Mục lục

[Những ứng dụng đầu tiên Cú pháp cơ bản và “ Xin chào thế giới !”](#_Toc177999702)  [2](#_Toc177999702)

[Điều khiển rồi thực thi lệnh (phân nhánh): giới thiệu](#_Toc177999703)  [8](#_Toc177999703)

[Chức năng](#_Toc177999704)  [21](#_Toc177999704)

[Danh sách](#_Toc177999705)  [30](#_Toc177999705)

[Chu kỳ](#_Toc177999706)  [48](#_Toc177999706)

[Dòng](#_Toc177999707)  [57](#_Toc177999707)

[Từ Điển](#_Toc177999708)  [70](#_Toc177999708)

[Tập tin](#_Toc177999709)  [82](#_Toc177999709)

[Nhiệm vụ cuối cùng](#_Toc177999710)  [91](#_Toc177999710)

Những ứng dụng đầu tiên Cú pháp cơ bản và “ Xin chào thế giới !”

Ngôn ngữ lập trình Python được coi là một trong những ngôn ngữ lập trình dễ học nhất. Việc sử dụng nó cho phép sử dụng cả hai cách tiếp cận chức năng, thủ tục và hướng đối tượng để lập trình. Ngôn ngữ này hỗ trợ các kiểu dữ liệu cấp cao cùng với khả năng thực hiện cách tiếp cận đối tượng, điều này làm cho ngôn ngữ này trở thành một công cụ mạnh mẽ để tạo các ứng dụng đa nền tảng khác nhau.

Để bắt đầu tạo ứng dụng đầu tiên của bạn bằng python , hãy nghiên cứu các cấu trúc cú pháp cơ bản cần thiết để viết bất kỳ ứng dụng nào.

Trước khi bắt đầu đào tạo, hãy giới thiệu một số quy ước:

>> - ký hiệu này biểu thị mã được nhập vào IDE (Môi trường phát triển tích hợp, môi trường phát triển tích hợp, ví dụ Spyder, Jupiter, Visual studio, v.v.).

XIN CHÀO! - chữ màu trắng trên nền đen cho biết kết quả đầu ra tới bảng điều khiển (một tiện ích đặc biệt cung cấp hỗ trợ liên lạc trực tiếp giữa người dùng và HĐH)

Bài học này sẽ bao gồm các cấu trúc như:

* Các kiểu dữ liệu được sử dụng phổ biến nhất (chuỗi và số)
* chức năng in (đầu ra bàn điều khiển)
* Biến
* Các thao tác với số
* Bình luận
* Các loại lỗi

Là một phần của công việc này, các ứng dụng đầu tiên sẽ được viết bằng ngôn ngữ Python đang được nghiên cứu .

**Các kiểu dữ liệu được sử dụng phổ biến nhất**

**Chuỗi**

Chuỗi là một tin nhắn văn bản có thể được xử lý bằng các công cụ tích hợp của ngôn ngữ lập trình. Trong python, chuỗi có thể được chỉ định theo nhiều cách:

* Chúng có thể được đặt trong dấu ngoặc đơn ('…')
* Chúng có thể được đặt trong dấu ngoặc kép (“…”)

Cả hai phương pháp đều hoàn toàn giống nhau và bạn có thể sử dụng một trong hai phương pháp đó. Tuy nhiên, hãy nhớ rằng phải có dấu ngoặc kép giống nhau ở đầu và cuối chuỗi đầu vào!

Chuỗi có kiểu dữ liệu Chuỗi .

**số**

Các số trong python được chỉ định làm tiêu chuẩn, giống như trong tất cả các ngôn ngữ lập trình khác. Có hai loại số: số nguyên và phân số (hoặc số dấu phẩy động). Kiểu dữ liệu tương ứng với số nguyên là số nguyên ( int ). Loại này chứa cả số nguyên âm và số nguyên dương (ví dụ: -1, -2, 2, 3) và 0. Khi viết số như vậy không được phép sử dụng mantissa (phần phân số). Những con số như vậy được sử dụng khi đếm đồ vật, đánh số dòng, v.v.

Mặt khác, số dấu phẩy động tồn tại để ghi các số phân số (số dấu phẩy động). Kiểu dữ liệu được sử dụng phổ biến nhất trong Python cho các mục đích này là float . Đối với các tác vụ yêu cầu độ chính xác cao hơn của phép đo và tính toán, kiểu dữ liệu kép được sử dụng .

Số dấu phẩy động được sử dụng để đo các giá trị tương đối, số liệu thống kê, v.v. Ví dụ: Để tính chiều cao trung bình của học sinh trong một lớp.

**chức năng in**

Để giao tiếp với máy tính, bạn cần một công cụ hiển thị các thông báo về trạng thái hiện tại của nó hoặc các hoạt động mà nó đang thực hiện. Python có sẵn chức năng in cho mục đích này . Hàm này hiển thị thông báo chuỗi. Để chỉ định chuỗi cần xuất ra, nội dung thông báo được viết trong ngoặc đơn dưới dạng chuỗi python (trong dấu ngoặc kép):

>> print ( “Chạm vào người đẹp, ôm lập trình viên!” )

Trong đoạn mã này, bạn cần hiển thị một đoạn trích từ cuốn sách ra bảng điều khiển. Kết quả của việc thực thi mã này sẽ là thông báo sau:

Chạm vào người đẹp, ôm lấy lập trình viên!

**Biến**

Ở trên, chúng tôi đã thảo luận về cách bạn có thể nhập một giá trị cụ thể vào môi trường python và xuất nó ra bảng điều khiển. Những giá trị như vậy được gọi là **chữ** . Ví dụ,

Chạm vào người đẹp, ôm lấy lập trình viên!

2

Trong trường hợp này, một chuỗi và một số được đưa ra đơn giản, tuy nhiên, chúng không được lưu ở bất cứ đâu. Để có thể sử dụng lại các giá trị này, bạn phải phân bổ dung lượng trong bộ nhớ máy tính nơi dữ liệu này sẽ được lưu trữ để sử dụng trong tương lai. Để truy cập các giá trị này, một vùng bộ nhớ máy tính nhất định được phân bổ và phải được đặt tên.

Các biến tồn tại trong ngôn ngữ lập trình cho những mục đích này. Để đặt biến, bạn phải nhập tên biến, dấu bằng (=) và giá trị cần đặt ở đó.

Ví dụ:

>> tin nhắn\_chuỗi = "CHÀO"

>>># sẽ in "Xin chào"

>> in ( tin nhắn \_ chuỗi )

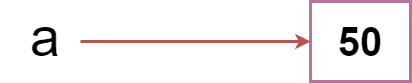
Trong trường hợp này, biến message\_string lưu trữ một chuỗi được xuất ra bảng điều khiển bằng cách truy cập vào biến đó.

Trên thực tế, trong Python, các biến là một tên tượng trưng dùng để tham chiếu hoặc con trỏ tới một đối tượng (vùng bộ nhớ).

Vì vậy, nếu có một mục:

>> a =50

Điều này có nghĩa là một địa chỉ ô cụ thể chứa giá trị 50 và chúng ta có thể truy cập số đó bằng tên a , nghĩa là a đề cập đến giá trị 50:



Hãy xem xét một trường hợp khác. Ví dụ,

>> a =50

>> b = a

Trong trường hợp này, cả a và b đều tham chiếu đến cùng một vùng bộ nhớ (đối tượng là số 50). Đó là:

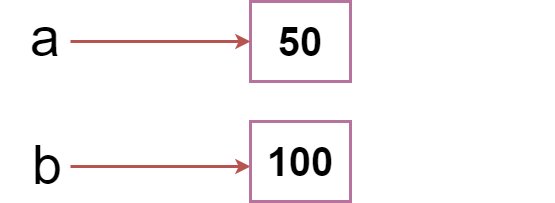


Nhưng nếu chúng ta viết nó như thế này:

>> a =50

>> b =50

Trong trường hợp này, a và b sẽ đề cập đến các đối tượng khác nhau.



**Tên biến**

Tên biến không được chứa dấu cách hoặc ký tự khác ngoài dấu gạch dưới (\_). Tên biến không thể bắt đầu bằng số nhưng có thể có số sau ký tự đầu tiên (ví dụ: cool\_variable\_5 có thể chấp nhận được). Tên biến không được chứa các ký tự đặc biệt (!, @, #,%, ^, &, \*). Tên biến không được trùng với các từ dành riêng cho python , ví dụ không được phép đặt tên biến print , type , def , v.v. Cũng cần nhớ rằng python phân biệt chữ hoa chữ thường, tức là My \_ var và my \_ var là những biến khác nhau!

Khi đặt tên biến, bạn phải chọn tên phản ánh được mục đích của nó. Ví dụ: đối với một biến chứa tuổi, tên tuổi sẽ phù hợp . Python hỗ trợ mã hóa utf -8 nên tên có thể chứa ký tự Cyrillic và điều này sẽ không gây ra lỗi. Tuy nhiên, theo tiêu chuẩn clean code, bảng chữ cái Cyrillic trong tên các đối tượng trong chương trình là không nên.

Khi gán tên cho một biến chứa nhiều từ, có thể chấp nhận sử dụng một số cách tiếp cận được chấp nhận rộng rãi:

* Trường hợp lạc đà - Mỗi từ hoặc chữ viết tắt ở giữa bắt đầu bằng chữ in hoa, không có dấu cách. Ví dụ: nameOfStudent, valueOfVaraible, v.v.
* Pascal Case giống như Camel Case nhưng ở đây từ đầu tiên được viết hoa. Ví dụ: NameOfStudent, v.v.
* Trường hợp rắn – các từ được phân cách bằng cách gạch chân. Ví dụ – tên\_of\_student và T.​ d .

Các giá trị trong biến có thể được sửa đổi trong quá trình thực thi chương trình, điều này giải thích nguồn gốc tên của cấu trúc ngôn ngữ lập trình này.

>>># Xin chào

>>chuỗi tin nhắn = " CHÀO "

>>in ( message\_string )

>>># Đang cập nhật nghĩa biến

>>chuỗi tin nhắn = "Hasta la vista"

>>in ( message\_string )

Trong ví dụ này, lúc đầu, văn bản chào mừng được nhập vào biến message\_string, sau đó xuất ra bảng điều khiển. Sau đó, giá trị của biến được thay thế bằng lời tạm biệt và sau đó cũng được in ra bảng điều khiển.

**Bài tập**

1. Hãy tạo một ứng dụng sẽ bắt đầu học tất cả các ngôn ngữ lập trình cổ điển. Ở giai đoạn đầu tiên, hãy in lời chào Xin chào thế giới!
2. Tạo một biến lời chào và đặt nó thành Hello , world !
3. In giá trị của biến lời chào . Chúc mừng! Bạn đã tạo ứng dụng đầu tiên của mình bằng python

**Định dạng chuỗi khi xuất ra bảng điều khiển**

Đôi khi, cần phải hiển thị một chuỗi cùng lúc với giá trị của một biến. Ví dụ: có một biến:

>> tên = 'Andrey'

Chúng tôi muốn xuất một chuỗi ra bàn điều khiển

Xin chào Andrey!

Trong trường hợp này, giá trị 'Andrey' được chứa trong biến tên mà chúng tôi muốn hiển thị bên trong chuỗi. Để giải quyết vấn đề này, có 2 cách tiếp cận:

1. Thêm chuỗi (nối)

Các chuỗi, giống như số, có thể được thêm vào. Điều quan trọng cần ghi nhớ là một chuỗi chỉ có thể được thêm vào một chuỗi! Toán tử + được sử dụng để thêm chuỗi.

>> tên = 'Andrey'

>> in ('Xin chào, '+ tên +'!')

Lệnh này sẽ cho phép chúng tôi hiển thị kết quả mong muốn.

1. Định dạng chuỗi.

định dạng để hiển thị một biến trong chuỗi . Để xuất chuỗi chúng ta muốn, lệnh sau sẽ được sử dụng:

>> tên = 'Andrey'

>> print ('Xin chào, {}!'. định dạng ( tên ))

**Các thao tác với số**

Trong ngôn ngữ Python , bạn có thể thực hiện các phép tính số học trên các số như cộng, trừ, chia và nhân (+, -, /, \*).

>> in ( 573 - 74 + 1 )

>> in ( 25 \* 2 )

>> in ( 10 / 5 )

Thao tác cuối cùng sẽ dẫn đến một số dấu phẩy động vì phép chia chuyển đổi kiểu dữ liệu số nguyên thành kiểu dữ liệu float .

Chia cho 0 sẽ dẫn đến ZeroDivisionError.

**Bình luận**

Một trong những công cụ hữu ích nhất trong python là bình luận. Chúng cho phép bạn làm cho mã của mình dễ đọc và dễ hiểu hơn. Để tạo bình luận, bạn phải nhập dấu #. Bất cứ điều gì được nhập vào một dòng sau dấu # sẽ được nhận dạng là một nhận xét và trình biên dịch sẽ bỏ qua dữ liệu trên dòng đó khi thực thi chương trình.

Bình luận có thể:

• Cung cấp ngữ cảnh để hiểu lý do tại sao chương trình được viết theo cách này (ví dụ: tại sao một biến bổ sung được đưa vào, v.v.):

>># Biến này chứa thông tin về số lượt xem của khóa học

>>số người xem = 0

• Giúp các lập trình viên khác đọc mã của bạn:

>>># Biến này chứa thông tin về nhân viên

>> Số \_ cá nhân = 10

• Bỏ qua một phần mã khi gỡ lỗi ứng dụng để tìm lỗi xảy ra ở dòng nào

>># value\_for\_test = 0

>>Giá\_trị\_mới\_for\_test = 1

**Lỗi**

Trong các ngôn ngữ lập trình, có thể tìm thấy lỗi mã và giải thích chúng.

Python gọi những lỗi này là " Errors " và cho biết vị trí lỗi bằng ký tự ^.

Có 2 loại lỗi có thể xảy ra trong python :

* Cú phápError - có nghĩa là có gì đó không đúng trong chính văn bản của chương trình (cú pháp) - ví dụ: dấu câu không phù hợp không được sử dụng cho một thao tác nhất định. Ngoài ra, thiếu dấu ngoặc đơn cũng có thể gây ra loại lỗi này.
* NameError - Xảy ra khi trình thông dịch Python không thể nhận dạng một từ. Ví dụ: một biến không tồn tại.

**Bài tập**

1. Bạn cần tính toán xem mình sẽ cần bao nhiêu viên gạch để cải tạo căn phòng! Tạo các biến length và width và gán cho chúng giá trị lần lượt là 8 và 10.
2. Hóa ra đã xảy ra lỗi trong tính toán và cần có độ dài 20 ô. Cần bao nhiêu viên gạch trong trường hợp này?

Riêng mình! Tính diện tích hình chữ nhật có cạnh 23, 13.

Quản lý thực thi lệnh sau (phân nhánh): giới thiệu

Mỗi ngày chúng ta phải đối mặt với vấn đề lựa chọn tùy theo điều kiện. Ví dụ, làm thế nào để đi làm? Nếu bằng phương tiện công cộng thì vấn đề là chúng ta đi đông người nhưng sẽ đến đó đúng lịch trình. Nếu bạn lái ô tô riêng thì chuyến đi sẽ thoải mái hơn rất nhiều nhưng bạn có thể gặp khó khăn trong việc tìm chỗ đậu xe. Những vấn đề như vậy được giải quyết bằng cách sử dụng quy tắc “Nếu, thì”. Ví dụ, nếu trời lạnh, bạn cần mặc áo khoác, v.v. Nói cách khác, một số thứ tự hành động của chúng ta phụ thuộc vào việc liệu một số điều kiện có được thỏa mãn hay không.

Trong Python, chuỗi hành động là một tập hợp các lệnh tuân theo một thứ tự cụ thể. Trình tự thực hiện lệnh được gọi là thread. Cho đến nay, chúng ta chỉ gặp phải việc thực thi trực tiếp các lệnh, tức là thực hiện từng dòng một từ trên xuống dưới. Nhưng đôi khi chúng ta cần một số nhánh của luồng thực thi, nghĩa là chỉ thực hiện một số lệnh nếu một số điều kiện được đáp ứng. Câu lệnh if / else phức hợp được sử dụng cho những mục đích này .

Trong chủ đề này, bạn sẽ tìm hiểu cách kiểm soát luồng thực thi lệnh bằng cách sử dụng biểu thức Boolean và câu lệnh if / else .

**Toán tử so sánh (Toán tử Boolean)**

Chúng ta hãy tìm hiểu các biểu thức logic là gì. Theo biểu thức logic, chúng tôi muốn nói đến một câu lệnh đặt ra một số điều kiện. Ví dụ: thành ngữ “Bên ngoài trời đang mưa”. Biểu thức này có thể đúng hoặc sai. Tùy thuộc vào việc trời có mưa hay không mà người ta đưa ra quyết định xem có cần mang ô hay không. Nghĩa là, một biểu thức Boolean có thể đúng hoặc sai.

Bây giờ bạn đã hiểu biểu thức logic là gì, bạn cần học cách tạo chúng trong Python. Chúng ta có thể tạo biểu thức Boolean bằng cách sử dụng các toán tử so sánh.

Đúng như tên gọi, các toán tử này so sánh 2 giá trị và trả về True (nếu đúng) hoặc Sai (nếu sai).

Có một số loại toán tử quan hệ. Đầu tiên, hãy xem xét một trường hợp đơn giản:

• Bằng: == (lưu ý rằng bằng bao gồm hai dấu ==)

• Không bằng:! =

Các toán tử này so sánh hai phần tử và trả về Đúng hoặc Sai nếu chúng bằng nhau hoặc không.

Chúng ta có thể tạo biểu thức Boolean bằng cách so sánh hai giá trị bằng các toán tử sau:

>>>in (1 == 1)

ĐÚNG VẬY

>>>in(2 != 4)

ĐÚNG VẬY

>>> in(3 == 5)

SAI

>>> in ('7' == 7)

SAI

Trên đây là ví dụ về các biểu thức logic. Như bạn có thể thấy từ các ví dụ, kết quả của việc thực thi lệnh chứa biểu thức Boolean là Đúng hoặc Sai .

**Hãy chú ý!** Đúng và Sai là các kiểu dữ liệu Boolean riêng biệt. Cho đến nay, chúng ta chỉ xem xét kiểu chuỗi , kiểu int và kiểu float . Kiểu boolean được gọi là **bool** và chỉ chấp nhận 2 loại giá trị: True hoặc False . Xin lưu ý rằng các giá trị này được viết bằng chữ **in hoa** . Hãy nhớ lại rằng python phân biệt chữ hoa chữ thường và True ≠ true . Bạn có thể nhận thấy rằng khi bạn nhập chúng vào trình soạn thảo mã (với chữ in hoa T và F), chúng xuất hiện với màu khác với các biến hoặc chuỗi.

**Bài tập:**

Xác định xem các biểu thức Boolean sau đây là đúng hay sai.

Tuyên bố một:

( 6 \* 6 ) - 1 == 8 + 1

Tuyên bố thứ hai:

13 - 7 != ( 3 \* 2 ) + 1

Tuyên bố thứ ba:

3 \* ( 2 - 1 ) == 4 - 1

**Các toán tử logic lớn hơn và nhỏ hơn.**

Ở trên chúng ta đã xem xét 2 loại toán tử so sánh: bằng và không bằng. Ngoài chúng, còn có các toán tử so sánh khác:

• Hơn:>

• Nhỏ hơn: <

• Lớn hơn hoặc bằng: > =

• Nhỏ hơn hoặc bằng: <=

**Bài tập:**

Xác định xem các biểu thức Boolean sau đây là đúng hay sai.

( 6 \* 6 ) - 1 >= 8 + 1

Tuyên bố thứ hai:

13 - 7 <= ( 3 \* 2 ) + 1

Tuyên bố thứ ba:

3 \* ( 2 - 1 ) > 4 - 1

**Biến Boolean**

Vì Đúng và Sai là một kiểu dữ liệu đặc biệt riêng biệt, bool, nên, như trong trường hợp số và chuỗi, bạn có thể tạo các biến chứa giá trị đúng hoặc sai.

Đúng và Sai là các loại bool duy nhất và bất kỳ biến nào được gán một trong các giá trị này đều được gọi là biến boolean. Các biến Boolean có thể được tạo theo nhiều cách. Cách dễ nhất là chỉ cần gán Đúng hoặc Sai cho biến:

>>điều kiện đúng = ĐÚNG VẬY

>>điều kiện sai = SAI

Bạn có thể đặt một biến bằng biểu thức boolean.

>>bool\_condition\_1 = 5 != 7

>>bool\_condition\_2 = 1 + 1 != 2

>>bool\_condition\_3 = 3 \* 3 == 9

Các biến này hiện chứa các giá trị boolean nên khi được tham chiếu chúng sẽ chỉ trả về giá trị Đúng hoặc Sai của biểu thức mà chúng được gán.

>>>in (bool\_ba)

ĐÚNG VẬY

>>> in(bool\_four)

SAI

>>>in(bool\_5)

ĐÚNG VẬY

**Bài tập :**

1. Tạo một biến có tên bool\_variable và đặt nó thành true. Hãy thử in nó ra bàn điều khiển. Bạn đã gặp phải lỗi gì? Tại sao?

2. Thay thế giá trị trong bool\_variable bằng ' true ' (trong dấu ngoặc kép) Kiểm tra loại bool\_variable bằng cách sử dụng loại hàm kiểm tra loại ( bool\_variable).

In giá trị ra bàn điều khiển. Hiện tại bool\_variable thuộc loại nào?

3. Tại sao đây không phải là biến boolean!

Tạo một biến có tên bool\_variable\_2 và đặt biến đó thành kiểu boolean true.

4. Kiểm tra loại bool\_variable\_2 và đảm bảo rằng bạn đã tạo thành công biến boolean.

**câu lệnh if .**

Hiểu các biến và biểu thức Boolean rất quan trọng vì chúng là nền tảng khi xây dựng các câu lệnh kiểm tra bằng cách sử dụng các câu lệnh điều kiện.

Hãy quay lại ví dụ về mưa.

Nếu trời mưa thì bạn sẽ mang ô

Nếu dịch sang tiếng Anh, chúng ta nhận được câu lệnh sau:

Nếu trời mưa thì bạn hãy mang ô

Khi đó "trời sẽ mưa" là một biểu thức boolean và câu lệnh điều kiện này kiểm tra xem nó có đúng hay không.

Nếu "it is rain" == True thì phần còn lại của câu lệnh điều kiện sẽ được thực thi và bạn sẽ mang theo ô.

Đây thực sự là một dạng câu lệnh có điều kiện:

Nếu [trời mưa] thì [mang ô]

Câu lệnh if kiểm tra tính đúng đắn của biểu thức theo sau nó. Điểm hay của ngôn ngữ lập trình Python là nó rất gần với cấu trúc của ngôn ngữ tiếng Anh tự nhiên.

Trong Python, việc kiểm tra một điều kiện sẽ trông rất giống với ví dụ ô, được nêu bằng tiếng Anh:

>>nếu trời đang mưa:

>> mang\_ô()

Xin lưu ý rằng thay vì từ “then” sau điều kiện, dấu hai chấm “:” được sử dụng. Điều này được máy tính giải thích để tiếp theo sẽ có một tập hợp các hành động (lệnh) phải được thực hiện nếu điều kiện là đúng.

Hãy chú ý! Các hành động được thực hiện nếu điều kiện đúng phải được thụt lề và phải có cùng số lượng. Nghĩa là, chúng ta viết chính if từ đầu dòng và mọi thứ được thực thi nếu đúng đều có cùng mức thụt lề từ mép dòng. Ví dụ:

>> tuổi = 13

>> nếu tuổi >=18:

>> print ('Chào mừng đến với trang web sản phẩm rượu')

>> in ('kiểm tra hoàn thành')

Trong ví dụ này, chúng ta thấy biến tuổi được đặt , sau đó điều kiện được kiểm tra, tuổi lớn hơn hoặc bằng 18. Nếu điều này đúng, thì xuất dòng “Chào mừng đến với trang web sản phẩm rượu” ra bảng điều khiển. Tuy nhiên, biến tuổi bằng 13, điều kiện trong if sai, theo đó, nó sẽ không được in ra bàn điều khiển mà dòng “kiểm tra đã hoàn thành” sẽ được in, vì nó là một phần của mã chính và không phải bao gồm trong câu lệnh if . Điều này là do bản in cuối cùng được viết từ đầu dòng và không được thụt lề, tức là nó sẽ được thực thi trong mọi trường hợp, bất kể điều kiện được kiểm tra trong câu lệnh if . Kết quả thực thi đoạn mã này sẽ là:

kiểm tra đã hoàn thành

Hãy xem xét một tuyên bố có điều kiện khác:

>> nếu 10 == 5 \* 2 :

>> print ( "biểu thức đúng" )

Mã này có in biểu thức đánh giá là đúng cho thiết bị đầu cuối không? Có, vì điều kiện của câu lệnh if 10 == 5 \* 2 là đúng.

**Bài tập:**

Tôi làm việc cho một công ty chuyên cung cấp dịch vụ bảo mật thông tin cho các doanh nghiệp khác. Giám đốc một công ty quảng cáo nhỏ tiếp cận chúng tôi. Vấn đề chính là họ có một nhân viên bảo vệ, Dmitry, người cài đặt trò chơi máy tính trên máy trạm tự động của nhân viên khi họ không làm việc do trực (đi nghỉ) và chơi suốt đêm. Do đó, bạn được yêu cầu phát triển một ứng dụng kiểm tra thông tin xác thực đã nhập và phân biệt quyền của nhân viên. Họ yêu cầu nhân viên bảo vệ Dmitry đưa ra một thông báo chuyên biệt: “Dmitry, nơi làm việc của bạn ở trong một phòng khác. Hãy tránh xa máy tính của người khác và bắt tay vào làm việc!”

1. Hãy giới thiệu biến “ user\_name ”
2. Tiếp theo, hãy giới thiệu một biến hiển thị văn bản cho Dmitry. Hãy gọi nó là Dmitriy\_check
3. Hãy giới thiệu một biến lưu trữ tin nhắn cho các nhân viên khác đã đăng nhập: “Chào mừng”
4. câu lệnh if để kiểm tra giá trị của biến tên người dùng
5. Xuất ra bảng điều khiển kết quả thực thi chương trình cho người dùng \_ name = “Dmitry” và cho người dùng \_ name = “Angelina”.

