esprit
Se former autrement

Python 3

Année universitaire 2020-2021

Python

Opération d'affichage : print() :

```
print ("message")
print(a+b)
```

Opération de lecture : input() :

- int() si on attend un entier
- float() si on attend un réel

```
a=int(input("donner un entier a="))
```

Les commentaires :

```
#Ceci est un commentaire
```

Application 1

Exercice1 :

Écrire un programme qui affiche "Bonjour le monde".

Exercice 2 :

Écrire un programme qui permet de saisir le nom de l'utilisateur et de renvoyer "Bonjour", suivi de ce nom.

Exercice 3 :

Calculer la somme de 2 nombres a et b introduits au clavier.

Exercice 4:

Écrire un programme qui demande de l'utilisateur un entier et affiche son carré.

Exemple : le carré de 7 est 49

Application 1_Correction

1.

```
print("Bonjour le monde")
```

```
Bonjour le monde
```

2.

```
a=input("saisir votre nom")
print("Bonjour",a)
```

```
saisir votre nomESB
Bonjour ESB
```

3.

```
a=int(input("saisir a   "))
b=int(input("saisir b   "))
print(a+b)
```

```
saisir a   8
saisir b   9
17
```

4.

```
a=int(input("saisir un entier   "))
print(a**2)
```

```
saisir un entier   8
64
```

Les variables et les types de valeurs Python

Une variable est constituée de 2 choses :

- Elle a une valeur : c'est la donnée qu'elle stocke (par exemple le nombre 5).
- Elle a un nom : c'est ce qui permet de la reconnaître. Nous n'aurons pas à retenir l'adresse de mémoire, nous allons juste indiquer des noms de variables à la place.

i=12

Les variables et les types de valeurs Python

- **Affectation simple:**

```
n=7
msg="Quoi de neuf"
pi=3.14
```

Affichage:

```
print(n)
```

```
7
```

```
print(msg)
```

```
Quoi de neuf
```

```
print(pi)
```

```
3.14
```

Les variables et les types de valeurs Python

Assigner une valeur à plusieurs variables:

```
x=y=7
```

```
x
```

```
7
```

```
y
```

```
7
```

Effectuer des affectations parallèles:

```
a,b=4,8
```

```
a
```

```
4
```

```
b
```

```
8
```

Type

Python définit trois types de valeurs numériques supportées :

- Le type *int* qui représente tout entier positif ou négatif ; +, -, *, ** …

  Les opérateurs de comparaisons:

```
x==y # x est égal à y
x!=y # x est différent de y
x>y # x est plus grand que y
x<y # x est plus petit que y
x>=y # x est plus grand que, ou égal à y
x<=y # x est plus petit que, ou égal à y
```

- Le type *float* qui représente les nombres décimaux et certaines expressions scientifiques comme le *e* pour désigner une exponentielle par exemple;

Le type str ou chaine de caractères

Les chaînes de caractères : opérations

Longueur :

```
s = "abcde"
len(s) # 5
```

Concaténation :

```
s1 = "abc"
s2 = "defg"
s3 = s1 + s2 # 'abcdefg'
```

Répétition :

```
s4 = "Fi!"
s5 = s4 * 3 # 'Fi! Fi! Fi!''
print(s5)
```

```
Fi!Fi!Fi!
```

Application 2

Comment corriger les instructions suivantes pour qu'elles ne provoquent plus d'erreur ?

- but recherché : on veut obtenir la chaîne de caractères "vous êtes 2 dans ce binôme TP"
- instruction : "vous êtes" + 2 + "dans ce binôme TP "
- instruction corrigée : "vous êtes" + " 2 " + "dans ce binôme TP
- but recherché : on veut afficher "blablablablablablablablablabla" c'est à dire"bla" 10 fois de suite.
- instruction : "bla"*10.0
- instruction corrigée : "bla"*10

Application 3

- Écrire un algorithme pour échanger les valeurs de 2 variables que vous nommerez var1 et var2.

```
a=8
b=9
a,b=b,a
print(a,b)
```

```
9 8
```

