Bài 13: KIỂU DỮ LIỆU LIST TRONG PYTHON (Phần 2)

Xem bài học trên website để ủng hộ Kteam: KIỂU DỮ LIỆU LIST 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 cho các bạn KIỂU DỮ LIỆU LIST TRONG PYTHON – Phần 1.

Ở bài này Kteam sẽ tiếp tục đề cập đến **KIỂU DỮ LIỆU LIST TRONG PYTHON** – Các phương thức List.

Nội dung

Các kiến thức để có thể hiểu tốt được bài viết này

- 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 SỐ và KIỂU DỮ LIỆU CHUỖI trong Python.
- KIỂU DỮ LIỆU LIST trong Python

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

- Giới thiệu về phương thức của kiểu dữ liệu List trong Python
- Các phương thức tiện ích
- Các phương thức cập nhật
- Các phương thức xử lí

Giới thiệu về phương thức của kiểu dữ liệu List trong Python

Kiểu dữ liệu List của Python có một số phương thức giúp chúng ta xử lí các vấn đề liên quan đến nó. Kteam sẽ giúp bạn tìm hiểu các phương thức đó.

Một số phương thức Kteam sẽ không nói rõ về nó vì có một số kiến thức bạn chưa nắm được. Điển hình đó là hàm.

Bên cạnh đó có một số phương thức có dạng biến thể là một hàm sẽ được Kteam đề cập ở một bài trong tương lai

---

Các phương thức tiện ích

Phương thức count

Cú pháp:

```
<List>.count(sub, [start, [end]])
```

Công dụng: Giống với phương thức count của kiểu dữ liệu chuỗi.

- Trả về một số nguyên, chính là số lần xuất hiện của **sub** trong chuỗi.
- Còn **start** và **end** là số kĩ thuật slicing (lưu ý không hề có bước).

```
>>> Kteam = [1, 5, 1, 6, 2, 7]
>>> Kteam.count(1)
2
>>> Kteam.count(3)
0
```

Phương thức index

Cú pháp:

```
<List>.index(sub[, start[, end]])
```

Công dụng: Tương tự phương thức index của kiểu dữ liệu chuỗi.

```
>>> Kteam = [1, 2, 3]
>>> Kteam.index(2)
1
>>> Kteam.index(4)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: 4 is not in list
```

Phương thức copy

Cú pháp:

```
<List>.copy()
```

Công dụng: Trả về một List tương tự với List[:]

```
>>> lst = [1, 2, 3]
>>> another_lst = lst.copy()  # tương tự lst[:]
>>> another_lst[0] = 4
>>> another_lst
[4, 2, 3]
>>> lst
[1, 2, 3]
```

Phương thức clear

Cú pháp:

```
<List>.clear()
```

Công dụng: Xóa mọi phần tử có trong List.

Lưu ý: Các phiên bản Python 2.X hoặc dưới Python 3.2 sẽ không có phương thức này

```
>>> Kteam = [1, 2, 3]
>>> Kteam.clear()
>>> Kteam
[]
```

Phương thức trên bản chất không như những cách gán với một List rỗng. Giống như dưới đây:

```
>>> lst = []
>>> lst = list()
```

Phương thức clear sẽ xóa đi các phần tử ở trong List. Các bạn sẽ biết thêm khi biết tới **câu lệnh del** sẽ được Kteam giới thiệu trong các bài sau.

Để thể hiện rõ sự khác biệt giữa hai trường hợp trên. Kteam sẽ lấy ví dụ để minh họa:

- Bạn còn nhớ ví dụ về việc Tèo và gấu của tèo dùng chung số tiền chứ?

```
>>> tien_teo = [50]
>>> tien_teo
[50]
>>> tien_gau_cua_teo = tien_teo  # Tèo và gấu của Tèo đang dùng chung 50 nghìn
>>> tien_gau_cua_teo
[50]
>>> tien_gau_cua_teo = []  # ta gán lại số tiền cô gấu của Tèo là một List rỗng
>>> tien_gau_cua_teo
[]
>>> tien_teo # và đương nhiên, tiền của Tèo không bị ảnh hưởng
[50]
```

