

Python语言及数据分析

计算机系 王学军

个人邮箱: wangxuejun@stdu.edu

教学反馈: sist_student@163.com

22-02

引言



PHP之父：拉斯马斯·勒德尔夫



Java之父：詹姆斯·高斯林



C++之父：本贾尼·施特劳斯特卢普



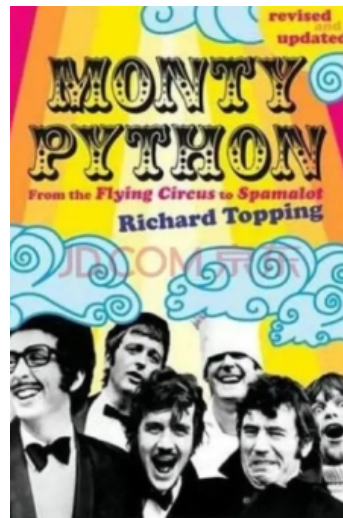
C语言之父：丹尼斯·里奇



python之父：吉多·范罗苏姆

引言

- 由荷兰人吉多·范罗苏姆(Guido van Rossum)于1989年发明，作为ABC的替代语言。据说是龟数为了打发无聊的圣诞节，创造了python。Python从ABC发展起来，而且结合了Unix shell和C的习惯。Python是面向对象编程，是一种解释型语言，具有很好的跨平台性。
- Python从一出生就具有了：类，函数，异常处理，包含表和词典在内的核心数据类型，以及模块为基础的拓展系统。



Monty Python' Flying Cricus

蒙提·派森的飞行马戏团

引言

❑ Python官网最新发行版本是python 3.10.2 21.10.4发布 (<https://www.python.org/getit/>), python 2.0版本已经停止更新, 学习python3即可。

❑ 整体特点:

1.跨平台编程语言

2.面向对象语言
















3.交互式语言, 命令提示符>>>之后直接运行代码

4.解释型语言, 不需要编译, 但消耗更多内存

5.对小白很友好的编程语言, 简洁易懂, 而且功能强大

引言

python语言趋势 (<https://www.tiobe.com/tiobe-index/>)

Feb 2022	Feb 2021	Change	Programming Language		Ratings	Change
1	3	▲		Python	15.33%	+4.47%
2	1	▼		C	14.08%	-2.26%
3	2	▼		Java	12.13%	+0.84%
4	4			C++	8.01%	+1.13%
5	5			C#	5.37%	+0.93%
6	6			Visual Basic	5.23%	+0.90%
7	7			JavaScript	1.83%	-0.45%
8	8			PHP	1.79%	+0.04%
9	10	▲		Assembly language	1.60%	-0.06%
10	9	▼		SQL	1.55%	-0.18%
11	13	▲		Go	1.23%	-0.05%
12	15	▲		Swift	1.18%	+0.04%
13	11	▼		R	1.11%	-0.45%
14	16	▲		MATLAB	1.03%	-0.03%
15	17	▲		Delphi/Object Pascal	0.90%	-0.12%

需求量大，薪资可观

引言

学时安排（32学时，理论实验各16学时）

理论教学内容（16学时）	学时
1 python语言基础	2
2 程序控制结构	2
3 函数	2
4 正则表达式	2
5 文件与文件夹操作	2
6 用matplotlib实现数据可视化	2
7.Numpy、pandas库的使用方法	4

引言

实 验 教 学 内 容（16学时）	学时
1 数据质量分析实验	2
2 数据预处理实验	2
3 常用数据分析方法实验	2
4 基于信用卡消费行为的客户违约分析实验	2
5 文本情感分析实验	2
6 数字手写体识别实验	4
7 网络爬虫实验	2

引言

考试
平时
完成
详见



参考书：《Python数据可视化之美》 张杰 著
《Python数据分析从入门到实践》 高春艳 著

第1章 Python语言基础

- 1.1 Python应用领域
- 1.2 Python语言特点
- 1.3 编写Python代码
- 1.4 Python中的对象
- 1.5 Python中的变量
- 1.6 Python中的基本数据类型
- 1.7 Python中的运算符
- 1.8 Python中的数据输入
- 1.9 Python中的数据输出
- 1.10 Python库的导入与扩展库的安装

第1章 Python语言基础

- ✓ 1.1 Python应用领域
- 1.2 Python语言特点
- 1.3 编写Python代码
- 1.4 Python中的对象
- 1.5 Python中的变量
- 1.6 Python中的基本数据类型
- 1.7 Python中的运算符
- 1.8 Python中的数据输入
- 1.9 Python中的数据输出
- 1.10 Python库的导入与扩展库的安装

1.1 Python应用领域

Python被广泛应用于众多领域，例如：

➤ web开发

Python拥有很多免费数据函数库、免费web网页模板系统、以及与web服务器进行交互的库，可以实现web开发，搭建web框架，目前比较有名的Python web框架为Django。

➤ 爬虫开发

在爬虫领域，Python几乎是霸主地位，将网络一切数据作为资源，通过自动化程序进行有针对性的数据采集以及处理。

➤ 云计算开发

Python是从事云计算工作需要掌握的一门编程语言，目前很火的云计算框架OpenStack就是由Python开发的。

1.1 Python应用领域

➤ 人工智能

MASA和Google早期大量使用Python，为Python积累了丰富的科学运算库，当AI时代来临后，目前市面上大部分的人工智能的代码都是使用Python来编写，尤其PyTorch之后，Python作为AI时代首先语言的位置基本确定。

➤ 自动化运维

Python是一门综合性的语言，能满足绝大部分自动化运维需求，前端和后端都可以做。

➤ 数据分析

Python已成为数据分析和数据科学事实上的标准语言 and 标准平台之一，Numpy、Pandas、Scipy和Matplotlib程序库共同构成了Python数据分析的基础。

1.1 Python应用领域

➤ 科学计算

随着NumPy、SciPy、Matplotlib、Enthought librarys等众多程序库的开发，使得Python越来越适合做科学计算、绘制高质量的2D和3D图像。

第1章 Python语言基础

- ☐ 1.1 Python应用领域
- ✓ 1.2 Python语言特点
- ☐ 1.3 编写Python代码
- ☐ 1.4 Python中的对象
- ☐ 1.5 Python中的变量
- ☐ 1.6 Python中的基本数据类型
- ☐ 1.7 Python中的运算符
- ☐ 1.8 Python中的数据输入
- ☐ 1.9 Python中的数据输出
- ☐ 1.10 Python库的导入与扩展库的安装

1.2 Python语言特点

- ❑ 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，具有丰富和强大的库。
- ❑ Python常被昵称为胶水语言，能够把用其他语言制作的模块（尤其是C/C++）很轻松地联结在一起。
- ❑ Python强制用空白符（white space）作为语句缩进。

Python标准库很庞大，可用来处理正则表达式、文档生成、单元测试、线程、数据库、网页浏览器、CGI、FTP、电子邮件、XML、XML-RPC、HTML、WAV文件、密码系统、GUI（图形用户界面）、其它与系统有关的操作。

第1章 Python语言基础

- ☐ 1.1 Python应用领域
- ☐ 1.2 Python语言特点
- ✓ 1.3 编写Python代码
- ☐ 1.4 Python中的对象
- ☐ 1.5 Python中的变量
- ☐ 1.6 Python中的基本数据类型
- ☐ 1.7 Python中的运算符
- ☐ 1.8 Python中的数据输入
- ☐ 1.9 Python中的数据输出
- ☐ 1.10 Python库的导入与扩展库的安装

1.3 编写Python代码

默认编程环境: **IDLE**

其他常用开发环境:

◆ **Eclipse+PyDev**

◆ **pyCharm+Anaconda 推荐**

◆ **zwPython**

◆ **Spyder**

https://zhuanlan.zhihu.com/p/36389880?ivk_sa=1024320u

python代码可视化平台 <https://pythontutor.com>

1.3 编写Python代码

- 在IDLE中，如果使用交互式编程模式，那么直接在提示符“>>>”后面输入相应的命令并回车执行即可，如果执行顺利的话，马上就可以看到执行结果，否则会抛出异常。

```
>>> 4+5
```

```
9
```

```
>>> 3/0
```

```
Traceback (most recent call last):
```

```
File "<pyshell#18>", line 1, in <module>
```

```
3/0
```

```
ZeroDivisionError: integer division or modulo by zero
```

