Python

[Python 1](#_Toc256000000)

[1. 基础 5](#_Toc256000002)

[1.1. 注释 5](#_Toc256000003)

[1.2. 常量 5](#_Toc256000004)

[1.2.1. 数字 6](#_Toc256000005)

[1.2.1.1. 整数 6](#_Toc256000006)

[1.2.1.2. 浮点数 6](#_Toc256000007)

[1.2.2. 字符串 6](#_Toc256000008)

[1.2.2.1. 构建 7](#_Toc256000009)

[1.2.2.1.1. 单行 7](#_Toc256000010)

[1.2.2.1.1.1. 单引号 7](#_Toc256000011)

[1.2.2.1.1.2. 双引号 7](#_Toc256000012)

[完全相同(单引号, 双引号) 7](#_Toc256000013)

[1.2.2.1.2. 多行 7](#_Toc256000014)

[1.2.2.1.2.1. 三引号 8](#_Toc256000015)

[1.2.2.1.2.2. 换行符/n 8](#_Toc256000016)

[1.2.2.2. 联立字符串 8](#_Toc256000017)

[1.2.2.3. 格式化 8](#_Toc256000018)

[1.2.2.3.1. 数字是可选选项 8](#_Toc256000019)

[1.2.2.3.2. 更详细的格式 8](#_Toc256000020)

[1.2.2.4. print 8](#_Toc256000021)

[1.2.2.4.1. 空白结尾 9](#_Toc256000022)

[1.2.2.4.2. 空格结尾 9](#_Toc256000023)

[1.2.2.5. 转义序列 9](#_Toc256000024)

[1.2.2.5.1. \' 9](#_Toc256000025)

[1.2.2.5.2. \\ 9](#_Toc256000026)

[1.2.2.5.3. \n 9](#_Toc256000027)

[1.2.2.5.4. \ 9](#_Toc256000028)

[1.2.2.6. 原始字符串 10](#_Toc256000029)

[1.3. 变量 10](#_Toc256000030)

[1.4. 缩进 10](#_Toc256000031)

[1.5. 不用分号 10](#_Toc256000032)

[2. 运算符与表达式 10](#_Toc256000033)

[2.1. 字符串 10](#_Toc256000034)

[2.1.1. 相加 11](#_Toc256000035)

[2.1.2. 相乘 11](#_Toc256000036)

[2.2. 数值 11](#_Toc256000037)

[2.2.1. 算术 11](#_Toc256000038)

[2.2.2. 逻辑 12](#_Toc256000039)

[2.3. 赋值的快捷方式 12](#_Toc256000040)

[2.4. 运算顺序 12](#_Toc256000041)

[3. 控制流 12](#_Toc256000042)

[3.1. if/elif/else 13](#_Toc256000043)

[3.2. while 13](#_Toc256000044)

[3.3. for 13](#_Toc256000045)

[3.4. break 13](#_Toc256000046)

[3.5. continue 13](#_Toc256000047)

[3.6. pass 13](#_Toc256000048)

[4. 模块 14](#_Toc256000049)

[4.1. sys 14](#_Toc256000050)

[4.1.1. sys.argv 15](#_Toc256000051)

[4.1.2. sys.path 15](#_Toc256000052)

[4.2. 导入模块 15](#_Toc256000053)

[4.2.1. import 15](#_Toc256000054)

[4.2.2. from..import 15](#_Toc256000055)

[4.3. pyc文件 15](#_Toc256000056)

[4.4. 模块的\_\_name\_\_ 16](#_Toc256000057)

[4.5. dir函数 16](#_Toc256000058)

[5. 函数 16](#_Toc256000059)

[5.1. 参数 17](#_Toc256000060)

[5.2. 局部变量 17](#_Toc256000061)

[5.2.1. global语句 17](#_Toc256000062)

[5.3. 默认参数 17](#_Toc256000063)

[5.4. 关键字参数 17](#_Toc256000064)

[5.5. 可变参数 18](#_Toc256000065)

