Bài đọc thêm: Hàm trong Python

Nội dung

Cú pháp

Khái niệm của hàm là một điều quan trọng nhất trong toán học. Cách sử dụng phổ biến của hàm trong ngôn ngữ máy tính là để cài đặt hàm toán học. Hàm là tính toán một hay nhiều kết quả, được xác định hoàn toàn bởi tham số được truyền đến nó

Thông thường, hàm là phần tử cấu trúc trong ngôn ngữ lập trình để nhóm một tập hợp các câu lệnh nên chúng có thể được sử dụng nhiều lần trong chương trình. Cách duy nhất để hoàn thành việc này mà không có hàm sẽ là tái sử dụng mã bằng cách sao chép nó và chỉnh sửa nó theo ngữ cảnh khác của nó. Dùng hàm thường tăng độ dễ hiểu và chất lượng của chương trình. Việc này cũng giảm chi phí phát triển và bảo vệ phần mềm.

Hàm được biết tới dưới một vài cái tên trong ngôn ngữ lập trình, ví dụ : subroutines, routines, produre, methods, or subprograms.

Hàm trong Python được định nghĩa từ khóa def. Cú pháp tổng quát như sau :

```
def function-name(Parameter list):

       statements, i.e. the function body
```

Danh sách tham số chứa không hoặc nhiều tham số. Tham số được gọi là arguments. Thân hàm chứa dấu thụt lề. Thân hàm được thực hiện mỗi khi hàm được nhắc tới.

Tham số có thể bắt buộc hoặc không . Tham số tùy ý (0 hoặc nhiều hơn) phải tuân theo tham số bắt buộc.

Thân hàm có thể chứa một hoặc nhiều từ khóa return. Chúng có thể đặt ở bất cứ đâu trong thân hàm. Từ khóa return kết thúc thực thi hàm gọi và đưa ra đáp án, ví dụ, giá trị biểu thức sau từ khoá return, đến người gọi. Nếu câu lệnh return không có biểu thức, giá trị đặc biệt None sẽ được trả lại. Nếu không có từ khóa return trong hàm, hàm sẽ kết thúc, khi dòng điều khiển đạt đến cuối của thân hàm và giá trị "None" sẽ được trả về.

Ví dụ:

```
def fahrenheit(T_in_celsius):

    """ returns the temperature in degrees Fahrenheit """

  return (T_in_celsius * 9 / 5) + 32


for t in (22.6, 25.8, 27.3, 29.8):

    print(t, ": ", fahrenheit(t))
```

Đầu ra đoạn mã trông như sau:

```
22.6 :  72.68
25.8 :  78.44
27.3 :  81.14
29.8 :  85.64
```

Tham số tùy chọn

Các hàm có thể có các tham số tùy chọn, còn được gọi là các tham số mặc định. Tham số mặc định là các tham số, không cần phải cung cấp, nếu hàm được gọi. Trong trường hợp này, các giá trị mặc định được sử dụng. Chúng ta sẽ diễn tả các nguyên tắc hoạt động của các tham số mặc định với một ví dụ. Đoạn mã nhỏ sau, không hữu ích lắm, chào đón một người. Nếu không có tên được đưa ra, nó sẽ chào mọi người:

```
def Hello(name="everybody"):
    """ Greets a person """
    print("Hello " + name + "!")
Hello("Peter")
Hello()
```

Đầu ra sẽ như sau:

```
Hello Peter!
Hello everybody!
```

Docstring

Câu lệnh đầu tiên trong thân hàm thường là một chuỗi, có thể được truy cập với function_name .__ doc__

Câu lệnh này được gọi là Docstring.

Ví dụ:

```
def Hello(name="everybody"):
    """ Greets a person """
    print("Hello " + name + "!")
print("The docstring of the function Hello: " + Hello.__doc__)
```

Đầu ra:

```
The docstring of the function Hello:  Greets a person
```

Keyword Parameters

Sử dụng các tham số từ khoá là một cách thay thế để thực hiện việc gọi hàm. Định nghĩa của hàm không thay đổi.