Công ty chúng tôi tiếp tục phát triển ứng dụng bảo mật cho một công ty quảng cáo. Bây giờ, nếu người dùng nhập sai mật khẩu 3 lần thì hệ thống phải bị khóa. Để thực hiện việc này, hãy thực hiện các bước sau:

1. Hãy giới thiệu một biến ghi lại số lần nhập liệu và gọi nó là enter \_ number
2. Sử dụng câu lệnh if , chúng ta sẽ viết chương trình nếu nhập \_ số nhỏ hơn 3 thì viết “Thử lại. Bạn còn lại (3- nhập \_ số ) lần thử.” Nếu số lần thử lớn hơn hoặc bằng 3 thì chúng tôi hiển thị “Bạn đã vượt quá số lần thử tối đa. Tài khoản của bạn bị chặn. Để mở khóa, vui lòng liên hệ với bộ phận hỗ trợ."
3. Kiểm tra hoạt động của chương trình bằng đầu ra của bàn điều khiển.

**Toán tử logic: và**

Chúng tôi đã xem xét các toán tử so sánh. Chúng là một loại toán tử logic. Có nhiều loại toán tử logic khác.

Có những tình huống cần phải kiểm tra nhiều điều kiện cùng một lúc. Ví dụ, chúng ta chỉ mang theo ô nếu định đi ra ngoài trời mưa. Tức là phải có cùng lúc 2 điều kiện đúng:

1. Ngoài trời đang mưa
2. Tôi cần phải đi ra ngoài

Để xây dựng các điều kiện phức tạp hơn, các toán tử logic được sử dụng:

* Và
* hoặc
* không

Hãy bắt đầu với và .

và kiểm tra việc thực thi hai biểu thức logic cùng một lúc (logic và) và sẽ đánh giá là Đúng nếu cả hai biểu thức logic của nó đánh giá là Đúng và nếu ngược lại là Sai.

Hãy xem một ví dụ

Con gà là con chim và con gấu trúc là con vật.

Biểu thức Boolean này được tạo thành từ hai biểu thức nhỏ hơn: gà là một con chim và gấu trúc là một con vật, cả hai đều đánh giá là True và được kết nối bởi toán tử Boolean và , do đó toàn bộ biểu thức là đúng.

Hãy xem một ví dụ về toán tử some và trong Python :

>>> (1 + 1 == 2) và (2 + 2 == 4)

ĐÚNG VẬY

>>> (1 + 1 == 2) và (2 < 1)

SAI

>>> (1 > 9) và (5 != 6)

SAI

>>> (0 == 10) và (1 + 1 == 1)

SAI

Lưu ý rằng trong ví dụ thứ hai và thứ ba, ngay cả khi một phần của biểu thức đánh giá là Đúng thì toàn bộ biểu thức vẫn đánh giá là Sai vì toán tử còn lại đánh giá là Sai. Câu thứ tư cũng sai vì cả hai thành phần đều sai.

**Bài tập:**

1. Kiểm tra tính đúng đắn của các biểu thức sau:

(2 + 2 + 2 >= 6) và (-1 \* -1 < 0)

(4 \* 2 <= 8) và (7 - 1 == 6)

Đặt kết quả kiểm tra vào câu lệnh biến \_ one và câu lệnh \_ two .

1. Hãy quay lại vấn đề kiểm tra an ninh tại một công ty quảng cáo. Hãy làm phức tạp nhiệm vụ. Bây giờ bạn cần kiểm tra không chỉ tên người dùng mà còn cả số máy trạm.

Để thực hiện việc này, hãy nhập mã số công việc của 4 nhân viên của công ty này:

Máy trạm Dmitry số 1

Máy trạm Angelina số 2

Số Vasily AWP 3

Trạm làm việc của Ekaterina số 4.

1. Hãy giới thiệu biến ARM , biểu thị số lượng nơi làm việc được tự động hóa.
2. Hãy tạo một toán tử có điều kiện if , trong đó chúng tôi kiểm tra sự tương ứng giữa số máy trạm và tên người dùng:

Nếu số máy trạm và tên người dùng khớp nhau thì hiển thị “Chào mừng!” trên bảng điều khiển.

Nếu số máy trạm không khớp và tên người dùng không phải là Dmitry thì “Đăng nhập hoặc mật khẩu không chính xác, hãy thử lại”

Nếu Nếu số máy trạm không khớp và tên người dùng là Dmitry, thì đó là Dmitry, nơi làm việc của bạn ở một phòng khác. Hãy tránh xa máy tính của người khác và bắt tay vào làm việc!”

**Toán tử logic: hoặc**

tử logic or ngụ ý rằng ít nhất một điều kiện là đúng, nghĩa là nó là True nếu bất kỳ thành phần nào là True .

Hãy xem xét biểu thức:

Con gà là con chim hay con gấu trúc là con cá.

Câu này bao gồm hai cách diễn đạt: gà là chim, điều này đúng và gấu trúc là cá, điều này sai. Vì hai biểu thức được kết nối bằng toán tử hoặc nên toàn bộ toán tử đều đúng. Chỉ một thành phần phải đúng để toán tử hoặc là đúng.

Nếu toán tử hoặc có hai thành phần True thì nó cũng là True .

Chúng ta hãy xem một vài ví dụ trong Python :

>>> Đúng hoặc (3 + 4 == 7)

ĐÚNG VẬY

>>> (1 - 1 == 0) hoặc Sai

ĐÚNG VẬY

>>> (2 < 0) hoặc Đúng

ĐÚNG VẬY

>>> (3 == 8) hoặc (3 > 4)

SAI

Lưu ý rằng mọi câu lệnh hoặc có ít nhất một thành phần Đúng là Đúng, nhưng câu lệnh cuối cùng có hai thành phần Sai, vì vậy nó là Sai.

**Bài tập:**

1. Kiểm tra tính đúng đắn của các biểu thức sau:

(2 - 1 > 3) hoặc (-5 \* 2 == -10)

(9 + 5 <= 15) hoặc (7 != 4 + 3)

**biểu thức khác**

Như bạn có thể thấy từ công việc của mình về một vấn đề bảo mật thông tin, khi bạn bắt đầu đưa nhiều câu lệnh if vào một hàm , mã sẽ hơi lộn xộn và rắc rối. May mắn thay, chúng ta có thể sử dụng những công cụ khác để kiểm soát luồng thực thi.

Các câu lệnh khác cho phép bạn mô tả những gì cần phải làm khi các điều kiện **không được** đáp ứng.

Các câu lệnh else luôn xuất hiện cùng với câu lệnh if . Hãy xem ví dụ đánh thức của chúng tôi để xem nó hoạt động như thế nào:

>>nếu ngày trong tuần :

>> Wake\_up ( "6:30" )

>>khác :

>> ngủ\_trong ( )

Bằng cách này, chúng ta có thể tạo các câu lệnh if cho phép chúng ta chọn lệnh nào sẽ được thực thi nếu điều kiện đúng và lệnh nào nếu điều kiện sai. Điều này giúp chúng ta không phải viết các câu lệnh if cho mọi điều kiện có thể, thay vào đó, chúng ta có thể viết một câu lệnh else chung cho tất cả các trường hợp mà điều kiện không được đáp ứng.

Hãy quay lại ví dụ về trang web sản phẩm rượu. Hãy sửa lại mã để nếu độ tuổi dưới 18 hiển thị thông báo “Chúng tôi xin lỗi, chúng tôi không bán sản phẩm cho người dưới 18 tuổi”. Sau đó, mã của chúng tôi sẽ trông như thế này:

>> tuổi = 13

>> nếu tuổi >=18:

>> print ('Chào mừng đến với trang web sản phẩm rượu')

>> khác :

>> print ('Chúng tôi xin lỗi, chúng tôi không bán sản phẩm cho người dưới 18 tuổi')

Hãy chú ý đến những vết lõm! else nằm ở cùng cấp độ với if và các điều kiện được thực thi nếu điều kiện sai cũng được thụt vào từ đầu dòng (nghĩa là khối mã sẽ được thực thi nếu tuổi nhỏ hơn 18 được tô sáng) .

**Nhiệm vụ:**

Refactor (làm lại mã) trong bài toán về Dmitry và công ty quảng cáo sử dụng toán tử else .

**Khác câu lệnh if**

câu lệnh if và câu lệnh else , còn có câu lệnh elif trong Python .

Hãy tìm hiểu cách nó hoạt động. Giả sử chúng ta muốn đến quán cà phê và mua một ít bánh sandwich. Chúng tôi đến đó hàng ngày và biết rõ ở đó bán những gì và tất nhiên, chúng tôi đã có sở thích về hương vị. Món ưa thích của chúng tôi là bánh mì gà, thứ hai là bánh mì thịt bò và cuối cùng là bánh mì cá ngừ. Nếu thiếu tất cả những điều trên, chúng tôi sẽ lấy thứ khác. Vì vậy, hãy thử viết mã bằng câu lệnh if / else . Hãy giới thiệu một biến chứa giá trị của loại bánh sandwich còn trong kho tại cửa hàng.

>> sandwich\_in\_stock = 'sandwich\_

>>nếu sandwich\_in\_stock == ' s thịt gà ':

>> in ('Tôi sẽ mua bánh mì gà')

>>khác:

>> if sandwich\_in\_stock == ' s thịt bò ':

>> in ('Tôi sẽ mua bánh mì kẹp thịt bò')

>> khác:

>> if sandwich\_in\_stock == ' s cá ngừ ':

>> in ('Tôi sẽ mua bánh sandwich cá ngừ')

>> khác :

>> in ('Tôi sẽ lấy thứ khác)

Từ ví dụ này, rõ ràng là cấu trúc kết quả rất phức tạp và khó đọc. Nếu điều kiện sai thì các điều kiện bổ sung phải được kiểm tra bên trong câu lệnh else . Chúng được gọi là các câu lệnh lồng nhau. Xin lưu ý rằng toán tử càng lồng nhau thì độ thụt lề so với đầu dòng càng lớn.

Để tránh phải viết một câu lệnh if lồng nhau , Python đã giới thiệu một toán tử Elif đặc biệt .

Nghĩa là, câu lệnh Elif trông giống hệt như "else if". Câu lệnh elif kiểm tra một điều kiện khác sau điều kiện if trước đó, điều kiện này không được thỏa mãn.

Chúng ta có thể sử dụng câu lệnh Elif để kiểm soát thứ tự mà chương trình sẽ kiểm tra từng câu lệnh điều kiện. Câu lệnh if được kiểm tra trước, sau đó mỗi câu lệnh elif được kiểm tra từ trên xuống dưới và cuối cùng, mã else được thực thi nếu không có điều kiện nào trước đó được đáp ứng.

Hãy lấy mã bánh sandwich và làm lại nó bằng toán tử elif .

>>if sandwich\_in\_stock == 'với thịt gà':

>> in ('Tôi sẽ mua bánh mì gà')

>>elif sandwich\_in\_stock == 'với thịt bò':

>> in ('Tôi sẽ mua bánh mì kẹp thịt bò')

>>elif sandwich\_in\_stock == 'với cá ngừ':

>> in ('Tôi sẽ mua bánh sandwich cá ngừ')

>> khác :

>> in ('Tôi sẽ lấy thứ khác')

Hãy xem một ví dụ khác. Giả sử rằng một trang web đang được phát triển để quyên góp và tùy thuộc vào số tiền bạn quyên góp, bạn sẽ nhận được thông báo sau:

>> nếu quyên góp >= 1000 :

>> print ( "Cảm ơn bạn đã quyên góp! Bạn đã đạt được trạng thái quyên góp bạch kim!

>> yêu tinh quyên góp >= 500 :

>> print ( "Cảm ơn bạn đã quyên góp! Bạn đã đạt được trạng thái quyên góp vàng!" )

>> yêu tinh quyên góp >= 100 :

>> print ( "Cảm ơn bạn đã quyên góp! Bạn đã đạt được trạng thái quyên góp bạc!" )

>> khác :

>> print ( "Cảm ơn bạn đã quyên góp! Bạn đã đạt được trạng thái quyên góp đồng!" ) " )

Hãy suy nghĩ về tính năng này trong một giây. Điều gì sẽ xảy ra nếu tất cả các câu lệnh Elif chỉ là câu lệnh if ? Nếu 1000 được quyên góp thì tất cả ba tin nhắn đầu tiên sẽ được in ra bảng điều khiển vì từng điều kiện đều được đáp ứng.

Nhưng vì chúng ta sử dụng câu lệnh Elif nên nó sẽ kiểm tra tuần tự từng điều kiện và chỉ in một thông báo. Nếu tôi quyên góp 600,00, trước tiên mã sẽ kiểm tra xem nó có lớn hơn 1000,00 hay không, sau đó nó sẽ kiểm tra xem nó có lớn hơn 500,00 hay không, vì vậy nó sẽ hiển thị một thông báo tương ứng, sau đó vì tất cả các câu lệnh khác đều được kiểm tra chỉ khi giá trị trước đó sai thì chúng không được kiểm tra và thông báo không còn xuất ra nữa.

**Bài tập:**

Trường đại học có một hệ thống chấm điểm, được ấn định cho sinh viên tùy thuộc vào điểm trung bình của họ. Bạn được yêu cầu viết một đơn đăng ký hiển thị điểm thích hợp cho từng học sinh theo các điều kiện sau:

• 4.0 trở lên sẽ trả về "A"

• 3.0 trở lên sẽ trả về "B"

• 2.0 trở lên sẽ trả về "C"

• 1.0 hoặc cao hơn sẽ trả về "D"

• 0,0 hoặc cao hơn sẽ trả về "F"

Để giải quyết vấn đề này, hãy sử dụng một biến cấp độ và sử dụng toán tử elif .

**Câu lệnh Thử và Ngoại trừ**

Các câu lệnh if , elif và else không phải là cách duy nhất để nhúng luồng lệnh vào chương trình. Có các câu lệnh thử và ngoại trừ để kiểm tra các lỗi có thể xảy ra mà người dùng có thể gặp phải.

Cú pháp chung cho câu lệnh thử và ngoại trừ là:

>>thử :

>> # vài câu nói

>>ngoại trừ Tên lỗi :

>> # một số tuyên bố

Đầu tiên, câu lệnh try sẽ được thực thi . Nếu tại bất kỳ thời điểm nào trong quá trình thực thi này xảy ra một ngoại lệ như NameError hoặc ValueError và ngoại lệ đó khớp với từ khóa trong câu lệnh ngoại trừ, thì câu lệnh try sẽ hoàn thành và câu lệnh ngoại trừ sẽ được thực thi.

Trong thực tế, lỗi có thể được kiểm tra bằng cách sử dụng if / else .

Có 2 phương pháp điều khiển luồng:

EAFP (dễ xin sự tha thứ hơn là xin phép, dịch : xin sự tha thứ dễ hơn là xin phép). Trong cách tiếp cận này, chúng tôi sử dụng cấu trúc try / trừ . Trong trường hợp này, khi chương trình chạy, trước tiên nó sẽ cố gắng thực hiện một điều gì đó (ví dụ: chia) và nếu xảy ra lỗi, mã khác sẽ được thực thi.

Hãy xem xét điều này với một ví dụ. Bạn cần viết mã chia a cho b. Nhưng có khả năng b là null, điều này sẽ gây ra lỗi nên bạn cần thêm try và ex để bắt lỗi này.

>>thử :

>> kết quả = Một / b

>> in ( kết quả )

>> ngoại trừ ZeroDivisionError :

>> in ( " không được phép chia cho 0 " )

LBYL (Nhìn trước khi nhảy, dịch: nhìn trước khi nhảy). Cách tiếp cận này trước tiên sẽ kiểm tra xem các giá trị có hợp lệ hay không và sau đó thực thi mã.

Một ví dụ về phép chia sẽ như thế này:

>>nếu b!=0:

>> kết quả = a/b

>> in(kết quả)

>> khác :

>> print ('không được phép chia cho 0')

**Tuân thủ cấu trúc mẫu ( Structure mẫu lập bản đồ )**

**xây dựng trận đấu \ trường hợp**

Python 3.10 đã giới thiệu cấu trúc match \ case . Câu lệnh match tương tự như câu lệnh switch được tìm thấy trong nhiều ngôn ngữ lập trình khác và cho phép bạn thực thi một lệnh cụ thể tùy thuộc vào việc mẫu có khớp hay không. Việc đưa toán tử này vào ngôn ngữ Python giúp tăng khả năng đọc mã với nhiều tùy chọn về mẫu cho một đối tượng.

Cú pháp chung của cấu trúc match/case là:

**>>** chủ đề phù hợp:

>> trường hợp <mẫu\_1>:

>> <hành động\_1>

>> trường hợp <mẫu\_2>:

>> <hành động\_2>

>> trường hợp <mẫu\_3>:

>> <hành động\_3>

>>trường hợp\_:

>> <action\_wildcard>

Ví dụ: hãy quay lại ứng dụng xin chào của chúng tôi thế giới . Bây giờ chúng tôi muốn tăng chức năng của ưu đãi này và hiển thị lời chào tùy thuộc vào ngôn ngữ được chọn. Điều này có thể được thực hiện bằng cách sử dụng cấu trúc if / else / elif . Ví dụ:

>>nếu ngôn ngữ == 'tiếng Nga':

>> print('Xin chào thế giới')

>>ngôn ngữ Elif == 'tiếng Anh':

>> print(' Xin chào thế giới')

>>ngôn ngữ Elif == 'deutsch':

>> print(' Xin chào Welt')

>>khác:

>> in('ngôn ngữ không xác định')

Với cấu trúc match \ case, mã sẽ trông như thế này:

>>ngôn ngữ phù hợp:

>> trường hợp "tiếng Nga":

>> print("Xin chào thế giới")

>> trường hợp "tiếng anh":

>> in("Xin chào")

>> trường hợp "Đức":

>> print("Xin chào chào mừng")

>> trường hợp\_ :

>> in (" không xác định ngôn ngữ ")

Lưu ý rằng các khối chữ được thụt lề từ đầu mệnh đề khớp . Và hướng dẫn của mỗi khối trường hợp được thụt lề từ đầu khối trường hợp nhất định .

**Bài tập**

Trường đại học có một hệ thống chấm điểm, được ấn định cho sinh viên tùy thuộc vào điểm trung bình của họ. Bạn được yêu cầu viết một đơn đăng ký hiển thị điểm thích hợp cho từng học sinh theo các điều kiện sau:

• 4.0 trở lên sẽ trả về "A"

• 3.0 trở lên sẽ trả về "B"

• 2.0 trở lên sẽ trả về "C"

• 1.0 hoặc cao hơn sẽ trả về "D"

• 0,0 hoặc cao hơn sẽ trả về "F"

Để giải, hãy nhập biến điểm và sử dụng toán tử match \ case

**Kết luận**

Trong phần này, chúng ta đã xem xét nhiều công cụ ngôn ngữ Python mới cho phép chúng ta thêm logic vào mã của mình. Hãy tóm tắt. Trong khuôn khổ phần này, những điều sau đây đã được xem xét:

• Biểu thức Boolean là các câu lệnh có thể đánh giá là Đúng hoặc Sai.

• Biến Boolean là biến được đặt thành Đúng hoặc Sai.

• Bạn có thể tạo các biểu thức Boolean bằng cách sử dụng các toán tử so sánh:

o Bằng: ==

o Không bằng:! =

o Hơn:>

o Lớn hơn hoặc bằng: > =

o Nhỏ hơn: <

o Nhỏ hơn hoặc bằng: <=

• Câu lệnh if có thể được sử dụng để tạo luồng điều khiển trong mã của bạn.

• Câu lệnh else có thể được sử dụng để thực thi mã khi các điều kiện của câu lệnh if không được đáp ứng.

• Câu lệnh Elif có thể được sử dụng để xây dựng các thử nghiệm bổ sung vào câu lệnh if.

• Câu lệnh thử và ngoại trừ có thể được sử dụng để xây dựng tính năng kiểm soát lỗi trong mã của bạn.

• Câu lệnh so khớp và trường hợp có thể được sử dụng để kiểm tra xem một mẫu có khớp hay không.

Chức năng

Ngôn ngữ Python có tính năng thực thi mã theo từng dòng. Ví dụ: nếu người theo dõi của chúng tôi muốn xuất tên và tuổi của một người vào bảng điều khiển, chúng tôi sẽ làm như sau:

>>tên = ' Anna '

>>tuổi = 18

>>in(tên)

>> in ( tuổi )

Giả sử chúng ta muốn hiển thị nhiều người khác nhau với tên và độ tuổi khác nhau. Những gì sẽ cần phải được thực hiện trong trường hợp này? Liên tục lặp lại các dòng mã trên nhiều lần. Ví dụ :

>>tên = ' Anna '

>>tuổi = 18

>>in(tên)

>>in(tuổi)

>>tên = ' Alexander '

>>tuổi = 19

>>in(tên)

>>in(tuổi)

>>tên = ' Oleg '

>>tuổi = 18

>>in(tên)

>>in(tuổi)

Càng nhiều dữ liệu về những người khác nhau mà chúng ta muốn xuất ra bảng điều khiển thì càng có nhiều dòng giống nhau xuất hiện trong mã. Mã trở nên lộn xộn và không thể đọc được. Làm cách nào chúng ta có thể làm cho các thao tác cùng loại có thể được viết một lần và sau đó được thực hiện đơn giản với các dữ liệu khác nhau? (trong ví dụ, thao tác gán tên cho các biến và xuất chúng ra bảng điều khiển và các tham số chính là tên (Anna, Alexander, Oleg) và độ tuổi (18,19,18))

Để giải quyết những vấn đề như vậy , Python sử dụng các đối tượng đặc biệt gọi là hàm.

Bài học này sẽ bao gồm:

* chức năng là gì
* Các phương pháp xác định hàm
* Lời gọi hàm
* Các loại tham số chức năng

**Khái niệm chức năng**

Hãy quay lại ví dụ của chúng tôi với tên. Để tách mã này thành một cấu trúc riêng sẽ kích hoạt mỗi khi chúng ta muốn in tên và tuổi của một người mới vào bảng điều khiển, chúng ta có thể tạo một hàm và gọi nó là in \_name\_ và \_age ( ) và gọi mỗi lần nếu cần, hãy hiển thị thông tin về tên và tuổi của những người khác nhau trên bảng điều khiển.

**Cách xác định chức năng**

Để tạo một hàm bạn phải sử dụng từ khóa def . Tiếp theo bạn cần đặt tên hàm. Các quy tắc tương tự áp dụng cho tên hàm cũng như tên biến. Sau tên hàm có dấu ngoặc đơn và sau dấu ngoặc đơn có dấu hai chấm. Hãy quay lại ví dụ của chúng ta với kết quả đầu ra là tên và tuổi cho bảng điều khiển. Hãy tạo một hàm in \_ name \_ và \_ age (), hàm này sẽ xuất các giá trị chúng ta cần ra console:

>>tên = ' Anna '

>>tuổi = 18

>>indef\_name\_and\_age():

>> in ( tên )

>> in ( tuổi )

Như bạn có thể thấy từ ví dụ, phần mã có trong hàm được thụt vào từ mép dòng (như trường hợp của if - else ). Để mã trong một hàm được thực thi, nó phải được gọi. Để gọi một hàm bạn cần viết tên của nó. Trong ví dụ của chúng tôi:

>>tên = ' Anna '

>>tuổi = 18

>>indef\_name\_and\_age():

>> in(tên)

>> in(tuổi)

>> print\_name\_and\_age()

Dòng cuối cùng là lời gọi hàm. Nếu bạn không gọi hàm thì mã bên trong nó (mọi thứ được thụt vào sau dấu hai chấm) sẽ không được thực thi. Sử dụng các hàm, mã của chúng tôi sẽ trở thành như sau:

>> chắc chắn print\_name\_and\_age ( ) :

>> in(tên)

>> in(tuổi)

>>tên = ' Anna '

>>tuổi = 18

>> print\_name\_and\_age()

>>tên = ' Alexander '

>>tuổi = 19

>> print\_name\_and\_age()

>>tên = ' Oleg '

>>tuổi = 18

>> print\_name\_and\_age()

Như bạn có thể thấy, chúng tôi đã giảm số dòng lặp lại. Tuy nhiên, chúng tôi không muốn gán giá trị cho các biến tên và tuổi mỗi lần mà chỉ gọi nó với các tham số khác nhau.

**Thông số đầu vào chức năng**

Python có khả năng thiết lập tham số đầu vào cho hàm. Các tham số đầu vào được viết trong ngoặc đơn sau tên hàm và cách nhau bằng dấu phẩy. Hãy quay lại ví dụ của chúng tôi. Chúng tôi muốn chức năng def in \_ name \_ và \_ age () lấy tên và tuổi của mọi người làm tham số đầu vào, sau đó in chúng ra bảng điều khiển. Sử dụng các tham số của hàm, nó sẽ trông như thế này:

>> chắc chắn in \_ tên \_ và \_ tuổi ( name , age ):

>> in ( tên )

>> in ( tuổi )

Giá trị của các tham số này sẽ được đặt khi gọi hàm bên trong dấu ngoặc. Để xuất dữ liệu từ ví dụ trước, mã sẽ trông như thế này:

>>def print\_name\_and\_age(tên, tuổi):

>> in(tên)

>> in(tuổi)

>>printing\_name\_and\_age(' Anna ', 18)

>>printing\_name\_and\_age(' Alexander ', 19)

>>printing\_name\_and\_age(' Oleg ', 18)

Sau đây sẽ được xuất ra bàn điều khiển:

Anna

18

Alexander

19

Oleg

18

Như có thể thấy từ ví dụ, khi gọi hàm trong ngoặc đơn, chúng ta đã chỉ ra các tham số được thay thế vào biến tương ứng.

