

Bài 22: NHẬP XUẤT TRONG PYTHON - HÀM NHẬP

Xem bài học trên website để ủng hộ Kteam: [Nhập xuất trong Python – Hàm nhập](#)

Mọi vấn đề về lỗi website làm ảnh hưởng đến bạn hoặc thắc mắc, mong muốn khóa học mới, nhằm hỗ trợ cải thiện Website. Các bạn vui lòng phản hồi đến Fanpage [How Kteam](#) nhé!

Dẫn nhập

Trong bài trước, Kteam đã giới thiệu đến bạn [HÀM PRINT](#) – một hàm giúp bạn xuất kết quả ra màn hình (Shell)

Ở bài này Kteam sẽ tiếp tục giới thiệu với các bạn việc **Nhập xuất trong Python**. Cụ thể là việc nhập!

Nội dung

Để đọc hiểu bài này tốt nhất bạn cần:

- Cài đặt sẵn [MÔI TRƯỜNG PHÁT TRIỂN CỦA PYTHON](#).
- Xem qua bài [CÁCH CHAY CHƯƠNG TRÌNH PYTHON](#).
- Nắm [CÁCH GHI CHÚ](#) và [BIẾN TRONG PYTHON](#).
- [KIỂU DỮ LIỆU LIST](#), [KIỂU DỮ LIỆU TUPLE](#), [KIỂU DỮ LIỆU SET](#), [KIỂU DỮ LIỆU DICT](#) trong Python.
- Biết cách [XỬ LÝ FILE TRONG PYTHON](#)
- Biết [CÁCH SỬ DỤNG HÀM PRINT TRONG PYTHON](#)

Bạn và Kteam sẽ cùng tìm hiểu những nội dung sau đây

- Vì sao cần hàm input?
- Tìm hiểu cách sử dụng hàm input
- Hàm input Python 3.X và raw_input Python 2.X
- Lưu ý giành cho hàm input Python 2.X

Vì sao cần hàm input

Tèo là một Kter “bờ rào” của Kteam. Hôm trước, Tèo có làm một chương trình đơn giản. Đó chính là in ra dòng chữ “Xin chào Tiên”. Và đây là code của Tèo.

```
name = 'Tien'  
print('Xin chào', name)
```

Mọi chuyện diễn ra tốt đẹp, chương trình chạy đúng theo ý Tèo và cũng nhờ đó Tèo đã tạo được cảm tình với Tiên.

Thu thấy thế, cũng muốn Tèo viết cho một chương trình như Tiên và phải in ra dòng chữ “Xin chào Thu”.

Tèo lại mở code lên mà sửa lại:

```
name = 'Thu'  
print('Xin chào', name)
```

Sau đó, một số bạn nữ khác cũng muốn Tèo viết cho mình một chương trình như Thu và Tiên bao gồm Quỳnh, Nhi, Giao, Như, Uyên, Hương, Loan, Trung, Nam,... Kể không xuể. Và bạn thấy vấn đề đã nảy sinh. Tèo phải sửa code hết lần này đến lần khác.

Có thể việc này không mất quá nhiều thời gian, vì Tèo vẫn có thể viết cho mỗi bạn một cái chương trình riêng hoặc là mỗi lần viết là Tèo viết cho một bạn và thay đổi mã nguồn.

Nhưng nếu dung lượng máy của Tèo có hạn, không thể chứa nhiều chương trình hoặc Tèo không có đủ thời gian để chỉnh sửa code hết lần này tới lần khác thì sao? Tèo muốn viết một mà lại có thể cho nhiều người.

Điều này đưa ra cho Tèo một yêu cầu, đó chính là biến **name** phải là một biến có dữ liệu được nhập mỗi khi chạy chương trình thay vì được đưa sẵn cho một giá trị.

Và nhờ một hàm có tên là **input**. Tèo đã giải quyết được vấn đề nan giải sau ba ngày ba đêm tìm kiếm trên **GOOGLE**.

Tìm hiểu cách sử dụng hàm input

Theo như Tèo tìm kiếm trong tài liệu trên trang chủ của Python, hàm input có cú pháp như sau

```
input(prompt=None)
```

Lưu ý: Có lúc bạn sẽ nhìn thấy cú pháp của nó là **input(prompt=None, /)**. Cái phần thêm vào là kí tự **/** chỉ là một kí tự cho biết parameter **prompt** chỉ nhận giá trị dưới dạng **positional argument**. Nghĩa là khi bạn truyền vào cho hàm, bạn không được phép điền thêm chữ prompt.

```
>>> input('string') # hợp lệ
>>> input(prompt='string') # không hợp lệ
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: input() takes no keyword arguments
```

Parameter prompt là một parameter tùy chọn. Bạn có thể nhập hoặc không vì nó đã có giá trị mặc định là **None**.

