第3章

流程控制

拓展阅读

学习目标

- 掌握条件语句的用法，能够根据需求使用合适的条件语句
- 掌握循环语句的用法，能够根据需求使用合适的循环语句
- 掌握跳转语句的用法，能够在循环中使用跳转语句跳出本次循环或者结束循环

程序中的语句默认会按照自上而下的顺序逐条执行，但通过一些特定的语句可以更改语句的执行顺序，使之产生跳跃、回溯等现象，进而灵活地控制程序的执行流程。Python 中用于实现流程控制的特定语句主要分为条件语句、循环语句和跳转语句。本章将结合这些特定语句介绍与流程控制相关的知识。

3.1 条件语句

在现实生活中，不同的条件可能会引发不同的情境。比如，在 12306 网站购票时，用户只有通过身份验证才可以购票。在实际开发中，我们可以使用条件语句为程序设置条件，产生与条件匹配的分支，从而有选择地执行对应分支的语句。本节将针对条件语句的相关知识进行详细讲解。

3.1.1 if 语句

if 语句是最简单的条件语句，该语句由关键字 if、判断条件和英文冒号组成，if 语句和从属于该语句的代码段可组成选择结构，其语法格式如下：

```
if 判断条件:
    代码段
```

以上语法格式中的 if 关键字和英文冒号分别标识 if 语句的起始和结束，判断条件与 if 关键字以空格分隔，代码段通过缩进与 if 语句产生关联。

执行 if 语句时，若 if 语句的判断条件成立，即判断条件的布尔值为 True，则执行之后的代码段；若 if 语句的判断条件不成立，即判断条件的布尔值为 False，则跳出选择结构，继

续向下执行。if 语句的执行流程如图 3-1 所示。

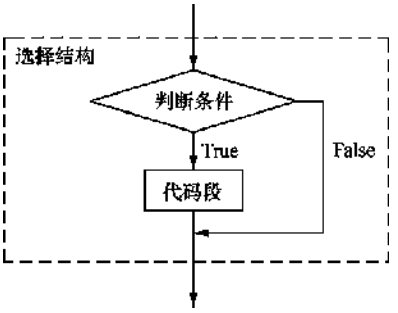

图3-1　if语句的执行流程

接下来，使用 if 语句实现一个考试成绩评估程序：如果考试成绩不低于 60 分，那么将此成绩评估为考试及格，假设小明的考试成绩为 88 分，输出小明的成绩评估结果。示例代码如下：

```
score = 88
if score >= 60:
    print("考试及格！")
```

运行代码，结果如下所示：

```
考试及格！
```

从上述输出结果可以看出，程序执行了 if 语句的代码段。

将以上示例代码中变量 score 的值修改为 55，再次运行代码，控制台中没有任何输出结果，说明程序未执行 if 语句的代码段。

3.1.2　if-else 语句

3.1.1 小节介绍的 if 语句只能处理满足条件的情况，但一些场景中不仅需要处理满足条件的情况，也需要对不满足条件的情况做特殊处理。因此，Python 提供了可以同时处理满足和不满足条件的情况的 if-else 语句。if-else 语句的语法格式如下：

```
if 判断条件:
   代码段 1
else:
   代码段 2
```

执行 if-else 语句时，如果判断条件成立，执行 if 语句之后的代码段 1，否则执行 else 语句之后的代码段 2。if-else 语句的执行流程如图 3-2 所示。

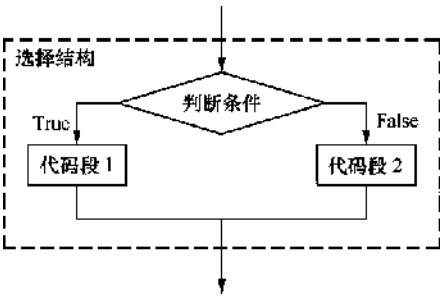

图3-2　if-else语句的执行流程

接下来，使用 if-else 语句优化考试成绩评估程序，使得该程序可以兼顾考试及格和考试不及格这两种评估结果。考试成绩评估程序优化后的代码如下：

```
score = 88
if score >= 60:
    print("考试及格！")
else:
    print("考试不及格！")
```

