Python编程指南

人民教育出版社信息技术编辑室　编著

编写人员

主　　编：郭　芳

作　　者：熊雪亭　王艳侠　谢　华　王学红　魏倩文

责任编辑：慈黎利

目录

第1章　程序语言基础

第2章　程序开发初步

目录

本章学习要点

- 交互式编程和文件式编程
- 变量、函数、数据类型
- 列表、元组、字典和集合
- 程序的三种基本结构
- break、continue等语句

第1章 程序语言基础

编写程序时，需要使用特定字符集里的词汇、标点和符号，并要按照一定的规则输入计算机中。这种人为规定的字符集和规则等就构成了程序设计语言。程序设计语言有许多种，例如，C语言、Logo语言、C++语言、PHP语言、Java语言等。其中Python语言简单易用，可以开发出各种功能的应用程序。本章以Python为例，介绍程序语言的相关基础知识。

活动任务

1. 通过修改已有的程序，体会编写Python程序的过程。
2. 通过交互式编程，熟悉Python常用的数据类型和数据结构。
3. 通过编程活动，熟悉变量、函数等基本概念。
4. 通过编写简单程序，熟悉程序的三种基本结构。
5. 小组协作商讨克服难点，然后独立开发稍复杂的Python程序。

活动计划

第1节：数据的表示与组织

通过识读、修改和自编Python程序，体会编写Python程序的基本过程，熟悉常用的数据类型和数据结构，初步掌握它们的基本特点。

第2节：程序的基本结构

通过利用顺序结构、分支结构和循环结构编程解决实际问题的活动，归纳三种基本结构的特点，进一步理解变量、常量、条件表达式等基本概念。

活动要求

1. 各编程任务涉及的难点内容，建议以小组形式讨论、研究，然后由个人独立完成。
2. 交流评价时，所有的Python程序运行无错误，且能完成指定的信息处理任务。
3. 在活动过程中应尽快从整体上熟悉数据类型、数据结构和程序的基本结构。

人们已经离不开各类应用程序。预报天气情况时，需要设计程序在短时间内对大量云图以及来自各地的数百万个数据进行计算；招生录取时，需要相应的程序，根据考生的成绩和填报志愿以及各校的招生名额等信息进行录取；在银行的日常工作中，需要通过程序实现储户信息的录入、保存、更新，以及每一笔存单利息的结算，等等。

虽然现在已经有了各种软件，但程序开发仍然具有重要的意义。因为没有了程序开发，现有的软件就无法更新换代，也不可能出现新的软件来满足人们的新需求。对学习者而言，在编程解决问题的过程中，能更深刻地认识计算机的内在工作机制，更充分地了解它的优势和局限性。

下面列出了本章需要用到编程库，更多的相关库通常会在安装下面的库时自动安装。

库	简　介	pip安装命令
pygame	常用于制作游戏	pip install pygame
pypinyin	给汉字标注拼音	pip install pypinyin
pyttsx3	把文字转换成语音	pip install pyttsx3
jieba	把中文文本分成词	pip install jieba

第1节 数据的表示与组织

学习目标

- 熟悉Python编程软件，会运行Python程序。
- 熟悉变量、运算符、函数等基本概念。
- 知道常用的数据类型和数据结构。
- 感受编写Python程序的乐趣。

Python是一种适合中学生学习使用的程序设计语言，这种语言的语法简洁、易懂，并且拥有功能丰富的编程模块和编程库。利用它，可以轻松开发出各种程序。

下表列出了要使用的Python程序，让我们开始编程之旅吧。

程　序	简要说明
snake.py	贪吃蛇小游戏
sunflower.py	绘制太阳花的小程序
smartagent.py	拼音小能手，给文字标注拼音的程序
wordcount.py	统计文本的长度和字的个数
ball.py	控制小球躲避障碍物的小游戏
super24.py	程序根据人输入的数，自动找到计算24的计算式
idioms.py	成语接龙，人机交替根据成语最后一个字音接龙
musicplay.py	播放音乐的小程序
play_ttt.py	人机博弈，玩井字棋
findsamefile.py	寻找计算机中的重复文件

一、体验编程

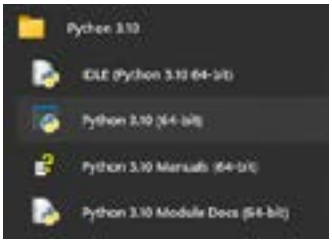
用编程语言写出一条条有特定功能的语句，然后把语句集合起来，就可以形成程序。编写Python程序主要有交互式和文件式两种方式。

交互式编程

交互式编程的特点是：每输入一个语句段，就可以立刻看到运行结果。

体验交互式编程。

1 参照下图找到并执行菜单中的IDLE命令，打开用于交互式编程的窗口。



2 输入式子“1+2+3+4+5+6+7+8+9−10”后敲回车键，观察窗口中的显示。

```
>>> 1+2+3+4+5+6+7+8+9-10      # 整数加法、减法运算
35
```

注意，这个算式就是一条Python语句。这条语句主要涉及整数的加法和减法运算，其运算符、运算顺序和运算结果等，都跟数学中的运算相同。

3 输入下面的语句后敲回车键，观察窗口中显示的计算结果。

```
>>> 3.14*2
6.28
```

这条语句可以看作是计算直径为2的圆的周长，其中不光涉及整数，还涉及带小数点的数。在程序设计语言中，一般把带小数点的数称为浮点数。一个整数与浮点数进行运算，其结果为浮点数。这个式子涉及乘法运算，运算符是*。

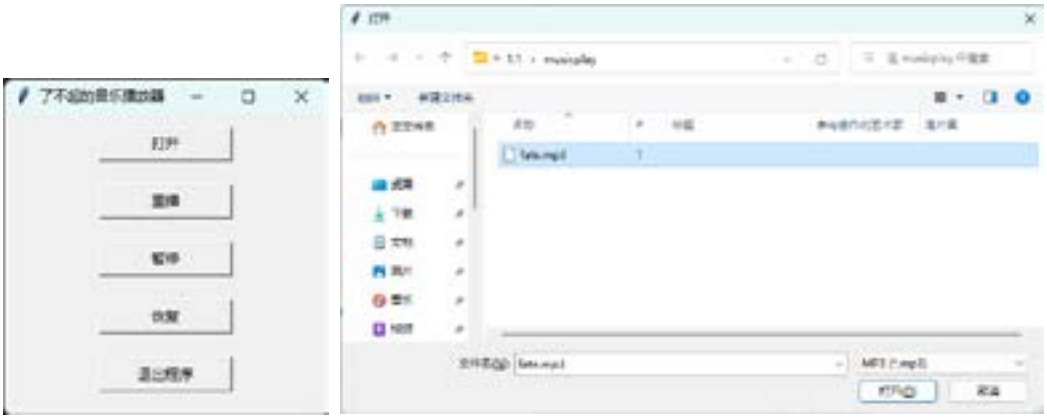
文件式编程

文件式编程的特点是一次性把一个程序所需的语句都存放到文件中，计算机得到“执行”的指令后，再执行相应的语句。

运行程序文件。

1 执行 File 菜单中的 Open 命令，打开文件musicplay.py。

2 执行 Run 菜单中的 Run Module 命令，运行程序并尝试使用。可以发现，这是一个播放音乐的程序。



3 执行 File 菜单中的 Open 命令，参照屏幕中的提示打开并运行名为idioms.py的程序，尝试与计算机来一次成语接龙的比赛。

```
成语接龙。随机选择的初始词是：有来有往
0 分，有来有往 (c)
参照 往 字读音输入成语（t 提示；e 解释）：e
y ǒ u lái y ǒ u w ǎ ng
参照 往 字读音输入成语（t 提示；e 解释）：枉费心机

10 分，有来有往 (c) → 枉费心机 (10) → 积雪囊萤 (c)
参照 萤 字读音输入成语（t 提示；e 解释）：t
提示：['营私罔利','迎刃以解','蝇头蜗角','迎来送往','迎奸卖俏','迎风待月']
参照 萤 字读音输入成语（t 提示；e 解释）：蝇头小利

20 分，有来有往 (c) → 枉费心机 (10) → 积雪囊萤 (c) → 蝇头小利 (20) → 利欲熏心 (c)
参照 心 字读音输入成语（t 提示；e 解释）：
```

操作时，随时可以根据自己的需要对程序文件进行适当修改，从而获得不同的运行结果。

修改程序并运行。

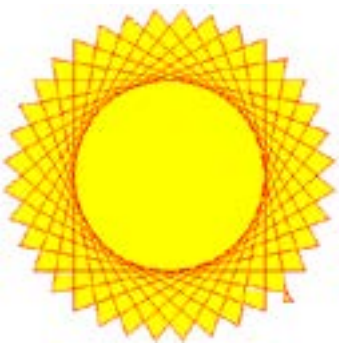
1 参考前面的操作，打开并运行名为sunflower.py的Python程序文件。仔细观察里面的语句，尝试根据注释理解语句的功能，但不必细究。

```
from turtle import *          # 引入 turtle 模块中的所有函数
color('red', 'yellow')        # 指定颜色
begin_fill()                  # 开始填充
for i in range(50):           # 构建循环
    forward(200)              # 沿当前方向前进 200
    left(170)                 # 向左旋转 170 度
end_fill()                    # 结束填充
```

注意，以#开头的内容属于程序注释，只用于帮助人理解程序。

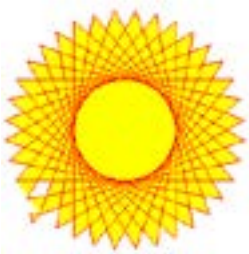
2 执行 Run 菜单中的 Run Module 命令，运行程序，观察运行结果。可以发现，这个程序是个画图程序。

3 尝试修改程序中的参数，比如把200改成180、250等，把170改成120、110等，然后运行程序，观察绘制图形的变化。



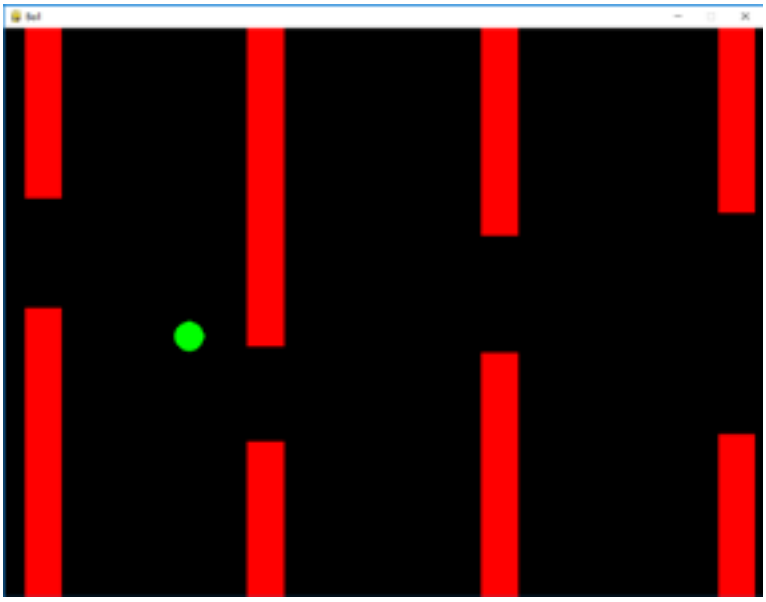
250,110

180,120



200,130

4 打开并运行名为ball.py的程序文件，观察运行结果。



这是一个躲避障碍物的游戏。程序运行过程中，小球会随时间不断下沉；敲空格键，可以控制小球暂时上升一段距离；敲↓键，可以控制小球主动向下移动；敲←或→键，可以控制小球前后移动。游戏时，要想办法通过敲键，控制绿色小球避开红色障碍物。

5 关闭游戏界面，参照下面的提示查看程序的源代码，然后尝试根据需要进行修改，如调整上升的距离，前后移动的距离等。修改后，重新运行，看看游戏的难度是否有了变化。

```
if event.type == pygame.KEYDOWN:        # 敲了键盘
    if event.key==pygame.K_r:           # 敲 r 键
        init()                          # 重新启动游戏
    if event.key==pygame.K_SPACE :      # 敲了空格键
        if (not pause): ball.action(-40)    # 上升 40 的距离
    if event.key == pygame.K_LEFT:      # 敲 ← 键
        if (not pause):ball.move(-50)       # 后移 50 的距离
    if event.key == pygame.K_RIGHT:     # 敲 → 键
        if (not pause):ball.move(50)        # 前移 50 的距离
    if event.key == pygame.K_DOWN:      # 敲 ↓ 键
        if (not pause):ball.move(50)        # 下移 50 的距离
```

除了修改程序，也可以根据需要，自行编写程序，并利用程序求解问题或完成指定的任务。

编写程序并运行。

1 关闭前面执行的程序。

2 执行File菜单中的New File命令，新建一个文件，然后把下面的代码输入新建的文件中。注意，#开头的内容为注释内容，可以不输入。

```
# 从 pypinyin 中引入 pinyin 函数
from pypinyin import pinyin
# 根据字符串生成拼音，拼音保存在变量 r 中
r=pinyin(' 我爱北京天安门 ')
# 利用 print 函数，输出变量 r 的值
print(r)
# 定义一个字符串变量
s=' 好大喜功是不好的 '
# 给变量 s 对应的字符串标注拼音，并交给 print 函数输出
print(pinyin(s))
```

运行结果

```
[['wǒ '], ['ài'], ['bù'], ['jī ng'], ['tiān'], [' ān'], ['mén']]
[['hào'], ['dà'], ['xǐ '], ['gōng'], ['shì'], ['bù'], ['hǎo'], ['de']]
```

3 保存文件，然后参考前面的操作运行程序，观察运行结果。可以发现，这个程序的功能是标注拼音。

4 修改字符串中的文字，然后再次运行程序，看看能否正确标注拼音。

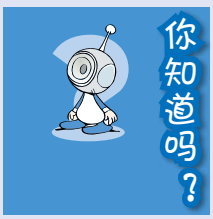
5 打开配套资源中的其他程序并尝试运行，了解这些程序的功能，感受Python的用途和文件式编程的特点。

交互式编程和文件式编程各有特点：交互式编程能立刻获取结果，适合进行简单计算或对某些语句、函数功能进行测试；文件式编程不能立刻得到结果，但编写过程中可以方便地对前后进行修改，适合编写较复杂的程序。

1. Python程序文件的默认扩展名是什么？

2. 通过前面的例子，你觉得Python语言编程都能用来处理哪些方面的问题？

3. 你还希望用Python做什么？列出愿望清单，然后通过搜索引擎搜索，看看是否有人进行了相应的Python编程。



“工欲善其事，必先利其器”，编写程序也是如此。一个友好、易用的开发环境可以有效降低学习门槛，帮助初学者把更多注意力集中到程序的编写和理解上。

学习Python语言时，可以使用多种开发工具。除了前面介绍的Python官方提供的IDLE外，很多人在使用Spyder、Jupyter、VSCode、Eclipse、PyCharm等。

二、数值、文本与布尔值

现实世界中有数值、文本、图像、声音、视频等各种信息，要用程序来处理这些信息，就需要先解决信息的表达问题，即如何把已知信息用程序语言描述出来。虽然计算机中的一切最终都可以归纳为二进制数，但为了便于人们思考和表达，现代的程序设计语言通常会根据数值、文本等信息的表达需求，提供特定的数据类型。

下面主要介绍数值、文本、布尔值在Python中的表达。

数值

Python程序中，主要用整型和浮点型来表示数值，其中浮点型可以理解为带小数点的数。

体验数值运算。

1 打开命令交互式窗口，然后参照下面的代码，尝试除法相关的运算。

```
>>> 4/2; 5//2; 5%2          # 除法相关运算，语句间用分号隔开
2.0
2
1
```

注意，这个案例在一行中输入了三条语句，语句间用分号隔开。这三条语句可简单视为涉及除、求商、求余三种运算：进行除法运算时，对应的符号是 /，它会自动把计算结果转成浮点数；进行求商运算时，对应的符号是 //；进行求余运算时，对应的符号是 %。

2 尝试带括号的运算。

```
>>> 1+2+3*4; 1+(2+3)*4; ((1+2)+3)*4
15
21
24
```

可以发现，括号内的运算先执行。多个括号嵌套时，内层括号里的优先执行。

3 尝试下面利用变量进行计算的例子。

```
>>> pi=3.14159265 ; r=3       # 圆周率和半径
>>> d=pi * r * 2              # 求圆的周长
>>> d                         # 显示相关数值
18.849555900000002

>>> s=pi * r * r              # 求圆的面积
>>> pi,r,s                    # 显示计算结果
(3.14159265, 3, 28.274333850000005)

# 求以圆为底面，以 10 为高的圆柱体的体积
>>> s * 10
282.74333850000005
```

在上面例子中，使用了pi、r、s等来标记特定的数值和计算结果，它们被称为变量。一个变量可以在后续语句中不断使用。在交互模式下，如果想知道某个变量的值，只需要输入变量名就可以了；如果需要同时显示多个变量，可以用逗号把它们隔开，Python会把它们当成一组数据显示出来。

使用变量时，要先给变量赋值。在大多数程序设计语言中，赋值符号与数学中的等号“=”形式相同，但意义完全不同。

4 尝试幂运算。

```
>>> 3 ** 2      # 乘方
9
>>> 9 ** 0.5    # 开方
3.0
```

上面的例子使用了Python的幂运算符**。当指数为2时，相当于求平方；当指数为0.5时，相当于求平方根。

文本

包含文字、符号的文本型数据，各种程序设计语言主要用字符串来进行存储和处理。在Python中，字符串用成对的单引号、双引号等括起来。处理时，可以对字符串进行拼接、选取、测量长度、替换等操作。

体验字符串。

1 拼接字符串。

```
>>> s1='我爱北京'        # 单引号括起来的字符串
>>> s2="天安门"          # 双引号括起来的字符串
>>> s=s1+s2              # 用 + 号拼接字符串
>>> s                    # 显示拼接后的字符串
'我爱北京天安门'
```

在上面的例子中，分别用半角的单引号和双引号定义了两个字符串，并赋给了变量s1和s2。处理字符串时，+ 号的作用是连接，连接的结果交给了变量s。注意，单引号和双引号必须成对使用。

2 测量字符串的长度。

```
>>> len(s)                          # 测量变量 s 代表的字符串的长度
7
>>> len('No pain, no gain.')        # 测量英文字符串的长度
17
>>> len('"len" 函数用于测量长度。')  # 中英文混合字符串的长度
14
```

测量字符串长度时，使用了类似“len（字符串)”形式的语句，其中的len被称为

函数。函数可以理解为事先编制好的，用于实现特定功能的程序块。编程时经常要用到各种函数。

在Python中，字符串的长度就是所含字符的数量，无论是一个中文的字符还是一个英文的字符，其长度都是1。另外，如有需要，单引号和双引号可以互相嵌套使用，但必须保证内外顺序匹配。

3 替换字符串中的字符。

```
>>> s=s.replace('北京天安门','故宫')
>>> s
'我爱故宫'
```

替换时使用了replace函数，它通常包含两个参数：旧字符串和新字符串，作用是把旧字符串换成新的，如把“北京天安门”换成“故宫”。

4 复制字符串。

```
>>> a='-' * 10+'我是分割线'+'-' * 10        #把字符串'-'复制10次再拼接
>>> a
'---------- 我是分割线 ----------'
```

Python语言具有强大而且便捷的字符串处理能力，后面的编程活动还会进一步介绍如何根据需要，灵活处理字符串。

布尔值

生活中经常要对某些问题进行判断，如判断变量a是不是大于变量b、“诸葛亮”这个名字在不在某句话中等。判断的结果在编程时一般用布尔值来表示，对应的是布尔型数据。

了解布尔型数据。

1 整数与布尔数。

```
>>> a,b,c=1,3,7                    # 多个变量同时赋值
>>> a>b,a<b,a<b<c,a==b             # 基于数值进行比较
(False, True, True, False)
```

上面的代码段采用了批量赋值的方式对变量a、b、c 进行了赋值，其具体做法是，把赋值号左右两端变量名和值分别用逗号隔开。注意，在Python语言中，判断两个数据是否相等的运算符是==。

2 浮点数、整数与布尔数。

```
>>>3==3.0,3<3.01
(True, True)
```

3 字符串与布尔数。

```
>>> '我' in s, '天安门' in s        #判断字符串是否在另一个字符串中
(True,False)
>>> s1==s2                          #判断字符串是否相等
False
```

上面的代码段使用了关键字in，它的作用是判断A是否在B中。与数类似，编程时也可以利用==判断两个字符串是否相等。

布尔型数据经常要用到的比较运算符可见下表。

比较符	说　明	举　例
<	小于	1<2，True；2<1，False
<=	小于等于	1<=2，True；2<=1，False
>	大于	2>1，True；1>2，False
>=	大于等于	3>=2，True；2>=3，False
==	等于	2==2，True；'A'=='B'，False
!=	不等于	2!=2，False；2!=3，True

1. 编程时用了哪几种数值类型？它们的特点分别是什么？

2. 编程时用到了哪些数学运算？它们对应的运算符分别是什么？带括号的运算如何执行？

3. 一行中要输入多条语句该如何处理？通过一条语句显示多个变量的值，该如何处理？

4. 你觉得变量是什么？使用变量编程有什么好处？

5. 在Python中，如何判断两个变量或数据是否相等？

三、序列、映射与集合

前面介绍了如何表达一个数、一串文本或一个判断结果，但这样还不够。生活中经常需要成组地处理数据。如处理一个单元的英语单词，处理一个人的姓名、性别、身高、体重、爱好等。很多程序语言提供了组织相关数据的方法，即数据结构。前面介绍的字符串，其实就是一种数据结构的应用。

Python编程时，一般会用到序列、映射、集合等类型的结构。在实际使用中，它们一般也被视为数据类型。

序列

顾名思义，序列就是按一定顺序组织元素的元素集，常见的序列类型有列表、元组等。其中列表属于可变序列，即里面的元素可以更改；元组属于不可变序列，即里面的元素不能更改。

了解列表和元组。

1 定义列表和元组。列表和元组可用来表示一个有意义的数据集，比如“学号为0235468的汉族男生张小明，年龄15岁，身高1.75米，就读初三四班，爱好旅游”就可以用下面的列表或元组加以表示。不难发现，一个列表或元组中，可以包括数值、字符串等多种类型的数据。

```
# 列表通常表现为中括号括起来的元素集合
>>> l = [ '0235468' , ' 张小明 ' ,' 男 ',' 汉 ', 15 ,1.75 , ' 初三四班 ',' 旅游 ']
# 元组通常表现为小括号括起来的元素集合
>>> t = ( '0235468' , ' 张小明 ' ,' 男 ',' 汉 ', 15 ,1.75 , ' 初三四班 ',' 旅游 ')
```

2 操作列表和元祖，通常会使用切片运算符[]。

```
# 获取指定位置的元素，序列起始位置为 0，操作符是 []
>>> l[0] , t[1]
('0235468', ' 张小明 ')
# 修改列表中指定元素的值
>>> l[1] = ' 王晓松 '
```

```
# 显示变量 l 对应的列表，可以发现第 2 个元素已变成“王晓松”
>>> l
['0235468', '王晓松', '男', '汉', 15, 1.75, '初三四班', '旅游']

# 修改元组中指定位置的元素
>>> t[1] = '王晓松'
# 提示错误信息，元组（tuple）中的元素不能修改
Traceback (most recent call last):
  File "<pyshell#9>", line 1, in <module>
    t[1] = '王晓松'
TypeError: 'tuple' object does not support item assignment
```

列表和元组分别具有可变和不可变的特性，实际使用时，可以根据需要混合使用。对于一名学生来说，学号、姓名、性别、民族等信息在就读期间通常是不变的，而年龄、身高、就读班级等是可变的，因此上面的信息可调整为：

```
>>> p = ('0235468' , '张小明 ' ,' 男 ',' 汉 ', [15 ,1.75 , ' 初三四班 ',' 旅游 '])
```

这样的元组嵌套列表的表示方式，可以防止程序其他处误改学号、姓名等不可变信息，同时还可以通过以下代码修改可变信息。

```
>>> p[4][0] = 16       # 把年龄改为 16 岁
```

执行时，计算机会根据[4]找出p序列中编号为4的元素，即第5个元素，它是一个列表，然后利用[0]找到列表的第1个元素，因为列表中的元素可修改，因此可以根据学生情况更改数据。

再比如，要展示一个球队的上场阵容，则适合列表嵌套元组的形式，如：

```
>>> team =[ ('0235468' , '张小明 ') ,('0235469' , '王大猛 ') , ('0235470' , '赵建 ') ]
```

球队的上场阵容要根据队员的状态不断变换，而人员的编号、姓名等信息通常是不可变的，这样的结构可以根据需要方便地替换阵容里的队员，同时还能防止误修改人员的基本信息。

1. 能利用以下代码修改人员的可变信息吗？为什么？

```
>>> p[4] = [15 ,1.75 , '初三四班 ',' 旅游 ']
```

2. 能利用以下代码修改球队的上场阵容吗？为什么？

```
>>> t[0] = ['0235470' , '李英杰 ']
```

映射

最典型的映射类型是字典，字典可以理解为以“键值对”方式组织起来的元素集合。键可以是数、字符串、元组等不可变元素，字典中的键不可重复，而值是可以重复的。

了解字典。

1 定义字典。从形式上看，字典是用大括号括起来的键值对的集合。比如，水果市场苹果5.0元一斤，梨3.6元一斤，香蕉6.5元一斤，这条信息就可以用字典来表示。

```
# 键和值之间用半角冒号分隔；一个个键值对之间用逗号分隔
>>> d = { ' 苹果 ' : 5.0 , ' 梨 ' : 3.6 , ' 香蕉 ' : 6.5 }
```

2 获取字典中的值。使用字典时，可以根据键获取对应的值。如果键不存在，则系统会提示异常。为了防止出错，可以用get方法增加默认值。

```
>>> d[' 苹果 ']
5.0
>>> d[' 菠萝 ']
Traceback (most recent call last):
  File "<pyshell#17>", line 1, in <module>
    d[' 菠萝 ']
KeyError: ' 菠萝 '
>>> d.get(' 菠萝 ',None)          # 也可以写成 d.get(' 菠萝 ',' 价格未知 ') 等形式
```

上述例子中使用了一个特殊的常量None，Python程序中，常用None这个常量表示没有。

3 修改字典中的键值对。

```
>>> d[' 苹果 '] = 6.0          # 键存在，则修改值
>>> d
{' 苹果 ': 6.0, ' 梨 ': 3.6, ' 香蕉 ': 6.5}
>>> d[' 菠萝 '] = 6.0          # 键不存在，则增加一个元素
>>> d
{' 苹果 ': 6.0, ' 梨 ': 3.6, ' 香蕉 ': 6.5, ' 菠萝 ': 6.0}
```

