

第三章 Python 编程基础

通过前一章节的学习，我们已完成了 Python 编程环境的搭建，以及如何在 Python 交互模式下编写并执行代码等基础知识。从本章开始，我们将正式开始学习 Python，体验 Python 给我们带来的快乐。本章将介绍 Python 语法特点、变量、基本数据类型、运算符与表达式、程序流程控制、正则表达式等基础知识。

学习计算机语言，必须经历的过程是敲代码——实战，在这个过程中一定会出现很多错误，只有从这些错误中不断总结经验，最后才能养成自己的编程风格，构建自己的知识体系，养成核心素养。我们约定：①凡是在代码部分出现的标点符号一律视为英文输入方式下的标点符号；②为了测试一条或几条语句的功能在 IDLE 的交互模式下执行，需要多条语句才能实现的功能使用 py 源文件（脚本）方式执行，并且在代码的前面使用 01、02、03……等标识，以便观察缩进量。

3.1 Python 语法特点

3.1.1 Python 的编程模式

1. 交互模式

交互模式是指在 Python 的“>>>”提示符后面直接编写代码的模式，例子：

```
>>>print('人生苦短，我用 Python!') #打印输出  
人生苦短，我用 Python!           #输出结果
```

Python 交互模式有以下几点需要注意：

①只能够输入 Python 命令，在 Python 交互模式下输入 Python 代码，而不要输入系统(DOS)命令；

②在交互模式下打印语句不是必须的，在交互模式下可以不输入打印语句，解释器会自动打印表达式的结果，但是在 py 脚本文件中则需要写 print 语句来输出结果；

③当在交互模式下输入两行或多行复合语句时，提示符会由>>> 变成 …（或空白），如果要结束复合语句的输入，需要连续按下两次 Enter 键；

④交互模式下一次运行一条语句，当你想测试某一条命令的时候，交互模式是一个很好的选择，回车即可看到执行结果，非常方便。

2. 脚本模式

就是利用前面的 IDE 编程工具，把 Python 程序代码以文件形式（.py 为扩展名）保存，然后以“python 脚本文件名”的形式运行程序的模式。

3.1.2 标识符与保留字

1. 标识符

标识符是对象的名字，如变量、函数的名字等，用于区分各个不同的对象。与人的名字一样，通过人名就能找到这个人。

在 Python 中，对标识符命名需要遵循的规则和规范如下：

- ①标识符可以包含字母、数字及下划线“_”，不能包含特殊字符，如\$、%、@等；
- ②第一个字符不能是数字；
- ③对字母大小写敏感；
- ④以单下划线（_）开头、双下划线（__）开头的标识符在类中有特殊的意义，一般情况不建议使用；
- ⑤虽然汉字也能作为标识符，但不建议使用；
- ⑥标识符不能是保留字。
- ⑦标识符尽量能“望文知义”，不建议用 a、b、c 等。

列举几个合法的标识符：name、my_age、num1、One

列举几个不合法的标识符：3name、my\$page、for

2. 保留字

保留字是一些具有特殊意义的字母组合，如 if、and、for 等，不能把这些保留字作为对象的名字。截止版本 3.7，Python 共定义了 33 个保留字，如表 3.1 所示。

表 3.1 Python 保留字

and	as	assert	break	class	continue
def	del	elif	else	except	finally
for	from	False	global	if	import
in	is	lambda	nonlocal	not	None
or	pass	raise	return	try	True
while	with	yield			

这些保留字中含大写字母的只有 True、False、None，其它全为小写字母。由于 Python 区分大小写，in 和 IN 是不一样的，IN 不是保留字。如果需要查看有哪些保留字，可以使用如下语句查看：

```
>>>import keyword
>>>keyword.kwlist
['False', 'None', 'True', 'and', 'as', 'assert', 'break', 'class', 'continue',
'def', 'del', 'elif', 'else', 'except', 'finally', 'for', 'from', 'global', 'if',
'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise', 'return',
'try', 'while', 'with', 'yield']
```

说明：可以在一行上书写多条语句也可以把一条语句写在多行上。如：

```
>>> print ('hello');print (' good morning')
```

在同一行书写多条语句时，使用分号“;” 隔开这些语句。

```
>>>print('made \
      in china' )
```

如果语句很长，可以使用反斜杠“\” 来实现续行书写。

3.1.3 缩进与注释

1. 缩进

```
01 # *_ coding:utf-8 *_
02 """演示代码缩进、注释"""
03 print("人生苦短，我学Python")
04 print("(=^=)")
```

什么叫缩进？在写文章时往往一个段落的第一行要空2个空格，这2个空格就叫做缩进。Python 对代码的缩进有非常严格的要求，同一代码块必须具有相同的缩进量，否则程序不能运行。在上面的代码中，如果把任意一行缩进1个空格，将会抛出“unexpected indent”异常，意为“意外的缩进”。

代码块：也叫语句块，是指具有相同缩进量的连续的语句所组成的语句集。在上面的代码中，由于这4行具有相同的缩进量，因此是1个语句块。再看一个例子：

```
01 num = int(input("请输入一个数字："))
02 sum = 0
03 n = len(str(num))
04 temp = num
05 while temp > 0:
06     digit = temp % 10
07     sum += digit ** n
08     temp //= 10
09 if num == sum:
```

```

10     print(num, "是阿姆斯特朗数")
11 else:
12     print(num, "不是阿姆斯特朗数")

```

01、02、03、04、05、09、11 是一个语句块；06、07、08 是一个语句块；10 单独一行作为一个语句块，12 单独一行作为一个语句块。

我们把没有任何缩进的语句块称为**主语句块**。除主语句块外，其它任何语句块的前面必须有一个“:”。

下面列出关于代码缩进的一些规则和规范：

- ①同一语句块必须具有相同的缩进量；
- ②当一个语句的后面出现“:”时，下面的代码必须相对于该行要有缩进；
- ③建议一个缩进量用 4 个空格，切记不要出现制表符（Tab）和空格混用。

3. 注释

在实际开发中，一个文件中的代码往往会很多，功能各异，时间久了，程序员自己再阅读这些代码也很困难。在团队开发中，程序员之间相互交换阅读代码也是必要的。为了让他人和自己了解代码实现的功能，就需要写注释。注释的内容会被解释器忽略，视而不见。Python 有两种方式写注释，分别是单行注释和多行注释。

单行注释：使用#作为单行注释的符号，以#开始直到行尾为止的所有内容都是注释的内容。

多行注释：使用成对的三个单引号（'''）或者三个双引号（"""）包裹的所有内容为多行注释的内容。如：

```

"""
    作者：XXXXX
    设计日期：2018-09-01
    版权所有：XXX@2018
"""

```

说明：在 Python 的 IDLE 中，1.单行注释的快捷键：选择需要注释的代码，Alt+3 增加注释，Alt+4 取消注释。2.多行注释的实质是一个字符串，如果该字符串在当前语义中被引用，就不再是注释了。

3.2 内置函数与库函数

3.2.1 函数基础知识

在计算机中，函数是能完成一定功能、可以被重复使用的代码块。1 个函数可以有 0 个

或多个参数，可以有 0 个或 1 个返回值。调用函数的语法格式如下：

```
funname(para1, para2, ...)
```

para1, para2, ……这些用“,”分开的叫做参数。本质上，函数的功能就是将这些参数根据需要进行相应运算并返回值。我们用加工“宫爆鸡丁”来打个简单比方，所有原料，如鸡肉、辣椒、油、盐等叫做参数，最后炒好的“宫爆鸡丁”叫做函数的返回值。数学上的 $y = f(x)$ ， f 就是函数， x 就是参数。 y 就是将参数 x 根据 f 规则进行计算、变换得到的值。

一个函数的参数个数并不总是固定的，如 `print()` 函数，可以不传参数，也可以传 1 个或多个参数。有些函数的参数个数是固定的。如求绝对值函数 `abs()`，有且只有 1 个参数。另外，并不是所有函数都有返回值，如 `print()` 函数只是打印信息，没有返回值。掌握函数的功能和使用方法是学好编程的关键。

编程语言的设计工程师为我们开发好了一些函数，我们直接调用即可，这种函数叫做内置函数，但我们也可以根据自己编写函数，这样的函数叫自定义函数。两者在本质是一致的，只是编写人员不一样而矣。

3.2.2 内置函数

Python 提供了许多内置函数（启动 Python 时已经加载到内存中，可直接调用），可以通过：`dir(__builtins__)` 查看具有哪些内置函数，`help(函数名)` 查看具体函数的使用说明。

