# Hàm

## Cách dùng function trong Python

Trong bài này chúng ta sẽ tìm hiểu về **function trong Python**, bạn sẽ được học khái niệm về function là gì? cách khai báo và đặt tên function như thế nào? Sau đó làm một vài ví dụ đơn giản để giúp bạn hiểu rõ hơn.

Trước khi học sâu vào function thì mình đưa ra một bất cập khi sử dụng cách code từ bài đầu đến giờ nhé. Giả sử bạn viết chương trình [kiểm tra số nguyên tố bằng Python](https://freetuts.net/ref/kiem-tra-so-nguyen-to-bang-python-404.html), và bạn sử dụng theo cách ở bài viết này thì đã thực hiện thành công. Tuy nhiên, nếu trong một chương trình mình cần kiểm tra số nguyên tố ở **100 vị trí** khác nhau thì bạn phải code **100 lần** như vậy. Điều này gây ra tình trạng code bị dư thừa, khó bảo dưỡng và nâng cấp sau này.

Vậy có cách nào giải quyết không? Thật may mắn là các ngôn ngữ lập trình hiện nay đều hỗ trợ function giúp bạn xử lý nhanh gọn.

**Mục lục**

* [1. Function là gì trong Python?](https://freetuts.net/function-trong-python-1704.html#goto-h2-0)
* [2. Cách gọi function trong Python](https://freetuts.net/function-trong-python-1704.html#goto-h2-1)
* [4. Ý nghĩa của docstring trong function](https://freetuts.net/function-trong-python-1704.html#goto-h2-2)
* [4. Hàm return trong Python](https://freetuts.net/function-trong-python-1704.html#goto-h2-3)
* [5. Phạm vi của biến trong function](https://freetuts.net/function-trong-python-1704.html#goto-h2-4)
* [6. Thực hành hàm (function) trong Python nâng cao](https://freetuts.net/function-trong-python-1704.html#goto-h2-5)
* [7. Hiểu rõ về tham số của Function trong Python](https://freetuts.net/function-trong-python-1704.html#goto-h2-6)
  + [Giá trị mặc định của tham số](https://freetuts.net/function-trong-python-1704.html#goto-h3-0)
  + [Truyền tham số theo key name](https://freetuts.net/function-trong-python-1704.html#goto-h3-1)
  + [Tham số không giới hạn](https://freetuts.net/function-trong-python-1704.html#goto-h3-2)
* [8. Lời kết](https://freetuts.net/function-trong-python-1704.html#goto-h2-7)

### Function là gì trong Python?

Trong Python, function hay còn gọi là hàm, nó là tổng hợp một đoạn code gồm một hoặc nhiều lệnh nhằm giải quyết một bài toán cụ thể nào đó.

Function ra đời giúp lập trình viên dễ dàng tối ưu chương trình hơn, bởi ta có thể sử dụng function nhiều lần mà không cần phải định nghĩa lại, ta gọi đây là hướng lập trình modun. Việc tái sử dụng này giúp code ngắn và gọn hơn rất nhiều.

**Cú pháp tạo function trong Python**:

|  |  |
| --- | --- |
| 1  2  3 | def function\_name(parameters):      """docstring"""      statement(s) |

Trong đó các thành phần được giải thích như sau:

* def là từ khóa bắt buộc dùng để tạo function, nó được đặt tại vị trí đầu tiên của function
* function\_name là tên của function mà bạn muốn đặt, ví dụ bạn viết function kiểm tra số lớn nhất thì có thể đặt tên là number\_max.
* docstring là phần mô tả ý nghĩa của function, cái này do bạn tự nhập vào để sau này dễ dàng hiểu ý nghĩa của function mà không cần phải đọc lại toàn bộ code.
* statement(s) là những dòng lệnh xử lý bên trong function
* parameters là các tham số truyền vào của function

**Ví dụ**

|  |  |
| --- | --- |
| 1  2  3 | def greet(name):      """Hàm này sẽ hiển thị câu chào!"""      print("Chào, " + name + ". Rất vui khi gặp bạn!") |

Quy tắc đặt tên của function giống như đặt tên biến vậy, nghĩa là dùng các chữ cái không dấu, không được chứa kí tự đặc biệt, có thể chứa số nhưng nó không được đứng ở vị trí đầu tiên.

Nếu bạn tạo nhiều function thì tên của chúng không được trùng nhau.

### Cách gọi function trong Python

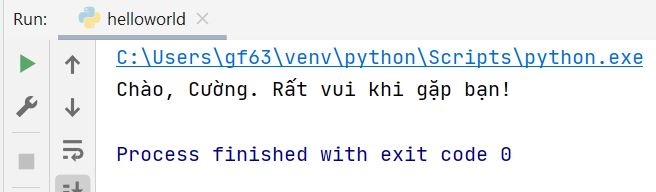
Trong Python, để gọi một function thì bạn chỉ việc nhập tên của function, sau đó truyền thêm các tham số nếu có. Bạn có thể gọi đến function ở bất kì đâu, miễn là trong phạm vi tồn tại của function.

