

**BỘ THÔNG TIN VÀ TRUYỀN THÔNG
HỌC VIỆN CÔNG NGHỆ BUUTURE CHÍNH VIỄN THÔNG**



Bài giảng:

LẬP TRÌNH VỚI PYTHON

Biên soạn: Ths. Nguyễn Hoàng Anh
Ths. Đặng Ngọc Hùng

2022

MỤC LỤC

MỤC LỤC.....	11
DANH MỤC HÌNH	17
LỜI GIỚI THIỆU	19
CHƯƠNG 1. GIỚI THIỆU	21
1.1. Lịch sử Python	21
1.2. Bắt đầu với Python.....	23
1.2.1. Cài đặt môi trường.....	23
1.2.2. Cài đặt trình soạn thảo	24
1.3. Viết chương trình đầu tiên ‘HelloWorld.py’	30
1.3.1. Chạy hello_world.py.....	31
1.3.2. Xử lý lỗi.....	32
1.3.3. Chạy chương trình Python từ Terminal.....	33
1.3.4. Chạy trên macOS and Linux	33
Kết chương	34
CHƯƠNG 2. BIẾN VÀ NHỮNG KIỂU DỮ LIỆU ĐƠN GIẢN	35
2.1. Biến	35
2.1.1. Định danh và sử dụng các biến.....	36
2.1.2. Tránh đặt tên lỗi khi sử dụng các biến	37
2.1.3. Biến là các nhãn	38
2.2. Chuỗi (Strings).....	38
2.2.1. Sử dụng chữ hoa trong chuỗi với phương thức.....	39
2.2.2. Sử dụng biến trong chuỗi.....	40
2.2.3. Thêm khoảng trắng vào chuỗi với Tab hoặc dòng mới	41
2.2.4. Bỏ khoảng trắng	42
2.2.5. Tránh lỗi cú pháp với String	43
2.3. Số (Numbers).....	44
2.3.1. Số nguyên - Integers	44
2.3.2. Số thực - Floats	45
2.3.3. Số nguyên và số thực	46
2.3.4. Các gạch dưới trong số.....	46
2.3.5. Gán nhiều biến cùng lúc	46
2.3.6. Hằng số - Constants	47
2.4. Chú thích (Comments)	47
2.4.1. Cách viết các chú thích.....	47
2.4.2. Loại chú thích nên viết.....	47
Bài tập chương 2	48
Kết chương	49
CHƯƠNG 3. DANH SÁCH (LIST).....	51
3.1. Định nghĩa Danh sách	51
3.1.1. Truy cập các phần tử trong danh sách	51
3.1.2. Đánh chỉ mục	52

3.1.3. Sử dụng giá trị riêng từ danh sách.....	53
3.2. Thay đổi, thêm, và xóa các phần tử	53
3.2.1. Thay đổi phần tử trong danh sách.....	53
3.2.2. Thêm phần tử vào danh sách	54
3.2.3. Xóa phần tử khỏi danh sách.....	55
3.3. Tô chúc danh sách.....	59
3.3.1. Sắp xếp danh sách vĩnh viễn với phương thức sort()	59
3.3.2. Sắp xếp danh sách tạm thời với phương thức sorted().....	59
3.3.3. In danh sách theo thứ tự ngược.....	60
3.3.4. Tìm độ dài của danh sách	61
3.4. Tránh lỗi chỉ mục	61
3.5. Lặp qua toàn bộ danh sách	62
3.5.1. Cái nhìn cận cảnh hơn về vòng lặp	63
3.5.2. Bổ sung thêm về vòng lặp	64
3.5.3. Một số việc sau vòng lặp For	65
3.5.4. Tránh lỗi thuật lè.....	66
3.6. Lập danh sách số	69
3.6.1. Sử dụng hàm range()	70
3.6.2. Sử dụng range() để tạo ra danh sách số.....	71
3.6.3. Thống kê đơn giản với danh sách số	72
3.6.4. Hiểu biết về danh sách.....	72
3.7. Làm việc với một phần của danh sách	73
3.7.1. Cắt lát một danh sách	73
3.7.2. Lặp qua một lát cắt.....	74
3.7.3. Sao chép danh sách	75
3.8. Tuples	77
3.8.1. Định nghĩa một Tuple	78
3.8.2. Lặp qua toàn bộ giá trị trong Tuple	79
3.8.3. Ghi lên Tuple.....	79
Bài tập chương 3	80
Kết chương	84
CHƯƠNG 4. CÂU LỆNH RẼ NHÁNH (IF)	86
4.1. Ví dụ	86
4.2. Kiểm tra có điều kiện	87
4.2.1. Kiểm tra đẳng thức	87
4.2.2. Bỏ qua chữ viết hoa khi kiểm tra đẳng thức	88
4.2.3. Kiểm tra bất đẳng thức	89
4.2.4. So sánh số.....	89
4.2.5. Kiểm tra nhiều điều kiện	90
4.3. Câu lệnh If	93
4.3.1. Câu lệnh if đơn giản	93
4.3.2. Câu lệnh if-else.....	94
4.3.3. Chuỗi if-elif-else	95
4.3.4. Sử dụng nhiều khối elif	96
4.3.5. Bỏ qua khối else	97
4.3.6. Kiểm tra nhiều điều kiện	98
4.4. Câu lệnh If với danh sách	99

4.4.1. Kiểm tra các phần tử đặc biệt	99
4.4.2. Kiểm tra một danh sách không rỗng	100
4.4.3. Sử dụng nhiều danh sách	101
Bài tập chương 4	102
Kết chương	106
CHƯƠNG 5. TỪ ĐIỂN (DICTIONARIES)	107
5.1. Ví dụ đơn giản về từ điển	107
5.2. Làm việc với từ điển	108
5.2.1. Truy cập các giá trị trong từ điển	108
5.2.2. Thêm các cặp khoá-giá trị mới	109
5.2.3. Bắt đầu với từ điển rỗng	109
5.2.4. Sửa giá trị trong từ điển	110
5.2.5. Xóa các cặp khoá-giá trị	111
5.2.6. Từ điển của các đối tượng tương tự	112
5.2.7. Sử dụng get() để truy cập các giá trị	113
5.3. Lặp qua toàn bộ từ điển	114
5.3.1. Lặp qua tất cả các cặp khoá-giá trị	114
5.3.2. Lặp qua tất cả các khoá trong từ điển	116
5.3.3. Lặp các khoá của từ điển theo một thứ tự cụ thể	118
5.3.4. Lặp qua tất cả các giá trị trong từ điển	118
5.4. Nesting	120
5.4.1. Danh sách các từ điển	120
5.4.2. Danh sách trong từ điển	123
5.4.3. Từ điển bên trong từ điển	125
Bài tập chương 5	126
Kết chương	128
CHƯƠNG 6. ĐẦU VÀO CỦA NGƯỜI DÙNG (INPUT) VÀ VÒNG LẶP	129
6.1. Cách hàm input() hoạt động	129
6.1.1. Viết lời nhắc rõ ràng	130
6.1.2. Sử dụng int() để nhận đầu vào số nguyên	130
6.1.3. Toán tử modulo	132
6.2. Giới thiệu vòng lặp while	133
6.2.1. Vòng lặp với hành động	133
6.2.2. Để người dùng lựa chọn khi nào thoát	133
6.2.3. Sử dụng Flag (cờ)	135
6.2.4. Sử dụng break để thoát khỏi vòng lặp	137
6.2.5. Sử dụng continue trong vòng lặp	138
6.2.6. Tránh lặp vô hạn	138
6.3. Sử dụng vòng lặp while với Danh sách và Từ điển	139
6.3.1. Chuyển phần tử từ danh sách này sang danh sách khác	140
6.3.2. Xóa tất cả các thẻ hiện của một giá trị trong một danh sách	141
6.3.3. Dièn từ điển với đầu vào người dùng	141
Bài tập chương 6	143
Kết chương	144
CHƯƠNG 7. HÀM	145
7.1. Định nghĩa hàm	145

7.1.1. Truyền thông tin tới một hàm.....	146
7.1.2. Đổi số và tham số	147
7.2. Truyền tham số	147
7.2.1. Đổi số có vị trí.....	147
7.2.2. Đổi số từ khóa	149
7.2.3. Giá trị mặc định	150
7.2.4. Lệnh gọi hàm tương đương.....	151
7.2.5. Tránh lỗi đổi số	152
7.3. Giá trị trả về	153
7.3.1. Trả về giá trị đơn giản.....	153
7.3.2. Tạo số đổi số tùy chọn	154
7.3.3. Trả về một từ điển.....	155
7.3.4. Sử dụng một hàm với vòng lặp while	156
7.4. Truyền một danh sách vào hàm	158
7.4.1. Sửa đổi danh sách trong một hàm.....	159
7.4.2. Ngăn một hàm sửa đổi danh sách	161
7.5. Truyền một số đổi số tùy ý	162
7.5.1. Kết hợp đổi số vị trí và tùy ý	163
7.5.2. Sử dụng đổi số từ khóa tùy ý	164
7.6. Hàm Lambda	165
7.6.1. Định nghĩa hàm Lambda	165
7.6.2. Sử dụng Lambda trong hàm filter	166
7.6.3. Sử dụng hàm Lambda trong hàm map().....	166
7.6.4. Sử dụng hàm Lambda trong Series và Dataframe	167
7.6.5. Sắp xếp dữ liệu nhờ hàm lambda	168
7.7. Lưu trữ hàm trong module	169
7.7.1. Import toàn bộ module.....	170
7.7.2. Import các hàm cụ thể	171
7.7.3. Sử dụng as để cấp cho hàm một bí danh (Alias).....	171
7.7.4. Sử dụng as để cấp cho một mô-đun một bí danh	172
7.7.5. Import tất cả các hàm trong module	173
Bài tập chương 7	173
Kết chương	175
CHƯƠNG 8. LỚP	177
8.1. Tạo và sử dụng lớp.....	178
8.1.1. Tạo ra lớp Dog	178
8.1.2. Tạo một thể hiện của lớp	180
8.2. Làm việc với lớp và thể hiện của lớp.....	182
8.2.1. Lớp Car.....	182
8.2.2. Thiết lập giá trị mặc định cho thuộc tính	183
8.2.3. Thay đổi giá trị thuộc tính	184
8.3. Ké thừa	187
8.3.1. Phương thức __init__() cho lớp con	187
8.3.2. Định nghĩa các thuộc tính và phương thức cho lớp con	189
8.3.3. Ghi đè phương thức từ lớp cha	190
8.3.4. Thuộc tính là một thể hiện lớp	190
8.4. Sử dụng các lớp.....	193

8.4.1. Nhập vào một lớp đơn.....	193
8.4.2. Lưu trữ nhiều lớp trong một module	194
8.4.3. Nhập vào nhiều lớp trong một module	196
8.4.4. Nhập vào toàn bộ module	196
8.4.5. Nhập vào toàn bộ các lớp trong một module.....	196
8.4.6. Nhập vào một module từ một module.....	197
8.4.7. Sử dụng bí danh.....	198
8.5. Thư viện chuẩn của python	198
8.5.1. Random	198
8.5.2. Numpy	200
8.5.3. Pandas.....	200
8.5.4. Statistics	201
Bài tập chương 8.....	202
Kết chương	205
CHƯƠNG 9. TỆP VÀ NGOẠI LỆ	206
9.1. Đọc từ file	206
9.1.1. Đọc toàn bộ tệp	207
9.1.2. Đường dẫn tệp.....	208
9.1.3. Đọc từng dòng.....	210
9.1.4. Tạo danh sách các dòng từ một tệp.....	211
9.1.5. Làm việc với nội dung của tệp	212
9.1.6. Tệp lớn: Một triệu chữ số	213
9.1.7. Tìm ngày sinh trong số PI	213
9.2. Ghi vào file	214
9.2.1. Ghi vào tệp rỗng.....	214
9.2.2. Ghi nhiều dòng.....	215
9.2.3. Thêm vào một tệp.....	216
9.3. Ngoại lệ	216
9.3.1. Xử lý ngoại lệ ZeroDivisionError	217
9.3.2. Sử dụng khối try-except	217
9.3.3. Sử dụng ngoại lệ để ngăn chặn sự cố.....	218
9.3.4. Khối Else	219
9.3.5. Xử lý ngoại lệ FileNotFoundError.....	220
9.3.6. Phân tích dữ liệu văn bản	221
9.3.7. Làm việc với nhiều tệp	222
9.3.8. Che giấu lỗi	224
9.3.9. Quyết định lỗi nào cần báo cáo	225
9.4. Lưu trữ dữ liệu	225
9.4.1. Sử dụng json.dump() và json.load()	226
9.4.2. Lưu và đọc dữ liệu do người dùng tạo	227
9.4.3. Tái cấu trúc	228
Bài tập chương 9	231
Kết chương	233
CHƯƠNG 10. DỰ ÁN	234
10.1. Trò chơi cuộc xâm lăng của quái vật	234
10.1.1. Tạo ra tàu bắn đạn	235
10.1.2. Tạo ra quái vật	262

<i>10.1.3. Ghi điểm</i>	282
10.2. Trực quan hóa dữ liệu	305
<i>10.2.1. Sản sinh dữ liệu</i>	306
<i>10.2.2. Tải dữ liệu về</i>	316
<i>10.2.3. Làm việc với APIs</i>	343
10.3. Ứng dụng web.....	357
<i>10.3.1. Bắt đầu với Django</i>	357
<i>10.3.2. Tài khoản người dùng</i>	388
<i>10.3.3. Định kiểu và triển khai ứng dụng</i>	416
10.4. Kết chương	442
TÀI LIỆU THAM KHẢO.....	443

DANH MỤC HÌNH

Hình 1.1. Lịch sử Python	23
Hình 1.2. Kiểm tra Python trên máy	25
Hình 1.3. Giao diện Console.....	26
Hình 1.4. Thiết lập cấu hình đường dẫn	27
Hình 10.1. Hình ảnh con tàu	242
Hình 10.2. Con tàu bắn đạn	258
Hình 10.3. Quái vật	262
Hình 10.4. Quái vật đầu tiên xuất hiện.....	264
Hình 10.5. Hạm đội quái vật.....	266
Hình 10.6. Đội quái vật đầy đủ	269
Hình 10.7. Bắn hạ quái vật	275
Hình 10.8. Quái vật bị bắn hạ	275
Hình 10.9. nút Play xuất hiện khi trò chơi chưa hoạt động.....	285
Hình 10.10. điểm số xuất hiện phía bên phải màn hình	293
Hình 10.11. Điểm cao (high score)	299
Hình 10.12. Cấp độ hoàn thành.....	301
Hình 10.13. Hệ thống tính điểm hoàn chỉnh.....	304
Hình 10.14. Biểu đồ đầu tiên	307
Hình 10.15. Bổ sung Tiêu đề và nhãn	308
Hình 10.16. Điều chỉnh trực tọa độ X	309
Hình 10.17. Định kiểu đồ thị	310
Hình 10.18. Scatter plot.....	311
Hình 10.19. Scatter plot với danh sách điểm.....	312
Hình 10.20. Vẽ 1000 điểm trên đồ thị	313
Hình 10.21. Tự định nghĩa màu	314
Hình 10.22. Màu gradient.....	315
Hình 10.23. Biểu đồ nhiệt độ đơn giản	321
Hình 10.24. Đồ thị có trục X theo thời gian	324
Hình 10.25. Đồ thị nhiệt độ cả năm	325
Hình 10.26. Đồ thị nhiệt độ cao – thấp năm 2018	326
Hình 10.27. Đồ bóng đồ thị nhiệt độ.....	327
Hình 10.28. Đồ thị kết quả.....	330
Hình 10.29. Biểu đồ Scattergeo	337
Hình 10.30. Hình ảnh các điểm trên biểu đồ	339
Hình 10.31. Biểu đồ có dải màu.....	340
Hình 10.32. Dải màu khác	341
Hình 10.33. Danh sách các kho hàng đầu trên Github.....	353
Hình 10.34. Tùy chỉnh biểu đồ Plotly	354
Hình 10.35. Thông tin khi di chuột	355
Hình 10.36. Giao diện cài đặt Django thành công	363
Hình 10.37. Giao diện quản trị.....	368
Hình 10.38. Trang chủ của trang Nhật ký học tập	378
Hình 10.39. Trang chủ đề	384
Hình 10.40. Trang chi tiết cho một chủ đề	387
Hình 10.41. Trang thêm chủ đề.....	393
Hình 10.42. Trang new_entry	397
Hình 10.43. Liên kết chỉnh sửa Entry	400

Hình 10.44. Trang Login	404
Hình 10.45. Xác nhận đăng xuất thành công.....	405
Hình 10.46. Áp dụng mẫu Bootstrap.....	418
Hình 10.47. Định kiểu trang đăng nhập bằng Bootstrap.....	424
Hình 10.48. Trang chủ đề với định kiểu Bootstrap.....	426

LỜI GIỚI THIỆU

Bài giảng này trình bày những khái niệm cơ bản mà chúng ta sẽ cần viết các chương trình Python. Nhiều những khái niệm này là chung cho tất cả các ngôn ngữ lập trình, vì vậy chúng sẽ hữu ích trong suốt cuộc đời lập trình viên.

Mục tiêu của bài giảng này là giúp chúng ta bắt kịp với Python một cách nhanh nhất có thể để chúng ta có thể xây dựng các chương trình hoạt động — trò chơi, trực quan hóa dữ liệu, và các ứng dụng web — đồng thời phát triển kỹ năng nền tảng trong lập trình. Bài giảng này dành cho những ai muốn tìm hiểu cơ bản về lập trình một cách nhanh chóng để có thể tập trung vào các dự án thú vị.

Mục đích của bài giảng này là giúp người đọc trở thành một lập trình viên giỏi nói chung và một lập trình viên Python giỏi nói riêng. Sau khi học bài giảng này, chúng ta có thể sẵn sàng chuyển sang các kỹ thuật Python nâng cao hơn và ngôn ngữ lập trình tiếp theo sẽ dễ dàng nắm bắt hơn nhờ các bài học rút ra sau bài giảng.

Bài giảng gồm 10 chương, trong đó:

Chương 1 giới thiệu và hướng dẫn cài đặt môi trường Python trên máy tính cá nhân và chạy chương trình Python đầu tiên, in ra dòng “Hello world” ra màn hình.

Chương 2 chúng ta sẽ học cách lưu thông tin vào các biến và làm việc với văn bản và giá trị số.

Chương 3 giới thiệu về Danh sách (list). Danh sách có thể chứa bao nhiêu thông tin tùy thích vào trong một biến, cho phép chúng ta làm việc với dữ liệu một cách hiệu quả. Chúng ta có thể làm việc với hàng trăm, hàng nghìn, thậm chí hàng triệu giá trị chỉ bằng một vài dòng code.

Trong chương 4, chúng ta sẽ học cách sử dụng câu lệnh rẽ nhánh (if). Câu lệnh rẽ nhánh phản hồi một cách nếu các điều kiện nhất định là đúng và phản hồi theo một cách khác nếu những điều kiện đó điều kiện không đúng.

Chương 5 giới thiệu cách sử dụng từ điển của Python. Từ điển cho phép chúng ta liên kết giữa phần khác nhau của thông tin. Cũng như danh sách, từ điển có thể chứa lượng thông tin tùy ý theo mong muốn.

Trong chương 6, chúng ta sẽ học cách ghi nhận thông tin đầu vào của người dùng để cho chương trình có tương tác. Chúng ta cũng sẽ học về vòng lặp while() để chạy một đoạn code lặp đi lặp lại cho đến khi điều kiện vẫn còn đúng.

Trong chương 7, chúng ta sẽ học cách viết hàm. Hàm là cách gọi một khối mã để thực thi một nhiệm vụ nhất định và có thể chạy bất cứ khi nào chúng ta cần.

Chương 8 giới thiệu về lớp (classes). Lớp cho phép chúng ta mô hình hóa những đối tượng trong thế giới thực như chó, mèo, con người, ô tô, tên lửa... nên chúng ta có thể lập trình bất cứ đối tượng thực hoặc trừu tượng nào.

Chương 9 chỉ cho chúng ta cách làm việc với tệp (files) và xử lý lỗi để chương trình không bị ngắt bát thình lình. Chúng ta sẽ lưu dữ liệu vào tệp trước khi chương trình được đóng lại và đọc dữ liệu lại khi chương trình chạy lại. Chúng ta sẽ học về ngoại lệ và cách xử lý ngoại lệ một cách linh hoạt.

Chương 10 giới thiệu về các dự án chúng ta sẽ hoàn thành trong khóa học. Chúng ta có thể lựa chọn dự án để hoàn thành bao gồm dự án về Game, Dự án về trực quan hóa dữ liệu và dự án về web. Khi hoàn thành các dự án này, chúng ta sẽ học được những kỹ năng khác nhau cho các dự án trong thực tế.

CHƯƠNG 1. GIỚI THIỆU

1.1. Lịch sử Python

Python được hình thành vào cuối những năm 1980. Thời gian đó, Guido van Rossum làm việc trong một dự án tại CWI, có tên là Amoeba, một hệ điều hành phân tán. Trong một cuộc phỏng vấn với Bill Venners¹, Guido van Rossum cho biết: "Vào đầu những năm 1980, tôi làm việc với tư cách là người triển khai nhóm xây dựng ngôn ngữ có tên là ABC tại Centrum voor Wiskunde en Informatica (CWI). Tôi không biết mọi người hiểu rõ như thế nào Ảnh hưởng của ABC đối với Python. Tôi cố gắng đề cập đến ảnh hưởng của ABC vì tôi mang ơn tất cả những gì tôi đã học được trong dự án đó và những người đã làm việc với nó". Sau đó, trong cùng một Cuộc phỏng vấn, Guido van Rossum tiếp tục: "Tôi nhớ lại tất cả kinh nghiệm của mình và một số thất vọng của tôi với ABC. Tôi quyết định cố gắng thiết kế một ngôn ngữ kịch bản đơn giản sở hữu một số đặc tính tốt hơn của ABC. Vì vậy tôi bắt đầu thực hiện. Tôi đã tạo một máy ảo đơn giản, một trình phân tích cú pháp đơn giản và một thời gian chạy đơn giản. Tôi đã tạo phiên bản của riêng mình cho các phần ABC khác nhau mà tôi thích. Tôi đã tạo một cú pháp cơ bản, sử dụng thuật lê để nhóm câu lệnh thay vì dấu ngoặc nhọn hoặc khối begin-end và phát triển một số lượng nhỏ các kiểu dữ liệu mạnh mẽ: bảng băm (hoặc từ điển, như chúng ta gọi), danh sách, chuỗi và số."

Vậy, lý do về tên "Python"? Hầu hết mọi người nghĩ về trăn, và thậm chí logo còn mô tả hai con trăn, nhưng nguồn gốc của cái tên bắt nguồn từ sự hài hước của người Anh. Guido van Rossum, người sáng tạo ra Python, đã viết vào năm 1996 về nguồn gốc của tên ngôn ngữ lập trình của mình: "Hơn sáu năm trước, vào tháng 12 năm 1989, tôi đang tìm kiếm một dự án lập trình 'sở thích' có thể khiến tôi bận rộn trong quá trình vào khoảng tuần lễ Giáng sinh. Tôi quyết định viết một thông dịch viên cho ngôn ngữ viết kịch bản mới mà tôi đã nghĩ đến gần đây: một kế thừa của ABC điều đó sẽ thu hút các tin tức Unix/C. Tôi đã chọn Python làm tiêu đề làm việc cho dự án, với tâm trạng hơi bất cần (và là một người hâm mộ lớn của Rạp xiếc bay của Monty Python).

Các định luật của Python:

1. Beautiful is better than ugly.

2. Explicit is better than implicit.
3. Simple is better than complex.
4. Complex is better than complicated.
5. Flat is better than nested.
6. Sparse is better than dense.
7. Readability counts.
8. Special cases aren't special enough to break the rules.
9. Although practicality beats purity.
10. Errors should never pass silently.
11. Unless explicitly silenced.
12. In the face of ambiguity, refuse the temptation to guess.
13. There should be one -- and preferably only one -- obvious way to do it.
14. Although that way may not be obvious at first unless you're Dutch.
15. Now is better than never.
16. Although never is often better than *right* now.
17. If the implementation is hard to explain, it's a bad idea.
18. If the implementation is easy to explain, it may be a good idea.
19. Namespaces are one honking great idea -- let's do more of those!

Guido Van Rossum đã xuất bản phiên bản đầu tiên của mã Python (phiên bản 0.9.0) tại alt.sources vào tháng 2 năm 1991. Bản phát hành này đã bao gồm xử lý ngoại lệ, các hàm và các kiểu dữ liệu cốt lõi của list, dict, str và các loại khác. Nó cũng hướng đối tượng và có một hệ thống mô-đun.



Hình 1.1. Lịch sử Python

Phiên bản Python 1.0 được phát hành vào tháng 1 năm 1994. Các tính năng mới chính bao gồm trong phiên bản này là các công cụ lập trình chức năng lambda, bản đồ, bộ lọc và giảm thiểu.

Sáu năm rưỡi sau vào tháng 10 năm 2000, Python 2.0 được giới thiệu. Bản phát hành này bao gồm toàn bộ danh sách, một bộ thu gom rác đầy đủ và nó hỗ trợ unicode.

Python phát triển mạnh trong 8 năm tiếp theo trong các phiên bản 2.x trước khi bản phát hành chính tiếp theo là Python 3.0 (còn được gọi là "Python 3000" và "Py3K") được phát hành. Python 3 không tương thích ngược với Python 2.x. Trọng tâm trong Python 3 là loại bỏ các cấu trúc và mô-đun lập trình trùng lặp, do đó hoàn thành hoặc gần như hoàn thành định luật thứ 13 của Zen của Python: "Nên có một - và tốt nhất là chỉ một - cách rõ ràng để làm nó."

1.2. Bắt đầu với Python

Trong chương này, ta sẽ chạy Python đầu tiên: chương trình, `hello_world.py`. Trước tiên, cần để kiểm tra xem phiên bản Python gần đây có được cài đặt trên máy tính hay không; nếu không, trước tiên cần cài đặt python. Tiếp theo cần cài đặt một trình soạn thảo văn bản để làm việc với các chương trình Python. Các trình soạn thảo văn bản nhận ra mã Python và đánh dấu các phần khi viết, làm tường minh các cấu trúc mã của chương trình.

1.2.1. Cài đặt môi trường

1.2.1.1. Các phiên bản của Python

Cách cài đặt python trên các môi trường khác nhau là khác nhau. Chúng ta cần để ý để thiết lập cho đúng. Thời điểm viết tài liệu này, phiên bản mới nhất là Python 3.7, nhưng mọi nội dung trong bài giảng này cần chạy trên Python 3.6 hoặc mới hơn.

Một số dự án Python cũ vẫn sử dụng Python 2, nhưng ta nên sử dụng Python 3. Nếu Python 2 được cài đặt trên hệ thống, thì có thể nó sẽ hỗ trợ một số chương trình cũ hơn mà hệ thống cần, tuy nhiên, cần cài bổ sung phiên bản mới hơn để chạy các dự án mới.

1.2.1.2. Chạy một đoạn mã Python

Chúng ta có thể chạy trình thông dịch của Python trong cửa sổ đầu cuối, cho phép ta thử các bit mã Python mà không cần phải lưu và chạy toàn bộ chương trình.

Ví dụ:

```
>>> print("Hello Python interpreter!")
Hello Python interpreter!
```

Dấu nhắc >>> chỉ ra rằng ta nên sử dụng cửa sổ đầu cuối và văn bản in đậm là mã nên nhập vào và sau đó thực thi bằng cách nhấn enter.

Hầu hết các ví dụ trong bài giảng là các chương trình nhỏ, độc lập mà ta sẽ chạy từ trình soạn thảo văn bản thay vì phiên đầu cuối, vì ta sẽ viết hầu hết mã của mình trong trình soạn thảo văn bản. Nhưng đôi khi các khái niệm cơ bản sẽ được hiển thị trong một loạt các đoạn mã chạy tại phiên đầu cuối Python để chứng minh các khái niệm cụ thể một cách hiệu quả hơn.

Khi thấy ba dấu ngoặc nhọn trong danh sách mã đang xem thì đó là đầu ra từ một phiên đầu cuối. Chúng ta cũng sẽ sử dụng trình chỉnh sửa văn bản để tạo một chương trình đơn giản có tên Hello World! đó đã trở thành một công việc cơ bản của việc học lập trình. Có một truyền thống lâu dài trong thế giới lập trình rằng in thông báo Hello world! đến màn hình tại chương trình đầu tiên bằng một ngôn ngữ mới sẽ mang lại điều may mắn. Một chương trình đơn giản như vậy phục vụ một mục đích rất thực tế. Nếu nó chạy chính xác trên hệ thống, bất kỳ chương trình Python nào được viết cũng sẽ hoạt động.

1.2.2. Cài đặt trình soạn thảo

Sublime Text là một trình soạn thảo văn bản đơn giản có thể được cài đặt trên tất cả các hệ điều hành. Sublime Text cho phép chúng ta chạy hầu hết tất cả các chương trình trực tiếp trong phần gốc code thay vì thông qua phiên đầu cuối. Code sẽ được chạy trong một phiên đầu cuối được nhúng trong cửa sổ Sublime Text, nhờ đó có thể dễ dàng thấy đầu ra. Sublime Text là một trình soạn thảo thân thiện với người

mới bắt đầu, nhưng nhiều lập trình viên chuyên nghiệp cũng sử dụng nó. Nếu cảm thấy thoải mái khi sử dụng nó trong khi học Python, chúng ta có thể tiếp tục sử dụng Sublime Text khi tiến tới các dự án lớn hơn và phức tạp hơn.

Python trên các hệ điều hành khác nhau

Python là ngôn ngữ đa nền tảng, do vậy, nó chạy trên hầu hết những hệ điều hành phổ biến. Bất cứ chương trình nào được viết bằng python đều có thể chạy trên máy tính có cài Python. Tuy nhiên, các phương pháp thiết lập Python trên các hệ điều hành khác nhau có hơi khác nhau.

Trong phần này, chúng ta học cách thiết lập Python trên nền tảng của mình. Chúng ta sẽ kiểm tra xem phiên bản gần đây của Python có được cài trên hệ điều hành hay không. Sau đó chúng ta cài Sublime Text. Hai bước đó là hai bước khác nhau cho mỗi hệ điều hành.

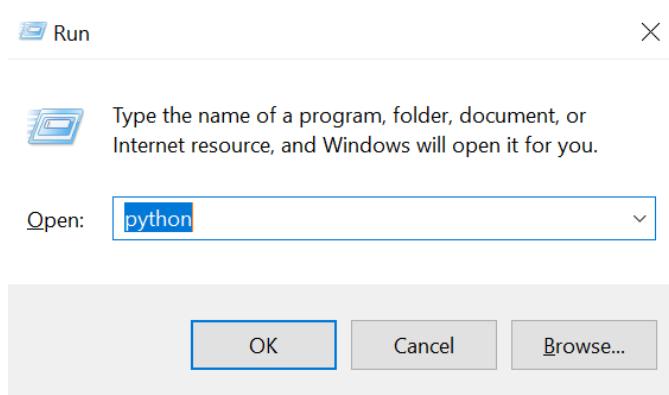
Trong phần tiếp theo, chúng ta sẽ chạy chương trình đầu tiên là Hello_world và sửa lỗi nếu có vấn đề xảy ra. Mỗi hệ điều hành sẽ được hướng dẫn những bước này để có một môi trường lập trình Python thân thiện.

1.2.2.1. Python trên Windows

Không phải lúc nào Windows cũng đi kèm với Python, vì vậy có thể ta sẽ cần cài đặt nó, và sau đó cài đặt Sublime Text

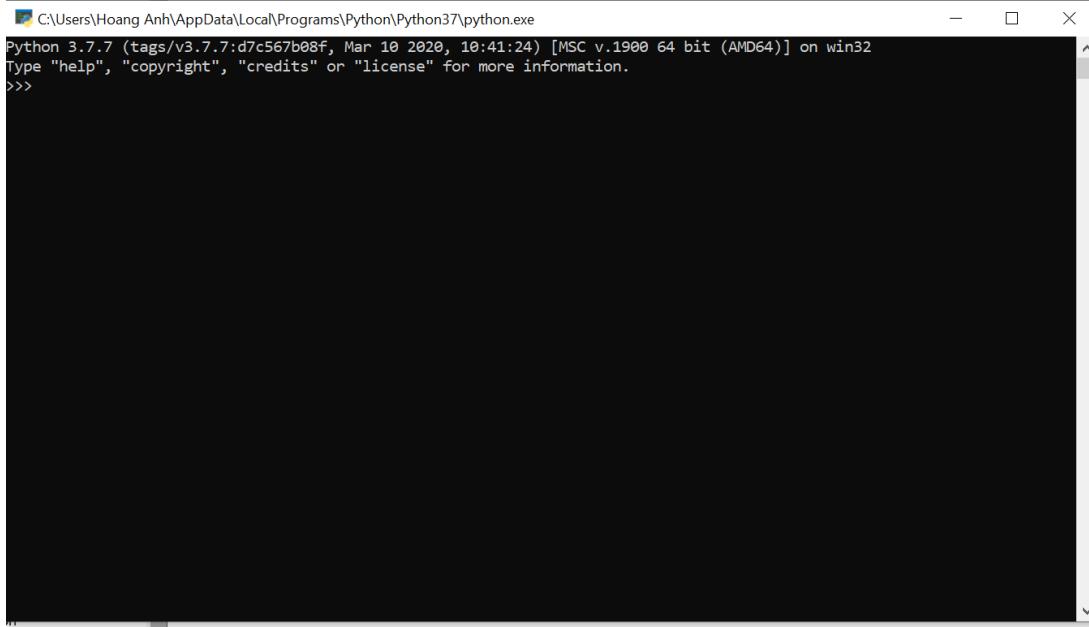
Cài đặt Python

Trước tiên, hãy kiểm tra xem Python đã được cài đặt trên hệ thống chưa. Mở một lệnh cửa sổ bằng cách nhập lệnh vào menu Start hoặc bằng cách giữ phím SHIFT trong khi nhấp chuột phải vào màn hình và chọn Mở cửa sổ lệnh tại đây từ menu.



Hình 1.2. Kiểm tra Python trên máy

Trong cửa sổ dòng lệnh, nhập python bằng chữ thường. Nếu ta nhận được lời nhắc Python (>>>) phản hồi, Python đã được cài đặt trên hệ thống. Nếu thấy thông báo lỗi cho ta biết rằng python không phải là lệnh được công nhận, Python chưa được cài đặt.



Hình 1.3. Giao diện Console

Trong trường hợp python đã cài, hoặc nếu thấy phiên bản Python cũ hơn Python 3.6, ta cần tải xuống trình cài đặt Python cho Windows. Truy cập <https://python.org/> và di chuột qua liên kết Tải xuống.

Ta sẽ thấy một nút để tải xuống phiên bản Python mới nhất. Nhấp vào nút, nút này sẽ tự động bắt đầu tải xuống trình cài đặt chính xác cho hệ thống. Sau khi đã tải xuống phiên bản mới, hãy chạy trình cài đặt. Đảm bảo rằng ta chọn tùy chọn *Thêm Python vào PATH*, điều này sẽ giúp dễ dàng xác nhận hệ thống một cách chính xác. Hình 1.4 cho thấy tùy chọn này được chọn.



Hình 1.4. Thiết lập cấu hình đường dẫn

Chạy Python ở Terminal

Mở cửa sổ lệnh và nhập python bằng chữ thường. Ta sẽ thấy lời nhắc Python (>>>), có nghĩa là Windows đã tìm thấy phiên bản của Python vừa được cài đặt.

```
Python 3.7.7 (tags/v3.7.7:d7c567b08f, Mar 10 2020, 10:41:24) [MSC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Nhập dòng sau vào phiên Python và đảm bảo rằng ta thấy đầu ra: Hello Python interpreter!

```
>>> print("Hello Python interpreter!")
Hello Python interpreter!
>>>
```

Bất kỳ lúc nào muốn chạy một đoạn mã Python, hãy mở cửa sổ lệnh và bắt đầu phiên đầu cuối Python. Để đóng phiên đầu cuối, nhấn **cTrL-Z** rồi nhấn **enTer** hoặc nhập lệnh `exit()`.

Cài đặt Sublime Text

Chúng ta có thể tải xuống trình cài đặt cho Sublime Text tại <https://sublimetext.com/>. Nhập vào liên kết tải xuống và tìm trình cài đặt Windows. Sau khi tải xuống trình cài đặt, hãy chạy trình cài đặt và chấp nhận tất cả các giá trị mặc định của nó.

1.2.2.2. Python trên macOS

Python đã được cài đặt trên hầu hết các hệ thống macOS, nhưng rất có thể đây là phiên bản lỗi thời mà chúng ta không muốn học. Trong phần này, ta sẽ cài đặt phiên bản Python mới nhất, sau đó ta sẽ cài đặt Sublime Text và đảm bảo rằng nó được định cấu hình chính xác.

Kiểm tra xem sự cài đặt của Python 3

Mở cửa sổ dòng lệnh bằng cách đi tới Applications-> Utilities-> Terminal. Ta cũng có thể nhấn tổ hợp phím command +s phím cách, nhập terminal, sau đó nhấn enter. Để xem phiên bản Python nào được cài đặt, hãy nhập python bằng chữ thường p — thao tác này cũng khởi động trình thông dịch Python bên trong terminal, cho phép nhập các lệnh Python.

Ta sẽ thấy đâu ra cho ta biết phiên bản Python nào được cài đặt trên hệ thống và một dấu nhắc >>> nơi có thể bắt đầu nhập các lệnh Python, như sau:

```
$ python
Python 2.7.15 (default, Aug 17 2018, 22:39:05)
[GCC 4.2.1 Compatible Apple LLVM 9.1.0 (clang-902.0.39.2)] on darwin
Type "help", "copyright", "credits", or "license" for more information.
>>>
```

Đầu ra này chỉ ra rằng Python 2.7.15 hiện là phiên bản mặc định được cài đặt trên máy tính này. Khi đã thấy đầu ra này, hãy nhấn Ctrl-D hoặc nhập exit() để thoát khỏi lời nhắc Python và quay lại lời nhắc đầu cuối.

Để kiểm tra xem đã cài đặt Python 3 chưa, hãy nhập lệnh python3. Nếu nhận được thông báo lỗi, nghĩa là ta không có bất kỳ phiên bản Python 3 đã được cài đặt. Ngược lại, chúng ta sẽ thấy có thông báo python được chạy ở phiên đầu cuối.

Lưu ý rằng bất cứ khi nào ta thấy lệnh python trong bài giảng này, ta cần sử dụng lệnh python3 để đảm bảo rằng ta đang sử dụng Python 3, không phải Python 2; chúng khác nhau đáng kể đến mức ta sẽ gặp sự cố khi cố gắng chạy mã trong cuốn sách này bằng Python 2.

Cài đặt phiên bản mới nhất của Python

Chúng ta có thể cài đặt trình cài đặt Python cho hệ thống của mình tại <https://python.org/>. Di chuột qua liên kết Tải xuống và ta sẽ thấy nút tải xuống phiên bản Python mới nhất. Nhấp vào nút, nút này sẽ tự động bắt đầu tải xuống đúng trình

cài đặt cho hệ thống. Sau khi tải xuống, hãy chạy trình cài đặt. Khi hoàn tất, hãy nhập thông tin sau tại lời nhắc của thiết bị đầu cuối.

```
$ python3 --version
```

Python 3.7.2

Ta sẽ thấy đầu ra tương tự như trên, trong trường hợp đó, ta đã sẵn sàng dùng thử Python. Bất cứ khi nào nhìn thấy lệnh python, hãy đảm bảo rằng ta sử dụng python3.

Chạy Python ở Terminal

Bây giờ ta có thể thử chạy các đoạn mã Python bằng cách mở một thiết bị đầu cuối và nhập python3. Nhập dòng sau vào phiên đầu cuối:

```
>>> print("Hello Python interpreter!")  
Hello Python interpreter!  
>>>
```

Thông điệp sẽ được in trực tiếp trong cửa sổ đầu cuối hiện tại. Cần nhớ rằng ta có thể đóng trình thông dịch Python bằng cách nhấn ctrl-D hoặc bằng cách nhập lệnh exit().

Cài đặt Sublime Text

Để cài đặt trình chỉnh sửa Sublime Text, cần tải xuống trình cài đặt tại <https://sublimetext.com/>. Nhập vào liên kết Tải xuống và tìm trình cài đặt cho hệ điều hành Mac. Sau khi tải xuống trình cài đặt, hãy mở nó và sau đó kéo Sublime Biểu tượng văn bản vào thư mục Application của máy.

1.2.2.3. Python trên Linux

Hệ thống Linux được thiết kế để lập trình, vì vậy Python đã được cài đặt trên hầu hết các máy tính Linux. Những người viết và bảo trì Linux mong đợi người dùng có thể tự lập trình tại một thời điểm nào đó và khuyến khích người dùng làm như vậy. Vì lý do này, có rất ít cái cài đặt và chỉ có một số cài đặt cần thay đổi để bắt đầu lập trình.

Kiểm tra phiên bản của Python

Mở cửa sổ đầu cuối bằng cách chạy ứng dụng Terminal trên hệ thống (trong Ubuntu, ta có thể nhấn ctrl-alt-T). Để biết phiên bản Python nào được cài đặt, hãy nhập python3 với chữ p viết thường. Khi Python được cài đặt, lệnh này sẽ khởi động trình thông dịch Python.

Ta sẽ thấy đầu ra cho biết phiên bản Python nào được cài đặt và dấu nháy >>> nơi ta có thể bắt đầu nhập các lệnh Python, như sau:

```
$ python3
Python 3.7.2 (default, Dec 27 2018, 04:01:51)
[GCC 7.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Kết quả này cho biết Python 3.7.2 hiện là phiên bản Python mặc định được cài đặt trên máy tính này. Khi thấy đầu ra này, hãy nhấn cTrL-D hoặc nhập exit() để thoát khỏi lời nháy Python và quay lại lời nháy phiên đầu cuối. Bất cứ khi nào ta thấy lệnh python trong bài giảng này, hãy nhập python3 thay thế.

Chúng ta sẽ cần Python 3.6 trở lên để chạy mã trong bài giảng này. Nếu phiên bản Python được cài đặt trên hệ thống cũ hơn Python 3.6, hãy tham khảo Phụ lục A để cài đặt phiên bản mới nhất.

Chạy Python trên Terminal

Chúng ta có thể thử chạy các đoạn mã Python bằng cách mở một phiên đầu cuối và nhập python3, như đã làm khi kiểm tra phiên bản python. Thực hiện lại điều này và khi đang chạy Python, hãy nhập dòng sau vào phiên đầu cuối:

```
>>> print("Hello Python interpreter!")
Hello Python interpreter!
>>>
```

Thông điệp sẽ được in trực tiếp trong cửa sổ đầu cuối hiện tại. Cần nhớ rằng ta có thể đóng trình thông dịch Python bằng cách nhấn cTrL-D hoặc bằng cách nhập lệnh exit().

Cài đặt Sublime Text

Trên Linux, ta có thể cài đặt Sublime Text từ Trung tâm Phần mềm Ubuntu. Nhập vào biểu tượng Phần mềm Ubuntu trong menu và tìm kiếm Sublime Text. Bấm để cài đặt nó, sau đó khởi chạy nó.

1.3. Viết chương trình đầu tiên ‘Helloworld.py’

Có thể chạy trình thông dịch của Python trong cửa sổ dòng lệnh, cho phép thử các bit mã Python mà không cần phải lưu và chạy toàn bộ chương trình.

```
>>> print("Hello Python interpreter!")
```

Hello Python interpreter!

Dấu nhắc >>> chỉ ra rằng nên sử dụng cửa sổ đầu cuối và văn bản in đậm là mã cần nhập vào và sau đó thực thi bằng cách nhấn enter. Hầu hết các ví dụ trong bài giảng là nhỏ, khép kín các chương trình mà ta sẽ chạy từ trình soạn thảo văn bản thay vì thiết bị đầu cuối, lý do là vì ta sẽ viết hầu hết mã của mình trong trình soạn thảo văn bản.

Với phiên bản Python và Sublime Text gần đây được cài đặt, ta gần như sẵn sàng chạy chương trình Python đầu tiên được viết bằng trình soạn thảo văn bản. Nhưng trước khi làm như vậy, cần đảm bảo Sublime Text được thiết lập để sử dụng đúng phiên bản Python trên hệ thống. Sau đó, ta sẽ viết Hello World! Lập trình và chạy nó.

1.3.1. Chạy hello_world.py

Trước khi viết chương trình đầu tiên, cần tạo một thư mục có tên là python_work ở đâu đó trên hệ thống cho các dự án. Tốt nhất là sử dụng các chữ cái viết thường và dấu gạch dưới cho khoảng trắng trong tên thư mục và tệp, bởi vì Python sử dụng những quy ước đặt tên.

Mở Sublime Text và lưu tệp hello_world.py trong thư mục python_work. Phần mở rộng .py thông báo với Sulime Text là mã nguồn được viết bằng python và Sublime Text sẽ chạy chương trình và đánh dấu văn bản một cách hữu ích.

Sau khi lưu tệp vào thư mục, gõ dòng code đầu tiên vào phần text editor:

```
print("Hello Python world!")
```

Nếu lệnh python chạy trong hệ thống, có thể chạy chương trình bằng lựa chọn Tool->Build trong menu hoặc nhấn Ctrl +B để chạy chương trình.

Màn hình đầu cuối (terminal) sẽ xuất hiện ở cuối cửa sổ Sublime Text, hiển thị kết quả sau:

```
Hello world
[Finished in 0.1s]
```

Nếu không thấy kết quả này, có thể đã xảy ra lỗi trong chương trình. Ví dụ, kiểm tra các ký tự đã gõ, có ký tự nào gõ chữ Hoa không? Hoặc có thể quên dấu ngoặc đơn, hoặc dấu nháy. Các ngôn ngữ lập trình mong đợi cú pháp rất cụ thể và

nếu không đáp ứng điều đó, chương trình sẽ gặp lỗi. Nếu không thể chạy chương trình, xem các gợi ý trong phần tiếp theo.

1.3.2. Xử lý lỗi

Nếu không thể chạy hello_world.py, đây là một số biện pháp khắc phục mà ta có thể thử. Đây cũng là các giải pháp chung tốt cho bất kỳ vấn đề lập trình nào:

Khi một chương trình chứa một lỗi rõ ràng, Python sẽ hiển thị một traceback, là một báo cáo lỗi. Python xem qua tệp và cố gắng xác định vấn đề. Kiểm tra phần traceback; nó có thể cung cấp một manh mối về vấn đề đang ngăn chương trình chạy.

Cần rời khỏi máy tính, nghỉ một chút rồi thử lại. Hãy nhớ rằng cú pháp rất quan trọng trong lập trình, vì vậy ngay cả thiếu dấu hai chấm, dấu ngoặc kép không khớp hoặc dấu ngoặc đơn không khớp có thể ngăn chương trình chạy bình thường. Đọc lại các phần liên quan đến chương trình trong các chương và nhìn lại code cũng như nhìn lại lỗi.

Bắt đầu lại! Chúng ta không cần gỡ cài đặt của phần mềm nào, nhưng chúng ta có thể xóa tệp hello_world.py và tạo lại nó từ đầu.

Yêu cầu người khác làm theo các bước trong chương này, trên máy tính của mình hoặc một máy tính khác và xem họ làm cẩn thận những gì. Ta có thể đã bỏ lỡ một bước nhỏ mà người khác tình cờ bắt được.

Tìm ai đó biết về Python và nhờ họ giúp cài đặt Python và trình soạn thảo.

Hướng dẫn thiết lập trong chương này cũng có sẵn thông qua trang web đồng hành của cuốn sách tại <https://nostarch.com/pythoncrashcourse2e/>. Phiên bản trực tuyến của những hướng dẫn này có thể phù hợp hơn bởi vì ta có thể cắt và dán mã một cách đơn giản.

Hỏi cộng đồng online. Do ngôn ngữ Python rất phổ biến nên có rất nhiều cộng đồng với số đông thành viên có thể sẵn sàng hỗ trợ các lỗi gặp phải của các thành viên khác. Không cần lo lắng khi các câu hỏi sẽ làm phiền các lập trình viên có kinh nghiệm. Mỗi lập trình viên đều đã bị tắc tại một số điểm và hầu hết các lập trình viên rất vui giúp ta thiết lập hệ thống một cách chính xác. Miễn là ta có thể nói rõ ràng những gì đang cố gắng làm, những gì ta đã thử và kết quả đã đang nhận được, rất có

thể ai đó sẽ có thể giúp ta gỡ lỗi. Cộng đồng của Python đông đảo và chào đón những người mới.

Python sẽ chạy tốt trên bất kỳ máy tính hiện đại nào. Sự cố thiết lập ban đầu có thể khiến ta khó chịu, nhưng chúng rất cần được phân loại.

Khi thấy được `hello_world.py` đang chạy, ta có thể bắt đầu học Python và bắt đầu các công việc lập trình.

1.3.3. Chạy chương trình Python từ Terminal

Hầu hết các chương trình viết trong trình soạn thảo văn bản sẽ chạy trực tiếp từ trình soạn thảo, nhưng cũng có lúc hữu ích khi chạy chương trình từ cửa sổ terminal. Ví dụ: ta có thể muốn chạy một chương trình hiện có mà không cần mở nó để chỉnh sửa.

Ta có thể thực hiện việc này trên bất kỳ hệ thống nào có cài đặt Python nếu biết cách truy cập vào thư mục lưu trữ chương trình. Để thực hiện việc này, lưu ý cần nhớ vị trí lưu tệp mã nguồn `python`.

Trên Window:

Có thể sử dụng lệnh terminal `cd`, để thay đổi thư mục, để điều hướng thông qua hệ thống tệp trong một cửa sổ lệnh. Lệnh `dir`, hiển thị tất cả các tệp tồn tại trong thư mục hiện tại.

Mở một cửa sổ dòng lệnh mới và nhập các lệnh sau để chạy tệp `hello_world.py`.

```
C:\> cd Desktop\python_work  
C:\Desktop\python_work> dir  
hello_world.py  
C:\Desktop\python_work> python hello_world.py  
Hello Python world!
```

Dòng thứ nhất dùng lệnh `cd` để chuyển về thư mục chứa tệp mã nguồn

Dòng thứ hai dùng lệnh `dir` để kiểm tra xem thư mục có chứa tệp mã nguồn hay không, hệ thống sẽ trả về tên tệp `.py`

Dòng thứ ba sử dụng lệnh `python hello_world.py` để chạy chương trình

1.3.4. Chạy trên macOS and Linux

Chạy chương trình Python từ một phiên đầu cuối cũng giống như trên Linux và macOS. Ta có thể sử dụng lệnh đầu cuối `cd`, để thay đổi thư mục, để điều hướng

qua hệ thống tệp trong một phiên đầu cuối. Lệnh `ls`, cho danh sách, hiển thị tất cả các tệp không ẩn tồn tại trong thư mục hiện tại.

Mở một cửa sổ dòng lệnh mới và nhập các lệnh sau để chạy `hello_world.py`:

```
~$ cd Desktop/python_work/  
~/Desktop/python_work$ ls  
hello_world.py  
~/Desktop/python_work$ python hello_world.py  
Hello Python world!
```

Dòng đầu tiên ta sử dụng lệnh `cd` để điều hướng đến thư mục `python_work`, nằm trong thư mục `Desktop`. Tiếp theo, ta sử dụng lệnh `ls` để đảm bảo `hello_world.py` nằm trong thư mục này. Sau đó, ta chạy tệp bằng cách sử dụng lệnh `python hello_world.py`.

Thao tác đơn giản vì ta chỉ cần sử dụng lệnh `python` (hoặc `python3`) để chạy các chương trình Python.

Kết chương

Trong chương này, ta đã học về Python nói chung và đã cài đặt Python trên hệ thống để bắt đầu lập trình. Ta cũng đã cài đặt phần mềm văn bản để gõ mã nguồn Python. Chúng ta đã chạy các đoạn mã Python trong cửa sổ dòng lệnh và đã chạy chương trình đầu tiên là `hello_world.py`. Chúng ta cũng biết một chút về xử lý sự cố và gỡ lỗi chương trình. Chương tiếp theo sẽ tìm hiểu về các loại dữ liệu khác nhau mà có thể hoạt động trong các chương trình Python và cách sử dụng các biến.

CHƯƠNG 2. BIẾN VÀ NHỮNG KIÊU DỮ LIỆU ĐƠN GIẢN

Trong chương này, chúng ta sẽ tìm hiểu về các loại dữ liệu khác nhau mà có thể sử dụng trong các chương trình Python. Chúng ta cũng sẽ học cách sử dụng các biến để biểu diễn dữ liệu trong các chương trình.

Điều gì thực sự xảy ra khi chạy file hello_world.py?

Chúng ta xem xét kỹ hơn những gì Python làm khi thực hiện chạy hello_world.py. Hóa ra Python làm rất nhiều việc khi chỉ chạy một chương trình đơn giản.

```
print("Hello Python world!")
```

Khi chạy đoạn code trên, python sẽ in ra:

```
Hello Python world!
```

Khi chạy hello_world.py, .py cho biết tệp chạy là một chương trình Python. Trình soạn thảo sẽ chạy tệp thông qua một bộ thông dịch, bộ biên dịch này sẽ đọc toàn bộ chương trình Python và xác định từng từ trong chương trình là gì. Ví dụ: khi bộ thông dịch thấy chữ in theo sau là dấu ngoặc đơn, nó in ra màn hình bất cứ điều gì bên trong dấu ngoặc đơn. Khi viết chương trình, trình chỉnh sửa đánh dấu các phần khác nhau của chương trình theo những cách khác nhau. Ví dụ, nó nhận ra rằng print() là tên của một hàm và hiển thị từ đó bằng một màu. Nó nhận ra rằng "Hello world Python!" không phải là mã Python và hiển thị cụm từ đó trong màu khác. Tính năng này được gọi là tô sáng cú pháp và khá hữu ích khi ta bắt đầu viết các chương trình.

2.1. Biến

Thử sử dụng biến trong tệp hello_world.py. Thêm một dòng ở đầu tệp và sửa dòng thứ hai:

```
message = "Hello Python world!"  
print(message)
```

Chạy chương trình này để xem điều gì sẽ xảy ra. Ta sẽ thấy cùng một đầu ra như trước đây:

```
Hello Python world!
```

Chúng ta đã thêm biến được gọi tên là *message*. Mỗi biến đều được gắn với một giá trị là thông tin của biến đó. Trong trường hợp này, giá trị của biến là "Hello Python world!" và dạng văn bản.

Thêm một biến làm cho trình thông dịch Python hoạt động nhiều hơn một chút. Khi nó xử lý dòng đầu tiên, nó liên kết biến *message* với văn bản "Hello Python world!". Khi thông dịch tới dòng thứ hai, nó in ra giá trị gắn với biến *message* ra màn hình.

Mở rộng chương trình với việc chỉnh sửa *hello_world.py*, để in ra thông điệp thứ hai. Thêm dòng trống vào tệp *hello_world.py* và thêm hai dòng code mới:

```
message = "Hello Python world!"
print(message)
message = "Hello Python Crash Course world!"
print(message)
```

Chạy lại tệp, ta sẽ thấy đầu ra ở màn hình:

```
Hello Python world!
Hello Python Crash Course world!
```

Chúng có thể thay đổi giá trị của một biến trong chương trình bất kỳ lúc nào, và Python sẽ luôn theo dõi giá trị hiện tại của nó.

2.1.1. Định danh và sử dụng các biến

Khi chúng ta đang sử dụng các biến trong Python, ta cần tuân thủ một số quy tắc và các hướng dẫn. Việc phá vỡ một số quy tắc này sẽ gây ra lỗi; các nguyên tắc khác chỉ giúp ta viết mã dễ đọc và dễ hiểu hơn. Cần ghi nhớ các quy tắc sau về biến:

- Tên biến chỉ có thể chứa các chữ cái, số và dấu gạch dưới. Chúng có thể bắt đầu bằng một chữ cái hoặc một dấu gạch dưới, nhưng không phải bằng một số. Ví dụ: có thể gọi một biến *message_1* nhưng không thể gọi *1_message*.
- Không được phép sử dụng dấu cách trong tên biến, nhưng có thể sử dụng dấu gạch dưới để tách các từ trong tên biến. Ví dụ: *welcome_message* hoạt động, nhưng *welcome message* sẽ gây ra lỗi.

- Tránh sử dụng các từ khóa Python và tên hàm làm tên biến; nghĩa là, không sử dụng các từ mà Python đã dành riêng cho một mục đích lập trình cụ thể, chẳng hạn như từ *print*.
- Tên biến phải ngắn nhưng mang tính mô tả. Ví dụ, *name* là tốt hơn *n*, *student_name* tốt hơn *s_n* và *name_length* tốt hơn *length_of_persons_name*.
- Hãy cẩn thận khi sử dụng chữ cái viết thường *l* và chữ cái viết hoa *O* vì chúng có thể bị nhầm lẫn với số *1* và số *0*

Có thể mất một số thời gian thực hành để học cách tạo tên biến tốt, đặc biệt là khi các chương trình trở nên thú vị và phức tạp hơn. Khi chúng ta viết nhiều chương trình hơn và bắt đầu đọc qua code của người khác, ta sẽ làm tốt hơn việc nghĩ ra những cái tên có ý nghĩa.

2.1.2. Tránh đặt tên lỗi khi sử dụng các biến

Mọi lập trình viên đều mắc lỗi và hầu hết đều mắc lỗi hàng ngày. Mặc dù các lập trình viên giỏi có thể tạo ra lỗi, nhưng họ cũng biết cách phản hồi những lỗi đó một cách hiệu quả. Hãy xem một lỗi mà chúng ta có thể mắc phải thực hiện sớm và học cách điều chỉnh nó.

Chúng ta sẽ viết một số mã có ý tạo ra lỗi. Viết mã sau, bao gồm thông báo từ sai chính tả được in đậm:

```
message = "Hello Python Crash Course reader!"  
print(mesage)
```

Khi một lỗi xảy ra trong chương trình, trình thông dịch Python sẽ cố gắng nhất để giúp ta tìm ra vấn đề ở đâu. Trình thông dịch cung cấp một dấu vết khi một chương trình không thể chạy thành công.

Dấu vết (traceback) là một bản ghi về nơi trình thông dịch gặp sự cố khi cố gắng thực thi code đã viết. Dưới đây là một ví dụ về dấu vết mà Python cung cấp sau khi ta vô tình viết sai chính tả tên của một biến:

```
Traceback (most recent call last):  
① File "hello_world.py", line 2, in <module>  
② print(mesage)  
③ NameError: name 'mesage' is not defined
```

Thông báo đầu ra tại ① cho biết lỗi xảy ra tại dòng thứ hai của đoạn code. Trình thông dịch hiển thị dòng ② để giúp chúng ta phát hiện lỗi nhanh chóng và cho chúng ta biết nó đã tìm thấy loại lỗi nào ③. Trong trường hợp này, python tìm thấy một lỗi tên và báo cáo rằng biến đang được in, *mesage*, chưa được định nghĩa. Python không thể xác định tên biến được cung cấp. Một lỗi về tên thường có nghĩa là chúng ta đã quên đặt giá trị của một biến trước khi sử dụng nó hoặc chúng ta đã mắc lỗi chính tả khi nhập tên của biến.

Tất nhiên, trong ví dụ này, chúng ta đã bỏ qua ký tự s trong tên biến *message* ở dòng thứ hai. Trình thông dịch Python không kiểm tra chính tả các biến nhưng nó đảm bảo rằng các tên biến được viết một cách nhất quán. Ví dụ, hãy xem điều gì sẽ xảy ra khi chúng ta viết sai message trong cùng một đoạn code:

```
mesage = "Hello Python Crash Course reader!"  
print(mesage)
```

Trong trường hợp này, Python chạy bình thường

```
Hello Python Crash Course reader!
```

Ngôn ngữ lập trình rất nghiêm ngặt, nhưng chúng không phân biệt đúng sai chính tả. Do đó, chúng ta không cần phải xem xét các quy tắc chính tả và ngữ pháp tiếng Anh khi ta đang cố gắng tạo tên biến và viết mã.

2.1.3. Biến là các nhãn

Các biến thường được mô tả như các hộp có thể lưu trữ các giá trị. Ý tưởng này có thể hữu ích khi mới sử dụng một vài biến, nhưng đó không phải là cách chính xác để mô tả cách các biến được biểu diễn bên trong bằng Python.

Cách tốt hơn là nên coi các biến dưới dạng nhãn mà chúng ta có thể gán cho các giá trị. Ta có thể cũng nói rằng một biến tham chiếu đến một giá trị nhất định.

2.2. Chuỗi (Strings)

Hầu hết các chương trình xác định và thu thập một số loại dữ liệu, sau đó thực hiện thao tác với dữ liệu, và giúp phân loại các loại dữ liệu khác nhau.

Đầu tiên loại dữ liệu mà chúng ta sẽ xem xét là chuỗi. Thoạt nhìn, các chuỗi khá đơn giản, nhưng ta có thể sử dụng chúng theo nhiều cách khác nhau.

Một chuỗi là một chuỗi các ký tự. Mọi thứ bên trong dấu ngoặc kép đều được xem xét một chuỗi với Python và ta có thể sử dụng dấu ngoặc đơn hoặc dấu ngoặc kép xung quanh chuỗi như sau:

```
"This is a string."  
'This is also a string.'
```

Sự linh hoạt này cho phép ta sử dụng dấu nháy đơn hoặc nháy kép trong chuỗi.

```
'I told my friend, "Python is my favorite language!"'  
"The language 'Python' is named after Monty Python, not the snake."  
"One of Python's strengths is its diverse and supportive community."
```

Một vài cách biến đổi chuỗi

2.2.1. Sử dụng chữ hoa trong chuỗi với phương thức

Một trong những tác vụ đơn giản nhất có thể làm với chuỗi là thay đổi chữ hoa chữ thường của từ trong một chuỗi. Hãy xem đoạn mã sau và cố gắng xác định những gì đang xảy ra:

```
name = "ada lovelace"  
print(name.title())
```

Lưu thành tệp name.py và chạy file, nhìn thấy màn hình

Ada Lovelace

Trong trường hợp này, biến name trả vào chuỗi chữ thường "ada lovelace". Phương thức title() xuất hiện sau biến name trong hàm print() được gọi. Một phương thức là một hành động của Python có thể thực hiện trên dữ liệu. Dấu chấm(.) sau biến name trong name.title() cho Python biết việc thực hiện phương thức title() trên biến name. Tất cả các phương thức được kèm theo cặp ngoặc đơn () bởi vì thông thường phương thức cần các thông tin bổ trợ để thực hiện. Các thông tin bổ trợ được cho vào bên trong ngoặc đơn. Phương thức title() không cần thêm bất kỳ thông tin bổ sung nào, do đó trong ngoặc đơn bỏ trống.

Phương thức title() thay đổi chữ thường thành chữ hoa như tiêu đề, do đó mỗi ký tự đầu chữ sẽ thành chữ hoa. Điều này có ích bởi ta luôn nghĩ là cái tên là một mẫu thông tin. Ví dụ ta muốn biết chương trình nhận biết các giá trị đầu vào như Ada, ADA và ada là trùng với nhau và được biểu diễn là Ada.

Một số phương thức hữu ích khác có sẵn để xử lý trường hợp chữ hoa, chữ thường. Ví dụ: ta có thể thay đổi một chuỗi thành tất cả chữ hoa hoặc tất cả chữ thường những chữ cái như sau:

```
name = "Ada Lovelace"  
print(name.upper())  
print(name.lower())
```

Kết quả:

```
ADA LOVELACE  
ada lovelace
```

Phương thức lower() đặc biệt hữu ích để lưu trữ dữ liệu. Nhiều lúc ta sẽ không muốn tin tưởng vào cách viết hoa mà người dùng cung cấp, vì vậy ta sẽ chuyển đổi chuỗi thành chữ thường trước khi lưu trữ chúng. Sau đó, khi muốn hiển thị thông tin, ta sẽ sử dụng dạng chữ hoa/thường có ý nghĩa nhất cho mỗi chuỗi.

2.2.2. Sử dụng biến trong chuỗi

Trong một số trường hợp, chúng ta sẽ muốn sử dụng giá trị của một biến bên trong một chuỗi. Ví dụ, ta có thể muốn hai biến đại diện cho tên đầu tiên và họ tên tương ứng, và sau đó muốn kết hợp các giá trị đó để hiển thị tên đầy đủ của ai đó:

```
first_name = "ada"  
last_name = "lovelace"  
❶ full_name = f"{first_name} {last_name}"  
print(full_name)
```

Để chèn giá trị của một biến vào một chuỗi, hãy đặt ngay ký tự f trước dấu ngoặc kép mở đầu ❶. Đặt dấu ngoặc nhọn xung quanh tên hoặc các tên của bất kỳ biến nào muốn sử dụng bên trong chuỗi. Python sẽ thay thế từng biến với giá trị của nó khi chuỗi được hiển thị.

Các chuỗi này được gọi là f-strings. F là cho định dạng, vì Python định dạng chuỗi bằng cách thay thế tên của bất kỳ biến nào trong dấu ngoặc nhọn bằng giá trị. Đầu ra từ mã trước đó là:

```
ada lovelace
```

Chúng ta có thể làm được nhiều điều với f-string. Ví dụ: ta có thể sử dụng f-string để tạo thông điệp hoàn chỉnh bằng cách sử dụng thông tin được liên kết với một biến, như được hiển thị ở đây:

```
first_name = "ada"  
last_name = "lovelace"
```

```
full_name = f"{first_name} {last_name}"
① print(f"Hello, {full_name.title()}!")
```

Tên đầy đủ được sử dụng trong câu chào người dùng ① và phương thức title() thay đổi tên thành trường tiêu đề. Mã này trả về một đơn giản nhưng lời chào được định dạng dễ nhìn:

```
Hello, Ada Lovelace!
```

Ta cũng có thể sử dụng f-string để soạn thông điệp, sau đó gán toàn bộ thông điệp cho một biến:

```
first_name = "ada"
last_name = "lovelace"
full_name = f"{first_name} {last_name}"
message = f"Hello, {full_name.title()}!"
print(message)
```

Kết quả giống như ở trên, đoạn mã hiển thị thông điệp chào người dùng nhưng được gán vào biến message và sau đó được truyền vào lệnh print().

Ghi chú: F-strings lần đầu tiên được giới thiệu trong Python 3.6. Nếu đang sử dụng Python 3.5 trở xuống, ta sẽ cần sử dụng phương thức format() thay vì cú pháp f này. Để sử dụng format(), cần liệt kê các biến muốn sử dụng trong chuỗi bên trong định dạng sau dấu ngoặc đơn. Mỗi biến được tham chiếu bởi một tập hợp các dấu ngoặc nhọn; các dấu ngoặc nhọn sẽ được điền bởi các giá trị được liệt kê trong ngoặc đơn theo thứ tự được cung cấp:

```
full_name = "{} {}".format(first_name, last_name)
```

2.2.3. Thêm khoảng trắng vào chuỗi với Tab hoặc dòng mới

Trong lập trình, khoảng trắng để cập đến bất kỳ ký tự không in nào, chẳng hạn như dấu cách, tab và ký hiệu cuối dòng. Ta có thể sử dụng khoảng trắng để sắp xếp đầu ra để người dùng dễ đọc hơn.

Để thêm Tab vào chuỗi, sử dụng tổ hợp ký tự: \t như dưới đây:

```
>>> print("Python")
Python
>>> print("\tPython")
Python
```

Để thêm một dòng mới trong một chuỗi, hãy sử dụng tổ hợp ký tự \n:

```
>>> print("Languages:\nPython\nC\nJavaScript")
Languages:
```

Python

C

JavaScript

2.2.4. Bỏ khoảng trắng

Khoảng trắng thừa có thể gây nhầm lẫn trong các chương trình. Với các lập trình viên 'python' và 'python' trông khá giống nhau. Nhưng đối với một chương trình, chúng là hai chuỗi khác nhau. Khi so sánh hai chuỗi, thường sẽ cần loại bỏ các khoảng trắng, ví dụ trong trường hợp kiểm tra tên người dùng khi họ đăng nhập vào trang web. Khoảng trắng thừa có thể gây nhầm lẫn, do cần loại bỏ trước khi thực hiện các tác vụ. Python giúp loại bỏ dễ dàng khoảng trắng thừa từ dữ liệu người dùng nhập vào.

```
>>> favorite_language = 'python '
>>> favorite_language
'python '
>>> favorite_language.rstrip()
'python'
>>> favorite_language
'python '
```

Chuỗi ở dòng đầu tiên được gán vào biến favorite_language có khoảng trắng thừa nên khi in ra biến, chúng ta thấy rõ kết quả in ra có khoảng trắng thừa đó. Phương thức rstrip() của chuỗi được sử dụng để xóa đi khoảng trắng ở bên phải và kết quả in ra chữ python không còn khoảng trắng. Tuy nhiên, phương thức này chỉ là kết quả tạm thời. Nếu chúng ta in ra lại biến favorite_language ở câu lệnh tiếp theo, khoảng trắng lại xuất hiện.

Để xóa khoảng trắng vĩnh viễn, ta thực hiện phép gán:

```
>>> favorite_language = 'python '
>>> favorite_language = favorite_language.rstrip()
>>> favorite_language
'python'
```

Để xóa khoảng trắng khỏi chuỗi, ta xóa khoảng trắng từ phía bên phải của chuỗi và sau đó liên kết giá trị mới này với biến ban đầu. Việc thay đổi giá trị của một biến được thực hiện thường xuyên trong lập trình. Đây là cách giá trị của một biến có thể được cập nhật khi chương trình được thực thi hoặc để phản hồi lại đầu vào của người dùng.

Ta cũng có thể loại bỏ khoảng trắng từ phía bên trái của một chuỗi bằng cách sử dụng phương thức `lstrip()` hoặc từ cả hai bên cùng một lúc bằng `strip()`:

```
>>> favorite_language = ' python '
>>> favorite_language.rstrip()
' python'
>>> favorite_language.lstrip()
'python '
>>> favorite_language.strip()
'python'
```

Ví dụ trên thể hiện các `python` xóa các khoảng trắng ở bên phải, bên trái, và cả hai bên sử dụng tương ứng các phương thức `rstrip()`, `lstrip()` và `strip()`. Thực nghiệm với các chức năng bỏ khoảng trắng này có thể giúp ta làm quen với việc thao tác với chuỗi. Trong thế giới thực, các hàm `strip()` này được sử dụng thường xuyên nhất để dọn dẹp thông tin đầu vào của người dùng trước khi nó được lưu trữ trong một chương trình.

Với yêu cầu xóa khoảng trắng giữa các từ trong một câu thì cần phải sử dụng kỹ thuật đối với danh sách (list). Nội dung này sẽ được trình bày trong chương số 4.

2.4.5. Tránh lỗi cú pháp với String

Một loại lỗi có thể gặp thường xuyên là lỗi cú pháp. Lỗi cú pháp xảy ra khi Python không nhận ra một phần trong chương trình là mã Python hợp lệ. Ví dụ: nếu ta sử dụng dấu nháy đơn trong một dấu ngoặc kép sẽ tạo ra một lỗi.

Điều này xảy ra bởi vì Python diễn giải mọi thứ giữa dấu nháy đơn đầu tiên và dấu nháy đơn dưới dạng chuỗi. Sau đó, nó cố gắng giải thích phần còn lại của văn bản dưới dạng mã Python, điều này gây ra lỗi.

Dưới đây là cách sử dụng chính xác dấu nháy kép và dấu nháy đơn. Lưu chương trình này dưới dạng dấu nháy đơn và sau đó chạy nó:

```
message = "One of Python's strengths is its diverse community."
print(message)
```

Dấu nháy đơn xuất hiện bên trong một tập hợp các dấu ngoặc kép, vì vậy thông dịch của Python không gặp khó khăn khi đọc chuỗi một cách chính xác:

```
One of Python's strengths is its diverse community.
```

Tuy nhiên, nếu ta sử dụng dấu nháy đơn, Python không thể xác định vị trí chuỗi nên kết thúc:

```
message = 'One of Python's strengths is its diverse community.'  
print(message)
```

Python trả ra:

```
File "apostrophe.py", line 1  
message = 'One of Python's strengths is its diverse community.'  
SyntaxError: invalid syntax
```

Trong đầu ra, có thể thấy rằng lỗi xảy ra ngay sau trích dẫn đơn thứ hai. Lỗi cú pháp này chỉ ra rằng trình thông dịch không nhận ra nội dung trong mã là mã Python hợp lệ. Lỗi có thể đến từ nhiều nguồn khác nhau.

2.3. Số (Numbers)

Các con số được sử dụng khá thường xuyên trong lập trình để lưu điểm số trong trò chơi, đại diện cho dữ liệu trong trực quan hóa, lưu trữ thông tin trong các ứng dụng web và các ứng dụng khác. Python xử lý các số theo các cách khác nhau, dựa trên việc các số được sử dụng như thế nào.

2.3.1. Số nguyên - Integers

Ta có thể cộng (+), trừ (-), nhân (*) và chia (/) số nguyên trong Python.

```
>>> 2 + 3  
5  
>>> 3 - 2  
1  
>>> 2 * 3  
6  
>>> 3 / 2  
1.5
```

Trong một phiên đầu cuối, Python trả về kết quả của toán tử. Python sử dụng hai ký hiệu nhân để biểu diễn số mũ:

```
>>> 3 ** 2  
9  
>>> 3 ** 3  
27  
>>> 10 ** 6  
1000000
```

Python cũng hỗ trợ thứ tự các phép toán, vì vậy ta có thể sử dụng nhiều các phép toán trong một biểu thức. Ta cũng có thể sử dụng dấu ngoặc đơn để sửa đổi thứ

tự của các phép toán để Python có thể đánh giá biểu thức theo thứ tự được chỉ định.

Ví dụ:

```
>>> 2 + 3*4  
14  
>>> (2 + 3) * 4  
20
```

Khoảng cách trong các ví dụ này không ảnh hưởng đến cách Python đánh giá các biểu thức; nó chỉ đơn giản giúp phát hiện nhanh hơn các hoạt động được ưu tiên khi ta đọc mã nguồn.

2.3.2. Số thực - Floats

Python gọi bất kỳ số nào có dấu thập phân là số thực (float). Thuật ngữ này được sử dụng trong hầu hết các ngôn ngữ lập trình và nó đề cập đến thực tế là một dấu phân tách số có thể xuất hiện ở bất kỳ vị trí nào trong một số. Mọi ngôn ngữ lập trình phải được thiết kế cẩn thận để quản lý đúng các số thập phân vì vậy các con số hoạt động thích hợp bất kể dấu thập phân xuất hiện ở đâu.

Phần lớn có thể sử dụng số thập phân mà không cần lo lắng về cách xử lý. Chỉ cần nhập các số cần sử dụng và Python sẽ xử lý số đúng như mong đợi:

```
>>> 0.1 + 0.1  
0.2  
>>> 0.2 + 0.2  
0.4  
>>> 2 * 0.1  
0.2  
>>> 2 * 0.2  
0.4
```

Nhưng hãy lưu ý rằng đôi khi ta có thể nhận được một số chữ số thập phân tùy ý trong câu trả lời:

```
>>> 0.2 + 0.1  
0.3000000000000004  
>>> 3 * 0.1  
0.3000000000000004
```

Điều này xảy ra ở tất cả các ngôn ngữ và ít được quan tâm. Python cố gắng và một cách để thể hiện kết quả càng chính xác càng tốt.

2.3.3. Số nguyên và số thực

Khi chia hai số bất kỳ, ngay cả khi chúng là số nguyên dẫn đến kết quả đều sẽ là số thực.

```
>>> 4/2
```

```
2.0
```

Nếu kết hợp số nguyên và số thực trong một phép toán, kết quả sẽ nhận được là số thực:

```
>>> 1 + 2.0
```

```
3.0
```

```
>>> 2 * 3.0
```

```
6.0
```

```
>>> 3.0 ** 2
```

```
9.0
```

2.3.4. Các gạch dưới trong số

Khi viết các số dài, ta có thể nhóm các chữ số bằng dấu gạch dưới để làm cho các số lớn dễ đọc hơn:

```
>>> universe_age = 14_000_000_000
```

Khi in ra số dùng gạch dưới, Python chỉ in ra các số

```
>>> print(universe_age)
```

```
14000000000
```

Python bỏ qua dấu gạch dưới khi lưu trữ các loại giá trị này. Ngay cả khi ta không nhóm các chữ số theo nhóm ba số, giá trị sẽ vẫn không bị ảnh hưởng. Đối với Python, 1000 giống với 1_000, giống với 10_00. Tính năng này hoạt động với số nguyên và số thực, nhưng nó chỉ khả dụng với Python 3.6 và các phiên bản sau đó.

2.3.5. Gán nhiều biến cùng lúc.

Chúng ta có thể gán giá trị cho nhiều biến chỉ bằng một dòng duy nhất. Điều này có thể giúp rút ngắn chương trình và làm cho chúng dễ đọc hơn; ta sẽ sử dụng kỹ thuật này thường xuyên nhất khi khởi tạo một tập hợp số.

Ví dụ: đây là cách khởi tạo các biến x, y và z về 0:

```
>>> x, y, z = 0, 0, 0
```

Ta cần tách các tên biến bằng dấu phẩy và thực hiện tương tự với các giá trị và Python sẽ chỉ định từng giá trị tương ứng với biến theo vị trí. Miễn là số lượng giá trị khớp với số lượng các biến, Python sẽ khớp chúng một cách chính xác

2.3.6. Hằng số - Constants

Một hằng số giống như một biến có giá trị không đổi trong suốt vòng đời của một chương trình. Python không có sẵn các loại hằng số, nhưng các lập trình viên Python sử dụng tất cả các chữ cái viết hoa để chỉ ra một biến nên được coi là không đổi và không bao giờ bị thay đổi:

```
MAX_CONNECTIONS = 5000
```

Khi muốn coi biến là một hằng số trong mã, hay viết hoa tất cả các chữ cái trong tên biến.

2.4. Chú thích (Comments)

Chú thích là một tính năng cực kỳ hữu ích trong hầu hết các ngôn ngữ lập trình. Mọi thứ ta đã viết trong các chương trình cho đến nay đều là mã Python. Khi các chương trình trở nên dài hơn và phức tạp hơn, ta nên thêm ghi chú trong các chương trình mô tả cách tiếp cận tổng thể đối với vấn đề bài toán đang giải quyết

2.4.1. Cách viết các chú thích

Trong Python, dấu # (#) chỉ ra một chú thích. Bất cứ điều gì sau dấu # trong mã bị trình thông dịch Python bỏ qua. Ví dụ:

```
# Say hello to everyone.  
print("Hello Python people!")
```

Python sẽ bỏ qua dòng đầu tiên vì có gặp dấu #, chỉ thực thi dòng thứ hai:

```
Hello Python people!
```

2.4.2. Loại chú thích nên viết

Lý do chính để viết chú thích là để giải thích mã nguồn để làm gì và cách làm cho nó hoạt động. Khi ta đang thực hiện một dự án, ta hiểu cách tất cả các phần kết hợp với nhau. Nhưng khi quay lại một dự án sau một thời gian xa, ta có thể đã quên một số chi tiết. Ta luôn có thể nghiên cứu mã nguồn đó trong một thời gian và thử nghiệm tìm ra cách các phân đoạn được cho là hoạt động, nhưng viết nhận xét tốt có thể giúp chúng ta tiết kiệm thời gian bằng cách tóm tắt cách tiếp cận tổng thể bằng tiếng Anh rõ ràng.

Nếu muốn trở thành một lập trình viên chuyên nghiệp hoặc cộng tác với các lập trình viên khác, ta nên viết chú thích có ý nghĩa. Ngày nay, hầu hết phần mềm được viết một cách hợp tác, cho dù bởi một nhóm nhân viên tại một công ty hoặc một nhóm người làm việc cùng nhau trong một dự án nguồn mở. Các lập trình viên lành nghề mong đợi thấy các nhận xét trong mã, vì vậy tốt nhất là bắt đầu thêm nhận xét mô tả vào chương trình ngay bây giờ. Viết rõ ràng, ngắn gọn nhận xét trong mã của ta là một trong những thói quen có lợi nhất có thể hình thành khi đang là một lập trình viên mới.

Khi xác định có nên viết bình luận hay không, hãy tự đặt câu hỏi xem ta đã phải xem xét một số cách tiếp cận trước khi tìm ra một cách hợp lý để xây dựng mã nguồn; nếu vậy, hãy viết chú thích về giải pháp đó. Sau này xóa các nhận xét bổ sung dễ dàng hơn nhiều so với quay lại và viết chú thích cho một chương trình được chú thích thưa thớt.

Bài tập chương 2

Bài 2-1. Simple Message: Gán thông điệp vào một biến và in ra thông điệp đó

Bài 2-2. Simple Message: Gán thông điệp vào một biến và in thông điệp đó ra. Sau đó thay đổi giá trị của biến thành một thông điệp khác và in thông điệp mới đó ra.

Bài 2-3. Personal Message: sử dụng một biến đại diện cho tên của một người và in thông điệp cho người đó. Thông điệp cần đơn giản như “*Hello Eric, would you like to learn some Python today?*”.

Bài 2-4. Name Cases: Sử dụng một biến để biểu diễn tên của một người, sau đó in tên người đó với chữ thường, chữ hoa, và chữ kiểu tiêu đề (title).

Bài 2-5. Famous Quote: Tìm một trích dẫn từ một người nổi tiếng, sau đó in trích dẫn và tên tác giả của trích dẫn đó. Đầu ra như dưới đây (bao gồm cả ký tự trích dẫn):

Albert Einstein once said, “A person who never made a mistake never tried anything new.”

Bài 2-6. Famous Quote2: Lặp lại ví dụ 2-5 nhưng lần này gán tên người nổi tiếng vào biến *famous_person*. Sau đó, gán thông điệp của người đó vào biến *message*, sau đó in ra thông điệp như trong bài tập 2-5

Bài 2-7. Stripping Names: Sử dụng một biến để đại diện cho tên một người bao gồm một số ký tự trắng tại phần đầu và cuối của tên. Đảm bảo rằng mỗi tổ hợp ký tự “\t” và “\n” được sử dụng ít nhất một lần.

In ra tên một lần để thấy các ký tự trắng quanh tên được hiển thị. Sau đó in tên có sử dụng các phương thức `lstrip()`, `rstrip()`, `strip()`.

Bài 2-8. Number Eight: Viết các phép tính cộng, trừ, nhân, chia các phép toán mà mỗi kết quả cho ra là số 8. Đầu ra có bốn dòng với số 8 xuất hiện một lần trên mỗi dòng.

Bài 2-9. Favorite Number: Sử dụng một biến để đại diện cho một con số yêu thích. Sau đó, sử dụng biến đó, tạo một thông điệp tiết lộ số yêu thích đó. In ra thông điệp đó.

Bài 2-10. Adding Comments: Chọn 2 chương trình đã viết trong các phần trước và thêm ít nhất mỗi ghi chú cho chương trình đó. Nếu không có gì đặc biệt để ghi chú thì ghi tên, và ngày viết chương trình vào đầu mỗi tệp chương trình. Sau đó ghi chú mô tả chương trình thực hiện vấn đề gì.

Bài 2.11. Zen of Python: gõ lệnh `import this` ở cửa sổ terminal và nhìn kết quả hiển thị để thấy được các nguyên lý của Python.

Kết chương

Trong chương này, chúng ta đã học cách làm việc với các biến. Ta đã học cách sử dụng tên biến mô tả và cách giải quyết lỗi tên và lỗi cú pháp khi chúng phát sinh. Ta đã học được chuỗi là gì và cách hiển thị chuỗi bằng cách sử dụng chữ thường, chữ hoa và chữ hoa tiêu đề. Ta cũng đã bắt đầu sử dụng khoảng trắng để sắp xếp đầu ra gọn gàng và ta đã học cách loại bỏ khoảng trắng không cần thiết khỏi các phần khác nhau của một chuỗi.

Chúng ta đã bắt đầu làm việc với số nguyên và số thực, đồng thời học được một số những cách có thể làm việc với dữ liệu số. Ta cũng đã học cách viết các chú thích để làm cho mã dễ đọc hơn cho tavà những người khác.

Trong Chương 3, chúng ta sẽ học cách lưu trữ các bộ thông tin trong cấu trúc dữ liệu được gọi là danh sách. Ta sẽ học cách làm việc trên một danh sách, sử dụng bất kỳ thông tin trong danh sách đó.

CHƯƠNG 3. DANH SÁCH (LIST)

Trong chương này và các chương tiếp theo, ta sẽ học danh sách là gì và làm thế nào để bắt đầu làm việc với các phần tử trong danh sách. Danh sách cho phép lưu trữ tập hợp thông tin ở một nơi, cho dù chỉ có một vài tin mục hoặc hàng triệu tin mục. Danh sách là một trong những tính năng mạnh mẽ nhất của Python đối với người dùng mới, và nó kết hợp nhiều khái niệm quan trọng trong lập trình.

3.1. Định nghĩa Danh sách

Danh sách là tập hợp các mục tin theo một thứ tự đặc biệt. Ta có thể tạo ra một danh sách bao gồm các chữ cái, các số tự nhiên hoặc là tên của tất cả các thành viên trong gia đình. Ta có thể đưa bất cứ thứ gì mong muốn vào danh sách và các tin mục trong danh sách không nhất thiết phải liên quan theo bất kỳ cách cụ thể nào. Bởi vì một danh sách thường chứa nhiều hơn một phần tử, ta nên tạo tên của danh sách ở số nhiều, chẳng hạn như letters, digits hay names.

Trong Python, ngoặc vuông ([]) chỉ định một danh sách và các phần tử trong danh sách được phân tách bởi dấu phẩy (,). Dưới đây là ví dụ danh sách chứa các loại xe đạp khác nhau.

```
bicycles = ['trek', 'cannondale', 'redline', 'specialized']
print(bicycles)
```

Khi yêu cầu Python in ra danh sách, Python sẽ trả về biểu diễn của danh sách, bao gồm cả ngoặc vuông.

```
['trek', 'cannondale', 'redline', 'specialized']
```

Bởi vì đây không phải là kết quả ta muốn người dùng thấy, hãy cùng tìm hiểu cách để truy cập các phần tử riêng lẻ trong một danh sách.

3.1.1. Truy cập các phần tử trong danh sách

Danh sách là các tập hợp có thứ tự, vì vậy ta có thể truy cập bất kỳ phần tử nào trong danh sách bằng cách cho Python biết vị trí hoặc chỉ mục của mục mong muốn.

Để truy cập một phần tử trong danh sách, hãy viết tên của danh sách sau là chỉ mục của mục được đặt trong dấu ngoặc vuông. Ví dụ: hãy lấy chiếc xe đạp đầu tiên trong danh sách xe đạp:

```
bicycles = ['trek', 'cannondale', 'redline', 'specialized']
① print(bicycles[0])
```

Cú pháp cho việc này như câu lệnh ở ①. Khi chúng ta yêu cầu một mục duy nhất từ một danh sách, Python chỉ trả về phần tử đó mà không có dấu ngoặc vuông:

trek

Đây là kết quả ta muốn người dùng nhìn thấy — được định dạng gọn gàng, rõ ràng tại đầu ra. Chúng ta cũng có thể sử dụng các phương thức chuỗi từ Chương 2 trên bất kỳ phần tử nào trong danh sách này. Ví dụ: ta có thể định dạng phần tử 'trek' gọn gàng hơn bằng cách sử dụng phương thức title():

```
bicycles = ['trek', 'cannondale', 'redline', 'specialized']
print(bicycles[0].title())
```

Ví dụ này tạo ra cùng một đầu ra như ví dụ trước ngoại trừ 'Trek' được viết hoa.

3.1.2. Đánh chỉ mục

Python coi mục đầu tiên trong danh sách nằm ở vị trí 0, không phải vị trí 1. Điều này đúng với hầu hết các ngôn ngữ lập trình và lý do liên quan đến cách các thao tác trong danh sách được triển khai ở cấp thấp hơn. Nếu ta đang nhận kết quả không như mong đợi, xác định xem ta có đang gặp phải lỗi chỉ mục hay không.

Mục thứ hai trong danh sách có chỉ số là 1. Sử dụng hệ thống đếm này, chúng ta có thể lấy bất kỳ phần tử nào ta muốn từ danh sách bằng cách trừ đi một phần tử vị trí của nó trong danh sách. Ví dụ: để truy cập mục thứ tư trong danh sách, ta yêu cầu phần tử chỉ mục 3.

Đoạn code sau yêu cầu các xe đạp tại chỉ mục 1 và chỉ mục 3:

```
bicycles = ['trek', 'cannondale', 'redline', 'specialized']
print(bicycles[1])
print(bicycles[3])
```

Đoạn code trả về chiếc xe đạp thứ hai và thứ tư trong danh sách:

```
cannondale
specialized
```

Python có một cú pháp đặc biệt để truy cập phần tử cuối cùng trong danh sách. Bằng cách yêu cầu mục ở chỉ mục -1, Python luôn trả về mục cuối cùng trong danh sách:

```
bicycles = ['trek', 'cannondale', 'redline', 'specialized']
print(bicycles[-1])
```

Mã này trả về giá trị 'specialized'. Cú pháp này khá hữu ích, bởi vì ta thường muốn truy cập các mục cuối cùng trong danh sách mà không biết chính xác là danh sách dài bao nhiêu. Quy ước này mở rộng cho các chỉ mục giá trị âm khác cũng vậy. Chỉ mục -2 trả về mục thứ hai từ cuối danh sách, chỉ số -3 trả về mục thứ ba từ cuối, v.v.

3.1.3. Sử dụng giá trị riêng từ danh sách

Chúng ta có thể sử dụng các giá trị riêng lẻ từ một danh sách giống như cách ta làm với bất kỳ biến nào khác. Ví dụ: ta có thể sử dụng f-string để tạo một thông báo dựa trên giá trị từ một danh sách.

Hãy thử lấy chiếc xe đạp đầu tiên khỏi danh sách và tạo thông điệp sử dụng giá trị đó.

```
bicycles = ['trek', 'cannondale', 'redline', 'specialized']
① message = f"My first bicycle was a {bicycles[0].title()}."
print(message)
```

Tại ①, chúng ta tạo ra một chuỗi sử dụng giá trị tại bicycles[0] và gán nó cho biến message. Đầu ra là một câu đơn giản về chiếc xe đạp đầu tiên có trong danh sách:

My first bicycle was a Trek.

3.2. Thay đổi, thêm, và xóa các phần tử

Hầu hết các danh sách tạo sẽ là động, nghĩa là ta sẽ xây dựng một danh sách và sau đó thêm và xóa các phần tử khỏi nó khi chương trình chạy. Ví dụ, ta có thể tạo một trò chơi trong đó người chơi phải bắn quái vật trên bầu trời. Ta có thể lưu trữ nhóm quái vật ban đầu trong một danh sách và sau đó xóa một quái vật trong danh sách mỗi khi quái vật bị bắn hạ. Mỗi lần một quái vật mới xuất hiện trên màn hình, ta thêm nó vào danh sách. Danh sách quái vật sẽ tăng lên và giảm độ dài trong suốt quá trình của trò chơi.

3.2.1. Thay đổi phần tử trong danh sách

Cú pháp để sửa đổi một phần tử tương tự như cú pháp để truy cập một phần tử trong danh sách. Để thay đổi một phần tử, hãy sử dụng tên của danh sách theo sau

bằng chỉ mục của phần tử ta muốn thay đổi, sau đó cung cấp giá trị ta muốn phần tử đó có.

Ví dụ: giả sử chúng ta có danh sách xe máy và mục đầu tiên trong danh sách là 'honda'. Làm thế nào chúng ta sẽ thay đổi giá trị của mục đầu tiên này?

```
① motorcycles = ['honda', 'yamaha', 'suzuki']
print(motorcycles)
② motorcycles[0] = 'ducati'
print(motorcycles)
```

Đoạn mã tại ① định nghĩa danh sách ban đầu với 'honda' là phần tử đầu tiên. Đoạn mã tại ② thay đổi giá trị của phần tử đầu tiên thành 'ducati'. Kết quả cho thấy phần tử đầu tiên đã thực sự thay đổi và các phần tử còn lại giữ nguyên giá trị.

```
['honda', 'yamaha', 'suzuki']
['ducati', 'yamaha', 'suzuki']
```

Ta có thể thay đổi giá trị của bất cứ phần tử nào trong danh sách, không chỉ là phần tử đầu tiên.

3.2.2. Thêm phần tử vào danh sách

Chúng ta có thể muốn thêm một phần tử mới vào danh sách vì nhiều lý do. Ví dụ, ta có thể muốn làm cho quái vật mới xuất hiện trong một trò chơi, thêm mới dữ liệu để hiển thị hoặc thêm người dùng đã đăng ký mới vào trang web mà ta đã xây dựng. Python cung cấp một số cách để thêm dữ liệu mới vào danh sách hiện có.

Thêm phần tử vào cuối danh sách

Cách đơn giản nhất để thêm một phần tử mới vào danh sách là `nối` (append) phần tử đó vào danh sách. Khi chúng ta nối một mục vào danh sách, phần tử mới sẽ được thêm vào cuối của danh sách. Sử dụng cùng một danh sách mà chúng ta đã có trong ví dụ trước, chúng ta sẽ thêm phần tử mới 'ducati' ở cuối danh sách:

```
motorcycles = ['honda', 'yamaha', 'suzuki']
print(motorcycles)
① motorcycles.append('ducati')
print(motorcycles)
```

Phương thức `append()` tại ① thêm 'ducati' vào cuối danh sách mà không ảnh hưởng tới bất kỳ phần tử nào trong danh sách.

Phương thức `append()` giúp dễ dàng tạo danh sách động. Ví dụ, ta có thể bắt đầu với một danh sách trống và sau đó thêm các mục vào danh sách sử dụng một loạt

các lệnh gọi append(). Sử dụng một danh sách trống, cùng thêm các phần tử 'honda', 'yamaha', và 'suzuki' vào trong danh sách:

```
motorcycles = []
motorcycles.append('honda')
motorcycles.append('yamaha')
motorcycles.append('suzuki')
print(motorcycles)
```

Kết quả trả về danh sách y hệt danh sách trong ví dụ trước:

```
['honda', 'yamaha', 'suzuki']
```

Việc xây dựng danh sách theo cách này rất phổ biến, vì chúng ta thường không biết dữ liệu mà người dùng muốn lưu trữ trong một chương trình cho đến sau khi chương trình đó chạy. Để kiểm soát người dùng, hãy bắt đầu bằng cách định nghĩa danh sách trống sẽ lưu các phần tử. Sau đó, thêm từng giá trị mới được cung cấp vào danh sách vừa tạo.

Thêm một phần tử vào danh sách

Chúng ta có thể thêm một phần tử mới ở bất kỳ vị trí nào trong danh sách bằng cách sử dụng phương thức insert(). Ta thực hiện việc này bằng cách chỉ định chỉ mục của phần tử mới và giá trị của phần tử mới.

```
motorcycles = ['honda', 'yamaha', 'suzuki']
①   motorcycles.insert(0, 'ducati')
print(motorcycles)
```

Trong ví dụ này, đoạn mã tại ① thêm 'ducati' vào đầu danh sách. Phương thức insert() tạo ra một khoảng trống tại giá trị 0 và lưu giá trị 'ducati' tại vị trí đó. Hành động này dịch chuyển tất cả các phần tử còn lại một vị trí sang phải.

```
['ducati', 'honda', 'yamaha', 'suzuki']
```

3.2.3. Xóa phần tử khỏi danh sách

Thông thường, chúng ta sẽ muốn xóa một mục hoặc một tập hợp các mục khỏi danh sách. Ví dụ: khi một người chơi bắn hạ một quái vật từ trên trời, ta sẽ có khả năng muốn xóa nó khỏi danh sách những quái vật đang hoạt động. Hoặc khi một người dùng quyết định hủy tài khoản của họ trên một ứng dụng web ta đã tạo, chúng ta sẽ muốn xóa người dùng đó khỏi danh sách người dùng đang hoạt động. Ta có thể loại bỏ một phần tử theo vị trí của nó trong danh sách hoặc theo giá trị của nó.

Xóa phần tử sử dụng lệnh del

Nếu ta biết vị trí của phần tử trong danh sách và ta muốn xóa nó, có thể dùng câu lệnh del.

```
motorcycles = ['honda', 'yamaha', 'suzuki']
print(motorcycles)
```

```
① del motorcycles[0]
print(motorcycles)
```

Đoạn mã tại ① sử dụng del để xóa phần tử đầu tiên là 'honda' trong danh sách các xe máy:

```
['honda', 'yamaha', 'suzuki']
['yamaha', 'suzuki']
```

Ta có thể xóa một mục khỏi bất kỳ vị trí nào trong danh sách bằng cách sử dụng câu lệnh del nếu biết chỉ mục của nó. Ví dụ: đây là cách xóa mục thứ hai, 'yamaha', trong danh sách:

```
motorcycles = ['honda', 'yamaha', 'suzuki']
print(motorcycles)
del motorcycles[1]
print(motorcycles)
```

Xe máy thứ hai được xóa khỏi danh sách:

```
['honda', 'yamaha', 'suzuki']
['honda', 'suzuki']
```

Trong cả hai ví dụ trên, chúng ta không thể truy cập vào được phần tử bị xóa khỏi danh sách sau khi sử dụng lệnh del.

Xóa một phần tử sử dụng phương thức pop()

Đôi khi chúng ta muốn sử dụng giá trị của một phần tử sau khi xóa nó khỏi danh sách. Ví dụ: ta có thể muốn lấy vị trí x và y của một quái vật vừa bị bắn hạ, để ta có thể tạo ra một vụ nổ tại vị trí đó. Trong một ứng dụng web, ta có thể muốn xóa một người dùng khỏi danh sách các thành viên đang hoạt động và sau đó thêm người dùng đó vào danh sách các thành viên không hoạt động.

Phương thức pop() loại bỏ mục cuối cùng trong danh sách, nhưng nó cho phép ta làm việc với mục đó sau khi loại bỏ nó. Thuật ngữ pop xuất phát từ suy nghĩ của một liệt kê dưới dạng một chồng các phần tử và lấy một mục ra khỏi đầu ngăn xếp. Trong sự tương tự này, phần đầu của một ngăn xếp tương ứng với phần cuối của một danh sách. Hãy cùng lấy một xe máy ra khỏi danh sách các xe máy:

```

①  motorcycles = ['honda', 'yamaha', 'suzuki']
print(motorcycles)
②  popped_motorcycle = motorcycles.pop()
③  print(motorcycles)
④  print(popped_motorcycle)

```

Đầu tiên, chúng ta định nghĩa danh sách tại ①. Tại ②, chúng ta lấy ra một phần tử ở danh sách và lưu nó trong biến popped_motorcycle. Chúng ta in ra danh sách tại ③ để thấy rằng phần tử đã bị gỡ khỏi danh sách, sau đó chúng ta in ra phần tử bị gỡ tại ④ để thấy rằng chúng ta vẫn có thể truy cập phần tử đã bị gỡ. Phần in ra chỉ biết giá trị 'suzuki' bị gỡ khỏi cuối danh sách và hiện tại được gán cho biến popped_motorcycle.

```

['honda', 'yamaha', 'suzuki']
['honda', 'yamaha']
suzuki

```

Phương thức pop() này hữu ích như thế nào? Hãy tưởng tượng rằng những chiếc xe máy trong danh sách được lưu trữ theo thứ tự thời gian theo thời điểm chúng ta sở hữu chúng. Nếu đúng như vậy, chúng ta có thể sử dụng phương thức pop() để in một câu lệnh về chiếc xe máy cuối cùng chúng ta mua:

```

motorcycles = ['honda', 'yamaha', 'suzuki']
last_owned = motorcycles.pop()
print(f"The last motorcycle I owned was a {last_owned.title()}.")

```

Thông báo ra đơn giản là chiếc mô tô gần nhất được mua:

```
The last motorcycle I owned was a Suzuki
```

Lấy phần tử từ bất kỳ vị trí nào trong danh sách

Chúng ta có thể sử dụng pop() để xóa một mục khỏi bất kỳ vị trí nào trong danh sách bằng cách bao gồm chỉ mục của mục mà ta muốn xóa trong ngoặc đơn.

```

motorcycles = ['honda', 'yamaha', 'suzuki']
①  first_owned = motorcycles.pop(0)
②  print(f"The first motorcycle I owned was a {first_owned.title()}.")

```

Tại ①, chúng ta lấy ra chiếc mô tô đầu tiên và in ra thông điệp về chiếc mô tô tại dòng ②. Phần in ra là thông điệp đơn giản về chiếc mô tô ta vừa sở hữu:

```
The first motorcycle I owned was a Honda.
```

Cần nhớ rằng mỗi lần sử dụng phương thức pop(), phần tử mà chúng ta lấy ra sẽ không còn được lưu trong danh sách. Nếu ta không chắc nên sử dụng câu lệnh del hay phương thức pop(), đây là một cách đơn giản để quyết định: khi ta muốn xóa một

mục khỏi danh sách và không sử dụng mục đó theo bất kỳ cách nào, hãy sử dụng câu lệnh del; nếu ta muốn sử dụng phần tử khi ta xóa nó, hãy sử dụng phương thức pop().

Xóa phần tử bằng giá trị

Đôi khi chúng ta không biết vị trí của giá trị muốn xóa từ một danh sách. Nếu ta chỉ biết giá trị của mục muốn xóa, ta có thể sử dụng phương thức remove().

Ví dụ, nếu chúng ta muốn xóa giá trị ‘ducati’ từ danh sách các xe gắn máy:

```
motorcycles = ['honda', 'yamaha', 'suzuki', 'ducati']
print(motorcycles)
①   motorcycles.remove('ducati')
print(motorcycles)
```

Phần code tại ① nói với Python là tìm ra vị trí của ‘ducati’ trong danh sách và xóa nó đi.

```
['honda', 'yamaha', 'suzuki', 'ducati']
['honda', 'yamaha', 'suzuki']
```

Chúng ta cũng có thể sử dụng phương thức remove() để làm việc với một giá trị đang bị xóa khỏi danh sách. Hãy xóa giá trị ‘ducati’ và in lý do xóa nó khỏi danh sách:

```
①   motorcycles = ['honda', 'yamaha', 'suzuki', 'ducati']
print(motorcycles)
②   too_expensive = 'ducati'
③   motorcycles.remove(too_expensive)
print(motorcycles)
④   print(f"\nA {too_expensive.title()} is too expensive for me.")
```

Sau khi định nghĩa danh sách tại ①, chúng ta gán ‘ducati’ cho biến *too_expensive* tại ②. Sau đó, chúng ta dùng biến này để nói cho Python giá trị cần xóa đi trong danh sách tại ③ và tại ④ thì giá trị ‘ducati’ đã bị xóa khỏi danh sách nhưng vẫn có thể truy cập được thông qua biến *too_expensive*, cho phép chúng ta in ra lý do xóa ‘ducati’ ra khỏi danh sách các xe.

```
['honda', 'yamaha', 'suzuki', 'ducati']
['honda', 'yamaha', 'suzuki']
A Ducati is too expensive for me.
```

Ghi chú: Phương thức remove() chỉ xóa lần xuất hiện đầu tiên của giá trị ta chỉ định. Nếu có khả năng giá trị xuất hiện nhiều lần trong danh sách, chúng ta sẽ cần sử dụng một vòng lặp để đảm bảo rằng tất cả các lần xuất hiện của giá trị đều bị loại bỏ.

3.3. Tổ chức danh sách

Thông thường, danh sách sẽ được tạo theo một thứ tự không thể biết trước, bởi vì không thể kiểm soát thứ tự mà người dùng cung cấp dữ liệu của họ. Vấn đề là ta luôn cần trình bày thông tin theo một trật tự nào đó. Đôi khi, ta cần giữ nguyên thứ tự ban đầu của dữ liệu, đôi khi lại cần thay đổi thứ tự theo lúc đầu. Python cung cấp một số cách khác nhau để thay đổi thứ tự tùy theo tình huống.

3.3.1. Sắp xếp danh sách vĩnh viễn với phương thức sort()

Phương thức `sort()` của Python giúp việc sắp xếp danh sách tương đối dễ dàng. Hãy tưởng tượng chúng ta có một danh sách các ô tô và muốn thay đổi thứ tự của danh sách để lưu trữ chúng theo thứ tự bảng chữ cái. Để giữ cho nhiệm vụ đơn giản, hãy giả sử rằng tất cả các giá trị trong danh sách đều là chữ thường.

```
cars = ['bmw', 'audi', 'toyota', 'subaru']
cars.sort()
print(cars)
```

Phương thức `sort()` của phần mã trên thay đổi thứ tự của danh sách vĩnh viễn. Những chiếc xe hiện đã được xếp theo thứ tự bảng chữ cái và chúng ta không bao giờ có thể hoàn nguyên về thứ tự ban đầu:

```
['audi', 'bmw', 'subaru', 'toyota']
```

Ta cũng có thể sắp xếp danh sách này theo thứ tự bảng chữ cái ngược lại bằng cách chuyển đổi số `reverse = True` vào phương thức `sort()`. Ví dụ sau sắp xếp danh sách ô tô theo thứ tự bảng chữ cái ngược lại:

```
cars = ['bmw', 'audi', 'toyota', 'subaru']
cars.sort(reverse=True)
print(cars)
```

Tương tự như trên, thứ tự các phần tử danh sách sẽ thay đổi vĩnh viễn

```
['toyota', 'subaru', 'bmw', 'audi']
```

3.3.2. Sắp xếp danh sách tạm thời với phương thức sorted()

Để duy trì thứ tự ban đầu của danh sách nhưng trình bày nó theo thứ tự đã sắp xếp, ta có thể sử dụng hàm `sorted()`. Hàm `sorted()` cho phép hiển thị danh sách theo một thứ tự cụ thể nhưng không ảnh hưởng đến thứ tự thực tế của danh sách.

Ta thử hàm này với danh sách `cars`:

```
cars = ['bmw', 'audi', 'toyota', 'subaru']
print("Here is the original list:")
```

```

print(cars)
print("\nHere is the sorted list:")
print(sorted(cars))
print("\nHere is the original list again:")
print(cars)

```

Đầu tiên, ta khai báo danh sách xe là cars và in ra danh sách với thứ tự gốc. Tiếp theo, chúng ta in ra kết quả của phương thức sorted(cars) để in ra danh sách cars đã được sắp xếp các phần tử. Cuối cùng, chúng ta lại in ra danh sách cars để thấy thứ tự ban đầu không bị thay đổi.

```

Here is the original list:
['bmw', 'audi', 'toyota', 'subaru']
Here is the sorted list:
['audi', 'bmw', 'subaru', 'toyota']
Here is the original list again:
['bmw', 'audi', 'toyota', 'subaru']

```

Hàm *sorted()* cũng có thể chấp nhận đối số *reverse = True* nếu ta muốn hiển thị danh sách theo thứ tự bảng chữ cái ngược lại.

Ghi chú: Sắp xếp danh sách theo thứ tự bảng chữ cái phức tạp hơn một chút khi tất cả các giá trị không ở dạng chữ thường. Có một số cách để diễn giải các chữ cái viết hoa khi xác định thứ tự sắp xếp và việc chỉ định thứ tự chính xác có thể phức tạp hơn chúng ta muốn giải quyết. tại thời điểm này. Tuy nhiên, hầu hết các phương pháp sắp xếp sẽ trực tiếp dựa trên những gì ta đã học trong phần này.

3.3.3. In danh sách theo thứ tự ngược

Để đảo ngược thứ tự ban đầu của danh sách, ta có thể sử dụng phương thức *reverse()*. Nếu ban đầu chúng ta lưu trữ danh sách ô tô theo thứ tự thời gian tùy theo thời điểm chúng ta sở hữu chúng, chúng ta có thể dễ dàng sắp xếp lại danh sách theo thứ tự thời gian ngược lại:

```

cars = ['bmw', 'audi', 'toyota', 'subaru']
print(cars)
cars.reverse()
print(cars)

```

Lưu ý rằng phương thức *reverse()* không sắp xếp theo thứ tự alphabet, nó chỉ đơn thuần là đảo ngược thứ tự của các phần tử trong danh sách hiện tại.

```

['bmw', 'audi', 'toyota', 'subaru']
['subaru', 'toyota', 'audi', 'bmw']

```

Phương thức *reverse()* thay đổi thứ tự của danh sách vĩnh viễn, nhưng ta có thể hoàn nguyên về thứ tự ban đầu bất kỳ lúc nào bằng cách áp dụng *reverse()* cho cùng một danh sách lần thứ hai.

3.3.4. Tìm độ dài của danh sách

Ta có thể nhanh chóng tìm độ dài của danh sách bằng cách sử dụng hàm *len()*.

Danh sách trong ví dụ này có bốn mục, vì vậy độ dài của nó là 4:

```
>>> cars = ['bmw', 'audi', 'toyota', 'subaru']
>>> len(cars)
4
```

Ta sẽ thấy *len()* hữu ích khi cần xác định số lượng quái vật vẫn cần bị bắn hạ trong một trò chơi, xác định lượng dữ liệu phải quản lý bằng hình ảnh trực quan hoặc tìm hiểu số lượng người dùng đã đăng ký một trang web, trong số các nhiệm vụ khác.

Chú ý: Python đếm các mục trong danh sách bắt đầu bằng một, vì vậy ta sẽ không gặp phải bất kỳ lỗi nào khi xác định độ dài của danh sách

3.4. Tránh lỗi chỉ mục

Một trong những lỗi phổ biến của người lập trình là lỗi chỉ mục (index). Giả sử danh sách có 3 phần tử, ta yêu cầu in ra phần tử thứ tư, Python sẽ thông báo lỗi chỉ mục

```
motorcycles = ['honda', 'yamaha', 'suzuki']
print(motorcycles[3])
```

Lỗi:

```
Traceback (most recent call last):
File "motorcycles.py", line 2, in <module>
print(motorcycles[3])
IndexError: list index out of range
```

Python được yêu cầu đưa ra phần tử thứ tư trong danh sách, tuy nhiên khi nó tìm kiếm trong danh sách, không có mục tin nào của danh sách motocycles có chỉ mục là 3. Vấn đề là việc lập chỉ mục với danh sách được bắt đầu từ 0 chứ không phải 1. Do đó, phần tử có chỉ mục là 3 tương ứng là phần tử thứ tư trong danh sách và hiện phần tử đó không tồn tại.

Lỗi chỉ mục có nghĩa là Python không thể kết xuất một mục tại chỉ mục được yêu cầu. Nếu lỗi chỉ mục xảy ra trong chương trình, hãy thử điều chỉnh chỉ mục đang được yêu cầu một đơn vị. Sau đó chạy lại chương trình để xem kết quả. Cần nhớ là

bất cứ khi nào ta muốn truy cập phần tử cuối cùng trong danh sách, hãy sử dụng chỉ mục (-1). Điều này luôn đúng ngay cả khi danh sách bị thay đổi từ lần truy cập cuối cùng.

```
motorcycles = ['honda', 'yamaha', 'suzuki']
print(motorcycles[-1])
```

Chỉ mục -1 luôn trả về mục cuối cùng trong danh sách, trong trường hợp này là giá trị 'suzuki':

```
'suzuki'
```

Các tiếp cận này chỉ xảy ra lỗi trong trường hợp duy nhất đó là khi danh sách trống.

```
motorcycles = []
print(motorcycles[-1])
```

Không có mục tin nào trong Danh sách motorcycles, do đó Python trả về một lỗi danh mục khác:

```
Traceback (most recent call last):
File "motorcycles.py", line 3, in <module>
print(motorcycles[-1])
IndexError: list index out of range
```

Nếu lỗi chỉ mục xảy ra và ta không thể tìm ra cách giải quyết, hãy thử in danh sách hoặc chỉ in độ dài của danh sách. Danh sách có thể trống khác nhiều so với định hình của ta, đặc biệt nếu nó đã được quản lý động bởi chương trình. Xem độ dài danh sách, hoặc các phần tử trong danh sách được in ra, ta có thể đoán ra lỗi đã xảy ra.

3.5. Lặp qua toàn bộ danh sách

Một tác vụ quan trọng mà chương trình thường phải làm là chạy qua toàn bộ các phần tử trong danh sách và thực hiện các việc giống nhau. Ví dụ: trong một trò chơi, ta có thể muốn di chuyển mọi phần tử trên màn hình theo cùng một khoảng hoặc trong danh sách các số ta có thể muốn thực hiện cùng một thao tác thống kê trên mọi phần tử, hoặc khi muốn hiển thị các Tiêu đề của các bài báo trong danh sách. Khi ta muốn thực hiện cùng một hành động với mọi mục trong danh sách, ta có thể sử dụng vòng lặp for của Python.

Giả sử ta có danh sách tên các nhà ảo thuật và muốn in ra mỗi tên trong danh sách. Ta có thể làm điều này bằng cách truy xuất từng tên từ liệt kê riêng lẻ, nhưng

cách tiếp cận này có thể gây ra một số vấn đề. Ví dụ, việc liệt kê riêng lẻ sẽ gặp vấn đề với một danh sách dài các tên, hoặc việc viết mã nguồn sẽ phải thực hiện lại mỗi khi độ dài của danh sách thay đổi. Vòng lặp for tránh được cả hai vấn đề nêu trên.

Ta dùng một vòng for để in ra mỗi tên trong danh sách các nhà ảo thuật:

```
magicians = ['alice', 'david', 'carolina']
for magician in magicians:
    print(magician)
```

Tại dòng đầu tiên, ta định nghĩa một danh sách có tên là `magicians` với ba phần tử. Dòng thứ hai khai báo một vòng lặp `for`. Dòng này cho Python biết sẽ lấy ra tên trong danh sách và gán nó vào biến `magician`. Dòng thứ ba yêu cầu Python in ra tên đã được gán vào biến `magician`. Python sẽ lặp lại dòng 2 và dòng 3 đối với mỗi tên trong danh sách. Điều này cho phép đọc toàn bộ tên trong danh sách các nhà ảo thuật và in ra tên mỗi nhà ảo thuật.

```
alice
david
carolina
```

3.5.1. Cái nhìn cận cảnh hơn về vòng lặp

Vòng lặp rất quan trọng vì nó là một trong những cách máy tính tự động hóa các tác vụ lặp đi lặp lại. Ví dụ, trong một vòng lặp đơn giản, giống như ví dụ bên trên, đầu tiên, Python đọc dòng đầu tiên của vòng lặp:

```
for magician in magicians:
```

Dòng này yêu cầu Python lấy giá trị đầu tiên từ danh sách `magicians` và gán nó với biến `magician`. Giá trị đầu tiên là ‘`alice`’. Sau đó Python đọc dòng tiếp theo:

```
print(magician)
```

In giá trị hiện tại của `magician` là ‘`alice`’. Vì danh sách này chứa nhiều giá trị hơn, Python trả về dòng đầu tiên của vòng lặp.

```
for magician in magicians:
```

Python lấy ra tên tiếp theo trong vòng lặp là ‘`david`’ và gán nó cho biến `magician`, sau đó python thực hiện lệnh:

```
print(magician)
```

Python in lại giá trị hiện tại của `magician`, giá trị hiện tại là ‘`david`’. Python lặp lại toàn bộ vòng lặp một lần nữa với giá trị cuối cùng trong danh sách, ‘`carolina`’. Bởi

vì không có giá trị nào nữa trong danh sách, Python chuyển sang dòng tiếp theo trong chương trình. Trong trường hợp này, không có gì xuất hiện sau vòng lặp for, vì vậy chương trình chỉ đơn giản là kết thúc.

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

Ngoài ra, hãy ghi nhớ khi viết các vòng lặp, ta có thể chọn bất kỳ tên nào chúng ta muốn cho biến tạm thời sẽ được liên kết với mỗi giá trị trong danh sách. Tuy nhiên, sẽ hữu ích khi chọn một cái tên có ý nghĩa đại diện cho một mục duy nhất từ danh sách. Ví dụ: đây là một cách tốt để bắt đầu vòng lặp for cho danh sách mèo, danh sách chó và danh sách chung các mục:

```
for cat in cats:  
for dog in dogs:  
for item in list_of_items:
```

Các quy ước đặt tên này có thể giúp ta theo dõi hành động đang được thực hiện trên mỗi mục trong vòng lặp for. Sử dụng tên số ít và số nhiều có thể giúp chúng ta xác định xem một phần mã có đang hoạt động với một phần tử duy nhất từ danh sách hoặc toàn bộ danh sách.

3.5.2. *Bổ sung thêm về vòng lặp*

Chúng ta có thể làm bất cứ điều gì với mỗi mục trong vòng lặp for. Hãy tiếp tục ví dụ trước bằng cách in một tin nhắn cho mỗi ảo thuật gia, nói với họ rằng họ đã thực hiện một thủ thuật tuyệt vời:

```
magicians = ['alice', 'david', 'carolina']  
for magician in magicians:  
①     print(f'{magician.title()}', that was a great trick!")
```

Sự khác nhau duy nhất là ở ① khi chúng ta in ra thông điệp cho các nhà ảo thuật, bắt đầu với tên của nhà ảo thuật đó. Lần đầu tiên qua vòng lặp giá trị của *magician* là '*alice*', vì vậy Python bắt đầu thông báo đầu tiên với tên '*Alice*'. Lần thứ hai chạy thông điệp sẽ bắt đầu với '*David*', và lần thứ ba thông điệp sẽ bắt đầu bằng '*Carolina*'. Đầu ra hiển thị một thông báo được cá nhân hóa cho từng ảo thuật gia trong danh sách:

```
Alice, that was a great trick!  
David, that was a great trick!  
Carolina, that was a great trick!
```

Chúng ta cũng có thể viết bao nhiêu dòng mã tùy thích trong vòng lặp *for*. Mỗi dòng thut lè sau dòng dành cho ảo thuật gia trong các nhà ảo thuật được coi là bên trong vòng lặp và mỗi dòng thut vào được thực hiện một lần cho mỗi giá trị trong danh sách. Do đó, ta có thể làm nhiều việc tùy thích với mỗi giá trị trong danh sách.

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

```
magicians = ['alice', 'david', 'carolina']  
for magician in magicians:  
    print(f"{magician.title()}, that was a great trick!")  
①    print(f"I can't wait to see your next trick, {magician.title()}.\\n")
```

Bởi vì chúng ta đã thut lè cả hai lệnh gọi đến *print()*, mỗi dòng sẽ được thực thi một lần cho mọi ảo thuật gia trong danh sách. Dòng mới ("\\n") trong lệnh *in* thứ hai ① chèn một dòng trống sau mỗi lần đi qua vòng lặp. Điều này tạo ra một tập hợp thông điệp được nhóm gọn gàng cho từng người trong danh sách:

```
Alice, that was a great trick!  
I can't wait to see your next trick, Alice.
```

```
David, that was a great trick!  
I can't wait to see your next trick, David.
```

```
Carolina, that was a great trick!  
I can't wait to see your next trick, Carolina.
```

Chúng ta có thể sử dụng bao nhiêu dòng tùy thích trong vòng lặp *for*. Trong thực tế, ta sẽ thường thấy hữu ích khi thực hiện một số thao tác khác nhau với từng mục trong danh sách khi ta sử dụng vòng lặp *for*.

3.5.3. Một số việc sau vòng lặp *For*

Điều gì xảy ra khi một vòng lặp *for* kết thúc quá trình thực thi? Thông thường, ta sẽ muốn để tóm tắt một khối đầu ra hoặc chuyển sang công việc khác mà chương trình ta phải hoàn thành.

Bất kỳ dòng mã nào sau vòng lặp *for* không được thut lè đều được thực thi một lần mà không lặp lại. Cùng viết lời cảm ơn đến nhóm ảo thuật gia nói chung, cảm ơn họ vì đã có một buổi biểu diễn xuất sắc. Để hiển thị thông điệp này sau khi tất cả các

thông điệp riêng lẻ đã được in, chúng ta đặt thông báo cảm ơn sau vòng lặp `for` mà không có thụt lề:

```
magicians = ['alice', 'david', 'carolina']
for magician in magicians:
    print(f"{magician.title()}, that was a great trick!")
    print(f"I can't wait to see your next trick, {magician.title()}.\\n")
①    print("Thank you, everyone. That was a great magic show!")
```

Hai lệnh gọi đầu tiên đến `print()` được lặp lại một lần cho mỗi ảo thuật gia trong danh sách, như ta đã thấy trước đó. Tuy nhiên, vì dòng tại ① không được thụt vào, nó chỉ in một lần:

```
Alice, that was a great trick!
I can't wait to see your next trick, Alice.
```

```
David, that was a great trick!
I can't wait to see your next trick, David.
```

```
Carolina, that was a great trick!
I can't wait to see your next trick, Carolina.
```

```
Thank you, everyone. That was a great magic show!
```

Khi chúng ta đang xử lý dữ liệu bằng vòng lặp `for`, ta sẽ nghĩ rằng đây là cách tốt để tóm tắt một thao tác đã được thực hiện trên toàn bộ tập dữ liệu. Ví dụ: ta có thể sử dụng vòng lặp `for` để khởi tạo trò chơi bằng cách chạy thông qua danh sách các ký tự và hiển thị từng ký tự trên màn hình. Sau đó, ta có thể viết một số mã bổ sung sau vòng lặp này hiển thị Nút Play Now sau khi tất cả các ký tự đã được vẽ trên màn hình.

