4

LÀM VIỆC VỚI DANH SÁCH

Trong Chương 3, bạn đã học cách tạo một danh sách đơn giản và cách làm việc với từng phần tử trong danh sách. Trong chương này, bạn sẽ học cách lặp qua toàn bộ danh sách chỉ bằng vài dòng mã, bất kể danh sách dài bao nhiêu. Lặp cho phép bạn thực hiện cùng một hành động, hoặc một tập hợp các hành động, với mọi mục trong danh sách. Nhờ đó, bạn sẽ có thể làm việc hiệu quả với các danh sách có độ dài bất kỳ, kể cả những danh sách có hàng nghìn hoặc thậm chí hàng triệu mục.

Lặp qua toàn bộ danh sách

Bạn thường muốn chạy qua tất cả các mục trong một danh sách, thực hiện cùng một tác vụ với mỗi mục. Ví dụ: trong một trò chơi, bạn có thể muốn di chuyển mọi phần tử trên màn hình với cùng một lượng. Trong một danh sách các số, bạn có thể muốn thực hiện cùng một phép toán thống kê trên mọi phần tử.

Hoặc có thể bạn muốn hiển thị từng tiêu đề từ danh sách bài viết trên một trang web. Khi bạn muốn thực hiện cùng một hành động với mọi mục trong danh sách, bạn có thể sử dụng vòng lặp for của Python .

Giả sử chúng ta có một danh sách tên các nhà ảo thuật và muốn in ra từng tên trong danh sách. Chúng ta có thể thực hiện việc này bằng cách lấy từng tên từ danh sách riêng lẻ, nhưng cách tiếp cận này có thể gây ra một số vấn đề. Thứ nhất, việc thực hiện việc này với một danh sách dài sẽ rất lặp lại. Ngoài ra, chúng ta sẽ phải thay đổi mã mỗi khi độ dài danh sách thay đổi. Sử dụng vòng lặp for sẽ tránh được cả hai vấn đề này bằng cách cho phép Python tự quản lý các vấn đề này.

Chúng ta hãy sử dụng vòng lặp for để in ra từng tên trong danh sách các nhà ảo thuật:

magicians.py magicians = ['alice', 'david', 'carolina']

```
cho ảo thuật gia trong các nhà ảo thuật:
    in (ảo thuật gia)
```

Chúng ta bắt đầu bằng cách định nghĩa một danh sách, giống như chúng ta đã làm trong Chương 3. Sau đó, chúng ta định nghĩa một vòng lặp for . Dòng này yêu cầu Python lấy một tên từ danh sách magicians và liên kết nó với biến magician. Tiếp theo, chúng ta yêu cầu Python in ra tên vừa được gán cho magician. Python sau đó lặp lại hai dòng cuối này, mỗi dòng một lần cho mỗi tên trong danh sách. Có thể hiểu đoạn mã này như sau: "Với mỗi magician trong danh sách magicians, hãy in ra tên của magician đó".

Đầu ra là bản in đơn giản của từng tên trong danh sách:

```
Alice
David
carolina
```

Nhìn kỹ hơn vào vòng lặp

Vòng lặp rất quan trọng vì nó là một trong những cách phổ biến nhất mà máy tính tự động hóa các tác vụ lặp lại. Ví dụ, trong một vòng lặp đơn giản như chúng ta đã sử dụng trong magicians.py, Python ban đầu sẽ đọc dòng đầu tiên của vòng lặp:

```
cho ảo thuật gia trong các nhà ảo thuật:
```

Dòng này yêu cầu Python lấy giá trị đầu tiên từ danh sách các nhà ảo thuật và liên kết nó với biến magician. Giá trị đầu tiên này là 'alice'. Python sau đó đọc dòng tiếp theo:

```
    in (ảo thuật gia)
```

Python in ra giá trị hiện tại của magician, vẫn là 'alice'. Bởi vì danh sách chứa nhiều giá trị hơn, Python trả về dòng đầu tiên của vòng lặp:

```
cho ảo thuật gia trong các nhà ảo thuật:
```

Python lấy tên tiếp theo trong danh sách, 'david', và liên kết tên đó giá trị với biến magician. Sau đó, Python thực thi dòng lệnh:

```
    in (ảo thuật gia)
```

Python in lại giá trị hiện tại của magician , lúc này là 'david'.
Python lặp lại toàn bộ vòng lặp một lần nữa với giá trị cuối cùng trong danh sách, 'carolina'. Vì không còn giá trị nào trong danh sách, Python chuyển sang dòng tiếp theo trong chương trình. Trong trường hợp này, không có gì theo sau vòng lặp for , nên chương trình kết thúc.

Khi bạn sử dụng vòng lặp lần đầu, hãy nhớ rằng tập hợp các bước được lặp lại một lần cho mỗi mục trong danh sách, bất kể danh sách có bao nhiêu mục. Nếu bạn có một triệu mục trong danh sách, Python sẽ lặp lại các bước này một triệu lần—và thường rất nhanh.

Khi viết vòng lặp for , hãy nhớ rằng bạn có thể chọn bất kỳ tên nào bạn muốn cho biến tạm thời sẽ được liên kết với mỗi giá trị trong danh sách. Tuy nhiên, sẽ hữu ích hơn nếu chọn một tên có ý nghĩa, đại diện cho một mục duy nhất trong danh sách. Ví dụ: đây là một cách hay để bắt đầu vòng lặp for cho danh sách mèo, danh sách chó và danh sách các mục chung:

```
cho mèo trong mèo:
cho chó trong chó:
cho mục trong list_of_items:
```

Những quy ước đặt tên này có thể giúp bạn theo dõi hành động được thực hiện trên từng mục trong vòng lặp for . Sử dụng tên số ít và số nhiều có thể giúp bạn xác định liệu một đoạn mã đang hoạt động với một phần tử duy nhất trong danh sách hay toàn bộ danh sách.

Làm nhiều việc hơn trong vòng lặp for

Bạn có thể làm bất cứ điều gì với mỗi mục trong vòng lặp for . Hãy dựa trên ví dụ trước bằng cách in ra một thông báo cho mỗi ảo thuật gia, thông báo rằng họ đã thực hiện một trò ảo thuật tuyệt vời:

magicians.py
```
magicians = ['alice', 'david', 'carolina']
cho ảo thuật gia trong các nhà ảo thuật:
    print(f"{magician.title()}, đó là một trò ảo thuật tuyệt vời!")
```

Điểm khác biệt duy nhất trong đoạn mã này là chúng ta soạn tin nhắn cho từng ảo thuật gia, bắt đầu bằng tên của ảo thuật gia đó. Lần đầu tiên qua vòng lặp, giá trị của ảo thuật gia là 'alice', vì vậy Python bắt đầu tin nhắn đầu tiên bằng tên 'Alice'. Lần thứ hai, tin nhắn sẽ bắt đầu bằng 'David', và lần thứ ba, tin nhắn sẽ bắt đầu bằng 'Carolina'.

Đầu ra hiển thị tin nhắn được cá nhân hóa cho từng ảo thuật gia trong danh sách:

```
Alice, đó quả là một trò lừa tuyệt vời!
David, đó quả là một mẹo tuyệt vời!
Carolina, đó thực sự là một trò lừa tuyệt vời!
```

Bạn cũng có thể viết bao nhiêu dòng mã tùy thích trong vòng lặp for . Mỗi dòng thụt lề theo sau dòng for magician trong magicians đều được xem xét bên trong vòng lặp, và mỗi dòng thụt lề được thực thi một lần cho mỗi giá trị trong danh sách.
Do đó, bạn có thể thực hiện nhiều công việc tùy thích với mỗi giá trị trong danh sách.

