第1章 Python 基 础

Python是一种跨平台、开源、免费的解释型高级动态编程语言，是一种通用编程语言。Python支持命令式编程和函数式编程两种方式，其语法简洁清晰，功能强大，易学易用。Python具有良好的编程生态，拥有大量的几乎支持所有领域应用开发的成熟扩展库和狂热支持者。

1.1 Python 语言基本语法

1.1.1 Python 的编程方式

1. 交互式编程

交互式编程不需要创建脚本文件，是通过 Python 解释器的交互模式来编写代码。在 Python 提示符中输入表达式 3+2，然后按 Enter 键查看运行结果，如图 1-1 所示。

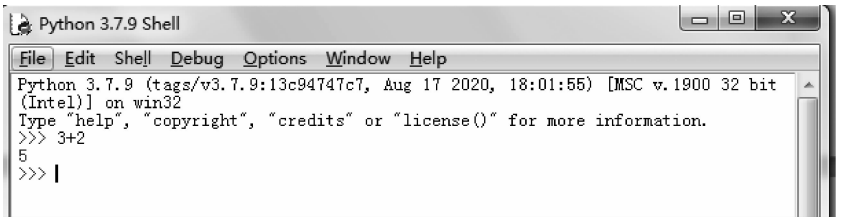

图 1-1 运行结果

2. 脚本式编程

通过脚本参数调用解释器开始执行脚本，直到脚本执行完毕。当脚本执行完成后，解释器不再有效。下面写一个简单的 Python 脚本程序，所有 Python 文件将以 .py 为扩展名。如图 1-2 所示。

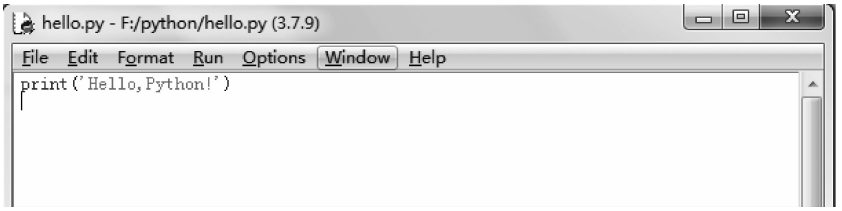

图 1-2 Python 脚本程序

将文件保存为 hello. py，在控制台运行结果如图 1-3 所示。

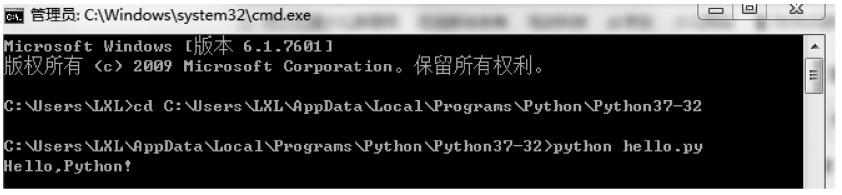

图 1-3 在控制台运行结果

1.1.2 Python 语句的缩进

Pyhton 语言与 Java、C#等编程语言最大的不同点是，Python 代码块使用缩进对齐表示代码逻辑，而不是使用大括号。这对习惯用大括号表示代码块的程序员来说，确实是学习 Python 的一个障碍。

Python 每段代码块缩进的空白数量可以任意，但要确保同段代码块语句必须包含相同的缩进空白数量。一般用 tab 键进行缩进，或者按 4 个空格进行缩进，切记不能混用。

例如，输入图 1-4 所示代码，运行时显示 unexpected indent 错误。

图 1-4　输入代码

正确的代码缩进应该是：

```
a=input(请输入第一个数")
b=input("请输入第二个数")
if a > b:
    print("a>b")
else:
    print("a<b")
```

1.1.3 Python 引号

在 Python 语言中，引号主要用于表示字符串。可以使用单引号（'）、双引号（"）、三引号（'''），引号必须成对使用。单引号和双引号用于程序中的字符串表示；三引号允许一个字符串可以跨多行，字符串中可以包含换行符、制表符及其他特殊字符，三引号也用于程序中的注释。如：

```
bookname = 'Python 数据分析与可视化'
bookbrief = "这是一本学习 Python 编程的书"
```

paragraph = """图书主要阐述 Python 的基础知识,Python 数据分析,Python 数据可视化"""

