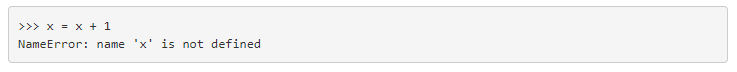
# Cập nhật các biến

Câu lệnh gán cập nhật một biến là một điều rất phổ biến, trong đó giá trị mới của biến phụ thuộc vào giá trị cũ.



Điều này có nghĩa là “lấy giá trị hiện tại của x, thêm 1, rồi cập nhật x với giá trị mới”.

Nếu bạn cố cập nhật một biến không tồn tại, bạn sẽ gặp lỗi vì Python đánh giá phía bên phải trước khi nó gán giá trị cho x:



Trước khi bạn có thể cập nhật một biến, bạn phải *khởi tạo* nó, thường thì chỉ cần một phép gán đơn giản:



Cập nhật một biến bằng cách thêm 1 được gọi là *increment* (*tăng*); trừ đi 1 được gọi là *decrement* (*giảm*).

# Câu lệnh while

Máy tính thường được sử dụng để tự động hóa các công việc lặp đi lặp lại. Tuy con người không giỏi việc lặp lại các tác vụ giống hệt nhau hoặc tương tự nhau và chúng ta thường mắc lỗi máy tính thì lại làm rất tốt điều này. Bởi vì việc lặp lại rất phổ biến, Python cung cấp một số tính năng ngôn ngữ để làm cho nó dễ dàng hơn.

Một dạng lặp trong Python là câu lệnh while. Đây là một chương trình đơn giản đếm ngược từ số 5 và sau đó nói "Blastoff!".



Bạn gần như có thể đọc câu lệnh while như thể nó là tiếng Anh. Nó có nghĩa là, "Khi n lớn hơn 0, hãy hiển thị giá trị của n và sau đó, giảm đi 1 từ giá trị của n. Khi n là 0, hãy thoát khỏi câu lệnh while và hiển thị từ Blastoff!"

Theo một cách chính thức hơn, đây là luồng thực thi câu lệnh while:

1. Đánh giá điều kiện, cho kết quả True hoặc False.
2. Nếu điều kiện sai, thoát khỏi câu lệnh while và tiếp tục thực thi ở câu lệnh tiếp theo.
3. Nếu điều kiện là đúng, hãy thực thi phần thân và sau đó quay lại bước 1.

Loại luồng này được gọi là *vòng lặp* bởi vì bước thứ ba lặp lại xung quanh bước đầu. Mỗi lần thực thi phần thân của vòng lặp được gọi là một *lần lặp*. Đối với vòng lặp trên, chúng ta sẽ nói, "Nó có 5 lần lặp", có nghĩa là phần thân của vòng lặp được thực thi năm lần.

Phần thân của vòng lặp sẽ thay đổi giá trị của một hoặc nhiều biến để cuối cùng điều kiện trở thành sai và vòng lặp kết thúc. Biến thay đổi mỗi khi vòng lặp thực thi và điều khiển khi nào vòng lặp kết thúc là được gọi là *biến lặp*. Nếu không có biến lặp, vòng lặp sẽ lặp lại mãi mãi, dẫn đến *vòng lặp vô hạn*.

# Vòng lặp vô hạn

Các lập trình viên rất hào hứng khi thấy hướng dẫn trên dầu gội, “Tạo bọt, xả, lặp lại”, đây là một vòng lặp vô hạn vì không có *biến lặp* cho bạn biết số lần thực thi vòng lặp.

Với countdown (đếm ngược), chúng ta có thể chứng minh rằng vòng lặp kết thúc bởi vì chúng ta biết rằng giá trị của n là hữu hạn và chúng ta có thể thấy rằng qua mỗi vòng lặp thì giá trị của n lại giảm đi, vì vậy, cuối cùng chúng ta sẽ tới 0. Trong những trường hợp khác, một vòng lặp rõ ràng là vô hạn vì nó không có biến lặp nào cả.

Đôi khi, phải tới giữa phần thân thì bạn mới biết đã đến lúc kết thúc một vòng lặp. Trong trường hợp đó, bạn có thể chủ ý viết một vòng lặp vô hạn và sau đó sử dụng câu lệnh break để nhảy ra khỏi vòng lặp.

