Python从入门到实践

第一部分　基础知识

2　变量和简单数据类型

2.1　变量

（1） 命名规则

- 变量名只包含字母、数字和下划线；
- 变量名不能包含空格；
- 不要将Python关键字和函数名用作变量名；
- 变量名应既简短又具有描述性（见名知意）；
- 慎用小写字母l和大写字母O，因为它们可能被错看成1和0；
- **注意**：就目前而言，应使用小写的Python变量名；

2.2　字符串

> **注**：在Python中，用引号括起的都是字符串，其中引号可以是单引号，也可以是双引号（如果子字符串中含有单引号，那么必须得用双引号将其括起，否则会报错）。

（1） title()/upper()/lower()

- 示例代码

```
name = "black magic"
print(name.title())
print(name.upper())
print(name.lower())
```

- 输出结果

```
Black Magic
BLACK MAGIC
black magic
```

（2） + 除用于四则运算以外，还可以用于字符串连接

- 示例代码

```
first_name = "black"
last_name = "magic"
full_name = first_name + " " + last_name
print("My name is " + full_name.title() + "!")
```

- 输出结果

```
'My name is Black Magic!'
```

（3） 空格/制表符(\t)/换行符(\n)

- 示例代码

```
print("eight\tA\nseven\tB\nthree\tC\nthree D")
```

- 输出结果

```
'eight   A'
'seven   B'
'three   C'
'three D'
```

（4） rstrip()/lstrip()/strip() 去除字符串的空格

> **注**：方法执行后不改变变量的值，除非对变量重新赋值，其值才会改变。

- 示例代码

```
test_str = " Python "
print(test_str)              # 输出" Python "
print(test_str.rstrip())     # 输出" Python"
print(test_str.lstrip())     # 输出"Python "
print(test_str.strip())      # 输出"Python"
```

（5） 字符串的格式化

> **注**：字符串格式化在某些场合或许会很实用，以下仅列出一些常用的格式化方式，更多格式化内容请参考Python字符串格式化

```
# 按位置访问参数
str_pos_format = "{0}, {1}, {2}".format("a", "b", "c")
print(str_pos_format)  # 输出"a, b, c"

str_pos_format = "{1}, {0}, {2}".format("a", "b", "c")
print(str_pos_format)  # 输出"b, a, c"

str_pos_format = "{2}, {1}, {0}".format("a", "b", "c")
print(str_pos_format)  # 输出"c, b, a"

# 按名称访问参数
str_name_format = "Name: {first} {second}".format(first="Black", second="Magic")
print(str_name_format)  # 输出"Name: Black Magic"

# 访问参数项
coord = (2, 4)
str_param_format = "X: {0[0]}, Y: {0[1]}".format(coord)
print(str_param_format)  # 输出"X: 2, Y: 4"

# 对齐文本并指定宽度
str_left_aligned = "{:<30}".format("left aligned")
print(str_left_aligned)  # 输出"left aligned                  "

str_right_aligned = "{:>30}".format("right aligned")
print(str_right_aligned)  # 输出"                 right aligned"

str_centered = "{:^30}".format("centered")
print(str_centered)  # 输出"           centered           "

str_centered = "{:*^30}".format("centered")
print(str_centered)  # 输出"**********centered**********"

# 不同进制显示
str_format = "int: {0:d}, hex: {0:x}, oct:{0:o}, bin: {0:b}".format(42)
print(str_format)  # 输出"int: 42, hex: 2a, oct:52, bin: 101010"

str_format = "int: {0:d}, hex: {0:#x}, oct:{0:#o}, bin: {0:#b}".format(42)
print(str_format)  # 输出"int: 42, hex: 0x2a, oct:0o52, bin: 0b101010"

# 使用逗号进行数字千分位分隔
str_num_format = "{:,}".format(1234567890)
print(str_num_format)  # 输出"1,234,567,890"

# 模板字符串
from string import Template

str_template = Template("$who loves $what.")
str_a = str_template.substitute(who="A", what="B")
print(str_a)  # 输出"A loves B."

str_b = str_template.substitute(who="B", what="money")
print(str_b)  # 输出"B loves money."
```

2.3 数字

（1） 浮点数

> **注**：结果包含的小数位可能不确定，其原因是Python为精确的标识结果，但鉴于计算机内部表示数字的方式，有些情况很难，因此出现这种情况。

- 示例代码

```
float_a = 0.1 + 0.1
float_b = 2 * 0.2
float_c = 0.2 + 0.1
float_d = 3 * 0.1
print(float_a)
print(float_b)
print(float_c)
print(float_d)
```

- 输出结果

```
0.2
0.4
0.30000000000000004
0.30000000000000004
```

(2) str() - 避免类型错误

- 示例代码

```
age = 23
meaasge = "Happy " + str(age) + "rd Birthday!"
print(message)
```

- 输出结果

```
Happy 23rd Birthday!
```

> **注**：如果message变量中age不使用str()方法，则无法识别age变量类型，从而报错。

(3) Python2 中的整数

- 示例代码

```
testNum_1 = 3 / 2
testNum_2 = 3.0 / 2
testNum_3 = 3 / 2.0
testNum_4 = 3.0 / 2.0
print(testNum_1)
print(testNum_2)
print(testNum_3)
print(testNum_4)
```

- 输出结果

```
1
1.5
1.5
1.5
```

(4) Python3 中的整数

- 示例代码

```
testNum_1 = 3 / 2
testNum_2 = 3.0 / 2
testNum_3 = 3 / 2.0
testNum_4 = 3.0 / 2.0
print(testNum_1)
print(testNum_2)
print(testNum_3)
print(testNum_4)
```

- 输出结果

```
1.5
1.5
1.5
1.5
```

2.4 注释

(1) 注释标识 -

- 示例代码

```
# 向大家问好 - 注释
print("Hello Everyone!")
```

- 输出结果

```
Hello Everyone!
```

（2） 多行注释标识 - """

- 示例代码

```
"""
这是第一行注释
这是第二行注释
这是第三行注释
...
"""
```

> **注**：三个引号的除了支持多行外，还可以用于说明类或者方法，类似Java中的“/**..**/”或者C#中的“///”。

3 列表

> **列表**：一系列按特定顺序排列的元素组成的集合。

3.1 列表索引(从0开始索引)

- 示例代码

```
fruits = ["apple", "pineapple", "banana", "watermelon", "pear"]
print(fruits[1])
print(fruits[3])
print(fruits[-1])
print(fruits[-2])
```

- 输出结果

```
pineapple
watermelon
pear
watermalon
```

3.2 修改/添加/删除

（1） 修改

- 示例代码

```
fruits = ["apple", "pineapple", "banana", "watermelon", "pear"]
print(fruits)
fruits[0] = "grape"
print(fruits)
```

- 输出结果

```
['apple', 'pineapple', 'banana', 'watermelon', 'pear']
['grape', 'pineapple', 'banana', 'watermelon', 'pear']
```

（2） 添加

- 示例代码

```
fruits = []
print(fruits)
fruits.append("apple")
fruits.append("pineapple")
fruits.append("banana")
print(fruits)
fruits.insert(1, "pear")
print(fruits)
```

- 输出结果

```
[]
['apple', 'pineapple', 'banana']
['apple', 'pear', 'pineapple', 'banana']
```

（3） 删除

- 示例代码

```
# del: 根据元素索引删除元素;
fruits = ["apple", "pineapple", "banana", "pear", "grape"]
print(fruits)
print("del:")
del fruits[2]
print(fruits)

# pop():
# 术语pop源自“栈”: 列表就像一个栈，而删除列表末尾元素相当于弹出栈顶元素;
# 不带索引的pop()弹出并移除列表末尾元素;
# 带索引的pop(index)弹出并移除列表中索引处的元素;
# 调用该方法后可以返回弹出的元素供后续使用;
print("pop()")
fruits = ["apple", "pineapple", "banana", "pear", "grape"]
print(fruits)
fruits_pop = fruits.pop()
print(fruits)
print(fruits_pop)
fruits_pop2 = fruits.pop(1)
print(fruits)
print(fruits_pop2)

# remove(): 根据值删除元素;
print("remove()")
fruits = ["apple", "pineapple", "banana", "pear", "grape"]
print(fruits)
fruits.remove("banana")
print(fruits)
```

- 输出结果

```
['apple', 'pineapple', 'banana', 'pear', 'grape']
del:
['apple', 'pineapple', 'pear', 'grape']
pop():
['apple', 'pineapple', 'banana', 'pear', 'grape']
['apple', 'pineapple', 'banana', 'pear']
grape
['apple', 'banana', 'pear']
pineapple
remove():
['apple', 'pineapple', 'banana', 'pear', 'grape']
['apple', 'pineapple', 'pear', 'grape']
```

