第 5 章　if 判别语句

无论在哪种编程语言中，都需要经常使用 if 语句来判定一系列条件，并依据条件决定采用何种应对措施。Python 当然也不例外。本章将介绍条件测试用以判定条件、简单的 if 语句，以及构建一系列复杂的 if 语句来判定当前条件是否满足执行对应的措施。然后，将所学的知识应用于列表处理，并以一种方式处理列表中的部分元素，以另外一种不同的方式处理额外的元素。

5.1　if-else 语句示例

下面的示例用于介绍怎样使用 if 语句处理符合条件的情况，用 else 语句处理额外的情况。假设有一个名人列表，需要将其中的元素打印出来。对于多数名人，需要以首字母大写的形式来打印其姓名，但对于'aristoteles'，需要以全大写的形式将其打印出来。

```
famousPeople = \
    ['caesar', 'homeros', 'platon', 'aristoteles', 'bacon', 'dickens', 'hugo']

for people in famousPeople:
    if people == 'aristoteles':
        print(people.upper())
    else:
        print(people.title())
```

该示例中，首先通过 for 循环在列表 famousPeople 中逐个提取元素，再使用 if 循环语句逐个比对当前所获取到的元素是否为'aristoteles'。如果是，则以全大写的形式打印出来，否则以首字母大写的形式打印，其中“＝＝”表示判定是否相等，终端中的打印结果如下所示。

```
D:\python_learn \helloWorld>python famous_people.py
Caesar
Homeros
Platon
ARISTOTELES
Bacon
Dickens
Hugo

D:\python_learn \helloWorld>_
```

5.2 条件测试

每条 if 语句的核心都涉及一个值为布尔类型(bool)的表达式,即返回值或者为 True(真),或者为 False(假),该表达式被称作条件测试。Python 会依照条件测试的值是 True 或 False 来最终决定是否执行 if 语句内缩进的代码。如果条件测试的值为 True,Python 就允许执行 if 语句内缩进的代码,否则,Python 就会忽略此代码。

5.2.1 编写约定

多数条件测试为判定变量的当前值与特定值是否一致,如下列示例所示。

```
D:\python_learn \helloWorld>python
Python 3.8.1 <tags/v3.8.1:1b293b6, Dec 18 2019, 22:39:24> [MSC v.1916 32 bit
<Intel>] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> people = ' artistoteles'  ①
>>> people = ' artistoteles'
True
>>>_
```

首先,利用单等号(=)为变量 people 赋值:'artistoteles'(详见上述①)。再利用双等号(==)判定 people 的值是否为'artistoteles',如果两边的值一致时,返回 True,否则返回 False。在该示例中,两边的值都为'artistoteles',因此 Python 会返回 True。同理,如果变量 people 的值不是'artistoteles',上述测试中 Python 将返回 False。

```
>>> people = 'hugo'
>>> people == 'artistoteles'
False
>>>_
```

5.2.2 判定是否相等时需考虑大小写

Python 判定两值是否一致时,需考虑大小写。如下列示例所示。

```
>>> people = 'HUGO'
>>> people == ' hugo'
False
>>>_
```

如果大小写无关紧要,仅需要判断等号两边变量的值,则可将变量的值通过 lower()函数转换为全小写格式,再进行比较。

```
>>> people = 'HUGO'
>>> people.lower() == 'hugo'
True
>>>_
```

值得强调的是，lower()函数不会改变储存于 people 变量内的值，故进行此种比较时并不会影响原本变量的值，详情请见以下示例。

```
>>> people = 'HUGO'
>>> people.lower() == 'hugo'
True
>>> people
'HUGO'
>>>
```

鉴于上述两个字符串一致，故此终端会返回 True。再次输入 people，打印其值，由接下来的输出可知，该条件测试并未实质影响到存储于 people 变量内的值，其值依旧为'HUGO'。

