**Python基础学习笔记之（一）**

[zouxy09@qq.com](mailto:zouxy09@qq.com)

<http://blog.csdn.net/zouxy09>

       前段时间参加微软的windows Azure云计算的一个小培训，其中Python被用的还是蛮多的。另外，一些大公司如Google（实现web爬虫和搜索引擎中的很多组件），Yahoo（管理讨论组），NASA，YouTube（视频分享服务大部分由Python编写）等等对Python都很青睐。而国内的豆瓣可以说是给Python予千万宠爱了，它的前台后台清一色的都是Python的身影。另外，我们计算机视觉这块用的很频繁的OpenCV也提供了Python的接口，网上还提供了不少Python的机器学习的库（例如[milk](http://pythonhosted.org/milk/)，[scikit-learn](http://scikit-learn.org/stable/)，[Pylearn2](https://github.com/lisa-lab/pylearn2)等），Deep learning的一个知名的Python的库[theano](http://deeplearning.net/software/theano/)，自然语言处理的库[NLTK](http://nltk.org/)。此外，Python为数学、科学、工程和绘图等提供了有趣的标准库（例如，NumPy ，SciPy和matplotlib等），这使得一部分Matlab的使用者慢慢的倒戈到Python阵营，没办法，谁叫Matlab贵呢！当然了，Python不仅是免费，它还具有Matlab较弱或者没有的一些其他的功能，例如文件管理、界面设计、网络通信等。这就使得Python占有的用户群更广。

       Python是“蟒蛇”的意思，这个名字是有点故事的。当然了，这个得追溯到它的发明者Guido van Rossum。在1989年圣诞节期间，Guido身处阿姆斯特丹。这个都市的美丽和繁华没能填满Guido空虚的内心，周边的喧闹使他内心的寂寞无处安放（夜的黑已不再纯粹，哈哈）。为了打发这种无趣，Guido决心为他之前孕育的ABC语言开发一个插件，这个插件就是大名鼎鼎的Python（男人因为孤独而优秀啊！）。一个新的脚本解释程序就此诞生。在给这个孩子起名字的时候，Guido取其所爱的一个叫Monty Python的喜剧团体的“Python”。



       官方点说，Python是一种解释型、面向对象、动态数据类型的高级程序设计语言。Python语法简洁而清晰，具有丰富和强大的类库。它常被昵称为胶水语言，它能够很轻松的把用其他语言制作的各种模块（尤其是C/C++）轻松地联结在一起。自从20世纪90年代初Python语言诞生至今，它逐渐被广泛应用于处理系统管理任务和Web编程。Python已经成为最受欢迎的程序设计语言之一。2011年1月，它被TIOBE编程语言排行榜评为2010年度语言。自从2004年以后，python的使用率是呈线性增长。

       这篇博文是我看了“中谷教育-Python视频教程”的一个笔记，记录的是一些Python的学习基础，整理到这里，一是总结，二是备查。网上也有比较好的教程，例如“Python 精要参考”。

**目录**

一、安装、编译与运行

二、变量、运算与表达式

三、数据类型

      1、数字

      2、字符串

      3、元组

      4、列表

      5、字典

四、流程控制

      1、if-else

      2、for

      3、while

      4、switch

五、函数

      1、自定义函数

      2、Lambda函数

      3、Python内置函数

六、包与模块

      1、模块module

      2、包package

七、正则表达式

      1、元字符

      2、常用函数

      3、分组

      4、一个小实例-爬虫

八、深拷贝与浅拷贝

九、文件与目录

      1、文件读写

      2、OS模块

      3、目录遍历

十、异常处理

**一、安装、编译与运行**

       Python的安装很容易，直接到官网：<http://www.python.org/>下载安装就可以了。Ubuntu一般都预安装了。没有的话，就可以#apt-get install python。Windows的话直接下载msi包安装即可。Python 程序是通过解释器执行的，所以安装后，可以看到Python提供了两个解析器，一个是IDLE (Python GUI)，一个是Python (command line)。前者是一个带GUI界面的版本，后者实际上和在命令提示符下运行python是一样的。运行解释器后，就会有一个命令提示符>>>，在提示符后键入你的程序语句，键入的语句将会立即执行。就像Matlab一样。

       另外，Matlab有.m的脚步文件，python也有.py后缀的脚本文件，这个文件除了可以解释执行外，还可以编译运行，编译后运行速度要比解释运行要快。

       例如，我要打印一个helloWorld。

方法1：直接在解释器中，>>> print ‘helloWorld’。

方法2：将这句代码写到一个文件中，例如hello.py。运行这个文件有三种方式：

1）在终端中：python hello.py

2）先编译成.pyc文件：

import py\_compile

py\_compile.compile("hello.py")

再在终端中：python hello.pyc

3）在终端中：

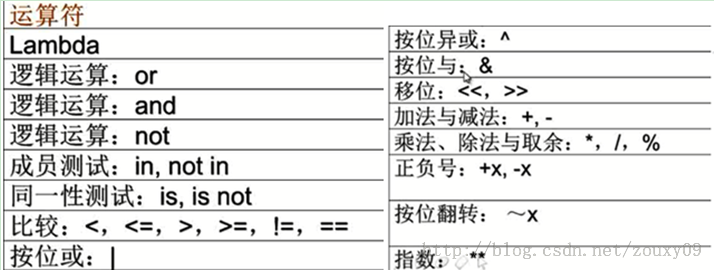
python -O -m py\_compile hello.py

python hello.pyo

       编译成.pyc和.pyo文件后，执行的速度会更快。所以一般一些重复性并多次调用的代码会被编译成这两种可执行的方式来待调用。

**二、变量、运算与表达式**

         这里没什么好说的，有其他语言的编程基础的话都没什么问题。和Matlab的相似度比较大。这块差别不是很大。具体如下：



         需要注意的一个是：5/2 等于2，5.0/2才等于2.5。

**[python]** [view plaincopy](http://blog.csdn.net/zouxy09/article/details/16920541)[在CODE上查看代码片](https://code.csdn.net/snippets/79655)

