第 3 章

Python 程序控制结构

程序控制结构是编程中的基本构建块，它们使程序能够根据输入、数据和逻辑条件做出决策，实现不同的任务和功能。

3.1 程序结构

3.1.1 程序流程图

程序流程图是一种用统一规定的图形化符号表示计算机程序执行流程的工具。它通常用于可视化程序的结构和逻辑，以便开发人员更容易理解和分析程序的运行方式。它是程序分析和过程描述的最基本也是最常用的方式。

程序流程图包括各种符号和线条，用于表示不同的程序元素和它们之间的关系。以下是程序流程图中常见的一些元素，对应的图形符号如图 3-1 所示。

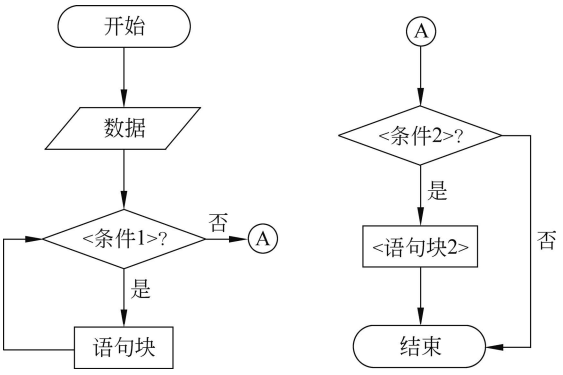


图 3-1　流程图示例

1. 开始或结束符号

通常用一个椭圆形 ⬭ 表示程序逻辑的开始或结束。

2. 流程步骤

流程步骤用矩形框 ▭ 表示，包含具体的操作或计算步骤。这些步骤通常按顺序排列，表示程序的主要逻辑。

3. 连接线

用箭头线条 ⇉ 表示不同元素之间的流程方向，表示程序执行的顺序或分支条件。

4. 条件语句

条件语句用菱形◇表示，表示根据条件表达式的不同结果，程序会选择不同的分支路径。通常在菱形内部写明条件，例如，x > 0。

5. 输入或输出

输入或输出用平行四边形▱表示，表示程序与外部世界的数据输入和输出。

6. 连接点

连接点用于多个流程图的连接，用圆圈○表示，圈内标注要连接子流程图的标号，常用于将多个较小流程图组织成较大流程图。

3.1.2 程序流程结构

程序的执行顺序由程序结构决定，Python 中的程序有三种结构：顺序结构、分支结构和循环结构。

顺序结构按顺序依次执行程序语句。顺序结构是最简单的一种程序结构，它是按照先后顺序依次执行程序中的每一条语句。如第 2 章的例 2-2～例 2-6。

但是在实际应用中，只有顺序结构远不能完成复杂问题，常常会用到分支结构和循环结构。分支结构是根据条件判断来选择相应的程序运行，循环结构可以实现多次运行一条或多条语句。

3.2 分支结构

分支结构是根据条件表达式的值来选择程序运行的语句，分为单分支结构、双分支结构和多分支结构。

此外，在一个分支结构内部可以包含另一个或多个分支结构，即条件语句嵌套。这样就可以完成更为复杂的功能。

3.2.1 单分支结构：if 语句

1. 语法格式

if 语句的语法格式如下：

```
if  <条件表达式> :
    <语句块>
```

2. 说明

（1）<条件表达式>：一个条件用>（大于）、<（小于）、==（等于）、>=（大于或等于）、<=（小于或等于）等来表示数据关系，条件表达式可以是一个或多个条件组合（and , or , not）。

条件表达式的值为布尔型（bool），真为 True，假为 False。

此外，Python 程序语言指定任何非 0 和非空（Null）值为真（True），0 或者 Null 为假（False）。

当条件表达式的值为“真”时，则执行其后的语句块。

当条件表达式的值为“假”时，跳过 if 结构，继续执行 if 结构体后面的语句。单分支结构的流程图如图 3-2 所示。

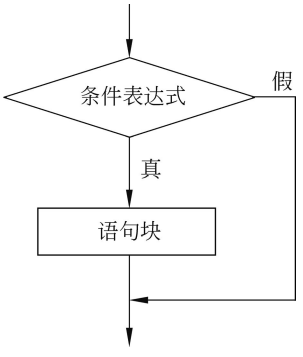


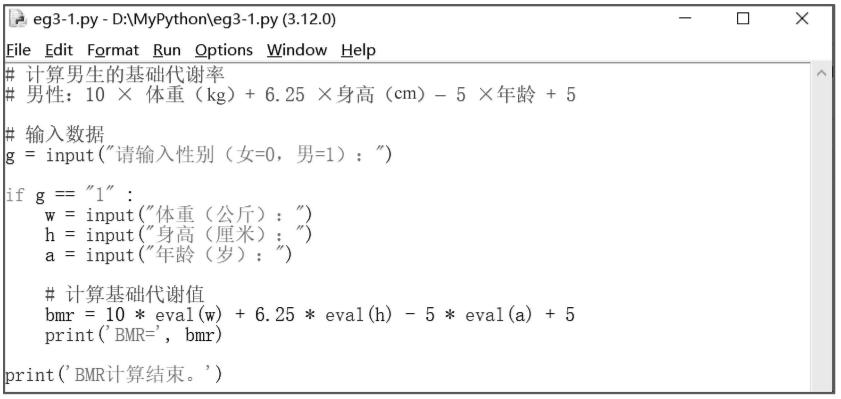
图 3-2 单分支结构的流程图

（2）<语句块>：当<条件表达式>的值为“真”时，所执行的程序语句。程序内容可以多行，以采用缩进方式来区分同一范围。

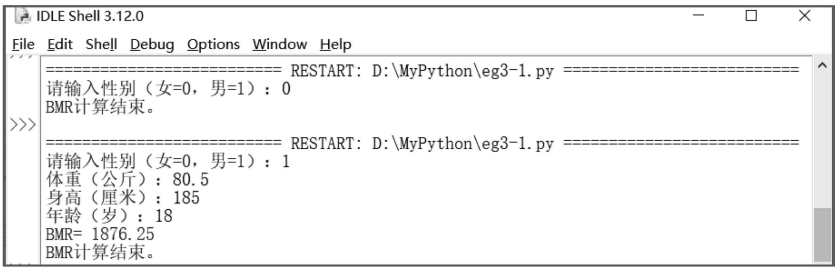
【例 3-1】 在例 2-4 的基础上，添加性别条件判断。只计算男生的基础代谢率。

