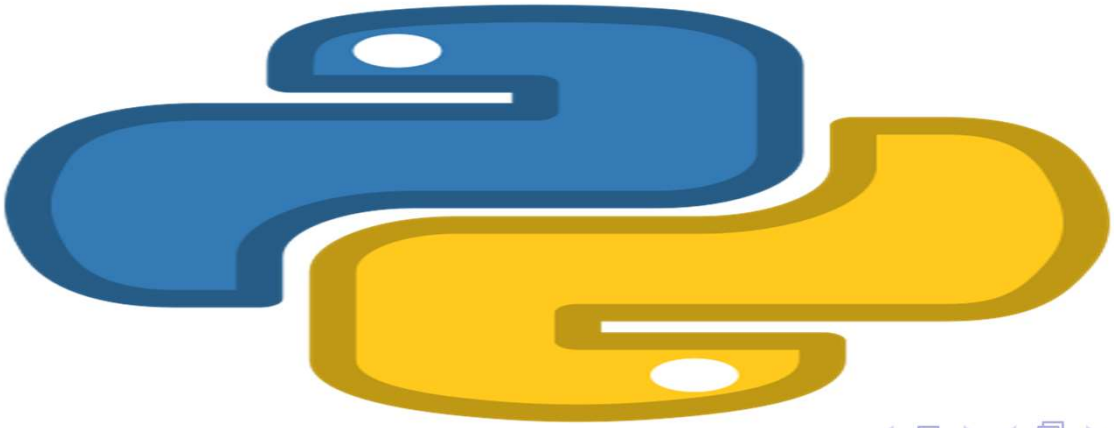
Python : itérables

Mohammed OUANAN

m.ouanan@umi.ac.ma



Plan

Python

Les itérables en **Python**

- Objets pouvant être parcourus séquentiellement.
- Comprenant des types de données prédéfinis et même personnalisés qui implémentent certaines méthodes spéciales telles que `__iter__()` et `__next__()`.
- Pouvant être parcourus via une boucle `for` ou des fonctions comme `map()`, `filter()`...

Python

Exemple d'itérables en **Python**

- `List` : collection ordonnée (selon l'ordre d'insertion) acceptant valeurs dupliquées et types différents
- `Tuple` : collection ordonnée (selon l'ordre d'insertion) de constante acceptant valeurs dupliquées et types différents
- `Set` : collection non-ordonnée (selon l'ordre d'insertion) et non-indexée acceptant les types différents et n'autorisant pas les valeurs dupliquées
- `Dictionary` : collection non-ordonnée acceptant les valeurs dupliquées avec des clés personnalisées
- `string`

Python

Pour déclarer une liste, on utilise les [ ]

```
liste = [2, 3, 8, 5]
```

Python

Pour déclarer une liste, on utilise les []

```
liste = [2, 3, 8, 5]
```

Pour accéder à un élément selon son indice (premier élément d'indice 0)

```
print(liste[1])
# affiche 3
```

Python

Pour déclarer une liste, on utilise les []

```
liste = [2, 3, 8, 5]
```

Pour accéder à un élément selon son indice (premier élément d'indice 0)

```
print(liste[1])
# affiche 3
```

Pour accéder au dernier élément d'une liste

```
print(liste[-1])
# affiche 5
```

Python

Pour déclarer une liste, on utilise les []

```
liste = [2, 3, 8, 5]
```

Pour accéder à un élément selon son indice (premier élément d'indice 0)

```
print(liste[1])
# affiche 3
```

Pour accéder au dernier élément d'une liste

```
print(liste[-1])
# affiche 5
```

Pour accéder à un élément selon son indice en commençant par la fin

```
print(liste[-2])
# affiche 8
```

Python

Accéder à un élément via un indice inexistant génère une erreur

```
print(list[10])
Traceback (most recent call last):
  File "c:/Users/admin/Desktop/cours-python/main.py"
    , line 2, in <module>
    print(list[10])
IndexError: list index out of range
```

Python

Accéder à un élément via un indice inexistant génère une erreur

```
print(list[10])
Traceback (most recent call last):
  File "c:/Users/admin/Desktop/cours-python/main.py"
    , line 2, in <module>
    print(list[10])
IndexError: list index out of range
```

Pour déterminer la taille d'une liste

```
print(len(liste))
# affiche 4
```

Python

Pour extraire une sous-liste

```
print(liste[1:3])
# affiche [3, 8]
```

Python

Pour extraire une sous-liste

```
print(liste[1:3])
# affiche [3, 8]
```

Ou avec des indices négatifs

```
print(liste[-3:-1])
# affiche [3, 8]
```

Python

Pour extraire une sous-liste

```
print(liste[1:3])
# affiche [3, 8]
```

Ou avec des indices négatifs

```
print(liste[-3:-1])
# affiche [3, 8]
```

Ou avec un seul indice

```
print(liste[-3:])
# affiche [3, 8, 5]
```

Python

Pour tester si un élément est dans la liste

```
if 3 in liste:
    print('oui')
# affiche oui
```

Python

Il est possible d'additionner (ou fusionner) les listes

```
liste.extend([1, 6])
print(liste)
# affiche [2, 3, 8, 5, 1, 6]
```

