Python 程序设计基础

江西财经大学-朱文强

第四章

常用序列

常用序列

Python提供了多种常用序列，以实现强大的程序功能，如**字符串序列**、**列表**、**元组**、**集合**、**字典**等，以下将对它们进行一一介绍。

1 字符串

1.1 字符串的定义和创建

字符串是由字符组成的一个不可变序列，它支持索引、切片、重复、合并等操作。为了简化操作，**Python对“字符”和“字符串”的概念进行了统一，不再有单独的“字符”概念，所有的“字符”都被统一视为“字符串”**。

字符串的创建主要有两种方式，**第1种直接通过一对引号创建字符串对象**(对于一些特殊字符可使用转义字符)，**第2种是使用str()函数将其他类型对象转化为字符串对象**。

1.2 字符串常用方法

str.title()：以首字母大写的方式来显示每个单词，其余部分都转化为小写; 如aDa, ADA和ada都将转化为Ada

str.capitalize()：将整个字符串的第一个字母大写，其余转换为小写字母;

str.upper(): 将字符串全部改为大写;

str.lower(): 将字符串全部改为小写;

注: 在Python中, 方法都是以()结尾; 所有修改字符串的方法, 实际并不是对原字符串修改, 而是返回一个修改后的结果字符串。

1.2 字符串常用方法

str.swapcase(): 将字符串的字母大小写进行反转;

str(s): 返回任意类型s对应的字符串;

chr(u): 返回Unicode编码u对应的单个字符;

ord(c): 返回单个字符c对应的Unicode编码。

注意：在Python中，字符串以Unicode进行编码。因此，中文字符和英文字符都算是一个字符；而Python解释器则采用UTF-8来存储和传输所有字符信息。

1.2 字符串常用方法

str.startswith(str1), str.endswith(str1): 判断str字符串是否以字符串str1开始或结束, 可接受整数参数来限定检测范围, 且可以接受字符串元组作为参数。

```
>>>s = 'Beautiful is better than ugly.’

>>>s.startswith('Be')  # True

>>>s.startswith('Be',5)    # False

>>>s.startswith(‘Be’,0, 5)   # 下例可用for循环打印, no推导式

>>>import os    # 列出D:\Py目录下所有指定扩展名的图片

>>>[fn for fn in os.listdir(r'D:\Py') if fn.endswith(('.bmp', '.jpg',))]
```

1.2 字符串常用方法

str.rstrip(str1): 删除字符串末尾指定的字符，默认为空白字符(临时性删除);

message = ‘python　’

message.rstrip()

print(message);

message = message.rstrip()

注意: 可用dir(‘’)查看字符串的操作方法，使用help(‘’.method)查看字符串处理方法method的使用帮助(or dir(str))。

1.2 字符串常用方法

str.lstrip(str1):删除字符串前端指定的字符，默认为空白字符(临时性删除);

str.strip(str1): 删除字符串指定的字符, 默认为空白字符, 包括空格、制表符和换行符(临时性删除);

```
>>> '  *** SPAM * for * everyone !!! ***'.strip(' *!')
'SPAM * for * everyone'
```

1.2 字符串常用方法

str.center(len, [str1]): 在字符串两边填充指定字符str1(默认为空格, 让字符串居中显示);

“The Middle by Jimmy Eat World”.center(40)

“The Middle by Jimmy Eat World”.center(40, ’*’)

str.ljust(n, [str1])和str.rjust(n, [str1])方法: 字符串按指定长度居左或居右显示, 并填充指定字符str1, 默认为空格符。

>>> "Hello".ljust(40)

>>>”Hello”.rjust(40, ‘*’)

1.2 字符串常用方法

str.find(str1, [start], [end]), str.rfind(str1, [start], [end]): 查找str1子串在当前字符串指定范围中首次或最后一次出现的索引, 不存在则返回-1(区分大小写);

“Monty python’s Flying Circus”.find(‘Monty’)

“Monty python’s Flying Circus”.find(‘Jony’)

“With a moo here, and a moo there”.find(‘moo’, 10, 20)

1.2 字符串常用方法

str.index(str1, [start], [end]), str.rindex(str1, [start], [end]): 查找子串在当前字符串指定范围中首次或最后一次出现的索引, 不存在则抛出异常;

>>>"This is a python lesson".rindex(‘ss’) # 19

>>>"This is a python lesson".index('s’) # 3

1.2 字符串常用方法

str.replace(str1，str2，[times]): 按规定次数times，将指定子串str1替换为另一个字符串str2, 并返回替换后的结果;

“This is a cat, and it is lovely”.replace(‘is’, ‘eez’)

1.2 字符串常用方法

str.translate(str1, str2)方法: 与replace()方法类似, 但它只能进行单字符替换, 且能够同时替换多个字符;

```
table = str.maketrans(‘cs’, ‘kz’)   # 建立字符映射表
‘This is an incredible test’.translate(table)
table = ''.maketrans('0123456789', '零一二三四五六七八九')
'2020年2月8日'.translate(table)
```

1.2 字符串常用方法

str.split(str1), str.rsplit(str1): 其作用与join方法相反, 用于将字符串从左端或右端开始拆分为列表(如果没有指定分隔符str1, 将默认在**空格、制表符和换行符**处进行拆分);

‘1+2+3+4+5’.split(‘+’)

‘/user/bin/env’.split(‘/’)

‘using the default value’.split(′ ′, 2)

‘using the   default’.split(None, 1)   # None比” ”范围更广

"This\n is \na".split()   # ['This', 'is', 'a']

1.2 字符串常用方法

str.partition(str1)，str.rpartition(str1): 以指定分隔符str1将原字符串从左或从右开始分隔为3部分，即分隔符前的字符串、分隔符、分隔符后的字符串;

```
>>>"This is a python lesson".partition('is')
 # ('Th', 'is', ' is a python lesson')
>>>"This is a python lesson".rpartition('s')
# ('This is a python les', 's', 'on')
>>>"This is a python lesson".partition('isss')
# ('This is a python lesson', '', '')
```

1.2 字符串常用方法

str.join(seq): 用于合并列表、元组的元素;

seq = [‘1’, ‘2’, ‘3’, ‘4’, ‘5’]

sep = ‘+’

sep.join(seq)

dirs = ‘’, ‘user’, ‘bin’, ‘env’

sep = ‘/’

sep.join(dirs)

1.2 字符串常用方法

str.isupper(): 判定字符串是否全为大写字符;

