Python 概述

Python 简介

Python 语言是一种面向对象的(现在的语言不支持面向对象的，已经非常少了。)解释型的编程语言。什么是解释型的语言？就是不需要先对源代码进行编译，就可以通过解释器运行的语言。Python、PHP、JavaScript 等都是解释型的语言。与解释型语言相对的是编译型语言。这类编程语言需要编译器将源代码编译成机器语言，然后直接运行。最著名的编译型语言就是 C/C++了。

Python 由 Guido van Rossum 于 1989 年底发明，第一个公开发行版发行于 1991 年。Python 源代码同样遵循 GPL(GNU General Public License)协议。

Python 语言应用非常广泛，既可以编写大型的应用程序，也可以当作一种日常工作的辅助工具。比如 Google 的一些产品就是使用 Python 开发的。Python 的最大特点就是有非常多的支持库，基本上很多基础的功能都有人实现了。你要做的有 3 点：

（1）了解 Python 的语法

（2）找到你要的支持库，比如 SNMP 协议支持库等

（3）调用需要的支持库，实现你要的功能

这对开发者是极大的帮助，也极大的提高了开发效率。开发者可以关注功能本身，而不是一些复杂的协议与算法。有专门的人，已经帮助你把这些基础的工作完成了。这也是互联网与开源软件对软件行业的贡献之一。

如何学习 Python 语言？有以下几种方法：

（1）购买书籍。实体书的作用还是很重要的，特别是对于没有编程经验的初学者。

（2）互联网资料。Python 非常容易上手，对于有编程经验的人，可以通过学习互联网上的资料。当然，如果要成为 Python 的高手于专家，还是要下一番苦功的。越是简单的东西，可能越是有很多的‘坑’。

（3）上培训课。如果你有时间与钱，这个方法也是非常不错的。

说了这么多，怎么使用 Python？

（1）首先要有一台计算机(不是开玩笑，很多电器的说明手册都强调，你要打开电源开关才能使用。如果机器不工作，第一个要做的就是检查电源开关）。

（2）其次能够访问 Python 的网站（强烈建议到 Python 的官方网站去下载 Python 程序包，不要去自己不熟悉的网站随便下载。Python 的官方网址：

https://www.python.org/

Python 的下载地址：

https://www.python.org/downloads/

根据你的计算机安装的操作系统，选择相应的平台版本下载到你的计算机。然后运行安装就可以了。如果你碰上问题怎么办？最好的方法是使用百度等搜索引擎寻找帮助。

（3）安装完成之后，打开一个命令行窗口，输入 python 或者 py 就可以运行。下图就是运

行界面。同样的，我执行的是 print(“Hello World!”)。

```
C:\>py
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:59:51) [MSC v.1914 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print('Hello World')
Hello World
>>>
```

本章的后 4 部分会简单的介绍 Python 语言的基本语法，更详细的介绍可以参考其他书籍。本书的重点是介绍 Python 内置的 tkinter 图形化编程。

1.1 变量

变量是一个语言的基础，是最基本的单元与要素。任何编程语言都离不开变量。Python 的变量非常自由，不需要指定变量的类型，直接就可以赋值使用。比如：

Var=1 #定义整数

Var=10.1 #定义浮点数

Var=’Hello World’ #定义字符串

Python 内置了下面几种数据类型或者数据结构：

（1）列表（list：用[ ]符号表示）

（2）元祖（tuple：用( )符号表示）

（3）字典（dict：用{ }符号表示）

（4）集合（使用 set() 创建集合）

```
>>> list
<class 'list'>
>>> tuple
<class 'tuple'>
>>> dict
<class 'dict'>
>>> set
<class 'set'>
>>>
```

Python 也不用去申请内存、释放内存，只要专注自己的功能就可以了。

1.1.1 列表（list）

Python 的列表和通常意义的数组类似。在实际的开发过程中，列表也的确是当成数组被广泛的使用。不过 Python 的列表更强大一些。一般语言的数字，只能放入相同类型变量，比如整型数组、字符串数组等。而在 Python 中，可以把混合的类型放在一个列表当中，程序员不用刻意区分。这是因为 Python 把数据类型都定义为类，而不是一般语言用字节等方法来表示。下图是使用调试工具得到的列表变量的属性，就是我们操作列表的函数或者方法。在后面的章节有这些函数的介绍。

Assistant | Object inspector

list @ 0x1EDE1C15CC8

Name	Value ID
append	0x1EDE1C90288
clear	0x1EDE2348DC8
copy	0x1EDE2348FC0
count	0x1EDE2348F78
extend	0x1EDE2348EE8
index	0x1EDE234E048
insert	0x1EDE234E090
pop	0x1EDE234E0D8
remove	0x1EDE234E120
reverse	0x1EDE234E168
sort	0x1EDE234E1B0

1.1.1.1 列表作为数组

array 和 string 分别是整形数组和字符串数组，不需要特别的变量说明。

```
>>> array = [1,2,3] #整形数组
>>> print(array)
[1, 2, 3]
>>> string = ['a','b','c'] #字符串数组
>>> print(string)
['a', 'b', 'c']
>>>
```

1.1.1.2 列表作为多维数组

Python 的多维数组，就是在列表中加入列表。比如 marray=[[1,2,3],[4,5,6],[7,8,9]]就是一个多维数组。

```
>>>
>>> marray=[[1,2,3],[4,5,6],[7,8,9]] #多维数组
>>> print(marray[1][2])
6
>>>
```