（4） 列表的其他操作

- 插入/位置索引/计数/清空

```python
# list.insert(i, x) 在索引位置i处插入元素x
words_list = ["A", "B", "C", "D", "E"]
print(words_list)   # 输出“['A', 'B', 'C', 'D', 'E']”
words_list.insert(2, "F")
print(words_list)   # 输出“['A', 'B', 'F', 'C', 'D', 'E']”

# list.index(x[,stat[,end]])
# 返回列表子序列中从零开始索引且值为x元素在原列表中的位置;
# 如果索引不到则会引发ValueError错误;
fruits_list = ["apple", "banana", "pear", "apple", "banana"]
pos = fruits_list.index("banana", 2, 5)
print(pos)  # 输出“4”

# list.count(x) 返回列表中值为x的元素个数
fruits_list = ["apple", "banana", "pear", "apple", "banana"]
count = fruits_list.count("apple")
print(count)    # 输出“2”

# list.clear() 清空列表，效果等同于del list[:]
words_list = ["A", "B", "C", "D", "E"]
print(words_list)   # 输出“['A', 'B', 'C', 'D', 'E']”
words_list.clear()
print(words_list)   # 输出“[]”
```

- 列表函数的应用 —— 栈

```python
# append()追加和pop()的使用
stack = [1,2,3,4,5,6]
print(stack)    # 输出“[1, 2, 3, 4, 5, 6]”
stack.append(6)
stack.append(7)
print(stack)    # 输出“[1, 2, 3, 4, 5, 6, 6, 7]”
print(stack.pop())  # 输出7
print(stack.pop())  # 输出6
print(stack)    # 输出“[1, 2, 3, 4, 5, 6]”
```

3.3 组织列表

(1) sort() —— 按一定规则对列表进行永久性排序

> **注**：sort()永久性改变列表元素的排列顺序。

- 示例代码

```python
cars = ["bmw", "audi", "toyota", "subaru"]
cars.sort()
print(cars)
cars.sort(reverse=True)
print(cars)
```

- 输出结果

```
["audi", "bmw", "subaru", "toyota"]
["toyota", "subaru", "bmw", "audi"]
```

(2) sorted() —— 按一定规则对列表进行临时排序

> **注**：sorted()方法对列表进行临时排序，且不影响其原始排列顺序。

- 示例代码

```python
cars = ["bmw", "audi", "toyota", "subaru"]
print("original list:")
print(cars)
print("\nsorted() list:")
print(sorted(cars))
print("\nsorted() list(reverse):")
print(sorted(cars, reverse=True))
```

- 输出结果

```
original list:
['bmw', 'audi', 'toyota', 'subaru']

sorted() list:
['audi', 'bmw', 'subaru', 'toyota']

sorted() list(reverse):
['toyota', 'subaru', 'bmw', 'audi']
```

(3) reverse() -- 反转列表元素的排列顺序

> **注**：reverse()方法永久性改变列表元素排列顺序，且随时可以恢复，只需再次调用reverse()即可。

- 示例代码

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

- 输出结果

```
['bmw', 'audi', 'toyota', 'subaru']
['subaru', 'toyota', 'audi', 'bmw']
['bmw', 'audi', 'toyota', 'subaru']
```

(4) len() -- 获取列表的长度

- 示例代码

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

- 输出结果

```
4
```

3.4 列表索引注意事项

- 列表索引从0开始，而不是1；
- 访问列表最后一个元素，可以使用索引-1；访问倒数第二个，使用索引-2，依次类推；
- 只有当列表元素数量不为0时，-1索引才不会出错；

4 操作列表

4.1 遍历列表

(1) for()循环

> **注**：千万不要遗漏for()语句末尾的冒号“:”。

- 示例代码

```
magicians = ["alice", "david", "caroline"]
for magician in magicians:
    print(magician)
    print(magician.title())
```

- 输出结果

```
alice
Alice
David
caroline
Caroline
```

4.2 缩进错误

- Python根据缩进来判断代码行与前一个代码行的关系，因此在写Python代码时要注意不必要的缩进；

- 位于for语句后面属于循环组成部分的代码一定要缩进，否则会报错；
- 忘记缩进的代码行不属于循环体，因此不会在循环中执行；
- 不必要的缩进可能会引发错误，比如定义一个变量后，下一行print语句进行了缩进，则会报错；

```
# 这种缩进将会导致程序运行错误
str = "Hello"
    print(str)  # 此处缩进不必要，运行后会报“Expecten an indented block”错误
```

- 循环后不必要的缩进一定不能缩进，一旦缩进，循环会将该语句当成循环体的一部分进行操作；

4.3 创建数值列表

(1) range()

> **注**：range(a,b)依次生成大于等于a,小于b的值。

- 示例代码

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

- 输出结果

```
1
2
3
4
```

(2) 使用range()创建数字列表

> **注**：要创建列表可以使用函数list()将range()的结果直接转换为列表。

- 示例代码

```
numbers = list(range(1, 6))
print(numbers)
# 使用range指定步长
numbers = list(range(2, 11, 2))
print(numbers)
squares = []
for value in range(1, 11):
    squares.append(value ** 2)
print(squares)
```

> **注**：代码中的(**)为乘方运算。

- 输出结果

```
[1, 2, 3, 4, 5]
[2, 4, 6, 8, 10]
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

(3) min()/max()/sum()

- 示例代码

```
digits = [1, 2, 3, 4, 5, 6, 7, 8, 9, 0]
print(min(digits))
print(max(digits))
print(sum(digits))
```

- 输出结果

```
0
9
45
```

(4) 列表解析

> **注**：列表解析可以将for循环和元素创建精简为一句话，在阅读别的人代码时可能会遇到，所以要对这种表示方式留个心。

- 示例代码

```
squares = [value**2 for value in range(1, 11)]
print(squares)
```

- 输出结果

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

4.4 使用列表的一部分

（1） 切片

> **切片**：指从列表中截取并访问列表子序列的操作。

- 示例代码

```
players = ["charles", "martina", "michael", "florence", "eli"]
print(players)
print(players[0:3])  # 输出第一个元素起到第四个元素前的元素
print(players[1:4])  # 输出第二个元素起到第五个元素前的元素
print(players[:4])  # 输出列表第五个元素前的所有元素
print(players[2:])  # 输出列表中第三个元素之后的所有元素
print(players[-3:])  # 输出列表中的后三个元素
```

- 输出结果

```
['charles', 'martina', 'michael', 'florence', 'eli']
['charles', 'martina', 'michael']
['martina', 'michael', 'florence']
['charles', 'martina', 'michael', 'florence']
['michael', 'florence', 'eli']
['michael', 'florence', 'eli']
```

（2） 遍历切片

- 示例代码

```
players = ["charles", "martina", "michael", "florence", "eli"]
print("Here are the first three players on my team:")
for player in players[:3]:
    print(player)
```

- 输出结果

```
Here are the first three players on my team:
charles
martina
michael
```

（3） 复制列表

- 示例代码

```
my_foods = ["pizza", "falafel", "carrot cake"]
friend_foods = my_foods[:]
# 也可以以这种方式复制:
# friend_foods = my_foods.copy()
print("My favorite foods are:")
print(my_foods)
print("\nMy friend's favorites foods are:")
print(friend_foods)
```

- 输出结果

```
My favorite foods are:
['pizza', 'falafel', 'carrot cake']

My friend's favorites foods are:
['pizza', 'falafel', 'carrot cake']
```

> **注**：在复制列表时，以上述代码为例，如果使用“friend_foods=my_foods”进行复制，其实质并不是复制操作，而是让变量friend_foods和变量my_foods都指向了同一个列表，复制操作“friend_foods=my_foods[:]”是生成了两个列表，即变量friend_foods和my_foods指向的是不同两个列表，但是列表内容相同。

4.5 元组

> **注**：元组与列表类似，不过列表用的是方括号[],元组用的是圆括号，且元组中的元素不能修改。

（1） 定义元组

- 示例代码

```
dimentions=(200, 50)  # 使用圆括号是元组，元组元素无法被修改
print(dimentions[0])
print(dimentions[1])  # dimentions[0]=250 元组元素无法修改，这样的赋值语句会报错
```

- 输出结果

```
200
50
```

（2） 遍历元组

- 和列表一样，可以使用for循环来进行遍历。
  - 示例代码

```
dimentions = (200, 400)
for dimention in dimentions:
    print(dimention)
```

  - 输出结果

    200
    400

（3） 修改元组变量

> **注**：虽然元组元素不能修改，但可以给存储元组元素的变量赋值。

- 示例代码

```
dimentions = (200, 50)
print("Orginal Dimentions:")
print(dimentions)
dimentions = (400, 100)
print("\nModified Dimentions:")
print(dimentions)
```

- 输出结果

```
Orginal Dimentions:
(200, 50)

Modified Dimentions:
(400, 100)
```

4.6 设置代码格式

> **注**：常用的格式方案为“PEP 8”。

- 缩进：PEP 8建议每级缩进4个空格；
- 行长：PEP 8建议每行不超过80个字符，注释行长不超过72个字符；不过该设置有的规定是每行不超过99字符，具体多少视情况而定；
- 空行：用来区分代码块，让程序看上去更直观；

5 if语句

5.2 条件测试

- 检查变量的值是否相等，相等返回True，否则返回False；
- 检查变量值是否相等时默认区分大小写，两个大小写不同的值会被视为不相等，如果不考虑大小写，可利用lower()或者upper()将变更量得值转变为全小写或者全大写进行对应得比较；
- 检查不相等和其他语言一样，使用“!=”；
- 数值型比较和其他语言相同；
- 检查多个条件时：

- 如果是“且”，使用and；

