Python基础知识（二）

陈赟

对外经济贸易大学金融学院
yunchen@uibe.edu.cn

2022年10月03日

目录

一、字符串

创建字符串

- 字符串(String, str()函数)是Python中最常用的数据类型之一，Python中可以使用一对单引号''或者双引号""生成字符串：

```
>>> s1 = "hello, world"
>>> s2 = 'hello, world'
>>> s1
'hello, world'
>>> s2
'hello, world'
>>> type(s2)
<class 'str'>
```

字符串的简单操作

- 字符串的加法、乘法、len函数、str函数

```
>>> 'hello ' + 'world'
'hello world'
>>> "echo" * 3
'echoechoecho'
>>> len("abc")
3
>>> str(3.1)
'3.1'
```

- 成员运算符in

```
>>> s1 = "hello, world"
>>> "h" in s1
True
>>> "he" in s1
True
>>> "ha" not in s1
True
```

字符串的简单操作

- 字符串的索引

```
>>> s1 = "hello, world"
>>> s1[3]
'l'
>>> s1[-1]
'd'
```

- Python字符串不能修改（immutable）。因此，为字符串中某个索引位置赋值会报错
- 字符串切片

```
>>> s1[:3]
'hel'
>>> s1[3:]
'lo, world'
>>> s1[2:5]
'llo'
>>> s1[::2]
'hlo ol'
```

字符串的方法

- s.title(): "标题化"的字符串, 输出单词都是以首字母大写，其余字母均为小写。

```
>>> greet = "hello python"
>>> greet.title()
'Hello Python'
```

- s.upper()返回s中字母全部大写的新字符串。s.lower()

```
>>> s = "HELLO WORLD"
>>> s.lower()
'hello world'
>>> "hello world".upper()
'HELLO WORLD'
>>> "hello world".title()
'Hello World'
>>> 'hello world'.capitalize()
'Hello world'
```

字符串的方法

- isdigit、isupper、isspace、isalpha、istitle、isalnum：判断字符串是否满足特定的条件，如果字符串具备特定的性质，则返回True，否则返回False。

```
>>> '3'.isdigit()
True
>>> '3a'.isalnum()
True
>>> '3a'.istitle()
False
```

- s.strip()返回将s两端多余空格除去的新字符串。s.lstrip()、s.rstrip()

```
>>> s = " hello world  "
>>> s.strip()
'hello world'
>>> s.rstrip()
'  hello world'
>>> s.lstrip()
'hello world  '
```

字符串的方法

- s.find(sub_str)返回子串sub_str所在位置的最左端索引，没有找到则返回-1。

```
>>> title = "Monty Python's Flying Circus"
>>> title.find('Monty')
0
>>> title.find('Python')
6
>>> title.find('Word')
-1
```

- s.replace(part1, part2)将字符串s中指定的部分part1替换成想要的部分part2，并返回新的字符串。

```
>>> s = "hello world"
>>> s.replace('world', 'python')
'hello python'
>>> s
'hello world'
```

字符串的方法

- s.split(sep)以给定的sep为分隔符对s进行分割，返回所有分割得到的字符串。sep默认为空格（包括多个空格，制表符\t，换行符\n等）

```
>>> line = "1 2 3 4 5"
>>> numbers = line.split()
>>> numbers
['1', '2', '3', '4', '5']
>>> line = "1,2,3,4,5"
>>> line.split(',')
['1', '2', '3', '4', '5']
```

- s.join(seq)：以s为连接符，将字符串序列seq中的元素连接起来，并返回连接后得到的新字符串：

```
>>> s = ' '
>>> s.join(numbers)
'1 2 3 4 5'
>>> ','.join(numbers)
'1,2,3,4,5'
```

转义字符

- 字符串中可以增加转义字符，用来输出特殊效果，如换行、制表等。
- 一般使用反斜杠(\)，常用的转义字符如下：
  - \n: 换行, 光标移动到下行首
  - \t: 水平制表符
  - \\：表示反斜杠
  - \在行末尾可以表示语句跨行。如果单行源代码太长，可以使用\，新行的开头与首行对齐。

