

STAGE DE FORMATION

PARTIE II

Python : fonctions récursives

par

Valérien Eberlin

1. Questions flash

1. Quel est le type de l'expression $10+3*5$?

- ☐ a) (erreur) ☐ b) bool ☐ c) str ☐ d) float ☐ e) int

2. Quel est le type de l'expression `'toto' + 8*5` ?

- ☐ a) (erreur) ☐ b) bool ☐ c) str ☐ d) float ☐ e) int

3. Quel est le type de l'expression $1+8*5 == 8$?

- ☐ a) (erreur) ☐ b) bool ☐ c) str ☐ d) float ☐ e) int

4. On considère la fonction suivante :

```
def mystere(n : int) -> str :  
    if n % 3 == 0 or n % 5 == 0 :  
        if n % 3 == 0 :  
            resultat = "A"  
        else :  
            resultat = "B"  
    else :  
        if n % 5 == 0 :  
            resultat = "C"  
        else :  
            resultat = "D"  
    return resultat
```

a) Quelle est la valeur de `mystere(2)` ?

- ☐ a. "A" ☐ b. "B" ☐ c. "C" ☐ d. "D"

b) Quelle est la valeur de `mystere(6)` ?

- ☐ a. "A" ☐ b. "B" ☐ c. "C" ☐ d. "D"

c) Quelle est la valeur de `mystere(10)` ?

- ☐ a. "A" ☐ b. "B" ☐ c. "C" ☐ d. "D"

d) Est-il possible que le programme renvoie "C" ?

5. On rappelle que l'instruction « `append` » ajoute un élément à une liste. Par exemple si une liste `L` est `L = [3, 9]`, alors `L.append(4)` modifie la liste `L` en ajoutant l'élément 4 à la fin de la liste. La liste `L` devient donc : `L = [3, 9, 4]`.

On considère les programmes suivants où l'instruction « $a \% b$ » est le reste de la division euclidienne de a par b .

Programme A

```
L = [8, 77, 5]
if L[0] % 2 == 0:
    L.append("Sangha")
else :
    L.append("Pool")
```

Programme B

```
L = [8, 77, 5]
if L[0] % 2 == 0:
    L.append("Sangha")
L.append("Pool")
```

Sophie affirme que les deux programmes donnent le même résultat à la fin.

Qu'en pensez-vous ? Testez ces deux programmes à la main puis à l'aide d'un éditeur Python.

6. On définit le script suivant :

```
def fonction(mot : str, lettre : str) -> int:
    compteur = 0
    for caractere in mot :
        if caractere == lettre :
            compteur = compteur + 1
    return compteur

n = fonction('trololo', 'o')
```

Quelle sera la valeur de n après l'exécution de ce script ?

7. Un petit rappel : un dictionnaire est une structure de données particulièrement utile pour stocker des données contenant plusieurs champs. On crée un dictionnaire en indiquant les couples **clé / valeur** entre accolades.

Exemple d'un dictionnaire représentant les notes de Paul :

```
notes_paul{"Mathématiques" : 17 , "Anglais" : 15}
```

Les clés de ce dictionnaire sont "Mathématiques" et "Anglais" et les valeurs associées sont 17 et 15.

- Ouvrir l'éditeur EduPython puis Créer un dictionnaire vide nommée `departements_congo`.
- Ajouter la clé "André" qui pour valeur 225 puis ajouter les 12 départements du Congo dont les clés seront les noms des départements et les valeurs, leur chef-lieu.

- c) Supprimer la donnée "André" en utilisant l'instruction `del`.
- d) Afficher toutes les clés en utilisant l'instruction `keys()`.
- e) Afficher toutes les valeurs en utilisant l'instruction `values()`.