Python

Il est possible d'additionner (ou fusionner) les listes

```
liste.extend([1, 6])
print(liste)
# affiche [2, 3, 8, 5, 1, 6]
```

Ou en utilisant l'opérateur +=

```
liste += [1, 6]
print(liste)
# affiche [2, 3, 8, 5, 1, 6]
```

Python

On peut aussi utiliser l'unpacking

```
liste = [*liste, 1, 6]
print(liste)
# affiche [2, 3, 8, 5, 1, 6]
```

Python

On peut aussi utiliser l'unpacking

```
liste = [*liste, 1, 6]
print(liste)
# affiche [2, 3, 8, 5, 1, 6]
```

Sans l'opérateur $*$, le résultat est différent

```
liste = [liste, 1, 6]
print(liste)
# affiche [[2, 3, 8, 5], 1, 6]
```

Python

Pour ajouter un nouvel élément à la liste (à la fin)

```
liste.append(9)
print(liste)
# affiche [2, 3, 8, 5, 9]
```

Python

Pour ajouter un nouvel élément à la liste (à la fin)

```
liste.append(9)
print(liste)
# affiche [2, 3, 8, 5, 9]
```

Pour ajouter un nouvel élément à la liste à un emplacement précis

```
liste.insert(2, 6)
print(liste)
# affiche [2, 3, 6, 8, 5]
```

Python

L'instruction suivante permet de modifier l'élément d'indice 2

```
liste[2] = 10
print(liste)
# affiche [2, 3, 10, 5]
```

Python

L'instruction suivante permet de modifier l'élément d'indice 2

```
liste[2] = 10
print(liste)
# affiche [2, 3, 10, 5]
```

Mais ne permet pas d'ajouter si l'indice n'est pas dans le tableau

```
liste[4] = 4
print(liste)
# erreur
```

Python

L'instruction suivante permet de modifier l'élément d'indice 2

```
liste[2] = 10
print(liste)
# affiche [2, 3, 10, 5]
```

Mais ne permet pas d'ajouter si l'indice n'est pas dans le tableau

```
liste[4] = 4
print(liste)
# erreur
```

Cette expression PHP déclenche aussi une erreur

```
liste[] = 4
# erreur
```

Python

Pour supprimer le dernier élément de la liste, on utilise `pop`

```
liste.pop()
print(liste)
# affiche [2, 3, 8]
```

Python

Pour supprimer le dernier élément de la liste, on utilise `pop`

```
liste.pop()
print(liste)
# affiche [2, 3, 8]
```

Pour supprimer un élément de la liste selon la valeur (première occurrence), on utilise `remove`. Si l'élément n'existe pas, une exception sera levée.

```
liste.remove(8)
print(liste)
# affiche [2, 3, 5]
```

Python

Pour supprimer un élément de la liste selon l'indice. Si l'indice est supérieur à la taille de la liste, une exception sera levée.

```
liste.pop(1)
print(liste)
# affiche [2, 8, 5]
```

Python

Pour supprimer un élément de la liste selon l'indice. Si l'indice est supérieur à la taille de la liste, une exception sera levée.

```
liste.pop(1)
print(liste)
# affiche [2, 8, 5]
```

Ou aussi

```
del  liste[1]
print(liste)
# affiche [2, 8, 5]
```

Python

Pour parcourir une liste

```
for elt in liste:
  print(elt)
```

Python

Pour parcourir une liste

```
for elt in liste:
  print(elt)
```

Le résultat est

```
2
3
8
5
```

Python

Pour afficher la liste dans le sens inverse (sans modifier la liste initiale)

```
for elt in reversed(liste):
  print(elt)
```

Python

Pour afficher la liste dans le sens inverse (sans modifier la liste initiale)

```
for elt in reversed(liste):
  print(elt)
```

Le résultat est

```
5
8
3
2
```

Python

Pour afficher les éléments et leurs indices

```
for ind in range(len(liste)):
    print(ind, liste[ind])
```

Python

Pour afficher les éléments et leurs indices

```
for ind in range(len(liste)):
    print(ind, liste[ind])
```

Le résultat est

```
0 2
1 3
2 8
3 5
```

Python

On peut aussi utiliser `enumerate`

```
for ind, elt in enumerate(liste):
    print(ind, elt)
```

Python

On peut aussi utiliser `enumerate`

```
for ind, elt in enumerate(liste):
    print(ind, elt)
```

Le résultat est

```
0 2
1 3
2 8
3 5
```

Python

Pour modifier la valeur initiale de l'indice, on utilise `start`

```
for ind, elt in enumerate(liste, start=1):
    print(ind, elt)
```

Python

Pour modifier la valeur initiale de l'indice, on utilise `start`

```
for ind, elt in enumerate(liste, start=1):
    print(ind, elt)
```

Le résultat est

```
1 2
2 3
3 8
4 5
```

Python

Autres méthodes sur les listes

- `clear()` : supprime tous les éléments de la liste.
- `count(x)` : compte le nombre de `x` dans la liste.
- `index(x)` : retourne l'indice de la première occurrence de `x` dans la liste, une exception sera levée si `x` n'est pas dans la liste.
- `sort()` : trie la liste (modifie la liste).
- `reverse()` : inverse l'ordre des éléments de la liste (modifie la liste).
- ...