【任务实现】

任务分析：由于只计算男生的基础代谢率，所以可以使用单分支程序结构。由于存在两种性别情况，所以程序测试两次，分别测试男或女。具体程序代码及运行结果如图 3-3 所示。



（a）程序代码



（b）运行结果

图 3-3　单分支示例

3.2.2　双分支结构：if–else 语句

1. 语法格式

if-else 语句的语法格式如下：

```
if  <条件表达式> :
    <语句块 1>
else:
    <语句块 2>
endif
```

2. 功能

根据条件表达式的真假情况执行不同的代码块，当条件表达式的值为真（True）时，则执行语句块 1；否则执行语句块 2，从而实现两种情况的分类处理。双分支结构流程图如图 3-4 所示。

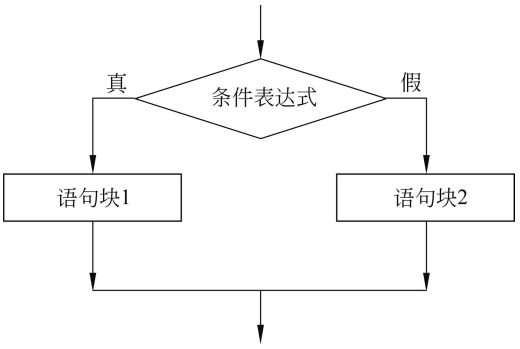


图 3-4　双分支结构流程图

【例 3-2】 在例 3-1 的基础上，完善性别条件判断。根据不同的性别，计算对应的基础代谢率。

【任务实现】

任务分析：由于性别存在两种情况，所以可以使用双分支程序结构。

具体程序代码及运行结果如图 3-5 所示。

```
eg3-2.py - D:\MyPython\eg3-2.py (3.12.0)
File  Edit  Format  Run  Options  Window  Help
# 基础代谢率
# 男性：10 × 体重（kg）+ 6.25 ×身高（cm）– 5 ×年龄 + 5
# 女性：10 × 体重（kg）+ 6.25 ×身高（cm）– 5 ×年龄 - 161

# 输入数据
g = input("请输入性别（女=0，男=1）：")
w = input("体重（公斤）：")
h = input("身高（厘米）：")
a = input("年龄（岁）：")

# 计算基础代谢率
if g == "1":
    bmr = 10 * eval(w) + 6.25 * eval(h) - 5 * eval(a) + 5
else:
    bmr = 10 * eval(w) + 6.25 * eval(h) - 5 * eval(a) -161

# 输出结果
print('BMR=', bmr)
```

（a）程序代码

```
IDLE Shell 3.12.0
File  Edit  Shell  Debug  Options  Window  Help
========== RESTART: D:\MyPython\eg3-2.py ==========
请输入性别（女=0，男=1）：0
体重（公斤）：55.5
身高（厘米）：165
年龄（岁）：18
BMR= 1335.25
>>>
========== RESTART: D:\MyPython\eg3-2.py ==========
请输入性别（女=0，男=1）：1
体重（公斤）：80.5
身高（厘米）：185
年龄（岁）：18
BMR= 1876.25
>>>
```

（b）运行结果

图 3-5　双分支示例

【例 3-3】 简化例 3-2。

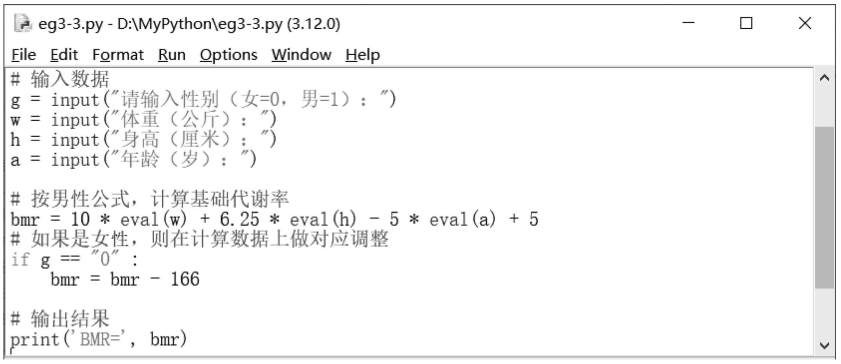
【任务实现】

任务分析：基础代谢率的计算公式如下：

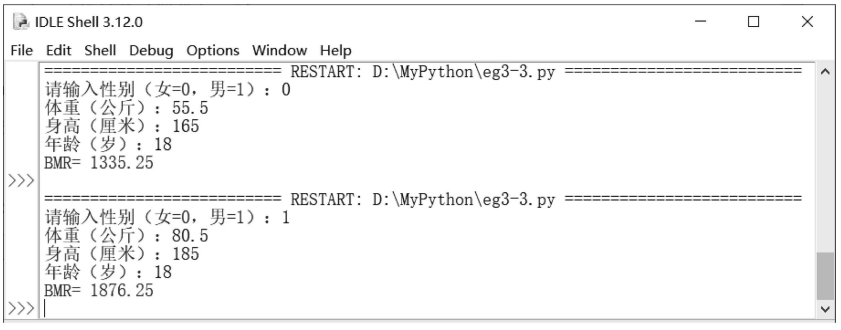
男性：10 × 体重（kg）+ 6.25 ×身高（cm）− 5 ×年龄 +5
女性：10 × 体重（kg）+ 6.25 ×身高（cm）− 5 ×年龄 −161