2. Récursivité

2. a. Activité

On considère les deux programmes ci-dessous :

Programme A

```
def somme(n):
    s = 0
    for i in range(1, n+1):
        s = s + i
    return s

print(somme(5))
```

Programme B

```
def somme(n):
    if n == 0:
        return 0
    return n + somme(n-1)

print(somme(5))
```

Sophie affirme que les deux programmes afficheront la même valeur après l'exécution des scripts.

Qu'en pensez-vous ? Testez à la main, puis à la machine ces deux programmes.

2. b. Définition

Une fonction récursive est une fonction qui fait appel à elle-même. On parle aussi d'algorithme récursif.

La fonction `somme(n)` du programme B de l'exemple ci-dessus est une fonction récursive.

Exercice 1

Avec le programme B, tester avec un éditeur Python `somme(-1)`. Qu'affiche le programme ? Pourquoi ?

Remarque

Un algorithme récursif doit conduire à un état d'arrêt c'est à dire qu'il doit y avoir une condition qui met fin aux appels récursifs. En d'autres termes, il doit y avoir un cas de base ou une condition d'arrêt qui, lorsqu'elle est remplie, empêche l'algorithme de continuer à s'appeler lui-même indéfiniment. Par exemple, la condition d'arrêt de l'algorithme récursif du programme B est réalisée lorsque le nombre atteint 0.

Le programme itératif A calcule la somme $0+1+2+ \dots + n$.

En constatant que :

$$\text{somme}(n) = \underbrace{0 + 1 + 2 + \dots + (n-1)}_{\text{somme } (n-1)} + n$$

on peut donc aborder le programme itératif A en définissant la fonction mathématique récursive :

$$\text{somme}(n) = \begin{cases} 0 & \text{si } n = 0 \\ n + \text{somme}(n-1) & \text{si } n > 1 \end{cases}$$

Ainsi,

$$\text{somme}(0) = 0$$

$$\text{somme}(1) = 1 + \text{somme}(0) = 1 + 0 = 1$$

$$\text{somme}(2) = 2 + \text{somme}(1) = 2 + 1 = 3$$

...

Exercice 2

Implémenter en Python une fonction récursive `puissance` admettant deux paramètres et qui renvoie $x^n = \underbrace{x \times x \times \dots \times x}_{n \text{ fois}}$

Exercice 3

On rappelle qu'on définit la factorielle d'un entier positif n par :

$$n! = 1 \times 2 \times \dots \times n$$

Implémenter en Python une fonction récursive `factorielle` qui renvoie $n!$.

Exercice 4

On considère le programme suivant :

```
def f(a,b):  
    if b == 1:  
        return a  
    return a + f(a, b-1)
```

1. Exécuter à la main `f(7, 8)`.

2. Conjecturer la valeur retournée par cette fonction pour deux entiers naturels non nul a et b quelconques.

Exercice 5

On considère le programme itératif ci-dessous :

```
def decTob(n, b):  
    assert (b > 1 and b < 17)  
    signes = ["0", "1", "2", "3", "4", "5", "6", "7", "8", "9", "A", "B", "C",  
             "D", "E", "F"]  
    # Cas de base  
    if n == 0:  
        return ""  
    # Récursion  
    return decTob(n // b, b) + signes[n % b]
```

1. a) Tester à la main ce programme pour $n=25$ et $b=2$.
1. b) Tester à la main ce programme pour $n=25$ et $b=3$.
2. Quel rôle joue ce programme

Voici une version itérative du programme récursif ci-dessus.

```
def decTob(n,b):  
    assert (b>1 and b<17)  
  
    signes=["0","1","2","3","4","5","6","7","8","9","A","B","C","D","E","F"]  
    mot = ""  
    while n!= 0:  
        mot = signes[n%b] + mot  
        n = n//b  
    return mot
```

Exercice 6

Un palindrome est un mot qui se lit dans les deux sens comme "radar".

