Structures de données: Tuple

```
>>> t = (2,3,5)
>>> type(t)
<class 'tuple'>
>>> t = tuple("chien")
>>> t
('c', 'h', 'i', 'e', 'n')
>>> t = tuple( range( 5 ) )
>>> t
(0, 1, 2, 3, 4)
>>> len(t)
5
>>> tuple( 5 )
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'int' object is not iterable
```

Structures de données: Tuple

```
>>> a = 3
>>> b = 'canard'
>>> t = (a,b)
>>> type(t)
<class 'tuple'>
>>> (c,d) = t
>>> c
3
>>> d
'canard'
>>> t = (1,2)
>>> t += (3,)
(1, 2, 3)
```

Structures de données: Tuple

```
>>> (b,a) = (a,b)
>>> (a,b)
('canard', 3)
>>> (a,b,c) = (a,b)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: not enough values to unpack (expected 3, got 2)
>>> (a,b) = (a,b,c)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: too many values to unpack (expected 2)
```

Structures de données: Liste (list)

```
>>> lst = [1,3,5]
>>> type(lst)
<class 'list'>
>>> lst[0]
1
>>> lst[2]
5
>>> lst[-1]
5
>>> lst[3]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
>>> lst[1:3]
[3, 5]
```

Structures de données: Liste (list)

```
>>> lst = [1, 3, 5, 4]
>>> lst.append(5)
>>> lst
[1, 3, 5, 4, 5]
>>> lst.remove(5)
>>> lst
[1, 3, 4, 5]
>>> lst.pop(0)
>>> lst
[3, 4, 5]
>>> lst.extend([1,2])
>>> lst
[3, 4, 5, 1, 2]
>>> lst += [1,2]
>>> lst
[3, 4, 5, 1, 2, 1, 2]
```

Structures de données: Liste (list)

```
>>> a = [1,2]
>>> b = [1,2]
>>> a.append(3)
>>> a
[1, 2, 3]
>>> b
[1, 2]
>>> a = [1,2]
>>> b = a
>>> a.append(3)
>>> a
[1, 2, 3]
>>> b
[1, 2, 3]
```

Structures de données: Liste (list)

```
>>> t = (1,2,3)
>>> t2 = t
>>> t += (4,5)
>>> t
(1, 2, 3, 4, 5)
>>> t2
(1, 2, 3)
>>> a = [1,2,3]
>>> b = a
>>> a += (4,5)
>>> a
[1, 2, 3, 4, 5]
>>> b
[1, 2, 3, 4, 5]
```

Structures de données: Liste (list)

```
>>> a = [1,2,3]
>>> b = list(a)
>>> a.append(1)
>>> a
[1, 2, 3, 1]
>>> b
[1, 2, 3]
```

Structures de données: Liste (list)

```
>>> ma_list = [1,2,3]
>>> def afficher_avec_un_4( lst ):
...     lst.append(4)
...     print("Ma liste avec un 4:",lst)
...
>>> afficher_avec_un_4(ma_list)
Ma liste avec un 4: [1, 2, 3, 4]
>>> lst
[1, 2, 3, 4]
```

Structures de données: Liste (list)

Exercice:

1. Écrivez une fonction *fusion_tri* qui:
   - 1.1 prend en paramètres deux listes *lst1* et *lst2* composées de nombres triés par ordre croissant;
   - 1.2 retourne une liste *lst3* composés des nombres de lst1 et lst2 triés par ordre croissant également.
   - 1.3 (Vous avez le droit de modifier *lst1* et *lst2* en faisant ça.)
2. Exemple: si *lst1* $= [1, 2, 4]$ et *lst2* $= [2, 3, 7]$ alors *lst3* $= [1, 2, 2, 3, 4, 7]$.

