# Python新手笔记

### 面向对象基础

#### 抽象

随着计算机的发展，编程语言也是在不断地抽象，抽象程度越高，编程语言就越高级，同样也就越来越远离机器，而贴近人类。

#### 面向对象思想

面向对象思想，就是用程序来描述现实世界。在Python中，一切皆是对象，这是一门纯粹的面向对象的编程语言。对象一般可以用名词来形容，因此现实中的事物，都可以用对象来表示。而类，则是对象的模板，一般都是相对抽象的名词，比如，“电脑”这个词汇，并没告诉你是哪个电脑，但是所有的电脑

示例如下：

### Python语法基础

#### 2.1 Python基础语句

##### 2.1.1 基本数据类型

4种基本数据类型：

注：python 为动态类型语言，因此不需要事先声明变量类型。

***在python中，万物皆对象，因此，python中的变量都是对象引用实现的。***

|  |  |
| --- | --- |
| 数据类型 | 示例 |
| **整数** | var = 40 |
| **浮点数** | var = 3.14 |
| **布尔类型** | var = False ; var = True |
| **复数** | 4+5j ; complex(4,5) |

代码示例：

# 基本数据类型

#示例1 路程 = 速度 \* 时间

speed = 12.8 #以m/s为单位，浮点数

time = 30 #以秒为单位，整数

#路程s

s = speed\*time

print(s)

#根据Python中一切皆是对象

intA = int(10) #整数

floatB = float(3.14) #浮点数

complexC = complex(3,5) #复数

print(intA)

print(floatB)

print(complexC)

boolA= bool(True) #布尔类型，值为True

boolB = bool(False) #布尔类型，值为False

print(boolA)

print(boolB)

##### 2.1.2 操作符

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 算术操作符 | 说明 | 逻辑操作符 | 说明 | 比较操作符 | 说明 |
| **+** | 加法 | **and** | 与 | **<** | 小于 |
| **-** | 减法 | **or** | 或 | **<=** | 小于等于 |
| **\*** | 乘法 | **not** | 非 | **>** | 大于 |
| **/** | 除法 |  |  | **>=** | 大于等于 |
| **//** | 浮点除法 |  |  | **==** | 等于 |
| **%** | 取模 |  |  | **!=** | 不等于 |
| **\*\*** | 取幂 |  |  |  |  |

#### 2.2 Python 控制结构

##### 2.2.1 条件结构

'''

**python 条件语句三种**

**① if语句**

**② if——else语句**

**③ if——elif——else语句**

'''

admin = "admin"

pswd = "12345"

if admin == "admin" and pswd =="12345":

print("登陆成功!")

if admin == "admn" and pswd =="12345":

print("登陆成功!")

else:

print("登陆失败！")

usercmd = "delete"

if usercmd == "create":

print("创建项目")

elif usercmd == "delete":

print("删除项目")

elif usercmd == "update":

print("更新项目")

else:

print("#####")

#在python中，不存在三元运算符

#但是可以这样得到相同的效果

x = 10

y = 45

smaller = x if x<y else y

print(smaller)

##### 2.2.2 循环结构

''''

**python循环结构**

**① while循环**

**while expression:**

**code ...**

**② for循环**

**for iter in iterable:**

**code ...**

'''

#while循环

count = 0

while count < 10 :

print(count,end=',')

#python中没有自增/自减运算符

count+=1

#换行

print()

#无限循环

count = 0

while True:

if(count<10):

print(count,end=",")

else:

break;

count+=1

print()

#for语句——用于序列类型

#通过序列项迭代

numList = [0,1,2,3,4,5,6,7,8,9]

for var in numList:

print(var,end=",")

print()

#通过索引index迭代

for index in range(len(numList)):

print(numList[index],end=";")

print()

#使用项和索引进行迭代

alphList=['a','b','c','d','e']

for index,item in enumerate(alphList):

print(index," ",item)

**结果：**

0,1,2,3,4,5,6,7,8,9,

0,1,2,3,4,5,6,7,8,9,

0,1,2,3,4,5,6,7,8,9,

0;1;2;3;4;5;6;7;8;9;