Python

Exemple avec `index(value, start, stop)`

```
liste2 = [2, 3, 8, 5, 0, 3, 1, 3]

print(liste2.index(3))
# affiche 1

print(liste2.index(3, 2))
# affiche 5

print(liste2.index(3, -1))
# affiche 7

print(liste2.index(3, 0, 5))
# affiche 1
```

Python

Exercice 1 : Étant donnée la liste suivante

```
ma_liste = [2, 7, 2, 1, 3, 9, 2, 4, 2]
```

Écrire un programme Python qui permet de supprimer l'avant dernière occurrence du chiffre 2 de la liste précédente

Python

Exercice 1 : Étant donnée la liste suivante

```
ma_liste = [2, 7, 2, 1, 3, 9, 2, 4, 2]
```

Écrire un programme Python qui permet de supprimer l'avant dernière occurrence du chiffre 2 de la liste précédente

Le résultat attendu

```
[2, 7, 2, 1, 3, 9, 4, 2]
```

Python

Une solution possible

```
ma_liste = [2, 7, 2, 1, 3, 9, 2, 4, 2]

ma_liste.reverse()
ma_liste.pop(ma_liste.index(2, ma_liste.index(2) + 1))
ma_liste.reverse()
print(ma_liste)
```

Python

Il est possible de décomposer les valeurs d'une liste et les stocker dans des variables

```
a, b, c, d = liste
print(a, b, c, d)
# affiche 2 3 8 5
```

Python

Il est possible de de´composer les valeurs d'une liste et les stocker dans des variables

```
a, b, c, d = liste
print(a, b, c, d)
# affiche 2 3 8 5
```

Et si le nombre de variables est infe´rieur à la taille de la liste, on aura une erreur

```
a, b, c = liste
print(a, b, c, d)
```

Python

Il est possible de décomposer les valeurs d'une liste et les stocker dans des variables

```
a, b, c, d = liste
print(a, b, c, d)
# affiche 2 3 8 5
```

Et si le nombre de variables est inférieur à la taille de la liste, on aura une erreur

```
a, b, c = liste
print(a, b, c, d)
```

On peut aussi récupérer le restant des valeurs dans une variable

```
a, b, *c = liste
print(a, b, c)
# affiche 2 3 [8 5]
```

Python

Les valeurs d'une liste peuvent être construites à partir de plusieurs variables

```
a, b, c, d = 2, 3, 8, 5
liste = [a, b, c, d]
print(liste)
# affiche [2, 3, 8, 5]
```

Python

Exercice

Écrire un programme **Python** qui

1. demande à l'utilisateur de remplir un tableau de notes : la saisie s'arrête si la valeur n'est pas comprise entre 0 et 20.
2. affiche le max, le min et la moyenne de notes

Python

Pour créer une liste vide

```
liste = list()
```

Python

Pour créer une liste vide

```
liste = list()
```

Ou

```
liste = []
```

Python

Pour créer une liste et l'initialiser avec dix zéros

```
liste_zeros = [0] * 10

print(liste_zeros)
# affiche [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

Python

Pour créer une liste et l'initialiser avec dix zéros

```
liste_zeros = [0] * 10

print(liste_zeros)
# affiche [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

Pour créer une liste et l'initialiser avec des valeurs allant de zéro à dix inclus

```
liste_de_zero_a_dix = list(range(11))

print(liste_de_zero_a_dix)
# affiche [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Python

Liste multi-dimensions ≡ matrice

- liste de sous-listes
- chaque sous-liste correspond à une ligne de la matrice

Python

Pour créer une matrice 2x2

```
matrice2 = [
     [1, 2],
     [3, 4]
]
```

Python

Pour créer une matrice 2x2

```
matrice2 = [
    [1, 2],
    [3, 4]
]
```

Pour créer une matrice 3x3

```
matrice3 = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
]
```

Python

Pour accéder à un élément

```
print(matrice3[0][0])
# affiche 1
```

Python

Pour accéder à un élément

```
print(matrice3[0][0])
# affiche 1
```

Pour parcourir la matrice ligne par ligne

```
for ligne in matrice3:
    print(ligne, end=" ")

# affiche [1, 2, 3] [4, 5, 6] [7, 8, 9]
```

Python

Pour accéder à un élément

```
print(matrice3[0][0])
# affiche 1
```

Pour parcourir la matrice ligne par ligne

```
for ligne in matrice3:
    print(ligne, end=" ")

# affiche [1, 2, 3] [4, 5, 6] [7, 8, 9]
```

Pour parcourir la matrice élément par élément

```
for i in range(len(matrice3)):
    for j in range(len(matrice3[i])):
        print(matrice3[i][j], end=" ")
# affiche 1 2 3 4 5 6 7 8 9
```

Python

Étant données les variables suivantes

```
valeur = 5
matrice = [
    [2, 3, 5],
    [1, 2, 5],
    [3, 1, 5]
]
```

Python

Étant données les variables suivantes

```
valeur = 5
matrice = [
    [2, 3, 5],
    [1, 2, 5],
    [3, 1, 5]
]
```

Exercice

Écrire un script **Python** qui permet de déterminer si `valeur` est présente dans chaque ligne de `matrice`.

