5

CÂU LẠC BỐ IF

Lập trình thường bao gồm việc kiểm tra một tập hợp các điều kiện và quyết định hành động nào cần thực hiện dựa trên các điều kiện đó.

Câu lệnh if của Python cho phép bạn kiểm tra trạng thái hiện tại của chương trình và phản hồi phù hợp với trạng thái đó.

Trong chương này, bạn sẽ học cách viết các bài kiểm tra có điều kiện, cho phép bạn kiểm tra bất kỳ điều kiện nào bạn muốn. Bạn sẽ học cách viết các câu lệnh if đơn giản và cách tạo một chuỗi các câu lệnh if phức tạp hơn để xác định khi nào các điều kiện chính xác bạn muốn xuất hiện. Sau đó, bạn sẽ áp dụng khái niệm này vào danh sách, nhờ đó bạn có thể viết một vòng lặp for xử lý hầu hết các mục trong danh sách theo một cách nhưng xử lý một số mục nhất định có giá trị cụ thể theo một cách khác.

Một ví dụ đơn giản

Ví dụ sau đây cho thấy cách các bài kiểm tra cho phép bạn phản hồi chính xác các tình huống đặc biệt. Hãy tưởng tượng bạn có một danh sách xe hơi và muốn in ra tên của từng xe. Tên xe là tên riêng, vì vậy tên của hầu hết các xe nên được in hoa chữ cái đầu. Tuy nhiên, giá trị 'bmw' nên được in hoa hoàn toàn. Đoạn mã sau lặp qua danh sách tên xe và tìm kiếm giá trị 'bmw'. Bất cứ khi nào giá trị là 'bmw', nó sẽ được in hoa thay vì chữ cái đầu:

```
cars.py cars = ['audi', 'bmw', 'subaru', 'toyota']

cho xe trong xe ô tô:
1 nếu xe == 'bmw':
        in(xe.upper())
    khác:
        in(xe.tiêu đề())
```

Vòng lặp trong ví dụ này trước tiên sẽ kiểm tra xem giá trị hiện tại của car có phải là 'bmw' 1 hay không. Nếu đúng, giá trị sẽ được in hoa. Nếu giá trị của car không phải là 'bmw', nó sẽ được in hoa chữ thường:

```
Audi
BMW
Subaru
Toyota
```

Ví dụ này kết hợp một số khái niệm bạn sẽ tìm hiểu trong chương này. Hãy bắt đầu bằng cách xem xét các loại kiểm tra bạn có thể sử dụng để kiểm tra các điều kiện trong chương trình của mình.

Kiểm tra có điều kiện

Cốt lõi của mỗi câu lệnh if là một biểu thức có thể được đánh giá là Đúng (True) hoặc Sai (False) và được gọi là kiểm tra điều kiện (conditional test). Python sử dụng các giá trị Đúng (True) và Sai (False) để quyết định xem đoạn mã trong câu lệnh if có nên được thực thi hay không. Nếu một kiểm tra điều kiện được đánh giá là Đúng (True), Python sẽ thực thi đoạn mã tiếp theo sau câu lệnh if. câu lệnh. Nếu bài kiểm tra trả về False, Python sẽ bỏ qua đoạn mã theo sau câu lệnh if .

Kiểm tra sự bình đẳng

Hầu hết các bài kiểm tra có điều kiện đều so sánh giá trị hiện tại của một biến với một giá trị cụ thể cần quan tâm. Bài kiểm tra có điều kiện đơn giản nhất sẽ kiểm tra xem giá trị của một biến có bằng giá trị cần quan tâm hay không:

```
>>> xe = 'bmw'
>>> xe == 'bmw'
ĐÚNG VẬY
```

Dòng đầu tiên đặt giá trị của car thành 'bmw' bằng một dấu bằng duy nhất, như bạn đã thấy nhiều lần. Dòng tiếp theo kiểm tra xem giá trị của car có phải là 'bmw' hay không bằng cách sử dụng hai dấu bằng (==). Toán tử bằng này trả về True nếu các giá trị ở về trái và phải của toán tử khớp nhau, và False nếu chúng không khớp. Các giá trị trong ví dụ này khớp nhau, vì vậy Python trả về True.

Khi giá trị của car không phải là 'bmw', bài kiểm tra này trả về False:

```
>>> xe hơi = 'audi'
>>> xe == 'bmw'
SAI
```

Một dấu bằng đơn thực chất là một câu lệnh; bạn có thể đọc dòng mã đầu tiên ở đây là "Đặt giá trị của car bằng 'audi'". Mặt khác, một dấu bằng kép sẽ đặt ra câu hỏi: "Giá trị của car có bằng 'bmw' không?" Hầu hết các ngôn ngữ lập trình đều sử dụng dấu bằng theo cách này.

Bỏ qua trường hợp khi kiểm tra sự bằng nhau

Kiểm tra tính bằng nhau trong Python phân biệt chữ hoa chữ thường. Ví dụ, hai giá trị có chữ viết hoa khác nhau không được coi là bằng nhau:

```
>>> xe hơi = 'Audi'
>>> xe == 'audi'
SAI
```

Nếu chữ hoa/thường quan trọng, cách này sẽ có lợi. Tuy nhiên, nếu chữ hoa/thường không quan trọng và bạn chỉ muốn kiểm tra giá trị của một biến, bạn có thể chuyển đổi giá trị của biến sang chữ thường trước khi thực hiện so sánh:

```
>>> xe hơi = 'Audi'
>>> car.lower() == 'audi'
ĐÚNG VẬY
```

Bài kiểm tra này sẽ trả về True bất kể giá trị 'Audi' được định dạng như thế nào vì bài kiểm tra hiện không phân biệt chữ hoa chữ thường. Phương thức lower() không thay đổi giá trị ban đầu được lưu trữ trong car, vì vậy bạn có thể thực hiện phép so sánh này mà không ảnh hưởng đến biến ban đầu:

```
>>> xe hơi = 'Audi'
>>> car.lower() == 'audi'
ĐÚNG VẬY
>>> xe hơi
'Audi'
```

Đầu tiên, chúng ta gán chuỗi viết hoa 'Audi' cho biến car. Sau đó, chúng ta chuyển đổi giá trị của car sang chữ thường và so sánh giá trị chữ thường đó với chuỗi 'audi'. Hai chuỗi khớp nhau, do đó Python trả về True. Ta có thể thấy rằng giá trị được lưu trữ trong car không bị ảnh hưởng bởi phương thức lower() .

