Bài: Kiểu dữ liệu chuỗi trong Python - Phần 2

Xem bài học trên website để ủng hộ Kteam: Kiểu dữ liệu chuỗi 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 sơ cho các bạn KIỂU DỮ LIỆU CHUỖI TRONG PYTHON – Phần 1

Ở bài này chúng ta sẽ tiếp tục đề cập đến **KIỂU DỮ LIỆU CHUỖI** trong Python.

Nội dung chính

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 CHUỖI (Phần 1)

Trong bài học này, chúng ta sẽ cùng tìm hiểu các vấn đề:

- Chuỗi trần là gì?
- Một số toán tử với chuỗi
- Indexing và cắt chuỗi
- Ép kiểu dữ liệu
- Thay đổi nội dung chuỗi

Chuỗi trần là gì?

Nếu bạn còn nhớ những ví dụ lần trước trong bài KIỂU DỮ LIỆU CHUỖI (Phần 1), **Kteam** đưa ra trong phần **Escape Sequence** là gì? Bạn dễ nhận thấy rằng, đôi khi bạn gặp trường hợp không mong muốn có escape sequen. Điển hình như ví dụ sau.

Bạn muốn in ra một dòng chuỗi với nội dung như sau:

> Haizz, \neu mot ngay nao do.

Và trong Python

Python:

```
>>> print('Haizz, \neu mot ngay nao do')
Haizz,
eu mot ngay nao do
```

Kết quả không mong muốn. May thay, bạn biết đó là do tác dụng của **Escape Sequence**. Và bạn cũng biết sử dụng **Escape Sequence** để có được kết quả như mình muốn

Python:

```
>>> print('Haizz, \\neu mot nay nao do')
Haizz,
\neu mot ngay nao do
```

Nhưng hãy đặt vấn đề, ví dụ như bạn thao tác với các đường dẫn file trên hệ điều hành Windows. Các đường dẫn file này sẽ có dạng

> **Ổ_đĩa:\Thư_mục\Thư_mục**

Sẽ rao sao nếu tên thư mục bắt đầu với các chữ cái t, n, a, v, b,... và kết hợp với kí tự \. Nó thành Escape Sequence, kết quả mà bạn không muốn. Và rất nhiều trường hợp khác mà việc bạn sửa Escape Sequence một cách thủ công là không chấp nhận được.

Vì lý do đó, Python cho phép bạn sử dụng một dạng chuỗi, gọi là **CHUỖI TRẦN**. Chuỗi trần này sẽ không quan tấm đến thứ gọi là Escape Sequence.

Cú pháp

> **r**'nội dung chuỗi'

Ví dụ:

Python:

```
>>> a = r'\neu mot ngay'   # chuỗi trần, bỏ qua Escape Sequence \n
>>> print(a)
\neu mot ngay'
```

Sự thật thì, chuỗi trần không phải bỏ qua các Escape Sequence, mà nó sẽ giúp chúng ta sửa những Escape Sequence đó, như cách chúng ta làm

Python:

```
>>> a = r'\neu mot ngay'
>>> a  # nội dung chuỗi trần gán vào biến a
'\\neu mot ngay'
```

Và bạn sẽ phải sử dụng chuỗi trần này một cách thường xuyên, đặc biệt là khi bạn làm việc với BIỂU THỨC CHÍNH QUY (Regular Expression) sẽ được **Kteam** giới thiệu trong tương lai.

Một số toán tử với chuỗi

Toán tử +

Đây là một toán tử rất được hay sử dụng trong việc nối các chuỗi.

Cú pháp

> A +B (với A và B là chuỗi)

Ví dụ:

Python:

```
>>> s = 'hello '
>>> s += 'Python'   # tương tự câu lệnh s = s + 'Python'
>>> s
'hello Python'
>>> s2 = ', good bye'
>>> s3 = s + s2
>>> s3
'hello Python, good  bye'
```

Toán tử *

Không mấy ngôn ngữ lập trình hỗ trợ toán tử này, toán tử này giúp tạo ra một chuỗi nhờ lặp đi lặp lại chuỗi với số lần bạn muốn.