0 a

1 b

2 c

3 d

4 e

##### 2.2.3 迭代器与生成器

#### Python 数据结构

##### 2.3.1 列表

列表，元组，字符串属于序列，序列类型都有**关系操作符（in），大小计算函数（len()），分片（[]），以及迭代操作**，序列基本包括：***创建，索引，切片，加，乘，检查成员***6种操作。

|  |  |  |
| --- | --- | --- |
| python表达式 | 结果 | 描述 |
| **mylist = [0,1,2,3,4]**  **mylist1 = list(“abcd”)**  **mylist2 = list( i for i in range(6))** |  | 创建列表 |
| **print(mylist[0])**  **print(mylist1[2])**  **print(mylist2[3])** | **0**  **c**  **3** | 列表索引操作 |
| **print(1 in mylist)**  **print('e' not in mylist1)**  **print(3 not in mylist2)** | **True**  **True**  **False** | 列表检查成员操作 |
| **mylist+mylist1** | **[0,1,2,3,4,’a’,’b’,’c’,’d’]** | 列表加操作/连接操作 |
| **mylist\*2** | **[0,1,2,3,4,0,1,2,3,4]** | 列表乘操作/重复操作 |
| **print(mylist)**  **print(mylist[:])**  **print(mylist[0:3])**  **print(mylist[:3])**  **print(mylist[2:5])**  **print(mylist[2:])**  **print(mylist[1:3])**  **print(mylist[2])**  **#步长控制切片**  **print(mylist[::-1])**  **print(mylist[::2])** | **[0,1,2,3,4]**  **[0,1,2,3,4]**  **[0,1,2]**  **[0,1,2]**  **[2,3,4]**  **[2,3,4]**  **[1,2]**  **[2]**  **[4,3,2,1,0]**  **[0,2,4]** | 列表切片操作 |
| **del mylist1[2]** | **[‘a’,’b’,’d’]** | 列表删除元素 |
| **mylist1[1] = 'f'** | **[‘a’,’f’,’d’]** | 列表更新元素 |

##### 2.3.2 元组

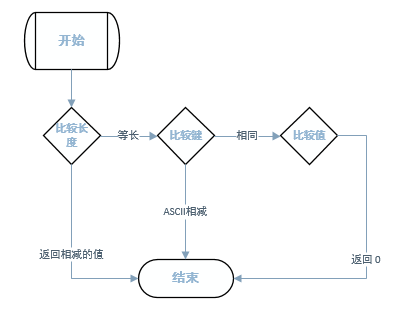
元组操作和列表操作相同，但是元组是**不允许修改元组值**的。也就是说除了更新操作元组没有之外，其他操作和列表操作相同。

##### 2.3.3 字典

字典 是python语言中唯一的映射类型。形式如：{key:value,key:value}

|  |  |  |
| --- | --- | --- |
| 代码示例 | 结果 | 说明 |
| adict = dict(([1,'a'],[2,'b']))  bdick = {1: 'a', 2: 'b'}  ddict = {}.fromkeys(('x','y'),-1) | {1: 'a', 2: 'b'}  {1: 'a', 2: 'b'}  {'x': -1, 'y': -1} | 创建字典 |
| for k in adict.keys():  print(k,adict[k])  for k in adict:  print(k,adict[k]) | * 1. a   2. b   1 a  2 b | 访问字典中的值 |
| addict[1] = ‘c’  addict[3]=’e’ | {1: 'c', 2: 'b'}  {1: 'c', 2: 'b',3:’e’} | 更新操作,如果没有该键，则进行增加操作 |
| del adct[1]  adict.clear()  del addict | { 2: 'b',3:’e’}  {}  无 | 删除操作 |
| 1 in adict  1 not in adict | True  False | 判断键是否存在 |

字典的比较算法 cmp(dict1,dict2):



注意：字典中的键只允许对应一个值

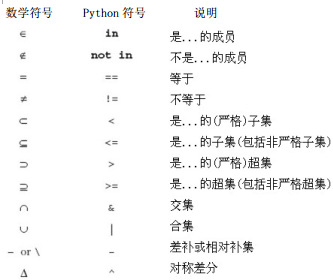
字典中的键必须是可哈希的，即键是不变的，因此列表和字典是不能作为键的。

##### 2.3.4 集合

集合set，和数学中的集合概念是相对应的，因此在集合中，重复的只会显示一个。并且是无序排列。具体操作如下：

|  |  |  |
| --- | --- | --- |
| 代码 | 结果 | 说明 |
| aset = set(“hello world)  cset = frozenset("hello world") | {' ', 'h', 'e', 'd', 'r', 'o', 'w', 'l'}  frozenset({' ', 'h', 'e', 'd', 'r', 'o', 'w', 'l'}) | 创建集合，去除重复的l  Frozenset创建的集合为不可变集合 |
| 'h' in aset  'c' not in aset | True  True | 检查成员 |
| aset.add()  aset.update() |  | 添加操作 |
| aset.remove()  aset -=set() |  | 删除操作 |