Các trang web áp dụng một số quy tắc nhất định cho dữ liệu mà người dùng nhập theo cách tương tự như thế này. Ví dụ: một trang web có thể sử dụng một bài kiểm tra có điều kiện như thế này để

Đảm bảo mỗi người dùng đều có một tên người dùng thực sự duy nhất, chứ không chỉ là một biến thể viết hoa của tên người dùng khác. Khi ai đó gửi tên người dùng mới, tên người dùng mới đó sẽ được chuyển đổi sang chữ thường và so sánh với các phiên bản chữ thường của tất cả tên người dùng hiện có. Trong quá trình kiểm tra này, tên người dùng như 'John' sẽ bị từ chối nếu bất kỳ biến thể nào của 'john' đã được sử dụng.

Kiểm tra bất đẳng thức

Khi bạn muốn xác định xem hai giá trị có bằng nhau hay không, bạn có thể sử dụng toán tử bất đẳng thức (!=). Hãy sử dụng một câu lệnh if khác để xem xét cách sử dụng toán tử bất đẳng thức. Chúng ta sẽ lưu một loại nhân bánh pizza được yêu cầu vào một biến và sau đó in ra thông báo nếu người đó không gọi món cá cơm:

toppings.py

```
requested_topping = 'nấm'

nếu requested_topping != 'cá cơm':
    print("Giữ chặt cá cơm!")
```

Mã này so sánh giá trị của requested_topping với giá trị 'anchovies'. Nếu hai giá trị này không khớp, Python trả về True và thực thi mã theo sau câu lệnh if . Nếu hai giá trị khớp, Python trả về False và không chạy mã theo sau câu lệnh if .

Bởi vì giá trị của requested_topping không phải là 'anchovies', nên print() chức năng được thực hiện:

```
Giữ chặt cá cơm!
```

Hầu hết các biểu thức điều kiện bạn viết sẽ kiểm tra sự bình đẳng, nhưng đôi khi bạn sẽ thấy việc kiểm tra bất bình đẳng hiệu quả hơn.

So sánh số

Việc kiểm tra các giá trị số khá đơn giản. Ví dụ, đoạn mã sau kiểm tra xem một người đã 18 tuổi hay chưa:

```
>>> tuổi = 18
>>> tuổi == 18
ĐÚNG VẬY
```

Bạn cũng có thể kiểm tra xem hai số có bằng nhau không. Ví dụ, đoạn mã sau sẽ in ra thông báo nếu câu trả lời đưa ra không đúng:

ảo thuật _number.py

```
câu trả lời = 17
nếu câu trả lời không = 42:
    print("Đó không phải là câu trả lời đúng. Vui lòng thử lại!")
```

Kiểm tra có điều kiện đạt, vì giá trị của câu trả lời (17) không bằng đến 42. Vì bài kiểm tra đã vượt qua, khối mã thụt lề sẽ được thực thi:

```
Đây không phải là câu trả lời đúng. Vui lòng thử lại!
```

Bạn cũng có thể đưa nhiều phép so sánh toán học vào các câu lệnh điều kiện của mình, chẳng hạn như nhỏ hơn, nhỏ hơn hoặc bằng, lớn hơn và lớn hơn hoặc bằng:

```
>>> tuổi = 19
>>> tuổi < 21
ĐÚNG VẬY
>>> tuổi <= 21
ĐÚNG VẬY
>>> tuổi > 21
SAI
>>> tuổi >= 21
SAI
```

Mỗi phép so sánh toán học có thể được sử dụng như một phần của câu lệnh if , giúp bạn phát hiện ra các điều kiện chính xác cần quan tâm.

Kiểm tra nhiều điều kiện

Bạn có thể muốn kiểm tra nhiều điều kiện cùng lúc. Ví dụ, đôi khi bạn cần hai điều kiện phải là "True" để thực hiện một hành động.
Những lúc khác, bạn có thể hài lòng với chỉ một điều kiện là Đúng. Các từ khóa và và hoặc có thể giúp bạn trong những tình huống này.

Sử dụng và để kiểm tra nhiều điều kiện

Để kiểm tra xem hai điều kiện có cùng đúng hay không , hãy sử dụng từ khóa và để kết hợp hai phép kiểm tra điều kiện; nếu mỗi phép kiểm tra đều đạt, biểu thức tổng thể sẽ được đánh giá là Đúng. Nếu một trong hai phép kiểm tra không đạt hoặc cả hai phép kiểm tra đều không đạt, biểu thức sẽ được đánh giá là Sai.

Ví dụ, bạn có thể kiểm tra xem hai người có trên 21 tuổi hay không bằng cách sử dụng bài kiểm tra sau:

```
>>> tuổi_0 = 22
>>> tuổi_1 = 18
>>> tuổi_0 >= 21 và tuổi_1 >= 21
SAI
>>> tuổi_1 = 22
>>> tuổi_0 >= 21 và tuổi_1 >= 21
ĐÚNG VẬY
```

Đầu tiên, chúng ta định nghĩa hai độ tuổi, age_0 và age_1. Sau đó, chúng ta kiểm tra xem cả hai độ tuổi đều từ 21 tuổi trở lên hay không. 1. Kiểm tra bên trái đạt, nhưng kiểm tra bên phải không đạt, do đó biểu thức điều kiện tổng thể được đánh giá là Sai. Sau đó, chúng ta đổi age_1 thành 22. Giá trị của age_1 bây giờ lớn hơn 21, do đó cả hai kiểm tra riêng lẻ đều đạt, khiến biểu thức điều kiện tổng thể được đánh giá là Đúng.

Để dễ đọc hơn, bạn có thể sử dụng dấu ngoặc đơn cho từng bài kiểm tra, nhưng điều này không bắt buộc. Nếu bạn sử dụng dấu ngoặc đơn, bài kiểm tra của bạn sẽ trông như thế này:

```
(tuổi_0 >= 21) và (tuổi_1 >= 21)
```

Sử dụng hoặc để kiểm tra nhiều điều kiện

Từ khóa or cũng cho phép bạn kiểm tra nhiều điều kiện, nhưng nó sẽ vượt qua khi một hoặc cả hai bài kiểm tra riêng lẻ đều vượt qua. Biểu thức or chỉ thất bại khi cả hai bài kiểm tra riêng lẻ đều thất bại.

Chúng ta hãy xem xét lại hai độ tuổi một lần nữa, nhưng lần này chúng ta sẽ chỉ tìm một người trên 21 tuổi:

```
>>> tuổi_0 = 22
>>> tuổi_1 = 18
1 >>> tuổi_0 >= 21 hoặc tuổi_1 >= 21
ĐÚNG VẬY
2 >>> tuổi_0 = 18
>>> tuổi_0 >= 21 hoặc tuổi_1 >= 21
SAI
```

Chúng ta lại bắt đầu với hai biến age. Vì phép thử age_0 1 đạt, biểu thức tổng thể được đánh giá là True. Sau đó, chúng ta hạ age_0 xuống 18. Trong phép thử cuối cùng 2, cả hai phép thử đều không đạt và biểu thức tổng thể được đánh giá là False.

