Python - Remover itens definidos

A classe set do Python fornece diferentes métodos para remover um ou mais itens de um objeto set.

Método remove()

O método remove() remove o item fornecido da coleção de conjuntos, se estiver presente nela. No entanto, se não estiver presente, gera KeyError.

Sintaxe

```
set.remove(obj)
```

Parâmetros

- **obj** - um objeto imutável

Exemplo

```
lang1 = {"C", "C++", "Java", "Python"}
print ("Set before removing: ", lang1)
lang1.remove("Java")
print ("Set after removing: ", lang1)
lang1.remove("PHP")
```

Ele produzirá a seguinte **saída** -

```
Set before removing: {'C', 'C++', 'Python', 'Java'}
Set after removing: {'C', 'C++', 'Python'}
  lang1.remove("PHP")
KeyError: 'PHP'
```

método descartar()

O método descartar() na classe set é semelhante ao método remove(). A única diferen que não gera erro mesmo que o objeto a ser removido ainda não esteja presente na co definida.

Sintaxe

```
set.discard(obj)
```

Parâmetros

- **obj** - Um objeto imutável

Exemplo

```
lang1 = {"C", "C++", "Java", "Python"}
print ("Set before discarding C++: ", lang1)
lang1.discard("C++")
print ("Set after discarding C++: ", lang1)
print ("Set before discarding PHP: ", lang1)
lang1.discard("PHP")
print ("Set after discarding PHP: ", lang1)
```

Ele produzirá a seguinte **saída** -

```
Set before discarding C++: {'Java', 'C++', 'Python', 'C'}
Set after discarding C++: {'Java', 'Python', 'C'}
Set before discarding PHP: {'Java', 'Python', 'C'}
Set after discarding PHP: {'Java', 'Python', 'C'}
```

Método pop()

O método pop() na classe set remove um item arbitrário da coleção set. O item removido é retornado pelo método. Sair de um conjunto vazio resulta em KeyError.

Sintaxe

```
obj = set.pop()
```

Valor de retorno

O método pop() retorna o objeto removido do conjunto.

Exemplo

```
lang1 = {"C", "C++"}
print ("Set before popping: ", lang1)
obj = lang1.pop()
print ("object popped: ", obj)
print ("Set after popping: ", lang1)
obj = lang1.pop()
obj = lang1.pop()
```

Ele produzirá a seguinte **saída** -

```
Set before popping: {'C++', 'C'}
object popped: C++
Set after popping: {'C'}
Traceback (most recent call last):
  obj = lang1.pop()
      ^^^^^^^^^^^^
KeyError: 'pop from an empty set'
```

No momento da chamada para pop() pela terceira vez, o conjunto está vazio, portanto KeyError é gerado.

Método claro()

O método clear() na classe set remove todos os itens em um objeto set, deixando um conjunto vazio.

Sintaxe

```
set.clear()
```

Exemplo

```
lang1 = {"C", "C++", "Java", "Python"}
print (lang1)
print ("After clear() method")
lang1.clear()
print (lang1)
```

Ele produzirá a seguinte **saída** -

```
{'Java', 'C++', 'Python', 'C'}
After clear() method
```

```
set()
```

Método diferença_update()

O método Difference_update() na classe set atualiza o conjunto removendo itens que são comuns entre ele e outro conjunto dado como argumento.

Sintaxe

```
set.difference_update(obj)
```

Parâmetros

- **obj** - um objeto definido

Exemplo

```
s1 = {1,2,3,4,5}
s2 = {4,5,6,7,8}
print ("s1 before running difference_update: ", s1)
s1.difference_update(s2)
print ("s1 after running difference_update: ", s1)
```

Ele produzirá a seguinte **saída** -

```
s1 before running difference_update: {1, 2, 3, 4, 5}
s1 after running difference_update: {1, 2, 3}
set()
```

Método diferença()

O método Difference() é semelhante ao método Difference_update(), exceto que retorna um novo objeto de conjunto que contém a diferença dos dois conjuntos existentes.