3.5.4. Tránh lỗi thụt lề

Python sử dụng thụt đầu dòng để xác định cách một dòng hoặc một nhóm dòng có liên quan phần còn lại của chương trình. Trong các ví dụ trước, các dòng `print()` in tin nhắn cho từng nhà ảo thuật là một phần của vòng lặp `for` vì chúng đã được thụt vào. Việc sử dụng thụt lề trong Python làm cho mã rất dễ đọc.

Về cơ bản, python sử dụng khoảng trắng để buộc viết mã có định dạng gọn gàng với một cấu trúc trực quan rõ ràng. Trong các chương trình Python dài hơn, chúng ta sẽ nhận thấy các khối mã được thụt lề ở một vài cấp độ khác nhau. Các mức thụt lề này giúp ta có được cảm nhận chung về tổ chức chung của chương trình.

Khi chúng ta bắt đầu viết mã dựa vào thuật lề thích hợp, ta sẽ cần để ý một vài lỗi thuật lề phổ biến. Ví dụ, đôi khi thuật lề dòng mã không cần phải thuật lề hoặc quên để thuật lề các dòng cần thuật lề. Xem ví dụ về những lỗi này sẽ giúp tránh chúng trong tương lai và sửa chúng khi chúng xuất hiện trong các chương trình của mình.

Quên thuật lề

Luôn thuật lề dòng sau câu lệnh for trong một vòng lặp. Nếu quên, Python sẽ nhắc nhở ta.

```
magicians = ['alice', 'david', 'carolina']
for magician in magicians:
    ① print(magician)
```

Câu lệnh print() tại ① cần phải thuật dòng nhưng nó đã không được thực hiện như vậy. Khi Python mong đợi thuật dòng mà nó không tìm thấy, nó sẽ cho ta biết dòng nào đang có lỗi.

```
File "magicians.py", line 3
print(magician)
^
IndentationError: expected an indented block
```

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

Quên thuật lề các dòng bổ sung

Đôi khi, vòng lặp chạy mà không có bất kỳ lỗi nào nhưng sẽ không tạo ra kết quả mong đợi. Điều này có thể xảy ra khi ta đang cố gắng thực hiện một số tác vụ trong một vòng lặp và quên thuật lề một số dòng của nó. Ví dụ: đây là những gì sẽ xảy ra khi chúng ta quên thuật lề thứ hai dòng trong vòng lặp nói với mỗi ảo thuật gia rằng chúng ta đang mong chờ tiết mục tiếp theo của họ:

```
magicians = ['alice', 'david', 'carolina']
for magician in magicians:
    print(f"{magician.title()}, that was a great trick!")
    ① print(f"I can't wait to see your next trick, {magician.title()}.\\n")
```

Hàm print() được gọi tại ① cần phải được thuật lề, tuy nhiên Python nhận ra rằng có ít nhất một dòng thuật lề sau câu lệnh for nên nó không báo lỗi gì.

Kết quả là, lệnh print() đầu tiên được thực hiện một lần cho mỗi tên trong danh sách vì nó được thuật vào. Lời gọi print() thứ hai không được thuật vào, vì vậy nó là chỉ

thực hiện một lần sau khi vòng lặp chạy xong. Vì giá trị cuối cùng liên quan đến ảo thuật gia là 'carolina', cô ấy là người duy nhất nhận được thông báo " I can't wait to see your next trick ":

```
Alice, that was a great trick!  
David, that was a great trick!  
Carolina, that was a great trick!  
I can't wait to see your next trick, Carolina.
```

Đây là một lỗi logic. Cú pháp mã Python hợp lệ, nhưng mã này không tạo ra kết quả mong muốn bởi vì một vấn đề xảy ra trong logic của nó. Nếu chúng ta mong đợi thấy một hành động nhất định được lặp lại một lần cho mỗi mục trong danh sách và nó chỉ được thực hiện một lần, xác định xem có cần chỉ cần thực lè một dòng hay một nhóm các dòng.

Thụt lè không cần thiết

Nếu ta vô tình thụt lè một dòng không cần phải thụt lè, Python sẽ thông báo về sự thụt lè không mong muốn:

```
message = "Hello Python world!"  
①     print(message)
```

Chúng ta không cần thụt lè lệnh gọi *print()* tại ① bởi nó không là một thành phần của vòng lặp. Do đó, Python thông báo lỗi sau:

```
File "hello_world.py", line 2  
print(message)  
^  
IndentationError: unexpected indent
```

Chúng có thể tránh lỗi thụt lè không mong muốn bằng cách chỉ thụt lè khi ta có một lý do cụ thể để làm như vậy. Trong các chương trình đã viết tới thời điểm này, những dòng duy nhất ta nên thụt lè là những hành động ta muốn lặp lại cho mỗi mục trong vòng lặp for.

Thụt lè không cần thiết sau vòng lặp for

Nếu ta vô tình thụt lè mã sẽ chạy sau khi một vòng lặp kết thúc, có nghĩa là mã sẽ được lặp lại một lần cho mỗi mục trong danh sách. Đôi khi điều này nhắc nhở Python để báo lỗi, nhưng thường thì điều này sẽ dẫn đến lỗi logic.

Ví dụ: hãy xem điều gì sẽ xảy ra khi chúng ta vô tình thụt lè dòng cảm ơn các ảo thuật gia với tư cách là một nhóm đã mang đến một buổi biểu diễn hay:

```

magicians = ['alice', 'david', 'carolina']
for magician in magicians:
    print(f"{magician.title()}, that was a great trick!")
    print(f"I can't wait to see your next trick, {magician.title()}.\\n")
①   print("Thank you everyone, that was a great magic show!")

```

Do dòng code tại ① là th undercut, nó sẽ in thông điệp cùng với vòng lặp của mỗi nhà ảo thuật:

```

Alice, that was a great trick!
I can't wait to see your next trick, Alice.

```

```

Thank you everyone, that was a great magic show!
David, that was a great trick!
I can't wait to see your next trick, David.

```

```

Thank you everyone, that was a great magic show!
Carolina, that was a great trick!
I can't wait to see your next trick, Carolina.

```

Thank you everyone, that was a great magic show!

Lỗi này tương tự như lỗi *Th undercut không cần thiết* ở trên, thuộc về lỗi logic. Nghĩa là Python không phát hiện ra lỗi cú pháp nào nên vẫn chạy. Nếu một hành động được lặp lại nhiều lần khi hành động đó nên được thực hiện chỉ một lần, chúng ta cần kiểm tra undercut của mã cho hành động đó.

Quên dấu hai chấm ‘:’

Dấu hai chấm ở cuối câu lệnh for yêu cầu Python giải thích câu lệnh tiếp theo dòng là điểm bắt đầu của một vòng lặp.

```

magicians = ['alice', 'david', 'carolina']
①   for magician in magicians
        print(magician)

```

Nếu ta vô tình quên dấu hai chấm, như được hiển thị ở ①, ta sẽ nhận được một cú pháp do Python không biết ta đang cố gắng làm gì. Mặc dù đây là một lỗi dễ sửa, không phải lúc nào cũng dễ tìm ra lỗi. Chúng ta sẽ ngạc nhiên bởi lượng thời gian mà các lập trình viên dành để tìm kiếm các lỗi ký tự đơn như thế này.

3.6. Lập danh sách số

Có nhiều lý do để thiết lập một danh sách các số. Ví dụ ta cần theo dõi vị trí của nhân vật trong trò chơi, hoặc theo dõi điểm số cao của người chơi. Trong trực

quan hóa dữ liệu, ta luôn cần làm việc với tập hợp các số như nhiệt độ, khoảng cách, kích cỡ dân số, kinh độ, vĩ độ. Các dữ liệu này đều là các kiểu dữ liệu số khác nhau.

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

3.6.1. Sử dụng hàm range()

Hàm range() của Python giúp dễ dàng tạo một chuỗi số. Ví dụ: ta có thể sử dụng hàm range() để in một chuỗi số như sau:

```
for value in range(1, 5):
    print(value)
```

Mặc dù trong đoạn mã trên có thể in ra các số từ 1 tới 5 nhưng thực tế không phải như vậy, nó không in ra số 5:

```
1
2
3
4
```

Trong ví dụ này, *range()* chỉ in các số từ 1 đến 4. Đây là một kết quả khác của hành vi bót đi một mà ta sẽ thấy thường xuyên trong các ngôn ngữ lập trình. Hàm *range()* khiến Python bắt đầu đếm ở lần đầu tiên giá trị ta cung cấp và nó *dùng lại* khi đạt đến giá trị thứ hai mà ta cung cấp. Bởi vì nó dùng lại ở giá trị thứ hai đó, đầu ra không bao giờ chứa giá trị cuối, sẽ là 5 trong trường hợp này.

Để in ra số 5 trong trường hợp này, chúng ta dùng *range(1, 6)*.

```
for value in range(1, 6):
    print(value)
```

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

```
1
2
3
4
5
```

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

3.6.2. Sử dụng `range()` để tạo ra danh sách số

Nếu muốn tạo một danh sách các số, ta có thể chuyển đổi kết quả của hàm `range()` trực tiếp vào một danh sách bằng cách sử dụng hàm `list()`. Khi chúng ta bọc `list()` xung quanh một hàm `range()`, đầu ra sẽ là một danh sách các số.

Trong ví dụ trên, chúng ta chỉ cần in ra một loạt những con số. Chúng ta có thể sử dụng `list()` để chuyển đổi cùng một bộ số đó thành một danh sách:

```
numbers = list(range(1, 6))
print(numbers)
```

Kết quả đầu ra:

```
[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 đã cho. Nếu ta truyền đối số thứ ba vào `range()`, Python sẽ sử dụng giá trị đó như một kích thước bước khi tạo số.

```
even_numbers = list(range(2, 11, 2))
print(even_numbers)
```

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

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

Chúng có thể tạo hầu như bất kỳ tập hợp số nào theo ý muốn bằng cách sử dụng hàm `range()`. Ví dụ: hãy xem xét cách ta có thể tạo danh sách các số bình phương từ 10 (tức là bình phương của mỗi số nguyên từ 1 đến 10). Trong Python, hai dấu hoa thị (**) đại diện cho số mũ. Đây là cách ta có thể đặt 10 số bình phương đầu tiên vào một danh sách:

```
① squares = []
② for value in range(1, 11):
③     square = value ** 2
④     squares.append(square)
⑤ print(squares)
```

Chúng ta bắt đầu bởi một danh sách rỗng tại ①. Tại ②, chúng ta yêu cầu Python lặp từ 1 đến 10 sử dụng hàm `range()`. Trong vòng lặp, giá trị hiện tại được bình phương lên và được gán vào biến `square` ③. Tại ④, mỗi giá trị `square` được

thêm vào danh sách squares. Cuối cùng, khi vòng lặp kết thúc, danh sách squares được in ra ở ⑤.

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

Để viết mã này ngắn gọn hơn, hãy bỏ qua bình phương biến tạm thời và nối trực tiếp từng giá trị mới vào danh sách:

```
squares = []
for value in range(1,11):
①     squares.append(value**2)
print(squares)
```

Dòng mã tại ① thay thế hai dòng ③④ trong đoạn mã bên trên. Mỗi giá trị trong vòng lặp được lũy thừa thứ hai và sau đó được thêm vào danh sách squares.

Chúng ta có thể sử dụng một trong hai cách tiếp cận này khi làm việc với danh sách phức tạp hơn. Đôi khi việc sử dụng một biến tạm thời làm cho mã của ta dễ đọc hơn; những trong trường hợp khác, nó làm cho mã dài một cách không cần thiết.

3.6.3. Thống kê đơn giản với danh sách số

Một vài hàm Python hữu ích khi làm việc với danh sách các số. Ví dụ, ta có thể dễ dàng tổng hợp các giá trị tối thiểu, tối đa và tổng của một danh sách số:

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

3.6.4. Hiểu biết về danh sách

Cách tiếp cận được mô tả trước đó để tạo danh sách squares bao gồm sử dụng ba hoặc bốn dòng mã. Khả năng hiểu danh sách cho phép tạo danh sách như này chỉ trong một dòng mã. Việc hiểu danh sách kết hợp vòng lặp for và việc tạo các phần tử mới thành một dòng và tự động thêm mỗi phần tử mới. Việc hiểu về danh sách không phải lúc nào cũng được trình bày cho người mới bắt đầu, nhưng đã được đưa vào đây vì rất có thể ta 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 ta đã thấy ở trên nhưng sử dụng khả năng hiểu danh sách:

```
squares = [value**2 for value in range(1, 11)]
print(squares)
```

Để sử dụng cú pháp này, chúng ta bắt đầu với tên được mô tả cho danh sách, ví dụ là squares trong trường hợp này. Tiếp theo, mở ngoặc vuông '[' và định nghĩa biểu thức cho giá trị chúng ta muốn lưu ở trong danh sách đó. Trong ví dụ này, biểu thức là value**2, để bình phương giá trị của value. Sau đó, viết vòng for cho giá trị trong range(1,11) để đưa giá trị từ 1 tới 10 vào trong biểu thức. Ghi nhớ là không sử dụng dấu hai chấm ở cuối câu lệnh for. Kết quả là danh sách tương tự danh sách squares chúng ta đã thấy ở trên:

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

3.7. Làm việc với một phần của danh sách

Trong các phần trước, ta đã học cách truy cập vào các phần tử đơn trong danh sách và cách duyệt toàn bộ các phần tử trong danh sách. Ở phần này, ta sẽ học cách làm việc với một nhóm cụ thể trong danh sách, được Python gọi là một lát cắt (slice)

3.7.1. Cắt lát một danh sách

Để tạo 1 slice, cần chỉ định chỉ mục đầu tiên và chỉ mục cuối cùng mà ta cần làm việc. Với hàm `range()`, python dừng lại ở vị trí trước index thứ hai được chỉ định. Để in ra 3 phần tử đầu tiên trong danh sách, ta cần chỉ định từ 0 tới 3, Python sẽ trả về 3 phần tử là 0,1,2. Ví dụ sau sẽ gọi một danh sách người chơi trong một đội

```
players = ['charles', 'martina', 'michael', 'florence', 'eli']
print(players[0:3])
```

Đoạn code trên in ra một đoạn danh sách, bao gồm ba phần tử đầu tiên. Output giữ nguyên cấu trúc của danh sách, bao gồm ba phần tử đó.

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

Ta có thể sinh ra bất kỳ tập con nào của danh sách, ví dụ muốn lấy ra phần tử thứ hai, thứ ba và thứ tư của danh sách, ta cần trượt từ chỉ mục 1 (bắt đầu) tới chỉ mục số 4 (kết thúc)

```
players = ['charles', 'martina', 'michael', 'florence', 'eli']
print(players[1:4])
```

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

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

Nếu ta bỏ qua chỉ mục đầu tiên trong một slice, Python sẽ tự động bắt đầu slice ở đầu danh sách:

```
players = ['charles', 'martina', 'michael', 'florence', 'eli']
print(players[:4])
```

Không có chỉ mục bắt đầu, Python bắt đầu với phần tử đầu danh sách
['charles', 'martina', 'michael', 'florence']

Một cú pháp tương tự hoạt động nếu muốn một slice bao gồm phần cuối của danh sách.

Ví dụ: nếu ta muốn tất cả các phần tử từ thứ ba đến cuối cùng, ta có thể bắt đầu với chỉ mục 2 và bỏ qua chỉ mục thứ hai.

```
players = ['charles', 'martina', 'michael', 'florence', 'eli']
print(players[2:])
```

Python trả về tất cả các mục tin từ mục thứ ba đến cuối danh sách:
['michael', 'florence', 'eli']

Cú pháp này cho phép lấy ra tất cả các phần tử từ bất kỳ điểm nào trong danh sách đến cuối bất kể độ dài của danh sách. Cần nhớ rằng một chỉ mục âm trả về một phần tử cách cuối danh sách một khoảng cách nhất định;

Do đó, ta có thể xuất bất kỳ lát nào từ cuối danh sách. Ví dụ, nếu ta muốn xuất ra ba cầu thủ cuối cùng trong danh sách, có thể sử dụng: `players[-3:]`:

```
players = ['charles', 'martina', 'michael', 'florence', 'eli']
print(players[-3:])
```

Việc này in tên của ba người chơi cuối cùng và vẫn chính xác khi danh sách người chơi thay đổi về kích thước.

Ghi chú: Có thể thêm giá trị thứ ba khi xác định một slice. Giá trị này sẽ nói với Python bao nhiêu phần tử sẽ bị bỏ qua giữa các phần tử ở trong slice được chỉ định.

3.7.2. Lặp qua một lát cắt

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

```
players = ['charles', 'martina', 'michael', 'florence', 'eli']
print("Here are the first three players on my team:")
```

```
③ for player in players[:3]:  
    print(player.title())
```

Thay vì lặp tất cả các thành viên, theo ①, python chỉ lặp qua ba thành viên đầu tiên của danh sách players:

Here are the first three players on my team:

```
Charles  
Martina  
Michael
```

Lát cắt rất hữu ích trong một số tình huống. Ví dụ: khi chúng ta tạo trò chơi, ta có thể thêm điểm số cuối cùng của người chơi vào danh sách mỗi khi người chơi kết thúc chơi. Sau đó, ta có thể nhận được ba điểm số cao nhất của một người chơi bằng cách sắp xếp danh sách theo thứ tự giảm dần và lấy một phần chỉ bao gồm ba điểm số đầu tiên. Khi làm việc với dữ liệu, ta có thể sử dụng các lát để xử lý dữ liệu ở dạng khối có kích thước cụ thể. Hoặc, khi đang xây dựng một ứng dụng web, ta có thể sử dụng các lát cắt để hiển thị thông tin trong một loạt các trang với một lượng thông tin thích hợp trên mỗi trang.

3.7.3. Sao chép danh sách

Thông thường, chúng sẽ muốn bắt đầu với một danh sách hiện có và tạo một danh sách hoàn toàn mới dựa trên cái đầu tiên. Hãy cùng khám phá cách hoạt động của việc sao chép danh sách và kiểm tra xác nhận một tình huống mà việc sao chép danh sách là hữu ích.

Để sao chép một danh sách, chúng ta có thể tạo một phần bao gồm toàn bộ danh sách gốc bằng cách bỏ qua chỉ mục đầu tiên và chỉ mục thứ hai ([:]). Điều này cho Python biết tạo một lát cắt bắt đầu từ phần tử đầu tiên và kết thúc bằng 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 loại thực phẩm yêu thích và muốn lập một danh sách riêng các món ăn mà một người ta thích. Người ta này thích mọi thứ trong danh sách của chúng ta, vì vậy ta có thể tạo danh sách của họ bằng cách sao chép của mình:

```
my_foods = ['pizza', 'falafel', 'carrot cake']  
① friend_foods = my_foods[:]  
print("My favorite foods are:")  
print(my_foods)  
print("\nMy friend's favorite foods are:")
```

```
print(friend_foods)
```

Tại dòng đầu tiên, chúng ta tạo ra danh sách món ăn ưa thích của mình là my_foods. Tại ①, chúng ta tạo ra danh sách món ăn ưa thích gọi là friend_foods. Chúng ta tạo một bản sao của my_foods bằng cách yêu cầu một lát cắt của my_foods mà không chỉ định bất kỳ chỉ số nào và lưu trữ bản sao trong friend_foods. Khi chúng ta in từng danh sách, ta thấy rằng cả hai đều chứa các loại thực phẩm giống nhau:

My favorite foods are:

```
['pizza', 'falafel', 'carrot cake']
```

My friend's favorite foods are:

```
['pizza', 'falafel', 'carrot cake']
```

Để chứng minh rằng chúng ta thực sự có hai danh sách riêng biệt, ta sẽ thêm một loại thực phẩm mới vào từng danh sách và cho thấy rằng mỗi danh sách chứa những thức ăn ưa thích của mỗi người:

```
my_foods = ['pizza', 'falafel', 'carrot cake']
① friend_foods = my_foods[:]
② my_foods.append('cannoli')
③ friend_foods.append('ice cream')
print("My favorite foods are:")
print(my_foods)
print("\nMy friend's favorite foods are:")
print(friend_foods)
```

Tại ①, chúng ta sao chép các phần tử gốc từ danh sách my_foods sang danh sách mới là friend_foods như trong ví dụ trước. Tiếp theo, chúng ta thêm mỗi danh sách một món mới: tại ②, chúng ta thêm 'canoli' vào my_foods và thêm 'ice_cream' vào friend_foods tại ③. Sau đó chúng ta in hai danh sách để xem liệu mỗi loại thực phẩm này có trong danh sách tương ứng hay không.

My favorite foods are:

```
④ ['pizza', 'falafel', 'carrot cake', 'cannoli']
```

My friend's favorite foods are:

```
⑤ ['pizza', 'falafel', 'carrot cake', 'ice cream']
```

Kết quả đầu ra tại ④ cho thấy rằng 'cannoli' hiện đã xuất hiện trong danh sách my_foods nhưng 'ice cream' thì không. Tại ⑤, chúng ta có thể thấy 'ice cream' bây giờ xuất hiện trong friend_foods nhưng 'cannoli' thì không. Nếu chúng ta chỉ đặt friend_foods bằng my_foods, chúng ta sẽ không tạo hai danh sách riêng biệt.

Ví dụ, đây là những gì sẽ xảy ra khi ta cố gắng sao chép một danh sách mà không sử dụng một lát cắt:

```
my_foods = ['pizza', 'falafel', 'carrot cake']
# This doesn't work:
① friend_foods = my_foods
my_foods.append('cannoli')
friend_foods.append('ice cream')
print("My favorite foods are:")
print(my_foods)
print("\nMy friend's favorite foods are:")
print(friend_foods)
```

Thay vì lưu trữ bản sao của `my_foods` trong `friend_foods` tại ①, chúng ta thiết lập `friend_foods` bằng `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 như vậy 'ice cream' sẽ xuất hiện trong cả hai danh sách, mặc dù nó dường như chỉ được thêm vào `friend_foods`.

Kết quả đầu ra cho thấy cả hai danh sách đều giống nhau, đó không phải là những gì chúng ta muốn:

```
My favorite foods are:
['pizza', 'falafel', 'carrot cake', 'cannoli', 'ice cream']
My friend's favorite foods are:
['pizza', 'falafel', 'carrot cake', 'cannoli', 'ice cream']
```

Ghi chú: Không cần lo lắng về các chi tiết trong ví dụ này ngay bây giờ. Về cơ bản, nếu ta đang cố gắng làm việc với một bản sao của danh sách và ta thấy hành vi không mong muốn, hãy đảm bảo rằng ta đang sao chép danh sách bằng cách sử dụng một lát cắt, như chúng ta đã làm trong ví dụ đầu tiên.

3.8. Tuples

Danh sách hoạt động tốt để lưu trữ các bộ sưu tập các tin mục có thể thay đổi xuyên suốt vòng đời của một chương trình. Khả năng sửa đổi danh sách là đặc biệt quan trọng khi ta đang làm việc với danh sách người dùng trên trang web hoặc danh sách các ký tự trong một trò chơi. Tuy nhiên, đôi khi chúng ta muốn tạo một danh sách các mục không thể thay đổi. Tuples cho phép ta làm điều đó. Python đề cập đến

các giá trị không thể thay đổi dưới dạng bất biến, và một danh sách không thay đổi được gọi là một tuple.

3.8.1. Định nghĩa một Tuple

Một bộ tuple trông giống như một danh sách ngoại trừ việc ta sử dụng dấu ngoặc đơn thay vì dấu ngoặc vuông. Khi ta xác định một bộ tuple, ta có thể truy cập các phần tử riêng lẻ bằng cách sử dụng chỉ mục của từng mục, giống như cách làm đối với danh sách.

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

```
① dimensions = (200, 50)
② print(dimensions[0])
     print(dimensions[1])
```

Chúng ta định nghĩa một tuple có tên là dimensions tại ① sử dụng ngoặc đơn thay vì ngoặc vuông. Tại ② chúng ta in ra mỗi giá trị của tuple, cùng với cú pháp như đã thực hiện với list:

```
200
50
```

Hãy thử thay đổi giá trị của một phần tử trong Tuple:

```
dimensions = (200, 50)
① dimensions[0] = 250
```

Mã nguồn tại ① cố gắng thay đổi giá trị của kích thước đầu tiên nhưng python trả lại kết quả lỗi. Về cơ bản, vì chúng ta đang cố gắng thay đổi một tuple, không thể thực hiện được với loại đối tượng đó, Python cho chúng ta biết rằng ta không thể chỉ định một giá trị mới cho một phần tử trong tuple:

```
Traceback (most recent call last):
  File "dimensions.py", line 2, in <module>
    dimensions[0] = 250
TypeError: 'tuple' object does not support item assignment
```

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

Ghi chú: Tuples được định nghĩa về mặt kỹ thuật bằng sự hiện diện của dấu phẩy; dấu ngoặc đơn làm cho chúng trông gọn gàng hơn và dễ đọc hơn. Nếu ta muốn xác định một tuple với một phần tử, ta cần bao gồm một dấu phẩy ở cuối:

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

Thường không hợp lý khi xây dựng một bộ tuple với một phần tử, nhưng điều này có thể xảy ra khi các bộ giá trị được tạo tự động.

3.8.2. Lặp qua toàn bộ giá trị trong Tuple

Chúng ta có thể lặp lại tất cả các giá trị trong một bộ bằng cách sử dụng vòng lặp for, giống như đã làm với danh sách:

```
dimensions = (200, 50)
for dimension in dimensions:
    print(dimension)
```

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

```
200
50
```

3.8.3. Ghi lên Tuple

Mặc dù chúng ta không thể sửa đổi tuple, nhưng ta có thể chỉ định một giá trị mới cho một biến đại diện cho một tuple. Vì vậy, nếu chúng ta muốn thay đổi thứ kích cỡ của hình chữ nhật, chúng ta sẽ định nghĩa lại tuple:

```
① dimensions = (200, 50)
print("Original dimensions:")
for dimension in dimensions:
    print(dimension)
② dimensions = (400, 100)
③ print("\nModified dimensions:")
for dimension in dimensions:
    print(dimension)
```

Dòng code tại ① xác định tuple ban đầu và in ra kích cỡ ban đầu. Tại ②, chúng ta gán một tuple mới với biến dimentions. Sau đó ta in ra kích cỡ mới tại ③. Python không đưa ra bất kỳ lỗi nào ở đây, vì việc gán lại một biến là hợp lệ:

```
Original dimensions:
```

```
200
50
```

```
Modified dimensions:
```

```
400
```

Khi so sánh với danh sách, tuple là cấu trúc dữ liệu đơn giản. Sử dụng chúng khi ta muốn lưu trữ một bộ giá trị không được thay đổi trong suốt vòng đời của chương trình.

Bài tập chương 3

3-1. Names: lưu trữ tên của một vài người ta trong danh sách là names. In ra tên mỗi cá nhân bằng cách truy cập mỗi phần tử trong danh sách, mỗi phần tử một lần.

3-2. Greetings: Khởi đầu giống bài tập 3-1 nhưng thay vì in ra mỗi tên của người ta đó thì in ra thông điệp tới họ. Dạng thông điệp có thể giống nhau nhưng mỗi thông điệp dành cho mỗi cá nhân riêng biệt.

3-3. Your Own List: tạo một danh sách các phương tiện giao thông ưa thích và lưu vào trong biến danh sách. In ra một loạt câu về các phương tiện trong danh sách đó. Ví dụ: “I would like to own a Honda motorcycle.”

3-4. Guest List: Lập danh sách bao gồm ít nhất ba người mà ta muốn mời đi ăn tối. Sau đó, sử dụng danh sách để in một thông điệp cho từng người, mời họ đi ăn tối.

3-5. Changing Guest List: Ta vừa nghe nói rằng một trong những khách không thể làm bữa tối, vì vậy ta cần gửi một bộ lời mời mới. Bạn sẽ phải nghĩ đến người khác để mời.

- Bắt đầu với bài tập 3-4. Thêm lệnh print() vào cuối chương trình ghi rõ tên của khách không thể tham gia.

- Thay đổi danh sách, thay thế tên của những khách không thể tham gia với tên của những người mới.

- In ra tập thứ hai các thông điệp mời, mỗi cái dành cho một người còn lại trong danh sách.

3-6. More Guests: Bạn vừa tìm thấy một bàn ăn tối lớn hơn, vì vậy hiện có nhiều không gian hơn. Hãy nghĩ đến việc mời thêm ba vị khách đến ăn tối.

- Bắt đầu với các dữ kiện trong bài tập 3-4 và 3-5. Thêm lệnh print() vào cuối chương trình thông báo cho mọi người là đã tìm được chiếc bàn ăn lớn hơn.

- Sử dụng insert() để thêm một người khách mới vào đầu danh sách
- Sử dụng insert() để thêm một người khách mới vào giữa danh sách
- Sử dụng append() để thêm một người khách mới vào cuối danh sách
- In ra bộ thông điệp mời mới, mỗi cái dành cho một khách có trong danh sách.

3-7. Shrinking Guest List: ta vừa phát hiện ra rằng bàn ăn tối mới sẽ không đến kịp giờ ăn tối, và ta chỉ có chỗ cho hai khách.

- Bắt đầu với chương trình trong bài 3-6. Thêm một dòng in ra thông điệp thông báo ta chỉ có thể mời 2 khách.

- Sử dụng pop() để xóa khách từ danh sách mỗi lần một người cho đến khi chỉ còn hai cái tên trong danh sách. Mỗi khi xóa một tên trong danh sách, in ra thông điệp xin lỗi không thể mời khách đó tới ăn tối.

- In tin nhắn cho từng người trong số hai người vẫn còn trong danh sách của ta, cho phép họ biết họ vẫn được mời

- Sử dụng del để xóa hai cái tên cuối cùng trong danh sách, kết quả là danh sách rỗng. In ra danh sách để đảm bảo ta đang có một danh sách rỗng ở cuối chương trình.

3-8. Seeing the World: Nghĩ về ít nhất 5 địa điểm ta đã từng đi qua
Lưu trữ các địa điểm trong một danh sách. Đảm bảo danh sách không theo thứ tự bảng chữ cái.

In danh sách địa điểm theo thứ tự ban đầu

In danh sách địa điểm theo thứ tự bảng chữ cái được sắp xếp dùng phương thức sorted()

In lại danh sách để thấy danh sách địa điểm vẫn giữ nguyên

Sử dụng phương thức sorted() để in danh sách theo thứ tự ngược của bảng chữ cái

In ra để thấy danh sách vẫn được giữ nguyên

Sử dụng phương thức reverse() để thay đổi thứ tự của danh sách ban đầu. In ra kết quả

Sử dụng phương thức reverse() để đổi về danh sách ban đầu

Sử dụng phương thức sort() để sắp xếp danh sách theo thứ tự bảng chữ cái. In ra danh sách để thấy thứ tự của nó đã thay đổi

Sử dụng phương thức sort() để sắp xếp danh sách ngược bảng chữ cái. In ra để thấy thứ tự của danh sách đã thay đổi.

3-9. Locations: Sử dụng danh sách trong ví dụ 3-8, dùng phương thức len() để in ra số lượng địa điểm trong danh sách.

3-10. Every Function: Khi tất cả mọi thứ có thể lưu trong một danh sách, nghĩ về một loại dữ liệu nào đó có thể lưu trong danh sách như những ngọn núi, những dòng sông, những quốc gia, những thành phố... Viết một chương trình tạo ra danh sách chứa dữ liệu trên và dùng các phương thức kể trên mỗi phương thức ít nhất một lần.

3-11. Pizzas: Hãy nghĩ về ít nhất ba loại bánh pizza yêu thích của ta. Lưu các tên pizza này vào danh sách, sau đó sử dụng vòng lặp for để in tên của từng pizza.

- Thay đổi vòng lặp và in ra cả câu thay vì in ra chỉ mỗi tên của loại pizza ưa thích. Ví dụ, đối với mỗi loại pizza có một câu in ra như: I like pepperoni pizza.

- Thêm một dòng vào cuối chương trình, bên ngoài vòng lặp for, cho biết ta thích pizza đến mức nào. Đầu ra phải bao gồm ba dòng trở lên về loại bánh pizza ta thích và sau đó là một câu bổ sung, chẳng hạn như: I really love pizza!

3-12. Animals: Hãy nghĩ về ít nhất ba loài động vật khác nhau có đặc điểm chung. Lưu tên của những con vật này trong một danh sách, sau đó sử dụng vòng lặp for để in ra tên của mỗi con vật.

- Sửa đổi chương trình để in cả một câu về từng loài động vật, chẳng hạn như: A dog would make a great pet.

- Thêm một dòng vào cuối chương trình nêu rõ những điểm chung của những con vật này. Có thể in một câu chẳng hạn như: Any of these animals would make a great pet!

3-13. Counting to Twenty: Sử dụng vòng lặp for để in các số từ 1 đến 20

3-14. One Million: Lập danh sách các số từ một đến một triệu, sau đó sử dụng vòng lặp for để in các số. (Nếu đầu ra mất quá nhiều thời gian, hãy dừng quá trình đó bằng cách nhấn ctrl-C hoặc bằng cách đóng cửa sổ đầu ra.)

3-15. Summing a Million: Lập danh sách các số từ một đến một triệu, sau đó sử dụng min() và max() để đảm bảo danh sách 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.

3-16. Odd Numbers: 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.

3-17. Threes: Lập danh sách các bội số của 3 từ 3 đến 30. Sử dụng vòng lặp for để in các số trong danh sách của ta.

3-18. Cubes: Một số được nâng lên lũy thừa thứ ba được gọi là một lập phương. Ví dụ: lập phương của 2 được viết là 2^{**3} trong Python. Tạo danh sách 10 số 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 số lập phương.

3-19. Cube Comprehension: sử dụng list comprehension để thực hiện yêu cầu trong bài 3-18.

3-20. Slices: Sử dụng một trong các chương trình ta đã viết trong chương này, thêm một số dòng vào cuối chương trình để thực hiện các thao tác sau:

- In ra: *the first three items in the list are:* . Sau đó sử dụng slice để in ra ba giá trị đầu tiên trong danh sách.

- In ra thông điệp: *Three items from the middle.* Sau đó sử dụng slice để in ra ba giá trị giữa của danh sách.

- In ra thông điệp: *The last three items in the list are:* sau đó sử dụng slice để in ra ba giá trị cuối cùng của danh sách.

3-21. My Pizzas, Your Pizzas: Bắt đầu như bài tập 3-11. Tạo một danh sách sao chép danh sách pizzas và gọi nó là friend_pizzas. Sau đó thực hiện:

- Thêm một pizza mới vào danh sách ban đầu

- Thêm một pizza khác vào danh sách friend_pizza

- Đảm bảo rằng ta có hai danh sách riêng rẽ. In thông điệp: My favorite pizzas are: sau đó dùng vòng lặp for đối với danh sách pizza ban đầu để in ra. In tiếp thông điệp My friend's favorite pizzas are: sau đó dùng vòng lặp for để in ra danh sách thứ hai. Cần đảm bảo mỗi pizza mới được thêm vào danh sách tương ứng.

3-22. More Loops: Tất cả các phiên bản của Foods.py trong phần này đã tránh sử dụng vòng lặp for khi in để tiết kiệm dung lượng. Chọn một phiên bản của Foods.py và viết hai vòng lặp for để in từng danh sách thực phẩm.

3-23. Buffet: Một cửa hàng buffet phục vụ chỉ 5 loại đồ ăn. Thêm 5 loại đồ ăn đơn giản mà cửa hàng phục vụ lưu vào trong một tuple.

- Sử dụng vòng lặp for để in ra các món ăn của cửa hàng
- Có gắng sửa đổi một trong các mục và đảm bảo rằng Python từ chối việc biến đổi.
 - Nhà hàng thay đổi thực đơn của mình, thay thế hai trong số các món bằng các loại thực phẩm khác nhau. Thêm một dòng viết lại bộ tuple, sau đó sử dụng vòng lặp for để in từng mục trên menu đã sửa đổi.

Kết chương

Trong chương này, chúng ta đã học cách làm việc hiệu quả với các phần tử trong danh sách. Chúng ta đã học cách làm việc thông qua danh sách bằng vòng lặp for, cách Python sử dụng thuật lè để cấu trúc một chương trình và cách tránh một số lỗi thuật lè. Ta đã học cách tạo danh sách số đơn giản, cũng như một số thao tác có thể thực hiện trên danh sách số. Ta đã học cách cắt lát danh sách để làm việc với một tập hợp con các phần tử và cách sao chép danh sách đúng cách bằng cách sử dụng lát cắt. Ta cũng đã tìm hiểu về tuple, cung cấp mức độ bảo vệ đến một tập hợp các giá trị không được thay đổi.

Chương tiếp theo, chúng ta học cách phản ứng thích hợp với các điều kiện khác nhau bằng cách sử dụng câu lệnh if. Ta sẽ học cách xâu chuỗi các bộ kiểm tra điều kiện tương đối phức tạp lại với nhau để đáp ứng một cách thích hợp với chính xác loại tình huống hoặc thông tin ta đang tìm kiếm. Ta cũng sẽ học cách sử dụng các câu lệnh trong khi lặp qua danh sách để thực hiện các hành động cụ thể với các phần tử từ một danh sách.

CHƯƠNG 4. CÂU LỆNH RẼ NHÁNH (IF)

Lập trình thường bao gồm việc kiểm tra một tập hợp các điều kiện và quyết định hành động cần thực hiện dựa trên những điều kiện đó. Câu lệnh if của Python cho phép kiểm tra trạng thái hiện tại của một chương trình và phản hồi một cách thích hợp.

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

4.1. Ví dụ

Ví dụ ngắn sau đây cho thấy cách thức nếu các bài kiểm tra cho phép phản hồi tình huống một cách chính xác. Hãy tưởng tượng ta có một danh sách ô tô và muốn in ra tên của từng chiếc xe. Tên ô tô là tên riêng, vì vậy tên của hầu hết các ô tô nên được in kiểu Title (chữ cái đầu viết hoa). Tuy nhiên, với giá trị 'bmw' thì cần được in ra hoàn toàn chữ hoa. Đoạn mã sau đây lặp lại danh sách tên xe và tìm kiếm giá trị 'bmw'. Bất cứ khi nào giá trị là 'bmw', nó sẽ được in bằng chữ hoa thay vì chữ hoa tiêu đề:

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

Vòng lặp trong ví dụ này đầu tiên kiểm tra nếu giá trị hiện thời của biến *car* là 'bmw' thì sẽ in hoa toàn bộ. Nếu giá trị hiện thời của biến *car* là bất cứ giá trị nào ngoài 'bmw' thì sẽ in ra kiểu Title

```
Audi
BMW
Subaru
Toyota
```

Ví dụ trên kết hợp một số khái niệm ta sẽ học trong phần này. Hãy bắt đầu bằng cách xem xét các loại kiểm tra ta có thể sử dụng kiểm tra các điều kiện trong chương trình.

4.2. Kiểm tra có điều kiện

Ở trung tâm của mọi câu lệnh if là biểu thức mà có thể được đánh giá là True hoặc False và nó được gọi là kiểm tra có điều kiện. Python sử dụng các giá trị True và False để quyết định xem mã trong câu lệnh if có được thực thi hay không. Nếu một kiểm tra có điều kiện đánh giá là True, Python thực thi mã sau câu lệnh if. Nếu kiểm tra đánh giá là False, Python sẽ bỏ qua mã sau câu lệnh if.

4.2.1. Kiểm tra đăng thíc

Hầu hết các kiểm tra có điều kiện đều so sánh giá trị hiện tại của một biến với một giá trị cụ thể. Câu kiểm tra có điều kiện đơn giản nhất là kiểm tra giá trị của một biến có bằng một giá trị cụ thể nào không.

Ví dụ:

```
>>> car = 'bmw'  
>>> car == 'bmw'  
True
```

Dòng đầu tiên thực hiện việc gán giá trị của car thành ‘bmw’ sử dụng dấu bằng ‘=’ như trong chương 2 đã đề cập. Dòng thứ hai kiểm tra xem giá trị của biến car có bằng ‘bmw’ hay không sử dụng hai dấu bằng ‘==’. Toán tử bằng trả về giá trị True nếu giá trị bên trái và bên phải bằng nhau và trả về giá trị False nếu ngược lại. Ở trong ví dụ trên, giá trị của bên phải và bên trái toán tử bằng nhau nên Python trả về True. Khi giá trị của biến car là giá trị bất kỳ khác ‘bmw’, phần kiểm tra sẽ trả về False:

```
>>> car = 'audi'  
>>> car == 'bmw'  
False
```

Trong ví dụ trên, dấu bằng ‘=’ chính là một câu, thực hiện lệnh gán. Trong khi đó, dấu bằng kép ‘==’ lại là khiến nó trở thành câu hỏi: “Liệu giá trị của biến car có bằng ‘bmw’ hay không?”. Hầu hết các ngôn ngữ lập trình sử dụng dấu bằng theo cách như vậy.

4.2.2. Bỏ qua chữ viết hoa khi kiểm tra đăng nhập

Kiểm tra đăng nhập là trường hợp phân biệt chữ hoa và chữ thường trong Python. Ví dụ, hai giá trị với chữ viết hoa khác nhau không được coi là bằng nhau

```
>>> car = 'Audi'  
>>> car == 'audi'  
False
```

Nếu trường hợp quan trọng về chữ hoa chữ thường, so sánh này là thuận lợi. Nhưng nếu trường hợp không quan trọng và thay vào đó, chúng ta chỉ muốn kiểm tra giá trị của một biến, ta có thể chuyển đổi giá trị của biến thành chữ thường trước khi thực hiện so sánh:

```
>>> car = 'Audi'  
>>> car.lower() == 'audi'  
True
```

Phản kiểm tra này trả về giá trị True ngay với bất kỳ định dạng nào của giá trị ‘Audi’ bởi vì nó sẽ chuyển về trường hợp so sánh chữ thường. Hàm lower() không thay đổi giá trị ban đầu được lưu trong biến car, do đó chúng ta có thể thực hiện loại so sánh này mà không làm ảnh hưởng tới giá trị ban đầu của biến.

```
>>> car = 'Audi'  
>>> car.lower() == 'audi'  
True  
>>> car  
'Audi'
```

Tại dòng đầu tiên, chúng ta gán giá trị ‘Audi’ cho biến car.

Tại dòng thứ hai, chúng ta biến các ký tự trong biến car thành chữ thường và so sánh giá trị này với chuỗi ‘audi’. Hai chuỗi khớp với nhau, do đó Python trả về True.

Tại dòng thứ ba, chúng ta có thể nhìn thấy giá trị được lưu trong biến car đã không bị ảnh hưởng bởi phương thức lower().

Các trang web thực thi các quy tắc nhất định đối với dữ liệu mà người dùng nhập vào cách tương tự như trên. Ví dụ: một trang web có thể sử dụng kiểm tra có điều kiện như vậy để đảm bảo rằng mọi người dùng đều có một tên người dùng thực sự duy nhất, chứ không phải là một biến thể về cách viết hoa tên người dùng của người khác. Khi ai đó đăng ký tên người dùng mới, tên người dùng mới đó sẽ được

chuyển đổi thành chữ thường và so với các phiên bản viết thường của tất cả các tên người dùng hiện có. Trong quá trình kiểm tra này, tên người dùng như 'John' sẽ bị từ chối nếu có bất kỳ biến thể nào của 'john' đã được sử dụng.

4.2.3. Kiểm tra bất đẳng thức

Khi chúng ta muốn kiểm tra hai giá trị không bằng nhau, ta kết hợp dấu chấm than với dấu bằng (\neq). Dấu chấm than có nghĩa là không(not) và được sử dụng trong nhiều ngôn ngữ lập trình.

Hãy sử dụng một câu lệnh if khác để xem cách kiểm tra bất đẳng thức (toán tử khác nhau). Chúng ta lưu yêu cầu loại pizza trong một biến và in ra thông báo khi người dùng không đặt hàng loại anchovies:

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

Dòng code thứ hai so sánh giá trị của biến requested_topping với chuỗi 'anchovies'. Nếu hai giá trị đó không khớp nhau, Python sẽ trả về True và thực thi câu lệnh sau lệnh rẽ nhánh if. Nếu hai giá trị khớp nhau, Python trả về False và không thực hiện câu lệnh sau lệnh rẽ nhánh if.

Do giá trị của biến requested_topping không phải là 'anchovies' nên hàm print() được thực thi và kết quả in ra là:

Hold the anchovies!

Hầu hết các kiểm tra có điều kiện là kiểm tra đẳng thức (toán tử bằng nhau), tuy nhiên trong một số trường hợp, sẽ hiệu quả hơn khi sử dụng khi kiểm tra bất đẳng thức (toán tử khác nhau).

4.2.4. So sánh số

Kiểm tra các giá trị số khá đơn giản. Ví dụ: mã nguồn sau đây kiểm tra xem một người đã đủ 18 tuổi hay chưa:

```
>>> age = 18  
>>> age == 18  
True
```

Chúng ta có thể kiểm tra nếu hai số không bằng nhau. Ví dụ đoạn mã nguồn sau đây sẽ in ra thông điệp nếu câu trả lời là không chính xác:

```
answer = 17
```

```
if answer != 42:  
    print("That is not the correct answer. Please try again!")
```

Kiểm tra có điều kiện tại mệnh đề if là đúng, do giá trị biến answer là 17 khác với 42. Do kiểm tra điều kiện được thông qua, khối mã sau mệnh đề if được thực thi

```
That is not the correct answer. Please try again!
```

Chúng ta có thể bao gồm các phép so sánh toán học khác nhau trong các câu lệnh, chẳng hạn như nhỏ hơn, nhỏ hơn hoặc bằng, lớn hơn, và lớn hơn hoặc bằng:

```
>>> age = 19  
>>> age < 21  
True  
>>> age <= 21  
True  
>>> age > 21  
False  
>>> age >= 21  
False
```

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

4.2.5. Kiểm tra nhiều điều kiện

Chúng ta có thể muốn kiểm tra nhiều điều kiện cùng một lúc. Ví dụ, đôi khi ta có thể cần hai điều kiện là True để thực hiện một hành động. Một lúc khác, ta có thể được thỏa mãn chỉ với một điều kiện là True. Các từ khóa *and* và *or* có thể giúp trong những tình huống này.

Sử dụng And để kiểm tra đa điều kiện:

Để kiểm tra xem hai điều kiện có đồng thời là True hay không, hãy sử dụng từ khóa *and* để kết hợp hai kiểm tra điều kiện; nếu mỗi lần kiểm tra vượt qua, biểu thức tổng thể sẽ đánh giá là True. Nếu một trong hai kiểm tra là False hoặc nếu cả hai đều là False, biểu thức đánh giá là False.

```
①    >>> age_0 = 22  
>>> age_1 = 18  
②    >>> age_0 >= 21 and age_1 >= 21  
False  
  
③    >>> age_1 = 22  
>>> age_0 >= 21 and age_1 >= 21
```

True

Tại ① chúng ta định nghĩa 2 biến tuổi, age_0 và age_1. Tại ② chúng ta kiểm tra cả hai tuổi cùng lớn hơn 21 hay không. Kiểm tra bên về trái thì qua nhưng kiểm tra bên về phải thì thất bại, do đó kết quả tổng hợp của bài kiểm tra này là False. Ở ③, khi chúng ta thay đổi age_1 thành 22. Giá trị của age_1 hiện lớn hơn 21, vì vậy cả hai bài kiểm tra riêng lẻ đều vượt qua, khiến cho biểu thức điều kiện tổng thể được đánh giá là True.

Để dễ đọc hơn, ta có thể sử dụng dấu ngoặc đơn xung quanh biểu thức kiểm tra, nhưng chúng không bắt buộc. Nếu ta sử dụng dấu ngoặc đơn, bài kiểm tra trên sẽ trông như sau:

```
(age_0 >= 21) and (age_1 >= 21)
```

Sử dụng Or để kiểm tra nhiều điều kiện

Từ khóa Or cho phép chúng ta kiểm tra nhiều điều kiện, nhưng nó vượt qua khi một trong hai hoặc cả hai bài kiểm tra riêng lẻ vượt qua. Một biểu thức Or chỉ thất bại khi cả hai bài kiểm tra riêng lẻ đều thất bại.

Chúng ta hãy xem ví dụ trên về hai tuổi lần nữa, nhưng lần này chúng ta sẽ chỉ tìm kiếm một người trên 21 tuổi:

```
①    >>> age_0 = 22  
>>> age_1 = 18  
②    >>> age_0 >= 21 or age_1 >= 21  
True  
③    >>> age_0 = 18  
>>> age_0 >= 21 or age_1 >= 21  
False
```

Tại ①, chúng ta gán hai giá trị của biến age_0 và age_1. Do kiểm tra điều kiện cho age_0 tại ② vượt qua nên cả biểu thức trả về True. Sau đó, chúng ta gán lại age_0 thành 18. Tại biểu thức kiểm tra ở ③, cả hai vé đều không đạt nên cả biểu thức nhận giá trị False.

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

Đôi khi, điều quan trọng là phải kiểm tra xem danh sách có chứa một giá trị nhất định hay không trước khi thực hiện một hành động. Ví dụ: ta có thể muốn kiểm tra xem tên người dùng mới đã tồn tại trong danh sách tên người dùng hiện tại trước

khi hoàn tất đăng ký của ai đó trên trang web. Trong một dự án lập bản đồ, ta có thể muốn kiểm tra xem một vị trí đã gửi đã tồn tại trong danh sách các vị trí đã biết hay chưa.

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 số mã ta có thể viết cho một tiệm bánh pizza. Chúng ta sẽ lập danh sách các lớp phủ mà khách hàng đã yêu cầu cho một chiếc bánh pizza và sau đó kiểm tra xem các lớp phủ nhất định có trong danh sách hay không.

```
>>> requested_toppings = ['mushrooms', 'onions', 'pineapple']
①   >>> 'mushrooms' in requested_toppings
True
②   >>> 'pepperoni' in requested_toppings
False
```

Tại ① và ②, từ khóa `in` yêu cầu Python kiểm tra sự tồn tại của 'mushrooms' và 'pepperoni' trong danh sách `requested_toppings`. Kỹ thuật này là khá mạnh mẽ vì chúng ta có thể tạo danh sách các giá trị thiết yếu và sau đó dễ dàng kiểm tra xem giá trị ta đang kiểm tra có khớp với một trong các giá trị trong danh sách hay không.

Kiểm tra một giá trị không nằm trong danh sách

Trường hợp khác, điều quan trọng là phải biết nếu một giá trị không xuất hiện trong danh sách. Ta có thể sử dụng từ khóa `not in` trong tình huống này. Ví dụ: hãy xem xét danh sách người dùng bị cấm bình luận trong một diễn đàn. Ta có thể kiểm tra xem người dùng đã bị cấm trước khi cho phép người đó gửi nhận xét:

```
banned_users = ['andrew', 'carolina', 'david']
user = 'marie'
①   if user not in banned_users:
    print(f"{user.title()}, you can post a response if you wish.")
```

Dòng mã tại ① khá rõ ràng. Nếu giá trị của người dùng không có trong danh sách người dùng bị cấm (`banned_users`), Python trả về `True` và thực thi dòng lệnh. Người dùng 'marie' không nằm trong danh sách người dùng bị cấm, vì vậy cô ấy thấy một thông báo mời cô ấy đăng phản hồi:

Marie, you can post a response if you wish.

4.3. Câu lệnh If

Khi hiểu các câu kiểm tra điều kiện, ta có thể bắt đầu viết câu lệnh if. Một số loại câu lệnh if khác nhau tồn tại và sự lựa chọn sử dụng phụ thuộc vào số lượng điều kiện ta cần kiểm tra. Ta 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 tìm hiểu sâu hơn về chủ đề này.

4.3.1. Câu lệnh if đơn giản

Loại câu lệnh if đơn giản nhất có một kiểm tra và một hành động:

```
if conditional_test:  
    do something
```

Chúng ta có thể đặt bất kỳ kiểm tra có điều kiện nào ở dòng đầu tiên và bất kỳ hành động trong khối thực lè sau mệnh đề kiểm tra. Nếu kiểm tra có điều kiện đánh giá là True, Python thực thi mã sau câu lệnh if. Nếu kiểm tra đánh giá là False, Python sẽ bỏ qua mã sau câu lệnh if.

Giả sử chúng ta có một biến đại diện cho tuổi của một người và chúng ta muốn biết người đó có đủ tuổi bỏ phiếu hay không. Đoạn mã sau kiểm tra xem người có thể bỏ phiếu:

```
age = 19  
① if age >= 18:  
②     print("You are old enough to vote!")
```

Tại ①, python kiểm tra giá trị của biến age lớn hơn hoặc bằng 18 hay không. Nếu có, python thực thi câu lệnh thực lè tại ②:

```
You are old enough to vote!
```

Thực lè đóng vai trò tương tự trong câu lệnh if như trong vòng lặp for.

Tất cả các dòng thực lè sau câu lệnh if sẽ được thực thi nếu quá trình kiểm tra vượt qua, và toàn bộ khối các dòng thực lè sẽ bị bỏ qua nếu kiểm tra không vượt qua.

Chúng ta có thể có bao nhiêu dòng mã tùy thích trong khối theo sau câu lệnh if. Đoạn sau thêm một dòng đầu ra khác nếu người đó lớn tuổi đủ để bình chọn, hỏi xem cá nhân đã đăng ký bình chọn chưa:

```
age = 19  
if age >= 18:  
    print("You are old enough to vote!")  
    print("Have you registered to vote yet?")
```

Kiểm tra có điều kiện vượt qua và cả hai lệnh gọi print() đều được thực hiện, vì vậy cả hai dòng được in:

```
You are old enough to vote!  
Have you registered to vote yet?
```

Nếu giá trị của biến *age* nhỏ hơn 18, chương trình này sẽ không in ra đầu ra nào cả.

4.3.2. Câu lệnh if-else

Thông thường, ta sẽ muốn thực hiện một hành động khi kiểm tra có điều kiện vượt qua và một hành động khác trong tất cả các trường hợp khác. Cú pháp if-else của Python giúp điều này trở nên khả thi.

Một khối if-else tương tự như một câu lệnh if đơn giản, nhưng câu lệnh else cho phép ta xác định một hành động hoặc một tập hợp các hành động được thực thi khi kiểm tra điều kiện không thành công.

Chúng ta sẽ hiển thị cùng một thông báo mà chúng ta đã có trước đây nếu người đó lớn tuổi đủ để bỏ phiếu, nhưng lần này chúng ta sẽ thêm thông báo cho bất kỳ ai không đủ tuổi để bỏ phiếu:

```
age = 17  
① if age >= 18:  
    print("You are old enough to vote!")  
    print("Have you registered to vote yet?")  
② else:  
    print("Sorry, you are too young to vote.")  
    print("Please register to vote as soon as you turn 18!")
```

Tại ① kiểm tra xem biến *age* có lớn hơn hoặc bằng 18 tuổi hay không. Nếu kiểm tra này thông qua, các câu lệnh print() sẽ được thực hiện. Nếu kiểm tra trả về giá trị False, khối lệnh else tại ② được thực hiện. Vì biến *age* có giá trị nhỏ hơn 18 nên kiểm tra có điều kiện không vượt qua và khối lệnh tại else được thực hiện:

```
Sorry, you are too young to vote.  
Please register to vote as soon as you turn 18!
```

Mã này hoạt động vì nó chỉ có hai tình huống có thể xảy ra để đánh giá: một người đủ tuổi bầu cử hoặc không đủ tuổi bầu cử. Cấu trúc if-else hoạt động tốt trong các tình huống mà ta 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 khối hành động sẽ luôn được thực thi.

4.3.3. Chuỗi if-elif-else

Thông thường, chúng ta sẽ cần kiểm tra nhiều hơn hai tình huống có thể xảy ra và để đánh giá ta 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 kiểm tra có điều kiện theo thứ tự cho đến khi một lần đi qua. Khi một kiểm tra vượt qua, mã sau kiểm tra có điều kiện đó được thực thi và Python bỏ qua phần còn lại của các bài kiểm tra.

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

- Miễn phí vé vào cửa cho bất kỳ ai dưới 4 tuổi.
- Vé vào cửa cho bất kỳ ai trong độ tuổi từ 4 đến 18 là \$ 25.
- Vé vào cửa cho bất kỳ ai từ 18 tuổi trở lên là \$ 40.

Làm cách nào chúng ta có thể sử dụng câu lệnh if để xác định phí vào công viên của một người? Đoạn mã sau đây kiểm tra nhóm tuổi của một người và sau đó in thông báo giá vé vào cửa:

```
age = 12  
① if age < 4:  
    print("Your admission cost is $0.")  
② elif age < 18:  
    print("Your admission cost is $25.")  
③ else:  
    print("Your admission cost is $40.")
```

Câu lệnh if ở ① kiểm tra xem một người dưới 4 tuổi. Nếu kiểm tra vượt qua, một thông báo thích hợp được in ra và Python bỏ qua phần còn lại của các bài kiểm tra. Dòng elif tại ② thực sự 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 đó ít nhất 4 tuổi vì lần kiểm tra đầu tiên không thành công. Nếu người đó dưới 18 tuổi, một thông báo thích hợp sẽ được in và Python bỏ qua khối khác. Nếu cả hai kiểm tra if và elif không thành công, Python chạy mã trong khối khác tại ③.

Trong trường hợp này, kiểm tra tại ① trả về giá trị False, do đó khối mã trong đó không thực hiện. Tuy nhiên, tại kiểm tra điều kiện thứ hai, giá trị kiểm tra là True (do $12 < 18$) do đó khối mã thụt lè tại đây được thực hiện. Đầu ra là một câu thông báo về giá vé tương ứng:

```
Your admission cost is $25.
```

Bất kỳ độ tuổi nào lớn hơn 17 sẽ khiến hai bài kiểm tra đầu tiên không thành công. Trong những tình huống đó, khối khác sẽ được thực hiện và giá vào công viên sẽ là \$ 40.

Thay vì in giá 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() đơn giản chạy sau khi chuỗi đã được đã đánh giá:

```
age = 12
if age < 4:
    ①     price = 0
elif age < 18:
    ②     price = 25
else:
    ③     price = 40
④     print(f"Your admission cost is ${price}.")
```

Các dòng tại ①② và ③ thiết lập giá trị của giá vé vào cửa theo giá trị của biến age. Sau khi giá được xác định bởi chuỗi if-elif-else, một câu lệnh print() không được thụt lè được gọi tại ④ sử dụng giá trị này để hiển thị thông báo về vé vào cửa của người dùng.

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

4.3.4. Sử dụng nhiều khối elif

Chúng ta có thể sử dụng bao nhiêu khối elif trong mã tùy thích. Ví dụ, nếu công viên giải trí đã thực hiện giảm giá cho người cao tuổi, ta có thể thêm một bài kiểm tra có điều kiện nữa đối với mã để xác định xem ai đó có đủ điều kiện nhận

chiết khấu cao cấp hay không. Giả sử rằng bất kỳ ai 65 tuổi trở lên trả một nửa số tiền vào cửa thông thường, hoặc \$ 20:

```
age = 12
if age < 4:
    price = 0
elif age < 18:
    price = 25
① elif age < 65:
    price = 40
② else:
    price = 20
print(f"Your admission cost is ${price}.")
```

Hầu hết mã này là không thay đổi. Khối elif thứ hai ở ① hiện đang kiểm tra để đảm bảo một người dưới 65 tuổi trước khi chỉ định rõ cho họ giá vào cửa là \$ 40. Lưu ý rằng giá trị được gán trong khối else tại ② cần được đổi thành \$ 20, vì độ tuổi duy nhất lọt vào khối này là những người từ 65 tuổi trở lên.

4.3.5. Bỏ qua khối else

Python không yêu cầu một khối else ở cuối chuỗi if-elif. Đôi khi một khối else hữu ích; đôi khi việc sử dụng một khối elif khác khiến nó rõ ràng hơn:

```
age = 12
if age < 4:
    price = 0
elif age < 18:
    price = 25
elif age < 65:
    price = 40
① elif age >= 65:
    price = 20
print(f"Your admission cost is ${price}.")
```

Khối elif bỏ sung ở ① áp định giá 20 đô la khi người đó 65 tuổi hoặc già hơn, rõ ràng hơn một chút so với khối else. Với sự thay đổi này, mọi khối mã phải vượt qua một bài kiểm tra riêng biệt để được thực thi.

Khối else là một câu lệnh catchall. Nó phù hợp với bất kỳ điều kiện nào không được khớp bởi một kiểm tra if hoặc elif cụ thể và đôi khi có thể bao gồm dữ liệu không hợp lệ hoặc thậm chí độc hại. Nếu chúng ta có một điều kiện cuối cùng cụ thể, xem xét sử dụng khối elif cuối cùng và bỏ qua khối else.

4.3.6. Kiểm tra nhiều điều kiện

Chuỗi if-elif-else rất mạnh, nhưng nó chỉ thích hợp để sử dụng khi chỉ cần một bài kiểm tra để vượt qua. Ngay sau khi Python kết thúc một bài kiểm tra có điều kiện, nó bỏ qua phần còn lại của các bài kiểm tra. Hành vi này có lợi, vì nó có hiệu quả và cho phép ta kiểm tra một điều kiện cụ thể.

Tuy nhiên, đôi khi điều quan trọng là phải kiểm tra tất cả các điều kiện của quan tâm. Trong trường hợp này, ta nên sử dụng một loạt các câu lệnh if đơn giản với không elif hoặc các khối khác. Kỹ thuật này có ý nghĩa khi nhiều điều kiện có thể là True và ta muốn thực hiện với mọi điều kiện là True.

Hãy xem xét lại ví dụ về tiệm bánh pizza. Nếu ai đó yêu cầu một hai lớp phủ pizza, ta sẽ cần đảm bảo bao gồm cả hai lớp phủ trên bánh pizza của họ. Nếu đúng, một thông báo sẽ được in tương ứng với lớp phủ đó.

```
① requested_toppings = ['mushrooms', 'extra cheese']
② if 'mushrooms' in requested_toppings:
    print("Adding mushrooms.")
③ if 'pepperoni' in requested_toppings:
    print("Adding pepperoni.")
④ if 'extra cheese' in requested_toppings:
    print("Adding extra cheese.")
print("\nFinished making your pizza!")
```

Chúng ta bắt đầu ở ① với một danh sách các lớp phủ pizza được yêu cầu. Câu lệnh if tại ② kiểm tra xem khách hàng có yêu cầu lớp phủ 'mushrooms' trên bánh của họ. Các kiểm tra có 'pepperoni' trong yêu cầu không tại ③ là một câu lệnh if đơn khác mà không trong khối elif hoặc else, do đó, nó chạy mà không cần biết điều kiện kiểm tra trước có qua hay không qua. Kiểm tra điều kiện tại ④ xem 'extra cheese' có trong yêu cầu của khách hàng không mà không phụ thuộc vào kết quả kiểm tra của hai kiểm tra có điều kiện ở trên. Ba câu kiểm tra có điều kiện độc lập này đều chạy mỗi khi chương trình chạy. Do vậy, kết quả là 'mushrooms' và 'extra cheese' được thêm vào bánh pizza.

Adding mushrooms.

Adding extra cheese.

Finished making your pizza!

Mã này sẽ không hoạt động đúng nếu chúng ta sử dụng khối if-elif-else, bởi vì mã sẽ ngừng chạy sau khi chỉ có một kiểm tra thành công. Mã chạy và kết quả như dưới đây:

```
requested_toppings = ['mushrooms', 'extra cheese']
if 'mushrooms' in requested_toppings:
    print("Adding mushrooms.")
elif 'pepperoni' in requested_toppings:
    print("Adding pepperoni.")
elif 'extra cheese' in requested_toppings:
    print("Adding extra cheese.")
print("\nFinished making your pizza!")
```

Bài kiểm tra 'mushrooms' là bài kiểm tra đầu tiên vượt qua, vì vậy 'mushrooms' được thêm vào bánh pizza. Tuy nhiên, các giá trị 'thêm pho mát' và 'pepperoni' không bao giờ được kiểm tra, bởi vì Python không chạy bất kỳ kiểm tra nào ngoài kiểm tra đầu tiên chuyển trong một chuỗi if-elif-else. Lớp phủ đầu tiên của khách hàng sẽ được thêm vào, nhưng tất cả các lớp phủ khác của chúng sẽ bị bỏ qua:

Adding mushrooms.

Finished making your pizza!

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

4.4. Câu lệnh If với danh sách

Ta có thể thực hiện một số công việc thú vị khi kết hợp danh sách và câu lệnh if. Ta có thể xem các giá trị đặc biệt cần được xử lý khác hơn so với các giá trị khác trong danh sách.

4.4.1. Kiểm tra các phần tử đặc biệt

Chương này bắt đầu với một ví dụ đơn giản cho thấy cách xử lý một giá trị đặc biệt như 'bmw', giá trị này cần được in ở định dạng khác với các giá trị khác trong danh sách. Bây giờ ta đã có hiểu biết cơ bản về các bài kiểm tra có điều kiện và câu lệnh if, hãy xem xét kỹ hơn cách ta có thể xem cho các giá trị đặc biệt trong danh sách và xử lý các giá trị đó một cách thích hợp.

Tiếp tục với ví dụ về tiệm bánh pizza. Cửa hàng bán bánh pizza hiển thị một thông báo bất cứ khi nào lớp phủ trên được thêm vào bánh pizza, vì nó đang được làm. Mã cho hành động này có thể được viết rất hiệu quả bằng cách lập một danh

sách các lớp phủ khách hàng đã yêu cầu và sử dụng một vòng lặp để thông báo từng lớp phủ khi thêm vào bánh pizza:

```
requested_toppings = ['mushrooms', 'green peppers', 'extra cheese']
for requested_topping in requested_toppings:
    print(f"Adding {requested_topping}.")
print("\nFinished making your pizza!")
```

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

Adding mushrooms.

Adding green peppers.

Adding extra cheese.

Finished making your pizza!

Nhưng nếu cửa hàng bán bánh pizza hết ót 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 = ['mushrooms', 'green peppers', 'extra cheese']
for requested_topping in requested_toppings:
    ① if requested_topping == 'green peppers':
        print("Sorry, we are out of green peppers right now.")
    ② else:
        print(f"Adding {requested_topping}.")
print("\nFinished making your pizza!")
```

Lần này ta kiểm tra từng món được yêu cầu trước khi thêm vào bánh pizza. Mã tại ① kiểm tra xem liệu người đó có yêu cầu ót xanh hay không. Nếu có, chúng ta hiển thị thông báo cho họ biết lý do tại sao họ không thể có ót xanh. Khối else tại ② đảm bảo rằng tất cả các lớp phủ khác sẽ được thêm vào pizza.

Kết quả cho thấy rằng mỗi lớp phủ được yêu cầu được xử lý một cách thích hợp.

Adding mushrooms.

Sorry, we are out of green peppers right now.

Adding extra cheese.

Finished making your pizza!

4.4.2. Kiểm tra một danh sách không rỗng

Chúng ta đã đưa ra một giả định đơn giản về mọi danh sách mà chúng ta đã làm việc từ trước đến nay; chúng ta đã giả định rằng mỗi danh sách có ít nhất một mục trong đó. 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 cho rằng danh sách có bất kỳ mục nào trong đó mỗi khi vòng lặp được chạy. Trong tình huống này, có ích khi kiểm tra xem danh sách có trống hay không trước khi chạy vòng lặp for.

Ví dụ: hãy kiểm tra xem danh sách các lớp phủ được yêu cầu có trống trước khi xây dựng bánh pizza. Nếu danh sách trống, chúng ta sẽ nhắc người dùng và đảm bảo rằng họ muốn có một chiếc bánh pizza đơn giản. Nếu danh sách không trống, chúng ta sẽ xây dựng bánh pizza giống như chúng ta đã làm trong các ví dụ trước:

```
① requested_toppings = []
② if requested_toppings:
    for requested_topping in requested_toppings:
        print(f"Adding {requested_topping}.")
    print("\nFinished making your pizza!")
③ else:
    print("Are you sure you want a plain pizza?")
```

Tại ví dụ này, chúng ta bắt đầu với danh sách lớp phủ rỗng tại ①. Thay vì nhảy ngay vào vòng lặp for, chúng ta kiểm tra nhanh tại ②. Khi tên của danh sách được sử dụng trong câu lệnh if, Python trả về True nếu danh sách chứa ít nhất một mục; một danh sách trống sẽ đánh giá là False. Nếu vượt qua bài kiểm tra có điều kiện, chúng ta chạy cùng một vòng lặp for mà chúng ta đã sử dụng trong phần thí dụ trước. Nếu kiểm tra có điều kiện không thành công, chúng ta sẽ in thông báo yêu cầu khách hàng nếu họ thực sự muốn một chiếc bánh pizza đơn giản không có lớp phủ bên ngoài tại ③.

Danh sách trống trong trường hợp này, vì vậy đầu ra hỏi người dùng có thực sự muốn một chiếc bánh pizza đơn giản:

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

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

4.4.3. Sử dụng nhiều danh sách

Mọi người sẽ hỏi bất cứ điều gì, đặc biệt là khi nói đến lớp phủ trên pizza. Điều gì sẽ xảy ra nếu một khách hàng thực sự muốn khoai tây chiên trên bánh pizza của họ? Ta có thể sử dụng danh sách và câu lệnh if để đảm bảo thông tin đầu vào của ta có ý nghĩa trước khi có một hành động nào đó.

Hãy để ý những yêu cầu bắt thường về lớp phủ trước khi chúng ta chế tạo một chiếc bánh pizza. Ví dụ sau xác định hai danh sách. Đầu tiên là danh sách các lớp phủ có sẵn tại tiệm bánh pizza và thứ hai là danh sách các lớp phủ mà người dùng có yêu cầu. Lần này, mỗi mục trong `request_toppings` được kiểm tra dựa trên danh sách các lớp phủ có sẵn trước khi thêm vào bánh pizza:

```
① available_toppings = ['mushrooms', 'olives', 'green peppers',
'pepperoni', 'pineapple', 'extra cheese']
② requested_toppings = ['mushrooms', 'french fries', 'extra cheese']
③ for requested_topping in requested_toppings:
④     if requested_topping in available_toppings:
            print(f"Adding {requested_topping}.")
⑤ else:
    print(f"Sorry, we don't have {requested_topping}.")
print("\nFinished making your pizza!")
```

Tại ①, chúng ta định nghĩa danh sách các lớp phủ có sẵn trong tiệm pizza. Ghi nhớ rằng cái này có thể là tuple nếu tiệm bánh có tập cố định các lớp phủ. Tại ②, chúng ta lập danh sách các lớp phủ mà người dùng có yêu cầu, trong đó có một yêu cầu khác thường là 'french fries'. Tại ③, chúng ta thực hiện vòng lặp qua các lớp phủ được yêu cầu. Bên trong vòng lặp, đầu tiên chúng ta kiểm tra nếu lớp phủ được yêu cầu có trong danh sách lớp phủ có sẵn tại ④. Nếu có sẵn, chúng ta thêm lớp phủ đó vào bánh pizza. Nếu lớp phủ đó không có trong cửa hàng, khối else sẽ chạy tại ⑤. Khối else sẽ in ra lớp phủ nào không có sẵn tại cửa hàng.

Cú pháp mã này tạo ra đầu ra rõ ràng, đầy đủ thông tin:

Adding mushrooms.

Sorry, we don't have french fries.

Adding extra cheese.

Finished making your pizza!

Chỉ trong một vài dòng mã, chúng ta đã quản lý một tình huống trong thế giới thực khá hiệu quả.

Bài tập chương 4

4-1. Conditional Tests: Viết một loạt các bài kiểm tra điều kiện. In ra câu mô tả mỗi bài kiểm tra và dự đoán của ta cho kết quả của mỗi bài kiểm tra. Mã nguồn sẽ trông giống như sau:

```

car = 'subaru'
print("Is car == 'subaru'? I predict True.")
print(car == 'subaru')
print("\nIs car == 'audi'? I predict False.")
print(car == 'audi')

```

- Xem xét kỹ kết quả của ta và đảm bảo rằng hiểu lý do tại sao mỗi dòng đánh giá True hoặc False.

- Tạo ra ít nhất 10 kiểm tra có điều kiện và có ít nhất 5 cái là True, ít nhất 5 cái là False.

4-2. More Conditional Tests: tạo tệp conditional_tests.py và thêm vào các kiểm tra có điều kiện:

- Kiểm tra đằng thức và bất đằng thức với chuỗi (string).
- Kiểm tra sử dụng phương thức lower().
- Kiểm tra có điều kiện với số trong các trường hợp: đằng thức, bất đằng thức, lớn hơn, nhỏ hơn, lớn hơn hoặc bằng, nhỏ hơn hoặc bằng.
- Kiểm tra có điều kiện sử dụng từ khóa *and* hoặc *or*.
- Kiểm tra xem một phần tử có trong danh sách không.
- Kiểm tra xem một phần tử không có trong danh sách.

4-3. Alien Colors #1: Tưởng tượng con quái vật bị bắn trong một trò chơi. Tạo ra một biến gọi là *alien_color* và gán vào các giá trị 'green', 'yellow', or 'red'.

- Viết câu lệnh if kiểm tra màu sắc của quái vật. Nếu như bắn hạ quái vật màu xanh thì in ra thông điệp người chơi kiếm được 5 điểm.
- Viết một phiên bản của chương trình này vượt qua kiểm tra if và một phiên bản khác không thành công. (Phiên bản không thành công sẽ không có đầu ra.)

4-4. Alien Colors #2: Chọn một màu cho quái vật trong ví dụ 5-3 và viết chuỗi if-else:

- Nếu bắn được quái vật màu xanh, người chơi kiếm được 5 điểm
- Nếu bắn được quái vật màu khác, người chơi kiếm được 10 điểm.
- Viết một phiên bản của chương trình này chạy khỏi if và một phiên bản khác chạy khỏi else.

4-5. Alien Colors #3: thay chuỗi if-else trong ví dụ 5-4 thành chuỗi if-elif-else:

- Nếu quái vật màu xanh, in ra thông báo người chơi kiếm được 5 điểm
- Nếu quái vật màu vàng, in ra thông báo người chơi kiếm được 10 điểm
- Nếu quái vật màu đỏ, in ra thông báo người chơi kiếm được 15 điểm
- Viết ba phiên bản của chương trình này, đảm bảo mỗi thông báo được in cho quái vật có màu sắc thích hợp.

4-6. Stages of Life: Viết chuỗi if-elif-else xác định trạng thái cuộc đời của mỗi con người. Thiết lập một giá trị cho biến *age* sau đó:

- Nếu tuổi nhỏ hơn 2, gọi là baby
- Nếu tuổi từ 2 đến 4 gọi là toddler
- Nếu tuổi từ 4 đến 13 gọi là kid
- Nếu tuổi từ 13 đến 20 gọi là teenager
- Nếu tuổi từ 20 đến 65 gọi là adult
- Nếu tuổi 65 trở lên gọi là elder

4-7. Favorite Fruit: Lập danh sách các loại trái cây yêu thích, sau đó viết một loạt các câu lệnh if độc lập để kiểm tra các loại trái cây nhất định trong danh sách.

- Tạo danh sách với ba loại trái cây gọi là *favorite_fruits*
- Viết 5 câu lệnh if, mỗi câu lệnh kiểm tra xem một loại trái cây nào đó có trong danh sách không. Nếu có thì in ra thông báo như: *You really like bananas!*

4-8. Hello Admin: Lập danh sách gồm nhiều tên người dùng, bao gồm tên 'quản trị viên'. Hãy tưởng tượng ta đang viết mã sẽ in lời chào cho mỗi người dùng sau khi họ đăng nhập vào một trang web. Lặp lại danh sách và in lời chào cho mỗi người dùng:

- Nếu tên người dùng là 'quản trị viên', hãy in lời chào đặc biệt, chẳng hạn như Xin chào quản trị viên, ta có muốn xem báo cáo trạng thái không?
- Nếu không, hãy in một lời chào chung chung, chẳng hạn như Xin chào Jaden, cảm ơn vì đăng nhập lại.

4-9. No Users: Thêm kiểm tra if vào hello_admin.py để đảm bảo danh sách người dùng là không rỗng.

- Nếu danh sách là rỗng, in ra thông báo: *We need to find some users!*
- Xóa tất cả tên người dùng khỏi danh sách và đảm bảo rằng thông báo chính xác được in.

4-10. Checking Usernames: Thực hiện như sau để tạo một chương trình mô phỏng cách các trang web đảm bảo rằng mọi người đều có một tên người dùng duy nhất.

- Tạo danh sách gồm nhiều hoặc nhiều tên người dùng được gọi là `current_users`.
- Tạo một danh sách tên người dùng khác được gọi là `new_users`. Đảm bảo một hoặc hai tên người dùng mới cũng có trong danh sách `current_users`.
- Lặp danh sách `new_users` để xem từng tên người dùng mới đã được sử dụng chưa. Nếu có, hãy in thông báo rằng người đó sẽ cần nhập tên người dùng mới. Nếu tên người dùng chưa được sử dụng, hãy in thông báo rằng tên người dùng đó khả dụng.
- Đảm bảo rằng so sánh của ta không phân biệt chữ hoa chữ thường. Nếu 'John' đã được sử dụng, thì 'JOHN' sẽ không được chấp nhận. (Để làm điều này, ta cần tạo một bản sao của `current_users` chứa các phiên bản viết thường của tất cả người dùng hiện có.)

4-11. Ordinal Numbers: Các số thứ tự cho biết vị trí của chúng trong danh sách, chẳng hạn như như 1 hoặc 2. Hầu hết các số thứ tự kết thúc bằng thứ, ngoại trừ 1, 2 và 3.

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

Kết chương

Trong chương này, chúng ta đã học cách viết các bài kiểm tra có điều kiện, luôn luôn đánh giá True hay False. ta đã học cách viết các câu lệnh if đơn giản, chuỗi if-else và chuỗi if-elif-else. Ta đã bắt đầu sử dụng các cấu trúc này để xác định các điều kiện cụ thể cần để kiểm tra và biết khi nào các điều kiện đó đã được đáp ứng trong các chương trình của mình. Ta đã học cách xử lý các phần tử nhất định trong danh sách khác với tất cả các mục khác trong khi tiếp tục sử dụng hiệu quả của vòng lặp for.

Trong chương tiếp theo, ta sẽ tìm hiểu về từ điển của Python. Từ điển tương tự như một danh sách, nhưng nó cho phép ta kết nối các phần thông tin. Ta sẽ học cách xây dựng từ điển, lặp qua chúng và sử dụng chúng kết hợp với danh sách và câu lệnh if. Tìm hiểu về từ điển sẽ cho phép chúng ta có thể mô hình hóa nhiều tình huống trong thế giới thực hơn nữa.

CHƯƠNG 5. TỪ ĐIỂN (DICTIONARIES)

Trong chương này, chúng ta sẽ học cách sử dụng từ điển của Python, cho phép kết nối các phần thông tin liên quan. Chúng ta sẽ học cách truy cập thông tin khi nó có trong từ điển và cách sửa đổi thông tin đó.

Bởi vì từ điển có thể lưu trữ gần như vô hạn lượng thông tin, ta học cách lặp lại thông qua dữ liệu trong từ điển. Ngoài ra, chúng ta sẽ học cách lồng các từ điển bên trong danh sách, danh sách bên trong từ điển và thậm chí cả các từ điển bên trong từ điển.

Hiểu từ điển cho phép lập mô hình nhiều đối tượng thế giới thực chính xác hơn. Ta có thể tạo từ điển đại diện cho một người và sau đó lưu trữ bao nhiêu thông tin mong muốn về người đó. Chúng ta có thể lưu trữ tên, tuổi, vị trí, nghề nghiệp của họ và bất kỳ khía cạnh nào khác của con người.

Chúng ta có thể lưu trữ bất kỳ hai loại thông tin có thể ánh xạ với nhau, chẳng hạn như danh sách các từ và ý nghĩa của chúng, danh sách tên của mọi người và sở yêu thích của họ, danh sách các ngọn núi và độ cao của chúng, v.v.

5.1. Ví dụ đơn giản về từ điển

Giả sử một trò chơi có các quái vật (aliens) có màu sắc và có các điểm số khác nhau. Một từ điển đơn giản lưu thông tin về một quái vật cụ thể:

```
alien_0={'color':'green', 'point':5}
print(alien_0['color'])
print(alien_0['point'])
```

Từ điển alien_0 lưu giá trị màu sắc và điểm của quái vật, 2 dòng code dưới truy cập và hiển thị thông tin đó và được in ra:

```
green
5
[Finished in 0.2s]
```

Với hầu hết các khái niệm lập trình mới, việc sử dụng từ điển cần thực hành. Sau khi đã làm việc quen với từ điển, chúng ta sẽ thấy chúng có thể mô hình hóa các tình huống trong thế giới thực một cách hiệu quả.

5.2. Làm việc với từ điển

Một từ điển trong Python là một tập hợp các cặp khoá-giá trị (key-value). Mỗi khoá được kết nối với một giá trị và chúng ta có thể sử dụng một khoá để truy cập đến giá trị được liên kết với khóa. Giá trị của khoá có thể là một số, một chuỗi, một danh sách hoặc thậm chí một từ điển khác. Trên thực tế, chúng ta có thể sử dụng bất kỳ đối tượng nào có thể tạo bằng Python làm giá trị trong từ điển.

Trong Python, một từ điển được đặt trong dấu ngoặc nhọn, {} , với một loạt các cặp khoá-giá trị bên trong dấu ngoặc nhọn:

```
alien_0 = {'color': 'green', 'points': 5}
```

Cặp khoá-giá trị là một tập hợp các giá trị được liên kết với nhau. Khi chúng ta cung cấp một khóa, Python sẽ trả về giá trị được liên kết với khóa đó. Mọi khóa được kết nối với giá trị của nó bằng dấu hai chấm (:) và các cặp khoá-giá trị riêng lẻ được tách riêng bằng dấu phẩy. Chúng ta có thể lưu trữ nhiều cặp khoá-giá trị tùy thích trong từ điển.

Từ điển đơn giản nhất có chính xác một cặp khoá-giá trị:

```
alien_0 = {'color': 'green'}
```

Từ điển này lưu trữ một phần thông tin về alien_0, cụ thể là màu của alien. Chuỗi 'color' là một khóa và các liên kết với giá trị là 'green'.

5.2.1. Truy cập các giá trị trong từ điển

Để lấy giá trị được liên kết với một khóa, cần đặt tên từ điển và sau đó đặt khóa bên trong một tập hợp các dấu ngoặc vuông:

```
alien_0 = {'color': 'green'}
print(alien_0['color'])
```

Lệnh này trả về giá trị được liên kết với khóa 'color' từ từ điển alien_0:

Green

Chúng ta có thể có số lượng cặp khoá-giá trị không giới hạn trong một từ điển.

Ví dụ:

```
alien_0 = {'color': 'green', 'points': 5}
```

Bây giờ chúng ta có thể truy cập màu hoặc giá trị điểm của alien_0. Nếu người chơi bắn hạ alien này, ta có thể xem điểm bằng cách:

```
alien_0 = {'color': 'green', 'points': 5}
```

```
new_points = alien_0['points']
print(f"You just earned {new_points} points!")
```

Kết quả:

```
You just earned 5 points!
```

Nếu chúng ta chạy code này mỗi khi alien bị bắn hạ, giá trị điểm của alien sẽ được truy xuất.

5.2.2. Thêm các cặp khóa-giá trị mới

Từ điển là cấu trúc động và chúng ta có thể thêm các cặp giá trị khóa mới vào từ điển bất cứ lúc nào. Ví dụ: để thêm một cặp giá trị-khóa mới, ta sẽ cung cấp tên của từ điển, sau đó là khóa mới trong ô vuông dấu ngoặc cùng với giá trị mới.

Cùng nhau thêm hai thông tin vào từ điển của alien_0: hai tọa độ x và y của quái vật, giúp hiển thị quái vật tại một vị trí nhất định trên màn hình. Ví dụ, cần định vị alien ở phía bên trái màn hình và có khoảng cách 25 pixels từ trên xuống. Để làm được như vậy, chúng ta đặt giá trị tọa độ x là 0 và tọa độ y là dương 25, như dưới đây:

```
alien_0['x_position'] = 0
alien_0['y_position'] = 25
print(alien_0)
```

Chúng ta bắt đầu bằng cách định nghĩa cùng một từ điển mà chúng ta đã làm việc. Sau đó chúng ta in thông tin của từ điển ra. Tại dòng đầu tiên, chúng ta thêm từ điển một cặp từ khóa-giá trị như sau: khóa: x_position, giá trị là 0. Chúng ta làm tương tự tại dòng thứ hai, thêm khóa y_position. Khi in từ điển ra, chúng ta thấy hai cặp khóa giá trị đã được thêm vào:

```
{'color': 'green', 'points': 5}
{'color': 'green', 'points': 5, 'y_position': 25, 'x_position': 0}
```

Phiên bản sau của từ điển có bốn cặp khóa-giá trị. Hai cặp đầu chỉ màu sắc và điểm của quái vật, hai cặp sau chỉ vị trí của con quái vật.

5.2.3. Bắt đầu với từ điển rỗng

Đôi khi chúng ta sẽ bắt đầu với một từ điển rỗng rồi sau đó mới thêm cặp khóa-giá trị vào từ điển. Để làm việc đó, chúng ta bắt đầu một từ điển với cặp ngoặc nhọn ({}) rỗng và sau đó thêm cặp giá trị-khóa vào từ điển trong một dòng mã riêng. Ví dụ dưới đây thể hiện điều đó:

```

alien_0 = {}
alien_0['color'] = 'green'
alien_0['points'] = 5
print(alien_0)

```

Ở đây chúng ta định nghĩa một từ điển alien_0 rỗng, sau đó thêm màu và thêm giá trị cho nó. Kết quả như sau:

```
{'color': 'green', 'points': 5}
```

Thông thường, ta sẽ sử dụng từ điển trống khi lưu trữ dữ liệu do người dùng cung cấp trong từ điển hoặc khi viết mã tạo ra một số lượng lớn các cặp key-value tự động.

5.2.4. Sửa giá trị trong từ điển

Để sửa giá trị trong từ điển, chúng ta thiết lập tên từ điển với giá trị của khóa trong ngoặc vuông, sau đó tới giá trị của khóa mà ta muốn gán cho khóa đó. Ví dụ, khi một con quái vật đổi màu từ màu xanh sang màu vàng:

```

alien_0 = {'color': 'green'}
print(f"The alien is {alien_0['color']}.")

alien_0['color'] = 'yellow'
print(f"The alien is now {alien_0['color']}.")

```

Chúng ta đầu tiên thiết lập quái vật alien_0 chỉ chứa thông tin về màu sắc, sau đó ta thay đổi giá trị màu của khóa ‘color’ sang ‘yellow’. Phần in ra màn hình cho thấy thực sự con quái vật đã chuyển từ màu xanh sang vàng.

The alien is green.

The alien is now yellow.

Để có một ví dụ thú vị hơn, hãy theo dõi vị trí của con quái vật có thể di chuyển với các tốc độ khác nhau. Chúng ta sẽ lưu trữ một giá trị đại diện cho tốc độ hiện tại của quái vật và sau đó sử dụng nó để xác định xem alien sẽ di chuyển bao xa về bên phải:

```

alien_0 = {'x_position': 0, 'y_position': 25, 'speed': 'medium'}
print(f"Original position: {alien_0['x_position']}")

# Move the alien to the right.

# Determine how far to move the alien based on its current speed.

① if alien_0['speed'] == 'slow':
    x_increment = 1
elif alien_0['speed'] == 'medium':
    x_increment = 2
else:

```

```

# This must be a fast alien.
x_increment = 3
# The new position is the old position plus the increment.
② alien_0['x_position'] = alien_0['x_position'] + x_increment
print(f"New position: {alien_0['x_position']}")

```

Chúng ta định nghĩa con quái vật và thiết lập ban đầu với trục tọa độ x, tọa độ y và tốc độ của quái vật là trung bình. Chúng ta đã bỏ qua các giá trị màu và điểm vì mục đích đơn giản hóa, nhưng ví dụ này sẽ hoạt động theo cách tương tự nếu ta vẫn bao gồm các cặp giá trị đó. Ta cũng in giá trị ban đầu của x_position để xem alien di chuyển sang phải bao xa.

Tại ①, một chuỗi if-elif-else xác định khoảng bao xa quái vật sẽ di chuyển sang phía phải và gán giá trị cho biến x_increment. Nếu tốc độ quái vật là slow-chậm, nó di chuyển 1 đơn vị sang bên phải, nếu tốc độ là trung bình-medium, nó di chuyển 2 đơn vị và nếu tốc độ là nhanh-fast, nó sẽ di chuyển 3 đơn vị sang phải.

Mỗi khi độ dịch được tính, nó sẽ thêm giá trị của x_position tại câu lệnh ② và giá trị được lưu tại x_position của từ điển. Bởi vì đây là một quái vật có tốc độ trung bình-medium, vị trí của nó dịch chuyển hai đơn vị sang bên phải:

```

Original x-position: 0
New x-position: 2

```

Điểm tốt của kỹ thuật này: bằng cách thay đổi một giá trị của từ điển thông tin quái vật, chúng ta có thể thay đổi toàn bộ hành vi của con thú. Ví dụ, để biến tốc độ của con quái vật từ trung bình thành nhanh, chúng ta chỉ cần thêm dòng code:

```
alien_0['speed'] = 'fast'
```

Sau đó, khỏi if-elif-else sẽ gán giá trị lớn hơn cho x_increment vào lần mã chạy tiếp theo.

5.2.5. Xóa các cặp khóa-giá trị

Khi chúng ta không còn cần một phần thông tin được lưu trữ trong một từ điển nữa, ta có thể sử dụng câu lệnh del để xóa hoàn toàn một cặp khóa-giá trị. Tất cả những gì del cần là tên của từ điển và khóa mà ta muốn xóa.

Ví dụ: hãy xóa các khoá 'points' khỏi từ điển alien_0 cùng với giá trị của nó:

```

alien_0 = {'color': 'green', 'points': 5}
print(alien_0)
del alien_0['points']

```

```
print(alien_0)
```

Dòng lệnh del yêu cầu Python xóa các khóa ‘points’ khỏi từ điển alien_0 và xóa cả giá trị được liên kết với khóa đó. Kết quả đầu ra cho thấy rằng các khóa ‘points’ và giá trị 5 của nó sẽ bị xóa khỏi từ điển, nhưng phần còn lại của từ điển không bị ảnh hưởng:

```
{'color': 'green', 'points': 5}  
{'color': 'green'}
```

5.2.6. Từ điển của các đối tượng tương tự

Ví dụ trên liên quan đến việc lưu trữ các loại thông tin khác nhau về một đối tượng, một con quái vật trong một trò chơi. Chúng ta cũng có thể sử dụng từ điển để lưu trữ một loại thông tin về nhiều đối tượng. Ví dụ, giả sử ta muốn thăm dò ý kiến một số người và hỏi họ ngôn ngữ lập trình yêu thích của họ là gì. Từ điển rất hữu ích để lưu trữ kết quả của một cuộc thăm dò đơn giản, như sau:

```
favorite_languages = {  
    'jen': 'python',  
    'sarah': 'c',  
    'edward': 'ruby',  
    'phil': 'python',  
}
```

Như chúng ta có thể thấy, ta đã chia một từ điển lớn hơn thành nhiều dòng. Mỗi khóa là tên của một người đã trả lời cuộc thăm dò và mỗi giá trị là sự lựa chọn ngôn ngữ của họ. Khi ta biết mình sẽ cần nhiều hơn một dòng để xác định từ điển, hãy nhấn enter sau dấu ngoặc nhọn mở. Sau đó, thụt lề dòng tiếp theo một mức (bốn dấu cách) và viết cặp giá trị khóa đầu tiên, theo sau là dấu phẩy. Từ thời điểm này trở đi khi ta nhấn enter, trình soạn thảo văn bản sẽ tự động thụt lề tất cả các cặp khoá-giá trị tiếp theo để khớp với cặp khoá-giá trị đầu tiên.

Sau khi ta xác định xong từ điển, cần thêm dấu ngoặc nhọn trên một dòng mới sau cặp khoá-giá trị cuối cùng và thụt lề xuống một cấp để nó căn chỉnh với các phím trong từ điển. Chúng ta cũng nên thêm dấu phẩy sau cặp khoá-giá trị cuối cùng, vì như vậy là sẵn sàng thêm một cặp khoá-giá trị mới trên dòng tiếp theo.

Để sử dụng từ điển này, với tên của một người đã tham gia cuộc thăm dò, ta có thể dễ dàng tra cứu ngôn ngữ yêu thích của họ:

```
① language = favorite_languages['sarah'].title()  
print(f"Sarah's favorite language is {language}.")
```

Để xem Sarah đã chọn ngôn ngữ nào, chúng ta yêu cầu giá trị tại:

```
favorite_languages['sarah']
```

Chúng ta sử dụng cấu trúc này để lấy ra ngôn ngữ yêu thích của Sarah từ từ điển tại ① và gán nó vào biến language. Tạo ra một biến mới tại đó làm cho mọi thứ rõ ràng hơn. Cửa sổ ra sẽ hiển thị kết quả:

```
Sarah's favorite language is C.
```

Chúng ta có thể sử dụng cùng một cú pháp này với bất kỳ người nào được trình bày trong từ điển.

5.2.7. Sử dụng get() để truy cập các giá trị

Việc sử dụng các khóa trong dấu ngoặc vuông để truy xuất giá trị mà chúng ta quan tâm từ từ điển có thể gây ra một vấn đề tiềm ẩn: nếu khóa ta yêu cầu không tồn tại, chương trình sẽ gặp lỗi.

Hãy xem điều gì sẽ xảy ra khi ta yêu cầu giá trị điểm của một alien không có bộ giá trị điểm:

```
alien_0 = {'color': 'green', 'speed': 'slow'}
print(alien_0['points'])
```

Điều này dẫn đến việc truy xuất lại, hiển thị KeyError:

```
Traceback (most recent call last):
  File "alien_no_points.py", line 2, in <module>
    print(alien_0['points'])
KeyError: 'points'
```

Đối với từ điển, ta có thể sử dụng phương thức get() để đặt giá trị mặc định sẽ được trả về nếu khóa được yêu cầu không tồn tại.

Phương thức get() yêu cầu một khóa làm đối số đầu tiên. Là đối số tùy chọn thứ hai, ta có thể chuyển giá trị được trả về nếu khóa không tồn tại:

```
alien_0 = {'color': 'green', 'speed': 'slow'}
```

```
point_value = alien_0.get('points', 'No point value assigned. ')
print(point_value)
```

Nếu khoá 'points' tồn tại trong từ điển, ta sẽ nhận được giá trị nhập tương ứng. Nếu không, ta sẽ nhận được giá trị mặc định. Trong trường hợp này, points không tồn tại và chúng ta nhận được một thông báo rõ ràng thay vì lỗi:

```
No point value assigned
```

Nếu có khả năng khóa ta đang yêu cầu có thể không tồn tại, hãy xem xét sử dụng phương thức get() thay vì ký hiệu dấu ngoặc vuông.

5.3. Lặp qua toàn bộ từ điển

Một từ điển Python có thể chỉ chứa một vài cặp khoá-giá trị hoặc hàng triệu cặp. Vì từ điển có thể chứa một lượng lớn dữ liệu, Python cho phép chúng ta lặp qua từ điển. Từ điển có thể được sử dụng để lưu trữ thông tin theo nhiều cách khác nhau; do đó, tồn tại một số cách khác nhau để lặp lại chúng. Ta có thể lặp lại tất cả các cặp giá trị khóa của từ điển, thông qua các khóa của từ điển hoặc thông qua các giá trị của nó.

5.3.1. Lặp qua tất cả các cặp khóa-giá trị

Trước khi chúng ta khám phá các phương pháp tiếp cận vòng lặp khác nhau, hãy xem xét một từ điển mới được thiết kế để lưu trữ thông tin về người dùng trên một trang web. Từ điển sau sẽ lưu trữ username, first name, and last name:

```
user_0 = {  
    'username': 'efermi',  
    'first': 'enrico',  
    'last': 'fermi',  
}
```

Chúng ta có thể truy cập bất kỳ thông tin đơn lẻ nào về user_0 dựa trên những gì ta đã học trong chương này. Nhưng nếu ta muốn xem mọi thứ được lưu trữ trong từ điển của người dùng này thì sao? Để làm như vậy, ta có thể lặp qua từ điển bằng vòng lặp for:

```
user_0 = {  
    'username': 'efermi',  
    'first': 'enrico',  
    'last': 'fermi',  
}  
  
① for key, value in user_0.items():  
    ② print(f"\nKey: {key}")  
    ③ print(f"Value: {value}")
```

Tại ①, để viết một vòng lặp for cho một từ điển, chúng ta tạo ra tên cho hai biến mà sẽ giữ giá trị của key và value của mỗi cặp khoá-giá trị. Ta có thể chọn cặp

tên bất kỳ cho hai biến này. Đoạn mã cũng sẽ hoạt động tốt nếu chúng ta sử dụng từ viết tắt cho các tên biến như sau:

```
for k, v in user_0.items()
```

Nửa sau của vòng lặp for tại ① bao gồm tên của từ điển cung cấp bởi phương thức items(), trả về danh sách các cặp khóa-giá trị. Vòng lặp for do đó gán những cặp này cho hai biến được khai báo ở trước đó. Ở ví dụ trên, chúng ta dùng biến để in ra mỗi khóa ②, sau đó là tới giá trị ③. “\n” tại dòng đầu in đầu tiên tạo ra dòng trống trước các cặp khóa-giá trị ở đầu ra.

Key: last

Value: fermi

Key: first

Value: enrico

Key: username

Value: efermi

Việc lặp lại tất cả các cặp khoá-giá trị hoạt động hiệu quả đối với các từ điển như ví dụ trên, nơi lưu trữ cùng một loại thông tin cho nhiều khóa khác nhau. Nếu ta lặp qua từ điển favorite_languages, ta sẽ nhận được tên của từng người trong từ điển và ngôn ngữ lập trình yêu thích của họ. Vì các khóa luôn đề cập đến tên của một người và giá trị luôn là một ngôn ngữ, chúng ta sẽ sử dụng biến name và language trong vòng lặp thay vì khóa và giá trị. Điều này sẽ giúp ta dễ dàng theo dõi những gì đang xảy ra bên trong vòng lặp:

```
favorite_languages = {
    'jen': 'python',
    'sarah': 'c',
    'edward': 'ruby',
    'phil': 'python',
}
①for name, language in favorite_languages.items():
②print(f"{name.title()}\'s favorite language is {language.title()}.")
```

Vòng lặp for yêu cầu Python lặp qua từng cặp khoá-giá trị trong từ điển. Khi nó lặp qua từng cặp, khóa được gán cho biến name và giá trị được gán cho biến language. Các tên mô tả này giúp ta dễ dàng xem lệnh print() đang làm gì.

Bây giờ, chỉ trong một vài dòng code, chúng ta có thể hiển thị tất cả thông tin từ việc thăm dò sở thích ngôn ngữ:

```
Jen's favorite language is Python.  
Sarah's favorite language is C.  
Edward's favorite language is Ruby.  
Phil's favorite language is Python.
```

Loại vòng lặp này vẫn sẽ hoạt động tốt nếu từ điển lưu trữ kết quả từ việc thăm dò ý kiến của một nghìn hoặc thậm chí một triệu người.

5.3.2. Lặp qua tất cả các khóa trong từ điển

Phương thức key() hữu ích khi ta không cần phải làm việc với tất cả các giá trị trong từ điển. Hãy xem qua từ điển favourite_languages và in tên của những người đã tham gia cuộc thăm dò ý kiến:

```
favorite_languages = {  
    'jen': 'python',  
    'sarah': 'c',  
    'edward': 'ruby',  
    'phil': 'python',  
}  
  
for name in favorite_languages.keys():  
    print(name.title())
```

Vòng lặp for yêu cầu Python lấy ra cả các khóa từ từ điển favourite_languages và gán chúng lần lượt cho biến name. Kết quả hiển thị tên của tất cả những người đã tham gia cuộc thăm dò:

```
Jen  
Sarah  
Edward  
Phil
```

Lặp qua các khóa thực sự là hành vi mặc định khi lặp qua từ điển, vì vậy code này sẽ có cùng đầu ra nếu ta đã viết:

```
for name in favorite_languages:
```

Cũng như là:

```
for name in favorite_languages.keys():
```

Chúng ta có thể chọn sử dụng phương thức key() một cách rõ ràng nếu nó làm cho code dễ đọc hơn hoặc có thể bỏ qua nếu muốn.

Chúng ta có thể truy cập giá trị được liên kết với bất kỳ khóa nào ta quan tâm bên trong vòng lặp bằng cách sử dụng khóa hiện tại. Hãy in thông điệp cho một vài

người ta về ngôn ngữ họ đã chọn. Chúng ta sẽ lặp lại các tên trong ví dụ như ta đã làm trước đây, nhưng khi tên đó khớp với một trong những người ta của chúng ta, ta sẽ hiển thị thông báo về ngôn ngữ yêu thích của họ:

```
favorite_languages = {  
    --snip--  
}
```

```
①   friends = ['phil', 'sarah']  
for name in favorite_languages.keys():  
    print(name.title())  
  
②   if name in friends:  
    ③       language = favorite_languages[name].title()  
        print(f"\t{name.title()}, I see you love {language}!")
```

Tại ① chúng ta tạo ra một danh sách friends để in ra thông điệp. Trong vòng lặp, chúng ta in ra các tên. Sau đó tại ②, chúng ta kiểm tra xem tên có nằm trong danh sách friends không. Nếu nó nằm trong danh sách, chúng ta xác định ngôn ngữ yêu thích của người đó bằng việc sử dụng khóa tên của từ điển và giá trị của khóa đó tại ③. Sau đó chúng ta in ra thông điệp chào mừng, bao gồm cả ngôn ngữ họ ưa thích.

Tất cả tên người sẽ được in ra, nhưng chỉ có ai trong danh sách friends mới được in ra ngôn ngữ yêu thích:

```
Hi Jen.  
Hi Sarah.  
    Sarah, I see you love C!  
Hi Edward.  
Hi Phil.  
    Phil, I see you love Python!
```

Chúng ta cũng có thể sử dụng phương thức key() để tìm hiểu xem một người cụ thể có được thăm dò ý kiến hay không. Lần này, hãy cùng tìm hiểu xem Erin có tham gia cuộc thăm dò hay không:

```
favorite_languages = {  
    'jen': 'python',  
    'sarah': 'c',  
    'edward': 'ruby',  
    'phil': 'python',
```

```

}
if 'erin' not in favorite_languages.keys():
    print("Erin, please take our poll!")

```

Phương thức key() không chỉ để lặp lại: nó thực sự trả về một danh sách tất cả các khóa và dòng if 'erin' not in favorite_languages.keys(): chỉ đơn giản là kiểm tra xem 'erin' có trong danh sách này hay không. Bởi vì cô ấy không tham gia nên một thông báo được in ra mời cô ấy tham gia cuộc thăm dò:

`Erin, please take our poll!`

5.3.3. Lặp các khóa của từ điển theo một thứ tự cụ thể

Việc lặp từ điển trả về các mục theo cùng thứ tự mà chúng đã được chèn vào. Tuy nhiên, đôi khi, chúng ta muốn xem lại từ điển theo một thứ tự khác.

Một cách để làm điều này là sắp xếp các khóa khi chúng được trả về trong vòng lặp for. Ta có thể sử dụng hàm sorted() để lấy bản sao của các khóa theo thứ tự:

```

favorite_languages = {
    'jen': 'python',
    'sarah': 'c',
    'edward': 'ruby',
    'phil': 'python',
}
for name in sorted(favorite_languages.keys()):
    print(f"{name.title()}, thank you for taking the poll.")

```

Câu lệnh for này giống như các câu lệnh for khác ngoại trừ việc chúng ta đã gói hàm sorted() xung quanh phương thức dictionary.keys(). Điều này yêu cầu Python liệt kê tất cả các khóa trong từ điển và sắp xếp danh sách đó trước khi lặp qua nó. Kết quả hiển thị tất cả những người đã tham gia cuộc thăm dò, với các tên được hiển thị theo thứ tự:

```

Edward, thank you for taking the poll.
Jen, thank you for taking the poll.
Phil, thank you for taking the poll.
Sarah, thank you for taking the poll.

```

5.3.4. Lặp qua tất cả các giá trị trong từ điển

Nếu chúng ta chủ yếu quan tâm đến các giá trị mà từ điển chứa, ta có thể sử dụng phương thức values() để trả về danh sách các giá trị mà không có bất kỳ khóa nào. Ví dụ: giả sử chúng ta chỉ muốn có một danh sách tất cả các ngôn ngữ được chọn

trong cuộc thăm dò ngôn ngữ lập trình mà không có tên của người đã chọn từng ngôn ngữ:

```
favorite_languages = {  
    'jen': 'python',  
    'sarah': 'c',  
    'edward': 'ruby',  
    'phil': 'python',  
}  
  
print("The following languages have been mentioned:")  
for language in favorite_languages.values():  
    print(language.title())
```

Câu lệnh for ở đây lấy từng giá trị từ từ điển và gán nó vào biến language. Khi các giá trị này được in, chúng ta nhận được danh sách tất cả các ngôn ngữ đã chọn:

```
The following languages have been mentioned:  
Python  
C  
Python  
Ruby
```

Cách tiếp cận này lấy tất cả các giá trị từ từ điển mà không cần kiểm tra các lần lặp lại. Điều đó có thể hoạt động tốt với một số lượng nhỏ giá trị, nhưng trong một cuộc thăm dò với một số lượng lớn người trả lời, điều này sẽ dẫn đến một danh sách lặp lại nhiều lần. Để xem từng ngôn ngữ được chọn mà không bị lặp lại, chúng ta có thể sử dụng một tập hợp. Set là một tập hợp trong đó mỗi mục phải là duy nhất:

```
favorite_languages = {  
    'jen': 'python',  
    'sarah': 'c',  
    'edward': 'ruby',  
    'phil': 'python',  
}  
  
print("The following languages have been mentioned:")  
for language in set(favorite_languages.values()):  
    print(language.title())
```

Khi bọc set() xung quanh danh sách có chứa các mục trùng lặp, Python sẽ xác định các mục duy nhất trong danh sách và xây dựng một tập hợp từ các mục đó. Tại vòng lặp for sử dụng set() để lấy ra các ngôn ngữ duy nhất trong favourite_languages.values().

Kết quả là một danh sách các ngôn ngữ được đề cập:

The following languages have been mentioned:

Python
C
Ruby

Khi chúng ta tiếp tục tìm hiểu về Python, ta thường sẽ tìm thấy một tính năng tích hợp sẵn của ngôn ngữ giúp ta thực hiện chính xác những gì mong muốn với dữ liệu của mình.

5.4. Nesting

Đôi khi chúng ta muốn lưu trữ nhiều từ điển trong một danh sách hoặc một danh sách các mục dưới dạng một giá trị trong từ điển. Điều này được gọi là nesting. Chúng ta có thể lồng các từ điển vào bên trong một danh sách, một danh sách các mục bên trong một từ điển hoặc thậm chí một từ điển bên trong một từ điển khác. Nesting là một tính năng mạnh mẽ, như các ví dụ sau đây sẽ chứng minh điều đó.

5.4.1. Danh sách các từ điển

Từ điển alien_0 chứa nhiều thông tin về một alien, nhưng nó không có chỗ để lưu thông tin về alien thứ hai, ít hơn nhiều là một màn hình đầy alien. Làm thế nào ta có thể quản lý một đội alien? Một cách là lập danh sách alien, trong đó mỗi alien là một từ điển thông tin về alien đó. Ví dụ: đoạn code sau xây dựng danh sách ba alien:

```
alien_0 = {'color': 'green', 'points': 5}
alien_1 = {'color': 'yellow', 'points': 10}
alien_2 = {'color': 'red', 'points': 15}
```

① aliens = [alien_0, alien_1, alien_2]

```
for alien in aliens:
    print(alien)
```

Đầu tiên, chúng ta tạo ba từ điển, mỗi từ điển đại diện cho một alien khác nhau. Tại ①, câu lệnh aliens = [alien_0, alien_1, alien_2] lưu trữ mỗi từ điển này trong một danh sách gọi là aliens. Cuối cùng, chúng ta lặp lại danh sách và in ra từng alien:

```
{'color': 'green', 'points': 5}
{'color': 'yellow', 'points': 10}
{'color': 'red', 'points': 15}
```

Một ví dụ thực tế hơn sẽ liên quan đến hơn ba alien với code tự động tạo ra từng alien. Trong ví dụ sau, chúng ta sử dụng range() để tạo một hạm đội gồm 30 alien:

```

# Make an empty list for storing aliens.
aliens = []

# Make 30 green aliens.
①for alien_number in range(30):
②    new_alien = {'color': 'green', 'points': 5, 'speed': 'slow'}
③    aliens.append(new_alien)

# Show the first 5 aliens.
④for alien in aliens[:5]:
    print(alien)
print("...")

# Show how many aliens have been created.
⑤print(f"Total number of aliens: {len(aliens)}")

```

Ví dụ bắt đầu bằng việc tạo ra một danh sách chứa các aliens. Tại ①, hàm `range()` trả về một chuỗi các số và nói cho Python biết số lượng các lần lặp của vòng lặp. Mỗi lần lặp, vòng lặp sẽ tạo ra một alien mới ② và đưa alien vào danh sách ③. Tại ④, chúng ta sử dụng danh sách trượt để in ra 5 aliens đầu tiên và tại ⑤ chúng ta in ra độ dài của danh sách để chứng minh chúng ta đã tạo ra đầy đủ 30 aliens.

```

{'speed': 'slow', 'color': 'green', 'points': 5}

...
Total number of aliens: 30

```

Tất cả những alien này đều có những đặc điểm giống nhau, nhưng Python coi mỗi alien là một đối tượng riêng biệt, điều này cho phép chúng ta sửa đổi từng alien riêng lẻ.

Làm thế nào ta có thể làm việc với một nhóm alien như thế này? Hãy tưởng tượng rằng một khía cạnh của trò chơi có một số alien thay đổi màu sắc và di chuyển nhanh hơn khi trò chơi bắt đầu. Khi đến lúc thay đổi màu sắc, chúng ta có thể sử dụng vòng lặp `for` và câu lệnh `if` để thay đổi màu sắc của alien. Ví dụ: để thay đổi ba alien đầu tiên thành màu vàng, alien có tốc độ trung bình, mỗi alien có giá trị 10 điểm, chúng ta có thể làm như sau:

```

# Make an empty list for storing aliens.
aliens = []

```

```

# Make 30 green aliens.
for alien_number in range(30):
    new_alien = {'color': 'green', 'points': 5, 'speed': 'slow'}
    aliens.append(new_alien)
for alien in aliens[:3]:
    if alien['color'] == 'green':
        alien['color'] = 'yellow'
        alien['speed'] = 'medium'
        alien['points'] = 10
# Show the first 5 aliens.
for alien in aliens[:5]:
    print(alien)
print("...")

```

Bởi vì chúng ta muốn sửa đổi ba alien đầu tiên, chúng ta lặp lại một phần chỉ bao gồm ba alien đầu tiên. Tất cả alien hiện có màu xanh lá cây nhưng điều đó không phải lúc nào cũng đúng, vì vậy chúng ta viết câu lệnh if để đảm bảo rằng chúng ta chỉ sửa đổi alien màu xanh lá cây. Nếu alien có màu xanh lá cây, chúng ta thay đổi màu thành 'yellow', tốc độ thành 'medium' và giá trị point thành 10, kết quả hiển thị như sau:

```

{'speed': 'medium', 'color': 'yellow', 'points': 10}
{'speed': 'medium', 'color': 'yellow', 'points': 10}
{'speed': 'medium', 'color': 'yellow', 'points': 10}
{'speed': 'slow', 'color': 'green', 'points': 5}
{'speed': 'slow', 'color': 'green', 'points': 5}
...

```

Ta có thể mở rộng vòng lặp này bằng cách thêm một khối elif biến những alien màu vàng thành màu đỏ, những alien di chuyển nhanh trị giá 15 điểm mỗi con. Nếu không hiển thị lại toàn bộ chương trình, vòng lặp đó sẽ giống như sau:

```

for alien in aliens[0:3]:
    if alien['color'] == 'green':
        alien['color'] = 'yellow'
        alien['speed'] = 'medium'
        alien['points'] = 10
    elif alien['color'] == 'yellow':
        alien['color'] = 'red'
        alien['speed'] = 'fast'
        alien['points'] = 15

```

Thông thường lưu trữ một số từ điển trong một danh sách khi mỗi từ điển chứa nhiều loại thông tin về một đối tượng. Ví dụ: ta có thể tạo một từ điển cho từng người dùng trên một trang web, như chúng ta đã làm trong user.py và lưu trữ các từ điển

riêng lẻ trong một danh sách được gọi là users. Tất cả các từ điển trong danh sách phải có cấu trúc giống nhau để ta có thể lặp qua danh sách và làm việc với từng đối tượng từ điển theo cách giống nhau.

5.4.2. Danh sách trong từ điển

Thay vì đặt một từ điển bên trong một danh sách, đôi khi việc đặt một danh sách bên trong một từ điển sẽ rất hữu ích. Ví dụ: Cách mà ta có thể mô tả một chiếc bánh pizza mà ai đó đang đặt. Nếu chỉ sử dụng một danh sách, tất cả những gì thực sự có thể lưu trữ là danh sách các lớp phủ của bánh pizza. Với từ điển, danh sách các lớp phủ có thể chỉ là một khía cạnh của chiếc bánh pizza mà chúng ta đang mô tả.

Trong ví dụ sau, hai loại thông tin được lưu trữ cho mỗi chiếc bánh pizza: loại vỏ bánh và danh sách lớp phủ bên ngoài. Danh sách các lớp phủ là một giá trị được liên kết với khóa 'toppings'. Để sử dụng các mục trong danh sách, chúng ta đặt tên của từ điển và khóa 'toppings', giống như bất kỳ giá trị nào trong từ điển. Thay vì trả về một giá trị duy nhất, ta nhận được danh sách toppings:

```
# Store information about a pizza being ordered.  
① pizza = {  
    'crust': 'thick',  
    'toppings': ['mushrooms', 'extra cheese'],  
}  
# Summarize the order.  
② print(f"You ordered a {pizza['crust']}-crust pizza "  
      "with the following toppings:")  
③ for topping in pizza['toppings']:  
    print("\t" + topping)
```

Chúng ta bắt đầu tại ① với một từ điển về pizza được đặt hàng. Một khóa trong pizza là 'crust' và giá trị tương ứng với khóa là 'thick'. Khóa tiếp theo là 'toppings' có giá trị là một danh sách các lớp phủ được yêu cầu. Tại ②, chúng ta in ra pizza được đặt hàng. Khi chúng ta muốn cắt một dòng dài và xuống dòng trong code, chúng ta chọn điểm muốn cắt dòng và thêm dấu nháy kép. Lùi dòng dòng tiếp theo và thêm dấu mở nháy kép và tiếp tục chuỗi xuống dòng. Python sẽ tự động nối tất cả chuỗi tìm thấy trong dấu nháy kép mà không xuống dòng như trong code.

Để in ra các lớp phủ, chúng ta dùng vòng lặp for ③. Để truy cập danh sách các lớp phủ, chúng ta sử dụng khóa 'toppings' và Python sẽ lấy ra các loại vỏ bánh trong danh sách.

Kết quả sau đây mô tả về chiếc bánh pizza mà chúng ta dự định xây dựng:

You ordered a thick-crust pizza with the following toppings:

```
mushrooms  
extra cheese
```

Chúng ta có thể lồng một danh sách vào bên trong từ điển bất kỳ lúc nào ta muốn nhiều giá trị được liên kết với một khóa duy nhất trong từ điển. Trong ví dụ trước đó về các ngôn ngữ lập trình yêu thích, nếu chúng ta lưu trữ câu trả lời của từng người trong một danh sách, thì mọi người có thể chọn nhiều ngôn ngữ yêu thích. Khi chúng ta lướt qua từ điển, giá trị được liên kết với mỗi người sẽ là một danh sách các ngôn ngữ thay vì một ngôn ngữ duy nhất. Bên trong vòng lặp for của từ điển, chúng ta sử dụng một vòng lặp for khác để chạy qua danh sách các ngôn ngữ được liên kết với từng người:

```
❶favorite_languages = {  
    'jen': ['python', 'ruby'],  
    'sarah': ['c'],  
    'edward': ['ruby', 'go'],  
    'phil': ['python', 'haskell'],  
}  
  
❷for name, languages in favorite_languages.items():  
    print(f"\n{name.title()}'s favorite languages are:")  
    ❸for language in languages:  
        print(f"\t{language.title()}")
```

Như chúng ta thấy, tại ❶, các ngôn ngữ lập trình ưa thích bây giờ không phải là một giá trị mà là một danh sách. Cần lưu ý rằng một số người thì thích một ngôn ngữ, một số người thì thích nhiều ngôn ngữ. Khi chúng ta lặp qua từ điển tại ❷, chúng ta sử dụng tên biến languages để lưu giá trị từ từ điển và chúng ta biết rằng mỗi giá trị là một danh sách. Chúng ta sử dụng một vòng lặp nữa ❸ để lặp qua các danh sách ngôn ngữ ưa thích của người dùng. Kết quả là chúng ta in ra được danh sách người dùng và các ngôn ngữ ưa thích của họ:

Jen's favorite languages are:

```
Python  
Ruby
```

Sarah's favorite languages are:

```
C
```

Phil's favorite languages are:

```
Python
```

Haskell

Edward's favorite languages are:

Ruby

Go

Ta có thể thêm câu lệnh if vào đầu vòng lặp for của từ điển để xem liệu mỗi người có nhiều hơn một ngôn ngữ yêu thích hay không bằng cách kiểm tra giá trị của len(languages). Nếu một người có nhiều hơn một mục yêu thích, kết quả đầu ra sẽ không đổi. Nếu người đó chỉ có một ngôn ngữ yêu thích, ta có thể thay đổi từ ngữ để phản ánh điều đó. Ví dụ: ta có thể nói ngôn ngữ yêu thích của Sarah là C.

5.4.3. Từ điển bên trong từ điển

Chúng ta có thể lồng một từ điển bên trong một từ điển khác, nhưng code có thể trở nên phức tạp nhanh chóng khi ta làm như vậy. Ví dụ: nếu ta có nhiều người dùng đối với một trang web, mỗi trang có một tên người dùng duy nhất, ta có thể sử dụng tên người dùng làm khóa trong từ điển. Sau đó, ta có thể lưu trữ thông tin về từng người dùng bằng cách sử dụng từ điển làm giá trị được liên kết với tên người dùng của họ. Trong danh sách sau, chúng ta lưu trữ ba phần thông tin về mỗi người dùng: họ, tên và vị trí của họ. Chúng ta sẽ truy cập thông tin này bằng cách lặp qua tên người dùng và từ điển thông tin được liên kết với mỗi tên người dùng:

```
users = {
    'aeinstein': {
        'first': 'albert',
        'last': 'einstein',
        'location': 'princeton',
    },
    'mcurie': {
        'first': 'marie',
        'last': 'curie',
        'location': 'paris',
    },
}
①for username, user_info in users.items():
②    print(f"\nUsername: {username}")
③    full_name = f"{user_info['first']} {user_info['last']}"
    location = user_info['location']

④    print(f"\tFull name: {full_name.title()}")
    print(f"\tLocation: {location.title()}")
```

Đầu tiên, chúng ta định nghĩa từ điển với hai khóa, mỗi khóa là một tên người dùng với giá trị là 'aeinstein' và 'mcurie'. Mỗi giá trị kết hợp với khóa là một từ điển bao gồm tên, họ, và vị trí. Tại ①, chúng ta lặp qua từ điển users. Python sẽ gán mỗi khóa vào biến username và từ điển tương ứng với các username được gán vào biến user_info. Trong vòng lặp chính, chúng ta in ra tên người dùng tại ②.

Tại ③, chúng ta bắt đầu truy cập vào từ điển bên trong. Biến user_info chứa từ điển thông tin của người dùng, bao gồm ba khóa là first, last, và location. Chúng ta sử dụng các khóa này để định dạng tên người dùng cùng vị trí của mỗi người dùng, sau đó in ra kết quả tại bước ④.

```
Username: aeinstein
    Full name: Albert Einstein
    Location: Princeton
Username: mcurie
    Full name: Marie Curie
    Location: Paris
```

Lưu ý rằng cấu trúc của từ điển của mỗi người dùng là giống nhau. Mặc dù Python không yêu cầu nhưng cấu trúc này làm cho các từ điển lồng nhau dễ làm việc hơn. Nếu từ điển của mỗi người dùng có các khóa khác nhau, thì code bên trong vòng lặp for sẽ phức tạp hơn.

