# 1 Python

## 标准数据类型

图1



### 1.1.1 列表（序列）

列表：列表是有序的元素的集合。

图1



#### 切片

切片的结果包含切片的第一个元素，不包含第二个元素。

图1



#### None

None表示什么都没有。

图1



#### in（成员资格）

in判断一个元素是否在列表中。

图1



### 1.1.2 元组

元组：

图2



（注意：列表是可以修改的，元组是不能修改的）

### 1.1.3 字典

字典：字段类似于Java的Map，是key/value的结构。

图3

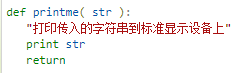


## 函数

图1



图2



### 匿名函数

python中的匿名函数用lamba来创建。

图1



图2



### 无返回值的函数

函数如果无返回值，则返回值是None。

图1



图2



### 返回值是一个函数名的函数

当一个函数的返回值是一个函数名时，可以用一个变量接收函数名，然后用()调用。

图1

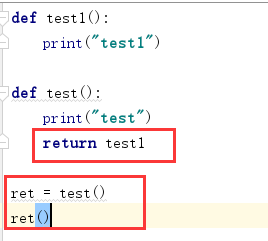
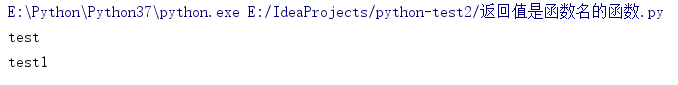


图2



### 匿名函数

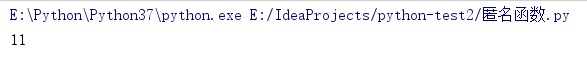
使用lambda 形参:函数体 的格式来定义匿名函数，匿名函数没有函数名。

（注意：lambda函数的函数体自动return）

图1（注意：a是变量，不是函数名）



图2



### map内置函数

map，对列表中的每一个元素，经过func计算后，计算出对应的另一个元素。

（注意：一对一）

（注意：map的结果是一个迭代器，该迭代器只能迭代一次，第二次迭代时迭代器会清空）

图1

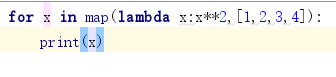
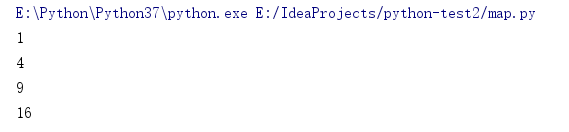


图2



### filter内置函数

和map类似，对列表中的每一个元素，经过func过滤后（只要那些func返回true的元素），保留剩余的元素。

（注意：filter的结果也是一个迭代器）

图1

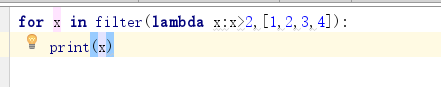
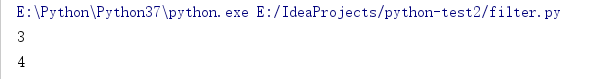


图2



### reduce内置函数

reduce，对列表中的两两元素执行func计算，两两的结果再和下一个组成两两（其实就是合并过程），最终得出一个值。

（注意：reduce的结果是一个值，不是一个迭代器）

图1

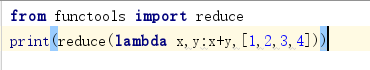
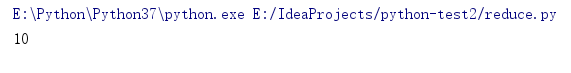


图2



### eval内置函数

eval将字符串解析出来。

图1

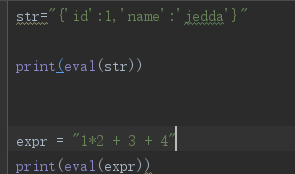
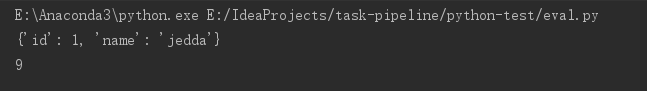


图2



### hash内置函数

可hash的则是不可变数据类型，不可hash的则是可变数据类型。

（注意：只要变量的值不变，hash值就不变）

图1

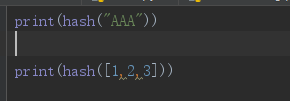
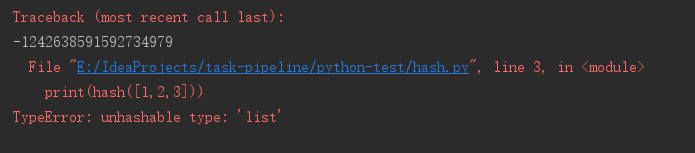