Python3.7.0 内置函数列表

<code>abs()</code>	<code>all()</code>	<code>any()</code>	<code>ascii()</code>	<code>bin()</code>	<code>bool()</code>	<code>breakpoint()</code>	<code>bytearray()</code>
<code>bytes()</code>	<code>callable()</code>	<code>chr()</code>	<code>classmethod()</code>	<code>compile()</code>	<code>complex()</code>	<code>copyright()</code>	<code>credits()</code>
<code>delattr()</code>	<code>dict()</code>	<code>dir()</code>	<code>divmod()</code>	<code>enumerate()</code>	<code>eval()</code>	<code>exec()</code>	<code>exit()</code>
<code>filter()</code>	<code>float()</code>	<code>format()</code>	<code>frozenset()</code>	<code>getattr()</code>	<code>globals()</code>	<code>hasattr()</code>	<code>hash()</code>
<code>help()</code>	<code>hex()</code>	<code>id()</code>	<code>input()</code>	<code>int()</code>	<code>isinstance()</code>	<code>issubclass()</code>	<code>iter()</code>
<code>len()</code>	<code>license()</code>	<code>list()</code>	<code>locals()</code>	<code>map()</code>	<code>max()</code>	<code>memoryview()</code>	<code>min()</code>
<code>next()</code>	<code>object()</code>	<code>oct()</code>	<code>open()</code>	<code>ord()</code>	<code>pow()</code>	<code>print()</code>	<code>property()</code>
<code>quit()</code>	<code>range()</code>	<code>repr()</code>	<code>reversed()</code>	<code>round()</code>	<code>set()</code>	<code>setattr()</code>	<code>slice()</code>
<code>sorted()</code>	<code>staticmethod()</code>	<code>str()</code>	<code>sum()</code>	<code>super()</code>	<code>tuple()</code>	<code>type()</code>	<code>vars()</code>
<code>zip()</code>							

了解了某个具体函数的功能及对参数的要求后，使用形如：函数名(参数 1, 参数 2, ……)的方式调用即可。这里需要强调的是：你不需要记住每个函数对参数的具体要求，只需要大概知道这些函数都有哪些功能。在实际编写程序时“百度一下”就可以了。

3.2.3 几个基本输入输出函数

程序从键盘（鼠标）读入数据、向屏幕输出信息是最基本的操作，为了方便上机实践，我们先介绍几个与输入输出相关的内置函数。

1. input()函数

功能：接收用户的键盘输入，返回字符串，语法格式如下：

```
varname = input([prompt])
```

varname: 变量，以字符串类型保存输入结果。prompt: 参数，提示信息，可以省略。

例子：

```
>>>name = input('请输入你的名字：')
```

```
>>>请输入你的名字：张三
```

```
>>>name
```

```
>>>'张三'
```

说明：在介绍函数的语法时，参数中凡是用“[]”括起来的表示可以省略。

3. print()函数

功能：打印输出，无返回值，语法格式如下：

```
print([*objects][, sep=' '][, end='\n'][, file=sys.stdout])
```

objects: 参数，一个或多个对象，对象可以是值、变量、表达式。如果有多个变量要使用“,”隔开。

sep: 参数，输出时多个对象之间的间隔符号，默认值是一个空格。

end: 参数，结尾符号，默认是换行。

file: 参数，要写入的文件对象。

实践：请在 IDLE 的交互模式下运行以下代码，并认真体会总结。

```
>>>a = 3
```

```
>>>b = 12
```

```
>>>c = a+b
```

```
>>>print(a, b, c) #3 12 15
```

```
>>>print(a, b, c, sep=':') #3:12:15
```

```
>>>print(a, b, c, end='。') #3 12 15。
```

>>>fp = open(r'd:\test.txt', 'a+') #在 d 盘上创建 test.txt 文件，r 的作用是防止字符转义

```
>>>print(a, b, c, file=fp) #把数据写入缓冲区
```

```
>>>fp.close() #写入数据，关闭文件
```

```
>>>print(a, '+', b, '=', c) # 3+12=15
```

```
>>>print(' %d+%d=%d' %(a, b, c))      # 3+12=15
>>>print(' {}+{}={}'.format(a, b, c))  # 3+12=15
```

小结

(1) print()函数不仅可以向屏幕输出数据，也可以向文件输出数据；

(2) print()函数有两种方法实现格式化输出：

①模式字符串：以“%”开始，一些特殊字母结束（如 d、f、s、x、r 等），中间可以是一些格式修饰符号（如+、数字等）

②字符串对象的 format()方法。后续介绍。

(3) int()函数

功能：把一个数字或字符串转换成整数，基本语法格式如下：

```
varname = int(x)
```

x: 参数，字符串或数字。

例子：

```
temp = int(3.14)           #3
temp = int('456')        #456
```

3.2.4 库函数

有时我们会说“XXX 对象的 XXX 方法”，如“字符串对象提供的方法”，为什么不叫“字符串对象提供的函数”呢？其实这只是一种习惯，习惯上，把与具体对象无关的称为函数，把只能作用于特定对象的称为方法。举个例子：

在 `str = input().strip().split()` 中，`input()` 称为函数，`strip()` 和 `split()` 都是字符串对象提供的，只能作用于字符串，`strip()`、`split()` 就称为方法。这个语句的执行过程是：`input()` 首先返回一个字符串对象，然后调用字符串对象的 `strip()` 方法去掉前后的空格，得到一个新的字符串，再调用字符串对象的 `split()` 方法分隔字符串，最后将结果赋值给变量 `str`。

传说，Python 的功能很强大，怎么内置函数就那么几十个？就连最基本的正弦、余弦函数都没有？其实 Python 是把很多功能捆绑在了称为库（模块）的对象上面，需要通过模块来调用。在安装 Python 时已经安装的库叫做标准库，需要单独下载安装库的叫做第三方库。两者本质上没有任何差别。在使用这些库时需要先导入才能使用。可以使用“import 库名”导入。例子：

```
import math          #导入 math 库
a = math.sin(2)     #调用 math 库的 sin() 函数
```

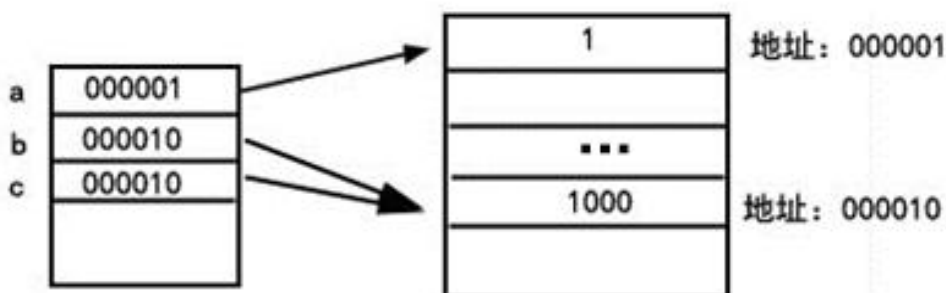
一般情况下，库函数的调用格式为：模块名.函数名（参数 1，参数 2，……）。与内置函数比较，除了前面需要指明模块名和一个点（.）外，其它完全一样。这里把点（.）理解为“的”的意思。Python 之所以强大，原因之一是他有非常丰富和强大的标准库和第三方库，

几乎你想实现的任何功能都有相应的 Python 库支持，你只需要把算法想好，然后调用这些方法就可以了。具体参见附录。当然，你也可以开发 Python 库放在 <https://github.com/>上供大家使用。

3.3 变量与表达式

3.3.1 变量

在前面我们对变量已有了感性认识，在这一节对变量进行详细说明，变量是标识符，其值是某个内存单元的地址，并不存放真正的数据。如执行语句 `a=1`，`b=1000`，`c=b`，结合下图说明执行过程：



`a = 1` 的执行过程：计算机在内存中找一块空间（假设这块空间的地址为 000001）存入数字 1，然后在另外一个地方找一个空间存放这个地址，并把这个空间命名为 `a`。

`b = 1000` 的执行过程：计算机在内存中找一块空间（假设这块空间的地址为 000010）存入数字 1000，然后在另外一个地方找一个空间存放这个地址，并把这个空间命名为 `b`。

`c = b` 的执行过程：找到 `b` 存储的地址（000010），然后在另外一个地方找一个空间存放 `b` 存储的地址（000010）并把这个空间命名为 `c`。

如果继续执行 `c = c+1`，其执行过程是：取出 `c` 指向的内存中的值（1000），然后进行加 1 运算得到 1001，然后在内存中找一块空间（假设这块空间的地址为 000015）存入数字 1001，最后再把 `c` 中的值修改为 000015。

