Python 基础 I

金融科技协会 2020 年 10 月 16 日

目录

一、基础知识

1.1 解释器和编译器的区别

解释器：直接执行用编程语言编写的指令的程序 （eg.python, javascript）

编译器：把源代码转换成（翻译）低级语言的程序（eg. C 语言代码被编译成二进制代码，在 windows 平台上执行）

1.2 算法的定义

算法：面向计算机，一系列解决问题的清晰指令。

菜谱和算法都由原料(对象)和操作说明(语句)组成。

鸡蛋火腿肠的菜谱： 先取一些火腿肠； 再加些鸡蛋； 如果喜欢吃辣，加些辣味火腿肠； 煮熟为止。记得每隔 10 分钟检查一次。

1.3 数

交互式的 Python 解释器可用作功能强大的计算器：

```
[1]: 2+2

[1]: 4
```

图 1 加法

```
[2]: 1/1

[2]: 1.0
```

图 2 除法

除法运算的结果为小数，即浮点数(float 或 floating-point number)

如果想丢弃小数部分，即执行整除运算，可使用双斜杠

```
[8]: 4//3

[8]: 1
```

图 3 整除

```
[9]: 10%3

[9]: 1
```

图 4 余数

```
[10]: 3**2

[10]: 9
```

图 5 乘方

```
[12]: -3**2

[12]: -9
```

图 6 乘方（无括号）

```
[13]: (-3)**2

[13]: 9
```

图 7 乘方（有括号）

1.4 变量

变量：没有固定的值，可以改变的数。

```
[14]: x=3
```

图 8 赋值

这称为赋值(assignment)，我们将值 3 赋给了变量 x。

1.5 表达式和语句

Python 代码由表达式和语句组成，并由 Python 解释器负责执行。

表达式 是 一些东西，而语句 做 一些事情。

1.5.1 表达式：值、变量和运算符的组合

```
[16]: 2 * 2

[16]: 4
```

图 9 表达式

1.5.2 语句：执行说明

```
[17]: print(2 * 2)
      4
```

图 10 语句

1.6 获取用户输入

```
[*]: name = input("请输入你的名字: ")
     请输入你的名字:

[*]: choice=input('请选择1或2')
     if choice=='1':
         print('正确')
     else:
         print('错误')
```

图 11 输入

1.7 内置函数及其调用

内置函数：编程语言中预先定义的函数

函数 pow：幂运算

```
[16]: pow(2, 3)

[16]: 8
```

图 12 pow

函数 abs：计算绝对值

```
[17]: abs(-10)

[17]: 10
```

图 13 abs

函数 round：圆整到最接近的整数

```
[18]: round(2 / 3)

[18]: 1
```

图 14 round

1.8 模块的导入及其简单使用

模块：可将模块视为扩展，通过将其导入可以扩展 Python 功能

先使用 import 导入模块，再以 module.function 的方式使用模块中的函数

模块 math：

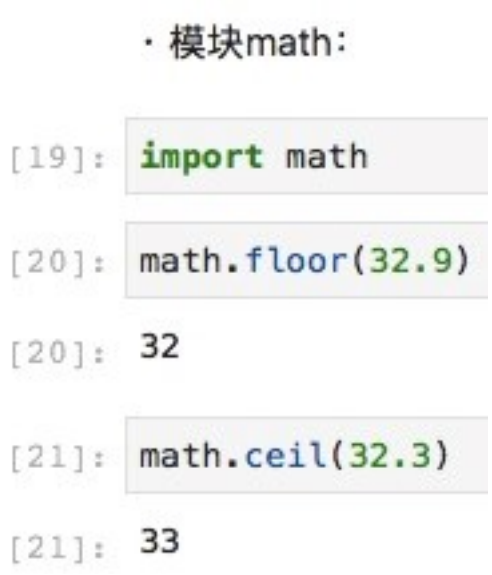

图 15 math 模块

模块 cmath：专门用于处理复数（由实部和虚部组成）的模块

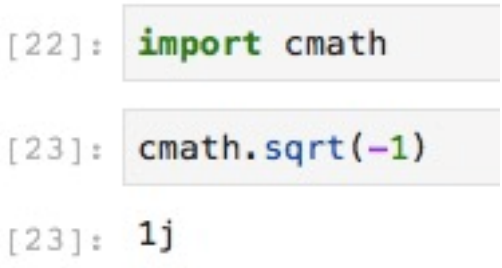

图 16 cmath 模块

1.9 保存与执行程序

（1）Spyder 中保存程序和执行程序的方法

（2）Jupyter Notebook 中保存程序和执行程序的方法

（3）cmd 中执行已有 python 脚本(*.py)的方法

Windows： 在“开始”菜单中点击“运行”，打开运行窗口（win+R），输入 cmd 打开命令窗口

Mac：command+space，输入终端

1.10 注释

1.10.1 单行注释与多行注释

井号(#)在代码中，井号后面到行尾的所有内容都将被忽略

图 17 注释

1.10.2 Spyder 与 Jupyter Notebook 中的使用方法

command+1/4/5

二、字符串

字符串：数字、字母、下划线组成的一串字符

2.1 单引号、双引号、三引号

2.1.1 单引号和双引号

Python 在打印字符串时，在大多数情况下，单引号和双引号是没有区别的

图 18 单引号与双引号

特例 1：

"Let's go!"

'Let's go!'