Công dụng: Hàm này giúp chúng ta đọc một chuỗi từ **standard input** (hiểu nôm na là việc bạn nhập dữ liệu lên trên **Shell**) sau đó trả về cho chúng ta. Và vì nó là đọc một chuỗi, nên dù bạn có nhập cái gì đi chẳng nữa thì nó vẫn là một chuỗi dù là số, list, tuple, set, dictionary,...

Việc nhập sẽ kết thúc sau khi bạn nhấn phím **enter**. Ở đây, khi bạn nhấn phím **enter** (phím return) thì cũng đồng nghĩa với việc bạn gửi vào một kí tự **newline**. Nhưng kí tự **newline** này sẽ bị bỏ đi.

Nếu trong lúc nhập bạn nhấn **EOF**

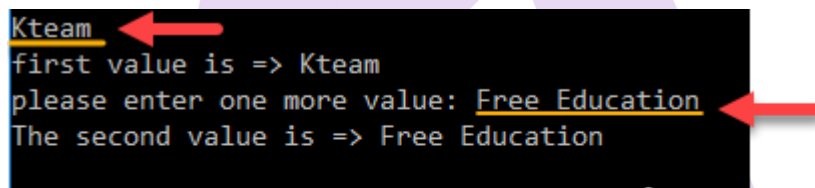
*nix: Ctrl + D, Windows: Ctrl + Z > Return (Enter) hoặc Ctrl + C

thì sẽ sinh lỗi **EOFError**.

Nếu **prompt** khác **None**, có nghĩa là bạn gửi cho prompt một giá trị. Thì giá trị này sẽ được in ra mà không có kí tự **newline** đi kèm trước khi đọc giá trị nhập vào.

Chúng ta đến với ví dụ. Hãy tạo một file có nội dung như sau

```
value = input() # prompt để None
print('first value is =>', value)
next_value = input('please enter one more value: ')
print('The second value is =>', next_value)
```



```
Kteam
first value is => Kteam
please enter one more value: Free Education
The second value is => Free Education
```

Đây là hình ảnh khi chạy chương trình trên. Trong đó:

- Những dòng có **mũi tên màu đỏ** là những dòng thực hiện hàm **input**.
- Những chữ **gạch chân màu vàng** chính là giá trị nhập vào.

Đầu tiên, ta sẽ được yêu cầu nhập dữ liệu vào cho biến **value**. Ở đây, Kteam nhập vào giá trị là **Kteam**.

Và điều đó được kiểm chứng bằng việc ở dòng tiếp theo, giá trị Kteam được in ra màn hình.

Tiếp đến, chúng ta tiếp tục được yêu cầu nhập dữ liệu. Bạn có thể thấy khác so với lần chúng ta sử dụng hàm `input` khi không truyền giá trị vào cho parameter `prompt`. Giờ đây, chúng ta có một dòng ghi chú yêu cầu nhập dữ liệu. Và với giá trị nhập vào là **Free Education**, giá trị đó đã được in ra ở dòng cuối cùng.

Kteam xin được lưu ý thêm một lần nữa đó là bạn nhập cái gì thì giá trị trả về **LUÔN LUÔN LÀ CHUỖI**.

Hãy thử đoạn code sau:

```
# reading input
int_num = input('Enter an integer: ')
float_num = input('Enter a float: ')
lst = input('Enter a list: ')
tup = input('Enter a tuple: ')
set_ = input('Enter a set: ')
dict_ = input('Enter a dict: ')

# print out output
print('Type of int_num', type(int_num))
print('Type of float_num', type(float_num))
print('Type of lst', type(lst))
print('Type of tup', type(tup))
print('Type of set_', type(set_))
print('Type of dict_', type(dict_))
```

```
Enter an integer: 69
Enter a float: 3.14
Enter a list: [1, 2, 'Kteam']
Enter a tuple: ('Kteam', 'Free Education', [1, 2, 101.00])
Enter a set: {(1, 2), 'abc'}
Enter a dict: {'team': 'Kteam', (90, 4): ['list', 'tuple']}
Type of int_num <class 'str'>
Type of float_num <class 'str'>
Type of lst <class 'str'>
Type of tup <class 'str'>
Type of set_ <class 'str'>
Type of dict_ <class 'str'>
```

Như bạn thấy, tất cả đều thuộc lớp chuỗi. Kteam sẽ tiếp tục thêm một số ví dụ với hàm `input`.

```
value = input('Enter something => ')\nprint('You just entered', value)\nprint('__repr__ method: %r' %value)
```

Lần này, Kteam sẽ chỉ nhấn phím **Enter**.

```
Enter something =>\nYou just entered\n__repr__ method: ''
```

Khi bạn không nhập thứ gì và nhấn phím **Enter**. Chuỗi bạn nhận được từ hàm `input` là một chuỗi rỗng (số kí tự trong chuỗi bằng 0).

