Code de partage avec Capytale : cf2b-7132688 **Corrigé**

Principe d'une fonction récursive

Une fonction récursive est une fonction qui s'appelle elle-même.
Il faut prendre garde à déterminer une condition d'arrêt pour que la procédure se termine. L'exemple-type d'une fonction récursive est celui des suites déterminées par une relation de récurrence du type : $u_{n+1} = f_n(u_n)$ où f_n est une fonction éventuellement dépendant de n. La condition d'arrêt correspond au point de départ : le u_0 dans les suites récursives.
L'objet de cette séance est de se familiariser avec la récursivité dans des cadres simples (et même si on pourrait se passer de récursivité pour certaines fonctions).

Exemple type : fonction factorielle

La fonction factorielle peut être définie de manière récursive :
$0! = 1$ (c'est le u_0) et $(n+1)! = (n+1) \times n!$ (c'est la relation $u_{n+1} = f(u_n)$)

```
def fact(n):
    if n==0:
        return 1
    else:
        return n*fact(n-1)
```

Exercice 1

1. Vérifier pour deux entiers n et p tels que $1 \leqslant p \leqslant n$: $\binom{n}{p} = \frac{n}{p}\binom{n-1}{p-1}$

   Par propriété :
   $\frac{n}{p}\binom{n-1}{p-1} = \frac{(n-1)!}{(p-1)!(n-1-(p-1))!} = \frac{n}{p}\frac{(n-1)!}{(p-1)!(n-p)!} = \frac{n(n-1)!}{p(p-1)!(n-p)!} = \frac{n!}{p!(n-p)!}$
   car $n(n-1)! = n!$ et $p(p-1)! = p!$ d'où le résultat.

2. Compléter la fonction récursive suivante permettant le calcul de $\binom{n}{p}$:

   Le programme part du principe qu'on calcule $\binom{n}{p}$ pour $n \geqslant p$ et de manière récursive à l'aide de la formule précédente. En diminuant n et p de une unité à chaque fois, c'est forcément « le coefficient du bas » qui atteindra 0 en premier. On utilise alors que $\forall k \in \mathbf{N}, \binom{k}{0} = 1$

   ```
   def CoefBin(p, n):
       if p==0:
           return 1
       else:
           return n/p*CoefBin(p-1, n-1)
   ```

Suites récurrentes

Exercice 2

On pose $u_0 \in [0,1]$ quelconque et, pour tout $n \in \mathbb{N}$, $u_{n+1} = e^{-u_n/2}$
On admettra que u converge vers un réel ℓ, et que, pour tout entier $n \in \mathbb{N}$, $|u_n - \ell| \leqslant 2^{-n}$

1. Compléter la fonction Python suivante permettant le calcul de u_n pour une valeur initiale `valeur_ initiale` (valeur de u_0) et une valeur n données en entrée :

   La fonction définit la suite de manière analogue à la définition mathématique de la suite : un premier terme et une relation récursive. Ici la fonction conserve u_0 (appelé `valeur_ initiale` dans le programme) à chaque rang pour pouvoir le faire évoluer lors de l'exécution du programme ensuite.

   ```
   def u(n, valeur_initiale):
       if n==0:
           return valeur_initiale
       else:
           return np.exp(-u(n-1, valeur_initiale)/2)
   ```

2. A l'aide de la majoration $\forall n \in \mathbb{N}$, $|u_n - \ell| \leqslant 2^{-n}$, compléter la fonction suivante permettant de trouver une valeur approchée de ℓ à 10^{-p} près pour le premier terme $u_0 = 0$

   Dès lors que 2^{-n} sera inférieur à la précision souhaitée, alors l'écart entre u_n et ℓ (qui vaut $|u_n - \ell|$ également, ce qui répondra à la question en renvoyant u_n (pour $u_0 = 0$).