1.1.1.3 列表作为混合类型数组

Python 的对变量的类型没有特别的要求，所有在列表中，可以添加各种变量，甚至是函数。比如：

```
def func(): #定义一个函数
```

```
    print('Hello World')
mix_array=[1,'abc',func]
print(mix_array[0]) #输出整形数
print(mix_array[1]) #输出字符串
print(mix_array[2]) #输出函数的标识
```

```
>>> def func(): #定义一个函数
...     print('Hello World')
...
>>> mix_array=[1,'abc',func]
>>> print(mix_array[0]) #输出整形数
1
>>> print(mix_array[1]) #输出字符串
abc
>>> print(mix_array[2]) #输出函数的标识
<function func at 0x0000021FD62BC1E0>
>>>
```

那么把函数放入到列表中有什么作用呢？作用就是列表中加入的函数是可以执行的。方法就是在变量的后面加入()：

```
mix_array[2]()
```

```
>>> mix_array[2]() #执行函数
Hello World
>>>
>>>
```

如果函数带有参数，在括号内输入参数就可以了。比如：

```
mix[2]('你好')
```

```
>>> def func_var(string):
...     print(string)
...
>>> mix2=[1,'abc',func_var]
>>> mix2[2]('你好')
你好
>>>
>>>
```

列表以及其他的 Python 变量能够使用混合的元素的根本原因是 Python 的变量只是指向了变量的地址，而不是存储的实际变量本身。什么意思呢？就是每一个在列表中的元素存储的都是一个 8 个字节长的地址，真正的变量是存储在这个地址里面的。利用这个方法，列表就可以容纳混合类型。怎么证明呢？用列表本身的__sizeof__()方法就可以。

```
print('空的列表占用字节:',list_t.__sizeof__())
```

* 使用 append 方法增加列表的元素，可以看到列表并不是只增加 8 个字节，也就是一个元素的空间，而是第一次增加 32 个字节（4 个元素）。如果增加的元素超过 4 个，列表第二次增加的长度就是 32 个字节（新增 4 个元素），第三次是 64 个字节（8 个元素），以此类推，增加的长度与前一次相比，是以 32，64，128，.....往上增长的。申请内存也是需要时间的，每次只增加 8 个字节的内存的确节省空间，但是增加了运行的时间。一次多申请一些

并且渐进式的增加内存的申请量，是一种很好的平衡时间与空间的方法，在很多的商业化产品中都有这种设计。我们在日常的设计中，也可以使用。

程序	结果
from def print_list(list_arg): 　　print("目前列表中的元素个数:%d, 列表占用的内存长度：%d"%(len(list_arg),list_arg._sizeof_())) list_1=[] print_list(list_1) for i in range(20): 　　list_1.append(i) 　　print_list(list_1)	目前列表中的元素个数:0,列表占用的内存长度：40 目前列表中的元素个数:1,列表占用的内存长度：72 目前列表中的元素个数:2,列表占用的内存长度：72 目前列表中的元素个数:3,列表占用的内存长度：72 目前列表中的元素个数:4,列表占用的内存长度：72 目前列表中的元素个数:5,列表占用的内存长度：104 目前列表中的元素个数:6,列表占用的内存长度：104 目前列表中的元素个数:7,列表占用的内存长度：104 目前列表中的元素个数:8,列表占用的内存长度：104 目前列表中的元素个数:9,列表占用的内存长度：168 目前列表中的元素个数:10,列表占用的内存长度：168 目前列表中的元素个数:11,列表占用的内存长度：168 目前列表中的元素个数:12,列表占用的内存长度：168 目前列表中的元素个数:13,列表占用的内存长度：168 目前列表中的元素个数:14,列表占用的内存长度：168 目前列表中的元素个数:15,列表占用的内存长度：168 目前列表中的元素个数:16,列表占用的内存长度：168 目前列表中的元素个数:17,列表占用的内存长度：240 目前列表中的元素个数:18,列表占用的内存长度：240 目前列表中的元素个数:19,列表占用的内存长度：240 目前列表中的元素个数:20,列表占用的内存长度：240

1.1.1.4 判断一个变量是否为列表

Python 由于变量的声明不需要指明类型，在使用的过程中如果需要判断变量的类型，就需要使用内置的函数 isinstance 和 type。推荐使用 isinstance。
Instance 的用法是：isinstance(var, <类型>)。type 的用法是 type(var) is <类型>。

```
1
2 list_a = ['1','2']
3
4 print(isinstance(list_a,list))
5 print(type(list_a) is list)
6
```

输出的结果都是 True。
其他的类型也可以用此种方法来判断。比如判断一个变量是否为整形数等等。

1.1.1.5 列表与内置函数

列表是 Python 中常用的数据类型。
如下内置函数可以返回列表信息:
1、len(list)：返回列表元素的个数
2、max(list)：返回列表中最大值
3、min(list)：返回列表中最小值
4、list(slice(start,stop,step)):slice 是切片函数，返回列表的一个子集。切片的从 start 开始，以 step 为步长，到 stop 位置结束。
* len,max,min 和 slice 都是内置函数

```
1
2  list_a = [1,3,5,6,9,11,25,13]
3
4  print("结果:")
5  print("len: %d"%len(list_a))
6  print("max:",max(list_a))
7  print("min:",min(list_a))
8  print("slice:",list_a[slice(2,8,2)])
9
```

```
结果:
len: 8
max: 25
min: 1
slice: [5, 9, 25]
```