Python 中变量是一种指向，可以指向任何对象。

说明：对于刚进入编程世界的你要理解这个过程可能有一定难度，但为了今后的顺利学习，一定要理解。

如果需要查看变量所指向的地址可以使用 `id()` 函数，如：

```
>>>a = 100
```

```
>>>id(a)
```

```
>>>1608042608
```

#你实验的未必是这个数字

由于 Python 中的变量是一种指向，存储的是对象的地址，在不同时刻可以指向不同对

象，所以不需要声明变量的类型。这就是认为 Python 是一种动态数据类型语言的原因。在访问变量时，该变量必须是存在的。否则会引发 NameError 异常。

实践：请在 IDLE 的交互模式下运行以下代码。

```
>>>a=100
>>>b=100
>>>a is b           #True
>>>c=300
>>>d=300
>>>c is d           #False
```

这是由于 Python 对一些数据采用了缓存机制，请自行查阅其它资料学习。

说明：

(1) 如果需要查看变量的类型可以使用内置函数 `type(变量名)`。

(2) 变量的命名规则参见标识符。

(3) Python 没有常量（不能被修改的量）的定义方法，一个习惯是使用全部大写的标识符来表示，如：`PI = 3.1415926`。

3.3.2 表达式

表达式与数学中的代数式类似，是指由变量和运算符组合而成的式子。特别地，单独的一个值或单独的变量也是一个表达式。表达式中不能含有“=”。

例子：`3`、`a`、`a+b`、`int('3')+8`、`a>b`、`a and b`、`(c==1) or (d is e)`都是表达式。`a=3+5`不是表达式。

知识拓展：Python 支持多变量赋值，即在一个语句中，支持对多个变量同时赋值，如：

```
>>> a, b, c = 4, 8, 'John'
```

相当于执行 `a = 4, b = 8, c = 'John'`，三条语句，采用这种方式赋值时，要注意的是左边变量的个数要与右边值的个数（或解包后值的个数）相等。

3.4 基本数据类型

不同的数据所表达的意义和在计算机内部的存储方式是不一样的，比如：`3`和`'3'`，`3`表示数量，`'3'`表示一个符号。存储`3`可能需要4个字节，存储`'3'`只需1个字节。根据数据在计算机中存储方式的不同把数据划分成不同的类型，称为数据类型。Python 中的数据类型有很多，本节介绍数字类型、字符串类型、布尔类型。

3.4.1 数字类型

数字类型是指表示大小、多少等的计量。Python 中数字类型主要包括整数、浮点数和复数。

1. 整数

整数包括正整数、0、负整数，没有位数限制。可以用十进制、二进制、八进制、十六进制的形式表示。用十进制数表示时不能以 0 开头。

二进制以 0b 或 0B 开头，八进制以 0o 或 0O 开头，十六进制以 0x 或 0X 开头。

如：0b1001011、0B111000111 表示二进制数。0o345670、0O54332 表示八进制数。0x34AE32、0X76fB 表示十六进制数。

实践：请在 IDLE 的交互模式下运行以下代码。

```
>>>a = 0b111111000111
>>>print(a)           #输出：4039
```

说明：默认是以十进制的形式输出，如果需要以其它进制输出，可以使用如下语句：

```
>>>a = 100
>>>print("%x,%o,%d,%s"%(a, a, a, bin(a)))
```

这里 s、o、d、s 分别表示十六进制、八进制、十进制、字符串格式。由于字符串格式化代码没有提供二进制格式，这里使用了 bin() 函数先把数值转换为二进制后再以字符串的格式输出。

2. 浮点数

浮点数是指带小数点的数，如：3.11、2.0、3.15。浮点数的位数没有限制。可以用科学计数法表示，如 4.5×10^3 可写成 4.5e3，e 后面的数字只能是整数，不能是浮点数。对于非常大的数或非常小的数用科学计数法表示很方便。

实践：请在 IDLE 的交互模式下运行以下代码。

```
>>>a=3.3
>>>b=4.5
>>>a+b
>>>a-b      #-1.2000000000000002
>>>a*b
>>>a**b      #幂运算
>>>a/b       #除法运算
>>>a//b      #整除运算
>>>a%b       #求余运算
```

a-b 并不等于-1.2，这与 Python 存储浮点数时的精度有关，存在误差，在处理实际问题

时应根据精度需要保留适当位小数即可。

3.复数

在形式上 Python 中的复数与数学中的复数完全一样，只是虚部使用 j（或 J）而不使用 i。

如：2+3j。

3.4.2 字符串类型

1.字符串的定义

字符串是由单引号（'）或双引号（"）或三引号（'''）括起来的字符序列，是 Python 中常用的数据类型，如表示名字的“迈克尔”，表示水果的“石榴”等。在表示字符串时：

（1）字符串的开始和结尾的引号必须一致，比如不能开头使用单引号，结尾使用双引号。

（2）使用单引号或双引号的字符串必须写在一行上。

（3）使用三引号可以将字符串写在连续的多行上。

（4）引号可以嵌套使用。如：book_name = "‘经典教材’系列之 Python"。

2.转义字符

在字符串中，有时需要表示一些特殊的控制字符，如换行、Tab 制表符、引号、退格键等，这些字符不能直接输入，只能使用一些特殊字符代替。Python 使用反斜杠“\”加一些特殊字符进行转义。常用转义字符表 3.3 所示。

表 3.3 常用转义字符

转义字符	描述	转义字符	描述
\	续行	\n	换行
\\	反斜杠\	\'	单引号
\"	双引号	\a	响铃
\b	退格	\0	空白
\t	水平制表符	\v	垂直制表符
\f	换页	\r	回车
\odd	八进制数，dd 代表字符	\xhh	十六进制数，hh 代表字符

说明：如果字符串本身需要有类似于“\t”这样的内容，即使“\t”不进行转义，只要在字符串前面加上 r（或 R）即可。

```
>>>a=r'aaa\tbbb'
```

```
>>>print(a)
```

输出：aaa\tbbb

4. 字符串对象提供的常用方法

Python 中字符串对象提供了很多方法可以实现强大的字符串处理功能。为便于后续内容的学习，本节列表几个常用方法，更详尽的内容参见后面的章节。

表 3.4 字符串处理功能

方法名称	语法	描述
lower	str.lower()	将字符串 str 转换为小写
upper	str.upper()	将字符串 str 转换为大写
title	str.title()	所有单词首字母大写且其他字母小写的格式
capitalize	str.capitalize()	首字母大写、其他字母全部小写
swapcase	str.swapcase()	所有字母做大小写转换
isdigit	str.isdigit()	测试字符串 str 是否是数字
isalpha	str.isalpha()	测试字符串 str 是否是字母
isalnum	str.isalnum()	测试字符串 str 是否是数字或字母
center	str.center(with, [, fillchar])	将字符串居中，左右两边使用 fillchar 进行填充，使得整个字符串的长度为 width。fillchar 默认为空格
ljust	str.ljust(with, [, fillchar])	字符串居左，右边使用 fillchar 进行填充
rjust	str.rjust(with, [, fillchar])	字符串居右，左边使用 fillchar 进行填充
endswith	str.endswith(suffix[, start[, end]])	检查字符串 str 是否以 suffix 结尾
startswith	str.startswith(prefix[, start[, end]])	检查字符串 str 是否以 prefix 开始
replace	str.replace(old, new[, count])	将字符串中的子串 old 替换为 new 字符串，如果给定 count，则表示只替换前 count 个 old 子串
split	str.split(sep=None, maxsplit=-1)	根据 sep 对 str 进行分割，maxsplit 用于指定分割次数，sep 默认为空格。并生成一个列表
strip	str.strip([chars])	移除左右两边的 chars，chars 默认为空格
rstrip	str.rstrip([chars])	移除右边的 chars，chars 默认为空格
lstrip	str.lstrip([chars])	移除左边的 chars，chars 默认为空格

说明：可以使用内置函数 len(str)计算字符串的长度。

3.4.3 布尔类型

布尔类型用来表示“真”和“假”，分别用标识符 True、False 表示。布尔值也可以转换为数值，True 表示 1，False 表示 0。

Python 中的所有对象都可以进行真值测试。

★Python 提供了 `NoneType`，即空类型，表示“什么都没有”，用 `None` 表示。既不表示空白字符也不表示数值 0。

3.4.4 数据类型转换

尽管 Python 不需要声明变量的类型，但有时还是需要进行类型转换，比如要计算从键盘输入的两个数的和就需要使用 `int()` 函数进行转换，否则就会引发 `TypeError` 异常。下面列出一些常用的类型转换函数。