Tiếp đến, ta sẽ dùng **phương thức clear**.

```
>>> tien_teo = [50]
>>> tien_teo
[50]
>>> tien_gau_cua_teo = tien_teo # Tèo và gấu của Tèo đang dùng chung 50 nghìn
>>> tien_gau_cua_teo
[50]
>>> tien_gau_cua_teo.clear()  # xử dụng phương thức clear
>>> tien_gau_cua_teo
[]
>>> tien_teo # tiền của Tèo đã bị xóa theo
[]
```

Các phương thức cập nhật

Phương thức append

Cú pháp:

```
<List>.append(x)
```

Công dụng: Thêm phần tử x vào cuối List

```
>>> howkteam = [1, 2]
>>> howkteam.append(3)
>>> howkteam
[1, 2, 3]
>>> howkteam.append([4, 5]) # chú ý trường hợp này
>>> howkteam
[1, 2, 3, [4, 5]]
```

Phương thức extend

Cú pháp:

```
<List>.extend(iterable)
```

Công dụng: Thêm từng phần tử một của iterable vào cuối List.

```
>>> Kteam = [1, 2, 3]
>>> Kteam.extend([4, 5]) # xem sự khác biệt giữa append và extend
>>> Kteam
[1, 2, 3, 4, 5]
>>> Kteam.extend([[6, 7], 8])
```

```
>>> Kteam
[1, 2, 3, 4, 5, [6, 7], 8]
```

Phương thức insert

Cú pháp:

```
<List>.insert (i, x)
```

Công dụng: Thêm phần **x** vào vị trí **i** ở trong List.

```
>>> kteam = [1, 2, 3]
>>> kteam.insert(1, 8) # thêm phần tử 8 vào trong List kteam ở vị trí 1
>>> kteam
[1, 8, 2, 3]
```

Nếu vị trí **i** lại lớn hơn hoặc bằng số phần tử ở trong List thì kết quả sẽ tương tự như phương thức append.

```
>>> kteam= [1, 2, 3]
>>> kteam.insert(4, 20) # vị trí 4, nhưng trong List chỉ có 3 phần tử
>>> kteam
[1, 2, 3, 20]
>>> kteam.insert(len(kteam), 5) # vị trí thứ 4, bằng số phần tử trong List
>>> kteam
[1, 2, 3, 20, 5]
```

Nếu vị trí **i** là một số âm, bạn cần lưu ý kỹ ví dụ sau. Bạn chắc vẫn còn nhớ về việc indexing với vị trí là một số âm? nếu không nhớ bạn có thể xem lại bài KIỂU DỮ LIỆU CHUỖI TRONG PYTHON – Phần 2 trước khi vào ví dụ này.

```
>>> kter = [1, 2, 3]
>>> kter[-1]
3
>>> kter[-2]
2
>>> kter[-3]
1
```

Khi bạn insert mà lại dùng vị trí **i** là số âm, thì vị trí được insert sẽ là **i –1**.

```
>>> kteam = [1, 2, 3]
>>> kteam[-1]
3
>>> kteam.insert(-1, 4) # thêm vào vị trí (-1 – 1) là -2
>>> kteam
[1, 2, 4, 3]
```

Nếu vị trí **i –1** (đang xét indexing âm) không có trong List, mặc định, phần tử **x** sẽ được thêm vào đầu List

```
>>> kteam= [1, 2, 3]
>>> kteam[-20] # không có phần tử -20 trong List
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
>>> kteam.insert(-20, 0)
>>> kteam
[0, 1, 2, 3]
```

Phương thức pop

Cú pháp:

```
<List>.pop([i])
```

Công dụng: Bỏ đi phần tử thứ **i** trong List và trả về giá trị đó. Nếu vị trí **i** không được cung cấp, phương thức này sẽ tự bỏ đi phần tử cuối cùng của List và trả về giá trị đó.

```
>>> kter= [1, 2, 3, 4, 5, 6]
>>> kter.pop(3)
4
>>> kter
[1,2, 3, 5, 6]
>>> kter.pop(-3)
>>> kter.pop(-3)
3
>>> kter
[1, 2, 5, 6]
>>> kter.pop() # mặc định sẽ pop phần tử cuối cùng nằm trong List
6
>>> kter
[1, 2, 5]
```