Nhưng làm sao chương trình biết nên đặt tham số nào ở đâu? Trên thực tế, các giá trị đã được thay thế vào các tham số của hàm theo thứ tự chúng xuất hiện. Vì khi tạo hàm, trước tiên chúng ta viết tên trong dấu ngoặc đơn , sau đó là tuổi , rồi 'Anna' được thay thế vào tham số tên , như được chỉ ra đầu tiên trong dấu ngoặc đơn khi gọi hàm. Nếu lệnh gọi hàm bị thay đổi, nghĩa là các tham số 'Anna' và 18 sẽ được hoán đổi như sau:

>>def print\_name\_and\_age(tên, tuổi):

>> in(tên)

>> in(tuổi)

>>printing\_name\_and\_age(18, ' Anna ')

Đầu ra sẽ trông như thế này:

18

Anna

Như bạn có thể thấy, giá trị 18 được ghi vào name và giá trị 'Anna' được ghi vào age . Nghĩa là, các giá trị tham số phải được liệt kê trong lệnh gọi hàm theo thứ tự mà chính các tham số đó đã được chỉ định khi hàm được tạo. Vì giá trị của các tham số khi gọi hàm được đặt tùy thuộc vào vị trí của các tham số khi tạo hàm nên các tham số đó được gọi là vị trí.

Để gọi một hàm, bạn phải viết trong ngoặc đơn các giá trị cho tất cả các tham số có trong hàm. Trong ví dụ của chúng tôi, hàm chứa 2 tham số name và age . Điều gì xảy ra nếu bạn chỉ đặt giá trị cho một trong các tham số? Ví dụ,

>> in \_ tên \_ và \_ tuổi ('Anna')

Trong trường hợp này, một thông báo lỗi sẽ được hiển thị:

TypeError: print\_name\_and\_age() thiếu 1 đối số vị trí bắt buộc: 'age'

**Thông số chính**

Giả sử rằng chúng tôi hiển thị thông tin về sinh viên năm thứ nhất trên bảng điều khiển. Chúng tôi biết rằng hầu hết học sinh đều 18 tuổi. Chỉ có 3 người ở độ tuổi khác nhau. Phải làm gì trong tình huống này?

Đối với những trường hợp như vậy, có những tham số có giá trị mặc định. Tức là chúng ta có thể đặt một giá trị sẽ được thay thế vào tham số tương ứng nếu chúng ta không chỉ định giá trị đó khi gọi hàm. Các đối số có giá trị mặc định gọi là đối số khóa

Trong phần của chúng tôi, với tên và tuổi, bạn có thể làm điều này:

>>indef\_name\_and\_age(tên, tuổi=18):

>> in(tên)

>> in(tuổi)

>>printing\_name\_and\_age(' Anna ')

Sau đó, đầu ra sẽ là:

Anna

18

Tức là các bạn có thể thấy khi gọi hàm chúng ta không chỉ định tuổi nhưng khi tạo hàm giá trị mặc định age = 18 đã được chỉ định nên không có lỗi hiển thị và giá trị 18 được xuất ra console là age.

Nếu chúng ta muốn tuổi là một số khác với giá trị mặc định thì chúng ta phải chỉ định nó khi gọi hàm. Ví dụ: để hiển thị dữ liệu của Alexander, 19 tuổi, kết quả đầu ra sẽ như sau:

>>printing\_name\_and\_age(' Alexander ', 19)

Sau đó, tất cả mã nguồn của chúng tôi sẽ trông như thế này:

>>indef\_name\_and\_age(tên, tuổi=18):

>> in(tên)

>> in(tuổi)

>>printing\_name\_and\_age(' Anna ')

>>printing\_name\_and\_age(' Alexander ', 19)

>>printing\_name\_and\_age(' Oleg ')

Cần phải nhớ rằng các đối số chính có giá trị mặc định luôn được chỉ định ở cuối, nghĩa là sau các đối số vị trí. Đó là lý do tại sao Kế tiếp mã số :

>>indef\_name\_and\_age(tuổi=18, tên):

>> in(tên)

>> in(tuổi)

>>printing\_name\_and\_age(' Anna ')

Sẽ phát hành lỗi :

Cú phápError: đối số không mặc định tuân theo đối số mặc định

**Bài tập**

1. Xác định hàm có tên create\_s Spreadsheet() nhận một tiêu đề đối số và chỉ in một dòng "Tạo bảng tính có tên" + tiêu đề ra bảng điều khiển.

2. Gọi create\_s Spreadsheet với giá trị tiêu đề "Tải xuống".

3. Thêm tham số row\_count vào định nghĩa hàm. Đặt giá trị mặc định thành 1000.

4. Sửa đổi câu lệnh in "Tạo bảng tính có tên là tiêu đề bằng các dòng row\_count" trong đó tiêu đề và row\_count được thay thế bằng các giá trị tương ứng của chúng.

Hãy nhớ rằng, để nối một số thành một đối tượng chuỗi, trước tiên bạn cần truyền row\_count thành một chuỗi bằng cách sử dụng str(). Nếu không bạn sẽ nhận được TypeError.

5. Gọi create\_s Lansheet() với tên được đặt trong Ứng dụng và row\_countset trong 10.

**Giá trị trả về**

Bài học này sẽ giải thích cách sử dụng kết quả của phép tính hàm để tiếp tục sử dụng chúng cho các phép tính.

**tuyên bố trở lại**

Cho đến nay chúng ta chỉ thấy các hàm in một số kết quả ra bàn điều khiển. Các hàm cũng có thể trả về một giá trị cho người dùng để giá trị đó có thể được thay đổi hoặc sử dụng sau này. Khi kết quả thực thi hàm có thể được lưu trữ trong một biến, nó được gọi là giá trị trả về của hàm. Chúng ta sử dụng từ khóa return để trả về giá trị của hàm.

Giả sử chúng ta có hai chức năng. Số thứ nhất cộng 2 số, số thứ hai lấy số thứ nhất trừ số thứ hai. Các chức năng này được thực hiện như sau:

>> chắc chắn tổng hợp ( a , b ):

>> kết quả = a + b

>>in(kết quả)

>>trừ def (a, b):

>>kết quả = ab

>>in(kết quả)

Giả sử chúng ta cần tính biểu thức sau: 2+3-1. Như bạn có thể thấy từ công thức, chúng tôi sử dụng cả phép cộng và phép trừ một cách tuần tự. Hãy gọi hàm cộng đầu tiên và thử gán giá trị cho một biến:

>> x =2

>>y=3

>>z=1

>>result\_sum = tổng(x, y)

>> in ( kết quả \_ tổng )

Bảng điều khiển sẽ hiển thị:

5

Không có

Ta thấy kết quả \_ sum không chứa kết quả của phép tính tổng mà chứa None . Làm cách nào để đưa kết quả của phép tính tổng vào biến result \_ sum ? Đối với điều này, chúng tôi sử dụng câu lệnh return .

>>tổng def(a, b):

>>kết quả = a+b

>>trả về kết quả

>>trừ def (a, b):

>>kết quả = ab

>> trả về kết quả

>>x=2

>>y=3

>>z=1

>>result\_sum = tổng(x, y)

>> in ( kết quả \_ tổng )

Kết quả sẽ là:

5

Nghĩa là, câu lệnh return cho phép chúng ta đặt giá trị của kết quả tính tổng vào biến kết quả \_ sum . Tại sao bạn không thể gọi biến kết quả từ một hàm? Điều này xảy ra do phạm vi.

**phạm vi**

Giả sử chúng ta có hàm từ bài tập trước tạo một chuỗi về một phần tử đặc biệt:

>>def create\_special\_string(special\_item):

>>return "Đặc biệt của chúng tôi là " + Special\_item + "."

Điều gì sẽ xảy ra nếu chúng ta muốn truy cập biến \_ item đặc biệt bên ngoài hàm? Chúng ta có thể sử dụng nó không ?

>>def create\_special\_string(special\_item):

>>return "Đặc biệt của chúng tôi là " + Special\_item + "."

>>print("Tôi không thích " + Special\_item)

Nếu cố chạy mã này, chúng tôi sẽ nhận được NameError , cho chúng tôi biết rằng ' special\_item ' không được xác định . Biến Special\_item chỉ được xác định trong không gian hàm, vì vậy nó không tồn tại bên ngoài hàm. Phạm vi của mục \_ đặc biệt chỉ là hàm tạo chuỗi \_ đặc biệt .

Các biến được xác định bên ngoài phạm vi của hàm có thể được truy cập bên trong thân hàm:

>>header\_string = "Đặc biệt của chúng tôi là "

>>def create\_special\_string(special\_item):

>>trả về chuỗi\_tiêu đề + mục đặc biệt + "."

>>print(create\_special\_string("nho"))

Không có sai lầm ở đây. header\_string có thể được sử dụng bên trong hàm create\_special\_string vì phạm vi của header\_string là toàn bộ tệp. Tập tin này sẽ xuất ra:

Của chúng tôi đặc biệt là quả nho .

**Bài tập**

1. Hàm xác định\_age tạo ra một biến gọi là tuổi, là sự khác biệt giữa năm hiện tại và năm sinh, cả hai đều là dữ liệu đầu vào của hàm. Thêm một dòng để trả về tuổi dưới dạng kết quả tính toán để sử dụng sau này.

>>def calc\_age (năm hiện tại, năm sinh):

#tuổi = năm\_hiện tại - năm\_sinh

2. Bên ngoài hàm, gọi Calculate\_age với các giá trị 2049 (current\_year) và 1993 (birth\_year) và lưu giá trị vào biến my\_age.

3. Gọi Calculate\_age với các giá trị 2049 (current\_year) và 1953 (birth\_year) và lưu giá trị vào một biến có tên là Dads\_age.

In ra bảng điều khiển dòng “Tôi X tuổi và bố tôi Y tuổi,” với my\_age, trong đó X và Dads\_age, trong đó Y.

**Nhiều giá trị trả về**

Đôi khi bạn có thể cần trả về nhiều giá trị từ một hàm. Chúng ta có thể trả về nhiều giá trị bằng cách phân tách chúng bằng dấu phẩy:

>> chắc chắn điểm \_ vuông ( x \_ value , y \_ value ):

>> x\_2 = x\_giá trị \* x\_giá trị

>> y\_2 = y\_value \* y\_value

>> trả về x\_2, y\_2

Hàm này lấy giá trị x và giá trị ay rồi trả về cả hai bình phương. Chúng ta có thể nhận được các giá trị này bằng cách chỉ định chúng khi gọi hàm:

>>x\_squared, y\_squared = Square\_point(1, 3)

>>in(x\_squared)

>>in(y\_squared)

Bảng điều khiển sẽ hiển thị:

1

9

**Bài tập**

1. Viết hàm get\_boundaries() nhận vào hai tham số: tham số đích dạng số và tham số lề dạng số.

Bạn nên tạo hai biến:

• low\_limit : mục tiêu trừ tiền ký quỹ .

• high\_limit: thêm ký quỹ nhắm mục tiêu

2.Trả về các giá trị low\_limit và high\_limit từ hàm theo thứ tự đã chỉ định.

3. Gọi hàm với mục tiêu là 100 và lề là 20. Lưu trữ các giá trị trả về trong các biến có tên low\_limit và high\_limit .

4.Xuất dòng sau vào bảng điều khiển:

Giới hạn dưới : low\_limit , giới hạn trên : high\_limit

Với các giá trị thấp và cao bạn nhận được từ hàm get\_boundaries().

**Bài tập**

1. Xác định hàm có tên là lặp lại\_stuff nhận hai đầu vào, nội dung và num\_repeat.

Chúng tôi muốn hàm này in ra bảng điều khiển một chuỗi có số lần lặp lại nre\_repeat. Hiện tại, chỉ cần đặt một câu lệnh in trống bên trong hàm .

1. Bên ngoài hàm, hãy gọi loop\_stuff.

Bạn có thể sử dụng giá trị "Hàng" cho nội dung và 3 cho num\_repeat.

1. Thay đổi câu lệnh đầu ra bên trong loop\_stuff thành câu lệnh return.

Nó sẽ trả về nội dung \* num\_repeat.

Lưu ý: nhân một chuỗi chỉ lặp lại giá trị của chuỗi đó! Ví dụ:

" à " \* 6

kết quả là chuỗi "nananananana".

4. Đặt tham số num\_repeat về giá trị mặc định là 10.

5. Nối kết quả của lặp lại \_ thứ (" Row ", 3) và chuỗi " Your Boat ." cùng nhau và lưu kết quả vào một biến gọi là lời bài hát.

6. Tạo một biến có tên là song và gán cho nó giá trị lặp lại\_stuff, chỉ được gọi với nội dung.

7. Xuất bài hát ra bảng điều khiển.

Danh sách

Đôi khi cần kết hợp nhiều đối tượng thành một cấu trúc. Ví dụ: khi bạn đến cửa hàng và họ đưa cho bạn danh sách các mặt hàng cần mua. Sẽ thuận tiện như thế nào khi kết hợp việc mua hàng vào một đối tượng duy nhất?

Phần này sẽ tìm hiểu:

* Các loại tuần tự là gì?
* Danh sách là gì và cách làm việc với nó?
* Phương pháp xử lý danh sách

**Danh sách là**

Vì vậy, giả sử bạn viết ra những sản phẩm bạn định mua. Danh sách của bạn trông giống như:

1. Bánh mì 1 miếng
2. Trứng 10 cái
3. Sữa 1 miếng
4. Cà phê 1 miếng

Tên của sản phẩm trong trường hợp này sẽ là một chuỗi và số lượng sản phẩm bạn định mua sẽ là một con số. Đến đây chúng ta đã học cách định nghĩa các biến. Sử dụng kiến thức này, danh sách mua sắm của bạn sẽ trông như thế này:

>> mục 1 = 'bánh mì'

>> mục 2 = 'trứng'

>> mục 3 = 'sữa'

>> mục 4 = 'cà phê'

Nếu bạn cũng viết số lượng riêng biệt, thì đây sẽ là một chuỗi các biến trong đó các giá trị của số biểu thị số lượng sản phẩm cần thiết sẽ được ghi.

Tuy nhiên, điều này thật bất tiện. Sẽ thuận tiện hơn nếu kết hợp toàn bộ danh sách sản phẩm lại với nhau và viết nó thành một biến. Có một kiểu dữ liệu danh sách cho việc này . Danh sách là kiểu dữ liệu tuần tự (nghĩa là chúng lưu trữ một chuỗi các phần tử) và là tập hợp các phần tử riêng lẻ được phân tách bằng dấu phẩy. Danh sách được biểu thị bằng dấu ngoặc vuông [ ]. Đối với ví dụ của chúng tôi về việc mua hàng, danh sách sẽ trông như sau:

>> danh sách mua sắm = ['bánh mì', 'trứng', 'sữa', 'cà phê']

Như chúng ta có thể thấy từ ví dụ, danh sách mua sắm \_ bao gồm một tập hợp các dòng cho biết tên của từng sản phẩm trong danh sách mua sắm. Tuy nhiên, không chỉ cần ghi tên hàng hóa mà còn phải ghi cả số lượng của chúng. Có thể tạo một danh sách riêng:

>> số lượng = [1, 10, 1, 1]

Trước đây, chúng ta đã xem xét hai danh sách bao gồm các đối tượng có cùng kiểu dữ liệu. Danh sách mua sắm \_ chỉ chứa các chuỗi và số lượng chỉ chứa số nguyên. Bạn có thể nhớ lại rằng python là một ngôn ngữ được gõ động. Do đó, một danh sách có thể chứa các phần tử thuộc nhiều loại khác nhau:

>>bánh mì = [ 'bánh mì' , 1]

**Bài tập**

1. Tạo danh sách chứa tên sản phẩm “bánh” và số 1.

**Danh sách danh sách**

Chúng tôi phát hiện ra rằng các thành phần trong danh sách có thể thuộc bất kỳ loại nào. Thực tế, một danh sách có thể chứa các danh sách khác.

Hãy quay lại danh sách mua sắm của chúng tôi. Chúng tôi muốn tên hàng hóa và số lượng yêu cầu của chúng được lưu trữ trong danh sách. Với những mục đích này, bạn có thể tạo một danh sách sẽ bao gồm các danh sách sau:

>> shopping\_list = [['bánh mì', 1], ['trứng', 10], ['sữa', 1], ['cà phê', 1 ] ]

**Bài tập**

1. Họ viết cho chúng tôi một danh sách bổ sung các sản phẩm cần mua ở quầy hóa chất gia dụng. Gồm 1 miếng bột giặt và 1 miếng nước rửa chén. Tạo một danh sách hộ gia đình mới, trong đó sẽ chứa một danh sách bao gồm các danh sách trong đó phần tử đầu tiên là tên sản phẩm và phần tử thứ hai là số lượng.

**Phương pháp nén**

Hãy quay lại danh sách mua sắm:

1. Bánh mì 1 miếng
2. Trứng 10 cái
3. Sữa 1 miếng
4. Cà phê 1 miếng

Giả sử rằng chúng ta có hai danh sách riêng biệt, một danh sách bao gồm tên hàng hóa, danh sách thứ hai - số lượng của chúng:

>> danh sách mua sắm = ['bánh mì', 'trứng', 'sữa', 'cà phê']

>> số lượng = [1, 10, 1, 1]

Nếu chúng ta muốn tạo danh sách các danh sách với từng mục cùng với số lượng được yêu cầu, chúng ta có thể sử dụng lệnh zip. zip lấy hai (hoặc nhiều) danh sách làm đầu vào và trả về một đối tượng chứa danh sách các cặp (bộ dữ liệu). Mỗi cặp chứa một phần tử từ mỗi danh sách đầu vào. Tuy nhiên, lệnh print sẽ không hiển thị cho chúng ta kết quả thực thi hàm zip .

>> shopping\_list\_with\_quantity = zip(shop\_list, số lượng)

>>in(shopping\_list\_with\_quantity)

Kết quả của việc thực thi đoạn mã trên, kết quả sau sẽ được xuất ra bàn điều khiển:

< zip sự vật tại 0 x 7 f 1631 e 86 b 48>

Dòng này cho biết vị trí của đối tượng zip đã tạo trong bộ nhớ. Tuy nhiên, chúng tôi có một hoạt động chuyển đổi loại. Trước đây, chúng ta đã gặp các hàm như str (), dùng để chuyển đổi kiểu dữ liệu thành chuỗi. Chức năng tương tự tồn tại để chuyển đổi một kiểu dữ liệu thành một danh sách. Hàm này là danh sách (). Trong ví dụ của chúng tôi, chúng tôi có thể sử dụng hàm này để xem kết quả của hàm zip :

>> shopping\_list\_with\_quantity = zip(shop\_list, số lượng)

>>print(danh sách(shopping\_list\_with\_quantity))

Kết quả thực thi đoạn mã này sẽ là:

[('bánh mì', 1), ('trứng', 10), ('sữa', 1), ('cà phê', 1)]

**Bài tập**

1. Sử dụng zip để tạo một biến mới gọi là tên \_ và \_ chó \_ tên kết hợp tên và chó thành một đối tượng zip .

Names=['Ben', 'Holly', 'Ann']

dogs\_names= ['Sharik', 'Gab', 'Beethoven']

1. Tạo nên mới biến Với đặt tên list\_of\_names\_and\_dogs\_names bằng cách gọi list() trên tên\_and\_dogs\_names. In biến mới ra bàn điều khiển.

**Danh sách trống**

Danh sách không cần phải chứa bất cứ điều gì! Bạn có thể tạo một danh sách trống như thế này:

>> danh sách trống = []

Tại sao cần có danh sách trống?

Thông thường, các danh sách như vậy được tạo ra với mục đích điền vào sau này dựa trên dữ liệu mới.

**phương pháp nối thêm**

Phương pháp này cho phép bạn thêm các phần tử vào cuối danh sách. Ví dụ: chúng tôi cũng cần mua một thanh sô cô la và chúng tôi muốn phản ánh điều này vào danh sách mua sắm. Để làm điều này chúng ta sẽ làm như sau:

>> danh sách mua sắm = ['bánh mì', 'trứng', 'sữa', 'cà phê']

>> shopping\_list.append(' sô cô la ')

>>in (danh sách mua sắm)

Mã này sẽ xuất ra như sau:

['bánh mì', 'trứng', 'sữa', 'cà phê', 'sô cô la']

Phương pháp này cũng hoạt động với danh sách trống. Ví dụ ,

>>danh sách trống = []

>>empty\_list.append(1)

>>in(danh sách trống)

Cái này mã số sẽ xuất ra :

[1]

Điều quan trọng là phải nhớ điều đó. Nối thêm () đứng sau danh sách. Điều này khác với các hàm như print đã được học trước đó.

**Bài tập**

1. Maria làm việc ở một cửa hàng hoa. Cô theo dõi các đơn hàng trong một danh sách gọi là đơn hàng.

Tạo danh sách đơn hàng và sử dụng bản in để kiểm tra các đơn hàng đã nhận được ngày hôm nay.

đơn đặt hàng = ['hoa cúc', 'hoa ngô'].

2. Maria vừa nhận được một đơn đặt hàng hoa tulip mới. Sử dụng phần bổ sung để thêm dòng này vào đơn hàng.

3. Một đơn hàng khác đã đến! Sử dụng phần bổ sung để thêm hoa hồng vào đơn hàng.

4. Sử dụng bản in để xem các đơn hàng mà Maria đã nhận được hôm nay.

**Toán tử + khi làm việc với danh sách**

Trước đây, chúng ta đã xem xét cách toán tử + hoạt động với các kiểu dữ liệu số và chuỗi. Đối với danh sách, toán tử này cũng được xác định. + cho phép bạn kết hợp nhiều danh sách lại với nhau. Ví dụ: một tiệm bánh có các loại sau:

>> mặt hàng \_ đã bán = [ ' bánh ' , ' bánh quy ' , ' bánh mì ' ]

Giả sử một tiệm bánh muốn bắt đầu bán bánh quy và bánh ngọt:

>> mặt hàng\_đã bán\_mới = mặt hàng\_đã bán + [ 'bánh quy' , 'bánh tart' ]

>> in ( items\_sold\_new )

[ 'bánh' , 'bánh quy' , 'bánh mì' , 'bánh quy' , 'bánh tart' ]

Trong ví dụ này, chúng tôi đã tạo một biến mới các mặt hàng \_ đã bán \_ mới chứa cả các mặt hàng đã bán ban đầu và các mặt hàng mới. Chúng ta có thể kiểm tra các mặt hàng gốc \_ đã bán và đảm bảo nó không bị thay đổi:

>> in ( items\_sold )

[ 'bánh' , 'bánh quy' , 'bánh mì' ]

Cũng giống như chuỗi và số, + chỉ có thể áp dụng cho danh sách. Nếu chúng ta nhập mã này:

>> danh sách của tôi = [ 1 , 2 , 3 ]

>> danh sách của tôi + 4

chúng ta sẽ gặp lỗi sau:

TypeError: chỉ có thể nối danh sách (không phải "int") vào danh sách

Nếu chúng ta muốn thêm một phần tử bằng +, chúng ta phải đặt nó vào danh sách bằng dấu ngoặc ([]):

>>danh sách của tôi + [4]

**Bài tập**

1. Maria vẫn đang cập nhật danh sách đơn hàng của mình. Cô ấy vừa nhận được đơn đặt hàng cho "lilac" và "iris".

Sử dụng + để tạo danh sách mới có tên new\_orders kết hợp các đơn hàng với hai đơn hàng mới.

đơn đặt hàng = ['daisy', 'buttercup', 'snapdragon', 'gardenia', 'lily']

#giá\_giá hỏng = [5, 3, 4, 5, 4] + 4

1. Xóa # trước danh sách giá \_ bị hỏng . Nếu bạn chạy mã này, bạn sẽ gặp lỗi
2. Sửa lệnh để nó chạy không có lỗi.

**Phạm vi I**

Đôi khi chúng ta cần tạo danh sách bao gồm các chuỗi. Ví dụ: bạn cần tạo một danh sách như:

>> danh sách của tôi = [ 0 , 1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9 ]

Việc gõ một số số như vậy sẽ mất một khoảng thời gian nhất định. Ngoài ra, chúng ta gõ càng nhiều số thì khả năng xảy ra lỗi đánh máy càng cao.

Python cung cấp cho chúng ta một cách dễ dàng để tạo các danh sách này bằng cách sử dụng hàm có tên là phạm vi. phạm vi lấy một số làm đầu vào và tạo các số bắt đầu từ 0 và kết thúc bằng số trước tham số đầu vào. Vì vậy, nếu chúng ta cần các số từ 0 đến 9, chúng ta sử dụng phạm vi (10) vì 10 lớn hơn 9 1:

>> phạm vi của tôi = phạm vi ( 10 )

Giống như zip, hàm phạm vi trả về một đối tượng mà chúng ta có thể chuyển đổi thành danh sách:

>>phạm vi (0, 10)

>>in(danh sách(my\_range))

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

**Bài tập**

1. Thay đổi danh sách 1 để nó là một dãy chứa các số bắt đầu từ 0 đến 9, nhưng không bao gồm 9.

danh sách 1=[1,8]

1. Tạo một phạm vi có tên là danh sách 2 với các số từ 0 đến 7.

**Phạm vi II**

Chúng ta có thể sử dụng phạm vi để tạo các chuỗi phức tạp hơn.

Theo mặc định, phạm vi tạo một danh sách bắt đầu từ 0. Tuy nhiên, nếu chúng ta chỉ định 2 số làm tham số đầu vào thì chúng ta có thể chỉ định phần đầu và phần cuối của chuỗi đã tạo. Ví dụ: phạm vi (2, 9) sẽ tạo ra các số bắt đầu từ 2 và kết thúc bằng 8 (không bao gồm 9):

>> danh sách của tôi = phạm vi ( 2 , 9 )

>>> in ( danh sách ( my\_list ))

[ 2 , 3 , 4 , 5 , 6 , 7 , 8 ]

Với một hoặc hai đối số, phạm vi sẽ tạo một danh sách các số liên tiếp (nghĩa là mỗi số lớn hơn số trước một). Nếu sử dụng đối số thứ ba, chúng ta có thể tạo một danh sách "bỏ qua" các số. Ví dụ: phạm vi (2, 9, 2) sẽ cho chúng ta một danh sách trong đó mỗi số lớn hơn số trước 2:

>>> my\_range2 = phạm vi ( 2 , 9 , 2 )

>>> in ( danh sách ( my\_range2 ))

[ 2 , 4 , 6 , 8 ]

Chúng ta có thể bỏ qua bao nhiêu số tùy thích! Trong ví dụ này, chúng ta sẽ bắt đầu từ 1 và bỏ qua 10 giữa mỗi số cho đến khi đạt tới 100:

>> my\_range3 = phạm vi ( 1 , 100 , 10 )

>> in ( danh sách ( my\_range3 ))

[ 1 , 11 , 21 , 31 , 41 , 51 , 61 , 71 , 81 , 91 ]

Danh sách của chúng tôi dừng ở 91 vì số tiếp theo trong dãy là 101, lớn hơn 100 (điểm dừng của chúng tôi).