1.1.4　Python 标识符

标识符用于 Python 语言的变量、关键字、函数、对象等数据的命名。标识符的命名需要遵循下面的规则。

（1）可以由字母（大写 A~Z 或小写 a~z）、数字（0~9）、下划线（_）和汉字组合而成，但不能由数字开头。

（2）不能包含除_以外的任何特殊字符，如:%、#、&、逗号、空格等。

（3）不能包含空白字符（换行符、空格和制表符称为空白字符）。

（4）标识符不能是 Python 语言的关键字和保留字。

（5）标识符区分大小写，num1 和 Num1 是两个不同的标识符。

（6）标识符的命名要有意义，做到见名知意。

例：正确标识符的命名示例

width、height、book、result、num、num1、num2、book_price。

例：错误标识符的命名示例

123rate(以数字开头)、Book Author(包含空格)、Address#(包含特殊字符)、class(calss 是类关键字)。

1.1.5　Python 关键字

Python 预先定义了一部分有特别意义的标识符，用于语言自身使用。这部分标识符称为关键字或保留字，不能用于其他用途，否则会引起语法错误，随着 Python 语言的发展，其预留的关键字也会有所变化。表 1-1 列出了 Python 预留的关键字。

表 1-1　Python 预留的关键字表

and	exec	not	assert	finally
or	break	for	pass	class
from	print	continue	global	raise
def	if	return	del	import
try	elif	in	while	else
is	with	except	lambda	yield

1.1.6　Python 变量

用标识符命名的存储单元的地址称为变量，变量是用来存储数据的，通过标识符可以

获取变量的值，也可以对变量进行赋值。对变量赋值的意思是将值赋给变量，当赋值完成后，变量所指向的存储单元存储了被赋的值，在 Pyhton 语言中赋值操作符为“=、+=、-=、*=、/=、%=、**=、//=”。

当程序使用变量存储数据时，必须要先声明变量，然后才能使用。声明变量的语法如下：

```
identifier [ = value] ;
```

其中 identifier 是标识符，也是变量名称。value 为变量的值，该项为可选项，可以在变量声明时给变量赋值，也可以不赋值。当声明变量时，不需要声明数据类型，Python 会自动选择数据类型进行匹配。

例：变量声明示例

① result;

② width。

例：变量声明并赋值示例

① result = 30;

② name="Peter"。

小结

Python 的语法有两点需要注意：①Python 同一代码块的缩进要对齐，不然就会出现语法错误；②Python 在字符串表示上和其他语言有所不同，Python 的字符串可以用单引号（'）、双引号（"）、三引号（'''）表示。

1.2 内置数据类型

1.2.1 Python 数据类型

计算机程序可以处理各种数值，不同的数据需要定义不同的数据类型。

1. 数值类型

Python 数值类型用于存储数值，Python 3 支持以下 3 种不同的数值类型。

* 整型（int）：通常被称为整数，是正整数、0 或负整数，不带小数点。

* 浮点型（float）：由整数部分与小数部分组成，浮点型也可以用科学计数法表示（1.25e2 就是 $1.25 * 10^2 = 125$）。

* 复数（complex）：由实数部分和虚数部分构成，可以用 a+bj 形式来表示，如 2+3j。

数据类型是不允许改变的，这就意味着如果改变数值数据类型的值，将重新分配内存空间。可以用 type( ) 函数判断变量的数据类型。

2. 字符串

字符串是 Python 中最常用的数据类型之一，使用单引号、双引号、三引号来括起来表示，三引号表示多行字符串，平常使用单引号或双引号就行。当有单引号、双引号嵌套时，可使用反斜杠【\】进行转义，或者使用不是嵌套中的引号，如 var = 'This is "dog"!'。

字符串是不可变的序列数据类型，不能修改字符串本身，和数字类型一样。Python 完全支持 Unicode 编码，所有的字符串都是 Unicode 字符串。虽然字符串本身不可变，但是可以像列表一样进行切片和截取子串操作，但不会引起字符串本身的变化。

