Bài 17: KIỂU DỮ LIỆU DICT TRONG PYTHON

Xem bài học trên website để ủng hộ Kteam: Kiểu dữ liệu Dict 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 SET trong Python

Ở bài này Kteam sẽ đề cập đến các bạn **KIỂU DỮ LIỆU DICT** trong Python. Một trong những kiểu dữ liệu cực kì quan trọng 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, KIỂU DỮ LIỆU CHUỖI trong Python
- KIỂU DỮ LIỆU LIST, KIỂU DỮ LIỆU TUPLE, KIỂU DỮ LIỆU SET trong Python.

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

- Giới thiệu về Dict trong Python
- Cách khởi tạo Dict
- Lấy value trong Dict bằng key
- Thay đổi nội dung Dict trong Python
- Thêm thủ công một phần tử vào Dict

Giới thiệu về Dict trong Python

Dict(Dictionary) cũng là một container như LIST, TUPLE. Có điều khác biệt là những container như List, Tuple **có các index** để phân biệt các phần tử thì Dict dùng các **key** để phân biệt.

Chắc bạn cũng dùng từ điển tiếng Anh để tra từ vựng rồi nhỉ? Có rất nhiều từ vựng trong đó nhưng mà không từ vựng nào giống nhau. Có chăng chúng chỉ giống nhau về nghĩa? Và đó cũng như Dict(Dictionary) hoạt động trong Python

Một Dict gồm các yếu tố sau:

- Được giới hạn bởi cặp ngoặc nhọn **{}**, tất cả những gì nằm trong đó là những phần tử của Dict.
- Các phần tử của Dict được phân cách nhau ra bởi dấu phẩy (**,**).
- Các phần tử của Dict phải là một cặp **key-value**
- Cặp **key-value** của phần tử trong Dict được phân cách bởi dấu hai chấm (**:**)
- Các key buộc phải là một **hash object**

Cách khởi tạo Dict

Sử dụng cặp dấu ngoặc {} và đặt giá trị bên trong

Cú pháp:

> **{<key_1: value_1>, <key_2: value_2>, .., <key_n: value_n>}**

Ví dụ:

```
>>> dic = {'name': 'Kteam', 'member': 69}
>>> dic
{'name': 'Kteam', 'member': 69}
>>> empty_dict = {} # khởi tạo dict rỗng
>>> empty_dict
{}
>>> type(dic) # kiểu dữ liệu dict thuộc lớp 'dict'
<class 'dict'>
```

Sử dụng Dict Comprehension

Ví dụ:

```
>>> dic = {key: value for key, value in [('name', 'Kteam'), ('member', 69)]}
>>> dic
{'name': 'Kteam', 'member': 69}
```

Sử dụng constructor Dict

Với dict, ta có 4 cách để khởi tạo một Dict bằng constructor:

Khởi tạo một Dict rỗng

Cú pháp:

```
dict()
```

Ví dụ:

```
>>> dic = dict()
>>> dic
{}
```

Khởi tạo một dict từ một mapping object

Cú pháp:

```
dict(mapping)
```

Trong đó:

Mapping object cũng gần giống so với dict object.

- Một object là Mapping object khi có đủ hai phương thức **keys** và **__getitem__**.
- Dict object cũng là một mapping object. Tuy nhiên, không phải mapping object nào cũng là dict object vì dict object không chỉ có hai phương thức keys và __getitem__ và còn nhiều phương thức khác.

Bạn có thể bỏ qua ví dụ bên dưới hoặc xem để tham khảo vì phần này có thể khá khó hiểu.

Ví dụ:

```
>>> class Map_Class:
...         def keys(self):
...             return [1, 2, 3]
...         def __getitem__(self, key):
...             return key * 2
...
>>> map_obj = Map_Class()
>>> dic = dict(map_obj)
>>> dic
{1: 2, 2: 4, 3: 6}
```

Khởi tạo bằng iterable

Cú pháp:

```
dict(iterable)
```

Trong đó:

iterable này đặc biệt hơn hơn các iterable mà bạn dùng để khởi tạo List hay Tuple, đó là các phần tử trong iterable phải có 2 value đó chính là cặp **key-value**.