Kiểm tra xem một giá trị có nằm trong danh sách hay không

Đôi khi, việc kiểm tra xem danh sách có chứa một giá trị nhất định hay không trước khi thực hiện hành động là rất quan trọng. Ví dụ: bạn có thể muốn kiểm tra xem tên người dùng mới đã tồn tại trong danh sách tên người dùng hiện tại hay chưa trước khi hoàn tất việc đăng ký của ai đó trên một trang web. Trong một dự án lập bản đồ, bạn có thể muốn kiểm tra xem vị trí đã gửi có tồn tại trong danh sách các vị trí đã biết hay không.

Để tìm hiểu xem một giá trị cụ thể đã có trong danh sách hay chưa, hãy sử dụng từ khóa in. Hãy xem xét một đoạn mã bạn có thể viết cho một tiệm bánh pizza. Chúng ta sẽ tạo một danh sách các loại topping mà khách hàng đã yêu cầu cho một chiếc pizza và sau đó kiểm tra xem các loại topping đó có trong danh sách hay không.

```
>>> requested_toppings = ['nấm', 'hành tây', 'dứa']
>>> 'nấm' trong requested_toppings
ĐÚNG VẬY
>>> 'pepperoni' trong requested_toppings
SAI
```

Từ khóa in yêu cầu Python kiểm tra sự tồn tại của 'nấm' và 'pepperoni' trong danh sách requested_toppings. Kỹ thuật này khá hiệu quả vì bạn có thể tạo một danh sách các giá trị thiết yếu, sau đó dễ dàng kiểm tra xem giá trị bạn đang kiểm tra có khớp với một trong các giá trị trong danh sách hay không.

Kiểm tra xem giá trị có nằm trong danh sách hay không

Những trường hợp khác, điều quan trọng là phải biết liệu một giá trị có xuất hiện trong danh sách hay không. Bạn có thể sử dụng từ khóa " not in this case" (không trong trường hợp này). Ví dụ: hãy xem xét danh sách người dùng bị cấm bình luận trên diễn đàn. Bạn có thể kiểm tra xem người dùng đó có bị cấm hay không trước khi cho phép người đó gửi bình luận:

```
banned_users.py banned_users = ['andrew', 'carolina', 'david']
                người dùng = 'marie'
```

```
nếu người dùng không có trong banned_users:
    print(f"{user.title()}, bạn có thể đăng phản hồi nếu muốn.")
```

Câu lệnh if ở đây đọc khá rõ ràng. Nếu giá trị của user không nằm trong danh sách banned_users, Python sẽ trả về True và thực thi dòng được thụt lề.

Người dùng 'marie' không có trong danh sách banned_users, vì vậy cô ấy thấy một thông báo mời cô ấy đăng phản hồi:

```
Marie, bạn có thể đăng phản hồi nếu muốn.
```

Biểu thức Boolean

Khi bạn tìm hiểu thêm về lập trình, bạn sẽ nghe thấy thuật ngữ biểu thức Boolean Tại một thời điểm nào đó. Biểu thức Boolean chỉ là một tên gọi khác của phép kiểm tra điều kiện. Giá trị Boolean có thể là True hoặc False, giống như giá trị của biểu thức điều kiện sau khi được đánh giá.

Giá trị Boolean thường được sử dụng để theo dõi một số điều kiện nhất định, chẳng hạn như trò chơi có đang chạy hay người dùng có thể chỉnh sửa nội dung nhất định trên trang web hay không:

```
game_active = Đúng
can_edit = Sai
```

Giá trị Boolean cung cấp một cách hiệu quả để theo dõi trạng thái của một chương trình hoặc một điều kiện cụ thể quan trọng trong chương trình của bạn.

HÃY TỰ THỬ

5-1. Kiểm tra có điều kiện: Viết một loạt các kiểm tra có điều kiện. In ra một câu lệnh mô tả từng kiểm tra và dự đoán của bạn về kết quả của mỗi kiểm tra. Mã của bạn sẽ trông giống như sau:

```
xe hơi = 'subaru'
print("Có phải xe == 'subaru' không? Tôi dự đoán là Đúng.")
in(xe == 'subaru')

print("\nCó phải xe == 'audi' không? Tôi dự đoán là Sai.")
in(xe == 'audi')
```

- Hãy xem xét kỹ kết quả của bạn và đảm bảo rằng bạn hiểu lý do tại sao mỗi dòng đánh giá là Đúng hoặc Sai.
- Tạo ít nhất 10 bài kiểm tra. Có ít nhất 5 bài kiểm tra được đánh giá là Đúng và một bài kiểm tra khác 5 bài kiểm tra đánh giá là Sai.

(tiếp theo)

5-2. Thêm các bài kiểm tra có điều kiện: Bạn không cần phải giới hạn số lượng bài kiểm tra tạo ra ở mức 10. Nếu bạn muốn thử nhiều phép so sánh hơn, hãy viết thêm các bài kiểm tra và thêm chúng vào conditional_tests.py. Có ít nhất một kết quả Đúng và một kết quả Sai cho mỗi trường hợp sau:

- Kiểm tra sự bằng nhau và bất bằng nhau với chuỗi
- Kiểm tra bằng phương thức lower()
- Các bài kiểm tra số liên quan đến sự bằng nhau và bất bằng nhau, lớn hơn và nhỏ hơn, lớn hơn hoặc bằng, và nhỏ hơn hoặc bằng
- Kiểm tra bằng cách sử dụng từ khóa the và the và từ khóa the hoặc
- Kiểm tra xem một mục có nằm trong danh sách không
- Kiểm tra xem một mục có nằm trong danh sách hay không

Câu lệnh if

Khi bạn hiểu về các bài kiểm tra có điều kiện, bạn có thể bắt đầu viết các câu lệnh if . Có nhiều loại câu lệnh if khác nhau , và việc bạn chọn loại nào phụ thuộc vào số lượng điều kiện cần kiểm tra. Bạn đã thấy một số ví dụ về câu lệnh if trong phần thảo luận về kiểm tra điều kiện, nhưng bây giờ chúng ta hãy cùng tìm hiểu sâu hơn về chủ đề này.

