**Giới thiệu về Dict trong Python**

**Dict**(Dictionary) cũng là một container như [LIST](http://www.howkteam.vn/course/kieu-du-lieu-set-trong-python/kieu-du-lieu-list-trong-python--phan-1-1548), [TUPLE](http://www.howkteam.vn/course/kieu-du-lieu-list-trong-python--phan-1/kieu-du-lieu-tuple-trong-python-1550). 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**

Điều kiện để sử dụng dict comprehension là mỗi giá trị khi được duyệt qua phải bao gồm một giá trị tương ứng với key và một giá trị tương ứng với value.

**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**.

**Ví dụ 1:**

>>> iter\_ = [('name', 'Kteam'), ('member', 69)]

>>> dic = dict(iter\_)

>>> dic

{'name': 'Kteam', 'member': 69}

**Ví dụ 2:**

>>> f = [('ab'), ('cd')]

>>> dict(f)

{'a': 'b', 'c': 'd'}

**Lưu ý:** Nếu các bạn muốn khởi tạo dict bằng chuỗi như trên thì bắt buộc độ dài của chuỗi phải là 2. Giá trị đầu sẽ là key, giá trị thứ 2 là value.

>>> f = [('abc'), ('cde')]

>>> dict(f)

Traceback (most recent call last):

File "<stdin>", line 1, **in** <module>

ValueError: dictionary update sequence element #0 has length 3; 2 is required

**Khởi tạo bằng keyword arguments**

**Cú pháp 1:**

**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

**Cú pháp 2:**

**dict(<key\_1> = <value\_1>, <key\_2> = <value\_2>, ...)**

Khi khởi tạo dict bằng **keyword arguments** chỉ có thể hoạt động khi các key là tên các biến, còn các value là giá trị của các key tương ứng với nó (các biến có thể được khởi tạo trong cặp ngoặc **{}**).

**Ví dụ:**

>>> name = 'SpaceX'

>>> member = 696969

>>> dic = dict(name='Kteam', member=69)

>>> dic

{'name': 'Kteam', 'member': 69}

>>> name

'SpaceX'

>>> member

696969

Việc cố gắng khởi tạo dict như dưới đây là **không** hợp lệ.

>>> dict((1, 2) = 3)

File "<stdin>", line 1

dict((1, 2) = 3)

^^^^^^^^

SyntaxError: expression cannot contain assignment, perhaps you meant "=="?

>>> dict('o' = 3)

File "<stdin>", line 1

dict('o' = 3)

^^^^^

SyntaxError: expression cannot contain assignment, perhaps you meant "=="?

**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'}

**Vấn đề cần lưu tâm khi sử dụng dict**

Các bạn còn nhớ phần “**Vấn đề cần lưu tâm khi sử dụng list”** chứ ? Đối với dict, ta cũng cần có các lưu ý tương tự như vậy. Kteam sẽ đưa ra một vài ví dụ để cho các bạn hiểu. Nếu cần thiết, các bạn có thể xem lại bài “Kiểu dữ liệu list trong Python – Phần 1”.

Các lỗi dễ xảy ra với list cũng có thể xảy ra với dict

>>> a = {1: '1', 2: '2', 3: '3'}

>>> b = a

>>> a **is** b

**True**

>>> b[1] = 't'

>>> a

{1: 't', 2: '2', 3: '3'}

>>> b

{1: 't', 2: '2', 3: '3'}

>>> a[4] = '4'

>>> b

{1: 't', 2: '2', 3: '3', 4: '4'}

>>> a

{1: 't', 2: '2', 3: '3', 4: '4'}

Và tất nhiên, có lỗi thì sẽ có cách sửa chữa

>>> a = {1: '1', 2: '2', 3: '3'}

>>> b = a.copy() # Phương thức copy

>>> b **is** a

**False**

>>> b

{1: '1', 2: '2', 3: '3'}

>>> c = dict(a) # Khởi tạo dict mới

>>> c **is** a

**False**

>>> c

{1: '1', 2: '2', 3: '3'}

**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](http://www.howkteam.vn/course/kieu-du-lieu-list-trong-python--phan-1/kieu-du-lieu-set-trong-python-1566).

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}