'ERT'.isupper()

str.islower(): 判定字符串是否全为小写字符;

str.isspace(): 判定字符串是否全为空白字符;

str.isdigit(): 判定字符串是否全为数字字符(汉字数字无法识别);

str.isnumeric(): 判定字符串是否全为数字字符(汉字数字也可识别); # '五千零伍拾'.isnumeric()

str.isalpha(): 判定字符串是否只包含字符, 并且非空;

1.2 字符串常用方法

str.isalnum(): 判断字符串是否只包含数字和字母, 并且非空;

str.istitle(): 判定字符串的单词是否首字母大写;

str.isidentifier(): 判断字符串str是否为合法的Python标识符;

```
>>> from random import choice
>>> x = choice(['abcdef', 'abcabc'])
>>> count_x = x.count('e')
>>> print("The choice is", 2 if count_x >= 2 else 1)
```

1.2 字符串常用方法

str.count(str1): 计算字符串中子串str1的数量(count方法也可用于列表或元组中);

max(): 返回字符串中的最大值字符;

min(): 返回字符串中的最小值字符。

1.2 string模块的常用字符串

模块string还提供了一些常用的字符串常量，可通过:**from string import xxx**加载。

digits: ′0123456789′;

ascii_letters: 52个大小写英文字母;

ascii_lowercase: 26个小写英文字母;

ascii_uppercase: 26个大写英文字母;

punctuation: 常用的分隔符。

1.3 字符串格式化输出

Python中提供了多种方式实现字符串的格式化输出，其中**最为常见的就是百分号方式和format()方式(Python2.6版本推出)**。其中，百分号方式在第2章第3节数据类型中介绍过。

自Python2.6版本以后，Python社区都建议优先使用format()函数来进行字符串的格式化输出。

在使用format()方法时，先用一对大括号{}进行占位，{}中可以什么都不指定，按照参数的默认顺序赋值；也可以按照指定对应参数的位置或参数的名称赋值。

1.3 字符串格式化输出

format()函数的参数说明:

[[fill]align][sign][#][0][width][,][.precision][type]

- fill: 设置空白处填充的字符, 默认为空格。
- align: 设置对齐方式(结合宽度来使用, 其中<为左对齐符号, >为右对齐符号, ^为居中对齐符号)。
- sign:有无符号数字。其中，+表示正数前加+，-表示正数前无符号, 空格表示正数前显示空格。
- #: 对于二进制、八进制、十六进制，显示前面的0b、0o、0x, 否则不显示。

1.3 字符串格式化输出

format()函数的参数说明:

- ，：为数字添加逗号分隔符，适用于大数表示，如: 1,000,000。
- width: 格式化所占宽度。
- .precision: 小数位保留的精度, 小数点不可省略。
- type: 格式化类型, s为字符串, b为二进制整数, c为字符, d为10进制整数，o为八进制整数，x 为十六进制整数，e为科学计数法, f为浮点数, g自动在e和f之间转化, %为百分比。

1.3 字符串格式化输出

最常用方式(自动位置对齐)

>>>“{}, {} and {}”.format(‘first’, ‘second’, ‘third’)

序号指定

>>>“{0} {1} {1} {2}”.format (‘first’, ‘second’, ‘third’)

变量名指定(关键字方式)

>>>“The number is {num: f}”.format(num = 42)

>>>“The number is {num: b}”.format(num = 42)

>>>“The number is {num:4.2f}”.format(num = 42.234)

1.3 字符串格式化输出

>>>“The number is {num: o}”.format(num = 4)

o，x分别对应八进制和十六进制(必须是整数)，e为指数形式, %为百分位形式

如果需要在format输出中使用花括号, 则需要在外部添加两个花括号括起。

>>> “alpha{{{}, {}}}{}{}".format("a", "b", "c", "d")

>>>print(‘{nam!s} {nam!r} {nam!a}’.format(nam=‘joe\n’))

注意: 需要控制输出格式时, 需加冒号隔开, 控制符号的顺序不能错。

1.3 字符串格式化输出

在设置了字段数据类型后，我们还可以设置字段的宽度，字段宽度用整数表示。

花括号内有num, 则num必须在圆括号内定义

>>>”{num:10}”.format(num=3)

‘ 3’

>>> "{name:10}".format(name='Bob')

‘Bob ’

1.3 字符串格式化输出

实数的精度也是用整数表示，但需要在它前面加上一个小数点。

```
>>> from math import *
>>> "Pi is {pi:.2f}".format(pi=pi)
'Pi is 3.14‘
# 10为宽度, 2为精度
>>> “Pi is {pi:10.2f}”.format(pi=pi)
'Pi is       3.14‘
```

1.3 字符串格式化输出

实际上，也可对其他类型指定精度，但这种情形不常见。

>>> “{:.5}”.format(“Guido van Rossum”) #5为精度

'Guido‘

>>> “{:5}”.format(“Guido van Rossum”) #5为宽度

'Guido van Rossum‘

>>> "The num is {num:5.2}".format(num='Guido van Rossom')

'The num is Gu '

1.3 字符串格式化输出

在指定宽度和精度的数前面，可添加一个标志来指定数据对齐方式和填充方式。其中<、^、>分别表示左对齐(默认)、居中和右对齐。

```
>>> '{:<10.2f}'.format(pi)
'3.14      '
>>> '{:^10.2f}'.format(pi)
'   3.14   '
>>> '{:>10.2f}'.format(pi)
'      3.14'
```

1.3 字符串格式化输出

可以使用填充字符来扩充对齐说明符，而不是使用默认的空格符来填充(要有对齐符，才能使用填充字符)。

0为填充字符

```
>>> ‘{:0>10.2f}’.format(pi)

'0000003.14‘

>>> "{:$^15}".format(" WIN BIG ")

'$$$ WIN BIG $$$'

>>> "{:^15}".format(" WIN BIG ")

'  WIN BIG    '
```

1.3 字符串格式化输出

还可以设定整数的千位分隔符(目前只能设定为逗号和其他几个有限的符号)。

```
>>> "One googol is {:,}".format(10**100)
'One googol is 10,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000'
```

1.3 字符串格式化输出

f关键字: format()函数的功能简化版 (3.6以后提供)。

```
>>> name = "alice"
>>> age = 21
>>> print(f"{name.title()} age is {age}")
>>> print(f"{name.title()} age is {age:2.1f}")
Alice age is 21.0
>>> print(f"{name.title():<10} age is {age+1:2.1f}")
Alice      age is 22.0
```

1.3 字符串格式化输出

```
>>> firstname = 'ada'
>>> lastname = 'lovelace'
>>> fullname = f'{firstname} {lastname}'
>>> message = f'Hello, {fullname.title()}!'
>>> print(message)
>>> print(f'{10**10:,}')
>>> num = 20
>>> print(f'{num+1:=^10.1f}')  # 回到九九乘法表例子
```