Câu lệnh if đơn giản

Loại câu lệnh if đơn giản nhất có một phép thử và một hành động:

```
nếu conditional_test:
    làm gì đó
```

Bạn có thể đặt bất kỳ bài kiểm tra điều kiện nào ở dòng đầu tiên và hầu như bất kỳ hành động nào trong khối thụt lề sau bài kiểm tra. Nếu bài kiểm tra điều kiện được đánh giá là True, Python sẽ thực thi mã theo sau câu lệnh if . Nếu bài kiểm tra được đánh giá là False, Python sẽ bỏ qua mã theo sau câu lệnh if .

Giả sử chúng ta có một biến biểu thị tuổi của một người và muốn biết liệu người đó đã đủ tuổi đi bầu hay chưa. Đoạn mã sau sẽ kiểm tra xem người đó có được phép đi bầu hay không:

```
voting.py tuổi = 19
nếu tuổi >= 18:
    print("Bạn đã đủ tuổi để bỏ phiếu!")
```

Python kiểm tra xem giá trị của age có lớn hơn hoặc bằng 18 hay không. Nếu bằng, Python sẽ thực hiện lệnh gọi print() thụt lề :

```
Bạn đã đủ tuổi để bỏ phiếu!
```

Thụt lề trong câu lệnh if có vai trò tương tự như trong vòng lặp for . Tất cả các dòng được thụt lề sau câu lệnh if sẽ được thực thi nếu bài kiểm tra thành công, và toàn bộ khối các dòng được thụt lề sẽ bị bỏ qua nếu bài kiểm tra không thành công.

Bạn có thể thêm bao nhiêu dòng mã tùy thích vào khối lệnh sau câu lệnh if . Hãy thêm một dòng đầu ra nữa nếu người đó đủ tuổi bỏ phiếu, hỏi xem họ đã đăng ký bỏ phiếu chưa:

```
tuổi = 19
nếu tuổi >= 18:
    print("Bạn đã đủ tuổi để bỏ phiếu!")
    print("Bạn đã đăng ký bỏ phiếu chưa?")
```

Kiểm tra có điều kiện đã thành công và cả hai lệnh gọi print() đều được thụt lề, do đó cả hai dòng đều được in:

```
Bạn đã đủ tuổi để bỏ phiếu!
Bạn đã đăng ký bỏ phiếu chưa?
```

Nếu giá trị tuổi nhỏ hơn 18, chương trình này sẽ không đưa ra kết quả nào.

Câu lệnh if-else

Thông thường, bạn sẽ muốn thực hiện một hành động khi một phép kiểm tra điều kiện thành công và một hành động khác trong tất cả các trường hợp còn lại. Cú pháp if-else của Python giúp thực hiện điều này. Khối if-else tương tự như một câu lệnh if đơn giản , nhưng câu lệnh else cho phép bạn định nghĩa một hành động hoặc một tập hợp các hành động được thực thi khi phép kiểm tra điều kiện thành công.

Chúng tôi sẽ hiển thị cùng một thông báo như trước đây nếu người đó đủ tuổi bỏ phiếu, nhưng lần này chúng tôi sẽ thêm thông báo cho bất kỳ ai chưa đủ tuổi bỏ phiếu:

```
tuổi = 17
1 nếu tuổi >= 18:
    print("Bạn đã đủ tuổi để bỏ phiếu!")
    print("Bạn đã đăng ký bỏ phiếu chưa?")
2 cái khác:
    print("Xin lỗi, bạn còn quá trẻ để bỏ phiếu.")
    print("Vui lòng đăng ký bỏ phiếu ngay khi bạn đủ 18 tuổi!")
```

Nếu bài kiểm tra có điều kiện 1 vượt qua, khối đầu tiên của print() thụt lề lệnh gọi được thực thi. Nếu kết quả kiểm tra là Sai, khối else 2 sẽ được thực thi.

Vì lần này tuổi nhỏ hơn 18 nên bài kiểm tra có điều kiện không thành công và mã trong khối else được thực thi:

```
Xin lỗi, bạn còn quá trẻ để bỏ phiếu.
Hãy đăng ký bỏ phiếu ngay khi bạn đủ 18 tuổi!
```

Mã này hoạt động vì nó chỉ có hai trường hợp có thể xảy ra để đánh giá: một người đủ tuổi bỏ phiếu hoặc chưa đủ tuổi bỏ phiếu. Câu lệnh if-else

Cấu trúc này hoạt động tốt trong các trường hợp bạn muốn Python luôn thực thi một trong hai hành động có thể. Trong một chuỗi if-else đơn giản như thế này, một trong hai hành động sẽ luôn được thực thi.

Chuỗi if-elif-else

Thông thường, bạn sẽ cần kiểm tra nhiều hơn hai tình huống có thể xảy ra, và để đánh giá những tình huống này, bạn có thể sử dụng cú pháp if-elif-else của Python . Python chỉ thực thi một khối trong chuỗi if-elif-else . Nó chạy từng bài kiểm tra điều kiện theo thứ tự, cho đến khi một bài kiểm tra đạt yêu cầu. Khi một bài kiểm tra đạt yêu cầu, mã theo sau bài kiểm tra đó sẽ được thực thi và Python bỏ qua phần còn lại của các bài kiểm tra.

Nhiều tình huống thực tế có thể liên quan đến nhiều hơn hai điều kiện có thể xảy ra. Ví dụ, hãy xem xét một công viên giải trí có mức giá khác nhau cho các nhóm tuổi khác nhau:

- Miễn phí vé vào cửa cho trẻ em dưới 4 tuổi.
- Giá vé vào cửa cho bất kỳ ai từ 4 đến 18 tuổi là 25 đô la.
- Giá vé vào cửa cho bất kỳ ai từ 18 tuổi trở lên là 40 đô la.

Làm thế nào chúng ta có thể sử dụng câu lệnh if để xác định tỷ lệ trúng tuyển của một người? Đoạn mã sau đây kiểm tra nhóm tuổi của một người và sau đó in ra thông báo giá vé vào cửa:

giải trí _park.py

```
tuổi = 12
1 nếu tuổi < 4:
    print("Phí vào cửa của bạn là 0 đô la.")
2 elif tuổi < 18:
    print("Phí vào cửa của bạn là 25 đô la.")
3 cái khác:
    print("Phí vào cửa của bạn là 40 đô la.")
```

Kiểm tra if 1 kiểm tra xem người dùng có dưới 4 tuổi hay không. Khi kiểm tra thành công, một thông báo phù hợp sẽ được in ra và Python bỏ qua các kiểm tra còn lại. Dòng elif 2 thực chất là một kiểm tra if khác , chỉ chạy nếu kiểm tra trước đó không thành công. Tại thời điểm này trong chuỗi, chúng ta biết người dùng ít nhất 4 tuổi vì kiểm tra đầu tiên không thành công. Nếu người dùng dưới 18 tuổi, một thông báo phù hợp sẽ được in ra và Python bỏ qua khối else . Nếu cả hai lệnh if và các bài kiểm tra elif không thành công, Python chạy mã trong khối else 3.

