

PROGRAMMATION EN PYTHON CHAINES DE CARACTERES & DICTIONNAIRE

Fatiha BENDAIDA

DWFS 1

2024/2025

CHAINE DE CARACTÈRES

○ Définition :

↪ une **chaîne de caractères** est une succession de caractères délimités par des guillemets (simples ou doubles)

○ Création :

```
>>> s1=str() | s1= ""          # chaîne vide  
>>> s2=str("Hello") | s2= "Hello"  
>>> s3=str(123)| s3= '123'
```



CHAINE DE CARACTÈRES

○ *Caractères spéciaux*

\n	permet d'insérer des marques de passage à la ligne
\t	permet d'insérer des marques de tabulation
\b	Retour arrière (suppression du caractère avant)

N.B : Même s'ils sont écrits avec deux symboles, les caractères spéciaux comptent pour un seul caractère.



CHAINE DE CARACTÈRES

○ Comparaison de deux chaines de caractères :

```
>>> 'green' == 'glow'
```

False

```
>>> 'green' != 'Green'
```

True

```
>>> 'green' >= 'glow'
```

True

```
>>> 'green' < 'Green'
```

False

```
>>> "aaaaaaaa" < "aabc"
```

True



CHAINE DE CARACTÈRES

- **Accès aux valeurs d'une chaîne de caractères :**

Etant un cas particulier des listes, nous allons réutiliser les indices pour manipuler les chaînes de caractères.

H	E	L	L	O
0	1	2	3	4
-5	-4	-3	-2	-1

N.B : Les chaîne de caractère support le slicing

```
>>> chaine="HELLO"  
>>> print(chaine[1:3]+chaine[-1])  
'ELO'
```

CHAINE DE CARACTÈRES

- Chaines de caractères sont **immuables** :

En python, les chaines de caractères **ne peuvent pas subir des modifications** de la même manière que les liste.

```
>>> chaine='HELLO'
```

```
>>> chaine[0]='P'
```

Error

- Mais, on peut créer des variations d'une chaine de caractères en créant de nouvelles chaines de caractères.

```
>>> A='MELLO'
```

```
>>> A='H'+A[1:]
```

```
>>> print(A)
```

HELLO

CHAÎNE DE CARACTÈRES

Les opérateurs de chaîne de caractères :

- Concaténation :

```
>>> chaine='DTS'+ 'DWFS'  
>>> chaine  
'DTSDWFS'
```

- Répétition d'une chaîne

```
>>> A='BLA' *5  
>>> A  
'BLABLABLABLABLA'
```

- Boucle à travers une chaîne

```
>>> for x in "banana":  
    print(x)  
'b'
```

Chaine de caractères

Fonctions utiles sur les chaînes de caractères:

❑ Taille (Longueur) :

```
>>> len('Bonjour')
```

```
7
```

❑ Conversion des nombre en texte

```
>>> str(1234)
```

```
'1234'
```

❑ Evaluation des chaînes :

```
>>> x=3
```

```
>>> eval('x+5')
```

```
8
```


Chaine de caractères

Méthodes utiles sur les chaînes de caractères:

❑ Majuscule :

```
>>> A='Bonjour idsd1'  
>>> A.upper()  
'BONJOUR DWFS1'
```

❑ Minuscule

```
>>> A.lower()  
'bonjour dwfs1'
```

❑ Rechercher l'index d'un caractère

```
>>> A.find('u')  
5  
>>> A.find('a') # la méthode find renvoie -1 si le caractère n'existe pas  
-1  
>>> A.find('U')  
-1
```

Chaine de caractères

Méthodes utiles sur les chaînes de caractères:

❑ Rechercher l'index d'un caractère (**index**)

Même fonctionnement que find sauf si le caractère n'est pas présent, on a une erreur.

```
>>> A.index('u')
```

```
5
```

```
>>> A.index('x')
```

```
ValueError: substring not found
```

❑ Partitionner en fonction du séparateur.

```
>>> A='bonjour dwfs1'
```

```
>>> A.split(' ')
```

```
['Bonjour', 'dwfs 1']
```

```
>>> A.split('o')
```

```
['B','nj','ur dwfs 1']
```