转义字符

- print() 函数会生成可读性更好的输出，它会省去引号并且打印出转义后的特殊字符

```
>>> print('a\\b')
a\b
>>> print("a\"b")
a"b
>>> print('a\'b')
a'b
>>> print('a\tb')
a	b
>>> print('a\nb')
a
b
>>> print('a\
... bcd')
abcd
```

转义字符

- r"": 字符串前加r可使转义字符失效，又称原生字符串

```
>>> s1 = 'a\nb'
>>> s1
'a\nb'
>>> print(s1)
a
b
>>> s2 = r'a\nb'
>>> s2
'a\\nb'
>>> print(s2)
a\nb
```

注释

- 注释是程序员在代码中加入的一行或者多行信息，用来对语句、函数、数据结构、方法等进行说明，提升代码的可读性。
- 注释是辅助性文字，会被解释器省略不被执行
- 用途:
  - 表明作者和版权信息
  - 解释代码原理和用途
  - 辅助程序调试：在调试程序时，可以通过单行或多行注释临时“去掉”与当前调试无关的代码，辅助找到程序发生问题的位置。

注释

Python语言有两种注释方法：

- 单行注释：以 #开头

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
```

- 多行注释：以 '''(三个引号)开头和结尾

```
"""
Created on Mon Feb 28 22:10:26 2022

@author: yunchen
"""
```

注释

- 可以利用三引号可以实现输出单行文本和多行文本，示例如下:

```
>>> a = ''' Hello ,
... world
... python '''
>>> a
'Hello,\nworld\npython'
>>> print(a)
Hello,
world
python
```

- 如果不需要回车换行，在行尾添加\即可

课堂练习- 字符串

- 编程练习：
  - 输入你的邮箱名称，将其存储在变量email中
  - 分别使用索引和split方法来查找邮箱的前缀（用户名）和后缀
  - 针对用户名，转换成小写、大写、首字母大写等
- 将字符串"123456789"变成"147258369"

课堂练习- 字符串

- 某股票交易数据文件名信息存放在一个字符串里面，stk_info = "sz1.csv@wind"。针对该字符串，有以下信息
  - 符号@前面表示数据文件名称，@后面表示数据来源（这里来自wind数据库）
  - 数据文件名称中，csv表示文件类型，前缀sz表示深圳交易所，1表示股票的代码，但实际上股票代码应该是6位数字（000001、600519）。这里可能是因为数据处理中的“误操作”，导致字符串"000001"变成整型1。
- 请编写代码提取以上信息，将股票代码还原成6位数字，并生成文件名SZ_000001.xlsx。（假想你有数千个这样的文件需要处理，你可以尝试搜索下如何将字符串进行填充）

二、字典与集合

字典

- 映射（mapping）：一种可通过名称来访问其各个值的数据结构。字典是Python中唯一的内置映射类型，其中的值不按顺序排列，而是存储在键下。
- 下面代码定义了一个字典：

```
transcript = {"Python": 100, "Econ": 95}
```

  - 字典由键及其相应的值组成，这种键-值对称为项（item）
  - 键与其值之间都用冒号（:）分隔，项之间用逗号分隔，整个字典放在花括号内
  - 字典的键可能是数、字符串或元组，字典的值可以是数、字符串、列表、字典，以及任何Python对象
  - 字典（以及其他映射类型）中，键必须是独一无二的(**不可变对象**)，而字典中的值无需如此

字典与嵌套

- 字典列表

```
>>> scripts = [{"Course": "Econ", "Score": 100}, {"Course": "Python",
    "Score": 90}]
```

- 字典中存储列表

```
menu = {"name": "DiningRoom1", "foods": ["Rice", "Pie", "Pancake"]}
```

- 字典中存储字典

