Python - Ensembles (set)

[Mise à jour le : 30/6/2021]

- **Sources**
  - **Documentation** sur Python.org : référence du langage, ensembles, fonctions natives (built-in)
- **Mots-clés** : ensemble, mutable, immuable, union, intersection, différence, différence symétrique, comparaison, égalité, inclusion, disjoint.

Les mots ci-dessous sont dits “réservés”. Ils ne peuvent pas être utilisés comme nom de variable. Les mots soulignés sont une nouveauté de Python 3. Les mots en **gras** sont utilisés dans cette page.

and	continue	finally	is	raise
as	def	for	lambda	return
assert	del	from	None	True
async	elif	global	nonlocal	try
await	else	if	not	while
break	except	import	or	with
class	False	in	pass	yield

- Fonctions natives **(built-in)**[1] utilisées dans les exemples : **frozenset**(), **len**() **print**(), **set**().

---

1. Introduction

Proche des dictionnaires, les sets ne stockent qu'une clé. Ils sont très utilisés pour **garder le nombre d'éléments uniques d'une séquence** ou faire le **test d'appartenance sur les éléments d'une séquence**. Un set peut stocker n'importe quel élément hashable (entier,caractère, booléen etc.).

Les ensembles ne sont pas ordonnés !

2. Création

2.1 Création en extension

On crée un ensemble avec les accolades, comme les dictionnaires, mais sans utiliser le caractère ":" ou avec la fonction **set**().

Exemple 1 : avec des accolades

*.py

```
# Un set sur une liste élimine les doublons
ensemble = {'marc', 12, 'pierre', (1, 2, 3), 'pierre'}
print(ensemble) # Résultat : {'marc', 'pierre', 12, (1, 2, 3)}
```

Exemple 2 : avec la fonction set()

*.py

```
# Un set sur une liste élimine les doublons
ensemble = set(['marc', 12, 'pierre', (1, 2, 3), 'pierre'])
print(ensemble) # Résultat : {'marc', 'pierre', 12, (1, 2, 3)}
```

2.1 Création d'un ensemble vide

Un ensemble vide se crée seulement avec la fonction **set**().

Exemple

*.py

```
ensemble_vide = set()
# ou
ensemble_vide = set([])
```

3. Eléments dans un ensemble

Les **éléments** d'un ensemble doivent être globalement **immuables**. Le type set étant **mutable**, on ne peut **pas** créer un **ensemble d'ensembles**.

4. Fonction native Frozenset

Source set, frozenset

Un frozenset est un ensemble qu'on ne peut pas modifier, et qui donc peut servir de clé dans un dictionnaire, ou être inclus dans un autre ensemble (mutable ou pas).

5. Opérations

5.1 Test d'appartenance

*.py

```
a = {'marc', 'pierre', 12, (1, 2, 3)}
'pierre' in a # Résultat :True
6 in a # Résultat : False
```

Lors d'un test d'appartenance sur une séquence, convertir la séquence en set accélère le traitement.

5.2 Cardinal

- **Nombre d'éléments dans un ensemble**

*.py

```
a = {'marc', 'pierre', 12, (1, 2, 3)}
len(a) # Résultat : 4
```

5.3 Manipulations

- **Suppression des doublons dans une liste**

*.py

```
a = {['marc', 'pierre', 12, (1, 2, 3), 12, 'marc']}
print(a)  # Résultat : {(1, 2, 3), 12, 'marc', 'pierre'}
```

- **Vidage**

*.py

```
# pour nettoyer
a = {'marc', 'pierre', 12, (1, 2, 3)}
a.clear() # Résultat : set()
```

- **Ajout d'un élément**

*.py

```
# ajouter un élément
a = {'marc', 'pierre', 12, (1, 2, 3)}
a.add(1) # Résultat : {1, (1, 2, 3), 12, 'marc', 'pierre'}
```

- **Mise à jour d'un ensemble avec un autre ensemble**

*.py

```
# ajouter tous les éléments d'un autre ensemble
a = {'marc', 'pierre', 12, (1, 2, 3)}
a.update({2, (1, 2, 3), (1, 3, 5)}) # Résultat : {2, (1, 3, 5),
'pierre', (1, 2, 3), 12, 'marc'}
```

- **Suppression d'un élément**

*.py

```
# enlever un élément avec discard
a = {2, (1, 3, 5), 'pierre', (1, 2, 3), 12, 'marc'}
a.discard((1, 3, 5)) # Résultat : {2, 'pierre', (1, 2, 3), 12, 'marc'}
```

discard fonctionne même si l'élément n'est pas présent

*.py

```
# enlever un élément avec remove
a = {2, (1, 3, 5), 'pierre', (1, 2, 3), 12, 'marc'}
a.remove('jean') # Résultat : exception
```

Contrairement à discard, l'élément doit être présent, sinon il se produit une exception.

*.py

```
# pop() ressemble à la méthode éponyme sur les listes
# sauf qu'il n'y a pas d'ordre dans un ensemble
a = {2, (1, 3, 5), 'pierre', (1, 2, 3), 12, 'marc'}
while a:
    a = a.pop()
    print("element", a)
print("l'ensemble est vide", a)

# Résultat
# element 2
# element (1, 3, 5)
# element pierre
# element (1, 2, 3)
# element 12
# element marc
# l'ensemble est vide set()
```

5.4 Opérations sur les ensembles

Les notations des opérateurs sur les ensembles rappellent les opérateurs “**bit-à-bit**” sur les entiers.
Ces opérateurs sont également disponibles sous la forme de méthodes.

- **Union**

*.py

```
A1 = set([0, 2, 4, 6])
A2 = set([0, 6, 3])
A1 | A2 # Résultat : {0, 2, 3, 4, 6}
```

- **Intersection**

*.py

```
A1 = set([0, 2, 4, 6])
A2 = set([0, 6, 3])
A1 & A2 # Résultat : {0, 6}
```

- **Différence**

*.py

```
A1 = set([0, 2, 4, 6])
```

```
A2 = set([0, 6, 3])
A1 - A2 # Résultat : {2, 4}
```

- **Différence symétrique**

□Δ□=(□−□) ∪ (□−□)

*.py

```
A1 = set([0, 2, 4, 6])
A2 = set([0, 6, 3])
A1 ^ A2 # Résultat : {2, 3, 4}
```

5.5 Comparaisons

- **Egalité**

*.py

```
a = {'marc', 'pierre', 12, (1, 2, 3)}
b = set(['pierre', 12, (1, 2, 3), 'marc'])
a==b # Résultat : True
```

- **Inclusion**

*.py

```
a = {0, 1, 2, 3}
b =  {1, 3}
b <= a # Résultat : True
```

*.py

```
a = {0, 1, 2, 3}
b =  {1, 3}
b < a # Résultat : True
```

- **Ensembles disjoints**

*.py

```
a = {0, 1, 2, 3}
```

```
b =  {'marc', 'pierre', 12, (1, 2, 3)}
a.isdisjoint(b) # Résultat : True
```

Quiz

- **Real Python** : Python Sets Quiz

[1)]
Fonctions toujours disponibles.