1.3.4 用带图形界面的Python Shell编写程序代码

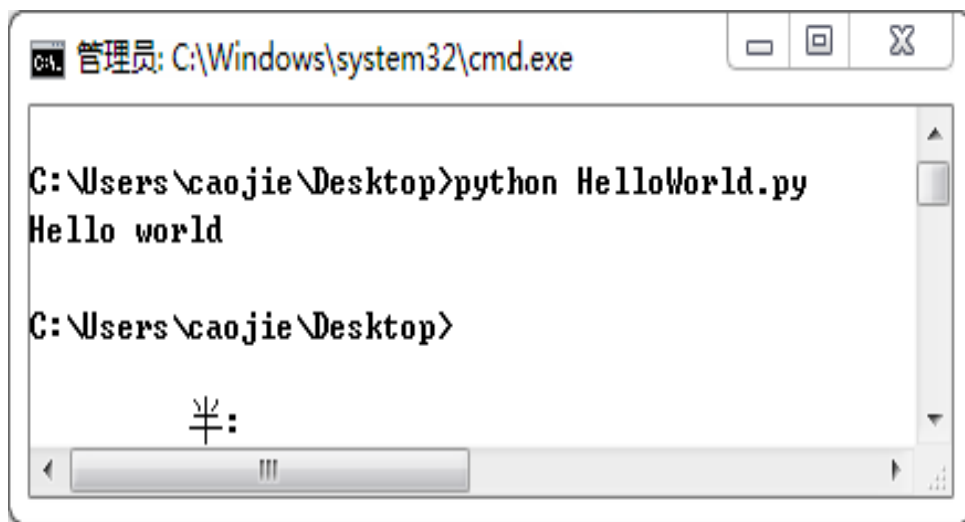
- 当需要编写大量Python代码行时，就需要通过编写程序（也叫脚本）来避免繁琐。
- 在IDLE中编写、运行程序的步骤：
 - ◆ （1）启动IDLE；
 - ◆ （2）选择菜单File > New File创建一个程序文件，输入代码并保存为扩展名为.py的文件。
 - ◆ （3）选择菜单Run > Run Module F5运行程序，程序运行结果将直接显示在IDLE交互界面上。

1.3.4 编写Python代码

- 在有些情况下可能需要在命令提示符环境中运行Python程序文件。在“开始”菜单的“附件”中单击“命令提示符”，然后执行Python程序。假设有程序HelloWorld.py内容如下：

```
print('Hello world')
```

如果HelloWorld.py不在当前路径下，需要在前面写上路径名，如
C:\Users\caojie\Desktop\HelloWorld.py



1.3 编写Python代码

- 如果能够熟练使用开发环境提供的一些快捷键，将会大幅度提高开发效率，在IDLE中一些比较常用的快捷键如下表所示：

增加代码块缩进 ↵	Ctrl+] ↵	+
减少代码块缩进 ↵	Ctrl+[↵	+
注释代码块 ↵	Alt+3 ↵	+
取消代码块注释↵	Alt+4 ↵	+
浏览上一条输入的命令 ↵	Alt+p ↵	+
浏览下一条输入的命令↵	Alt+n ↵	+
Enter↵	鼠标选中后进行复制↵	+
Tab↵	补全单词，列出全部可选单词供选择↵	+

第1章 Python语言基础

- ☐ 1.1 Python应用领域
- ☐ 1.2 Python语言特点
- ☐ 1.3 编写Python代码
- ✓ 1.4 Python中的对象
- ☐ 1.5 Python中的变量
- ☐ 1.6 Python中的基本数据类型
- ☐ 1.7 Python中的运算符
- ☐ 1.8 Python中的数据输入
- ☐ 1.9 Python中的数据输出
- ☐ 1.10 Python库的导入与扩展库的安装

第1章 Python语言基础

- ☐ 1.1 Python应用领域
- ☐ 1.2 Python语言特点
- ☐ 1.3 编写Python代码
- ✓ 1.4 Python中的对象
- ☐ 1.5 Python中的变量
- ☐ 1.6 Python中的基本数据类型
- ☐ 1.7 Python中的运算符
- ☐ 1.8 Python中的数据输入
- ☐ 1.9 Python中的数据输出
- ☐ 1.10 Python库的导入与扩展库的安装

1.4 Python中的对象

- ❑ Python中的对象就是编程中把数据和功能包装后形成的一个对外具有特定交互接口的内存块。
 - ❑ 每个对象都有三个属性分别是：
 - 身份(identity)，就是对象在内存中的地址；
 - 类型(type)，用于表示对象所属的数据类型(类)，对象的类型决定了对象可以存储什么类型的值，有哪些属性和方法，可以进行哪些操作；
 - 值(value)，对象所表示的数据。
- ```
>>> a=123 #123创建了一个int(整型)对象，并用a来代表
>>> id(a)
492688880 #身份用这样一串数字表示
```

# 1.4 Python中的对象

```
>>> b=6
```

简单来看，上边的代码执行了以下操作：

- （1）使用变量b来代表对象6。为了使用对象，必须通过赋值操作“=”把对象赋值给一个变量（也称之为把对象绑定到变量），这样便可通过该变量来操作内存数据块中的数据。
- （2）如果变量b不存在，创建一个新的变量b。
- （3）将变量b和数字6进行连接，即变量b成为对象6的一个引用，变量可看作是指向对象的内存空间的一个指针。

# 1.4 Python中的对象

□ 当多个变量都引用了相同的对象，称为共享引用。

```
>>> a=1
```

```
>>> b=a #b成为1的引用
```

```
>>> a=2 #a成为对象2的一个引用
```

```
>>> print(b)
```

```
1
```

#由于变量仅是对对象的一个引用，因此改变a的引用并不会导致b的变化

# 第1章 Python语言基础

- ☐ 1.1 Python应用领域
- ☐ 1.2 Python语言特点
- ☐ 1.3 编写Python代码
- ☐ 1.4 Python中的对象
- ✓ 1.5 Python中的变量
- ☐ 1.6 Python中的基本数据类型
- ☐ 1.7 Python中的运算符
- ☐ 1.8 Python中的数据输入
- ☐ 1.9 Python中的数据输出
- ☐ 1.10 Python库的导入与扩展库的安装

# 1.5 Python中的变量

□ 在Python中，变量是用一个变量名表示，变量名的命名规则：

- 变量名只能是字母、数字或下划线的任意组合。
- 变量名的第一个字符不能是数字。
- 以下Python关键字不能声明为变量名：

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

# 1.5 Python中的变量

□ `>>> x='Python'`

上述代码创建了一个变量x，x是字符串对象'Python'的引用，即变量x指向的对象的值为'Python'。

注意：类型属于对象，变量是没有类型的，变量只是对象的引用，所谓变量的类型指的是变量所引用的对象的类型。变量的类型随着所赋值的类型的变化而改变。



# 第1章 Python语言基础

- ❑ 1.1 Python应用领域
- ❑ 1.2 Python语言特点
- ❑ 1.3 编写Python代码
- ❑ 1.4 Python中的对象
- ❑ 1.5 Python中的变量
- ✓ 1.6 Python中的基本数据类型
- ❑ 1.7 Python中的运算符
- ❑ 1.8 Python中的数据输入
- ❑ 1.9 Python中的数据输出
- ❑ 1.10 Python库的导入与扩展库的安装

# 1.6 Python中的基本数据类型

- Python语言中，所有对象都有一个数据类型，Python数据类型定义为一个值的集合以及定义在这个值集上的一组运算操作。一个对象上可执行且只允许执行其对应数据类型定义的操作。
- Python中有六个标准的数据类型：
  - number（数字）
  - string（字符串）
  - list（列表）
  - tuple（元组）
  - dictionary（字典）
  - set（集合）

## 1.6.1 number（数字）

□ Python包括4种内置的数字数据类型。

- (1) **int**整型。用于表示整数，如12，1024，10。
- (2) **float**浮点型。用于表示实数，如3.14，1.2，2.5e2  
( $= 2.5 \times 10^2 = 250$ )。
- (3) **bool**布尔型。**bool**布尔型对应两个布尔值：**True**和**False**，分别对应1和0。

```
>>> True+1
```

```
2
```

- (4) **complex**复数型。其表示由两种，一种是 $a+bj$ ，另一种是`complex(a,b)`，例如 $3 + 4j \longleftrightarrow \text{complex}(3,4)$ 。

## 1.6.2 string（字符串）

□ Python中的字符串属于不可变序列，是用单引号(')、双引号(")、三单引号(''')或三双引号('\"'')等界定符括起来的字符序列。

### ➤ （1）创建字符串

只要为变量分配一个用字符串界定符括起来的字符序列即可创建一个字符串。例如：

```
var1 = 'Hello World!'
```

### ➤ （2）Python 3的字符串编码

我们可以通过以下代码查看Python 3的字符串默认编码：

```
>>> import sys
>>> sys.getdefaultencoding()
'utf-8'
```

## 1.6.2 string（字符串）

### ➤ （3）字符串运算符

| 操作符    | 描述                           |
|--------|------------------------------|
| +      | 字符串连接                        |
| *      | 重复输出字符串                      |
| []     | 通过索引获取字符串中字符                 |
| [ : ]  | 截取字符串中的一部分                   |
| in     | 成员运算符--如果字符串中包含给定的字符串返回True  |
| not in | 成员运算符--如果字符串中不包含给定的字符串返回True |
| %      | 格式化字符串                       |

