

Exercices – Récursivité et pile d'exécution

Exercice 1 – (Palindrome)

Un palindrome est un mot pouvant se lire à l'endroit comme à l'envers, comme « été » ou « radar ».

On veut écrire une fonction `est_palindrome` prenant en entrée une chaîne de caractères `mot` et renvoyant un booléen indiquant si ce mot est un palindrome ou non.

1. Les cas de base sont les cas où `mot` est de longueur 0 ou 1. Que doit renvoyer la fonction dans ces cas ?

|

2. Dans tous les autres cas, `mot` est un palindrome si :
 - la première et la dernière lettre du mot sont identiques ;
 - le reste du mot (sans la première et la dernière lettre) est un palindrome.

Sur quelle valeur l'appel récursif se fait-il ?

|

3. Écrire la fonction `est_palindrome` incluant ces différents cas.

|

Aide

Pour prendre une partie d'une chaîne de caractères, on peut faire ce qu'on appelle du *slicing*, c'est-à-dire un découpage de la chaîne entre deux indices : pour une chaîne `chaine`, on peut choisir de considérer `chaine[i:j]` qui prendra les éléments de `i` à `j-1` de `chaine`.

Exercice 2 – (Somme d'une liste)

Une fonction récursive sur un paramètre de type liste Python fonctionne de la même manière qu'avec un entier : il faut, à chaque nouvel appel, réduire le problème jusqu'au cas de base. C'est-à-dire ici, réduire la liste initiale.

1. Écrire une fonction itérative (utilisant une boucle) permettant de calculer la somme des éléments d'une liste de nombres `somme_it(lst)`. (Il est interdit d'utiliser la fonction Python `sum` !)

|

2. Pour quelle valeur de la liste est-il le plus facile de faire ce calcul, et que vaut-il dans ce cas ? (cas de base)

|

3. On peut faire du *slicing* sur les listes Python de la même manière que sur les chaînes de caractères.

(a) Comment prendre tous les éléments d'une liste sauf le premier ?

(b) En déduire l'appel récursif de la fonction `somme_rec` qui sera fait pour faire le calcul de la somme d'une liste de manière récursive.

4. Écrire la fonction récursive `somme_rec(lst)`.

Exercice 3 – (Le rendu de monnaie)

Le code suivant permet de rendre la monnaie avec l'algorithme glouton vu en classe de première. L'algorithme est cette fois-ci implémenté de manière récursive. Compléter le code :

```
1 def rendu_glouton(a_rendre, pieces):
2     """ rendu_glouton(int, list) -> list
3     a_rendre : la somme d'argent à rendre
4     pieces : une liste de pièces possibles à rendre """
5     piece_max = pieces[0]
6
7     if a_rendre == 0:
8         return .....
9
10    if ..... >= piece_max:
11        return ..... + rendu_glouton(a_rendre - piece_max, pieces)
12
13    return rendu_glouton(a_rendre, .....)
```

14

```
15 valeurs_possibles = [100, 50, 20, 10, 5, 2, 1]
16
17 assert rendu_glouton(67, valeurs_possibles) == [50, 10, 5, 2]
18 assert rendu_glouton(291, valeurs_possibles) == [100, 100, 50, 20, 20, 1]
19 # si on ne dispose pas de billets de 100 :
20 assert rendu_glouton(291, valeurs_possibles[1:]) == [50, 50, 50, 50, 50, 20, 20, 1]
```