表 3.5 常用的类型转换函数

函数	描述
<code>int(x)</code>	将 <code>x</code> 转换成整数类型
<code>float(x)</code>	将 <code>x</code> 转换成浮点数类型
<code>complex(real[, imag])</code>	创建一个复数， <code>real</code> 实部， <code>imag</code> 虚部
<code>str(x)</code>	将 <code>x</code> 转换成字符串
<code>repr(x)</code>	将 <code>x</code> 转换成表达式字符串
<code>eval(str)</code>	计算 <code>str</code> 中的有效 Python 表达式，并返回一个对象
<code>chr(x)</code>	将 <code>ascii</code> 码 <code>x</code> 转换为对应的一个字符
<code>ord(x)</code>	将字符 <code>x</code> 转换为对应的 <code>ascii</code> 码
<code>hex(x)</code>	将整数 <code>x</code> 转换为对应的十六进制字符串
<code>oct(x)</code>	将整数 <code>x</code> 转换为对应的八进制字符串

注意：`eval(str)` 中的 `str` 只能是字符串表达式。

说明：Python 的每个对象都分为可变和不可变，请你先记住：数字、字符串、元组是不可变的，列表、字典是可变的，详细介绍在第五章叙述。

3.5 运算符

在表达式“1+2”中，+叫作运算符，1、2叫作操作数。运算符的意义是规定操作数的运算规则。Python 中的运算符主要包括算术运算符、赋值运算符、关系运算符、逻辑运算符、位运算符。

3.5.1 算术运算符

算术运算符如表 3.6 所示。

表 3.6 算术运算符实例

运算符	描述	实例
+	两个对象相加	$a + b$ 输出结果 6
-	得到负数或是一个数减去另一个数	$a - b$ 输出结果 -2
*	两个数相乘或是返回一个被重复若干次的字符串	$a * b$ 输出结果 8
/	两个数相除	a / b 输出结果 0.5
%	除法的余数	$a \% b$ 输出结果 2
**	幂运算	$a^{**}b$ 为 2 的 4 次方, 输出结果 16
//	商的整数部分	$7 // 2$ 输出结果 3, $7.4 // 2$ 输出结果 3.0

注：实例中， $a=2$ ， $b=4$ 。

3.5.2 赋值运算符

赋值运算符如表 3.7 所示。

表 3.7 赋值运算符实例

运算符	描述	实例
=	简单的赋值运算符	$c = a + b$ 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 $c = c ** a$
//=	取整除赋值运算符	$c // = a$ 等价于 $c = c // a$

3.5.3 关系运算符

关系运算符如表 3.8 所示。

表 3.8 关系运算符 示例中, a=2, b=4

运算符	描述	实例
==	比较对象是否相等	(a == b) 返回 False
!=	比较两个对象是否不相等	(a != b) 返回 True
>	比较 a 是否大于 b	(a > b) 返回 False
<	比较 a 是否小于 b	(a < b) 返回 True
>=	比较 a 是否大于或等于 b	(a >= b) 返回 False
<=	比较 a 是否小于或等于 b	(a <= b) 返回 True

注意: Python3.X 不支持<>运算符。

3.5.4 逻辑运算符

逻辑运算符如表 3.9 所示

表 3.9 逻辑运算符

运算符	描述	实例
and	如果 a 为 False, 返回 False, 否则返回 b 的值	a and b
or	如果 a 为 True, 返回 True, 否则返回 b 的值	a or b
not	如果 a 为 True, 返回 False, 否则返回 True	not a

说明: 可以使用内置函数 bool(参数)将参数转换为布尔值, 参数可以是一个复杂的表达式。

实践: 请在 IDLE 的交互模式下运行以下代码。

```
>>>bool(1)
>>>bool(None)
>>>bool(3>=4)
>>>3 and 8
>>>bool(3 and 8)
>>>0 or 5
```

3.5.5 位运算符

Python 中,位运算符是指对操作数的二进制位进行操作的运算符。其操作数只能是整数,运算规则如表 3.10 所示。

表 3.10 位运算符实例

运算符	描述	实例
&	按位与运算符：如果参与运算的两个值的相应位都为 1，则结果为 1，否则为 0	(a & b) 的值为(0000 1100) ₂ 或(12) ₁₀
	按位或运算符：只要对应的二个二进制位有一个为 1 时，结果位就为 1。	(a b) 的值为(0011 1101) ₂ 或(61) ₁₀
^	按位异或运算符：当两对应的二进制位相异时，结果为 1	(a ^ b) 的值为(0011 0001) ₂ 或 (49) ₁₀
~	按位取反运算符：对数据的每个二进制位取反,即把 1 变为 0，把 0 变为 1	(~a) 的值为(1100 0011) ₂ 或 (-61) ₁₀
<<	左移运算符：对操作数的二进制位全部左移若干位，由<<右边的数字指定移动的位数，高位丢弃，低位补 0。	a << 2 的值为(1111 0000) ₂ 或(240) ₁₀
>>	右移运算符：对操作数的二进制位（除符号位外）全部右移若干位，由>>右边的数字指定移动的位数，符号位不变，高位补 0。	a >> 2 的值为(0000 1111) ₂ 或 (15) ₁₀

实例中，a = 60，b = 13。二进制形式：a = (0011 1100)₂，b = (0000 1101)₂

实践：请在 IDLE 的交互模式下运行以下代码。

```
>>>2<<2           #输出 8
>>>2<<4           #输出 32
>>>9>>2           #输出 2
>>>9>>1           #输出 1
>>>6^5             #输出 3
>>>3^5             #输出 6
```

请总结<<、>>、^的运算规律。如果你与你的朋友想说“暗语”，可以怎么做呢？

3.5.6 成员运算符与身份运算符

成员运算符是判断 1 个对象是不是某个序列中的元素的运算符。如：集合 A={1,2,3,4}，判断 1 在 A 中吗？就可以使用 1 in A 进行判断。

表 3.11 成员运算符实例

运算符	描述	实例
in	如果在指定的序列中找到值返回 True，否则返回 False。	x in y，如果 x 在序列 y 中返回 True，否则返回 False。
not in	如果在指定的序列中没有找到值返回 True，否则返回 False。	x not in y，如果 x 不在序列 y 中返回 True，否则返回 False。

身份运算符是判断两个对象是不是同一个对象的运算符，如表 3.12 所示。

表 3.12 成员运算符实例

运算符	描述	实例
is	is 是判断两个对象是不是指向同一个对象	x is y, 等价于 id(x) == id(y), 如果指向的是同一个对象则返回 True, 否则返回 False
is not	is not 是判断两个对象是不是指向不同对象	x is not y, 等价于 id(x) != id(y)。如果指向的不是同一个对象则返回 True, 否则返回 False。

实践：请在 IDLE 的交互模式下运行以下代码。

```
>>>a = 300
>>>b = 300
>>>a is b           #输出 False
>>>a == b          #输出 True
>>>c=d=300
>>>c is d           #输出 True
```

小结：(1) == 是比较值是否相等，is 是比较内存地址是否相等。(2) 如果需要查看对象的内存地址，可以使用内置函数 id(对象)获取。

3.5.7 运算符的优先级

运算符的优先级是指在一个式子中，如果有多个运算符，哪个运算符先运算，哪个后运算。与数学中的“先乘除，后加减”是一样的。下表列出了从高到低优先级的所有运算符。

运算符	描述
**	指数 (最高优先级)
~、+、-	取反、正号、负号
*、/、%、//	乘、除、取模、整除
+、-	加法减法
>>、<<	右移、左移运算符
&	按位与运算符
^、	按位异或运算符、按位或运算符
<=、<、>、>=、==、!=	比较运算符
=、%=、/=、//=、-=、+=、*=、**=	赋值运算符
is、is not	身份运算符
in、not in	成员运算符
not、and、or	逻辑运算符

运算符的运算规则是：优先级高的先执行，优先级低的后执行，同一优先级的按照从左到右的顺序执行。例子： $2+3**6//2\%3$ ，由于 $**$ 的优先级最高，先执行，表达式转变为： $2+729//2\%3$ ， $//$ 和 $\%$ 具有相同优先级，先执行 $//$ ，表达式转变为： $2+364\%3$ ， $\%$ 的优先级高于 $+$ ，先执行，表达式转变为： $2+1$ ，最后执行 $+$ ，结果为 3。

实际上，无需记住这些运算符的优先级，因为可以使用 $()$ 来改变运算顺序，从而避免发生逻辑错误。如上面这个表达式可以写成： $2+(((3**6)//2)\%3)$ 。