1）创建和访问字符串

创建字符串很简单，只要为变量分配一个字符串类型的值即可。例：

```
a='Hello'
b="Python"
```

2）字符串运算类型（见表 1-2）

表 1-2　字符串运算类型

操作	描述	例子	结果
+	字符串拼接（速度慢，少用）	a + b	"Hello Python"

续表

操作	描述	例子	结果
*	重复字符串，相当于乘法	a * 2	"Hello Hello"
[]	通过索引获取字符串中的字符	a [2]	"l"
[:]	截取、切片	a [2:]	"llo"
in	成员运算符，如果字符串中包括给定的字符串，返回 True	"X" in a	False
not in	成员运算符，如果字符串中包括给定的字符串，不包含返回 True	"X" not in a	True
r/R	原始字符串，不会转义特殊字符	print (r"\n")	\n

3）转义字符

转义字符都有特殊含义，见表 1-3。

表 1-3　转义字符及含义

转义字符	描述	转义字符	描述
\ （在行尾时）	续行符	\000	空
\\	反斜杠	\n	换行
\'	单引号	\v	纵向制表符
\"	双引号	\t	横向制表符
\a	响铃	\r	回车
\b	退格	\f	换页
\e	转义	\o33	颜色控制

4）字符串内置方法

表 1-4 从常用字符串内置方法开始梳理，常用的都要记住如何使用，不常用的要有印象。

表 1-4　常用字符串内置方法

方法	描述
string. split (sep, maxsplit=-1)	字符串分割，返回对象为分割后的子串列表。默认使用空格分割，可指定分隔符，分隔符不包含在分割后的子串中；默认贪婪分割，可指定分割次数
string. splitlines ([keepends])	按照 ('\r', '\r\n', '\n') 分割字符串，返回一个包含各行作为元素的列表，如果参数 keepends 为 False，不包含换行符，否则保留换行符
string. strip ([chars])	默认返回去除字符串两边空格的字符串，其中有 string. lstrip( ) 和 string. rstrip( ) 是分别去除左、右各边的空格。可指定去除的字符
string. find (sub [, star [, end]])	返回查找子串出现的第一个位置索引，可以指定查找范围，没有找到子串返回 -1。其中 string. rfind( ) 是从右边开始查找

续表

方法	描述
string. index（sub［, start［, end］］）	返回查找子串出现的一个位置索引，可指定查找范围，没有找到会异常 ValueError，其中 string. rindex( ) 是从右边开始检索
string. count（sub［, start［, end］］）	返回查找子串在字符串中出现的次数，可指定查找范围，没有找到，返回 0
string. lower( )	返回一个全部为小写的字符串
string. upper( )	返回一个全部为大写的字符串
string. startswith（prefix［, start［, end］］）	返回在给定范围中是否以指定字符串开头，是返回 True，否则返回 False
string. endswith（prefix［, start［, end］］）	返回在给定范围中是否以指定字符串结尾，是返回 True，否则返回 False
string. replace（old, new［, count］）	返回使用字符串 new 替换字符串 old count 次的新的字符串
string. encode（encoding='utf-8', errors='strict'）	返回以 encoding 指定的编码格式编码的 bytes 对象，如果出错会报一个 UnicodeEncodeError 异常，除非指定 errors 是'ignore'或'replace'
bytes. decode（encoding='utf-8', errors='strict'）	和 string. encode 是逆向过程，将 bytes 以指定编码格式解码为 string，如果出错会报一个 UnicodeEncodeError 异常，除非指定 errors 是'ignore'或'replace'
string. format( )	格式化字符串，常用方式为位置参数和关键字参数
string. join（seq）	以指定的 string 为分隔符，将序列 seq 中的元素（其中的元素必须是以字符串类型的形式才可以）合并为一个新的字符串
string. center（width）	返回一个原字符串居中，并默认使用空格填充至长度 width 的新字符串，可指定填充字符串
string. ljust（width）	返回一个原字符串左对齐，并默认使用空格在右侧填充至长度 width 的新字符串，可指定填充字符串
string. rjust（width）	返回一个原字符串右对齐，并默认使用空格在左侧填充至长度 width 的新字符串，可指定填充字符串
string. zfill（width）	返回长度为 width 的字符串，原字符串右对齐，前面填充 0
string. expandtabs（tabsize=8）	把字符串 string 中的 tab 符号转化为空格，tab 默认空格数是 8
string. capitalize( )	返回首字母大写的字符串
string. isalnum( )	如果 string 中至少出现一个字符并且所有的字符都是字母或数字的则返回 True，否则返回 False
string. isalpha( )	如果 string 中至少出现一个字符且所有字符都是字母的则返回 True，否则返回 False

