Chapitre 8:
Les Listes ou tableau ,Les Tuples et dictionnaires en Python

Pourquoi avons-nous besoin de listes?

Il peut arriver que vous deviez lire, stocker, traiter et enfin imprimer des dizaines, voire des centaines, voire des milliers de nombres. Et alors? Avez-vous besoin de créer une variable distincte pour chaque valeur? Aurez-vous à passer de longues heures à rédiger des déclarations comme celle ci-dessous?

```
var1 = int(input())
var2 = int(input())
var3 = int(input())
var4 = int(input())
var5 = int(input())
var6 = int(input())
.
.
.
```

Les listes

Jusqu'à présent, vous avez appris à déclarer des variables capables de stocker exactement une valeur donnée à la fois.

Pensez à la façon dont il serait commode de déclarer une variable qui pourrait stocker plus d'une valeur .

Nous allons vous montrer comment déclarer de telles variables à valeurs multiples, Les listes.

C’est un nouveau type de variable qui peut contenir des entiers, des flottants, des chaînes de caractères voire des listes. Il est naturellement intégré dans python.

Les listes

Une liste est une collection ordonnée d'objets python. De même que pour les chaînes de caractères, la fonction print() permet d'afficher la liste.

Voici un exemple de liste :

```
Liste1=[3,5,2] // une liste de trois elements
print(Liste1)
Liste2=[2,"bonjour",liste1,1.5] // une liste qui contient différents élément.
print(Liste2)
```

Listes d'indexation

numbers = [10, 5, 7, 2, 1]

print("le contenu de la liste originale :", numbers)

Execution ➔ 10 5 7 2 1

1- Nous allons attribuer une nouvelle valeur 111 au premier élément dans la liste.

Nous le faisons de cette façon:

numbers[0] = 111

print(" le contenu de la nouvelle liste : ", numbers)

Execution ➔ 111 5 7 2 1

2- Et maintenant, nous voulons que la valeur du cinquième élément soit copiée dans le deuxième élément:

numbers[1] = numbers[4]

print(" le contenu de la nouvelle liste : ", numbers)

Execution ➔ 111 1 7 2 1

La fonction len ()

La longueur d'une liste peut varier pendant l'exécution.

De nouveaux éléments peuvent être ajoutés à la liste, tandis que d'autres peuvent en être supprimés. Cela signifie que la liste est une entité très dynamique.

Si vous souhaitez vérifier la longueur actuelle de la liste, vous pouvez utiliser une fonction nommée len( )(son nom vient de la longueur).

La fonction prend le nom de la liste comme argument et retourne le nombre d'éléments actuellement stockés dans la liste

List== [10, 5, 7, 2, 1]

len(List) ➔5

Les opérateurs in et not in

Python propose deux opérateurs très puissants, capables de **parcourir la liste afin de vérifier si une valeur spécifique est stockée dans la liste ou non** . (retourne true ou false)

Ces opérateurs sont:

elem **in** myList

elem **not in** myList

Utilisation des listes

Pour **parcourir une liste**

```
L = [10, 1, 8, 3, 5]
S = 0
for i in range(len(L)):
    S=S+L[i]
print("la somme est ",S)
```

```
L= [10, 1, 8, 3, 5]
S = 0
for elt in L:
    S=S+elt
print("la somme est ",S)
```

Les listes

La fonction	utilisation
Len(list)	La longueur de la liste
list.append(x)	Ajouter x à la fin de la liste
list.insert(i, x)	Inserer l'element x dans la position i
list.remove(x)	Supprimer le premier element x de la liste
list.pop([i])	Supprimer l'element d'indice i
list.clear()	Supprimer tous les elements de la liste
list.count(x)	Retourne le nombre d'occurrence de x dans list

Les listes

La fonction	utilisation
list.reverse()	Inverser la liste sur place
list.copy()	Retourne un copie de la liste
Del list [i]	Supprimer le i eme élément de la liste

```
#la taille de la liste
list=[3,7,1,9,4]
print("la taille de la liste :",len(list))
#liste de taille 5
list=[]
for i in range(5):
    x= int(input("donner une valeur:"))
    list.append(x)
print(list)
#inserer un element x dans la position 3
x=int(input("donner une valeur:"))
list.insert(3,x)
print(list)
#supprimer une valeur
sup=int(input("donner une valeur:"))
list.remove(sup)
print(list)
```

```
#supprimer valeur dans la position 2
list.pop(2)
print(list)
#supprimer la dernière valeur
list.pop()
print(list)
#supprimer valeur dans la position 2
del list[2]
print(list)
#le nombre d'occurrence de la valeur x
x=int(input("donner une valeur:"))
print(list.count(x))
#inverser une liste
print(list.reverse())
#copier la liste l dans une nouvelle liste T
T=list.copy()
Print(T)
#supprimer la liste
list.clear()
```

Echange des éléments de la liste

Vous pouvez facilement échanger les éléments de la liste pour inverser leur ordre :

```
myList = [10, 1, 8, 3, 5]

myList[0], myList[4] = myList[4], myList[0]

myList[1], myList[3] = myList[3], myList[1]

print(myList)
```

Les indices négatifs sont légaux:

Cela peut sembler étrange, mais les indices négatifs sont légaux et peuvent être très utiles.