[5.6. return语句 18](#_Toc256000066)

[5.7. DocStrings 18](#_Toc256000067)

[5.7.1. 形式 18](#_Toc256000068)

[5.7.2. 查看 18](#_Toc256000069)

[6. 数据结构 19](#_Toc256000070)

[6.1. 列表 19](#_Toc256000071)

[6.1.1. 创建 20](#_Toc256000072)

[6.1.2. 移除 20](#_Toc256000073)

[6.1.3. 补充 20](#_Toc256000074)

[6.1.4. 排序 20](#_Toc256000075)

[6.2. 元组 20](#_Toc256000076)

[6.2.1. 创建 20](#_Toc256000077)

[6.2.2. 移除 21](#_Toc256000078)

[6.3. 字典 21](#_Toc256000079)

[6.3.1. 创建 21](#_Toc256000080)

[6.3.2. 添加 21](#_Toc256000081)

[6.3.3. 移除 21](#_Toc256000082)

[6.4. 集合 21](#_Toc256000083)

[6.4.1. 子集 22](#_Toc256000084)

[6.4.2. 交集 22](#_Toc256000085)

[6.5. 索引 22](#_Toc256000086)

[6.5.1. 从0开始计数 22](#_Toc256000087)

[6.5.2. 负数 22](#_Toc256000088)

[6.5.3. 指定序列 22](#_Toc256000089)

[6.6. 资格测试 22](#_Toc256000090)

[6.6.1. in或not in 23](#_Toc256000091)

[6.7. 引用 23](#_Toc256000092)

[6.7.1. 变量名赋予 23](#_Toc256000093)

[6.7.2. 切片操作 23](#_Toc256000094)

[7. 面向对象的编程 23](#_Toc256000095)

[7.1. 类 24](#_Toc256000096)

[7.1.1. 属性 24](#_Toc256000097)

[7.1.1.1. 类变量 24](#_Toc256000098)

[7.1.1.2. 方法 25](#_Toc256000099)

[7.1.2. 实例化方法 25](#_Toc256000100)

[7.1.2.1. 隐式调用 25](#_Toc256000101)

[7.1.3. 私有变量 25](#_Toc256000102)

[7.2. 对象 25](#_Toc256000103)

[7.2.1. 属性 25](#_Toc256000104)

[7.2.1.1. 对象变量 26](#_Toc256000105)

[7.2.1.2. 方法 26](#_Toc256000106)

[7.3. 继承 26](#_Toc256000107)

[7.3.1. 基类 26](#_Toc256000108)

[7.3.2. 派生类 26](#_Toc256000109)

[7.3.2.1. 调用基类的方法 27](#_Toc256000110)

[7.3.2.1.1. 实例化 27](#_Toc256000111)

[7.3.2.1.1.1. 显式调用 27](#_Toc256000112)

[7.3.2.1.2. 普通 28](#_Toc256000113)

[7.4. 文档字符串 28](#_Toc256000114)

[7.4.1. 类的 28](#_Toc256000115)

[7.4.2. 类的方法的 28](#_Toc256000116)

[8. 输入与输出 28](#_Toc256000117)

[8.1. 用户输入内容 28](#_Toc256000118)

[8.1.1. input 29](#_Toc256000119)

[8.1.2. print 29](#_Toc256000120)

[8.2. 文件 29](#_Toc256000121)

[8.2.1. write 29](#_Toc256000122)

[8.2.2. pickle 29](#_Toc256000123)

[8.3. unicode与utf-8 29](#_Toc256000124)

[9. 异常 29](#_Toc256000125)

[9.1. 分类 30](#_Toc256000126)

[9.1.1. 内置异常 30](#_Toc256000127)

[9.1.2. 自定义异常 30](#_Toc256000128)

[9.2. 处理异常 30](#_Toc256000129)

[9.2.1. try..except..(else) 30](#_Toc256000130)

[9.2.2. 抛出异常: try..raise..except.. 31](#_Toc256000131)

