第二章 面向过程编程

2.1 变量和常量

程序就是用来处理数据的，而变量就是用来存储数据的，对应于计算机内存中的一块区域。变量通过唯一的标识符（即变量名）来表示，并且可以通过各种运算符对变量的值进行操作。

2.1.1 变量的定义

Python是一种动态类型的编程语言，不需要显示的声明变量的数据类型，可以直接对变量进行赋值，变量赋值以后该变量才会被创建。

赋值语句的一般语法格式如下：

```
变量名 = 表达式
```

等号(=)为赋值运算符，用来给变量赋值：

- 左边为变量名
- 右边为表达式

2.1.2 整数池

pythobn中，所有数据类型都是对象，即使是值类型，变量中存放的也不是数据值，而是值类型对象的内存地址。这种通过内存地址间接访问数据对象的方式称为引用。

通过变量之间的赋值可以使两个变量引用相同的对象，而使用身份运算符is则可以判定两个变量是否引用同一个对象

```
In [1]: a = 3

In [2]: b = a  # 将a中的内存地址赋值给b

In [3]: print(a is b) # 判断是否引用了同一个对象
True
```

实际上，Python为了优化速度，使用了小整数对象池， 避免为整数频繁申请和销毁内存空间。[-5, 256] 这些整数对象是提前建立好的，不会被垃圾回收。所有位于这个范围内的整数使用的都是同一个对象

```
In [1]: a = 3

In [2]: b = 3

In [3]: print(a is b)
True
```

而对于其他的值类型，即使值相同，也不是引用的同一个对象：

```
In [6]: d = 3.1

In [7]: e = 3.1

In [8]: print(d is e)
False
```

对于其他的基本数据类型，只要值相同，它们引用的也是同一个对象：

```
In [1]: str_a = "abc"

In [2]: str_b = "abc"

In [3]: print(str_a is str_b)
True

In [4]: a = True

In [5]: b = True

In [6]: print(a is b)
True

In [7]: c = None

In [8]: d = None

In [9]: print(c is d)
True
```

基本数据类型的池中的值是不可变的，也就是说当修改数据的值时，会将修改后的值所在的内存地址赋值给变量，而不会影响原来的值：

```
In [16]: a = 3

In [17]: print(id(a)) # id函数可以获取变量引用的内存地址
4492330400

In [18]: a += 1

In [19]: print(id(a))
4492330432
```

2.1.3 常量

常量是指首次赋值后保持固定不变的值。在Python中没有常量机制，通常用一个不改变值的变量来表示常量，比如用下面的方式表示一个常量π：

```
PI = 3.14
```

2.1.4 变量命名规则

(1). 标识符

标示符就是程序员定义的 变量名、函数名。标识符的约束规则：

- 标示符可以由 字母、下划线和数字组成
- 不能以数字开头
- 不能与关键字重名

思考：下面的标示符哪些是正确的，哪些不正确，为什么？

```
fromNo12
from#12
my_Boolean
my-Boolean
Obj2
2ndObj
myInt
My_tExt
_test
test!32
haha(da)tt
jack_rose
jack&rose
GUI
G.U.I
```

(2). 关键字

关键字就是在Python内部已经使用的标识符。关键字具有特殊的功能和含义，开发者 不允许定义和关键字相同的名字的标示符 通过以下命令可以查看Python中的关键字：

```
In [29]: import keyword

In [30]: print(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']
```

(3). 变量命名规则

命名规则可以被视为一种惯例，并无绝对与强制，目的是为了增加代码的识别和可读性。常见的规则如下：

- 标识符区分大小写。Hero 与hero是不一样的
- 在定义变量时，为了保证代码格式，=号的左右应该各留一个空格
- 如果变量名需要由二个或多个单词组成时，可以按照以下方式命名
    - 每个单词都使用小写字母。例如，firstname
    - 单词与单词之间使用下划线连接。例如，first_name
    - 小驼峰式命名法，即驼峰命名。如，firstName
    - 大驼峰命名法，即帕斯卡（Pascal）命名如，FirstName

(4). 命名规范建议

- **模块**： 模块尽量使用小写命名，首字母保持小写，尽量不要用下划线(除非多个单词，且数量不多的情况)，并且将相关的类和顶级函数放在同一个模块里. 不像Java, 没必要限制一个类一个模块.

    ```
    # 正确的模块名
    import decoder
    import html_parser

    # 不推荐的模块名
    import Decoder
    ```

- **类名**：类名使用帕斯卡命名风格，首字母大写，私有类可用一个下划线开头

```
class Farm():
    pass

class AnimalFarm(Farm):
    pass

class _PrivateFarm(Farm):
    pass
```

- **函数**：函数名一律小写，如有多个单词，用下划线隔开

```
class Person():
    def run():
    pass

    def run_with_env():
    pass
    def _private_func(): # 私有函数在函数前加一个或两个下划线
        pass
```

- **变量名**：变量名尽量小写, 如有多个单词，用下划线隔开

```
if __name__ == '__main__':
    count = 0
    school_name = '
```

- **常量**：常量采用全大写，如有多个单词，使用下划线隔开

```
MAX_CLIENT = 100
MAX_CONNECTION = 1000
CONNECTION_TIMEOUT = 600
```

- **引号**:

  简单说，自然语言使用双引号，机器标示使用单引号，因此 **代码里** 多数应该使用 **单引号**

  - **自然语言 使用双引号** `"..."`
    例如错误信息；很多情况还是 unicode，使用`u"你好世界"`
  - **机器标识 使用单引号** `'...'`
    例如 dict 里的 key
  - **正则表达式 使用原生的双引号** `r"..."`
  - **文档字符串 (docstring) 使用三个双引号** `"""......"""`

2.2 基本数据类型

数据类型在数据结构中的定义是一个值的集合以及定义在这个值集上的一组操作。在python中可以通过type()函数查看数据类型：

```
a =3
print(type(a)) # <class 'int'>
```

2.2.1 数值类型

数值类型用于表示数据，数值类型的数据可以参与算术运算。数值类型包括了整型、浮点型和复数型

(1) 整型(int)