3.6 流程控制语句

我们在前面编写的绝大多数代码中，程序都是按照书写顺序从上往下执行，直到所有语句执行完毕为止。但是，仅靠这种顺序执行方式并不能完全满足实际需求，比如：当用户输入一个整数，如果这个数是偶数，则打印“偶数”，否则打印“奇数”。这个问题显然不能用顺序执行的方式来模拟，程序需要先进行判断，然后再根据判断的结果有选择地执行相应语句。即有时需要一些可以改变程序运行顺序的指令才能解决某些问题。本节我们就来学习 Python 中关于改变程序运行顺序（程序流程控制）方面的知识。

实际上，计算机在解决某个具体问题时，主要有 3 种情形：顺序执行所有语句、选择执行部分语句、循环执行部分语句。事实证明，任何一个能用计算机解决的问题，都可运用这三种基本结构来编写程序。

3.6.1 选择语句

首先我们用中文写出前面的要求：

- 01 用户输入一个整数 (n)
- 02 如果这个整数 (n) 是偶数，那么：
- 03 打印“偶数”
- 04 否则：
- 05 打印“奇数”

Python 中需要这样翻译上面这段代码，计算机才能识别：

```
01 n = int(input())
02 if n 是偶数 :
03     print('偶数' )
04 else:
05     print('奇数' )
```

即：“如果”用关键字“if”表示，“否则”用关键字“else”表示。

Python 中选择语句主要有 3 种形式：(1) if 语句；(2) if ...else 语句；(3) if ...elif...else 语句。

1. if 语句

if 语句的语法格式如下：

```
if 表达式：
    语句块
```

当表达式的值为 True 时，则执行语句块，如果值为 False，则不执行语句块。有两点需要注意：一是表达式后面需要一个“:”，二是语句块中的每条语句需要具有相同的缩进量，缩进量的规范是相对于前面 if 的位置缩进 4 个空格。

例子：分析、实践比较下面两段代码的执行结果。

```
age = 16                                age = 16
if age > 18:                             if age > 18:
    print('你是成年人')                 print('你是成年人')
    print('你还不是成年人')           print('你还不是成年人')
```

说明：在前面的叙述中，“表达式的值为 True”是指表达式的值可以通过 bool() 函数转换为 True。后面凡是说‘表达式的值为 True’都与此类似，不再赘述。

2. if...else 语句

if ...else 语句的语法格式如下：

```
if 表达式：
    语句块 1
else：
    语句块 2
```

这种结构是一种二选一的结构，根据表达式的值，如果值为 True，程序执行语句块 1，否则执行语句块 2。相当于汉语中的“如果……就……，否则……就……”语句。

例子：假设某年高考二本划线 500 分，请编写一段代码判断某个学生是否能上二本。程序代码如下：

```
01 score = int(input('请输入学生成绩：'))
02 if score >= 500:
03     print('能上二本')
04 else:
05     print('不能上二本')
```

知识拓展：(1) 如果语句块中只有一条语句，可以直接书写在“:”的后面。例如，可

将上面代码的判断部分改写成（为了代码的可读性，并不推荐这样做）：

```
02 if score >=500: print('能上二本')
03 else: print('不能上二本')
```

（2）Python 没有三目运算符，是用条件表达式代替的。如：

```
01 var1 = 2
02 if var1 >0:
03     var2 = 1
04 else:
05     var2 = -1
```

可以改写成：`var2 = 1 if var1>0 else -1`。其执行逻辑是如果条件成立就返回 if 前面的值，否则就返回 else 后面表达式的值。

例子：假设 n 为一个整数，如果 n 既是 3 的倍数又是 7 的倍数，则输出“T”，否则输出“F”。

```
01 if n % 3==0 and n % 7 ==0:
02     print("T")
03 else:
04     print("F")
```

可改写为：`print ("T " if n % 3==0 and n % 7 ==0 else "F")`

3. if···elif···else 语句

if···elif···else 语句的语法格式如下：

if 表达式 1:

 语句块 1

elif 表达式 2:

 语句块 2

.....

else:

 语句块 n

这种结构是一种多选一的结构，其执行逻辑是首先判断表达式 1 的值，如果为 True 则执行语句块 1，如果为 False 则判断表达式 2 的值，如果为 True 则执行语句块 2，如此继续……，在这个过程中，一旦某个表达式的值为 True，在执行后面语句块后，不再判断后面的所有表达式。只有当所有表达式的值都为 False 时才执行 else 后面的语句块。

例子：判断学生成绩的等级，规则是：成绩小于 60 分为不合格，大于或等于 60 分小于

70分为合格，大于或等于70分小于80分为良好，大于80分为优秀。程序代码如下：

```
01 score = int(input('请输入学生成绩(整数): '))
02 if score < 60:
03     print('不合格')
04 elif score >= 60 and score < 70:
05     print('合格')
06 elif score >= 70 and score < 80:
07     print('良好')
08 else:
09     print('优秀')
```

如果将上面这段代码改写为：

```
01 score = int(input('请输入学生成绩(整数): '))
02 if score < 60:
03     print('不合格')
04 elif score < 70:
05     print('合格')
06 elif score < 80:
07     print('良好')
08 else:
09     print('优秀')
```

代码中的表达式并没有严格按规则书写，请你仔细想想这段代码能正确判断吗？为什么？

4. if 语句的嵌套

if 语句的嵌套是指在 if 语句的语句块中，还可以包含一个或多个 if 语句。语法格式如下：

if 表达式 1:

 if 表达式 2:

 语句块 1

 else:

 语句块 2

elif 表达式 3:

 语句块 3

理解嵌套结构只需把里面的 if 语句当成是一条语句即可。虽然在 Python 中对嵌套的层数没有限制，但如果层数过多会对理解代码的执行逻辑带来困难，因此，建议不要嵌套较多的层数。另外需要注意的是不同级别语句块的缩进量不同。

例子：当前，高校自主招生的名额正在逐年增加，为一些具有特长的学生进入理想高校拓宽了渠道，比如 2016 年参加“全国青少年信息学奥林匹克竞赛”获得省级一等奖，中国人民公安大学降至投档线下 30 分录取，南开大学降至投档线下 40 分录取。假设 2016 年中国人民公安大学的投档线是 530 分，南开大学的投档线是 600 分。请编写一段代码判断一个学生可以被哪些高校通过自主招生渠道录取。

输入要求：从键盘输入学生高考成绩和是否获得省级一等奖（用空格隔开）

输入样例：580 已获得 输出样例：中国人民公安大学 南开大学

输入样例：590 未获得 输出样例：不能通过自主招生渠道录取

程序代码如下：

```
01 score_str, get_str = input('请输入: ').strip().split()
02 score = int(score_str)
03 if get_str == '已获得':
04     if score >= 530 - 30:
05         print('中国人民公安大学', end=' ')
06     if score >= 600 - 40:
07         print('南开大学', end=' ')
08 else:
09     print('不能通过自主招生渠道录取')
```

说明：strip()方法的作用是去掉字符串前后的空白，split()方法的作用是把字符串按指定字符进行切片，本例中为按空格切片。01 行的左边有两个变量，这是 Python 中多变量赋值的写法。

3.6.2 循环语句

循环语句是指控制一段代码重复执行多次的语句。首先看一个实际生活中的情景：

在体育课堂中，长跑项目通常是在学校的运动场上沿跑道奔跑，只有当听到体育老师吹口哨的声音时才能停下来，如果体育老师一直不吹口哨，将跑完一圈又一圈，……

Python 中有两种方式实现这种循环结构，分别是 while 循环和 for 循环。下面分别介绍这两种循环结构。

1. while 循环

while 循环的基本语法如下：

while 表达式:

 语句块（循环体）

其执行逻辑是，首先判断表达式的值，如果为 True 则执行语句块，否则不执行语句块，当语句块执行完后，再次判断表达式的值，如果为 True 则执行语句块，否则不执行语句块，如此继续……。与 if 语句类似，一是表达式后面需要一个“:”，二是语句块中的每条语句需要具有相同的缩进量，缩进量的规范是相对于前面 while 的位置缩进 4 个空格。

例子：用 while 循环实现计算 $1+2+3+4+\dots+100$ 的和。代码如下：

```
01 sum = 0
02 i = 1
03 while i<=100:
04     sum += i
05     i += 1
06 print(sum)
```

代码执行过程分析：01、02 行为赋值语句，当执行到 03 行时，程序首先判断表达式 $i \leq 100$ 是否为 True，由于此时 i 的值为 1，表达式 $1 \leq 100$ 的值为 True，执行 $sum += i$ 和 $i += 1$ 这两条语句，语句块执行完毕，此时 i 的值变为 2，再次判断表达式 $i \leq 100$ 的值，由于表达式 $2 \leq 100$ 的值为 True，再次执行语句块，……，当执行 100 次以后，i 的值变为 101，由于表达式 $101 \leq 100$ 的值为 False，不再执行语句块，循环结束。

