# Bài 28: KIỂU DỮ LIỆU FUNCTION TRONG PYTHON - SƠ LƯỢNG VỀ HÀM

*Xem bài học trên website để ủng hộ Kteam:* Kiểu dữ liệu Function trong Python - Sơ lượng về hàm

*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 VÒNG LẶP FOR TRONG PYTHON.

Và ở bài này Kteam sẽ lại tìm hiểu với các bạn **KIỂU DỮ LIỆU FUNCTION TRONG PYTHON - Sơ lược về hàm.**

---

## 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.
- CÁC KIỂU DỮ LIỆU ĐƯỢC GIỚI THIỆU TRONG PYTHON
- CÂU ĐIỀU KIỆN IF TRONG PYTHON
- VÒNG LẶP WHILE và VÒNG LẶP FOR TRONG PYTHON
- NHẬP XUẤT TRONG PYTHON

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

- Vấn đề.
- Khai báo hàm. (Create function)
- Gọi hàm. (Call function)
- Đừng viết lại code! (DRY – Don't Repeat Yourself)
- Parameter và Argument.
- Giá trị mặc định của parameter. (default argument)

---

# Vấn đề

Bạn đã được giới thiệu qua HÀM PRINT. Vậy bạn có biết người ta đã tạo nên hàm đó như thế nào không? Kteam nghĩ bạn cũng không nên tìm hiểu làm gì vì vốn nó cũng rất phức tạp!

Vậy nếu bây giờ không có hàm print thì có phải mỗi lần bạn muốn in ra thứ gì đó trên **Shell** thì bạn phải viết một dãy lệnh dài để có thể làm điều đó đúng chứ?

Ta thử tính đơn giản thôi!

**Ví dụ:** hàm print chỉ tốn 10 dòng để có thể in ra một chuỗi (thật sự là nhiều hơn vậy rất nhiều), vậy nếu bạn dùng 10 lần print thì nó đã tới 100 dòng.

Mà với một chương trình, liệu bạn chỉ có sử dụng mỗi hàm **print**? Nếu không nhờ những kĩ sư đã viết sẵn cho chúng ta rất nhiều hàm để cho chúng ta sử dụng liệu chúng ta tốn mất bao lâu và bao nhiêu dòng code cho một script in ra dòng chữ "Hello Kteam!" ở trên Shell?

Thời gần xưa, con người ta khi viết các dòng code thì sẽ viết từ trên viết xuống, lệnh nào làm trước thì viết trước và cứ thế hoàn thành đoạn script. Ta gọi, đó là **Lập trình tuyến tính** (linear programming).

Và khi nhiều vấn đề phát sinh từ **linear programming** như việc sửa đổi, cập nhật, rất khó khăn và nhiều nguyên nhân khác đã đưa ra một thời kì lập trình mới, đó chính là **Lập trình thủ tục** (procedural programming)

Để có thể có một chương trình theo hướng **procedural programming**, thì ta phải biết khái niệm hàm và cụ thể trong bài này, Kteam sẽ giới thiệu với các bạn về nó.

---

# Khai báo hàm (create function)

> Ở đây, Kteam sử dụng từ "**khai báo**", và với tựa bài có cụm từ "**kiểu dữ liệu**" để muốn nói với bạn rằng trong Python, những hàm mà ta tạo là những biến đặc biệt mà ta khai báo.

Bạn nên nắm rõ điều này để sau này có khi bạn sẽ tiếp cận tới khái niệm meta class (siêu lớp) sẽ hiểu rõ hơn.

Để khai báo một hàm, ta sử dụng từ khóa "**def**" với cú pháp như sau

## Cú pháp:

```
def <function_name>(parameter_1, parameter_2, .., parameter_n):
function-block
```

Trong cú pháp đó, bạn không được bỏ sót bất kì thứ nào ngoại trừ bạn có thể bỏ trống các parameter.

**Ví dụ:**

```
>>> def kteam():
...     pass
...
>>> kteam
<function kteam at 0x014EC5D0>
```

```
>>> type(kteam)
<class 'function'>
```

**Lưu ý:**

Lệnh pass ở trên là một lệnh "**giữ chỗ**" (placeholder statement) để giúp cho các block của Python không bị thiếu câu lệnh trong trường hợp bạn chưa biết viết gì cho phù hợp.

Bạn có thể thấy, khi in ra hàm kteam, bạn sẽ nhận được một dòng khá tương một một generator expression.

---

# Gọi hàm (call function)

Việc gọi hàm, ta có cú pháp sau đây

## Cú pháp:

```
<function>()
```

Khi gọi hàm, các câu lệnh có trong hàm sẽ được thực thi

**Ví dụ:**

```
>>> def kteam():
...     print('Hello Kteam!')
...
>>> kteam
<function kteam at 0x0323FD68>
>>> kteam()
Hello Kteam!
```

Ta gọi hàm kteam, vậy nên hàm kteam sẽ thực thi các lệnh mà nó có. Cụ thể ở đây là nó dùng hàm **print** in ra màn hình một dòng chuỗi.

---

# Đừng viết lại code (DRY - Don't Repeat Yourself)

Giả sử, bạn có một script với nhiệm vụ in ra 8 dòng in ra "Hello Kteam!" và "Free Education"

```
print('Hello Kteam!')

print("Free Education")

print('Hello Kteam!')

print("Free Education")

print('Hello Kteam!')

print("Free Education")

print('Hello Kteam!')

print("Free Education")
```

**Lưu ý**:

Việc sử dụng vòng lặp để làm chuyện này là khả thi, nhưng có nhiều trường hợp các câu lệnh không nằm liền kề nhau như thế này, bạn không thể dùng vòng lặp rút gọn.