[9.2.3. try..except..finally 31](#_Toc256000132)

[9.2.4. with..as.. 31](#_Toc256000133)

[10. 更多 31](#_Toc256000134)

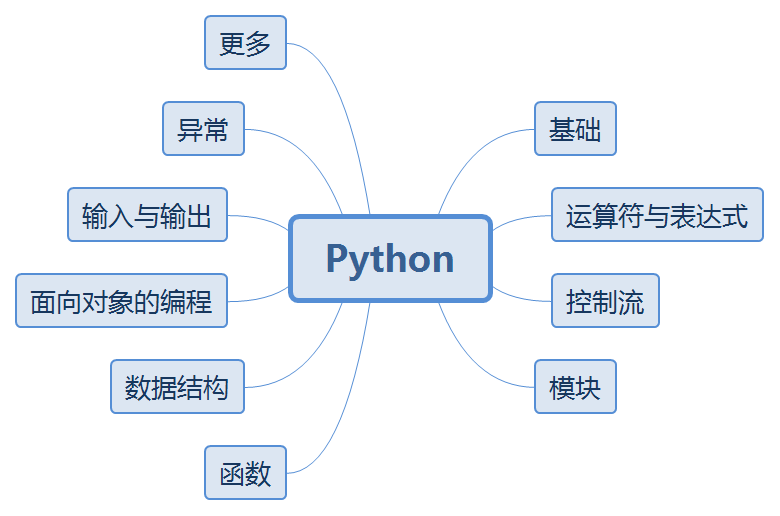
[10.1. 传递元组 32](#_Toc256000135)

[10.2. lambda语句 32](#_Toc256000136)

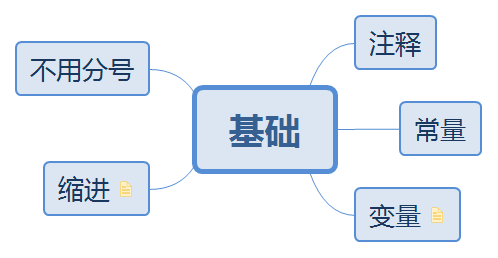
[10.3. 列表推导 32](#_Toc256000137)

[10.4. assert 语句 32](#_Toc256000138)

[10.5. 装饰器 32](#_Toc256000139)