Như ở ví dụ trên mình sẽ gọi đến function greet như sau:

|  |  |
| --- | --- |
| 1 | greet('Cường') |

**Kết quả**:

|  |  |
| --- | --- |
| 1 | Chào, Cường. Rất vui khi gặp bạn! |



Khác với các ngôn ngữ khác, bạn không cần khai báo kiểu dữ liệu cho các tham số truyền vào.

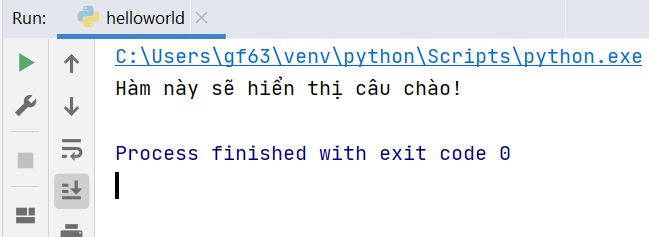
### Ý nghĩa của docstring trong function

Chuỗi đầu tiên sau tên hàm được gọi là docstring và viết tắt của document string. Nó được sử dụng để giải thích ngắn gọn ý nghĩa của function.

Mặc dù bạn có thể nhập docstring hoặc không cần cũng được. Tuy nhiên, document sẽ giúp bạn hiểu ý nghĩa của function mà không cần phải dò lại code, vì vậy với những lập trình viên có kinh nghiệm thì không bao giờ bỏ qua phần này.

Quay lại ví dụ trên, chúng ta có một docstring bên dưới tiêu đề hàm, và mình sử dụng đoạn code sau để xem ý nghĩa của hàm này.

|  |  |
| --- | --- |
| 1  2 | print(greet.\_\_doc\_\_)  => Hàm này sẽ hiển thị câu chào! |



Như bạn thấy, kết quả nó trả về chính là phần mà mình đã comment lúc khai báo hàm.

### Hàm return trong Python

Mỗi function có thể có giá trị trả về hoặc không. Câu hỏi đặt ra là khi nào nên sử dụng hàm có giá trị trả về và khi nào thì không? Để trả lời câu hỏi này thì bạn hãy làm một vài ví dụ trước nhé.

Quay lại ví dụ ở phần 1, đó là một function không có giá trị trả về bởi không tồn tại lệnh **return** bên trong hàm. Nhưng với function dưới đây thì khác:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | def absolute\_value(num):      """Hàm này sẽ trả về giá trị tuyệt đối của một số"""        if num >= 0:          return num      else:          return -num    # Output: 2  print(absolute\_value(2))    # Output: 4  print(absolute\_value(-4)) |

Function này có nhiệm vụ tính giá trị truyệt đối của một số, sau đó trả kết quả về, giá trị trả về chính là giá trị mà bạn đã gắn vào lệnh **return**. Vậy nếu bạn muốn viết một function mà có trả kết quả về thì sử dụng lệnh **return**.

Ví dụ bạn cần viết chương trình kiểm tra một số có phải là số nguyên tố hay không thì bạn có sử dụng lệnh return để trả về True hoặc False.

### Phạm vi của biến trong function

Trong function của Python bạn có thể tạo thêm bao nhiêu biến cũng được. Tuy nhiên, các biến đó chỉ tồn tại bên trong function mà thôi, bạn không thể sử dụng nó ở bên ngoài.

Hãy xem ví dụ dưới đây, mình đã tạo ra một function và biến x ở cả bên trong function và bên ngoài:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | def my\_func():      x = 10      print("Giá trị bên trong function:", x)    x = 20  my\_func()  print("Giá trị bên ngoài function:", x) |

Tuy nhiên kết quả trong function sẽ trả về giá trị 10 bởi vì phạm vi của function là cục bộ, nó không nhận được biến x ở bên ngoài. Kết quả sẽ như sau:

|  |  |
| --- | --- |
| 1  2 | Giá trị bên trong function: 10  Giá trị bên ngoài function: 20 |

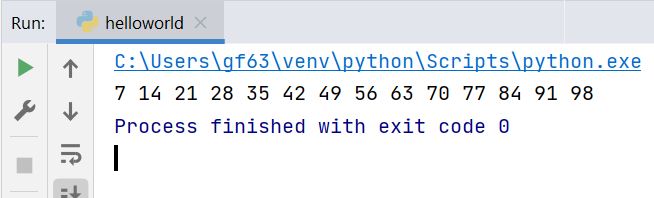
### Thực hành hàm (function) trong Python nâng cao

Bây giờ ta sẽ làm một vài ví dụ để bạn hiểu rõ hơn về cách sử dụng hàm trong Python nhé.

