

计算机程序之所以功能强大,主要在于程序具有逻辑推理能力,而最基本的逻辑推理操作是条件判断。条件判断可以控制程序执行的流程,因此程序控制结构非常重要。所有程序语言都提供顺序执行、条件判断执行(只执行某一部分程序块,如 if 语句)、循环执行(反复执行某一程序块,如 for 语句)三种最基本的程序控制方式。

3.1 顺序语句

3.1.1 导入语句

1. 模块导入

Python 中,库、包、模块等概念,本质上就是操作系统中的目录和文件。模块导入就是将软件包、模块、函数等程序一次性载入计算机内存,方便程序快速调用。

【例 3-1】 导入语句 `import matplotlib.pyplot` 实际上就是执行 `D:\Python37\Lib\site-packages\matplotlib\pyplot.py` 程序。其中 `matplotlib` 是包名,`pyplot` 是模块名。

Python 程序以函数为主体,程序中有自定义函数、标准库函数、第三方软件包中的函数。程序中自定义函数无须导入。标准库函数分为两部分:一部分内置函数(如 `print()`、`len()` 等 72 个内置函数)和内置方法(如 `str.format()`、`str.join()` 等 45 个内置方法),在 Python 启动时已经导入内存,无须再次导入;另外的大部分标准库(如 `math`、`sys`、`time` 等模块)中的函数,都需要先导入再调用。第三方软件包中所有模块和函数都需要先导入再调用。导入语句一般放在程序开始部分,语法格式如下。

1	<code>import 模块名</code>	# 绝对路径导入,导入软件包或模块
2	<code>import 模块名 as 别名</code>	# 绝对路径导入,别名为简化调用
3	<code>from 模块名 import 函数名或子模块名</code>	# 相对路径导入,导入指定函数或子模块
4	<code>from 模块名 import *</code>	# 导入模块所有函数和变量(慎用)

(1) 以上“模块名”包括库、包、模块、函数等(可采用点命名形式)。

(2) 同一模块多次导入时,它只会执行一次,这防止了同一模块的多次执行。

(3) 绝对路径导入时,调用时用“模块名.函数名()”的形式,优点是避免了同名函数;相对路径导入时,调用时用“函数名()”的形式,非常方便,缺点是容易导致同名函数。

(4) 一个函数库有哪些软件包? 一个软件包有哪些模块? 一个模块有哪些函数? 某个函数如何调用(API,应用程序接口)? 这些问题都必须参考软件包使用指南。

(5) 如果当前目录下存在与导入模块同名的 .py 文件,就会把导入模块屏蔽掉。

2. 模块路径查找

Python 解释器执行 import 语句时,如果模块在当前路径中,模块马上就会被导入。如果需要导入的模块不在当前路径中,Python 解释器就会查找搜索路径。搜索路径是 Python 在安装时设置的环境变量,安装第三方软件包时也会增加搜索路径。搜索路径存储在 Python 标准库中 sys 模块的 path 变量中。

【例 3-2】 查看 sys 模块中的搜索路径,了解 Python 设置了哪些路径。

1	>>> import sys	# 导入标准模块 - 系统功能
2	>>> sys.path	# 查看路径变量
	['', 'D:\\Python37\\Lib\\idlelib', ...](输出略) # 输出当前路径设置	

说明: 以上输出列表中,第一项是空串(''),它代表当前目录。

【例 3-3】 查看 matplotlib.pyplot 模块存放路径。

1	>>> import matplotlib.pyplot	# 导入第三方包 - 绘图
2	>>> print(matplotlib.pyplot)	# 查看模块存放路径
	<module 'matplotlib.pyplot' from 'D:\\Python37\\lib\\site-packages\\matplotlib\\pyplot.py'>	

【例 3-4】 用 sys.path 在路径列表中临时添加新路径 d:\\work\\apps。

1	import sys	# 导入标准模块 - 系统功能
2	sys.path.insert(0, 'd:\\work\\apps')	# 添加新路径,0 为路径插入在开始位置

说明: insert() 函数是动态插入路径,作用范围仅限于当前.py 文件。

3. import 绝对路径导入

【例 3-5】 根据勾股定理 $a^2 + b^2 = c^2$, 计算直角三角形的边长。

1	# E0305.py	# 【计算三角形的边长】
2	import math	# 导入标准模块 - 数学计算
3	a = float(input('输入直角三角形第 1 条边的边长:'))	# 输入边长,转换为浮点数
4	b = float(input('输入直角三角形第 2 条边的边长:'))	# 输入边长,转换为浮点数
5	c = math.sqrt(a*a + b*b)	# 调用数学开方函数计算边长
6	print('直角三角形第 3 条边的边长为:', c)	# 打印计算值
	>>>	# 程序运行结果
	输入直角三角形第 1 条边的边长:3	# 输入边长 3
	输入直角三角形第 2 条边的边长:4	# 输入边长 4
	直角三角形第 3 条边的边长为: 5.0	# 输出运行结果

程序第 1 行,每个模块都有各自独立的变量符号表(命名空间),导入模块(math)中的变量名会放入内存中模块符号表中。程序员可以放心地使用模块内部的全局变量,不用担心变量名的重名问题。使用该模块中的函数或属性时,必须加上模块名称。

程序第 5 行,math.sqrt() 为调用 math(数学)模块中的开方函数 sqrt()。注意,一个模块中往往会有多个函数,这些函数的调用方法(API)可以查看相关使用指南。

4. 对导入软件包或模块取别名

导入软件包时,对包或模块取别名后,调用模块时会简化很多。

【例 3-6】 对导入软件包或模块取别名示例。

1	# E0306.py	# 【绘制散点曲线图 A】
2	import matplotlib.pyplot as plt	# 导入第三方包 - 绘图包.绘图模块(别名 plt)
3	import numpy as np	# 导入第三方包 - 科学计算(别名 np)
4		
5	t = np.arange(0., 5., 0.2)	# 调用科学计算包 NumPy 中的 arange() 函数
6	plt.plot(t, t, 'r--', t, t**2, 'bs', t, t**3, 'g^')	# 调用时,用别名 plt 代替 matplotlib.pyplot
7	plt.show()	# 调用时,用别名 plt 代替 matplotlib.pyplot
	>>>	

程序第 2 行,matplotlib 为第三方软件包名,pyplot 为子模块名;matplotlib.pyplot 表示只导入了 matplotlib 软件包中的绘图子模块 pyplot,因为程序第 6、7 行需要用到 pyplot 模块中的 plot()和 show()函数;matplotlib.pyplot 模块在调用时可简写为 plt(别名)。

程序第 3 行,numpy 为导入软件包全部模块,numpy 包别名为 np。

【例 3-7】 程序与例 3-6 功能完全相同,语句差别是没有对软件包和模块取别名,因此调用这些包的函数时,要书写“包名.模块名.函数名()”的全称路径形式。

1	# E0307.py	# 【绘制散点曲线图 B】
2	import matplotlib.pyplot	# 导入第三方包 - 绘图(没有别名)
3	import numpy	# 导入第三方包 - 科学计算(没有别名)
4		
5	t = numpy.arange(0., 5., 0.2)	# 调用 numpy 包时写全称
6	matplotlib.pyplot.plot(t, t, 'r--', t, t**2, 'bs', t, t**3, 'g^')	# 调用时写"包名.模块名.函数名"
7	matplotlib.pyplot.show()	# 调用时写"包名.模块名.函数名"
	>>>	

