列表

用方括号表示列表（自认为是数组） "[]"

```
# 定义列表
bicyles = ['trek', 'cannondale', 'redline','specialized']
#输出列表
print(bicycles)
```

此段代码可将列表bicycles打印出来

访问列表元素与数组同理下标0即表示第一个元素

但不同于数组的是，列表是动态的，可以增加删除元素

列表元素的修改

```
# 定义列表
motorcycles = ['honda','yamaha','suzuki']
print(motorcycles)
# 修改列表元素
motorcycles[0] = 'ducati'
print(motorcycles)
```

列表中添加元素

在列表尾部添加元素

```
# 在列表尾部添加元素 使用 append方法
motorcycles.append('ducati')
```

在列表中插入元素

```
# 在列表中插入元素使用insert方法 insert()方法要指出插入的位置和字段值
motorcycles.insert(0,'ducati')
```

在列表中删除元素

1.使用del方法

```
del motorcycles[0]
```

使用del方法将值删除后，就无法再访问它了

2.使用pop()方法删除元素

有时将元素从列表中删除，并接着使用它的值时需要使用pop()方法

```
# 方法pop()可以删除列表末尾的元素，并让你能够接着使用它
mortorcycles.pop()
# 实际上可以用pop弹出列表中任何位置的元素，只需在括号中加入对应的索引即可
```

3.根据值删除元素

当只知道要删除的元素的值，可使用方法remove()，方法remove只删除第一个指定的值，若该值出现多次则需要使用循环来判断是否删除了所有这样的值。

```
motorcycles.remove('yamaha')
```

组织列表

使用方法sort()对列表进行永久性排序

```
carss = ['bmw','audi','toyota','subaru']
cars.sort()
print(cars)
```

方法sort()永久性地修改了列表元素的排序，改为按字母顺序排序，再也无法恢复到原来的排列顺序

但还可以按字母顺序相反的顺序，只需向sort方法传递reverse=True参数即可

```
cars.sort(reverse=True)
```

使用函数sorted()对列表进行临时性排序

```
print("Here is the original list:")
print(cars)
# sorted () 方法临时排序
print("\nHere is the sorted list:")
print(sorted(cars))

print("\nHere is the original list again:")
print(cars)

```

下面为输出

```
Here is the original list:
['bmw', 'audi', 'toyota', 'subaru']
Here is the sorted list:
['audi', 'bmw', 'subaru', 'toyota']
Here is the original list again:
['bmw', 'audi', 'toyota', 'subaru']
```

还可用reverse()方法将列表原地逆置

```
cars = ['bmw', 'audi', 'toyota', 'subaru']
print(cars)
cars.reverse()
print(cars)
```

确定列表的长度

```
cars = ['bmw', 'audi', 'toyota', 'subaru']
len(cars)
```

python计算列表元素数时从1开始，不必担心从0开始而出现错误

深入研究循环

```
for item in list_of_items:
```

这种循环命名方式有助于明白for循环中对每个元素的理解

在for循环中执行更多的操作

因为python以缩进为代码块，故在for循环冒号后的下一行缩进都为for循环结构体

注意避免缩进错误，或忘记缩进而导致循环体出现错误，也不要忘了冒号，for循环的冒号告诉Python下一行为for循环的循环体

创建数值列表

使用函数range()

```
1  for value in range(1,5)
2  print(value)
```

上述代码好像应该打印数字1～5，但实际上不会打印数字5，所以即为1，2，3，4，且一行一个

使用range()创建数字列表

要创建数组列表，可使用函数list()将range()的结果直接转为列表

```
1  numbers = list(range(1,6))
2  print(numbers)
```

使用range时还可以指定步长

切片

```
1  motorcycles = ['yamaha','suzuki']
2  motorcycles1 = motorcycles[:]
```

可对列表进行切片操作[M:N:K]

从M开始到N结束，步长为K，全都可以缺省

元组

元组使用圆括号来标识，定义元组后，就可以用索引来访问其元素，就像访问列表元素一样，但修改元组的操作是被禁止的。虽不能修改元组的元素，但可以给存储元组的变量赋值

```
1  dimensions = (200, 50)
2  print("Original dimensions:") for dimension in dimensions:
3    print(dimension)
4  dimensions = (400, 100)
5  print("\nModified dimensions:")
6  for dimension in dimensions:
7    print(dimension)
```

首先定义了一个元组，并将其存储的尺寸打印了出来；接下来，将一个新元组存储到变量dimensions中；然后，打印新的尺寸。

字典

在Python中，字典是一系列键-值对。每个键都与一个值相关联，你可以使用键来访问与之相关联的值。与键相关联的值可以是数字、字符串、列表乃至字典。事实上，可将任何Python对象用作字典中的值。
在Python中，字典用放在花括号{}中的一系列键-值对表示

```
1  alien_0 = {'color':'green','point':5}
```

添加键-值对

字典是一种动态结构，可随时在其中添加键值对，可依次指定字典名、用方括号括起的键和相关联的值。

```
alien_0 = {'color': 'green', 'points': 5}
print(alien_0)
alien_0['x_position'] = 0
alien_0['y_position'] = 25
print(alien_0)
```

以下为输出

```
{'color': 'green', 'points': 5}
{'color': 'green', 'points': 5, 'y_position':25,
 'x_position': 0}
```

先创建一个空字典

```
alien_0 = {}
alien_0['color'] = 'green'
alien_0['points'] = 5
print(alien_0)
```

修改字典中的值

要修改字典中的值，可依次指定字典名、用方括号括起的键以及与该键相关联的新值

例如

```
alien_0 = {'color': 'green'}
print("The alien is " + alien_0['color'] + ".")
alien_0['color'] = 'yellow'
print("The alien is now " + alien_0['color'] + ".")
```

