Bài 27: VÒNG LẶP FOR TRONG PYTHON – PHẦN 2

Xem bài học trên website để ủng hộ Kteam: Vòng lặp For trong Python – Phần 2

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 tiếp tục tìm hiểu với các bạn **Vòng lặp For trong Python**.

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

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

- Kiểu dữ liệu range (dãy số)
- Sự khác nhau giữa sequence scan và indexing scan
- Comprehension
- Giới thiệu hàm enumerate

Kiểu dữ liệu range (dãy số)

Bạn gặp kiểu dữ liệu này suốt các phần liên quan đến **comprehension** hoặc là liên quan đến **iterator object**.

Đây là một kiểu dữ liệu rất đặc biệt vì ta có thể lấy nhiều giá trị từ nó nhưng bản chất thì nó không lưu giữ những giá trị mà chúng ta lấy. Trước khi đến với điều thú vị này, chúng ta cùng ngó tổng quát về kiểu dữ liệu này.

Chúng ta có hai cách khởi tạo.

Cách khởi tạo thứ nhất

Cú pháp:

```
range(stop)
```

Với cách này, ta sẽ tạo một dãy số bắt đầu bằng số **0** và kết thúc là stop **– 1**. Dãy số này là một cấp số cộng với công sai là **1**.

```
>>> k = range(3)
>>> type(k)
<class 'range'>
>>> k[0] # range có hỗ trợ indexing
0
>>> k[1]
1
>>> k[-1]
```

```
2
>>> k[3]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: range object index out of range
>>> list(k)
[0, 1, 2]
>>> k[0] = 10 # range object là hasable object
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'range' object does not support item assignment
```

Cách khởi tạo thứ hai

Cú pháp:

```
range(start, stop[, step])
```

Với cú pháp này, ta sẽ tạo một dãy số bắt đầu bằng **start** và kết thúc là **stop – 1**. Dãy số này là một cấp số cộng với công sai là **1**.

Trong trường hợp step (buộc phải khác **0**) được đưa vào thì công sai sẽ là **step**.

```
>>> list(range(2, 5))
[2, 3, 4]
>>> list(range(4, 1, -1))
[4, 3, 2]
>>> list(range(2, -3, -1))
[2, 1, 0, -1, -2]
```

Và đây là điều thú vị của hàm **range**. Hãy tạo một List chứa một dãy số cộng từ **0** tới một số kha khá lớn. Đương nhiên là cũng sẽ có một Range có một dãy số tương tự.

```
>>> k = range(9999999) # nếu máy bạn có khỏe thì hãy cho số lớn hơn tí nữa để thấy rõ sự khác biệt
>>> lst = list(k)
```

Tiếp đến, hãy dùng toán tử **in**

```
>>> 9999999999 in k
False
>>> 9999999999 in lst
False
```

Nếu bạn chưa thấy gì thì hãy thử số nào to hơn chút. Còn nếu thấy rồi, thì đó chính là tốc độ. Chệnh nhau vài mili giây. Đối với máy tính hiện đại, một vài mili giây là đủ để làm rất nhiều thứ. Vậy điều gì làm nên khác biệt đó?

Range là một lớp được thiết kế riêng để lưu giữ những dãy số. Vậy nên nó đã được những kĩ sư Python sử dụng các thuật toán để có thể có được sự linh hoạt này.

Mỗi lần bạn lấy một giá trị trong một đối tượng thuộc hàm range thì đối tượng này sẽ lấy các giá trị của **start**, **stop**, **step** và một vài thứ khác để tính toán và sinh ra một con số.

Để hiểu rõ hơn bạn tham khảo câu hỏi này trên **Stack Overflow**

> Why is "1000000000000000 in range(1000000000000001)" so fast in Python 3?

Sử dụng range để duyệt một List, Tuple, Chuỗi

Chúng ta sử dụng một dãy số để dùng **indexing** lấy các giá trị trong một List, Tuple hoặc Chuỗi.

Chúng ta có hàm **range** sinh ra một dãy số.