```
students = {"Wang": {"id": 202201, "department": "Finance", "score":
    95},
            "Li": {"id": 202202, "department": "Business", "score":
                100}}
```

基本的字典操作

- 定义空字典

```
>>> dict1 = {}
>>> dict1
{}
>>> dict2 = dict()
>>> dict2
{}
```

- 访问字典中的值：dict_name[key]

```
>>> transcript = {"Python": 100, "Econ": 95}
>>> transcript["Python"]
100
```

基本的字典操作

- 添加键值对：dict_name[newkey] = value

```
>>> transcript["Finance"] = 88
>>> transcript
{'Python': 100, 'Econ': 95, 'Finance': 88}
>>> transcript["Finance"] = 90
>>> transcript
{'Python': 100, 'Econ': 95, 'Finance': 90}
```

- 删除键值对：del dict_name[key]

```
>>> transcript
{'Python': 100, 'Econ': 95, 'Finance': 88}
>>> del transcript["Finance"]
>>> transcript
{'Python': 100, 'Econ': 95}
```

基本的字典操作

- len(d)返回字典d包含的项（键-值对）数

```
>>> transcript = {"Python": 100, "Econ": 95}
>>> len(transcript)
2
```

- k in d检查字典d是否包含键为k的项

```
>>> "Python" in transcript
True
>>> "Macro" in transcript
False
```

字典方法

- get()方法：d[k]方式访问字典中的值时使用的键不存在于字典中会出错，get()方法可避免这种情况

```
>>> transcript
{'Python': 100, 'Econ': 95}
>>> transcript["Finance"]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'Finance'
>>> transcript.get("Finance", "No value assigned")
'No value assigned'
```

  - get()中若未指定第二个参数，且指定的键不存在时，Python将返回值None。注意：None并非错误，只是一个表示所需值不存在的特殊值。

```
>>> transcript.get("Finance")
>>> print(transcript.get("Finance"))
None
>>> transcript.get("Python")
100
```

字典方法

- items()：返回一个包含所有字典项的列表，其中每个元素都为(key, value)的形式。返回值属于一种名为字典视图的特殊类型，它可用于迭代
- keys()：返回一个由字典中的键组成的字典视图
- values()：返回一个由字典中的值组成的字典视图

```
>>> dict1 = {'a': 100, 'b': 85, 'c': 90}
>>> dict1.items()
dict_items([('a', 100), ('b', 85), ('c', 90)])
>>> dict1.keys()
dict_keys(['a', 'b', 'c'])
>>> dict1.values()
dict_values([100, 85, 90])
```

字典方法

- update()：使用一个字典中的项来更新另一个字典

```
>>> dict1
{'a': 100, 'b': 85, 'c': 90}
>>> dict2 = {'c': 84, 'd': 98}
>>> dict1.update(dict2)
>>> dict1
{'a': 100, 'b': 85, 'c': 84, 'd': 98}
```

- pop:用于获取与指定键相关联的值，并将该键-值对从字典中删除

```
>>> dict1.pop('c')
84
>>> dict1
{'a': 100, 'b': 85, 'd': 98}
```

字典与列表的区别

- 键的类型：字典中的键可以是整数，但并非必须是整数。字典中的键可以是任何不可变的类型，如浮点数（实数）、字符串或元组
- 自动添加：即便是字典中原本没有的键，也可以给它赋值，这将在字典中创建一个新项。然而，如果不使用append或其他类似的方法，就不能给列表中没有的元素赋值。
- 成员资格：表达式k in d（其中d是一个字典）查找的是键而不是值，而表达式v in l（其中l是一个列表）查找的是值而不是索引

集合

- 集合set是一种无序的序列，故不存在索引和切片操作
- 唯一性：当集合中存在两个同样的元素的时候，Python只会保存其中的一个
- 确定性：为确保不包含同样的元素，集合中放入的元素只能是不可变的对象
- 故集合中的元素可以是：整数、复数、字符串、元组等；不能是：列表、字典。

集合

- 集合的生成：set()函数或一对花括号{}