上述代码中，首先定义了一个表示考试成绩的变量 score，并将其赋为 88，然后使用 if-else 语句判断 score 的值是否大于或等于 60，其中 if 语句处理了 score 的值大于或等于 60 的情况，输出“考试及格!”；else 语句处理了 score 的值的其他情况，输出“考试不及格!”。

运行代码，结果如下所示：

```
考试及格！
```

将以上示例代码中变量 score 的值修改为 55，再次运行代码，结果如下所示：

```
考试不及格！
```

通过比较两次的输出结果可知，程序第一次执行了 if 语句后的代码段，输出了“考试及格!”；程序第二次执行了 else 语句后的代码段，输出了“考试不及格!”。

3.1.3 if-elif-else 语句

根据 3.1.2 小节的考试成绩评估程序可知，该程序只能评估考试及格和不及格的情况，但实际评估成绩时可能会将考试成绩划分为“优秀”“良好”“中等”“差”4 个等级。if-else 语句局限于两种情况，像这种存在 4 个等级的场景无法通过 if-else 语句进行处理。为了处理类似上述的一个事项涉及多种情况的场景，Python 提供了可以产生多个分支的 if-elif-else 语句。if-elif-else 语句的语法格式如下所示：

```
if 判断条件 1:
    代码段 1
elif 判断条件 2:
    代码段 2
elif 判断条件 3:
    代码段 3
…
elif 判断条件 n-1:
    代码段 n-1
else:
    代码段 n
```

以上语法格式的 if 关键字与判断条件 1 构成一个分支，elif 关键字与其他判断条件构成其他的任意个分支，else 关键字构成最后一个分支；每个分支与代码段之间均采用缩进的形式进行关联。

执行 if-elif-else 语句时，若 if 语句的判断条件 1 成立，执行代码段 1；若 if 语句的判断条件 1 不成立，判断 elif 语句的判断条件 2，判断条件 2 成立则执行代码段 2，否则继续向下执行。以此类推，直至所有的判断条件均不成立，执行 else 语句之后的代码段。if-elif-else 语句的执行流程如图 3-3 所示。

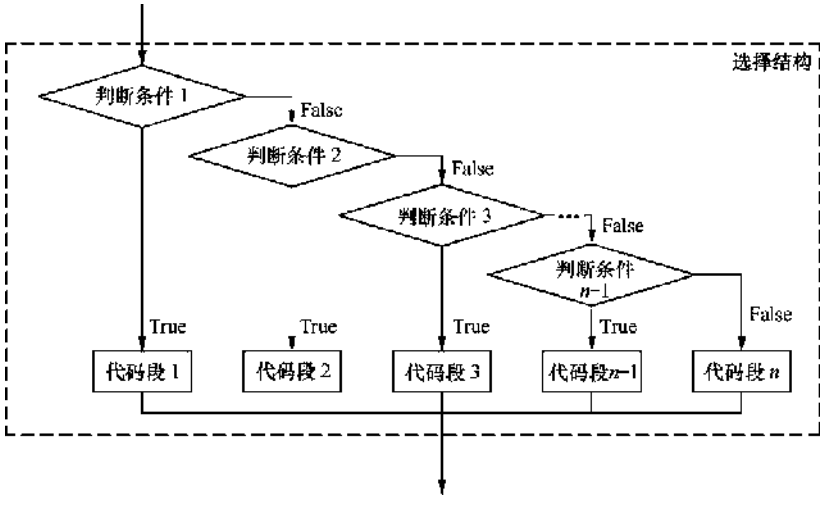

图3-3 if-elif-else语句的执行流程

接下来，使用 if-elif-else 语句优化考试成绩评估程序，使得该程序可以根据考试成绩做出“优秀”“良好”“中等”“差”这 4 个等级的评估，评估标准为：考试成绩不低于 85 分时，评估结果为“优秀”；考试成绩低于 85 且不低于 75 分时，评估结果为“良好”；考试成绩低于 75 且不低于 60 分时，评估结果为“中等”；考试成绩低于 60 分时，评估结果为“差”。

考试成绩评估程序优化后的代码如下：

```
score = 88
if score >= 85:
    print("优秀")
elif 75 <= score < 85:
    print("良好")
elif 60 <= score < 75:
    print("中等")
else:
    print("差")
```