删除键值对

```
alien_0 = {'color': 'green', 'points': 5}
print(alien_0)
del alien_0['points']
print(alien_0)
```

删除的键值对永远消失了。

遍历字典

```
user_0 = {
'username': 'efermi',
'first': 'enrico',
'last': 'fermi',
}
for key, value in user_0.items():
  print("\nKey: " + key)
  print("Value: " + value)
```

遍历字典中的所有键

```
favorite_languages = { 'jen': 'python',
'sarah': 'c',
'edward': 'ruby',
'phil': 'python', }
for name in favorite_languages.keys():
  print(name.title())
```

按顺序遍历字典中的所有键

要以特定的顺序返回元素，一种办法时在for循环中对返回的键进行排序。为此可使用函数sorted()来获得按特定顺序排列的键列表的副本

```
favorite_languages = {
  'jen':'python',
  'sarah':'c',
  'edward':'ruby',
  'phil':'python',
}
for name in sorted(favorite_languages.keys()):
  print(name.title()+",thank you for taking the poll")
```

嵌套字典列表

```
alien_0 = {'color': 'green', 'points': 5}
alien_1 = {'color': 'yellow', 'points': 10}
alien_2 = {'color': 'red', 'points': 15}
aliens = [alien_0, alien_1, alien_2]
for alien in aliens:
  print(alien)
```

字典自认为可以理解为C中的结构体

用户输入和while循环

函数input()的工作原理

函数input()让程序暂停运行，等待用户输入一些文本。获取用户输入后，python将其存储在一个变量中，以方便使用。

使用int()来获取数值输入

input输入的是字符串，而想获得数值输入时可以使用int()

while循环简介

```
current_number = 1
while current_number <= 5:
  print(current_number)
  current_number += 1
```

同大部分高级编程语言相同，有break和continue关键字可退出循环

函数

定义函数

```
def greet_user():
  """显示简单的问候语"""
  print("Hello!")

greet_user()
```

向函数传递信息

```
def greet_user(username):
  """显示简单的问候语"""
  print("Hello",+ username.title() + "!")
greet_user('jesse')
```

实参和形参

懂得都懂

传递任意数量的实参

```
def make_pizza(*toppings):
  print(toppings)
make_pizza('pepperoni')
make_pizza('mushrooms','green peppers','extra cheese')
```

类

创建和使用类

使用类几乎可以模拟任何东西

创建Dog类

```
class Dog():
  def __init__(self,name,age):
    self.name = name
    self.age = age

  def sit(self):
    print(self.name.title()+" is now sitting.")

  def roll_over(self):
    print(self.name.title()+"rolled over!")
```

方法__ init __()
类中的函数称为方法，这是一个特殊的方法，每当你根据Dog类创建实例时，Python都会自动运行他。在这个方法的名称中，开头和末尾各有两个下划线，这是一种约定，旨在避免Python默认方法与普通方法发生名称冲突，而形参self必须被包含进去，因为python调用init函数时，将自动传入实参self，它是一个指向实例本身的引用，让实例能够访问类中的属性和方法。我们创建Dog实例时，Python将调用Dog类的方法init。self会自动传递，因此不需要传

递它。

根据类创建实例

```
class Dog():
略
my_dog = Dog('willie',6)

print("My dog's name is" + my_dog.name.title()+".")
print("My dog is" + str(my_dog.age) + "yars.old.")
```

文件和异常

从文件中读取数据

```
with open('pi_digits.txt') as file_object:
  contents = file_object.read()
  print(contents)
```

函数open接受一个参数：要打开文件的名称，函数open返回一个表示文件的对象,python将这个对象存储在我们将在后面使用的变量中

关键字with在不再需要访问文件后将其关闭。

逐行读取

读取文件时，常常需要检查其中的每一行

```
filename = 'pi_digits.txt'

with open(filename) as file_object:
  for line in file_object:
    print(line)
```

创建一个包含文件各行内容的列表

```
filename = 'pi_digits.txt'
with open(filename) as file_object:
  lines = file_object.readline()
for line in lines:
  print(line.rstrip())
```

readline方法从文件中读取每一行，并将其存储在一个列表；接下来，该列表被存储到变量lines中；在with代码块外，我们依然可以使用这个变量。

使用文件的内容

```
filename = 'pi_digits.txt'
with open(filename) as file_object:
lines = file_object.readlines()
pi_string = ''
for line in lines:
        pi_string += line.rstrip()
print(pi_string)
print(len(pi_string))
```

写入空文件

要将文本写入文件，在调用open()时需要提供另一个实参，告诉Python你要写入打开的文件。

```
filename = 'programming.txt'
with open (filename,'w') as file_object:
  file_object.write("i love programming.")
```

open提供了两个实参，第一个实参也是要打开的文件的名称；而第二个实参 w 告诉python要以写入模式打开这个文件。打开文件时，可指定读取模式'r',写入模式'w',附加模式'a',读取和写入文件模式'r+'，若省略了模式实参，Python将以默认的只读模式打开文件。

python只能将字符串写入文本文件，要将数值数据存储到文本文件中，必须先使用函数str()将其转换为字符串模式

写入多行数据

要想写入的语句在多行中需要加入换行符

附加到文件

如果要给文件添加内容，而不是覆盖原有的内容，可以附加模式打开文件。以附加模式打开文件时，Python不会再返回文件对象前清空文件，而你写入到文件的行都将添加到文件末尾。如果指定的文件不存在，Python将为你创建一个空文件。

```
filename = 'programming.txt'
with open(filename,'a') as file_object:
  file_object.write(
  "i also love finding meaning in large datasets.\n")
  file_object.write(
  "i love creating apps that can run in a browser.\n")
```