图2



### zip内置函数

zip是压缩/.拉链，参数是多个可迭代对象，将多个可迭代对象中对应位置的元素组成元组，最后返回元组组成的zip对象。

可以用list将zip对象转换成列表。

图1

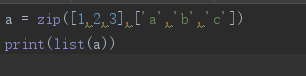
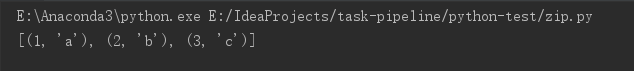


图2



## 可变和不可变对象

在python中，strings，tuples和numbers是不可变的对象，list和dict是可变的对象。

图1



### 可变对象和不可变对象的参数传递

不管是可变对象和不可变对象，传递的都是对象的地址。但当尝试去修改一个不可变的对象时，因为该对象不可变，所以只能创建一个新的对象，然后让引用指向这个新对象。

图1



图2



## 全局变量和局部变量

函数内是局部变量，函数外是全局变量。

（注意：全局变量是没有任何缩进的，局部变量是有缩进的）

图1



## 模块（Module）

在python中，一个模块就是一个Python文件（\*.py文件）。

图1



### 引入模块

模块定义好后，可以用import引入模块。

图1



图2



### 引入模块的一部分函数

模块定义好后，可以用from...import...引入模块的一部分函数。

图1（从模块math中导入sqrt函数）



### 使用模块中的函数

使用import导入模块后，使用module.function的方式调用模块的函数。

图1



## #! /usr/bin/env python

如果不加#! /usr/bin/env python，则必须指定python解释器来执行\*.py脚本，比如c:\python hello.py。

但UNIX支持在py脚本的第一行用#! /usr/bin/env python指定python的绝对路径后，不管python库在什么地方，都不需要再指定python解释器，直接hello.py运行py脚本。

当同一台就机器安装了多个python版本时，使用python3/python2代替python。

## 1.7多态

python中的多态，让我们不需要知道对象所属什么类，因为不管对象是什么类的实例，它只要该类覆盖了该函数，就可以利用多态来调用。

图1

（不需要x到底是字符串还是列表，因为多态的存在，只要字符串和列表都覆盖了count函数即可）





图2

（因为多态的关系，所以 + 既可以用于整数，也可以用于字符串，因为整数和字符串类型都支持加法）

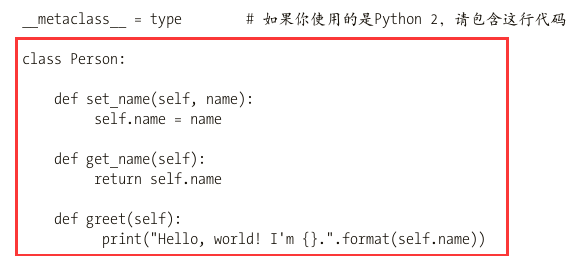


## 1.8类

### 1.8.1 自定义类

self类似于Java的this，表示对象实例本身。

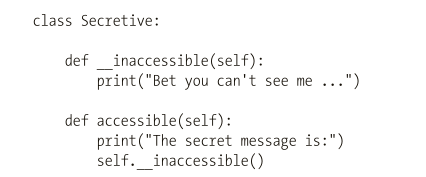
图1



### 1.8.2 私有属性和私有方法

使用2个下划线加在属性/方法前面，表示这个属性/方法是私有的。

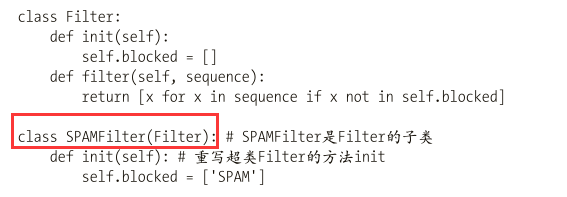
图1



### 1.8.3 指定超类

在类名后面用()指定超类。

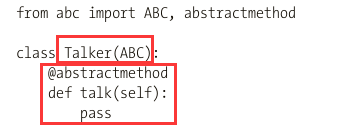
图1



### 1.8.4 抽象基类

抽象类不能被实例化。

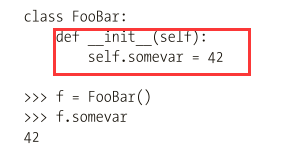
图1



### 1.8.5 构造函数

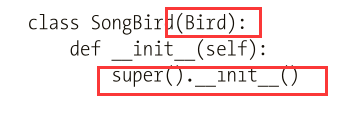
构造函数命名为\_\_int\_\_，构造函数不同于普通函数在于：对象创建后会自动调用构造函数。

