第2章 数据类型

学习目标

- Python中变量的概念与基本操作
- Python的基本数据类型的概念与基本操作
  - 数字、列表、元组、字符串、字典和集合
- Python的基本运算符

2.1 概述——变量

- Python中的变量（variable）表示某值的名字，每创建一个变量，计算机就根据对应值的类型分配一定的内存空间来储存该值。
- 变量的创建与删除
  - Python通过变量赋值来创建变量。变量赋值通过等号“=”来实现，等号左边为变量名，右边为储存在变量中的值：

```
>>> a = "abc"
>>> b = a
>>> a = "def"
>>> b
'abc'
```

  - Python中的变量命名可用大小写英文、数字、“_”的任意组合，但数字不能放在开头，同时在命名时也应当注意避开Python中的保留字。
  - 删除变量可以使用del(变量名)，若要同时删除多个变量，用英文逗号隔开多个变量即可：

```
>>> del a,b
```

2.1 概述——变量

- 多重赋值
  - Python允许同时为多个变量赋予同一值：

```
>>> a = b = c = 1
>>> print(a,b,c)
1 1 1
```

  - 也可以将多个值赋予多个变量，这些值的类型可以不相同：

```
>>> a,b,c = 1,2,"python"
>>> print(a,b,c)
1 2 python
```

2.1 概述——变量

- 增量赋值
  - 通过将表达式运算符放在赋值运算符“=”的左边来实现，对于*、/、%等标准运算符都适用：

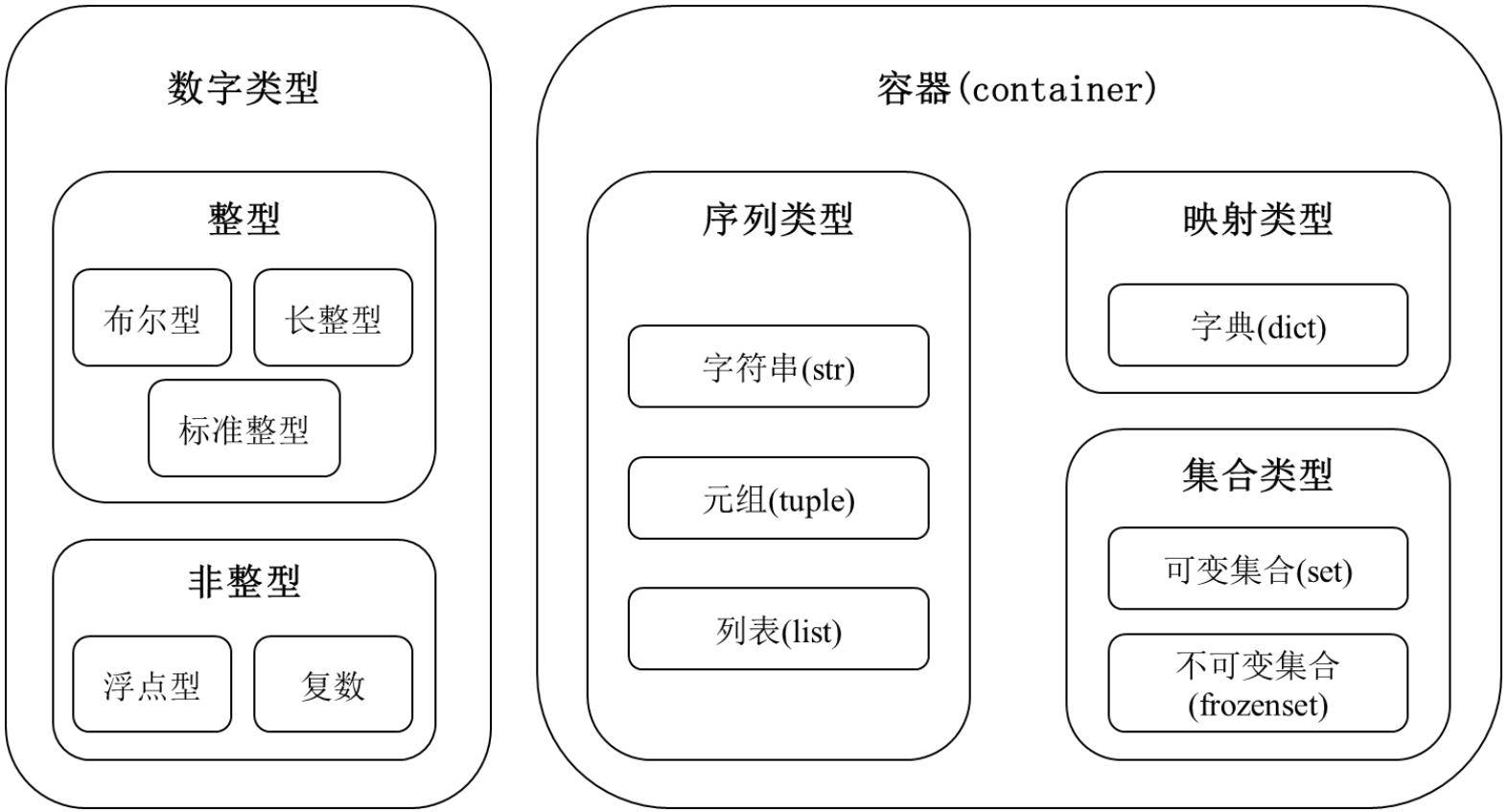
```
>>> x = 1
>>> x += 2
>>> x *= 2
>>> x
6
```