1. ###################################
2. ### compute #######
3. # raw\_input() get input from keyboard to string type
4. # So we should transfer to int type
5. # Some new support computing type:
6. # and or not in is < <= != == | ^ & << + - / % ~ \*\*
7. **print** 'Please input a number:'
8. number = int(raw\_input())
9. number += 1
10. **print** number\*\*2 # \*\* means ^
11. **print** number **and** 1
12. **print** number **or** 1
13. **print** **not** number
14. 5/2 # is 2
15. 5.0/2 # is 2.5, should be noted

**三、数据类型**

**1、数字**

         通常的int, long,float,long等等都被支持。而且会看你的具体数字来定义变量的类型。如下：

**[python]** [view plaincopy](http://blog.csdn.net/zouxy09/article/details/16920541)[在CODE上查看代码片](https://code.csdn.net/snippets/79655)

1. ###################################
2. ### type of value #######
3. # int, long, float
4. # do not need to define the type of value, python will
5. # do this according to your value
6. num = 1   # stored as int type
7. num = 1111111111111   # stored as long int type
8. num = 1.0   # stored as float type
9. num = 12L # L stands for long type
10. num = 1 + 12j # j stands for complex type
11. num = '1' # string type

**2、字符串**

         单引号，双引号和三引号都可以用来定义字符串。三引号可以定义特别格式的字符串。字符串作为一种序列类型，支持像Matlab一样的索引访问和切片访问。

**[python]** [view plaincopy](http://blog.csdn.net/zouxy09/article/details/16920541)[在CODE上查看代码片](https://code.csdn.net/snippets/79655)

1. ###################################
2. ### type of string #######
3. num = "1" # string type
4. num = "Let's go" # string type
5. num = "He's \"old\"" # string type
6. mail = "Xiaoyi: \n hello \n I am you!"
7. mail = """Xiaoyi:
8. hello
9. I am you!
10. """ # special string format
11. string = 'xiaoyi' # get value by index
12. copy = string[0] + string[1] + string[2:6] # note: [2:6] means [2 5] or[2 6)
13. copy = string[:4] # start from 1
14. copy = string[2:] # to end
15. copy = string[::1] # step is 1, from start to end
16. copy = string[::2] # step is 2
17. copy = string[-1] # means 'i', the last one
18. copy = string[-4:-2:-1] # means 'yoa', -1 step controls direction
19. memAddr = id(num) # id(num) get the memory address of num
20. type(num) # get the type of num

**3、元组**

         元组tuple用()来定义。相当于一个可以存储不同类型数据的一个数组。可以用索引来访问，但需要注意的一点是，里面的元素不能被修改。

**[python]** [view plaincopy](http://blog.csdn.net/zouxy09/article/details/16920541)[在CODE上查看代码片](https://code.csdn.net/snippets/79655)

1. ###################################
2. ### sequence type #######
3. ## can access the elements by index or slice
4. ## include: string, tuple(or array? structure? cell?), list
5. # basis operation of sequence type
6. firstName = 'Zou'
7. lastName = 'Xiaoyi'
8. len(string) # the length
9. name = firstName + lastName # concatenate 2 string
10. firstName \* 3 # repeat firstName 3 times
11. 'Z' **in** firstName # check contain or not, return true
12. string = '123'
13. max(string)
14. min(string)
15. cmp(firstName, lastName) # return 1, -1 or 0
17. ## tuple(or array? structure? cell?)
18. ## define this type using ()
19. user = ("xiaoyi", 25, "male")
20. name = user[0]
21. age = user[1]
22. gender = user[2]
23. t1 = () # empty tuple
24. t2 = (2, ) # when tuple has only one element, we should add a extra comma
25. user[1] = 26 # error!! the elements can not be changed
26. name, age, gender = user # can get three element respectively
27. a, b, c = (1, 2, 3)

**4、列表**

         列表list用[]来定义。它和元组的功能一样，不同的一点是，里面的元素可以修改。List是一个类，支持很多该类定义的方法，这些方法可以用来对list进行操作。

**[python]** [view plaincopy](http://blog.csdn.net/zouxy09/article/details/16920541)[在CODE上查看代码片](https://code.csdn.net/snippets/79655)

1. ## list type (the elements can be modified)
2. ## define this type using []
3. userList = ["xiaoyi", 25, "male"]
4. name = userList[0]
5. age = userList[1]
6. gender = userList[2]
7. userList[3] = 88888 # error! access out of range, this is different with Matlab
8. userList.append(8888) # add new elements
9. "male" **in** userList # search
10. userList[2] = 'female' # can modify the element (the memory address not change)
11. userList.remove(8888) # remove element
12. userList.remove(userList[2]) # remove element
13. **del**(userList[1]) # use system operation api
14. ## help(list.append)
16. ################################
17. ######## object and class ######
18. ## object = property + method
19. ## python treats anything as class, here the list type is a class,
20. ## when we define a list "userList", so we got a object, and we use
21. ## its method to operate the elements

**5、字典**

         字典dictionary用{}来定义。它的优点是定义像key-value这种键值对的结构，就像struct结构体的功能一样。它也支持字典类支持的方法进行创建和操作。

**[python]** [view plaincopy](http://blog.csdn.net/zouxy09/article/details/16920541)[在CODE上查看代码片](https://code.csdn.net/snippets/79655)