Bạn có thể dùng List, Tuple hoặc bất cứ container nào (trừ mapping object) để chứa cặp **key-value**.

```
>>> iter_ = [('name', 'Kteam'), ('member', 69)]
>>> dic = dict(iter_)
>>> dic
{'name': 'Kteam', 'member': 69}
```

Khởi tạo bằng keyword arguments

Cú pháp:

```
dict(**kwargs)
```

Trong đó:

Bạn chưa tìm hiểu đến hàm, nên khái niệm **keyword arguments** vẫn còn rất xa lạ!

- Cứ hiểu đơn giản là giống như việc bạn khởi tạo biến và giá trị rồi đưa cho dict đó giữ giùm.
- Một lưu ý là những biến này không bị ảnh hưởng hoặc ảnh hưởng gì đến các biến bên ngoài

Ví dụ:

```
>>> name = 'SpaceX'
>>> member = 696969
>>> dic = dict(name='Kteam', member=69)
>>> dic
{'name': 'Kteam', 'member': 69}
>>> name
'SpaceX'
>>> member
696969
```

Sử dụng Phương thức fromkeys

Cú pháp:

```
dict.fromkeys(iterable, value)
```

Công dụng: Cách này cho phép ta khởi tạo một dict với các **keys** nằm trong một iterable. Các giá trị này đều sẽ nhận được một giá trị với mặc định là None

Ví dụ:

```
>>> iter_ = ('name', 'number')
>>> dic_none = dict.fromkeys(iter_)
>>> dic_none
{'name': None, 'number': None}
>>> dic = dict.fromkeys(iter_, 'non None value')
>>> dic
{'name': 'non None value', 'number': 'non None value'}
```

Lấy value trong Dict bằng key

Cú pháp:

```
Your_dict[key]
```

Ví dụ:

```
>>> dic # ta có một dict như sau
{'name': 'Kteam', 'member': 69}
>>> dic['name']
'Kteam'
>>> dic['member']
69
>>> dic['non_exist'] # hãy chắc chắn rằng key bạn dùng có trong dict
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'non_exist'
```

Thay đổi nội dung Dict trong Python

Dict là một **unhashable object**. Do đó, chắc bạn cũng biết ta có thể thay đổi được nội dung nó hay không. Nếu bạn nào nhanh trí, chắc cũng đã biết được cách thay đổi rồi. Tương tự như List thôi!

Ví dụ:

```
>>> dic # ta có một dict như sau
{'name': 'Kteam', 'member': 69}
>>> dic['name'] = 'How Kteam'
>>> dic
{'name': 'How Kteam', 'member': 69}
>>> dic['member'] = dic['member'] + 1
>>> dic
{'name': 'How Kteam', 'member': 70}
```

Thêm thủ công một phần tử vào dict

Cách này khá giống với cách bạn thay đổi nội dung của Dict. Khác ở chỗ, bây giờ bạn sẽ sử dụng một **key** chưa hề có trong dict.

Ví dụ:

```
>>> dic # ta có một dict như sau
{'name': 'Kteam', 'member': 69}
>>> dic['slogan'] = 'Free Education'
>>> dic
{'name': 'Kteam', 'member': 69, 'slogan': 'Free Education'}
```

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 SET TRONG PYTHON.

Vì khi:

```
>>> a = {1, 2}
>>> b = a
```

Ta đã cho a và b cùng trỏ vào một chỗ. Do đó thay đổi b thì a cũng sẽ bị tác động.

Muốn giải quyết chuyện này ta nên sử dụng phương thức copy

```
>>> a = {1, 2}
>>> b = a.copy()
>>> b.clear()
>>> b
set()
>>> a
{1, 2}
```

Kết luận

Bài viết này đã giới thiệu cho các bạn cơ bản về KIỂU DỮ LIỆU DICT TRONG PYTHON.

Ở bài sau, Kteam sẽ đề cập về các phương thức của KIỂU DỮ LIỆU DICT - Phần 2

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**".