Bài tập chương 5

5-1. Person: Sử dụng từ điển để lưu trữ thông tin về một người mà ta biết. Lưu trữ họ, tên, tuổi và thành phố nơi họ sinh sống. Ta nên có các khóa như first_name, last_name, age và city. In từng phần thông tin được lưu trữ trong từ điển đó.

5-2. Favorite Numbers: Sử dụng từ điển để lưu trữ các số yêu thích của mọi người. Hãy nghĩ về những cái tên khác và sử dụng chúng làm khóa trong từ điển của ta. Hãy nghĩ về một con số yêu thích của mỗi người và lưu trữ mỗi con số như một giá trị trong từ điển của ta. In tên của từng người và số yêu thích của họ. Để thú vị hơn nữa, hãy thăm dò ý kiến một vài người ta và nhận một số dữ liệu thực tế cho chương trình của ta.

5-3. Glossary: Một từ điển Python có thể được sử dụng để mô hình hóa một từ điển thực tế. Tuy nhiên, để tránh nhầm lẫn, hãy gọi nó là Glossary.

- Hãy nghĩ về các từ lập trình mà ta đã học trong các chương trước. Sử dụng những từ này làm chìa khóa trong bảng thuật ngữ của ta và lưu trữ ý nghĩa của chúng dưới dạng giá trị.

- In từng từ và nghĩa của nó dưới dạng đầu ra được định dạng gọn gàng. Bạn có thể in từ theo sau bởi dấu hai chấm và sau đó là nghĩa của từ đó hoặc in từ trên một dòng rồi in nghĩa của nó được thụt lề trên dòng thứ hai.

5-4. Glossary 2: Như bài 5-3 nhưng bằng cách thay thế chuỗi lệnh gọi print() của ta bằng một vòng lặp chạy qua các khóa và giá trị của từ điển. Khi nào ta chắc chắn rằng vòng lặp của ta hoạt động, hãy thêm nhiều thuật ngữ Python hơn vào bảng thuật ngữ của ta. Khi ta chạy lại chương trình của mình, những từ và nghĩa mới này sẽ tự động được đưa vào đầu ra.

5-5. Rivers: Tạo một từ điển có chứa ba con sông lớn và quốc gia mà mỗi con sông chảy qua. Một cặp khóa-giá trị có thể là 'nile': 'egypt'.

- Sử dụng vòng lặp để in ra câu về con sông chảy qua quốc gia. Ví dụ *The Nile runs through Egypt*.

- Sử dụng một vòng lặp để in tên của mỗi con sông có trong từ điển.

- Sử dụng một vòng lặp để in tên của mỗi quốc gia có trong từ điển.

5-6. Polling: Sử dụng mã trong favourite_languages.py.

- Lập danh sách những người nên tham gia cuộc thăm dò về ngôn ngữ yêu thích. Bao gồm một số tên đã có trong từ điển và một số tên chưa có.

- Lặp lại danh sách những người nên tham gia cuộc thăm dò. Nếu họ đã thực hiện cuộc thăm dò ý kiến, hãy in tin nhắn cảm ơn họ đã trả lời. Nếu họ chưa tham gia cuộc thăm dò ý kiến, hãy in thông báo mời họ tham gia các cuộc thăm dò.

5-7. People: Bắt đầu với phần chương trình đã làm trong bài 5-1. Tạo hai từ điển mới đại diện cho những người khác nhau và lưu trữ cả ba từ điển trong một danh sách được gọi là people. Lặp qua danh sách people này và in ra tất cả thông tin biết về mỗi người.

5-8. Pets: Tạo một số từ điển, trong đó mỗi từ điển đại diện cho một con vật cưng khác nhau. Trong mỗi từ điển, hãy bao gồm loại động vật và tên của chủ sở hữu.

Lưu các từ điển này trong danh sách gọi là pets. Lặp qua danh sách pets và in ra tất cả các thông tin về thú cưng đã lưu.

5-9. Favorite Places: Tạo một từ điển có tên là favorite_places. Hãy nghĩ ra ba cái tên để sử dụng làm chìa khóa trong từ điển và lưu trữ một đến ba địa điểm yêu thích cho mỗi người. Để làm cho bài tập này thú vị hơn một chút, hãy nhớ một số ta bè kể tên một vài địa điểm yêu thích của họ. Lặp lại từ điển và in tên của từng người và địa điểm yêu thích của họ.

5-10. Favorite Numbers: Thay đổi chương trình từ bài 5-2 sao cho mỗi người có thể thích nhiều hơn một số. Sau đó, in tên của từng người cùng với các số yêu thích của họ.

5-11. Cities: Tạo một từ điển gọi là cities. Sử dụng tên của ba thành phố làm chìa khóa trong từ điển của ta. Tạo từ điển thông tin về từng thành phố và bao gồm quốc gia có thành phố đó, dân số ước tính của nó và một số thông tin thực tế về thành phố đó. Các khóa cho từ điển của mỗi thành phố phải là một cái gì đó như quốc gia, dân số và thông tin thực tế. In tên của mỗi thành phố và tất cả thông tin ta đã lưu trữ về thành phố đó.

5-12. Extensions: Chúng ta hiện đang làm việc với các ví dụ đủ phức tạp để có thể mở rộng chúng theo bất kỳ cách nào. Sử dụng một trong các chương trình ví dụ từ chương này và mở rộng nó bằng cách thêm các khóa và giá trị mới, thay đổi ngữ cảnh của chương trình hoặc cải thiện định dạng của đầu ra.

Kết chương

Trong chương này, chúng ta đã học cách định nghĩa từ điển và cách làm việc với thông tin được lưu trữ trong từ điển. Ta đã học cách truy cập và sửa đổi các phần tử riêng lẻ trong từ điển và cách lặp lại tất cả thông tin trong từ điển. Ta đã học cách lặp lại từ điển các cặp key-value, các khóa và các giá trị của nó. Chúng ta cũng đã học cách lồng nhiều từ điển trong danh sách, lồng danh sách trong từ điển và lồng từ điển vào bên trong một từ điển.

Trong chương tiếp theo, Chúng ta sẽ tìm hiểu về vòng lặp while và cách nhận đầu vào từ những người đang sử dụng chương trình. Đây sẽ là một điều thú vị vì ta sẽ học cách làm cho tất cả các chương trình có tính tương tác: chúng sẽ có thể phản hồi thông tin đầu vào của người dùng.

CHƯƠNG 6. ĐẦU VÀO CỦA NGƯỜI DÙNG (INPUT) VÀ VÒNG LẶP

Hầu hết các chương trình được viết ra để giải quyết vấn đề người dùng cuối. Để làm được điều đó, chương trình nhận các thông tin đầu vào từ người dùng. Ví dụ người dùng cần biết xem họ có đủ độ tuổi đi bầu cử chưa. Nếu viết một chương trình để trả lời câu hỏi này, ta cần biết tuổi của người dùng trước khi cung cấp một câu trả lời.

Chương trình sẽ cần yêu cầu người dùng nhập ở đầu vào tuổi của họ; khi chương trình có đầu vào này, nó có thể so sánh nó với tuổi bình chọn để xác định xem người dùng đã đủ tuổi hay chưa và sau đó báo cáo kết quả.

Trong phần này, chúng ta sẽ học cách chương trình nhận đầu vào từ người dùng để có thể làm việc với đầu vào đó. Khi chương trình cần một cái tên, nó sẽ hiển thị thông báo để người dùng gõ vào. Khi chương trình cần một danh sách các tên, chương trình cần phải đưa ra yêu cầu nhập một chuỗi các tên. Để làm điều đó, ta dùng hàm input().

6.1. Cách hàm input() hoạt động

Hàm input() tạm dừng chương trình của chúng ta và đợi người dùng nhập một số. Khi Python nhận được đầu vào của người dùng, nó sẽ gán đầu vào đó cho biến để giúp thuận tiện khi làm việc. Ví dụ: chương trình sau yêu cầu người dùng nhập một số văn bản, sau đó hiển thị lại thông báo đó cho người dùng:

```
message = input("Tell me something, and I will repeat it back to you: ")
print(message)
```

Hàm input() nhận một đối số: dấu nhắc hoặc hướng dẫn, mà chúng ta muốn hiển thị cho người dùng để họ biết phải làm gì. Trong ví dụ này, khi Python chạy dòng đầu tiên, người dùng sẽ thấy lời nhắc “Tell me something, and I will repeat it back to you: ”.

Chương trình đợi trong khi người dùng nhập phản hồi của họ và tiếp tục sau khi người dùng nhấn enter. Câu trả lời là được gán cho biến message, sau đó hàm print (message) hiển thị tại đầu vào cho người dùng:

```
Tell me something, and I will repeat it back to you: Hello everyone!
Hello everyone!
```

6.1.1. Viết lời nhắc rõ ràng

Mỗi khi sử dụng hàm input(), ta nên bao gồm một lời nhắc rõ ràng, để theo dõi cho người dùng biết chính xác loại thông tin chương trình đang tìm kiếm. Bất kỳ câu lệnh nào cho người dùng biết những gì cần nhập. Ví dụ:

```
name = input("Please enter your name: ")  
print(f"\nHello, {name}!")
```

Thêm khoảng trắng vào cuối lời nhắc (sau dấu hai chấm trong ví dụ trước) để tách lời nhắc khỏi phản hồi của người dùng và để nó rõ ràng cho người dùng nơi để nhập văn bản của họ. Ví dụ:

```
Please enter your name: Eric  
Hello, Eric!
```

Đôi khi ta muốn viết lời nhắc dài hơn một dòng. Ví dụ: có thể cho người dùng biết lý do tại sao chúng ta yêu cầu một số đầu vào. Ta có thể gán lời nhắc của mình cho một biến và chuyển biến đó vào hàm input(). Điều này cho phép tạo lời nhắc qua nhiều dòng, sau đó viết một câu lệnh input() rõ ràng.

```
prompt = "If you tell us who you are, we can personalize the messages you  
see."  
prompt += "\nWhat is your first name? "  
name = input(prompt)  
print(f"\nHello, {name}!")
```

Ví dụ này cho thấy một cách để xây dựng một chuỗi nhiều dòng. Dòng đầu tiên gán phần đầu tiên của thông báo cho dấu nhắc biến. Trong lần thứ hai dòng, toán tử += lấy chuỗi được gán để nhắc và thêm chuỗi mới vào cuối. Lời nhắc bây giờ kéo dài hai dòng, lại có khoảng trắng và dấu hỏi nên rõ ràng hơn.

```
If you tell us who you are, we can personalize the messages you see.  
What is your first name? Eric  
Hello, Eric!
```

6.1.2. Sử dụng int() để nhận đầu vào số nguyên

Khi ta sử dụng hàm input(), Python sẽ thông dịch mọi thứ mà người dùng nhập dưới dạng một chuỗi. Cùng xem xét phiên thông dịch viên sau đây, phiên này yêu cầu tuổi của người dùng:

```
>>> age = input("How old are you? ")  
How old are you? 21  
>>> age  
'21'
```

Người dùng nhập số 21, nhưng khi chúng ta hỏi Python về giá trị của age, nó trả về '21', biểu diễn chuỗi của giá trị số đã nhập. Chúng ta biết Python đã diễn giải đầu vào là một chuỗi vì số bây giờ là được đặt trong dấu ngoặc kép.

Nếu chỉ yêu cầu là in đầu vào thì cách này hoạt động tốt. Nhưng nếu ta cố gắng sử dụng đầu vào là một số, ta sẽ gặp lỗi:

```
>>> age = input("How old are you? ")
How old are you? 21
❶ >>> age >= 18
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
❷     TypeError: unorderable types: str() >= int()
```

Khi ta cố gắng sử dụng đầu vào để thực hiện so sánh số ❶, Python tạo ra lỗi vì nó không thể so sánh một chuỗi với một số nguyên: không thể so sánh chuỗi '21' được gán cho tuổi với giá trị số 18 ❷.

Chúng ta có thể giải quyết vấn đề này bằng cách sử dụng hàm int(), cho biết Python để coi đầu vào là một giá trị số. Hàm int() chuyển đổi biểu diễn chuỗi của một số thành biểu diễn số, như dưới đây:

```
>>> age = input("How old are you? ")
How old are you? 21
❶ >>> age = int(age)
>>> age >= 18
True
```

Trong ví dụ này, khi chúng ta nhập 21 tại dấu nhắc, Python sẽ diễn giải số dưới dạng một chuỗi, nhưng giá trị sau đó được chuyển đổi thành biểu diễn số bởi int() ❶. Bây giờ Python có thể chạy kiểm tra có điều kiện: nó so sánh tuổi (đang là giá trị số 21) và 18 để xem liệu tuổi có lớn hơn hoặc bằng 18. Bài kiểm tra này đánh giá là True.

Làm thế nào để sử dụng hàm int() trong một chương trình thực tế? Hãy xem xét một chương trình xác định xem mọi người có đủ cao để lái tàu lượn hay không:

```
height = input("How tall are you, in inches? ")
height = int(height)
if height >= 48:
    print("\nYou're tall enough to ride!")
else:
```

```
print("\nYou'll be able to ride when you're a little older.")
```

Chương trình có thể so sánh chiều cao với 48 vì chiều cao = int(chiều cao) chuyển đổi giá trị đầu vào thành biểu diễn số trước khi so sánh được thực hiện. Nếu số được nhập lớn hơn hoặc bằng 48, chúng ta cho biết người dùng đủ chiều cao:

```
How tall are you, in inches? 71
You're tall enough to ride!
```

Khi ta sử dụng đầu vào số để thực hiện các phép tính và so sánh, đảm bảo chuyển đổi giá trị đầu vào thành biểu diễn số đầu tiên.

6.1.3. Toán tử modulo

Một công cụ hữu ích để làm việc với thông tin số là toán tử modulo(%), chia một số cho một số khác và trả về phần còn lại:

```
>>> 4 % 3
1
>>> 5 % 3
2
>>> 6 % 3
0
>>> 7 % 3
1
```

Toán tử mô-đun không cho biết một số gấp bao nhiêu lần số khác; nó chỉ cho ta biết phần còn lại là bao nhiêu.

Khi một số chia hết cho một số khác thì số dư là 0, vì vậy toán tử modulo luôn trả về 0. Ta có thể sử dụng điều kiện này để xác định nếu một số chẵn hoặc lẻ:

```
number = input("Enter a number, and I'll tell you if it's even or odd: ")
number = int(number)
if number % 2 == 0:
    print(f"\nThe number {number} is even.")
else:
    print(f"\nThe number {number} is odd.")
```

Các số chẵn luôn chia hết cho hai, vì vậy nếu số dư của một số cho hai là số không (ở đây, nếu số% 2 == 0) là số chẵn. Nếu không thì số đó là lẻ.

```
Enter a number, and I'll tell you if it's even or odd: 42
The number 42 is even.
```

6.2. Giới thiệu vòng lặp while

Vòng lặp for lấy một tập hợp các mục và thực thi một khối mã một lần cho mỗi mục trong tập hợp. Ngược lại, vòng lặp while chạy *miễn là*, hoặc *trong khi*, một điều kiện nhất định là đúng.

6.2.1. Vòng lặp với hành động

Chúng ta có thể sử dụng vòng lặp while để đếm dần một chuỗi số. Ví dụ, vòng lặp while sau đếm từ 1 đến 5:

```
current_number = 1
while current_number <= 5:
    print(current_number)
    current_number += 1
```

Trong dòng đầu tiên, chúng ta bắt đầu đếm từ 1 bằng cách gán `current_number` giá trị 1. Sau đó, vòng lặp while được đặt để tiếp tục chạy khi giá trị của `current_number` nhỏ hơn hoặc bằng 5. Mã bên trong vòng lặp in ra giá trị của `current_number` và sau đó thêm 1 vào giá trị đó với `current_number += 1`. (Toán tử `+=` là viết tắt của `current_number = current_number + 1`)

Python lặp lại vòng lặp miễn là điều kiện `current_number <= 5` là đúng. Vì 1 nhỏ hơn 5 nên Python in ra 1 và sau đó thêm 1, tạo thành số hiện tại là 2. Vì 2 nhỏ hơn 5 nên Python in ra 2 và thêm 1 lần nữa, tạo thành số 3 hiện tại, v.v. Một khi giá trị của `current_number` lớn hơn 5, vòng lặp ngừng chạy và chương trình kết thúc:

```
1
2
3
4
5
```

Các chương trình ta sử dụng hàng ngày rất có thể chứa các vòng lặp while. Ví dụ, một trò chơi cần một vòng lặp while khi tiếp tục chạy khi ta muốn và nó có thể ngừng chạy ngay sau khi ta yêu cầu nó thoát.

6.2.2. Để người dùng lựa chọn khi nào thoát

Chúng ta xây dựng chương trình `parrot.py` để người dùng muốn luôn chạy bằng cách đặt hầu hết chương trình bên trong vòng lặp while. Chúng ta sẽ xác định một giá trị thoát và sau đó giữ chương trình chạy miễn là người dùng chưa nhập giá trị thoát:

```

①  prompt = "\nTell me something, and I will repeat it back to you:"
prompt += "\nEnter 'quit' to end the program. "
②  message = ""
③  while message != 'quit':
    message = input(prompt)
    print(message)

```

Tại ①, chúng ta định nghĩa một lời nhắc gồm 2 lựa chọn: nhập vào một thông điệp hoặc là nhập vào giá trị để thoát chương trình (ở đây là ‘quit’). Tiếp đó, chúng ta định nghĩa một biến message ② là rỗng để kiểm tra trong lần đầu tiên chạy vòng lặp. Lần đầu tiên chương trình chạy và Python đến câu lệnh while, nó cần phải so sánh biến message với ‘quit’, nhưng chưa có đầu vào của người dùng nào được nhập. Nếu Python có không có gì để so sánh, nó sẽ không thể tiếp tục chạy chương trình. Để giải quyết vấn đề này, chúng ta đảm bảo cung cấp cho thông báo một giá trị ban đầu. Mặc dù nó chỉ là một chuỗi rỗng, nó sẽ có ý nghĩa với Python và cho phép nó hoạt động so sánh làm cho vòng lặp while hoạt động. Vòng lặp while tại ③ luôn chạy miễn là giá trị của biến message không thành ‘quit’.

Lần đầu tiên thông qua vòng lặp, biến message chỉ là một chuỗi rỗng, vì vậy Python vào vòng lặp. Tại message = input(prompt), Python hiển thị lời nhắc và đợi người dùng nhập thông tin đầu vào của họ. Bất cứ thứ gì người dùng nhập đều được gán vào biến message và in ra; sau đó, Python đánh giá lại điều kiện của vòng lặp while. Miễn là người dùng chưa nhập từ ‘quit’, lời nhắc được hiển thị lại và Python chờ thêm đầu vào. Khi người dùng cuối cùng nhập ‘quit’, Python dừng thực hiện vòng lặp while và chương trình kết thúc:

```

Tell me something, and I will repeat it back to you:
Enter 'quit' to end the program. Hello everyone!
Hello everyone!

```

```

Tell me something, and I will repeat it back to you:
Enter 'quit' to end the program. Hello again.
Hello again.

```

```

Tell me something, and I will repeat it back to you:
Enter 'quit' to end the program. quit
quit

```

Chương trình này hoạt động tốt, ngoại trừ việc nó in từ ‘quit’ như thể nó là một thông điệp thực tế. Một kiểm tra có điều kiện đơn giản nếu giải quyết được điều này:

```
prompt = "\nTell me something, and I will repeat it back to you:"  
prompt += "\nEnter 'quit' to end the program."  
message = ""  
  
while message != 'quit':  
    message = input(prompt)  
    if message != 'quit':  
        print(message)
```

Bây giờ chương trình sẽ kiểm tra trước khi hiển thị thông báo và chỉ in thông báo nếu nó không khớp với giá trị ‘quit’:

```
Tell me something, and I will repeat it back to you:  
Enter 'quit' to end the program. Hello everyone!  
Hello everyone!
```

```
Tell me something, and I will repeat it back to you:  
Enter 'quit' to end the program. Hello again.  
Hello again.
```

```
Tell me something, and I will repeat it back to you:  
Enter 'quit' to end the program. Quit
```

6.2.3. Sử dụng Flag (cờ)

Trong ví dụ trước, chúng ta đã yêu cầu chương trình thực hiện các tác vụ nhất định trong khi một điều kiện đã cho là đúng. Nhưng còn những chương trình phức tạp hơn trong nhiều sự kiện khác nhau có thể khiến chương trình ngừng chạy?

Ví dụ, trong một trò chơi, một số sự kiện khác nhau có thể kết thúc trò chơi. Khi người chơi hết tàu, thời gian của họ hết hoặc các thành phố mà họ đáng lẽ phải bảo vệ đều bị phá hủy, trò chơi sẽ kết thúc. Nó cần kết thúc nếu bất kỳ một trong những sự kiện này xảy ra. Nếu nhiều sự kiện có thể xảy ra để dừng chương trình, cần kiểm tra tất cả các điều kiện này trong một câu lệnh và nó sẽ phức tạp.

Đối với một chương trình sẽ chạy miễn là có nhiều điều kiện đúng, ta có thể xác định một biến xác định xem toàn bộ chương trình có đang hoạt động hay không. Biến này, được gọi là flag, hoạt động như một tín hiệu cho chương trình. Chúng ta có thể viết các chương trình của mình để chúng chạy trong khi flag được đặt thành True

và ngừng chạy khi bất kỳ sự kiện nào trong số một số sự kiện đặt giá trị của flag thành False.

Kết quả là, câu lệnh while tổng thể của chúng ta chỉ cần kiểm tra một điều kiện: liệu flag hiện là True hay không. Sau đó, tất cả các kiểm tra khác của chúng ta (để xem liệu một sự kiện có xảy ra nên đặt flag thành False) có thể được sáp xếp gọn gàng trong phần còn lại của chương trình.

Hãy thêm flag vào parrot.py từ phần trước. Ta đặt tên flag là *active* (mặc dù có thể gọi nó là bất kỳ thứ gì), sẽ theo dõi xem chương trình sẽ tiếp tục chạy hay không:

```
① active = True
② while active:
    message = input(prompt)
③     if message == 'quit':
        active = False
④     else:
        print(message)
```

Khi chúng ta đặt biến *active* thành True ①, chương trình bắt đầu hoạt động. Làm như vậy làm cho câu lệnh while đơn giản hơn vì không có so sánh nào được thực hiện trong chính câu lệnh while; logic được quan tâm trong các phần khác của chương trình. Miễn là biến *active* vẫn là True, vòng lặp sẽ tiếp tục chạy ②. Trong câu lệnh *if* bên trong vòng lặp *while*, chúng ta kiểm tra giá trị của thông điệp khi người dùng nhập thông tin đầu vào. Nếu người dùng nhập 'quit' ③, chúng ta đặt *active* thành False, và vòng lặp *while* dừng lại. Nếu người dùng nhập bất kỳ điều gì khác ngoài 'quit' x, chúng ta in ra thông điệp là đầu vào người dùng đã nhập.

Chương trình này có đầu ra giống như ví dụ trước, nơi chúng ta đã đặt kiểm tra điều kiện trực tiếp trong câu lệnh while nhưng bây giờ chúng ta có một flag để cho biết liệu chương trình tổng thể có ở trạng thái hoạt động hay không, nó sẽ dễ dàng thêm nhiều kiểm tra hơn (chẳng hạn như câu lệnh *elif*) cho các sự kiện làm cho *active* trở thành False. Điều này rất hữu ích trong các chương trình phức tạp như các trò chơi trong đó có thể có nhiều sự kiện mà mỗi sự kiện sẽ khiến chương trình ngừng chạy. Khi bất kỳ sự kiện nào trong số này làm cho flag *active* trở thành False, vòng lặp trò chơi chính sẽ thoát ra, thông báo "Trò chơi kết thúc" có thể hiển thị và người chơi có thể được cung cấp tùy chọn để chơi lại

6.2.4. Sử dụng break để thoát khỏi vòng lặp

Để thoát khỏi vòng lặp while ngay lập tức mà không cần chạy bất kỳ mã nào còn lại trong vòng lặp, bất kể kết quả của bất kỳ kiểm tra điều kiện nào, hãy sử dụng câu lệnh break. Câu lệnh break định hướng chương trình được viết; ta có thể sử dụng nó để kiểm soát dòng mã nào được thực thi và dòng nào không, vì vậy chương trình chỉ thực thi mã mà ta muốn, và khi muốn mã đó chạy.

Ví dụ: hãy xem xét một chương trình hỏi người dùng về những địa điểm họ đã đến đã đến thăm. Chúng ta có thể dừng vòng lặp while trong chương trình này bằng cách gọi break ngay khi người dùng nhập giá trị 'quit':

```
prompt = "\nPlease enter the name of a city you have visited:  
prompt += "\nEnter 'quit' when you are finished.) "  
①   while True:  
        city = input(prompt)  
        if city == 'quit':  
            break  
        else:  
            print(f"I'd love to go to {city.title()}!")
```

Một vòng lặp bắt đầu bằng while True ① sẽ chạy mãi mãi trừ khi nó đạt đến câu lệnh break. Vòng lặp trong chương trình này tiếp tục yêu cầu người dùng nhập tên của các thành phố họ đã đến cho đến khi họ nhập 'quit'. Khi họ nhập 'quit', câu lệnh break chạy, khiến Python thoát khỏi vòng lặp:

```
Please enter the name of a city you have visited:  
(Enter 'quit' when you are finished.) New York  
I'd love to go to New York!
```

```
Please enter the name of a city you have visited:  
(Enter 'quit' when you are finished.) San Francisco  
I'd love to go to San Francisco!
```

```
Please enter the name of a city you have visited:  
(Enter 'quit' when you are finished.) quit
```

Ghi chú: Ta có thể sử dụng câu lệnh break trong bất kỳ vòng lặp nào của Python. Ví dụ, có thể sử dụng break để thoát khỏi vòng lặp for đang hoạt động qua danh sách hoặc từ điển.

6.2.5. Sử dụng continue trong vòng lặp

Thay vì thoát ra khỏi vòng lặp hoàn toàn mà không thực hiện phần còn lại của nó, ta có thể sử dụng câu lệnh continue để quay lại phần đầu của vòng lặp dựa trên kết quả của một bài kiểm tra có điều kiện. Ví dụ, hãy xem xét một vòng lặp đếm từ 1 đến 10 nhưng chỉ in các số lẻ trong phạm vi đó:

```
current_number = 0
while current_number < 10:
    ①        current_number += 1
    if current_number % 2 == 0:
        continue
    print(current_number)
```

Đầu tiên, chúng ta đặt `current_number` thành 0. Vì `current_number` nhỏ hơn 10, Python vào vòng lặp while. Khi ở trong vòng lặp, chúng ta tăng số đếm lên 1 tại ①, do đó `current_number` là 1. Câu lệnh if sau đó kiểm tra phần dư của `current_number` chia cho 2. Nếu số dư là 0 (có nghĩa là `current_number` là chia hết cho 2), câu lệnh `continue` yêu cầu Python bỏ qua phần còn lại của vòng lặp và quay lại phần đầu. Nếu số hiện tại không chia hết cho 2, phần còn lại của vòng lặp được thực thi và Python in dòng hiện tại số hiện tại:

```
1
3
5
7
9
```

6.2.6. Tránh lặp vô hạn

Mỗi vòng lặp while cần một cách để dừng chạy để nó sẽ không tiếp tục chạy mãi mãi. Ví dụ, vòng lặp đếm này sẽ đếm từ 1 đến 5:

```
x = 1
while x <= 5:
    print(x)
    x += 1
```

Nhưng nếu vô tình bỏ qua dòng `x += 1` (như hình bên dưới), vòng lặp sẽ chạy mãi mãi:

```
# This loop runs forever!
x = 1
while x <= 5:
    print(x)
```

Bây giờ giá trị của x sẽ bắt đầu từ 1 nhưng không bao giờ thay đổi. Do đó, kiểm tra điều kiện $x \leq 5$ sẽ luôn đánh giá là True và vòng lặp while sẽ chạy mãi mãi, in một loạt các số 1, như sau:

```
1
1
1
1
1
--snip--
```

Mọi lập trình viên vô tình viết một vòng lặp while vô hạn theo thời gian, đặc biệt là khi các vòng lặp của chương trình có các điều kiện thoát nhỏ. Nếu chương trình bị mắc kẹt trong một vòng lặp vô hạn, hãy nhấn cTrL-C hoặc chỉ cần đóng cửa sổ đầu cuối hiển thị đầu ra chương trình.

Để tránh viết các vòng lặp vô hạn, hãy kiểm tra mọi vòng lặp while và đảm bảo vòng lặp dừng lại khi ta mong đợi. Nếu muốn chương trình kết thúc khi người dùng nhập một giá trị đầu vào nhất định, hãy chạy chương trình và nhập giá trị đó.

Nếu chương trình không kết thúc, hãy xem xét kỹ cách chương trình xử lý giá trị khiến vòng lặp thoát ra. Đảm bảo ít nhất một phần của chương trình có thể làm cho điều kiện của vòng lặp True hoặc chương trình tới được một câu lệnh break.

Ghi chú: Sublime Text và một số trình soạn thảo khác có một cửa sổ đầu ra được nhúng. Cái này có thể làm cho việc dừng một vòng lặp vô hạn trở nên khó khăn và có thể phải đóng trình chỉnh sửa để kết thúc vòng lặp. Thay nhập vào vùng đầu ra của trình chỉnh sửa trước khi nhấn ctrl-C, và ta sẽ có thể hủy bỏ một vòng lặp vô hạn.

6.3. Sử dụng vòng lặp while với Danh sách và Từ điển

Đến thời điểm này, chúng ta chỉ làm việc với một phần thông tin người dùng tại một thời điểm. Ta đã nhận thông tin đầu vào của người dùng và sau đó in thông tin đầu vào hoặc phản hồi cho nó. Tiếp theo qua vòng lặp while, chúng ta sẽ nhận được một giá trị đầu vào khác và đáp ứng điều đó. Nhưng để theo dõi nhiều người dùng và các thông tin, chúng ta sẽ cần sử dụng danh sách và từ điển với vòng lặp while.

Vòng lặp for có hiệu quả để lặp qua một danh sách, nhưng chúng ta không nên sửa đổi danh sách bên trong vòng lặp for vì Python sẽ gặp khó khăn trong việc theo dõi các phần tử trong danh sách. Để sửa đổi danh sách khi ta làm việc với nó, hãy sử

dụng vòng lặp while. Sử dụng vòng lặp while với danh sách và từ điển cho phép ta thu thập, lưu trữ và tổ chức nhiều đầu vào để kiểm tra và báo cáo về sau.

6.3.1. Chuyển phần tử từ danh sách này sang danh sách khác

Xem xét danh sách những người dùng mới đăng ký nhưng chưa được xác minh của một trang web. Sau khi chúng ta xác minh những người dùng này, làm cách nào ta có thể chuyển họ sang một danh sách riêng biệt gồm những người dùng đã xác minh? Một cách là sử dụng vòng lặp while để kéo người dùng khỏi danh sách người dùng chưa được xác minh và sau đó thêm họ vào một danh sách riêng người dùng đã xác minh. Dưới đây là đoạn mã thực hiện công việc trên:

```
# Start with users that need to be verified,
# and an empty list to hold confirmed users.
❶ unconfirmed_users = ['alice', 'brian', 'candace']
confirmed_users = []
# Verify each user until there are no more unconfirmed users.
# Move each verified user into the list of confirmed users.
❷ while unconfirmed_users:
❸     current_user = unconfirmed_users.pop()
    print(f"Verifying user: {current_user.title()}")
❹     confirmed_users.append(current_user)
# Display all confirmed users.
print("\nThe following users have been confirmed:")
for confirmed_user in confirmed_users:
    print(confirmed_user.title())
```

Chúng ta bắt đầu với danh sách những người dùng chưa được xác nhận tại ❶ (Alice, Brian, và Candace) và một danh sách trống để giữ những người dùng đã xác nhận. Vòng lặp while tại ❷ chạy cho tới khi danh sách unconfirmed_users không rỗng. Trong vòng lặp này,

Hàm pop() tại ❸ xóa từng người dùng chưa được xác nhận từ cuối của danh sách unconfirmed_users. Tại đây, vì Candace là người cuối cùng trong danh sách chưa được xác nhận, tên của cô ấy sẽ là tên đầu tiên được xóa, gán cho biến current_user, và được thêm vào danh sách người dùng được xác nhận – confirmed_users tại ❹. Tiếp theo là Brian, sau đó là Alice.

Chúng ta mô phỏng việc xác nhận từng người dùng bằng cách in thông báo xác minh và sau đó thêm họ vào danh sách người dùng đã xác nhận. Khi danh sách người dùng chưa được xác nhận thu hẹp lại, danh sách người dùng được xác nhận sẽ

tăng lên. Khi danh sách của người dùng chưa được xác nhận trống, vòng lặp dừng và danh sách người dùng đã xác nhận được in ra:

```
Verifying user: Candace
Verifying user: Brian
Verifying user: Alice
```

The following users have been confirmed:

```
Candace
Brian
Alice
```

6.3.2. Xóa tất cả các thẻ hiện của một giá trị trong một danh sách

Trong chương 3, chúng ta đã sử dụng phương thức remove() để xóa một giá trị đặc biệt khỏi danh sách. Hàm remove() hoạt động đúng vì giá trị chúng ta quan tâm chỉ xuất hiện một lần trong danh sách. Nhưng điều gì sẽ xảy ra khi ta muốn xóa toàn bộ các thẻ hiện của giá trị đó trong danh sách?

Giả sử ta có một danh sách các vật nuôi có giá trị 'cat' được lặp lại nhiều lần. Để loại bỏ tất cả các trường hợp của giá trị đó, ta có thể chạy một vòng lặp while cho đến khi 'cat' không còn còn trong danh sách, như được hiển thị ở đây:

```
pets = ['dog', 'cat', 'dog', 'goldfish', 'cat', 'rabbit', 'cat']
print(pets)
while 'cat' in pets:
    pets.remove('cat')
print(pets)
```

Chúng ta bắt đầu với một danh sách chứa nhiều thẻ hiện của 'cat'. Sau khi in danh sách, Python nhập vào vòng lặp while vì nó tìm thấy giá trị 'cat' trong danh sách ít nhất một lần. Khi ở trong vòng lặp, Python sẽ loại bỏ thẻ hiện đầu tiên của 'cat', trở lại dòng while, sau đó nhập lại vòng lặp khi thấy rằng 'cat' vẫn còn trong danh sách. Nó loại bỏ từng thẻ hiện của 'cat' cho đến khi giá trị không còn trong danh sách, lúc này Python thoát khỏi vòng lặp và in lại danh sách:

```
['dog', 'cat', 'dog', 'goldfish', 'cat', 'rabbit', 'cat']
['dog', 'dog', 'goldfish', 'rabbit']
```

6.3.3. Diện từ diễn với đầu vào người dùng

Chúng ta có thể nhắc nhập bao nhiêu thông tin cần thiết trong mỗi lần chuyển qua một vòng lặp while. Chúng ta tạo một chương trình thăm dò ý kiến, trong đó mỗi người đều được đưa ra lời nhắc về tên và phản hồi của người tham gia. Chúng ta sẽ

lưu trữ dữ liệu mà chúng ta tập hợp trong một từ điển, bởi vì ta muốn kết nối mỗi câu trả lời với một người dùng cụ thể:

```
responses = {}  
# Set a flag to indicate that polling is active.  
polling_active = True  
while polling_active:  
    # Prompt for the person's name and response.  
    ①        name = input("\nWhat is your name? ")  
    response = input("Which mountain would you like to climb someday? ")  
    # Store the response in the dictionary.  
    ②        responses[name] = response  
    # Find out if anyone else is going to take the poll.  
    ③        repeat = input("Would you like to let another person respond?  
    (yes/ no) ")  
    if repeat == 'no':  
        polling_active = False  
    # Polling is complete. Show the results.  
    print("\n--- Poll Results ---")  
    ④for name, response in responses.items():  
        print(f"{name} would like to climb {response}.")
```

Chương trình đầu tiên định nghĩa một từ điển trống (response) và đặt một flag (polling_active) để cho biết rằng hoạt động thăm dò ý kiến đang chạy. Khi polling_active là True, Python sẽ chạy mã trong vòng lặp while.

Trong vòng lặp, người dùng được nhắc nhập tên của họ và một ngọn núi mà họ muốn leo lên ①. Thông tin đó được lưu trữ trong từ điển responses ②, và người dùng được hỏi có nên tiếp tục cuộc thăm dò ý kiến ③ hay không. Nếu họ nhập yes, chương trình sẽ nhập lại vòng lặp while. Nếu họ nhập no, polling_active flag được đặt thành False, vòng lặp while ngừng chạy và khỏi mã cuối cùng tại ④ hiển thị kết quả của cuộc thăm dò.

Nếu ta chạy chương trình này và nhập các câu trả lời mẫu, ta sẽ thấy đầu ra như sau:

```
What is your name? Eric  
Which mountain would you like to climb someday? Denali  
Would you like to let another person respond? (yes/ no) yes
```

```
What is your name? Lynn  
Which mountain would you like to climb someday? Devil's Thumb  
Would you like to let another person respond? (yes/ no) no
```

--- Poll Results ---

Lynn would like to climb Devil's Thumb.

Eric would like to climb Denali.

Bài tập chương 6

6-1. Rental Car: Viết một chương trình hỏi người dùng loại xe nào họ muốn thuê, sau đó viết ra một thông điệp về loại xe đó. Ví dụ: “*Let me see if I can find you a Subaru.*”

6-2. Restaurant Seating: viết một chương trình hỏi người dùng muốn đặt bàn có bao nhiêu chỗ. Nếu câu trả lời lớn hơn 8, in ra thông điệp là họ cần chờ để sắp xếp bàn. Nếu không, in ra thông điệp là bàn họ đặt đã sẵn sàng.

6-3. Multiples of Ten: viết chương trình yêu cầu nhập vào một số, sau đó in ra câu trả lời là số đó có phải là bội của 10 hay không.

6-4. Pizza Toppings: viết một chương trình có vòng lặp yêu cầu người dùng nhập vào tên các lớp phủ bánh pizza (toppings) cho tới khi người dùng gõ vào giá trị ‘quit’. Với mỗi giá trị lớp phủ bánh người dùng đã thêm, in ra một thông điệp rằng ta đã thêm lớp phủ đó vào bánh cho họ.

6-5. Movie Tickets: một rạp chiếu phim bán vé có giá thay đổi dựa trên tuổi của khách hàng. Nếu khách hàng dưới 3 tuổi – miễn phí vé; Nếu khách hàng từ 3 tới 12 tuổi – giá vé là 10\$; Nếu khách hàng lớn hơn 12 tuổi – giá vé là 15\$. Viết chương trình yêu cầu khách hàng nhập vào tuổi, sau đó in ra giá vé cho người dùng.

6-6. Three Exits: làm các ví dụ 6-4, 6-5 với các cách sau, mỗi cái ít nhất 1 lần:

- sử dụng kiểm tra có điều kiện trong câu lệnh while để dừng vòng lặp
- sử dụng một biến active để điều khiển số lượt chạy của vòng lặp
- sử dụng break để thoát vòng lặp khi người dùng gõ giá trị ‘quit’

6-7. Infinity: viết một vòng lặp không kết thúc và chạy nó. (Để kết thúc vòng lặp nhấn tổ hợp ctrl-C hoặc đóng cửa sổ đang hiển thị đầu ra).

6-8. Deli: Tạo một danh sách có tên là sandwich_orders và điền vào đó tên của các loại bánh sandwich khác nhau. Sau đó, tạo một danh sách trống được gọi là finish_sandwiches. Lặp lại danh sách các đơn đặt hàng bánh sandwich và in thông báo cho mỗi đơn đặt hàng, chẳng hạn như tôi đã làm bánh mì cá ngừ cho ta. Khi mỗi

chiếc bánh sandwich được làm xong, hãy chuyển nó vào danh sách finish_sandwiches. Sau khi tất cả các loại bánh mì đã được làm xong, hãy in một thông báo liệt kê từng loại bánh sandwich đã được làm.

6-9. No Pastrami: Giả sử trong danh sách bài 6-8 có 3 cái thuộc loại pastrami. Tại phần đầu của mã, in ra là cửa hàng hết pastrami và sử dụng vòng lặp while để loại bỏ tất cả các bánh sandwich loại pastrami trong sandwich_orders. In ra danh sách sandwich_orders để đảm bảo đã loại bỏ hết các sandwich loại pastrami.

6-10 Dream Vacation: Viết một chương trình thăm dò ý kiến người dùng về kỳ nghỉ mơ ước của họ. Viết lời nhắc tương tự như Nếu ta có thể đến thăm một nơi trên thế giới, ta sẽ đi đâu? Bao gồm một khối mã in kết quả của cuộc thăm dò ý kiến.

Kết chương

Trong chương này, chúng ta đã học cách sử dụng input() để cho phép người dùng cung cấp thông tin riêng của họ trong các chương trình của mình. Ta đã học cách làm việc với cả hai đầu vào văn bản và số và cách sử dụng vòng lặp while để khiến chương trình chạy cho tới khi người dùng muốn thoát. Chúng đã thấy một số cách để kiểm soát luồng thực hiện một vòng lặp while bằng cách đặt một giá trị active flag, sử dụng câu lệnh break, và bằng cách sử dụng câu lệnh continue.

Chúng ta đã học cách sử dụng vòng lặp while để di chuyển các phần tử từ danh sách này sang danh sách khác và cách xóa tất cả các thẻ hiện của một giá trị từ một danh sách. Ta cũng đã biết cách sử dụng vòng lặp while với từ diễn.

Trong Chương 7, chúng ta sẽ tìm hiểu về các hàm. Các hàm cho phép ta chia các chương trình thành các phần nhỏ, mỗi phần thực hiện một công việc cụ thể. Ta có thể gọi một hàm bao nhiêu lần tùy thích và ta có thể lưu trữ các hàm trong các tệp riêng biệt. Bằng cách sử dụng các hàm, ta sẽ có thể viết nhiều hơn mã hiệu quả giúp khắc phục sự cố và duy trì dễ dàng hơn và điều đó có thể được sử dụng lại trong nhiều chương trình khác nhau.

CHƯƠNG 7. HÀM

Trong chương này, chúng ta sẽ học cách viết các hàm (functions), hàm là các khối mã được thiết kế để thực hiện một công việc cụ thể. Khi muốn thực hiện một nhiệm vụ cụ thể đã được ta định nghĩa cho một hàm, ta sẽ gọi hàm thực hiện việc đó.

Nếu chúng ta cần thực hiện nhiệm vụ đó nhiều lần trong suốt chương trình, không cần phải nhập tất cả mã cho cùng một nhiệm vụ lặp đi lặp lại; mà chỉ cần gọi hàm chuyên dụng để xử lý tác vụ đó và lệnh gọi yêu cầu Python chạy mã bên trong hàm. Chúng ta sẽ thấy rằng việc sử dụng hàm giúp cho chương trình dễ hơn để đọc, viết, và sửa lỗi.

Trong chương này, chúng ta cũng sẽ tìm hiểu các cách truyền thông tin đến các hàm. Chúng ta sẽ học cách viết một số hàm nhất định có công việc chính là hiển thị thông tin và các chức năng khác được thiết kế để xử lý dữ liệu và trả về giá trị hoặc tập hợp các giá trị. Cuối cùng, chúng ta sẽ học cách lưu trữ các chức năng trong các tệp riêng biệt được gọi là mô-đun để giúp tổ chức các tệp chương trình chính.

7.1. Định nghĩa hàm

Dưới đây là một hàm đơn giản có tên greet_user() in lời chào:

```
def greet_user():
    """Display a simple greeting."""
    print("Hello!")
```

greet_user()

Ví dụ này cho thấy cấu trúc đơn giản nhất của một hàm. Dòng lệnh đầu tiên sử dụng từ khóa *def* để thông báo cho Python rằng ta đang định nghĩa một hàm. Đây là định nghĩa hàm, cho Python biết tên của hàm và nếu có thể, nó cho biết hàm cần loại thông tin nào để thực hiện công việc của nó. Dấu ngoặc đơn chứa thông tin đó. Trong trường hợp này, tên của hàm là *greet_user()* và nó không cần thông tin để thực hiện công việc của mình, vì vậy dấu ngoặc đơn của nó trống. (Mặc dù vậy, các dấu ngoặc đơn là bắt buộc.) Cuối cùng, việc định nghĩa kết thúc bằng dấu hai chấm.

Bất kỳ dòng thuật lè nào sau *def greet_user()*: tạo nên phần thân của hàm. Dòng *"""Display a simple greeting."""* là một chú thích được gọi là mỗi chuỗi tài liệu,

nó mô tả những gì hàm thực hiện. Các chuỗi tài liệu được đặt trong dấu ngoặc kép, Python sẽ tìm kiếm khi nó tạo tài liệu cho các hàm trong chương trình được viết ra.

Dòng `print("Hello!")` Là dòng code duy nhất trong phần thân của hàm này, vì vậy, `greet_user()` chỉ có một lệnh: `print("Hello!")`.

Khi chúng ta muốn sử dụng hàm này, ta gọi nó. Một lệnh gọi hàm yêu cầu Python thực thi code trong hàm. Để gọi một hàm, cần viết tên của hàm, theo sau là bất kỳ thông tin cần thiết nào trong dấu ngoặc đơn. Vì không cần thông tin ở đây nên việc gọi hàm của chúng ta cũng đơn giản như nhập vào `greet_user()`. Như mong đợi, nó in Hello !:

```
Hello!
```

7.1.1. Truyền thông tin tới một hàm

Chỉ cần thay đổi một chút, hàm `greet_user()` không chỉ có thể in ra cho người dùng chuỗi Hello! mà còn chào họ bằng tên. Để hàm thực hiện việc này, chúng nhập `username` trong dấu ngoặc đơn của định nghĩa hàm tại `def greet_user()`. Bằng cách thêm tên người dùng vào đây, ta cho phép hàm chấp nhận bất kỳ giá trị nào của `username` mà được chỉ định. Bây giờ, hàm này yêu cầu ta cung cấp một giá trị cho `username` mỗi khi gọi nó. Khi gọi `greet_user()`, ta có thể truyền cho nó một tên, chẳng hạn như 'jesse', bên trong dấu ngoặc đơn:

```
def greet_user(username):
    """Display a simple greeting."""
    print(f"Hello, {username.title()}!")

greet_user('jesse')
```

Lệnh `greet_user('jesse')` sẽ gọi hàm `greet_user()` và cung cấp cho hàm thông tin cần thiết để thực hiện lệnh gọi `print()`. Hàm chấp nhận tên ta đã chuyển và hiển thị lời chào cho tên đó:

```
Hello, Jesse!
```

Tương tự như vậy, `greet_user('sarah')` sẽ gọi `greet_user()` và in Hello, Sarah! Ta có thể gọi `greet_user()` thường xuyên nếu ta muốn và truyền cho nó bất kỳ tên nào mà muốn in ra lời chào với tên đó.

7.1.2. Đối số và tham số

Trong hàm greet_user() trước đó, chúng ta đã định nghĩa greet_user() để yêu cầu một giá trị cho biến username. Sau khi chúng ta gọi hàm và cung cấp cho nó thông tin (tên của một người), nó sẽ in lời chào phù hợp

Biến username của greet_user() là một ví dụ về một tham số, một phần thông tin mà hàm cần để thực hiện công việc của nó. Giá trị 'jesse' trong greet_user('jesse') là một ví dụ về đối số. Đối số là một phần thông tin được truyền từ một lệnh gọi hàm đến một hàm. Khi chúng ta gọi hàm, chúng ta đặt giá trị mà chúng ta muốn hàm hoạt động trong dấu ngoặc đơn. Trong trường hợp này, đối số 'jesse' đã được chuyển đến hàm greet_user() và giá trị được gán cho tham số username.

7.2. Truyền tham số

Bởi vì một định nghĩa hàm có thể có nhiều tham số, một lệnh gọi hàm có thể cần nhiều đối số. Chúng ta có thể truyền các đối số cho các hàm của mình theo một số cách. Ta có thể sử dụng các đối số có vị trí, các đối số này cần theo thứ tự mà các tham số đã được viết; các đối số từ khóa, trong đó mỗi đối số bao gồm một tên biến và một giá trị; và danh sách và từ điển các giá trị.

7.2.1. Đối số có vị trí

Khi ta gọi một hàm, Python phải khớp từng đối số trong lệnh gọi hàm với một tham số trong định nghĩa hàm. Cách đơn giản nhất để làm điều này là dựa trên thứ tự của các đối số được cung cấp. Các giá trị được so khớp theo cách này được gọi là các đối số vị trí.

Để xem cách hoạt động, hãy xem xét một hàm hiển thị thông tin về vật nuôi. Hàm cho chúng ta biết mỗi loại vật nuôi là gì và tên của vật nuôi:

```
def describe_pet(animal_type, pet_name):
    """Display information about a pet."""
    print(f"\nI have a {animal_type}.")
    print(f"My {animal_type}'s name is {pet_name.title()}.")

describe_pet('hamster', 'harry')
```

Định nghĩa cho thấy rằng hàm này cần một animal_type và một pet_name. Khi chúng ta gọi describe_pet(), chúng ta cần cung cấp một animal_type và một pet_name, theo thứ tự đó. Ví dụ: trong lệnh gọi hàm, đối số 'hamster' được gán cho

tham số animal_type và đối số 'harry' được gán cho tham số pet_name. Trong phần thân hàm, hai tham số này được sử dụng để hiển thị thông tin về vật nuôi được mô tả.

Kết quả đầu ra mô tả một chú chuột lang tên là Harry:

```
I have a hamster.  
My hamster's name is Harry.
```

Nhiều lời gọi hàm

Chúng ta có thể gọi một hàm nhiều lần nếu cần. Việc mô tả một con vật cung thứ 2 khác, chỉ cần thêm một lệnh gọi tới description_pet():

```
def describe_pet(animal_type, pet_name):  
    """Display information about a pet."""  
    print(f"\nI have a {animal_type}.")  
    print(f"My {animal_type}'s name is {pet_name.title()}.")  
  
describe_pet('hamster', 'harry')  
describe_pet('dog', 'willie')
```

Trong lần gọi hàm thứ hai này, chúng ta truyền description_pet() các đối số 'dog' và 'willie'. Như với tập đối số trước đây mà chúng ta đã sử dụng, Python đối sánh 'dog' với tham số animal_type và 'willie' với tham số pet_name. Như trước đây, hàm thực hiện công việc của nó, nhưng lần này nó in ra các giá trị cho một con chó tên là Willie. Bây giờ chúng ta có một chú chuột lang tên là Harry và một chú chó tên là Willie:

```
I have a hamster.  
My hamster's name is Harry.
```

```
I have a dog.  
My dog's name is Willie.
```

Gọi một hàm nhiều lần là một cách làm việc rất hiệu quả. Code mô tả một con vật cung được viết một lần trong hàm. Sau đó, bất cứ lúc nào ta muốn mô tả một vật nuôi mới, ta gọi hàm với thông tin của vật nuôi mới. Ngay cả khi code để mô tả một con vật cung được mở rộng thành mười dòng, ta vẫn có thể mô tả một con vật cung mới chỉ trong một dòng bằng cách gọi lại hàm.

Chúng ta có thể sử dụng nhiều đối số vị trí tùy thích trong các hàm của mình. Python hoạt động thông qua các đối số mà ta cung cấp khi gọi hàm và khớp từng đối số với tham số tương ứng trong định nghĩa của hàm.

Vấn đề thứ tự trong Đối số có vị trí

Ta có thể nhận được kết quả không mong muốn nếu trộn thứ tự của các đối số trong một lệnh gọi hàm khi sử dụng các đối số vị trí:

```
def describe_pet(animal_type, pet_name):
    """Display information about a pet."""
    print(f"\nI have a {animal_type}.")
    print(f"My {animal_type}'s name is {pet_name.title()}.")

describe_pet('harry', 'hamster')
```

Trong lệnh gọi hàm này, chúng ta liệt kê tên trước và loại động vật thứ hai. Vì đối số 'harry' được liệt kê lần đầu tiên nên giá trị đó được gán cho tham số animal_type. Tương tự như vậy, 'hamster' được gán cho pet_name. Bây giờ chúng ta có một "harry" tên là "Hamster":

```
I have a harry.  
My harry's name is Hamster.
```

Nếu chúng ta nhận được kết quả hài hước như thế này, hãy kiểm tra để đảm bảo thứ tự của các đối số trong lệnh gọi hàm khớp với thứ tự của các tham số trong định nghĩa của hàm.

7.2.2. Đối số từ khóa

Đối số từ khóa là một cặp tên-giá trị được truyền cho một hàm. Chúng ta liên kết trực tiếp tên và giá trị bên trong đối số, vì vậy khi ta truyền đối số vào hàm, sẽ không có sự nhầm lẫn nào (sẽ không gặp phải một con harry có tên Hamster). Các đối số từ khóa giúp chúng ta không phải lo lắng về việc sắp xếp chính xác các đối số của mình trong lệnh gọi hàm và chúng làm rõ vai trò của từng giá trị trong lệnh gọi hàm. Ví dụ:

```
def describe_pet(animal_type, pet_name):
    """Display information about a pet."""
    print(f"\nI have a {animal_type}.")
    print(f"My {animal_type}'s name is {pet_name.title()}.")

describe_pet(animal_type='hamster', pet_name='harry')
```

Hàm description_pet() không thay đổi. Nhưng khi chúng ta gọi hàm, chúng ta sẽ nói rõ ràng với Python mà mỗi đối số nên được khớp với tham số nào. Khi Python đọc lệnh gọi hàm, nó sẽ biết gán đối số 'hamster' cho tham số animal_type và đối số

'harry' cho pet_name. Kết quả đầu ra chính xác cho thấy chúng ta có một con chuột lang tên là Harry.

Thứ tự của các đối số từ khóa không quan trọng vì Python biết mỗi giá trị sẽ đi đến đâu. Hai lệnh gọi hàm sau đây là tương đương:

```
describe_pet(animal_type='hamster', pet_name='harry')
describe_pet(pet_name='harry', animal_type='hamster')
```

7.2.3. Giá trị mặc định

Khi viết một hàm, ta có thể xác định một giá trị mặc định cho mỗi tham số. Nếu một đối số cho một tham số được cung cấp trong lệnh gọi hàm, thì Python sẽ sử dụng giá trị đối số. Nếu không, nó sử dụng giá trị mặc định của thông số. Vì vậy, khi ta xác định giá trị mặc định cho một tham số, ta có thể loại trừ đối số tương ứng mà ta thường viết trong lệnh gọi hàm. Sử dụng các giá trị mặc định có thể đơn giản hóa các lệnh gọi hàm và làm rõ các cách mà các hàm thường được sử dụng.

Ví dụ, nếu ta thấy rằng hầu hết các lệnh gọi tới description_pet() đang được sử dụng để mô tả những con chó, ta có thể đặt giá trị mặc định của loại động vật thành 'dog'. Giờ đây, bất kỳ ai gọi description_pet() cho một chú chó đều có thể bỏ qua thông tin đó:

```
def describe_pet(pet_name, animal_type='dog'):
    """Display information about a pet."""
    print(f"\nI have a {animal_type}.")
    print(f"My {animal_type}'s name is {pet_name.title()}.")

describe_pet(pet_name='willie')
```

Chúng ta đã thay đổi định nghĩa của description_pet() với animal_type='dog'. Bây giờ, khi hàm được gọi mà không có animal_type nào được chỉ định, Python biết cách sử dụng giá trị 'dog' cho tham số này:

```
I have a dog.
My dog's name is Willie.
```

Lưu ý rằng thứ tự của các tham số trong định nghĩa hàm phải được thay đổi. Vì giá trị mặc định khiến việc chỉ định một loại động vật làm đối số là không cần thiết, đối số duy nhất còn lại trong lệnh gọi hàm là pet_name. Python vẫn hiểu đây là một đối số vị trí, vì vậy nếu hàm được gọi chỉ với pet_name, đối số đó sẽ khớp với

tham số đầu tiên được liệt kê trong định nghĩa của hàm. Đây là lý do tại sao tham số đầu tiên cần phải là `pet_name`.

Cách đơn giản nhất để sử dụng hàm này bây giờ là chỉ cung cấp tên của một chú chó trong lệnh gọi hàm:

```
describe_pet('willie')
```

Lời gọi hàm này sẽ có đầu ra giống như ví dụ trước. Đôi số duy nhất được cung cấp là 'willie', vì vậy nó được khớp với tham số đầu tiên trong định nghĩa, `pet_name`. Bởi vì không có đôi số nào được cung cấp cho `animal_type`, nên Python sử dụng giá trị mặc định là 'dog'.

Để mô tả một con vật không phải con chó, ta có thể sử dụng một lệnh gọi hàm như sau:

```
describe_pet(pet_name='harry', animal_type='hamster')
```

Bởi vì một đôi số rõ ràng cho `animal_type` được cung cấp, Python sẽ bỏ qua giá trị mặc định của tham số.

Ghi chú: Khi ta sử dụng giá trị mặc định, bất kỳ thông số nào có giá trị mặc định cần được liệt kê sau tất cả các tham số không có giá trị mặc định. Điều này cho phép Python tiếp tục diễn giải các đôi số vị trí một cách chính xác.

7.2.4. Lệnh gọi hàm tương đương

Bởi vì các đôi số vị trí, đôi số từ khóa và giá trị mặc định đều có thể được sử dụng cùng nhau, nên thường ta sẽ có một số cách tương đương để gọi một hàm. Hãy xem xét định nghĩa sau cho `description_pet()` với một giá trị mặc định được cung cấp:

```
def describe_pet(pet_name, animal_type='dog'):
```

Với định nghĩa này, một đôi số luôn cần được cung cấp cho `pet_name` và giá trị này có thể được cung cấp bằng cách sử dụng định dạng vị trí hoặc từ khóa. Nếu con vật được mô tả không phải là con chó, thì một đôi số cho `animal_type` phải bao gồm trong lệnh gọi và đôi số này cũng có thể được chỉ định bằng cách sử dụng định dạng vị trí hoặc từ khoá.

Tất cả các lệnh gọi sau sẽ hoạt động cho hàm này:

```
# A dog named Willie.  
describe_pet('willie')  
describe_pet(pet_name='willie')
```

```
# A hamster named Harry.
describe_pet('harry', 'hamster')
describe_pet(pet_name='harry', animal_type='hamster')
describe_pet(animal_type='hamster', pet_name='harry')
Mỗi lệnh gọi hàm này sẽ có đầu ra giống như các ví dụ trước.
```

7.2.5. Tránh lỗi đối số

Khi ta bắt đầu sử dụng các hàm, đừng ngạc nhiên nếu ta gặp lỗi về các đối số chưa khớp. Các đối số không khớp xảy ra khi ta cung cấp ít hoặc nhiều đối số hơn một hàm cần thực hiện công việc của nó. Ví dụ: đây là những gì sẽ xảy ra nếu chúng ta cố gắng gọi `description_pet()` mà không có đối số:

```
def describe_pet(animal_type, pet_name):
    """Display information about a pet."""
    print(f"\nI have a {animal_type}.")
    print(f"My {animal_type}'s name is {pet_name.title()}.")

describe_pet()
```

Python nhận ra rằng một số thông tin bị thiếu trong lệnh gọi hàm và thông báo lại cho chúng ta biết rằng:

```
Traceback (most recent call last):
  File "pets.py", line 6, in <module>
    describe_pet()
TypeError: describe_pet() missing 2 required positional arguments: 'animal_
type' and 'pet_name'
```

Tại dòng File "pets.py", line 6, in <module>, traceback cho chúng ta biết vị trí của vấn đề, cho phép chúng ta nhìn lại và thấy rằng có điều gì đó không ổn trong lệnh gọi hàm. Tại dòng `describe_pet()`, lệnh gọi hàm vi phạm được viết ra để chúng ta xem. Tại `TypeError: describe_pet() missing 2 required positional arguments: 'animal_type' and 'pet_name'`, traceback cho chúng ta biết đang bị thiếu hai đối số và thông tin tên của các đối số bị thiếu. Nếu hàm này nằm trong một tệp riêng biệt, chúng ta có thể viết lại lệnh gọi một cách chính xác mà không cần phải mở tệp đó và đọc code hàm.

Python hữu ích ở chỗ nó đọc code của hàm và cho biết tên của các đối số cần cung cấp. Đây là một động lực khác để đặt tên mô tả cho các biến và hàm. Nếu ta làm vậy, thông báo lỗi của Python sẽ hữu ích hơn cho chúng ta và bất kỳ ai khác có thể sử dụng code được viết ra.

Nếu ta cung cấp quá nhiều đối số, ta sẽ nhận được một traceback tương tự có thể giúp khớp chính xác lời gọi hàm của mình với định nghĩa hàm.

7.3. Giá trị trả về

Không phải lúc nào một hàm cũng phải hiển thị trực tiếp đầu ra của nó. Thay vào đó, nó có thể xử lý một số dữ liệu và sau đó trả về một giá trị hoặc tập hợp các giá trị. Giá trị mà hàm trả về được gọi là giá trị trả về. Câu lệnh trả về nhận một giá trị từ bên trong một hàm và gửi trở lại dòng được gọi là hàm. Giá trị trả về cho phép ta chuyển phần lớn công việc khó khăn của chương trình sang các hàm, điều này có thể đơn giản hóa phần nội dung chương trình của mình.

7.3.1. Trả về giá trị đơn giản

Hãy xem xét một hàm lấy họ và tên và trả về tên đầy đủ được định dạng gọn gàng:

```
def get_formatted_name(first_name, last_name):
    """Return a full name, neatly formatted."""
    full_name = f"{first_name} {last_name}"
    return full_name.title()

musician = get_formatted_name('jimi', 'hendrix')
print(musician)
```

Định nghĩa của hàm `get_formatted_name()` nhận các tham số là `first_name` và `last_name`. Hàm kết hợp hai tên này, thêm khoảng cách giữa chúng và gán kết quả cho `full_name`. Giá trị của `full_name` được chuyển đổi thành tiêu đề và sau đó được trả về tại dòng `return full_name.title()`.

Khi ta gọi một hàm trả về một giá trị, ta cần cung cấp một biến mà giá trị trả về có thể được gán cho. Trong trường hợp này, giá trị trả về được gán cho biến `musician`. Kết quả hiển thị tên được định dạng gọn gàng từ các phần của tên người:

Jimi Hendrix

Điều này có vẻ như cần làm nhiều công việc để có được một cái tên được định dạng gọn gàng trong khi chúng ta có thể chỉ cần viết:

```
print("Jimi Hendrix")
```

Nhưng khi chúng xem xét làm việc với một chương trình lớn cần lưu trữ nhiều họ và tên riêng biệt, các hàm như `get_formatted_name()` trở nên rất hữu ích. Ta lưu trữ họ và tên riêng biệt rồi gọi hàm này bất cứ khi nào muốn hiển thị tên đầy đủ.

7.3.2. Tạo số đối số tùy chọn

Đôi khi, việc đặt một đối số tùy chọn để những người sử dụng hàm có thể chọn cung cấp thêm thông tin chỉ khi họ muốn. Chúng ta có thể sử dụng các giá trị mặc định để làm cho một đối số tùy chọn.

Ví dụ, giả sử chúng ta muốn mở rộng `get_formatted_name()` để xử lý cả tên đệm. Nỗ lực đầu tiên để bao gồm tên đệm có thể trông như thế này:

```
def get_formatted_name(first_name, middle_name, last_name):
    """Return a full name, neatly formatted."""
    full_name = f"{first_name} {middle_name} {last_name}"
    return full_name.title()

musician = get_formatted_name('john', 'lee', 'hooker ')
print(musician)
```

Hàm này hoạt động khi có họ, tên đệm và tên. Hàm nhận cả ba phần của tên và sau đó xây dựng một chuỗi từ chúng. Hàm thêm dấu cách nếu thích hợp và chuyển đổi tên đầy đủ thành chữ hoa tiêu đề:

```
John Lee Hooker
```

Nhưng không phải lúc nào cũng cần đến tên đệm và hàm này như đã viết sẽ không hoạt động nếu ta cố gắng gọi nó bằng tên và họ. Để đặt tên đệm là tùy chọn, chúng ta có thể cung cấp cho đối số `middle_name` một giá trị mặc định trống và bỏ qua đối số trừ khi người dùng cung cấp giá trị. Để làm cho `get_formatted_name()` hoạt động mà không có tên đệm, chúng ta đặt giá trị mặc định của `middle_name` thành một chuỗi trống và di chuyển nó đến cuối danh sách các tham số:

```
def get_formatted_name(first_name, last_name, middle_name=''):
    """Return a full name, neatly formatted."""
    if middle_name:
        full_name = f"{first_name} {middle_name} {last_name}"
    else:
        full_name = f"{first_name} {last_name}"
    return full_name.title()

musician = get_formatted_name('jimi', 'hendrix')
print(musician)

musician = get_formatted_name('john', 'hooker', 'lee')
print(musician)
```

Trong ví dụ này, tên được xây dựng từ ba phần có thể. Vì luôn có họ và tên nên các tham số này được liệt kê đầu tiên trong định nghĩa của hàm. Tên đệm là tùy chọn, vì vậy nó được liệt kê cuối cùng trong định nghĩa và giá trị mặc định của nó là một chuỗi trống.

Trong phần nội dung của hàm, kiểm tra xem liệu tên đệm đã được cung cấp hay chưa. Python diễn giải các chuỗi không trống là True, vì vậy nếu middle_name là True nếu đối số tên đệm nằm trong lệnh gọi hàm if middle_name:. Nếu tên đệm được cung cấp sẽ tạo thành tên đầy đủ. Sau đó, tên này được đổi thành chữ hoa tiêu đề và được trả về dòng lệnh gọi hàm, nơi nó được gán cho biến musician và được in. Nếu không có tên đệm nào được cung cấp, chuỗi trống không thực hiện kiểm tra if và khôi else được chạy. Tên đầy đủ được tạo chỉ bằng họ và tên, và tên đã định dạng được trả về nơi nó được gán cho musician và được in ra.

Gọi hàm này bằng họ và tên rất đơn giản. Tuy nhiên, nếu chúng ta đang sử dụng tên đệm, chúng ta phải đảm bảo rằng tên đệm là đối số cuối cùng được truyền để Python sẽ khớp với các đối số tại musician = get_formatted_name('john', 'hooker', 'lee') một cách chính xác.

Phiên bản sửa đổi này của hàm hoạt động cho cả những người chỉ có họ và tên, và những người có tên đệm:

```
Jimi Hendrix  
John Lee Hooker
```

Các giá trị tùy chọn cho phép các hàm xử lý nhiều trường hợp sử dụng trong khi vẫn cho phép các lệnh gọi hàm đơn giản nhất có thể.

7.3.3. Trả về một từ điển

Một hàm có thể trả về bất kỳ loại giá trị nào ta cần, bao gồm cả các cấu trúc dữ liệu phức tạp hơn như danh sách và từ điển. Ví dụ, hàm sau nhận các phần của tên và trả về từ điển đại diện cho một người:

```
def build_person(first_name, last_name):  
    """Return a dictionary of information about a person. """  
    person = {'first': first_name, 'last': last_name}  
    return person  
  
musician = build_person('jimi', 'hendrix')  
print(musician)
```

Hàm build_person() nhận họ và tên, và đưa các giá trị này vào từ điển. Giá trị của first_name được lưu trữ với khóa 'first' và giá trị của last_name được lưu trữ với khóa 'last'. Toàn bộ từ điển đại diện cho người được trả về. Giá trị trả về được in với hai phần thông tin văn bản ban đầu hiện được lưu trữ trong từ điển:

```
{'first': 'jimi', 'last': 'hendrix'}
```

Hàm này nhận thông tin dạng văn bản đơn giản và đưa nó vào một cấu trúc dữ liệu có ý nghĩa hơn cho phép ta làm việc với thông tin ngoài việc in ra. Các chuỗi 'jimi' và 'hendrix' hiện được gắn nhãn là tên và họ. Chúng ta có thể dễ dàng mở rộng chức năng này để chấp nhận các giá trị tùy chọn như tên đệm, tuổi, nghề nghiệp hoặc bất kỳ thông tin nào khác mà ta muốn lưu trữ về một người. Ví dụ, thay đổi sau đây cũng cho phép ta lưu trữ tuổi của một người:

```
def build_person(first_name, last_name, age=None):
    """Return a dictionary of information about a person."""
    person = {'first': first_name, 'last': last_name}
    if age:
        person['age'] = age
    return person

musician = build_person('jimi', 'hendrix', age=27)
print(musician)
```

Thêm tham số tùy chọn age mới vào định nghĩa hàm và gán cho tham số giá trị đặc biệt là None, được sử dụng khi một biến không có giá trị cụ thể nào được gán cho nó. Ta có thể coi None như một giá trị giữ chỗ. Trong các kiểm tra có điều kiện, None đánh giá là False. Nếu lệnh gọi hàm bao gồm một giá trị cho tuổi, giá trị đó sẽ được lưu trữ trong từ điển. Hàm này luôn lưu trữ tên của một người, nhưng nó cũng có thể được sửa đổi để lưu trữ bất kỳ thông tin nào khác mà ta muốn về một người.

7.3.4. Sử dụng một hàm với vòng lặp while

Chúng ta có thể sử dụng các hàm với tất cả các cấu trúc Python mà ta đã học cho đến thời điểm này. Ví dụ, hãy sử dụng hàm get_formatted_name() với vòng lặp while để chào hỏi người dùng chính thức hơn. Đây là một cách để chào hỏi mọi người bằng họ và tên của họ:

```
def get_formatted_name(first_name, last_name):
    """Return a full name, neatly formatted."""
    full_name = f"{first_name} {last_name}"
    return full_name.title()
```

```

# This is an infinite loop!
while True:
    print("\nPlease tell me your name:")
    f_name = input("First name: ")
    l_name = input("Last name: ")

    formatted_name = get_formatted_name(f_name, l_name)
    print(f"\nHello, {formatted_name}!")

```

Đối với ví dụ này, chúng ta sử dụng một phiên bản đơn giản của `get_formatted_name()` không liên quan đến tên đệm. Vòng lặp `while` yêu cầu người dùng nhập tên của họ và chúng ta nhắc nhở họ và tên riêng biệt.

Nhưng có một vấn đề với vòng lặp `while` này: Chưa xác định điều kiện dừng. Ta đặt điều kiện dừng ở đâu khi yêu cầu một loạt đầu vào? Chúng ta muốn người dùng có thể dừng dễ dàng nhất có thể, vì vậy mỗi lời nhắc nên đưa ra một cách để dừng. Câu lệnh `break` cung cấp một cách đơn giản để thoát khỏi vòng lặp tại một trong hai dấu nhắc:

```

def get_formatted_name(first_name, last_name):
    """Return a full name, neatly formatted."""
    full_name = f"{first_name} {last_name}"
    return full_name.title()

while True:
    print("\nPlease tell me your name:")
    print("(enter 'q' at any time to quit)")

    f_name = input("First name: ")
    if f_name == 'q':
        break
    l_name = input("Last name: ")
    if l_name == 'q':
        break

    formatted_name = get_formatted_name(f_name, l_name)
    print(f"\nHello, {formatted_name}!")

```

Chúng ta thêm một thông báo cho người dùng biết cách thoát và sau đó chúng ta thoát khỏi vòng lặp nếu người dùng nhập giá trị thoát tại một trong hai lời nhắc. Jetzt ist der Programmfortschritt fortgesetzt, nachdem die Benutzung des 'q'-Befehls den Loop verlassen hat.

Bây giờ chương trình sẽ tiếp tục chào mọi người cho đến khi ai đó nhập '`q`' cho một trong hai tên:

```
Please tell me your name:  
(enter 'q' at any time to quit)  
First name: Eric  
Last name: matthes
```

```
Hello, Eric Matthes!  
Please tell me your name:  
(enter 'q' at any time to quit)  
First name: q
```

7.4. Truyền một danh sách vào hàm

Chúng ta thường sẽ thấy hữu ích khi truyền một danh sách vào một hàm, cho dù đó là danh sách tên, số hay các đối tượng phức tạp hơn, chẳng hạn như từ điển. Khi ta truyền một danh sách cho một hàm, hàm sẽ có quyền truy cập trực tiếp vào nội dung của danh sách. Hãy sử dụng các hàm để làm việc với danh sách hiệu quả hơn.

Giả sử chúng ta có một danh sách người dùng và muốn in lời chào cho mỗi người dùng. Ví dụ sau đây gửi một danh sách các tên đến một hàm có tên gọi là `greet_users()`, hàm này chào từng người trong danh sách:

```
def greet_users(names):  
    """Print a simple greeting to each user in the list."""  
    for name in names:  
        msg = f"Hello, {name.title()}!"  
        print(msg)  
  
usernames = ['hannah', 'ty', 'margot']  
greet_users(usernames)
```

Chúng ta định nghĩa `greet_users()` vì vậy nó mong đợi một danh sách các tên, mà nó sẽ gán cho các tham số `names`. Hàm lặp qua danh sách mà nó nhận được và in lời chào cho mỗi người dùng. Sau đó chúng ta định nghĩa một danh sách người dùng và truyền các tên người dùng trong danh sách tới `greet_users()` trong lời gọi hàm:

```
Hello, Hannah!  
Hello, Ty!  
Hello, Margot!
```

Đây là điều ta muốn. Mọi người dùng đều thấy một lời chào được cá nhân hóa và ta có thể gọi hàm bất kỳ lúc nào muốn để chào một nhóm người dùng cụ thể.

7.4.1. Sửa đổi danh sách trong một hàm

Khi ta truyền một danh sách vào một hàm, hàm này có thể sửa đổi danh sách. Mọi thay đổi được thực hiện đối với danh sách bên trong nội dung của hàm là vĩnh viễn, cho phép ta làm việc hiệu quả ngay cả khi ta đang xử lý một lượng lớn dữ liệu.

Hãy xem xét một công ty tạo ra các mô hình in 3D của các thiết kế mà người dùng gửi. Các thiết kế cần in sẽ được lưu trữ trong một danh sách và sau khi in chúng sẽ được chuyển sang một danh sách riêng. Đoạn code sau thực hiện điều này mà không cần sử dụng các hàm:

```
# Start with some designs that need to be printed.  
unprinted_designs = ['phone case', 'robot pendant', 'dodecahedron']  
completed_models = []  
  
# Simulate printing each design, until none are left.  
# Move each design to completed_models after printing.  
while unprinted_designs:  
    current_design = unprinted_designs.pop()  
    print(f"Printing model: {current_design}")  
    completed_models.append(current_design)  
  
# Display all completed models.  
print("\nThe following models have been printed:")  
for completed_model in completed_models:  
    print(completed_model)
```

Chương trình này bắt đầu với một danh sách các thiết kế cần được in và một danh sách trống được gọi là các completed_models đã hoàn thành mà mỗi thiết kế sẽ được chuyển đến sau khi nó được in. Miễn là các thiết kế vẫn ở trong unprinted_designs, vòng lặp while mô phỏng việc in từng thiết kế bằng cách xóa thiết kế khỏi cuối danh sách, lưu trữ thiết kế đó trong current_design và hiển thị thông báo rằng thiết kế hiện tại đang được in. Sau đó, nó thêm thiết kế vào danh sách các mẫu đã hoàn thành. Khi vòng lặp chạy xong, danh sách các thiết kế đã được in sẽ hiển thị:

```
Printing model: dodecahedron  
Printing model: robot pendant  
Printing model: phone case
```

```
The following models have been printed:  
dodecahedron  
robot pendant  
phone case
```

Chúng ta có thể tổ chức lại đoạn code này bằng cách viết hai hàm, mỗi hàm thực hiện một công việc cụ thể. Hầu hết code sẽ không thay đổi; chúng ta chỉ đang làm cho nó được cấu trúc cẩn thận hơn. Hàm đầu tiên sẽ xử lý việc in các thiết kế và hàm thứ hai sẽ tóm tắt các bản in đã được thực hiện:

```
def print_models(unprinted_designs, completed_models):
    """
    Simulate printing each design, until none are left.
    Move each design to completed_models after printing.
    """
    while unprinted_designs:
        current_design = unprinted_designs.pop()
        print(f"Printing model: {current_design}")
        completed_models.append(current_design)

def show_completed_models(completed_models):
    """Show all the models that were printed."""
    print("\nThe following models have been printed:")
    for completed_model in completed_models:
        print(completed_model)

unprinted_designs = ['phone case', 'robot pendant', 'dodecahedron']
completed_models = []
print_models(unprinted_designs, completed_models)
show_completed_models(completed_models)
```

Chúng ta định nghĩa hàm `print_models()` với hai tham số: `unprinted_designs` và `completed_models`. Với hai danh sách này, hàm mô phỏng việc in từng thiết kế bằng cách làm trống danh sách các thiết kế chưa in và điền vào danh sách các mô hình đã hoàn thành. Tại hàm thứ 2, định nghĩa hàm `show_completed_models()` với một tham số: `completed_models`. Với danh sách này, `show_completed_models()` hiển thị tên của từng mô hình đã được in.

Chương trình này có đầu ra giống với phiên bản không có hàm, nhưng code được tổ chức hơn nhiều. Code thực hiện hầu hết công việc đã được chuyển sang hai hàm riêng biệt, điều này làm cho phần chính của chương trình dễ hiểu hơn. Nhìn vào phần nội dung của chương trình để biết chương trình này đang làm gì dễ dàng hơn nhiều so với phần trên:

```
unprinted_designs = ['phone case', 'robot pendant', 'dodecahedron']
completed_models = []
```

```
print_models(unprinted_designs, completed_models)
show_completed_models(completed_models)
```

Chúng ta thiết lập một danh sách các thiết kế chưa in và một danh sách trống sẽ chứa các mô hình đã hoàn thành. Sau đó, bởi vì chúng ta đã xác định hai hàm của mình, tất cả những gì chúng ta phải làm là gọi chúng và truyền cho chúng các đối số phù hợp. Chúng ta gọi print_models() và truyền cho nó hai danh sách mà nó cần; như mong đợi, print_models() mô phỏng việc in các thiết kế. Sau đó, gọi show_completed_models() và truyền nó vào completed_models để nó có thể báo cáo các mô hình đã được in. Tên hàm mô tả cho phép người khác đọc và hiểu code này, ngay cả khi không có chú thích.

Chương trình này dễ mở rộng và bảo trì hơn so với phiên bản không có hàm. Nếu sau này chúng ta cần in thêm các thiết kế, chúng ta có thể gọi lại print_models() một lần nữa. Nếu nhận thấy code in cần được sửa đổi, có thể thay đổi code một lần và các thay đổi sẽ diễn ra ở mọi nơi mà hàm được gọi. Kỹ thuật này hiệu quả hơn so với việc phải cập nhật code riêng lẻ ở một số nơi trong chương trình.

Ví dụ này cũng thể hiện ý tưởng rằng mỗi hàm nên có một công việc cụ thể. Hàm đầu tiên in từng thiết kế và hàm thứ hai hiển thị các mẫu đã hoàn thành. Điều này có lợi hơn so với việc sử dụng một hàm để thực hiện cả hai công việc. Nếu ta đang viết một hàm và nhận thấy hàm đang thực hiện quá nhiều tác vụ khác nhau, hãy thử chia mã thành hai hàm. Hãy nhớ rằng ta luôn có thể gọi một hàm từ một hàm khác, điều này có thể hữu ích khi tách một nhiệm vụ phức tạp thành một loạt các bước.

7.4.2. Ngăn một hàm sửa đổi danh sách

Đôi khi ta muốn ngăn một hàm sửa đổi danh sách. Ví dụ, giả sử ta bắt đầu với một danh sách các thiết kế chưa in và viết một hàm để truyền chúng vào danh sách các mô hình đã hoàn thành, như trong ví dụ trước. Ta có thể quyết định rằng mặc dù đã in tất cả các thiết kế, ta vẫn muốn giữ lại danh sách ban đầu các thiết kế chưa in để làm hồ sơ của mình. Nhưng vì ta đã di chuyển tất cả các tên thiết kế ra khỏi unprinted_designs, danh sách bây giờ trống và danh sách trống là phiên bản duy nhất ta có; bản gốc đã biến mất. Trong trường hợp này, ta có thể giải quyết vấn đề này bằng cách chuyển cho hàm một bản sao của danh sách, không phải bản gốc. Bất kỳ

thay đổi nào mà hàm thực hiện đối với danh sách sẽ chỉ ảnh hưởng đến bản sao, giữ nguyên danh sách gốc.

Ta có thể gửi một bản sao của danh sách đến một chức năng như sau:

```
function_name(list_name[:])
```

Kí hiệu lát cắt [:] tạo một bản sao của danh sách để gửi đến hàm. Nếu không muốn làm trống danh sách các thiết kế chưa in, chúng ta có thể gọi print_models() như sau:

```
print_models(unprinted_designs[:], completed_models)
```

Hàm print_models() có thể thực hiện công việc của nó vì nó vẫn nhận được tên của tất cả các thiết kế chưa được in. Nhưng lần này nó sử dụng một bản sao của danh sách thiết kế chưa in ban đầu, không phải danh sách unprinted_designs thực tế. Danh sách complete_models sẽ điền tên các mẫu đã in giống như trước đây, nhưng danh sách ban đầu của các mẫu thiết kế chưa in sẽ không bị ảnh hưởng bởi hàm.

Mặc dù ta có thể bảo toàn nội dung của một danh sách bằng cách chuyển một bản sao của nó cho các hàm của mình, nhưng ta nên chuyển danh sách gốc cho các hàm trừ khi ta có một lý do cụ thể để chuyển một bản sao. Sẽ hiệu quả hơn khi một hàm làm việc với danh sách hiện có để tránh sử dụng thời gian và bộ nhớ cần thiết để tạo một bản sao riêng biệt, đặc biệt khi ta đang làm việc với các danh sách lớn.

7.5. Truyền một số đối số tùy ý

Đôi khi ta sẽ không biết trước có bao nhiêu đối số mà một hàm cần chấp nhận. May mắn thay, Python cho phép một hàm thu thập một số đối số tùy ý từ câu lệnh gọi.

Ví dụ, hãy xem xét một hàm làm một chiếc bánh pizza. Nó cần phải có một số lớp phủ, nhưng ta không thể biết trước một người sẽ muốn bao nhiêu lớp phủ bánh. Hàm trong ví dụ sau có một tham số, * toppings, nhưng tham số này thu thập nhiều đối số như dòng gọi cung cấp:

```
def make_pizza(*toppings):
    """Print the list of toppings that have been requested."""
    print(toppings)

make_pizza('pepperoni')
make_pizza('mushrooms', 'green peppers', 'extra cheese')
```

Dấu hoa thị trong tham số * toppings cho Python biết tạo một bộ giá trị trống được gọi là toppings và đóng gói bất kỳ giá trị nào nó nhận được vào bộ này. Lời gọi print() trong thân hàm tạo ra kết quả cho thấy Python có thể xử lý một lệnh gọi hàm với một giá trị và một lệnh gọi có ba giá trị. Nó xử lý các lần gọi khác nhau một cách tương tự. Lưu ý rằng Python đóng gói các đối số thành một bộ, ngay cả khi hàm chỉ nhận một giá trị:

```
('pepperoni',)  
('mushrooms', 'green peppers', 'extra cheese')
```

Bây giờ chúng ta có thể thay thế lệnh gọi print() bằng một vòng lặp chạy qua danh sách các topping và mô tả bánh pizza đang được đặt hàng:

```
def make_pizza(*toppings):  
    """Summarize the pizza we are about to make."""  
    print("\nMaking a pizza with the following toppings:")  
    for topping in toppings:  
        print(f"- {topping}")  
make_pizza('pepperoni')  
make_pizza('mushrooms', 'green peppers', 'extra cheese')
```

Hàm phản hồi một cách thích hợp, cho dù nó nhận một giá trị hay ba giá trị:

```
Making a pizza with the following toppings:  
- pepperoni
```

```
Making a pizza with the following toppings:  
- mushrooms  
- green peppers  
- extra cheese
```

Cú pháp này hoạt động bất kể hàm nhận được bao nhiêu đối số.

7.5.1. Kết hợp đối số vị trí và tùy ý

Nếu ta muốn một hàm chấp nhận một số loại đối số khác nhau, thì tham số chấp nhận một số đối số tùy ý phải được đặt cuối cùng trong định nghĩa hàm. Python khớp các đối số vị trí và từ khóa trước rồi thu thập bất kỳ đối số nào còn lại trong tham số cuối cùng.

Ví dụ, nếu hàm cần tính kích thước cho bánh pizza, thì tham số đó phải đứng trước tham số * toppings:

```
def make_pizza(size, *toppings):  
    """Summarize the pizza we are about to make."""
```

```

print(f"\nMaking a {size}-inch pizza with the following toppings:")
for topping in toppings:
    print(f"- {topping}")

make_pizza(16, 'pepperoni')
make_pizza(12, 'mushrooms', 'green peppers', 'extra cheese')

```

Trong định nghĩa hàm, Python chỉ định giá trị đầu tiên mà nó nhận được cho tham số size. Tất cả các giá trị khác sau đó được lưu trữ trong bộ giá trị toppings. Các lệnh gọi hàm bao gồm một đối số cho size trước tiên, sau đó là toppings nếu cần.

Giờ đây, mỗi chiếc bánh pizza đều có kích thước và một số lớp phủ, và mỗi phần thông tin được in ở vị trí thích hợp, hiển thị size trước và toppings sau:

```
Making a 16-inch pizza with the following toppings:
- pepperoni
```

```
Making a 12-inch pizza with the following toppings:
- mushrooms
- green peppers
- extra cheese
```

7.5.2. Sử dụng đối số từ khóa tùy ý

Đôi khi ta muốn chấp nhận một số lượng đối số tùy ý, nhưng ta sẽ không biết trước loại thông tin nào sẽ được chuyển đến hàm. Trong trường hợp này, ta có thể viết các hàm chấp nhận nhiều cặp khóa-giá trị như câu lệnh gọi cung cấp. Một ví dụ liên quan đến việc xây dựng hồ sơ người dùng: ta biết mình sẽ nhận được thông tin về người dùng, nhưng ta không chắc mình sẽ nhận được loại thông tin nào. Hàm build_profile() trong ví dụ sau luôn nhận họ và tên, nhưng nó cũng chấp nhận một số đối số từ khóa tùy ý:

```

def build_profile(first, last, **user_info):
    """Build a dictionary containing everything we know about a user."""
    user_info['first_name'] = first
    user_info['last_name'] = last
    return user_info

user_profile = build_profile('albert', 'einstein',
                             location='princeton',
                             field='physics')
print(user_profile)

```

Định nghĩa của build_profile() yêu cầu họ và tên, sau đó nó cho phép người dùng chuyển nhiều cặp tên-giá trị tùy thích. Dấu hoa thị kép trước tham số **

`user_info` khiến Python tạo một từ điển trống có tên là `user_info` và đóng gói bất kỳ cặp tên-giá trị nào mà nó nhận được vào từ điển này. Trong hàm, ta có thể truy cập các cặp khóa-giá trị trong `user_info` giống như cách làm đối với bất kỳ từ điển nào.

Trong phần nội dung của `build_profile()`, chúng ta thêm họ và tên vào từ điển `user_info` vì chúng ta sẽ luôn nhận được hai phần thông tin này từ người dùng và chúng chưa được đưa vào từ điển. Sau đó, chúng ta trả lại từ điển `user_info` cho dòng gọi hàm.

Chúng ta gọi `build_profile()`, truyền cho nó tên là 'albert', họ 'einstein' và hai cặp khóa-giá trị là `location = 'Princeton'` và `field = 'Physics'`. Chúng ta gán hồ sơ đã trả về cho `user_profile` và in `user_profile`:

```
{'location': 'princeton', 'field': 'physics',
'first_name': 'albert', 'last_name': 'einstein'}
```

Từ điển trả về chứa họ và tên của người dùng, trong trường hợp này là cả vị trí và lĩnh vực nghiên cứu. Hàm sẽ hoạt động bất kể có bao nhiêu cặp khóa-giá trị bổ sung được cung cấp trong lệnh gọi hàm.

Có thể kết hợp các giá trị vị trí, từ khóa và tùy ý theo nhiều cách khác nhau khi viết các hàm của riêng mình. Sẽ rất hữu ích khi biết rằng tất cả các loại đối số này đều tồn tại vì ta sẽ thấy chúng thường xuyên khi ta bắt đầu đọc code của người khác. Cần thực hành để học cách sử dụng các loại khác nhau một cách chính xác và biết khi nào sử dụng mỗi loại. Hiện tại, hãy nhớ sử dụng cách tiếp cận đơn giản nhất để hoàn thành công việc.

7.6. Hàm Lambda

7.6.1. Định nghĩa hàm Lambda

Về cơ bản, một hàm lambda cũng giống như bất kỳ hàm python bình thường nào, ngoại trừ việc nó không có tên khi định nghĩa nó và nó được chứa trong một dòng mã.

Cú pháp của hàm lambda:

```
lambda argument(s): expression
```

Một hàm lambda đánh giá một biểu thức cho một đối số nhất định. Ta cung cấp cho hàm một giá trị (đối số) và sau đó cung cấp hoạt động (biểu thức). Từ khóa

lambda phải xuất hiện đầu tiên. Đầu hai chấm đầy đủ (:) ngăn cách đối số và biểu thức.

```
#Normal function Python
def a_name(x) :
    return x+x

#Lambda function
lambda x: x+x
```

Hàm lambda trả về một con trỏ hàm và có thể gán con trỏ hàm cho một biến.

Ví dụ:

```
y=lambda x:x*x*4
print(y(12))
#48
```

7.6.2. Sử dụng Lambda trong hàm filter

Hàm filter là thư viện có sẵn trong python, hàm chỉ trả về các giá trị phù hợp với tiêu chí nhất định.

Cú pháp:

```
filter(function, iterable)
```

Trong đó function là hàm lambda được định nghĩa trong dòng và iterable là một đối tượng có nhiều phần tử theo tuần tự như List, Set, hoặc danh sách các đối tượng.

Ví dụ:

```
list1=[1,2,3,4,5,6,7,8,9]
list2=list(filter(lambda x:x%2==0, list1))
print(list2)
```

Kết quả:

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

Lý giải: các phần tử trong danh sách list1 sẽ lần lượt được lọc qua biểu thức hàm lambda, bộ lọc chỉ cho qua những phần tử hàm lambda trả về True và giữ lại những phần tử hàm lambda trả về False. Kết quả là danh sách list2 các phần tử được bộ lọc cho qua.

7.6.3. Sử dụng hàm Lambda trong hàm map()

Giống như hàm filter(), hàm map() là hàm có sẵn (build-in) trong python.

Cấu trúc:

```
map(function, iterable).
```

Trong đó function là hàm lambda được định nghĩa trong dòng và iterable là một đối tượng có nhiều phần tử theo tuần tự như List, Set, hoặc danh sách các đối tượng.

Kết quả là một danh sách thay đổi so với danh sách ban đầu, thay đổi được mô tả bởi hàm function.

```
list1=[1, 2, 3, 4, 5, 6, 7, 8, 9]
list3=list(map(lambda x:pow(x, 3), list1))
print(list3)
```

Kết quả:

```
[1, 8, 27, 64, 125, 216, 343, 512, 729]
```

Lý giải: từng phần tử trong danh sách list1 sẽ được ánh xạ lần lượt qua hàm lambda và trở thành phần tử trong danh sách list3.

7.6.4. Sử dụng hàm Lambda trong Series và Dataframe

Series và Dataframe là hai kiểu dữ liệu phổ biến được cung cấp trong thư viện Pandas để xử lý dữ liệu dạng bảng hoặc dạng chuỗi.

Đối tượng Series là một cột trong dataframe, hay nói cách khác là một chuỗi các giá trị với các chỉ số tương ứng.

Hàm lambda có thể được sử dụng để thao tác dữ liệu trong mô Pandas frame như sau:

```
import pandas as pd
df = pd.DataFrame({'Name': ['Luke', 'Gina', 'Sam', 'Emma'],
                   'Status': ['Father', 'Mother', 'Son', 'Daughter'],
                   'Birthyear': [1976, 1984, 2013, 2016],
                   })

```

	Name	Status	Birthyear
0	Luke	Father	1976
1	Gina	Mother	1984
2	Sam	Son	2013
3	Emma	Daughter	2016

Để có được tuổi hiện tại của mỗi thành viên, chúng ta lấy năm hiện tại trừ đi năm sinh của họ. Trong hàm lambda bên dưới, x tham chiếu đến một giá trị trong cột năm sinh:

```
df['age']=df['Birthyear'].apply(lambda x:2022-x)
```

Kết quả cho thấy cột age mới được bổ sung và hiển thị tuổi tính tới năm 2022 tương ứng với năm sinh.

	Name	Status	Birthyear	age
0	Luke	Father	1976	46
1	Gina	Mother	1984	38
2	Sam	Son	2013	9
3	Emma	Daughter	2016	6

Lọc dữ liệu từ Series:

```
print(list(filter(lambda x:x>18, df['age'])))
```

Kết quả: in ra danh sách các số trong tuổi có giá trị lớn hơn 18.

[46, 38]

Ánh xạ dữ liệu từ Series, sau đó thêm cột vào Dataframe:

```
df['Gender']=df['Status'].map(lambda x: 'Male' if x=='Father' or x=='Son' else 'Female')
```

Các dữ liệu có giá trị bằng Father hoặc Son trong cột Status sẽ chuyển thành Male, các dữ liệu có giá trị bằng Mother sẽ chuyển thành Female, sau đó thêm các dữ liệu này vào cột Gender, tương ứng theo hàng. Kết quả:

	Name	Status	Birthyear	age	Gender
0	Luke	Father	1976	46	Male
1	Gina	Mother	1984	38	Female
2	Sam	Son	2013	9	Male
3	Emma	Daughter	2016	6	Female

7.6.5. Sắp xếp dữ liệu nhờ hàm lambda

Với dữ liệu là danh sách lồng nhau (danh sách trong từ điển, danh sách trong danh sách hoặc từ điển trong danh sách), chúng ta có thể sử dụng từ khóa key và hàm lambda trong hàm sort() hoặc sorted() để lựa chọn yếu tố sắp xếp.

```
data = [("B", 5, "20"), ("A", 1, "5"), ("C", 6, "10")]
data.sort(key=lambda x: x[0])
```

```
print(data)
```

Danh sách data gồm 3 phần tử, trong đó mỗi phần tử lại là một danh sách gồm các phần tử. Hàm sort(key=) sẽ xác định yếu tố được sắp xếp, trong trường hợp này là lấy phần tử con đầu tiên của mỗi phần tử trong danh sách data để sắp xếp theo thứ tự tăng dần (mặc định).

Kết quả:

```
[('A', 1, '5'), ('B', 5, '20'), ('C', 6, '10')]
```

Trong trường hợp chúng ta muốn sắp xếp theo phần tử con thứ ba (index=2), chúng ta có thể thay đổi như sau:

```
data = [("B", 5, "20"), ("A", 1, "5"), ("C", 6, "10")]
data.sort(key=lambda x: x[2])
print(data)
```

Kết quả:

```
[('C', 6, '10'), ('B', 5, '20'), ('A', 1, '5')]
```

Hàm sort(), và sorted() đều được sử dụng trong trường hợp này, và có thể sắp xếp theo thứ tự ngược lại nếu dùng lựa chọn reverse=True trong hàm.

Ví dụ:

```
# Declare a list of string with number values
n_list = ['11', '50', '5', '1', '37', '19']
print(sorted(n_list))

# Sort the list using lambda and sorted function
sorted_list = sorted(n_list, key=lambda x: int(x[:]), reverse=True)
# Print the sorted list
print("The list of the sorted values are:")
for value in sorted_list:
    print(value, end=' ')
```

Trong đoạn code trên, đầu vào là danh sách các ký tự, do đó nếu chỉ dùng sắp xếp thông thường thì sẽ được kết quả sắp xếp theo mã ASCII của ký tự, do vậy nếu muốn sắp xếp dạng số, cần sử dụng từ khóa key và lambda. x:int(x[:]) có nghĩa là yếu tố để sắp xếp chính là giá trị int của mỗi phần tử dạng chuỗi trong danh sách ban đầu.

7.7. Lưu trữ hàm trong module

Một ưu điểm của các hàm là cách chúng tách các khối mã nguồn khỏi chương trình chính. Bằng cách sử dụng tên mô tả cho các hàm, chương trình chính sẽ dễ theo

dõi hơn nhiều. Ta có thể lưu trữ các hàm của mình trong một tệp riêng được gọi là mô-đun và sau đó nhập mô-đun đó vào chương trình chính. Một câu lệnh nhập (import) yêu cầu Python làm cho code trong một mô-đun có sẵn trong tệp chương trình hiện đang chạy.

Việc lưu trữ các hàm trong một tệp riêng biệt cho phép ẩn các chi tiết của code chương trình và tập trung vào logic cấp cao hơn của nó. Nó cũng cho phép sử dụng lại các hàm trong nhiều chương trình khác nhau. Khi ta lưu trữ các hàm của mình trong các tệp riêng biệt, ta có thể chia sẻ các tệp đó với các lập trình viên khác mà không cần phải chia sẻ toàn bộ chương trình của mình. Biết cách import các hàm cũng cho phép ta sử dụng thư viện các hàm mà các lập trình viên khác đã viết.

Có một số cách để nhập một mô-đun như sau đây.

7.7.1. Import toàn bộ module

Để bắt đầu import các hàm, trước tiên chúng ta cần tạo một mô-đun. Module là một tệp kết thúc bằng .py có chứa code muốn import vào chương trình chính. Hãy tạo một mô-đun có chứa hàm make_pizza(). Để tạo mô-đun này, chúng ta sẽ xóa mọi thứ khỏi tệp pizza.py ngoại trừ hàm make_pizza():

```
def make_pizza(size, *toppings):
    """Summarize the pizza we are about to make."""
    print(f"\nMaking a {size}-inch pizza with the following toppings:")
    for topping in toppings:
        print(f"- {topping}")
```

Bây giờ chúng ta sẽ tạo một tệp riêng có tên là making_pizzas.py trong cùng thư mục với pizza.py. Tệp này import mô-đun chúng ta vừa tạo và sau đó thực hiện hai lệnh gọi đến make_pizza():

```
import pizza

pizza.make_pizza(16, 'pepperoni')
pizza.make_pizza(12, 'mushrooms', 'green peppers', 'extra cheese')
```

Khi Python đọc tệp này, dòng import pizza yêu cầu Python mở tệp pizza.py và sao chép tất cả các hàm của nó vào chương trình này. Ta không thực sự thấy code được sao chép giữa các tệp vì Python sao chép code phía sau ngay trước khi chương trình chạy. Tất cả những gì ta cần biết là bất kỳ hàm nào được định nghĩa trong pizza.py bây giờ sẽ có sẵn trong make_pizzas.py.

Để gọi một hàm từ một mô-đun đã import, hãy nhập tên của mô-đun ta đã import, pizza, theo sau là tên của hàm, make_pizza(), được phân tách bằng dấu chấm. Code này tạo ra đầu ra giống như chương trình gốc không import mô-đun:

Making a 16-inch pizza with the following toppings:

- pepperoni

Making a 12-inch pizza with the following toppings:

- mushrooms
- green peppers
- extra cheese

Cách tiếp cận đầu tiên để import, trong đó ta chỉ cần viết import sau đó là tên của mô-đun, làm cho mọi chức năng từ mô-đun có sẵn trong chương trình của mình. Nếu ta sử dụng loại câu lệnh import này để import toàn bộ mô-đun có tên module_name.py, mỗi hàm trong mô-đun có sẵn thông qua cú pháp sau:

```
module_name.function_name()
```

7.7.2. Import các hàm cụ thể

Ta cũng có thể import một hàm cụ thể từ một mô-đun. Đây là cú pháp chung cho cách tiếp cận này:

```
from module_name import function_name
```

Ta có thể import bao nhiêu hàm tùy thích từ một mô-đun bằng cách phân tách tên của từng hàm bằng dấu phẩy:

```
from module_name import function_0, function_1, function_2
```

Ví dụ make_pizzas.py sẽ trông giống như thế này nếu chúng ta chỉ muốn import hàm mà chúng ta sẽ sử dụng:

```
from pizza import make_pizza
```

```
make_pizza(16, 'pepperoni')
make_pizza(12, 'mushrooms', 'green peppers', 'extra cheese')
```

Với cú pháp này, ta không cần sử dụng ký hiệu dấu chấm khi gọi một hàm. Vì chúng ta đã import hàm make_pizza() một cách rõ ràng trong câu lệnh import, nên chúng ta có thể gọi nó theo tên khi sử dụng hàm.

7.7.3. Sử dụng as để cấp cho hàm một bí danh (Alias)

Nếu tên của một hàm ta đang import có thể xung đột với tên tồn tại trong chương trình đang viết hoặc nếu tên hàm dài, ta có thể sử dụng một bí danh ngắn, duy

nhất — một tên thay thế tương tự như biệt hiệu cho hàm. Ta sẽ đặt biệt hiệu đặc biệt này cho hàm khi import hàm.

Ở đây chúng ta đặt cho hàm make_pizza() một bí danh, mp(), bằng cách import make_pizza dưới dạng mp. Từ khóa as đổi tên một hàm bằng bí danh được người lập trình chọn:

```
from pizza import make_pizza as mp

mp(16, 'pepperoni')
mp(12, 'mushrooms', 'green peppers', 'extra cheese')
```

Câu lệnh import được hiển thị ở đây đổi tên hàm make_pizza() thành mp() trong chương trình này. Bất cứ khi nào chúng ta muốn gọi make_pizza(), chúng ta có thể chỉ cần viết mp() và Python sẽ chạy code trong make_pizza() đồng thời tránh bất kỳ sự nhầm lẫn nào với một hàm make_pizza() khác mà ta có thể ghi trong tệp chương trình này.

Cú pháp chung để cung cấp bí danh là:

```
from module_name import function_name as fn
```

7.7.4. Sử dụng as để cung cấp cho một mô-đun một bí danh

Chúng cũng có thể cung cấp bí danh cho tên mô-đun. Đặt cho mô-đun một bí danh ngắn, như p cho pizza, cho phép gọi các chức năng của mô-đun nhanh hơn. Gọi p.make_pizza() ngắn gọn hơn gọi pizza.make_pizza():

```
import pizza as p

p.make_pizza(16, 'pepperoni')
p.make_pizza(12, 'mushrooms', 'green peppers', 'extra cheese')
```

Module Pizza được đặt bí danh p trong câu lệnh import, nhưng tất cả các chức năng của mô-đun vẫn giữ nguyên tên ban đầu. Việc gọi các hàm bằng cách viết p.make_pizza() không chỉ ngắn gọn hơn viết pizza.make_pizza() mà còn chuyển hướng sự chú ý khỏi tên mô-đun và cho phép tập trung vào tên mô tả của các hàm của nó. Các tên hàm này, cho ta biết rõ ràng chức năng của mỗi hàm, quan trọng hơn đối với khả năng đọc code hơn là sử dụng tên mô-đun đầy đủ.

Cú pháp chung cho cách tiếp cận này là:

```
import module_name as mn
```

7.7.5. Import tất cả các hàm trong module

Chúng ta có thể yêu cầu Python import mọi hàm trong một mô-đun bằng cách sử dụng toán tử dấu hoa thị (*):

```
from pizza import *
make_pizza(16, 'pepperoni')
make_pizza(12, 'mushrooms', 'green peppers', 'extra cheese')
```

Dấu hoa thị trong câu lệnh import yêu cầu Python sao chép mọi hàm từ mô-đun pizza vào tệp chương trình này. Vì mọi hàm đều được import nên ta có thể gọi từng hàm theo tên mà không cần sử dụng ký hiệu dấu chấm. Tuy nhiên, tốt nhất là không nên sử dụng phương pháp này khi đang làm việc với các mô-đun lớn hơn mà ta không viết: nếu mô-đun có tên hàm khớp với tên hiện có trong dự án đang viết, ta có thể nhận được một số kết quả không mong muốn. Python có thể thấy một số hàm hoặc biến có cùng tên và thay vì nhập tất cả các hàm một cách riêng biệt, nó sẽ ghi đè lên các hàm.

Cách tiếp cận tốt nhất là import hàm hoặc các hàm ta muốn hoặc import toàn bộ mô-đun và sử dụng ký hiệu dấu chấm. Điều này dẫn đến mã rõ ràng, dễ đọc và dễ hiểu.

Bài tập chương 7

7-1. Message: Viết một hàm có tên display_message() in ra một câu nói cho mọi người biết ta đang học gì trong chương này. Gọi và đảm bảo rằng thông báo hiển thị chính xác.

7-2. Favorite Book: Viết một hàm được gọi là favourite_book() chấp nhận một tham số, tiêu đề. Hàm sẽ in một tin nhắn, chẳng hạn như Một trong những cuốn sách yêu thích là Alice in Wonderland. Gọi hàm, đảm bảo bao gồm tên sách làm đối số trong lệnh gọi hàm

7-3. T-Shirt: Viết một hàm có tên make_shirt() chấp nhận kích thước và nội dung của thông điệp sẽ được in trên áo. Hàm cần in câu tóm tắt size áo và thông điệp in trên đó. Gọi hàm lần thứ nhất bằng cách sử dụng các đối số vị trí để tạo một cái áo. Gọi hàm lần thứ hai bằng cách sử dụng các đối số từ khóa.

7-4. Large Shirts: Sửa đổi hàm make_shirt() để áo sơ mi có kích thước lớn theo mặc định với thông điệp có nội dung I love Python. Tạo một chiếc áo sơ mi lớn

và một chiếc áo sơ mi vừa với thông điệp mặc định và một chiếc áo sơ mi có kích thước bất kỳ với thông điệp khác.

7-5. Cities: Viết một hàm được gọi là `description_city()` chấp nhận tên của một thành phố và đất nước của thành phố. Hàm sẽ in ra một câu kiểu như: *Reykjavik is in Iceland*. Cho tham số `country` giá trị mặc định. Gọi hàm với ba thành phố khác nhau và ít nhất một lời gọi hàm không sử dụng giá trị mặc định.

7-6. City Names: Viết một hàm có tên `city_country()` lấy tên của một thành phố và quốc gia của nó. Hàm sẽ trả về một chuỗi có định dạng như sau:

"`Santiago, Chile`"

Gọi hàm của ta với ít nhất ba cặp thành phố-quốc gia và in các giá trị được trả về.

7-7. Album: Viết một hàm có tên `make_album()` để xây dựng một từ điển mô tả một album nhạc. Hàm sẽ nhận tên nghệ sĩ và tiêu đề album và nó sẽ trả về một từ điển chứa hai phần thông tin này. Sử dụng chức năng này để tạo ba từ điển đại diện cho các album khác nhau. In từng giá trị trả về để cho thấy rằng từ điển đang lưu trữ thông tin album một cách chính xác. Sử dụng `None` có để thêm một tham số tùy chọn vào `make_album()` cho phép lưu trữ số lượng bài hát trong một album. Nếu gọi hàm bao gồm một giá trị cho số lượng bài hát, hãy thêm giá trị đó vào từ điển của album. Thực hiện ít nhất một lệnh gọi chức năng mới bao gồm số lượng bài hát trong album

7-8. User Albums: Bắt đầu với chương trình tại bài 7-7. Xây dựng vòng lặp `while` cho phép nhập tác giả và tiêu đề của album nhạc. Sau khi ta có thông tin đó, hãy gọi `make_album()` với thông tin nhập của người dùng và in từ điển đã tạo. Đảm bảo bao gồm một giá trị thoát trong vòng lặp `while`.

7-9. Messages: Tạo danh sách chứa một loạt tin nhắn văn bản ngắn. Chuyển danh sách đến một hàm có tên `show_messages()`, hàm này sẽ in từng tin nhắn văn bản.

7-10. Sending Messages: Bắt đầu với chương trình trong bài 7-9. Viết một hàm tên là `send_message()` in ra thông điệp chuỗi và chuyển các thông điệp đã in ra vào danh sách `sent_messages`. Sau khi gọi hàm, hãy in cả hai danh sách của ta để đảm bảo rằng các tin nhắn đã di chuyển một cách chính xác.

7-11. Archived Messages: Bắt đầu với chương trình trong bài 7-10. Gọi hàm send_messages() với một bản sao của danh sách messages. Sau khi gọi hàm, hãy in cả hai danh sách của ta để cho thấy rằng danh sách ban đầu vẫn giữ nguyên tin nhắn.

7-12. Sandwiches: Viết một hàm chấp nhận danh sách các mục mà một người muốn có trên bánh mì sandwich. Hàm phải có một tham số thu thập nhiều mục như lời gọi hàm cung cấp và nó sẽ in ra một bản tóm tắt về bánh sandwich đang được đặt hàng. Gọi hàm ba lần, mỗi lần sử dụng một số đối số khác nhau.

7-13. User Profile: Bắt đầu với bản sao của user_profile.py trong bài học. Xây dựng hồ sơ của chính ta bằng cách gọi build_profile(), sử dụng họ và tên của ta và ba cặp khóa-giá trị khác mô tả ta.

7-14. Cars: Viết một hàm lưu thông tin về ô tô trong từ điển. Hàm phải luôn nhận được nhà sản xuất và tên kiểu máy. Sau đó, nó sẽ chấp nhận một số lượng đối số từ khóa tùy ý. Gọi hàm với thông tin bắt buộc và hai cặp tên-giá trị khác, chẳng hạn như màu hoặc tính năng tùy chọn. Hàm của ta sẽ hoạt động cho một lời gọi như sau:

```
car = make_car('subaru', 'outback', color='blue', tow_package=True)
```

In từ điển được trả lại để đảm bảo tất cả thông tin đều được lưu trữ một cách chính xác.

7-15. Printing Models: Đặt các hàm cho ví dụ print_models.py trong một tệp riêng có tên là print_functions.py. Viết câu lệnh nhập ở đầu print_models.py và sửa đổi tệp để sử dụng các hàm đã nhập.

7-16. Imports: Sử dụng một chương trình ta đã viết có một chức năng trong đó, hãy lưu trữ chức năng đó trong một tệp riêng biệt. Nhập hàm vào tệp chương trình chính của ta và gọi hàm bằng cách sử dụng từng phương pháp sau:

```
import module_name
from module_name import function_name
from module_name import function_name as fn
import module_name as mn
from module_name import *
```

Kết chương

Trong chương này, chúng ta đã học cách viết hàm và truyền đối số để các hàm có quyền truy cập vào thông tin cần thiết để thực hiện công việc của hàm. Ta đã học

cách sử dụng các đối số vị trí và từ khóa cũng như cách để chấp nhận một số lượng đối số tùy ý. Ta đã thấy các hàm hiển thị đầu ra và các hàm trả về giá trị. Chúng ta đã học cách sử dụng các hàm với danh sách, từ điển, câu lệnh if và vòng lặp while. Bạn cũng đã thấy cách lưu trữ các hàm đang viết trong các tệp riêng biệt được gọi là môđun, do đó, chương trình sẽ đơn giản và dễ hiểu hơn.

Một trong những mục tiêu của một lập trình viên là viết mã đơn giản làm những gì ta muốn và các chức năng giúp ta làm điều này. Hàm cho phép ta viết các khối mã và để chúng một mình và biết chúng hoạt động. Khi nào chúng ta biết một chức năng thực hiện đúng công việc của nó, ta có thể tin tưởng rằng và ta có thể chuyển sang một công việc tiếp theo trong dự án

Các hàm cho phép ta viết mã một lần và sau đó sử dụng lại mã đó bất kỳ lúc nào ta muốn. Khi ta cần chạy mã trong một hàm, tất cả những gì cần làm là viết một lệnh gọi một dòng và hàm thực hiện công việc của nó. Khi ta cần sửa đổi hành vi của một hàm, ta chỉ phải sửa đổi một khối mã và thay đổi có hiệu lực ở mọi nơi đã thực hiện lời gọi đến hàm đó.

Việc sử dụng các hàm giúp chương trình dễ đọc hơn và các tên hàm tốt sẽ tóm tắt chức năng của mỗi phần trong chương trình. Đọc một chuỗi lệnh gọi hàm cung cấp cho ta cảm giác nhanh hơn nhiều về những gì một chương trình thực hiện hơn là đọc một loạt các khối mã dài.

Các hàm cũng giúp mã nguồn dễ kiểm tra và gỡ lỗi hơn. Khi số lượng lớn công việc trong chương trình được thực hiện bởi một tập hợp các hàm, mỗi hàm trong số đó có công việc cụ thể, việc kiểm tra và duy trì mã nguồn đã viết sẽ dễ dàng hơn nhiều. Ta có thể viết một chương trình riêng biệt gọi từng hàm và kiểm tra xem mỗi hàm hoạt động trong tất cả các tình huống mà nó có thể gặp phải. Khi ta làm điều này, có thể yên tâm rằng các hàm sẽ hoạt động bình thường mỗi lần được gọi.

Trong Chương 8, ta sẽ học cách viết các lớp. Các lớp kết hợp các hàm và dữ liệu thành một gói gọn gàng có thể được sử dụng theo những cách hữu ích và hiệu quả.

CHƯƠNG 8. LỚP

Lập trình hướng đối tượng là một trong những cách hiệu quả nhất khi viết phần mềm. Trong lập trình hướng đối tượng, ta viết ra những lớp (classes) trong lập trình hướng đối tượng, ta viết các lớp đại diện cho những thứ trong thế giới thực và các tình huống, đồng thời ta tạo các đối tượng dựa trên những lớp đó. Khi viết một lớp, ta xác định hành vi chung mà tất cả các đối tượng có thể có.

Khi ta tạo các đối tượng riêng lẻ từ lớp, mỗi đối tượng sẽ tự động được trang bị hành vi chung; sau đó ta có thể cung cấp cho từng đối tượng bất cứ đặc điểm riêng biệt nào mà ta mong muốn. Chúng ta sẽ ngạc nhiên về các tình huống trong thế giới thực có thể được mô hình hóa tốt như thế nào bằng lập trình hướng đối tượng. Tạo một đối tượng từ một lớp được gọi là khởi tạo và ta làm việc với các thể hiện (instances) của một lớp.

Trong chương này, chúng ta sẽ viết các lớp và tạo các thể hiện của các lớp đó. Ta sẽ chỉ định loại thông tin có thể được lưu trữ trong các thể hiện và sẽ xác định các hành động có thể được thực hiện với các thể hiện này. Ta cũng sẽ viết các lớp kế thừa chức năng của các lớp hiện có, vì vậy các lớp tương tự có thể chia sẻ mã một cách hiệu quả. Ta sẽ lưu trữ các lớp trong các mô-đun và import các lớp do các lập trình viên khác viết vào các tệp chương trình.

Hiểu lập trình hướng đối tượng sẽ giúp nhìn thấy thế giới của một lập trình viên. Nó sẽ giúp chúng ta thực sự hiểu mã của mình, không chỉ những gì đang xảy ra từng dòng một mà còn cả những khái niệm lớn hơn đằng sau nó.

Hiểu logic đằng sau các lớp sẽ giúp rèn luyện cách suy nghĩ logic để chúng ta có thể viết các chương trình giải quyết hiệu quả hầu hết mọi vấn đề gặp phải.

Các lớp cũng giúp cuộc sống của chúng và các lập trình viên khác trở nên dễ dàng hơn trong khi làm việc và khi đối mặt với những thách thức ngày càng phức tạp. Khi chúng ta và các lập trình viên khác viết mã dựa trên cùng một loại logic, chúng ta sẽ có thể hiểu công việc của nhau. Chương trình cũng sẽ có ý nghĩa với các cộng tác viên, cho phép người dùng hoàn thành công việc được nhiều hơn.

8.1. Tạo và sử dụng lớp

Chúng ta có thể mô hình hóa hầu hết mọi thứ bằng cách sử dụng các lớp. Hãy bắt đầu bằng cách viết một lớp, Dog, đại diện cho một con chó — không phải một con chó cụ thể, mà là bất kỳ con chó nào. Chúng ta biết gì về hầu hết những con chó cưng?

Tất cả đều có tên và tuổi.

Chúng ta cũng biết rằng hầu hết các con chó đều ngồi và cuộn lại. Hai phần thông tin (tên và tuổi) và hai hành vi đó (ngồi và cuộn lại) sẽ có trong lớp Dog của vì chúng phỏ biến với hầu hết các loài chó. Lớp này sẽ nói Python cách để tạo một đối tượng đại diện cho một con chó. Sau khi lớp của chúng ta được xây dựng, chúng ta sẽ sử dụng nó để tạo các phiên bản riêng lẻ, mỗi phiên bản đại diện cho một con chó cụ thể.

8.1.1. Tạo ra lớp Dog

Mỗi cá thể được tạo từ lớp Dog sẽ lưu trữ tên và tuổi, và chúng ta sẽ cung cấp cho mỗi chú chó khả năng sit() và roll_over():

```
① class Dog:  
②     """A simple attempt to model a dog."""  
③     def __init__(self, name, age):  
④         self.name = name  
        self.age = age  
⑤     def sit(self):  
        """Simulate a dog sitting in response to a command."""  
        print(f"{self.name} is now sitting.")  
    def roll_over(self):  
        """Simulate rolling over in response to a command."""  
        print(f"{self.name} rolled over!")
```

Tại ①, chúng ta định nghĩa một lớp được gọi là Dog. Theo quy ước, các tên viết hoa đè cập đến các lớp bằng Python. Không có dấu ngoặc đơn nào trong định dạng lớp bởi vì chúng ta tạo lớp này từ đầu. Tại ②, chúng ta viết một chuỗi tài liệu mô tả những gì lớp này thực hiện.

Phương thức __init__()

Một hàm trong lớp được gọi là phương thức. Tất cả các nội dung ta đã học trong hàm đều áp dụng được cho phương thức. Sự khác biệt thực tế duy nhất cho bây

giờ là c chúng ta sẽ gọi hàm là các phương thức. Phương thức `__init__()` tại ③ là một phương thức đặc biệt và Python chạy tự động bắt cứ khi nào chúng ta tạo một thể hiện mới dựa trên lớp Dog. Phương thức này có hai dấu gạch dưới ở đầu và hai dấu gạch dưới ở cuối, một quy ước giúp ngăn chặn phương thức mặc định của Python trùng tên với phương thức khác của lập trình viên. Đảm bảo sử dụng hai gạch dưới ở mỗi bên của `__init__()`. Nếu ta chỉ sử dụng một ở mỗi bên, phương thức sẽ không được gọi tự động khi ta sử dụng lớp, điều này có thể dẫn đến lỗi khó xác định.

Chúng ta định nghĩa phương thức `__init__()` để có ba tham số: `self`, `name`, và `age`. Tham số `self` là bắt buộc trong quá trình xác định phương thức và nó phải đến đầu tiên trước các tham số khác. Nó phải được bao gồm trong định nghĩa vì khi Python gọi phương thức này sau đó (để tạo một phiên bản của Dog), lệnh gọi phương thức sẽ tự động chuyển đổi số tự động. Mỗi lời gọi phương thức được liên kết với một thể hiện tự động truyền biến `self`, đó là một tham chiếu đến chính cá thể đó; nó cung cấp cho cá thể quyền truy cập vào các thuộc tính và phương thức trong lớp. Khi chúng ta tạo một thể hiện về Dog, Python sẽ gọi phương thức `__init__()` từ lớp Dog. Chúng ta sẽ truyền cho phương thức khởi tạo `Dog()` tên và tuổi làm đối số; `self` tự động được truyền, vì vậy ta không cần phải truyền nó. Bắt cứ khi nào chúng ta muốn tạo một thể hiện từ lớp Dog, chúng ta sẽ chỉ cung cấp giá trị cho hai thông số cuối cùng, `name` và `age`.

Hai biến được định nghĩa tại ④ có tiền tố `self`. Bất kỳ biến tiền tố là `self` khả dụng cho mọi phương thức trong lớp và chúng ta cũng có thể truy cập các biến này thông qua bất kỳ thể hiện nào được tạo từ lớp. Dòng `self.name = name` nhận giá trị được liên kết với tên tham số và gán nó cho tên biến, sau đó được gắn vào thể hiện đang được tạo. Quá trình tương tự cũng xảy ra với `self.age = age`. Các biến mà có thể truy cập thông qua các thể hiện như thế này được gọi là thuộc tính. Lớp Dog có hai phương thức khác được định nghĩa: `sit()` và `roll_over()` ⑤. Vì các phương thức này không cần thêm thông tin để chạy, chúng ta chỉ xác định chúng có một tham số là `self`. Các thể hiện chúng ta tạo sau này sẽ có quyền truy cập vào các phương thức này. Nói cách khác, các chú chó cụ thể có thể ngồi và lăn lộn. Hiện tại, `sit()` và `roll_over()` không có tác dụng gì nhiều. Chúng chỉ đơn giản là in thông báo cho biết con chó đang ngồi hoặc lăn qua. Nhưng khái niệm có thể là mở rộng cho các tình huống thực tế: nếu lớp này là một phần của một trò chơi máy tính thực tế, thì các phương thức này

sẽ chứa mã để tạo ra một con chó hoạt hình ngồi và lăn lộn. Nếu lớp này được viết để điều khiển robot, các phương thức này sẽ chỉ đạo các chuyển động khiến một con chó robot ngồi và lăn.

