,1,2,3,4)

还可以：

def fun2(\*a):

    print(a)  #元组

lis=[1,2,3,4]

fun2(\*lis)

不定长参数传递：接收任意多的关键字参数

def fun(\*\*a):  
 print(a) #字典  
fun(name=123,age=123,user=123,password=123)

还可以：

def fun(\*\*a):

    print(a) #字典

lis={'name':123,'age':123,'user':123,'password':123}

fun(\*\*lis)

### 局部变量和全局变量

-基本数据类型，无论是否重名，局部变量与全局变量不同

-可以通过global保留字在函数内部声明全局变量

-组合数据类型(列表)，如果局部变量未真实创建，则是全局变量

### 小结：

（保留字def函数定义）-使用保留字def定义函数，lambda定义匿名函数

（参数）-可选参数（赋初值）、可变参数（\*b）、名称传递

（返回值）-保留字return可以返回任意多个结果

（声明全局变量-global保留字）-保留字global声明使用全局变量，一些隐式规则

## 5.7实例7：七段数码管绘制

### 理解方法思维

-模块化思维：确定模块接口，封装功能

-规则化思维：抽象过程为规则，计算机自动执行（利用参数控制了绘制哪个数字）

-化繁为简：将大功能变为小功能组合，分而治之

### 小结：

模块化设计：松耦合、紧耦合

函数递归的2个特征：基例和链条

### 第六周 组合数据类型

导学：

方法论：

集合、序列、字典的模式来考虑对一组数据的表达和处理

集合：{}

序列：字符串” ”、元组()和列表[]

字典：{}

## 6.1 集合类型及操作

-集合用大括号{}表示，元素间用逗号分隔

-集合中每个元素唯一，不存在相同元素

-集合元素之间无序

### 定义：

集合使用{}和set()函数创建

### 操作：

&，|，-，^，>=<

### 函数和方法：

add(),.discard(),.pop()等

### 应用场景：

包含关系比较，数据去重

## 6.2 序列类型及操作

### 定义:

-序列是基本类型，扩展类型包括：字符串””、元组()和列表[]

-元组用()和tuple()创建，列表用[]和list()创建

### 操作:

六个通用操作符：x in s, x not in s,s + t,s\*n,s[i],s[i,j]或s[i,j,k]

-元组操作与序列操作基本相同,因为创建后不能修改，因此没有特殊操作

-列表操作在序列操作基础上，增加了更多的灵活性（12种听写）：ls[i]=x,ls[i:j:k]=It,del ls[i],del ls[i:j:k],ls+=It,ls\*=n

### 应用场景:

-元组用于元素不改变应用场景，更多用于固定搭配场ps:用元组类型作为接口，保护数据不被修改

-列表表示一组有序数据，进而操作它们

## 6.4 字典类型及操作

### 定义：

-映射关系采用键值对表达

-字典类型使用{}和dict()创建，键值对之间用:分隔

### 操作：

-d[key] 方式既可以索引，也可以赋值

### 函数方法：

d.get(k, <default>) 键k存在，则返回相应值，不在则返回<default>值

del d[k] 删除字典d中键k对应的数据值

k in d 判断键k是否在字典d中，如果在返回True，否则False

d.keys() 返回字典d中所有的键信息

d.values() 返回字典d中所有的值信息

d.items() 返回字典d中所有的键值对信息

### 字典类型应用场景：

-映射无处不在，键值对无处不在 ps:统计数据出现的次数，数据是键，次数是值

-最主要作用：表达键值对数据，进而操作它们

# 第七周 文件

## 7.1 文件的使用

### -文件的使用方式：

打开-操作-关闭

### 打开模式

'r' 只读模式，默认值，如果文件不存在，返回FileNotFoundError

'w' 覆盖写模式，文件不存在则创建，存在则完全覆盖

'x' 创建写模式，文件不存在则创建，存在则返回FileExistsError

'a' 追加写模式，文件不存在则创建，存在则在文件最后追加内容

'b' 二进制文件模式

't' 文本文件模式，默认值

'+' 与r/w/x/a一同使用，在原功能基础上增加同时读写功能

### -文本文件&二进制文件，open()和close()

## 第二部分：Python基础语法（5周）

从5个方面讲解基础语法全体系，提供10个实例，理性学习

### -文件内容的读取：

.read() .readline() .readlines()

### -数据的文件写入：

.write() .writelines() .seek()

1.<f>.write(s)

向文件写入一个字符串或字节流

>>>f.write("中国是一个伟大的国家!")

2.<f>.writelines(lines)

将一个元素全为字符串的列表写入文件

>>>ls = ["中国", "法国",

"美国"]

>>>f.writelines(ls)

中国法国美国

3.<f>.seek(offset)

改变当前文件操作指针的位置，offset含义如下：

0 – 文件开头； 1 – 当前位置； 2 – 文件结尾

>>>f.seek(0) #回到文件开头

## 7.3 一维数据的格式化和处理

数据存储 <-> 数据表示 <-> 数据操作

存储格式 数据类型 操作方式

### 小结：

-数据的维度：一维，二维，多维（ps；二维+时间），高维（ps:键值对+关系）

表示一维数据的表示：列表类型（有序）和集合类型（无序）

存储-一维数据的存储：空格分隔、逗号分隔、特殊符号分隔

处理-一维数据的处理：字符串方法、.split()和.join()

## 7.4 二维数据的格式化和处理

### 小结：

表示-二维数据的表示：列表类型，其中每个元素也是一个列表

存储-CSV格式：逗号分隔表示一维，按行分隔表示二维

处理-二维数据的处理：for循环+.split()和.join()

Join():处理迭代类型，变为一个字符串整体，并以指定字符串分割

# 第八周 程序设计方法学

## 8.1 体育竞技分析

-理解自顶向下的设计思维：分而治之

-理解自底向上的执行思维：模块化集成

-自顶向下是“系统”思维的简化

## 8.2 python程序设计思维

### 提高用户体验：

方法一：进度条

方法二：异常处理

-当获得用户输入，对合规性需要检查时

-当读写文件时，对结果进行判断

-当进行输入输出时，对运算结果进行判断

其他类方法：

-打印输出：特定位置，输出程序运行的过程信息

-日志文件：对程序异常及用户使用进行定期记录

-帮助信息：给用户多种方式提供帮助信息

### 小结：

-计算思维：抽象计算过程和自动化执行

-计算生态：竞争发展，相互依存，快速更迭

-用户体验：进度展示，异常处理，文件日志等

-IPO、自顶向下、模块化、配置化、应用开发的四个步骤

## 8.3 PYthon第三方库的安装

### 小结：

-PyPI:Python Package Index 地址：https://pypi.org/

-pip命令的各种用法：pip -h , pip instal , pip install -U ,pip uninstall ,pip download ,pip show ,pip search , pip list ,

-Anaconda集成开发工具及安装 ps:已经集成800多第三方库，无需安装

-UCI页面的“补丁”安装方法 地址：https://www.lfd.uci.edu/~gohlke/pythonlibs/#wordcloud

# 第九周

## 9.1 从数据处理到人工智能

### 小结：

数据处理:Numpy、Pandas、Scipy

数据可视化:Matplotlib、Seaborn 、Mayavi

文本处理:PyPDF2、NLTK、python-docx

机器学习:Scikit-learn 、TensorFlow 、MXNet

## 9.3 从web解析到网络空间

爬虫方面：Requests,Scrapy,pyspider

信息解析：Beautiful Soup ,Re ，Python-Goose

网站开发：Django (专业网站框架),Pyramid （中型web框架）,Flask(微型web框架s)

有趣的：WeRobot,aip ,MyQR

## 9.4 从人机交互到艺术设计

人机交互界面：PyQt5,wxPython,PyGobject

人机交互的游戏：PyGame,Panda3D,cocos2d

人机虚拟现实交互：VR Zero ,pyovr,Vizard

艺术设计：Quads,ascii\_art,turtle

# 第九周+ 全课程总结与学习展望

## 第一部分：Python快速入门（2周）

围绕2个具体实例，讲解Python基本语法元素，感性认识

## 第三部分：Python编程思维（2周）

从方法学角度开阔认识，提升整体编程能力，展望未来

## 小结：

① 基本数据类型

- 整数、浮点数、复数

- 字符串

② 程序的控制结构

- 分支结构与异常处理

- 遍历循环、无限循环

③ 函数和代码复用

- 函数定义和使用

- 函数递归

④ 组合数据类型

- 集合类型

- 序列类型：元组和列表

- 字典类型

⑤ 文件和数据格式化

- 文件的使用

- 一二维数据的表示存储和处理

# 第二阶 面向对象

# 一、Python 编程新思维及实践

## p0 判断第三方库可用性的一些方法

地址：https://pypi.org/

-开发人员推荐

-查看开发历史，近半年有更新记录

-访问项目主页，文档齐全

-评星较高

## p13 1.2 Python常用标准库解析

### 正则表达式：

. :表示任何单个字符

[] :字符集，对单个字符给出取值范围,ps:[abcl表示a、b、c，[a-zl表示a到z单个字符

[^] :非字符集，对单个字符给出排除范围,ps:[Aabc]表示非a或b或c的单个字符

\* :前一个字符0次或无限次扩展,ps:abc\*表示ab、abc、abcc、abccc等|

+ :前一个字符1次或无限次扩展,ps:abc表示abc、abcc、abccc等？

? :前一个字符0次或1次扩展,ps:abc 表示ab、abc

| :左右表达式任意一个,ps:abc|def表示abc、def

{m} :扩展前一个字符m次，ps:ab{2}c表示abbc

{m,n} :扩展前一个字符m至n次 （含n）,ps:ab{1,2}c表示abc、abbc

^ :匹配字符串开头，ps:^abc表示abc且在一个字符串的开头

$ :匹配字符串结尾，ps:abc$表示abc且在一个字符串的结尾

() :分组标记，内部只能使用|操作符，ps:(abc)表示：abc，(abc|def)表示abc、def

\d :数字，等价于[0-9]

\w :单词字符，等价于[A-Za-Z0-9]

### 经典正则表达式实例

^[A-Za-z]+$：由26个字母组成的字符串

^[A-Za-z0-9]+$：由26个字母和数字组成的字符串

^-?\d+$：整数形式的字符串

^[O-9]\*[1-9][0-9]\*$：正整数形式的字符串

[1-9]\d{5}：中国境内邮政编码，6位

[\u4e00-\u9fa5]：匹配中文字符

\d{3}-\d{8}|\d{4}-\d{7}：国内电话号码，010-68913536

### re库

compile(pattern,flags=0)：将正则表达式的字符串形式编译成正则表达式对象

re.search():在一个字符串中搜索匹配正则表达式的第一个位置，返回match对象

re.match():从一个字符串的开始位置起匹配正则表达式，返回match对象

re.findall():搜索字符串，以列表类型返回全部能匹配的子串,返回列表类型

re.split():将一个字符串按照正则表达式匹配结果进行分割，返回列表类型

re.finditer():搜索字符串，返回一个匹配结果的迭代类型，每个选代元素是match对象，返回迭代类型

re.sub():在一个字符串中替换所有匹配正则表达式的子串，返回替换后的字符串，返回字符串类型

## P18 1.4 Python常用标准库解析（中）2

### os库之路径操作

引用：import os.path /import os.path as op

#### 1、os.path.abspath(path)

返回path在当前系统中的绝对路径

>>>os.path.abspath("file.txt")

'C:\\Users\\Tian Song\\Python36-32\\file.txt'

#### 2、os.path.normpath(path)

归一化path的表示形式，统一用\\分隔路径

>>>os.path.normpath("D://PYE//file.txt")

'D:\\PYE\\file.txt'

#### 3、os.path.relpath(path)

返回当前程序与文件之间的相对路径

>>>os.path.relpath("C://PYE//file.txt")

'..\\..\\..\\..\\..\\..\\..\\PYE\\file.txt'

#### 4、os.path.dirname((path)

返回path的目录名称

>>>os.path.dirname("D://PYE//file.txt")

'D://PYE'

#### 5、os.path.basename(path)

返回path中最后的文件名称

>>>os.path.basename("D://PYE//file.txt")

'file.txt'

#### 6、os.path.join(path, \*paths)

组合path与paths，返回一个路径字符串

>>>os.path.join("D:/", "PYE/file.txt")

'D:/PYE/file.txt'

#### 7、os.path.exists(path)

判断path对应文件或目录是否存在，返回True或False

>>>os.path.exists("D://PYE//file.txt")

False

#### 8、os.path.isfile(path)

判断path所对应是否为已存在的文件，返回True或False

>>>os.path.isfile("D://PYE//file.txt")

True

#### 9、os.path.isdir(path)

判断path所对应是否为已存在的目录，返回True或False

>>>os.path.isdir("D://PYE//file.txt")

False

#### 10、os.path.getatime(path)

返回path对应文件或目录上一次的访问时间

>>>os.path.getatime("D:/PYE/file.txt")

1518356633.7551725

#### 11、os.path.getmtime(path)

返回path对应文件或目录最近一次的修改时间

>>>os.path.getmtime("D:/PYE/file.txt")