```
age_0 = 22
age_1 = 18
a = age_0 >= 21 and age_1 >=21  # a = False
b = age_0 >= 18 and age_1 >=18  # b = True
```

- 如果是“或”，使用or；

```
age_0 = 22
age_1 = 18
a = age_0 >= 21 or age_1 >=21  # a = True
b = age_0 < 18 and age_1 <18   # b = False
```

- 检查特定值是否包含在列表内，使用in；

```
pets = ["dog", "cat", "rabbit"]
a = "lion" in pets  # a = False
```

- 检查特定值是否不包含在列表内，使用not in；

```
pets = ["dog", "cat", "rabbit"]
b = "lion" not in pets  # b = True
```

- Python含有布尔表达式，布尔表达式的结果要么是True，要么是False；

```
can_speek = True
can_fly = False
```

5.3 if语句

> **注**：在if-elif-else结构中，最后一个else如果不需要可以省略。

- 示例代码

```
grades = [43, 65, 93, 77, 22, 84]
for grade in grades:
    if grade >= 90:
        print("A")
    elif grade >= 80:
        print("B")
    elif grade >= 70:
        print("C")
    elif grade >= 60:
        print("D")
    else:
        print("E")
```

- 输出结果

```
E
D
A
C
E
B
```

6 字典

> **字典**：另一种可变容器模型，且可存储任意类型对象；存储的元素是一系列对象极其值构成的键-值对。

6.2 使用字典

- 示例代码

```python
# 字典的新建有两种方式，可以开始赋值变量时初始化赋值，也可以后续新增
# 其中新建一个空字典语句：
#   alien_0 = {}
alien_0 = {
    "color": "green",
    "points": 5
}
print(alien_0)  # 输出为{'color': 'green', 'points': 5}

alien_0["speed"] = "fast"  # 新增键为“speed”，值为“fast”的键-值对
print(alien_0)  # 输出为{'color': 'green', 'points': 5, 'speed': 'fast'}

alien_0["speed"] = "medium"  # 修改字典中的值
print(alien_0["speed"])  # 输出为medium

del alien_0["speed"]    # 删除键-值对，删除后键-值对永久消失
print(alien_0)  # 输出{'color': 'green', 'points': 5}
```

- 输出结果

```
{'color': 'green', 'points': 5}
{'color': 'green', 'points': 5, 'speed': 'fast'}
medium
{'color': 'green', 'points': 5}
```

6.3 遍历字典

- 示例代码

```python
favorite_languages = {
    "jen":"python",
    "sarah":"c",
    "edward":"ruby",
    "phil":"python"
}

# 遍历字典中所有的键-值对
for name,language in favorite_languages.items():
    print(name.title() + "'s favorite language is" + language.title() + ".")
print("-----------------------------------------------")

# 遍历字典中的所有键
for name in favorite_languages.keys():
    print(name.title())
print("-----------------------------------------------")

# 按顺序遍历字典中的所有键
# 注：字典虽然明确记录键-值的关联关系，但获取字典元素时，默认顺序不可预测
for name in sorted(favorite_languages.keys()):
    print(name.title())
print("-----------------------------------------------")

# 遍历字典中所有的值
for language in favorite_languages.values():
    print(language.title())
```

- 输出结果

```
Jen's favorite language isPython.
Sarah's favorite language isC.
Edward's favorite language isRuby.
Phil's favorite language isPython.
-----------------------------------------
Jen
Sarah
Edward
Phil
-----------------------------------------
Edward
Jen
Phil
Sarah
-----------------------------------------
Python
C
Ruby
Python
```

6.4 嵌套

- 示例代码

```python
# 字典列表
alien_1 = {"color": "green", "point": 5}
alien_2 = {"color": "yellow", "point": 10}
alien_3 = {"color": "yellow", "point": 15}

aliens = [alien_1, alien_2, alien_3]

for alien in aliens:
    print(alien)
print("-----------------------------------------")

# 批量生成多个字典列表
aliens = []

for alien_number in range(0, 30):
    if alien_number % 3 == 0:
        new_alien = {"color": "green", "points": 5, "speed": "slow"}
    elif alien_number % 3 == 1:
        new_alien = {"color": "yellow", "points": 10, "speed": "medium"}
    else:
        new_alien = {"color": "red", "points": 15, "speed": "fast"}
    aliens.append(new_alien)

for alien in aliens[:5]:
    print(alien)
print("...")

print("Total number of aliens: " + str(len(aliens)))
print("-----------------------------------------")

# 在字典中存储列表
favorite_languages = {
    "jen": ["python", "ruby"],
    "sarah": ["c"],
    "edward": ["ruby", "go"],
    "phil": ["python", "haskell"]
}

for name, languages in favorite_languages.items():
    print("\n" + name.title() + "'s favorite language are:")
    for language in languages:
        print("\t" + language.title())
print("-----------------------------------------")

# 在字典中存储字典
users = {
    "aeinstein": {
        "first": "albert",
        "last": "aninstein",
        "location": "princeton"
    },
    "mcurie": {
        "first": "marie",
        "last": "curie",
        "location": "paris"
    }
}

for username,user_info in users.items():
    print("\nUsername: " + username)
    full_name = user_info["first"] + " " + user_info["last"]
    location = user_info["location"]

    print("\tFull name: " + full_name.title())
    print("\tLocation: " + location.title())
```

- 输出结果

```
{'color': 'green', 'point': 5}
{'color': 'yellow', 'point': 10}
{'color': 'yellow', 'point': 15}
----------------------------------------
{'color': 'green', 'points': 5, 'speed': 'slow'}
{'color': 'yellow', 'points': 10, 'speed': 'medium'}
{'color': 'red', 'points': 15, 'speed': 'fast'}
{'color': 'green', 'points': 5, 'speed': 'slow'}
{'color': 'yellow', 'points': 10, 'speed': 'medium'}
...
Total number of aliens: 30
----------------------------------------

Jen's favorite language are:
    Python
    Ruby

Sarah's favorite language are:
    C

Edward's favorite language are:
    Ruby
    Go

Phil's favorite language are:
    Python
    Haskell
----------------------------------------

Username: aeinstein
    Full name: Albert Aninstein
    Location: Princeton

Username: mcurie
    Full name: Marie Curie
    Location: Paris
```

7 用户输入和while循环

7.1 input()

- 示例代码

```python
# input()读取用户键入内容字符串
name = input("Please input your name: ")    # 用户键入Eric
print("Hello, " + name + "!")   # 输出Hello, Eric!

# int()/float()获取数值输入
age = input("How old are you? ")    # 获取用户键入数字字符串
age = int(age)  # 转换数字字符串为整型数值类型(float()转为浮点型数值)
if age < 18:
    print(str(age) + ". I'm still a child.")
else:
    print(str(age) + ". I'm a adult.")

# 取模(符号相同，取模和取余结果一样，但是如果异号，结果不同)
# 取余与取模的区别：取余数是向0舍入，取模是向负无穷方向舍入
mod = 4 % 3 # 输出1
print(mod)
mod = -1 % 3    # 输出2
print(mod)# input()读取用户键入内容字符串
```

- 输出结果

```
Please input your name: Eric
Hello, Eric!
How old are you? 21
1.  I'm a adult.
1
2
```

7.2 while循环

- 示例代码

```python
# 本例中主要为while循环的使用，注意其中break和continue的用法；
# break和continue也可用于for循环；
# break作用：跳出一层循环，多层循环嵌套，break只推出一层循环；
# continue作用：在一次循环中，忽略本次循环中余下的代码，直接进入下一次循环；

prompt = "\nPlease enter the name of a city you have visited: "
prompt += "\n(Enter 'quit' when you are finished.)"
active = True  # 标志变量，可以用于判断程序是否继续运行，使程序更加直观
while active:
    city = input(prompt)

    if city == "quit":
        break  # break 用于跳出一层循环
    else:
        print("I'd love to go to " + city.title() + "!")
print("-------------------------------------------")

current_number = 0
while current_number < 10:
    current_number += 1
    if current_number % 2 == 0:
        continue    # 跳过余下代码并退出本次循环

    print(current_number)
```

- 输出结果

```
Please enter the name of a city you have visited:
(Enter 'quit' when you are finished.)New York
I'd love to go to New York!

Please enter the name of a city you have visited:
(Enter 'quit' when you are finished.)Tokyo
I'd love to go to Tokyo!