Bây giờ, bạn muốn thay đổi dòng "Hello Kteam!" thành "Hi Kteam!", vậy là bạn phải chỉnh sửa lại 4 dòng lệnh.

Giờ ta đưa vấn đề xa hơn một tí nữa. Nếu nhiệm vụ của bạn không chỉ là **print** ra tám dòng chữ mà còn phải làm nhiều thứ khác, thì có phải bạn đang viết lại rất nhiều code không? Và khi chỉnh sửa mà nếu chỉnh sửa nhiều thì bạn sẽ phải mất rất nhiều công sức.

Để có thể tránh được việc đó, ta hãy sử dụng hàm

```
def kteam():
    print('Hello Kteam!')
```

```
    print("Free Education")
kteam()
kteam()
kteam()
kteam()
```

Và khi muốn chỉnh sửa, ta chỉ cần chỉnh sửa bên trong hàm, thì ta sẽ thay đổi được tất cả.

---

# Parameter và Argument

Đầu tiên, ta khởi tạo một hàm có các parameter

```
>>> def kteam(text):
...     print(text)
```

Và khi gọi hàm có parameter, bạn phải truyền vào argument tương ứng.

```
>>> kteam('Hello Kteam!')
Hello Kteam!
```

Ở đây, argument chúng ta đưa vào là một chuỗi. Chuỗi này sẽ được đưa vào gán cho parameter tương ứng là text. Và rồi hàm thực hiện việc in text ra.

Đương nhiên là chúng ta có thể biến hóa nhiều ra nữa

```
>>> def kteam(greating, name):
...     print(greating, name + '!')
...
>>> kteam('Hi', 'Kteam')
Hi Kteam!
>>> kteam('Hello', 'SpaceX')
Hello SpaceX!
```

---

# Giá trị mặc định của parameter (Default argument)

Hãy coi ví dụ sau:

```
>>> def kteam(greating, name):
...     print(greating, name + '!')
...
>>> kteam('Hi', 'Kteam')
Hi Kteam!
>>> kteam('Hello', 'SpaceX')
Hello SpaceX!
>>> kteam('Hi', 'Tesla')
Hi Tesla!
>>> kteam('Hi', 'Python')
Hi Python!
>>> kteam('Hi', 'Jack')
Hi Jack!
```

Ta thấy, tần suất xuất hiện chuỗi "Hi" cho parameter greating rất cao. Giờ ta cần một parameter giữ giá trị là chuỗi "Hi" nhưng vẫn cho ta thay đổi khi cần. Bấy giờ, ta nên sử dụng **default argument**.

```
>>> def kteam(name, greating='Hi'):
...     print(greating, name + '!')
...
>>> kteam('Kteam')
Hi Kteam!
>>> kteam('SpaceX')
Hi SpaceX!
>>> kteam('SpaceX', 'Hello')
Hello SpaceX!
```

**Lưu ý:**

Khi bạn đưa default argument cho các parameter, phải để các parameter có default argument ở sau cùng.

Default argument là một **unhashable container**

Như các bạn đã biết, **unhashable container** phổ biến mà ta đã từng biết như LIST, DICT, SET. Ở đây có một cảnh báo cho bạn việc bạn sử dụng default argument cho parameter là một **unhashable container** đó là giá trị của nó không được  làm mới (refresh) sau mỗi lần gọi hàm mà không pass argument mới cho parameter đó. Đương nhiên là nếu bạn có pass cho nó một argument mới thì **container** đó vẫn không hề mất giá trị nếu lần sau bạn gọi nó.

```
>>> def f(kteam=[]):
...     kteam.append('F')
...     print(kteam)
...
>>> f()
['F']
>>> f()
['F', 'F']
>>> f()
['F', 'F', 'F']
>>> f([1, 2, 3])
[1, 2, 3, 'F']
>>> f()
['F', 'F', 'F', 'F']
```

---

# Củng cố bài học

## Đáp án bài trước

Bạn có thể tìm thấy câu hỏi của phần này tại CÂU HỎI CỦNG CỐ trong bài VÒNG LẶP FOR TRONG PYTHON - Phần 2

1.

```
>>> for l in lst:
...     l[0] = None
```

2.

```
n = int(input('Enter size of matrix: '))
dx, dy = 1,0
x, y = 0,0
spiral_matrix = [[None] * n for j in range(n)]

for i in range(n ** 2):
    spiral_matrix[x][y] = i
    nx, ny = x + dx, y + dy
    if 0 <= nx < n and 0 <= ny < n and spiral_matrix[nx][ny] == None:
        x, y = nx, ny
    else:
        dx, dy = -dy, dx
        x, y = x + dx, y + dy

for y in range(n):
    for x in range(n):
        print("%02i" % spiral_matrix[x][y], end=' ')
    print()

print()
```

---

# Kết luận

Qua bài viết này, Bạn đã biết một chút về Kiểu dữ liệu Function trong Python.

Ở bài viết sau. Kteam sẽ tiếp tục giới thiệu thêm với các bạn về KIỂU DỮ LIỆU FUNCTION TRONG PYTHON – Phần 2.

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"**.