1518356633.7551725

#### 12、os.path.getctime(path)

返回path对应文件或目录的创建时间

>>time.ctime(os.path.getctime("D:/PYE/file.txt"))

'Sun Feb 11 21:43:53 2018'

#### 13、os.path.getsize(path)

返回path对应文件的大小，以字节为单位

>>>os.path.getsize("D:/PYE/file.txt")

180768

### os库之进程管理

#### os.system(command)

### os库之环境参数

#### 1、os.chdir(path)

修改当前程序操作的路径

>>>os.chdir("D:")

#### 2、os.getcwd()

返回程序的当前路径

>>>os.getcwd()

'D:\\'

#### 3、os.getlogin()

获得当前系统登录用户名称

>>>os.getlogin()

'Tian Song'

#### 4、os.cpu\_count()

获得当前系统的CPU数量

>>>os.cpu\_count()

8

#### 5、os.urandom(n)

获得n个字节长度的随机字符串，通常用于加解密运算

>>>os.urandom(10)

b'7\xbe\xf2!\xc1=\x01gL\xb3'

### 小结:

#### os库之路径操作：

os.path.abspath(path)

os.path.normpath(path)

os.path.relpath(path)

os.path.dirname(path)

os.path.basename(path)

os.path.join(path)

os.path.exists(path)

os.path.isfile(path)

os.path.isdir(path)

os.path.getatime(path)

os.path.getmtime(path)

os.path.getctime(path)

os.path.getsize(path)

#### os库之进程管理：

os.system(command)

#### os库之环境参数：

os.chdir(path)

os.getcwd()

os.getlogin()

os.cpu\_count()

os.urandom(n)

## P19 1.4 Python常用标准库解析（中）3

### filecmp 库小结：

文件比较：cmp()、cmpfiles()

目录比较：dircmp类，其属性包括:

left,right、left list、right\_list、left\_only、right\_only、same\_files、common、diff\_files、common\_files、common\_dirs

ps:返回值为对象：比较一次，比较后的结果保存在对象里

## P22 1.5 Python常用标准库解析（下）2

### sys库常用参数：

sys.float\_info：系统浮点浮点数类型的参数信息

sys.int\_info:系统整数类型的参数信息

sys.hash\_info:系统哈希函数的参数信息

sys.thread\_info：系统线程实现的参数信息

sys.byteorder:字节序标识，大端返回“big”,小端返回“little”

sys.executable:当前解释器可执行文件的绝对路径

sys.maxunicode:系统支持的最大Unicode值，一般是1114111（0x10FFFF）

sys.path:解释器用来加载Python库的路径列表

sys.argv:命令行参数的列表，其中argv[0]表示执行程序本身

sys.dont\_write\_bytecode:如果为True，则不生成.pyc字节码

### sys常用函数：

sys.getrecursionlimit():获取系统递归最大深度

sys.setrecursionlimit(n):设置系统递归最大深度为n

sys.getdefaultencoding():获得当前默认的字符串编码名称

sys.getsizeof(object)获得object的字节长度，object可以是任意类型

sys.exit([arg]):退出Python程序，附带状态码arg

### sys库 小结

常用参数：

sys.float\_info、sys.int\_info、sys.hash\_info、sys.thread\_info、sys.byteorder、sys.executable、sys.maxunicode、sys.path、sys.argv、sys.dont\_write\_bytecode

常用函数：sys.getrecursionlimit()、sys.setrecursionlimit()、sys.getdafaultencoding()、sys.getsizeof()、sys.exit()

## P23 1.5 Python常用标准库解析（下）3

### getopt库

#### 处理函数：

getopt.getopt(args,shortopts,longopts=[])

args:

程序的命令行参数，不包括程序名称本身，一般是sys.argv[1:]

shortopts:

定义-x或-x <值>形式的短参数，带值的增加冒号:

例:"abc:d:" 表示命令行可解析-a -b -c <值> -d <值> 参数

longopts:

定义--name 或--name <值> 形式的长参数，带值加等号=

例：["name1","name2="]表示 --name1 和 --name2 <值>

函数返回两个参数：optlist和args

其中，optlist包含(option,values)组成的列表

args是未解析参数

定义异常：getopt.GetoptError（参数解析错误时反馈的异常）

## p37 2.2Python 第三方库解析（上）5

### 小结：

1，Pyinstaller 库的使用：Pyinstaller命令参数：-F，-i等

2，jieba库的使用：精准模式，全模式，搜索引擎模式

3，wordcloud库的使用：WordCloud类及参数配置

## p41 2.4Python 第三方库解析（下）2

### pipenv库小结

pipenv库：使用Pipfile和Pipfile.lock来管理虚拟环境

tabulate库：打印二维列表等数据类型

qrcode库：生成二维码

# 二、Python 面向对象语法精讲

## 第一周内容

面向对象编程模式

面向对象编程思维

面向对象的三个特征

Python面向对象术语

### Python类的构建

类的基本构建

类的属性和方法

类的构造函数和析构函数

### 实例1：银行ATM等待时间分析

对象的设计和构建

生活现象的程序分析

### Python类的封装

私有属性和公开属性

私有方法和公开方法

保留属性和保留方法

### Python类的继承

子类、父类与超类

类的方法重载和属性重载

类的多继承

### Python语法实践

## 第二周内容

### Python类的运算

运算符的理解

各类运算符的重载

### Python类的对态

多态的理解

参数类型的多态

参数形式的多态

### 实例2：图像的四则运算

PIL库和Numpy库实践

图像的加减乘除操作

### Python对象的引用

引用的理解

浅拷贝和深拷贝

### Python类的高级话题

类的特殊装饰器

命名空间的理解

类的名称修饰

### Python语法实践

一小时类运算操作实践

### 面向过程和面向对象编程对比

例：商品价格统计

有3个商品：电脑，打印机，和投影仪

每个商品有1个原始售价和1个折扣售价

求3个商品原始售价的和以及折扣售价的和

面向规程的程序解法：

列表1，存商品：电脑、打印机、投影仪

列表2，存原价：电脑原价、打印机原价、投影仪原价

列表3，存实价：电脑实价、打印机实价、投影仪实价

根据列表1、列表2和列表3进行计算，列表1/2/3功能需要文档标记

面向对象的程序解法

设计“商品”类，属性包括：商品名、原价和实价

从“商品”类产生3个对象：电脑、打印机、投影仪

这三个对象中保存商品名、原价、实价等与商品相关的信息

使用对象1、对象2和对象3进行计算

### OOP三个重要特征·

封装：属性和方法的抽象，用数据和操作数据的方法来形成对象逻辑

继承：代码复用的高级抽象，用对象之间的继承关系来形成代码复用

多态：方法灵活性的抽象，让对象的操作更加灵活、更多复用代码

#### 封装Encapsulation：属性和方法的抽象·

属性的抽象：对类的属性（变量）进行定义、隔离及保护·

方法的抽象：对类的方法（函数）进行定义、隔离及保护·

目标是形成一个类对外可操作属性和方法的接口

#### 继承 Inheritance：代码复用的高级抽象

继承是面向对象程序设计的精髓之一

实现了以类为单位的的高抽象级别代码复用

继承是新定义类能够几乎完全使用原有类属性与方法的过程

#### 多态Polymorphism：仅针对方法，方法灵活性的抽象

参数类型的多态：一个方法能够处理多个类型的能力

参数形式的多态：一个方法能够接受多个参数的能力

多态是OOP的一个传统概念，Python天然支持多态，不需要特殊语法

### 常用术语概述：

#### 类Class和对象Object

类：逻辑抽象和产生对象的模板，一组变量和函数的特定编排·

对象：具体表达数据及操作的实体，相当于程序中的“变量”

实例化：从类到对象的过程，所有“对象”都源于某个“类”

#### 面向对象术语概述·

对象：类对象、实例对象·

属性：存储数据的“变量”，包括：类属性、实例属性·

方法：操作数据的“函数"

包括：类方法、实例方法、自由方法、静态方法、保留方法

类对象V5.实例对象·

类对象：Class Object，维护每个Python类基本信息的数据结构·

实例对象：Instance Object，Python类实例后产生的对象，简称：对象·

这是一组概念，类对象全局只有一个，实例对象可以生成多个

三个特性：封装、继承、多态·

继承：基类、派生类、子类、父类、超类、重载·

命名空间：程序元素作用域的表达·

构造和析构：生成对象和删除对象的过程

## P11 1.2 python类的构建2

注：类对象内直接包含的语句被执行，因此，一般不在类定义中直接包含语句（目的为了加载属性和方法）

### 小结：

类对象Vs.实例对象·

类对象：Class Object，维护每个Python类基本信息的数据结构·

实例对象：Instance Object，Python类实例后产生的对象，简称：对象·

这是一组概念，类对象全局只有一个，实例对象可以生成多个

## P12 1.2 python类的构建3

### 类的构造函数

定义：

Python使用预定义的\_init（）作为构造函数

class <类名>：

def \_\_init\_\_(self，<参数列表>）

<语句块）

\_\_init\_\_()的使用说明：

参数：第一个参数约定是self，表示类实例自身，其他参数是实例参数·

函数名：Python解释器内部定义，由双下划线（\_\_）开始和结束·

返回值：构造函数没有返回值，或返回None，否则产生TypeError异常

self在类定义内部代表类的实例：

self是Python面向对象中约定的一个类参数·

self代表类的实例，在类内部，self用于组合访问实例相关的属性和方法·

相比，类名代表类对象本身

注：self为类内部的实例对象

## P13 1.2 python类的构建4

### 类的属性

#### 类属性：

类对象的属性，由所有实例对象所共享·

#### 实例属性：

实例对象的属性，由各实例对象所独享

#### 定义：属性是类内部定义的变量

class<类名>：

<类属性名> = <类属性初值）

def \_\_init\_\_(self,<参数列表>):

self.<实例属性名> = <实例属性初值>

## P14 1.2 python类的构建5

### 类的方法

方法是类内部定义的函数·

#### 实例方法：

实例对象的方法，由各实例对象独享，最常用的形式·

#### 类方法：

类对象的方法，由所有实例对象共享

#### 自由方法：

类中的一个普通函数，由类所在命名空间管理，类对象独享·

#### 静态方法：

类中的一个普通函数，由类对象和实例对象共享·

#### 保留方法：

由双下划线开始和结束的方法，保留使用，如\_len\_()

## P15 1.2 python类的构建6

### 类的析构函数

#### 定义：

Python使用预定义的\_\_del\_\_()作为析构函数

class <类名>：

def \_\_del\_\_(self)

<语句块>

#### \_\_del\_\_()的使用说明：

函数名和参数：Python解释器内部约定，保留方法·

调用条件：当实例对象被真实删除时，才调用该函数内语句·

真实删除：当前对象的引用数为0或当前程序退出（垃圾回收）

#### sys.getrefcount(<对象名>）获得对象的引用次数:

返回对象引用次数的方法，辅助删除对象时的分析·

sys.getrefcount（）函数返回值为被引用值+1·

非特定目的，不建议自己撰写析构函数，利用Python垃圾回收机制

### 小结：

1，类的基本构成

类对象、实例对象

2，类的构造函数

\_\_init\_\_()

3，类的属性

类属性、实例属性·

4，类的方法

类方法、实例方法、自由方法、静态方法、保留方法

5，类的析构函数

\_\_def\_\_()、sys.getrefcount()

## P20 1.4 Python类的封装2

封装Encapsulation：属性和方法的抽象

私有属性：只能在类内部访问

公开属性：可以通过类/对象名访问

私有方法：只能在类内部使用

公开方法：可以通过类/对象名访问

### 私有类属性：

仅供当前类访问的类属性，子类亦不能访问

#### 定义：

class<类名>：

<私有类属性名>=<类属性初值>

def \_\_init\_\_(self,<参数列表））

#### 区别：私有类属性名开始需要有两个下划线(\_\_)，如\_count

### 私有实例属性：

仅供当前类内部访问的实例属性，子类亦不能访问

#### 定义：

class<类名）：

def \_\_init\_\_(self,<参数列表>）

self.<实例属性名>=<实例属性初值>

#### 区别：私有实例属性名开始需要有两个下划线（\_\_），如\_\_name

### 私有方法：

是类内部定义并使用的函数

#### 定义：

class<类名）：

def <方法名>（self，<参数列表>）

私有方法名开始需要有两个下划线（\_\_），如\_\_getCount()

