La récursivité

Définition
Une fonction récursive est une fonction qui s'appelle elle même lors de son exécution.

Activité 1

On considère le script suivant :

```
def gag(n):
   if n%2==0: a =gag(n+1)
  else: a=gag(n-1)
  return a

print(gag(6))
```

Exécuter-le : que se passe-t-il.

Il est donc indispensable de spécifier une condition d'arrêt.

Un algorithme récursif ressemble à la notion de récurrence en mathématiques. Il utilise le principe du « diviser pour régner ». L'appel récursif se fait sur un échantillon plus petit ou alors il faut être sûr que l'on ne va pas partir fait des appels sans fin.

Activité 2

On veut étudier la suite (u_n) définie par son premier terme $u_0=3$ et, pour tout entier naturel n, par la relation de récurrence $u_{n+1}=0{,}2u_n+1{,}4$.

1°/ Exécuter le script suivant avec rang = 3.

```
def suite(u0,n) :
    if n>0 :
        u = 0.2*suite(u0, n-1)+1.4
    else :
        u = u0
    return u

u0 = 3
rang = int(input("Indiquer le rang du terme à calculer"))
if rang<0 : rang=0
print("Le terme de rang", rang,"est", suite(u0, rang))
```

2°/ Modifier le script pour qu'il affiche tous les termes de la suite.

On peut écrire un script plus universel pour les suites de la forme $u_{n+1}=a\,u_n+b$.

```
def suite(u0,a, b, n) :
    if n>0 :
        u = a*suite(u0,a, b, n-1)+b
    else :
        u = u0
    return u

u0, a, b = 3, 0.2, 1.4
rang = int(input("Indiquer le rang du terme à calculer"))
if rang<=0 : rang=0
print("Le terme de rang", rang,"est", suite(u0,a, b, rang))
```

3°/ a) Essayer ce script avec $u_0=3$ u0 = 3, a = 0,2 et b = 2,4 et diverses valeurs de rang.
b) Que vaut u_n ?

Activité 3

Pour tout entier naturel n, on définit le nombre « factorielle de n », que l'on note n!, par :
0! = 1 et pour n > 0 $n!=1\times2\times3\times\cdots\times n$

1°/ Calculer 1!, 2!, 3! et 4!
2°/ Vérifier que pour tout entier naturel n non nul, on a : $n!=n\times(n-1)!$
3°/ En déduire l'écriture d'un script Python qui calcule la factorielle d'un entier naturel donné à l'aide d'une fonction récursive.

Activité 4

On considère la fonction suivante où a et b sont deux entiers naturels :

```
def mystere(a,b) :
    if b == 0 :
        return b
    elif b == 1 :
        return a
    else :
        return a + mystere(a, b – 1)
```

1°/ Que renvoie mystere(3, 5) ?

2°/ Que renvoie mystere(a, b) ?

Activité 5

Compléter la fonction suivante pour qu'elle retourne la somme des entiers naturels a et b

```
def somme(a,b) :
    if b == 0 :
        return . . . . . .
    else :
        return somme(. . . . . , b – 1)
```

Activité 6

On considère la fonction suivante :

```
def pal(chaine) :
    if len(chaine) <= 1 :
        return True
    if chaine[0]!= chaine[-1] :
        return False
    return pal(chaine[1 :-1])
```

1°/ Que renvoie pal("rever") ?

2°/ Que renvoie pal("avec") ?

3°/ Que renvoie pal("elle") ?

4°/ Que détecte cette fonction ?

Activité 7

On considère l'algorithme suivant :

On part d'une entier naturel non nul *n*.
Si *n* vaut 1 alors on s'arrête.
Sinon : si *n* est pair alors on le divise par 2
si *n* est impair alors on prend le successeur du triple de *n*
et on recommence avec ce nouvel entier

1°/ Essayer cet algorithme avec diverses valeurs de *n*.

2°/ Implémenter (en Pyhton) cet algorithme en utilisant une fonction récursive que l'on pourra appeler Syracuse…

3°/ Prouver que cet algorithme se termine (bon, attendez les vacances d'été pour vous attaquer à cette question...)

Activité 8 Les tours de Hanoï

On dispose de trois tours. Sur la première tour sont empilés, par ordre décroissant de diamètre, n disques avec $n>0$. L'objectif est de passer tous les disques sur la troisième tour avec la contrainte de ne pouvoir déplacer un disque que dans une tour vide ou sur un disque de diamètre plus grand.

1°/ Examinons trois cas simples :

a) S'il y a un seul disque. Décrire l'opération de transfert.
b) S'il y a deux disques. Décrire les opérations de transfert en minimisant le nombre de déplacements. Combien faut-il effectuer de déplacements ?
c) S'il y a trois disques. Décrire les opérations de transfert en minimisant le nombre de déplacements. Combien faut-il effectuer de déplacements ?

2°/ Reprendre l'étape 1°/c) en admettant que l'on dispose d'une fonction qui réalise l'étape 1°/b).

3°/ Décrire, en français, un algorithme récursif permettant de transférer tous les disques de la première tour à la troisième.

4°/ Implémenter cet algorithme.

5°/ Désignons par D(n-1) le nombre minimum de déplacements qu'il faut effectuer le déplacement de n-1 disques d'une tour à une autre.
Établir une relation de récurrence entre D(n) et D(n-1) puis résoudre cette récurrence.

Activité 9

On cherche à décrire les n-uplets ordonnés $(a_1, a_2, a_3, \cdots, a_n)$ avec $1 \leqslant a_1 < a_2 < a_3 < \cdots < a_n \leqslant B$ où B est un entier donné, supérieur à n.

Exemples :

avec B = 4 et n = 1 on a les singletons (1), (2), (3) et (4)
avec B = 4 et n = 2 on a les couples (1 ; 2), (1 ; 3), (1 ; 4), (2 ; 3), (2 ; 4) et (3 ; 4)
avec B = 4 et n = 3 on a les triplets (1 ; 2 ; 3), (1 ; 2 ; 4), (1 ; 3 ; 4) et (2 ; 3 ; 4)
avec B = 4 et n = 3 on a le quadruplet (1 ; 2 ; 3 ; 4)

Écrire un algorithme qui pour une borne B fixée et une valeur de n fixée (mais toutes deux fournies par l'utilisateur) donne la liste des n-uplets.

Hanoï

Principe de la récursivité :

On déplace les *n*-1 disques de la tour 1 vers la tour 2 ;
On déplace le disque restant (celui de plus gros diamètre) de la tour 1 vers la tour 3
On déplace les *n*-1 disques de la tour 2 vers la tour 3 ;

```
def Hanoi(nb_disques, source, destination) :
    if nb_disques>0 :
        Hanoi(nb_disques-1, source, 6 - (source + destination))
        print("Déplacer le disque de la tour", source, "vers la tour", destination)
        Hanoi(nb_disques-1, 6 - (source + destination), destination)

N = int(input("Nombre de disques sur la première tour :"))
Hanoi(N, 1, 3)
```