2 列表

2.1 列表的定义、创建和删除

列表是Python程序开发中应用最广泛的一种基本序列结构。

列表是一种有序可变序列，列表的所有元素放在一对中括号“[]”中，并使用逗号隔开，元素的数据类型可以不同。

列表的创建主要有两种方式，**第一种是直接通过一对方括号创建列表对象，第二种是使用list()函数将元组、range对象、字符串或其他类型的可迭代对象转换为列表**。

当不再使用列表时，可**通过del命令删除列表**，删除后的列表不可再调用，如果调用将会报错并提示列表没有被定义。

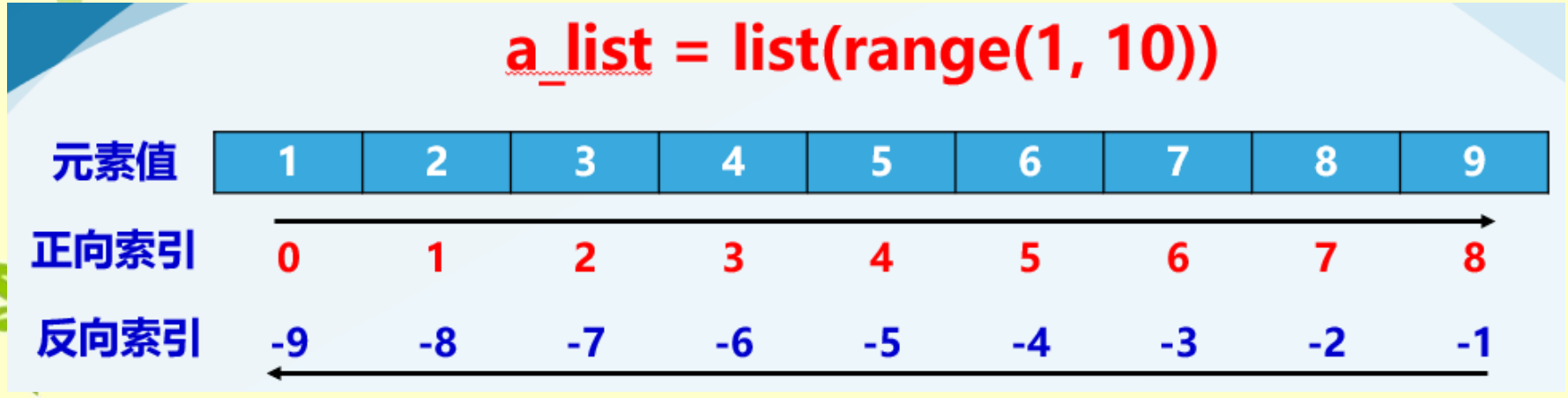
2.1 列表的定义、创建和删除

```
>>>la = []     # 创建空列表
>>>lb = [20, "张三", 177.6]
>>>lc = list(range(10))
>>>print(la)
>>>print(lb)
>>>print(lc)
```

2.2 列表元素的访问

创建列表时，将会开辟一块连续的空间，用于存放列表元素的引用，**每个元素被分配一个序号即元素的位置(也叫索引)**。

索引有两种方式，正向索引和反向索引。正向索引是索引值从 0 开始，从左到右不断递增。反向索引是从最后一个元素开始计数，此时，索引值从-1开始，从右到左不断递减(这可以帮助你不知列表长度的情况下访问最后的元素)。



2.2 列表元素的访问

```
>>>la = list(range(2, 10))
>>>print(la)
>>>print(la[6])
>>>print(la[-2])
>>> ls = [452, 'bat', [10, 'cs'], 425]
>>> ls[2][-1][0]
>>>for i in ls:
        print(i)
```

2.3 列表的切片操作

通过列表的索引可以访问列表的某一个元素，而列表的切片操作则可以同时访问列表中的多个元素。**列表的切片操作是指从一个列表中，根据位置特点获取部分元素，然后将这些元素组合成一个子列表返回**。其语法结构如下：

列表对象 [start: end: step]

其中，参数start表示起始位置索引，省略时表示包含end前的所有元素；end表示结束位置索引，但结果不包含结束位置对应的元素，省略时表示包含start后的所有元素；step表示是步长，默认为1，步长可以是正数也可以是负数，正数表示切片从左到右，负数表示切片从右到左。

2.3 列表的切片操作

```
>>>tag = '<a href = http://www.python.org>python web</a>'

>>>website = tag[9:31]

>>>print(website)

>>>website = tag[9: -15]

>>>print(website)
```

2.3 列表的切片操作

1、如果切片始于列表开头, 可省略第一个索引。

```
nums = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
it  = nums[:3]
print(it)
```

2、如果切片终于列表结尾, 可省略第二个索引。

```
it  = nums[3:]
print(it)
it= nums[:]    # 不管有无索引, 列表切片至少要有一个冒号
```

2.3 列表的切片操作

3、切片的步长

在执行切片操作时，**默认的步长为1，即每次向后移动一位，但也可以显式的设定步长**。

```
nums = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

its  = nums[3:6:1]

its = nums[::2]   # list(nums)[::-1] 反转

its = nums[11:0:-2]
```

注意：步长为正时，切片第一个索引必须要比第二个索引小。步长为负时，则刚好相反。

2.4 列表的常用方法及应用

修改列表元素的值和访问列表元素值的语法类似，要**先指定列表名和要修改的元素索引，然后指定该元素的新值**。

```
motorcycles = [‘honda’, ‘yamaha’, ‘suzuki’]

print(motorcycles)

motorcycles[0] = ‘ducati’

print(motorcycles)
```

2.4 列表的常用方法及应用

1、在列表末尾添加元素

在列表中添加新元素时，最简单的方式是**使用append()方法将元素附到列表末尾**。

```
motorcycles = []

motorcycles.append(‘honda’)

motorcycles.append(‘yamaha’)

motorcycles.append(‘suzuki’)

print(motorcycles)
```

2.4 列表的常用方法及应用

2、在列表中插入元素

使用**insert()方法可以在列表的任何位置添加新元素**，但需要指定新元素的索引和值(因为涉及到元素移动，insert方法通常比append方法计算代价更高)。

```
motorcycles = [‘honda’, ‘yamaha’, ‘suzuki’]

motorcycles.insert(1, ‘ducati’)

print(motocycles)
```

2.4 列表的常用方法及应用

1、使用del语句删除列表元素

del语句可以删除列表中的元素，前提是需要指定元素的索引位置。

```
motorcycles = [‘honda’, ‘yamaha’, ‘suzuki’]  # a = motorcycles

del motorcycles[1]      #del motorcycles[0: 1]

print(motorcycles)      # a
```

