Session 7 - Les ensembles - A.U. 23-24

November 29, 2023

1 Ensemble: Définition et propriétés

[1]:
```
# Un ensemble en python est une collection d'éléments (données) uniques␣
 ↪(immutables) et non ordonnés.
e = {0, 10, 1, 5.5, "bjr", True, False}
len(e)  # ---> 7
type(e)  # ---> <class 'set'>
set("fsdfgsdfgsdf")  # ---> {'d', 'f', 'g', 's'}
set( ["0:0", "1:0", "1:0", "1:0", "1:0", "1:0", "1:0", "1:0", "1:0", "0:0"] ) ␣
 ↪# ---> {'0:0', '1:0'}
# set((10)) # ---> TypeError: 'int' object is not iterable
```

[1]:
```
{'0:0', '1:0'}
```

[2]:
```
(1, 2)
(1, 2, 3)  # Les parentheses sont utilisées pour definir un tuple
( 1 + 2 )  # Operation sur les ensembleses parentheses sont utilisées pour␣
 ↪grouper des expressions et non pour definir un tuple
# (ax+b+(c+d),) # les parentheses sont utilisées contenant un seul element et␣
 ↪une virgule pour definir un tuple
(1 + 2) == 3  # Les parentheses sont utilisées pour grouper des expressions #␣
 ↪---> True
(1,)  # C'est un tuple ( contenant un seul element et une virgule pour definir␣
 ↪un tuple)
(1)  # C'est un entier
```

[2]:
```
1
```

[3]:
```
set((10,))  # ---> {10}
set("1")  # ---> {'1'}
```

[3]:
```
{'1'}
```

[4]:
```
# Les ensembles sont mutables
e  # ---> {0, 1, 5.5, 10, False, True, 'bjr'}
```

```
e.add(100)  # la methode .add() permet d'ajouter un element dans un ensemble # ---> None
e.union( "gsfgsdfg" )  # la methode .union() permet d'ajouter les elements d'un iterable dans e # ---> None
```

[4]:
```
{0, 1, 10, 100, 5.5, 'bjr', 'd', 'f', 'g', 's'}
```

2 Element par rapport à un ensemble

[5]:
```
20 in e  # inclusion # ---> False
0 in e  # inclusion # ---> True
20 not in e  # exclusion # ---> True
not (20 in e)  # exclusion # ---> True
```

[5]:
```
True
```

3 Parcourir un ensemble

[6]:
```
for k3iba in e:
    print(k3iba)
```

```
0
1
100
5.5
10
bjr
```

4 Convertion ensemble <=> liste/tuple/str

[7]:
```
# la fonction set(I) prend un iterable en parametre et retourne un ensemble contenant les elements de I. C'est un outil pour supprimer les doublons d'un iterable
set("12323211")  # ---> {1, 2, 3}
set({1, 2, 3, 2, 2, 2})  # ---> {1, 2, 3}
set([1, 2, 3, 2, 2, 2])  # ---> {1, 2, 3}
set((1, 2, 3, 2, 2, 2))  # ---> {1, 2, 3}
set({1: "a", 2: "b", 3: "c"})  # ---> {1, 2, 3}

e  # ---> {0, 1, 5.5, 10, False, True, 'bjr'}
```

```
str(e)  # ---> "{0, 1, 5.5, 10, False, True, 'bjr'}"
list(e)  # ---> [0, 1, 5.5, 10, False, True, 'bjr']
tuple(e)  # ---> (0, 1, 5.5, 10, False, True, 'bjr')
```

[7]:
```
(0, 1, 100, 5.5, 10, 'bjr')
```

[8]:
```
s1 = set(range(8))  # ---> {0, 1, 2, 3, 4, 5, 6, 7}
```

[9]:
```
s2 = set(range(3, 10))  # ---> {3, 4, 5, 6, 7, 8, 9}
s2
```

[9]:
```
{3, 4, 5, 6, 7, 8, 9}
```

5 Ensemble par rapport à un ensemble

[10]:
```
s1 < s2  # < designe l'inclusion stricte # ---> True
s1 < s1  # < designe l'inclusion stricte # ---> False
s1 <= s1  # <= designe l'inclusion large # ---> True

s1 > s2
s1 >= s2
```

[10]:
```
False
```

[11]:
```
# La reunion de deux ensembles
# l'operateur | permet de faire l'union de deux ensembles #---> {0, 1, 2, 3, 4,␣
 ↪5, 6, 7, 8, 9}
s1 | s2
s1 | set("sdfsdfsdfsd")  # ---> {0, 1, 2, 3, 4, 5, 6, 7, 'd', 'f', 's'}
s1.union( "sdfsdfsdfsd" )  # la methode .union(I) prend un iterable en␣
 ↪parametre et retourne un ensemble contenant les elements de s1 et I #--->␣
 ↪{0, 1, 2, 3, 4, 5, 6, 7, 'd', 'f', 's'}

# La reunion de deux ensembles avec mise à jour
s1 = s1 | set("sdfsdfsdfsd")  # ---> {0, 1, 2, 3, 4, 5, 6, 7, 'd', 'f', 's'}
# ou bien
s1 = s1.union("sdfsdfsdfsd")
# ou bien
s1.update("sdfsdfsdfsd")
s1
```