Please enter the name of a city you have visited:
(Enter 'quit' when you are finished.)quit
-------------------------------------------
1
3
5
7
9
```

7.3 使用while循环来处理列表和字典

（1）处理列表

- 示例代码

```python
# 注意本例中对各个知识的应用以及while和for的对比

unconfirmed_users = ["alice", "brian", "candace"]
confirmed_users = []

while unconfirmed_users:    # 此处判断列表是否为空，不为空则为True，继续遍历列表
    current_user = unconfirmed_users.pop()  # 此处使用列表的pop()方法弹出列表末尾的元素

    print("Verifying user: " + current_user.title())
    confirmed_users.append(current_user)    # 列表的append()方法追加元素到表尾

print("\nThe following users have been confirmed: ")
for confirmed_user in confirmed_users:   # for循环遍历列表
    print(confirmed_user.title())
print("-------------------------------------------")

pets = ["lion", "dog", "cat", "lion", "goldfish", "rabbit", "lion"]
print(pets)

while "lion" in pets:   # 此处为whie和in的结合使用，判断元素是否在列表中，并移除所有同名元素
    pets.remove("lion")

print(pets)
```

- 输出结果

```
Verifying user: Candace
Verifying user: Brian
Verifying user: Alice

The following users have been confirmed:
Candace
Brian
Alice
--------------------------------------------
['lion', 'dog', 'cat', 'lion', 'goldfish', 'rabbit',   'lion']
['dog', 'cat', 'goldfish', 'rabbit']
```

（2） 处理字典

- 示例代码

```python
# 本例是用户输入、字典和while循环等的结合使用

responses={}

 # 在使用循环时，如果直接利用变量进行判断，有些不明所以 ，所以为使代码看起来更加直观，可以使用布尔变量或者整型变量作为标志进行循环判
polling_active = True

while polling_active:    # 使用布尔变量作为标志进行循环遍历
    name = input("\nWhat is your name? ")
    response = input("Which mountain would you like to climb someday? ")

    responses[name] = response

    repeat = input("Would you like to let another person respond? (yes/no)")
    if repeat == "no":
        polling_active = False

print("\n--- Poll Results ---")
for name, response in responses.items():
    print(name + " would like to climb " + response + ".")
```

- 输出结果

```
What is your name? Eric
Which mountain would you like to climb someday? Denali
Would you like to let another person respond? (yes/no)  yes

What is your name? Lynn
Which mountain would you like to climb someday? Devil's   Thumb
Would you like to let another person respond? (yes/no) no

--- Poll Results ---
Eric would like to climb Denali.
Lynn would like to climb Devil's Thumb.
```

8 函数

> **注**：在学习类是，也会用到函数，而类中的函数叫做方法。

8.1 定义函数

- def关键字指示要定义函数；
- 参数：很多时候我们写函数，都是要对传入的值进行处理，而要传入的值，即为函数的参数；
- 形参：在定义函数时，函数括号中的变量即为形参，用于指示函数要处理的信息；
- 位置实参：在调用函数时输入的参数内容顺序，要和函数定义中指示的参数顺序一致的参数关联方式；
- 关键字实参：为了防止位置实参错误，可以在调用函数时，利用函数形参关键字指明实参对应的内容的参数关联方式；
- Python中函数定义时，可以给某些参数指定默认值；
- 函数的命名最好见名知意，且要添加函数说明，方便维护和使用；

8.2 参数传递

- 示例代码

```python
def describe_pet(pet_name, animal_type="cat"):     # 此处括号中的参数为形参，且其中一个形参指定了默认值
    """一个描述宠物的函数"""
    print("\nI've a " + animal_type + ".")
    print("My " + animal_type + "'s name is " + pet_name.title() + ".")


# 等效调用
describe_pet("tom")      # 此处括号中的为实参
describe_pet(pet_name="tom")

describe_pet("jerry", "hamster")     # 位置实参
describe_pet(pet_name="jerry", animal_type="hamster")    # 关键字实参
describe_pet(animal_type="hamster", pet_name="jerry")    # 使用关键字实参，参数位置不一定要一一对应
```

- 输出结果

```
I've a cat.
My cat's name is Tom.

I've a cat.
My cat's name is Tom.

I've a hamster.
My hamster's name is Jerry.

I've a hamster.
My hamster's name is Jerry.

I've a hamster.
My hamster's name is Jerry.
```

8.3 返回值

- 示例代码

```python
def get_formatted_name1(first_name, last_name,  middle_name=""):
    """格式化姓名函数1"""
    if middle_name:
        full_name = first_name + " " + middle_name + " " + last_name
    else:
        full_name = first_name + " " + last_name
    return full_name.title()    # 使用return可以使函数发返回处理结果


def get_formatted_name2(person):
    """格式化姓名函数2"""
    first_name = person["first"]
    last_name = person["last"]
    middle_name = person["middle"]

    if middle_name:
        full_name = first_name + " " + middle_name + " " + last_name
    else:
        full_name = first_name + " " + last_name
    return full_name.title()


def build_person(first_name, last_name, middle_name=""):
    """造人函数"""
    person = {"first": first_name, "last": last_name, "middle": middle_name}
    return person


musician = build_person("john", "hooker", "lee")
full_name = get_formatted_name2(musician)
print(full_name)

while True:
    print("\nPlease tell me your name: ")
    print("(Enter 'q' at any time to quit.)")

    first_name = input("First name: ")
    if first_name == "q":
        break

    last_name = input("Last name: ")
    if last_name == "q":
        break

    formatted_name = get_formatted_name1(first_name, last_name)
    print("\nHello, " + formatted_name + "!")
```

- 输出结果

```
John Lee Hooker

Please tell me your name:
(Enter 'q' at any time to quit.)
First name: eric
Last name: matthes

Hello, Eric Matthes!

Please tell me your name:
(Enter 'q' at any time to quit.)
First name: q
```

8.4 传递列表

- 示例代码

```python
# 构建函数
def print_models(unprinted_designs, completed_models):
    """打印模型函数"""
    while unprinted_designs:
        current_design = unprinted_designs.pop()

        print("Printing model: " + current_design)
        completed_models.append(current_design)


def show_completed_models(completed_models):
    """展示全部的模型函数"""
    print("\nThe following models have been printed:")
    for completed_model in completed_models:
        print(completed_model)


# 主程序开始执行(使用函数可使代码逻辑表达更加直观)
unprinted_designs = ["iphone case", "robot pendent", "dodecahedron"]
completed_models = []

print_models(unprinted_designs, completed_models)
show_completed_modes(completed_models)
print("----------------------------------------------")

# 语句print_models(unprinted_designs,   completed_models)执行后,
# 列表unprinted_designs被清空，但有时候不需要修改原列表，因此可以使用
# 切片表示法[:]创建副本以达到精致函数修改列表的目的，即:
# print_models(unprinted_designs[:], completed_models)

unprinted_designs1 = ["iphone case", "robot pendent", "dodecahedron"]
unprinted_designs2 = ["iphone case", "robot pendent", "dodecahedron"]
completed_models1 = []
completed_models2 = []

print("\nBefore: " + str(unprinted_designs1) + "\n")
print_models(unprinted_designs1, completed_models1)
print("\Later: " + str(unprinted_designs1))
print("----------------------------------------------")

print("\nBefore: " + str(unprinted_designs2) + "\n")
print_models(unprinted_designs2[:], completed_models2)
print("\nLater: " + str(unprinted_designs2))
```

- 输出结果

```
Printing model: dodecahedron
Printing model: robot pendent
Printing model: iphone case

The following models have been printed:
dodecahedron
robot pendent
iphone case
----------------------------------------------

Before: ['iphone case', 'robot pendent', 'dodecahedron']

Printing model: dodecahedron
Printing model: robot pendent
Printing model: iphone case

Later: []
----------------------------------------------

Before: ['iphone case', 'robot pendent', 'dodecahedron']

Printing model: dodecahedron
Printing model: robot pendent
Printing model: iphone case

Later: ['iphone case', 'robot pendent', 'dodecahedron']
```

8.5 传递任意数量的实参

(1) 任意数量的实参

- 任意数量实参表示方法是在形参前面加一个“*”；
- 任意数量的形参将传入函数的实参存入一个元组；
- 示例代码

```
def make_pizza(*toppings):
    """制作披萨函数"""
    print(toppings)


make_pizza(16, "pepperoni")
make_pizza(12, "mushrooms", "green peppers", "extra cheese")
```

- 输出结果

```
(16, 'pepperoni')
(12, 'mushrooms', 'green peppers', 'extra cheese')
```

（2） 结合使用位置实参和任意数量实参

- 位置实参在函数中的位置必须在任意数量实参前面；
- 示例代码

```
def make_pizza(size, *toppings):
    """制作披萨函数"""
    print("\nMaking a " + str(size) +
          "-inch pizza with the following topping:")
    for topping in toppings:
        print("- " + topping)


make_pizza(16, "pepperoni")
make_pizza(12, "mushrooms", "green peppers", "extra cheese")
```

- 输出结果

```
Making a 16-inch pizza with the following topping:
- pepperoni

Making a 12-inch pizza with the following topping:
- mushrooms
- green peppers
- extra cheese
```

（3） 使用任意数量的关键字实参

- 任意数量的关键字实参使用“**”；
- 任意数量的关键字实参是将名称和值存入一个字典；
- 示例代码