  - 对于其他数据类型也适用，只要二元运算符本身适用于这些数据类型即可：

```
>>> x = "foo"
>>> x += "bar"
>>> x *= 2
>>> x
'foobarfoobar'
```

2.1 概述——数据类型框架

- Python的数据类型框架：



2.1 概述——数据类型框架

- Python中主要有六个标准的数据类型，可分为数字、字符串（str）、列表（list）、元组（tuple）、字典（dict）、集合（set）。其中，数字、字符串、元组为不可变数据类型。而列表、字典、集合（指可变集合set，另有不可变集合（frozenset）为可变数据类型。
- Python中常见的数据结构，即相互之间存在关系的数据元素的集合都可称作容器（container）。序列、映射、集合是Python中主要的容器。
  - 最基本的容器为序列（sequence），序列中元素的位置称为索引，以0、1、2...的方式递增。字符串、列表、元组都属于序列。
  - 映射中每个元素都有一个名字，互不相同，也称为键，字典就属于映射。
  - 至于既不是序列也不是映射的容器类型，集合就是一个例子。

2.2 数字类型——分类

- Python3的数字类型分为整数型（int）、浮点型（float）和复数型（complex）。
- 整数
  - 整数可用十进制、二进制（0b或0B开头）、八进制（0o或0O开头）、十六进制（0x或0X开头）进行表示。
- 浮点数
  - 浮点数也就是小数，可近似表示任意一个实数。对于用科学记数法表示的小数，Python中用e代替10。
  - 浮点数与整数在内存中储存方式不同，整数能够进行完全精确的运算，但浮点数运算会存在小的误差：

```
>>> a = 1.1
>>> b = 2.2
>>> c = a + b
>>> c
3.3000000000000003
>>> print(c == 3.3)
False
```

2.2 数字类型——分类

- 复数
  - 复数由实数（整数、浮点数）和虚数组成。虚数单位用j表示。
  - 使用普通math模块中函数处理复数时可能会报错：

```
>>> from math import sqrt
>>> print(sqrt(-1))
Traceback (most recent call last):
  File "<pyshell#32>", line 1, in <module>
    print(sqrt(-1))
ValueError: math domain error
```

  - 处理复数时一般使用cmath模块：

```
>>> import cmath
>>> print(cmath.sqrt(-1))
1j
```

2.2 数字类型—— 相关函数

- 数字类型转换
  - 转换函数包括：

转换函数	说明
int(x)	将浮点数x转换为整数，保留其整数部分
float(x)	将整数转换为浮点数
complex(x,y)	将x作为实数部分、y作为虚数部分转化为一个复数，其中x、y为数字表达式，若虚数部分为0，可省略y不写。

  - 举例如下：

```
>>> int(1.9)
1
>>> float(2)
2.0
>>> complex(1+2,3.2)
(3+3.2j)
```

2.2 数字类型—— 相关函数

- 数学函数
  - 一些常用的数学函数：

转换函数	说明
abs(x)	取绝对值
max(x1, x2...)	返回参数x1、x2...中的最大值，参数可为序列
min(x1, x2...)	返回参数x1、x2...中的最小值，参数可为序列
pow(x, y)	返回x**y（即x的y次方）的值
round(x, n)	将浮点数x四舍五入，参数n表示舍入到小数点后第n位，可以省略不写
sqrt(x)	取平方根，x不可为复数

  - 举例如下：

```
>>> abs(-10)
10
>>> pow(2,3)
8
```

2.2 数字类型—— 相关函数

- 数学函数
  - math模块中还有许多数学函数，可通过math.函数名()来调用函数：

转换函数	说明
exp(x)	返回自然常数e的x次幂
fabs(x)	返回x的绝对值
log(x,y)	返回以y为底x的对数

  - 举例如下：

```
>>> import math
>>> math.sqrt(4)
2.0
>>> math.log(100,10)
2.0
```

2.3 列表与元组——序列通用操作

- 字符串、列表、元组都属于序列。所有类型的序列有通用的操作，包括索引（indexing）、切片（sliceing）、加法（adding）、乘法（multiplying）、审查成员资格（即检查某个元素是否在该序列内）和迭代（iteration），此外还有求序列长度、找最大元素、最小元素的内置函数。
- 索引
  - 索引指通过序列中元素的编号（即索引值）访问该元素，注意第一个元素的编号为0：

```
>>> sentence = 'Hello,world!'
>>> sentence[4]
'o'
```

  - 也可使用负数索引，此时编号由后往前递减，最后一个元素的编号为-1：

```
>>> sentence[-2]
'd'
```

  - 除了通过变量进行索引外，还可以直接通过序列或函数返回结果进行索引：

```
>>> sixth = input('Date:')[6]
Date:2020.02.10
>>> print(sixth)
2
```