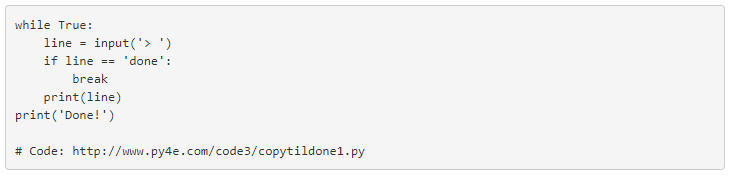
Vòng lặp này rõ ràng là một *vòng lặp vô hạn* vì biểu thức logic trên câu lệnh while đơn giản là hằng số logic True:



Nếu có lỗi trong code và bạn vẫn chạy code này thì chẳng mấy chốc, bạn sẽ học được cách dừng quá trình Python đang chạy trên hệ thống hoặc tìm vị trí nút tắt nguồn máy tính. Chương trình này sẽ chạy mãi mãi hoặc cho đến khi hết pin vì biểu thức logic ở đầu vòng lặp luôn đúng do biểu thức là giá trị hằng True.

Mặc dù đây là một vòng lặp vô hạn không có chức năng, chúng ta vẫn có thể sử dụng pattern (mô hình) này để xây dựng các vòng lặp hữu ích miễn là chúng ta viết thêm code một cách cẩn thận vào phần thân của vòng lặp để thoát khỏi vòng lặp một cách rõ ràng bằng cách sử dụng break khi chúng ta đã đạt được điều kiện thoát.

Ví dụ: bạn muốn lấy input (đầu vào) từ người dùng cho đến khi họ nhập done. Bạn có thể viết:



Điều kiện của vòng lặp là True (True thì luôn đúng), thì vòng lặp chạy lặp đi lặp lại cho đến khi nó tới câu lệnh break.

Mỗi lần, nó sẽ nhắc người dùng bằng một dấu ngoặc nhọn. Nếu người dùng nhập done, câu lệnh break sẽ thoát khỏi vòng lặp. Nếu không, chương trình sẽ lặp lại bất kỳ thứ gì người dùng nhập và quay trở lại đầu vòng lặp. Đây là một lần chạy mẫu:



Cách viết vòng lặp while này rất phổ biến vì bạn có thể kiểm tra điều kiện ở bất kỳ đâu trong vòng lặp (không chỉ ở trên cùng) và bạn có thể diễn đạt điều kiện dừng một cách khẳng định (“dừng khi điều này xảy ra”) thay vì phủ định (“tiếp tục cho đến khi điều đó xảy ra.").

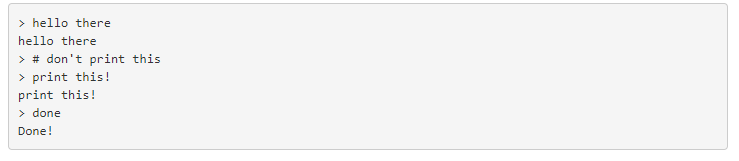
# Kết thúc các lần lặp với continue

Đôi khi bạn đang lặp lại một vòng lặp và muốn kết thúc lần lặp hiện tại và ngay lập tức chuyển sang lần lặp tiếp theo. Trong trường hợp đó, bạn có thể sử dụng câu lệnh continue để chuyển tới lần lặp tiếp theo mà không cần kết thúc phần thân của vòng lặp cho lần lặp hiện tại.

Đây là một ví dụ về một vòng lặp sao chép input của nó cho đến khi người dùng nhập "done", nhưng nó xử lý các dòng bắt đầu với dấu thăng như các dòng không được in (giống như các comment (chú thích) trong Python).



Đây là bản chạy mẫu của chương trình mới này khi đã thêm continue.



Tất cả các dòng được in ngoại trừ dòng bắt đầu bằng dấu thăng vì khi continue được thực thi, nó sẽ kết thúc lần lặp hiện tại và nhảy trở lại câu lệnh while để bắt đầu lần lặp tiếp theo, do đó, nó bỏ qua câu lệnh print.

# Các vòng lặp xác định sử dụng for

Đôi khi chúng ta muốn lặp qua một *tập hợp* những thứ như danh sách các từ, các dòng trong file hoặc danh sách các số. Khi chúng ta có một danh sách gồm những thứ cần lặp qua, chúng ta có thể tạo một *vòng lặp xác định* bằng cách sử dụng câu lệnh for. Chúng ta gọi câu lệnh while là một *vòng lặp không xác định* bởi vì nó chỉ lặp lại cho đến khi một điều kiện trở thành False, còn vòng lặp for lặp lại qua một tập hợp các mục đã biết nên có bao nhiêu thứ trong tập hợp thì nó chạy qua bấy nhiêu lần lặp.

Cú pháp của vòng lặp for tương tự như vòng lặp while ở chỗ có câu lệnh for và thân vòng lặp:



Theo thuật ngữ Python, biến friends là một danh sách gồm ba string (xâu) và vòng lặp for đi qua danh sách và thực thi phần thân một lần cho mỗi string trong 3 string trong danh sách, dẫn đến output này:



Dịch vòng lặp for này sang tiếng Anh sẽ không trực tiếp như while, nhưng nếu bạn coi bạn bè là một tập hợp, thì nó sẽ như thế này: “Chạy các câu lệnh trong phần thân của vòng lặp for một lần cho mỗi người bạn *trong* tập hợp có tên là friends.”