Phương thức remove

Cú pháp:

```
<List>.remove(x)
```

Công dụng: Bỏ đi phần tử đầu tiên trong List có giá trị **x**. Nếu trong List không có giá trị x sẽ có lỗi được thông báo

```
>>> kteam = [1, 5, 6, 2, 1, 7]
>>> kteam.remove(1)
>>> kteam
[5, 6, 2, 1, 7]
>>> kteam.remove(3)
Traceback (most recent call last):
```

```
  File "<stdin>", line 1, in <module>
ValueError: list.remove(x): x not in list
```

Các phương thức xử lí

Phương thức reverse

Cú pháp:

```
<List>.reverse()
```

Công dụng: Đảo ngược các phần tử ở trong List.

```
>>> kteam= [1, 2, 3]
>>> kteam.reverse()
>>> kteam
[3, 2, 1]
```

Phương thức sort

Đây là phương thức mà Kteam sẽ chỉ giới thiệu sơ lược về nó. Kteam sẽ bỏ qua **key** trong phần giới thiệu cú pháp của phương thức bên dưới.

Cú pháp:

```
<List>.sort(key=None,  reverse=False)
```

Công dụng: Sắp xếp các phần tử từ bé đến lớn bằng cách so sánh trực tiếp.

```
>>> howkteam= [3, 6, 7, 1, 2, 4]
>>> howkeam.sort()
>>> howkteam
[1, 2, 3, 4, 6, 7]
```

Vì sao nói nó là so sánh trực tiếp. Bởi vì không chỉ số, nó còn so sánh cả chuỗi, cả List, và mọi thứ khác.

```
>>> lst = ['k', 'free', '9kteam', 'howkteam']
>>> lst.sort()
>>> lst
['9kteam', 'free', 'howkteam', 'k']
```

Ghi nhớ rằng, các phần tử phải có thể so sánh với nhau. Trường hợp dưới đây bạn không thể so sánh chuỗi với số được, do đó sẽ có lỗi hiện lên.

```
lst = ['kteam', 69]
>>> lst.sort()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: '<' not supported between instances of 'int' and 'str'
```

Chúng ta sẽ nói đến từ khóa **reverse**. Từ khóa này bạn chỉ có thể cho 2 giá trị, một là **True**, hai là **False**.

- Nếu là False, các phần tử được sắp xếp từ bé đến lớn, còn ngược lại là từ lớn đến bé.

```
>>> kteam = [6, 8, 2, 5, 1, 10, 4]
>>> true_reverse = kteam.copy() #tạo một bản sao của kteam và không ảnh hưởng đến kteam
>>> kteam.sort() # không đưa giá trị cho reverse thì mặc định là False
>>> true_reverse.sort(reverse=True)
>>> kteam
[1, 2, 4, 5, 6, 8, 10]
>>> true_reverse
[10, 8, 6, 5, 4, 2, 1]
```

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 LIST TRONG PYTHON – Phần 1.

1. Những đáp án c, d là các cách khởi tạo đúng
2. Không, vì ta có thể thay đổi nội dung của List
3. Đáp án

```
>>> code = s.split('&&')[-1].split('%%')[0]
```

Câu hỏi củng cố

1. Chuyện gì xảy ra khi ta dùng phương thức pop lên một List rỗng

```
>>> lst = list()
>>> lst
[]
>>> lst.pop
```

2. Ta có thể sắp xếp được List dưới đây bằng phương thức sort hay không?

```
>>> lst = [[1, 2], ['abc', 'def']]
```

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 hiểu thêm về các phương thức của KIỂU DỮ LIỆU LIST TRONG PYTHON

Ở bài sau. Kteam sẽ giới thiệu tới bạn một container nữa đó chính là KIỂU DỮ LIỆU TUPLE 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**".