**Bài tập 1**: Viết chương trình in ra tất cả các số chia hết cho 7 từ 1 đến 100.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | # HỌC PYTHON TẠI FREETUTS.NET  # TÁC GIẢ: CƯỜNG NGUYỄN  # Function trong Python  # Tìm các số chia hết cho 7 từ 1 đến 100    def chia\_het\_cho\_7(number):      if number % 7 == 0:          return True      else:          return False      for i in range(1, 101):      if chia\_het\_cho\_7(i):          print(i, end=' ') |

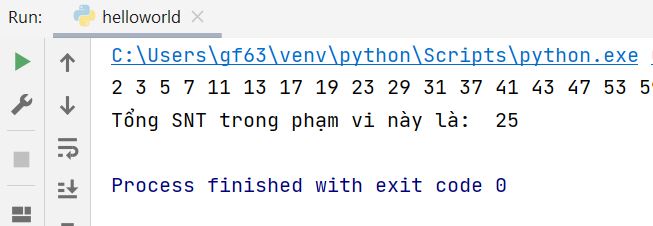
Kết quả:



**Bài tập 2**: Đếm tổng và in ra các số nguyên tố từ 1 đến 100.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28 | # HỌC PYTHON TẠI FREETUTS.NET  # TÁC GIẢ: CƯỜNG NGUYỄN  # Function trong Python  # Sử dụng hàm để kiểm tra SNT    def la\_SNT(n):      if (n < 2):          return False      elif (n == 2):          return True      elif (n % 2 == 0):          return False      else:          # Lặp qua các số lẻ nên bắt đầu từ 3 với bước nhảy là 2          for i in range(3, n, 2):              if (n % i == 0):                  return False      return True      counter = 0;    for i in range(1, 101):      if la\_SNT(i):          counter += 1          print(i, end=' ')  print("\_\_\_\_\_\_\_\_\_\_\_\_\_\_")  print("Tổng SNT trong phạm vi này là: ", counter) |

Kết quả:



### Hiểu rõ về tham số của Function trong Python

Trong phần này chúng ta sẽ học kỹ hơn về tham số truyền vào của hàm Python.

Trong Python bạn có thể tạo một hàm với nhiều tham số. Khi gọi hàm thì bắt buộc phải nhập giá trị cho tham số, tuy nhiên với những tham số có giá trị mặc định thì không cần.

#### Giá trị mặc định của tham số

Giá trị mặc định tức là nếu bạn không nhập giá trị lúc gọi hàm thì tham số sẽ lấy giá trị mặc định đó. Để tạo giá trị mặc định thì bạn dùng toán tử = và gán ngay lúc tạo hàm.

**Ví dụ**

|  |  |
| --- | --- |
| 1 | def greet(msg = "Good morning!"): |

Biến msg sẽ có giá trị mặc định là 'Good morning!". Hãy xem ví dụ đầy đủ sau để hiểu rõ hơn.

|  |  |
| --- | --- |
| 1  2  3  4  5 | def greet(name, msg = "Chào buổi sáng!"):     print("Xin chào ",name + ', ' + msg)    greet("Kate")  greet("Bruce","Bạn có khỏe không?") |

Ở đoạn gọi hàm mình đã gọi với hai cách khác nhau,

* greet("Kate") là cách gọi chỉ truyền vào một tham số nên function sẽ lấy giá trị mặc định cho tham số thứ hai.
* greet("Bruce","Bạn có khỏe không?") là các gọi đầy đủ nên giá trị mặc định không được sử dụng.

Chạy chương trình này thì kết quả sẽ như sau:

|  |  |
| --- | --- |
| 1  2 | Xin chào Kate, Chào buổi sáng!  Xin chào Bruce, Bạn có khỏe không? |

**Lưu ý**: Bạn không thể đặt một tham số có giá trị mặc định đằng trước những tham số không có giá trị mặc định, như ví dụ dưới đây là sai.

|  |  |
| --- | --- |
| 1 | def greet(msg = "Đặt tại đây là sai!", name): |

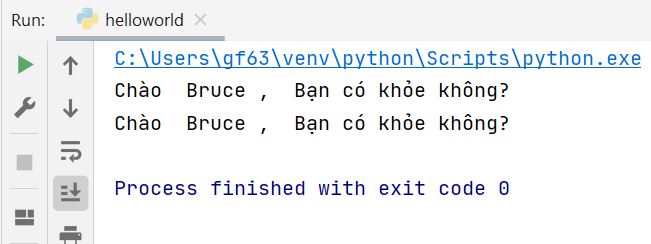
#### Truyền tham số theo key name

Truyền tham số theo **key**name tức là lúc gọi hàm và truyền tham số bạn sẽ nhập thêm tên của tham số, cách này có ưu điểm ở chỗ là bạn không cần phải nhớ chính xác thứ tự của các tham số, nhưng bù lại bạn phải nhớ tên của tham số.