集合

集合用来保存不重复的、无序的元素。集合从形式上看和字典很相似，也是用大括号括起来的元素集合，只是里面的元素不是冒号隔开的键值对，而是用逗号隔开的单个元素。

了解集合。

1 定义集合。

```
# 定义一个集合，计算机会自动去掉其中重复的元素
>>> basket = {'apple', 'orange', 'apple', 'pear', 'orange', 'banana'}
>>> basket
{'pear', 'orange', 'banana', 'apple'}
```

2 操作集合元素。集合不是序列，不能通过切片符获取指定位置的元素，而要使用add、pop等方法。注意，集合没有“序”，因此每次运行的结果可能会不同。

```
# 用 add 方法增加元素
>>> basket.add('grape')
>>> basket
{'orange', 'banana', 'apple', 'grape', 'pear'}

# 用 pop 方法删除一个元素
>>> basket.pop()
'orange'
>>> basket
{ 'banana', 'apple', 'grape', 'pear'}

# 用 remove 方法删除指定的元素
>>> basket.remove('grape')
>>> basket
{ 'banana', 'apple',  'pear'}
```

四、类型转换

Python把数值、字符串、列表、元组、字典、集合等看作内置类型，为了解决问题，有时需要在类型之间进行转换。

利用类型转换，实现字的统计。

1 打开名为wordcount.py的程序，仔细观察要求。

```
# 要求 1. 统计文本的总长度
# 要求 2. 显示文本所用的字，并统计字的个数
# 要求 3. 给文本标注拼音

# 三个单引号为一组，前后配套使用，可用来定义含有多段的较长的字符串
text = '''
        石室诗士施氏，嗜狮，誓食十狮。施氏时时适市视狮。
        十时，适十狮适市。是时，适施氏适市。施氏视是十狮，恃矢势，使是十狮逝世。
        氏拾是十狮尸，适石室。石室湿，氏使侍拭石室。石室拭，施氏始试食是十狮尸。
        食时，始识是十狮尸，实十石狮尸。试释是事。
        '''
```

2 统计文本的总长度，就是统计字符串的长度，可用len函数来解决。

```
str_len=len(text)
print( f' 文本总长度为：{str_len}')
```

上述例子使用了带有f 前缀的字符串，这种字符串里可以包含由大括号括起来的表达式、变量等元素，计算机会先计算出大括号中元素的值，然后形成字符串。

3 统计文本中所用字的个数，也就是统计不重复字的个数，可以使用集合。

```
# 利用 set 把字符串强制转换成集合
words = set (text)
# 去掉空格和其他符号，其中 \n 表示换行符
words.remove('。');words.remove('，')
words.remove('\n');words.remove(' ')
print(words)
words_num=len(words)
print( f' 字的个数为：{words_num}' )
```

4 运行程序，观察运行结果。

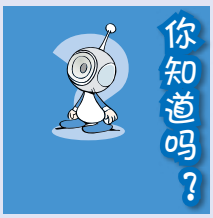
```
文本总长度为：142
{'侍','使','恃','湿','识','拭','势','士','始','实','尸','事','是','市','时',
'誓','氏','食','室','诗','逝','石','十','矢','拾','试','视','施','狮','嗜',
'适','世','释'}
字的个数为：33
```

5 参照前面所学，给文字标注拼音，并把下面的代码填写完整。

```
# 引入 pypinyin 模块
import _____
# 去掉空格、回车等特殊符号
text2=text.replace('\n','').replace(' ','')
# 标注拼音，并交给变量 r
r = pypinyin._____
print(r)
```

上面的例子介绍了如何用set函数把字符串转换成集合，因此set函数也称为集合构造器。常用的构造器可见下表。

函数	说　明	举　例
int	构造整数	int('123')，123；int(1.523)，1；int(True)，1
float	构造浮点数	float(1)，1.0；float('1.23;)，1.23；float(False)，0.0
str	构造字符串	str(123)，'123'；str([1,2,3])，'[1, 2, 3]'；
bool	构造布尔数	bool(1)，True；bool([1,2])，True；bool(None)，False
list	构造列表	list('1231')，['1','2','3','1']；list({'A','B','C'})，['A','B','C']
tuple	构造元组	tuple([1,2,3,1])，(1,2,3,1)；tuple({1:2,3:4})，(1, 3)
dict	构造字典	dict([('two', 2), ('one', 1), ('three', 3)])，{'two': 2, 'one': 1, 'three': 3}
set	构造集合	set('1231')，{'3','1','2'}；set([1,2,3,1])，{1,3,2}



1. 列表、元组、集合中的元素非常灵活，可以是数、字符串，也可以是其他列表、元组、集合、字典等。

2. 列表可以自由添加新元素，可以增加、删除、修改指定位置的元素，元组中的元素则不能修改

3. 集合没有“序”，因而无法处理位于某个位置的元素，而且里面的元素也不能重复。

4 字典中的键可以是数、字符串、元组等，但不能是列表和集合。

活动资料

变量

编程时，经常会使用各种变量。Python 语言的变量名可以由中文字符、英文字母、下画线开头，再加上数字等组成，但不能以数字开头。

为了便于记忆和使用，一般可用表示类型的前缀和表示作用的字符共同构成变量的名称，如list_team_member。这种方式表面看起来有些烦琐，但它能较好地反映变量在程序中的作用，增强程序的可读性。

不过在简单的程序中，特别是在初学编程时，为了简化输入，很多人仍会使用单个字母或简单的字母组合来作变量名。

内置函数与关键字

内置函数是由Python语言的开发者提供的函数。前面所使用的print、len、int、set等都是内置函数。内置函数还有很多，如数学函数、时间函数等。

关键字是Python保留的、有特殊用途的字符串。比如，前面程序中用到的True、False、None、in等都是关键字。

定义变量时，要避开内置函数的函数名和关键字。相关资料请自行查阅。

输入代码时要遵循的规则

输入代码时，要注意以下几点，否则会出错。

・字母大小写

Python程序严格区分大小写。例如，name、Name、NAME是不同的变量。如果不小心混淆了大小写，程序执行就可能出问题。

・语句

程序的语句一般由变量、函数、运算符、关键字等组成。一般来说，一行存放一条语句。如果要在一行中存放多条语句，则需把这些语句用分号隔开。

・缩进

Python通过每行前的空格来区分层级，同层级代码前的空格需相同。后面将对此作详细介绍。

import语句

用Python编程时，经常要引入编程模块，以便调用其中的函数等编程资源。这个时候就需要使用import语句，它的用法主要包括三种。

1. 导入模块中的所有函数，如 from turtle import *。

2. 导入模块中的特定函数，如 from pypinyin import pinyin。

3. 只导入模块，如import math。

如果用了第三种方法，则需要在函数名前加模块名，如math.factorial(10)，这样就可以求10的阶乘了。

注意，导入编程模块的语句习惯上放在程序的最前面。

活动总结

1. 参照下表，总结使用过的数据类型和数据结构。

条目	说　明	举　例
整型		
浮点型	带小数点的数	
布尔型		
字符串		
列表		
元组		
字典		
集合		

2. 参照下表，总结使用过的函数。

函数	说　明	举　例
len	测量长度	len([1,2])，2；len('123')，3
print		
replace		
get		
add	增加一个元素	{1,2}.add(3)，{1,2,3}
pop	删除一个元素	{1,2}.pop()；[1,2,3].pop()
remove	删除指定元素	{1,2}.remove(1)；[1,2,3].remove(2)
int		
float		
str		
bool		
list		
tuple		
dict		
set		

3. 总结使用过的关键字。

关键字	说　明	举　例
True	表示真	
False	表示假	
None	表示空	
in		1 in [1,2,3]，True

4. 总结使用过的运算符。

运算符	说　明	举　例
	数值运算的加减乘除	
	字符串连接	
	数值运算的乘方运算	
	切片，获取字符串或序列中的指定元素	[1,2,3][2]，3
	比较运算，大于、小于、不等于，等等	

切片运算符在程序中经常使用，但使用方法相对复杂。在后面还会对相关内容作更详细的介绍。

1. 创建一个列表，记录3位好朋友的姓名。

2. 字符串型数据“5”与整型数据8，直接进行计算“5”+8，程序会怎样？应该如何解决此问题？

3. 如果一篇文章中的汉字只采用同一个音，音调不限、标点不限，这样的文章叫作“同音文”。请编程找出下列同音文使用了多少个字，并给文字标上拼音。

人人仁人人忍人，认仁人忍人刃人。仁人仁忍人人刃，人忍人人人人仁。忍人仁人任人刃，任人刃人任仁人。

季姬寂，集鸡，鸡即棘鸡。棘鸡饥叽，季姬及箕稷济鸡。鸡既济，跻姬笈，季姬忌，急咭鸡，鸡急，继圾几，季姬急，即籍箕击鸡，箕疾击几伎，伎即齑，鸡叽集几基，季姬急极戟击鸡，鸡既殛，季姬激，即记《季姬击鸡记》。

第2节　程序的基本结构

学习目标

- 认识顺序、分支和循环三种基本程序结构。
- 认识常用的语句，熟悉更多的函数。
- 进一步认识变量、运算符、表达式等相关概念。
- 培养程序设计的思想，训练思维能力。

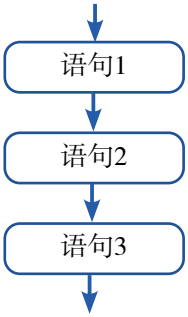
程序有三种基本结构，即顺序结构、分支结构和循环结构。在顺序结构中，计算机逐条执行语句。分支结构往往有两个或两个以上的分支，计算机可以根据不同的情况，选择执行不同分支中的语句。循环结构能够控制计算机自动重复执行某些程序段。

接下来学习程序结构的相关知识，下表是要涉及的Python程序。

程　序	简要说明
aa.py	计算AA制分餐费用
chatterbox.py	能正向和反向念绕口令
sumn.py	计算累加值
poetry.py	让人根据提示，把名句补充完整的游戏
findpasswd.py	寻找密码锁密码
s99.py	显示九九乘法表
simple24.py	一个简单的计算24的程序
guessnum.py	猜数游戏
f13.py	寻找13的倍数
perfectnumber.py	寻找完全数
poetryv2.py	增强版的补充名句游戏

一、顺序结构

执行前面的大部分程序时，计算机只要从上至下逐条执行，可以得到处理结果。这样的程序结构通常称为顺序结构。顺序结构简单明了，但计算机只能根据预先规定的方案一步步执行，不具备判断能力。下面看几个例子。



计算AA制聚餐的人均费用。

1 问题分析。多人聚会，要计算出AA制聚餐的人均费用很简单，只要用消费总额除以总人数就可以了。假设消费总额是*s*，聚餐人数是*n*，人均费用就是*s*/*n*。

2 编程实现。用编程软件新建一个文件，然后输入下面的代码。

```
s = input('输入消费总额：')      # input 函数用于接收用户的输入，默认为字符串
s = float(s)                    # 强制转换成浮点数
n =int(input('输入聚餐人数：'))  # 套用 int 和 input 函数，获得聚餐人数
print (f'人均需支付 {s/n} 元')    # 带 f 前缀的字符串，里面可包含计算式
```

3 运行程序，输入消费总额和聚餐人数，观察计算的结果。

```
输入消费总额：100
输入聚餐人数：5
人均需支付 20.0 元
```

实际编程时，经常会出现类似 s=float(s) 或 s=s+1 这样的式子，这里的等号是赋值符号，而不是等号。它的计算过程是：先计算赋值号右侧的式子，然后把计算结果赋给左边的变量。因此，在float(s)中，s是字符串变量，转换后的浮点数赋给了左侧的变量s，这时，s就成了浮点型变量。

让计算机朗读绕口令。

1 问题分析。把绕口令保存在字符串类型的变量中，然后让计算机朗读出来就

可以了，而pyttsx3是一个专注于让计算机朗读文字的编程库，调用其中的函数，就能实现文字朗读。

2 编程实现。让计算机朗读字符串中的文字。

```
import pyttsx3                          # 引入 pyttsx3
engine = pyttsx3.init()                 # 语音引擎初始化
s = '黑化黑灰化肥灰会挥发发灰黑讳为黑灰花会回飞；灰化灰黑化肥会会挥发发黑灰为讳飞花回化为灰'
engine.say(s)                           # 读变量 s 标记的绕口令
engine.runAndWait()
```

3 运行程序，可以发现，计算机会自动朗读这个绕口令。

这些程序虽然只包括顺序结构，但是也能完成特定的信息处理任务。

接下来，让要解决的问题更复杂一些，如让计算机倒背绕口令。稍微思考一下就能想到：倒背绕口令的关键在于让字符串反转过来。如何做到呢？这时可以利用切片操作符[]。

前面介绍过，利用切片操作符[]和表示位置的参数，就可以得到字符串、列表、元组等某个位置的元素，如[1,2,3][0]就可以得到1。但实际上，切片操作符的功能极其强大，它完整的模式包含用冒号隔开的3个参数，形式如下：

[起始位置：终止位置：步长]

其中步长默认为1，即“挨个切”。下表是常用的几种方式。

切片运算	说　　明	运算结果
'123'[1]	从起始位置“切”一个元素，即获取某个位置的元素	'2'
'成绩：优'[-1]	位置为负数，表示从后面开始“切”，−1 表示尾部	'优'
[1,2,3][0:2]	从起始位置0“切”到终止位置2，但不包括终止位置	[1，2]
[1,2,3,4][:]	不给数值，则默认起始为0，终止为序列最后	[1,2,3,4]
[1,2,3,4][::2]	步长为2，表示切一次，向后移两格，即跳着“切”	[1,3]
[1,2,3,4][::-1]	步长为−1，表示从尾到头，反着切	[4,3,2,1]

了解了切片操作后，就可以用它来反转字符串了。

修改前面的程序，让计算机反向朗读绕口令。

1 技术测试。用交互模式，测试是否能通过切片操作符让字符串反转。

```
>>> ts = ' 赤橙黄绿青蓝紫 '
>>> ts[::-1]
' 紫蓝青绿黄橙赤 '
```

可以发现，利用这个方式，确实可以让字符串反转。

2 修改程序。在程序中添加几条语句，让计算机反向朗读绕口令，并把下面的代码补充完整。

```
engine.say(s)                    # 读变量 s 标记的绕口令
engine.say(' 反向读 ')
engine.say(s[ _____ ])
engine.runAndWait()
```

1. 一个自然数列表中存放了 10 个数，如何用切片运算符，“切”出其中的奇数和偶数？

```
list1=[0,1,2,3,4,5,6,7,8,9]
“切”出奇数序列的代码：_____
“切”出偶数序列的代码：_____
```

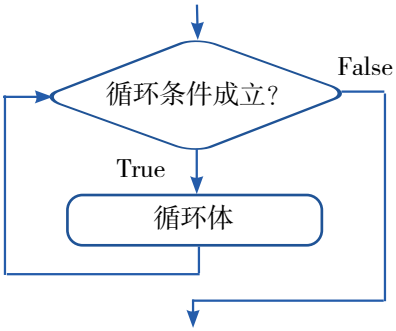
2. 文件名的后三位一般是文件的扩展名，假设变量 fn 对应着一个文件名，如何“切”出扩展名？

```
代码：_____
```

国内外的货物往往会用不同的编码体系，比如鞋子，国内的鞋号和国外的鞋号就很不相同。现在有一种鞋号 m，它与国内对应鞋号 n 的关系为 n=(m+10)/2。

请编写一个顺序结构的程序，完成鞋号的转换。

二、循环结构



要让计算机重复做一件事时，可以使用循环结构。Python中构建循环主要有两种方式：for循环和while循环。

for循环

for循环可以看作是基于序列、集合等的循环，循环执行时，for语句会不断获取列表、元组、字符串、集合等中的新元素，直到无新元素可取为止。它的基本格式如下：

```
for 变量 in 序列（或字典、集合等）:
    循环体
```

使用时要注意两点：一、for语句最后有个英文冒号，不能丢；二、循环体被认为在for语句的范围内，比for语句的层级低一级，因此循环体内各条语句的前面都要加上相同数量的表示缩进的空格。比如，循环体内每条语句比for语句前面多加4个空格。

从自然数1累乘到自然数10。

1 问题分析。要参与运算的自然数保存在列表或元组中，把其中的数挨个取出来进行累乘就可以了。

2 编程实现。

```
nums=[1,2,3,4,5,6,7,8,9,10]     # 保存数的列表
s=1                              # 保存累乘的变量
# 构建循环结构，依次从序列中取元素
for n in nums:
    s = s * n                    # 累乘结果交给变量 s，注意语句前面的缩进
print(s)
```

3 运行程序。

```
3628800
```

上面的例子使用了一条看起来有些奇怪的语句：

```
s = s * n
```

这条语句该如何理解？请指出它的执行顺序和作用。

列表、元组等中的元素比较少时，很容易写出代码，但如果元素比较多，如1到1 000的整数相加，再利用前面的方法就很烦琐了。针对这种情况，Python提供了很多用于生成序列的函数，其中range最为常用，它的格式是：

```
range( 起始值，终值，步长 )
```

默认情况下步长为1，起始值为0。只有一个参数时视为终值，规则如下：

```
range(10) 等同于 range(0,10,1 )
```

生成的序列中不包括终值。

从自然数1累加到自然数1 000。

1 问题分析。含有1 000个元素的序列，挨个写出来非常烦琐，可以用range函数来生成所需的序列，然后在此基础上构建用于累加的循环。

2 程序实现并运行。

```
s=0
# 从 1 开始而不是从默认的 0 开始，不包括终值，因此终值要多加 1
for n in range(1,1001):
    s=s+n
print(s)
```

运行结果：500500

如果要求1到1 000以内奇数的和，该怎么修改？

```
for n in range(_____):
    s=s+n
print(s)
```

for语句也可用于生成各种序列，常见操作可见下表。

语　　句	结　　果	说　　明
[x for x in range(0,10,2)]	[0,2,4,6,8]	0到10的偶数列表，不包括终值10
tuple(x**2 for x in [1,2,3,4])	(1, 4, 9, 16)	基于数列生成存有平方的元组
[x for x in '一二三四五']	['一', '二', '三', '四', '五']	把字符串元素转换成列表元素
[x*2 for x in '寻觅冷清凄惨戚']	['寻寻', '觅觅', '冷冷', '清清', '凄凄', '惨惨', '戚戚']	基于字符串，生成新的含有字符串的序列

while循环

while循环可理解为基于条件的循环。执行时，计算机会检测while语句中的循环条件是否成立，如果成立就执行循环体，否则结束循环。它的基本语句格式为：

```
while 条件表达式：
    循环体
```

编写古文名句填空的小游戏。

要求：用户根据提示输入名句，输对了结束，否则反复输。

1 问题分析。需要用户反复输入，应该使用循环结构，循环条件就是用户的输入是否正确。

2 程序实现并运行，然后把下面的代码补充完整。

```
line=[' 天生我材必有用 ',' 千金散尽还复来 ']     # 古文名句
guess=''                                  # 变量 guess 保存输入的句子，初始为空
while guess!=_______:                     # 输入不正确，就继续输入，从而形成循环
    guess=input(f"{line[0]} 的下一句是：")       # 提示用户输入，输入为字符串
print(f' 答对了！完整的答案是：{line[0]}，{line[1]}。' )
```

运行结果

```
天生我材必有用 的下一句是：明镜高堂悲白发
天生我材必有用 的下一句是：千金散尽还复来
答对了！完整的答案是：天生我材必有用，千金散尽还复来。
```

使用while循环时需要注意，循环体中通常应该具备能让循环条件发生改变的语句，否则就可能形成死循环。

代码段1	代码段2	代码段3
i=0 while i<5:　#循环条件是i<5 　print(i)　#输出i值 　i=i+1　#让i值发生变动	i=0 while i<5: 　print(i)	i=0 while i<5: 　print(i) 　i=i−1
正常循环，随着程序的执行，变量i的值不断增加，会使得循环条件不成立	死循环，变量i的值不变，循环条件永远成立	死循环，随着程序执行，变量i的值不断减小，循环条件永远成立

循环嵌套

一个循环嵌套在别的循环体中时，称为循环嵌套。循环嵌套可以有很多层，执行时内层循环先执行，完整执行一遍后，外层循环执行一次，然后继续执行内循环，直到外循环结束。

列出密码锁可能的密码。

要求：4位数字密码锁，前两位是3和2，第3位是3的倍数，第4位是4的倍数，要求编程列出所有可能的数字密码。

1 问题分析。这道题就是要列出第3位和第4位可能数字的组合，第3位是3的倍数，可能是0、3、6、9，第4位可能是0、4、8。

2 参考下面的代码编程，并把代码补充完整。

```
r=[]          # 用于保存可能的密码
for i in [0,3,6,9]:
    for j in _______:
        r.append( f'32{_}{_}') # 调用列表的 append 方法，增加元素
print(r)
```

运行结果

```
['3200', '3204', '3208', '3230', '3234', '3238', '3260', '3264', '3268', '3290', '3294',
'3298']
```

前面的程序含有两个for循环，可以根据循环变量分别称为i循环和j循环，显然i循环是外循环，j循环是内循环。j循环整体是i循环的循环体，因此j循环的起始位置要比i循环多一层缩进。j循环的循环体要比j循环的起始位置多一层缩进。也就是说，j循环的循环体实际上要比i循环多两层缩进。

分析运行结果不难发现，i循环的循环变量每获取一个新值，j循环都要完整地执行一遍，因此循环次数就是4乘以3，共12次，也就获得了12个结果。

展示九九乘法表。

1 问题分析。九九乘法表可以看作两个变量i、j及其乘积的组合，如果规定变量i的变化范围是1到9（包括9），那么j的变化范围就是1到i（包括i）。

2 参考下面的代码编程，并把代码补充完整。

```
for i in range(1,10):
    for j in range(1,_____):
        # 参数 end 默认情况下是 \n，即换行，而符号 \t 表示向后跳动一定的距离
        print(f'{ _ }×{ _ } = {i*j}',end='\t')
    print()                                    # 输出一个空行
```

运行结果

```
1×1=1
1×2=2  2×2=4
1×3=3  2×3=6  3×3=9
1×4=4  2×4=8   3×4=12  4×4=16
1×5=5  2×5=10  3×5=15  4×5=20  5×5=25
1×6=6  2×6=12  3×6=18  4×6=24  5×6=30  6×6=36
1×7=7  2×7=14  3×7=21  4×7=28  5×7=35  6×7=42  7×7=49
1×8=8  2×8=16  3×8=24  4×8=32  5×8=40  6×8=48  7×8=56  8×8=64
1×9=9  2×9=18  3×9=27  4×9=36  5×9=45  6×9=54  7×9=63  8×9=72  9×9=81
```

3 分别指出内层循环和外层循环的循环体，并说明最后一个print的执行次数。

这个程序与前一程序有一点显著不同：内层循环所依据的序列是根据外层循环变量的值动态计算出来的。也就是说，每执行一次外层循环，即外层循环的循环变量每获得一个新值，内层循环依据的序列都会重新计算。

编程时，循环嵌套可以有很多层，可以是while循环和for循环的任意组合，只要循环的缩进层级不出错就可以了。下面是2层嵌套结构的常见组合。

```
for x in 序列1:
    循环体1
    for y in 序列2:
        循环体2
    循环体1
```

```
for x in 序列：
    循环体1
    while 循环条件:
        循环体2
    循环体1
```

```
while 循环条件：
    循环体1
    for x in 序列:
        循环体2
    循环体1
```

```
while 循环条件1：
    循环体1
    while 循环条件2:
        循环体2
    循环体1
```

“小步快跑、快速迭代”是程序开发过程中一个非常重要的思想，它的大意是，先快速开发出一个具有基本功能的原型程序，然后在这个程序的基础上，根据使用反馈快速修改完善。可能每次修改的地方都很小，但通过快速的、不断的修改完善，就能实现程序功能的升级更新。

在介绍while循环时，举了一个填写古文名句的例子。在那个例子中，备选古文名句只有一条，显然是不够的。下面通过循环嵌套，增加备选名句的个数，增大挑战难度。参考下面的代码编写程序，并把空白处补充完整。

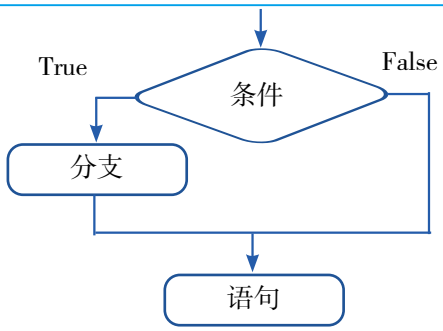
```
lines=[('天生我材必有用','千金散尽还复来')]
lines.append(('苔花如米小','也学牡丹开'))
lines.append(('七八个星天外','两三点雨山前'))
lines.append(('醉里挑灯看剑','梦回吹角连营'))
lines.append(('八千里路云和月','三十功名尘与土'))
lines.append(('人生若只如初见','何事秋风悲画扇'))
lines.append(('万物得其本者生','百事得其道者成'))
lines.append(('青春虚度无所成','白首衔悲亦何及'))
lines.append(('天下之势不盛则衰','天下之治不进则退'))
……
guess=''
for line in _______:
    while guess!=line[1]:
        guess=input(f"{line[0]}的下一句是：")
    print('答对了，请看下一题。')
print('恭喜你，全答对了！')
```

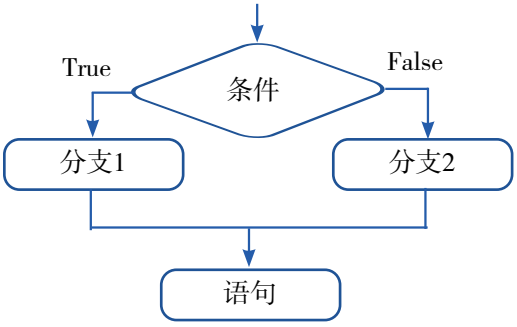
三、分支结构

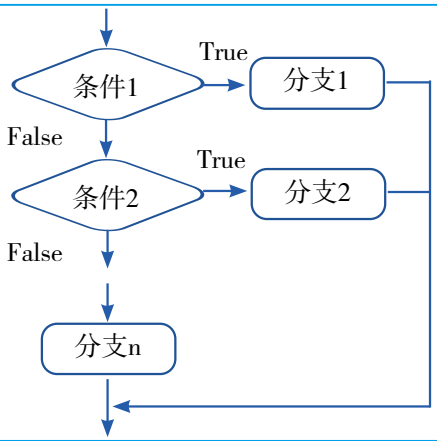
要让计算机根据不同的情况执行不同的任务，就要用到分支结构。

if分支

if分支由if语句构成，可分为单分支、双分支、多分支等多种类型。

分支形式	说明及对应代码
	单分支 if n%2 ==0: 　print(f'{n}是偶数')