```
def build_person(first_name, last_name, **user_info):
    """造人函数"""
    profile = {}
    profile["first_name"] = first_name
    profile["last_name"] = last_name
    for key, value in user_info.items():
        profile[key] = value
    return profile


user_profile = build_person("albert", "einstein",
                            location="princeton",
                            field="phsics")
print(user_profile)
```

- 输出结果

```
{'first_name': 'albert', 'last_name': 'einstein', 'location': 'princeton', 'field': 'phsics'}
```

8.6 将函数存储在模块中

- 函数的模块化即将处理某一类问题的函数统一放在一个独立的文件中，从而在主程序中导入该模块文件进行调用；
- 导入模块：如果模块文件名称为pizza(pizza.py文件名)，则在另一个文件(如make_pizza.py)中调用时使用import pizza导入模块；

- 导入模块方式：
  - 方式一：导入整个模块

```
import modle_name
```

  - 方式二：导入特定的函数

```
from moudle_name import function_name
```

  - 方式三：使用as给函数指定别名

```
from moudle_name import function_name as new_name
```

  - 方式四：使用as给模块指定别名

```
import moudle_name as new_name
```

  - 方式五：导入模块中所有的函数

```
from moudle_name import *
```

- 示例代码

```
# pizza.py文件，即pizza模块
def make_pizza(size, *toppings):
    """制作披萨函数"""
    print("\nMaking a " + str(size) +
          "-inch pizza with the following toppings:")
    for topping in toppings:
        print("- " + topping)


def say_hello():
    print("Hello, I'm here!")
```

```
# making_pizzas.py文件
import pizza
from pizza import make_pizza
from pizza import make_pizza as mp
import pizza as p
from pizza import *

print("Mode A: import pizza")
pizza.make_pizza(16, "pepperoni")
pizza.make_pizza(12, "mushroom", "green peppers", "extra cheeses")
print("----------------------------------------")

print("Mode B: from pizza import make_pizza")
make_pizza(16, "pepperoni")
make_pizza(12, "mushroom", "green peppers", "extra cheeses")
print("----------------------------------------")

print("Mode C: from pizza import make_pizza as mp")
mp(16, "pepperoni")
mp(12, "mushroom", "green peppers", "extra cheeses")
print("----------------------------------------")

print("Mode D: import pizza as p")
p.make_pizza(16, "pepperoni")
p.make_pizza(12, "mushroom", "green peppers", "extra cheeses")
print("----------------------------------------")

print("Mode E: from pizza import *")
say_hello()
```

- 输出结果

```
Mode A: import pizza

Making a 16-inch pizza with the following toppings:
- pepperoni

Making a 12-inch pizza with the following toppings:
- mushroom
- green peppers
- extra cheeses
------------------------------------------
Mode B: from pizza import make_pizza

Making a 16-inch pizza with the following toppings:
- pepperoni

Making a 12-inch pizza with the following toppings:
- mushroom
- green peppers
- extra cheeses
------------------------------------------
Mode C: from pizza import make_pizza as mp

Making a 16-inch pizza with the following toppings:
- pepperoni

Making a 12-inch pizza with the following toppings:
- mushroom
- green peppers
- extra cheeses
------------------------------------------
Mode D: import pizza as p

Making a 16-inch pizza with the following toppings:
- pepperoni

Making a 12-inch pizza with the following toppings:
- mushroom
- green peppers
- extra cheeses
------------------------------------------
Mode E: from pizza import *
Hello, I'm here!
```

8.7 函数编写指南

- 给定形参指定默认值时，等号两边不要有空格：

```python
def function_name(parameter_0, parameter_1="default_bvalue")
```

- 对于函数调用中的关键字实参，也应遵循这种约定：

```python
function_name(value_0, parammeter_1="value")
```

- 由于PEP 8建议代码行长度不超过79字符，因此如果参数过多可按需换行并保持同样的缩进，以是代码整洁美观；

```python
def function_name(
        parameter_0, parameter_1, parameter_2,
        parameter_3, parameter_4, parameter_5):
    function body...
```

9 类

9.1 创建和使用类

- 示例代码

```python
class Dog():  # 此处的括号在最新的Python版本中可以省略
    """这是一个狗类"""

    def __init__(self, name, age):
        """初始化狗的名字name和年龄age"""
        self.name = name
        self.age = age

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

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


my_dog = Dog("willie", 6)
your_dog = Dog("lucy", 3)

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

print("Your dog's name is " + your_dog.name.title() + ".")
print("Your dog is " + str(your_dog.age) + " years old.")
your_dog.roll_over()
```

- 输出结果

```
My dog's name is Willie.
My dog is 6 years old.
Willie is now sitting.

Your dog's name is Lucy.
Your dog is 3 years old.
Lucy rolled over!
```

- 类中的函数常称为方法；
- 类名通常首字母大写；
- 方法__init__()：
  - 该方法中self必不可少，且必须位于其他形参的前面，原因是Python在调用这个方法创建实例时会自动传入实参self；
  - 每个与类相关联的方法调用都自动传递参数self，它是一个指向本身的引用，让实例能够访问类中的属性和方法；
- 以self为前缀的变量都可供类中的所有方法使用；
- 在Python2.7中创建类时，括号内要包含单词object：

```python
class Dog(object):
    --snip--
```

9.2 类和实例的使用

- 示例代码

```python
class Car():
    """这是个车类"""

    def __init__(self, make, model, year):
        """初始化车的属性"""
        self.make = make
        self.model = model
        self.year = year
        self.odometer_reading = 0

    def get_descriptive_name(self):
        """返回车辆描述信息"""
        long_name = str(self.year) + " " + self.make + " " + self.model
        return long_name.title()

    def read_odometer(self):
        """打印一条车里程消息"""
        print("This car has " + str(self.odometer_reading) + " miles on it.")

    def update_odometer(self, mileage):
        """更新并修改车的里程"""
        self.odometer_reading = mileage

    def increment_odometer(self, miles):
        """增加车里程"""
        self.odometer_reading += miles


my_new_car = Car("audi", "a4", 2016)
print(my_new_car.get_descriptive_name())
my_new_car.read_odometer()

# 直接修改实例属性
my_new_car.odometer_reading = 23500
print("\nAfter attribute has been changed:")
my_new_car.read_odometer()

# 通过方法修改实例属性
my_new_car.update_odometer(23600)
print("\nAfter attribute has been updated by function:")
my_new_car.read_odometer()

# 通过方法对实例属性值进行递增
my_new_car.increment_odometer(300)
print("\nAfter attribute has been increased by function:")
my_new_car.read_odometer()
```

- 输出结果

```
2016 Audi A4
This car has 0 miles on it.

After attribute has been changed:
This car has 23500 miles on it.

After attribute has been updated by function:
This car has 23600 miles on it.

After attribute has been increased by function:
This car has 23900 miles on it.2016 Audi A4
This car has 0 miles on it.

After attribute has been changed:
This car has 23500 miles on it.

After attribute has been updated by function:
This car has 23600 miles on it.

After attribute has been increased by function:
This car has 23900 miles on it.
```

9.3 继承

- 继承：一个类继承另一个类后自动获得另一个类的所有属性和方法；原有的类称为父类，新类称为子类；
- 子类在继承其父类的所有属性和方法的基础上，可以定义自己的属性和方法，并可对父类的方法进行重写；
- super()是个特殊的函数，帮助Python讲父类和子类关联起来；

- Python3.7中的继承：

```
class Car():
    """这是个车类"""

    def __init__(self, make, model, year):
        """初始化车的属性"""
        self.make = make
        self.model = model
        self.year = year


class ElectricCar(car):
    """此乃电动车类"""

    def __init__(self, make, model, year):
        """初始化父类属性"""
        super().__init__(make, model, year)
```

- Python2.7中的继承：

```
class Car():
    """这是一个车类"""

    def __init__(self, make, model, year):
        """初始化车类属性"""
        self.make = make
        self.model = model
        self.year = year


class ElectricCar(car):
    """此乃电动车类"""

    def __init__(self, make, model, year):
        """初始化父类属性"""
        super(ElectricCar, self).__init__(make, model, year)
```

- 示例代码

```python
class Car():
    """这是一个车类"""

    def __init__(self, make, model, year):
        """初始化车的属性"""
        self.make = make
        self.model = model
        self.year = year

    def get_descriptive_name(self):
        """返回车的描述信息"""
        long_name = str(self.year) + " " + self.make + " " + self.model
        return long_name.title()

    def fill_gas_tank(self):
        """加满车的油箱"""
        print("This car's gas tank has been filled.")


class Battery():
    """这是一个车的电瓶类"""

    def __init__(self, battery_size=70):
        """初始化电瓶属性"""
        self.battery_size = battery_size

    def describe_battery(self):
        """打印电瓶的描述信息"""
        print("This car has a " + str(self. battery_size) + "-kWh battery.")

    def get_range(self):
        """获取电瓶可以供车跑多远"""
        car_range = 0
        if self.battery_size == 70:
            car_range = 240
        elif self.battery_size == 85:
            car_range = 270
        message = "This car can go approximately " + str(car_range)
        message += " miles on a full charge."
        print(message)


class ElectricCar(Car):
    """此乃电动车类"""

    def __init__(self, make, model, year):
        """初始化电动车属性"""
        super().__init__(make, model, year)  # 初始化父类的属性
        self.battery_size = 70  # 子类特有属性
        self.battery = Battery()  # 子类特有的属性，且该属性为一个实例

    # 子类特有方法
    def describe_battery(self):
        """打印电动车电瓶描述信息"""
        print("This ar has a " + str(self.battery_size) + "-kWh battery.")