上述代码中，首先定义了一个表示考试成绩的变量 score，并将其赋为 88，然后使用 if-elif-else 语句根据 score 的值分不同情况进行处理，其中 if 语句用于处理 score 的值大于或等于 85 的情况，输出“优秀”；第一个 elif 语句用于处理 score 的值大于或等于 75 且小于 85 的情况，输出“良好”；第二个 elif 语句用于处理 score 的值大于或等于 60 且小于 75 的情况，输出“中等”；else 语句用于处理 score 值的其他情况，输出“差”。

运行代码，结果如下所示：

```
优秀
```

3.1.4　if 嵌套

读者在某些火车站乘坐高铁出行时需要经历检票和安检两道程序：检票符合条件后方可进入安检程序，安检符合条件后方可进站乘坐高铁。这个场景中虽然涉及两个判断条件，但这两个判断条件并非选择关系，而是嵌套关系：先判断外层判断条件，该判断条件满足后才判断内层判断条件；两层判断条件都满足时才执行内层的操作。

Python 中通过 if 嵌套可以实现程序中条件语句的嵌套逻辑。if 嵌套的语法格式如下所示：

```
if 判断条件 1:                                    # 外层选择结构
    代码段 1
    if 判断条件 2:                                # 内层选择结构
            代码段 2
…
```

执行 if 嵌套时，若外层判断条件 1 的值为 True，执行代码段 1，并对内层判断条件 2 进行判断：若内层判断条件 2 的值为 True，则执行代码段 2，否则跳出内层选择结构，顺序执行外层选择结构中内层选择结构之后的代码；若外层判断条件 1 的值为 False，直接跳过外层选择结构，既不执行代码段 1，也不执行内层选择结构。if 嵌套的执行流程如图 3-4 所示。

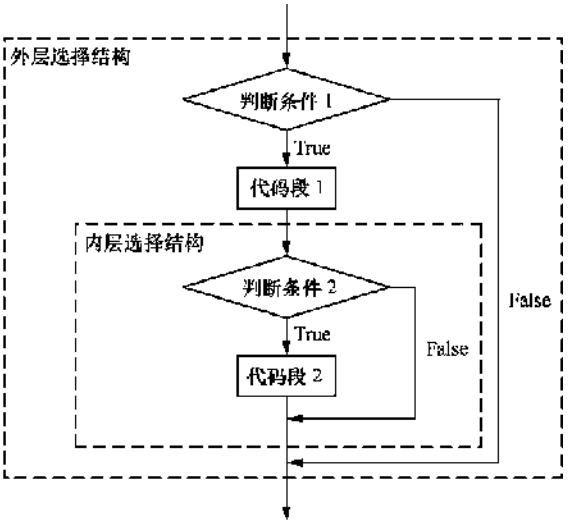

图3-4　if嵌套的执行流程

下面通过计算当月天数的案例演示 if 嵌套的用法。一年有 12 个月，每个月的总天数具有一定的规律：1、3、5、7、8、10、12 月有 31 天；4、6、9、11 月有 30 天；2 月的情

况复杂一些，闰年的 2 月有 29 天，平年的 2 月有 28 天。本案例要求根据年份和月份计算当月的天数，具体代码如下：

```
year = 2024     # 年份
month = 2       # 月份
if month in  [1, 3, 5, 7, 8, 10, 12]: # 判断月份是否为 1、3、5、7、8、10、12
    print("%d 月有 31 天 " % month)
elif month in [4, 6, 9, 11]:          # 判断月份是否为 4、6、9、11
    print("%d 月有 30 天 " % month)
elif month == 2:                      # 判断月份是否为 2 月
    # 判断年份是否为闰年
    if year % 400 == 0 or year % 4 == 0 and year % 100 != 0:
        print("%d 年%d 月有 29 天" % (year, month))
    else:
        print("%d 年%d 月有 28 天" % (year, month))
```

上述代码中，首先定义了分别表示年份和月份的变量 year 和 month，然后使用 if-elif-else 语句根据 month 的值分 3 种情况进行处理。

（1）若 month 的值存在于列表[1, 3, 5, 7, 8, 10, 12]中，说明当前月份可能为 1 月、3 月、5 月、7 月、8 月、10 月或 12 月，输出当前月份有 31 天的提示信息。