Un élément avec un indice égal à **-1 est le dernier de la liste**.

numbers = [111, 7, 21, 15]

print(numbers[-1]) ➔ 15

print(numbers[-2]) ➔ 21

Tranches puissantes

Heureusement, la solution est à portée de main - son nom est la tranche .

Une tranche est un élément de la syntaxe Python qui vous permet de faire une toute nouvelle copie d'une liste ou de parties d'une liste .

list[min : max : pas]

```
L = [1, 2, 3, 4, 5]

Lnew1 = L[2: ]

Lnew2 = L[ :2]

Lnew3 = L[-2: ]

print(Lnew1) # outputs: [3, 4, 5]

print(Lnew2) # outputs: [1, 2]

print(Lnew3) # outputs: [4, 5]
```

L'instruction del

N'importe quel élément de la liste peut être supprimé à tout moment - cela se fait avec une instruction nommée del (delete). Remarque: c'est une instruction, pas une fonction.

Vous devez pointer l'élément à supprimer - il disparaîtra de la liste et la longueur de la liste sera réduite d'une unité.

L'instruction del est capable de supprimer plus qu'un simple élément de liste à la fois - elle peut également supprimer des tranches :

```
myList = [10, 8, 6, 4, 2]

del myList[1:3]

print(myList)
```

La suppression de tous les éléments à la fois est également possible:

```
myList = [10, 8, 6, 4, 2]

del myList[:]

print(myList)
```

Tri d'une liste

Python a ses propres mécanismes de tri. Personne n'a besoin d'écrire leur propre type, car il existe un nombre suffisant d' outils prêts à l'emploi .

Si vous voulez que Python trie votre liste, vous pouvez le faire comme ceci:

```
myList = [8, 10, 6, 2, 4]
myList.sort()
print(myList)
```

La sortie de l'extrait est la suivante:

```
[2, 4, 6, 8, 10]
```

La vie intérieure des listes

Exemple:

```
list1 = [1]
list2 = list1
list1[0] = 2
print(list2)
```

L'affectation: list2 = list1 copie le nom du tableau, pas son contenu. En effet, les deux noms (list1et list2) identifient le même emplacement dans la mémoire de l'ordinateur. La modification de l'un d'eux affecte l'autre, et vice versa.

Comment gérez-vous cela?

La vie intérieure des listes

```
list1 = [1]

list2 = list1[:]  // copie de la liste entiere ou list2.copie(list1)

list1[0] = 2

print(list2)

myList = [10, 8, 6, 4, 2]

newList = myList[1:3]

print(newList)
```

Il copie en fait le contenu de la liste, pas le nom de la liste.

C'est exactement ce dont vous avez besoin.

```
list1 = [1]

list2 = list1[:]

list1[0] = 2

print(list2)
```

Manipulation des Listes

- l' +opérateur peut joindre des listes ensemble

  list2=myList+myList

- l'opérateur * peut multiplier les listes;

  list2= myList * 3

myList=[2,4,8,9]

print(3*myList) ➔[2, 4, 8, 9, 2, 4, 8, 9, 2, 4, 8, 9]

print(myList+ myList)➔[2, 4, 8, 9, 2, 4, 8, 9]

Remarque:

del myList[1:3]

print(myList)

compréhension de liste

C’est la syntaxe spéciale utilisée par Python pour remplir des listes massives.

Une compréhension de liste est en fait une liste, mais créée à la volée pendant l'exécution du programme, et n'est pas décrite statiquement .

Exemple 1:

for x in range(10): **squares.append(x**2)**	**squares = [x ** 2 for x in range(10)]**
L'extrait produit une liste de dix éléments remplie de carrés de dix nombres entiers à partir de zéro (0, 1, 4, 9, 16, 25, 36, 49, 64, 81)	

Exemple 2:

for i in range(8): **twos.append(2**i)**	**twos = [2 ** i for i in range(8)]**
L'extrait crée un tableau à huit éléments contenant les huit premières puissances de deux (1, 2, 4, 8, 16, 32, 64, 128)	

Listes dans les listes: tableau a deux dimensions

Liste = [[valeur for j in range(col)] for i in range(ligne)]

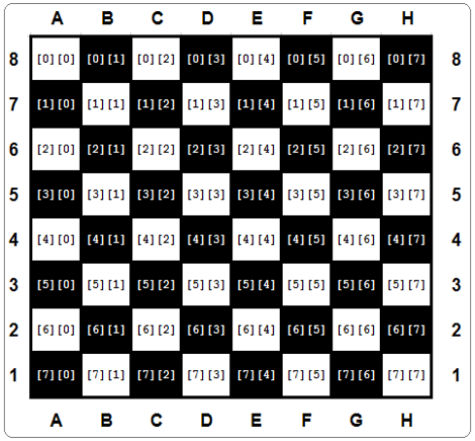
exemple

Liste = [[i ** j for j in range(3)] for i in range(4)]
print(Liste)

