**目录**

[一、 阅读说明 4](#_Toc1754321)

[二、 Linux帮助命令 4](#_Toc1754322)

[1. 帮助文档基本说明 4](#_Toc1754323)

[2. whatis 4](#_Toc1754324)

[3. 内部命令（builtin）与外部命令 4](#_Toc1754325)

[4. 查看内部命令使用方法 4](#_Toc1754326)

[5. 查看外部命令使用方法 4](#_Toc1754327)

[6. man手册中字段说明 5](#_Toc1754328)

[7. man手册查看方式 5](#_Toc1754329)

[8. man手册章节 6](#_Toc1754330)

[三、 查找命令 6](#_Toc1754331)

[1. find 6](#_Toc1754332)

[2. locate 6](#_Toc1754333)

[3. which 6](#_Toc1754334)

[4. whereis 6](#_Toc1754335)

[5. type 7](#_Toc1754336)

[6. 查看环境变量 7](#_Toc1754337)

[四、 Linux根目录结构 7](#_Toc1754338)

[1. 向现有表插入数据 8](#_Toc1754339)

[2. 查看某一列 9](#_Toc1754340)

[3. 复制数据表 9](#_Toc1754341)

[4. 删除数据表 9](#_Toc1754342)

[5. 更改数据表 9](#_Toc1754343)

[6. 重命名数据表 10](#_Toc1754344)

[7. 移动数据表至另一数据库 10](#_Toc1754345)

[8. 重排序数据表 11](#_Toc1754346)

[9. \_\_str\_\_与\_\_repr\_\_方法 11](#_Toc1754347)

[10. 查找模块下的方法(函数)、属性 11](#_Toc1754348)

[11. Import与from…import…的区别 11](#_Toc1754349)

[12. Tuple与()的使用区别 11](#_Toc1754350)

[13. Python时间格式化输出 12](#_Toc1754351)

[14. 格式化输出 12](#_Toc1754352)

[15. Windows环境下文件路径表示 12](#_Toc1754353)

[16. Python接收命令行参数 12](#_Toc1754354)

[17. '\u'前缀字符串 12](#_Toc1754355)

[18. 定义1个元素的tuple 13](#_Toc1754356)

[19. 创建生成器(generator)的两种方式 13](#_Toc1754357)

[20. python代码规范 13](#_Toc1754358)

[21. 迭代器总结 13](#_Toc1754359)

[22. Map 函数总结 13](#_Toc1754360)

[23. Reduce函数总结 14](#_Toc1754361)

[24. 匿名函数(lambda)总结 14](#_Toc1754362)

[25. 全局变量的使用 14](#_Toc1754363)

[26. python 定义类时，内部方法的互相调用 15](#_Toc1754364)

[27. filter函数总结 15](#_Toc1754365)

[28. sorted函数总结 15](#_Toc1754366)

[29. 返回函数(闭包) 16](#_Toc1754367)

[30. 16](#_Toc1754368)

[31. 16](#_Toc1754369)

[32. 装饰器 16](#_Toc1754370)

[33. 函数参数 16](#_Toc1754371)

[34. 字符串里面的引号 17](#_Toc1754372)

[35. 偏函数 17](#_Toc1754373)

[36. 算法速度的定义 17](#_Toc1754374)

[37. 计算运行时间的一般法则 17](#_Toc1754375)

[38. 导入自定义模块 18](#_Toc1754376)

[39. 子类继承父类的属性问题 18](#_Toc1754377)

[40. 子类扩展父类属性 19](#_Toc1754378)

[41. 附录 20](#_Toc1754379)

[1. 方法解析顺序（Method Resolution Order, MRO）列表 20](#_Toc1754380)

[7. 查询模块的帮助文档 20](#_Toc1754381)

[8. python中时间日期格式化符号 20](#_Toc1754382)