Etant donné un string "mot", l'idée ici est de comparer la première lettre du string ("mot"[0]) avec la dernière lettre du string ("mot"[-1]) :

- s'il y a coïncidence, on poursuit en comparant la deuxième lettre ("mot"[1]) avec l'avant dernière lettre ("mot"[-2]) ;
- s'il y a coïncidence, on poursuit en comparant ("mot"[2]) avec l'avant dernière lettre "mot"[-3] ;
- ...
- jusqu'à parvenir à "la moitié" de la longueur du string "mot".

Compléter alors le programme itératif ci-dessous :

```
def est_palindrome(mot):  
    for i in range(len(mot)//2):  
        if mot[i] != mot[-i-1]:  
            return ...  
    return ...  
  
print(est_palindrome('kayak'))
```

On souhaite maintenant utiliser un programme récursif en Python pour vérifier si un mot est un palindrome.

On rappelle que ``mot[1:-1]`` permet de supprimer le premier et le dernier caractère. Par exemple : ``brazzaville[1:-1]`` affiche ``razzavill``.

Compléter le programme puis expliquer en quoi ce programme est-il récursif ?

```
def est_palindrome(mot):  
    # La fonction '[1:-1]' retire les lettres deux par deux. Cela  
    # signifie qu'en appliquant plusieurs fois cette fonction, il restera à  
    # la fin, soit une lettre (qui est un palindrome) soit un vide (qui est un  
    # string).  
    # Ces deux cas constituent donc les conditions d'arrêt  
    if len(mot) == 0:  
        return ...  
    if len(mot) == 1:  
        return ...  
    # Cas récursif : vérifier si le premier et le dernier caractère sont  
    # les mêmes  
    if mot[0] == ... :  
        # Appel récursif en supprimant le premier et le dernier  
        # caractère  
        return ...  
    else:  
        return ...  
  
# Exemple d'utilisation  
est_palindrome("radar")
```

Exercice 7

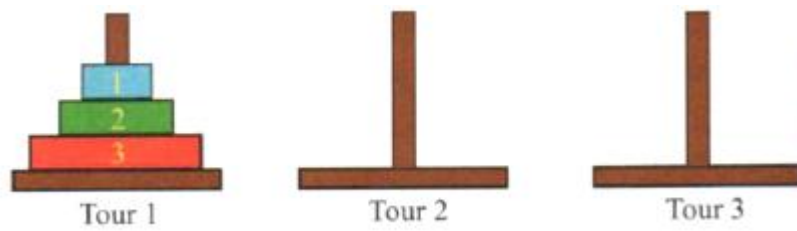
Les « tours de Hanoï » est un jeu imaginé par le mathématicien français *Édouard Lucas (1842-1891)*. Il consiste à déplacer n disques de diamètres différents d'une tour de « départ » à une tour d'« arrivée » en passant par une tour « intermédiaire » et ceci en un minimum de coups, tout en respectant les règles suivantes :

- on ne peut déplacer qu'un disque à la fois,
- on ne peut placer un disque que sur un autre disque plus grand que lui ou sur une tour vide.

Dans l'état initial, les n disques sont placés sur la tour « départ ».

Dans l'état final, tous les disques se retrouvent placés dans le même ordre sur la tour « arrivée ».

Ainsi, la configuration de départ pour $n = 3$ est la suivante :



1. Résoudre « à la main » le jeu pour $n = 3$.
2. Expliquer comment résoudre le jeu pour $n = 4$.
3. Pour déplacer n disques, on commence d'abord par déplacer les $n - 1$ disques supérieurs de la tour de départ à la tour d'arrivée sans toucher au n -ième disque le plus grand. Expliquer toute la suite de la procédure qui permet de résoudre le problème.
4. Dédire une fonction récursive :

```
```hanoi(n, depart, arrivee, intermediaire)```
```

Qui permet d'afficher tous les mouvements jusqu'à résolution complète du jeu, où ``depart`` désigne le numéro de la tour d'où provient un disque, ``arrivee`` celui de la tour vers laquelle on déplace le disque et ``intermediaire`` celui de la tour auxiliaire.