## 1.6.2 string（字符串）

### ➤（3）字符串运算符

```
>>> str1= 'Python'
```

```
>>> str2= ' good'
```

```
>>> str3=str1+str2 #字符串连接
```

```
>>> print(str3)
```

Python good

```
>>> print (str1 * 2) # 输出字符串两次
```

PythonPython

```
>>> print(2* str1)
```

PythonPython

## 1.6.2 string（字符串）

### ➤（3）字符串运算符

Python中的字符串有两种索引方式，从左往右以0开始，从右往左以-1开始。

```
>>> print (str1[0]) # 通过索引输出字符串第一个字符
```

**P**

```
>>> print (str1[2:5]) # 输出从第三个开始到第五个的字符
```

**tho**

```
>>> print (str1[0:-1]) # 输出第一个到倒数第二个的所有字符
```

**Pytho**



## 1.6.2 string（字符串）

### ➤ （3）字符串运算符

```
>>> 'y' in str1 #测试一个字符串是否存在另一个字符串
中
```

```
True
```

```
>>> 'ac' in 'abcd'
```

```
False
```

```
>>> 'ac' not in 'abcd'
```

```
True
```

## 1.6.2 string（字符串）

### ➤（4）字符串对象常用的方法

一旦创建字符串对象str，可以使用字符串对象str的方法来操作字符串。

① `str.strip([chars])`: 不带参数的`str.strip()`方法，表示去除字符串str开头和结尾的空白符，包括：“\n(换行)”，“\t（制表符）”，“\r”，“ ”等；带参数的`str.strip(chars)`函数，表示去除字符串str开头和结尾指定的chars字符序列，只要有就删除。

```
>>> b, c = '\t\ns\tpython\n', '16\t\ns\tpython\n16'
```

```
>>> b.strip()
```

```
's\tpython'
```

```
>>> c.strip('16')
```

```
'\t\ns\tpython\n'
```

## 1.6.2 string（字符串）

### ➤（5）字符串对象常用的方法

#### ② 字符串大小写转换

**str.lower():** 将字符串str中的大写字母转小写字母。

```
>>> 'ABba'.lower()
```

```
'abba'
```

**str.upper():** 将str中的小写字母转成大写字母。

```
>>> 'ABba'.upper()
```

```
'ABBA'
```

**str.swapcase():** 将str中的大小写互换。

```
>>> 'ABba'.swapcase()
```

```
'abBA'
```

## 1.6.2 string（字符串）

### ➤（5）字符串对象常用的方法

#### ② 字符串大小写转换

`str.capitalize()`: 返回一个只有首字母大写的字符串

```
>>> 'ABba'.capitalize()
```

```
'Abba'
```

`string.capwords(str[, sep])`: 以`sep`作为分隔符(不带参数`sep`时, 默认以空格为分隔符), 分割字符串`str`, 然后将每个字段的首字母换成大写, 将每个字段除首字母外的字母均置为小写, 最后合并连接到一起组成一个新字符串。 **`capwords(str)`是 `string` 模块中的函数, 使用之前需要先导入 `string` 模块, 即 `import string`。**

## 1.6.2 string（字符串）

### ➤（5）字符串对象常用的方法

#### ② 字符串大小写转换

`string.capwords(str[, sep])`: 以`sep`作为分隔符(不带参数`sep`时, 默认以空格为分隔符), 分割字符串`str`, 然后将每个字段的首字母换成大写, 将每个字段除首字母外的字母均置为小写, 最后合并连接到一起组成一个新字符串。**`capwords(str)`是`string`模块中的函数, 使用之前需要先导入`string`模块, 即**`import string`**。**

```
>>> import string
```

```
>>> string.capwords("ShaRP tools make good work.")
'Sharp Tools Make Good Work.'
```

## 1.6.2 string（字符串）

### ➤（5）字符串对象常用的方法

#### ③ 字符串分割

`str.split(s,num) [n]`: 按s中指定的分隔符(默认为所有的空字符, 包括空格、换行(\n)、制表符(\t)等), 将一个字符串str分裂成num+1个字符串所组成的列表。若带有[n], 表示选取分割后的第n个分片, n表示返回的列表中元素的下标, 从0开始。

```
>>> str='hello world'
>>> str.split()
['hello', 'world']
```

```
>>> str.split('.',1) #分隔一次
['www', 'baidu.com']
>>> str = 'www.baidu.com'
>>> str.split('.')[2]
#选取分割后的第2片作为结果返回
'com'
```

## 1.6.2 string（字符串）

### ➤（5）字符串对象常用的方法

#### ③ 字符串分割

`>>> s1, s2, s3=str.split('.', 2)`     `#s1, s2, s3分别被赋值得到`  
被切割的三个部分

`>>> s1`

`'www'`

`>>> s = 'call\nme\nbaby'`     `#按换行符“\n”进行分割`

`>>> s.split('\n')`

`['call', 'me', 'baby']`

## 1.6.2 string（字符串）

### ➤（5）字符串对象常用的方法

#### ④ 字符串搜索与替换

`str.find(substr [,start [, end ] ])`: 返回`str`中指定范围(默认是整个字符串)第一次出现的`substr`的第一个字母的标号, 也就是说从左边算起的第一次出现的`substr`的首字母标号, 如果`str`中没有`substr`则返回-1。

```
>>> 'He that can have patience. '.find('can')
```

8

`str.replace(oldstr, newstr [, count])`: 把`str`中的`oldstr`字符串替换成`newstr`字符串, 如果指定了`count`参数, 表示替换最多不超过`count`次。如果未指定`count`参数, 表示全部替换, 有多少替换多少。



## 1.6.2 string（字符串）

### ➤（5）字符串对象常用的方法

#### ④ 字符串搜索与替换

```
>>> str.replace(" is", " was")
```

```
'This was a string example.This was a really string.'
```

`str.count(substr[, start, [end] ])`：在字符串`str`中统计子字符串`substr`出现的次数，如果不指定开始位置`start`和结束位置`end`，表示从头统计到尾。

```
>>> 'aadgdxdfadfaadfgaa'.count('aa')
```

```
3
```

## 1.6.2 string（字符串）

### ➤（5）字符串对象常用的方法

#### ⑤ 字符串映射

**str.maketrans(instr, outstr):** 用于创建字符映射的转换表（映射表），第一个参数instr表示需要转换的字符串，第二个参数outstr表示要转换的目标字符串。两个字符串的长度必须相同，为一一对应的关系。

**str.translate(table):** 使用str.maketrans(instr, outstr)生成的映射表table，对字符串str进行映射。

## 1.6.2 string（字符串）

### ➤（5）字符串对象常用的方法

#### ⑤ 字符串映射

```
>>> table=str.maketrans('abcdef','123456')
```

#创建映射表，将'abcdef'一一转换为'123456'

```
>>> s1='Python is a greate programming language.I like it.'
```

```
>>> s1.translate(table) #使用映射表table对字符串s1进行映射
```

```
'Python is 1 gr51t5 progr1mming l1ngu1g5.I lik5 it.'
```

## 1.6.2 string（字符串）

### ➤（5）字符串对象常用的方法

#### ⑥ 判断字符串的开始和结束

**str.startswith(substr[, start, [end]])**: 用于检查字符串str是否是以字符串substr开头，如果是则返回True，否则返回False。

```
>>> s='Work makes the workman.'
```

```
>>> s.startswith('Work') #检查整个字符串
```

```
True
```

**str.endswith(substr[, start[, end]])**: 用于检查字符串str是否是以字符串substr结尾，如果是则返回True，否则返回 False。

```
>>> s='Constant dropping wears the stone.'
```

```
>>> s.endswith('stone.')
```

```
True
```

## 1.6.2 string（字符串）

➤（5）字符串对象常用的方法

### ⑥ 判断字符串的开始和结束

下面的代码可以列出指定目录下扩展名为.txt或.docx的文件

```
import os
```

```
items = os.listdir("C:\\Users\\caojie\\Desktop") #返回指定路径
下的文件和文件夹列表
```

```
newlist = []
```

```
for names in items:
```

```
 if names.endswith((".txt", ".docx")):
```

```
 newlist.append(names)
```

```
print(newlist)
```

## 1.6.2 string（字符串）

### ➤（5）字符串对象常用的方法

#### ⑦ 连接字符串

**str.join(sequence):** 返回通过指定字符str连接序列sequence中元素后生成的新字符串。

```
>>> str = "-"
```

```
>>> seq = ('a', 'b', 'c', 'd')
```

```
>>> str.join(seq)
```

```
'a-b-c-d'
```

```
>>> seq1 = ['Keep','on','going','never','give','up']
```

```
>>> print(' '.join(seq1))
```

```
Keep on going never give up
```