    # 重写父类方法
    def fill_gas_tank(self):
        """给车加满油"""
        print("This car doesn't need a gas tank!")


my_tesla = ElectricCar("tesla", "model s", 2016)
print(my_tesla.get_descriptive_name())
print("\nSubclass-specific attribute:")
my_tesla.describe_battery()  # 子类特有的属性

print("\nAfter overwrite:")
my_tesla.fill_gas_tank()  # 子类重写父类的方法

print("\nInstance attribute:")
my_tesla.battery.describe_battery()  # 子类特有的实例属性方法
my_tesla.battery.get_range()  # 子类特有的实例属性方法
```

- 输出结果

```
2016 Tesla Model S

Subclass-specific attribute:
This ar has a 70-kWh battery.

After overwrite:
This car doesn't need a gas tank!

Instance attribute:
This car has a 70-kWh battery.
This car can go approximately 240 miles on a full charge.
```

9.4 导入类

- 导入类和导入函数方式相同;
- 导入类的几种方式:
  - 导入单个类

    ```
    from moudle_name import class_name
    ```

  - 从一个模块中导入多个类

    ```
    from moudle_name import class_name1,class_name2
    ```

  - 导入整个模块

    ```
    import moudle_name
    ```

  - 导入模块中的所有类

    ```
    from moudle_name import *
    ```

  - 使用as给类指定别名

    ```
    from moudle_name import class_name as new_name
    ```

- 示例代码

```
# car.py文件
class Car():
    """这是一个车类"""

    def __init__(self, make, model, year):
        """初始化车的属性"""
        self.make = make
        self.model = model
        self.year = year
        self.odometer_reading = 0

    def get_descriptive_name(self):
        """返回车的描述信息"""
        long_name = str(self.year) + " " + self.model + " " + self.make
        return long_name.title()

    def read_odometer(self):
        """打印车的里程信息"""
        print("This car has " + str(self.odometer_reading) + " miles on it.")

    def update_odometer(self, mileage):
        """更新车的里程"""
        if mileage >= self.odometer_reading:
            self.odometer_reading = mileage
        else:
            print("You can't roll back an odometer!")

    def increment_odometer(self, miles):
        """增加车的里程"""
        self.odometer_reading += miles

    def fill_gas_tank(self):
        """给车加满油"""
        print("This car's gas tank has been filled.")
```

```python
# electric_car.py文件
from car import Car


class Battery():
    """这是一个电瓶类"""

    def __init__(self, battery_size=60):
        """初始化电瓶属性"""
        self.battery_size = battery_size

    def describe_battery(self):
        """打印电瓶描述信息"""
        print("This car has a " + str(self.battery_size) + "-kWh battery.")

    def get_range(self):
        """获取该电瓶供车跑多远"""
        car_range = 0
        if self.battery_size == 70:
            car_range = 240
        elif self.battery_size == 85:
            car_range = 270

        message = "This car can go approximately " + str(car_range)
        message += " miles on a full charge."
        print(message)


class ElectricCar(Car):
    """此乃电动车类"""

    def __init__(self, make, model, year):
        """初始属性"""
        super().__init__(make, model, year)
        self.battery = Battery()

    def fill_gas_tank(self):
        """给车加满油"""
        print("This car doesn't need a gas tank!")
```

```python
# my_car.py文件
import car
from car import Car
from electric_car import ElectricCar as Ec
from electric_car import ElectricCar, Battery
from electric_car import *

my_car = Car("audi", "a4", 2016)
print(my_car.get_descriptive_name())
print("----------------------------------------------")

my_car.odometer_reading = 23
my_car.read_odometer()
print("----------------------------------------------")

my_tesla = ElectricCar("tesla", "model s", 2016)

print(my_tesla.get_descriptive_name())
my_tesla.battery.describe_battery()
my_tesla.battery.get_range()
print("----------------------------------------------")

my_beetle = Ec("volkswagen", "beetle", 2016)
print(my_beetle.get_descriptive_name())
print("----------------------------------------------")

electric_car_battery = Battery()
electric_car_battery.describe_battery()
```

- 输出结果

```
2016 A4 Audi
-------------------------------------------
This car has 23 miles on it.
-------------------------------------------
2016 Model S Tesla
This car has a 60-kWh battery.
This car can go approximately 0 miles on a full charge.
-------------------------------------------
2016 Beetle Volkswagen
-------------------------------------------
This car has a 60-kWh battery.
```

9.5 Python标准库

- Python标准库是一组模块，安装的Python都包含；
- 除了标准库以外，Python开发者社区还有大量各种编程爱好者的开源库可供使用；
- 示例代码

```python
# 字典虽然可以讲信息关联起来，但它不记录键-值对的顺序，
# 如果想要创建字典并记录其录入顺序，可以使用标准库中模块
# collections中的OrderedDict类
from collections import OrderedDict

favorite_languages = OrderedDict()

favorite_languages["jen"] = "python"
favorite_languages["sarah"] = "c"
favorite_languages["edward"] = "ruby"
favorite_languages["phil"] = "python"

for name, language in favorite_languages.items():
    print(name.title() + "'s favorite language is " +
            language.title() + ".")
```

- 输出结果

```
Jen's favorite language is Python.
Sarah's favorite language is C.
Edward's favorite language is Ruby.
Phil's favorite language is Python.
```

9.6 类编码风格

- 类名应采用驼峰命名法，即将类名中每个单词的首字母都大写，而不使用下划线；
- 实例名和模块名都应采用小写格式，且在单词间加上下划线；
- 对于每个类，都应紧跟在类定义后面包含一个文档字符串。这种文档字符串简要地描述类的功能，并遵循编写函数的文档字符串时采用的格式约定。每个模块也都应包含一个文档字符串，对其中的类可用于做什么进行描述；
- 可使用空行来组织代码，但不要滥用。在类中，可使用一个空行来分隔方法；而在模块中，可使用两个空行来分隔类；
- 需要同时导入标准库中的模块和你编写的模块时，先编写导入标准库模块的import语句，再添加一个空行，然后编写导入你自己编写的模块的import语句；
- 在包含多条import语句的程序中，这种做法让人更容易明白程序使用的各个模块都来自何方；

10 文件和异常

10.1 读取文件

- 对于open(file, mode='r', encoding=None)内的部分参数，说明：
  - file参数为待读取的文件；
  - mode为对数据的读取模式，具体参数说明如下(说明内容为官方说明)：
    - 'r': 打开并读取(默认)；
    - 'w': 打开并写入，如果文件存在会先覆盖；
    - 'x': 创建一个新文件打开并写入；
    - 'a': 打开并写入，如果文件存在，将内容追加到文件末尾；
    - 'b': 二进制模式；以二进制方式读取文件；
    - 't': 文本模式(默认)
    - '+': 打开一个磁盘文件用于更新(读取并写入)
    - 'U': 通用换行模式(不建议使用)
  - encoding为读取文件的编码类型；

- 示例代码

```
# 同路径下读取
# with的作用是不再需要文件访问后将其关闭
# 另外，既然有open，那么就有close与其对应，但是close如果在文件还被使用的情况下执行，那么就会报错，所以为了避免这种情况，一般用wi
with open("my_secret.txt") as file_object:
    contents = file_object.read()
    print(contents)

# 其他路径文件读取
# 注：在linux和OS X系统中，文件路径使用“/”，windows系统中使用“\”，但是有时候windows也能读正确识别“/”
file_path = "my_secret/my_secret.txt"
with open(file_path, "r", encoding="UTF-8") as  file_object:
    contents = file_object.read()
    print(contents)

# 逐行读取
filename = "pi_digits.txt"
with open(filename) as file_object:
    for line in file_object:
        print(line.rstrip())  # 此处用rstrip是为了移除行尾的换行符

# 创建一个包含文件各行内容的列表并使用其内容
with open(filename) as file_object:
    lines = file_object.readlines()

pi_string = ""
for line in lines:
    pi_string += line.strip()

print("\n" + pi_string)
print(len(pi_string))

# 读取百万位的大型文件
filename = "pi_million_digits.txt"

with open(filename) as file_object:
    lines = file_object.readlines()

pi_string = ""
for line in lines:
    pi_string += line.strip()

print("\n" + pi_string[:52] + "...")  # 小数位只显示前50位
print(len(pi_string))
```

- 输出结果

```
Who am I?
Where am I from?
Where am I going?

我是谁?
我从哪里来?
我要到哪里去?