图1



重写构造函数时，如果要继承超类的属性和方法，必须在构造函数中调用超类的构造函数。

图2

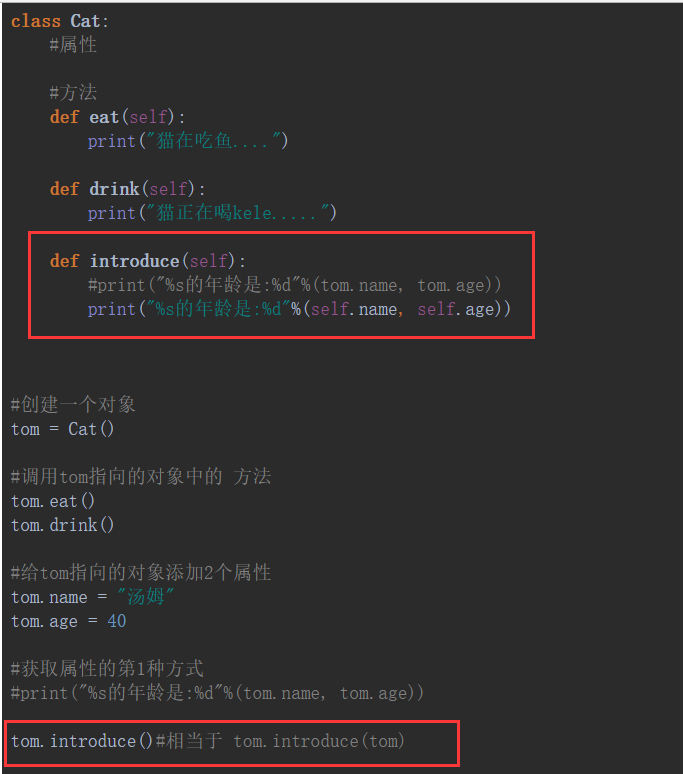


### 1.8.6 self

self类似于Java的this，表示对象实例本身。

在调用对象的方法时，python解释器会自动将对象实例作为方法的第一个参数传入，所以调用者只需要传入其他参数即可。

图1



### 1.8.7 property

用property函数创建了一个特性，然后将名称size关联到这个特性上，这个名称size使用时就像该类的属性一样，但size的计算依赖于property函数中的get和set方法。

图1

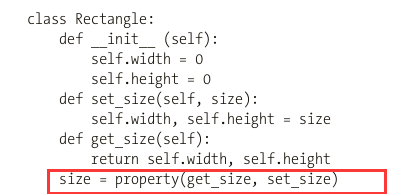
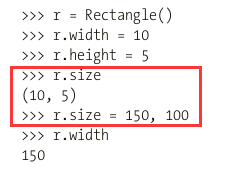


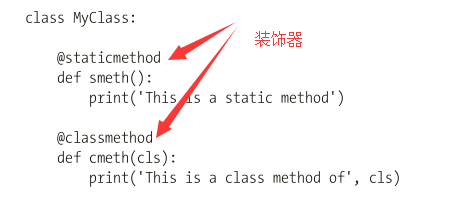
图2



### 1.8.8 静态方法和类方法

静态方法和类方法都可以通过类直接调用，但类方法有参数cls表示该类。

图1



### 1.8.9\_\_getsttr\_\_和\_\_setattr\_\_

\_\_getsttr\_\_和\_\_setattr\_\_用于属性被访问或者赋值时被调用。

图1

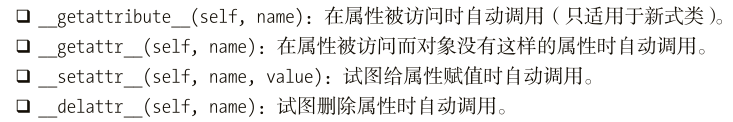
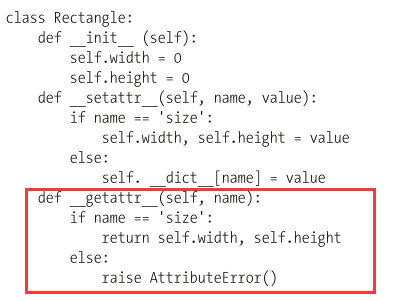


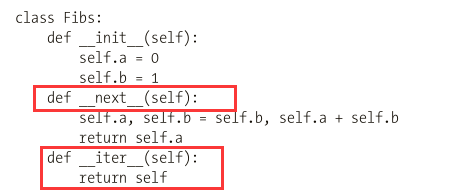
图2



### 1.8.10 \_\_iter\_\_

\_\_iter\_\_魔法方法返回一个迭代器，\_\_next\_\_获取迭代器中的下一个元素。

图1



## 1.9 异常

在python中，几乎所有的异常都是Exception类的之类。

### 1.9.1 自定义异常

自定义异常时，只需要指定超类是Exception类。

图1



### 1.9.2 捕捉异常进行处理

使用try/except捕捉异常。

图1



# 2 Python 爬虫

原文链接：<https://www.cnblogs.com/sss4/p/7809821.html>

如果把互联网比喻成一张蜘蛛网，爬虫就是一只小蜘蛛。

爬虫就是通过程序来模拟浏览器，去请求站点，将站点返回的HTML代码/JSON数据/二进制数据（图片、视频）爬到本地，然后提取自己需要的数据来供自己使用。

图1