➔ [[1, 0, 0], [1, 1, 1], [1, 2, 4], [1, 3, 9]]

Pour accéder aux éléments :

Liste[i][j]



Liste comme paramètre de fonction

```
def list_sum(lst):
    s = 0
    for elem in lst:
        s += elem
    return s
print(list_sum([5, 4, 3]))
```

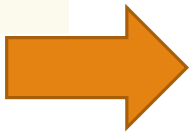
```
print(list_sum([5, 4, 3]))
```

12

Fonction retourne une liste

```
def strangeListFunction(n):
    strangeList = []
    for i in range(0, n):
        strangeList.insert(0, i)
    return strangeList


print(strangeListFunction(5))
```



```
[4, 3, 2, 1, 0]
```

Tuples en Python

La distinction la plus claire entre les listes et les tuples est la syntaxe utilisée pour les créer - les **tuples préfèrent utiliser des parenthèses** , tandis que les listes aiment voir les crochets, bien qu'il soit également **possible de créer un tuple uniquement à partir d'un ensemble de valeurs séparées par des virgules** .

Un tuple est une liste **non modifiables**

t1=(1,2,3,4) ou **t1=1,2,3,4**

t1.append(9)➔ erreur car le tuple est non modifiable

```
t1=(1,3,4,5)
l1=[1,3,4,5]
print(t1)
print(l1)
t1.append(9)
l1.append(9)
print(t1)
print(l1)
```

Exemple

myTuple = (1, 2, True, "a string", (3, 4), [5, 6], None)

print(myTuple)

La sortie:

(1, 2, True, 'a string', (3, 4), [5, 6], None)

Comment utiliser un tuple?

Si vous souhaitez obtenir les éléments d'un tuple afin de les relire, vous pouvez utiliser les mêmes conventions auxquelles vous êtes habitué lors de l'utilisation des listes.

```
myTuple = (1, 10, 100, 1000)
print(myTuple[0])
print(myTuple[-1])
print(myTuple[1:])
for elem in myTuple:
    print(elem)
```

```
1  tuple1=(2,4,5,7,19)
2  tuple2=(5,9,5)
3
4  # afficher l'élément 4 de tuple1
5  print("4éme élément : ",tuple1[3])
6
7  # afficher les 3 premiers éléments
8  print("les 3 premiers éléments sont : ",tuple1[:3])
9
10 # afficher le max de tuple1
11 print("max : ",max(tuple1))
12
13 # afficher la somme des éléments de tuple1
14 print("somme : ",sum(tuple1))
15
16 tuple3=tuple1 + tuple2
17 print("concaténation de 2 tuples : ",tuple3)
18
19 # répéter un tuple 2 fois
20 tuple4=tuple1 * 2
21 print("double : ", tuple4)
22
23 for elem in tuple1:
24     print(elem,end=' , ')
```

Remarque :

Sum() : fonction qui calcule la somme des éléments d'une liste ou un tuple

Manipulation des tuples

- la len()fonction accepte les tuples et retourne le nombre d'éléments contenus à l'intérieur;
- l' +opérateur peut joindre des tuples ensemble (nous vous l'avons déjà montré)

```
tuple2=myTuple+myTuple
```

- l'opérateur * peut multiplier les tuples, tout comme les listes;

```
tuple2 = myTuple * 3
```

- les opérateurs in et not in fonctionnent de la même manière que dans les listes.

```
print(10 in myTuple)

print(-10 not in myTuple)
```

Dictionnaire en python

Le **dictionnaire** est une autre structure de données Python.

un dictionnaire n'est pas une liste - une liste contient un ensemble de valeurs numérotées, tandis qu'un dictionnaire contient **des paires de valeurs** ;

• la fonction len() fonctionne également pour les dictionnaires - elle renvoie le nombre d'éléments de **valeur-clé** dans le dictionnaire;

• un dictionnaire est un outil à sens unique - si vous avez un dictionnaire **anglais-français,** vous pouvez rechercher des **équivalents français de termes anglais, mais pas l'inverse.**

Dictionnaire en python

Remarque:

• chaque clé doit être **unique - il n'est pas possible d'avoir plus d'une clé de la même valeur;**

• une clé peut être des **données de tout type (sauf liste): elle peut être un nombre (entier ou flottant), voire une chaîne;**

Comment faire un dictionnaire?

La liste des paires est **entourée d'accolades** , tandis que les paires elles-mêmes sont **séparées par des virgules** et les **clés et valeurs par des deux-points** .