3.1415926535
  8979323846
  2643383279

3.14159265358979323846264338327950288419716939937510...
1000002
```

10.2 写入文件

- 示例代码

```
filename = "programming.txt"

# 写入空文件
with open(filename, "w") as file_object:
    file_object.write("I love programming.\n")
    file_object.write("I lave creating new games.\n")

# 附加到文件
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")
filename = "programming.txt"

# 写入空文件
with open(filename, "w") as file_object:
    file_object.write("I love programming.\n")
    file_object.write("I lave creating new games.\n")

# 附加到文件
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")
```

- 结果文件

```
# programming.txt
I love programming.
I lave creating new games.
I also love finding meaning in large datasets.
I love creating apps that can run in a browser.
```

10.3 异常

- 很多时候程序到错误会直接停止并显示异常报告，但很多时候我们需要程序报告错误且程序继续执行，所以我们需要对异常进行处理；
- 程序异常的处理使用try-except代码块以及else代码块；
- 示例代码

```
print("Give me two numbers and I'll divide them.")
print("Enter 'q' to quit.")

while True:
    first_number = input("\nFirst number: ")
    if first_number == "q":
        break
    second_number = input("Second number: ")
    if second_number == "q":
        break
    try:
        answer = int(first_number) / int(second_number)
    except ZeroDivisionError:   # 处理0不能做除数的错误，打印错误提示但代码正常运行
        print("You can't divide by zero!")
    else:
        print(answer)

print("-----------------------------------------------")


def count_words(filename):
    try:
        with open(filename) as file_object:
            contents = file_object.read()
    except FileNotFoundError:   # 处理文件无法找到异常
        msg = "Sorry, the file " + filename + " does not exist."
        print(msg)
    else:
        # split()方法以空格为分隔符将字符串拆分为多个部分
        words = contents.split()
        num_words = len(words)
        print("The file " + filename + " has about " + str(num_words) + " words.")


file_names = ["alice.txt", "siddhartha.txt", "moby_dick.txt", "little_women.txt"]
for file_name in file_names:
    count_words(file_name)
print("-----------------------------------------------")


def count_words2(filename):
    try:
        with open(filename) as file_object:
            contents = file_object.read()
    except FileNotFoundError:
        pass  # pass作用是让程序对错误不做处理，跳过错误继续执行
    else:
        words = contents.split()
        num_words = len(words)
        print("The file " + filename + " has about " + str(num_words) + " words.")


file_names = ["alice.txt", "siddhartha.txt", "moby_dick.txt", "little_women.txt"]
for file_name in file_names:
    count_words2(file_name)
```

- 输出结果

```
Give me two numbers and I'll divide them.
Enter 'q' to quit.

First number: 5
Second number: 0
You can't divide by zero!

First number: 5
Second number: 2
2.5

First number: q
-----------------------------------------
The file alice.txt has about 29461 words.
The file siddhartha.txt has about 42172 words.
Sorry, the file moby_dick.txt does not exist.
The file little_women.txt has about 189079 words.
-----------------------------------------
The file alice.txt has about 29461 words.
The file siddhartha.txt has about 42172 words.
The file little_women.txt has about 189079 words.
```

- split(sep, maxsplit)的使用
  - sep参数为文本拆分字符，按参数指定的字符拆分字符串；
  - maxsplit参数为文本拆分索引拆分字符的最大数量；
  - 示例代码

```python
string = "1,2,3"
print(string.split(","))
# 输出为['1', '2', '3']

string = "1,2,3,4,5"
print(string.split(",", maxsplit=2))
# 输出为['1', '2', '3,4,5']，即列表中只有三个元素"1","2"和  "3,4,5"
```

> **注**：更多用法的学习请移步Python官方帮助文档

10.4 存储数据

- 很多程序要求用户输出某种信息，如让用户存储游戏首选项或提供要可视化的数据，这就涉及到了数据的存储，而模块json可以将简单的Python数据结构转储到json文件中；
- 示例代码

```Python
import json


# 获取保存的用户名称
def get_stored_username():
    filename = "username.json"
    try:
        with open(filename) as file_object:
            username = json.load(file_object)
    except FileNotFoundError:
        return None
    else:
        return username


# 获取新用户名称并保存数据
def get_new_name():
    filename = "username.json"
    username = input("What's your name? ")
    with open(filename, "w") as file_object:
        json.dump(username, file_object)
    return username


# 欢迎用户
def greet_user():
    username = get_stored_username()
    if username:
        print("Welcome back, " + username + "!")
    else:
        username = get_new_name()
        print("We'll remember you when you come back, " + username + "!")


greet_user()
```

- 输出结果

```
# 第一次执行
What's your name? Eric
We'll remember you when you come back, Eric!
```
```Python
# username.json文件
"Eric"

# 第二次执行
Welcome back, Eric!
```

11 测试代码

11.1 测试函数

- 测试代码使用Python标准库中的模块unittest代码测试工具；
- 单元测试：用于核实函数的各方面没有问题；
- 测试用例：一组单元测试，这些单元测试一起核实在各种情形下都符合要求；
- 可通过的测试
  - 示例代码

```Python
# name_function.py
def get_formatted_name(first, last):
    """格式化姓名"""
    full_name = first + " " + last
    return full_name.title()
```

```python
# test_name_function.py
import unittest
from name_function import get_formatted_name


class NamesTestCase(unittest.TestCase):
    """测试name_function.py"""
    def test_first_last_name(self):
        formatted_name1 = get_formatted_name1   ("janis", "joplin")
        self.assertEqual(formatted_name1, "Janis Joplin")


unittest.main() # 运行文件中的测试
```

- 输出结果

```
.
----------------------------------------------
Ran 1 test in 0.000s

OK
```

- 不能通过的测试
  - 示例代码

```python
# name_function.py
def get_formatted_name(first, midle, last):
    """格式化姓名"""
    full_name = first + " " + midle + " " + last
    return full_name.title()
```

```python
# test_name_function.py
import unittest
from name_function import get_formatted_name


class NamesTestCase(unittest.TestCase):
    """测试name_function.py"""
    def test_first_last_name(self):
        formatted_name1 = get_formatted_name   ("janis", "joplin")
        self.assertEqual(formatted_name, "Janis Joplin")


unittest.main() # 运行文件中的测试
```

  - 输出结果

```
E
======================================================
ERROR: test_first_last_name (__main__.NamesTestCase)
------------------------------------------------------
Traceback (most recent call last):
  File "E:\xxx\test_name_function.py", line 10, in     test_first_last_name
    formatted_name = get_formatted_name("janis", "joplin")
TypeError: get_formatted_name() missing 1 required     positional argument: 'last'

------------------------------------------------------
Ran 1 test in 0.000s

FAILED (errors=1)
```

    - 结果说明：错误结果中首先字母“E”指有一个单元导致了错误，后续ERROR指出具体有问题的函数调用，并说明该问题是缺少了一个位置实参“last”；最后两行页可以看出该测试用例中运行的测试数量、用时以及失败的测试数量；
- 测试用例中多个测试
  - 示例代码

```python
# name_function.py
def get_formatted_name(first, last, middle=""):
    """返回格式化的姓名"""

    if middle:
        full_name = first + " " + middle + " " + last
    else:
        full_name = first + " " + last
    return full_name.title()
```

```python
# test_name_function.py
import unittest
from name_function import get_formatted_name


class NamesTestCase(unittest.TestCase):
    """测试name_function.py"""

    def test_first_last_name(self):
        """能够正确地处理像Janis Joplin这样的姓名吗？"""
        formatted_name = get_formatted_name("janis", "joplin")
        self.assertEqual(formatted_name, "Janis Joplin")

    def test_first_last_middle(self):
        """能够正确地处理像Wolfgang Amadeus Mozart这样的姓名吗？"""
        formatted_name = get_formatted_name("wolfgang", "mozart", "amadeus")
        self.assertEqual(formatted_nam, "Wolfgang Amadeus Mozart")


unittest.main()
```

- 输出结果

```
..
----------------------------------------------
Ran 2 tests in 0.000s

OK
```

11.2 测试类

（1） 各种断言方法

> 以下表格为unittest Moudle中的断言方法。

方　法	用　途
assertEqual(a, b)	核实a == b
assertNotEqual(a, b)	核实 a != b
assertTrue(x)	核实x为True
assertFalse(x)	核实x为False
assertIn(item, list)	核实item在list中
assertNotIn(item, list)	核实item不在list中

（2） 测试类

- 新建一个测试类

```python
# survey.py
class AnonymousSurvey():
    """收集匿名调查问卷答案"""

    def __init__(self, question):
        """存储一个问题，并为存储答案做准备"""
        self.question = question
        self.responses = []

    def show_question(self):
        """显示调查问卷"""
        print(self.question)

    def store_response(self, new_response):
        """存储单份调查问卷"""
        self.responses.append(new_response)

    def show_result(self):
        """显示收集到的所有答卷"""
        print("Survey results:")
        for response in self.responses:
            print("- " + response)
```

```python
# language_survey.py
from survey import AnonymousSurvey

# 定义一个问题，并创建一个表示调查的AnonymousSurvey对象
question = "What language did you first learn to speak?"
my_survey = AnonymousSurvey(question)

# 显示问题并存储答案
my_survey.show_question()
print("Enter 'q' at any time to quit.\n")
while True:
    response = input("Language: ")
    if response == "q":
        break
    my_survey.store_response(response)

# 显示调查结果
print("\nThank you to everyone who participated in the survey!")
my_survey.show_result()
```

- 输出结果

```
What language did you first learn to speak?
Enter 'q' at any time to quit.

