Bài 14: KIỂU DỮ LIỆU TUPLE TRONG PYTHON

Xem bài học trên website để ủng hộ Kteam: KIỂU DỮ LIỆU TUPLE TRONG PYTHON

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 các bài trước, Kteam đã giới thiệu đến bạn KIỂU DỮ LIỆU LIST, một container tuyệt vời trong Python

Ở bài này Kteam sẽ giới thiệu tới bạn một container khác đó chính **KIỂU DỮ LIỆU TUPLE** 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.
- KIỂU DỮ LIỆU SỐ và KIỂU DỮ LIỆU CHUỖI trong Python.
- KIỂU DỮ LIỆU LIST trong Python

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

- Giới thiệu về Tuple trong Python.
- Cách khởi tạo Tuple.
- Một số toán tử với Tuple trong Python.

- Indexing và cắt Tuple trong Python.
- Thay đổi nội dung Tuple trong Python.
- Ma trận.
- Tuple có phải luôn luôn là một Hash object?
- Các phương thức của Tuple.
- Khi nào thì chọn Tuple thay cho List?

Giới thiệu về Tuple trong Python

Tuple là một container cũng được sử dụng rất nhiều trong các chương trình Python không thua kém gì List. (List đã được giới thiệu trong bài KIỂU DỮ LIỆU LIST TRONG PYTHON)

Một Tuple gồm các yếu tố sau:

- Được giới hạn bởi cặp ngoặc **()**, tất cả những gì nằm trong đó là những phần tử của Tuple.
- Các phần tử của Tuple được phân cách nhau ra bởi dấu phẩy (**,**).
- Tuple có khả năng chứa mọi giá trị, đối tượng trong Python.

Ví dụ:

```
>>> (1, 2, 3, 4, 5) # Một Tuple chứa 5 số nguyên
(1, 2, 3, 4, 5)
>>> ('k', 't', 'e', 'r') # Một Tuple chứa 4 chuỗi
('k', 't', 'e', 'r')
>>> ([1, 2], (3, 4)) # Một Tuple chứa 1 List là [1, 2] và 1 Tuple là (3, 4)
([1, 2], (3, 4))
>>> (1, 'kteam', [2, 'k9']) # Tuple chứa số nguyên, chuỗi, và List
(1, 'kteam', [2, 'k9'])
```

Cách khởi tạo Tuple

Sử dụng cặp dấu ngoặc () và đặt giá trị bên trong

Cú pháp:

> **(<giá trị thứ nhất>, <giá trị thứ hai>, .., <giá trị thứ n – 1>, <giá trị thứ n>)**

Ví dụ:

```
>>> tup = (1, 2, 3, 4)
>>> tup
(1, 2, 3, 4)
>>> empty_tup = ()  # khởi tạo tuple rỗng
>>> empty_tup
()
>>> type(tup) # kiểu dữ liệu Tuple thuộc lớp tuple
<class 'tuple'>
```

Bạn hãy chú ý khi khởi tạo tuple với một giá trị.

```
>>> tup = (9) # Tuple có một giá trị là số 9
>>> tup # có kết quả lạ
9
>>> type(tup) # không thuộc lớp Tuple
<class 'int'>
>>> str_tup = ('howkteam') # thử một trường hợp khác
>>> str_tup
'howkteam'
>>> type(tup)
<class 'str'>
```

Vì sao khi khởi tạo một Tuple với một phần tử thì kiểu kiểu dữ liệu của Tuple đó lại là kiểu dữ liệu của phần tử duy nhất đó?

- Đó là do khi bạn viết một giá trị nào đó đặt trong cặp dấu ngoặc đơn thì nó được xem là một giá trị.

Vì sao lại phải xem là một giá trị?