```
>>> set([1, 2, 3, 3])
{1, 2, 3}
>>> s = {1, 2, 3, 3}
>>> s
{1, 2, 3}
>>> type(s)
<class 'set'>
```

- 定义空集合：

```
>>> a = set()
>>> type(a)
<class 'set'>
>>> s = {}
>>> type(s)
<class 'dict'>
```

集合的常见操作

- 初始化两个集合

```
>>> a = {1, 2, 3, 4}
>>> b = {3, 4, 5, 6}
```

- 两个集合的并，返回包含两个集合所有元素的集合（去重）：a.union(b) 或者a | b

```
>>> a | b
{1, 2, 3, 4, 5, 6}
>>> a.union(b)
{1, 2, 3, 4, 5, 6}
```

- 两个集合的交，返回包含两个集合共有元素的集合：a.intersection(b) 或者a & b

```
>>> a & b
{3, 4}
>>> a.intersection(b)
{3, 4}
```

集合的常见操作

- 集合a和b的差集，返回只在a不在b的元素组成的集合：a.difference(b) 或者a - b

```
>>> a - b
{1, 2}
>>> a.difference(b)
{1, 2}
```

- a 和b的对称差集，返回在a或在b中，但不同时在a和b中的元素组成的集合：a.symmetric_difference(b) 或者a ^ b

```
>>> a ^ b
{1, 2, 5, 6}
>>> a.symmetric_difference(b)
{1, 2, 5, 6}
```

集合的关系比较

- b 是不是a的子集：b.issubset(a) 或者b <= a，a.issuperset(b) 或者a >= b

```
>>> b.issubset(a)
True
>>> b <= a
True
>>> a.issuperset(b)
True
>>> a >= b
True
```

- 判断真子集

```
>>> a <= a
True
>>> a < a
False
```

集合的其他常用方法

- a.add(x)：集合a中添加单个元素x

```
>>> a = {1, 2, 3, 4}
>>> a.add(5)
>>> a
{1, 2, 3, 4, 5}
```

- a.update(seq)：集合a中添加序列seq中的元素

```
>>> a = {1, 2, 3, 4}
>>> a.update([4, 5, 6])
>>> a
{1, 2, 3, 4, 5, 6}
```

集合的其他常用方法

- a.remove(x)：集合a中移除元素x，如果不存在会报错

```
>>> a = {1, 2, 3, 4}
>>> a.remove(4)
>>> a
{1, 2, 3}
>>> a.remove(4)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 4
```

- a.discard(x)：集合a中移除元素x，如果不存在不会报错

```
>>> a = {1, 2, 3, 4}
>>> a.discard(4)
>>> a
{1, 2, 3}
>>> a.discard(4)
```

集合的其他常用方法

- 集合其他的方法：

```
>>> a = {1, 2, 3, 4}
>>> len(a)
4
>>> 1 in a
True
>>> 1 not in a
False
```

课堂练习- 字典与集合

- 初始化一个字典，记录你本学期选课信息，包括课程的名称、学分和任课教师的姓名
- 实现以下操作：
  - 退一门课
  - 更新某一门课的授课信息（两种方法）
  - 查找某门课程，如果不存在提示“你没有选这门课”
  - 获取所有的课程名称
- 统计" Beautiful is better than ugly Explicit is better than implicit "这句话中有多少个单词。
- 基于现在已经讲授的内容，设计一个作业提交检查系统，目的是检查还有多少人没有交。

三、格式化字符串*

格式化字符串

- 格式化是对字符串进行格式表达的方式，格式化输出包括以下几种方法：
  - 字符串的str.format()方法
  - 使用格式化字符串字面值
  - 旧式字符串格式化方法

1. 字符串format()方法

- 花括号及之内的字符（称为格式字段）被替换为传递给str.format() 方法的对象。花括号中的数字表示传递给str.format() 方法的对象所在的位置。