	双分支
	
	if n%2 ==0:
	　print(f'{n}是偶数。')
	else:
	　print(f'{n}是奇数。')

	多分支
	
	if n>=85:
	　print('成绩为：优')
	elif 70<=n<85:
	　print('成绩为：良')
	elif 60<=n<70:
	　print('成绩为：达标')
	else:
	　print('成绩为：未达标')

计算24的小游戏。

要求：程序给出1~13之间的4个数，人根据这4个数输入一个包含四则运算的式子，如果式子的运算结果是24就认为成功。

1 问题分析。这个问题是要根据式子的运行结果进行判断，要完成这一任务就必须克服一个难点，即如何把用户输入的字符串型的式子，转换成真正可计算的式子，这里要用到eval函数。

```
>>> s='1+2'
>>> eval(s)
3
>>> eval('print("eval 函数很神奇 !")')      # 注意单引号、双引号、括号的配套使用
eval 函数很神奇 !
```

不难发现，eval函数的一大作用就是把字符串参数当作指令来执行。

2 参考下面的代码编程，并把代码补充完整。

```
a,b,c,d = 3,5,8,6    # 多个变量同时赋值
s = input(f' 根据 {a}、{b}、{c}、{d} 构想计算 24 的式子：')
r = eval(s)
if _____:
   print(' 成功 !')
else : print(' 失败 !')      # 分支语句比较简单时，可以直接写在冒号后面
```

运行结果

```
根据 3、5、8、6 构想计算 24 的式子：8*(5-6/3)
成功 !
```

实际上，if分支也可以用来赋值，即根据不同的条件赋给变量不同的值。

```
r = ' 偶数 ' if n%2==0 else ' 奇数 '
```

上面这句代码的作用等同于下面的代码。

```
if n%2==0:
   r = ' 偶数 '
else:
   r = ' 奇数 '
```

显然前面的代码更加简洁。

用程序判断用户输入的内容是否属于回文（不包括标点）。

输入一句话（不包括标点）：黄山落叶松叶落山黄
属于回文：黄山落叶松叶落山黄

提示：s==s[::−1]

猜数的小游戏。

要求：计算机选好一个整数，让人猜，猜高猜低都给出提示，猜对了则结束游戏。

1 问题分析。猜错了需要不断地猜，显然需要一个循环结构，猜高、猜低、猜对都要给出提示，显然可以使用多分支结构。

2 参考下面的代码编程。

```
goal=125                    # 等待猜测的数字
while True:
   n=int(input(' 猜一个整数：'))
   if n < goal:
      print(' 猜小了！ 请继续。')
   elif n > goal:
      print(' 猜大了！ 请继续。')
   else:
      print(' 恭喜，猜对了！ ')
      break;                # 跳出循环
```

运行结果

```
猜一个整数：100
猜小了！请继续。
猜一个整数：150
猜大了！请继续。
猜一个整数：125
恭喜，猜对了！
```

在这个程序中，多分支结构位于while循环的循环体中，反复判断执行。尽管while循环的循环条件是True，看起来像是形成了死循环，但由于循环体内包含break语句，使得程序可以在某种条件下跳出循环，因此它是一个正常的循环结构。

寻找100以内的13的倍数。

1 问题分析。显然，可以用100以内的自然数序列来形成循环结构，然后用分支结构筛选满足条件的数。

2 代码实现。

```
r=[]
for i in range(0,100):
    #continue 的作用是省略循环体内的其他语句，跳回循环初始处继续执行
    if i%13!=0 : continue
    else : r.append(i)        # 分支语句比较简单时，可以直接写在冒号后面
print(r)
```

运行结果

```
[0, 13, 26, 39, 52, 65, 78, 91]
```

上面的程序用到了一个跟循环相关的新命令continue，它的作用是省略之后循环体的其他语句，跳回当前循环初始处，然后继续执行下一次循环。

前面两个程序有一个共同点，都是在循环体中带有分支结构，但又分别使用了break和continue语句。能说说自己对这两个语句的理解吗？如果有必要，可以写出相应的代码，验证自己的看法。

while分支和for分支

顾名思义，就是利用while语句和for语句实现的分支，当for循环无法取得新的元素或者while循环的循环条件不成立了，就可以生成一个分支。

while分支改写补充古文名句的游戏。

要求：限定用户的补充次数，10次未对，就结束游戏。

1 问题分析。显然，现在有两个结束循环的条件，一个是补充对了需要结束循环，另一个是错10次也要结束循环。

② 根据下面的代码片段，编程实现。

```
guess=''; count=0                                    # 变量 count 用于记录补充次数
while count<10 :
    guess=input(f"{line[0]} 的下一句是：")
    if guess==line[1] :
        break                                        # 注意前面有两层缩进
    count=count+1
if (guess==line[1]):                                 # 区分跳出循环的原因
    print(f' 答对了！完整的答案是：{line[0]}，{line[1]}。')
else:
    print(' 错了 10 次，请再接再厉 !')
```

循环结束后，还需要进行一次判断才知道是答对了还是错10次了。

③ 代码改进。

```
guess=''; count=0                                    # 变量 count 用于记录输入次数
while count<10 :
    guess=input(f"{line[0]} 的下一句是：")
    if guess==line[1] :
        print(f' 答对了！完整的答案是：{line[0]}，{line[1]}。')  # 注意前面有两层缩进
        break                                                  # 注意前面有两层缩进
    count=count+1
else:                                                # 与 while 同层级，是 while 分支
    print(' 错了 10 次，请再接再厉 !')
```

使用while分支以后，就可以清晰地区分出是什么原因跳出循环了，因为错10次这种情况，已经放到while的分支中了。

可以这样简单理解：完整的while循环或for循环包括两个部分，一个是反复执行的循环体，另一个是执行一次的分支，在循环条件不成立时执行。

实践探究

读下面含有for分支的程序，猜测运行结果，然后运行一下，看看实际运行结果与你想的是否一致。

```
for i in range(3):
    print(i)
else:
    print('hello world')
```

```
for i in range(3):
    print(i)
    break
else:
    print('hello world')
```

四、程序结构的综合应用

实际编程时，往往需要综合利用这三种基本结构。下面编写一个程序，查找10 000以内的完全数。编程之前，先进行简单分析：

1. 要查找10 000内的完全数，这意味着列举1～10 000之间的所有整数；

2. 根据“如果一个数等于除它本身以外的因子的和，这个数就是完全数”可知，必须找出一个数除本身以外的所有因子；

3. 如果要判断一个数是否为另一个数的因子，那么就要看这个数能否整除另一个数；

4. 让一个数依次除以所有比它小的整数，并把余数为零的除数记录下来，就可以得到这个数除本身以外的所有因子；

5. 获得相应的因子后，把因子之和与这个数相比较，若相等则为完全数。

编程寻找10 000以内的完全数。

1 问题分析。列举10 000以内的整数需要用到一个循环结构，找出一个数的因子需要不断地与比它小的数做除法，也需要一个循环结构，还需要利用分支结构找出因子并判断是不是完全数。

2 代码实现。

```
r=[]                          # 保存找到的完全数
for i in range(1,10001):      # 列举 10000 以内的数
    a = 0                     # 用于累加，i 每次获得一个新值，a 就重新赋值为 0
    for j in range(1,i):      # 列举比 i 小的数，寻找因子
        if i % j ==0 :        # 判断 j 是不是 i 的因子
            a = a + j         # 累加，位于内循环中，前有 3 层缩进
    if a == i :               # 判断是不是完全数
        r.append(i)           # 位于外循环中，前有 2 层缩进

print(r)                      # 显示结果
```

运行结果

```
[6, 28, 496, 8128]
```

继续修改完善填写古文名句的程序，新要求如下：一、每道题有3次回答机会；二、累计有3题答错，则结束程序。

继续完善猜古文名句的游戏。

1 问题分析。需要两个变量来分别记录每一条的答错次数和总的答错题数。每更换一题，答错次数都要重新赋值为0。

2 参照下面的代码继续完善程序。

```
lines=[('天生我材必有用','千金散尽还复来')]
lines.append(('苔花如米小','也学牡丹开'))
lines.append(('七八个星天外','两三点雨山前'))
lines.append(('醉里挑灯看剑','梦回吹角连营'))
lines.append(('八千里路云和月','三十功名尘与土'))
lines.append(('人生若只如初见','何事秋风悲画扇'))
lines.append(('万物得其本者生','百事得其道者成'))
lines.append(('青春虚度无所成','白首衔悲亦何及'))
lines.append(('天下之势不盛则衰','天下之治不进则退'))
guess=''
z=0                          # 答错的题数
for line in lines:
    mz=0                     # 每题回答次数，每次更新一题，mz 就重新赋值为 0
    while mz<3:              # 内层循环
        guess=input(f"{line[0]} 的下一句是：")
        if guess==line[1]:
            print('答对了，请看下一题。')
            break                          # 跳出 while 循环，前面有 3 层缩进
        mz=mz+1                            # 累加每道题的答错次数
    else:                                  # while 分支，不属于内层循环体
        print('本题次数已用完，进入下一题。')
        z=z+1                              # 累计未答出的题数
        if z==3 :                          # 3 题未答出，
            print('很可惜，失败了！')       # 前面有 3 层缩进
            break                          # 跳出 for 循环
else :                                     # for 支，不属于外层循环体
    print('恭喜你，通过了考验！')
```

接下来继续看综合利用基本结构编程解决问题的例子。

若三位数各位上数字的立方和等于它本身，就称它为水仙花数。例如，$153=1^3+5^3+3^3$，它就是一个水仙花数。如何找出所有的水仙花数?

编程寻找水仙花数。

1 问题分析。水仙花数可描述为下面的形式：

$i \times 100 + j \times 10 + k = i^3 + j^3 + k^3$

2 明确解题思路：

① 定义变量i、j和k，代表各数位上的数字；

② i从1开始递增到9，j和k从0开始递增到9；

③ 计算i、j和k的立方和；

④ 把i、j和k组合成一个三位数；

⑤ 如果立方和与组成的三位数相等，则找到了一个水仙花数；

⑥ 继续寻找其他水仙花数，直到所有的三位数都验证完毕。

3 把解题思路用基本结构表达出来。

循环结构：

```
for i in range(1,10):
    for j in range(10):
        for k in range(10):
        ……
```

分支结构：

```
if i*100 + j*10 + k == i**3+j**3+k**3 :
    print (i*100 + j*10 + k)
```

4 程序实现。

```
for i in range(1,10):
    for j in range(10):
        for k in range(10):
            if i*100 + j*10 + k == i**3+j**3+k**3 :
                print (i*100 + j*10 + k)
```

5 运行程序解决问题。

运行结果

```
153
370
371
407
```

如此一来，就找到了所有的水仙花数，共有4个。

类似地，如果一个四位数各位上的数的4次方之和等于它自身，则称为四叶玫瑰数；如果一个五位数各位上的数的5次方之和等于它自身，则称为五角数……如何用一个程序来寻找这些数呢？

首先，要解决“如何得到各位上的数”这一问题。实际上，把数转换成字符串，然后提取字符串不同位置的字符，就可以了。

```
for n in range(100,1000):                    # 列举三位数
    ns = [ int(i) for i in str(n)]           # 取出各位上的数
    s = 0
    for k in ns:                             # 遍历各位上的数字
        s=s+k**3                             # 求立方和
    if s==n:                                 # 如果立方和与数本身相等
        print(n)                             # 显示找到的水仙花数
```

其次，要适应三位数、四位数、五位数等不同情况。编程时，可以通过变量来表示位数。

```
a = int(input(' 输入感兴趣的位数 : '))
for n in range(10**(a-1),10**a):             # 列举数
    ns = [ int(i) for i in str(n)]           # 取出各位上的数
    s = 0
    for k in ns:                             # 遍历各位上的数
        s=s+k**a                             # 有多少位就求多少次方的和
    if s==n:                                 # 如果和与数本身相等
        print(n)                             # 显示找到数
```

运行结果

```
输入感兴趣的位数 : 5
54748
92727
93084
```

最后看一个更为复杂的例子。

相传英国数学家哈代前往探望另一个数学家拉马努金。哈代说：“我坐的士（出租车）来的，车牌号是1729，这数真没趣，希望不是不祥之兆。”拉马努金答道：“不，那是个很有趣的数，1 729可以用两个立方之和来表达而且有两种表达方式。”

后来，人们把能以 n 个不同的方法表示成两个正立方数之和的最小的正整数称为的士数，即哈代–拉马努金数，记作Ta(n)。例如：

$Ta(1) = 2 = 1^3+1^3$

$Ta(2) = 1\ 729 = 1^3+12^3 = 9^3+10^3$

现在已知第三个的士数是87 539 319，那么它可以由哪三种方法表示成两个正立方数之和呢？下面利用Python编程来解决这个问题。

首先，把问题表示为下面的数学形式：

$Ta(3) = m_1^3 + n_1^3 = m_2^3 + n_2^3 = m_3^3 + n_3^3$

接着，明确解题思路：

① 定义变量m和n，让它们从1依次变化；

② 如果m的立方或者n的立方超过了87 539 319，则跳出当前循环；

③ 把m和n的立方和加起来；

④ 如果立方和与87 539 319相等则输出m和n，如果不相等则继续寻找。

然后，把解题思路用基本结构表达出来。

1. 循环结构。

```
m=1                          # m 从 1 开始
while m**3 <= 87539319 :     # m 的立方和是否小于等于 87539319
    ......
    m = m+1                  # m 依次增加
```

2. 循环嵌套。

```
m=1                              # m 从 1 开始
while m**3 <= 87539319 :         # m 的立方和是否小于等于 87539319
   n=1                           # n 从 1 开始，循环嵌套
   while m**3 + n**3 <= 87539319:      # m 和 n 的立方和是否小于等于 87539319
      ……
      n = n+1                    # n 依次累加
   ……
   m = m+1                       # m 依次增加
```

3. 分支结构。

```
      if m**3 + n**3 == 87539319 : print (m,n) ; break    # 输出数对，跳出当前循环
```

转换完成后，就可以写出如下的程序。

```
m=1                              # m 从 1 开始
while m**3 <= 87539319 :         # m 的立方和是否小于等于 87539319
   n=1                           # n 从 1 开始，循环嵌套
   while m**3 + n**3 <= 87539319:      # m 和 n 的立方和是否小于等于 87539319
      if m**3 + n**3 ==87539319:       # 如果相等
         print(m,n); break             # 输出数对，并跳出当前的 while 循环
      n = n+1                    # n 依次增加
   m = m+1                       # m 依次增加
```

运行结果

```
167  436
228  423
255  414
414  255
423  228
436  167
```

程序找到了6个数对，但其中3个数对是重复的，只是前后排列顺序出现了变化。如何防止这种情况发生呢？其实只要保证变量n大于等于变量m可以了，即保证数对后面的数不小于前面的数。

操作时，可以调整程序的第3句：

```
n = m                            # n 从 m 的当前值开始
```

运行修改后的程序，观察运行结果。

运行结果

```
167 436
228 423
255 414
```

进一步分析程序可以发现，程序中求m立方的运算出现了3次，求n立方的运算出现了2次，求两者立方和的运算出现了2次。

```
m=1
while m**3 <= 87539319 :                    # 求 m 的立方
   n=m
   while m**3 + n**3 <= 87539319:           # 求 m 的立方、n 的立方，然后求和
      if m**3 + n**3 ==87539319:            # 求 m 的立方、n 的立方，然后求和
         print(m,n);break
      n = n+1
   m = m+1
```

实际上求m的立方、求n的立方，求立方和等运算只要出现1次就足够了。根据分析，继续优化程序。

```
m=1
while True:
   cm=m**3                                  # 求 m 的立方
   if cm > 87539319 : break
   n=m
   while True:
      cn = n**3                             # 求 n 的立方
      s = cm + cn                           # 求立方和
      if s >= 87539319:
         if s == 87539319: print(m,n)
         break
      n = n+1
   m = m+1
```

修改之后，程序的运行结果没有改变，但求立方、求立方和的次数都减少了。程序中的变量m和n要变换很多次，因而修改后的程序可以减少很多次的求立方运算和求和运算，这样就提高了运行效率。

接下来继续分析，这个程序只能用来寻找特定数字87 539 319的立方和数对，能不能通过适当修改，让程序寻找其他的士数的数对呢？其实只要把程序中的具体数字87 539 319改成变量就可以了。

```
a = int(input(' 输入一个的士数：'))
m=1
while True:
    cm=m**3
    if cm > a : break
    n=m
    while True:
        cn = n**3
        s = cm + cn
        if s > a :
            if s == a: print(m,n)
            break
        n = n+1
    m = m+1
```

运行结果

```
输入的士数：1729
1 12
9 10
```

为了改善程序的输出形式，有同学进行了如下调整，你知道所调整语句的作用吗？

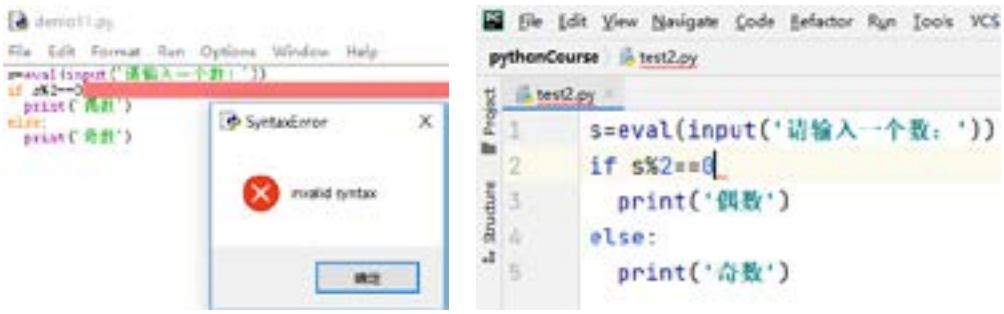
```
r = []
……
while True:
    ……
    while True:
    ……
        if s > a :
            if s == a: r.append((m,n))
    ……
exp = [ f'={ri[0]}**3+{ri[1]}**3 ' for ri in r ]
print( f '{a} {''.join(exp)}'
```

五、错误与调试简介

程序中的错误

编程时，难免会出现各种错误。程序中的错误一般可以分为语法错误、运行错误和逻辑错误三种。

·语法错误，通常指输入的代码不符合语法规定，例如for语句、if语句等后面忘了加冒号，程序中的单引号、双引号、冒号，括号等使用了中文符号，等等。初学者很容易出现这种错误。IDLE会在运行时把出错区域标成红色，而更专业的PyCharm等软件在程序运行前就会提供相应的提示。



·运行错误，一般是指运行时因硬件故障、非正常操作等产生的错误。例如，程序正要读取一个文件，而这个文件被别的软件删除了，程序就会报错。这类错误处理起来非常烦琐。

```
import zipfile
zfile=zipfile.ZipFile('temp.zip', 'w')
zfile.write('001.docx')
zfile.close()
```

运行结果

```
Traceback (most recent call last):
  File "D:/Code/pythonCourse/test2.py", line 5, in <module>
    zfile.write('001.docx')
  File "D:\Python\Python38\lib\zipfile.py", line 1741, in write
    zinfo = ZipInfo.from_file(filename, arcname,
  File "D:\Python\Python38\lib\zipfile.py", line 523, in from_file
    st = os.stat(filename)
FileNotFoundError: [WinError 2] 系统找不到指定的文件。: '001.docx'
```

·逻辑错误，有些程序表面上看起来一切正常，但就是得不到正确的结果，这些程序中往往存在逻辑错误。逻辑错误一般是因人为疏忽或设计失误而产生的，所编的程序越复杂，逻辑错误产生的可能性越大，修改起来也越难。

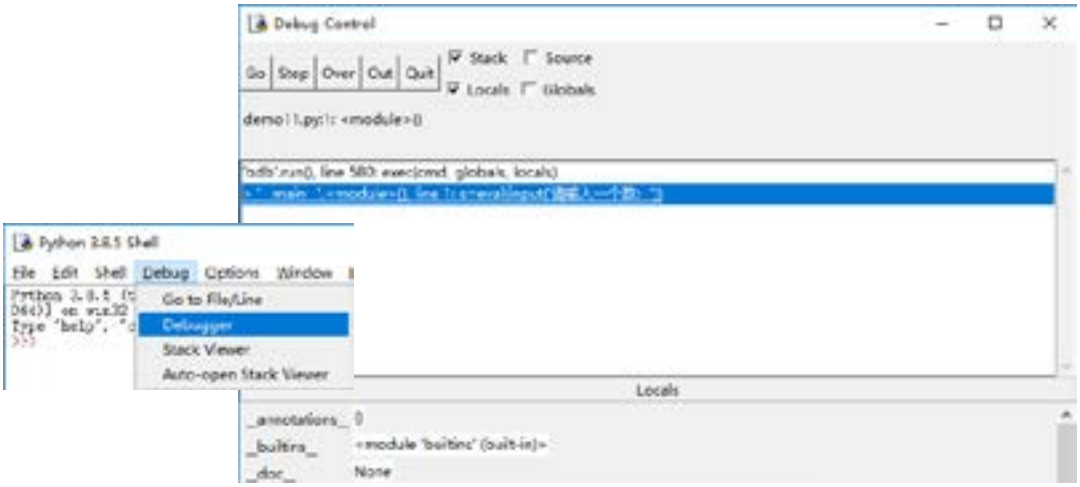
调试程序

在程序设计过程中，试运行编好的程序，发现并纠正其中的错误，或进行适当的调整等操作叫作调试程序。

最简单的调试方法是利用print函数输出所关心变量的值，看看它是否符合设想。如果不符合，说明出错了。实际操作时，可以通过不断移动print在程序中的位置，寻找出错的地方。

```
存在的问题：累加 1 到 1 000 的整数，但 s 的始终值等于 1 000
s=0
for n in range(1,1001):
    s = n
    # 通过 print 语句输出变量 s 的值，从而发现没有进行累加操作
    print(s)
print(s)
```

更专业的方法则是利用编程软件提供的调试工具，通过设置断点等操作，让开发者逐步观察程序的详细运行状态，从而寻找存在的错误。不同的开发软件，调试工具的使用方法有所不同。下面是进入调试状态的IDLE软件显示的信息。



下面是编程时经常会遇到的错误提示，以及相应的修改建议。

出错语句：print(a)
错误提示：NameError: name 'a' is not defined
错误原因：变量 a 没有定义。
修改建议：a=7
print(a)

出错语句：age=18
print(‘我的年龄是’+age)
错误提示：TypeError: can only concatenate str (not "int") to str
错误原因：字符串和整数类型不同，不能进行“+”运算。
修改建议：print(f ' 我的年龄是 {age}')
或 print(‘我的年龄是’+str(age))

出错语句：if m>n:
print(m)
错误提示：IndentationError: expected an indented block
错误原因：缩进有误。
修改建议：if m>n:
print(m)

出错语句：if a >37.3
print(“您已发烧！请及时就诊。”)
错误提示：SyntaxError: invalid syntax
错误原因：语法错误，if 语句后面要带有半角“:”。
修改建议：if a >37.3 :
print(“您已发烧！请及时就诊。”)

看看下面的程序代码存在什么错误，然后与同学交流修改建议。

```
result=0
while m<100:
result=result+m
   if s>1000
      break
   m=m+1
print(resolt)
```

活动总结

1. 参照下表总结自己对三种基本结构的认识。

	顺序结构	分支结构	循环结构
特点	计算机按照代码的排列顺序，从上至下逐条执行语句。每次运行，执行过程不变。		
常用语句	无特定语句。		

2. 总结前面使用过的语句和函数。

条　　目	说　　明	代码样例
continue		
break		
append		
eval		

3. 列出最感兴趣的代码段，并说明原因。

_____

_____

_____

_____

1. 说出下面代码的作用。

```
(1)
n=int(input(' 输入一个自然数：'))
print(' 偶数 ' if n%2==0 else ' 奇数 ')

(2)
[ f'{x}×{y}={x*y}'  for x in range(1,10) for y in range(1,x+1) ]
```

2. 有人说只要对寻找完全数的程序进行简单修改，比如参照下边红色部分的代码进行修改，就可以大大提高寻找的速度。试一试这样做是否真的有效，并说明理由。

```
r=[]
for i in range ( 2,10001) :
    a =1
    for j in range ( 2,int(i/2)+1) :
        if i % j ==0 :
            a = a + j
    if a == i :
        r.append(i)
print(r)
```

3. 厚度是 0.3 毫米的纸对折多少次就会比珠穆朗玛峰还高？编程计算对折次数。

4. 若三位数各位上数字的立方和等于这个三位数，就把这个三位数称为“水仙花数”。如 $153 = 1^3 + 5^3 + 3^3$ 就是一个水仙花数。编程找出所有的水仙花数，并对程序加以说明。

提示：int(f'{i}{j}{k}') == i ** 3 + j ** 3 + k ** 3

交流评价

1. 参照下表，对自己的学习过程作一个评价。

学习内容	掌握程度
交互式编程与文件式编程	□ 不知道 □ 知道 □ 熟悉
常用的数据类型	□ 不知道 □ 知道 □ 熟悉
常用的数据结构	□ 不知道 □ 知道 □ 熟悉
变量、运算符等基本概念	□ 不知道 □ 知道 □ 熟悉
调用函数	□ 不会 □ 会 □ 熟练
编写含有顺序结构的程序	□ 不会 □ 会 □ 熟练
编写含有循环结构的程序	□ 不会 □ 会 □ 熟练
编写含有分支结构的程序	□ 不会 □ 会 □ 熟练

2. 分组交流、评价各种总结表和编写的程序，谈谈自己对Python的认识。

评价内容	评价项目
关于总结表	程序基本结构总结表内容正确、完整。
	函数、语句等总结表内容正确、完整。
	总结时，添加了自己的理解和体会。
关于Python程序	按要求完成了编写Python程序的任务。
	程序能正常运行，具备指定的功能。
	根据自己的需要，增加了新的功能。

3. 组内评选出具有个性的Python程序作品，请设计者向全班介绍编写过程以及所使用的函数、语句等。

本章学习要点

- 函数、模块与库
- 智能编程开发平台
- 智能程序和智能模型
- 人脸识别和手写数字识别
- 感受编程乐趣，增强学习动力

第2章 程序开发初步

掌握了编程语言的基础知识后，就可以开始尝试编程解决问题了。现在使用的程序语言大多会提供功能丰富的函数、编程库等，以方便人们在开发程序的过程中调用。近年来，随着人工智能技术的快速发展，Python等编程语言开始提供用于编写智能程序的库，人们常称之为人工智能开发平台。

这一章就来介绍Python编程时常用的函数、模块和第三方库，特别是用于编写智能程序的编程库。

活动任务

1. 通过编程对比，感受函数的重要作用。
2. 通过一系列编程活动，熟悉常用标准模块的用途。
3. 通过编程解决问题，了解常用第三方库的用途。
4. 通过画风迁移、物体识别等编程活动，感受智能程序。
5. 通过识别人脸、手写数字等活动，理解智能模型的作用。

