Python 学习笔记

1. 以#开头的语句是注释，注释是给人看的，可以是任意内容，解释器会忽略掉注释。其他每一行都是一个语句，当语句以冒号“:”结尾时，缩进的语句视为代码块。
2. 字符串是以''或""括起来的任意文本。"I'm OK"。'I\'m \"OK\"!' Python还允许用r''表示''内部的字符串默认不转义。

>>> print r'\\\t\\'

[\\\t\\](file:///\\\t\\)

1. 用'''...'''的格式表示多行内容：

>>> print '''line1  
... line2  
... line3'''  
line1  
line2  
line3

1. 可以把任意数据类型赋值给变量，同一个变量可以反复赋值，而且可以是不同类型的变量。这种变量本身类型不固定的语言称之为动态语言，与之对应的是静态语言。静态语言在定义变量时必须指定变量类型，如果赋值的时候类型不匹配，就会报错。

字符编码

1. Unicode把所有语言都统一到一套编码里，这样就不会再有乱码问题了。Unicode标准也在不断发展，但最常用的是用两个字节表示一个字符（如果要用到非常偏僻的字符，就需要4个字节）。UTF-8编码把Unicode编码转化为“可变长编码”。UTF-8编码把一个Unicode字符根据不同的数字大小编码成1-6个字节，常用的英文字母被编码成1个字节，汉字通常是3个字节，只有很生僻的字符才会被编码成4-6个字节。在计算机内存中，统一使用Unicode编码，当需要保存到硬盘或者需要传输的时候，就转换为UTF-8编码。
2. Python提供了ord()和chr()函数，可以把字母和对应的数字相互转换：  
   >>> ord('A')  
   65  
   >>> chr(65)  
   'A'
3. 以Unicode表示的字符串用u'...'表示，比如：

>>> print u'中文'

中文

>>> u'中'

u'\u4e2d'

写u'中'和u'\u4e2d'是一样的，\u后面是十六进制的Unicode码

1. 把u'xxx'转换为UTF-8编码的'xxx'用encode('utf-8')方法：

>>> u'ABC'.encode('utf-8')

'ABC'

>>> u'中文'.encode('utf-8')

'\xe4\xb8\xad\xe6\x96\x87'

1. 把UTF-8编码表示的字符串'xxx'转换为Unicode字符串u'xxx'用decode('utf-8')方法：>>> 'abc'.decode('utf-8') u'abc'
2. len()函数可以返回字符串的长度：>>> len(u'ABC') 3
3. 源代码中包含中文的时候，通常在文件开头写上这两行：

#!/usr/bin/env python

# -\*- coding: utf-8 -\*-

第一行注释是为了告诉Linux/OS X系统，这是一个Python可执行程序，Windows系统会忽略这个注释；第二行注释是为了告诉Python解释器，按照UTF-8编码读取源代码，否则，你在源代码中写的中文输出可能会有乱码。