（2）若 month 的值存在于列表[4, 6, 9, 11]中，说明当前月份为 4 月、6 月、9 月或 11 月，输出当前月份有 30 天的提示信息。

（3）若 month 的值等于 2，说明当前月份为 2 月，需要使用 if-else 语句根据 year 的值分两种情况进行处理：如果 year 的值能被 400 整除或能被 4 整除且不能被 100 整除，说明当年为闰年，输出当前月份有 29 天的提示信息；如果 year 的值为其他情况，说明当年为平年，输出当前月份有 28 天的提示信息。

运行代码，结果如下所示：

```
2024 年 2 月有 29 天
```

3.2 实训案例

3.2.1 会员等级评定

员等级评定

在现代商业社会中，会员等级制度已成为吸引和回馈忠诚客户的常见方式。通过建立会员等级制度，企业不仅可以提供个性化的服务和特权，还能激励客户保持长期的合作关系。假设某平台的会员等级是根据客户的消费金额和积分评定的，具体规则如表 3-1 所示。

表 3-1 会员等级的评定规则

消费金额/元	积分/分	会员等级
$M \geqslant 1000$	$S \geqslant 10000$	钻石会员
$500 \leqslant M < 1000$	$5000 \leqslant S < 10000$	白金会员
$200 \leqslant M < 500$	$2000 \leqslant S < 5000$	黄金会员
$100 \leqslant M < 200$	$1000 \leqslant S < 2000$	白银会员
$50 \leqslant M < 100$	$500 \leqslant S < 1000$	青铜会员
$M < 50$	$0 \leqslant S < 500$	普通会员

本案例要求编写程序，根据表 3-1 提供的规则实现会员等级的评定。

3.2.2　物流费用计算


物流费用计算

我国快递行业引入新技术和创新业务模式，使我国目前已经成为全球较大、较为活跃的快递市场之一。快递行业的高速发展，使得我们邮寄物品变得方便快捷。某快递点提供华东地区、华南地区、华北地区的寄件服务，其中华东地区编号为 01、华南地区编号为 02、华北地区编号为 03。该快递点寄件价目如表 3-2 所示。

表 3-2　该快递点寄件价目

地区编号	首重寄件价目（≤2kg）	续重寄件价目（元/kg）
华东地区（01）	13 元	3
华南地区（02）	12 元	2
华北地区（03）	14 元	4

本案例要求编写程序，根据表 3-2 提供的数据实现物流费用的计算。

3.3　循环语句

在某些情况下，程序可能需要重复执行某个操作。例如，当用户登录平台时，因为其长时间未登录可能会忘记密码，所以需要反复输入账号和密码，直到达到平台设定的输入次数限制或成功登录为止。为了满足这种需求，Python 提供了循环语句，使用该语句能以简洁的代码重复执行某个操作。Python 中的循环语句包括 while 语句和 for 语句。本节将针对循环语句进行详细讲解。

3.3.1　while 语句

while 语句一般用于实现条件循环，该语句由关键字 while、循环条件和英文冒号组成，while 语句和从属于该语句的代码段组成循环结构，其语法格式如下：

```
while 循环条件:
    代码段
```

以上语法格式中的 while 关键字和英文冒号分别标识 while 语句的起始和结束，循环条件与 while 关键字以空格分隔，代码段通过缩进与 while 语句产生关联。

执行 while 语句时，若循环条件的值为 True，则执行之后的代码段，执行完代码段之后再次判断循环条件，如此往复，直至循环条件的值为 False 时循环终止，执行循环之后的代码。while 语句的执行流程如图 3-5 所示。

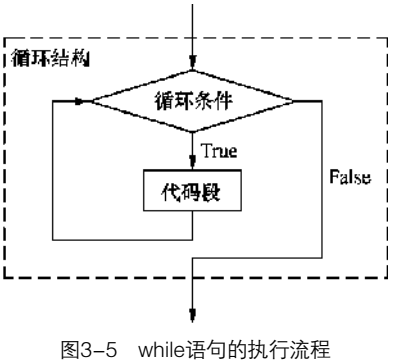

图3-5　while语句的执行流程

接下来，使用 while 语句计算 1+2+3+…+10 的结果，示例代码如下：

```
i = 1              # 保存要计算的数字，初始值为 1
result = 0         # 保存累加的结果，初始值为 0
while i <= 10:     # 使用 while 语句实现 1～10 的累加
    result += i
    i += 1
print(result)      # 输出累加后的结果
```