## P23 1.4 Python类的封装5

### 类的保留属性

Python解释器预留的类属性，以双下划线开头和结尾·

也叫：特殊属性，Special Attributes·

特点：双下划线开头和结尾·

作用：为理解Python类提供了统一的属性接口·

属性值：具有特定含义，类定义后直接使用

### 仅用<类名>访问的保留属性：

\_\_name\_\_:类的名称

\_\_qualname\_\_:以.分隔从模块全局命名空间开始的类名称

\_\_bases\_\_:类所继承的基类名称

### 可以用<类名>和<对象名>访问的保留属性：

<类>.\_\_dict\_\_:包含类成员信息的字典，key是属性和方法名称，value是地址

<对象>.\_\_dict\_\_:包含对象属性信息的字典，key是属性名称，value是值

\_\_class\_\_:对象所对应的类信息，即type信息

\_\_doc\_\_:类描述，写在类定义下的首行字符串，不能继承

\_\_module\_\_:类所在模块的名称

## P24 1.4 Python类的封装6

### 类的保留方法

Python解释器预留的类方法，以双下划线开头和结尾·

也叫：特殊方法，Special Methods·

特点：双下划线开头和结尾·

作用：为操作Python类提供了统一的方法接口·

方法逻辑：具有特定含义，一般与操作符关联，类定义需要重载

obj.\_\_init\_\_():obj=ClassName(),初始化实例对象

obj.\_\_del\_\_():del obj,删除实例对象的函数逻辑

obj.\_\_repr\_\_():repr(obj),定义对象打印字符串的函数逻辑

obj.\_\_str\_\_():str(obj),定义对象字符串转换操作的函数逻辑

obj.\_\_bytes\_\_():bytes(obj),定义对象字节串转换的函数逻辑

obj.\_\_format\_\_():obj.format(),定义对象格式化输出函数逻辑

obj.\_\_hash\_\_():hash(obj),定义对象哈希操作的函数逻辑

obj.\_\_bool\_\_():bool(obj),定义对象布尔运算的函数逻辑

obj.\_\_len\_\_():len(obj),定义对象长度操作的函数逻辑

obj.\_\_reversed():obj.reversed(),定义对象逆序的函数逻辑

obj.\_\_abs\_\_():abs(obj),定义对象绝对值操作的函数逻辑

obj.\_\_int\_\_():int(obj),定义对象整数转换的函数逻辑

obj.\_\_It\_\_():obj1<obj2，对象间比较操作的保留方法

obj.\_\_le\_\_():obj1<=obj2，对象间比较操作的保留方法

obj.\_\_eq\_\_():obj1==obj2，对象间比较操作的保留方法

obj.\_\_ne\_\_():obj1!=obj2，对象间比较操作的保留方法

obj.\_\_gt\_\_():obj1>obj2，对象间比较操作的保留方法

obj.\_\_ge\_\_():obj1>=obj2，对象间比较操作的保留方法

### 小结：

-Python类保留方法使用说明·Python保留了超过100个各种保留方法·

-保留方法对应对象、对象间、对象方法的各种操作·

-有哪些保留方法？请参考各种数据类型的使用

## P25 1.4 Python类的封装7

### 单元小结：

（1）封装的理解·：属性的抽象、方法的抽象

（2）私有属性和公开属性：一私有类属性、私有实例属性

（3）私有方法和公开方法：私有方法的双下划线表示和理解

（4）类的保留属性：类的7个保留属性

（5）类的保留方法：类的>100个保留方法

## P26 1.5 Python类的继承1

(1)继承的理解

(2)类继承的构建

(3)Python最基础类

(4)类的属性重载

(5)类的方法重载

(6)类的多继承

### 继承 Inheritance：代码复用的高级抽象·

-继承是面向对象程序设计的精髓之一·

-实现了以类为单位的的高抽象级别代码复用·

-继承是新定义类能够几乎完全使用原有类属性与方法的过程

## P28 1.5Python类的继承3

### 类继承的构建

在定义类时声明继承关系：

#### 定义：

class<类名>（<基类名>）：

def\_init(self,<参数列表>）

<语句块>

#### 基类名可以带有路径：ModuleName.BaseClassName

派生类可以直接使用基类的属性和方法：

基类的属性和方法基本等同于定义在派生类中·

派生类可以直接使用基类的类属性、实例属性·

派生类可以直接使用基类的各种方法·

使用基类的类方法和类属性时，要用基类的类名调用

#### 派生类的约束：

-派生类只能继承基类的公开属性和方法·

-派生类不能继承基类的私有属性和私有方法

## P29 1.5 Python类的继承4

### Python最基础类

#### object类是Python所有类的基类：

-object是Python最基础类的名字，不建议翻译理解·

-所有类定义时默认继承object类·

-保留属性和保留方法本质上是object类的属性和方法

#### Python对象的三个要素：标识、类型和值·

标识identity：对象一旦构建不会改变，用id()获得，一般是内存地址·

类型type：对象的类型，用type()获得·

值value：分为可变mutable与不可变immutable两种

#### 2个与基础类有关的Python内置功能：

id(x)：返回x的标识，Cpython用内存地址表示

x is y：判断x和y的标识是否相等，返回True/False，不判断值

## P30 1.5 Python类的继承5

### 类的属性重载

#### 重载：派生类对基类属性或方法的再定义：

属性重载：派生类定义并使用了与基类相同名称的属性·

方法重载：派生类定义并使用了与基类相同名称的方法

**属性重载**

#### 最近覆盖原则：重载无需特殊标记·

步骤1：优先使用派生类重定义的属性和方法·

步骤2：然后寻找基类的属性和方法·

步骤3：再寻找超类的属性和方法

## P31 1.5 Python类的继承6

### 类的方法重载

#### 定义：

方法重载：派生类对基类方法的再定义：

#### 完全重载：

派生类完全重定义与基类相同名称的方法

直接在派生类中定义同名方法即可·

#### 增量重载：

派生类扩展定义与基类相同名称的方法

## P32 1.5 Python类的继承7

### 多继承的构建：

在定义类时声明继承关系

class<类名>（<基类名1>，<基类名2>…<基类名N>）：

def \_\_init\_\_(self,<参数列表>）

<语句块）

基类名可以带有路径：ModuleName.BaseClassName

### 类多继承的使用说明:

所有属性和方法的使用按照“深度优先、从左到右”的方式选取·

构造函数也参照上述原则，super（)也参照上述原则·

多个基类的顺序是关键

## P33 1.5 Python类的继承8

### 单元小结：

（1）继承的理解：基类、派生类、父类、子类、超类

（2）类继承的构建：派生类的构建、2个内置函数

（3）Python最基础类：object类、id()、is

（4）类的属性的重载：最近覆盖原则、实例属性重载

（5）类的方法重载：增量重载、super()函数

（6）类的多继承：深度优先、从左到右

## P36 2.1 Python类的运算1

### 1，运算的理解

### 2，算术运算的重载

### 3，比较运算的重载

### 4，其他运算的重载

### 5，成员运算的重载

## P37 2.1 Python类的运算2

### 运算重载的限制：

-不能重载Python语言内置类型的运算符

-不能新建运算符，只能通过重载完成

-is and not or不能被重载

## P38 2.1 Python类的运算3

### 算术运算符的重载

#### 算术运算的种类·

一元算术运算符：

+、、~

二元算术运算符：

+、、\*、/、//、%、divmod（）、pow（）、\*\*<<、>>、&、^、|

#### 算术运算符的重载：一元算术运算符:

.\_\_neg\_\_(self):-obj,定义对象取负的运算逻辑

.\_\_pos\_\_(self):+obj,定义对象取正的运算逻辑

.\_\_abs\_\_(self):abs(obj),定义对象绝对值的运算逻辑

.\_\_invert\_\_(self):~obj,定义对象取反的运算逻辑

#### 算术运算符的重载：二元算术运算符:

.\_\_add\_\_(self other):obj+other,定义两个对象加法的运算逻辑

.\_\_sub\_\_(self other):obj-other,定义两个对象减法的运算逻辑

.\_\_mul\_\_(self,other):obj\*other,定义两个对象乘法的运算逻辑

.\_\_truediv\_\_(self,other):obj/other,定义两个对象除法的运算逻辑

.\_\_floordiv\_\_(self,other):obj//other,定义两个对象整数除的运算逻辑

.\_\_mod\_\_(self,other):obj%other,定义两个对象模的运算逻辑

.\_\_divmod\_\_(self,other):divmod(obj,other),定义两个对象除模的运算逻辑

.\_\_pow\_\_(self,other):obj\*\*other,定义对象幂的运算逻辑

.\_\_lshift\_\_(self,other):obj<<other,定义对象左移的运算逻辑

.\_\_rshift\_\_(self other):obj>>other,定义对象右移的运算逻辑

.\_\_and\_\_(self,other):obj&other,定义两个对象位与的运算逻辑

.\_\_xor\_\_(self,other):objn^other,定义两个对象位异或的运算逻辑

.\_\_or\_\_(self,other):obj&other,定义两个对象位或的运算逻辑

## P39 2.1 Python类的运算4

### 比较运算符的重载

#### 比较运算：

<,<=,==,!=,>,>=

#### 比较运算符的重载:

.\_\_It\_\_(self,other):obj<other，两个对象比较操作的运算重载

.\_\_le\_\_(self,other):obj<=other，两个对象比较操作的运算重载

.\_\_eq\_\_(self,other):obj==other，两个对象比较操作的运算重载

.\_\_ne\_\_(self,other):obj!=other，两个对象比较操作的运算重载

.\_\_gt\_\_(self,other):obj>other，两个对象比较操作的运算重载

.\_\_ge\_\_(self,other):obj>=other，两个对象比较操作的运算重载

## P40 2.1 Python类的运算5

### 成员运算的重载

#### 成员运算的种类·

成员获取：[]、del、.reversed()

成员判断：in、not in

#### 成员运算符的重载：成员获取

.\_\_getitem\_\_(self,key):obj[k],定义获取对象中序号k元素的运算逻辑，k为整数

.\_\_setitem\_\_(selfkey,v):obj[k]=V,定义赋值对象中序号k元素的运算逻辑

.\_\_delitem\_\_(self key):del obj[k],定义删除对象中序号k元素的运算逻辑

.\_\_reversed\_\_(self):obj.reversed(),定义对象逆序的运算逻辑

.\_\_contains\_\_(self,item):item in obj,定义in操作符对应的运算逻辑

## P41 2.1 Python类的运算6

### 其他运算的重载

#### 小结：

其他运算的种类:

Python内置函数：repr()、str()、len()、int()、float()、complex()、round()、bytes()、bool()、format()

类的常用方法：.format()

## P42 2.1 Python类的运算7

### 小结：

1，运算的理解：运算的种类，运算符

2，算术运算的重载：一元/二元算术运算符

3，比较运算的重载：<、<=、==、！=、>、>=

4，成员运算的重载:成员获取/判断操作符

5，其他运算的重载：内置函数相关运算符

## P43 2.2 Python类的多态1

### 1，对态的理解

### 2，参数类型的多态

### 3，参数形式的多态

## P44 2.2 Python类的多态2

### 多态的理解

多态Polymorphism：仅针对方法，方法灵活性的抽象

-参数类型的多态：一个方法能够处理多个类型的能力

-参数形式的多态：一个方法能够接受多个参数的能力

-多态是OOP的一个传统概念，Python天然支持多态，不需要特殊语法

## P45 2.2 Python类的多态3

### 参数类型的多态

天然支持：Python方法无类型声明限制：

-Python的函数/方法没有类型声明限制，天然支持参数类型的多态性·

-Python编程理念在于：文档约束，而非语法约束·

-对不同参数类型的区分及功能，需要由程序员完成

## P46 2.2 Python类的多态4

### 参数形式的多态

天然支持：Python方法/函数支持多种参数形式：

-Python的函数/方法可以支持可变参数，支持参数形式的多态性·

-Python的类方法也是函数，函数的各种定义方式均有效·

-对不同参数个数及默认值的确定，需要由程序员完成

## P47 2.2 Python类的多态5

### 单元小结：

1,多态的理解：方法灵活性的抽象

2,参数类型的多态：天然支持：参数无类型限制

3,参数形式的多态：天然支持：函数的可变参数

## P50 2.4 Python对象的引用1

### 1，引用的理解

### 2，浅拷贝和深拷贝

### 3，类的方法引用

## P51 2.4 Python对象的引用2

### 引用的理解

#### 引用Reference：对象的指针：

引用是内存中真实对象的指针，表示为变量名（程序中）或内存地址（系统中）·

每个对象存在至少1个引用，id()函数用于获得引用·

在传递参数和赋值时，Python传递对象的引用，而不是复制对象

#### 不可变对象的引用：字符串

Python解释器内部对a和c维护1个对象

#### 不可变对象的引用：字符串

运算后产生的对象由解释器重新建立

#### 可变对象引用：列表

每个可变对象都由解释器重新创建，不复用内存

#### 导致引用+1的情况：

对象被创建：d=DemoClass0·

对象被引用：a=d·

对象被作为函数或方法的参数：sys.getrefcount（d)·

对象被作为一个容器中的元素：Is=[d]

#### 导致引用-1的情况·

对象被删除：del d·

对象的名字被赋予新的对象：d=123

对象离开作用域：foo()函数的局部变量count

对象所在容器被删除：del ls

#### 小结：

引用Reference：对象的指针：

引用是内存中真实对象的指针，表示为变量名或内存地址·

在传递参数和赋值时，Python传递对象的引用，而不是复制对象·

不可变对象与可变对象的内存管理略有不同

## P52 2.4 Python对象的引用3

### 浅拷贝和深拷贝

-拷贝：复制一个对象为新的对象，内存空间有“变化"