1. ################################
2. ######## dictionary type ######
3. ## define this type using {}
4. item = ['name', 'age', 'gender']
5. value = ['xiaoyi', '25', 'male']
6. zip(item, value) # zip() will produce a new list:
7. # [('name', 'xiaoyi'), ('age', '25'), ('gender', 'male')]
8. # but we can not define their corresponding relationship
9. # and we can define this relationship use dictionary type
10. # This can be defined as a key-value manner
11. # dic = {key1: value1, key2: value2, ...}, key and value can be any type
12. dic = {'name': 'xiaoyi', 'age': 25, 'gender': 'male'}
13. dic = {1: 'zou', 'age':25, 'gender': 'male'}
14. # and we access it like this: dic[key1], the key as a index
15. **print** dic['name']
16. **print** dic[1]
17. # another methods create dictionary
18. fdict = dict(['x', 1], ['y', 2]) # factory mode
19. ddict = {}.fromkeys(('x', 'y'), -1) # built-in mode, default value is the same which is none
20. # access by for circle
21. **for** key **in** dic
22. **print** key
23. **print** dic[key]
25. # add key or elements to dictionary, because dictionary is out of sequence,
26. # so we can directly and a key-value pair like this:
27. dic['tel'] = 88888
28. # update or delete the elements
29. **del** dic[1] # delete this key
30. dic.pop('tel') # show and delete this key
31. dic.clear() # clear the dictionary
32. **del** dic # delete the dictionary
33. dic.get(1) # get the value of key
34. dic.get(1, 'error') # return a user-define message if the dictionary do not contain the key
35. dic.keys()
36. dic.values()
37. dic.has\_key(key)
38. # dictionary has many operations, please use help to check out

**四、流程控制**

在这块，Python与其它大多数语言有个非常不同的地方，Python语言使用缩进块来表示程序逻辑（其它大多数语言使用大括号等）。例如：

if age < 21:

    print("你不能买酒。")

    print("不过你能买口香糖。")

print("这句话处于if语句块的外面。")

         这个代码相当于c语言的：

if (age < 21)

{

    print("你不能买酒。")

    print("不过你能买口香糖。")

}

print("这句话处于if语句块的外面。")

       可以看到，Python语言利用缩进表示语句块的开始和退出（Off-side规则），而非使用花括号或者某种关键字。增加缩进表示语句块的开始（注意前面有个:号），而减少缩进则表示语句块的退出。根据PEP的规定，必须使用4个空格来表示每级缩进（不清楚4个空格的规定如何，在实际编写中可以自定义空格数，但是要满足每级缩进间空格数相等）。使用Tab字符和其它数目的空格虽然都可以编译通过，但不符合编码规范。

       为了使我们自己编写的程序能很好的兼容别人的程序，我们最好还是按规范来，用四个空格来缩减（注意，要么都是空格，要是么都制表符，千万别混用）。

**1、if-else**

         If-else用来判断一些条件，以执行满足某种条件的代码。

**[python]** [view plaincopy](http://blog.csdn.net/zouxy09/article/details/16920541)[在CODE上查看代码片](https://code.csdn.net/snippets/79655)

1. ################################
2. ######## procedure control #####
3. ## if else
4. **if** expression: # bool type and do not forget the colon
5. statement(s) # use four space key
7. **if** expression:
8. statement(s) # error!!!! should use four space key
10. **if** 1<2:
11. **print** 'ok, ' # use four space key
12. **print** 'yeah' # use the same number of space key
14. **if** True: # true should be big letter True
15. **print** 'true'
17. **def** fun():
18. **return** 1
20. **if** fun():
21. **print** 'ok'
22. **else**:
23. **print** 'no'
25. con = int(raw\_input('please input a number:'))
26. **if** con < 2:
27. **print** 'small'
28. **elif** con > 3:
29. **print** 'big'
30. **else**:
31. **print** 'middle'
33. **if** 1 < 2:
34. **if** 2 < 3:
35. **print** 'yeah'
36. **else**:
37. **print** 'no'
38. **print** 'out'
39. **else**:
40. **print** 'bad'
42. **if** 1<2 **and** 2<3 **or** 2 < 4 **not** 0: # and, or, not
43. **print** 'yeah'

**2、for**

         for的作用是循环执行某段代码。还可以用来遍历我们上面所提到的序列类型的变量。

**[python]** [view plaincopy](http://blog.csdn.net/zouxy09/article/details/16920541)[在CODE上查看代码片](https://code.csdn.net/snippets/79655)

1. ################################
2. ######## procedure control #####
3. ## for
4. **for** iterating\_val **in** sequence:
5. statements(s)
6. # sequence type can be string, tuple or list
8. **for** i **in** "abcd":
9. **print** i
11. **for** i **in** [1, 2, 3, 4]:
12. **print** i
14. # range(start, end, step), if not set step, default is 1,
15. # if not set start, default is 0, should be noted that it is [start, end), not [start, end]
16. range(5) # [0, 1, 2, 3, 4]
17. range(1, 5) # [1, 2, 3, 4]
18. range(1, 10, 2) # [1, 3, 5, 7, 9]
19. **for** i **in** range(1, 100, 1):
20. **print** i
22. # ergodic for basis sequence
23. fruits = ['apple', 'banana', 'mango']
24. **for** fruit **in** range(len(fruits)):
25. **print** 'current fruit: ', fruits[fruit]
27. # ergodic for dictionary
28. dic = {1: 111, 2: 222, 5: 555}
29. **for** x **in** dic:
30. **print** x, ': ', dic[x]
32. dic.items() # return [(1, 111), (2, 222), (5, 555)]
33. **for** key,value **in** dic.items(): # because we can: a,b=[1,2]
34. **print** key, ': ', value
35. **else**:
36. **print** 'ending'
38. ################################
39. **import** time
40. # we also can use: break, continue to control process
41. **for** x **in** range(1, 11):
42. **print** x
43. time.sleep(1) # sleep 1s
44. **if** x == 3:
45. **pass** # do nothing
46. **if** x == 2:
47. **continue**
48. **if** x == 6:
49. **break**
50. **if** x == 7:
51. exit() # exit the whole program
52. **print** '#'\*50