分析基础代谢率的计算公式可以发现，不同性别的公式就是最后一项不同，所以可以先用男性公式计算，如果是男性则可以直接输出结果，如果是女性则在这个数据的基础上减 166 即可。

简化后的程序就变成了单分支结构，具体程序代码及运行结果如图 3-6 所示。



（a）程序代码



（b）运行结果

图 3-6　例 3-2 程序的简化

3.2.3　多分支结构：if–elif–else 语句

1. 语法格式

if-elif-else 语句的语法格式如下：

```
if  <条件表达式 1> :
    <语句序列 1>
elif <条件表达式 2>:
    <语句块 2>
    ……
elif <条件 n-1> :
    <语句块 n-1>
else :
    <语句块 n>
endif
```

2. 功能

多分支结构允许根据不同条件的多个可能性来分别执行不同的语句块，以实现多种情况的分类处理。该结构流程图如图 3-7 所示。

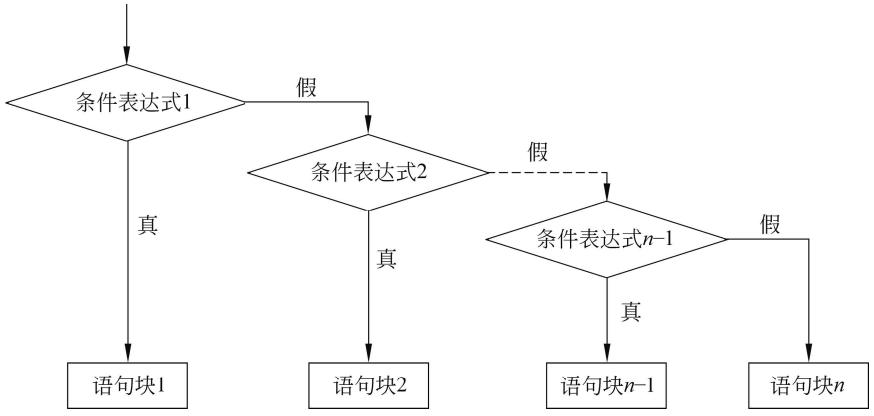


图 3-7　多分支 if-elif-else 结构流程图

【例 3-4】 在例 3-3 的基础上，继续完善性别条件判断。

【任务实现】

任务分析：用户可能输入的性别数据不只是 0（女）或 1（男），还有其他可能，那么可以使用多分支程序结构。具体程序代码及运行结果如图 3-8 所示。

```
# 输入数据
g = input("请输入性别（女=0，男=1）：")
w = input("体重（公斤）：")
h = input("身高（厘米）：")
a = input("年龄（岁）：")

# 按男性公式，计算基础代谢率
bmr = 10 * eval(w) + 6.25 * eval(h) - 5 * eval(a) + 5

if g == "0" :
    print('该女性的BMR=', bmr - 166) # 如果是女性，则在计算数据上做对应调整
elif g == "1" :
    print('该男性的BMR=', bmr)
else:
    print('性别输入错误，无法计算。')
```

（a）程序代码

```
========== RESTART: D:\MyPython\eg3-4.py ==========
请输入性别（女=0，男=1）：0
体重（公斤）：55.5
身高（厘米）：165
年龄（岁）：18
该女性的BMR= 1335.25
>>>
========== RESTART: D:\MyPython\eg3-4.py ==========
请输入性别（女=0，男=1）：1
体重（公斤）：80.5
身高（厘米）：185
年龄（岁）：18
该男性的BMR= 1876.25
>>>
========== RESTART: D:\MyPython\eg3-4.py ==========
请输入性别（女=0，男=1）：2
体重（公斤）：80.5
身高（厘米）：185
年龄（岁）：28
性别输入错误，无法计算。
```

（b）运行结果

图 3-8　多分支示例

【例 3-5】 完善例 3-4，当性别输入数据不对时不再进行其他数据的输入。

【任务实现】

任务分析：用户可能输入的性别数据不是 0（女）或 1（男）时，就不需要输入体重等数据。这时可以使用条件语句的嵌套来实现。

第一层分支语句先进行性别数据判断，符合输入要求的情况下，再进行第二层计算男女不同的基础代谢率。

具体程序代码及运行结果如图 3-9 所示。

```
# 输入性别数据
g = input("请输入性别（女=0，男=1）：")

if ( g != "0" and g != "1"):
    print('性别输入错误，无法计算。')
else:
    #输入数据
    w = input("体重（公斤）：")
    h = input("身高（厘米）：")
    a = input("年龄（岁）：")
    # 按男性公式，计算基础代谢率
    bmr = 10 * eval(w) + 6.25 * eval(h) - 5 * eval(a) + 5
    if g == "0" :
        print('该女性的BMR=', bmr - 166) # 如果是女性，则在计算数据上做对应调整
    elif g == "1" :
        print('该男性的BMR=', bmr)
```

（a）程序代码

```
IDLE Shell 3.12.0
File  Edit  Shell  Debug  Options  Window  Help
>>> ========================== RESTART: D:\MyPython\eg3-5.py ==========================
请输入性别（女=0，男=1）：女
性别输入错误，无法计算。
>>> ========================== RESTART: D:\MyPython\eg3-5.py ==========================
请输入性别（女=0，男=1）：0
体重（公斤）：55.5
身高（厘米）：165
年龄（岁）：18
该女性的BMR= 1335.25
>>> ========================== RESTART: D:\MyPython\eg3-5.py ==========================
请输入性别（女=0，男=1）：1
体重（公斤）：80.5
身高（厘米）：185
年龄（岁）：18
该男性的BMR= 1876.25
>>>
```