Kết hợp chúng lại, ta có thể duyệt một List, Tuple hoặc Chuỗi:

```
>>> lst = [s, (1, 2, 3), {'abc', 'xyz'}]
>>> for i in range(len(lst)):
...     print(lst[i])
...
How Kteam
(1, 2, 3)
{'abc', 'xyz'}
```

Sự khác nhau giữa sequence scan và indexing scan

Trong bài trước, bạn thấy rằng ta không cần dùng tới hàm **range** vẫn có thể duyệt hết các phần tử của một List. Vậy điều gì khiến chúng ta đôi lúc phải dùng tới hàm **range** để xử lí một List?

Đó là khi ta cần update (cập nhật) List. Hãy xem hai ví dụ sau đây:

Đầu tiên là **sequence scan**

```
>>> lst = [1, 2, 3]
>>> for value in lst:
...     value += 1
...
>>> lst
[1, 2, 3]
```

Biến variable là một biến riêng lẻ, nên không thể cập nhật được List ban đầu.

Còn đối với **indexing scan**

```
>>> lst = [1, 2, 3]
>>> for i in range(len(lst)):
...     lst[i] += 1
```

```
...
>>> lst
[2, 3, 4]
```

Hãy lựa chọn cách sử dụng vòng lặp một cách thông minh phù hợp với mục đích của mình.

Comprehension

Có lẽ bây giờ những **comprehension** không còn phức tạp với các bạn nữa.

Comprehension là một công cụ rất hiệu quả của Python để xử lí rất nhiều việc mà chỉ cần một dòng.

Bên cạnh đó. Người ta còn so sánh những **comprehension** và những đoạn code với chức năng tương tự thì comprehension có tốc độ nhanh hơn.

Lời tác giả:

- Mọi người sẽ phải Ồ lên khi thấy bạn có một comprehension chỉ tốn một dòng và thời gian thực thi nhanh hơn. Thế nên bạn nên luyện tập sử dụng comprehension thường xuyên.
- Sau này khi kết hợp với **anonymous function** là lambda bạn sẽ tạo ra được những thứ mang đậm thương hiệu one-liner.
- Python không khó. Quan trọng là bạn phải nằm lòng các **API** của Python (các chức năng mà ngôn ngữ hỗ trợ) là một trong những thứ đó

Ta có thể tổng quát đơn giản cú pháp của một **comprehension** như sau

Cú pháp:

> **[output-expression for-statement optional-predicate]**

Ở đây Kteam sử dụng **[** cho List, các bạn có thể sử dụng các cặp ngoặc khác nhưng phải để output-expression phù hợp với kiểu dữ liệu. Như dict thì bạn phải để output-expression là một cặp **key-value**.

Một số ví dụ

```
>>> ['--'.join((a.capitalize(), b.upper() + c.lower())) for a, b, c in [('how', 'kteam',
'EDUCATION'), ('chia', 'sẻ', 'FREE')]] # bỏ trống optional-predicate
['How--KTEAMeducation', 'Chia--SẺfree']
```

Nếu không sử dụng **comprehension** thì sẽ như sau:

```
>>> lst = []
>>> for a, b, c in [('how', 'kteam', 'EDUCATION'), ('chia', 'sẻ', 'FREE')]:
...     a = a.capitalize()
...     b = b.upper()
...     c = c.lower()
...     lst.append('--'.join((a, b + c)))
...
>>> lst
['How--KTEAMeducation', 'Chia--SẺfree']


>>> {key:value + 1 for key, value in (('Kteam', 69), ('Tèo', 50), ('Tũn', 14), ('Free
Education', 93)) if value % 2 != 0}
{'Kteam': 70, 'Free Education': 94}
```

Khi không sử dụng **comprehension**

```
>>> dic = {}
>>> for key, value in (('Kteam', 69), ('Tèo', 50), ('Tũn', 14), ('Free Education', 93)):
...     if value % 2 != 0:
...         dic[key] = value + 1
...
>>> dic
{'Kteam': 70, 'Free Education': 94}
```

Giới thiệu hàm enumerate

Giả sử bạn có một danh sách học sinh.

```
>>> student_list = ['Long', 'Trung', 'Giàu', 'Thành']
```

Việc in ra danh sách này thì rất đơn giản.

```
>>> for student in student_list:
...     print(student)
...
Long
Trung
Giàu
Thành
```

Nhưng như vậy thì không rõ ràng cho lắm vì danh sách này không hề có số thứ tự. Bạn nghĩ đến việc sử dụng hàm **range**.