- Vì khi ta tính toán, hay sử dụng cặp ngoặc () để được ưu tiên.

```
>>> 1 + 3 * 2 # 3 * 2 sau đó + 1 vì nhân trước cộng sau theo như toán học
7
>>> (1 + 3) * 3 # giờ thi ta sẽ làm phép tính trong ngoặc trước
12
```

Thế nên, trường hợp đó không thể tính là một Tuple. Do đó, khi muốn khởi tạo một Tuple chỉ duy nhất một phần tử, ta phải thêm dấu `,` vào sau giá trị đó, để báo cho Python biết, đây là Tuple.

```
>>> tup = (9,)
>>> tup
(9,)
>>> type(tup) # kết quả đã như mong đợi
<class 'tuple'>
```

Sử dụng Tuple Comprehension

Với Tuple thì khái niệm Comprehension này không được áp dụng

```
>>> tup  = (value for value in range(3))
>>> tup
<generator object <genexpr> at 0x039F5D20>
```

Mà đó được coi là **Generator Expression** (Kteam sẽ giới thiệu trong tương lai).

Đối tượng được tạo từ Generator Expression cũng là một dạng iterable.

Sử dụng constructor Tuple

Cú pháp:

> **tuple(iterable)**

Công dụng: Giống hoàn toàn với việc bạn sử dụng constructor List. Khác biệt duy nhất là constructor Tuple sẽ tạo ra một Tuple.

```
>>> tup = tuple([1, 2, 3])
>>> tup
(1, 2, 3)
>>> str_tup = tuple('KTEAM')
>>> str_tup
('K', 'T', 'E', 'A', 'M')
>>> generator = (value for value in range(10) if value % 2 == 0)
>>> generator # bạn không cần phải cố gắng hiểu khi chưa rõ comprehension
<generator object <genexpr> at 0x039F5D20>
>>> tuple(generator)
(0, 2, 4, 6, 8)
>>> tuple(123)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'int' object is not iterable
```

Một số toán tử với Tuple trong Python

Các toán tử của Tuple giống với toán tử của chuỗi. Nếu bạn đọc kĩ phần này ở bài List thì bạn sẽ thấy Kteam đề cập là toán tử của List chỉ là gần giống với toán tử của chuỗi. Lí do vì sao sẽ được giải thích trong bài sự khác biệt các toán tử của **hash object** (immutable như chuỗi, Tuple) và **unhash object** (mutable như List)

Toán tử +

```
>>> tup = [1, 2]
>>> tup += ('how', 'kteam')
>>> tup
[1, 2, 'how', 'kteam']
```

Toán tử *

```
>>> tup = tuple('kter') * 3
>>> tup
('k', 't', 'e', 'r', 'k', 't', 'e', 'r', 'k', 't', 'e', 'r')
>>> (1,) * 0
()
>>> (1,) * 3
(1, 1, 1)
```

Toán tử in

```
>>> 1 in (1, 2, 3)
True
>>> 4 in ('k', 'kteam', 9)
False
```

Indexing và cắt Tuple trong Python

Indexing và cắt Tuple hoàn toàn tương tự như với kiểu dữ liệu List. (Nếu chưa biết về List bạn có thể tham khảo qua các bài về KIỂU DỮ LIỆU LIST TRONG PYTHON)

```
>>> tup = (1, 2, 'a', 'b', [3, 4])
>>> len(tup) # lấy số phần tử có trong tuple
5
>>> tup[0]
1
```

```
>>> tup[-1]
[3, 4]
>>> tup[3]
'b'
>>> tup[1:3]
(2, 'a')
>>> tup[:2]
(1, 2)
>>> tup[2:]
('a', 'b', [3, 4])
>>> tup[::-1]
([3, 4], 'b', 'a', 2, 1)
```

Thay đổi nội dung Tuple trong Python

Tuple và chuỗi đều là một dạng **hash object** (immutable). Do đó việc bạn muốn thay đổi nội dung của nó trên lí thuyết là không.

```
>>> tup = ('kter', 'howkteam', 69)
>>> tup[1]
'howkteam'
>>> tup[1] = 'changed'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

Vì sao lại nói là trên lí thuyết? Bạn sẽ biết được ngay ở phần sau.

Ma trận

Nếu bạn nắm vững khái niệm này ở List. Thì xin chúc mừng bạn vì không phải đau đầu. Nó hoàn toàn tương tự.