-浅拷贝：仅复制最顶层对象的拷贝方式，默认拷贝方式·

-深拷贝：迭代复制所有对象的拷贝方式

#### 深拷贝：

完全拷贝对象内容·

采用copy库的deepcopy()方法·

迭代拷贝对象内各层次对象，完全新开辟内存建立对象·

深拷贝仅针对可变类型，不可变类型无需创建新对象

#### 小结：

-浅拷贝：仅复制最顶层对象的拷贝方式，默认拷贝方式·

-深拷贝：迭代复制所有对象的烤贝方式，一般采用copy库的deepcopy()

-一般深拷贝都与可变类型关联

## P53 2.4 Python对象的引用4

### 类的方法引用

#### 再看类的实例方法：

-定义方式：def<实例方法名>（self,<参数列表>）·

-实例方法名也是一种引用，即对方法本身的引用·

-当方法被引用时，方法（即函数）将产生一个对象：方法对象

<对象名>.<方法>:是一个对方法的引用 ps：lucky=dc1.lucky

<对象名>.<方法>（方法参数）等价于

<类名>.<方法名>（<对象名>，方法参数）

## P54 2.4 Python对象的引用5

### 单元小结：

1，引用的理解：可变类型与不可变类型的引用

2，浅拷贝和深拷贝：一层拷贝，迭代拷贝

3，类的方法引用：方法亦引用

## P55 2.5 Python类的高级话题1

1，命名空间的理解

2，类的特性装饰器

3，自定义的异常类型

4，类的名称修饰

5，Python最小空类

## P56 2.5 Python类的高级话题2

### 命名空间的理解

#### 命名空间Namespace：从名字到对象的一种映射：

作用域：全局变量名在模块命名空间，局部变量名在函数命名空间·

属性和方法在类命名空间，名字全称：<命名空间>.<变量/函数名>·

命名空间底层由一个dict实现，变量名是键，变量引用的对象是值

#### 命名空间Namespace：从名字到对象的一种映射:

复数z，z.real和z.imag是对象z命名空间的两个属性·

对象d，d.name和d.printName)是对象d命名空间的属性和方法·

global和nonlocal是两个声明命名空间的保留字

## P57 2.5 Python类的高级话题3

### 类的特性装饰器

#### @property:类的特性装饰器:

-使用@property把类中的方法变成对外可见的“属性”

-类内部：表现为方法

-类外部：表现为属性

## P58 2.5 Python类的高级话题4

### 自定义的异常类型

#### 异常Exception 也是一种Python类：

-try-except捕捉自定义的异常·

-继承Exception类，可以给出自定义的异常类·

-自定义异常类是类继承的正常应用过程

## P59 2.5 Python类的高级话题5

### 类的名称修饰

#### 名称修饰 Name Mangling：类中名称的变换约定:

-Python通过名称修饰完成一些重要功能·

-采用下划线(\_)进行名称修饰，分为5种情况

-\_X、X\_、\_\_X、\_\_X\_\_、\_

#### 1，\_X:单下划线开头的名称修饰:

单下划线开头属性或方法为类内部使用 PEP8·

只是约定，仍然可以通过<对象名>.<属性名>方式访问·

功能：from XX import\*时不会导入单下划线开头的属性或方法

#### 2，X\_：单下划线结尾的名称修饰·

-单下划线结尾属性或方法为避免与保留字或已有命名冲突PEP8·

-只是约定，无任何功能性对应

#### 3，\_\_X:双下划线开头的名称修饰:

-双下划线开头属性或方法将被解释器修改名称，避免命名冲突·

-不是约定，而是功能性，实现私有属性、私有方法·

-\_\_X会被修改为：\_<类名>\_\_X

#### 4，\_\_X\_\_:双下划线开头和结尾的名称修饰

-双下划线开头和结尾的属性或方法任何特殊功能，名字不被修改

-部分名称是保留属性或保留方法

#### 5，\_:单下划线

-单下划线是一给无关紧要的名字，无特殊功能

#### 小结：

\_X：约定内部使用，仅在import\*时不被引用

X\_：避免与保留字冲突，无特殊功能

\_\_X：不被子类继承、可用于设定私有、改变为：\_<类名>\_\_X

\_\_X\_\_：无特殊功能，部分用于保留属性和保留方法·

\_：无特殊功能，不重要的命名

## P60 2.5 Python类的高级话题6

### Python最小空类

#### 定义：

class <类名>():

pass

#### Python最小空类的作用:

-类是一个命名空间，最小空类可以当作命名空间使用

-最小空类可以辅助数据存储和使用

-动态增加属性是Python类的一个特点

## P61 2.5 Python类的高级话题7

### 单元小结：

1,命令空间的理解：命名作用域，nonlocal,global

2,类的特性装饰器：@property,@.setter

3,自定义的异常类型：Exception类

4,类的名称修饰：类中下划线命名的5种情况

5,Python最小空类：属性的动态增加及作用

# 三、Python 计算生态构建

## P1 1.课程基本情况

### 课程目标

（1）管握采用Python语言构建Python计算生态的方法

（2）掌握采用C/C++语言构建Python计算生态的方法

（3）能独立完成50-100行多语言混合的Python第三方库程序的编写

### 第1周内容结构

#### 1，深入理解Python库·

库、模块和包·

模块和包的命名空间·

模块和包的属性

#### 2，模块和包的构建·

模块的构建.·

常规包的构建·

命名空间包的构建

#### 3，实例1：矩阵乘法模块的构建·

矩降乘运算·

模块构造方法实践

#### 4，深入理解import系统·

import系统·

模块对象·

模块的查找和加载

#### 5，Python第三方库的发布·

发布前的准备和发布概念·

第三方库的发布流程

#### 6，Python语法实践

一小时Python库构建操作实践

### 第2周内容

#### 1，C语言精简概述

C语言概述、语法及实例

C编辑环境配置

编译和解释

#### 2，Python与C的交互方法

Python的扩展方法

Python的嵌入方法

Python的调用方法

#### 3，Python扩展的CFFI方法

CFFI功能接口

CFFI库的使用

#### 4，实例2：矩阵乘法的C语言加速

C语言加速文件组织结构及示例

Python和C的性能比较

#### 5，Python语法实践

一小时多语言库构建操作实践

## P2 1.0 第1周内容导学

### 第一周学习目标

1，理解Python库的相关概念及组织方式

2，掌握构建及发布Python库的基本方法

3，采用Python语言独立编写第三方库

## P5 1.1 深入理解Python库·33

### 模块的命名空间

#### 模块是一个命名空间：

模块对应单独的.py文件，它是一个独立的命名空间·

模块内可能包含：类、函数、语句（直接可执行）、变量等元素·

模块内还包括一些其他对模块进行约束和管理的语法元素

#### 模块是一个命名空间：

模块中语句：在import时一次性执行·

模块内变量、类和函数：在import时采用<模块名>方式可访问·

单下划线的顶层命名元素：不会在from.import\*时被导入

## P6 1.1 深入理解Python库4·

### 包的命名空间

#### 包是一个有层次的命名空间：

* 通过包的组织可以形成由英文句号（.）分隔的层次化命名空间·
* \_\_init\_\_.py用来构成包的定义，区分于包含.py文件的普通目录**(万物皆可py文件，即\_\_init\_\_.py文件就是包的py表现形式)**
* 包、子包和模块可以用import进行导入或单独导入

## P7 1.1 深入理解Python库5·

### 模块的名称属性

#### 名称属性：表达模块名称的预定义变量：

-当程序以脚本方式直接执行时，\_\_name\_\_的值为'\_\_main\_\_'·

-当程序以模块方式被引用执行时，\_\_name\_\_的值为模块名称·

-作用：区分程序以何种方式执行

#### if \_\_name\_\_=='\_\_main\_\_:

当程序以脚本方式执行时，后续代码块可直接执行，否则不执行·

作用1：作为模块主体功能的单元测试部分·

作用2：作为模块内部保留的额外功能部分

## P8 1.1 深入理解Python库6

### 单元小结：

1，库，模块和包：命名空间、有层次的命名空间

2，模块的命名空间：变量，类，函数，语句等作用域

3，包的命名空间：\_\_init\_\_.py,点分隔的层次结构

4，模块的名称属性：\_\_name\_\_,'\_\_main\_\_'

## P9 1.2 模块和包的构建1

### 1，模块的构建

### 2，常规包的构建·

### 3，命名空间包的构建

## P10 1.2 模块和包的构建2

### 模块的构建

#### 模块的构建原则：如何编写好一个.py文件？

功能闭包：单一.py文件实现单一且完整的功能·

抽象适度：用函数或类进行抽象，结合功能选择合适抽象·

操作闭包：模块无顶层可执行语句，导入时无输出

#### 模块的构建原则·

功能闭包：功能定义要清晰、设计要合理（紧耦合vs松耦合）·

抽象适度：采用类或函数，尽量选择一种；多种也无妨·

操作闭包：采用\_\_name\_\_、无全局可执行语句、尽量无全局变量

## P11 1.2 模块和包的构建3

### 常规包的构建·

#### 包有两种：常规包和命名空间包:

-常规包：Regular Packages，通过\_\_init\_\_.py对文件和目录组织形成的包·

-命名空间包：Namespace Packages，由更分散子包组成的包·

子包的位置可以在文件系统中不连续·

子包可以是压缩文件或网络连接或其他系统资源

#### 常规包 Regular Packages·连续目录空间表达的、有层次的命名空间:

每个目录中包含一个\_init\_.py文件，可以是空文件·

当包/子包被导入时，对应目录的\_\_init\_\_.py文件将被执行

#### \_\_all\_\_属性：

-from 包名 import \*形式需要在包下的\_\_init\_\_.py文件中增加\_\_all\_\_属性赋值

-\_\_all\_\_需要被赋值为列表对象，包含当前包下所有希望被导入的模块名称

-\_\_all\_\_用来辅助导入模块，但不能辅助导入列表

#### 常规包的构建流程：

-设计好包层次结构（命名空间），每个目录配置一个\_init\_.py文件·

-完善\_\_init\_\_.py文件，并酌情赋值\_\_all\_\_属性·

-再次理解包和模块的不同,模块提供功能，包提供命名空间

#### 常规包的导入流程（import）：

-每个包仅被导入一次，且包导入按照层次结构进行

-直接导入包不能调用功能，需要导入模块层次

-from .. import 直接导入具体模块，可以简化调用时命名空间表达

-\_\_init\_\_.py文件中通过\_\_all\_\_属性支持from 常规包名 import \* 功能

### 命名空间包的构建

#### 命名空间包：表达命名空间层次结构的一种逻辑包形式：

-命名空间中各部分可以在不同的文件系统位置·

-命名空间中各子包并不包含\_\_init\_\_.py文件（普通目录！）·

-Python解释器通过sys.path变量来隐式维护命名空间包

#### sys.path:指定搜索路径的字符串列表：

-指定import时搜索模块或包的路径列表，路径是相对或绝对路径·

-sys.path是一个列表类型，可以用sys.path.append(p)增加新路径p·

-载入后，根据其中包的名称和层次结构自然组成了命名空间包

#### \_\_path\_\_属性：

记录了某个包（命名空间）的绝对或相对路径，列表类型

常规包：路径是单一的，列表中只有一个元素

命名空间包：路径只多元的，列表中可能有多个元素

pkg1命名空间的路径：['project1\\pkg1','project2\\pkg1']

## P13 1.2 模块和包的构建5

### 单元小结:

1,模块的基本构建

构建原则，\_\_name\_\_,dir(m):以列表的形式返回模块所使用的命名

2,常规包的构建：\_\_init\_\_.py,\_\_all\_\_属性

3，命名空间包的构建

sys.path：需要利用它添加分散在外面的路径

\_\_path\_\_：查看该属性：包名.\_\_path\_\_为：\_NamespacePath([....])