Le type de valeurs bool ou booléen

- Le type de valeur booléen est un type qui ne contient que deux valeurs qui servent à représenter deux états.
- Les deux valeurs sont True (vrai) et False (faux).

```
2 > 8 # False
2 <= 8 < 15 # True
```

- Opérateurs logiques: not, or et and

```
(3 == 3) or (9 > 24) # True (dès le premier membre)
(9 > 24) and (3 == 3) # False (dès le premier membre)
not(3 == 3) or (9 > 24) # false
```

Le type de valeurs bool ou booléen

- L'opérateur and a une précédence sur or

```
True or False and False
```

```
True
```

```
True or(False and False)
```

```
True
```

- L'opérateur not a une précédence sur or et and

```
not True or True
```

```
True
```

```
not (True or True)
```

```
False
```

Application 4

Quel résultat donne le code suivant ?

```
a = 2 # Integer
b = 1.99 # Floating
c = '2' # String
print(a>b)
print(a==c)
print( (a>b) and (a==c) )
print( (a>b) or (a==c) )
```

Le programme affiche:

```
True
False
False
True
```

Les Règles de nommage

- ne peut pas commencer par un nombre

- Il y a quelques mots réservés :

and, assert, break, class, continue, def, del, elif, else, except, exec, finally, for, from, global, if, import, in, is, lambda, not, or, pass, print, raise, return, try, while

Pour utiliser certaines de ces fonctions, vous devez ajouter à votre programme python le code suivant :

```
from math import * #ou from cmath import *
```

Command name	Description
`abs(`***value***`)`	absolute value
`ceil(`***value***`)`	rounds up
`cos(`***value***`)`	cosine, in radians
`floor(`***value***`)`	rounds down
`log(`***value***`)`	logarithm, base *e*
`log10(`***value***`)`	logarithm, base 10
`max(`***value1***`, `***value2***`)`	larger of two values
`min(`***value1***`, `***value2***`)`	smaller of two values
`round(`***value***`)`	nearest whole number
`sin(`***value***`)`	sine, in radians
`sqrt(`***value***`)`	square root

Constant	Description
`e`	2.7182818...
`pi`	3.1415926...

Les structures de contrôle Python

Les structures conditionnelles simples :

```
if expression 1:
    bloc d'insructions 1
else:
    bloc d'insructions 4
```

Exercice 1:

Ecrire un programme qui demande d'introduire la moyenne de l'utilisateur et affiche s'il a de moyenne ou s'il est sous la moyenne

Solution

```
chaine = input("Note sur 20 : ")
note = float(chaine)
if note>=10.0:
    print("J'ai la moyenne")
else:
    print("C'est en dessous de la moyenne")
print("Fin du programme")
```

Deux niveaux de test

```
if condition 1:
    if condition1_1:
        bloc_instruction1
else:
    bloc_instruction2
```

Exercice 2:

Ecrire un programme qui demande d'introduire la moyenne de l'utilisateur et affiche la mention:

- Bien si supérieur à 14
- Passable si entre 10 et 14
- Sous la moyenne si inférieur à 10

Solution

```
chaine = input("Note sur 20 : ")
note = float(chaine)
if note>=10.0:
    if note>=13:
        print("mention Bien")
    else:
        print("mention Passable")
else:
    print("C'est en dessous de la moyenne")
print("Fin du programme")
```

If/elif/else

```
if expression 1:
    bloc d'insructions 1
elif expression 2:
    bloc d'insructions 2
elif expression 3:
    bloc d'insructions 3
else:
    bloc d'insructions 4
#suite du programme
```

Exercice

Écrire un script en Python qui fait office de **Simulateur de Mention au Bac**. Il est simplement demandé à l'utilisateur pressé de faire un pari sur la moyenne générale qu'il espère et le simulateur lui retourne le message suivant en fonction de la note saisie :

Moyenne est comprise entre **[0;10[**: Allez, bon courage pour le rattrapage.