Trong ví dụ này, kiểm tra if 1 trả về False, do đó khối mã của nó không được thực thi. Tuy nhiên, kiểm tra elif trả về True (12 nhỏ hơn 18) nên mã của nó được thực thi. Đầu ra là một câu, thông báo cho người dùng về chi phí nhập học:

```
Phí vào cửa của bạn là 25 đô la.
```

Bất kỳ độ tuổi nào lớn hơn 17 sẽ khiến hai bài kiểm tra đầu tiên không đạt. Trong những trường hợp này, khối else sẽ được thực thi và giá vé vào cửa sẽ là 40 đô la.

Thay vì in giá vé vào cửa trong khối if-elif-else , sẽ ngắn gọn hơn nếu chỉ đặt giá bên trong chuỗi if-elif-else

và sau đó có một lệnh gọi print() duy nhất chạy sau khi chuỗi đã được đánh giá:

```
tuổi = 12

nếu tuổi < 4:
    giá = 0
elif tuổi < 18:
    giá = 25
khác:
    giá = 40

print(f"Chi phí nhập học của bạn là ${price}.")
```

Các dòng thụt lề đặt giá trị của giá theo độ tuổi của người đó, như trong ví dụ trước. Sau khi giá được đặt bởi chuỗi if-elif-else , một lệnh gọi print() riêng biệt không thụt lề sẽ sử dụng giá trị này để hiển thị thông báo báo cáo giá vé vào cửa của người đó.

Mã này tạo ra cùng kết quả như ví dụ trước, nhưng mục đích của chuỗi if-elif-else hẹp hơn. Thay vì xác định giá và hiển thị thông báo, nó chỉ đơn giản xác định giá vé vào cửa. Ngoài việc hiệu quả hơn, mã được sửa đổi này còn dễ sửa đổi hơn so với phương pháp ban đầu. Để thay đổi văn bản của thông báo đầu ra, bạn chỉ cần thay đổi một lệnh gọi print() thay vì ba lệnh gọi print() riêng biệt .

Sử dụng nhiều khối elif

Bạn có thể sử dụng bao nhiêu khối elif trong mã tùy thích. Ví dụ: nếu công viên giải trí áp dụng giảm giá cho người cao tuổi, bạn có thể thêm một bài kiểm tra điều kiện nữa vào mã để xác định xem ai đó có đủ điều kiện được giảm giá cho người cao tuổi hay không. Giả sử bất kỳ ai từ 65 tuổi trở lên trả một nửa giá vé vào cửa thông thường, tức là 20 đô la:

```
tuổi = 12

nếu tuổi < 4:
    giá = 0
elif tuổi < 18:
    giá = 25
elif tuổi < 65:
    giá = 40
khác:
    giá = 20

print(f"Chi phí nhập học của bạn là ${price}.")
```

Phần lớn mã này không thay đổi. Khối elif thứ hai hiện kiểm tra để đảm bảo người đó dưới 65 tuổi trước khi gán cho họ mức giá vé vào cửa đầy đủ là 40 đô la. Lưu ý rằng giá trị được gán trong khối else cần được đổi thành 20 đô la, vì độ tuổi duy nhất được đưa vào khối này là dành cho người từ 65 tuổi trở lên.

Bỏ qua khối else

Python không yêu cầu khối else ở cuối chuỗi if-elif .
Đôi khi, một khối else lại hữu ích. Những lúc khác, sẽ rõ ràng hơn nếu sử dụng một câu lệnh elif bổ sung để nắm bắt điều kiện cụ thể cần quan tâm:

```
tuổi = 12

nếu tuổi < 4:
    giá = 0
elif tuổi < 18:
    giá = 25
elif tuổi < 65:
    giá = 40
nếu tuổi >= 65:
    giá = 20

print(f"Chi phí nhập học của bạn là ${price}.")
```

Khối elif cuối cùng sẽ gán giá 20 đô la khi người dùng từ 65 tuổi trở lên, rõ ràng hơn một chút so với khối else thông thường . Với thay đổi này, mỗi khối mã phải vượt qua một bài kiểm tra cụ thể mới có thể được thực thi.

Khối else là một câu lệnh catchall. Nó khớp với bất kỳ điều kiện nào không được khớp bởi một phép thử if hoặc elif cụ thể , và đôi khi có thể bao gồm dữ liệu không hợp lệ hoặc thậm chí độc hại. Nếu bạn có một điều kiện cuối cùng cụ thể cần kiểm tra, hãy cân nhắc sử dụng khối elif cuối cùng và bỏ qua khối else . Nhờ đó, bạn sẽ tự tin hơn rằng mã của mình sẽ chỉ chạy trong các điều kiện chính xác.

Kiểm tra nhiều điều kiện

Chuỗi if-elif-else rất mạnh, nhưng nó chỉ phù hợp khi bạn chỉ cần một bài kiểm tra đạt yêu cầu. Ngay khi Python tìm thấy một bài kiểm tra đạt yêu cầu, nó sẽ bỏ qua các bài kiểm tra còn lại. Hành vi này rất hữu ích, vì nó hiệu quả và cho phép bạn kiểm tra một điều kiện cụ thể.

Tuy nhiên, đôi khi việc kiểm tra tất cả các điều kiện quan tâm là rất quan trọng. Trong trường hợp này, bạn nên sử dụng một loạt các câu lệnh if đơn giản mà không có elif hoặc else. khối. Kỹ thuật này có ý nghĩa khi có nhiều hơn một điều kiện có thể là Đúng và bạn muốn hành động trên mọi điều kiện là Đúng.

Hãy xem xét lại ví dụ về tiệm bánh pizza. Nếu ai đó yêu cầu một chiếc bánh pizza có hai lớp phủ pizza, bạn cần đảm bảo cho cả hai loại topping vào pizza:

```
toppings.py requested_toppings = ['nấm', 'phô mai thêm']

nếu 'nấm' trong requested_toppings:
    print("Đang thêm nấm.")
1 nếu 'pepperoni' trong requested_toppings:
    print("Thêm pepperoni.")
```

```
nếu 'phô mai thêm' trong requested_toppings:
    print("Thêm phô mai.")

print("\nĐã làm xong pizza!")
```

Chúng tôi bắt đầu với một danh sách chứa các loại topping được yêu cầu. Đầu tiên nếu Câu lệnh kiểm tra xem người dùng có yêu cầu thêm nấm trên pizza của họ hay không. Nếu có, một thông báo sẽ được in ra xác nhận việc thêm nấm. Kiểm tra cho pep-peroni 1 là một câu lệnh if đơn giản khác , không phải câu lệnh elif hay else , vì vậy kiểm tra này được chạy bất kể kiểm tra trước đó có đạt hay không. Câu lệnh if cuối cùng kiểm tra xem có yêu cầu thêm phô mai hay không, bất kể kết quả của hai kiểm tra đầu tiên. Ba kiểm tra độc lập này được thực thi mỗi khi chương trình này được chạy.