5. from 相对路径导入

使用“from 模块名 import 子模块名”相对路径导入时,子模块名可以是软件包中的子模块,也可以是子模块中定义的函数、类、公共变量(如 pi、e)等。

【例 3-8】 导入标准库中开方函数进行计算。

1	>>> from math import sqrt	# 导入标准模块 - 导入 math 模块中的 sqrt() 函数
2	>>> sqrt(2)	# 调用开方函数
	1.4142135623730951	

【例 3-9】 采用“from...import”形式导入时,也可以使用别名(很少使用)。

1	>>> from math import sqrt as kf	# 导入标准模块 - 导入 sqrt 函数,别名为 kf(开方)
2	>>> kf(2)	
	1.4142135623730951	

【例 3-10】 使用“from...import”导入模块中某几个函数,而不是所有函数。

from math import sqrt, sin, cos, log	# 导入标准模块-导入 math 模块中几个函数
--------------------------------------	--------------------------

6. “from 模块名 import *”导入

“from 模块名 import *”方式将导入模块中所有非下画线开始的对象,它可能导致命名空间中存在相同的变量名,这种导入方式容易污染命名空间,尽量谨慎使用。

3.1.2 赋值语句

1. 变量赋值常规方法

Python 是动态程序语言,不需要预先声明变量类型和变量长度,变量值和类型在首次赋值时产生。变量赋值很简单,取一个变量名然后给它赋值即可,语法格式如下。

变量名 = 表达式	# 赋值语句的语法格式
-----------	-------------

赋值语句中,“=”是把等号右边的值或表达式赋给等号左边的变量名,这个变量名就代表了变量的值。

【例 3-11】 变量赋值的正确方法。

1	>>> path = 'd:\test\03\'	# 字符串赋值
2	>>> pi = 3.14159	# 浮点数赋值
3	>>> s = pi * (5.0 ** 2)	# 表达式赋值
4	>>> s = ['张飞', '程序设计', 60]	# 列表赋值
5	>>> b = True	# 布尔值 bool 赋值

Python 赋值语句不允许嵌套(重叠赋值);不允许引用没有赋值的变量;不允许连续赋值;不允许数据类型混用;否则将引发程序异常。

【例 3-12】 变量赋值的错误方法。

1	>>> x = (y = 0)	# 错误,不允许赋值语句嵌套
2	>>> print(y)	# 错误,不允许引用没有赋值的变量
3	>>> x = 2, y = 5	# 错误,不允许连续赋值
4	>>> x = 2; y = 5	# 可运行,但是 PEP 不推荐这种赋值方法
5	>>> s = '日期' + 2018	# 错误,不允许不同数据类型混淆赋值
6	>>> 2 + 3 = c	# 错误,不允许赋值语句颠倒

【例 3-13】 Python 中,一个变量名可以通过重复赋值,定义为不同的数据类型。

1	>>> ai = 1314	# 变量名 ai 赋值为整数
2	>>> ai = '一生一世'	# 变量名 ai 重新赋值为字符串

2. 变量赋值的特殊方法

【例 3-14】 序列赋值实际上是给元组赋值,因此可以赋多个值,赋不同类型的值。

1	>>> A = 1, 2, 3	# 序列赋值,允许一个变量赋多个值(实际为元组)
2	>>> x, y, z = A	# 元组 A 中的值顺序赋给多个变量
3	>>> n, m, k = 10, 6, 5	# 按顺序赋值为:n=10,m=6,k=5
4	>>> n, m, k = 10, 0.5, '字符'	# 按顺序赋值为:n=10,m=0.5,k='字符'(变量为元组)

【例 3-15】 变量链式赋值方法。

1	>>> x = y = k = 0	# 链式赋值,等价于 x=0,y=0,k=0
2	>>> x = y = student = {'姓名':'贾宝玉', '年龄':18}	# 允许变量重新赋值,并改变数据类型

【例 3-16】 变量增量赋值方法。

1	>>> x = 1	# x=1(语义:将 1 赋值给变量名 x)
2	>>> x += 1	# x=2(语义:将 x+1 后赋值给 x;等价于 x=x+1)
3	>>> x *= 3	# x=6(语义:将 x*3 后赋值给 x;等价于 x='***')
4	>>> x -= 1	# x=5(语义:将 x-1 后赋值给 x;等价于 x=x-1)
5	>>> x %= 3	# x=2(模运算,语义:将 x 除以 3 取余数赋值给 x)
6	>>> s = '世界, '	# s = '世界, '(语义:字符串赋值给变量名 s)
7	>>> s += '你好!'	# s = '世界,你好!'(语义:字符串连接)
8	>>> s *= 2	# s = '世界,你好! 世界,你好!'(语义:字符串重复)

【例 3-17】 变量交换赋值方法。

1	>>> x, y = 10, 20	# 变量序列赋值(x=10,y=20)
2	>>> x, y = y, x	# 变量内容交换, $x \leftarrow y, y \leftarrow x$ (x=20,y=10)
3	>>> a, b = '天上', '人间'	# 变量序列赋值(a='天上',b='人间')
4	>>> a, b = b, a	# 字符串变量交换, $a \leftarrow b, b \leftarrow a$ (a='人间',b='天上')

【例 3-18】 对方程 $44x^2 + 123x - 54 = 0$ 求根(方法 1,比较例 3-29、例 3-33)。

一元二次方程标准式: $Ax^2 + Bx + C = 0$

一元二次方程判别式: $\Delta = B^2 - 4AC$

$$\text{解: } \begin{cases} \Delta < 0 \text{ 时, 无解;} \\ \Delta = 0 \text{ 时, } x = B/2A; \\ \Delta > 0 \text{ 时, } x_1 = -((B + \sqrt{\Delta})/2A), x_2 = -((B - \sqrt{\Delta})/2A)。 \end{cases}$$

1	>>> delta = 123**2 - 4*44*(-54)	# 计算一元二次方程判别式 Δ 值
2	>>> x1 = (123 + delta**0.5)/(-2*44)	# 计算方程的根: $x_1 = -((B + \sqrt{\Delta})/2A)$
3	>>> x2 = (123 - delta**0.5)/(-2*44)	# 计算方程的根: $x_2 = -((B - \sqrt{\Delta})/2A)$
4	>>> print('方程的根:x1 = ', x1)	
	方程的根:x1 = -3.18123904902659	
5	>>> print('方程的根:x2 = ', x2)	
	方程的根:x2 = 0.3857845035720447	

3.1.3 输入输出语句

输入是用户告诉程序所需要的信息,输出是程序运行后告诉用户的结果,通常把输入输出简称为 I/O。Python 有三种输入输出方式:表达式、函数和文件。文件输入输出方式在后续章节专门讨论。input() 和 print() 是最基本的输入和输出函数。