（b）运行结果

图 3-9　多分支示例

3.2.4　多分支结构：match–case 语句

Python 3.10 引入了 match-case 语句，也称为“模式匹配”或“结构化匹配”。match-case 语句允许用户根据不同的模式匹配执行不同的代码块，使得多情况处理更为强大和灵活，且代码简洁，提高了可读性。

1. 语法格式

match-case 语句的语法格式如下：

```
match <表达式>  :
    case  <模式 1>:
        <语句块 1>
    case  <模式 2>:
```

```
        <语句块 2>
        ……
    case  <模式 n-1>:
        <语句块 n-1>
    case _ :
        <语句块 n>
```

2. 功能

match-case 可以用于各种不同的模式匹配情况，包括常量、范围、类型、条件等。它的程序流程图如图 3-10 所示。

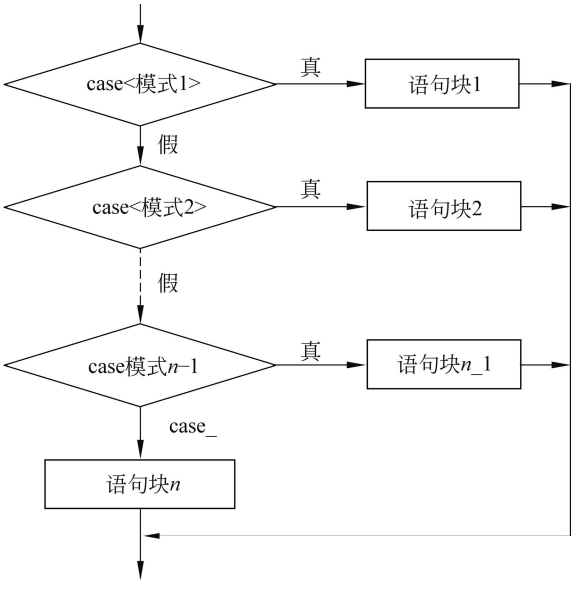


图 3-10　多分支 match-case 结构流程图

match 后面跟要匹配的变量，case 后面跟不同的条件，之后是符合条件需要执行的语句。最后一个“case_”表示默认匹配，即如果前面的 case 条件都没有匹配上就执行这项，类似之前的 else。

3. 说明

case 常用的模式匹配：

① 单一值：直接写具体数据。

② 多个值：用竖线“|”隔开；或者用“in [值 1,值 2,…, 值 n]”列表方式。例如，case x if x in [95,96,97]，其中 x 是临时变量。

③ 范围：用“if”条件模式判断，例如，case x if x >= 90。

【例 3-6】 根据所输入的成绩，按五分制给出不同的评价。对于个别特别优秀的同学，给予特别的评价。

【任务实现】

任务分析：五分制的评价标准为不及格（0～59）、及格（60～69）、中（70～79）、良（80～89）和优（90～100）。对于 95 之上的特别优秀的同学给予特别的评价。这里涉及多种情况分类处理，所以可以考虑用 match-case 语句实现。

具体程序代码如图 3-11 所示。

```
'''
成绩转换
五分制：不及格（0~59）、及格（60~69）、中（70~79）、良（80~89）和优秀 E（90~100）
对于 95 之上的特别优秀的同学给予特别的评价
'''

a = input("请输入成绩：")
score = eval(a)

match score:
    case 100:                          # score 为 100
        print("满分")
    case 98 | 99:                      # score 为 98 或 99
        print("非常优秀")
    case x if x in [95,96,97]:   # score 为 95 或 96 或 97
        print("优秀")
    case x if x >= 90:             # score 大于或等于 90 且小于 95
        print("优")
    case x if x >= 80:             # score 大于或等于 80 且小于 90
        print("良")
    case x if x >= 70:             # score 大于或等于 70 且小于 80
        print("中")
    case x if x >= 60:             # score 大于或等于 60 且小于 70
        print("及格")
    case _:                            #  其他情况：score 小于 60
        print("不及格")
```

图 3-11　成绩转换

【例 3-7】 根据所输入的基础代谢率数据，计算不同活动状态下的每日能量消耗（TDEE，Total Daily Energy Expenditure）。

每日能量消耗估算方法：

每日能量消耗=基础代谢率×活动强度系数

其中，活动强度系数如下：

- 久坐或基本不运动：1.2。
- 轻度活动：1.375，例如，每周运动 1~3 次。
- 中度活动：1.55，例如，每天运动或每周剧烈运动 3~5 次。
- 积极活动：1.725，例如，每周剧烈运动 6~7 次。
- 高强度活动：1.9，例如，每天高强度运动或重体力劳动者。

【任务实现】

这里涉及多种活动状态下的系数处理，所以可以考虑用 match-case 语句实现。具体程

序代码如图 3-12 所示。

```
a = input("请输入基础代谢率BMR: ")
atype = input("请输入日常活动强度类别（1=久坐或基本不运动，2=轻度活动，3=中度活动，4=积极活动，5=高强度活动）：")

BMR = eval(a)

match atype:
    case '1':                    # 久坐或基本不运动
        TDEE = 1.2 * BMR
    case '2':                    # 轻度活动
        TDEE = 1.375 * BMR
    case '3':                    # 中度活动
        TDEE = 1.55 * BMR
    case '4':                    # 积极活动
        TDEE = 1.725 * BMR
    case '5':                    # 高强度活动
        TDEE = 1.9 * BMR
    case _:                      # 其他
        TDEE = 0

print("每日能量消耗TDEE=",TDEE)
```