Như ở ví dụ trên mình có thể gọi các cách như sau:

|  |  |
| --- | --- |
| 1  2  3  4  5 | def greet(name, msg):      print("Chào ", name, ', ', msg)    greet(name = "Bruce", msg = "Bạn có khỏe không?")  greet(msg = "Bạn có khỏe không?", name = "Bruce") |

Chạy đoạn code này thì có kết quả giống nhau.



#### Tham số không giới hạn

Tham số không giới hạn hay còn gọi là tham số Arbitrary. Khi sử dụng tham số này thì bạn có thể truyền bao nhiêu cũng được lúc gọi hàm.

Để tạo một tham số không giới hạn thì bạn thêm ký tự \* đằng trước tham số đó. Hãy xem ví dụ dưới đây.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | def greet(\*names):     """This function greets all     the person in the names tuple."""       # names is a tuple with arguments     for name in names:         print("Hello",name)    greet("Monica","Luke","Steve","John") |

Như bạn thấy, mình phải sử dụng vòng lặp để lặp qua từng tham số, và lúc gọi hàm thì nhập bao nhiêu cũng được.

**Chạy lên thì kết quả chương trình sẽ là**:

|  |  |
| --- | --- |
| 1  2  3  4 | Hello Monica  Hello Luke  Hello Steve  Hello John |

Đây là cách gọi hàm với nhiều đối số, khi bạn truyền vào thì nó sẽ tạo thành một Tuple nên bắt buộc phải sử dụng vòng lặp để lặp qua từng giá trị.

### Lời kết

Như vậy là bạn đã học được sơ lược về cách tạo hàm (function) trong Python. Vẫn còn rất nhiều vấn đề liên quan đến function nhưng mình sẽ viết nó ở một bài khác, việc phân tán bài học như vậy sẽ giúp bạn dễ dàng năm bắt hơn. Chúc bạn học tốt và hẹn gặp lại ở bài tiếp theo.

## Hàm đệ quy trong Python

Trong bài này mình sẽ nói đến một kiến thức rất hay và thuộc dạng nâng cao đó là xây dựng hàm **đệ quy trong Python**, đây là phần mình đã giải thích rất rõ ở trong series học [PHP căn bản](https://freetuts.net/hoc-php/hoc-lap-trinh-php-can-ban), tuy nhiên mình cũng sẽ trình bày sơ lược lại ở trong bài học này.

Để biết được cách tạo đệ quy thì trước tiên bạn phải hiểu khái niệm về **đệ quy là gì** đã nhém sau đó ta sẽ thực hành một vài ví dụ cơ bản nhất.

**Mục lục**

* [1. Đệ quy trong Python là gì?](https://freetuts.net/ham-de-quy-trong-python-1706.html#goto-h2-0)
* [2. Khử đệ quy trong Python](https://freetuts.net/ham-de-quy-trong-python-1706.html#goto-h2-1)
* [2. Ưu điểm và nhược điểm của đệ quy Python](https://freetuts.net/ham-de-quy-trong-python-1706.html#goto-h2-2)
* [3. Lời kết](https://freetuts.net/ham-de-quy-trong-python-1706.html#goto-h2-3)

### Đệ quy trong Python là gì?

Đệ quy trong Python hay còn gọi là **recursion python**. Nói về toán học thì đệ quy là thuật toán giải quyết bài toán bằng cách gọi lại chính thuật toán đó, thao tác này sẽ thực hiện liên tục cho đến khi gặp điều kiện dừng.

Đệ quy được thể hiện rất tốt khi áp dụng với hàm trong Python. Hàm đệ quy là chương trình sẽ gọi lại chính hàm đó và ngưng gọi khi gặp điều kiện dừng. Nếu quay lại kiến thức về vòng lặp thì bản chất đây cũng là một loại vòng lặp đặc biệt phải không các bạn.

*Bài viết này được đăng tại [free tuts .net]*

Chương trình đệ quy sẽ có điều kiện dừng, nếu không nó sẽ tạo ra một vòng đời đệ quy vô hạn, điều này giống như vòng lặp while trong Python là bạn đã được học.

Hãy làm một ví dụ đơn giản đó là tính giai thừa của một số.

Ví dụ sử dụng đệ quy để tính giai thừa của 4 thì sẽ là 1\*2\*3\*4 = 24.

**Bước 1**: yêu cầu người dùng nhập vào số 4

**Bước 2**: Sử dụng đệ quy để lặp tính tích từ 4 trở về 1. Gọi x là giá trị cho mỗi lần lặp thì ta có điều kiện dừng là x = 1.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | def calc\_factorial(x):      if x == 1:          return 1      else:          return (x \* calc\_factorial(x-1))    num = 4  print("Dãy fibo của ", num, "là ", calc\_factorial(num)) |

Bạn hãy để ý bên trong phần thân của hàm calc\_factorial nhé, điều kiện để dừng đệ quy là x == 1, ngược lại chương trình sẽ thực hiện lặp đệ quy bởi đoạn code return (x \* calc\_factorial(x-1)).