Đó cũng là một cách, nhưng Python có hỗ trợ cho bạn một hàm hay hơn đó chính là **enumerate**. Hàm có cú pháp như sau:

Cú pháp:

```
enumerate(iterable[, start])
```

Nếu start không được gửi vào thì mặc định là **0**

Hàm này là một generator nhờ câu lệnh yield trong hàm. Nó sẽ tạo ra mỗi giá trị là một cặp gồm số thứ tự và giá trị có cấu trúc như sau

```
(start + 0, seq[0]), (start + 1, seq[1]), (start + 2, seq[2]), ...
```

Ví dụ:

```
>>> gen = enumerate(student_list)
>>> gen
<enumerate object at 0x02D6D850>
>>> list(gen)
[(0, 'Long'), (1, 'Trung'), (2, 'Giàu'), (3, 'Thành')]
```

Và khi đó, ta có thể sử dụng vòng for như sau

```
>>> for idx, student in enumerate(student_list):
...     print(idx, '=>', student)
...
0 => Long
1 => Trung
2 => Giàu
3 => Thành
```

Nếu bạn không thích bắt đầu từ số 0 thì ta cũng có thể thay đổi

```
>>> for idx, student in enumerate(student_list, 1):
...     print(idx, '=>', student)
...
1 => Long
2 => Trung
3 => Giàu
4 => Thành
```

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

1. Kết quả là **1**. Chính xác là giá trị thứ hai của biến iter_

```
>>> next(iter_)
1
```

Python là ngôn ngữ thông dịch. Vậy nên nó sẽ đọc từng câu lệnh. Và như đã đề cập trong cách làm việc của vòng lặp này. Nó sẽ lấy giá trị từ **sequence** gán cho biến rồi mới vào trong for-block. Vậy nên sau khi có giá trị, vòng trong for-block mới có lỗi phát sinh. Khi đó, chúng ta đã vừa lấy mất đi một giá trị của biến iter_. Vậy nên khi dùng hàm next thì kết quả sẽ là kết quả thứ hai.

2.

```
>>> set_ = {5, 8, 1, 9, 4}
>>> sum_of_set = 0
>>> for value in set_:
...     sum_of_set += value
...
>>> sum_of_set
27
```

Câu hỏi củng cố

1. Sử dụng **sequence scan** để thay đổi phần tử đầu tiên của mỗi phần tử trong List dưới đây thành **None**

```
>>> lst = [[1, 2, 3], [4, 5, 6]]
Sau khi thay đổi
>>> lst
[[None, 2, 3], [None, 5, 6]]
```

2. Một spiral matrix là một ma trận vuông **nxn** (n cột, n hàng) gồm N^2 số tự nhiên đầu tiên. Trong đó số tăng tuần tự đi xung quanh các mép của mảng xoắn bên trong nó.

Ví dụ với một spiral matrix 5x5 thì sẽ như sau:

```
0   1   2   3   4
15  16  17  18  5
14  23  24  19  6
13  22  21  20  7
12  11  10  9   8
```

Viết một đoạn script yêu cầu nhập số n (chính là số cột - hàng) của một spiral matrix. Sau đó dùng vòng lặp tạo một spiral matrix in ra shell (Nếu in ra số có một chữ số như 0, 1, 2,.. thì thêm trước đó là chữ số 0 -> 00, 01, 02,...)

Với spiral matrix như trên sẽ được in ra như sau:

```
00  01  02  03  04
15  16  17  18  05
14  23  24  19  06
13  22  21  20  07
12  11  10  09  08
```

Đáp án của phần này sẽ được trình bày ở bài tiếp theo. Tuy nhiên, Kteam khuyến khích bạn tự trả lời các câu hỏi để củng cố kiến thức cũng như thực hành một cách tốt nhất!

Kết luận

Qua bài viết này, Bạn đã biết nhiều hơn về VÒNG LẶP FOR TRONG PYTHON.

Ở bài viết sau. Kteam sẽ giới thiệu với các bạn HÀM 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**".