续表

方法	描述
string. isdigit( )	string 中除汉字、数字返回 False，其余数字［unicode 数字，bytes 数字（单字节），全角数字（双字节），罗马数字］都返回 True，无 Error
string. isdecimal( )	因为 bytes 类型数字没有 isdecimal 属性，会报异常 AttributeError，unicode 数字、全角数字返回 True，罗马数字、汉字、数字会返回 False
string. isnumeric( )	因为 bytes 类型数字没有 isnumeric 属性，会报异常 AttributeError，其余类型数字都返回 True
string. islower( )	只要 string 中不包含大写字符并且包含至少一个小写字符就会返回 True，否则返回 False
string. isupper( )	只要 string 中不包含小写字符并且包含至少一个大写字符就会返回 True，否则返回 False
string. istitle( )	如果 string 是标题式的字符串，就返回 True，否则返回 False，标题式是：字符串中的单词首字母大写，如"Title 10Dd"
string. isspace( )	如果 string 中只包含空格类型，就返回 True，否则返回 False，空格类型：'\ n \ t \ v \ r \ f'
string. title（width）	返回‘标题化’的 string，即单词的首字母全部大写，其余字母小写
string. swapcase( )	翻转大小写
str. maketrans（intab，outtab）	接受两个长度相同的字符串，第一个字符串是需要转换的字符串，第二个字符串是转化的目标字符串，用于创建字符串映射的转化表
string. translagte	使用 str. maketrans( ) 方法转化的转化表进行字符串的转化
string. partion（str）	从左边找 str 出现的第一个位置起，把字符串 string 分成一个 3 元素的元祖（string_pre_str，str，string_post_str），如果 string 中不包含 str，则 string_pre_str = = string
string. rpartion( )	从右边找 str 出现的第一个位置起，把字符串 string 分成一个 3 元素的元祖（string_pre_str，str，string_post_str），如果 string 中不包含 str，则 string_pre_str = = string

3. 布尔类型

Python 支持布尔类型的数据，布尔类型只有 True 和 False 两种值，布尔类型支持 and、or 和 not 运算，布尔运算在计算机中用来做条件判断，根据计算结果为 True 或 False，计算机可以自动执行后续代码。

and 运算，只有当两个布尔值都为 True 时，计算结果才是 True；

or 运算，只要有一个布尔值为 True，计算结果就是 True；

not 运算，把 True 变为 False，或者把 False 变为 True。

在 Python 中，布尔类型还可以与其他数据类型做 and、or 和 not 运算，以下几种类型

会被认为是 False：为 0 的数字；空字符串；表示空值的 None；空集合、空元祖、空序列和空字点；其他的值都为 True。如：

```
a=""
print(a and True)     #结果为 False
```

4. Python 数字类型转换

1）int 函数

把符合数学格式的数字型字符串转换成整数；把浮点数转换成整数，但只是简单地取整，而非四舍五入。

```
aa=int("124")
print("aa=",aa)       #输出 aa= 124
bb=int(123.45)
print("bb=",bb)       #输出 bb= 123
```

2）float 函数将整数和字符串转换成浮点数

```
aa=float("124")
print("aa=",aa)       #输出 aa= 124.0

bb=float(123.45)
print("bb=",bb)       #输出 bb= 123.45
```

5. str 函数将数字转换成字符

```
aa=str(124)
print("aa=",aa)       #输出 aa= 124

bb=str(123.45)
print("bb=",bb)       #输出 bb= 123.45
print(type(bb))       #输出<class 'str'>
```