1. 用 input() 函数读取键盘数据

input() 函数是一个内置标准函数,它的功能是从键盘读取输入数据,并返回一个字符串(注意,返回值不是数值)。如果希望输入的数据为整数,需要用 int() 函数将输入的数据

转换为整数；如果希望输入的数据为小数，需要用 `float()` 函数将输入的数据转换为小数；也可以用 `eval()` 函数，将输入的数据转换为实数。

【例 3-19】 用 `input()` 函数读取键盘输入数据。

1	>>> s = input('请输入产品名称:') 请输入产品名称:计算机	# 从键盘读取字符串,并且赋值给变量名 s # 注意,即使输入的是数字,结果也是字符串
2	>>> x = int(input('请输入一个整数:')) 请输入一个整数:105	# 从键盘读取一个整数,赋值给变量名 x # 注意,输入小数时会出错
3	>>> y = float(input('请输入一个浮点数:')) 请输入一个浮点数:88.66	# 从键盘读取一个小数,赋值给变量名 y # 注意,输入会自动转换为浮点数
4	>>> z = eval(input('请输入一个数字:')) 请输入一个数字:16.5	# 从键盘读取数字 # 注意,输入字符时会出错

2. 用 `eval()` 函数读取多个数据

【例 3-20】 用 `eval()` 函数从键盘读取多个数据。

1	>>> a, b, c = eval(input('请输入 3 个数字【数据之间用逗号分隔】:')) 请输入 3 个数字【数据之间用逗号分隔】:1,2,3	# 注意,输入个数少于 # 变量个数时会出错
2	>>> x, y = eval(input('数字 x = ')), eval(input('数字 y = ')) 数字 x = 1 数字 y = 2	# 分行输入数据

3. 表达式输出

【例 3-21】 表达式输出方法。

1	>>> '程序' + '设计' '程序设计'	# 字符串表达式输出
2	>>> 5200000 + 1314 5201314	# 数字表达式输出

4. 用 `print()` 函数输出

`print()` 函数是一个内置的打印输出函数,它并不是向打印机输出数据,而是向屏幕输出数据。`print()` 函数有多个参数,可以用来控制向屏幕的输出格式。

【例 3-22】 用 `print()` 函数连续打印输出。

1	print("\u20dd")	# 打印符号(\u20dd 为"明月"符号编码)
2	print("~" * 12)	# 连续打印 12 个"~"字符串
3	print("海上生明月,天涯共此时。")	# 打印字符串
>>> ~~~~~ 海上生明月,天涯共此时。		# 程序运行结果

【例 3-23】 用 `prnt()` 函数输出的方法。

1	>>> love = 5201314	# 变量赋值
2	>>> print('爱情计算值 = ', love)	# 输出提示信息和变量值
	爱情计算值 = 5201314	
3	>>> print("学号\t姓名\t成绩")	# 用制表符\t控制输出空格
	学号 姓名 成绩	# 注意,\t和\n只在引号内才起作用
4	>>> print("休言女子非英物\n夜夜龙泉壁上鸣")	# 用换行符\n控制换行输出
	休言女子非英物	
	夜夜龙泉壁上鸣	

【例 3-24】 用 print()函数连续输出不换行。

1	myList = ['碧云天', '黄花地', '西风紧', '北雁南飞']	# 列表赋值
2	for i in myList:	# 循环输出列表
3	print(' ', i, end = '')	# 参数 end = ''为不换行输出
	>>>碧云天 黄花地 西风紧 北雁南飞	# 程序运行结果

5. 利用占位符控制输出格式

我们经常会遇到“亲爱的×××您好！您××月的电话费是××，余额是××”之类的字符串，其中×××的内容经常变化。所以，需要一种简便的格式化字符串方式。print()函数的输出格式可以用占位符控制。在字符串内部，%是占位符，有几个%，后面就需要几个变量或者值，顺序对应。占位符后的常见情况有 d 为整数、f 为浮点数、s 为字符串、x 为十六进制整数。如果不太确定用什么占位符，%s 永远起作用，它会把任何数据类型转换为字符串。占位符前面的 x 表示占几位，如 10.2f 表示占 10 位，其中小数占 2 位。

【例 3-25】 用占位符控制输出格式案例。

1	>>>'您好，%s，您有 %s 元到账了。' % ('小明', 1000.00)	# % 为占位符,s 为字符串
	'您好，小明，您有 1000.00 元到账了。'	# 第 1 个 %s 对应'小明',其余类推
2	>>> print('教室面积是 %10d 平方米。' % (80))	# %10d = 右对齐,占位符占 10 位
	教室面积是 80 平方米。	
3	>>> print('教室面积是 %-10d 平方米。' % (80))	# %-10d = 左对齐,占位符占 10 位
	教室面积是 80 平方米。	
4	>>> print('教室面积是 %10.2f 平方米。' % (80))	# %10.2f = 右对齐,占位符占 10 位
	教室面积是 80.00 平方米。	# 其中小数占 2 位

6. 利用函数格式化输出

使用{:}.format()函数格式化输出，函数用{:}来代替占位符%。其语法格式如下。

```
{0:[填充字符][对齐参数][宽度]}.format()
```

对齐参数<为左对齐；对齐参数^为居中；对齐参数>为右对齐。

【例 3-26】 {:}.format()函数格式化输出方法。

1	>>>'{:0:>10}'.format('计算机')	# 符号>为右对齐输出,10 表示占 10 位,1 个汉字算 1 位
	' 计算机'	

2	>>>' {0:<10}'.format('计算机')	# 符号<为左对齐输出,10 表示占 10 位
	'计算机'	
3	>>>' {0:^10}'.format('计算机')	# 符号^为居中对齐输出,10 表示占 10 位
	' 计算机 '	
4	>>>' {0: = ^10}'.format('计算机')	# 符号 = 为填充符号,10 表示填充 10 个" = "符号
	' === 计算机 === '	
5	>>>' {0:.4f}'.format(1/3)	# 符号".4f"为 4 位小数输出
	'0.3333'	
6	>>>' {:,}'.format(12369132698)	# 符号","为按千分位格式输出
	'12,369,132,698'	

3.2 条件判断语句

3.2.1 if-else 条件判断语句

程序设计中,经常需要根据特定条件来判断程序运行哪部分代码。在 Python 中,可以用 if 语句来检查程序当前状态,并根据当前状态采取相应的措施。因此,if 语句和循环在编程语言中被称为“控制语句”,用来控制程序执行的顺序。

1. if 语句语法格式

Python 条件判断语句的语法格式如下:

1	if 条件表达式:	# 条件表达式只允许用关系运算符"==" ;不允许用赋值运算符"="
2	语句块 1	# 如果条件为 True,执行语句块 1,执行完后结束 if 语句(不执行语句块 2)
3	else:	# 否则
4	语句块 2	# 如果条件为 False,执行语句块 2,执行完后结束 if 语句(不执行语句块 1)