Cú pháp

> A * N (Với A là một chuỗi, N là một số nguyên)

Ví dụ:

Python:

```
>>> 'a' * 3  # tạo ra chuỗi bằng cách lặp lại chuỗi 'a' 3 lần
'aaa'
>>> s = 'abc'
>>> s *= 2  # tương tự câu lệnh s = s * 2
>>> s
'abcabc'
>>> s * 0  # bất cứ chuỗi nào nhân với 0 cũng đều có kết quả là ''
''
>>> 'idoufhaionrewnrwnerlqwrwqr' * 0
''
>>> '8523nsalfnsdf' * -2  # đối với số âm cũng sẽ trả về một chuỗi ''
''
```

Toán tử in

Khi sử dụng toán tử này, bạn chỉ có thể nhận được một trong hai đáp án đó là True hoặc False.

Cú pháp:

> s in A (Với s và A là chuỗi)

Kết quả sẽ là True nếu chuỗi s xuất hiện trong chuỗi A. Ngược lại sẽ là False. (True và False là kiểu dữ liệu Boolean sẽ được Kteam giới thiệu trong bài KIỂU DỮ LIỆU BOOLEAN trong Python)

Ví dụ:

Python:

```
>>> 'a' in 'abc'
True
>>> 'ab' in 'abc'
True
>>> 'ac' in 'abc'
False
```

Các toán tử so sánh với chuỗi

Cú pháp:

> **A** <**toán tử so sánh**> **B** (A và B là 2 chuỗi)

Các toán tử so sánh là: <, >, ==, !=, <=, >=,....

Đầu tiên, ta cần lưu ý là: Việc áp dụng các toán tử so sánh với các chuỗi sẽ trả về một trong 2 giá trị (**True** hoặc **False**)

Khi so sánh 2 ký tự với nhau, Python sẽ so sánh vị trí của chúng trong bảng mã Unicode và trả về kết quả tương ứng

Python:

```
>>> 'ă' < 'â' # vì ‘ă’ đứng sau ‘â’ trong bảng mã Unicode
False
>>> 'a' > 'b' # vì ‘b’ đứng sau ‘a’ trong bảng mã Unicode
False
>>> 'd' < 'g'
True
>>> 'ă' < 'â'
False
>>> 'o' < 'ơ'
True
```

Khi so sánh 2 chuỗi, chương trình sẽ so sánh các **cặp kí tự** của 2 chuỗi theo thứ tự từ trái sang phải. Khi gặp 2 kí tự khác nhau, kết quả so sánh 2 chuỗi sẽ là kết quả khi so sánh 2 kí tự này. Trong trường hợp xét hết tất cả các kí tự của một trong 2 chuỗi mà vẫn chưa tìm thấy điểm khác biệt, thì yếu tố cuối cùng là độ dài của 2 chuỗi. Khi đó, kết quả so sánh 2 chuỗi chính là kết quả khi so sánh độ dài của 2 chuỗi đó.

Python:

```
>>> 'ac' > 'abc' # Ở kí tự thứ hai (có chỉ số index là 1) thì ‘c’ > ‘b’
True
>>> 'z' > 'abcdefz' # Kí tự đầu tiên: ‘z’ > ‘a’
True
>>> 'abc' < 'b' # Kí tự đầu tiên: ‘z’ > ‘a’
True
>>> 'ab' < 'abc' # Đã xét hết 2 cặp kí tự, nhưng không tìm thấy sự khác nhau, chương trình so sánh độ dài của 2 chuỗi
True
```

Để xem vị trí của một kí tự trong bảng mã Unicode, ta sử dụng hàm **ord()**

Cú pháp:

> **ord**(<kí tự>)

Indexing và cắt chuỗi

Indexing

Trong một chuỗi của Python, các kí tự tạo nên chuỗi đó sẽ được đánh số từ 0 tới n – 1 từ trái qua phải và từ **-1** đến **-n** theo chiều từ phải sang trái, với n là số kí tự có trong chuỗi.