```
>>> '{} + {} = {}'.format(1, 1, 2)
'1 + 1 = 2'
>>> '{2} {0} {1}'.format(3, 4, 5)
'5 3 4'
```

- str.format() 方法中使用关键字参数名引用值

```
>>> '{name} {age}'.format(age=28, name='Yun')
'Yun 28'
```

1.字符串format()方法

- str.format()方法中可以给字符串中的花括号中加入**标准格式说明符**，语法形式如下：

```
'{:format_spec}'.format()
format_spec - [fill][align][sign][width][.precision][type]

>>> 'Pi: {:*>6.2f}'.format(math.pi)
'Pi: **3.14'
```

  - align表示对齐选项：<>^
  - 如果指定了一个有效的align值，则可以在该值前面加一个fill字符，它可以为任意字符，如果省略则默认为空格符
  - sign选项仅对数字类型有效：+ -
  - width.precision：width表示最小总字段宽度，precision表示精度
  - type：确定了数据应如何呈现。s, d, f, %

2. 使用格式化字符串字面值

- 格式化字符串字面值：在字符串开头的引号（或三引号）前添加f。在这种字符串中，可以在{和}字符之间输入引用的变量，或字面值的Python表达式

```
>>> year = 2016
>>> course = 'Python'
>>> f'Information: {year} - {course}'
'Information: 2016 - Python'

>>> import math
>>> pi = math.pi
>>> f'{pi:<7.3f}'
'3.142  '
>>> f'{pi:<7.2%}'
'314.16%'
```

3. 旧式字符串格式化方法

- % 运算符（求余符）也可用于字符串格式化。给定'string' % values，则string 中的% 实例会以零个或多个values 元素替换。此操作被称为字符串插值

```
>>> import math
>>> 'Pi: %6.2f, e = %3.2f' % (math.pi, math.e)
'Pi:   3.14, e = 2.72'

>>> '%10.3f' % pi
'     3.142'
>>> '%-10.3f' % pi
'3.142    '
>>> '%+10.3f' % pi
'    +3.142'
>>> '%10.3f' % -pi
'    -3.142'
```

四、赋值语句

赋值语句

- 序列解包：将多个值的序列解开，然后放到变量的序列中

```
>>> x, y, z = 1, 2, 3
>>> print(x, y, z)
1 2 3
>>> a = b = c = 100
>>> print(a, b, c)
100 100 100
```

- 如何替换两个变量的值

```
>>> x, y = 3, 4
>>> x, y = y, x
>>> print(x, y)
4 3
```

赋值语句

- 增量赋值：使得代码更加的紧凑和简练

```
>>> x = 2
>>> x += 1
>>> x *= 2
>>> print(x)
6
>>> y = 6
>>> y *= 3
>>> y
18
>>> y /= 9
>>> y
2.0
```

五、条件语句

if条件语句

- 语法结构：

```
if <condition>:
   <statement>
```

- if语句的核心<condition>是一个值为True和False的表达式，这种表达式称之为**条件测试**。条件测试又称为布尔表达式，其结果为布尔型数据（True / False）
- 如果条件测试的值为True，Python就执行紧跟在if语句后面的代码；如果为False，就忽略这些代码
- 这里<statement>为缩进的代码块，同一个代码块使用同样的缩进值，不同的缩进值表示不同的代码块
- Python中常用四个空格（Tab）来表示缩进

if条件语句

- 不同的缩进值表示不同的代码块：

```
x = 0.5
if x > 0:
    print("Hey!")
    print("x is positive")
    print("This is still part of the block")
print("This isn't part of the block, and will always print.")
```

- 当$x = -0.5$时，再执行上面的代码，并查看下结果

if-else语句

- 语法结构：

```
if <condition>:
   <statements 1>
else:
   <statements 2>
```

如果<condition>结果为True，则执行<statements 1>，否则执行<statements 2>

- 示例：

```
>>> num = -10
>>> if num > 0:
...     print("positive number")
... else:
...     print("not positive")
...
not positive
```

if-elif-else语句

- 语法结构：