Web 登录可采取类似方式来使用户所输入的个人信息符合特定的格式要求。举例来讲，登录 Web 时就可能会涉及类似的测试以确保用户所输入的用户信息是独一无二的，而并不仅是与其他用户的用户信息的大小写格式不一致。当用户提交新的用户信息时，将其先转换为小写格式，再与服务器上存储的全部既有用户信息的小写版本进行比较。类似操作在游戏的登录系统中很常见。

5.2.3　判定不相等与不等号的写法

判定两个值是否不相等，可以使用不等号(!=)。其中，感叹号可理解为否，完整理解即为不等于，该符号在很多编程语言中都很常见。接下来，再使用 if 判别语句演示不等号运算符的使用方法。

将一款配料的名称字符串信息存储在变量 requested_topping 中，使用 if 语句判定，当顾客所要求的配料 requested_topping 不为 anchovies 时，打印一条信息："添加 anchovies!"。

```
requested_topping = 'chicken'

if requested_topping != 'anchovies':
    print("Hold the anchovies!")
```

在上述程序中，第二行代码判断 requested_topping 的值是否与'anchovies' 的值一致。如果 requested_topping 的值与'anchovies' 的值不一致，终端中将返回 True，进而执行 print 语句，并输出"Hold the anchovies!"这段话。如果两个值一致，终端就会返回 False，if 语句后面的代码不再执行。上述示例中，requested_topping 的值与'anchovies' 的值不一致，因此终端中会输出"Hold the anchovies!"这段话，终端中所示的程序运行结果如下。

```
D:\python_learn\helloWorld>python toppings.py
Hold the anchovies!

D:\python_learn \helloWorld>_
```

5.2.4 比较数字大小

在实际场景应用中，比较数字的大小也很实用。举例来讲，请判定一个人是否超过 18 岁，从而满足获取驾照的条件。

```
driving_age = 18
if driving_age == 18:
    print("You meet the conditions for obtaining a driver's license.")
```

```
D:\python_learn \helloWorld>python magic_number.py
You meet the conditions for obtaining a driver's license.

D:\python_learn \helloWorld>_
```

当然，还可以判定两个数字是否不一致，修改上述示例代码，当驾驶年龄超过最大阈值 60 岁时，指出“You age has exceeded the optimal driving age”。

```
driving_age = 82
if driving_age != 60:
    print("Your age has exceeded the optimal driving age.")
```

这里首先声明了 driving_age = 82，if 语句判定 driving_age 是否不等于(!=)最大阈值 60，如果不等于，就执行 if 语句后有 4 个空格缩进的语句：打印“You age has exceeded the optimal driving age”，其终端中显示的结果如下。

```
D:\python_learn \helloWorld>python magic_number.py
Your age has exceeded the optimal driving age.

D:\python_learn \helloWorld>_
```

此外，条件语句中还包含各类数字比较关系，诸如，>、>=、<、<=，如下列示例所示。

```
D:\python_learn \helloWorld>python   ①
Python 3.8.1 (tags/v3.8.1:1b293b6, Dec 18 2019, 22:39:24) [MSC v.1916 32 bit
(Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> age = 18
>>> age < 20
True
>>> age <= 20
True
>>> age > 20
False
>>>age >= 20
False
>>> ^Z   ②

D:\python_learn \helloWorld>_
```

5.2.5　同时判定多个条件

很多任务中都可能需要同时判定多个条件是否满足，有些任务需要两个条件都满足时，才能执行对应操作，而有时仅满足一个条件就能执行对应操作。这就不得不介绍关键字 and 和 or 了。

1. 使用 and 判定多个条件

要想判定两个条件是否都满足，就可以采用 and 关键字将两个条件合并在一起；当且仅当全部条件都满足时，整个表达式才能返回 True，而至少有一个测试不被满足时，整个表达式都将返回 False。