**3、while**

         while的用途也是循环。它首先检查在它后边的循环条件，若条件表达式为真，它就执行冒号后面的语句块，然后再次测试循环条件，直至为假。冒号后面的缩近语句块为循环体。

**[python]** [view plaincopy](http://blog.csdn.net/zouxy09/article/details/16920541)[在CODE上查看代码片](https://code.csdn.net/snippets/79655)

1. ################################
2. ######## procedure control #####
3. ## while
4. **while** expression:
5. statement(s)
7. **while** True:
8. **print** 'hello'
9. x = raw\_input('please input something, q for quit:')
10. **if** x == 'q':
11. **break**
12. **else**:
13. **print** 'ending'

**4、switch**

         其实Python并没有提供switch结构，但我们可以通过字典和函数轻松的进行构造。例如：

**[python]** [view plaincopy](http://blog.csdn.net/zouxy09/article/details/16920541)[在CODE上查看代码片](https://code.csdn.net/snippets/79655)

1. #############################
2. ## switch ####
3. ## this structure do not support by python
4. ## but we can implement it by using dictionary and function
5. ## cal.py ##
6. #!/usr/local/python
8. **from** \_\_future\_\_ **import** division
9. # if used this, 5/2=2.5, 6/2=3.0
11. **def** add(x, y):
12. **return** x + y
13. **def** sub(x, y):
14. **return** x - y
15. **def** mul(x, y):
16. **return** x \* y
17. **def** div(x, y):
18. **return** x / y
20. operator = {"+": add, "-": sub, "\*": mul, "/": div}
21. operator["+"](1, 2) # the same as add(1, 2)
22. operator["%"](1, 2) # error, not have key "%", but the below will not
23. operator.get("+")(1, 2) # the same as add(1, 2)
25. **def** cal(x, o, y):
26. **print** operator.get(o)(x, y)
27. cal(2, "+", 3)
28. # this method will effect than if-else

**五、函数**

**1、自定义函数**

         在Python中，使用def语句来创建函数：

**[python]** [view plaincopy](http://blog.csdn.net/zouxy09/article/details/16920541)[在CODE上查看代码片](https://code.csdn.net/snippets/79655)

1. ################################
2. ######## function #####
3. **def** functionName(parameters): # no parameters is ok
4. bodyOfFunction
6. **def** add(a, b):
7. **return** a+b # if we do not use a return, any defined function will return default None
9. a = 100
10. b = 200
11. sum = add(a, b)
13. ##### function.py #####
14. #!/usr/bin/python
15. #coding:utf8  # support chinese
16. **def** add(a = 1, b = 2): # default parameters
17. **return** a+b  # can return any type of data
18. # the followings are all ok
19. add()
20. add(2)
21. add(y = 1)
22. add(3, 4)
24. ###### the global and local value #####
25. ## global value: defined outside any function, and can be used
26. ##              in anywhere, even in functions, this should be noted
27. ## local value: defined inside a function, and can only be used
28. ##              in its own function
29. ## the local value will cover the global if they have the same name
30. val = 100 # global value
31. **def** fun():
32. **print** val # here will access the val = 100
33. **print** val # here will access the val = 100, too
35. **def** fun():
36. a = 100 # local value
37. **print** a
38. **print** a # here can not access the a = 100
40. **def** fun():
41. **global** a = 100 # declare as a global value
42. **print** a
44. **print** a # here can not access the a = 100, because fun() not be called yet
45. fun()
46. **print** a # here can access the a = 100
48. ############################
49. ## other types of parameters
50. **def** fun(x):
51. **print** x
52. # the follows are all ok
53. fun(10) # int
54. fun('hello') # string
55. fun(('x', 2, 3))  # tuple
56. fun([1, 2, 3])    # list
57. fun({1: 1, 2: 2}) # dictionary
59. ## tuple
60. **def** fun(x, y):
61. **print** "%s : %s" % (x,y) # %s stands for string
62. fun('Zou', 'xiaoyi')
63. tu = ('Zou', 'xiaoyi')
64. fun(\*tu)    # can transfer tuple parameter like this
66. ## dictionary
67. **def** fun(name = "name", age = 0):
68. **print** "name: %s" % name
69. **print** "age: " % age
70. dic = {name: "xiaoyi", age: 25} # the keys of dictionary should be same as fun()
71. fun(\*\*dic) # can transfer dictionary parameter like this
72. fun(age = 25, name = 'xiaoyi') # the result is the same
73. ## the advantage of dictionary is can specify value name
75. #############################
76. ## redundancy parameters ####
77. ## the tuple
78. **def** fun(x, \*args): # the extra parameters will stored in args as tuple type
79. **print** x
80. **print** args
81. # the follows are ok
82. fun(10)
83. fun(10, 12, 24) # x = 10, args = (12, 24)
85. ## the dictionary
86. **def** fun(x, \*\*args): # the extra parameters will stored in args as dictionary type
87. **print** x
88. **print** args
89. # the follows are ok
90. fun(10)
91. fun(x = 10, y = 12, z = 15) # x = 10, args = {'y': 12, 'z': 15}
93. # mix of tuple and dictionary
94. **def** fun(x, \*args, \*\*kwargs):
95. **print** x
96. **print** args
97. **print** kwargs
98. fun(1, 2, 3, 4, y = 10, z = 12) # x = 1, args = (2, 3, 4), kwargs = {'y': 10, 'z': 12}

**2、Lambda函数**

         Lambda函数用来定义一个单行的函数，其便利在于：

**[python]** [view plaincopy](http://blog.csdn.net/zouxy09/article/details/16920541)[在CODE上查看代码片](https://code.csdn.net/snippets/79655)