```
if <condition 1>:
   <statements>
elif <condition 2>:
   <statements>
else:
   <statements>
```

- 这里elif的个数没有限制，可以是0个或多个。else最多只有1个。[注意：这里不涉及<statements>里面的嵌套条件语句]
- 示例：

```
>>> num = 0
>>> if num > 0:
...     print("positive number")
... elif num < 0:
...     print("negative number")
... else:
...     print("Zero")
...
Zero
```

关于else语句

- 下面代码可以没有else语句：

```
age = 3
if age >= 18:
    price = 10
else:
    price = 5

if age >= 18:
    price = 10
elif 0 < age < 18:
    price = 5
```

思考下这样做的好处？

条件测试

- Python不仅可用布尔型变量作为条件，还可直接在if中使用任何表达式作为条件
- 大部分表达式的值都会被当作True，但以下表达式值会被当作False：None，0，空字符串，空列表，空字典，空集合

```
>>> mylist = []
>>> if mylist:
...     print("The first element is:", mylist[0])
... else:
...     print("There is no first element.")
...
There is no first element.
```

- 注意：实际编程中，可以简单测试下表达式的结果是True还是False，尤其是复杂的表达式组合

条件测试

- 比较运算符：==、!=、>、<、>=、<=
- 还可以使用and ， or ，not 等关键词结合多个判断条件：

```
>>> x = 10
>>> y = -5
>>> x > 0 and y < 0
True
>>> not x > 0
False
>>> x < 0 or y < 0
True
```

三元操作

- Python的条件语句还有更为简练的方式：所有代码都放置在一个物理行中即可完成条件语句程序。例如，变量a和b大小条件判断的代码可简化成如下形式:

```
>>> a = 6
>>> b = 4
>>> "a > b" if a > b else "b >= a"
'a > b'
```

- 这种条件语句简单的代码程序还可以用于赋值操作，即三元操作。语法形式：

```
a = x if expression else z
```

其功能为：当expression返回值为True时，执行语句a = x，否则执行语句a = z。

```
>>> a = 4 ** 3 if {} else "A"
>>> a
'A'
>>> a = 4 ** 3 if {1} else "A"
>>> a
64
```

课堂练习- 条件语句

- 设计一个登陆系统，用户需要输入用户名和密码
  - 如果用户名和密码都正确，提示登陆成功
  - 否则提示用户名不存在（当用户名错误时提示）或密码错误（当用户名正确但密码错误时提示）

六、循环语句

for循环

- for循环语法形式：

```
for <variable> in <sequence>:
    <statements>
```

for循环会遍历完<sequence>中所有元素为止，并依次执行代码块<statements>，其中<sequence>序列对象通常是列表、字符串或其他可迭代元素

- 示例：

```
>>> plays = set(["Hamlet", "Macbeth", "King Lear"])
>>> for play in plays:
...     print("Perform", play)
...
Perform Macbeth
Perform King Lear
Perform Hamlet
```

遍历字典

- for循环遍历字典的所有键值对，语法如下：

```
for k, v in dict_name.items()
```

  - items()方法返回一个键值对列表
  - 变量k，v用于存储键值对中的键和值，它们可以使用任意名称

```
>>> transcript = {"Python": 100, "Econ": 95}
>>> for key, value in transcript.items():
...     print(key, value)
...
Python 100
Econ 95
```

遍历字典和列表

- 遍历所有的键：dict_name.keys()

```
>>> for key in transcript.keys():
...     print(key)
...
Python
Econ
>>> for key in transcript:
...     print(key)
...
Python
Econ
```

- 遍历所有的值：dict_name.values()

```
>>> for value in transcript.values():
...     print(value)
...
100
95
```

range()函数

- range()是个内建函数，最常用于for循环，它可以创建一个数字元素组成的列表。一般形式是range(start, stop[, step])。
- 函数的参数必须是整数，默认从start=0开始，step默认值是1。
  - range(5)可以生成0-4的整数，总共5个元素
  - range(1,5): 生成的是从1开始到4的整数，总共4个元素