集合操作符与关系符号对应：



##### 2.3.5 字符串

字符串的操作，基本和列表的操作相同，不同点如下：

1） 创建字符串是用**单双引号**来表示，比如：aStr = ‘hello’ bStr = “hello ”

2）字符串的删除除了使用**del语句**之外，还可以将**字符串赋值为空串**。

3）字符串的比较是按照**ASCII值的大小**来比较的。

4）**格式化字符串/字符串模板/原始字符串**（r”hello”）等字符串特有操作。

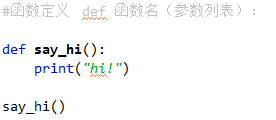
**注意**： **字符串可以使用三引号来创建跨行以及包含特殊字符的字符串。**

**字符串是一种不可变的数据类型，每次的改变都是分配新串。**

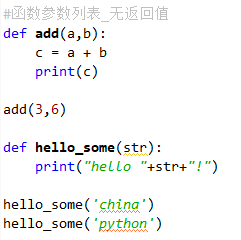
### 第三章 Python进阶

#### 3.1 Python 方法

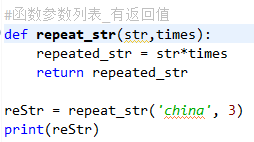
##### 3.1.1 方法定义



##### 3.1.2 方法示例\_无返回值



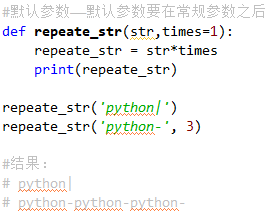
##### 3.1.3 方法示例\_有返回值



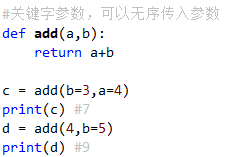
##### 3.1.4 全局变量与局部变量



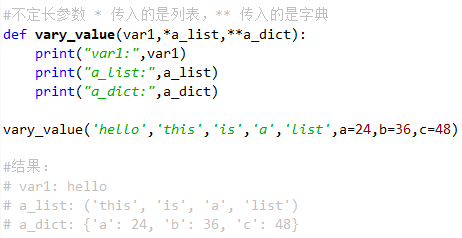
##### 3.1.5 函数参数列表—默认参数



##### 3.1.6 函数参数列表—关键字参数



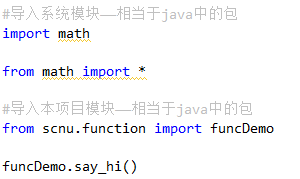
##### 3.1.7 函数参数列表—不定长参数



#### 3.2 Python 模块

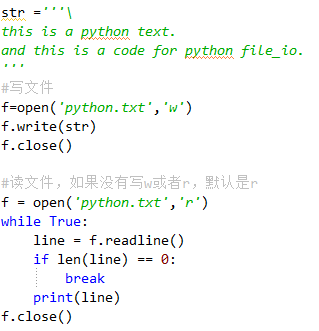
##### 3.2.1 导入模块

**Python中的模块概念相当于java中的包的概念：**



#### 3.3 Python 文件

##### 3.3.1 读文件与写文件



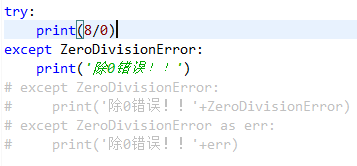
#### 3.4 Python 异常

##### 3.4.1 python异常介绍

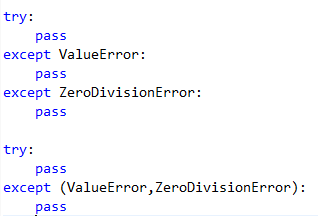
1. python语法错误——不能执行，需要调试，比如while True print(‘hello’) 没有‘：’也没有缩进。

2. python异常——是逻辑上的错误，比如print(8/0)

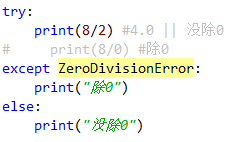
##### 3.4.2 处理单个异常



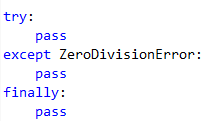
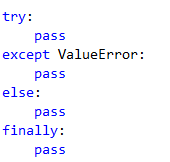
##### 3.4.3 处理多个异常