Le dictionnaire dans son ensemble peut être imprimé avec une seule `print()` invocation.

dictionary = {**"cat"** : **"chat"**, **"dog"** : **"chien"**, **"horse"** : **"cheval"**}

phone_numbers = {**'Ali'** : 0623345623, **'Laila'** : 0698345612}

empty_dictionary = {}

print(dictionary) ➔ {'cat': 'chat', 'dog': 'chien', 'horse': 'cheval'}

print(phone_numbers) ➔ {'Ali': 623345623, 'Laila': 698345612}

print(empty_dictionary) ➔ {}

Comment utiliser un dictionnaire?

Si vous souhaitez obtenir l'une des valeurs, vous devez fournir une valeur de clé valide:

dictionary = {**"cat"** : **"chat"**, **"dog"** : **"chien"**, **"horse"** : **"cheval"**}
phone_numbers = {**'Ali'** : 0623345623, **'Laila'** : 0698345612}

print(dictionary['cat']) ➔ chat

print(phone_numbers['Laila'])➔ 698345612

Remarque:

vous ne devez pas utiliser de clé inexistante .

L’operateur in et not in

Heureusement, il existe un moyen simple d'éviter une telle situation(clé inexistant).

L' opérateur in, avec son compagnon not in, peut sauver cette situation.

```
dictionary = {"cat" : "chat", "dog" : "chien", "horse" : "cheval"}

words = ['cat', 'lion', 'horse']

for word in words:
    if word in dictionary:
        print(word, "->", dictionary[word])
    else:
        print(word, "is not in dictionary")
```

Comment parcourir un dictionnaire

Le premier d'entre eux est une méthode nommée **keys(),** possédée par chaque dictionnaire. La méthode **renvoie un objet itérable composé de toutes les clés rassemblées dans le dictionnaire** . Avoir un groupe de clés vous permet d'accéder à l'ensemble du dictionnaire de manière simple et pratique.

dictionary = {"cat" : "chat", "dog" : "chien", "horse" : "cheval"}

for cle in **dictionary.keys():** print(cle, "->", dictionary[cle] La sortie du code : horse -> cheval dog -> chien cat -> chat	for cle in **sorted(dictionary.keys()):** print(cle, "->", dictionary[cle] La sortie du code : cat -> chat dog -> chien horse -> cheval

Comment parcourir un dictionnaire

Une autre façon est basée sur l'utilisation d'une méthode de dictionnaire nommée **items().**
La méthode **renvoie des tuples où chaque tuple est une paire clé-valeur** .

```
dictionary = {"cat" : "chat", "dog" : "chien", "horse" : "cheval"}

for cle, valeur in dictionary.items():

        print(cle, "->", valeur)
```

La sortie du code :

cat -> chat

dog -> chien

horse -> cheval

Comment parcourir un dictionnaire

Il existe également une méthode nommée values(), qui fonctionne de manière similaire à keys(), mais **renvoie des valeurs** .

Voici un exemple simple:

```
dictionary = {"cat" : "chat", "dog" : "chien", "horse" : "cheval"}

for valeur in dictionary.values():

        print(valeur)
```

Voici la sortie attendue:

```
cheval

chien

chat
```

Modifier et ajouter des valeurs

Attribuer une nouvelle valeur à une clé existante est simple - comme les dictionnaires sont entièrement **modifiables** , il n'y a aucun obstacle à les modifier.

Nous allons remplacer la valeur "chat" par "minou", qui n'est pas très précise, mais cela fonctionnera bien avec notre exemple.

```
dictionary = {"cat" : "chat", "dog" : "chien", "horse" : "cheval"}

dictionary['cat'] = 'minou'

print(dictionary)
```

La sortie est:

```
{'dog': 'chien', 'horse': 'cheval', 'cat': 'minou'}
```

Modifier et ajouter des valeurs

Ajout d'une nouvelle clé

L'ajout d'une nouvelle paire **clé-valeur** à un dictionnaire est aussi simple que de changer une valeur - vous n'avez qu'à affecter une valeur à une nouvelle **clé** , **auparavant inexistante.**

Ajoutons une nouvelle paire de mots au dictionnaire - un peu bizarre, mais toujours valide:

```
dictionary = {"cat" : "chat", "dog" : "chien", "horse" : "cheval"}

dictionary['swan'] = 'cygne'

print(dictionary)
```

L'exemple affiche:

```
{'cat': 'chat', 'dog': 'chien', 'horse': 'cheval', 'swan': 'cygne'}
```

Modifier et ajouter des valeurs

Vous pouvez également **insérer** un élément dans un dictionnaire en utilisant la méthode **update()**, par exemple:

```
dictionary = {"cat" : "chat", "dog" : "chien", "horse" : "cheval"}

dictionary.update({"duck" : "canard"})

print(dictionary)
```

La sortie:

```
{'cat': 'chat', 'dog': 'chien', 'horse': 'cheval', 'duck': 'canard'}
```

Retrait d'une clé

Pouvez-vous deviner comment **supprimer** une clé d'un dictionnaire?

Remarque: la suppression d'une clé entraînera toujours la **suppression de la valeur associée** . **Les valeurs ne peuvent exister sans leurs clés** .