1.1.1.6 列表的方法

1、list.append(object)：在列表末尾添加新的元素
2、list.count(object)：统计某个元素在列表中出现的次数
3、list.extend(seq)：在列表末尾一次性追加另一个序列中的多个值
4、list.index(object)：从列表中找出第一个匹配对象(object)的索引位置
5、list.insert(index, object)：将对象(object)插入列表，插入的索引位置是 index
6、list.pop(index)：移除列表中 index 位置的一个元素，并且返回该元素的值
7、list.remove(object)：移除列表中指定对象(object)的第一个匹配项
8、list.reverse()：反向列表中元素
9、list.sort([func])：对原列表进行排序

```
list_a = [1,3,5,6,9,11,25,13]

print("结果:")
list_a.append(3)
print("append list_a:",list_a)
print("count:",list_a.count(3))
list_a.extend([4,5,6])
print("extend:",list_a)
print("index:",list_a.index(5))
list_a.pop(4)
print("list:",list_a)
list_a.remove(5)
print("remove:",list_a)
list_a.reverse()
print("reverse:",list_a)
list_a.sort()
print("sort:",list_a)
```

```
结果:
append list_a: [1, 3, 5, 6, 9, 11, 25, 13, 3]
count: 2
extend: [1, 3, 5, 6, 9, 11, 25, 13, 3, 4, 5, 6]
index: 2
list: [1, 3, 5, 6, 11, 25, 13, 3, 4, 5, 6]
remove: [1, 3, 6, 11, 25, 13, 3, 4, 5, 6]
reverse: [6, 5, 4, 3, 13, 25, 11, 6, 3, 1]
sort: [1, 3, 3, 4, 5, 6, 6, 11, 13, 25]
```

1.1.2 元组（tuple）

元组和列表非常类似，区别就是元组的元素是不能修改的。修改元组中的元素会产生 TypeError 异常：

```
tuple_a=(1,3,4,5)
tuple_a[1]=100
```

结果如下：

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

1.1.2.1 元组与内置函数

1、len(tuple)：返回元组元素的个数
2、max(tuple)：返回元组中最大值
3、min(tuple)：返回元组中最小值
4、tuple(slice(start,stop,step)):slice 是切片函数，返回列表的一个子集。切片的从 start 开始，以 step 为步长，到 stop 位置结束。

```
tuple_a = [1,3,5,6,9,11,25,13]

print("结果:")
print("len: %d"%len(tuple_a))
print("max:",max(tuple_a))
print("min:",min(tuple_a))
print("slice:",tuple_a[slice(2,8,2)])
```

```
结果:
len: 8
max: 25
min: 1
slice: [5, 9, 25]
```

1.1.2.2 元组的方法

1、tuple.count(object)：统计某个元素在元组中出现的次数
2、tuple.index(object)：从元组中找出第一个匹配对象(object)的索引位置

```
tuple_a = (1,3,5,6,9,1,25,3)

print("结果:")
print("count:",tuple_a.count(3))
print("index:",tuple_a.index(1))
```

```
结果:
count: 2
index: 0
```

元组一般用于不可修改的程序变量。可以认为是常量。

1.1.3 字典（dict）

字典是另外一个经常使用的类型。字典以 key,value 的形式存储数据。比如：
dict_a={'数学':95, '语文':88, '英语':100}
字典的访问速度快，但是占用的内存多。字典的 key 是唯一的，对已经存在的 key 操作，相当于赋值：

```
dict_a['数学']=100    #此时替换了以前的数值 95
```

1.1.3.1 字典与内置函数：

1、len(dict)：返回字典中元素的个数
2、max(dict)：返回字典键值中最大值
3、min(dict)：返回字典键值中最小值

```
dict_a={'数学':95, '地理':88, '英语':100}

print("结果:")
print(dict_a)
print("len:",len(dict_a))
print("min:",min(dict_a))
print("max:",max(dict_a))
```

```
结果:
{'数学': 95, '地理': 88, '英语': 100}
len: 3
min: 地理
max: 英语
```

1.1.3.2 字典的方法

1、dict.clear(): 清空字典

2、dict.copy(object): 拷贝一个字典到新的字典中。通常情况下，'='并不会产生一个新的变量拷贝，使用内置的 copy()函数才会产生一个新的变量拷贝。

3、dict.fromkeys([seq,value]): 以 seq 为键，以 value 为初始值，构造一个新的字典

4、dict.get(key,default=None): 返回指定 key 的值，如果 key 不存在则返回 default 指定的值。

5、dict.items()：以列表形式返回可遍历的(键, 值) 元组数组。最常见的遍历字典的方法：

```
for key,value in dict.items():
    print(key,value)
```

6、dict.keys():返回一个字典的所有键，以列表的形式。此种方法也可以遍历字典：

```
for key in dict.keys():
    print(key,dict[key])
```

7、dict.pop(key[,default]):删除字典给定键 key 所对应的值，返回值为被删除的值。key 值必须给出。 如果 key 不在字典中，而且没有设定 default 值，会产生一个'KeyError'的异常。

8、dict.popitem():删除字典中最后一个(key,value)，返回删除的(key,value)。如果字典为空，则产生一个'KeyError'异常。

9、dict.setdefault(key, default=None):和 get() 类似, 如果键不存在于字典中，将会添加键并将值设为默认值。

10、dict.update(dict2):把字典 dict2 的键/值对拷贝到 dict 里。

11、dict.values():以列表返回字典中的所有值。

字典的赋值要比列表慢很多，不过查找字典中的元素要比列表稍微快一点。主要还是看在程序中使用那种类型比较方便。有的时候字典更直接与直观。

1.1.4 集合（set()）

集合是一个无序的不重复元素序列，也就是说集合中的元素没有重复的。这点非常关键。

1.1.4.1 集合初始化

集合可以用{}或者 set()初始化。方法如下：

```
set_first = {'苹果','桔子','香蕉','桔子'}
```

或者

```
set_second=('accounting')
```

集合中的元素是不重复的，那么上面两种初始化的方法，其中都有重复的元素，这些重复的元素都会被去掉而只保留一个：

```
set_first = {'苹果','桔子','香蕉','桔子'}

set_second=set('accounting')

print('\n\n')
print("set_first: ",set_first)
print("set_second: ",set_second)
```

```
set_first:  {'桔子', '香蕉', '苹果'}
set_second:  {'n', 'u', 'a', 'o', 'c', 'g', 't', 'i'}
```