整型数据即整数，没有小数部分，但可以有正负号。 在Python内部对整数的处理分为普通整数和长整数，普通整数长度为机器位长，通常都是32位，超过这个范围的整数就自动当长整数处理，而长整数在计算机内的表示不是固定长度的，在内存容量允许的情况下，长整数的取值范围几乎没有限制，这为大数据处理带来了便利。

```
 a = 3
 b = -3
```

在python中，整数常量可以用十进制、二进制、八进制和十六进制来表示：

```
In [25]: d2  = 0b1010 # 0b或0B做为前缀，表示后面的值为二进制

In [26]: print(d2)
10

In [27]: d8 = 0o12 # 0o或0O做为前缀,表示后面的值为八进制

In [28]: print(d8)
10

In [29]: d10 = 10 # 数据没有前缀，默认为十进制

In [30]: print(d10)
10

In [34]: d16 = 0xA # 0x或0X做为前缀，表示后面的数据为十六进制

In [35]: print(d16)
10
```

(2) 浮点型（float）

浮点数表示一个实数。对于浮点数，Python3.x默认提供17位有效数字的精度。 浮点型数据有两种表示方式:

- 十进制小数形式
- 指数形式

```
In [36]: f1 = 3.14159265

In [37]: f2 = 3.14e8  # 3.14 × 10^8
```

(3) 复数型(complex)

复数的表示形式为a+bj, 其中a为复数的实部，b为复数的虚部，j为虚数单位（表示-1的平方根），字母j也可以写成大写形式J。

```
In [40]: complex =3+4j

In [42]: print(complex.real) # real属性获取实数部分
3.0

In [43]: print(complex.imag) # imags属性获取虚数部分
4.0
```

2.2.2 字符串类型

字符串是使用单引号、双引号和三引号（三个单引号或三个双引号）扩起来的任意文本

```
name1 = '我是python' # 单引号

name2 = "我是python" # 双引号

name3 = """我是python""" # 三引号

name4 = '''我是python''' # 三引号
```

单引号和双引号括起来的字符只能是单行的，如果代码中显示为多行需要添加上反斜杠()：

```
name = '我' \
       "是" \
       'python'
print(name) # 我是python
```

三引号常用于定义文档字符串。类似于html中的pre标签的效果，可以将多行文字以及空格原样输出。

```
code = '''
    int main(){
        #代码
    }
'''

'''
输出结果:
    int main(){
        #代码
    }
'''
```

2.2.3 布尔类型

布尔类型数据用于描述逻辑判断的结果。布尔类型的数据只有两个值：

- 逻辑真：True
- 逻辑假：False

```
In [1]:  a = 3>2

In [2]: print(a)
True
```

2.2.4 空值

在python中，空值用None来表示

```
In [3]: a = None

In [4]: print(type(a))
<class 'NoneType'>
```

2.3 运算符

2.3.1 算术运算符

算法运算符可以用于对操作数进行算术运算，其运算结果是数值类型

运算符	描述	实例
+	加或单目运算中的正号	10 + 20 = 30
-	减或单目运算中的负号	10 - 20 = -10
*	乘	10 * 20 = 200
/	除	10 / 20 = 0.5
//	取整除	返回除法的整数部分（商） 9 // 2 输出结果 4
%	取余数	返回除法的余数 9 % 2 = 1
**	幂	又称次方、乘方，2 ** 3 = 8

2.3.2 关系运算符

关系运算符也叫比较运算符，用来比较两个操作数的大小，其运算结果是一个布尔值。关系运算符的操作数可以是数字或字符串。若操作数是字符串，系统会从左到右逐个比较每个字符的Unicode码，直到出现不同的字符或字符串为止。 | 运算符| 描述 | 实例 | | :---: | :---: | --- | |等于|23返回False；“abc”="ABC"返回False <|小于|2<5 返回True;"this"<"This"返回False

> |大于|2>5 返回False;"this">"This"返回True <=|小于等于| 2<=5返回True;"this"<="This"返回False =|大于等于|5>=5 返回True;"this">="This"返回True !=|不等于 |3!=5返回True;"this"!="This"返回True

2.3.3 赋值运算符

|运算符| 描述| 实例 | :---: | :---: | --- | |= |简单的赋值运算符| c = a + b 将 a + b 的运算结果赋值为 c| |+= |加法赋值运算符| c += a 等效于 c = c + a |-= |减法赋值运算符| c -= a 等效于 c = c - a |*= |乘法赋值运算符| c *= a 等效于 c = c * a |/= |除法赋值运算符| c /= a 等效于 c = c / a |%= |取模赋值运算符| c %= a 等效于 c = c % a |**=|幂赋值运算符| c** = a 等效于 c = c ** a |//=| 取整除赋值运算符| c //= a 等效于 c = c // a

2.3.4 逻辑运算符

逻辑运算符用于布尔值的运算，包括逻辑与、逻辑或、逻辑非。其中逻辑与、逻辑或是双目运算符、逻辑非是单目运算符

逻辑与的真值表：

	T	F
T	T	F
F	F	F

逻辑或的真值表：

	T	F
T	T	T
F	T	F

2.3.5 位运算符

位运算用于对数字的二进制位进行运算。

| 运算符| 描述 | 实例 | | :---: | --- | --- | << | 左移运算符（将左操作数的二进制数位全部左移若干（右操作数）位，高位丢弃球，低位补零）|2<<3 返回16。相当于 `$2^3$`

> | 右移运算符（将左操作数的二进制数位全部右移若干（右操作数）位，高位补0，低位丢弃）|16>>3返回2.相当于 `$2^{1/3}$` &|按位与运算符：参与运算的两个值,如果两个相应位都为1,则该位的结果为1,否则为0|22&3返回2 ||按位或运算符：只要对应的二个二进位有一个为1时，结果位就为1。|32|3返回35 |按位异或运算符：当两对应的二进位相异时，结果为1|18^6返回20 ~|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 。~x 类似于 -x-1|~32返回-33

2.3.6 成员运算符

成员运算符用于判定对象是否存在于字符串等序列中。 |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。|"y" in "python" 返回True |not in| 如果在指定的序列中没有找到值返回 True，否则返回 False。|"y" not in "python" 返回False

2.3.7 身份运算符

身份运算符用于比较两个对象的内存地址是否相同。

|运算符| 描述| 说明 | | :---: | --- | --- |---| |is| is 是判断两个标识符是不是引用自一个对象 |x is y, 类似 id(x) == id(y) , 如果引用的是同一个对象则返回 True，否则返回 False| x=1;y=x; |is not| is not 是判断两个标识符是不是引用自不同对象 |x is not y ， 类似 id(a) != id(b)。如果引用的不是同一个对象则返回结果 True，否则返回 False。