1.2.2　运算符和表达式

1. 算术运算符

+、-、*、/、//（整除）、%（取余）、**（幂）

注意：

① 除法运算中除数不为 0，'/'运算结果为浮点数，'//'运算结果为整数；

② 在算术操作符中使用%求余，结果只与除数（第二个操作数）的正负有关。

2. 赋值运算符

赋值运算符与四则运算符组合，实现把赋值运算符左右两侧的内容进行相应的四则运算后，把结果赋值给变量，见表 1-5。

表 1-5　常用字符串内置方法

运算符	描述	实例
=	简单赋值运算符	c=a+b 将 a+b 的运算结果赋值给 c
+=	加法赋值运算符	c+=a 等价于 c=c+a
-=	减法赋值运算符	c-=a 等价于 c=c-a
*=	乘法赋值运算符	c*=a 等价于 c=c*a
/=	除法赋值运算符	c/=a 等价于 c=c/a
%=	取模赋值运算符	c%=a 等价于 c=c%a
**=	幂赋值运算符	c**=a 等价于 c=c**a
//=	整除赋值运算符	c//=a 等价于 c=c//a

3. 比较运算符

比较运算符用于对变量或表达式的结果进行大小、真假等比较。比较的结果为真，则返回 True，结果为假，则返回 False。通常用在条件语句中作为判断的依据，见表 1-6。

表 1-6　比较运算符

运算符	描述
==	等于，比较两个对象是否相等
!=	不等于，比较两个对象是否不相等
<>	不等于，比较两个对象是否不相等。Python 3 已废弃
>	大于，判断 x 是否大于 y
<	小于，判断 x 是否小于 y
>=	大于等于，判断 x 是否大于等于 y
<=	小于等于，判断 x 是否小于等于 y

例如：判断 python 和 english 的成绩是否合格。

```
python = float(input("please enter your python score:"))
english = float(input("please enter your english score:"))
if python >= 60:
    print("Your python score is qualified.")
if python < 60:
    print("Your python score is disqualified.")
if english >= 60:
```

```
    print("Your english score is qualified.")
if english < 60:
    print("Your english score is disqualified.")
```

执行结果：

```
please enter your python score:56
please enter your english score:72
Your python score is disqualified.
Your english score is qualified.
```

4. 位运算符

位运算符是把数据看成二进制数进行计算的。

（1）位与运算（&）：两个操作数按二进制数表示，对应位都为 1，结果位才为 1。

（2）位或运算（|）：两个操作数按二进制数表示，对应位都为 0，结果位才为 0。

（3）位异或运算（^）：两个操作数按二进制数表示，对应位同为 1 或同为 0，结果为 0，否则为 1。

（4）位取反运算（~）：把二进制操作数对应位 1 变为 0，0 变为 1。

（5）左移位运算符（<<）：把二进制操作数向左移动相应位数，当左边最高位溢出时被丢弃，右边空位用 0 补齐（左移位相当于乘以 2 的 n 次幂）。

（6）右移位运算符（>>）：把二进制操作数向右移动相应位数，当右边溢出位溢出时被丢弃，左边最高位如果是 0 补 0，是 1 补 1（右移位相当于除以 2 的 n 次幂）。

5. 成员运算符

成员运算符有 in、not in。

6. 身份运算符

身份运算符有 is、is not。

7. 运算符的优先级

按运算符的优先级从高到低运算，同一级别从左到右顺序执行，可以使用( ) 改变优先级。具体见表 1-7。

表 1-7　运算符的优先级

类型	运算符	说明
单目运算符	~，+，-	取反、正号和负号
算术运算符	*，/,%，//	乘、除、求余、整除
	+，-	加、减
位运算符	<<，>>	左移、右移

续表

类型	运算符	说明
位运算符	&	位与
	^	位异或
	|	位或
比较运算符	<，<=，>，>=，==，!=	小于、小于等于、大于、大于等于、等于、不等于
逻辑运算符	and，or，not	与、或、非
赋值运算符	=，+=，-=，*=，/=,%=	赋值、加法赋值、减法赋值、乘法赋值、除法赋值、取模赋值

1.2.3 Python 序列类型

1. 序列的运算

所谓序列，指的是一块可存放多个值的连续内存空间，这些值按一定顺序排列，可通过每个值所在位置的编号（称为索引）访问它们。在 Python 中，序列类型包括字符串、列表、元组、集合和字典，这些序列支持以下几种通用的操作，但比较特殊的是，集合和字典不支持索引、切片、相加和相乘操作。