1. 阅读说明

绿色斜体代表个人的思考理解，黄色斜体代表阅读理解过程中的疑问，红色正体代表关键重要信息，下划线代表次关键重要信息

1. Linux帮助命令
2. 帮助文档基本说明

* [] 可选内容
* <> 必选内容
* a | b 二选一
* { } 分组
* ... 同一内容可出现多次

1. whatis

输入“whatis echo”，会显示如下信息。每行包含三部分，第一部分是命令名称；第二部分是命令在man手册出现的位置，第三部分是简述命令或函数的作用。

1. **>>>whatis echo**
2. **>>>echo (1)** - display a line of text

如果想详细了解命令信息，可以输入如下命令：

1. **>>>man 1 echo** 或者 **man echo**(仅当数字为1时候可省略，其他数字不可省略)
2. 内部命令（builtin）与外部命令

* man bash： NAME字段后面的命令都是内部命令

1. 查看内部命令使用方法

* help COMMAND 显示COMMAND这个命令的用法
* man help 显示所有内部命令列表及使用方法

1. 查看外部命令使用方法

* COMMAND --help

1. **>>>bash –-help //**部分命令，也可以简写为COMMAND -h

* man COMMAND 原始格式为：man [章节] COMMAND，默认第1章省略

1. **>>>man bash**

* info COMMAND

1. **>>>info bash**
2. man手册中字段说明
3. NAME 名称及简要说明
4. SYNOPSIS 用法格式说明
   * [] 可选内容
   * <> 必选内容
   * a|b 二选一
   * { } 分组
   * ... 同一内容可出现多次
5. DESCRIPTION 详细说明
6. OPTIONS 选项说明
7. EXAMPLES 示例
8. FILES 相关文件
9. AUTHOR 作者
10. COPYRIGHT 版本信息
11. REPORTING BUGS bug信息
12. SEE ALSO 其它帮助参考
13. man手册查看方式

* q Q ZZ 退出
* g 1g 光标跳至文档首部
* G 光标跳至文档尾部
* e j 文档前进N行
* y k 文档后退N行
* f space 文档前进N页
* b ^B 文档后退N页
* /pattern n/N 向后查询、
* ?pattern 向前查询
* &pattern 只显示匹配到的行

1. man手册章节
2. User Commands // 用户命令
3. System Calls // 系统调用
4. C Library Functions // C函数库调用
5. Devices and Special Files // 设备文件和特殊文件
6. File Formats and Conventions // 配置文件及格式
7. Games et. Al. // 游戏
8. Miscellanea // 杂项
9. System Administration tools and Deamons // 管理类命令
10. 查找命令
11. find

find <指定目录> <指定条件> <指定动作>

- <指定目录>： 所要搜索的目录及其所有子目录。默认为当前目录。

- <指定条件>： 所要搜索的文件的特征。

- <指定动作>： 对搜索结果进行特定的处理。

1. **find . -name ‘my\*’ -ls**
2. locate

locate命令其实是“find -name”的另一种写法，它不搜索具体目录，而是搜索一个数据库（/var/lib/locatedb），这个数据库中含有本地所有文件信息。Linux系统自动创建这个数据库，每天自动更新一次，所以使用locate命令查不到最新变动过的文件。为了避免这种情况，可以先使用updatedb命令手动更新数据库再使用locate。

1. which

which命令的作用是，在PATH变量指定的路径中，搜索某个系统命令的位置，并且返回第一个搜索结果.也就是说，使用which命令，就可以看到某个系统命令是否存在，以及执行的到底是哪一个位置的命令.

1. whereis

whereis命令只能用于程序名的搜索，而且只搜索二进制文件（参数-b）、man说明文件（参数-m）和源代码文件（参数-s）.如果省略参数，则返回所有信息.

1. type

type命令其实不能算查找命令，它是用来区分某个命令到底是由shell自带的，还是由shell外部的独立二进制文件提供的。如果一个命令是外部命令，那么使用-p参数，会显示该命令的路径，相当于which命令.