集合的不重复用处很大，另外还要记住集合是无序的。同样的遍历集合，会输出的次序不一样。这一点与列表、元组和字典是一个很大的区别。列表、元组和字典，用同样的方法遍历，输出的结果是一致的，而集合不是。比如：

list=[1,2,6,3,4]

print(list)

输出的结果一定是 [1,2,6,3,4]

set_second=('accounting')

print(set_second)

输出的结果是不一致的。每一次调用的集合输出结果是不同的。

1.1.4.2 集合的方法：

1、set.add(element)：将元素 element 添加到集合中。如果元素 element 已经存在，则不进行任何改变。

2、set.clear()：清空集合

3、set.copy(set1)：拷贝一个集合。用法是 new_set = set.copy(old_set)

4、set.difference(set1,set2)：返回集合的差集。用法是 difference_set = target_set.difference(set1,set2)。结果是 difference_set 中与 set1 和 set2 不同的元素集合。可以输入多个集合参数。

5、set.difference_update(set1,set2)：与 difference 方法类似，不过会移除在其他集合也存在的元素。也就是说，在原始的集合中如果存在与其他集合相同的元素，那么这些元素会被从原始集合中移除，只在原始集合中保留差集。和数学里面的集合运算是一个道理。

6、set.discard(element)：从集合中移除元素 element

7、set.intersection(set1,set2)：返回集合的交集。也就是原始集合与其他集合共同的元素集合。

8、set.intersection_update(set1,set2):把原始集合更新为与其他集合的交集。

9、set.isdisjoint()：判断两个集合是否包含相同的元素，如果没有返回 True，否则返回 False。

10、set.issubset(set1)：判断是否是集合 set1 的子集

11、set.issuperset(set1)：判断是否是集合 set1 的超集

12、set.pop(): 随机删除一个元素，并返回这个值

13、set.remove(element)：从集合中删除元素 element，如果元素 element 不存在会报错。

14、set.symmetric_difference(set1):返回两个集合中不重复(不同)的元素。就是 set 中有而 set1 中没有的元素以及 set 中没有而 set1 中有的元素。

15、set.symmetric_difference_update(set1): 和 symmetric_difference 类似，不过会更新 set 集合。

16、set.union(set1)：并集。返回两个集合中不重复的所有元素。

17、set.update(element):update 方法和 add 类似。不过 update 方法会把 element 拆开加入到集合。比如 set.update('abc')是把'a','b','c'分别加入到集合 set 中。而 set.add('abc')是把'abc'加入到集合 set 中。

数学符号	Python 中的符号	意义
- 或是 \	-	差集或者相对补集
∩	&	交集
∪	|	和集，并集
△	^	对称差集
=	==	等于
∈	in	成员关系
∉	not in	不是成员关系

1.2 逻辑控制

Python 的语句是顺序执行的，其逻辑控制语句有：条件语句，for 循环语句、while 循环语句、break 语句、continue 语句和 pass 语句。

1.2.1if...else

if 语句在判断条件为真的时候执行后面的语句块，否则就跳过该语句块而执行 else 的语句块(如果有 else 的话)。如果需要多重判断，可以使用 elif。Python 没有 switch 语句。
如果条件判断中有多个条件，可以使用 and 和 or 来组合，比如 i>10 and i<5。也可以使用 not。判断条件不需要()括起来。if (a>1): 是会报错的。不过 if (a>1 and a < 10) or b ==1: 可以的。

```
if 判断条件 1:
    执行语句块 1……
elif 判断条件 2:
    执行语句块 2……
elif 判断条件块 3:
    执行语句块 3……
else:
    执行语句 4……
```

1.2.2for 循环

for 循环根据条件多次执行满足条件的语句体。

for i in range(5):

执行语句块……

程序	结果	说明
for i in range(5): print(i)	0 1 2 3 4	

1.2.3while

在给定的判断条件为 true 时执行循环体，否则退出循环体。

while 判断条件==True:

执行语句块

程序	结果	说明
pets = ['dog','cat','goldfish','rabbit','cat'] while 'cat' in pets: pets.remove('cat') print(pets)	['dog','goldfish','rabbit']	

1.2.4break

退出执行循环语句。有的时候，在循环执行的中间要结束循环，就可以使用 break 语句，尤其是 while 语句中，经常使用 break 语句。

程序	结果	说明

for i in range(10):
if i == 5:
break
print(i)
	0
	1
	2
	3
	4
		当 i 等于 5 的时候跳出了循环

1.2.5continue

在语句块执行过程中终止当前循环，执行下一次循环。Continue 语句只是结束本次循环，并不是跳出循环。这与 break 语句不同。

程序	结果	说明

for i in range(10):
    if i == 5:
        continue
    print(i)
	0
	1
	2
	3
	4
	6
	7
	8
	9
		当 i 等于 5 的时候，结束了循环，开始下一个循环。

