TP Python 4 - Chaînes de caractères, tuples, dictionnaires

Chaînes de caractères

Une chaîne de caractères est une séquence de caractères de longueur fixée (on ne peut pas lui ajouter un élément sans recréer la chaîne) et non mutable (on ne peut pas modifier un élément). Pour créer une chaîne de caractères, on peut l'entourer d'apostrophes (') ou de guillemets (").

On accède au i^e caractère d'une chaîne `c` par la syntaxe `c[i]` mais, contrairement aux listes, on ne peut pas modifier `c[i]` directement :

```
>>> c = 'abcd'
>>> c[1]
'b'
>>> c[1] = 'x'
TypeError: 'str' object does not support item assignment
```

Dans une chaîne, `\n` représente un passage à la ligne.

```
>>> print('abc\ndef\nghi')
abc
def
ghi
```

En utilisant ''' ou """ comme délimiteur on peut créer des chaînes de plusieurs lignes :

```
>>> print("""On peut mettre ' ou " ici
...
... et même écrire sur plusieurs lignes.""")
On peut mettre ' ou " ici


et même écrire sur plusieurs lignes.
```

Comme pour les listes, la fonction `len` renvoie la longueur d'une chaîne.

```
>>> len('abcd')
4
```

La fonction `str` permet de convertir certains objets en chaînes de caractères :

```
>>> str(123)
'123'
```

On peut concaténer deux chaînes avec `+` :

```
>>> 'abc' + 'def'
'abcdef'
```

On peut utiliser `*` pour concaténer plusieurs copies d'une même chaîne :

```
>>> 'Bonjour' * 5
'BonjourBonjourBonjourBonjourBonjour'
```

La méthode `append` ne s'applique pas aux chaînes. Pour construire une chaîne caractère par caractère, on utilisera la concaténation. Exemple :

```
>>> chiffres = ''                       # chaîne vide
>>> for i in range(10):
...     chiffres = chiffres + str(i)   # ou chiffres += str(i)
>>> chiffres
0123456789
```

Comme pour les listes, on peut utiliser le slicing pour extraire une partie d'une chaîne. La syntaxe est `c[debut:fin:pas]`, où le caractère d'indice `debut` est inclus mais celui d'indice `fin` est exclu.

```
>>> c = 'abcdef'
>>> c[1:4]   # caractères d'indices 1, 2 et 3
'bcd'
>>> c[1::2]  # avec un pas
'bdf'
>>> c[::-1]  # avec un pas négatif
'fedcba'
```

On peut tester l'appartenance d'un caractère ou d'une sous-chaîne à une chaîne avec `in` :

```
>>> 'd' in 'abcde'
True
>>> 'bcd' in 'abcde'
True
```

On peut itérer sur une chaîne caractère par caractère :

```
>>> for lettre in 'alphabet':
...     print(lettre, end=' ')
a l p h a b e t
```

La méthode `split` renvoie la liste des mots d'une chaîne (les mots d'une chaîne sont ses sous-chaînes maximales sans espaces). On peut préciser un séparateur.

```
>>> 'Ceci est une chaîne de caractères'.split()
['Ceci', 'est', 'une', 'chaîne', 'de', 'caractères']
>>> 'Ceci est une chaîne de caractères'.split('c')
['Ce', 'i est une ', 'haîne de ', 'ara', 'tères']
```

Dans le standard Unicode, chaque caractère est associé à un entier naturel (65 à 90 pour les lettres A-Z et 97 à 122 pour a-z par exemple). La fonction `ord` associe à un caractère son identifiant Unicode. Réciproquement, la fonction `chr` associe à un entier naturel le caractère Unicode correspondant.

```
>>> ord('A')
65
>>> chr(65)
'A'
```

Exercice 1

1) Écrire une fonction `est_voyelle` qui, recevant une lettre (chaîne de caractères de longueur 1), renvoie `True` si la lettre est une voyelle (minuscule ou majuscule) et `False` sinon.

```
>>> est_voyelle('a')
True
```

2) Écrire une fonction `nombre_de_voyelles` qui, recevant une chaîne de caractères, renvoie le nombre de voyelles que contient cette chaîne.

```
>>> nombre_de_voyelles('Un exemple')
4
```

3) Écrire une fonction `voyelles_etoiles` qui, recevant une chaîne de caractères, renvoie la chaîne où toutes les voyelles ont été transformées en astérisques (`*`).

```
>>> voyelles_etoiles('Un exemple')
'*n *x*mpl*'
```

Exercice 2 Écrire une fonction `est_palindrome` qui, recevant une chaîne de caractères, renvoie `True` si la chaîne est un palindrome (i.e. une chaîne qui se lit dans les deux sens) et `False` sinon. On donnera une version avec slicing et une version sans slicing.

```
>>> est_palindrome('radar')
True
```