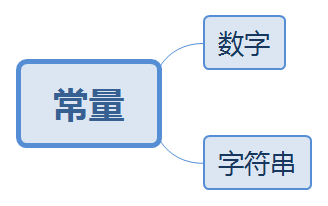


# 基础

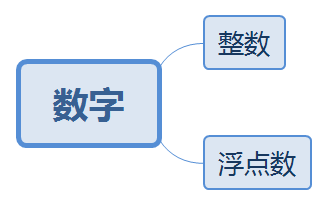


## 注释

## 常量



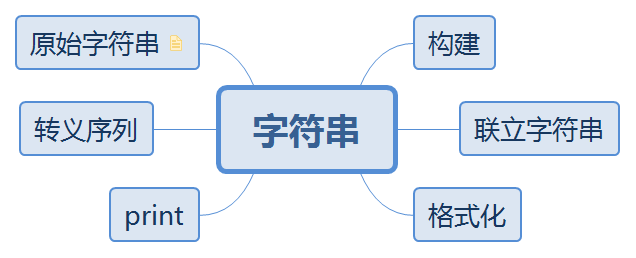
### 数字



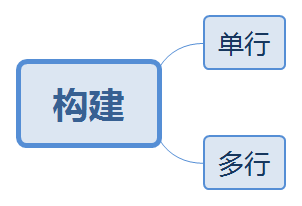
#### 整数

#### 浮点数

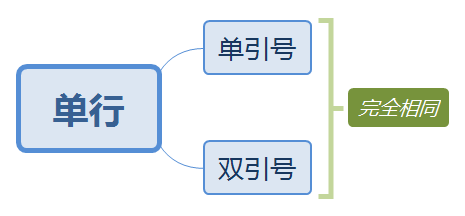
### 字符串



#### 构建



##### 单行

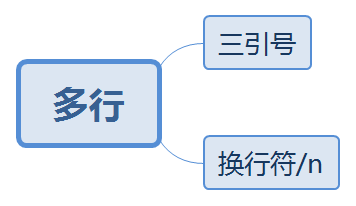


###### 单引号

###### 双引号

###### 完全相同([单引号](#_____3), [双引号](#_____4))

##### 多行



###### 三引号

###### 换行符/n

#### 联立字符串

#### 格式化

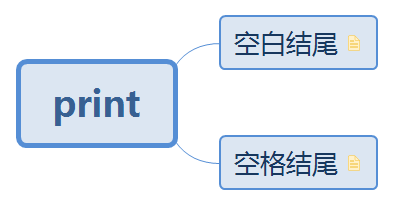


##### 数字是可选选项

##### 更详细的格式

如小数位数、字符串长度等

#### print



##### 空白结尾

end = ''

##### 空格结尾

end = ' '

#### 转义序列



##### \'

##### \\

##### \n

##### \

一个放置在末尾的反斜杠表示字符串将在下一行继续，但不会添加新的一行。适用于长字符串。

#### 原始字符串

r"xxx"

## 变量

可以用字母、下划线或数字，但不可以数字开头。

区分大小写。

## 缩进

一个block的语句必须拥有相同的缩进。

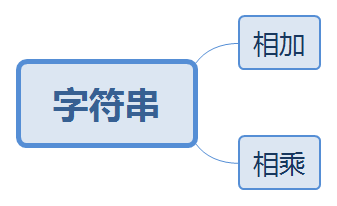
错误的缩进将会导致错误。

## 不用分号

# 运算符与表达式



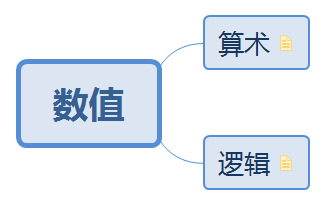
## 字符串



### 相加

### 相乘

## 数值



### 算术

加

减

乘

除

整除//

取余%

左移<<

右移>>

位与&

位或|

位异或^

位取反~:x 的按位取反结果为-(x+1)

### 逻辑

小于<

大于>

小于等于<=

大于等于>=

等于==

不等于!=

非not

与and

或or

## 赋值的快捷方式

a = 2

a = a \* 3

同样也可写作：

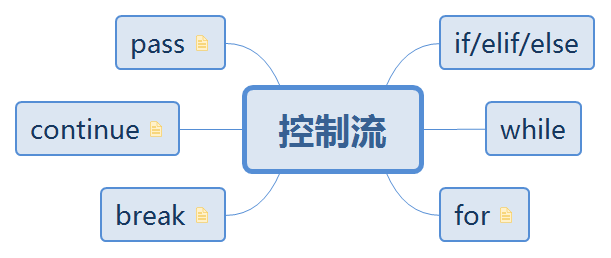
a = 2

a \*= 3

## 运算顺序

可以用括号改变运算顺序

# 控制流



## if/elif/else

## while

## for

for ... in 任何队列

range()每次生成一个数字，依次分配

range(1, 5)输出[1, 2, 3, 4]

range(1, 5, 2)输出[1, 3]

## break

终止循环语句的执行

## continue

continue不执行本次循环接下来的语句，直接进入下次循环

## pass

pass则执行本次循环接下来的语句

一个例子展示continue和pass的区别

print("using continue")

for i in a\_list:

if not i:

continue

print(i)

print("using pass")