Ví dụ: ta có một chuỗi với nội dung là ‘abc xyz’. Ta sẽ mượn biến s giữ dùm ta giá trị này

Python:

```
>>> s = ‘abc xyz’
>>> s
‘abc xyz’
```

Và các kí tự trong chuỗi sẽ được đánh số như sau

a	b	c		x	y	z
0	1	2	3	4	5	6
-7	-6	-5	-4	-3	-2	-1

Dựa vào đây, ta có thể lấy được bất cứ kí tự nào ta muốn trong chuỗi.

Cú pháp

```
<chuỗi>[vị trí]
```

Ví dụ:

Python:

```
>>> s[0]
'a'
>>> s[6]
'z'
>>> s[3]
' '
>>> s[-3]
'x'
>>> s[-2]
'y'
>>> s[-6]
'b'
>>> s[7]  # truy cập vị trí không có trong chuỗi
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: string index out of range
>>> s[-8] # truy cập vị trí không có trong chuỗi
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: string index out of range
>>> s[1.2]  # vị trí phải là một số nguyên, không phải số thực
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: string indices must be integers
>>> s['1']  # vị trí là số nguyên, không phải chuỗi
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: string indices must be integers
>>> s[len(s) - 1]  # truy cập phần tử cuối cùng trong trường hợp ta không biết vị trí cuối
'z'
```

Cắt chuỗi

Đây là một thứ lợi hại của Python. Dựa trên Indexing, Python cho phép chúng ta cắt chuỗi. Đương nhiên, các bạn cần nắm rõ được phương pháp Indexing.

Cú pháp

<chuỗi>[vị trí bắt đầu : vị trí dừng]

Khi ta sử dụng cú pháp này, ta sẽ nhận được một chuỗi. Chuỗi này chính là bản sao của chuỗi mà chúng ta muốn cắt. Và chúng ta sẽ cắt (lấy) từng các kí tự có vị trí từ **<vị trí bắt đầu>** đến **<vị trí dừng>** - 1 và từ trái sang phải.

Ví dụ:

Python:

```
>>> s = 'abc xyz'
>>> s[1:5]  # cắt từng kí tự có vị trí từ 1 đến 4
'bc x'
>>> s[0:3]  # cắt từng kí tự có vị trí từ 0 đến 2
'abc'
```

Như đã giới thiệu, mỗi kí tự được đánh 2 số vị trí, và lẽ dĩ nhiên, ta có thể sử dụng cả ví trí số âm.

Python:

```
>>> s = 'abc xyz'
>>> s[-4: -1]  # cắt từng kí tự có vị trí từ -4 đến -2
' xy'
>>> s[1: -1]  # cắt từng kí tự có vị trí từ 1(-6) đến 5(-2) vì vị trí dừng là 6(-1)
'bc xy'
```

Ở trường hợp ta sử dụng vị trí **vừa số âm vừa số dương**, bạn phải hiểu rằng, vị trí số âm hay số dương thì nó cũng sẽ chỉ ra một vị trí và vị trí đó nó sẽ xem xét để cắt. Thế nên, bạn phải nắm rõ được phần INDEXING.

Bên cạnh việc đặt các giá trị bắt đầu, cũng như là vị trí dừng, ta có thể dùng giá trị None trong một vài trường hợp đặc biệt:

Python:

```
>>> s = 'abc xyz'
>>> s[1:None]  # lấy các kí tự có vị trí 1 đến hết chuỗi
'bc xyz'
>>> s[3:None] # lấy các kí tự có vị trí 3 đến hết chuỗi
' xyz'
>>> s[1:]  # đặc biệt, ta chỉ cần bỏ trống, Python sẽ tự hiểu là None
'bc xyz'
>>> s[-3:]
'xyz'
```

Đó là bạn đặt None ở vị trí dừng. **Sẽ ra sao nếu bạn đặt None ở vị trí bắt đầu?**

Câu trả lời khi ta đặt None ở vị trí bắt đầu thì ta sẽ cắt chuỗi từ vị trí đầu tiên.