活动计划

第1节：函数、模块与库

通过完成多个编程任务，积累编程经验，认识更多的函数、模块和编程库，提高编程解决实际问题的能力，培养计算思维。

第2节：基于智能的编程

通过调用已有模型，用智能程序完成画风迁移、物体识别等活动，感受智能程序的魅力。

第3节：训练智能模型

先训练模型，然后利用模型进行预测，从而完成人脸识别、手写数字识别等任务。

活动要求

1. 活动1涉及的编程任务要求个人参照教材独立编写完成，活动2、活动3中的程序只要求理解，不要求自行编程实现。
2. 交流评价活动1中的程序时，能够清晰说明所用的函数、模块或库。
3. 讨论活动2、活动3中较复杂的程序时，只要能明白代码段的作用即可，细节可不作要求。

1. 下面列出了本章需要用的编程库，更多的相关库通常会在安装下面的库时自动安装。

库	简　介	pip安装命令
pypinyin	给汉字标注拼音	pip install pypinyin
pyttsx3	文字转成语音	pip install pyttsx3
qrcode	生成二维码	pip install qrcode
beautifulsoup	处理网页	pip install beautifulsoup4
tensorflow	智能程序开发框架	pip install tensorflow
tensorflow-hub	tensorflow增强包	pip install tensorflow-hub
opencv	计算机视觉处理	pip install opencv-python
opencv-contrib	opencv增强	pip install opencv-contrib-python
numpy	专业的数据分析库	pip install numpy
matplotlib	专业的绘图库	pip install matplotlib
pillow	专注于图像处理	pip install pillow
scipy	专业的科学计算库	pip install scipy

2. 下面列出了本章要用的智能模型。

模　型	简　介
arbitrary-image-stylization-v1-256	绘图风格迁移
mobilenet_v2	识别图中的物体
cat-dog.h5	判断图中是猫还是狗
haarcascade_frontalface_default.xml	识别图中的人脸区域

3. 本章还需要用到名为 mnist.npz 的数据集，里面存放有数万张手写数字的图像。

第1节　函数、模块与库

学习目标

- 认识函数，知道内置函数和自定义函数。
- 熟悉常用的标准模块，会导入模块。
- 了解常用的第三方库。
- 体验编程乐趣，增强编程解决问题的信心。

前面大家学习了程序设计的基础知识，利用Python编写了不少有趣的程序。实际上，Python还有很多有用的函数、很多功能各异的模块和编程库。下面继续学习这些知识，编写功能更强的Python程序。下表是学习过程中要涉及的程序。

程　序	简要说明
fp.py	用于判断素数和寻找素数
gcd.py	计算公约数
findpasswdv2.py	功能更强的寻找密码锁密码
team.py	寻找符合要求的排队方式
gift.py	寻找符合要求的礼物组合
compresses.py	研所和解压缩指定文件
guessnumv2.py	功能更强大的猜数游戏
genqrcode.py	生成二维码
ttt.py	判断井字棋输赢
htmlreader.py	获取网页文本和超链接
simpleidiom.py	简单的补充成语的小游戏
drawfx.py	绘制函数曲线

一、函数

内置函数

前面介绍过，函数可以理解为完成某个功能的代码段，Python提供了很多函数，前面编程时使用的print、input、int、append、range等都是函数。这些函数也被称为Python的内置函数。

合理利用这些函数，可以简化编程过程。比如，计算1+2+3+ ... +100

```
s = 0
for i in range(1,101):
    s=s+i
```

利用函数sum则可以写为：

```
s = sum(range(1,101))
```

显然，用函数计算代码简化了很多。Python提供的函数非常丰富，有些函数加上不同的参数后，会有更强的功能。大家知道int函数可以把字符串转换成整型数据，其实它还可以用来进行数制转换：

```
int('1011',2)   # 把表示二进制数的字符串转换为十进制数
int('13',8)     # 把表示八进制数的字符串转换为十进制数
int('b',16)     # 把表示十六进制数的字符串转换为十进制数
```

同样的，Python也提供了十进制数与其他进制转换的函数：

```
bin(11)      # 十进制数转换成表示二进制数的字符串
oct(11)      # 十进制数转换成表示八进制数的字符串
hex(11)      # 十进制数转换成表示十六进制数的字符串
```

求最大值和最小值，可以分别用max函数和min函数来完成：

```
max(1 , 2 , 7 , 3 )       # 在多个参数中寻找最大值
min(1 , 2 , 7 , 3 )       # 在多个参数中寻找最小值
max([ 1, 2 , 3 , 7 ])     # 在某个序列中寻找值最大的元素
min ( 'abcde' )           # 字符串也可视为序列，从字符串中寻找值最小的字符
```

除了数值计算，Python也提供用于处理列表、元组等的函数，如sorted、zip、enumerate等。合理使用它们，可以快速解决某些问题。

体验几个针对序列的函数。

1 体验排序函数sorted。

```
# 记录数学成绩
>>> list_math=[100.0,98.5,97.5,99.0]
 # 对列表中的元素进行排序，默认升序
>>> sorted(list_math)

[97.5, 98.5, 99.0, 100.0]

# 对列表中的元素进行排序，指定降序
>>> sorted(list_math , reverse=True)

[100.0, 99.0, 98.5, 97.5]
```

2 用zip函数，把序列中相同位置的元素组合成新元素，并形成新序列。

```
>>> list_Chinese=[94.0,95.5,96.0,99.0]
>>> list_English= [96.5,98.0,100.0,95.5]
# 利用三个列表，生成序列
>>> z = zip(list_math,list_Chinese,list_English)
# 遍历 z 中的元素生成新列表，里面的元素是之前三个列表相应位置元素的组合
>>>list_goal=[goal for goal in z]
>>>list_goal

[(100.0, 94.0, 96.5), (98.5, 95.5, 98.0), (97.5, 96.0, 100.0), (99.0, 99.0, 95.5)]
```

3 用enumerate函数，得到元素在序列中的位置和元素的值。

```
# 用 sum 函数计算总分，并形成新的序列
>>> g = [sum(item) for item in list_goal]
# 降序排序
>>> g= sorted ( g , reverse=True)
# 显示名次和分数，注意，Python 中以 0 标识第 1 位
>>> [(index,goal) for index,goal in enumerate(g)]

[(0, 293.5), (1, 293.5), (2, 292.0), (3, 290.5)]
```

列表也有个用于排序的函数sort，它与sorted函数的不同在于：sort函数只能由列表来执行，执行后列表本身会变得有序；而sorted函数可对列表、元组等进行排序，排序结果会形成一个新列表。

下面介绍利用内置函数解决问题的例子。

495是一个非常奇妙的数：任何一个各数位上的数不完全相同的三位数，经过有限次的“重排求差”，最终总会变成495。例如，把592各数位上的数降序和升序排列，可以得到952和259，差等于693；693各数位上的数进行降序和升序排列，可以得到963和369，差等于594；594进行上述操作得到495。

真有这么奇妙吗？下面编程验证一下。

编程验证奇妙的495。

1 问题分析。列举符合条件的三位数，按规则对数位上的数进行降序和升序排列，然后求差；反复进行直到得到495。如果所有符合条件的三位数最后都能推导为495，则495的奇妙性得到验证。

2 代码实现。

```
result=[]                                  # 变量 result 记录得到验证的数
for n in range(100,1000):                  # 列举所有的三位数
    step=0                                 # 变量 step 记录推导次数，
    sn=str(n)                              # 数字转换成字符串
    if len(set(sn))==1:continue            # 如果各位数字相同，则进行下一次循环
    while step<100:                        # 假定 100 步推导不出来，就认为失败
        b = int(''.join(sorted(sn,reverse=True)))    # 降序排列，得到较大的数
        l = int(''.join(sorted(sn)))                 # 升序排列，得到较小的数
        r = b-l                                      # 求差
        if r == 495:                                 # 两者差为 495
            result.append(n)                         # 记录中增加得到验证的数 n
            break                                    # 跳出 while 循环
        sn=str(r)                                    # 把两者之差转成字符串赋给 sn
        if r<100:sn='0'+sn                           # 如果差是两位数，就补个 0
        step=step+1                                  # 推导次数加 1
print(f'共有 {len(result)} 个数字转换成了 495')      # 显示有多少个数转换成功
```

运行结果

```
共有 891 个数字转换成了 495
```

三位数总共有900个，去掉111、222等9个不符合条件的数，还有891个。上面的程序证明，这891个数最终都可以转换成495。从而得出结论：所有满足条件的三位数按规则推导，最终都会转换成495。

自定义函数

顾名思义，就是根据自己的需要编写的函数，比如完成“判断一个数是不是素数”“列出某个范围内的素数”等任务的程序代码经常使用，就可以写成自定义函数。自定义函数的格式如下。

```
def func(paras):                #paras 是参数，可以没有，也可以有多个
    ……
    ……
    return result               # 返回运行结果，可以没有
```

下面编写自定义函数进行素数的相关处理。

编写函数，判断一个数是不是素数。

1 根据自定义函数的格式，编写名为isP的函数，判断传入的参数是不是素数。

```
# 判断数 n 是不是素数，如果是返回 True，如果不是返回 False
# 函数需要一个参数 n
def isP(n):
    for i in range(2,n):
        # 发现因数，返回 False
        if n%i==0:
            return False
    else:
        # 没有发现因数，返回 True
        return True
```

2 编写代码调用函数，判断一个数是不是素数。

```
# 利用 input 函数获取用户输入的字符串，再利用 int 函数转换成整数
n=int(input(' 输入一个大于 2 的整数：'))
# 利用 if 语句，根据 isP 函数的运行结果，选择不同的参数交给 print 函数
print(' 素数 ' if isP(n) else ' 合数 ')
```

运行结果

```
输入一个大于 2 的整数：7
素数
```

3 继续编写函数，通过调用isP函数寻找n以内的所有素数。

```
# 寻找 n 以内的素数
def findPs(n):
    if n<2 : return []              # 参数小于 2，返回空列表
    ps=[2]                          # 加上唯一的偶素数 2
    for i in range(3,n+1,2):        # 依次取从 3 到 n+1 的奇数，不包括 n+1
        if isP(i): ps.append(i)     # 调用自定义函数 isP 进行判断
    return ps
```

4 删除函数之外的代码，然后把程序保存起来，如fp.py。

5 新建一个程序文件，然后参照下面的代码，调用fp.py文件中的函数。

```
import fp                                  # 导入 fp
n = int(input(' 判断输入的整数 n 是不是素数 (n>2)：'))
# 通过 fp.isP 来调用函数
print(' 素数 ' if fp.isP(n) else ' 合数 ')

from fp import findPs                      # 导入具体要用的函数
n = int(input(' 寻找 n 以内的素数，请输入整数 n(n>2)：'))
print(findPs(n))
```

运行结果

```
判断输入的整数 n 是不是素数 (n>2)：91
合数
寻找 n 以内的素数，请输入整数 n(n>2)：100
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83,
89, 97]
```

不难发现，通过这样的方式，可以让已有的代码不断重复使用，避免每次都去写最基础的功能，从而提高编程效率。这种操作体现了程序开发中的“代码复用”思想。

实践探究

有同学用类似下面的代码来判断100~10 000之间的数是不是素数，你觉得对吗？为什么？

```
import fp
ps = fp.findPs(100)
n=int(input(' 输入 100~10000 的整数：'))
for p in ps:
    if n%p==0 : print(' 合数 ');break
else :print(' 素数 ')
```

二、标准模块

模块，可以视为函数的集合。前面编写的含有自定义函数的fp.py文件，就可以看作一个模块，模块名就是文件名（不包括扩展名），使用时用import命令导入就可以了。

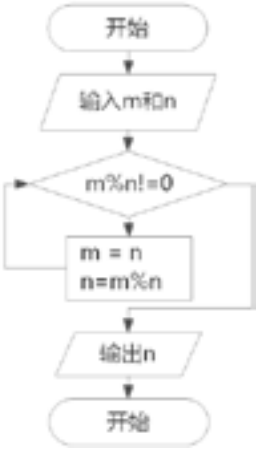
Python语言本身提供了很多编程模块，称之为标准模块。

math模块

math模块主要针对数学运算，其中有各种关于数学运算的函数。下面来对比一下。请先参照下面的代码，感受计算最大公约数的过程。

```
def my_gcd(m,n):
    while m%n!=0:              # 余数不等于 0，进入循环
        m,n = n,m%n            # 把 n 的值赋给 m，把余数值赋给 n
    return n                   # 返回的 n 为最大公约数
```

这个函数计算最大公约数的方法，被称为辗转相除法，它的计算过程可参考下面左图。以34和12为例，具体数值变化可见下面右表。



次数	m	n	m%n
1	34	12	10
2	12	10	2
3	10	2	0

如果每个人在编程计算公约数时，都这样实现一遍显然是种浪费。math模块提供了gcd函数来计算公约数。

```
>>> import math
>>> math.gcd(12,34)
2
```

除了函数，模块也可以提供数学运算使用的常量。比如，math模块就提供了圆周率π。

```
>>> math.pi
3.141592653589793
```

毫无疑问，利用math模块中的函数、常量等进行计算，更加方便、快捷。math模块中常用的函数可见下表。

函数	说　明	举　例
factorial(x)	x的阶乘	factorial(10)，3628800
fabs(x)	x的绝对值	fabs(1)，1 fabs(-1)，1
comb(n, k)	n项中取k项，有多少种取法	comb(10 , 3)，120 comb(7 , 4)，35
lcm(*integers)	多个参数的最小公倍数	lcm(3,7,13)，273 *Python3.9之后才有该函数
sqrt(x)	x的平方根	sqrt(4)，2.0 sqrt(2)，1.4142135623730951

用math模块中的函数解答以下问题：

1. 求34、66、128、132这四个数的最大公约数和最小公倍数。

2. 从5位同学中任意选取3位组队，请问总共有多少种选取方法？

itertools模块

itertools模块可视为一个工具集合，里面包含的常用函数被称为迭代器，大体分为两类：一类用于生成序列，另一类用于解决排列组合问题。

序列迭代器

这类函数有很多，常用的有count、cycle、repeat、chain、accumulate等。下面通过交互命令模式熟悉一下。

体验itertools模块中生成无限和有限序列的函数。

1 体验count函数，它可用于生成等差无限序列。

```
>>> import itertools
# 正奇数无限序列
>>> c=itertools.count(start=1,step=2)

# 需要通过 next 函数让它计算下一个值
>>> next(c),next(c),next(c)
(1, 3, 5)

# 不断计算下一个值，并形成序列
>>> [next(c) for x in range(5)]
[7, 9, 11, 13, 15]
```

2 体验repeat函数，它可以让给定元素无限次地重复出现。

```
>>> r = itertools.repeat(' 多多益善 ')
>>> [next(c) for x in range(5)]
[' 多多益善 ',' 多多益善 ',' 多多益善 ',' 多多益善 ',' 多多益善 ']

# 也可以指定循环次数，形成有限序列
>>> r = itertools.repeat(' 重要事情说 3 遍 ',3)
>>> tuple(r)
(' 重要事情说 3 遍 ',' 重要事情说 3 遍 ',' 重要事情说 3 遍 ')
```

3 体验cycle函数，它可以让给定序列中的元素循环出现，形成无限序列。

```
>>> c=itertools.cycle([1,-1])
>>> [next(c) for x in range(5)]
[1, -1, 1, -1, 1]

>>> c=itertools.cycle([' 从前有座山 ',' 山里有座庙 ',' 庙里有个老和尚说 ' ])
>>> [next(c) for x in range(7)]
[' 从前有座山 ',' 山里有座庙 ',' 庙里有个老和尚说 ',' 从前有座山 ',' 山里有座庙 ','
庙里有个老和尚说 ',' 从前有座山 ']
```

4 体验chain函数，它可以把多个序列串起来。

```
# 字符串可视为一个序列
>>> r = itertools.chain(' 白日依山尽 ',' 黄河入海流 ' , [1,3,5,7] )
>>> list(r)
[' 白 ',' 日 ',' 依 ',' 山 ',' 尽 ',' 黄 ',' 河 ',' 入 ',' 海 ',' 流 ', 1, 3, 5, 7]
```

5 体验accumulate函数，它可以计算某序列前n项的累加值，并形成新序列。

```
>>> a = itertools.accumulate([1,3,5,7,9,11])
>>> list(a)
# 可以发现，根据奇数序列生成的前 n 项累加值序列，正好是平方数序列
[1, 4, 9, 16, 25, 36]
```

排列组合迭代器

这类函数特别适合用来解决排列组合问题。它主要包括4个函数，具体可见下表。

函　　数	举例说明
product(*iterables, repeat=1)	问题：从3个字母中任取2个形成密码组合，求所有可能 代码：product('abc',repeat=2) 结果：aa ab ac ba bb bc ca cb cc
permutations(iterable, r=None)	问题：从3位学生中取2人排队，求所有可能 代码：permutations('abc',2) 结果：ab ac ba bc ca cb
combinations(iterable, r)	问题：从3份礼物中任取2份，求所有可能 代码：combinations('abc',2) 结果：ab ac bc

编程解决密码锁问题。

问题描述：一个4位密码锁，每位可能的值是0~9的整数，用计算机尝试所有的密码。

1 问题分析。4层的循环嵌套自然可以解决这个问题，不过用product函数来生成可能的密码，会更简单。

2 编码实现。

```
import itertools

# 输入一个密码，作为检验条件
kpasswd=input(' 输入一个四位数字密码：')
# 从 0~9 的数字序列中，任取 4 个形成密码组合
passwds=itertools.product(range(10),repeat=4)
```

```
# 列举密码，passwds 是序列迭代器，其中的元素是个元组，包括 4 个元素
for p in passwds:
    # 把密码转换成字符串形式
    gpasswd=f'{p[0]}{p[1]}{p[2]}{p[3]}'
    # 判断是否找到了
    if kpasswd==gpasswd:
        print(f' 找到了，密码是 {gpasswd}')
        break
else :
    print(' 没找到。请检查输入的密码是不是符合要求。')
```

编程解决排队问题。

问题描述：6人站成一排，找出甲、乙既不在排头也不在排尾的组合方法。

1. 问题分析。排队问题，各元素不可能重复，但要考虑它们在队列中的顺序。
2. 参考下面的代码编写程序，并把空白处补充完整。

```
import itertools
# 用 A 表示甲，用 B 表示乙
teams = itertools.permutations('ABCDEF' , _____)
r=[]
for t in teams:
    # 甲、乙是否在队头
    e1 = t [ __ ] in ['A','B']
    # 甲、乙是否在队尾
    e2 = t [-1] in ['A','B']
    # any 函数用于判定序列中的条件，是否有一个成立
    if any([e1,e2]):
        continue
    else :
        r.append(p)
print(f' 共有 {len(r)} 种 ')
```

运行结果

```
共有 288 种
```

这个程序中使用了any函数，它的参数通常是表示条件的序列，当其中有一个条件成立时，any函数的值就为True，全都不成立才为False。

编程解决礼物组合问题。

问题描述：从5件物品中任取3件作为礼物，但不能同时拿走物品A和物品B，找出所有可能的组合。

1 问题分析。礼物组合问题，各元素不可重复，礼物的顺序也无须考虑。

2 参考下面的代码编写程序，并把空白处补充完整。

```
import itertools
cbnt = itertools._____('ABCDE' , ____)
r=[]
for c in cbnt:
    e1 = 'A' in c; e2 = _____
    # all 函数用于判断序列中的条件是否都成立
    if all([e1,e2]) : continue
    else : r.append(p)
print(r)
```

运行结果

```
[('A', 'C', 'D'), ('A', 'C', 'E'), ('A', 'D', 'E'), ('B', 'C', 'D'), ('B', 'C', 'E'), ('B', 'D', 'E'), ('C', 'D', 'E')]
```

这个程序使用了all函数，它的参数通常是表示条件的序列，只有所有条件都成立时，all函数的值才为True，否则为False。

itertools模块中函数的运行结果不是列表、元组等类型，而是称为“迭代器”的对象。利用next函数可以让迭代器对象不断产生序列的下一个元素。想一想：

1. 能把指向无限序列的迭代器对象转换成列表或元组吗？为什么？

2. 使用迭代器时，会根据需要动态计算下一个元素。迭代器与一次性计算出所有元素的列表或元组等相比，两者各有什么优势和不足？

zipfile模块

zipfile模块可用来完成文件的压缩或解压缩等处理任务。

编程体验文件的压缩与解压缩。

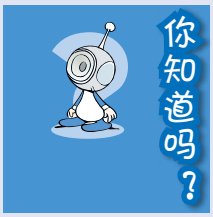
1 参照下面的代码，编写用于压缩文件的程序。注意要根据实际情况，调整压缩文件和被压缩文件的路径。

```
import zipfile
# temp.zip 是要生成的压缩文件，w 表示写数据
zfile=zipfile.ZipFile('cdata/temp.zip', 'w')
 # 把文件 dog.jpg 的内容写入压缩文件
zfile.write('data/dog.jpg')
# 关闭打开的压缩文件。
zfile.close()
```

2 参照下面的代码，编写用于解压缩文件的程序。注意要根据实际情况，调整文件的路径。

```
import zipfile
# temp.zip 是要解开的压缩文件，r 表示读数据
zfile=zipfile.ZipFile('cdata/temp.zip', 'r')
zfile.extractall(path='./edata');
zfile.close()
```

3 利用程序完成压缩和解压缩操作，并尝试与计算机中的压缩软件配合使用，看看程序能不能解开压缩软件生成的ZIP压缩文件，看看压缩软件能不能打开程序生成的压缩文件。



编写程序操作某个文件并写入数据后，应该及时调用close等函数关闭打开的文件。如果不这样做，有可能会导致写入数据的操作失败，也有可能会影响文件在其他地方的正常使用。

fractions模块

这个模块主要用来处理分数运算，比如要计算二分之一和三分之一的和，用普通方式只能得到近似值：

```
>>> 1/2+1/3
0.8333333333333333
```

而利用fractions模块则可以获得精确的分数运算结果。

```
>>> from fractions import Fraction
>>> Fraction(1,2)+Fraction(1,3)
Fraction(5, 6)
```

需要时，也可以通过类型转换，获得相应的数值。

```
>>> float(Fraction(5, 6))
0.8333333333333334
```

可以看到，第二种方式得到的小数与第一种方式的最末一位有些不同，这是因为计算机在进行浮点数（带小数点的数）运算时，总会产生细微的误差，但日常使用时，这些误差基本不会引起问题。如果想精确控制数值的精度，可以使用decimal模块。后面活动用到时再作介绍。

random模块

random模块又称随机模块，主要功能包括随机生成指定范围的数字、随机从序列中取元素等。

利用交互模式熟悉random模块中的函数。

```
>>> import random
# 生成随机数，每次运行的数会不同
>>> random.random(),random.random(),random.random()
(0.6090269089566922, 0.3620563150561946, 0.5442413764359031)

# 生成随机整数 n,1<=n<=10，每次运行得到的数会不同
>>> random.randint(1,10),random.randint(1,10),random.randint(1,10)
(3, 1, 6)

>>> s=' 一二三四五六七八九十 '
# 随机选取序列中的元素，每次运行得到的元素会不同
>>> random.choice(s),random.choice(s),random.choice(s)
(' 四 ',' 三 ',' 四 ')
```

在程序中使用random模块后，可以让程序更富有变化。下面看一个随机生成整数的例子。

改写猜数的小游戏，让计算机随机生成一个整数让人猜。

1 打开猜数小游戏程序。可以发现，等待猜测的数字是固定的，这让程序显得很死板。

```
# 等待猜测的数字
goal=125
while True:
    n=int(input(' 猜一个整数：'))
    ……
```

2 引入random模块，让计算机随机生成要猜测的数。

```
import random
goal = random.randint(1,500)
……
```

3 运行程序，感受引入随机数后程序的变化。

接下来看一个从序列中随机选取元素的例子。

编写成语填空小游戏。

问题描述：从备选的成语中随机选出一个，然后随机替换其中的文字，要求使用者根据其他文字填空，把成语补充完整。

1 问题分析。要随机挑选成语，随机从成语中挑字，这显然要用random模块。

2 导入random模块，并用一个列表存放一些成语。

```
import random
# 备选的成语，越多越好
idioms=[' 平易近人 ',' 宽宏大度 ',' 冰清玉洁 ',' 持之以恒 ',' 锲而不舍 ',
        ' 废寝忘食 ',' 大义凛然 ',' 临危不惧 ',' 光明磊落 ',' 不屈不挠 ',
        ' 鞠躬尽瘁 ',' 死而后已 ',' 高山仰止 ',' 嫉恶如仇 ',' 国泰民安 ']
```

3 编写函数，随机生成一道题。

```
# 随机生成一道题
def genTest():
    # 随机选成语
    idiom = random.choice(idioms)
    # 从成语中随机选字
    kw = random.choice(idiom)
    # 把选出来的字用下画线替换掉，规定只替换一次
    pidiom = idiom.replace(kw, '__', 1)
    # 返回值是一个元组，里面是三个元素的值
    return idiom,kw,pidiom
```

4 根据下面的代码，编写游戏的其他部分。

```
# 分数
goal=0
# 没达到 60 分，继续测试
while goal<60:
    # 随机生成一道题
    idiom, kw, pidiom = genTest()
    # 重置每道题的猜测次数
    count=0
    # 每道题猜 3 次，超过就换下一题
    while count<3:
        gw = input(f' 请把 “ {pidiom} ” 补充完整：')
        if gw == kw:
            goal = goal + 10
            print(f' 太棒了，当前分数：{goal} ！ 成语是：{idiom}')
            break
        count=count+1
    else:
        print(f' 太可惜了！ 成语是：{idiom}')
else:
    print(' 顺利闯关！ ')
```

参考前面使用 random 模块的程序代码，修改之前编写的补充古文名句的程序，要求如下：

1. 随机从备选的古文名句中选择一条；
2. 随机选取前半句或后半句生成题目；
3. 随机遮盖显示的那半句中的一个字。

os模块

os模块是一个与操作系统相关的模块，最常用的操作是利用这个模块来处理计算机中的文件。

列出一个文件夹（及子文件夹）中指定格式的文件。

1 参照下面的代码进行编程。

```
import os
def list_file(path, filetypes):   # 列出指定格式的文件
    result = []
    #os.walk 函数用于生成文件序列，文件序列格式类似
    #[([home1],[folder1],[files1]),([home2],[folder2],[files2])
    # 其中 folder 为 home 中的子文件夹序列
    for home, folder, files in os.walk(path):
        for f in files:
            suffix = f.split('.')[-1]            # 获取文件扩展名
            suffix = suffix.upper()              # 转换成大写字符串
            if suffix in filetypes:              # 判断是不是指定的类型
                result.append(f'{home}/{f}')
    return result
rfs = list_file('d:/test' , ['JPG','PNG','JPEG','TIF'] )
print(rfs)
```