Chúng ta hãy thêm một dòng thứ hai vào thông điệp của chúng ta, nói với mỗi nhà ảo thuật rằng chúng ta mong chờ trò ảo thuật tiếp theo của họ:

```
các nhà ảo thuật = ['alice', 'david', 'carolina']
cho ảo thuật gia trong các nhà ảo thuật:
    print(f"{magician.title()}, đó là một trò ảo thuật tuyệt vời!")
    print(f"Tôi rất mong được xem màn trình diễn tiếp theo của bạn, {magician.title()}.\n")
```

Vì chúng ta đã thụt lề cả hai lệnh gọi print(), mỗi dòng sẽ được thực thi một lần cho mỗi pháp sư trong danh sách. Ký tự xuống dòng ("\n") trong lệnh gọi print() thứ hai sẽ chèn một dòng trống sau mỗi lần lặp. Điều này tạo ra một tập hợp các tin nhắn được nhóm gọn gàng cho mỗi người trong danh sách:

```
Alice, đó thực sự là một trò lừa tuyệt vời!
Tôi rất mong được xem trò ảo thuật tiếp theo của bạn, Alice.

David, đó quả là một mẹo tuyệt vời!
Tôi rất mong được xem trò ảo thuật tiếp theo của anh, David.

Carolina, đó thực sự là một trò lừa tuyệt vời!
Tôi rất mong được xem trò ảo thuật tiếp theo của bạn, Carolina.
```

Bạn có thể sử dụng bao nhiêu dòng lệnh tùy thích trong vòng lặp for . Trên thực tế, bạn thường thấy hữu ích khi thực hiện nhiều thao tác khác nhau với mỗi mục trong danh sách khi sử dụng vòng lặp for .

Làm gì đó sau vòng lặp for

Điều gì xảy ra sau khi vòng lặp for hoàn tất? Thông thường, bạn sẽ muốn tóm tắt một khối kết quả đầu ra hoặc chuyển sang công việc khác mà chương trình của bạn phải hoàn thành.

Bất kỳ dòng mã nào sau vòng lặp for mà không thụt lề sẽ được thực thi một lần mà không lặp lại. Hãy viết một lời cảm ơn đến toàn thể nhóm ảo thuật gia, cảm ơn họ đã mang đến một màn trình diễn tuyệt vời. Để hiển thị thông báo nhóm này sau khi tất cả các thông báo riêng lẻ đã được in ra, chúng ta đặt thông báo cảm ơn sau vòng lặp for mà không thụt lề:

```
các nhà ảo thuật = ['alice', 'david', 'carolina']
cho ảo thuật gia trong các nhà ảo thuật:
    print(f"{magician.title()}, đó là một trò ảo thuật tuyệt vời!")
    print(f"Tôi rất mong được xem màn trình diễn tiếp theo của bạn, {magician.title()}.\n")

print("Cảm ơn mọi người. Thật là một màn ảo thuật tuyệt vời!")
```

Hai lệnh gọi đầu tiên tới print() được lặp lại một lần cho mỗi pháp sư trong danh sách, như bạn đã thấy trước đó. Tuy nhiên, vì dòng cuối cùng không được thụt lề nên nó chỉ được in một lần:

```
Alice, đó quả là một trò lừa tuyệt vời!
Tôi rất mong được xem trò ảo thuật tiếp theo của bạn, Alice.

David, đó quả là một mẹo tuyệt vời!
```

```
Tôi rất mong được xem trò ảo thuật tiếp theo của anh, David.

Carolina, đó thực sự là một trò lừa tuyệt vời!
Tôi rất mong được xem trò ảo thuật tiếp theo của bạn, Carolina.

Cảm ơn mọi người. Thật là một màn ảo thuật tuyệt vời!
```

Khi xử lý dữ liệu bằng vòng lặp for , bạn sẽ thấy đây là một cách hay để tóm tắt một thao tác đã được thực hiện trên toàn bộ tập dữ liệu. Ví dụ: bạn có thể sử dụng vòng lặp for để khởi tạo một trò chơi bằng cách chạy qua danh sách các ký tự và hiển thị từng ký tự trên màn hình.
Sau đó, bạn có thể viết thêm một số mã sau vòng lặp này để hiển thị nút Phát ngay sau khi tất cả các ký tự đã được đưa lên màn hình.

Tránh lỗi thụt lề

Python sử dụng thụt lề để xác định mối quan hệ giữa một dòng hoặc một nhóm dòng với phần còn lại của chương trình. Trong các ví dụ trước, các dòng in thông báo cho từng ảo thuật gia là một phần của vòng lặp for vì chúng được thụt lề. Việc Python sử dụng thụt lề giúp mã rất dễ đọc.
Về cơ bản, nó sử dụng khoảng trắng để buộc bạn phải viết mã được định dạng gọn gàng với cấu trúc trực quan rõ ràng. Trong các chương trình Python dài hơn, bạn sẽ thấy các khối mã được thụt lề ở một vài mức khác nhau. Các mức thụt lề này giúp bạn có được cái nhìn tổng quan về tổ chức tổng thể của chương trình.

Khi bắt đầu viết mã dựa trên việc thụt lề đúng cách, bạn sẽ cần lưu ý một số lỗi thụt lề thường gặp. Ví dụ, đôi khi người ta thụt lề các dòng mã không cần thiết hoặc quên thụt lề các dòng cần thiết. Việc xem các ví dụ về những lỗi này ngay bây giờ sẽ giúp bạn tránh được chúng trong tương lai và sửa chúng khi chúng xuất hiện trong chương trình của bạn.

Chúng ta hãy cùng xem xét một số lỗi thụt lề phổ biến.

Quên thụt lề

Luôn thụt lề dòng sau câu lệnh for trong vòng lặp. Nếu bạn quên, Python sẽ nhắc bạn:

```
magicians.py magicians = ['alice', 'david', 'carolina']
cho ảo thuật gia trong các nhà ảo thuật:
1 bản in (ảo thuật gia)
```

Lệnh gọi print() 1 phải được thụt lề, nhưng thực tế lại không phải vậy. Khi Python mong đợi một khối được thụt lề nhưng không tìm thấy, nó sẽ cho bạn biết dòng nào gặp sự cố:

```
  Tệp "magicians.py", dòng 3
    in (ảo thuật gia)
    ^

IndentationError: mong đợi một khối thụt lề sau câu lệnh 'for' trên dòng 2
```

Bạn thường có thể giải quyết loại lỗi thụt lề này bằng cách thụt lề dòng hoặc các dòng ngay sau câu lệnh for .

Quên thụt lề các dòng bổ sung

Đôi khi vòng lặp của bạn chạy mà không có lỗi nào nhưng lại không tạo ra kết quả mong đợi. Điều này có thể xảy ra khi bạn đang cố gắng thực hiện nhiều tác vụ trong một vòng lặp và quên thụt lề một số dòng.

