La récursivité

Langages et programmation

Contenu : Récursivité.
Capacités attendues : Écrire un programme récursif.
Analyser le fonctionnement d'un programme récursif.

NSI - Tle
TD de renforcement

1. Exercice 1

On rappelle qu'un nombre entier compris entre 0 et 127 peut être codé sur 8 bits. Voici une fonction mystère nommée myst2 qui prend en argument une **liste d'entiers naturels compris entre 0 et 127** :

```
def myst2(l: list) -> int:
    if l==[]:
        return 0
    else:
        l.pop(0)                    # on supprime le premier élément de la liste l
        return 8 + myst2(l)
```

a. Pourquoi cette fonction **myst2** est une fonction récursive ?
b. Tester cette fonction avec quelques listes, dont [3], [127], [3,7], [3,100,75,7,0].
c. Quel est le rôle de cette fonction **myst2** ?

2. Exercice 2

a. Écrire une fonction **récursive *somme*** qui prend comme argument un entier non nul n et qui renvoie la somme de tous les nombres entiers compris entre 1 et n.
b. Quelle est la condition d'arrêt de cette fonction récursive ?
c. Pourquoi pouvez-vous être certain.e.s que la situation de terminaison sera atteinte après un nombre fini d'appels récursifs ?
d. Rajouter une précondition.

3. Exercice 3

On considère désormais que les couples de lapins sont mortels de sorte que la suite ***ln*** est modifiée ainsi :
Si on note ln le nombre de lapins au bout de n mois, on peut modéliser le problème "pratique" par une suite (***ln***). Cette suite (ln) est définie par la **relation de récurrence** suivante :

$$l_n = \begin{cases} 1 & ,\text{si } n = 0. \\ 1 & ,\text{si } n = 1. \\ 2 & ,\text{si } n = 2. \\ 3 & ,\text{si } n = 3. \\ l_{n-1} + l_{n-2} - l_{n-4} & ,\text{si } n > 3. \end{cases}$$

Écrire une fonction récursive lapins qui prend comme argument un entier naturel n et qui renvoie le nombre de lapins ***ln***

4. Exercice 4

Le but est d'écrire une fonction récursive somme qui prend en argument une liste de nombres et qui renvoie la somme de ces nombres.
Quel cas simple peut correspondre à une condition d'arrêt pour cette fonction ?
Pour le cas général, proposez un appel récursif à la fonction somme où la liste est diminuée d'un élément.

Vous pouvez vous aider en utilisant le ***slicing*** :
liste[1:] correspond à la liste initiale où le premier élément a été ôté.

La récursivité

Langages et programmation

Contenu : Récursivité.
Capacités attendues : Écrire un programme récursif.
Analyser le fonctionnement d'un programme récursif.

Correction partielle :

```
def somme(liste: list) -> int:

    if len(liste) == 0:     # condition d'arrêt : liste vide

        return 0

    else:                   # cas général

        return liste[0] + somme(liste[1:])  # on ajoute à la première
valeur la somme de toutes les autres valeurs.
```

1. Testez votre fonction `somme`.

   Vous devez par exemple obtenir comme exécution de `somme([1,2,3])` le nombre 6.

2. Quelle suite de nombres entiers strictement décroissante peut être exhibée ici afin de justifier la terminaison de l'algorithme écrit ?

On veut obtenir une fonction `inverser` qui inverse une chaîne de caractères ; par exemple, `inverser("aBc45f!")` renvoie `"!f54cBa"`. Pour cela, la fonction extrait le premier élément puis fait un appel récursif (à elle-même) pour ajouter à la fin cet élément extrait. Voici une partie du script de cette fonction `inverser` :

On admet que `ch[1:]` correspond à la chaîne de caractère `ch` auquel le premier élément, celui `ch[0]`, a été ôté.
Par exemple, si `ch="aBc45f!"`, alors `ch[1:]` correspond à `"Bc45F!"`

```
def inverser(ch: str) -> str:

    n = len(ch)

    return inverser(ch[1:]) + ch[0]   # on ajoute le premier terme au
résultat de l'inversion du reste de la chaîne
```

1. Le script précédent est-il fonctionnel ? Pourquoi ?

> Si vous n'arrivez pas à répondre à la question, exécutez le code précédent et s'il ne fonctionne pas lisez l'éventuel message d'erreur renvoyé par l'interpréteur.

2. Rajoutez une condition d'arrêt à ce script pour le rendre fonctionnel.

> Pensez au cas le plus simple d'une chaîne de caractères à inverser.

3. Testez votre script augmenté d'une condition d'arrêt.

Exemple d'exécution à obtenir:

```
>>>inverser("aBc45f!"")

"!f54cBa"
```

On appelle **palindrome** un mot (ou une phrase) qui se lit de la même façon à de gauche à droite comme de droite à gauche, si on ne tient pas compte des espaces.
Exemples :

- "KAYAK" est un palindrome,
- La phrase "ESOPE RESTE ICI ET SE REPOSE" est aussi un palindrome, puisqu'en ôtant les espaces on a la chaîne devient "ESOPERESTEICIETSEREPOSE".
- Par contre, "AIMA" n'est pas un palindrome car "AIMA" est différent de "AMIA".

Voici une partie d'une fonction `est_palindrome` qui prend comme argument une chaîne de caractères (sans espace) et qui renvoie un booléen : `True` si le mot saisi comme argument est un palindrome, `False` sinon.

```
def est_palindrome(ch: str) -> bool:

    bool = True

    n = len(ch)

    if ch[0]==ch[n-1]:     # caractères extrêmes de la chaîne égaux

        ch2 = ch[1:n-1]    # chaîne de caractères extraites de ch où le
premier caractère (ch[0]) et le dernier (ch[n-1]) ne sont plus pris.

        return est_palindrome(ch2)

    else:

        bool = False

        return bool
```

Rajouter une condition d'arrêt à cette fonction et modifier le code précédent afin de rendre fonctionnelle la fonction `est_palindrome`.

LYCÉE BOISSY D'ANGLAS
LYCÉE DES MÉTIERS DE LA CRÉATION INDUSTRIELLE

La récursivité

Langages et programmation

Contenu : Récursivité.

Capacités attendues : Écrire un programme récursif.
Analyser le fonctionnement d'un programme récursif.

NSI - T^le^
TD de renforcement