Cela se fait avec l'instruction del.

Voici l'exemple:

```
dictionary = {"cat" : "chat", "dog" : "chien", "horse" : "cheval"}
del dictionary['dog']
print(dictionary)
```

La sortie :

```
{'cat': 'chat', 'horse': 'cheval'}
```

Retrait d'une clé

Pour **supprimer le dernier élément d'un dictionnaire**, vous pouvez utiliser la méthode **popitem()** :

```
dictionary = {"cat" : "chat", "dog" : "chien", "horse" : "cheval"}

dictionary.popitem()

print(dictionary)
```

La sortie :

```
{'cat' : 'chat', 'dog' : 'chien'}
```

EXERCICES

- TP 8 :

Les Listes , tuples et dictionnaires

Exercice

1- Créer un programme qui crée et initialise un tableau, puis insère un élément à la position spécifiée dans ce tableau (de 0 à N-1).

- *Pour insérer un nouvel élément dans le tableau, déplacez les éléments de la position d'insertion donnée vers une position vers la droite.*

2- Créer un programme qui crée et initialise un tableau, puis supprimez un élément de ce tableau à la position spécifiée (de 0 à N-1).

- *Pour supprimer un élément du tableau, déplacez les éléments juste après la position donnée vers une position à gauche et réduisez la taille du tableau.*

Exercice

Soit la liste: L = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

A partir de L:

- Ajouter la valeur 1 à chacun de ses éléments.
- Ajouter la valeur 11 à la fin de la liste.
- Ajouter les valeurs 12 et 13 à la fin de la liste.
- Afficher le premier élément, les deux premiers éléments, le dernier élément, les deux derniers éléments.
- Construire la liste "paires" qui contient les nombres paires de L et la liste "impaire" qui contient les nombres "impaires" de L.
- Ajouter la valeur 3.5 entre 3 et 4.
- Supprimer la valeur 3.5.
- Inverser l'ordre des éléments de L.
- Demander à l'utilisateur de fournir un nombre au hasard et dire si ce nombre est présent dans L.

Exercice

Soit le dictionnaire :

d = {'nom': 'Dupuis', 'prenom': 'Jacque', 'age': 30}

- Corriger l'erreur dans le prénom, la bonne valeur est 'Jacques'.
- Afficher la liste des clés du dictionnaire.
- Afficher la liste des valeurs du dictionnaire.
- Afficher la liste des paires clé/valeur du dictionnaire.
- Ecrire la phrase "Jacques Dupuis a 30 ans".

Chapitre 9:

Les chaines en Python

Introduction

Sous Python, une donnée de type string(chaîne de caractere) est une suite quelconque de caractères délimitée soit par des apostrophes (simple quotes), soit par des guillemets (double quotes), soit par des triples quotes (''' ou """), elles sont des séquences immuables .
Pour accéder à un caractère bien déterminé, on utilise le nom de la variable qui contient la chaîne, et on lui accole entre deux crochets l'index numérique qui correspond à la position du caractère dans la chaîne.

Opérations sur les chaînes

Si vous voulez modifier une chaîne de caractères, vous devez en construire une nouvelle. Pour cela, n'oubliez pas que les opérateurs de concaténation (+) et de duplication (*) peuvent vous aider.

str1 = 'a'

str2 = 'b'

print(str1 + str2) ➔ab

print(str2 + str1)➔ba

print(5 * ‘a’)➔aaaaa

print('b' * 4)➔bbbb

Taille des chaines

La fonction **len()** Donne la longueur d'une chaine de caractere saisie en argument.

N'oubliez pas qu'une barre oblique inverse (\) utilisée comme caractère d'échappement n'est pas incluse dans la longueur totale de la chaîne.

Taille des chaines

# Example 1 **var = 'by'** **print(len(var))**	**2**
# Example 2 var = '' print(len(var))	0
# Example 3 var= 'I\'m' print(len(var))	3

Accès aux éléments: indexation

Nous vous avons dit précédemment que les chaînes de Python sont des séquences . Il est temps de vous montrer ce que cela signifie réellement.

Par exemple, si vous souhaitez accéder à l'un des caractères d'une chaîne, vous pouvez le faire en utilisant l' indexation , comme dans l'exemple.

```
ch= 'silly walks'

for ix in range(len(ch)):

    print(ch[ix], end=' ')

print()
```

L'exemple de sortie est:

s i l l y w a l k s

Accès aux éléments : itération

Pour en faciliter l'écriture, Python vous propose une structure de boucle plus appropriée

```
ch= 'silly walks’

 for car in ch:
        print(car , end=' ‘)
 print()
```

La variable car contiendra successivement tous les caractères de la chaîne, du premier jusqu'au dernier

L'exemple de sortie est:

s i l l y w a l k s

Les sous chaînes

Chaîne[min: max:pas] ➔ chaîne[: :]=chaîne[0:len(chaîne) : 1]

alpha = "a b d e f g" 0 1 2 34 5	
print(alpha[1:3])	bd
print(alpha[3:])	efg
print(alpha[:3])	abd
print(alpha[3:-2])	e
print(alpha[1:-2])	bde
print(alpha[-3:4])	e
print(alpha[::2])	adf
print(alpha[1::2])	beg

Les opérateurs in et not in

L' opérateur in ne devrait pas vous surprendre lorsqu'il est appliqué à des chaînes - il vérifie simplement si son argument de gauche peut être trouvé n'importe où dans l'argument de droite.