1）序列索引

在序列中，每个元素都有属于自己的编号（索引）。从起始元素开始，索引值从 0 开始递增，如图 1-5 所示。

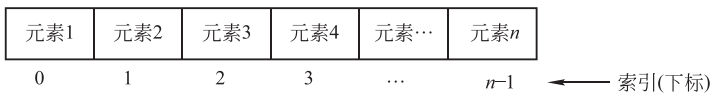

图 1-5　元素正向索引

除此之外，Python 还支持索引值是负数，此类索引是从右向左计数，换句话说，从最后一个元素开始计数，从索引值 -1 开始，如图 1-6 所示。

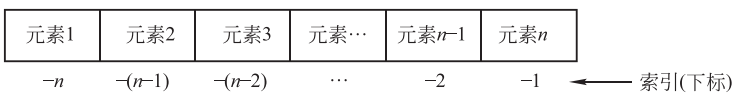

图 1-6　元素负向索引

无论是采用正索引值，还是负索引值，都可以访问序列中的任何元素。以字符串为例，访问“数据分析与可视化”的首元素和尾元素，可以使用以下的代码：

```
str="数据分析与可视化"
print(str[0],"==",str[-8])
```

```
print(str[7],"==",str[-1])
```

输出结果为：

```
数 == 数
化 == 化
```

2）序列切片

切片操作是访问序列中元素的另一种方法，它可以访问一定范围内的元素，通过切片操作，可以生成一个新的序列。

序列实现切片操作的语法格式如下：

```
sname[start: end: step]
```

其中，各个参数的含义如下。

sname：表示序列的名称。

start：表示切片的开始索引位置（包括该位置），此参数也可以不指定，会默认为 0，也就是从序列的开头进行切片。

end：表示切片的结束索引位置（不包括该位置），如果不指定，则默认为序列的长度。

step：表示在切片过程中，隔几个存储位置（包含当前位置）取一次元素，也就是说，如果 step 的值大于 1，则在进行切片去序列元素时，会“跳跃式”地取元素。如果省略设置 step 的值，则最后一个冒号就可以省略。

例如，对字符串“数据分析与可视化”进行切片：

```
str="数据分析与可视化"
#取索引区间为[0,2]之间(不包括索引 2 处的字符)的字符串
print(str[:2])
#隔 1 个字符取一个字符,区间是整个字符串
print(str[::2])
#取整个字符串,此时 [] 中只需一个冒号即可
print(str[:])
```

运行结果为：

```
数据
数分与视
数据分析与可视化
```

3）序列相加

在 Python 中，支持两种类型相同的序列使用“+”运算符做相加操作，它会将两个序

列进行连接，但不会去除重复的元素。

这里所说的“类型相同”，指的是“+”运算符的两侧序列要么都是列表类型，要么都是元组类型，要么都是字符串。如下所示：

```
str="可视化"
print("数据分析"+"与"+str)
```

输出结果为：

```
数据分析与可视化
```

4）序列相乘

在 Python 中，使用数字 n 乘以一个序列会生成新的序列，其内容为原来序列被重复 n 次的结果。例如：

```
str="数据分析与可视化"
print(str*3)
```

输出结果为：

```
数据分析与可视化数据分析与可视化数据分析与可视化
```

5）检查元素是否包含在序列中

在 Python 中，可以使用 in 关键字检查某元素是否为序列的成员，其语法格式为：

```
value in sequence
```

其中，value 表示要检查的元素，sequence 表示指定的序列。

例如，检查字符‘数’是否包含在字符串“数据分析与可视化”中，可以执行以下代码：

```
str="数据分析与可视化"
print('数'in str)
```

运行结果为：

```
True
```

和 in 关键字用法相同，但功能恰好相反的，还有 not in 关键字，它用来检查某个元素是否不包含在指定的序列中，如：

```
str="数据分析与可视化"
print('数'not in str)
```