2.2.1 序列通用操作——切片

- 切片
  - 索引只能访问某一个元素，而切片可以访问一定范围内连续的元素，通过[索引1:索引2]来实现：

```
>>> sentence = 'Hello,world!'
>>> sentence[6:11]
'world'
```

  - 与负数索引类似，切片也可以用负数编号表达：

```
>>> sentence[-6:-1]
'world'
```

  - 索引1、2也可以省去，如[索引1:]表示访问索引1元素及之后所有的元素，[:索引2]表示访问索引2元素之前的所有元素，[:]表示访问序列中所有元素：

```
>>> sentence[5:]
',world!'
>>> sentence[:5]
'Hello'
>>> sentence[:]
'Hello,world!'
```

2.2.1 序列通用操作——切片

- 切片
  - 步长可以用通过显式设置改变，形式为[索引1:索引2:步长]:

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

  - 步长不能取零，但可以取负数，表示从后往前提取元素。在步长为正时，索引1应小于索引2，步长为负时，索引1应大于索引2:

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

2.2.1 序列通用操作——加法与乘法

- 加法
  - 多个序列可以通过“+”连在一起，但这些序列的类型必须相同：

```
>>> 'python' + '3.0'
'python3.0'
>>> [1, 2, 3] + ['x', 'y', 'z']
[1, 2, 3, 'x', 'y', 'z']
```

- 乘法
  - 这里的乘法表示序列的重复：

```
>>> 'Hello World' * 3
'Hello WorldHello WorldHello World'
```

  - 创建占用一定空间的空列表也可使用乘法：

```
>>> [None] * 3
[None, None, None]
```

2.2.1 序列通用操作——成员资格

- 成员资格
  - 布尔运算符“in”用来判断某个条件是否为真，若为真则返回True，若为假则返回False：

```
>>> 'y' in 'python'
True
>>> 'x' in 'python'
False
```

- 其它
  - 求序列长度、最大值、最小值使用相应的len()、max()、min()函数即可：

```
>>> a = [1,2,3]
>>> len(a)
3
>>> max(a)
3
>>> min(a)
1
```

2.2.2 列表——列表操作

- 元素赋值
  - 列表能够通过元素赋值改变某个元素的值：

```
>>> item_1 = [30, 'blue-collar', 'basic.9y', 'yes', 'no']
>>> item_1[0] = 35
>>> item_1
[35, 'blue-collar', 'basic.9y', 'yes', 'no']
```

  - 但对于位置不存在的元素无法进行赋值：

```
>>> item_1[100]
Traceback (most recent call last):
  File "<pyshell#57>", line 1, in <module>
    item_1[100]
IndexError: list index out of range
```

- 元素删除

```
>>> item_2 = [39, 'services', 'university.degree', 'yes', 'yes']
>>> del(item_2[-2 : ])
>>> item_2
[39, 'services', 'university.degree']
```

2.2.2 列表——列表操作

- 分片赋值
  - 分片赋值可以用与原序列不等长的序列进行替换：

```
>>> name = list('play')
>>> name[1 : ] = list('ython')
>>> name
['p', 'y', 't', 'h', 'o', 'n']
```

  - 分片赋值语句可以在不需要替换任何原有元素的情况下插入新的元素：

```
>>> numbers = [1, 5]
>>> numbers[1 : 1] = [2, 3, 4]
>>> numbers
[1, 2, 3, 4, 5]
```

  - 通过分片赋值来删除元素也是可行的：

```
>>> numbers[1 : 4] = []
>>> numbers
[1, 5]
```

2.2.2 列表——列表方法

- 与函数不同，方法的调用离不开对象，其调用格式为：对象.方法（参数），下面是列表方法：
- 添加一个元素：append()
  - 用于在原列表后添加新的元素，每次只能添加一个元素：

```
>>> sentence = ['I', 'love', 'ba', 'na', 'na']
>>> sentence.append('!')
>>> sentence
['I', 'love', 'ba', 'na', 'na', '!']
```

- 统计元素次数：count()
  - 用于统计列表中某个元素出现的次数：

```
>>> ['I', 'love', 'ba', 'na', 'na'].count('na')
2
>>> ['I', 'love', ['ba','na'], 'na'].count('na')
1
```

2.2.2 列表——列表方法

- 扩展原列表：extend()
  - 能够使用新列表扩展原列表：

```
>>> a = [1, 2, 3]
>>> b = ['x', 'y', 'z']
>>> a.extend(b)
>>> a
[1, 2, 3, 'x', 'y', 'z']
```

  - 序列相加不会改变原序列，而extend方法修改了被扩展的原列表。
- 找出元素索引：index()

```
>>> sentence = ['I', 'love', ['ba','na'], 'na']
>>> sentence.index('na')
3
```

2.2.2 列表——列表方法

- 插入元素：insert()
  - 用于在列表某个位置插入某个对象，格式为对象.insert(索引号,被插入的对象)：

```
>>> numbers = [0, 1, 2, 3, 5]
>>> numbers.insert(4, 'four')
>>> numbers
[0, 1, 2, 3, 'four', 5]
```