Le résultat du contrôle est simplement True ou False.

```
alphabet = "abcdefghijklmnopqrstuvwxyz"
print("f" in alphabet)
print("F" in alphabet)
print("ghi" in alphabet)
print("Xyz" in alphabet)
```

Les chaînes Python sont immuables

Nous vous avons également dit que les chaînes de Python sont immuables . C'est une caractéristique très importante. Qu'est-ce que ça veut dire?

Tout ce que vous pouvez faire avec une liste ne peut pas être fait avec une chaîne. La première différence importante ne vous permet pas d'utiliser l'instruction del pour supprimer quoi que ce soit d'une chaîne . L'exemple ici ne fonctionnera pas:

alphabet = "abcdefghijklmnopqrstuvwxyz"

del alphabet[0]

La seule chose que vous pouvez faire avec del et une chaîne est de supprimer la chaîne dans son ensemble .

Ch="hello"

Ch[0]='y' donne une erreur

Ch='y'+Ch[1:4] correcte

Conversion des caractères:

Si tu veux pour connaître la valeur du point de code ASCII / UNICODE d'un caractère spécifique, vous pouvez utiliser une fonction nommée ord()

```
ch1 = 'a'
ch2 = ' ' # space
print(ord(ch1))
print(ord(ch2))
```

L'extrait de code sort:
97 32

Conversion des caractères:

Si vous connaissez le point de code (numéro) et souhaitez obtenir le caractère correspondant, vous pouvez utiliser une fonction nommée chr().

La fonction prend un point de code et renvoie son caractère .

```
print(chr(97))
print(chr(945))
```

L'exemple d'extrait de code génère:

a
α

Remarque:

- chr(ord(x)) =x
- ord(chr(x)) = x

Fonctions plus spécifiques aux chaînes de caractères

min().

La fonction **trouve l'élément minimum de la séquence passé en argument** .

print(min("aAbByYzZ")) ➔ A

Ch="aAbByYzZ"

Print(min(ch)) ➔A

il s'agit d'un *A* majuscule . Pourquoi? Rappelez-vous le tableau ASCII

max ()

De même, une fonction nommée max() **trouve l'élément maximum de la séquence** .

print(max("aAbByYzZ"))➔z

Ch="aAbByYzZ"

Print(max(ch)) ➔z

Fonctions plus spécifiques aux chaînes de caractères:

index()

La methode recherche la séquence depuis le début et renvoie l' index de la première occurrence de l'argument.

print("aAbByYzZaA".index("b"))➔ 2

Ch="aAbByYzZaA"

print(ch.index("Z"))➔ 7

count()

La méthode compte toutes les occurrences de l'élément à l'intérieur de la séquence . L'absence de tels éléments ne pose aucun problème.

print("abcabc".count("b"))➔2

Ch= 'abcabc'

print(ch.count("d"))➔0

Fonctions plus spécifiques aux chaînes de caractères:

capitalize()

La méthode fait exactement ce qu'elle dit - **elle crée une nouvelle chaîne remplie de caractères tirés de la chaîne source** , mais elle essaie de les modifier de la manière suivante:

•**si le premier caractère à l'intérieur de la chaîne est une lettre** (note: le premier caractère est un élément avec un index égal à 0 , pas seulement le premier caractère visible), **il sera converti en majuscules** ;

•**toutes les lettres restantes de la chaîne seront converties en minuscules** .

print('aBcD'.capitalize())➔ Abcd

print(‘ aBcD'.capitalize())➔ abcd

Fonctions plus spécifiques aux chaînes de caractères:

endswith()

La méthode vérifie si la chaîne donnée se termine par l'argument spécifié et

renvoie TrueouFalse , selon le résultat de la vérification.

print("epsilon".endswith("on"))➔true

startswith()

vérifie si une chaîne donnée commence par la sous-chaîne spécifiée

print("omega".startswith("meg"))

print("omega".startswith("om"))

Fonctions plus spécifiques aux chaînes de caractères:

find()

La méthode est similaire à index(), que vous connaissez déjà - **elle recherche une sous-chaîne et renvoie l'index de la première occurrence de cette sous-chaîne** , mais:

•cela **ne génère pas d'erreur pour un argument contenant une sous-chaîne inexistante** (il retourne -1 alors)

print("Eta".find("ta"))➔1

print("Eta".find("mma"))➔-1

Si vous souhaitez effectuer la recherche, non pas depuis le début de la chaîne, mais depuis n'importe quelle position , vous pouvez utiliser:

print('kappa'.find('a', 2))➔4

Le deuxième argument **spécifie l'index auquel la recherche sera lancé**

rfind()

commence la recherche à partir de la fin de la chaîne

print("tau tau tau".rfind("ta"))➔8

Fonctions plus spécifiques aux chaînes de caractères:

isalnum()

La méthode **vérifie si la chaîne ne contient que des chiffres ou des caractères alphabétiques (lettres) et renvoie** True**ou** False **en** fonction du résultat.

print('lambda'.isalnum())➔ true

print('@'.isalnum())➔ false

isalpha ()

La isalpha()méthode est plus spécialisée - elle ne s'intéresse qu'aux lettres .

print("Moooo".isalpha())➔true

print('Mu40'.isalpha())➔ false

Fonctions plus spécifiques aux chaînes de caractères:

isdigit ()

À son tour, la isdigit()méthode ne regarde que les chiffres - tout le reste produit False comme résultat.

print('2018'.isdigit())➔true

print("Year2019".isdigit())➔false

islower ()

La islower()méthode est une variante difficile de isalpha()- elle n'accepte que les lettres minuscules .

print("Moooo".islower())➔false

print('moooo'.islower())➔true

Fonctions plus spécifiques aux chaînes de caractères:

isspace ()

La isspace()méthode identifie uniquement les espaces - elle ne tient pas compte de tout autre caractère (le résultat est Falsealors).

print(" ".isspace()) ➔ true

print("mooo mooo mooo".isspace()) ➔ false

isupper ()

La isupper()méthode est la version majuscule de islower()- elle se concentre uniquement sur les lettres majuscules .

print("Moooo".isupper()) ➔ false

print('MOOOO'.isupper()) ➔ true

Fonctions plus spécifiques aux chaînes de caractères:

lower ()

La méthode lower() fait une copie d'une chaîne source, remplace toutes les lettres majuscules par leurs équivalents minuscules et renvoie la chaîne comme résultat. Encore une fois, la chaîne source reste intacte.

print("SiGmA=60".lower())➔sigma=60

upper ()

La méthode upper() fait une copie d'une chaîne source, remplace toutes les lettres par leurs équivalents majuscules minuscules et renvoie la chaîne comme résultat. Encore une fois, la chaîne source reste intacte.

print("SiGmA=60".upper())➔SIGMA=60

Fonctions plus spécifiques aux chaînes de caractères:

replace ()

La méthode à deux paramètres renvoie une copie de la chaîne d'origine dans laquelle toutes les occurrences du premier argument ont été remplacées par le deuxième argument .replace()

print("www.netacad.com".replace("netacad.com", "pythoninstitute.org"))➔www.pyhoninstitute.org

print("This is it!".replace("is", "are")) ➔ Thare are it!

Comparer des chaînes

Les chaînes de Python **peuvent être comparées en utilisant le même ensemble d'opérateurs** que celui utilisé par rapport aux nombres.

==,!=,>,>=,<,<=

Les deux comparaisons donnent True comme résultat:

'alpha' == 'alpha'

'alpha' != 'Alpha'

La relation est True

'alpha' < 'alphabet'

La conversion de chaînes de nombres

Il y a deux problèmes supplémentaires qui devraient être discutés ici: comment convertir un nombre (un entier ou un flottant) en une chaîne, et vice versa . Il peut être nécessaire d'effectuer une telle transformation. De plus, c'est un moyen courant de traiter les données d'entrée / sortie.

La conversion de chaînes de nombres est simple, car elle est toujours possible. C'est fait par une fonction nommée str().

La transformation inverse (string-number) est possible lorsque et seulement lorsque la chaîne représente un nombre valide.

Utilisez la int()fonction si vous souhaitez obtenir un entier et float()si vous avez besoin d'une valeur à virgule flottante.

Liste et chaîne de caracteres

- Transformer une liste en chaine de caractère : ch='separateur'.join(liste)

```
liste=['a','b','c','d']
ch=':'.join(liste)
print(ch) ➔ a:b:c:d
```

- Transformer une chaine de caractère en liste : liste=ch.split('separateur')

```
ch='a:b:c:a:b:c'
liste1=ch.split(':')
print(liste1)➔ ['a', 'b', 'c', 'a', 'b', 'c']
```

La fonction liste

La fonction list() **prend son argument (une chaîne) et crée une nouvelle liste contenant tous les caractères de la chaîne, un par élément de liste** .

Remarque: ce n'est pas strictement une fonction de chaîne - list()est capable de créer une nouvelle liste à partir de nombreuses autres entités

(par exemple, à partir de tuples et de dictionnaires à voir par la suite).

Exemple:

```
print(list("abcabc"))
```

➔['a', 'b', 'c', 'a', 'b', 'c']

EXERCICES

- TP 9 :

  Les chaînes de caractères

Chapitre 11:

Les fichiers en Python

Editer un fichier

Une grande partie de l'information est stockée sous forme de texte dans des fichiers. Pour traiter cette information, vous devez le plus souvent lire ou écrire dans un ou plusieurs fichiers. Python possède pour cela de nombreux outils qui vous simplifient la vie.