以上示例代码中变量 i 是循环因子，其初始值为 1，

会随循环次数累加；变量 result 用于保存累加的结果，其初始值为 0。首次执行 while 语句时，因为 i<=10 的值为 True，所以会执行循环中的代码段，使得 result 的值由 0 变为 1、i 的值由 1 变为 2；再次判断 i<=10，如此往复，直至 i 的值变为 11 时，i<=10 的值变为 False，循环结束，执行循环之后的输出语句，输出 result 的值。

运行代码，结果如下所示：

```
55
```

若希望程序可以一直重复执行某个操作，则可以将循环条件的值设为 True，如此便进入无限循环。下面是无限循环的示例代码：

```
while True:
    print("我是无限循环")
```

以上示例代码执行后会在控制台中一直输出“我是无限循环”。若希望程序能够停止输出，需要通过单击终止运行按钮或其他方式手动终止程序。

需要注意的是，虽然在实际开发中有些程序需要无限循环，比如游戏的主程序、操作系统中的监控程序等，但无限循环会占用大量内存，影响程序和系统的性能，开发人员需酌情使用。

3.3.2 for 语句

for 语句一般用于实现遍历循环。遍历指逐一访问目标对象中的元素，例如逐个访问字符串中的字符；遍历循环指在循环中完成对目标对象的遍历。for 语句的语法格式如下：

```
for 临时变量 in 目标对象:
    代码段
```

以上语法格式中的目标对象可以是字符串、文件或后续将会学习的其他组合类型数据；临时变量用于保存每次循环访问的目标对象中的元素。目标对象的元素个数决定了循环的次数，目标对象中的元素被访问完之后循环结束。

下面使用 for 语句遍历字符串"Python"的每个字符，示例代码如下：

```
for word in "Python":
    print(word)
```

运行代码，结果如下所示：

```
P
y
t
h
o
n
```

for 语句常与 range()函数搭配使用，以控制循环中代码段的执行次数。range()函数中若只有一个整数 n，则会生成一组 0～n−1 的整数；若有两个整数 m 和 n，则会生成一组 m～n−1 的整数。示例代码如下：

```
for i in range(5):
    print(i)
```

运行代码，结果如下所示：

```
0
1
2
3
4
```

3.3.3　循环嵌套

循环之间可以互相嵌套，进而实现更为复杂的逻辑。循环嵌套按不同的循环语句可以划分为 while 循环嵌套和 for 循环嵌套，关于这两种循环嵌套的介绍如下。

1.　while 循环嵌套

while 循环嵌套是指 while 语句中嵌套 while 或 for 语句。以 while 语句中嵌套 while 语句为例，while 循环嵌套的语法格式如下：

```
while 循环条件 1：                                    # 外层循环
   代码段 1
    while 循环条件 2：                                # 内层循环
        代码段 2
        …
```

执行 while 循环嵌套时，若外层循环的循环条件 1 的值为 True，则执行代码段 1，并对内层循环的循环条件 2 进行判断。若循环条件 2 的值为 True，则执行代码段 2，否则结束内层循环。内层循环执行完毕后继续判断外层循环的循环条件 1，如此往复，直至循环条件 1 的值为 False 时结束外层循环。

下面使用 while 循环嵌套输出一个由“*”构成的直角三角形，示例代码如下：

```
i = 1
while i < 6:
    j = 0
    while j < i:
        print("*", end='')
        j += 1
    print()
    i += 1
```

以上示例的外层循环用于控制直角三角形的行数，内层循环用于控制每行“*”的数量。在内层循环中，只需要根据行数输出指定数量的“*”即可，不需要负责换行的操作，因此在调用 print()函数输出“*”时，将 end 参数的值由默认的换行符'\n'修改为空字符串''，这样可以确保每一行所有的“*”是相邻的。