Ví dụ, đây là điều xảy ra khi chúng ta quên thụt lề dòng thứ hai trong vòng lặp để thông báo cho mỗi ảo thuật gia rằng chúng ta đang mong chờ màn biểu diễn tiếp theo của họ:

```
các nhà ảo thuật = ['alice', 'david', 'carolina']
cho ảo thuật gia trong các nhà ảo thuật:
    print(f"{magician.title()}, đó là một trò ảo thuật tuyệt vời!")
print(f"Tôi rất mong được xem màn trình diễn tiếp theo của bạn, {magician.title()}.\n")
```

Lệnh gọi print() 1 thứ hai được cho là thụt lề, nhưng vì Python tìm thấy ít nhất một dòng thụt lề sau câu lệnh for , nên nó không báo lỗi. Do đó, lệnh gọi print() đầu tiên được thực thi một lần cho mỗi tên trong danh sách vì nó đã thụt lề. Lệnh gọi print() thứ hai không thụt lề, do đó nó chỉ được thực thi một lần sau khi vòng lặp chạy xong.
Vì giá trị cuối cùng liên quan đến magician là 'carolina', nên cô ấy là người duy nhất nhận được thông báo "mong chờ màn ảo thuật tiếp theo":

```
Alice, đó quả là một trò lừa tuyệt vời!
David, đó quả là một mẹo tuyệt vời!
Carolina, đó thực sự là một trò lừa tuyệt vời!
Tôi rất mong được xem trò ảo thuật tiếp theo của bạn, Carolina.
```

Đây là lỗi logic. Cú pháp là mã Python hợp lệ, nhưng mã không tạo ra kết quả mong muốn vì có vấn đề về logic. Nếu bạn mong đợi một hành động nhất định được lặp lại một lần cho mỗi mục trong danh sách nhưng nó chỉ được thực thi một lần, hãy xác định xem bạn chỉ cần thụt lề một dòng hay một nhóm dòng.

Thụt lề không cần thiết

Nếu bạn vô tình thụt lề một dòng không cần thiết, Python sẽ thông báo cho bạn về việc thụt lề không mong muốn này:

```
hello_world.py message = "Xin chào thế giới Python!"
    in(tin nhắn)
```

Chúng ta không cần thụt lề lệnh gọi print() vì nó không phải là một phần của vòng lặp; do đó, Python báo lỗi:

```
  Tệp "hello_world.py", dòng 2
    in(tin nhắn)
    ^

IndentationError: thụt lề không mong muốn
```

Bạn có thể tránh các lỗi thụt lề không mong muốn bằng cách chỉ thụt lề khi có lý do cụ thể. Trong các chương trình bạn đang viết tại thời điểm này, những dòng duy nhất bạn nên thụt lề là các hành động bạn muốn lặp lại cho từng mục trong vòng lặp for .

Thụt lề không cần thiết sau vòng lặp

Nếu bạn vô tình thụt lề đoạn mã cần chạy sau khi vòng lặp kết thúc, đoạn mã đó sẽ được lặp lại một lần cho mỗi mục trong danh sách. Đôi khi, điều này sẽ nhắc Python báo lỗi, nhưng thường thì kết quả sẽ là một lỗi logic.
lỗi.

Ví dụ, hãy xem điều gì xảy ra khi chúng ta vô tình thụt lề dòng cảm ơn nhóm ảo thuật gia đã mang đến một màn trình diễn tuyệt vời:

```
magicians.py magicians = ['alice', 'david', 'carolina']
cho ảo thuật gia trong các nhà ảo thuật:
      print(f"{magician.title()}, đó là một trò ảo thuật tuyệt vời!")
      print(f"Tôi rất mong được xem màn trình diễn tiếp theo của bạn, {magician.title()}.\n")

1 bản in("Cảm ơn mọi người, đây là một màn ảo thuật tuyệt vời!")
```

Vì dòng cuối cùng 1 được thụt lề nên nó sẽ được in một lần cho mỗi người trong danh sách:

```
Alice, đó thực sự là một trò lừa tuyệt vời!
Tôi rất mong được xem trò ảo thuật tiếp theo của bạn, Alice.

Cảm ơn mọi người, đây thực sự là một màn ảo thuật tuyệt vời!
David, đó quả là một mẹo tuyệt vời!
Tôi rất mong được xem trò ảo thuật tiếp theo của anh, David.

Cảm ơn mọi người, đây thực sự là một màn ảo thuật tuyệt vời!
Carolina, đó thực sự là một trò lừa tuyệt vời!
Tôi rất mong được xem trò ảo thuật tiếp theo của bạn, Carolina.

Cảm ơn mọi người, đây thực sự là một màn ảo thuật tuyệt vời!
```

Đây là một lỗi logic khác, tương tự như lỗi trong “Quên thụt lề
"Dòng Bổ sung" ở trang 54. Vì Python không biết bạn đang cố gắng đạt được mục đích gì với mã của mình, nó sẽ chạy tất cả mã được viết theo cú pháp hợp lệ. Nếu một hành động được lặp lại nhiều lần trong khi nó chỉ nên được thực hiện một lần, có thể bạn cần bỏ thụt lề mã cho hành động đó.

Quên dấu hai chấm

Dấu hai chấm ở cuối câu lệnh for cho Python biết phải diễn giải dòng tiếp theo là bắt đầu của một vòng lặp.

```
các nhà ảo thuật = ['alice', 'david', 'carolina']
1 cho ảo thuật gia trong ảo thuật gia
      in (ảo thuật gia)
```

Nếu bạn vô tình quên dấu hai chấm 1, bạn sẽ gặp lỗi cú pháp vì Python không biết chính xác bạn đang cố làm gì:

```
  Tệp "magicians.py", dòng 2
    cho ảo thuật gia trong ảo thuật gia
                                       ^
SyntaxError: mong đợi ':'
```

Python không biết liệu bạn chỉ đơn giản là quên dấu hai chấm, hay bạn định viết thêm mã để thiết lập một vòng lặp phức tạp hơn. Nếu trình thông dịch có thể xác định được một cách sửa lỗi khả thi, nó sẽ đề xuất, chẳng hạn như thêm dấu hai chấm vào cuối dòng, như ở đây với phản hồi mong đợi là ':'. Một số lỗi có cách sửa dễ dàng và rõ ràng, nhờ các gợi ý trong chức năng theo dõi của Python. Một số lỗi khó giải quyết hơn nhiều, ngay cả khi cách sửa cuối cùng chỉ liên quan đến một ký tự duy nhất. Đừng cảm thấy tệ khi một bản sửa lỗi nhỏ mất nhiều thời gian để tìm ra; bạn hoàn toàn không phải là người duy nhất gặp phải tình trạng này.

HÃY TỰ THỬ

4-1. Pizza: Hãy nghĩ đến ít nhất ba loại pizza yêu thích của bạn. Lưu tên các loại pizza này vào một danh sách, sau đó sử dụng vòng lặp for để in tên của từng loại pizza.

- Sửa đổi vòng lặp for của bạn để in ra một câu sử dụng tên của chiếc bánh pizza, thay vì chỉ in ra tên pizza. Với mỗi chiếc pizza, bạn nên có một dòng đầu ra chứa một câu lệnh đơn giản như "Tôi thích pizza pep-peroni".

- Thêm một dòng vào cuối chương trình, bên ngoài vòng lặp for , để nêu rõ mức độ yêu thích pizza của bạn. Đầu ra nên bao gồm ba hoặc nhiều dòng về các loại pizza bạn thích và sau đó là một câu bổ sung, chẳng hạn như "Tôi thực sự thích pizza!".

4-2. Động vật: Hãy nghĩ đến ít nhất ba loài động vật khác nhau có cùng đặc điểm. Lưu tên của những loài động vật này vào một danh sách, sau đó sử dụng vòng lặp for để in ra tên của từng loài.

- Sửa đổi chương trình của bạn để in ra câu lệnh về mỗi loài động vật, chẳng hạn như Một con chó sẽ là một vật nuôi tuyệt vời.
- Thêm một dòng vào cuối chương trình, nêu rõ điểm chung của những loài động vật này. Bạn có thể in ra một câu, chẳng hạn như: Bất kỳ loài động vật nào trong số này cũng sẽ là một thú cưng tuyệt vời!