Quy trình hoạt động của nó như sau:

|  |  |
| --- | --- |
| 1  2  3  4  5 | calc\_factorial(4)              # Lần 1 gọi với số 4  4 \* calc\_factorial(3)          # Lần 2 gọi với số 3  4 \* 3 \* calc\_factorial(2)      # Lần 3 gọi với số 2  4 \* 3 \* 2 \* calc\_factorial(1)  # Lần 4 gọi với số 1  4 \* 3 \* 2 \* 1                  # Cuối cùng ta được chuỗi này => kết quả là 24 |

Lần gọi đên quy cuối cùng vì giá trị tham số x truyền vào là 1 nên sẽ không thực hiện đệ quy nữa, sau đó trả kết quả là **24**.

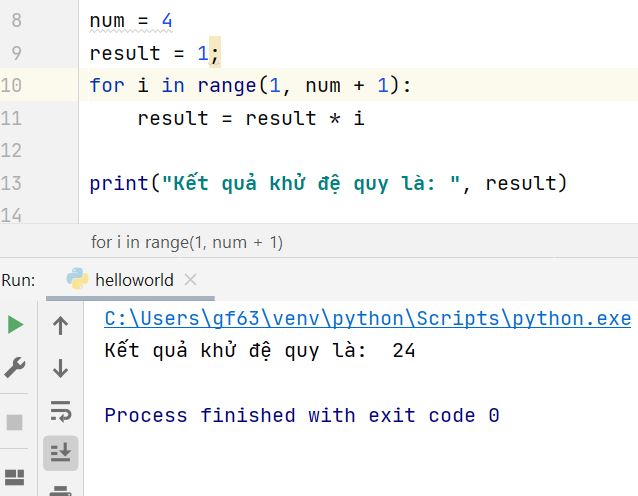
### Khử đệ quy trong Python

Thực tế thì sử dụng đệ quy sẽ tốn rất nhiều tài nguyên của máy tính, bởi nó sẽ phải lưu trữ khá nhiều thông tin để tạo ra một biểu thức cuối cùng. Vì vậy người ta thưởng sử dụng khử đệ quy để chuyển đổi từ đệ quy thành vòng lặp.

Như bài toán tính Fibo trên ta có thể sử dụng trong vòng lặp rất dễ dàng và nhanh chóng.

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | num = 4  result = 1;  for i in range(1, num + 1):      result = result \* i    print("Kết quả khử đệ quy là: ", result) |

Kết quả giống nhau:



### Ưu điểm và nhược điểm của đệ quy Python

Sau đây là một vài ưu điểm và nhược điểm của đệ quy trong lập trình Python.

**Ưu điểm**:

* Các hàm đệ quy làm cho mã trông sạch sẽ
* Một tác vụ phức tạp có thể được chia thành các vấn đề phụ đơn giản hơn bằng cách sử dụng đệ quy.
* Tạo trình tự dễ dàng với đệ quy hơn là sử dụng một số lần lặp lồng nhau.

**Nhược điểm**:

* Đôi khi logic đằng sau đệ quy rất khó theo dõi.
* Chi phí gọi đệ quy rất tốn kém (không hiệu quả) vì chúng chiếm rất nhiều bộ nhớ và thời gian.
* Các hàm đệ quy khó gỡ lỗi.

### Lời kết

Khi đi làm thực tế thì người ta rất ít khi chọn giải pháp đệ quy, trừ khi bắt buộc, bởi mất khá nhiều tài nguyên về bộ nhớ và thời gian để chạy một ứng dụng đệ quy, điều này là không tốt cho những ứng dụng trong thực tế. Bạn thử nghĩ xem nếu một website hoạt động quá chậm thì sẽ giảm đi phần trải nghiệm của người dùng một cách nghiêm trọng.

Có khá nhiều loại đệ quy như đệ quy tuyến tính, đệ quy hỗ tương, .. nhưng mình không đề cập đến trong bài viết, bạn tự tìm hiểu nhé.

## Hàm lambda trong Python

Trong bài viết này chúng ta sẽ tìm hiểu hàm **lambda trong Python**, nó còn được gọi là **hàm ẩn danh trong Python**. Đây là cách tạo hàm ngắn gọn và không có tên hàm.

Nếu bạn đã từng học các ngôn ngữ khác thì chắc chắn đã nghe qua khái niệm này rồi. Còn nếu bạn mới học Python lần đầu thì hãy tìm hiểu khái niệm lambda Python là gì đã nhé.