这时候就不适合用单引号：解释器将报错，不知道如何处理第二个单引号后余下的内容

特例 2：

图 19 双引号的错误

这时候就不适合用双引号：解释器将报错，不知道如何处理第二个双引号后余下的内容

2.2.2 三引号

多行字符串/注释用三个单引号 ''' 或者三个双引号 """ 将注释括起来

```
[35]: chat = """how are you?
      i'm fine, than you,and you?
          me too!"""

      print(chat)

      how are you?
      i'm fine, than you,and you?
          me too!
```

图 20 三引号

2.2 转义字符

使用反斜杠\对引号进行转义(将“\”加在需要转义的引号前)

```
[36]: 'Let\'s go!'

[36]: "Let's go!"

[37]: "\"Hello, world!\" she said"

[37]: '"Hello, world!" she said'
```

图 21 转义字符

2.3 字符串拼接(+)

```
[38]: x = "New"
      y = "York"
      z = "City"

      print(x+y+z)

      NewYorkCity
```

图 22 字符串拼接

2.4 str()与 repr()

使用 str 能以合理的方式将值转换为用户能够看懂的字符串

```
[39]: print(str("Hello,\nworld!")) #换行符的编码\n

Hello,
world!
```

图 23 str

使用 repr 时，通常会获得值的合法 Python 表达式

```
[40]: print(repr("Hello,\nworld!"))

'Hello,\nworld!'
```

图 24 repr

2.5 用 r 快速转义(路径变量)

```
[41]: path = 'C:\nowhere'

[42]: print(path)

C:
owhere

[43]: #对反斜杠进行转义

[44]: print('C:\\nowhere')

C:\nowhere
```

图 25 转义字符\

对于很长的路径，具有大量的反斜杠?

原始字符串：在字符串前加 r，它们根本不会对反斜杠做特殊处理

```
[45]: print(r'C:\nowhere\nowhere\nowhere\nowhere')

C:\nowhere\nowhere\nowhere\nowhere
```

图 26 原始字符串

2.6 encode()与 decode()

```
[46]: "Hello, world!".encode("UTF-32") #UTF-32编码将字符串转换为bytes
[46]: b'\xff\xfe\x00\x00H\x00\x00\x00e\x00\x00\x00l\x00\x00\x00l\x00\x00\x00o\
00\x00\x00w\x00\x00\x00o\x00\x00\x00r\x00\x00\x00l\x00\x00\x00d\x00\x00\
[47]: b'\xff\xfe\x00\x00H\x00\x00\x00e\x00\x00\x00l\x00\x00\x00l\x00\x00\x00o\
[47]: 'Hello, world!'
```

图 27 字符编码

2.7 chr()与 ord()

字符是根据顺序值排列的，要获悉字母的顺序值，可使用函数 ord

```
[48]: ord("a") #ord()函数主要用来返回对应字符的ascii码
[48]: 97
[49]: ord("b")
[49]: 98
[50]: ord("c")
[50]: 99
```

图 28 ord

函数 chr 与函数 ord 相反

```
[51]: chr(97)
[51]: 'a'
```

图 29 chr

2.8 字符串基本操作

2.8.1 索引

索引：序列中的所有元素都有编号——从 0 开始递增

```
[52]: greeting = 'Hello'

[53]: greeting[0]

[53]: 'H'

[54]: greeting[-1]

[54]: 'o'
```

图 30 索引

2.8.2 切片

切片(slicing)：访问特定范围内的元素

numbers[x:y] 第一个索引 x 是包含的第一个元素的编号，第二个索引 y 是切片后余下的第一个元素的编号(包括头不包括尾)

```
[56]: numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

[57]: numbers[3:6]

[57]: [4, 5, 6]

[58]: numbers[-3:-1]

[58]: [8, 9]

[59]: numbers[-3:0]

[59]: []
```

图 31 切片

如果切片结束于序列末尾，可省略第二个索引

```
[60]: numbers[-3:]
[60]: [8, 9, 10]
[61]: numbers[:3]
[61]: [1, 2, 3]
[62]: numbers[:]
[62]: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

图 32 切片时省略索引

更大的步长

```
[63]: numbers[0:10:2] #步长为2时，将从起点和终点之间每隔一个元素提取一个元素
[63]: [1, 3, 5, 7, 9]
[64]: numbers[10:0:-2]
[64]: [10, 8, 6, 4, 2]
```

图 33 改变步长

2.8.3 乘法

将序列与数 x 相乘时，将重复这个序列 x 次来创建一个新序列

```
[65]: 'python' * 5
[65]: 'pythonpythonpythonpythonpython'
[66]: [42] * 10
[66]: [42, 42, 42, 42, 42, 42, 42, 42, 42, 42]
```

图 34 序列乘法

2.8.4 成员资格检查

要检查特定的值是否包含在序列中，可使用运算符 in

```
[68]: permissions = 'rw'
      'w' in permissions
[68]: True
```

图 35 成员资格检查

2.8.5 长度

```
[69]: numbers = [100, 34, 678]
[70]: len(numbers)
[70]: 3
```

图 36 序列长度

2.8.6 最大值和最小值

```
[71]: numbers = [100, 34, 678]

[72]: max(numbers)

[72]: 678

[73]: min(numbers)

[73]: 34
```

图 37 序列最大最小值

2.9 format 格式化函数

格式化字符串的函数：一种向命名替换字段提供值的方式

2.9.1 替换字段名

（1）按顺序将字段和参数配对 （2）给参数指定名称，这种参数将被用于相应的替换字段中

```
[3]: "{foo} {} {bar} {}".format(1, 2, bar=4, foo=3)

[3]: '3 1 4 2'

[4]: "{foo} {1} {bar} {0}".format(1, 2, bar=4, foo=3)

[4]: '3 2 4 1'
```

注意：不能同时使用手工编号和自动编号，因为这样很快会变得混乱不堪

图 38 format 基本用法

不仅使用提供的值本身，而是可访问其组成部分

· 不仅使用提供的值本身，而是可访问其组成部分

```
[6]: fullname = ["Alfred", "Smoketoomuch"]
     "Mr {name[1]}".format(name=fullname)

[6]: 'Mr Smoketoomuch'

[7]: import math
     tmpl = "The {mod.__name__} module defines the value {mod.pi} for π"
     tmpl.format(mod=math)

[7]: 'The math module defines the value 3.141592653589793 for π'
```

图 39 format 访问提供的值的组成部分

2.9.2 基本转换

三个标志(s、r 和 a)指定分别使用 str、repr 和 ascii 进行转换

```
[39]: print("{pi!s} {pi!r} {pi!a}".format(pi="π"))
      π 'π' '\u03c0'
```

图 40 format 参数之转换字符

字符串格式设置中的类型说明符