Vì mọi điều kiện trong ví dụ này đều được đánh giá nên cả nấm và phô mai đều được thêm vào bánh pizza:

```
Thêm nấm.
Thêm phô mai.

Đã làm xong pizza rồi!
```

Mã này sẽ không hoạt động bình thường nếu chúng ta sử dụng khối if-elif-else , vì mã sẽ ngừng chạy chỉ sau một lần kiểm tra thành công. Mã sẽ trông như thế này:

```
requested_toppings = ['nấm', 'thêm phô mai']

nếu 'nấm' trong requested_toppings:
    print("Đang thêm nấm.")
elif 'pepperoni' trong requested_toppings:
    print("Thêm pepperoni.")
elif 'thêm phô mai' trong requested_toppings:
    print("Thêm phô mai.")

print("\nĐã làm xong pizza!")
```

Bài kiểm tra "nấm" là bài kiểm tra đầu tiên vượt qua, vì vậy nấm được thêm vào pizza. Tuy nhiên, các giá trị "thêm phô mai" và "pepperoni" không bao giờ được kiểm tra, vì Python không chạy bất kỳ bài kiểm tra nào ngoài bài kiểm tra đầu tiên vượt qua trong chuỗi if-elif-else . Phần topping đầu tiên của khách hàng sẽ được thêm vào, nhưng tất cả các phần topping khác của họ sẽ bị bỏ qua:

```
Thêm nấm.

Đã làm xong pizza rồi!
```

Tóm lại, nếu bạn chỉ muốn chạy một khối mã, hãy sử dụng if-elif-else chuỗi. Nếu cần chạy nhiều hơn một khối mã, hãy sử dụng một loạt các câu lệnh if độc lập .

HÃY TỰ THỬ

5-3. Màu sắc của người ngoài hành tinh #1: Hãy tưởng tượng một người ngoài hành tinh vừa bị bắn hạ trong trò chơi. Tạo một biến có tên là alien_color và gán cho nó một giá trị là 'xanh lá cây', 'vàng' hoặc 'đỏ'.

- Viết câu lệnh if để kiểm tra xem màu của người ngoài hành tinh có phải là màu xanh lá cây hay không. Nếu đúng, hãy in ra thông báo rằng người chơi vừa kiếm được 5 điểm.
- Viết một phiên bản của chương trình này vượt qua bài kiểm tra if và một phiên bản khác không vượt qua. (Phiên bản không vượt qua sẽ không có kết quả đầu ra.)

5-4. Màu sắc của người ngoài hành tinh #2: Chọn một màu cho người ngoài hành tinh như bạn đã làm trong Bài tập 5-3 và viết một chuỗi if-else .

- Nếu màu của người ngoài hành tinh là màu xanh lá cây, hãy in ra thông báo rằng người chơi vừa kiếm được 5 điểm khi bắn người ngoài hành tinh.
- Nếu màu của người ngoài hành tinh không phải là màu xanh lá cây, hãy in ra một tuyên bố mà người chơi vừa kiếm được 10 điểm.
- Viết một phiên bản của chương trình này chạy khối if và một phiên bản khác chạy khối else .

5-5. Màu sắc của người ngoài hành tinh #3: Biến chuỗi if-else của bạn từ Bài tập 5-4 thành chuỗi if-elif-else .

- Nếu người ngoài hành tinh có màu xanh lá cây, hãy in thông báo cho biết người chơi đã kiếm được 5 điểm.
- Nếu người ngoài hành tinh có màu vàng, hãy in thông báo cho biết người chơi đã kiếm được 10 điểm.
- Nếu người ngoài hành tinh có màu đỏ, hãy in thông báo cho biết người chơi đã kiếm được 15 điểm.
- Viết ba phiên bản của chương trình này, đảm bảo mỗi thông điệp được in ra cho người ngoài hành tinh có màu sắc phù hợp.

5-6. Các giai đoạn của cuộc đời: Viết một chuỗi if-elif-else xác định giai đoạn cuộc đời của một người. Đặt giá trị cho biến age, sau đó:

- Nếu người đó dưới 2 tuổi, hãy in thông báo rằng người đó một em bé.
- Nếu người đó ít nhất 2 tuổi nhưng dưới 4 tuổi, hãy in thông báo rằng người đó là trẻ mới biết đi.
- Nếu người đó ít nhất 4 tuổi nhưng dưới 13 tuổi, hãy in thông báo rằng người đó là trẻ em.
- Nếu người đó ít nhất 13 tuổi nhưng dưới 20 tuổi, hãy in một thông báo rằng người đó là một thiếu niên.
- Nếu người đó ít nhất 20 tuổi nhưng dưới 65 tuổi, hãy in thông báo cho biết người đó là người trưởng thành.
- Nếu người đó từ 65 tuổi trở lên, hãy in thông báo cho biết người đó là người cao tuổi.

5-7. Loại trái cây yêu thích: Hãy lập danh sách các loại trái cây yêu thích của bạn, sau đó viết một loạt câu lệnh if độc lập để kiểm tra một số loại trái cây nhất định trong danh sách.

- Liệt kê ba loại trái cây yêu thích của bạn và gọi đó là favorite_fruits.
- Viết năm câu lệnh if . Mỗi câu lệnh sẽ kiểm tra xem một loại trái cây nhất định có trong danh sách của bạn hay không. Nếu loại trái cây đó có trong danh sách, khối if sẽ in ra một câu lệnh, chẳng hạn như Bạn thực sự thích chuối!

Sử dụng câu lệnh if với danh sách

Bạn có thể thực hiện một số công việc thú vị khi kết hợp danh sách và câu lệnh if . Bạn có thể theo dõi các giá trị đặc biệt cần được xử lý khác với các giá trị khác trong danh sách. Bạn có thể quản lý hiệu quả các điều kiện thay đổi, chẳng hạn như tình trạng sẵn có của một số mặt hàng nhất định trong nhà hàng trong suốt ca làm việc. Bạn cũng có thể bắt đầu chứng minh rằng mã của bạn hoạt động như mong đợi trong mọi tình huống có thể xảy ra.