1. 查看环境变量

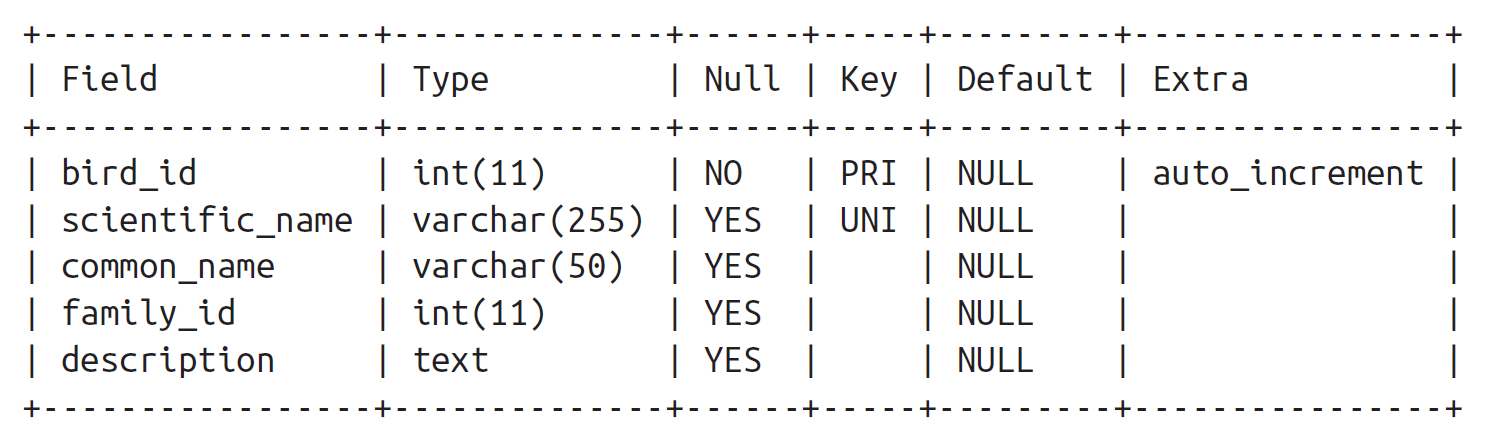
env命令

1. Linux根目录结构

|  |  |  |
| --- | --- | --- |
| bin | 存放普通用户可执行的指令 | 即使在单用户模式下也能够执行处理 |
| boot | 开机引导目录 | 包括Linux内核文件与开机所需要的文件 |
| dev | 设备目录 | 所有的硬件设备及周边均放置在这个设备目录中 |
| etc | 各种配置文件目录 | 大部分配置属性均存放在这里 |
| lib/lib64 | 开机时常用的动态链接库 | bin及sbin指令也会调用对应的lib库 |
| media | 可移除设备挂载目录 | 类似软盘 U盘 光盘等临时挂放目录 |
| mnt | 用户临时挂载其他的文件系统 | 额外的设备可挂载在这里,相对临时而言 |
| opt | 第三方软件安装目录 | 现在习惯性的放置在/usr/local中 |
| proc | 虚拟文件系统 | 通常是内存中的映射,特别注意在误删除数据文件后，比如DB，只要系统不重启,还是有很大几率能将数据找回来 |
| root | 系统管理员主目录 | 除root之外,其他用户均放置在/home目录下 |
| run | 系统运行是所需文件 | 以前防止在/var/run中,后来拆分成独立的/run目录。重启后重新生成对应的目录数据 |
| sbin | 只有root才能运行的管理指令 | 跟bin类似,但只属于root管理员 |
| snap | ubunut全新软件包管理方式 | snap软件包一般在/snap这个目录下 |
| srv | 服务启动后需要访问的数据目录 |  |
| sys | 跟proc一样虚拟文件系统 | 记录核心系统硬件信息 |
| tmp | 存放临时文件目录 | 所有用户对该目录均可读写 |
| usr | 应用程序放置目录 |  |