2 运行程序，观察显示效果。

运行结果

```
['d:/test/001.png', 'd:/test/002-1.jpg', 'd:/test/002-2.jpg', 'd:/test/002.jpg', 'd:/
test\\pics/4a-04-01.tif', 'd:/test\\pics/4a-04-02.tif', 'd:/test\\pics/4a-04-03.tif', 'd:/test\\
pics/4a-04-04.tif', 'd:/test\\pics/4a-04-05.tif', 'd:/test\\pics/4a-04-06.tif']
```

os模块中还有很多有用的函数，请同学自行查阅下列函数的说明，并尝试使用。

os.mkdir　os.remove　os.rmdir　os.listdir
os.path.isfile　os.path.isdir　os.path.exists
os.paht.split　os.path.splittext　os.system

re模块

re模块负责处理如何用正则表达式来处理文本。正则表达式，又称规则表达式，通常用来检索、替换符合某个模式的文本。

在实际使用时，根据问题自行设计一个正则表达式比较困难，需要经过长时间的学习和实践才能掌握，但如果知道了某个正则表达式，用它来处理文本则非常简单。下面给几个案例，体验一下。

编程找出文本中的所有英文单词。

1 问题分析：英文单词由大小写字母组成，字母可能出现多次，据此可以使用这个正则表达式：

```
# [a-zA-Z] 表示任意大小写英文字母，
# + 表示之前字符出现了任意多次（0 次以上）
p= '[a-zA-Z]+'
```

2 根据下面的代码编程，找出文本中的所有英文单词。

```
import re

# 文本，注意单引号和双引号
s=' 正则表达式这个概念最初是由 Unix 中的工具软件普及开的。正则表达式通常缩写成“regex”，单数形式有：regexp、regex 等，复数形式有：regexps、regexes、regexen 等。'

p= '[a-zA-Z]+'                  # 表示正则表达式的字符串
rp = re.compile(p)              # 根据字符串，编译正则表达式对象 rp
r=rp.findall(s)                 # 利用 rp 在 s 中进行寻找
print(r)                        # 输出寻找结果
```

运行结果

```
['Unix', 'regex', 'regexp', 'regex', 'regexps', 'regexes', 'regexen']
```

下面再看一个使用正则表达式判断输入的密码是否满足要求的例子。

编程判断输入的密码是否满足要求。

1 问题分析：一个合适的密码通常含有数字、大小写字母和符号等，且有一定的长度。编程时，可以一步步完善正则表达式。

2 假设密码在ATM机上使用，它只含有数字且要有6~8位，可以通过下面的程序进行判断。

```
import re

# ^ 表示起始，$ 表示结束 ,[0-9] 表示元素是 0~9 之间的数
# {6,8} 表示之前的元素出现 6~8 次
p = '^[0-9]{6,8}$'                                    # 正则表达式

pattern = re.compile(p)
s = input(' 输入只由数字组成的密码（6~8 位）：')
if (pattern.fullmatch(s)): print(' 密码符合要求 ')          # 根据表达式进行判断
else : print(' 密码不符合要求 ')
```

3 假设对密码的要求升级了，要求密码可以含有数字和大小写字母，这时只要修改正则表达式就可以了。

```
# ^ 表示起始，$ 表示结束
# [0-9a-zA-Z] 表示数字和大小写字母
# {6,8} 表示之前的元素出现 6~8 次
p = '^[0-9a-zA-Z]{6,8}$'
```

4 现在对密码的要求再次升级，要求密码必须同时包含数字和大小写字母，这种情况下可以使用下面的正则表达式。

```
# ^ 表示起始，$ 表示结束，
# [0-9a-zA-Z] 表示元素包括数字和大小写字母
# (?=.*[0-9]) 表示需要有数字
# (?=.*[a-z]) 表示需要有小写字母
# (?=.*[A-Z]) 表示需要有大写字母
# {6,8} 表示之前的元素出现 6~8 次
p = '^(?=.*[0-9])(?=.*[a-z])(?=.*[A-Z])[0-9a-zA-Z]{6,8}$'
```

可以发现，随着要求的增加，正则表达式的形式也变得越来越复杂，但程序修改起来其实很容易，只要修改表示正则表达式的字符串就可以了。

实践探究

在实际使用时，除了数字和字母，通常还会要求密码中要含有特殊符号。这时可以用下面的正则表达式来进行编程。

```
# [0-9a-zA-Z_#!~] 表示可能是数字、字母或 _、#、!、~ 等特殊符号
# (?=.*[_#!~]) 表示需要有特殊符号，特殊符号可能是 _、#、!、~
p = '^(?=.*[0-9])(?=.*[a-z])(?=.*[A-Z])(?=.*[_#!~])[0-9a-zA-Z_#!~]{6,8}$'
```

三、第三方库

库，可以视为模块的集合，Python的标准模块形成了标准库，它会跟随Python语言环境同步安装到计算机中。此外，很多人还根据不同的任务需求，编写了功能各异的模块，形成了Python官方之外的编程库，这称为第三方库。前面编程使用过pygame、pypinyin、pyttsx3等都属于第三方库。

第三方库使用前，必须先安装到所用的计算机上。安装方法有很多，如利用pip命令安装等。

```
pip install pygame
pip install pypinyin
pip install pyttsx3
```

安装了第三方库后，就可以导入相应的模块，利用其中的函数进行编程了。第三方库的功能非常丰富，下面介绍几个常用的。

生成二维码的库——qrcode

qrcode库专门用于生成二维码，利用这个库中的模块，可以方便地把指定的文本，如文字、网址等转换成二维码。

利用qrcode中的函数生成二维码。

1. 参考下面的代码编程，把某个字符串转换成二维码。

```
import qrcode
s=' 书山有路勤为径，学海无涯苦作舟。'
# 根据字符串生成二维码
img=qrcode.make(s)
# 把二维码保存到文件中。
img.save('qr001.png')
```

2. 打开生成的二维码文件，然后用手机等设备扫描二维码，观察扫描后获得的信息。

3. 修改程序，尝试把一个网址转换成二维码。

专注网页内容处理——beautifulsoup

利用beautifulsoup中的函数，可以解析HTML文件。

把网页中的文字和超链接都列出来。

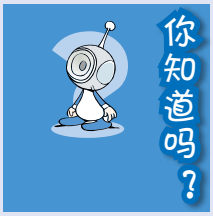
参考下面的代码编程。

```
# 导入 urllib.request 模块，这是个标准模块
import urllib.request
# 第三方库 beautifulsoup 的模块名是 bs4
import bs4

# 访问指定的网址
f = urllib.request.urlopen('http://xxx.xxx.xxx')
# 获取网页对应的 HTML 文件
html_doc=f.read().decode('utf-8')
# 对 HTML 进行解析
soup = bs4.BeautifulSoup(html_doc, 'html.parser')
# 从 HTML 中提取文字
print(soup.get_text())

# 从 HTML 中提取所有的 a 标签
for link in soup.find_all('a'):
    # 从 a 标签中提取 href 属性，即其中的超链接
    print(link.get('href'))
```

在这个程序中，先使用了标准模块urllib.request访问网址，获取网页对应的HTML数据，然后利用第三方库beautifulsoup的bs4模块，并利用它对HTML数据进行解析处理，包括获取文本、获取超链接等。



库的名字与模块的名字并不总是相同，比如前面使用的第三方库beautifulsoup，编程时导入的模块是bs4。

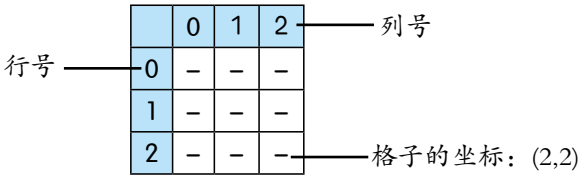
一个模块可能包含多个子模块，比如urllib就包括urllib.request、urllib.parse、urllib.error等模块，这些模块以类似“文件夹.子文件夹”的形式进行管理。人们把这种管理方式称作包管理，因此urllib也可以被视为含有多个子模块的包。

面向矩阵运算——numpy

很多人认为，第三方库numpy是Python语言数据分析的基石，是数据分析领域无冕的“官方”库。很多第三方库，包括后面要介绍的与人工智能技术相关的库，都要调用numpy进行复杂的数值运算。这个库包含的内容非常多，下面通过一个例子简单介绍一下。

用程序判断井字棋棋局的输赢。

1 井字棋棋盘通常是3×3的规格，因而可以用3×3的二维的数组，即二维矩阵来表示棋盘上的落子点。




```
# 引入 numpy 模块，为了简化书写，后面用 np 来指代这个模块
import numpy as np
GCS = 'XO'              # 选手的落子符号
# 根据序列生成一个 3*3 的矩阵，矩阵各个元素都是字符串
puz=np.zeros((3,3), dtype=np.str)
# 用 '-' 填充整个矩阵
puz[:]='-'
```

2 根据棋谱，在棋盘上落子。

```
# 标识当前选手，0 表示第 1 位选手，1 表示第 2 位选手
right=0
# 落子顺序，记录的是坐标，由两个人交替进行
steps=[(1,1),(0,0),(0,1),(2,1),(1,2),(2,0),(1,0)]
for step in steps:
    x,y=step      # 从步骤中获取坐标
    # 根据坐标，给矩阵中某个元素赋值，值为当前选手的落子符
    puz[x,y]=GCS[right]
    # 通过异或运算 ^，在 0 和 1 之间交替更新，即表示不断切换选手
    right=right^1
# 展示井字棋对弈结果
print(puz)
```

3 编程检查输赢。根据井字棋规则，要检查行、列和对角线，看是否有选手形成了3子连接的现象。对弈结果已经保存到了矩阵puz中，只要参考下面的代码，就可以获得行、列和对角线。

```
# 按矩阵行获取元素
rows=[puz[0,:],puz[1,:],puz[2,:]]
# 按矩阵列获取元素
cols=[puz[:,0],puz[:,1],puz[:,2]]
# 按对角线和反向对角线获取元素
ds=[puz.diagonal(),np.flip(puz).diagonal()]

for line in rows+cols+ds:
    # 统计选手落子符
    for right in [0,1]:
        if sum(line==GCS[right])==3:
            print(f'{right} 号选手获胜！ ')
            break;
```

运行结果

```
[['O' 'X' '-']
 ['X' 'X' 'X']
 ['O' 'O' '-']]
0 号选手获胜！
```

上面只是简单介绍了如何用numpy模块对二维矩阵进行简单操作，包括填充整个矩阵，获取行、列、对角线，对矩阵某个位置的元素进行赋值等，其实这个模块的功能远不止这些，而且numpy中还有numpy.random等模块，对数据分析感兴趣的同学可以自行研究摸索。

专注绘图——matplotlib

利用matplotlib库，可以方便地绘制数学函数曲线，或制作折线图、饼图等各种统计图表。它经常跟numpy等库一起使用。

绘制函数曲线。

1 利用numpy生成函数的自变量和应变量序列，然后绘图。

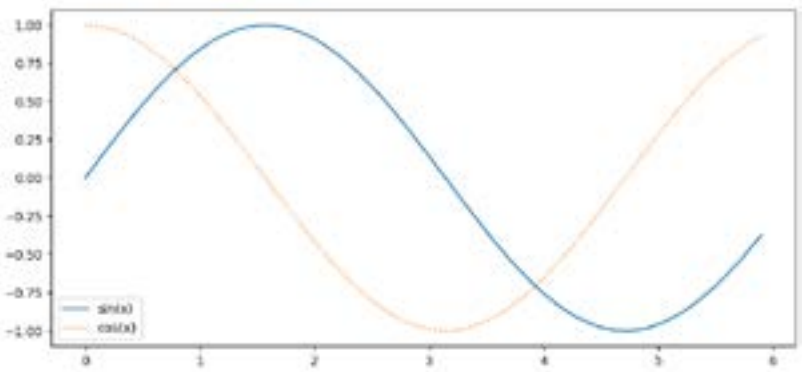
```
# 导入 numpy，用来完成计算
import numpy as np
```

```
# 导入 matplotlib 的 pyplot 模块，用来作图
import matplotlib.pyplot as plt

# 生成等差浮点数序列，用作自变量，range 函数无法完成这个任务
x=np.arange(0,6,0.1)
# 调用 sin、cos 函数，根据 x，计算 y1 和 y2 序列
y1 = np.sin(x)
y2 = np.cos(x)

# 根据序列 x 和序列 y1 绘图，标签是 sin(x)，默认实线
plt.plot(x,y1,label='sin(x)')
# 根据序列 x 和序列 y2 绘图，标签是 cos(x)，指定用点线
plt.plot(x,y2,linestyle='dotted',label='cos(x)')
# 设定图例，即各曲线的标识
plt.legend()
# 显示图
plt.show()
```

2 运行程序，观察绘制效果。



图像处理利器——pillow

pilow是专门用于处理图像的库，利用它可以完成转换格式、缩放图像、抽取轮廓，强化图像等多种处理。

生成网站常用的验证码。

1 参考下面的程序，随机生成一个字符串，然后把字符串放到随机生成背景中，形成登录网站时常用的验证码。

```
from PIL import Image, ImageDraw, ImageFont, ImageFilter
import random

def rndChar():                                  # 随机生成数字，然后转换成 ASCII 字母
    return  chr(random.randint(65, 90))
def rndColor(m,n):                              # 随机生成颜色
    return tuple(random.randint(m,n) for i in range(3))

image = Image.new('RGB', (240, 60), (255, 255, 255))     # 白色背景的图
font = ImageFont.truetype('arial.ttf', 36)               # 所用字体
draw = ImageDraw.Draw(image)
for x in range(240):                                     # 绘制随机颜色点
    for y in range(60):
        draw.point((x, y), fill=rndColor(64,255))
for t in range(5):                                       # 输出文字
    draw.text((40*t+20, 10), rndChar(), font=font, fill=rndColor(32,128))

image = image.filter(ImageFilter.BLUR)                   # 调用滤镜，实现模糊化
image.show()
```

2 运行程序，观察生成的图像。

Python的编程模块极其丰富，甚至有人戏称，所有的计算机处理任务都能找到对应的Python模块来进行处理。大家在编程处理信息时，最好先搜索一下，看看有没有合适的模块可以使用。

思考与练习

1. 你认为Python中的函数、模块、库之间有什么关系？如何使用模块中的函数？

2. 编写程序，让计算机随机生成两个分数，然后由用户输入两个分数之和的最简方式，最后由计算机判断用户的输入是否正确。

3. 试用numpy和matplotlib库绘制二次函数的曲线。

4. jieba是文字处理过程中经常使用的库，查阅相关资料，然后尝试用jieba对文本进行分词处理。

第2节 基于智能的编程

学习目标

- 了解画风迁移问题，用程序实现画风迁移。
- 会编写程序自动识别图像中的物体。
- 掌握用程序分辨图中猫狗的实现方法。
- 感受智能程序的魅力，提高学习兴趣。

近年来，随着机器学习、大数据等技术的发展，人工智能技术也得到了新的发展。为了方便人们根据自己的需要编写、开发各种智能程序，研究人员开发出了PyTorch、Tensorflow等编程库。下面以Tensorflow为例，体验几种智能程序的开发过程。

一、画风迁移

不同的创作形式，其作品通常具有不同的的风格。





让计算机学习某种风格的艺术作品，然后就可以让它依据那种风格，改编其他图像了。

下面利用已训练好的模型编写具有类似功能的程序。编程之前，至少需要准备两张图：一张是代表了某种艺术风格的图像、一张是准备修改的图像。此外，还要准备存放有模型的文件夹arbitrary-image-stylization-v1-256。

实现画风迁移的程序。

1 新建一个文件夹，把图像和模型保存到里面。

2 把程序文件style_artist.py也复制进去，然后根据注释了解各段的功能。

```
import matplotlib.pylab as plt
import numpy as np
import tensorflow as tf
import tensorflow_hub as hub

# 加载模块
hub_module = hub.load('arbitrary-image-stylization-v1-256')
```

```
# 加载要学习风格的图像
style_image = plt.imread('type/type01.jpg')
# 进行必要的数据处理
style_image = style_image.astype(np.float32)[np.newaxis, ...] / 255
style_image = tf.image.resize(style_image, (256, 256))

# 加载要修改的图像
content_image = plt.imread('img/001.jpg')
content_image = content_image.astype(np.float32)[np.newaxis, ...] / 255

# 开始风格迁移
outputs = hub_module(tf.constant(content_image), tf.constant(style_image))
stylized_image = outputs[0]

# 显示修改后的效果
plt.imshow(stylized_image[0])
plt.axis('off')
plt.show()
```

3 修改程序，选择一图用于学习风格，另一图用于迁移修改。然后运行程序，观察运行后的效果。

实践探究

用下面的语句可以保存生成的图像。修改程序，让计算机学习了某种风格后，自动把某个文件夹中的所有图像都改成那种风格的图像。

```
plt.save('123.jpg')
```

二、识人识物

通过智能程序，可以让计算机自动识别图像中的人、物体等。

与前面类似，下面利用一个已经训练好的模型来编写具有类似功能的程序。活动前，需要准备存放模型的文件夹mobilenet_v2和几张用于识别测试的图像。

感受可以从图像中识别人和物的程序。

1 新建一个文件夹，把图像和模型保存到里面。

2 复制并打开名为master.py的程序文件，根据其中的注释，了解各段的功能。

```
import cv2
from random import randint
import tensorflow as tf
import tensorflow_hub as hub

img_file='img/001.jpg'                    # 指定要识别的图像

# tf 加载图像并进行必要的预处理
img = tf.io.read_file(img_file)
img = tf.image.decode_jpeg(img, channels=3)
c_img = tf.image.convert_image_dtype(img, tf.float32)[tf.newaxis, ...]

# 加载模型
detector = hub.load('./mobilenet_v2').signatures['default']
```

```
# 检测图像，得到对象名称、评估分数和区域范围等
result = detector(c_img)
names = result["detection_class_entities"].numpy()
scores = result["detection_scores"].numpy()
boxs = result["detection_boxes"].numpy()

#opencv 加载图像用于显示
cimg = cv2.imread(img_file)
im_h, im_w = cimg.shape[:2]
if im_h > 800:
    im_w=int(im_w/(im_h/800));im_h=800
    cimg = cv2.resize(cimg, (im_w, im_h))

index=0                                  # 变量 index 用于记录标出的对象数量
font=cv2.FONT_HERSHEY_PLAIN  # 变量 font 对应文字的字体
for score,name,box in zip(scores,names,boxs):
    # 评估分数超过 0.1，标记的对象数小于 5
    if score >0.1 and index < 5:
        # 把检测结果中的对象区域坐标转换成图像中的坐标
        left, right = int(box[1] * im_w), int(box[3] * im_w),
        top, bottom = int(box[0] * im_h), int(box[2] * im_h)

        # 随机生成一种颜色，然后画矩形框
        color = (randint(0, 255), randint(0, 255), randint(0, 255))
        obj_rect = left, top, right - left, bottom - top
        cv2.rectangle(cimg, obj_rect, color, 2)

        # 把检测的名字和评分转换成要显示的字符，并显示
        dstr = f'{name.decode("ascii")} : {int(score * 100)}%'
        cv2.putText(cimg,dstr,(left,top),font, 1, (255, 0, 255), 2)
        index = index + 1

cv2.imshow("image",cimg)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

3 运行程序，观察程序的识别效果。

尽管当前计算机图像识别技术已经有了长足的进步，但在实际应用中，仍然会有很多错漏。比如，前面使用的模型就无法识别火箭等物体，还可能把人或物认错。但是相信随着技术的进一步提升，这种错漏会越来越少。

三、猫狗分类

通过编写智能程序，也能让计算机对某些照片进行分类，比如对猫和狗的照片进行分类。

同样的，这个活动也要调用预先已经训练好的模型进行编程。活动前，应当先准备好要用的模型文件cat-dog.h5，并找几张猫和狗的图像文件。

用程序区分猫和狗的图像。

1 新建一个文件夹，把图像和模型保存到里面。其中猫和狗的图像文件建议保存到test子文件夹中。

2 复制并打开名为cat_or_dog.py的程序文件，了解各段的功能。

```
import cv2
import numpy as np
from tensorflow.keras import models
from keras_preprocessing import image

# 加载模型
model=models.load_model('cat-dog.h5')
# 显示模型信息
model.summary()

fn=f'test/lele03.jpg'                    # 指定用于检测的文件
# 用 tensorflow 中的 image 加载文件，并进行必要的数据处理
img = image.load_img(fn, target_size=(150, 150))
```

```
data = image.img_to_array(img)
data = np.expand_dims(data, axis=0)/ 255.

# 利用模型进行预测，即判断，结果为 0 表示猫，1 表示狗
p = model.predict(data)
p = int(p[0])

# 用 opencv 加载文件用于显示
cimg = cv2.imread(fn)
dstr=f'{["CAT","DOG"][p]}'                    # 把结果转换成 CAT 或 DOG
font=cv2.FONT_HERSHEY_PLAIN
cv2.putText(cimg,dstr,(200,200),font,5, (255, 255, 0), 4)
cv2.imshow("image",cimg)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

3 修改程序代码，指定要检测文件，然后运行程序，观察检测结果。

在运行程序的过程中，计算机会显示类似下面的信息。

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
resnet50v2 (Functional)      (None, 5, 5, 2048)        23564800
_________________________________________________________________
flatten (Flatten)            (None, 51200)             0
_________________________________________________________________
dense (Dense)                (None, 1)                 51201
=================================================================
Total params: 23,616,001
Trainable params: 51,201
Non-trainable params: 23,564,800
```

这个信息展示了所用模型的内部结构，总体上看，它有3层，第一层最为重要，是计算机进行深度学习时经常使用的resnet50网络，即残差网络。图像特征识别等主要功能都由这个网络完成，总计有2 300多万个参数。虽然在这里看起来只是一层，但实际上它可以视为一个含有50层结构的超级网络。具体的工作原理非常复杂。

后面两层flatten和dense相对简单，主要用于处理前一层的运算结果。

计算机进行学习的过程，可以理解为“训练”模型的过程。如果学习使用的模型层很多、很复杂，就认为计算机在进行深度学习。深度学习的成果，往往能让计算机具备非凡的能力，但学习过程通常需要大量的数据并需要进行长时间的运算，因而普通的计算机并不适合。

对于普通人来说，可以优先选择已经训练好的模型进行智能编程。如果现有的模型都不适用，且模型的层比较简单时，也可以自行训练模型。后面的活动将介绍如何训练简单的模型。

实践探究

当前，猫和狗的图像文件混杂在test文件夹中。请完善下面的程序，控制计算机自动把猫和狗的图像分别存放到cats和dogs文件夹中。

```
import os,shutil
import numpy as np
from tensorflow.keras import models
from keras_preprocessing import image
model=models._____
base_dir='test/'
dn_dir = ['cats/','dogs/']
for file in os.listdir(base_dir):
    fn=f'{base_dir}{file}'
    if os.path.isdir(fn):continue
    img = image.load_img(fn, target_size=(150, 150))
    x = image.img_to_array(img)
    x = np.expand_dims(x, axis=0)/ 255.
    p = _____
    p = int(p[0])
    shutil.move(fn, f'{base_dir}{dn_dir[p]}{file}')
```

思考与练习

1. 根据你现在的体验，谈一谈在具备某些智能的程序中，模型能起什么作用？

2. 查找PyTorch、CNTK等可用于智能编程的第三方库，然后尝试运行一两个例子。

第3节　训练智能模型

学习目标

- 了解人脸识别应用，熟悉人脸识别的过程。
- 了解准备训练数据的方法。
- 能够利用已有的人脸数据训练新的识别模型。
- 能够自行训练模型识别手写的数字图像。

在前面的学习过程中，都是利用他人训练好的模型开发智能程序。那么，这些模型到底是如何产生的呢？如何根据问题自行生成需要的模型？训练模型的过程比较复杂，需要大量的数据和运算资源，普通人一般难以完成这种任务。不过，对于人脸识别、手写数字识别等较简单的问题，可以根据需要自行训练。

一、人脸识别

人脸识别技术目前已经在社会中得到了广泛应用。比如，可以利用人脸识别实现门禁管理，可以用人脸识别技术建立平台寻找失散的儿童，还可以用人脸识别来支付货款等。

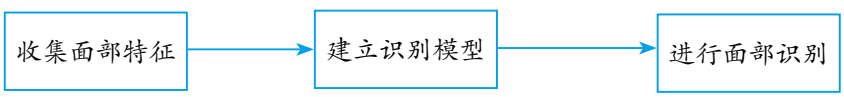
下面就来学习如何编程实现人脸识别。

人脸识别过程

如何进行人脸识别呢？我们先来看看人是怎么识别其他人的。

遇见了一个不认识的人或者看一个陌生人的照片时，会不自觉地观察那人的脸、眼睛、鼻子、嘴巴……这是在收集那个人面部的特征。再次遇到或看到照片时，尽管场景、服饰等可能都发生了很大的变化，但人的大脑还是会立刻认出来。

这就是人识别他人的过程，具体可以分为三步：一、观察某人的面部，获取相关的特征信息；二、根据这些信息在头脑中建立相应的模型；三、根据建立的模型，识别出特定的人。



计算机进行人脸识别时，过程也与此相同。

准备数据

要想获取某个人的面部特征，首先需要把这个人的面部“找”出来，人很容易做到，但计算机就有点困难。当然，可以用人工的方式把照片中的人脸部分截取出来，但照片数量多了以后，这样做显然就不合适了。

实际上可以使用一些编程工具来辅助完成，如专门用于处理计算机视觉问题的OpenCV库。这个库提供了一个现成的模型，可以把照片中的人脸单独截取出来。进行编程时，建议把程序所需要的模型文件保存到程序文件所在的文件夹里。

了解检测面部的函数和测试程序。

1. 准备一两张用来测试的图像文件，图像中要包括一张人脸。
2. 编写寻找人脸的函数和测试程序。

```
# 引入编程模块，opencv 对应 cv2 模块
import cv2

# 检测面部的函数，参数 img 为图像数据
def detect_face(classifier , imgfile):
    img = cv2.imread(imgfile,cv2.IMREAD_GRAYSCALE) # 灰度图方式读取图像
    faces = classifier.detectMultiScale(img)                    # 检测图像
    # 返回检测到的面部数据
    x,y,w,h=faces[0]
    return img[y:y+h, x:x+w]

