# 第5章 if 语句

编程时经常需要检查一系列条件，并据此决定采取什么措施。 在Python中，if语句能够检查程序的当前状态，并采取相应的措施。

在本章中，将学习条件测试，以检查所关心的任何条件。 将学习简单的if语句，以及创建一系列复杂的if语句来确定当前到底处于什么情形。 接下来，将把学到的知识应用于列表，编写一个for循环，以一种方式处理列表中的大多数元素，并以另一种方式处理包含特定值的元素。

## 5.1 一个简单示例

下面是一个简短的示例，演示了如何使用if语句来正确地处理特殊情形。 假设你有一个汽车列表，并想将其中每辆汽车的名称打印出来。 对于大多数汽车，应以首字母大写的方式打印其名称，但对于汽车名'bmw'，应以全大写的方式打印。 下面的代码遍历这个列表，并以首字母大写的方式打印其中的汽车名，不过对于'bmw'，则以全大写的方式打印：

In [3]:
```
cars = ['audi', 'bmw', 'subaru', 'toyota']
for car in cars:
    if car == 'bmw':
        print(car.upper())
    else:
        print(car.title())
```

```
Audi
BMW
Subaru
Toyota
```

## 5.2 条件测试

每条if语句的核心都是一个值为True或False的表达式，这种表达式称为条件测试。 Python根据条件测试的值为True还是False来决定是否执行if语句中的代码。 如果条件测试的值为True，Python就执行紧跟在if语句后面的代码； 如果为False，Python就忽略这些代码。

### 5.2.1 检查是否相等

大多数条件测试将一个变量的当前值同特定值进行比较。 最简单的条件测试检查变量的值是否与特定值相等：

In [4]:
```
car = 'bmw'
print(car == 'bmw')

car = 'audi'
print(car == 'bmw')
```

```
True
False
```

### 5.2.2 检查是否相等时忽略大小写

在Python中检查是否相等时区分大小写。 例如，两个大小写不同的值被视为不相等：

In [7]:
```
car = 'Audi'
car == 'audi'
print(ord('A'), ord('a'))
```

```
65 97
```

In [ ]:
```
car.lower() == 'audi'
```

### 5.2.3 检查是否不相等

要判断两个值是否不等，可结合使用惊叹号和等号（!=），其中的惊叹号表示不，其他很多编程语言中也是如此。 下面再使用一条if语句来演示如何使用不等运算符。 我们将把要求的比萨配料赋给一个变量，再打印一条消息，指出顾客要求的配料是否是青辣椒（peper）：

In [8]:
```
requested_topping = 'mushrooms'
if requested_topping == 'pepper':
    print("Hold the peper!")
```

```
Hold the peper!
```

### 5.2.4 数值比较

In [ ]:
```
age = 18
age == 18
```

#### 浮点数比较

In [9]:
```
if 0.3 == 0.1*3:
    print("equal")
else:
    print("not equal")
```

```
not equal
```

### 5.2.5 检查多个条件

你可能想同时检查多个条件。例如，有时候需要在两个条件都为True时才执行相应的操作，而有时候只要求一个条件为True。 在这些情况下，关键字and和or可助你一臂之力。

**使用and检查多个条件：**

In [ ]:
```
age_0 = 22
age_1 = 18
age_0 >= 21 and age_1 >= 21
```

In [ ]:
```
age_1 = 22
age_0 >= 21 and age_1 >= 21
```

为改善可读性，可将每个测试分别放在一对圆括号内，但并非必须这样做。 如果你使用圆括号，测试将类似于下面这样：

In [ ]:
```
(age_0 >= 21) and (age_1 >= 21)
```

**使用or检查多个条件**

关键字or也能够让你检查多个条件，但只要至少一个条件满足，就能通过整个测试。 仅当两个测试都没有通过时，使用or的表达式才为False。

下面再次检查两个人的年龄，但检查的条件是至少一个人的年龄不小于21岁：

In [ ]:
```
age_0 = 22
age_1 = 18
age_0 >= 21 or age_1 >= 21
```

In [ ]:
```
age_0 = 18
age_0 >= 21 or age_1 >= 21
```

In [ ]:
```
age_0 >= 21 ^ age_1 >= 21
```

挑战：

```
a = 3
b = 5
what is a ^ b
```

In [12]:
```
a = 3
b = 5
print(bin(a))
print(bin(b))
print(a^b)
print(bin(a^b))
```

```
0b11
0b101
6
0b110
```

## 5.2.6 检查特定值是否包含在列表中

有时候，执行操作前必须检查列表是否包含特定的值。例如，结束用户的注册过程前，可能需要检查他提供的用户名是否已包含在用户名列表中。 在地图程序中，可能需要检查用户提交的位置是否包含在已知位置列表中。