Kiểm tra các mặt hàng đặc biệt

Chương này bắt đầu bằng một ví dụ đơn giản cho thấy cách xử lý một giá trị đặc biệt như 'bmw', cần được in ở định dạng khác với các giá trị khác trong danh sách. Giờ đây, khi bạn đã hiểu cơ bản về các phép kiểm tra điều kiện và câu lệnh if , hãy cùng xem xét kỹ hơn cách theo dõi các giá trị đặc biệt trong danh sách và xử lý chúng một cách phù hợp.

Hãy tiếp tục với ví dụ về tiệm bánh pizza. Tiệm bánh pizza sẽ hiển thị thông báo mỗi khi có topping được thêm vào pizza trong quá trình làm. Mã cho hành động này có thể được viết rất hiệu quả bằng cách tạo danh sách các topping mà khách hàng đã yêu cầu và sử dụng vòng lặp để thông báo từng topping khi nó được thêm vào pizza:

toppings.py

```
requested_toppings = ['nấm', 'ớt xanh', 'thêm phô mai']

đối với requested_topping trong requested_toppings:
    print(f"Đang thêm {requested_topping}.")

print("\nĐã làm xong pizza!")
```

Đầu ra rất đơn giản vì đoạn mã này chỉ là một vòng lặp for đơn giản :

```
Thêm nấm.
Thêm ớt xanh.
Thêm phô mai.

Đã làm xong pizza rồi!
```

Nhưng nếu tiệm pizza hết ớt xanh thì sao? Một câu lệnh if bên trong vòng lặp for có thể xử lý tình huống này một cách thích hợp:

```
requested_toppings = ['nấm', 'ớt xanh', 'thêm phô mai']

đối với requested_topping trong requested_toppings:
    nếu requested_topping == 'ớt xanh':
        print("Xin lỗi, hiện tại chúng tôi hết ớt xanh.")
    khác:
        print(f"Đang thêm {requested_topping}.")

print("\nĐã làm xong pizza!")
```

Lần này, chúng tôi kiểm tra từng món được yêu cầu trước khi thêm vào pizza. Câu lệnh if kiểm tra xem người dùng có yêu cầu ớt xanh hay không. Nếu có, chúng tôi sẽ hiển thị thông báo cho họ biết lý do tại sao họ không thể nhận được ớt xanh. Khối else đảm bảo rằng tất cả các loại topping khác sẽ được thêm vào bánh pizza.

Kết quả đầu ra cho thấy mỗi yêu cầu phủ lớp đều được xử lý phù hợp.

```
Thêm nấm.
Rất tiếc, hiện tại chúng tôi đã hết ớt xanh.
Thêm phô mai.

Đã làm xong pizza rồi!
```

Kiểm tra xem danh sách có trống không

Chúng ta đã đưa ra một giả định đơn giản về mọi danh sách đã làm việc cho đến nay: chúng ta đã giả định rằng mỗi danh sách đều có ít nhất một phần tử. Chúng ta sẽ sớm cho phép người dùng cung cấp thông tin được lưu trữ trong danh sách, vì vậy chúng ta sẽ không thể giả định rằng danh sách có bất kỳ phần tử nào mỗi khi chạy vòng lặp. Trong trường hợp này, việc kiểm tra xem danh sách có rỗng hay không trước khi chạy vòng lặp for là rất hữu ích .

Ví dụ, hãy kiểm tra xem danh sách topping được yêu cầu có trống không trước khi tạo pizza. Nếu danh sách trống, chúng ta sẽ nhắc người dùng và đảm bảo họ muốn một chiếc pizza trơn. Nếu danh sách không trống, chúng ta sẽ tạo pizza như trong các ví dụ trước:

```
requested_toppings = []

nếu yêu cầu_toppings:
    đối với requested_topping trong requested_toppings:
        print(f"Đang thêm {requested_topping}.")
    print("\nĐã làm xong pizza!")
khác:
    print("Bạn có chắc chắn muốn một chiếc pizza không?")
```

Lần này, chúng ta bắt đầu với một danh sách rỗng các phần tử được yêu cầu. Thay vì nhảy ngay vào vòng lặp for , trước tiên chúng ta sẽ kiểm tra nhanh. Khi tên của một danh sách được sử dụng trong câu lệnh if , Python trả về True nếu danh sách chứa ít nhất một phần tử; danh sách rỗng sẽ trả về False. Nếu requested_toppings vượt qua bài kiểm tra điều kiện, chúng ta sẽ chạy lại vòng lặp for đã sử dụng ở phần trước.

Ví dụ. Nếu kiểm tra có điều kiện không thành công, chúng tôi sẽ in ra thông báo hỏi khách hàng xem họ có thực sự muốn một chiếc pizza thông thường không có nhân không.

Trong trường hợp này, danh sách trống nên đầu ra sẽ hỏi liệu người dùng có thực sự muốn một chiếc pizza thông thường hay không:

```
Bạn có chắc chắn muốn ăn pizza không?
```

Nếu danh sách không trống, đầu ra sẽ hiển thị từng yêu cầu thêm được thêm vào bánh pizza.

Sử dụng nhiều danh sách

Mọi người sẽ yêu cầu gần như bất cứ thứ gì, đặc biệt là topping pizza. Nếu khách hàng thực sự muốn khoai tây chiên trên pizza thì sao? Bạn có thể sử dụng danh sách và câu lệnh if để đảm bảo thông tin đầu vào của bạn hợp lý trước khi thực hiện.

Chúng ta hãy chú ý đến những yêu cầu về topping lạ trước khi làm pizza. Ví dụ sau đây định nghĩa hai danh sách. Danh sách đầu tiên là danh sách các loại topping có sẵn tại tiệm pizza, và danh sách thứ hai là danh sách các loại topping mà người dùng đã yêu cầu. Lần này, mỗi mục trong requested_toppings sẽ được đối chiếu với danh sách các loại topping có sẵn trước khi được thêm vào pizza:

```
available_toppings = ['nấm', 'ô liu', 'ớt xanh',
                      'pepperoni', 'dứa', 'thêm phô mai']

requested_toppings = ['nấm', 'khoai tây chiên', 'thêm phô mai']

đối với requested_topping trong requested_toppings:
    nếu requested_topping trong available_toppings:
        print(f"Đang thêm {requested_topping}.")
    cái khác:
        print(f"Xin lỗi, chúng tôi không có {requested_topping}.")

print("\nĐã làm xong pizza!")
```

Trước tiên, chúng ta định nghĩa một danh sách các loại topping có sẵn tại tiệm pizza này. Lưu ý rằng đây có thể là một bộ nếu tiệm pizza có một danh sách topping ổn định. Sau đó, chúng ta tạo một danh sách các loại topping mà khách hàng đã yêu cầu. Có một yêu cầu bất thường về topping trong ví dụ này: 'khoai tây chiên' 1. Tiếp theo, chúng ta lặp qua danh sách các loại topping được yêu cầu. Bên trong vòng lặp, chúng ta kiểm tra xem mỗi loại topping được yêu cầu có thực sự nằm trong danh sách các loại topping có sẵn hay không 2. Nếu có, chúng ta thêm topping đó vào pizza. Nếu topping được yêu cầu không có trong danh sách các loại topping có sẵn, khối else sẽ chạy 3. Khối else in ra một thông báo cho người dùng biết loại topping nào không có sẵn.