Chaine de caractères

Méthodes utiles sur les chaînes de caractères:

❑ Récupérer une chaîne en fonction du séparateur.

```
>>> L=['ab', 'racada', 'bra']  
>>> B=''.join(L)  
>>> B  
'abracadabra'  
>>> C='#'.join(L)  
>>> C  
'ab#racada#bra'
```

❑ Remplacer une sous chaîne :

```
>>> A='bonjour dwfs 1'  
>>> A.replace('jour','soir')  
'Bonsoir dwfs 1'
```

❑Autres Méthodes

- **s.lstrip() / s.rstrip() / s.strip()** -- returns a string with whitespace removed from the start/end/ both.
- **s.isalpha()/s.isdigit()/s.isspace()...** -- tests if all the string chars are in the various character classes
- **s.startswith('other'), s.endswith('other')** -- tests if the string starts or ends with the given other string

Chaine de caractères

Exercice 1:

Ecrire une fonction **supp_espace(ch)** qui reçoit en argument une chaîne de caractères **ch**, la fonction retourne la chaîne après avoir supprimé tous les espaces s'ils existent.

Exemple: **ch=' bonjour ca va '** alors **supp_espace(ch)** renvoie : **'bonjourcava'**

```
def supp_espace(ch):  
    u=''  
    for x in ch:  
        if x!=' ':  
            u+=x  
    return u
```



Chaine de caractères

Exercice 2:

Ecrire une fonction **supp_char(ch , c)** qui reçoit en argument une chaine de caractères **ch** et un caractère **c**, la fonction retourne la chaine après avoir supprimé tous les occurrences de **c** s'ils existent.

Exemple: **ch=' bonjour ca va '** alors **supp_char(ch,' ')** renvoie : **'bonjourcava'**

```
def supp_char(ch, c):  
    while(c in ch):  
        ind=ch.find(c)  
        ch=ch[:ind]+ch[ind+1:]  
    return ch
```



Exercice 3:

Ecrire une fonction **isAnagram(CH, SH)**, prenant deux chaînes de caractères en argument, retourne vrai si les deux chaînes sont de même taille et se composent de mêmes lettres.

Exemple :

```
>>> isAnagram("DUT1", "1TUD")
```

```
True
```

```
>>> isAnagram("abcc" , "abcd")
```

```
False
```

```
>>> isAnagram("abc" , "abcd")
```

```
False
```

```
def isAnagramme(CH1,CH2):  
    n,m=len(CH1),len(CH2)  
    if n!=m:  
        return False  
    L,M=list(CH1),list(CH2)  
    for e in CH1:  
        if L.count(e)!=M.count(e):  
            return False  
    return True
```

```
def isAnagramme(CH,SH):  
    n,m=len(CH),len(SH)  
    if n!=m:  
        return False  
    L=[SH[i] in CH and CH[i] in SH for i in range(n)]  
    return False not in L
```



DICTIONNAIRE :



○ Définition et propriétés :

- ↪ Un dictionnaire est une structure de données qui permet de stocker des éléments sous forme **clé : valeur**
- ↪ Contrairement aux listes qui sont délimitées par des crochets, on utilise des **accolades** pour les **dictionnaires**.
- ↪ La **clé** est généralement un objet non **mutable (non modifiable)**
- ↪ La **valeur** peut être de n'importe quel type
- ↪ Les clés sont toutes **distinctes**
- ↪ **Avantages** : Les dictionnaires nous permettent de stocker des valeurs en utilisant **nos propres indices**.



DICTIONNAIRE :

Exemples :

```
>>> stock = {'carotte' : 51, 'pommes' : 243 }  
>>> stock  
{'pommes': 243, 'carotte': 51}  # aucun ordre !
```

```
>>> len(stock)  
2
```

```
>>> 'pommes' in stock  
True
```

```
>>> stock['pommes']  
243  
>>> stock['orange']  
KeyError
```



DICTIONNAIRE :

○ Qu'est-ce qu'un dictionnaire ?

