Python语法

数据分析与数据挖掘

2.1 Python基本语法

- Python标识符
  - 在 Python 里，标识符由字母、数字、下划线组成。
  - 在 Python 中，所有标识符可以包括英文、数字以及下划线(_)，但不能以数字开头。
  - Python 中的标识符是区分大小写的。
  - Python 可以同一行显示多条语句，方法是用分号；分开。
  - 以下划线开头的标识符是有特殊意义的。以单下划线开头 _foo 的代表不能直接访问的类属性，需通过类提供的接口进行访问，不能用 from xxx import * 而导入
  - 以双下划线开头的 __foo 代表类的私有成员；以双下划线开头和结尾的 __foo__ 代表 Python 里特殊方法专用的标识，如 __init__() 代表类的构造函数。

- **Python 保留字符**

  下面的列表显示了在Python中的保留字。这些保留字不能用作常数或变数，或任何其他标识符名称。所有 Python 的关键字只包含小写字母。

and	exec	not
assert	finally	or
break	for	pass
class	from	print
continue	global	raise
def	if	return
del	import	try
elif	in	while
else	is	with
except	lambda	yield

- **行和缩进**

学习 Python 与其他语言最大的区别就是，Python 的代码块不使用大括号 {} 来控制类，函数以及其他逻辑判断。python 最具特色的就是用缩进来写模块。

缩进的空白数量是可变的，但是所有代码块语句必须包含相同的缩进空白数量，这个必须严格执行。以下代码会执行错误：

```
#!/usr/bin/python
# -*- coding: UTF-8 -*-
# 文件名：test.py

 if True:
    print "Answer"
    print "True"
else:
    print "Answer"
    # 没有严格缩进，在执行时会报错
  print "False"
```

- **Python引号**

Python 可以使用引号(')、双引号(")、三引号(''' 或 """) 来表示字符串，引号的开始与结束必须的相同类型的。

其中三引号可以由多行组成，编写多行文本的快捷语法，常用于文档字符串，在文件的特定地点，被当做注释。

```
word = 'word'
sentence = "这是一个句子。"
paragraph = """这是一个段落。
包含了多个语句"""
```

- **Python注释**
  - python中单行注释采用 # 开头
  - python 中多行注释使用三个单引号(''')或三个双引号(""")。

```
#!/usr/bin/python
# -*- coding: UTF-8 -*-
# 文件名：test.py

# 第一个注释
print "Hello, Python!";  # 第二个注释
```

```
#!/usr/bin/python
# -*- coding: UTF-8 -*-
# 文件名：test.py


'''
这是多行注释，使用单引号。
这是多行注释，使用单引号。
这是多行注释，使用单引号。
'''

"""
这是多行注释，使用双引号。
这是多行注释，使用双引号。
这是多行注释，使用双引号。
"""
```

2.2 Python数据类型

- **标准数据类型**

  Python 定义了一些标准类型，用于存储各种类型的数据。

  Python有五个标准的数据类型：

  - Numbers（数字）
  - String（字符串）
  - List（列表）
  - Tuple（元组）
  - Dictionary（字典）

- **Python数字**
  - 数字数据类型用于存储数值。他们是不可改变的数据类型，这意味着改变数字数据类型会分配一个新的对象。当你指定一个值时，Number对象就会被创建。

```
var1 = 1
var2 = 10
```

  - 可以通过使用del语句删除单个或多个对象的引用。例如：

```
del var
del var_a, var_b
```

  - Python支持四种不同的数字类型：
    - int（有符号整型）
    - long（长整型[也可以代表八进制和十六进制]）
    - float（浮点型）
    - complex（复数）

- **Python字符串**
  - 字符串或串(String)是由数字、字母、下划线组成的一串字符。

    s="a1a2···an"(n>=0)

  - python的字串列表有2种取值顺序:
    - 从左到右索引默认0开始的，最大范围是字符串长度少1
    - 从右到左索引默认-1开始的，最大范围是字符串开头
  - 如果你要实现从字符串中获取一段子字符串的话，可以使用变量 **[头下标:尾下标]**，就可以截取相应的字符串，其中下标是从 0 开始算起，可以是正数或负数，下标可以为空表示取到头或尾。