Tạo danh sách số

Có nhiều lý do để lưu trữ một tập hợp số. Ví dụ, bạn cần theo dõi vị trí của từng nhân vật trong trò chơi và bạn có thể muốn

để theo dõi điểm số cao của người chơi. Trong trực quan hóa dữ liệu, bạn hầu như luôn phải làm việc với các tập hợp số, chẳng hạn như nhiệt độ, khoảng cách, quy mô dân số, hoặc giá trị vĩ độ và kinh độ, cùng nhiều loại tập hợp số khác.

Danh sách là công cụ lý tưởng để lưu trữ các tập hợp số, và Python cung cấp nhiều công cụ giúp bạn làm việc hiệu quả với danh sách số. Một khi bạn hiểu cách sử dụng hiệu quả các công cụ này, mã của bạn sẽ hoạt động tốt ngay cả khi danh sách chứa hàng triệu phần tử.

Sử dụng hàm range()

Hàm range() của Python giúp tạo ra một chuỗi số một cách dễ dàng.
Ví dụ, bạn có thể sử dụng hàm range() để in một chuỗi số như thế này:

```
first_numbers.py cho giá trị trong phạm vi (1, 5):
    in(giá trị)
```

Mặc dù đoạn mã này có vẻ như sẽ in ra các số từ 1 đến 5, nhưng nó lại không in ra số 5:

```
1
2
3
4
```

Trong ví dụ này, hàm range() chỉ in ra các số từ 1 đến 4. Đây là một kết quả khác của hiện tượng lệch một mà bạn thường thấy trong các ngôn ngữ lập trình. Hàm range() khiến Python bắt đầu đếm ở giá trị đầu tiên bạn cung cấp và dừng lại khi đạt đến giá trị thứ hai bạn cung cấp.
Vì dừng lại ở giá trị thứ hai nên đầu ra không bao giờ chứa giá trị cuối, trong trường hợp này là 5.

Để in các số từ 1 đến 5, bạn sẽ sử dụng range(1, 6):

```
đối với giá trị trong phạm vi (1, 6):
    in(giá trị)
```

Lần này đầu ra bắt đầu từ 1 và kết thúc ở 5:

```
1
2
3
4
5
```

Nếu kết quả đầu ra khác với những gì bạn mong đợi khi sử dụng range(), hãy thử điều chỉnh giá trị cuối cùng thêm 1.

Bạn cũng có thể truyền range() chỉ một đối số và nó sẽ bắt đầu chuỗi số từ 0. Ví dụ, range(6) sẽ trả về các số từ 0 đến 5.

Sử dụng range() để tạo danh sách số

Nếu bạn muốn tạo một danh sách các số, bạn có thể chuyển đổi kết quả của range() trực tiếp vào danh sách bằng hàm list() . Khi bạn gói list() quanh lệnh gọi hàm range() , đầu ra sẽ là một danh sách các số.

Trong ví dụ ở phần trước, chúng tôi chỉ đơn giản in ra một loạt số. Chúng ta có thể sử dụng list() để chuyển đổi cùng một tập hợp số đó thành một danh sách:

```
số = danh sách(phạm vi(1, 6))
in(số)
```

Đây là kết quả:

```
[1, 2, 3, 4, 5]
```

Chúng ta cũng có thể sử dụng hàm range() để yêu cầu Python bỏ qua các số trong một phạm vi nhất định. Nếu bạn truyền đối số thứ ba cho range(), Python sẽ sử dụng giá trị đó làm kích thước bước khi tạo số.

Ví dụ, đây là cách liệt kê các số chẵn từ 1 đến 10:

even_numbers.py
```
even_numbers = danh sách(phạm vi(2, 11, 2))
in(số_chẵn)
```

Trong ví dụ này, hàm range() bắt đầu với giá trị 2 rồi cộng thêm 2 vào giá trị đó. Hàm này cộng 2 liên tục cho đến khi đạt hoặc vượt qua giá trị cuối cùng là 11, và tạo ra kết quả sau:

```
[2, 4, 6, 8, 10]
```

Bạn có thể tạo ra hầu như bất kỳ tập hợp số nào bạn muốn bằng cách sử dụng range() hàm. Ví dụ, hãy xem xét cách bạn có thể tạo danh sách 10 số chính phương đầu tiên (tức là bình phương của mỗi số nguyên từ 1 đến 10). Trong Python, hai dấu sao (**) biểu diễn số mũ. Sau đây là cách bạn có thể đưa 10 số chính phương đầu tiên vào một danh sách:

quảng trường_numbers.py
```
hình vuông = []
đối với giá trị trong phạm vi (1, 11):
1 bình phương = giá trị ** 2
2 ô vuông.append(ô vuông)

in (hình vuông)
```

Chúng ta bắt đầu với một danh sách rỗng gọi là squares. Sau đó, chúng ta yêu cầu Python lặp qua từng giá trị từ 1 đến 10 bằng hàm range() . Bên trong vòng lặp, giá trị hiện tại được nâng lên lũy thừa hai và được gán cho biến square 1. Mỗi giá trị mới của squares sau đó được thêm vào danh sách squares 2.
Cuối cùng, khi vòng lặp chạy xong, danh sách các ô vuông sẽ được in ra:

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

Để viết đoạn mã này ngắn gọn hơn, hãy bỏ biến tạm thời square và thêm trực tiếp mỗi giá trị mới vào danh sách:

```
hình vuông = []
đối với giá trị trong phạm vi (1,11):
    squares.append(giá trị**2)

in (hình vuông)
```

Dòng này thực hiện cùng chức năng như các dòng bên trong vòng lặp for ở danh sách trước. Mỗi giá trị trong vòng lặp được nâng lên lũy thừa hai và sau đó được thêm ngay vào danh sách các ô vuông.

Bạn có thể sử dụng một trong hai cách tiếp cận này khi tạo danh sách phức tạp hơn. Đôi khi, việc sử dụng biến tạm thời giúp mã của bạn dễ đọc hơn; đôi khi lại khiến mã dài dòng không cần thiết. Trước tiên, hãy tập trung vào việc viết mã mà bạn hiểu rõ và thực hiện đúng những gì bạn muốn. Sau đó, hãy tìm kiếm các cách tiếp cận hiệu quả hơn khi bạn xem lại mã.

Thống kê đơn giản với danh sách số

Một số hàm Python hữu ích khi làm việc với danh sách số. Ví dụ, bạn có thể dễ dàng tìm giá trị nhỏ nhất, lớn nhất và tổng của một danh sách số:

```
>>> chữ số = [1, 2, 3, 4, 5, 6, 7, 8, 9, 0]
>>> min(chữ số)
0
>>> max(chữ số)
9
>>> tổng(chữ số)
45
```

LƯU Ý: Các ví dụ trong phần này sử dụng danh sách ngắn các số vừa vặn trên trang. Chúng cũng sẽ hiệu quả nếu danh sách của bạn chứa một triệu số hoặc hơn.

Liệt kê các hiểu biết

Cách tiếp cận được mô tả trước đó để tạo các ô danh sách bao gồm ba hoặc bốn dòng mã. List comprehension cho phép bạn tạo cùng một danh sách này chỉ trong một dòng mã. List comprehension kết hợp vòng lặp for và việc tạo các phần tử mới thành một dòng, và tự động thêm mỗi phần tử mới vào. List comprehension không phải lúc nào cũng được trình bày cho người mới bắt đầu, nhưng tôi đã đưa chúng vào đây vì rất có thể bạn sẽ thấy chúng ngay khi bắt đầu xem mã của người khác.