Các giá trị đối số của hàm phạm vi có thể được ký hiệu là ( stert , stop , step ) (bắt đầu, kết thúc, bước).

**Bài tập**

1. Thay đổi hàm phạm vi đã tạo list1 để nó:

• Bắt đầu từ 5

• Sự khác biệt giữa mỗi phần tử là 3 đơn vị.

• Kết thúc bằng 15

list1 = phạm vi (6, 15, 2)

1. Tạo một đối tượng phạm vi có tên list2:

• Bắt đầu từ 0

• Sự khác biệt giữa mỗi phần tử là 5 đơn vị.

• Kết thúc trước 40

**Phần kết luận**

Trong bài học này chúng ta đã học:

• Cách tạo danh sách

• Cách tạo danh sách các danh sách bằng kho lưu trữ zip.

• Cách thêm các mục vào danh sách bằng cách sử dụng .append() hoặc +

• Cách sử dụng phạm vi để tạo danh sách số nguyên

**Nhiệm vụ**

1. Maria nhập dữ liệu khách hàng cho công việc thiết kế web của mình. Bạn sẽ giúp cô ấy sắp xếp dữ liệu của mình.

Bắt đầu bằng cách chuyển danh sách tên khách hàng này thành danh sách có tên first\_names. Hãy nhớ nhập tên theo thứ tự sau:

• Anna

• Boris

• Alexander

• Denis

2. Tạo một danh sách trống tên là tuổi.

3. Tuổi của Denis là 42 tuổi. Sử dụng .append() để thêm 42 vào tuổi.

4. Maria cần danh sách độ tuổi của tất cả khách hàng. Tạo một danh sách mới có tên all\_ages để thêm các độ tuổi với danh sách sau chứa độ tuổi của Anna, Boris và Alexander:

[32, 41, 29]

Đảm bảo rằng all\_ages bắt đầu bằng độ tuổi của Anna, Boris và Alexander và kết thúc bằng độ tuổi của Denis (được lưu trữ theo độ tuổi)!

5. Tạo một biến mới name\_and\_age kết hợp first\_names và all\_ages bằng zip.

6. Tạo một phạm vi có tên id với số ID cho mỗi khách hàng. Vì có 4 khách hàng nên giá trị id phải thay đổi từ 0 đến 3.

**Các thao tác với danh sách**

Trước đây, bạn đã học cách tạo danh sách và thêm các mục vào danh sách.

Trong bài học này bạn sẽ học:

• Chức năng lấy độ dài danh sách.

• Chọn các tập con của một danh sách (gọi là slice).

• Khả năng tính toán số lần một mục xuất hiện trong danh sách.

• Sắp xếp danh sách các mục

**Độ dài danh sách**

Thông thường chúng ta cần tìm số phần tử trong một danh sách, thường được gọi là độ dài của danh sách.

Chúng ta có thể làm điều này bằng cách sử dụng hàm len. Khi áp dụng len cho một danh sách, chúng ta sẽ nhận được số phần tử trong danh sách đó:

>>my\_list = [1, 2, 3, 4, 5]

>>print(len(my\_list))

Kết quả của mã này sẽ là::

5

**Bài tập**

1. Tính độ dài của list1 và lưu nó vào biến list1\_len.

list1 = phạm vi (2, 20, 2)

2. Sử dụng print để kiểm tra list1\_len.

3. Thay đổi lệnh phạm vi tạo list1 để nó bỏ qua 3 phần tử thay vì 2 phần tử.

Danh sách1\_len này sẽ thay đổi như thế nào?

**Chọn các mục danh sách tôi**

Chris phỏng vấn ứng viên xin việc. Nó sẽ đặt tên cho từng ứng cử viên theo thứ tự được biểu thị bằng danh sách Python:

>> cuộc gọi = ['Alice', 'Ivan', 'Sergey', 'Denis', 'Anna']

Đầu tiên anh ấy sẽ gọi " Alice ", sau đó là " Ivan ", v.v.

Trong Python, thứ tự của một phần tử trong danh sách được gọi là chỉ mục của nó.

Danh sách Python bắt đầu ở chỉ số 0. Điều này có nghĩa là phần tử đầu tiên trong danh sách có chỉ số 0 chứ không phải 1.

Dưới đây là số sê-ri của danh sách này:

| **Yếu tố** | **chỉ mục** |
| --- | --- |
| ' Alice ' | 0 |
| ' Ivan ' | 1 |
| ' Sergey ' | 2 |
| ' Denis ' | 3 |
| ' Anna ' | 4 |

Trong ví dụ này, phần tử có chỉ số 2 là " Sergey ".

Chúng ta có thể chọn một phần tử từ danh sách bằng dấu ngoặc vuông ([]) và chỉ mục của phần tử danh sách. Ví dụ: nếu chúng tôi muốn chọn phần tử thứ ba từ danh sách, chúng tôi sẽ sử dụng lệnh gọi [2]:

>>in(cuộc gọi[2])

' Sergey '

1. Sử dụng dấu ngoặc vuông ([]) để chọn mục ở chỉ số 4 từ danh sách nhân viên. Lưu trữ nó trong biến chỉ số 4.

nhân viên = ['Michael', 'Dwight', 'Jim', 'Pam', 'Ryan', 'Andy', 'Robert']

1. Sử dụng print và len để hiển thị số lượng mặt hàng mà nhân viên có.
2. Dán lệnh sau vào mã:

in(nhân viên[8])

1. Lỗi IndexError được đưa ra khi chọn phần tử không tồn tại.

Trong dòng mã bạn đã dán, hãy thay đổi số 8 thành một số khác để bạn không gặp lỗi IndexError.

**Chọn mục danh sách II**

Nếu chúng ta muốn chọn phần tử cuối cùng của danh sách thì sao?

Python có các chỉ mục tiêu cực cho mục đích này . Bạn có thể sử dụng chỉ mục -1 để chọn phần tử cuối cùng của danh sách.

Hãy xem xét danh sách 5 mục sau đây:

>>danh sách1 = [ 'a' , 'b' , 'c' , 'd' , 'e' ]

Nếu chúng ta chọn phần tử -1, chúng ta sẽ nhận được phần tử cuối cùng, ' e ':

>> in ( danh sách 1 [ - 1 ])

' e '

Điều này giống như việc chọn phần tử ở chỉ mục 4:

>>> in ( danh sách1 [ 4 ])

'e'

**Bài tập**

1. Sử dụng print và len để hiển thị độ dài của shopping\_list.

shopping\_list = ['trứng', 'bơ', 'sữa', 'dưa chuột', 'nước ép', 'ngũ cốc']

2. Lấy phần tử cuối cùng của shopping\_list bằng chỉ mục -1. Lưu phần tử này vào biến Last\_element.

3. Bây giờ chọn phần tử có chỉ số 5 và lưu nó vào biến element5.

4. Sử dụng bản in để hiển thị cả phần tử5 và phần tử cuối cùng.

**Cắt danh sách**

Giả sử chúng ta có một danh sách các chữ cái:

>>chữ cái = [ 'a' , 'b' , 'c' , 'd' , 'e' , 'f' , 'g' ]

Giả sử chúng ta muốn chọn từ b đến f .

Chúng ta có thể thực hiện việc này bằng cú pháp sau: các chữ cái [start:end] trong đó:

• start là chỉ mục của phần tử đầu tiên chúng ta muốn đưa vào vùng chọn của mình. Trong trường hợp này, chúng tôi muốn bắt đầu bằng b , có chỉ mục 1.

• end là chỉ mục lớn hơn chỉ mục cuối cùng mà chúng ta muốn đưa vào. Phần tử cuối cùng chúng ta cần là f ở chỉ số 5, vì vậy end phải là 6.

>>danh sách phụ = các chữ cái [ 1 : 6 ]

>>in ( danh sách phụ )

Cái này mã số sẽ xuất ra :

['b', 'c', 'd', 'e', 'f']

Lưu ý rằng phần tử ở chỉ số 6 (nghĩa là g) không được đưa vào lựa chọn của chúng tôi.

**Bài tập**

1. Sử dụng print để kiểm tra biến đầu tiên.

vali = ['áo sơ mi', 'áo sơ mi', 'quần', 'quần', 'đồ ngủ', 'sách']

bắt đầu = vali [0: 2]

Có bao nhiêu phần tử trong danh sách?

2. Thay đổi phần đầu để chọn 4 phần tử đầu tiên của vali.

3. Tạo một danh sách mới có tên là middle chứa hai phần tử ở giữa từ vali.

**Danh sách lát II**

Nếu chúng ta muốn chọn 3 thành phần đầu tiên của danh sách, chúng ta có thể sử dụng đoạn mã sau:

>> trái cây = [ 'táo' , 'chuối' , anh đào' , 'dưa' ]

>> in ( trái cây [ 0 : 3 ])

[ 'táo' , 'chuối' , 'anh đào' ]

Bắt đầu từ đầu danh sách, cũng có thể bỏ qua 0:

>> in ( trái cây [: 3 ])

[ 'táo' , 'chuối' , 'anh đào' ]

Chúng ta có thể làm điều gì đó tương tự khi chọn một vài mục cuối cùng của danh sách:

>> in ( trái cây [ 2 :])

[ 'anh đào' , 'dưa' ]

Bạn có thể bỏ qua chỉ mục cuối cùng khi chọn các phần tử cuối cùng từ danh sách.

Nếu cần chọn 3 phần tử trái cây cuối cùng, chúng ta cũng có thể sử dụng cú pháp sau:

>> in ( trái cây [ - 3 :])

[ ' chuối ' , ' anh đào ' , ' dưa ' ]

Có thể chấp nhận sử dụng chỉ số âm để đếm ngược từ phần tử cuối cùng.

**Bài tập**

Tạo một danh sách mới có tên là start chứa 3 phần tử đầu tiên của vali.

vali = ['áo sơ mi', 'áo thun', 'vớ', 'kính', 'đồ ngủ', 'sách']

**Đếm các phần tử trong danh sách**

Giả sử chúng ta có một danh sách các chữ cái đại diện cho các chữ cái trong từ "Mississippi":

>> chữ cái = [ 'm' , 'i' , 's ' , 's' , ' tôi' , 's' , 'i' , 'p' , 'i' ]

Nếu muốn biết “và” xuất hiện bao nhiêu lần trong từ này, chúng ta có thể sử dụng hàm count :

>>num\_i = các chữ cái . đếm ( 'tôi' )

>> in ( num\_i )

Kết quả của việc thực thi đoạn mã này sẽ như sau:

4

**Bài tập**

Lớp cô Wilson bầu lớp trưởng. Cô lưu phiếu bầu của mỗi học sinh vào danh sách bình chọn.

Sử dụng bộ đếm để xác định có bao nhiêu học sinh đã bình chọn cho " Jake ". Lưu câu trả lời của bạn dưới dạng jake\_votes.

phiếu bầu = [' Jake ', ' Jake ' ' Laurie ', ' Laurie ', ' Laurie ', ' Jake ' ' Jake ' ' Jake ' ' Laurie ', ' Cassi ', ' Cassi ', ' Jake ' ' Jake ' ' Cassi ', ' Laurie ', ' Cassi ', ' Jake ' ' Jake ' ' Cassi ', ' Laurie ']

Sử dụng bản in để kiểm tra jake\_votes.

**Sắp xếp danh sách tôi**

Đôi khi chúng ta cần sắp xếp danh sách theo thứ tự tăng dần (1, 2, 3,...) hoặc theo thứ tự bảng chữ cái (a, b, c,...).

Chúng ta có thể sắp xếp danh sách bằng cách sử dụng .sort(). Giả sử chúng ta có một danh sách tên:

>>tên = [ 'Xander' , 'Buffy' , 'Thiên thần' , 'Liễu' , 'Giles' ]

>>in ( tên )

Cái này sẽ xuất ra :

['Xander', 'Buffy', 'Thiên thần', 'Liễu', 'Giles']

Hãy nộp đơn ngay bây giờ. loại ():

>> tên . loại ()

>>in ( tên )

Và chúng tôi nhận được:

['Thiên thần', 'Buffy', 'Giles', 'Liễu', 'Xander']

Lưu ý rằng hàm sắp xếp được viết sau danh sách tên. Nếu chúng tôi thử sắp xếp ( tên ), chúng tôi sẽ nhận được NameError .

sắp xếp không trả lại bất cứ điều gì. Vì vậy, nếu chúng ta cố gắng gán tên cho biến . Sort (), biến mới của chúng ta sẽ là None :

>>sort\_names = những cái tên . loại ()

>>in ( đã sắp xếp\_names )

điều này sẽ trở lại:

Không có

Mặc dùsort\_names là Không có, nhưng dòng Sort\_names =names.sort() vẫn chỉnh sửa tên:

>> in ( tên )

[ 'Thiên thần' , 'Buffy' , 'Giles' , 'Liễu' , 'Xander' ]

**Bài tập**

Loại địa chỉ .

địa chỉ = ['221 B Baker St.', '42 Wallaby Way', '12 Grimmauld Place', '742 Evergreen Terrace', '1600 Pennsylvania Ave', '10 Downing St.']

Sử dụng bản in để xem địa chỉ đã thay đổi như thế nào

**Danh sách sắp xếp II**

Cách thứ hai để sắp xếp danh sách là sử dụng tính năng sắp xếp. được sắp xếp khác với .sort() ở những điểm sau:

1. Nó đứng trước danh sách, không đứng sau danh sách.

2. Tạo một danh sách mới.

Hãy quay trở lại danh sách tên của chúng tôi:

>>tên = [ 'Xander' , 'Buffy' , 'Thiên thần' , 'Liễu' , 'Giles' ]

Sử dụng được sắp xếp, chúng ta có thể tạo một danh sách mới có tên làsort\_names :

>>sort\_names = sắp xếp ( tên )

>>in ( đã sắp xếp\_names )

Điều này đưa ra một danh sách được sắp xếp theo thứ tự abc:

['Thiên thần', 'Buffy', 'Giles', 'Liễu', 'Xander']

Lưu ý rằng việc sử dụng được sắp xếp không thay đổi tên :

>> in ( tên )

[ 'Xander' , 'Buffy' , 'Thiên thần' , 'Liễu' , 'Giles' ]

**Bài tập**

1. Sử dụng tính năng sắp xếp để sắp xếp trò chơi của bạn và tạo danh sách mới có tên games\_sorted.

trò chơi = ['Cổng thông tin', 'Minecraft', 'Pacman', 'Tetris', 'The Sims', 'Pokemon']

1. Sử dụng bản in để kiểm tra trò chơi và trò chơi được sắp xếp.

**Kết luận**

Đã học:

• Cách lấy độ dài của danh sách

• Chọn các tập con của một danh sách (gọi là cắt).

• Tính số lần một mục xuất hiện trong danh sách.

• Sắp xếp danh sách các phần tử

**Bài tập**

1. hàng tồn kho là danh sách các mặt hàng có trong kho đồ nội thất. Có bao nhiêu sản phẩm trong kho?

Lưu câu trả lời của bạn vào Inventory\_len.

hàng tồn kho = ["giường đôi", "giường đôi", "đầu giường", "giường đôi", "giường đôi", "tủ ngăn kéo", "tủ ngăn kéo", "bàn", "bàn", "bàn cạnh giường ngủ", "bàn đầu giường", "giường cỡ king", "giường đôi", "giường đôi", "giường đôi", "khăn trải giường", "khăn trải giường", "gối", "gối"]

2. Chọn vật phẩm đầu tiên trong kho của bạn. Đầu tiên lưu trữ nó trong biến đầu tiên .

3. Chọn mục cuối cùng từ kho và lưu nó vào biến cuối cùng .

4. Chọn các mục từ kho, bắt đầu từ chỉ mục 2 trở lên nhưng không bao gồm chỉ mục 6.

Lưu câu trả lời của bạn vào kho \_2\_6.

1. Chọn 3 mặt hàng tồn kho đầu tiên và lưu trữ chúng trong biến first\_3 .
2. Có bao nhiêu giường đơn trong kho? Lưu câu trả lời của bạn vào Twin\_beds .
3. Sắp xếp hàng tồn kho bằng cách sử dụng . loại ().

**Các phương pháp làm việc với danh sách**

Là một phần của việc này, các phương pháp sau sẽ được nghiên cứu:

* mở rộng()
* chèn()
* di dời()
* nhạc pop()
* thông thoáng( )
* chỉ số()
* đảo ngược()
* sao chép()

**Mở rộng danh sách**

Giả sử chúng ta có một danh sách những việc muốn làm hôm nay:

>> to \_ do \_ list = ['nấu bữa sáng', 'đi giặt quần áo', 'gọi điện cho khách hàng',

'nấu bữa tối', 'rửa bát']

Và sau đó chúng tôi phát hiện ra rằng chúng tôi muốn treo quần áo đã giặt trước khi đi ngủ nhưng lại hoàn toàn quên thêm nó vào danh sách việc cần làm! Có một số phương pháp để thêm giá trị bổ sung vào danh sách:

1. Sử dụng toán tử + để thêm vào cuối danh sách

>>to\_do\_list = ['nấu bữa sáng', 'giặt đồ', 'gọi điện cho khách hàng', 'nấu bữa tối', 'rửa bát']

>>to\_do\_list+=[' treo đồ lót ']

>>in(to\_do\_list)

kết quả sẽ :

['nấu bữa sáng', 'đi giặt đồ', 'gọi điện cho khách hàng', 'nấu bữa tối', 'rửa bát', 'treo đồ giặt']

1. Sử dụng phép cắt và tính toán độ dài danh sách

>>to\_do\_list = ['nấu bữa sáng', 'giặt đồ', 'gọi điện cho khách hàng', 'nấu bữa tối', 'rửa bát']

>>to\_do\_list[len(to\_do\_list):]=[' treo đồ lót ']

>> in ( việc \_ làm \_ danh sách )

Kết quả tương tự như ví dụ trước.

Trong ví dụ này, độ dài của danh sách đã được tính toán và sau đó được chỉ ra rằng bắt đầu từ phần tử có chỉ mục bằng giá trị độ dài hiện tại và cho đến cuối, chèn các giá trị của danh sách khác. Kết quả là các giá trị của danh sách khác đã được thêm vào cuối danh sách

1. Sử dụng phương thức mở rộng()

>>to\_do\_list = ['nấu bữa sáng', 'giặt đồ', 'gọi điện cho khách hàng', 'nấu bữa tối', 'rửa bát']

>>to\_do\_list.extend(' treo đồ lót ')

>>in(to\_do\_list)

Kết quả tương tự như ví dụ trước.

Phương thức mở rộng cũng cho phép bạn thêm các phần tử vào trong danh sách. Bạn có thể sử dụng bất kỳ phương pháp được trình bày. Điều chính cần nhớ là phương pháp này chỉ chấp nhận các kiểu dữ liệu tuần tự.

**Bài tập**

1. Mark làm bồi bàn trong một nhà hàng. Anh ta chấp nhận đơn đặt hàng và thêm nó vào danh sách. Tạo một danh sách đơn hàng và thêm mì ống, pizza và salad caprese vào đó.
2. Một khách hàng khác ngồi vào bàn và Mark gọi món gồm salad Caesar và cà phê. Thêm dữ liệu vị trí vào cuối danh sách đơn hàng
3. Khách hàng nhớ rằng họ đã hoàn toàn quên mất đồ uống! Thêm rượu vang đỏ khô vào danh sách đặt hàng của bạn.

**phương pháp chèn**

Đôi khi cần thêm một phần tử không phải vào cuối danh sách mà vào một vị trí nhất định được chỉ định bởi một chỉ mục. Có một phương thức chèn() cho việc này . Phương thức này có 2 đối số:

* Vị trí (chỉ mục) mà bạn muốn đặt phần tử tại đó. Đánh số bắt đầu từ đầu. Lập chỉ mục tiêu cực được hỗ trợ.
* Phần tử cần thêm vào danh sách.

Ví dụ, chúng tôi nhớ ra rằng chúng tôi đã quên rửa bát sau bữa sáng! Cần thêm nhiệm vụ này vào danh sách sau mục “Chuẩn bị bữa sáng”. Để làm điều này chúng ta sẽ làm như sau:

>>to\_do\_list = ['nấu bữa sáng', 'giặt đồ', 'gọi điện cho khách hàng',

'nấu bữa tối', 'rửa bát']

>> việc \_ làm \_ danh sách . chèn (1, 'rửa bát')

>> in ( việc \_ làm \_ danh sách )

Kết quả là, thông tin sau sẽ được xuất ra bàn điều khiển:

['nấu bữa sáng', 'rửa bát, 'đi giặt', 'gọi điện cho khách hàng', 'nấu bữa tối', 'rửa bát']

Như bạn có thể thấy, dòng bắt buộc đã được thêm vào danh sách ở vị trí dưới chỉ mục 1.

**Bài tập**

1. Mark đưa danh sách gọi món cho nhà bếp, nhưng khách hàng yêu cầu món khai vị rau trước khi toàn bộ đơn hàng được giao. Thêm vị trí này vào đầu danh sách trong đơn hàng .

**Các phương pháp xóa mục khỏi danh sách**

trong python : Remove(), pop() và clear(). Ngoài chúng ra còn có từ khóa del.

Chúng ta hãy xem xét từng cái một cách riêng biệt.

**phương thức xóa ()**

Phương pháp này cho phép bạn xóa một phần tử khỏi danh sách theo giá trị. Ví dụ, chúng tôi đã thay đổi ý định nấu bữa tối vì chúng tôi đến nhà hàng với đồng nghiệp.

>>to\_do\_list.insert(1, 'rửa bát')

>>to\_do\_list.remove('nấu bữa tối')

Kết quả sẽ là:

['nấu bữa sáng', 'rửa bát', 'giặt đồ', 'gọi điện cho khách hàng', 'rửa bát']

Như bạn có thể thấy, việc nấu bữa tối đã bị xóa khỏi danh sách việc cần làm. Tuy nhiên, vẫn còn việc rửa bát sau bữa tối! Nhưng trong danh sách có 2 việc rửa bát, một việc sau bữa sáng, một việc sau bữa tối. Nếu bạn sử dụng phương thức Remove () thì phần tử có giá trị chỉ số thấp hơn sẽ bị loại bỏ! Trong trường hợp của chúng tôi:

>> việc \_ làm \_ danh sách . xóa ('nấu bữa tối')

Sẽ xuất ra:

['nấu bữa sáng', 'giặt đồ', 'gọi điện cho khách hàng', 'rửa bát']

Như chúng ta có thể thấy, chỉ có mục gặp phải đầu tiên (chỉ mục 1) đã bị xóa và mục thứ hai “rửa bát” vẫn còn trong danh sách (mục cuối cùng trong danh sách).

Hãy chú ý! Nếu không tìm thấy phần tử, một lỗi sẽ được hiển thị cho biết phần tử không được tìm thấy.

**Bài tập**

1. Khách hàng cho rằng có quá nhiều đồ ăn và quyết định bỏ qua món salad caprese. Xóa mục này khỏi danh sách

**phương thức pop()**

Phương pháp này cho phép bạn xóa một phần tử khỏi danh sách theo giá trị chỉ mục. Trong ví dụ trước, chúng tôi muốn xóa “rửa bát” (mục cuối cùng trong danh sách) khỏi danh sách việc cần làm. Trong trường hợp này chúng ta có thể sử dụng phương thức pop ():

>> việc \_ làm \_ danh sách . bật lên (-1)

Kết quả là chúng tôi nhận được:

['nấu bữa sáng', 'rửa bát', 'giặt đồ', 'gọi điện cho khách hàng']

Như bạn có thể thấy, phần tử cuối cùng trong danh sách đã bị xóa.

Hãy chú ý! Nếu không tìm thấy chỉ mục này hoặc nằm ngoài phạm vi, phương thức sẽ đưa ra ngoại lệ IndexError: pop index.

pop () trả về giá trị đã bị loại bỏ và cho phép nó được ghi vào một biến riêng. Ví dụ: chúng tôi muốn đặt giá trị vừa bị xóa khỏi danh sách vào biến del \_ item , cho việc này:

>>del\_item = to\_do\_list.pop(-1)

**Bài tập**

1. Khách hàng của Mark nhớ ra rằng hôm nay họ có một cuộc gọi quan trọng nên họ đã thay đổi ý định về việc uống rượu. Họ yêu cầu Mark loại bỏ mặt hàng này khỏi đơn hàng. Xóa rượu khỏi danh sách đặt hàng ban đầu.