## P16 1.4 深入理解import系统1

### 单元开篇:

1,import系统·

2,模块查找

3,模块加载

4,import使用方式

## P17 1.4 深入理解import系统2

### import系统·

#### import系统：扩展命名空间及功能的方法：

构成：import保留字、\_\_import\_\_()函数和importib标准库·

步骤：模块的查找、模块的加载·

价值：import系统是Python代码复用和命名空间管理的精髓

#### import系统：扩展命名空间及功能的方法:

import保留字：调用\_\_import\_\_()进行模块查找，以及模块的加载·

\_import\_\_()函数：模块的查找，建立模块对象

importib标准库：与import系统相关的丰富APl

### 模块对象

#### 定义：

万物皆对象：模块被导入后成为了对象:

模块的对象形式：模块在程序中使用都是以对象形式体现·

类似类对象，模块对象只有一个·

模块对象生成时，模块中代码会被执行;如果模块中包含类的定义，因此会有类对象产生

## P18 1.4 深入理解import系统3

### 模块查找

（）-- 模块查找 -->模块对象

#### 查找的开始·

输入参数：模块的名字，例如：pkg.pkg1.m1·

输入方式：import保留字、\_\_import\_\_()、importlib标准库·

基本行为：按层次结构逐层查找，例如：pkg->pkg1->m1

#### 查找路径：1

第1步.sys.modules:

第一步查找sys.modules，之前被引入模块的缓存（cache)·

sys.modules是一个字典，<模块名/对象引用：<加载路径>·

如果模块不在sys.modules中，则进入下一步

#### 查找策略：2-4

第2步,用户通过import钩子扩展的查找模式

import钩子：import hook·

扩展查找模块的方式：meta\_path方式和import路径方式·

meta\_path：将查找方法增加到sys.meta\_path列表变量·

Import路径：将查找方法注册到sys.path hooks列表变量

第3步,内置模块的路径

第4步,sys.path(列表)提供的加载路径，可以是zip文件或URL

## P19 1.4 深入理解import系统4

### 模块的加载·

模块对象-- 模块加载 -->当前命名空间

#### 建立模块对象的过程：

1.找到模块后，如果模块对象存在，则使用现有模块对象·

2.创建一个新的模块对象，将其加入sys.modules·

3.在程序当前命名空间执行模块代码

#### 建立模块对象的过程：

创建对象：importlib.abc.Loader.create\_module()

执行对象：importlib.abc.Loader.exec\_module()

#### 模块对象的命名空间:

模块的命名空间与引用位置有直接关系·

如果引用在文件顶层，则使用文件顶层命名空间访问模块·

如果引用在非顶层，则使用局部命名空间访问模块

## P20 1.4 深入理解import系统5

### import使用方式

#### import的三种使用方式：

* import <模块名>·
* from <模块名> import <类/函数名/\*>·，自定义：form <包名> import <模块名>
  + from <模块名> import \*
  + form <包名> import <模块名>
* import <模块名> as <别名>

#### import <模块名>：

当前命名空间下的一个子命名空间·

成功加载后，产生一个与<模块名>同名的<模块对象名>·

实际上：<模块对象名>.<子命名空间内元素>方式访问

#### from<模块名>import<类/函数/\*>：

将导入元素加载到当前命名空间下·

成功加载后，产生类对象或函数对象，覆盖同名对象·

实际上：<类对象名>或<函数对象名>方式访问

#### import<模块名>as<别名>：

当前命名空间下的一个子命名空间·

成功加载后，产生一个与<别名>同名的<模块对象名>·

实际上：<模块对象名>.<子命名空间内元素>方式访问

## P21 1.4 深入理解import系统6

### 单元小结

1，import系统：模块查找和加载、模块对象

2，模块的查找：sys.modules、查找策略，钩子

3，模块加载：模块对象的创建和执行

4，import使用方式：三种使用方式再回顾

## P22 1.5Python第三方库的发布1

### 单元开篇

1，发布的准备

2，第三方库的发布流程

3，第三方库的发布概念

## P23 1.5Python第三方库的发布2

### 发布前的准备

#### Python第三方库发布:

PyPI:Python Package Index,用来登记第三方库信息·

Github、bitbucket:存储第三方库源代码及文档·

目标：通过pip进行安装和管理、源代码和文档网络可管理

#### 一些基本概念·

项目project：PyPl上一组发布和文件的统称·

发布release：项目的一个特定版本，每个发布有一个确定的版本号·

文件file：即package，一次发布包含的具体文件

python -m pip install --user --upgrade setuptools wheel twine

## P24 1.5Python第三方库的发布3

### 第三方库的发布流程

#### 第一步：整理目录结构

mxmul\_st:最顶层目录，相当于当前项目

所有setuptools 命令在最顶层目录中执行

mxmul\_pkg:包目录，分发的主体

所有源代码及层次化命名空间放到这里

#### 第二步：创建其他相关文件

setup.py:配置发布信息的文件，很重要

README.md:markdown格式的说明文件

LICENSE:版权声明文件

#### 第三步：执行打包命令

python setup.py sdist bdist\_wheel

#### 第四步:执行发布命令

twine upload dist/\*

#### 整体步骤：

第一步：整理目录

第二步：创建其他相关文件，主要是setup.py

第三步：执行打包命令：python setup.py ...

第四步：执行发布命令：twine upload dist/\*

## P25 1.5Python第三方库的发布4

### 第三方库发布概念

#### 与发布Distribution相关的概念：

打包发布：Built Distribution，对文件整理打包后进行发布·

源发布：Source Distribution，发布源代码的方式，简称sdist·

可执行发布：Binary Distribution，发布编译后可执行代码的方式

#### 与发布Distribution相关的概念逻辑：

-打包发布：一次发布一个打包文件，比如.tar.gz或wheel文件

-源发布：对于纯Python编写的库，源发布有最佳的适用性

-可执行发布：对于各类语言的扩展库，节省了本机编译需求，适用性较好

#### 与打包格式相关的概念：

-Egg：一种打包格式，已经被wheel替换·

-Wheel：一种打包格式，Python主流打包方式，PEP427，zip压缩·

-Egg和Wheel都是打包格式，可以包括源代码或可执行代码

## P26 1.5Python第三方库的发布5

### 单元小结：

1，发布前的准备：基本概念，发布工具，账号

2，第三方库的发布流程：四个步骤：目录，setup.py,打包，发布

3，第三方的发布概念：打包发布，源发布，可执行发布

# 四、Python精进路线展望

## 课程目标

1，掌握Python语法及语言深入的概念及内容

2，全面了解Python计算生态在各领域的发展

3，了解进一步学习Python语言的路线

## 第1周内容

### 精进的Python语法：

上下文管理器、异步协程

迭代器、生成器

装饰器与语法糖

### 精进的Python语言：

全部保留字

全部特殊符号

全部内置函数

### 精进的Python计算生态：

数据分析到人工智能

Web解析到网络空间

人机交互到艺术设计

### 实例1：基于OpenCV的人脸识别：

OpenCV运用实例

计算生态编程思想运用

### Python学习路线展望：

Python知识体系回顾

学习及精进路线

## P3 1.1精进的Python语法1

### 1，上下文管理器

### 2，迭代器类型

### 3，生成器设计

### 4，异步协程函数

### 5，装饰器与语法糖

## P4 1.1精进的Python语法2

### 上下文管理器

#### 上下文Context：程序执行中某个状态：

-程序执行所需的一些内外部参数，构成了程序运行时状态·

-上下文是用来表达程序运行状态的概念，对应内存状态·

-上下文是程序中断保留或恢复运行的重要状态信息

#### 上下文管理器：一个可以在程序中加载独立上下文的对象

万物皆对象：上下文管理器也是一个对象，管理者一个独立上下文区域·

上下文管理器使用with显式创建·

进入和退出分别对应\_\_enter\_\_()和\_\_exit\_\_()方法

#### 定义：

"cm.\_\_entet\_\_()执行开始，返回结果绑定给对象引用"

with <表达式> as <对象引用>：

<语句块>

"cm.exit() 执行结束，退出上下文区域"

#### 如何使用上下文管理器 ：

1.with要配合特定对象使用

2.Python支持：

file

decimal.Context

thread.LockType

threading.Lock

threading.RLock

threading.Condition

threading.Semaphore

threading.BoundedSemaphore

3.用户可自定义对象

#### 上下文管理器：一个可以在程序中加载独立上下文的对象：

-with上下文管理器与特定对象配合使用·

-特定对象包括：Python支持对象、自定义with兼容对象·

-自定义with兼容对象：重载\_\_enter\_\_()和\_\_exit\_\_()方法

## P5 1.1精进的Python语法3

### 迭代器类型

#### 迭代器类型 lterator Types：实现容器中元素的迭代：

容器Container：包含一组元素的数据类型，如全部组合数据类型·

迭代器：容器之上的一种迭代，迭代方式以对象形式表达·

迭代器定义了逐一获取容器中元素的方法

#### 迭代器类型 lterator Types：实现容器中元素的迭代:

迭代器是一个对象，常与for...in...配合使用，需要重载两个方法·

\_\_iter\_\_()返回迭代器对象自身，配合for.in..使用时需要·

\_\_next\_\_()返回容器的下一个元素，如果遍历结束，产生Stoplteration异常

#### 迭代器类型Iterator Types:实现容器中元素的迭代:

迭代器非常常用，尤其是涉及管理一组数据的应用场景·

自定义类+for..in..管理一组数据，作为数据管理的核心自定义数据类型·

迭代器返回下一个元素的方式为优化程序内存使用提供了重要支持

## P6 1.1精进的Python语法4

### 生成器设计

#### 生成器Generator：快速实现迭代器类型的方式：

-生成器是迭代器的另一个角度诠释，生成器都是迭代器·

-生成器以逐一产生数据为角度，重点在于迭代逻辑的方便实现·

-生成器只需要yield保留字即可定义并表达，不用重载对象方法

#### 生成器函数Generator Function：实现迭代逻辑的函数：

-生成器函数是生成器的核心，用来实现迭代逻辑

-任何包含yield语句的函数，都是生成器函数

-对于函数来说，yield和return功能接近，但不完全相同

#### 生成器函数的设计原则：

-若想不断生成数据，生成器函数需要存在迭代/循环过程

-每次迭代/循环产生的结果以yield方式表达

-生成器函数要与for...in...配合使用，注意：生成器是一种特殊迭代器

## P7 1.1精进的Python语法5

### 装饰器与语法糖

#### 装饰器 Decorator：以函数为参数实现其额外功能更新：

需求：对于一个已有函数，如何更新其功能？（同名使用）·

解决方式1：重新编写该函数，把功能补充进去·

解决方式2：不改变原来函数，增补一个装饰器给它

#### 装饰器Decorator:以函数为参数实现器额外功能更新：

-classmethod()和staticmethod()内置函数

-@classmethod

-@staticmethod

def f(..):

...

f=staticmethod(f)

等价于

@staticmethod

def f(...)

...

#### 语法糖Syntactic Sugar:方便使用但无实质功能的语法：

-语法糖：一种功能的简化使用方法，非功能性改变

-PYthon中采用@作为语法糖，辅助装饰器使用

-任何语法糖表达的装饰器都对应一个函数

#### 装饰器设计原理：

-语法糖不重要，重要的是装饰器函数，该函数以函数为参数·

-由于可能存在装饰器迭代，每个装饰器函数需要返回一个函数引用·

-装饰器函数内部，进行功能扩展

#### 装饰器使用场景：

-对原函数功能的补充：测量时间、增加打印等·

-对原函数功能的调整：利用原函数运行结果，再次运算产生新的结果·

-对原函数功能的重写：只是借助原来的名字，谨慎修改旧函数

## P8 1.1精进的Python语法6

### 异步协程函数

#### 协程Coroutine：在程序层面的多段代码异步执行：

-Python3.5新增内置功能：async和await保留字·

-async：用于定义异步执行的协程代码段·

-await：异步代码段之间的手工同步方式

#### async的用法：

1、协程函数

async def foo():

pass

2、协程上下文管理器

async with lock:

pass

3、协程迭代器

async for data in pipeline:

pass

#### await的用法：

获取异步协程的结果，结果为awitable对象

async def foo(db):

data = await db.fetch('SELECT ...')

pass

## P9 1.1精进的Python语法7

### 单元小结：

1,上下文管理器：with ... as ,\_\_enter\_\_()，\_\_exit\_\_()

2,迭代器类型：for .. in、\_\_ite\_\_()、\_\_next\_\_()

3,生成器设计：yield、迭代/循环的设计过程

4，装饰器与语法糖：装饰器作用、@语法糖、设计原理

5，异步协程函数：async def/with/for 、await

## P10 1.2精进的Python语言1

### 1，Python保留字

### 2，Python特殊字符

### 3，Python内置函数

## P11 1.2精进的Python语言2

### Python保留字：

as:对象引用增加保留字，相当于绑定别名

-import 模块 as 别名（对象引用）

-(try...) except 异常 as 对象引用

-with ... as 对象引用

assert：断言语句保留字，通过异常进行程序辅助调试

-assert 判断条件

-如果条件False,产生AssertionError异常

break、continue:循环控制保留字

break:退出当前层循环

continue：退出当次循环

class:类定义保留字

class 类名：

class 派生类名（基类名）：

def :函数定义保留字

def 函数名（参数列表）：

async def 函数名（参数列表）：

if、、elif、else:条件判断保留字

条件判断：if-else ,if-elif-else

else:可以与循环和异常结合使用

try、except、finally：异常处理保留字

try-except

try-except-else-finally

for、while:循环处理保留字

遍历循环：for .. in

无限循环：while

from、import：模块导入保留字

import 模块名

from 模块名 import \*

in:成员关系操作保留字

成员关系提取：for ... in ...