## 1.6.2 string（字符串）

### ➤（5）字符串对象常用的方法

#### ⑦ 连接字符串

**str.join(sequence):** 返回通过指定字符str连接序列sequence中元素后生成的新字符串。

```
>>> str = "-"
```

```
>>> seq = ('a', 'b', 'c', 'd')
```

```
>>> str.join(seq)
```

```
'a-b-c-d'
```

```
>>> seq1 = ['Keep','on','going','never','give','up']
```

```
>>> print(' '.join(seq1))
```

```
Keep on going never give up
```

## 1.6.2 string（字符串）

### ➤（5）字符串对象常用的方法

#### ⑧ 判断字符串是否全为数字、字符等

**str.isalnum():** str所有字符都是数字或者字母，返回 Ture

**str.isalpha():** str所有字符都是字母，返回 Ture

**str.isdigit():** str所有字符都是数字，返回 Ture

**str.islower():** str所有字符都是小写，返回 Ture

**str.isupper():** str所有字符都是大写，返回 Ture

**str.istitle():** str所有单词都是首字母大写，返回 Ture

**str.isspace():** str所有字符都是空白字符，返回 Ture



## 1.6.2 string（字符串）

### ➤（5）字符串对象常用的方法

#### ⑨ 字符串对齐及填充

**str.center(width[, fillchar]):** 返回一个宽度为width、str居中的新字符串，如果width小于字符串str的宽度，则直接返回字符串str，否则使用填充字符fillchar去填充，默认填充空格。

```
>>> 'Hello world!'.center(20)
```

```
' Hello world! '
```

```
>>> 'Hello world!'.center(20,'-')
```

```
'----Hello world!----'
```

## 1.6.2 string（字符串）

### ➤ （6）字符串常量

Python标准库string中定义了数字、标点符号、英文字母、大写英文字母、小写英文字母等字符串常量。

```
>>> import string
```

```
>>> string.ascii_letters #所有字母
```

```
'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ
WXYZ'
```

```
>>> string.ascii_lowercase #所有小写字母
```

```
'abcdefghijklmnopqrstuvwxyz'
```

```
>>> string.ascii_uppercase #所有大写字母
```

```
'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
```

```
>>> string.digits #所有数字0-9
```

```
'0123456789'
```

## 1.6.3 list（列表）

- ❑ 列表是写在方括号[]之间、用逗号分隔开的元素列表。列表的大小是可变的，它可以根据需求增加或减少。列表中元素的类型可以不相同，列表中可以同时存在数字、字符串、元组、字典、集合等数据类型的对象，甚至可以包含列表（即嵌套）。

下面几个都是合法的列表对象：

- ['Google', 'Baidu', 1997, 2008]
- [1, 2, 3, 4, 5]
- ["a", "b", "c", "d"]
- [123, ["das", "aaa"], 234]

## 1.6.3 list（列表）

### （1）列表创建、删除

可以使用列表list的构造方法list()来创建列表，如下所示：

```
>>> list1 = list() #创建空列表
```

```
>>> list2 = list('chemistry')
```

```
>>> list2
```

```
['c', 'h', 'e', 'm', 'i', 's', 't', 'r', 'y']
```

也可以使用“=”直接将一个列表赋值给变量来创建一个列表对象：

```
>>> lista= []
```

```
>>> listb = ['good', 123 , 2.2, 'best', 70.2]
```

## 1.6.3 list（列表）

### （2）列表截取（也称分片、切片）

- 列表中的元素可以使用下标操作符`list[index]`访问列表中下标为`index`的元素。
- 列表下表是从0开始的，下标的范围从0到`len(list)-1`，`len(list)`获取列表`list`的长度。
- `list[index]`可以像变量一样使用，被称为下标变量。
- Python允许使用负数作为下标来引用相对于列表末端的位置，将列表长度和负数下标相加就可以得到实际的位置。例如：

```
>>> list1=[1,2,3,4,5]
```

```
>>> list1[-1]
```

```
5
```

## 1.6.3 list（列表）

### （2）列表截取（也称分片、切片）

- 列表截取（也称分片、切片）操作使用语法`list[start:end]`返回列表的一个片段。这个片段是下标从`start`到`end-1`的元素构成的一个子列表。列表被截取后返回一个包含指定元素的新列表。

```
>>> list1 = ['good', 123 , 2.2, 'best', 70.2]
```

```
>>> print (list1[1:3]) # 输出第二个至第三个元素
```

```
[123, 2.2]
```

## 1.6.3 list（列表）

### （3）改变列表

#### ➤ 列表元素改变：

```
>>> x = [1,1,3,4]
```

```
>>> x[1] = 2 #将列表中第二个1改为2
```

#### ➤ 列表元素分段改变：

```
>>> name = list('Perl')
```

```
>>> name[1:] = list('ython')
```

```
>>> name
```

```
['P', 'y', 't', 'h', 'o', 'n']
```

## 1.6.3 list（列表）

### （3）改变列表

- 在列表中插入序列：

```
>>> number=[1,6]
>>> number[1:1]=[2,3,4,5]
>>> number
[1, 2, 3, 4, 5, 6]
```

- 在列表中删除一段元素：

```
>>> names = ['one', 'two', 'three', 'four', 'five', 'six']
>>> del names[1] #删除names的第二个元素
>>> names[1:4]=[] #删除names的第二至第四个元素
>>> names
['one', 'five', 'six']
```



## 1.6.3 list（列表）

### （4）列表是一种序列类型

在Python中字符串、列表和以及后面要讲的元组都是序列类型。

- 所谓序列，即成员有序排列，并且可以通过偏移量访问到它的一个或者几个成员。
- 序列中的每个元素都被分配一个数字——它的位置，也称为索引，第一个索引是0，第二个索引是1，依此类推。
- 序列都可以进行的操作包括索引、切片、加、乘、检查成员。
- Python已经内置确定序列的长度以及确定最大和最小的元素的方法。

## 1.6.3 list（列表）

（4）列表是一种序列类型，序列的常用操作：

| 操作                                            | 描述                  |
|-----------------------------------------------|---------------------|
| <code>x in s</code>                           | 如果元素x在序列s中则返回True   |
| <code>x not in s</code>                       | 如果元素x不在序列s中则返回True  |
| <code>s1+s2</code>                            | 连接两个序列s1和s2，得到一个新序列 |
| <code>s*n, n*s</code>                         | 序列s复制n次得到一个新序列      |
| <code>s[i]</code>                             | 得到序列s的第i个元素         |
| <code>s[i:j]</code>                           | 得到序列s从下标i到j-1的片段    |
| <code>len(s)</code>                           | 返回序列s包含的元素个数        |
| <code>max(s)</code>                           | 返回序列s的最大元素          |
| <code>min(s)</code>                           | 返回序列s的最小元素          |
| <code>sum(x)</code>                           | 返回序列s中所有元素之和        |
| <code>&lt;, &lt;=, &gt;, &gt;=, ==, !=</code> | 比较两个序列              |

## 1.6.3 list（列表）

（4）列表是一种序列类型

```
>>> list1=['C', 'Java', 'Python']
```

```
>>> list2=['good', 123 , 2.2, 'best', 70.2]
```

```
>>> list1+list2
```

```
['C', 'Java', 'Python', 'good', 123, 2.2, 'best', 70.2]
```

```
>>> max(list1)
```

```
'Python'
```

```
>>> min(list1)
```

```
'C'
```

```
>>> sum([1,2,3])
```

```
6
```

## 1.6.3 list（列表）

（5）用于列表的一些常用函数

① **reversed()**函数： 函数功能是反转一个序列对象，将其元素从后向前颠倒构建成一个迭代器。

```
>>> a=[9, 8, 7, 6, 5, 4, 3, 2, 1, 0] print(list(b))
```

```
>>> reversed(a) 显示为迭代器对象的内存地址
```

```
<list_reverseiterator object at 0x0000000002F174E0>
```

② **sorted()**函数： **sorted(iterable[, key][, reverse])**返回一个排序后的新序列，不改变原始的序列。**sorted**的第一个参数是一个可迭代的对象，第二个参数**key**用来指定带一个参数的函数，此函数将在每个元素比较前被调用；第三个参数用来指定排序方式（正序还是倒序）。

## 1.6.3 list（列表）

### （5）用于列表的一些常用函数

`>>> sorted([46, 15, -12, 9, -21, 30], key=abs)` #按绝对值大小进行排序

`[9, -12, 15, -21, 30, 46]`

`key`指定的函数将作用于`list`的每一个元素上，并根据`key`函数返回的结果进行排序。

`>>> sorted(['bob', 'about', 'Zoo', 'Credit'], key=str.lower, reverse=True)` #按小写反向排序