图 3-12　计算每日能量消耗

【思考题】

参考例 3-5 和例 3-7 的程序实现过程，根据用户所输入的性别、体重、身高、年龄和平时活动状态，计算该用户的每日能量消耗。

3.3　循环结构

循环结构是指在程序中需要反复执行某个功能而设置的一种程序结构。它由循环体中的条件，判断继续执行某个功能还是退出循环。

根据判断条件，循环结构又分为两种形式：遍历循环（for 循环）和条件循环（while 循环）。此外，Python 还提供了控制循环的关键字和语句，如 break（用于提前结束当前循环）、continue（用于结束当此循环）、以及 else（用于在循环正常结束时执行一段代码块）。

3.3.1　遍历循环：for 循环

遍历循环又称为 for 循环，用于遍历可迭代对象（如列表、元组、字符串等）中的元素，并执行代码块。它通常在已知迭代次数且有规律变化时的情况下使用。for 循环是一种先判断后执行的循环结构。

1. 语法格式

for 循环的语法格式如下：

```
for <循环变量> in <遍历对象>:
    <循环体>
```

2. 说明

<遍历对象>：可迭代对象，如列表、元组、字符串等。

<循环体>：当有可遍历的对象元素时所执行的程序语句。循环执行持续到所有元素遍历完全后或遇到 break 语句时才结束循环。循环体中的一部分语句也可以是一个循环结构，即循环嵌套：循环体内又有循环体，这样可以构成双重循环、三重循环……

for 循环结构的流程图如图 3-13 所示。

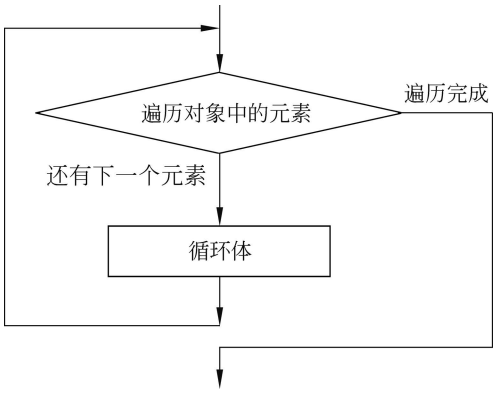


图 3-13　for 循环结构的流程图

3. range()函数

在 for 循环中，通常会用到 range()函数来控制循环的迭代次数。range()函数可以创建一个表示一系列整数的不可变序列。

range()函数的语法格式如下。

```
range(<start>, [stop], <step>)
```

参数说明如下。

① start（可选）：整数，指定序列的起始值，默认为 0。

② stop：整数，指定序列的结束值，但不包括该值。

③ step（可选）：整数，指定步长（序列中相邻整数之间的差），默认为 1。

举例：

```
range(10)：[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
range(1,10)：[1, 2, 3, 4, 5, 6, 7, 8, 9]
range(1,10,2)：[1, 3, 5, 7, 9]
```

4. 常见 for 循环应用方式

遍历字符串的字符：

```
word = "Python"
forch in word:
    print(ch)
```

遍历数字序列：

```
for number in range(10):
    print(number)
```

```
for number in range(1, 10):
    print(number)
```

```
for number in range(1, 10, 2):
    print(number)
```

遍历列表中的元素：

```
university = ["CCMU", "PKU", "THU"]
for u in university:
    print(u)
```

遍历字典的键：

```
university = {"CCMU":1960, "PKU":1898, "THU":1911}
for key in university.keys():
    print(f"{key}: { university.get(key }")
```

遍历字典的键和值：

```
university = {"CCMU":1960, "PKU":1898, "THU":1911}
for key, value in university.items():
    print(f"{key}: {value}")
```

【例 3-8】 求 1～100 的整数之和。

【任务实现】

任务分析：由于是从 1 开始的整数序列且间隔为 1，所以可以用 rang(1,101)的整数序列来进行循环累计求和。具体程序代码如图 3-14 所示。

【例 3-9】 求 1～20 的偶数之积。

【任务实现】

任务分析：由于是从 2 开始的整数序列，且间隔为 2，所以可以用 rang(2, 21, 2)的整数序列来进行循环累计求积。具体程序代码和运行结果如图 3-15 所示。

```
sum = 0
for n in range(1, 101):
    sum = sum + n

print("1~100之和=", sum)
```

图 3-14　1～100 的整数之和

```
prod = 1
for n in range(2, 21, 2):
    prod = prod * n

print("1~20之偶数积=", prod)
```

图 3-15　1～20 的偶数之积

【例 3-10】 制作如图 3-16 所示的九九乘法表。

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

图 3-16　九九乘法表效果

【任务实现】

任务分析：九九乘法表共 9 行从 1～9，可用 for 循环控制乘法表的行数，这为外层循环；在每一行内有 9 列，也从 1～9，在这内部 for 循环用于每一行中的列。

在内层循环中计算两个数的乘积，使用 print()函数打印乘法表的一行。“end="\t"”用于在输出时使用制表符分隔每个乘法表条目，使其看起来更整齐。具体程序代码如图 3-17 所示。

```
for i in range(1, 10):
    for j in range(1, 10):
        product = i * j
        print(f"{i} x {j} = {product}", end="\t")   #\t：制表符间隔
    print()  # 换行
```

图 3-17　九九乘法表程序实现

【例 3-11】根据一条单链上的碱基序列，给出对应的互补链上的碱基序列。A-T、G-C，如果不是这四个字母，用*代替。

在 DNA 中，碱基之间存在互补关系，这是 DNA 双螺旋结构的重要特点之一。碱基互补意味着一种碱基的存在总是伴随着另一种碱基的存在，它们通过氢键结合在一起。在 DNA 中，碱基之间的互补关系如下：

- 腺嘌呤（Adenine，简写为 A）与胞嘧啶（Cytosine，简写为 C）是互补碱基。
- 鸟嘌呤（Guanine，简写为 G）与胸腺嘧啶（Thymine，简写为 T）是互补碱基。