如果循环语句中表达式的值永远为 True，那么将无限次的循环执行语句块，这样的循环称为“死循环”。如果没有特别需要，不要将代码写成“死循环”。

例子：请编写程序求方程： $2x+y=100$ 在 [1, 100] 内的整数解。

分析：一个二元一次方程的解的个数可能有很多，不能利用数学上常规的通过变形、化简来求解。只能使用枚举法逐一尝试某个组合是否是方程的解。由于计算机的运算速度很快，非常适合使用枚举法进行求解。程序代码如下：

```
01 x =1
02 while x <=100:
03     y =100-2*x
04     if 1<=y<=100:
05         print("x={},y={}".format(x,y))
06     x +=1
```

知识拓展：Python 中没有 do……while 循环，但是 Python 的 while 循环支持 else 关键字。

语法格式如下：

```
while 表达式:
    语句块 1
else:
    语句块 2
```

执行逻辑是：先执行完语句块 1，再执行语句块 2，需要注意两点，一是语句块 2 是否被执行与表达式的值无关，二是当在语句块 1 中使用 `break` 关键字终止循环时，语句块 2 不会被执行。

2. for 循环

for 循环的基本语法如下：

for 变量 in 对象:

 语句块（循环体）

对象是指有一个或多个元素的序列，如字符串、以及后面将要介绍的列表、元组、集合、字典等对象。其执行逻辑是首先从对象中取出第一个元素赋值给变量，执行语句块，然后从对象中取出第二个元素，再执行语句块，……直到取完对象中的所有元素时为止。与 `while` 循环类似，语句块也需要缩进。下面看一个打印字符串中每个字符的例子。

```
01 sentence = "I am a student"
```

```
02 for ch in sentence:
```

```
03     print(ch, end=' ', )
```

将输出：I, , a, m, , a, , s, t, u, d, e, n, t,

在编程活动中把对一个对象中的每个元素进行一次且仅做一次的访问称为**遍历**。for 循环非常适合于对对象进行遍历。

range()函数介绍：range()函数是 Python 的内置函数，其作用是生成一个整数迭代器，多用于 for 循环中，语法格式如下：

```
range(start,end,step)
```

start：起始值，默认值为 0。

end：终止值，但不包括这个值。

step：步长（两个数之间的间隔），默认值为 1。

说明：除参数 end 外，其它参数都可以省略，参数的个数可能有 1 个或 2 个或 3 个。

```
range(10)           #只有 1 个参数时，这个参数 10 表示 end。
```

```
range(2,10)        #只有 2 个参数时，第 1 个参数 2 表示 start，第 2 个参数表示 end。
```

```
range(1,100,2)     #同时有三个参数时，分别表示 start、end、step。
```

例子：

range(5)将产生一个元素为 0,1,2,3,4 的 range 对象。

range(2,10)将产生一个元素为 2,3,4,5,6,7,8,9 的 range 对象。

range(2,10,2)将产生一个元素为 2,4,6,8 的 range 对象。

例子：请使用 for 循环实现 $1+2+3+4+\dots+100$ 的和。代码如下：

```
01 sum = 0
```



```

02 for i in range(1, 101):
03     sum += i
04 print(sum)

```

知识拓展：for 循环也支持 else 关键字。语法格式如下：

```

for 表达式:
    语句块 1
else:
    语句块 2

```

其执行逻辑与 while 循环完全相同。

3. 循环嵌套

与 if 嵌套类似，也可以在循环体中嵌入另外一个循环，称为循环嵌套。while 循环和 for 循环可以相互嵌套。下面用一个简单例子说明。

例子：打印九九乘法表。输出格式如下：

```

1 × 1 = 1
1 × 2 = 2   2 × 2 = 4
1 × 3 = 3   2 × 3 = 6   3 × 3 = 9
.....
1 × 9 = 3   2 × 9 = 18   3 × 9 = 27   4 × 9 = 36   .....

```

分析：仔细观察九九乘法表的结构，可以发现一共有 9 行，并且每行式子的个数与行号相等。我们用 n 表示行号，范围是 $[1,9]$ ， m 表示每行式子的个数，范围是 $[1,n]$ ；再观察每个式子，被乘数的变化规律是从 1 到行号，即 $[1,n]$ ，乘数都是行号，即通式为 $m \times n$ 。于是可写出如下的循环嵌套程序代码：

```

01 for n in range(1, 10):                #控制行数
02     for m in range (1, n+1):          #控制每行式子的个数
03         print("{} × {} = {}".format(m, n, m*n), end=' ') #不换行
04     print()

```

在这段代码中，03 行代码一共执行了 $1+2+3+\dots+9=45$ 次，你知道为什么吗？

4. break 和 continue 语句

break 意为打破、中断，continue 意为继续。在 Python 中这两个关键字都是用于改变循环体中语句执行顺序的。break 用于终止当前循环过程，continue 用于忽略本次循环体中后面的语句，直接开始下一次循环。举个例子：

学生在运动场进行长跑比赛的过程中，因为犯规直接退出比赛，就相当于使用了 break

的作用。当跑到 1 圈半的时候，因为裁判特许直接回到起点从第 2 圈继续开始，就相当于使用了 `continue` 的作用。

例子：输入一个英语句子，统计字母 n 的个数，当遇到空格时终止统计。

```
01 sum = 0
02 word = input('请输入英语单词: ')
03 for w in word:
04     if w=='n':
05         sum +=1
06         continue
07     elif w == ' ':
08         break
09 print(sum)
```

例子：请编写一个程序找出[3,100]中的所有质数（素数）。

分析：质数的定义是：在大于 1 的自然数中，除了 1 和它本身以外不再有其它因数的数。

最小的质数是 2。解决这个问题，我们可以划分为这样的步骤：

第一步：依次取出[3,100]中的每一个数，即遍历[3,100]。

第二步：对第一步中取出的每个数（n），都进行如下操作：

分别用 2,3,4, ……n-1 去除 n，如果都不能整除，则这个数是质数，只要有一个数能整除 n，则这个数就不是质数。

```
01 for n in range(3, 101):
02     for m in range(2, n):
03         if n % m ==0 :
04             break
05     else:
06         print(n)
```

知识拓展：将 02 行改为 `for m in range(2, int(math.sqrt(n)) + 1):`后也能正确求解，并且次数明显减少【`int(math.sqrt(n))`是对 n 的算术平方根取整，使用前需要使用 `import math` 语句导入 `math` 模块】。其理论根据是：如果一个数是合数，那么它的最小质因数一定小于或等于它的平方根。

3.7 正则表达式

在处理字符串时，经常需要判断字符串是否符合某种规则，或者从字符串中替换符合某种规则的子字符串以及提取子字符串。正则表达式就是描述这个规则的工具。正则表达式本身也是一个字符串，只是字符串里面的字符具有特殊意义。本节我们简单介绍正则表达式的编写规则和 `re` 库的基本使用。本节内容具有一定难度，如果阅读有些困难，可以暂时跳过。

学习编写正则表达式，主要就是学习一些字符表示的特殊意义，如“^”和“\$”分别表示字符串的开始和结束。下文中为了方便描述和实践，我们作如下规定：

- (1) 一律使用变量 `pattern` 来使用正则表达式；
- (2) 为了防止字符串本身的转义字符发生作用，在正则表达式前一律加 `r`；
- (3) 使用下面的代码片段测试与正则表达式的匹配情况。

```
01 import re
02 pattern = r"^abc"           #模式字符串
03 dst_str = "abcaaaaaaaaa"   #待检验的字符串
04 m = re.findall(pattern,dst_str)
05 if len(m) !=0:
06     print(m)
07 else:
08     print('不匹配')
```

3.7.1 正则表达式的编写规则

正则表达式中，主要由表示字符类型（匹配什么）、数量（匹配多少）、位置（在哪里匹配）等字符组成。在这些字符中，有些字符并不是指匹配自己，而是表示其它意义的字符。如“a\$”，不是表示字符串中需要有“\$”，而是表示字符串要以“a”结尾，我们把这样的字符叫做元字符。元字符有：`.` `^` `$` `*` `+` `?` `\` `|` `()` `[]` `{}` 11个。“\”叫做转义字符，后边跟元字符去除特殊功能，后边跟普通字符实现特殊功能。

1. 表示类型的字符

表 3.14 表示类型的字符

元字符	意义
<code>.</code>	匹配除换行符以外的任意字符
<code>\w</code>	匹配字母、数字、下划线、汉字
<code>\s</code>	匹配任何空白字符，包括空格、制表符、换页符等
<code>\d</code>	匹配数字字符
<code>\W</code>	匹配非字母、数字、下划线、汉字
<code>\S</code>	匹配任何非空白字符
<code>\D</code>	匹配非数字字符
<code>[]</code>	字符集合，匹配其中的任意一个字符