Cú pháp mã này tạo ra đầu ra rõ ràng và nhiều thông tin:

```
Thêm nấm.
Rất tiếc, chúng tôi không có khoai tây chiên.
Thêm phô mai.

Đã làm xong pizza rồi!
```

Chỉ với vài dòng mã, chúng ta đã xử lý được một tình huống thực tế khá hiệu quả!

HÃY TỰ THỬ

5-8. Xin chào Quản trị viên: Tạo danh sách năm tên người dùng trở lên, bao gồm cả tên 'admin'. Hãy tưởng tượng bạn đang viết mã để in lời chào đến từng người dùng sau khi họ đăng nhập vào một trang web. Lặp lại danh sách và in lời chào đến từng người dùng.

- Nếu tên người dùng là 'admin', hãy in lời chào đặc biệt, chẳng hạn như Xin chào admin, Bạn có muốn xem báo cáo tình hình không?
- Nếu không, hãy in lời chào chung chung, chẳng hạn như Xin chào Jaden, cảm ơn bạn đã đăng nhập lại.

5-9. Không có người dùng: Thêm lệnh kiểm tra if vào hello_admin.py để đảm bảo danh sách người dùng không trống.

- Nếu danh sách trống, hãy in thông báo Chúng tôi cần tìm một số người dùng!
- Xóa tất cả tên người dùng khỏi danh sách của bạn và đảm bảo in đúng thông báo.

5-10. Kiểm tra tên người dùng: Thực hiện các bước sau để tạo chương trình mô phỏng cách các trang web đảm bảo rằng mọi người đều có tên người dùng duy nhất.

- Tạo danh sách năm hoặc nhiều tên người dùng có tên là current_users.
- Tạo một danh sách năm tên người dùng khác có tên là new_users. Đảm bảo một hoặc hai tên người dùng mới cũng nằm trong danh sách current_users .
- Lặp qua danh sách new_users để xem mỗi tên người dùng mới đã được sử dụng hay chưa. Nếu đã có, hãy in thông báo yêu cầu người dùng nhập tên người dùng mới. Nếu tên người dùng chưa được sử dụng, hãy in thông báo cho biết tên người dùng đó khả dụng.
- Đảm bảo phép so sánh của bạn không phân biệt chữ hoa chữ thường. Nếu đã sử dụng 'John' , 'JOHN' sẽ không được chấp nhận. (Để làm điều này, bạn cần tạo một bản sao của current_users chứa phiên bản chữ thường của tất cả người dùng hiện có.)

5-11. Số thứ tự: Số thứ tự cho biết vị trí của chúng trong danh sách, chẳng hạn như 1 hoặc 2. Hầu hết các số thứ tự đều kết thúc bằng th, ngoại trừ 1, 2 và 3.

- Lưu trữ các số từ 1 đến 9 trong một danh sách.
- Lặp lại danh sách.
- Sử dụng chuỗi if-elif-else bên trong vòng lặp để in ra kết thúc thứ tự thích hợp cho mỗi số. Đầu ra của bạn phải là "1st 2nd 3rd 4th 5th 6th 7th 8th 9th", và mỗi kết quả phải nằm trên một dòng riêng biệt.

Định dạng câu lệnh if của bạn

Trong mọi ví dụ trong chương này, bạn đã thấy những thói quen định dạng tốt. Khuyến nghị duy nhất mà PEP 8 đưa ra cho việc định dạng các bài kiểm tra có điều kiện là sử dụng một khoảng trắng duy nhất xung quanh các toán tử so sánh, chẳng hạn như ==, >= và <=. Ví dụ:

```
nếu tuổi < 4:
```

tốt hơn là:

```
nếu tuổi < 4:
```

Khoảng cách như vậy không ảnh hưởng đến cách Python diễn giải mã của bạn; nó chỉ giúp bạn và người khác dễ đọc mã của bạn hơn.

HÃY TỰ THỬ

5-12. Định dạng câu lệnh if : Xem lại các chương trình bạn đã viết trong chương này và đảm bảo bạn đã định dạng các bài kiểm tra có điều kiện của mình một cách phù hợp.

5-13. Ý tưởng của bạn: Đến thời điểm này, bạn đã là một lập trình viên có năng lực hơn so với khi bắt đầu cuốn sách này. Giờ đây, khi đã hiểu rõ hơn về cách các tình huống thực tế được mô phỏng trong chương trình, bạn có thể đang nghĩ đến một số vấn đề mà bạn có thể giải quyết bằng chương trình của riêng mình. Hãy ghi lại bất kỳ ý tưởng mới nào bạn có về các vấn đề mà bạn có thể muốn giải quyết khi kỹ năng lập trình của bạn tiếp tục được cải thiện. Hãy xem xét các trò chơi bạn có thể muốn viết, các tập dữ liệu bạn có thể muốn khám phá và các ứng dụng web bạn muốn tạo.

Bản tóm tắt

Trong chương này, bạn đã học cách viết các bài kiểm tra điều kiện, luôn trả về kết quả Đúng hoặc Sai. Bạn đã học cách viết các câu lệnh if , if-else đơn giản.
chuỗi lệnh và chuỗi lệnh if-elif-else . Bạn đã bắt đầu sử dụng các cấu trúc này để xác định các điều kiện cụ thể cần kiểm tra và biết khi nào các điều kiện đó được đáp ứng trong chương trình. Bạn đã học cách xử lý một số mục trong danh sách theo cách khác với tất cả các mục khác, đồng thời vẫn tận dụng hiệu quả của vòng lặp for . Bạn cũng đã xem lại các khuyến nghị về phong cách của Python để đảm bảo rằng các chương trình ngày càng phức tạp của bạn vẫn tương đối dễ đọc và dễ hiểu.

Trong Chương 6, bạn sẽ tìm hiểu về từ điển (dictionary) của Python. Từ điển tương tự như danh sách, nhưng cho phép bạn kết nối các phần thông tin. Bạn sẽ học cách xây dựng từ điển, lặp qua chúng và sử dụng chúng kết hợp với danh sách và câu lệnh if . Việc tìm hiểu về từ điển sẽ cho phép bạn mô hình hóa nhiều tình huống thực tế hơn nữa.