Sintaxe

```
set.difference(obj)
```

Parâmetros

- **obj** - um objeto definido

Valor de retorno

O método Difference() retorna um novo conjunto com itens restantes após a remoção daqueles em obj.

Exemplo

```
s1 = {1,2,3,4,5}
s2 = {4,5,6,7,8}
print ("s1: ", s1, "s2: ", s2)
s3 = s1.difference(s2)
print ("s3 = s1-s2: ", s3)
```

Ele produzirá a seguinte **saída** -

```
s1: {1, 2, 3, 4, 5} s2: {4, 5, 6, 7, 8}
s3 = s1-s2: {1, 2, 3}
```

Método intersecção_update()

Como resultado do método cross_update(), o objeto definido retém apenas os itens que são comuns em si e em outro objeto definido fornecido como argumento.

Sintaxe

```
set.intersection_update(obj)
```

Parâmetros

- **obj** - um objeto definido

Valor de retorno

O método cross_update() remove itens incomuns e mantém apenas os itens que são comuns a ele e ao objeto.

Exemplo

```
s1 = {1,2,3,4,5}
s2 = {4,5,6,7,8}
print ("s1: ", s1, "s2: ", s2)
s1.intersection_update(s2)
print ("a1 after intersection: ", s1)
```

Ele produzirá a seguinte **saída** -

```
s1: {1, 2, 3, 4, 5} s2: {4, 5, 6, 7, 8}
s1 after intersection: {4, 5}
```

Método interseção()

O método interseção() na classe set é semelhante ao método intersecção_update(), exceto que retorna um novo objeto de conjunto que consiste em itens comuns aos conjuntos existentes.

Sintaxe

```
set.intersection(obj)
```

Parâmetros

- **obj** - um objeto definido

Valor de retorno

O método interseção() retorna um objeto definido, retendo apenas os itens comuns a ele e ao objeto.

Exemplo

```
s1 = {1,2,3,4,5}
s2 = {4,5,6,7,8}
print ("s1: ", s1, "s2: ", s2)
s3 = s1.intersection(s2)
print ("s3 = s1 & s2: ", s3)
```

Ele produzirá a seguinte **saída** -

```
s1: {1, 2, 3, 4, 5} s2: {4, 5, 6, 7, 8}
s3 = s1 & s2: {4, 5}
```

Método symmetric_difference_update()

A diferença simétrica entre dois conjuntos é a coleção de todos os itens incomuns, rejeitando os elementos comuns. O método symmetric_difference_update() atualiza um conjunto com diferença simétrica entre ele e o conjunto dado como argumento.

Sintaxe

```
set.symmetric_difference_update(obj)
```

Parâmetros

- **obj** - um objeto definido

Exemplo

```
s1 = {1,2,3,4,5}
s2 = {4,5,6,7,8}
print ("s1: ", s1, "s2: ", s2)
s1.symmetric_difference_update(s2)
print ("s1 after running symmetric difference ", s1)
```

Ele produzirá a seguinte **saída** -

```
s1: {1, 2, 3, 4, 5} s2: {4, 5, 6, 7, 8}
s1 after running symmetric difference {1, 2, 3, 6, 7, 8}
```

Método symmetric_difference()

O método symmetric_difference() na classe set é semelhante ao método symmetric_difference_update(), exceto que ele retorna um novo objeto set que contém todos os itens de dois conjuntos menos os itens comuns.

Sintaxe

```
set.symmetric_difference(obj)
```

Parâmetros

- **obj** - um objeto definido

Valor de retorno

O método symmetric_difference() retorna um novo conjunto que contém apenas os itens que não são comuns entre os dois objetos do conjunto.

Exemplo

```
s1 = {1,2,3,4,5}
s2 = {4,5,6,7,8}
print ("s1: ", s1, "s2: ", s2)
s3 = s1.symmetric_difference(s2)
print ("s1 = s1^s2 ", s3)
```

Ele produzirá a seguinte **saída** -

```
s1: {1, 2, 3, 4, 5} s2: {4, 5, 6, 7, 8}
s1 = s1^s2 {1, 2, 3, 6, 7, 8}
```