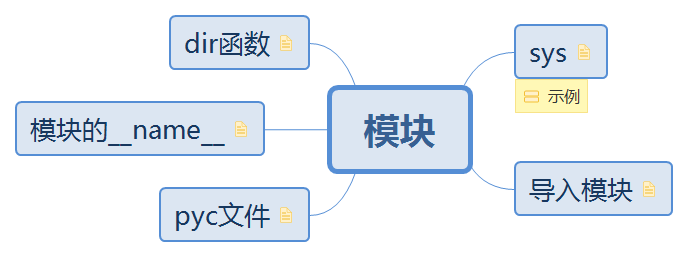
for i in a\_list:

if not i:

pass

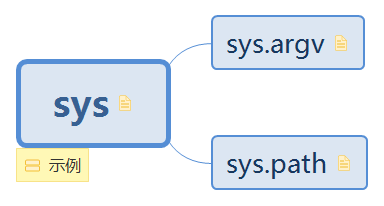
print(i)

# 模块



## sys

示例



模块包含了与Python解释器及其环境相关的功能，也就是所谓的系统功能

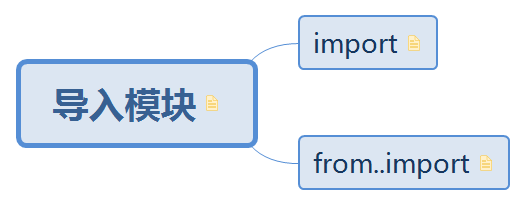
### sys.argv

sys.argv 包含了命令行参数（Command Line Arguments）这一列表

### sys.path

包含了导入模块的字典名称列表

## 导入模块



1. 导入内置模块

2. 导入自己编写的模块：从sys.path 变量所提供的目录中进行搜索。

### import

import sys

sys.argv[num]

sys.path

### from..import

避免使用

## pyc文件

 pyc文件只有在文件被当成模块导入时才会生成。也就是说，Python解释器认为，只有import进行的模块才需要被重用。 生成pyc文件的好处显而易见，当我们多次运行程序时，不需要重新对该模块进行重新的解释。主文件一般只需要加载一次不会被其他模块导入，所以一般主文件不会生成pyc文件。

## 模块的\_\_name\_\_

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

print('This program is being run by itself')

else:

print('I am being imported from another module')

每一个Python 模块都定义了它的\_\_name\_\_ 属性。如果它与\_\_main\_\_ 属性相同则代表这一

模块是由用户独立运行的，因此我们便可以采取适当的行动。

## dir函数

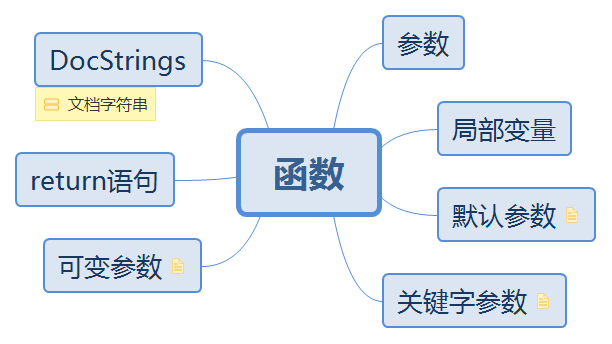
内置的dir() 函数能够返回由对象所定义的名称列表。如果这一对象是一个模块，则该列

