NSI (TERMINALE) :

Programmation par récursivité

Vous Pouvez utiliser le fichier Jupyter ci-contre pour effectuer les exercices ci-dessous : Fiche2

Exercices *Fiche 2*

Exercices 1

L'un des exemples les plus classiques est une fonction mathématique, la fonction factorielle.
On dit « factorielle n » et on note avec un point d'exclamation n!.
Cette fonction est définie « par récurrence » sur les entiers naturels : $0! = 1$ et pour tout entier naturel n non nul, $n! = n \times (n - 1)!$.
Autrement dit, pour tout n non nul, $n! = n \times (n - 1) \times (n - 2) \times \ldots \times 2 \times 1$.

En utilisant le langage **Python**

1) Ecrire une fonction itérative calculant factorielle n, ayant pour paramètre un entier positif, et retourne le calcul.
   a) La fonction `fact_for(n)` pour la boucle « for »
   b) La fonction `fact_while(n)` pour la boucle « while»

2) Ecrire une fonction récursive `fact_rec(n)`, résolvant ce problème.

Exercices 2 **Calcul de pgcd de deux nombres entiers**

Écrire en Python une fonction récursive pgcd (a , b) renvoyant le plus grand diviseur commun de deux nombres a et b.
Pour cela, on utilisera le résultat mathématique suivant : « pgcd(a , b) = pgcd(b , r), où $a = bq + r$ »

Exercices 3 **Nombres d'adhérents**

Une association a remarqué que d'une année à l'autre,

- elle perd 5 % de ses adhérents ;
- elle gagne 200 adhérents.

1) En admettant que le nombre d'adhérents de cette association était égal à 2000 au Ier janvier 2019, écrire en Python une fonction récursive nommée `nombre(n)` affichant le nombre théorique d'adhérents après n années, $n \geq 1$.
2) Dans ce même programme, afficher le nombre théorique d'adhérents au cours des 20 prochaines années.

Exercices 4 **Suite de Fibonacci**

La suite de Fibonacci est la suite numérique définie par :

$$F_0 = F_1 = 1 \quad , \quad \text{pour tout } n \geq 0,\ F_{n+2} = F_{n+1} + F_n$$

Écrire en Python une fonction récursive `fibo(n)` permettant d'afficher le terme F_n , où $n \geq 0$, puis afficher les termes de F_0 à F_{20}.

Exercices 5

Les seules opérations autorisées sur les nombres entiers sont soit ajouter 1, soit retrancher 1. La fonction `somme` renvoie la somme des entiers positifs `a` et `b`.

```
def somme(a, b):
    while b > 0:
        a = a + 1
        b = b - 1
    return a
```

Ecrire une version récursive de cette fonction

Exercices 6

En s'inspirant de l'exercice précédent, il est demandé d'écrire une fonction qui renvoie la somme de deux entiers quelconques. Les seules opérations autorisées sur les nombres entiers sont ajouter ou retrancher 1.

1. Écrire une fonction `somme` itérative avec une boucle `while`.
2. Écrire une fonction `somme_rec` récursive non terminale.
3. Écrire une fonction `somme_rec_term` récursive terminale.

Indications : Pour effectuer la somme a + b dans les trois questions, distinguer les cas b > 0, b < 0 et b = 0.

Exercices 7

On reprend la fonction du cours permettant de savoir si un entier naturel est pair ou non.

```
def pair(n):
    while n > 0:
        n = n - 2
    return n == 0
```

Écrire une version récursive de cette fonction.

Exercices 8

1. Écrire une fonction récursive `puissance` qui prend en paramètres un flottant `x` non nul et un entier naturel `n` et qui renvoie x^n. On se base sur la définition mathématique : $x^0 = 1$ et $x^n = x \times x^{n-1}$ pour $n > 0$.
2. Dans la première question, on se contente de traduire la définition, et la version obtenue n'est pas récursive terminale. Modifier le code pour que la fonction soit récursive terminale. On peut s'inspirer des exemples du cours concernant la fonction factorielle.

Exercices 9

Il est demandé, comme à l'exercice précédent, d'écrire une fonction `puissance` qui prend en paramètres un nombre `x` et un entier naturel `n` et qui renvoie x^n. Mais la méthode est différente.

Pour cela, on note que $x^0 = 1$ et on remarque que si $n = 2k$, alors $x^n = x^{2k} = (x^k)^2$ et si $n = 2k + 1$ alors $x^n = x^{2k+1} = x(x^k)^2$. Le problème est divisé en deux sous-problèmes.

Exercices 10

1. Écrire une fonction récursive `somme` qui prend en paramètre une liste de nombres et renvoie la somme des termes de cette liste.
2. Expliquer avec l'exemple `somme([4,7,2])` le déroulement de l'exécution de la fonction.

Exercices 11

1. Expliquer quel est le résultat renvoyé par la fonction `mystere`.

```
def mystere(n):
    if n < 2:
        return str(n)
    else:
        return mystere(n//2) + str(n%2)
```

2. Écrire une fonction récursive `binaire` qui prend en paramètres un entier relatif `r` et un entier naturel `n` strictement positif, et qui renvoie la représentation en machine de `r` sur `n` bits. La méthode utilisée est celle du complément à deux et on n'écrira pas les 0 au début.
3. Compléter le code avec un compteur passé en paramètre afin d'obtenir un résultat composé de n caractères, donc avec éventuellement des 0 au début.

Exercices 12

Écrire une fonction récursive `inverse` qui prend en paramètre une chaîne de caractères `ch` et renvoie la chaîne obtenue en inversant l'ordre des caractères.
Par exemple, `inverse("azerty")` a pour valeur la chaîne `"ytreza"`.

Exercices 13

On dispose d'une fonction `nettoie` qui prend en paramètre une liste triée et élimine les éléments identiques. Par exemple, si la liste est `liste=[1,1,2,6,6,6,8,8,9,10]`, après l'instruction `nettoie(liste)`, cette liste a pour valeur `[1, 2, 6, 8, 9, 10]`.

```
def nettoie(L):
    n = len(L)
    k = 0
    while k < n - 1:
        if L[k] != L[k+1]:
            k = k + 1
        else:
            del L[k]
            n = len(L)
```

Écrire une version récursive de la fonction `nettoie`.

Indications : La fonction récursive prend deux paramètres, le liste et un indice k qui a pour valeur initiale par défaut 0