```
def valeur_dans_chaque_ligne(matrice, valeur):
                return [valeur in ligne for ligne in matrice]
```

Python

Exercice

Écrire un script **Python** qui calcule

1. la somme de deux matrices carrées,
2. le produit de deux matrices carrées.

Python

Pour déclarer un tuple, on utilise les ()

```
marques = ("peugeot", "ford", "toyota")
```

Python

Pour déclarer un tuple, on utilise les ()

```
marques = ("peugeot", "ford", "toyota")
```

Pour accéder à un élément selon son indice (premier élément d'indice 0)

```
print(marques[1])
# affiche ford
```

Python

Pour déclarer un tuple, on utilise les ()

```
marques = ("peugeot", "ford", "toyota")
```

Pour accéder à un élément selon son indice (premier élément d'indice 0)

```
print(marques[1])
# affiche ford
```

Pour accéder au dernier élément d'une tuple

```
print(marques[-1])
# affiche toyota
```

Python

Accéder à un élément via un indice inexistant génère une erreur

```
print(marques[10])
Traceback (most recent call last):
  File "c:/Users/admin/Desktop/cours-python/main.py", line 2,
    in <module>
    print(marques[10])
IndexError: tuple index out of range
```

Python

Accéder à un élément via un indice inexistant génère une erreur

```
print(marques[10])
Traceback (most recent call last):
  File "c:/Users/admin/Desktop/cours-python/main.py", line 2,
     in <module>
    print(marques[10])
IndexError: tuple index out of range
```

Pour déterminer la taille d'un tuple

```
print(len(marques))
# affiche 3
```

Python

Modifier un élément d'un tuple génère une erreur

```
marques[1] = "fiat"
Traceback (most recent call last):
  File "c:/Users/admin/Desktop/cours-python/main.py", line 2, in <
     module>
    marques[1] = "fiat"
TypeError: 'tuple' object does not support item assignment
```

Python

Modifier un élément d'un tuple génère une erreur

```
marques[1] = "fiat"
Traceback (most recent call last):
  File "c:/Users/admin/Desktop/cours-python/main.py", line 2, in <
     module>
    marques[1] = "fiat"
TypeError: 'tuple' object does not support item assignment
```

Remarques

- Il est impossible d'ajouter ou supprimer un élément d'un tuple
- `count(x)` et `index(x)` sont applicables sur les tuples

Python

Pour parcourir un tuple

```
for elt in marques:
  print(elt)
```

Python

Pour parcourir un tuple

```
for elt in marques:
  print(elt)
```

Le résultat est

```
peugeot
ford
toyota
```

Python

Attention aux détails suivants

```
t = (1)

print(t, type(t).__name__)
# affiche 1 int

t = (1,)

print(t, type(t).__name__)
# affiche (1,) tuple

print(len(t))
# affiche 1
```

Python

Le comportement est différent pour les listes

```
t = [1]

print(t, type(t).__name__)
# affiche [1] list
t = [1,]

print(t, type(t).__name__)
# affiche [1] list

print(len(t))
# affiche 1
```

Python

Set

- collection non ordonnée et non indexée
- contenant des éléments uniques
- ses éléments sont immuables mais le set est muable.

Python

Pour déclarer un set, on utilise les *{}*

```
ensemble = {2, 3, 8, 5}
```

Python

Pour déclarer un set, on utilise les *{}*

```
ensemble = {2, 3, 8, 5}
```

Impossible d'accéder à un élément d'un ensemble via son indice (un ensemble n'est pas ordonné et est donc non indexé)

```
print(ensemble[1])
Traceback (most recent call last):
  File "c:/Users/User/cours-python/main.py", line 2, in <module>
    print(ensemble[1])
TypeError: 'set' object is not subscriptable
```

Python

Pour déclarer un set, on utilise les *{}*

```
ensemble = {2, 3, 8, 5}
```

Impossible d'accéder à un élément d'un ensemble via son indice (un ensemble n'est pas ordonné et est donc non indexé)

```
print(ensemble[1])
Traceback (most recent call last):
  File "c:/Users/User/cours-python/main.py", line 2, in <module>
    print(ensemble[1])
TypeError: 'set' object is not subscriptable
```

Pour déterminer la taille d'un set

```
print(len(ensemble))
# affiche 4
```

Python

Pour ajouter un nouvel élément à l'ensemble

```
ensemble.add(6)
print(ensemble)
# affiche {2, 3, 5, 6, 8}
```

Python

Pour ajouter un nouvel élément à l'ensemble

```
ensemble.add(6)
print(ensemble)
# affiche {2, 3, 5, 6, 8}
```

Pour ajouter plusieurs éléments simultanément

```
ensemble.update([6, 1])
print(ensemble)
# affiche {1, 2, 3, 5, 6, 8}
```

Python

Pour ajouter un nouvel élément à l'ensemble

```
ensemble.add(6)
print(ensemble)
# affiche {2, 3, 5, 6, 8}
```

Pour ajouter plusieurs éléments simultanément