**Mục lục**

* [1. Lambda Python là gì?](https://freetuts.net/ham-lambda-trong-python-1707.html#goto-h2-0)
* [2. Tại sao phải sử dụng lambda trong Python?](https://freetuts.net/ham-lambda-trong-python-1707.html#goto-h2-1)
* [2. Một vài kết hợp của hàm lambda trong Python](https://freetuts.net/ham-lambda-trong-python-1707.html#goto-h2-2)
  + [Kết hợp với filter()](https://freetuts.net/ham-lambda-trong-python-1707.html#goto-h3-0)
  + [Kết hợp với map()](https://freetuts.net/ham-lambda-trong-python-1707.html#goto-h3-1)
* [3. Lời kết](https://freetuts.net/ham-lambda-trong-python-1707.html#goto-h2-3)

### Lambda Python là gì?

Lambda Python là một hàm ẩn danh được tạo ra mà không có tên hàm. Như chúng ta đã học, để tạo một hàm trong Python thì ta sử dụng từ khóa def ten\_ham, nhưng đối với hàm ẩn danh thì chúng ta sử dụng từ khóa lambda và đằng sau nó là danh sách các tham số và **một biểu thức**duy nhất.

Mặc dù các hàm bình thường được xác định bằng từ khóa def, nhưng trong các hàm ẩn danh Python được xác định bằng từ khóa lambda. Do đó, các hàm ẩn danh cũng được gọi là các hàm lambda.

Cú pháp tạo hàm lambda như sau:

|  |  |
| --- | --- |
| 1 | lambda arguments: expression |

Trong đó:

* lambda là từ khóa khai báo hàm lambda
* arguments là danh sách các tham số truyền vào hàm
* expression là biểu thức tính toán của hàm

Lambda function có thể có nhiều tham số, tuy nhiên nội dung bên trong của nó chỉ có một biểu thức mà thôi. Mỗi hàm lambda đều có giá trị trả về bởi biểu thức bên trong mà không cần từ khóa return.

Hãy xem ví dụ dưới đây:

|  |  |
| --- | --- |
| 1  2  3  4  5 | # Chương trình sử dụng lambda python function  double = lambda x: x \* 2    # Output: 10  print(double(5)) |

Trong ví dụ này mình đã tạo một lambda function có tên gọi là **double**, biểu thức bên trong của nó nhân hai giá trị của tham số truyên vào x \* 2.

Nếu viết theo hàm thông thường thì lambda function trên sẽ tương đương với hàm sau:

|  |  |
| --- | --- |
| 1  2 | def double(x):     return x \* 2 |

Như bạn thấy, đối với hàm lanbda thì chúng ta không cần từ khóa return vì mặc định biểu thức của hàm sẽ là giá trị return. Còn hàm thông thường thì khác, bạn phải sử dụng từ khóa return nếu muốn trả về một giá trị cho hàm.

Vậy câu hỏi đặt ra là tại sao lại sử dụng lambda function? Chúng ta cùng tìm hiểu qua phần 2 nhé.

### Tại sao phải sử dụng lambda trong Python?

Sức mạnh nổi bậc nhất khi sử dụng lambda đó là sử dụng hàm lambda trong một function bình thường khác.

Giả sử bạn có một function có một tham số x, bạn muốn viết một function có chức năng khi truyền x lần đầu xong thì lưu trạng thái function đó lại, để những lần gọi sau sẽ kế thừa giá trị x đó.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | def myfunc(n):    return lambda a : a \* n    # Biến mydoubler lúc này sẽ là một lambda function  mydoubler = myfunc(2)    # Vì vậy bạn có thể gọi thoải mái và nhiều lần ở nhièu vị trí  # Và vẫn kế thừa giá trị n của hàm myfunc  print(mydoubler(11)) # Kết quả 22  print(mydoubler(10)) # Kết quả 20 |

Qua ví dụ này thì bạn thấy rất dễ hiểu phải không nào? Lambda Python quả thật rất vi diệu.

### Một vài kết hợp của hàm lambda trong Python

Chúng ta sử dụng lambda function trong trường hợp cần một function trong thời gian ngắn, và thường được kết hợp với các hàm ở bậc cao hơn, chính xác hơn thì thường được sử dụng như một tham số.

Bây giờ hãy làm hai ví dụ thường thấy nhất khi sử dụng lambda function nhé, đó là kết hợp với **filter**và **map**.

#### Kết hợp với filter()

Hàm filter có công dụng là lọc dữ liệu theo tham số truyên vào, tham số đầu tiên là hàm kiểm tra điều kiện lọc, tham số thứ hai là dữ liệu cần lọc. Hãy xem ví dụ lọc các số chẵn trong list dưới đây:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | # Chương trình lấy những số chẵn trong danh sách  my\_list = [1, 5, 4, 6, 8, 11, 3, 12]  # Hàm lambda sẽ trả về True nếu là số chẵn, False nếu là số lẻ  new\_list = list(filter(lambda x: (x%2 == 0) , my\_list))    # Output: [4, 6, 8, 12]  print(new\_list) |

**Kết quả**:

|  |  |
| --- | --- |
| 1 | [4, 6, 8, 12] |

#### Kết hợp với map()

Hàm map có công dụng là lặp qua từng phần tử và thay đổi giá trị của nó dựa vào hai tham số, tham số đầu tiên là một hàm dùng để xử lý dữ liệu và trả kết quả về, tham số thứ hai là dữ liệu cần lặp.