Python:

```
>>> s = 'abc xyz'
>>> s[None: 4]  # lấy các kí tự có vị trí từ 0 đến 3
'abc '
>>> s[:-1]  # ta cũng có thể để trống, Python sẽ tự hiểu là None
'abc xy'
>>> s[:]  # một cách sao chép chuỗi
'abc xyz'
```

Lưu ý: Khi bạn đã đặt None ở vị trí bắt đầu, có nghĩa vị trí bắt đầu là 0, và khi đặt None ở vị trí kết thúc, thì có nghĩa là vị trí kết thúc sẽ là n với n là số kí tự trong chuỗi.

Như đã đề cập ở trên, việc cắt chuỗi này sẽ được cắt từ trái qua phải. **Vậy nếu muốn cắt từ phải qua trái thì sao?**

Vì phát sinh vấn đề đó, Python đã hỗ trợ chúng ta một cú pháp cắt khác.

Cú pháp

> <chuỗi>[vị trí bắt đầu : vị trí dừng : bước]

Với cú pháp đầu tiên, thì bạn không cần phải nhập số bước và số bước này sẽ được đặt là 1. Có nghĩa là vị trí của kí tự tiếp theo hơn vị trí của kí tự kế tiếp 1 đơn vị (tính theo vị trí số dương).

Python:

```
>>> s = 'abc xyz'
>>> s[2: 5: 1]  # ta có bước bằng 1
'c x'
>>> s[2:5]  # bước mặc định là 1
'c x'
```

Hãy nhớ rằng, bước chính là thứ để tính được vị trí tiếp theo cách vị trí trước đó bao nhiêu đơn vị.

Python:

```
>>> s = 'abc xyz'
>>> s[1: 7: 2]  #  bước là 2
'b y'
```

Ở ví dụ trên, với bước là 2, ta sẽ có các vị trí trong khoảng 1 đến 6 đó là 1, 3, 5. Vì thế các kí tự ở vị trí này sẽ là kết quả của phép cắt trên.

Ta có thể điều chỉnh việc cắt từ trái sang phải thành phải sang trái bằng việc đặt bước là số âm.

Python:

```
>>> s = 'abc xyz'
>>> s[1: 3]  # bắt đầu ở 1 và dừng ở 3. Các vị trí lấy là 1 và 2
'bc'
>>> s[3:1:-1]  # bắt đầu ở 3 và dừng ở 1. Các vị trí lấy là 3, 2
' c'
```

Một lưu ý nhỏ khi các bạn đặt bước là một số âm, thì vị trí bắt đầu mà bạn để là None thì nó sẽ được đặt là n – 1(-1) với n là độ dài chuỗi. Còn với vị trí dừng thì sẽ là cắt hết trọn vẹn tới đầu chuỗi có nghĩa là vị trí dừng ở trường hợp này không phải là 0.

Python:

```
>>> s = 'abc xyz'
>>> s[4::-1] # lấy các kí tự có vị trí từ 4 đến 0
'x cba'
>>> s[::-1]  # một cách lấy chuỗi ngược nhờ có bước âm.
'zyx cba'
```

> **Lưu ý:** bạn không được phép đặt bước bằng 0

Python:

```
>>> s = 'abc xyz'
>>> s[::0]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: slice step cannot be zero
```

Ngoài ra, nếu <**vị trí bắt đầu**> hoặc <**vị trí dừng**> là một giá trị vượt ngoài phạm vi của chuỗi, thì python cũng sẽ cho chúng ta cắt chuỗi một cách tối ưu nhất:

Ví dụ:

Python:

```
>>> a = '123456'
>>> a[1: 100] // lấy từ kí tự a[0] đến cuối chuỗi
'23456'
>>> a[-1000: 1000] // lấy toàn bộ chuỗi
'123456'
>>> a[-10:-3] // lấy từ a[-6] đến a[-4]
'123'
```

Ép kiểu dữ liệu

Như bạn đã biết. Hai biến a và b dưới đây là khác nhau

Python:

```
>>> a = ‘69’
>>> b = 69
>>> type(a)  # biến a thuộc lớp ‘str’, kiểu dữ liệu chuỗi
<class ‘str’>
>>> type(b)  # biến b thuộc lớp ‘int’, kiểu dữ liệu số nguyên
<class ‘int’
```

Vì lí do đó, bạn sẽ nhận ra được vì sao có sự khác biệt trong hai biểu thức sau đây.