`['Zoo', 'Credit', 'bob', 'about']`

## 1.6.3 list（列表）

### （5）用于列表的一些常用函数

③ `zip()`函数：`zip([it0,it1...])`返回一个列表，其第一个元素是 `it0`、`it1...`这些序列元素的第一个元素组成的一个元组，其它元素依次类推。若传入参数的长度不等，则返回列表的长度和参数中长度最短的对象相同。`zip()`是可迭代对象，对其进行`list`可一次性显示出所有结果。

```
>>> a, b = [1,2,3], ['a','b','c']
```

```
>>> list(zip(a,b))
```

```
[(1, 'a'), (2, 'b'), (3, 'c')]
```

## 1.6.3 list（列表）

（5）用于列表的一些常用函数

④**enumerate()函数**：将一个可遍历的数据对象如列表，组合为一个索引序列，序列中每个元素是由数据对象的元素下标和元素组成的元组。

```
>>> seasons = ['Spring', 'Summer', 'Fall', 'Winter']
```

```
>>> list(enumerate(seasons))
```

```
[(0, 'Spring'), (1, 'Summer'), (2, 'Fall'), (3, 'Winter')]
```

```
>>> list(enumerate(seasons, start=1)) # 将下标从 1 开始
```

```
[(1, 'Spring'), (2, 'Summer'), (3, 'Fall'), (4, 'Winter')]
```

## 1.6.3 list（列表）

（5）用于列表的一些常用函数

⑤**shuffle()函数**：random模块中的shuffle()函数可实现随机排列列表中的元素。

```
>>> list1=[2,3,7,1,6,12]
```

```
>>> import random #导入模块
```

```
>>> random.shuffle(list1)
```

```
>>> list1
```

```
[1, 2, 12, 3, 7, 6]
```



# 1.6.3 list（列表）

## （6）列表对象常用的方法

一旦列表对象被创建，可以使用列表对象的方法来操作列表：

| 方 法                                            | 描述                                                                     |
|------------------------------------------------|------------------------------------------------------------------------|
| <code>list.append(x)</code>                    | 在列表list末尾添加新的对象x                                                       |
| <code>list.count(x)</code>                     | 返回x在列表list中出现的次数                                                       |
| <code>list.extend(seq)</code>                  | 在列表list末尾一次性追加seq序列中的所有元素                                              |
| <code>list.index(x)</code>                     | 返回列表list中第一个值为x的元素的下标，若不存在抛出异常                                         |
| <code>list.insert(index, x)</code>             | 在列表list中index位置处添加元素x                                                  |
| <code>list.pop([index])</code>                 | 删除并返回列表指定位置的元素，默认为最后一个元素                                               |
| <code>list.remove(x)</code>                    | 移除列表list中x的第一个匹配项                                                      |
| <code>list.reverse()</code>                    | 反向列表list中的元素                                                           |
| <code>list.sort(key=None, reverse=None)</code> | 对列表list进行排序，key参数的值为一个函数，此函数只有一个参数且返回一个值，此函数将在每个元素比较前被调用，reverse表示是否逆序 |
| <code>list.clear()</code>                      | 删除列表list中的所有元素，但保留列表对象                                                 |
| <code>list.copy()</code>                       | 用于复制列表，返回复制后的新列表                                                       |

## 1.6.3 list（列表）

### （6）列表对象常用的方法

一旦列表对象被创建，可以使用列表对象的方法来操作列表：

```
>>> list2=['a','Andrew','is','from','string','test','This']
```

```
>>> list2.sort(key=str.lower) # key指定的函数用来在排序前
把列表的每个元素都转化为小写
```

```
>>> print(list2)
```

```
['a', 'Andrew', 'from', 'is', 'string', 'test', 'This']
```

## 1.6.3 list（列表）

### （7）列表生成（推导）式

列表生成式是利用其他列表创建新列表的一种方法，格式为：

[生成列表元素的表达式 for 表达式中的变量 in 变量要遍历的序列]

[生成列表元素的表达式 for 表达式中的变量 in 变量要遍历的序列 if 过滤条件]

注意：

①要把生成列表元素的表达式放到前面，执行的时候，先执行后面的for循环。

②可以有多个for循环，也可以在for循环后面加个if 过滤条件。

③变量要遍历的序列，可以是任何方式的迭代器(元组，列表，生成器...).

## 1.6.3 list（列表）

（7）列表生成（推导）式

```
>>> a = [1,2,3,4,5,6,7,8,9,10]
```

```
>>> [2*x for x in a]
```

```
[2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
```

如果没有给定列表，也可以用range()方法：

```
>>> [2*x for x in range(1,11)]
```

```
[2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
```

for循环后面还可以加上if判断，比如要取列表a中4的倍数：

```
>>> [2*x for x in range(1, 11) if x%4 == 0]
```

```
[4, 8, 12, 16, 20]
```

## 1.6.3 list（列表）

### （7）列表生成（推导）式

还可以使用三层循环，生成三个数的全排列：

```
>>> [i+ j + k for i in '123' for j in '123' for k in '123' if (i != k) and (i
!= j) and (j != k)]
```

```
['123', '132', '213', '231', '312', '321']
```

一个由男人列表和女人列表组成的嵌套列表，取出姓名中带有“涛”的姓名，组成列表：

```
>>> names = [['王涛','元芳','吴言','马汉','李光地','周文涛'],
 ['李涛蕾','刘涛','王丽','李小兰','艾丽莎','贾涛慧']]
```

```
>>> [name for lst in names for name in lst if '涛' in name]
```

#注意遍历顺序，这是实现的关键

```
['王涛','周文涛','李涛蕾','刘涛','贾涛慧']
```

## 1.6.4 tuple（元组）

元组（tuple）与列表类似，但元组的元素不能修改。元组元素写在小括号()里，元素之间用逗号隔开，元组中的元素类型可以不相同。元组可以被索引且下标索引从0开始，-1为从末尾开始，也可以进行截取。元组对象举例：(1, 2, 3, 4, 5)、("a", "b", "c")

### （1）访问元组

使用下标索引来访问元组中的值，如下实例：

```
>>> tuple1 = ('hello', 18 , 2.23, 'world', 2+4j)
```

```
>>> print(tuple1[0]) # 输出元组的第一个元素
```

```
hello
```

```
>>> print(tuple1) # 输出完整元组
```

```
('hello', 18, 2.23, 'world', (2+4j))
```

## 1.6.4 tuple（元组）

### （1）访问元组

```
>>> print(tuple1[1:3])
```

# 输出从第二个元素开始到第

三个元素

```
(18, 2.23)
```

```
>>> print(tuple1[2:])
```

# 输出从第三个元素开始的所

有元素

```
(2.23, 'world', (2+4j))
```

```
>>> print (tuple1 * 2)
```

# 输出两次元组

```
('hello', 18, 2.23, 'world', (2+4j), 'hello', 18, 2.23, 'world', (2+4j))
```

## 1.6.4 tuple（元组）

```
>>> tuple(range(6))
```

#将迭代对象转换为元组

```
(0, 1, 2, 3, 4, 5)
```

注意：构造包含0个或1个元素的元组比较特殊：

```
>>> tuple3 = ()
```

# 空元组

```
>>> tuple4 = (20,)
```

# 一个元素，需要在元素后添加逗号

任意无符号的对象，以逗号隔开，默认为元组，如下实例：

```
>>> A='a', 5.2e30, 8+6j, 'xyz'
```

```
>>> A
```

```
('a', 5.2e+30, (8+6j), 'xyz')
```



## 1.6.4 tuple（元组）

### （2）修改元组

元组属于不可变序列，一旦创建，元组中的元素是不允许修改的，也无法增加或删除元素。因此，元组没有提供`append()`、`extend()`、`insert()`、`remove()`、`pop()`方法，也不支持对元组元素进行`del`操作，但能用`del`命令删除整个元组。

➤ 元组中的元素值是不允许修改的，但我们可以对元组进行连接组合，得到一个新元组：

```
>>> tuple1 = ('hello', 18 , 2.23, 'world', 2+4j)
```

```
>>> tuple2 = ('best', 16)
```

```
>>> tuple3 = tuple1 + tuple2 # 连接元组
```

```
>>> print(tuple3)
```

```
('hello', 18, 2.23, 'world', (2+4j), 'best', 16)
```

## 1.6.4 tuple（元组）

### （2）修改元组

虽然tuple的元素不可改变，但它可以包含可变的对象，比如list列表，可改变元组中可变对象的值。

```
>>> tuple4 = ('a', 'b', ['A', 'B'])
```

```
>>> tuple4[2][0] = 'X'
```

```
>>> tuple4[2][1] = 'Y'
```

```
>>> tuple4[2][2:] = 'Z'
```

```
>>> tuple4
```

```
('a', 'b', ['X', 'Y', 'Z'])
```

## 1.6.4 tuple（元组）

### （3）生成器推导式

推导式只适用于列表，字典和集合，元组没有推导式。

```
>>> a = [1,2,3,4,5,6,7,8,9,10]
```

```
>>> b =(2*x for x in a)
```

```
>>> b #这里b变成一个生成器对象了！并不是元组！
```

```
<generator object <genexpr> at 0x0000000002F3DBA0>
```