Pour travailler avec le contenu des fichiers (lire ou écrire), il faut « ouvrir » ces fichiers avec la fonction « open() ». Cette fonction prend obligatoirement deux paramètres : le fichier (avec son chemin relatif ou absolu), et le mode d'ouverture. C'est le « mode d'ouverture » qui défini si on va simplement lire le contenu du fichier, ou si on va écrire dedans.

La fonction open

Voici la syntaxe pour ouvrir un fichier

fichier = open(chemin ,mode d’ouverture)

Chemin

Un **chemin relatif** en informatique est un chemin qui prend en compte l'emplacement de lecture.

Un **chemin absolu** est un chemin complet qui peut être lu quelque soit l'emplacement de lecture.

Les types d'ouverture

r, pour une ouverture en lecture (READ).

w, pour une ouverture en écriture (WRITE), à chaque ouverture le contenu du fichier est écrasé. Si le fichier n'existe pas python le crée.

a, pour une ouverture en mode ajout à la fin du fichier (APPEND). Si le fichier n'existe pas python le crée.

b, pour une ouverture en mode binaire.

t, pour une ouverture en mode texte.

x, crée un nouveau fichier et l'ouvre pour écriture

Parcourir un fichier

Après avoir ouvert un fichier, il existe différentes méthodes pour le parcourir :

- la méthode **read()** retourne tout le contenu du fichier dans une variable de type `string`

```
contenu = fichier.read()
```

- la méthode **readline()** retourne la première ligne du fichier lors de son premier appel, la seconde ligne à son second appel, la troisième, etc.

```
ligne1 = fichier.readline()  # ligne 1
ligne2 = fichier.readline()  # ligne 2
```

- la méthode **readlines()** retourne toutes les lignes du fichier dans une variable de type list

```
liste_lignes = fichier.readlines()
```

- la boucle **for,** c'est la plus efficace pour parcourir de très gros fichiers, elle s'utilise directement sur l'objet.

```
for ligne in fichier:
    [...]
```

Ouvrir un fichier en lecture

Pour notre exemple, créez d'abord un fichier « liste-courses.txt » avec pour contenu, par exemple :
salade
tomates
oignons
Ensuite, dans le même dossier, créez un fichier Python « lecture.py » :

```
# ouverture en lecture
fichier = open('liste-courses.txt', 'r')
```

```
# on parcours le contenu du fichier ligne par ligne
for ligne in fichier:
    print(ligne)
```

```
# lecture du fichier entier
contenu = fichier.read()
print(contenu)
```

```
# il ne faut pas oublier de fermer le fichier
fichier.close()
```

Ouvrir un fichier en lecture

Pour notre exemple, créez d'abord un fichier « liste-courses.txt » avec pour contenu, par exemple :
salade
tomates
oignons
Ensuite, dans le même dossier, créez un fichier Python « lecture.py » :

```
# ouverture en lecture
fichier = open('liste-courses.txt', 'r')
```

```
ligne1 = fichier.readline()  # ligne 1
ligne2 = fichier.readline()  # ligne 2
...
```

```
#Toutes les lignes du fichier dans une variable de type `list`
liste_lignes = fichier.readlines()
print(contenu)
```

```
# il ne faut pas oublier de fermer le fichier
fichier.close()
```

Ouvrir un fichier en écriture

Maintenant avec le script suivant, qu'on peut appeler « ecriture.py », on va écrire dans notre fichier.

```
# ouverture en écriture
fichier = open('liste-courses.txt', 'w')
# on écrit des éléments dans notre fichier
fichier.write('bananes\n')
fichier.write('pommes\n')
fichier.write('fraises\n')
fichier.close()
```

Remarque:

Dans le cas où le fichier n'existait pas initialement, il sera créer.

Ouvrir un fichier en ajout

Si on veut écrire à la suite du fichier, il faut utiliser le mode « a » (pour « append »).

```
# ouverture en écriture (à la suite)
fichier = open('liste-courses.txt', 'a')


fichier.write('salade\n')
fichier.write('tomates\n')
fichier.write('oignons\n')
fichier.close()
```

Remarque:

Dans le cas où le fichier n'existait pas initialement, il sera créer.

Le mot clé with

Lors du parcours du fichier, si une erreur se produit et que le fichier n'est pas fermé, il devient inutilisable dans la suite du programme. Dans certains cas, il est donc préférable d'utiliser la fonction `open()` avec le mot clé ‘with’ qui s'occupera de fermer le fichier en cas d'exception. Dans ce cas il n'est plus nécessaire de fermer explicitement le fichier.

Voici la syntaxe:

```
with open('liste-courses.txt', 'r') as fichier:
contenu = fichier.read()
print(contenu)
```

EXERCICES

- TP 11 :

Les fichiers