说明：如果这些元字符后面没有表示数量的元字符，则表示匹配 1 个。

`[]`字符集合，可匹配其中任意一个字符，除了 `^`、`-`、`]`、`\` 以外，其它字符都表示字符本身。“`^`”放在第 1 个位置时，表示否定，放在其它位置时就表示“`^`”本身；“-”放在中间位置表示“到”的意思，放在最前或最后时都表示“-”本身。“`]`”放在第 1 个位置时表

示“]”本身，其它位置表示与前面“[”配对的终止符号，“\”转义字符。

2. 表示数量的字符

表 3.15 表示数量的字符

元字符	意义
*	匹配任意数量
?	匹配 0 次或 1 次
+	匹配 1 次或多次
{n}	匹配 n 次
{n, }	匹配最少 n 次
{n, m}	匹配最少 n 次，最多 m 次

说明：表示数量的元字符需要跟在表示类型的元字符的后面。

3. 表示位置的字符

表 3.16 表示位置的字符

元字符	意义
^	匹配字符串的起始位置，在 [] 中表示否定
\$	匹配字符串的结束位置
\b	匹配单词边界
\B	匹配非单词边界

例子：编写满足“第 1 个字符为字母，后面紧跟 1-3 个数字，以 OK 结尾的字符串”的正则表达式。

```
pattern = r"[a-zA-Z]d{1,3}OK$"
```

"a25OK"会匹配成功，"a256fgfdgdfg5OK"会匹配失败。

4. 分组，小括号 ()

对单个字符进行重复，可以在字符后面跟上表示位置的字符即可，但如果要对多个字符进行重复怎么办呢？此时就要用到分组，可以使用小括号 () 来指定要重复的子表达式，然后对这个子表达式进行重复，例如：(abc)? 表示 0 个或 1 个 abc 这里一个括号里的表达式就是一个分组。

再如，要从一个类似于“010-88888888”的固定电话号码字符串中提取区号和电话号码。同样需要分组，可以使用 r'(\d{3,4})-(\d{8})' 得到结果。

() 的功能比较复杂，这里只介绍分组中几个较为简单的功能。

1. 捕获组：(...)，语法格式如下：

(rule)，匹配 rule 并捕获匹配结果，自动设置组号。这里的捕获是指仅提取与 rule 匹配的内容，而不提取 () 外面的内容。例子：

```
pattern = r' (\d{3,4})-(\d{8})'          #有两个分组
```

```
dst_str = "010-88888888"
```

将提取出：010、88888888，但不提取“-”。

2. 无捕获组：(?: …)，语法格式如下：

(?: rule)，匹配 rule 但不捕获匹配结果，这里的不捕获是指不单独提取与 rule 匹配的内容，而是要与 () 外面的内容一起提取。例子：

```
pattern = r' (?:\d{3,4})-(?:\d{8})'      #有两个分组
```

```
dst_str = "010-88888888"
```

将提取出：010-88888888。

也许你会发现 r' (?:\d{3,4})-(?:\d{8})' 与 r' \d{3,4}-\d{8}' 得到的结果是一样的，(?: rule) 究竟有什么作用呢？如果需要匹配形如：999.999.999.999 这样的字符串，就可以使用 r' (?:\d{3}\.){3}\d{3}'，如果不用分组，就需要使用 r' \d{3}\.\d{3}\.\d{3}\.\d{3}'，这就是他们的区别。

3. 前向界定：(?<=…)、前向否定界定：(?<!=…)、后向界定：(?=…)、后向否定界定：(?!=…)

这些界定符号的作用是表达需要匹配的内容“在（不在）什么之后，在（不在）什么之前”这种判断。

例子：写出“在<h1>之后且在</h1>之前”的正则表达式

```
pattern = r' (?<=<h1>).*?(?=</h1>)'
```

```
dst_str = r' <html><title><h1>HTML 页面</h1></title>'
```

将提取出：HTML 页面

注意：前向界定括号中的表达式必须是常值，即不能在前向界定的括号里写正则表达式。

5. 或规则 “|”

当在正则表达式中使用“|”时，例如：A|B，表示只要满足 A 或满足 B 就可以匹配。A、B 的范围是它两边的整条规则，如果想限定它的范围，必需使用一个无捕获组(?:)包起来。如：

pattern = r" I an a student | teacher"，则 A = " I an a student"，B = " teacher"，若修改为：

```
pattern = r" I an a (?:student | teacher)"，则 A = " student"，B = " teacher"。
```

6. 匹配模式：贪婪匹配与非贪婪匹配

贪婪匹配：在满足匹配时，匹配尽可能长的字符串，非贪婪匹配：在满足匹配时，匹配尽可能短的字符串，只要能匹配即可。默认情况下，采用贪婪匹配模式。

如果需要修改为非贪婪匹配模式，只要在表示数量的字符的后面加上“?”即可。如：

```
*? 、 +? 、 ?? 、 {n,m}? 、 {n,}? 
```

说明：字符串中的转义字符与正则表达式中的转义字符没有任何关系。如果你使用一个

未加 `r` 的字符串作为正则表达式, 那么在实际匹配时会首先进行字符串转义然后再进行正则表达式转义。

动手实践: 请编写描述下列规则的正则表达式。

1. 由大小写字母、数字组成, 且长度在[8,20]之间。
2. `yyyy-yy-yy`, 其中 `y` 表示数字
3. 检验手机号: 以 13、15、18 开头的手机号
4. 从字符串中提取包含在 `XXX` 之间的子字符串 `XXX`

参考答案:

1. `pattern = r'^[a-zA-Z0-9]{8,20}$'`
2. `pattern = r'^(?:\d{4})-(?:\d{2})-(?:\d{2})$'`
3. `pattern = r'^(?:13|15|18)\d{9}$'`
4. `pattern = r'(?<=). * ? (? = < / ul >)'`

本节我们只是简单介绍了编写正则表达式的一些规则, 更深入的内容请你参考相关资源。

3.7.2 re 库的基本使用

`re` 库是 Python 的标准库, 主要用于字符串匹配和替换。在使用前需要用 `import re` 导入。

Re 库主要功能函数

函数名称	说明
<code>compile</code>	将正则表达式编译成一个正则表达式对象
<code>findall</code>	搜索字符串, 以列表形式返回全部能匹配的子串
<code>search</code>	从一个字符串中搜索匹配正则表达式的第一个匹配, 返回 <code>match</code> 对象
<code>match</code>	从一个字符串的开始位置起匹配正则表达式, 返回 <code>match</code> 对象
<code>sub</code>	替换字符串中的匹配项, 返回替换后的字符串
<code>subn</code>	替换字符串中的匹配项, 返回一个元组, 存放替换结果和替换次数
<code>split</code>	将一个字符串按照正则表达式匹配结果进行分割, 返回列表
<code>finditer</code>	搜索字符串, 返回一个匹配结果的迭代类型, 每个元素是 <code>match</code> 对象

1. `compile` 函数

功能: 对正则表达式进行编译, 返回正则表达式对象。对正则表达式先编译, 可以大幅提高匹配速度。

语法格式: `re.compile(pattern, [flags])`

参数: `pattern`, 正则表达式, 参数 `flag`, 可选参数, 指定匹配模式, 取值如表 3.18 所示。

表 3.18 修饰符

修饰符	描述
re.I	使匹配对大小写不敏感
re.L	做本地化识别 (locale-aware) 匹配
re.M	多行匹配, 影响 ^ 和 \$
re.S	使 . 匹配包括换行在内的所有字符
re.U	根据 Unicode 字符集解析字符。这个标志影响 \w, \W, \b, \B.
re.X	正则表达式可以是多行, 忽略空白字符, 并可以加入注释。主要是为了让正则表达式更易读

后面 re 库函数中 flags 参数的意义同上。

例:

```
>>>str = 'made in china'
>>> pattern = 'china'
>>>a=re.compile(pattern)
>>>a.findall(str)           #匹配 str 中的字符 'china'
[' china']                 #匹配到 'china', 返回一个列表
```

说明: Re 库的两种使用方法:

- ①函数式: 直接使用形如 re.函数名(pattern, dst_str)使用;
- ②对象式: 首先把 pattern 使用 compile(pattern)函数转变为正则表达式对象后, 再使用: 对象.方法名(dst_str)使用。这里的方法名与 1 中的函数名完全相同。