在 if 语句中,关键词 if 可以理解为“如果”;条件表达式往往就是关系表达式(如 $x >= 60$),条件表达式的值只有 True 或者 False;语句结尾的冒号(:)可以理解为“则”;关键词“else:”可以理解为“否则”。注意,if 语句的冒号(:)不可省略,if 语句块可以有多行,但是 if 语句块内部必须缩进 4 个空格,并且保持垂直对齐。

Python 根据条件表达式的值为 True 还是为 False,来决定怎样执行 if 语句中的代码。如果条件表达式的值为 True,Python 就执行 if 语句块 1;如果条件表达式的值为 False,Python 将忽略语句块 1,选择执行语句块 2。由此可见,程序代码并不是全部都需要执行。这样就使得程序具有了很大的灵活性,为逻辑推理提供了良好基础。

值得注意的是,if 语句中的“else:”为可选语句块。

【例 3-27】 条件判断语句简单案例。

1	x = 80	# 变量赋值
2	if x >= 60:	# 条件判断语句,如果 $x >= 60$ 为真
3	print('成绩及格')	# 则执行本语句,执行完后结束 if 语句(不执行语句 4 和 5)
4	else:	# 否则($x >= 60$ 为假时,跳过语句 3)
5	print('成绩不及格')	# 执行本语句,执行完后结束 if 语句

程序第 2 行,如果 $x \geq 60$ 条件成立(表达式的值为 True),则执行语句 3,执行完后结束 if 语句;否则(表达式的值为 False)跳过语句 3,执行语句 4 和 5,执行完后结束 if 语句。

2. 读取用户输入的数据进行条件判断

【例 3-28】 利用 input()函数读取用户输入的数据。

1	num = input('请输入课程成绩:')	# 数据输入语句
2	if num > 59:	# 条件判断语句
3	print('不错,课程及格了!')	
4	else:	
5	print('成绩尚未及格,同学仍需努力!')	
	>>>	# 程序运行结果
	请输入课程成绩:80	
	Traceback (most recent call last):...(输出略)	# 抛出异常信息

程序初学者很容易犯以上错误,这是因为 input()函数返回的数据是字符串,而字符串不能与 if 表达式中的整数进行比较。

程序第 1 行,将原语句修改为: num = int(input('请输入课程成绩:')),这样从键盘读取的数据通过 int()函数转换为整数,就不会出现数据类型错误了。当然,如果输入的数据是浮点数或英文字符,同样会出现程序运行错误。

【例 3-29】 对一元二次方程 $44x^2 + 123x - 54 = 0$ 求根(方法 2,比较例 3-18、例 3-33)。

Python 可以用 input()函数读取用户输入信息,但是 Python 默认将输入信息保存为字符串形式。所以,需要用 float()函数将输入信息强制转换为浮点数类型,这样在计算时才可以避免出现错误。

1	# E0329.py	# 【解一元二次方程 B】
2	import math	# 导入标准模块 - 数学计算
3		
4	print('请输入方程 A * x ** 2 + B * x + C = 0 的系数:')	
5	a = float(input('二次项系数 A = '))	# 将输入数据转换为浮点数
6	b = float(input('一次项系数 B = '))	
7	c = float(input('常数项系数 C = '))	
8	p = b * b - 4 * a * c	# 计算方程判别式
9	if p < 0:	# 如果判别式小于 0
10	print('方程无解')	
11	exit()	# 函数 exit()为退出程序
12	else:	
13	x1 = (-b + math.sqrt(p)) / (2 * a)	# 计算方程的根 1
14	x2 = (-b - math.sqrt(p)) / (2 * a)	# 计算方程的根 2
15	print('x1 = ', x1, "\t", 'x2 = ', x2)	# 参数"\t" = 制表符(空 4 格)
	>>>	# 程序运行结果
	请输入方程 A * x ** 2 + B * x + C = 0 的系数:	
	二次项系数 A = 44	
	一次项系数 B = 123	
	常数项系数 C = -54	
	x1 = 0.3857845035720447 x2 = -3.18123904902659	

3. 三元条件判断语句

三元运算是有 3 个操作数(左表达式、右表达式、条件表达式)的程序语句。程序中经常用三元运算进行条件赋值,三元运算的语法格式如下。

```
a = 左表达式 if 条件表达式 else 右表达式      # 三元运算语法格式
```

三元语句中,条件表达式的值为 True 时,返回左表达式的值;条件表达式的值为 False 时,返回右表达式的值;返回值赋值在变量 a 中。

【例 3-30】 条件判断语句的三元运算。

1	score = 80	# 操作数赋值
2	s = '及格' if score >= 60 else '不及格'	# 三元为:'及格'、score、'不及格'
3	print('三元运算结果:', s)	
	>>>三元运算结果: 及格	# 程序运行结果

3.2.2 if-elif 多分支判断语句

1. if-elif 多分支判断语句语法

当条件判断为多个值时,可以使用多分支 if-elif 判断语句。elif 是 else if 的缩写,if-elif 语法格式如下。

1	if 条件表达式 1:
2	条件 1 成立时,执行语句块 1
3	elif 条件表达式 2:
4	条件 2 成立时,执行语句块 2
5	elif 条件表达式 3:
6	条件 3 成立时,执行语句块 3
7	else:
8	以上条件都不成立时,执行语句块 4

在以上语法中,if、elif 都需要写条件表达式,但是 else 不需要写条件表达式; if 可单独使用,而 else、elif 需要与 if 一起使用。注意,if、elif、else 行尾都有冒号(:)。

if 语句执行有一个特点,它是从上往下判断的,如果某个判断的值是 True,把该判断对应的语句块执行完后,就忽略掉剩下的 elif 和 else 语句块。

Python 并不要求 if-elif 结构后面必须有 else 代码块。在有些情况下,else 代码块很有用;而在其他一些情况下,使用一条 elif 语句来处理特定情形更清晰。

2. if-elif 多条件判断语句应用案例

【例 3-31】 体重是衡量一个人健康状况的重要标志之一,过胖和过瘦都不利于身体健康。BMI(身体质量指数)是国际上常用衡量人体健康程度的指标,其参考标准如表 3-1 所示。

表 3-1 BMI 参考标准

BMI 分类	WHO 标准	中国参考标准 NAT	相关疾病发病的危险性
体重过低	$\text{BMI} < 18.5$	$\text{BMI} < 18.5$	低(其他疾病危险性增加)
正常范围	$18.5 \leq \text{BMI} < 25$	$18.5 \leq \text{BMI} < 24$	平均水平
超重	$\text{BMI} \geq 25$	$\text{BMI} \geq 24$	增加
肥胖前期	$25 \leq \text{BMI} < 30$	$24 \leq \text{BMI} < 28$	增加
I 度肥胖	$30 \leq \text{BMI} < 35$	$28 \leq \text{BMI} < 30$	中度增加
II 度肥胖	$35 \leq \text{BMI} < 40$	$30 \leq \text{BMI} < 40$	严重增加
III 度肥胖	$\text{BMI} \geq 40.0$	$\text{BMI} \geq 40.0$	非常严重增加