    S='ilovepython‘

    s[1:5]的结果是love。

- **Python列表**
  - List（列表） 是 Python 中使用最频繁的数据类型。
  - 列表可以完成大多数集合类的数据结构实现。它支持字符，数字，字符串甚至可以包含列表（即嵌套）。
  - 列表用 [ ] 标识，是 python 最通用的复合数据类型。
  - 列表中值的切割也可以用到变量 [头下标:尾下标] ，就可以截取相应的列表，从左到右索引默认 0 开始，从右到左索引默认 -1 开始，下标可以为空表示取到头或尾。
  - 加号 + 是列表连接运算符，星号 * 是重复操作。如下实例：

```
#!/usr/bin/python
# -*- coding: UTF-8 -*-

list = [ 'runoob', 786 , 2.23, 'john', 70.2 ]
tinylist = [123, 'john']

print list                # 输出完整列表
print list[0]             # 输出列表的第一个元素
print list[1:3]           # 输出第二个至第三个的元素
print list[2:]            # 输出从第三个开始至列表末尾的所有元素
print tinylist * 2        # 输出列表两次
print list + tinylist     # 打印组合的列表
```

- **Python元组**
  - 元组是另一个数据类型，类似于List（列表）。
  - 元组用"()"标识。内部元素用逗号隔开。但是元组不能二次赋值，相当于只读列表。

```
#!/usr/bin/python
# -*- coding: UTF-8 -*-

tuple = ( 'runoob', 786 , 2.23, 'john', 70.2 )
tinytuple = (123, 'john')

print tuple               # 输出完整元组
print tuple[0]            # 输出元组的第一个元素
print tuple[1:3]          # 输出第二个至第三个的元素
print tuple[2:]           # 输出从第三个开始至列表末尾的所有元素
print tinytuple * 2       # 输出元组两次
print tuple + tinytuple   # 打印组合的元组
```

- **Python 字典**
  - 字典(dictionary)是除列表以外python之中最灵活的内置数据结构类型。列表是有序的对象结合，字典是无序的对象集合。
  - 两者之间的区别在于：字典当中的元素是通过键来存取的，而不是通过偏移存取。
  - 字典用"{ }"标识。字典由索引(key)和它对应的值value组成。

```
#!/usr/bin/python
# -*- coding: UTF-8 -*-

dict = {}
dict['one'] = "This is one"
dict[2] = "This is two"

tinydict = {'name': 'john','code':6734, 'dept': 'sales'}


print dict['one']          # 输出键为'one' 的值
print dict[2]              # 输出键为 2 的值
print tinydict             # 输出完整的字典
print tinydict.keys()      # 输出所有键
print tinydict.values()    # 输出所有值
```

Python数据类型转换

函数	描述
int(x [,base])	将x转换为一个整数
long(x [,base])	将x转换为一个长整数
float(x)	将x转换到一个浮点数
complex(real [,imag])	创建一个复数
str(x)	将对象 x 转换为字符串
repr(x)	将对象 x 转换为表达式字符串
eval(str)	用来计算在字符串中的有效Python表达式,并返回一个对象
tuple(s)	将序列 s 转换为一个元组
list(s)	将序列 s 转换为一个列表
set(s)	转换为可变集合
dict(d)	创建一个字典。d 必须是一个序列 (key,value)元组。
frozenset(s)	转换为不可变集合
chr(x)	将一个整数转换为一个字符
unichr(x)	将一个整数转换为Unicode字符
ord(x)	将一个字符转换为它的整数值
hex(x)	将一个整数转换为一个十六进制字符串
oct(x)	将一个整数转换为一个八进制字符串