Ví dụ sau đây xây dựng cùng một danh sách các số bình phương mà bạn đã thấy trước đó nhưng sử dụng danh sách hiểu biết:

```
squares.py squares = [giá trị**2 cho giá trị trong phạm vi(1, 11)]
in (hình vuông)
```

Để sử dụng cú pháp này, hãy bắt đầu bằng một tên mô tả cho danh sách, chẳng hạn như "squares". Tiếp theo, mở một cặp dấu ngoặc vuông và định nghĩa biểu thức cho các giá trị bạn muốn lưu trữ trong danh sách mới. Trong ví dụ này, biểu thức là value**2, tức là nâng giá trị lên lũy thừa hai. Sau đó, viết một vòng lặp for để tạo các số bạn muốn đưa vào biểu thức, và đóng dấu ngoặc vuông. Vòng lặp for trong ví dụ này là for value in range(1, 11), đưa các giá trị từ 1 đến 10 vào biểu thức value**2. Lưu ý rằng không có dấu hai chấm nào được sử dụng ở cuối câu lệnh for .

Kết quả là danh sách các số chính phương giống như bạn đã thấy trước đó:

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

Bạn cần phải luyện tập để viết danh sách hiểu của riêng mình, nhưng bạn sẽ thấy chúng rất hữu ích khi bạn đã quen với việc tạo danh sách thông thường.
Khi bạn viết ba hoặc bốn dòng mã để tạo danh sách và bắt đầu cảm thấy lặp lại, hãy cân nhắc việc viết danh sách hiểu biết của riêng bạn.

HÃY TỰ THỬ

4-3. Đếm đến hai mươi: Sử dụng vòng lặp for để in các số từ 1 đến 20, bao gồm cả 1 và 20.

4-4. Một triệu: Tạo danh sách các số từ một đến một triệu, sau đó sử dụng vòng lặp for để in các số đó. (Nếu đầu ra mất quá nhiều thời gian, hãy dừng bằng cách nhấn CTRL-C hoặc đóng cửa sổ đầu ra.)

4-5. Tính tổng một triệu: Tạo một danh sách các số từ một đến một triệu, sau đó sử dụng min() và max() để đảm bảo danh sách của bạn thực sự bắt đầu từ một và kết thúc ở một triệu. Ngoài ra, hãy sử dụng hàm sum() để xem Python có thể cộng một triệu số nhanh như thế nào.

4-6. Số lẻ: Sử dụng đối số thứ ba của hàm range() để tạo danh sách các số lẻ từ 1 đến 20. Sử dụng vòng lặp for để in từng số.

4-7. Số ba: Tạo danh sách các bội số của 3, từ 3 đến 30. Sử dụng vòng lặp for để in các số trong danh sách của bạn.

4-8. Lập phương: Một số mũ ba được gọi là lập phương. Ví dụ, lập phương của 2 được viết là 2**3 trong Python. Hãy lập danh sách 10 lập phương đầu tiên (tức là lập phương của mỗi số nguyên từ 1 đến 10) và sử dụng vòng lặp for để in ra giá trị của mỗi lập phương.

4-9. Hiểu về khối lập phương: Sử dụng hiểu biết về danh sách để tạo danh sách gồm 10 khối lập phương đầu tiên.

Làm việc với một phần của danh sách

Trong Chương 3, bạn đã học cách truy cập từng phần tử trong danh sách, và trong chương này, bạn sẽ học cách xử lý tất cả các phần tử trong danh sách. Bạn cũng có thể làm việc với một nhóm mục cụ thể trong danh sách, được gọi là một lát cắt (slice) trong Python.

Cắt một danh sách

Để tạo một lát cắt, bạn chỉ định chỉ số của phần tử đầu tiên và phần tử cuối cùng mà bạn muốn xử lý. Tương tự như hàm range() , Python sẽ dừng một phần tử trước chỉ số thứ hai mà bạn chỉ định. Để xuất ra ba phần tử đầu tiên trong một danh sách, bạn sẽ yêu cầu các chỉ số từ 0 đến 3, kết quả trả về sẽ là các phần tử 0, 1 và 2.

Ví dụ sau đây liên quan đến danh sách các cầu thủ trong một đội:

players.py
```
players = ['charles', 'martina', 'michael', 'florence', 'eli']
in(người chơi[0:3])
```

Đoạn mã này in ra một phần của danh sách. Kết quả đầu ra vẫn giữ nguyên cấu trúc của danh sách và bao gồm ba người chơi đầu tiên trong danh sách:

```
['charles', 'martina', 'michael']
```

Bạn có thể tạo bất kỳ tập con nào của danh sách. Ví dụ: nếu bạn muốn tìm phần tử thứ hai, thứ ba và thứ tư trong danh sách, bạn sẽ bắt đầu lát cắt ở chỉ mục 1 và kết thúc ở chỉ mục 4:

```
người chơi = ['charles', 'martina', 'michael', 'florence', 'eli']
in(người chơi[1:4])
```

Lần này lát cắt bắt đầu bằng 'martina' và kết thúc bằng 'florence':

```
['martina', 'michael', 'florence']
```

Nếu bạn bỏ qua chỉ mục đầu tiên trong một lát cắt, Python sẽ tự động bắt đầu lát cắt ở đầu danh sách:

```
người chơi = ['charles', 'martina', 'michael', 'florence', 'eli']
in(người chơi[:4])
```

Nếu không có chỉ mục bắt đầu, Python sẽ bắt đầu từ đầu danh sách:

```
['charles', 'martina', 'michael', 'florence']
```

Cú pháp tương tự sẽ hiệu quả nếu bạn muốn một lát cắt bao gồm phần cuối của danh sách. Ví dụ, nếu bạn muốn tất cả các mục từ mục thứ ba đến mục cuối cùng, bạn có thể bắt đầu với chỉ mục 2 và bỏ qua chỉ mục thứ hai:

```
người chơi = ['charles', 'martina', 'michael', 'florence', 'eli']
in(người chơi[2:])
```

Python trả về tất cả các mục từ mục thứ ba cho đến cuối danh sách:

```
['michael', 'florence', 'eli']
```

Cú pháp này cho phép bạn xuất tất cả các phần tử từ bất kỳ điểm nào trong danh sách đến cuối, bất kể độ dài của danh sách. Hãy nhớ rằng chỉ mục âm trả về một phần tử cách cuối danh sách một khoảng cách nhất định; do đó, bạn có thể xuất bất kỳ lát cắt nào từ cuối danh sách. Ví dụ: nếu chúng ta muốn xuất ba cầu thủ cuối cùng trong danh sách, chúng ta có thể sử dụng lát cắt players[-3:]:

```
người chơi = ['charles', 'martina', 'michael', 'florence', 'eli']
in(người chơi[-3:])
```

Điều này in tên của ba cầu thủ cuối cùng và sẽ tiếp tục hoạt động khi danh sách người chơi thay đổi về kích thước.

LƯU Ý: n có thể thêm giá trị thứ ba vào trong ngoặc đơn để biểu thị một lát cắt. Nếu thêm giá trị thứ ba, giá trị này sẽ cho Python biết số lượng mục cần bỏ qua giữa các mục trong phạm vi được chỉ định.

Lặp qua một lát cắt

Bạn có thể sử dụng một lát cắt trong vòng lặp for nếu muốn lặp qua một tập hợp con các phần tử trong danh sách. Trong ví dụ tiếp theo, chúng ta lặp qua ba cầu thủ đầu tiên và in tên của họ như một phần của danh sách đơn giản:

```
người chơi = ['charles', 'martina', 'michael', 'florence', 'eli']

print("Đây là ba cầu thủ đầu tiên trong đội của tôi:")
1 cho người chơi trong người chơi[:3]:
    in(player.title())
```

Thay vì lặp qua toàn bộ danh sách người chơi, Python chỉ lặp qua ba tên đầu tiên 1:

```
Sau đây là ba cầu thủ đầu tiên trong đội của tôi:
Charles
Martina
Michael
```

Slice rất hữu ích trong một số trường hợp. Ví dụ, khi tạo trò chơi, bạn có thể thêm điểm số cuối cùng của người chơi vào danh sách mỗi khi người chơi đó kết thúc trò chơi. Sau đó, bạn có thể lấy ba điểm số cao nhất của người chơi bằng cách sắp xếp danh sách theo thứ tự giảm dần và lấy một slice chỉ bao gồm ba điểm số đầu tiên. Khi làm việc với dữ liệu, bạn có thể sử dụng slice để xử lý dữ liệu thành các khối có kích thước cụ thể. Hoặc, khi xây dựng ứng dụng web, bạn có thể sử dụng slice để hiển thị thông tin thành một loạt trang với lượng thông tin phù hợp trên mỗi trang.

