Chap 11. Python 3 : Chaînes & listes

Livre p 73 – Chap 5 Tableaux

1. Chaînes de caractères

a. Indiçage

En Python, le type string (″str″) est une suite de caractères unicode, encadrés par des guillemets simples ('…') ou doubles (″…″).

Remarque : ″″ représente une chaîne de caractères vide.

On peut récupérer un caractère ou une sous-chaîne de la chaîne initiale grâce aux indices des caractères par une méthode appelée « slicing » (découpage en tranches)

Attention : L'indice du premier caractère est zéro !

Remarque : On peut aussi compter en négatif, –1 correspondant au dernier caractère.

- *len*(chaine) : permet de connaître la longueur de la chaîne.
- chaine[a] : renvoie le (a + 1)^{ème} caractère de la chaîne.
- chaine[a :b] : renvoie la sous-chaîne du (a + 1)^{ème} caractère au b^{ème}.
- chaine[a :] : renvoie la sous-chaîne du (a + 1)^{ème} caractère jusqu'à la fin.
- chaine[:b] : renvoie la sous-chaîne du début jusqu'au b^{ème} caractère.
- chaine[:b] : renvoie la sous-chaîne du début jusqu'au b^{ème} caractère.
- chaine[::n] : renvoie la sous-chaîne en prenant les caractères de n en n à partir du premier.

Exemples

mot = "bonjour"
len(mot) → 7
mot[3] → "j"
mot[-1] → "r"
mot[3:6] → "jou"
mot[5:] → "ur"
mot[:3] → "bon"
mot[::3] → "bjr"

b. Opérations

On peut utiliser certaines opérations et même les opérateurs de comparaison…

- ch_1 + ch_2 : concaténation des chaînes ch_1 et ch_2.
- ch * n : répétition de n fois la chaîne ch.
- ch_1 in ch_2 : teste si la chaîne ch_1 est incluse dans la chaîne ch_2.
- ch_1 < ch_2 : teste l'ordre alphabétique (ASCII) entre ch_1 et ch_2.
  Attention : Les majuscules sont classées avant les minuscules dans le code ASCII…

Exemples

"da" + "re" → "dare"
"da" * 3 → "dadada"
"da" in "tada" → True
"dans" < "do" → True

c. Méthodes

Les chaînes de caractères sont des objets sur lesquels on peut appliquer des méthodes.

- chaine.*lower*() : renvoie la même chaîne mais avec tous les caractères en minuscules
- chaine.*uper*() : renvoie la même chaîne mais avec tous les caractères en majuscules
- chaine.*strip*() : renvoie la même chaîne mais en supprimant tous les espaces présents au début et à la fin de la chaîne.
- chaine.*find*(mot) : renvoie le plus petit indice où l'on peut trouver la sous-chaîne « mot ».
  Remarque : renvoie –1 si le mot n'est pas inclus dans la chaîne.
- chaine.*count*(mot) : renvoie le nombre de fois où le mot est présent dans la chaîne.
- chaine.*replace*(ch_1, ch_2) : renvoie la même chaîne mais où chaque sous-chaîne « ch_1 » est remplacée par la sous-chaîne « ch_2 ».

d. Conversions

- *ord*(c) : renvoie la valeur unicode du caractère « c ».
- *chr*(n) : renvoie le caractère correspondant à la valeur unicode n.
- *str*(objet) : transforme l'objet en une chaîne de caractère.
- *int*(chaine) : transforme la chaîne de caractère en un nombre entier, si cela est possible !
- *float*(chaine) : transforme la chaîne de caractère en un nombre réel, si cela est possible !
- *eval*(chaine) : évalue la chaîne de caractère en effectuant les opérations, si cela est possible !
- chaine.*split*() : transforme la chaîne en une liste de sous-chaîne, en coupant au niveau des espaces
  Remarque : chaine.*split*(ch_1) permet d'utiliser n'importe quel sous-chaîne comme séparateur

2. Listes et t-uplets

a. Indiçage

En Python, les types liste (″list″) et t-uplet (″tuple″) sont des listes ordonnées d'objets divers.
Les listes, encadrées par des crochets […, …], sont modifiables alors que les t-uplets, encadrées par des parenthèses (…, …) ne sont pas modifiables !
Remarque : [] représente une liste vide.
On peut récupérer un élément ou une sous-liste de la liste initiale grâce aux indices des éléments par la méme méthode de « slicing » décrite dans la première partie.
Attention : L'indice du premier élément est zéro !
Remarque : On peut aussi compter en négatif, –1 correspondant au dernier élément.