```
ensemble.update([6, 1])
print(ensemble)
# affiche {1, 2, 3, 5, 6, 8}
```

Remarque

La méthode `update` prend comme paramètre un itérable : `list`, `set`...

Python

On peut ajouter un tuple dans un `set`

```
ensemble.add((1, 4))
print(ensemble)
# affiche {2, 3, 5, 8, (1, 4)}
```

Python

On peut ajouter un tuple dans un `set`

```
ensemble.add((1, 4))
print(ensemble)
# affiche {2, 3, 5, 8, (1, 4)}
```

On ne peut ajouter un type muable (comme `list` ou `dict`) dans un `set`

```
ensemble.add([1, 4])
print(ensemble)
# affiche erreur
```

Python

Pour supprimer un élément

```
ensemble.remove(3)
print(ensemble)
# affiche {8, 2, 5}
```

Python

Pour supprimer un élément

```
ensemble.remove(3)
print(ensemble)
# affiche {8, 2, 5}
```

Si l'élément à supprimer n'existe pas, `remove` **déclenche une erreur**

```
ensemble.remove(9)
Traceback (most recent call last):
  File "c:/Users/User/cours-python/main.py", line 2, in <module
    >
    ensemble.remove(9)
KeyError: 9
```

Python

On peut utiliser `discard` pour supprimer (ne déclenche pas d'erreur si l'élément n'existe pas)

```
ensemble.discard(9)
print(ensemble)
# affiche {8, 2, 3, 5}
```

Python

On peut utiliser `discard` pour supprimer (ne déclenche pas d'erreur si l'élément n'existe pas)

```
ensemble.discard(9)
print(ensemble)
# affiche {8, 2, 3, 5}
```

Pour supprimer arbitrairement un élément, on peut utiliser `pop` (L'élément supprimé n'est pas forcément le dernier)

```
ensemble.pop()
print(ensemble)
# affiche {2, 3, 5}
```

Python

Pour parcourir un set

```
for elt in ensemble:
  print(elt)
```

Python

Pour parcourir un set

```
for elt in ensemble:
  print(elt)
```

Le résultat est

```
2
3
8
5
```

Python

Autres méthodes sur les ensembles et opérateurs surchargés

- `clear()` : supprime tous les éléments de la liste.
- `union()` : retourne l'ensemble de tous les éléments des deux ensembles sans doublons (opérateur : |).
- `intersection()` : retourne les éléments en commun entre deux ensembles (opérateur : `&`).
- `difference()` : retourne un ensemble contenant les éléments différents de deux ensembles ou plus (opérateur : -).
- `symmetric_difference()` : retourne l'ensemble de tous les éléments qui se trouvent soit dans le premier ensemble, soit dans le deuxième ensemble mais pas dans les deux (opérateur : ^).
- `issubset()` : retourne `True` si tous les éléments d'un ensemble sont contenus dans un autre, `False` sinon (opérateur : <=).
- `issuperset()` : retourne `True` si un ensemble contient tous les éléments d'un autre, `False` sinon (opérateur : >=).
- `isdisjoint()` : retourne `True` si deux ensembles n'ont aucun élément en commun, `False` sinon.
- ...

Python

Considérons les deux ensembles suivants

```
set1 = {1, 2, 3, 4}
set2 = {3, 4, 5, 6}
```

Python

Considérons les deux ensembles savants

```
set1 = {1, 2, 3, 4}
set2 = {3, 4, 5, 6}
```

Pour avoir l'intersection

```
print(set1.intersection(set2))
# affiche {3, 4}
```

Python

Considérons les deux listes suivantes

```
set1 = {1, 2, 3, 4}
set2 = {3, 4, 5, 6}
```

Pour avoir l'intersection

```
print(set1.intersection(set2))
# affiche {3, 4}
```

Ou

```
print(set1 & set2)
# affiche {3, 4}
```

Python

Pour avoir l'union

```
print(set1.union(set2))
# affiche {1, 2, 3, 4, 5, 6}
```

Python

```
print(set1.union(set2))
# affiche {1, 2, 3, 4, 5, 6}
```

Ou

```
print(set1 | set2)
# affiche {1, 2, 3, 4, 5, 6}
```

Python

Pour avoir la différence

```
print(set1.difference(set2))
# affiche {1, 2}
```

Python

Pour avoir la différence

```
print(set1.difference(set2))
# affiche {1, 2}
```

Ou

```
print(set1 - set2)
# affiche {1, 2}
```

Considérons les trois ensembles suivants

```
ensemble1 = {2, 3, 8, 5}
ensemble2 = {7, 2, 9, 3}
ensemble3 = {1, 2, 4, 5}
```

Considérons les trois ensembles suivants

```
ensemble1 = {2, 3, 8, 5}
ensemble2 = {7, 2, 9, 3}
ensemble3 = {1, 2, 4, 5}
```

Exercice

Écrire un programme **Python** qui

1. vérifie que l'intersection entre `ensemble3` et `ensemble2` est un sous-ensemble de `ensemble1`
2. affiche les valeurs présentes dans `ensemble1` **ou** `ensemble2` mais pas dans `ensemble3`
3. affiche les valeurs présentes dans `ensemble1` **et** `ensemble2` mais pas dans `ensemble3`
4. affiche les valeurs présentes dans `ensemble1` **ou** `ensemble2` mais pas dans les deux ensembles à la fois

Python

Correction