Tiếp tục với đoạn code trên, lần này Kteam sẽ nhấn **EOF**.

```
Enter something => ^Z\nTraceback (most recent call last):\n  File "a.py", line 1, in <module>\n    value = input('Enter something => ')\nEOFError
```

- Lỗi **EOFError** hiện lên. Chương trình kết thúc ngay lập tức.

Hàm input Python 3.X và raw_input Python 2.X

Hàm **raw_input** **không** tồn tại trong Python 3.X, nó đã được đổi tên thành **input** ở phiên bản Python 3.X.

Lưu ý: giành cho hàm input Python 2.X

Trong **Python 2.X**, còn một hàm nữa cũng gần giống với hàm **raw_input** (chính là hàm **input** ở **Python 3.X**) là hàm **input**.

Cú pháp của hàm này hoàn toàn tương tự với hàm **input** trong **Python 3.X**. Nó cũng sẽ nhận vào một chuỗi như hàm **input Python 3.X** (**raw_input Python 2.X**). Tuy nhiên, chuỗi đó sẽ được truyền vào hàm **eval**.

Do đó input Python 2.X có cú pháp

```
input(prompt=None)
```

Sẽ tương tự

```
eval(raw_input(prompt=None))
```

Và tương đương ở Python 3.X sẽ

```
eval(input(prompt=None))
```

Hàm `eval` có khả năng thực thi một `expression` với `expression` đưa vào dưới dạng chuỗi.

Một `expression` là một giá trị nào đó như một con số, một chuỗi, một list.

Sau đây là một vài ví dụ về hàm `eval`:

```
>>> eval('123')
123
>>> eval('[1, 2, 3]')
[1, 2, 3]
>>> x = 1
>>> eval('x + 2')
3
>>> eval('print("This is exec by eval fucntion")') # hàm print là một expression với
giá trị là None
This is exec by eval fucntion
>>> eval('a = 3') # đây là một statement. Không phải expression.
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<string>", line 1
    a = 3
    ^
SyntaxError: invalid syntax
```

Lưu ý: Ở đây, Kteam có một lưu ý với các bạn đó là **không nên** sử dụng hàm `eval` trừ khi thực sự rất cần thiết.

Có một số lí do để bạn nên tránh sử dụng hàm `eval`:

- Khiến việc debug khó khăn
- Làm chậm chương trình
- Luôn có cách tốt hơn thay thế
- Rất nguy hiểm và không an toàn.

Nếu bạn thắc mắc tại sao lại nguy hiểm. Thì Kteam có thể đưa ra một số ví dụ đơn giản.

Ví dụ: bạn cho phép người dùng sử dụng chương trình của bạn. Bạn yêu cầu họ nhập một số thứ nhưng lại sử dụng hàm eval bọc lên hàm input. Thế nên, họ có thể sử dụng nó để phá chương trình của bạn.

Giả sử bạn có một ứng dụng web. Nếu một kẻ xấu nào đó nhập vào với nội dung dạng thế này thì coi như ứng dụng của bạn toi.

```
Enter something: __import__('shutil').rmtree('/root')
```

Câu lệnh dưới, có thể xóa sạch cây thư mục của bạn. Đó là một dạng của **command injection**. Điều này rất nguy hiểm cho hệ thống của bạn.

```
>>> __import__('shutil').rmtree('/root')
```

Do đó, việc sử dụng **eval** phải được cân nhắc. Đương nhiên sẽ có trường hợp eval không nguy hiểm như trên, hoặc là bạn phải dùng tới nó. Nhưng hãy hạn chế!

Kết luận

Qua bài viết này, Bạn đã biết về việc yêu cầu người dùng NHẬP NỘI DUNG từ bàn phím trong Python.

Ở bài viết sau. Kteam sẽ nói về [KIỂU DỮ LIỆU BOOLEAN TRONG PYTHON](#).

Cảm ơn bạn đã theo dõi bài viết. Hãy để lại bình luận hoặc góp ý của mình để phát triển bài viết tốt hơn. Đừng quên **"Luyện tập – Thử thách – Không ngại khó"**.