- 删除元素：pop()
  - 用于删除列表中的元素，同时返回被删除元素的值，格式为对象.pop(索引号)：

```
>>> numbers.pop(4)
'four'
```

  - pop方法默认删除列表中最后一个元素：

```
>>> numbers.pop()
5
```

2.2.2 列表——列表方法

- 删除元素：pop()
  - pop方法与append方法可以实现后进先出（last-in，first-out，LIFO）原则，这也正是“栈”这种数据结构的原理，相应的两个操作称为“入栈（push）”与“出栈（pop）”。
  - 与栈相反，队列（queue）是一种有先进先出（FIFO）特征的数据结构，可以通过insert(0,对象名)和pop()结合或append()和pop(0)结合来实现，也可以使用collection模块中的deque对象：

```
>>> from collections import deque
>>> from collections import deque
>>> queue = deque(['A', 'B', 'C'])
>>> queue.append('D')
>>> queue.append('E')
>>> queue.popleft()
'A'
>>> queue.popleft()
'B'
>>> queue
deque(['C', 'D', 'E'])
```

2.2.2 列表——列表方法

- 删除元素：remove()
  - 用于删除列表中某个值的第一个匹配项：

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

- 倒置列表：reverse()
  - 用于将列表中的元素前后倒置：

```
>>> x = [1, 2, 3, 4, 5]
>>> x.reverse()
>>> x
[5, 4, 3, 2, 1]
```

  - 只限于列表，若要实现对序列的反向迭代，可以使用reversed()函数：

```
>>> x = 'abc'
>>> for i in reversed(x):
        print(i, end='')

cba
```

2.2.2 列表——列表方法

- 列表排序：sort()
  - 用于对列表进行排序：

```
>>> x = [1, 6, 3, 7, 11, 4]
>>> x.sort()
>>> x
[1, 3, 4, 6, 7, 11]
```

  - 有时需要对列表进行排序且不改变原列表，可以先复制列表得到列表副本，再对副本进行排序。但直接y=x不会产生列表副本，而是让两个变量指向同一个列表：

```
>>> x = [1, 6, 3, 7, 11, 4]
>>> y = x[ : ]
>>> y.sort()
>>> x
[1, 6, 3, 7, 11, 4]
>>> y
[1, 3, 4, 6, 7, 11]
```

```
>>> x = [1, 6, 3, 7, 11, 4]
>>> y = x
>>> y.sort()
>>> x
[1, 3, 4, 6, 7, 11]
>>> y
[1, 3, 4, 6, 7, 11]
```

2.2.2 列表——列表方法

- 列表排序：sort()
  - 该方法有两个关键字参数：key和reverse。reverse指排序规则，取值为布尔值（False或True），默认为reverse=False，即升序。key指在排序中使用的函数，以列表中每个元素分别作为参数调用该函数，得到相应的返回值作为列表中该元素的键，再根据键来进行排序，默认为key=None。
  - key也可以取自定义函数：

```
>>> strlist = ['age', 'job', 'marital', 'education']
>>> strlist.sort(key = len, reverse = True)
>>> strlist
['education', 'marital', 'age', 'job']
>>> database = [[30, 'blue-collar', 'basic.9y'], [39, 'services', 'high.school'],[4
7, 'admin.', 'basic.9y'],[25, 'services', 'basic.9y']]
>>> def takejob(x):
        return x[1]

>>> database.sort(key = takejob)
>>> database
[[47, 'admin.', 'basic.9y'], [30, 'blue-collar', 'basic.9y'], [39, 'services', 'hig
h.school'], [25, 'services', 'basic.9y']]
```

2.2.3 元组

- 元组与列表的区别在于元组一旦创建就不能修改，创建一个空元组如下：

```
>>> ()
()
```

- 逗号对于元组的创建很重要，每两个元素之间以逗号隔开：

```
>>> x = (1, 2, 3)
>>> x
(1, 2, 3)
```

- 注意在创建只有一个元素的元组时，该元素后的逗号不可省略：

```
>>> x = 2 * (3)
>>> x
6
```

```
>>> x = 2 * (3,)
>>> x
(3, 3)
```

- tuple函数与list函数类似，它能将一个序列转换为元组：

```
>>> tuple('python')
('p', 'y', 't', 'h', 'o', 'n')
```

2.2.3 元组

- 元组的操作与方法
  - 元组的基本操作即为序列的通用操作中的索引、分片等。
  - 方法有index()、count()等不修改序列的方法。
- 元组的意义
  - 元组可以在映射（和集合的成员）中当作键使用。
  - 元组作为很多内建函数和方法的返回值存在。

2.4.1 字符串概述

- 字符串是由字符组成的不可变序列，可以用单引号或双引号括起来。单引号和双引号在Python种无区别：

```
>>> print("hello world")
hello world
>>> print('hello world')
hello world
```