```
[12]: "The number is {num}".format(num=42) 'The number is 42'
[12]: 'The number is 101010'
[13]: "The number is {num:b}".format(num=42)
[13]: 'The number is 101010'
```

表3-1 字符串格式设置中的类型说明符

类型	含 义
b	将整数表示为二进制数
c	将整数解读为Unicode码点
d	将整数视为十进制数进行处理，这是整数默认使用的说明符
e	使用科学表示法来表示小数（用e来表示指数）
E	与e相同，但使用E来表示指数
f	将小数表示为定点数
F	与f相同，但对于特殊值（nan和inf），使用大写表示
g	自动在定点表示法和科学表示法之间做出选择。这是默认用于小数的说明符，但在默认情况下至少有1位小数
G	与g相同，但使用大写来表示指数和特殊值
n	与g相同，但插入随区域而异的数字分隔符
o	将整数表示为八进制数
s	保持字符串的格式不变，这是默认用于字符串的说明符
x	将整数表示为十六进制数并使用小写字母
X	与x相同，但使用大写字母
%	将数表示为百分比值（乘以100，按说明符f设置格式，再在后面加上%）

图 41 format 类型说明符

2.9.3 宽度、精度和千位分隔符

```
[14]: "{num:10}".format(num=3)
[14]: '         3'
[15]: "{name:10}".format(name="Bob")
[15]: 'Bob       '
```

注意：数和字符串的对齐方式不同

图 42 format 参数之宽度

```
[20]: from math import pi
      "{name} is approximately {value:.2f}.".format(value=pi, name="π")
[20]: 'π is approximately 3.14.'
```

图 43 format 参数之精度

```
[21]: 'One googol is {:,}'.format(10**100) #用逗号来添加千位分隔符
[21]: 'One googol is 10,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,00
0,000,000,000,000,000,000,000,000,000,000,000,000,000'
```

图 44 format 参数之千位分隔符

2.9.4 符号、对齐和用 0 填充

在指定宽度和精度的数前面，可添加一个标志。这个标志可以是零、加号、减号或空格，其中零表示使用 0 来填充数字。

```
[41]: '{:010.2f}'.format(pi)
[41]: '0000003.14'
```

图 45 format 参数之填充

要指定左对齐、右对齐和居中，可分别使用<、>和^

```
[23]: print('{0:<10.2f}\n{0:^10.2f}\n{0:>10.2f}'.format(pi))
3.14
   3.14
      3.14
```

图 46 format 参数之对齐

说明符=，它指定将填充字符放在符号和数字之间

```
[24]: print('{0:10.2f}\n{1:10.2f}'.format(pi, -pi))
      3.14
     -3.14
[25]: print('{0:10.2f}\n{1:=10.2f}'.format(pi, -pi))
      3.14
-     3.14
```

图 47 format 参数之“=”

给正数加上符号，可使用说明符+，而不是默认的-。如果将符号说明符指定为空格，会在正数前面加上空格而不是+。

```
[27]: print('{0:-.2}\n{1:-.2}'.format(pi, -pi)) #默认设置
3.1
-3.1
[30]: print('{0:+.2}\n{1:+.2}'.format(pi, -pi))
+3.1
-3.1
[31]: print('{0: .2}\n{1: .2}'.format(pi, -pi))
 3.1
-3.1
```

图 48 format 参数之“+”

#号：可将其放在符号说明符和宽度之间，这个选项将触发另一种转换方式，转换细节随类型而异。

```
[37]: "{:b}".format(42) #b：将整数表示为二进制数
[37]: '101010'
[33]: "{:#b}".format(42)
[33]: '0b101010'
[38]: "{:g}".format(42) #g:保留小数点后面的零
[38]: '42'
[36]: "{:#g}".format(42)
[36]: '42.0000'
```

图 49 format 参数之“#”

2.10 字符串常用函数

2.10.1 center

通过在两边添加填充字符(默认为空格)让字符串居中

```
[78]: "The Middle by Jimmy Eat World".center(39)
[78]: '     The Middle by Jimmy Eat World     '
```

图 50 center

2.10.2 find

方法 find 在字符串中查找子串。如果找到，就返回子串的第一个字符的索引，否则返回-1

```
[80]: 'With a moo-moo here, and a moo-moo there'.find('moo') #注意：空格也算
[80]: 7
```

图 51 find

2.10.3 join

字符串.join(列表):将字符串插入到 join 传入列表的字符串元素之间，生成一个新的字符串

```
[82]: li = ['hello','python18','!']
      print(' '.join(li))

      hello python18 !
```

图 52 join

2.10.4 split

split：其作用与 join 相反，用于将字符串拆分为序列

```
[84]: '1+2+3+4+5'.split('+')

[84]: ['1', '2', '3', '4', '5']
```

图 53 split

2.10.5 upper

string.upper()：将字符串中所有的字母都转换为大写，并返回结果

```
[86]: s = 'Aloha'
      trans_v1 = s.upper()
      print(trans_v1)

      ALOHA
```

图 54 upper

2.10.6 lower

string.lower()：将字符串中所有的字母都转换为小些，并返回结果

```
[88]: s = 'ALOHA'
      trans_v1 = s.lower()
      print(trans_v1)

      aloha
```

图 55 lower

2.10.7 title

string.title()：将字符串中所有单词的首字母都大写，并返回结果

```
[90]: s = 'new yrok city'
      trans_v1 = s.title()
      print(trans_v1)

      New Yrok City
```

图 56 title

2.10.8 capitalize

string.capitalize():返回字符串的副本，但将第一个字符大写

```
[92]: s = 'there for u'
      trans_v1 = s.capitalize()
      print(trans_v1)

      There for u
```

图 57 capitalize

2.10.9 isupper

检查是否大写

```
[94]: print('AB'.isupper())
      True

[95]: print('Aa'.isupper())
      False
```

图 58 isupper

2.10.10 islower

检查是否小写

```
[97]: print('ab'.islower())
      True

[98]: print('Aa'.islower())
      False
```

图 59 islower

2.10.11 istitle

检查是否首字母大写

```
· 检查是否首字母大写

[100]: print('Aa Bc'.istitle())
       True

[101]: print('Aa bc'.istitle())
       False
```

图 60 istitle

2.10.12 isspace

判断字符串是否是空白