```
>>> list(range(5))
[0, 1, 2, 3, 4]
>>> list(range(1, 5))
[1, 2, 3, 4]
>>> list(range(1, 10, 3))
[1, 4, 7]
>>> for x in range(3):
...     print(x)
...
0
1
2
```

while循环

- while循环语法形式：

```
while <expression>:
   <statements>
```

while循环中，python会循环执行<statements>，直到<expression>结果为False为止

- 示例：

```
>>> current_num = 1
>>> while current_num <= 5:
...     print(current_num)
...     current_num += 1
...
1
2
3
4
5
```

while循环

- while / else 循环语法形式：

```
>>> count = 0
>>> while count < 2:
...     print(count, " is less than 2")
...     count = count + 1
... else:
...     print(count, " isn't less than 2")
...
0  is less than 2
1  is less than 2
2  isn't less than 2
```

while循环

- 复杂的程序中，很多不同的事件会导致循环程序停止运行，此时可以定义一个变量，用于判断整个程序是否处于活动状态。这个变量称为标志（flag）

```
>>> active = True
>>> cnt = 0
>>> while active:
...     cnt += 1
...     if cnt > 3:
...         active = False
...     else:
...         print(cnt)
...
1
2
3
```

嵌套循环

- 嵌套循环指的是在一个循环语句的代码块中再嵌入一个循环。主要形式如下:

```
# for — nested loop
for variable1 in sequence1:
    for variable2 in sequence2:
        statement1
    statement2

# while — nested loop
while expression:
    while expression:
        statements
    statements
```

嵌套循环

- for嵌套循环

```
>>> z = []
>>> for i in range(1, 4):
...     for j in range(i, 4):
...         z.append([i, j])
...
>>> z
[[1, 1], [1, 2], [1, 3], [2, 2], [2, 3], [3, 3]]
```

嵌套循环

- while嵌套循环

```
>>> i = 1
>>> while i <= 5:
...     j = 1
...     text = ''
...     while j <= i:
...         text += '*'
...         j +=1
...     print(text)
...     i += 1
...
*
**
***
****
*****
```

continue、break和pass语句

- 在条件控制和循环控制中通常结合break、continue、pass等语句形式，用来辅助控制程序的执行，以加代码执行的灵活度。
- continue语句：循环中遇到continue时，程序会返回到循环的最开始重新执行。
- break语句：循环中遇到break时，程序会跳出循环，不管循环条件是不是满足。
- pass语句：表示空语句、空操作，为了保持程序结构的完整性，执行时系统没有任何反应，一般做占位语句

continue、break和pass语句

- 示例：

```
>>> values = [7, 6, 4, 7, 19, 2, 1]
>>> for i in values:
...     if i % 2 == 0:
...         continue
...     print(i / 2)
...
3.5
3.5
9.5
0.5
>>> for i in values:
...     if i % 2 == 0:
...         break
...     print(i / 2)
...
3.5
```

continue、break和pass语句

- 示例：

```
>>> a = 3
>>> if a > 0:
...     print('positive')
... else:
...     pass
...
positive
>>> a = - 3
>>> if a > 0:
...     print('positive')
... else:
...     pass
...
>>>
```

课堂练习- 循环语句

- 阅读下面的代码，输出结果为：

```
i, k = 0, 0
while i <= 10:
    j = i
    if i <= 7:
        i += 2
    k += 1
    if j == i:
        break
print(k)
```

课堂练习- 循环语句

- 计算1到1000的所有奇数的和（要求分别用for循环和while循环实现）
- 打印9*9乘法表
- 编程求1-2+3-4+5...99的所有数的和

七、列表推导式（List Comprehension）

列表推导式

- 列表推导式的语法大致如下