Sao chép danh sách

Thông thường, bạn sẽ muốn bắt đầu với một danh sách hiện có và tạo một danh sách hoàn toàn mới dựa trên danh sách đầu tiên. Hãy cùng khám phá cách sao chép danh sách hoạt động và xem xét một tình huống mà việc sao chép danh sách hữu ích.

Để sao chép một danh sách, bạn có thể tạo một lát cắt bao gồm toàn bộ danh sách gốc bằng cách bỏ chỉ mục đầu tiên và chỉ mục thứ hai ([:]). Điều này yêu cầu Python tạo một lát cắt bắt đầu từ phần tử đầu tiên và kết thúc ở phần tử cuối cùng, tạo ra một bản sao của toàn bộ danh sách.

Ví dụ, hãy tưởng tượng chúng ta có một danh sách các món ăn yêu thích và muốn tạo một danh sách riêng về những món ăn mà một người bạn thích. Người bạn này thích tất cả mọi thứ trong danh sách của chúng ta cho đến nay, vì vậy chúng ta có thể tạo danh sách của họ bằng cách sao chép danh sách của mình:

foods.py
```
my_foods = ['pizza', 'falafel', 'bánh cà rốt']
1 friend_foods = my_foods[:]

print("Món ăn yêu thích của tôi là:")
in(my_foods)

print("\nMón ăn yêu thích của bạn tôi là:")
in(friend_foods)
```

Đầu tiên, chúng ta tạo một danh sách các loại thực phẩm chúng ta thích, gọi là my_foods. Sau đó, chúng ta tạo một danh sách mới, gọi là friend_foods. Chúng ta tạo một bản sao của my_foods bằng cách yêu cầu một phần tử my_foods mà không chỉ định bất kỳ chỉ số nào là 1, và gán bản sao đó cho friend_foods. Khi in ra mỗi danh sách, chúng ta thấy rằng cả hai đều chứa cùng một loại thực phẩm:

```
Những món ăn tôi thích nhất là:
['pizza', 'falafel', 'bánh cà rốt']

Những món ăn yêu thích của bạn tôi là:
['pizza', 'falafel', 'bánh cà rốt']
```

Để chứng minh rằng chúng ta thực sự có hai danh sách riêng biệt, chúng ta sẽ thêm một loại thực phẩm mới vào mỗi danh sách và chỉ ra rằng mỗi danh sách đều theo dõi các loại thực phẩm yêu thích của từng người:

```
my_foods = ['pizza', 'falafel', 'bánh cà rốt']
1 friend_foods = my_foods[:]

2 my_foods.append('cannoli')
3 friend_foods.append('kem')

print("Món ăn yêu thích của tôi là:")
in(my_foods)

print("\nMón ăn yêu thích của bạn tôi là:")
in(friend_foods)
```

Chúng ta sao chép các mục gốc trong my_foods vào danh sách mới friend_foods, như đã làm trong ví dụ trước 1. Tiếp theo, chúng ta thêm một loại thực phẩm mới vào mỗi danh sách: chúng ta thêm 'cannoli' vào my_foods 2 và chúng ta thêm 'ice cream' vào friend_foods 3. Sau đó, chúng ta in hai danh sách để xem liệu mỗi loại thực phẩm này có nằm trong danh sách thích hợp hay không:

```
Những món ăn tôi thích nhất là:
['pizza', 'falafel', 'bánh cà rốt', 'cannoli']

Những món ăn yêu thích của bạn tôi là:
['pizza', 'falafel', 'bánh cà rốt', 'kem']
```

Kết quả cho thấy 'cannoli' hiện xuất hiện trong danh sách các món ăn yêu thích của chúng tôi nhưng 'ice cream' thì không. Chúng ta có thể thấy 'ice cream' giờ đã xuất hiện trong danh sách bạn bè, còn 'cannoli' thì không. Nếu chúng ta chỉ cần đặt friend_foods bằng my_foods, chúng ta sẽ không tạo ra hai danh sách riêng biệt. Ví dụ, đây là điều xảy ra khi bạn cố gắng sao chép một danh sách mà không sử dụng slice:

```
my_foods = ['pizza', 'falafel', 'bánh cà rốt']

# Điều này không hiệu quả:
friend_foods = my_foods

my_foods.append('cannoli')
friend_foods.append('kem')

print("Món ăn yêu thích của tôi là:")
in(my_foods)

print("\nMón ăn yêu thích của bạn tôi là:")
in(friend_foods)
```

Thay vì gán một bản sao của my_foods cho friend_foods, chúng ta đặt friend_foods bằng với my_foods. Cú pháp này thực sự yêu cầu Python liên kết biến friend_foods mới với danh sách đã được liên kết với my_foods, vì vậy bây giờ cả hai biến đều trỏ đến cùng một danh sách. Kết quả là, khi chúng ta thêm 'cannoli' vào my_foods, nó cũng sẽ xuất hiện trong friend_foods. Tương tự, 'ice cream' sẽ xuất hiện trong cả hai danh sách, mặc dù nó có vẻ chỉ được thêm vào friend_foods.

Kết quả đầu ra cho thấy cả hai danh sách đều giống nhau, đây không phải là điều chúng tôi mong muốn:

```
Những món ăn tôi thích nhất là:
['pizza', 'falafel', 'bánh cà rốt', 'cannoli', 'kem']

Những món ăn yêu thích của bạn tôi là:
['pizza', 'falafel', 'bánh cà rốt', 'cannoli', 'kem']
```

LƯU Ý: ng lo lắng về các chi tiết trong ví dụ này ngay bây giờ. Nếu bạn đang cố gắng làm việc với một bản sao của danh sách và thấy có hiện tượng bất thường, hãy đảm bảo bạn đang sao chép danh sách bằng một lát cắt, như chúng ta đã làm trong ví dụ đầu tiên.

HÃY TỰ THỬ

4-10. Các lát cắt: Sử dụng một trong những chương trình bạn đã viết trong chương này, thêm một số dòng vào cuối chương trình để thực hiện những chức năng sau:

- In tin nhắn Ba mục đầu tiên trong danh sách là:. Sau đó sử dụng một lát cắt để in ra ba mục đầu tiên trong danh sách của chương trình đó.
- In thông báo Ba mục từ giữa danh sách là:. Sau đó, sử dụng một lát cắt để in ba mục từ giữa danh sách.
- In thông báo Ba mục cuối cùng trong danh sách là:. Sau đó, sử dụng lát cắt để in ba mục cuối cùng trong danh sách.

4-11. Pizza của tôi, Pizza của bạn: Bắt đầu với chương trình từ Bài tập 4-1 (trang 56). Tạo một bản sao của danh sách pizza và đặt tên là friend_pizzas. Sau đó, thực hiện như sau:

- Thêm một chiếc pizza mới vào danh sách ban đầu.
- Thêm một loại pizza khác vào danh sách friend_pizzas.
- Chứng minh rằng bạn có hai danh sách riêng biệt. In thông báo My favorite pizzas là:, sau đó sử dụng vòng lặp for để in danh sách đầu tiên. In thông báo My friend's favorite pizzas are:, sau đó sử dụng vòng lặp for để in danh sách thứ hai. Đảm bảo mỗi chiếc pizza mới đều được lưu trữ trong danh sách phù hợp.