1. #############################
2. ## lambda function ####
3. ## define a fast single line function
4. fun = **lambda** x,y : x\*y # fun is a object of function class
5. fun(2, 3)
6. # like
7. **def** fun(x, y):
8. **return** x\*y
10. ## recursion
11. # 5=5\*4\*3\*2\*1, n!
12. **def** recursion(n):
13. **if** n > 0:
14. **return** n \* recursion(n-1) ## wrong
16. **def** mul(x, y):
17. **return** x \* y
18. numList = range(1, 5)
19. reduce(mul, numList) # 5! = 120
20. reduce(**lambda** x,y : x\*y, numList) # 5! = 120, the advantage of lambda function avoid defining a function
22. ### list expression
23. numList = [1, 2, 6, 7]
24. filter(**lambda** x : x % 2 == 0, numList)
25. **print** [x **for** x **in** numList **if** x % 2 == 0] # the same as above
26. map(**lambda** x : x \* 2 + 10, numList)
27. **print** [x \* 2 + 10 **for** x **in** numList] # the same as above

**3、Python内置函数**

       Python内置了很多函数，他们都是一个个的.py文件，在python的安装目录可以找到。弄清它有那些函数，对我们的高效编程非常有用。这样就可以避免重复的劳动了。下面也只是列出一些常用的：

**[python]** [view plaincopy](http://blog.csdn.net/zouxy09/article/details/16920541)[在CODE上查看代码片](https://code.csdn.net/snippets/79655)

1. ###################################
2. ## built-in function of python ####
3. ## if do not how to use, please use help()
4. abs, max, min, len, divmod, pow, round, callable,
5. isinstance, cmp, range, xrange, type, id, int()
6. list(), tuple(), hex(), oct(), chr(), ord(), long()
8. callable # test a function whether can be called or not, if can, return true
9. # or test a function is exit or not
11. isinstance # test type
12. numList = [1, 2]
13. **if** type(numList) == type([]):
14. **print** "It is a list"
15. **if** isinstance(numList, list): # the same as above, return true
16. **print** "It is a list"
18. **for** i **in** range(1, 10001) # will create a 10000 list, and cost memory
19. **for** i **in** xrange(1, 10001)# do not create such a list, no memory is cost
21. ## some basic functions about string
22. str = 'hello world'
23. str.capitalize() # 'Hello World', first letter transfer to big
24. str.replace("hello", "good") # 'good world'
25. ip = "192.168.1.123"
26. ip.split('.') # return ['192', '168', '1', '123']
27. help(str.split)
29. **import** string
30. str = 'hello world'
31. string.replace(str, "hello", "good") # 'good world'
33. ## some basic functions about sequence
34. len, max, min
35. # filter(function or none, sequence)
36. **def** fun(x):
37. **if** x > 5:
38. **return** True
39. numList = [1, 2, 6, 7]
40. filter(fun, numList) # get [6, 7], if fun return True, retain the element, otherwise delete it
41. filter(**lambda** x : x % 2 == 0, numList)
42. # zip()
43. name = ["me", "you"]
44. age = [25, 26]
45. tel = ["123", "234"]
46. zip(name, age, tel) # return a list: [('me', 25, '123'), ('you', 26, '234')]
47. # map()
48. map(None, name, age, tel) # also return a list: [('me', 25, '123'), ('you', 26, '234')]
49. test = ["hello1", "hello2", "hello3"]
50. zip(name, age, tel, test) # return [('me', 25, '123', 'hello1'), ('you', 26, '234', 'hello2')]
51. map(None, name, age, tel, test) # return [('me', 25, '123', 'hello1'), ('you', 26, '234', 'hello2'), (None, None, None, 'hello3')]
52. a = [1, 3, 5]
53. b = [2, 4, 6]
54. **def** mul(x, y):
55. **return** x\*y
56. map(mul, a, b) # return [2, 12, 30]
57. # reduce()

**Python基础学习笔记之（二）**

[zouxy09@qq.com](mailto:zouxy09@qq.com)

<http://blog.csdn.net/zouxy09>

**六、包与模块**

**1、模块module**

         Python中每一个.py脚本定义一个模块，所以我们可以在一个.py脚本中定义一个实现某个功能的函数或者脚本，这样其他的.py脚本就可以调用这个模块了。调用的方式有三种，如下：

**[python]** [view plaincopy](http://blog.csdn.net/zouxy09/article/details/16920661)[在CODE上查看代码片](https://code.csdn.net/snippets/79663)

1. ###################################
2. ## package and module ####
3. ## a .py file define a module which can be used in other script
4. ## as a script, the name of module is the same as the name of the .py file
5. ## and we use the name to import to a new script
6. ## e.g., items.py, import items
7. ## python contains many .py files, which we can import and use
8. # vi cal.py
9. **def** add(x, y):
10. **return** x + y
11. **def** sub(x, y):
12. **return** x - y
13. **def** mul(x, y):
14. **return** x \* y
15. **def** div(x, y):
16. **return** x / y
18. **print** "Your answer is: ", add(3, 5)
20. **if** \_\_name\_\_ == "\_\_main\_\_"
21. r = add(1, 3)
22. **print** r
24. # vi test.py
25. **import** cal # will expand cal.py here
26. # so, this will execute the following code in cal.py
27. # print "Your answer is: ", add(3, 5)
28. # it will print "Your answer is: 8"
29. # but as we import cal.py, we just want to use those functions
30. # so the above code can do this for me, the r=add(1, 3) will not execute
31. result = cal.add(1, 2)
32. **print** result
33. # or
34. **import** cal as c
35. result = c.add(1, 2)
36. # or
37. **from** cal **import** add
38. result = add(1, 2)

**2、包package**

       python 的每个.py文件执行某种功能，那有时候我们需要多个.py完成某个更大的功能，或者我们需要将同类功能的.py文件组织到一个地方，这样就可以很方便我们的使用。模块可以按目录组织为包，创建一个包的步骤：