Moyenne est comprise entre **[10;12[**: *Bravo! Vous êtes admis(e).*

Moyenne est comprise entre **[12;14[**: *Bravo! Vous êtes admis(e) avec la **Mention Assez Bien**.*

Moyenne est comprise entre **[14;16[**: *Bravo! Vous êtes admis(e) avec la **Mention Bien.***

Moyenne est comprise entre **[16;20]:** *Bravo! Vous êtes admis(e) avec la **Mention Très Bien**.*

Moyenne est supérieure à **20**: *Bravo! **Vous êtes surdoué(e) inclassable**! On ne se sait que faire de vous!*

Solution

```
moyenne = float(input("Donnez la moyenne du candidat"))
if 0<=moyenne<10 :
    print("Allez, bon courage pour le rattrapage")
elif 10<=moyenne<12 :
    print("Bravo! Vous êtes admis(e)")
elif 12<=moyenne<14 :
    print("Bravo! Vous êtes admis(e) avec la Mention Assez Bien")
elif 14<=moyenne<16 :
    print("Bravo! Vous êtes admis(e) avec la Mention Bien")
elif 16<=moyenne<=20 :
    print("Bravo! Vous êtes admis(e) avec la Mention Très Bien")
else:
    print("Bravo! Vous êtes surdoué(e) inclassable! On ne se sait que faire de vous!")
```

Les structures itératives

1. La boucle « for»
2. La boucle « while »
3. Utilisation avancée des boucles

Structures itératives: for

```
for element in iterable:
    instructions
```

- Grâce à la fonction ***range([début],fin,[pas])*** la boucle for peut parcourir un intervalle de début à la fin-1 ou en définissant la bor,e initiale
- Elle peut également parcourir d'autres objets altérables (listes, chaines, tuples, ensembles, dictionnaires, tableaux…).

Exemple

Exemple 1

```
for i in range(5):
    print(i)
print("fin de la boucle")
```

```
0
1
2
3
4
fin de la boucle
```

Exemple 2

```
for i in range(1,11):
    print(i)
print("fin de la boucle")
```

```
1
2
3
4
5
6
7
8
9
10
fin de la boucle
```

Exemple 3

```
for i in range(1,8,2):
    print(i**2)
print("fin de la boucle")
```

```
1
9
25
49
fin de la boucle
```

Exercice

Écrire un programme en Python pour calculer:

a) 1+2+3+….+1001+2+3+….+100

```
s=0
for i in range(1,101):
    s=s+i
print(s)
```

```
5050
```

b) 1+3+5+….+99

```
s=0
for i in range(1,100):
    s=s+i
print(s)
```

```
4950
```

Structures itératives: while

```
Compteur = 1
while compteur<5:
    print(compteur, compteur<5)
    compteur += 1
print(compteur<5)
print("Fin de la boucle")
```

```
>>>
1 True
2 True
3 True
4 True
False
Fin de la boucle
```

Exercice :Vérifier si un nombre est correct (1)

Description:

Écrire un programme qui demande à l'utilisateur des nombres jusqu'à ce que le nombre soit entre 1 et 3 inclus. Lorsque l'utilisateur entre un nombre correct, le programme affiche "Bravo" et s'arrête.

Exemple d'exécution:

Entrez un nombre: 5

Entrez un nombre: 8

Entrez un nombre: -3

Entrez un nombre: 1

Bravo !

Exercice Correction

a=int(input("entrez un chiffre entre 1 et 3 inclu: "))

while not (a<=3 and a>=1) :

    a=int(input("entrez un chiffre entre 1 et 3 inclu: "))

print("Bravo")

```
a=int(input("entrez un chiffre entre 1 et 3 inclu: "))
while not (a<=3 and a>=1) :
    a=int(input("entrez un chiffre entre 1 et 3 inclu: "))
print("Bravo")
```

Utilisation avancée des boucles

❑ ***Instruction break***

L'instruction break provoque une sortie immédiate de la boucle *for* ou d'une boucle *while*

Exemple:

```
somme=0
while True:
    n=int(input("entrez un nombre 0 pour arrêter"))
    if n==0:
        break
    somme=somme+n
print("la somme des nombre est",somme)
```