↪ Le dictionnaire vide se note { } ou dict().

```
>>> d1={}
>>> d2=dict()           # d1 et d2 sont des dictionnaires vides
```

↪ On peut **modifier** la valeur associée à une clé, ou **rajouter** un nouveau couple clé/valeur

```
>>> stock = {'carotte' : 51, 'pommes' : 243 }
>>> stock['carotte']=101
>>> stock
{'carotte' : 101, 'pommes' : 243 }
```

```
>>> stock['bananes']=300
>>> stock
{'carotte' : 101, 'pommes' : 243, 'bananes' : 300 }
```



DICTIONNAIRE :

○ Suppression d'une valeur ?

```
>>> D= {"name": "Alice", "age": 19, "class": "DWFS1"}  
>>> del D['age']          # supprimer la paire de cle 'age'  
>>> D  
{"name": "Alice", "class": "DWFS1"}
```

○ Vider le dictionnaire

```
>>> D.clear()  
>>> D  
{}
```

○ Supprimer le dictionnaire tout entier

```
>>> del D
```



DICTIONNAIRE :

○ Operations de bases sur les dictionnaires

$D = \{ 'N451236': 'Karim Issam' , 'E985477': 'Amal Abbassi' \}$

Taille du dictionnaire

`len(D)`

2

Teste d'appartenance
d'une clé

`'C451236' in D`

False

Afficher le dictionnaire

`print(D)`

`{'N451236': 'K..ssi'}`

Itération par boucle for

`for k in D:
print(D[k])`

`'Karim Issam'
'Amal Abbassi'`

Itération par Liste

`L=list(D)
for e in L:
Print(D[e])`

`'Karim Issam'
'Amal Abbassi'`

DICTIONNAIRE :



○ Manipulation des dictionnaires (méthodes)?

↪ Python dispose de plusieurs méthodes permettant la manipulation des dictionnaires:

```
D = {'nom': 'Karim' , 'prenom': 'Omar', 'age': 20, 'classe': 'IDSD1'}
```

D.copy()

Retourne une copie du dictionnaire 'D'

D.fromkeys(keys, value)

Crée et retourne un dictionnaire à partir de la liste keys et la valeur 'value'

D.get(key, default)

Return la valeur de la clé key si ça existe et la valeur **default sinon.**

D.items()

Retourne la liste de paires (clé, valeur)

D.keys()

Retourne la liste des clés du dictionnaire 'D'

D.values()

Retourne la liste de valeurs stockées dans le dictionnaire 'D'

D.update(D2)

Met à jour le dictionnaire 'D' par le dictionnaire 'D2'



DICTIONNAIRE :

○ Exercice 2 :

On considère un dictionnaire «service» déclarée d'une manière globale comme suit et qui représente l'âge et les jours de travail de chaque employé :

```
service = {  
    'Ahmed': [ 30 , ["Lu","Ma","Me","Je"] ],  
    'Rachid' : [ 25, ["Lu","Me","Di"] ],  
    'Omar' : [ 55, ["Lu","Me","Ve"] ],  
    ...  
}
```

1. Ecrire une fonction **employes_par_age(age)** qui renvoie la liste des noms des employés âgé plus que l'âge passé en paramètre.





DICTIONNAIRE :

```
def  employes_par_age(age):  
    M=[]  
    for x in service:  
        if service[x][0]>=age :  
            M.append(x)  
    return M
```

2.Ecrire une fonction **employes_par_jour(jour)** qui renvoie la liste des noms des employés qui travaillent le jour passé en paramètre.



DICTIONNAIRE



```
def employes_par_jour(jour):  
    M=[]  
    for e in service:  
        if jour in service[e][1]:  
            M.append(e)  
    return M
```

3. Ecrire une fonction **employes_bosseur()** qui renvoie la liste des noms des employés qui travaillent le plus grand nombre de jours.

```
def employes_bosseur():  
    d={}  
    for e in service:  
        d[e]=len(service[e][1])  
    m=max(list(d.values()))  
    bos=[e for e in d if d[e]==m]  
    return bos
```