# 1、建立一个名字为包名字的文件夹

# 2、在该文件夹下创建一个\_\_init\_\_.py空文件

# 3、根据需要在该文件夹下存放.py脚本文件、已编译拓展及子包

# 4、import pack.m1,pack.m2 pack.m3

**[python]** [view plaincopy](http://blog.csdn.net/zouxy09/article/details/16920661)[在CODE上查看代码片](https://code.csdn.net/snippets/79663)

1. #### package 包
2. ## python 的模块可以按目录组织为包，创建一个包的步骤：
3. # 1、建立一个名字为包名字的文件夹
4. # 2、在该文件夹下创建一个\_\_init\_\_.py 空文件
5. # 3、根据需要在该文件夹下存放.py脚本文件、已编译拓展及子包
6. # 4、import pack.m1, pack.m2 pack.m3
7. mkdir calSet
8. cd calSet
9. touch \_\_init\_.py
10. cp cal.py .
12. # vi test.py
13. **import** calSet.cal
14. result = calSet.cal.add(1, 2)
15. **print** result

**七、正则表达式**

       正则表达式，（英语：RegularExpression，在代码中常简写为regex、regexp或RE），计算机科学的一个概念。正则表达式使用单个字符串来描述、匹配一系列符合某个句法规则的字符串。在很多文本编辑器里，正则表达式通常被用来检索、替换那些符合某个模式的文本。

         Python提供了功能强大的正则表达式引擎re，我们可以利用这个模块来利用正则表达式进行字符串操作。我们用import re来导入这个模块。

         正则表达式包含了很多规则，如果能灵活的使用，在匹配字符串方面是非常高效率的。更多的规则，我们需要查阅其他的资料。

**1、元字符**

         很多，一些常用的元字符的使用方法如下：

**[python]** [view plaincopy](http://blog.csdn.net/zouxy09/article/details/16920661)[在CODE上查看代码片](https://code.csdn.net/snippets/79663)

1. ##############################
2. ## 正则表达式 RE
3. ## re module in python
4. **import** re
5. rule = r'abc' # r prefix, the rule you want to check in a given string
6. re.findall(rule, "aaaaabcaaaaaabcaa") # return ['abc', 'abc']
8. # [] 用来指定一个字符集 [abc] 表示 abc其中任意一个字符符合都可以
9. rule = r"t[io]p"
10. re.findall(rule, "tip tep twp top") # return ['tip', 'top']
12. # ^ 表示 补集，例如[^io] 表示除i和o外的其他字符
13. rule = r"t[^io]p"
14. re.findall(rule, "tip tep twp top") # return ['tep', 'twp']
16. # ^ 也可以 匹配行首，表示要在行首才匹配，其他地方不匹配
17. rule = r"^hello"
18. re.findall(rule, "hello tep twp hello") # return ['hello']
19. re.findall(rule, "tep twp hello") # return []
21. # $ 表示匹配行尾
22. rule = r"hello$"
23. re.findall(rule, "hello tep twp hello") # return ['hello']
24. re.findall(rule, "hello tep twp") # return []
26. # - 表示范围
27. rule = r"x[0123456789]x" # the same as
28. rule = r"x[0-9]x"
29. re.findall(rule, "x1x x4x xxx") # return ['x1x', 'x4x']
30. rule = r"x[a-zA-Z]x"
32. # \ 表示转义符
33. rule = r"\^hello"
34. re.findall(rule, "hello twp ^hello") # return ['^hello']
35. # \d 匹配一个数字字符。等价于[0-9]。
36. # \D 匹配一个非数字字符。等价于[^0-9]。
37. # \n 匹配一个换行符。等价于\x0a和\cJ。
38. # \r 匹配一个回车符。等价于\x0d和\cM。
39. # \s 匹配任何空白字符，包括空格、制表符、换页符等等。等价于[ \f\n\r\t\v]。
40. # \S 匹配任何非空白字符。等价于[^ \f\n\r\t\v]。
41. # \t 匹配一个制表符。等价于\x09和\cI。
42. # \w 匹配包括下划线的任何单词字符。等价于“[A-Za-z0-9\_]”。
43. # \W 匹配任何非单词字符。等价于“[^A-Za-z0-9\_]”。
45. # {} 表示重复规则
46. # 例如我们要查找匹配是否是 广州的号码，020-八位数据
47. # 以下三种方式都可以实现
48. rule = r"^020-\d\d\d\d\d\d\d\d$"
49. rule = r"^020-\d{8}$" # {8} 表示前面的规则重复8次
50. rule = r"^020-[0-9]{8}$"
51. re.findall(rule, "020-23546813") # return ['020-23546813']
53. # \* 表示将其前面的字符重复0或者多次
54. rule = r"ab\*"
55. re.findall(rule, "a") # return ['a']
56. re.findall(rule, "ab") # return ['ab']
58. # + 表示将其前面的字符重复1或者多次
59. rule = r"ab+"
60. re.findall(rule, "a") # return []
61. re.findall(rule, "ab") # return ['ab']
62. re.findall(rule, "abb") # return ['abb']
64. # ? 表示前面的字符可有可无
65. rule = r"^020-?\d{8}$"
66. re.findall(rule, "02023546813") # return ['020-23546813
67. re.findall(rule, "020-23546813") # return ['020-23546813']
68. re.findall(rule, "020--23546813") # return []
70. # ? 表示非贪婪匹配
71. rule = r"ab+?"
72. re.findall(rule, "abbbbbbb") # return ['ab']
74. # {} 可以表示范围
75. rule = r"a{1,3}"
76. re.findall(rule, "a") # return ['a']
77. re.findall(rule, "aa") # return ['aa']
78. re.findall(rule, "aaa") # return ['aaa']
79. re.findall(rule, "aaaa") # return ['aaa', 'a']
81. ## compile re string
82. rule = r"\d{3,4}-?\d{8}"
83. re.findall(rule, "020-23546813")
84. # faster when you compile it
85. # return a object
86. p\_tel = re.compile(rule)
87. p\_tel.findall("020-23546813")
89. # the parameter re.I 不区分大小写
90. name\_re = re.compile(r"xiaoyi", re.I)
91. name\_re.findall("Xiaoyi")
92. name\_re.findall("XiaoYi")
93. name\_re.findall("xiAOyi")