2.3 条件和循环

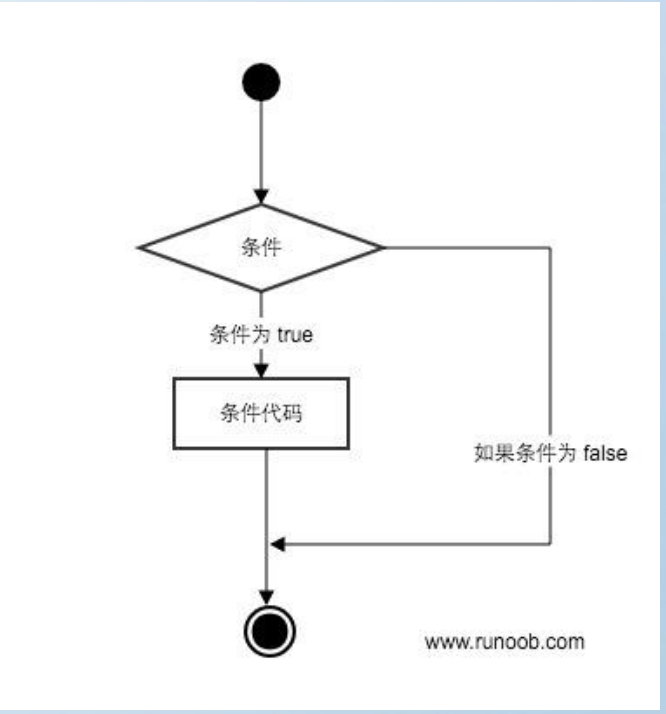
- Python 条件语句

  Python条件语句是通过一条或多条语句的执行结果（True或者False）来决定执行的代码块。

  – Python程序语言指定任何非0和非空（null）值为true，0 或者 null为false。

  – Python 编程中 if 语句用于控制程序的执行，基本形式为

```
if 判断条件：
    执行语句……
else：
    执行语句……
```

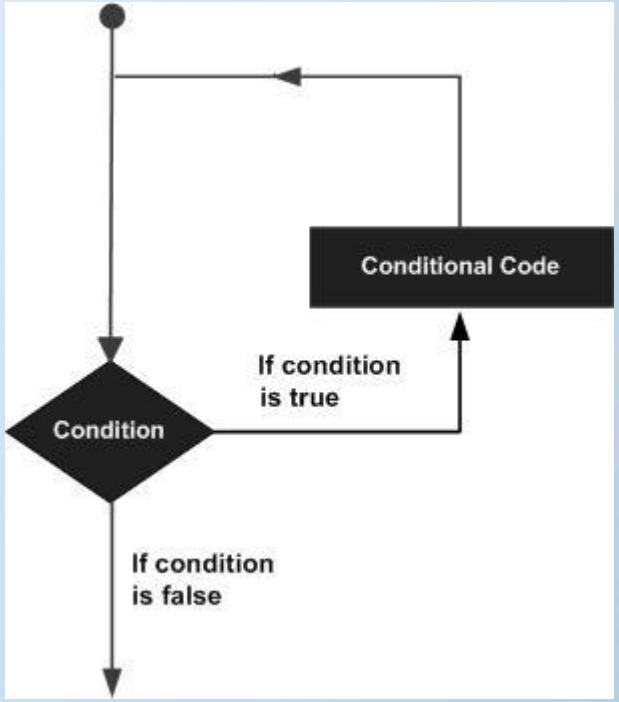


- **Python 循环语句**

  循环语句允许我们执行一个语句或语句组多次。

  – Python提供了for循环和while循环（在Python中没有do..while循环）

循环类型	描述
while 循环	在给定的判断条件为 true 时执行循环体，否则退出循环体。
for 循环	重复执行语句
嵌套循环	你可以在while循环体中嵌套for循环



演示 Python while 语句执行过程

```
1  numbers = [12, 37, 5, 42, 8, 3]
2  even = []
3  odd = []
4  while len(numbers) > 0 :
5      number = numbers.pop()
6      if(number % 2 == 0):
7          even.append(number)
8      else:
9          odd.append(number)
```

2.4 Python 函数

函数是组织好的，可重复使用的，用来实现单一，或相关联功能的代码段。

- 定义函数遵循的相关规则
  - 函数代码块以 **def** 关键词开头，后接函数标识符名称和圆括号**()**。
  - 任何传入参数和自变量必须放在圆括号中间。圆括号之间可以用于定义参数。
  - 函数的第一行语句可以选择性地使用文档字符串—用于存放函数说明。
  - 函数内容以冒号起始，并且缩进。
  - **return [表达式]** 结束函数，选择性地返回一个值给调用方。不带表达式的return相当于返回 None。

- 函数语法

```
def functionname( parameters ):
   "函数_文档字符串"
   function_suite
   return [expression]
```

- 实例

```
#!/usr/bin/python
# -*- coding: UTF-8 -*-

# 定义函数
def printme( str ):
   "打印任何传入的字符串"
   print str;
   return;

# 调用函数
printme("我要调用用户自定义函数!");
printme("再次调用同一函数");
```

2.5 Python 模块

Python 模块(Module)，是一个 Python 文件，以 .py 结尾，包含了 Python 对象定义和Python语句。

- 定义模块好处
  - 模块让你能够有逻辑地组织你的 Python 代码段。
  - 把相关的代码分配到一个模块里能让你的代码更好用，更易懂。
  - 模块能定义函数，类和变量，模块里也能包含可执行的代码。

support.py 模块：

```
def print_func( par ):
   print "Hello : ", par
   return
```

- 模块的引入

模块定义好后，我们可以使用 import 语句来引入模块，语法如下：

```
import module1[, module2[,... moduleN]
```

比如要引用模块 math，就可以在文件最开始的地方用 **import math** 来引入。在调用 math 模块中的函数时，必须这样引用：

```
模块名.函数名
```

test.py 文件代码：

```
#!/usr/bin/python
# -*- coding: UTF-8 -*-

# 导入模块
import support

# 现在可以调用模块里包含的函数了
support.print_func("Runoob")
```