Một ví dụ:

```
def sumsub(a, b, c=0, d=0):
    return a - b + c - d


print(sumsub(12,4))
print(sumsub(42,15,d=10))
```

Tham số từ khoá (Keyword parameters) chỉ có thể là những tham số không được sử dụng làm đối số vị trí (positional arguments). Chúng ta có thể thấy lợi ích của việc sử dụng trong ví dụ. Nếu chúng ta không có các tham số từ khoá, lời gọi thứ hai tới hàm sẽ cần tất cả bốn đối số, mặc dù c chỉ cần giá trị mặc định:

```
print(sumsub(42,15,0,10))
```

Giá trị trả về

Trong các ví dụ trước của chúng ta, chúng ta sử dụng từ khóa return trong hàm sumsub nhưng không có trong Hello. Vì vậy, chúng ta có thể thấy rằng từ khóa return không phải là bắt buộc. Nhưng những gì sẽ được trả về, nếu chúng ta không đưa ra từ khóa return. Hãy xem:

```
def no_return(x,y):
    c = x + y


res = no_return(4,5)
print(res)
```

Nếu chúng ta bắt đầu tập lệnh nhỏ này, None sẽ được in ra, nghĩa là giá trị đặc biệt None sẽ được trả về bởi một hàm không có từ khóa return. None cũng sẽ được trả lại, nếu chúng ta chỉ có một từ khóa return trong một hàm mà không có một biểu thức:

```
def empty_return(x,y):

    c = x + y

    return


res = empty_return(4,5)

print(res)
```

Nếu không, giá trị của biểu thức sau return sẽ được trả về. Trong ví dụ tiếp theo 9 sẽ được in:

```
def return_sum(x,y):

    c = x + y

    return c


res = return_sum(4,5)

print(res)
```

Trả về nhiều giá trị

Một hàm có thể trả về chính xác một giá trị, hoặc tốt hơn nên nói là một đối tượng. Một đối tượng có thể là một giá trị số, như một số nguyên hoặc một float. Nhưng nó cũng có thể là ví dụ một danh sách hoặc từ điển. Vì vậy, nếu chúng ta phải trả về ví dụ 3 giá trị số nguyên, chúng ta có thể trả về một danh sách hoặc một bộ với ba giá trị nguyên. Điều này có nghĩa là chúng ta có thể gián tiếp trả lại nhiều giá trị. Ví dụ sau, được tính ranh giới Fibonacci cho một số dương, trả về một 2-tuple. Phần tử đầu tiên là số Fibonacci lớn nhất nhỏ hơn x và thành phần thứ hai là số Fibonacci nhỏ nhất lớn hơn x. Giá trị trả về ngay lập tức được lưu trữ bằng cách giải nén vào các biến i và sup:

```
def fib_intervall(x):
    """ returns the largest fibonacci
    number smaller than x and the lowest
    fibonacci number higher than x"""
    if x < 0:
        return -1
    (old,new, lub) = (0,1,0)
    while True:
        if new < x:
            lub = new
            (old,new) = (new,old+new)
        else:
            return (lub, new)


while True:
    x = int(input("Your number: "))
    if x <= 0:
        break
    (lub, sup) = fib_intervall(x)
    print("Largest Fibonacci Number smaller than x: " + str(lub))
    print("Smallest Fibonacci Number larger than x: " + str(sup))
```

Biến cục bộ và toàn cục trong hàm

Tên biến thay đổi theo mặc định cục bộ đối với hàm, mà chúng được định nghĩa bên trong.

```
def f():
    print(s)
s = "Python"
f()
```

Output:

```
Python
```

```
def f():
    s = "Perl"
    print(s)

s = "Python"
f()
print(s)
```

Output:

```
Perl
Python
```

```
def f():
    print(s)
    s = "Perl"
    print(s)

s = "Python"
f()
print(s)
```