注意：del语句还可用于删除字典元素和变量。删除变量实际上是删除变量名，当一个对象值不存在引用时，Python将会自动回收对象值所占用的内存空间。

2.4 列表的常用方法及应用

2、使用pop()方法删除列表元素

pop()方法可以删除列表末尾的元素，并返回该元素的值，其遵循栈的后进先出的原理(与append方法对应)。

```
motorcycles = [‘honda’, ‘yamaha’, ‘suzuki’]

motorcycles.append(90)

cars = motorcycles.pop()

print(motorcycles)

print(cars)
```

2.4 列表的常用方法及应用

3、使用pop()方法弹出任意位置元素

实际上，**可以使用pop()弹出列表任意位置的元素，只需要在括号中指定元素索引即可**。

```
motorcycles = [‘honda’, ‘yamaha’, ‘suzuki’, ‘ducati’]

car = motorcycles.pop(2)

print(motorcycles)

print(car)
```

注意: del语句不会返回元素的值，但pop()方法会。

2.4 列表的常用方法及应用

4、使用remove(value)方法删除元素

如果你知道列表元素的值，但不知道它的位置，这时候可以使用remove()方法来删除它。

```
motorcycles = [‘honda’, ‘yamaha’, ‘suzuki’, ‘ducati’, ‘yamaha’]

print(motorcycles)

motorcycles.remove(‘yamaha’)

print(motorcycles)
```

注意: remove()方法只删除第一个指定的值。

2.4 列表的常用方法及应用

5、通过切片修改列表

```
>>>name = list('Perl')
>>>name[1:] = list('ython')   #这里可以不使用list()方法
>>>name
>>>nums = [1, 5]  # nums[2:3] = ''
>>>nums[1:1] = [2, 3, 3, 4]  # nums[2] = 20
>>>nums    # nums[0:6:2] = 'abc' (左右序列长度要一致)
>>>nums[2:3] = []       # del nums[0:6:2]
```

2.4 列表的常用方法及应用

在Python中, **可以使用加法运算符来对列表进行相加。**

```
>>>a = [1, 3, 5]    # a, b = (1, 2, 3), (4, 5, 6), a += b
>>>b = [2, 4, 5]
>>>c = a + b
>>>c.sort()
>>>d = a + ‘world!’  # 报错
```

注意：必须同类型序列才能相加，如字符串可以和字符串相加，但列表不能和字符串直接相加。

2.4 列表的常用方法及应用

将列表与数x相乘，将可以重复产生x个列表。

```
a = [1, 3, 5]

b = a *5

c = ‘Hello’

c *= 5

s = [None] * 10
```

2.4 列表的常用方法及应用

使用sort()方法可对列表进行排序，默认排序方式为升序排列。

```
cars = [‘bmw’, ‘audi’, ‘toyota’, ‘subaru’, ‘lexus’]

cars.sort()

print(cars)
```

注意: sort()方法为永久性排序。

2.4 列表的常用方法及应用

使用sort()方法对列表进行排序时，还可通过reverse或key参数(使用某个函数)来设置列表的排列方式。

```
nums = [1, 2, 3, 9, 7, 6, 5]

nums.sort(reverse = True)

print(nums)

words = [‘ada’, ‘Dlone’, ‘a’, ‘An’, ‘dabc’, ‘cpack’]   #key=str.lower

words.sort(key=len)      #reverse和key参数可以同时使用
```

注意：使用sort()方法为列表排序时，参与排序的元素类型必须保持一致。使用key参数时，注意列表元素的位置关系(稳定排序)。

2.4 列表的常用方法及应用

使用sort()方法还可对多维列表进行排序，并指定排序的元素列。

```
>>>scores = [['seal', 91, 87], ['bob', 89, 68], ['ada', 90, 72],
        ['tom', 82, 87], ['smith', 76, 59]]
# 默认对第0列元素进行对比
# 这里指定为按第1列元素进行排序, 还可以求平均分数
>>>scores.sort(key=lambda x: x[1])  # lambda函数后续会讲
```

2.4 列表的常用方法及应用

要**反转列表元素的排列顺序, 可使用方法reverse()方法**。

```
cars = [‘bmw’, ‘audi’, ‘toyota’, ‘subaru’, ‘lexus’]

print(cars)

cars.reverse()

print(cars)

"hello"[::-1]    #  'olleh' 使用切片反转列表
```

注意：reverse()方法不是对列表元素进行排序，而是按照原有的顺序进行反转, 但对列表的修改也是永久性的。

2.4 列表的常用方法及应用

使用clear()方法可清空列表的元素。

```
cars = 'audi'

lc = list(cars)

lc.clear()    #注意clear()方法和del语句作用的区别
```

2.4 列表的常用方法及应用

使用copy()方法可复制列表的元素。

```
a = [1, 2, 3]

b = a

b[1] = 4

c = a.copy()

c[1] = 10

print(a)                #如果列表有嵌套, 则可使用deepcopy()方法
```

注意：copy()方法和切片复制类似, 都是浅复制(只复制不可变数据类型, 可变数据类型如列表, 则只复制引用)

2.4 列表的常用方法及应用

使用count()方法可计算列表的重复元素数量，同时，count()方法还可用于字符串。

a = [‘to’, ‘be’, ‘is’, ‘be’, ‘to’]

x = a.count(‘be’)

‘This is a deisc’.count(‘is’)

2.4 列表的常用方法及应用

使用index(value, start, end)方法可在列表中查找指定value值第一次出现的位置, start为查找起始位置, 默认为0, end指定查找结束位置。 index()方法还可用于字符串。如果不存在该元素, 则会返回异常提示信息。

a = [‘to’, ‘be’, ‘is’, ‘be’, ‘to’]

x = a.index(‘be’)

‘This is a deisc’.index(‘is’)

a.index(′be′, 2,)

2.4 列表的常用方法及应用

extend(iterable)方法可用一个列表来扩展另一个列表，并且比'+'拼接效率高(+号拼接是使用副本来创建新列表，extend()和append()方法是就地扩展)。

```
a = [1, 2, 3]

b = [4, 5, 6]

c = [1, 2, 3]

a = a + b   # 与a += b不等价, 后者是就地复制, 与extend()等价

c.extend(a)    #c.extend((1, 2, 3)), 但元组的加法都是复制
```

2.4 列表的常用方法及应用

要保留序列元素原来的排列顺序, 同时以特定的顺序返回列表, 可使用函数sorted(list)。

```
cars = [‘bmw’, ‘audi’, ‘toyota’, ‘subaru’, ‘lexus’]

print(cars)

print(sorted(cars))

print(sorted(cars, reverse = True))
```