Nhìn vào vòng lặp for, *for* và *in* là các từ khóa dành riêng của Python, và friend và friends là các biến.



Cụ thể, friend là *biến lặp* của vòng lặp for. Biến friend thay đổi với mỗi lần lặp lại của vòng lặp và kiểm soát khi nào vòng lặp for hoàn thành. *Biến lặp* bước liên tiếp qua 3 string được lưu trữ trong biến friends.

# Các pattern của vòng lặp

Thường thì, chúng ta sử dụng vòng lặp for hoặc while để xem danh sách các mục hoặc nội dung của file và khi chúng ta đang tìm kiếm thứ gì đó chẳng hạn như giá trị lớn nhất hoặc nhỏ nhất của dữ liệu mà chúng ta đang quét qua.

Các vòng này thường được xây dựng bằng các bước như sau:

* Khởi tạo một hoặc nhiều biến trước khi bắt đầu vòng lặp
* Thực hiện tính toán trên từng mục trong thân vòng lặp, có thể thay đổi các biến trong phần thân của vòng lặp
* Nhìn vào các biến kết quả khi vòng lặp hoàn thành

Chúng ta sẽ sử dụng một danh sách các số để biểu diễn các khái niệm và cấu trúc của các pattern của vòng lặp này.

# Các vòng lặp đếm và vòng lặp tính tổng

Ví dụ, để đếm số lượng mục trong một danh sách, chúng ta sẽ viết vòng lặp for như sau:

Chúng ta đặt biến count thành 0 trước khi bắt đầu vòng lặp, sau đó, chúng ta viết một vòng lặp for để chạy qua danh sách các số. Biến *lặp* được đặt tên là itervar và dù chúng ta không sử dụng itervar trong vòng lặp, nó kiểm soát vòng lặp và khiến phần thân của vòng lặp được thực thi một lần cho mỗi giá trị trong danh sách.

Trong phần thân của vòng lặp, với mỗi giá trị trong danh sách, chúng ta thêm 1 vào giá trị hiện tại của count. Trong khi vòng lặp đang thực thi, giá trị của count là số lượng giá trị mà chúng ta đã thấy “cho đến nay”.

Khi vòng lặp hoàn tất, giá trị của count là tổng số giá trị. Tổng số giá trị mà chúng ta nhận được ở cuối vòng lặp. Chúng ta xây dựng vòng lặp để nhận những gì chúng ta muốn khi vòng lặp kết thúc.

Một vòng lặp tương tự để tính tổng một tập hợp số như sau:



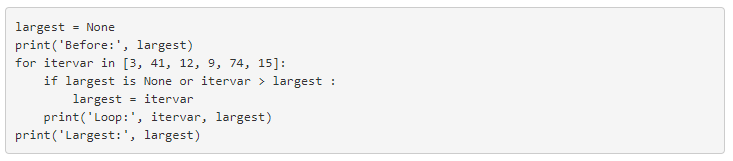
Trong vòng lặp này, chúng ta sử dụng *biến lặp*. Thay vì chỉ thêm 1 vào count như trong vòng lặp trước, chúng ta thêm con số thực sự (3, 41, 12, v.v.) vào tổng số đang chạy (running total) trong mỗi lần lặp của vòng lặp. Nếu bạn nghĩ về biến total, nó chứa "tổng số giá trị đang chạy cho đến nay". Vì vậy, trước khi vòng lặp bắt đầu, total bằng 0 vì chúng ta chưa thấy bất kỳ giá trị nào, trong suốt vòng lặp, total là tổng số đang chạy và ở cuối vòng lặp, total là tổng của tất cả giá trị trong danh sách.

Khi vòng lặp thực thi, total tích lũy tổng của các phần tử; đôi khi, một biến được sử dụng theo cách này được gọi là *biến tích lũy (accumulator)*.

Cả vòng lặp đếm và vòng lặp tính tổng đều không thực sự hữu ích trong thực tế vì có các hàm tích hợp len() và sum() để tính toán số lượng mục trong danh sách và tổng số mục trong danh sách đó.

# Vòng lặp tối đa và tối thiểu

Để tìm giá trị lớn nhất trong một danh sách hoặc một chuỗi, chúng ta xây dựng vòng lặp sau:



Khi chương trình thực thi, chúng ta được output như sau:



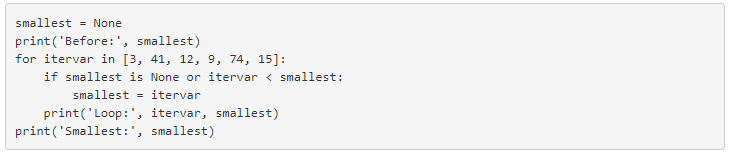
Tốt nhất, chúng ta coi biến largest là “giá trị lớn nhất mà chúng ta thấy cho đến nay”. Trước vòng lặp, chúng ta đặt largest thành hằng số None. None là một giá trị hằng số đặc biệt dùng để lưu trữ trong một biến để đánh dấu rằng biến đó là “rỗng”.