**phương thức clear()**

Phương pháp này cho phép bạn xóa hoàn toàn danh sách (xóa tất cả các thành phần khỏi nó). Ngày này đã kết thúc và chúng tôi quyết định xóa danh sách việc cần làm vì tất cả chúng tôi đã hoàn thành chúng:

>>to\_do\_list.clear()

>>in(to\_do\_list)

Kết quả là một danh sách trống:

[]

**Xóa từ khóa**

Từ khóa del , như pop (), cho phép bạn xóa một phần tử khỏi danh sách theo chỉ mục. Tuy nhiên, nó cũng cho phép bạn xóa một loạt giá trị cùng một lúc! Giả sử chúng ta có một danh sách các số và chúng ta muốn xóa các giá trị trong đó với các chỉ số bắt đầu từ 1 đến 3:

>>số = [1, 2, 3, 5, 6, 8, 3]

>>del số[1:4]

>>in(số)

Mã này sẽ xuất ra:

[1, 6, 8, 3]

Bài tập

1. Tạo danh sách các số từ 0 đến 7
2. Xóa 2 phần tử ở giữa danh sách bằng del

**phương thức đảo ngược()**

Phương pháp này sắp xếp các phần tử trong danh sách theo thứ tự ngược lại. Ví dụ ,

>>my\_list = [1, 'two', 'a', 4]   
>>my\_list.reverse() # None>>my\_list # [4, 'a', 'two', 1]

**Bài tập**

1. In danh sách theo thứ tự ngược lại

x = [15, 11, 13, 12, 14, 10]

**phương thức sao chép()**

Giả sử chúng ta có một danh sách lst 1 = [0, 1, 2, 3, 4, 5]. Chúng tôi muốn tạo một bản sao của nó. Hãy thử sử dụng toán tử gán và tạo biến lst 2 trong đó chúng ta sẽ lưu trữ một bản sao:

>>lst2 = lst1

Giả sử chúng ta muốn xóa phần tử có chỉ mục 2 khỏi danh sách đầu tiên, chúng ta sử dụng phương thức pop () mà chúng ta đã học hôm nay:

>> lst 2 = lst 1

>> lst 1. pop (2)

>> in ( lst 2)

>> in ( lst 1)

Hãy xuất cả hai danh sách ra bàn điều khiển. Chúng ta thấy gì?

[0, 1, 3, 4, 5]

[0, 1, 3, 4, 5]

Như bạn có thể thấy, chúng tôi đã xóa phần tử có chỉ mục 2 khỏi chỉ một danh sách, nhưng nó đã bị xóa khỏi cả hai danh sách. Tại sao lại như vậy?

Hãy nhớ chủ đề của các biến. Toán tử gán chỉ xác định tham chiếu đến một đối tượng trong bộ nhớ. Nghĩa là, tại thời điểm lst 2 và lst 1 trỏ đến cùng một vị trí trong bộ nhớ (tham khảo cùng một danh sách).

Để tạo một bản sao đầy đủ của danh sách, bạn phải sử dụng phương thức copy (), phương thức này sẽ cho phép bạn có được một bản sao đầy đủ của danh sách; việc xóa khỏi danh sách này sẽ không đồng nghĩa với việc xóa khỏi danh sách khác.

>>lst2 = lst1.copy()

>>lst1.pop(2)

>> in ( lst 2)

>> in ( lst 1)

Mã này sẽ xuất ra bàn điều khiển:

[0, 1, 2, 3, 4, 5]

[0, 1, 3, 4, 5]

Như bạn có thể thấy trong ví dụ, phần tử chỉ bị xóa khỏi lst 1.

Chu kỳ

Hãy tưởng tượng rằng chúng ta có một danh sách các loại kem được bán tại ki-ốt. Nhiệm vụ của chúng ta là hiển thị lần lượt tất cả các loại kem này vào bảng điều khiển. Sau đó, mã sẽ trông như thế này:

>>kem\_kem = ['dâu tây', 'quả hồ trăn', 'sô cô la', 'quả mâm xôi', 'kurasawa xanh', 'chanh', 'vani']

>>print(ice\_cream[0])

>>print(ice\_cream[1])

>>print(ice\_cream [2])

>>print(ice\_cream [3])

>>print(ice\_cream [4])

>>print(ice\_cream [5])

>> in ( kem \_ kem [6])

Điều này có vẻ không hiệu quả và cồng kềnh. Trong Python (và hầu hết các ngôn ngữ lập trình khác), có một cách dễ dàng hơn để lặp qua từng phần tử trong danh sách. Chúng ta có thể sử dụng vòng lặp! Vòng lặp là cách lặp lại mã chương trình nhiều lần.

Bài học này sẽ giải thích:

từ khóa for .

• Vòng lặp lặp lại mã đã gõ cho đến khi đáp ứng một số điều kiện, được biểu thị bằng từ khóa while.

• Vòng lặp tạo danh sách mới, gọi là biên soạn danh sách.

**Tạo vòng lặp for**

Có thể in từng phần tử của danh sách bằng vòng lặp for. Vòng lặp for cho phép chúng ta thực hiện một hành động trên từng phần tử trong danh sách. Việc sử dụng từng phần tử của danh sách được gọi là lặp lại hoặc lặp lại (các kiểu dữ liệu mà vòng lặp có thể được áp dụng được gọi là lặp lại).

Hãy tạo một vòng lặp để hiển thị các loại kem từ ví dụ trước:

>>kem\_kem = ['dâu tây', 'quả hồ trăn', 'sô cô la', 'quả mâm xôi', 'kurasawa xanh', 'chanh', 'vani']

>>đối với mặt hàng trong Ice\_cream:

>> in ( mục )

Cú pháp chung để tạo vòng lặp for là:

>> cho < tạm thời biến > TRONG < danh sách biến > :

>> <hành động>

Trong ví dụ về kem của chúng ta, item là biến tạm thời, Ice\_cream là biến danh sách và print ( item ) là hành động được thực hiện trên mỗi mục trong danh sách.

Biến tạm thời của chúng tôi có thể có bất kỳ tên nào và không cần phải xác định trước. Mỗi đoạn mã sau đây thực hiện công việc tương tự như ví dụ của chúng tôi:

>>cho tôi ở Ice\_cream:

>>in(i)

>>đối với Ice\_cream\_sort trong Ice\_cream:

>> in ( kem \_ kem \_ sắp xếp )

Lưu ý rằng trong tất cả các ví dụ này, câu lệnh print đều được thụt vào. Mọi thứ ở cùng mức thụt lề sau khi khai báo vòng lặp for đều được đưa vào vòng lặp for và chạy trên mỗi lần lặp.

Nếu chúng ta quên thụt lề, chúng ta sẽ gặp lỗi IndentationError.

**Bài tập**

1.Chạy mã bên dưới. Bạn sẽ nhận được IndentationError vì dòng in (trò chơi) không được thụt lề.

board\_games = ['Người định cư Catan', 'Carcassone', 'Lưới điện', 'Agricola', 'Scrabble']

sport\_games = ['bóng đá', 'bóng đá - Mỹ', 'khúc côn cầu', 'bóng chày', 'cricket']

cho trò chơi trong board\_games:

in ( trò chơi )

2. Thụt lề dòng 6 để bạn không gặp lỗi IndentationError khi chạy mã.

3. Viết vòng lặp in từng môn thể thao trong sport\_games.

**Sử dụng phạm vi trong vòng lặp**

Trước đây, vòng lặp for được sử dụng để lặp qua các phần tử của danh sách hiện có.

Đôi khi cần phải lặp lại không phải qua một danh sách cụ thể mà phải thực hiện một hành động nhất định nhiều lần. Ví dụ: nếu chúng tôi muốn hiển thị “CHÚ Ý!” trên bảng điều khiển. nhắn tin ba lần, chúng tôi muốn nói điều gì đó như:

>>cho Tôi TRONG < một danh sách của chiều dài 3 > :

>> in ( " CHÚ Ý! " )

Lưu ý rằng chúng ta cần lặp lại danh sách có độ dài 3, nhưng chúng ta không quan tâm đến danh sách đó có gì. Để tạo các danh sách có độ dài n này, chúng ta có thể sử dụng hàm phạm vi . Phạm vi lấy số n làm đầu vào và trả về danh sách từ 0 đến n-1. Ví dụ :

>>zero\_thru\_5 = phạm vi ( 6 )

>># zero\_thru\_5 bây giờ là [0, 1, 2, 3, 4, 5]

>>zero\_thru\_one = phạm vi ( 2 )

>># zero\_thru\_one bây giờ là [0, 1]

Vì vậy, một cách đơn giản để thực hiện “CHÚ Ý!” một ví dụ sẽ là:

>> cho i trong phạm vi (3):

>> in ( " CHÚ Ý ! " )

**Bài tập**

1. Sử dụng hàm phạm vi trong vòng lặp for để in ra lời hứa 5 lần.

Promise = "Tôi sẽ không nhai kẹo cao su trong lớp"

**Vòng lặp vô hạn**

Trong các ví dụ trên, chúng tôi đã lặp lại các thành phần của danh sách có độ dài cố định (bắt đầu và kết thúc). Tuy nhiên Hãy coi như cái này ví dụ :

>> số\_favorite\_của tôi = [ 4 , 8 , 15 , 16 , 42 ]

>>cho con số TRONG my\_favorite\_numbers :

>> my\_favorite\_numbers . nối thêm ( 1 )

Cái gì Đây đang xảy ra ? Mỗi lần vào vòng lặp, chúng ta thêm 1 vào cuối danh sách có các phần tử mà chúng ta đang lặp lại. Kết quả là chúng ta không bao giờ đi đến cuối danh sách! Nó tiếp tục phát triển!

Vòng lặp không bao giờ kết thúc được gọi là vòng lặp vô hạn. Điều này rất nguy hiểm cho mã!

Một chương trình bị vướng vào vòng lặp vô hạn thường trở nên hoàn toàn không thể sử dụng được. Phải tránh các vòng lặp vô hạn. Nhưng nếu vô tình gặp phải một cái, bạn có thể kết thúc vòng lặp bằng cách nhấn Ctrl + c để kết thúc chương trình.

**Bài tập**

1. Giả sử chúng ta có hai danh sách sinh viên: sinh viên\_thời gian\_A và sinh viên\_thời\_B. Chúng tôi muốn hợp nhất tất cả sinh viên vào sinh viên\_thời\_B.

Viết một vòng lặp for đi qua từng học sinh trong sinh viên\_thời gian\_A và nối nó vào cuối sinh viên\_thời\_B.

sinh viên\_thời\_A = ["Alex", "Briana", "Cheri", "Daniele"]

sinh viên\_thời\_B = ["Dora", "Minerva", "Alexa", "Obie"]

2. Bên trong vòng lặp for, sau khi thêm sinh viên vào sinh viên\_thời\_B, hãy in sinh viên.

3. Giả sử bạn mắc lỗi đánh máy trong phần thân của vòng lặp for .

Bên trong vòng lặp for , thay đổi đối tượng câu lệnh add từ sinh viên \_ giai đoạn \_ B thành sinh viên \_ giai đoạn \_ A . Trong trường hợp này, bạn sẽ nhận được một vòng lặp vô hạn

Hãy thoát khỏi vòng lặp vô tận! Sau đó loại bỏ lỗi gây ra vòng lặp vô hạn.

**Phá vỡ**

Chúng ta thường muốn sử dụng vòng lặp for để tìm giá trị nào đó trong danh sách:

>>các mặt hàng đang giảm giá = [ "blue\_shirt" , "stripped\_socks" , "knit\_dress" , "red\_headband" , "dinosaur\_onesie" ]

# Chúng tôi muốn kiểm tra tình trạng còn hàng của sản phẩm có ID " knit\_dress " :

>>cho mục TRONG items\_on\_sale :

>> nếu mục == "đan\_đầm" :

>> print ( "Váy dệt kim đang giảm giá!" )

Mã này xem xét từng thành phần trong items\_on\_sale và kiểm tra sự trùng khớp. Khi chúng tôi thấy chiếc váy dệt kim nằm trong danh sách các mặt hàng đang giảm giá , chúng tôi không cần phải xem qua các mặt hàng còn lại trong danh sách giảm giá . Vì nó chỉ bao gồm 5 phần tử nên trong trường hợp này không quá khó để lặp lại toàn bộ danh sách. Nhưng nếu mặt hàng \_ on \_ sale sau " đan \_ váy " có 1000 mặt hàng thì sao? Điều gì sẽ xảy ra nếu có 100.000 mặt hàng sau " knit\_dress " ?

Bạn có thể dừng vòng lặp for từ bên trong vòng lặp bằng cách sử dụng break . Khi chương trình gặp câu lệnh break , điều khiển sẽ trả về mã bên ngoài vòng lặp for . Ví dụ:

>>các mặt hàng đang giảm giá = [ "blue\_shirt" , "stripped\_socks" , "knit\_dress" , "red\_headband" , "dinosaur\_onesie" ]

>>print ( "Kiểm tra danh sách bán hàng!" )

>>cho mục TRONG items\_on\_sale :

>> in ( mục )

>> nếu mục == "đan\_đầm" :

>> nghỉ

>>print ( "Kết thúc tìm kiếm!" )

Điều này sẽ cho kết quả:

Kiểm tra danh sách bán hàng!

áo xanh

sọc\_vớ

đan\_đầm

Kết thúc của tìm kiếm !

Chúng tôi không cần phải kiểm tra " red\_headband " hoặc " Dinosaur\_ onesie " !

**Tiếp tục**

Khi lặp lại danh sách, chúng ta có thể muốn bỏ qua một số giá trị. Giả sử chúng ta muốn in tất cả các số trong danh sách trừ khi chúng âm. Chúng ta có thể sử dụng continue để đi tới chữ i tiếp theo trong danh sách:

>>big\_number\_list = [ 1 , 2 , - 1 , 4 , - 5 , 5 , 2 , - 9 ]

>>cho Tôi TRONG big\_number\_list :

>> nếu Tôi < 0 :

>> Tiếp tục

>> in ( tôi )

Điều này sẽ cho kết quả:

1

2

4

5

2

**Bài tập**

1. Bạn có một danh sách các giống chó mà bạn có thể nhận nuôi thú cưng, dog\_breeds\_available\_for\_adoption. Sử dụng vòng lặp for, lặp qua danh sách dog\_breeds\_available\_for\_adoption và in ra từng giống chó.

dog\_breeds\_available\_for\_adoption = ['french\_bulldog', 'dalmatian', 'shihtzu', 'poodle', 'collie']

dog\_breed\_I\_want = 'chó đốm'

1. Bên trong vòng lặp for, kiểm tra xem giá trị giống hiện tại có khớp với dog\_breed\_I\_want hay không. Nếu có, hãy in ra "Họ có con chó mà tôi muốn!"
2. Thêm câu lệnh ngắt khi vòng lặp của bạn gặp dog\_breed\_I\_want để bạn không phải kiểm tra phần còn lại của danh sách.

**lặp while**

Có một loại vòng lặp khác gọi là vòng lặp while. Vòng lặp while thực thi mã cho đến khi đáp ứng một số điều kiện.

Vòng lặp while có thể được sử dụng để lặp qua các phần tử của danh sách, giống như vòng lặp for:

>>dog\_breeds = ['bulldog', 'dalmation', 'shihtzu', 'poodle', 'collie']

>>chỉ số = 0

>>trong khi chỉ mục < len(dog\_breeds):

>> print(dog\_breeds[index])

>> chỉ số += 1

Mỗi khi điều kiện của vòng lặp while được thỏa mãn (trong trường hợp này là chỉ số < len ( dog\_bres ) ) , đoạn mã bên trong vòng lặp while sẽ được thực thi .

Vòng lặp while có thể hữu ích khi bạn không biết cần bao nhiêu lần lặp để thỏa mãn một điều kiện.

**Vòng lặp lồng nhau**

Chúng ta đã thấy cách bạn có thể lặp lại các phần tử của một danh sách. Điều gì sẽ xảy ra nếu chúng ta có một danh sách được tạo thành từ nhiều danh sách? Làm cách nào chúng ta có thể lặp lại tất cả các phần tử riêng lẻ?

Giả sử chúng ta phụ trách một lớp khoa học được chia thành ba nhóm dự án:

>>dự án\_teams = [[ "Ava" , "Samantha" , "James" ], [ "Lucille" , "Zed" ], [ "Edgar" , "Gabriel" ]]

Nếu muốn lấy từng học sinh, chúng ta phải đặt một vòng lặp vào trong một vòng lặp khác:

>>cho đội TRONG dự án\_teams :

>> cho học sinh TRONG đội :

>> in ( sinh viên )

Điều này dẫn đến:

Ava

Samantha

James

Lucille

Zed

Edgar

Gabriel

**Bài tập**

Chúng tôi đã cung cấp danh sách sales\_data hiển thị số lượng hương vị kem khác nhau được bán ở ba địa điểm khác nhau trong một cửa hàng hư cấu. Chúng tôi muốn tóm tắt tổng số giống đã bán. Bắt đầu bằng cách xác định biến Scops\_sold và đặt nó về 0.

sales\_data = [[12, 17, 22], [2, 10, 3], [5, 12, 13]]

2. Duyệt liệt kê sales\_data. Tạo vòng lặp lặp qua từng danh sách trong danh sách sales\_data .

, lặp qua các giá trị bên trong mỗi danh sách phụ từ sales\_data và thêm Scops\_sold vào biến của bạn.

Cuối cùng, bạn sẽ có tổng của tất cả các số trong danh sách lồng nhau sales\_data.

4. In giá trị của số muỗng đã bán.

**Danh sách máy phát điện**

Giả sử chúng ta đã duyệt qua một trang web nhất định và nhận được những từ sau:

>>từ ngữ = [ "@coolguy35" , "#nofilter" , "@kewldawg54" , "trả lời" , "dấu thời gian" , "@matchamom" , "theo dõi" , "#updog" ]

Chúng tôi muốn tạo một danh sách mới tên là tên người dùng sẽ có tất cả các chuỗi bằng từ có "@" là ký tự đầu tiên. Chúng tôi biết chúng tôi có thể làm điều này với vòng lặp for :

>>từ ngữ = [ "@coolguy35" , "#nofilter" , "@kewldawg54" , "trả lời" , "dấu thời gian" , "@matchamom" , "theo dõi" , "#updog" ]

>>tên người dùng = []

>>cho từ TRONG từ :

>> nếu từ [ 0 ] == '@' :

>> tên người dùng . nối thêm ( từ )

Trước tiên, chúng tôi tạo một danh sách tên người dùng trống mới và khi xem qua danh sách các từ, chúng tôi đã thêm từng từ phù hợp với tiêu chí của mình. Bây giờ danh sách tên người dùng trông như thế này:

>>> in(tên người dùng)

[ "@coolguy 35 " , "@kewldawg 54 " , " @matchamom " ]

Python có một phím tắt tiện dụng để tạo danh sách một dòng như thế này :

>>tên người dùng = [ từ vì từ TRONG từ nếu như từ [ 0 ] == '@' ]

Đây được gọi là trình tạo danh sách. Kết quả sẽ giống như trong vòng lặp for :

[ "@coolguy 35 " , "@kewldawg 54 " , " @matchamom " ]

Hiểu cú pháp này:

1. Chấp nhận một phần tử trong từ

2. Gán phần tử này cho một biến tên là word

3. Kiểm tra xem từ [0] == '@' có xuất hiện hay không và nếu có, hãy thêm từ đó vào danh sách tên người dùng mới. Nếu không thì không có gì xảy ra.

4. Lặp lại các bước 1-3 cho tất cả các dòng trong từ.

Lưu ý: nếu chúng tôi không thực hiện bất kỳ kiểm tra nào (giả sử chúng tôi đã bỏ qua if word[0] == '@'), danh sách mới sẽ chỉ là bản sao của các từ:

>>tên người dùng = [ từ vì từ TRONG từ ]

>>#tên người dùng ngay bây giờ trông vì vậy ["@coolguy35", "#nofilter", "@kewldawg54", "trả lời", "dấu thời gian", "@matchamom", "theo dõi", "#updog"]

Giả sử chúng ta đang làm việc với danh sách tên người dùng từ bài tập trước:

>>> in ( tên người dùng )

[ "@coolguy 35 " , "@kewldawg 54 " , " @matchamom " ]

Chúng tôi muốn tạo một danh sách mới với chuỗi "vui lòng theo dõi tôi!" được thêm vào cuối mỗi tên người dùng. Chúng tôi muốn gọi đây là tin nhắn danh sách mới. Chúng ta có thể sử dụng khả năng hiểu danh sách để xây dựng danh sách này từ một dòng:

tin nhắn = [ người dùng + "Xin hãy đi theo tôi!" vì người dùng TRONG tên người dùng ]

Hiểu danh sách này:

1. Chấp nhận một chuỗi trong tên người dùng.

2. Gán chuỗi này cho một biến có tên user .

3. Thêm “hãy theo dõi tôi!” người dùng

4. Thêm liên kết này vào danh sách mới có tên là messages .

5. Lặp lại các bước 1–4 cho tất cả các dòng trong tên người dùng .

Bây giờ tin nhắn chứa các giá trị sau:

[ "@coolguy35 hãy theo dõi tôi!" , "@kewldawg54 hãy theo dõi tôi!" , "@matchamom hãy theo dõi tôi!" ]

Khả năng tạo danh sách với các giá trị được sửa đổi đặc biệt hữu ích khi làm việc với các con số. Giả sử chúng ta có danh sách này:

>> my\_upvote = [ 192 , 34 , 22 , 175 , 75 , 101 , 97 ]

Chúng tôi muốn thêm 100 vào mỗi giá trị. Chúng ta có thể đạt được mục tiêu này chỉ bằng một dòng:

>>update\_upvote = [ phiếu\_giá trị + 100 vì phiếu\_giá trị TRONG my\_upvote ]

Hiểu danh sách này:

1. Chấp nhận một số trong my\_upvotes

2. Gán số này cho một biến có tên là vote\_value .

3. Thêm 100 vào vote\_value

4. Thêm số tiền này vào danh sách \_ upvote được cập nhật mới .

5. Lặp lại các bước 1–4 cho tất cả các số trong my\_upvotes .

Update\_upvotes hiện chứa các giá trị sau :

[ 292 , 134 , 122 , 275 , 175 , 201 , 197 ]

**Phần kết luận**

Trong bài học này bạn đã học

• cách viết vòng lặp for

• cách sử dụng một phạm vi trong vòng lặp

• Vòng lặp vô tận là gì và cách tránh chúng

• cách bỏ qua các giá trị trong một vòng lặp

• cách viết vòng lặp while

• cách tạo danh sách một dòng

Hãy cùng luyện tập thêm với những khái niệm này nhé!

**Bài tập**

1. Tạo một danh sách có tên single\_digits gồm các số từ 0 đến 9 (đã bao gồm).

2. Tạo một vòng lặp for lặp qua các chữ số đơn và in từng chữ số.

3. Trước vòng lặp, hãy tạo một danh sách có tên là hình vuông. Chỉ định một danh sách trống để bắt đầu.

4. Bên trong vòng lặp lặp qua single\_digits, hãy cộng giá trị bình phương của mỗi phần tử single\_digits vào các ô vuông. Bạn có thể thực hiện việc này trước hoặc sau khi phần tử được hiển thị.

5. Sau vòng lặp for, in ra các hình vuông.

6. Tạo danh sách các hình khối bằng cách sử dụng so sánh danh sách trong danh sách chữ số đơn. Mỗi phần tử khối phải là phần tử single\_digits được lập phương.

7. In hình khối.

Dây

Từ và câu là phương tiện giao tiếp nên máy tính cũng cần có công cụ để làm việc với từ và câu.

Trong Python, chúng tôi lưu trữ các từ, câu hoặc thậm chí toàn bộ đoạn văn dưới dạng chuỗi. Một chuỗi là một chuỗi các ký tự. Chuỗi có thể có độ dài bất kỳ và chứa bất kỳ chữ cái, số, ký hiệu hoặc dấu cách nào.

Trong bài học này, bạn sẽ tìm hiểu thêm về chuỗi và cách chúng được xử lý trong Python. Bạn sẽ học cách cắt một chuỗi, chọn các ký tự cụ thể từ chuỗi, tìm kiếm các ký tự trong chuỗi, lặp qua các chuỗi và sử dụng chuỗi trong các câu lệnh có điều kiện.

**Bài tập**

1. Lưu trữ từ yêu thích của bạn dưới dạng một chuỗi trong biến favorite\_word.

2. In từ ưu tiên.

**Đây là tất cả danh sách các nhân vật!**

Một chuỗi có thể được coi là một chuỗi các ký tự.

Giống như bất kỳ danh sách nào khác, mỗi ký tự trong dòng đều có một chỉ mục. Hãy xem xét dòng

>> trái cây \_ yêu thích = "táo"

Chúng ta có thể chọn các chữ cái cụ thể từ chuỗi này bằng cách sử dụng chỉ mục. Chúng ta hãy nhìn vào chữ cái đầu tiên của dòng.

>>> \_ trái cây yêu thích [1]

'b'

Xin lưu ý rằng chữ cái ở chỉ số 1 của từ apple không phải là i mà là b. Điều này là do các chỉ số hàng bắt đầu từ 0. b ở chỉ số 0 và chúng ta có thể chọn nó bằng cách sử dụng:

>>> favorite\_fruit[0]

'TÔI'

Điều quan trọng cần lưu ý là chỉ số hàng phải là số nguyên. Nếu bạn cố gắng chọn một chỉ mục không nguyên, chúng tôi sẽ nhận được TypeError:

>>> favorite\_fruit[1.5]

Traceback (cuộc gọi gần đây nhất):

Tệp "<stdin>", dòng 1, trong <module>

TypeError: chỉ số chuỗi phải là số nguyên, không phải float

**Làm thế nào để có được một lát chuỗi**

Chúng ta không chỉ có thể chọn một ký tự từ một chuỗi mà còn có thể chọn các đoạn ký tự từ một chuỗi. Có thể thực hiện việc này bằng cú pháp sau:

chuỗi \_ tên [ chỉ số \_ đầu tiên : chỉ số cuối cùng ]

Đây được gọi là một lát cắt dòng. Khi tạo một lát cắt, chúng ta tạo một hàng mới bắt đầu bằng (và bao gồm) first\_index và kết thúc bằng (nhưng loại trừ) Last\_index . Hãy xem xét một vài ví dụ. Hãy nhớ trái cây yêu thích của bạn:

>>> \_ trái cây yêu thích

'quả táo'

Giả sử chúng ta cần một chuỗi mới chứa các chữ cái ok. Chúng ta có thể cắt favorite\_fruit như thế này:

>>> \_ trái cây yêu thích [3:5]

'ĐƯỢC RỒI'

Lưu ý rằng ký tự trong chỉ mục đầu tiên, i, được BAO GỒM, nhưng ký tự trong chỉ mục cuối cùng, o, bị LOẠI TRỪ.

Chúng tôi cũng có thể không có chỉ mục được liệt kê. Nếu chúng ta loại bỏ chỉ mục đầu tiên, lát cắt sẽ bắt đầu từ đầu hàng và nếu chúng ta loại bỏ chỉ mục thứ hai, lát cắt sẽ tiếp tục đến cuối hàng.