表会包括函数内所定义的函数、类与变量。

该函数接受参数。如果参数是模块名称，函数将返回这一指定模块的名称列表。如果没有提

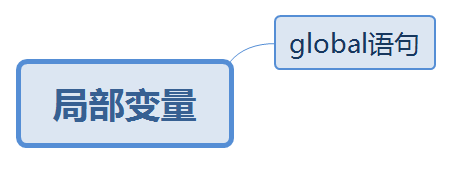
供参数，函数将返回当前模块的名称列表。

# 函数



## 参数

## 局部变量



### global语句

## 默认参数

在函数的参数列表中拥有默认参数值的参数不能位于没有默认参数值的参数之前

## 关键字参数

这样做有两大优点：

其一，我们不再需要考虑参数的顺序，函数的使用将更加容易。

其二，我们可以只对那些我们希望赋予的参数以赋值，只要其它的参数都具有默认参数值。

## 可变参数

当我们声明一个诸如\*param 的星号参数时，从此处开始直到结束的所有**位置参数**都将被收集并汇集成一个称为“param”的**元组**中。

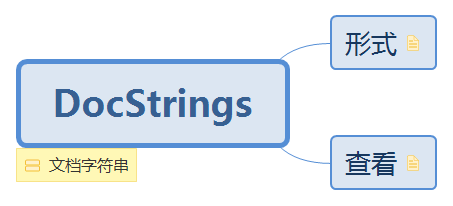
类似地，当我们声明一个诸如\*\*param 的双星号参数时，从此处开始直至结束的所有**关键字**

**参数**都将被收集并汇集成一个名为param 的**字典**中。

## return语句

## DocStrings

文档字符串



### 形式

"""

"""

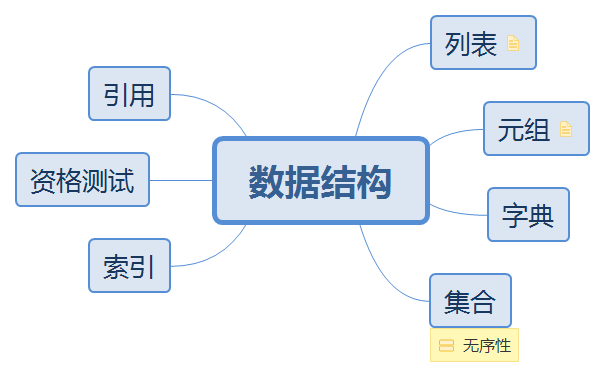
文档字符串所约定的是一串多行字符串，其中第一行以某一大写字母开始，以句号结束。第二行为空行，后跟的第三行开始是任何详细的解释说明。

### 查看

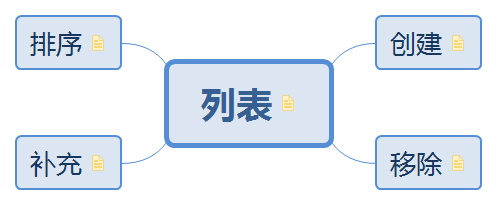
print(print\_max.\_\_doc\_\_)

help(print\_max)

# 数据结构



## 列表



可变

不同类型的元素

### 创建

方括号

逗号

### 移除

del

移除元素、移除列表

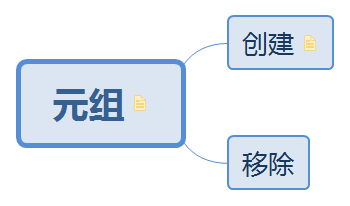
### 补充

list\_name.append('xxx')

### 排序

list\_name.sort()

## 元组



不可变

### 创建

括号

### 移除

## 字典



### 创建

花括号

### 添加

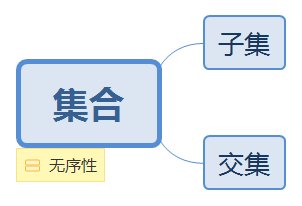
ab['Guido'] = guido@python.org'

### 移除

del ab['Spammer']

## 集合

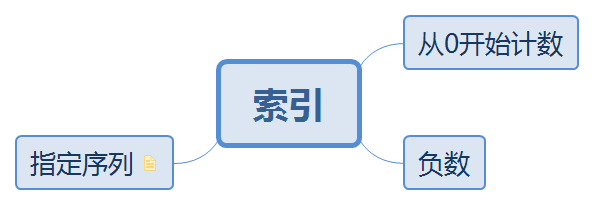
无序性



### 子集

### 交集

## 索引



### 从0开始计数

### 负数

### 指定序列

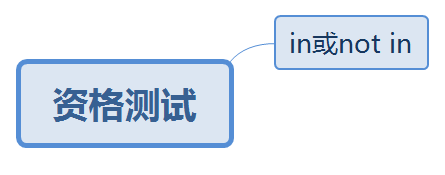
起始位置:结束位置(:步长)

包括起始位置,但不包括结束位置

可以是负数

数字是可有可无的

## 资格测试



### in或not in

## 引用



### 变量名赋予

变量名只是指向你计算机内存中存储了相应对象的那一部分.