成员关系判断：.. in...

is：对象判断保留字

对象一致性判断，即比较id()函数结果

以内存对象为判断依据：x is y

lambda:匿名函数保留字

定义匿名函数，产生函数对象

lambda 参数列表：表达式(返回值)

c=lambda a,b:a+b  
print(c(1,3))

pass: 空操作保留字

代表空操作，执行后不产生如何功能

在程序中用作占位符

raise:异常产生保留字

产生一个异常，辅助try-except程序功能

raise 异常名称

return:函数返回保留字

结束当前函数，返回0个或多个结果

return 返回值

with:上下文管理器保留字

构建并管理一个上下文区域

with 表达式 as 对象：

yield:生成器定义保留字

定义一个同步或异步（协程）生成器

yield 表达式

del:删除保留字

解除引用与对象的绑定关系，"删除" 引用

del 引用

global:全局声明保留字

全局变量声明，声明后可以赋值

global 变量名

nonlocal：非本地声明保留字

声明变量为当前与全局命名空间之间

nonlocal 变量名

Tru、False:布尔保留字

True：表示真

False:表示假，含0,0.0,[],{},""等

None:空无保留字

表示空无，与False不同

Python最基础的一个对象

async:协程声明保留字

声明一段协程程序

async def 协程函数名（参数名）

async with ...:

await:协程等待保留字

等待一段协程程序执行并返回结果

await 协程函数调用()

## P12 1.2精进的Python语言3

### Python特殊符号

#### 特殊符号：表达Python语法的三种逻辑：

-操作符：表达操作运算

+，-，\*，\*\*，/,//,%,

@,>>,<<,&,|,^,~

<,>,<=,>=,==,!=

-分界符：表达式之间的分界·

-特殊含义：具有特殊含义的符号

#### 操作符·

@矩阵乘法操作符尚未有功能支持\_matmul\_（）

<< 向左移n个比特，例如：x<<n

>> 向右移n个比特，例如：x>>n

& 按位与操作，例如：x&y·

| 按位或操作，例如：xly·

^ 按位异或操作，例如：x^y·

~ 按位取反操作，例如：~x

#### 分界符:

; 当多行语句在一行表示时，可以用分号分隔，但不建议

-> 函数注解，辅助对函数参数及返回值类型进行注解

...

#### 特殊含义

' " # \

#### Python语法不使用的符号

$ ? 丶e

## P13 1.2精进的Python语言4

### Python内置函数：69个

abs() chr() exec() hex() map() print() staticmethod()

all() classmethod() filter() id() max() property() str()

any() compile() float() input() memoryview() range() sum()

ascii() complex() float() input() memoryview() range() sum()

bin() delattr() frozenset() isinstance() next() reversed() tuple()

bool() dict() getattr() issubclass() object() round() type()

breakpoint() dir() globals() iter() oct() set() vars()

bytearrary() divmod() hasattr() len() open() setattr() zip()

bytes() enumerate() hash() list() ord() slice() \_import\_()

callable() eval() help() locals() pow() sorted()

## P14 1.2精进的Python语言5

### 单元小结：

1，Python保留字：35个保留字

2，Python特殊符号：->以及三个不使用的符号$ ?

3，Pythono内置函数：69个内置函数，32+37

## P15 1.3精进的Python计算生态1

### 1，Python计算生态概述

### 2，数据分析到人工智能

### 3，Web解析到网络空间

### 4，人机交互到艺术设计

## P23 1.5 Python学习路线展望1

### 1，Python知识体系

### 2，编程小白的学习路线

### 4，专业择业的学习路线

### 3，专业求职的学习路线

### 5，专业小牛的精进路线

# 安全开发部分：

测试数据：

dict={1:'a',2:'b',3:'c',4:'d'}

list=[1,2,3,4,5,6]

tup=(1,2,3,4,5,6)

set={1,2,34,5,6}

str="123456"

#安全工具开发

开始！

## 2 基础篇-Python基础语法

变量

## 0x01格式化字符串：

### 第一种方式：%s 格式化字符串

usage：

print("%s" %'hello')

%d 格式化整数

usage：

print("%d" %123)

%f 格式化浮点数字，可指定小数点后的精度

usage:

print("%.2f" %123.123)

%c 格式化字符及其ASCII码

print("%c" %65)

输出多个字符串

usages:

print("%s %s" %("hello","world"))

输出100%

print("%d%%" %100) 或 print("{}%".format(100))

### 第二种方式 :format()

print("--{}---".format("hell world"))

### 第三种方式:f’’

name="hello"

print(f"{name}")

## 0x02 print的高级用法

\033[显示方式;字体色;背景色m <接你要打印的文本> \033[显示方式;字体色;背景色m

usage:

print('\033[1;31m' + '\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*' + '\033[0m')

数据类型与转换数据类型-1

## 0x03 bool数据类型：

在python中，万物皆可bool

bool为False的几种情况：

False

None

0（整数）

0.0（浮点数）

0.0+0.0j （复数）

‘’（空字符串）

“”（空字符串）

[] （空列表）

() （空元组）

{} ｛空字典｝

ps:布尔可以进行整数运算

a=True

print(a+1)

## 0x04列表的增删改操作：

#### 列表追加元素的方法，一共有四种：add

1、append( ) 在列表的末尾追加一个元素

2、extend( ) 在列表的末尾追加多个元素

3、insert( ) 在列表的任意位置添加一个元素

4、切片 在列表的任意位置添加多个元素

list\_01 = [1, 2, 3]

print(list\_01)

list\_01.append(4)

print(list\_01)

list\_01.extend([5, 6, 7, 8])

print(list\_01)

list\_01.insert(0, 9)

print(list\_01)

list\_01[1:1] = [10, 20, 30]

print(list\_01)

#### 列表删除元素的方法，一共有五种：delete

1、remove( ) 删除指定的列表元素

2、pop( ) 删除指定列表索引的列表元素

3、del( ) 删除指定索引的列表元素，用法和pop类似

4、clear 清空列表的所有元素

5、切片 删除一个或多个列表元素

list\_01 = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

print(list\_01)

list\_01.remove(2)

print(list\_01)

list\_01.pop(0)

print(list\_01)

del list\_01[0]

print(list\_01)

list\_01[0:2] = []

print(list\_01)

list\_01.clear()

print(list\_01)

#### 列表修改元素的方法，一共有两种：update

使用索引修改单个元素

使用切片修改多个元素

list\_01[0] = 5

list\_01[0:2] = [10, 20]

## 0x05数据类型与转换数据类型-2

### 元组

Python的元组与列表类似

不同之处在于元组的元素不能修改元组使用小括号

创建空元组：

tup1 = ()

tup2 = tuple()

创建单个元素的元组：

tup1 = (1, )

### 字典

增加一个键值对

dictname[key] = value

修改一个键值对

dict['语文'] = 99

删除一个键值对

del a['语文']

1. 不允许同一个键出现两次。创建时如果同一个键被赋值两次，在python 2.x中

后一个值会被记住，但是在python 3.x中会直接报错

2. 键必须不可变，所以可以用数字，字符串或元组充当，所以用列表就不行，

但值没有限制

### 集合

集合是一个无序的不重复元素序列。

集合是一个无序的不重复元素序列。

可以使用大括号 { } 或者 set() 函数创建集合

注意：创建一个空集合必须用 set() 而不是 { }，因为 { } 是用来创建一个空字典。

使用map实现input为多个变量赋值：

a, b, c = map(int, input('请输入三个数（以空格分开）').split(' '))

print(a + b + c, type(a))

map()函数接收两个参数，一个是函数，一个是序列map（）

将传入的函数依次作用到序列的每个元素，并把结果作为新的list返回

我们将传入的函数设置为强制类型转换函数int，则split()函数返回的列表里的每一个值都将被强制

转换类型，从而实现多输入的类型转换

运算符

## 0x06逻辑运算符

and：如果其中有一个是False，那么返回值就是这个为False的计算值

如果两个都是True，那就返回y的值

or : 如果其中有一个是False，那么返回值就是这个为True的计算值

如果两个都是True，那就返回x的值

not : 如果x为True，返回False，x为False，返回True

Uages:

### "or 的使用方法：布尔或"

print(4 or 3)

print([] or '')

print([1, 2, 3, 4] or 0)

# 两个为True：返回左边这个的计算值

# 两个False：返回右边这个的计算值

# 一个True一个False,返回这个为True的计算值

### "and 的使用方法：布尔与"

print('hello' and [1, 2, 3])

print(0 and [])

print('hello' and {})

# 两个为True：返回右边这个的计算值

# 两个False：返回左边这个的计算值

# 一个True一个False,返回这个为 False的计算值

### "not 的使用方法：布尔非"

print(not '12345' )

print(not {})

# 为True：返回 False

# 为False：返回 True

## 3 基础篇-Python基础语法

公共方法和推导式

## 0x07列表生成式：

1、基本

lis = [i\*i for i in range(1, 11)]

print(lis)

2、生成式里套 if：

lis = [x\*x for x in range(1, 11) if x % 2 == 0]

print(lis)

使用 if 按条件进行筛选（筛选出x可以整除2的元素）

3、生成式多层嵌套：

lis = [i + j for i in 'ABCD' for j in 'XYZ']

print(lis)

4、生成式里套 if else：

lis = [i if i % 2 == 0 else -i for i in range(1, 11)]

print(lis)

else必须写在for的前面

## 0x08字典生成式：

1、基本

list=[1,2,3,4,5,6]

dict={ f'{i}1:{i}2' for i in list}

print(dict)

list=[('语文', 23), ('数学', 12), ('英语', 45), ('李忠', 300)]

dict={ i:j for i,j in list }

print(dict)

2、利用zip函数

dict = {key : value for (key,value) in iterable}

Usage:

keys = ['语文', '数学', '英语']

values = [23, 12, 45]

dic = {key:value for key, value in zip(keys, values)}

print(dic)

## 0x09集合生成式：与列表生成式相同，只不过是无序

s = {i \*\* i for i in range(1, 10)}

函数进阶

## 0x10高阶函数:

把函数作为参数传入，这样的函数称为高阶函数，高阶函数是函数式编程的体现。函数式编程就是指这种高度抽象的编程范式。自定义的函数也可以当做参数参入，甚至是当做一个列表的元素

Usage1:

def sum\_num(a, b, f):

return f(a) + f(b)

result = sum\_num(-1, 2, abs)

print(result) # 3

扩展：按照列表元组元素里的第一个元素排序

lis=[(1,'2'),(5,'4'),(2,'1')]

print(sorted(lis,key= lambda x : x[0] ))

结果：[(1, '2'), (2, '1'), (5, '4')]

## 0x11文件操作

fp = open("1.txt",mode='r',encoding='utf-8')

data=fp.readlines()

fp.close()

for i in data:

i=i.strip()

print(i)

完全等价：

fp = open("1.txt",mode='r',encoding='utf-8')

for i in fp:

i=i.strip()

print(i)

fp.close()

# 使用srtip()去除字符串两边的空格或换行符

例子1：os获取文件路径

import os

path=os.path.abspath(\_\_file\_\_) #获取当前py文件的绝对路径

PATH=os.path.dirname(path) #取当前py文件的上一层目录

print(path)#F:\python study\project\python\_ security\_tools\_development\123.py

print(PATH)#F:\python study\project\python\_ security\_tools\_development

例子2：上下文管理器打开多个文件

import os

PATH=os.path.dirname(os.path.abspath(\_\_file\_\_))

with open(f'{PATH}\\1.txt',mode='r',encoding='utf-8') as fp,open(f'{PATH}\\3.txt',mode='w',encoding='utf-8') as f2:

data=fp.read()

f2.write(data)

## 0x12面向对象编程

面向对象基础

只要是有这个下划线的方法，我们都叫他魔法方法，我们可以理解为创建一个类就可以用的方法

常用的魔法方法：

\_\_dict\_\_：查看当前对象的所有属性（返回一个字典对象）

\_\_str\_\_：打印这个对象

\_\_call\_\_：对象加括号 对象()

## 0x13 Mysql数据库

#聚合函数：

count 计数

avg 平均数

sum 求和

#约束查询

where

and

or

not

like 相似查询

between ti and t2 连续区间

in(18,19) #非连续区间、

#分组

group by

#排序

order by

desc 降序

asc 升序(默认)

查看mysql版本号：

mysql --version

进入mysql

mysql -uroot -proot

查看数据库：

show databases;

创建数据库：

create database test;

删除数据库：

drop database test;

选中自己创建的数据库:

use test;

查看数据库中的数据表：

show tables;

创建表：

create table pet(

name varchar(20),

age int(4),

daeth date

);

查看数据表结构：

describe pet;

插入数据：

insert into pet values ('小明',11,'1995-11-24'),('小红',11,'1995-11-24'),('小天',11,'1995-11-24');

insert into student VALUES

(1,'汉武大帝','女',18),

(2,'张良','女',18),

(3,'狄仁杰','男',19),

(4,'秦始皇','女',21),

(5,'汉高祖','男',22),

(6,'李元芳','女',18),

(7,'李艳芳','女',18),

(8,'熊大','女',18),

(9,'熊二','女',18),

(10,'孙悟空','女',18);

查询数据：

select \* from pet;

select \* from student where gender='女'

查询学生总人数

select count(id) from student;

统计学生的平均年龄

select avg(age) from student;

查询年龄18的女生的学生信息

select \* from student where age=18 and gender='女'

统计男女各多少人

select gender,count(gender) from student group by gender;

统计姓李的人有多少？

select count(name) from student where name like '李%';

select count(name) from student where name like '李元\_';