1. list里面的元素的数据类型也可以不同，比如：>>> L = ['Apple', 123, True]
2. tuple另一种有序列表叫元组：tuple。tuple和list非常类似，但是tuple一旦初始化就不能修改，
3. 只有1个元素的tuple定义时必须加一个逗号,，来消除歧义：>>> t = (1,)
4. if <条件判断1>:  
   <执行1>  
   elif <条件判断2>:  
   <执行2>  
   elif <条件判断3>:  
   <执行3>  
   else:  
   <执行4>
5. range()函数，可以生成一个整数序列，比如range(5)生成的序列是从0开始小于5的整数：>>> range(5) [0, 1, 2, 3, 4]
6. sum = 0  
   for x in range(101):  
   sum = sum + x  
   print sum
7. sum = 0  
   n = 99  
   while n > 0:  
   sum = sum + n  
   n = n - 2  
   print sum
8. birth = int(raw\_input('birth: ')) # raw\_input()读取的内容永远以字符串的形式返回，必须先用int()把字符串转换为我们想要的整型：  
   if birth < 2000:  
   print '00前'  
   else:  
   print '00后'
9. dict全称dictionary，在其他语言中也称为map，使用键-值（key-value）存储，具有极快的查找速度。  
   >>> d = {'Michael': 95, 'Bob': 75, 'Tracy': 85}  
   >>> d['Michael']  
   95  
   把数据放入dict的方法，除了初始化时指定外，还可以通过key放入：  
   >>> d['Adam'] = 67  
   >>> d['Adam']  
   67  
   要避免key不存在的错误，有两种办法，一是通过in判断key是否存在：  
   >>> 'Thomas' in d  
   False  
   二是通过dict提供的get方法，如果key不存在，可以返回None，或者自己指定的value：  
   >>> d.get('Thomas')  
   要删除一个key，用pop(key)方法，对应的value也会从dict中删除：  
   >>> d.pop('Bob')  
   75  
   务必注意，dict内部存放的顺序和key放入的顺序是没有关系的。  
   和list比较，dict有以下几个特点：  
   1. 查找和插入的速度极快，不会随着key的增加而增加；  
   2. 需要占用大量的内存，内存浪费多。  
   需要牢记的第一条就是dict的key必须是不可变对象,例如整数，字符串， （不能用list，因为其可变）, 通过key计算位置的算法称为哈希算法（Hash）。
10. set和dict类似，也是一组key的集合，但不存储value。由于key不能重复，所以，在set中，没有重复的key。要创建一个set，需要提供一个list作为输入集合：  
    >>> s = set([1, 2, 3])  
    >>> s  
    set([1, 2, 3])  
    注意，传入的参数[1, 2, 3]是一个list，而显示的set([1, 2, 3])只是告诉你这个set内部有1，2，3这3个元素，显示的[]不表示这是一个list。  
    add(key)方法可以添加元素到set中，s.add(4).remove(key)方法可以删除元素  
    set可以看成数学意义上的无序和无重复元素的集合，因此，两个set可以做数学意义上的交集、并集等操作：  
    >>> s1 = set([1, 2, 3])  
    >>> s2 = set([2, 3, 4])  
    >>> s1 & s2  
    set([2, 3])  
    >>> s1 | s2  
    set([1, 2, 3, 4])
11. 不可变对象。对于不变对象来说，调用对象自身的任意方法，也不会改变该对象自身的内容。相反，这些方法会创建新的对象并返回，这样，就保证了不可变对象本身永远是不可变的。  
    >>> a = 'abc'  
    >>> b = a.replace('a', 'A')  
    >>> b  
    'Abc'  
    >>> a  
    'abc'
12. 函数。在Python中，定义一个函数要使用def语句，依次写出函数名、括号、括号中的参数和冒号:，然后，在缩进块中编写函数体，函数的返回值用return语句返回。  
    def my\_abs(x):  
     if x >= 0:  
     return x  
     else:  
     return -x
13. 空函数  
    def nop():  
    pass  
    pass语句什么都不做，那有什么用？实际上pass可以用来作为占位符，比如现在还没想好怎么写函数的代码，就可以先放一个pass。
14. 返回多个值  
    函数返回值是一个tuple！但是，在语法上，返回一个tuple可以省略括号，而多个变量可以同时接收一个tuple，按位置赋给对应的值，所以，Python的函数返回多值其实就是返回一个tuple，但写起来更方便。
15. 默认参数  
    一个注册函数把年龄和城市设为默认参数：  
    def enroll(name, gender, age=6, city='Beijing'):  
     print 'name:', name  
     print 'gender:', gender  
     print 'age:', age  
     print 'city:', city  
    有多个默认参数时，调用的时候，既可以按顺序提供默认参数，比如调用enroll('Bob', 'M',7)  
    也可以不按顺序提供部分默认参数。当不按顺序提供部分默认参数时，需要把参数名写上。比如调用enroll('Adam', 'M', city='Tianjin')，  
    定义默认参数要牢记一点：默认参数必须指向不变对象！因为Python函数在定义的时候，默认参数的值就被计算出来了，如果默认参数指向的是可变对象，那么当调用该函数时改变了对象的值后，下次调用时，默认参数的内容就变了，导致逻辑错误。
16. 可变参数  
    def calc(\*numbers):  
     sum = 0  
     for n in numbers:  
     sum = sum + n \* n  
     return sum  
