Nom prénom :

Récursivité

NSI - Tle
DEVOIR
SURVEILLÉ

LYCÉE BOISSY D'ANGLAS
LYCÉE DES METIERS DE LA CREATION INDUSTRIELLE

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

1. Fonction mystère1

On définît en python la fonction ci-dessous :

```
def mystere1(a,b) :
    if b == 1 :
        return(a)
    else :
        return a*mystere1(a, b-1)

print(mystere1(5,4))
```

Cette fonction prend comme argument a et b, deux entiers naturels non nuls.

a. Que retourne **mystere1(5,4)** ? Justifier par un calcul.

mystere1(5,4)=625 ; en effet :
mystere1(5,4) a besoins de mystere1(5,3)
mystere1(5,3) a besoins de mystere1(5,2)
mystere1(5,2) a besoins de mystere1(5,1)
mystere1(5,1)=5
On peut donc calculer mystere1(5,2)=5* mystere1(5,1)=25
On peut donc calculer mystere1(5,3)=5* mystere1(5,2)=125
On peut donc calculer mystere1(5,4)=5* mystere1(5,3)=625

b. Indiquer où se situe l'appel récursif.

$a_x$mystere1(a,b-1)

c. Comment s'appelle le ***if b==1*** ?

Condition d'arrêt

d. Qu'est-ce qui garantit que le programme finira par s'arrêter ?

Car b diminue à chaque appel récursif

e. À quoi correspond mystere1(a,b) ?

a^b

2. Fonction mystère2

On rappelle qu'un nombre entier compris entre 0 et 127 peut être codé sur 8 bits. Voici une fonction mystère nommée **mystere2** qui prend en argument une **liste d'entiers naturels compris entre 0 et 127** :

```
def mystere2(l: list) :
    if l == [] :
        return 0
    else :
        l.pop(0)                # on supprime le premier élément de la liste l
        return 8 + mystere2(l)
```

a. Pourquoi cette fonction **mystere2** est une fonction récursive ?

Grâce :

1. **Au point d'arrêt (lorsque la liste est vide, on retourne 0)**
2. **À l'appel *8 + mystere2(liste)***

b. Quel sera le résultat de cette fonction avec les listes suivantes : [3], [127], [3,7], [3,100,75,7,0].

[3] : **8**

[127] : **8**

[3,7] : **16**

[3,100,75,7,0] : **40**

c. Quel est le rôle de cette fonction **mystere2** ?

Donner le nombre de bits nécessaires pour coder en binaire tous les nombres de la liste

3. Fonction somme

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.

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

4. Affichage des entiers de 1 à n dans l'ordre décroissant

Écrire une fonction récursive **descente(n)** affichant les entiers de 1 à n dans l'ordre décroissant.

```
def descente(n) :
  if n == 0 :
    return
  else :
    print(n)
    return descente(n-1)
```

5. Mélange des éléments d'une liste

On s'intéresse dans cet exercice à un algorithme de mélange des éléments d'une liste.

1. Pour la suite, il sera utile de disposer d'une fonction `echange` qui permet d'échanger dans une liste `lst` les éléments d'indice `i1` et `i2`.
Expliquer pourquoi le code Python ci-dessous ne réalise pas cet échange et en proposer une modification.

```
def echange(lst, i1, i2):
    lst[i2] = lst[i1]
    lst[i1] = lst[i2]
```

Cela fait l'égalité entre lst[i1] et lst[i2]

```
def echange(lst, i1, i2) :
    lst[i1], lst[i2] = lst[i2], lst[i1]
    return lst

print(echange([1,5,6,8], 1, 0))
```

2. La documentation du module `random` de Python fournit les informations ci-dessous concernant la fonction `randint(a,b)` :
`Renvoie un entier aléatoire N tel que a <= N <= b. Alias pour randrange(a, b+1).`
Parmi les valeurs ci-dessous, quelles sont celles qui peuvent être renvoyées par l'appel `randint(0, 10)` ?

`0` `1` `3.5` `9` `10` `11`

0 1 9 10

3. Le mélange de Fischer Yates est un algorithme permettant de permuter aléatoirement les éléments d'une liste. On donne ci-dessous une mise en œuvre récursive de cet algorithme en Python.

```
from random import randint

def melange(lst, ind):
    print(lst)
    if ind > 0 :
        j = randint(0, ind)
        echange(lst, ind, j)
        melange(lst, ind-1)
```

a. Expliquer pourquoi la fonction `melange` se termine toujours.

Car la fonction est appelée avec index-1 et se rappellera si index<0.
Elle s'arrêtera donc quand index sera égal à 0

b. Lors de l'appel de la fonction `melange`, la valeur du paramètre `ind` doit être égal au plus grand indice possible de la liste `lst`.
Pour une liste de longueur n, quel est le nombre d'appels récursifs de la fonction `melange` effectués, sans compter l'appel initial ?

Le nombre d'appels récursifs de la fonction `melange` effectués, sans compter l'appel initial sera de *n-1*

c. On considère le script ci-dessous :

```
lst = [v for v in range(5)]
melange(lst, 4)
```

On suppose que les valeurs successivement renvoyées par la fonction `randint` sont `2, 1, 2` et `0`.
Les deux premiers affichages produits par l'instruction `print(lst)` de la fonction `melange` sont :

```
[0, 1, 2, 3, 4]
[0, 1, 4, 3, 2]
```

Donner les affichages suivants produits par la fonction `melange`.

[0, 1, 2, 3, 4]
[0, 1, 4, 3, 2]
Puis :
[0, 3, 4, 1, 2]
[0, 3, 4, 1, 2]
[3, 0, 4, 1, 2]

d. Proposer une version itérative du mélange de Fischer Yates.

```
def melange_iteration(lst, ind) :
    while ind > 0 :
        print(lst)
        j = randint(0, ind)
        echange(lst, ind, j)
        ind -= 1
    return lst
```

Barrême :

1a	1
1b	0,5
1c	0,5
1d	0,5
1e	1
2a	1
2b	2
2c	0,5
3	3
4	3
5-1	2
5-2	1
5-3a	0,5
5-3b	0,5
5-3c	1
5-3d	2