8.1.2. Tạo một thể hiện của lớp

Hãy nghĩ về một lớp như một tập hợp các câu lệnh về cách tạo một thể hiện. Các lớp Dog là một tập hợp các câu lệnh cho Python biết cách tạo các thể hiện đại diện cho những con chó cụ thể.

Đoạn sau tạo ra thể hiện cho một chú chó cụ thể:

```
class Dog:  
    --snip--  
    ①     my_dog = Dog('Willie', 6)  
    ②     print(f"My dog's name is {my_dog.name}.")  
    ③     print(f"My dog is {my_dog.age} years old.")
```

Lớp Dog mà chúng ta đang sử dụng ở đây là lớp ta vừa viết trong phần trước. Tại ①, chúng ta yêu cầu Python tạo ra một con chó có tên là 'Willie' và có tuổi là 6. Khi Python đọc dòng này, nó gọi phương thức `__init__()` trong Dog với các đối số 'Willie' và 6. Phương thức `__init__()` tạo ra một thể hiện cho con chó cụ thể này và đặt các thuộc tính tên và tuổi sử dụng các giá trị mà chúng ta đã cung cấp. Python sau đó trả về một thể hiện đại diện cho con chó này. Chúng ta gán thể hiện đó cho biến `my_dog`. Quy ước đặt tên rất hữu ích ở đây: chúng ta thường có thể cho rằng tên viết hoa như Dog đề cập đến một lớp và tên viết thường như `my_dog` đề cập đến một thể hiện duy nhất được tạo ra từ một lớp.

Truy cập các thuộc tính

Để truy cập các thuộc tính của một thể hiện ta sử dụng ký hiệu dấu chấm. Tại ② chúng ta truy cập giá trị của tên thuộc tính của `my_dog` bằng cách viết:

```
my_dog.name
```

Ký hiệu dấu chấm thường được sử dụng trong Python. Cú pháp này trình bày cách Python tìm giá trị của thuộc tính. Ở đây Python xem xét thể hiện `my_dog` và sau đó tìm tên thuộc tính được liên kết với `my_dog`. Đây là thuộc tính tương tự được gọi là `self.name` trong lớp Dog. Tại ③, chúng ta sử dụng cùng một cách tiếp cận để làm việc với thuộc tính `age`.

Kết quả được in ra nhưng gì chúng ta hiểu biết về `my_dog`:

My dog's name is Willie.

My dog is 6 years old.

Gọi phương thức

Sau khi tạo một thể hiện từ lớp Dog, chúng ta có thể sử dụng ký hiệu dấu chấm để gọi bất kỳ phương thức nào được định nghĩa trong Dog. Ta có thể bắt con chó của chúng ta ngồi và lăn lại:

```
class Dog:  
    --snip--  
    my_dog = Dog('Willie', 6)  
    my_dog.sit()  
    my_dog.roll_over()
```

Để gọi một phương thức, ta cung cấp tên của thể hiện (trong trường hợp này là my_dog) và phương thức ta muốn gọi, được phân tách bằng dấu chấm. Khi Python đọc my_dog.sit(), nó tìm kiếm phương thức sit() trong lớp Dog và chạy mã. Python diễn giải dòng my_dog.roll_over() theo cách tương tự.

Bây giờ Willie làm những gì chúng ta nói với nó:

Willie is now sitting.

Willie rolled over!

Cú pháp này khá hữu ích. Khi các thuộc tính và phương thức đã đưa ra các tên mô tả thích hợp như name, age, sit() và roll_over(), chúng ta có thể dễ dàng suy ra khói mã nào, thậm chí là khói mà chúng ta chưa từng thấy trước đây.

Tạo ra nhiều thể hiện

Ta có thể tạo bao nhiêu thể hiện từ một lớp nếu muốn. Đoạn mã tạo một con chó thứ hai được gọi là your_dog:

```
class Dog:  
    --snip--  
    my_dog = Dog('Willie', 6)  
    your_dog = Dog('Lucy', 3)  
    print(f"My dog's name is {my_dog.name}.")  
    print(f"My dog is {my_dog.age} years old.")  
    my_dog.sit()  
    print(f"\nYour dog's name is {your_dog.name}.")  
    print(f"Your dog is {your_dog.age} years old.")  
    your_dog.sit()
```

Trong ví dụ này, chúng ta tạo ra một con chó tên là Willie và một con chó tên là Lucy. Mỗi chú chó là một cá thể riêng biệt với tập hợp các thuộc tính riêng của nó nhưng có chung các hoạt động:

My dog's name is Willie.

My dog is 6 years old.

Willie is now sitting.

Your dog's name is Lucy.

Your dog is 3 years old.

Lucy is now sitting.

Ngay cả khi chúng ta sử dụng cùng tên và tuổi cho con chó thứ hai, Python sẽ vẫn tạo một thẻ hiện riêng biệt từ lớp Dog. Chúng ta có thể tạo ra bao nhiêu thẻ hiện từ một lớp nếu muốn, miễn là ta cung cấp từng thẻ hiện một tên biến duy nhất hoặc nó chiếm một vị trí duy nhất trong danh sách hoặc từ điển.

8.2. Làm việc với lớp và thẻ hiện của lớp

Chúng ta có thể sử dụng các lớp để biểu diễn nhiều tình huống trong thế giới thực. Một khi ta viết một lớp, ta sẽ dành phần lớn thời gian của mình để làm việc với các thẻ hiện được tạo từ lớp đó. Một trong những nhiệm vụ đầu tiên ta sẽ muốn làm là sửa đổi các thuộc tính được liên kết với một thẻ hiện cụ thể. Ta có thể sửa đổi các thuộc tính của một thẻ hiện một cách trực tiếp hoặc viết các phương thức cập nhật các thuộc tính theo những cách riêng.

8.2.1. Lớp Car

Hãy viết một lớp mới đại diện cho một chiếc ô tô. Lớp chúng ta sẽ lưu trữ thông tin về loại ô tô mà ta đang làm việc và nó sẽ có một phương thức tóm tắt thông tin này:

```
class Car:  
    """A simple attempt to represent a car."""  
    ①     def __init__(self, make, model, year):  
        """Initialize attributes to describe a car."""  
        self.make = make  
        self.model = model  
        self.year = year  
    ②     def get_descriptive_name(self):  
        """Return a neatly formatted descriptive name."""  
        long_name = f"{self.year} {self.make} {self.model}"  
        return long_name.title()
```

```
(3) my_new_car = Car('audi', 'a4', 2019)
print(my_new_car.get_descriptive_name())
```

Tại ① trong lớp Car, chúng ta xác định phương thức `__init__()` với tự tham số thứ nhất, giống như chúng ta đã làm trước đây với lớp Dog của chúng ta. Chúng ta cũng cho nó ba tham số khác: make, model và year. Phương thức `__init__()` nhận các tham số này và gán chúng cho các thuộc tính sẽ liên kết với các thẻ hiện được tạo từ lớp này. Khi chúng ta tạo ra một thẻ hiện của chiếc ô tô mới, chúng ta sẽ cần chỉ định kiểu dáng, kiểu máy và năm cho thẻ hiện của mình.

Tại ②, chúng ta xác định một phương thức có tên `get_descriptive_name()` để gộp năm sản xuất, chế tạo và mô hình thành một chuỗi mô tả chiếc xe một cách gọn gàng. Điều này sẽ giúp ta không phải in giá trị của từng thuộc tính riêng lẻ. Để làm việc với các giá trị thuộc tính trong phương thức này, chúng ta sử dụng `self.make`, `self.model` và `self.year`.

Tại ③, chúng ta tạo một thẻ hiện từ lớp Car và gán nó cho biến `my_new_car`. Sau đó, chúng ta gọi phương thức `get_descriptive_name()` để hiển thị loại ô tô chúng ta có:

2019 Audi A4

Để làm cho lớp thú vị hơn, ta thêm một thuộc tính thay đổi theo thời gian. Chúng ta sẽ thêm một thuộc tính lưu trữ quãng đường tổng thể của ô tô.

8.2.2. Thiết lập giá trị mặc định cho thuộc tính

Khi một thẻ hiện được tạo, các thuộc tính có thể được xác định mà không cần truyền vào dưới dạng tham số. Các thuộc tính này có thể được định nghĩa trong phương thức `__init__()`, nơi chúng được gán một giá trị mặc định. Hãy thêm một thuộc tính gọi là `odometer_reading` luôn bắt đầu bằng giá trị bằng 0. Chúng ta cũng sẽ thêm phương thức `read_odometer()` để giúp chúng ta đọc đồng hồ đo đường của ô tô:

```
class Car:
    def __init__(self, make, model, year):
        """Initialize attributes to describe a car."""
        self.make = make
        self.model = model
        self.year = year
    ①     self.odometer_reading = 0
    def get_descriptive_name(self):
```

```
--snip--
② def read_odometer(self):
    """Print a statement showing the car's mileage."""
    print(f"This car has {self.odometer_reading} miles on it.")
my_new_car = Car('audi', 'a4', 2019)
print(my_new_car.get_descriptive_name())
my_new_car.read_odometer()
```

Lần này khi Python gọi phương thức `__init__()` để tạo thẻ hiện, nó lưu trữ các giá trị kiểu dáng, kiểu máy và năm dưới dạng các thuộc tính như đã làm trong ví dụ trước. Sau đó, Python tạo một thuộc tính mới được gọi là `odometer_reading` và đặt giá trị ban đầu của nó là 0 ①. Chúng ta cũng có một phương thức mới có tên `read_odometer()` tại ② giúp dễ dàng đọc số dặm mà ô tô đã đi được.

Xe của chúng ta bắt đầu với số dặm là 0:

```
2019 Audi A4
This car has 0 miles on it.
```

Không có nhiêu ô tô được bán ra khi mới chỉ đi được 0 dặm, do đó chúng ta cần một cách để thay đổi giá trị của thuộc tính này.

8.2.3. Thay đổi giá trị thuộc tính

Chúng ta có thể thay đổi giá trị của thuộc tính theo ba cách: có thể thay đổi giá trị trực tiếp thông qua một thẻ hiện, đặt giá trị thông qua một phương thức hoặc gán giá trị (thêm một số lượng nhất định vào nó) thông qua một phương thức. Hãy xem xét từng trong số các cách tiếp cận này.

Thay đổi giá trị thuộc tính trực tiếp

Cách đơn giản nhất để sửa đổi giá trị của một thuộc tính là truy cập trực tiếp vào thuộc tính đó thông qua một thẻ hiện. Ở đây chúng ta đặt số đọc đồng hồ đo đường thành 23 một cách trực tiếp:

```
class Car:
    --snip--
my_new_car = Car('audi', 'a4', 2019)
print(my_new_car.get_descriptive_name())

① my_new_car.odometer_reading = 23
my_new_car.read_odometer()
```

Tại ①, chúng ta sử dụng ký hiệu dấu chấm để truy cập thuộc tính `odometer_reading` của ô tô và đặt trực tiếp giá trị của nó. Dòng này yêu cầu Python

lấy thực thể my_new_car, tìm thuộc tính odometer_reading được liên kết với nó và đặt giá trị của thuộc tính đó thành 23:

```
2019 Audi A4
This car has 23 miles on it.
```

Đôi khi ta muốn truy cập trực tiếp vào các thuộc tính như trên, lúc khác, ta lại muốn viết một phương thức cập nhật giá trị cho các thuộc tính.

Thay đổi giá trị thuộc tính thông qua phương thức

Có thể hữu ích nếu có các phương thức cập nhật các thuộc tính nhất định cho chúng ta. Thay vì truy cập trực tiếp thuộc tính, ta chuyển giá trị mới cho phương thức xử lý cập nhật giá trị nội bộ.

Dưới đây là ví dụ về phương thức được gọi là update_odometer():

```
class Car:
    --snip--
    ①     def update_odometer(self, mileage):
        """Set the odometer reading to the given value."""
        self.odometer_reading = mileage
my_new_car = Car('audi', 'a4', 2019)
print(my_new_car.get_descriptive_name())
②     my_new_car.update_odometer(23)
my_new_car.read_odometer()
```

Thay đổi duy nhất cho lớp Car là bổ sung phương thức update_odometer() tại ①. Phương thức này nhận một giá trị số dặm và gán nó cho self.odometer_reading. Tại ②, chúng ta gọi update_odometer() và cung cấp cho nó 23 làm đối số (tương ứng với tham số số dặm trong định nghĩa phương thức). Nó thiết lập đồng hồ đo đường đến 23 và phương thức read_odometer() in kết quả:

```
2019 Audi A4
This car has 23 miles on it.
```

Chúng ta mở rộng phương thức update_odometer() để thực hiện công việc bổ sung mỗi khi số đọc đồng hồ đo đường được thay đổi. Ta thêm một chút logic vào đảm bảo rằng không ai cố gắng quay ngược lại số đọc của đồng hồ đo đường đi:

```
class Car:
    --snip--
    def update_odometer(self, mileage):
        """
            Set the odometer reading to the given value.
            Reject the change if it attempts to roll the odometer back.
        """

```

```

①         if mileage >= self.odometer_reading:
            self.odometer_reading = mileage
        else:
②             print("You can't roll back an odometer!")

```

Bây giờ update_odometer() kiểm tra xem số dặm mới có hợp lý trước đó không khi sửa đổi thuộc tính. Nếu số dặm mới, biến mileage, lớn hơn hoặc bằng với số dặm hiện có, self.odometer_reading, ta có thể cập nhật đồng hồ đo đường đọc số dặm mới ①. Nếu số dặm mới ít hơn số dặm hiện có, sẽ nhận được cảnh báo rằng không thể lùi đồng hồ đo đường ②.

Tăng giá trị của thuộc tính thông qua phương thức

Đôi khi chúng ta muốn tăng giá trị của một thuộc tính lên một khoảng thay vì đặt một giá trị hoàn toàn mới. Giả sử chúng ta mua một chiếc ô tô đã qua sử dụng và tăng giá trị đi được lên 100 dặm giữa khoảng thời gian mua nó với thời gian đăng ký nó. Đây là một phương thức cho phép chúng ta chuyển số giá tăng này và thêm giá trị đó cho số đọc đồng hồ đo đường đi:

```

class Car:
    --snip--
    def update_odometer(self, mileage):
        --snip--
①    def increment_odometer(self, miles):
        """Add the given amount to the odometer reading."""
        self.odometer_reading += miles
②    my_used_car = Car('subaru', 'outback', 2015)
print(my_used_car.get_descriptive_name())
③    my_used_car.update_odometer(23_500)
my_used_car.read_odometer()
④    my_used_car.increment_odometer(100)
my_used_car.read_odometer()

```

Phương thức mới increment_odometer() tại ① nhận giá trị số dặm, và cộng thêm giá trị này vào self.odometer_reading. Tại ②, chúng ta tạo ra một chiếc xe đã qua sử dụng, my_used_car. Chúng ta đặt đồng hồ đo đường của nó thành 23.500 bằng cách gọi update_odometer() và truyền giá trị 23500 tại ③. Tại ④, chúng ta gọi increment_odometer() và chuyển nó 100 để thêm 100 dặm từ lúc chúng ta mua đến lúc đi đăng ký nó:

```

2015 Subaru Outback
This car has 23500 miles on it.
This car has 23600 miles on it.

```

Ta có thể dễ dàng sửa đổi phương thức này để từ chối các giá số âm để ngăn một người sử dụng chức năng này quay lại đồng hồ đo đường.

Ghi chú: Chúng ta có thể sử dụng các phương pháp như thế này để kiểm soát cách người dùng chương trình cập nhật các giá trị chặng hạn như đọc đồng hồ đo đường, nhưng bất kỳ ai có quyền truy cập vào chương trình đều có thể đặt số đọc đồng hồ đo đường thành bất kỳ giá trị nào bằng cách truy cập trực tiếp vào thuộc tính. Do vậy, cần phải đảm bảo việc bảo mật và phân quyền các quyền về cập nhật các giá trị đó.

8.3. Kế thừa

Không phải lúc nào chúng ta cũng phải bắt đầu lại từ đầu khi viết một lớp. Nếu lớp ta đang viết là một phiên bản chuyên biệt của một lớp khác mà đã được viết, ta có thể sử dụng kế thừa. Khi một lớp kế thừa từ lớp khác, nó sẽ sử dụng các thuộc tính và phương thức của lớp thứ nhất. Lớp ban đầu được gọi là lớp cha, và lớp mới là lớp con. Lớp con có thể kế thừa bất kỳ hoặc tất cả các thuộc tính và phương thức của lớp cha của nó, nhưng cũng có thể tự định nghĩa các thuộc tính và phương thức mới của riêng nó.

8.3.1. Phương thức `__init__()` cho lớp con

Khi ta đang viết một lớp mới dựa trên một lớp hiện có, ta sẽ thường muốn gọi phương thức `__init__()` từ lớp cha. Điều này sẽ khởi tạo bất kỳ thuộc tính nào đã được định nghĩa trong phương thức `__init__()` của lớp cha và làm các thuộc tính có sẵn trong lớp con.

Ví dụ, hãy mô hình hóa một chiếc ô tô điện. Ô tô điện chỉ là một loại ô tô cụ thể, vì vậy chúng ta có thể dựa xây dựng lớp ElectricCar dựa trên lớp Car đã được xây dựng trước đó. Sau đó, chúng ta sẽ chỉ phải viết mã cho các thuộc tính và hành vi cụ thể đối với ô tô điện.

Hãy bắt đầu bằng cách tạo một phiên bản đơn giản của lớp ElectricCar, thực hiện mọi thứ mà lớp Car làm:

```
class Car:  
    """A simple attempt to represent a car."""  
    def __init__(self, make, model, year):  
        self.make = make  
        self.model = model
```

```

        self.year = year
        self.odometer_reading = 0
    def get_descriptive_name(self):
        long_name = f"{self.year} {self.make} {self.model}"
        return long_name.title()
    def read_odometer(self):
        print(f"This car has {self.odometer_reading} miles on it.")
    def update_odometer(self, mileage):
        if mileage >= self.odometer_reading:
            self.odometer_reading = mileage
        else:
            print("You can't roll back an odometer!")
    def increment_odometer(self, miles):
        self.odometer_reading += miles

```

```

① class ElectricCar(Car):
    """Represent aspects of a car, specific to electric vehicles."""
②     def __init__(self, make, model, year):
        """Initialize attributes of the parent class."""
③         super().__init__(make, model, year)
④ my_tesla = ElectricCar('tesla', 'model s', 2019)
print(my_tesla.get_descriptive_name())

```

Đầu tiên, chúng ta bắt đầu với lớp Car. Khi tạo một lớp con, lớp cha phải là một phần của tệp hiện tại và phải xuất hiện trước lớp con trong tập tin. Tại ①, chúng ta xác định lớp con, ElectricCar. Tên của lớp cha phải được bao gồm trong dấu ngoặc đơn trong định nghĩa của một lớp con. Phương thức `__init__()` tại ② lấy thông tin cần thiết để tạo một thể hiện của lớp Car. Hàm super tại ③ là một chức năng đặc biệt cho phép gọi một phương thức từ lớp cha. Dòng này yêu cầu Python gọi `__init__()` từ Car, cung cấp cho một thể hiện ElectricCar tất cả các thuộc tính xác định trong phương thức đó. Tên super xuất phát từ quy ước gọi lớp cha là superclass và lớp con là subclass.

Chúng ta kiểm tra xem kế thừa có hoạt động bình thường hay không bằng cách cố gắng tạo ô tô điện với cùng loại thông tin mà chúng ta sẽ cung cấp khi sản xuất một chiếc ô tô thông thường. Tại ④, chúng ta tạo một thể hiện của lớp ElectricCar và gán nó vào `my_tesla`. Dòng này gọi phương thức `__init__()` được định nghĩa trong ElectricCar, đến lượt nó ra lệnh cho Python gọi phương thức `__init__()` được định nghĩa trong lớp cha Car. Chúng ta cung cấp các đối số 'tesla', 'model s' và 2019. Ngoài `__init__()`, không có thuộc tính hoặc phương thức nào đặc biệt của một chiếc ô tô

điện. Tại thời điểm này, chúng ta chỉ đảm bảo ô tô điện có các hành vi tương ứng của ô tô:

2019 Tesla Model S

Thực thể ElectricCar hoạt động giống như một thực thể Car, vì vậy bây giờ chúng ta có thể bắt đầu xác định các thuộc tính và phương thức cụ thể cho ô tô điện.

8.3.2. Định nghĩa các thuộc tính và phương thức cho lớp con

Khi ta có một lớp con kế thừa từ một lớp cha, ta có thể thêm bất kỳ thuộc tính và phương thức mới nào cần thiết để phân biệt lớp con với lớp cha.

Thêm một thuộc tính đặc biệt cho ô tô điện (ví dụ là pin) và một phương thức để thông báo về thuộc tính này. Chúng ta sẽ lưu trữ kích thước pin và viết phương thức prints mô tả về pin:

```
class Car:  
    --snip--  
class ElectricCar(Car):  
    """Represent aspects of a car, specific to electric vehicles."""  
    def __init__(self, make, model, year):  
        """  
        Initialize attributes of the parent class.  
        Then initialize attributes specific to an electric car.  
        """  
        super().__init__(make, model, year)  
        self.battery_size = 75  
    ②    def describe_battery(self):  
        """Print a statement describing the battery size."""  
        print(f"This car has a {self.battery_size}-kWh battery.")  
my_tesla = ElectricCar('tesla', 'model s', 2019)  
print(my_tesla.get_descriptive_name())  
my_tesla.describe_battery()
```

Tại ①, chúng ta thêm một thuộc tính mới self.battery_size và đặt giá trị ban đầu của nó thành 75. Thuộc tính này sẽ được liên kết với tất cả các thể hiện được tạo từ Lớp ElectricCar nhưng sẽ không được liên kết với bất kỳ thể hiện nào của lớp Car. Chúng ta cũng thêm một phương thức được gọi là description_battery() để in thông tin về pin ở ②. Khi ta gọi phương thức này, ta nhận được mô tả rõ ràng cụ thể cho một chiếc ô tô điện:

2019 Tesla Model S

This car has a 75-kWh battery.

Không có giới hạn về số lượng ta có thể chuyên biệt hóa lớp ElectricCar. Ta có thể thêm nhiều thuộc tính và phương thức tùy ý để tạo mô hình một chiếc ô tô điện ở bất kỳ mức độ chính xác nào ta cần. Nếu một thuộc tính hoặc phương thức có thể thuộc về bất kỳ chiếc ô tô nào, dành riêng cho xe điện, nên được thêm vào lớp Car thay vì lớp ElectricCar. Sau đó bất kỳ ai sử dụng lớp Car cũng sẽ có sẵn chức năng đó, và lớp ElectricCar sẽ chỉ chứa mã cho thông tin và hành vi cụ thể đối với xe điện.

8.3.3. Ghi đè phương thức từ lớp cha

Chúng ta có thể ghi đè bất kỳ phương thức nào từ lớp cha mà không phù hợp với mô hình chúng ta muốn với lớp con. Để làm điều này, ta định nghĩa một phương thức trong lớp con có cùng tên với phương thức muốn ghi đè của lớp cha. Python sẽ bỏ qua phương thức của lớp cha và chỉ chú ý đến phương thức ta định nghĩa trong lớp con.

Giả sử lớp Car có một phương thức gọi là `fill_gas_tank()`. Phương thức này là vô nghĩa đối với một chiếc xe chạy hoàn toàn bằng điện, vì vậy ta có thể muốn ghi đè phương thức này. Đây là một cách để làm điều đó:

```
class ElectricCar(Car):
    --snip--
    def fill_gas_tank(self):
        """Electric cars don't have gas tanks."""
        print("This car doesn't need a gas tank!")
```

Bây giờ nếu ai đó cố gắng gọi `fill_gas_tank()` bằng một chiếc ô tô điện, Python sẽ bỏ qua phương thức `fill_gas_tank()` trong lớp Car và chạy mã này thay thế. Khi nào ta sử dụng kế thừa, ta có thể làm cho các lớp con giữ lại những gì cần thiết và ghi đè bất kỳ thứ không cần từ lớp cha.

8.3.4. Thuộc tính là một thẻ hiện lớp

Khi lập mô hình thứ gì đó từ thế giới thực trong mã, có thể thấy rằng ta đang thêm ngày càng nhiều chi tiết hơn vào một lớp. Ta sẽ thấy rằng ta có một danh sách các thuộc tính và phương thức ngày càng tăng và các tệp được viết đang trở thành dài dòng. Trong những tình huống này, ta có thể nhận ra rằng một phần của một lớp có thể được viết như một lớp riêng biệt. Ta có thể chia lớp lớn của mình thành các lớp nhỏ hơn làm việc cùng nhau.

Ví dụ: nếu chúng ta tiếp tục thêm chi tiết vào lớp ElectricCar, có thể nhận thấy rằng ta đang thêm nhiều thuộc tính và phương thức cụ thể cho pin của ô tô. Khi chúng ta thấy điều này xảy ra, có thể dừng lại và di chuyển các thuộc tính và phương thức cho một lớp riêng biệt được gọi là Battery. Sau đó, chúng ta có thể sử dụng một thẻ hiện của Battery như một thuộc tính trong lớp ElectricCar:

```
class Car:  
    --snip--  
①    class Battery:  
        """A simple attempt to model a battery for an electric car."""  
②        def __init__(self, battery_size=75):  
            """Initialize the battery's attributes."""  
            self.battery_size = battery_size  
③        def describe_battery(self):  
            """Print a statement describing the battery size."""  
            print(f"This car has a {self.battery_size}-kWh battery.")  
  
class ElectricCar(Car):  
    """Represent aspects of a car, specific to electric vehicles."""  
    def __init__(self, make, model, year):  
        """  
        Initialize attributes of the parent class.  
        Then initialize attributes specific to an electric car.  
        """  
        super().__init__(make, model, year)  
④        self.battery = Battery()  
my_tesla = ElectricCar('tesla', 'model s', 2019)  
print(my_tesla.get_descriptive_name())  
my_tesla.battery.describe_battery()
```

Tại ①, chúng ta định nghĩa một lớp mới có tên là Battery không kế thừa từ bất kỳ lớp khác. Phương thức `__init__()` tại ② có một tham số, `pin_size`, tham số `self`. Đây là một thông số tùy chọn đặt kích thước của pin thành 75 nếu không có giá trị nào được cung cấp. Phương thức `description_battery()` đã có cũng chuyển đến lớp này ③. Trong lớp ElectricCar, bây giờ chúng ta thêm một thuộc tính gọi là `self.battery` ④.

Dòng này yêu cầu Python tạo một thẻ hiện mới của Battery (với kích thước mặc định là 75, bởi vì chúng ta không chỉ định một giá trị) và chỉ định thực thể đó cho thuộc tính `self.battery` của lớp ElectricCar. Điều này sẽ xảy ra mỗi khi phương

thức `__init__()` được gọi ; bất kỳ thê hiện nào của ElectricCar bây giờ sẽ có thực thê Battery được tạo tự động.

Chúng ta tạo một chiếc ô tô điện và gán nó vào biến `my_tesla`. Khi nào chúng ta muốn mô tả pin, ta cần làm việc thông qua thuộc tính pin của ô tô:

```
my_tesla.battery.describe_battery()
```

Dòng này yêu cầu Python xem xét thực thê `my_tesla`, tìm thuộc tính `battery` của nó và gọi phương thức `describe_battery()` được liên kết với thực thê `Battery` được lưu trữ trong thuộc tính. Đầu ra giống với những gì chúng ta đã thấy trước đây:

```
2019 Tesla Model S
This car has a 75-kWh battery.
```

Điều này trông giống như rất nhiều công việc bổ sung, nhưng bây giờ chúng ta có thể mô tả pin chi tiết như mong muốn mà không làm lộn xộn lớp `ElectricCar`. Hãy thêm một phương thức khác vào `Battery` báo cáo phạm vi hoạt động của ô tô dựa trên kích thước pin:

```
class Car:
    --snip--
class Battery:
    --snip--
①     def get_range(self):
        """Print a statement about the range this battery provides."""
        if self.battery_size == 75:
            range = 260
        elif self.battery_size == 100:
            range = 315
        print(f"This car can go about {range} miles on a full charge.")
class ElectricCar(Car):
    --snip--
my_tesla = ElectricCar('tesla', 'model s', 2019)
print(my_tesla.get_descriptive_name())
my_tesla.battery.describe_battery()
②     my_tesla.battery.get_range()
```

Phương thức mới `get_range()` tại ① thực hiện một số phân tích đơn giản. Nếu năng lực pin là 75 kWh, `get_range()` thiết lập `range` 260 dặm, và nếu năng lực pin là 100 kWh, nó thiết lập `range` thành 315 dặm. Sau đó nó thông báo giá trị này. Khi chúng ta muốn sử dụng phương thức này, chúng ta lại phải gọi nó qua thuộc tính `battery` của ô tô tại ②.

Đầu ra cho chúng ta biết phạm vi hoạt động của xe dựa trên kích thước pin của nó:

```
2019 Tesla Model S
This car has a 75-kWh battery.
This car can go about 260 miles on a full charge.
```

8.4. Sử dụng các lớp

Khi thêm nhiều chức năng hơn vào các lớp, các tệp lưu trữ mã nguồn có thể dài ra, ngay cả khi ta đã sử dụng kế thừa đúng cách. Để phù hợp với triết lý tổng thể của Python, ta sẽ muốn giữ các tệp mã nguồn gọn gàng nhất có thể. Để trợ giúp, Python cho phép lưu trữ các lớp trong các mô-đun và sau đó nhập các lớp cần thiết vào chương trình chính.

8.4.1. Nhập vào một lớp đơn

Chúng ta cần tạo một mô-đun chỉ chứa lớp Car. Điều này mang lại một vấn đề về đặt tên: chúng ta đã có một tệp có tên car.py trong chương này, mô-đun này phải được đặt tên là car.py vì nó chứa mã đại diện cho một chiếc ô tô. Chúng ta sẽ giải quyết vấn đề đặt tên này bằng cách lưu trữ lớp Car trong một mô-đun có tên car.py, thay thế tệp car.py chúng ta đã sử dụng trước đây. Kể từ thời điểm này, bất kỳ chương trình sử dụng mô-đun này sẽ cần một tên tệp cụ thể hơn, chẳng hạn như my_car.py. Dưới đây là car.py chỉ với mã từ Car class:

```
"""A class that can be used to represent a car."""
class Car:
    """A simple attempt to represent a car."""
    def __init__(self, make, model, year):
        """Initialize attributes to describe a car."""
        self.make = make
        self.model = model
        self.year = year
        self.odometer_reading = 100

    def get_descriptive_name(self):
        """Return a neatly formatted descriptive name."""
        long_name = f"{self.year} {self.model} {self.model}"
        return long_name.title()

    def read_odometer(self):
        """Print a statement showing the car's mileage."""
        print(f"This car has {self.odometer_reading} miles on it.")
```

```

def update_odometer(self, mileage):
    """
        Set the odometer reading to the given value.
        Reject the change if it attempts to roll the odometer back.
    """
    if mileage >= self.odometer_reading:
        self.odometer_reading = mileage
    else:
        print("You can't roll back an odometer!")

def increment_odometer(self, miles):
    """Add the given amount to the odometer reading."""
    self.odometer_reading += miles

```

Đầu tiên, ta thêm một chuỗi tài liệu cấp mô-đun tóm tắt mô tả nội dung của mô-đun này. Chúng ta nên viết một chuỗi tài liệu cho mỗi mô-đun khi tạo ra. Bây giờ chúng ta tạo một tệp riêng có tên là `my_car.py`. Tệp này sẽ nhập lớp Car và sau đó tạo một thể hiện từ lớp đó:

```

from car import Car
my_new_car = Car('audi', 'a4', 2019)
print(my_new_car.get_descriptive_name())
my_new_car.odometer_reading = 23
my_new_car.read_odometer()

```

Câu lệnh import yêu cầu Python mở module car và import lớp Car. Sau đó, ta có thể sử dụng lớp Car như là ta đã định nghĩa trong tệp hiện tại. Chương trình in ra kết quả tương tự như phần trên:

```

2019 Audi A4
This car has 23 miles on it.

```

Nhập các lớp là một cách hiệu quả để lập trình. Hãy hình dung tệp chương trình này sẽ có độ dài bao nhiêu nếu bao gồm toàn bộ lớp Car. Khi ta thay vào đó di chuyển lớp sang một mô-đun và nhập mô-đun, ta vẫn nhận được tất cả chức năng như vậy, nhưng giữ cho chương trình chính gọn gàng và dễ dàng đọc. Ta nên lưu trữ hầu hết logic trong các tệp riêng biệt; một khi các lớp làm việc đúng theo chức năng, ta có thể để chúng yên và tập trung vào logic cấp cao hơn của chương trình chính.

8.4.2. Lưu trữ nhiều lớp trong một module

Ta có thể lưu trữ bao nhiêu lớp tùy thích trong một mô-đun duy nhất, mặc dù mỗi lớp trong một mô-đun nên có liên quan với nhau bằng cách nào đó. Các lớp

Battery và ElectricCar đều giúp đại diện cho ô tô, vì vậy thêm chúng vào mô-đun car.py.

```
class Battery:  
    """A simple attempt to model a battery for an electric car."""  
    def __init__(self, battery_size=75):  
        """Initialize the battery's attributes."""  
        self.battery_size = battery_size  
    def describe_battery(self):  
        """Print a statement describing the battery size."""  
        print(f"This car has a {self.battery_size}-kWh battery.")  
    def get_range(self):  
        """Print a statement about the range this battery provides."""  
        if self.battery_size == 75:  
            range = 260  
        elif self.battery_size == 100:  
            range = 315  
        print(f"This car can go about {range} miles on a full charge.")  
class ElectricCar(Car):  
    """Models aspects of a car, specific to electric vehicles."""  
    def __init__(self, make, model, year):  
        """  
        Initialize attributes of the parent class.  
        Then initialize attributes specific to an electric car.  
        """  
        super().__init__(make, model, year)  
        self.battery = Battery()
```

Bây giờ chúng ta có thể tạo một chương trình mới có tên là my_electric_car.py, nhập lớp ElectricCar và tạo ra một chiếc ô tô điện:

```
from car import ElectricCar  
my_tesla = ElectricCar('tesla', 'model s', 2019)  
print(my_tesla.get_descriptive_name())  
my_tesla.battery.describe_battery()  
my_tesla.battery.get_range()
```

Kết quả đầu ra giống ở các ví dụ trên, mặc dù hầu hết các logic của chương trình được giấu trong các module.

```
2019 Tesla Model S  
This car has a 75-kWh battery.  
This car can go about 260 miles on a full charge.
```

8.4.3. Nhập vào nhiều lớp trong một module

Ta có thể nhập bao nhiêu lớp tùy thích vào một chương trình. Nếu chúng ta muốn làm một chiếc ô tô thông thường và một chiếc ô tô điện trong cùng một tệp, chúng ta nhập cả hai lớp, Car và ElectricCar:

```
from car import Car, ElectricCar  
my_beetle = Car('volkswagen', 'beetle', 2019)  
print(my_beetle.get_descriptive_name())  
my_tesla = ElectricCar('tesla', 'roadster', 2019)  
print(my_tesla.get_descriptive_name())
```

Ta có thể nhập hai hay nhiều lớp từ một module bằng cách phân tách các lớp với dấu phẩy. Khi ta đã nhập các lớp cần thiết, ta có thể tự do để tạo nhiều thể hiện của mỗi lớp theo mong muốn.

Trong ví dụ trên, chúng ta tạo ra một chiếc xe beetle sử dụng hàm khởi tạo của lớp Car và tạo ra một chiếc xe tesla sử dụng hàm khởi tạo của lớp ElectricCar.

8.4.4. Nhập vào toàn bộ module

Ta cũng có thể nhập toàn bộ mô-đun và sau đó truy cập các lớp ta cần sử dụng ký hiệu dấu chấm. Cách tiếp cận này đơn giản và tạo ra mã dễ đọc. Bởi vì mọi lệnh gọi tạo ra một thể hiện của một lớp bao gồm tên mô-đun, ta sẽ không sợ bị trùng với bất kỳ tên nào được sử dụng trong thời điểm hiện tại.

```
import car  
my_beetle = car.Car('volkswagen', 'beetle', 2019)  
print(my_beetle.get_descriptive_name())  
my_tesla = car.ElectricCar('tesla', 'roadster', 2019)  
print(my_tesla.get_descriptive_name())
```

Đoạn code ở trên, đầu tiên chúng ta nhập vào module car. Sau đó, để truy cập vào các lớp, ta sử dụng qua cú pháp: module_name.ClassName. Tiếp đó, chúng ta tạo xe Volkswagen Beetle bằng cách sử dụng lớp Car và Tesla Roadster bằng cách sử dụng lớp ElectricCar.

8.4.5. Nhập vào toàn bộ các lớp trong một module

Ta có thể nhập mọi lớp từ một module bằng cú pháp sau:

```
from module_name import *
```

Phương pháp này không được khuyến khích vì hai lý do. Đầu tiên, sẽ hữu ích khi có thể đọc các câu lệnh import ở đầu trang và hiểu rõ ràng về chương trình sử

dụng những lớp nào. Với cách tiếp cận này, không rõ lớp nào ta đang sử dụng từ mô-đun. Cách tiếp cận này cũng có thể dẫn đến nhầm lẫn với tên trong tệp. Nếu vô tình nhập một lớp có cùng tên trong chương trình, ta có thể tạo ra các lỗi khó để chẩn đoán. Chúng ta tìm hiểu nội dung này ở đây vì mặc dù nó không được khuyến nghị bởi ta có thể gặp trong mã của người khác vào một lúc nào đó.

Nếu cần nhập nhiều lớp từ một mô-đun, tốt hơn hết là ta nên nhập toàn bộ mô-đun và sử dụng cú pháp module `_name`.`ClassName`. Bạn sẽ không thấy tất cả các lớp được sử dụng ở đầu trang, nhưng sẽ thấy rõ nơi mô-đun được sử dụng trong chương trình. Ta cũng sẽ tránh được khả năng lỗi có thể phát sinh khi ta nhập mọi lớp trong một mô-đun.

8.4.6. Nhập vào một module từ một module

Đôi khi ta muốn chia các lớp của mình qua một số mô-đun để giữ cho bất kỳ cái nào không phát triển quá lớn và tránh lưu trữ các lớp không liên quan trong cùng một mô-đun. Khi lưu trữ các lớp của mình trong một số mô-đun, ta có thể thấy rằng một lớp trong một mô-đun phụ thuộc vào một lớp trong một mô-đun khác. Khi điều này xảy ra, ta có thể nhập lớp được yêu cầu vào module đầu tiên.

Ví dụ: hãy lưu trữ loại Xe hơi trong một mô-đun và các lớp Battery và ElectricCar trong một mô-đun riêng biệt. Chúng ta sẽ tạo một mô-đun mới có tên là `Electric_car.py` — thay thế tệp `điện_car.py` chúng ta đã tạo trước đó — và sao chép chỉ các lớp Battery và ElectricCar vào lớp này:

```
from car import Car
class Battery:
    --snip--
class ElectricCar(Car):
    --snip--
```

Lớp ElectricCar cần quyền truy cập vào cha của nó là lớp Car, vì vậy chúng ta nhập lớp Car trực tiếp vào mô-đun tại dòng đầu tiên. Nếu chúng ta quên dòng này, Python sẽ sinh ra một lỗi khi chúng ta cố gắng nhập mô-đun `electric_car`. Chúng ta cũng cần cập nhật mô-đun Car để nó chỉ chứa lớp Car:

Giờ đây, chúng ta có thể import từng mô-đun riêng biệt và tạo bất kỳ thứ gì loại xe chúng ta cần:

```
from car import Car
```

```
from electric_car import ElectricCar
my_beetle = Car('volkswagen', 'beetle', 2019)
print(my_beetle.get_descriptive_name())
my_tesla = ElectricCar('tesla', 'roadster', 2019)
print(my_tesla.get_descriptive_name())
```

Chúng ta nhập Car từ mô-đun của nó và ElectricCar từ mô-đun của nó. Sau đó, chúng ta tạo ra một chiếc ô tô thông thường và một chiếc ô tô điện. Cả hai loại ô tô đều được tạo chính xác:

```
2019 Volkswagen Beetle
2019 Tesla Roadster
```

8.4.7. Sử dụng bí danh

Như đã thấy trong Chương 7, bí danh có thể khá hữu ích khi sử dụng mô-đun để sắp xếp mã dự án. Ta có thể sử dụng bí danh khi nhập lớp.

Ví dụ, hãy xem xét một chương trình mà ta muốn tạo một nhóm của ô tô điện. Việc gõ (và đọc) ElectricCar có thể trở nên tẻ nhạt và lặp lại. Ta có thể đặt bí danh cho ElectricCar trong câu lệnh nhập:

```
from electric_car import ElectricCar as EC
```

Giờ đây, ta có thể sử dụng bí danh này bất cứ khi nào muốn tạo ra một chiếc ô tô điện:

```
my_tesla = EC('tesla', 'roadster', 2019)
```

8.5. Thư viện chuẩn của python

Thư viện chuẩn Python là một tập hợp các mô-đun đi kèm với mọi cài đặt Python. Bây giờ ta đã có hiểu biết cơ bản về cách các hàm và lớp hoạt động, ta có thể bắt đầu sử dụng các mô-đun mà các lập trình viên khác đã viết. Ta có thể sử dụng bất kỳ hàm hoặc lớp nào trong thư viện chuẩn bằng cách bao gồm một câu lệnh import đơn giản ở đầu tệp mã nguồn. Nội dung này giới thiệu một số module thư viện được sử dụng phổ biến như: module random, module numpy, module statistics.

8.5.1. Random

Module random là module được dựng sẵn trong Python, chúng ta có thể sử dụng thư viện này để sinh ra các số ngẫu nhiên. Module gồm các phương thức phổ biến sau:

random.seed(n): sử dụng để cố định việc chọn ngẫu nhiên cho các lần chạy tiếp theo (cùng n thì các lần random khác nhau đều ra các số giống nhau).

random.randint(x,y): trả về một số kiểu integer trong một khoảng cố định bởi tham số chẵn dưới x và chẵn trên y (bao gồm cả 2 số đó).

```
import random  
a= random.randint(3,9)  
print(a)
```

Kết quả trả về một số ngẫu nhiên trong khoảng 3 đến 9: 8

random.randrange(x,y): trả về một số ngẫu nhiên trong khoảng x (bao gồm) và y (không bao gồm).

random.choice(list): trả về một phần tử ngẫu nhiên trong danh sách cho trước.

```
mylist = ["apple", "banana", "cherry"]  
  
print(random.choice(mylist))
```

Kết quả trả về một phần tử ngẫu nhiên: banana

random.choices(list,params..): trả về một danh sách có số lượng phần tử được định trước (mặc định bằng 1) từ danh sách ban đầu. Có thể lựa chọn tần suất chọn của từng phần tử trong danh sách ban đầu qua tham số weight.

random.sample(): trả về một số các phần tử ngẫu nhiên trong danh sách ban đầu.

ví dụ:

```
import random  
mylist = ["apple", "banana", "cherry"]  
print(random.sample(mylist, k=2))
```

Kết quả trả về danh sách 2 phần tử ngẫu nhiên từ mylist

```
['cherry', 'banana']
```

Ví dụ về dãy số:

```
list_number=random.sample(range(10,101),5)  
print(list_number)
```

Kết quả: dãy số 5 phần tử trong khoảng (10,100):

```
[53, 62, 57, 65, 94]
```

random.shuffle(mylist): trả về danh sách có số lượng phần tử giống danh sách ban đầu nhưng thứ tự được trộn.

random.uniform(x,y): trả về một số thực ngẫu nhiên trong khoảng x,y.

Ví dụ:

```
fnum=random.uniform(10,20)  
print(fnum)
```

Kết quả:

14.431767170764473

random.random(): trả về một số thực giữa 0 và 1

8.5.2. Numpy

Numpy là bộ thư viện Python được dùng để làm việc với các mảng (arrays).

Numpy có các hàm có thể làm việc trong các lĩnh vực như đại số tuyến tính, biến đổi chuỗi hay ma trận. Bộ thư viện được tạo ra năm 2005 bởi Travis Oliphant, lập trình viên có thể sử dụng nó miễn phí.

Các chuỗi tuần tự trong python có thể xử lý được bởi list, tuy nhiên khi sử dụng Numpy, tốc độ xử lý tăng gấp nhiều lần so với list truyền thống. Danh sách các đối tượng trong Numpy gọi là ndarray, cung cấp các hàm hỗ trợ và chúng ta có thể làm việc với danh sách các đối tượng này một cách dễ dàng.

Tham khảo thư viện Numpy:

<https://numpy.org/>

<https://www.w3schools.com/python/numpy/default.asp>

8.5.3. Pandas

Pandas là một công cụ phân tích và thao tác dữ liệu nguồn mở nhanh, mạnh, linh hoạt và dễ sử dụng, được xây dựng dựa trên ngôn ngữ lập trình Python.

Thư viện Pandas cung cấp 2 đối tượng chính để thao tác dữ liệu là Series (cho dữ liệu dưới dạng cột) và Dataframe (cho dữ liệu dưới dạng bảng).

Series: là mảng 1 chiều có nhãn có khả năng chứa dữ liệu bất kỳ thuộc các loại như: số nguyên, chuỗi, số thực, và các đối tượng python. Các nhãn chung thường được gọi là các chỉ mục (index). Ví dụ sau tạo ra một Series:

```
import pandas as pd
```

```
x = pd.Series([1, 2, 2, np.nan], index=['p', 'q', 'r', 's'])
```

Với Series có thể tìm kiếm, sắp xếp, thông qua mối quan hệ giữa giá trị và chỉ mục.

Dataframe: là cấu trúc dữ liệu dạng bảng có thể thay đổi được kích thước bao gồm ba thành phần chính là: Hàng, Cột và Dữ liệu.

Ví dụ khởi tạo dataframe:

```
import pandas as pd  
df = pd.DataFrame({ 'X':[78,85,96,80,86], 'Y':[84,94,89,83,86], 'Z':[86,97,96,72,83] });  
display(df)
```

Kết quả:

	X	Y	Z
0	78	84	86
1	85	94	97
2	96	89	96
3	80	83	72
4	86	86	83

Các thao tác với dataframe bao gồm:

- Xem và kiểm tra dữ liệu
- Chọn dữ liệu (theo hàng, theo cột, theo vị trí, theo nhãn)
- Làm sạch dữ liệu (kiểm tra trống trống, xóa trống trống, làm đầy ô trống)
- Lọc, sắp xếp, gom nhóm
- Gộp, nối bảng
- Thống kê mô tả
- Nhập dữ liệu từ tệp (import data) và xuất dữ liệu ra tệp (export data).

8.5.4. Statistics

Module này cung cấp các chức năng để tính toán thống kê toán học của dữ liệu số (Giá trị thực).

Sử dụng statistic để tính giá trị trung bình (statistics.mean(data)):

```
import statistics  
import random
```

```
random.seed(10)
data=random.sample(range(10,101),10)
print(data)
statistics.mean(data)
```

Kết quả:

```
[83, 14, 64, 71, 11, 36, 69, 72, 45, 93]
```

```
55.8
```

Sử dụng statistic để tính median, mode, quantiles

```
median(), mode(), quantiles()
```

Ngoài ra, có thể sử dụng thư viện này để thực hiện:

- Tính toán độ trai của dữ liệu: stdev(), variance()
- Thông kê mối qua hệ của 2 đầu vào: covariance(), correlation(), linear_regression()

Tham khảo:

<https://docs.python.org/3/library/statistics.html>

Bài tập chương 8

8-1. Restaurant: Tạo một lớp là Restaurant. Phương thức `__init__()` cho lớp này cần chứa hai thuộc tính là `restaurant_name` và `cuisine_type`. Tạo một phương thức gọi là `description_restaurant()` để in hai phần thông tin này và một phương thức có tên `open_restaurant()` in một thông báo cho biết nhà hàng đang mở cửa.

Tạo ra một thể hiện là restaurant từ lớp Restaurant, in hai thuộc tính, sau đó gọi cả hai phương thức để in ra thông tin.

8-2. Three Restaurants: Dữ kiện như bài 8-1, tạo ba thể hiện khác nhau của lớp Restaurant và sử dụng phương thức `description_restaurant()` cho mỗi thể hiện.

8-3. Users: Tạo ra một lớp có tên là User. Tạo hai thuộc tính là `first_name` và `last_name` và tạo ra một vài thuộc tính khác trong hồ sơ người dùng. Tạo ra phương thức `describe_user()` in ra thông tin chung của người dùng và một phương thức khác là `greet_user()` in ra lời chào tùy biến cho mỗi người dùng (theo tên đầy đủ). Tạo ra một vài thể hiện khác nhau của người dùng và gọi cả hai phương thức cho mỗi người dùng.

8-4. Number Served: Bắt đầu như bài tập 8-1. Tạo thêm một thuộc tính tên là `number_served` có giá trị mặc định là 0. Tạo ra một thể hiện gọi là `restaurant` từ lớp này. In số lượng khách hàng mà `restaurant` đã phục vụ. Thay đổi giá trị này và in lại ra kết quả.

Thêm một phương thức có tên `set_number_served()` cho phép đặt số lượng khách hàng đã được phục vụ. Gọi phương thức này bằng một số mới và in lại giá trị.

Thêm một phương thức có tên là `increment_number_served()` cho phép tăng số lượng khách hàng đã được phục vụ. Gọi phương thức này với bất kỳ số nào thích mà có thể thể hiện số lượng khách hàng đã được phục vụ trong một ngày làm việc.

8-5. Login Attempts: Thêm thuộc tính `login_attempts` cho lớp `User` trong bài tập 8-3. Viết phương thức `increment_login_attempts()` để tăng giá trị số lần login thêm 1. Viết phương thức khác gọi là `reset_login_attempts()` để thiết lập giá trị `login_attempts` về 0.

Tạo ra một thể hiện của lớp `User` và gọi phương thức `increment_login_attempts()` một vài lần. In giá trị của `login_attempts` để đảm bảo phương thức hoạt động đúng và gọi phương thức `reset_login_attempts()` và in ra giá trị của `login_attempts` để đảm bảo nó đã được thiết lập về 0.

8-6. Ice Cream Stand: Quầy kem là một loại hình cửa hàng đặc biệt. Viết một lớp gọi là `IceCreamStand` kế thừa từ lớp `Restaurant` trong bài tập 8-1 hoặc bài tập 8-4 (tùy chọn). Thêm một thuộc tính gọi là `flavors` lưu các danh sách các loại kem. Viết phương thức hiển thị các loại kem này. Tạo ra một thể hiện của lớp `IceCreamStand` và gọi phương thức vừa viết để hiển thị các loại kem.

8-7. Admin: một quản trị viên là một loại người dùng đặc biệt. Viết một lớp gọi là `Admin` kế thừa từ lớp `User` trong bài tập 8-3 hoặc 8-5. Thêm một thuộc tính tên là `privileges` lưu trữ danh sách các chuỗi như "can add post", "can delete post", "can ban user"... Viết phương thức `show_privileges()` in ra danh sách các quyền của người quản trị. Tạo ra một thể hiện của lớp `Admin` và gọi phương thức `show_privileges()` để thấy kết quả.

8-8. Privileges: Tạo ra một lớp `Privileges` độc lập. Lớp này có thuộc tính `privileges` lưu trữ các chuỗi trong bài tập 8-7. Di chuyển phương thức

show_privileges() từ lớp Admin sang lớp này. Tạo ra một thẻ hiện của lớp Privileges như một thuộc tính trong lớp Admin. Tạo một thẻ hiện của lớp Admin và sử dụng phương thức đã viết để in ra các quyền của thẻ hiện đó.

8-9. Battery Upgrade: Sử dụng phiên bản của cùng của tệp electric_car.py trong chương này. Thêm một phương thức vào lớp Battery có tên là upgrade_battery(). Phương thức này sẽ kiểm tra kích thước pin và đặt dung lượng thành 100 nếu chưa bằng 100. Tạo ra một xe điện với kích thước pin mặc định, gọi phương thức get_range() một lần và sau đó gọi phương thức get_range() lần thứ hai sau khi đã nâng cấp kích cỡ pin. Cần thấy sự gia tăng của phạm vi của ô tô đi được.

8-10. Imported Restaurant: Sử dụng phiên bản cuối cùng của lớp Restaurant, lưu nó vào trong module. Tạo ra một tệp riêng để nhập lớp Restaurant. Tạo ra một thẻ hiện của lớp Restaurant và gọi các phương thức của lớp restaurant để thấy là câu lệnh import hoạt động đúng.

8-11. Imported Admin: Bắt đầu như bài 8-8. Lưu các lớp User, Admin, Privileges trong một module. Tạo một tệp riêng và tạo ra thẻ hiện của lớp Admin, sau đó gọi phương thức show_privileges() để thấy mọi thứ hoạt động đúng.

8-12. Multiple Modules: Lưu trữ lớp User trong một module và lưu trữ lớp Privileges và Admin trong một module khác. Trong một tệp riêng khác, tạo ra một thẻ hiện của Admin và gọi show_privileges() để thấy rằng mọi thứ vẫn hoạt động đúng.

8-13. Dice (Xúc xắc): Tạo ra một lớp Dice với thuộc tính là sizes, với giá trị mặc định là 6. Tạo ra một phương thức gọi là roll_die() in ra một số ngẫu nhiên giữa 1 và sizes của xúc xắc. Tạo ra một con xúc xắc kích cỡ 6 và quay nó 10 lần.

Tạo ra con xúc xắc kích cỡ 10, 20 và quay mỗi con 10 lần.

8-14. Lottery: Tạo ra một danh sách 10 số và 5 chữ cái bằng list hoặc tuple. Chọn ngẫu nhiên bốn số hoặc chữ cái từ danh sách và in thông báo nói rằng bất kỳ vé nào khớp với bốn số hoặc chữ cái này sẽ thắng phần thưởng.

8-15. Lottery Analysis: Bạn có thể sử dụng một vòng lặp để xem mức độ khó có thể trúng loại xổ số mà ta vừa lập mô hình. Tạo một danh sách hoặc bộ có tên là

my_ticket. Viết một vòng lặp tiếp tục lấy các số cho đến khi vé của ta thắng. In một tin nhắn báo cáo số lần vòng lặp phải chạy để cung cấp cho ta một vé trúng thưởng.

8-16. Python Module of the Week: Một tài nguyên tuyệt vời để khám phá thư viện tiêu chuẩn Python là một trang web được gọi là Python Module of the Week. Đến <https://pymotw.com/> và xem mục lục. Tìm một mô-đun mà ta thấy thú vị và đọc về nó, có thể bắt đầu với mô-đun random.

Kết chương

Trong chương này, ta đã học cách viết các lớp theo mong muốn. Ta đã học cách lưu trữ thông tin trong một lớp bằng cách sử dụng các thuộc tính và cách viết các phương thức cung cấp cho các lớp và hành vi mà chúng cần. Ta đã học cách viết các phương thức `__init__()` để tạo các thể hiện từ các lớp với chính xác các thuộc tính mong muốn.

Chúng ta đã thấy cách sửa đổi các thuộc tính của một thể hiện trực tiếp và thông qua các phương thức. Ta đã học được rằng kế thừa có thể đơn giản hóa việc tạo các lớp có liên quan với nhau và đã học cách sử dụng các thể hiện của một lớp làm thuộc tính trong một lớp khác để giữ mỗi lớp đơn giản.

Ta đã thấy cách lưu trữ các lớp trong mô-đun và nhập các lớp nếu cần vào tệp nơi chúng sẽ được sử dụng có thể giúp các dự án được tổ chức quy củ. Ta bắt đầu tìm hiểu về thư viện chuẩn Python và đã thấy một ví dụ dựa trên mô-đun random.

Trong Chương 9, chúng sẽ học cách làm việc với tệp để có thể lưu công việc đã thực hiện trong một chương trình và công việc đã cho phép người dùng thực hiện. Ta cũng sẽ tìm hiểu về ngoại lệ (exceptions) - một lớp Python đặc biệt được thiết kế để giúp phản hồi các lỗi khi chúng phát sinh.

CHƯƠNG 9. TỆP VÀ NGOẠI LỆ

Bây giờ ta đã thành thạo các kỹ năng cơ bản từ kiến thức học từ các chương trước. Ta cần viết các chương trình có tổ chức, dễ sử dụng, cũng đã đến lúc suy nghĩ về làm cho các chương trình phù hợp hơn và có thể sử dụng được. Trong chương này, chúng ta sẽ học cách làm việc với các tệp để các chương trình có thể nhanh chóng phân tích nhiều dữ liệu.

Ta sẽ học cách xử lý lỗi để chương trình của ta không gặp sự cố khi gặp những tình huống bất ngờ. Ta sẽ tìm hiểu về các trường hợp ngoại lệ, đó là các đối tượng đặc biệt mà Python tạo ra để quản lý các lỗi phát sinh trong khi một chương trình đang chạy.

Ta cũng sẽ tìm hiểu về mô-đun json, mô-đun này cho phép lưu dữ liệu người dùng để dữ liệu không bị mất khi chương trình ngừng chạy.

Học cách làm việc với tệp và lưu dữ liệu sẽ làm cho các chương trình dễ dàng hơn cho mọi người sử dụng. Người dùng sẽ có thể chọn dữ liệu để nhập và khi nào nhập dữ liệu. Mọi người có thể chạy chương trình, thực hiện một số công việc và sau đó đóng chương trình và tiếp tục chương trình đã dừng lại trước đó.

Học cách xử lý ngoại lệ sẽ giúp ta đối phó với các tình huống trong đó tệp không tồn tại và giải quyết các sự cố khác có thể khiến chương trình gặp sự cố.

Điều này sẽ làm cho các chương trình mạnh mẽ hơn khi chúng gặp phải dữ liệu xấu, cho dù dữ đến từ những sai lầm không cố ý hoặc từ những nỗ lực ác ý để phá vỡ các chương trình. Với các kỹ năng học trong chương này, ta sẽ làm cho các chương trình trở nên dễ áp dụng hơn, có thể sử dụng được và ổn định hơn.

9.1. Đọc từ file

Một lượng dữ liệu đáng kinh ngạc có sẵn trong các tệp văn bản. Tệp văn bản có thể chứa dữ liệu thời tiết, dữ liệu giao thông, dữ liệu kinh tế xã hội, tác phẩm văn học, v.v. Đọc từ một tệp đặc biệt hữu ích trong các ứng dụng phân tích dữ liệu, nhưng nó cũng có thể áp dụng cho bất kỳ trường hợp nào ta muốn phân tích hoặc sửa đổi thông tin được lưu trữ trong tệp. Ví dụ: chúng ta có thể viết một chương trình đọc nội dung của một tệp văn bản và viết lại tệp với định dạng cho phép trình duyệt hiển thị nó.

Khi ta muốn làm việc với thông tin trong tệp văn bản, bước đầu tiên là đọc tệp đó vào bộ nhớ. Ta có thể đọc toàn bộ nội dung của tệp hoặc ta có thể làm việc trên tệp từng dòng một.

9.1.1. Đọc toàn bộ tệp

Để bắt đầu, chúng ta cần một tệp có một vài dòng văn bản trong đó. Hãy bắt đầu với một tệp chứa số pi đến 30 chữ số thập phân, với 10 chữ số thập phân trên mỗi dòng:

```
3.1415926535  
8979323846  
2643383279
```

Để tự thử các ví dụ sau, ta có thể nhập những dòng này vào trình chỉnh sửa và lưu tệp dưới dạng `pi_digits.txt` hoặc ta có thể tải xuống tệp từ tài nguyên của sách thông qua <https://nostarch.com/pythoncrashcourse2e/>. Lưu tệp trong cùng một thư mục nơi ta sẽ lưu trữ các chương trình của chương này.

Đây là chương trình mở tệp này, đọc và in nội dung của tệp ra màn hình:

```
with open('pi_digits.txt') as file_object:  
    contents = file_object.read()  
print(contents)
```

Dòng đầu tiên của chương trình này có rất nhiều thứ đang diễn ra. Hãy bắt đầu bằng cách xem xét hàm `open()`. Để thực hiện bất kỳ công việc nào với tệp, thậm chí chỉ in nội dung của nó, trước tiên ta cần mở tệp để truy cập. Hàm `open()` cần một đối số: tên của tệp ta muốn mở. Python tìm kiếm tệp này trong thư mục lưu trữ chương trình hiện đang được thực thi. Trong ví dụ này, `file_reader.py` hiện đang chạy, vì vậy Python tìm kiếm `pi_digits.txt` trong thư mục lưu trữ `file_reader.py`. Hàm `open()` trả về một đối tượng đại diện cho tệp. Ở đây, `open('pi_digits.txt')` trả về một đối tượng đại diện cho `pi_digits.txt`. Python chỉ định đối tượng này cho `file_object`, đối tượng mà chúng ta sẽ làm việc sau này trong chương trình.

Từ khóa `with` sẽ đóng tệp khi không cần truy cập vào nó nữa. Lưu ý cách chúng ta gọi `open()` trong chương trình này nhưng không gọi `close()`. Ta có thể mở và đóng tệp bằng cách gọi `open()` và `close()`, nhưng nếu một lỗi trong chương trình ngăn không cho phương thức `close()` được thực thi, thì tệp có thể không bao giờ đóng. Điều này có vẻ tầm thường, nhưng các tệp được đóng không đúng cách có thể khiến dữ liệu bị

mất hoặc bị hỏng. Và nếu gọi close() quá sớm trong chương trình, ta sẽ thấy mình đang cố gắng làm việc với một tệp đã đóng (tệp mà không thể truy cập), điều này dẫn đến nhiều lỗi hơn. Không phải lúc nào cũng dễ dàng biết chính xác khi nào nên đóng một tệp, nhưng với cấu trúc được hiển thị ở đây, Python sẽ tìm ra điều đó giúp ta. Tất cả những gì cần phải làm là mở tệp và làm việc với nó như mong muốn, tin tưởng rằng Python sẽ tự động đóng nó khi khỏi *with* kết thúc quá trình thực thi.

Khi chúng ta có một đối tượng tệp đại diện cho pi_digits.txt, chúng ta sử dụng phương thức read() trong dòng thứ hai của chương trình để đọc toàn bộ nội dung của tệp và lưu trữ nó dưới dạng một chuỗi dài trong nội dung. Khi chúng ta in giá trị của nội dung, ta nhận lại toàn bộ tệp văn bản:

```
3.1415926535  
8979323846  
2643383279
```

Sự khác biệt duy nhất giữa đầu ra này và tệp gốc là thêm dòng trống ở cuối đầu ra. Dòng trống xuất hiện vì read() trả về một chuỗi trống khi nó đến cuối tệp; chuỗi trống này hiển thị dưới dạng một dòng trống. Nếu muốn loại bỏ dòng trống thừa, ta có thể sử dụng rstrip() trong lệnh gọi print():

```
with open('pi_digits.txt') as file_object:  
    contents = file_object.read()  
print(contents.rstrip())
```

Nhớ lại rằng phương thức rstrip() của Python loại bỏ hoặc tách bất kỳ ký tự khoảng trắng nào từ phía bên phải của một chuỗi. Böyle giờ đầu ra khớp chính xác với nội dung của tệp gốc:

```
3.1415926535  
8979323846  
2643383279
```

9.1.2. Đường dẫn tệp

Khi ta chuyển một tên tệp đơn giản như pi_digits.txt vào hàm open(), Python sẽ tìm trong thư mục nơi tệp hiện đang được thực thi (tức là tệp chương trình .py) được lưu trữ.

Đôi khi, tùy thuộc vào cách chúng ta tổ chức công việc, tệp mong muốn mở sẽ không nằm trong cùng thư mục với tệp chương trình đang chạy. Ví dụ: ta có thể lưu trữ các tệp chương trình của mình trong một thư mục có tên là python_work; bên

trong `python_work`, ta có thể có một thư mục khác có tên là `text_files` để phân biệt tệp chương trình với tệp văn bản mà chúng đang chạy. Mặc dù `text_files` nằm trong `python_work`, chỉ cần chuyển `open()` tên của tệp trong `text_files` sẽ không hoạt động, vì Python sẽ chỉ tìm kiếm trong `python_work` và dừng lại ở đó; nó sẽ không tiếp tục và tìm trong `text_files`. Để Python mở các tệp từ một thư mục khác với thư mục nơi tệp chương trình được lưu trữ, ta cần cung cấp một đường dẫn tệp, đường dẫn này cho Python biết để tìm kiếm ở một vị trí cụ thể trên hệ thống.

Vì `text_files` nằm trong `python_work`, ta có thể sử dụng đường dẫn tệp tương đối để mở tệp từ `text_files`. Một đường dẫn tệp tương đối yêu cầu Python tìm kiếm một vị trí nhất định liên quan đến thư mục nơi tệp chương trình hiện đang chạy được lưu trữ. Ví dụ: ta sẽ viết:

```
with open('text_files/filename.txt') as file_object:
```

Dòng này yêu cầu Python tìm kiếm tệp `.txt` mong muốn trong thư mục `text_files` và giả định rằng thư mục `text_files` nằm bên trong `python_work`.

Ghi chú: Hệ thống Windows sử dụng dấu gạch chéo ngược (`\`) thay vì dấu gạch chéo (`/`) khi hiển thị đường dẫn tệp, nhưng ta vẫn có thể sử dụng dấu gạch chéo về phía trước trong mã của mình.

Chúng ta cũng có thể cho Python biết chính xác vị trí của tệp trên máy tính bất kể chương trình đang được thực thi được lưu trữ ở đâu. Đây được gọi là đường dẫn tệp tuyệt đối. Ta sử dụng một đường dẫn tuyệt đối nếu một đường dẫn tương đối không hoạt động. Ví dụ: nếu ta đã đặt `text_files` trong một số thư mục không phải là `python_work` - giả sử, thư mục có tên `other_files` - thì việc chỉ truyền cho phương thức `open()` tham số đường dẫn '`text_files/filename.txt`' sẽ không hoạt động vì Python sẽ chỉ tìm kiếm vị trí đó bên trong `python_work`. Ta sẽ cần viết ra một đường dẫn đầy đủ để làm rõ nơi ta muốn Python tìm kiếm.

Đường dẫn tuyệt đối thường dài hơn đường dẫn tương đối, vì vậy sẽ hữu ích nếu chỉ định chúng cho một biến và sau đó chuyển biến đó vào `open()`:

```
file_path = '/home/ehmatthes/other_files/text_files/filename.txt'  
with open(file_path) as file_object:
```

Sử dụng đường dẫn tuyệt đối, ta có thể đọc tệp từ bất kỳ vị trí nào trên hệ thống của mình. Hiện tại, việc lưu trữ tệp trong cùng thư mục với tệp chương trình hoặc

trong một thư mục như `text_files` trong thư mục lưu trữ tệp chương trình là dễ dàng nhất.

Ghi chú: Nếu ta cố gắng sử dụng dấu gạch chéo ngược trong một đường dẫn ngắn, ta sẽ gặp lỗi vì dấu gạch chéo ngược là dùng để loại bỏ các ký tự trong chuỗi. Ví dụ: trong đường dẫn "C:\path\to\file.txt", và tổ hợp \t được hiểu là một tab. Nếu cần sử dụng dấu gạch chéo ngược, ta có thể thêm mỗi ký tự chéo ngược trong đường dẫn, như sau: "C:\\path\\to\\file.txt".

9.1.3. Đọc từng dòng

Khi đang đọc một tệp, ta thường muốn kiểm tra từng dòng của tệp. Chúng ta có thể đang tìm kiếm thông tin nhất định trong tệp hoặc có thể muốn sửa đổi văn bản trong tệp theo một cách nào đó. Ví dụ: ta có thể muốn đọc qua một tệp dữ liệu thời tiết và làm việc với bất kỳ dòng nào có từ `nắng` trong phần mô tả thời tiết của ngày hôm đó. Trong một báo cáo tin tức, ta có thể tìm bất kỳ dòng nào có thẻ `<headline>` và viết lại dòng đó với một loại định dạng cụ thể.

Ta có thể sử dụng vòng lặp `for` trên đối tượng tệp để kiểm tra từng dòng từ tệp tại một thời điểm:

```
filename = 'pi_digits.txt'  
with open(filename) as file_object:  
    for line in file_object:  
        print(line)
```

Tại câu lệnh đầu tiên, gán tên của tệp đang đọc cho biến `filename`. Đây là quy ước chung khi làm việc với tệp. Bởi vì biến `filename` không đại diện cho tệp thực - nó chỉ là một chuỗi cho Python biết nơi tìm tệp - ta có thể dễ dàng hoán đổi '`pi_digits.txt`' cho tên của tệp khác mà ta muốn làm việc. Sau khi chúng ta gọi `open()`, một đối tượng đại diện cho tệp và nội dung của nó được gán cho biến `file_object`. Chúng ta lại sử dụng cú pháp `with` để cho phép Python mở và đóng tệp đúng cách. Để kiểm tra nội dung của tệp, chúng ta làm việc qua từng dòng trong tệp bằng cách lặp qua đối tượng tệp.

Khi chúng in từng dòng, ta tìm thấy nhiều dòng trống hơn:

3.1415926535

8979323846

Những dòng trống này xuất hiện vì một ký tự dòng mới ẩn ở cuối mỗi dòng trong tệp văn bản. Hàm print sẽ thêm dòng mới của riêng nó mỗi khi chúng ta gọi nó, vì vậy chúng ta kết thúc bằng hai ký tự dòng mới ở cuối mỗi dòng: một từ tệp và một từ print(). Sử dụng rstrip() trên mỗi dòng trong lệnh gọi print() sẽ loại bỏ các dòng trống thừa sau:

```
filename = 'pi_digits.txt'
with open(filename) as file_object:
    for line in file_object:
        print(line.rstrip())
```

Bây giờ, nội dung in ra khớp với nội dung của tệp

3.1415926535

8979323846

2643383279

9.1.4. Tạo danh sách các dòng từ một tệp

Khi chúng ta sử dụng *with*, đối tượng tệp được trả về bởi open() chỉ khả dụng bên trong khôi *with* có chứa nó. Nếu ta muốn duy trì quyền truy cập vào nội dung của tệp bên ngoài khôi *with*, ta có thể lưu trữ các dòng của tệp trong danh sách bên trong khôi và sau đó làm việc với danh sách đó. Ta có thể xử lý các phần của tệp ngay lập tức và hoàn một số quá trình xử lý cho phần sau của chương trình.

Ví dụ sau lưu trữ các dòng của pi_digits.txt trong danh sách bên trong khôi *with* và sau đó in các dòng bên ngoài khôi *with*:

```
filename = 'pi_digits.txt'
with open(filename) as file_object:
    lines = file_object.readlines()
for line in lines:
    print(line.rstrip())
```

Tại filename = 'pi_digits.txt', phương thức readlines() lấy từng dòng từ tệp và lưu trữ nó trong một danh sách. Danh sách này sau đó được gán cho biến *lines*, chúng ta có thể tiếp tục làm việc với danh sách đó sau khi khôi *with* kết thúc. Chúng ta sử dụng một vòng lặp for đơn giản để in từng dòng từ *lines*. Bởi vì mỗi phần tử trong danh sách *lines* tương ứng với mỗi dòng trong tệp, đầu ra khớp chính xác với nội dung của tệp.

9.1.5. Làm việc với nội dung của tệp

Sau khi chúng ta đã đọc một tệp vào bộ nhớ, ta có thể làm bất cứ điều gì mong muốn với dữ liệu đó, ta hãy cùng khám phá ngắn gọn các chữ số của số pi. Đầu tiên, chúng ta sẽ cố gắng tạo một chuỗi đơn chứa tất cả các chữ số trong tệp mà không có khoảng trắng trong đó:

```
filename = 'pi_digits.txt'
with open(filename) as file_object:
    lines = file_object.readlines()

pi_string = ''
for line in lines:
    pi_string += line.rstrip()

print(pi_string)
print(len(pi_string))
```

Chúng ta bắt đầu bằng cách mở tệp và lưu trữ từng dòng chữ số trong một danh sách, giống như đã làm trong ví dụ trước. Sau đó, tạo một biến, pi_string, để chứa các chữ số của số pi. Sau đó, ta tạo một vòng lặp thêm từng dòng chữ số vào chuỗi pi_string và xóa ký tự dòng mới khỏi mỗi dòng. Tại print(), chúng ta in chuỗi này và cũng hiển thị độ dài của chuỗi:

```
3.1415926535 8979323846 2643383279
36
```

Biến pi_string chứa khoảng trắng ở bên trái của các chữ số trong mỗi dòng, nhưng chúng ta có thể loại bỏ điều đó bằng cách sử dụng strip() thay vì rstrip():

```
for line in lines:
    pi_string += line.strip()

print(pi_string)
print(len(pi_string))
```

Bây giờ chúng ta có một chuỗi chứa số pi đến 30 chữ số thập phân. Chuỗi dài 32 ký tự vì nó cũng bao gồm 3 ở đầu và dấu thập phân:

```
3.141592653589793238462643383279
32
```

Ghi chú: Khi Python đọc từ một chuỗi văn bản, nó sẽ chuyển tất cả văn bản trong tệp dưới dạng một chuỗi. nếu ta đọc một số và muốn làm việc với giá trị đó

trong ngữ cảnh số, ta sẽ phải chuyển đổi nó thành một số nguyên bằng cách sử dụng hàm int() hoặc chuyển nó thành một số thực bằng cách sử dụng hàm float().

9.1.6. Tệp lớn: Một triệu chữ số

Đến thời điểm hiện tại, chúng ta tập trung vào việc phân tích một tệp văn bản chỉ chứa ba dòng, nhưng code trong các ví dụ này vẫn hoạt động tốt đối với các tệp lớn hơn nhiều. Nếu chúng ta bắt đầu với một tệp văn bản chứa pi đến 1.000.000 chữ số thập phân thay vì chỉ 30, chúng ta có thể tạo một chuỗi đơn chứa tất cả các chữ số này. Chúng ta không cần phải thay đổi chương trình của mình ngoại trừ việc chuyển nó sang một tệp khác. Chúng ta cũng sẽ chỉ in 50 chữ số thập phân đầu tiên để chúng ta không phải xem một triệu chữ số cuộn trong cửa sổ in kết quả:

```
filename = 'pi_million_digits.txt'
with open(filename) as file_object:
    lines = file_object.readlines()
pi_string = ''
for line in lines:
    pi_string += line.strip()
print(f"{pi_string[:52]}...")
print(len(pi_string))
```

Kết quả cho thấy rằng chúng ta thực sự có một chuỗi chứa số pi đến 1.000.000 chữ số thập phân:

```
3.14159265358979323846264338327950288419716939937510...
1000002
```

Python không có giới hạn có hữu về lượng dữ liệu ta có thể làm việc; ta có thể làm việc với nhiều dữ liệu mà bộ nhớ của hệ thống có thể xử lý.

9.1.7. Tìm ngày sinh trong số PI

Chúng ta luôn tò mò muốn biết liệu ngày sinh của mình có xuất hiện ở bất kỳ đâu trong các chữ số của số pi hay không. Hãy sử dụng chương trình vừa viết để tìm hiểu xem sinh nhật của ai đó có xuất hiện ở bất kỳ đâu trong một triệu chữ số đầu tiên của số pi hay không. Chúng ta có thể làm điều này bằng cách biểu thị mỗi ngày sinh dưới dạng một chuỗi chữ số và xem liệu chuỗi đó có xuất hiện ở bất kỳ đâu trong pi_string hay không:

```
for line in lines:
    pi_string += line.strip()
```

```

birthday = input("Enter your birthday, in the form mmddyy: ")
if birthday in pi_string:
    print("Your birthday appears in the first million digits of pi!")
else:
    print("Your birthday does not appear in the first million digits of
pi.")

```

Tại `birthday = input("Enter your birthday, in the form mmddyy: ")`, chúng ta nhắc ngày sinh của người dùng và sau đó, chúng ta kiểm tra xem chuỗi đó có trong `pi_string` không:

```

Enter your birthdate, in the form mmddyy: 120372
Your birthday appears in the first million digits of pi!

```

Ngày sinh này xuất hiện bằng các chữ số của số pi! Khi đã đọc từ một tệp, ta có thể phân tích nội dung của nó theo bất kỳ cách nào chúng ta có thể tưởng tượng được.

9.2. Ghi vào file

Một trong những cách đơn giản nhất để tiết kiệm dữ liệu là ghi nó vào một tệp. Khi ta ghi văn bản vào một tệp, đầu ra sẽ vẫn có sẵn sau khi ta đóng cửa sổ chứa đầu ra của chương trình (output). Ta có thể kiểm tra đầu ra sau một chương trình đã hoàn thành và ta cũng có thể chia sẻ kết quả đầu ra với những người khác. Ta cũng có thể viết các chương trình đọc lại văn bản vào bộ nhớ và làm việc lại với nó sau này.

9.2.1. Ghi vào tệp rỗng

Để ghi văn bản vào tệp, ta cần gọi `open()` với đối số thứ hai cho Python biết rằng ta muốn ghi vào tệp. Để xem cách này hoạt động như thế nào, ta viết một thông điệp đơn giản và lưu trữ trong tệp thay vì in ra màn hình:

```

filename = 'programming.txt'
with open(filename, 'w') as file_object:
    file_object.write("I love programming.")

```

Lời gọi `open()` trong ví dụ này có hai đối số. Đối số đầu tiên vẫn là tên của tệp chúng ta muốn mở. Đối số thứ hai, '`w`', cho Python biết rằng chúng ta muốn mở tệp ở chế độ ghi. Ta có thể mở tệp ở chế độ đọc ('`r`'), chế độ ghi ('`w`'), chế độ nối thêm ('`a`') hoặc chế độ cho phép đọc và ghi vào tệp ('`r +`'). Nếu ta bỏ qua đối số thứ hai, Python sẽ mở tệp ở chế độ chỉ đọc (read only) theo mặc định.

Hàm open() tự động tạo tệp ta đang ghi nếu tệp đó chưa tồn tại. Tuy nhiên, hãy cẩn thận khi mở tệp ở chế độ ghi ('w') vì nếu tệp tồn tại, Python sẽ xóa nội dung của tệp trước khi trả lại đối tượng tệp.

Chúng ta sử dụng phương thức write() trên đối tượng tệp để ghi một chuỗi vào tệp. Chương trình này không có đầu ra màn hình, nhưng nếu ta mở tệp tin program.txt, ta sẽ thấy một dòng:

```
I love programming.
```

Tệp này hoạt động giống như bất kỳ tệp nào khác trên máy tính. Ta có thể mở nó, viết văn bản mới vào nó, sao chép từ nó, dán vào nó, v.v.

Ghi chú: Python chỉ có thể viết chuỗi vào một văn bản. Nếu muốn lưu trữ dữ liệu số trong một tệp văn bản, ta sẽ phải chuyển đổi dữ liệu sang định dạng chuỗi trước tiên bằng cách sử dụng hàm str().

9.2.2. Ghi nhiều dòng

Hàm write() không thêm bất kỳ dòng mới nào vào văn bản ta ghi vào. Vì vậy, nếu ta viết nhiều hơn một dòng mà không bao gồm các ký tự dòng mới, tệp được ghi có thể không giống như mong muốn:

```
filename = 'programming.txt'  
with open(filename, 'w') as file_object:  
    file_object.write("I love programming.")  
    file_object.write("I love creating new games.")
```

Nếu ta mở programming.txt, ta sẽ thấy hai dòng ghép lại với nhau:

```
I love programming.I love creating new games.
```

Thêm tổ hợp dòng mới vào chuỗi, ta sẽ khiến các chuỗi được ghi vào tệp đúng theo dòng mong muốn:

```
filename = 'programming.txt'  
with open(filename, 'w') as file_object:  
    file_object.write("I love programming.\n")  
    file_object.write("I love creating new games.\n")
```

Đầu ra bây giờ xuất hiện trên các dòng riêng biệt:

```
I love programming.
```

```
I love creating new games.
```

Ta cũng có thể sử dụng dấu cách, ký tự tab và dòng trống để định dạng đầu ra của mình, giống như ta đã làm với đầu ra dựa trên cửa sổ đầu cuối.

9.2.3. Thêm vào một tệp

Nếu muốn thêm nội dung vào tệp thay vì ghi đè lên nội dung hiện có, ta có thể mở tệp ở chế độ nối thêm. Khi mở tệp ở chế độ nối thêm, Python không xóa nội dung của tệp trước khi trả lại đối tượng tệp. Bất kỳ dòng nào được ghi vào tệp sẽ được thêm vào cuối tệp. Nếu tệp chưa tồn tại, Python sẽ tạo một tệp trống.

Ta sửa đoạn mã ở trên bằng cách thêm vào một dòng có nội dung về lý do yêu thích lập trình:

```
filename = 'programming.txt'  
with open(filename, 'a') as file_object:  
    file_object.write("I also love finding meaning in large  
datasets.\n")  
    file_object.write("I love creating apps that can run in a  
browser.\n")
```

Chúng ta sử dụng đối số 'a' để mở tệp để thêm vào thay vì ghi lên tệp hiện có. Sau đó, chúng ta viết hai dòng mới, được thêm vào programming.txt. Do tệp đã có sẵn hai dòng và được thêm hai dòng mới nên nội dung tệp giờ sẽ có 4 dòng:

```
I love programming.  
I love creating new games.  
I also love finding meaning in large datasets.  
I love creating apps that can run in a browser.
```

9.3. Ngoại lệ

Python sử dụng các đối tượng đặc biệt được gọi là ngoại lệ để quản lý các lỗi phát sinh trong quá trình thực thi chương trình. Bất cứ khi nào một lỗi xảy ra khiến Python không chắc chắn phải làm gì tiếp theo, nó sẽ tạo ra một đối tượng ngoại lệ. Nếu ta viết code xử lý ngoại lệ, chương trình sẽ tiếp tục chạy. Nếu không xử lý ngoại lệ, chương trình sẽ tạm dừng và hiển thị theo dõi, bao gồm báo cáo về ngoại lệ đã được nêu ra.

Các trường hợp ngoại lệ được xử lý bằng các khối *try-except*. Một khối *try-except* yêu cầu Python làm điều gì đó, nhưng nó cũng cho Python biết phải làm gì nếu một ngoại lệ được đưa ra. Khi sử dụng các khối *try-except*, chương trình sẽ tiếp tục chạy ngay cả khi mọi thứ bắt đầu không ổn. Thay vì theo dõi, có thể gây nhầm lẫn cho người dùng khi đọc, người dùng sẽ thấy các thông báo lỗi thân thiện mà ta viết.

9.3.1. Xử lý ngoại lệ ZeroDivisionError

Ta xem một lỗi đơn giản khiến Python đưa ra một ngoại lệ. Ta có thể biết rằng không thể chia một số cho số 0, nhưng dù sao thì hãy yêu cầu Python làm điều đó:

```
print(5/0)
```

Tất nhiên Python không thể làm điều này, vì vậy ta nhận được một thông báo:

```
Traceback (most recent call last):
  File "division_calculator.py", line 1, in <module>
    print(5/0)
ZeroDivisionError: division by zero
```

Lỗi được báo cáo tại ZeroDivisionError: division by zero trong traceback, ZeroDivisionError, là một đối tượng ngoại lệ. Python tạo ra loại đối tượng này để đối phó với tình huống mà nó không thể làm những gì chương trình yêu cầu. Khi điều này xảy ra, Python sẽ dừng chương trình và cho chúng ta biết loại ngoại lệ đã được đưa ra. Chúng ta có thể sử dụng thông tin này để sửa đổi chương trình. Chúng ta sẽ cho Python biết phải làm gì khi loại ngoại lệ này xảy ra; theo cách đó, nếu nó xảy ra một lần nữa, chúng ta đã chuẩn bị sẵn sàng.

9.3.2. Sử dụng khối try-except

Khi cho rằng có thể xảy ra lỗi, ta có thể viết một khối *try-except* để xử lý trường hợp ngoại lệ có thể được đưa ra. Ta yêu cầu Python thử chạy một số code và cho nó biết phải làm gì nếu code dẫn đến một loại ngoại lệ cụ thể.

Dưới đây là khối try-except để xử lý ngoại lệ ZeroDivisionError:

```
try:
    print(5/0)
except ZeroDivisionError:
    print("You can't divide by zero!")
```

Chúng ta đặt `print(5/0)`, dòng gây ra lỗi, bên trong một khối *try*. Nếu code trong khối *try* hoạt động, Python sẽ bỏ qua khối *except*. Nếu code trong khối *try* gây ra lỗi, Python sẽ tìm một khối *except* có lỗi khớp với khối đã được nêu ra và chạy code trong khối đó.

Trong ví dụ này, code trong khối *try* tạo ra một ZeroDivisionError, vì vậy Python tìm kiếm một khối *except* cho nó biết cách phản hồi. Python sau đó chạy code trong khối đó và người dùng thấy một thông báo lỗi thân thiện thay vì một traceback:

```
You can't divide by zero!
```

Nếu có nhiều code hơn theo sau khối try-except, chương trình sẽ tiếp tục chạy vì chúng ta đã cho Python biết cách xử lý lỗi. Ta xem xét một ví dụ trong đó việc bắt lỗi có thể cho phép chương trình tiếp tục chạy.

9.3.3. Sử dụng ngoại lệ để ngăn chặn sự cố

Xử lý lỗi một cách chính xác đặc biệt quan trọng khi chương trình có nhiều việc phải làm sau khi lỗi xảy ra. Điều này xảy ra thường xuyên trong các chương trình nhắc người dùng nhập liệu. Nếu chương trình phản ứng với đầu vào không hợp lệ một cách thích hợp, nó có thể nhắc nhập đầu vào hợp lệ hơn thay vì gặp sự cố.

Hãy tạo một phép tính đơn giản chỉ thực hiện phép chia:

```
print("Give me two numbers, and I'll divide them.")
print("Enter 'q' to quit.")

while True:
    first_number = input("\nFirst number: ")
    if first_number == 'q':
        break
    second_number = input("Second number: ")
    if second_number == 'q':
        break
    answer = int(first_number) / int(second_number)
    print(answer)
```

Chương trình này nhắc người dùng nhập số thứ nhất, nếu người dùng không nhập q để thoát, thì người dùng sẽ nhập tiếp số thứ 2. Sau đó chúng ta chia hai số này để nhận được câu trả lời. Chương trình này không làm gì để xử lý lỗi, vì vậy yêu cầu nó chia cho 0 sẽ khiến nó gặp sự cố:

```
Give me two numbers, and I'll divide them.
Enter 'q' to quit.
```

```
First number: 5
Second number: 0
Traceback (most recent call last):
  File "division_calculator.py", line 9, in <module>
    answer = int(first_number) / int(second_number)
ZeroDivisionError: division by zero
```

Rõ ràng trong trường hợp này, sự cố đã xảy ra một cách đáng tiếc. Và nguy hiểm hơn khi để người dùng nhìn thấy traceback. Với người dùng không thiên về kỹ thuật, họ sẽ thấy bối rối khi traceback xuất hiện. Với một kẻ tấn công, việc nhìn thấy

traceback lại càng nguy hiểm hơn vì chúng có thể biết tên tệp, hoặc cấu trúc chương trình, hoặc logic chương trình. Với những kẻ tấn công có kỹ năng tốt, chúng sẽ lợi dụng thông tin này để tấn công vào chương trình.

9.3.4. Khối Else

Chúng ta có thể làm cho chương trình này có khả năng chống lỗi cao hơn bằng cách gói dòng, điều này có thể tạo ra lỗi trong một khối *try-except*. Lỗi xảy ra trên dòng thực hiện phép chia, vì vậy đó là nơi chúng ta sẽ đặt khối *try-except*. Ví dụ này cũng bao gồm một khối *else*. Bất kỳ code nào phụ thuộc vào khối *try* thực thi thành công sẽ đi vào khối *else*:

```
--snip--  
while True:  
    --snip--  
    if second_number == 'q':  
        break  
    try:  
        answer = int(first_number) / int(second_number)  
    except ZeroDivisionError:  
        print("You can't divide by 0!")  
    else:  
        print(answer)
```

Ta yêu cầu Python cố gắng hoàn thành phép chia trong khối *try*, khối này chỉ bao gồm code có thể gây ra lỗi. Bất kỳ code nào phụ thuộc vào khối *try* thành công sẽ được thêm vào khối *else*. Trong trường hợp này nếu phép chia thành công, chúng ta sử dụng khối *else* để in kết quả.

Khối *except* cho Python biết cách phản hồi khi phát sinh Lỗi *ZeroDivisionError*. Nếu khối *try* không thành công do lỗi chia cho 0, chúng ta sẽ in một thông báo thân thiện cho người dùng biết cách tránh loại lỗi này. Chương trình tiếp tục chạy và người dùng không bao giờ nhìn thấy traceback:

```
Give me two numbers, and I'll divide them.  
Enter 'q' to quit.  
First number: 5  
Second number: 0  
You can't divide by 0!
```

```
First number: 5  
Second number: 2
```

```
First number: q
```

Khối try-except-else hoạt động như sau: Python cố gắng chạy code trong khối try. Code duy nhất nên đi trong một khối try là code có thể gây ra một ngoại lệ được đưa ra. Đôi khi, ta sẽ có code bổ sung chỉ chạy nếu khối try thành công; code này nằm trong khối else. Khối except cho Python biết phải làm gì trong trường hợp một ngoại lệ nhất định phát sinh khi nó cố gắng chạy code trong khối try.

Bằng cách dự đoán các mã nguồn có khả năng xảy ra lỗi, ta có thể viết các chương trình mạnh mẽ tiếp tục chạy ngay cả khi chúng gặp phải dữ liệu không hợp lệ và tài nguyên bị thiếu. Code sẽ có khả năng chống lại các lỗi vô tình của người dùng và các cuộc tấn công nghiêm trọng.

9.3.5. Xử lý ngoại lệ FileNotFoundError

Một vấn đề phổ biến khi làm việc với tệp là xử lý tệp bị thiếu. Tệp ta đang tìm kiếm có thể ở một vị trí khác, tên tệp có thể bị sai chính tả hoặc tệp có thể hoàn toàn không tồn tại. Chúng ta có thể xử lý tất cả các tình huống này một cách dễ dàng với khối *try-except*.

Ta thử đọc một tệp không tồn tại. Chương trình sau cố gắng đọc nội dung của Alice in Wonderland, nhưng chưa có tệp alice.txt trong cùng thư mục với alice.py:

```
filename = 'alice.txt'
with open(filename, encoding='utf-8') as f:
    contents = f.read()
```

Có hai thay đổi ở đây. Một là việc sử dụng biến f để biểu diễn đối tượng tệp, đây là một quy ước chung. Thứ hai là việc sử dụng đối số encoding. Đối số này là cần thiết khi mã hóa mặc định của hệ thống không khớp với mã hóa của tệp đang được đọc.

Python không thể đọc từ một tệp bị thiếu, vì vậy nó tạo ra một ngoại lệ:

Traceback (most recent call last):

```
  File "alice.py", line 3, in <module>
      with open(filename, encoding='utf-8') as f:
FileNotFoundError: [Errno 2] No such file or directory: 'alice.txt'
```

Dòng cuối cùng của traceback báo cáo lỗi FileNotFoundError: đây là trường hợp ngoại lệ mà Python tạo ra khi nó không thể tìm thấy tệp mà nó đang cố gắng mở.

Trong ví dụ này, hàm open() tạo ra lỗi, vì vậy để xử lý nó, khối try sẽ bắt đầu bằng dòng chứa open():

```
filename = 'alice.txt'  
try:  
    with open(filename, encoding='utf-8') as f:  
        contents = f.read()  
except FileNotFoundError:  
    print(f"Sorry, the file {filename} does not exist.")
```

Trong ví dụ này, code trong khối try tạo ra một FileNotFoundError, vì vậy Python sẽ tìm một khối except phù hợp với lỗi đó. Python sau đó chạy code trong khối đó và kết quả là một thông báo lỗi thân thiện thay vì một traceback:

```
Sorry, the file alice.txt does not exist.
```

Chương trình không có gì phải làm nếu tệp không tồn tại, vì vậy code xử lý lỗi không bổ sung nhiều cho chương trình này. Ta sẽ xây dựng dựa trên ví dụ này và xem việc xử lý ngoại lệ có thể hữu ích như thế nào khi ta đang làm việc với nhiều tệp.

9.3.6. Phân tích dữ liệu văn bản

Chúng ta có thể phân tích các tệp văn bản chứa toàn bộ sách. Nhiều tác phẩm văn học cổ điển có sẵn dưới dạng tệp văn bản đơn giản vì chúng thuộc phạm vi công cộng. Các văn bản được sử dụng trong phần này đến từ Dự án Gutenberg (<http://gutenberg.org/>). Project Gutenberg duy trì một bộ sưu tập các tác phẩm văn học có sẵn trong công cộng và đó là một nguồn tài nguyên tuyệt vời nếu ta muốn làm việc với các văn bản văn học trong các dự án lập trình của mình.

Hãy xem văn bản của Alice in Wonderland và có gắng đếm số từ trong văn bản. Chúng ta sẽ sử dụng phương thức chuỗi split(), có thể tạo danh sách các từ từ một chuỗi. Dưới đây là những gì split() thực hiện với một chuỗi chỉ chứa tiêu đề "Alice in Wonderland":

```
>>> title = "Alice in Wonderland"  
>>> title.split()  
['Alice', 'in', 'Wonderland']
```

Phương thức split() tách một chuỗi thành các phần ở bất kỳ nơi nào nó tìm thấy khoảng trắng và lưu trữ tất cả các phần của chuỗi trong một danh sách. Kết quả là một danh sách các từ trong chuỗi, mặc dù một số dấu câu cũng có thể xuất hiện

cùng với một số từ. Để đếm số từ trong Alice in Wonderland, chúng ta sẽ sử dụng `split()` trên toàn bộ văn bản. Sau đó, chúng ta sẽ đếm các mục trong danh sách để biết sơ bộ về số lượng từ trong văn bản:

```
filename = 'alice.txt'
try:
    with open(filename, encoding='utf-8') as f:
        contents = f.read()
except FileNotFoundError:
    print(f"Sorry, the file {filename} does not exist.")
else:
    # Count the approximate number of words in the file.
    words = contents.split()
    num_words = len(words)
    print(f"The file {filename} has about {num_words} words.")
```

Chúng ta đã di chuyển tệp `alice.txt` đến đúng thư mục, vì vậy khối `try` sẽ hoạt động. Chúng ta lấy nội dung chuỗi, hiện chưa toàn bộ văn bản của Alice in Wonderland dưới dạng một chuỗi dài và sử dụng phương thức `split()` để tạo danh sách tất cả các từ trong cuốn sách. Khi chúng ta sử dụng `len()` trong danh sách này để kiểm tra độ dài của nó, chúng ta sẽ nhận được một số lượng từ gần đúng trong chuỗi ban đầu. Tại câu lệnh `print()` cuối cùng, chúng ta in một câu lệnh báo cáo có bao nhiêu từ được tìm thấy trong tệp. Code này được đặt trong khối `else` vì nó sẽ chỉ hoạt động nếu code trong khối `try` được thực thi thành công hoàn toàn. Kết quả cho chúng ta biết có bao nhiêu từ trong `alice.txt`:

```
The file alice.txt has about 29465 words.
```

Con số này hơi cao vì thông tin bổ sung được nhà xuất bản cung cấp trong tệp văn bản được sử dụng ở đây, nhưng đó là con số gần đúng với độ dài của Alice in Wonderland.

9.3.7. Làm việc với nhiều tệp

Ta sẽ thêm nhiều sách hơn để phân tích. Nhưng trước khi thực hiện, hãy chuyển phần lớn chương trình này sang một hàm có tên là `count_words()`. Làm như vậy, sẽ dễ dàng hơn để chạy phân tích cho nhiều sách:

```
def count_words(filename):
    """Count the approximate number of words in a file."""
    try:
        with open(filename, encoding='utf-8') as f:
            contents = f.read()
```

```

except FileNotFoundError:
    print(f"Sorry, the file {filename} does not exist.")
else:
    words = contents.split()
    num_words = len(words)
    print(f"The file {filename} has about {num_words} words.")
filename = 'alice.txt'
count_words(filename)

```

Hầu hết code này là không thay đổi. Chúng ta chỉ cần thay lè và chuyển nó vào phần thân của `count_words()`. Một thói quen tốt là luôn cập nhật nhận xét khi đang sửa đổi một chương trình, vì vậy chúng ta đã thay đổi nhận xét thành chuỗi tài liệu và đổi tên nó một chút.

Bây giờ chúng ta có thể viết một vòng lặp đơn giản để đếm các từ trong bất kỳ văn bản nào mà chúng ta muốn phân tích. Chúng ta thực hiện việc này bằng cách lưu trữ tên của các tệp mà chúng ta muốn phân tích trong một danh sách, sau đó chúng ta gọi `count_words()` cho mỗi tệp trong danh sách. Chúng ta sẽ cố gắng đếm các từ cho *Alice in Wonderland*, *Siddhartha*, *Moby Dick* và *Little Women*, tất cả đều có sẵn trong miền công cộng.

Ta có ý để *siddhartha.txt* ra khỏi thư mục chứa *word_count.py*, vì vậy ta có thể thấy chương trình đã được viết xử lý trường hợp tệp bị thiếu hoạt động chính xác:

```

def count_words(filename):
    --snip--
filenames = ['alice.txt', 'siddhartha.txt', 'moby_dick.txt',
'little_women.txt']
for filename in filenames:
    count_words(filename)

```

Tệp *siddhartha.txt* bị thiếu không ảnh hưởng đến phần còn lại của quá trình thực thi chương trình:

```

The file alice.txt has about 29465 words.
Sorry, the file siddhartha.txt does not exist.
The file moby_dick.txt has about 215830 words.
The file little_women.txt has about 189079 words.

```

Sử dụng khôi `try-except` trong ví dụ này cung cấp hai lợi thế đáng kể. Ta ngăn người dùng nhìn thấy traceback và ta để chương trình tiếp tục phân tích các văn bản mà nó có thể tìm thấy. Nếu ta không bắt được `FileNotFoundException` mà *siddhartha.txt* đã nêu ra, người dùng sẽ thấy toàn bộ traceback và chương trình sẽ ngừng chạy khi

cố gắng phân tích Siddhartha. Nó sẽ không bao giờ phân tích Moby Dick hoặc Little Women.

9.3.8. *Cách giải lỗi*

Trong ví dụ trước, chúng ta đã thông báo cho người dùng rằng một trong các tệp không khả dụng. Nhưng ta không cần phải báo cáo mọi trường hợp ngoại lệ được bắt gặp. Đôi khi, ta muốn chương trình im lặng khi một ngoại lệ xảy ra và tiếp tục như thể không có gì xảy ra. Để làm cho một chương trình không hoạt động một cách âm thầm, ta viết một khối try như bình thường, nhưng ta yêu cầu Python không làm gì trong khối except một cách rõ ràng. Python có một câu lệnh truyền yêu cầu nó không làm gì trong một khối:

```
def count_words(filename):
    """Count the approximate number of words in a file."""
    try:
        --snip--
    except FileNotFoundError:
        pass
    else:
        --snip--
filenames = ['alice.txt', 'siddhartha.txt', 'moby_dick.txt',
'little_women.txt']
for filename in filenames:
    count_words(filename)
```

Sự khác biệt duy nhất giữa danh sách này và danh sách trước đó là câu lệnh pass. Bây giờ khi một FileNotFoundError được nâng lên, code trong khối except sẽ chạy, nhưng không có gì xảy ra. Không có dấu vết nào được tạo ra và không có đầu ra nào để phản hồi lại lỗi đã phát sinh. Người dùng thấy số lượng từ cho mỗi tệp tồn tại, nhưng họ không thấy bất kỳ dấu hiệu nào cho thấy không tìm thấy tệp:

```
The file alice.txt has about 29465 words.
The file moby_dick.txt has about 215830 words.
The file little_women.txt has about 189079 words.
```

Câu lệnh pass cũng hoạt động như một trình giữ chỗ. Đó là lời nhắc nhở rằng ta đang chọn không làm gì tại một thời điểm cụ thể trong quá trình thực hiện chương trình và ta có thể muốn làm điều gì đó ở đó sau. Ví dụ: trong chương trình này, chúng ta có thể quyết định ghi bất kỳ tên tệp nào bị thiếu vào tệp có tên missing_files.txt. Người dùng sẽ không thấy tệp này, nhưng chúng ta có thể đọc tệp và xử lý mọi văn bản bị thiếu.

9.3.9. Quyết định lỗi nào cần báo cáo

Làm cách nào để biết khi nào cần thông báo lỗi cho người dùng của mình và khi nào thì lỗi âm thầm? Nếu người dùng biết văn bản nào cần được phân tích, họ có thể đánh giá cao một thông báo cho họ biết lý do tại sao một số văn bản không được phân tích. Nếu người dùng muốn xem một số kết quả nhưng không biết sách nào được cho là sẽ được phân tích, họ có thể không cần biết rằng một số văn bản không có sẵn. Cung cấp cho người dùng thông tin mà họ không tìm kiếm có thể làm giảm khả năng sử dụng của chương trình. Các cấu trúc xử lý lỗi của Python cung cấp cho ta khả năng kiểm soát tốt hơn đối với số lượng chia sẻ với người dùng khi có sự cố; tùy thuộc vào ta để quyết định lượng thông tin cần chia sẻ.

Code được viết tốt, được kiểm tra đúng cách sẽ không dễ mắc các lỗi nội bộ, chẳng hạn như lỗi cú pháp hoặc lỗi logic. Nhưng mỗi khi chương trình phụ thuộc vào thứ gì đó bên ngoài, chẳng hạn như đầu vào của người dùng, sự tồn tại của tệp hoặc tính khả dụng của kết nối mạng, thì sẽ có khả năng xảy ra ngoại lệ. Một chút kinh nghiệm sẽ giúp ta biết nơi đưa các khối xử lý ngoại lệ vào chương trình và mức độ báo cáo cho người dùng về các lỗi phát sinh.

9.4. Lưu trữ dữ liệu

Nhiều chương trình sẽ yêu cầu người dùng nhập một số loại thông tin nhất định. Ta có thể cho phép người dùng lưu trữ các tùy chọn trong trò chơi hoặc cung cấp dữ liệu để hiển thị. Cho dù trọng tâm của chương trình là gì, ta sẽ lưu trữ thông tin mà người dùng cung cấp trong cấu trúc dữ liệu như danh sách và từ điển. Khi người dùng đóng một chương trình, hầu như ta sẽ luôn muốn lưu thông tin họ đã nhập. Một cách đơn giản để thực hiện việc này liên quan đến việc lưu trữ dữ liệu của ta bằng cách sử dụng mô-đun json.

Module json cho phép kết xuất các cấu trúc dữ liệu Python đơn giản vào một tệp và tải dữ liệu từ tệp đó vào lần chạy chương trình tiếp theo. Ta cũng có thể sử dụng json để chia sẻ dữ liệu giữa các chương trình Python khác nhau. Tốt hơn nữa, định dạng dữ liệu JSON không dành riêng cho Python, vì vậy ta có thể chia sẻ dữ liệu ta lưu trữ ở định dạng JSON với những người làm việc bằng nhiều ngôn ngữ lập trình khác. Đó là một định dạng hữu ích và di động, đồng thời dễ học.

9.4.1. Sử dụng `json.dump()` và `json.load()`

Ta viết một chương trình ngắn lưu trữ một tập hợp các số và một chương trình khác đọc những số này trở lại bộ nhớ. Chương trình đầu tiên sẽ sử dụng `json.dump()` để lưu trữ tập hợp các số và chương trình thứ hai sẽ sử dụng `json.load()`.

Hàm `json.dump()` nhận hai đối số: một phần dữ liệu để lưu trữ và một đối tượng tệp mà nó có thể sử dụng để lưu trữ dữ liệu. Đây là cách ta có thể sử dụng `json.dump()` để lưu trữ danh sách các số:

```
import json
numbers = [2, 3, 5, 7, 11, 13]
filename = 'numbers.json'
with open(filename, 'w') as f:
    json.dump(numbers, f)
```

Đầu tiên chúng ta nhập mô-đun `json` và sau đó tạo một danh sách các số để làm việc. Chúng ta chọn một tên tệp để lưu danh sách các số. Thông thường sử dụng đuôi tệp `.json` để cho biết rằng dữ liệu trong tệp được lưu trữ ở định dạng JSON. Sau đó, chúng ta mở tệp ở chế độ ghi, cho phép `json` ghi dữ liệu vào tệp. Chúng ta sử dụng hàm `json.dump()` để lưu trữ danh sách số trong tệp `numbers.json`.

Chương trình này không có đầu ra, nhưng khi mở `numbers.json` và xem, dữ liệu được lưu trữ ở định dạng giống như Python:

```
[2, 3, 5, 7, 11, 13]
```

Bây giờ chúng ta sẽ viết một chương trình sử dụng `json.load()` để đọc lại danh sách trong bộ nhớ:

```
import json
filename = 'numbers.json'
with open(filename) as f:
    numbers = json.load(f)
print(numbers)
```

Chúng ta đảm bảo đọc từ cùng một tệp mà chúng ta đã viết. Lần này khi chúng ta mở tệp, chúng ta mở nó ở chế độ đọc vì Python chỉ cần đọc từ tệp. Sau đó, sử dụng hàm `json.load()` để tải thông tin được lưu trữ trong `numbers.json` và chúng ta gán nó vào biến `numbers`. Cuối cùng, chúng ta in danh sách các số đã khôi phục và thấy rằng đó là danh sách giống như danh sách được tạo trong `number_writer.py`:

```
[2, 3, 5, 7, 11, 13]
```

Đây là một cách đơn giản để chia sẻ dữ liệu giữa hai chương trình.

9.4.2. Lưu và đọc dữ liệu do người dùng tạo

Lưu dữ liệu bằng json rất hữu ích khi đang làm việc với dữ liệu do người dùng tạo, bởi vì nếu ta không lưu trữ thông tin của người dùng bằng cách nào đó, ta sẽ mất thông tin đó khi chương trình ngừng chạy. Hãy xem một ví dụ trong đó chúng ta nhắc người dùng tên của họ trong lần đầu tiên họ chạy một chương trình và sau đó ghi nhớ tên của họ khi họ chạy lại chương trình.

Hãy bắt đầu bằng cách lưu trữ tên của người dùng:

```
import json
username = input("What is your name? ")
filename = 'username.json'
with open(filename, 'w') as f:
    json.dump(username, f)
    print(f"We'll remember you when you come back, {username}!")
```

Chúng ta nhắc nhập tên người dùng để lưu trữ. Tiếp theo, ta sử dụng json.dump(), chuyển cho nó một tên người dùng và một đối tượng tệp, để lưu tên người dùng trong một tệp. Sau đó, chúng ta in một thông báo cho người dùng biết rằng ta đã lưu trữ thông tin của họ:

```
What is your name? Eric
We'll remember you when you come back, Eric!
```

Bây giờ, ta viết một chương trình mới chào mừng người dùng có tên đã được lưu trữ:

```
import json
filename = 'username.json'
with open(filename) as f:
    username = json.load(f)
    print(f"Welcome back, {username}!")
```

Chúng ta sử dụng json.load() để đọc thông tin được lưu trữ trong username.json và gán nó cho biến username. Bây giờ chúng ta đã khôi phục tên người dùng, chúng ta có thể chào mừng họ trở lại:

```
Welcome back, Eric!
```

Chúng ta cần kết hợp hai chương trình này thành một tệp. Khi ai đó chạy remember_me.py, chúng ta muốn truy xuất tên người dùng của họ từ bộ nhớ nếu có thể; do đó, ta sẽ bắt đầu với một khối try để khôi phục tên người dùng. Nếu tệp

username.json không tồn tại, chúng ta sẽ có lời nhắc khói ngoại trừ cho tên người dùng và lưu trữ nó trong username.json cho lần sau:

```
import json
# Load the username, if it has been stored previously.
# Otherwise, prompt for the username and store it.
filename = 'username.json'
try:
    with open(filename) as f:
        username = json.load(f)
except FileNotFoundError:
    username = input("What is your name? ")
    with open(filename, 'w') as f:
        json.dump(username, f)
    print(f"We'll remember you when you come back, {username}!")
else:
    print(f"Welcome back, {username}!")
```

Logic code vẫn giữ nguyên; các khối code từ hai ví dụ cuối cùng chỉ được kết hợp thành một tệp. Chúng ta cố gắng mở tệp username.json. Nếu tệp này tồn tại, ta đọc lại tên người dùng vào bộ nhớ và in thông báo chào mừng người dùng trở lại trong khối else. Nếu đây là lần đầu tiên người dùng chạy chương trình, username.json sẽ không tồn tại và lỗi FileNotFoundError sẽ xảy ra. Python sẽ chuyển sang khối except nơi chúng ta nhắc người dùng nhập tên người dùng của họ. Sau đó, chúng ta sử dụng json.dump() để lưu tên người dùng và in lời chào.

Cho dù khối nào thực thi, kết quả là tên người dùng và lời chào thích hợp. Nếu đây là lần đầu tiên chương trình chạy, đây là kết quả:

```
What is your name? Eric
We'll remember you when you come back, Eric!
Nếu không thì kết quả là:
Welcome back, Eric!
```

Đây là kết quả ta thấy nếu chương trình đã được chạy ít nhất một lần.

9.4.3. Tái cấu trúc

Thông thường, sẽ đến một thời điểm mà code được viết sẽ hoạt động, nhưng ta sẽ nhận ra rằng có thể cải thiện code bằng cách chia nó thành một loạt các hàm có các công việc cụ thể. Quá trình này được gọi là tái cấu trúc. Tái cấu trúc làm cho code sạch hơn, dễ hiểu hơn và dễ mở rộng hơn.

Chúng ta có thể tái cấu trúc lại remember_me.py bằng cách chuyển phần lớn logic của nó vào một hoặc nhiều hàm. Trọng tâm của remember_me.py là chào hỏi người dùng, vì vậy hãy chuyển tất cả code hiện có của chúng ta vào một hàm có tên là greet_user():

```
import json

def greet_user():
    """Greet the user by name."""
    filename = 'username.json'
    try:
        with open(filename) as f:
            username = json.load(f)
    except FileNotFoundError:
        username = input("What is your name? ")
        with open(filename, 'w') as f:
            json.dump(username, f)
            print(f"We'll remember you when you come back,\n{username}!")
    else:
        print(f"Welcome back, {username}!")

greet_user()
```

Bởi vì chúng ta hiện đang sử dụng một hàm, chúng ta cập nhật các nhận xét bằng một chuỗi tài liệu phản ánh cách chương trình hiện đang hoạt động. Tệp này gọn gàng hơn một chút, nhưng hàm greet_user() còn làm được nhiều việc hơn là chỉ chào hỏi người dùng — nó còn truy xuất tên người dùng được lưu trữ nếu có và nhắc nhập tên người dùng mới nếu không tồn tại.

Hãy cấu trúc lại greet_user() để nó không thực hiện quá nhiều tác vụ khác nhau. Chúng ta sẽ bắt đầu bằng cách di chuyển code để truy xuất tên người dùng được lưu trữ sang một chức năng riêng biệt:

```
import json

def get_stored_username():
    """Get stored username if available."""
    filename = 'username.json'
    try:
        with open(filename) as f:
            username = json.load(f)
    except FileNotFoundError:
        return None
    else:
        return username
```

```

def greet_user():
    """Greet the user by name."""
    username = get_stored_username()
    if username:
        print(f"Welcome back, {username}!")
    else:
        username = input("What is your name? ")
        filename = 'username.json'
        with open(filename, 'w') as f:
            json.dump(username, f)
        print(f"We'll remember you when you come back,\n{username}!")
    greet_user()

```

Hàm mới `get_stored_username()` có mục đích rõ ràng, như đã nêu trong docstring. Hàm này truy xuất tên người dùng được lưu trữ và trả về tên người dùng nếu nó tìm thấy. Nếu tệp `username.json` không tồn tại, hàm trả về `None`. Đây là một phương pháp hay: một hàm phải trả về giá trị mà được mong đợi hoặc trả về `None`. Điều này cho phép chúng ta thực hiện một bài kiểm tra đơn giản với giá trị trả về của hàm. Chúng ta sẽ in một thông báo chào mừng trở lại cho người dùng nếu nỗ lực truy xuất tên người dùng thành công và nếu không, chúng ta sẽ nhắc nhập tên người dùng mới.

Chúng ta nên thêm một khối code nữa ra khỏi `greet_user()`. Nếu tên người dùng không tồn tại, chúng ta sẽ di chuyển code nhắc nhập tên người dùng mới sang một chức năng dành riêng cho mục đích đó:

```

import json
def get_stored_username():
    """Get stored username if available."""
    --snip--
def get_new_username():
    """Prompt for a new username."""
    username = input("What is your name? ")
    filename = 'username.json'
    with open(filename, 'w') as f:
        json.dump(username, f)
    return username
def greet_user():
    """Greet the user by name."""
    username = get_stored_username()
    if username:

```

```

        print(f"Welcome back, {username}!")
else:
    username = get_new_username()
    print(f"We'll remember you when you come back, {username}!")
greet_user()

```

Mỗi chức năng trong phiên bản cuối cùng này của remember_me.py đều có một mục đích rõ ràng. Chúng ta gọi greet_user() và hàm đó in ra một thông báo thích hợp: nó chào mừng trở lại một người dùng hiện tại hoặc chào mừng một người dùng mới. Nó thực hiện điều này bằng cách gọi get_stored_username(), chỉ chịu trách nhiệm truy xuất tên người dùng được lưu trữ nếu tên người dùng tồn tại. Cuối cùng, greet_user() gọi get_new_username() nếu cần thiết, chỉ có nhiệm vụ lấy tên người dùng mới và lưu trữ nó. Việc phân chia công việc này là một phần thiết yếu của việc viết code rõ ràng sẽ dễ bảo trì và mở rộng.

Bài tập chương 9

9-1. Learning Python: Mở một tệp trống trong trình soạn thảo văn bản và viết một vài dòng tóm tắt những gì ta đã học về Python cho đến nay. Bắt đầu từng dòng với cụm từ Trong Python, ta có thể. . . Lưu tệp dưới dạng learning_python.txt trong cùng thư mục với các bài tập trong chương này. Viết chương trình đọc tệp và in những gì đã viết ba lần. Lần thứ nhất in toàn bộ nội dung đọc được của file; Lần thứ hai lặp qua đối tượng file và lần thứ ba bằng cách lưu trữ các dòng trong danh sách và làm việc với danh sách ngoài khói đọc.

9-2. Learning C: Ta có thể sử dụng phương thức Replace() để thay thế bất kỳ từ nào trong chuỗi bằng một từ khác. Dưới đây là một ví dụ nhanh về cách thay thế "dog" bằng "cat" trong một câu:

```

>>> message = "I really like dogs."
>>> message.replace('dog', 'cat')
'I really like cats.'

```

Đọc từng dòng từ tệp learning_python.txt tại bài 9-1, và thay thế từ Python bằng tên của ngôn ngữ khác, chẳng hạn như C. In từng dòng đã sửa đổi ra màn hình.

9-3. Guest: Viết chương trình nhắc người dùng nhập tên của họ. Khi họ phản hồi, hãy viết tên của họ vào một tệp tin có tên là guest.txt.

9-4. Guest Book: Viết một vòng lặp trong khi nhắc người dùng tên của họ. Khi họ nhập tên, hãy in lời chào ra màn hình và thêm một dòng ghi lại chuyến thăm của

họ trong một tệp có tên là guest_book.txt. Đảm bảo mỗi mục nhập xuất hiện trên một dòng mới trong tệp.

9-5. Programming Poll: Viết một vòng lặp while hỏi mọi người tại sao họ thích lập trình. Mỗi khi ai đó nhập lý do, hãy thêm lý do của họ vào tệp lưu trữ tất cả các câu trả lời.

9-6. Addition: Một vấn đề phổ biến khi nhắc nhập số xảy ra khi mọi người cung cấp chuỗi thay vì số. Khi cố gắng chuyển đổi đầu vào thành int, ta sẽ nhận được ValueError. Viết chương trình yêu cầu người dùng nhập vào hai số, sau đó in ra giá trị của tổng. Bắt lỗi ValueError nếu giá trị đầu vào không phải là số và in một thông báo lỗi thân thiện. Kiểm tra chương trình bằng cách nhập hai số và sau đó nhập một số văn bản thay vì số.

9-7. Addition Calculator: đưa mã nguồn trong bài 9-6 vào trong vòng lặp while để người dùng có thể tiếp tục nhập số ngay cả khi họ tạo ra lỗi hay nhập chuỗi thay vì nhập số.

9-8. Cats and Dogs: Tạo ra 2 tệp: cats.txt và dogs.txt và lưu ít nhất tên của 3 con mèo vào tệp thứ nhất, và ít nhất tên của 3 con mèo vào tệp thứ hai. Viết một chương trình cố gắng đọc các tệp này và in nội dung của tệp ra màn hình. Gom mã nguồn vào khối try-except để bắt lỗi FileNotFoundError và in ra thông báo thân thiện nếu một tệp bị thiếu. Di chuyển một tệp sang thư mục khác để thấy mã nguồn chạy chính xác.

9-9. Silent Cats and Dogs: Thay đổi khôi except trong bài 9-8 để chương trình không đưa ra thông báo nào nếu tệp không tìm thấy.

9-10. Common Words: Tìm một đoạn văn bản trên internet và lưu vào tệp posts.txt. Sử dụng phương thức count() để đếm xem một từ xuất hiện trong văn bản đó bao nhiêu lần. Sử dụng phương thức lower() để tăng tính chính xác (đếm cả chữ hoa, chữ thường, chữ tiêu đề).

9-11. Favorite Number: Viết chương trình nhắc số yêu thích của người dùng. Sử dụng json.dump() để lưu trữ số này trong một tệp. Viết một chương trình riêng biệt đọc giá trị này và in thông báo: “Số yêu thích của ta là...”

9-12. Favorite Number Remembered: Ghép hai chương trình của bài tập 9-11 thành một tệp. Nếu số đã được lưu trữ thì in ra thông báo. Nếu không, nhắc người dùng nhập vào số và lưu trong tệp .txt, chạy chương trình lần nữa để thấy có thông báo.

Kết chương

Trong chương này, chúng ta đã học cách làm việc với tệp. Ta đã học cách đọc một toàn bộ tệp cùng một lúc và đọc qua nội dung của tệp từng dòng một. Ta đã học cách ghi vào tệp và nối văn bản vào cuối tệp. Ta cũng đọc về các ngoại lệ và cách xử lý các ngoại lệ mà ta có thể gặp trong các chương trình của mình. Cuối cùng, ta đã học được cách lưu trữ cấu trúc dữ liệu Python để có thể lưu thông tin mà người dùng cung cấp, ngăn họ phải bắt đầu lại mỗi lần chạy chương trình.

CHƯƠNG 10. DỰ ÁN

Phần này bao gồm ba loại dự án và ta đọc có thể chọn thực hiện bất kỳ hoặc tất cả các dự án này theo bất kỳ thứ tự nào. Đây là một mô tả ngắn gọn của mỗi dự án để giúp ta đọc quyết định nên đi sâu vào trước tiên.

Dự án 1 - Cuộc xâm lăng của quái vật: Làm game với Python

Trong dự án Cuộc xâm lăng của quái vật, ta sẽ sử dụng Gói Pygame để phát triển một trò chơi 2D trong đó mục đích là bắn hạ những con quái vật khi chúng được thả xuống màn hình ở các cấp độ tăng tốc độ và các mức độ khác nhau. Khi kết thúc dự án, ta sẽ học được các kỹ năng giúp cho phép ta phát triển các trò chơi 2D trong Pygame.

Dự án 2 - Trực quan hóa dữ liệu (Data Visualization)

Dự án Trực quan hóa dữ liệu, trong đó ta sẽ học tạo dữ liệu và tạo một loạt các hình ảnh hóa chức năng và đẹp mắt dữ liệu đó bằng Matplotlib và Plotly. Dự án dạy ta cách truy cập dữ liệu từ các nguồn trực tuyến và cung cấp dữ liệu vào một gói trực quan hóa để tạo lô dữ liệu thời tiết và bản đồ hoạt động động đất toàn cầu. Cuối cùng, phần kết hướng dẫn ta cách viết một chương trình để tự động tải xuống và trực quan hóa dữ liệu.

Học cách tạo hình ảnh trực quan cho phép ta khám phá lĩnh vực khai thác dữ liệu, là một kỹ năng rất được săn đón trên thế giới ngày nay.