来看一组 and 关键字的示例，判定两个人的年龄是否不小于 21 岁。

```
D:\python_learn \helloWorld>python
Python 3.8.1 (tags/v3.8.1:1b293b6, Dec 18 2019, 22:39:24) [MSC v.1916 32 bit
(Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> age_00 = 23
>>> age_01 = 16
>>> age_00 >= 21 and age_01 >= 21
False
>>> age_01 = 23
>>> age_00 >= 21 and age_01 >= 21
True
>>>_
```

首先，age_00 = 23 与 age_01 = 16 定义了两个年龄。随后，age_00 >= 21 and age_01 >= 21 用以判定变量 age_00 与 age_01 是否都大于或等于 21，明显 age_00 大于 21，age_01 小于 21，而 and(且)需要两边条件均满足才能为真，也就是有一个条件为假，则全部表达式均为假。因此，整个条件表达式的返回值为 False。接下来，重新定义 age_01，使 age_01= 23，这句话将为 age_01 重新赋值为 23，这样 age_01 的值也大于 21，因此两个条件均符合要求，这使得整个条件表达式的返回值为 True。

此外，为提升代码的可读性，可将两个条件放置在一对圆括号之内，但该操作并非必须的。若使用其他标识符，则书写语法如下。

```
(age_00 >= 21)_and (age_01 >= 21)
```

2. 使用 or 判定多个条件

关键字 or 也能用于判定多个条件，但只要有一个条件得到满足，就会返回 True。or 关键字当且仅当多个条件都不被满足时，or 表达式才会返回 False。

接下来，继续以年龄判定举例，但与使用 and 的表达式不同，使用 or 的表达式的判定条件为有一个条件为真，示例如下。

```
>>> age_00 = 23
>>> age_01 = 16
>>> age_00 >= 21 or age_01 >= 21
```

```
True
>>> age_00 = 16
>>> age_00 >= 21 or age_01 >= 21
False
>>>_
```

5.2.6 判定特定值是否包含在列表内

在执行某些操作之前，需判定列表是否包含有特定的阈值。例如，在打开特定 Web 前，可能就需要判定用户通过输入框所提交的用户名是否已经包含在注册列表之内。而在导航应用中，也需要判定用户提供的位置信息是否包含在已有的位置列表中。这些操作就涉及关键字 in。下列代码将定义一个用户名列表，其中包含所有已注册过的用户，之后在用户 Tim 登录时，判定其是否已经注册，示例代码如下所示。

```
D:\python_learn \helloWorld>python
Python 3.8.1 (tags/v3.8.1:1b293b6, Dec 18 2019, 22:39:24) [MSC v.1916 32 bit
(Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> setUpList = ['Tim', 'Sandy', 'Peggy', 'Danel', 'Alen']
>>> 'Tim' in setUpList
True
>>> 'Davison' in setUpList
False
>>>_
```

在代码 'Tim' in setUpList 与 'Davison' in setUpList 中，in 关键字可以使 Python 解析器检查 setUpList 列表中是否包含有 'Tim' 与 'Davison' 两位用户。

5.2.7 判定特定值是否未包含在列表内

某些特定任务中，判定特定的值是否不包含于列表内非常必要，这就涉及 not in 关键字。举例来讲，有一个在论坛上被禁止发表评论的用户名列表，这就可以使用 not in 关键字，在允许用户发布评论前检查其是否已经被禁言了，如下列示例所示。

```
banned_users = ['Alene', 'Jim', 'Dave', 'Carolina']
user = 'Tim'

if user not in banned_users:
    print(user + ", you can post a response if you wish!")
```

上述程序中代码“if user not in banned_users:”表示为，如果 user 的值不被包含于列表 banned_users 内，终端中 Python 就会返回 True，从而执行缩进的 print 代码块。'Tim' 并未包含在列表 banned_users 之中，因此就会在终端中打印出“Tim, you can post a response if you wish!”这句话。

```
D:\python_learn \helloWorld>python banned_List.py
Tim, you can post a response if you wish!

D:\python_learn \helloWorld>_
```

