Kiểu `str` trong Python

December 18, 2020

1 Kiểu dữ liệu chuỗi ký tự

1.1 Giới thiệu

Chuỗi ký tự, hay `str`, là một trong những kiểu dữ liệu cơ bản của python.

`str` là một chuỗi các ký tự được nằm trong (được giới hạn bởi) dấu nháy đơn `''` hoặc nháy kép `""`

Ví dụ :

```
In [1]: print("Đây là một chuỗi ký tự")
```

```
Đây là một chuỗi ký tự
```

```
In [2]: print('Đây cũng là một chuỗi ký tự')
```

```
Đây cũng là một chuỗi ký tự
```

Về cơ bản, độ dài `str` của python chỉ bị giới hạn bởi bộ nhớ của máy tính. Nghĩa là, nếu bộ nhớ cho phép, bạn có thể tạo ra một `str` chứa 1,000,000,000 ký tự.

Và bạn cũng có thể tạo ra một `str` trống, không chứa ký tự nào

```
In [1]: type('')
```

```
Out[1]: str
```

Để biết được độ dài (hay số ký tự) của một `str`, bạn có thể dùng hàm `len()` như sau :

```
In [2]: len('Đây là một chuỗi rất dài!')
```

```
Out[2]: 25
```

1.2 Cách giới hạn một `str`

Như ở phần giới thiệu, một `str` của python có thể được giới hạn bằng hai cách, đó là dùng `''` hoặc `""`. Tại sao lại thế?

Chẳng hạn, bạn muốn tạo chuỗi như thế này :

```
In [5]: 'I've a dream'
```

```
  File "<ipython-input-5-2bb91a7c806b>", line 1
    'I've a dream'
       ^
SyntaxError: invalid syntax
```

Như các bạn có thể thấy, khi gặp dấu ' thứ hai, python đã tự động xác định I là str.

Phần còn lại, do không nằm trong cú pháp nào của python nên bị báo lỗi.

Để giải quyết vấn đề này, người ta có 3 cách : 1. Dùng "" để giới hạn chuỗi khi muốn dùng '' trong chuỗi và ngược lại. 2. Dùng ký tự đặc biệt \ 3. Dùng đến nháy ba '''...''' hoặc """..."""

Cách thứ nhất, bạn có thể xem những ví dụ sau đây :

```
In [1]: a = "I've a dream"
        print(a)
```

```
I've a dream
```

```
In [2]: a = 'She said: "I want to go home"'
        print(a)
```

```
She said: "I want to go home"
```

Tuy nhiên, cách thứ nhất chỉ hiệu quả trong trường hợp chỉ có 1 dấu nằm lồng trong nhau, cùng xem ví dụ sau :

```
In [3]: b = "'She said : "I've a dream"'"
        print(b)
```

```
  File "<ipython-input-3-68b12521463c>", line 1
    b = 'She said : "I've a dream"'
                        ^
SyntaxError: invalid syntax
```

Như các bạn có thể thấy, trong trường hợp này, việc tồn tại của I've khiến cho không thể giới hạn chuỗi một cách hiệu quả. Vì vậy, người ta phải sử dụng đến ký tự \

1.3 Ký tự \

Trong nhiều ngôn ngữ lập trình, người ta thường dùng ký tự \ (xuyệt phải) với ý nghĩa là một ký tự đặc biệt.

Khi một trình biên dịch hoặc thông dịch gặp ký tự \ này, trình biên dịch hoặc thông dịch sẽ hiểu rằng một (hoặc vài) ký tự đi sau \ có ý nghĩa khác với bình thường.

Ký tự \ còn được gọi là escaped character. Hàm ý là một (hoặc vài) ký tự theo sau nó được thoát (escaped) khỏi ý nghĩa ban đầu.