生成器是用来创建一个Python序列的一个对象。使用它可以迭代庞大序列，且不需要在内存创建和存储整个序列，这是因为它的工作方式是每次处理一个对象，而不是一口气处理和构造整个数据结构。

## 1.6.4 tuple（元组）

### （3）生成器推导式与列表推导式的区别

- 生成器推导式使用圆括号而列表推导式使用方括号。
- 生成器推导式的结果是一个生成器对象，而不是元组。若想使用生成器对象中的元素时，可以通过`list()`或`tuple()`方法将其转换为列表或元组，然后使用列表或元组读取元素的方法来使用其中的元素。
- 此外，也可以使用生成器对象的`--next--()`方法或者内置函数`next()`进行遍历，或者直接将其作为迭代器对象来使用。
- 但无论使用哪种方式遍历生成器的元素，当所有元素遍历完之后，如果需要重新访问其中的元素，必须重新创建该生成器对象。

## 1.6.4 tuple (元组)

### (3) 生成器推导式

```
>>> list(b) #将生成器对象转换为列表
```

```
[2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
```

```
>>> list(b) #生成器对象已遍历结束，没有元素了
```

```
[]
```

```
>>> c = (x for x in range(11) if x%2==1)
```

```
>>> c.__next__() #使用生成器对象的__next__()方法获取元素
```

```
1
```

```
>>> next(c) #使用内置函数next()获取生成器对象的元素
```

```
3
```

```
>>> [x for x in c] #使用列表推导式访问生成器对象剩余的元素
```

```
[5, 7, 9]
```

## 1.6.5 dictionary（字典）

- 字典是一种映射类型，字典用"`{ }`"标识，它是一个无序的“键(key)：值(value)”对集合。“键(key)”必须使用不可变类型，如整型、浮点型、复数型、布尔型、字符串、元组等，但不能使用诸如列表、字典、集合或其它可变类型作为字典的键。在同一个字典中，“键(key)”必须是唯一的，但“值(value)”是可以重复的。
- 列表是有序的对象集合，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。

## 1.6.5 dictionary（字典）

### （1）创建字典

使用赋值运算符将使用{ }括起来的“键:值”对赋值给一个变量即可创建一个字典变量。

```
>>> dict1 = {'Alice': '2341', 'Beth': '9102', 'Cecil': '3258'}
```

```
>>> dict1['Jack'] = '1234' #为字典添加元素
```

```
>>> print(dict1) # 输出完整的字典
```

```
{'Alice': '2341', 'Beth': '9102', 'Cecil': '3258', 'Jack': '1234'}
```

```
>>> type(dict1)
```

```
<class 'dict'> #显示dict1的类型为dict
```

## 1.6.5 dictionary (字典)

### (1) 创建字典

可以使用字典dict的构造方法dict(), 利用键值对序列构建字典:

```
>>> items=[('one',1),('two',2),('three',3),('four',4)]
```

```
>>> dict2 = dict(items)
```

```
>>> print(dict2)
```

```
{'one': 1, 'two': 2, 'three': 3, 'four': 4}
```

可以通过关键字创建字典, 如下所示:

```
>>> dict3 = dict(one=1,two=2,three=3)
```

```
>>> print(dict3)
```

```
{'one': 1, 'two': 2, 'three': 3}
```

使用zip创建字典, 如下所示:

```
>>> key = 'abcde'
```

```
>>> value = range(1, 6)
```

```
>>> dict(zip(key, value))
```

```
{'a': 1, 'b': 2, 'c': 3, 'd':
4, 'e': 5}
```



## 1.6.5 dictionary（字典）

### （2）访问字典里的值

通过“字典变量[key]”的方法返回键key对应的值value:

```
>>> dict1={'Alice': '2341', 'Beth': '9102', 'Cecil': '3258', 'Jack': '1234'}
```

```
>>> print (dict1['Beth']) # 输出键为'Beth'的值
```

```
9102
```

```
>>> print (dict1.values()) # 输出所有值
```

```
dict_values(['2341', '9102', '3258', '1234'])
```

```
>>> print(dict1.keys()) # 输出所有键
```

```
dict_keys(['Alice', 'Beth', 'Cecil', 'Jack'])
```

```
>>> dict1.items() #返回所有元素
```

```
dict_items([('Alice', '2341'), ('Beth', '9102'), ('Cecil', '3258'), ('Jack',
'1234')])
```

## 1.6.5 dictionary（字典）

### （3）字典元素添加、修改与删除

向字典添加新元素的方法是增加新的键/值对：

```
>>> school={'class1': 60, 'class2': 56, 'class3': 68, 'class4': 48}
```

```
>>> school['class5']=70 # 添加新的元素
```

```
>>> school['class1']=62 # 更新 class1 的值
```

```
>>> school
```

```
{'class1': 62, 'class2': 56, 'class3': 68, 'class4': 48, 'class5': 70}
```

由上可知，当以指定“键”为下标为字典元素赋值时，有两种含义：

①若该“键”不存在，则表示为字典添加一个新元素，即一个“键:值”对；②若该“键”存在，则表示修改该“键”所对应的“值”。

## 1.6.5 dictionary（字典）

### （3）字典元素添加、修改与删除

使用字典对象的update()方法可以将另一个字典的元素一次性全部添加到当前字典对象中，如果两个字典中存在相同的“键”，则只保留另一个字典中的键值对，如下所示：

```
>>> school1={'class1': 62, 'class2': 56, 'class3': 68, 'class4': 48,
'class5': 70}
>>> school2={ 'class5': 78,'class6': 38}
>>> school1.update(school2)
>>> school1 #'class5'所对应的值取school2中'class5'所对应的值78
{'class1': 62, 'class2': 56, 'class3': 68, 'class4': 48, 'class5': 78, 'class6':
38}
```

## 1.6.5 dictionary（字典）

### （4）字典对象常用的方法

一旦字典对象被创建，可以使用字典对象的方法来操作字典，字典对象的常用方法如表1-8所示，其中dict1是一个字典对象：

| 方法                           | 描述                                                                           |
|------------------------------|------------------------------------------------------------------------------|
| dict1.copy()                 | 返回一个字典的浅复制，即复制时只会复制父对象，而不会复制对象的内部的子对象，复制后对原dict的内部子对象进行操作时，浅复制dict会受操作影响而变化。 |
| dict1.fromkeys(seq[, value]) | 创建一个新字典，以序列seq中元素做字典的键，value为字典所有键对应的初始值                                     |
| dict1.get(key, default=None) | 返回指定键key的值，如果值不在字典中返回默认值                                                     |
| dict1.items()                | 以列表形式返回可遍历的(键, 值) 元组数组                                                       |
| dict1.keys()                 | 以列表返回一个字典所有的键                                                                |
| dict1.update(dict2)          | 把字典dict2的键/值对更新到dict1里                                                       |
| dict1.values()               | 以列表返回字典中的所有值                                                                 |

## 1.6.5 dictionary (字典)

### (4) 字典对象常用的方法

```
>>> dict5 = {'Spring': '春', 'Summer': '夏', 'Autumn': '秋', 'Winter':
'冬'}
```

```
>>> dict5.items()
```

```
dict_items([('Spring', '春'), ('Summer', '夏'), ('Autumn', '秋'),
('Winter', '冬')])
```

```
>>> for key,values in dict5.items(): # 遍历字典列表
 print(key,values)
```

Spring 春

Summer 夏

Autumn 秋

Winter 冬

## 1.6.5 dictionary（字典）

### （5）字典推导式

字典推导和列表推导的使用方法是类似的，只不过是把中括号改成大括号。

```
>>> dict6 = {'physics': 1, 'chemistry': 2, 'biology': 3, 'history': 4}
```

#把dict6的每个元素键的首字母大写、键值2倍

```
>>> dict7 = { key.capitalize(): value*2 for key,value in dict6.items()
}
```

```
>>> dict7
```

```
{'Physics': 2, 'Chemistry': 4, 'Biology': 6, 'History': 8}
```

## 1.6.6 set（集合）

集合是无序可变序列，使用一对大括号作为界定符，元素之间使用逗号分隔，集合中的元素互不相同。集合的基本功能是进行成员关系测试和删除重复元素。集合不能有可变元素(如列表、集合或字典)

### （1）创建集合

使用赋值操作“=”直接将一个集合赋值给变量来创建一个集合对象：

```
>>> student = {'Tom', 'Jim', 'Mary', 'Tom', 'Jack', 'Rose'}
```

也可以使用set()函数将列表、元组等其它可迭代对象转换为集合，如果原来的数据中存在重复元素，则在转换为集合的时候只保留一个。

```
>>> set1 = set('cheeseshop')
```

```
>>> set1
```

```
{'s', 'o', 'p', 'c', 'e', 'h'}
```