- 对于本身包含了双引号或单引号的字符串，创建时应使用相应的另一种引号将该字符串括起来加以区分：

```
>>> sentence_1 = '"Hello!"he said'
>>> sentence_2 = "let's go"
>>> print(sentence_1, '\n', sentence_2)
"Hello!"he said
 let's go
```

2.4.1 字符串概述

- 在普通字符串中，反斜线（\）会对特殊字符进行转义，构成转义字符：

转义字符	说明	转义字符	说明
\	续行符	\n	换行
\\	反斜杠符号	\v	纵向制表符
\’	单引号	\t	横向制表符
\”	双引号	\r	回车
\a	响铃	\f	换页
\b	退格	\o	八进制数
\e	转义	\x	十六进制数
\000	空	\other	其他的字符以普通格式输出

- 转义字符也可被用于内容中有特殊字符的字符串的表示中：

```
>>> sentence = 'let\'s play games'
>>> sentence
"let's play games"
```

2.4.1 字符串概述

- str函数
  - 用于将值转换成字符串形式，输入需转换的值返回相应的字符串形式的值，但并不改变原值的类型：

```
>>> a = 1.23
>>> b = str(a)
>>> print(type(a), type(b))
<class 'float'> <class 'str'>
```

- input函数
  - 用于获取用户的输入，无论输入值为什么类型，返回值都为字符串类型，括号中可写提示性文本或直接为空：

```
>>> age = input('enter your age:')
enter your age:25
>>> print(type(age))
<class 'str'>
```

- print函数
  - 用于输出，括号中可以为一个或多个表达式，多个表达式之间用逗号隔开：

```
>>> print('I\'m', 10+8, 'years old.')
I'm 18 years old.
```

2.4.1 字符串概述

- 其它
  - 在编写跨多行的字符串时，可以用转义字符“\n”来进行换行：

```
>>> print('first\nsecond\nthird')
first
second
third
```

  - 也可以直接在字符串中用键盘的Enter键进行换行，但注意此时字符串外的引号必须为三个双引号或三个单引号：

```
>>> print('''first
second
third''')
first
second
third
```

  - 对于字符串中有与转义字符形式相同字符的情况，可以使用反斜线“\”进行转义，或在字符串前加“r”：

```
>>> path = r'C:\Program Files\nothing'
>>> print(path)
C:\Program Files\nothing
```

2.4.2 字符串格式化

- 字符串格式化指将值按要求格式化后置于字符串中相应的位置。百分号%为字符串格式化操作符，%左边为格式化字符串，%右边为希望格式化的值（该值可以是字符串、数字、列表、元组、字典）：

```
>>> sentence = 'Hello! I\'m %s from %s.'
>>> value = ('Amy', 'Beijing')
>>> sentence%value
"Hello! I'm Amy from Beijing."
```

- 字符串格式化的转换说明符（conversion specifier）有：

格式化符号	说明	格式化符号	说明
%c	转换成字符	%x, %X	(Unsigned)转成无符号十六进制数（x / X 代表转换后的十六进制字符的大小写）
%r	优先用repr()函数进行字符串转换		
%s	优先用str()函数进行字符串转换	%e, %E	转成科学计数法（e / E控制输出e / E）
%d, %i	转成有符号十进制数	%f, %F	转成浮点数（小数部分自然截断）
%u	转成无符号十进制数	%g, %G	%e和%f / %E和%F 的简写
%o	转成无符号八进制数	%%	输出%

2.4.2 字符串格式化

- 转换说明符可以包括字段宽度和精度。字段宽度是转换后的值所保留的最小字符个数；精度对于数字转换来说是结果中应该包含的小数位数，对于字符串转换来说是转换后的值所能包含的最大字符个数。这两个参数都是整数，通过点号 “.” 分隔：

```
>>> 'Pi:%5.3f...' % 3.1415926
'Pi:3.142...'
```

- 也可以用*作为字符宽度或精度，具体值从格式化操作符%后的元组中读取：

```
>>> '%*.*s' % (10, 6, 'python 3.0')
'    python'
```

- 若格式化字符串本身内容中有%时，使用%%来避免歧义。

2.4.3 字符串方法

- 查找子字符串：find()
  - 用于在字符串中查找子字符串，返回子其中第一个字符的索引，未找到则返回-1，格式为对象.find(子字符串,起始点索引,结束点索引)：

```
>>> 'to be or not to be'.find('e o')
4
>>> 'to be or not to be'.find('e o', 8, 15)
-1
```

- 连接字符串：join()
  - 用一个元素来连接字符串列表，即在列表中的每两个字符串元素之间添加该连接元素：

```
>>> dirs = ['', 'users', 'xxx', 'desktop']
>>> '/'.join(dirs)
'/users/xxx/desktop'
```

2.4.3 字符串方法

- 去除字符串两端字符：strip()
  - 用于除去字符串两侧包含参数字符串中字符的部分，返回一个新的字符串：

```
>>> '!!**??find an apple???'.strip('!*?')
'find an apple'
>>> '!! **??find an apple???'.strip('!*?')
' **??find an apple'
```