Language: English
Language: Spanish
Language: Mandarin
Language: English
Language: q

Thank you to everyone who participated in the survey!
Survey results:
- English
- Spanish
- Mandarin
- English
```

- 测试类以及setUp()使用

```python
# test_survey.py
import unittest
from survey import AnonymousSurvey


class TestAnonymousSurvey(unittest.TestCase):
    """针对AnonymousSurvey类的测试"""

    def setUp(self):
        """
        创建一个调查对象和一组答案，供测试方法使用；
        该方法相当于赋值一系列测试类的属性，以对多个接收相同输入的测试用例测试，避免重复定义；
        """
        question = "What language did you first speak?"
        self.my_survey = AnonymousSurvey(question)
        self.responses = ["English", "Spanish", "Mandarin"]

    def test_store_sing_response(self):
        """测试单个答案会被妥善的存储"""
        self.my_survey.store_response(self.responses[0])
        self.assertIn(self.responses[0], self.my_survey.responses)

    def test_store_three_response(self):
        """测试三个答案是否会被妥善存储"""
        for response in self.responses:
            self.my_survey.store_response(response)

        for response in self.responses:
            self.assertIn(response, self.my_survey. responses)


unittest.main()
```

- 输出结果

```
..
----------------------------------------------
Ran 2 tests in 0.000s

OK
```

第二部分 项目

11.12 11与12的间隙

- 更新conda或包：

```
#############################################################

# 更新指令的使用先看这里
重要说明：对于以下的更新指令，在修改源为国内源后，conda update conda可以达到更新所有包的目的，但conda update --all则会让所有

# 更新conda
conda update conda

# 更新Anaconda，试了下如果安装的是anaconda其实和更新conda指令效果相同
conda update anaconda

# 更新所有包
conda update --all
#############################################################

# 添加清华源，其实这种方式添加了清华源就没问题了，百度一下，晚上也基本都是这种方法，但是我个人觉得下面清华源推荐的配置更安逸一点
conda config --add channels https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/free/
conda config --add channels https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/main/
conda config --set show_channel_urls yes

# Ubuntu Anaconda 使用改文件的方式添加清华源
https://mirrors.tuna.tsinghua.edu.cn/help/ubuntu/

#############################################################
```

- Anaconda清华源“.condarc”配置：

```
# 清华源官网指定的配置形式
channels:
  - defaults
show_channel_urls: true
channel_alias: https://mirrors.tuna.tsinghua.edu.cn/anaconda
default_channels:
  - https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/main
  - https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/free
  - https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/r
  - https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/pro
  - https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/msys2
custom_channels:
  conda-forge: https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud
  msys2: https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud
  bioconda: https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud
  menpo: https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud
  pytorch: https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud
  simpleitk: https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud
ssl_verify: true
```

```
# 我自己的配置，配置了default-channel和channel-alias会报一堆警告
channels:
  - https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/main
  - https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/free
  - https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/r
  - https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/pro
  - https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/msys2
  - https://mirrors.ustc.edu.cn/anaconda/pkgs/main/
  - https://mirrors.ustc.edu.cn/anaconda/cloud/conda-forge/
how_channel_urls: true
custom_channels:
  conda-forge: https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud
  msys2: https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud
  bioconda: https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud
  menpo: https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud
  pytorch: https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud
  simpleitk: https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud
ssl_verify: true
```

- Anaconda中科大源：

```
https://mirrors.ustc.edu.cn/anaconda/pkgs/main/
https://mirrors.ustc.edu.cn/anaconda/cloud/conda-forge/
```

- pip国内源：

```
############################################################
阿里云 ：http://mirrors.aliyun.com/pypi/simple/
中国科技大学： https://pypi.mirrors.ustc.edu.cn/simple/
豆瓣(douban) ：http://pypi.douban.com/simple/
清华大学 ：https://pypi.tuna.tsinghua.edu.cn/simple/
中国科学技术大学： http://pypi.mirrors.ustc.edu.cn/simple/

############################################################
# 使用豆瓣源安装
pip install opencv-python -i http://pypi.douban.com/simple
# 报错说不信任该源，执行如下：
pip install opencv-python -i http://pypi.douban.com/simple --trusted-host pypi.douban.com

############################################################
```

- Pycharm安装第三方包会报信任问题，解决方法如下：
  - 需要在options处添加形如这种形式的指令：

    ```
    --trusted-host mirrors.aliyun.com
    ```

  - 还有种办法是在c:\users\xxx\文件夹新建pip文件夹，然后再文件夹中新建pip.ini文本文件，内容如下：

```
注：其中global和install下保留一条即可
[global]
index-url = https://pypi.tuna.tsinghua.edu.cn/simple/
index-url = http://pypi.mirrors.ustc.edu.cn/simple/
index-url = http://mirrors.aliyun.com/pypi/simple/
index-url = https://pypi.doubanio.com/simple/
[install]
trusted-host=pypi.tuna.tsinghua.edu.cn
trusted-host=pypi.mirrors.ustc.edu.cn
trusted-host=mirrors.aliyun.com
trusted-host=pypi.doubanio.com
```

    - Pycharm默认源无法安装或者下载很慢，可以添加以上pip国内源；
- 更新conda包后，可能会遇到之前写的代码在更新后跑不通的问题，这时候可以回退到conda的以前版本；

```
# 查看更新前各种回退版本
conda list --revision

# 回退至某个版本
conda install --rev x
```

- Pycharm中requsests.get(url)报Can't connect to HTTPS URL because the SSL module is not available错误，可能解决方法为：

```
方法一：此链接下载对应版本的ssl，https://slproweb.com/products/Win32OpenSSL.html（这方法我试了下，对我这个情况不管用）；

方法二：如果是使用Anaconda和Pycharm的话，直接切换解释器（settings-project:xxx-project interpreter）为Anaconda就可以

说明：这个错误很神奇，google了半天也没找到个好点的解决办法，这问题似乎是pip安装的opelssl和anaconda安装的openssl冲突导致的
```

- Anaconda安装第三方库：

```
Step1. 解压文件
Step2. cmd切换到文件setup.py的目录
Step3. python setup.py build
Step4. python setup.py install
```

- 清华大学开源镜像站：https://mirrors.tuna.tsinghua.edu.cn/
- 控制台错误警告“This Python interpreter is in a conda environment, but the environment hasnot been activated”解决办法：

```
Step1: activate base
Step2: python
```

- Anaconda3搭建Django

```
1.查看虚拟环境
conda env list
2.创建虚拟环境
conda create -n django
3.删除虚拟环境
conda remove -n django --all
--------------------------------------------------------------------------------
下面是真正的创建步骤：
新建虚拟环境
(base) C:\Users\xxx>conda create -n django
激活虚拟环境
(base) C:\Users\xxx>avtivate django
安装Django
(django) c:\Users\xxx>conda install django
进入项目文件夹路径
(django) C:\Users\xxx>cd /d d:\test\django
创建Django1_prj项目
(django) d:\test\django>django-admin startproject test_proj .
创建数据库
(django) d:\test\django>python manage.py migrate
运行Django1_prj项目服务器
(django) d:\test\django>python manage.py runserver
--------------------------------------------------------------------------------

原文地址：https://blog.csdn.net/wbdxz/article/details/81224319
```

- conda在指定目录下创建虚拟环境，conda使用国内镜像

```
# 在目录E:\trmp\下创建名为test_proj的虚拟环境
conda create --prefix=E:\trmp\test_proj python=3.7.6
conda create -p E:\trmp\test_proj python=3.7.6

# 启动虚拟环境
source activate E:\trmp\test_proj

# 关闭虚拟环境
source deactivate E:\trmp\test_proj

# 删除虚拟环境
conda remove -n assign --all
```

- 关于Djang2.0中的reverse导入失败解决方案：

```
# Django1.0
from django.core.urlresolvers import reverse

# Django2.0
from django.urls import reverse
```

- Django2.0异常：Specifying a namespace in include() without providing an app_name is not supported. 处理如下：

```
# 有异常的代码
urlpatterns = [
    path(r'^admin/', admin.site.urls),
    path(r'', include("learning_logs.urls", namespace="learning_logs")),
]

# 异常处理
urlpatterns = [
    path(r'^admin/', admin.site.urls),
    path(r'', include(("learning_logs.urls", "learning_logs"), namespace="learning_logs")),
]
```

- Djanjo用户账户中users文件夹下的urls.py问题解决方案：TypeError: login() got an unexpected keyword argument 'template_name'

```
# Django版本1.0
from django.contrib.auth import login
url(r'^login/$',login,{'template_name'='users/login.html'},name='login')

# Djanjo版本2.0
from django.conf.urls import url
from django.contrib.auth.views import LoginView
urlpatterns = [
    url(r'^login/$',LoginView.as_view(template_name='users/login.html'),name='login')
]
```