* 方法一：DESCRIBE命令

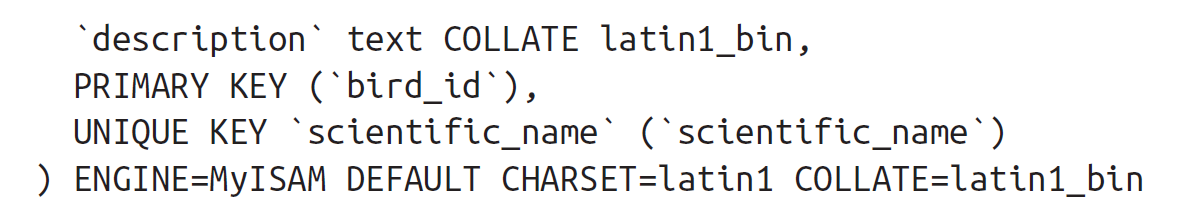
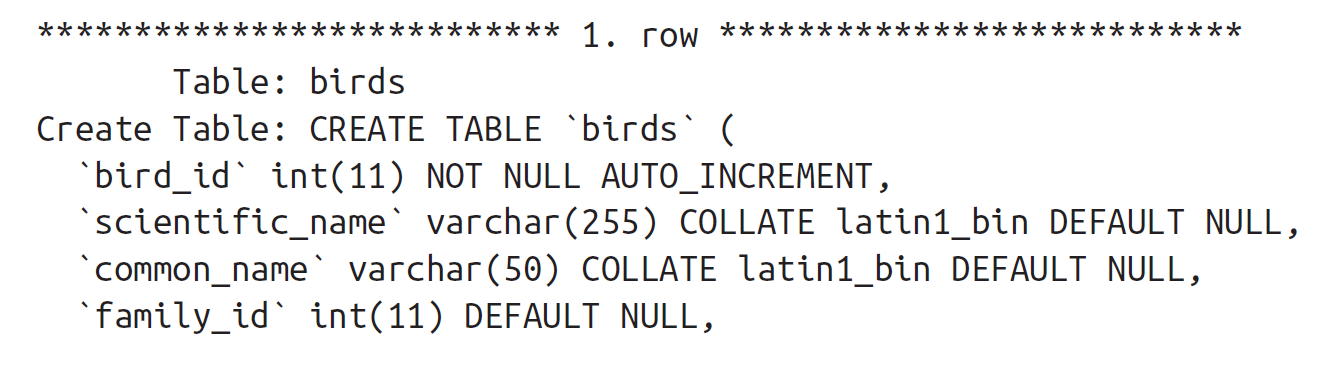
1. **DESCRIBE table\_name;**



第一列为列名。第二列为类型，第三列用于说明各列能否含有NULL值，第四列说明该列是否为键(key)即索引列，第五列用于说明各列默认值，最后一列为备注，额外信息。

* 方法二：SHOW CREATE TABLE 命令

1. **SHOW CREATE TABLE table\_name;**
2. **SHOW CREATE TABLE table\_name \G** //用\G代替分号，结果将不以表格显示



1. 向现有表插入数据
2. INSERT INTO birds (scientific\_name, common\_name)
3. VALUES ('Charadrius vociferus', 'Killdeer'),
4. ('Gavia immer', 'Great Northern Loon'),
5. ('Aix sponsa', 'Wood Duck'),
6. ('Chordeiles minor', 'Common Nighthawk'),
7. ('Sitta carolinensis', ' White-breasted Nuthatch'),
8. ('Apteryx mantelli', 'North Island Brown Kiwi');

命令基本格式：**INSERT INTO** table\_name (list1, list2) **VALUES** ('Charadrius vociferus', 'Killdeer'),('Gavia immer', 'Great Northern Loon');

1. 查看某一列
2. SHOW COLUMNS FROM table\_name LIKE 'list\_name' \G
3. 复制数据表

* 仅复制表结构

1. CREATE TABLE table\_name\_new LIKE table\_name
2. INSERT INTO table\_name\_new SELECT \* FROM table\_name //从原表复制数据插入

* 通过建表实现复制表结构及数据(该做法不会将设定(PRIMARY KEY等)复制过来)

1. CREATE TABLE table\_name\_new SELECT \* FROM table\_name //复制全部数据
2. CREATE TABLE table\_name\_new SELECT list\_name1, list\_name2 FROM table\_name //复制其中两列
3. 删除数据表
4. DROP TABLE table\_name
5. 更改数据表

更改表的基本语法: ALTER TABLE table\_name changes, table\_name为表名，changes为具体更改命令/更改子句，常见的子句包含：CHANGE/ADD/ALTER/MODIFY/DROP/ORDER BY

* 增加列(在list\_name2列后增加)

1. ALTER TABLE table\_name ADD COLUMN list\_new INT AFTER list\_name2;
2. ALTER TABLE table\_name ADD COLUMN list\_new INT FIRST //插入为第一列

* 增加多列

1. ALTER TABLE table\_name
2. ADD COLUMN body\_id CHAR(2) AFTER wing\_id,
3. ADD COLUMN bill\_id CHAR(2) AFTER body\_id,
4. ADD COLUMN endangered BIT DEFAULT b'1' AFTER bill\_id;