# 加载已有模型
classifier = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
face=detect_face(classifier , '001.jpg')
cv2.imshow("image",face) # 显示图像
cv2.waitKey(0)
cv2.destroyAllWindows()
```

3. 把模型文件haarcascade_frontalface_default.xml保存到程序所在的文件夹中。这个文件通常在OpenCV的安装文件夹里。

利用这个方法，可以自动截取图像中的人脸部分。接下来就可以用它准备人脸识别的训练数据了。

利用程序自动截取素材中的人脸数据。

1 建立名为“pics”的文件夹，把几个人的独照分别存入不同的子文件夹，如p01、p02、p03、p04。照片最好准备多张，人的面部要正对镜头，背景越淡越好。一般来说，符合标准的照片越多，收集到的数据越丰富，训练出来的模型质量就会越高。

2 根据前面所学，理解下面的程序。其中，程序使用了os模块的相关函数，来列出某个文件夹中的所有文件。

```
import os , cv2
def detect_face(classifier , imgfile):
    img = cv2.imread(imgfile,cv2.IMREAD_GRAYSCALE)
    faces = classifier.detectMultiScale(img)
    x,y,w,h=faces[0]
    return img[y:y+h, x:x+w]

faces,labels=[],[]   # faces 用来存放面部数据，label 用来做标记，区分是哪个人
classifier = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
for p,m in zip (['p01','p02','p03','p04'],[0,1,2,3]) :          #p01 标记为 1，以此类推
    for img_file in os.listdir(f'pics/{p}'):
        face=detect_face(classifier ,f'pics/{p}/{img_file}')
        faces.append(face); labels.append(m)
for face in faces:
    cv2.imshow("image",face) # 显示图像
    cv2.waitKey(1000)          # 显示 1000 毫秒，即 1 秒后自动关闭
    cv2.destroyAllWindows()
```

3 运行程序，可以发现，程序可以把每张照片中的人脸都识别出来，并会把相关的数据保存到faces列表中。




训练模型

接下来，我们将根据前面生成的列表中的数据，来训练一个识别特定人员的模型。

训练识别模型。

1 参照下面的代码，编写生成识别模型的函数。

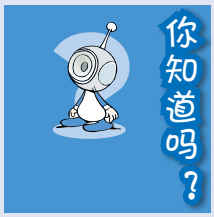
```
# 生成识别模型
def gen_recognizer(faces,labels):
    # 调用 OpenCV 提供的特定的识别训练器
    recognizer = cv2.face.LBPHFaceRecognizer_create()
    # 利用面部数据和标记，训练模型
    recognizer.train(faces, np.array(labels))
    # 保存模型
    recognizer.save('face_mod.xml')
```

2 在主程序的最后插入下面的代码，完成训练程序的编写工作。

```
gen_recognizer(faces,labels)
```

3 运行程序，等图像都展示完毕后，文件夹中就会出现名为face_mod.xml的文件，这个文件就是训练得到的模型。

接下来就可以利用这个模型进行人脸识别了。



1. 在检测、识别的过程中，通常需要把图像都转换成灰度图的形式。

2. 原始图像可能大小不一，检测到的人脸区域也可能大小不一。为了提高训练质量，可以考虑用下面的代码把图像数据强制转换成相同大小。

```
cv2.resize(img[y:y+h, x:x+w], (128,128))
```

3. 原始图像是否符合规范，会影响模型的有效性。因此训练时需要观察程序的截取效果。如果发现在某些原始图像上不能很好地截取人脸，可以考虑删掉那些图像。

用模型进行人脸识别

对照片进行人脸识别，其实包括两个步骤：一、找出照片中的人脸区域；二、利用模型对人脸区域进行识别。识别过程其实就是预测过程。

编程进行人脸识别。

1 再准备一些用来进行识别的照片，并把这些照片杂乱地放在一个文件夹中，如pics的test子文件夹。

2 参照下面的代码进行编程，调用已有的模型进行人脸识别操作。

```
import os,cv2
names=['yingzi','xiao xue ','hehe','nuan nuan']        # 不同标记对应的不同人员

# 加载已有模型，用于检测人脸区域
classifier = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
recognizer = cv2.face.LBPHFaceRecognizer_create()
# 加载生成的模型，用于识别人脸
recognizer.read('face_mod.xml')

font=cv2.FONT_HERSHEY_PLAIN
for file in os.listdir('pics/test'):     # 读取图像文件
    img = cv2.imread(f'pics/test/{file}')
    gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY) # 转化一个灰度图

    faces = classifier.detectMultiScale(gray_img)
    x,y,w,h=faces[0]
    # 获取人脸区域的数据
    face_data=gray_img[y:y + h, x:x + w]
    # 获取标记和可信度
    mark, confidence = recognizer.predict(face_data)
    # 根据获得的标记，得到照片对应的人员名字
    name=names[mark]
    # 把名字放到图像上
    cv2.putText(img,name, (50, 50),font , 2, (255, 0, 255), 2)
    cv2.imshow("image",img) # 显示图像
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

3 运行程序观察识别效果。可以看到，利用生成的模型把待测照片中的人都识别出来了。

你知道吗？

前面训练的模型，是一个非常简易的模型。受算法、原始图像数据集等因素的影响，它的识别率很有限。比如，它在识别某些特征相近的人时，会出错，也不适合用来识别测试者之外的人员。真正实用的人脸识别系统要进行大量的数据训练，数据处理过程也更加复杂。不过，整体流程是基本相同的。

实践探究

如果把之前编写的人脸识别系统跟摄像头组合起来，就可以用来模拟以人脸识别为基础的点名或门禁系统。大体流程是：一、利用摄像头给每个人录入原始数据；二、利用数据生成识别模型；三、利用模型对摄像头新摄取的影像进行识别，得到人员的名称，从而实现自动点名或者模拟开门操作。OpenCV 库也提供了控制摄像头的函数，主要的操作代码如下：

```
cap=cv2.VideoCapture(0)      # 连接摄像头
ret_flag, vdata = cap.read()  # 获取图像数据
```

请同学们自行查阅相关资料，设计一个利用摄像头实现自动点名或自动门禁的系统。

二、识别书写数字

前面编写的“人脸识别”程序虽然生成了模型，但主要是在OpenCV提供的多个模型的基础上完成的，模型的层次结构等都不受控制。下面介绍如何完整构建一个简单的模型，并用训练后的模型进行预测——识别手写数字。

准备数据

为了便于人们开发识别手写数字的程序，科研人员特意提供了手写数字图像数据集，数据集中的一张图就包含一个数字。在这种数据集中，minist最为有名，它可通过以下代码加以调用。

```
import tensorflow.keras.datasets

# 其中 t_data 是用于训练的数据，v_data 是用于验证的数据
# 如果无法在指定位置找到这个文件，计算机就会自动到网络中下载
t_data, v_data = datasets.mnist.load_data(path='mnist.npz')
```

注意，训练模型时一般需要两组数据：一组用于训练，如学习如何提取图像的特征；另一组用于验证，也就是验证前面的学习成果。

搭建模型

搭建模型的过程，就是规定模型包含多少层，以及每层的参数特征等。

搭建简单的用于手写识别的模型。

1 打开step1.py文件，了解搭建识别模型的过程。

```
import os
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
def gen_model():
    model = models.Sequential()
    # 以下各层主要用于学习提取特征
    # 第 1 层，其中 input_shape 规定待训练图片的大小为 28*28，且为灰度图
    model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
    model.add(layers.MaxPooling2D((2, 2)))
```

```
    model.add(layers.Conv2D(64, (3, 3), activation='relu'))
    model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.Conv2D(64, (3, 3), activation='relu'))

    # 以下各层主要用于根据前面的结果进行输出
    model.add(layers.Flatten())
    model.add(layers.Dense(64, activation='relu'))
    model.add(layers.Dense(10, activation='softmax'))      #10 表示有 10 种结果

    # 根据各种参数编译模型
    model.compile(optimizer='adam',loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])
    return model

model=gen_model()
model.summary()
```

运行结果

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 26, 26, 32)        320
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 13, 13, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 11, 11, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2D) (None, 5, 5, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 3, 3, 64)          36928
_________________________________________________________________
flatten (Flatten)            (None, 576)               0
_________________________________________________________________
dense (Dense)                (None, 64)                36928
_________________________________________________________________
dense_1 (Dense)              (None, 10)                650
=================================================================
Total params: 93,322
Trainable params: 93,322
Non-trainable params: 0
```

可以发现，这个模型包括8层，共有9万多个参数。

② 理解下面的代码，并训练模型。注意，Python自带的IDLE编辑器在输出训练过程的信息时，会出现一些问题，因此建议大家在PyCharm等开发软件中运行，或者在命令提示符窗口中以类似“python step2.py”的方式运行。

```
t_data, v_data = datasets.mnist.load_data(path='mnist.npz')
# 6 万张训练图像，1 万张验证图像，进行必要的数据处理
t_imgs = t_data[0].reshape((60000, 28, 28, 1))
v_imgs = v_data[0].reshape((10000, 28, 28, 1))
t_imgs, v_imgs = t_imgs / 255.0, v_imgs / 255.0

model=gen_model()
model.summary()
# 训练
model.fit(t_imgs,t_data[1],epochs=5,
                validation_data=(v_imgs,v_data[1]),batch_size=128)
model.save('digital.h5')
```

运行结果

```
Epoch 1/10
1875/1875 [==============================] - 35s 18ms/step - loss: 0.8195 -
accuracy: 0.8781 - val_loss: 0.0620 - val_accuracy: 0.9811
……
Epoch 10/10
1875/1875 [==============================] - 33s 17ms/step - loss: 0.0150 -
accuracy: 0.9955 - val_loss: 0.0614 - val_accuracy: 0.9865
```

10次训练后，得到了准确度预计超过99%的模型。

训练时间和效果跟模型的层次复杂度、参数的多少、待训练数据的多少、训练次数等密切相关。一般来说，层次越多、参数越多、数据越多，训练次数越多，得到的模型越好，但有时又会过犹不及。具体如何设定，现在没有科学定论，人们通常根据使用经验进行选择。

识别数字

调用刚刚生成的模型，就可以识别图像中的数字了。这个模型的功能比较简单，只能用来处理只包含一个手写数字的图像。

运行step2.py程序，识别指定图像中的手写数字。

```
import os
import numpy as np
from PIL import Image
from tensorflow.keras import models

model=models.load_model('digital.h5')

for file in os.listdir('test'):
    img = Image.open(f'test/{file}').convert('L').resize((28, 28))
    img = np.reshape(img, (28, 28, 1)) / 255.
    data = np.array([1 - img])
    p = model.predict(data)
    # 找出最可能的数字
    n=np.argmax(p[0])
    # 计算可信度
    b=int(max(p[0]) * 100)
    print(f' 图像：{file}，识别的数：{n}，可信度：{b}%')
```

运行结果

当图像中含有多个手写数字时，通常需要先把数字分割成不同图像，然后再识别。过程比较复杂，不再介绍。

1. 要搭建一个模型需要考虑哪些因素？是不是训练次数越多越好？

2. 尝试参照类似的方法，训练一个可以识别各种字体字母的模型，然后用训练出来的模型进行识别。

交流评价

1. 参照下表，针对自己的学习过程作一个评价。

学习内容	掌握程度
函数、模块和库的概念	□ 不知道 □ 知道 □ 熟悉
编写自定义函数	□ 不会 □ 会 □ 熟练
用程序实现画风迁移	□ 不会 □ 会 □ 熟练
用程序识别图像中的人和物	□ 不会 □ 会 □ 熟练
用程序识别含有猫或狗的图像	□ 不会 □ 会 □ 熟练
人脸识别的一般过程	□ 不知道 □ 知道 □ 熟悉
识别手写数字的一般过程	□ 不知道 □ 知道 □ 熟悉

2. 分组交流、评价各种总结表和编写的程序，谈谈自己对Python的认识。

评价内容	评价项目
关于函数、模块和库	会安装要使用的第三方库
	能正确调用模块中的函数，完成编程任务
	交流时，增加了额外的函数、模块和库的介绍
关于智能编程	会调用已有的智能模型解决问题
	能够根据课本中的提示，完成训练简单模型的任务
	能够根据自己的体会，描述智能编程的过程

3. 组内评选出具有个性的Python程序作品，请设计者向全班介绍编写过程以及所使用的函数、编程模块等。

本章学习要点

- 人与计算机解决问题方法的异同
- 基于公式、基于枚举等编程思想
- 数字仿真、递归等编程思想
- 机器学习的一般过程
- 一些经典问题的求解过程

第3章 编程思想简介

前面大家学习了程序设计的基础知识，利用Python编写了不少有趣的程序。这一章则要学习如何根据问题，找到适合计算机的解决方法，然后编程实现。

编程时可以依据某些编程思想来设计解题的步骤，常见的思想包括基于公式求解、通过枚举求解、通过数字仿真求解、基于递归思想求解、通过机器学习求解等。

下面就让我们一起开始新的学习旅程，认识常用的编程思想。

活动任务

1. 基于公式编程解决问题，体会这种思想的特点。
2. 基于枚举、递归等思想编程解决问题，体会它们的特点。
3. 了解数字仿真，感受它的特点。
4. 回顾之前编写的程序，总结基于机器学习解决问题的过程。
5. 初步尝试用程序解决一些经典问题，感受编程的乐趣。

活动计划

第1节：计算机解题思想

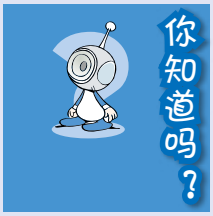
了解人与计算机求解问题的异同，了解基于公式解决问题、通过枚举解决问题、通过递归解决问题、通过数字仿真解决问题等解题思想。

第2节：编程解决经典问题

通过解决八皇后、八数码、汉诺塔等经典问题的过程，了解建模等基本思想，深化对递归、枚举等编程思想的理解。

活动要求

1. 活动1涉及的编程任务要求由个人独立完成，活动2中的程序只要求理解，不要求自行编程实现。
2. 交流评价活动1中的程序时，需能明确指出所编程序依据的编程思想，并能清晰介绍主要的步骤。
3. 讨论活动2中的程序时，只要求说清主要思路，不必关注程序实现的细节。



计算机解决问题需要根据一定的思路来设计解题步骤，这被看作是设计算法的过程。算法是程序设计的灵魂。很多人认为：算法＋数据结构＝程序。

早在公元前300年，古希腊著名数学家欧几里得在前人基础上写成的数学专著《几何原本》中，就提出了欧几里得算法（又称辗转相除法）。我国于公元1世纪编写的数学专著《九章算术》中也涵盖了一些算法的知识。一直以来的研究表明，人们普遍认为一切数学命题都存在算法，但直到20世纪初，研究者们才发现算法竟然还没有精确的定义。在随后的研究中，尽管有关算法的各种定义不断被提出，但是这些定义始终无法让人满意。

1936年，图灵在他的《论可计算数及其在判定问题中的应用》（*On Computable Numbers, with an Application to the Entscheidungsproblem*）一文中，对前人的研究做了重新论述，并全面分析了人的计算过程，在此基础上提出了一种假想的机器抽象模型。同时，他还第一次把计算和这个抽象模型联系起来，解决了算法定义的难题。

1. 下面列出了本章需要用到编程库，更多的相关库通常会在安装下面的库时自动安装。

库	简　介	pip安装命令
scipy	科学计算	pip install scipy
tensorflow	智能程序开发框架	pip install tensorflow
tensorflow-hub	tengsorflow增强	pip install tensorflow-hub
pygame	游戏开发包	pip install pygame
skimage	图像处理	pip install scikit-image

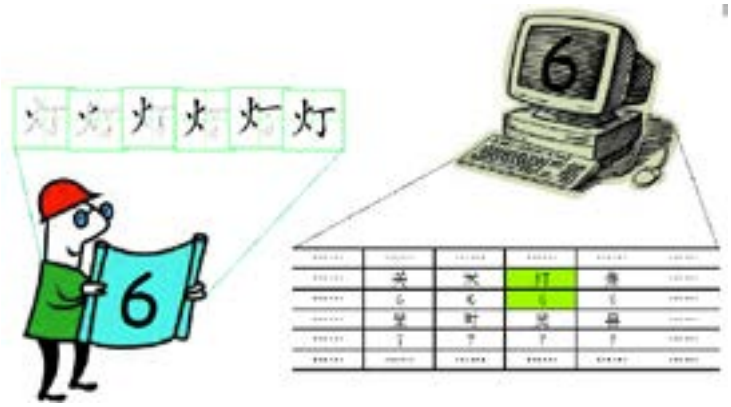
2. 本章需要使用名为delf的智能模型。

第 1 节　计算机解题思想

学习目标

- 了解人与计算机解决问题的异同。
- 掌握用公式求解和枚举求解的方法。
- 了解递归的概念，感受递归求解的过程。
- 了解数字仿真的概念，感受数字仿真求解的过程。

如何才能根据需求，设计适合在计算机上编程实现的解决方案呢？先来考虑这样一个问题：“灯”这个汉字有多少画？



对于人而言，把这个字写一遍，就可以得出它的笔画数，但编程让计算机做相同的工作却非常困难。不过计算机运算速度快，机械记忆能力强，如果把每一个汉字的笔画数都输入计算机，它就可以根据记录，马上找出答案。不难看出，对于人来说比较简单的方法，对于计算机可能难以实现。同样的，对计算机来说比较简单的方法，人很可能无法实现。人与计算机解决问题的方法有同有异。因此，人们用计算机解决问题时，应该充分考虑计算机的运算特点，设计出适合计算机的方法。设计的方法恰当与否，直接影响着程序实现的可行性和复杂程度。

在实践中，人们总结了一些求解问题的思想，如利用公式求解、枚举求解、递归求解等。

一、基于公式编程求解

在数学、物理等学科中，经常会用公式求解问题，例如，求解方程的根、解应用题、用坐标系描述物体的运动轨迹等。如果能把公式用程序语言描述出来，就可以用计算机来运算求解了。

公式求解过程

基于公式编程求解时，核心步骤是用程序语言把公式描述出来，此外还要考虑导入模块，输入、输出等细节问题。

求解方程 $ax^2+bx+c=0$ 的实数根。

1 寻找公式。可以利用求根公式进行解决。

$$x=\begin{cases}\dfrac{-b\pm\sqrt{b^2-4ac}}{2a} & (b^2-4ac\geq 0)\\ 0 & (b^2-4ac<0)\end{cases}$$

2 公式转换。用程序语言来描述求根公式，可以写为：

```
d = b**2 - 4 * a * c
if d >= 0 : x = ( (-b + d**0.5) / (2*a), (-b - d**0.5) / (2*a) )
else :  x = None
```

3 编程实现。需要考虑系数输入、结果输出等细节问题，大家可以参考下面的代码加以实现。

```
# 输入方程系数，利用 split 函数得到系数序列
paras = input(' 输入 a, b, c ： ').split(',')

# 利用 map 函数，把序列中的每一个元素都转换成整数
a , b , c = map ( int , paras )
d=b**2-4*a*c
if d>=0:
    x = ((-b + d**0.5) / (2 * a), (-b - d**0.5) / (2 * a))
```

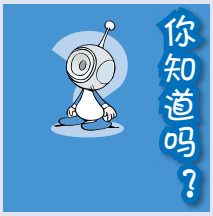
```
else :
    x = None
print(' 方程的实数根为：' , x )
```

4 运行求解。输入系数后，计算机就会立刻给出运算结果。操作时可以反复运行程序，输入不同的系数测试计算机的运行结果是否正确。

```
输入方程系数 a, b, c ： 1 , -4 , 3
方程的实数根为： (3.0, 1.0)

输入方程系数 a, b, c ： 1 , 0 , -4
方程的实数根为： (2.0, -2.0)

输入方程系数 a, b, c ： 1 , 0 , 4
方程的实数根为： None
```



程序用到了 map 函数，这个函数被称为映射函数，可以在函数名和参数序列之间建立关联。例如：

map(int , [1.1,2.2,3.3]) 可以把后面序列中的浮点数都转换成整数；map(pow, [1,2,3],[1,2,3]) 可以用于计算 1 的 1 次方、2 的 2 次方、3 的 3 次方。

不同公式对同一问题求解的影响

要解决同一个问题，很可能有多个公式可加以利用。选择不同的公式加以编程实现，程序一般会在运算速度、运算精度等方面有不同的表现。下面通过圆周率求解的过程加以介绍。

求圆周率。

1 寻找公式。

公式一：17世纪，人们找到了一个计算圆周率的公式，根据这个公式，只要计算足够多的项，就可以获得圆周率的近似值。

$$\pi = 4 \times (\frac{1}{1} - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \ldots)$$

公式二：后来，科学家们有提出了如下所示的计算圆周率的公式，同样的，只要计算足够多的项，也可以得到圆周率的近似值。

$$\pi = 16 \times (\frac{1}{1 \times 5^1} - \frac{1}{3 \times 5^3} + \frac{1}{5 \times 5^5} \ldots) - 4 \times (\frac{1}{1 \times 239^1} - \frac{1}{3 \times 239^3} + \frac{1}{5 \times 239^5} \ldots)$$

2 公式转换。

转换一：通过观察可以发现，第一个公式有三大特征：分数项无穷多，分子在−1和1之间不断转换；分母是1开头的奇数序列……分析后，可以利用下面的代码来表示。

```
# 数学公式无限，程序却需要有限，这里限定项的个数
for i in range (n) :
    nu = (-1)**i                    # 分子
    de = 2*i+1                      # 分母
    item = Fraction ( nu , de )     # 利用 Fraction 进行分数运算
    r = r + item
r = 4*r
```

转换二：第二个公式与第一个公式的基本特征相同：无穷多的项，1和−1转换的序列，以及以1开头的奇数序列等。

```
# 数学公式无限，程序却需要有限，这里限定项的个数
for i in range (n) :
    nu, f = (-1)**i, 2*i+1
    de1 = f * pow(5,f)          # 分母 1
    de2 = f * pow(239,f)        # 分母 2
    item1 = Fraction(nu,de1)
    item2 = Fraction(nu,de2)
    r1 = r1 + item1
    r2 = r2 + item2
r = 16*r1 - 4 * r2
```

3 编程实现。

程序一：

```
from fractions import Fraction

n , r = 10000 , 0
for i in range ( n ):
    nu = (-1)**i
    de = 2*i + 1
    item = Fraction(nu,de)
    r = r + item
print(float( 4*r ))
```

程序二：

```
from fractions import Fraction

n ,r1 ,r2 = 6 , 0, 0
for i in range ( n ) :
    nu , f = (-1)**i , 2*i+1
    de1 = f * pow(5,f)                    # 分母 1
    de2 = f * pow(239,f)                  # 分母 2
    item1 = Fraction(nu,de1)
    item2 = Fraction(nu,de2)
    r1 = r1 + item1
    r2 = r2 + item2
r = 16 * r1 - 4 * r2
print ( float ( r ) )
```

4 运行求解。

运行一：

```
3.1414926535900434
```

运行二：

```
3.1415926526153086
```

操作时会发现，运行第一个程序后需要等一段时间才能得到结果，圆周率被精确到了3.141……运行第二个程序几乎可以立刻得到计算结果，圆周率被精确到了3.1415926……

显而易见，根据公式二得到的程序二，运行效率更高。实际上，程序二每多循环一次，大体可以多得到一位有效数字。

根据公式编写程序，然后用计算机解决问题，这种情况下，程序的运行效率跟公式本身的效率有很大关系，如果能找到适合计算机使用的高效公式，那公式对应的问题就容易用计算机解决。不过，现实中有很多问题的求解过程无法简单地用公式来表示，人们还必须想其他方法，也就是换其他思维方式来解决问题。

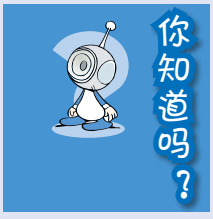
1. Python 中，浮点数的有效数字有 10 多个，一般情况够用，但在计算圆周率这类问题时，就会显得精度不足。这时，可以利用 decimal 模块来指定计算精度。参考下面的代码，利用程序二计算圆周率的 100 位有效数字。

```
from decimal import Decimal
from decimal import getcontext
getcontext().prec=100              # 指定精度
……
d=r.as_integer_ratio()             # 对 r 序列中的分数通分
pi=Decimal(d[0])/Decimal(d[1])     # 按指定精度进行运算
```

2. 幂运算比较费时。当指数很大时，即使是关于 1 的幂运算，也会耗费大量的计算时间。有同学想对上述程序分子的计算过程进行调整，你能想出更有效的方法吗？尝试写出新方法，并参考下面的代码比较计算所需的时间。

原方法：nu = (-1)**i　　新方法：_____

```
import time
t1=time.time()
…… # 运算代码段
print ( time.time() - t1 )
```



2002 年，科学家用超级计算机，算出了 1 万多亿位圆周率的近似值。存储这个数值需要超过 1 000 GB 的存储空间，如果一秒读一位数，读完它需要 3 万多年。由于精确计算圆周率对设备要求很高，因此计算圆周率慢慢变成了检验计算机运算速度和准确性的一种方法。

Super PI 是科学家开发的一个计算圆周率的软件，用它可以在普通的计算机上算出圆周率的小数点后几千万位，很多人就用它来检验计算机的性能。

二、枚举求解

枚举法也叫穷举法，其基本思想是根据提出的问题，列举所有可能情况，并用问题中提出的条件检验哪些是需要的，哪些是不需要的。枚举常用于解决“是否存在”或“有多少种可能”等问题。

枚举求解的过程

计算机枚举求解大体分为两个阶段：分析问题找出枚举集合；对枚举集合中的元素一一进行验证。

从面值为1元、2元、5元的钞票中取60张凑齐180元，有多少种方法?

1 问题分析。假设1元的有 x 张，2元的有 y 张，5元的有 z 张，根据问题可以列下面的方程组。

$$\begin{cases} x+y+z=60 \\ x+2y+5z=180 \end{cases}$$

虽然列出了方程组，但未知数有3项，等式只有2个，而且不能进一步化简，所以仍然无法通过公式计算直接得到结果。不过每种钞票的张数都超不过60张，也就是钞票的数量范围是知道的，这时不妨把每种情况都验证一下，从而找出答案。

2 确定枚举集合。根据前面的分析，不难想象，利用循环嵌套结构，可以轻松枚举出可能的数量组合。

```
for x in range(61):    # 最少 0 张，最多 60 张
    for y in range (61) :
        for z in range (61) :
```

3 确定检验条件。依据公式对可能的组合进行验证，能满足等式1和等式2的就显示出来。

```
if x + 2 * y + 5 * x == 180 and x + y + z == 60: print ( x , y , z )
```

4 编程实现。把枚举集合和检验条件组合起来，编写完整的基于枚举的问题求解程序，寻找可能的组合。

```
for x in range(61):
    for y in range (61) :
        for y in range (61) :
            if x + 2 * y + 5 * z == 180 and x + y + z == 60: print ( x , y , z )
```

5 运行求解。