BMI 计算方法为: $\text{BMI} = \text{体重} / \text{身高}^2$, 其中体重的单位为 kg, 身高的单位为 m。

对于这个问题编程的难点在于同时输出国际和国内对应的 BMI 分类。我们可以先对 BMI 指标进行分类, 合并相同项, 列出差别项, 然后利用 if-elif 语句进行编程。

1	# E0331.py	# 【BMI 计算】
2	height = eval(input("请输入您的身高(米):"))	# 输入数据
3	weight = eval(input("请输入您的体重(公斤):"))	# 输入数据
4	BMI = weight / (height ** 2)	# 计算 BMI 值
5	print("您的 BMI 为:{:.2f}".format(BMI))	# 输出 BMI 值
6	who = nat = ""	# who 为国际标准值
7	if BMI < 18.5:	# nat 为国内标准值
8	who, nat = "偏瘦", "偏瘦"	# 根据 BMI 值判断
9	elif 18.5 <= BMI < 24:	# 多重判断
10	who, nat = "正常", "正常"	
11	elif 24 <= BMI < 25:	
12	who, nat = "正常", "偏胖"	
13	elif 25 <= BMI < 28:	
14	who, nat = "偏胖", "偏胖"	
15	elif 28 <= BMI < 30:	
16	who, nat = "偏胖", "肥胖"	
17	else:	
18	who, nat = "肥胖", "肥胖"	
19	print("国际 BMI 标准:'{0}', 国内 BMI 标准:'{1}'".format(who, nat))	# 输出结论
	>>>	# 程序运行结果
	请输入您的身高(米):1.75	
	请输入您的体重(公斤):76	
	您的 BMI 为:24.82	
	国际 BMI 标准:'正常', 国内 BMI 标准:'偏胖'	

3.2.3 if 嵌套语句

在 if 嵌套语句中, 可以把 if-elif-else 结构嵌套在另外一个 if-else 结构中。

if 嵌套语法格式如下:

```

1  if 条件表达式 1:                                # 开始外层 if - else 语句
2      语句块 1
3      if 条件表达式 2:                            # 开始内层 if - elif - else 嵌套语句
4          语句块 2
5      elif 条件表达式 3:
6          语句块 3
7      else:
8          语句块 4                                # 结束内层 if - elif - else 嵌套语句
9  else:
10     语句块 5                                    # 结束外层 if - else 语句

```

【例 3-32】 判断用户输入的整数是否能够被 2 或者 3 整除。

```

1  # E0332.py                                     # 【if 嵌套】
2  num = int(input('请输入一个整数:'))
3  if num % 2 == 0:                               # if 语句块 1 开始,%为模运算
4      if num % 3 == 0:                           # 内层 if 嵌套语句块 2 开始
5          print('输入的数字可以整除 2 和 3')
6      else:                                       # if 语句块 2 部分
7          print('输入的数字可以整除 2,但不能整除 3') # 内层 if 嵌套语句块 2 结束
8  else:                                           # if 语句块 1 部分
9      if num % 3 == 0:                           # 内层 if 嵌套语句块 3 开始
10         print('输入的数字可以整除 3,但不能整除 2')
11     else:                                       # if 语句块 3 部分
12         print('输入的数字不能整除 2 和 3')      # if 语句块 3,语句块 1 结束
>>>                                             # 程序运行结果
输入一个数字:55
输入的数字不能整除 2 和 3

```

程序第 3~12 行为 if 语句块 1; 程序第 4~7 行为 if 语句块 2; 程序第 9~12 行为 if 语句块 3。

【例 3-33】 一元二次方程 $44x^2 + 123x - 54 = 0$ 求根(方法 3,比较例 3-18、例 3-29)。

```

1  # E0333.py                                     # 【解一元二次方程 C】
2  import math                                    # 导入标准模块 - 数学计算
3
4  a = float(input("请输入二次项系数 a:"))        # 将输入信息转换为浮点数
5  b = float(input("请输入一次项系数 b:"))
6  c = float(input("请输入常数项系数 c:"))
7  if a != 0:                                     # 如果判别式不等于 0
8      delta = b**2 - 4 * a * c                    # 计算方程判别式
9      if delta < 0:                               # 如果判别式小于 0
10         print("无根")
11     elif delta == 0:                             # 如果判别式等于 0
12         s = -b/(2 * a)                           # 计算方程的唯一根
13         print("唯一根 x = ",s)
14     else :
15         root = math.sqrt(delta)                  # 计算方程判别式
16         x1 = (-b + root)/(2 * a)                  # 计算方程的根 1
17         x2 = (-b - root)/(2 * a)                  # 计算方程的根 2
18         print("x1 = ", x1, "\t", "x2 = ", x2)     # 参数"\t" = 制表符(空 4 格)

```

>>>	# 程序输出
请输入二次项系数 a:44	
请输入一次项系数 b:123	
请输入常数项系数 c:-54	
x1 = 0.3857845035720447	x2 = -3.18123904902659

3.3 循环语句

3.3.1 for 计数循环

1. for 循环基本结构

循环是为了实现程序中部分语句的重复执行。Python 有两种循环结构,for 计数循环(用于循环次数确定的情况)和 while 条件循环(用于循环次数不确定的情况)。

for 循环可以对序列中每个元素逐个访问(称为遍历)。序列可以是列表、字符串、元组等,也可以用 range()函数生成顺序整数序列(整数等差数列)。

for 循环语法格式 1:

1	for 临时变量 in 序列:	# 将序列中每个元素逐个代入临时变量
2	循环语句块	

【例 3-34】 简单字符串 for 循环语句设计。

1	names = ['唐僧', '孙悟空', '猪八戒', '沙和尚']	# 定义列表(序列)
2	for n in names:	# 将 names 中的元素逐个顺序代入临时变量 n
3	print(n)	# 执行循环语句块
	>>>唐僧 孙悟空 猪八戒 沙和尚	# 注:输出为竖行

【例 3-35】 用 for 循环语句设计一个打字机输出效果。

1	# E0335.py	# 【循环 - 打字机效果】
2	import sys	# 导入标准模块 - 系统功能调用
3	from time import sleep	# 导入标准模块 - 睡眠函数
4		
5	poem = '''\n	# 加\n不空行输出,删除\n空 1 行输出
6	pain past is pleasure.	# 定义字符串
7	过去的痛苦就是快乐。'''	
8	for char in poem:	# 把 poem 序列中每个元素代入临时变量 char
9	sleep(0.2)	# 睡眠 0.2s(用暂停形成打字机效果)
10	sys.stdout.write(char)	# 输出字符串
	>>> pain past is pleasure...(输出略)	# 程序运行结果

2. for 循环执行过程

步骤 1: 循环开始时,for 语句内部计数器自动设置索引号为 0,并读取序列(如列表等)中 0 号元素,如果序列为空,则循环自动结束并退出循环;

步骤 2: 如果序列不为空,则 for 语句读取索引号指定元素,并将它复制到临时变量;

