**目录**

[1. 为什么操作系统只用C语言： 4](#_Toc532895074)

[2. 递归的调用 4](#_Toc532895075)

[3. 编写函数参数，或者调用对象的方法、属性的时候一定要注意逗号和句号 4](#_Toc532895076)

[4. 算法是通用/常用/频繁的方法，数据结构是通用/常用/频繁的数据组织形式 4](#_Toc532895077)

[5. super()函数/方法 4](#_Toc532895078)

[6. \_\_str\_\_与\_\_repr\_\_方法 4](#_Toc532895079)

[7. 查找模块下的方法(函数)、属性 5](#_Toc532895080)

[8. Import与from…import…的区别 5](#_Toc532895081)

[9. Tuple与()的使用区别 5](#_Toc532895082)

[10. Python时间格式化输出 5](#_Toc532895083)

[11. 格式化输出 6](#_Toc532895084)

[12. Windows环境下文件路径表示 6](#_Toc532895085)

[13. Python接收命令行参数 6](#_Toc532895086)

[14. '\u'前缀字符串 6](#_Toc532895087)

[15. 定义1个元素的tuple 6](#_Toc532895088)

[16. 创建生成器(generator)的两种方式 6](#_Toc532895089)

[17. python代码规范 7](#_Toc532895090)

[18. 迭代器总结 7](#_Toc532895091)

[19. Map 函数总结 7](#_Toc532895092)

[20. Reduce函数总结 7](#_Toc532895093)

[21. 匿名函数(lambda)总结 8](#_Toc532895094)

[22. 全局变量的使用 8](#_Toc532895095)

[23. python 定义类时，内部方法的互相调用 8](#_Toc532895096)

[24. filter函数总结 9](#_Toc532895097)

[25. sorted函数总结 9](#_Toc532895098)

[26. 返回函数(闭包) 9](#_Toc532895099)

[27. 9](#_Toc532895100)

[28. 9](#_Toc532895101)

[29. 装饰器 9](#_Toc532895102)

[30. 函数参数 10](#_Toc532895103)

[31. 字符串里面的引号 10](#_Toc532895104)

[32. 偏函数 11](#_Toc532895105)

[33. 算法速度的定义 11](#_Toc532895106)

[34. 计算运行时间的一般法则 11](#_Toc532895107)

[35. 导入自定义模块 11](#_Toc532895108)

[36. 子类继承父类的属性问题 12](#_Toc532895109)

[37. 子类扩展父类属性 12](#_Toc532895110)

[38. 附录 14](#_Toc532895111)

[1. 方法解析顺序（Method Resolution Order, MRO）列表 14](#_Toc532895112)

[2. 查询模块的帮助文档 14](#_Toc532895113)

[3. python中时间日期格式化符号 14](#_Toc532895114)

1. 为什么操作系统只用C语言：

第一个操作系统一定是用机器语言或者汇编语言直接编写的。之后就写编译器，有了编译器之后就能用高级语言写操作系统，之后编译为机器代码——能和硬件直接交互。

诞生的时间顺序：操作系统(汇编/机器语言写成)——编译器——高级语言——重写操作系统。理论上说，大部分语言都可以用来写操作系统内核。但是为什么大部分操作系统内核都只用C语言来写？因为目前所有的语言的开发环境里，C语言能做到编译成不依赖操作系统的形式二进制代码，C语言的各种脱离系统的库最丰富，最完整，C语言用来开发操作系统的工具最多。

1. 递归的调用

递归本身就是循环，所以在递归内部使用循环语句务必要非常小心。

递归函数的书写必须逻辑清楚，否则极容易出现混乱，因为递归的过程有些跟人类思考的逻辑顺序相违背。递归的条件必须非常地清晰，每一种情况和条件都穷举尽。

牢记递归调用的形式：1.基准情形，无需递归就能解出 2.不断推进，每一次递归调用都使得求解状况朝基准情形方向推进 3.设计法则，所有的递归调用都能运行

1. 编写函数参数，或者调用对象的方法、属性的时候一定要注意逗号和句号
2. 算法是通用/常用/频繁的方法，数据结构是通用/常用/频繁的数据组织形式

经典的数据结构形式非常固定，我们需要做的就是运用这些经典的数据结构，数据结构的困难在于证明或理解采用这些数据结构的运行时间(效率)

1. super()函数/方法

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》](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