>>> favorite\_fruit[:4]

' quả táo '

>>> favorite\_fruit[4:]

'ko'

Một lần nữa, hãy chú ý cách loại trừ k của chuỗi khỏi ví dụ đầu tiên và được đưa vào ví dụ thứ hai.

**Bài tập**

1. Bạn là một lập trình viên đang làm việc trong một công ty lớn. Tại công ty này, tên người dùng của mỗi nhân viên được tạo bằng cách lấy năm chữ cái đầu tiên trong họ của họ.

Một nhân viên mới, Vitaly Krasilov, bắt đầu làm việc từ hôm nay và bạn cần tạo tài khoản cho anh ấy. Viết tên và họ của nó.

Tạo biến new\_account bằng cách loại bỏ năm chữ cái đầu tiên của họ.

1. Mật khẩu tạm thời cho nhân viên mới cũng được tạo từ họ của họ.

Tạo một biến tên là temp\_password bằng cách tạo một đoạn từ chữ cái thứ ba đến thứ sáu trong họ của nó.

**Nối (nối chuỗi)**

Bạn cũng có thể nối hai chuỗi hiện có thành một chuỗi mới. Hãy xem xét hai dòng sau đây.

>> tiền\_quả = " màu xanh da trời "

>> Fruit\_suffix = " quả mọng "

Chúng ta có thể tạo một chuỗi mới bằng cách nối chúng lại với nhau như thế này:

>>>favorite\_fruit = Fruit\_prefix + Fruit\_suffix

>>> yêu thích\_trái cây

' quả việt quất '

Lưu ý rằng không có khoảng trắng được thêm vào đây. Bạn phải thêm dấu cách theo cách thủ công khi nối chuỗi nếu muốn bao gồm chúng.

**Bài tập**

1. Ban quản lý tại Công ty mà bạn làm việc đã nhận ra rằng chính sách sử dụng năm chữ cái đầu tiên trong họ của nhân viên làm tên người dùng là không lý tưởng nếu họ có nhiều nhân viên có cùng họ.

Viết một hàm có tên account\_generator nhận hai đầu vào, first\_name và Last\_name, nối ba chữ cái đầu tiên của mỗi đầu vào rồi trả về tên tài khoản mới.

1. Kiểm tra hàm của bạn và lưu kết quả thực thi vào biến new\_account.

**Đo chiều dài chuỗi**

Python có một số hàm dựng sẵn để làm việc với chuỗi. Một trong những hàm được sử dụng phổ biến nhất là len(). len() trả về số ký tự trong một dòng

>>> favorite\_fruit = "quả việt quất"

>>> len(favorite\_fruit)

9

Nếu bạn đo độ dài câu, khoảng trắng sẽ được tính đến.

>>> Fruit\_sentence = "Tôi yêu quả việt quất"

>>> len ( câu trái cây )

18

Bây giờ hãy xem xét đoạn mã sau:

>>> chiều dài = len(favorite\_fruit)

>>> favorite\_fruit[độ dài]

Nhưng mã này sẽ đưa ra IndexError vì hãy nhớ rằng các chỉ số bắt đầu từ 0, vì vậy ký tự cuối cùng trong chuỗi có chỉ mục là len ( string \_ name ) - 1.

>>> \_ trái cây yêu thích [ chiều dài -1]

' ừ '

Bạn cũng có thể xuất ra một vài ký tự cuối cùng của chuỗi bằng len():

>>> favorite\_fruit[length-4:]

' er '

**Bài tập**

1. Ban quản lý công ty một lần nữa muốn cập nhật cách tạo mật khẩu tạm thời cho nhân viên mới.

Viết một hàm có tên là pass\_generator nhận hai đầu vào, first\_name và Last\_name, sau đó nối ba chữ cái cuối cùng của mỗi đầu vào và trả về chúng dưới dạng một chuỗi.

1. Kiểm tra chức năng của bạn và lưu trữ chúng trong biến temp\_password.

**chỉ số tiêu cực**

Trong bài tập trước, chúng ta đã sử dụng len() để lấy một đoạn ký tự ở cuối chuỗi.

Có một cách dễ dàng hơn để làm điều này - sử dụng chỉ số âm! Các chỉ số âm được đếm ngược từ cuối chuỗi, vì vậy string\_name[-1] là ký tự cuối cùng của chuỗi, string\_name[-2] là ký tự cuối cùng thứ hai của chuỗi, v.v.

Dưới đây là một số ví dụ:

>>> \_ trái cây yêu thích = ' việt quất `

>>> favorite\_fruit[-1]

'y'

>>> favorite\_fruit[-2]

' r '

>>> \_ trái cây yêu thích [-3:]

' xin lỗi '

Lưu ý rằng chúng ta có thể xuất ba ký tự cuối cùng của từ blueberry bằng cách có chỉ số bắt đầu là -3 và bỏ qua chỉ mục kết thúc.

**Bài tập**

1. Sử dụng chỉ số phủ định để tìm ký tự thứ hai đến ký tự cuối cùng trong company\_motto. Lưu trữ cái này trong biến thứ hai\_to\_last.

company\_motto= "Giấc mơ trở thành hiện thực"

1. Sử dụng chỉ mục phủ định để tạo một đoạn từ 4 ký tự cuối cùng trong company\_motto. Lưu trữ cái này trong biến Final\_word.

**Chuỗi là bất biến (bất biến)**

Cho đến nay trong hướng dẫn này, chúng ta đã chọn các ký tự từ chuỗi, các lát chuỗi được tạo thành và các chuỗi được nối. Mỗi lần chúng tôi thực hiện một trong những thao tác này, chúng tôi sẽ tạo một hàng hoàn toàn mới.

Điều này là do chuỗi là bất biến. Điều này có nghĩa là chúng ta không thể thay đổi hàng sau khi nó được tạo. Chúng ta có thể sử dụng một chuỗi để tạo các chuỗi khác nhưng không thể thay đổi chính chuỗi đó.

Tính chất này thường được gọi là tính bất biến (immutability). Các loại dữ liệu có thể thay đổi có thể thay đổi được, nhưng các loại dữ liệu như chuỗi không thể thay đổi thì không thể thay đổi được.

**Bài tập**

1. Lần thuê cuối cùng của chúng tôi là một anh chàng tên Rob Daly. Thật không may, HR dường như đã mắc một lỗi đánh máy nhỏ và gửi sai tên\_name.

Hãy thử thay đổi ký tự đầu tiên của first\_name bằng cách chạy

first\_name[0] = "P"

1. Chuỗi là bất biến nên chúng ta không thể thay đổi một ký tự đơn lẻ. Được rồi, đó không phải là vấn đề, chúng ta vẫn có thể khắc phục được!

Nối chuỗi "P" với đoạn first\_name, bao gồm mọi thứ ngoại trừ ký tự đầu tiên và lưu nó trong một chuỗi mới, fix\_first\_name.

**Trình tự thoát**

Đôi khi khi làm việc với chuỗi, bạn cần đưa vào các ký tự đã có ý nghĩa đặc biệt trong Python. Ví dụ : giả sử chúng ta tạo đường kẻ

>>favorite\_fruit\_conversation = "Anh ấy nói, "quả việt quất là món tôi thích nhất!""

Chúng ta đã vô tình kết thúc chuỗi sớm hơn mong muốn bằng cách thêm ký tự ". Chúng ta có thể làm điều này bằng cách nhập các ký tự thoát . Bằng cách thêm dấu gạch chéo ngược trước ký tự đặc biệt mà chúng ta muốn thoát, \", chúng ta có thể đưa nó vào chuỗi.

>>favorite\_fruit\_conversation = "Anh ấy nói, \"quả việt quất là món tôi thích nhất!\""

**Bài tập**

1. Khi Rob Daly thiết lập tài khoản, anh ấy đã đặt mật khẩu của mình:

họ gọi tôi là " điên "91

Mật khẩu của anh ấy đã gây ra một số lỗi trong hệ thống do "dấu". Viết lại mật khẩu của anh ấy bằng các ký tự thoát và lưu nó bằng một mật khẩu thay đổi.

**Lặp lại các hàng**

Bây giờ bạn đã biết đủ về chuỗi, chúng ta có thể bắt đầu thực hiện một số điều thực sự thú vị!

Vì chuỗi là các chuỗi, điều này có nghĩa là chúng ta có thể lặp qua một chuỗi bằng cách sử dụng vòng lặp for hoặc while. Điều này mở ra nhiều khả năng mà chúng ta có thể thao tác và phân tích chuỗi. Hãy Chúng ta sẽ thấy TRÊN ví dụ .

>>def print\_each\_letter(word):

>> cho chữ cái trong word:

>> in ( thư )

Mã này sẽ lặp qua từng chữ cái trong một từ nhất định và xuất nó ra thiết bị đầu cuối.

>>> favorite\_color = "xanh"

>>> print\_each\_letter(favorite\_color)

'b'

'tôi'

'bạn'

'e'

**Bài tập**

1. Hãy tái tạo hàm len() mà bạn đã quen thuộc.

Viết một hàm mới có tên get\_length() lấy một chuỗi làm đầu vào và trả về số ký tự trong chuỗi đó. Hãy thực hiện điều này bằng cách lặp lại một chuỗi bằng len()!

**Chuỗi và điều kiện (Phần một)**

Bây giờ chúng ta có thể lặp qua các chuỗi, chúng ta thực sự có thể khám phá tiềm năng của chuỗi. Khi lặp lại các ký tự của một chuỗi, chúng ta có thể thực hiện các hành động trên từng ký tự. Nếu chúng ta đưa các câu lệnh có điều kiện vào các lần lặp này, chúng ta có thể bắt đầu thực hiện một số điều thực sự thú vị.

Hãy xem đoạn mã sau:

>> trái cây \_ yêu thích = " việt quất "

>>bộ đếm = 0

>>cho nhân vật trong favorite\_fruit:

>> nếu ký tự == "b":

>> bộ đếm = bộ đếm + 1

>> in ( bộ đếm )

Mã này sẽ đếm số b trong chuỗi "blueberry" (gợi ý: có hai). Hãy cùng tìm hiểu xem chính xác đoạn mã này làm gì.

Đầu tiên, chúng ta xác định chuỗi favorite\_fruit và một biến gọi là bộ đếm mà chúng ta đặt thành 0. Vòng lặp for sau đó sẽ lặp qua từng ký tự trong favorite\_fruit và so sánh nó với chữ cái b.

Mỗi khi một ký tự bằng b, mã sẽ tăng bộ đếm của biến đó lên một. Sau đó, khi tất cả các ký tự đã được kiểm tra, mã sẽ xuất ra một biến đếm, cho chúng ta biết có bao nhiêu chữ b trong "blueberry". Đây là một ví dụ tuyệt vời về cách sử dụng phép lặp chuỗi để giải quyết một vấn đề cụ thể, trong trường hợp này là đếm số lần xuất hiện của một chữ cái cụ thể trong một từ.

**Bài tập**

1. Viết hàm có tên letter\_check nhận hai đầu vào: một từ và một chữ cái.

Hàm này sẽ trả về True nếu từ đó chứa một chữ cái và Sai nếu không có.

**Chuỗi và điều kiện (Phần thứ hai)**

Có một cách thậm chí còn dễ dàng hơn việc lặp lại toàn bộ chuỗi để xác định xem một ký tự có trong chuỗi hay không. Chúng ta có thể thực hiện loại kiểm tra này hiệu quả hơn bằng cách sử dụng in. Kiểm tra xem một chuỗi có phải là một phần của chuỗi khác hay không.

Đây là cú pháp trong trông như thế nào:

thư **TRONG** từ

Thư đây TRONG word là một biểu thức Boolean đánh giá là True nếu ký tự trong chuỗi nằm trong chuỗi word. Đây một số ví dụ :

>>>"e" trong "quả việt quất"

ĐÚNG VẬY

>>> "a" trong "quả việt quất"

SAI

Trên thực tế, phương pháp này mạnh hơn hàm bạn đã viết trong bài tập trước vì nó hoạt động không chỉ trên các chữ cái mà còn trên toàn bộ chuỗi.

>>> "màu xanh" trong "quả việt quất"

ĐÚNG VẬY

>>> “xanh” trong “dâu tây”

SAI

**Bài tập**

1. Viết một hàm có tên contains nhận vào hai đối số, big\_string và little\_string, rồi trả về True nếu big\_string chứa little\_string.

Ví dụ: chứa (" dưa hấu ", " dưa ") sẽ trả về Đúng và chứa (" dưa hấu ", " berry ") sẽ trả về Sai.

1. Viết một hàm có tên common\_letters nhận vào hai đối số, string\_one và string\_two, sau đó trả về danh sách tất cả các chữ cái chung của chúng.

Các chữ cái trong danh sách trả về phải là duy nhất. Ví dụ ,

common\_letters("chuối", "kem")

nên trả về ['a'].

**Phần kết luận**

Trong bài học này bạn đã học:

• Chuỗi là danh sách các ký tự (kiểu dữ liệu tuần tự).

• Một ký tự có thể được chọn từ một chuỗi bằng cách sử dụng tên chuỗi chỉ mục [index] của nó. Các chỉ số này bắt đầu từ 0.

• Có thể chọn “Slice” từ dòng. Chúng có thể nằm giữa hai chỉ mục hoặc có thể mở, chọn toàn bộ hàng từ một điểm.

• Các chuỗi có thể được kết hợp thành các chuỗi lớn hơn.

• len() có thể được sử dụng để xác định số lượng ký tự trong một dòng.

• Chuỗi có thể được lặp bằng vòng lặp for.

• Phép lặp chuỗi mang lại tiềm năng ứng dụng to lớn, đặc biệt khi kết hợp với các câu lệnh điều kiện.

**Bài tập**

1. Ban quản lý công ty một lần nữa đã tranh thủ sự giúp đỡ của bạn để tạo ra tính năng tạo tên người dùng và mật khẩu. Trong bài tập này, bạn sẽ tạo hai hàm: username\_generator và pass\_generator.

Hãy bắt đầu với tên người dùng\_generator. Tạo một hàm có tên username\_generator nhận hai đầu vào, first\_name và Last\_name, rồi trả về tên người dùng. Tên người dùng phải bao gồm ba chữ cái đầu tiên của tên và bốn chữ cái đầu tiên của họ. Nếu tên ít hơn ba chữ cái hoặc họ ít hơn bốn chữ cái thì phải sử dụng tên đầy đủ.

Ví dụ: nếu tên nhân viên là Abe Simpson thì hàm sẽ tạo tên người dùng AbeSimp.

1. Bây giờ đối với mật khẩu tạm thời, họ muốn hàm lấy tên người dùng đã nhập và chuyển tất cả các chữ cái sang bên phải để chữ cái cuối cùng của tên người dùng trở thành chữ cái đầu tiên, v.v. Ví dụ: nếu tên người dùng là AbeSimp thì mật khẩu tạm thời được tạo sẽ là pAbeSim.

Bắt đầu bằng cách xác định hàm pass\_generator để nó nhận một đầu vào là tên người dùng và xác định một chuỗi trống trong đó được gọi là mật khẩu.

1. Bên trong pass\_generator, hãy tạo một vòng lặp for lặp qua các ký tự trong tên người dùng.

**Phương pháp định dạng**

Có ba phương thức chuỗi có thể thay đổi kiểu chữ của chuỗi. Cái này . thấp hơn (), . trên () và . tiêu đề ().

• . low () trả về một chuỗi chứa tất cả các ký tự chữ thường.

• . Upper () trả về một chuỗi chứa tất cả các ký tự viết hoa.

• . title () trả về một chuỗi trong trường hợp tiêu đề, có nghĩa là chữ cái đầu tiên của mỗi từ được viết hoa.

Đây là một ví dụ về việc sử dụng. thấp hơn () đang hoạt động:

>>> bài hát yêu thích = 'Trơn tru'

>>> yêu thích\_song\_chữ thường = bài hát yêu thích . thấp hơn ()

>>> yêu thích\_song\_chữ thường

' trơn tru '

Mỗi ký tự đã được thay đổi thành chữ thường. Điều quan trọng cần nhớ là các phương thức chuỗi chỉ có thể tạo chuỗi mới, chúng không sửa đổi chuỗi gốc.

>>> bài hát yêu thích

' Trơn tru '

Hãy xem, mọi thứ đều giống nhau! Các phương thức chuỗi này rất tốt để làm sạch đầu vào của người dùng và chuẩn hóa định dạng chuỗi của bạn.

**Bài tập**

1. Bạn là lập trình viên làm việc cho một tổ chức đang cố gắng số hóa và bảo tồn thơ mang tên Preserve the Verse.

Bạn được đưa cho hai dòng, tựa đề bài thơ và tác giả, và được yêu cầu định dạng lại chúng một chút để phù hợp với quy tắc cơ sở dữ liệu của tổ chức.

Đặt tiêu đề bài thơ\_title và lưu nó trong bài thơ\_title\_fixed.

Poem\_title = "cơn bão mùa xuân"

Poem\_author = "William Carlos Williams"

1. In Poem\_title và Poem\_title\_fixed.

**Tách chuỗi**

.split() được thực thi trên một chuỗi, lấy một đối số và trả về danh sách các chuỗi con được tìm thấy giữa đối số đã cho (trong trường hợp .split() được gọi là dấu phân cách). Nên sử dụng cú pháp sau:

>> chuỗi \_ tên . chia ( dấu phân cách )

Nếu bạn không cung cấp đối số cho .split(), mặc định sẽ là phân chia theo dấu cách.

Ví dụ , hãy xem xét tiếp theo dòng :

>>> man\_its\_a\_hot\_one = "Cách mặt trời giữa trưa bảy inch"

>>> man\_its\_a\_hot\_one.split()

['Thích', 'bảy', 'inch', 'từ', 'the', 'giữa trưa', 'mặt trời']

. tách trả về một danh sách với mỗi từ trên một dòng riêng biệt. Điều quan trọng cần lưu ý: nếu chúng tôi chạy . Split () đối với chuỗi không có dấu cách, chúng ta sẽ nhận được chuỗi tương tự trong phản hồi.

**Bài tập**

1. Biến chứa một dòng từ dòng đầu tiên trong bài thơ "Cơn bão mùa xuân" của William Carlos Williams.

Sử dụng .split() để tạo một danh sách có tên line\_one\_words chứa mọi từ trong dòng thơ đó.

line\_one = "Bầu trời đã nhường chỗ"

Nếu chúng tôi cung cấp một đối số cho . tách (), chúng ta có thể chỉ định ký tự mà chúng ta muốn phân tách chuỗi của mình. Đối số này phải được cung cấp dưới dạng chính chuỗi đó.

Hãy xem xét ví dụ sau:

>>> vĩ đại nhất\_guitarist = "santana"

>>> lớn nhất\_guitarist.split('n')

[' sa ', ' ta ', ' a ']

Chúng tôi đã cung cấp "n" làm đối số cho .split(), do đó chuỗi "santana" của chúng tôi được chia ở mỗi "n" thành một danh sách gồm ba chuỗi.

Bạn nghĩ điều gì sẽ xảy ra nếu chúng ta chia cùng một chuỗi cho "a"?

>>> lớn nhất\_guitarist.split('a')

['s', 'nt', 'n', ' ']

Lưu ý rằng có một dòng bổ sung không mong muốn trong danh sách này. Nếu bạn chia một chuỗi theo ký tự mà nó cũng kết thúc bằng ký tự đó thì sẽ có một chuỗi trống ở cuối danh sách.

Bạn có thể sử dụng bất kỳ chuỗi nào làm đối số cho .split(), biến nó thành một công cụ linh hoạt và mạnh mẽ.

**Bài tập**

1. Sếp của bạn đã gửi tên tác giả mà ông ấy muốn bạn chuẩn bị để nhập vào cơ sở dữ liệu. Thật không may, nó gửi chúng dưới dạng một chuỗi dài với các tên được phân tách bằng dấu phẩy.

Sử dụng .split() và chuỗi đã chỉ định, tạo một danh sách có tên là Author\_names chứa tên của từng tác giả dưới dạng chuỗi riêng.

tác giả = "Audre Lorde, Carlos Williams, Gabriela Mistral, Jean Toomer, An Qi, Walt Whitman, Shel Silverstein, Carmen Boullosa, Kamala Suraiyya, Langston Hughes, Adrienne Rich, Nikki Giovanni"

1. Làm tốt lắm, nhưng bây giờ hóa ra họ không cần tên các nhà thơ (tại sao họ không nói điều đó ngay lần đầu tiên!?)
2. Tạo một danh sách khác tên là Author\_last\_names chỉ chứa họ của các nhà thơ trong chuỗi được cung cấp.

**Trình tự thoát E**

Chúng ta cũng có thể phân tách chuỗi bằng cách sử dụng chuỗi thoát. Chuỗi thoát được sử dụng để chỉ ra rằng chúng ta muốn tách một cái gì đó trong một chuỗi không nhất thiết phải là một ký tự. Chúng ta sẽ xem xét hai chuỗi thoát:

•\n Dòng mới

• tab \t

Dòng mới hoặc \n sẽ cho phép chúng ta chia một chuỗi nhiều dòng ở các ngắt dòng, trong khi \t sẽ cho phép chúng ta chia một chuỗi ở các tab. \t đặc biệt hữu ích khi làm việc với một số tập dữ liệu nhất định vì việc các hàng dữ liệu được phân tách bằng tab không phải là hiếm.

Hãy xem một ví dụ về việc phân tách bằng chuỗi thoát:

>>smooth\_chorus = \

>>"""Và nếu em nói, "Cuộc sống này chưa đủ tốt."

>>Anh sẽ dùng cả thế giới của mình để nâng em lên

>>Tôi có thể thay đổi cuộc sống của mình để phù hợp hơn với tâm trạng của bạn

>>Bởi vì em quá mịn màng"""

>>chorus\_lines = Smooth\_chorus.split('\n')

>> in ( điệp khúc \_ dòng )

Mã này chia một chuỗi nhiều dòng thành các dòng mới (\n) tồn tại ở cuối mỗi dòng và lưu trữ nó trong một danh sách mới gọi là chorus\_lines. Sau đó Anh ta in ra điệp khúc\_lines, trong đó sẽ sản xuất Phần kết luận

['Và nếu em nói, 'Cuộc sống này chưa đủ tốt'', 'Anh sẽ dùng cả thế giới của mình để nâng em lên', 'Anh có thể thay đổi cuộc sống của mình để phù hợp hơn với tâm trạng của em', 'Bởi vì em thật mượt mà"]

Danh sách mới chứa mỗi dòng của chuỗi gốc dưới dạng chuỗi nhỏ hơn của chính nó. Cũng lưu ý rằng Python tự động thoát khỏi ký tự ' khi tạo danh sách mới.

**Bài tập**

1. Tổ chức đã gửi tới các bạn toàn văn bài thơ “Giông tố mùa xuân” của William Carlos Williams. Họ muốn bạn chia bài thơ thành từng dòng riêng lẻ.

Tạo một danh sách có tên spring\_storm\_lines chứa một dòng cho mỗi dòng Spring Storm.

spring\_storm\_text = \

"""Bầu trời đã khuất phục

vị đắng của nó.

Sự thay đổi từ bóng tối

suốt cả ngày

mưa rơi và rơi

như thể nó sẽ không bao giờ kết thúc.

Tuyết vẫn cứ rơi

nó giữ trên mặt đất.

Nhưng nước, nước

từ một ngàn lần chạy!

Nó thu thập nhanh chóng,

lốm đốm màu đen

tự tìm đường đi cho mình

qua lớp băng xanh trong máng xối.

Từng giọt rơi xuống

từ những thân cỏ khô héo

của bờ kè nhô ra."""

**hợp chất dòng**

Bây giờ bạn đã học cách tách các chuỗi bằng .split(), hãy tìm hiểu cách nối chúng lại với nhau bằng .join(). .join() về cơ bản là đối lập với .split(), nó nối một danh sách các chuỗi cùng với một dấu phân cách nhất định. Cú pháp .join():

'dấu phân cách' . tham gia ( list\_you\_want\_to\_join )

Điều này có vẻ hơi lạ vì với . đối số chia () là một dấu phân cách và bây giờ đối số là một danh sách. Điều này là do join vẫn là một phương thức chuỗi, có nghĩa là nó phải hoạt động trên một chuỗi. Chuỗi trong phương thức. join () là dấu phân cách mà bạn muốn nối các giá trị từ một danh sách thành một chuỗi, do đó danh sách bạn muốn nối phải là một đối số.

Điều này có thể hơi khó hiểu, vì vậy hãy xem một ví dụ.

>>> my\_munequita = ['Của tôi', 'Tây Ban Nha', 'Harlem', 'Mona', 'Lisa']

>>> ' '.join(my\_munequita)

Chúng tôi lấy danh sách các chuỗi my\_munequita và nối nó bằng dấu phân cách '', đây là khoảng trắng. Khoảng cách rất quan trọng nếu bạn đang cố gắng xây dựng một câu không có từ, nếu không chúng ta sẽ nhận được:

' Của tôi tiếng Tây Ban Nha Harlem Mona Lisa '

Từ lovari

Từ điển là một tập hợp các cặp khóa:giá trị không có thứ tự.

Điều này cho chúng ta khả năng so sánh dữ liệu với nhau để có thể nhanh chóng tìm thấy các giá trị có liên quan với nhau.

Giả sử chúng ta muốn lưu trữ giá của nhiều loại hàng hóa được bán trong quán cà phê:

* Bánh mì nướng bơ - $6
* Nước ép cà rốt - $5
* Bánh nướng xốp việt quất có giá 2 USD

Trong Python, chúng ta có thể tạo một từ điển có tên menu để lưu trữ dữ liệu này:

>> thực đơn = {"Bánh mì nướng bơ": 6, "Nước ép cà rốt": 5, "Bánh nướng xốp việt quất": 2}

Xin lưu ý:

* Từ điển bắt đầu và kết thúc bằng dấu ngoặc nhọn { và }.
* Mỗi phần tử bao gồm một khóa ("bánh mì nướng bơ") và một giá trị (6).
* Mỗi cặp khóa:giá trị được phân tách bằng dấu phẩy.
* Việc chèn dấu cách sau mỗi dấu phẩy được coi là cách tốt nhưng mã vẫn hoạt động nếu không có dấu cách.