步骤 3: 执行循环中语句块, 循环语句块执行完后, 一次循环执行完毕;
 步骤 4: for 语句内部计数器将索引号自动加 1, 继续访问下一个元素;
 步骤 5: for 语句自动判断, 如果序列中存在下一个元素, 则重复执行步骤 2~5;
 步骤 6: 如果序列中已经没有元素了, for 语句会自动退出当前循环语句。

【例 3-36】 用循环结构计算 1~10 的整数和。

1	sum = 0	# 变量初始化, sum 变量用于存放累加值
2	for x in [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]:	# 按列表进行循环计算
3	sum = sum + x	# 累加计算
4	print('1 + 2 + 3 + ... + 10 = ', sum)	
	>>> 1 + 2 + 3 + ... + 10 = 55	# 程序运行结果

3. for 循环扩展结构

循环程序经常用 range() 函数生成顺序整数序列(称为列表生成式), 语法如下。

range(起始索引号, 终止索引号, 步长)	# 整数序列生成语法
-------------------------	------------

参数“起始索引号”默认从 0 开始, 如 range(5) 等价于 range(0, 5), 共 5 个元素。
 参数“终止索引号”即停止数, 但不包括停止数本身, 如 range(0, 5) 共 5 个元素。
 参数“步长”即每个整数的增量, 默认为 1, 如: range(0, 5) 等价于 range(0, 5, 1)。
 range() 函数返回一个[顺序整数列表]。

for 循环语法格式 2:

1	for 临时变量 in range():	# 将 range() 函数生成的整数序列顺序代入临时变量
2	循环语句块	

【例 3-37】 一个球从 100m 高度自由落下, 每次落地后反弹回原高度的一半再落下。球在第 10 次落地时的反弹高度是多少? 球一共经过的路程是多少?

1	# E0337.py	# 【球的反弹】
2	high = 100	# 高度赋值
3	total = 0	# 总共经过的路程
4	for i in range(10):	# 计数循环, range(10) = 生成 0~10 的整数
5	high /= 2	# 计算反弹高度(反弹高度每次除 2)
6	total += high	# 计算总路程
7	print('球第', i + 1, '次反弹高度:', high)	# 注意, 索引号 i 从 0 开始, 因此需要加 1
8	print('球总共经过的路程为[米]:', total)	
	>>> 球第 1 次反弹高度: 50.0... (输出略)	# 程序运行结果

4. 列表推导式

列表推导是构建列表的快捷方式, 它可读性更好, 而且效率更高。如果想简单、高效地生成满足特定需要的列表, 可以使用列表推导式。

1	表达式 for 临时变量 in 列表	# 语法格式 1
2	表达式 for 临时变量 in 列表 if 条件	# 语法格式 2

【例 3-38】 利用列表推导生成一个等差数列列表。

1	>>> ls = [1, 2, 3, 4, 5, 6, 7, 8]	# 定义列表 ls
2	>>> ls = [i * i for i in ls]	# 用 ls 列表中各元素的平方生成一个新列表
3	>>> ls	# 输出列表 ls
	[1, 4, 9, 16, 25, 36, 49, 64]	

【例 3-39】 假设有一个学生成绩列表,请选出分数在 80~90 分的成绩。

1	>>> scores = [45, 82, 75, 88, 91, 68, 90, 70]	# 定义成绩列表
2	>>> print('成绩过滤:', [s for s in scores if s >= 80 and s < 90])	# 打印列表推导式
	成绩过滤: [82, 88]	

程序第 2 行,列表推导式语句看上去很复杂,并且难以理解。如图 3-1 所示,对复杂语句可以从右到左,按子句逐个分解,这样理解起来就容易多了。

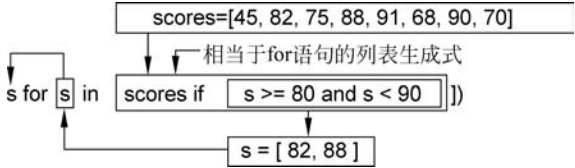


图 3-1 列表推导式结构示意图

3.3.2 while 条件循环

1. while 循环结构

while 循环在运行前先判断条件表达式,如果满足条件再循环。while 语法如下。

1	while 条件表达式:	# 如果条件表达式 = True,则执行循环语句块
2	循环语句块	# 如果条件表达式 = False,则结束循环语句

【例 3-40】 用 while 循环计算 1 到 100 的累加和。

1	n = 100	# 定义循环终止条件
2	sum = 0	# sum 存放累加和,初始化变量
3	counter = 1	# 定义计数器变量,记录循环次数,并作为累加增量
4	while counter <= n:	# 循环判断,如果 counter <= n 为 True,则执行下面语句
5	sum = sum + counter	# 累加和(sum 值 + counter 值后,再存入 sum 单元)
6	counter += 1	# 循环计数(counter <= n 为 False 时结束循环)
7	print('1 到 %d 之和 = %d' % (n, sum))	# 循环外语句,打印累加和
	>>> 1 到 100 之和 = 5050	# 程序运行结果

2. 无限循环

如果循环语句的条件表达式永远不为 False,就会导致出现无限循环(也称为死循环)的情况。无限循环在网络服务端程序设计中非常有用,但是在大部分时候,在程序设计中应当尽量避免。出现无限循环可以通过按 Ctrl + C 组合键,强制退出循环;或者通过关闭 Python 调试窗口强制退出循环。

【例 3-41】 无限循环示例。

1	while True:	# 循环条件表达式永远 = True
2	num = int(input('请输入一个整数:'))	
3	print('你输入的数字是:', num)	
4	print('按[Ctrl + C]键强制退出循环')	
>>>...(输出略)		# 程序运行结果

3. while 循环程序案例**【例 3-42】** 输出简单正三角形。

1	i = 1	# 变量初始化
2	while i <= 5:	# 循环终止条件
3	print(" " * i)	# 打印 * 号
4	i += 1	# 自加运算
>>>		
* ** *** **** *****		

【例 3-43】 输出简单倒三角形。

1	i = 5	# 变量初始化
2	while i >= 0:	# 循环终止条件
3	print(" " * i)	# 打印 * 号
4	i -= 1	# 自减运算
>>>		
***** **** *** ** *		

3.3.3 循环中止**1. 用 continue 语句跳过循环块中剩余语句**

continue 语句用来跳过当前循环块中的剩余语句,然后继续进行下一轮循环。

【例 3-44】 在循环语句中,用 continue 语句跳过某个字符的输出。

1	for s in '人间四月芳菲尽':	# 依次循环取出字符串列表中的字符
2	if s == '芳':	# 如果字符为'芳'
3	continue	# 则跳过这个字符,返回到循环开始处重新循环
4	print(s)	
>>>人 间 四 月 菲 尽		# 注,输出为竖行

2. 用 break 语句强制跳出循环

break 语句可以强制跳出 for 和 while 循环体。break 语句一般与 if 语句配合使用,在特定条件满足时,达到跳出循环体的目的。

【例 3-45】 在循环语句中,用 break 语句强制跳出循环。

1	for s in '人间四月芳菲尽':	# 依次循环取出字符串列表中的字符
2	if s == '芳':	# 如果字符 = "芳"
3	break	# 则强制退出循环(比较与例 3-44 的区别)
4	print(s)	
>>>人 间 四 月		# 注:输出为竖行