如果仅仅将变量名赋予另一个名称,他们将绑定同一个对象

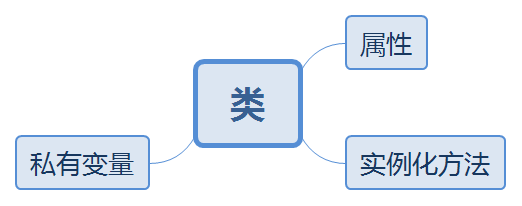
### 切片操作

切片操作可以生成副本

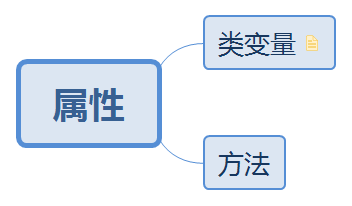
# 面向对象的编程



## 类



### 属性



#### 类变量

可以被属于该类的所有实例访问。

该类变量只拥有一个副本，当任何一个对象对类变量作出改变时，发生的变动将在其它所有

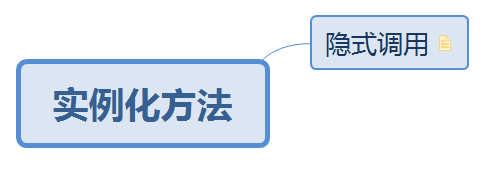
实例中都会得到体现。

CLASSName.CLASSVar

#### 方法

**参见:** [方法 (@classmethod)](#____43)

### 实例化方法



#### 隐式调用

**参见:** [显式调用](#______16)

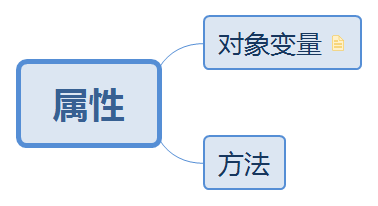
当我们在Person 类下创建新的实例p 时，我们采用的方法是先写下类的名称，后跟括在括号中的参数，形如：p = Person('Swaroop') 。我们不会显式地调用\_\_init\_\_ 方法。

### 私有变量

## 对象



### 属性



#### 对象变量

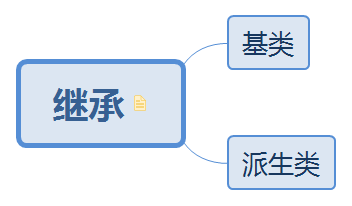
每一个独立的对象或实例所拥有。在这种情况下，每个对象都拥有属于它自己的字段的副本，也就是说，它们不会被共享，也不会以任何方式与其它不同实例中的相同名称的字段产生关联。

ObjcetName.ObjectVarName

#### 方法

**参见:** [方法 (@classmethod)](#____42)

## 继承



面向对象的编程的一大优点是对代码的重用。

### 基类

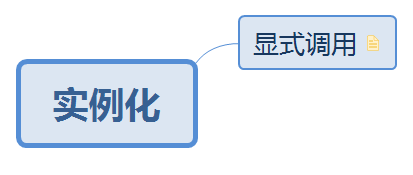
### 派生类



#### 调用基类的方法



##### 实例化



###### 显式调用

**参见:** [隐式调用](#______15)

如果在子类中定义了\_\_init\_\_ 方法，Python 不会自动调用基类的构造函数，你必须自己显式地

调用它。

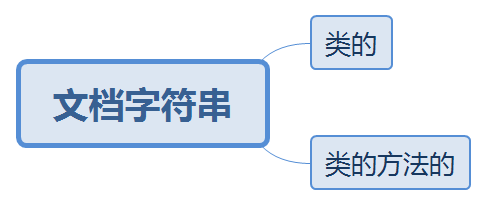
BASECLASS\_NAME.\_\_init\_\_(self, ..)