```
[103]: print(' '.isspace())
       True

[104]: print('Aa BC'.isspace())
       False
```

图 61 isspace

2.10.13 isdigit

检查字符串是否是数字

```
[106]: print('34'.isdigit())
       True

[107]: print('a'.isdigit())
       False
```

图 62 isdigit

2.10.14 isdecimal

检查字符串是否是十进制数

```
[109]: print('123'.isdecimal())
       True

[110]: print('a12'.isdecimal())
       False
```

图 63 isdecimal

2.10.15 isnumeric

检查字符串中的所有字符是否都是数字字符

```
[112]: print('34'.isnumeric())
       True

[113]: print('a'.isnumeric())
       False
```

图 64 isnumeric

2.10.16 isprintable

判断是否是可打印字符（例如制表符、换行符不是可打印字符，空格是可打印字符）

```
[115]: print('\n'.isprintable())
       False

[116]: print('\t'.isprintable())
       False

[117]: print(' '.isprintable())
       True
```

图 65 isprintable

2.10.17 replace

将指定子串都替换为另一个字符串，并返回替换后的结果

```
[119]: 'This is a test'.replace('is', 'eez')

[119]: 'Theez eez a test'
```

图 66 replace

2.10.18 translate

能够同时替换多个字符，因此效率比 replace 高

将一段英语文本转换为带有德国口音的版本，为此必须将字符 c 和 s 分别替换为 k 和 z

```
[121]: table = str.maketrans('cs', 'kz')  #使用translate前必须创建一个转换表，指出对应关系
       'this is an incredible test'.translate(table)

[121]: 'thiz iz an inkredible tezt'
```

图 67 translate

2.10.19 strip

移除左右两边字符

```
[123]: 'Martin Garrix'.strip('M''x')

[123]: 'artin Garri'
```

图 68 strip

2.10.20 lstrip

移除左边字符

```
[125]: 'Martin Garrix'.lstrip('M')

[125]: 'artin Garrix'
```

图 69 lstrip

2.10.21 rstrip

移除右边字符

```
[127]: 'Martin Garrix'.rstrip('x')

[127]: 'Martin Garri'
```

图 70 rstrip

三、列表与元组

3.1 序列的定义与分类

数据结构是以某种方式（如通过编号）组合起来的数据元素（如数、字符乃至其他数据结构）集合。在 Python 中，最基本的数据结构为序列。序列中的每个元素都有编号，即其位置或索引，其中第一个元素的索引为 0，第二个元素的索引为 1，依此类推。

Python 内置了多种序列，包括列表、元组和字符串。

3.2 序列常用操作

3.2.1 索引

序列中的所有元素都有编号——从 0 开始递增。通过使用编号来访问各个元素，这称之为索引。这种索引方式适用于所有序列。当你使用负数索引时，Python 将从右（即从最后一个元素）开始往左数，因此-1 是最后一个元素的位置。

```
greeting = "Hello"
print(greeting[0])
```

```
H
```

图 71 索引

3.2.2 切片

除使用索引来访问单个元素外，还可使用切片来访问特定范围内的元素。为此，可使用两个索引，并用冒号分隔。

切片适用于提取序列的一部分，其中的编号非常重要：第一个索引是包含的第一个元素的编号，但第二个索引是切片后余下的第一个元素的编号。一般格式为 List[首位元素：末位元素：步长]，末位数为-1，如果步长为负表述反向索引。如果切片结束于序列末尾，可省略第二个索引。同样，如果切片始于序列开头，可省略第一个索引。

```
: numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
numbers[3:6]
numbers[0::2]
numbers[-1:-8:-2]
numbers[:3]
```

```
: [1, 2, 3]
```

图 72 切片

3.2.3 相加与乘法

可使用加法运算符来拼接序列。从错误消息可知，不能拼接列表和字符串，虽然它们都是序列。一般而言，不能拼接不同类型的序列。

将序列与数 x 相乘时，将重复这个序列 x 次来创建一个新序列。

```
[1, 2, 3] + [4, 5, 6]
'Hello,' + 'world!'
'python' * 5
```

```
'pythonpythonpythonpythonpython'
```

图 73 相加与乘法

3.2.4 成员资格检查

要检查特定的值是否包含在序列中，可使用运算符 in。它检查是否满足指定的条件，并返回相应的值：满足时返回 True，不满足时返回 False。

```
permissions = 'rw'
'w' in permissions
```

```
True
```

图 74 成员资格检查

3.3 列表

3.3.1 list()

鉴于不能像修改列表那样修改字符串，因此在有些情况下使用字符串来创建列表很有帮助。为此，可使用函数 list。请注意，可将任何序列（而不仅仅是字符串）作为 list 的参数。

```
list('Hello')
```

```
['H', 'e', 'l', 'l', 'o']
```

图 75 list

3.3.2 增、删、查

方法 append 用于将一个对象附加到列表末尾。

```
lst = [1,2,3]
lst.append(4)
print(lst)
```

```
[1, 2, 3, 4]
[1, 2, 3, 4, 5, 6]
[1, 2, 'four', 3, 4, 5, 6]
```

图 76 append

方法 extend 让你能够同时将多个值附加到列表末尾，为此可将这些值组成的序列作为参数提给方法 extend。换而言之，你可使用一个列表来扩展另一个列表。

```
a = [1,2,3]
b = [4,5,6]
a.extend(b)
print(a)
```

```
[1, 2, 3, 4, 5, 6]
```

图 77 extend

方法 insert 用于将一个对象插入列表。

```
a = [1,2,3]
a.insert(2,"four")
print(a)
```

```
[1, 2, 'four', 3]
```

图 78 insert

方法 pop 从列表中删除一个元素（末尾为最后一个元素），并返回这一元素，但是也可以加位置。

```
x = [1,2,3]
x.pop()
x.pop(1)
```

```
2
```

图 79 pop

方法 remove 用于删除第一个为指定值的元素，如果值不存在，报错。remove 是就地修改且不返回值的方法之一。不同于 pop 的是，它修改列表，但不返回任何值。