5.2.8　布尔表达式

布尔表达式(Boolean expression)是条件测试的别称。与条件表达式一样,布尔表达式类似开关,或者为 True,或者为 False。布尔表达式常被用来记录条件,如判定游戏是否正在运行,或是用户是否可以编辑 Web 的特定内容。

```
activeGame = True
edit_Web = False
```

5.3　if 语句

if 语句有很多种,选择哪一种取决于要测试的条件数,前面在探讨条件测试时,已经列举了多种 if 语句的实例,接下来,本节将更深入地介绍该主题。

5.3.1　基础语句

最基础的 if 语句仅包含一个测试与一种对应操作,如下所示。

```
if conditional_test:
    do something
```

上述代码中,“if conditional_test: ”可包含任意条件测试;“do something”则可对应执行任意操作。若条件测试的结果为 True,Python 会执行缩进代码,否则 Python 将忽略这些代码。下面是一个有关某人年龄的代码,要求使用 if 语句判定这个人是否已经达到了参与政治选举的年龄。

```
ages = 20
if ages >= 18:
    print("Any eighteen people are eligible to vote.")
```

鉴于 ages = 20,所以“if ages >= 18”判定成立,因此会打印“Any eighteen people are eligible to vote.”,终端中执行的结果如下所示。

```
D:\python_learn \helloWorld>python voting.py
Any eighteen people are eligible to vote.

D:\python_learn \helloWorld>_
```

if 语句后缩进的代码块的作用与 for 循环中缩进的代码块基本类似。如果 if 语句后的

条件得到了满足，将继续执行 if 之后所有缩进的代码；如果 if 语句后的条件得不到满足，将忽略 if 之后所有缩进的代码。

此外，紧跟在 if 语句下面缩进的代码块，可依据需求包含任意行代码。接下来，继续完善前面的示例代码，依据需求再补充一行代码，指出："You are old enough to vote !"，请看下列示例。

```
ages = 16
if ages >= 18:
    print("Any eighteen people are eligible to vote.")
    print("You are old enough to vote!")
```

当满足 if 语句下面的条件后，紧随 if 语句所有缩进的代码都将被执行，终端中的执行结果如下所示。

```
D:\python_learn \helloWorld>python voting.py
Any eighteen people are eligible to vote.
You are old enough to vote!

D:\python_learn \helloWorld>_
```

5.3.2 if-else 语句

很多任务中，经常需要在满足条件时对应执行一些操作；在条件未被满足时执行另外一些操作。为此，可采用 Python 所提供的 if-else 语句。if-else 语句与 if 语句类似，但其中的 else 语句能够执行未满足 if 语句条件的操作。

来看一组示例，使用 if-else 语句完成。当受访者满足投票年龄时显示一段信息，否则显示另一段信息，内容如下所示。

```
ages = 20
if ages >= 18:
    print("Any eighteen people are eligible to vote.")
    print("You are old enough to vote!")
else:
    print("You are under the age to participate in political activities!")
    print("Please register to vote as soon as you turn 18.")
```

当上述测试不通过时，Python 解析器就忽略 if 语句后缩进的代码，转而执行 else 后缩进的代码块，终端中的执行结果如下所示。

```
D:\python_learn \helloWorld>python voting.py
You are under the age to participate in political activities!
Please register to vote as soon as you turn 18.

D:\python_learn hello World>_
```

if-else 引导的语句仅存在两种可能性，达到了投票年龄，或者未达到。if-else 结构适用

于执行两种操作其中之一，且总会执行其中的一种可能。

5.3.3　if-elif-else 语句

在实际任务中，也经常需要应对判定超过两种可能的情形，为此可采用 Python 提供的 if-elif-else 语句结构。该语句结构中 Python 仅会执行 if-elif-else 语句的任一代码块，因此，Python 会依次判定 if、elif、else 每个条件，直至遇见可被满足的条件。条件被满足后，Python 会继续执行其后缩进的代码，并直接忽略其余的条件，执行完毕后直接退出循环。实际任务中，许多情况需要判定的条件都可能超过两种，以停车场的收费明细为例。