  - 该方法从字符串两端开始寻找参数字符串中的字符，删去与之相同的字符，直到遇到不在参数字符串中的字符。
- 转化成小写字母：lower()
  - 用于将所有大写字母变为相应的小写字母：

```
>>> 'Lower ME'.lower()
'lower me'
```

2.4.3 字符串方法

- 替换字符串：replace()
  - 用指定的字符串替代原字符串中所有的匹配项，返回新的字符串：

```
>>> 'to be or not to be'.replace('to', 'To')
'To be or not To be'
```

- 分割字符串：split()
  - 是join的逆方法，用于按照指定的分隔符将字符串分割成序列：

```
>>> '/users/xxx/desktop'.split('/')
['', 'users', 'xxx', 'desktop']
```

2.5.1 字典概述

- 创建字典
  - 字典中一个键与对应的值称为一项，每个键与对应的值之间用冒号“:”隔开，每一项之间用逗号“,”隔开：

```
>>> consumer = {'Alice' : [30, 'blue-collar', 'basic.9y', 'yes'],'Bob' : [39, 'serv
ices', 'university.degree', 'yes']}
```

  - 也可使用dict函数可以通过其他映射（如字典）、或（键、值）序列对来建立字典：

```
>>> consumer = [('Alice', 30), ('Bob', 39)]
>>> dict_cons = dict(consumer)
>>> dict_cons
{'Alice': 30, 'Bob': 39}
```

  - 还可通过关键字来创建字典，关键字名即为键名，对应的值即为字典中的值：

```
>>> consumer = dict(Alice = 30, Bob = 39)
>>> consumer
{'Alice': 30, 'Bob': 39}
```

2.5.1 字典概述

- 基本操作
  - 字典的基本操作与序列的通用操作类似，但应注意在进行索引、修改值、删除项、成员资格审查时都是使用键名调用相应的值，形式为字典名[键名]：

```
>>> consumer['Bob'] = 40
>>> consumer
{'Alice': 30, 'Bob': 40}
```

  - 字典还可以直接增加新的项：

```
>>> consumer['Carol'] = 25
>>> consumer
{'Alice': 30, 'Bob': 40, 'Carol': 25}
```

  - len（字典名）返回的是项的数量：

```
>>> len(consumer)
3
```

2.5.2 格式化字符串

- 格式化操作符%后可以是元组或字典，当%后为字典时，转换说明符%后通过（键名）加s来说明元素，可以将键对应的值格式化：

```
>>> 'Alice is %(Alice)s years old.' % consumer
'Alice is 30 years old.'
```

- 通过字典格式化字符串可以在一个字典中多次格式化多个键对应的值：

```
>>> print('''Alice is %(Alice)s years old.
Bob is %(Bob)s years old.
%(Alice)s is younger than %(Bob)s''' % consumer)
Alice is 30 years old.
Bob is 40 years old.
30 is younger than 40
```

2.5.3 字典方法

- 清除字典所有项：clear()
  - 用于清除字典中所有的项，直接改变原字典，无返回值：

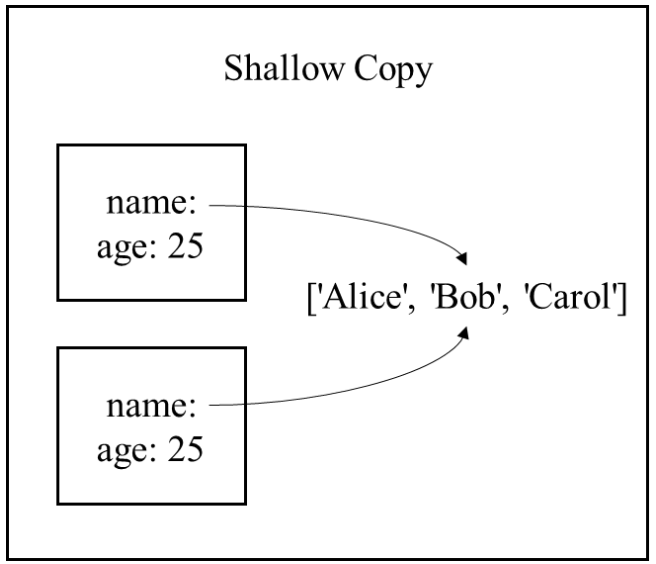
```
>>> x = {'key' : 'value'}
>>> y = x
>>> x.clear()
>>> y
{}
```

