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\\
   ```

3. 用’’’...’’’的格式表示多行内容：

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

4. 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

5. 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。

6. 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'

7. 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

8. 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'

9. 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'

10. len()函数可以返回字符串的长度：>>> len(u'ABC') 3

11. 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。

12. list 里面的元素的数据类型也可以不同，比如：>>> L = ['Apple', 123, True]

13. 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，

14. 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,)

15. 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>

16. 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]

17. sum = 0
    for x in range(101):
    sum = sum + x
    print sum

18. sum = 0
    n = 99
    while n > 0:
    sum = sum + n
    n = n - 2
    print sum

19. 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'

20. 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）。

21. 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])
```

22. 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'
```

23. 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
```

24. 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。

25. 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。

26. 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。

27. 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）

28. 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'}
```

29. 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}
```

30. 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})。

31. 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。

32. 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'
```

33. 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
```

34.

35.

36.

37.

38.

39.