【任务实现】

任务分析：由于碱基序列是字符串，所以可以用 for 循环依次处理每个字符。碱基序列有四种字符，所以要针对不同的字符分类进行处理，适合使用 match-case 多分支结构。

具体程序代码如图 3-18 所示。

```
s = input('请输入碱基链ATGC：') # 原碱基链
t = '' # 互补碱基链

for c in s:
    match c :
        case 'A':
            t += "T"
        case 'T':
            t += "A"
        case 'C':
            t += "G"
        case 'G':
            t += "C"
        case _ :
            t += "*"

print(f'原碱基链为：{s}，互补碱基链为：{t}')
```

图 3-18　碱基互补链

拓展与思考
1965 年 9 月 17 日，以王应睐为首的中国科研团队，历时 6 年完成了结晶牛胰岛素的全合成，这是世界上第一个人工合成的蛋白质。人工合成结晶牛胰岛素的成功，有力地证明了蛋白质的一级结构决定其高级结构的科学论断。 人工合成牛胰岛素凝聚了中国老一辈科学家的智慧，倾注了广大科研人员的心血和汗水。老一辈科学家们的艰苦奋斗、团结协作精神，对“科学强国梦”的执著追求，对科学真理的不懈探索、严谨求实的科学态度，淡泊名利、乐于奉献的精神让人敬佩。

3.3.2　条件循环：while 循环

条件循环又称为 while 循环。while 循环也是一种先判断后执行的循环结构。

1. 语法格式

while 循环的语法格式如下：

```
while <条件表达式>:
    <循环体>
```

2. 说明

<条件表达式>：条件表达式的值为布尔型（bool），真为 True，假为 False。此外，Python 程序语言指定任何非 0 和非空（Null）值为真（True），0 或者 Null 为假（False）。所以，有时候可以看到 while True 或者 while 1 这样的永远为真的判断条件。

<循环体>：当在条件表达式为真时所执行的程序语句。循环执行持续到条件表达式为假或遇到 break 语句时才结束循环，所以要注意循环变量的变化或循环条件结束的条件，否则容易产生“死循环”。此外，循环体中的一部分语句也可以使用循环嵌套。

while 循环结构的流程图如图 3-19 所示。

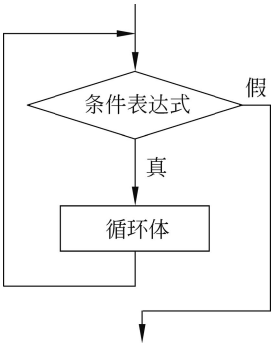


图 3-19　while 循环结构的流程图

【例 3-12】 用 while 循环改写例 3-8 的程序，实现 1~100 的整数之和。

【任务实现】

任务分析：在 for 循环中，系统会自动遍历每个元素，但在 while 循环中，循环条件中的循环变量值是不会自动变化的，所以要特别注意循环条件中循环变量的变化设置。

具体程序代码如图 3-20 所示。

```
sum = 0
n = 1

while n<=100:
    sum += n
    n += 1
print("1~100之和=", sum)
```

图 3-20　用 while 循环计算 1~100 整数和

【例 3-13】 依次输入多个数据，直到输入“q”表示结束。将所输入的数据（不包含“q”字符）存为列表，打印输出时数据用逗号隔开。

【任务实现】

任务分析：可以采用循环方式接收依次输入的数据并保存，要考虑除了“q”字符之外才可保存下来。打印输出要考虑最后一个数据末尾不加逗号。

具体程序代码和运行结果如图 3-21 所示。

【思考题】

在接收数据时，用了两次判断，是否有改进方法只判断一次？

```
user_input = ""  # 接收一次所输入的数据
dataList = []    # 存储所有输入的数据

# 依次输入数据并保存(不包含"q")
while user_input != "q":
    user_input = input("请输入数据，输入“q”结束：")
    if user_input != "q":
        dataList.append(user_input)

# 打印所保存的数据
dlen = len(dataList)
if ( dlen<1):
    print("没有输入有效数据。")
else:
    print("输入的数据为：", end="")
    for i in range(dlen):
        if i == dlen - 1:
            print(dataList[i], end=".")
        else:
            print(dataList[i], end=', ')
```

（a）程序代码

```
========== RESTART: D:\MyPython\eg3-13.py ==========
请输入数据，输入“q”结束：q
没有输入有效数据。
>>>
========== RESTART: D:\MyPython\eg3-13.py ==========
请输入数据，输入“q”结束：1
请输入数据，输入“q”结束：q
输入的数据为：1.
>>>
========== RESTART: D:\MyPython\eg3-13.py ==========
请输入数据，输入“q”结束：1
请输入数据，输入“q”结束：2
请输入数据，输入“q”结束：3
请输入数据，输入“q”结束：4
请输入数据，输入“q”结束：5
请输入数据，输入“q”结束：q
输入的数据为：1, 2, 3, 4, 5.
>>>
```

（b）运行结果

图 3-21　依次接收多个数据保存并打印输出

【例 3-14】 现已有 5 位测试者的基本数据（见表 3-1），依次计算这些学生的基本代谢率及每日能量消耗。

表 3-1　测试者基本数据

编号	性别 （女=**0**，男=**1**）	体重 **/kg**	身高 **/cm**	年龄 /岁	平日活动强度 （分类 **1~5**，见例 **3-4**）
202301	0	55.5	165	18	2
202302	1	80.5	185	18	4
202303	0	50.2	161.3	23	1
202304	1	72.3	175.4	19	3
202305	1	76.6	178.5	20	5