```
x = ['to', 'be', 'or', 'not', 'to', 'be']
x.remove('be')
x
```

```
['to', 'or', 'not', 'to', 'be']
```

图 80 remove

方法 del 从列表中删除元素。

```
names = ['Alice', 'Beth', 'Cecil', 'Dee-Dee', 'Earl']
del names[2]
names
```

```
['Alice', 'Beth', 'Dee-Dee', 'Earl']
```

图 81 del

方法 clear 就地清空列表的内容。

```
lst = [1, 2, 3]
lst.clear()
lst
```

```
[]
```

图 82 clear

方法 count 计算指定的元素在列表中出现了多少次。

```
['to', 'be', 'or', 'not', 'to', 'be'].count('to')#返回对应元素的数量
x = [[1, 2], 1, 1, [2, 1, [1, 2]]]
x.count(1)
```

```
2
```

图 83 count

方法 index 在列表中查找指定值第一次出现的索引。

```
knights = ['We', 'are', 'the', 'knights', 'who', 'say', 'ni']
knights.index('who')
```

```
4
```

图 84 index

方法 copy 复制列表。前面说过，常规复制只是将另一个名称关联到列表。要让 a 和 b 指向不同的列表，就必须将 b 关联到 a 的副本。

```
a = [1,2,3]
b = a.copy()
b[1]=4
a
```

```
[1, 2, 3]
```

图 85 copy

方法 reverse 按相反的顺序排列列表中的元素。注意到 reverse 修改列表，但不返回任何值。

```
x = [1,2,3]
x.reverse()
x
```

```
[3, 2, 1]
```

图 86 reverse

如果要按相反的顺序迭代序列，可使用函数 reversed。这个函数不返回列表，而是返回一个迭代器（迭代器将在第 9 章详细介绍）。你可使用 list 将返回的对象转换为列表。

```
x = [1,2,3]
list(reversed(x))
```

```
[3, 2, 1]
```

图 87 reversed

方法 sort 用于对列表就地排序。就地排序意味着对原来的列表进行修改，使其元素按顺序排列，而不是返回排序后的列表的副本。

```
x = [4,6,2,1,7,8]
x.sort()
x
```

```
[1, 2, 4, 6, 7, 8]
```

图 88 sort

3.4 元组：不可修改的序列

与列表一样，元组也是序列，唯一的差别在于元组是不能修改的（你可能注意到了，字符串也不能修改）。元组语法很简单，只要将一些值用逗号分隔，就能自动创建一个元组。

空元组和单元素元组的定义

空元组用两个不包含任何内容的圆括号表示。

单元素元组表示只包含一个值的元组。虽然只有一个值，也必须在它后面加上逗号。

```
()

42,
```

```
(42,)
```

图 89 单元组

函数 tuple 的工作原理与 list 很像：它将一个序列作为参数，并将其转换为元组。如果参数已经是元组，就原封不动地返回它。

```
tuple([1,2,3])
tuple('abc')
tuple((1,2,3))
```

```
(1, 2, 3)
```

图 90 tuple

序列解包（或可迭代对象解包）：将一个序列（或任何可迭代对象）解包，并将得到的值存储到一系列变量中。下面用例子进行解释。

```
values = 1,2,3
x,y,z = values #分别赋值
x
```

```
1
```

图 91 序列解包

四、字典

4.1 字典的定义与结构特征

字典的名称指出了这种数据结构的用途。字典（日常生活中的字典和 Python 字典）旨在让你能够轻松地找到特定的单词（键），以获悉其定义（值）。字典由键及其相应的值组成，这种键-值对称为项。在前面的示例中，键为名字，而值为电话号码。每个键与其值之间都用冒号（:）分隔，项之间用逗号分隔，而整个字典放在花括号内。

4.2 空字典的创建

空字典（没有任何项）用两个花括号表示，类似于下面这样：{}。

4.3 dict()

可使用函数 dict()从其他映射（如其他字典）或键-值对序列创建字典。

```
items = [('name', 'Gumby'), ('age', 42)]
d = dict(items)
print(d)
```

```
{'name': 'Gumby', 'age': 42}
```

图 92 dict

4.4 作为 key 的要求

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。

4.5 字典的基本操作

4.5.1 len()

方法 len()计算字典的长度。

```
d = {'num' : 123 , 'name' : "doiido" }
len(d)
```

```
2
```

图 93 计算字典长度

4.5.2 增（dict[key]=value update）

方法 dict[key]=value，用于创建一个新的字典，返回一个新的字典。

```
dict1 = dict({'name': 'li', 'age': 24})
print('dict1:', dict1)
```

```
dict1: {'name': 'li', 'age': 24}
```

图 94 字典添加新值

方法 update 使用一个字典中的项来更新另一个字典。

```
d = {'title': 'Python Web Site','changed': 'Mar 14 22:09:15 MET 2016'}
x = {'title': 'Python Language Website'}
d.update(x)
d
```

```
{'changed': 'Mar 14 22:09:15 MET 2016', 'title': 'Python Language Websit
e'}
```

图 95 update

4.5.3 删（del clear pop popitem）

方法 clear 删除所有的字典项，这种操作是就地执行的（就像 list.sort 一样），因此什么都不返回（或者说返回 None）。

```
d = {}
d['name'] = 'Gumby'
d['age'] = 42
returned_value = d.clear()
print(returned_value)
```

```
None
```

图 96 clear

方法 pop 可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。

```
d = {'x': 1, 'y': 2}
d.pop('x')
d
```

```
{'y': 2}
```

图 97 pop

方法 popitem 类似于 list.pop，但 list.pop 弹出列表中的最后一个元素，而 popitem 随机地弹出一个字典项，因为字典项的顺序是不确定的，没有“最后一个元素”的概念。如果你要以高效地方式逐个删除并处理所有字典项，这可能很有用，因为这样无需先获取键列表。

```
d = {'url': 'http://www.python.org', 'spam': 0, 'title': 'Python Web Site'}
d.popitem()
```