Nếu chúng ta thực thi đoạn mã trước, chúng ta sẽ nhận được thông báo lỗi:

UnboundLocalError: biến địa phương 's' được tham chiếu trước khi được gán

Các biến s là mơ hồ trong f (), tức là trong lần in đầu tiên trong f (), global s có thể được sử dụng với giá trị "Python". Sau đó chúng ta định nghĩa một biến cục bộ s với việc gán s = "Perl"

```
def f():
    global s
    print(s)
    s = "dog"
    print(s)
s = "cat"
f()
print(s)
```

Chúng ta đã làm cho biến s trở nên toàn cục trong đoạn mã trên. Vì vậy bất cứ điều gì chúng ta thực hiện với s để bên trong thân hàm f được thực hiện cho các biến toàn cục bên ngoài f.

Output:

```
cat
dog
dog
```

Số tham số tuỳ ý

Có rất nhiều tình huống trong lập trình, trong đó không thể xác định chính xác số lượng các tham số cần thiết. Một số tham số tùy ý có thể được hoàn thành trong Python với được gọi là tuple references. Một dấu hoa thị "*" được sử dụng ở phía trước của tên thông số cuối cùng để biểu thị nó như là một tuple reference. Dấu hoa thị này không được nhầm lẫn với cú pháp C, nơi ký hiệu này được kết nối với các con trỏ.

Ví dụ:

```
def arithmetic_mean(first, *values):
    """ This function calculates the arithmetic mean of a non-empty
        arbitrary number of numerical values """

    return (first + sum(values)) / (1 + len(values))


print(arithmetic_mean(45,32,89,78))
print(arithmetic_mean(8989.8,78787.78,3453,78778.73))
print(arithmetic_mean(45,32))
print(arithmetic_mean(45))
```

Results:

```
61.0
42502.3275
38.5
45.0
```

Điều này thật tuyệt, nhưng chúng ta vẫn còn một vấn đề. Bạn có thể có một danh sách các giá trị số. Ví dụ như:

```
x = [3, 5, 9]
```

Bạn không thể gọi nó bằng

```
arithmetic_mean(x)
```

bởi vì "arithmetic_mean" không thể xử lý với một danh sách. Gọi nó bằng

```
arithmetic_mean(x[0], x[1], x[2])
```

là rườm rà và trên hết là không thể trong một chương trình, bởi vì danh sách có thể có độ dài tùy ý.

Giải pháp thật dễ dàng. Chúng ta thêm một ngôi sao ở phía trước của x, khi chúng ta gọi hàm.

```
arithmetic_mean(*x)
```

Thao tác này sẽ giải phóng danh sách.

Một ví dụ thực tiễn:

Chúng ta có một danh sách

```
my_list = [('a', 232),
           ('b', 343),
           ('c', 543),
           ('d', 23)]
```

Chúng ta muốn chuyển danh sách này vào danh sách sau:

```
[('a', 'b', 'c', 'd'),
 (232, 343, 543, 23)]
```

Điều này có thể được thực hiện bằng cách sử dụng toán tử * và hàm zip theo cách sau:

```
list(zip(*my_list))
```

Số Tham số Từ khoá tùy ý

Trong chương trước chúng ta đã chứng minh làm thế nào để truyền một số tùy ý của các tham số vị trí vào một hàm. Cũng có thể truyền một số lượng tùy ý các tham số từ khoá vào một hàm. Để thực hiện mục đích này, chúng ta phải sử dụng dấu hoa thị đôi "**"

```
>>> def f(**kwargs):
...     print(kwargs)
...
>>> f()
{}
>>> f(de="German",en="English",fr="French")
{'fr': 'French', 'de': 'German', 'en': 'English'}
>>>
```

Một trường hợp sử dụng như sau:

```
>>> def f(a,b,x,y):
...     print(a,b,x,y)
…
>>> d = {'a':'append', 'b':'block','x':'extract','y':'yes'}
>>> f(**d)
('append', 'block', 'extract', 'yes')
```