例如：

```
a = 20
b = 20
print(a is b) # True
b = 30
print(a is b) # False
print(a is not b) # True
```

2.3.8 表达式与优先级

表达式是运算符和操作数组成的有意义的组合，它可以是常量、变量，也可以是函数的返回值。通过运算符对表达式中的值进行若干次运算，最终得到表达式的返回值。

按照运算符的种类，可以将表达式分为算术表达式、关系表达式、逻辑表达式等。多种运算符混合运算可形成复合表达式，此时系统会按照运算符的优先级和结合性依次进行运算。如果需要，用户可以使用圆括号来改变运算顺序。

常见的运算符按照优先级从高到低的顺序排列如下：

优先级（从高到底）	运算符
1	索引([])
2	幂(**)
3	单目(+、-)
4	*，/， //，%
5	加减（+，-）
6	关系（>，>=，<，<=，!=）is，身份运算符(is not)
7	赋值（=，+=，-=，/=，//=，%=,**=）
8	逻辑非(not)
9	逻辑与(and)
10	逻辑或（or）

2.4.数值类型的转换

2.4.1 其他数据类型转换为整数

```
In [1]: integer = int(3.14) # 浮点型转换为整型

In [2]: integer2 = int("3",10) # 字符串转换为整型.字符串中有数字(0-9)和正负号(+/-)以外的字符，就会报错。

In [3]: integer3 = int(True) # 布尔型转换为字符串

In [4]: print(integer)
3

In [5]: print(integer2)
3

In [6]: print(integer3)
```

```
1
```

> 注意：空值是无法转换为整数类型的。

2.4.2 其他数据类型转换为浮点型

```
In [9]: float1 = float(3) # int 转换为 float 时，会自动给添加一位小数。

In [11]: float2= float('-3.14') # 如果字符串含有正负号(+/-)、数字(0-9)和小数点(.)以外的字符，则不支持转换。

In [12]: float3 = float(True)

In [17]: print(float1)
3.0

In [18]: print(float2)
-3.14

In [19]: print(float3)
1.0
```

2.4.3 任意对象转换为字符串

1. 基本数据类转换为字符串

```
In [20]: str1 = str(3) # int 转换 str 会直接完全转换

In [21]: str2 = str(-12.00) #loat 转换 str 会去除末位为 0 的小数部分。

In [22]: str3 = str(True) # 转换为字符串

In [24]: str4 = str(complex(12+9j)) #先将值转化为标准的 complex 表达式，然后再转换为字符串。

In [25]: print(str1)
3

In [26]: print(str2)
-12.0

In [27]: print(str3)
True

In [28]: print(str4)
```

```
(12+9j)
```

2.5 流程控制

计算机程序主要由数据结构和算法两个要素组成。数据结构即数据的存储形式，算法则是对操作步骤的描述。任何简单或复杂的算法都可以由顺序结构、选择结构和循环结构组合而成。通过这三种结构就可以实现程序的流程控制。

2.5.1 顺序结构

程序的工作流程一般分为输入数据、处理数据、输出结果。顺序结构是一种最简单的流程控制结构，其特点是程序中的各个操作是按照它们在源代码中的排列顺序执行的。

(1). 数据输入

为了让用户通过程序与计算机进行交互，程序通常应具有数据的输入\输出功能。在Python程序中，可以使用键盘输入数据，也可以从文件中或数据库中读取数据，然后由程序对输入的数据进行处理，处理结果可以输出到屏幕上，也可以保存到文件或数据库中。

实际应用中，最常见的情形是从键盘输入数据并通过屏幕输出数据，这是标准的控制台输入\输出模式。

在Python中，标准输入通过内置函数input实现，其格式如下：

```
variable = input("提示字符串")
```

例如，输入姓名，并用变量捕获：

```
In [31]: name = input('请输入姓名:')
请输入姓名:hao

In [32]: print(name)
hao
```

(2). 数据的处理

为了对输入的数据进行处理，首先需要将数据保存到变量所引用的内存中，这个要通过赋值语句来实现。

在Python中，赋值语句分为：

- 简单赋值语句
- 复合赋值语句
- 多变量赋值语句

1) 简单赋值语句

简单赋值语句用于对单个变量的赋值，其一般格式为：

```
变量 = 表达式
```

例如：

```
name = input('请输入姓名:')
```

当键盘中输入字符，回车时，会将输入的字符以字符串的形式保存内name变量中。

2) 复合赋值语句

复合赋值语句是利用复合赋值运算符对变量当前值进行某种运算后执行赋值操作的。变量既是运算对象又是赋值对象。

例如，求输入值的平方：

```
In [34]: num *= float(input('输入值: '))
输入值: 12

In [35]: num  # 在交互模式下，直接使用变量即可以输出值
Out[35]: 144.0
```

3) 多变量赋值语句

在python中，可以使用赋值语句的变化形式对多个变量进行赋值。赋值语句有两种变化形式：

- 链式赋值语句。用于对多个变量赋予同一个值，语法格式：
  - 变量1=变量2=...=变量n = 表达式
- 同步赋值语句。使用不同表达式的值分别对不同变量赋值,语法格式：
  - 变量1,变量2,...,变量n = 表达式1，表达式2,...,表达式n

链式赋值案例：

```
In [36]: x = y = z = 123

In [37]: x
Out[37]: 123

In [38]: y
Out[38]: 123

In [39]: z
Out[39]: 123
```

同步赋值语句案例：

```
In [41]: x,y,z = 3,4,5

In [42]: x
Out[42]: 3

In [43]: y
Out[43]: 4

In [44]: z
Out[44]: 5
```

4) 数据的处理

获取数据后，就可以进行下一步的处理了。我们可以对数据进行运算、排序、查找等各种操作，这里不再赘述。

(3). 数据的输出

1) 标准输出

标准输出可以通过两种方式实现：

- 在交互模式下，使用表达式语句来输出表达式的值。
- 在脚本模式下，使用print函数输出

print函数的语法结构：

```
print(output1[,output2,...][,sep=分隔符][,end=结束符])
```

例如：

```
In [51]: name,age = 'zs',18

In [52]: print(name,age,sep='-',end=';')
zs-18;
```