运行代码，结果如下所示：

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

2.　for 循环嵌套

for 循环嵌套是指 for 语句中嵌套了 while 或 for 语句。以 for 语句中嵌套 for 语句为例，for 循环嵌套的语法格式如下：

```
for 临时变量 in 目标对象:                              # 外层循环
   代码段 1
    for 临时变量 in 目标对象:                          # 内层循环
       代码段 2
       …
```

执行 for 循环嵌套时，程序会访问外层循环中目标对象的首个元素、执行代码段 1，访问内层循环目标对象的首个元素、执行代码段 2，然后访问内层循环中的下一个元素、执行代

码段 2，如此往复，直至访问完内层循环的目标对象后结束内层循环，转而继续访问外层循环中的下一个元素，访问完外层循环的目标对象后结束外层循环。因此，外层循环每执行一次，都会执行一轮内层循环。

下面使用 for 循环嵌套输出一个由“*”构成的直角三角形，示例代码如下：

```
for i in range(1, 6):
    for j in range(i):
        print("*", end=' ')
    print()
```

运行代码，结果如下所示：

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

3.4 实训案例

3.4.1 账号密码检测功能


账号密码检测功能

登录系统一般具有账号密码检测功能，即检测用户输入的账号、密码是否正确。若用户输入的账号或密码不正确，提示“用户名或密码错误”和“您还有 *N* 次机会”；若用户输入的账号和密码正确，提示“登录成功”；若输入的账号或密码的错误次数超过 3 次，提示“输入错误次数过多，请稍后再试”

本案例要求编写程序，模拟登录系统的账号密码检测功能，并限制账号或密码的错误次数至多 3 次。

3.4.2 输出五子棋棋盘


输出五子棋棋盘

五子棋是一种双人对弈的纯策略型棋类游戏，它使用的棋盘一般由横纵等距的各 15 条平行线构成，这些线垂直交叉形成的 225 个交叉点为对弈双方的落子点。本案例要求编写代码，实现按用户要求输出指定大小的五子棋棋盘的程序。10×10 的五子棋棋盘如图 3-6 所示。

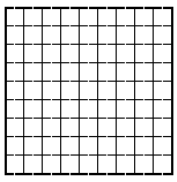

图3-6　10×10的五子棋棋盘

3.5 跳转语句

循环语句在条件满足的情况下会一直执行，但在某些情况下需要跳出循环，例如音乐播

放器循环模式的切歌功能等。Python 提供了控制循环的跳转语句：break 语句和 continue 语句。本节将针对跳转语句进行详细讲解。

3.5.1　break 语句

break 语句用于结束循环，若循环中使用了 break 语句，程序执行到 break 语句时会结束循环；若循环嵌套中使用了 break 语句，程序执行到 break 语句时会结束本层循环。break 语句通常与 if 语句配合使用，以便在条件满足时结束循环。

例如，使用 for 循环遍历字符串"Python"，一旦遍历到字符'o'就使用 break 语句结束循环，具体代码如下：

```
for word in "Python":
    if word == 'o':
        break                    # 结束循环
    print(word, end="  ")
```

运行代码，结果如下所示：

```
P  y  t  h
```

从以上输出结果可以看出，程序没有输出字符'o'及其后面的字符，说明程序遍历到字符'o'时结束了循环。

3.5.2　continue 语句

continue 语句用于在满足条件的情况下跳出本次循环，该语句通常也与 if 语句配合使用。例如，在使用 for 语句遍历字符串"Python"时，若遍历到字符'o'则使用 continue 语句跳出本次循环，具体代码如下：

```
for word in "Python":
    if word == 'o':
        continue                 # 跳出本次循环
    print(word, end="  ")
```

运行代码，结果如下所示：

```
P  y  t  h  n
```

从以上输出结果可以看出，程序没有输出字符'o'，说明满足字符为'o'的条件时跳出了本次循环。

3.6　阶段案例——房贷计算器

阶段案例——房贷计算器

房贷计算器是一种用于计算购房贷款相关数据的工具，它可以帮助用户估算利率、还款总额以及总利息等数据，便于用户更好地规划和管理房贷。本案例要求编写程序开发一个房贷计算器，具体要求如下。