4-12. Thêm vòng lặp: Tất cả các phiên bản foods.py trong phần này đều tránh sử dụng vòng lặp for khi in để tiết kiệm dung lượng. Chọn một phiên bản foods.py và viết hai vòng lặp for để in mỗi danh sách thực phẩm.

Bộ

Danh sách hoạt động tốt để lưu trữ các tập hợp mục có thể thay đổi trong suốt vòng đời của một chương trình. Khả năng sửa đổi danh sách đặc biệt quan trọng khi bạn làm việc với danh sách người dùng trên một trang web hoặc danh sách nhân vật trong trò chơi.
Tuy nhiên, đôi khi bạn sẽ muốn tạo một danh sách các mục không thể thay đổi.
Tuple cho phép bạn làm điều đó. Python gọi các giá trị không thể thay đổi là bất biến, và một danh sách bất biến được gọi là tuple.

Định nghĩa một Tuple

Một tuple trông giống hệt một danh sách, ngoại trừ việc bạn sử dụng dấu ngoặc đơn thay vì dấu ngoặc vuông. Sau khi định nghĩa một tuple, bạn có thể truy cập từng phần tử bằng cách sử dụng chỉ mục của từng mục, giống như cách bạn làm với một danh sách.

Ví dụ, nếu chúng ta có một hình chữ nhật luôn có một kích thước nhất định, chúng ta có thể đảm bảo kích thước của nó không thay đổi bằng cách đưa các kích thước vào một bộ:

```
dimensions.py kích thước = (200, 50)
in(kích thước[0])
in(kích thước[1])
```

Chúng ta định nghĩa kích thước của bộ , sử dụng dấu ngoặc đơn thay vì dấu ngoặc vuông. Sau đó, chúng ta in từng phần tử trong bộ riêng lẻ, sử dụng cùng cú pháp mà chúng ta đã sử dụng để truy cập các phần tử trong danh sách:

```
200
50
```

Chúng ta hãy xem điều gì xảy ra nếu chúng ta thử thay đổi một trong các mục trong chiều của bộ:

```
kích thước = (200, 50)
kích thước[0] = 250
```

Mã này cố gắng thay đổi giá trị của chiều đầu tiên, nhưng Python trả về lỗi kiểu. Vì chúng ta đang cố gắng thay đổi một bộ, điều không thể thực hiện được với kiểu đối tượng đó, Python cho chúng ta biết rằng chúng ta không thể gán giá trị mới cho một mục trong bộ:

```
Theo dõi (cuộc gọi gần đây nhất cuối cùng):
  Tệp "dimensions.py", dòng 2, trong <module>
    kích thước[0] = 250
TypeError: đối tượng 'tuple' không hỗ trợ gán mục
```

Điều này có lợi vì chúng ta muốn Python đưa ra lỗi khi một dòng mã cố gắng thay đổi kích thước của hình chữ nhật.

LƯU Ý mặt kỹ thuật, các bộ được định nghĩa bằng dấu phẩy; dấu ngoặc đơn giúp chúng trông gọn gàng và dễ đọc hơn. Nếu bạn muốn định nghĩa một bộ với một phần tử, bạn cần thêm dấu phẩy ở cuối:

```
my_t = (3,)
```

Việc xây dựng một bộ với một phần tử thường không có ý nghĩa, nhưng điều này có thể xảy ra khi các bộ dữ liệu được tạo tự động.

Lặp qua tất cả các giá trị trong một Tuple

Bạn có thể lặp qua tất cả các giá trị trong một tuple bằng vòng lặp for , giống như bạn đã làm với một danh sách:

```
kích thước = (200, 50)
cho kích thước trong kích thước:
    in(kích thước)
```

Python trả về tất cả các phần tử trong tuple, giống như cách nó làm với một danh sách:

```
200
50
```

Viết trên một Tuple

Mặc dù bạn không thể sửa đổi một bộ, bạn có thể gán một giá trị mới cho một biến đại diện cho một bộ. Ví dụ, nếu chúng ta muốn thay đổi kích thước của hình chữ nhật này, chúng ta có thể định nghĩa lại toàn bộ bộ:

```
kích thước = (200, 50)
print("Kích thước ban đầu:")
cho kích thước trong kích thước:
    in(kích thước)

kích thước = (400, 100)
print("\nKích thước đã sửa đổi:")
cho kích thước trong kích thước:
    in(kích thước)
```

Bốn dòng đầu tiên định nghĩa bộ dữ liệu gốc và in ra các chiều ban đầu. Sau đó, chúng ta liên kết một bộ dữ liệu mới với biến dimensions và in ra các giá trị mới. Lần này Python không phát sinh lỗi nào, vì việc gán lại biến là hợp lệ:

```
Kích thước ban đầu:
200
50

Kích thước đã sửa đổi:
400
100
```

So với danh sách, bộ dữ liệu là những cấu trúc dữ liệu đơn giản. Sử dụng chúng khi bạn muốn lưu trữ một tập hợp các giá trị không nên thay đổi trong suốt vòng đời của chương trình.

HÃY TỰ THỬ

4-13. Tiệc buffet: Nhà hàng kiểu buffet chỉ phục vụ năm món ăn cơ bản. Hãy nghĩ ra năm món ăn đơn giản và xếp chúng vào một bộ.

- Sử dụng vòng lặp for để in ra từng món ăn mà nhà hàng cung cấp.
- Hãy thử sửa đổi một trong các mục và đảm bảo rằng Python từ chối thay đổi.
- Nhà hàng thay đổi thực đơn, thay thế hai món bằng các món ăn khác. Thêm một dòng để viết lại bộ dữ liệu, sau đó sử dụng vòng lặp for để in từng món trong thực đơn đã sửa đổi.

Định dạng mã của bạn

Giờ đây, khi bạn đang viết những chương trình dài hơn, việc học cách định dạng mã một cách nhất quán là một ý tưởng hay. Hãy dành thời gian để làm cho mã của bạn dễ đọc nhất có thể. Viết mã dễ đọc giúp bạn theo dõi những gì chương trình đang làm và giúp người khác hiểu mã của bạn.

Các lập trình viên Python đã thống nhất một số quy ước về kiểu dáng để đảm bảo mã của mọi người đều được cấu trúc gần như giống nhau. Một khi đã học được cách viết mã Python sạch, bạn sẽ có thể hiểu được cấu trúc tổng thể của mã Python của bất kỳ ai khác, miễn là họ tuân theo cùng một nguyên tắc. Nếu bạn hy vọng trở thành một lập trình viên chuyên nghiệp, bạn nên bắt đầu làm theo những nguyên tắc này càng sớm càng tốt để hình thành thói quen tốt.

Hướng dẫn phong cách

Khi ai đó muốn thay đổi ngôn ngữ Python, họ sẽ viết một Đề xuất Cải tiến Python (PEP). Một trong những PEP lâu đời nhất là PEP 8, hướng dẫn các lập trình viên Python cách định dạng mã của họ. PEP 8 khá dài, nhưng phần lớn nội dung của nó liên quan đến các cấu trúc mã hóa phức tạp hơn những gì bạn đã thấy cho đến nay.

Hướng dẫn về phong cách Python được viết với sự thấu hiểu rằng mã được đọc thường xuyên hơn là được viết ra. Bạn sẽ viết mã một lần và sau đó bắt đầu đọc nó khi bắt đầu gỡ lỗi. Khi bạn thêm tính năng vào chương trình, bạn sẽ dành nhiều thời gian hơn để đọc mã. Khi bạn chia sẻ mã của mình với các lập trình viên khác, họ cũng sẽ đọc mã của bạn.