【例 3-46】 在“石头-剪刀-布”游戏程序中强制跳出循环。

1	# E0346.py	# 【石头-剪刀-布】
2	import random	# 导入标准模块-随机数
3		
4	guess_list = ['石头', '剪刀', '布']	# 定义字符串列表
5	win_combination = [['布', '石头'], ['石头', '剪刀'], ['剪刀', '布']]	# 定义输赢标准
6	while True:	# 无限循环(死循环)
7	computer = random.choice(guess_list)	# 生成随机数
8	people = input('请输入【石头,剪刀,布】= ').strip()	# 用户输入
9	if people not in guess_list:	# 判断输赢方
10	people = input('请重新输入【石头,剪刀,布】= ').strip()	# 重新输入
11	continue	# 回到循环起始处
12	if computer == people:	# 判断输赢
13	print('平手,再玩一次!')	
14	elif [computer, people] in win_combination:	# 判断输赢
15	print('电脑获胜!')	
16	else:	# 否则,人获胜
17	print('你获胜!')	
18	break	# 人获胜则退出循环
	>>> 请输入【石头,剪刀,布】= 石头 平手,再玩一次! 请输入【石头,剪刀,布】= 布 你获胜!!!	# 程序运行结果

3.3.4 循环嵌套

循环嵌套就是一条循环语句里面还有另外一条循环语句,当两个以上的循环语句相互嵌套时,位于外层的循环结构简称为外循环,位于内层的循环结构简称为内循环。循环嵌套会导致代码阅读性非常差,因此要避免出现三个以上的循环嵌套。

输出一个多行字符串需要两个嵌套在一起的循环。第 2 个循环(内循环)在第 1 个循环(外循环)内部。循环嵌套是每一次外层循环时,都要进行一遍内层循环。

【例 3-47】 利用循环嵌套打印乘法口诀表。

1	# E0347.py	# 【打印乘法口诀表】
2	for i in range(1, 10):	# 外循环打印行(最大 9 行)
3	for j in range(1, i + 1):	# 内循环打印一行中的列(最大 9 列)
4	print('{} × {} = {}'.format(j, i, i * j), end= '')	# 按格式打印输出每一个乘法口诀
5	print()	# 换行
	>>> 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	# 程序运行结果

程序第 2 行,for i in range(1, 10)为外循环,它控制行输出,一共打印 9 行。

程序第 3 行,for j in range(1, i+1)为内循环,它控制一行中每个表达式(如 $1 \times 1 = 1$)的输出,由于循环变量 $i=10$,因此每行最多输出 9 个乘法口诀。

程序第 4 行,print('{}×{}={}{}\t'.format(j, i, i*j), end='')为控制打印格式;j 值对应第 1 个 {}, i 值对应第 2 个 {}, i*j 的值对应第 3 个 {}; end=''为不换行打印。

程序第 5 行,print()语句属于外循环,因此语句缩进与第 3 行 for 语句对齐,外循环每次循环中,它都会执行一次。print()语句没有写任何东西,它只起到换行作用。

【例 3-48】 判断 1~100 有多少个素数,并输出所有素数。

判断素数的方法:公元前 250 年,古希腊数学家厄拉多赛(Eratosthenes)提出了一个构造出不超过 n 的素数算法。它基于一个简单的性质:对正整数 n,如果用 $2 \sim \sqrt{n}$ 的所有整数去除,均无法整除,则 n 为素数。

1	# E0348.py	# 【求素数】
2	import math	# 导入标准模块 - 数学
3	for n in range(1, 100):	# 外循环,n=1~100 的顺序整数
4	for j in range(2, round(math.sqrt(n))+1):	# 内循环,i 为 2 至 sqrt(n)之间的整数
5	if n%j == 0:	# 求余运算:n%j=0 是合数(能整除)
6	break	# 退出内循环
7	else:	# 否则
8	print('素数', n)	# n%j≠0 是素数
	>>>素数 1 素数 2 素数 3...(输出略)	# 程序运行结果(注,输出为竖行)

程序第 4 行,n 为外循环顺序整数,j 为内循环 2 至 sqrt(n)之间的整数;math.sqrt(n)为求 n 开方值;round()为取整数;range()为创建顺序整数。语句为循环筛选素数。

程序第 5 行,n 和 j 求余为 0,则 n 不是素数;如果 n 和 j 求余不为 0,则 n 为素数。

3.3.5 案例：猜数字游戏

猜数字大小是一种古老的密码破译类小游戏,一般由两个人玩,也可以由一个人与计算机玩。下面用 Python 设计这个经典小游戏。在游戏中,应用了以下程序设计知识:变量赋值、函数参数传递、随机数生成、条件判断、循环嵌套、强制退出循环、计数、用户数据输入、错误处理等。

【例 3-49】 “猜年龄”游戏程序如下所示。

1	# E0349.py	# 【游戏 - 猜年龄】
2	import random	# 导入标准模块 - 随机数
3		
4	print('=== 猜猜 mm 芳龄几何 ===')	
5	def judge(x):	# 定义数据类型错误处理函数
6	while not x.isdigit():	# 输入是否为非数值类型
7	print('拜托,不要用脚敲键盘! 还有'+count+'次')	
8	x = input('敲黑板! 好好输[1-60]:')	# 用户重新输入数据

9	num = int(x)	# 对输入数据取整
10	if (num < 0) or (num > 60):	# 检查数据是否超出范围
11	print('请不要乱敲键盘哦,还有' + count + '次')	
12	x = input('敲黑板! 好好输[1-60]:')	
13	judge(x)	# 调用自身,处理输入错误
14	return num	# 则返回用户输入数据
15		
16	T = 'Y'	# 初始化循环条件
17	while (T == 'Y') or (T == 'y'):	# 外循环,判断是否再玩游戏
18	num = random.randint(1, 60)	# randint()生成1~60的随机整数
19	for i in range(0, 7):	# 内循环,range()生成顺序整数
20	if i != 6:	# 判断猜的次数
21	count = str(6 - i)	# 循环计数 - 1
22	print('你有6次机会,还剩' + count + '次')	
23	x = input('猜猜我的年龄多大[1-60]:')	# 用户输入数据
24	x = judge(x)	# 调用函数检测输入是否错误
25	if x == num:	# 如果猜数 = 随机数
26	print('^_^猜对了! 我们做朋友吧')	
27	break	# 强制退出循环
28	else:	
29	if x > num:	# 猜数 > 随机数
30	print('帅哥,我有那么老吗?')	
31	else:	# 否则,猜数 < 随机数
32	print('小弟,比你稍稍大点!')	
33	else:	# 否则,猜的次数等于6次
34	print('【游戏结束】')	
35	break	# 强制退出循环
36	T = input('继续游戏输入 y,回车键退出:')	# 检测是否退出外循环 while
	>>> ... (输出略)	# 程序运行结果