(1) 1 小时以下免费；

(2) 2～8 小时，每小时收费 10 美元；

(3) 超过 8 小时，超出部分每小时收费 4 美元。

若仅使用 if 语句，如何确定收费标准呢？

```
time = 6

if time < 1:
    print("Your parking fee is $0.")
elif time > 8:
    print("Your parking fee is $4.")
else:
    print("Your parking fee is $10.")
```

其中，“if time ＜ 1：”能够判定停车时间是否小于 1 小时，若小于，终端就会打印“Your parking fee is ＄0.”，并跳过余下的测试。程序第 5 行 elif 代码其实为另一个(else)if 语句，其仅在前面的 if 测试不通过时才被执行。这里，已知 time ＝ 6，因此第一个 if 测试将无法通过。此外，若 time 大于 8 小时，Python 将打印“Your parking fee is ＄4.”，并忽略 else 代码块，跳出整个循环。若 if 测试和 elif 测试均不能通过，Python 会执行 else 代码块内的代码，并打印“Your parking fee is ＄10.”。

在程序中，“if time ＜ 1：”的测试结果为 False，因此其缩进的代码块将不被执行；“elif time ＞ 8：”的测试结果也为 False，因此将执行 else 语句中的 print 语句。终端中输出的结果如下所示。

```
D:\python_learn \helloWorld>python parking_Fee.py
Your parking fee is $10.

D:\python_learn \helloWorld>_
```

如果时间 time 为 6，那么会直接执行 else，并在终端中指出停车费为＄10。此外，还可对代码进行以下优化。

```
time = 6

if time < 1:
    price = 0
```

```
elif time > 8:
    price = 4
else:
    price = 10

print("Your parking fee is $" + str(price) + ".")
```

优化后，“if time < 1：”、“elif time > 8：”、“else：”与前一个示例一致，依据时间 time 设置变量 price 的值。在 if-elif-else 语句中嵌入 price 的值后，唯一未缩进的 print 语句会依据 price 变量的值打印"Your parking fee is $" + str(price) + ".",以指出停车费用。

```
D:\python_learn \helloWorld>python parking_Fee.py
Your parking fee is $10.

D:\python_learn \helloWorld>_
```

修改后程序的输出与前一个示例一致，但 if-elif-else 语句的作用就变小了，其仅被用来确定 price 价格变量的值，而不再同时承担打印信息这一使命。除提升执行效率外，修改后的程序更易修改：若要调整输出的内容，仅需修改最终的 print 语句即可。

5.3.4 使用多个 elif 语句

elif 语句可依据需求堆叠任意数量的代码块。例如，添加一条 elif 测试，以判定停车时间是否超过 24 小时。若超过 24 小时，超出部分予以相应折扣。

```
time = 6

if time < 1:
    price = 0
elif time > 8:
    price = 4
elif time > 24:
    price = 3
else:
    price = 10

print("Your parking fee is $" + str(price) + ".")
```

在 if-elif-else 中，只要不满足 if 与 elif 中条件的测试，都可归结于 else 语句中，并执行 else 语句中缩进的内容，但这可能会引入恶意的执行数据。因此，如果知道最终要测试的条件，建议考虑使用一系列的 elif 语句代码块来替代包罗万象的 else 代码块。这样，可确保被执行的代码更可控。

5.3.5 基于连续 if 语句的多条件测试

if-elif-else 语句的应用十分广泛，但仅能适用于一个条件得以满足的情况。当遇到满足条件的测试后，Python 就忽略其余测试条件，跳出整个循环。然而，有些任务中需要判定所