2) 格式化输出

使用字符串格式化运算符%可以将输出格式化，然后调用print函数，按照一定格式输出数据，具体调用格式如下：

```
print(格式化字符串%(输出项))
```

其中格式化字符串由普通字符串和格式说明符组成。普通字符按原样输出，格式说明符则用于指定对应输出项的输出格式。

常用的格式说明符：

格式说明符	含义
%%	输出百分号
%d	输出十进制整数
%c	输出字符
%s	输出字符串
%O	输出八进制整数
%x或%X	输出十六进制整数
%e或%E	以科学计数法输出浮点数
%[w][.p]f	以小数形式输出浮点数，数据长度为w，默认为0；小数部分有p为，默认有6位。

例如：

```
In [53]: print("姓名：%s,年龄：%d"%('zs',18))
姓名：zs,年龄：18
```

还可以用格式化辅助指令：

符号	功能
m.n	m是显示的最小总宽度，n是小数位数
-	用于左对齐
+	在正数前面显示加号
0	显示的数字前面填充'0'取代空格
#	在输出的八进制数前添加0o，在输出的十六进制前添加0x

例如：

```
In [54]: print('%10.4f'%12.34) #要求最终返回的数字的宽度是10位，小数后4位小数。默认右对齐
    12.3400

In [55]: print('%-10.4f'%12.34) # 左对齐
12.3400
```

```
In [57]: print('%-+10.4f'%12.34) # 显示正号
+12.3400

In [61]: print('%010.4f'%12.34) # 右对齐，左边以0填充
00012.3400
In [62]: print("%#x"%11) # 在十六进制数前加0x
0xb
In [63]: print("%#o"%11) # 在八进制数前加0o
0o13
```

除了print提供的格式化输出外，还可以使用str.format函数实现字符串的格式化。语法结构如下：

```
str.format(输出项)
```

其中，str（格式化字符串）由普通字符与格式说明符组成。普通字符按原样输出，格式说明符则用于指定对应输出项的输出格式。格式说明符用花括号括起来，其一般形式如下：

```
{[序号或键名]:格式控制符}
```

> 序号：可选性。用于指定要格式化的输出项的位置，0表示第一项。若序号全部省略，则按自然顺序输出。\ 键名：可选性。一个标识符，对应于输出项的名字或字典的键值\

常见的格式控制符：

符号	说明
d	输出十进制整数
b	输出二进制整数
o	输出八进制整数
x	输出十六进制整数
c	输出整数对应的ascii字符
f	输出浮点数
e	输出科学计数法形式的浮点数
%	输出百分号

```
In [64]: print("{0:c}".format(97)) # 输出数字对应的ascii字符
a

In [65]: print("{0:d},{1:b},{2:o},{3:x}".format(10,10,10,10)) # 以不同进制输出
10
10,1010,12,a
In [67]: print("{:d},{:b},{:o},{:x}".format(10,10,10,10)) # 省略序号
10,1010,12,a
In [68]: print("{0:#d},{1:#b},{2:#o},{3:#x}".format(10,10,10,10)) # 可以使用
格式化辅助指令
10,0b1010,0o12,0xa

In [74]: print("{:08.4f}".format(3.14)) # 输出浮点数
003.1400
```

综合案例：编写一个简单的成绩录入程序

```
In [97]: name,gender,classes = eval(input("请输入姓名，性别与班级"))
请输入姓名，性别与班级"zs","male",18

In [98]: chinese,math,eng = eval(input("请输入语文，数学与英文成绩"))
请输入语文，数学与英文成绩89,80,76

In [100]: print("姓名：{:},性别：{:},班级：{:}".format(name,gender,classes))
姓名：zs,性别：male,班级：18

In [101]: print("语文：{:},数学：{:},英文：{:}".format(chinese,math,eng))
语文：8,数学：80,英文：76
```

2.5.2 选择结构

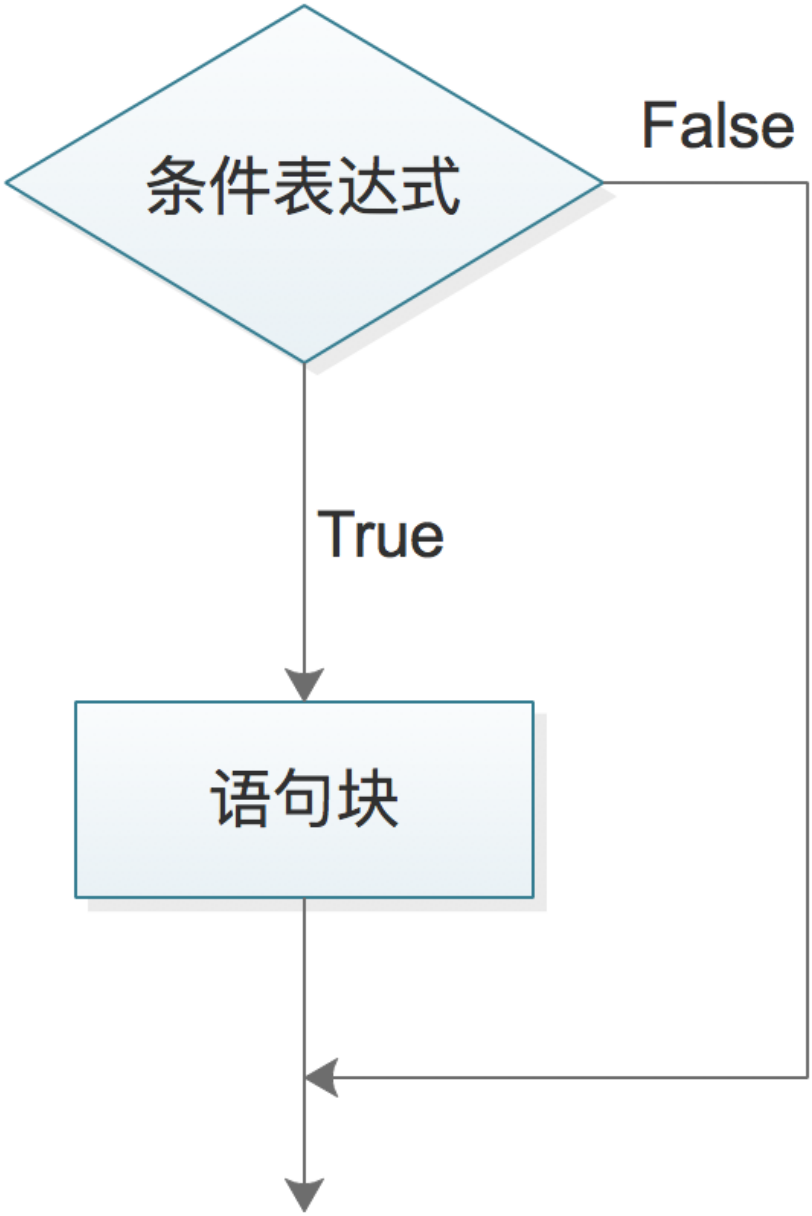
选择结构是指程序在运行时根据某个特定条件选择一个分支执行。根据分支的多少，分为选择结构分为单分支选择结构、双分支选择结构、多分支选择结构