【任务实现】

任务分析：涉及多位测试者，可用循环遍历实现。具体程序代码和运行结果如图 3-22 所示。

【思考题】

在打印输出结果时，把“性别”的数据 0 或 1 改为“男”或“女”，把“平日活动强度”的数字数据改为对应的文字描述。

```
    # 构建数据集列表：性别，体重，身高，年龄，平日活动强度
    data = list()
    data.append([202301,0, 55.5, 165, 18, 2])
    data.append([202302,1, 80.5, 185, 18, 4])
    data.append([202303,0, 50.2, 161.3, 23, 1])
    data.append([202304,1, 72.3, 175.4, 19, 3])
    data.append([202305,1, 76.6, 178.5, 20, 5])

    # 变量赋初值
    BMR = 0   #基础代谢率
    TDEE = 0  #每日能量消耗
    result = dict()    # 保存所有测试者数据的集合,如：{202304:{"data":[], BRM:, TDEE: },...}

    # 输出结果显示的表头
    print(f"{'编号':^4}{'性别':^4}{'体重':^2}{'身高':^8}{'年龄':^4}{'平日活动强度':^8}{'基础代谢
率'}{'每日能量消耗':^8}")

    # 遍历所有测试者数据，并计算每一位测试者的 BMR 和 TDEE
    i = 0  #循环变量赋初值
    while i <len(data):
        # 依次取每一位测试者的数据
        pdata = data[i]
        # 先按男性公式，计算基础代谢率
        BMR = 10 * data[i][2] + 6.25 * data[i][3] - 5 * data[i][4] + 5
        # 如果是女生，则在计算数据上做对应调整
        if data[i][1] == 0 :
            BMR = BMR - 166
        # 计算每日能量消耗
        match data[i][5]:
            case 1:  # 久坐或基本不运动
                TDEE = 1.2 * BMR
            case 2:  # 轻度活动
                TDEE = 1.375 * BMR
            case 3:  # 中度活动
                TDEE = 1.55 * BMR
            case 4:  # 积极活动
                TDEE = 1.725 * BMR
            case 5:  # 高强度活动
                TDEE = 1.9 * BMR
            case _:  # 其他
                TDEE = 0

        # 构建每位测试者的数据集合 {"data":[,,,], "BRM":, "TDEE": }
        persondata = dict()
        persondata["data"] = [data[i][1], data[i][2], data[i][3], data[i][4], data[i][5]]
        persondata["BMR"] = BMR
        persondata["TDEE"] = TDEE
        # 将每位测试者的所有数据加入到结果集合中 { 202301:{"data":[], BRM:, TDEE: }, 202302:{},...}
        result[data[i][0]] = persondata

        # 输出数据及计算结果
        print(f"{data[i][0]:^6}{data[i][1]:^5}{data[i][2]:^6}{data[i][3]:^9.1F}
{data[i][4]:^6}{data[i][5]:^13}{BMR:^11.2F}{TDEE:^14.2F}")
        # 循环变量+1，定位下一条数据
        i += 1
```

（a）程序代码

图 3-22　计算基本代谢率及每日能量消耗

```
========================= RESTART: D:\MyPython\eg3-14.py =================
 编号   性别 体重    身高    年龄   平日活动强度 基础代谢率 每日能量消耗
202301   0   55.5   165.0   18        2        1335.25     1835.97
202302   1   80.5   185.0   18        4        1876.25     3236.53
202303   0   50.2   161.3   23        1        1234.12     1480.95
202304   1   72.3   175.4   19        3        1729.25     2680.34
202305   1   76.6   178.5   20        5        1786.62     3394.59
>>>
```

（b）运行结果

图 3-22　（续）

3.4　循环控制

当程序在执行分支结构或循环结构时，可能遇到特殊情况要中断处理，或提前结束当前循环的情况，Python 提供了相应关键字和语句，如 break（用于提前结束循环）、continue（用于结束当次循环）、以及 else（用于在循环正常结束时执行一段代码块）。

3.4.1　结束当前循环：break

在 while 循环或 for 循环中使用 break 语句可以提前终止当前循环的执行，即使循环条件仍然为真，或 for 循环遍历对象中仍有未遍历的元素。当 break 语句被执行时，程序会立刻跳出当前循环并继续执行循环后的代码。“循环-break”结构的流程图如图 3-23 所示。

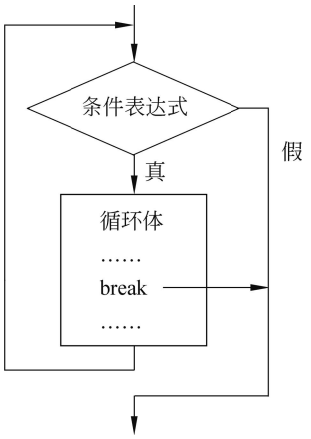


图 3-23 循环- break 结构的流程图

【例 3-15】 改进例 3-13，实现在接收数据时，只判断一次。

【任务实现】

任务分析：可以采用“while True”语句来一直循环接收依次输入的数据，当输入“q”字符时使用 break 结束循环，从而实现接收数据的结束。具体程序代码如图 3-24 所示。

【例 3-16】 一张厚度为 0.1 毫米的足够大的纸，对折多少次以后能达到珠穆朗玛峰的高度？

【任务实现】

任务分析：可以采用“while True”语句来一直循环接收依次输入的数据，当厚度超过珠穆朗玛峰的高度时用 break 结束循环。具体程序代码和运行结果如图 3-25 所示。

```
user_input = ""  # 接收一次所输入的数据
dataList = []    # 存储所有输入的数据

# 依次输入数据并保存(不包含"q")
while True:
    user_input = input("请输入数据，输入"q"结束：")
    if user_input == "q":
        break
    else:
        dataList.append(user_input)

# 打印所保存的数据
dlen = len(dataList)
if ( dlen<1):
    print("没有输入有效数据。")
else:
    print("输入的数据为：", end="")
    for i in range(dlen):
        if i == dlen - 1:
            print(dataList[i], end=".")
        else:
            print(dataList[i], end=', ')
```