Exercice 3 Écrire une fonction `positions` qui, recevant une lettre et une chaîne de caractères, renvoie la liste des positions de la lettre dans la chaîne.

```
>>> positions('e', "Ses ailes de géant l'empêchent de marcher")
[1, 7, 11, 21, 27, 32, 39]
```

Tuples

Un tuple est une séquence d'objets de longueur fixée (on ne peut pas lui ajouter un élément sans recréer le tuple) et non mutable (on ne peut pas modifier un élément). Ils sont délimités par des parenthèses. De nombreuses méthodes et fonctions sont communes aux listes et aux tuples (mais il n'y a pas de `append` pour les tuples).

```
>>> t = (5, 9, 2, 1)
>>> t[1]
9
>>> t[1] = 3
TypeError: 'tuple' object does not support item assignment
>>> len(t)          # longueur
4
>>> 2 in t          # test d'appartenance
True
>>> t[1:3]          # slicing
(9, 2)
>>> u = (7, 6, 3)
>>> t + u           # concaténation
(5, 9, 2, 1, 7, 6, 3)
```

Dictionnaires

Un dictionnaire est une structure de données formée d'associations entre deux ensembles d'objets appelés **clés** et **valeurs**. On crée un dictionnaire en donnant à chaque clé la valeur qui lui est associée :

```
>>> notes = {'Albert': 12, 'Amélie': 14, 'Léon': 6}
```

Les clés de ce dictionnaire sont `'Albert'`, `'Amélie'` et `'Léon'`. Les valeurs associées à ces clés sont respectivement `12`, `14` et `6`.

On peut récupérer la valeur associée à une clé :

```
>>> notes['Amélie']
14
```

On peut modifier la valeur associée à une clé :

```
>>> notes['Albert'] = 14
>>> notes
{'Albert': 14, 'Amélie': 14, 'Léon': 6}
```

On peut ajouter un élément au dictionnaire :

```
>>> notes['Simon'] = 8
>>> notes
{'Albert': 14, 'Amélie': 14, 'Léon': 6, 'Simon': 8}
```

On peut itérer sur un dictionnaire (l'itération se fait sur les clés) :

```
>>> for eleve in notes:
        print(eleve, "a eu", notes[eleve])

Albert a eu 14
Amélie a eu 14
Léon a eu 6
Simon a eu 8
```

Le dictionnaire vide est noté `{}`.

Dans l'exemple suivant on crée un dictionnaire à partir du vide en le remplissant à l'aide d'une boucle :

```
>>> d = {}
>>> for k in range(6):
        if k % 2 == 0:
            d[k] = 'pair'
        else:
            d[k] = 'impair'
>>> d
{0: 'pair', 1: 'impair', 2: 'pair', 3: 'impair', 4: 'pair', 5: 'impair'}
```

Exercices

Exercice 4

1) Créer (avec une boucle) un dictionnaire `caracteres` qui associe à chaque entier compris entre 65 et 90 le caractère Unicode correspondant.

```
>>> caracteres
{65: 'A', 66: 'B', 67: 'C', 68: 'D', 69: 'E', 70: 'F', 71: 'G', 72: 'H',
 73: 'I', 74: 'J', 75: 'K', 76: 'L', 77: 'M', 78: 'N', 79: 'O', 80: 'P',
 81: 'Q', 82: 'R', 83: 'S', 84: 'T', 85: 'U', 86: 'V', 87: 'W', 88: 'X',
 89: 'Y', 90: 'Z'}
```

2) Créer un dictionnaire `codes` qui associe à chaque lettre majuscule son numéro Unicode.

```
>>> codes
{'A': 65, 'B': 66, 'C': 67, 'D': 68, 'E': 69, 'F': 70, 'G': 71, 'H': 72,
 'I': 73, 'J': 74, 'K': 75, 'L': 76, 'M': 77, 'N': 78, 'O': 79, 'P': 80,
 'Q': 81, 'R': 82, 'S': 83, 'T': 84, 'U': 85, 'V': 86, 'W': 87, 'X': 88,
 'Y': 89, 'Z': 90}
```

Exercice 5

1) Écrire une fonction `moyenne` qui, recevant une liste de nombres, renvoie sa moyenne.

```
>>> moyenne([4, 1, 12, 6])
5.75
```

2) On suppose qu'on dispose d'un dictionnaire qui associe à chaque élève la liste de ses notes. Écrire une fonction `moyennes` qui, recevant un dictionnaire de ce type, renvoie un dictionnaire associant à chaque élève sa moyenne.

```
>>> notes = {"Laura": [5, 12, 6], "Bob": [15, 12, 12], "Judy": [8, 15]}
>>> moyennes(notes)
{'Judy': 11.5, 'Bob': 13.0, 'Laura': 7.666666666666667}
```

