# Bài 21: NHẬP XUẤT TRONG PYTHON - HÀM XUẤT

*Xem bài học trên website để ủng hộ Kteam:* Nhập xuất trong Python – Hàm xuất

*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 ITERATION & MỘT SỐ HÀM HỖ TRỢ CHO ITERABLE OBJECT trong Python

Ở bài này Kteam sẽ giới thiệu với các bạn việc **Nhập xuất trong Python**. Một điều rất cần thiết!

---

## Nội dung

Để đọc hiểu bài này tốt nhất bạn cần:

- Cài đặt sẵn MÔI TRƯỜNG PHÁT TRIỂN CỦA PYTHON.
- Xem qua bài CÁCH CHẠY 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 LÍ FILE TRONG PYTHON

Trong bài này, bạn và Kteam sẽ cùng tìm hiểu những nội dung sau đây

- Vì sao cần hàm print?
- Tìm hiểu cách sử dụng hàm print thông qua các parameter.
- Print Python 3.X và Python 2.X có gì khác nhau?

---

# Vì sao cần hàm print

Nếu bạn hay dùng **interactive prompt** thì bạn nhân ra rằng, kết quả luôn xuất hiện sau mỗi dòng code của bạn. Tuy nhiên, nó sẽ không như vậy khi bạn viết những dòng code vào trong một file Python và chạy chương trình đó.

Bạn cần một hàm giúp bạn xuất các nội dung mà bạn muốn cụ thể ở đây là xuất ra Shell (terminal, command prompt, powershell,...). Đó là lí do hàm print ra đời!

---

# Tìm hiểu cách sử dụng hàm print thông qua các parameter

Hàm print có cú pháp như sau

## Cú pháp:

```
print(*objects, sep=' ', end='\n', file=sys.stdout, flush=False)
```

Chúng ta sẽ tìm hiểu parameter đầu tiên

---

# *objects

* chính là packing argument. Ở đây hiểu nôm na sẽ là nó sẽ gom lại các argument của bạn lại thành một Tuple.

```
>>> packing = 1, 2, 3, 4 # giống như gọi hàm function(1, 2, 3, 4)
>>> packing
(1, 2, 3, 4)
```

Khi bạn truyền các argument vào hàm (giá trị 1, giá trị 2, giá trị 3,...) thì nó sẽ gói lại thành một Tuple giống như trên.

```
>>> print('Kteam')
Kteam
>>> print('Kteam', 'Free Education')
Kteam Free Education
>>> print('Kteam', 'Free Education', 'one more argument')
Kteam Free Education one more argument
```

Nhờ như vậy, bạn có thể truyền argument vào hàm **print** với số lượng bất kì. Điều này giúp bạn không phải ép kiểu dữ liệu, để rồi nối chúng lại với nhau thành một giá trị rồi mới truyền cho hàm print.

```
>>> print('Kteam' + 69)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: must be str, not int
>>> print('Kteam' + str(69))
Kteam69
>>> print('Kteam', 69)
Kteam 69
>>> print(123, [1, 2, 3], 'Kteam')
123 [1, 2, 3] Kteam
```

Chắc bạn cũng nhận ra một chút khác biệt ở hai trường hợp bên dưới.

```
>>> print('Kteam' + 'Python')
```

```
KteamPython
>>> print('Kteam', 'Python')
Kteam Python
```

Để hiểu điều đó, chúng ta tới với parameter tiếp theo

---

## sep (separate – chia ra, phân ra)

Giá trị mặc định của parameter này là **một khoảng trắng**. Khi các argument bạn ném vào cho hàm print để hàm print in ra nội dung, như đã biết là nó sẽ được gói vào một Tuple. Các giá trị trong Tuple sẽ được nối với nhau bằng parameter sep.