运行结果为：

```
False
```

6）与序列相关的内置函数

Python 提供了几个与序列相关的内置函数（见表 1-8），可用于实现与序列相关的一些常用操作。

表 1-8　与序列相关的内置函数

函数	功能
len( )	计算序列的长度，即返回序列中包含多少个元素
max( )	找出序列中的最大元素。注意，当对序列使用 sum( ) 函数时，做加和操作的必须都是数字，不能是字符或字符串，否则该函数将抛出异常，因为解释器无法判定是要做连接操作（+ 运算符可以连接两个序列），还是做加和操作
min( )	找出序列中的最小元素
list( )	将序列转换为列表
str( )	将序列转换为字符串
sum( )	计算元素和
sorted( )	对元素进行排序
reversed( )	反向序列中的元素
enumerate( )	将序列组合为一个索引序列，多用在 for 循环中

2. 在 Python 中的序列类型

在 Python 中的序列类型包括列表（list）、元组（tuple）、字典（dict）和集合（set）。列表（list）和元组（tuple）比较相似，它们都按顺序保存元素，所有的元素占用一块连续的内存，每个元素都有自己的索引，因此列表和元组的元素都可以通过索引（index）来访问。它们的区别在于：列表是可以修改的，而元组是不可以修改的。字典（dict）和集合（set）存储的数据都是无序的，每份元素占用不同的内存，其中字典元素以 key-value 的形式保存。

1）列表

从形式上看，列表会将所有元素都放在一对中括号［ ］里面，相邻元素之间用逗号（,）分隔，如下所示：

```
[element1, element2, element3, …, elementn]
```

在此格式中，element1 ~ elementn 表示列表中的元素，个数没有限制，只要是 Python 支持的数据类型就可以。

从内容上看，列表可以存储整数、小数、字符串、列表、元组等任何类型的数据，并且同一个列表中元素的类型也可以不同。如：

```
["数据分析与可视化", 1, [2,3,4] , 3.0]
```

可以看到，列表中同时包含字符串、整数、列表、浮点数这些数据类型。

注意：在使用列表时，虽然可以将不同类型的数据放入同一个列表中，但通常情况下不这么做，同一列表中只放入同一类型的数据，这样可以提高程序的可读性。

（1）Python 创建列表。在 Python 中，创建列表的方法可分为两种，下面分别进行介绍。

① 使用［ ］直接创建列表。使用［ ］创建列表后，一般使用=将它赋值给某个变量，具体格式如下：

```
listname = [element1 , element2 , element3 , …, elementn]
```

其中，listname 表示变量名，element1 ~ elementn 表示列表元素。

例如，下面定义的列表都是合法的：

```
num = [1, 2, 3, 4, 5, 6, 7]
program = ["C 语言", "Python", "Java"]
```

另外，当使用此方式创建列表时，列表中元素可以有多个，也可以一个都没有，如：

```
emptylist = [ ]
```

这表明，emptylist 是一个空列表。

② 使用 list( ) 函数创建列表。除了使用［ ］创建列表外，Python 还提供了一个内置的函数 list( )，使用它可以将其他数据类型转换为列表类型。如：

```
#将字符串转换成列表
list1 = list("hello")
print(list1)

#将元组转换成列表
tuple1 = ('Python', 'Java', 'C++', 'JavaScript')
list2 = list(tuple1)
print(list2)

#将字典转换成列表
dict1 = {'a':100, 'b':42, 'c':9}
list3 = list(dict1)
print(list3)

#将区间转换成列表
range1 = range(1, 6)
```

```
list4 = list(range1)
print(list4)

#创建空列表
print(list())
```

运行结果为：

```
['h', 'e', 'l', 'l', 'o']
['Python', 'Java', 'C++', 'JavaScript']
['a', 'b', 'c']
[1, 2, 3, 4, 5]
[]
```

（2）访问列表元素。列表是 Python 序列的一种，可以使用索引（index）访问列表中的某个元素（得到的是一个元素的值），也可以使用切片访问列表中的一组元素（得到的是一个新的子列表）。

使用索引访问列表元素的格式为：

```
listname[i]
```

其中，listname 表示列表名字，i 表示索引值。列表的索引可以是正数，也可以是负数。

使用切片访问列表元素的格式为：

```
listname[start: end: step]
```

其中，listname 表示列表名字，start 表示起始索引，end 表示结束索引，step 表示步长。

以上两种方式已在前面讲解，这里就不再赘述了，仅作示例演示，请看下面代码：

```
li = list("数据分析与可视化")

#使用索引访问列表中的某个元素
print(li[3])  #使用正数索引
print(li[-4])  #使用负数索引

#使用切片访问列表中的一组元素
print(li[2: 4])  #使用正数切片
print(li[: 8: 2])  #指定步长
```

```
print(li[-6: -1])  #使用负数切片
```