Hãy xem ví dụ dưới đây, ví dụ này sẽ nhân đôi giá trị của các phần tử trong list.

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | # Chương trình nhân đôi giá trị các phần tử của List  my\_list = [1, 5, 4, 6, 8, 11, 3, 12]  new\_list = list(map(lambda x: x \* 2 , my\_list))    # Output: [2, 10, 8, 12, 16, 22, 6, 24]  print(new\_list) |

Chạy lên kết quả sẽ như sau:

|  |  |
| --- | --- |
| 1 | [2, 10, 8, 12, 16, 22, 6, 24] |

### Lời kết

Như vậy là mình đã giới thiệu sơ lược về hàm **lambda trong Python**. Ban phải hiểu rằng bản chất lambda function là một hàm ẩn danh, hoạt động trong một thời gian ngắn và tự mất đi khi phạm vi hoạt động của nó không còn nữa.

Trong thực tế chúng ta rất hay sử dụng lambda function, không chỉ ở Python mà ở hầu hết các ngôn ngữ hiện nay như PHP, Javascript đều hỗ trợ lambda function.

## Biến toàn cục và biến cục bộ trong Python

Trong bài này chúng ta sẽ tìm hiểu về **biến cục bộ** và **biến toàn cục trong Python**. Đây là hai phạm vi hoạt động chủ yếu của một biến trong Python, vì vậy bạn phải hiểu để sủ dụng cho đúng.

Đọc đến bài này thì chắc chắn bạn đã hiểu cách tạo hàm trong Python rồi nhỉ? Bạn cũng biết việc sử dụng biến trong một hàm là quá bình thường. Nhưng chuyện gì xảy ra nếu bạn khai báo tên biến trong hàm trùng với tên của một biến nằm bên ngoài hàm?

Rất đơn giản, chỉ cần biết phạm vi hoạt động của biến là bạn sẽ hiểu được cách hoạt động của Python.

**Mục lục**

* [1. Biến toàn cục trong Python là gì?](https://freetuts.net/bien-toan-cuc-va-bien-cuc-bo-trong-python-1708.html#goto-h2-0)
* [2. Biến cục bộ trong Python là gì?](https://freetuts.net/bien-toan-cuc-va-bien-cuc-bo-trong-python-1708.html#goto-h2-1)
* [3. Từ khóa global trong Python](https://freetuts.net/bien-toan-cuc-va-bien-cuc-bo-trong-python-1708.html#goto-h2-2)
* [4. Lời kết](https://freetuts.net/bien-toan-cuc-va-bien-cuc-bo-trong-python-1708.html#goto-h2-3)

### Biến toàn cục trong Python là gì?

Trong Python, một biến được khai báo **bên ngoài hàm** hoặc trong **phạm vi toàn cục** được gọi là biến toàn cục. Điều này có nghĩa, biến toàn cục có thể được truy cập bên trong hoặc bên ngoài hàm.

Biến toàn cục có tên gọi là **Global Variable**.

Hãy xem một ví dụ về cách tạo một biến toàn cục trong Python.

**Global Variable**

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | x = "Học Python tại Freetuts.net"    def foo():      print("Trong hàm:", x)  foo()  print("Ngoài hàm:", x) |

Khi chạy chương trình này thì kết quả sẽ như sau:

|  |  |
| --- | --- |
| 1  2 | Trong hàm : Học Python tại Freetuts.net  Ngoài hàm: Học Python tại Freetuts.net |

Trong đoạn code trên thì biến x là **biến toàn cục** bởi nó được khai báo bên ngoài hàm, nằm tại vị trí cấp cao nhất của ứng dụng. Giống như Javascript, trong hàm có thể sử dụng biến cục bộ mà không cần phải thông qua việc truyền tham số.

Khi sử dụng biến toàn cục bên trong một hàm thì bạn không được thay đổi giá trị cho biến. Như ví dụ dưới đây là sai, bởi vì mình đã thay đổi giá trị cho biến x.

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | x = 10    def foo():      x = x \* 2      print(x)  foo() |

Khi chạy chương trình thì sẽ bị lỗi như sau:

|  |  |
| --- | --- |
| 1 | UnboundLocalError: local variable 'x' referenced before assignment |

**Lý do là đây**: Vì bên trong hàm foo() bạn đã thay đổi giá trị cho biến x nên mặc nhiên nó sẽ hiểu biến x là biệc cục bộ, mà biến cục bộ thì đoạn code x = x \* 2 sẽ sai vì biến x chưa được khai báo.

Để giải quyết vấn đề này thì bạn sử dụng từ khóa **global** nhé.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | x = 10    def foo():      global x      x = x \* 2      print(x)    # In giá trị 20  foo()    # In giá trị 20 vì x = 20  print(x) |

Như bạn thấy, biến cục bộ x đã bị thay đổi bên trong hàm **foo**.