Exercice 6

1) Écrire une fonction `occurrences` qui, recevant une chaîne de caractères, renvoie un dictionnaire qui associe à chaque lettre minuscule de l'alphabet le nombre de fois qu'elle apparaît dans la chaîne.

```
>>> occurrences('portez ce vieux whisky au juge blond qui fume')
{'a': 1, 'b': 1, 'c': 1, 'd': 1, 'e': 5, 'f': 1, 'g': 1, 'h': 1, 'i': 3,
'j': 1, 'k': 1, 'l': 1, 'm': 1, 'n': 1, 'o': 2, 'p': 1, 'q': 1, 'r': 1,
's': 1, 't': 1, 'u': 5, 'v': 1, 'w': 1, 'x': 1, 'y': 1, 'z': 1}
```

2) En utilisant la fonction précédente, écrire une fonction `lettre_frequence_max` qui, recevant une chaîne de caractères, renvoie la lettre minuscule qui y apparaît le plus souvent.

```
>>> lettre_frequence_max('portez ce vieux whisky au juge blond qui fume')
'e'
```

Exercice 7

1) Écrire une fonction `est_sur` qui vérifie si un mot de passe est sûr ou non. On considérera qu'un mot de passe est sûr s'il contient au moins dix caractères dont au moins un chiffre, une lettre minuscule et une lettre majuscule.

```
>>> est_sur('azerty12345')
False
>>> est_sur('PaSsWoRd007')
True
```

Pour les questions suivantes on utilisera la fonction `input` qui permet de demander une chaîne de caractères à l'utilisateur.

```
>>> x = input('Quel est votre nom ? ')
Quel est votre nom ? Basile
>>> x
'Basile'
```

2) Écrire une fonction `inscription` sans paramètre qui demande à l'utilisateur son nom et son mot de passe et, si celui-ci est sûr, les stocke dans un dictionnaire appelé `MOTSDEPASSE` (qui sera une variable globale). La fonction doit se comporter comme suit :

```
>>> inscription()
Nom d'utilisateur : boris
Mot de passe : invincible
Ce mot de passe n'est pas sûr.
Mot de passe : J3su1s1nv1nc1bl3
Confirmer mot de passe : j3su1s1nv1nc1bl3
Confirmer mot de passe : J3su1s1nv1nc1bl3
Inscription terminée
>>> MOTSDEPASSE
{'boris': 'J3su1s1nv1nc1bl3'}
```

3) Écrire une fonction `login` sans paramètre qui demande à l'utilisateur son nom et son mot de passe et, si celui-ci est correct, affiche `Accès autorisé` et renvoie `True`. La fonction doit se comporter comme suit :

```
>>> login()
Nom d'utilisateur : james
'Utilisateur non inscrit'
>>> login()
Nom d'utilisateur : boris
Mot de passe : Jesuisinvincible
Mot de passe incorrect
Mot de passe : J3su1s1nv1nc1bl3
Accès autorisé
True
```

Exercice 8 Le ROT13 est une technique très simple de cryptage d'un texte : chaque lettre est décalée de 13 lettres dans l'alphabet. Par exemple, a devient n, b devient o, etc. et z devient m. L'avantage est que pour coder le texte et pour le décoder, on utilise le même algorithme. Écrire une fonction `rot13` qui code/décode un texte selon cette méthode.

```
>>> rot13('ceci est un exemple')
'prpv rfg ha rkrzcyr'
>>> rot13('prpv rfg ha rkrzcyr')
'ceci est un exemple'
>>> rot13('gncre qnaf yn pbafbyr vzcbeg nagvtenivgl')
```

Exercice 9 Écrire un programme qui affiche les jours de l'année 2022 (sous la forme jour - numéro - mois). Le 1er janvier était un samedi.

```
samedi 1 janvier
dimanche 2 janvier
lundi 3 janvier
mardi 4 janvier
mercredi 5 janvier
jeudi 6 janvier
...
```

Exercice 10 Programmer un jeu de pendu. On pourra créer une liste de mots à partir d'un fichier texte quelconque et utiliser la fonction `choice` du module `random` pour choisir au hasard un mot dans cette liste. La fonction `input` permet de demander une chaîne de caractères à l'utilisateur.

Exercice 11 Écrire une fonction `encodage_morse` qui, recevant une chaîne de caractères ne contenant que des lettres minuscules et des espaces, renvoie sa traduction en Morse (on laissera un espace entre deux lettres et deux espaces entre deux mots), puis écrire une fonction `decodage_morse` réciproque de la première.

```
>>> encodage_morse("alphabet morse")
'.- .-.. .--. .... .- -... . -  -- --- .-. ... .'
>>> decodage_morse(_)   # _ rappelle le résultat précédent
'alphabet morse'
```