    定义可变参数和定义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。  
    如果已经有一个list或者tuple，要调用一个可变参数怎么办？Python允许在list或tuple前面加一个\*号，把list或tuple的元素变成可变参数传进去。  
    >>> nums = [1, 2, 3]  
    >>> calc(\*nums)  
    （>>> calc(nums[0], nums[1], nums[2])这种写法当然是可行的，问题是太繁琐）
17. 关键字参数  
    可变参数允许你传入0个或任意个参数，这些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。  
    def person(name, age, \*\*kw):  
     print 'name:', name, 'age:', age, 'other:', kw  
    >>> person('Adam', 45, gender='M', job='Engineer')  
     name: Adam age: 45 other: {'gender': 'M', 'job': 'Engineer'}
18. 参数组合  
    在Python中定义函数，可以用必选参数、默认参数、可变参数和关键字参数，这4种参数都可以一起使用，或者只用其中某些，但是请注意，参数定义的顺序必须是：必选参数、默认参数、可变参数和关键字参数。  
    def func(a, b, c=0, \*args, \*\*kw):  
     print 'a =', a, 'b =', b, 'c =', c, 'args =', args, 'kw =', kw  
    >>> func(1, 2, 3, 'a', 'b', x=99)  
    a = 1 b = 2 c = 3 args = ('a', 'b') kw = {'x': 99}  
    或通过tuple和dict调用  
    >>> args = (1, 2, 3, 4)  
    >>> kw = {'x': 99}  
    >>> func(\*args, \*\*kw)  
    a = 1 b = 2 c = 3 args = (4,) kw = {'x': 99}
19. 默认参数一定要用不可变对象，如果是可变对象，运行会有逻辑错误！  
    要注意定义可变参数和关键字参数的语法：  
    \*args是可变参数，args接收的是一个tuple；  
    \*\*kw是关键字参数，kw接收的是一个dict。  
    以及调用函数时如何传入可变参数和关键字参数的语法：  
    可变参数既可以直接传入：func(1, 2, 3)，又可以先组装list或tuple，再通过\*args传入：func(\*(1, 2, 3))；  
    关键字参数既可以直接传入：func(a=1, b=2)，又可以先组装dict，再通过\*\*kw传入：func(\*\*{'a': 1, 'b': 2})。
20. 递归函数：一个函数在内部调用自身本身。递归函数的优点是定义简单，逻辑清晰。理论上，所有的递归函数都可以写成循环的方式，但循环的逻辑不如递归清晰。  
    def fact(n):  
     if n==1:  
     return 1  
     return n \* fact(n - 1)  
    使用递归函数需要注意防止栈溢出。解决递归调用栈溢出的方法是通过尾递归优化，事实上尾递归和循环的效果是一样的，所以，把循环看成是一种特殊的尾递归函数也是可以的。尾递归是指，在函数返回的时候，调用自身本身，并且，return语句不能包含表达式。这样，编译器或者解释器就可以把尾递归做优化，使递归本身无论调用多少次，都只占用一个栈帧，不会出现栈溢出的情况。
21. Python 的高级特性，切片  
    一个list如下：  
    >>> L = ['Michael', 'Sarah', 'Tracy', 'Bob', 'Jack']  
    取前3个元素  
    >>> L[0:3]  
    ['Michael', 'Sarah', 'Tracy']  
    L[0:3]表示，从索引0开始取，直到索引3为止，但不包括索引3。  
    Python支持L[-1]取倒数第一个元素，也支持倒数切片：  
    >>> L[-2:]  
    ['Bob', 'Jack']  
    >>> L[-2:-1]  
    ['Bob']  
    前10个数，每两个取一个：  
    >>> L[:10:2]  
    所有数，每5个取一个：  
    >>> L[::5]  
    tuple也可以用切片操作，只是操作的结果仍是tuple。  
    >>> (0, 1, 2, 3, 4, 5)[:3]  
    (0, 1, 2)  
    字符串也可以用切片操作，只是操作结果仍是字符串：  
    >>> 'ABCDEFG'[:3]  
    'ABC'  
    >>> 'ABCDEFG'[::2]  
    'ACEG'
22. 迭代  
    Python的for循环不仅可以用在list或tuple上，还可以作用在其他可迭代对象上，比如dict就可以迭代：  
    >>> d = {'a': 1, 'b': 2, 'c': 3}  
    >>> for key in d:  
    ... print key  
    ...  
    默认情况下，dict迭代的是key。如果要迭代value，可以用for value in d.itervalues()，如果要同时迭代key和value，可以用for k, v in d.iteritems()。  
    怎么对list实现类似Java那样的下标循环。Python内置的enumerate函数可以把一个list变成索引-元素对，这样就可以在for循环中同时迭代索引和元素本身：  
    >>> for i, value in enumerate(['A', 'B', 'C']):  
    ... print i, value  
    ...  
    0 A  
    1 B  
    2 C  
    如何判断一个对象是可迭代对象呢？方法是通过collections模块的Iterable类型判断：  
    >>> from collections import Iterable  
    >>> isinstance('abc', Iterable) # str是否可迭代  
    True  
    >>> isinstance([1,2,3], Iterable) # list是否可迭代  
    True  
    >>> isinstance(123, Iterable) # 整数是否可迭代  
    False