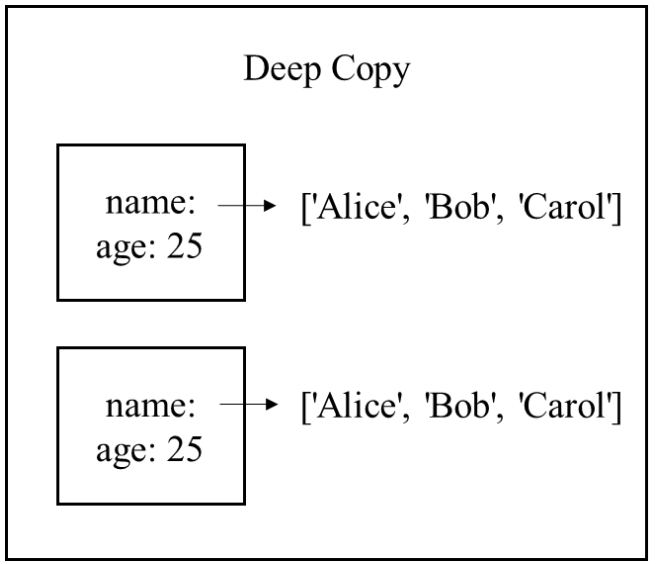
- 复制字典：copy()
  - 用于实现浅复制（shallow copy），返回一个与原字典有相同键-值对的新字典：

```
>>> x = {'name' : ['Alice', 'Bob', 'Carol'], 'age' : 25}
>>> y = x.copy()
>>> y['age'] = 30
>>> y['name'].remove('Bob')
>>> print('x:', x, '\ny:', y)
x: {'name': ['Alice', 'Carol'], 'age': 25}
y: {'name': ['Alice', 'Carol'], 'age': 30}
```

2.5.3 字典方法

- 浅复制与深复制
  - 在浅复制时，对于简单数据部分（'age'），复制对象会在内存中开辟新地址，因此改变复制对象的值不会影响原对象的值；对于子对象部分（'name'），复制对象仅引用原对象的地址，因此改变复制对象的值会改变原始对象的值。
  - 在深复制时，不管是简单数据部分还是子对象部分，复制的对象都会直接在内存中开辟新的地址空间，复制对象和原对象值的变化互不影响。





2.5.3 字典方法

- 复制字典：deepcopy()
  - 为避免修改副本时对原字典造成影响，可采用copy模块的deepcopy()函数实现深复制（deep copy）：

```
>>> from copy import deepcopy
>>> x = {'name' : ['Alice', 'Bob', 'Carol'], 'age' : 25}
>>> y = deepcopy(x)
>>> y['age'] = 30
>>> y['name'].remove('Bob')
>>> print('x:', x, '\ny:', y)
x: {'name': ['Alice', 'Bob', 'Carol'], 'age': 25}
y: {'name': ['Alice', 'Carol'], 'age': 30}
```

2.5.3 字典方法

- 使用给定的键创建字典：fromkeys()
  - 用于使用给定的键建立新的字典，值的默认值默认为None，也可自己设置。建立方式有两种：

```
>>> {}.fromkeys(['name', 'age'])
{'name': None, 'age': None}

>>> dict.fromkeys(['name', 'age'], 111)
{'name': 111, 'age': 111}
```

- 访问字典项：get()
  - 用于访问字典项，当访问的键不存在时会返回None，也可以设置默认值：

```
>>> x = {'name' : ['Alice', 'Bob', 'Carol'], 'age' : 25}
>>> x.get('job', 'no')
'no'
```

2.5.3 字典方法

- 返回字典所有项：items()
  - 将字典中所有项以迭代器的形式返回，各项的排列没有特殊顺序：

```
>>> x = {'name' : ['Alice', 'Bob', 'Carol'], 'age' : 25}
>>> x.items()
dict_items([('name', ['Alice', 'Bob', 'Carol']), ('age', 25)])
```

  - 类似地，keys()方法以可迭代对象形式返回字典中的键：

```
>>> x.keys()
dict_keys(['name', 'age'])
```

  - values()方法以可迭代对象形式返回字典中的值：

```
>>> x.values()
dict_values([['Alice', 'Bob', 'Carol'], 25])
```

2.5.3 字典方法

- 删除字典键值对：pop()和popitem()
  - pop用于在字典中删除给定键的键-值对，并返回该值：

```
>>> x = {'name' : ['Alice', 'Bob', 'Carol'], 'age' : 25, 'job' : 'services'}
>>> x.pop('age')
25
```

  - popitem用于随机删除键-值对并以元组形式返回该项：

```
>>> x.popitem()
('job', 'services')
```

- 更新字典：update()
  - 将给定字典添加到另一个字典中，若有相同的键则进行覆盖：

```
>>> x = {'name' : 'Alice', 'age' : 25}
>>> y = {'age' : 30, 'job' : 'serives'}
>>> x.update(y)
>>> x
{'name': 'Alice', 'age': 30, 'job': 'serives'}
```

2.6.1 集合概述

- 集合（set）是由序列（或其他可迭代对象）构建的，集合中的元素是不重复的、无序的，主要用于检查成员资格和消除重复元素：

```
>>> x = set('python python')
>>> x
{'p', 'y', 'o', 't', ' ', 'n', 'h'}
```

- 创建集合时使用大括号{}或set()函数：

```
>>> x = {'python', 1, 2, 2, 3}
>>> x
{3, 1, 'python', 2}

>>> x = set(['python', 1, 2, 2, 3])
>>> x
{3, 1, 'python', 2}
```