注意: sorted()函数也可以接受reverse, len, key等参数。

2.4 列表的常用方法及应用

```
>>> x = ['aaa', 'ab', 'aa', 'c', 'abcde']
# 先按长度排序, 长度一样则正常升序排列
>>> sorted(x, key=lambda item: (len(item), item))
['c', 'aa', 'ab', 'aaa', 'abcde']
```

2.4 列表的常用方法及应用

reversed(list)函数返回列表a翻转后的迭代对象(不影响a)。

```
>>>a = [5, 4, 2, 5, 3, 1]

>>>b = list(reversed(a))

>>>b

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

注意: reversed()函数不接受len等参数。

2.4 列表的常用方法及应用

使用函数list(seq)可将字符串、元组或其他可迭代对象类型的数据转换为列表。

```
cars = 'audi'

lc = list(cars)

print(lc)

''.join(lc)
```

''.join()方法, 可将字符列表转换为字符串

2.5 序列的常用操作及应用

使用函数len(seq)可快速获悉列表的长度。

cars = [‘bmw’, ‘audi’, ‘toyota’, ‘subaru’, ‘lexus’]

len(cars)

2.5 序列的常用操作及应用

使用函数max(seq)可快速获得序列的最大元素，也可指定key参数。

```
>>>numbers = [100, 23, 78]
>>>max(numbers)   # 也可用于字符串
>>>max([‘aa’, ‘abc’], key=len)     #  调用len()函数
'abc'
```

2.5 序列的常用操作及应用

使用函数min(seq)可快速获得序列的最小元素，也可指定key参数。

```
>>>numbers = [100, 23, 78] # 对字符串序列同样有效
>>>min(numbers)
>>>min(19, 12, 13, 35)
```

2.5 序列的常用操作及应用

使用函数sum(seq)可快速获得序列(元组，集合，range对象，map对象等可迭代对象)的元素总和。

```
>>>numbers = [1, 2, 3, 4, 5]  # 对元组和集合同样有效
>>>sum(numbers) # 15
```

注意：这里列表的常用方法如min，max，len等，都可用于元组, 字符串, 集合, range对象, map对象等可迭代对象。

2.5 序列的常用操作及应用

zip(list1，list2，…，listn)，生成1个zip对象，其每个元素为列表中对应位置元素形成的元组。

```
names = ['Seal', 'John', 'Dan', 'Tom']

ages = [12, 15, 21, 22]

talls = [130, 150, 182, 176]

for name, age, tall in zip(names, ages, talls):

    print(name + " is " + str(age) + " years old. " +

    "And  he is " + str(tall) + "cm.")
```

2.5 序列的常用操作及应用

当序列长度不同时，函数zip()将在最短的序列用完后停止拼接。

```
>>> x = [1, 2, 3]
>>> x2 = [1, 4, 9]
>>> x3 = [1, 8, 27]
>>> list(zip(x, x2, x3))
[(1, 1, 1), (2, 4, 8), (3, 9, 27)]
>>> list(zip('abcde', [1, 2, 3]))  # [('a', 1), ('b', 2), ('c', 3)]
```

2.5 序列的常用操作及应用

enumerate()函数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。

```
>>>seasons = ['Spring', 'Summer', 'Fall', 'Winter']
>>>list(enumerate(seasons))
[(0, 'Spring'), (1, 'Summer'), (2, 'Fall'), (3, 'Winter')]
>>>for index, season in enumerate(seasons):
    print(index, searson)
```

2.5 序列的常用操作及应用

标准模块bisect实现了对列表的二分查找和已排序列表的插值(但要求列表本身已经排序好)。

```
>>>import bisect
>>>c = [1, 2, 3, 4, 4, 5, 7]
>>>bisect.bisect(c, 5)  # 返回元素5的对应位置(不是索引)
6
>>>bisect.insort(c, 6)     # 插入元素6
>>>c          # [1, 2, 3, 4, 4, 5, 6, 7]
```

2.6 列表推导式

列表推导式是利用for循环从已有序列中快速生成满足特定需求的列表。**列表推导式在逻辑上相当于一个循环，只是形式更加简洁**。

语法结构: [表达式 for 表达式中的变量 in 已有序列 if 过滤条件]

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

>>>print(squares)
```

2.6 列表推导式

首先要指定一个描述性的列表名; 然后, 指定一个左方括号, **并定义一个表达式**, 用于生成存储到列表的值; 接下来, **编写一个for循环, 用于给表达式提供值**, 再加上右方括号。

列表解析使用for循环时, **还可以与if语句进行融合，甚至多个for语句进行融合**。

```
>>> square = [x*x for x in range(10) if x%3 == 0]
>>> print(square)
>>> xy = [(x, y) for x in range(3) for y in range(3)]
>>> print(xy)
```

2.6 列表推导式

```
>>>la = [(1, 2, 3), (4, 5, 6), (7, 8, 9)]

>>>flat_la = [x for tup_a in la for x in tup_a]

>>>flat_la # [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

2.7 列表小练习

```
>>> import random  # 随机生成5个可重复元素的列表
>>> num = random.choices(range(1, 10), k=5)
>>> num  # 如果用Java或C语言来写, 该怎么写?
[5, 4, 2, 9, 1]    # 生成这5个数字所能组成的最大数
```

2.7 列表小练习

```
>>> int(''.join(sorted(map(str, num), reverse=True)))
95421
```

2.7 列表小练习

```
>>>x = [randint(1, 100) for i in range(20)]

>>>x

# [85, 9, 64, 68, 82, 40, 71, 16, 29, 42, 34, 4, 79, 85, 7, 66, 3, 100, 36, 74]

# 对x的元素进行分类, 奇数放在前面, 偶数放在后面
```

2.7 列表小练习

```
>>>sorted(x, key=lambda it:it%2==0)

# [85, 9, 71, 29, 79, 85, 7, 3, 64, 68, 82, 40, 16, 42, 34, 4, 66,
100, 36, 74]
```

2.8 列表使用的常见错误

1、索引越界

```
cars = [‘bmw’, ‘audi’, ‘toyota’, ‘subaru’, ‘lexus’]

print(cars[5])
```

2、空列表元素引用

```
cars = []

print(cars[0])

print(cars[-1])
```

提示：当发生索引错误时，可尝试将列表的长度打印出来，来判断索引是否越界。

3 元组

3.1 元组的定义、创建和删除

元组属于不可变序列，一旦创建，不可修改其中的元素。元组中的元素放在一对圆括号“()”中，并用逗号分隔，其元素类型可以不同。

元组的创建主要有两种方式，**第一种是直接通过一对圆括号创建元组对象，第二种使用tuple()函数将列表、range对象、字符串或其他类型的可迭代对象转换为元组**。

当不再使用元组时，可通过del命令删除整个元组，删除后的元组将不可再调用，但要注意元组的元素是不可单独删除的。

3.1 元组的定义、创建和删除