* 删除列

1. ALTER TABLE table\_name DROP COLUMN list\_name;

* 修改列(非修改表中数据的内容)

1. ALTER TABLE table\_name CHANGE COLUMN list\_name1 list\_name2 VARCHAR(255);

如果修改列名，则填入新列名list\_name2，否则仅填入原列名list\_name1即可，修改类型同理

* 修改列的类型(非前述方法，此方法不可修改列名)

1. ALTER TABLE table\_name
2. MODIFY COLUMN list\_name
3. ENUM('Extinct',
4. 'Extinct in Wild',
5. 'Threatened - Critically Endangered',
6. 'Threatened - Endangered',
7. 'Threatened - Vulnerable',
8. 'Lower Risk - Conservation Dependent',
9. 'Lower Risk - Near Threatened',
10. 'Lower Risk - Least Concern')
11. AFTER family\_id;

修改列的类型为枚举类型(ENUM)

* 修改表中数据内容

1. UPDATE table\_name SET endangered=0 WHERE bird\_id IN(1,2,4,5)

* 修改默认值

方法1——采用CHANGE 子句

1. ALTER TABLE table\_name CHANGE COLUMN list\_name INT DEFAULT 8;

方法二——采用ALTER子句

1. ALTER TABLE table\_name ALTER list\_name SET DEFAULT 7;

* 设置AUTO\_INCREMENT初始值

1. ALTER TABLE table\_name AUTO\_INCREMENT = 10;

Auto\_increment的值来自于information\_schema数据库中tables数据表，tables表中就有一列叫auto\_increment，新建一个数据表则tables表对应增加一行

1. 重命名数据表
2. RENAME TABLE table\_name\_old TO table\_name\_new;
3. 移动数据表至另一数据库
4. RENAME TABLE databse1.table1 TO database2.table2；

RENAME的用法与linux中mv用法类似，兼具重命名和移动数据表的功能。

1. 重排序数据表
2. ALTER TABLE table\_name ORDER BY list\_name;

super() 函数是用于调用父类(超类)的一个方法。单继承：super 与直接用类名调用父类方法无异；多继承，会涉及到查找顺序（MRO）、重复调用（钻石继承）等问题，此时应区分super与直接类名调用父类。

MRO 就是类的方法解析顺序表, 其实也就是继承父类方法时的顺序表(见[附录1](#MRO))

1. \_\_str\_\_与\_\_repr\_\_方法

\_\_repr\_\_：repr() 函数将对象转化为供解释器读取的形式，返回一个对象的string格式

\_\_str\_\_：str() 函数将对象转化为适于人阅读的形式，返回一个对象的string格式，

两个方法均用于返回对象供人阅读，\_\_str\_\_()用于显示给用户，而\_\_repr\_\_()用于显示给开发人员。print调用的是\_\_str\_\_方法，直接输出实例调用的是\_\_repr\_\_方法

1. >>> class Student(object):
2. ... def \_\_init\_\_(self, name):
3. ... self.name = name
4. ...
5. >>> print(Student('Michael'))
6. <\_\_main\_\_.Student object at 0x109afb190>
7. **>>>** s **=** Student**(**'Michael'**)**
8. >>> s
9. **<**\_\_main\_\_**.**Student object at 0x109afb310**>**
10. 查找模块下的方法(函数)、属性

均在python解释器下查询，详见[附录2.查询模块的帮助文档](#查询帮助)

1. Import与from…import…的区别
2. Tuple与()的使用区别

tuple( seq )，seq -- 要转换为元组的序列。用法如下：

1. aList **=** **[**123**,** 'xyz'**,** 'zara'**,** 'abc'**];**
2. aTuple **=** tuple**(**aList**)**
3. **print** "Tuple elements : "**,** aTuple
4. **>>>**Tuple elements **:** **(**123**,** 'xyz'**,** 'zara'**,** 'abc'**)**

()则直接使用：

1. **aTuple =** **(**123**,** 'xyz'**,** 'zara'**,** 'abc'**)**
2. Python时间格式化输出

strftime()函数接收以时间元组(struct\_time对象)，并返回以可读字符串表示的当地时间，格式由参数format决定

1. t **=** time**.**time**()//**获得以秒为单位的时间
2. **print(**time**.**strftime**(**"%b %d %Y %H:%M:%S"**,** time**.**gmtime**(**t**)))//**gmtime获得struct\_time对象