2.6.1 集合概述

- 集合的运算
  - 类似于数学中集合的运算，可以求交集（&）、并集（|）、不同时被两个集合包含的元素（^）等：

```
>>> x = {1, 2, 3, 4, 5, 6, 7}
>>> y = {4, 5, 6, 7, 8, 9,10}
>>> print('x与y的交集：', x & y, '\nx与y的并集：', x | y,
'\nx-y的集合：', x - y, '\n既不属于x也不属于y：', x ^ y,
'\n判断x是否是y的子集：', x <= y)
x与y的交集：  {4, 5, 6, 7}
x与y的并集：  {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
x-y的集合：  {1, 2, 3}
既不属于x也不属于y：  {1, 2, 3, 8, 9, 10}
判断x是否是y的子集：  False
```

2.6.1 集合概述

- frozenset不可变集合
  - 集合中不能包含其它集合，但将两个集合相加是很常用的，可使用frozenset函数将集合set转换为不可变集合：

```
>>> KPI = {'basic' : {'item1', 'item2', 'item3'}, 'changeable' : {'item4', 'item5'}}
>>> new = KPI['changeable']
>>> new.add(frozenset(KPI['basic']))
>>> new
{'item5', frozenset({'item3', 'item1', 'item2'}), 'item4'}
```

2.6.2 集合方法

- 更新集合：add()和update()
  - add方法用于向集合中添加一个元素：

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

  - update方法可同时向集合中添加多个元素，参数可以为字符串、列表、元组、字典（只会添加字典的键）等：

```
>>> x.update('string')
>>> x.update(['list_item1', 'list_item2'])
>>> x.update({'name' : 'Alice'})
>>> x
{1, 2, 3, 4, 't', 'i', 's', 'name', 'list_item2', 'list_item1', 'r', 'n', 'g'}
```

2.6.2 集合方法

- 删除元素：remove()和discard()
  - remove方法用于将指定元素从集合中移除，若指定元素不存在则会报错：

```
>>> x = {1, 2, 3}
>>> x.remove(3)
>>> x.remove(4)
Traceback (most recent call last):
  File "<pyshell#263>", line 1, in <module>
    x.remove(4)
KeyError: 4
```

  - discard方法也用于移除指定元素，但若该指定元素不存在也不会报错：

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

2.7 基本运算符——算术运算符与比较运算符

- Python中的基本运算符分为算术运算符、比较运算符、赋值运算符等多种类型。
- Python中的算术运算符：

运算符	说明
+	加法，将运算符两侧的值相加
-	减法，左侧操作数减去右侧操作数
*	乘法，将运算符两侧的值相乘
/	除法，左侧操作数除右侧操作数
%	求模，即求左侧操作数除左侧操作数后的余数
**	求幂，即求左侧操作数的右侧操作数次幂
//	地板除，将小数点后的位数截除、只保留整数部分

- Python中的比较运算符：

运算符	说明
==	判断两侧操作数的值是否相等，如果是，结果为真
!=	判断两侧操作数的值是否不相等，如果不相等，结果为真
>	判断左操作数的值是否大于右操作数的值，如果是，结果为真
<	判断左操作数的值是否小于右操作数的值，如果是，结果为真
>=	判断左操作数的值是否大于或等于右操作数的值，如果是，结果为真
<=	判断左操作数的值是否小于或等于右操作数的值，如果是，结果为真

2.7 基本运算符——赋值运算符

- Python中的赋值运算符：

运算符	说明
=	赋值运算符，将右侧操作数的值赋给左侧操作数
+=	将右操作数和左操作数相加，并将结果赋给左操作数，a+=b就等于a=a+b，下列类似
-=	将左操作数减去右边操作数的结果赋给左操作数
*=	将左操作数乘右操作数的结果赋给左操作数
/=	将左操作数除右操作数的结果赋给左操作数
%=	将左右操作数取余，并将结果赋给左操作数
**=	求左侧操作数的右侧操作数次幂，并将结果赋给左操作数
//=	执行地板除操作并将结果赋给左操作数

2.7 基本运算符——其他运算符

- Python中的其他运算符：

运算符	说明
and	表示逻辑“与”，若操作符两边都为真，则结果为真
or	表示逻辑“或”，若操作符两边有一个为真，则结果为真
not	表示逻辑“非”，若操作符右侧为假，则结果为真
in	若左操作数是右操作数的成员，结果为真。
not in	若左操作数不是右操作数的成员，结果为真。
is	若两侧变量指向相同的对象，结果为真。
is not	若两侧变量指向相同的对象，结果为假
&	二进制位“与”运算符
|	二进制位“或”运算符
^	二进制位“异或”运算符
~	二进制“补码”运算符
<<	二进位向左移位运算符
>>	二进位向右移位运算符

2.7 运算符优先级表

- 当不同运算符同时出现在同一个表达式时，表达式的计算步骤有优先级（优先级越大的越先执行）：

运算符	说明	优先级
**	幂	14
~	按位补码	13
*，/，%，//	乘，除，取余，地板除	12
+，-	加法，减法	11
>>，<<	按位右移，按位左移	10
&	按位“与”	9
^	按位“异或”	8
|	按位“或”	7
<=，<，>，>=，!=，==	比较运算符	6
is，is not	标识运算符	5
in，not in	成员运算符	4
not	逻辑“非”	3
and	逻辑“与”	2
or	逻辑“或”	1