- Dans cet exemple, l'expression True est toujours vraie: on a une boucle sans fin.
- L'instruction break est donc le sel moyen de sortir de la boucle

Utilisation avancée des boucles

❑ ***Instruction continue***

Permet de passer immédiatement à l'itération suivante de la boucle for ou while

Exemple:

```
for x in range(1,11):
    if x==5:
        continue
    print(x,end="")
```

```
1234678910
```

❖ ***la boucle a sauté la valeur 5***

Les fonctions

- Les fonctions sont très utiles pour réaliser plusieurs fois la même opération au sein d'un programme.
- Elles permettent également de rendre le code plus lisible et plus clair en le fractionnant en blocs logiques.
- Il existe des fonctions prédéfinies. Certaines nécessitant le chargement d'un module par exemple math.cos(angle)
- Pour définir une fonction, Python utilise le mot-clé **def** et si on veut que celle-ci renvoie une valeur, il faut utiliser le mot-clé **return**. Par exemple :

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

```
carre(12)
```

```
144
```

```
def hello():
    print("bonjour")
```

```
hello()
```

```
bonjour
```

Applications

1. Écrire une fonction qui renvoie toujours 1.
2. Ecrire une fonction puissance_3(x) qui renvoie le nombre x à la puissance 3, du même type que x.
3. Écrire une fonction qui renvoie cos(x) si x>0, et sin(x) sinon.
4. Écrire une fonction qui calcule les racines d'un polynôme du type

$$ax^2 + bx + c$$

Correction

1.
```
def un(x):
    return 1
```

2.
```
def puissance_3(x):
    return x**3
```

3.
```
from math import cos,sin
def cos_ou_sin(x):
    if x>0:
        return cos(x)
    else:
        return sin(x)
```

4.
```
from math import sqrt
def racines(a,b,c):
    delta=(b**2)-(4*a*c)
    if delta>0:
        x1=(-b-sqrt(delta))/(2*a)
        x2=(-b+sqrt(delta))/(2*a)
        return (x1,x2)
    if delta==0:
        x = -b/(2*a)
        return(x)
    else:
        return None
```

Les conteneurs standard

De façon générale, un conteneur est un objet composite destiné à contenir d'autres objets.

- ***Listes***

  est une collection ordonnée et modifiable d'éléments éventuellement hétérogènes.

Une liste est représentée entre crochets, et ses éléments sont séparés par des virgules:

```
[1,10,2,3,2,2]
```

Une liste peut contenir des objets de n'importe-quel type. Par exemple, des chaînes de caractères:

```
['a','b','toto']
```

On peut aussi mélanger des objets de types différents dans la même liste:

```
[1,'a',2,3,3.14159,True,False,1]
```

Exemple

On peut même avoir une liste qui contient une liste:

```
[1,[2,3],'toto']
```

Une liste peut être vide:

```
[]
```

Les éléments d'une liste peuvent contenir des expressions complexes, faisant appel à des variables ou des fonctions:

```
a=2
[(2+3)*a,a*2]
```

On accède aux éléments individuels d'une liste en indiquant leur indice entre crochets. La numérotation des éléments commence à zéro. Par exemple:

```
>>>a=[1,7,5]
>>>a[0]
1
>>>a[2]
5
```

On peut utiliser des indices négatifs pour compter à partir de la fin de la liste:

```
>>>a=[1,7,5]
>>>a[-1]
5
>>>a[-3]
1
```

La fonction len donne la longueur d'une liste:

```
>>>liste=[3,7,6]
>>>len(liste)
3
```

On peut extraire un sous-ensemble des éléments d'une liste en spécifiant un intervalle d'indices de la façon suivate: liste[min:max]. Les indices conservés sont ceux qui sont supérieurs ou égaux à min, et inférieurs strictement à max:

```
>>>liste=[1,4,1,5,9]
>>>liste[1:3]
[4, 1]
>>>liste[2:2]
[]
```

min ou max peut être omis. Dans ce cas, tout se passe comme si min était 0, et max était len(liste):

```
>>>liste[2:]
[1, 5, 9]
>>>liste[:2]
[1, 4]
>>>liste[:]
[1, 4, 1, 5, 9]
```