Dự án 3 - Ứng dụng web (web applications)

Trong dự án Ứng dụng web, ta sẽ sử dụng Gói Django để tạo một ứng dụng web đơn giản cho phép người dùng tạo một tạp chí về bất kỳ chủ đề nào mà họ đang tìm hiểu. Người dùng sẽ tạo tài khoản với tên người dùng và mật khẩu, nhập chủ đề và sau đó viết các mục về những gì họ đang học. Ta cũng sẽ học cách triển khai ứng dụng để mọi người trên thế giới có thể truy cập. Sau khi hoàn thành dự án này, ta sẽ có thể bắt đầu xây dựng các ứng dụng web đơn giản và sẽ sẵn sàng nghiên cứu kỹ hơn tài nguyên về xây dựng ứng dụng với Django.

10.1. Trò chơi cuộc xâm lăng của quái vật

Nội dung trong phần này sẽ hướng dẫn từng bước xây dựng trò chơi cuộc xâm lăng của quái vật dựa trên thư viện pygame.

10.1.1. Tạo ra tàu bắn đạn

Chúng ta cùng xây dựng một trò chơi có tên là Alien Invasion-Cuộc xâm lăng của quái vật! Chúng ta sẽ sử dụng Pygame, một bộ sưu tập các mô-đun Python mạnh mẽ, thú vị để quản lý đồ họa, hoạt ảnh và thậm chí cả âm thanh, giúp cho dễ dàng hơn để ta dựng các trò chơi phức tạp. Pygame giúp xử lý các tác vụ trên màn hình như vẽ hình ảnh, chúng ta có thể tập trung vào các nhiệm vụ cao hơn của logic trò chơi.

Đầu tiên cần thiết lập Pygame, sau đó tạo một tàu tên lửa di chuyển sang phải và trái và những viên đạn mới để đáp ứng với đầu vào của người chơi. Ở phần tiếp theo, chúng ta sẽ tạo ra các con quái vật để tiêu diệt và sau đó tiếp tục cài tiến trò chơi bằng cách đặt giới hạn về số lượng tàu ta có và có thể sử dụng thêm bảng điểm.

Trong khi xây dựng trò chơi này, ta cũng sẽ học cách quản lý các dự án lớn kéo dài nhiều phần. Chúng ta sẽ cấu trúc lại rất nhiều mã và quản lý nội dung để tổ chức dự án và làm cho mã hiệu quả.

10.1.1.1. Lập kế hoạch dự án

Khi đang xây dựng một dự án lớn, điều quan trọng là phải chuẩn bị một kế hoạch trước khi ta bắt đầu viết mã. Kế hoạch sẽ giúp chúng ta tập trung và khiến ta có nhiều khả năng hoàn thành dự án hơn.

Ta cùng viết mô tả về cách chơi chung. Mặc dù mô tả sau đây không bao gồm mọi chi tiết của Cuộc xâm lược của quái vật, nhưng nó cung cấp một ý tưởng rõ ràng về cách bắt đầu xây dựng trò chơi:

Trong Alien Invasion, người chơi điều khiển một con tàu tên lửa xuất hiện ở dưới cùng giữa màn hình. Người chơi có thể di chuyển con tàu phải và trái bằng cách sử dụng các phím mũi tên và bắn đạn bằng cách sử dụng phím cách. Khi trò chơi bắt đầu, một đội quái vật bay lượn trên bầu trời và di chuyển trên và xuống màn hình. Người chơi bắn và tiêu diệt quái vật. Nếu người chơi bắn tắt cả những con quái vật, một đội mới sẽ xuất hiện và di chuyển nhanh hơn đội trước. Nếu có quái vật tấn công tàu của người chơi hoặc đến cuối màn hình, người chơi mất một con tàu. Nếu người chơi mất ba tàu, trò chơi kết thúc.

Đối với giai đoạn phát triển đầu tiên, chúng ta sẽ tạo ra một con tàu có thể di chuyển sang trái và phải và đạn bắn khi người chơi nhấn phím cách. Sau khi thiết lập hành vi này, chúng ta có thể tạo con quái vật và tinh chỉnh lối chơi.

10.1.1.2. Cài đặt Pygame

Trước khi ta bắt đầu viết mã, hãy cài đặt Pygame. Module pip giúp ta tải xuống và cài đặt các gói Python. Để cài đặt Pygame, hãy nhập thông tin sau lệnh tại dấu nhắc đầu cuối:

```
$ python -m pip install --user pygame
```

Lệnh này yêu cầu Python chạy mô-đun pip và cài đặt gói pygame vào bản cài đặt Python của người dùng hiện tại. Nếu sử dụng một lệnh không phải là python để chạy các chương trình hoặc bắt đầu một phiên đầu cuối, chẳng hạn như python3, lệnh sẽ giống như sau:

```
$ python3 -m pip install --user pygame
```

Ghi chú: Nếu lệnh này không hoạt động trên macOS, hãy thử chạy lại lệnh mà không có --user.

10.1.1.3. Bắt đầu dự án trò chơi

Chúng ta sẽ bắt đầu xây dựng trò chơi bằng cách tạo một cửa sổ Pygame trống. Sau đó, chúng ta sẽ vẽ các yếu tố trò chơi, chẳng hạn như con tàu và quái vật trên cửa sổ này. Chúng ta cũng sẽ làm cho trò chơi phản hồi với thông tin nhập của người dùng, đặt màu nền và tải hình ảnh con tàu.

Tạo cửa sổ trò chơi và đáp ứng yêu cầu đầu vào của người chơi

Chúng ta sẽ tạo một cửa sổ Pygame trống bằng cách tạo một lớp để đại diện cho trò chơi. Trong trình soạn thảo văn bản, hãy tạo một giao diện mới và lưu nó dưới dạng alien_invasion.py; sau đó thêm code như sau:

```
import sys
import pygame
from settings import Settings
from ship import Ship

class AlienInvasion:
    """Overall class to manage game assets and behavior."""
    def __init__(self):
        """Initialize the game, and create game resources."""
①     pygame.init()
```

```

        self.settings = Settings()

②    self.screen = pygame.display.set_mode(
            (1200, 800))
        pygame.display.set_caption("Alien Invasion")

        self.ship = Ship(self)

def run_game(self):
    """Start the main loop for the game."""
③    while True:
        # Watch for keyboard and mouse events.
④    for event in pygame.event.get():
⑤        if event.type == pygame.QUIT:
            sys.exit()
        self.screen.fill(self.settings.bg_color)
        self.ship.blitme()

        # Make the most recently drawn screen visible.
⑥    pygame.display.flip()

if __name__ == '__main__':
    # Make a game instance, and run the game.
    ai = AlienInvasion()
    ai.run_game()

```

Đầu tiên, chúng ta nhập các mô-đun `sys` và `pygame`. Module `pygame` chứa chức năng chúng ta cần để tạo trò chơi. Chúng ta sẽ sử dụng các công cụ trong mô-đun hệ thống để thoát trò chơi khi người chơi thoát.

Alien Invasion bắt đầu như một lớp gọi là `AlienInvasion`. Trong `__init__()`, hàm `pygame.init()` khởi tạo cài đặt nền Pygame cần hoạt động bình thường. Tại phần ghi chú thứ hai, chúng ta gọi `pygame.display.set_mode()` để tạo một cửa sổ hiển thị, trên đó chúng ta sẽ vẽ tất cả các yếu tố đồ họa của trò chơi. Tuple(1200,800) xác định kích cỡ của cửa sổ chơi trong đó 1200 là chiều rộng và 800 là chiều cao và ta có thể điều chỉnh giá trị này. Chúng ta gán cửa sổ hiển thị này cho thuộc tính `self.screen`, vì vậy nó sẽ có sẵn trong tất cả các phương thức trong lớp.

Đối tượng mà chúng ta gán cho `self.screen` được gọi là bìa mặt (surface). Bìa mặt trong Pygame là một phần của màn hình nơi một phần tử trò chơi có thể được hiển thị. Mỗi yếu tố trong trò chơi, con như quái vật hay một con tàu, là bìa mặt riêng

của nó. Bề mặt được trả về bởi `display.set_mode()` đại diện cho toàn bộ cửa sổ trò chơi.

Khi chúng ta kích hoạt vòng lặp hoạt ảnh của trò chơi, bề mặt này sẽ được vẽ lại trên mỗi lần đi qua vòng lặp, do đó, nó có thể được cập nhật với bất kỳ thay đổi nào được kích hoạt bởi thông tin nhập của người dùng.

Trò chơi được điều khiển bởi phương thức `run_game()`. Phương pháp này chứa vòng lặp `while` (chú thích 3) chạy liên tục.

Vòng lặp `while` chứa một vòng lặp sự kiện và mã quản lý các bản cập nhật màn hình. Sự kiện là một hành động mà người dùng thực hiện trong khi chơi trò chơi, chẳng hạn như nhấn phím hoặc di chuyển chuột.

Để làm cho chương trình phản hồi các sự kiện, chúng ta viết vòng lặp sự kiện này để lắng nghe các sự kiện và thực hiện các nhiệm vụ thích hợp tùy thuộc vào loại sự kiện xảy ra. Vòng lặp `for` tại chú thích thứ tư là một vòng lặp sự kiện.

Để truy cập các sự kiện mà Pygame phát hiện, chúng ta sẽ sử dụng `pygame.event` hàm `.get()`. Hàm này trả về danh sách các sự kiện đã diễn ra kể từ lần cuối cùng hàm này được gọi.

Bất kỳ sự kiện bàn phím hoặc chuột nào sẽ làm cho vòng lặp `for` này chạy. Bên trong vòng lặp, chúng ta sẽ viết một loạt các câu lệnh `if` để phát hiện và phản hồi các sự kiện cụ thể. Ví dụ: khi người chơi nhấp vào nút đóng cửa sổ trò chơi, sự kiện `pygame.QUIT` được phát hiện và chúng ta gọi `sys.exit()` để thoát trò chơi (chú thích 5).

Lệnh gọi tới `pygame.display.flip()` tại chú thích 6 yêu cầu Pygame hiển thị màn hình được vẽ gần đây nhất. Trong trường hợp này, nó chỉ cần vẽ một màn hình trống trên mỗi lần đi qua vòng lặp `while`, xóa màn hình cũ để chỉ màn hình mới hiển thị.

Khi chúng ta di chuyển các phần tử trò chơi xung quanh, `pygame.display.flip()` liên tục cập nhật màn hình để hiển thị vị trí mới của các phần tử trò chơi và ẩn các phần tử cũ, tạo ảo giác chuyển động mượt mà.

Ở phần cuối của trò chơi, chúng ta tạo một thể hiện của trò chơi và sau đó gọi run_game(). Chúng ta đặt run_game() trong một khối if chỉ chạy nếu tệp được gọi trực tiếp. Khi ta chạy tệp alien_invasion.py này, ta sẽ thấy một cửa sổ Pygame trống.

Thiết lập màu nền

Pygame tạo màn hình đen theo mặc định, nhưng điều đó thật nhảm chán. Ta đặt một màu nền khác. Chúng ta sẽ thực hiện việc này ở cuối phương thức __init__().

```
self.screen = pygame.display.set_mode((1200, 800))
pygame.display.set_caption("Alien Invasion")
self.bg_color = (230, 230, 230)
#self.ship = Ship(self)

def run_game(self):
    """Start the main loop for the game."""
    while True:
        # Watch for keyboard and mouse events.
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.display.quit()
                sys.exit()

        #self.screen.fill(self.settings.bg_color)
        #self.ship.blitme()
self.screen.fill(self.bg_color)
        # Make the most recently drawn screen visible.
        pygame.display.flip()
```

Màu sắc trong Pygame được xác định là màu RGB: sự pha trộn của màu đỏ, xanh lá cây và xanh lam. Mỗi giá trị màu có thể nằm trong khoảng từ 0 đến 255. Giá trị màu (255, 0, 0) là màu đỏ, (0, 255, 0) là màu xanh lá cây và (0, 0, 255) là màu xanh lam.

Ta có thể kết hợp các giá trị RGB khác nhau để tạo ra tối đa 16 triệu màu. Giá trị màu (230, 230, 230) trộn lượng đỏ, lam và lục bằng nhau, tạo ra ánh sáng màu nền xám. Chúng ta gán màu này cho self.bg_color.

Tại dòng in đậm thứ hai, chúng ta làm mờ màn hình với màu nền bằng cách sử dụng phương thức fill(), phương thức này hoạt động trên một bề mặt và chỉ nhận một đối số: một màu.

Tạo lớp Setting

Mỗi khi chúng ta giới thiệu chức năng mới vào trò chơi, chúng ta thường sẽ tạo một số cài đặt mới. Thay vì thêm cài đặt trong toàn bộ mã, ta viết một mô-đun được gọi là cài đặt có chứa một lớp được gọi là Setting để lưu trữ tất cả các giá trị này ở một nơi.

Cách tiếp cận này cho phép chúng ta chỉ làm việc với một đối tượng setting bất kỳ lúc nào chúng ta cần truy cập vào một cài đặt riêng lẻ. Điều này cũng giúp việc sửa đổi giao diện và hành vi của trò chơi trở nên dễ dàng hơn khi dự án của chúng ta phát triển: để sửa đổi trò chơi, chúng ta sẽ chỉ cần thay đổi một số giá trị trong settings.py, mà chúng ta sẽ tạo tiếp theo, thay vì tìm kiếm các cài đặt khác nhau trong suốt dự án.

Tạo một tệp mới có tên settings.py bên trong thư mục alien_invasion và thêm lớp Setting ban đầu này:

```
class Settings:
    """A class to store all settings for Alien Invasion."""

    def __init__(self):
        """Initialize the game's settings."""
        # Screen settings
        self.screen_width = 1200
        self.screen_height = 800
        self.bg_color = (230, 230, 230)
```

Để tạo một phiên bản của Setting trong dự án và sử dụng nó để truy cập cài đặt, chúng ta cần sửa đổi alien_invasion.py như sau:

```
from settings import Settings
from ship import Ship

class AlienInvasion:
    """Overall class to manage game assets and behavior."""
    def __init__(self):
        """Initialize the game, and create game resources."""
        pygame.init()
        self.settings = Settings()

        self.screen=pygame.display.set_mode((self.settings.screen_width,
                                             self.settings.screen_height))
        pygame.display.set_caption("Alien Invasion")
        #self.ship = Ship(self)

    def run_game(self):
```

```

    """Start the main loop for the game."""
    while True:
        # Watch for keyboard and mouse events.
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.display.quit()
                sys.exit()
            self.screen.fill(self.settings.bg_color)
            #self.ship.blitme()

```

Chúng ta import Setting vào chương trình chính và tạo ra một thể hiện của Setting và gán nó cho self.setting sau khi chúng ta gọi phương thức khởi tạo `__init__()`. Khi chúng ta tạo một màn hình, chúng ta sử dụng `screen_width` và thuộc tính `screen_height` của `self.settings`, sau đó chúng ta sử dụng `self.settings` để truy cập màu nền khi di chuyển màn hình tại phần in đậm thứ hai và ba.

Khi chạy `alien_invasion.py` ta sẽ chưa thấy bất kỳ thay đổi nào, bởi vì tất cả những gì chúng ta đã làm là di chuyển các cài đặt chúng ta đã sử dụng sang nơi khác. Nay giờ chúng ta đã sẵn sàng để bắt đầu thêm các yếu tố mới vào màn hình.

10.1.1.4. Tạo ảnh con tàu

Hãy thêm con tàu vào trò chơi của chúng ta. Để vẽ tàu của người chơi trên màn hình, chúng ta sẽ tải một hình ảnh và sau đó sử dụng phương thức Pygame `blit()` để vẽ hình ảnh. Khi chọn hình ảnh cho trò chơi của mình, hãy nhớ chú ý đến việc cấp phép. Cách an toàn nhất và rẻ nhất để bắt đầu là sử dụng đồ họa được cấp phép tự do mà ta có thể sử dụng và sửa đổi, từ một trang web như <https://pixabay.com/>.

Ta có thể sử dụng hầu hết mọi loại hình ảnh trong trò chơi của mình, nhưng dễ nhất là khi sử dụng một bitmap (.bmp) vì Pygame tải bitmap theo mặc định.

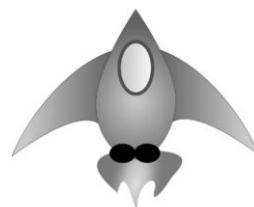
Mặc dù ta có thể yêu cầu Pygame sử dụng các loại tệp khác, nhưng một số loại tệp phụ thuộc vào một số thư viện hình ảnh nhất định phải được cài đặt trên máy tính chạy chương trình. Hầu hết các hình ảnh ta tìm thấy đều có định dạng .jpg hoặc .png, nhưng ta có thể chuyển đổi chúng thành ảnh bitmap bằng các công cụ như Photoshop, GIMP và Paint.

Đặc biệt chú ý đến màu nền trong hình ảnh ta đã chọn. Có gắng trang trí một lớp nền bằng nền trong suốt hoặc đơn mà ta có thể thay thế với bất kỳ màu nền nào bằng trình chỉnh sửa hình ảnh. Trò chơi của ta sẽ trông đẹp nhất nếu màu nền của

hình ảnh phù hợp với hình ảnh của phần tử khác. Ngoài ra, ta có thể kết hợp nền trò chơi của mình với hình ảnh nền của phần tử.

Đối với Cuộc xâm lược của quái vật, ta có thể sử dụng tệp ship.bmp, có sẵn trong tài nguyên của cuốn sách tại <https://nostarch.com/pythoncrashcourse2e/>.

Màu nền của màu phù hợp với cài đặt chúng ta đang sử dụng trong dự án này. Tạo một thư mục có tên là Hình ảnh bên trong thư mục dự án alien_invasion và lưu tệp ship.bmp trong thư mục hình ảnh.



Hình 10.1. Hình ảnh con tàu

Tạo lớp Ship

Sau khi chọn hình ảnh cho con tàu, chúng ta cần hiển thị nó trên màn hình. Để sử dụng con tàu của mình, chúng ta sẽ tạo một mô-đun ship mới có chứa lớp Ship. Lớp này sẽ quản lý hầu hết các hành vi trên con của người chơi:

```
import pygame
class Ship:
    """A class to manage the ship."""

    def __init__(self, ai_game):
        """Initialize the ship and set its starting position."""
        self.screen = ai_game.screen
        self.screen_rect = ai_game.screen.get_rect()

        # Load the ship image and get its rect.
        self.image = pygame.image.load('images/ship.bmp')
        self.rect = self.image.get_rect()

        # Start each new ship at the bottom center of the screen.
        self.rect.midbottom = self.screen_rect.midbottom

    def blitme(self):
        """Draw the ship at its current location."""
        self.screen.blit(self.image, self.rect)
```

Pygame hiệu quả vì nó cho phép ta coi tất cả các yếu tố trò chơi giống như hình chữ nhật (rects), ngay cả khi chúng không có hình dạng chính xác như hình chữ nhật. Xử lý một phần tử dưới dạng hình chữ nhật là hiệu quả vì hình chữ nhật là hình dạng hình học đơn giản. Chẳng hạn, khi Pygame cần tìm hiểu xem liệu hai yếu tố trò chơi có va chạm hay không, nó có thể thực hiện việc này nhanh hơn nếu coi mỗi đối tượng là một hình chữ nhật. Cách tiếp cận này thường hoạt động đủ tốt để không ai chơi trò chơi sẽ nhận thấy rằng chúng ta đang không làm việc với hình dạng của từng yếu tố trò chơi. Chúng ta sẽ coi con tàu và màn hình là các hình chữ nhật trong lớp này.

Chúng ta nhập mô-đun pygame trước khi xác định lớp. Phương thức `__init__()` của Ship nhận hai tham số: tham chiếu `self` và tham chiếu tới phiên bản hiện tại của lớp AlienInvasion. Điều này sẽ cung cấp cho Ship quyền truy cập vào tất cả các tài nguyên trò chơi đã được xác định trong AlienInvasion.

Đầu tiên, chúng ta gán màn hình cho một thuộc tính của Ship, vì vậy chúng ta có thể truy cập nó một cách dễ dàng trong tất cả các phương thức trong lớp này.

Tiếp theo, chúng ta truy cập thuộc tính `rect` của màn hình bằng phương thức `get_rect()` và gán nó cho `self.screen_rect`. Làm như vậy cho phép chúng ta đặt con tàu vào đúng vị trí trên màn hình.

Để tải hình ảnh, chúng ta gọi `pygame.image.load()` và cung cấp cho nó vị trí của hình ảnh con tàu. Hàm này trả về một bề mặt đại diện cho con tàu mà chúng ta gán cho `self.image`.

Khi hình ảnh được tải, chúng ta gọi `get_rect()` để truy cập thuộc tính `rect` của bề mặt tàu để sau này chúng ta có thể sử dụng thuộc tính này để đặt tàu.

Khi ta đang làm việc với một đối tượng hình chữ nhật, ta có thể sử dụng tọa độ x và y của các cạnh trên, dưới, trái và phải của hình chữ nhật, cũng như tâm, để đặt đối tượng.

Ta có thể đặt bất kỳ giá trị nào trong số này để thiết lập vị trí hiện tại của hình chữ nhật. Khi tập trung vào một yếu tố trò chơi, ta làm việc với các thuộc tính `center`, `centerx` hoặc `center` của một hình chữ nhật.

Khi ta đang làm việc ở một cạnh của màn hình, hãy làm việc với các thuộc tính trên cùng, dưới cùng, trái hoặc phải.

Ngoài ra còn có các thuộc tính kết hợp các thuộc tính này, chẳng hạn như midbottom, giữa đỉnh, giữa trái và giữa phải. Khi ta đang điều chỉnh vị trí ngang hoặc dọc của hình chữ nhật, ta chỉ có thể sử dụng các thuộc tính x và y, các thuộc tính này tọa độ x và y của góc trên cùng của nó. Những thuộc tính này giúp ta khỏi phải tính toán mà các nhà phát triển trò chơi trước đây phải làm theo cách thủ công và ta sẽ sử dụng chúng thường xuyên.

10.1.1.5. Tái cấu trúc các phương thức

Trong các dự án lớn, ta thường sẽ cấu trúc lại mã đã viết trước khi thêm mã. Cấu trúc lại cấu trúc đơn giản hóa cấu trúc của mã ta đã viết.

Trong phần này, phương thức run_game() được chia thành hai phương thức nhỏ hơn. Phương thức helper không làm việc trong một lớp, nhưng không có nghĩa không được gọi qua đối tượng. Một phương thức bắt đầu với một dấu gạch dưới là phương thức helper.

Phương thức _check_events()

Chúng ta sẽ di chuyển mã quản lý các sự kiện sang một phương thức riêng được gọi là _check_events(). Điều này sẽ đơn giản hóa run_game() và cô lập vòng lặp quản lý sự kiện. Việc cô lập vòng lặp sự kiện cho phép ta quản lý các sự kiện một cách riêng biệt từ các khía cạnh khác của trò chơi, chẳng hạn như cập nhật màn hình.

Đây là lớp AlienInvasion với phương thức _check_events() mới, điều này chỉ ảnh hưởng đến mã trong run_game():

```
def run_game(self):
    """Start the main loop for the game."""
    while True:
        self._check_events()
        # Redraw the screen during each pass through the loop.
        --snip--
def _check_events(self):
    """Respond to keypresses and mouse events."""
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            sys.exit()
```

Phương thức `_update_screen()`

Để đơn giản hóa hơn nữa chúng ta sẽ chuyển mã cập nhật màn hình sang một phương thức riêng biệt được gọi là `_update_screen()`:

```
def run_game(self):
    """Start the main loop for the game."""
    while True:
        self._check_events()
        self._update_screen()

    def _check_events(self):
        --snip--

    def _update_screen(self):
        self.screen.fill(self.settings.bg_color)
        self.ship.blitme()
        pygame.display.flip()
```

Bây giờ là phần thân của vòng lặp chính trong `run_game()` đơn giản hơn nhiều. Có thể dễ dàng nhận thấy rằng chúng ta đang tìm kiếm các sự kiện mới và cập nhật trong màn hình trên mỗi lần vượt qua vòng lặp.

Nếu ta đang xây dựng một số trò chơi, Ta có thể bắt đầu bằng cách chỉnh sửa mã của ta thành các phương thức như thế này. Nhưng nếu chúng ta chưa bao giờ giải quyết dự án như trên, có thể ta sẽ không biết cách cấu trúc lại mã của mình.

Thử nghiệm khác

Blue Sky: Tạo một cửa sổ Pygame với nền màu xanh lam.

Nhân vật trò chơi: Tìm hình ảnh bitmap của nhân vật trò chơi mà ta thích hoặc chuyển đổi hình ảnh sang bitmap. Tạo một lớp vẽ nhân vật ở giữa màn hình và khớp màu nền của hình ảnh với màu nền của màn hình hoặc ngược lại.

10.1.1.6. Điều khiển tàu

Tiếp theo, chúng ta sẽ cung cấp cho người chơi khả năng di chuyển con tàu sang trái và phải. Viết mã phản hồi khi người chơi nhấn phím mũi tên phải hoặc trái.

Trước tiên, chúng ta sẽ tập trung vào chuyển động sang phải, sau đó áp dụng mã tương tự để điều khiển chuyển động sang trái. Khi thêm mã này, ta sẽ học cách kiểm soát chuyển động của hình ảnh trên màn hình và phản hồi thông tin nhập của người dùng.

a. Phản hồi sau mỗi lần nhấn phím

Bất cứ khi nào người chơi nhấn một phím, phím đó sẽ được đăng ký trong Pygame với tên một sự kiện. Mỗi sự kiện được chọn bởi phương thức pygame.event.get(). Chúng ta cần chỉ định trong phương thức _check_events() loại sự kiện nào mà ta muốn kiểm tra. Mỗi lần nhấn phím là một sự kiện KEYDOWN.

Khi Pygame phát hiện sự kiện KEYDOWN, chúng ta cần kiểm tra xem phím được nhấn là phím nào để kích hoạt một hành động nhất định cho tàu. Ví dụ, nếu người chơi nhấn phím mũi tên phải, chúng ta sẽ tăng giá trị rect.x của tàu để di chuyển con tàu sang bên phải:

```
def _check_events(self):
    """Respond to keypresses and mouse events."""
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            sys.exit()
        elif event.type == pygame.KEYDOWN:
            if event.key == pygame.K_RIGHT:
                # Move the ship to the right.
                self.ship.rect.x += 1
```

Bên trong _check_events() thêm một khối elif vào trong vòng lặp để phản hồi khi Pygame phát hiện một sự kiện KEYDOWN . event.key kiểm tra xem phím có được nhấn hay không. Phím mũi tên phải được biểu thị bằng pygame.K_RIGHT. Nếu phím mũi tên phải được nhấn, chúng ta di chuyển tàu đến đúng bằng cách tăng giá trị của self.ship.rect.x lên 1.

b. Cho phép chuyển động liên tục

Khi người chơi giữ phím mũi tên phải, chúng ta muốn con tàu tiếp tục di chuyển sang phải cho đến khi người chơi nhả phím. Game sẽ phát hiện sự kiện pygame.KEYUP để ta biết khi nào phím mũi tên phải được nhả ra; vì vậy ta sẽ sử dụng các sự kiện KEYDOWN và KEYUP cùng với một trạng thái gọi là moving_right để thực hiện chuyển động liên tục.

Khi cờ moving_right là False, con tàu sẽ bất động. Khi nào người chơi nhấn phím mũi tên phải, chúng ta sẽ đặt trạng thái thành True và khi người chơi nhả phím ta sẽ đặt lại cờ thành False.

Lớp Ship kiểm soát tất cả các thuộc tính của con tàu, vì vậy chúng ta sẽ cung cấp cho nó một thuộc tính gọi là moving_right và một phương thức update() để kiểm tra trạng thái của move_right. Phương thức update() sẽ thay đổi vị trí của con tàu nếu cờ được đặt thành True. Chúng tôi sẽ gọi phương thức này một lần cho mỗi lần chuyển qua vòng lặp while để cập nhật vị trí của tàu.

Đây là những thay đổi đối với con tàu:

```
class Ship:  
    """A class to manage the ship."""  
    def __init__(self, ai_game):  
        --snip--  
        self.rect.midbottom = self.screen_rect.midbottom  
# Movement flag  
self.moving_right = False  
    def update(self):  
        if self.moving_right:  
            self.rect.x += 1  
    def blitme(self):  
        --snip--
```

Chúng ta thêm thuộc tính self.moving_right trong phương thức __init__() và đặt nó thành False. Sau đó, thêm update(), chuyển con tàu sang bên phải nếu trạng thái là True. Phương thức update() sẽ được gọi thông qua một phiên bản của Ship, vì vậy nó không được coi là một phương thức trợ giúp.

Bây giờ chúng ta cần sửa đổi _check_events() để move_right được đặt thành True khi phím mũi tên phải được nhấn và False khi thả phím:

```
def _check_events(self):  
    """Respond to keypresses and mouse events."""  
    for event in pygame.event.get():  
        --snip--  
        elif event.type == pygame.KEYDOWN:  
            if event.key == pygame.K_RIGHT:  
                self.ship.moving_right = True  
    elif event.type == pygame.KEYUP:  
        if event.key == pygame.K_RIGHT:  
            self.ship.moving_right = False
```

Tiếp theo, chúng ta sửa đổi vòng lặp while trong run_game() để nó cập nhật con tàu khi mỗi lần đi qua vòng lặp:

```
def run_game(self):
```

```

"""Start the main loop for the game."""
    While True:
        self._check_events()
        self.ship.update()
        self._update_screen()

```

Vị trí của con tàu sẽ được cập nhật sau khi chúng ta kiểm tra bàn phím sự kiện và trước lúc cập nhật màn hình. Điều này cho phép vị trí của con tàu được cập nhật để đáp ứng với đầu vào của người chơi và đảm bảo vị trí cập nhật sẽ được sử dụng khi vẽ con tàu lên màn hình. Khi ta chạy alien_invasion.py và giữ phím mũi tên bên phải, tàu sẽ di chuyển liên tục sang phải cho đến khi ta nhả phím.

c. Di chuyển cả trái và phải

Bây giờ con tàu có thể di chuyển liên tục sang phải, thêm chuyển động vào bên trái. Một lần nữa, chúng ta sẽ sửa đổi lớp Ship và _check_events(). Dưới đây là những thay đổi liên quan đối với __init__() và update() trong tàu:

```

def __init__(self, ai_game):
    --snip--
    # Movement flags
    self.moving_right = False
    self.moving_left = False
def update(self):
    """Update the ship's position based on movement flags."""
    if self.moving_right:
        self.rect.x += 1
    if self.moving_left:
        self.rect.x -= 1

```

Trong __init__(), thêm self.moving_left. Trong update(), chúng ta sử dụng các khối if thay vì elif để cho phép giá trị vị trí của tàu là tăng và sau đó giảm khi nhấn giữ cả hai phím mũi tên. Cái này kết quả là tàu đứng yên. Nếu chúng ta sử dụng elif cho chuyển động sang trái, phím mũi tên phải sẽ luôn được ưu tiên. Làm theo cách này làm cho chuyển động chính xác hơn khi chuyển từ phải sang trái , người chơi có thể nhấn giữ cả hai phím cùng một lúc.

Chúng ta phải thực hiện hai điều chỉnh đối với _check_events():

```

def _check_events(self):
    """Respond to keypresses and mouse events."""
    for event in pygame.event.get():
        --snip--
        elif event.type == pygame.KEYDOWN:

```

```

        if event.key == pygame.K_RIGHT:
    self.ship.moving_right = True
        elif event.key == pygame.K_LEFT:
            self.ship.moving_left = True
    elif event.type == pygame.KEYUP:
        if event.key == pygame.K_RIGHT:
            self.ship.moving_right = False
        elif event.key == pygame.K_LEFT:
            self.ship.moving_left = False

```

Khi ta chạy alien_invasion.py ngay bây giờ, ta có thể di chuyển con tàu liên tục sang phải và trái. Nếu giữ cả hai phím, con tàu sẽ ngừng di chuyển. Tiếp theo, chúng ta sẽ chỉnh sửa thêm chuyển động của con tàu. Hãy điều chỉnh tốc độ và giới hạn khoảng cách con tàu có thể di chuyển để nó không thể biến mất khỏi hai bên của màn hình.

d. Điều chỉnh tốc độ của tàu

Hiện tại, con tàu di chuyển một pixel mỗi chu kỳ qua vòng lặp while, nhưng chúng ta có thể kiểm tốc độ của tàu bằng cách thêm thuộc tính ship_speed vào lớp Setting. Chúng ta sẽ sử dụng thuộc tính này để xác định quãng đường di chuyển tàu sau mỗi lần đi qua vòng lặp. Đây là thuộc tính mới trong settings.py:

```

class Settings:
    """A class to store all settings for Alien Invasion."""
    def __init__(self):
        --snip--
        # Ship settings
        self.ship_speed = 1.5

```

Chúng ta đặt giá trị ban đầu của ship_speed thành 1,5. Khi con tàu di chuyển, vị trí của nó được điều chỉnh 1,5 pixel thay vì 1 pixel sau mỗi lần vượt qua vòng lặp.

Chúng ta sử dụng các giá trị thập phân cho cài đặt tốc độ để cung cấp tốc độ con tàu tốt hơn khi tăng nhịp độ của trò chơi sau này. Tuy nhiên, các thuộc tính vị trí chẵng hạn như x chỉ lưu trữ các giá trị nguyên, vì vậy chúng ta cần thực hiện một số sửa đổi đối với lớp Ship:

```

class Ship:
    """A class to manage the ship."""
    def __init__(self, ai_game):
        """Initialize the ship and set its starting position."""
        self.screen = ai_game.screen
        self.settings = ai_game.settings

```

```

--snip--
    # Start each new ship at the bottom center of the screen.
--snip--
    # Store a decimal value for the ship's horizontal position.
    self.x = float(self.rect.x)
    # Movement flags
    self.moving_right = False
    self.moving_left = False
def update(self):
    """Update the ship's position based on movement flags."""
    # Update the ship's x value, not the rect.
    if self.moving_right:
        self.x += self.settings.ship_speed
    if self.moving_left:
        self.x -= self.settings.ship_speed
    # Update rect object from self.x.
    self.rect.x = self.x
def blitme(self):
--snip--

```

Chúng ta tạo một thuộc tính `self.setting` cho `Ship`, vì vậy ta có thể sử dụng nó trong `update()`. Bởi vì chúng ta đang điều chỉnh vị trí của con tàu theo từng phần nhỏ của pixel nên cần gán vị trí cho một biến có thể lưu giá trị thập phân. Bạn có thể sử dụng giá trị thập phân để đặt một thuộc tính của `rect`, nhưng `rect` sẽ chỉ giữ nguyên phần nguyên của giá trị đó. Để theo dõi vị trí của con tàu một cách chính xác, chúng ta xác định một thuộc tính `self.x` có thể chứa các giá trị thập phân. Sử dụng hàm `float()` để chuyển đổi giá trị của `self.rect.x` thành số thập phân và gán giá trị này cho `self.x`.

Bây giờ muốn thay đổi vị trí của con tàu trong `update()`, giá trị của `self.x` được điều chỉnh bởi số lượng được lưu trữ trong `settings.ship_speed`. Sau khi `self.x` đã được cập nhật, chúng ta sử dụng giá trị mới để cập nhật `self.rect.x`, điều khiển vị trí của con tàu. Chỉ phần nguyên của `self.x` sẽ được lưu trữ trong `self.rect.x`, điều đó tốt cho việc hiển thị con tàu.

Bây giờ chúng ta có thể thay đổi giá trị của `ship_speed` với bất kỳ giá trị nào lớn hơn từ đó làm cho con tàu chuyển động nhanh hơn. Điều này giúp tàu phản hồi nhanh để bắn hạ quái vật, và nó sẽ cho phép chúng ta thay đổi nhịp độ của trò chơi khi người chơi tăng level.

e. Giới hạn phạm vi di chuyển của tàu

Tại thời điểm này, con tàu sẽ biến mất khỏi một trong hai cạnh của màn hình do ta giữ một phím mũi tên khá lâu. Hãy sửa lỗi này để con tàu dừng lại khi nó chạm đến mép màn hình bằng cách sửa đổi phương thức update() trong Ship:

```
def update(self):
    """Update the ship's position based on movement flags."""
    # Update the ship's x value, not the rect.
    if self.moving_right and self.rect.right < self.screen_rect.right:
        self.x += self.settings.ship_speed
    if self.moving_left and self.rect.left > 0:
        self.x -= self.settings.ship_speed
    # Update rect object from self.x.
    self.rect.x = self.x
```

Đoạn code này kiểm tra vị trí của tàu trước khi thay đổi giá trị của self.x. self.rect.right trả về tọa độ x của cạnh bên phải của con tàu. Nếu giá trị này nhỏ hơn giá trị self.screen_rect.right, con tàu chưa đến mép bên phải của màn hình. Tương tự cho cạnh trái: nếu giá trị của cạnh trái của rect lớn hơn 0, con tàu chưa đến mép trái của màn hình. Điều này đảm bảo tàu nằm trong các giới hạn trước khi điều chỉnh giá trị của self.x.

Khi chạy alien_invasion.py ngay bây giờ, con tàu sẽ ngừng di chuyển tại một trong hai cạnh của màn hình. Tất cả những gì chúng ta đã làm là thêm lệnh điều kiện trong câu lệnh if, nhưng có cảm giác như con tàu va vào tường hoặc một lực trường ở hai cạnh của màn hình!

f. Cấu trúc lại _check_events()

Phương thức _check_events() sẽ tăng độ dài code khi chúng ta tiếp tục phát triển trò chơi, vì vậy hãy chia _check_events() thành hai phương thức khác: một phương thức xử lý các sự kiện KEYDOWN và một phương thức khác xử lý các sự kiện KEYUP:

```
def _check_events(self):
    """Respond to keypresses and mouse events."""
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            sys.exit()
        elif event.type == pygame.KEYDOWN:
            self._check_keydown_events(event)
        elif event.type == pygame.KEYUP:
            self._check_keyup_events(event)
```

```

def _check_keydown_events(self, event):
    """Respond to keypresses."""
    if event.key == pygame.K_RIGHT:
        self.ship.moving_right = True
    elif event.key == pygame.K_LEFT:
        self.ship.moving_left = True

def _check_keyup_events(self, event):
    """Respond to key releases."""
    if event.key == pygame.K_RIGHT:
        self.ship.moving_right = False
    elif event.key == pygame.K_LEFT:
        self.ship.moving_left = False

```

Tạo hai phương thức trợ giúp mới: `_check_keydown_events()` và `_check_keyup_events()`. Mỗi thứ cần một tham số `self` và một tham số `event`. Các phần thân của hai phương thức này được sao chép từ `_check_events()` và chúng ta đã đã thay thế mã cũ bằng các lệnh gọi đến các phương thức mới.

`_Check_events()` bây giờ đơn giản hơn, điều này sẽ làm cho nó dễ dàng hơn để phát triển các phản hồi hơn nữa đối với đầu vào của người chơi.

g. Nhấn Q để thoát

Bây giờ chúng ta đang phản hồi các lần nhấn phím một cách hiệu quả, chúng ta có thể thêm một cách để thoát khỏi trò chơi. Thật là tệ nhạt khi nhập vào X ở đầu trò chơi cửa sổ để kết thúc trò chơi, vì vậy ta sẽ thêm một phím tắt để kết thúc trò chơi khi người chơi nhấn Q:

```

def _check_keydown_events(self, event):
    --snip--
    elif event.key == pygame.K_LEFT:
        self.ship.moving_left = True
    elif event.key == pygame.K_q:
        sys.exit()

```

Trong `_check_keydown_events()`, chúng ta thêm một khối `elif` mới để kết thúc trò chơi khi người chơi nhấn Q.

h. Chạy trò chơi ở chế độ toàn màn hình

Pygame có chế độ toàn màn hình mà ta có thể thích hơn là chạy trò chơi trong một cửa sổ thông thường. Một số trò chơi trông đẹp hơn khi ở chế độ toàn màn hình.

Để chạy trò chơi ở chế độ toàn màn hình, hãy thực hiện các thay đổi sau trong `__init__()`:

```
def __init__(self):
    """Initialize the game, and create game resources."""
    pygame.init()
    self.settings = Settings()

    self.screen=pygame.display.set_mode((0,0),pygame.FULLSCREEN)
    self.settings.screen_width = self.screen.get_rect().width
    self.settings.screen_height=self.screen.get_rect().height
    pygame.display.set_caption("Alien Invasion")
```

Khi cài đặt toàn màn hình, chúng ta chuyển kích thước(0, 0) và tham số `pygame.FULLSCREEN`. Điều này yêu cầu Pygame tìm ra kích thước cửa sổ sẽ lấp đầy màn hình. Bởi vì ta không biết chiều rộng và chiều cao của màn hình trước nên chúng ta cập nhật các cài đặt này sau khi màn hình được tạo.

Lưu ý : Đảm bảo rằng ta có thể thoát bằng cách nhấn Q trước khi chạy trò chơi ở chế độ toàn màn hình; Pygame không cung cấp cách mặc định để thoát trò chơi khi ở chế độ toàn màn hình.

Trong phần tiếp theo, chúng ta sẽ thêm khả năng bắn đạn, bằng cách thêm một tệp mới có tên là `bullet.py` và thực hiện một số sửa đổi đối với các tệp đang sử dụng. Ngay bây giờ, chúng ta có ba tệp chứa một số lớp và phương thức. Để rõ hơn về cách tổ chức dự án, chúng ta hãy xem xét từng tệp này trước khi thêm các chức năng khác.

Alien_invasion.py

Tệp chính `alien_invasion.py`, chứa lớp `AlienInvasion`. Lớp này tạo ra một số thuộc tính quan trọng được sử dụng trong suốt trò chơi: chỉ định cho `setting`, giao diện hiển thị chính cho màn hình, và một cá thể tàu cũng được tạo trong tệp này. Vòng lặp chính của trò chơi, `while loop`, cũng được lưu trữ trong mô-đun này. Vòng lặp `while` cũng gọi các method `_check_events()`, `ship.update()` và `_update_screen()`.

Phương thức `_check_events()` phát hiện các sự kiện có liên quan, chẳng hạn như các lần nhấn và thả phím, đồng thời xử lý từng loại sự kiện này thông qua các phương thức `_check_keydown_events()` và `_check_keyup_events()`. Hiện tại, các phương thức này quản lý chuyển động của con tàu. Lớp `AlienInvasion` cũng chứa `_update_screen()`, sẽ chỉnh lại màn hình sau mỗi lần đi qua vòng lặp chính.

Tệp alie_invasion.py là tệp duy nhất ta cần chạy khi ta muốn chơi Alien Invasion.

settings.py

Tệp settings.py chứa lớp Setting. Lớp này chỉ có __init__() nhằm khởi tạo các thuộc tính kiểm soát giao diện của trò chơi và tốc độ của con tàu.

ship.py

Tệp ship.py chứa lớp Ship. Lớp Ship có phương thức __init__(), một phương thức update() để quản lý vị trí của con tàu và một blitme() để vẽ con tàu ra màn hình. Hình ảnh con tàu được lưu trữ trong ship.bmp, nằm trong thư mục hình ảnh.

Ta hãy thử

Tài liệu Pygame: Bây giờ chúng ta đã tiến xa trong trò chơi mà ta muốn xem một số tài liệu Pygame. Trang chủ Pygame có tại <https://www.pygame.org/> và trang chủ tài liệu tại <https://www.pygame.org/docs/>. Bây giờ chỉ cần đọc lướt tài liệu. Bạn sẽ không cần nó để hoàn thành dự án này, nhưng nó sẽ hữu ích nếu ta muốn sửa đổi Ailen Invasion hoặc tạo trò chơi của riêng ta .

Tên lửa: Tạo trò chơi bắt đầu bằng tên lửa ở giữa màn. Cho phép người chơi di chuyển tên lửa lên, xuống, sang trái hoặc sang phải bằng cách sử dụng bốn phím mũi tên. Đảm bảo rằng tên lửa không bao giờ di chuyển vượt ra ngoài bất kỳ cạnh nào của màn.

Phím: Tạo tệp Pygame tạo màn hình trống. Trong sự kiện lặp lại, in thuộc tính event.key bắt cứ khi nào phát hiện thấy sự kiện pygame.KEYDOWN. Chạy chương trình và nhấn các phím khác nhau để xem Pygame phản hồi như thế nào.

10.1.1.7. Bắt đạn lửa

Bây giờ, hãy thêm khả năng bắn đạn cho con tàu. Chúng ta sẽ viết mã để bắn một viên đạn, được biểu thị bằng một hình chữ nhật nhỏ, xuất hiện khi người chơi nhấn phím cách. Các viên đạn sau đó sẽ di chuyển thẳng lên màn hình cho đến khi chúng biến mất khi đến phía trên cùng của màn hình.

a. Thêm cài đặt đạn

Ở cuối phương thức __init__(), chúng ta sẽ cập nhật settings.py :

```

def __init__(self):
    --snip--
    # Bullet settings
    self.bullet_speed = 1.0
    self.bullet_width = 3
    self.bullet_height = 15
    self.bullet_color = (60, 60, 60)

```

Các cài đặt này tạo ra các viên đạn màu xám đậm với chiều rộng 3 pixel và chiều cao 15 pixel. Đạn sẽ di chuyển chậm hơn một chút so với con tàu.

b. Tạo lớp Bullet

Bây giờ hãy tạo thêm một tệp bullet.py để lưu trữ lớp Bullet. Đây là phần đầu tiên của bullet.py:

```

import pygame
from pygame.sprite import Sprite
class Bullet(Sprite):
    """A class to manage bullets fired from the ship"""
    def __init__(self, ai_game):
        super().__init__()
        self.screen = ai_game.screen
        self.settings = ai_game.settings
        self.color = self.settings.bullet_color
        self.rect=pygame.Rect(0, 0, self.settings.bullet_width,
                             self.settings.bullet_height)
        self.rect.midtop = ai_game.ship.rect.midtop
        # Store the bullet's position as a decimal value.
        self.y = float(self.rect.y)

```

Lớp Bullet kế thừa từ Sprite, cài đặt từ pygame.sprite. Khi ta sử dụng sprites, ta có thể nhóm các phần tử liên quan trong trò chơi và hành động trên tất cả các yếu tố được nhóm cùng một lúc. Để tạo viên đạn, __init__() cần phiên bản hiện tại của AlienInvasion và chúng ta gọi super() để kế thừa đúng cách từ Sprite. Chúng ta cũng đặt các thuộc tính self cho screen và settings và cho màu của viên đạn.

Trước tiên chúng ta tạo thuộc tính vị trí của viên đạn. Viên đạn thì không thể tạo dựa trên một hình ảnh, vì vậy chúng ta phải tạo một vị trí trực tiếp bằng cách sử dụng lớp pygame.Rect(). Lớp này yêu cầu tọa độ x và y của góc trên cùng bên trái của hình chữ nhật, chiều rộng và chiều cao của hình chữ nhật. Khởi tạo vị trí tại (0, 0), di chuyển nó đến vị trí chính xác trong dòng tiếp theo, vì vị trí viên đạn phụ thuộc

vào vị trí của tàu. Chúng ta nhận được chiều rộng và chiều cao của viên đạn từ các giá trị được lưu trữ trong self.settings.

Sau đó chúng ta đặt thuộc tính midtop của viên đạn để khớp với thuộc tính ở giữa của tàu. Điều này sẽ làm cho viên đạn trồi lên khỏi đầu tàu, khiến nó giống như viên đạn được bắn ra từ con tàu. Chúng ta lưu trữ một giá trị thập phân cho tọa độ y của viên đạn để chúng ta có thể điều chỉnh tốc độ của viên đạn (2).

Đây là phần thứ hai của bullet.py, update() và draw_bullet():

```
def update(self):
    """Move the bullet up the screen."""
    # Update the decimal position of the bullet.
    self.y -= self.settings.bullet_speed
    # Update the rect position.
    self.rect.y = self.y
def draw_bullet(self):
    """Draw the bullet to the screen."""
    pygame.draw.rect(self.screen, self.color, self.rect)
```

Phương thức update() quản lý vị trí của viên đạn. Khi một viên đạn là được kích hoạt, nó di chuyển lên trên màn hình, tương ứng với một tọa độ y giảm dần. Để cập nhật vị trí, ta giảm độ lớn được lưu trữ trong setting.bullet_speed từ self.y .

Sau đó, chúng ta sử dụng giá trị self.y để đặt giá trị của self.rect.y. Cài đặt bullet_speed cho phép ta tăng tốc độ của đạn khi trò chơi tiến. Một lần viên đạn được bắn ra, giá trị tọa độ x của nó không bao giờ thay đổi, vì vậy nó sẽ di chuyển thẳng đứng trên một đường thẳng ngay cả khi tàu chuyển động.

Khi chúng ta muốn vẽ một viên đạn, chúng ta gọi draw_bullet().

c. Lưu trữ những viên đạn trong một nhóm

Bây giờ chúng ta có một lớp Bullet và các cài đặt cần thiết được xác định, chúng ta có thể viết mã để bắn một viên đạn mỗi khi người chơi nhấn phím cách. Có thể tạo một nhóm trong AlienInvasion để lưu trữ tất cả đạn thật nhằm quản lý những viên đạn đã được bắn. Nhóm này được thể hiện trong lớp pygame.sprite.Group, hoạt động giống như một danh sách với một số chức năng hữu ích khi xây dựng trò chơi. Chúng ta cũng sử dụng nhóm này để vẽ các viên đạn lên màn hình khi mỗi lần đi qua vòng lặp chính và để cập nhật vị trí của từng viên đạn.

Tạo nhóm trong __init__():

```

def __init__(self):
    --snip--
    self.ship = Ship(self)
    self.bullets = pygame.sprite.Group()

```

Sau đó cần cập nhật vị trí của những viên đạn mỗi khi qua vòng lặp while chính:

```

def run_game(self):
    """Start the main loop for the game."""
    while True:
        self._check_events()
        self.ship.update()
        self.bullets.update()
        self._update_screen()

```

Khi ta gọi update() trên một nhóm , nhóm đó sẽ tự động gọi update() cho mỗi sprite trong nhóm. Dòng self.bullets.update() gọi bullet.update() cho mỗi viên đạn chúng ta đặt trong các viên đạn nhóm.

c. Bắn đạn

Trong AlienInvasion, chúng ta cần sửa đổi _check_keydown_events() để bắn một viên đạn khi người chơi nhấn phím cách. Không cần thay đổi _checkkeyup_events() vì không có gì xảy ra khi phím cách được nhả. Chúng ta cũng cần sửa đổi _update_screen() để đảm bảo mỗi viên đạn được vẽ vào màn hình trước khi chúng ta gọi flip().

Cần làm một số việc khi bắn một viên đạn, vì vậy hãy viết một phương thức mới _fire_bullet() để xử lý công việc này:

```

--snip--
from ship import Ship
from bullet import Bullet

class AlienInvasion:
    --snip--
    def _check_keydown_events(self, event):
        --snip--
        elif event.key == pygame.K_q:
            sys.exit()
        elif event.key == pygame.K_SPACE:
            self._fire_bullet()
    def _check_keyup_events(self, event):
        --snip--

```

```

def _fire_bullet(self):
    """Create a new bullet and add it to the bullets group."""
    new_bullet = Bullet(self)
    self.bullets.add(new_bullet)

def _update_screen(self):
    """Update images on the screen, and flip to the new screen."""
    self.screen.fill(self.settings.bg_color)
    self.ship.blitme()
    for bullet in self.bullets.sprites():
        bullet.draw_bullet()
    pygame.display.flip()
--snip--

```

Đầu tiên, chúng ta import Bullet. Sau đó, gọi _fire_bullet() khi nhấn phím cách. Trong _fire_bullet(), chúng ta tạo một thẻ hiện của Bullet và gọi nó new_bullet . Sau đó, thêm nó vào các nhóm viên đạn bằng cách sử dụng phương thức add(). Phương thức add() tương tự như append(), nhưng nó là một phương thức được viết riêng cho các nhóm Pygame.

Phương thức bullet.sprites() trả về danh sách tất cả các sprites trong nhóm đạn. Để vẽ tất cả các viên đạn đã bắn ra màn hình, chúng ta dùng vòng for lặp các viên đạn qua các sprites và gọi draw_bullet().

Khi chạy alien_invasion.py, ta sẽ có thể di chuyển con tàu phái và trái, và bắn bao nhiêu viên đạn tùy thích. Những viên đạn đi lên màn hình và biến mất khi chúng lên đến đỉnh, như trong hình dưới. Ta có thể thay đổi kích thước, màu sắc và tốc độ của các viên đạn trong settings.py.



Hình 10.2. Con tàu bắn đạn

d. Xóa viên đạn cũ

Hiện tại, những viên đạn biến mất khi chúng lên đến đỉnh, bởi vì Pygame không thể vẽ chúng lên phía trên đầu màn hình. Nhưng viên đạn thực sự vẫn tiếp tục tồn tại; giá trị tọa độ y của chúng ngày càng tăng. Đây là một vấn đề, bởi vì chúng tiếp tục sử dụng bộ nhớ.

Chúng ta cần loại bỏ những viên đạn cũ này, nếu không trò chơi sẽ chậm lại và làm quá nhiều công việc không cần thiết. Để làm điều này, ta cần phát hiện khi giá trị bottom của vị trí (rect) viên đạn có giá trị là 0, điều này cho biết viên đạn đã vượt ra khỏi đầu màn hình:

```
def run_game(self):
    """Start the main loop for the game."""
    while True:
        self._check_events()
        self.ship.update()
        self.bullets.update()
        # Get rid of bullets that have disappeared.
        for bullet in self.bullets.copy():
            if bullet.rect.bottom <= 0:
                self.bullets.remove(bullet)
                print(len(self.bullets))
        self._update_screen()
```

Khi ta sử dụng vòng lặp for với một danh sách (hoặc một nhóm trong Pygame), Python kỳ vọng rằng danh sách sẽ giữ nguyên độ dài miễn là vòng lặp đang chạy.

Vì chúng ta không thể xóa các mục khỏi danh sách hoặc nhóm viên đạn trong vòng lặp for nên phải lặp lại một bản sao của nhóm. Chúng tôi sử dụng phương thức copy() để thiết lập vòng lặp for, để sửa đổi các viên đạn bên trong vòng lặp. Chúng ta kiểm tra từng viên đạn để xem nó có biến mất khỏi đầu màn hình hay không. Nếu có, ta xóa viên đạn.

Sau đó, chúng ta chèn một lệnh gọi print() tới hiển thị có bao nhiêu viên đạn hiện đang tồn tại trong trò chơi và xác minh rằng chúng bị xóa khi chúng lên đến đầu màn hình.

Nếu đoạn mã trên hoạt động chính xác, chúng ta có thể thấy rõ trong khi bắn đạn và thấy rằng số lượng đạn giảm xuống 0 sau mỗi loạt đạn đã xóa khi chạm đến đầu màn hình. Sau khi ta bắt đầu chơi và xác minh rằng viên đạn đang được xóa

đúng cách, hãy xóa lệnh gọi print(). Nếu như ta để nó vào, trò chơi sẽ chậm lại đáng kể vì nó cần nhiều hơn thời gian để ghi đầu ra vào máy ta hơn là thời gian để vẽ đồ họa vào cửa sổ trò chơi.

e. Giới hạn số lượng đạn

Nhiều trò chơi giới hạn số lượng đạn mà người chơi có thể có trên màn hình cùng một lúc; làm như vậy sẽ khuyến khích người chơi bắn chính xác.

Đầu tiên, hãy lưu trữ số lượng viên đạn được phép sử dụng trong settings.py:

```
# Bullet settings  
--snip--  
self.bullet_color = (60, 60, 60)  
self.bullets_allowed = 3
```

Điều này giới hạn người chơi chỉ có ba viên đạn cùng một lúc. Chúng ta sẽ sử dụng cài đặt này trong AlienInvasion để kiểm tra xem có bao nhiêu viên đạn tồn tại trước khi tạo viên đạn mới trong _fire_bullet():

```
def _fire_bullet(self):  
    """Create a new bullet and add it to the bullets group."""  
    if len(self.bullets) < self.settings.bullets_allowed:  
        new_bullet = Bullet(self)  
        self.bullets.add(new_bullet)
```

Khi người chơi nhấn phím cách, chúng ta sẽ kiểm tra độ dài của đạn. Nếu độ dài (self.bullets) nhỏ hơn 3 tạo một viên đạn mới. Nhưng nếu ba đạn đang hoạt động, không có gì xảy ra khi nhấn phím cách.

f. Tạo phương thức _update_bullets()

Muốn giữ cho lớp AlienInvasion được tổ chức hợp lý, Chúng ta sẽ tạo một phương thức mới có tên _update_bullets() và thêm nó ngay trước _update_screen():

```
def _update_bullets(self):  
    """Update position of bullets and get rid of old bullets."""  
    # Update bullet positions.  
    self.bullets.update()  
    # Get rid of bullets that have disappeared.  
    for bullet in self.bullets.copy():  
        if bullet.rect.bottom <= 0:  
            self.bullets.remove(bullet)
```

_update_bullets() được cắt và dán từ run_game. Vòng lặp while trong run_game() trông lại đơn giản hơn :

```

while True:
    self._check_events()
    self.ship.update()
    self._update_bullets()
    self._update_screen()

```

Giờ đây, vòng lặp chính while chỉ chứa mã tối thiểu, vì vậy chúng ta có thể nhanh chóng đọc tên phương thức và hiểu những gì đang xảy ra trong trò chơi. Vòng lặp chính kiểm tra đầu vào của người chơi, sau đó cập nhật vị trí của tàu và bất kỳ viên đạn nào đã được bắn. Sau đó, ta sử dụng các vị trí đã cập nhật để vẽ một màn hình mới.

Chạy alien_invasion.py một lần nữa và đảm bảo rằng ta vẫn có thể kích hoạt đạn không có lỗi.

Hay thử

Sideways Shooter: *Viết một trò chơi đặt một con tàu ở phía bên trái của màn hình và cho phép người chơi di chuyển con tàu lên và xuống. Cho phép con tàu bắn một viên đạn di chuyển ngay trên màn hình khi người chơi nhấn phím cách. Đảm bảo rằng viên đạn sẽ bị xóa khi chúng biến mất khỏi màn hình.*

Trong chương này, ta đã học cách lập kế hoạch cho một trò chơi và học những kiến thức cơ bản cấu trúc một trò chơi được viết bằng Pygame. Ta còn học cách đặt màu nền và lưu trữ các cài đặt trong một lớp riêng biệt, nơi có thể điều chỉnh chúng dễ dàng hơn. Ta đã thấy cách vẽ hình ảnh lên màn hình và cấp cho người chơi quyền kiểm soát sự chuyển động của các thành phần trong trò chơi. Tạo các phần tử di chuyển, giống như viên đạn bay lên màn hình và các đối tượng đã xóa không còn cần thiết. Ta cũng đã học cách cấu trúc lại mã trong một dự án một cách thường xuyên để tạo điều kiện cho sự phát triển liên tục.

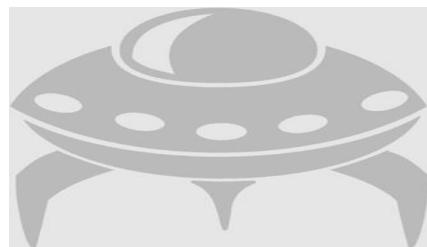
Trong phần tiếp theo, chúng ta sẽ thêm quái vật vào Alien Invasion-Cuộc xâm lược của quái vật. Đến cuối chương này, ta sẽ có thể bắn hạ quái vật trước khi chúng tiếp cận tàu của ta!

10.1.2. Tạo ra quái vật

10.1.2.1. Tạo ra con quái vật đầu tiên

Đặt một quái vật trên màn hình giống như đặt một con tàu trên màn hình. Mỗi hành động của quái vật được kiểm soát bởi một lớp class được gọi là Alien, tương tự như class Ship.

Chúng ta sẽ tiếp tục sử dụng hình ảnh bitmap để đơn giản hóa. Ta có thể tìm hình ảnh riêng cho một quái vật hoặc sử dụng hình ảnh được hiển thị trong Hình 13-1, hiện có trong tài nguyên của sách tại <https://nostarch.com/pythoncrashcourse2e/>. Hình ảnh này có nền màu xám, phù hợp với nền của màn hình màu sắc. Đảm bảo ta lưu tệp hình ảnh ta chọn trong thư mục hình ảnh.



Hình 10.3. Quái vật

a. Tạo lớp Alien:

Bây giờ chúng ta sẽ tạo lớp Alien và lưu nó tại Alien.py:

```
import pygame
from pygame.sprite import Sprite
class Alien(Sprite):
    """A class to represent a single alien in the fleet."""
    def __init__(self, ai_game):
        """Initialize the alien and set its starting position."""
        super().__init__()
        self.screen = ai_game.screen

        # Load the alien image and set its rect attribute.
        self.image = pygame.image.load('images/alien.bmp')
        self.rect = self.image.get_rect()

        # Start each new alien near the top left of the screen.
        self.rect.x = self.rect.width
        self.rect.y = self.rect.height

        # Store the alien's exact horizontal position.
```

```
self.x = float(self.rect.x)
```

Hầu hết lớp này giống như lớp Ship ngoại trừ vị trí của quái vật trên màn hình. Ban đầu, chúng ta đặt từng quái vật gần góc trên bên trái của màn hình; thêm một khoảng trống ở bên trái bằng với chiều rộng của quái vật và một khoảng trống phía trên nó bằng chiều cao để có thể dễ dàng nhìn thấy. Chúng ta chủ yếu là quan tâm đến tốc độ ngang của quái vật, vì vậy ta sẽ theo dõi vị trí phương ngang của mỗi quái vật một cách chính xác.

Lớp Alien này không cần phương thức để vẽ nó ra màn hình; thay vào đó, chúng ta sẽ sử dụng phương thức nhóm Pygame tự động rút ra tất cả các phần tử của một nhóm lên màn hình.

b. Tạo một bản sao của quái vật

Tạo một bản Alien để có thể nhìn thấy quái vật đầu tiên trên màn hình. Ta sẽ hiện mã này ở cuối phương thức `__init__()` trong `AlienInvasion`. Sau cùng, ta sẽ tạo ra hạm đội quái vật, sẽ có khá nhiều công việc, vì vậy cần tạo một phương thức trợ giúp mới có tên `_create_fleet()`.

Thứ tự của các phương thức trong một lớp không quan trọng, miễn là có một số nhất quán về cách chúng được đặt. Ta sẽ đặt `_create_fleet()` ngay trước Phương thức `_update_screen()`.

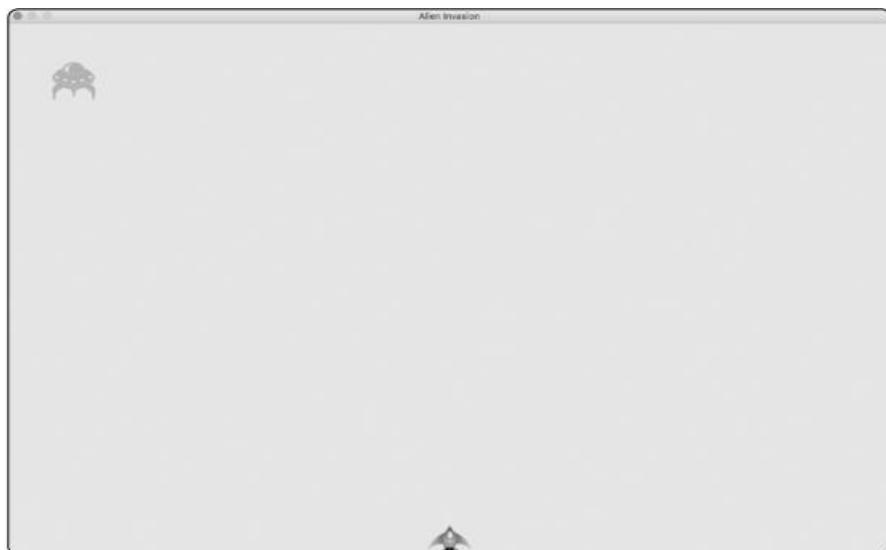
Đầu tiên, chúng ta sẽ import class Alien:

```
--snip--  
from bullet import Bullet  
from alien import Alien  
  
def __init__(self):  
    --snip--  
    self.ship = Ship(self)  
    self.bullets = pygame.sprite.Group()  
    self.aliens = pygame.sprite.Group()  
    self._create_fleet()  
  
def _create_fleet(self):  
    """Create the fleet of aliens."""  
    # Make an alien.  
    alien = Alien(self)  
    self.aliens.add(alien)
```

Trong phương thức này, chúng ta đang tạo một phiên bản Alien, sau đó thêm nó cho nhóm. Quái vật sẽ được đặt ở chế độ mặc định tại phía trên bên trái của màn hình. Để làm cho quái vật xuất hiện, chúng ta cần gọi phương thức draw() của nhóm trong _update_screen():

```
def _update_screen(self):  
    --snip--  
    for bullet in self.bullets.sprites():  
        bullet.draw_bullet()  
self.aliens.draw(self.screen)  
    pygame.display.flip()
```

Khi ta gọi draw() trên một nhóm, Pygame sẽ vẽ từng phần tử trong nhóm tại vị trí được xác định bởi thuộc tính vị trí của nó. Phương thức draw() yêu cầu một đối số: một bề mặt để vẽ các phần tử từ nhóm. Hình 13-2 cho thấy quái vật đầu tiên trên màn hình.



Hình 10.4. Quái vật đầu tiên xuất hiện

10.1.2.2. Xây dựng Hạm đội quái vật

Để vẽ một hạm đội, chúng ta cần tính xem có bao nhiêu quái vật có thể phù hợp với màn hình và có bao nhiêu hàng quái vật có thể vừa với màn hình.

a. Xác định có bao nhiêu quái vật phù hợp trong một hàng

Để biết có bao nhiêu quái vật xếp thành một hàng, hãy xem chiều ngang là bao nhiêu. Chiều rộng màn hình được lưu trữ trong settings.screen_width, nhưng ta cần một lề trống ở hai bên của màn hình. Mỗi lề bằng chiều rộng của một quái vật.

Bởi vì chúng ta có hai biên, không gian khả dụng cho quái vật là chiều rộng màn hình trừ đi hai lần chiều rộng của quái vật:

```
available_space_x = settings.screen_width - (2 * alien_width)
```

Chúng ta cũng cần thiết lập khoảng cách giữa những quái vật bằng chiều rộng một quái vật. Không gian cần thiết để hiển thị một quái vật gấp đôi chiều rộng của nó: một chiều rộng cho quái vật và một chiều rộng cho không gian trống ở bên phải của nó. Để tìm số lượng quái vật vừa với màn hình, chúng ta chia không gian khả dụng cho chiều rộng gấp hai lần chiều rộng của quái vật. Sử dụng phép chia (//), từ đó nhận được một số nguyên quái vật:

```
number_aliens_x = available_space_x // (2 * alien_width)
```

b. Tạo một hàng quái vật

Chúng ta sẽ viết lại `_create_fleet()` để tạo toàn bộ quái vật hoạt động:

```
def _create_fleet(self):
    """Create the fleet of aliens."""
    # Create an alien and find the number of aliens in a row.
    # Spacing between each alien is equal to one alien width.
    alien = Alien(self)

    alien_width = alien.rect.width
    available_space_x = self.settings.screen_width - (2 * alien_width)
    number_aliens_x = available_space_x // (2 * alien_width)

    # Create the first row of aliens.
    for alien_number in range(number_aliens_x):
        # Create an alien and place it in the row.
        alien = Alien(self)
        alien.x = alien_width + 2 * alien_width * alien_number
        alien.rect.x = alien.x
        self.aliens.add(alien)
```

Chúng ta cần biết chiều rộng và chiều cao của quái vật trước khi thực hiện các phép tính. Sau đó chúng ta nhận được chiều rộng của quái vật từ thuộc tính độ lớn của nó và lưu trữ giá trị này trong `alien_width`. Tiếp đó, chúng ta tính toán chiều ngang có sẵn cho quái vật và số lượng quái vật có thể phù hợp với không gian đó.

Tiếp theo, chúng ta thiết lập một vòng lặp đếm từ 0 đến số lượng quái vật mà chúng ta cần thực hiện. Trong phần chính của vòng lặp, chúng ta tạo ra một quái vật mới và đặt giá trị tọa độ x của nó vào hàng.

Tiếp theo, chúng ta nhân chiều rộng của quái vật với 2 để tính khoảng không gian mà mỗi quái vật chiếm giữ, bao gồm không gian trống ở bên phải của nó và nhân số lượng này với vị trí của quái vật trong hàng. Chúng ta sử dụng thuộc tính `x` của quái vật để đặt vị trí của `rect`, thêm mỗi quái vật mới vào nhóm quái vật.

Khi ta chạy Alien Invasion ngay bây giờ, ta sẽ thấy hàng quái vật đầu tiên xuất hiện, như trong Hình.



Hình 10.5. Hạm đội quái vật

Hàng đầu tiên được đặt lệch sang bên trái, điều này thực sự tốt cho trò chơi. Lý do là chúng ta muốn hạm đội di chuyển sang phải cho đến khi nó chạm mép của màn hình, sau đó thả xuống một chút rồi di chuyển sang trái, v.v. Giống như trò chơi cổ điển Space Invaders, chuyển động này thú vị hơn là để hạm đội lao thẳng xuống. Chúng tôi sẽ tiếp tục chuyển động này cho đến khi tất cả quái vật bị bắn hạ hoặc cho đến khi quái vật chạm vào tàu hoặc cuối màn hình. Không tùy thuộc vào độ rộng màn hình ta đã chọn, việc căn chỉnh hàng đầu tiên của quái vật có thể hơi khác trên hệ thống.

c. Tái cấu trúc `_create_fleet()`

Chúng ta sẽ thêm một phương thức trợ giúp mới, `_create_alien()` và gọi nó từ `_create_fleet()`:

```
def _create_fleet(self):
    --snip--
    # Create the first row of aliens.
    for alien_number in range(number_of.aliens_x):
        self._create_alien(alien_number)
```

```

def _create_alien(self, alien_number):
    """Create an alien and place it in the row."""
    alien = Alien(self)
    alien_width = alien.rect.width
    alien.x = alien_width + 2 * alien_width * alien_number
    alien.rect.x = alien.x
    self.aliens.add(alien)

```

Phương thức `_create_alien()` yêu cầu một tham số `self`: nó cần số lượng quái vật hiện đang được tạo. Chúng ta vẫn giữ nguyên code cho `_create_fleet()` ngoại trừ việc thêm chiều rộng của một quái vật bên trong phương thức. Tái cấu trúc này dễ dàng giúp ta thêm các hàng mới và tạo toàn hạm đội alien.

d. Thêm hàng

Để hoàn thành hạm đội, chúng ta sẽ xác định số hàng vừa với màn hình và sau đó lặp lại vòng lặp để tạo quái vật trong một hàng cho đến khi chúng ta có số hàng chính xác.

Để xác định số hàng, ta cần tìm không gian trống đúng bằng cách trừ đi chiều cao của quái vật từ đỉnh, con tàu chiều cao từ dưới cùng và hai lần chiều cao của quái vật từ cuối màn hình:

```

available_space_y = settings.screen_height - (3 * alien_height)
- ship_height

```

Kết quả là sẽ tạo ra một số không gian trống phía trên con tàu, để người chơi có một khoảng thời gian bắt đầu bắn quái vật ở đầu mỗi cấp độ. Mỗi hàng cần một số không gian trống bên dưới nó, chúng ta sẽ tạo ra chiều cao của một quái vật. Để tìm số hàng, chúng ta chia không gian có sẵn bằng hai lần chiều cao của quái vật.

```

number_rows = available_height_y // (2 * alien_height)

```

Bây giờ chúng ta đã biết có bao nhiêu hàng phù hợp với một hạm đội, chúng ta có thể lặp lại mã để tạo một hàng tiếp:

```

def _create_fleet(self):
    --snip--
    alien = Alien(self)
    alien_width, alien_height = alien.rect.size
    available_space_x = self.settings.screen_width - (2 * alien_width)
    number_aliens_x = available_space_x // (2 * alien_width)

    # Determine the number of rows of aliens that fit on the screen.
    ship_height = self.ship.rect.height

```

```

available_space_y = (self.settings.screen_height -
                     (3 * alien_height) - ship_height)
number_rows = available_space_y // (2 * alien_height)

# Create the full fleet of aliens.
for row_number in range(number_rows):
    for alien_number in range(number.aliens_x):
        self._create_alien(alien_number, row_number)

def _create_alien(self, alien_number, row_number):
    """Create an alien and place it in the row."""
    alien = Alien(self)
    alien_width, alien_height = alien.rect.size
    alien.x = alien_width + 2 * alien_width * alien_number
    alien.rect.x = alien.x
    alien.rect.y = alien.rect.height + 2 * alien.rect.height *
                   row_number
    self.aliens.add(alien)

```

Để sử dụng chiều rộng và chiều cao của quái vật, chúng ta sử dụng thuộc tính kích thước, chứa một bộ dữ liệu với chiều rộng và chiều cao của một đối tượng. Để tính toán số hàng mà chúng ta có thể xếp vừa trên màn hình, chúng ta dùng ready_space_y ngay sau phép tính cho ready_space_x .

Để tạo nhiều hàng, ta sử dụng hai vòng lặp lồng nhau: một vòng ngoài và một vòng lặp bên trong. Vòng lặp bên trong tạo ra những quái vật trong một hàng. Vòng ngoài tạo ra những hàng quái vật.

Bây giờ khi chúng ta gọi _create_alien(), bao gồm một đối số cho số hàng để mỗi hàng có thể được đặt xa hơn xuống màn hình. _create_alien() cần một tham số để giữ số hàng. Trong _create_alien(), chúng ta thay đổi giá trị tọa độ y của quái vật khi nó không nằm ở hàng đầu tiên(4) bằng cách bắt đầu bằng chiều cao của một quái vật để tạo không gian trống ở đầu màn hình. Mỗi hàng bắt đầu = hai lần chiều cao của quái vật hàng trước, vì vậy ta nhân chiều cao của quái vật với hai và sau đó với số hàng. Số hàng đầu tiên là 0, do đó, vị trí thẳng đứng của hàng đầu tiên là không thay đổi. Tất cả các hàng tiếp theo được đặt ở phía dưới.

Khi ta chạy trò chơi ngay bây giờ, ta sẽ thấy một đội đầy đủ quái vật, như được thể hiện trong Hình dưới.



Hình 10.6. Đội quái vật đầy đủ

Hãy thử

Ngôi sao: Tìm hình ảnh của một ngôi sao. Tạo một lưới các ngôi sao xuất hiện trên màn hình.

Những ngôi sao đẹp hơn: ta có thể tạo ra một mô hình ngôi sao thực tế hơn bằng cách giới thiệu ngẫu nhiên khi ta đặt mỗi ngôi sao. Nhớ lại rằng ta có thể nhận được một số ngẫu nhiên như thế này:

```
from random import randint
random_number = randint(-10, 10)
```

Mã này trả về một số nguyên ngẫu nhiên từ -10 đến 10. Sử dụng mã của ta trong Bài tập ngôi sao, điều chỉnh vị trí của mỗi ngôi sao một cách ngẫu nhiên.

10.1.2.3. Cho quái vật di chuyển

Bây giờ, hãy làm cho hạm đội quái vật di chuyển sang bên phải trên màn hình cho đến khi nó chạm mép, sau đó làm cho nó giảm một lượng đã định và di chuyển theo hướng khác. Chúng tôi sẽ tiếp tục chuyển động này cho đến khi tất cả quái vật bị bắn hạ, va chạm với tàu hoặc đến cuối màn hình. Hãy bắt đầu bằng cách làm cho đội xe di chuyển sang bên phải.

a. Di chuyển quái vật sang phải

Để di chuyển quái vật, ta sẽ sử dụng phương thức update() trong alien.py, phương thức này gọi cho từng quái vật trong nhóm.

Đầu tiên, hãy thêm một cài đặt để kiểm soát tốc độ của từng quái vật, sau đó triển khai trong update():

settings.py

```
def __init__(self):
    --snip--
    # Alien settings
    self.alien_speed = 1.0
```

alien.py

```
def __init__(self, ai_game):
    """Initialize the alien and set its starting position."""
    super().__init__()
    self.screen = ai_game.screen
    self.settings = ai_game.settings
    --snip--

def update(self):
    """Move the alien to the right."""
    self.x += self.settings.alien_speed
    self.rect.x = self.x
```

Chúng ta tạo một tham số cài đặt trong __init__() để có thể truy cập vào tốc độ trong update(). Mỗi khi cập nhật vị trí của quái vật, chúng ta sẽ di chuyển nó sang phải đúng bằng số lượng được lưu trữ trong alien_speed. Chúng ta theo dõi vị trí chính xác của quái vật với thuộc tính self.x, có thể chứa các giá trị thập phân. Sau đó ta sử dụng giá trị của self.x để cập nhật vị trí của quái vật.

Trong vòng lặp while chính, chúng ta đã cập nhật tàu và vị trí đạn. Bây giờ ta sẽ thêm cập nhật vị trí của từng quái vật :

```
alien_invasion.py      while True:
                        self._check_events()
                        self.ship.update()
                        self._update_bullets()
                        self._update.aliens()
                        self._update_screen()
```

Để quản lý sự di chuyển của hạm đội, chúng ta tạo một phương thức mới có tên _update_aliens(). Cập nhật vị trí của quái vật sau khi các viên đạn được cập nhật, vì chúng ta sẽ sớm kiểm tra xem có viên đạn nào trúng quái vật hay không.

Nơi ta đặt phương pháp này trong mô-đun không quan trọng. Nhưng để giữ cho code có tổ chức, ta sẽ đặt code này ngay sau `_update_bullets()` để khớp thứ tự của các lệnh gọi phương thức trong vòng lặp `while`. Đây là phiên bản đầu tiên của `_update.aliens()`:

```
alien_invasion.py
def _update.aliens(self):
    """Update the positions of all aliens in the fleet."""
    self.aliens.update()
```

b. Cài đặt hướng di chuyển của hạm đội

Bây giờ chúng ta sẽ tạo các cài đặt để làm cho hạm đội di chuyển xuống màn hình và sang trái khi nó chạm vào mép phải của màn hình. Dưới đây là cách để cập đến trạng thái này:

```
settings.py          # Alien settings
self.alien_speed = 1.0
self.fleet_drop_speed = 10
# fleet_direction of 1 represents right;-1 represents left.
self.fleet_direction = 1
```

Cài đặt `fleet_drop_speed` kiểm soát tốc độ của hạm đội mỗi khi có quái vật đến một trong hai cạnh. Sẽ rất hữu ích nếu tách ra tốc độ này từ tốc độ ngang của quái vật để ta có thể điều chỉnh hai tốc độ độc lập.

Để triển khai thiết lập `fleet_direction`, chúng ta có thể sử dụng một giá trị, chẳng hạn như là "left" hoặc "right", nhưng chúng ta sẽ kết thúc với việc kiểm tra các câu lệnh `if-elif` cho hướng hạm đội. Vì chúng ta chỉ có hai hướng giải quyết, sử dụng các giá trị 1 và -1 và chuyển đổi giữa chúng mỗi lần đội thay đổi hướng.

c. Kiểm tra xem quái vật có tần công biên giới hay không

Chúng tôi cần một phương thức để kiểm tra xem liệu quái vật có ở hai bên lề hay không và cần sửa đổi `update()` để cho phép mỗi quái vật di chuyển theo hướng thích hợp. Mã này là một phần của lớp Alien:

```
def check_edges(self):
    """Return True if alien is at edge of screen."""
    screen_rect = self.screen.get_rect()
    if self.rect.right >= screen_rect.right
        or self.rect.left <= 0:
        return True
```

```

def update(self):
    """Move the alien right or left."""
    self.x += (self.settings.alien_speed *
               self.settings.fleet_direction)
    self.rect.x = self.x

```

Chúng ta có thể gọi phương thức mới `check_edges()` để xem liệu alien bất kỳ nào đó ở cạnh trái hoặc phải, sửa đổi phương thức `update()` để cho phép chuyển động sang trái hoặc phải bằng cách nhân tốc độ của quái vật với giá trị của `fleet_direction`. Nếu `fleet_direction` là 1, giá trị của `alien_speed` sẽ được thêm vào vị trí hiện tại của quái vật, di chuyển quái vật sang bên phải; nếu `fleet_direction` là -1, giá trị sẽ bị trừ khỏi vị trí của quái vật, di chuyển quái vật sang bên trái.

d. Thả hạm đội và đổi hướng

Khi một quái vật đến rìa, toàn bộ hạm đội cần phải thả xuống và chuyển hướng. Do đó, chúng ta cần thêm một số mã vào `AlienInvasion` bởi vì đó là nơi chúng ta sẽ kiểm tra xem có quái vật nào ở lề bên trái hay lề bên phải. Chúng tôi sẽ biến điều này thành hiện thực bằng cách viết các phương thức `_check_fleet_edges()` và `_change_fleet_direction()`, sau đó sửa đổi `_update.aliens()`. Đặt các phương thức mới này sau `_create_alien()`.

```

def _check_fleet_edges(self):
    """Respond appropriately if any aliens have reached an edge."""
    for alien in self.aliens.sprites():
        if alien.check_edges():
            self._change_fleet_direction()
            break

def _change_fleet_direction(self):
    """Drop the entire fleet and change the fleet's direction."""
    for alien in self.aliens.sprites():
        alien.rect.y += self.settings.fleet_drop_speed
    self.settings.fleet_direction *= -1

```

Trong `_check_fleet_edges()`, chúng ta lặp qua hạm đội và gọi `check_edges()` trên mỗi quái vật. Nếu `check_edges()` trả về True, chúng ta biết một quái vật đang ở cạnh và toàn bộ hạm đội cần phải đổi hướng; nếu vậy chúng ta gọi `_change_fleet_direction()` và thoát ra khỏi vòng lặp. Trong `_change_fleet_direction()`, chúng ta lặp qua tất cả những quái vật và thả từng người một bằng cách sử dụng `fleet_drop_speed`; sau đó chúng ta thay đổi giá trị của `fleet_direction` bằng cách nhân giá trị với -1.

Đây là thay đổi của `_update.aliens()`:

```
def _update.aliens(self):
    """
    Check if the fleet is at an edge,
    then update the positions of all aliens in the fleet.
    """
    self._check_fleet_edges()
    self.aliens.update()
```

Chúng ta đã sửa đổi phương thức này bằng cách gọi `_check_fleet_edges()` cập nhật vị trí của từng quái vật. Khi ta chạy trò chơi ngay bây giờ, đội xe sẽ di chuyển qua lại giữa các cạnh của màn hình và thả xuống mỗi khi nó chạm vào một cạnh. Bây giờ chúng ta có thể bắt đầu bắn hạ quái vật và thấy được quái vật tấn công con tàu hoặc nó sẽ di chuyển đến cuối màn hình.

Hãy thử

Hạt mưa: Tìm hình ảnh một hạt mưa và tạo một mạng lưới các hạt mưa. Làm cho các hạt mưa rơi về phía dưới cùng của màn hình cho đến khi chúng biến mất.

Mưa ổn định: Sửa đổi mã của ta trong Bài tập trên để khi một hàng hạt mưa biến mất khỏi cuối màn hình, một hàng mới sẽ xuất hiện ở đầu màn hình và bắt đầu rơi.

10.1.2.4. Bắn quái vật

Chúng ta đã xây dựng con tàu của mình và một hạm đội quái vật, nhưng khi đạn chạm tới quái vật, chúng sẽ xuyên qua vì chúng ta không kiểm tra va chạm. Trong lập trình trò chơi, va chạm xảy ra khi các yếu tố của trò chơi chồng chéo lên nhau. Để thực hiện các viên đạn bắn hạ quái vật, ta sẽ sử dụng phương thức `sprite.groupcollide()` để tìm kiếm va chạm giữa các thành phần của hai nhóm.

a. Phát hiện va chạm đạn

Khi một viên đạn trúng quái vật và chúng biến mất ngay khi trúng đạn. Để làm điều này, chúng ta sẽ tìm kiếm các va chạm sau khi cập nhật vị trí của tất cả các viên đạn.

Hàm `sprite.groupcollide()` so sánh các phiên bản của mỗi phần tử trong một nhóm với các phiên bản của mỗi phần tử trong một nhóm khác. Trong trường hợp này, nó so sánh vị trí của mỗi viên đạn với vị trí của mỗi quái vật và trả về một từ

điển có chứa các viên đạn và quái vật đã va chạm. Mỗi khóa trong từ điển sẽ là một viên đạn và giá trị tương ứng sẽ là quái vật đã bị bắn trúng.

Thêm mã sau vào cuối `_update_bullets()` để kiểm tra va chạm giữa đạn và quái vật:

```
def _update_bullets(self):
    """Update position of bullets and get rid of old bullets."""
    --snip--
    # Check for any bullets that have hit aliens.
    # If so, get rid of the bullet and the alien.
    collisions = pygame.sprite.groupcollide(
        self.bullets, self.aliens, True, True)
```

Code mới mà chúng ta thêm vào sẽ so sánh vị trí của tất cả các viên đạn trong `self.bullets` và tất cả những quái vật trong `self.aliens`, xác định vật nào trùng lặp. Bất cứ các hành động nào của một viên đạn và quái vật trùng lặp, `groupcollide()` sẽ thêm một cặp giá trị khóa vào từ điển mà nó trả về. Hai đối số `True` báo cho Pygame xóa đạn và quái vật đã va chạm. (Để tạo ra một viên đạn công suất cao có thể đi đến đầu màn hình, tiêu diệt mọi quái vật trên đường đi của nó, ta có thể đặt đối số Boolean đầu tiên thành `False` và giữ đối số Boolean thứ hai đặt thành `True`. Quái vật bắn trúng sẽ biến mất, nhưng tất cả các viên đạn sẽ vẫn hoạt động cho đến khi chúng biến mất khỏi đầu màn hình.)

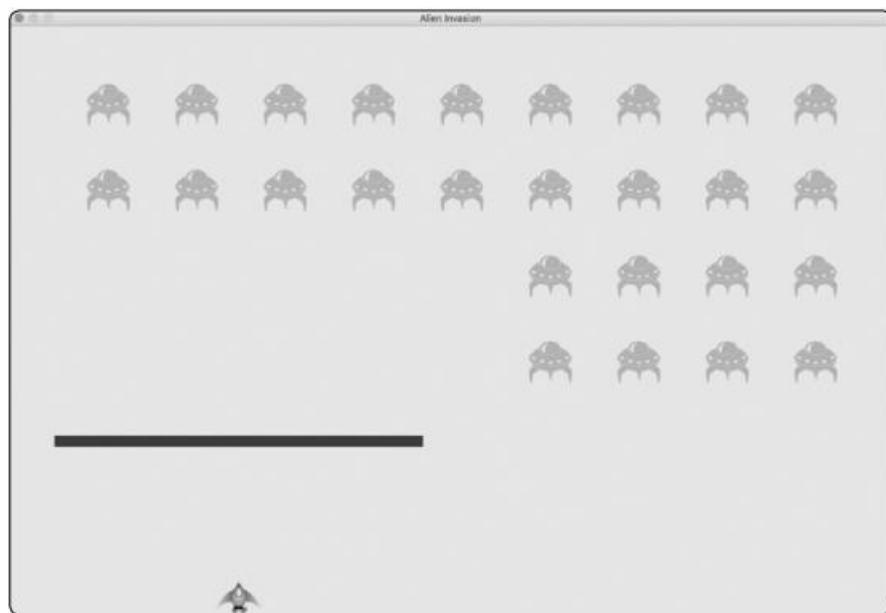
Khi ta chạy Alien Invasion bây giờ, những quái vật ta đụng phải sẽ biến mất. Hình 10.7 cho thấy một hạm đội đã bị bắn hạ một phần.



Hình 10.7. Bắn hạ quái vật

b. Tạo các viên đạn lớn hơn để thử nghiệm

Thay đổi yêu thích của ta khi thử nghiệm Alien Invasion là phạm vi sử dụng những viên đạn thật rộng, chúng vẫn hoạt động ngay cả khi chúng đã bắn trúng quái vật (xem Hình 10.8). Hãy thử đặt bullet_width thành 300, hoặc thậm chí là 3000, để xem ta có thể bắn hạ hạm đội nhanh như thế nào



Hình 10.8. Quái vật bị bắn hạ

Những thay đổi như thế này sẽ giúp ta kiểm tra trò chơi hiệu quả hơn và có thể khơi nguồn cho những ý tưởng để trao quyền hạn điểm thưởng cho người chơi. Chỉ cần nhớ khôi phục cài đặt về bình thường khi ta thử nghiệm xong một tính năng.

c. Tái lập Hạm đội

Một đặc điểm chính của Alien Invasion là quái vật không ngừng hoạt động: mỗi khi hạm đội bị tiêu diệt, một hạm đội mới sẽ xuất hiện. . Nếu đúng như vậy, chúng ta thực hiện gọi tới `_create_fleet()`.

```
def _update_bullets(self):
    --snip--
    if not self.aliens:
        # Destroy existing bullets and create new fleet.
        self.bullets.empty()
        self._create_fleet()
```

Đầu tiên, chúng ta kiểm tra xem nhóm quái vật có trống không. Một nhóm trống sẽ đánh giá là False. Sau đó ta bỏ mọi viên đạn hiện có bằng cách sử dụng phương thức empty(), phương thức này sẽ xóa tất cả các sprite còn lại khỏi một nhóm. Chúng ta cũng gọi _create_fleet(), sẽ lập đầy màn hình với quái vật một lần nữa ngay sau khi ta tiêu diệt hạm đội hiện tại.

d. Tăng tốc độ đạn

Nếu ta đã thử bắn vào quái vật ở trạng thái hiện tại của trò chơi, ta có thể thấy rằng những viên đạn không di chuyển với tốc độ tốt nhất. Chúng có thể hơi chậm trên hệ thống hoặc quá nhanh. Tại thời điểm này, ta có thể sửa đổi cài đặt để làm cho trò chơi trở nên thú vị trên hệ thống của mình.

Ta sửa đổi tốc độ của đạn bằng cách điều chỉnh giá trị của bullet_speed trong settings.py:

```
# Bullet settings
self.bullet_speed = 1.5
self.bullet_width = 3
--snip--
```

e. Tái cấu trúc lại _update_bullet()

Hãy tái cấu trúc _update_bullet() để nó không thực hiện quá nhiều nhiệm vụ khác nhau. Chúng ta sẽ chuyển mã xử lý các vụ va chạm của đạn và quái vật sang một phương thức riêng biệt:

```
def _update_bullets(self):
    --snip--
    # Get rid of bullets that have disappeared.
    for bullet in self.bullets.copy():
        if bullet.rect.bottom <= 0:
            self.bullets.remove(bullet)

    self._check_bullet_alien_collisions()

def _check_bullet_alien_collisions(self):
    """Respond to bullet-alien collisions."""
    # Remove any bullets and aliens that have collided.
    collisions = pygame.sprite.groupcollide(
        self.bullets, self.aliens, True, True)

    if not self.aliens:
        # Destroy existing bullets and create new fleet.
```

```
self.bullets.empty()
self._create_fleet()
```

Chúng ta đã tạo một phương thức mới, `_check_bullet_alien_collisions()`, để xem xét đối với va chạm giữa đạn và quái vật, và phản ứng thích hợp nếu toàn bộ hạm đội đã bị tiêu diệt. Làm như vậy sẽ giúp `_update_bullets()` đơn giản hóa việc phát triển thêm.

Hãy thử

Sideways Shooter Phần 2: Đối với bài tập này, hãy cố gắng phát triển Sideways Shooter giống như chúng ta đã phát triển Alien Invasion. Thêm một đội quái vật, và khiến họ di chuyển sang một bên về phía con tàu. Hoặc, viết mã đặt quái vật tại các vị trí đã chạy dọc theo bên phải của màn hình và sau đó đưa chúng về phía con tàu. Ngoài ra, hãy viết mã làm cho quái vật biến mất khi họ bị bắn.

10.1.2.5. Kết thúc trò chơi

Thử thách thú vị trong một trò chơi là gì để ta không bị thua? Nếu người chơi không bắn hạ hạm đội đủ nhanh, con tàu sẽ bị phá hủy. Đồng thời, chúng ta cũng giới hạn số lượng tàu mà người chơi có thể sử dụng và con tàu sẽ bị phá hủy khi có quái vật đến cuối màn hình. Trò chơi sẽ kết thúc khi người chơi đã sử dụng hết tàu của mình.

a. Phát hiện quái vật và va chạm tàu

Chúng ta sẽ kiểm tra các vụ va chạm của quái vật và tàu ngay sau khi cập nhật vị trí của từng quái vật trong AlienInvasion:

```
def _update.aliens(self):
    --snip--
    self.aliens.update()
    # Look for alien-ship collisions.
    if pygame.sprite.spritecollideany(self.ship, self.aliens):
        print("Ship hit!!!")
```

Hàm `spritecollideany()` nhận hai đối số: một sprite và một nhóm. Hàm này tìm kiếm bất kỳ viên đạn nào của nhóm đã va chạm với sprite và ngừng lặp lại nhóm ngay khi nó tìm thấy một viên đạn đã va chạm với sprite. Tại đây, nó đi qua nhóm aliens và trả lại quái vật đầu tiên mà nó tìm thấy đã va chạm với tàu.

Nếu không có va chạm xảy ra, spritecollideany() trả về None và sau khói if (1)sẽ không thực thi. Nếu nó tìm thấy quái vật đã va chạm với con tàu, nó sẽ trả lại quái vật đó và khói if thực thi: in ra Ship hit !!! . Khi quái vật tấn công con tàu, chúng ta cần thực hiện một số nhiệm vụ: cần xóa tất cả những quái vật và đạn còn lại, cập nhật con tàu và tạo một hạm đội mới. Viết lệnh print() là một cách đơn giản để đảm bảo chúng ta đang phát hiện những va chạm này một cách chính xác.

Bây giờ khi chạy Alien Invasion, thông báo Ship hit !!! sẽ xuất hiện bất cứ khi nào có quái vật bị bắn bởi con tàu. Khi ta đang thử nghiệm tính năng này, hãy đặt alien_drop_speed thành giá trị cao hơn, chẳng hạn như 50 hoặc 100, để quái vật tiếp cận tàu của ta nhanh hơn.

b. Phản hồi khi có va chạm của quái vật và tàu

Bây giờ chúng ta cần tìm hiểu chính xác điều gì sẽ xảy ra khi một quái vật va chạm với con tàu. Thay vì phá hủy phiên bản con tàu và tạo một con tàu mới, chúng ta sẽ đếm số lần con tàu bị tấn công bằng cách theo dõi số liệu thống kê cho trò chơi. Theo dõi thống kê cũng sẽ hữu ích cho việc ghi điểm. Hãy viết một lớp mới, GameStats, để theo dõi thống kê trò chơi và lưu nó dưới dạng game_stats.py:

```
class GameStats:
    """Track statistics for Alien Invasion."""

    def __init__(self, ai_game):
        """Initialize statistics."""
        self.settings = ai_game.settings
        self.reset_stats()

    def reset_stats(self):
        """Initialize statistics that can change during the game."""
        self.ships_left = self.settings.ship_limit
```

Chúng ta sẽ tạo một phiên bản GameStats trong toàn bộ thời gian Alien Invasion đang chạy. Để thực hiện việc này, chúng ta sẽ khởi tạo hầu hết các thống kê trong phương thức reset_stats() thay vì trực tiếp trong __init__(). Hiện tại, chúng ta chỉ có một thống kê, ship_left, giá trị sẽ thay đổi trong suốt trò chơi. Số lượng tàu mà người chơi bắt đầu phải được lưu trữ trong settings.py dưới dạng ship_limit:

```
# Ship settings
self.ship_speed = 1.5
self.ship_limit = 3
```

Chúng ta cũng cần thực hiện một số thay đổi trong `alie_invasion.py` để tạo phiên bản của `GameStats`. Đầu tiên, chúng ta cập nhật các câu lệnh nhập ở đầu tập tin:

```
import sys
from time import sleep
import pygame
from settings import Settings
from game_stats import GameStats
from ship import Ship
--snip--

Chúng ta sẽ tạo một phiên bản của GameStats trong __init__():
def __init__(self):
    --snip--
    self.screen = pygame.display.set_mode(
        (self.settings.screen_width, self.settings.screen_height))
    pygame.display.set_caption("Alien Invasion")

    # Create an instance to store game statistics.
    self.stats = GameStats(self)

    self.ship = Ship(self)
--snip--
```

Khi quái vật tấn công tàu, chúng ta sẽ trừ một trong số tàu còn lại, tiêu diệt tất cả quái vật và đạn hiện có, tạo một hạm đội mới và đặt lại vị trí con tàu ở giữa màn hình. Chúng ta cũng sẽ tạm dừng trò chơi trong giây lát để người chơi có thể nhận thấy vụ va chạm và tập hợp lại trước khi một hạm đội mới xuất hiện.

Hãy đặt mã này trong một phương thức mới có tên `_ship_hit()`. Chúng ta sẽ gọi phương thức này từ `_update_aliens()` khi quái vật tấn công con tàu

```
def _ship_hit(self):
    """Respond to the ship being hit by an alien."""
    # Decrement ships_left.
    self.stats.ships_left -= 1

    # Get rid of any remaining aliens and bullets.
    self.aliens.empty()
    self.bullets.empty()

    # Create a new fleet and center the ship.
    self._create_fleet()
    self.ship.center_ship()
```

```
# Pause.  
sleep(0.5)
```

Phương thức mới `_ship_hit()` điều phối phản ứng khi có quái vật đánh một con tàu. Bên trong `_ship_hit()`, số lượng tàu còn lại giảm đi 1, sau đó nhóm quái vật và đạn chuyển thành rỗng.

Tiếp theo, chúng ta tạo một hạm đội mới và đặt con tàu vào giữa. Sau đó, chúng ta tạm dừng tất cả sau khi cập nhật xong, để cho người chơi có thể thấy rằng tàu của họ đã bị bắn. Lệnh `sleep()` tạm dừng thực hiện chương trình trong nửa giây, đủ thời gian để người chơi thấy rằng quái vật đã va vào con tàu. Khi hàm `sleep()` kết thúc, việc thực thi mã chuyển sang phương thức `_update_screen()`, phương thức này sẽ đưa nhóm mới ra màn hình.

Trong `_update.aliens()`, chúng ta thay thế lệnh gọi `print()` bằng lệnh gọi tới `_ship_hit()` khi một quái vật tấn công con tàu:

```
def _update.aliens(self):  
    --snip--  
    if pygame.sprite.spritecollideany(self.ship, self.aliens):  
        self._ship_hit()
```

Dưới đây là phương thức `center_ship()`, thêm nó vào cuối `ship.py`:

```
def center_ship(self):  
    """Center the ship on the screen."""  
    self.rect.midbottom = self.screen_rect.midbottom  
    self.x = float(self.rect.x)
```

Chạy trò chơi, bắn một vài quái vật và để một quái vật tấn công con tàu. Các trò chơi sẽ tạm dừng và một hạm đội mới sẽ xuất hiện với con tàu làm trung tâm ở cuối màn hình một lần nữa.

c. Quái vật chạm tới cuối màn hình

Nếu có quái vật đến cuối màn hình, trò chơi sẽ phản hồi giống như khi quái vật va vào tàu. Để kiểm tra thời điểm xảy ra lỗi này, hãy thêm một phương thức mới trong `alien_invasion.py`:

```
def _check.aliens_bottom(self):  
    """Check if any aliens have reached the bottom of the screen."""  
    screen_rect = self.screen.get_rect()  
    for alien in self.aliens.sprites():  
        if alien.rect.bottom >= screen_rect.bottom:
```

```

    # Treat this the same as if the ship got hit.
    self._ship_hit()
    break

```

Phương thức `_check.aliens_bottom()` kiểm tra xem có quái vật nào có xuống cuối màn hình hay không. Một quái vật chạm đến cuối màn hình khi giá trị `alien.rect.bottom >= screen_rect.bottom`. Nếu có quái vật chạm đến đáy, chúng ta gọi `_ship_hit()` và thoát khỏi vòng lặp sau khi gọi `_ship_hit()`, gọi phương thức này từ `_update.aliens()`:

```

def _update.aliens(self):
    --snip--
    # Look for alien-ship collisions.
    if pygame.sprite.spritecollideany(self.ship, self.aliens):
        self._ship_hit()
    # Look for aliens hitting the bottom of the screen.
    self._check.aliens_bottom()

```

Gọi `_check.aliens_bottom()` sau khi ta cập nhật vị trí của tất cả quái vật và sau khi tìm kiếm va chạm giữa quái vật và tàu. Bây giờ một hạm đội mới sẽ xuất hiện mỗi khi tàu bị quái vật va vào hoặc quái vật xuống đáy của màn hình.

d. Kết thúc trò chơi

Alien Invasion đã dần hoàn thiện hơn nhiều, nhưng trò chơi không bao giờ kết thúc. Giá trị của `ship_left` càng ngày càng tăng. Hãy thêm `game_active` làm một thuộc tính của `GameStats` để kết thúc trò chơi khi người chơi hết tàu. Chúng tôi sẽ đặt cờ này ở cuối phương thức `__init__()` trong `GameStats`

```

def __init__(self, ai_game):
    --snip--
    # Start Alien Invasion in an active state.
    self.game_active = True

```

Bây giờ chúng ta thêm mã vào `_ship_hit()` đặt `game_active` thành `False` khi người chơi đã sử dụng hết các tàu của họ:

```

def _ship_hit(self):
    """Respond to ship being hit by alien."""
    if self.stats.ships_left > 0:
        # Decrement ships_left.
        self.stats.ships_left -= 1
        --snip--
        # Pause.
        sleep(0.5)

```

```

else:
    self.stats.game_active = False

```

Hầu hết `_ship_hit()` là không thay đổi. Chúng ta chỉ di chuyển tất cả mã hiện có vào một khái if, khái này sẽ kiểm tra để đảm bảo rằng người chơi còn lại ít nhất một con tàu. Nếu vậy, chúng ta tạo một hạm đội mới, tạm dừng và tiếp tục. Nếu người chơi không còn tàu, chúng ta đặt `game_active` thành False.

e. Xác định thời điểm các phần của trò chơi nên chạy

Chúng ta cần xác định các phần của trò chơi luôn luôn chạy và các phần chỉ nên chạy khi trò chơi đang hoạt động:

```

def run_game(self):
    """Start the main loop for the game."""
    while True:
        self._check_events()
        if self.stats.game_active:
            self.ship.update()
            self._update_bullets()
            self._update.aliens()
            self._update_screen()

```

Trong vòng lặp chính, chúng ta luôn gọi `_check_events()`, ngay cả khi trò chơi không hoạt động.

Hay thử

Trò chơi kết thúc: Trong Sideways Shooter, hãy theo dõi số lần tàu bị đâm và số lần quái vật bị tàu đâm. Quyết định điều kiện thích hợp để kết thúc trò chơi và dừng trò chơi khi tình huống này xảy ra.

10.1.3. Ghi điểm

10.1.3.1. Thêm nút Play

Trong phần này, chúng ta sẽ thêm nút Play xuất hiện trước khi trò chơi bắt đầu và xuất hiện lại khi trò chơi kết thúc để người chơi có thể chơi lại. Hiện tại trò chơi sẽ bắt đầu ngay sau khi ta chạy `alien_invasion.py`. Vì vậy hãy bắt đầu trò chơi ở trạng thái không hoạt động và sau đó nhắc người chơi nhấp vào nút **Play** để bắt đầu. Để thực hiện việc này, hãy sửa đổi phương thức `__init__()` của GameStats:

```

def __init__(self, ai_game):
    """Initialize statistics."""

```

```

self.settings = ai_game.settings
self.reset_stats()
# Start game in an inactive state.
self.game_active = False

```

a. Tạo một lớp Button

Bởi vì Pygame không có phương pháp tích hợp để tạo nút, nên ta sẽ viết một lớp Button để tạo một hình chữ nhật có nhãn như nút Play. Bạn có thể sử dụng mã này để tạo bất kỳ nút nào trong trò chơi. Đây là phần đầu tiên của lớp Button; lưu nó dưới dạng button.py:

```

import pygame.font
class Button:
    def __init__(self, ai_game, msg):
        """Initialize button attributes."""
        self.screen = ai_game.screen
        self.screen_rect = self.screen.get_rect()

        # Set the dimensions and properties of the button.
        self.width, self.height = 200, 50
        self.button_color = (0, 255, 0)
        self.text_color = (255, 255, 255)
        self.font = pygame.font.SysFont(None, 48)

        # Build the button's rect object and center it.
        self.rect = pygame.Rect(0, 0, self.width, self.height)
        self.rect.center = self.screen_rect.center

        # The button message needs to be prepped only once.
        self._prep_msg(msg)

```

Đầu tiên, chúng ta cài đặt mô-đun pygame.font, mô-đun này cho phép Pygame hiển thị văn bản lên màn hình. Phương thức `__init__()` nhận các tham số, đối tượng `ai_game` và `msg`, chứa văn bản của nút. Chúng ta đặt kích thước của nút tại, sau đó đặt `button_color` để tô màu cho đối tượng hình chữ nhật của nút là màu xanh lục sáng và đặt `text_color` để hiển thị văn bản bằng màu trắng.

Sau đó, ta cài đặt một thuộc tính phông chữ để kết xuất văn bản. Đối số `None` yêu cầu Pygame sử dụng phông chữ mặc định và 48 là chỉ định kích thước của văn bản. Để căn giữa nút trên màn hình, đặt thuộc tính trung tâm của nó để khớp với thuộc tính của màn hình. Pygame hoạt động với văn bản bằng cách hiển thị chuỗi ta muốn

hiển thị dưới dạng hình ảnh. Cuối cùng, chúng ta gọi `_prep_msg()` để xử lý kết xuất này. Đây là mã cho `_prep_msg()`:

```
def _prep_msg(self, msg):
    """Turn msg into a rendered image and center text on the button."""
    self.msg_image = self.font.render(msg, True, self.text_color,
                                      self.button_color)
    self.msg_image_rect = self.msg_image.get_rect()
    self.msg_image_rect.center = self.rect.center
```

Phương thức `_prep_msg()` cần một tham số `self` và văn bản được định dạng dưới dạng hình ảnh (`msg`). Lệnh gọi tới `font.render()` biến văn bản được lưu trữ trong `msg` thành một hình ảnh, sau đó lưu trữ trong `self.msg_image`. Phương thức `font.render()` cũng nhận giá trị Boolean để bật hoặc tắt. Các đối số còn lại là màu phông chữ và màu nền được chỉ định. Chúng ta căn giữa hình ảnh văn bản trên nút bằng cách tạo một hình chữ nhật từ hình ảnh và đặt thuộc tính trung tâm của nó để khớp với thuộc tính của nút.

Cuối cùng là tạo một phương thức `draw_button()` để hiển thị nút trên màn hình:

```
def draw_button(self):
    # Draw blank button and then draw message.
    self.screen.fill(self.button_color, self.rect)
    self.screen.blit(self.msg_image, self.msg_image_rect)
```

Chúng ta gọi `screen.fill()` để vẽ phần hình chữ nhật của nút. Sau đó gọi `screen.blit()` để vẽ hình ảnh văn bản ra màn hình, truyền cho nó một hình ảnh và đổi tương vị trí được liên kết với hình ảnh. Điều này hoàn thành lớp `Button`.

b. Vẽ nút lên màn hình

Chúng ta sẽ sử dụng class `Button` để tạo nút `Play` trong `AlienInvasion`. Trước tiên, ta cập nhật các câu lệnh cài đặt:

```
--snip--
from game_stats import GameStats
from button import Button
```

Bởi vì game mà ta xây dựng chỉ cần một nút `Play` nên sẽ tạo nút trong phương thức `__init__()` của `AlienInvasion.py`:

```
def __init__(self):
    --snip--
    self._create_fleet()
```

```
# Make the Play button.
self.play_button = Button(self, "Play")
```

Mã này tạo một bản của Button có tên nhãn là Play, nhưng nó chưa hiện nút ra màn hình. Vì vậy ta sẽ gọi phương thức `draw_button()` của button trong `_update_screen()`:

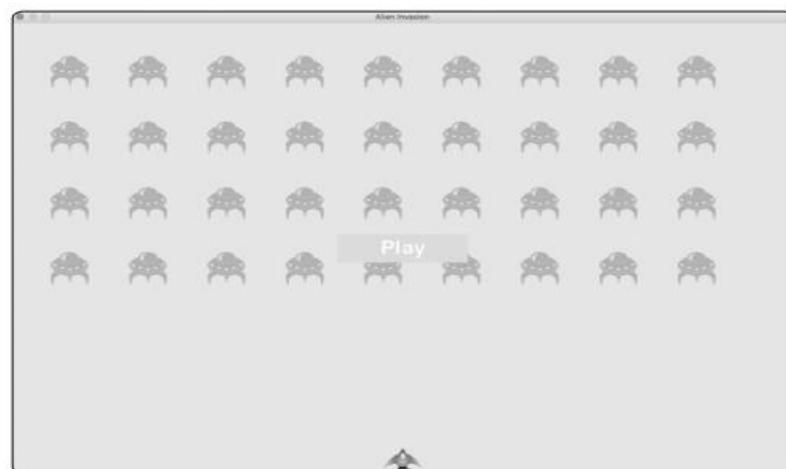
```
def _update_screen(self):
    --snip--
    self.aliens.draw(self.screen)

    # Draw the play button if the game is inactive.
    if not self.stats.game_active:
        self.play_button.draw_button()

pygame.display.flip()
```

Để làm cho nút Play hiển thị phía trên tất cả các đối tượng trò chơi trên màn hình, chúng ta vẽ nó sau khi các đối tượng đã được vẽ, cài đặt nó trong một khối if, để nút chỉ xuất hiện khi trò chơi không hoạt động.

Bây giờ khi chạy Alien Invasion, ta sẽ thấy nút Play giữa màn hình, như trong Hình 14-1



Hình 10.9. nút Play xuất hiện khi trò chơi chưa hoạt động

c. Bắt đầu trò chơi

Để bắt đầu một trò chơi mới khi người chơi nhấn vào nút **Play**, hãy thêm biểu tượng sau chặn ở cuối `_check_events()` để theo dõi các sự kiện chuột qua nút:

```
def _check_events(self):
    """Respond to keypresses and mouse events."""
    for event in pygame.event.get():
```

```

if event.type == pygame.QUIT:
    --snip--
elif event.type == pygame.MOUSEBUTTONDOWN:
    mouse_pos = pygame.mouse.get_pos()
    self._check_play_button(mouse_pos)

```

Pygame phát hiện sự kiện MOUSEBUTTONDOWN khi người chơi nhấp vào bất kỳ đâu trên màn hình , nhưng chúng ta muốn giới hạn trò chơi của mình chỉ phản hồi ở những lần nhấp chuột vào nút Play. Để thực hiện điều này, chúng ta sử dụng pygame.mouse.get_pos(), trả về một bộ chứa tọa độ x và y của con trỏ chuột khi nhấp vào nút chuột. Sau đó gửi các giá trị này tới phương thức mới _check_play_button().

Đây là _check_play_button được chọn để đặt sau _check_events():

```

def _check_play_button(self, mouse_pos):
    """Start a new game when the player clicks Play."""
    if self.play_button.rect.collidepoint(mouse_pos):
        self.stats.game_active = True

```

Chúng ta sử dụng phương thức rect.collidepoint() để kiểm tra xem tọa độ của vị trí nhấp chuột có trùng với vị trí của nút Phát hay không. Nếu trùng, đặt game_active thành True và trò chơi bắt đầu!

Tại thời điểm này, ta sẽ có thể bắt đầu và chơi một trò chơi đầy đủ. Khi nào trò chơi kết thúc, giá trị của game_active sẽ trở thành False và nút Play sẽ xuất hiện lại.

d. Đặt lại trò chơi

Mã nút Play mà chúng ta vừa viết hoạt động khi người chơi nhấp vào Play lần đầu tiên. Tuy nhiên nó không hoạt động sau khi trò chơi đầu tiên kết thúc vì các điều kiện chưa được đặt lại.

Để chơi lại trò chơi mỗi khi người chơi nhấp vào Play, chúng ta cần đặt lại số liệu thống kê của trò chơi, dọn sạch những quái vật và đạn cũ, xây dựng một hạm đội mới và căn giữa con tàu, như hiển thị dưới đây:

```

def _check_play_button(self, mouse_pos):
    """Start a new game when the player clicks Play."""
    if self.play_button.rect.collidepoint(mouse_pos):
        # Reset the game statistics.
        self.stats.reset_stats()
        self.stats.game_active = True

```

```

# Get rid of any remaining aliens and bullets.
self.aliens.empty()
self.bullets.empty()

# Create a new fleet and center the ship.
self._create_fleet()
self.ship.center_ship()

```

Đầu tiên, chúng ta đặt lại số liệu thống kê trò chơi, cung cấp cho người chơi ba chiếc tàu. Sau đó đặt game_active thành **True** để trò chơi sẽ bắt đầu ngay sau khi hàm này chạy xong. Tiếp theo, ta xóa tất cả quái vật và nhóm đạn, sau đó tạo một hạm đội mới và con tàu mới hiện ở trung tâm màn hình.

Bây giờ trò chơi sẽ đặt lại đúng cách mỗi khi ta nhấp vào **Play**, cho phép ta chơi bao nhiêu lần tùy thích!

e. Tắt nút Play

Một vấn đề với nút Play là vùng nút trên màn hình sẽ tiếp tục phản hồi các lần nhấp ngay cả khi nút Play không hiển thị. Nếu vô tình nhấp vào đúng vị trí nút Play sau khi trò chơi bắt đầu thì trò chơi sẽ khởi động lại!

Để khắc phục điều này, hãy đặt trò chơi chỉ bắt đầu khi game_active là `False`:

```

def _check_play_button(self, mouse_pos):
    """Start a new game when the player clicks Play."""
    button_clicked=self.play_button.rect.collidepoint(mouse_pos)
    if button_clicked and not self.stats.game_active:
        # Reset the game statistics.
        self.stats.reset_stats()
    --snip--

```

f. Ân con trỏ chuột

Chúng ta muốn con trỏ chuột hiển thị để bắt đầu chơi, nhưng khi quá trình chơi, nó sẽ cản trở. Để khắc phục điều này, chúng ta sẽ ẩn nó khi trò chơi hoạt động. Ta có thể thực hiện việc này ở cuối câu lệnh `if` trong `_check_play_button()`:

```

def _check_play_button(self, mouse_pos):
    """Start a new game when the player clicks Play."""
    button_clicked=self.play_button.rect.collidepoint(mouse_pos)
    if button_clicked and not self.stats.game_active:
    --snip--
    # Hide the mouse cursor.
    pygame.mouse.set_visible(False)

```

Chuyển set_visible() thành False để làm cho Pygame điều khiển và ẩn con trỏ khi chuột ở trên cửa sổ trò chơi. Chúng ta sẽ làm cho con trỏ xuất hiện lại sau khi trò chơi kết thúc để người chơi có thể bấm Play để bắt đầu một trò chơi mới.

Đây là đoạn code để làm điều đó:

```
def _ship_hit(self):
    """Respond to ship being hit by alien."""
    if self.stats.ships_left > 0:
        --snip--
    else:
        self.stats.game_active = False
        pygame.mouse.set_visible(True)
```

Hiển thị lại con trỏ ngay khi trò chơi không hoạt động, điều này xảy ra trong _ship_hit(). Việc chú ý đến những chi tiết như thế này khiến trò chơi của ta trông chuyên nghiệp hơn và cho phép người chơi tập trung vào việc chơi hơn là tìm hiểu giao diện người dùng.

Hãy thử

Nhấn P để chơi: Vì Alien Invasion sử dụng bàn phím nhập để điều khiển tàu, nên sẽ rất hữu ích nếu ta bắt đầu trò chơi bằng một lần nhấn phím. Thêm đoạn code cho phép người chơi nhấn P để bắt đầu. Có thể hữu ích khi chuyển một số mã từ _check_play_button() sang phương thức _start_game() có thể được gọi từ _check_play_button() và _check_keydown_events().

Thực hành: Tạo một hình chữ nhật ở cạnh phải của màn hình di chuyển lên và xuống với tốc độ ổn định. Sau đó, có một con tàu xuất hiện ở phía bên trái của màn hình mà người chơi có thể di chuyển lên xuống trong khi bắn đạn vào mục tiêu hình chữ nhật đang di chuyển. Thêm nút Play để bắt đầu trò chơi và khi người chơi bắn trượt mục tiêu ba lần, hãy kết thúc trò chơi và làm cho nút Play xuất hiện lại. Cho phép người chơi khởi động lại trò chơi bằng nút Play này.

10.1.3.2. Tăng mức độ chơi

a. Sửa đổi cài đặt tốc độ

Trước tiên, chúng ta sẽ cài đặt lại lớp Setting. Đây là phương thức __init__() cho settings.py:

```
def __init__(self):
    """Initialize the game's static settings."""
    # Screen settings
```

```

self.screen_width = 1200
self.screen_height = 800
self.bg_color = (230, 230, 230)
# Ship settings
self.ship_limit = 3
# Bullet settings
self.bullet_width = 3
self.bullet_height = 15
self.bullet_color = 60, 60, 60
self.bullets_allowed = 3
# Alien settings
self.fleet_drop_speed = 10

# How quickly the game speeds up
self.speedup_scale = 1.1
self.initialize_dynamic_settings()

```

Tiếp tục khởi tạo những cài đặt trong phương thức `__init__()`. Thêm cài đặt `speedup_scale` để kiểm soát tốc độ trò chơi tăng nhanh như thế nào: giá trị 2 sẽ tăng gấp đôi tốc độ trò chơi mỗi khi người chơi đạt đến cấp độ mới; giá trị 1 sẽ giữ cho tốc độ không đổi. Giá trị như 1.1 sẽ tăng tốc độ đủ để làm cho trò chơi trở nên thử thách nhưng vẫn chưa phải là khó. Cuối cùng, gọi phương thức `initialize_dynamic_settings()` để khởi tạo các giá trị cho các thuộc tính cần thay đổi trong suốt trò chơi.

Đây là đoạn code cho `initialize_dynamic_settings()`:

```

def initialize_dynamic_settings(self):
    """Initialize settings that change throughout the game."""
    self.ship_speed = 1.5
    self.bullet_speed = 3.0
    self.alien_speed = 1.0

    # fleet_direction of 1 represents right; -1 represents left.
    self.fleet_direction = 1

```

Phương thức này đặt các tốc độ ban đầu cho con tàu, viên đạn và quái vật. Chúng ta sẽ tăng các tốc độ này khi người chơi tiến triển trong trò chơi và đặt lại chúng mỗi khi người chơi bắt đầu một trò chơi mới.

Bao gồm `fleet_direction` trong phương thức này để quái vật luôn chuyển hướng khi bắt đầu trò chơi mới. Chúng ta không cần tăng giá trị của `fleet_drop_speed`, vì khi

quái vật di chuyển nhanh hơn trên màn hình, họ cũng sẽ đi xuống màn hình nhanh hơn.

Để tăng tốc độ của tàu, đạn và quái vật khi người chơi đạt đến một cấp độ mới, chúng ta sẽ viết một phương thức mới có tên là increase_speed():

```
def increase_speed(self):
    """Increase speed settings."""
    self.ship_speed *= self.speedup_scale
    self.bullet_speed *= self.speedup_scale
    self.alien_speed *= self.speedup_scale
```

Để tăng tốc độ cho các thành phần trong trò chơi này, hãy nhân mỗi tốc độ với giá trị của speedup_scale.

Chúng ta tăng nhịp độ của trò chơi bằng cách gọi gain_speed() trong _check_bullet_alien_collisions() khi quái vật cuối cùng trong hạm đội bị bắn hạ:

```
def _check_bullet_alien_collisions(self):
    --snip--
    if not self.aliens:
        # Destroy existing bullets and create new fleet.
        self.bullets.empty()
        self._create_fleet()
        self.settings.increase_speed()
```

b. Đặt lại tốc độ

Bây giờ chúng ta cần trả mọi cài đặt đã thay đổi về giá trị ban đầu của chúng mỗi khi người chơi bắt đầu một trò chơi mới; nếu không, mỗi trò chơi mới sẽ bắt đầu với cài đặt tốc độ tăng lên của trò chơi trước:

```
def _check_play_button(self, mouse_pos):
    """Start a new game when the player clicks Play."""
    button_clicked = self.play_button.rect.collidepoint(mouse_pos)
    if button_clicked and not self.stats.game_active:
        # Reset the game settings.
        self.settings.initialize_dynamic_settings()
    --snip--
```

Chơi Alien Invasion bây giờ sẽ thú vị và đầy thử thách hơn. Mỗi lần ta bắn hạ được hết alien, trò chơi sẽ tăng tốc độ và trở nên khó khăn hơn một chút. Nếu trò chơi

trở nên quá khó quá nhanh, hãy giảm giá trị của settings.speedup_scale. Hoặc nếu trò chơi không đủ thử thách, hãy tăng giá trị lên một chút.

Hay thử

Mục tiêu đầy thử thách: Làm cho mục tiêu di chuyển nhanh hơn khi trò chơi diễn ra và khởi động lại mục tiêu ở tốc độ ban đầu khi người chơi nhấp vào Play.