   ```
   def valeur_approchee(p):
       n=0
       while 2**(-n)>10**(-p):
         n=n+1
       return u(n,0)
   ```

Exercice 3

On pose : $u_0 = 0$, $u_1 = 1$, et, pour $n \in \mathbb{N}$, $u_{n+2} = u_{n+1} - 2u_n$
Compléter la fonction Python suivante permettant le calcul du couple (u_{n-1}, u_n) pour une valeur $n \in \mathbb{N}^*$ donnée en entrée.

La fonction `Couple` doit renvoyer u_{n-1}, u_n et, à part pour $n = 1$ où le programme doit renvoyer $u_0, u_1 = 0, 1$, on calcule u_n à l'aide de la formule récursive :
$u_n = u_{n-1} - 2u_{n-2}$ et donc du résultat de `Couple(n-1)` qui contient u_{n-2}, u_{n-1}

```
def Couple(n):
    if n==1:
        return (0, 1)
    else:
        v, w=Couple(n-1)
        return w, w-2*v
# TEST
print(Couple(5))
```

Une autre manière de coder :
compléter la fonction suivante permettant de calculer u_n pour une valeur n donnée en entrée et deux valeurs initiales u_0 et u_1 :

Le programme se rapproche du premier de l'exercice 2, on calcule u_n (`suite(..., n)`) à l'aide de la formule récursive et donc de u_{n-1} (`suite(..., n-1)`) et u_{n-2} (`suite(..., n-2)`). Sauf pour les cas $n = 0$ ou $n = 1$ où l'on renvoie respectivement u_0 et u_1

```
def suite(valeur0, valeur1, n):
    if n==0:
        return valeur0
    elif n==1:
        return valeur1
    else:
        valeur=valeur1-2*valeur0
        return suite(valeur0, valeur1, n-1)-2*suite(valeur0, valeur1, n-2)

#TEST
print(suite(0, 1, 5))
```

Exercice 4 - suite de Syracuse

C'est la suite définie par : $u_0 \in \mathbb{N}$ donné, et pour tout $n \in \mathbb{N}$: $u_{n+1} = \begin{cases} \frac{1}{2}u_n & \text{si } u_n \text{ est pair} \\ 1+3u_n & \text{sinon} \end{cases}$

Pour tester si un nombre `x` est pair ou impair, on utilisera la commande `if 2*(x//2)==x` qui renvoie `True` si `x` est pair et `False` sinon.

Nota bene : on utilise par commodité ici la commande `//` qui permet de renvoyer un entier (le quotient de la division euclidienne). Cette commande n'est pas à connaitre.

Par exemple : tester les commandes

```
x=32
2*(x//2)==x

x=33
2*(x//2)==x
```

1. Ecrire une fonction récursive `def Syracuse(n, valeur_ initiale):` qui permet le calcul de u_n à partir de la valeur u_0 dans la variable `valeur_ initiale`.

   Exemple : la commande `Syracuse(9, 17)` doit renvoyer `8`

   ```
   def Syracuse(n, valeur_initiale):
       if n==0:
           return valeur_initiale
       else:
           u=Syracuse(n-1, valeur_initiale)
           if 2*(u//2)==u:
               return u//2
           else:
               return 3*u+1

   # pour tester
   print(Syracuse(9, 17))
   ```

   Une conjecture qui date des années 1950 et toujours non démontrée à ce jour consiste à dire que, quelle que soit la valeur initiale, la suite retombera nécessairement à la valeur 1

2. Compléter la fonction `def atterrissage(valeur_ initiale)` : qui renvoie le premier entier n pour lequel $u_n = 1$ à partir d'une valeur initiale donnée en entrée.

   ```
   def atterrissage(valeur_initiale):
       if valeur_initiale==1:
           return 0
       else:
           if 2*(valeur_initiale//2)==valeur_initiale:
               return 1+atterrissage(valeur_initiale//2)
           else:
   ```

```
        return 1+atterrissage(3*valeur_initiale+1)
```

Exemples :

`atterrissage(17)` doit renvoyer `12`
`atterrissage(32)` doit renvoyer `5`
`atterrissage(33)` doit renvoyer `26`