## 1.6.6 set（集合）

### （2）增加集合元素

可以添加或删除集合中的元素。可以使用集合对象的`add()`方法添加单个元素，使用`update()`方法添加多个元素，`update()`可以使用元组、列表、字符串或其他集合作为参数。

```
>>> set3 = {'a', 'b'}
```

```
>>> set3.add('c') # 添加一个元素
```

```
>>> set3
```

```
{'b', 'a', 'c'}
```

```
>>> set3.update(['d', 'e', 'f']) # 添加多个元素
```

```
>>> set3
```

```
{'a', 'f', 'b', 'd', 'c', 'e'}
```



## 1.6.6 set（集合）

### （3）删除集合中的元素

- 可以使用集合对象的`discard()`和`remove()`方法删除集合中特定的元素。区别：如果集合中不存在指定的元素，使用`discard()`，集合保持不变；但在这种情况下，使用`remove()`会引发 `KeyError`。
- 集合对象的`pop()`方法用于随机删除并返回集合中的一个元素，如果集合为空则抛出异常。
- 集合对象的`clear()`方法用于删除集合的所有元素。

```
>>> set4 = {1, 2, 3, 4}
```

```
>>> set4.discard(4)
```

```
>>> set4
```

```
{1, 2, 3}
```

```
#删除元素，不存在就抛出异常
```

```
>>> set4.remove(5) Traceback
(most recent call last):
```

```
File "<pyshell#91>", line 1, in
<module>
```

```
 set4.remove(5)
```

```
KeyError: 5
```

## 1.6.6 set（集合）

### （4）集合运算

Python集合支持交集、并集、差集、对称差集等运算，如下所示：

```
>>> A={1,2,3,4,6,7,8}
```

```
>>> B={0,3,4,5}
```

使用“&”操作符执行交集操作，也可使用集合对象的方法 `intersection()` 完成，如下所示：

```
>>> A&B #求集合A和B的交集
```

```
{3, 4}
```

```
>>> A.intersection(B)
```

```
{3, 4}
```

## 1.6.6 set（集合）

### （4）集合运算

使用操作符“|”执行并集操作，也可使用集合对象的方法`union()`完成，如下所示：

```
>>> A | B
```

```
{0, 1, 2, 3, 4, 5, 6, 7, 8}
```

```
>>> A.union(B)
```

```
{0, 1, 2, 3, 4, 5, 6, 7, 8}
```

使用操作符“-”执行差集操作，也可使用集合对象的方法`difference()`完成，如下所示：

```
>>> A - B
```

```
{1, 2, 6, 7, 8}
```

```
>>> A.difference(B)
```

```
{1, 2, 6, 7, 8}
```

## 1.6.6 set（集合）

### （4）集合运算

对称差：集合A与集合B的对称差集是由只属于其中一个集合，而不属于另一个集合的元素组成的集合，使用“^”操作符执行对称差集操作，也可使用集合对象的方法 `symmetric_difference()` 完成，如下所示：

```
>>> A ^ B
```

```
{0, 1, 2, 5, 6, 7, 8}
```

```
>>> A.symmetric_difference(B)
```

```
{0, 1, 2, 5, 6, 7, 8}
```

## 1.6.6 set（集合）

### （4）集合运算

子集：由某个集合中一部分元素所组成的集合，使用操作符“<”判断“<”左边的集合是否是“<”右边的集合的子集，也可使用集合对象的方法issubset()完成，如下所示：

```
>>> A={1,2,3,4,6,7,8}
```

```
>>> C={1,3,4}
```

```
>>> C < A #C集合是A集合的子集，返回True
```

```
True
```

```
>>> C.issubset(A)
```

```
True
```

## 1.6.6 set（集合）

### （5）集合推导式

集合推导式跟列表推导式差不多，跟列表推到式的区别在于：①不使用中括号，使用大括号；②结果中无重复。

```
>>> strings =
```

```
['All','things','in','their','being','are','good','for','something']
```

```
>>> {len(s) for s in strings} #有长度相同的会只留一个
```

```
{2, 3, 4, 5, 6, 9}
```

```
>>> {s.upper() for s in strings}
```

```
{'THINGS', 'ALL', 'SOMETHING', 'THEIR', 'GOOD', 'FOR',
'IN', 'BEING', 'ARE'}
```

# 第1章 Python语言基础

- ☐ 1.1 Python应用领域
- ☐ 1.2 Python语言特点
- ☐ 1.3 编写Python代码
- ☐ 1.4 Python中的对象
- ☐ 1.5 Python中的变量
- ☐ 1.6 Python中的基本数据类型
- ✓ 1.7 Python中的运算符
- ☐ 1.8 Python中的数据输入
- ☐ 1.9 Python中的数据输出
- ☐ 1.10 Python库的导入与扩展库的安装

# 1.7 Python中的运算符

Python语言支持的运算符类型有：算术运算符、比较（关系）运算符、赋值运算符、逻辑运算符、位运算符、成员运算符、身份运算符。

## （6）Python成员运算符

Python成员运算符测试给定值是否为序列中的成员，例如字符串，列表或元组。成员运算符有两个如表1-16所示：

| 成员运算符         | 逻辑表达式             | 描述                                  |
|---------------|-------------------|-------------------------------------|
| <b>in</b>     | <b>x in y</b>     | 如果 x 在 y 序列中返回 True，<br>否则返回 False。 |
| <b>not in</b> | <b>x not in y</b> | 如果x不在 y 序列中返回 True，<br>否则返回 False   |



# 1.7 Python中的运算符

## (7) Python身份运算符

身份运算符比较两个对象的内存位置。常用的有两个身份运算符如表所示：

| 运算符    | 描述                        | 实例                                                                 |
|--------|---------------------------|--------------------------------------------------------------------|
| is     | is 是判断两个标识符是不是引用自同一个对象    | x is y, 类似 id(x) == id(y) , 如果引用的是同一个对象则返回 True, 否则返回 False        |
| is not | is not 是判断两个标识符是不是引用自不同对象 | x is not y , 类似 id(a) != id(b)。如果引用的不是同一个对象则返回结果 True, 否则返回 False。 |

# 第1章 Python语言基础

- ☐ 1.1 Python应用领域
- ☐ 1.2 Python语言特点
- ☐ 1.3 编写Python代码
- ☐ 1.4 Python中的对象
- ☐ 1.5 Python中的变量
- ☐ 1.6 Python中的基本数据类型
- ☐ 1.7 Python中的运算符
- ✓ 1.8 Python中的数据输入
- ☐ 1.9 Python中的数据输出
- ☐ 1.10 Python库的导入与扩展库的安装

# 1.8 Python中的数据输入

- Python程序通常包括输入和输出，以实现程序与外部世界的交互：程序通过输入接收待处理的数据，然后执行相应的处理，最后通过输出返回处理的结果。
- Python内置了输入函数()，()从标准输入读入一行文本，默认的标准输入是键盘。()无论接收何种输入，都被存为字符串。

#将输入的内容作为字符串赋值给name变量

```
>>> name = input("请输入：")
```

请输入： zhangsan                      #请输入： 为输入提示信息

```
>>> type(name)
```

<class 'str'>                      #显示name的类型为字符串str

# 1.8 Python中的数据输入

➤ `input()`结合`eval()`可同时接受多个数据输入，多个输入之间的间隔符必须是逗号：

```
>>> a, b, c=eval(input())
```

```
1,2,3
```

```
>>> print(a,b,c)
```

```
1 2 3
```

# 第1章 Python语言基础

- ☐ 1.1 Python应用领域
- ☐ 1.2 Python语言特点
- ☐ 1.3 编写Python代码
- ☐ 1.4 Python中的对象
- ☐ 1.5 Python中的变量
- ☐ 1.6 Python中的基本数据类型
- ☐ 1.7 Python中的运算符
- ☐ 1.8 Python中的数据输入
- ☒ 1.9 Python中的数据输出
- ☐ 1.10 Python库的导入与扩展库的安装

# 1.9 Python中的数据输出

Python有三种输出值的方式：

- 表达式语句
- `print()`函数
- 字符串对象的`format()`方法。

## 1.9.1 表达式语句输出

Python中表达式的值可直接输出。

```
>>> 1+2
```

```
3
```

```
>>> "Hello World"
```

```
'Hello World'
```

```
>>> [1,2,'a']
```

```
[1, 2, 'a']
```

## 1.9.2 print()函数输出

**print([object1,...],sep= "",end='\n',file=sys.stdout)**

参数说明:

(1) [object1,...]待输出的对象, 表示可以一次输出多个对象, 输出多个对象时, 需要用“,”分隔, 会依次打印每个object, 遇到逗号“,”会输出一个空格。举例如下:

```
>>> a1,a2,a3="aaa","bbb","ccc"
```

```
>>> print(a1,a2,a3)
```

```
aaa bbb ccc
```