Mức độ khó: Tạo một bộ nút cho Alien Invasion cho phép người chơi chọn mức độ khó bắt đầu thích hợp cho trò chơi. Mỗi cấp độ phải chỉ định các giá trị thích hợp cho các thuộc tính trong Cài đặt cần thiết để tạo ra các mức độ khó khác nhau.

10.1.3.3. Ghi điểm

Hãy triển khai hệ thống tính điểm để theo dõi điểm của người chơi trong thời gian thực và hiển thị điểm cao, cấp độ và số lượng tàu còn lại. Điểm số là thông kê trò chơi, vì vậy chúng ta sẽ thêm thuộc tính điểm số vào GameStats:

```
class GameStats:  
    --snip--  
    def reset_stats(self):  
        """Initialize statistics that can change during the game."""  
        self.ships_left = self.ai_settings.ship_limit  
        self.score = 0
```

Để đặt lại điểm số mỗi khi trò chơi mới bắt đầu, chúng ta khởi tạo điểm số trong reset_stats() thay vì __init__().

a. Hiển thị điểm số

Để hiển thị điểm số trên màn hình, trước tiên chúng ta tạo một lớp mới Scoreboard. Lớp này sẽ chỉ hiển thị điểm hiện tại, nhưng cuối cùng chúng ta cũng sẽ sử dụng nó để báo cáo điểm cao, cấp độ và số lượng tàu còn lại.

Đây là phần đầu tiên của lớp Scoreboard; lưu nó dưới dạng scoreboard.py:

```
import pygame.font  
class Scoreboard:  
    """A class to report scoring information."""  
    def __init__(self, ai_game):  
        """Initialize scorekeeping attributes."""  
        self.screen = ai_game.screen  
        self.screen_rect = self.screen.get_rect()  
        self.settings = ai_game.settings
```

```

self.stats = ai_game.stats

# Font settings for scoring information.
self.text_color = (30, 30, 30)
self.font = pygame.font.SysFont(None, 48)
game_stats.py
scoreboard.py
Scoring 289
# Prepare the initial score image.
self.prep_score()

```

Vì Scoreboard ghi văn bản lên màn hình, chúng ta bắt đầu bằng cách cài đặt mô-đun pygame.font. Tiếp theo, cung cấp cho `__init__()` tham số `ai_game` để nó có thể truy cập vào các đối tượng cài đặt, màn hình và số liệu thống kê, mà nó sẽ cần để báo cáo các giá trị mà chúng ta đang theo dõi. Sau đó, ta đặt một màu văn bản và khởi tạo một đối tượng phông chữ.

Để biến văn bản được hiển thị thành hình ảnh, ta gọi là `prep_score()`, cái hàm mà ta sẽ định nghĩa ở dưới đây:

```

def prep_score(self):
    """Turn the score into a rendered image."""
    score_str = str(self.stats.score)
    self.score_image = self.font.render(score_str, True,
                                         self.text_color, self.settings.bg_color)

    # Display the score at the top right of the screen.
    self.score_rect = self.score_image.get_rect()
    self.score_rect.right = self.screen_rect.right - 20
    self.score_rect.top = 20

```

Trong `prep_score()`, biến `stats.score` là giá trị số thành một chuỗi, sau đó chuyển chuỗi này đến `render()`, tạo ra hình ảnh. Để hiển thị điểm rõ ràng trên màn hình, chúng ta chuyển màu nền của màn hình và văn bản màu để `render()`.

Ta đặt điểm số ở góc trên bên phải của màn hình và để nó mở rộng sang trái khi điểm số tăng lên và chiều rộng của số lượng tăng lên. Để đảm bảo điểm luôn thẳng hàng với phía bên phải của màn hình, hãy tạo một hình chữ nhật gọi là `score_rect`, đặt cạnh phải của nó, cách cạnh phải của màn hình 20 pixel. Sau đó, chúng ta đặt cạnh trên cùng 20 pixel xuống từ phía trên cùng của màn hình. Tạo một phương thức `show_score()` để hiển thị điểm số :

```
def show_score(self):
```

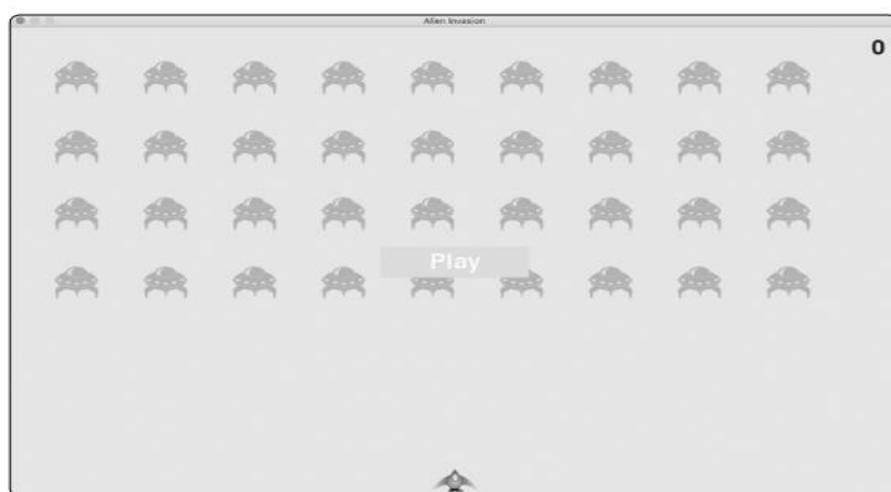
```
"""Draw score to the screen."""
self.screen.blit(self.score_image, self.score_rect)
```

b. Lập bảng điểm

Để hiển thị điểm số, chúng ta sẽ tạo một phiên bản Bảng điểm trong AlienInvasion. Trước tiên, hãy cập nhật các câu lệnh nhập:

```
--snip--
from game_stats import GameStats
from scoreboard import Scoreboard
--snip--
Tiếp theo, chúng ta tạo một bản của Scoreboard trong __init__():
def __init__(self):
    --snip--
    pygame.display.set_caption("Alien Invasion")
    # Create an instance to store game statistics,
    # and create a scoreboard.
    self.stats = GameStats(self)
    self.sb = Scoreboard(self)
--snip--
Sau đó, vẽ bảng điểm lên màn hình trong _update_screen():
def _update_screen(self):
    --snip--
    self.aliens.draw(self.screen)
    # Draw the score information.
    self.sb.show_score()
    # Draw the play button if the game is inactive.
    --snip--
```

Chúng ta gọi show_score() ngay trước khi vẽ nút Play. Khi chạy Alien Invasion ngay bây giờ, số 0 sẽ xuất hiện ở trên cùng bên phải của màn hình.



Hình 10.10. điểm số xuất hiện phía bên phải màn hình

c. Cập nhật điểm số khi quái vật bị bắn hạ

Để ghi tỷ số trực tiếp trên màn hình, chúng ta cập nhật giá trị của stats.score khi một quái vật bất kỳ nào bị tấn công, sau đó gọi prep_score() để cập nhật hình ảnh tỷ số. Nhưng trước tiên, hãy xác định xem người chơi nhận được bao nhiêu điểm mỗi khi bắn hạ một quái vật:

```
def initialize_dynamic_settings(self):  
    --snip--  
  
    # Scoring  
    self.alien_points = 50
```

Chúng ta sẽ tăng giá trị điểm của mỗi quái vật khi trò chơi diễn ra. Để đảm bảo giá trị điểm này được đặt lại mỗi khi một trò chơi mới bắt đầu, chúng ta đặt giá trị trong initialize_dynamic_settings().

Hãy cập nhật điểm số mỗi khi một quái vật bị bắn hạ trong check_bullet_alien_collisions():

```
def _check_bullet_alien_collisions(self):  
    """Respond to bullet-alien collisions."""  
    # Remove any bullets and aliens that have collided.  
    collisions = pygame.sprite.groupcollide(  
        self.bullets, self.aliens, True, True)  
    if collisions:  
        self.stats.score += self.settings.alien_points  
        self.sb.prep_score()  
    --snip--
```

Khi một viên đạn bắn trúng quái vật, Pygame trả về một từ điển biểu thị sự va chạm. Chúng ta sẽ kiểm tra xem từ điển có tồn tại hay không và nếu có, giá trị của quái vật sẽ được cộng vào điểm số. Sau đó, chúng ta gọi prep_score() để tạo điểm số mới rồi cập nhật nó.

Bây giờ khi ta chơi Alien Invasion, ta sẽ có thể tính được điểm!

d. Đặt lại điểm

Hiện tại, chúng ta chỉ chuẩn bị một điểm số mới sau khi một quái vật bị bắn trúng nhưng chúng ta vẫn thấy điểm số cũ khi một trò chơi mới bắt đầu cho đến khi quái vật đầu tiên bị bắn trúng trong trò chơi mới.

Khắc phục điều này bằng cách đặt lại điểm trước khi bắt đầu một trò chơi mới:

```

def _check_play_button(self, mouse_pos):
    --snip--
    if button_clicked and not self.stats.game_active:
        --snip--
        # Reset the game statistics.
        self.stats.reset_stats()
        self.stats.game_active = True
        self.sb.prep_score()
    --snip--

```

e. Đảm bảo tính điểm cho tất cả các lần bắn hạ quái vật

Như code viết hiện tại, chúng ta có thể không ghi được điểm cho một số quái vật. Ví dụ: nếu hai viên đạn va chạm với quái vật trong cùng một lần đi qua vòng lặp hoặc nếu chúng ta tạo một viên đạn cực rộng để bắn trúng nhiều quái vật, người chơi sẽ chỉ nhận được điểm khi bắn trúng một trong số những quái vật. Để khắc phục điều này, hãy điều chỉnh cách phát hiện va chạm giữa đạn và quái vật.

Trong `_check_bullet_alien_collisions()`, bất kỳ viên đạn nào va chạm với quái vật sẽ trở thành một khóa trong từ điển va chạm. Giá trị được liên kết với mỗi viên đạn là danh sách những quái vật mà nó đã va chạm. Chúng ta lặp lại các giá trị trong từ điển va chạm để đảm bảo thưởng điểm cho mỗi lần bắn quái vật:

```

def _check_bullet_alien_collisions(self):
    --snip--
    if collisions:
        for aliens in collisions.values():
            self.stats.score+=self.settings.alien_points*len(aliens)
            self.sb.prep_score()
    --snip--

```

Nếu từ điển va chạm đã được xác định, chúng ta lặp qua tất cả các giá trị trong từ điển. Hãy nhớ rằng mỗi giá trị là một danh sách những quái vật bị trúng một viên đạn. Chúng ta nhân giá trị của mỗi quái vật với số lượng quái vật trong mỗi danh sách và cộng số điểm này vào điểm số hiện tại. Để kiểm tra điều này, hãy thay đổi chiều rộng của viên đạn thành 300 pixel và xác minh rằng ta nhận được điểm cho mỗi quái vật mà ta bắn trúng bằng viên đạn cực rộng của mình; sau đó trả lại độ rộng viên đạn về giá trị bình thường.

f. Tăng điểm

Vì trò chơi trở nên khó hơn mỗi khi người chơi đạt đến cấp độ mới, quái vật ở các cấp độ sau sẽ khó khăn hơn để bắn hạ. Để triển khai tính chất vui nhộn này, chúng ta sẽ thêm mã để tăng giá trị điểm khi tốc độ của trò chơi tăng lên:

```
class Settings:  
    """A class to store all settings for Alien Invasion."""  
    def __init__(self):  
        --snip--  
        # How quickly the game speeds up  
        self.speedup_scale = 1.1  
        # How quickly the alien point values increase  
        self.score_scale = 1.5  
        self.initialize_dynamic_settings()  
    def initialize_dynamic_settings(self):  
        --snip--  
  
    def increase_speed(self):  
        """Increase speed settings and alien point values."""  
        self.ship_speed *= self.speedup_scale  
        self.bullet_speed *= self.speedup_scale  
        self.alien_speed *= self.speedup_scale  
  
        self.alien_points=int(self.alien_points*self.score_scale)
```

Chúng ta xác định tỷ lệ tại các điểm tăng lên đó là score_scale. Một sự gia tăng nhỏ về tốc độ (1.1) làm cho trò chơi trở nên khó khăn hơn một. Nhưng để thấy sự khác biệt đáng chú ý hơn trong việc ghi điểm, chúng ta cần thay đổi giá trị điểm của quái vật một lượng lớn hơn (1,5). Nay khi tăng tốc độ của trò chơi, chúng ta cũng tăng giá trị điểm sau mỗi lần bắn hạ. Sử dụng hàm int() để tăng giá trị điểm theo số nguyên.

Để xem giá trị sau mỗi lần bắn hạ, hãy thêm lệnh print() vào phương thức increase_speed() trong Settings:

```
def increase_speed(self):  
    --snip--  
    self.alien_points= int(self.alien_points * self.score_scale)  
    print(self.alien_points)
```

Giá trị điểm mới sẽ xuất hiện mỗi khi ta đạt đến một cấp độ mới.

Lưu ý : Hãy nhớ xóa lệnh print() sau khi xác minh rằng giá trị điểm đang tăng lên, nếu không nó có thể ảnh hưởng đến hiệu suất trò chơi của ta và khiến người chơi mất tập trung.

g. Làm tròn điểm

Hầu hết các trò chơi bắn súng đều cho điểm số là bội của 10. Ngoài ra, định dạng điểm số bao gồm cả dấu phân cách bằng là dấu phẩy với số điểm lớn. Chúng ta sẽ thực hiện thay đổi này trong Scoreboard:

```
def prep_score(self):  
    """Turn the score into a rendered image."""  
    rounded_score = round(self.stats.score, -1)  
    score_str = "{:,}".format(rounded_score)  
    self.score_image = self.font.render(score_str, True,  
                                         self.text_color, self.settings.bg_color)  
--snip--
```

Hàm round() thường làm tròn một số thập phân đến một số vị trí thập phân đã cho làm đôi số thứ hai. Tuy nhiên, khi ta chuyển một số âm làm đôi số thứ hai, round() sẽ làm tròn giá trị thành 10, 100, 1000 gần nhất, v.v.

Bây giờ khi ta chạy trò chơi, ta sẽ thấy điểm được làm tròn, được định dạng gọn gàng ngay cả khi ta xếp nhiều điểm, như thể hiện trong Hình 14 -3.



Hình 14-3

h. Điểm cao

Chúng ta sẽ lưu trữ điểm số cao trong Game Stats:

```
def __init__(self, ai_game):  
    --snip--  
    # High score should never be reset.  
    self.high_score = 0
```

Vì điểm cao không bao giờ được đặt lại, chúng ta khởi tạo điểm cao trong `__init__()` chứ không phải trong `reset_stats()`.

Tiếp theo, chúng ta sẽ sửa đổi Bảng điểm để hiển thị điểm cao. Hãy bắt đầu với phương thức `__init__()`:

```
def __init__(self, ai_game):
    --snip--
    # Prepare the initial score images.
    self.prep_score()
    self.prep_high_score()
```

Điểm cao sẽ được hiển thị riêng biệt với điểm, vì vậy chúng ta cần một phương thức mới, `prep_high_score()`, để chuẩn bị cho hình ảnh hiện ra điểm cao.

Đây là phương thức `prep_high_score()`:

```
def prep_high_score(self):
    """Turn the high score into a rendered image."""
    high_score = round(self.stats.high_score, -1)
    high_score_str = "{:,}.".format(high_score)
    self.high_score_image = self.font.render(high_score_str,
        True, self.text_color, self.settings.bg_color)

    # Center the high score at the top of the screen.
    self.high_score_rect = self.high_score_image.get_rect()
    self.high_score_rect.centerx = self.screen_rect.centerx
    self.high_score_rect.top = self.score_rect.top
```

Chúng ta làm tròn điểm cao đến 10 gần nhất và định dạng nó bằng dấu phẩy. Sau đó, chúng ta tạo một hình ảnh điểm cao, căn giữa điểm cao theo chiều ngang và đặt thuộc tính `center` của nó để khớp với `center` của hình ảnh.

Phương thức `show_score()` hiện lấy điểm hiện tại ở trên cùng bên phải và điểm cao ở giữa trên cùng của màn hình:

```
def show_score(self):
    """Draw score to the screen."""
    self.screen.blit(self.score_image, self.score_rect)
    self.screen.blit(self.high_score_image, self.high_score_rect)
```

Để kiểm tra điểm cao, chúng ta sẽ viết một phương pháp mới, `check_high_score()`, trong Scoreboard :

```
def check_high_score(self):
    """Check to see if there's a new high score."""
    if self.stats.score > self.stats.high_score:
```

```

self.stats.high_score = self.stats.score
self.prep_high_score()

```

Phương thức `check_high_score()` kiểm tra điểm hiện tại so với điểm cao. Nếu điểm hiện tại lớn hơn, chúng ta cập nhật giá trị của `high_score` và gọi `prep_high_score()` để cập nhật hình ảnh của điểm cao.

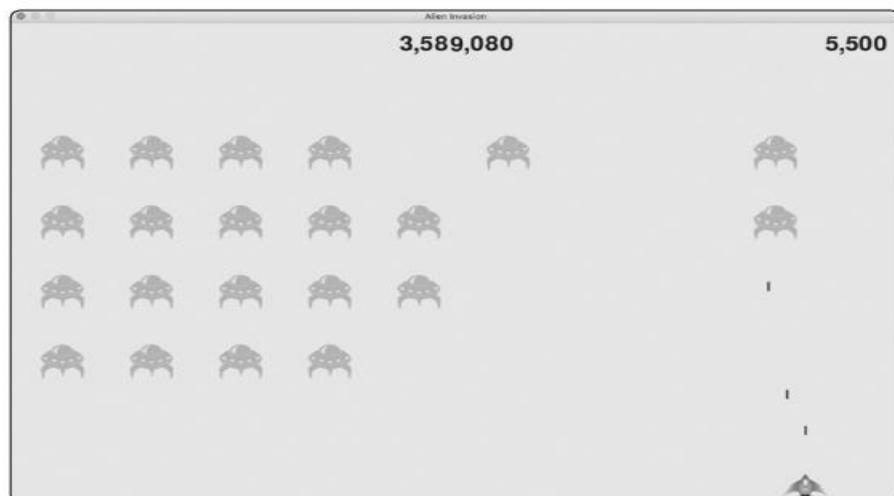
Sau khi cập nhật điểm trong `_check_bullet_alien_collisions()` chúng ta cần gọi `check_high_score()` mỗi khi một quái vật bị bắn hạ :

```

def _check_bullet_alien_collisions(self):
    --snip--
    if collisions:
        for aliens in collisions.values():
            self.stats.score += self.settings.alien_points *
len(aliens)
            self.sb.prep_score()
        self.sb.check_high_score()
    --snip--

```

Lần đầu tiên ta chơi Alien Invasion, điểm của ta sẽ là điểm cao, vì vậy nó sẽ được hiển thị dưới dạng điểm hiện tại và điểm cao. Nhưng khi ta bắt đầu ván thứ hai, điểm cao của ta sẽ xuất hiện ở giữa và điểm hiện tại của ta ở bên phải, như trong Hình 10.11.



Hình 10.11. Điểm cao (high score)

i. Hiển thị cấp độ

Để hiển thị cấp độ của người chơi trong trò chơi, trước tiên chúng ta cần một thuộc tính trong GameStats đại diện cho cấp độ hiện tại. Để đặt lại cấp độ khi bắt đầu mỗi trò chơi mới, hãy khởi tạo cấp độ đó trong `reset_stats()`:

```

def reset_stats(self):
    """Initialize statistics that can change during the game."""
    self.ships_left = self.settings.ship_limit
    self.score = 0
self.level = 1

```

Để Bảng điểm hiển thị cấp độ hiện tại, chúng ta gọi một phương thức mới, prep_level(), từ __init__():

```

def __init__(self, ai_game):
    --snip--
    self.prep_high_score()
    self.prep_level()

Đây là prep_level():

def prep_level(self):
    """Turn the level into a rendered image."""

    level_str = str(self.stats.level)
    self.level_image = self.font.render(level_str, True,
                                         self.text_color, self.settings.bg_color)

    # Position the level below the score.
    self.level_rect = self.level_image.get_rect()
    self.level_rect.right = self.score_rect.right
    self.level_rect.top = self.score_rect.bottom + 10

```

Phương thức prep_level() tạo một hình ảnh từ giá trị được lưu trữ trong stats.level và đặt thuộc tính right của hình ảnh để khớp với thuộc tính right của điểm. Sau đó, hình ảnh này được đặt dưới hình ảnh điểm khoảng 10 pixels để có khoảng trống giữa điểm số và mức.

Chúng ta cũng cần cập nhật show_score():

```

def show_score(self):
    """Draw scores and level to the screen."""

    self.screen.blit(self.score_image, self.score_rect)
    self.screen.blit(self.high_score_image, self.high_score_rect)
self.screen.blit(self.level_image, self.level_rect)

```

Chúng tôi sẽ tăng stats.level và cập nhật hình ảnh cấp trong _check_bullet_alien_collisions():

```

def _check_bullet_alien_collisions(self):
    --snip--
    if not self.aliens:
        # Destroy existing bullets and create new fleet.
        self.bullets.empty()
        self._create_fleet()

```

```

    self.settings.increase_speed()

    # Increase level.
    self.stats.level += 1
    self.sb.prep_level()

```

Nếu một hạm đội bị phá hủy, chúng ta tăng giá trị của stats.level và gọi prep_level() để đảm bảo cấp mới hiển thị chính xác.

```

def _check_play_button(self, mouse_pos):
    --snip--
    if button_clicked and not self.stats.game_active:
        --snip--
        self.sb.prep_score()
        self.sb.prep_level()
        --snip--

```

Bây giờ ta sẽ thấy mình đã hoàn thành bao nhiêu cấp độ, như được hiển thị trong Hình 10.12



Hình 10.12. Cấp độ hoàn thành

Lưu ý : Trong một số trò chơi cổ điển, điểm số có các nhãn, chẳng hạn như Điểm số, Điểm cao và Cấp độ. Chúng tôi đã bỏ qua các nhãn này vì ý nghĩa của mỗi số trở nên rõ ràng sau khi ta chơi trò chơi. Để có các nhãn này, hãy thêm chúng vào chuỗi điểm ngay trước lệnh gọi đến font.render() trong Bảng điểm.

j. Hiển thị số lượng tàu

Cuối cùng, hãy hiển thị số lượng tàu mà người chơi đã sử dụng, nhưng lần này, hãy sử dụng đồ họa. Để làm như vậy, chúng ta sẽ vẽ các con tàu ở góc trên bên

trái của màn hình để biểu thị số lượng tàu còn lại, giống như nhiều trò chơi cổ điển khác. Đầu tiên, chúng ta cần cho class Ship kế thừa Sprite để có thể tạo một nhóm tàu:

```
import pygame
from pygame.sprite import Sprite
class Ship(Sprite):
    """A class to manage the ship."""

    def __init__(self, ai_game):
        """Initialize the ship and set its starting position."""
        super().__init__()
        --snip--
```

Ở đây chúng ta import Sprite, đảm bảo Ship kế thừa từ Sprite và gọi super() ở đầu __init__().

Tiếp theo, chúng ta cần sửa đổi Scoreboard để tạo một nhóm tàu. Dưới đây là các câu lệnh nhập cho Scoreboard:

```
import pygame.font
from pygame.sprite import Group
from ship import Ship
Đây là __init__():
def __init__(self, ai_game):
    """Initialize scorekeeping attributes."""
    self.ai_game = ai_game
    self.screen = ai_game.screen
    --snip--
    self.prep_level()
    self.prep_ships()
```

Chúng ta gán phiên bản trò chơi này cho một thuộc tính, bởi vì sẽ cần nó để tạo một số tàu. Gọi prep_ships() sau lệnh gọi prep_level().

Đây là prep_ships():

```
def prep_ships(self):
    """Show how many ships are left."""
    self.ships = Group()
    for ship_number in range(self.stats.ships_left):
        ship = Ship(self.ai_game)
        ship.rect.x = 10 + ship_number * ship.rect.width
        ship.rect.y = 10
        self.ships.add(ship)
```

Phương thức prep_ships() tạo một nhóm trống self.ships để giữ các trường hợp của tàu. Để lấp đầy nhóm này, một vòng lặp chạy qua mỗi con tàu mà người chơi đã sử dụng. Trong vòng lặp, chúng ta tạo một con tàu mới và đặt giá trị tọa độ x cho mỗi tàu để các chúng xuất hiện cạnh nhau cách nhau 10 pixel về phía bên trái của nhóm tàu. Đồng thời đặt cách đầu màn hình giá trị tọa độ y = 10 pixel để các con tàu xuất hiện ở góc trên bên trái của màn hình. Sau đó, chúng ta thêm mỗi tàu mới vào nhóm tàu.

Bây giờ chúng ta cần vẽ các con tàu lên màn hình:

```
def show_score(self):  
    """Draw scores, level, and ships to the screen."""  
    self.screen.blit(self.score_image, self.score_rect)  
    self.screen.blit(self.high_score_image, self.high_score_rect)  
    self.screen.blit(self.level_image, self.level_rect)  
    self.ships.draw(self.screen)
```

Để hiển thị các tàu trên màn hình, ta gọi draw() và Pygame vẽ ra từng con tàu. Để cho người chơi biết họ bắt đầu bằng bao nhiêu tàu, chúng ta gọi prep_ships() khi một trò chơi mới bắt đầu. Thực hiện việc này trong _check_play_button() trong AlienInvasion:

```
def _check_play_button(self, mouse_pos):  
    --snip--  
    if button_clicked and not self.stats.game_active:  
        --snip--  
        self.sb.prep_score()  
        self.sb.prep_level()  
        self.sb.prep_ships()  
        --snip--
```

Chúng ta cũng gọi prep_ships() khi tàu bị phá hủy để cập nhật hiển thị hình ảnh tàu nhằm thông báo người chơi đã mất một tàu:

```
def _ship_hit(self):  
    """Respond to ship being hit by alien."""  
    if self.stats.ships_left > 0:  
        # Decrement ships_left, and update scoreboard.  
        self.stats.ships_left -= 1  
        self.sb.prep_ships()  
        --snip--
```

Chúng ta gọi prep_ships() sau khi giảm giá trị của ship_left, hiển thị chính xác số tàu mỗi khi tàu bị phá hủy. Hình 10.13 cho thấy hệ thống tính điểm hoàn chỉnh :



Hình 10.13. Hệ thống tính điểm hoàn chỉnh

Thử thêm

Điểm cao nhất: Điểm cao được đặt lại mỗi khi người chơi đóng và khởi động lại trò chơi. Khắc phục điều này bằng cách ghi điểm cao vào tệp trước khi gọi sys.exit() và đọc điểm cao khi khởi tạo giá trị của nó trong GameStats.

Tái cấu trúc: Tìm kiếm các phương thức đang thực hiện nhiều nhiệm vụ và cấu trúc lại chúng để tổ chức mã của ta và làm cho nó hoạt động hiệu quả. Ví dụ: di chuyển một số mã trong _check_bullet_alien_collisions(), bắt đầu một cấp mới khi hạm đội quái vật đã bị tiêu diệt, đến một hàm có tên start_new_level(). Ngoài ra, hãy chuyển bốn lệnh gọi phương thức riêng biệt trong phương thức __init__() trong ScoreBoard sang một phương thức có tên là prep_images() để rút gọn __init__(). Phương thức prep_images() cũng có thể giúp simple_check_play_button() hoặc start_game() nếu ta đã cấu trúc lại _check_play_button().

Lưu ý : Trước khi cố gắng cấu trúc lại dự án, hãy xem Phụ lục D để tìm hiểu cách khôi phục dự án về trạng thái hoạt động nếu ta bị lỗi trong quá trình tái cấu trúc.

Mở rộng trò chơi: Hãy nghĩ cách để mở rộng Cuộc xâm lược của quái vật. Ví dụ, ta có thể lập trình cho quái vật bắn đạn xuống tàu hoặc thêm lá chắn cho tàu,

có thể tàu bị tiêu diệt bởi đạn từ hai bên. Hoặc sử dụng thứ gì đó như mô-đun pygame.mixer để thêm hiệu ứng âm thanh như tiếng nổ và tiếng bắn.

Sideways Shooter, Phiên bản cuối cùng: Tiếp tục phát triển Sideways Shooter, sử dụng mọi thứ chúng ta đã làm trong dự án này. Thêm nút Play, làm cho trò chơi tăng tốc ở những điểm thích hợp và phát triển hệ thống tính điểm. Đảm bảo khi cấu trúc lại mã, game của ta sẽ hiệu quả và tìm kiếm cơ hội để tùy chỉnh trò chơi ngoài những gì được trình bày trong chương này.

Tiêu kết

Trong phần này, ta đã học cách triển khai nút Play để bắt đầu một trò chơi mới, phát hiện các sự kiện chuột và ấn con trỏ trong các trò chơi đang hoạt động. Ta có thể sử dụng những gì đã học để tạo các nút khác trong trò chơi của mình, chẳng hạn như nút Help để hiển thị hướng dẫn về cách chơi. Chúng ta cũng đã học cách sửa đổi tốc độ của một trò chơi khi nó diễn ra, triển khai hệ thống tính điểm và hiển thị thông tin theo cách văn bản hoặc không theo văn bản.

10.2. Trực quan hóa dữ liệu

Trực quan hóa dữ liệu liên quan đến việc khám phá dữ liệu thông qua các hình ảnh biểu diễn trực quan. Nó kết hợp với phân tích dữ liệu, sử dụng mã để khám phá các mẫu và kết nối trong bộ dữ liệu. Một tập dữ liệu có thể được tạo thành từ một danh sách nhỏ các số nằm trong một dòng mã hoặc nó có thể là nhiều gigabyte dữ liệu. Tạo biểu diễn dữ liệu đẹp không chỉ là hình ảnh đẹp. Khi biểu diễn của một tập dữ liệu đơn giản và hấp dẫn về mặt hình ảnh, ý nghĩa của nó trở nên rõ ràng đối với người xem. Mọi người sẽ thấy các mẫu và mức độ quan trọng trong tập dữ liệu mà họ chưa bao giờ biết là đã tồn tại. May mắn là chúng ta không cần siêu máy tính để trực quan hóa dữ liệu phức tạp. Với tính hiệu quả của Python, ta có thể nhanh chóng khám phá các tập dữ liệu được tạo ra từ hàng triệu các điểm dữ liệu riêng lẻ chỉ trên một máy tính xách tay. Ngoài ra, các điểm dữ liệu không cần phải là những con số. Với những kiến thức cơ bản ta đã học trong phần đầu tiên của cuốn sách này, ta cũng có thể phân tích dữ liệu phi số.

10.2.1. Sản sinh dữ liệu

Mọi người sử dụng Python cho các công việc chuyên sâu về di truyền học, nghiên cứu khí hậu, phân tích chính trị và kinh tế, và nhiều lĩnh vực khác. Các nhà khoa học dữ liệu đã viết một loạt các công cụ trực quan hóa và phân tích ấn tượng bằng Python, nhiều cái khả dụng cho chúng ta. Một trong những công cụ phổ biến nhất là Matplotlib, một thư viện vẽ đồ thị toán học. Chúng ta sẽ sử dụng Matplotlib để tạo các sơ đồ đơn giản, chẳng hạn như biểu đồ đường và biểu đồ phân tán. Sau đó, chúng ta sẽ tạo ra một tập dữ liệu dựa trên khái niệm về ngẫu nhiên, và trực quan hóa các dữ liệu ngẫu nhiên đó.

Chúng ta cũng sẽ sử dụng một gói có tên là Plotly, gói này tạo ra các hình ảnh hóa hoạt động tốt trên các thiết bị kỹ thuật số. Plotly tạo hình ảnh trực quan tự động thay đổi kích thước theo nhiều loại thiết bị hiển thị. Những hình ảnh hóa này cũng có thể bao gồm một số tính năng tương tác, chẳng hạn như nhấn mạnh các khía cạnh của tập dữ liệu khi người dùng di chuột qua các phần khác nhau của hình ảnh trực quan. Chúng ta sẽ sử dụng Plotly để phân tích kết quả của việc tung xúc xắc.

10.2.1.1. Cài đặt Matplotlib

Để sử dụng Matplotlib cho bộ hình ảnh hóa ban đầu, ta cần cài đặt nó sử dụng pip, một mô-đun tải xuống và cài đặt các gói Python. Thực hiện lệnh sau tại dấu nhắc đầu cuối:

```
$ python -m pip install --user matplotlib
```

Lệnh này yêu cầu Python chạy mô-đun pip và cài đặt gói matplotlib cho cài đặt Python của người dùng hiện tại. Nếu sử dụng một lệnh khác ngoài python trên hệ thống để chạy các chương trình hoặc bắt đầu một phiên đầu cuối, chẳng hạn như python3, lệnh cần gõ sẽ giống như sau:

```
$ python3 -m pip install --user matplotlib
```

Để xem các loại hình ảnh trực quan ta có thể thực hiện với Matplotlib, hãy truy cập thư viện mẫu tại <https://matplotlib.org/gallery/>. Khi nhấp vào hình ảnh trực quan trong thư viện, ta sẽ thấy mã được sử dụng để tạo các đồ thị.

10.2.1.2. Vẽ biểu đồ một đường đơn giản

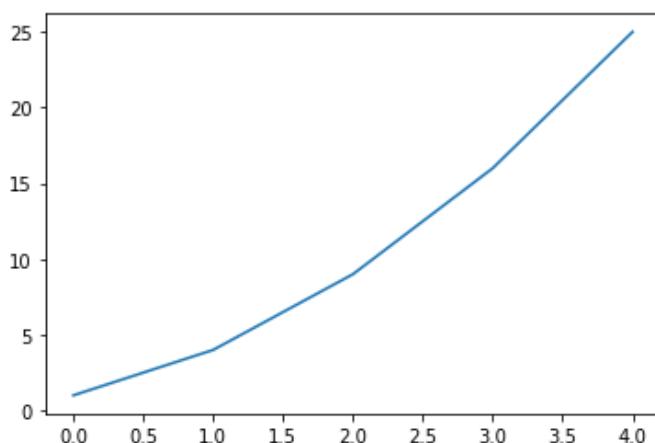
Hãy vẽ một biểu đồ đường đơn giản bằng Matplotlib, sau đó tùy chỉnh nó thành hình ảnh dữ liệu nhiều thông tin hơn. Chúng ta sẽ sử dụng số bình phương

chuỗi 1, 4, 9, 16, 25 làm dữ liệu cho biểu đồ. Chỉ cần cung cấp cho Matplotlib các số, như được hiển thị ở dưới đây, và Matplotlib sẽ làm phần còn lại:

```
import matplotlib.pyplot as plt
squares = [1, 4, 9, 16, 25]
① fig, ax = plt.subplots()
ax.plot(squares)
plt.show()
```

Trước tiên, chúng ta nhập mô-đun pyplot bằng bí danh plt để không phải gõ pyplot nhiều lần. Module pyplot chứa một số chức năng tạo biểu đồ và đồ thị. Chúng ta tạo một danh sách được gọi là squares để chứa dữ liệu mà chúng ta sẽ vẽ. Sau đó chúng ta tuân theo một quy ước Matplotlib phổ biến khác bằng cách gọi hàm subplots() ①. Hàm này có thể tạo ra một hoặc nhiều biểu đồ trong cùng một hình. Biến fig đại diện toàn bộ hình vẽ hoặc tập các biểu đồ được sinh ra. Biến ax đại diện cho một biểu đồ đơn trong hình và biến đó sẽ được sử dụng rất nhiều.

Sau đó chúng ta sử dụng phương thức plot() để vẽ biểu đồ cho dữ liệu được cung cấp theo một cách có nghĩa. Hàm plt.show() sẽ mở cửa sổ hiển thị Matplotlib và hiển thị biểu đồ như hình dưới đây. Cửa sổ hiển thị cho phép chúng ta phóng to biểu đồ và di chuyển các biểu đồ. Khi nhấp vào biểu tượng đĩa mềm, ta có thể lưu hình ảnh của đồ thị.



Hình 10.14. Biểu đồ đầu tiên

Thay đổi loại nhãn và độ dày đường line

Mặc dù biểu đồ trên cho thấy rằng các con số đang tăng lên, loại nhãn quá nhỏ và dòng hơi mỏng để dễ đọc. Rất may là Matplotlib cho phép ta điều chỉnh mọi đặc điểm của hình ảnh trực quan.

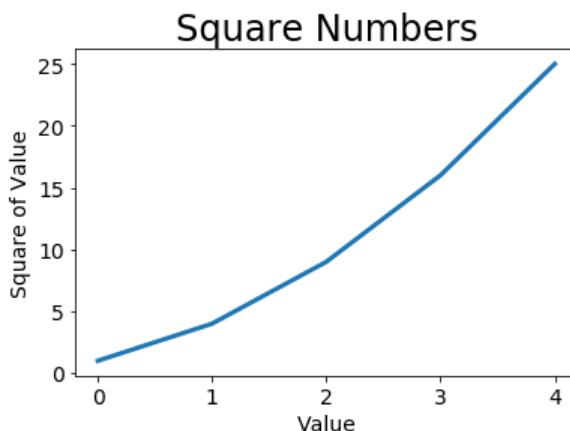
Chúng ta sẽ sử dụng một số tùy chỉnh có sẵn để cải thiện khả năng đọc của đồ thị này, như được hiển thị ở đây:

```
import matplotlib.pyplot as plt
squares = [1, 4, 9, 16, 25]
fig, ax = plt.subplots()
① ax.plot(squares, linewidth=3)
# Set chart title and label axes.
② ax.set_title("Square Numbers", fontsize=24)
③ ax.set_xlabel("Value", fontsize=14)
ax.set_ylabel("Square of Value", fontsize=14)
# Set size of tick labels.
④ ax.tick_params(axis='both', labelsize=14)
plt.show()
```

Tham số newline tại ① điều khiển độ dày của dòng do hàm plot() sinh ra. Phương thức set_title() tại ② thiết lập tiêu đề cho biểu đồ. Tham số fontsize xuất hiện tiếp theo điều khiển cỡ chữ của các thành phần trong biểu đồ.

Phương thức set_xlabel() và set_ylabel() cho phép ta đặt tiêu đề cho mỗi trục ③ và phương thức tick_params() định kiểu các dấu tích ④. Các đối số được hiển thị ở đây ảnh hưởng đến các đánh dấu trên cả trục x và trục y (axis = 'both') và đặt kích thước phông chữ của nhãn theo trục x và y thành 14 (labelsize = 14).

Như ta có thể thấy trong hình dưới, biểu đồ kết quả dễ đọc hơn nhiều. Loại nhãn lớn hơn và biểu đồ đường dày hơn. Ta nên thử nghiệm với những giá trị này để có ý tưởng về hiển thị đẹp nhất trong biểu đồ kết quả.



Hình 10.15. Bổ sung Tiêu đề và nhãn

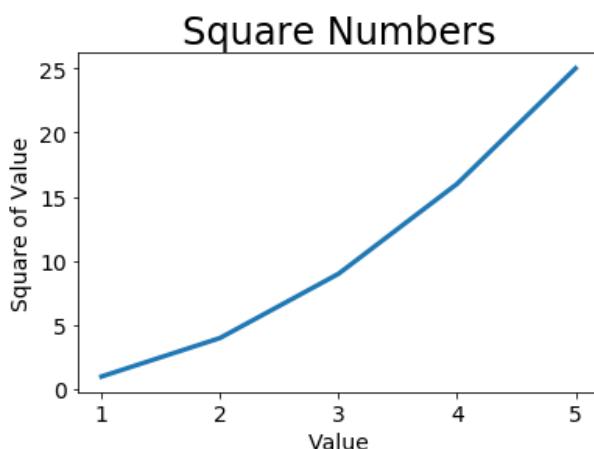
Chỉnh sửa các điểm

Nhưng bây giờ chúng ta có thể đọc biểu đồ tốt hơn, chúng ta thấy rằng dữ liệu không được vẽ một cách chính xác. Lưu ý ở cuối biểu đồ rằng bình phương 4,0 được hiển thị là 25! Ta cần khắc phục điều đó.

Khi ta cung cấp cho `plot()` một dãy số, nó sẽ giả định là dữ liệu đầu tiên điểm tương ứng với giá trị tọa độ x là 0, nhưng điểm đầu tiên của chúng ta tương ứng với giá trị x là 1. Chúng ta có thể ghi đè hành vi mặc định bằng cách cho `plot()` các giá trị đầu vào và đầu ra được sử dụng để tính toán các bình phương:

```
import matplotlib.pyplot as plt
input_values = [1, 2, 3, 4, 5]
squares = [1, 4, 9, 16, 25]
fig, ax = plt.subplots()
ax.plot(input_values, squares, linewidth=3)
# Set chart title and label axes.
--snip--
```

Bây giờ, `plot()` sẽ vẽ biểu đồ dữ liệu một cách chính xác vì chúng ta đã cung cấp giá trị đầu vào và đầu ra, vì vậy không cần phải giả định cách các số đầu ra được tạo. Biểu đồ kết quả, thể hiện trong hình 10.16, là chính xác.



Hình 10.16. Điều chỉnh trục tọa độ X

Chúng ta có thể chỉ định nhiều đối số khi sử dụng `plot()` và sử dụng một số hàm để tùy chỉnh các biểu đồ được vẽ ra. Chúng ta sẽ tiếp tục khám phá những chức năng tùy chỉnh khi làm việc với các bộ dữ liệu thú vị hơn trong suốt phần này.

Sử dụng các định kiểu có sẵn (Build-in styles)

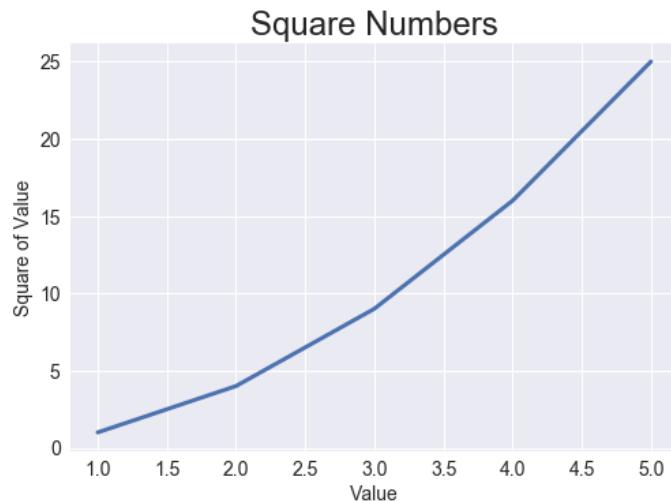
Matplotlib có sẵn một số kiểu định sẵn, với khởi đầu mặc định thiết lập cho màu nền, đường lưới, độ rộng dòng, phông chữ, kích thước phông chữ và các giá trị khác sẽ làm cho hình ảnh hấp dẫn mà không đòi hỏi nhiều sự tùy biến.

```
>>> import matplotlib.pyplot as plt  
>>> plt.style.available  
['seaborn-dark', 'seaborn-darkgrid', 'seaborn-ticks', 'fivethirtyeight',  
--snip--
```

Để sử dụng bất kỳ kiểu nào trong số này, ta thêm một dòng mã trước khi bắt đầu tạo biểu đồ:

```
import matplotlib.pyplot as plt  
input_values = [1, 2, 3, 4, 5]  
squares = [1, 4, 9, 16, 25]  
plt.style.use('seaborn')  
fig, ax = plt.subplots()  
--snip--
```

Mã này tạo ra biểu đồ được hiển thị trong hình dưới. Với nhiều loại phong cách có sẵn; ta có thể thử các phong cách này để tìm một số kiểu ưa thích.



Hình 10.17. Định kiểu đồ thi

Vẽ biểu đồ và định kiểu riêng các điểm với scatter()

Đôi khi, rất hữu ích khi lập biểu đồ và tạo kiểu cho các điểm riêng lẻ dựa trên các đặc điểm nhất định. Ví dụ: chúng ta có thể vẽ biểu đồ các giá trị nhỏ bằng một màu và các giá trị lớn hơn bằng một màu khác. Ta cũng có thể vẽ một tập dữ liệu lớn với một tập hợp các tùy chọn tạo kiểu và sau đó nhấn mạnh các điểm riêng lẻ bằng cách vẽ lại chúng bằng các tùy chọn khác nhau.

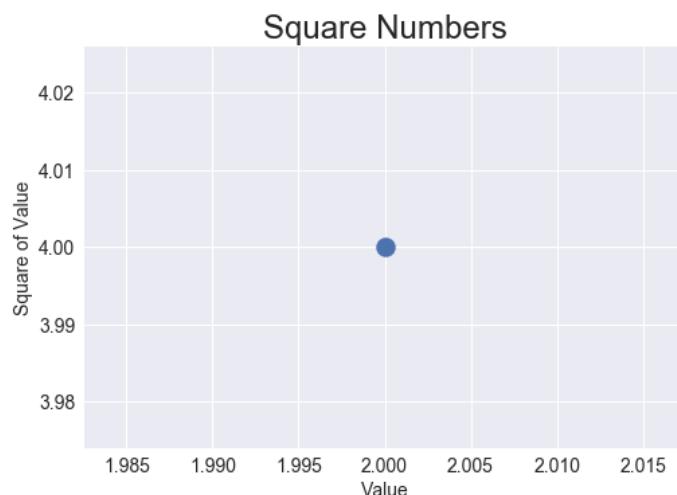
Để vẽ một điểm duy nhất, ta sử dụng phương thức scatter(). Truyền các giá trị đơn lẻ (x, y) của điểm mong muốn vào scatter() để vẽ biểu đồ các giá trị đó:

```
import matplotlib.pyplot as plt
plt.style.use('seaborn')
fig, ax = plt.subplots()
ax.scatter(2, 4)
plt.show()
```

Ta tạo kiểu đầu ra để làm cho điểm trên đồ thị thú vị hơn. Chúng ta sẽ thêm tiêu đề, nhãn các trục và đảm bảo rằng tất cả văn bản đủ lớn để đọc:

```
import matplotlib.pyplot as plt
plt.style.use('seaborn')
fig, ax = plt.subplots()
ax.scatter(2, 4, s=200)
# Set chart title and label axes.
ax.set_title("Square Numbers", fontsize=24)
ax.set_xlabel("Value", fontsize=14)
ax.set_ylabel("Square of Value", fontsize=14)
# Set size of tick labels.
ax.tick_params(axis='both', which='major', labelsize=14)
plt.show()
```

Tại dòng in đậm, chúng ta gọi phương thức scatter() và sử dụng tham số để thiết lập kích cỡ của điểm được sử dụng để vẽ biểu đồ. Khi chạy code, chúng ta sẽ thấy một điểm đơn ở giữa biểu đồ như hình dưới:



Hình 10.18. Scatter plot

Vẽ tập điểm với scatter()

Để vẽ một chuỗi các điểm, chúng ta có thể chuyển các danh sách riêng biệt của scatter() gồm các giá trị x và y, như sau:

```

import matplotlib.pyplot as plt

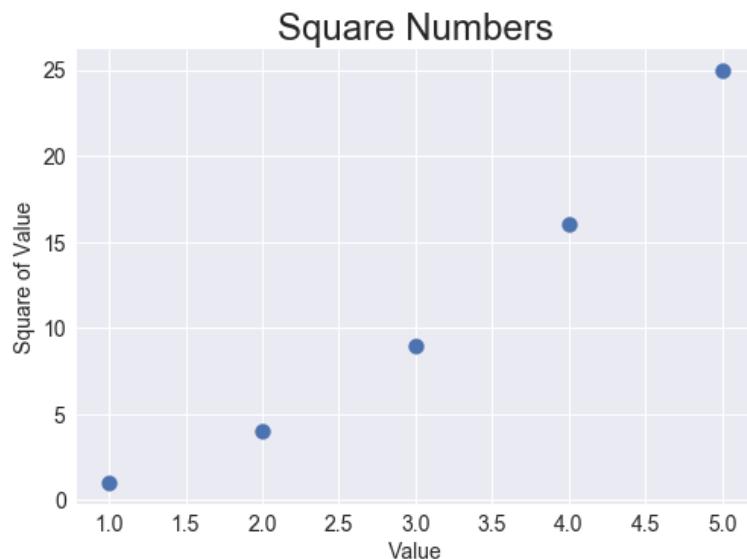
x_values = [1, 2, 3, 4, 5]
y_values = [1, 4, 9, 16, 25]

plt.style.use('seaborn')
fig, ax = plt.subplots()
ax.scatter(x_values, y_values, s=100)

# Set chart title and label axes.
--snip--

```

Danh sách `x_values` chứa các số được bình phương và `y_values` chứa bình phương của mỗi số. Khi các danh sách này được chuyển đến `scatter()`, Matplotlib đọc một giá trị từ mỗi danh sách khi nó vẽ từng điểm. Các điểm được vẽ là (1, 1), (2, 4), (3, 9), (4, 16), và (5, 25); Hình dưới cho thấy kết quả.



Hình 10.19. Scatter plot với danh sách điểm

Tính toán dữ liệu tự động

Viết danh sách bằng tay có thể không hiệu quả, đặc biệt là khi chúng ta có nhiều điểm. Thay vì chuyển điểm trong danh sách, ta sử dụng một vòng lặp trong Python để thực hiện các phép tính.

```

import matplotlib.pyplot as plt
x_values = range(1, 1001)
y_values = [x**2 for x in x_values]
plt.style.use('seaborn')
fig, ax = plt.subplots()
ax.scatter(x_values, y_values, s=10)

```

```

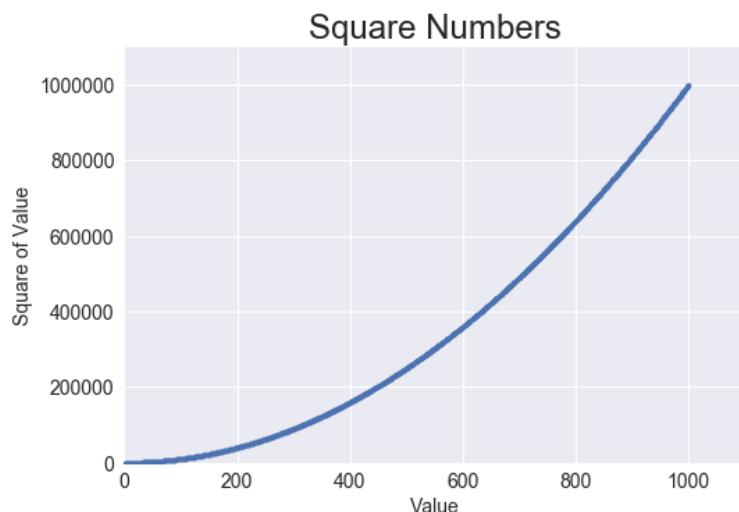
# Set chart title and label axes.
--snip--
# Set the range for each axis.
ax.axis([0, 1100, 0, 1100000])
plt.show()

```

Đầu tiên, chúng ta thiết lập một danh sách từ 1 tới 1000 sử dụng hàm range() và gán nó cho danh sách x_values. Tương ứng với danh sách vừa tạo, ta tạo ra danh sách y bằng cách lặp qua danh sách x_values và tính giá trị bình phương của mỗi giá trị x, sau đó gán cho biến y_values.

Tại phần mã bôi đậm thứ hai, chúng ta sử dụng phương thức scatter() và truyền vào đó danh sách x_values và y_values. Do có nhiều điểm nên kích cỡ điểm được thiết lập giá trị nhỏ là 10.

Phần mã bôi đậm thứ ba, chúng ta sử dụng phương thức axis() để xác định khoảng của mỗi trục. Phương thức axis() yêu cầu bốn giá trị: giá trị nhỏ nhất và giá trị lớn nhất cho trục x và trục y. Ở đây, chúng ta chạy trục x từ 0 đến 1100 và trục y từ 0 đến 1.100.000. Hình 10.20 cho thấy kết quả.



Hình 10.20. Vẽ 1000 điểm trên đồ thị

Tự định nghĩa màu

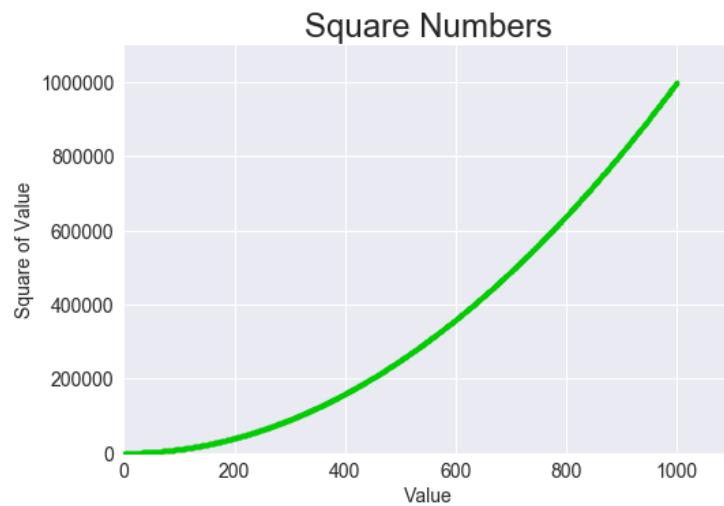
Để thay đổi màu của các điểm, hãy chuyển c đến scatter() với tên của một màu để sử dụng trong dấu ngoặc kép, như được hiển thị ở đây:

```
ax.scatter(x_values, y_values, c='red', s=10)
```

Ta cũng có thể xác định các màu tùy chỉnh bằng cách sử dụng mô hình màu RGB. Định nghĩa một màu, chuyển đổi số c một bộ giá trị với ba giá trị thập phân

(mỗi giá trị một cho màu đỏ, xanh lục và xanh lam theo thứ tự đó), sử dụng các giá trị từ 0 đến 1. Ví dụ, dòng sau tạo ra một biểu đồ với các chấm màu xanh lục nhạt:

```
ax.scatter(x_values, y_values, c=(0, 0.8, 0), s=10)
```



Hình 10.21. Tự định nghĩa màu

Các giá trị gần 0 tạo ra màu tối và các giá trị gần 1 tạo ra màu sáng hơn.

Sử dụng bản đồ màu

Bản đồ màu là một loạt các màu trong một gradient di chuyển từ điểm bắt đầu đến một màu kết thúc. Chúng sử dụng các bản đồ màu trong hình ảnh hóa để nhấn mạnh một mẫu trong dữ liệu. Ví dụ: ta có thể đặt các giá trị thấp thành màu sáng và giá trị cao với một màu tối hơn. Module pyplot bao gồm một tập hợp các bản đồ màu cài sẵn. Để sử dụng một trong những các bản đồ màu này, ta cần chỉ định cách pyplot chỉ định màu cho mỗi điểm trong tập dữ liệu. Đây là cách chỉ định mỗi điểm một màu dựa trên trên yvalue của nó:

```
import matplotlib.pyplot as plt

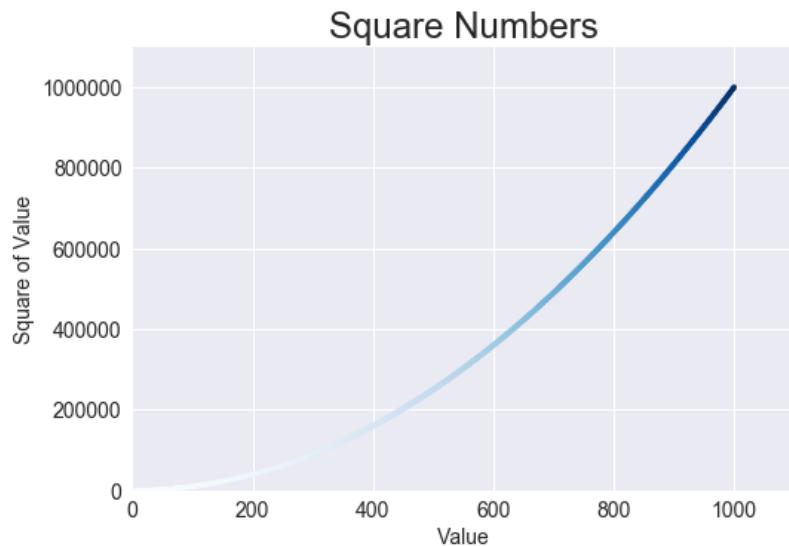
x_values = range(1, 1001)
y_values = [x**2 for x in x_values]

ax.scatter(x_values, y_values, c=y_values, cmap=plt.cm.Blues, s=10)

# Set chart title and label axes.
--snip--
```

Chúng ta chuyển danh sách các giá trị y cho c, sau đó cho pyplot biết bản đồ màu nào sử dụng bằng cách sử dụng đối số cmap. Mã này tô màu các điểm có giá trị

thấp hơn màu xanh lam nhạt và tô điểm các điểm có giá trị cao hơn màu xanh lam đậm. Hình 10.22 hiển thị biểu đồ kết quả.



Hình 10.22. Mầu gradient

Lưu biểu đồ tự động

Nếu muốn chương trình tự động lưu biểu đồ vào một tệp, ta có thể thay thế lệnh gọi plt.show() bằng lệnh gọi plt.savefig():

```
plt.savefig('squares_plot.png', bbox_inches='tight')
```

Đối số đầu tiên là tên tệp cho hình ảnh biểu đồ, tên tệp này sẽ được lưu trong cùng một thư mục với file mã nguồn đang thực thi. Đối số thứ hai cắt bớt khoảng trắng từ biểu đồ. Nếu ta muốn có thêm khoảng trắng xung quanh biểu đồ, chỉ cần bỏ qua đối số này.

10.2.1.3. Bước ngẫu nhiên

Trong phần này, chúng ta sẽ sử dụng Python để tạo dữ liệu cho một chuyến đi ngẫu nhiên và sau đó sử dụng Matplotlib để tạo ra một bản trình bày dữ liệu trực quan hấp dẫn. Bước đi ngẫu nhiên là con đường không có phương hướng rõ ràng nhưng được xác định bởi một loạt các quyết định ngẫu nhiên, mỗi quyết định trong số đó hoàn toàn phó mặc cho sự may rủi. Ta có thể tưởng tượng một cuộc đi bộ ngẫu nhiên giống như con đường mà một con kiến bối rối sẽ đi nếu nó đi từng bước theo một hướng ngẫu nhiên.

Bước ngẫu nhiên có ứng dụng thực tế trong tự nhiên, vật lý, sinh học, hóa học và kinh tế. Ví dụ, một hạt phấn hoa ăn vào giọt nước di chuyển trên bề mặt nước bởi

vì nó liên tục bị đẩy xung quanh bởi các phân tử nước. Chuyển động phân tử trong giọt nước là ngẫu nhiên, vì vậy con đường mà một hạt phấn đi trên bề mặt là một con đường ngẫu nhiên. Đoạn mã chúng ta sẽ viết tiếp theo mô hình hóa nhiều tình huống trong thế giới thực.

Tạo lớp RandomWalk

Để tạo một cuộc đi bộ ngẫu nhiên, chúng ta sẽ tạo một lớp RandomWalk, lớp này sẽ đưa ra các quyết định ngẫu nhiên về hướng đi bộ sẽ đi. Lớp cần ba thuộc tính: một biến để lưu trữ số điểm trong đi bộ và hai danh sách để lưu trữ các giá trị x và y là tọa độ của các điểm.

Chúng ta sẽ chỉ cần hai phương thức cho lớp RandomWalk: `__init__()` và `fill_walk()`, sẽ tính toán các điểm trong bước đi. Ta bắt đầu bằng `__init__()` như được hiển thị ở đây:

```
①  from random import choice
class RandomWalk:
    """A class to generate random walks."""
②      def __init__(self, num_points=5000):
        """Initialize attributes of a walk."""
        self.num_points = num_points
        # All walks start at (0, 0).
③          self.x_values = [0]
        self.y_values = [0]
```

10.2.2. Tải dữ liệu về

Trong phần này, ta sẽ tải xuống các tập dữ liệu từ các nguồn trực tuyến và trực quan hóa hoạt động của dữ liệu đó. Ta có thể tìm thấy một dữ liệu trực tuyến đa dạng đáng kinh ngạc, phần lớn trong số đó chưa được khai phá. Khả năng phân tích dữ liệu này cho phép ta khám phá các mẫu và kết nối mà chưa ai tìm thấy. Chúng ta sẽ truy cập và trực quan hóa dữ liệu được lưu trữ ở hai định dạng dữ liệu phổ biến, CSV và JSON. Chúng ta sẽ sử dụng mô-đun csv của Python để xử lý dữ liệu thời tiết được lưu trữ trong định dạng CSV (các giá trị được phân tách bằng dấu phẩy) và phân tích nhiệt độ cao và thấp theo thời gian ở hai vị trí khác nhau. Sau đó, chúng ta sẽ sử dụng Matplotlib để tạo biểu đồ dựa trên dữ liệu đã tải xuống của chúng ta để hiển thị các biến thể về nhiệt độ trong hai môi trường khác nhau: Sitka, Alaska và Thung lũng Chết, California. Ở phần sau của chương, chúng ta sẽ sử dụng mô-đun json để truy

cập dữ liệu động đất được lưu trữ ở định dạng JSON và sử dụng Plotly để vẽ bản đồ thế giới hiển thị vị trí và cường độ của các trận động đất gần đây. Đến cuối mục này, ta sẽ chuẩn bị làm việc với các loại và định dạng tập dữ liệu, đồng thời ta sẽ hiểu sâu hơn về cách để xây dựng các hình dung phức tạp. Có thể truy cập và trực quan hóa dữ liệu trực tuyến gồm nhiều loại và định dạng khác nhau là điều cần thiết để làm việc với nhiều loại tập dữ liệu trong thế giới thực.

10.2.2.1. Định dạng tệp CSV

Một cách đơn giản để lưu trữ dữ liệu trong một chuỗi văn bản là viết dữ liệu dưới dạng một chuỗi các giá trị được phân tách bằng dấu phẩy, được gọi là các giá trị được phân tách bằng dấu phẩy. Các kết quả là tệp được gọi là CSV. Ví dụ: đây là một đoạn thời tiết dữ liệu ở định dạng CSV:

```
"USW00025333","SITKA AIRPORT, AK US","2018-01-01","0.45",,"48","38"
```

Đây là phần trích dẫn một số dữ liệu thời tiết từ ngày 1 tháng 1 năm 2018 ở Sitka, Alaska. Nó bao gồm nhiệt độ cao và thấp trong ngày, cũng như một số của các phép đo khác kể từ ngày đó. Tệp CSV có thể khó đối với con người để đọc, nhưng chúng dễ dàng cho các chương trình xử lý và trích xuất các giá trị, điều này giúp tăng tốc quá trình phân tích dữ liệu.

Chúng ta sẽ bắt đầu với một tập hợp nhỏ dữ liệu thời tiết định dạng CSV được ghi lại trong Sitka, có sẵn trong tài nguyên của sách tại <https://nostarch.com/pythoncrashcourse2e/>. Tạo thư mục *data* là thư mục cùng thư mục với tệp chứa mã nguồn. Sao chép tệp *sitka_weather_07-2018_simple.csv* vào thư mục mới.

Phân tích định dạng tệp CSV

Module *csv* của Python trong thư viện chuẩn phân tích cú pháp các dòng trong một nhóm CSV và cho phép ta nhanh chóng trích xuất các giá trị được quan tâm.

Ta bắt đầu bằng cách kiểm tra dòng đầu tiên của tệp, trong đó có một loạt các tiêu đề cho dữ liệu. Những tiêu đề này cho chúng ta biết dữ liệu chứa loại thông tin nào:

```
import csv
filename = 'data/sitka_weather_07-2018_simple.csv'
① with open(filename) as f:
②     reader = csv.reader(f)
③     header_row = next(reader)
```

```
print(header_row)
```

Sau khi nhập mô-đun csv, chúng ta chỉ định tên của nhóm mà ta đang làm việc với tên tệp. Sau đó, chúng ta mở tệp và gán tệp kết quả cho đối tượng f. Tiếp đó, ta gọi phương thức csv.reader() và truyền vào phương thức tham số là đối tượng tệp f và đặt tên biến trả về của phương thức là reader.

Module csv chưa hàm next() trả về dòng tiếp theo của tệp khi ta truyền vào đối tượng reader. Phần xử lý này, ta gọi next() chỉ một lần để trả về dòng đầu tiên của tệp, dòng này chứa header của tệp. Chúng ta lưu dữ liệu header vào biến header_row. Sau đó, ta in ra header_row để thấy các thông tin cần thiết của header:

```
['STATION', 'NAME', 'DATE', 'PRCP', 'TAVG', 'TMAX', 'TMIN']
```

Đối tượng reader xử lý dòng đầu tiên của các giá trị được phân tách bằng dấu phẩy trong tệp và lưu trữ từng mục dưới dạng một mục trong danh sách. Tiêu đề STATION đại diện cho mã cho trạm thời tiết đã ghi lại dữ liệu này. Vị trí của tiêu đề này cho chúng ta biết rằng giá trị đầu tiên trong mỗi dòng sẽ là mã trạm thời tiết. Tiêu đề NAME chỉ ra rằng giá trị thứ hai trong mỗi dòng là tên của trạm thời tiết đã thực hiện việc ghi lại. Phần còn lại của tiêu đề chỉ định loại thông tin nào đã được ghi lại trong mỗi lần đọc. Dữ liệu chúng ta quan tâm nhất bây giờ là ngày, nhiệt độ cao (TMAX), và nhiệt độ thấp (TMIN). Đây là một tập dữ liệu đơn giản chỉ chứa lượng mưa và dữ liệu liên quan đến nhiệt độ. Khi ta tải xuống dữ liệu thời tiết theo ý muốn, ta có thể chọn bao gồm một số phép đo khác liên quan đến tốc độ gió, hướng và dữ liệu lượng mưa chi tiết hơn.

In ra Header và vị trí của chúng

Để dễ hiểu hơn về dữ liệu tiêu đề tệp, chúng ta in từng tiêu đề và vị trí của nó trong danh sách:

```
import csv
filename = 'data/sitka_weather_07-2018_simple.csv'
with open(filename) as f:
    reader=csv.reader(f)
    header_row=next(reader)
    print(header_row)
for index, column_header in enumerate(header_row):
    print(index, column_header)
```

Hàm enumerate() trả về cả chỉ mục của từng mục và giá trị của từng mục khi ta duyệt một danh sách. Kết quả như dưới đây:

```
0 STATION
1 NAME
2 DATE
3 PRCP
4 TAVG
5 TMAX
6 TMIN
```

Ở đây, chúng ta thấy rằng ngày và nhiệt độ cao của chúng được lưu trữ trong cột 2 và 5. Để khám phá dữ liệu này, ta sẽ xử lý từng hàng dữ liệu trong sitka_weather_07-2018_simple.csv và trích xuất các giá trị với các chỉ mục 2 và 5.

Trích xuất và đọc dữ liệu

Bây giờ chúng ta biết những cột dữ liệu nào chúng ta cần, hãy đọc một số trong số dữ liệu đó. Đầu tiên, chúng ta sẽ đọc ở cột nhiệt độ cao cho mỗi ngày:

```
import csv
filename = 'data/sitka_weather_07-2018_simple.csv'
with open(filename) as f:
    reader=csv.reader(f)
    header_row=next(reader)
    print(header_row)
    for index, column_header in enumerate(header_row):
        print(index, column_header)
    #get hight temperature from file
    highs=[]
    for row in reader:
        high=int(row[5])
        highs.append(high)

print(highs)
```

Trong phần mã được in đậm, dòng đầu tiên, chúng ta tạo ra một danh sách rỗng. Tại dòng thứ hai, chúng ta lặp qua các dòng còn lại trong file (dùng vòng for). Đối tượng reader tiếp tục từ nơi nó đã dừng lại trong file CSV chạy và tự động trả về từng dòng theo vị trí hiện tại của nó. Bởi vì chúng ta đã đọc hàng tiêu đề, vòng lặp sẽ bắt đầu ở dòng thứ hai nơi dữ liệu thực tế bắt đầu. Trên mỗi lần đi qua vòng lặp, chúng ta lấy dữ liệu từ chỉ mục 5, tương ứng với TMAX và gán nó cho biến high. Chúng ta sử dụng hàm int() để chuyển đổi dữ liệu được lưu trữ dưới dạng chuỗi, sang

định dạng số để chúng ta có thể sử dụng nó. Sau đó, chúng ta thêm vào danh sách *highs*.

Danh sách sau đây cho thấy dữ liệu hiện được lưu trữ trong danh sách *highs*:

```
[62, 58, 70, 70, 67, 59, 58, 62, 66, 59, 56, 63, 65, 58, 56, 59, 64, 60, 60,  
61, 65, 65, 63, 59, 64, 65, 68, 66, 64, 67, 65]
```

Chúng ta đã trích xuất nhiệt độ cao cho từng ngày và lưu trữ từng giá trị trong danh sách. Nay, hãy tạo hình ảnh trực quan về dữ liệu này.

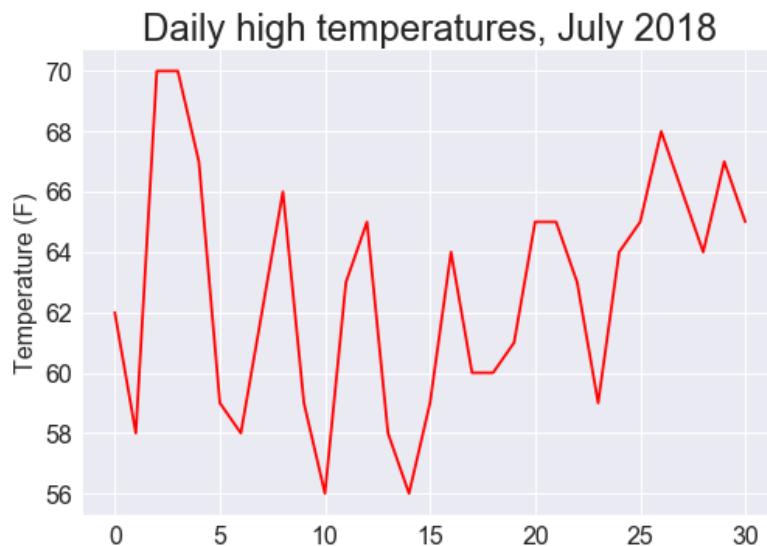
Lập đồ thị dữ liệu trong biểu đồ nhiệt độ

Để trực quan dữ liệu nhiệt độ mà chúng ta có, trước tiên ta sẽ tạo một biểu đồ đơn giản về mức nhiệt độ cao hàng ngày bằng cách sử dụng Matplotlib, như được hiển thị ở đây:

```
import csv  
  
import matplotlib.pyplot as plt  
  
  
filename = 'data/sitka_weather_07-2018_simple.csv'  
with open(filename) as f:  
    reader=csv.reader(f)  
    header_row=next(reader)  
    print(header_row)  
    for index, column_header in enumerate(header_row):  
        print(index, column_header)  
    #get hight temperature from file  
    highs=[]  
    for row in reader:  
        high=int(row[5])  
        highs.append(high)  
    # Plot the high temperatures.  
    plt.style.use('seaborn')  
    fig, ax = plt.subplots()  
    ax.plot(highs, c='red')  
  
  
    # Format plot.  
    plt.title("Daily high temperatures, July 2018", fontsize=24)  
    plt.xlabel('', fontsize=16)  
    plt.ylabel("Temperature (F)", fontsize=16)  
    plt.tick_params(axis='both', which='major', labelsize=16)
```

Ở phần trên, chúng ta đã hình thành danh sách các điểm nhiệt độ cao là *highs*. Chúng ta truyền danh sách *highs* vào phương thức *plot()* và truyền *c='red'* vẽ đường màu đỏ. (Ta vẽ các điểm nhiệt độ cao màu đỏ và nhiệt độ thấp màu xanh). Tiếp đó,

chúng ta thiết lập các chi tiết của biểu đồ như tiêu đề, phông chữ, và các nhãn như đã được hướng dẫn trong phần trước. Bởi vì chúng ta vẫn chưa thêm ngày, ta sẽ không gắn nhãn trục x, nhưng plt.xlabel() sẽ sửa đổi kích thước phông chữ để các nhãn mặc định dễ đọc hơn. Hình 10.23 cho thấy biểu đồ kết quả: một biểu đồ đường đơn giản về nhiệt độ cao trong Tháng Bảy 2018 ở Sitka, Alaska.



Hình 10.23. Biểu đồ nhiệt độ đơn giản

Module datetime

Chúng ta thêm ngày vào biểu đồ của để làm cho nó hữu ích hơn. Dữ liệu ngày tháng đầu tiên từ tệp dữ liệu thời tiết nằm ở hàng thứ hai của tệp:

```
"USW00025333", "SITKA AIRPORT, AK US", "2018-07-01", "0.25", , "62", "50"
```

Dữ liệu sẽ được đọc dưới dạng một chuỗi, vì vậy chúng ta cần một cách để chuyển đổi chuỗi "2018-07-01" thành một đối tượng đại diện cho ngày này. Ta có thể xây dựng một đối tượng đại diện cho ngày 1 tháng 7 năm 2018 bằng cách sử dụng phương thức strftime() từ module datetime. Ta xem cách strftime() hoạt động trong một phiên đầu cuối:

```
>>> from datetime import datetime
>>> first_date = datetime.strptime('2018-07-01', '%Y-%m-%d')
>>> print(first_date)
2018-07-01 00:00:00
```

Đầu tiên chúng ta nhập lớp datetime từ module datetime. Sau đó chúng ta gọi phương thức strftime() bằng cách sử dụng chuỗi chứa ngày mà chúng ta muốn làm việc với như là đối số đầu tiên của nó. Đối số thứ hai cho Python biết cách ngày được

định dạng. Trong ví dụ này, Python diễn giải '% Y-' có nghĩa là phần của chuỗi trước dấu gạch ngang đầu tiên là năm có bốn chữ số; '% m-' có nghĩa là một phần của chuỗi trước dấu gạch ngang thứ hai là một số đại diện cho tháng; và '% d' có nghĩa là phần cuối cùng của chuỗi là ngày trong tháng, từ 1 đến 31.

Phương thức strftime() có thể sử dụng nhiều đối số khác nhau để xác định cách diễn giải ngày tháng. Bảng 10.1 cho thấy một số tham số này.

Bảng 10.1. Tham số biểu thị thời gian

Tham số	Ý nghĩa
%A	Weekday name, such as Monday
%B	Month name, such as January
%m	Month, as a number (01 to 12)
%d	Day of the month, as a number (01 to 31)
%Y	Four-digit year, such as 2019
%y	Two-digit year, such as 19
%H	Hour, in 24-hour format (00 to 23)
%I	Hour, in 12-hour format (01 to 12)
%p	am or pm
%M	Minutes (00 to 59)
%S	Seconds (00 to 61)

Vẽ biểu đồ thời gian

Giờ đây, chúng ta có thể cải thiện biểu đồ dữ liệu nhiệt độ của mình bằng cách trích xuất ngày tháng cho mức cao hàng ngày và chuyển các mức cao và ngày đó vào plot(), như được hiển thị ở đây:

```
import csv
import matplotlib.pyplot as plt
from datetime import datetime
filename = 'data/sitka_weather_07-2018_simple.csv'
with open(filename) as f:
    reader=csv.reader(f)
    header_row=next(reader)
    print(header_row)
```

```

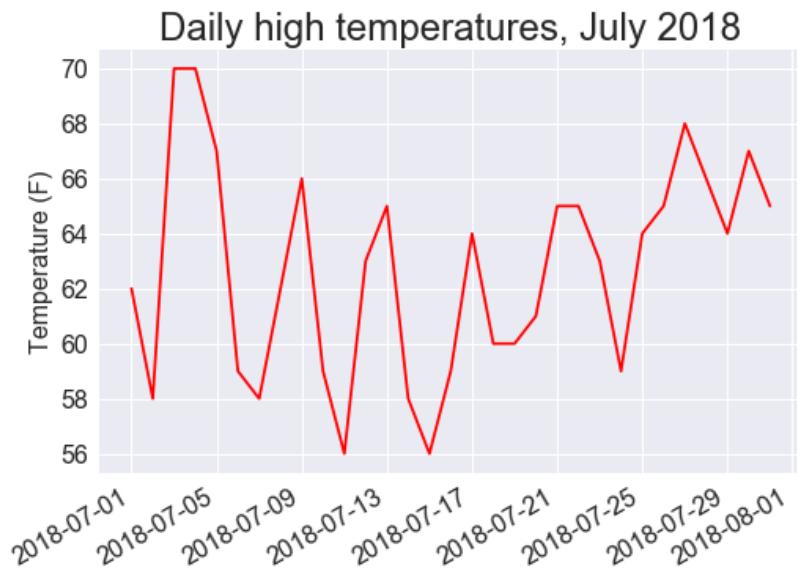
#     for index, column_header in enumerate(header_row):
#         print(index, column_header)
# get hight temperature from file
dates=[]
highs=[]
for row in reader:
    current_date = datetime.strptime(row[2], '%Y-%m-%d')
dates.append(current_date)
    high=int(row[5])
    highs.append(high)

# Plot the high temperatures.
plt.style.use('seaborn')
fig, ax = plt.subplots()
ax.plot(dates,highs, c='red')

# Format plot.
plt.title("Daily high temperatures, July 2018", fontsize=24)
plt.xlabel('', fontsize=16)
fig.autofmt_xdate()
plt.ylabel("Temperature (F)", fontsize=16)
plt.tick_params(axis='both', which='major', labelsize=16)

```

Bổ sung phần mã nguồn từ phần trên là các dòng in đậm. Phần in đậm đầu tiên, ta khai báo thêm một mảng lưu trữ dữ liệu ngày là dates. Tiếp đó, chúng ta chuyển dữ liệu ngày tháng từ kiểu chuỗi sang kiểu datetime và thêm vào mảng dates tại phần in đậm thứ hai. Tiếp đó, tại phần in đậm thứ ba, chúng ta truyền hai danh sách là dates và highs cho phương thức plot() để vẽ hai trục và tại phần in đậm thứ tư, ta gọi hàm fig.autofmt_xdate() để vẽ các nhãn của trục x theo đường chéo để tránh việc các nhãn này đè lên nhau. Hình dưới thể hiện đồ thị kết quả:



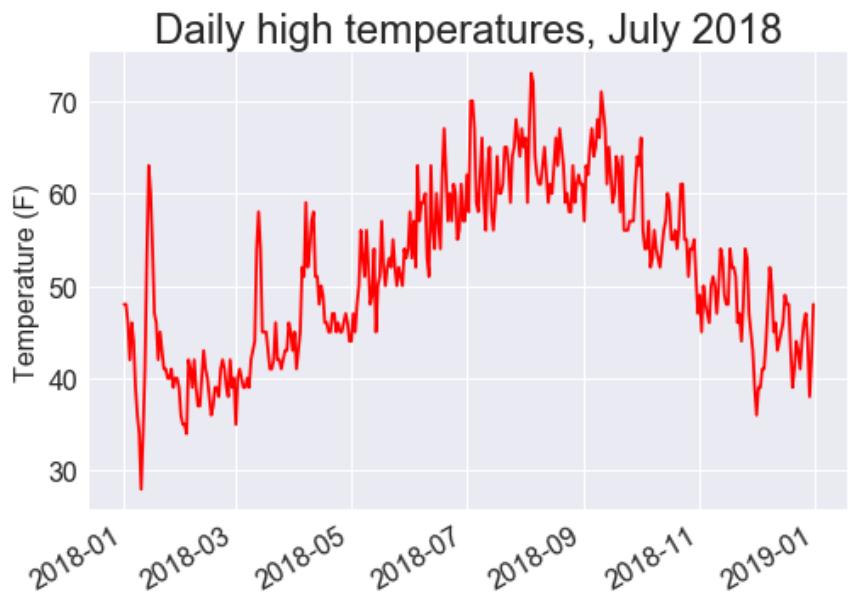
Hình 10.24. Đồ thị có trục X theo thời gian

Vẽ một khung thời gian dài hơn

Với biểu đồ của chúng ta đã được thiết lập, ta thêm nhiều dữ liệu hơn để có được bức tranh hoàn chỉnh hơn về thời tiết ở Sitka. Sao chép tệp sitka_weather_2018_simple.csv, tệp này chứa dữ liệu thời tiết cả năm cho Sitka, vào thư mục nơi đang lưu trữ dữ liệu mã nguồn của phần này. Giờ đây, chúng ta có thể tạo biểu đồ cho thời tiết của cả năm:

```
--snip--
filename = 'data/sitka_weather_2018_simple.csv'
with open(filename) as f:
--snip--
# Format plot.
plt.title("Daily high temperatures - 2018", fontsize=24)
plt.xlabel('', fontsize=16)
--snip--
```

Có hai phần thay đổi so với đoạn mã ở trên. Phần thứ nhất, chúng ta thay đổi tệp dữ liệu từ dữ liệu của tháng 7 thành dữ liệu của cả năm (file sitka_weather_2018_simple.csv). Phần thứ hai, chúng ta thay đổi tiêu đề của biểu đồ từ July -2018 thành 2018 (bỏ July). Kết quả của biểu đồ như hình dưới:



Hình 10.25. Đồ thị nhiệt độ cả năm

Vẽ biểu đồ chuỗi dữ liệu thứ hai

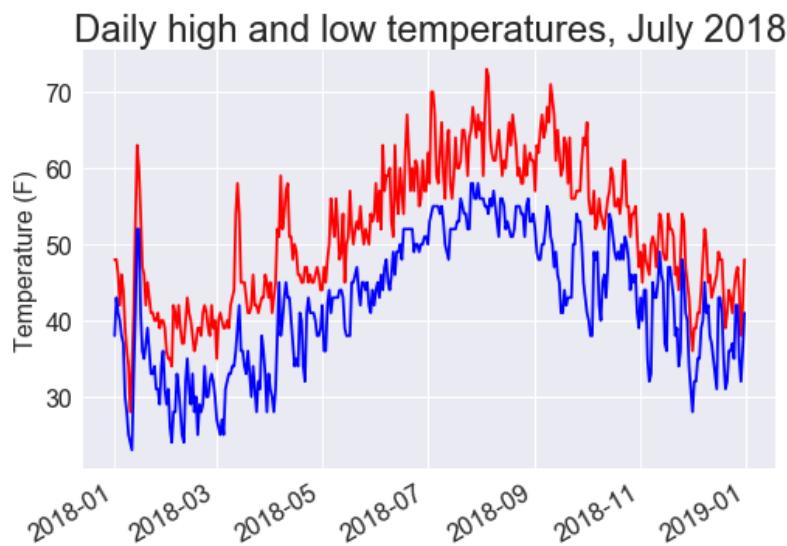
Chúng ta có thể làm cho biểu đồ thông tin của mình hữu ích hơn nữa bằng cách bao gồm nhiệt độ thấp. Chúng ta cần trích xuất nhiệt độ thấp từ luồng dữ liệu và sau đó thêm chúng vào biểu đồ, như được hiển thị ở đây:

```
with open(filename) as f:
    reader=csv.reader(f)
    header_row=next(reader)
    print(header_row)
    # for index, column_header in enumerate(header_row):
    #     print(index, column_header)
    #get hight temperature from file
    dates=[]
    highs=[]
    lows=[]
    for row in reader:
        current_date = datetime.strptime(row[2], '%Y-%m-%d')
        high=int(row[5])
        highs.append(high)
        dates.append(current_date)
        low=int(row[6])
        lows.append(low)

    # Plot the high temperatures.
    plt.style.use('seaborn')
    fig, ax = plt.subplots()
    ax.plot(dates,highs, c='red')
    ax.plot(dates,lows, c='blue')
```

```
plt.title("Daily high and low temperatures, July 2018", fontsize=24)
```

Nội dung đoạn mã nguồn được bổ sung và thay đổi ở các phần in đậm. Chúng ta thêm danh sách nhiệt độ thấp lows trống và trong vòng lặp ta lấy các giá trị nhiệt độ thấp ở các dòng thuộc cột thứ 7 (danh mục 6) và thêm vào danh sách lows. Tiếp đó, ở ngoài vòng lặp, chúng ta sử dụng hàm plot để vẽ biểu đồ thứ hai, truyền vào đó các tham số dates, lows và màu là xanh (blue). Cuối cùng, việc cập nhật tên biểu đồ là cần thiết để phù hợp với nội dung dữ liệu đã vẽ. Kết quả được thể hiện ở hình sau đây:



Hình 10.26. Đồ thị nhiệt độ cao – thấp năm 2018

Tô bóng một vùng trong biểu đồ

Sau khi thêm hai chuỗi dữ liệu, giờ đây chúng ta có thể kiểm tra phạm vi nhiệt độ cho mỗi ngày. Hãy thêm phần hoàn thiện vào biểu đồ bằng cách sử dụng tô bóng để hiển thị phạm vi giữa nhiệt độ cao và thấp của mỗi ngày.

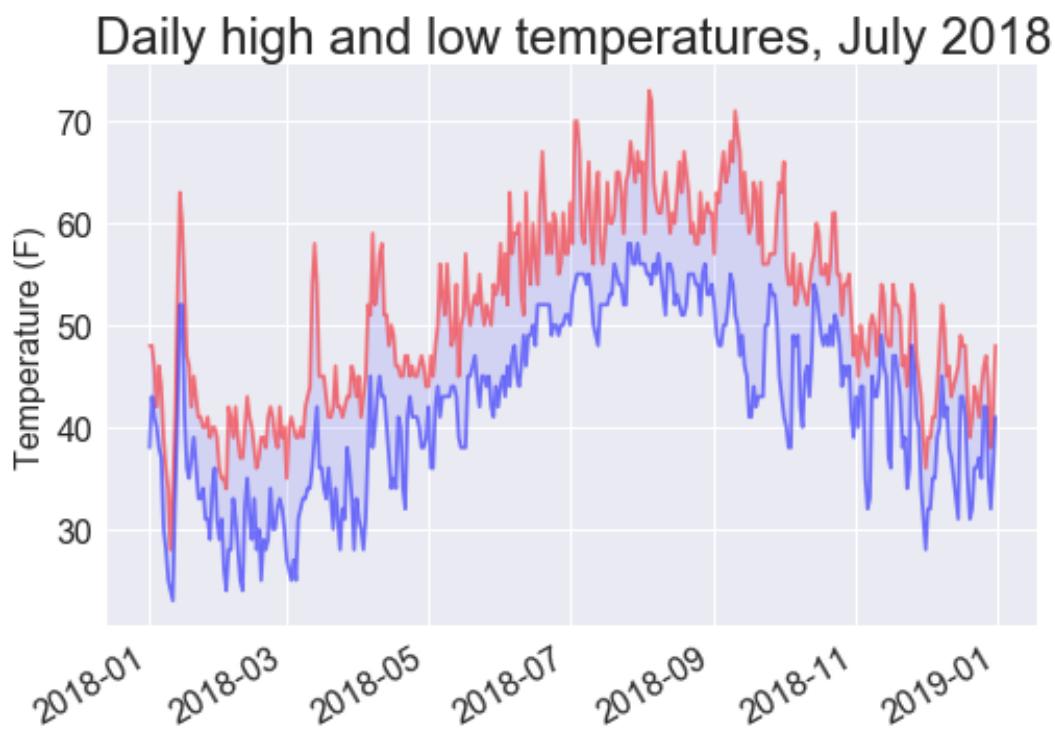
Để làm như vậy, chúng ta sẽ sử dụng phương thức fill_between(), phương thức này nhận một chuỗi giá trị x và hai chuỗi giá trị y và lấp đầy khoảng trống giữa hai chuỗi giá trị y:

```
--snip--  
# Plot the high and low temperatures.  
plt.style.use('seaborn')  
fig, ax = plt.subplots()  
ax.plot(dates, highs, c='red', alpha=0.5)  
ax.plot(dates, lows, c='blue', alpha=0.5)  
plt.fill_between(dates, highs, lows, facecolor='blue', alpha=0.1)  
--snip--
```

Trong đoạn mã trên, chúng ta thấy có thêm giá trị alpha =0.5 tại hai phương thức plot(). Alpha thiết lập độ trong suốt của màu vẽ các đường. Một giá trị alpha của 0 là hoàn toàn trong suốt và 1 (mặc định) là hoàn toàn không trong suốt. Bằng cách thiết lập alpha thành 0,5, chúng ta làm cho các đường ô màu đỏ và xanh lam có vẻ nhạt hơn.

Chúng ta truyền vào fill_between() danh sách ngày cho các giá trị x và sau đó hai giá trị cao và thấp của chuỗi giá trị y. Đôi số *facecolor* xác định màu của vùng bóng mờ; chúng ta cung cấp cho nó một giá trị alpha thấp là 0,1 để vùng kết nối hai chuỗi dữ liệu mà không làm phân tán thông tin mà chúng đại diện.

Kết quả như hình dưới, có xuất hiện hình bóng mờ giữa mức cao và mức thấp:



Hình 10.27. Đồ bóng đồ thị nhiệt độ

Đồ bóng giúp làm cho phạm vi giữa hai tập dữ liệu rõ ràng ngay lập tức so với trước đây.

Kiểm tra lỗi

Chúng ta có thể chạy mã nguồn ở trên bằng cách sử dụng dữ liệu cho vị trí bất kỳ. Nhưng một số trạm thời tiết thu thập dữ liệu khác với những trạm khác, và một số đôi khi trực trặc và không thu thập được một số dữ liệu mà chúng phải thực hiện.

Thiếu dữ liệu có thể dẫn đến các trường hợp ngoại lệ làm hỏng các chương trình trừ khi chúng ta xử lý chúng đúng cách.

Ví dụ: hãy xem điều gì sẽ xảy ra khi chúng ta cố gắng tạo biểu đồ nhiệt độ cho Thung lũng Chết, California. Sao chép tệp death_valley_2018_simple.csv vào thư mục ta đang lưu trữ dữ liệu cho mã nguồn này.

Đầu tiên, chúng ta chạy tệp với tệp dữ liệu death_valley_2018_simple.csv và in ra header của tệp:

```
import csv
import matplotlib.pyplot as plt
from datetime import datetime
#filename = 'data/sitka_weather_2018_simple.csv'
filename = 'data/death_valley_2018_simple.csv'
with open(filename) as f:
    reader=csv.reader(f)
    header_row=next(reader)
    print(header_row)
    for index, column_header in enumerate(header_row):
        print(index, column_header)
```

Dữ liệu được in ra:

```
0 STATION
1 NAME
2 DATE
3 PRCP
4 TMAX
5 TMIN
6 TOBS
```

Ngày ở cùng vị trí ở chỉ số 2. Nhưng nhiệt độ cao và thấp ở chỉ số 4 và 5, vì vậy chúng ta cần thay đổi các chỉ số trong mã để trỏ sang các vị trí mới này. Thay vì bao gồm chỉ số nhiệt độ trung bình trong ngày, trạm này bao gồm TOBS, chỉ số cho một đặc điểm thời gian quan sát.

Ta xóa một trong các kết quả đo nhiệt độ khỏi chương trình này để hiển thị kết quả khi một số dữ liệu bị thiếu trong một tệp. Thay đổi sitka_highs_lows.py để tạo một biểu đồ cho Thung lũng Chết bằng cách sử dụng các chỉ mục mà chúng ta vừa lưu ý, và xem điều gì sẽ xảy ra:

```
with open(filename) as f:
    reader=csv.reader(f)
    header_row=next(reader)
```

```

print(header_row)
for index, column_header in enumerate(header_row):
    print(index, column_header)
#Get hight temperature from file
dates=[]
highs=[]
lows=[]
for row in reader:
    current_date = datetime.strptime(row[2], '%Y-%m-%d')
    high=int(row[4])
    highs.append(high)
    dates.append(current_date)
    low=int(row[5])
    lows.append(low)

```

Tại đây, chúng ta cập nhật giá trị danh mục cho các phần tử nhiệt độ cao và nhiệt độ thấp. Khi chạy chúng ta thấy lỗi xảy ra ở cuối của cửa sổ chạy:

Traceback (most recent call last):

```

File "death_valley_highs_lows.py", line 15, in <module>
    high = int(row[4])
ValueError: invalid literal for int() with base 10: ''

```

Traceback cho chúng ta biết python không thể xử lý dữ liệu của nhiệt độ cao khi tìm cách chuyển chuỗi rỗng thành kiểu số nguyên. Thay vì xem qua dữ liệu và tìm ra giá trị bị thiếu, chúng ta sẽ chỉ trực tiếp xử lý các trường hợp thiếu dữ liệu.

Chúng ta sẽ chạy mã kiểm tra lỗi khi các giá trị đang được đọc từ CSV chạy để xử lý các trường hợp ngoại lệ có thể phát sinh. Đây là mã nguồn tương ứng:

```

dates=[]
highs=[]
lows=[]
for row in reader:
    current_date = datetime.strptime(row[2], '%Y-%m-%d')
    try:
        high=int(row[4])
        low=int(row[5])
    except ValueError:
        print(f'Missing data for {current_date}')
    else:
        dates.append(current_date)
        highs.append(high)
        lows.append(low)

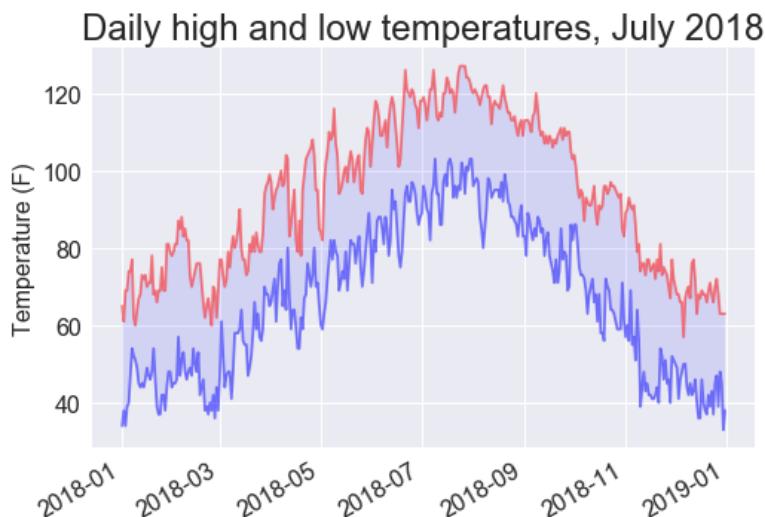
```

Mỗi khi chúng ta kiểm tra một hàng, ta có gắng trích xuất ngày tháng và giá trị nhiệt độ cao và nhiệt độ thấp. Nếu bất kỳ dữ liệu nào bị thiếu, Python sẽ tạo ra một ValueError và chúng ta xử lý nó bằng cách in một thông báo lỗi bao gồm ngày thiếu dữ liệu và dữ liệu tương ứng với hàng này sẽ bị bỏ qua. Sau khi in lỗi, vòng lặp sẽ tiếp tục xử lý hàng tiếp theo. Nếu tất cả dữ liệu cho một ngày được truy xuất mà không có lỗi, khói khác sẽ chạy và dữ liệu sẽ được nối vào danh sách tương ứng.

Do chúng ta đã bắt và xử lý lỗi, màn hình in ra dòng có ngày tháng thiếu dữ liệu:

```
Missing data for 2018-02-18 00:00:00
```

Bởi vì lỗi được xử lý thích hợp, mã của chúng ta có thể tạo ra một biểu đồ, bỏ qua dữ liệu bị thiếu. Hình 10.28 cho thấy biểu đồ kết quả.



Hình 10.28. Đồ thị kết quả

So sánh biểu đồ này với biểu đồ Sitka, chúng ta có thể thấy rằng Thung lũng Chết nói chung là ám hơn đông nam Alaska, như chúng ta mong đợi. Ngoài ra, phạm vi của nhiệt độ mỗi ngày trên sa mạc cao hơn. Chiều cao của bóng mờ khu vực làm rõ điều này.

Nhiều tập dữ liệu mà ta làm việc sẽ bị thiếu, được định dạng không đúng hoặc dữ liệu không chính xác. Bạn có thể sử dụng các công cụ ta đã học trong nửa đầu của bài giảng này để xử lý các tình huống này. Ở đây, chúng ta đã sử dụng khối try-except-else để xử lý dữ liệu bị thiếu. Đôi khi, ta sẽ sử dụng continue để bỏ qua một số dữ liệu hoặc sử dụng remove() hoặc del để loại bỏ một số dữ liệu sau khi nó

được trích xuất. Sử dụng bất kỳ cách tiếp cận nào hoạt động, miễn là kết quả là hình ảnh trực quan chính xác, có ý nghĩa.

10.2.2.2. Định dạng JSON

Trong phần này, ta sẽ tải xuống tập dữ liệu đại diện cho tất cả các trận động đất đã xảy ra trên thế giới trong tháng trước. Sau đó, ta sẽ tạo một bản đồ hiển thị vị trí của những trận động đất này và mức độ quan trọng mỗi trận động đất đó. Vì dữ liệu được lưu trữ ở định dạng JSON, chúng ta sẽ làm việc với nó bằng cách sử dụng mô-đun json. Sử dụng công cụ lập bản đồ thân thiện với người mới bắt đầu của Plotly cho dữ liệu dựa trên vị trí, ta sẽ tạo hình ảnh hóa hiển thị rõ ràng toàn cầu sự phân bố của các trận động đất.

Tải dữ liệu về động đất

Sao chép file eq_1_day_m1.json vào thư mục đang lưu trữ dữ liệu cho các chương trình. Các trận động đất được phân loại theo độ lớn của chúng trên thang độ Richter. Chương trình này bao gồm dữ liệu cho tất cả các trận động đất với độ lớn M1 hoặc lớn hơn diễn ra trong 24 giờ qua (tại thời điểm của văn bản này).

Kiểm tra dữ liệu JSON

Khi ta mở eq_1_day_m1.json, ta sẽ thấy rằng nó rất đặc và khó để đọc:

```
{"type": "FeatureCollection", "metadata": {"generated": 1550361461000, ...  
{"type": "Feature", "properties": {"mag": 1.2, "place": "11km NNE of Nor...  
{"type": "Feature", "properties": {"mag": 4.3, "place": "69km NNW of Ayn...  
{"type": "Feature", "properties": {"mag": 3.6, "place": "126km SSE of Co...  
{"type": "Feature", "properties": {"mag": 2.1, "place": "21km NNW of Teh...  
{"type": "Feature", "properties": {"mag": 4, "place": "57km SSW of Kakto...  
--snip--
```

Tệp này được định dạng cho máy nhiều hơn cho người. Nhưng chúng ta có thể thấy rằng tệp chứa một số từ điển cũng như thông tin mà chúng ta quan tâm, chẳng hạn như cường độ động đất và các địa điểm.

Module json cung cấp nhiều công cụ khác nhau để khám phá và làm việc với dữ liệu JSON. Một số công cụ này sẽ giúp chúng ta định dạng lại chúng ta có thể xem dữ liệu thô dễ dàng hơn trước khi bắt đầu làm việc với nó theo chương trình.

```
import json  
filename = 'data/eq_data_1_day_m1.json'  
with open(filename) as f:
```

```

all_eq_data = json.load(f)
readable_file = 'data/readable_eq_data.json'
with open(readable_file, 'w') as f:
    json.dump(all_eq_data, f, indent=4)

```

Đầu tiên chúng ta sẽ nhập module json để nạp dữ liệu chính xác từ tệp và lưu trong biến all_eq_data. Hàm Json.load() chuyển đổi dữ liệu thành một định dạng mà Python có thể làm việc được: trong trường hợp này là một từ điển không lò. Tiếp theo, chúng ta tạo một tệp để ghi cùng một dữ liệu này vào một định dạng dễ đọc hơn. Hàm json.dump() nhận một đối tượng dữ liệu JSON và một đối tượng tệp và ghi dữ liệu vào tệp đó. đối số indent = 4 ra lệnh cho dump() định dạng dữ liệu bằng cách sử dụng thụt lề phù hợp với cấu trúc của dữ liệu.

Khi nhìn vào thư mục dữ liệu của mình và mở tệp readable_eq_data.json, đây là phần đầu tiên của những gì ta sẽ thấy:

```

{
    "type": "FeatureCollection",
    "metadata": {
        "generated": 1550361461000,
        "url": "https://earthquake.usgs.gov/earthquakes/feed/v1.0/summary/1.0_day.geojson",
        "title": "USGS Magnitude 1.0+ Earthquakes, Past Day",
        "status": 200,
        "api": "1.7.0",
        "count": 158
    },
    "features": [
        {
            "type": "Feature",
            "properties": {

```

Phần đầu tiên của chương trình bao gồm một phần có khóa "metadata". Mục này cho chúng ta biết khi nào tệp dữ liệu được tạo và nơi chúng ta có thể tìm thấy dữ liệu trực tuyến. Nó cũng cung cấp cho chúng ta một tiêu đề mà con người có thể đọc được và số lượng các trận động đất bao gồm trong chương trình này. Trong khoảng thời gian 24 giờ này, 158 trận động đất đã được ghi nhận.

GeoJSON này có cấu trúc hữu ích cho dữ liệu dựa trên vị trí. Thông tin được lưu trữ trong danh sách được liên kết với khóa "feature". Bởi vì trang này chứa dữ liệu động đất, dữ liệu ở dạng danh sách trong đó mọi mục trong danh sách tương ứng với một trận động đất. Cấu trúc này có thể trông khó hiểu, nhưng nó khá mạnh mẽ.

Nó cho phép các nhà địa chất lưu trữ như nhiều thông tin cần thiết trong từ điển về mỗi trận động đất, và sau đó nhét tất cả các từ điển đó vào một danh sách lớn.

Hãy xem từ điển đại diện cho một trận động đất:

```
{  
    "type": "Feature",  
    "properties": {  
        "mag": 1.2,  
        "place": "11km NNE of North Nenana, Alaska",  
        "time": 1550358909272,  
        "updated": 1550359211283,  
        "tz": -540,  
        "url":  
        "https://earthquake.usgs.gov/earthquakes/eventpage/ak0192641ikq",  
        "detail":  
        "https://earthquake.usgs.gov/earthquakes/feed/v1.0/detail/ak0192641ikq.geojson"  
        ,  
        "felt": null,  
        "cdi": null,  
        "mmi": null,  
        "alert": null,  
        "status": "automatic",  
        "tsunami": 0,  
        -snip-  
        "title": "M 1.2 - 11km NNE of North Nenana, Alaska"  
    },  
    "geometry": {  
        "type": "Point",  
        "coordinates": [  
            -148.9865,  
            64.6673,  
            0  
        ]  
    },  
    "id": "ak0192641ikq"  
}
```

Khóa properties chứa rất nhiều thông tin về trận động đất. Chúng ta chủ yếu quan tâm đến cường độ của mỗi trận động đất, liên kết với khóa "mag". Chúng ta cũng quan tâm đến tiêu đề của mỗi trận động đất, cung cấp một bản tóm tắt rõ ràng về cường độ và vị trí của nó.

Khóa ‘geometry’ giúp chúng ta hiểu được vị trí của trận động đất xảy ra. Chúng ta sẽ cần thông tin này để lập bản đồ từng sự kiện. Chúng ta có thể tìm thấy kinh độ x và vĩ độ y cho mỗi trận động đất trong danh sách được liên kết với khóa “coordinates”.

Tệp này chứa nhiều cách lồng hơn so với cách chúng ta sử dụng trong mã đã viết, vì vậy nếu nó trông khó hiểu, không cần lo lắng: Python sẽ xử lý hầu hết các độ phức tạp. Chúng ta sẽ chỉ làm việc với một hoặc hai cấp lồng nhau cùng một lúc. Chúng ta sẽ bắt đầu bằng cách lấy ra một từ điển cho từng trận động đất được ghi lại trong khoảng thời gian 24 giờ.

Tạo ra danh sách các trận động đất

Đầu tiên, chúng ta sẽ tạo một danh sách chứa tất cả thông tin về mọi trận động đất đã xảy ra.

```
import json
# Explore the structure of the data.
filename = 'data/eq_data_1_day_m1.json'
with open(filename) as f:
    all_eq_data = json.load(f)
all_eq_dicts = all_eq_data['features']
print(len(all_eq_dicts))
```

Chúng ta lấy dữ liệu được liên kết với khóa ‘feature’ và lưu trữ nó trong biến all_eq_dicts. Chúng ta biết trang này chứa các hồ sơ về 158 trận động đất, và điều ra xác minh rằng ta đã nắm bắt được tất cả các trận động đất trong thời gian ngắn:

Chú ý đoạn mã này ngắn như thế nào. Tệp được định dạng rõ ràng readable_eq_data.json có hơn 6.000 dòng. Nhưng chỉ trong vài dòng, chúng ta có thể đọc qua tất cả dữ liệu đó và lưu trữ nó trong một danh sách Python. Tiếp theo, chúng ta sẽ kéo các cường độ từ từng trận động đất.

Lấy ra thông tin cường độ

Sử dụng danh sách chứa dữ liệu về từng trận động đất, chúng ta có thể lặp lại danh sách đó và trích xuất bất kỳ thông tin nào chúng ta muốn. Bây giờ chúng ta sẽ lấy độ lớn của mỗi trận động đất:

```
import json
# Explore the structure of the data.
filename = 'data/eq_data_1_day_m1.json'
```

```

with open(filename) as f:
    all_eq_data = json.load(f)
all_eq_dicts = all_eq_data['features']
print(len(all_eq_dicts))
mags = []
for eq_dict in all_eq_dicts:
    mag = eq_dict['properties']['mag']
    mags.append(mag)
print(mags[:10])

```

Tại phần mã in đậm, đầu tiên ta tạo ra một danh sách rỗng mags chứa thông tin cường độ các trận động đất. Sau đó, ta lặp qua từ điển *all_eq_dicts*. Bên trong vòng lặp, mỗi trận động đất được đại diện bằng một từ điển *eq_dict* và mỗi cường độ của trận động đất được lưu trong phần ‘*properties*’ của từ điển với khóa ‘*mag*’. Chúng ta lưu cường độ bởi biến *mag* và thêm nó vào trong danh sách *mags*[]. Lệnh in ra 10 cường độ đầu tiên có kết quả chính xác:

```
[0.96, 1.2, 4.3, 3.6, 2.1, 4, 1.06, 2.3, 4.9, 1.8]
```

Tiếp theo, chúng ta sẽ lấy ra dữ liệu về vị trí để hiển thị chính xác vị trí tọa độ của trận động đất.

Lấy dữ liệu vị trí

Dữ liệu vị trí được lưu trữ trong khóa “geometry”. Bên trong từ điển *geometry* là một khóa “coordinates” và hai giá trị đầu tiên trong danh sách này là kinh độ và vĩ độ. Đây là cách chúng ta sẽ lấy dữ liệu này:

```

mags, lons, lats = [], [], []
for eq_dict in all_eq_dicts:
    mag = eq_dict['properties']['mag']
    lon = eq_dict['geometry']['coordinates'][0]
    lat = eq_dict['geometry']['coordinates'][1]
    mags.append(mag)
    lons.append(lon)
    lats.append(lat)
print(mags[:10])
print(lons[:5])
print(lats[:5])

```

Tại các phần mã in đậm trong phần này, chúng ta đầu tiên tạo ra hai danh sách trống để lưu kinh độ và vĩ độ của các trận động đất. Tại trong từ điển *eq_dict* trong danh sách mà chúng ta lặp, khóa *geometry* cho chúng ta biết đó là phần lưu tọa độ của các trận động đất. Chúng ta truy nhập phần này qua khóa *geometry*, sau đó tới

coordinates để lấy ra các vị trí. Tiếp đó, chỉ mục 0 và 1 cho biết chúng ta sẽ lấy ra kinh độ và vĩ độ rồi gán vào các biến *lon* và *lat* tương ứng. Cuối cùng, chúng ta thêm các biến *lon* và *lat* vào danh sách tương ứng ban đầu tạo ra và in ra các danh sách đó để cho thấy kết quả chính xác:

```
[0.96, 1.2, 4.3, 3.6, 2.1, 4, 1.06, 2.3, 4.9, 1.8]  
[-116.7941667, -148.9865, -74.2343, -161.6801, -118.5316667]  
[33.4863333, 64.6673, -12.1025, 54.2232, 35.3098333]
```

Với kết quả này, chúng ta có thể đi xây dựng bản đồ trực quan về các trận động đất.

Xây dựng bản đồ thế giới về động đất

Với thông tin mà chúng ta đã thu thập được cho đến thời điểm này, chúng ta có thể xây dựng một bản đồ thế giới đơn giản. Mặc dù nó trông chưa thể hiện được nhưng chúng ta muốn đảm bảo thông tin được hiển thị chính xác trước khi tập trung vào các vấn đề về phong cách và trình bày. Đây là bản đồ ban đầu:

```
① from plotly.graph_objs import Scattergeo, Layout  
from plotly import offline  
② data = [Scattergeo(lon=lons, lat=lats)]  
③ my_layout = Layout(title='Global Earthquakes')  
④ fig = {'data': data, 'layout': my_layout}  
offline.plot(fig, filename='global_earthquakes.html')
```

Chúng ta nhập loại biểu đồ Scattergeo và lớp Layout, sau đó nhập mô-đun offline để hiển thị bản đồ ①. Như chúng ta đã làm khi thực hiện một biểu đồ, chúng ta xác định một danh sách được gọi là data. Chúng ta tạo đối tượng Scattergeo bên trong danh sách này ②, vì ta có thể vẽ nhiều hơn một tập dữ liệu trên bất kỳ hình ảnh trực quan nào chúng ta làm. Loại biểu đồ Scattergeo cho phép ta phủ một biểu đồ phân tán của dữ liệu địa lý trên bản đồ. Trong cách sử dụng đơn giản nhất của loại biểu đồ này, ta chỉ cần cung cấp danh sách các kinh độ và danh sách các vĩ độ. Chúng ta đặt cho biểu đồ một tiêu đề thích hợp ③ và tạo một từ điển được gọi là fig chứa dữ liệu và bố cục ④. Cuối cùng, chúng ta truyền biến fig tới hàm plot() cùng với tên tệp mô tả cho đầu ra. Khi chạy chương trình này, ta sẽ thấy một bản đồ giống như bản đồ trong Hình 10.29 . Động đất thường xảy ra gần ranh giới trái đất, phù hợp với những gì chúng ta thấy trong biểu đồ.



Hình 10.29. Biểu đồ Scattergeo

Chúng ta có thể thực hiện nhiều sửa đổi để làm cho bản đồ này có ý nghĩa hơn và dễ đọc hơn, vì vậy ta thực hiện một số thay đổi sau đây.

Một cách khác để xác định dữ liệu biểu đồ

Trước khi xác định biểu đồ, chúng ta hãy xem xét một cách hơi khác để chỉ định dữ liệu cho biểu đồ Plotly. Trong biểu đồ hiện tại, danh sách dữ liệu được xác định trong một đường thẳng:

```
data = [Scattergeo(lon=lons, lat=lats)]
```

Đây là một trong những cách đơn giản nhất để xác định dữ liệu cho biểu đồ trong Plotly. Nhưng đó không phải là cách tốt nhất khi ta muốn tùy chỉnh bản trình bày.

Dưới đây là một cách tương đương để xác định dữ liệu cho biểu đồ hiện tại:

```
data = [ {
    'type': 'scattergeo',
    'lon': lons,
    'lat': lats,
} ]
```

Trong cách tiếp cận này, tất cả thông tin về dữ liệu được cấu trúc dưới dạng các cặp khóa-giá trị trong từ điển. Nếu chúng ta đặt mã này vào eq_plot.py, ta sẽ thấy cùng một biểu đồ mà chúng ta vừa tạo. Định dạng này cho phép chúng ta chỉ định các tùy chỉnh dễ dàng hơn so với định dạng trước đó.

Tùy chỉnh kích cỡ của Marker

Khi tìm hiểu cách cải thiện kiểu dáng của bản đồ, chúng ta nên tập trung vào các khía cạnh của dữ liệu mà chúng ta muốn truyền đạt rõ ràng hơn. Bản đồ hiện tại hiển thị vị trí của từng trận động đất, nhưng nó không thông báo mức độ nghiêm trọng của bất kỳ trận động đất nào.

Để thực hiện việc này, chúng ta sẽ thay đổi kích thước của các điểm đánh dấu tùy thuộc vào cường độ của từng trận động đất:

```
#data = [Scattergeo(lon=lons, lat=lats)]
data = [
    {
        'type': 'scattergeo',
        'lon': lons,
        'lat': lats,
        'marker': {
            'size': [5*mag for mag in mags],
        },
    },
]
```

Plotly cung cấp rất nhiều tùy chỉnh mà ta có thể thực hiện đối với dữ liệu chuỗi, mỗi chuỗi có thể được biểu thị dưới dạng một cặp khóa-giá trị. Ở đây, chúng ta đang sử dụng khóa 'marker' để chỉ định kích thước của mỗi điểm đánh dấu trên bản đồ.

Chúng ta sử dụng từ điển lồng nhau làm giá trị được liên kết với 'marker', vì ta có thể chỉ định một số cài đặt cho tất cả các markers trong một chuỗi.

Chúng ta muốn kích thước tương ứng với cường độ của mỗi trận động đất. Nhưng nếu chúng ta chỉ chuyển vào danh sách mags, các điểm đánh dấu sẽ quá nhỏ để có thể dễ dàng nhìn thấy kích thước sự khác biệt. Chúng ta cần nhân độ lớn với hệ số tỷ lệ để có được kích thước điểm đánh dấu thích hợp. Do đó, danh sách kích thước (size) là danh sách tương ứng của mags nhưng được nhân lên 5 lần.

Kết quả chúng ta thấy bản đồ như sau:



Hình 10.30. Hình ảnh các điểm trên biểu đồ

Tùy chỉnh màu của các điểm đánh dấu

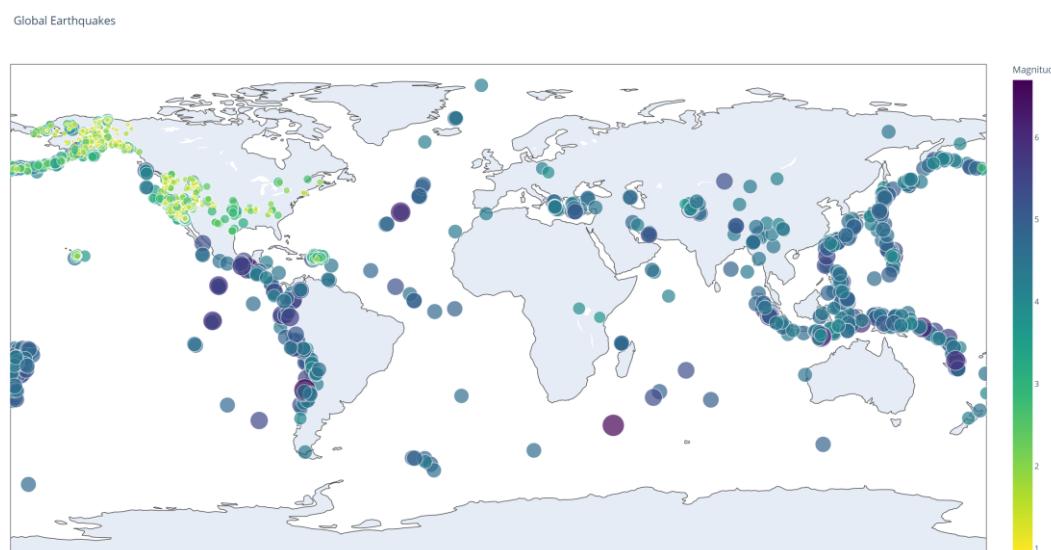
Chúng ta cũng có thể tùy chỉnh màu của từng marker để cung cấp một số phân loại về mức độ nghiêm trọng của từng trận động đất. Chúng ta sẽ sử dụng thang màu của Plotly để làm điều đó.

Trước khi chúng ta thực hiện những thay đổi này, chúng ta cần sao chép tệp eq_data_30_day_m1.json vào thư mục dữ liệu đang thực thi. Tệp này bao gồm dữ liệu động đất trong 30 ngày và bản đồ sẽ thú vị hơn nhiều khi sử dụng tập dữ liệu lớn hơn này. Dưới đây là các nội dung thay đổi về màu sắc của các trận động đất phù hợp với cường độ.

```
data = [ {
    'type': 'scattergeo',
    'lon': lons,
    'lat': lats,
    'marker': {
        'size': [5*mag for mag in mags],
        'color': mags,
        'colorscale': 'Viridis',
        'reversescale': True,
        'colorbar': {'title': 'Magnitude'},
    },
},
```

Trước tiên, chúng ta thay đổi tên tệp trong mã nguồn thành tệp dữ liệu 30 ngày thay vì 1 ngày. Tất cả các thay đổi nằm trong từ điển ‘marker’ vì chúng ta chỉ cần sửa

đổi từ điển này. Thiết lập 'color' cho Plotly biết nó sẽ sử dụng những giá trị nào để xác định vị trí của mỗi điểm đánh dấu trên thang màu. Chúng ta sử dụng danh sách mags để xác định các màu được sử dụng. Khóa colorscale cho Plotly biết dài màu được dùng là 'Viridis', dài màu này là màu trai từ xanh đen tới vàng sáng và phù hợp với tập dữ liệu này. Chúng ta đặt 'reverseescale' thành True, vì sẽ sử dụng màu vàng sáng cho các giá trị thấp nhất và màu xanh lam đậm cho các trận động đất nghiêm trọng nhất. Cài đặt 'thanh màu' cho phép chúng ta kiểm soát sự xuất hiện của thang màu hiển thị bên cạnh bản đồ. Ở đây chúng ta đặt tên thang màu là 'Magnitude' để làm rõ những cái màu sắc đại diện.



Hình 10.31. Biểu đồ có dài màu

Dài màu khác

Ta cũng có thể chọn từ một số thang màu khác. Để xem các thang màu có sẵn, hãy lưu chương trình ngắn dưới đây:

```
from plotly import colors
for key in colors.PLOTLY_SCALES.keys():
    print(key)
```

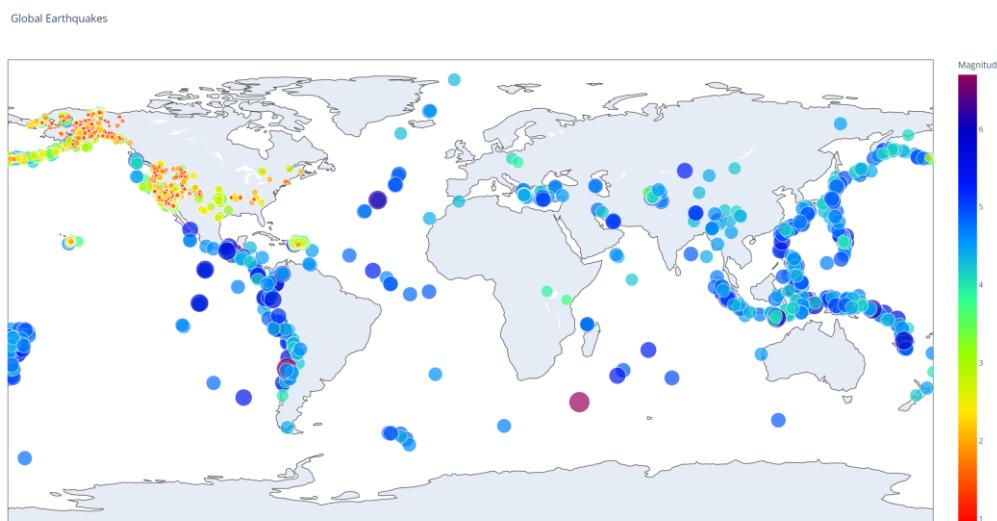
Plotly lưu trữ các thang màu trong mô-đun màu sắc. Các thang màu là được định nghĩa trong từ điển PLOTLY_SCALES và tên của các thang màu đóng vai trò là các khóa trong từ điển. Đây là kết quả hiển thị tất cả các thang màu có sẵn:

```
Greys
YlGnBu
Greens
--snip--
```

Viridis

Ta nên thử những màu sắc này; hãy nhớ rằng ta có thể đảo ngược bất kỳ tỷ lệ nào trong số này bằng cách sử dụng cài đặt tỷ lệ đảo ngược.

Ghi chú: Nếu in từ điển PLOTLY_SCALES, ta có thể xem cách phân loại màu sắc. Mọi dải màu đều có màu đầu và màu kết thúc, và một số dải màu cũng có một hoặc nhiều màu trung gian được định nghĩa. Plotly nội suy các sắc thái giữa mỗi màu được làm mờ này. Hình minh họa dải màu rainbow



Hình 10.32. Dải màu khác

Thêm chữ hiển thị

Để hoàn thiện bản đồ này, chúng ta sẽ thêm một số văn bản thông tin xuất hiện khi di chuột qua điểm đánh dấu đại diện cho một trận động đất. Ngoài việc hiển thị kinh độ và vĩ độ, xuất hiện theo mặc định, chúng ta cũng sẽ hiển thị độ lớn và cung cấp mô tả về vị trí gần đúng.