Python:

```
>>> ‘69’ * 2  # một chuỗi nhân với một số
‘6969’
>>> 69 * 2   # một số nhân với một số
138
```

Đôi lúc, bạn sẽ nhận được một số dưới dạng một chuỗi. Thế nên, trong trường hợp bạn muốn tính toán. Bạn phải đưa nó về từ kiểu dữ liệu chuỗi sang kiểu dữ liệu số. Ở trường hợp ví dụ ở đây là số nguyên.

Một hàm lí tưởng để làm việc đó chính là hàm **int**()

Cú pháp:

```
int(<giá trị>)
```

Ví dụ:

Python:

```
>>> a = '69'
>>> int(a)   # trả về giá trị được chuyển về số nguyên từ giá  trị của biến a
69
>>> type(a)  # biến a thuộc lớp 'str'
<class 'str'>
>>> b = int(a)  # biến b giữ giá trị được chuyển sang số nguyên từ giá trị của biến a
>>> type(b)
<class 'int'>
>>> c = '-3'   # biến c giữ chuỗi với giá trị '-3'
>>> type(c )  # và dĩ nhiên giá trị biến c thuộc lớp 'str'
<class 'str'>
>>> d = int(c )   # trả về giá trị được chuyển sang số nguyên từ giá trị của biến c
>>> d
-3
>>> type(d)   # số nguyên, thuộc lớp 'int'
<class 'int'>
```

Đó là số nguyên, còn với số thực, xin bạn hãy lưu ý cho điều này. Khi sử dụng hàm **int()**. Ta có khả năng biến đổi một **số thực thành số nguyên** bằng cách bỏ đi phần thập phân.

Ví dụ:

Python:

```
>>> a = 3.1  # a là một biến giữ giá trị số thực
>>> type(a)
<class 'float'>
>>> b = int(a)   # trả về một giá trị được chuyển đổi thành số nguyên từ giá trị của biến a
>>> b
3
>>> type(b)
<class 'int'>
>>> int(3.9)  # bỏ đi phần thập phân, không phải làm tròn, các bạn lưu ý
3
```

> **Lưu ý:**
>
> Bạn sẽ **không thể** chuyển đổi một số thực dưới dạng chuỗi bằng hàm int

Ví dụ:

Python:

```
>>> int('3.1')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: '3.1'
```

Đương nhiên ta có giải pháp thay thế. Đó là hàm **float**().

Cú pháp

> **float**(<giá trị>)

Ví dụ:

Python:

```
>>> a = ‘3.1’
>>> type(a)
<class ‘str’>
>>> b = float(a)
>>> b
3.1
>>> type(b)
<class ‘float’>
```

Vậy giả sử ta muốn làm điều ngược lại, chuyển một giá trị nào đó từ **số** sang một **chuỗi** thì sao ? Những lúc như vậy, ta sẽ cần sự trợ giúp của hàm **str()**. Hàm này cho phép chuyển bất kì một giá trị nào thành chuỗi miễn là nó được **hỗ trợ**

Cú pháp:

> **str**(<giá trị>)

Ví dụ:

:

```
>>> a = 143
>>> str(a) # Chuyển a từ int sang string
'143'
>>> b = 12.32
>>> str(b) # Chuyển b từ float sang string
'12.32'
>>> t = [1, 2]
>>> str(t) # Chuyển t từ một list sang string (ta sẽ cùng tìm hiểu về list trong các bài tiếp theo)
'[1, 2]'
```

Thay đổi nội dung chuỗi

Trở về với khái niệm Indexing. Bạn có nghĩ tới việc thay đổi nội dung chuỗi nhờ Indexing không? Nếu như bạn đã từng học với các ngôn ngữ như Pascal, C, C++ thì có thể bạn sẽ sử dụng phương pháp Indexing.