2.6 Python文件I/O

- **打印到屏幕**

  最简单的输出方法是用print语句，你可以给它传递零个或多个用逗号隔开的表达式。此函数把你传递的表达式转换成一个字符串表达式，并将结果写到标准输出如下：

```
#!/usr/bin/python
# -*- coding: UTF-8 -*-

print "Python 是一个非常棒的语言，不是吗？";
```

- **读取键盘输入**

  Python提供了两个内置函数从标准输入读入一行文本，默认的标准输入是键盘。

```
raw_input
input
```

  二者区别

  – raw_input会提示你输入任意字符串，然后在屏幕上显示相同的字符串。
  – **input**函数和 **raw_input**函数基本类似，但是 input 可以接收一个Python表达式作为输入，并将运算结果返回。

打开和关闭文件

Python 提供了必要的函数和方法进行默认情况下的文件基本操作。你可以用 **file** 对象做大部分的文件操作。

- **open 函数**

  你必须先用Python内置的open()函数打开一个文件，创建一个file对象，相关的方法才可以调用它进行读写。 语法：

```
file object = open(file_name [, access_mode][, buffering])
```

  - file_name：file_name变量是一个包含了你要访问的文件名称的字符串值。
  - buffering:如果buffering的值被设为0，就不会有寄存。如果buffering的值取1，访问文件时会寄存行。如果将buffering的值设为大于1的整数，表明了这就是的寄存区的缓冲大小。如果取负值，寄存区的缓冲大小则为系统默认。
  - access_mode：access_mode决定了打开文件的模式：只读，写入，追加等。所有可取值见如下的完全列表。这个参数是非强制的，默认文件访问模式为只读(r)。

- **close()方法**

File 对象的 close（ ）方法刷新缓冲区里任何还没写入的信息，并关闭该文件，这之后便不能再进行写入。当一个文件对象的引用被重新指定给另一个文件时，Python 会关闭之前的文件。语法：

```
fileObject.close();
```

```
#!/usr/bin/python
# -*- coding: UTF-8 -*-

# 打开一个文件
fo = open("foo.txt", "wb")
print "文件名: ", fo.name

# 关闭打开的文件
fo.close()
```

- **write()方法**

  write()方法可将任何字符串写入一个打开的文件。需要重点注意的是，Python字符串可以是二进制数据，而不是仅仅是文字。write()方法不会在字符串的结尾添加换行符('\n')。语法：

```
fileObject.write(string);
```

```
#!/usr/bin/python
# -*- coding: UTF-8 -*-

# 打开一个文件
fo = open("foo.txt", "wb")
fo.write( "用write函数写入文件");

# 关闭打开的文件
fo.close()
```

- **read()方法**

read（ ）方法从一个打开的文件中读取一个字符串。需要重点注意的是，Python字符串可以是二进制数据，而不是仅仅是文字。语法：

```
fileObject.read([count]);
```

```
#!/usr/bin/python
# -*- coding: UTF-8 -*-

# 打开一个文件
fo = open("foo.txt", "r+")
str = fo.read(10);
print "读取的字符串是 : ", str
# 关闭打开的文件
fo.close()
```

Python File(文件) 方法

序号	方法及描述
1	file.close()　关闭文件。关闭后文件不能再进行读写操作。
2	file.flush()　刷新文件内部缓冲，直接把内部缓冲区的数据立刻写入文件, 而不是被动的等待输出缓冲区写入。
3	file.fileno()　返回一个整型的文件描述符(file descriptor FD 整型), 可以用在如os模块的read方法等一些底层操作上。
4	file.isatty()　如果文件连接到一个终端设备返回 True，否则返回 False。
5	file.next()　返回文件下一行。
6	file.read([size])　从文件读取指定的字节数，如果未给定或为负则读取所有。
7	file.readline([size])　读取整行，包括 "\n" 字符。
8	file.readlines([sizehint])　读取所有行并返回列表，若给定sizeint>0，返回总和大约为sizeint字节的行, 实际读取值可能比sizhint较大, 因为需要填充缓冲区。
9	file.seek(offset[, whence])　设置文件当前位置
10	file.tell()　返回文件当前位置。
11	file.truncate([size])　截取文件，截取的字节通过size指定，默认为当前文件位置。
12	file.write(str)　将字符串写入文件，没有返回值。
13	file.writelines(sequence)　向文件写入一个序列字符串列表，如果需要换行则要自己加入每行的换行符。