本书中, 我们以函数式进行介绍。

2. findall 函数

功能: 返回字符串 dst_str 中匹配 pattern 格式的所有子串, 以列表形式返回, 如果没有找到匹配的, 则返回空列表。

语法格式: re.findall(pattern, dst_str, flags=0)

例子:

```
>>>dst_str = 'piy poy pky piy pry psy'   #定义字符串变量 dst_str
>>>re.findall('piy', dst_str)           #用 findall 函数在 dst_str 中匹配字符串
piy'
[' piy', ' piy']                       #匹配出两个字串 ' piy', 作为列表元素输出
```

注意: 当正则表达式中有 () 时, 其输出的内容有多种变化。

3. search 函数

功能: 扫描整个字符串并返回第一个匹配的值, 匹配成功返回 Match 对象 否则返回 None。

语法格式：`re.search(pattern, dst_str, [flags])`

例子：

```
>>> dst_str = 'www.baidu.com'
>>> r=re.search('com', dst_str)
>>> r.group()           #用 Match 对象的 group() 方法返回匹配的字符串
'com'
>>> r.span()           #用 Match 对象的 span() 方法返回匹配的位置(开始, 结
束)
(10, 13)               #span() 函数返回一个位置元组
```

4. match 函数

功能：从字符串的开始位置进行匹配，匹配成功返回 Match 对象 否则返回 None。

语法格式：`re.match(pattern, dst_str, [flags])`

例子：

```
>>> dst_str = 'www.baidu.com'
>>> r=re.match('www', dst_str)
>>> r.group()
'www'
```

`re.match` 与 `re.search` 的区别：`re.match` 从字符串的开始位置进行匹配，如果字符串开始不符合正则表达式，则匹配失败，函数返回 None；而 `re.search` 匹配整个字符串，直到找到一个匹配。

表 3.19 所示为 `match` 对象的方法。

表 3.19 `match` 对象的方法

方法	描述
<code>group()</code>	返回被 re 匹配的字符串
<code>span()</code>	返回一个元组包含匹配的位置(开始, 结束)
<code>start()</code>	返回匹配开始的位置
<code>end()</code>	返回匹配结束的位置

5. sub 函数

功能：替换字符串中的匹配项。

语法格式：`re.sub(pattern, repl, dst_str, [count], [flags])`

参数：

pattern : 正则中的模式字符串。

repl : 替换的字符串, 也可为一个函数。

dst_str : 要被查找替换的原始字符串。

count : 可选参数, 模式匹配后替换的最大次数, 默认值为 0, 表示替换所有的匹配。

例子:

```
>>> dst_str = '400-888-345-789'
>>> t=re.sub('0', '2', dst_str)
>>> print(t)                #输出 422-888-345-789
>>> r=re.sub('\D', '', dst_str) #用空字符替换 dst_str 中的非数字字符 '-'
>>> print(r)                #输出 400888345789
>>> p=re.sub('8', '6', dst_str, count=2) #count=2 表示只替换 2 次
>>> print(p)                #输出 400-668-345-789
```

6. subn 函数

功能: 替换字符串中的匹配项, 返回一个元组, 存放替换结果和替换次数

语法格式: re.subn(pattern, repl, dst_str, [count], [flags])

参数:

pattern : 正则中的模式字符串。

repl : 替换的字符串, 也可为一个函数。

dst_str : 要被查找替换的原始字符串。

count : 可选参数, 模式匹配后替换的最大次数, 默认值为 0, 表示替换所有的匹配。

例子:

```
>>> r='I like you I like you'
>>> re.subn('you', 'him', r) #将变量 r 中的 'you' 替换为 'him'
('I like him I like him', 2) #返回一个元组, '2' 表示替换了 2 次
```

7. split 函数

功能: 将一个字符串按照正则表达式匹配结果进行分割, 返回列表

语法格式: re.split(pattern, dst_str, [maxsplit], [flags])

参数:

pattern : 正则中的模式字符串。

dst_str : 要被查找替换的原始字符串。

maxsplit: 可选参数, 最大的分割次数, 默认全部分割, 剩余部分作为最后一个元素。

例子:

```
>>>re.split(r'[1-9]\d{5}', 'BIT100081 TSU100084') #输出['BIT', ' TSU',
'']
```

```
>>>re.split(r'[1-9]\d{5}', 'BIT100081 TSU100084', maxsplit=1) #输出
['BIT', ' TSU100084']
```

技巧:如果使用带括号的正则表达式,则可以将正则表达式匹配的内容也添加到列表内。

```
>>> re.split(r'([1-9]\d{5})', 'BIT100081 TSU100084')
```

```
输出: ['BIT', '100081', ' TSU', '100084', '']
```

8. finditer 函数

功能: 搜索字符串, 返回一个匹配结果的迭代类型, 每个元素是 match 对象

语法格式: `re.finditer(pattern, dst_str, [flags])`

参数:

`pattern` : 正则中的模式字符串。

`dst_str` : 待匹配的字符串。

例子:

```
01 import re
02 for m in re.finditer(r"[0-9]\d{5}", "HHU211100 HHU211000"):
03     if m:
04         print(m.group(0))
```

输出:

```
211100
```

```
211000
```

本章小结

本章详细介绍了 Python 的基本语法规则, 主要包括代码缩进、注释、各种运算符、表达式、字符串及正则表达式、程序流程控制语句。这些是 Python 的基础内容, 需要重点掌握, 为后续内容的学习打下良好的基础。在教学实践中, 我们发现许多同学不熟悉键盘、不能透彻理解计算机执行代码的逻辑, 能用中文表达出解决问题的步骤, 但不能书写出程序代码。为此我们列出了许多具有完整功能的代码实例, 希望同学们不要仅停留在阅读理解的层面, 一定要动手实践这些代码, 并仔细体会计算机执行这些代码的逻辑顺序以及解决问题的方法。当然在实践过程中, 一定会遇到很多错误, 排除一次又一次错误的过程便是你成长的过程。内容顺序是按照我们在实际教学过程中总结得出的, 易于理解和上机实践操作进行安排的。

练习题

1. 不同的运算符具有不同的优先级, 优先级高的运算先执行, 优先级低的运算后执行, 同一优先级的运算按从左到右顺序进行。实际上往往难以记住这些运算符的优先级别, 通常

是采用 () 来改变运算次序。请在 IDLE 的交互模式执行下面的语句并分析结果填空。

```
>>>-2*3**4
```

```
>>>-6**2
```

```
>>>6 and 3
```

```
>>>6 or 3
```

```
>>>5>4 and 6<8
```

```
>>>not 3 and 8<7
```

(1) - 的优先级 _____ (高于/低于) ** 的优先级。

(2) 在表达式 $a \text{ and } b$ 中, 如果 a 为 True, 则表达式的值为 _____, 如果 a 为 False, 则表达式的值为 _____; 在表达式 $a \text{ or } b$ 中, 如果 a 为 True, 则表达式的值为 _____, 如果 a 为 False, 则表达式的值为 _____;

(3) and 的优先级 _____ (高于/低于) 比较运算符。 not 的优先级 _____ (高于/低于) and 的优先级。

2. 由键盘输入一个非负整数, 判断这个非负整数有多少位数, 如 99, 输出 2, 1024, 输出 4。

3. 分别计算整数 10 到 1000(包括 1 和 1000)之间的所有奇数的和、所有偶数的和。

4. 由键盘输入 n 个学生的成绩, 找出最高分, 分数之间用空格隔开。

5. 角谷猜想, 是指对于任意一个正整数, 如果是奇数, 则乘 3 加 1, 如果是偶数, 则除以 2, 得到的结果再按照上述规则重复处理, 最终总能够得到 1。如, 假定初始整数为 5, 计算过程分别为 16、8、4、2、1。程序要求输入一个整数, 将经过处理得到 1 的过程输出出来。

输入样例: 5

输出样例:

$5*3+1=16$

$16//2=8$

$8//2=4$

$4//2=2$

$2//2=1$

End

6. 国王将金币作为工资, 发放给忠诚的骑士。第 1 天, 骑士收到一枚金币; 之后两天(第 2 天和第 3 天)里, 每天收到两枚金币; 之后三天(第 4、5、6 天)里, 每天收到三枚金币; 之后四天(第 7、8、9、10 天)里, 每天收到四枚金币……这种工资发放模式会一直这样延续下去: 当连续 n 天每天收到 n 枚金币后, 骑士会在之后的连续 $n+1$ 天里, 每天收到 $n+1$ 枚金币(n 为任意正整数)。你需要编写一个程序, 确定从第一天开始的给定天数内, 骑士一共获

得了多少金币。

输入样例：6

输出样例：14

7.编写程序，打印出下面的图形：

```
*  
**  
***  
****  
*****  
*****  
****  
***  
**  
*
```