[11]:
```
{0, 1, 2, 3, 4, 5, 6, 7, 'd', 'f', 's'}
```

[12]:
```
# Des element separes par des virgules form un tuple meme en absence des␣
 ↪parentheses #(s1,s2)
```

```
s1, s2
1, 2, 3  # ---> (1, 2, 3)
x, y, z = 1, 2, 3  # ---> x=1, y=2, z=3
```

[13]:
```
# L'intersection de deux ensembles
s1 & s2
# ou bien
s1.intersection( "sdfsdfsdfsd" )  # La methode .intersection(I) prend un␣
 ↪iterable en parametre et retourne un ensemble contenant les elements communs␣
 ↪entre s1 et I #---> {0, 1, 2, 3, 4, 5, 6, 7, 'd', 'f', 's'}
# Ou bien
s1 & set("sdfsdfsdfsd")

# L;intersection de deux ensembles avec mise à jour
s1 = s1.intersection("sdfsdfsdfsd")
# Ou bien
s1 = s1 & set("sdfsdfsdfsd")
# Ou bien
s1.intersection_update( "sdfsdfsdfsd" )  # La methode .intersection_update(I)␣
 ↪prend un iterable en parametre et met à jour s1 avec les elements communs␣
 ↪entre s1 et I #---> None
```

[14]:
```
# La difference de deux ensembles
s1 - s2
# ou bien
s1.difference( "dfsdfsdfsdfsdgdsfdsf" )  # La methode .difference(I) prend un␣
 ↪iterable en parametre et retourne un ensemble contenant les elements de s1␣
 ↪qui ne sont pas dans I #---> {0, 1, 2, 3, 4, 5, 6, 7, 'd', 'f', 's'}
# ou bien
s1 - set("dfsdfsdfsdfsdgdsfdsf")

# La difference de deux ensembles avec mise à jour
s1 = s1.difference("dfsdfsdfsdfsdgdsfdsf")
# ou bien
s1 = s1 - set("dfsdfsdfsdfsdgdsfdsf")
# Ou bien
s1.difference_update( "dfsdfsdfsdfsdgdsfdsf" )  # La methode .
 ↪difference_update(I) prend un iterable en parametre et met à jour s1 avec␣
 ↪les elements de s1 qui ne sont pas dans I #---> None
```

[15]:
```
# La difference symetrique de deux ensembles
s1 ^ s2
# Ou bien
s1.symmetric_difference( "adfasdfdf" )  # La methode .symmetric_difference(I)␣
 ↪prend un iterable en parametre et retourne un ensemble contenant les␣
 ↪elements de s1 qui ne sont pas dans I et les elements de I qui ne sont pas␣
 ↪dans s1 #---> {0, 1, 2, 3, 4, 5, 6, 7, 'd', 'f', 's'}
```

```
# ou bien
s1 ^ set("adfasdfdf")

# La difference symetrique de deux ensembles avec mise à jour
s1 = s1.symmetric_difference("adfasdfdf")
# ou bien
s1 = s1 ^ set("adfasdfdf")
# ou bien
s1.symmetric_difference_update( s2 )  # La methode .
 ↪symmetric_difference_update(I) prend un iterable en parametre et met à jour␣
 ↪s1 avec les elements de s1 qui ne sont pas dans I et les elements de I qui␣
 ↪ne sont pas dans s1 #---> None
```

6 Operation sur les ensembles

[16]:
```
s1
if 9 in s1:
    s1.remove(9)
s1
```

[16]: {3, 4, 5, 6, 7, 8}

[17]:
```
l = len(s2)
for i in range(l):
    print(i, "eme 9alb")
    s2.pop()
s2
```

```
0 eme 9alb
1 eme 9alb
2 eme 9alb
3 eme 9alb
4 eme 9alb
5 eme 9alb
6 eme 9alb
```

[17]: set()

[18]:
```
e1 = e
e.add(9999)
e
```

[18]: {0, 1, 10, 100, 5.5, 9999, 'bjr'}

[19]:
```
e1.pop()
e
```

[19]:
```
{1, 10, 100, 5.5, 9999, 'bjr'}
```

[20]:
```
set()
{}
s = set()
s.add(1)
s.add("bjr")
s.add((2, 4))
s
```

[20]:
```
{(2, 4), 1, 'bjr'}
```

[ ]:
```
# Les ensembles ne peuvent pas contenir des elements mutables. Ces operations␣
↪ne sont pas autorisees:
s.add([20, 40])
s.add({20, 40})
s.add({"a": 20, "b": 40})
```