```
('title', 'Python Web Site')
```

图 98 popitem

查（成员资格检查 get）

方法 get 为访问字典项提供了宽松的环境。通常，如果你试图访问字典中没有的项，将引发错误，而使用 get 不会这样。

```
d = {}
print(d.get('name'))
```

```
None
```

图 99 get

4.6 format_map()

可使用字符串格式设置功能来设置值的格式，这些值是作为命名或非命名参数提供给方法 format 的。我们也可在字典中包含各种信息，这样只需在格式字符串中提取所需的信息即可。为此，必须使用 format_map 来指出你将通过一个映射来提供所需的信息。

```
People={
        'name':['john','peter'],
        'age':[56,64]
        }
'My name is {name[0]},i am {age[0]} old'.format_map(People)
```

```
'My name is john,i am 56 old'
```

图 100 format_map

4.7 复制（copy() deepcopy）

方法 copy 返回一个新字典，其包含的键-值对与原来的字典相同（这个方法执行的是浅复制，因为值本身是原件，而非副本）。

```
x = {'username': 'admin', 'machines': ['foo', 'bar', 'baz']}
y = x.copy()
y['username'] = 'mlh'
y['machines'].remove('bar')
print(y)
print(x)
```

```
{'username': 'mlh', 'machines': ['foo', 'baz']}
{'username': 'admin', 'machines': ['foo', 'baz']}
```

图 101 copy

当替换副本中的值时，原件不受影响。然而，如果修改副本中的值（就地修改而不是替换），原

件也将发生变化，因为原件指向的也是被修改的值（如这个示例中的'machines'列表所示）。避免这种问题，方法深复制即同时复制值及其包含的所有值。可使用模块 copy 中的函数 deepcopy。

```
from copy import deepcopy
d = {}
d['names'] = ['Alfred', 'Bertrand']
c = d.copy()
dc = deepcopy(d)
d['names'].append('Clive')
print(c)
print(dc)
```

```
{'names': ['Alfred', 'Bertrand', 'Clive']}
{'names': ['Alfred', 'Bertrand']}
```

图 102 deepcopy

4.8 keys() values() items()

方法 keys 返回一个字典视图，其中包含指定字典中的键。

```
car = {
  "brand": "Porsche",
  "model": "911",
  "year": 1963
}

x = car.keys()

print(x)
```

```
dict_keys(['brand', 'model', 'year'])
dict_values(['Porsche', '911', 1963])
```

图 103 keys

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。字典项在列表中的排列顺序不确定。

```
car = {
  "brand": "Porsche",
  "model": "911",
  "year": 1963
}

car.items()
```

```
dict_items([('brand', 'Porsche'), ('model', '911'), ('year', 1963)])
```

图 104 items

方法 values 返回一个由字典中的值组成的字典视图。不同于方法 keys，方法 values 返回的视图可能包含重复的值。

```
car = {
  "brand": "Porsche",
  "model": "911",
  "year": 1963
}

x = car.values()

print(x)
```

```
dict_values(['Porsche', '911', 1963])
```

图 105 values

4.9 setdefault()

方法 setdefault 有点像 get，因为它也获取与指定键相关联的值，但除此之外， setdefault 还在字典不包含指定的键时，在字典中添加指定的键-值对。

```
d = {}
d.setdefault('name', 'N/A')
'N/A'
print(d)
d['name'] = 'Gumby'
d.setdefault('name', 'N/A')
'Gumby'
print(d)
```

```
{'name': 'N/A'}
{'name': 'Gumby'}
```

图 106 setdefault

五、流程控制与其他 Trick

5.1 再谈 print 与 import

5.1.1 打印多个参数

print 实际上可同时打印多个表达式，条件是用逗号分隔它们，这时 print 会自动在参数之间插入了一个空格字符。

```
1 print('age:',15)
```

```
age: 15
```

图 107 print 打印多个对象

如果我们不想包含空格怎么办呢？print 还可以自定义分隔符。

```
1 print('age',15,sep=':')
age:15
```

图 108 print 自定义分隔符

5.1.2 导入时重命名

我们可以用 import 导入一个模块或者使用 import from 的形式导入一个模块下的某个子模块，也可以用 from import *的形式导入一个模块下的全部子模块。

```
1 import pandas
```

```
1 from pandas import DataFrame
```

```
1 from pandas import DataFrame,Series
```

```
1 from pandas import *
```

图 109 import

当有两个模块的子模块同名时，如果仅仅用 import 导入容易造成混乱，这时我们一般采用 import as 的形式为模块重命名，以避免命名空间的混乱。

```
1 import pandas as pd
```

图 110 导入时重命名

5.2 赋值魔法

5.2.1 序列解包

```
1 x, y, z = 1, 2, 3
2 print(x, y, z)
```

```
1 2 3
```

图 111 多变量赋值

在 python 中我们可以给多个变量一起赋值。

```
1 x, y = y, x
2 print(x, y, z)
```

```
2 1 3
```

图 112 交换赋值

使用这种方式还可以交换两个变量的值。

```
1 values = 1, 2, 3
2 print(values)
```

```
(1, 2, 3)
```

```
1 x, y, z = values
2 print(x)
```

```
1
```

图 113 序列解包

实际上，这里执行的操作称为序列解包（或可迭代对象解包）：将一个序列（或任何可迭代对象）解包，并将得到的值存储到一系列变量中。以上的赋值就是在解包 values 这个元组。

```
1 x, y, z = 1, 2
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-16-4ef780a3c1ea> in <module>
----> 1 x, y, z = 1, 2

ValueError: not enough values to unpack (expected 3, got 2)
```

图 114 序列解包时变量个数不同

注意序列解包时变量的个数必须要与序列中值的个数相等，否则会造成错误。