(1). 单分支选择结构

用于处理单个条件、单个分支的情况。语法结构如下：

```
if 条件表达式 :
    语句块
```

if语句的执行流程如下图：



例题：判断是否闰年

> 闰年的必要条件：1. 能被4整除2。2.不能被100整除。世纪闰年中能被400整除

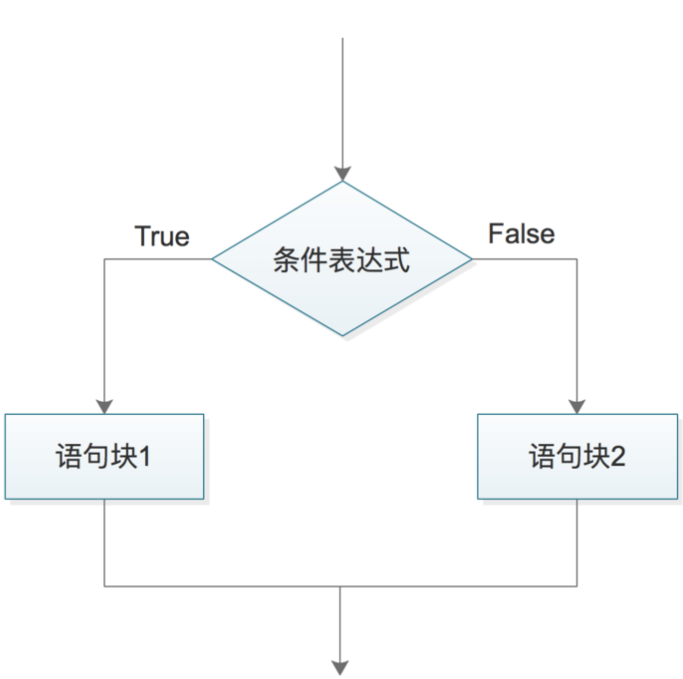
```
leapYear = '不是'

year = int(input('输入一个年份：'))

if year%4 == 0 and year%100 != 0 or year%400 == 0 :
    leapYear ="是"

print("{0}年{1}闰年。".format(year,leapYear))
```

(2). 双分支选择结构

用于处理单个条件、多个分支的情况。语法结构如下：

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

if-else语句的执行流程：



例题：从键盘输入三角形的三条边，计算三角形的面积

> 构成三角形的充要条件：任意两边之和大于第三边。

```
a,b,c = eval(input("输入三条边"))

if a+b>c and a+c>b and b+c>a:
    p=(a+b+c)/2 # 半周长
    area = (p*(p-a)*(p-b)*3,4,(p-c))**0.5 # 海伦公式
    print("当三角形的三条边为a={},b={},c={}时：".format(a,b,c))
    print("三角形的面积：area={}".format(area))
else:
    print("a={},b={},c={}不构成三角形".format(a,b,c))
```

例题：输入一个整数，判断它是不是水仙花数

> 水仙花数是自幂数的一种，严格来说3位数的3次幂数才称为水仙花数。它的每个位上的数字的3次幂之和等于它本身。

if... else 还能作为条件运算。条件运算是一个三目运算，它有三个运算对象，其一般格式如下：

```
表达式1 if 逻辑表达式 else 表达式2
```

> 首先计算逻辑表达式的值，如果为True，则计算表达式1的值，否则计算表达式2的值。

例如： 比较两个数的大小

```
a,b = eval(input("输入两个数。两个数用逗号分隔："))

result = "{}>{}".format(a,b) if a>b else "{}<{}".format(a,b) # 三目运算的结果赋值给result

print(result)
```

例题：判断输入的数字是否是水仙花数

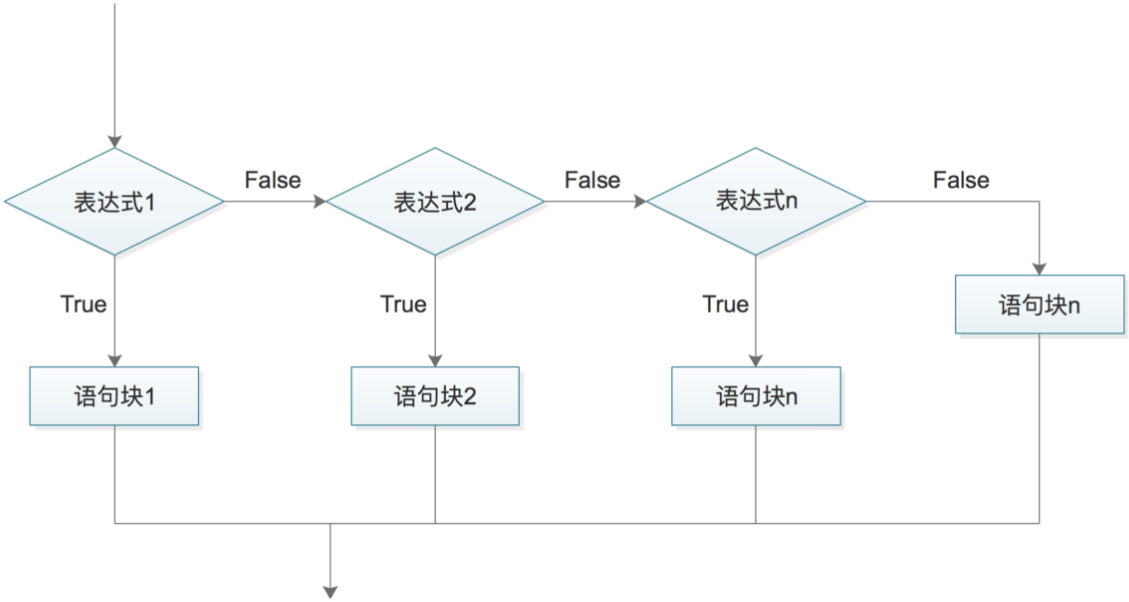
```
num = int(input("输入一个三位数："))

unit = num % 10  # 个位数

ten = num // 10 % 10  # 十位数

hundred = num // 100  # 百位数

result = "是" if unit ** 3 + ten ** 3 + hundred ** 3 == num else "不是"

print("{}{}水仙花数".format(num, result))
```