格式参数format，详见[附录3.python中时间日期格式化符号](#日期格式)

1. 格式化输出

格式输出详见参考文档 [《python的格式化输出》](file:///C:\Users\Alta\Desktop\编程笔记\python的格式化输出.pdf)

1. Windows环境下文件路径表示

Python代码里面，反斜杠“\”是转义符，例如“\n”表示回车，采用以下三种方式表示路径：

* 斜杠“/”，如“c:/test.txt”
* 两个反斜杠“\\”，如“c:\\test.txt”
* 字符串前面加上字母r，表示后面是一个原始字符串raw string，如“r“c:\\test.txt””

1. Python接收命令行参数

导入argv，结果即为参数列表

1. **from** sys **import** argv
2. **print(**argv**)**
3. **>>>**python xx**.**py xxx
4. **>>>[**'xx.py'**,**'xxx'**]**
5. '\u'前缀字符串

\u4f60十六进制代表对应汉字的utf-16编码

1. 定义1个元素的tuple

定义1个元素的tuple: t = (1,)，加上一个逗号，避免成为数学意义上的括号

1. 创建生成器(generator)的两种方式

方式1: g = (x \* x for x in range(10))，列表生成式的[]更改为()

方式2: 如果一个函数定义中包含yield关键字，那么函数就不再是一个普通函数，而是一个generator，函数是顺序执行，遇到return语句或者最后一行函数语句返回。而generator在每次调用next()的时候执行，遇到yield语句返回，再次执行时从上次返回的yield语句处继续执行

1. **def** fib**(**max**):**
2. n**,** a**,** b **=** 0**,** 0**,** 1
3. **while** n **<** max**:**
4. **yield** b
5. a**,** b **=** b**,** a **+** b
6. n **=** n **+** 1
7. **return** 'done'
8. python代码规范

请参考[《python代码规范》](file:///C:\Users\Alta\Desktop\编程笔记\python代码规范.pdf)

1. 迭代器总结

* 凡是可作用于for循环的对象都是Iterable类型；
* 凡是可作用于next()函数的对象都是Iterator类型，它们表示一个惰性计算的序列；
* 集合数据类型如list、dict、str等是Iterable但不是Iterator，不过可以通过iter()函数获得一个Iterator对象。
* Python的for循环本质上就是通过不断调用next()函数实现的

1. Map 函数总结

map()函数接收两个参数，一个是函数，一个是Iterable，map将传入的函数依次作用到序列的每个元素，返回新的Iterator。

map(function, iterable1,iterable2 ...)

例如三个列表相乘：

1. list1 **=** **[**1**,**2**,**3**,**4**,**5**,**6**,**7**,**8**,**9**]**
2. list2 **=** **[**1**,**2**,**3**,**4**,**5**,**6**,**7**,**8**,**9**]**
3. list3 **=** **[**9**,**8**,**7**,**6**,**5**,**4**,**3**,**2**,**1**]**
4. **def** foo**(**l1**,**l2**,**l3**):**
5. **return** l1**\***l2**\***l3
6. **print(**list**(**map**(**foo**,**list1**,**list2**,**list3**)))**
7. **>>>[9,32,63,96,125,144,147,128,81]**
8. Reduce函数总结

reduce把一个函数作用在一个序列[x1, x2, x3, ...]上，这个函数必须接收两个参数，reduce把结果继续和序列的下一个元素做累积计算。

例如序列求和：

1. **>>>** **from** functools **import** reduce
2. **>>>** **def** add**(**x**,** y**):**
3. **...** **return** x **+** y
4. **...**
5. **>>>** reduce**(**add**,** **[**1**,** 3**,** 5**,** 7**,** 9**])**
6. 25
7. 匿名函数(lambda)总结

lambda parameters: expression

parameters：可选，如果提供，通常是逗号分隔的变量表达式形式，即位置参数。

expression：**不能包含分支或循环（但允许条件表达式）**，也不能包含return（或yield）函数。如果为元组，则应用圆括号将其包含起来。调用lambda函数，**返回的结果是对表达式计算产生的结果**