```
1 x, y, *z = [1, 2, 3, 4]
2 print(z)
```

```
[3, 4]
```

图 115 使用*吸收多余的值

为了避免出现变量个数不匹配的问题，我们可以在最后加一个*，它会将多余的值都包含在这个星号中。

```
1 x, *y, z = [1, 2, 3, 4, 5]
2 print(x, y, z)
```

```
1 [2, 3, 4] 5
```

图 116 *放在中间

*也可以放在中间，这时它会收集除第一个和最后一个之外的值。

5.2.2 链式赋值

```
1 x = y = 5
2 print(x, y)
```

```
5 5
```

图 117 链式赋值

链式赋值是一种快捷方式，用于将多个变量关联到同一个值。

5.2.3 增强赋值

```
1 x = 1
2 x += 1
3 print(x)
```

```
2
```

图 118 增强赋值

可以不编写代码 x = x + 1，而将右边表达式中的运算符（这里是+）移到赋值运算符（ =）的前

面，从而写成 x += 1。这称为增强赋值，适用于所有标准运算符，如*、 /、 %等。

```
1 x = 'a'
2 x *= 2
3 print(x)
```

```
aa
```

图 119 序列乘法

增强赋值也可以用于其它数据类型，使用增强赋值会让代码看起来更简洁。

5.3 代码块与缩进

代码块是一组语句，可在满足条件时执行（ if 语句），可执行多次（循环），等等。代码块是通过缩进代码（即在前面加空格）来创建的。

```
1 a = 5
2 if(a == 5):
3 a=3
```

```
File "<ipython-input-9-9e7ba452e653>", line 3
  a=3
  ^
IndentationError: expected an indented block
```

图 120 错误缩进

这是一个没有缩进的例子，会报错”expected an indented block”。

```
1 a = 5
2 if(a==5):
3     a=3
```

图 121 正确缩进

这是一个正确缩进的例子。

5.4 条件与条件语句

5.4.1 if 语句中的真与假

```
1 bool([])
```

```
False
```

```
1 bool(1)
```

```
True
```

```
1 bool(True)
```

```
True
```

```
1 bool('fdsjafjdklsjfs')
```

```
True
```

图 122 布尔值

在 if 语句中，标准值 False 和 None、各种类型（包括浮点数、复数等）的数值 0、空序列（如空字符串、空元组和空列表）以及空映射（如空字典）都被视为假，而其他各种值都被视为真，包括特殊值 True。

5.4.2 单独的 if 语句

```
1 a = 2
2 if(a==2):
3     print(True)
```

```
True
```

图 123 单独的 if 语句

5.4.3 if-else 语句

```
1 a = 3
2 if(a==2):
3     print(True)
4 else:
5     print(False)
```

```
False
```

图 124 if-else

5.4.4 if-elif-else 语句

```
1 a = 5
2 if(a<2):
3     print(1)
4 elif(a>10):
5     print(2)
6 else:
7     print(3)
```

```
3
```

图 125 if-elif-else

5.4.5 嵌套的 if 语句

```
1 a = 5
2 if(a>2):
3     if(a>8):
4         print(True)
5     else:
6         if(a>6):
7             print(True)
8         else:
9             print(False)
```

```
False
```

图 126 嵌套的 if 语句

5.5 比较运算符

表 达 式	描 述
x == y	x 等于y
x < y	x小于y
x > y	x大于y
x >= y	x大于或等于y
x <= y	x小于或等于y
x != y	x不等于y
x is y	x和y是同一个对象
x is not y	x和y是不同的对象
x in y	x是容器（如序列）y的成员
x not in y	x不是容器（如序列）y的成员

图 127 常见的比较运算符

以上是 python 常见的比较运算符，下面我们简单介绍几个。

```
1 'foo' == 'foo'
```

```
True
```

```
1 a = 2
2 a ==2
```

```
True
```

图 128 “==”

==是比较两个对象的值是否相等。

```
1 a is 2
```

```
True
```

```
1 b = 2
2 a is b
```

```
True
```

```
1 c = [1]
2 d = [1]
3 print(c==d)
4 print(c is d)
```

```
True
False
```

图 129 is

is 的用法有些复杂，对于 list 这种对象，尽管它们值相同，但并不是一个对象，所以 is 的结果是 False，但对于单独一个数，分别赋予不同的变量，用 is 比较一下得到的结果是 True。一般在实践中还是要避免使用 is。

```
1 2 in [2, 4]
```

```
True
```

```
1 'a' in 'abc'
```

```
True
```

图 130 in

in 的作用是检查一个对象是否在一个序列中，这个序列可以是 list，也可以是字符串。

```
1 'a' < 'b'
```

```
True
```

```
1 [1, [2, 3]] < [1, [2, 4]]
```

```
True
```

图 131 <

大于小于号的用法与常识相符，只不过在 python 中进一步扩宽了它们的应用对象。

```
1 2<3<4
```

```
True
```

图 132 链式比较

Python 同样支持链式比较。

5.6 布尔运算符

```
1 True and False
```

```
False
```

```
1 True or False
```

```
True
```

```
1 not False
```

```
True
```

图 133 逻辑运算符

在 python 中布尔运算符和其它语言中的逻辑是一样的，包含“与”“或”“非”三种逻辑运算，分别用 and or not 来表示。

```
1 a = [1]
2 False and (a.append(2))
3 print(a)
4 False or (a.append(2))
5 print(a)
```

```
[1]
[1, 2]
```

图 134 短路逻辑

同样地，短路逻辑在 python 中一样适用。

5.7 断言

```
1 a = 3
2 assert a<2 ,'a并不小于2'
```

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
<ipython-input-37-c29d9a6bc303> in <module>
      1 a = 3
----> 2 assert a<2 ,'a并不小于2'

AssertionError: a并不小于2
```

图 135 assert 断言

当我们需要核实函数参数满足要求或为初始测试和调试提供帮助时，可以考虑使用 assert 判断，假如条件不满足的话程序会直接崩溃。

5.8 循环

5.8.1 while 循环

```
In [ ]:
1 x = 1
2 while(x<10):
3     print(x)
4     x += 1
```

```
1
2
3
4
5
6
7
8
9
```

图 136 while 语句

While 循环会一直进行，只有在括号中的条件不满足时才会退出。

5.8.2 for 循环

for 是使用频率最高的 python 关键字之一，其主要用途是用来迭代。

迭代列表与元组

```
1 a = [1, 2, 3]
2 for i in a:
3     print(i)
```

```
1
2
3
```

```
1 b = (1, 2, 3)
2 for j in b:
3     print(j)
```

```
1
2
3
```