**Bài tập**

1. Giả sử chúng ta có một từ điển về cảm biến nhiệt độ trong một ngôi nhà và các giá trị nhiệt độ hiện tại. Chúng tôi vừa thêm một cảm biến vào "tủ đựng thức ăn" và nó hiển thị 22 độ.

Thêm cặp này vào từ điển:

cảm biến = {"phòng khách": 21, "bếp": 23, "phòng ngủ": 20}

1. Sao chép dòng mã bên dưới và xóa ký hiệu # trước khi xác định từ điển num\_Cameras , đại diện cho số lượng camera ở mỗi khu vực xung quanh nhà.

Nếu bạn chạy mã này, bạn sẽ gặp lỗi:

Lỗi cú pháp : không hợp lệ cú pháp

Hãy thử tìm và sửa lỗi cú pháp để chạy đoạn mã này.

#num\_CAMERA = {"sân sau": 6, "gara": 2 "đường lái xe" 1}

**Tạo một từ điển**

Trong bài tập trước, chúng ta đã thấy một từ điển khớp với các chuỗi và số (ví dụ: "bánh mì nướng bơ": 6). Tuy nhiên, khóa cũng có thể là số.

Ví dụ: nếu chúng ta khớp tổng phụ của hóa đơn nhà hàng với tổng hóa đơn sau tiền boa thì từ điển có thể trông như thế này:

>> subtotal\_to\_total = {20:24, 10:12, 5:6, 15:18}

Giá trị có thể thuộc bất kỳ loại nào. Chúng ta có thể sử dụng một chuỗi, một số, một danh sách hoặc thậm chí một từ điển khác làm giá trị được liên kết với khóa!

Ví dụ :

>>students\_in\_classes = {"thiết kế phần mềm": ["Aaron", "Delila", "Samson"], "bản đồ": ["Christopher", "Juan", "Marco"], "triết học": ["Frederica" , "Manuel"]}

Danh sách ["Aaron", "Delila", "Samson"], là giá trị của khóa " phần mềm" thiết kế ,” đại diện cho các học sinh trong lớp này.

Chúng ta cũng có thể trộn và kết hợp các loại khóa và giá trị. Ví dụ :

>>người = {"tên": "Shuri", "tuổi": 18, "gia đình": ["T'Chaka", "Ramonda"]}

**Bài tập**

1. Tạo một từ điển dịch khớp các từ tiếng Anh sau đây với định nghĩa của chúng bằng tiếng Sindarin (ngôn ngữ của Yêu tinh):

| **Tiếng Anh** | **Sindarin** |
| --- | --- |
| núi | orod |
| bánh mỳ | âm trầm |
| người bạn | dưa |
| ngựa | roch |

**Khóa không hợp lệ**

Chúng ta có thể có một danh sách hoặc một từ điển làm giá trị của một phần tử trong từ điển, nhưng chúng ta không thể sử dụng các kiểu dữ liệu này làm khóa từ điển. Nếu chúng ta cố gắng làm điều này, chúng ta sẽ nhận được TypeError.

Ví dụ:

>>sức mạnh = {[1, 2, 4, 8, 16]: 2, [1, 3, 9, 27, 81]: 3}

Cái này mã số sẽ cho :

TypeError: loại không thể xóa được: 'danh sách'

Từ "không thể băm" trong ngữ cảnh này có nghĩa là "danh sách" này là một đối tượng có thể được sửa đổi.

Từ điển trong Python dựa vào mỗi khóa có giá trị băm, một mã định danh cụ thể cho khóa. Nếu khóa có thể thay đổi thì giá trị băm này sẽ không đáng tin cậy. Do đó, khóa phải luôn là loại dữ liệu bất biến, có thể băm như số hoặc chuỗi.

**Từ điển trống**

Không nên có bất cứ điều gì trong từ điển. Đôi khi chúng ta cần tạo một từ điển trống khi dự định điền nó sau này dựa trên các dữ liệu đầu vào khác.

Chúng ta có thể tạo một từ điển trống như thế này:

>>empty\_dict = {}

**Thêm một chìa khóa**

Để thêm một cặp khóa:giá trị vào từ điển, chúng ta có thể sử dụng cú pháp:

từ điển[khóa] = giá trị

Ví dụ: nếu chúng ta có từ điển menu từ bài tập đầu tiên:

>>menu = {"bột yến mạch": 3, "bánh mì nướng bơ": 6, "nước ép cà rốt": 5, "bánh nướng xốp việt quất": 2}

Và chúng tôi muốn thêm một mặt hàng mới, "bánh pho mát" với giá 8 USD, chúng tôi có thể sử dụng:

>>thực đơn["bánh phô mai"] = 8

Bây giờ menu trông như thế này:

>>{"bột yến mạch": 3, "bánh mì nướng bơ": 6, "nước ép cà rốt": 5, "bánh nướng xốp việt quất": 2, "bánh phô mai": 8}

**Bài tập**

1. Tạo một từ điển trống có tên là Animals\_in\_zoo.
2. Dạo quanh vườn thú, bạn nhìn thấy 8 con ngựa vằn. Thêm "ngựa vằn" vào Animals\_in\_zoo làm khóa có giá trị là 8.
3. Thêm "khỉ" vào Animals\_in\_zoo làm khóa có giá trị là 12.
4. Khi rời sở thú, bạn rất buồn vì không nhìn thấy con khủng long nào. Thêm "khủng long" vào Animals\_in\_zoo làm khóa có giá trị 0.
5. In Animals\_in\_zoo ra bảng điều khiển.

**Thêm nhiều phím**

Nếu chúng ta muốn thêm nhiều cặp key:value vào từ điển cùng một lúc, chúng ta có thể sử dụng phương thức .update().

Hãy xem xét từ điển có kết quả đọc bằng cảm biến từ bài tập trước:

>>cảm biến = {"phòng khách": 21, "nhà bếp": 23, "phòng ngủ": 20}

Nếu chúng tôi muốn thêm 3 phòng mới, chúng tôi có thể sử dụng:

>>sensors.update({"phòng đựng thức ăn": 22, "phòng khách": 25, "sân": 34})

Điều này sẽ thêm cả ba mục vào từ điển cảm biến.

Bây giờ các cảm biến trông như thế này:

>> {"phòng khách": 21, "bếp": 23, "phòng ngủ": 20, "phòng đựng thức ăn": 22, "phòng khách": 25, "sân": 34}

**Bài tập**

1. Vào từ điển user\_ids bên dưới, hãy thêm hai người dùng mới:

* theLooper, với id là 138475
* stringQueen, với id là 85739

>> ser\_ids = {"teraCoder": 9018293, "proProgrammer": 119238}

1. In kết quả ra bàn điều khiển

**Ghi đè giá trị**

Chúng tôi biết rằng chúng tôi có thể thêm khóa bằng cú pháp này:

>>thực đơn["bánh mì nướng bơ "] = 7

Điều này sẽ tạo khóa "bánh mì nướng bơ" và đặt giá trị thành 7. Nhưng nếu chúng ta đã có mục "bánh mì nướng bơ" trong từ điển thì sao?

Trong trường hợp này, việc gán giá trị của chúng tôi sẽ ghi đè giá trị hiện có được gắn vào khóa "bánh mì nướng bơ".

>>menu = {"bột yến mạch": 3, "bánh mì nướng bơ": 6, "nước ép cà rốt": 5, "bánh nướng xốp việt quất": 2}   
>>menu["bột yến mạch"] = 5   
>>print(menu)

n này sẽ tạo ra kết quả sau:

{"bột yến mạch": 5, "bánh mì nướng bơ": 6, "nước ép cà rốt": 5, "bánh nướng xốp việt quất": 2}

Lưu ý rằng "bột yến mạch" đã thay đổi thành 5.

**Bài tập**

* 1. Thêm khóa "Nữ diễn viên phụ" vào từ điển oscar\_winners và đặt giá trị thành "Viola Davis".

oscar\_winners = {"Phim hay nhất": "La La Land", "Nam diễn viên chính xuất sắc nhất": "Casey Affleck", "Nữ diễn viên chính xuất sắc nhất": "Emma Stone", "Phim hoạt hình": "Zootopia"}

* 1. Không thay đổi định nghĩa từ điển oscar\_winners, hãy thay đổi giá trị được liên kết với khóa "Hình ảnh đẹp nhất" thành "Ánh trăng".

**Tạo từ điển**

Giả sử chúng ta có hai danh sách muốn kết hợp thành một từ điển, ví dụ: danh sách sinh viên và danh sách chiều cao của họ tính bằng inch:

>>names = ['Jenny', 'Alexus', 'Sam', 'Grace']   
>>chiều cao = [61, 70, 67, 64]

Python cho phép bạn tạo một từ điển bằng cách tạo từ điển, với cú pháp sau:

>>students = {key:value for key, value in zip(names, Heights)}   
>>#students hiện là {'Jenny': 61, 'Alexus': 70, 'Sam': 67, 'Grace': 64 }

Hãy nhớ rằng zip () kết hợp hai danh sách thành một kiểu có thể lặp lại chứa các bộ dữ liệu với các phần tử danh sách được nối với nhau. Đây là thế hệ từ điển:

* Lấy một cặp từ một trình lặp tuple
* Đặt tên các phần tử trong một cặp theo khóa (tên ban đầu có trong danh sách tên) và giá trị (tên ban đầu có trong danh sách tăng trưởng của học sinh)
* Tạo phần tử key:value trong từ điển sinh viên.
* Lặp lại các bước 1-3 cho toàn bộ cặp lặp

**Bài tập**

1. Bạn có hai danh sách hiển thị một số đồ uống được bán tại quán cà phê và số miligam caffeine trong mỗi loại. Đầu tiên, tạo một biến có tên zipped\_drinks, đây là một loại có thể lặp lại bao gồm các cặp giữa danh sách đồ uống và danh sách caffeine.
2. Tạo một từ điển có tên Drinks\_to\_caffeine bằng cách sử dụng tính năng tạo từ điển lặp qua trình vòng lặp zipped\_drinks và biến mỗi cặp bộ dữ liệu thành một phần tử key:value.

đồ uống = ["espresso", "chai", "decaf", "nhỏ giọt"]

caffein = [64, 40, 0, 120]

**Phần kết luận**

Cho đến nay chúng ta đã học được:

* Cách tạo từ điển
* Cách thêm mục vào từ điển
* Cách cập nhật các mục trong từ điển
* Cách sử dụng tính năng hiểu danh sách để tạo từ điển từ hai danh sách

**Bài tập**

1. Chúng tôi đang tạo ra một dịch vụ phát nhạc trực tuyến. Chúng tôi đã cung cấp hai danh sách hiển thị các bài hát trong thư viện của người dùng và số lần mỗi bài hát được phát.

Bằng cách sử dụng tính năng tạo từ điển, tạo một từ điển có tên lượt phát lặp qua zip(bài hát, số lượt phát) và tạo một cặp bài hát: số lượt phát cho mỗi bài hát trong bài hát và mỗi số lượt phát trong số lượt phát.

bài hát = ["Giống như một hòn đá lăn", "Sự hài lòng", "Hãy tưởng tượng", "Chuyện gì đang xảy ra", "Tôn trọng", "Rung động tốt"]

số lượt chơi = [78, 29, 44, 21, 89, 5]

1. In vở kịch.
2. Sau đó, thêm một mục mới vào vở kịch. Bài dự thi phải dành cho bài hát "Purple Haze" và số lượt phát phải là 1.
3. Người dùng này đã gây sốt Aretha Franklin và đã nghe “Respect” tới 5 lần.
4. Tạo một từ điển tên là thư viện có hai cặp khóa:giá trị:

"Những bài hát hay nhất" với các vở kịch ý nghĩa, (tức là từ điển bạn tạo ra để sử dụng làm ý nghĩa)

khóa "Cảm xúc ngày chủ nhật" với giá trị từ điển trống

1. Thư viện in.

**Sử dụng từ điển**

Bài học này đề cập đến các vấn đề sau:

* Sử dụng khóa để lấy giá trị từ từ điển
* Kiểm tra chìa khóa
* Lặp qua các khóa và giá trị trong từ điển

**Nhận giá trị theo khóa**

Nếu bạn có một từ điển, bạn có thể truy cập các giá trị của nó bằng cách chỉ định một khóa. Ví dụ: hãy tưởng tượng rằng chúng ta có một từ điển trong đó chiều cao của các tòa nhà được hiển thị bằng mét:

>>building\_heights = {"Burj Khalifa": 828, "Tháp Thượng Hải": 632, "Abraj Al Bait": 601, "Ping An": 599, "Tháp Thế giới Lotte": 554,5, "Thương mại Một Thế giới": 541,3}

Sau đó chúng ta có thể truy cập dữ liệu trong đó như thế này:

>>> Building\_heights["Burj Khalifa"]   
828

**Bài tập**

1. Chúng tôi đã cung cấp một từ điển khớp các yếu tố của các yếu tố với các dấu hiệu của cung hoàng đạo. Xuất ra bảng điều khiển danh sách các cung hoàng đạo được liên kết với phần tử “trái đất”.

cung hoàng đạo\_elements = {"nước": ["Cự Giải", "Bọ Cạp", "Song Ngư"], "lửa": ["Bạch Dương", "Sư Tử", "Nhân Mã"], "đất": ["Kim Ngưu", "Xử Nữ" " ", "Ma Kết"], "air":["Song Tử", "Thiên Bình", "Bảo Bình"]}

1. Hiển thị danh sách các dấu hiệu yếu tố lửa trên bảng điều khiển.

**Khóa không hợp lệ**

Giả sử chúng ta có từ điển chiều cao tòa nhà từ bài tập trước:

>>building\_heights = {"Burj Khalifa": 828, "Tháp Thượng Hải": 632, "Abraj Al Bait": 601, "Ping An": 599, "Tháp Thế giới Lotte": 554,5, "Thương mại Một Thế giới": 541,3}

Nếu chúng ta muốn biết chiều cao của Landmark 81 tại Thành phố Hồ Chí Minh thì sao? Chúng tôi có thể sẽ thử :

>>print(building\_heights["Landmark 81"])

81 " không tồn tại dưới dạng khóa trong từ điển Building\_heights ! Vì vậy, điều này sẽ dẫn đến KeyError :

Lỗi chính: 'Mốc 81'

Một cách để tránh lỗi này là trước tiên hãy kiểm tra xem khóa có tồn tại trong từ điển hay không:

>>key\_to\_check = "Landmark 81"   
>>if key\_to\_check trong Building\_heights:   
>> print(building\_heights["Landmark 81"])

Điều này sẽ không gây ra lỗi vì khóa \_ để \_ kiểm tra độ cao của tòa nhà sẽ trả về Sai và vì vậy chúng tôi không bao giờ thử truy cập vào khóa.

**Bài tập**

1. Chạy mã. Nó sẽ ném KeyError! "năng lượng" không tồn tại như một trong những yếu tố.
2. Thêm vào "năng lượng" chính của Zodiac\_elements. Nó phải khớp với giá trị "Không phải là phần tử Hoàng đạo". Chạy mã. Điều này có giải quyết được KeyError không?

cung hoàng đạo\_elements = {"nước": ["Cự Giải", "Bọ Cạp", "Song Ngư"], "lửa": ["Bạch Dương", "Sư Tử", "Nhân Mã"], "đất": ["Kim Ngưu", "Xử Nữ" " ", "Ma Kết"], "air":["Song Tử", "Thiên Bình", "Bảo Bình"]}

**Thử/Ngoại trừ để lấy chìa khóa**

Chúng tôi thấy rằng chúng tôi có thể tránh KeyErrors bằng cách trước tiên kiểm tra xem khóa có trong từ điển hay không. Một phương pháp khác chúng ta có thể sử dụng là thử/ngoại trừ:

key\_to\_check = "Mốc 81"   
thử :   
print( Building\_heights [ key\_to\_check ])   
ngoại trừ KeyError :   
print( "Khóa đó không tồn tại!" )

Khi chúng ta cố gắng truy cập vào một khóa không tồn tại, chương trình sẽ đi vào khối ngoại trừ và in ra "Khóa này không tồn tại!"

**Bài tập**

1. Sử dụng khối thử / ngoại trừ để cố gắng suy ra mức độ caffeine của một kết quả phù hợp. Nếu xảy ra KeyError , hãy in "Mức caffeine không xác định".

caffeine\_level = {"espresso": 64, "chai": 40, "decaf": 0, "nhỏ giọt": 120}

1. Phía trên khối try , thêm " matcha " vào từ điển có giá trị 30

**Nhận chìa khóa an toàn**

Trong bài tập trước, chúng ta đã thấy rằng mình có thể thêm cặp khóa:giá trị vào từ điển để tránh KeyError. Giải pháp này không bền vững. Chúng tôi không thể dự đoán mọi khóa mà người dùng có thể gọi và thêm tất cả các giá trị giữ chỗ đó vào từ điển của chúng tôi!

Từ điển có phương thức .get() để tra cứu một giá trị thay vì my\_dict[key] mà chúng ta đã sử dụng trước đó. Nếu khóa bạn đang cố sử dụng trong phương thức .get() không tồn tại, nó sẽ trả về None theo mặc định:

>>building\_heights = {"Burj Khalifa": 828, "Tháp Thượng Hải": 632, "Abraj Al Bait": 601, "Ping An": 599, "Tháp Thế giới Lotte": 554,5, "Thương mại Một Thế giới": 541,3}   
>>#dòng này sẽ trả về 632:   
>>building\_heights.get("Shanghai Tower")   
>>#dòng này sẽ trả về Không:   
>>building\_heights.get("My House")

Có thể chỉ định một giá trị được trả về nếu khóa không tồn tại. Ví dụ: chúng ta có thể trả về chiều cao của tòa nhà là 0 nếu tòa nhà mong muốn không có trong từ điển:

>>> Building\_heights.get('Shanghai Tower', 0)   
632   
>>> Building\_heights.get('Mt Olympus', 0)   
0   
>>> Building\_heights.get('Kilimanjaro', 'No Value')   
'No Value'

**Bài tập**

1. Sử dụng .get() để lấy giá trị của ID người dùng teraCoder, với giá trị mặc định là 100000 nếu người dùng không tồn tại. Lưu nó trong biến tc\_id. In tc\_id ra bàn điều khiển.
2. Sử dụng .get() để lấy giá trị của ID người dùng superStackSmash, với giá trị mặc định là 100000 nếu người dùng không tồn tại. Lưu trữ nó trong biến stack\_id. In stack\_id ra bàn điều khiển.

user\_ids = {"teraCoder": 100019, "pythonGuy": 182921, "samTheJavaMaam": 123112, "lyleLoop": 102931, "keysmithKeith": 129384}

**Xóa phím**

Đôi khi chúng ta muốn lấy một khóa và xóa nó khỏi từ điển. Hãy tưởng tượng chúng ta đang chạy một bản vẽ và có một từ điển khớp số vé với giải thưởng:

>>xổ số = {223842: "Gấu bông", 872921: "Vé hòa nhạc", 320291: "Giỏ quà", 412123: "Vòng cổ", 298787: "Máy làm mì ống"}

Khi nhận được số vé, chúng tôi muốn trả lại giải thưởng và cũng xóa cặp đó khỏi từ điển vì giải thưởng đã được trao rồi. Đối với điều này, chúng ta có thể sử dụng . nhạc pop (). Như trường hợp với . get (), chúng ta có thể cung cấp giá trị mặc định nếu khóa không tồn tại trong từ điển:

>>> raffle.pop(320291, "No Prize")   
"Giỏ quà"   
>>> xổ số   
{223842: "Teddy Bear", 872921: "Vé hòa nhạc", 412123: "Vòng cổ", 298787: "Máy làm mì ống"}   
>>> raffle.pop(100000, "No Prize")   
"No Prize"   
>>> xổ số   
{223842: "Teddy Bear", 872921: "Vé hòa nhạc", 412123: "Vòng cổ", 298787: "Máy làm mì ống"}   
>>> raffle.pop(872921, "No Prize")   
"Vé hòa nhạc"   
>>> xổ số   
{223842: "Teddy Bear", 412123: "Vòng cổ", 298787: "Máy làm mì ống"}

**Nhận tất cả các phím**

Đôi khi chúng ta cần thao tác trên tất cả các khóa trong từ điển. Ví dụ: nếu chúng ta có một từ điển về học sinh trong một lớp toán và điểm của các em:

>>test\_scores = {"Grace":[80, 72, 90], "Jeffrey":[88, 68, 81], "Sylvia":[80, 82, 84], "Pedro":[98, 96, 95], "Martin":[78, 80, 78], "Dina":[64, 60, 75]}

Chúng tôi muốn có được danh sách học sinh trong một lớp mà không tính đến điểm số của họ. Chúng ta có thể thực hiện việc này bằng cách sử dụng hàm list () có sẵn :

>>> list(test\_scores)   
["Grace", "Jeffrey", "Sylvia", "Pedro", "Martin", "Dina"]

Từ điển cũng có một phương pháp. phím (), trả về đối tượng dict\_keys . Đối tượng dict\_keys là một đối tượng khung nhìn cung cấp chế độ xem trạng thái hiện tại của từ điển mà người dùng không thể thay đổi bất cứ điều gì. Đối tượng dict\_keys được hàm trả về . phím (), đại diện cho một bộ khóa trong từ điển. Bạn không thể thêm hoặc xóa các thành phần khỏi đối tượng dict\_keys , nhưng nó có thể được sử dụng thay cho danh sách để lặp lại:

>>dành cho học sinh trong test\_scores.keys():   
>> print(student)

sẽ xuất ra :

Grace   
Jeffrey   
Sylvia   
Pedro   
Martin   
Dina

**Bài tập**

1. Tạo một biến có tên là user và gán cho nó một đối tượng dict\_keys cho tất cả các khóa trong từ điển user\_ids.

user\_ids = {"teraCoder": 100019, "pythonGuy": 182921, "samTheJavaMaam": 123112, "lyleLoop": 102931, "keysmithKeith": 129384}

num\_exercises = {"hàm": 10, "cú pháp": 13, "luồng điều khiển": 15, "vòng lặp": 22, "danh sách": 19, "lớp": 18, "từ điển": 18}

1. Tạo một biến có tên là các lớp và gán nó cho đối tượng dict\_keys cho tất cả các khóa trong từ điển num\_exercises.
2. In người dùng ra bàn điều khiển.
3. In bài học ra bảng điều khiển.

**Nhận tất cả các giá trị**

Từ điển có phương thức .values() trả về một đối tượng dict\_values (giống như đối tượng dict\_keys, nhưng dành cho các giá trị!) với tất cả các giá trị trong từ điển. Nó có thể được sử dụng thay cho danh sách lặp lại:

>>test\_scores = {"Grace":[80, 72, 90], "Jeffrey":[88, 68, 81], "Sylvia":[80, 82, 84], "Pedro":[98, 96, 95], "Martin":[78, 80, 78], "Dina":[64, 60, 75]}   
  
>>for Score\_list trong test\_scores.values():   
>> print(score\_list)

Sẽ cho:

[80, 72, 90]   
[88, 68, 81] [   
80, 82, 84]   
[98, 96, 95]   
[78, 80, 78]   
[ 64, 60, 75]

Không có hàm tích hợp nào để lấy tất cả các giá trị dưới dạng danh sách, nhưng nếu thực sự muốn, bạn có thể sử dụng:

>>danh sách(test\_scores.values())

**Bài tập**

1. Tạo một biến có tên Total\_exercises và đặt nó thành 0.
2. Xem qua các giá trị trong danh sách num\_exercises và thêm từng giá trị vào biến Total\_exercises.
3. In biến Total\_exercises ra bảng điều khiển.

>>num\_exercises = {"hàm": 10, "cú pháp": 13, "luồng điều khiển": 15, "vòng lặp": 22, "danh sách": 19, "lớp": 18, "từ điển": 18}

**Nhận tất cả các khóa và giá trị**

Bạn có thể lấy cả khóa và giá trị bằng cách sử dụng . mặt hàng (). Tương tự. phím () và . value (), nó trả về một đối tượng dict\_list . Mỗi phần tử dict\_list được trả về bởi . items () là một bộ dữ liệu bao gồm:

(khóa, giá trị)

vì vậy để lặp lại bạn có thể sử dụng cú pháp này:

>>biggest\_brands = {"Apple": 184, "Google": 141,7, "Microsoft": 80, "Coca-Cola": 69,7, "Amazon": 64,8}   
  
>>đối với công ty, giá trị trong big\_brands.items():   
>> print(company + " có giá trị là " + str(value) + " tỷ đô la. ")

TRONG kết quả sẽ đầu ra ::

Apple có giá trị 184 tỷ USD.   
Google có giá trị 141,7 tỷ USD.   
Microsoft có giá trị 80 tỷ USD.   
Coca-Cola có giá trị 69,7 tỷ USD.   
Amazon có giá trị 64,8 tỷ USD.

**Phần kết luận**

Trong bài học này, bạn đã học cách xem qua từ điển cũng như truy cập các khóa và giá trị theo nhiều cách khác nhau. Đặc biệt, bạn đã học được cách:

* Sử dụng khóa để lấy giá trị từ từ điển
* Kiểm tra chìa khóa
* Xóa cặp khóa:giá trị khỏi từ điển
* Lặp qua các khóa và giá trị trong từ điển

**Bài tập**

1. Chúng tôi đã cung cấp một bộ bài tarot. Bạn sắp rút được ba lá bài từ bộ bài về quá khứ, hiện tại và tương lai của bạn.

tarot = { 1: "The Magician", 2: "The High Priestess", 3: "The Empress", 4: "The Emperor", 5: "The Hierophant", 6: "The Lovers", 7: "The Xe ngựa", 8: "Sức mạnh", 9: "The Hermit", 10: "Bánh xe may mắn", 11: "Công lý", 12: "Người treo cổ", 13: "Cái chết", 14: "Sự ôn hòa", 15: "The Devil", 16: "The Tower", 17: "The Star", 18: "The Moon", 19: "The Sun", 20: "Judgement", 21: "The World", 22: "Kẻ ngốc"}

Tạo một từ điển trống có tên là spread.