1.2.6pass

pass 是空语句，是为了保持程序结构的完整性

程序	结果	说明

for i in range(10):
    if i == 5:
        pass
    print(i)
	0
	1
	2
	3
	4
	5
	6
	7
	8
	9
		当 i 等于 5 的时候，继续执行循环，没有任何其他的动作。

1.3 内置函数

Python 的一些内置函数在前面的章节已经有一些介绍了。这里再统一归纳总结一下：

1.3.1 len

len 函数返回变量的长度。变量可以是很多种，比如列表、字典、集合或者自己定义的类。所以，我们自己定义的类只要有 len()函数，就可以使用内置函数 len 得到自定义类的长度。比如下面的输出的类 A 的结果就是 10。

程序	结果	说明

class A():
    pass
def len(self):
    return 10
print(len(A))
	10	

1.3.2 min(args1,args2,args3,.....)

min 返回输入参数中的最小值。Python 比较字符串大小时，根据的是 ord 函数得到的编码值。基于它的排序函数 sort 可以很容易为数字和英文字母排序，因为它们在编码表中就是顺序排列的。

对于中文的排序就复杂了。汉字的编码由好几种方法，比如 GB2312、GBK、UTF-8 等。不同的编码方法，会得到不同的大小结果，因为同一个汉字在编码表中的位置是不同的。因此，如果要使用汉字排序、判断大小要清楚使用的编码格式才可以。一般都是使用自定义的处理函数，否则结果难以预期。网络上有很多的处理中文排序的文章，读者可以自行去查找合适的方法。

当然，输入的参数要是同一种类型的才可以。否则会返回 TypeError 异常。那么两个列表是不是能够比较呢？可以的！只要输入的类型支持"<"运算就可以了。

程序	结果	说明
print(min([3,4],[1,200,300]))	[1,200,300]	返回最小数值的列表

1.3.3 max

max 与 min 类似。只不过 max 是返回最大的一个数值，而 min 是返回最小的。

1.3.4 range([start], stop[, step])

range 产生一个序列，常用于初始化或者循环中。

参数说明:

参数名称	作用
start	起始数值，默认是 0。
stop	结束数值。必须输入。不能为空
step	步长间隔。

1.3.5 迭代器 iter()

迭代器是访问元素集合的一种方式。迭代器对象从集合的第一个元素开始访问，直到所有的元素被访问完结束。迭代器只能往前不会后退。

迭代器提供了一个统一的访问集合的接口。只要是实现了__iter__()或__getitem__()方法的对象，就可以使用迭代器进行访问。

序列：字符串、列表、元组

非序列：字典、集合、文件

自定义类：用户自定义的类实现了__iter__()或__getitem__()方法的对象

1.3.6 map(function, iterable, ...)

map 的功能很强大，主要就是把 iterable 中的每一个元素带入 function 中进行处理，并且返回处理后的 iterable 类型。map 和数学里面的映射很类似，就是把一些元素映射成另外的数值。比如要完成所有的首字母大写，同时其它字母要小写：

程序	结果	说明

def cap(word):
new_word=word[0:1].upper()+word[1:].lower();
return new_word;
 	
word=['heLLo','WORLD','cool']
print(list(map(cap,['heLLo','WORLD','cool']))
	['Hello','World','Cool']	注意 map 返回的是一个 object。

注意：map 返回的是一个 map 对象，而不是输入的参数类型。因此需要做一个显式的转换。比如转换为列表类型等才能打印输出内容，或者自己使用 for 循环，调用 map 对象的 next() 方法去打印输出。

1.3.7 enumerate(sequence,[start=0])

enumerate 内置函数与数学和 Python Enum 类的含有并不一样。enumerate 内置函数的意义在于生成传入参数的索引值。在遍历列表、字典等数据类型的时候，可以方便的使用索引值。这是 enumerate 的典型应用场景。

程序	结果	说明
weekday = ['星期日','星期一','星期二','星期三','星期四','星期五','星期六'] for index,item in enumerate(weekday,10): print(index,item)	10 星期日 11 星期一 12 星期二 13 星期三 14 星期四 15 星期五 16 星期六	start 是索引的起始值。默认是 0。

Python 也有和数学定义一致的枚举类。需要使用 import 引入 enum 模块，并定义继承 Enum 的类。Enum 和 enumerate 内置函数是完全不同的。Enum 的主要作用就是定义有穷序列成员集合，这些成员有名称和数值两个属性。这些成员的名称不能重名，也就是说，不能定义两个‘星期日’在一个 Enum 的类中，尽管我们都非常期待有多个星期日。Enum 中的成员只可以判断是否相等，但是不能比较大小。Enum 中的成员还可以有别名，就是成员的名称不同，但是数值相同。也可以使用@unique 强制不能出现成员有相同的数值，就是不能有别名。

程序	结果	说明

from enum import Enum
class color(Enum):
dog = 1
cat = 2
for item in color:
print(item)
	pets.dog
	pets.cat	

from enum import Enum class pets(Enum): dog = 1 dog = 2	TypeError: Attempted to reuse key: 'dog'	不能出现同名的名称。
from enum import Enum class pets(Enum): dog_alias = 1 dog = 1 cat = 2	pets.dog_alias pets.cat	可以有别名，但是只打印第一个名称。
print(pets.dog == pets.cat) print(pets.dog == pets.dog_alias)	False True	可以比较是否相等
print(pets.dog > pets.cat)	TypeError: '>' not supported between instances of 'pets' and 'pets'	不能比较大小。
from enum import Enum class pets(Enum): @unique dog_alias = 1 dog = 1 cat = 2	SyntaxError: invalid syntax	加入 @unique 之后，成员的名称和数值都必须唯一。