```
# Question 1
resultat1 = (ensemble3 & ensemble2) <= ensemble1
print(resultat1)
# affiche True

# Question 2
resultat2 = (ensemble1 | ensemble2) - ensemble3
print(resultat2)
# affiche {8, 9, 3, 7}

# Question 3
resultat3 = (ensemble1 & ensemble2) - ensemble3
print(resultat3)
# affiche {3}
# Question 4
resultat4 = (ensemble1 ^ ensemble2)
print(resultat4)
# affiche {8, 9, 5, 7}
```

Python

Pour créer un set vide

```
ensemble = set()
```

Python

Pour créer un set vide

```
ensemble = set()
```

Attention, l'écriture suivante ne permet pas de créer un set

```
ensemble = {}
```

Python

`frozenset` : cas particulier de `set`

- `frozenset` est une collection non-modifiable (immuable).
- on ne peut pas modifier son contenu en ajoutant, supprimant ou modifiant des éléments.
- `frozenset` possède toutes les méthodes d'un `set` (telles que `diff´erence()`, `symmetric_difference()`, `union()`...), mais comme il est non-modifiable, il ne dispose pas de méthodes pour ajouter/supprimer des éléments.

Python

Pour déclarer un dictionnaire (`key: value`), on utilise les *{}*

```
fcb = {
    "messi": 10,
    "suarez": 9,
    "rakitic": 4,
    "umtiti": 23
}
```

Python

Pour déclarer un dictionnaire (`key: value`), on utilise les *{}*

```
fcb = {
    "messi": 10,
    "suarez": 9,
    "rakitic": 4,
    "umtiti": 23
}
```

Pour accéder à une valeur selon la clé

```
print(fcb['messi'])
# affiche 10
```

Python

Pour déclarer un dictionnaire (`key: value`), on utilise les *{}*

```
fcb = {
    "messi": 10,
    "suarez": 9,
    "rakitic": 4,
    "umtiti": 23
}
```

Pour accéder à une valeur selon la clé

```
print(fcb['messi'])
# affiche 10
```

Ou aussi

```
print(fcb.get('messi'))
# affiche 10
```

Python

Si la clé n'existe pas, la fonction `get` retourne `None`

```
print(fcb.get('ronaldo'))
# affiche None
```

Python

Si la clé n'existe pas, la fonction `get` retourne `None`

```
print(fcb.get('ronaldo'))
# affiche None
```

Pour spécifier une valeur par défaut si jamais la cle´ n'existe pas

```
print(fcb.get('ronaldo', -1))
# affiche -1
```

Python

Pour extraire la liste de valeurs d'un dictionnaire

```
print(fcb.values())
# affiche dict_values([10, 9, 4, 23])
```

Python

Pour extraire la liste de valeurs d'un dictionnaire

```
print(fcb.values())
# affiche dict_values([10, 9, 4, 23])
```

Pour extraire la liste de clés d'un dictionnaire

```
print(fcb.keys())
# affiche dict_keys(['messi', 'suarez', 'rakitic', '
   umtiti'])
```

Python

Pour extraire la liste de valeurs d'un dictionnaire

```
print(fcb.values())
# affiche dict_values([10, 9, 4, 23])
```

Pour extraire la liste de clés d'un dictionnaire

```
print(fcb.keys())
# affiche dict_keys(['messi', 'suarez', 'rakitic', '
   umtiti'])
```

Pour connaître le nombre d'items d'un dictionnaire

```
print(len(fcb))
# affiche 4
```

Python

Pour vérifier si une clé est présente dans un dictionnaire

```
print('messi' in fcb)
# affiche True

print('mitroglou' in fcb)
# affiche False
```

Pour ajouter un nouvel item au dictionnaire

```
fcb['iniesta'] = 8
print(fcb)
# affiche {'messi': 10, 'suarez': 9, 'rakitic': 4, 'umtiti': 23, '
   iniesta': '8'}
```

Pour ajouter un nouvel item au dictionnaire

```
fcb['iniesta'] = 8
print(fcb)
# affiche {'messi': 10, 'suarez': 9, 'rakitic': 4, 'umtiti': 23, '
   iniesta': '8'}
```

Si la clé existe, la valeur associée sera modifiée

```
fcb['messi'] = 11
print(fcb)
# affiche {'messi': 11, 'suarez': 9, 'rakitic': 4, 'umtiti': 23}
```

Pour ajouter un nouvel item au dictionnaire

```
fcb['iniesta'] = 8
print(fcb)
# affiche {'messi': 10, 'suarez': 9, 'rakitic': 4, 'umtiti': 23, '
   iniesta': '8'}
```

Si la clé existe, la valeur associée sera modifiée

```
fcb['messi'] = 11
print(fcb)
# affiche {'messi': 11, 'suarez': 9, 'rakitic': 4, 'umtiti': 23}
```

On peut utiliser la méthode `update` pour ajouter plusieurs items ou modifier des items existants