Les indices négatifs peuvent être utilisés dans les intervalles aussi:

```
>>>liste[:-1]
[1, 4, 1, 5]
```

Opérateurs sur les listes

De la même manière que les chaînes, les listes peuvent être concaténées avec l'opérateur `+`, et multipliées par un entier:

```
>>>[1,2]+[2]
[1, 2, 2]
>>>[1,2]*3
[1, 2, 1, 2, 1, 2]
```

On peut comparer deux listes avec l'opérateur `==`:

```
>>>[1,2]==[2,3]
False
>>>[1,2]==[1,2]
True
```

Modification d'une liste

On peut affecter une valeur à un élément d'une liste:

```
>>>a=[1,7,5]
>>>a[1]=0
>>>a
[1, 0, 5]
```

On peut aussi affecter une liste à un intervalle:

```
>>>a=[1,7,5]
>>>a[0:2]=[2,3]
>>>a
[2, 3, 5]
```

Lors d'une affectation à un intervalle, il n'est pas nécessaire que les listes soient de longueur identique:

```
>>>a=[1,7,5]
>>>a[0:2]=[2,3,4]
>>>a
[2, 3, 4, 5]
>>>a[0:2]=[]
>>>a
[4, 5]
```

La méthode append permet d'ajouter un élément à la fin de la liste:

```
>>>a=[1,4,5]
>>>a.append(0)
>>>a
[1, 4, 5, 0]
```

Sémantique de pointeur

L'affectation d'une liste à une variable ne crée pas une copie de la liste. Voici un exemple de programme qui illustre ce phénomène:

```
>>>a=[1,7,5]
>>>b=a
>>>a[1]=0
>>>b
[1, 0, 5]
```

Pour faire une nouvelle copie de la liste, il est nécessaire d'utiliser une expression comme ci-dessous:

```
>>>a=[1,7,5]
>>>b=a[:]# b = a + [], ou b = list(a) fonctionnent aussi
>>>a[1]=0
>>>b
[1, 7, 5]
```

Autres fonctions

On peut trier une liste sur place avec la méthode sort:

```
>>>a=[1,5,3,2]
>>>a.sort()
>>>print(a)
[1, 2, 3, 5]
```

Si on souhaite conserver la liste a intacte, et créer une nouvelle liste, on peut utiliser la fonction sorted:

```
>>>a=[1,5,3,2]
>>>b=sorted(a)
>>>print(b)
[1, 2, 3, 5]
```

Autres fonctions

```
>>>print(a)
[1, 5, 3, 2]
>>>c=sorted(a, reverse=True)
>>>print(c)
[5,3,2,1]
```

L'appartenance à une liste peut se tester avec l'opérateur in:

```
>>>1 in[3,4,5]
False
>>>4 in[3,4,5]
True
```

Le minimum et le maximum peuvent se trouver avec les fonctions min et max:

```
>>>min([3,1,4,1,5,9,2])
1
>>>max([3,1,4,1,5,9,2])
9
```

Autres fonctions

del permet de supprimer un élément d'une liste:

```
>>>a=[0,1,2,3,4]
>>>del a[2]
>>>a
[0, 1, 3, 4]
```

Notez qu'on peut faire la même opération en affectant une liste vide:

```
>>>a=[0,1,2,3,4]
>>>a[2:3]=[]
>>>a
[0, 1, 3, 4]
```

Application 1

Écrire un programme qui créé une liste d'au moins 5 entiers puis successivement:

1. affiche la valeur de L[3]

2. modifie la liste en remplaçant L[2] par 5 et L[3] par la somme des cases voisines L[2] et L[4]

Application 2

Définir la liste : liste = [17, 38, 10, 25, 72], puis effectuez les actions suivantes :

– triez et affichez la liste

– ajoutez l'élément 12 à la liste et affichez la liste

– renversez et affichez la liste

– affichez l'indice de l'élément 17

– enlevez l'élément 38 et affichez la liste

– affichez la sous-liste du 2eau 3eélément

– affichez la sous-liste du début au 2eélément

– affichez la sous-liste du 3eélément à la fin de la liste

– affichez la sous-liste complète de la liste

– affichez le dernier élément en utilisant un indexage négatif

Application 3

1. Initialisez T comme une liste vide, et M comme une liste de cinq flottants nuls.

2. Affichez ces listes.

3. Utilisez la fonction range() pour afficher :

– les entiers de 0 à 3 ;

– les entiers de 4 à 7 ;

– les entiers de 2 à 8 par pas de 2.