**Lưu ý:** Khi truyền giá trị vào cho parameter theo cách keyword argument thì sẽ không bị **packing**. Nghĩa là sẽ không bị gói vào trong giá trị của parameter object.

```
>>> print('Kteam', 'Python', 'Course') # sep mặc định là 1 khoảng trắng
Kteam Python Course
>>> print('Kteam', 'Python', 'Course', sep='---')
Kteam---Python---Course
>>> print('Kteam', 'Python', 'Course', sep='|||')
Kteam|||Python|||Course
>>> print('Kteam', 'Python', 'Course', sep='\n')
Kteam
Python
Course
>>> print('Kteam', 'Python', 'Course', sep='')
KteamPythonCourse
```

Tiếp theo là một parameter khá rắc rối

---

# end (kết thúc bằng)

Đầu tiên, hãy chạy một file Python với nội dung sau đây.

```
print('line 1')
print('line 2')
print('line 3')
```

Kết quả bạn nhận được chắc chắn sẽ là

```
line 1
line 2
line 3
```

Nếu bạn từng học qua ngôn ngữ **lập trình C** hoặc **C++** hay là **Java** cũng có thể là **C#**. Bạn sẽ nhận thấy, mỗi lần print, chúng sẽ tự xuống dòng.

Đó là nhờ parameter end. Nó sẽ tự thêm một kí tự newline (**\n**) vào cuối để có thể đưa con trỏ xuống dòng mới thay vì bạn phải tự thêm \n như một số ngôn ngữ lập trình khác (một số ngôn ngữ lập trình có hỗ trợ thêm phương thức giúp xuất nội dung và tự động xuống dòng)

Và đương nhiên, chúng ta cũng có thể thay đổi giá trị của parameter này.

```
>>> print('a line without newline', end='')
a line without newline>>> print('a line without newline', end='|||')
a line without newline|||>>> print()


>>>
```

Bạn cũng thấy nếu không có end bằng một kí tự newline thì **interactive prompt** lộn xộn thế nào.

Nhưng đó không phải vấn đề. Hãy cẩn thận khi sử dụng print mà không có newline.

Hãy tạo một file Python có nội dung như sau:

```
from time import sleep # nhập hàm sleep từ thư viện time

print('start....')
sleep(3) # dừng chương trình 3 giây
print('end...')
```

Khi chạy chương trình, bạn sẽ thấy xuất hiện dòng **`start....`** sau đó 3 giây sau sẽ xuất hiện tới dòng **`end...`**.

Kết quả này hoàn toàn bình thường và đúng như những gì dự đoán. Nhưng hãy thử thay đổi một tí:

```
from time import sleep # nhập hàm sleep từ thư viện time

print('start....', end='') # in ra nội dung và kết thúc bởi một chuỗi  rỗng
sleep(3) # dừng chương trình 3 giây
print('end...')
```

Lần này đã có khác biệt. Bạn sẽ không thấy gì xuất hiện ban đầu, mãi đến 3 giây sau bạn mới thấy dòng **`start....end...`**. Kết quả thì đúng, nhưng cách kết quả được xuất ra thì không giống như bạn nghĩ.

Vì sao lại vậy? Đó là do mỗi lần hàm print nhận được các giá trị bạn muốn in. Các giá trị đó được gói trong một Tuple. Tiếp đến, hàm print nạp từng giá trị trong Tuple vào bộ nhớ đệm. Nếu giá trị đó là một chuỗi và có kí tự newline (ở vị trí bất kì) thì hàm print sẽ yêu cầu bộ nhớ đệm xuất những gì có trong bộ nhớ đệm từ nãy nạp đến giờ.

Hoặc khi kết thúc chương trình, những gì còn trong bộ đệm cũng sẽ được xuất ra.

---

## Một số ví dụ

**Ví dụ 1:** Hãy thử một vài ví dụ khác để hiểu thêm

```
from time import sleep # nhập hàm sleep từ thư viện time
```

```
print('line 1\n', 'line2', end='')
sleep(3) # dừng chương trình 3 giây
print('end...')
```

Kết quả xuất hiện sẽ là **`line1`** > đợi 3 giây > xuất hiện các nội dung còn lại. Vì chuỗi **'line 1\n'** có kí tự newline nên chuỗi đó được xuất ra. Còn chuỗi **'line 2'** thì không nên vẫn nằm trong bộ nhớ đệm.**Ví dụ 2:**

```
from time import sleep # nhập hàm sleep từ thư viện time

print('line 1', 'lin\ne2', end='')
sleep(3) # dừng chương trình 3 giây
print('end...')
```

Kết quả sẽ là xuất in hai chuỗi **`line 1`** và **`line 2`** > đợi 3 giây > xuất nội dung còn lại.