```
fcb.update({'messi': 11, 'iniesta': 8})
print(fcb)
# affiche {'messi': 11, 'suarez': 9, 'rakitic': 4, 'umtiti': 23, '
   iniesta': 8}
```

Python

Pour supprimer un item selon la clé

```
fcb.pop('rakitic')
print(fcb)
# affiche {'messi': 10, 'suarez': 9, 'umtiti': 23}
```

Python

Pour supprimer un item selon la clé

```
fcb.pop('rakitic')
print(fcb)
# affiche {'messi': 10, 'suarez': 9, 'umtiti': 23}
```

Ou aussi

```
del fcb['rakitic']
print(fcb)
# affiche {'messi': 10, 'suarez': 9, 'umtiti': 23}
```

Python

Pour supprimer un item selon la clé

```
fcb.pop('rakitic')
print(fcb)
# affiche {'messi': 10, 'suarez': 9, 'umtiti': 23}
```

Ou aussi

```
del fcb['rakitic']
print(fcb)
# affiche {'messi': 10, 'suarez': 9, 'umtiti': 23}
```

Pour supprimer le dernier item inséré

```
fcb.popitem()
print(fcb)
# affiche {'messi': 10, 'suarez': 9, 'rakitic': 4}
```

Python

Pour parcourir un dictionnaire

```
for key in fcb:
  print(key + ' ' + str(fcb[key]))
```

Python

Pour parcourir un dictionnaire

```
for key in fcb:
  print(key + ' ' + str(fcb[key]))
```

Le résultat est

```
messi 10
suarez 9
rakitic 4
umtiti 23
```

Python

Ou aussi

```
for key, value in fcb.items():
  print(key + ' ' + str(value))
```

Python

Ou aussi

```
for key, value in fcb.items():
  print(key + ' ' + str(value))
```

Le résultat est le même

```
messi 10
suarez 9
rakitic 4
umtiti 23
```

Python

Autres méthodes sur les dictionnaires

- `clear()` : supprime tous les items d'un dictionnaire
- `copy()` : copie un dictionnaire
- ...

Python

Dans Python 3.9, on peut fusionner deux dictionnaires en utilisant l'opérateur d'union|

```
fcb1 = {
    "messi": 10,
    "suarez": 9
}

fcb2 = {

    "rakitic": 4,
    "umtiti": 23
}
fcb = fcb1 | fcb2

print(fcb1)
# affiche {'messi': 10, 'suarez': 9}

print(fcb2)
# affiche {'rakitic': 4, 'umtiti': 23}

print(fcb)
# affiche {'messi': 10, 'suarez': 9, 'rakitic': 4, 'umtiti': 23}
```

Python

Étant donné le dictionnaire suivant

```
repetition = {
    "Java": 2,
    "PHP": 5,
    "C++": 1,
    "HTML": 4
}
```

Exercice 1

Écrire un programme **Python** qui permet de répéter l'affichage de chaque dé de ce dictionnaire selon la valeur associée

Résultat attendu (l'ordre n'a pas d'importance) :

```
JavaJava PHPPHPPHPPHPPHP C++ HTMLHTMLHTMLHTML
```

Python

Exercice 2 : Étant donnée la liste suivante :

```
liste = [2, 5, "Bonjour", True, 'c', "3", "b", False, 10]
```

Écrire un programme Python qui permet de stocker dans un dictionnaire (`compteur`) les types contenus dans la liste `liste` ainsi que le nombre d'éléments de cette liste appartenant à chaque type.

Résultat attendu :

```
{'int': 3, 'str': 4, 'bool': 2}
```

Python

Une première solution

```
liste = [2, 5, "Bonjour", True, 'c', "3", "b", False, 10]

compteur = {}

for item in liste:
    key_type = type(item).__name__
    if key_type not in compteur:
        compteur[key_type] = 0
    compteur[key_type] += 1

print(compteur)
# affiche {'int': 3, 'str': 4, 'bool': 2}
```

Python

Une deuxième solution

```
liste = [2, 5, "Bonjour", True, 'c', "3", "b", False, 10]

compteur = {}

for item in liste:
    key_type = type(item).__name__
    compteur[key_type] = compteur.get(key_type, 0) + 1

print(compteur)
# affiche {'int': 3, 'str': 4, 'bool': 2}
```

Python

Pour créer un dictionnaire vide

```
dictionnaire = dict()
```

Python

Pour créer un dictionnaire vide

```
dictionnaire = dict()
```

Ou

```
dictionnaire = {}
```

map, filter, reduce

- Pour appliquer une fonction à chaque object d'une entité itérable (ex une liste)
- Syntaxe `map(function, iterable, ...)`
  '... 'exprime la possibilité de passer une liste contenant les paramètres pour la fonction
- `map` est une built-in function qui renvoie une liste en Python 2, un iterable avec Python 3
- Exemples

Code

```
def square(value):
  return (value * value)