##### 普通

通过在方法名前面加上基类名作为前缀，再传入self和其余变量，来调用基类的方法。

当我们使用基类的方法时，我们可以将派生类的实例看作基类的实例。

## 文档字符串



### 类的

### 类的方法的

# 输入与输出



## 用户输入内容



### input

### print

## 文件

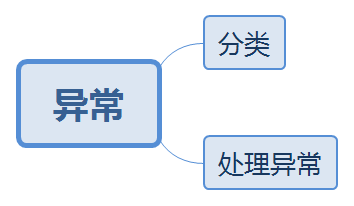


### write

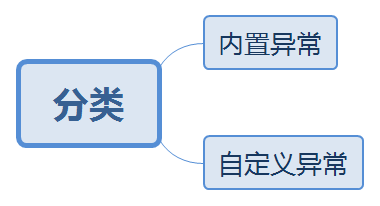
### pickle

## unicode与utf-8

# 异常



## 分类

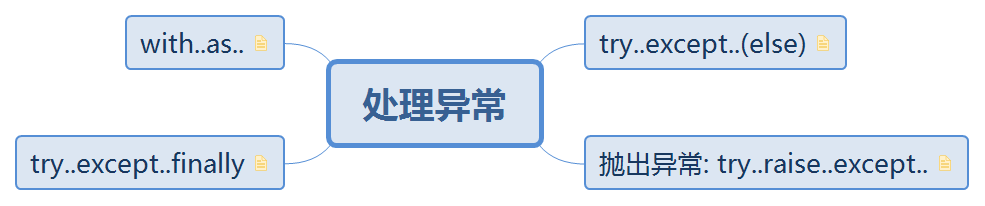


### 内置异常

### 自定义异常

**参见:** [抛出异常: try..raise..except.. (自定义异常只能主动抛出)](#_______try__raise__except__)

## 处理异常



### try..except..(else)

1. 一般来说我们会把通常的语句放在try 代码块中，将我们的错误处理器代码放置在except 代码块中。

2. except 子句可以处理某种特定的错误或异常，或者是一个在括号中列出的错误或异常。如果没有提供错误或异常的名称，它将处理所有错误与异常。

3. 如果没有任何错误或异常被处理，那么将调用Python 默认处理器，它只会终端程序执行并打

印出错误信息。

4. else 子句将在没有发生异常的时候执行。

### 抛出异常: try..raise..except..

**参见:** [自定义异常 (自定义异常只能主动抛出)](#_______9)

抛出异常相当于是说：“停止运行这个函数中的代码，将程序执行转到 except 语句 ”。如果没有 try 和 except 语句覆盖抛出异常的 raise 语句，该程序就会崩溃，并显示异常的出错信息。

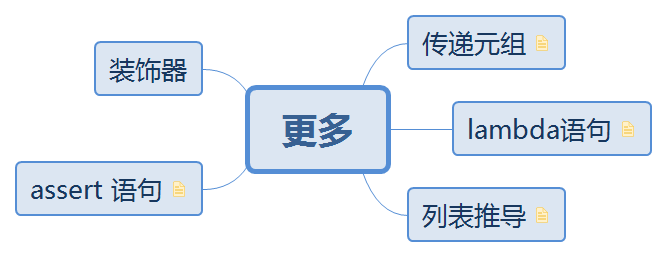
### try..except..finally

finally内的代码不管有无异常发生，都会执行。由于finally的这个特点，它一般用于释放资源，如数据库连接、文件流等。

### with..as..

with-as表达式极大的简化了每次写finally的工作.建议之后文件读写都用该写法

# 更多



## 传递元组

a, b = 1, 2

a, b = b, a

## lambda语句

lambda argument\_list: expression

## 列表推导

1. 代码

listone = [2, 3, 4]

listtwo = [2\*i for i in listone if i > 2]

print(listtwo)

2. 输出

[6, 8]

## assert 语句

语句断言失败时，将会抛出AssertionError

## 装饰器