（1）用户选择贷款类型。贷款类型主要有商业贷款和公积金贷款，贷款类型不同，利率也是不同的。假定商业贷款 5 年以下（含 5 年）的基准利率是 4.75%，5 年以上的基准利率是 4.90%；假定公积金贷款 5 年以下（含 5 年）的贷款利率是 2.6%，5 年以上的贷款利率是 3.1%。

（2）用户输入贷款金额（元）、贷款期限（年），房贷计算器可以按照等额本息的还款方式计算得出每月月供参考（元）、还款总额（元）、总利息（元）这几项数据。

关于每月月供参考、还款总额、总利息这几项数据的计算公式具体如下。

- 每月月供参考 = 贷款金额 × [月利率 × (1 + 月利率) ^ 还款月数] ÷ { [(1 +月利率) ^ 还款月数] - 1}。其中，月利率指以月为计息周期计算的利率，它的计算公式为：月利率=利率÷12。
- 还款总额 = 每月月供参考 × 贷款期限 × 12。
- 总利息 = 还款总额 - 贷款金额。

如果用户在选择贷款类型时输入 exit，则直接退出房贷计算器。

3.7 本章小结

本章主要讲解了流程控制的相关知识，包括条件语句、循环语句、跳转语句，并结合众多实训案例演示了如何利用各种语句实现流程控制。通过对本章的学习，读者能够掌握程序的执行流程和流程控制语句的用法，为后续的学习打好扎实的基础。

3.8 习题

一、填空题

1. Python 中通过_____实现程序中条件语句的嵌套逻辑。
2. Python 中的循环语句有_____和_____。
3. 若循环条件的值变为_____，说明程序进入无限循环。
4. _____循环一般用于实现遍历循环。
5. _____语句可以跳出本次循环，执行下一次循环。

二、判断题

1. if-else 语句可以包含多个判断条件。（　　）
2. if 语句不支持嵌套使用。（　　）
3. elif 语句可以单独使用。（　　）
4. break 语句用于结束循环。（　　）
5. for 语句只能遍历字符串。（　　）

三、选择题

1. 下列选项中，程序运行后会输出 1、2、3 的是（　　）。

A.

```
for i in range(3):
    print(i)
```

B.

```
for i in range(2):
    print(i + 1)
```

C.

```
nums = [0, 1, 2]
for i in nums:
    print(i + 1)
```

D.

```
i = 1
while i < 3:
    print(i)
    i = i + 1
```

2. 现有如下代码：

```
sum = 0
for i in range(100):
    if(i % 10):
        continue
    sum = sum + i
print(sum)
```

若运行代码，输出的结果为（　　）。

A. 5050　　B. 4950　　C. 450　　D. 45

3. 已知 x=10，y=20，z=30，以下代码执行后 x、y、z 的值分别为（　　）。

```
if x < y:
   z = x
   x = y
   y = z
```

A. 10、20、30　　B. 10、20、20　　C. 20、10、10　　D. 20、10、30

4. 已知 x 与 y 的关系如表 3-3 所示。

表 3-3　x 与 y 的关系

x	y
x<0	x–1
x=0	x
x>0	x+1

以下选项中，可以正确地表达 x 与 y 之间关系的是（　　）。

A.

```
y = x + 1
if  x >= 0:
    if x == 0:
        y = x
    else:
        y = x - 1
```

B.

```
y = x - 1
if x! = 0:
    if x > 0:
        y = x + 1
    else:
        y = x
```

C.

```
if x <= 0:
    if x < 0:
        y = x - 1
    else:
```

```
        y = x
else:
    y = x + 1
```

D.

```
y = x
if x <= 0:
    if x < 0:
        y = x - 1
    else:
        y = x + 1
```

5. 下列语句中，可以跳出循环结构的是（　　）。

A. continue　　B. break　　C. if　　D. while

四、简答题

1. 简述 break 和 continue 语句的区别。
2. 简述 while 和 for 语句的区别。

五、编程题

1. 编写程序，实现利用 while 语句输出 100 以内偶数的功能。
2. 编写程序，实现判断用户输入的数是正数还是负数的功能。
3. 编写程序，实现输出 100 以内质数的功能。