要判断特定的值是否已包含在列表中，可使用关键字in。 下面来看看你可能为比萨店编写的一些代码。 这些代码首先创建一个列表，其中包含用户点的比萨配料，然后检查特定的配料是否包含在该列表中。

In [13]:
```
requested_toppings = ['mushrooms', 'onions', 'pineapple']

print('mushrooms' in requested_toppings)
print('pepperoni' in requested_toppings)
```

```
True
False
```

## 5.2.7 检查特定值是否不包含在列表中

还有些时候，确定特定的值未包含在列表中很重要。 在这种情况下，可使用关键字not in。 例如，有一个列表，其中包含被禁止在论坛上发表评论的用户，可以在允许用户提交评论前检查他是否被禁言：

In [14]:
```
banned_users = ['andrew', 'carolina', 'david']
```

```
user = 'marie'
if user not in banned_users:
    print(f"{user.title()}, you can post a response if you wish.")
```

```
Marie, you can post a response if you wish.
```

### 5.2.8 布尔表达式

随着你对编程的了解越来越深入，将遇到术语布尔表达式，它不过是条件测试的别名。 与条件表达式一样，布尔表达式的结果要么为True，要么为False。

布尔值通常用于记录条件，如游戏是否正在运行，或者用户是否可以编辑网站的特定内容：

In [ ]:
```
car = 'subaru'
print("Is car == 'subaru'? I predict True.")
print(car == 'subaru')
print("\nIs car == 'audi'? I predict False.")
print(car == 'audi')
```

## 5.3 if 语句

理解条件测试后，就可以开始编写if语句了。if语句有很多种，选择使用哪种取决于要测试的条件数。 前面讨论条件测试时，列举了多个if 语句示例，下面更深入地讨论这个主题。

### 5.3.1简单的if语句

最简单的if语句只有一个测试和一个操作：

```
if conditional_test:
    do something
```

In [ ]:
```
num_covid_test = 2
if num_covid_test >= 2:
    print("完成三天两检，可以入校!")
```

### 5.3.2 if-else 语句

我们经常需要在条件测试通过时执行一个操作，在没有通过时执行另一个操作。 在这种情况下，可使用Python提供的if-else语句。 if-else语句块类似于简单的if语句，但其中的else语句让你能够指定条件测试未通过时要执行的操作。

In [16]:
```
num_covid_test = 1
if num_covid_test >= 2:
    print("完成三天两检，可以入校!")
else:
    print("未完成三天两检，不可以入校")
```

```
未完成三天两检，不可以入校
```

### 5.3.3 if-elif-else 结构

我们经常需要检查超过两个的情形，为此可使用Python提供的if-elif-else结构。 Python只执行if-elif-else结构中的一个代码块。 它依次检查每个条件测试，直到遇到通过了的条件测试。 测试通过后，Python将执行紧跟

在它后面的代码，并跳过余下的测试。 在现实世界中，很多情况下需要考虑的情形超过两个。 例如，来看一个根据年龄段购买机票：

- 2岁以下免费；
- 2～60岁收费全票；
- 60岁（含）以上收费八折。

In [ ]:
```
age = 12
if age < 2:
    print("Your flight ticket is free.")
elif age < 60:
    print("Your flight ticket has no special offer.")
else:
    print("Your flight ticket has 20% off.")
```

## 5.3.4 使用多个elif 代码块

可根据需要使用任意数量的elif 代码块。 例如，假设游乐场要给不同年龄段人打折，可再添加若干个条件测试，判断顾客是否符合打折条件。

In [ ]:
```
age = 12
if age < 4:
    price = 0
elif age < 18:
    price = 25
elif age < 65:
    price = 40
else:
    price = 20

print(f"Your admission cost is ¥{price}.")
```

## 5.3.5 省略else代码块

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语句来处理特定的情形更清晰：

In [ ]:
```
age = 12
if age < 4:
    price = 0
elif age < 18:
    price = 25
elif age < 65:
    price = 40
elif age >=65:
    price = 20

print(f"Your admission cost is ¥{price}.")
```

python 3.10 之前无switch语句

## 5.3.6 测试多个条件

if-elif-else结构功能强大，但仅适合用于只有一个条件满足的情况： 遇到通过了的测试后，Python就跳过余下的测试。 这种行为很好，效率很高，让你能够测试一个特定的条件。

然而，有时候必须检查你关心的所有条件。 在这种情况下，应使用一系列不包含elif和else代码块的简单if语句。 在可能有多个条件为True且需要在每个条件为True时都采取相应措施时，适合使用这种方法。 下面再来看看前面的比萨店示例。