1. #根据参数是否为1 决定s为yes还是no
2. **>>>** s **=** **lambda** x**:**"yes" **if** x**==**1 **else** "no"
3. 全局变量的使用

使用到的全局变量只是作为引用，不在函数中修改它的值的话，不需要加global关键字

1. a **=** 1
2. **def** func**():**
3. **if** a **==** 1**:**
4. **print(**"a: %d" **%**a**)**

使用到的全局变量，需要在函数中修改的话，就涉及到歧义问题，因此，需要修改全局变量a，可以在"a = 2"之前加入global a声明

1. a **=** 1
2. **def** func**():**
3. **global** a
4. a **=** 2
5. **print(**"in func a:"**,** a)
6. python 定义类时，内部方法的互相调用

每次调用内部的方法时，方法前面加 self

1. class MyClass:
2. def \_\_init\_\_(self):
3. pass
4. def func1(self):
5. print('a')
6. self.common\_func()
7. def func2(self):
8. self.common\_func()
10. def common\_func(self):
11. pass
12. filter函数总结

filter()也接收一个函数和一个序列。filter()把传入的函数依次作用于每个元素，然后根据返回值是True还是False决定保留还是丢弃该元素

1. sorted函数总结

sorted()函数也是一个高阶函数，它还可以接收一个key函数来实现自定义的排序，key指定的函数将作用于list的每一个元素上，并根据key函数返回的结果进行排序

1. **>>>** sorted**([**36**,** 5**,** **-**12**,** 9**,** **-**21**],** key**=**abs**)**
2. **[**5**,** 9**,** **-**12**,** **-**21**,** 36**]**
3. 返回函数(闭包)

6. 装饰器

这种在代码运行期间动态增加功能的方式，称之为“装饰器”（Decorator）。本质上，decorator就是一个返回函数的高阶函数。

1. **def** log**(**func**):**
2. **def** wrapper**(\***args**,** **\*\***kw**):**
3. **print(**'call %s():' **%** func**.**\_\_name\_\_**)**
4. **return** func**(\***args**,** **\*\***kw**)**
5. **return** wrapper

如果将上述的log函数作为装饰器，则在其装饰的函数前添加@log

1. @log
2. **def** now**():**
3. **print(**'2015-3-25'**)**
4. >>> now()
5. call now():
6. 2015-3-25

装饰器原理及引入机制较为复杂，详情可参考[《python装饰器解释》](file:///C:\Users\Alta\Desktop\编程笔记\python装饰器解释.pdf)

1. 函数参数

* 必选参数(位置参数)：
* 默认参数：如def power(x, n=2)，x即为位置参数，n为默认参数，必选参数在前，默认参数在后
* 可变参数：传入的参数个数是可变的，不必自行将所有参数组装成一个list或tuple，如def calc(\*numbers)。定义可变参数和定义一个list或tuple参数相比，仅仅在参数前面加了一个\*号。在函数内部，参数numbers接收到的是一个tuple

1. **def** calc**(\***numbers**):**
2. sum **=** 0
3. **for** n **in** numbers**:**
4. sum **=** sum **+** n **\*** n
5. **return** sum

* 关键字参数:可变参数在函数调用时自动组装为一个tuple。而关键字参数允许你传入0个或任意个含参数名的参数，这些关键字参数在函数内部自动组装为一个dict:

1. **def** person**(**name**,** age**,** **\*\***kw**):**
2. **print(**'name:'**,** name**,** 'age:'**,** age**,** 'other:'**,** kw**)**
4. **>>>** person**(**'Michael'**,** 30**)**
5. name**:** Michael age**:** 30 other**:** **{}**
6. **>>>** person**(**'Bob'**,** 35**,** city**=**'Beijing'**)**
7. name**:** Bob age**:** 35 other**:** **{**'city'**:** 'Beijing'**}**
8. 字符串里面的引号

单引号' '定义字符串的时候，它就会认为你字符串里面的双引号" "是普通字符，从而不需要转义。反之当你用双引号定义字符串的时候，就会认为你字符串里面的单引号是普通字符无需转义