```
0 40 20
3 36 21
6 32 22
9 28 23
……
```

在前面的例子中，枚举集合和检验条件都容易得出。但有些问题，如何用程序语言表示枚举集合，如何用程序语言表达检验条件却比较困难。这时，需要先对问题进行抽象。

红帽子、蓝帽子问题。

五顶帽子，三红两蓝。随机将三顶戴在A、B、C三人头上。每个人只能看到其他人的帽子，也不知道剩余两顶帽子的颜色。问A自己戴什么颜色的帽子，A不知道。再问B，B也不知道。最后问C，C说他知道自己戴的帽子是什么颜色。请问他们分别戴什么颜色的帽子。

1 问题分析。问题可抽象为从“红红红蓝蓝”中任选三种。A不知道帽子颜色，意味着B和C没同时戴蓝色帽子；B不知道帽子颜色，意味着A和C没同时戴蓝色帽子；C知道，意味着A和B同时戴蓝帽子。

2 确定枚举集合。Python中的itertools模块，有很多用于处理元素组合的函数，大家可以参考下面的代码。

```
import itertools
s=' 红红红蓝蓝 '
a=itertools.permutations(s,3)      # 任取三个元素进行组合
d=set(a)                           # 去掉重复的组合，形成枚举集合
```

3 确定验证条件。A、B、C三人戴的帽子分别用p中的元素来表示，此时检验条件的表述可以参考下面的代码。

```
# 条件 1，A 看到 B 和 C 没有同时戴蓝帽子
ta = not ( p[1]==' 蓝 '  and p[2] == ' 蓝 ' )
```

```
# 条件 2，B 看到 A 和 C 没有同时戴蓝帽子
tb = not ( p[0]==' 蓝 '  and p[2] == ' 蓝 ' )
# 条件 3，C 看到 A 和 B 同时戴蓝帽子
tc = p[0]==' 蓝 '  and p[1] == ' 蓝 '
if all( [ ta, tb , tc]) : print ( p )
```

4 编写程序。根据自己的分析，把下面的程序补充完整。

```
import itertools
s=' 红红红蓝蓝 '
a=itertools.permutations(s,3)
d=set(a)
for _____ in _____:
    ta = not (p[1] == ' 蓝 ' and p[2] == ' 蓝 ')
    tb = not (p[0] == ' 蓝 ' and p[2] == ' 蓝 ')
    tc = p[0] == ' 蓝 ' and p[1] == ' 蓝 '
    if all ( [ta, tb, tc ] ) : print ( p )
```

运行结果

```
(' 蓝 ', ' 蓝 ', ' 红 ')
```

在上面的编程过程中，把随机分配帽子的问题，转换成了求组合的问题，把每个人的回答转换成了条件表达式。这样处理后，才找到了枚举集合和验证条件的表达方法，进而才可以进行枚举运算。

针对A、B、C三人，问谁是罪犯，后面的事实成立。事实：1. 至少有一人有罪；2. A有罪时，B和C与之同案；3. C有罪时，A、B与之同案；4. B有罪时，没有同案者；5. A、C中至少有一人无罪。有位同学编写了下面的程序进行运算，请问t1、t2……t5分别对应哪个事实？

```
# 有罪为 True，无罪为 False
a = itertools.product([True, False], repeat=3)
d = list(a)
for p in d:
    t1 = any ( [p[0], p[1], p[2]] )
    t2 = not all([p[0], p[2]])
    t3 = not p[0] and not p[2] if p[1] else True
    t4 = ( p[1] and p[2] ) if p[0] else True
    t5 = ( p[0] and p[1] ) if p[2]  else True
    if all( [ t1, t2, t3, t4, t5 ] ): print(p)
```

明确枚举法的适用范围

在使用枚举法时，需要验证问题的所有可能情况，而有些问题的可能性无法一一枚举，这类问题就不适合用枚举思想求解。例如，证明$\sqrt{2}$是无理数。让计算机一一列出$\sqrt{2}$所有的数位，并验证其没有规律性是不可行的，而用反证法则可以轻松证明。

假设$\sqrt{2}$是有理数，得：$\sqrt{2}=\frac{q}{p}$，其中p，q互质

两边平方，得：$2p^2=q^2$

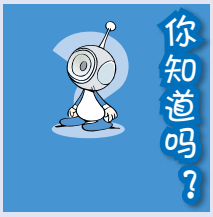
可见，q是偶数，记为$q=2m$，m为整数。于是

$2p^2=(2m)^2$

又得$p^2=2m^2$

可见，p是偶数。p，q都是偶数与假设p，q互质相矛盾，因此假设不成立，$\sqrt{2}$是无理数。

再例如，在求证歌德巴赫猜想的过程中，人们已经用计算机验证了，对于400万亿内的数，这个结论都是成立的。但无论验证多大的数，都无法证明猜想是否成立。因为数无穷多，不可能一一列举验证。



枚举法不适合用来证明某些猜想成立，却可以用来寻找一些猜想的反例。例如，大数学家费马曾经提出了一个猜想“若$x=2^{2^n}+1$，则x是素数”。

n分别为0、1、2、3、4时，x为3、5、17、257、65 537，都是素数。$n=5$时，$x=4\ 294\ 967\ 297=641\times 6\ 700\ 417$，猜想不成立。历史上，人们就是通过计算$n=5$时的数，推翻了费马的这个猜想。

感兴趣的同学可以编写程序进行验证。

优化要枚举的元素

适当地设置循环变量，可以缩小列举范围，提高计算效率。例如，前面计算180元有多少种取法时，计算机需执行$61\times 61\times 61$（226 981）次循环体。如果考虑到$x=60-y-z$，可以把算法改成下面的形式。

```
for y in range(61):
    for z in range (61) :
        if (60 - y - z) + 2 * y + 5 * z == 180 :
            print ( 60 - y - z , y , z )
```

此时计算机只需执行61 × 61 = 3 721次循环体。再考虑 z 的取值范围，z 不可能大于36（ 36 = 180 ÷ 5 ），又考虑到 y 不能超过 60— z，那么算法可以改成下面的形式。

```
for z in range (37) :
    for y in range(61-z):
        if (60 - y - z) + 2 * y + 5 * z == 180 :
            print ( 60 - y - z , y , z )
```

修改后，程序只需要执行 1 591次循环就可以解决问题了。

编程验证101 ～ 10 000之间的偶数都可以写成两个素数的和。下面有两种方案。

方案一：枚举指定范围的偶数，然后验证这些偶数是否等于两个素数的和。

方案二：找出10 000以内的素数，然后让素数两两相加，验证其结果是否包括了10 000以内的偶数。

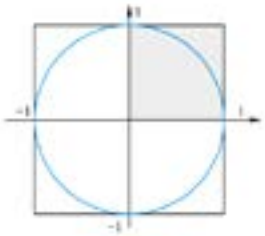
分别编程实现这两种解决方案，然后说明每种方案的特点。

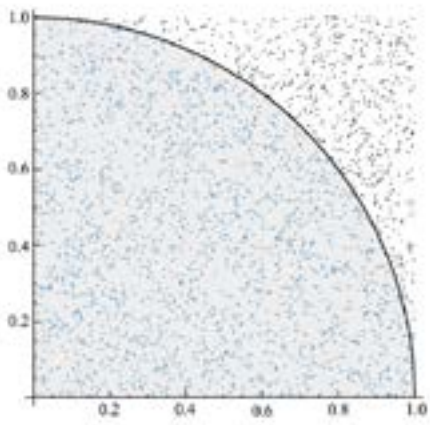
三、数字仿真求解

当针对一个问题设计一个具体的解决方法比较困难时，还可以考虑采用数字仿真的方法。

实验法求圆周率。

1 问题分析。计算圆周率有一种有趣且直观的方法：首先，在正方形纸上画它的内接圆；其次，向纸上随机投针，投足够多次；最后，统计正方形内和圆内的针数，然后进行运算，就可以得到圆周率了。





2 仿真环境搭建。为了便于计算，构建边长为2的正方形，其内接圆半径为1；由于对称性，圆和正方形的问题可以简化为一个四分之一扇形和它的外接正方形的问题； 针落在圆内，意味着落点到原点的距离肯定小于1（小于圆半径），反之则落在圆外，这样针落点的情况就转换成了距离计算。

3 编程实现。

```
from random import random
n=1000000                  # 假定投了 1 000 000 次
hits = 0.0
for i in range(0, n):
    x, y = random(), random()          # 利用随机数，模拟投针
    dist = pow(x ** 2 + y ** 2, 0.5)   # 计算到原点的距离
    if dist <= 1.0:                    # 小于等于 1，意味着在圆内或圆上
        hits = hits + 1                # 统计圆上和圆内的次数
pi = 4 * (hits/n)
print(pi)
```

4 运行求解。注意，每次运行结果都不完全相同，大约可精确到3.14。

```
3.14212
```

四、递归求解

如果一个函数需要在解决问题的过程中不断调用自身，就认为这个程序采用了递归的思想。例如，计算 $n!$ 时，可以利用循环把1～n 之间的数依次相乘从而得出结果，也可以采用递归的方式进行求解。

参照下面的代码，编写基于递归思想计算阶乘的程序。

```
def factorial(n):
    # 如果变量 n 等于 1，给出结果 1
    if n==1:return 1
    else :
        # 如果变量 n 大于 1，调用 factorial(n-1)，从而缩小规模
        return n * factorial(n - 1)
print(factorial(3))
```

运行结果

```
6
```

假设n=3，这个程序的计算过程如下。

① 调用factorial(3)，因为n大于1，所以factorial(3) = 3 * factorial(2)。

② 调用factorial(2)，因为n大于1，所以factorial(2) = 2 * factorial(1)。

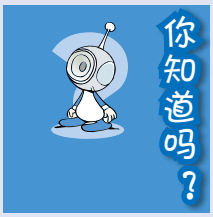
③ 调用factorial(1)，因为n等于1，所以factorial(1) = 1，无需调用factorial自身，递归结束，返回调用factorial(1)的函数factorial(2)。

④ 计算factorial(2) = 2 * factorial(1) = 2 * 1 = 2，无需再调用factorial自身，递归结束，返回调用factorial(2)的函数factorial(3)。

⑤ 计算factorial(3) = 3 * factorial(2) = 3 * 2 = 6，无需再调用factorial自身，递归结束，得到结果 6。

不难发现，递归其实是通过不断地调用自身，从而不断缩小待求解问题的规模，直至到达一个明确的知道答案的点，然后再反推回去的过程。前面的程序在变量n等于1时，得到明确的结果，然后停止递归调用并反推回去。

递归思想非常适合用来解决汉诺塔等经典问题。



基于递归思想编写的程序，通常会呈现这样的特点：某个函数在执行过程中，不断地调整参数并调用自身。调整参数的过程，就是不断缩写问题规模，直到一个明确节点的过程。注意，递归调用时，先调用的函数不一定先执行完。

五、基于机器学习的求解

现实中的很多问题很难找到明确的、有效的解决方法，如围棋博弈问题、人脸识别问题等。研究者们另辟蹊径，他们让计算机自己学习某个特定领域的已有经验，即在大量数据的基础上训练模型，然后用训练的模型对新出现的同类问题进行预测，从而实现问题的解决。

2017年举世瞩目的围棋“人机大战”标志着这种思想的成熟，并迅速应用到各行各业。这种思想及实现通常被视为人工智能技术重要组成。

动手实践

感受更多人工智能程序。

1 打开名为的delf.py的程序文件，指定两张图，让计算机自动寻找图中的相似之处。

2 搜索并下载名为KeTrain的软件，这是用Python语言调用围棋引擎Katago开发而成的围棋软件。尝试运行软件，与计算机下下围棋。

3 回想之前编写的人脸识别、数字识别等智能程序，总结通过机器学习解决问题的过程。

思考与练习

1. 基于公式编程解决问题时，一般要经过怎样的过程？你认为基于公式的方法适合求解怎样的问题？

2. 谈谈自己对枚举、数字仿真、递归等思想的认识。说说你认为这些思想分别适合用来求解什么样的问题。

3. 编写程序，输入三角形三边的长度后，计算机自动判断这样的三角形是否存在，如果三角形存在，则计算它的周长和面积。

4. 据说韩信点兵有独特的方法。比如，它令士兵排成5列余1人，排成6列余5人，排成7列余4人，排成11列余10人，韩信就知道士兵的人数了（士兵总数在1 000人左右）。请编程尝试计算士兵人数。

第2节 编程解决经典问题

学习目标

- 了解八皇后问题的解决过程。
- 了解八数码问题的解决过程
- 了解汉诺塔问题的解决过程。
- 体验计算机解决问题的过程，激发学习兴趣。

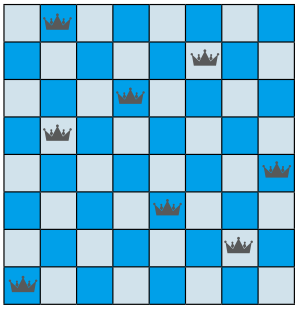
一、八皇后问题

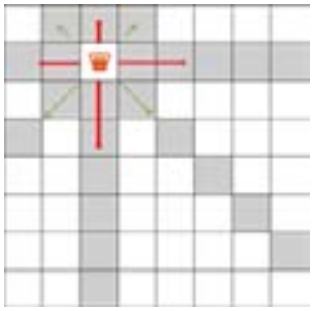
八皇后问题是一个与国际象棋相关的问题，具体内容是：在8×8格的国际象棋棋盘上摆放8个“皇后”，使其不能互相攻击，问有多少种摆法。

这个问题在1948年被提出，鼎鼎大名的数学家高斯也曾专门研究过。计算机普及后，很多人尝试编程解决这个问题。

问题分析

根据规则，“皇后”可以沿直线或对角线展开攻击。问题要求“皇后”不能互相攻击，这意味着任意两个“皇后”不能处于同一行、同一列或同一斜线上。如下面右图放入一个“皇后”，图中灰色的格子都不能再放“皇后”。





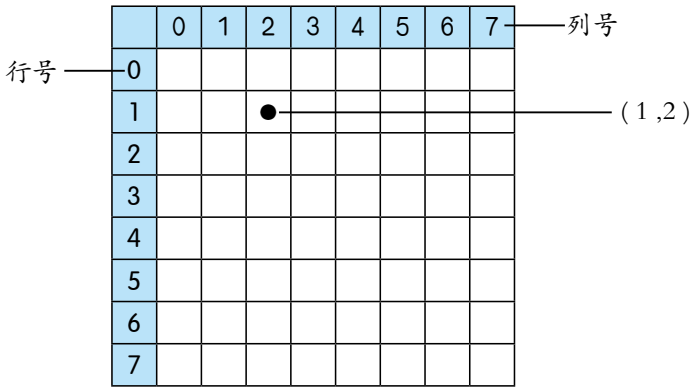
很容易想到，只要把8个“皇后”在棋盘中可能的位置都试一遍，然后把满足条件的摆法记录下来，就可以解决问题了。这种想法，其实就是基于枚举思想的解决方案。

建模一

计算机处理的最终都是二进制数，要解决八皇后问题，最终也需要把问题数字化，也就是说，要把摆放了棋子的棋盘数字化。

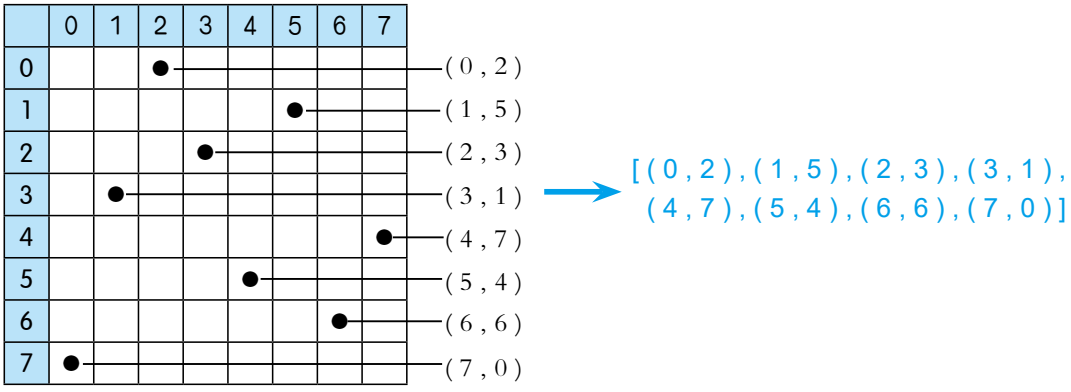
棋盘数字化

如果把棋盘的行分别标记为0、1、2……列分别标记为0、1、2……这样一来，就可以利用行列的组合表示棋盘中的每个格子了。下图中带有黑点的格子就可以记为（1，2）。



在解决这个问题的过程中，并不需要关注每个格子，只要关注放置了“皇后”的那8个格子就可以了。所以整个棋盘的问题，可以转换成8个格子的问题，进而转换为8个格子的坐标问题。

例如，下图所示的放了“皇后”的棋盘，实际上就可以转换成一个包含8个坐标的列表。

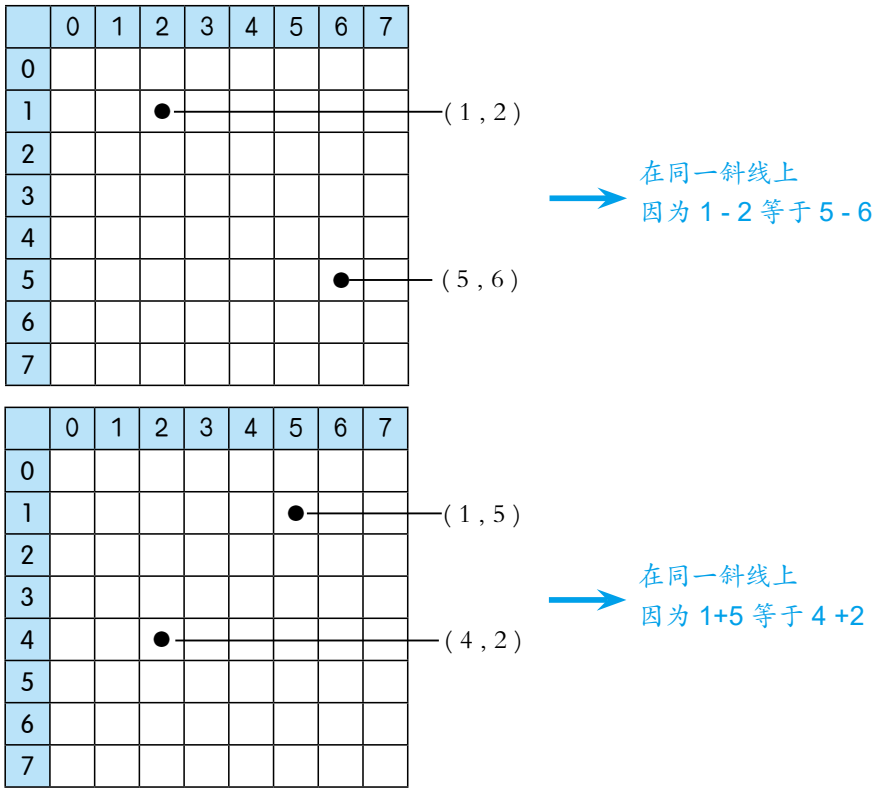


检测规则数字化

把棋盘数字化后，下面就要考虑如何把检测条件数字化。要求“皇后”不在同一行和同一列很简单，只要8个格子的横坐标和纵坐标都不相等就可以了。可以用类似下面的方式：

```
# 利用集合元素的唯一性，判断行列是否都不相同
len ( {x0 , x1 , x2 , x3 , x4 , x5 , x6 , x7} ) == 8  # 都不相同意味着有 8 个行号
len ( {y0 , y1 , y2  ,y3 , y4 , y5 , y6 , y7} ) == 8  # 都不相同意味着有 8 个列号
```

问题的难点在于，如何判断两个“皇后”是不是在同一斜线上。仔细观察就会发现，如果两个格子的行列坐标值的差相等或者行列坐标值的和相等，它们就在同一斜线上。



由此可以得到判断条件：

```
# 利用集合元素的唯一性，判断是否在不同的斜线上
len ( { x0-y0 , x1-y1 , x2-y2 , x3-y3 , x4-y4 , x5-y5 , x6-y6 , x7-y7 } ) == 8
len ( { x0+y0 , x1+y1 , x2+y2 , x3+y3 , x4+y4 , x5+y5 , x6+y6 , x7+y7 } ) == 8
```

程序实现一

根据建模获得的数字化棋盘和数字化的判断条件，编程求解。

根据建模一，求解八皇后问题。

1 把棋盘64个格子数字化。

```
r=[]
for x in range(0,7):
    for y in range(0,7):
        r.append((x,y))          #64 个格子的坐标，都保存到了变量 r 中
```

2 获取所有可能的摆放方法。也就是说，要从所有的格子中任取8个用于摆放“皇后”，这变成了组合问题。注意，从64个格子中任取8个，取的方式非常多，不要把组合迭代器转换成列表或元组，否则可能导致计算机因内存不足而崩溃。

```
import itertools
solutions = itertools.combinations(r,8)       # 从 r，即从 64 个格子中，任取 8 个
print(next(solutions))
```

运行程序可以发现，得到的摆放方案就是一个元组，里面包含了8个坐标：

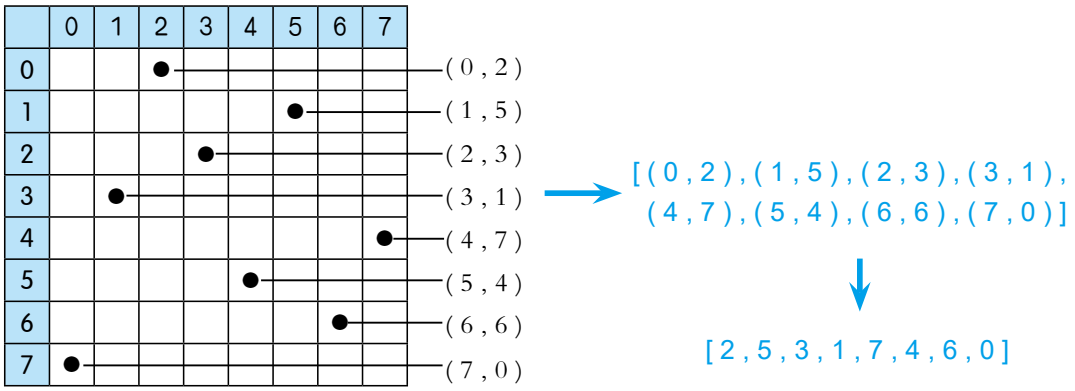
```
((0, 0), (0, 1), (0, 2), (0, 3),(0, 4), (0, 5), (0, 6), (0, 7))
```

3 完善程序，寻找解决方案。注意，这个程序可能需要运行几个小时。

```
result=[]
for solution in solutions:
    xs={grid[0] for grid in solution}                 # 行号的集合
    if (len(xs)!=8):continue
    ys={grid[1] for grid in solution}                 # 列号集合
    if (len(ys)!=8):continue
    ds1={grid[0]-grid[1] for grid in solution}        # 斜线 1 的集合
    if (len(ds1)!=8):continue
    ds2={grid[0]+grid[1] for grid in solution}        # 斜线 2 的集合
    if (len(ds2)!=8):continue
    result.append(solution)
    print(solution)
print(len(result))
```

建模二

前面的建模过程中使用了行列组合来表示格子，其实还可以对这种方式进一步优化。设想有这样一个序列，元素在序列中的位置表示行，元素的值表示列，8个格子在棋盘上的摆放就可以表述为包括8个数的序列（见下图）。



元素的位置代表行，而元素在序列中的位置肯定是不同的，这样就自然保证了格子不在同一行；元素的值表示列，如果元素的值不一样，就可以保证格子不在同一列。因此，可能的方案就可以转换成0~7这8个数的排列问题。

```
import itertools
#8 个数的排列
solutions = itertools.permutations ( [0,1,2,3,4,5,6,7] , 8 )
```

而检测条件也可以进一步简化，只需要判断在不在同一斜线就可以了。

```
for solution in solutions:                    #x 是位置表示行；y 是值表示列
    ds1= set((x + y) for x, y in enumerate(sulution))
    if len(ds1!=8):continue
    ds2= set((x - y) for x, y in enumerate(sulution))
    if len(ds1!=8):continue
```

程序实现二

根据建模二，求解“八皇后”问题的答案。

1 把棋盘和8个皇后数字化，组合可能的方案。

```
import itertools
#8 个数的排列组合
solutions = itertools.permutations([0,1,2,3,4,5,6,7], 8)
```

2 进行检测，找出满足条件的摆放方案。