查询学生信息，年龄从高到底进行显示，年龄相同就按照id降序

select \* from student order by age desc,id desc;

not的用法：

select \* from student where not gender='男'

between的用法：

select \* from student where age between 18 and 22

修改数据：

update pet set name='小王' where name='小明';

删除数据；

delete from pet where name='小王';

课后作业

1、为萌芽实现ssh弱口令插件

**补充**

## 0.(初始化)VScode安装并设置环境

### (1)下载地址：

https://code.visualstudio.com/

### (2)设置中文：

插件搜索:chinese



### (3)配置python环境

插件搜索：python



### (4)vscoded快捷键

1）显示所有命令

ctrl+shift+p

2）自动整理代码格式

shift+alt+f

3）快速复制当前行到下一行

shift+alt+↓

4）VSCode删除整行快捷键

ctrl+shift+k

5）自动生成html代码模板

!+Tab

6）使用正则表达式替换（保留部分内容不变）的方法：

将要保留的字符串啥的用括号括起来，然后可以用($1)的方式调用还原。

例子：

172.31.50.1

172.31.50.2

172.31.51.3

172.31.51.4

正则：

(\d+\.\d+\.\d+\.)\d+

替换为：

($1)0

结果：

172.31.50.0

172.31.50.0

172.31.51.0

172.31.51.0

7）正则表达式:非贪婪式匹配

例子：

（二）Win32k 特权提升漏洞（CVE-2021-34449|CNNVD-202107- 782、CVE-2021-34516|CNNVD-202107-719）

正则：

（.+?）

结果：

（二）

或

（CVE-2021-34449|CNNVD-202107- 782、CVE-2021-34516|CNNVD-202107-719）

8）vscode查看markdown格式文件：

可以按下快捷键 Ctrl+K ，然后 V

## 1.(疑问)解释器路径

#! /usr/bin//python3 或

说明：指定解释器，路径：/usr/bin//python3，只能出现在第一行才能有作用

命令行执行时：直接 123.py 否则 python3 123.py

或#! /usr/bin/env/python3

说明：系统自动找解释器

## 2.(疑问)编码方式

#coding=utf-8

# -\*- coding: utf-8 -\*-

#coding:utf-8

说明：告诉 Python 解释器以utf-8编码方式运行，python3默认utf-8执行，python2需要指定。

## 3.(疑问)r""和b""

r"" 的作用是去除转义字符.

b" "前缀表示：后面字符串是bytes 类型。

例: response = b'<h1>Hello World!</h1>' # b' ' 表示这是一个 bytes 对象

用处：

网络编程中，服务器和浏览器只认bytes 类型数据。

如：send 函数的参数和 recv 函数的返回值都是 bytes 类型

在 Python3 中，bytes 和 str 的互相转换方式是

str.encode('utf-8')

bytes.decode('utf-8')

## 4.(扩展)创建虚拟环境：

1、首先，安装virtualenv，在默认的python2下的pip就行：  
sudo pip install virtualenv

1. 创建虚拟环境:-p:指定解释器，创建env虚拟环境文件夹  
   linux：

virtualenv -p /usr/bin/python3 env

windows：

virtualenv -p D:\python3.9.1\python3.exe env

1. 激活虚拟环境：  
   linux:

source env/bin/activate

Windows:

env\Scripts\activate.bat

1. 如果要退出python3虚拟环境，输入命令即可：  
   linux:

deactivate

Windows:

env\Scripts\deactivate.bat

## 5.(扩展)依赖的批量安装与导出

#将所有依赖库导出到 requirements.txt 文件

pip freeze > requirements.txt

安装requirements.txt依赖

pip install -r requirements.txt

## 6.(扩展)Windows平台print输出颜色文本，例：

import colorama

colorama.init(autoreset=True)

class bcolors:

    HEADER = '\033[95m'

    OKBLUE = '\033[94m'

    OKGREEN = '\033[92m'

    WARNING = '\033[93m'

    FAIL = '\033[91m'

    ENDC = '\033[0m'

    BOLD = '\033[1m'

    UNDERLINE = '\033[4m'

print(bcolors.WARNING + "警告的颜色字体?" + bcolors.ENDC)

## 7.(初始化)[VSCode](https://so.csdn.net/so/search?q=VSCode&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/weixin_44560698/article/details/_blank)找不到相对路径文件

Launch.json文件在configurations中加入语句"cwd": "${fileDirname}"即可变成常规的相对路径模式，否则是以VSCode打开工作区的根目录为当前路径。



设置前的当前路径：“F:\python study”



设置后的当前路径：

“F:\python study\project\个人\python\_deal\_json\json\_deal\_file\”



查看当前路径代码：

import os

current\_path = os.getcwd()

print(current\_path)

## 8.(初始化)选择虚拟环境解释器

注：D:\python3.9.1\路径下需有python.exe



Ctrl + Alt + P 选择 Python: Select Interpreter



Enter interpreter path



选择F:\python study\萌芽\_v3\venv\Scripts\python.exe



Vscode 右下角可以看到选择成功



已添加



## 9.(初始化)添加项目

### (1)【文件】—>【将文件夹添加到工作区】



添加成功：



### (2)选择对应launch.json文件，



### (3)选择解释器：Ctrl + Alt + P 输入：Interpreter，选择： Python: Select Interpreter



## 10.(扩展)安装依赖

比如：

python -m pip install pymysql --trusted-host mirrors.aliyun.com -i http://mirrors.aliyun.com/pypi/simple/



国内pip源

阿里云 http://mirrors.aliyun.com/pypi/simple/

中国科技大学 https://pypi.mirrors.ustc.edu.cn/simple/

豆瓣(douban) http://pypi.douban.com/simple/

清华大学 https://pypi.tuna.tsinghua.edu.cn/simple/

中国科学技术大学 http://pypi.mirrors.ustc.edu.cn/simple/

## 11.(疑问)如何判断是类的实例化还是函数调用？

类名的命名规则通常遵循首字母大写的命名规范，而函数名通常使用小写字母和下划线。

## 12.(疑问)各种print(type(xx))

### 1）(疑问)print(type(z))，输出：<class '\_\_main\_\_.Abc'> ，（实例）

这表示z是Abc类的实例，Abc是当前脚本中定义的类名，因此它的类型是来自当前脚本的Type对象。

### 2）(疑问)print(type(Abc))，输出：<class 'type'>，（类）

这表示Abc是一个Type对象，表示一个类的类型。

### 3）(疑问)print(type(b))，输出：<class 'function'>，（函数名）

这表示b是某个函数的函数名

### 4）(疑问)print(type(argparse))，输出:<class 'module'>，（.py文件）

这表示argparse是一个模块，即是.py文件

### 5）(疑问)print(type(data))，输出：<class 'bytes'>，（二进制数据流）

这表明 data 变量是一个 bytes 类型的对象，即二进制数据

如：

data = b'\x00\x01\x02\x03'

print(type(data)) # <class 'bytes'>

## 13.(疑问)python3中，import 模块名，一般用vscode按住ctrl+点击可以跳转至对应的.py文件，但是，点击sys模块时，跳转的是.pyi文件，看不到.py文件，这是为什么？

在Python 3中，一些内置模块（如sys、os、io等）的实现是用C语言编写的，并且它们的源代码不在Python解释器的安装目录中。这些模块的源代码通常是作为Python解释器的一部分编译的，因此你不能像导入其他Python模块一样导入它们的源代码。

为了解决这个问题，Python官方提供了一个叫做“类型注释”的特性，用于对内置模块的API进行类型注释和文档化。类型注释文件的扩展名为.pyi，它们类似于常规的Python源代码，但只包含类型注释和文档字符串，而不包含实际的代码实现。

因此，当你在VS Code中导入sys模块并尝试跳转到源代码时，VS Code会自动跳转到sys.pyi文件，而不是实际的C实现代码文件。这是因为VS Code通过类型注释文件提供了内置模块的类型信息和文档，以便你能够使用自动补全和其他功能来更好地了解模块的API。

总之，跳转到sys.pyi文件是正常的行为，因为它是内置模块的类型注释文件，而不是实际的源代码文件。如果你想查看sys模块的源代码实现，你需要查找Python解释器的源代码，或者查看Python官方文档中的sys模块文档。

## 14.(初始化)VSCode 配置快速输入Python的Main函数方法

【文件】->【首选项】->【配置用户代码片段】



输入python选择python.json



添加以下信息:

"Print to console": {

        "prefix": "pymain",

        "body": [

            "if \_\_name\_\_ == '\_\_main\_\_':",

            "    ${1:pass}",

            "",

        ],

        "description": "python--main"

解释：

"prefix"：快速输入的关键字是pymain

如图：



## 15.(初始化)vscode打开代码提示

设置搜索‘prevent’,取消勾选‘控制活动代码是否阻止快速建议’。似乎不设置也有代码提示



## 16.(疑问)什么是命名空间

在 Python 3 中，命名空间是一个存储变量名称和对象的映射关系的数据结构。每个变量都必须在一个命名空间中定义，命名空间可以理解为一个作用域，即变量名称在该作用域内有效。不同的命名空间可以相互独立存在，避免了变量名称冲突的问题。

Python 3 中有以下几种类型的命名空间：

1. 内置命名空间：包含 Python 内置的函数、模块和常量等，这些对象在解释器启动时自动加载，无需进行导入。
2. 全局命名空间：包含程序中定义的全局变量和函数等，可以在程序的任何地方访问。
3. 局部命名空间：包含在函数内部定义的变量和函数等，只能在函数内部访问，函数调用结束后局部命名空间被销毁。
4. 表达式命名空间：在执行表达式时动态生成的命名空间，包含表达式中使用的变量和函数等。

在 Python 3 中，当你使用一个变量时，解释器会按照以下顺序在命名空间中查找该变量：

1. 当前作用域的局部命名空间。
2. 当前作用域之外的全局命名空间。
3. 内置命名空间。

如果在任何一个命名空间中找到了该变量，解释器就会停止查找并使用该变量。如果在所有的命名空间中都没有找到该变量，则会引发 NameError 异常。

## 17.(疑问)Window下平台编写的Python代码移植Linux平台运行，最大的问题是什么？

在 Windows 平台编写的 Python 代码移植到 Linux 平台运行，最大的问题可能是文件路径和行结束符的不同。

### 1）文件路径问题：

在 Windows 平台上，文件路径使用反斜杠 \ 分隔，例如 C:\Users\example\file.txt，而在 Linux 平台上，文件路径使用正斜杠 / 分隔，例如 /home/example/file.txt。因此，如果在代码中使用了硬编码的文件路径，可能需要手动修改这些路径，或者**使用 Python 内置的 os.path 模块来处理路径。**

解决：

使用 Python 内置的 os.path 模块来处理路径。

比如：

sys.path +=['project1\pkg1']

改为：

import os

pkg\_path = os.path.join('project1', 'pkg1')

sys.path.append(pkg\_path)

### 2）行结束符问题：

在 Windows 平台上，行结束符是 "\r\n"，而在 Linux 平台上，行结束符是 "\n"。如果在代码中使用了硬编码的行结束符，可能会导致在 Linux 平台上出现意想不到的错误。可以**使用 Python 的 open() 函数来打开文件并自动识别行结束符**，或者使用 Python 的 io 模块来处理文件的读写。

解决:

使用 Python 的 open() 函数来打开文件并自动识别行结束符

### 3）第三方库是否在 Linux 平台上可用

解决：

* 确认第三方库在 Linux 平台上的可用性：在 Linux 上尝试安装和使用这些库，并查看是否存在与 Windows 上不同的问题。
* 使用虚拟环境：在 Linux 上创建一个虚拟环境，安装 Python 和依赖包，以便代码在此环境下运行。这样可以避免与系统已安装的库发生冲突。
* 使用包管理工具：像 pip 和 conda 这样的包管理工具可以帮助在 Linux 上安装和管理 Python 包，可以方便地安装和更新第三方库。
* 手动安装：在 Linux 上手动安装第三方库。可以从官方网站下载源代码或二进制文件，按照安装说明进行安装。

### 4）**是否与在 Linux 平台上使用的编码格式（比如：常用的编码格式有 ASCII、UTF-8、GBK 等）一致**

解决：

* 在代码文件中指定编码格式，使用 Python 的编码声明来指定文件的编码格式，例如在文件开头添加如下声明：“**# -\*- coding: utf-8 -\*-”**
* 使用文本编辑器转换编码格式，将 Windows 平台上的代码文件转换为 Linux 平台上的编码格式。在 Windows 平台上，常用的编码格式为 GBK 或者 GB2312，而在 Linux 平台上，常用的编码格式为 UTF-8。可以使用文本编辑器（例如 Notepad++）来转换编码格式。