```
>>>ta = ()      # 创建空元组
>>>tb = (20, "张三", 177.5)
>>>tc = tuple(range(10))
# 当元组中只包含一个元素时, 元素后面的逗号不能省略
>>>td = ("A",)
>>>print(tb[2])  # a, b = (1, 2, 3), (4, 5, 6), a += b
>>>a = (1, [2, 3, ], 5)
>>>a[1].append(4)        # (1, [2, 3, 4], 5)
```

3.2 元组与列表的异同

元组和列表非常相似, 它们的**相同之处**为:

(1)二者都属于可迭代对象, 支持索引和切片操作;

(2)二者都支持加(+)乘(*)运算;

(3)二者都支持一些常见的序列处理函数，例如len()、max()、min()等;

(4)二者之间可相互转化，可使用tuple()将列表转化为元组, list()将元组转化为列表。

3.2 元组与列表的异同

元组和列表的**区别**为:

(1)元组中的数据一旦定义就不允许更改，而列表中的数据可以任意修改;

(2)元组没有append()、 extend()和insert()等方法, 无法向元组中添加元素;

(3)元组没有remove()和pop()方法，也无法对元组元素进行del操作, 不能从元组中删除元素, 但可以删除整个元组。

3.2 元组与列表的异同

元组相对于列表的**优势**为:

(1)元组的操作速度比列表更快;

(2)元组对不需要改变的数据进行“写保护”，这使得数据更加安全。

3.3 生成器推导式(进阶)

推导式只适用于列表、字典和集合，**元组没有推导式**。**尝试通过已有序列快速生成满足特定需求的元组时，产生的是一个生成器对象**。生成器推导式的写法与列表推导式非常类似。

语法形式: (表达式 for 表达式中的变量 in 已有序列 if 过滤条件……)

3.3 生成器推导式(进阶)

生成器是用来创建Python序列的一个对象。**使用它可以迭代出庞大的序列，而且不需要在内存创建和存储整个序列**。它的工作方式是每次处理一个对象，而不是一次性处理和构造整个数据结构。每次迭代生成器时，它会记录上一次调用的位置，并且返回下一个值。可通过tuple()、list()等函数将其转化为元组或列表。

可通过生成器对象的__next__()方法或者系统的next()方法逐个访问其中的元素(当读取完最后一个元素后，将不能再往下读取，或往回读取)。

3.3 生成器推导式(进阶)

```
ga = (i * i for i in range(1, 10))
print(ga)        # 产生一个生成器对象
#  通过生成器对象的__next__()方法逐个访问其中的元素
print(ga.__next__())
# 通过next()函数逐个访问其中的元素
print(next(ga))
for item in ga:          # 从第3个元素开始循环
    print(item)
# print(ga.__next__())        # 报错: StopIteration
```

3.4 序列小练习

传说大臣西塔发明了国际象棋而使国王十分高兴, 他决定要重赏西塔。 # 3种方式 (18446744073709551615)

西塔说："我不要你的重赏, 陛下, 只要你在我的棋盘上赏一些麦子就行了。 在棋盘的第1个格放1粒, 在第2个格子里放2粒, 在第3个格子里放4粒, 在第4个格子里放8粒。 依此类推, 以后每一个格子里放的麦粒数都是前一个格子里放的麦粒数的2倍, 直到放满第64个格子就行了"。

请计算国王应该支付给大臣的总麦粒数量。

4 集合

4.1 集合的定义、创建和删除

前面介绍的**列表、元组、字符串**这几个常见的数据结构实际上都属于序列，**它们的元素都是有顺序的，可以通过索引访问元素，也可以支持切片操作，同时这几种数据结构的元素是可以重复的**。

而在实际应用中，当**数据内容不能存在重复时，就需要借助于集合来实现**。

4.1 集合的定义、创建和删除

集合是无序可变容器，集合中的元素放在一对大括号“{}”中，并用逗号分隔，元素类型可以不同，但集合中的元素不能重复，**且只能是固定数据类型**，如整数、浮点数、字符串、元组等；列表、字典和集合本身都是可变数据类型，不能作为集合的元素出现。

Python编译器**界定固定数据类型与否主要考察类型是否能够进行哈希运算**，能够进行哈希运算的类型都可作为集合元素。

4.1 集合的定义、创建和删除

集合的创建主要有两种方式，第一种是**直接通过一对大括号{}包裹元素创建集合对象**，第二种是**使用set()函数将列表、range对象、字符串或其他类型的可迭代对象转换为集合**，此时会自动删除其中的重复元素。

当集合不再使用时，可通过del命令删除集合，删除后集合便不可再调用。

注意：不能直接通过a={}创建空集合，此时创建的是一个空字典(字典的概念后续会进行介绍)

4.1 集合的定义、创建和删除

```
seta = {20, "张三", 177.5}

setb = set()    # 创建空集合

setc = set(range(10))

setd= set('hello')

sete = {}

print(setb)      #   set()

print(setd)      #    {'o', 'e', 'h', 'l'}

print(type(sete))      #    <class 'dict'>
```

4.2 集合运算

Python中集合支持使用简单的运算符进行交集、并集、差集、对称差集等运算, 同时也提供了对应的操作函数。

```
>>> a = {1, 3, 5}
>>> b = {2, 4, 5}
>>> a.union(b)     #{1, 2, 3, 4, 5}  并集, 等价于a | b
>>> a & b  #  {5} 交集, 等价于a.intersection(b)
>>> a – b   # {1, 3}  差集, 等价于a.difference(b)
>>> a ^ b   #    {1, 2, 3, 4}对称差集 a.symmetric_difference(b)
```

4.2 集合运算

```
>>> a = {1, 2, 3}

>>> b = {1, 2, 5}

>>> c = {1, 2, 3, 4}   # 集合还支持<, >, ==, <=, >=操作

>>> a <= b   # False  a.issubset(b)