Để thực hiện thay đổi này, chúng ta cần lấy thêm một ít dữ liệu từ tệp và thêm nó vào từ điển trong dữ liệu:

```
mags, lons, lats, hover_texts = [], [], [], []

for eq_dict in all_eq_dicts:
    mag = eq_dict['properties']['mag']
    lon = eq_dict['geometry']['coordinates'][0]
    lat = eq_dict['geometry']['coordinates'][1]
    title = eq_dict['properties']['title']

    mags.append(mag)
    lons.append(lon)
    lats.append(lat)

    if title:
        hover_text = f'{title} - Magnitude: {mag}'
    else:
        hover_text = f'Magnitude: {mag}'
```

```

hover_texts.append(title)
data = [
    {
        'type': 'scattergeo',
        'lon': lons,
        'lat': lats,
        'text': hover_texts,
        'marker': {
            'size': [5*mag for mag in mags],
            'color': mags,
            'colorscale': 'Rainbow',
            'reversescale': True,
            'colorbar': {'title': 'Magnitude'},
        },
    },
]

```

Đầu tiên, chúng ta tạo một danh sách có tên `hover_texts` để lưu trữ nhãn mà chúng ta sẽ sử dụng cho mỗi marker. Phần "title" của dữ liệu động đất chứa mô tả tên của cường độ và vị trí của mỗi trận động đất cùng với kinh độ và vĩ độ. Tại phần mã in đậm thứ hai, chúng ta lấy dữ liệu và gán nó cho biến `title` và thêm nó vào danh sách `hover_texts`. Khi chúng ta đưa khóa 'text' vào đối tượng dữ liệu, Plotly sử dụng giá trị này làm nhãn cho mỗi điểm đánh dấu khi người xem di chuột qua điểm đánh dấu. Khi chúng ta chuyển một danh sách khớp với số điểm đánh dấu, Plotly kéo một nhãn riêng lẻ cho mỗi điểm đánh dấu mà nó tạo ra.

Khi chạy chương trình này, ta sẽ có thể di chuột qua bất kỳ điểm đánh dấu nào, xem mô tả về vị trí động đất đã xảy ra, và đọc cường độ chính xác của nó.

Trong khoảng 40 dòng mã, chúng ta đã tạo bản đồ trực quan hấp dẫn và có ý nghĩa về hoạt động động đất toàn cầu cũng minh họa cấu trúc địa chất của hành tinh. Plotly cung cấp nhiều cách ta có thể tùy chỉnh giao diện và hành vi của hình ảnh trực quan. Sử dụng nhiều tùy chọn của Plotly, chúng ta có thể tạo biểu đồ và bản đồ hiển thị chính xác những gì mong muốn.

10.2.2.3. Tiêu kết

Trong phần này ta đã học cách làm việc với các tập dữ liệu trong thế giới thực. Chúng ta đã xử lý CSV và JSON, và trích xuất dữ liệu ta muốn tập trung vào. Sử dụng dữ liệu thời tiết, ta đã biết thêm về cách làm việc với Matplotlib, bao gồm cách sử dụng mô-đun `datetime` và cách vẽ nhiều loại dữ liệu trên một biểu đồ. Ta cũng đã

vẽ dữ liệu địa lý trên bản đồ thế giới trong Plotly cũng như các bản đồ và biểu đồ theo kiểu Plotly.

Khi ta có kinh nghiệm làm việc với tệp CSV và JSON, ta sẽ có thể xử lý hầu hết mọi dữ liệu ta muốn phân tích. Chúng ta có thể tải xuống hầu hết các tập dữ liệu trực tuyến ở một trong hai hoặc cả hai định dạng này. Bằng cách làm việc với các định dạng này, ta cũng có thể học cách làm việc với các định dạng dữ liệu khác dễ dàng hơn.

Trong phần tiếp theo, chúng ta sẽ viết các chương trình tự động thu thập dữ liệu của riêng mình từ các nguồn trực tuyến và sau đó ta sẽ tạo hình ảnh hóa của dữ liệu đó. Đây là những kỹ năng thú vị cần có nếu ta muốn lập trình như một sở thích và các kỹ năng quan trọng nếu ta quan tâm đến việc lập trình một cách chuyên nghiệp.

10.2.3. Làm việc với APIs

Trong phần này, chúng ta sẽ học cách viết một chương trình độc lập tạo ra hình ảnh trực quan dựa trên dữ liệu mà nó truy xuất. Chương trình sẽ sử dụng giao diện lập trình ứng dụng web (API web) để tự động yêu cầu thông tin cụ thể từ một trang web — thay vì toàn bộ trang — và sau đó sử dụng thông tin đó để tạo hình ảnh trực quan. Các chương trình được viết theo cách này sẽ luôn sử dụng dữ liệu hiện tại để tạo hình ảnh trực quan, ngay cả khi dữ liệu đó có thể thay đổi nhanh chóng, nó sẽ luôn được cập nhật.

10.2.3.1. Sử dụng Web API

API web là một phần của trang web được thiết kế để tương tác với các chương trình. Các chương trình đó sử dụng các URL rất cụ thể để yêu cầu một số thông tin nhất định. Loại yêu cầu này được gọi là lệnh gọi API (API Call). Dữ liệu được yêu cầu sẽ được trả về ở định dạng dễ dàng xử lý, chẳng hạn như JSON hoặc CSV. Hầu hết các ứng dụng dựa vào nguồn dữ liệu bên ngoài, chẳng hạn như các ứng dụng tích hợp với các trang web truyền thông xã hội, đều dựa vào lệnh gọi API.

Git và Github

Chúng ta sẽ xây dựng hình ảnh trực quan dựa trên thông tin từ GitHub, một trang web cho phép lập trình viên để cộng tác trong các dự án lập trình. Chúng ta sẽ sử dụng API của GitHub để yêu cầu thông tin về các dự án Python trên trang web,

sau đó tạo một hình ảnh tương tác về mức độ phổ biến tương đối của các dự án này bằng cách sử dụng Plotly.

GitHub (<https://github.com/>) lấy tên từ Git, một phiên bản phân phối hệ thống điều khiển. Git giúp mọi người quản lý công việc của họ trong một dự án, vì vậy những thay đổi do một người thực hiện sẽ không ảnh hưởng đến những thay đổi mà người khác đang thực hiện. Khi chúng ta triển khai một tính năng mới trong một dự án, Git sẽ theo dõi những thay đổi mà ta thực hiện cho mỗi lần. Khi mã mới của ta hoạt động, ta cam kết (commit) các thay đổi đã thực hiện và Git ghi lại trạng thái mới của dự án của chúng ta. Nếu ta gây ra một lỗi và muốn hoàn nguyên các thay đổi của mình, ta có thể dễ dàng quay lại bất kỳ trạng thái làm việc nào trước đó. Các dự án trên GitHub được lưu trữ trong kho chứa(repositories)-mọi thứ liên quan đến dự án: mã của dự án, thông tin về các cộng tác viên, mọi vấn đề hoặc báo cáo lỗi, v.v.

Khi người dùng trên GitHub thích một dự án, họ có thể “gắn dấu sao” cho dự án đó để thể hiện sự ủng hộ và theo dõi các dự án mà họ có thể muốn sử dụng. Trong phần này, chúng ta sẽ viết một chương trình để tự động tải xuống thông tin về các dự án Python được đánh dấu sao nhiều nhất trên GitHub và sau đó chúng ta sẽ tạo hình ảnh trực quan đầy đủ thông tin về các dự án này.

Yêu cầu dữ liệu sử dụng lệnh gọi API

API của GitHub cho phép ta yêu cầu nhiều loại thông tin thông qua API call. Để xem lệnh gọi API trông như thế nào, hãy nhập thông tin sau vào thanh địa chỉ của trình duyệt và nhấn enter:

<https://api.github.com/search/repositories?q=language:python&sort=stars>

```
{
  "total_count": 7485398,
  "incomplete_results": false,
  "items": [
    {
      "id": 83222441,
      "node_id": "MDExO1JlcG9zaXRvcnk4MzIyMjQ0MQ==",
      "name": "system-design-primer",
      "full_name": "donnemartin/system-design-primer",
      "private": false,
      "owner": {
        "login": "donnemartin",
        "id": 5458997,
        "node_id": "MDQ6VXNlcjU0NTg5OTc=",
        "avatar_url": "https://avatars.githubusercontent.com/u/5458997?v=4",
        "gravatar_id": "",
        "url": "https://api.github.com/users/donnemartin",
        "html_url": "https://github.com/donnemartin",
        "followers_url": "https://api.github.com/users/donnemartin/followers",
        "following_url": "https://api.github.com/users/donnemartin/following{/other_user}",
        "gists_url": "https://api.github.com/users/donnemartin/gists{/gist_id}",
        "starred_url": "https://api.github.com/users/donnemartin/starred{/owner}{/repo}",
        "subscriptions_url": "https://api.github.com/users/donnemartin/subscriptions",
        "organizations_url": "https://api.github.com/users/donnemartin/orgs",
        "repos_url": "https://api.github.com/users/donnemartin/repos",
        "events_url": "https://api.github.com/users/donnemartin/events{/privacy}",
        "received_events_url": "https://api.github.com/users/donnemartin/received_events",
        "type": "User",
        "site_admin": false
      },
    }
  ]
}
```

Lệnh gọi này trả về số lượng dự án Python hiện được lưu trữ trên GitHub, cũng như thông tin về các kho lưu trữ Python phổ biến nhất. Ta kiểm tra cuộc gọi API. Phần đầu tiên, <https://api.github.com/>, hướng yêu cầu đến phần GitHub phản hồi các lệnh gọi API. Phần tiếp theo, tìm kiếm / kho lưu trữ, yêu cầu API tiến hành tìm kiếm thông qua tất cả các kho lưu trữ trên GitHub.

Dấu chấm hỏi sau kho lưu trữ báo hiệu rằng chúng ta sắp truyền một đối số. Q là viết tắt của truy vấn và dấu bằng (=) cho phép chúng ta bắt đầu chỉ định một truy vấn (q =). Bằng cách sử dụng ngôn ngữ: python, chúng ta cho biết rằng chúng ta muốn chỉ thông tin về các kho lưu trữ có Python là ngôn ngữ chính. Phần cuối cùng, & sort = stars, sắp xếp các dự án theo số lượng sao mà chúng đã được cung cấp.

Ta có thể thấy từ phản hồi rằng URL này không phải là mục đích chính được con người nhập vào, vì nó ở định dạng được một chương trình xử lý. GitHub đã tìm thấy 7.485.398 dự án Python tính đến thời điểm viết bài giảng này. Vì giá trị cho "incomplete_results" là false, chúng ta biết rằng yêu cầu đã thành công (nó không phải là không đầy đủ). Nếu GitHub không thể xử lý hoàn toàn yêu cầu API, thì nó sẽ trả về true tại đây. Các mục (items) trả về được hiển thị trong danh sách sau đó, chua thông tin chi tiết về các dự án Python phổ biến nhất trên GitHub.

Cài đặt Requests

Gói Requests cho phép một chương trình Python dễ dàng yêu cầu thông tin từ một trang web và kiểm tra phản hồi. Sử dụng pip để cài đặt Requests:

```
$ python -m pip install --user requests
```

Dòng này yêu cầu Python chạy mô-đun pip và cài đặt gói Requests vào thiết lập Python của người dùng hiện tại. Nếu ta sử dụng python3 hoặc một lệnh khác khi chạy chương trình hoặc cài đặt gói, hãy đảm bảo ta sử dụng cùng một lệnh như trên.

Xử lý API Response

Bây giờ chúng ta sẽ bắt đầu viết một chương trình để tự động đưa ra lệnh gọi API và xử lý kết quả bằng cách xác định các dự án Python được đánh dấu sao nhiều nhất trên GitHub:

```
import requests

# Make an API call and store the response.
url = 'https://api.github.com/search/repositories?q=language:python&sort=stars'
headers = {'Accept': 'application/vnd.github.v3+json'}
r = requests.get(url, headers=headers)
print(f"Status code: {r.status_code}")

# Store API response in a variable.
response_dict = r.json()
# Process results.
print(response_dict.keys())
```

Đầu tiên, chúng ta sẽ nhập module requests. Tiếp đó, chúng ta lưu trữ API vào biến url. GitHub hiện đang ở phiên bản thứ ba của API, vì vậy chúng ta xác định tiêu đề cho lệnh gọi API và lưu nó vào biến headers để cho biết API call sẽ sử dụng phiên bản này. Khi đã chuẩn bị xong, ta sẽ dùng requests để thực hiện cuộc gọi API bằng cách sử dụng phương thức get() và truyền vào đó hai tham số là url và headers và gán giá trị trả về vào biến r. Đối tượng phản hồi có một thuộc tính được gọi là status_code, cho chúng ta biết liệu yêu cầu có thực hiện thành công hay không. (Mã trạng thái 200 cho biết phản hồi thành công). Để chắc chắn, chúng ta in giá trị của status_code sau lời gọi để biết được trạng thái. API trả về thông tin ở định dạng JSON, vì vậy chúng ta sử dụng phương thức json() để chuyển đổi thông tin sang từ điển Python. Chúng ta lưu trữ từ điển kết quả trong response_dict.

Cuối cùng, chúng ta in các khóa từ response_dict và có kết quả như sau:

```
Status code: 200
dict_keys(['total_count', 'incomplete_results', 'items'])
```

Vì mã trạng thái là 200, chúng ta biết rằng yêu cầu đã thành công. Từ điển phản hồi chỉ chứa ba khóa: 'total_count', 'incomplete_results' và 'item'.

Ghi chú: với các lời gọi đơn giản như trên, ta có thể yên tâm khi bỏ qua kết quả lời gọi vì thường sẽ trả về đầy đủ. Tuy nhiên, với những lời gọi phức tạp, chương trình cần phải kiểm tra biến này.

Làm việc với từ điển Response

Với thông tin từ lệnh gọi API được lưu trữ dưới dạng từ điển, chúng ta có thể làm việc với dữ liệu được lưu trữ ở đó. Ta sẽ tạo một số đầu ra tóm tắt thông tin. Đây là một cách tốt để đảm bảo rằng ta đã nhận được thông tin mong đợi và bắt đầu kiểm tra thông tin mà chúng ta quan tâm:

```
import requests

# Make an API call and store the response.
url = 'https://api.github.com/search/repositories?q=language:python&sort=stars'
headers = {'Accept': 'application/vnd.github.v3+json'}
r = requests.get(url, headers=headers)
print(f"Status code: {r.status_code}")

# Store API response in a variable.
response_dict = r.json()
# Process results.
print(response_dict.keys())

print(f"Total repositories: {response_dict['total_count']}")
# Explore information about the repositories.
repo_dicts = response_dict['items']
print(f"Repositories returned: {len(repo_dicts)}")

# Examine the first repository.
repo_dict = repo_dicts[0]
print(f"\nKeys: {len(repo_dict)}")
for key in sorted(repo_dict.keys()):
    print(key)
```

Đầu tiên, chúng ta in giá trị được liên kết với 'total_count', đại diện cho tổng số kho lưu trữ Python trên GitHub. Giá trị được liên kết với 'items' là danh sách chứa một số từ điển, mỗi từ điển chứa dữ liệu về một kho lưu trữ Python riêng lẻ. Tiếp theo, chúng ta lưu trữ danh sách các từ điển trong biến repo_dicts và sau đó in ra độ dài của danh sách để biết được bao nhiêu kho chứa mà chúng ta có thông tin. Để xem xét kỹ hơn thông tin được trả về về từng kho lưu trữ, chúng ta lôi ra mục đầu tiên từ

repo_dicts và lưu trữ nó trong repo_dict. Sau đó chúng ta in số lượng khóa trong từ điển để xem chúng ta có bao nhiêu thông tin trong từ điển đó. Và cuối cùng, chúng ta dùng một vòng lặp để in ra tất cả các khóa ở trong từ điển.

```
Total repositories: 7486266
```

```
Repositories returned: 30
```

```
Keys: 74
```

```
archive_url
```

```
archived
```

```
assignees_url
```

```
blobs_url
```

```
-snip-
```

API của GitHub trả về nhiều thông tin về từng kho lưu trữ: có 74 khóa trong repo_dict. Khi nhìn qua các phím này, ta sẽ hiểu loại thông tin có thể trích xuất về một dự án. (Cách duy nhất để biết thông tin nào có sẵn thông qua API là đọc tài liệu hoặc kiểm tra thông tin thông qua mã, như chúng ta đang làm ở đây.)

Ta thử lấy ra giá trị của một vài khóa trong repo_dict:

```
repo_dict = repo_dicts[0]
print(f"\nKeys: {len(repo_dict)}")
for key in sorted(repo_dict.keys()):
    print(key)

print("\nSelected information about first repository:")
print(f"Name: {repo_dict['name']}")
print(f"Owner: {repo_dict['owner']['login']}")
print(f"Stars: {repo_dict['stargazers_count']}")
print(f"Repository: {repo_dict['html_url']}

print(f"Created: {repo_dict['created_at']}")
print(f"Updated: {repo_dict['updated_at']}")
print(f"Description: {repo_dict['description']}")
```

Tại đây, chúng ta in các giá trị cho một số khóa từ từ điển của kho lưu trữ thứ nhất. Đầu tiên, chúng ta in tên của dự án. Toàn bộ từ điển đại diện cho chủ sở hữu của dự án, vì vậy tiếp theo, chúng ta sử dụng khóa owner để truy cập từ điển đại diện cho chủ sở hữu, và sau đó sử dụng khóa login để lấy tên đăng nhập của chủ sở hữu.

Sau đó, chúng ta in bao nhiêu sao mà dự án đã kiếm được và URL cho kho lưu trữ GitHub của dự án. Sau đó, chúng ta hiển thị ngày nó được tạo và ngày nó được

cập nhật lần cuối cùng. Cuối cùng, chúng ta in mô tả của kho lưu trữ; đầu ra sẽ trông giống như sau:

```
Selected information about first repository:  
Name: system-design-primer  
Owner: donnemartin  
Stars: 136306  
Repository: https://github.com/donnemartin/system-design-primer  
Created: 2017-02-26T16:15:28Z  
Updated: 2021-06-24T08:41:25Z  
Description: Learn how to design large-scale systems. Prep for the system  
design interview. Includes Anki flashcards.
```

Chúng ta có thể thấy rằng dự án Python được đánh dấu sao nhiều nhất trên GitHub tính đến thời điểm bài giảng này là system-design-primer, chủ sở hữu của nó là người dùng donnemartin và nó đã được gắn dấu sao bởi hơn 130.000 người dùng GitHub. Chúng ta có thể thấy URL của dự án kho lưu trữ, ngày tạo của nó vào tháng 2 năm 2017 và nó đã được cập nhật gần đây. Mô tả của dự án cho biết đây là cách chúng ta học cách thiết kế một hệ thống có quy mô lớn.

Tổng hợp các kho hàng đầu

Khi chúng ta trực quan hóa dữ liệu này, chúng ta sẽ muốn bao gồm nhiều hơn một kho lưu trữ. Ta thử viết một vòng lặp để in thông tin đã chọn về từng kho lưu trữ mà lệnh gọi API trả về để chúng ta có thể đưa tất cả chúng vào đồ thị:

```
# Explore information about the repositories.  
repo_dicts = response_dict['items']  
print(f"Repositories returned: {len(repo_dicts)}")  
print("\nSelected information about each repository:")  
for repo_dict in repo_dicts:  
    print(f"\nName: {repo_dict['name']}")  
    print(f"Owner: {repo_dict['owner']['login']}")  
    print(f"Stars: {repo_dict['stargazers_count']}")  
    print(f"Repository: {repo_dict['html_url']}")  
    print(f"Description: {repo_dict['description']}")
```

Đoạn code trên, chúng ta in ra một dòng giới thiệu thông tin, sau đó chúng ta lặp qua tất cả các từ điển ở trong repo_dicts. Bên trong vòng lặp, chúng ta in ra tất cả các dự án, chủ của dự án, số sao, URL trên github và mô tả của dự án.

```
Name: system-design-primer  
Owner: donnemartin  
Stars: 136904
```

Repository: <https://github.com/donnemartin/system-design-primer>
Description: Learn how to design large-scale systems. Prep for the system design interview. Includes Anki flashcards.

Name: public-apis
Owner: public-apis
Stars: 132945
Repository: <https://github.com/public-apis/public-apis>
Description: A collective list of free APIs

Name: Python
Owner: TheAlgorithms
Stars: 110914
Repository: <https://github.com/TheAlgorithms/Python>
Description: All Algorithms implemented in Python

Name: Python-100-Days
Owner: jackfrued
Stars: 105271
Repository: <https://github.com/jackfrued/Python-100-Days>
Description: Python - 100天刷题计划

Name: youtube-dl
Owner: ytdl-org
Stars: 96859
Repository: <https://github.com/ytdl-org/youtube-dl>
Description: Command-line program to download videos from YouTube.com and other video sites

Name: models
Owner: tensorflow
Stars: 70398
Repository: <https://github.com/tensorflow/models>
Description: Models and examples built with TensorFlow

Name: thefuck
Owner: nvbn
Stars: 62753
Repository: <https://github.com/nvbn/thefuck>
Description: Magnificent app which corrects your previous console command.

Name: django
Owner: django
Stars: 58212

Repository: <https://github.com/django/django>

Description: The Web framework for perfectionists with deadlines.

Name: flask

Owner: pallets

Stars: 55858

Repository: <https://github.com/pallets/flask>

Description: The Python micro framework for building web applications.

Name: keras

Owner: keras-team

Stars: 51793

Repository: <https://github.com/keras-team/keras>

Description: Deep Learning for humans

Một số dự án thú vị xuất hiện trong các kết quả này và có thể thú vị khi xem một vài dự án. Nhưng đừng dành quá nhiều thời gian, vì chúng ta sẽ sớm tạo một hình ảnh trực quan giúp ta đọc kết quả dễ dàng hơn nhiều.

Giám sát giới hạn tốc độ API

Hầu hết các API đều bị giới hạn về tốc độ, có nghĩa là có giới hạn về số lượng yêu cầu ta có thể thực hiện trong một khoảng thời gian nhất định. Để xem liệu chúng ta có đang đến gần giới hạn của GitHub, hãy nhập https://api.github.com/rate_limit vào trình duyệt web. Ta sẽ thấy một phản hồi bắt đầu như sau:

```
{"resources": {"core": {"limit": 60, "remaining": 60, "reset": 1624963120, "used": 0, "resource": "core"}, "graphql": {"limit": 0, "remaining": 0, "reset": 1624963120, "used": 0, "resource": "graphql"}, "integration_manifest": {"limit": 5000, "remaining": 5000, "reset": 1624963120, "used": 0, "resource": "integration_manifest"}, "search": {"limit": 10, "remaining": 10, "reset": 1624959580, "used": 0, "resource": "search"}}, "rate": {"limit": 60, "remaining": 60, "reset": 1624963120, "used": 0, "resource": "core"}}
```

Thông tin chúng ta quan tâm là giới hạn tỷ lệ cho tìm kiếm API. Chúng ta thấy rằng giới hạn là 60 yêu cầu mỗi phút và chúng ta còn lại 60 yêu cầu cho phút hiện tại. Giá trị đặt lại đại diện cho thời gian trong Unix hoặc thời gian kỷ nguyên (reset). Nếu ta đạt đến hạn ngạch của mình, ta sẽ nhận được một phản hồi ngắn cho biết ta đã đạt đến giới hạn API. Nếu đạt đến giới hạn, chỉ cần đợi cho đến khi định mức được thiết lập lại.

Ghi chú: Nhiều API yêu cầu đăng ký và lấy khóa API để thực hiện các lệnh gọi API. Theo bài giảng này, GitHub không có yêu cầu như vậy, nhưng nếu ta nhận được khóa API, giới hạn sẽ cao hơn nhiều.

10.2.3.2. Trực quan hóa dữ liệu sử dụng Ploty

Ta trực quan hóa bằng cách sử dụng dữ liệu chúng ta có để hiển thị mức độ phổ biến của các dự án Python trên GitHub. Chúng ta sẽ tạo một biểu đồ thanh tương tác: chiều cao của mỗi thanh sẽ đại diện cho số sao mà dự án có được và ta có thể nhấp vào nhãn của thanh đó để chuyển đến trang chủ của dự án đó trên GitHub. Chúng ta bổ sung các đoạn mã cho yêu cầu trực quan hóa dữ liệu:

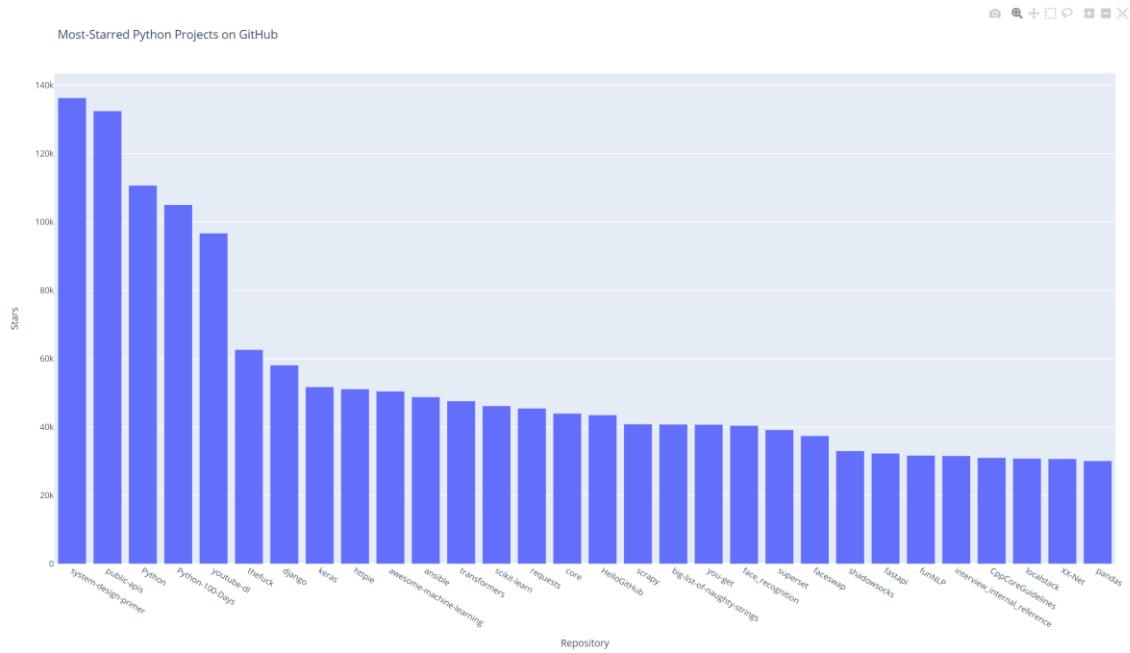
```
from plotly.graph_objs import Bar
from plotly import offline
repo_names, stars = [], []
for repo_dict in repo_dicts:
    repo_names.append(repo_dict['name'])
    stars.append(repo_dict['stargazers_count'])
data = [{"type": "bar",
          "x": repo_names,
          "y": stars,
        }]
my_layout = {
    'title': 'Most-Starred Python Projects on GitHub',
    'xaxis': {'title': 'Repository'},
    'yaxis': {'title': 'Stars'},
}
fig = {'data': data, 'layout': my_layout}
offline.plot(fig, filename='python_repos.html')
```

```
from plotly.graph_objs import Bar
from plotly import offline
data = [{"type": "bar",
          "x": repo_names,
          "y": stars,
          "marker": {
              'color': 'rgb(60, 100, 150)',
              'line': {'width': 1.5, 'color': 'rgb(25, 25, 25)'}
            },
          'opacity': 0.6,
        }]
my_layout = {
    'title': 'Most-Starred Python Projects on GitHub',
    'titlefont': {'size': 28},
    'xaxis': {'title': 'Repository',
```

```

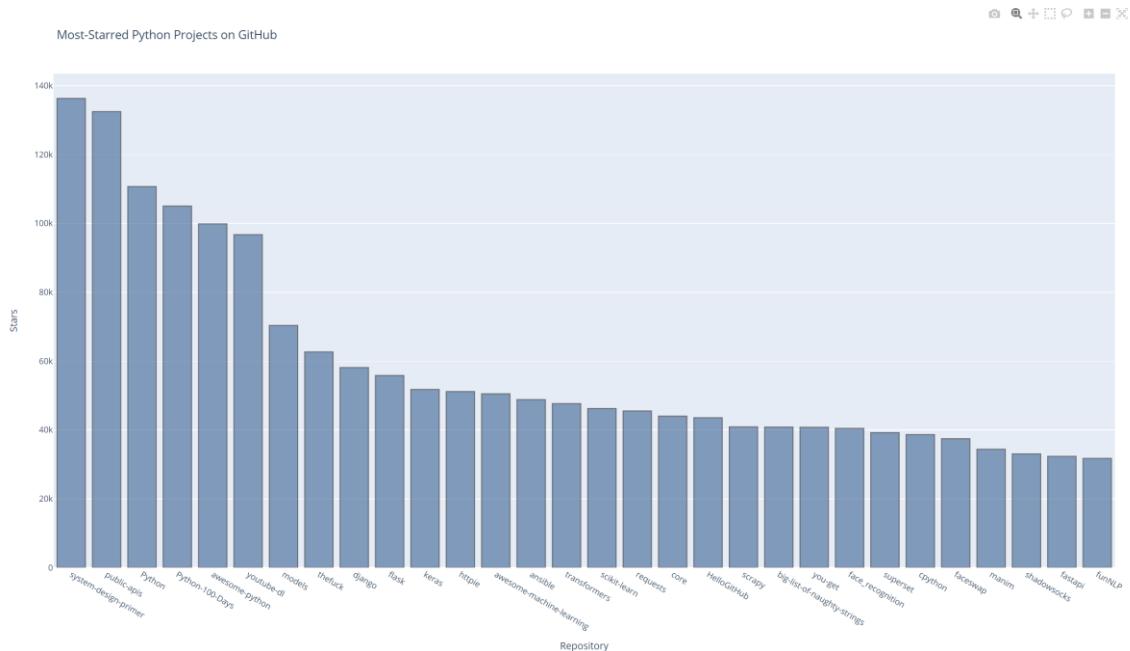
        'titlefont': {'size': 24},
        'tickfont': {'size': 14}, },
    'yaxis': {'title': 'Stars',
        'titlefont': {'size': 24},
        'tickfont': {'size': 14}, },
    }
fig = {'data': data, 'layout': my_layout}
offline.plot(fig, filename='python_repos.html')

```



Hình 10.33. Danh sách các kho hàng đầu trên Github

Tinh chỉnh biểu đồ Plotly



Hình 10.34. Tùy chỉnh biểu đồ Plotly

Thêm Tooltip tùy biến

Trong Plotly, ý có thể di con trỏ qua một thanh riêng lẻ để hiển thị thông tin mà thanh đó đại diện. Đây thường được gọi là công cụ chú giải(tooltip) và trong trường hợp này, nó hiện đang hiển thị số sao mà một dự án có. Ta sẽ tạo chú giải công cụ tùy chỉnh để hiển thị mô tả của từng dự án cũng như chủ sở hữu của dự án.

Chúng ta cần lấy thêm một số dữ liệu để tạo chú giải công cụ và sửa đổi đối tượng dữ liệu:

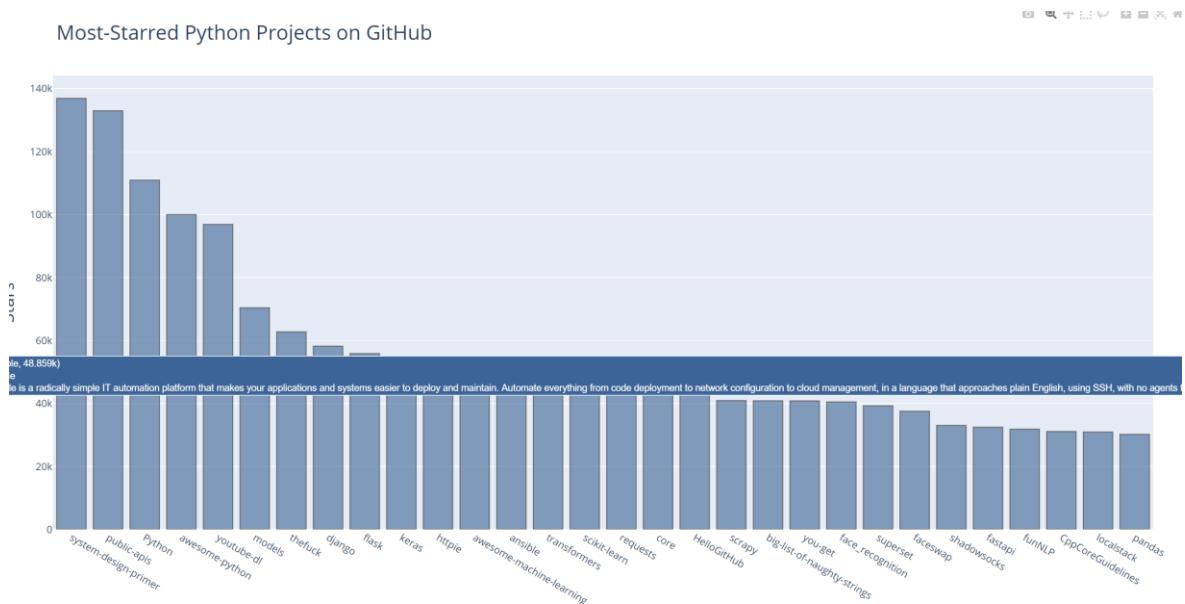
```
repo_names, stars, labels = [], [], []
for repo_dict in repo_dicts:
    repo_names.append(repo_dict['name'])
    stars.append(repo_dict['stargazers_count'])
    owner = repo_dict['owner']['login']
    description = repo_dict['description']
    label = f'{owner}<br />{description}'
    labels.append(label)

data = [
    {
        'type': 'bar',
        'x': repo_names,
        'y': stars,
        'hovertext': labels,
        'marker': {
            'color': 'rgb(60, 100, 150)',
            'line': {'width': 1.5, 'color': 'rgb(25, 25, 25)'}
        },
        'opacity': 0.6,
    }
]
my_layout = {
    'title': 'Most-Starred Python Projects on GitHub',
    'titlefont': {'size': 28},
    'xaxis': {'title': 'Repository',
              'titlefont': {'size': 24},
              'tickfont': {'size': 14}, },
    'yaxis': {'title': 'Stars',}
```

Đầu tiên, chúng ta tạo ra một danh sách labels để chứa các nhãn của các thanh của biểu đồ (tương ứng với mỗi dự án). Trong vòng for, chúng ta lấy ra giá trị của nhãn của các dự án qua khóa ‘description’ và định dạng nó dưới dạng HTML (thêm

ký tự br) để tách tên chủ sở hữu và phần mô tả. Sau đó, ta thêm nhãn đã được định dạng theo mã HTML vào trong danh sách labels.

Tiếp đó, ta thêm khóa ‘hoertext’ là danh sách các nhãn labels vào từ điển data. Khi plotly tạo ra các thanh (bar), nó sẽ lấy các nhãn từ danh sách labels này để hiển thị mỗi khi người dùng di chuột lên cách thanh đại diện cho dự án.



Hình 10.35. Thông tin khi di chuột

Thêm liên kết khả click vào biểu đồ

Vì Plotly cho phép sử dụng HTML trên các phần tử văn bản, chúng ta có thể dễ dàng thêm liên kết đến một biểu đồ. Ta sử dụng các nhãn trực x như một cách để cho phép người xem truy cập trang chủ của bất kỳ dự án nào trên GitHub. Chúng ta cần lấy các URL từ dữ liệu và sử dụng chúng khi tạo các nhãn trực x.

Tìm hiểu thêm về Plotly và API GitHub

10.2.3.3. The Hacker News API

Để khám phá cách sử dụng lệnh gọi API trên các trang web khác, chúng ta hãy xem nhanh Tin tức về Hacker (<http://news.ycombinator.com/>). Trên Hacker News, mọi người chia sẻ các bài báo về lập trình và công nghệ, đồng thời tham gia vào các cuộc thảo luận sôi nổi về các bài báo đó. Hacker News API cung cấp quyền truy cập

vào dữ liệu về tất cả các bài gửi và nhận xét trên trang web và ta có thể sử dụng API mà không cần phải đăng ký khóa.

Lời gợi sau trả về thông tin bài báo hàng đầu tại thời điểm bài giảng này được xây dựng:

<https://hacker-news.firebaseio.com/v0/item/19155826.json>

Khi nhập URL này vào trình duyệt, ta sẽ thấy văn bản trên trang được bao quanh bởi dấu ngoặc nhọn, có nghĩa là nó là một từ điển. Nhưng rất khó để kiểm tra phản hồi nếu không có một số định dạng tốt hơn. Chạy URL này thông qua phương thức json.dump(), giống như chúng ta đã làm trong dự án động đất ở chương trước, để chúng ta có thể khám phá loại thông tin được trả về về một bài báo:

```
import requests
import json

# Make an API call, and store the response.
url = 'https://hacker-news.firebaseio.com/v0/item/19155826.json'
r = requests.get(url)
print(f"Status code: {r.status_code}")

# Explore the structure of the data.
response_dict = r.json()
readable_file = 'data/readable_hn_data.json'
with open(readable_file, 'w') as f:
    json.dump(response_dict, f, indent=4)
```

Các nội dung trong đoạn code là khá quen thuộc vì chúng ta đã thực hiện nó trong các chương trên. Dữ liệu lưu vào tệp theo định dạng json:

```
{
  "by": "jimktrains2",
①  "descendants": 220,
  "id": 19155826,
②  "kids": [
      19156572,
      19158857,
      --snip--
    ],
  "score": 722,
  "time": 1550085414,
③  "title": "Nasa's Mars Rover Opportunity Concludes a 15-Year Mission",
  "type": "story",
④  "url": "https://www.nytimes.com/.../mars-opportunity-rover-dead.html"
}
```

Tệp json này chứa một số khóa (keys) mà chúng ta có thể làm việc với như: khóa descendants nói về số lượng bình luận của bài báo có, khóa kids cung cấp ID của tất cả các bình luận, một số khóa khác như title, url, time, score.

10.2.3.4. Tiểu kết

Trong phần này, ta đã học cách sử dụng API để viết các chương trình độc lập tự động thu thập dữ liệu cần thiết và sử dụng dữ liệu đó để tạo hình ảnh trực quan. Ta đã sử dụng API GitHub để khám phá các dự án Python được đánh dấu sao nhiều nhất trên GitHub và cũng đã xem qua API Hacker News.

Ta đã học cách sử dụng gói Request để tự động đưa ra lệnh gọi API tới GitHub và cách xử lý kết quả của lệnh gọi đó. Một số cài đặt Plotly cũng được giới thiệu để tùy chỉnh thêm giao diện của các biểu đồ được tạo ra.

Ở phần tiếp theo, chúng ta sẽ sử dụng Django để xây dựng ứng dụng web cho các dự án được yêu cầu.

10.3. Ứng dụng web

10.3.1. Bắt đầu với Django

Những ứng dụng web hiện nay có thể hoạt động tốt tương tự như những ứng dụng cho máy tính để bàn. Ngôn ngữ Python có một công cụ rất tuyệt vời để phát triển các ứng dụng Web, đó là Django. Django là một Web Framework - một bộ công cụ thiết kế giúp chúng ta xây dựng các website tương tác với người dùng. Trong phần này, ta sẽ được học cách sử dụng Django ([https://django-project.com/](https://.djangoproject.com/)) để xây dựng một ứng dụng gọi là Learning Log - một hệ thống nhật ký trực tuyến giúp lưu trữ thông tin mà chúng ta đã học được về các chủ đề cụ thể.

Chúng ta sẽ viết mô tả cho dự án này, và sẽ xác định các mô hình cho những phần dữ liệu mà ứng dụng sử dụng. Chúng ta sẽ sử dụng hệ thống quản trị của Django để thêm một vài dữ liệu khi khởi tạo, và ta sẽ học cách viết các giao diện cũng như các mẫu để Django có thể xây dựng các trang web.

Django có thể phản hồi các yêu cầu của trang web và giúp công việc đọc và ghi dữ liệu vào cơ sở dữ liệu, quản lý người dùng và các tính năng khác. Trong phần tiếp theo, ta sẽ tinh chỉnh dự án Learning Log và sau đó triển khai nó lên một máy chủ trực tuyến để chúng ta(và ta bè) có thể sử dụng nó.

10.3.1.1. Thiết lập dự án

Khi bắt đầu một dự án, trước tiên cần mô tả dự án đó trong bản mô tả chi tiết, hoặc ‘spec’ (viết tắt của ‘specification’). Sau đó, chúng ta sẽ thiết lập một môi trường ảo để xây dựng dự án.

Viết bản mô tả chi tiết

Một bản mô tả chi tiết các mục tiêu của dự án, mô tả chức năng của dự án và thảo luận về giao diện người dùng của dự án. Giống như bất kỳ dự án hoặc kế hoạch kinh doanh tốt nào, một bản mô tả sẽ giúp ta tập trung và giúp dự án của ta đi đúng hướng. Chúng ta sẽ không viết một bản mô tả đầy đủ của dự án ở đây, nhưng ta sẽ đưa ra một vài mục tiêu rõ ràng để giữ cho quá trình phát triển được tập trung. Đây là bản mô tả mà chúng ta sẽ sử dụng:

Chúng ta sẽ viết một ứng dụng web có tên là Nhật ký học tập cho phép người dùng ghi lại các chủ đề họ quan tâm và ghi nhật ký khi họ tìm hiểu về từng chủ đề. Trang chủ Nhật ký học tập sẽ mô tả trang web và mời người dùng đăng ký hoặc đăng nhập. Sau khi đăng nhập, người dùng có thể tạo chủ đề mới, thêm mục nhập mới, đọc và chỉnh sửa các mục hiện có.

Khi ta tìm hiểu về một chủ đề mới, hãy ghi nhật ký về những gì ta đã học có thể hữu ích trong việc theo dõi và xem lại thông tin. Một ứng dụng tốt làm cho quá trình này hiệu quả.

Tạo môi trường ảo

Để làm việc với Django, trước tiên, chúng ta hãy thiết lập một môi trường ảo. Môi trường ảo là một nơi trên hệ thống nơi ta có thể cài đặt các gói và cài đặt chúng từ tất cả các gói Python khác. Việc tách các thư viện của một dự án khỏi các dự án khác là có lợi và sẽ cần thiết khi chúng ta triển khai Nhật ký học tập cho một máy chủ.

Tạo một thư mục mới cho dự án của ta có tên là learning_log, chuyển đổi

Vào thư mục đó trong một thiết bị đầu cuối và nhập mã sau để tạo một môi trường ảo:

```
learning_log$ python -m venv ll_env  
learning_log$
```

Ở đây, chúng ta đang chạy mô-đun môi trường ảo venv và sử dụng nó để tạo một môi trường ảo có tên ll_env (lưu ý rằng đây là ll_env với hai chữ L viết thường,

không phải hai chữ cái). Nếu ta sử dụng lệnh như python3 khi chạy chương trình hoặc cài đặt gói, hãy đảm bảo sử dụng lệnh đó tại đây.

Kích hoạt môi trường ảo

Bây giờ chúng ta cần kích hoạt môi trường ảo bằng cách sử dụng yêu cầu sau:

```
learning_log$ source ll_env/bin/activate  
(ll_env)learning_log$
```

Lệnh này chạy các tập lệnh kích hoạt trong ll_env/bin. Khi môi trường đang hoạt động, ta sẽ thấy tên của môi trường như đã hiện thị trong dấu ngoặc đơn, vì vậy ta có thể cài đặt các gói vào môi trường và sử dụng các gói đã được cài đặt. Các gói ta cài đặt trong ll_env sẽ chỉ khả dụng khi môi trường đang hoạt động.

Chú ý: Nếu ta đang sử dụng Windows, hãy sử dụng lệnh ll_env\Scripts\activate (không có từ “source”) để kích hoạt môi trường ảo. Nếu ta đang sử dụng PowerShell, ta cần viết hoa từ “Activate”.

Để dừng sử dụng môi trường ảo, dùng ”deactivate” :

```
(ll_env)learning_log$  
deactivate learning_log$
```

Môi trường cũng sẽ không hoạt động khi ta đóng thiết bị đầu cuối khi mà nó đang chạy.

Cài đặt Django

Khi môi trường ảo được kích hoạt, hãy nhập thông tin sau để cài đặt Django:

```
(ll_env)learning_log$ pip install django  
Collecting django  
--snip--  
Installing collected packages: pytz, django  
Successfully installed django-2.2.0 pytz-2018.9 sqlparse-0.2.4  
(ll_env)learning_log$
```

Vì chúng ta đang làm việc trong một môi trường ảo, là môi trường tự kiểm soát của riêng nó, nên lệnh này giống nhau trên tất cả các hệ thống. Không cần sử dụng cờ

--user và không cần sử dụng các lệnh dài hơn chặng hạn như python -m pip install package_name.

Hãy nhớ rằng Django sẽ chỉ khả dụng khi môi trường ll_env đang hoạt động.

Note: Django phát hành phiên bản mới khoảng tầm tháng một lần, vì vậy ta có thể thấy phiên bản mới hơn khi ta cài đặt Django. Dự án này rất có thể sẽ hoạt động như được viết ở đây, ngay cả trên các phiên bản mới hơn của Django. Nếu ta muốn đảm bảo sử dụng cùng một phiên bản Django như đã thấy ở đây, hãy sử dụng lệnh pip install django == 2.2.*. Thao tác này sẽ cài đặt bản phát hành mới nhất của Django 2.2. Nếu gặp bất kỳ vấn đề nào liên quan đến phiên bản ta đang sử dụng, hãy xem tài liệu trực tuyến cho cuốn sách tại <https://nostarch.com/pythoncrashcourse2e/>.

Tạo một dự án trong Django

Không rời khỏi môi trường ảo đang hoạt động (hãy nhớ tìm dấu ngoặc đơn ll_envin trong dấu nhắc đầu cuối), hãy nhập các lệnh sau để tạo một dự án mới:

```
(ll_env)learning_log$ django-admin startproject learning_log .
(ll_env)learning_log$ ls
learning_log ll_env manage.py
(ll_env)learning_log$ ls learning_log
__init__.py settings.py urls.py wsgi.py
```

Lệnh đầu tiên yêu cầu Django thiết lập một dự án mới có tên là learning_log. Dấu chấm ở cuối lệnh tạo dự án mới với cấu trúc thư mục giúp dễ dàng triển khai ứng dụng tới máy chủ khi chúng ta phát triển xong.

Chú ý: Đừng quên dấu chấm này, nếu không ta có thể gặp một số vấn đề về cấu hình khi triển khai ứng dụng. Nếu ta quên dấu chấm, hãy xóa các tệp và thư mục đã được tạo (ngoại trừ ll_env) và chạy lại lệnh.

Chạy lệnh ls (dir trên Windows) cho thấy Django đã tạo một thư mục mới có tên learning_log. Nó cũng tạo ra một tệp manage.py, đây là một chương trình ngắn lấy các lệnh và cung cấp chúng cho phần có liên quan của Django để chạy chúng. Chúng ta sẽ sử dụng các lệnh này để quản lý các tác vụ, chẳng hạn như làm việc với cơ sở dữ liệu và máy chủ đang chạy.

Thư mục learning_log chứa bốn tệp quan trọng nhất là settings.py, urls.py và wsgi.py. Tệp settings.py kiểm soát cách Django hoạt động với hệ thống và quản lý dự án của ta. Chúng ta sẽ sửa đổi một vài thứ của cài đặt này và thêm một số cài đặt của riêng khi chúng ta phát triển dự án. TệpUrls.py của Django biết những trang nào cần tạo để đáp ứng các yêu cầu của trình duyệt. Tệp wsgi.py giúp Django phân phát

các tệp mà nó tạo ra. Tên tệp là từ viết tắt của giao diện cổng máy chủ web (*web sever gateway interface*).

Tạo Cơ sở dữ liệu

Django lưu trữ hầu hết thông tin của một dự án trong cơ sở dữ liệu, vì vậy tiếp theo chúng ta cần tạo một cơ sở dữ liệu mà Django có thể làm việc. Nhập lệnh sau (vẫn trong môi trường hoạt động):

```
(11_env)learning_log$ python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  --snip--
  Applying sessions.0001_initial... OK
(11_env)learning_log$ ls
db.sqlite3 learning_log 11_env manage.py
```

Bất cứ khi nào sửa đổi cơ sở dữ liệu, chúng ta nói rằng chúng ta đang di chuyển cơ sở dữ liệu. Việc phát hành lệnh `migrate` lần đầu tiên cho Django biết để đảm bảo cơ sở dữ liệu khớp với trạng thái hiện tại của dự án. Lần đầu tiên chúng ta chạy lệnh này trong một dự án mới sử dụng SQLite (sẽ tìm hiểu thêm về SQLite sau), Django sẽ tạo một cơ sở dữ liệu mới cho chúng ta. Django báo cáo rằng nó sẽ chuẩn bị cơ sở dữ liệu để lưu trữ thông tin cần thiết để xử lý các tác vụ quản trị và xác thực.

Chạy lệnh `ls` cho thấy Django đã tạo một tệp khác có tên `db.sqlite`. SQLite là một cơ sở dữ liệu chạy trên một tệp duy nhất; lý tưởng để ghi các ứng dụng đơn giản vì ta sẽ không phải quan tâm nhiều đến việc quản lý kho dữ liệu.

Chú ý: Trong môi trường ảo đang hoạt động, hãy sử dụng cmd `python` để chạy các lệnh `management.py`, ngay cả khi ta sử dụng thứ gì đó khác, như `python3`, để chạy các chương trình khác. Trong một môi trường ảo, lệnh `python` cmd để cập đến phiên bản Python đã tạo môi trường ảo.

Rà soát dự án

Hãy đảm bảo rằng Django đã thiết lập dự án đúng cách. Nhập lệnh `runserver` như sau để xem dự án ở trạng thái hiện tại của nó:

```
(11_env)learning_log$ python manage.py runserver
```

```
Watchman unavailable: pywatchman not installed.  
Watching for file changes with StatReloader  
Performing system checks...  
System check identified no issues (0 silenced).  
February 18, 2019 - 16:26:07  
Django version 2.2.0, using settings 'learning_log.settings'  
Starting development server at http://127.0.0.1:8000/  
Quit the server with CONTROL-C.
```

Django nên khởi động một máy chủ được gọi là development server, vì vậy ta có thể xem dự án trên hệ thống để xem nó hoạt động tốt như thế nào. Khi ta yêu cầu một trang bằng cách nhập URL vào trình duyệt, máy chủ Django sẽ phản hồi điều đó yêu cầu bằng cách xây dựng trang thích hợp và gửi đến trình duyệt.

Đầu tiên Django kiểm tra để đảm bảo rằng dự án được thiết lập đúng cách; nó báo cáo phiên bản Django đang được sử dụng và tên của tệp cài đặt đang được sử dụng; và URL nơi dự án đang được phục vụ. URL `http://127.0.0.1:8000/` cho biết rằng dự án đang lắng nghe các yêu cầu trên cổng 8000 trên máy tính, được gọi là máy chủ cục bộ. Thuật ngữ localhost đề cập đến một máy chủ chỉ xử lý các yêu cầu trên hệ thống của ta; nó không cho phép bất kỳ ai khác xem các trang ta đang phát triển.

Mở trình duyệt web và nhập URL `http://localhost:8000/` hoặc `http://127.0.0.1:8000/` nếu cái đầu tiên không hoạt động. Ta sẽ thấy một cái gì đó giống như Hình 18-1, một trang mà Django tạo để cho ta biết tất cả đều hoạt động bình thường cho đến nay. Hiện máy chủ đang thực thi trên cổng 8000, nhưng khi muốn dừng máy chủ, nhấn `ctrl-C` trong thiết bị đầu cuối nơi lệnh trình chạy được phát hành.

django View release notes for Django 4.0



The install worked successfully! Congratulations!

You are seeing this page because DEBUG=True is in your settings file and you have not configured any URLs.

Hình 10.36. Giao diện cài đặt Django thành công

Chú ý : Nếu ta nhận được thông báo lỗi “Cổng đó đã được sử dụng”, hãy yêu cầu Django sử dụng một cổng khác bằng cách nhập “python management.py runserver 8001”, sau đó chuyển qua các số cao hơn cho đến khi ta tìm thấy một cổng đang mở.

Ghi chú

Dự án mới: Để hiểu rõ hơn về những gì Django làm, hãy xây dựng một vài dự án trống và xem những gì Django tạo ra. Tạo một thư mục mới với một tên đơn giản, như snap_gram hoặc insta_chat (bên ngoài thư mục learning_log của ta), điều hướng đến thư mục đó trong một thiết bị đầu cuối và tạo một môi trường ảo. Cài đặt Django và chạy lệnh django-admin.py startproject snap_gram. (đảm bảo ta bao gồm dấu chấm ở cuối lệnh).

Xem các tệp và thư mục lệnh này tạo và so sánh chúng với Learning Log. Làm điều này một vài lần cho đến khi ta quen với những gì Django tạo ra khi bắt đầu một dự án mới. Sau đó xóa các thư mục dự án nếu ta muốn.

10.3.1.2. Khởi động một ứng dụng

Dự án Django được tổ chức như một nhóm các ứng dụng riêng lẻ hoạt động cùng nhau để làm cho dự án hoạt động chung. Hiện tại, chúng ta sẽ chỉ tạo một ứng dụng để thực hiện hầu hết công việc của dự án. Chúng ta sẽ thêm một ứng dụng khác vào Chương tiếp theo để quản lý tài khoản người dùng.

Mở một cửa sổ đầu cuối (hoặc tab) mới và điều hướng đến thư mục có chứa management.py. Kích hoạt môi trường ảo, sau đó chạy lệnh startapp:

```
learning_log$ source ll_env/bin/activate
(ll_env)learning_log$ python manage.py startapp learning_logs
(ll_env)learning_log$ ls
db.sqlite3 learning_log learning_logs ll_env manage.py
(ll_env)learning_log$ ls learning_logs/
__init__.py admin.py apps.py migrations models.py tests.py
          views.py
```

Lệnh startapp appname yêu cầu Django tạo cơ sở hạ tầng cần thiết để xây dựng một ứng dụng. Khi nhìn vào thư mục dự án ngay bây giờ, ta sẽ thấy một thư mục mới có tên là learning_logs. Mở thư mục đó để xem Django đã tạo ra như thế nào. Các tệp quan trọng nhất là models.py, admin.py và views.py. Chúng ta sẽ sử dụng models.py để xác định dữ liệu mà chúng ta muốn quản lý trong ứng dụng của mình. Ta sẽ xem xét admin.py và views.py một chút về sau.

Xác định mô hình

Mỗi người dùng sẽ cần tạo một số chủ đề trong nhật ký học tập của họ. Mỗi mục mà họ thực hiện sẽ gắn liền với một chủ đề và những mục này sẽ được hiển thị dưới dạng văn bản. Chúng ta cũng sẽ cần lưu trữ nhãn thời gian của từng mục nhập, vì vậy ta có thể cho người dùng biết khi họ thực hiện từng lối vào.

Mở tệp models.py và xem nội dung hiện có của nó:

Models.py

```
from django.db import models
# Create your models here.
```

Module có tên là models đang được nhập cho chúng ta và chúng ta được mời tạo các mô hình của riêng mình. Một mô hình cho Django biết cách làm việc với dữ liệu sẽ được lưu trữ trong ứng dụng. Về mặt mã, một mô hình chỉ là một lớp; nó có các thuộc tính và phương thức, giống như mọi lớp mà chúng ta đã thảo luận. Dưới đây là mô hình cho các chủ đề mà người dùng sẽ lưu trữ:

```
from django.db import models
# Create your models here.
class Topic(models.Model):
    """A topic the user is learning about."""
    text = models.CharField(max_length=200)
    date_added = models.DateTimeField(auto_now_add=True)
    def __str__(self):
        """Return a string representation of the model."""
        return self.text
```

Chúng ta tạo một lớp có tên Topic, lớp này kế thừa từ Model — một lớp cha có trong Django để xác định chức năng cơ bản của mô hình. Thêm hai thuộc tính vào lớp Topic: text và date_added.

Thuộc tính văn bản là CharField — một phần dữ liệu được tạo thành từ các bộ truyền ký tự hoặc văn bản. Ta sử dụng CharField khi muốn lưu trữ một lượng nhỏ văn bản, chẳng hạn như tên, tiêu đề hoặc thành phố. Khi xác định một thuộc tính CharField, chúng ta phải cho Django biết nó nên dự trữ bao nhiêu dung lượng trong cơ sở dữ liệu. Ở đây, ta cung cấp cho nó độ dài tối đa là 200 ký tự, đủ để chứa hầu hết các tên chủ đề.

Thuộc tính date_added là DateTimeField — một phần dữ liệu sẽ ghi lại ngày và giờ . Chúng ta chuyển đổi số auto_now_add = True, yêu cầu Django tự động đặt thuộc tính này thành ngày và giờ hiện tại bất cứ khi nào người dùng tạo chủ đề mới.

Chú ý : Để xem các loại trường khác nhau mà ta có thể sử dụng trong một mô hình,hãy tham khảo Trường Mô hình Django tại <https://docs.djangoproject.com/en/2.2/ref/models/fields/>. Ta sẽ không cần tất cả thông tin ngay bây giờ, nhưng nó sẽ cực kỳ hữu ích khi ta đang phát triển ứng dụng của riêng mình.

Chúng ta cho Django biết thuộc tính nào sẽ sử dụng theo mặc định khi nó hiển thị thông tin về một chủ đề. Django gọi một phương thức __str__() để hiển thị một biểu diễn đơn giản của một mô hình. Ở đây ta viết một phương thức __str__() trả về chuỗi được lưu trữ trong thuộc tính văn bản.

Kích hoạt mô hình

Để sử dụng các mô hình, chúng ta phải yêu cầu Django đưa ứng dụng vào dự án tổng thể. Mở settings.py (trong thư mục learning_log / learning_log); ta sẽ thấy một phần, nó sẽ nói cho Django biết những ứng dụng nào đã được cài đặt và hoạt động cùng nhau trong dự án:

```
--snip--  
INSTALLED_APPS = [  
    'django.contrib.admin',  
    'django.contrib.auth',  
    'django.contrib.contenttypes',  
    'django.contrib.sessions',  
    'django.contrib.messages',  
    'django.contrib.staticfiles',  
]  
--snip--
```

Thêm ứng dụng của chúng ta vào danh sách này bằng cách sửa đổi INSTALLED_APPS để nó trông giống như sau:

```
-snip--  
INSTALLED_APPS = [  
    # My apps  
    'learning_logs',  
    # Default django apps.  
    'django.contrib.admin',
```

```
--snip--
```

```
]
```

```
--snip--
```

Nhóm các ứng dụng lại với nhau trong một dự án giúp theo dõi chúng khi dự án phát triển để bao gồm nhiều ứng dụng hơn. Ở đây chúng ta bắt đầu một phần có tên Ứng dụng của tôi, hiện chỉ bao gồm learning_logs. Điều quan trọng là phải đặt ứng dụng của riêng mình trước ứng dụng mặc định trong trường hợp ta cần ghi đè bất kỳ hành vi nào của ứng dụng mặc định bằng hành vi tùy chỉnh của riêng mình.

Tiếp theo, chúng ta cần yêu cầu Django sửa đổi cơ sở dữ liệu để nó có thể lưu trữ thông tin liên quan đến mô hình Topic. Từ thiết bị đầu cuối, chạy lệnh sau:

```
(11_env)learning_log$python manage.py makemigrations learning_logs
Migrations for 'learning_logs':
learning_logs/migrations/0001_initial.py
- Create model Topic
(11_env)learning_log$
```

Lệnh makemigrations yêu cầu Django tìm ra cách sửa đổi cơ sở dữ liệu để nó có thể lưu trữ dữ liệu được liên kết với bất kỳ mô hình mới nào mà chúng ta xác định. Kết quả ở đây cho thấy Django đã tạo một tệp có tên 0001_initial.py. Việc di chuyển này sẽ tạo một bảng cho mô hình Topic trong cơ sở dữ liệu.

Bây giờ chúng ta sẽ áp dụng quá trình di chuyển này và yêu cầu Django sửa đổi cơ sở dữ liệu:

```
(11_env)learning_log$ python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, learning_logs, sessions
Running migrations:
  Applying learning_logs.0001_initial... OK
```

Bất cứ khi nào chúng ta muốn sửa đổi dữ liệu trong quản lý Nhật ký học tập, chúng ta sẽ thực hiện theo ba bước sau: sửa đổi models.py, gọi makemigrations trên learning_logs và yêu cầu Django di chuyển dự án.

Trang quản trị của Django

Django giúp ta dễ dàng làm việc với các mô hình của mình thông qua trang web quản trị. Chỉ quản trị viên của trang web mới sử dụng trang quản trị, không phải người dùng thông thường. Trong phần này, chúng ta sẽ thiết lập trang web quản trị và sử dụng nó để thêm một số chủ đề thông qua Mô hình Topic.

Thiết lập một Superuser

Django cho phép ta tạo một superuser, một người dùng có tất cả các đặc quyền có sẵn trên trang web. Đặc quyền của người dùng kiểm soát các hành động mà họ có thể thực hiện. Các cài đặt đặc quyền hạn chế nhất cho phép người dùng chỉ đọc thông tin công khai trên trang web.

Họ thường có đặc quyền đọc dữ liệu riêng tư của họ và một số thông tin chỉ dành cho thành viên. Để quản trị một ứng dụng web một cách hiệu quả, chủ sở hữu trang web thường cần quyền truy cập vào tất cả thông tin được lưu trữ trên trang web. Một quản trị viên giỏi luôn cẩn thận với thông tin nhạy cảm của người dùng vì người dùng tin tưởng rất nhiều vào các ứng dụng mà họ truy cập.

Để tạo một superuser, hãy nhập lệnh sau:

```
(11_env)learning_log$ python manage.py createsuperuser
Username (leave blank to use 'eric'): 11_admin
Email address:
Password:
Password (again):
Superuser created successfully.
(11_env)learning_log$
```

Khi ta ra lệnh createuperuser, Django sẽ nhắc chúng ta nhập tên người dùng cho superuser . Ở đây ta đang sử dụng ll_admin, nhưng ta có thể nhập bất kỳ tên người dùng nào mà ta muốn. Có thể nhập địa chỉ email nếu muốn hoặc chỉ để trống phần này .Và ta cần nhập mật khẩu của mình hai lần.

Chú ý: Một số thông tin nhạy cảm có thể bị ẩn khỏi quản trị viên của trang web. Ví dụ: Django không lưu trữ mật khẩu ta nhập; thay vào đó, nó lưu trữ một chuỗi bắt nguồn từ mật khẩu, được gọi là băm. Mỗi lần ta nhập mật khẩu của mình, Django sẽ băm mục nhập của ta và so sánh nó với hàm băm được lưu trữ. Nếu hai hàm băm khớp nhau, mật khẩu được xác thực. Bằng cách yêu cầu các hàm băm phải khớp, nếu kẻ tấn công có quyền truy cập vào cơ sở dữ liệu của trang web, chúng sẽ có thể đọc các hàm băm được lưu trữ của nó nhưng không đọc được mật khẩu. Khi một trang web được thiết lập đúng cách, hầu như không thể lấy được mật khẩu ban đầu từ các hàm băm.

Đăng ký Mô hình với Trang web Quản trị viên

Django tự động bao gồm một số mô hình trong trang web quản trị, chẳng hạn như Nhóm và người dùng, nhưng các mô hình chúng ta tạo cần phải được thêm theo cách thủ công.

Khi chúng ta bắt đầu ứng dụng learning_logs, Django đã tạo một tệp admin.py trong cùng thư mục với models.py. Để đăng ký Chủ đề với trang quản trị, hãy nhập như sau:

```
from django.contrib import admin
from .models import Topic
admin.site.register(Topic)
```

Đầu tiên mã này nhập mô hình Topic mà chúng ta muốn đăng ký. Đầu chấm phía trước các mô hình yêu cầu Django tìm kiếm các models.py trong cùng thư mục với admin.py. Mã admin.site.register() yêu cầu Django quản lý mô hình của chúng ta thông qua trang web quản trị.

Bây giờ sử dụng tài khoản superuser để truy cập trang quản trị. Truy cập http://localhost: 8000 / admin /, nhập tên người dùng và mật khẩu cho superuser mà ta vừa tạo. Bạn sẽ thấy một màn hình như trong Hình 18-2. Trang này cho phép ta thêm người dùng và nhóm mới và thay đổi những người hiện có. Ta cũng có thể làm việc với dữ liệu liên quan đến mô hình Topic mà chúng ta vừa xác định.



Hình 10.37. Giao diện quản trị

Chú ý : Nếu thấy thông báo trong trình duyệt của mình rằng trang web không khả dụng, hãy đảm bảo rằng ta vẫn có máy chủ Django đang chạy trong cửa sổ đầu cuối. Nếu không, hãy kích hoạt một môi trường ảo và phát hành lại lệnh python management.py runserver. Nếu ta gặp sự cố khi xem dự án của mình tại bất kỳ

thời điểm nào trong quá trình phát triển, hãy đóng thiết bị đầu cuối đang mở và viết lại lệnh runningserver là bước khắc phục sự cố đầu tiên.

Thêm chủ đề

Bây giờ Chủ đề đã được đăng ký với trang quản trị, hãy thêm chủ đề đầu tiên của chúng ta. Nhập vào **Topic** để chuyển đến trang Chủ đề, trang này hầu như trống vì chúng ta chưa có chủ đề để quản lý. Nhập vào **Add Topic** và một biểu mẫu để thêm chủ đề mới xuất hiện. Nhập **Chess** vào hộp đầu tiên và nhấp vào **Save**. Ta sẽ được đưa trở lại trang quản trị Chủ đề và sẽ thấy chủ đề ta vừa tạo.

Hãy tạo chủ đề thứ hai để chúng ta sẽ có nhiều dữ liệu hơn để làm việc. Nhập vào Add Topic một lần nữa và nhập Rock Climbing. Nhập vào Lưu và ta sẽ được đưa trở lại trang Chủ đề chính. Bây giờ ta sẽ thấy Chess và Rock Climbing.

Xác định mô hình đầu vào (Entry Model)

Để người dùng ghi lại những gì họ đã học về cờ vua và leo núi, chúng ta cần xác định một mô hình cho các loại mục nhập mà người dùng có thể thực hiện trong nhật ký học tập của họ. Mỗi mục nhập cần được liên kết với một đề tài. Mỗi quan hệ này được gọi là mối quan hệ nhiều-một, có nghĩa là nhiều mục nhập được liên kết với một chủ đề.

Đây là mã cho mô hình Entry. Đặt nó vào tệp models.py:

```
from django.db import models
class Topic(models.Model):
    --snip--
class Entry(models.Model):
    """Something specific learned about a topic."""
    topic = models.ForeignKey(Topic, on_delete=models.CASCADE)
    text = models.TextField()
    date_added = models.DateTimeField(auto_now=True)

    class Meta:
        verbose_name_plural = 'entries'
    def __str__(self):
        """Return a string representation of the model."""
        return f"{self.text[:50]}..."
```

Lớp Entry kế thừa từ lớp Model của Django. Thuộc tính đầu tiên là topic- chủ đề, là một cá thể ForeignKey- khóa ngoại . Một khóa ngoại là một thuật ngữ cơ sở dữ liệu; nó là một tham chiếu đến một bản ghi khác trong cơ sở dữ liệu. Đây là mã kết

nối mỗi mục nhập với một chủ đề cụ thể. Mỗi chủ đề được chỉ định một khóa hoặc ID, khi nó được tạo. Khi Django cần thiết lập kết nối giữa hai phần dữ liệu, nó sử dụng khóa được liên kết với mỗi phần thông tin. Chúng ta sẽ sớm sử dụng các kết nối này để truy xuất tất cả các mục nhập có liên quan đến một chủ đề nhất định. Đối số on_delete = models.CASCADE cho Django biết rằng khi một chủ đề bị xóa, tất cả các mục nhập liên quan đến chủ đề đó cũng nên bị xóa. Đây được gọi là xóa theo tầng.

Tiếp theo là một thuộc tính được gọi là văn bản, là một thể hiện của TextField. Loại trường này không cần giới hạn kích thước vì chúng ta không muốn giới hạn kích thước của các mục nhập riêng lẻ. Thuộc tính date_added cho phép chúng ta trình bày các mục nhập theo thứ tự chúng được tạo và đặt dấu thời gian bên cạnh mỗi mục nhập.

Sau đó chúng ta lồng lớp Meta vào bên trong lớp Entry. Lớp Meta chứa thông tin bổ sung để quản lý một mô hình; ở đây, nó cho phép chúng ta thiết lập một thuộc tính đặc biệt yêu cầu Django sử dụng Mục nhập khi nó cần tham chiếu đến nhiều mục nhập. Nếu không có điều này, Django sẽ gọi nhiều mục nhập là Entries.

Phương thức __str__() cho Django biết thông tin nào sẽ hiển thị khi nó tham chiếu đến các mục nhập riêng lẻ. Bởi vì một mục nhập có thể là một nội dung dài của văn bản, chúng ta yêu cầu Django chỉ hiển thị 50 ký tự đầu tiên của văn bản. Chúng ta cũng thêm một dấu chấm lửng để làm rõ rằng chúng ta không phải lúc nào cũng hiển thị toàn bộ mục nhập.

Dịch chuyển mô hình đầu vào

Vì chúng ta đã thêm một mô hình mới nên cần phải di chuyển lại cơ sở dữ liệu. Quá trình này sẽ trở nên khá quen thuộc: hãy sửa đổi models.py, chạy lệnh python management.py makemigrations app_name, sau đó chạy lệnh python management.py migrate.

Di chuyển cơ sở dữ liệu và kiểm tra kết quả đầu ra bằng cách nhập các lệnh sau:

```
(11_env)learning_log$ python manage.py makemigrations learning_logs
Migrations for 'learning_logs':
  learning_logs/migrations/0002_entry.py
```

```
- Create model Entry  
(11_env)learning_logs$ python manage.py migrate  
Operations to perform:  
--snip--  
Applying learning_logs.0002_entry... OK
```

Một file mới có tên 0002_entry.py được tạo ra, cho Django biết cách sửa đổi cơ sở dữ liệu để lưu trữ thông tin liên quan đến mô hình Entry. Khi chúng ta đưa ra lệnh di chuyển, tathay rằng Django đã áp dụng quá trình di chuyển này và mọi thứ đều ổn.

Đăng ký Entry với trang quản trị Django

Chúng ta cũng cần đăng ký mô hình Entry. Dưới đây là sửa đổi file admin.py:

```
from django.contrib import admin  
from .models import Topic, Entry  
admin.site.register(Topic)  
admin.site.register(Entry)
```

Quay lại <http://localhost/admin/> và ta sẽ thấy các Mục được liệt kê trong Learning_Logs. Bấm vào liên kết **Add** cho Mục nhập hoặc bấm **Entries**, sau đó chọn **Add entry**. Ta sẽ thấy danh sách thả xuống để chọn chủ đề ta đang tạo mục nhập và hộp văn bản để thêm mục nhập. Chọn Cờ vua từ danh sách thả xuống và thêm một mục nhập. Đây là ví dụ cho mục đầu tiên ta thực hiện:

Mở đầu là phần đầu tiên của trò chơi, khoảng mười nước đi đầu tiên. Trong phần mở đầu, ta nên làm ba điều — đưa ra các giám mục và hiệp sĩ của ta, cố gắng kiểm soát trung tâm của bàn cờ và nhà vua của ta.

Tất nhiên, đây chỉ là những hướng dẫn. Điều quan trọng là phải tìm hiểu khi nào thì tuân theo các hướng dẫn này và khi nào thì bỏ qua các đề xuất này.

Khi nhấp vào Lưu, ta sẽ được đưa trở lại trang quản trị chính cho các mục nhập. Tại đây, chúng ta thấy lợi ích của việc sử dụng văn bản [: 50] làm đại diện chuỗi cho mỗi mục nhập; sẽ dễ dàng hơn nhiều khi làm việc với nhiều mục nhập trong giao diện quản trị nếu ta chỉ thấy phần đầu tiên của mục nhập thay vì toàn bộ văn bản của từng mục nhập.

Tạo một mục thứ hai cho Cờ vua và một mục cho Leo núi để chúng ta có một số dữ liệu ban đầu. Đây là mục thứ hai cho Cờ vua:

Trong giai đoạn mở đầu của trò chơi, điều quan trọng là phải đem ra giám mục và hiệp sĩ của mình. Những quân cờ này đủ mạnh và cơ động để đóng một vai trò quan trọng trong các bước di chuyển đầu của trò chơi.

Và dưới đây là mục đầu tiên của Leo núi:

Một trong những khái niệm quan trọng nhất khi leo núi là giữ trọng lượng trên đôi chân của ta càng nhiều càng tốt. Có một huyền thoại cho rằng những người leo núi có thể treo mình cả ngày trên cánh tay của họ. Trên thực tế, những người leo núi giỏi đã thực hành những cách cụ thể để giữ trọng lượng của họ trên đôi chân của họ bất cứ khi nào có thể.

Django Shell

Với một số dữ liệu đã nhập, chúng ta có thể kiểm tra dữ liệu đó theo chương trình thông qua một phiên đầu cuối tương tác. Môi trường tương tác này được gọi là Django shell và đó là một môi trường tuyệt vời để kiểm tra và khắc phục sự cố cho dự án. Đây là một ví dụ về shell :

```
(11_env)learning_log$ python manage.py shell
>>> from learning_logs.models import Topic
>>> Topic.objects.all()
<QuerySet [<Topic: Chess>, <Topic: Rock Climbing>]>
```

Lệnh python manage.py shell, chạy trong môi trường ảo đang hoạt động, khởi chạy trình thông dịch Python mà ta có thể sử dụng để khám phá dữ liệu được lưu trữ trong cơ sở dữ liệu của dự án của mình. Ở đây, chúng ta nhập mô hình Topic từ module learning_logs.models. Sau đó ta sử dụng phương thức Topic.objects.all() để lấy tất cả các phiên bản của mô hình Chủ đề; danh sách được trả về được gọi là bộ truy vấn.

Chúng ta có thể lặp qua một bộ truy vấn giống như chúng ta lặp qua một danh sách. Dưới đây là cách ta có thể xem ID được chỉ định cho từng đối tượng chủ đề:

```
>>> topics = Topic.objects.all()
>>> for topic in topics:
...     print(topic.id, topic)
...
1 Chess
2 Rock Climbing
```

Chúng ta lưu trữ bộ truy vấn trong các chủ đề, sau đó in từng thuộc tính id chủ đề và biểu diễn chuỗi của mỗi chủ đề. Chúng ta có thể thấy rằng Cờ vua có ID là 1 và Rock Climbing có ID là 2.

Nếu ta biết ID của một đối tượng cụ thể, ta có thể sử dụng phương thức Topic.objects.get() để truy xuất đối tượng đó và kiểm tra bất kỳ thuộc tính nào mà đối tượng có. Hãy xem văn bản và giá trị date_added cho Cờ vua:

```
>>> t = Topic.objects.get(id=1)
>>> t.text
'Chess'
>>> t.date_added
datetime.datetime(2019, 2, 19, 1, 55, 31, 98500, tzinfo=<UTC>)
```

Chúng ta cũng có thể xem các mục liên quan đến một chủ đề nào đó. Trước đó chúng ta đã xác định thuộc tính chủ đề cho mô hình Entry. Đây là một ForeignKey, một kết nối giữa mỗi mục và một chủ đề. Django có thể sử dụng kết nối này để nhận mọi mục nhập liên quan đến một chủ đề nhất định, như sau:

```
>>> t.entry_set.all()
<QuerySet [
```

Để lấy dữ liệu thông qua mối quan hệ khóa ngoại, ta sử dụng tên viết thường của mô hình liên quan, theo sau là dấu gạch dưới và từ “set”. Ví dụ: giả sử ta có mô hình Pizza và Topping, và Topping có liên quan đến Pizza thông qua một khóa ngoại. Nếu đối tượng của ta được gọi là my_pizza, đại diện cho một chiếc bánh pizza, ta có thể lấy tất cả các lớp phủ của bánh pizza bằng cách sử dụng mã my_pizza.topping_set.all().

Chúng ta sẽ sử dụng loại cú pháp này khi bắt đầu viết mã các trang mà người dùng có thể yêu cầu. Shell rất hữu ích để đảm bảo mã truy xuất dữ liệu mà ta muốn. Nếu mã tạo ra lỗi hoặc không truy xuất dữ liệu mà ta mong đợi, thì việc khắc phục sự cố mã trong môi trường shell đơn giản sẽ dễ dàng hơn nhiều so với trong các tệp tạo trang web. Chúng ta sẽ không đề cập đến shell nhiều, nhưng chúng ta nên tiếp tục sử dụng nó để làm việc với cú pháp của Django cho việc truy cập vào dữ liệu được lưu trữ trong dự án.

Chú ý : Mỗi lần sửa đổi mô hình của mình, ta sẽ cần khởi động lại shell để xem những thay đổi đó. Để thoát một phiên trình sell, hãy nhấn ctrl-D; trên Windows, nhấn ctrl-Z rồi nhấn enter.

Hãy thử

Mục nhập ngắn: Phương thức `__str__()` trong mô hình Entry hiện thêm dấu chấm lửng vào mọi trường hợp của Mục nhập khi Django hiển thị nó trong trang web quản trị hoặc trình bao. Thêm câu lệnh `if` vào phương thức `__str__()` chỉ thêm dấu chấm lửng nếu mục nhập dài hơn 50 ký tự. Sử dụng trang web quản trị để thêm một mục nhập có độ dài dưới 50 ký tự và kiểm tra xem nó có dấu chấm lửng hay không.

API Django: Khi ta viết mã để truy cập dữ liệu trong dự án của mình, ta đang viết một truy vấn. Đọc tài liệu truy vấn dữ liệu tại <https://docs.djangoproject.com/en/2.2/topics/db/queries/>. Phần lớn những gì ta thấy sẽ có vẻ mới mẻ đối, nhưng nó sẽ rất hữu ích khi ta bắt đầu thực hiện các dự án của riêng mình.

Pizzeria: Bắt đầu một dự án mới có tên là pizzeria với một ứng dụng có tên là pizza. Xác định mô hình Pizza với trường tên, trường này sẽ chứa các giá trị tên, chẳng hạn như Hawaiian và Meat Lovers. Xác định một mô hình được gọi là Topping với các trường được gọi là pizza và tên. Trường bánh pizza phải là khóa ngoại đối với Pizza và tên phải có thể chứa các giá trị như dứa, thịt xông khói Canada và xúc xích.

10.3.1.3. Tạo trang chủ

Tạo trang web với Django bao gồm ba giai đoạn: xác định URL, chế độ xem và viết khuôn mẫu. Ta có thể thực hiện những điều này theo bất kỳ thứ tự nào, nhưng trong dự án này, chúng ta sẽ luôn bắt đầu bằng cách xác định mẫu URL. Mẫu URL mô tả cách trình bày URL. Nó cũng cho Django biết những gì cần tìm khi đối sánh yêu cầu của trình duyệt với URL của trang web để nó biết trang nào sẽ quay lại.

Sau đó, mỗi URL ánh xạ tới một chế độ xem cụ thể — chức năng chế độ xem truy xuất và xử lý dữ liệu cần thiết cho trang đó. Chức năng xem thường hiển thị trang bằng cách sử dụng mẫu chứa cấu trúc tổng thể của trang. Để xem cách này hoạt động

như thế nào, hãy tạo trang chủ cho Nhật ký học tập. Chúng ta sẽ xác định URL cho trang chủ, viết chức năng xem của nó và tạo một mẫu đơn giản.

Bởi vì tất cả những gì chúng ta đang làm là đảm bảo Nhật ký học tập hoạt động như bình thường,hãy tạo một trang đơn giản. Một ứng dụng web đang hoạt động rất thú vị để tạo kiểu khi nó hoàn tất; một ứng dụng có vẻ tốt nhưng không hoạt động tốt là điều vô nghĩa. Hiện tại, trang chủ sẽ chỉ hiển thị tiêu đề và mô tả ngắn gọn.

Ánh xạ URL

Người dùng yêu cầu trang bằng cách nhập URL vào trình duyệt và nhấp vào liên kết, vì vậy, ta cần quyết định URL nào là cần thiết. Trước tiên là URL của trang chủ: đó là URL cơ sở mà mọi người sử dụng để truy cập vào dự án. Hiện tại, URL cơ sở, `http://localhost:8000/`, trả về trang Django mặc định cho chúng ta biết dự án đã được thiết lập chính xác. Chúng ta sẽ thay đổi điều này bằng cách ánh xạ URL cơ sở tới trang chủ của Nhật ký học tập.

Trong thư mục dự án `learning_log` chính, hãy mở tệp `urls.py`. Đây là mã ta sẽ thấy:

```
from django.contrib import admin
from django.urls import path
urlpatterns = [
    path('admin/', admin.site.urls),
]
```

Hai dòng đầu tiên nhập một mô-đun và một chức năng để quản lý URL cho trang web quản trị . Nội dung của tệp xác định biến `urlpatterns` . Trong tệp `urls.py` này, đại diện cho toàn bộ dự án, giá trị của `urlpatterns` bao gồm các tập hợp URL từ các ứng dụng trong dự án. Mã bao gồm mô-đun `admin.site.urls`, xác định tất cả các URL có thể yêu cầu từ trang web quản trị.

Chúng ta cần URL cho Nhật kí học tập, hãy thêm mã sau:

```
from django.contrib import admin
from django.urls import path, include
urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('learning_logs.urls')),
]
```

`urls.py` mặc định nằm trong thư mục `learning_log`; bây giờ chúng ta cần tạo tệp `urls.py` thứ hai trong thư mục `learning_logs`. Tạo một tệp Python mới và lưu nó dưới dạng `urls.py` trong `learning_logs` và nhập mã này vào đó.

```
"""Defines URL patterns for learning_logs."""
from django.urls import path
from . import views
app_name = 'learning_logs'
urlpatterns = [
    # Home page
    path('', views.index, name='index'),
]
```

Để làm rõ urls.py đang hoạt động trong url nào, chúng ta thêm chú thích ở đầu tệp . Sau đó, hãy nhập hàm đường dẫn path, hàm này cần thiết khi ánh xạ URL tới các chế độ xem . Chúng ta cũng nhập mô-đun chế độ xem views; dấu chấm yêu cầu Python nhập mô-đun views.py từ cùng một thư mục với mô-đun urls.py hiện tại. Biến app_name giúp Django phân biệt tệp urls.py này với các tệp cùng tên trong các ứng dụng khác trong dự án . Biến urlpatterns là danh sách các trang riêng lẻ có thể được yêu cầu từ ứng dụng learning_logs.

Mẫu URL thực tế là một lệnh gọi đến hàm path(), hàm này sẽ ba đối số . Đối số đầu tiên là một chuỗi giúp Django định tuyến yêu cầu hiện tại một cách chính xác. Django nhận được URL được yêu cầu và cố gắng định tuyến yêu cầu đến một chế độ xem. Nó thực hiện điều này bằng cách tìm kiếm tất cả các mẫu URL mà chúng ta đã xác định để tìm một mẫu phù hợp với yêu cầu hiện tại. Django bỏ qua URL cơ sở cho dự án (<http://localhost:8000/>), vì vậy chuỗi trống ("") khớp với URL cơ sở. Django sẽ trả về trang lỗi nếu URL được yêu cầu không khớp với bất kỳ các mẫu URL hiện có.

Đối số thứ hai trong path() chỉ định hàm nào sẽ gọi trong views.py. Khi một URL được yêu cầu khớp với mẫu mà chúng ta đang xác định, Django sẽ gọi hàm index() từ views.py (chúng ta sẽ viết hàm xem này trong phần tiếp theo). Đối số thứ ba cung cấp chỉ mục tên cho mẫu URL này để chúng ta có thể tham khảo nó trong các phần mã khác. Bất cứ khi nào ta muốn cung cấp một liên kết đến trang chủ, hãy sử dụng tên này thay vì viết ra một URL.

Viết mã Giao diện

Một chức năng views lấy thông tin từ một yêu cầu, chuẩn bị dữ liệu cần thiết để tạo một trang, sau đó gửi dữ liệu trở lại trình duyệt, thường bằng cách sử dụng mẫu xác định trang sẽ trông như thế nào.

Tệp views.py trong learning_logs được tạo tự động khi chúng ta chạy lệnh python management.py startapp. Đây là những gì có trong views.py hiện tại:

```
from django.shortcuts import render
# Create your views here.
```

Hiện tại, tệp này chỉ nhập hàm render(), hàm này hiển thị phản hồi dựa trên dữ liệu được cung cấp bởi các chế độ xem. Mở tệp views và thêm mã sau cho trang chủ:

```
from django.shortcuts import render
def index(request):
    """The home page for Learning Log."""
    return render(request, 'learning_logs/index.html')
```

Khi một yêu cầu URL khớp với mẫu mà chúng ta vừa xác định, Django sẽ tìm kiếm một hàm có tên là index() trong tệp views.py. Django sau đó chuyển đổi đối tượng yêu cầu đến chức năng views này. Trong trường hợp này, chúng ta không cần xử lý bất kỳ dữ liệu nào cho trang, vì vậy mã duy nhất trong hàm là gọi hàm render(). Ở đây, hàm render() có hai đối số — đối tượng yêu cầu ban đầu request và một mẫu mà nó có thể sử dụng để xây dựng trang. Hãy viết mẫu này.

Viết mã Template

Mẫu xác định trang sẽ trông như thế nào và Django điền vào dữ liệu có liên quan mỗi khi trang được yêu cầu. Mẫu cho phép ta truy cập vào bất kỳ dữ liệu nào được cung cấp bởi dạng views. Bởi vì chế độ xem cho trang chủ không cung cấp dữ liệu, mẫu này khá đơn giản.

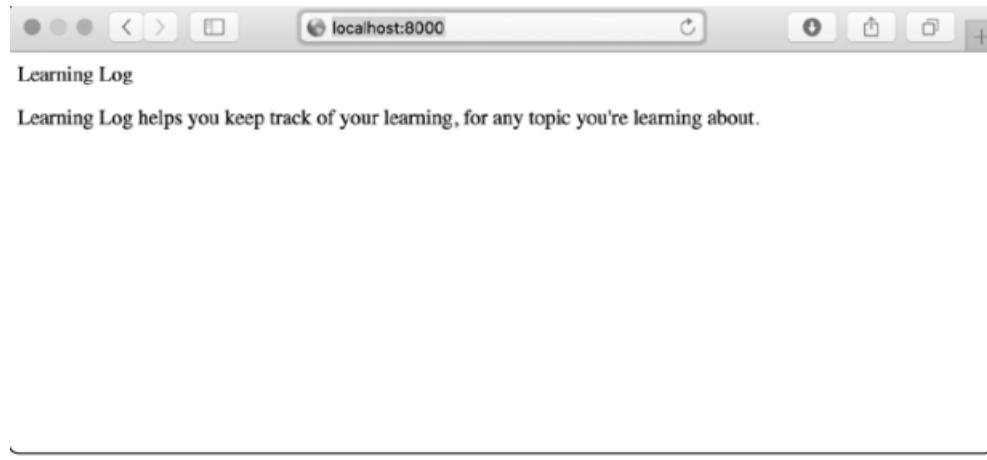
Bên trong thư mục learning_logs, hãy tạo một thư mục mới có tên là templates. Bên trong thư mục templates, tạo một thư mục khác có tên learning_logs. Điều này có vẻ hơi thừa (vì chúng ta đã có một thư mục có tên learning_logs), nhưng nó thiết lập một cấu trúc mà Django có thể diễn giải một cách rõ ràng, ngay cả trong bối cảnh của một dự án lớn chứa nhiều ứng dụng riêng lẻ. Bên trong thư mục learning_logs, hãy tạo một tệp mới có tên là index.html. Đường dẫn đến tệp sẽ là learning_log / learning_logs / Template / learning_logs / index.html. Nhập mã sau vào tệp đó:

```
<p>Learning Log</p>
<p>Learning Log helps you keep track of your learning, for any topic
you're learning about.</p>
```

Đây là một tập tin rất đơn giản, với HTML, thì thẻ `<p> </p>` biểu thị đoạn văn. Thẻ `<p>` mở và thẻ `</p>` đóng đoạn văn. Chúng ta có hai đoạn: đoạn đầu tiên

đóng vai trò như một tiêu đề và đoạn thứ hai mô tả những gì người dùng có thể làm với Nhật ký học tập.

Bây giờ, khi ta yêu cầu URL cơ sở của dự án, `http://localhost:8000/`, ta sẽ thấy trang vừa tạo thay vì trang Django mặc định. Django đã lấy URL được yêu cầu và URL đó sẽ khớp với mẫu ''; từ đó Django gọi hàm `views.index()`, hàm này sẽ hiển thị trang bằng cách sử dụng mẫu có trong `index.html`. Hình 10-3 cho thấy trang kết quả.



Hình 10.38. Trang chủ của trang Nhật kí học tập

Mặc dù nó có vẻ là một quá trình phức tạp để tạo một trang, nhưng sự tách biệt này giữa các URL, chế độ xem và mẫu hoạt động khá tốt. Nó cho phép ta suy nghĩ về từng khía cạnh của một dự án một cách riêng biệt. Trong các dự án lớn hơn, nó cho phép các cá nhân làm việc trong dự án tập trung vào các lĩnh vực mà họ mạnh nhất. Ví dụ: một chuyên gia cơ sở dữ liệu có thể tập trung vào các mô hình, một lập trình viên có thể tập trung vào chế độ `views` và một nhà thiết kế web có thể tập trung vào các mẫu `templates`.

Chú ý: Ta phải chú ý đến thông báo lỗi sau :`ModuleNotFoundError: No module named 'learning_logs.urls'`. Nếu gặp phải, hãy dừng hoạt động dự án bằng cách nhấn `ctrl-C` trong cửa sổ đầu cuối. Sau đó phát hành lại lệnh `python management.py runserver`.

10.3.1.4. Xây dựng các trang phụ

Bây giờ hãy thiết lập quy trình xây dựng trang, chúng ta có thể bắt đầu xây dựng dự án Nhật ký học tập. Ta sẽ xây dựng hai trang hiển thị dữ liệu: một trang liệt kê tất cả các chủ đề và một trang hiển thị tất cả các mục nhập cho một chủ đề cụ thể.

Đối với mỗi trang, chúng ta sẽ chỉ định một mẫu URL, viết một hàm views và viết một templates. Nhưng trước khi thực hiện việc này, hãy tạo một mẫu cơ sở mà tất cả các mẫu trong dự án có thể kế thừa.

Kế thừa các template

Khi xây dựng một trang web, một số yếu tố sẽ luôn cần được lặp lại trên mỗi trang. Thay vì phải viết các phần tử này trực tiếp vào mỗi trang, ta có thể viết một mẫu cơ sở chứa các phần tử lặp lại và sau đó để mỗi trang kế thừa phần cơ sở này. Cách tiếp cận này cho phép ta tập trung vào việc phát triển các khía cạnh độc đáo của mỗi trang và giúp thay đổi giao diện tổng thể của dự án dễ dàng hơn nhiều.

a. Mẫu gốc

Chúng ta sẽ tạo một mẫu có tên là base.html trong cùng một thư mục với index.html. Tệp này sẽ chứa các phần tử chung cho tất cả các trang; mọi mẫu khác sẽ kế thừa từ base.html. Yếu tố duy nhất chúng ta muốn lặp lại trên mỗi trang ngay bây giờ là tiêu đề ở trên cùng.

```
base.html    <p>
              <a href="{% url 'learning_logs:index' %}">LearningLog</a>
            </p>
            {% block content %}{% endblock content %}
```

Phần đầu tiên của tệp này tạo một đoạn văn có chứa tên của dự án, cũng hoạt động như một liên kết trang chủ. Để tạo liên kết, chúng ta sử dụng thẻ mẫu, được biểu thị bằng dấu ngoặc nhọn và dấu phẩy trăm {%%}. Thẻ mẫu tạo ra thông tin được hiển thị trên một trang. Thẻ mẫu {% url 'learning_logs: index' %} tạo một URL khớp với mẫu URL được xác định trong learning_logs / urls.py với tên 'index'. Trong ví dụ này, learning_logs là không gian tên, chỉ mục là một mẫu URL được đặt tên duy nhất trong không gian tên đó. Không gian tên đến từ giá trị mà chúng ta đã gán cho app_name trong tệp learning_logs / urls.py.

Trong một trang HTML đơn giản, một liên kết được bao quanh bởi thẻ liên kết <a>:

```
<a href="link_url">link text</a>
```

Việc có thẻ mẫu tạo URL giúp cho việc cập nhật các liên kết của chúng ta dễ dàng hơn nhiều. Ta chỉ cần thay đổi mẫu URL trong urls.py và Django sẽ tự động chèn URL được cập nhật vào lần tiếp theo trang được yêu cầu. Mọi trang trong dự án

của chúng ta sẽ kế thừa từ base.html, vì vậy kể từ bây giờ, mọi trang sẽ có một liên kết trả lại trang chủ.

b.Mẫu con:

Bây giờ chúng ta cần viết lại index.html để kế thừa từ base.html. Thêm mã sau vào index.html:

```
{% extends "learning_logs/base.html" %}  
{% block content %}  
<p>Learning Log helps you keep track of your learning, for any topic  
you're  
learning about.</p>  
{% endblock content %}
```

So sánh cái này với index.html ban đầu, có thể thấy rằng ta đã thay thế tiêu đề Nhật ký học tập bằng mã kế thừa từ mẫu mẹ . Mẫu con phải có thẻ {% extends%} trên dòng đầu tiên để cho Django biết mẫu mẹ nào sẽ kế thừa. Tệp base.html là một phần của learning_logs, vì vậy chúng ta đưa learning_logs vào đường dẫn đến mẫu mẹ.

Chúng ta xác định khôi nội dung bằng cách chèn thẻ {% block%} với nội dung tên. Mọi thứ mà ta không kế thừa từ mẫu mẹ sẽ đi vào bên trong khôi nội dung. Tại dòng mã cuối cho biết ta đã hoàn tất việc xác định nội dung bằng cách sử dụng thẻ {% endblock content%}. Thẻ {% endblock%} không yêu cầu tên, nhưng nếu một mẫu phát triển để chứa nhiều khôi, thì việc biết chính xác khôi nào đang kết thúc có thể hữu ích.

Ta có thể bắt đầu thấy lợi ích của việc kế thừa mẫu: trong mẫu con, chúng ta chỉ cần đưa nội dung duy nhất vào trang đó. Điều này không chỉ đơn giản hóa từng mẫu mà còn giúp việc sửa đổi trang web dễ dàng hơn nhiều. Để sửa đổi một phần tử chung cho nhiều trang, chỉ cần sửa đổi mẫu mẹ. Các thay đổi sau đó được chuyển sang mọi trang kế thừa từ mẫu đó. Trong một dự án bao gồm hàng chục hoặc hàng trăm trang, cấu trúc này có thể giúp cải thiện trang web dễ dàng và nhanh chóng hơn nhiều.

Chú ý : Trong một dự án lớn, thông thường sẽ có một mẫu mẹ được gọi là base.html cho toàn bộ trang web và các mẫu mẹ cho từng phần chính của trang web. Tất cả các mẫu kế thừa từ base.html và mỗi trang trong trang web cũng kế thừa từ các mẫu đó. Bằng cách này, ta có thể dễ dàng sửa đổi giao diện của toàn bộ trang web

hoặc bất kỳ trang cá nhân nào. Cấu hình này cung cấp một cách rất hiệu quả để làm việc và nó khuyến khích chúng ta cập nhật đều đặn trang web của mình theo thời gian.

Trang các chủ đề

Bây giờ chúng ta đã có một cách tiếp cận hiệu quả để xây dựng các trang, hãy tập trung vào hai trang tiếp theo: trang chủ đề chung và trang để hiển thị các mục nhập cho một chủ đề duy nhất. Trang chủ đề sẽ hiển thị tất cả các chủ đề mà người dùng đã tạo và đây là trang đầu tiên sẽ liên quan đến việc làm việc với dữ liệu.

Mẫu URL cho trang chủ đề

Đầu tiên, chúng ta xác định URL cho trang chủ đề. Thông thường người ta thường chọn một đoạn URL đơn giản phản ánh loại thông tin được trình bày trên trang. Chúng ta sẽ sử dụng từ Topics, vì vậy URL http://localhost:8000/themes/ sẽ trả về trang này. Đây là cách chúng ta sửa đổi learning_logs/urls.py:

```
"""Defines URL patterns for learning_logs."""
--snip--
urlpatterns = [
    # Home page.
    path('', views.index, name='index'),
    # Page that shows all topics.
    path('topics/', views.topics, name='topics'),
]
```

Chúng ta chỉ cần thêm topics/ vào đối số chuỗi được sử dụng cho URL trang chủ. Khi Django kiểm tra URL được yêu cầu, mẫu này sẽ khớp với bất kỳ URL nào có URL cơ sở theo sau là các chủ đề. Ta có thể thêm hoặc bỏ dấu gạch chéo ở cuối. Sau đó, bất kỳ yêu cầu nào có URL phù hợp với mẫu này sẽ được chuyển đến các chủ đề hàm() trong views.py.

Chế độ xem Chủ đề

Hàm topics() cần lấy một số dữ liệu từ cơ sở dữ liệu và gửi đến mẫu. Đây là những gì chúng ta cần thêm vào views.py:

```
from django.shortcuts import render
from .models import Topic
def index(request):
    --snip--
def topics(request):
    """Show all topics."""

```

```

topics = Topic.objects.order_by('date_added')
context = {'topics': topics}
return render(request, 'learning_logs/topics.html', context)

```

Đầu tiên, chúng ta import mô hình được liên kết với dữ liệu chúng ta cần . Hàm topics() cần một tham số: đối tượng yêu cầu mà Django nhận được từ máy chủ . Sau đó, ta truy vấn cơ sở dữ liệu bằng cách yêu cầu các đối tượng Chủ đề, được sắp xếp theo thuộc tính date_added. Chúng ta lưu trữ bộ truy vấn kết quả vào trong các chủ đề.

Xác định ngữ cảnh mà chúng ta sẽ gửi đến mẫu. Context-Ngữ cảnh là một từ điển, trong đó các khóa là tên ta sẽ sử dụng trong mẫu để truy cập dữ liệu và các giá trị là dữ liệu chúng ta cần gửi đến mẫu. Trong trường hợp này, có một cặp khóa-giá trị, chưa tập hợp các chủ đề sẽ hiển thị trên trang. Khi xây dựng một trang sử dụng dữ liệu, ta chuyển biến ngữ cảnh để render() cũng như đối tượng yêu cầu và đường dẫn đến mẫu .

Mẫu chủ đề

Mẫu cho trang chủ đề nhận từ điển ngữ cảnh, vì vậy mẫu có thể sử dụng dữ liệu mà chủ đề cung cấp. Tạo một tệp có tên là topic.html trong cùng thư mục với index.html. Đây là cách chúng ta có thể hiển thị các chủ đề trong mẫu:

```

{% extends "learning_logs/base.html" %}
{% block content %}
<p>Topics</p>
<ul>
    {% for topic in topics %}
        <li>{{ topic }}</li>
    {% empty %}
        <li>No topics have been added yet.</li>
    {% endfor %}
</ul>
{% endblock content %}

```

Chúng ta sử dụng thẻ `{% extends %}` để kế thừa base.html, sau đó mở nội dung. Phần nội dung của trang này chứa một danh sách có dấu đầu dòng về các chủ đề đã được nhập. Trong HTML chuẩn, một danh sách có dấu đầu dòng được gọi là danh sách không có thứ tự và được biểu thị bằng các thẻ `` ``.

Sau đó, chúng ta có một thẻ mẫu khác tương đương với vòng lặp for, thẻ này lặp qua các chủ đề danh sách trong từ điển ngữ cảnh. Mã được sử dụng trong các mẫu

khác với Python theo một số cách quan trọng. Python sử dụng thụt đầu dòng để cho biết dòng nào là của câu lệnh for. Trong một mẫu, mọi vòng lặp for cần một thẻ `{% endfor%}` rõ ràng cho biết đã kết thúc vòng for. Vì vậy, trong một mẫu, ta sẽ thấy các vòng được viết như thế này:

```
{% for item in list %}  
do something with each item  
{% endfor %}
```

Bên trong vòng lặp, chúng ta muốn biến mỗi chủ đề thành một mục trong danh sách và đánh dấu chúng theo đầu dòng. Để in một biến trong một mẫu, hãy đặt tên biến trong dấu ngoặc kép. Dấu ngoặc nhọn sẽ không xuất hiện trên trang; chúng chỉ cho Django biết rằng chúng ta đang sử dụng một biến mẫu. Vì vậy, mã `{topic}` sẽ được thay thế bằng giá trị của topic khi mỗi lần đi qua vòng lặp. Thẻ HTML `` cho biết một mục danh sách. Bất kỳ thứ gì bên trong một cặp thẻ ``, sẽ xuất hiện dưới dạng một mục được đánh dấu đầu dòng.

Sau đó, chúng ta sử dụng thẻ mẫu `{% empty%}`, thẻ này cho Django biết phải làm gì nếu không có mục nào trong danh sách. Trong trường hợp này, ta in một thông báo cho người dùng biết rằng chưa có chủ đề nào được thêm vào. Hai dòng cuối cùng đóng vòng lặp for và sau đó đóng danh sách dấu đầu dòng.

Bây giờ chúng ta cần sửa đổi mẫu cơ sở để bao gồm một liên kết đến trang chủ đề. Thêm mã sau vào base.html:

```
<p>  
<a href="{% url 'learning_logs:index' %}">Learning Log</a> -  
<a href="{% url 'learning_logs:topics' %}">Topics</a>  
</p>  
{% block content %}{% endblock content %}
```

Chúng ta thêm một dấu gạch ngang sau liên kết đến trang chủ Learning Log, và thêm một liên kết đến trang chủ đề Topics bằng cách sử dụng lại thẻ mẫu `{% url%}`. Dòng này yêu cầu Django tạo liên kết khớp với mẫu URL với tên 'topics' trong learning_logs / urls.py.



Hình 10.39. Trang chủ đề

Các trang chủ đề cá nhân

Tiếp theo, chúng ta cần tạo một trang có thể tập trung vào một chủ đề duy nhất, hiển thị tên chủ đề và tất cả các mục cho chủ đề đó. Vì vậy ta cần phải xác định lại một mẫu URL mới, viết một chế độ xem và tạo một mẫu. Đồng thời cũng sửa đổi trang chủ đề để mỗi mục trong danh sách có dấu đầu dòng sẽ tự liên kết đến trang chủ đề tương ứng của nó.

Mẫu URL chủ đề

Mẫu URL cho trang chủ đề hơi khác so với các mẫu URL trước đó vì nó sẽ sử dụng thuộc tính id của chủ đề để cho biết chủ đề nào đã được yêu cầu. Ví dụ: nếu người dùng muốn xem trang chi tiết của chủ đề Cờ vua, trong đó id là 1, URL sẽ là `http://localhost:8000/themes/1/`.

Đây là một mẫu để khớp với URL này, ta nên đặt nó trong `learning_logs/urls.py`:

```
--snip--  
urlpatterns = [  
    --snip--  
    # Detail page for a single topic.  
    path('topics/<int:topic_id>', views.topic, name='topic'),  
]
```

Hãy kiểm tra chuỗi 'topics / <int: topic_id> /' trong mẫu URL này. Phần đầu tiên của chuỗi yêu cầu Django tìm kiếm các URL có chủ đề từ sau URL cơ sở. Phần thứ hai của chuỗi, / <int: topic_id> /, khớp với một số nguyên giữa hai dấu gạch chéo về phía trước và lưu trữ giá trị số nguyên trong một đối số được gọi là `topic_id`.

Khi Django tìm thấy một URL phù hợp với mẫu này, nó sẽ gọi hàm view `topic()` với giá trị được lưu trữ trong `topic_id` làm đối số. Chúng ta sẽ sử dụng giá trị của `topic_id` để có được chủ đề chính xác bên trong hàm.

Chế độ xem chủ đề

Hàm `topic()` cần lấy chủ đề và tất cả các mục nhập liên quan từ cơ sở dữ liệu, như được hiển thị ở đây:

```
--snip--  
def topic(request, topic_id):  
    """Show a single topic and all its entries."""  
    topic = Topic.objects.get(id=topic_id)  
    entries = topic.entry_set.order_by('-date_added')  
    context = {'topic': topic, 'entries': entries}  
    return render(request, 'learning_logs/topic.html', context)
```

Đây là hàm xem đầu tiên yêu cầu một tham số khác với đối tượng yêu cầu. Hàm chấp nhận giá trị được ghi lại bởi biểu thức /<int: topic_id>/ và lưu trữ nó trong `topic_id`. Tiếp đó, chúng ta sử dụng `get()` để truy xuất chủ đề, giống như chúng ta đã làm trong Django shell. Từ đó ta nhận được các mục nhập liên quan đến chủ đề này và sắp xếp chúng theo `date_added`. Đầu trừ phi trước `date_added` sắp xếp kết quả theo thứ tự ngược lại, sẽ hiển thị các mục nhập gần đây nhất trước tiên. Chúng ta lưu trữ chủ đề và các mục nhập trong từ điển ngữ cảnh – `context` đồng thời gửi ngữ cảnh đến `topic.html` mẫu.

Mẫu chủ đề

Mẫu cần hiển thị tên chủ đề và các mục. Chúng ta cũng cần thông báo cho người dùng nếu chưa có mục nhập nào được thực hiện cho chủ đề này. Trong `topic.html` ta có:

```
{% extends 'learning_logs/base.html' %}  
{% block content %}  
    <p>Topic: {{ topic }}</p>  
    <p>Entries:</p>  
    <ul>  
        {% for entry in entries %}  
            <li>  
                <p>{{ entry.date_added|date:'M d, Y H:i' }}</p>  
                <p>{{ entry.text|linebreaks }}</p>  
            </li>  
        {% empty %}  
            <li>There are no entries for this topic yet.</li>
```

```

{ % endfor %}
</ul>
{ % endblock content %}

```

Chúng ta mở rộng base.html, như chúng ta làm cho tất cả các trang trong dự án. Tiếp theo hãy hiển thị chủ đề được lưu trữ trong biến mẫu {{topic}}. Chủ đề có thể thay đổi được vì nó được đưa vào từ điển ngữ cảnh context. Sau đó, chúng ta bắt đầu một danh sách có dấu đầu dòng để hiển thị từng mục nhập và lặp lại chúng như chúng ta đã làm các chủ đề trước đó.

Mỗi dấu đầu dòng liệt kê hai phần thông tin: mốc thời gian và nội dung của mỗi mục. Đối với dấu mốc thời gian, chúng ta hiển thị giá trị của thuộc tính date_added. Trong các mẫu Django, một đường thẳng đứng () đại diện cho một bộ lọc mẫu — một hàm sửa đổi giá trị trong một biến mẫu. Ngày lọc: 'M d, Y H: i' hiển thị dấu thời gian ở định dạng 23:00 ngày 1 tháng 1 năm 2018. Dòng tiếp theo hiển thị giá trị đầy đủ của văn bản thay vì chỉ 50 ký tự đầu tiên từ mục nhập. Dấu ngắt dòng của bộ lọc linebreaks đảm bảo rằng các mục nhập văn bản dài hiển thị một khối văn bản không bị gián đoạn. Cuối cùng, chúng ta sử dụng thẻ mẫu {% empty%} để in thông báo cho người dùng biết rằng không có mục nhập nào được thực hiện.

Liên kết từ Trang chủ đề

Trước khi xem trang chủ đề trong trình duyệt, chúng ta cần sửa đổi mẫu chủ đề để mỗi chủ đề liên kết đến trang thích hợp. Đây là thay đổi cần thực hiện đối với topic.html:

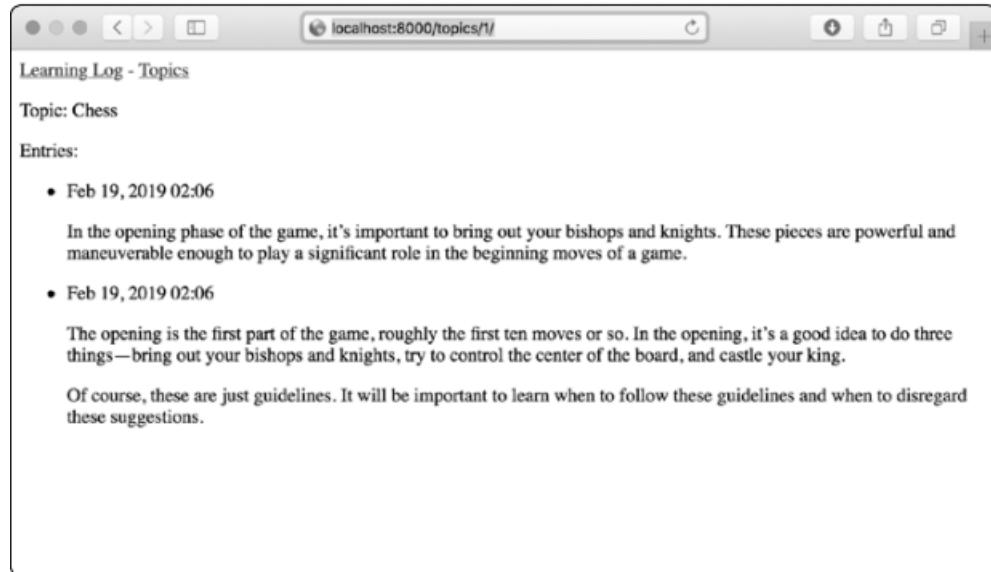
```

--snip--
{ % for topic in topics %}
<li>
<a href="{ % url 'learning_logs:topic' topic.id %}">{{ topic }}</a>
</li>
{ % empty %
--snip-

```

Chúng ta sử dụng thẻ mẫu URL để tạo liên kết thích hợp, dựa trên mẫu URL trong learning_logs với tên 'topic'. Mẫu URL này yêu cầu đối số topic_id, vì vậy chúng cần thêm thuộc tính topic.id vào thẻ mẫu URL. Bây giờ mỗi chủ đề trong danh sách các chủ đề là một liên kết đến một trang chủ đề, chẳng hạn như http://localhost:8000/themes/1/.

Khi đã làm mới trang chủ đề và nhấp vào một chủ đề, ta sẽ thấy trang giống như Hình 10.40



Hình 10.40. Trang chi tiết cho một chủ đề

10.3.1.4. Kết mục

Trong chương này, ta học được cách xây dựng một trang web đơn giản bằng cách sử dụng Django. Ta đã viết một đặc tả dự án ngắn gọn, đã cài đặt Django sang môi trường ảo, thiết lập một dự án và kiểm tra xem dự án đã được thiết lập đúng chưa. Ta thiết lập một ứng dụng và xác định các mô hình để đại diện cho dữ liệu cho ứng dụng của chính mình. Bên cạnh đó, Ta cũng đã tìm hiểu về cơ sở dữ liệu và cách Django giúp di chuyển cơ sở dữ liệu sau khi thực hiện thay đổi đối với mô hình của mình. Đồng thời tạo một superuser cho trang quản trị và sử dụng trang quản trị để nhập một số dữ liệu ban đầu.

Ta cũng đã khám phá Django shell, chúng cho phép làm việc với dữ liệu của dự án trong phiên đầu cuối, học cách xác định URL, tạo chức năng xem và viết mẫu để tạo các trang cho trang web của mình. Đã sử dụng kế thừa các mẫu để đơn giản hóa cấu trúc của các mẫu riêng lẻ và giúp việc sửa đổi trang web dễ dàng hơn khi dự án phát triển.

Trong Chương tiếp theo, ta sẽ học cách tạo các trang trực quan, thân thiện với người dùng cho phép người dùng thêm các chủ đề và mục nhập mới và chỉnh sửa các mục hiện có mà không cần thông qua trang web quản trị. Ta cũng sẽ thêm hệ thống đăng ký người dùng, cho phép người dùng tạo tài khoản và tạo nhật ký học tập của

riêng họ. Đây sẽ là trung tâm của ứng dụng web — khả năng tạo thứ gì đó mà bất kỳ số lượng người dùng nào cũng có thể tương tác.

10.3.2. Tài khoản người dùng

Trọng tâm của ứng dụng web là đáp ứng nhu cầu cho bất kỳ người dùng nào, ở bất kỳ đâu trên thế giới, đăng ký tài khoản với ứng dụng của mình và bắt đầu sử dụng nó. Trong chương này, ta sẽ xây dựng các biểu mẫu để người dùng có thể thêm các chủ đề và mục nhập của riêng họ, đồng thời chỉnh sửa các mục nhập hiện có. Ta cũng sẽ tìm hiểu cách Django bảo vệ chống lại các cuộc tấn công phổ biến vào các trang dựa trên biểu mẫu, do đó chúng ta không phải mất nhiều thời gian suy nghĩ về việc bảo mật ứng dụng của mình.

Ta cũng sẽ triển khai hệ thống xác thực người dùng, xây dựng một trang đăng ký để người dùng tạo tài khoản, sau đó chỉ giới hạn quyền truy cập vào các trang nhất định đối với người dùng đã đăng nhập. Sau đó, ta sẽ sửa đổi một số chức năng của chế độ xem để người dùng chỉ có thể xem dữ liệu của riêng họ. Đồng thời học cách giữ cho dữ liệu của người dùng của mình an toàn và bảo mật.

10.3.2.1. Người dùng nhập liệu

Trước khi xây dựng hệ thống xác thực để tạo tài khoản, trước tiên chúng ta sẽ thêm một số trang cho phép người dùng nhập dữ liệu của riêng họ. Ta sẽ cung cấp cho người dùng khả năng thêm chủ đề mới, thêm mục nhập mới và chỉnh sửa các mục nhập trước đó của họ. Hiện tại, chỉ siêu người dùng mới có thể nhập dữ liệu thông qua trang web quản trị. Chúng ta không muốn người dùng tương tác với trang web quản trị, vì vậy, chúng ta sẽ sử dụng các công cụ xây dựng biểu mẫu của Django để tạo các trang cho phép người dùng nhập dữ liệu.

a. Thêm chủ đề mới

Hãy bắt đầu bằng cách cho phép người dùng thêm một chủ đề mới. Việc thêm trang dựa trên biểu mẫu hoạt động giống như các trang chúng ta đã tạo: xác định URL, viết hàm view và viết mẫu. Một điểm khác biệt chính là việc bổ sung một mô-đun mới có tên là form.py, mô-đun này sẽ chứa các biểu mẫu.

Mô hình Chủ đề

Bất kỳ trang nào cho phép người dùng nhập và gửi thông tin trên trang web là một biểu mẫu, ngay cả khi nó không giống như một biểu mẫu. Khi người dùng nhập

thông tin, chúng ta cần xác thực rằng thông tin được cung cấp là loại dữ liệu phù hợp và không độc hại, chẳng hạn như mã làm gián đoạn máy chủ. Sau đó, chúng ta cần xử lý và lưu thông tin hợp lệ vào vị trí thích hợp trong cơ sở dữ liệu. Django tự động hóa phần lớn công việc này.

Cách đơn giản nhất để tạo biểu mẫu trong Django là sử dụng ModelForm, sử dụng thông tin từ các mô hình mà chúng ta đã xác định trong Chương trước để tự động tạo biểu mẫu. Viết biểu mẫu đầu tiên vào tệp form.py, tệp mà ta nên tạo trong cùng một thư mục với models.py:

```
from django import forms
from .models import Topic

class TopicForm(forms.ModelForm):
    class Meta:
        model = Topic
        fields = ['text']
        labels = {'text': ''}
```

Trước tiên, chúng ta import mô-đun forms và models Topic mà chúng ta sẽ làm việc, được gọi là Chủ đề. Chúng ta định nghĩa một lớp được gọi là TopicForm, lớp này kế thừa từ các form.ModelForm.

Phiên bản đơn giản nhất của ModelForm bao gồm một lớp Meta lồng nhau cho Django biết mô hình nào sẽ dựa trên biểu mẫu và những trường nào cần đưa vào biểu mẫu. Ta xây dựng một biểu mẫu từ mô hình Topic và chỉ bao gồm trường văn bản.

URL new_topic

URL cho một trang mới phải ngắn gọn và mang tính mô tả. Khi người dùng muốn thêm một chủ đề mới, chúng ta sẽ cho họ đến `http://localhost:8000/new_topic/`. Đây là mẫu URL cho trang new_topic mà ta thêm vào `learning_logs/urls.py`:

```
--snip--
urlpatterns = [
    --snip--
    # Page for adding a new topic
    path('new_topic/', views.new_topic, name='new_topic'),
]
```

Hàm xem new_topic()

Hàm new_topic() cần xử lý hai tình huống khác nhau: các yêu cầu ban đầu cho trang new_topic(trong trường hợp đó, nó sẽ hiển thị một biểu mẫu trống) và xử lý bất kỳ dữ liệu nào được gửi trong biểu mẫu. Sau khi dữ liệu từ biểu mẫu đã gửi được xử lý, nó cần chuyển hướng người dùng trở lại trang chủ đề:

```
from django.shortcuts import render, redirect
from .models import Topic
from .forms import TopicForm
--snip--
def new_topic(request):
    """Add a new topic."""
    if request.method != 'POST':
        # No data submitted; create a blank form.
        form = TopicForm()
    else:
        # POST data submitted; process data.
        form = TopicForm(data=request.POST)
        if form.is_valid():
            form.save()
            return redirect('learning_logs:topics')
    # Display a blank or invalid form.
    context = {'form': form}
    return render(request, 'learning_logs/new_topic.html', context)
```

Chúng ta import hàm redirect, cái mà chúng ta sẽ sử dụng để chuyển hướng người dùng quay lại trang chủ đề sau khi họ gửi chủ đề của mình. Hàm redirect() lấy tên của một dạng xem và chuyển hướng người dùng đến dạng xem đó.

GET and POST Requests

Hai loại yêu cầu chính mà ta sẽ sử dụng khi xây dựng ứng dụng web là GET và POST. Sử dụng yêu cầu GET cho các trang chỉ đọc dữ liệu từ máy chủ. Ta thường sử dụng yêu cầu POST khi người dùng cần gửi thông tin thông qua một biểu mẫu. Chúng ta sẽ chỉ định phương thức POST để xử lý tất cả các biểu mẫu của mình.(Một số loại yêu cầu khác tồn tại, nhưng chúng ta sẽ không sử dụng chúng trong dự án này.)

Hàm new_topic() nhận đối tượng request làm tham số.Khi người dùng yêu cầu trang này, trình duyệt của họ sẽ gửi một yêu cầu GET. Khi người dùng đã điền và gửi biểu mẫu, trình duyệt của họ sẽ gửi yêu cầu POST. Tùy thuộc vào yêu cầu, chúng ta sẽ biết liệu người dùng đang yêu cầu biểu mẫu trống (yêu cầu GET) hay yêu cầu chúng ta xử lý biểu mẫu đã hoàn thành (yêu cầu POST).

Kiểm tra “`if request.method != 'POST':`” xác định xem phương thức yêu cầu là GET hay POST. Nếu phương thức yêu cầu không phải là POST, yêu cầu có thể là GET, vì vậy chúng ta cần trả lại một biểu mẫu trống (nếu đó là một loại yêu cầu khác, vẫn an toàn để trả lại biểu mẫu trống). Chúng ta tạo một phiên bản của TopicForm , gán nó cho biến form và gửi nó đến mẫu trong từ điển ngữ cảnh “`context = {'form': form}'`”.

Nếu phương thức yêu cầu là POST, khối else sẽ chạy và xử lý dữ liệu được gửi trong biểu mẫu. Chúng ta tạo một phiên bản của TopicForm và chuyển nó dữ liệu do người dùng nhập vào, được lưu trữ trong `request.POST`. Đối tượng form được trả về chứa thông tin do người dùng gửi.

Chúng ta không thể lưu thông tin đã gửi trong cơ sở dữ liệu cho đến khi kiểm tra được rằng thông tin đó là hợp lệ. Phương thức `is_valid()` kiểm tra xem tất cả các trường bắt buộc đã được điền vào hay chưa (tất cả các trường trong biểu mẫu đều được yêu cầu theo mặc định) và dữ liệu đã nhập khớp với loại trường mong đợi — ví dụ: độ dài của văn bản dưới 200 ký tự, như chúng ta đã chỉ định trong `models.py` trong Chương trước. Việc xác thực tự động này giúp chúng ta tiết kiệm rất nhiều công việc. Nếu mọi thứ hợp lệ, chúng ta có thể gọi `save()`, nó ghi dữ liệu từ biểu mẫu vào cơ sở dữ liệu.

Sau khi đã lưu dữ liệu, chúng ta có thể rời khỏi trang này. Chúng ta sử dụng `redirect()` để chuyển hướng trình duyệt của người dùng đến trang chủ đề, nơi người dùng sẽ thấy chủ đề họ vừa nhập trong danh sách chủ đề.