```
[<expression1> <for expression1> <for expressions> <if expressions>]
```

其中在列表方括号中，一定要有表达式<expression1>和for语句<for expression1>，之后可以跟0个或多个for语句和if语句。

```
>>> [x for x in range(5)]
[0, 1, 2, 3, 4]
>>> [x ** 2 for x in range(5) if x % 2 == 0]
[0, 4, 16]
>>> [(x, y) for x in [1,2,3] for y in [3,1,4] if x != y]
[(1, 3), (1, 4), (2, 3), (2, 1), (2, 4), (3, 1), (3, 4)]
```

需要注意的是，表达式<expression1>只能是一个值或者是一个元组：

```
>>> [x, x for x in range(5)]
  File "<stdin>", line 1
    [x, x for x in range(5)]
           ^
SyntaxError: invalid syntax
```

列表推导式

- 多个for语句表示多层嵌套：

```
>>> vec = [[1,2,3], [4,5,6], [7,8,9]]
>>> [num for elem in vec for num in elem]
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

- 嵌套的列表推导式：

```
>>> [[i for i in range(3)] for j in range(3)]
[[0, 1, 2], [0, 1, 2], [0, 1, 2]]
>>> [x * y for x in range(1,5) if x > 2 for y in range(1, 4) if y < 3]
[3, 6, 4, 8]
```

字典推导式

- 语法形式1：

```
{key_exp: value_exp for key, value in dict.items() if condition}

>>> dict1 = {'a': 1, 'b': 2, 'c': 3, 'd': 4}
>>> {x: y for x, y in dict1.items() if x != 'c'}
{'a': 1, 'b': 2, 'd': 4}
```

- 语法形式2：

```
{key_exp: value_exp1 if condition else value_exp2 for key, value in
    dict.items()}
>>> {x: y if x != 'c' else y ** 2 for x, y in dict1.items()}
{'a': 1, 'b': 2, 'c': 9, 'd': 4}
```

enumerate()枚举函数

- enumerate()函数用于将一个可遍历的数据对象(如列表、元组或字符串)组合为一个索引序列，同时列出数据和数据下标。语法：

```
enumerate(sequence, [start=0])
```

其中sequence为可迭代对象，start为下标起始位置，返回一个enumerate对象

```
>>> a = list('abcd')
>>> list(enumerate(a))
[(0, 'a'), (1, 'b'), (2, 'c'), (3, 'd')]
>>> for i, j in enumerate(a):
...     print(i, j)
...
0 a
1 b
2 c
3 d
>>> list(enumerate(a, 2))
[(2, 'a'), (3, 'b'), (4, 'c'), (5, 'd')]
```

创建字典：zip()和for循环

- zip()函数用于将可迭代的对象作为参数，将对象中对应的元素打包成一个个元组，返回由这些元组组成的对象。对返回的元组对象，使用list()输出列表，dict()输出字典

```
>>> a = [0, 1, 2, 3, 4]
>>> b = list("abcde")
>>> zip(a, b)
<zip object at 0x7f93abf4d140>
>>> list(zip(a, b))
[(0, 'a'), (1, 'b'), (2, 'c'), (3, 'd'), (4, 'e')]
>>> dict(zip(a, b))
{0: 'a', 1: 'b', 2: 'c', 3: 'd', 4: 'e'}
```

- for循环创建字典：

```
>>> res = {}
>>> for key, value in zip(a, b):
...     res[key] = value
...
>>> res
{0: 'a', 1: 'b', 2: 'c', 3: 'd', 4: 'e'}
```

课堂练习- 列表推导式

- 表达式[str(i) for i in range(4)]的结果是?
- 毕达哥拉斯三元组是满足$a^2 + b^2 = c^2$的三个整数，使用列表推导式计算所有元素小于30的毕达哥拉斯三元组，即a、b、c均小于30。
- 阅读下列代码，其结果是：

```
x = list(range(10))
for index, value in enumerate(x):
   if value == 3:
      x[index] = 5
   else:
      x[index] = x[index-5]
print(x)
```