1. Str1 = "We all know that 'A' and 'B' are two capital letters."
2. Str2 = 'The teacher said: "Practice makes perfect" is a very famous proverb.'
3. 偏函数

functools.partial的作用就是，把一个函数的某些参数给固定住（也就是设置默认值），返回一个新的函数，调用这个新函数会更简单

1. **>>>** **import** functools
2. **>>>** int2 **=** functools**.**partial**(**int**,** base**=**2**)**
3. **>>>** int2**(**'1000000'**)**
4. 64
5. 算法速度的定义

* 大O：T(N) = O(f(N))，T增长率小于等于f
* **Ω**：T(N) =Ω (f(N))，T增长率大于f
* **Θ**：T(N) =Θ (f(N))，T增长率等于f
* 小o：T(N) =o (f(N))，T增长率小于f

1. 计算运行时间的一般法则

* 法则1：for循环：一个for循环的运行时间至多是该循环内语句的运行时间乘以迭代次数
* 法则2：嵌套for循环：嵌套循环内部的一条语句总运行时间为该语句运行时间乘以所有for循环的大小，如下程序片段运行时间为O(N2)

1. **for** i **in** range**(**N**):**
2. **for** j **in** range**(**N**):**
3. j **+=** 1

* 法则3：顺序语句：各个语句运行时间求和
* 法则4：if/else语句：运行时间至多是判断时间+S1或S2中运行时间长者

1. **if** **(**condition**):**
2. S1
3. **else:**
4. S2
5. 导入自定义模块
6. 子类继承父类的属性问题

更加细致的继承中的属性和方法问题，请参考《[python类的继承、属性总结和方法总结](file:///C:\Users\Alta\Desktop\编程笔记\python类的继承、属性总结和方法总结.pdf)》

如果子类自己定义了\_\_init\_\_方法，那么父类的属性是不能调用的，如下：

1. class Animal:
2. def \_\_init\_\_(self):
3. self.a = 'aaa'
4. class Cat(Animal):
5. def \_\_init\_\_(self):
6. pass
7. cat = Cat()
8. print(cat.a)
9. >>>AttributeError: 'Cat' object has no attribute 'a'

可以在子类的 \_\_init\_\_中调用一下父类的 \_\_init\_\_ 方法,这样就可以调用父类的属性

1. class Animal:
2. def \_\_init\_\_(self):
3. self.a = 'aaa'
4. class Cat(Animal):
5. def \_\_init\_\_(self):
6. super().\_\_init\_\_()
7. cat = Cat()
8. print(cat.a)
9. >>>aaa
10. 子类扩展父类属性
11. 附录
12. 方法解析顺序（Method Resolution Order, MRO）列表

MRO 列表的顺序遵循以下三条原则：

* 子类永远在父类前面
* 如果有多个父类，会根据它们在列表中的顺序被检查
* 如果对下一个类存在两个合法的选择，选择第一个父类

1. 查询模块的帮助文档

* 先导入模块，再查询普通模块的使用方法：help(module\_name)，例：help(math)
* 先导入sys，再查询系统内置模块的使用方法：sys.bultin\_modulenames
* 查看模块下所有函数：dir(module\_name)，例：dir(math)
* 查看模块下特定函数：help(module\_name.func\_name)，例：help(math.sin)
* 查看函数信息的另一种方法：print(func\_name.\_\_doc\_\_)，例：print(sin.\_\_doc\_\_)

1. python中时间日期格式化符号

* %y 两位数的年份表示（00-99）
* %Y 四位数的年份表示（000-9999）
* %m 月份（01-12）
* %d 月内中的一天（0-31）
* %H 24小时制小时数（0-23）
* %I 12小时制小时数（01-12）
* %M 分钟数（00=59）
* %S 秒（00-59）
* %a 本地简化星期名称
* %A 本地完整星期名称
* %b 本地简化的月份名称
* %B 本地完整的月份名称
* %c 本地相应的日期表示和时间表示
* %j 年内的一天（001-366）
* %p 本地A.M.或P.M.的等价符
* %U 一年中的星期数（00-53）星期天为星期的开始
* %w 星期（0-6），星期天为星期的开始
* %W 一年中的星期数（00-53）星期一为星期的开始
* %x 本地相应的日期表示
* %X 本地相应的时间表示
* %Z 当前时区的名称
* %% %号本身