Khi được lựa chọn giữa việc viết mã dễ viết hơn hoặc mã dễ đọc hơn, các lập trình viên Python gần như luôn khuyến khích bạn viết mã dễ đọc hơn. Các hướng dẫn sau đây sẽ giúp bạn viết mã rõ ràng ngay từ đầu.

thụt lề

PEP 8 khuyến nghị bạn nên sử dụng bốn khoảng trắng cho mỗi mức thụt lề. Việc sử dụng bốn khoảng trắng giúp cải thiện khả năng đọc, đồng thời vẫn có đủ chỗ cho nhiều mức thụt lề trên mỗi dòng.

Trong một tài liệu soạn thảo văn bản, mọi người thường sử dụng phím tab thay vì phím khoảng trắng để thụt lề. Cách này hiệu quả với các tài liệu soạn thảo văn bản, nhưng trình thông dịch Python sẽ bị nhầm lẫn khi phím tab được kết hợp với phím khoảng trắng. Mọi trình soạn thảo văn bản đều cung cấp một thiết lập cho phép bạn sử dụng phím TAB nhưng sau đó chuyển đổi mỗi tab thành một số lượng khoảng trắng nhất định. Bạn chắc chắn nên sử dụng phím TAB, nhưng cũng đảm bảo trình soạn thảo của bạn được thiết lập để chèn phím khoảng trắng thay vì phím tab vào tài liệu.

Việc trộn lẫn tab và khoảng trắng trong tệp có thể gây ra các vấn đề rất khó chẩn đoán. Nếu bạn nghĩ tệp của mình có sự trộn lẫn giữa tab và khoảng trắng, bạn có thể chuyển đổi tất cả tab trong tệp thành khoảng trắng trong hầu hết các trình soạn thảo.

Độ dài dòng

Nhiều lập trình viên Python khuyến nghị mỗi dòng nên ít hơn 80 ký tự. Trước đây, nguyên tắc này được đưa ra vì hầu hết máy tính chỉ có thể chứa 79 ký tự trên một dòng trong cửa sổ terminal.

Hiện nay, mọi người có thể viết nhiều dòng hơn trên màn hình, nhưng vẫn còn nhiều lý do khác để tuân thủ độ dài dòng chuẩn là 79 ký tự.

Các lập trình viên chuyên nghiệp thường mở nhiều tệp trên cùng một màn hình, và việc sử dụng độ dài dòng tiêu chuẩn cho phép họ xem toàn bộ các dòng trong hai hoặc ba tệp được mở cạnh nhau trên màn hình. PEP 8 cũng khuyến nghị bạn nên giới hạn tất cả các chú thích ở mức 72 ký tự mỗi dòng, vì một số công cụ tạo tài liệu tự động cho các dự án lớn sẽ thêm các ký tự định dạng vào đầu mỗi dòng được chú thích.

Hướng dẫn về độ dài dòng của PEP 8 không phải là bất di bất dịch, và một số nhóm thích giới hạn 99 ký tự. Đừng quá lo lắng về độ dài dòng trong mã của bạn khi đang học, nhưng hãy lưu ý rằng những người làm việc nhóm hầu như luôn tuân theo hướng dẫn của PEP 8. Hầu hết các trình soạn thảo đều cho phép bạn thiết lập một dấu hiệu trực quan, thường là một đường thẳng đứng trên màn hình, để cho bạn biết giới hạn này nằm ở đâu.

LƯU Ý

lục B cho bạn biết cách cấu hình trình soạn thảo văn bản của bạn để nó luôn chèn bốn khoảng trắng mỗi khi bạn nhấn phím TAB và hiển thị hướng dẫn theo chiều dọc để giúp bạn tuân thủ giới hạn 79 ký tự.

Dòng trống

Để nhóm các phần trong chương trình một cách trực quan, hãy sử dụng các dòng trống. Bạn nên sử dụng các dòng trống để sắp xếp các tệp, nhưng đừng lạm dụng. Bằng cách làm theo các ví dụ được cung cấp trong sách này, bạn sẽ đạt được sự cân bằng phù hợp.
Ví dụ, nếu bạn có năm dòng mã để xây dựng một danh sách và ba dòng mã khác thực hiện một thao tác nào đó với danh sách đó, thì việc đặt một dòng trống giữa hai phần là hợp lý. Tuy nhiên, bạn không nên đặt ba hoặc bốn dòng trống giữa hai phần.

Các dòng trống sẽ không ảnh hưởng đến cách mã của bạn chạy, nhưng chúng sẽ ảnh hưởng đến khả năng đọc mã. Trình thông dịch Python sử dụng thụt lề ngang để diễn giải ý nghĩa mã của bạn, nhưng lại bỏ qua khoảng cách dọc.

Các hướng dẫn về phong cách khác

PEP 8 có nhiều khuyến nghị bổ sung về kiểu dáng, nhưng hầu hết các hướng dẫn đều đề cập đến các chương trình phức tạp hơn những gì bạn đang viết tại thời điểm này. Khi bạn tìm hiểu các cấu trúc Python phức tạp hơn, tôi sẽ chia sẻ các phần liên quan của hướng dẫn PEP 8.

HÃY TỰ THỬ

4-14. PEP 8: Xem qua hướng dẫn về phong cách PEP 8 gốc tại https://python.org/dev/peps/pep-0008. Bạn sẽ không sử dụng nhiều nội dung này bây giờ, nhưng có thể sẽ thú vị nếu lướt qua.

4-15. Xem xét mã: Chọn ba chương trình bạn đã viết trong chương này và sửa đổi từng chương trình để tuân thủ PEP 8.

- Sử dụng bốn khoảng trắng cho mỗi mức thụt lề. Cài đặt trình soạn thảo văn bản của bạn để chèn bốn khoảng trắng mỗi khi bạn nhấn phím TAB, nếu bạn chưa thực hiện (xem Phụ lục B để biết hướng dẫn về cách thực hiện).
- Sử dụng ít hơn 80 ký tự trên mỗi dòng và thiết lập trình soạn thảo của bạn để hiển thị đường dẫn dọc ở vị trí ký tự thứ 80.
- Không nên sử dụng quá nhiều dòng trống trong tệp chương trình của bạn.

Bản tóm tắt

Trong chương này, bạn đã học cách làm việc hiệu quả với các phần tử trong danh sách. Bạn đã học cách xử lý danh sách bằng vòng lặp for , cách Python sử dụng thụt lề để cấu trúc chương trình và cách tránh một số lỗi thụt lề thường gặp. Bạn đã học cách tạo danh sách số đơn giản, cũng như một số thao tác bạn có thể thực hiện trên danh sách số. Bạn đã học cách cắt một danh sách để làm việc với một tập hợp con các mục và cách sao chép danh sách đúng cách bằng cách sử dụng lát cắt. Bạn cũng đã học về bộ dữ liệu, cung cấp một mức độ bảo vệ nhất định cho một tập hợp các giá trị không nên thay đổi, và cách định dạng mã ngày càng phức tạp của bạn để dễ đọc.

Trong Chương 5, bạn sẽ học cách phản hồi phù hợp với các điều kiện khác nhau bằng cách sử dụng câu lệnh if . Bạn sẽ học cách kết hợp các tập hợp kiểm tra điều kiện tương đối phức tạp để phản hồi phù hợp với chính xác loại tình huống hoặc thông tin bạn đang tìm kiếm. Bạn cũng sẽ học cách sử dụng câu lệnh if khi lặp qua một danh sách để thực hiện các hành động cụ thể với các phần tử được chọn từ danh sách đó.