3.3.6 案例：走迷宫游戏

【例 3-50】 利用循环语句和多条件判断语句,设计一个走迷宫游戏。

(1) 迷宫设计。走迷宫程序首先要确定迷宫的宽和高、迷宫矩阵的形式以及迷宫的入口和出口。迷宫矩阵中的每一个元素可以设置为 0(输出为空格)或 1(输出为■),0 表示道路,可走,1 表示墙,走不通。我们用坐标(x,y)表示走迷宫的精灵(输出为☆)。程序首先读入迷宫矩阵数据,然后显示迷宫矩阵。

(2) 输入判断。用循环语句+多条件判断语句检测玩家的键盘输入(w、s、a、d)。玩家在键盘输入中,可能会出现输入错误的情况,程序需要进行错误处理。

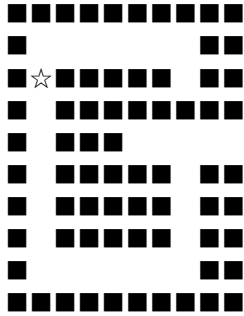
(3) 碰撞检测。通过玩家输入的字符,计算精灵移动的坐标。如果精灵坐标的上、下、左、右是 1(墙),则跳出循环,结束游戏;如果上、下、左、右不是 1,则精灵按输入方向移动。

(4) 胜利判断。精灵移动到出口位置(endx,endy)时,强制跳出循环,玩家胜利。

```

1 # E0350.py # 【游戏 - 走迷宫】
2 # 【定义迷宫路径】
3 my_map = [ # 利用列表嵌套定义迷宫地图
4     [1, 1, 1, 1, 1, 1, 1, 1, 1, 1], [1, 0, 0, 0, 0, 0, 0, 0, 1, 1], # 定义一个 10×10 的二维列表
5     [1, 2, 1, 1, 1, 1, 1, 0, 1, 1], [1, 0, 1, 1, 1, 1, 1, 1, 1, 1], # 1 表示墙, 0 表示路, 2 表示精灵☆位置
6     [1, 0, 1, 1, 1, 0, 0, 0, 0, 0], [1, 0, 1, 1, 1, 1, 1, 0, 1, 1],
7     [1, 0, 1, 1, 1, 1, 1, 0, 1, 1], [1, 0, 1, 1, 1, 1, 1, 0, 1, 1],
8     [1, 0, 0, 0, 0, 0, 0, 0, 1, 1], [1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
9 ]
10 # 【精灵起点与迷宫出口】
11 x, y = 1, 2 # 精灵起始 x,y 坐标
12 endx, endy = 9, 4 # 迷宫出口 x,y 坐标
13 # 【打印迷宫】
14 def print_map(): # 定义迷宫打印函数
15     for m in my_map: # 循环输出全部行
16         for n in m: # 循环输出一行中列的字符
17             if n == 1: # 如果地图数据 = 1
18                 print('■', end = '') # 输出■(墙)
19             elif(n == 0): # 如果地图数据 = 0
20                 print(' ', end = '') # 输出空格(路)end = ""不换行
21             else:
22                 print('☆', end = '') # 否则,输出精灵☆
23         print('') # 打印精灵,并换行
24 print_map() # 打印迷宫地图
25 # 【游戏主循环】
26 while True: # 游戏主循环(无限循环)
27     print('☆表示精灵当前位置') # 提示信息
28     # 【检测玩家输入】
29     key = input('请输入指令【w 上, s 下, a 左, d 右】:') # 等待玩家输入
30     # 【碰撞检测】
31     if key == 'a': # a 为玩家输入
32         x = x - 1 # 计算 x 坐标值
33         if my_map[y][x] == 1: # 碰撞检测,判断 my_map[y][x]
34             print('囧,碰壁了,游戏结束!') # (第 y 列第 x 行)的值是否为 1
35             break # 强制退出循环,结束游戏
36         else: # 否则
37             my_map[y][x], my_map[y][x+1] = \ # 交换坐标值(A,B = B,A)
38             my_map[y][x+1], my_map[y][x] # \为续行符
39             print_map() # print_map()打印迷宫地图
40     elif key == 's': # s 为玩家输入
41         y = y + 1 # 计算 y 坐标值
42         if my_map[y][x] == 1: # x,y = 1 时表示碰壁
43             print('囧,碰壁了,游戏结束!')
44             break # 强制退出循环
45         else:
46             my_map[y][x], my_map[y-1][x] = \
47             my_map[y-1][x], my_map[y][x] # 计算迷宫坐标值
48             print_map() # 打印迷宫地图
49     elif key == 'd': # d 为玩家输入
50         x = x + 1 # 计算 x 坐标值

```

51	if my_map[y][x] == 1:	# x,y=1 时表示碰壁
52	print('囧,碰壁了,游戏结束!')	
53	break	# 强制退出循环
54	else:	
55	my_map[y][x], my_map[y][x-1] = \	
56	my_map[y][x-1], my_map[y][x]	# 计算迷宫坐标
57	print_map()	# 打印迷宫地图
58	if my_map[y][x] == my_map[endy][endx]:	
59	print('恭喜你,过关了!^^')	
60	break	# 强制退出循环
61	elif key == 'w':	# w为玩家输入
62	y = y - 1	# 计算y坐标值
63	if my_map[y][x] == 1:	# x,y=1 时表示碰壁
64	print('囧,碰壁了,游戏结束!')	
65	break	# 强制退出循环
66	else:	
67	my_map[y][x], my_map[y+1][x] = \	
68	my_map[y+1][x], my_map[y][x]	# 计算迷宫坐标
69	print_map()	# 打印迷宫地图
70	# 【检测玩家输入错误】	
71	else:	
72	print('输入指令错误,请重新输入指令:')	# 等待玩家输入
73	continue	# 回到循环头,继续循环
>>>  ☆表示精灵当前位置 请输入指令【w上,s下,a左,d右】:		
		# 程序运行结果

习 题 3

- 3-1 模块导入的功能是什么?
- 3-2 模块导入有哪些原则?
- 3-3 说明语句 `import matplotlib.pyplot as plt` 各部分的功能。
- 3-4 赋值语句应当注意哪些问题?
- 3-5 例 3-18、例 3-29、例 3-33 都是对一元二次方程求根,它们有什么不同?
- 3-6 编程:输入三个整数,请把这三个数按由小到大的排序输出。
- 3-7 编程:学习成绩大于或等于 85 分用“优”表示;75~84 分用“良”表示;60~74 分

用“及格”表示；60 分以下用“不及格”表示。用键盘输入成绩，显示成绩等级。

3-8 编程：兔子问题(斐波那契数列)。有一对兔子，从出生后第 3 个月起，每个月都生一对兔子，小兔子长到第 3 个月后，每个月又生一对兔子。假如兔子都不死，问每个月兔子总数为多少？

3-9 编程：打印 100~1000 的所有“水仙花数”。“水仙花数”是指一个三位数，各位数字的立方和等于该数本身。例如，153 是一个“水仙花数”，因为 $153=1^3+5^3+3^3$ 。

3-10 编程：输入一行字符，统计出其中字母、空格、数字和其他字符的个数。