**2、常用函数**

         Re模块作为一个对象，它还支持很多的操作，例如：

**[python]** [view plaincopy](http://blog.csdn.net/zouxy09/article/details/16920661)[在CODE上查看代码片](https://code.csdn.net/snippets/79663)

1. # the object contain some methods we can use
2. # match 去搜索字符串开头，如果匹配对，那就返回一个对象，否则返回空
3. obj = name\_re.match('Xiaoyi, Zou')
4. # search 去搜索字符串（任何位置），如果匹配对，那就返回一个对象
5. obj = name\_re.search('Zou, Xiaoyi')
6. # 然后可以用它来进行判断某字符串是否存在我们的正则表达式
7. **if** obj:
8. **pass**
9. # findall 返回一个满足正则的列表
10. name\_re.findall("Xiaoyi")
12. # finditer 返回一个满足正则的迭代器
13. name\_re.finditer("Xiaoyi")
15. # 正则替换
16. rs = r"z..x"
17. re.sub(rs, 'python', 'zoux ni ziox me') # return 'python ni python me'
18. re.subn(rs, 'python', 'zoux ni ziox me') # return ('python ni python me', 2), contain a number
20. # 正则切片
21. str = "123+345-32\*78"
22. re.split(r'[\+\-\\*]', str) # return ['123', '345', '32', '78']
24. # 可以打印re模块支持的属性和方法，然后用help
25. dir(re)
27. ##### 编译正则表达式式 可以加入一些属性，可以增加很多功能
28. # 多行匹配
29. str = """
30. hello xiaoyi
31. xiaoyi hello
32. hello zou
33. xiaoyi hello
34. """
35. re.findall(r'xiaoyi', str, re.M)

**3、分组**

         分组有两个作用，它用()来定义一个组，组内的规则只对组内有效。

**[python]** [view plaincopy](http://blog.csdn.net/zouxy09/article/details/16920661)[在CODE上查看代码片](https://code.csdn.net/snippets/79663)

1. # () 分组
2. email = r"\w{3}@\w+(\.com|\.cn|\.org)"
3. re.match(email, "zzz@scut.com")
4. re.match(email, "zzz@scut.cn")
5. re.match(email, "zzz@scut.org")

    另外，分组可以优先返回分组内匹配的字符串。

**[python]** [view plaincopy](http://blog.csdn.net/zouxy09/article/details/16920661)[在CODE上查看代码片](https://code.csdn.net/snippets/79663)

1. # 另外，分组可以优先返回分组内匹配的字符串
2. str = """
3. idk hello name=zou yes ok d
4. hello name=xiaoyi yes no dksl
5. dfi lkasf dfkdf hello name=zouxy yes d
6. """
7. r1 = r"hello name=.+ yes"
8. re.findall(r1, str) # return ['hello name=zou yes', 'hello name=xiaoyi yes', 'hello name=zouxy yes']
9. r2 = r"hello name=(.+) yes"
10. re.findall(r2, str) # return ['zou', 'xiaoyi', 'zouxy']
11. # 可以看到，它会匹配整个正则表达式，但只会返回()括号分组内的字符串，
12. # 用这个属性，我们就可以进行爬虫，抓取一些想要的数据

**4、一个小实例-爬虫**

         这个实例利用上面的正则和分组的优先返回特性来实现一个小爬虫算法。它的功能是到一个给定的网址里面将.jpg后缀的图片全部下载下来。

**[python]** [view plaincopy](http://blog.csdn.net/zouxy09/article/details/16920661)[在CODE上查看代码片](https://code.csdn.net/snippets/79663)

1. ## 一个小爬虫
2. ## 下载贴吧 或 空间中的所有图片
3. ## getJpg.py
5. #!/usr/bin/python
6. **import** re
7. **import** urllib
9. # Get the source code of a website
10. **def** getHtml(url):
11. **print** 'Getting html source code...'
12. page = urllib.open(url)
13. html = page.read()
14. **return** html
16. # Open the website and check up the address of images,
17. # and find the common features to decide the re\_rule
18. **def** getImageAddrList(html):
19. **print** 'Getting all address of images...'
20. rule = r"src=\"(.+\.jpg)\" pic\_ext"
21. imReg = re.compile(rule)
22. imList = re.findall(imReg, html)
23. **return** imList
25. **def** getImage(imList):
26. **print** 'Downloading...'
27. name = 1;
28. **for** imgurl **in** imList:
29. urllib.urlretrieve(imgurl, '%s.jpg' % name)
30. name += 1
31. **print** 'Got ', len(imList), ' images!'
33. ## main
34. htmlAddr = "http://tieba.baidu.com/p/2510089409"
35. html = getHtml(htmlAddr)
36. imList = getImageAddrList(html)
37. getImage(imList)

**八、深拷贝与浅拷贝**

Python中对数据的复制有两个需要注意的差别：

浅拷贝：对引用对象的拷贝（只拷贝父对象），深拷贝：对对象资源的拷贝。具体的差别如下：

**[python]** [view plaincopy](http://blog.csdn.net/zouxy09/article/details/16920661)[在CODE上查看代码片](https://code.csdn.net/snippets/79663)