```
>>> tup = ((1, 2, 3), [4, 5])
>>> tup[0]
(1, 2, 3)
>>> tup[0][2]
3
>>> tup[1][-2]
4
```

Tuple có phải luôn luôn là một hash object?

Như đã định nghĩa ở bài chuỗi, một **hash object** là một đối tượng bạn không thể thay đổi nội dung của nó. Và trong phần thay đổi nội dung Tuple, bạn cũng thấy ta không thể thay đổi giá trị ở bên trong Tuple. Tuy nhiên, không phải lúc nào cũng vậy.

```
>>> tup = ([1, 2],)
>>> tup[0]
[1, 2]
>>> tup[0][0]
1
>>> tup[0][1]
2
```

Giá trị bên trong tuple đó là một List. Và, List là một **unhash object**. Suy ra, ta có thể thay đổi nội dung của nó.

```
>>> tup[0][0]
1
>>> tup[0][0] = 'howkteam'
>>> tup
(['howkteam', 2],)
```

Ta đã thay đổi nội dung của Tuple bằng một cách đó là thay đổi nội dung của một phần tử trong Tuple.

Vì thế, một Tuple sẽ được coi là một hash object khi nó chứa các phần tử đều là hash object.

Các phương thức của Tuple

Phương thức count

Cú pháp:

```
<Tuple>.count(value)
```

Công dụng: Trả về một số nguyên, chính là số lần xuất hiện của value trong Tuple.

```
>>> tup = (1, 5, 3, 5, 6, 1, 1)
>>> tup.count(1)
3
>>> tup.count(4)
0
```

Phương thức index

Cú pháp:

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

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

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

Khi nào thì chọn Tuple thay cho List?

Tuple khác List ở chỗ Tuple không cho phép bạn sửa chữa nội dung, còn List thì có. Vì đặc điểm đó, Tuple mạnh hơn List ở những điểm sau:

- **Tốc độ truy xuất** của Tuple **nhanh hơn** so với List
- **Dung lượng** chiếm trong bộ nhớ của Tuple **nhỏ hơn** so với List
- Bảo vệ dữ liệu của bạn sẽ không bị thay đổi
- Có thể dùng làm key của Dictonary (một kiểu dữ liệu sẽ được giới thiệu). Điều mà List không thể vì List là unhash object.

Những điểm trên là những điều giúp bạn có thể cân nhắc việc chọn Tuple hay List để lưu dữ dữ liệu dưới một mảng.

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 2.

1. Sẽ có lỗi **IndexError**

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: pop from empty list
```

2. Không. Vì khi đó, ta phải so sánh hai List [1, 2] và ['abc', 'def']. Mà khi so sánh hai List này một cách trực tiếp. Python sẽ phải so sánh từng phần tử của mỗi hai List đó với nhau. Nhưng một bên là số, một bên là chuỗi, nên việc so sánh trực tiếp là không được.

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

Câu hỏi củng cố

1. Tìm các cách khởi tạo List hợp lệ dưới đấy
   a. tup = tuple((1,2, 3) + [3, 4])
   b. tup = (1)
   c. tup = 1
   d. tup = 1, 2

2. Dự đoán kết quả của chương đoạn code dưới đây

```
>>> tup = (1, 2, [3, 4])
>>> tup[2] += [50, 60]
```

Lựa chọn phương án đúng

a. tup = (1, 2, [3, 4, 50, 60])
b. TypeError: 'tuple' object does not support item assignment
c. a và b đúng

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 sơ lược cho các bạn KIỂU DỮ LIỆU TUPLE TRONG PYTHON.

Ở bài sau, Kteam sẽ nói về sự khác nhau giữa toán tử ở HASH OBJECT VÀ UNHASH OBJECT.

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**".