Quy trình sẽ là nạp chuỗi line 1 vào bộ nhớ đệm, nạp tiếp chuỗi line 2 vào bộ nhớ đệm, thấy chuỗi line 2 có kí  tự newline, xuất những gì có trong bộ nhớ đệm ra. Sau đó đợi 3 giây và rồi xuất nội dung còn lại.

---

## file

Mặc định hàm print sẽ ghi nội dung vào file sys.stdout. Cũng nhờ vậy, bạn mới thấy được nội dung trên shell. Đương nhiên, dựa vào đây, ta cũng có thể sử dụng hàm print như là phương thức write trong việc ghi file.

```
>>> with open('printtext.txt', 'w') as f:
...     print('printed by print function', file=f)
...
>>> with open('printtext.txt') as f:
...     f.read()
...
'printed by print function\n'
```

## flush

Parameter cuối cùng - **flush**. Giá trị mặc định giá trị là False. Liên quan khá nhiều đến parameter end lúc nãy thế nên ta hãy quay lại ví dụ lúc nãy.

```
from time import sleep # nhập hàm sleep từ thư viện time

print('start...', end='')
sleep(3) # dừng chương trình 3 giây
print('end...')
```

Sau 3 giây chương trình mới có kết quả. Bạn cũng đã biết vì sao rồi, đúng chứ?

Nào, hãy để cho parameter flush giá trị True

```
from time import sleep # nhập hàm sleep từ thư viện time

print('start...', end='', flush=True)
sleep(3) # dừng chương trình 3 giây
print('end...')
```

Kết quả bây giờ vẫn vậy, nhưng quá trình xuất kết quả có chút khác biệt. Bạn ngay lập tức nhìn thấy nội dung dòng print đầu tiên. Đó là nhờ parameter flush. Nếu là True, nó sẽ yêu cầu bộ đệm xuất những gì có trong bộ đệm ra.

# Print trong Python 3.X và Python 2.X có gì khác nhau?

Print trong Python 3.X là một hàm, như đã giới thiệu. Còn với Python 2.X nó là một câu lệnh.

```
# print trong Python 2.X
```

```
print 'Kteam'
print 'Kteam', 'Free Education'
# tương tự với trong Python 3.X sẽ là
print('Kteam')
print('Kteam', 'Free Education')
```

Một số bạn nhầm lần rằng Print Python 2.X cũng có thể sử dụng như Python 3.X

```
# print trong Python 2.X
print('Kteam')
# và nhận được kết quả giống như Python 3.X
print('Kteam')
```

Nhưng bản chất là khác nhau

```
# print trong Python 2.X
print('Kteam')
# tương đương với Python 3.X là
print(('Kteam'))
```

Đây là **interactive prompt** của Python 2.X. Ta sẽ thử một ví dụ để làm rõ điều này

```
>>> print('Kteam')
Kteam
>>> print('Kteam', 'Free Education')
('Kteam', 'Free Education')
```

Bạn cũng thấy, cặp dấu **()** không phải là  một cặp dấu ngoặc như cách gọi hàm. Đó giống như việc bạn đặt một giá trị trong cặp dấu ngoặc đơn mà  thôi. Và vì nó có một giá trị nên không có sự khác biệt

Còn khi bạn đặt hai giá trị trở lên, Python hiểu đó là một Tuple.

**Một đoạn code nhỏ dành cho bạn tự nhiên cứu:**

```
from time import sleep

your_name = "Henry"
your_great = "Hello! My name is "

for c in your_great + your_name:
    print(c, end='', flush=True)
    sleep(0.1)
print()
```

# Kết luận

Qua bài viết này, Bạn đã biết về việc xuất nội dung trong Python.

Ở bài viết sau. Kteam sẽ nói về NHẬP XUẤT TRONG PYTHON – HÀM NHẬP.

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**".