>>> a <= c   # True   c.issuperset(a)
```

注意: 集合不支持下标读取元素, 如要读取其中的元素,可将其转化为列表；使用in关键字检查某个元素是否在列表中(for循环除外)，效率比其检查集合和字典要慢，因为列表是逐个扫描的, 集合和字典是基于哈希表的。

4.3 集合的常用方法

集合中的元素是无序可变不重复的, 不支持索引、切片等操作。**常用的集合操作方法如下。**

S.add(x): 如果数据x不在集合S中, 则添加x到S;

S.clear(): 清空集合S;

S.copy(): 返回集合S的一个副本;

S.pop(): 随机返回并删除S的一个元素;

S.discard(x): 如果数据x在集合S中, 则移除x;

S.remove(x): 从集合S中删除x, 不存在则报错;

4.3 集合的常用方法

S.update(S1): 将集合S的内容设置为S和S1的并集;

len(S): 返回集合S的元素数量;

x in S: 判断数据x是否在集合S中。 #x not in S

```
>>>seta = {1, 8, 5, "A", 9}
>>>seta.add(10)      # 添加元素10
>>>seta.add(8)       # 元素8已存在, 集合没有变化
# update()传的是可迭代对象, 可以是列表
>>>seta.update((2, 4, 5))
```

4.3 集合的常用方法

```
>>>setb = seta.copy()     # 集合的复制
>>>print(setb.pop())      # 随机弹出集合的一个元素
>>>seta.remove(19)    # 元素19不存在, 报错
>>>seta.discard(19)   # 返回None, 不报错
```

4.4 集合推导式

集合推导式写法类似于列表推导式，只不过集合推导式是用一对大括号表示，而不是一对中括号，而且使用集合推导式时会自动去除结果中的重复元素。

语法形式: {表达式 for变量 in 已有序列 if 过滤条件}

```
# 随机生成多个不重复的[1, 500]之间的整数, sample()
>>>x = {random.randint(1, 500) for i in range(20)}
#  获得x, y列表所有元素和的集合, 且丢弃重复元素
>>>setf = {x + y for x in [2, 4, 6, 8] for y in [1, 3, 5]}
```

4.5 集合运算(小例子)

```
>>>hash('a')
>>> s = {1, 2, 3, 45, 'a'} # set()函数会自动去重
>>> w = set('apple')      # 空集合的生成使用set()函数
>>> 'a' in w   # True
>>> len(lst) == len(set(lst))   # 判断lst是否存在重复元素
>>> def most_frequent(lst):
        return max(set(lst), key = lst.count)
>>> lst = [1, 2, 2, 1, 4, 2, 4, 4, 4]
>>> most_frequent(lst)    # 4
```

5 字典

5.1-5.2 字典的定义、创建和访问

字典是一种映射类型，**由若干“键(key)：值(value)”对组成，“键”和“值”之间用冒号隔开，所有“键值对”放在一对大括号“{}”内，并用逗号分隔**。其中**“键”必须为不可变类型，在同一个字典中，“键”必须是唯一的，但“值”可以重复**。

字典的创建主要有两种方式，第一种是直接通过一对大括号包裹键值对创建字典对象，第二种是使用dict()函数创建字典对象。

当字典不再使用时，可通过del命令删除字典，删除后字典便不可再调用。

5.1-5.2 字典的定义、创建和访问

字典是一种动态结构，**可随时在其中添加键值对**。要添加键值对，可依次指定字典名、键及相关联的值。

```
alien0 = {‘color’: ‘green’, ‘points’: 5}

print(alien0)

alien0[‘x_position’] = 0

alien0[‘y_position’] = 25

print(alien0)
```

```
In [25]: d1 = {1: 'one', 2: 'two'}

In [26]: 1 in d1
Out[26]: True
```

注意：字典键不允许修改，因此用元组存储字典的键会更适合

5.1-5.2 字典的定义、创建和访问

```
alien0['points'] = 5

print(alien0)

alien0['color'] = 'yellow'

alien0['x_position'] = 0

del alien0['points']

print(alien0)
```

注意:字典值的内容可以是数字、字符串、列表乃至字典,事实上, 可将任何Python对象作为字典的值

5.1-5.2 字典的定义、创建和访问

函数dict()可以从其他映射或键值对序列中创建字典，也可以使用关键字实参来创建字典。

```
>>>items = [('name', 'Gumby'), ('age', 42)]  #元组也可
>>>d = dict(items)  # {'age': 42, 'name': 'Gumby'}
>>>c = dict(name = 'Gumby', age = 42)
>>>mapping = dict(zip(range(5), reversed(range(5))))
>>>mapping   # enumerate()
{0: 4, 1: 3, 2: 2, 3: 1, 4: 0}
```

5.3 字典的常用方法及应用(遍历)

遍历字典的键值对时，需声明两个变量，以用于存储字典键值对中的键和值。

返回所有键值对的方法: **items()**。

```
>>> s = {'a':1, 'b':2, 'c':3}

>>> b, c, d = s.items()

>>> b

('a', 1)
```

注意: 在Python3.7以后, items(), keys(), values()方法的返回值和添加字典键值对的顺序都相同。

5.3 字典的常用方法及应用(遍历)

```
favorite_languages = {

    ……

    }

for name, language in favorite_languages.items():

    print(name.title() + "'s favorite language is " +

          language.title() +

          ".")
```

5.3 字典的常用方法及应用(遍历)

```
favorite_languages = {
    ……
    }
for name in favorite_languages.keys():
    print(name.title())
```

keys()方法用于返回字典所有的键, 如果只返回字典的键, 也可以将.keys()方法省略。

5.3 字典的常用方法及应用(遍历)

```
favorite_languages = {
    ……
    }
for language in favorite_languages.values():
    print(language.title())
```

values()方法用于返回字典的所有值。

另外，我们还可使用**set函数**对keys()，values()、items()的返回内容剔除重复项, 使用list()方法将其内容转换成列表。

5.3 字典的常用方法及应用

fromkeys()方法：创建一个新字典，其中包含指定的键，而每个键对应的值为None或指定值。

```
>>> alien = {}.fromkeys([‘name’, ‘age’]) # 去除方括号试试
>>> alien
{'name': None, 'age': None}
>>> alien.fromkeys(['name', 'age'], 'unknown')
{'name': 'unknown', 'age': 'unknown'}
```

5.3 字典的常用方法及应用

get()方法：返回相应键的值，不存在，则返回默认值(当访问字典不存在的键时, 将会引发异常, 而使用get方法则不会)。

```
>>> alien = {}   # 推荐使用get()方法来获得字典的元素值
>>> print(alien['name'])
Traceback (most recent call last):
....    # 如果指定的键可能不存在, 应考虑使用get()方法
>>> print(alien.get('name'))
>>> print(alien.get('name', 'N/A'))
```

5.3 字典的常用方法及应用

update()方法：使用一个字典中的键值对来更新另一个字典同名键值对；如果不存在键相同的项，则变为追加模式(cmd中)。