```
r=[]
for solution in solutions:
    ds1= set((x + y) for x, y in enumerate(solution))
    if len(ds1)!=8:continue
    ds2= set((x - y) for x, y in enumerate(solution))
    if len(ds2)!=8:continue
    r.append(solution)
    print(solution)
print(len(r))
```

3 运行程序，计算机在几分钟内就能找出所有的方案。

实践探究

1. 尝试描述一下：如何用有序的8个数来表示八皇后问题可能的答案。

2. 用计算机求解八皇后问题时，得到了下面两个序列，请把它们表示的摆放方案画出来。

(5, 2, 4, 6, 0, 3, 1, 7)　　(4, 6, 1, 5, 2, 0, 3, 7)

	0	1	2	3	4	5	6	7
0						●		
1			●					
2								
3								
4								
5								
6								
7								

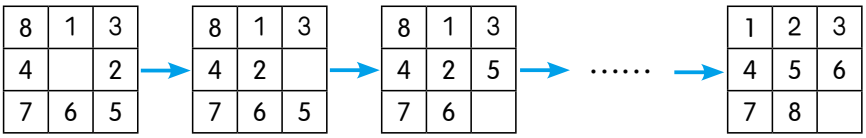
	0	1	2	3	4	5	6	7
0								
1								
2								
3								
4								
5								
6								
7								

实践探究

八皇后问题也经常采用其他思想来进行编程求解。感兴趣的同学请自行查阅相关资料，然后根据资料的内容自行编程实现。

二、八数码问题

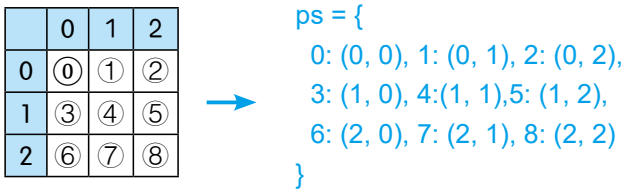
八数码问题，来自一个经典的小游戏，主要任务是把一个九宫格内排乱的八个数码格利用空格，按要求重新排列好。移动时，只能横向或纵向将空格与其他格子交换位置。具体可参见下图。



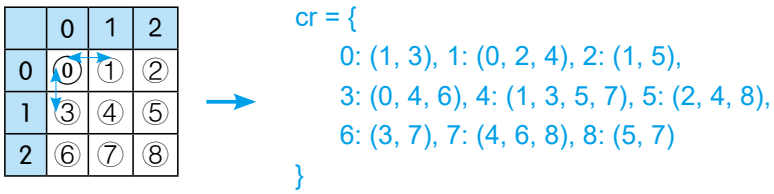
建模

格子及格子移动数字化

类似的，如果把九宫格的行分别标记为0、1、2，列分别标记为0、1、2，再把格子从左到右、从上大小依次标上序号，那么就可以用Python中的字典来表示九宫格的每个格子。



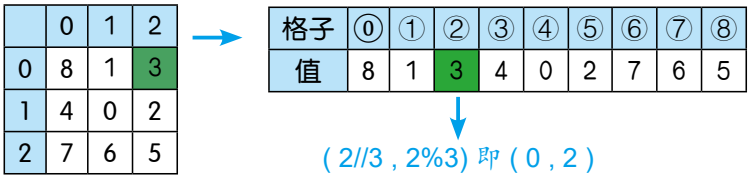
根据问题可知，移动空格时，只能跟同一行或同一列相邻的数码格进行交换，比如0号格只能跟1号和3号格交换，1号格能跟0号、4号、2号格交换，因此数码格的移动条件可以用下面的字典来表示。



排列方案数字化

参考前面的八皇后问题，很容易想到一个排列方案可以转化成一个序列

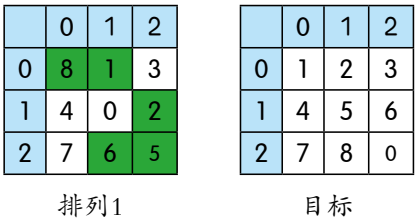
（空格用 0 表示），如初始的数码排列可以转换为 (8,1,3,4,0,2,7,6,5)，目标则可以转换为 (1,2,3,4,5,6,7,8,0)。其中，每个数码的实际位置坐标可以用它在序列中的位置进行转换。例如，初始布局中的3，在序列中的位置是2，即位于序号为2的格子中，其坐标可以表示为：x2=2//3、y2=2%3。



差异评估数字化

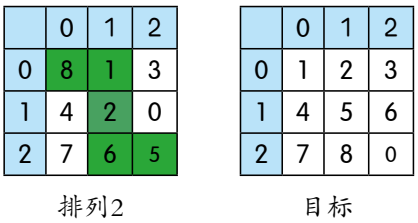
接下来就要考虑，如何引导程序在可能的方案中，一步步寻找合适的方案，进而完成排列任务。也就是说，如何找到合适的解题路径。要是每一步都跟最后的结果靠近一点，一步步走下来，自然就可以解决问题了。但问题在于，怎么知道一个排列方案比另一个排列方案更接近目标呢?

每个人都可以根据自己的设想提出评估方法，下面介绍目前经常使用的差异评估法，它分为两部分：一、统计有多少个数字格不在自己的位置上，结果记为*f(n)*；二、所有位置不对的数字格，其横、纵坐标与目标位置横、纵坐标的差的绝对值，结果记为*h(n)*。当一个方案的*f(n)+h(n)*比另一方案小时，就认为这个方案更接近目标。



排列1　　目标

5 个数码格格不对，*f(n)* = 5
h(8) = |0-2|+|0-1|=3　*h(6)* = |2-1|+|1-2|=2
h(1) = |0-0|+|0-1|=1　*h(5)* = |2-1|+|2-1|=2
h(2) = |1-0|+|2-1|=2
h(n)=3+1+2+2+2=10
差异度 =*f(n)* + *h(n)* = 15



排列2　　目标

5 个数码格格不对，*f(n)* = 5
h(8) = |0-2|+|0-1|=3　*h(6)* = |2-1|+|1-2|=2
h(1) = |0-0|+|0-1|=1　*h(5)* = |2-1|+|2-1|=2
h(2) = |1-0|+|1-1|=1
h(n)=3+1+2+2+2=9
差异度 =*f(n)* + *h(n)* = 14

	0	1	2
0	8	1	3
1	0	4	2
2	7	6	5

排列3

	0	1	2
0	1	2	3
1	4	5	6
2	7	8	0

目标

6 个数码格格不对，*f(n)* = 6
h(8) = |0-2|+|0-1|=3 *h(4)* = |1-1|+|1-0|=1
h(1) = |0-0|+|0-1|=1 *h(5)* = |2-1|+|2-1|=2
h(2) = |1-0|+|2-1|=2 *h(6)* = |2-1|+|1-2|=2
h(n)=3+1+2+2+2=11
差异度 =*f(n)* + *h(n)* = 17

按照移动规则，排列1可以转换为排列2或排列3等方式，而考察差异度评估值可以发现，排列1的差异值是15、排列2的值是14、排列3的值是17……所以认为排列2更接近目标，应该从排列1变换为排列2。

这个差异值的评估可以通过下面的函数来实现。

```
# 评估差异度
def comp_differ(data):
    f, h = [], []
    for index, n in enumerate(data):
        # 0 代表空格，不进行统计
        if n == 0: continue

        # 计算 f(n)
        # 数码值与位置是否匹配，如数码 3 是否在 3-1 的位置上
        if index == n - 1:
            f.append(0)
        else:
            f.append(1)

        # 计算 h(n)
        # 根据位置，获得当前坐标
        cx, cy = index // 3, index % 3
        # 目标布局中，1 在 0 号格，2 在 1 号格……
        # 因此可利用 ps 字典得到最终的坐标
        x, y = ps[n - 1]
        h.append(abs(cx - x) + abs(cy - y))
    return sum(h) + sum(m)
```

实践探究

根据建模信息可知，空格在中心位置，也就是在4号格子时，它可以跟上、下、左、右4个数码格进行交换。前面的例子只计算了其中两种交换后的差异度变化，请同学们自行计算其他两种排列的差异度。

程序实现

根据建模信息编程求解八数码问题。

1 理解下面的代码，把建模过程中的数字化信息写入程序，并增加两个变量来记录操作步骤和尝试过的排列方案。

```
# 格子坐标
ps = {0: (0, 0), 1: (0, 1), 2: (0, 2),
      3: (1, 0), 4:(1, 1),5: (1, 2),
      6: (2, 0), 7: (2, 1), 8: (2, 2)}
# 可移动位置关系
cr = {0: (1, 3), 1: (0, 2, 4), 2: (1, 5),
    3: (0, 4, 6), 4: (1, 3, 5, 7), 5: (2, 4, 8),
    6: (3, 7), 7: (4, 6, 8), 8: (5, 7)}
# steps 存储步骤，puzs 存储用过排列布局
steps, puzs = [], []

# 评估差异度
def comp_differ(data):
    f, h = [], []
    for index, n in enumerate(data):
        if n == 0: continue
        if index== n - 1: f.append(0)
        else: f.append(1)
        cx, cy = index // 3, index % 3
        x, y = ps[n - 1]
        h.append(abs(cx - x) + abs(cy - y))
    return sum(f) + sum(h)
```

2 理解下面的函数，它可以用来选择差异度更小的排列方案。

```
# 根据差异评估值，筛选出差异度更小的排列方案
def get_steps(data):
    i0 = data.index(0)              # 找到空格的位置
    ds = [data[ci] for ci in cr[i0]]  # 根据空格位置，获取可能交换的格子
    r, dif = None, 100      # 默认交换的格子为 None，默认差异度是 100
    for d in ds:            # 对可能的格子进行处理
```

```
        t, id = list(data), data.index(d)
        t[i0], t[id] = t[id], t[i0]              # 交换空格和数码格，形成备选排列 t
        if tuple(t) in puzs: continue            # 如果排列用过，就尝试下一种排列
        td = comp_differ(t)                      # 评估差异度
        if td < dif: dif = td; r = tuple(t)      # 把差异度小的方案留下来
    return r
```

3 理解统筹调用其他函数解决问题的函数。

```
# 求解，参数 data 为初始条件
def solve(data):
    result=False
    while data != (1, 2, 3, 4, 5, 6, 7, 8,0) :          # 未达成目标就循环
        steps.append(data)                               # 记录步骤
        if data not in puzs: puzs.append(data)           # 记录搜索过的布局
        data = get_steps(data)                           # 获得下一个排列
        # 当前尝试的排列方案不可行了，退回上一步，尝试其他排列
        if data == None:
            steps.pop()                          # 操作步骤删除最后一个元素
            data = steps[-1]                     # 调用前一步的排列方案，继续尝试
        if len(steps) >  10000: break            # 步数太多就强制退出
    else :result=True
    return result
```

4 理解调用函数解决问题的代码。

```
# 初始条件
a = (8, 1, 3, 4, 0, 2, 7, 6, 5)
if solve(a) :
    print(f' 解出来了，共 {len(steps)} 步。')
else :
    print(' 失败了！ ')
```

5 运行程序，查看运行结果。

```
解出来了，共 68 步。
```

1. 修改程序，自行输入一个初始的排列，然后用这个程序尝试解决。

2. 修改程序，在解决了问题后，让计算机用更直观的方式把移动过程显示出来。

程序优化

测试过程中会发现，并不是每一种初始的八数码排列布局，最后都能转换为目标布局。不能转换时，就可以认为当前初始条件下，八数码问题无解。在长期的研究过程中，人们发现可以通过计算逆序数的方法来判断八数码问题是否有解。

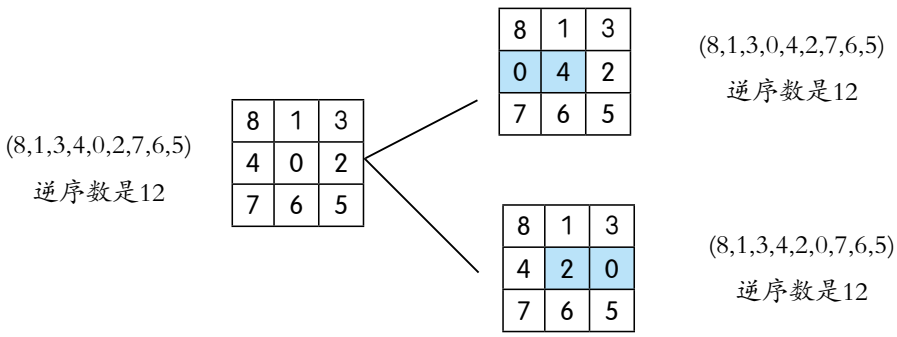
在一个序列中，如果一对数，前面的数大于后面的数，那么它们就是一个逆序，一个序列中逆序的总数就称为这个序列的逆序数。例如，对于八数码问题的起始序列：

(8,1,3,4,0,2,7,6,5) 中 (0 表示空格，不参与逆序数计算）存在：
8>1、8>3、8>4、8>2、8>7、8>6、8>6，记为 7。
3>2，记为 1
4>2，记为 1
7>6、7>5，记为 2
6>5，记为 1
逆序数：7+1+1+2+1=12

通过计算可以得到，表示八数码问题初始状态的序列的逆序数是12，而目标状态(1,2,3,4,5,6,7,8,0)的逆序数是0，两者都是偶数，所以可以从初始状态转换为目标状态。

也就是说，对八数码问题而言，如果初始序列与目标序列的逆序数的奇偶性相同，则问题有解，否则无解。这是因为，移动过程不改变序列逆序数的奇偶性。下面进行讲解。

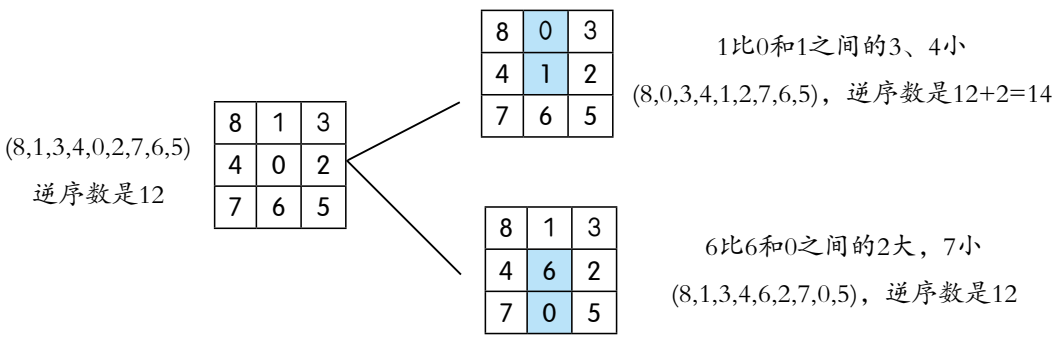
1. 当空格位置进行左右移动时（记住0表示空格，不参与逆序数计算），序列的逆序数不变。



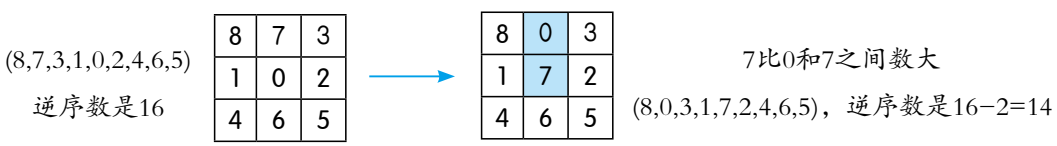
2. 当空格位置进行上下移动时，有3种情况：

（1）移来的数比中间的两个数都小，因此序列的逆序数要+2，显然逆序数的奇偶性不变；

（2）移来的数比中间的一个数大，比另一个数小，因此序列的逆序数不变，即逆序数的奇偶性不变；

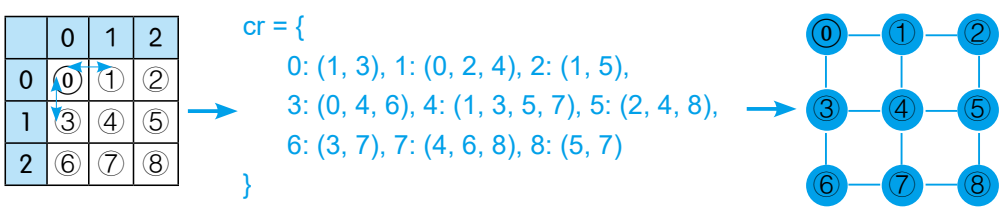


（3）移来的数比中间的两个数都大，因此序列的逆序数要−2，显然奇偶性也不变。



由此可知，无论左右还是上下移动，都不会改变目标序列逆序数的奇偶性，所以逆序数奇偶性不同，肯定无法转换。现在就面临一个新的问题了：逆序数奇偶性相同，就肯定能进行转换吗？

答案是肯定的。可以这样思考：把移动条件转换成连通图，如果两个位置能够互相交换，那它们之间必然存在相通的线。观察图可以发现，任意两个格子之间都有通道，因此只要初始序列与目标序列的逆序数同为奇偶，八数码问题就一定有解（即一个序列能转换为逆序数奇偶相同的任意序列）。



分析完毕，接下来就要考虑如何编程实现了。

完善前面的程序，让它能迅速判断是否有解。

1 阅读下面计算序列逆序数的代码。注意，序列中的0并不参与逆序数运算。

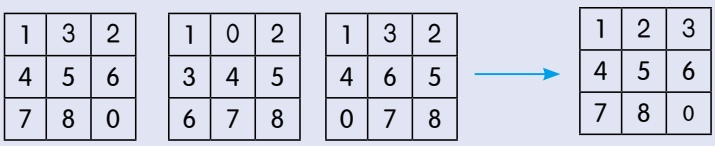
```
# 计算序列对应的逆序数
def inverse(data):
    ns=0
    for i in range(1,9):
        #0 不参与运算
        if data[i]==0 : continue
        for j in range(0,i):
            # 累加统计逆序数
            if data[j ]> data [i] : ns=ns+1
    # 返回逆序数
    return ns
```

2 修改程序，调用新加入的函数，判断八数码问题是否有解。

```
# 初始状态
a = (8, 1, 3, 4, 0, 2, 7, 5, 6)
if inverse(a) % 2 != 0:
    print(' 无解 ')
else:
    if solve(a) :
        print(f' 解出来了，共 {len(steps)} 步。')
    else :
        print(' 失败了！ ')
```

3 修改表示初始状态的序列，看看程序能否快速判断出相应的八数码问题有解还是无解。

左侧的排列方式能否转换成右侧的排列方式？为什么？



三、汉诺塔问题

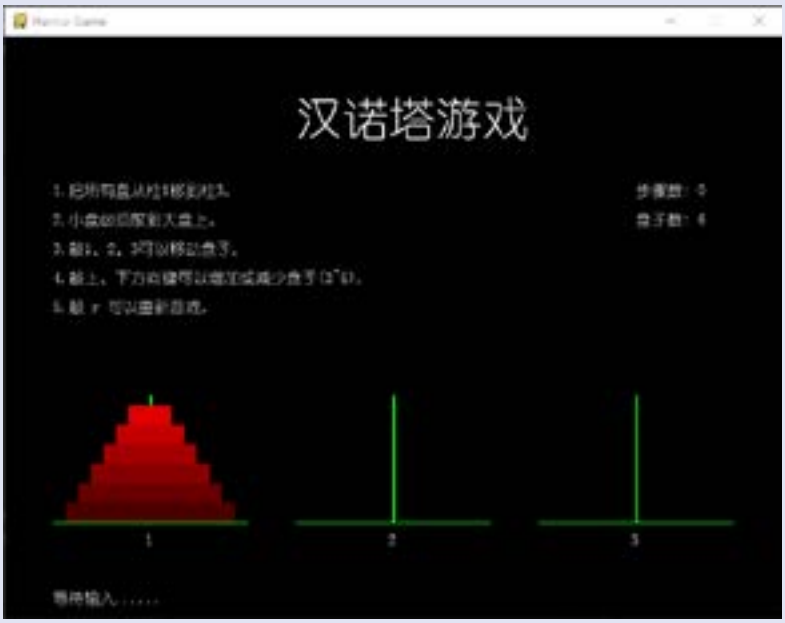
问题分析

问题描述：有三根柱子，其中一根柱子自上而下叠放了逐渐变大的多个圆盘。现在要求人们把柱子上的圆盘移到另一根柱子上，而且移动时要遵守以下规定：

- 一次只能移一个圆盘；
- 每个圆盘只能从一根柱子移到另一根柱子；
- 任何时候都不能把大的圆盘放在小的圆盘上面。

实践探究

运行汉诺塔游戏程序，试着把第1根柱子上的盘子按规则移到第3根柱子上，进一步理解汉诺塔问题。



建模

问题场景数字化

这个问题场景涉及三根柱子，以及多个大小不等的圆盘。很容想到，可以

利用列表来表示柱子，用数字来表示大小不等的圆盘，因此整个游戏的场景可以描述为：

```
# 一个列表中包含三个列表元素，用于表示三根柱子
# 第 1 个列表代表 1 号柱，第 2 个代表 2 号柱，第 3 个代表 3 号柱
p=[ [1,2,3,4,5,6] , [] , [] ]
```

移动圆盘数字化

在两个柱子上移动圆盘的过程，可以用两个列表增删元素来表示：

```
# 把原始列表中的第一个元素移到目标列表的最前面
def move(origin,goal):
    # 获得起始列表最前面的元素，并从起始列表中删除
    disk=p[origin].pop(0)
    # 把圆盘保存到目标列表的最前面
    p[goal].insert(0,disk)
```

移动过程数字化

盘子比较多时，分析移动细节会很困难。操作时可以发现：如果要把 n 号盘从柱子1移到柱子3上，那么1～（$n-1$）号盘肯定在柱子2上；要把 $n-1$号盘从柱子1移到柱子2上，那么1～（$n-2$）号盘肯定在柱子3上；要把 $n-2$号盘从柱子1移到柱子3上，那么1～（$n-3$）号盘肯定在柱子2上……

因此，移动过程可以分为三步。

① 把1～（$n-1$）号盘移到过渡柱上。

② 把 n 号盘移到目标柱上。

③ 把1～（$n-1$）号盘移到目标柱上。

综上所述，可以用下面的代码表达完整的移动过程。

```
# 参数 n 表示圆盘，参数 start 表示起始列表，参数 end 表示目标列表，
# 参数 temp 表示过渡列表，在执行过程中，三者的角色在不断转换
def hanno(n,start , end ,temp):
    # 只有 1 个盘子了，就从 start 移入 end
    if n == 1 : move(start,end)
    elif n > 1:
        hanno(n - 1, start, temp, end)
        move(start,end)
        hanno(n - 1, temp, end, start)
```

程序实现

根据建模信息，用程序求解汉诺塔问题。

1 阅读并尝试理解下面程序的代码。注意，为了便于观察，程序增加了展示操作过程的代码。

```
# 用列表表示三根柱子，最前面的 1 号柱从小到大放着一些圆盘
p=[[1,2,3],[],[]]
# 变量 steps 保存操作步骤
steps=[]

# 从起始位置，移到动目标位置
def move(origin,goal):
    # 获得起始列表最前面的元素，并从起始列表中删除
    disk=p[origin].pop(0)
    # 把圆盘保存到目标列表的最前面
    p[goal].insert(0,disk)
    # 记录操作步骤
    steps.append(f' 把 {disk} 号盘从 {origin+1} 号柱子移到 {goal+1} 号柱子。')

# 递归调用的函数
def hanno (n,start , end , temp) :
    if n == 1 :
        move(start,end)
    elif n > 1:
        hanno(n - 1, start, temp, end)
        move(start,end)
        hanno(n - 1, temp, end, start)

# 展示操作过程
def display():
    for step in steps:  print(step)

# 递归调用求解，可描述为利用过渡柱把 n 个盘从起始柱移到目标柱
hanno(len(p[0]),0,2,1)
display()
```

2 运行程序，观察运行结果。

```
把 1 号盘从 1 号柱子移到 3 号柱子。
把 2 号盘从 1 号柱子移到 2 号柱子。
把 1 号盘从 3 号柱子移到 2 号柱子。
把 3 号盘从 1 号柱子移到 3 号柱子。
把 1 号盘从 2 号柱子移到 1 号柱子。
把 2 号盘从 2 号柱子移到 3 号柱子。
把 1 号盘从 1 号柱子移到 3 号柱子。
```

过程分析

使用递归法时，只要考虑“满足什么条件后停止递归调用”和“每次递归调用时，设置的条件是什么”就可以了。例如，在上面的程序中，当变量n等于1时，停止递归调用。

运行这个程序，先调用的函数不一定会先执行，计算机会层层深入直到满足相关条件后，才停止递归调用。

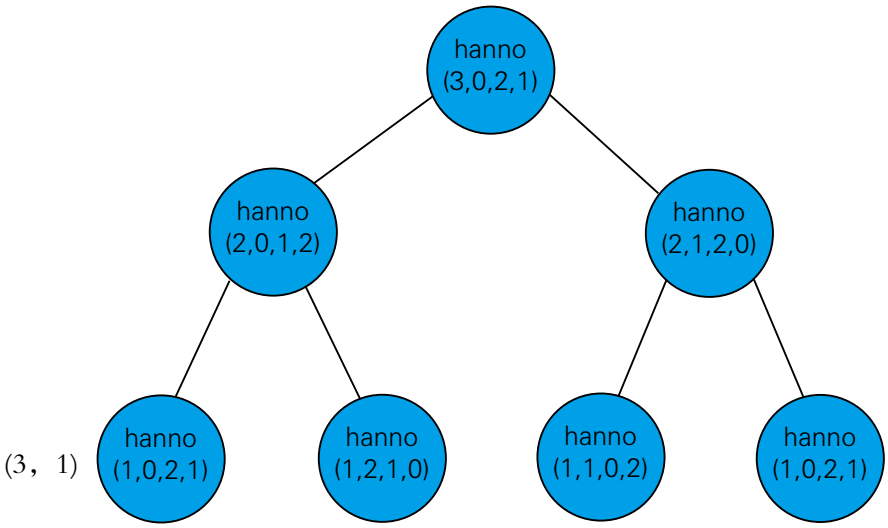
分析递归调用过程。

1 假设有3个圆盘要移动，根据递归思想，填写下表。

调用顺序	圆盘	起始柱	目标柱	过渡柱	执行完毕顺序
1	3	0	2	1	
2	2	0	1	2	
3	1	0	2	1	1
4					
5					
6					
7					

② 修改程序增加调试信息，看看表格中填写的内容是否正确，然后在下图注明调用顺序和执行完毕顺序。例如，hanno(1,0,2,1)的调用序号为3，执行完毕的顺序序号是1，就可以记为(3，1)。提示：在hanno函数的首尾增加下面的代码。

```
print(' 调用 ',n, start, end, temp)
……
print(' 结束 ', n, start, end, temp)
```



解决含有n个圆盘的汉诺塔问题，至少需要多少次操作步骤呢？当n特别大时，递归调用过程会占用大量的计算机资源，甚至可能无法求解。不过，大家可以参考下面的过程进行推算。

① 统计移动 3 、4、5、6、 7 个圆盘时需要多少次操作，然后根据获得的数总结规律，列出公式。

提示：把操作次数转换成二进制数。

② 通过公式计算移动 8 个或 9 个圆盘需多少次操作。

③ 用程序验证移动 8 个或 9 个圆盘需多少次操作，从而验证公式的正确性。

④ 获得能够通过验证的公式后，推算移动n个圆盘至少需要多少次操作。

交流评价

1. 参照下表，对自己的学习过程进行评价。

学习内容	掌握程度
人与计算机解决问题的异同	□ 不知道 □ 知道 □ 熟悉
基于公式的编程思想	□ 不知道 □ 知道 □ 熟悉
基于枚举的编程思想	□ 不知道 □ 知道 □ 熟悉
基于数字仿真的编程思想	□ 不知道 □ 知道 □ 熟悉
基于递归的编程思想	□ 不知道 □ 知道 □ 熟悉
基于机器学习的编程思想	□ 不知道 □ 知道 □ 熟悉
编程求解八数码、八皇后问题	□ 不会 □ 会 □ 熟练
编程求解汉诺塔问题	□ 不会 □ 会 □ 熟练

2. 分组交流、评价活动涉及的程序，谈谈自己对编程思想的认识。

评价内容	评价项目
程序	按要求完成了编写Python程序的任务。
	程序能正常运行，具备指定的功能。
	能够正确解读解决经典问题的程序。
对编程思想的认识	熟悉基于公式、基于枚举的编程思想
	了解通过数字仿真求解的编程思想
	了解递归的概念，感受这一编程思想的特点
	回顾、提升对机器学习的认识

3. 组内评选出具有特色的Python程序或对已有程序的精彩解读，并向全班介绍学习所得。

附录　主要中英文词汇对照表

英　文	中　文	英　文	中　文
AI artificial intelligence	人工智能	nesting	嵌套
algorithm	算法	open	打开
application	应用	operator	运算符
branch	分支	program	程序
character string	字符串	recognition	识别
code	代码	recursive	递归
data	数据	regular expression	正则表达式
data set	数据集	run	运行
dictionary	字典	sequence	序列
file	文件	set	集合
function	函数	software	软件
key word	关键字	sort	排序
label	标签	statement	语句
library	库	string	串
list	列表	train	训练
loop	循环	tuple	元组
module	模块	type	类型
model	模型	variable	变量