Bài đọc thêm: Kiểu dữ liệu nâng cao trong Python

Nội dung

Tuples

Tuple là chuỗi dữ liệu không thay đổi, thường được sử dụng để lưu trữ các bộ dữ liệu không đồng nhất (chẳng hạn như 2-tuples được tạo bởi hàm được tích hợp sẵn enumerate()). Tuple cũng được sử dụng cho các trường hợp đòi hỏi một chuỗi dữ liệu thống nhất không thay đổi (chẳng hạn như cho phép lưu trữ trong một set hoặc dict).

```
class tuple([iterable])
```

Tuple có thể được tạo theo nhiều cách:

- Sử dụng một cặp dấu ngoặc đơn để khai báo tuple trống: ()
- Sử dụng dấu phẩy sau cho một tuple đơn: a, hoặc (a,)
- Tách các thành phần bằng dấu phẩy: a, b, c hoặc (a, b, c)
- Sử dụng hàm tuple () tích hợp sẵn: tuple () hoặc tuple (iterable)

Hàm khởi tạo tạo một tuple có các phần tử giống nhau và theo thứ tự như phần tử iterable. iterable có thể là một dãy, một bộ dữ liệu hỗ trợ lặp hoặc một đối tượng iterator. Nếu iterable đã là một tuple, nó sẽ trả về mà không thay đổi gì. Ví dụ, tuple ('abc') trả về ('a', 'b', 'c') và tuple ([1, 2, 3]) trả về (1, 2, 3). Nếu không có đối số được cung cấp, hàm khởi tạo tạo ra một bộ trống rỗng mới, ().

Lưu ý rằng dấu phải mới thực sự là cái tạo ra tuple chứ không phải dấu ngoặc đơn. Dấu ngoặc đơn là tùy chọn, ngoại trừ trường hợp tuple rỗng, hoặc khi chúng cần thiết để tránh sự mơ hồ của cú pháp. Ví dụ, f (a, b, c) là một lời gọi hàm với ba đối số, trong khi f ((a, b, c)) là một lời gọi hàm với một bộ ba là đối số duy nhất.

Tuple cài đặt tất cả các toán tử dùng cho chuỗi dữ liệu phổ biến.

Đối với các bộ dữ liệu không đồng nhất mà việc truy cập theo tên rõ ràng hơn truy cập theo chỉ mục (index), collections.namedtuple () có thể là một sự lựa chọn thích hợp hơn so với một đối tượng tuple đơn giản.

Các kiểu dữ liệu tập hợp (Set) — set, frozenset

Phần cơ bản

Một đối tượng set là một tập hợp không có thứ tự của các đối tượng hashable(có thể băm) phân biệt. Các ứng dụng phổ biến bao gồm kiểm tra phần tử, xóa các bản sao giống nhau từ một chuỗi, và tính toán các hàm toán học như phép giao, phép hợp, phép trừ tập hợp và phép trừ tập hợp đối xứng. (Đối với các bộ dữ liệu khác, hãy xem thêm dict, list, và tuple được cung cấp sẵn, và module collections.)

Giống như các bộ dữ liệu khác, kiểu dữ liệu Set hỗ trợ các cú pháp x in set, len (set), và for x in set. Là một bộ dữ liệu không có thứ tự, các bộ không ghi lại vị trí phần tử hoặc thứ tự chèn. Theo đó, bộ không hỗ trợ lập chỉ mục, cắt lát hoặc các hành vi giống như các chuỗi dữ liệu khác.

Hiện có hai loại set được cài đặt sẵn, set và frozenset. Kiểu dữ liệu set là có thể thay đổi - nội dung có thể được thay đổi bằng cách sử dụng các phương thức như add () và remove (). Vì nó là có thể thay đổi, nó không có giá trị băm và không thể được sử dụng như là một khóa từ điển hoặc như một phần tử của set khác. Kiểu dữ liệu frozenset là không thay đổi và hashable - nội dung của nó không thể được thay đổi sau khi nó được tạo ra; do đó nó có thể được sử dụng như là một khóa từ điển hoặc như là một phần tử của set khác.

Các tập không rỗng (không phải frozensets) có thể được tạo ra bằng cách đặt một danh sách các phần tử được cách nhau bằng dấu phẩy trong dấu ngoặc nhọn, ví dụ: `{'jack', 'sjoerd'},` bên cạnh việc khởi tạo set.