(3). 多分支选择结构

多分支结构用于处理多个条件，多个分支的情况。其一般格式如下：

```
    if 表达式1:
        语句块1
    elif 表达式2:
        语句块2
    elif 表达式n:
        语句块n
    else:
        缺省语句块
```

if-elif-else语句的执行流程：



例题：输入成绩，判断等级

> 85-100为优秀；70~84为良好；60-69为及格；60以下不及格

```
score = float(input("请输入学生的成绩："))

if score >= 85:
    grade = "优秀"
elif score >= 70:
    grade = "良好"
elif score >= 60:
    grade = "及格"
else:
    grade = "不及格"
print("百分制成绩：{},成绩等级：{}".format(score, grade))
```

> 从上面的例题可知。if语句中定义的变量在当前的作用域中都可用。

(4). 嵌套选择

例题：编写一个登陆程序

> 首先对输入的用户名进行验证。如果存在，再对输入的密码进行验证。

```
# 模拟用户名与密码
USER = "admin"
PWD = "888"

# 从键盘输入用户名
userName = input("请输入用户名：")

# 验证用户名
if userName == USER:
```

```
    pwd = input("输入密码: ")
    # 验证密码
    if pwd == PWD:
        print("登陆成功! ")
        print("欢迎{}".format(userName))
    else:
        print("密码不正确")
else:
    print("用户名不存在")
```

2.5.3 循环结构

循环结构是控制一个语句块重复执行的程序结构。它由两部分构成：

- 循环体：重复执行的语句块
- 循环条件：控制是否继续执行循环体的表达式

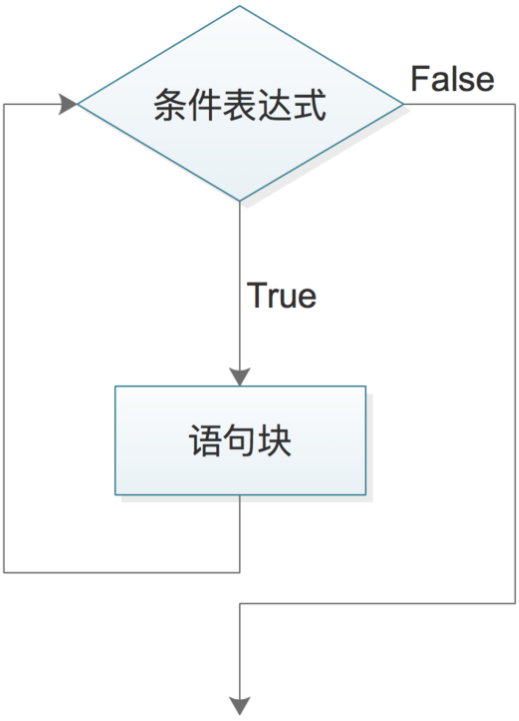
循环结构的特点是在一定条件下重复执行，直至重复执行到一定次数或该条件不再成立为止。

(1). while语句

while语句的功能是在满足指定条件时执行一个语句块。其语法结构如下：

```
while 循环条件表达式:
    语句块
```

while语句的执行流程如下：



例题：1加到100

```
i, num = 0,0

while i<=100:
    num+=i
    i+=1
print(num)
```

(2). for语句

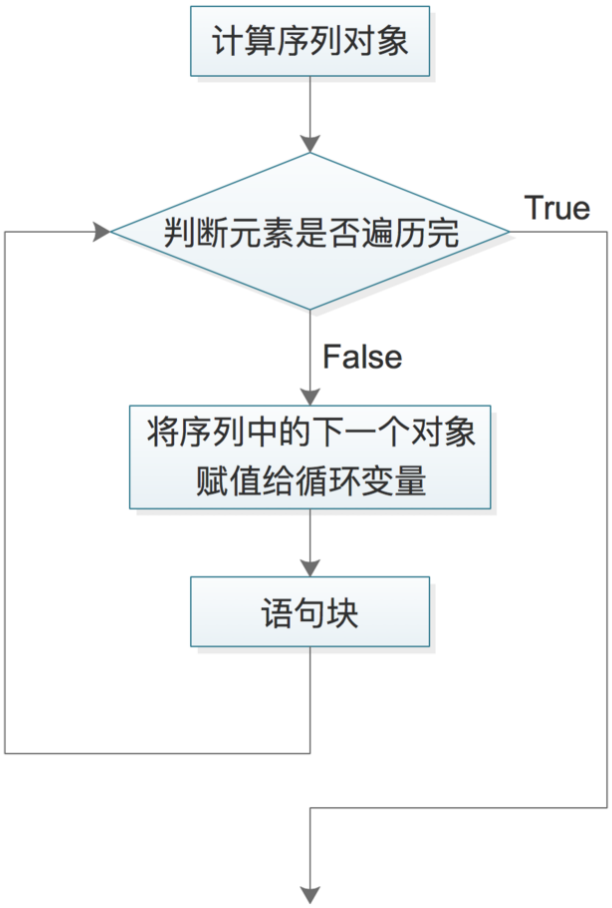
在Python中，for语句是一个通用的序列迭代器，可以用于遍历（是指沿着某条搜索路线，依次对序列中每个对象做且仅做一次访问）任何有序序列对象中的所有元素。语法格式：

```
for 循环遍历 in 序列对象：
    语句块
```