图 3-24　循环接收数据

```
n = 0                    # 对折次数
hd = 0.0001              # 纸厚，单位米

while True:
   if hd> 8844.43:       # 超过珠峰高度就停止循环
      break
   else:
      hd *= 2            # 对折一次厚度翻倍
      n += 1             # 对折次数加 1
print(f"纸对折{n}次后的厚度为{hd:.2f}米,超过了珠穆朗玛峰。")
```

（a）程序代码

```
========== RESTART: D:\MyPython\eg3-16.py ==========
纸对折27次后的厚度为13421.77米,超过了珠穆朗玛峰。
>>>
```

（b）运行结果

图 3-25　循环接收数据

拓展与思考

“合抱之木，生于毫末；九层之台，起于累土；千里之行，始于足下”。一张 0.1 毫米厚的纸，如果足够大，则对折 27 次以后就能超过珠穆朗玛峰的高度。所以，只要我们不断努力，假以时日，也会终有所成的。

【例 3-17】 查找某测试者是否在测试集中。

【任务实现】

任务分析：可以采用 for 循环来遍历数据集，当遇到所要查找的测试者编号时使用 break 结束循环。具体程序代码和运行结果如图 3-26 所示。

```
numbers = [202301, 202302, 202303, 202304, 202305]
target = eval(input('请输入要查找的测试者编号：'))
index = 0

fornum in numbers:
    if numbers[index] == target:
        print(f"找到测试者：{target}，序号为：{index + 1}。")
        break
    index += 1

print("查找结束。")
```

（a）程序代码

```
======================== RESTART: D:\MyPython\eg3-17.py ========================
请输入要查找的测试者编号：2023001
查找结束。

======================== RESTART: D:\MyPython\eg3-17.py ========================
请输入要查找的测试者编号：202303
找到测试者：202303，序号为：3。
查找结束。
```

（b）运行结果

图 3-26　查找某测试者

【思考题】

在输出结果时，如果没有找到该患者数据时，如何实现对应的输出说明？

3.4.2　结束当次循环：continue

continue 语句用于跳过当前循环迭代中的剩余代码，并回到循环的开始，继续执行下一次迭代。从而实现在不终止整个循环的情况下，跳过某些迭代步骤。

continue 用来结束当前当次循环，即跳出循环体中后面尚未执行的语句，但不跳出当前循环。使用 continue 语句可以减少代码重复和逻辑嵌套，从而提高代码效率。

循环-continue 结构的流程图如图 3-27 所示。

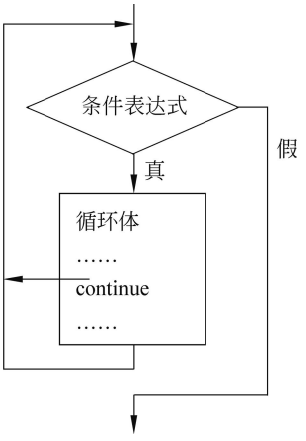


图 3-27　循环- continue 结构的流程图

【例 3-18】输出 10～50 中的不能被 7 整除的整数。

【任务实现】

任务分析：依次判断每个数，当某数能被 7 整除则不输出此数，继续下一次循环。

具体程序代码和运行结果如图 3-28 所示。

```
for x in range(10,50):
    if x % 7 == 0:
        continue
    print(x, end=' ')
```

（a）程序代码

```
======================== RESTART: D:\MyPython\eg3-18.py ========================
10 11 12 13 15 16 17 18 19 20 22 23 24 25 26 27 29 30 31 32 33 34 36 37 38 39 40
 41 43 44 45 46 47 48
```

（b）运行结果

图 3-28　输出 10～50 中的不能被 7 整除的整数

【思考题】

如果将程序中的 continue 改为 break，则输出结果是什么？

【例 3-19】 从患者信息列表中，显示年龄大于 60 岁的老年患者信息。

【任务实现】

具体程序代码和运行结果如图 3-29 所示。

```
    # 患者信息
    patients = [{"name": "张三", "age": 45, "gender": "男", "condition": "高血
压"},
            {"name": "李四", "age": 68, "gender": "女", "condition": "糖尿病"},
            {"name": "王五", "age": 75, "gender": "女", "condition": "关节炎"},
            {"name": "赵六", "age": 52, "gender": "男", "condition": "心脏病"} ]

    # 大于 60 岁的存储老年患者信息
    elderly_patients = []
    # 循环处理每一位患者
    for patient in patients:
        if patient["age"] <= 60:
            continue        # 跳过年龄小于或等于 60 岁的患者
        elderly_patients.append(patient)
    print("年龄大于 60 岁的老年患者：")
    for patient in elderly_patients:
        print(f" 姓 名 ： {patient['name']}, 年 龄 ： {patient['age']}, 性 别 ：
{patient['gender']}, 疾病: {patient['condition']}")
```

（a）程序代码

```
======================== RESTART: D:\MyPython\eg3-19.py ========================
年龄大于60岁的老年患者：
姓名: 李四, 年龄: 68, 性别: 女, 疾病: 糖尿病
姓名: 王五, 年龄: 75, 性别: 女, 疾病: 关节炎
```

（b）运行结果

图 3-29 显示年龄大于 60 岁的患者信息

3.4.3 else 语句

在 Python 中，else 语句除了可以和 if 一起使用外，也可以和 for 或 while 循环一起使用。else 语句提供了一种在循环正常结束时执行特定代码的方式。

1. 语法格式

else 语句的语法格式如下：

```
for <循环变量> in <遍历对象>:
    <循环体>
else:
    <语句块>
```

或

```
while <条件表达式>:
    <循环体>
else:
    <语句块>
```