1.3.8 其他

Python 还有其他的内置函数，比如 int、float、str 等。详细的说明见下表。

函数名称	作用	类型
bool(arg)	根据传入的参数的逻辑值创建一个新的布尔值。0,None,空字符串、空列表等都会被认为是 False，其他的情况是 True。	类型转换
int(arg)	根据传入的参数创建一个新的整数	类型转换
float(arg)	根据传入的参数创建一个新的浮点数	类型转换
complex(arg1,[arg2)	根据传入参数创建一个新的复数	类型转换
str(arg)	根据传入参数创建一个字符串。字符串是不可修改的。	类型转换
bytearray(arg)	根据传入的参数创建一个新的字节数组	类型转换
bytes(arg)	根据传入的参数创建一个新的不可变字节数组	类型转换
memoryview	查看参数指定的内存区域。传入的参数要支持缓冲区协议，内置的类型是 bytes 和 bytearray。	类型转换
ord(arg)	返回 Unicode 字符对应的整数	类型转换

chr(arg)	返回整数所对应的 Unicode 字符	类型转换
bin(arg)	将整数转换成 2 进制字符串	类型转换
oct(arg)	将整数转换成 8 进制字符串	类型转换
hex(arg)	将整数转换成 16 进制字符串	类型转换
tuple(arg)	根据传入的参数创建一个新的元组	类型转换
list(arg)	根据传入的参数创建一个新的列表	类型转换
dict(arg)	根据传入的参数创建一个新的字典	类型转换
set(arg)	根据传入的参数创建一个新的集合	类型转换
frozenset(arg)	根据传入的参数创建一个新的不可变集合	类型转换
iter()	根据传入的参数创建一个新的可迭代对象	类型转换
slice()	根据传入的参数创建一个新的切片对象	类型转换
super()	根据传入的参数创建一个新的子类和父类关系的代理对象	类型转换
object()	创建一个新的 object 对象	类型转换
abs()	求输入数值的绝对值	数学运算
divmod()	返回两个数值的商和余数	数学运算
pow()	取两个值的幂运算值，或者与第三个参数的余数。比如 pow(2,3)的返回值是 8。而 pow(2,3,4)的值是 0。	数学运算
round()	对浮点数进行四舍五入求值	数学运算
sum(seq)	对元素类型是数值的可迭代对象中的每个元素求和	数学运算
all	判断可迭代对象的每个元素是否都为 True 值。0，None,[]等都是 False。	序列操作
any	判断可迭代对象的元素是否有为 True 值的元素	序列操作
filter(func,seq)	使用指定函数过滤可迭代对象的元素，并返回一个可迭代对象。	序列操作
next	返回可迭代对象中的下一个元素值	序列操作
reversed	反转序列生成新的可迭代对象	序列操作
sorted	对可迭代对象进行排序，返回一个新的列表	序列操作
zip	聚合传入的每个迭代器中相同位置的元素，返回一个新的元组类型迭代器。zip 的作用非常类似线性代数的矩阵转置。	序列操作
help(arg)	返回对象的帮助信息。比如 help(int)	对象操作
dir(arg)	返回对象或者当前作用域内的属性列表。尝试输入 print(dir(1))。很有趣的结果。	对象操作
id(arg)	返回对象的唯一标识符。对于自定义的变量，返回的就是内存地址。但是有些内置的对象，是固定的数值。	对象操作
hash(arg)	返回对象的哈希值	对象操作
type(arg)	返回对象的类型，或者根据传入的参数创建一个新的类型	对象操作
ascii(arg)	返回对象的可打印表字符串表现方式	对象操作

format	格式化显示	对象操作
vars	返回当前作用域内的局部变量和其值组成的字典，或者返回对象的属性列表	对象操作
isinstance	判断对象是否是类或者类型元组中任意类元素的实例	对象操作
issubclass	判断类是否是另外一个类或者类型元组中任意类元素的子类	对象操作
hasattr	检查对象是否含有属性	对象操作
getattr	获取对象的属性值	对象操作
setattr	设置对象的属性值	对象操作
delattr	删除对象的属性	对象操作
callable	检测对象是否可被调用	对象操作
globals	返回当前作用域内的全局变量和其值组成的字典	变量操作
locals	返回当前作用域内的局部变量和其值组成的字典	变量操作
print	向标准输出对象打印输出	交互操作
input	读取用户输入值	交互操作
open	使用指定的模式和编码打开文件，返回文件读写对象	文件操作
compile	将字符串编译为代码或者 AST 对象，使之能够通过 exec 语句来执行或者 eval 进行求值	编译执行
eval	执行动态表达式求值	编译执行
exec	执行动态语句块	编译执行
repr	返回一个对象的字符串表现形式(给解释器)	编译执行
property	标示属性的装饰器	装饰器
classmethod	标示方法为类方法的装饰器	装饰器
staticmethod	标示方法为静态方法的装饰器	装饰器

注 1：字符串声明后，是不能够修改的。如果要修改可以使用将字符串定义为字节数组 bytearray。
注 2：byte 声明后，也是不能修改的。如果要修改，见注 1。
注 3：memoryview 其实就是直接访问一块内存区域。这个区域的存放的类型是字节或者字节数组。memoryview 函数可以让程序员直接操作这片内存区域。bytes 使用的内存区域是不可修改的，bytearray 使用的内存区域是可以修改的。