Các biến set và frozenset cung cấp các hàm sau:

`len(s)`

Trả về số phần tử trong set s (cardinality của s).

`x in s`

Kiểm tra x có nằm trong s không

`x not in s`

Kiểm tra x có không nằm trong s không

```
isdisjoint(other)
```

Trả về True nếu set không có phần tử nào giống nhau đôi một. Sets là disjoint nếu và chỉ nếu giao của chúng là một tập rỗng

```
issubset(other)
set <= other
```

Kiểm tra xem mỗi phần tử trong set có nằm trong tập other không

```
set < other
```

Kiểm tra xem set là một tập phụ bình thường của tập other không, tức là, set <= other và set != other

```
issuperset(other)
set >= other
```

Kiểm tra xem mỗi phần tử trong tập other có nằm trong tập set không

```
set > other
```

Kiểm tra xem other là một tập phụ bình thường của tập set không, tức là, set >= other và set != other.

```
union(*others)
set | other | ...
```

Trả về set mới với phần tử từ cả set và tất cả tập others

```
intersection(*others)
set & other & ...
```

Trả về set mới với phần tử là phần chung của set và tất cả tập others

```
difference(*others)
set - other - ...
```

Trả về set mới với phân tử trong set không nằm trong others

```
symmetric_difference(other)
set ^ other
```

Trả về set mới với phần tử hoặc trong set hoặc trong other, chứ không trong cả hai

```
copy()
```

Trả về set mới với một bản sao nông của s

Các hàm chỉ riêng set:

```
add(elem)
```

Thêm phần tử elem vào set

```
remove(elem)
```

Loại phần tử elem khỏi set. Đưa ra ngoại lệ KeyError nếu elem không chứa trong set

```
discard(elem)
```

Loại phần tử elem khỏi set nếu nó tồn tại

```
pop()
```

Loại bỏ và trả về phần tử tùy tiện của set. Đưa ra ngoại lệ KeyError nếu set rỗng

```
clear()
```

Xóa tất cả phần tử của set

Cả set và frozenset hỗ trợ so sánh set và set. Hai bộ bằng nhau khi và chỉ khi mọi phần tử của mỗi tập được chứa trong tập kia (mỗi tập là tập con của tập kia). Một bộ nhỏ hơn một bộ khác nếu và chỉ khi tập đầu tiên là một tập con thích hợp của bộ thứ hai (là tập con, nhưng không bằng). Một tập hợp lớn hơn một tập khác khi và chỉ khi tập đầu tiên là tập hợp đúng của tập thứ hai (là tập lớn hơn, nhưng không bằng).

Các biến set được so sánh với biến frozenset dựa trên thành phần của chúng. Ví dụ, `set ('abc') == frozenset ('abc')` trả về True và tương tự như `set('abc') in set ([frozenset ('abc')])`.

Phần nâng cao

Hàm khởi tạo cho cả hai lớp làm việc giống nhau:

```
class set([iterable])
```

```
class frozenset([iterable])
```

Trả về một đối tượng set or frozenset mới có các phần tử được lấy từ iterable. Các phần tử của một tập hợp phải có khả năng băm. Để đại diện cho set của set, các set bên trong phải là đối tượng frozenset. Nếu iterable không được chỉ định, một tập rỗng mới sẽ được trả về.

Lưu ý rằng, các phiên bản không phải toán tử của `union(), intersection (), difference (),` và `symmetric_difference (), issubset (),` và `issuperset ()` sẽ chấp nhận bất kỳ iterable như một đối số. Ngược lại, các thành phần dựa trên toán tử của chúng yêu cầu đối số phải là một set. Điều này ngăn cản các cấu trúc dễ bị lỗi như `set ('abc') & 'cbs'` nhằm ủng hộ cách ghi chú dễ đọc hơn `set('abc').intersection('cbs').`

Vì các bộ chỉ xác định thứ tự từng phần (các mối quan hệ subset), đầu ra của phương thức `list.sort ()` không được xác định cho list các set.

Các phần tử của set, như các khóa từ điển, phải hashable

Các phép toán nhị phân trộn các biến set với frozenset trả về kiểu toán hạng đầu tiên.

Ví dụ: `frozenset ('ab') | set ('bc')` trả về một thể hiện của frozenset.