序列对象有：

- 字符串
- 列表
- 元祖
- 集合
- 字典

for语句的执行流程：



例题1：获取字符串中的所有字符

```
str = input("输入字符串")

for char in str:
    print(char)
```

例题2： 遍历0-99的整数序列

> range函数可以创建一个整数序列

```
# 打印0到99
for i in range(100): # range创建的是一个半开区间。包左不包右
    print(i)
```

例题3：找到1000以内的素数

> 素数是大于1的自然数。特点：除了1和自身外不能被其他自然数整除。要判断一个数是否素数，可以用2，4,...,`$\sqrt{n}$`去整除它。如果不能被整除，则n为素数。

2.5.4 循环控制语句

如果想要改变循环的执行流程，可以使用Python提供的循环控制语句

(1). break语句

break语句用于终止当前循环的执行操作。

例题： 前100个自然数之和

```
i, sum = 1, 0

while 1:
    sum += i
    i += 1
    if i == 101: break
print("1+2+...+1000={}".format(sum))
```

(2). continue语句

continue语句用于跳出本次循环。

```
for str in "Python":
    if str == "t": continue
    print(str) # p,y,h,o,n
```

(3). pass语句

为了保持程序结构的完整性，Python提供了一个空语句pass。pass语句一般仅作为占位语句，不做任何事情。

```
for str in "Python":
    if str == "t":
        pass
        print("---")
    print(str) # p,y,---,t,h,o,n
```

2.6 函数

函数是从英文function翻译过来的。function在英文中的意思是"功能"。从本质意义上来说，函数就是用来完成一定功能的。在编程中，函数就是一组用于完成特定功能的语句块。

2.6.1 函数的定义与调用

函数定义语法格式如下：

```
def 函数名(形式参数):
    函数体
```

- 函数体第一行可以使用文档字符串，用于存放函数说明；
- 函数体可以使用return来结束函数，使系统有选择性地返回一个值给调用代码；如果未使用return语句，或者使用了不带表达式的return语句，系统会返回None。
- 函数体中使用一个pass语句，将定义一个空函数。

【例题】定义一个计算矩形面积的函数，并查看其帮助

```
def area(width, height):
    '''
    function: 计算矩形面积
    :param width: 宽度
    :param height: 高度
    :return: 矩形的面积
    '''
    return width * height

help(area) # 查看函数
```

函数调用语法：

```
函数名(实际参数)
```

- 实际参数应当与形式参数按顺序一一对应，而且参数的数据类型需要保持一致
- 函数定义和函数调用可以在同一个程序文件中，此时函数定义必须位于函数调用之前。
- 函数定义和函数调用可以放在不同的程序文件中，此时需要先导入函数定义所在的模块，然后才能调用函数

【例题】调用area函数，并计算值

```
a = area(3, 4)
print(a) # 12
```

2.6.2 参数的传递

函数的参数传递主要有两种类型：

- 值传递： 传递的是实参变量的值。不会影响到实参变量
- 引用传递：传递的是实参变量的地址。会影响到实参变量

在Python中，函数参数传递机制采用的是对象引用传递方式。这种方式是值传递与引用传递的结合。在参数内部对形参变量所指向对象的修改是否会影响到函数外部。这要取决于对象本身的性质。 在python中对象分为可变对象和不可变对象：

- 可变对象：包括列表、集合和字典。如果参数属于可变对象，则相当于按引用传递
- 不可变对象：包括了数字、字符串、元祖、不可变集合，相当于按值传递

```
def fun(num, str, st, tup):
    num += 10
    str = str.upper()
    st.add("set")
    tup = (1, 2, 3)


num = 10
str = "hello"
st = {1, 2, 3}
tup = (1, 2, 3)
fun(num, str, st, tup)
print(num)  # 10
print(str)  # hello
print(st)  # {1, 2, 3, 'set'}  # 只有可变对象值受到函数的影响
print(tup)  # (1, 2, 3)
```

2.6.3 参数的类型

(1). 位置参数

调用参数时，通常是按照位置匹配方式传递参数，即按照从左到右的顺序将各个实参依次传递给相应的形参，此时要求实参的数目和形参的数目相等。

```
def add(x, y, z):
    return x + y + z


print(add(3, 4, 5))  # 12
```

(2). 关键字参数

调用函数时，如果不想按照位置匹配的方式传递参数，则可以使用传递关键字参数，即通过形参的名称来指定将实参值传递给哪个形参。 语法格式：

```
def add(x, y, z):
    return x + y + z


result = add(x=3, z=4, y=2)
print(result) # 9
```

(3). 默认值参数

定义函数时可以为形参指定默认值。 语法格式如下：

```
形参名称=默认值
```

默认值参数必须位于形参列表的最右端。如果对于一个形参设置了默认值，则必须对其右边的所有形参设置默认值。 调用带有默认值参数的函数时，如果未提供参数值，则形参会取默认值。

```
def student(name, gender="M"):
    print("姓名：{}\t性别：{}".format(name, gender))
student("张三")  # 姓名：张三  性别：M
```

(4). 元祖类型的变长参数

定义参数时，如果参数数目不固定，则可以定义元祖类型变长参数。方法是在形参前面加“*”号。这样的形参可以接受任意多个实参并将其封装成一个元祖。

这种元祖类型的变长参数可以看成是可选项，调用函数时可以向其传递任意多个实参值，各个实参值用逗号分割。当不提供任何实参时，相当于提供了一个空元祖作为参数。

```
def add(*args):
    x = 0
    if len(args) > 0:
        for i in args:
            x += i
    return x


print(add(3, 4, 4))  # 11
```

如果函数还有其他形参，一般放在这类变长参数之前。如果放在变长参数之后，调用时需要使用关键字参数的形式：

```
def add(*args, power):
    x = 0
    if len(args) > 0:
        for i in args:
            x += i ** power
    return x

# 在元祖变长参数之后的参数，在函数调用时需要用关键字参数形式
print(add(3, 4, power=2)) # 25
```

(5). 字典类型变长参数

定义函数时，可以定义字典类型的变长参数，方法是在形参名称前面加两个星号“**”。如果函数还有其他形参，则必须放在这类变长参数之前。 调用函数时，定义的字典类型变长参数可以接受任意多个实参，各个实参之间以逗号分隔。语法：

```
键=实参值
```

```
def student(sid, **args):
    print(sid, end="\t")
    if len(args):
        for stu in args.items():
            print("{}={}".format(stu[0], stu[1]), end="\t")


student("20190101", name="zs", age=18) # 20190101    name=zs age=18
```