(2) sep=""用来间隔多个对象, 默认值是一个空格, 还可以设置成其他字符。

```
>>> print(a1,a2,a3,sep="***")
```

```
aaa***bbb***ccc
```

## 1.9.2 print()函数输出

**print([object1,...],sep= "",end='\n',file=sys.stdout)**

参数说明:

(3) **end="\n"**参数用来设定以什么结尾,默认值是换行符,也可以换成其它字符串,用这个选项可以实现不换行输出,如使用**end=" "**:

程序:

```
a1,a2,a3="aaa","bbb","ccc"
```

```
print(a1 ,end="@")
```

```
print(a2 ,end="@")
```

```
print(a3)
```

输出: **aaa@bbb@ccc**



## 1.9.2 print()函数输出

**print([object1,...],sep= "",end='\n',file=sys.stdout)**

参数说明:

(4) 参数file设置把print中的值打印到什么地方, 可以是默认的系统输出sys.stdout, 即默认输出到终端, 可以设置file=文件储存对象, 把内容存到该文件中, 如下:

```
>>> f = open(r'a.txt', 'w') #以写的方式打开文件a.txt
```

```
>>> print('python is good', file=f)
```

```
>>> f.close() #关闭文件
```

则把python is good保存到 a.txt 文件中。

## 1.9.2 print()函数的格式化输出

**print()**函数可使用一个字符串模板进行格式化输出。

- 模板中有格式符，这些格式符为真实值输出预留位置，并说明真实数值应该呈现的格式。
- Python用一个元组将多个值传递给模板，每个值对应一个格式符。

```
>>> print("%s speak plainer than %s." % ('Facts', 'words'))
```

**Facts speak plainer than words.      #事实胜于雄辩**

- "%s speak plainer than %s."为格式化输出时的字符串模板。
- %s为一个格式符，表示一个字符串。元组('Facts', 'words')的两个元素'Facts'和'words'为分别替换第一个%s和第二个%s的真实值。
- 在模板和元组之间，有一个%号分隔，它代表了格式化操作。

## 1.9.2 print()函数的格式化输出

**print()**函数可使用一个字符串模板进行格式化输出。

整个"**%s speak plainer than %s."** % ('Facts', 'words')实际上构成一个字符串表达式，可以像一个正常的字符串那样，将它赋值给某个变量。比如：

```
>>> a = "%s speak plainer than %s." % ('Facts', 'words')
```

```
>>> print(a)
```

**Facts speak plainer than words.**

还可以对格式符进行命名，用字典来传递真实值，如下：

```
>>> print("I'm %(name)s. I'm %(age)d year old." %
```

```
{'name':'Mary', 'age':18})
```

**I'm Mary. I'm 18 year old.**

## 1.9.2 print()函数的格式化输出

Python支持大量的格式字符如表所示：

| 格式字符 | 描述               |
|------|------------------|
| %s   | 字符串 (采用str()的显示) |
| %c   | 单个字符             |
| %b   | 二进制整数            |
| %d   | 十进制整数            |
| %o   | 八进制整数            |
| %x   | 十六进制整数           |
| %e   | 指数 (基底写为e)       |
| %f   | 浮点数              |
| %%   | 字符"%"            |

## 1.9.2 print()函数的格式化输出

可以用如下的方式，对输出的格式字符进行进一步的控制：

`'%[(name)][flags][width].[precision]type'%x`

- **name**可为空，对格式符进行命名，其为键名。
- **flags**可以有+、-、' '或0。+表示右对齐。-表示左对齐。' '为一个空格，表示在正数的左侧填充一个空格，从而与负数对齐。0表示使用0填充空位。
- **width**表示显示的宽度。
- **precision**表示小数点后精度。
- **type**表示数据输出的格式类型。

**x**表示待输出的表达式。

```
>>> print("%04d" % 5)
```

```
0005
```

```
>>> print("%6.3f%%" % 2.3)
```

```
2.300%
```

## 1.9.3 字符串对象的format方法的格式化输出

`str.format()`格式化输出使用花括号`{}`来包围替换字段，也就是待替换的字符串。而未被花括号包围的字符会原封不动地出现在输出结果中。

### (1) 使用位置索引

以下两种写法是等价的：

```
>>> "Hello, {} and {}!".format("John", "Mary")
```

```
'Hello, John and Mary!'
```

```
>>> "Hello, {0} and {1}!".format("John", "Mary")
```

```
'Hello, John and Mary!'
```

花括号内部可以写上待输出的目标字符串的索引，也可以省略。

如果省略，则按`format`后面的括号里的待输出的目标字符串顺序依次替换。

## 1.9.3 字符串对象的format方法的格式化输出

`str.format()`格式化输出使用花括号`{}`来包围替换字段，也就是待替换的字符串。而未被花括号包围的字符会原封不动地出现在输出结果中。

### (1) 使用位置索引

```
>>> '{1}{0}{1}'.format('言','文')
'文言文'
```

以下两种写法是等价的：

```
>>> "Hello, {} and {}!".format("John", "Mary")
```

```
'Hello, John and Mary!'
```

```
>>> "Hello, {0} and {1}!".format("John", "Mary")
```

```
'Hello, John and Mary!'
```

花括号内部可以写上待输出的目标字符串的索引，也可以省略。

如果省略，则按`format`后面的括号里的待输出的目标字符串顺序依次替换。

## 1.9.3 字符串对象的format方法的格式化输出

`str.format()`格式化输出使用花括号`{}`来包围替换字段，也就是待替换的字符串。而未被花括号包围的字符会原封不动地出现在输出结果中。

### (2) 使用关键字索引

除了通过位置来指定待输出的目标字符串的索引，还可以通过关键字来指定待输出的目标字符串的索引。

```
>>> "Hello, {boy} and {girl}!".format(boy="John", girl="Mary")
'Hello, John and Mary!'

>>> print("{a}{b}".format(b="3",a="Python")) #输出Python3
Python3
```



# 第1章 Python语言基础

- ☐ 1.1 Python应用领域
- ☐ 1.2 Python语言特点
- ☐ 1.3 编写Python代码
- ☐ 1.4 Python中的对象
- ☐ 1.5 Python中的变量
- ☐ 1.6 Python中的基本数据类型
- ☐ 1.7 Python中的运算符
- ☐ 1.8 Python中的数据输入
- ☐ 1.9 Python中的数据输出
- ✓ 1.10 Python库的导入与扩展库的安装

# 1.10 Python库的导入与扩展库的安装

Python启动后，默认情况下它并不会将它所有的功能都加载（也称之为“导入”）进来，使用某些模块（或库，一般不做区分）之前必须把这些模块加载进来，这样就可以使用这些模块中的函数。模块把一组相关的函数或类组织到一个文件中，一个文件即是一个模块。函数是一段可以重复多次调用的代码。

## （1）库的常规导入

常规导入是最常使用的导入方式，导入方式如下所示：

**import 库名**

在导入模块时，还可以重命名这个模块，如下所示：

**import sys as system**

既可以按照和以前“**sys.方法**”的方式调用模块的方法，也可以用“**system.方法**”的方式调用模块的方法。

# 1.10 Python库的导入与扩展库的安装

## (2) 使用from语句导入

很多时候只需要导入一个模块或库中的某个部分，这时候可通过联合使用import和from来实现这个目的：

```
from math import sin
```

上面这行代码可以让我们直接调用sin：

```
>>> from math import sin
```

```
>>> sin(0.5) #计算0.5弧度的正弦值
```

```
0.479425538604203
```

也可以一次导入多个函数：

```
>>> from math import sin, exp, log
```

也可以直接导入math库中的所有函数，导入方式如下所示：

```
>>> from math import *
```

# 1.10 Python库的导入与扩展库的安装

## (3) 扩展库的安装

当前，**pip**已成为管理Python扩展库和模块的主流方式，使用**pip**不仅可以查看本机已安装的Python扩展库和模块，还支持Python扩展库和模块的安装、升级和卸载等操作。**pip**命令的常用方法如表所示：

| pip命令示例                   | 描述         |
|---------------------------|------------|
| pip install xxx           | 安装xxx模块    |
| pip list                  | 列出已安装的所有模块 |
| pip install --upgrade xxx | 升级xxx模块    |
| pip uninstall xxx         | 卸载xxx模块    |

# 1.10 Python库的导入与扩展库的安装

## (3) 扩展库的安装

使用pip命令安装Python扩展库，需要保证计算机联网，然后在命令提示符环境中通过`pip install xxx`进行安装，这里分两种情况：

(1) 如果Python安装在默认路径下，打开控制台直接输入：`pip install 扩展库或模块名`即可。

(2) 如果Python安装在非默认环境下，在控制台中，需进入到`pip.exe`所在目录（位于Scripts文件夹下），然后再输入“`pip install 扩展库或模块名`”即可。

**THE END**