1.4 类

Python 是一种面向对象的语言。对象的实现是通过类 class 来实现的。Python 中的类支持继承、多态等。

程序语言里面的对象，就是现实世界物体的抽象。类拥有成员和方法或者函数。类可以被继承，相当于定制化开发。一个子类也可以继承多个父类，比如一个汽车的类，可以继承发动机和轮胎两个父类。

对象：通过类定义的一种结构。对象包括两个数据成员（类变量和实例变量）和方法。
Python 内置类属性：

属性	说明
__dict__	类的属性（包含一个字典，由类的数据属性组成）
__doc__	类的文档字符串
__name__	类名
__module__	类定义所在的模块（类的全名是'__main__.className'，如果类位于一个导入模块 mymod 中，那么 className.__module__ 等于 mymod）
__bases__	类的所有父类构成元素（包含了一个由所有父类组成的元组）

继承：即一个派生类（derived class）继承基类（base class）的字段和方法。继承也允许把一个派生类的对象作为一个基类对象对待。

方法重写：如果从父类继承的方法不能满足子类的需求，可以对其进行改写，这个过程叫方法的覆盖（override），也称为方法的重写。从后面的例子可以看出，类的方法都是单独的地址。

实例化：创建一个类的实例，类的具体对象。

类变量：类变量在整个实例化的对象中是公用的，是在类内的全局静态变量。类变量定义在类中且在函数体之外。类变量通常不作为实例变量使用。
实例变量：定义在方法中的变量，只作用于当前实例的类。这和类变量是有区别的。改变实例的类的变量，只影响这个这个实例。深入研究一下 Python 的类变量与类方法(类函数)是非常有必要的。我们定义一个类 base_class。这个类中，我们定义基本的 Python 数据类型，包括整型、浮点型、字符串、布尔值，列表、字典与集合。一个变量，不管是在类里面还是不是，它的值如何变化是与它的内存地址有关的。只有不同的实例，影响了同一个地址，才会造成混乱。通俗的说法就是，不同的类的实例使用了相同的内存地址，这个时候，改变一个类的实例变量也会影响到另外一个类的实例。在这个例子中，我们就定义两个 base_class 的实例 a、b 和 c。

程序	结果	说明

```
class base_class():
    class_int = 10
    class_float= 2.0
    class_bool = False
    class_str = ""

    class_list = []
    class_dict = {}
    class_set = set()

    def _init_(self):
        pass

    def set_int(self,i):
        self.class_int = i

    def set_list(self,arg):
        self.class_list.append(arg)

    def set_dict(self,key,value):
        self.class_dict[key] = value
    def set_set(self,value):
        self.class_set.add(value)
    def set_float(self,value):
        self.class_float=value
    def set_str(self,value):
        self.class_str = value
    def set_bool(self,value):
        self.class_bool = value
    def output(self):
        print("int:",self.class_int)
        print("float:",self.class_float)
        print("bool:",self.class_bool)
        print("str:",self.class_str)
        print("list:",self.class_list)
        print("dict:",self.class_dict)
        print("set:",self.class_set)


if _name_ == "_main_":
    a = base_class()
    b = base_class()
    c = base_class()

    a.set_int(1)
    b.set_int(2)
    a.set_float(100.1)
    b.set_float(200.2)
    a.set_bool(True)
    b.set_bool(False)
    a.set_str('a class')
    b.set_str('b class')
    a.set_list('a')
    b.set_list('b')
    a.set_dict('a',100)
    b.set_dict('b',200)
    a.set_set('a')
    b.set_set('b')
```

Assistant × Object inspector ×

type @ 0x29D60E6C718

Name	Value ID
class_bool	0x7FFB6DC8B970
class_dict	0x29D616CDBD0
class_float	0x29D5ECDBA08
class_int	0x7FFB6DD0D540
class_list	0x29D60FC5BC8
class_set	0x29D60D3CF28
class_str	0x29D5ECC7CE0
output	0x29D616BAE18
set_bool	0x29D616BAD90
set_dict	0x29D616BAB70
set_float	0x29D616BAC80
set_int	0x29D616BAA60
set_list	0x29D616BAAE8
set_set	0x29D616BABF8
set_str	0x29D616BAD08

类定义。可以看见 Python 为定义的类变量分配了内存空间，也为类方法分配了地址入口。

Assistant × Object inspector ×

int @ 0x7FFB6DD0D540

10

选择 class_Int，可以看见它的值是 10，也就是初始化赋值。

Assistant × Object inspector ×

__main__.base_class @ 0x29D616CCC18

Name	Value ID
class_bool	0x7FFB6DC8B950
class_dict	0x29D616CDBD0
class_float	0x29D5ECDB9D8
class_int	0x7FFB6DD0D420
class_list	0x29D60FC5BC8
class_set	0x29D60D3CF28
class_str	0x29D616BD4C8
output	0x29D6097EBC8
set_bool	0x29D5ED76A88
set_dict	0x29D60BE0D88
set_float	0x29D60C0E088
set_int	0x29D60BE0DC8
set_list	0x29D60BFE548
set_set	0x29D60B785C8
set_str	0x29D60C44788

实例 a 的内存数据。可以看到 bool、int、float、str 的地址与 base_class 是不同的。而 list、dict 和 set 的地址与 base_class 是一致的。也就是说 list、dict 和 set 的类变量与实例变量共享受一个地址。因此对于他们，多个实例变量的赋值会互相产生影响