图 137 for 循环迭代列表与元组

使用 for 循环可以输出列表或者元组中的每一个对象。

```
1 for i in range(10):
2     print(i)
```

```
0
1
2
3
4
5
6
7
8
9
```

图 138 range

鉴于我们经常需要迭代 0~n 之间的数值，python 为此发明了 range 关键字，range(n)即返回一个 0~n-1 的列表。

迭代字典

```
1 c = {'a':'1','b':'2'}
2 for key in c:
3     print(key,c[key])
```

```
a 1
b 2
```

图 139 迭代字典

直接迭代字典实际上只是在迭代其键值。

```
1 for key,value in c.items():
2     print(key,value)
```

```
a 1
b 2
```

图 140 字典的 items 方法

如果想同时迭代 key 和 value，我们可以使用字典的 items()方法。

并行迭代

```
1 names = ['anne', 'beth', 'george', 'damon']
2 ages = [12, 45, 32, 102]
3 for i,j in zip(names,ages):
4     print(i,j)
5 print(zip(names,ages))
6 print(list(zip(names,ages)))
```

```
anne 12
beth 45
george 32
damon 102
<zip object at 0x0000012146776888>
[('anne', 12), ('beth', 45), ('george', 32), ('damon', 102)]
```

图 141 zip

使用 zip 方法我们可以将两个 list 组合成一个键值对形式的 list 来进行迭代。Zip 生成的是一个可迭代对象，直接观察不到，我们可以将其转化成 list 来观察。

迭代时获取索引

```
1 names = ['anne', 'beth', 'george', 'damon']
2 for i,j in enumerate(names):
3     print(i,j)
4 print(enumerate(names))
5 print(list(enumerate(names)))
```

```
0 anne
1 beth
2 george
3 damon
<enumerate object at 0x000001214584AC78>
[(0, 'anne'), (1, 'beth'), (2, 'george'), (3, 'damon')]
```

图 142 enumerate

使用 enumerate 方法我们将一个列表的索引和值生成一个键值对序列。

反向迭代与排序后迭代

```
1 a = [5,8,3,7,6]
2 for i in sorted(a):
3     print(i)
```

```
3
5
6
7
8
```

```
1 for i in reversed(sorted(a)):
2     print(i)
```

```
8
7
6
5
3
```

图 143 反向迭代和排序后迭代

使用 sorted 方法和 reversed 方法可以对 list 进行排序和反向，然后再进行迭代。

break 与 continue

```
In [ ]:
1  x = 1
2  while(x<10):
3      print(x)
4      x += 1
5      if(x==5):
6          break
```

```
1
2
3
4
```

图 144 break

break 可以在满足一定的条件时直接跳出循环。

```
1  x = 1
2  while(x<10):
3      if(x==5):
4          x += 1
5          continue
6      print(x)
7      x += 1
```

```
1
2
3
4
6
7
8
9
```

图 145 continue

continue 可以将满足循环的本轮条件跳过，如上面并没有打印 5.

for-else 语句

```
from math import sqrt
for n in range(99, 81, -1):
    root = sqrt(n)
    if root == int(root):
        print(n)
        break
else:
    print("Didn't find it!")
```

```
Didn't find it!
```

图 146 for-else 语句

else 只有在没有执行 break 时才会执行。

5.9 列表与字典的推导

5.9.1 列表推导式

```
[x * x for x in range(10)]
```

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

图 147 简单的列表推导式

我们可以利用这种形式从一个列表创造另一个列表。

```
[x*x for x in range(10) if x%3 == 0]
```

```
[0, 9, 36, 81]
```

图 148 带条件的列表推导式

也可以设置一些条件。

```
[(x, y) for x in range(3) for y in range(3)]
```

```
[(0, 0), (0, 1), (0, 2), (1, 0), (1, 1), (1, 2), (2, 0), (2, 1), (2, 2)]
```

图 149 并行的列表推导式

也可以从多个列表创造一个列表。

5.9.2 字典推导式

```
1 squares = {i:"{} squared is {}".format(i, i**2) for i in range(10)}
```

```
1 squares
```

```
{0: '0 squared is 0',
 1: '1 squared is 1',
 2: '2 squared is 4',
 3: '3 squared is 9',
 4: '4 squared is 16',
 5: '5 squared is 25',
 6: '6 squared is 36',
 7: '7 squared is 49',
 8: '8 squared is 64',
 9: '9 squared is 81'}
```

图 150 字典推导式

在列表推导中， for 前面只有一个表达式，而在字典推导中，for 前面有两个用冒号分隔的表达式，这两个表达式分别为键及其对应的值。

5.10 占位符

```
1 pass
```

图 151 pass 占位符

是的，运行 pass 什么都没有发生，它经常作为占位符来使用，等待程序员以后再补充代码。

5.11 del

```
1 x = 1
2 del x
3 x
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-60-9633605c2fd3> in <module>
      1 x = 1
      2 del x
----> 3 x

NameError: name 'x' is not defined
```

图 152 del

del 会将变量名删除，但 del 无法删除值本身，例如下面的例子：

```
:  1 a = [1, 2]
   2 b = a
   3 del a
   4 print(b)
```

```
[1, 2]
```

图 153 del 只能删除变量名

这里仅仅是将 a 这个变量名删除了，并不会影响 b。

5.12 exec 与 eval

5.12.1 exec

```
1 exec("print('Hello, world!')")
```

```
Hello, world!
```

图 154 exec

exec 将字符串作为代码执行。

```
1 from math import sqrt
2 scope = {}
3 # 这里scope是一个命名空间
4 exec('sqrt = 1', scope)
5 sqrt(4)
```

```
2.0
```

```
1 scope['sqrt']
```

```
1
```

图 155 exec 设置命名空间

有时候 exec 执行一些赋值命令会造成命名空间混乱，因此在 exec 可以为其字符串中的变量单独设置一个命名空间。

5.12.2 eval

```
1 x = 2
2 eval('x * x')
```

```
4
```

图 156 eval

exec 是执行语句，而 eval 仅仅计算表达式的值。