运行结果为：

```
析
与
['分', '析']
['数', '分', '与', '视']
['分', '析', '与', '可', '视']
```

例：生成包含 20 个随机数的列表，然后将前 10 个元素升序排列，后 10 个元素降序排列，并输出结果。

```
import random
x = [random.randint(0,100) for i in range(20)]
print("原列表:",x)
y = x[0:10]
y.sort()
x[0:10] = y
y = x[10:20]
y.sort(reverse=True)
x[10:20] = y
print("排序后:",x)
```

运行结果为：

```
原列表: [15, 73, 34, 53, 66, 40, 46, 76, 98, 93, 69, 37, 32, 19, 79, 21, 4, 14, 0, 0]
排序后: [15, 34, 40, 46, 53, 66, 73, 76, 93, 98, 79, 69, 37, 32, 21, 19, 14, 4, 0, 0]
```

（3）Python 删除列表。对于已经创建的列表，如果不再使用，可以使用 del 关键字将其删除。

实际开发中并不经常使用 del 来删除列表，因为 Python 自带的垃圾回收机制会自动销毁无用的列表，即使开发者不手动删除，Python 也会自动将其回收。

del 关键字的语法格式为：

```
del listname
```

其中，listname 表示要删除列表的名称。

Python 删除列表实例演示：

```
intlist = [1, 45, 8, 34]
print(intlist)
del intlist
print(intlist)
```

运行结果为：

```
[1, 45, 8, 34]
Traceback (most recent call last):
File "F:/python/1.py", line 4, in <module>
print(intlist)
NameError: name 'intlist'is not defined
```

（4）向列表添加元素。在实际开发中，经常需要对 Python 列表进行更新，包括向列表中添加元素、修改表中元素及删除元素。下面学习如何向列表中添加元素。

① append( ) 方法添加元素。append( ) 方法用于在列表的末尾追加元素，该方法的语法格式如下：

```
listname.append(obj)
```

其中，listname 表示要添加元素的列表；obj 表示要添加到列表末尾的数据，它可以是单个元素，也可以是列表、元组等。

```
l = ['Python', 'C++', 'Java']
#追加元素
l.append('PHP')
print(l)

#追加元组,整个元组被当成一个元素
t = ('JavaScript', 'C#', 'Go')
l.append(t)
print(l)

#追加列表,整个列表也被当成一个元素
l.append(['Ruby', 'SQL'])
print(l)
```

运行结果为：

```
'Python', 'C++', 'Java', 'PHP']
['Python', 'C++', 'Java', 'PHP', ('JavaScript', 'C#', 'Go')]
['Python', 'C++', 'Java', 'PHP', ('JavaScript', 'C#', 'Go'), ['Ruby',
'SQL']]
```

可以看到，当给 append( ) 方法传递列表或元组时，此方法会将它们视为一个整体，作为一个元素添加到列表中，从而形成包含列表和元组的新列表。

append 函数用法示例：

```
打印一个数的所有因子(如 8 的因子为 1,2,4,8)
a=eval(input('请您输入一个正整数'))
if a>0:
    yinzi=[]
    for i in range(1,a+1):
      if a% i==0:
        yinzi.append(i)
    print('{}的因子有:'.format(a))
    print(yinzi)
else:
    print('您输入的不是正数')
```

运行结果：

```
请您输入一个正整数:16
16 的因子有:
[1, 2, 4, 8, 16]
```

② extend( ) 方法添加元素。extend( ) 方法和 append( ) 方法的不同之处在于：extend( ) 不会把列表或元组视为一个整体，而是把它们包含的元素逐个添加到列表中。

extend( ) 方法的语法格式如下：

```
listname.extend(obj)
```

其中，listname 指的是要添加元素的列表；obj 表示要添加到列表末尾的数据，它可以是单个元素，也可以是列表、元组等，但不能是单个的数字。

```
l =['Python', 'C++', 'Java']
#追加元素
l.extend('C')
print(l)
```