Một số chuỗi thường gặp trong python

Chuỗi	Ý nghĩa
\t	Nhảy 1 tab
\n	Xuống dòng
\'	Ký tự '
\"	Ký tự "
\\	Ký tự \

Tuy nhiên, các chuỗi này chỉ có thể thể hiện được khi dùng lệnh `print()`
Một số ví dụ :

```
In [4]: print('She said:\n "I\'ve a \t dream"')
```

```
She said:
 "I've a          dream"
```

```
In [4]: print('- Ai là người lấy cắp nỏ thần của An Dương Vương?\n- Dạ, không liên qu
```

```
- Ai là người lấy cắp nỏ thần của An Dương Vương?
- Dạ, không liên quan đến em, em không biết bạn Vương!
- Để em đi báo công an ạ!
```

```
In [28]: print('a\t+\tb\t=\tc')
```

```
a       +       b       =       c
```

1.4 Dấu nháy ba `'''...'''` hoặc `"""..."""`

Ngoài ra, python còn có một cách để giới hạn chuỗi, đó là dùng dấu nháy ba. Công dụng chính của dấu nháy ba là cho phép bạn viết một `str` trên nhiều dòng khác nhau.

```
In [29]: print('a
         b')
```

```
          File "<ipython-input-29-8de7381e0d36>", line 1
        print('a
              ^
    SyntaxError: EOL while scanning string literal
```

```
In [32]: print('''Trăm năm trong cõi người ta
         Ai ai cũng phải thở ra hít vào
         Gần gần như cái nước Lào
         Người người đều phải hít vào thở ra''')
```

```
Trăm năm trong cõi người ta
Ai ai cũng phải thở ra hít vào
Gần gần như cái nước Lào
Người người đều phải hít vào thở ra
```

Ngoài ra, nó có công dụng phụ là bạn có thể dùng *nháy đơn* và *nháy kép* trong `str` mà không cần đi kèm với \

2 Một số thao tác trên chuỗi ký tự

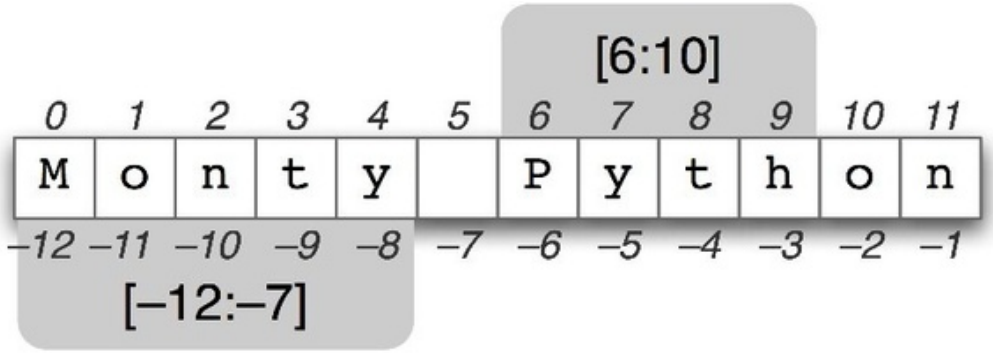
2.1 Trích xuất chuỗi con

Về cơ bản, bạn có thể xem chuỗi là một mảng các ký tự, vì vậy, bạn có thể lấy một chuỗi con từ chuỗi cho trước theo cách sau :

`<tên_chuỗi>[bắt_đầu:kết_thúc]`

Với ý nghĩa là, trích chuỗi con từ `tên_chuỗi`, lấy từ vị trí `bắt_đầu` cho đến **ngay trước** vị trí `kết_thúc`

Python hỗ trợ hai cách đánh số `str`, đó là số thứ tự dương và số thứ tự âm như hình sau :



Cách đánh số `str`

Một số điểm cần lưu ý : - Bạn có thể bỏ trống `bắt_đầu`, python sẽ lấy từ đầu chuỗi. - Bạn cũng có thể bỏ trống `kết_thúc`, python sẽ lấy đến hết chuỗi. - Nhắc lại, trong trường hợp có `kết_thúc`, Python sẽ **không** lấy phần tử ở vị trí `kết_thúc`. - Bạn có thể sử dụng cách đánh số thứ tự dương hoặc số thứ tự âm hoặc cả hai đều được.