```
>>> web = {'title':'python','url':'www.python.org'}
>>> up = {'title':'py'}
>>> web.update(up)
>>> down = {'do':'it'}
>>> web.update(down)
```

5.3 字典的常用方法及应用

setdefault()方法：以追加方式为字典的某个键设置默认值，如果该键已存在，则不做任何改动。

```
>>> spam = {'name': 'Pooka', 'age': 5}
>>> spam.setdefault('color', 'black')
'black'
>>> spam
{'name': 'Pooka', 'age': 5, 'color': 'black'}
>>> spam.setdefault('color', 'white')
'black'
```

5.3 字典的常用方法及应用

copy()方法: 用于返回一个新字典，其包含的键值对与原字典相同(浅复制);

clear()方法: 用于清空字典内容;

pop(key)方法: 用于删除相应键值对，并返回该键的值;

popitem()方法: 以元组的形式，随机返回字典中的一个键值对;

len()方法: 返回字典包含的键值对数量。

5.3 字典的常用方法及应用

注意：字典的键值对不是严格排序的(这里指3.6以前的字典，3.6以后的字典，会按键值对赋值顺序生成内容)，**只要两个字典的键值对内容相同， Python就认为它们相等或等价(集合也是如此)。**

5.4 字典推导式

字典推导式的写法和集合推导式的写法类似，也是放在一对大括号中。表达式中包含键和值两部分，并分别指定这两部分的值。

语法形式：{键表达式:值表达式 for变量 in 已有序列 if 过滤条件}

```
>>> squares = {i: i**2 for i in range(5)}
>>> print(squares)
{0: 0, 1: 1, 2: 4, 3: 9, 4: 16}
```

5.4 字典推导式

```
>>> squares = {i: f"{i} squared is {i**2}" for i in range(5)}

>>> print(squares)

>>>b = ['ab', 'abc', 'abcde', 'abcd']

>>>loc = {value: index for index, value in enumerate(b)}

>>>loc

{'ab': 0, 'abc': 1, 'abcd': 3, 'abcde': 2}
```

5.5 字典综合小例子

```
>>> from random import choices  # 统计字符频数(小练习)
>>> from string import ascii_letters, digits
# 进行1000次随机选择
>>> chars = ''.join(choices(ascii_letters+digits, k=1000))
```

5.5 字典综合小例子

```
>>> d = dict()   # ascii_letters, digits各为字母、数字字符串
>>> for ch in z:  # {}.fromkeys()方法也可以
...     d[ch] = d.get(ch, 0) + 1
>>> print(d)  # 练习, 大小写字母合并统计
```

5.5 字典综合小例子

```
# sorted()方法在字典中的深入应用, 字典不直接支持sort()
>>>persons = [{'name': 'Dong', 'age': 37}, {'name':'Li', 'age': 41}, {'name':'Dong', 'age':32}, {'name': 'Li', 'age': 53}]
# 对’name’值升序, ’age’降序排列, 负号只针对整数有效
>>>sorted(persons, key=lambda x:(x['name'], -x['age’]))
>>>from operator import itemgetter
>>>phonebook = {'Linda': '7750', 'Bob': '9345', 'Carol': '5834’}
# 依次对字典的键、值升序排列
```

5.5 字典综合小例子

```
>>>sorted(phonebook.items(), key=itemgetter(0, 1))

>>>gameresults = [['Bob', 95.2, 'A'], ['Alan', 86.0, 'C'], ['Mandy', 83.5, 'A'], ['Bob', 89.3, 'E']]

>>>sorted(gameresults, key=itemgetter(0, 1))  # persons尝试

>>>gameresults=[{'name':'Bob','wins':10,'losses':3,'rating':75.0},{'name':'David','wins':3,'losses':5,'rating':57.0},{'name':'Carol','wins':4,'losses':5,'rating':67.0},{'name':'Patty','wins':9,'losses':3,'rating':72.8},]

>>>sorted(gameresults, key=itemgetter('losses', 'name'))
```

5.6 再谈赋值语句

在Python中，赋值语句极为灵活。**可同时给多个变量进行赋值(实际上是元组及列表自动拆包)，以提高程序执行效率**。

```
>>>x, y, z = 1, ‘Hello’, 3        # 等价于x, y, z = (1, ‘Hello’, 3)
# z为列表，等价于x, y, *z = [1, 2, 3, 4] 或(1, 2, 3, 4)
>>>x, y, *z = 1, 2, 3, 4    # z → [3, 4]
>>>values = 1, ‘He’, 3          # 等价于values = (1, ‘He’, 3)
>>>a, b, (c, d) = 1, 2, (3, 4)
>>>c     # 3
```

5.6 再谈赋值语句

```
>>>c, d = d, c    # 交换值

>>>seq = [(1, 2, 3), (4, 5, 6), (7, 8, 9)]

>>>for a, b, c in seq:    # 序列自动拆包的另一个典型用法

        print('a = {}, b = {}, c = {}'.format(a, b, c))
```

5.6 再谈赋值语句

为了方便用户个性化使用, 还可以**直接将函数或方法直接赋给变量(函数和方法名后面的括号不需要添加)**。

```
>>> a = print
>>> a('hello')
hello
>>> import math
>>> b = math.sqrt          #>>>a = 2 + 3; b = 5 + 6(一起执行)
>>> b(9)
```

Answers

>>>animals = [‘cat’, ‘dog’, ‘fish’, ‘monkey’]

>>>animals.insert(9, ‘bird’)

>>>’Welcome to python world’.center(5, ‘*’)

>>>’Welcome to python world’

>>>print(“Hello,”, “world!”) 与 print(“Hello,” + “world!”)

>>>['this', 'is','a', 'python', 'lesson'].count('is')

>>>"this is a python lesson".count('is')

在idle中默认没有清屏功能, 需要另外设置

Answers

通常来讲, **C、C++、Java要比Python快好几个数量级;**

Python的目标是易于使用和帮助提高开发速度, 这种灵活性是以牺牲效率为代价的;

就目前的机器性能而言, Python语言开发的程序运行速度, 并不会让用户感觉不适;

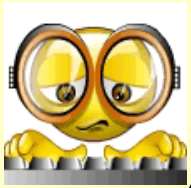
如果确实需要进一步提升程序运行速度, 建议采用以下方案:

(1)使用Python开发原型;

(2)对程序性能进行分析, 以找出瓶颈;

(3)使用C、C++、Java或其他高性能语言重写瓶颈部分内容。

练习



01 完成教材和PPT上的练习和例子

02 完成课后编程题1、2、3、5

谢 谢