Abstract

——>Means 告诉计算机做什么而不是怎么去做

callable() 函数返回 true/false 是否能调用

文档字符串——docstring

所有的函数都返回值，如果你没告诉他们返回什么，就会返回None

参数

在函数内部重新关联参数（即给它赋值）时，函数外部的变量不受影响。

参数出存在**局部作用域**中

对序列执行切片操作时，返回的切片都是副本

dict中，key的排列顺序是不定的，每次打印都会有可能不同

类型：

位置参数、关键字参数、加*意思是不限数量参数

收集参数：

将参数收集到tuple或dict中

带星号的参数会返回一个tuple。

例如 def func(*a)：

print（a）

func（'abcd'）

>> ('abcd',)

func (1,2,3)

>> (1,2,3)

带星号的参数可以放其他位置，例如 def func(a, *b, c) 调用时赋予c需要用关键字

带星号的参数不会收集关键字参数，如果收集的话，要用**两个星号**

带两个星号的是返回一个dict

例如 def func(**a):

print(a)

func(a=1,b=2,c=3)

>> ('a'=1,'b'=2, 'c'=3)

分配参数：

将 为dict或tuple的参数 分配到函数中去

例如

```
a=(1,2)
def pr(x, y):
    print(x, y)
pr(*a)
>> 1 2
```

作用域：

执行赋值语句时，例如a=1，实际上是创建了一个"看不见的dict"，将x=1放进去。可以用一个叫vars的内建函数看这个dict。

例如：

```
x=1
b=vars()
print(b['x'])
>> 1
```

这个看不见的dict，就叫 **命名空间（Namespace）** 或 **作用域（scope）**
除了全局作用域，每个函数调用都会创建一个，在函数里用的变量叫**局部变量，**是在内部作用域中执行的，不会影响到**外部作用域（全局变量）。（慎用全局变量！容易出bug！）**
函数**globals**返回全局变量的dict，**locals**返回局部变量的dict，如果在函数内一个变量与全局变量同名，在函数内全局的会被遮盖，要访问就要用 例如 globals()[‘parameter_name’] 或者在函数里定义变量时, global parameter_name ，然后使用时会调用全局变量。
python函数可以嵌套，作用域也会被嵌套
递归一般比循环慢，但是可读性高。
bisect model 提供了标准的二分查找实现

Object-Orientation

使用Object的好处:
多态（polymorphism)：可对不同的类型的object执行相同的操作
封装（encapsulation)：对外隐藏有关object工作原理的细节
继承（inheritance)：可基于通用类创造出专用类

多态：

无需知道object的类型，都可对其执行操作
例如： 1+2 >> 3
‘a’+’b’ >>’ab’

封装：

和多态有点像
调用方法时，不会影响到全局变量，因此，将其封装在对象中，称之为“**属性**”

继承：

内置方法 issubclass(a,b)返回a是否为b的子类
一个类的特殊属性__bases__返回它的基类
isinstance(a,A)返回a是否为A类的实例
object的特殊属性__class__返回它属于的类
hasattr(a,’b’) 返回a类中有没有属性b
setattr(a,’b’,’c’)把a类中的b属性设置为c
getattr(a,’b’,’c’)获取a类中的b属性，如果不存在，则返回c
尽量避免使用多重继承

类（Class):

定义：一种对象。每个对象都属于特定的类，并被称为该类的实例。
例如：一只鸟，是鸟类的实例。这只鸟可能属于“云雀”，那“云雀”就是“鸟类”的**子类，**“鸟类”为“云雀”的**超类。**
子类有超类的所有方法，因此定义子类只需要定义多出来的方法或是重写原来的方法。

python没有对private提供直接的支持，要让属性或方法成为私有的，要让其名称**以两个下划线开头。**__

在类定义中，对所有以两个下划线开头的名称都转换成，在开头加上一个下划线和类名。（可以用这种方法访问private方法）即，python中无法禁止别人访问private属性或方法
from module_name import * 不会导入以_开头的方法。

类的命名空间：

在class语句中定义的代码都是在一个特殊的命名空间（类的命名空间）内执行的。类的所有成员都可以访问这个空间。

接口（Interface):

定义：对外暴露的方法和属性

抽象基类：

abc module为抽象基类提供了支持。
一般而言，抽象类是不能实例化的类，其职责是定义子类应实现的一组抽象方法
例如:

```
from abc import ABC, abstractmethod

class Talker(ABC):
        @abstractmethod
        def talk(self):
                pass
```

oo design的一些思考：

将相关的东西放在一起。如果一个函数操作一个全局变量，最好将他们作为一个类的属性和方法。
不要让对象之间过于紧密
慎用继承，尤其是多重继承
保持简单

每日小技巧：

1.

```
a=[1,2]
print(a) >> [1,2]
print(*a) >> 1 2
```

2.
random module 提供一个叫choice的function，可以从**序列**中随机选择一个元素。
例如：

```
from random import choice
a=choice( [1,2] )
```