Trước khi vòng lặp bắt đầu, giá trị lớn nhất mà chúng ta thấy cho đến nay là None có vì chúng ta chưa thấy bất kỳ giá trị nào. Trong khi vòng lặp đang thực thi, nếu largest là None thì giá trị đầu tiên mà chúng ta thấy chính là giá trị lớn nhất cho đến nay. Bạn có thể thấy trong lần lặp đầu tiên khi giá trị của itervar là 3, vì largest là None, ngay lập tức, chúng ta có thể đặt largest là 3.

Sau lần lặp đầu tiên, largest không còn là None, vì vậy, phần thứ hai của biểu thức lôgic ghép kiểm tra itervar > largest chỉ kích hoạt khi chúng ta thấy một giá trị lớn hơn giá trị "lớn nhất cho đến nay". Khi chúng ta thấy một giá trị mới “thậm chí còn lớn hơn”, chúng ta lấy giá trị mới đó làm giá trị cho largest. Bạn có thể thấy trong output của chương trình, largest đi từ ​​3 đến 41 đến 74.

Ở cuối vòng lặp, chúng ta đã quét tất cả các giá trị và bây giờ, biến largest chứa giá trị lớn nhất trong danh sách.

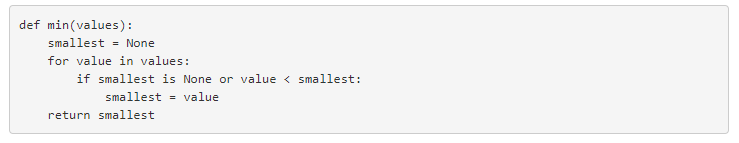
Để tính số nhỏ nhất, chúng ta chỉ cần thay đổi code một chút:



Một lần nữa, smallest là “nhỏ nhất cho đến nay” trước, trong và sau khi thực thi vòng lặp. Khi vòng lặp đã hoàn thành, smallest chứa giá trị nhỏ nhất trong danh sách.

Một lần nữa, cũng giống như vòng lặp đếm và tính tổng, vì đã có các hàm tích hợp max() và min(), chúng ta cũng không cần viết các vòng lặp này.

Sau đây là phiên bản đơn giản của hàm min() tích hợp sẵn trong Python:



Trong phiên bản hàm của code nhỏ nhất, chúng ta đã loại bỏ tất cả các câu lệnh print để chúng tương đương với hàm min đã được tích hợp sẵn trong Python.

1. **Gỡ lỗi**

Khi bạn bắt đầu viết các chương trình lớn hơn, bạn sẽ thấy mình dành nhiều thời gian hơn để gỡ lỗi. Nhiều code hơn có nghĩa là dễ tạo ra lỗi hơn và bug có nhiều nơi để trốn hơn.

Có một cách để giảm thời gian gỡ lỗi là “gỡ lỗi bằng cách chia nhỏ”. Ví dụ: nếu có 100 dòng trong chương trình và bạn kiểm tra từng dòng một, thì sẽ mất 100 bước.

Thay vào đó, hãy cố chia nhỏ vấn đề ra làm đôi. Nhìn vào giữa chương trình hoặc gần vị trí đó để tìm giá trị trung gian mà bạn có thể kiểm tra. Thêm một câu lệnh print (hoặc một cái gì đó khác có thể dùng để xác minh được) và chạy chương trình.

Nếu kiểm tra ở phần giữa không đúng, vấn đề phải nằm trong nửa đầu của chương trình. Nếu kiểm tra ở giữa là đúng, vậy, vấn đề nằm ở nửa sau của chương trình.

Mỗi khi bạn thực hiện kiểm tra như vậy, bạn đã giảm được một nửa số dòng cần phải tìm kiếm. Sau 6 bước (ít hơn 100 rất nhiều), ít nhất là trên lý thuyết, bạn sẽ giảm xuống được chỉ còn 1 hoặc 2 dòng code.

Trong thực tế, không phải lúc nào chúng ta cũng thấy rõ “phần giữa của chương trình” là gì và không phải lúc nào bạn cũng có thể kiểm tra được. Đừng đếm số dòng và tìm chính xác điểm ở giữa. Thay vào đó, hãy nghĩ về những nơi có thể có lỗi trong chương trình và những nơi có thể dễ dàng kiểm tra. Sau đó, chọn một vị trí mà bạn cho rằng khả năng xảy ra lỗi ở trước hoặc sau kiểm tra là giống nhau.