2.6.4 高阶函数

在python中，调用函数时也可以将其它函数名称作为实参来使用，这种能够接受函数名称作为参数的函数称为高阶函数。

(1). 函数式编程

在python中，可以将函数名称赋值给变量，赋值后变量指向函数对象；也允许将函数名称作为参数传入另一个函数，还允许从函数中返回另一个函数。

```
def add(x, y):  # 定义加法函数
    return x + y


def substract(x, y):  # 定义减法函数
    return x - y


def multiply(x, y):  # 定义乘法函数
    return x * y


def divide(x, y):  # 定义乘法函数
    return x / y


def artithmetic(x, y, operate):  # 定义算术运算函数。形参operate接受函数名称
    return operate(x, y)


a, b = eval(input("请输入两个数（以，号分割：）"))
c = input("请输入运算符")
if c == "+":
    op = add  # 函数名称赋值给变量
elif c == "-":
    op = substract
elif c == "*":
    op = multiply
elif c == "/":
    op = divide
else:
    op = -1
if op != -1:
    print(artithmetic(a, b, op))  # 将实参op指向的函数作为参数赋值给函数
```

(2). map函数

内置的map()函数是一个可用于序列对象的高阶函数，其调用格式如下：

```
    map(func,iterable)
```

- func 用于指定映射函数
- iterable：用于指定可迭代对象

- return: map()函数将映射函数func()作用到可迭代对象iterable的每一个元素并组成新的map对象返回

【例题】将输入的所有单词转换为大写

```
upper = str.upper  # 为了防止str变量覆盖内置函数


def get_strs():
    strs = []
    print("请输入英文单词（q=退出）")
    while 1:
        str = input("请输入：")
        if str.lower() == "q": break
        strs.append(str)
    return strs


strs = get_strs()

print("=" * 30)

print("输入的单词：")
for str in strs:
    end_str = "," if strs.index(str) < len(strs) - 1 else "\n"
    print(str, end=end_str)

print("=" * 30)
print("格式化后的单词")
# 调用map函数，以 upper作为参数，也就是将其执行的upper函数传入map函数。第二个参数为可迭代的对象
strs = map(upper, strs)
list = list(strs)  # map对象转换为列表。目的是使用列表中的index方法
i = 0
for str in list:
    i += 1
    end_str = "," if list.index(str) < len(list) - 1 else "\r"
    print(str, end=end_str)
```

(3). filter函数

内置函数filter()也是一个高阶函数，可用于对序列进行筛选。格式如下：

```
filter(func,iterable)
```

- func：指定筛选函数
- iterable：指定可迭代对象

- func()函数将作用于迭代对象中的每一个元素，并根据func的返回值是True还是False类判断是保留还是丢弃该元素
- return： 返回值是一个filter对象。

【例题】 奇数筛选

```
# 奇数筛选函数
def is_odd(num):
    return num % 2 == 1


nums = [x for x in range(1, 51)]
# 筛选奇数
nums_fliter = filter(is_odd, nums)

filter_list = list(nums_fliter)

i = 0
for lst in filter_list:
    i += 1
    print(lst, end="\t")
    if i % 5 == 0: print("\r")
```

2.6.5 匿名函数

匿名函数是指没有名称的函数，它只能包含一个表达式，而不能包含其他语句，该表达式的值就是函数的返回值。

使用def语句创建用户自定义函数时必须指定一个函数名称，以后需要时便可以通过该名称来调用函数，这样有助于提高代码的复用性。如果某项计算功能只需要临时使用一次而不需要在其他地方重复使用，则可以考虑通过定义匿名函数来实现这项功能。

(1). 匿名函数的定义

语法格式：

```
lambda 参数列表:表达式
```

- 关键字lamba 表示匿名函数，因此匿名函数也称为lambda函数
- 参数列表中的各个参数以逗号分隔
- 表达式的结果就是函数的返回值

(2). 匿名函数的调用

函数式编程是匿名函数的主要应用场景。虽然没有名称，但匿名函数仍然是函数对象，在程序中可以将匿名函数赋值给一个变量，然后通过该变量来调用匿名函数。

```
f = lambda x, y: x * y

res = f(3, 4)
print(res)  # 12
```

(3). 作为高阶函数的参数

【例题】 匿名函数作为过滤函数的参数，用来筛选出1-10之间的所有奇数

```
odd = filter(lambda x: x % 2 == 1, range(1, 11))
print(list(odd))  # [1, 3, 5, 7, 9]
```

(4). 作为序列或字典的元素

```
op = {"add": lambda x, y: x + y, "sub": lambda x, y: x - y, "multi": lambda
x, y: x * y, "divide": lambda x, y: x / y}

res = op["add"](3, 4)
print(res) # 7
```

(5). 作为函数的返回值

【例题】 算术四则运算

```
def arithmetric(c):
    if c == "+":
        return lambda x, y: x + y
    elif c == "-":
        return lambda x, y: x - y
    elif c == "*":
        return lambda x, y: x * y
    elif c == "/":
        return lambda x, y: x / y


if __name__ == '__main__':
    x, y = eval(input("请输入两个数字"))
    choice = input("请选择一种运算方式:")
    if choice in ["+", "-", "*", "/"]:
        print("{0}{1}{2}={3}".format(x, choice, y, arithmetric(choice)(x,
y)))
    else:
        print("无效的运算方式")
```

2.6.6 递归函数

递归函数是指自我调用的函数，即在一个函数内部直接或间接调用该函数自身。 递归函数具有以下的特性：

- 必须有一个明确的递归结束条件
- 每当进入更深一层递归时，问题规模相比上次递归都应该有所减少
- 相邻两次递归之间有紧密的联系。通常前一次的输出会作为后一次的输入。

【例题】 从键盘输入一个正整数，然后计算其阶乘 阶乘运算可以表示为：

$$\begin{cases} 1 & n \leq 1 \\ n(n-1)! & n > 1 \end{cases}$$

```
def fact(n):
    if n>1:
        return n*fact(n-1)
    else:
        return 1
print(fact(4))
```