Structures de données: Liste (list)

Correction:

```
def fusion_tri(lst1,lst2):
    lst3 = []
    while len(lst1)>0 and len(lst2)>0:
        if lst1[0] <= lst2[0]:
            lst3.append( lst1.pop(0) )
        else :
            lst3.append( lst2.pop(0) )
    lst3 = lst3 + lst1 + lst2
    return lst3
```

Structures de données: Ensemble (Set)

```
>>> set1 = set( range(1,5) )
>>> set1
{1, 2, 3, 4}
>>> set1 = set( [6, 4, 5, 4] )
>>> set1
{4, 5, 6}
>>> lst = list( set1 )
>>> lst
[4, 5, 6]
>>> tpl = tuple( set1 )
>>> tpl
(4, 5, 6)
```

Structures de données: Ensemble (Set)

```
>>> set1 = set()
>>> set1.add(1)
>>> set1
{1}
>>> set1.add(2)
>>> set1.add(4)
>>> set2
{1, 2, 4}
>>> set1.add(1)
>>> set1
{1, 2, 4}
>>> set1.remove(2)
>>> set1
{1, 4}
>>> set1.pop()
1
>>> set1
{4}
```

Structures de données: Ensemble (Set)

```
>>> set1 = {1, 2}
>>> set2 = {2, 3}
>>> set2[0]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'set' object does not support indexing
>>> set1
{1, 2}
>>> set1.update(set2)
>>> set1
{1, 2, 3}
>>> set2
{2, 3}
>>> set1.update( (3,4) )
>>> set1
{1, 2, 3, 4}
>>> set1.update( [4,5] )
>>> set1
```

Structures de données: Ensemble (Set)

```
>>> set1 = {1, 2, 3}
>>> set2 = {3, 4, 5}
>>> set1 | set2 # union
{1, 2, 3, 4, 5}
>>> set1 & set2 # intersection
{3}
>>> set1 - set2 # différence
{1, 2}
>>> set1 ^ set2 #  symmetric différence
{1, 2, 4, 5}
```

Structures de données: Ensemble (Set)

Exercice:

1. À l'aide de la structure *set*, faites un programme qui demande à l'utilisateur d'entrer un message et qui affiches les différents caractères qui le composent. Exemple: Si l'utilisateur entre: 'maman' le programme peut répondre:

```
Le caractère "a" apparait.
Le caractère "m" apparait.
Le caractère "n" apparait.
```

Structures de données: Ensemble (Set)

Exercice:

1. À l'aide de la structure *set*, faites un programme qui demande à l'utilisateur d'entrer un message et qui affiches les différents caractères qui le composent. Exemple: Si l'utilisateur entre: 'maman' le programme peut répondre:

   ```
   Le caractère "a" apparait.
   Le caractère "m" apparait.
   Le caractère "n" apparait.
   ```

Correction:

```
message = input("Entrez un message:")
set1 = set(message)
while len(set1) > 0: #ou while set1 :
    lettre = set1.pop()
    print('Le caractère"',lettre,'"apparait.')
```

Boucle Pour tout: (for)

```
>>> for i in range(3,5):
...     print(i)
...
3
4
>>> for nom in ("poule", "renard","viper" ) : #tuple
...     print(nom)
...
poule
renard
viper
>>> for c in "mots": #str
...     print(c)
...
m
o
t
s
```

Boucle Pour tout: (for)

```
>>> for e in [12,29,"canard",True]: #liste
...     print(e)
...
12
29
canard
True
>>> for e in { 1, 9 , 7, 2}: #ne pas se fier à l'ordre
...     print(e)
...
1
2
9
7
```

Boucle Pour tout: (for)

Exercice:

1. Écrire une fonction *afficher_multiplication* qui reçoit en paramètre une liste de nombre lst et qui affiche tous les produits qu'on peut obtenir en multipliant deux de ses nombres.
2. Exemple: si $lst = [1, 2]$ alors on affiche 1,2 et 4 (une seule fois chacun).

Boucle Pour tout: (for)

Exercice:

1. Écrire une fonction *afficher_multiplication* qui reçoit en paramètre une liste de nombre lst et qui affiche tous les produits qu'on peut obtenir en multipliant deux de ses nombres.
2. Exemple: si $lst = [1, 2]$ alors on affiche 1,2 et 4 (une seule fois chacun).

Correction:

```
def afficher_multiplication(lst):
    seen = set()
    for i in lst:
        for j in lst:
            if i*j not in seen:
                print(i*j)
                seen.add(i*j)
```