实例 b 的内存数据。和实例 a 是一样的

print("class a output:") a.output() print("class b output:") b.output() print("class c output:") c.output()	Assistant × Object inspector × __main__.base_class @ 0x29D616CCC50 Name Value ID class_bool 0x7FFB6DC8B970 class_dict 0x29D616CDBD0 class_float 0x29D5ECDBA38 class_int 0x7FFB6DD0D440 class_list 0x29D60FC5BC8 class_set 0x29D60D3CF28 class_str 0x29D616BD500 output 0x29D61180188 set_bool 0x29D61180848 set_dict 0x29D609A3A88 set_float 0x29D6110DAC8 set_int 0x29D610AC608 set_list 0x29D61133508 set_set 0x29D61180808 set_str 0x29D61180C88 Assistant × Object inspector × __main__.base_class @ 0x29D616CCC88 Name Value ID class_bool 0x7FFB6DC8B970 class_dict 0x29D616CDBD0 class_float 0x29D5ECDBA08 class_int 0x7FFB6DD0D540 class_list 0x29D60FC5BC8 class_set 0x29D60D3CF28 class_str 0x29D5ECC7CE0 output 0x29D60977508 set_bool 0x29D61182908 set_dict 0x29D61180F88 set_float 0x29D61180A48 set_int 0x29D61180AC8 set_list 0x29D61180C48 set_set 0x29D611828C8 set_str 0x29D61182608	实例 c 的。和 base_class 的内存地址一样的。

	class a output:
	int: 1
	float: 100.1
	bool: True
	str: a class
	list: ['a', 'b']
	dict: {'a': 100, 'b': 200}
	set: {'a', 'b'}
	class b output:
	int: 2
	float: 200.2
	bool: False
	str: b class
	list: ['a', 'b']
	dict: {'a': 100, 'b': 200}
	set: {'a', 'b'}
	class c output:
	int: 10
	float: 2.0
	bool: False
	str:
	list: ['a', 'b']
	dict: {'a': 100, 'b': 200}
		c 就是使用的类变量的值。
		这告诉我们，类变量最好用作类的全局常量为好，可以节约空间。

	set: {'b', 'a'}	

那么如果要在类中使用列表等数据怎么办？在__init__()中，初始化一下就可以了。改造一下上面例子中的代码：

程序	结果	说明

def _init_(self,l=None,d=None,s=None):
if l:
self.class_list=l
else:
self.class_list=[]
if d:
self.class_dict=d
else:
self.class_dict={}
if s:
self.class_set=s
else:
self.class_set=set()
	Assistant × Object inspector ×
	type @ 0x207EFB1A558
	Name / Value ID
	class_bool 0x7FFB6DC8B970
	class_dict 0x207F03A0828
	class_float 0x207ED87BA08
	class_int 0x7FFB6DD0D540
	class_list 0x207EFC67A08
	class_set 0x207EF9DCF28
	class_str 0x207ED867CE0
	output 0x207F038AE18
	set_bool 0x207F038AD90
	set_dict 0x207F038AB70
	set_float 0x207F038AC80
	set_int 0x207F038AA60
	set_list 0x207F038AAE8
	set_set 0x207F038ABF8
	set_str 0x207F038AD08
		base_class 类的内存

	Assistant × Object inspector × __main__.base_class @ 0x207F039F5F8 Name / Value ID class_bool 0x7FFB6DC8B950 class_dict 0x207F03A0798 class_float 0x207ED87B9D8 class_int 0x7FFB6DD0D420 class_list 0x207F039E488 class_set 0x207F036BC88 class_str 0x207F038D4C8 output 0x207EF8C4548 set_bool 0x207EFCBBCC8 set_dict 0x207EF87E4C8 set_float 0x207EF7ED808 set_int 0x207EDB438C8 set_list 0x207EFCCC748 set_set 0x207EFCCC208 set_str 0x207EF888F48	实例 a 的内存

	Assistant × Object inspector × __main__.base_class @ 0x207F039F630 Name / Value ID class_bool 0x7FFB6DC8B970 class_dict 0x207F03A0708 class_float 0x207ED87BA38 class_int 0x7FFB6DD0D440 class_list 0x207F039E4C8 class_set 0x207F036BD68 class_str 0x207F038D500 output 0x207EFCCC808 set_bool 0x207EFE3EFC8 set_dict 0x207EFE3EF88 set_float 0x207EFCCC188 set_int 0x207EFE3EF48 set_list 0x207EFE51048 set_set 0x207EFE51088 set_str 0x207EFE510C8	实例 b 的内存（class_bool 的内存与 base_class 一致。是因为 class_bool 的值一样）
	Assistant × Object inspector × __main__.base_class @ 0x207F039F668 Name / Value ID class_bool 0x7FFB6DC8B970 class_dict 0x207F03A0678 class_float 0x207ED87BA08 class_int 0x7FFB6DD0D540 class_list 0x207F039EA88 class_set 0x207F036BAC8 class_str 0x207ED867CE0 output 0x207EFE51A48 set_bool 0x207EFE519C8 set_dict 0x207EFE51788 set_float 0x207EFE517C8 set_int 0x207EFE51C08 set_list 0x207EFE51F08 set_set 0x207EFE51BC8 set_str 0x207EFE51F88	实例 c 的内存

	class a output:
	int: 1
	float: 100.1
	bool: True
	str: a class
	list: ['a']
	dict: {'a': 100}
	set: {'a'}
	class b output:
	int: 2
	float: 200.2
	bool: False
	str: b class
	list: ['b']
	dict: {'b': 200}
	set: {'b'}
	class c output:
	int: 10
	float: 2.0
	bool: False
	str:
	list: []
	dict: {}
	set: set()
		结果：
		a 和 b 的值，现在不一样