Définir L comme une liste des entiers de 0 à 5 et testez l'appartenance des éléments 3 et 6 à L.

Tuple

- Un tuple est une liste qui ne peut plus être modifiée.
- Éléments séparés par des virgules:

1. ```
   mon_tuple = (1, "ok", "olivier")
   ```
2. ```
   mon_tuple = 1, "ok", "olivier"
   ```

- Lorsque vous créez un tuple avec une seule valeur, n'oubliez pas d'y ajouter une virgule, sinon ce n'est pas un tuple

```
mon_tuple = ("ok",)
```

Afficher une valeur d'un tuple

- Le tuple est une sorte de liste, on peut donc utiliser la même syntaxe pour lire les données du tuple.

```
mon_tuple=10,2,3,4
print(mon_tuple[2])
```

- Les tuples ne sont pas **modifiables** (ou **mutable**), cela signifie qu'il est impossible de modifier un de leurs éléments. Par conséquent, la ligne d'affectation suivante n'est pas correcte :

```
mon_tuple=10,2,3,4
mon_tuple[1] = "ok"
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-44-281a95c38040> in <module>
      1 mon_tuple=10,2,3,4
----> 2 mon_tuple[1] = "ok"

TypeError: 'tuple' object does not support item assignment
```

A quoi sert un tuple alors?

- Le tuple permet une affectation multiple:

```
(v1, v2,v3 )= (11, 22,"a")
print(v1)
print(v2)
print(v3)
```

```
11
22
a
```

- Il permet également de renvoyer plusieurs valeurs lors d'un appel d'une fonction:

```
def donne_moi_ton_nom():
    return "olivier", "engel"
```

```
donne_moi_ton_nom()
```

```
('olivier', 'engel')
```

- On utilisera un tuple pour définir des sortes de constantes qui n'ont donc pas vocation à changer.

Application 1

Définir un tuple: T = (17, 38, 10, 25, 72), puis effectuez les actions suivantes :

– affichez l'indice de l'élément 17

– affichez le tuple du 2e au 3eélément

– affichez le tuple du début au 2e élément

– affichez le tuple du 3e élément à la fin du tuple

– affichez le tuple complète du tuple

– testez l'appartenance des éléments 10 et 6 à T

Application 2

Ecrivez les résultats obtenus:

1.
```
t=("a","b","c","d","e")
t[0]
```

2.
```
t=("a","b","c","d","e")
t[1:3]
```

3.
```
t=("a","b","c","d","e")
t[0]="A"
```

4.
```
t=("A",)+t[1:]
print(t)
```

5.
```
(0,1,2)<(0,3,4)
```

6.
```
(0,1,20000)<(0,3,4)
```

Ensembles (set)

- Les ensembles ont la particularité d'être non modifiables, non ordonnés et de ne contenir qu'une seule copie maximum de chaque élément.
- Pour créer un nouveau *set* on peut utiliser les accolades : `s = {1, 2, 3, 3}`
- Créer un ensemble vide: `S=set()`
- Il est donc impossible de récupérer un élément par sa position. Il est également impossible de modifier un de ses éléments.

```
s = {1, 2, 3, 3}
s[1]
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-87-20b175a9200a> in <module>
      1 s = {1, 2, 3, 3}
----> 2 s[1]

TypeError: 'set' object is not subscriptable
```

Ensembles (set)

En général, on utilisera la fonction interne à Python ***set()*** pour générer un nouveau *set*. Celle-ci prend en argument n'importe quel objet itérable et le convertit en *set*