如果顾客点了两种配料，就需要确保在其比萨中包含这些配料：

In [ ]:
```
requested_toppings = ['mushrooms', 'extra cheese']

if 'mushrooms' in requested_toppings:
    print("Adding mushrooms.")

if 'pepperoni' in requested_toppings:
    print("Adding pepperoni.")

if 'extra cheese' in requested_toppings:
    print("Adding extra cheese.")

print("\nFinished making your pizza!")
```

# 5.4 使用if语句处理列表

通过结合使用if 语句和列表，可完成一些有趣的任务：

对列表中特定的值做特殊处理； 高效地管理不断变化的情形，如餐馆是否还有特定的食材； 证明代码在各种情形下都将按预期那样运行。

## 5.4.1 检查特殊元素

本章开头通过一个简单示例演示了如何处理特殊值'bmw' ——它需要采用不同的格式进行打印。现在你对条件测试和if语句有了大致的认识，下面就来进一步研究如何检查列表中的特殊值，并对其做合适的处理。

继续使用前面的比萨店示例。这家比萨店在制作比萨时，每添加一种配料都打印一条消息。 通过创建一个列表，在其中包含顾客点的配料，并使用一个循环来指出添加到比萨中的配料，能以极高的效率编写这样的代码：

In [17]:
```
requested_toppings = ['mushrooms', 'green peppers', 'extra cheese']

for requested_topping in requested_toppings:
    print(f"Adding {requested_topping}.")

print("\nFinished making your pizza!")
```

```
Adding mushrooms.
Adding green peppers.
Adding extra cheese.

Finished making your pizza!
```

然而，如果比萨店的青椒用完了，该如何处理呢？为妥善地处理这种情况，可在for 循环中包含一条if 语句：

In [ ]:
```
requested_toppings = ['mushrooms', 'green peppers', 'extra cheese']
for requested_topping in requested_toppings:
    if requested_topping == 'green peppers':
        print("Sorry, we are out of green peppers right now.")
    else:
        print(f"Adding {requested_topping}.")

print("\nFinished making your pizza!")
```

### 5.4.2 确定列表不是空的

到目前为止，我们对于处理的每个列表都做了一个简单的假设——假设它们都至少包含一个元素。 因为马上就要让用户来提供存储在列表中的信息，所以不能再假设循环运行时列表不是空的。 有鉴于此，在运行for循环前确定列表是否为空很重要。 下面在制作比萨前检查顾客点的配料列表是否为空。 如果列表为空，就向顾客确认是否要点原味比萨；如果列表不为空，就像前面的示例那样制作比萨：

In [18]:
```
requested_toppings = []

if requested_toppings:
    for requested_topping in requested_toppings:
        print(f"Adding {requested_topping}.")
    print("\nFinished making your pizza!")

else:
    print("Are you sure you want a plain pizza?")
```

```
Are you sure you want a plain pizza?
```

### 5.4.3 使用多个列表

顾客的要求往往五花八门，在比萨配料方面尤其如此。 如果顾客要在比萨中添加炸薯条，该怎么办呢？可使用列表和if语句来确定能否满足顾客的要求。

来看看在制作比萨前如何拒绝怪异的配料要求。 下面的示例定义了两个列表，其中第一个列表包含比萨店供应的配料，而第二个列表包含顾客点的配料。 这次对于requested_toppings中的每个元素，都检查它是否是比萨店供应的配料，再决定是否在比萨中添加 它：

In [19]:
```
available_toppings = ['mushrooms', 'olives', 'green peppers','pepperoni', 'pineapple', '

requested_toppings = ['mushrooms', 'french fries', 'extra cheese']

for requested_topping in requested_toppings:
    if requested_topping in available_toppings:
        print(f"Adding {requested_topping}.")
    else:
        print(f"Sorry, we don't have {requested_topping}.")

print("\nFinished making your pizza!")
```

```
Adding mushrooms.
Sorry, we don't have french fries.
Adding extra cheese.

Finished making your pizza!
```

## 5.6 小结

在本章中，你学习了：

- 如何编写结果要么为True要么为False的条件测试；
- 如何编写简单的if 语句、if-else 语句和if-elif-else结构，并且在程序中使用这些结构来测试特定的条件，以确定这些条件是否满足；
- 如何在利用高效的for 循环的同时，以不同于其他元素的方式对特定的列表元素进行处理。

你还再次学习了Python就代码格式提出的建议，从而确保即便编写的程序越来越复杂，其代码依然易于阅读和理解。

在第6章，将学习Python字典。 字典类似于列表，但让你能够将不同的信息关联起来。 你将学习如何创建和遍历字典，以及如何将字典同列表和if语句结合起来使用。 学习字典让你能够模拟更多 现实世界的情形。

In [ ]:
```
# The end
```