1. ##############################
2. ### memory operation
3. ## 浅拷贝：对引用对象的拷贝（只拷贝父对象）
4. ## 深拷贝：对对象资源的拷贝
6. a = [1, 2, 3]
7. b = a # id(a) == id (b), 同一个标签，相当于引用
8. a.append(4) # a = [1, 2, 3, 4], and b also change to = [1, 2, 3, 4]
10. **import** copy
11. a = [1, 2, ['a', 'b']] # 二元列表
12. c = copy.copy(a)  # id(c) != id(a)
13. a.append('d') # a = [1, 2, ['a', 'b'], 'd'] but c keeps not changed
14. # 但只属于浅拷贝，只拷贝父对象
15. # 所以 id(a[0]) == id(c[0])，也就是说对a追加的元素不影响c，
16. # 但修改a被拷贝的数据后，c的对应数据也会改变，因为拷贝不会改变元素的地址
17. a[2].append('d') # will change c, too
18. a[1] = 3 # will change c, too
20. # 深拷贝
21. d = copy.deepcopy(a) # 全部拷贝，至此恩断义绝，两者各走
22. # 各的阳关道和独木桥，以后毫无瓜葛

**九、文件与目录**

**1、文件读写**

        Python的文件操作和其他的语言没有太大的差别。通过open或者file类来访问。但python支持了很多的方法，以支持文件内容和list等类型的交互。具体如下：

**[python]** [view plaincopy](http://blog.csdn.net/zouxy09/article/details/16920661)[在CODE上查看代码片](https://code.csdn.net/snippets/79663)

1. ########################
2. ## file and directory
3. # file\_handler = open(filename, mode)
4. # mode is the same as other program langurage
5. ## read
6. # method 1
7. fin = open('./test.txt')
8. fin.read()
9. fin.close()
11. # method 2, class file
12. fin = file('./test.txt')
13. fin.read()
14. fin.close()
16. ## write
17. fin = open('./test.txt', 'r+') # r, r+, w, w+, a, a+, b, U
18. fin.write('hello')
19. fin.close()
21. ### 文件对象的方法
22. ## help(file)
24. **for** i **in** open('test.txt'):
25. **print** i
27. str = fin.readline() # 每次读取一行
28. list = fin.readlines() # 读取多行，返回一个列表，每行作为列表的一个元素
29. fin.next() # 读取改行，指向下一行
31. # 用列表来写入多行
32. fin.writelines(list)
34. # 移动指针
35. fin.seek(0, 0)
36. fin.seek(0, 1)
37. fin.seek(-1, 2)
39. # 提交更新
40. fin.flush() # 平时写数据需要close才真正写入文件，这个函数可以立刻写入文件

**2、OS模块**

         os模块提供了很多对系统的操作。例如对目录的操作等。我们需要用import os来插入这个模块以便使用。

**[python]** [view plaincopy](http://blog.csdn.net/zouxy09/article/details/16920661)[在CODE上查看代码片](https://code.csdn.net/snippets/79663)

1. #########################
2. ## OS module
3. ## directory operation should import this
4. **import** os
6. os.mkdir('xiaoyi') # mkdir
7. os.makedirs('a/b/c', mode = 666) # 创建分级的目录
8. os.listdir() # ls 返回当前层所有文件或者文件夹名到一个列表中（不包括子目录）
9. os.chdir() # cd
10. os.getcwd() # pwd
11. os.rmdir() # rm

**3、目录遍历**

       目录遍历的实现可以做很多普遍的功能，例如杀毒软件，垃圾清除软件，文件搜索软件等等。因为他们都涉及到了扫描某目录下所有的包括子目录下的文件。所以需要对目录进行遍历。在这里我们可以使用两种方法对目录进行遍历：

1）递归

**[python]** [view plaincopy](http://blog.csdn.net/zouxy09/article/details/16920661)[在CODE上查看代码片](https://code.csdn.net/snippets/79663)

1. #!/usr/bin/python
2. #coding:utf8
3. **import** os
5. **def** dirList(path):
6. fileList = os.listdir(path)
7. allFile = []
8. **for** fileName **in** fileList:
9. # allFile.append(dirPath + '/' + fileName) # the same as below
10. filePath = os.path.join(path, fileName)
11. **if** os.path.isdir(filePath):
12. dirList(filePath)
13. allFile.append(filePath)
14. **return** allFile

2）os.walk函数

**[python]** [view plaincopy](http://blog.csdn.net/zouxy09/article/details/16920661)[在CODE上查看代码片](https://code.csdn.net/snippets/79663)

1. # os.walk 返回一个生成器，每次是一个三元组 [目录, 子目录, 文件]
2. gen = os.walk('/')
3. **for** path, dir, filelist **in** os.walk('/'):
4. **for** filename **in** filelist:
5. os.path.join(path, filename)

**十、异常处理**

      异常意味着错误，未经处理的异常会中止程序运行。而异常抛出机制，为程序开发人员提供一种在运行时发现错误，并进行恢复处理，然后继续执行的能力。

**[python]** [view plaincopy](http://blog.csdn.net/zouxy09/article/details/16920661)[在CODE上查看代码片](https://code.csdn.net/snippets/79663)

1. ###################################
2. ### 异常处理
3. # 异常抛出机制，为程序开发人员提供一种在运行时发现错误，
4. # 进行恢复处理，然后继续执行的能力
6. # 用try去尝试执行一些代码，如果错误，就抛出异常，
7. # 异常由except来捕获，并由我们写代码来处理这种异常
8. **try**:
9. fin = open("abc.txt")
10. **print** hello
11. ### your usually process code here
12. **except** IOError, msg:
13. **print** "On such file!"
14. ### your code to handle this error
15. **except** NameError, msg:
16. **print** msg
17. ### your code to handle this error
18. **finally**: # 不管上面有没有异常，这个代码块都会被执行
19. **print** 'ok'
21. # 抛出异常，异常类型要满足python内定义的
22. **if** filename == "hello":
23. **raise** TypeError("Nothing!!")