### Biến cục bộ trong Python là gì?

Trong Python, biến cục bộ được hiểu là biến được khai báo bên trong hàm hoặc phạm vi cục bộ, những biến này gọi là biến cục bộ.

Biến cục bộ có tên gọi là **Local Variable**.

Hãy xem ví dụ dưới đây:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | def foo():      y = "local"    foo()    # Lệnh này sai vì biến y không tồn tại  print(y) |

Do biến y là biến cục bộ nằm bên trong hàm foo() nên đoạn code print(y) là sai, bạn không thể sử dụng biến cục bộ ở bên ngoài hàm được.

Chạy lên kết quả sẽ như sau:

|  |  |
| --- | --- |
| 1 | NameError: name 'y' is not defined |

### Từ khóa global trong Python

Mình đã lấy một ví dụ về từ khóa Global ở phần 1 rồi, tuy nhiên mình sẽ giải thích nó kỹ hơn một chút qua phần này nhé.

Để tường minh trong việc sử dụng biến cục bộ trong hàm thì Python cung cấp từ khóa **global**, khi khai báo từ khóa này cho một biến bất kì thì trình biên dịch sẽ tự hiểu biến đó sẽ trỏ tới một biến cục bộ nằm bên ngoài hàm.

**Ví dụ**: Sử dụng từ khóa global

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | x = "global"    def foo():      global x      y = "local"      x = x \* 2      print(x)      print(y)    foo() |

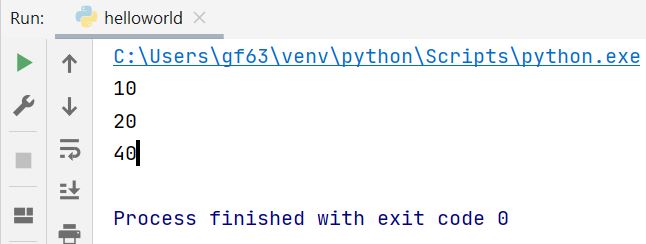
Chạy lên kết quả sẽ như sau:

|  |  |
| --- | --- |
| 1  2 | global global  local |

Khi chúng ta thay đổi giá trị cho biến được khai báo global nằm ở trong hàm thì giá trị của biến đó sẽ thay đổi theo. Xem ví dụ dưới đây để hiểu hơn.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | x = 5    def plusNum():      global x      x = x \* 2    plusNum()  print(x) # kết quả là 10    plusNum()  print(x) # kết quả là 20    plusNum()  print(x) # kết quả là 40 |

Xem hình:



### Lời kết

Khó khăn nhất là việc khai báo trùng tên ở biến toàn cục và biến cục bộ, vì vậy lời khuyên bạn nên sử dụng biến một cách khoa học, khai báo bằng tham số truyền vào nếu muốn sử dụng biến toàn cục ở bên trong hàm.

Như vậy là mình đã giới thiệu xong khái niệm về biến cục bộ và biến toàn cục trong Python, hy vọng bạn sẽ hiểu ý nghĩa của từng loại biến để sử dụng cho đúng mục đích.

## Bài tập Python: Thực hành tạo hàm trong Python

Trong bài này mình sẽ tổng hợp một số bài tập thực hành về hàm trong Python. Qua những bài tập tạo hàm này sẽ giúp bạn vững hơn trong việc phân chia module khi lập trình Python.

Nếu bạn đã từng học qua ngôn ngữ khác rồi thì có thể bỏ qua bài này. Nhưng nếu bạn là người mới tìm hiểu Python thì nên làm những bài tập này, nó sẽ giúp ích cho bạn rất nhiều trong việc rèn luyện tư duy logic.

Sau này khi làm việc với lập trình hướng đối tượng (OOP) thì bạn phải hiểu về hàm trước, bởi vì trong OOP có khái niệm về phương thức (mothod) có cách sử dụng rất giống với hàm.

**Câu hỏi thường gặp liên quan:**

* [Viết hàm Python tính điểm trung bình học sinh](https://freetuts.net/ref/viet-ham-python-tinh-diem-trung-binh-hoc-sinh-412.html)
* [Viết hàm Python tính tổng các số nguyên tô từ 0 - 1000](https://freetuts.net/ref/viet-ham-python-tinh-tong-cac-so-nguyen-to-413.html)
* [Viết hàm Python tìm giá trị lớn nhất trong 3 số](https://freetuts.net/ref/chua-khoi-tao-slug-414.html)
* [Viết hàm Lambda Python in ra thông tin sinh viên](https://freetuts.net/ref/viet-ham-lambda-python-in-ra-thong-tin-sinh-vien-415.html)
* [Viết hàm đệ quy Python tính tổng S = 1 + 2 + 3 + 4 + 5 ... + n](https://freetuts.net/ref/viet-ham-de-quy-python-tinh-tong-s-1-+-2-+-3-+-4-+-5-+-n-416.html)