Biến ngữ cảnh `context` được xác định ở cuối hàm `views` và trang được hiển thị bằng cách sử dụng mẫu `new_topic.html` mà chúng ta sẽ tạo tiếp theo. Mã này được đặt bên ngoài bất kỳ khối `if` nào; nó sẽ chạy nếu một biểu mẫu trống được tạo và nó sẽ chạy nếu một biểu mẫu đã gửi được xác định là không hợp lệ. Biểu mẫu không hợp lệ sẽ bao gồm một số thông báo lỗi mặc định để giúp người dùng gửi dữ liệu có thể chấp nhận được.

Mẫu new_topic

Bây giờ, chúng ta sẽ tạo một mẫu mới có tên là `new_topic.html` để hiển thị biểu mẫu chúng ta vừa tạo.

```
{% extends "learning_logs/base.html" %}
```

```

{%- block content %}

    <p>Add a new topic:</p>
    <form action="{% url 'learning_logs:new_topic' %}" method='post'>
        {% csrf_token %}
        {{ form.as_p }}
        <button name="submit">Add topic</button>
    </form>

{%- endblock content %}

```

Mẫu này mở rộng base.html, vì vậy nó có cấu trúc cơ sở giống như các trang còn lại trong Nhật ký Học tập. Đầu tiên chúng ta xác định một biểu mẫu HTML. Đổi số action cho trình duyệt biết nơi gửi dữ liệu; trong trường hợp này, chúng ta gửi nó trở lại hàm view new_topic(). Đổi số phương thức yêu cầu trình duyệt gửi dữ liệu dưới dạng yêu cầu POST.

Django sử dụng thẻ mẫu `{% csrf_token%}` để ngăn những kẻ tấn công sử dụng biểu mẫu này truy cập trái phép vào máy chủ (kiểu tấn công này được gọi là giả mạo yêu cầu trên nhiều trang web). Sau đó, chúng ta hiển thị biểu mẫu; ta sẽ thấy Django có thể thực hiện một số tác vụ đơn giản như thế nào, chẳng hạn như hiển thị biểu mẫu. Chúng ta chỉ cần biến mẫu `{{form.as_p}}` cho Django để tạo tất cả các trường cần thiết nhằm hiển thị biểu mẫu tự động. Công cụ sửa đổi `as_p` yêu cầu Django hiển thị tất cả các thành phần biểu mẫu ở định dạng đoạn văn, như một cách đơn giản để hiển thị biểu mẫu một cách gọn gàng.

Liên kết đến Trang new_topic

Tiếp theo, tạo một liên kết đến trang new_topic trên trang chủ đề:

```

{%- extends "learning_logs/base.html" %}

{%- block content %}

    <p>Topics</p>
    <ul>
        --snip--
    </ul>
    <a href="{% url 'learning_logs:new_topic' %}">Add a new topic</a>
{%- endblock content %}

```

Đặt liên kết sau danh sách các chủ đề hiện có. Hình dưới đây cho thấy dạng kết quả. Sử dụng biểu mẫu để thêm một vài chủ đề mới của riêng ta.



Hình 10.41. Trang thêm chủ đề

b. Thêm các mục mới:

Bây giờ người dùng có thể thêm một chủ đề mới, và họ cũng muốn thêm các mục mới. Chúng ta sẽ xác định lại một URL, viết một chức năng views, một mẫu và liên kết đến trang. Nhưng trước tiên, chúng ta sẽ thêm một lớp khác vào form.py.

Mô hình mục nhập

Chúng ta cần tạo một biểu mẫu liên kết với Entry model nhưng lần này có nhiều tùy chỉnh hơn TopicForm:

```
from django import forms
from .models import Topic, Entry
class TopicForm(forms.ModelForm):
    --snip--
class EntryForm(forms.ModelForm):
    class Meta:
        model = Entry
        fields = ['text']
        labels = {'text': 'Entry:'}
        widgets = {'text': forms.Textarea(attrs={'cols': 80})}
```

Chúng ta tạo một lớp mới được gọi là EntryForm kế thừa từ các form.ModelForm. Các Lớp EntryForm có một lớp Meta lồng nhau để liệt kê mô hình mà nó dựa trên và trường để đưa vào biểu mẫu.

widgets là một định dạng HTML, chẳng hạn như hộp văn bản một dòng, vùng văn bản nhiều dòng hoặc danh sách thả xuống. Bằng cách bao gồm thuộc tính widget,

ta có thể ghi đè các lựa chọn widget mặc định của Django. Bằng cách yêu cầu Django sử dụng phần tử biểu mẫu.

URL new_entry

Các mục nhập mới phải được liên kết với một chủ đề cụ thể, vì vậy chúng ta cần có đối số topic_id trong URL để thêm mục nhập mới. Đây là URL ta sẽ thêm vào learning_logs / urls.py:

```
--snip--  
urlpatterns = [  
    --snip--  
    # Page for adding a new entry  
    path('new_entry/<int:topic_id>', views.new_entry,  
name='new_entry'),  
]
```

Mẫu URL này khớp với bất kỳ URL nào có dạng http://localhost:8000 / new_entry / id /, trong đó id là một số khớp với ID chủ đề. Đoạn mã <int: topic_id> nắm bắt một giá trị số và gán nó cho biến topic_id. Khi một URL khớp với mẫu này được yêu cầu, Django sẽ gửi yêu cầu và chủ đề của ID đến hàm views.new_entry().

Hàm view new_entry()

Hàm view cho new_entry giống như hàm để thêm một chủ đề mới. Thêm mã sau vào tệp views.py :

```
from django.shortcuts import render, redirect  
from .models import Topic  
from .forms import TopicForm, EntryForm  
--snip--  
  
def new_entry(request, topic_id):  
    """Add a new entry for a particular topic."""  
    topic = Topic.objects.get(id=topic_id)  
  
    if request.method != 'POST':  
        # No data submitted; create a blank form.  
        form = EntryForm()  
    else:  
        # POST data submitted; process data.  
        form = EntryForm(data=request.POST)  
        if form.is_valid():  
            new_entry = form.save(commit=False)  
            new_entry.topic = topic  
            new_entry.save()  
    return redirect('learning_logs:topic', topic_id=topic_id)
```

```

# Display a blank or invalid form.
context = {'topic': topic, 'form': form}
return render(request, 'learning_logs/new_entry.html', context)

```

Định nghĩa hàm new_entry() có tham số topic_id để lưu trữ giá trị mà nó nhận được từ URL. Chúng ta sẽ cần chủ đề để hiển thị trang và xử lý dữ liệu của biểu mẫu, vì vậy cần sử dụng topic_id để có được đối tượng chủ đề chính xác.

Tiếp theo, chúng ta kiểm tra xem phương thức yêu cầu là POST hay GET. Khỏi if thực thi nếu đó là yêu cầu GET thì tạo một phiên bản EntryForm.

Nếu phương thức yêu cầu là POST, hãy xử lý dữ liệu bằng cách tạo một phiên bản EntryForm, được điền với dữ liệu POST từ request. Sau đó, chúng ta kiểm tra xem biểu mẫu có hợp lệ hay không. Nếu có, chúng ta cần đặt thuộc tính chủ đề của đối tượng mục nhập trước khi lưu nó vào cơ sở dữ liệu. Khi gọi save(), đối số commit = False để yêu cầu Django tạo một mục nhập mới và gán nó cho new_entry mà không cần lưu nó vào cơ sở dữ liệu. Chúng ta đặt thuộc tính của new_entry bằng với chủ đề mà ta đã lấy từ cơ sở dữ liệu ở đầu hàm “new_entry.topic = topic”. Sau đó, gọi save(), lưu mục nhập vào cơ sở dữ liệu với chủ đề liên quan chính xác.

Lệnh gọi redirect() yêu cầu hai đối số - tên của chế độ xem chúng ta muốn chuyển hướng đến và đối số mà hàm view. Ở đây, chúng ta đang chuyển hướng đến topic(), cần đổi số topic_id. Sau đó, chế độ xem này hiển thị trang chủ đề mà người dùng đã tạo mục nhập và họ sẽ thấy mục nhập mới của mình trong danh sách các mục nhập. Ở cuối hàm, chúng ta tạo một từ điển ngữ cảnh context và hiển thị trang bằng mẫu new_entry.html. Mã này sẽ thực thi đối với biểu mẫu trống hoặc đối với biểu mẫu đã gửi được đánh giá là không hợp lệ.

new_entry Template

Như ta có thể thấy trong đoạn mã sau, mẫu cho new_entry tương tự như mẫu cho new_topic(file new_entry.html):

```

{% extends "learning_logs/base.html" %}
{% block content %}
    <p><a href="{% url 'learning_logs:topic' topic.id %}">{{ topic }}</a></p>
    <p>Add a new entry:</p>
    <form action="{% url 'learning_logs:new_entry' topic.id %}" method='post'>
        {% csrf_token %}

```

```

{{ form.as_p }}
<button name='submit'>Add entry</button>
</form>
{% endblock content %}

```

Chúng ta hiển thị chủ đề ở đầu trang , vì vậy người dùng có thể thấy chủ đề mà họ đang thêm.

Đối số action của biểu mẫu bao gồm giá trị topic_id trong URL, vì vậy, hàm view có thể liên kết entry mới với chủ đề topic chính xác. Ngoài ra, mẫu này trông giống như new_topic.html.

Liên kết đến Trang new_entry

Tiếp theo, chúng ta cần tạo một liên kết đến trang new_entry từ mỗi trang chủ đề: (topic.html)

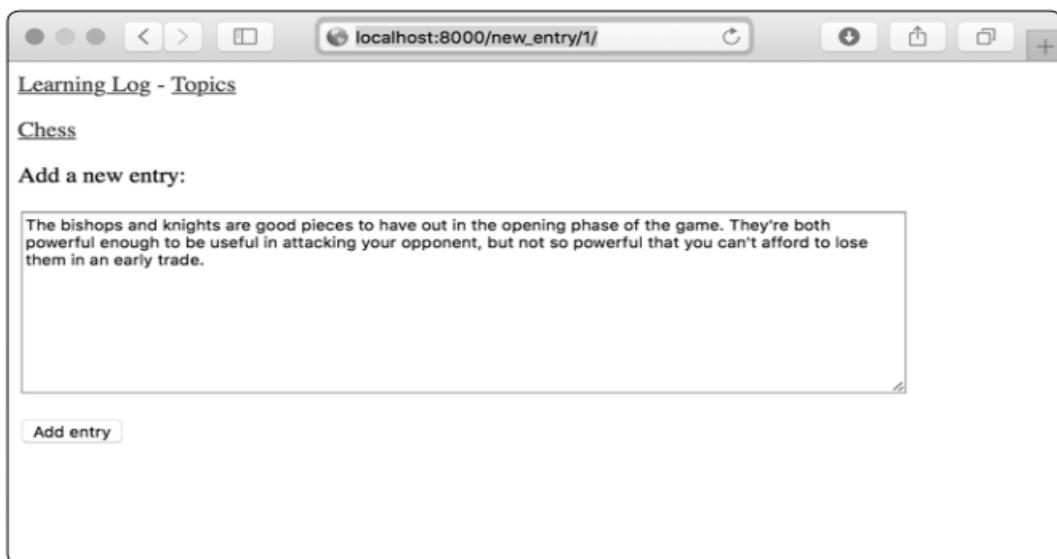
```

{% extends "learning_logs/base.html" %}
{% block content %}
<p>Topic: {{ topic }}</p>

<p>Entries:</p>
<p>
<a href="{% url 'learning_logs:new_entry' topic.id %}">Add new
entry</a>
</p>
<ul>
--snip--
</ul>
{% endblock content %}

```

Chúng ta tạo liên kết để thêm mục nhập ngay trước khi hiển thị mục nhập, vì thêm mục nhập mới sẽ là hành động phổ biến nhất trên trang này. Hình dưới đây hiển thị trang new_entry. Giờ đây, người dùng có thể thêm các chủ đề mới và bao nhiêu mục nhập tùy thích cho mỗi chủ đề. Hãy thử trang new_entry bằng cách thêm một vài mục nhập vào một số chủ đề ta đã tạo.



Hình 10.42. Trang new_entry

c. Chính sửa Mục nhập

Bây giờ chúng ta sẽ tạo một trang để người dùng có thể chỉnh sửa các mục họ đã thêm.

Edit_entry URL

Đường dẫn tới trang cần ID của mục nhập entry .Dưới đây là mã của learning_logs/urls.py:

```
--snip--
urlpatterns = [
    --snip--
    # Page for editing an entry.
    path('edit_entry/<int:entry_id>', views.edit_entry, name='edit_entry'),
]
```

ID được chuyển vào URL (ví dụ: http://localhost:8000/edit_entry/1/) được lưu trữ trong tham số entry_id. Mẫu URL gửi các yêu cầu phù hợp với định dạng này đến hàm view edit_entry().

Hàm view edit_entry():

Khi trang edit_entry nhận được yêu cầu GET, hàm edit_entry() trả về một biểu mẫu để chỉnh sửa mục nhập. Khi trang nhận được một yêu cầu POST với văn bản mục nhập đã sửa đổi, nó lưu văn bản đã sửa đổi vào cơ sở dữ liệu:

```
from django.shortcuts import render, redirect
from .models import Topic, Entry
from .forms import TopicForm, EntryForm
```

```
--snip--
def edit_entry(request, entry_id):
    """Edit an existing entry."""
    entry = Entry.objects.get(id=entry_id)
    topic = entry.topic

    if request.method != 'POST':
        # Initial request; pre-fill form with the current entry.
        form = EntryForm(instance=entry)
    else:
        # POST data submitted; process data.
        form = EntryForm(instance=entry, data=request.POST)
        if form.is_valid():
            form.save()
    return redirect('learning_logs:topic', topic_id=topic.id)
context = {'entry': entry, 'topic': topic, 'form': form}
return render(request, 'learning_logs/edit_entry.html', context)
```

Đầu tiên, chúng ta nhận được mục nhập mà người dùng muốn chỉnh sửa và chủ đề được liên kết với mục nhập này. Trong khôi if, khi yêu cầu là GET, chúng ta tạo một phiên bản EntryForm với đối số instance = entry. Đối số này yêu cầu Django tạo biểu mẫu được điền sẵn thông tin từ đối tượng mục nhập hiện có. Người dùng sẽ thấy dữ liệu hiện có của họ và có thể chỉnh sửa dữ liệu đó.

Khi xử lý một yêu cầu POST, hãy truyền đối số instance = entry và đối số data = request.POST. Các đối số này yêu cầu Django tạo ra một cá thể biểu mẫu dựa trên thông tin được liên kết với đối tượng mục nhập hiện có, được cập nhật với bất kỳ dữ liệu liên quan nào từ request.POST. Sau đó, chúng ta kiểm tra xem biểu mẫu có hợp lệ hay không; nếu hợp lệ hãy gọi save() bởi vì mục nhập đã được kết hợp với chủ đề chính xác. Sau đó, chúng ta chuyển hướng đến trang chủ đề, nơi người dùng sẽ thấy phiên bản cập nhật của mục nhập mà họ đã chỉnh sửa.

Nếu chúng đang hiển thị biểu mẫu ban đầu hoặc nếu biểu mẫu đã gửi không hợp lệ, chúng ta tạo từ điển ngữ cảnh context và hiển thị trang bằng mẫu edit_entry.html.

Mẫu edit_entry

Tiếp theo, chúng ta tạo một mẫu edit_entry.html, tương tự như new_entry.html:

```
{% extends "learning_logs/base.html" %}
{% block content %}
    <p><a href="{% url 'learning_logs:topic' topic.id %}">{{ topic
} }</a></p>
```

```

<p>Edit entry:</p>
<form action="{% url 'learning_logs:edit_entry' entry.id %}"
method='post'>
    {% csrf_token %}
    {{ form.as_p }}
    <button name="submit">Save changes</button>
</form>
{% endblock content %}

```

Đối số action sẽ gửi biểu mẫu trả lại hàm edit_entry() để xử lý. Chúng ta có ID của mục nhập làm đối số trong {% url%}, do đó, chức năng xem có thể sửa đổi đối tượng mục nhập chính xác. Gắn nhãn nút gửi dưới dạng Save change để nhắc nhở người dùng rằng họ đang lưu các chỉnh sửa, không phải tạo một mục mới.

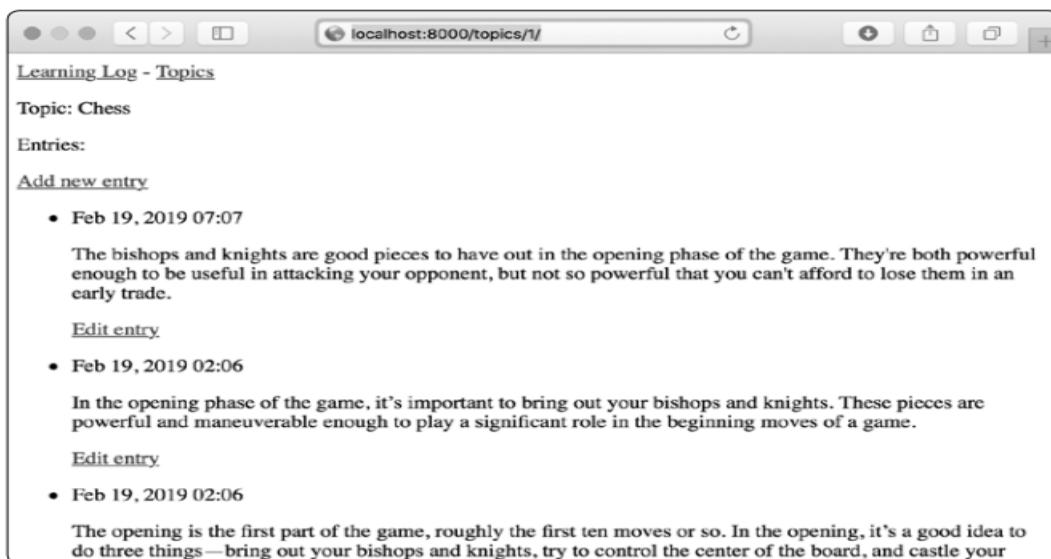
Liên kết tới trang edit_entry:

```

--snip--
{% for entry in entries %}
<li>
<p>{{ entry.date_added|date:'M d, Y H:i' }}</p>
<p>{{ entry.text|linebreaks }}</p>
<p>
<a href="{% url 'learning_logs:edit_entry' entry.id %}">Edit entry</a>
</p>
</li>
--snip--

```

Chúng ta có liên kết chỉnh sửa sau khi ngày và văn bản của mỗi mục nhập đã được hiển thị. Ta sử dụng thẻ mầu {% url%} để xác định URL cho mâu URL edit_entry được đặt tên, cùng với thuộc tính ID của mục nhập hiện tại trong vòng lặp (entry.id). Cụm từ *Edit entry* xuất hiện sau mỗi mục nhập trên trang. Hình dưới đây cho thấy trang chủ để trông sẽ như thế nào với các liên kết này.



Hình 10.43. Liên kết chỉnh sửa Entry

Nhật ký học tập hiện có hầu hết các chức năng mà nó cần. Người dùng có thể thêm chủ đề và mục nhập và đọc qua bất kỳ nhóm mục nhập nào họ muốn. Trong phần tiếp theo, chúng ta sẽ triển khai hệ thống đăng ký người dùng để bất kỳ ai cũng có thể tạo tài khoản bằng Nhật ký học tập và tạo nhóm chủ đề của riêng họ.

10.3.2.2. Thiết lập tài khoản người dùng

Trong phần này, chúng ta sẽ thiết lập hệ thống đăng ký và ủy quyền cho người dùng để mọi người có thể đăng ký tài khoản, đăng nhập và đăng xuất. Chúng ta sẽ tạo một ứng dụng mới để chứa tất cả các chức năng liên quan đến hoạt động của người dùng. Sử dụng hệ thống xác thực người dùng mặc định đi kèm với Django để thực hiện nhiều công việc nhất có thể. Chúng ta cũng sẽ sửa đổi một chút mô hình Topic để mọi chủ đề thuộc về một người dùng nhất định.

a. Ứng dụng người dùng

Chúng ta sẽ bắt đầu bằng cách tạo một ứng dụng mới có tên là User, sử dụng lệnh startapp:

```
(ll_env) learning_log$ python manage.py startapp users
(ll_env) learning_log$ ls
db.sqlite3 learning_log learning_logs ll_env manage.py users
(ll_env) learning_log$ ls users
__init__.py admin.py apps.py migrations models.py tests.py views.py
```

Lệnh này tạo một thư mục mới có tên là users với cấu trúc giống với ứng dụng learning_logs.

Thêm users vào settings.py

Chúng ta cần thêm ứng dụng mới của mình vào INSTALLED_APPS trong settings.py, tương tự như vậy:

```
--snip--  
INSTALLED_APPS = [  
    # My apps  
    'learning_logs',  
    'users',  
    # Default django apps.  
    --snip--  
]  
--snip--
```

Các URL từ người dùng

Tiếp theo, chúng ta cần sửa đổi urls.py để nó có các URL mà chúng ta sẽ viết cho ứng dụng người dùng:

```
from django.contrib import admin  
from django.urls import path, include  
urlpatterns = [  
    path('admin/', admin.site.urls),  
    path('users/', include('users.urls')),  
    path('', include('learning_logs.urls')),  
]
```

Chúng ta thêm một dòng mã để thêm tệp urls.py từ người dùng. Dòng này sẽ khớp với bất kỳ URL nào bắt đầu bằng từ người dùng, chẳng hạn như `http://localhost: 8000 / users / login /`.

b. Trang login

Trước tiên, chúng ta sẽ triển khai một trang login để đăng nhập. Hãy sử dụng chế độ xem đăng nhập mặc định mà Django cung cấp, vì vậy mẫu URL cho ứng dụng này trông hơi khác một chút. Tạo một tệp urls.py mới trong thư mục learning_log / users / và thêm thông tin sau vào đó:

```
"""Defines URL patterns for users"""  
from django.urls import path, include  
app_name = 'users'  
urlpatterns = [  
    # Include default auth urls.  
    path('', include('django.contrib.auth.urls')),  
]
```

Chúng ta import hàm đường dẫn path, hàm include để ta có thể có một số URL xác thực mặc định mà Django đã xác định. Các URL mặc định này bao gồm các mẫu URL được đặt tên, chẳng hạn như 'login' và 'logout'. Ta đặt biến app_name thành 'users' để Django có thể phân biệt các URL này với các URL thuộc các ứng dụng khác.

Mẫu của trang đăng nhập khớp với URL `http://localhost:8000/users/login/`. Khi Django đọc URL này, từ "users" yêu cầu Django tìm kiếm trong `users/urls.py` và `login`.

Mẫu đăng nhập

Khi người dùng yêu cầu đến trang đăng nhập, Django sẽ sử dụng một chức năng xem mặc định, tuy nhiên chúng ta vẫn cần một mẫu template cho trang. Bên trong thư mục `learning_log/users/`, tạo một thư mục có tên là `template`; bên trong đó, tạo một thư mục khác được gọi là `registration`. Dưới đây là mẫu `login.html` mà ta nên lưu trong `learning_log/users/templates/register`:

```
{% extends "learning_logs/base.html" %}  
{% block content %}  
{% if form.errors %}  
    <p>Your username and password didn't match. Please try again.</p>  
{% endif %}  
<form method="post" action="{% url 'users:login' %}">  
    {% csrf_token %}  
    {{ form.as_p }}  
  
    <button name="submit">Log in</button>  
    <input type="hidden" name="next"  
    value="{% url 'learning_logs:index' %}" />  
</form>  
{% endblock content %}
```

Mẫu này kế thừa `base.html` để đảm bảo rằng trang đăng nhập sẽ có giao diện giống như phần còn lại của trang web. Lưu ý rằng mẫu trong một ứng dụng có thể kế thừa từ mẫu trong ứng dụng khác. Nếu thuộc tính `lỗi` của biểu mẫu được đặt, chúng ta sẽ hiển thị thông báo lỗi để báo cáo rằng tên người dùng và mật khẩu không khớp với dữ liệu được lưu trữ trong cơ sở dữ liệu.

Chúng ta muốn chế độ đăng nhập xử lý biểu mẫu, hãy đặt đối số `action` làm URL của trang `login`. Chế độ đăng nhập sẽ gửi biểu mẫu đến template và thêm một

nút submit. Chúng tôi có một phần tử loại “hidden” tên “name” để cho Django biết nơi chuyển hướng người dùng sau khi họ đăng nhập thành công. Trong trường hợp này, chúng ta đưa người dùng trở lại trang chủ.

Liên kết đến Trang đăng nhập

Hãy thêm liên kết đăng nhập vào base.html để nó xuất hiện trên mọi trang. Chúng ta không muốn liên kết hiển thị khi người dùng đã đăng nhập, vì vậy hãy lồng nó vào bên trong thẻ {% if%}:

```
<p>
    <a href="{% url 'learning_logs:index' %}">Learning Log</a> -
    <a href="{% url 'learning_logs:topics' %}">Topics</a> -
    {% if user.is_authenticated %}
        Hello, {{ user.username }}.
    {% else %}
        <a href="{% url 'users:login' %}">Log in</a>
    {% endif %}
</p>
{% block content %}{% endblock content %}
```

Trong hệ thống xác thực của Django, mọi mẫu đều có sẵn biến người dùng, biến này luôn có bộ thuộc tính `is_authenticated`: thuộc tính là `True` nếu người dùng đã đăng nhập và `False` nếu chưa đăng nhập. Thuộc tính này cho phép hiển thị một thông báo cho người dùng đã xác thực và một thông báo khác cho người dùng chưa xác thực.

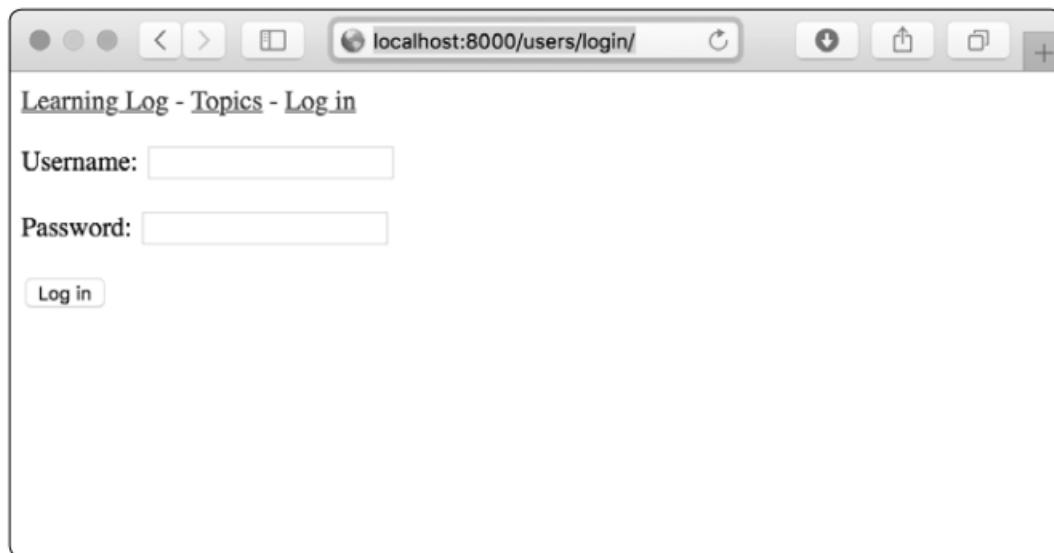
Chúng ta sẽ hiển thị lời chào tới người dùng hiện đang đăng nhập . Người dùng được xác thực có một bộ thuộc tính tên `username` sẵn có, chúng ta sử dụng để đưa ra lời chào và nhắc nhở người dùng họ đã đăng nhập . Nếu user chưa được xác thực thì hãy hiển thị liên kết đến trang đăng nhập login.

Sử dụng trang Login

Chúng ta đã thiết lập một tài khoản người dùng, vì vậy hãy đăng nhập để xem trang có hoạt động hay không. Truy cập `http://localhost:8000/admin/`. Nếu ta vẫn đăng nhập với tư cách quản trị viên, hãy tìm liên kết đăng xuất trong tiêu đề và nhấp vào liên kết đó.

Khi ta đã đăng xuất, hãy truy cập `http://localhost:8000/users/login/`. Ta sẽ thấy một trang đăng nhập tương tự như trong Hình dưới đây. Nhập tên người dùng và mật khẩu ta đã thiết lập trước đó và ta nên trở lại trang chỉ mục index. Tiêu đề trên

trang chủ phải hiển thị lời chào được cá nhân hóa với tên người dùng của mà ta vừa đăng kí.



Hình 10.44. Trang Login

c. Đăng xuất

Bây giờ chúng ta cần đăng xuất tài khoản người dùng. Hãy đặt một liên kết trong base.html đăng xuất người dùng; khi nhấp vào liên kết này, chúng sẽ chuyển đến trang xác nhận rằng chúng đã đăng xuất.

Thêm một liên kết đăng xuất trong base.html

Chúng ta sẽ đưa nó vào phần `{% if user.is_authenticated %}` để chỉ những người dùng đã đăng nhập rồi mới có thể đăng xuất được:

```
--snip--  
  {% if user.is_authenticated %}  
    Hello, {{ user.username }}.  
    <a href="{% url 'users:logout' %}">Log out</a>  
  {% else %}  
--snip--
```

Trang xác nhận đăng xuất

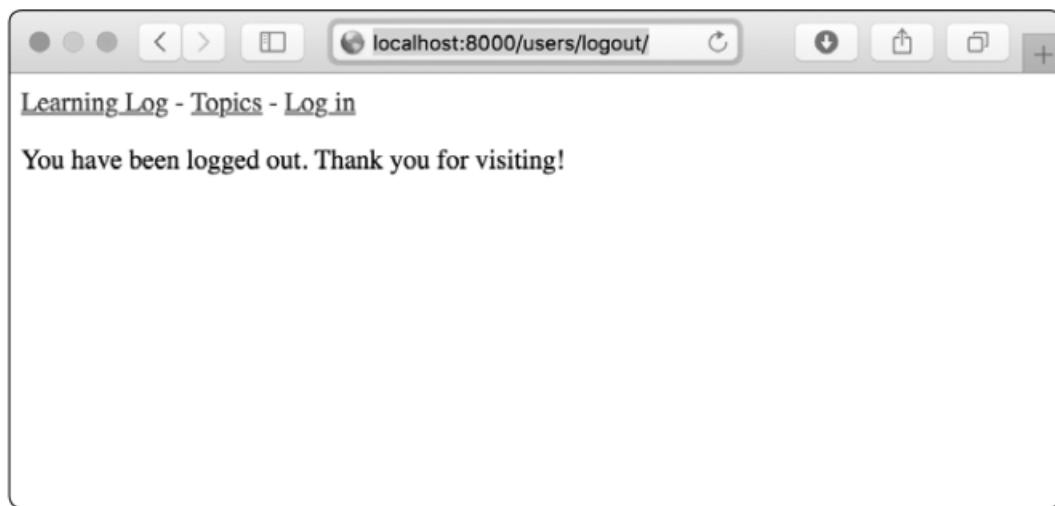
Để người dùng biết họ đã đăng xuất thành công, hãy hiển thị trang xác nhận bằng cách sử dụng mẫu logged_out.html mà chúng ta sẽ tạo ngay sau đây. Đây là một trang đơn giản xác nhận rằng người dùng đã đăng xuất. Lưu tệp này trong template/registration, cùng nơi ta đã lưu login.html:

```
{% extends "learning_logs/base.html" %}  
{% block content %}
```

```
<p>You have been logged out. Thank you for visiting!</p>
{%
    endblock content %}
```

Chúng ta không cần bất kỳ thứ gì khác trên trang đăng xuất này, vì base.html cung cấp các liên kết quay lại trang chủ và trang đăng nhập nếu người dùng muốn quay lại một trong hai trang.

Hình dưới đây cho thấy trang đã đăng xuất khi nó xuất hiện với người dùng vừa nhấp vào liên kết Log out.



Hình 10.45. Xác nhận đăng xuất thành công

d. Trang đăng kí

Tiếp theo chúng ta sẽ xây dựng một rang mà người dùng mới có thể đăng kí. Sử dụng UserCreationForm của Django nhưng ta tự viết hàm mà mẫu template của riêng mình.

URL đăng kí

Đoạn mã sau đây cung cấp đường dẫn URL tới trang đăng kí, hiện trong file users/urls.py:

```
"""Defines URL patterns for users"""
from django.urls import path, include
from . import views
app_name = 'users'
urlpatterns = [
    # Include default auth urls.
    path('', include('django.contrib.auth.urls')),
    # Registration page.
    path('register/', views.register, name='register'),
]
```

Hàm register()

Hàm register() cần hiển thị một biểu mẫu đăng ký trống khi trang đăng ký được yêu cầu lần đầu tiên và sau đó xử lý các biểu mẫu đăng ký đã hoàn thành khi chúng được gửi. Khi đăng ký thành công, chức năng cũng cần đăng nhập người dùng mới. Thêm mã sau vào users / views.py:

```
from django.shortcuts import render, redirect
from django.contrib.auth import login
from django.contrib.auth.forms import UserCreationForm
def register(request):
    """Register a new user."""
    if request.method != 'POST':
        # Display blank registration form.
        form = UserCreationForm()
    else:
        # Process completed form.
        form = UserCreationForm(data=request.POST)

        if form.is_valid():
            new_user = form.save()
            # Log the user in and then redirect to home page.
            login(request, new_user)
            return redirect('learning_logs:index')
        # Display a blank or invalid form.
        context = {'form': form}
        return render(request, 'registration/register.html', context)
```

Hãy import render() và hàm redirect(). Sau đó import hàm login() để chỉ ra người dùng đã đăng ký thông tin chính xác. Chúng ta cũng import hàm mặc định UserCreationForm. Tiếp đó, hãy kiểm tra xem chúng ta có đang phản hồi yêu cầu POST hay không. Nếu không, hãy tạo một phiên bản của UserCreationForm không có dữ liệu ban đầu.

Ngược lại, nếu chúng ta đang phản hồi yêu cầu POST, hãy tạo một phiên bản form của UserCreationForm dựa trên dữ liệu đã gửi . Chúng ta kiểm tra xem dữ liệu có hợp lệ không, nếu có, tên người dùng có các ký tự thích hợp, mật khẩu khớp và người dùng không cố gắng làm bất cứ điều gì độc hại trong quá trình gửi của họ.

Nếu dữ liệu đã gửi là hợp lệ, chúng ta gọi phương thức save() để lưu tên người dùng và băm mật khẩu vào cơ sở dữ liệu . Phương thức save() trả về đối tượng người dùng mới được tạo, đối tượng mà chúng ta gán cho new_user. Khi thông tin của người

dùng được lưu, chúng ta đăng nhập chúng bằng cách gọi hàm login() với các đối tượng request và new_user , hàm này tạo ra một phiên hợp lệ cho người dùng mới. Cuối cùng, chúng ta chuyển hướng người dùng đến trang chủ , nơi một lời chào được cá nhân hóa trong tiêu đề cho họ biết đăng ký của họ đã thành công.

Template đăng kí

Bây giờ hãy tạo một template cho trang đăng kí, nó giống với trang login. Tên file là register.html:

```
{% extends "learning_logs/base.html" %}  
{% block content %}  
    <form method="post" action="{% url 'users:register' %}">  
        {% csrf_token %}  
        {{ form.as_p }}  
        <button name="submit">Register</button>  
        <input type="hidden" name="next" value="{% url  
'learning_logs:index' %}" />  
    </form>  
{% endblock content %}
```

Liên kết đến trang đăng kí:

Tạo mã để liên kết đến trang đăng kí cho những người dùng chưa đăng nhập:

```
--snip--  
    {% if user.is_authenticated %}  
        Hello, {{ user.username }}.  
        <a href="{% url 'users:logout' %}">Log out</a>  
    {% else %}  
        <a href="{% url 'users:register' %}">Register</a> -  
        <a href="{% url 'users:login' %}">Log in</a>  
    {% endif %}  
--snip--
```

Chú ý : Hệ thống đăng ký mà chúng ta đã thiết lập cho phép mọi người tạo bất kỳ số lượng tài khoản nào cho Nhật ký học tập. Nhưng một số hệ thống yêu cầu người dùng xác nhận danh tính của họ bằng cách gửi email xác nhận mà người dùng phải trả lời. Bằng cách đó, hệ thống tạo ra ít tài khoản spam hơn hệ thống đơn giản mà chúng ta đang sử dụng ở đây. Tuy nhiên, khi ta đang học cách xây dựng ứng dụng, ta hoàn toàn thích hợp để thực hành với hệ thống đăng ký người dùng đơn giản như hệ thống chúng ta đang sử dụng.

10.3.2.3. Cho phép người dùng sở hữu dữ liệu của họ

Người dùng có thể nhập dữ liệu dành riêng cho họ, vì vậy chúng ta sẽ tạo một hệ thống để tìm ra dữ liệu nào thuộc về người dùng nào. Sau đó, hạn chế quyền truy cập vào các trang nhất định để người dùng chỉ có thể làm việc với dữ liệu của riêng họ.

Chúng ta sẽ sửa đổi mô hình Topic để mọi chủ đề thuộc về một người dùng cụ thể. Hãy bắt đầu bằng cách hạn chế quyền truy cập vào các trang nhất định.

a. Hạn chế quyền truy cập với @login_required

Django giúp dễ dàng hạn chế quyền truy cập vào các trang nhất định đối với người dùng đã đăng nhập thông qua `@login_required`. Decorator là một chỉ thị được đặt ngay trước định nghĩa hàm mà Python áp dụng cho hàm trước khi nó chạy, để thay đổi cách hoạt động của mã hàm. Hãy xem một ví dụ.

Hạn chế quyền truy cập tới trang Topic:

Mỗi chủ đề sẽ có người dùng sở hữu riêng, vì vậy người dùng có thể yêu cầu đến trang Topic của họ. Hãy thêm mã sau vào `learning_log/views.py`:

```
from django.shortcuts import render, redirect
from django.contrib.auth.decorators import login_required
from .models import Topic, Entry
--snip--
@login_required
def topics(request):
    """Show all topics."""
--snip--
```

Mã trong `login_required()` kiểm tra xem người dùng đã đăng nhập hay chưa và Django chỉ chạy mã trong các `topics()` nếu họ có. Nếu người dùng chưa đăng nhập, chúng sẽ được chuyển hướng đến trang đăng nhập.

Để làm cho chuyển hướng này hoạt động ta cần sửa đổi `settings.py` để Django biết nơi tìm trang đăng nhập. Thêm phần sau vào cuối `settings.py`.

```
--snip--
# My settings
LOGIN_URL = 'users:login'
```

Giờ đây, khi người dùng chưa được xác thực yêu cầu tới một trang được bảo vệ bởi `@login_required`, Django sẽ đưa người dùng đến URL được xác định bởi `LOGIN_URL` trong `settings.py`.

Ta có thể kiểm tra cài đặt này bằng cách đăng xuất khỏi bất kỳ tài khoản người dùng nào và truy cập trang chủ. Nhập vào liên kết Topics, liên kết này sẽ chuyển hướng ta đến trang đăng nhập. Sau đó, đăng nhập vào bất kỳ tài khoản nào của mình và từ trang chủ, hãy nhập lại vào liên kết Topics.

Hạn chế quyền truy cập trong Nhật ký học tập

Django giúp ta dễ dàng hạn chế quyền truy cập vào các trang, nhưng chúng ta phải có quyết định bảo vệ trang nào. Tốt nhất ta nên nghĩ xem trang nào không bị hạn chế trước, sau đó hạn chế tất cả các trang còn lại trong dự án. Ta có thể dễ dàng sửa quyền truy cập vượt quá định mức và ít nguy hiểm hơn là để các trang nhạy cảm không bị hạn chế. Trong Nhật ký học tập, chúng ta sẽ giữ trang chủ và trang đăng ký không bị giới hạn và hạn chế quyền truy cập vào mọi trang khác.

Đây là learning_logs / views.py với @login_required decorator được áp dụng cho mọi chế độ xem ngoại trừ index():

```
--snip--  
@login_required  
def topics(request):  
    --snip--  
@login_required  
def topic(request, topic_id):  
    --snip--  
@login_required  
def new_topic(request):  
    --snip--  
  
@login_required  
def new_entry(request, topic_id):  
    --snip--  
@login_required  
def edit_entry(request, entry_id):  
    --snip--
```

Thử truy cập từng trang này khi đã đăng xuất: chúng ta sẽ được chuyển hướng trở lại trang đăng nhập, ta cũng sẽ không thể nhập vào liên kết đến các trang như new_topic. Ta nên hạn chế quyền truy cập vào bất kỳ URL nào có thể truy cập công khai và liên quan đến dữ liệu người dùng riêng tư.

b. Kết nối dữ liệu với một số người dùng nhất định

Tiếp theo, chúng ta cần kết nối dữ liệu với người dùng đã gửi nó. Chúng ta chỉ cần kết nối dữ liệu cao nhất trong hệ thống phân cấp với người dùng và dữ liệu cấp

thấp hơn sẽ theo sau. Ví dụ: trong Nhật ký học tập, topics là cấp dữ liệu cao nhất trong ứng dụng và tất cả các mục nhập đều được kết nối với một chủ đề. Miễn là mỗi chủ đề thuộc về một người dùng cụ thể, chúng ta có thể theo dõi quyền sở hữu của mỗi mục nhập trong cơ sở dữ liệu.

Chúng ta sẽ sửa đổi mô hình Topics bằng cách thêm mối quan hệ khóa ngoài cho người dùng. Sau đó di chuyển cơ sở dữ liệu. Cuối cùng, chúng ta sẽ sửa đổi một số chế độ xem để chúng chỉ hiển thị dữ liệu được liên kết với người dùng hiện đang đăng nhập.

Sửa đổi mô hình Topic:

Việc sửa đổi models.py chỉ có hai dòng:

```
from django.db import models
from django.contrib.auth.models import User
class Topic(models.Model):
    """A topic the user is learning about."""
    text = models.CharField(max_length=200)
    date_added = models.DateTimeField(auto_now_add=True)
    owner = models.ForeignKey(User, on_delete=models.CASCADE)
    def __str__(self):
        """Return a string representation of the model."""
        return self.text
class Entry(models.Model):
--snip--
```

Chúng ta thêm trường chủ sở hữu vào Topic, trường này thiết lập mối quan hệ khóa ngoại với mô hình User. Nếu người dùng bị xóa, tất cả các chủ đề được liên kết với người dùng đó cũng sẽ bị xóa.

Xác định Người dùng Hiện tại:

Khi chúng ta di chuyển cơ sở dữ liệu, Django sẽ sửa đổi cơ sở dữ liệu để nó có thể lưu trữ kết nối giữa mỗi chủ đề và người dùng. Để thực hiện việc di chuyển, Django cần biết người dùng nào sẽ kết hợp với từng chủ đề hiện có. Các cách tiếp cận đơn giản nhất là bắt đầu bằng cách đưa tất cả các chủ đề hiện có cho một người dùng — ví dụ: superuser. Nhưng trước tiên, chúng ta cần biết ID của người dùng đó.

Hãy xem ID của tất cả người dùng đã tạo cho đến nay. Bắt đầu phiên shell Django và đưa ra lệnh sau:

```
(ll_env)learning_log$ python manage.py shell
>>> from django.contrib.auth.models import User
```

```

>>> User.objects.all()
<QuerySet [<User: ll_admin>, <User: eric>, <User: willie>]>
>>> for user in User.objects.all():
...     print(user.username, user.id)
...
ll_admin 1
eric 2
willie 3
>>>

```

Đầu tiên, chúng ta import mô hình User vào phiên shell. Sau đó xem xét tất cả người dùng đã được tạo cho đến nay. Kết quả hiển thị ba người dùng: ll_admin, eric và willie.

Chúng ta duyệt qua danh sách người dùng và in tên người dùng cùng với ID của họ. Khi Django hỏi người dùng liên kết các chủ đề hiện có với, chúng ta sẽ sử dụng một trong các giá trị ID này.

Di chuyển cơ sở dữ liệu

Bây giờ chúng ta đã biết các ID nên ta có thể di chuyển cơ sở dữ liệu. Khi thực hiện việc này, Python sẽ yêu cầu chúng ta tạm thời kết nối mô hình Topics với một chủ sở hữu cụ thể hoặc thêm một tệp mặc định vào tệp models.py để cho nó biết phải làm gì. Chọn tùy chọn 1:

```

(ll_env)learning_logs$ python manage.py makemigrations learning_logs
You are trying to add a non-nullable field 'owner' to topic without a
default; we can't do that (the database needs something to populate
existing rows).

Please select a fix:
1) Provide a one-off default now (will be set on all existing rows with
a
    null value for this column)
2) Quit, and let me add a default in models.py

Select an option: 1

Please enter the default value now, as valid Python
The datetime and django.utils.timezone modules are available, so you can
do
e.g. timezone.now

Type 'exit' to exit this prompt
>>> 1

Migrations for 'learning_logs':
  learning_logs/migrations/0003_topic_owner.py
  - Add field owner to topic
(ll_env)learning_logs$

```

Chúng ta bắt đầu bằng cách phát hành lệnh makemigrations. Trong kết quả đầu ra , Django chỉ ra rằng chúng ta đang cố gắng thêm trường bắt buộc (không thể trống) vào mô hình hiện có (chủ đề) mà không có giá trị mặc định nào được chỉ định. Django cung cấp cho chúng ta hai tùy chọn: một là có thể cung cấp giá trị mặc định ngay bây giờ hoặc chúng ta có thể thoát và thêm giá trị mặc định trong models.py. Tiếp theo, chọn lựa chọn 1. Django sau đó yêu cầu chúng ta nhập giá trị mặc định.

Để liên kết tất cả các chủ đề hiện có với người dùng quản trị ban đầu, ll_admin, ta đã nhập ID người dùng là 1. Ta có thể sử dụng ID của bất kỳ người dùng nào mà ta đã tạo; nó không cần phải là một siêu người dùng. Sau đó, Django di chuyển cơ sở dữ liệu bằng cách sử dụng giá trị này và tạo tệp di chuyển 0003_topic_owner.py, tệp này thêm chủ sở hữu trường vào mô hình Topics.

Bây giờ chúng ta có thể thực hiện quá trình di chuyển. Nhập thông tin sau vào môi trường ảo đang hoạt động:

```
(ll_env) learning_log$ python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, learning_logs, sessions
Running migrations:
  Applying learning_logs.0003_topic_owner... OK
(ll_env) learning_log$
```

Chúng ta có thể xác minh rằng quá trình di chuyển hoạt động như mong đợi trong shell như sau:

```
>>> from learning_logs.models import Topic
>>> for topic in Topic.objects.all():
...     print(topic, topic.owner)
...
Chess ll_admin
Rock Climbing ll_admin
>>>
```

Chúng ta import Topic từ learning_logs.models , rồi lặp lại tất cả các chủ đề hiện có, in từng chủ đề và người dùng mà chủ đề đó thuộc về . Từ đó ta có thể thấy rằng mỗi chủ đề hiện thuộc về người dùng ll_admin.

Chú ý : Ta có thể chỉ cần đặt lại cơ sở dữ liệu thay vì di chuyển, nhưng điều đó sẽ làm mất tất cả dữ liệu hiện có. Chúng ta nên tìm hiểu cách di chuyển cơ sở dữ liệu mà vẫn duy trì tính toàn vẹn của dữ liệu của người dùng. Nếu muốn bắt đầu với

một cơ sở dữ liệu mới, hãy sử dụng lệnh python management.py flush để xây dựng lại cấu trúc cơ sở dữ liệu. Ta sẽ phải tạo một siêu người dùng mới và tất cả dữ liệu cũ sẽ biến mất.

c. Hạn chế quyền truy cập chủ đề đối với người dùng phù hợp

Hiện tại, nếu chúng ta đăng nhập, ta sẽ có thể xem tất cả các chủ đề, bất kể chúng ta đang đăng nhập với tư cách người dùng nào. Chúng ta sẽ thay đổi điều đó bằng cách chỉ hiển thị cho người dùng những chủ đề thuộc về họ. Thực hiện thay đổi sau đối với hàm topics() trong views.py:

```
--snip--  
@login_required  
def topics(request):  
    """Show all topics."""  
    topics=Topic.objects.filter(owner=request.user).order_by('date_added')  
    context = {'topics': topics}  
    return render(request, 'learning_logs/topics.html', context)  
--snip--
```

Khi người dùng đã đăng nhập, đối tượng yêu cầu có bộ thuộc tính request.user để lưu trữ thông tin về người dùng. Truy vấn Topic.objects.filter(owner = request.user) yêu cầu Django chỉ truy xuất các đối tượng Topics từ cơ sở dữ liệu có thuộc tính chủ sở hữu khớp với người dùng hiện tại. Bởi vì chúng ta không thay đổi gì về cách hiển thị chủ đề, không cần phải thay đổi mẫu cho trang chủ đề.

Để xem cách này có hiệu quả hay không, hãy đăng nhập với tư cách là người dùng mà chúng ta đã kết nối tất cả các chủ đề hiện có và truy cập trang chủ đề. Ta sẽ thấy tất cả các chủ đề. Nay giờ hãy đăng xuất và đăng nhập lại với tư cách là một người dùng khác. Trang chủ đề không được liệt kê chủ đề.

d. Bảo vệ chủ đề của người dùng

Chúng ta chưa hạn chế quyền truy cập vào các trang chủ đề, vì vậy bất kỳ người dùng đã đăng ký nào cũng có thể thử một loạt URL, như http://localhost: 8000 / themes / 1 / và truy xuất các trang chủ đề trùng khớp.

Trong khi đăng nhập với tư cách là người dùng sở hữu tất cả các chủ đề, hãy sao chép URL hoặc ghi chú ID trong URL của một chủ đề, sau đó đăng xuất và đăng nhập lại với tư cách là một người dùng khác. Nhập URL của chủ đề đó. Và ta có thể

đọc các mục nhập entry ngay cả khi chúng ta đang đăng nhập với tư cách là một người dùng khác.

Khắc phục sự cố này bằng cách thực hiện kiểm tra trước khi truy xuất các mục được yêu cầu trong hàm topic() trong views.py:

```
from django.shortcuts import render, redirect
from django.contrib.auth.decorators import login_required
from django.http import Http404
--snip--
@login_required
def topic(request, topic_id):
    """Show a single topic and all its entries."""
    topic = Topic.objects.get(id=topic_id)
    # Make sure the topic belongs to the current user.
    if topic.owner != request.user:
        raise Http404

    entries = topic.entry_set.order_by('-date_added')
    context = {'topic': topic, 'entries': entries}
    return render(request, 'learning_logs/topic.html', context)
--snip--
```

Phản hồi 404 là phản hồi lỗi được trả lại khi tài nguyên được yêu cầu không tồn tại trên máy chủ. Tại đây, chúng ta import một ngoại lệ Http404, để chỉ ra người dùng yêu cầu một chủ đề mà họ không nhìn thấy. Sau khi nhận được yêu cầu chủ đề, kiểm tra người dùng của chủ đề có khớp với người dùng hiện đang đăng nhập hay không. Nếu người dùng hiện tại không sở hữu chủ đề được yêu cầu hãy đưa ra ngoại lệ Http404 và Django trả về trang lỗi 404.

e. Bảo vệ trang edit_entry

Các trang edit_entry có URL ở dạng http://localhost: 8000 / edit_entry / entry_id /, trong đó entry_id là một số. Hãy bảo vệ trang này để không ai có thể sử dụng URL để truy cập vào các mục nhập của người khác:

```
--snip--
@login_required
def edit_entry(request, entry_id):
    """Edit an existing entry."""
    entry = Entry.objects.get(id=entry_id)
    topic = entry.topic
    if topic.owner != request.user:
        raise Http404
```

```
    if request.method != 'POST':  
        --snip--
```

f. Liên kết các chủ đề mới với người dùng hiện tại

Hiện tại, trang thêm chủ đề mới bị hỏng vì trang này không liên kết chủ đề mới với bất kỳ người dùng cụ thể nào. Django đang nói rằng ta không thể tạo chủ đề mới mà không chỉ định giá trị cho trường chủ sở hữu của chủ đề.

Có một cách khắc phục đơn giản cho sự cố này,bởi vì chúng ta có quyền truy cập vào người dùng hiện tại thông qua request. Thêm mã sau vào views.py, mã này liên kết chủ đề mới với người dùng hiện tại:

```
--snip--  
@login_required  
def new_topic(request):  
    """Add a new topic."""  
    if request.method != 'POST':  
        # No data submitted; create a blank form.  
        form = TopicForm()  
    else:  
        # POST data submitted; process data.  
        form = TopicForm(data=request.POST)  
        if form.is_valid():  
            new_topic = form.save(commit=False)  
            new_topic.owner = request.user  
            new_topic.save()  
            return redirect('learning_logs:topics')  
    # Display a blank or invalid form.  
    context = {'form': form}  
    return render(request, 'learning_logs/new_topic.html', context)  
--snip--
```

Khi gọi form.save(), chúng ta truyền đối số commit = False vì chúng ta cần sửa đổi chủ đề mới trước khi lưu nó vào cơ sở dữ liệu. Sau đó ta đặt thuộc tính chủ sở hữu của chủ đề mới cho người dùng hiện tại. Cuối cùng gọi save() trên phiên bản chủ đề vừa được xác định . Bây giờ chủ đề đã có tất cả dữ liệu cần thiết và sẽ lưu thành công.

Ta có thể thêm bao nhiêu chủ đề mới tùy thích cho bao nhiêu người dùng khác nhau tùy thích. Mỗi người dùng sẽ chỉ có quyền truy cập vào dữ liệu của riêng họ, cho dù họ đang xem dữ liệu, nhập dữ liệu mới hay sửa đổi dữ liệu cũ.

Kết mục

Trong mục này, chúng ta đã học cách sử dụng các biểu mẫu để cho phép người dùng thêm các chủ đề và mục nhập mới, đồng thời chỉnh sửa các mục nhập hiện có. Sau đó, ta đã học cách triển khai tài khoản người dùng. Chúng ta đã cho phép người dùng hiện tại đăng nhập và đăng xuất, đồng thời sử dụng UserCreationForm mặc định của Django để cho phép mọi người tạo tài khoản mới.

Sau khi xây dựng hệ thống đăng ký và xác thực người dùng đơn giản, chúng ta đã hạn chế quyền truy cập đối với người dùng đã đăng nhập đối với các trang nhất định bằng `@login_required` decorator. Sau đó phân bổ dữ liệu cho những người dùng cụ thể thông qua khóa ngoại. Ta cũng đã học cách di chuyển cơ sở dữ liệu khi quá trình di chuyển yêu cầu chỉ định một số dữ liệu mặc định.

Cuối cùng là học được cách đảm bảo người dùng chỉ có thể xem dữ liệu thuộc về họ bằng cách sửa đổi các chức năng xem. Chúng ta đã truy xuất dữ liệu thích hợp bằng phương thức `filter()` và so sánh chủ sở hữu của dữ liệu được yêu cầu với người dùng hiện đang đăng nhập. Không phải lúc nào ta cũng có thể rõ ràng ngay lập tức dữ liệu nào nên cung cấp và dữ liệu nào nên bảo vệ, nhưng kỹ năng này sẽ đi kèm với thực hành. Các quyết định mà chúng ta đã đưa ra trong chương này để bảo mật dữ liệu của người dùng cũng minh họa lý do tại sao làm việc với những người khác là một ý tưởng hay khi xây dựng dự án: nhờ người khác xem qua dự án khiến ta có nhiều khả năng phát hiện ra các khu vực dễ bị tấn công hơn.

Bây giờ chúng ta có một dự án hoạt động đầy đủ đang chạy trên máy cục bộ của mình. Trong chương cuối cùng, chúng ta sẽ thiết kế Nhật ký học tập để làm cho nó hấp dẫn hơn và sẽ triển khai dự án tới một máy chủ để bất kỳ ai có quyền truy cập internet đều có thể đăng ký và tạo tài khoản.

10.3.3. Định kiểu và triển khai ứng dụng

Nhật ký học tập hiện có đầy đủ chức năng, nhưng nó không có kiểu dáng và chỉ chạy trên máy cục bộ. Trong chương này, ta sẽ thiết kế dự án theo cách đơn giản nhưng chuyên nghiệp và sau đó triển khai nó lên một máy chủ trực tiếp để mọi người trên thế giới có thể tạo tài khoản và sử dụng nó.

Để tạo kiểu, chúng ta sẽ sử dụng thư viện Bootstrap - một bộ sưu tập các công cụ để tạo kiểu ứng dụng web để chúng trông chuyên nghiệp trên tất cả các thiết bị hiện đại, từ màn hình phẳng lớn đến điện thoại thông minh. Để làm điều này, chúng

ta sẽ sử dụng ứng dụng django-bootstrap4, ứng dụng này cũng sẽ giúp thực hành sử dụng các ứng dụng do các nhà phát triển Django khác tạo.

Triển khai Nhật ký học tập bằng Heroku, một trang web cho phép chúng ta đẩy dự án của mình lên một trong các máy chủ của nó, cung cấp cho bất kỳ ai có kết nối internet. Chúng ta cũng sẽ bắt đầu sử dụng hệ thống kiểm soát phiên bản có tên Git để theo dõi các thay đổi đối với dự án.

10.3.3.1. Định kiểu website

a. Ứng dụng django-bootstrap4

Chúng ta sẽ sử dụng django-bootstrap4 để tích hợp Bootstrap vào dự án của mình. Ứng dụng này tải xuống các tệp Bootstrap cần thiết, đặt chúng vào một vị trí thích hợp trong dự án và cung cấp các chỉ thị tạo kiểu trong các mẫu của dự án.

Để cài đặt django-bootstrap4, hãy cài đặt lệnh sau trong môi trường ảo :

```
(ll_env) learning_log$ pip install django-bootstrap4  
--snip--  
Successfully installed django-bootstrap4-0.0.7
```

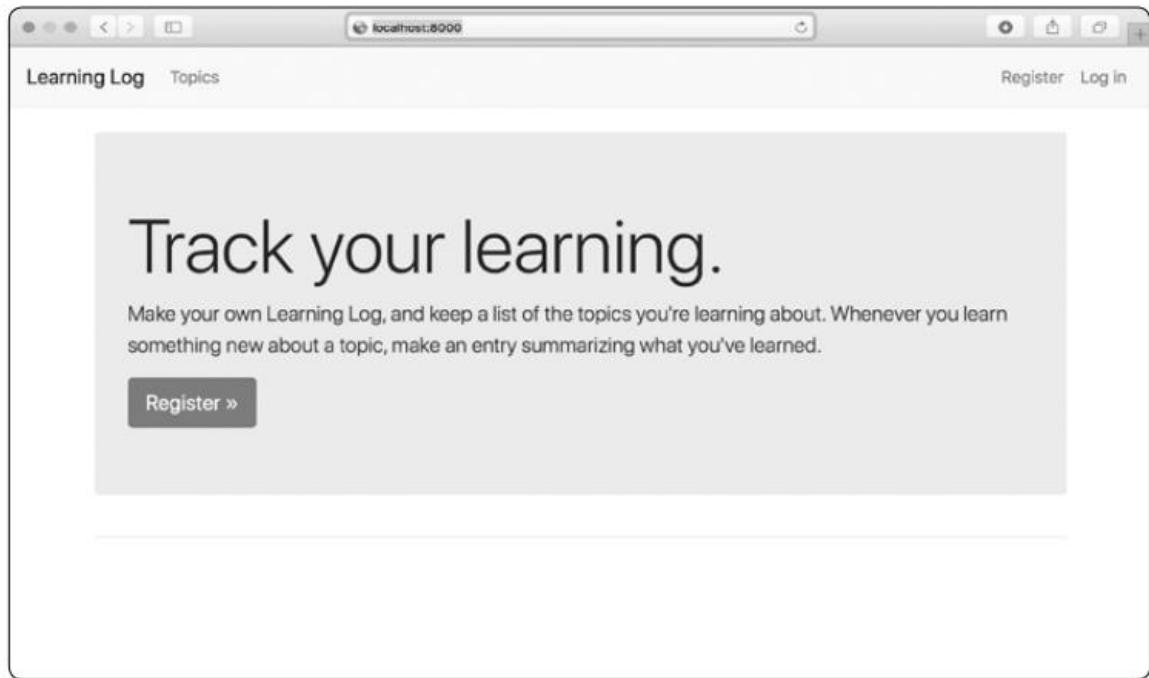
Tiếp theo, chúng ta cần cài đặt django-bootstrap4 trong INSTALLED_APPS trong settings.py:

```
--snip--  
INSTALLED_APPS = [  
    # My apps.  
    'learning_logs',  
    'users',  
    # Third party apps.  
    'bootstrap4',  
    # Default django apps.  
    'django.contrib.admin',  
--snip--
```

b. Sử dụng Bootstrap để định kiểu cho Nhật ký học tập

Bootstrap là một bộ sưu tập lớn các công cụ tạo kiểu. Nó cũng có một số mẫu mà ta có thể áp dụng cho dự án của mình để tạo ra một phong cách tổng thể. Sử dụng các mẫu này dễ dàng hơn nhiều so với việc sử dụng các công cụ tạo kiểu riêng lẻ. Để xem các mẫu mà Bootstrap cung cấp, hãy truy cập <https://getbootstrap.com/>, nhấp vào Ví dụ và tìm phần Navbars. Chúng ta sẽ sử dụng mẫu Navbar static, mẫu này cung cấp một thanh điều hướng đơn giản trên cùng và một vùng chứa cho nội dung của trang.

Hình sau đây cho thấy trang chủ sẽ trông như thế nào sau khi chúng ta áp dụng mẫu của Bootstrap cho base.html và sửa đổi một chút index.html.



Hình 10.46. Áp dụng mẫu Bootstrap

c. Sửa đổi base.html

Chúng ta cần sửa đổi mẫu base.html để phù hợp với mẫu Bootstrap.

Xác định các tiêu đề HTML

Thay đổi đầu tiên mà chúng ta sẽ thực hiện đối với base.html là xác định các tiêu đề HTML trong tệp, vì vậy khi trang Nhật ký Học tập được mở, thanh tiêu đề của trình duyệt sẽ hiển thị tên trang web. Chúng ta cũng sẽ thêm một số yêu cầu để sử dụng Bootstrap trong các mẫu của mình. Xóa mọi thứ trong base.html và thay thế bằng mã sau:

```
{% load bootstrap4 %}  
<!doctype html>  
<html lang="en">  
<head>  
    <meta charset="utf-8">  
    <meta name="viewport" content="width=device-width, initial-scale=1,  
        shrink-to-fit=no">  
<title>Learning Log</title>  
{% bootstrap_css %}  
{% bootstrap_javascript jquery='full' %}  
</head>
```

Đầu tiên, chúng ta tải bộ sưu tập các thẻ mẫu có sẵn trong django-bootstrap4. Tiếp theo, chúng ta khai báo tệp này là một tài liệu HTML được viết bằng tiếng Anh. Tệp HTML được chia thành hai phần chính, phần đầu và phần thân. Phần đầu của tệp HTML không chứa bất kỳ nội dung nào: nó chỉ cho trình duyệt biết những gì nó cần biết để hiển thị trang một cách chính xác. <title>” ”</title>, là tiêu đề cho trang, phần tử này sẽ hiển thị trên thanh tiêu đề của trình duyệt bất cứ khi nào Nhật ký học tập được mở.

Cuối cùng chúng ta sử dụng một trong các thẻ mẫu tùy chỉnh của django-bootstrap4, thẻ này yêu cầu Django bao gồm tất cả các tệp kiểu Bootstrap. Thẻ này cho phép tất cả các hành vi tương tác mà ta có thể sử dụng trên một trang, chẳng hạn như các thanh điều hướng có thể thu gọn.

Xác định Thanh điều hướng

Mã xác định thanh điều hướng ở đầu trang khá dài, vì nó phải hoạt động tốt trên màn hình điện thoại hẹp và màn hình máy tính để bàn rộng. Chúng ta sẽ làm việc thông qua thanh điều hướng trong các phần. Đây là phần đầu tiên của thanh điều hướng:

```
--snip--  
</head>  
<body>  
    <nav class="navbar navbar-expand-md navbar-light bg-light mb-4 border">  
        <a class="navbar-brand" href="{% url 'learning_logs:index' %}">  
            Learning Log</a>  
        <button class="navbar-toggler" type="button" data-toggle="collapse"  
            data-target="#navbarCollapse" aria-controls="navbarCollapse" aria-  
            expanded="false" aria-label="Toggle navigation">  
            <span class="navbar-toggler-icon"></span></button>
```

Phần tử đầu tiên là thẻ mở <body>. Phần nội dung của tệp HTML chứa nội dung mà người dùng sẽ thấy trên một trang. Phần tử <nav> cho biết phần liên kết điều hướng của trang. Mọi thứ chưa trong phần tử này được tạo kiểu theo các quy tắc kiểu Bootstrap được xác định bởi các bộ chọn navbar, navbar-expand-md và phần còn lại mà ta thấy ở đây. Bộ chọn navbar-light và bg-light tạo kiểu cho thanh điều hướng với nền có tông màu sáng. Mb trong mb-4 là viết tắt của margin-bottom; bộ chọn này đảm bảo rằng một ít khoảng trống xuất hiện giữa thanh điều hướng và phần

còn lại của trang. Bộ chọn đường viền cung cấp một đường viền mỏng xung quanh nền sáng để đặt nó lệch một chút so với phần còn lại của trang.

Chúng ta đặt tên của dự án xuất hiện ở ngoài cùng bên trái của thanh điều hướng và đặt nó thành một liên kết đến trang chủ; nó sẽ xuất hiện trên mọi trang trong dự án. Công cụ chọn navar-brand tạo kiểu cho liên kết này để nó nổi bật so với các liên kết còn lại và là một cách xây dựng thương hiệu cho trang web.

Dùng button xuất hiện nếu cửa sổ trình duyệt quá hẹp để hiển thị toàn bộ thanh điều hướng theo chiều ngang. Khi người dùng nhấp vào nút, các yếu tố điều hướng sẽ xuất hiện theo danh sách. Tham chiếu collapse làm cho thanh điều hướng thu gọn khi người dùng thu nhỏ cửa sổ trình duyệt hoặc khi trang web được hiển thị trên thiết bị di động có màn hình nhỏ.

Đây là phần mã tiếp theo xác định thanh điều hướng:

```
--snip--  
<span class="navbar-toggler-icon"></span></button>  
<div class="collapse navbar-collapse" id="navbarCollapse">  
    <ul class="navbar-nav mr-auto">  
        <li class="nav-item">  
            <a class="nav-link" href="{% url 'learning_logs:topics' %}">  
                Topics</a></li>  
    </ul>
```

Thuật ngữ div là viết tắt của phép chia; giúp ta xây dựng một trang web bằng cách chia nó thành các phần và xác định các quy tắc về phong cách và hành vi áp dụng cho phần đó. Bất kỳ quy tắc kiểu hoặc hành vi nào được xác định trong thẻ mở div sẽ ảnh hưởng đến mọi thứ chúng ta thấy cho đến thẻ div đóng tiếp theo, được viết là </div>. Đây là phần bắt đầu của phần thanh điều hướng sẽ được thu gọn trên màn hình và cửa sổ hẹp.

Dưới đây là phần kế tiếp của thanh điều hướng:

```
--snip--  
</ul>  
<ul class="navbar-nav ml-auto">  
    {% if user.is_authenticated %}  
        <li class="nav-item">  
            <span class="navbar-text">Hello, {{ user.username }}.</span>  
        </li>  
        <li class="nav-item">  
            <a class="nav-link" href="{% url 'users:logout' %}">Log out</a>  
        </li>
```

```

{ % else %}
    <li class="nav-item">
        <a class="nav-link" href="{% url 'users:register' %}">Register</a>
    </li>
    <li class="nav-item">
        <a class="nav-link" href="{% url 'users:login' %}">Log in</a>
    </li>
{ % endif %}
</ul>
</div>
</nav>

```

Chúng ta bắt đầu một tập hợp các liên kết mới bằng cách sử dụng một thẻ mở `` khác. Có thể có nhiều nhóm liên kết mà ta cần trên một trang. Đây sẽ là nhóm các liên kết liên quan đến đăng nhập và đăng ký xuất hiện ở phía bên phải của thanh điều hướng. Bộ chọn `ml-auto` là viết tắt của tự động đặt lề trái.

Khối `if { % if user.is_authenticated %}` là khối có điều kiện mà chúng ta đã sử dụng trước đó để hiển thị thông báo thích hợp cho người dùng tùy thuộc vào việc họ có đăng nhập hay không. Khối này hiện lâu hơn một chút vì một số quy tắc tạo kiểu nằm bên trong thẻ điều kiện. Phần tử `` tạo kiểu cho các phần văn bản hoặc các phần tử của trang, là một phần của một dòng dài hơn. Trong khi các phần tử `div` tạo sự phân chia của riêng chúng trong một trang, các phần tử `span` liên tục trong một phần lớn hơn. Điều này có thể gây nhầm lẫn lúc đầu, bởi vì nhiều trang có các phần tử `div` được lồng sâu vào nhau. Ở đây, chúng ta đang sử dụng phần tử `span` để tạo kiểu cho văn bản thông tin trên thanh điều hướng, chẳng hạn như tên người dùng đã đăng nhập.

Chúng ta đóng phần tử `div` chứa các phần của thanh điều hướng và ở cuối phần này, đóng toàn bộ thanh điều hướng `</nav>`. Nếu muốn thêm nhiều liên kết hơn vào thanh điều hướng, hãy thêm một mục `` khác vào bất kỳ nhóm nào trong số nhóm `` mà chúng ta đã xác định trong thanh điều hướng bằng cách sử dụng các lệnh tạo kiểu giống hệt như những gì ta đã thấy ở đây.

Bước tiếp theo, Chúng ta cần xác định hai khối mà các trang riêng lẻ có thể sử dụng để đặt nội dung cụ thể cho các trang đó.

Xác định các phần chính của trang:

--snip--

```

</nav>
<main role="main" class="container">
    <div class="pb-2 mb-2 border-bottom">
        {% block page_header %}{% endblock page_header %}
    </div>
    <div>
        {% block content %}{% endblock content %}
    </div>
</main>
</body>
</html>

```

Mở thẻ `<main>`. Phần tử main được sử dụng cho phần quan trọng nhất của nội dung trang. Ở đây chúng ta gán bộ chọn bootstrap container, đây là một cách đơn giản để nhóm các phần tử trên một trang.

Phần tử div đầu tiên chứa một khối `page_header`. Chúng ta sẽ sử dụng khối này để đặt tiêu đề cho hầu hết các trang. Để làm cho phần này nổi bật so với phần còn lại của trang, chúng ta đặt một số phần đệm bên dưới tiêu đề. Phần đệm đề cập đến khoảng cách giữa nội dung của một phần tử và đường viền của phần tử đó. Bộ chọn `pb-2` là một chỉ thị bootstrap cung cấp một lượng đệm vừa phải ở dưới cùng của phần tử được tạo kiểu. Lẽ là khoảng cách giữa đường viền của phần tử và các phần tử khác trên trang. Muốn có một đường viền ở cuối trang, chúng ta nên sử dụng bộ chọn `boder-bottom`, cung cấp một đường viền mỏng ở cuối khối `page_header`.

Tiếp đó, chúng ta xác định thêm một phần tử div chứa nội dung khối. Không áp dụng bất kỳ kiểu cụ thể nào cho khối này, ta có thể tạo kiểu cho nội dung của bất kỳ trang nào khi chúng ta thấy phù hợp với trang đó. Kết thúc tệp `base.html` bằng các thẻ đóng cho các phần tử `main`, `body` và `html`.

d. Tạo kiểu cho Trang chủ bằng Jumbotron

Để cập nhật trang chủ, chúng ta sẽ sử dụng một phần tử của Bootstrap được gọi là jumbotron, là một hộp lớn nổi bật so với phần còn lại của trang và có thể chứa bất kỳ thứ gì ta muốn. Thông thường, nó được sử dụng trên các trang chủ để mô tả ngắn gọn về dự án tổng thể và lời kêu gọi hành động mời người xem tham gia.

```

{% extends "learning_logs/base.html" %}
{% block page_header %}
<div class="jumbotron">
    <h1 class="display-3">Track your learning.</h1>

```

```

<p class="lead">Make your own Learning Log, and keep a list of the
topics you're learning about. Whenever you learn something new about a
topic, make an entry summarizing what you've learned.</p>
<a class="btn btn-lg btn-primary" href="{% url 'users:register' %}"
role="button">Register &raquo;</a>
</div>
{% endblock page_header %}

```

block page_header nói với Django rằng chúng ta muốn xác định những gì diễn ra trong khái page_header. Một jumbotron chỉ là một phần tử div với một tập hợp các lệnh tạo kiểu được áp dụng cho nó. Bộ chọn jumbotron áp dụng nhóm các lệnh tạo kiểu này từ thư viện Bootstrap cho phần tử này.

Bên trong jumbotron có ba phần tử. Đầu tiên là một thông điệp ngắn gọn, Theo dõi quá trình học tập của ta, mang đến cho những khách truy cập lần đầu tiên biết về những gì Nhật ký học tập làm được. Lớp h1 là tiêu đề cấp đầu tiên và bộ chọn hiển thị-3 tạo thêm giao diện mỏng hơn và cao hơn cho tiêu đề cụ thể này. Tiếp theo, chúng ta đưa ra một thông báo dài hơn cung cấp thêm thông tin về những gì người dùng có thể làm với nhật ký học tập của họ.

Thay vì chỉ sử dụng một liên kết văn bản, chúng ta cần tạo một nút mời người dùng đăng ký tài khoản Nhật ký Học tập của họ. Đây là liên kết tương tự như trong tiêu đề, nhưng nút nỗi bật trên trang và cho người xem biết họ cần làm gì để bắt đầu sử dụng dự án. Mã & raquo; là một thực thể html trông giống như hai dấu ngoặc vuông được kết hợp với nhau (>>).

e. Định kiểu cho trang login:

Chúng ta đã chỉnh sửa giao diện tổng thể của trang đăng nhập nhưng chưa có biểu mẫu đăng nhập. Hãy làm cho biểu mẫu trông nhất quán với phần còn lại của trang bằng cách sửa đổi tệp login.html:

```

{% extends "learning_logs/base.html" %}
{% load bootstrap4 %}
{% block page_header %}
    <h2>Log in to your account.</h2>
{% endblock page_header %}
{% block content %}
<form method="post" action="{% url 'users:login' %}" class="form">
    {% csrf_token %}
    {% bootstrap_form form %}
    {% buttons %}

```

```

    <button name="submit" class="btn btn-primary">Log
    in</button>
    {%- endbuttons %}

<input type="hidden" name="next"
value="{% url 'learning_logs:index' %}" />
</form>

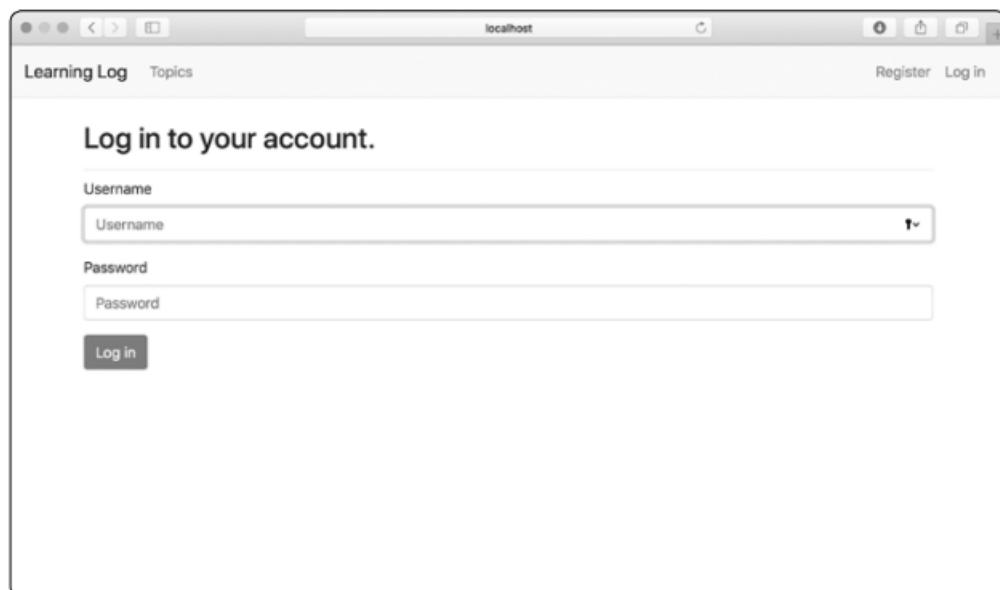
{%- endblock content %}

```

Trước hết, chúng ta tải các thẻ mẫu bootstrap4 vào mẫu này sau đó định nghĩa khói page_header, khói này cho người dùng biết trang đó dùng để làm gì. Lưu ý rằng chúng ta đã xóa khói {% if form.errors%} khỏi mẫu; django-bootstrap4 tự động quản lý các lỗi biểu mẫu.

Chúng ta thêm thuộc tính class = "form" và sử dụng thẻ mẫu {% bootstrap_form%} để hiển thị biểu mẫu ; thẻ này thay thế thẻ {{form.as_p}} mà chúng ta đã sử dụng trong phần trước. Thẻ mẫu {% bootstrap_form%} chèn các quy tắc kiểu Bootstrap vào các phần tử riêng lẻ của biểu mẫu khi biểu mẫu được hiển thị. Mở thẻ mẫu bootstrap4 {% button%}, thẻ này thêm các nút.

Hình sau đây cho thấy biểu mẫu đăng nhập ngay bây giờ. Trang đẹp hơn nhiều và có kiểu dáng nhất quán và mục đích rõ ràng. Thủ đăng nhập bằng tên người dùng hoặc mật khẩu không chính xác; ta sẽ thấy rằng ngay cả các thông báo lỗi cũng được tạo kiểu nhất quán và tích hợp tốt với trang web tổng thể.



Hình 10.47. Định kiểu trang đăng nhập bằng Bootstrap

f. Định kiểu trang chủ đề Topics:

Trong file topics.html, hãy sửa đổi :

```
{% extends "learning_logs/base.html" %}  
{% block page_header %}  
    <h1>Topics</h1>  
{% endblock page_header %}  
{% block content %}  
    <ul>  
        {% for topic in topics %}  
            <li><h3>  
                <a href="{% url 'learning_logs:topic' topic.id %}">{{ topic }}</a>  
                </h3></li>  
            {% empty %}  
            <li><h3>No topics have been added yet.</h3></li>  
        {% endfor %}  
    </ul>  
    <h3><a href="{% url 'learning_logs:new_topic' %}">Add a new  
topic</a></h3>  
    {% endblock content %}
```

Chúng ta không cần thẻ {% load bootstrap4%} vì ta không cần sử dụng bất kỳ thẻ mẫu bootstrap4 tùy chỉnh nào trong tệp này. Di chuyển tiêu đề Topics vào khỏi page_header và tạo cho nó một kiểu đầu trang thay vì sử dụng thẻ đoạn đơn giản. Hãy đặt mỗi chủ đề dưới dạng phần tử <h3> để làm cho chúng lớn hơn một chút trên trang và làm tương tự đối với liên kết để thêm chủ đề mới.

g. Định kiểu cho các mục nhập Entry trong Trang Topics:

Trang chủ đề có nhiều nội dung hơn hầu hết các trang, vì vậy nó cần phải làm việc nhiều hơn một chút. Chúng ta sẽ sử dụng thành phần thẻ Bootstrap để làm cho mỗi mục nhập trở nên nổi bật. Thẻ là một div với một tập hợp các kiểu linh hoạt, được xác định trước, hoàn hảo để hiển thị các mục nhập của chủ đề:

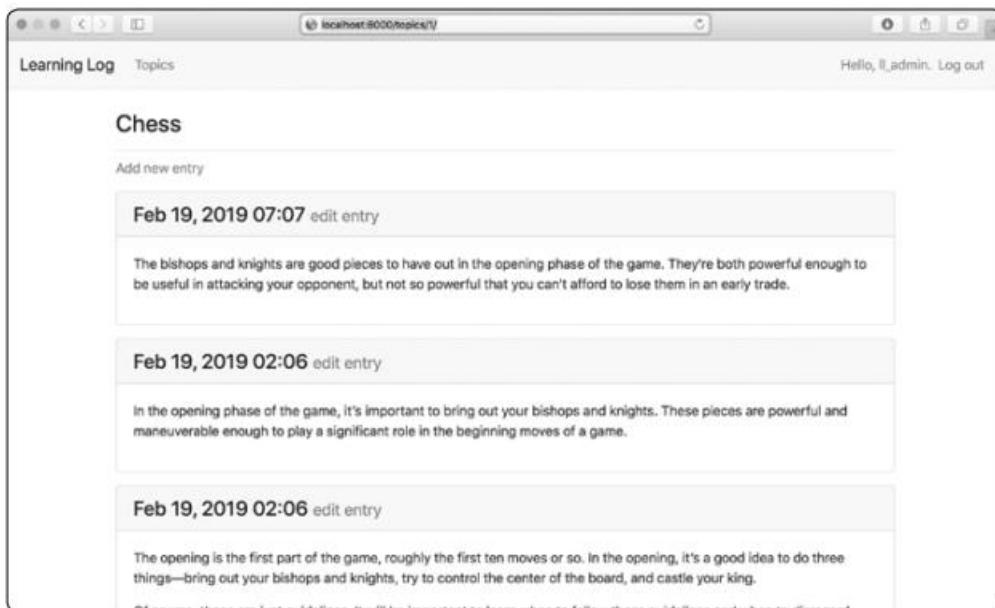
```
{% extends 'learning_logs/base.html' %}  
    {% block page_header %}  
        <h3>{{ topic }}</h3>  
    {% endblock page_header %}  
    {% block content %}  
        <p>  
            <a href="{% url 'learning_logs:new_entry' topic.id %}">Add new  
entry</a>  
        </p>  
        {% for entry in entries %}
```

```

<div class="card mb-3">
    <h4 class="card-header">
        {{ entry.date_added|date:'M d, Y H:i' }}
        <small><a href="{% url 'learning_logs:edit_entry' entry.id %}"> edit entry</a></small>
    </h4>
    <div class="card-body">
        {{ entry.text|linebreaks }}
    </div>
</div>
{%
empty %}
<p>There are no entries for this topic yet.</p>
{%
endfor %}
{%
endblock content %}

```

Đầu tiên, chúng ta đặt chủ đề trong khôi page_header. Sau đó xóa cấu trúc danh sách không có thứ tự được sử dụng trước đó trong mẫu này. Thay vì đặt mỗi mục nhập thành một mục danh sách, chúng ta tạo một phần tử div bằng thẻ bộ chọn card mb-3. Thẻ này có hai phần tử lồng nhau: một phần tử để đánh dấu mốc thời gian đồng thời có liên kết đến mục chỉnh sửa mục nhập “edit entry” và một phần tử khác để chứa phần nội dung của mục nhập .



Hình 10.48. Trang chủ để với định kiểu Bootstrap

Chú ý : Nếu chúng ta muốn sử dụng một mẫu Bootstrap khác, hãy làm theo quy trình tương tự như những gì đã thực hiện phía trên trong phần này. Sao chép mẫu ta muốn sử dụng vào base.html và sửa đổi các phần tử chứa nội dung thực tế để mẫu

hiển thị thông tin dự án. Sau đó, sử dụng các công cụ tạo kiểu riêng của Bootstrap để tạo kiểu cho nội dung trên mỗi trang.

10.3.3.2. Triển khai website

Bây giờ chúng ta có một dự án trông chuyên nghiệp hơn, hãy triển khai dự án đó lên một máy chủ trực tiếp để bất kỳ ai có kết nối Internet đều có thể sử dụng nó. Chúng ta sẽ sử dụng Heroku, một nền tảng dựa trên web cho phép tự chúng ta quản lý việc triển khai các ứng dụng web. Hãy đăng ký trang Nhật ký Học tập và chạy trên Heroku.

Tạo tài khoản heroku:

Để tạo tài khoản, hãy truy cập <https://heroku.com/> và chọn một trong các liên kết đăng ký. Heroku có một cấp miễn phí cho phép chúng ta kiểm tra các dự án của mình trong quá trình triển khai trực tiếp trước khi triển khai đúng cách.

Cài đặt Heroku CLI

Để triển khai và quản lý dự án trên máy chủ của Heroku, chúng ta sẽ cần các công cụ có sẵn trong Giao diện dòng lệnh Heroku (CLI). Để cài đặt phiên bản mới nhất của Heroku CLI, hãy truy cập <https://devcenter.heroku.com/articles/heroku-cli/> và làm theo hướng dẫn cho hệ điều hành.

Cài đặt các gói bắt buộc

Ta cũng sẽ cần cài đặt ba gói giúp phân tán các dự án Django trên một máy chủ trực tiếp. Trong một môi trường ảo đang hoạt động, hãy thực hiện các lệnh sau:

```
(11_env)learning_log$ pip install psycopg2==2.7.*  
(11_env)learning_log$ pip install django-heroku  
(11_env)learning_log$ pip install gunicorn
```

Gói psycopg2 được dùng để quản lý cơ sở dữ liệu mà Heroku sử dụng. Gói django-heroku xử lý gần như toàn bộ cấu hình mà ứng dụng của chúng ta cần để chạy đúng cách trên máy chủ Heroku. Điều này bao gồm quản lý cơ sở dữ liệu và lưu trữ các tệp tĩnh ở một nơi mà chúng có thể được phân phát đúng cách. Gói gunicorn cung cấp một máy chủ có khả năng cung cấp các ứng dụng trong môi trường trực tiếp.

Tạo tệp requirements.txt

Heroku cần biết dự án của chúng ta thuộc vào gói nào, vì vậy hãy sử dụng pip để tạo tệp liệt kê chúng. Một lần nữa, từ một môi trường ảo đang hoạt động, hãy đưa ra lệnh sau:

```
(11_env)learning_log$ pip freeze > requirements.txt
```

Lệnh freeze yêu cầu pip ghi tên tất cả các gói hiện được cài đặt trong dự án vào tệp tin requirements.txt. Mở tệp này để xem các gói và số phiên bản được cài đặt trong dự án :

```
dj-database-url==0.5.0
Django==2.2.0
django-bootstrap4==0.0.7
django-heroku==0.3.1
gunicorn==19.9.0
psycopg2==2.7.7
pytz==2018.9
sqlparse==0.2.4
whitenoise==4.1.2
```

Khi chúng ta triển khai Nhật ký học tập, Heroku sẽ cài đặt tất cả các gói được liệt kê trong tệp requirement.txt, tạo ra một môi trường với các gói tương tự mà chúng ta đang sử dụng cục bộ. Vì lý do này ta có thể tin tưởng rằng chương trình triển khai sẽ hoạt động giống như trên hệ thống cục bộ. Đây là một lợi thế rất lớn khi chúng ta bắt đầu xây dựng và duy trì các dự án khác nhau trên hệ thống của mình.

Chú ý: Nếu một gói được liệt kê trên hệ thống của ta nhưng số phiên bản khác với số được hiển thị ở trên thì hãy giữ phiên bản mà ta có trên hệ thống.

Chỉ định thời gian chạy Python

Trừ khi chúng ta chỉ định phiên bản Python, Heroku sẽ sử dụng phiên bản Python mặc định hiện tại của nó. Hãy đảm bảo rằng Heroku sử dụng cùng một phiên bản Python mà chúng ta đang sử dụng. Trong một môi trường ảo đang hoạt động, hãy sử dụng lệnh python --version:

```
(11_env)learning_log$ python --version
Python 3.7.2
```

Trong ví dụ này, chúng ta đang chạy Python phiên bản 3.7.2. Tạo một tệp mới có tên runtime.txt trong cùng thư mục với management.py và nhập thông tin sau:

```
Python-3.7.2
```

Chú ý : Nếu chúng ta gặp lỗi báo cáo rằng thời gian chạy Python mà ta yêu cầu không khả dụng, hãy truy cập <https://devcenter.heroku.com/categories/language-support/> và tìm liên kết để Chỉ định thời gian chạy Python. Xem qua bài viết để tìm các thời gian chạy có sẵn và sử dụng thời gian phù hợp nhất với phiên bản Python .

Sửa đổi settings.py cho Heroku

Bây giờ chúng ta cần thêm một phần ở cuối settings.py để xác định một số cài đặt cụ thể cho môi trường Heroku:

```
--snip--  
# My settings  
LOGIN_URL = 'users:login'  
# Heroku settings.  
import django_heroku  
django_heroku.settings(locals())
```

Tạo một Procfile để bắt đầu quá trình

Một Procfile cho Heroku biết quy trình nào sẽ bắt đầu để phục vụ đúng dự án. Lưu tệp sau dưới dạng Procfile, với chữ P viết hoa và không có phần mở rộng tệp, trong cùng thư mục với management.py. Dưới đây là dòng trong Procfile:

```
web: gunicorn learning_log.wsgi --log-file -
```

Dòng này yêu cầu Heroku sử dụng gunicorn làm máy chủ và sử dụng cài đặt trong learning_log / wsgi.py để khởi chạy ứng dụng. Cờ log-file cho Heroku biết các loại sự kiện cần ghi.

Sử dụng Git để theo dõi các tệp của dự án

Git là một chương trình kiểm soát phiên bản cho phép chúng ta nhanh chóng có được mã trong dự án của mình mỗi khi chúng ta triển khai thành công một tính năng mới. Nếu có gì sai, ta có thể dễ dàng quay lại ảnh chụp nhanh hoạt động cuối cùng của dự án của mình; ví dụ: nếu ta vô tình gây ra một lỗi khi đang làm việc trên một tính năng mới. Mỗi ảnh chụp nhanh được gọi là một cam kết.

Sử dụng Git, ta có thể thử triển khai các tính năng mới mà không lo bị phá vỡ dự án của mình. Khi triển khai tới một máy chủ trực tiếp, chúng ta cần đảm bảo rằng mình đang triển khai một phiên bản hoạt động.

Cài đặt Git:

```
(11_env)learning_log$ git --version  
git version 2.17.0
```

Cấu hình Git:

Git theo dõi những người thực hiện thay đổi đối với một dự án, ngay cả khi chỉ có một người đang làm việc trong dự án. Để làm điều này, Git cần biết tên người dùng và email của người dùng Git.

```
(11_env)learning_log$ git config --global user.name "ehmatthes"
```

```
(11_env)learning_log$ git config --global user.email eric@example.com
```

BỎ QUA TỆP:

Chúng ta không cần Git theo dõi mọi tệp trong dự án, vì vậy hãy yêu cầu Git bỏ qua một số tệp. Tạo một tệp có tên `.gitignore` trong thư mục cùng chứa `management.py`. Lưu ý rằng tên tệp này bắt đầu bằng dấu chấm và không có phần mở rộng tệp. Đây là mã có trong `.gitignore`:

```
11_env/  
__pycache__/  
*.sqlite3
```

Chúng ta yêu cầu Git bỏ qua toàn bộ thư mục `11_env`, vì chúng ta có thể tạo lại nó tự động bất kỳ lúc nào. Và ta cũng không theo dõi thư mục `__pycache__` chứa các tệp `.pyc` được tạo tự động khi Django chạy các tệp `.py`. Đồng thời không theo dõi các thay đổi đối với cơ sở dữ liệu cục bộ vì đó là một thói quen xấu: nếu ta đang sử dụng SQLite trên máy chủ, ta có thể vô tình ghi đè cơ sở dữ liệu trực tiếp bằng cơ sở dữ liệu thử nghiệm cục bộ của mình khi đẩy dự án lên máy chủ. Dấu hoa thị trong `*.sqlite3` cho Git biết bỏ qua bất kỳ tệp nào kết thúc bằng phần mở rộng `.sqlite3`.

LÀM CHO CÁC TỆP ẨN ĐƯỢC HIỂN THỊ:

Hầu hết các hệ điều hành ẩn các tệp và thư mục bắt đầu bằng dấu chấm, chẳng hạn như `.gitignore`. Khi mở trình duyệt tệp hoặc có gắng mở tệp từ một tệp của ứng dụng, chẳng hạn như Sublime Text, chúng ta sẽ không thấy các loại tệp này theo mặc định. Dưới đây là cách xem các tệp ẩn, tùy thuộc vào hệ điều hành :

- Trên Windows, mở Windows Explorer, sau đó mở một thư mục Desktop. Nhập vào tab View và đảm bảo rằng File name extension và Hidden items được chọn.
- Trên macOS, Ta có thể nhấn `⌘-shift-.` (dấu chấm) trong bất kỳ cửa sổ trình duyệt tệp nào để xem các tệp và thư mục ẩn.
- Trên các hệ thống Linux như Ubuntu, ta có thể nhấn `ctrl-H` trong bất kỳ trình duyệt tệp nào để hiển thị các tệp và thư mục ẩn. Để đặt cài đặt này vĩnh viễn, hãy mở trình duyệt tệp như Nautilus và nhập vào tab tùy chọn (được biểu thị bằng ba dòng). Chọn hộp Show Hidden Files.

CAM KẾT DỰ ÁN:

Chúng ta cần khởi tạo kho lưu trữ Git cho Nhật ký học tập, thêm tất cả các tệp cần thiết vào kho lưu trữ và cam kết trạng thái ban đầu của dự án. Đây là cách thực hiện điều đó:

```
(11_env)learning_log$ git init
Initialized empty Git repository in /home/ehmatthes/pcc/learning_log/.git/
(11_env)learning_log$ git add .
(11_env)learning_log$ git commit -am "Ready for deployment to heroku."
[master (root-commit) 79fef72] Ready for deployment to heroku.
 45 files changed, 712 insertions(+)
 create mode 100644 .gitignore
 create mode 100644 Procfile
--snip--
 create mode 100644 users/views.py
(11_env)learning_log$ git status
On branch master
nothing to commit, working tree clean
(11_env)learning_log$
```

Chúng ta phát hành lệnh **git init** để khởi tạo một kho lưu trữ trống trong thư mục chứa Nhật ký học tập. Sau đó sử dụng **git add**. lệnh này sẽ thêm tất cả các tệp không bị bỏ qua vào kho lưu trữ. (Đừng quên dấu chấm.) Chúng ta đưa ra lệnh **git commit -am** cam kết thông báo: cờ -a yêu cầu Git bao gồm tất cả các tệp đã thay đổi trong cam kết này và cờ -m yêu cầu Git ghi lại một nhật ký thông báo. Đưa ra lệnh git trạng thái cho chúng ta biết rằng mình đang ở nhánh master . Đây là trạng thái ta sẽ muốn thấy bất cứ khi nào ta đẩy dự án của mình lên Heroku.

Đẩy dự án lên Heroku:

Cuối cùng thì chúng ta cũng sẵn sàng để đẩy dự án lên Heroku. Trong một môi trường ảo đang hoạt động, hãy phát hành các lệnh sau:

```
(11_env)learning_log$ heroku login
heroku: Press any key to open up the browser to login or q to exit:
Logging in... done
Logged in as eric@example.com
(11_env)learning_log$ heroku create
Creating app... done, ã secret-lowlands-82594
https://secret-lowlands-82594.herokuapp.com/ |
https://git.heroku.com/secret-lowlands-82594.git
(11_env)learning_log$ git push heroku master
--snip--
remote: -----> Launching...
```

```
remote: Released v5
remote: https://secret-lowlands-82594.herokuapp.com/ deployed to Heroku
remote: Verifying deploy... done.
To https://git.heroku.com/secret-lowlands-82594.git
 * [new branch] master -> master
(11_env)learning_log$
```

Đầu tiên hãy đưa ra lệnh để đăng nhập heroku, lệnh này sẽ đưa chúng ta đến một trang trong trình duyệt nơi mà ta có thể đăng nhập vào tài khoản Heroku của mình. Sau đó hãy yêu cầu Heroku xây dựng một dự án trống. Heroku tạo ra một cái tên được tạo thành từ hai từ và một số tacó thể thay đổi tên này. Tiếp theo, chúng ta phát hành lệnh **git push heroku master**, lệnh này yêu cầu Git đẩy nhánh chính của dự án vào kho lưu trữ mà Heroku vừa tạo. Sau đó, Heroku xây dựng dự án trên các máy chủ của mình bằng cách sử dụng các tệp này.

Khi chúng ta đưa ra các lệnh này, dự án sẽ được triển khai nhưng chưa được định cấu hình đầy đủ. Để kiểm tra xem quá trình máy chủ đã bắt đầu đúng chưa, hãy sử dụng lệnh **heroku ps**:

```
(11_env)learning_log$ heroku ps
u Free dyno hours quota remaining this month: 450h 44m (81%)
Free dyno usage for this app: 0h 0m (0%)
For more information on dyno sleeping and how to upgrade, see:
https://devcenter.heroku.com/articles/dyno-sleeping
v === web (Free): gunicorn learning_log.wsgi --log-file - (1)
web.1: up 2019/02/19 23:40:12 -0900 (~ 10m ago)
(11_env)learning_log$
```

Bây giờ chúng ta có thể mở ứng dụng trong trình duyệt bằng lệnh heroku open:

```
(11_env)learning_log$ heroku open
(11_env)learning_log$
```

Lệnh này ngăn ta mở trình duyệt và nhập URL mà Heroku đã hiển thị cho ta, nhưng đó là một cách khác để mở trang web. Chúng ta sẽ thấy trang chủ của Nhật ký Học tập, được viết đúng kiểu. Tuy nhiên vẫn chưa thể sử dụng ứng dụng vì chúng ta chưa thiết lập cơ sở dữ liệu.

Thiết lập cơ sở dữ liệu trên Heroku:

Chạy migrate một lần để thiết lập cơ sở dữ liệu trực tiếp và áp dụng tất cả các di chuyển mà ta đã tạo trong quá trình phát triển. Chúng ta có thể chạy các lệnh

Django và Python trên một dự án Heroku bằng cách sử dụng lệnh chạy heroku. Dưới đây là cách chạy di chuyển khi triển khai Heroku:

```
(11_env)learning_log$ heroku run python manage.py migrate
Running 'python manage.py migrate' on ⌁ secret-lowlands-82594... up, run.3060
--snip--
Running migrations:
--snip--
Applying learning_logs.0001_initial... OK
Applying learning_logs.0002_entry... OK
Applying learning_logs.0003_topic_owner... OK
Applying sessions.0001_initial... OK
(11_env)learning_log$
```

Đầu tiên chúng ta phát hành lệnh **heroku run python management.py migrate**. Sau đó, Heroku tạo một phiên đầu cuối để chạy lệnh di chuyển.

Bây giờ khi truy cập ứng dụng đã triển khai của mình, ta có thể sử dụng nó giống như ta đã làm trên hệ thống cục bộ của mình. Nhưng sẽ không thấy bất kỳ dữ liệu nào ta đã nhập khi triển khai cục bộ, bao gồm cả tài khoản superuser, bởi vì chúng ta đã không sao chép dữ liệu vào máy chủ trực tiếp. Đây là một vấn đề bình thường: chúng ta không thường sao chép dữ liệu cục bộ vào một triển khai trực tiếp vì dữ liệu cục bộ thường là dữ liệu thử nghiệm.

Chúng ta có thể chia sẻ liên kết Heroku của mình để cho phép bất kỳ ai sử dụng phiên bản Nhật ký học tập. Trong phần tiếp theo hoàn thành một số tác vụ khác để kết thúc quá trình triển khai và giúp ta tiếp tục phát triển Nhật ký học tập.

Điều chỉnh việc triển khai Heroku:

Bây giờ chúng ta sẽ điều chỉnh việc triển khai bằng cách tạo superuser. Chúng ta cũng sẽ làm cho dự án an toàn hơn bằng cách thay đổi cài đặt DEBUG thành False, vì vậy người dùng sẽ không thấy bất kỳ thông tin bổ sung nào trong thông báo lỗi mà họ có thể sử dụng để tấn công máy chủ.

Tạo một superuser trên Heroku:

Bash là ngôn ngữ chạy trong nhiều thiết bị đầu cuối Linux. Chúng ta sẽ sử dụng phiên Bash terminal để tạo siêu người dùng để có thể truy cập trang web quản trị trên ứng dụng trực tiếp:

```
(11_env)learning_log$ heroku run bash
Running 'bash' on ⌁ secret-lowlands-82594... up, run.9858
```

```
~ $ ls
learning_log learning_logs manage.py Procfile requirements.txt runtime.txt
staticfiles users
~ $ python manage.py createsuperuser
Username (leave blank to use 'u47318'): ll_admin
Email address:
Password:
Password (again):
Superuser created successfully.
~ $ exit
exit
(ll_env)learning_log$
```

Đầu tiên, chúng ta chạy ls để xem các tệp và thư mục nào tồn tại trên máy chủ, đó phải là các tệp giống như chúng ta có trên hệ thống cục bộ. Ta có thể điều hướng hệ thống tệp này giống như bất kỳ hệ thống tệp nào khác.

Chú ý : Người dùng Windows sẽ sử dụng các lệnh dir thay cho ls vì chúng ta đang chạy một thiết bị đầu cuối Linux thông qua kết nối từ xa.

Tiếp theo, chúng ta chạy lệnh tạo superuser, lệnh này xuất ra các lời nhắc giống như chúng ta đã thấy trên hệ thống cục bộ của mình khi chúng ta tạo superuser trong chương trước. Khi hoàn tất việc tạo superuser trong phiên đầu cuối này, hãy chạy lệnh exit để quay lại. đến phiên đầu cuối của hệ thống cục bộ .

Bây giờ ta có thể thêm / admin / vào cuối URL của ứng dụng trực tiếp và đăng nhập vào trang quản trị. Nếu những người khác đã bắt đầu sử dụng dự án của chúng ta, hãy lưu ý rằng chúng ta sẽ có quyền truy cập vào tất cả dữ liệu của họ! Đừng xem nhẹ điều này và người dùng sẽ tiếp tục tin tưởng chúng ta cung cấp dữ liệu của họ.

Tạo URL thân thiện với người dùng trên Heroku:

Hầu hết, mọi người đều muốn URL của mình thân thiện và dễ nhớ hơn <https://secret-lowlands-82594.herokuapp.com/> vì vậy ta có thể đổi tên ứng dụng bằng một lệnh duy nhất:

```
(ll_env)learning_log$ heroku apps:rename learning-log
Renaming secret-lowlands-82594 to learning-log-2e... done
https://learning-log.herokuapp.com/ | https://git.heroku.com/learning-log.git
Git remote heroku updated
Don't forget to update git remotes for all other local checkouts of the app.
(ll_env)learning_log$
```

Chú ý : Khi chúng ta triển khai dự án của mình bằng dịch vụ miễn phí của Heroku, Heroku sẽ đặt việc triển khai ở trạng thái ngủ nếu nó không nhận được bất kỳ yêu cầu nào sau một khoảng thời gian nhất định hoặc nếu nó quá hoạt động đối với cấp miễn phí. Lần đầu tiên người dùng truy cập vào trang web sau khi nó ở chế độ ngủ, sẽ mất nhiều thời gian tải hơn, nhưng máy chủ sẽ phản hồi các yêu cầu tiếp theo nhanh hơn. Đây là cách Heroku có đủ khả năng để cung cấp các triển khai miễn phí.

Bảo mật Dự án Trực tiếp

Các trang lỗi của Django cung cấp cho ta thông tin gỡ lỗi quan trọng khi ta đang phát triển một dự án; tuy nhiên, chúng cung cấp quá nhiều thông tin cho những kẻ tấn công nếu chúng ta bật chúng trên một máy chủ trực tiếp. Biến môi trường là các giá trị được đặt trong một môi trường cụ thể. Đây là một trong những cách thông tin nhạy cảm được lưu trữ trên máy chủ, giữ nó tách biệt với phần còn lại của mã của dự án. Hãy sửa đổi settings.py để nó tìm kiếm biến môi trường khi dự án đang chạy trên Heroku:

```
--snip--  
# Heroku settings.  
import django_heroku  
django_heroku.settings(locals())  
if os.environ.get('DEBUG') == 'TRUE':  
    DEBUG = True  
elif os.environ.get('DEBUG') == 'FALSE':  
    DEBUG = False
```

Phương thức os.environ.get() đọc giá trị được liên kết với một biến môi trường cụ thể trong bất kỳ môi trường nào mà dự án đang chạy. Nếu biến yêu cầu được đặt, phương thức sẽ trả về giá trị của nó; nếu nó không được đặt, phương thức trả về None. Việc sử dụng các biến môi trường để lưu trữ các giá trị Boolean có thể gây nhầm lẫn. Trong hầu hết các trường hợp, các biến môi trường được lưu trữ dưới dạng chuỗi và chúng ta phải cẩn thận về điều này. Hãy xem xét đoạn mã này từ một phiên đầu cuối Python đơn giản:

```
>>> bool('False')  
True
```

Giá trị Boolean của chuỗi 'False' là True, bởi vì bất kỳ chuỗi không trống nào đều đánh giá là True. Vì vậy, chúng ta sẽ sử dụng các chuỗi "TRUE" và "FALSE" viết

in hoa, để chỉ ra rằng chúng ta không lưu trữ các giá trị Boolean True và False thực tế của Python. Khi Django đọc trong biến môi trường bằng khóa 'DEBUG' trên Heroku, ta đặt DEBUG thành True nếu giá trị là 'TRUE' và False nếu giá trị là 'FALSE'.

Cam kết và đẩy mã thay đổi:

Bây giờ chúng ta cần chuyển các thay đổi đó với settings.py vào kho lưu trữ Git, sau đó đẩy các thay đổi lên Heroku. Dưới đây là phiên đầu cuối cho quá trình này:

```
(ll_env)learning_log$ git commit -am "Set DEBUG based on environment variables."  
[master 3427244] Set DEBUG based on environment variables.  
 1 file changed, 4 insertions(+)  
(ll_env)learning_log$ git status  
On branch master  
nothing to commit, working tree clean  
(ll_env)learning_log$
```

Chúng ta phát hành lệnh git commit với một thông báo commit ngắn nhưng mang tính mô tả. Hãy nhớ rằng cờ -am đảm bảo Git cam kết tất cả các tệp đã thay đổi và ghi lại thông báo nhật ký. Git nhận ra rằng một tệp đã thay đổi và cam kết thay đổi này với kho lưu trữ.

Git status cho thấy rằng chúng ta đang làm việc trên nhánh chính của kho lưu trữ và hiện không có thay đổi nào để cam kết. Điều cần thiết là chúng ta phải kiểm tra trạng thái của thông báo này trước khi chuyển sang Heroku. Nếu không thấy thông báo này, một số thay đổi chưa được cam kết và những thay đổi đó sẽ không được đưa lên máy chủ.

Bây giờ chúng ta hãy đẩy kho lưu trữ cập nhật lên Heroku:

```
(ll_env)learning_log$ git push heroku master  
remote: Building source:  
remote:  
remote: -----> Python app detected  
remote: -----> Installing requirements with pip  
--snip--  
remote: -----> Launching...  
remote: Released v6  
remote: https://learning-log.herokuapp.com/ deployed to Heroku  
remote:
```

```
remote: Verifying deploy... done.  
To https://git.heroku.com/learning-log.git  
 144f020..d5075a1 master -> master  
(11_env)learning_log$
```

Heroku nhận ra rằng kho lưu trữ đã được cập nhật và nó xây dựng lại dự án để đảm bảo rằng tất cả các thay đổi đã được tính đến. Nó không xây dựng lại cơ sở dữ liệu, vì vậy chúng ta sẽ không phải chạy di chuyển cho bản cập nhật này.

Đặt các biến môi trường trên Heroku:

Bây giờ chúng ta có thể đặt giá trị chúng ta muốn cho DEBUG trong settings.py thông qua Heroku. Lệnh heroku config: set đặt một biến môi trường:

```
(11_env)learning_log$ heroku config:set DEBUG='FALSE'  
Setting DEBUG and restarting ⏱ learning-log... done, v7  
DEBUG: FALSE  
(11_env)learning_log$
```

Bất cứ khi nào chúng ta đặt một biến môi trường trên Heroku, nó sẽ tự động khởi động lại dự án để biến môi trường có thể có hiệu lực.

Để kiểm tra xem việc triển khai hiện đã an toàn hơn chưa, hãy nhập URL của dự án bằng đường dẫn mà chúng ta chưa xác định. Ví dụ: hãy thử truy cập <http://learning-log.herokuapp.com/letmein/>. Ta sẽ thấy một trang lỗi chung trong quá trình triển khai trực tiếp của mình, trang này không cung cấp bất kỳ thông tin cụ thể nào về dự án. Nếu chúng ta thử yêu cầu tương tự trên phiên bản cục bộ của Nhật ký học tập tại <http://localhost:8000/letmein/>, ta sẽ thấy trang lỗi Django đầy đủ. Kết quả là hoàn hảo: ta sẽ thấy các thông báo lỗi cung cấp thông tin khi đang phát triển thêm dự án trên hệ thống của riêng mình. Nhưng người dùng trên trang web trực tiếp sẽ không thấy thông tin quan trọng về mã của dự án.

Nếu chỉ đang triển khai một ứng dụng và đang gỡ rối quá trình triển khai ban đầu, ta có thể chạy câu lệnh heroku: set DEBUG = 'TRUE' và tạm thời xem một báo cáo lỗi đầy đủ trên trang web trực tiếp. Chỉ cần đảm bảo đặt lại giá trị thành 'FALSE' sau khi hoàn tất khắc phục sự cố. Ngoài ra, hãy cẩn thận không làm điều này khi người dùng thường xuyên truy cập vào trang web.

Tạo các trang lỗi tùy chỉnh:

Trong phần trước, chúng ta đã định cấu hình Nhật ký học tập để trả về lỗi 404 nếu người dùng yêu cầu một chủ đề hoặc mục nhập không thuộc về họ. Có thể chúng

ta đã thấy khoảng 500 lỗi máy chủ (lỗi nội bộ) cho đến thời điểm này. Lỗi 404 thường có nghĩa là mã Django đúng, nhưng đối tượng được yêu cầu không tồn tại; lỗi 500 thường có nghĩa là có lỗi trong mã đã viết, chẳng hạn như lỗi trong một hàm trong views.py. Hiện tại, Django trả về cùng một trang lỗi chung trong cả hai trường hợp. Nhưng chúng ta có thể viết các mẫu trang lỗi 404 và 500 của riêng mình phù hợp với giao diện tổng thể của Nhật ký học tập. Các mẫu này phải đi trong thư mục mẫu gốc.

Tạo mẫu tùy chỉnh

Trong thư mục learning_log ngoài cùng, hãy tạo một thư mục mới có tên là các mẫu,templates. Sau đó, tạo một tệp mới có tên 404.html; đường dẫn đến tệp này phải là learning_log / templates / 404.html. Đây là mã cho tệp này:

```
{% extends "learning_logs/base.html" %}  
{% block page_header %}  
    <h2>The item you requested is not available. (404)</h2>  
{% endblock page_header %}
```

Mẫu đơn giản này cung cấp thông tin chung về trang lỗi 404 nhưng được tạo kiểu để phù hợp với phần còn lại của trang web. Tạo một tệp khác có tên 500.html bằng cách sử dụng mã sau:

```
{% extends "learning_logs/base.html" %}  
{% block page_header %}  
    <h2>There has been an internal error. (500)</h2>  
{% endblock page_header %}
```

Các tệp mới này yêu cầu một chút thay đổi đối với settings.py.

```
--snip--  
TEMPLATES = [  
    {  
        'BACKEND': 'django.template.backends.django.DjangoTemplates',  
        'DIRS': [os.path.join(BASE_DIR, 'templates')],  
        'APP_DIRS': True,  
        '--snip--  
    },  
]
```

Xem cục bộ các trang lỗi:

Nếu chúng ta muốn xem các trang lỗi trông như thế nào trên hệ thống của mình trước khi đẩy chúng lên Heroku, trước tiên cần đặt Debug = False trên cài đặt cục bộ của mình để loại bỏ các trang gỡ lỗi Django mặc định. Để làm như vậy, hãy thực hiện

thay đổi sau đói với settings.py (đảm bảo rằng ta đang làm việc trong phần settings.py áp dụng cho môi trường cục bộ, không phải phần áp dụng cho Heroku):

```
--snip--  
# SECURITY WARNING: don't run with debug turned on in production!  
DEBUG = False  
--snip--
```

Bây giờ, hãy yêu cầu một chủ đề hoặc mục nhập không thuộc dữ liệu mà ta có để xem trang lỗi 404. Để kiểm tra trang lỗi 500, hãy yêu cầu một chủ đề hoặc mục nhập không tồn tại. Ví dụ: URL http://localhost: 8000 / themes / 999 / sẽ tạo ra lỗi 500 trừ khi ta đã tạo 999 chủ đề mẫu!

Khi kiểm tra xong các trang lỗi, hãy đặt giá trị cục bộ của DEBUG trở lại True để phát triển thêm Nhật ký học tập. (Đảm bảo rằng ta không thay đổi cách xử lý DEBUG trong phần quản lý cài đặt trong môi trường Heroku.)

Chú ý: Trang lỗi 500 sẽ không hiển thị bất kỳ thông tin nào về người dùng đã đăng nhập vì Django không gửi bất kỳ thông tin ngữ cảnh nào trong phản hồi khi có lỗi máy chủ.

Đây sự thay đổi mã lên heroku:

Bây giờ chúng ta cần cam kết các thay đổi trang báo cáo lỗi mà chúng ta vừa thực hiện và đẩy chúng trực tiếp lên Heroku:

```
(11_env)learning_log$ git add .  
(11_env)learning_log$ git commit -am "Added custom 404 and 500 error pages."  
 3 files changed, 15 insertions(+), 10 deletions(-)  
  create mode 100644 templates/404.html  
  create mode 100644 templates/500.html  
(11_env)learning_log$ git push heroku master  
--snip--  
remote: Verifying deploy.... done.  
To https://git.heroku.com/learning-log.git  
 d5075a1..4bd3b1c master -> master  
(11_env)learning_log$
```

Sử dụng phương thức get_object_or_404():

Tại thời điểm này, nếu người dùng yêu cầu chủ đề hoặc mục nhập không tồn tại theo cách thủ công, họ sẽ gặp lỗi máy chủ 500. Django có gắng hiển thị trang không tồn tại, nhưng nó không có đủ thông tin để làm như vậy và kết quả là lỗi 500. Tình huống này được xử lý chính xác hơn là lỗi 404 và chúng ta có thể thực hiện hành

vi này bằng cách sử dụng hàm của Django `get_object_or_404()`. Hàm này có găng lấy đối tượng được yêu cầu từ cơ sở dữ liệu, nhưng nếu đối tượng đó không tồn tại, nó sẽ tạo ra một ngoại lệ 404. Chúng ta sẽ nhập hàm này vào `views.py` và sử dụng nó thay cho `get()`:

```
from django.shortcuts import render, redirect, get_object_or_404
from django.contrib.auth.decorators import login_required
--snip--
@login_required
def topic(request, topic_id):
    """Show a single topic and all its entries."""
    topic = get_object_or_404(Topic, id=topic_id)
    # Make sure the topic belongs to the current user.
    --snip-
```

Dang phat trien:

Ta có thể muốn phát triển hơn nữa Nhật ký học tập sau lần đầu tiên lên máy chủ trực tiếp hoặc phát triển các dự án của riêng chúng ta để triển khai. Có một quy trình khá nhất quán để cập nhật các dự án.

Trước tiên, chúng ta sẽ thực hiện bất kỳ thay đổi nào cần thiết cho dự án địa phương của mình. Nếu các thay đổi dẫn đến bất kỳ tệp mới nào, hãy thêm các tệp đó vào kho lưu trữ Git bằng cách sử dụng lệnh `git add`. (đảm bảo bao gồm dấu chấm ở cuối lệnh). Bất kỳ thay đổi nào yêu cầu di chuyển cơ sở dữ liệu sẽ cần lệnh này, vì mỗi lần di chuyển tạo ra một tệp di chuyển mới.

Thứ hai, cam kết các thay đổi đối với kho lưu trữ bằng cách sử dụng `git commit -am "commit message"`. Sau đó đẩy các thay đổi lên Heroku bằng lệnh `git push heroku master`. Nếu chúng ta đã di chuyển cơ sở dữ liệu của mình cục bộ, ta cũng cần phải di chuyển cơ sở dữ liệu trực tiếp. Chúng ta có thể sử dụng một lần `com mand heroku run python management.py migrate` hoặc mở một thiết bị đầu cuối từ xa `ses sion` với `heroku run bash` và chạy lệnh `python management.py migrate`. Sau đó, hãy truy cập dự án trực tiếp của mình ta và đảm bảo rằng những thay đổi ta mong đợi đã có hiệu lực.

Cài đặt Secret_Key:

Django sử dụng giá trị của cài đặt `SECRET_KEY` trong `settings.py` để triển khai một số giao thức bảo mật. Trong dự án này, chúng ta đã cam kết tệp cài đặt của mình vào kho lưu trữ có bao gồm cài đặt `SECRET_KEY`. Điều này là tốt cho một dự

án thực hành, nhưng cài đặt SECRET_KEY nên được xử lý cẩn thận hơn đối với địa điểm sản xuất. Nếu ta xây dựng một dự án đang được sử dụng có ý nghĩa, hãy đảm bảo nghiên cứu cách xử lý cài đặt SECRET_KEY của mình một cách an toàn hơn.

Xóa dự án trên heroku:

Ta cần biết cách xóa dự án đã được triển khai. Heroku cũng giới hạn số lượng dự án có thể lưu trữ miễn phí và chúng ta không muốn làm lộn xộn tài khoản của mình với các dự án thực tế.

Đăng nhập vào trang web Heroku (<https://heroku.com/>); ta sẽ được chuyển hướng đến một trang hiển thị danh sách các dự án của chính mình. Nhập vào dự án ta muốn xóa. Chúng ta sẽ thấy một trang mới với thông tin về dự án. Nhập vào liên kết Cài đặt và cuộn xuống cho đến khi thấy liên kết để xóa dự án. Không thể hoàn tác hành động này, vì vậy Heroku sẽ yêu cầu xác nhận yêu cầu xóa bằng cách nhập tên dự án theo cách thủ công.

Nếu thích làm việc từ một thiết bị đầu cuối, ta cũng có thể xóa dự án bằng cách ra lệnh hủy:

```
(11_env)learning_log$ heroku apps:destroy --app appname
```

Chú ý : Việc xóa dự án trên Heroku không ảnh hưởng gì đến phiên bản dự án cục bộ . Nếu chưa có ai sử dụng dự án đã triển khai và chúng ta chỉ đang thực hành quy trình triển khai, thì việc xóa dự án trên Heroku và triển khai lại là hoàn toàn hợp lý.

10.3.3.3. Kết mục

Trong chương này, chúng ta đã học cách tạo cho các dự án của mình một diện mạo đơn giản nhưng chuyên nghiệp bằng cách sử dụng thư viện Bootstrap và ứng dụng django-bootstrap4. Sử dụng Bootstrap, các kiểu mà ta chọn sẽ hoạt động nhất quán trên hầu hết mọi thiết bị mà mọi người sử dụng để truy cập dự án.

Ta đã tìm hiểu về các mẫu của Bootstrap và sử dụng mẫu tĩnh Navbar để tạo giao diện đơn giản cho Nhật ký học tập. Sử dụng jumbotron để làm nổi bật thông điệp của trang chủ và học cách tạo kiểu nhất quán cho tất cả các trang trong một trang web.

Ta đã tạo tài khoản Heroku và cài đặt một số công cụ giúp quản lý quá trình triển khai. Sử dụng Git để cam kết dự án đang hoạt động vào một kho lưu trữ và sau

đó đây kho lưu trữ đó đến các máy chủ của Heroku. Cuối cùng, ta đã học cách bắt đầu bảo mật ứng dụng của mình bằng cách đặt DEBUG = False trên máy chủ trực tiếp.

Bây giờ chúng ta đã hoàn thành Nhật ký học tập, có thể bắt đầu xây dựng các dự án của riêng mình. Bắt đầu đơn giản và đảm bảo dự án hoạt động trước khi thêm phức tạp. Hãy tiếp tục học hỏi và chúc may mắn với các dự án của chúng ta!

10.4. Kết chương

Chương dự án đã trình bày nội dung chi tiết của các dự án bao gồm ba lĩnh vực chính là làm trò chơi (ứng dụng thư viện Pygame), trực quan hóa dữ liệu (ứng dụng matplotlib, seaborn, plotly) và làm web (ứng dụng framework Django). Áp dụng các kiến thức cơ bản đã học trong các chương trước, nội dung của chương đã hướng dẫn chi tiết cụ thể các bước tạo ra các ứng dụng thuộc các dạng trên. Đó là những dự án cụ thể, có quy mô và có tính ứng dụng rất cao của Python.

Giải quyết thành công những bài toán trên, lập trình viên Python chứng tỏ được khả năng làm chủ ngôn ngữ lập trình, kết hợp các kiến thức tích hợp để sẵn sàng cho những thử thách lớn hơn.

TÀI LIỆU THAM KHẢO

Tiếng Anh

1. Eric Matthes. Python crash course, 2nd Edition: A hands-on, project-based introduction to programming, No Starch Press, 2019.
2. Allen B. Downey, Think Python: How to Think Like a Computer Scientist, O'Reilly, 2015
3. Zed A. Shaw, Learn Python 3 the Hard Way, Addison-Wesley, 2016