1. Liste:

```
set([1, 2, 4, 1])
```

```
{1, 2, 4}
```

2. Tuple:

```
set((2, 2, 2, 1))
```

```
{1, 2}
```

3. Chaine de caractère:

```
set("ESB2021")
```

```
{'0', '1', '2', 'B', 'E', 'S'}
```

=> pas de duplication

Ensembles (set)

- L'appartenance à un ensemble peut se tester avec l'opérateur ***in:***

```
S={"a","b","c"}
"a" in S
```

```
True
```

```
S={"a","b","c"}
"Z" in S
```

```
False
```

- Ajouter un seul élément à un ensemble: ***add():***

```
S={"a","b","c"}
S.add("e")
print(S)
```

```
{'c', 'a', 'e', 'b'}
```

- Ajouter plusieurs éléments à un ensemble: ***update():***

```
S={"a","b","c"}
S.update({"o","e","p"})
print(S)
```

```
{'c', 'a', 'p', 'e', 'o', 'b'}
```

Ensembles (set)

- Supprimer un élément donné d'un ensemble ***discard():***

```
S={"a","b","c"}
S.discard("a")
print(S)
```

```
{'c', 'b'}
```

- Supprimer un ensemble ***clear() :***

```
S={"a","b","c"}
S.clear()
print(S)
```

```
set()
```

- Créer une copie d'un ensemble ***copy():***

```
S1={"a","b","c"}
S2=S1.copy()
print(S2)
```

Ensembles (set)

```
X={1,7,8,9}
Y={8,6,3}
X-Y #ensemble des éléments de X qui ne sont paas dans Y {1, 9, 7}
Y-X #ensemble des éléments de Y qui ne sont paas dans X {3, 6}
X^Y #ensemble des éléments qui sont soit dans X soit dans Y {1, 3, 6, 7, 9}
X|Y #Union {1, 3, 6, 7, 8, 9}
X&Y #intersection {8}
```

Application

1. Écrire un programme Python pour créer un nouveau ensemble vide.

2. Écrire un programme Python pour ajouter un ou plusieurs éléments à un ensemble.

3. Écrire un programme Python pour supprimer un élément donné d'un ensemble.

4. Écrire un programme Python pour supprimer un ensemble.

5. Ecrire un programme Python pour trouver la longueur d'un ensemble.

6. Ecrire un programme Python pour créer une copie d'un ensemble.

7. Écrire un programme Python pour afficher l'intersection des ensembles.

8. Ecrire un programme Python pour trouver la valeur maximale et minimale d'un ensemble.

9. Ecrire un programme Python pour trouver la longueur d'un ensemble.

Dictionnaire

- Collection de couples clé : valeur entourée d'accolades

```
dict={'computer': 'ordinateur', 'mouse': 'souris', 'keyboard': 'clavier'}
```

- Dictionnaire vide:

```
d1= {}
d2=dict()
```

```
dico['computer'] = 'ordinateur'
dico['mouse'] ='souris'
dico['keyboard'] ='clavier'
print (dico)
```

```
{'computer': 'ordinateur', 'mouse': 'souris', 'keyboard': 'clavier'}
```

Dictionnaire

- La fonction ***len()*** est utilisable avec un dictionnaire

```
dict={'computer': 'ordinateur', 'mouse': 'souris', 'keyboard': 'clavier'}
del dict["computer"]
print(dict)
```

```
{'mouse': 'souris', 'keyboard': 'clavier'}
```

- La méthode ***keys()*** renvoie la liste des clés utilisées dans le dictionnaire

```
print(dict.keys())
```

```
dict_keys(['computer', 'mouse', 'keyboard'])
```

- La méthode ***values()*** renvoie la liste des valeurs mémorisées dans le dictionnaire

```
print(dict.values())
```

```
dict_values(['ordinateur', 'souris', 'clavier'])
```

Application

- Choisissez 5 mots de la langue française et créez un dictionnaire qui associe à chacun de ces mots sa traduction en anglais.
- Affichez la liste des valeurs
- Affichez la liste des clés
- Supprimer le premier élément du dictionnaire