Exemples
pair = [0, 2, 4, 6, 8]
len(pair) → 5
pair[3] → 6
pair[-1] → 8
pair[2:4] → [4, 6]
pair[3:] → [6, 8]
pair[:3] → [0, 2, 4]
pair[::2] → [0, 4, 8]

b. Opérations

On peut utiliser certaines opérations et même les opérateurs de comparaison…

- li_1 + li_2 : concaténation des listes li_1 et li_2.
- li * n : répétition de n fois la liste li.
- li_1 in li_2 : teste si la liste li_1 est incluse dans la liste li_2.
- li_1 < li_2 : teste l'ordre entre li_1 et li_2, en commençant par les 1^{ers} éléments, à conditions qu'ils soient de même type !
- *del* liste[a] : supprime le $(a + 1)^{ème}$ élément de la liste.
- *del* liste[a : b] : supprime les éléments à partir du $(a + 1)^{ème}$ jusqu'au $b^{ème}$ de la liste.
- *min*(liste) et *max*(liste) : renvoient le plus petit élément et le plus grand élément de la liste
  Attention : Les deux provoquent une erreur si tous les éléments ne sont pas du même type !

Exemples
[1, 2] + ["a", "b"]
→ [1, 2, "a", "b"]
[1, 2] * 3 → [1, 2, 1, 2, 1, 2]
"1" in [1, 2] → False
[1, 2, 3] < [1, 3] → True

c. Copie de listes

Pour copier une liste dans une autre liste, il faut procéder élément par élément ou utiliser l'instruction *deepcopy* présente dans le module *copy*.
Remarque : La commande « li_2 = li_1 » va simplement lier la liste li_2 à la liste li_1, toute modification sur une liste prendra effet sur l'autre liste, automatiquement …
On peut écrire : li_2 = [e *for* e *in* li_1] mais cela ne fonctionne pas avec les listes de listes.

d. Méthodes

Les listes sont des objets sur lesquels on peut appliquer des méthodes.

- liste.*index*(objet) : renvoie le plus petit indice où l'on peut trouver l'objet.
  Attention : Provoque une erreur si l'objet n'est pas présent dans la liste
- liste.*count*(objet) : renvoie le nombre de fois où l'objet est présente dans la liste

Attention : Les méthode suivantes modifient la liste sur laquelle elles agissent !

- liste.*reverse*() : inverse l'ordre des éléments de la liste (du dernier au premier).
- liste.*sort*() : trie les éléments de la liste dans l'ordre du plus petit au plus grand.
  Attention : Provoque une erreur si tous les éléments ne sont pas du même type !
  Remarque : *sorted*(liste) permet de faire un tri tout en créant une copie de la liste.
- liste.*append*(objet) : ajoute le nouvel objet à la fin de la liste.
- liste.*pop*(a) : renvoie le $(a + 1)^{ème}$ élément de la liste puis le supprime de la liste.
- liste.*remove*(objet) : supprime la première apparition de l'objet dans la liste.
  Attention : Provoque une erreur si l'objet n'est pas présent dans la liste

e. Conversions

- *list*(*map*(fct, liste)) : Crée une copie de la liste en appliquant la fonction à chacun des éléments
  Exemple : Si on a : li_1 = [1, 2, 3] et li_2 = *list*(*map*(*str*, li_1)) alors : li_2 = ["1", "2", "3"]
- chaine.*join*(liste) : Pour une liste de *str*, transforme la liste en une grande chaîne de caractères dans laquelle tous les éléments sont mis bout à bout avec le mot « chaine » entre chacun d'eux.