Bảng dưới đây liệt kê các hàm có sẵn cho set mà không áp dụng cho các trường hợp không thay đổi của frozenset:

```
update(*others)
```

```
set |= other | ...
```

Cập nhật set, thêm phần tử từ các set khác

```
intersection_update(*others)
set &= other & ...
```

Cập nhật set, giữ chỉ các phần tử tìm thấy trong nó và tất cả set others

```
difference_update(*others)
set -= other | ...
```

Cập nhật set, loại bỏ các phần tử trong các set others

```
symmetric_difference_update(other)
set ^= other
```

Cập nhật set, hợp set và other và loại bỏ các phần tử chung

Lưu ý, các phiên bản không phải toán tử của các phương thức update (), intersection_update (), difference_update () và symmetric_difference_update () sẽ chấp nhận bất kỳ iterable nào như một đối số.

Chú ý, đối số elem đối với các phương thức __contains__ (), remove () và discard () có thể là set. Để hỗ trợ cho việc tìm kiếm một frozenset tương đương, một biến tạm thời được tạo ra từ elem.

Kiểu dữ liệu Mapping — dict

Phần cơ bản

Đối tượng Mapping ánh xạ giá trị hashable (có thể băm) tới các đối tượng tùy ý. Mapping là các đối tượng có thể thay đổi. Hiện tại chỉ có một loại Mapping chuẩn - dictionary (từ điển). (Đối với các container khác, xem list, set, và tuple được cung cấp sẵn và module collections.)

Một phần tử trong mapping bao gồm một cặp khóa/giá trị (key/value). Một khóa (key) của từ điển gần như tùy ý. Các giá trị (value) không phải là giá trị hashable, nghĩa là các giá trị chứa lists, dictionaries hoặc các kiểu có thể thay đổi khác (được so sánh theo giá trị hơn là theo đối tượng) không được sử dụng làm khóa. Các kiểu số được sử dụng cho các khóa tuân theo các quy tắc bình thường để so sánh số: nếu hai số so sánh bằng nhau (như 1 và 1.0) thì chúng có thể được sử dụng hoán đổi cho cùng một mục trong từ điển. (Tuy nhiên, vì máy tính lưu các số dấu chấm động như xấp xỉ, thường là không khôn ngoan để sử dụng chúng làm các khóa từ điển.)

Từ điển có thể được tạo ra bằng cách đặt một danh sách các cặp khóa chính: cặp giá trị trong dấu ngoặc đơn, ví dụ:

`{'jack': 4098, 'sjoerd': 4127}` hoặc `{4098: 'jack', 4127: 'sjoerd'}`

Đây là một số phương thức mà từ điển hỗ trợ (các loại dữ liệu mapping tùy chỉnh cũng nên hỗ trợ):

```
len(d)
```

Trả về số phần tử trong từ điển d

```
d[key]
```

Trả về phần tử của d với khóa key. Đưa ngoại lệ KeyError nếu key không có trong d

```
d[key] = value
```

Gán giá trị d[key] là value

```
del d[key]
```

Xóa d[key] khỏi d. Đưa ngoại lệ KeyError nếu khóa không có trong map

```
key in d
```

Trả về True nếu d có khóa key, ngược lại False

```
key not in d
```

Tương đương với not key in d

```
clear()
```

Xóa tất cả phần tử khởi từ điển

```
copy()
```

Trả về bản sao nông của từ điển

Phần nâng cao

Các hàm khởi tạo dict:

```
class dict(**kwarg)
class dict(mapping, **kwarg)
class dict(iterable, **kwarg)
```

Trả về một từ điển mới được khởi tạo từ một tham số vị trí (positional argument) tùy chọn và một tập các tham số khóa có thể có.

Nếu không có tham số vị trí được đưa ra, một từ điển rỗng sẽ được tạo ra. Nếu một tham số vị trí được đưa ra và nó là một đối tượng mapping, một từ điển được tạo ra với cùng một cặp khóa-giá trị như là đối tượng mapping. Nếu không, tham số vị trí phải là đối tượng iterable. Mỗi mục trong iterable tự nó phải là một iterable với chính xác hai đối tượng. Đối tượng đầu tiên của mỗi phần tử trở thành một từ khóa trong từ điển mới, và đối tượng thứ hai là giá trị tương ứng. Nếu khóa được gán nhiều lần, giá trị cuối cùng của khóa đó trở thành giá trị tương ứng trong từ điển mới.