Cùng xem các ví dụ sau :

```
In [5]: a = 'Hello'
        a[0:1]

Out[5]: 'H'

In [40]: # chỉ rõ bắt đầu và kết thúc
         a[1:4]

Out[40]: 'ell'

In [41]: # bỏ trống bắt đầu
         a[:3]

Out[41]: 'Hel'

In [42]: # bỏ trống kết thúc
         a[2:]

Out[42]: 'llo'

In [43]: # dùng cách đánh số thứ tự âm
         a[-3:]
```

```
Out[43]: 'llo'
```

```
In [44]: # dùng kết hợp số thứ tự âm và dương
         a[1:-1]
```

```
Out[44]: 'ell'
```

Câu hỏi : 1. Trong trường hợp `bắt_đầu` và `kết_thúc` cùng tính âm dương, nếu như `bắt_đầu >= kết_thúc` thì sẽ trả về kết quả như thế nào? 2. Trong cách đánh số thứ tự dương, nếu `kết_thúc` lớn hơn độ dài của chuỗi thì chuyện gì sẽ xảy ra?

2.2 Nối chuỗi

Để nối hai hay nhiều chuỗi với nhau, bạn có thể dùng một trong 2 cách sau : 1. Dùng phép + 2. Dùng dấu `()`

Cùng theo dõi ví dụ các ví dụ :

```
In [4]: print('a \n b')
```

```
a
 b
```

```
In [49]: print((
             'Apple'
             ', '
             'Banana'
         ))
```

```
Apple, Banana
```

Tuy nhiên, cách 2 chỉ áp dụng được cho chuỗi chứ không áp dụng được cho biến kiểu chuỗi

```
In [0]: a = 'Apple'
        b = 'Banana'
```

```
In [52]: print(a + '\n' + b)
```

```
Apple
Banana
```

```
In [55]: print((
             a
             '\n'
             b
         ))
```

```
          File "<ipython-input-55-aba0cb4ff504>", line 3
        '\n'
          ^
    SyntaxError: invalid syntax
```

2.3 Một số phương thức của `str`

Mục này sẽ giới thiệu một số phương thức của kiểu `str`.

Chúng ta cũng quy ước gọi chuỗi thực hiện phương thức là chuỗi gốc.

Phương thức `.upper()` biến toàn bộ chuỗi gốc thành ký tự in hoa.

Phương thức `.lower()` biến toàn bộ chuỗi gốc thành ký tự in thường.

```
In [6]: s = 'a á à ả ã ạ ơ ớ ờ ở ỡ ợ'
        print(s.upper())
        print(s.lower())
```

```
A Á À Ả Ã Ạ Ơ Ớ Ờ Ở Ỡ Ợ
a á à ả ã ạ ơ ớ ờ ở ỡ ợ
```

Phương thức `.strip()` loại bỏ toàn bộ khoảng trắng ở đầu và cuối chuỗi gốc.

```
In [7]: s2 = '   \tHello  !  '
        print(s2)
        print(s2.strip())
```

```
   	Hello  !
Hello  !
```

Phương thức `.startswith(<chuỗi_thử>)` sẽ kiểm tra xem chuỗi gốc có bắt đầu bằng `chuỗi_thử` hay không.

Phương thức `.endswith(<chuỗi_thử>)` sẽ kiểm tra xem chuỗi gốc có kết thúc bằng `chuỗi_thử` hay không.

```
In [8]: s3 = 'Apple, Banana, Cherry'
        print(s3.startswith('Apple'))
        print(s3.endswith('Banana'))
```

```
True
False
```

Phương thức `.find(<chuỗi_tìm>)` sẽ tìm kiếm `chuỗi_tìm` trong chuỗi gốc : - Nếu có `chuỗi_tìm` thì sẽ trả về vị trí đầu tiên xuất hiện. - Nếu không có sẽ trả về -1

```
In [10]: s4 = 'Hello'
         print(s4.find('ll'))
         print(s4.find('la'))
```

```
2
-1
```

Phương thức `.split(<chuỗi_tách>)` sẽ tách chuỗi gốc theo `chuỗi_tách`.

```
In [17]: s5 = 'Apple, Banana, Cherry'
         print(s5.replace(',', '.'))
         print(s5)
```

```
Apple. Banana. Cherry
Apple, Banana, Cherry
```

Câu hỏi : 1. Nếu `chuỗi_tách` không tồn tại trong chuỗi gốc, điều gì sẽ xảy ra khi gọi `.split(<chuỗi_tách>)`? 2. Bạn có thể bỏ trống phần `chuỗi_tách`, nhưng điều gì sẽ xảy ra khi gọi `.split()`?

Trên đây là một số phương thức của chuỗi, bạn có thể xem toàn bộ phương thức của chuỗi tại đây