# will be [0, 1, 4, 9, 16]
squares = map(square, range(0, 5))

# display
for x in squares:
  print(x, end=" ")
print()

##
def my_pow(value1, value2):
  return (value1 ** value2)

# will be [0**1, 1**2, 2**3, 3**4, 4**5]
pows_list = map(my_pow, range(0, 5), [1, 2, 3, 4, 5])

# display
for pow in pows_list:
  print(pow, end=" ")
print()
```

Exécution

```
# square function usage
0 1 4 9 16

# my_pow function usage
0 1 8 81 1024
```

map, filter, reduce

- filter(function, iterable)
  - Pour filtrer les objets d'une entité itérable (ex une liste)
  - Syntaxe : filter(function, iterable)
  - filter est une built-in function qui renvoie une liste en Python 2, un iterable avec Python 3
  - La fonction passée en paramètre de filter doit renvoyer un booléen
  - Exemple : Liste des mots qui commencent par une lettre majuscule

Code

```
def starts_with_capitalized_char(value):
  return((value[0]).isupper())

msg = "Python is an INTERESTING language"
msg_as_list = msg.split();
words = filter(starts_with_capitalized_char, msg_as_list)

#  will be ['Python', 'INTERESTING']
for word in words:
  print(word, end=" ")
```

Exécution

```
Python INTERESTING
```

- `reduce(function, iterable[, initializer=None])`
  - Pour appliquer une fonction à 2 arguments, et réduire le résultat à une valeur retournée par la fonction.
  - Cette valeur retournée sera le 1er argument, le 2eme étant pris dans la séquence lors de l'itération suivante, etc.
  - Syntaxe : `reduce(function, iterable[, initializer])`
  - `reduce` est une built-in function en Python 2, mais a été déplacée dans le module `functools` en Python 3 : `from functools import reduce`
  - Exemple : Calcul de la somme des N premiers nombres

Code

```
from functools import reduce

def my_add(v1, v2):
  return (v1 + v2)

my_sum = reduce(my_add, range(1, 6))
print(my_sum)

# other example with init value
# it will be used once, during the first iteration
# result will be 100 + sum of 5 first numbers => 115
my_sum = reduce(my_add, range(1, 6), 100)
print(my_sum)
```

Exécution

```
15

115
```

- lambda
  - Pour définir une fonction anonyme
  - Syntaxe : lambda argument_list : expression
  - Exemple

Classiquement, avec définition d'une fonction

```
def my_sqr(x):
  return (x**2)

print(my_sqr(123))
```

Avec l'utilisation d'expression lambda

```
your_sqr = lambda x: x**2

print(your_sqr(123))
```

  - Autre exemple, déjà vu avec la fonction `filter`, mais ré-ecrit avec le mécanisme lambda

```
# lambda allows function definition and usage at the same time

msg = "Python is an INTERESTING language"
msg_as_list = msg.split();
words = filter(lambda c: c[0].isupper(), msg_as_list)

#  will be ['Python', 'INTERESTING']
for word in words:
  print(word, end=" ")
```

  - Contrainte majeure
    Sur seule ligne et une seule instruction dans la fonction !

Python

En **Python**

- Toute variable est un objet (instance d'une classe)
- Deux types d'objet
  - **Mutable** (ou muable) : peut être modifié après sa création (en préservant la même identité)
  - **Immutable** (ou immuable) : ne peut être modifié après sa création (peut être réaffecté)

Python

En **Python**

- Toute variable est un objet (instance d'une classe)
- Deux types d'objet
  - **Mutable** (ou muable) : peut être modifié après sa création (en préservant la même identité)
  - **Immutable** (ou immuable) : ne peut être modifié après sa création (peut être réaffecté)

Ne pas confondre modification et affectation

Python

Exemple d'objets mutables

- `list`
- `dict`
- `set`
- classes personnalisées

Python

Exemple d'objets mutables

- `list`
- `dict`
- `set`
- classes personnalisées

Exemple d'objet mutable

```
liste = [2, 3, 8, 5]

print(id(liste))
# affiche 20373256

liste[2] = 4
print(id(liste))
```

Python

Exemple d'objets immuables

- `int` et `float`
- `str`
- `bool`
- `tuple`

Python

Exemple d'objets immuables

- `int` et `float`
- `str`
- `bool`
- `tuple`

Exemple d'objets immuables

```
ch = "bonjour"

print(id(ch))
# affiche 9417152

ch = "bonsoir"
print(id(ch))
# affiche 9417216
```

Python

Un objet immuable peut être réaffecté mais ne peut être modifié

```
ch = "bonjour"

print(ch)
# affiche bonjour

ch[3] = "t"
# affiche TypeError: 'str' object does not support
   item assignment
```

Python

Objet muable vs objet immuable

- Objets immuables : accès plus rapide et modification coûteuse (création d'une nouvelle copie)
- Objets muables : modification moins coûteuse et accès moins rapide

Python

Un tuple est un objet immuable, donc il ne peut être modifié

```
tpl = ([2, 3], [8, 5])

tpl[0] = 1
# affiche TypeError: 'tuple' object does not support item
    assignment
```

Python

Un tuple est un objet immuable, donc il ne peut être modifié

```
tpl = ([2, 3], [8, 5])

tpl[0] = 1
# affiche TypeError: 'tuple' object does not support item
    assignment
```

Si un élément d'un tuple est muable, alors il est possible de le modifier

```
tpl = ([2, 3], [8, 5])

tpl[0][1] = 1

print(tpl)
# affiche ([2, 1], [8, 5])
```