Nếu tham số khoá được đưa ra, tham số khóa và giá trị của chúng được thêm vào từ điển được tạo từ tham số vị trí. Nếu khoá được thêm vào đã có, giá trị từ tham số từ khoá sẽ thay thế giá trị từ tham số vị trí.

Để minh họa, các ví dụ sau đây tất cả đều trả lại một từ điển bằng `{"one": 1, "two": 2, "three": 3}`:

```
>>>
>>> a = dict(one=1, two=2, three=3)
>>> b = {'one': 1, 'two': 2, 'three': 3}
>>> c = dict(zip(['one', 'two', 'three'], [1, 2, 3]))
>>> d = dict([('two', 2), ('one', 1), ('three', 3)])
>>> e = dict({'three': 3, 'one': 1, 'two': 2})
>>> a == b == c == d == e
True
```

Việc cung cấp các đối số từ khóa như trong ví dụ đầu tiên chỉ hoạt động cho các khóa là các số nhận dạng Python hợp lệ. Nếu không, bất kỳ khóa hợp lệ nào đều có thể được sử dụng.

Nếu một phân lớp của dict định nghĩa một phương thức __missing __ () và không có khóa, thì thao tác d [key] gọi phương thức đó bằng khoá key là đối số. Các d [key] hoạt động sau đó trả về hoặc đưa ngoại lệ bất cứ điều gì được trả lại hoặc đưa ra bởi cách gọi hàm __missing __ (key). Không có phương thức khác gọi __missing __ (). Nếu __missing __ () không được định nghĩa, KeyError được đưa ra. __missing __ () phải là một phương thức; nó không thể là một biến:

```
>>>
>>> class Counter(dict):
...     def __missing__(self, key):
...         return 0
>>> c = Counter()
>>> c['red']
0
>>> c['red'] += 1
>>> c['red']
1
```

Ví dụ trên đây cho thấy cách cài đặt collections Counter. Một phương thức __missing__ khác được sử dụng bởi collections.

Các hàm hỗ trợ từ điển cung cấp:

```
iter(d)
```

Trả về một iterator qua khóa của từ điển. Đây là viết tắt của iter(d.keys())

```
classmethod fromkeys(seq[, value])
```

Tạo một từ điển mới với khóa từ seq và tập giá trị value

fromkeys() là một phương thức lớp trả về một từ điển mới. Giá trị mặc định là None.

```
get(key[, default])
```

Trả về giá trị của khóa nếu khóa trong từ điển nếu không trả về mặc định. Nếu mặc định không được cung cấp, mặc định là None, để cho phương thức này không bao giờ đưa ngoại lệ

`items()`

Trả về một view mới của phần tử trong từ điển (các cặp(key, value)). Xem tài liệu của đối tượng view để biết thêm chi tiết.

`keys()`

Trả về một view mới của khóa từ điển. Xem tài liệu của đối tượng view để biết thêm chi tiết.

`pop(key[, default])`

Nếu key nằm trong từ điển, xóa nó và trả về giá trị của nó, nếu không trả về mặc định. Nếu mặc định không được cung cấp và khóa không nằm trong từ điển, ngoại lệ KeyError được đưa ra

`popitem()`

Xóa và trả về một cặp (key,value) tùy ý từ từ điển

popitem() hữu ích để duyệt và phá xuyên qua từ điển, như thường sử dụng trong thuật toán set. Nếu từ điển trống gọi popitem() đưa ra ngoại lệ KeyError.

`setdefault(key[, default])`

Nếu khóa nằm trong từ điển, trả về giá trị của nó. Nếu không chèn khóa với giá trị mặc định và trả về giá trị mặc định. Giá trị mặc định mặc định là None

`update([other])`

Cập nhật từ điển với cặp key/value từ other, ghi đè khóa đã tồn tại. Trả về None.

update() chấp nhận hoặc một đối tượng từ điển khác hoặc một cặp key/value iterable (như tuples hoặc iterable khác với độ dài là hai). Nếu tham số khóa được đề cập, từ điển được cập nhật với các cặp key/value đó: d.update(red=1, blue=2)

`values()`

Trả về một view mới của giá trị từ điển. Xem tài liệu của đối tượng view để biết thêm chi tiết.

Từ điển so sánh bằng khi và chỉ khi chúng có chung cặp (key,value). So sánh có thứ tự (‘<’, ‘<=’, ‘>=’, ‘>’) đưa ra ngoại lệ TypeError.