Nhưng điều đáng buồn, Python **không** cho phép điều đó

Python:

```
>>> s = ‘abc xyz’
>>> s[0]
‘a’
>>> s[0] = ‘k’
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
```

Bạn chỉ có thể thay thế nó một cách gián tiếp giá trị chuỗi mà biến của bạn lưu giữ bằng cách sử dụng việc cắt chuỗi và toán tử + để tạo ra một chuỗi mới và gán lại vào biến của bạn.

Ví dụ:

Python:

```
>>> s = ‘abc xyz’
>>> s = ‘k’ + s[1:]  # lấy các kí tự từ vị trí 1 đến hết chuỗi
>>> s
‘kbc xyz’
```

Vì ta không thể thay thế nội dung chuỗi, do đó kiểu dữ liệu chuỗi là một đối tượng có thể băm (hashable object).

Vì nó là một hashable object. Nên ta có thể sử dụng hàm **hash**.

Python:

```
>>> hash('abc')
-720462249
```

> **Lưu ý:** Khi chạy một chương trình Python, thì giá trị của hàm hash lên một giá trị nhất định không thay đổi. Nhưng giá trị đó sẽ thay đổi nếu như đó là lần chạy tiếp theo. Do đó, kết quả của bạn có thể sẽ khác so với Kteam. Và khi bạn chạy chương trình lần tiếp theo cũng sẽ nhận được kết quả khác so với kết quả ban đầu của bạn.

Bạn có thể hiểu nôm na hashable object là hằng, một giá trị không bao giờ thay đổi. Và có một vài trường hợp bắt buộc bạn phải sử dụng kiểu dữ liệu là hashable object điển hình như khóa (key) trong kiểu dữ liệu Dict của Python (một kiểu dữ liệu sẽ được Kteam giới thiệu ở bài DICTONARY TRONG PYTHON).

Hashable object đôi lúc cũng có thể gọi là immutable object.

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 KIỂU DỮ LIỆU CHUỖI TRONG PYTHON - Phần 1 .

1. Các chuỗi hợp lệ là

Python:

```
'nasdfiuqwnerp', "234a'adadf",  ''
```

2. Sự khác nhau giữa hai biến a và b dưới đây là

Python:

```
>>> type(a)   # biến a thuộc lớp int vì là một số
<class 'int'>
>>> type(b)  # biến b thuộc lớp str vì là một chuỗi
<class 'str'>
```

3. Các Escape Sequence là

Python:

```
Chuỗi 1: không có

Chuỗi 2: `\\`

Chuỗi 3: `\\`
```

Câu hỏi củng cố

1. Có bao nhiêu escape sequence trong giá trị của biến s dưới đây?

Python:

```
>>> s = r'\gte\teng\n\vz\adf\t'
```

2. Giá trị của biến s sau khi thực hiện toán tử + dưới đây là gì?

Python:

```
>>> s = ‘’ + ‘’ + ‘’ + ‘’ + ‘’ + ‘’ + ‘’ + ‘’ + ‘’
```

3. Cho biến s với giá trị chuỗi

Python:

```
>>> s = ‘abc xyz’
```

Phép cắt chuỗi nào dưới đây sẽ nhận được kết quả là một chuỗi rỗng

a. s[:]
b. s[len(s):]
c. s[1:1]
d. s[0::-1]
e. s[0:0:-1]

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

Bài viết này đã giới thiệu thêm cho các bạn KIỂU DỮ LIỆU CHUỖI TRONG PYTHON.

Ở bài sau, Kteam sẽ tiếp tục nói về KIỂU DỮ LIỆU CHUỖI TRONG PYTHON (Phần 3) về phần định dạng và một số phương thức của chuỗi

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**".