##### 3.4.4 异常else子句



##### 3.4.5 异常finally子句

##### 3.4.6 raise语句

##### 3.4.7 断言assert语句

### 第四章 Python 高级

#### 4.1 Python 高级面向对象

##### 4.1.1 面向对象技术简介

Python中一切都是对象，并且在Python3中统一了类型与类的概念：类型就是类。

类，类变量，数据成员，方法重写，实例变量，继承，实例化，方法，对象

##### 4.1.2 类的定义与使用方法

#类的定义，必须继承一个父类

class ClassName(object):

"""docstring for ClassName"""

def \_\_init\_\_(self, arg):

super(ClassName, self).\_\_init\_\_()

self.arg = arg

#示例：

class Person(object):

"""docstring for Person"""

name = "mike" #这是一个类属性

\_\_age = 23 #这是一个私有属性

def \_\_init\_\_(self, name):

super(Person, self).\_\_init\_\_()

self.name = name

#外部访问私有属性

def get\_age(self):

return self.\_\_age

#这是一个私有方法

def \_\_say():

print("say hello ")

#类的实例化

p = Person("jack")

#实例变量会屏蔽掉类变量，但是类变量并没改变

p.name = "jack"

p.addr = "beijing" #这是一个实例属性

#面向对象中的静态方法，类方法以及属性方法

class Employee(object):

"""docstring for Employee"""

name = "mike"

#构造方法

def \_\_init\_\_(self, name):

super(Employee, self).\_\_init\_\_()#调用父类构造方法

self.name = name

#静态方法，使方法脱离类，只是名义上属于该类

#静态方法，不需要self也不需要cls,因此只能类名访问。

@staticmethod

def staticSay():

print("static")

pass

#类方法，参数只能获取类变量，不能获取实例变量

@classmethod

def clsSay(cls):

print("classmethod")

pass

#属性方法，把一个方法变为静态属性

@property

def doing(self):

print("property")

@doing.setter

def doing(self,arg):

print("arg")

@doing.deleter

def doing(self):

print("deleter")

#析构方法

def \_\_del\_\_(self):

print("析构方法")

#静态方法调用

employee = Employee("mike")

employee.staticSay()

#类方法调用

employee.clsSay()

#属性方法有参调用

employee.doing = "arg"

#属性方法删除

del employee.doing3.1.3 类的属性与方法及其使用

##### 4.1.3 总结：

在类中的基本概念包括：

类：即模板

实例：即对象

类属性：类中所必须的属性，类名.属性名，不要和实例属性名字相同，要不然会被实例属性屏蔽

实例属性：可以随时自由添加，每个对象中的特有属性

类方法：类中所必须的方法 ，第一个参数cls，需要@classmethod注释

类方法不能操作实例传入的参数，只能操作类属性，类对象以及静态方法

实例方法：类实例化才能调用的方法，实例名.方法名，第一个参数self

在实例方法中，主要负责实例传入参数，不会操作类属性，可以操作静态方法

静态方法：需要使用@staticmethod注释，类和实例都可以操作静态方法

静态方法，和普通方法一样，不会有self之类的默认参数。

属性方法：可以将方法作为属性应用。具体应用场景？

##### 4.1.4 Python中的继承，多态与方法重载

在python中可以使用多继承

多态，在python中常用重载

##### 4.1.5 使用继承与组合

#### 4.2 Python 命名规范

##### 4.2.1 标识符的命名

所谓的标识符，就是对于自定义元素的命名。即，给自己声明的变量，方法，类等起一个合适的名字。而默认的命名规范如下：

|  |  |
| --- | --- |
| Python中的各种元素 | 命名示例 |
| **模块名/包名** | my\_package\_one |
| **类名** | MyClass |
| **函数名/方法名** | myfunc，my\_example |
| **变量名** | color=‘白色’ |
| **全局常量** | PI=3.14 |
| **全局变量** | my\_value |
| **异常** | MyException |

#### 4.3 Python 网络编程

#### 4.4 Python GUI编程

#### 4.5 Python设计模式

### 第五章 Python 爬虫

##### 5.1 正则表达式

### 第六章 Python 数据分析

### 第七章 Python数据可视化

### 第八章 Python机器学习