1. Lá bài đầu tiên bạn rút là lá bài 13. Lấy giá trị được gán cho phím 13 từ từ điển Tarot và gán nó làm giá trị cho phím quá khứ.
2. Lá bài thứ hai bạn rút là lá bài 22. Lấy giá trị được gán cho phím 22 từ từ điển Tarot và gán nó làm giá trị cho phím “hiện tại”.
3. Lá bài thứ ba bạn rút là lá bài 10. Lấy giá trị được gán cho phím 10 từ từ điển Tarot và gán nó làm giá trị cho phím “tương lai”.

Tập tin

**Đọc một tập tin**

Máy tính sử dụng hệ thống tập tin để lưu trữ và truy xuất dữ liệu. Mỗi tệp là một thùng chứa riêng biệt với thông tin liên quan. Nếu bạn đã từng lưu tài liệu, tải xuống một bài hát hoặc thậm chí gửi email thì bạn đã tạo một tệp ở đâu đó trên máy tính. Ngay cả script.py, chương trình Python bạn đang chỉnh sửa, cũng là một tệp.

Vậy làm cách nào để chúng ta tương tác với các tệp bằng Python? Chúng ta sẽ học cách đọc và ghi các loại tệp khác nhau bằng mã. Giả sử chúng ta có một tệp có tên real\_cool\_document.txt với nội dung sau:

real\_cool\_document.txt

Tuyệt vời!

Chúng ta có thể đọc tập tin này như thế này:

với open('real\_cool\_document.txt') là cool\_doc:

cool\_contents = cool\_doc.read()

in(cool\_contents)

Thao tác này sẽ mở một đối tượng tệp có tên cool\_doc và tạo một khối mới trong đó bạn có thể đọc nội dung của tệp đang mở. Sau đó, chúng tôi đọc nội dung của tệp cool\_doc bằng cách sử dụng cool\_doc.read() và lưu chuỗi kết quả vào biến cool\_contents. Sau đó, chúng tôi xuất ra cool\_contents, xuất ra hướng dẫn Wowsers!

**Lặp lại các hàng**

Khi chúng ta đọc một tệp, chúng ta có thể muốn lấy toàn bộ tài liệu trên một dòng, chẳng hạn như với phương thức .read(). Nhưng nếu chúng ta muốn lưu trữ mỗi hàng trong một biến thì sao? Chúng ta có thể sử dụng hàm .readlines() để đọc từng dòng tệp văn bản thay vì đọc toàn bộ nội dung cùng một lúc. Hãy nói​ chúng ta có tài liệu :

keats\_sonnet.txt

Gửi đến người đã bị dồn nén lâu ngày trong thành phố,

'Thật thú vị khi được nhìn vào hội chợ

Và khuôn mặt rộng mở của thiên đường—để thở ra một lời cầu nguyện

Tràn đầy nụ cười của bầu trời xanh.

script.py

>>với open('keats\_sonnet.txt') là keats\_sonnet:

>> cho dòng trong keats\_sonnet.readlines():

>> in(dòng)

Đoạn script trên tạo một đối tượng tệp tạm thời, keats\_sonnet, trỏ đến tệp keats\_sonnet.txt. Sau đó, nó lặp qua từng dòng trong tài liệu và in toàn bộ tệp ra bàn điều khiển.

**Đọc một dòng**

Đôi khi bạn không cần phải duyệt qua toàn bộ tệp. Có một phương thức tệp khác cho việc này, .readline(), sẽ chỉ đọc một dòng mỗi lần. Nếu toàn bộ tài liệu được đọc từng dòng theo cách này, các lệnh gọi tiếp theo tới .readline() sẽ không gây ra lỗi nhưng sẽ bắt đầu trả về một dòng trống (""). Hãy giả sử chúng ta có như là tài liệu :

millay\_sonnet.txt

Anh sẽ quên em ngay bây giờ, em yêu,

Vì thế hãy tận dụng tối đa điều này, ngày nhỏ của bạn,

Tháng nhỏ của bạn, nửa năm nhỏ bé của bạn,

Nếu tôi quên, hoặc chết, hoặc rời đi,

script.py

>>với open('millay\_sonnet.txt') là sonnet\_doc:

>> first\_line = sonnet\_doc.readline()

>> giây\_line = sonnet\_doc.readline()

>> in(second\_line)

Tập lệnh này cũng tạo một đối tượng tệp có tên sonnet\_doc trỏ đến tệp millay\_sonnet.txt. Sau đó, nó đọc dòng đầu tiên bằng cách sử dụng sonnet\_doc.readline() và lưu nó vào biến first\_line. Sau đó, nó lưu dòng thứ hai (vì vậy hãy tận dụng tối đa điều này, ngày nhỏ của bạn) vào biến two\_line rồi in ra.

**Viết một tập tin**

Đọc một tập tin là tốt, nhưng nếu chúng ta muốn tạo tập tin của riêng mình thì sao? Với Python chúng ta có thể làm được điều này. Hóa ra hàm open() mà chúng ta sử dụng để mở tệp để đọc, cần một đối số khác để mở tệp để ghi.

script.py

>>với open('generated\_file.txt', 'w') là gen\_file:

>> gen\_file.write("Thật là một tập tin đáng kinh ngạc!")

Ở đây chúng ta chuyển đối số 'w' cho hàm open() để biểu thị việc mở tệp ở chế độ ghi. Đối số mặc định là 'r' và chuyển 'r' tới open() sẽ mở tệp ở chế độ đọc, giống như chúng ta đã làm.

Mã này tạo một tệp mới trong cùng thư mục với script.py và đặt cho nó dòng chữ "Thật là một tệp đáng kinh ngạc!" Điều quan trọng cần lưu ý là nếu đã có tệp có tên generate\_file.txt, nó sẽ ghi đè hoàn toàn tệp đó, xóa tất cả nội dung đã có trước đó.

**Thêm vào tập tin**

Vì vậy, bạn có thể lo lắng về việc xóa và ghi đè hoàn toàn các tệp hiện có. Không thể chỉ thêm một dòng vào tệp mà không xóa nó hoàn toàn sao? Tất nhiên là có! Thay vì mở tệp với đối số "w" cho chế độ ghi, chúng ta mở tệp bằng "a" cho chế độ chắp thêm. Nếu chúng tôi có một tệp được tạo có nội dung sau:

đã tạo\_file.txt

Đây là một tập tin phổ biến...

Sau đó chúng ta có thể thêm một dòng khác vào tệp này với đoạn mã sau:

script.py

>>với open('generated\_file.txt', 'a') là gen\_file:

>> gen\_file.write("...và nó vẫn vậy")

Trong đoạn mã trên, chúng ta đang mở một đối tượng tệp trong biến tạm thời gen\_file. Biến này trỏ đến tệp create\_file.txt và vì nó được mở ở chế độ chắp thêm nên sẽ thêm một dòng... và vẫn là một dòng mới trong tệp. Nếu bạn mở tệp sau khi chạy tập lệnh, nó sẽ trông như thế này:

đã tạo\_file.txt

Đây là một tập tin phổ biến...

...và nó vẫn vậy

S

Lưu ý rằng việc mở tệp ở chế độ chắp thêm với 'a' làm đối số cho hàm open() có nghĩa là việc sử dụng phương thức .write() của đối tượng tệp sẽ nối thêm bất cứ thứ gì được chuyển vào cuối tệp trên một dòng mới. Nếu chúng ta chạy lại script.py, nó sẽ trông như thế này: create\_file.txt:

đã tạo\_file.txt

Đây là một tập tin phổ biến...

...và nó vẫn vậy

...và nó vẫn vậy

Xin lưu ý rằng chúng tôi đã thêm "... và nó vẫn như vậy" vào tệp lần thứ hai! Điều này là do trong script.py chúng tôi đã mở tệp create\_file.txt ở chế độ chắp thêm.

**"với"**

Cho đến nay chúng ta đã mở các tệp này bằng cách sử dụng khối with này, nhưng có vẻ hơi lạ khi chúng ta chỉ có thể sử dụng biến tệp của mình trong một khối thụt lề. Tại sao lại thế này? Từ khóa with gọi cái gọi là trình quản lý bối cảnh cho tệp, trên đó chúng tôi gọi open(). Trình quản lý bối cảnh này đảm nhiệm việc mở tệp khi chúng ta gọi open() và sau đó đóng tệp sau khi chúng ta thoát khỏi khối thụt lề.

Tại sao việc đóng một tập tin lại khó đến vậy? Chà, hầu hết các khía cạnh khác trong mã của chúng tôi đều liên quan đến những gì Python tự kiểm soát. Tất cả các biến bạn tạo—số nguyên, danh sách, từ điển—đều là đối tượng Python và Python biết cách xóa chúng khi làm việc với chúng. Vì các tệp của bạn tồn tại bên ngoài tập lệnh Python nên chúng tôi cần thông báo cho Python khi chúng tôi xử lý xong chúng để nó có thể đóng kết nối với tệp đó. Việc để kết nối tới một tệp mở một cách không cần thiết có thể ảnh hưởng đến hiệu suất hoặc ảnh hưởng đến các chương trình khác trên máy tính của bạn có thể đang cố truy cập vào tệp đó.

Cú pháp with thay thế các cách truy cập tệp cũ, trong đó bạn phải gọi thủ công .close() trên một đối tượng tệp. Chúng ta vẫn có thể mở một tệp và thêm cú pháp cũ vào đó, miễn là chúng ta nhớ đóng kết nối tới tệp sau khi thực hiện việc đó.

fun\_cities\_file = open('fun\_cities.txt', 'a')

# Bây giờ chúng ta có thể thêm một dòng vào "fun\_cities".

fun\_cities\_file.write("Montréal")

# Nhưng chúng ta cần nhớ đóng file lại

fun\_cities\_file.close()

Trong tập lệnh trên, chúng tôi đã thêm "Montréal" làm dòng mới trong tệp fun\_cities.txt của mình. Tuy nhiên, vì chúng tôi đang sử dụng cú pháp kiểu cũ nên chúng tôi phải nhớ đóng tệp sau đó. Vì nó nhất thiết phải dài dòng hơn (yêu cầu ít nhất một dòng mã nữa) nhưng không mang tính biểu cảm hơn nên sử dụng with được ưu tiên hơn.

**Tệp CSV là gì?**

Các tệp văn bản không phải là thứ duy nhất Python có thể đọc, nhưng chúng là thứ duy nhất chúng ta không cần thêm thư viện phân tích cú pháp để hiểu. Tệp CSV là một ví dụ về tệp văn bản áp đặt cấu trúc trên dữ liệu của nó. CSV là viết tắt của Giá trị được phân tách bằng dấu phẩy và tệp CSV thường được sử dụng để xuất dữ liệu từ phần mềm bảng tính (chẳng hạn như Microsoft Excel hoặc Google Trang tính) sang định dạng di động. Một bảng trông như thế này:

|  |  |  |
| --- | --- | --- |
| Tên | Tên người dùng | E-mail |
| Roger Smith | rsmith | wigginsryan@yahoo.com |
| Michelle Beck | mlbeck | hcosta@hotmail.com |
| Ashley Barker | a\_bark\_x | a \_ sủa \_ x @ Turner . com |
| Lynn Gonzales | goodmanjames | lynniegonz@hotmail.com |

Trong tệp CSV, dữ liệu tương tự sẽ được hiển thị như sau:

người dùng.csv

Tên, Tên người dùng, Email

Roger Smith,rsmith,wigginsryan@yahoo.com

Michelle Beck, mlbeck, hcosta@hotmail.com

Ashley Barker,a\_bark\_x,a\_bark\_x@turner.com

Lynn Gonzales,goodmanjames,lynniegonz@hotmail.com

Lưu ý rằng dòng đầu tiên của tệp CSV không thực sự đại diện cho bất kỳ dữ liệu nào mà chỉ có các nhãn dữ liệu có trong phần còn lại của tệp. Các dòng còn lại của tệp chỉ giống như trong chương trình bảng tính thay vì được phân tách thành các ô khác nhau, chúng được phân tách bằng... à, tôi đoán công bằng mà nói chúng được phân tách bằng dấu phẩy.

**Đọc tệp CSV**

người dùng.csv

Tên, Tên người dùng, Email

Roger Smith,rsmith,wigginsryan@yahoo.com

Michelle Beck, mlbeck, hcosta@hotmail.com

Ashley Barker,a\_bark\_x,a\_bark\_x@turner.com

Lynn Gonzales,goodmanjames,lynniegonz@hotmail.com

Mặc dù chúng ta có thể dễ dàng đọc các chuỗi này dưới dạng văn bản nhưng vẫn có nhiều cách để truy cập dữ liệu ở định dạng phù hợp hơn cho mục đích lập trình. Trong Python, chúng ta có thể chuyển đổi dữ liệu này thành từ điển bằng đối tượng DictReader của thư viện CSV. Đây là cách chúng tôi tạo danh sách địa chỉ email của tất cả người dùng trong bảng trên:

>>nhập csv

>>list\_of\_email\_addresses = []

>>với open('users.csv', newline='') là user\_csv:

>> user\_reader = csv.DictReader(users\_csv)

>> cho hàng trong user\_reader:

>> list\_of\_email\_addresses.append(row['Email'])

Trong đoạn mã trên, trước tiên chúng tôi nhập thư viện csv để cung cấp cho chúng tôi các công cụ để phân tích cú pháp tệp CSV. Sau đó, chúng tôi tạo một danh sách trống, list\_of\_email\_addresses, sau này chúng tôi sẽ điền các địa chỉ email từ CSV của mình. Sau đó mở tệp user.csv với biến tạm thời user\_csv.

Chúng tôi chuyển một đối số từ khóa bổ sung newline = '' cho hàm open() của tệp để đảm bảo rằng chúng tôi không vô tình nhầm ngắt dòng ở một trong các trường dữ liệu với dòng mới trong CSV của chúng tôi (đọc tài liệu Python để biết thêm về điều này ).

Sau khi mở tệp CSV mới, chúng tôi sử dụng csv.DictReader(users\_csv) để chuyển đổi các dòng của tệp CSV thành từ điển Python mà chúng tôi có thể sử dụng các trình truy cập. Khóa từ điển mặc định là các mục ở hàng đầu tiên của tệp CSV của chúng tôi. Vì hàng đầu tiên của CSV gọi trường thứ ba trong CSV "Email" nên chúng tôi có thể sử dụng trường này làm khóa trong mỗi hàng của DictReader.

Khi lặp lại các hàng của đối tượng user\_reader, chúng tôi truy cập tất cả các hàng trong CSV dưới dạng từ điển (ngoại trừ hàng đầu tiên mà chúng tôi đã sử dụng để gắn nhãn các khóa từ điển của mình). Bằng cách truy cập khóa "Email" trong mỗi hàng này, chúng tôi có thể lấy địa chỉ email trong hàng đó và thêm nó vào list\_of\_email\_addresses của chúng tôi.

**Đọc các loại tệp CSV khác nhau**

Chúng tôi đã hành động như thể tệp CSV là tệp có giá trị được phân tách bằng dấu phẩy. Đúng là CSV có nghĩa như vậy, nhưng cũng đúng là ngày nay các cách phân tách giá trị khác đều được chấp nhận.

Mọi người thường gọi các tệp giá trị được phân tách bằng tab là tệp TSV, nhưng khi các dấu phân cách khác ngày càng phổ biến, mọi người đều nhận ra rằng việc tạo một định dạng tệp mới. [Az] sv cho mỗi ký tự phân cách được sử dụng không mạnh mẽ.

Vì vậy, chúng tôi gọi tất cả các tệp có danh sách các giá trị khác nhau vào một tệp CSV, sau đó sử dụng các dấu phân cách khác nhau (chẳng hạn như dấu phẩy hoặc tab) để cho biết vị trí bắt đầu và kết thúc của các giá trị khác nhau (tách giá trị).

Giả sử chúng ta có một sổ địa chỉ. Vì các địa chỉ thường sử dụng dấu phẩy nên chúng tôi sẽ cần sử dụng một dấu phân cách khác cho thông tin của mình. Vì không có dữ liệu nào của chúng tôi có dấu chấm phẩy (;), nên chúng tôi có thể sử dụng chúng.

địa chỉ.csv

Tên;Địa chỉ;Điện thoại

Donna Smith;126 Orr Corner Suite 857\nEast Michael, LA 54411;906-918-6560

Aaron Osborn;6965 Miller Station Suite 485\nNorth Michelle, KS 64364;815.039.3661x42816

Jennifer Barnett;8749 Alicia Vista Apt. 288\nHồ Victoriaberg, TN 51094;397-796-4842x451

Joshua Bryan;20116 Stephanie Stravenue\nWhitneytown, IA 87358;(380)074-6173

Andrea Jones;558 Melissa Keys Apt. 588\nBắc Teresahaven, WA 63411;+57(8)7795396386

Victor Williams;725 Gloria Lượt xem Suite 628\nEast Scott, IN 38095;768.708.3411x954

Lưu ý ký tự \n, đây là chuỗi thoát cho dòng mới. Khả năng thoát khỏi dòng mới trong dữ liệu của chúng tôi bằng ký tự \n là lý do chúng tôi chuyển đối số từ khóa newline='' cho hàm open().

Cũng lưu ý rằng nhiều địa chỉ trong số này có dấu phẩy! Không sao, chúng ta vẫn có thể đọc được nó. Ví dụ: nếu chúng tôi muốn in ra tất cả các địa chỉ trong tệp CSV này, chúng tôi có thể thực hiện các thao tác sau:

nhập csv

>> với open('addresses.csv', newline='') là địa chỉ\_csv:

>> address\_reader = csv.DictReader(addresses\_csv, delimiter=';')

>> cho hàng trong address\_reader:

>> print(row['Địa chỉ'])

Lưu ý rằng khi gọi csv.DictReader, chúng ta sẽ truyền tham số dấu phân cách, đây là một chuỗi được sử dụng để đánh dấu các trường riêng lẻ trong CSV. Sau đó, chúng tôi xem CSV và in ra từng địa chỉ.

**Viết tệp CSV**

Đương nhiên, nếu chúng tôi có khả năng đọc các tệp CSV khác nhau, chúng tôi muốn có thể ghi các tệp này để sử dụng thêm dữ liệu nhận được. Giả sử chúng tôi có một danh sách lớn dữ liệu mà chúng tôi muốn lưu vào tệp CSV. Chúng ta có thể làm như sau:

big\_list = [{'name': 'Fredrick Stein', 'userid': 6712359021, 'is\_admin': False}, {'name': 'Wiltmore Denis', 'userid': 2525942, 'is\_admin': False}, { 'name': 'Greely Plonk', 'userid': 15890235, 'is\_admin': False}, {'name': 'Dendris Stulo', 'userid': 572189563, 'is\_admin': True}]

>>nhập csv

>>với open('output.csv', 'w') là out\_csv:

>> trường = ['tên', 'userid', 'is\_admin']

>> out\_writer = csv.DictWriter(output\_csv, fieldnames=fields)

>> out\_writer.writeheader()

>> đối với mục trong big\_list:

>> out\_writer.writerow(item)

Trong mã ở trên, chúng tôi có một bộ từ điển có cùng khóa cho mỗi từ điển, một ứng cử viên chính cho CSV. Chúng tôi nhập thư viện csv rồi mở tệp CSV mới ở chế độ ghi bằng cách chuyển đối số w cho hàm open().

Sau đó, chúng tôi xác định các trường mà chúng tôi sẽ sử dụng thành một biến được gọi là các trường. Sau đó, chúng tôi khởi tạo đối tượng bản ghi CSV của mình và chuyển hai đối số. Đầu tiên là out\_csv, một đối tượng xử lý tập tin. Thứ hai là danh sách các trường trường mà chúng tôi chuyển vào tên trường tham số từ khóa.

Bây giờ chúng ta đã khởi tạo trình ghi tệp CSV của mình, chúng ta có thể bắt đầu thêm dòng vào chính tệp đó! Đầu tiên chúng ta cần các tiêu đề, vì vậy chúng ta gọi hàm writeheader() trên đối tượng ghi. Điều này ghi tất cả các tên trường được chuyển đến tên trường dưới dạng dòng đầu tiên trong tệp của chúng tôi. Sau đó, chúng tôi lặp lại danh sách dữ liệu lớn của mình. Mỗi phần tử trong big\_list là một từ điển với mỗi trường trong các trường là khóa. Chúng tôi gọi out\_writer.writerow() với các từ điển thành phần ghi từng dòng vào tệp CSV.

**Đọc tệp JSON**

CSV không phải là định dạng tệp duy nhất mà Python có thư viện tích hợp sẵn. Chúng ta cũng có thể sử dụng các công cụ tệp Python để đọc và ghi JSON. JSON, viết tắt của Ký hiệu đối tượng JavaScript, là định dạng tệp được lấy cảm hứng từ ngôn ngữ lập trình JavaScript.

Định dạng JSON rất giống với cú pháp từ điển Python, vì vậy các tệp JSON có thể dễ dàng đọc được từ góc nhìn của nhà phát triển Python. Tuy nhiên, Python đi kèm với gói json sẽ giúp chúng ta chuyển đổi các tệp JSON thành từ điển Python thực. Giả sử chúng ta có một tệp JSON như sau:

mua\_14781239.json

{

“người dùng”: “ellen\_greg”,

“hành động”: “mua hàng”,

“ item\_id ” : “ 14781239 ” ,

}

Chúng ta có thể đọc nó dưới dạng từ điển Python với đoạn mã sau:

json\_reader.py

>> nhập json

>>với open('purchase\_14781239.json') là buy\_json:

>> buy\_data = json.load(purchase\_json)

>>print(purchase\_data['user'])

Đầu tiên chúng ta nhập gói json. Chúng tôi đã mở tệp bằng lệnh open(). Vì chúng tôi đang mở nó ở chế độ đọc nên chúng tôi chỉ cần nhập tên tệp. Chúng tôi lưu tệp trong biến tạm thời Purchase\_json.

Chúng tôi tiếp tục phân tích cú pháp Purchase\_json bằng cách sử dụng json.load(), tạo một từ điển Python từ tệp. Lưu kết quả vào file buy\_data nghĩa là chúng ta có thể tương tác với nó. Chúng ta in một trong các giá trị tệp JSON bằng cách nhập đối tượng buy\_data.

**Viết một tệp JSON**

Đương nhiên, chúng ta cũng có thể sử dụng thư viện json để dịch các đối tượng Python sang JSON. Điều này đặc biệt hữu ích trong trường hợp bạn đang sử dụng thư viện Python để phân phát các trang web, bạn cũng có thể phân phát JSON. Giả sử chúng tôi có một từ điển Python mà chúng tôi muốn lưu dưới dạng tệp JSON:

Turn\_to\_json = {

'Id sự kiện': 674189,

'dateTime': '2015-02-12T09:23:17.511Z',

'sô cô la': 'Vị đậm vừa ngọt',

'isTomatoAFruit': Đúng

}

Chúng tôi sẽ có thể tạo tệp JSON với thông tin này bằng cách thực hiện như sau:

>> nhập json

>>với open('output.json', 'w') là json\_file:

>> json.dump(turn\_to\_json, json\_file)

Chúng ta nhập mô-đun json, mở tệp chế độ ghi dưới biến json\_file, sau đó sử dụng phương thức json.dump() để ghi vào tệp. json.dump() nhận hai đối số: đầu tiên là đối tượng dữ liệu, sau đó là đối tượng tệp bạn muốn lưu.

Nhiệm vụ cuối cùng

1. Tạo một tệp csv chứa thông tin sau:

|  |  |  |  |
| --- | --- | --- | --- |
| Tên đầy đủ | Chức danh | Ngày tuyển dụng | Lương |
| Ivanov Ivan Ivanovich | Giám đốc | 22/10/2013 | 250.000 |
| Sorokina Ekaterina Matveevna | Nhà phân tích | 12/03/2020 | 75.000 |
| Strukov Ivan Sergeevich | Lập trình viên cao cấp | 23/04/2012 | 150.000 |
| Korneeva Anna Igorevna | Lập trình viên chính | 22/02/2015 | 120.000 |
| Starchikov Sergey Anatolievich | Lập trình viên trẻ | 12/11/2021 | 50.000 |
| Butenko Artem Andreevich | Kiến trúc sư | 12/02/2010 | 200.000 |
| Savchenko Alina Sergeevna | Chuyên viên phân tích cao cấp | 13/04/2016 | 100.000 |

1. Ban quản lý quyết định trả thưởng cho Ngày Lập trình viên vào tháng 9 (ngày 13 tháng 9 là Ngày Lập trình viên). Tiền thưởng sẽ là 3% lương của mỗi lập trình viên. Thực hiện một chức năng sẽ tính phí bảo hiểm.
2. Ban quản lý quyết định trả tiền thưởng cho tất cả nhân viên nữ trước ngày 8 tháng 3 và cho 2000 nhân viên trước ngày 23 tháng 2. Triển khai chức năng tính tiền thưởng.
3. Nó đã được quyết định lập chỉ mục tiền lương của nhân viên. Đối với những nhân viên đã làm việc cho công ty hơn 10 năm, chỉ số sẽ bằng 7% tiền lương của họ, đối với những người khác - 5%.
4. Cần xây dựng lịch nghỉ phép cho nhân viên. Để làm được điều này, cần đưa vào danh sách riêng những nhân viên đã làm việc cho công ty trên 6 tháng. Thực hiện chức năng.
5. Triển khai các hàm viết bằng csv và json .

**Dành cho nâng cao:**

Thực hiện tất cả các nhiệm vụ tương tự, chỉ cần tạo các lớp Nhân viên , sẽ bao gồm các thuộc tính Họ, tên, họ hàng, chức vụ, ngày tuyển dụng, mức lương. Thực hiện đóng gói. Đừng quên getters và setters. Bên trong lớp, triển khai các phương thức từ bài tập trên.

Tạo hàm tính lương.

Tạo hàm xây dựng biểu đồ thanh hiển thị mức lương theo vị trí.

Tạo hàm tính các khoản giảm trừ thuế (thuế thu nhập, quỹ an sinh xã hội).