## 18.(疑问)Python3 -m pip install安装的包安装路径在哪？

**site-packages目录**是Python安装后自动创建的一个目录，用于存储第三方包。

## (疑问)在导入 Python 模块时，Python 解释器会按照以下顺序查找模块所在的路径：

#### 1）当前目录：

Python 解释器会首先在当前工作目录中查找模块所在的路径。

#### 2）环境变量 PYTHONPATH：

如果 Python 解释器没有在当前目录找到该模块，则会从 PYTHONPATH 环境变量所指定的路径中查找。

#### **3）sys.path中指定的路径**：

Python解释器会查找环境变量PYTHONPATH所包含的路径，以及默认路径（如安装Python时默认指定的路径），在这些路径下查找是否有该模块。

#### 4）site-packages目录：

site-packages是第三方库的安装目录，Python解释器也会在这个目录下查找是否有该模块。

#### 5）sys.path中自定义添加的路径。

**总结：其实路径都在sys.path包含的路径里，执行顺序也一样**



## (疑问)为什么要用命名空间包，又为什么用常规包

**命名空间包**的主要作用是解决包名冲突的问题。当多个包具有**相同的名字**时，通过命名空间包可以将它们组织在一起，使它们能够共存并且不会冲突。此外，命名空间包还可以方便地将多个独立的包组织在一起形成一个大的逻辑单元，使代码更加清晰易懂。

比如：

分散路径1：C:\Users\Administrator\Desktop\project1

project1/

├── pkg1

│   └── m1.py

/project1/pkg1/m1.py:

def mecho (any):

    print("pkg1-m1:{}".format(any))

分散路径2：C:\Users\Administrator\Desktop\project2

project2/

├── pkg1

│   └── m3.py

/project2/pkg1/m3.py:

def mecho (any):

    print("pkg1-m3:{}".format(any))

开始使用（任意路径）：test.py:

#命名空间包：pkg1 和 pkg2

import sys

sys.path+=[r'C:\Users\Administrator\Desktop\project1',r'C:\Users\Administrator\Desktop\project2'] #添加命名空间包所在路径

import pkg1.m1              #导入

import pkg1.m3              #导入

pkg1.m1.mecho("hello")     #引用

pkg1.m3.mecho("world")     #引用

**常规包**则是最常见的包形式，它可以包含多个模块和子包，可以通过 import 语句直接引用包中的模块和函数。常规包的主要优点是使用方便，不需要特别的设置，可以直接引用包中的模块和函数。

通常情况下，如果包名不会和其他包发生冲突，我们可以直接使用常规包来组织代码。但是如果存在命名冲突的情况，或者需要将多个包组织在一起形成一个大的逻辑单元，使用命名空间包会更加合适。

## 21.(疑问)import使用的常见用法：标绿的与以上官方的方式一致

1. import module\_name: 导入一个模块，可以通过 module\_name.func() 的方式来使用模块中的函数或变量。**与被导入文件同级**

**导入：**import module\_name

**引用：**module\_name.func()

1. from module\_name import func\_name: 从模块中导入一个指定的函数或变量，可以直接使用 func\_name() 调用函数或者使用变量。**与被导入文件同级**

**导入：**from module\_name import func\_name:

**引用：**func\_name()

3）from module\_name import \*: 导入模块中的所有函数和变量，可以直接使用函数名调用函数，或者使用变量名引用变量。但这种方式并不推荐使用，因为容易产生命名冲突，使代码不易维护。**与被导入文件同级**

**导入：**from module\_name import \*

**引用：**func\_name()

4）from package\_name import \*:需要在包下的\_\_init\_\_.py文件中增加\_\_all\_\_属性赋值,如：\_\_all\_\_ = ['m1','m2']。**与被导入的常规包同级**

**导入：**from package\_name import \*

**引用：**m1.func1()

5）import module\_name as alias：将整个模块导入到当前作用域中，并将其重命名为指定的别名，引用方式为alias.function\_name、alias.class\_name或alias.variable\_name。**与被导入的文件同级**

**导入：**import module\_name as alias

**引用：**alias.function\_name

6）from package\_name import module\_name: 从包中导入一个指定的模块，可以通过 module\_name.func() 的方式来使用模块中的函数或变量。**与被导入的常规包/命名空间包同级**

**导入：**from package\_name import module\_name

**引用：**module\_name.func()

7）import package\_name.module\_name: 导入一个包中的指定模块，可以通过 package\_name.module\_name.func() 的方式来使用模块中的函数或变量。**与被导入的常规/命名空间包同级**

**导入：**import package\_name.module\_name

**引用：**package\_name.m odule\_name.func()

8）from package\_name.module\_name import func\_name: 从包中的指定模块中导入一个指定的函数或变量，可以直接使用 func\_name() 调用函数或者使用变量。**与被导入的常规/命名空间包同级**

**导入：**from package\_name.module\_name import func\_name

**引用：**func\_name()

9）（偏门）import package\_name:例如：mypkg包下有module1.py模块，module1.py模块里有func1()函数，需要在包下添加\_\_init\_\_.py文件，且添如下代码：from . import module1。**与被导入的常规包同级**

**导入：**import mypkg

**引用：**mypkg.module1.func1()

**注意：**导入包或模块时（即加载import语句时），会自动执行一遍，如果是模块则会执行模块，如果是常规包会执行常规包的py形式(即：\_\_init\_\_.py)，命名空间包没有\_\_init\_\_.py文件。

## 22.(疑问)代码提示：

### 1）对象

比如：

class MyClass:

    pass

my\_object = MyClass()

print(type(my\_object))

输出：

<class’\_\_main\_\_.MyClass’>

my\_object

提示：(variable) my\_object: MyClass

解释：（variable 对象名:类名）

### 2）类名

MyClass

提示：(class) MyClass

### 3）方法名

提示：(method) def hexdigest() -> str

## 23.(扩展)vscode设置代理，设置后重启vscode即可



## 24.(初始化)设置自动换行

【设置】——>搜索【Editor:Word Wrap】，将off改为【on】



## 25.(疑问)迭代器()：for ip in IPSet([target])，IPSet不是类吗，为什么可以像列表一样 for遍历呢？

因为IPSet类实现了迭代器协议，即实现了 iter() 方法以及迭代器对象必须实现的next() 方法。当使用 for i in IPSet([target]) 时，Python 会自动调用 IPSet 实例的 iter() 方法来获取一个迭代器对象，然后循环调用该迭代器对象的 next() 方法来逐个获取 IPSet 实例中的元素。

例子：自定义类来实现迭代器协议

class MyIterator:

    def \_\_init\_\_(self, start, end):

        self.current = start

        self.end = end

    def \_\_iter\_\_(self):

        return self

    def \_\_next\_\_(self):

        if self.current < self.end:

            value = self.current

            self.current += 1

            return value

        else:

            raise StopIteration

# 使用自定义的 MyIterator 类实现迭代器

my\_iterator = MyIterator(1, 5)

# 使用 for 循环遍历迭代器对象

for num in my\_iterator:

print(num)

# 输出结果：

# 1

# 2

# 3

# 4

## 26.(初始化) shift+alt+f 自动整理代码格式，注释相关设置

.vscode文件夹下，新建settings.json文件，并添加如下内容：单行最大长度

{

    "python.formatting.autopep8Args": [

        "--max-line-length=500",

    ]

}

如果你希望在使用 Autopep8 格式化工具时，不要自动在注释符号 "#" 后面添加空格，可以使用 --ignore 参数来排除 E26 规则。该规则是与注释相关的规则。

在 "python.formatting.autopep8Args" 设置中添加以下内容：

{

    "python.formatting.autopep8Args": [

        "--max-line-length=500",

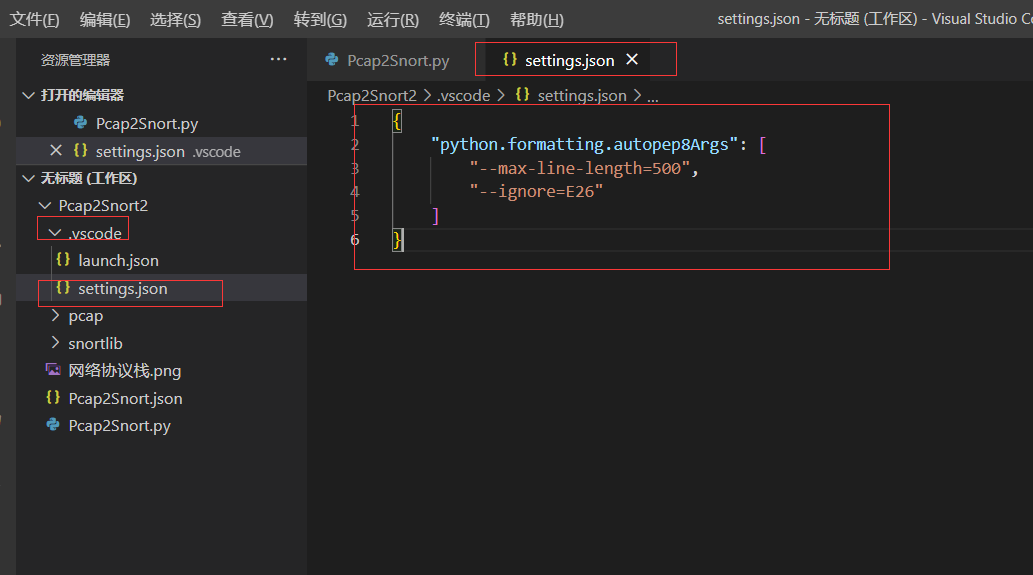
        "--ignore=E26"

    ]

}

这样设置后，Autopep8 在格式化代码时将忽略与注释有关的规则，不会自动在 "#" 后面添加空格。

如图：



## 27.(疑问)生成器:生成器函数和普通函数的区别，看起来就是多了一个yield关键字?

yield是Python中用于定义生成器函数的关键字。生成器函数与普通函数的区别在于，生成器函数在执行过程中可以暂停并在下一次调用时恢复执行，而普通函数则是一次性执行完毕。

在生成器函数中使用yield关键字，可以将函数执行的状态保存下来，并返回一个生成器对象。每次调用生成器的\_\_next\_\_()方法或使用for...in...循环遍历生成器对象时，都会从上一次暂停的位置开始执行，直到遇到下一个yield语句时再次暂停。

例如：

def even\_numbers(n): #生成从0开始的偶数序列：

    for i in range(n):

        if i % 2 == 0:

            yield i #yield关键字，简单理解为：返回值并暂停

for i in even\_numbers(10): #使用这个生成器函数可以得到一个偶数序列：

     print(i)

#输出：

#0

#2

#4

#6

#8

在这个示例中，even\_numbers()函数返回了一个生成器对象，而不是一个完整的列表或元组等数据结构。在每次调用\_\_next\_\_()方法或使用for...in...循环遍历生成器对象时，函数执行到yield语句处并返回一个数，然后暂停并等待下一次调用，直到函数执行完毕。

## (疑问）关键字解释：0-255 范围、2^8、8位、字节、二进制、十进制

含义：

单个字节：8位

十进制数表示：

最小：0

最大：255

数量：256个（0-255），计算2^8,举例:2位的二进制数可以表示多少个：00 01 10 11,2^2=4个

## (疑问)for 循环 字节串，输出的是十进制数？

是的，当使用 for 循环遍历字节串时，可以得到每个字节对应的十进制数值。例如，在 Python 中可以使用以下代码来展示字节串中每个字节的十进制值：

byte\_string = b"hello"

for b in byte\_string:

    print(b)

以上代码将输出以下内容：

104

101

108

108

111

其中，每行的数字是每个字节的十进制表示。

#### (疑问）难道不是b'h'、 b'e'、b'l'、b'l'、b'o'?

答：这是在 Python 中打印字节串时看到的结果，这是因为 Python 将字节串中的每个字节以 ASCII 字符的形式展示出来。

但是，实际上字节串中存储的是二进制数据，可以通过遍历每个字节并将其转换为十进制数值来获取其对应的数值。如果你想在 Python 中同时输出每个字节的字符形式和十进制数值，可以使用以下代码：

byte\_string = b"hello"

for b in byte\_string:

    print(f"{chr(b)} ({b})")

以上代码将输出以下内容：

h (104)

e (101)

l (108)

l (108)

o (111)

## 30.(疑问)十六进制数对应的字符

例如：64

>>> print(chr(0x64))

d

>>>

## 31.“以文本模式打开文件并写入文本数据”和“以二进制模式打开同一个文件并写入相同的数据”有什么区别？

# 以文本模式打开文件并写入文本数据

with open('test.rules', 'w') as f:

    f.write('Hello, world!\n')

# 以二进制模式打开同一个文件并写入相同的数据

with open('test.rules', 'wb') as f:

    f.write(b'Hello, world!\n')

答：的确，在某些情况下，使用文本模式和二进制模式写入文件可能会达到相同的效果。但这并不意味着这两种模式是可以互换的。以下是一个举例来区分这两种模式的情况:

假设我们需要写入一张图片文件。如果我们使用文本模式打开图像文件并尝试将其写入磁盘，那么我们最终写入的文件将不是原始的二进制数据。这是因为在文本模式下，Python默认将特定字节（例如0x0A）转换为操作系统所需的换行符，这可能会导致我们失去图像文件的完整性。而在二进制模式下，我们可以将二进制数据准确地写入文件，从而保留了图像文件的完整性。

因此，虽然在一些情况下使用文本模式和二进制模式可能会达到相同的效果，但在处理二进制数据时，使用二进制模式是更好的选择，以便确保数据的完整性。