1BCPST2 Comment calculer $\sum_{k=p}^{q} a_k$ ou $\prod_{k=p}^{q} a_k$ avec une boucle for ?

Routine pour le calcul d'une somme : on initialise une variable à 0 puis on crée une boucle for à l'intérieur du bloc de laquelle on augmente la variable du terme général de la somme.

Routine pour le calcul d'un produit : on initialise une variable à 1 puis on crée une boucle for à l'intérieur du bloc de laquelle on multiplie la variable par le terme général du produit.

1 Lorsqu'on sait exprimer a_k en fonction de k

On notera `a` la fonction Python de paramètre k qui renvoie a_k.

Le programme suivant affiche $\sum_{k=p}^{q} a_k$:

```
p = eval(input('Entrer p : '))
q = eval(input('Entrer q : '))
s = 0
for k in range(p,q+1):
   s = s + a(k)  # qui peut aussi s'écrire s += a(k)
print(s)
```

La fonction Python `somme`, de paramètres entiers naturels p et q ($p \leqslant q$), renvoie $\sum_{k=p}^{q} a_k$:

```
def somme(p,q):
   """Renvoie ap + ... + aq"""
   s = 0
   for k in range(p,q+1):
      s += a(k)
   return s
```

Le programme suivant affiche $\prod_{k=p}^{q} a_k$:

```
p = eval(input('Entrer p : '))
q = eval(input('Entrer q : '))
pro = 1
for k in range(p,q+1):
   pro *= a(k)
print(pro)
```

La fonction Python `produit`, de paramètres entiers naturels p et q ($p \leqslant q$), renvoie $\prod_{k=p}^{q} a_k$:

```
def produit(p,q):
   """Renvoie ap ... aq"""
   pro = 1
   for k in range(p,q+1):
      pro *= a(k)
   return pro
```

Exemples.
On définit la fonction Python `factoriel` de paramètre n qui renvoie $n!$:

```
def factoriel(n):
   """Renvoie n!"""
   pro = 1                    # la variable pro est initialisée à 1
   for k in range(1,n+1): #k varie de1 à n. La commande range(1,n+1)nécessite que n soit de type entier
      pro = pro*k             # cette instruction peut aussi s'écrire pro *= k
   return pro
```

Si $n=0$ alors la boucle *for* est vide (`range(1,1)` est vide) donc `factoriel(0)` renvoie la valeur initiale de `pro` c'est-à-dire 1.

Le programme suivant affiche $\sum_{k=0}^{n} \frac{1}{k!}$:

```
n=eval(input('n = '))
s=0
for k in range(0,n+1):  # range(0,n+1) s'écrit aussi range(n+1)
     s += 1/factoriel(k)
print(s)
```

La fonction python `somfact` d'argument n renvoie $\sum_{k=0}^{n} \frac{1}{k!}$ sans utiliser la fonction `factoriel` et en minimisant les calculs :

```
def somfact(n):
    """renvoie 1/0! + 1/1! + ... + 1/n!"""
    s = p = 1 # on initialise p à 0! et s à 1/0!
    for k in range(1,n+1): # k varie de 1 à n
        p *= k # p passe de la valeur (k-1)! à la valeur k!
        s += 1/p # s passe de la valeur 1/0!+...+1/(k-1)! à la valeur 1/0!+...+1/k!
    return s
```

En appelant `somfact(n)` pour des valeurs de n égales à 0,1,2,3,4,5,6, vérifier que $\sum_{k=0}^{n} \frac{1}{k!}$ tend rapidement vers le nombre d'Euler e lorsque n tend vers $+\infty$ (e ≈ 2.718).

2 Lorsqu'on ne sait pas exprimer a_k en fonction de k

2.1 Lorsque la suite (a_n) vérifie la relation de récurrence $a_{k+1} = f(a_k)$

Soit (a_n) la suite définie par son premier terme $a0$ et la relation de récurrence $\forall n \in \mathbb{N},\ a_{n+1} = f(a_n)$.

On notera `f` la fonction Python de paramètre x qui retourne $f(x)$.

On procède comme à la section 1 après avoir codé la fonction Python `a` de paramètres $n, a0$ (premier terme) qui renvoie a_n :

```
def a(n, a0):
   """Renvoie an"""
   for k in range(n):
      a0 = f(a0)
   return a0
```

Les paramètres formels d'une fonction python (ici `n` et `a0`) sont des variables locales donc, à ce titre, leur contenu peut évoluer dans le déroulement du programme.

Exemple.

Première méthode.
On considère la fonction Python `a` de paramètres $n, a0$ qui renvoie a_n où (a_n) est la suite définie par son premier terme $a0$ et la relation de récurrence $\forall n \in \mathbb{N},\ a_{n+1} = \sqrt{a_n}$:

```
from math import sqrt   # la fonction sqrt renvoie la racine carrée du paramètre (square root)
def a(n, a0):
   """Renvoie an si pour tout k, a[k+1] = sqrt(ak)"""
   for k in range(n):
      a0 = sqrt(a0)
   return a0
```

On considère la fonction `somme_racine` de paramètres $p, q, a0$ qui renvoie $\sum_{k=p}^{q} a_k$:

```
def somme_racine(p, q, a0):
   """Renvoie ap + ... + aq si pour tout k, a[k+1] = sqrt(ak)"""
   s = 0
   for k in range(p, q+1):
      s += a(k, a0)
   return s
```

L'instruction `somme_racine(0,4,65536)` affiche la valeur de $\sum_{k=0}^{4} a_k$ où $a_0 = 65536$ et $\forall k \in \mathbb{N}$, $a_{k+1} = \sqrt{a_k}$.

Deuxième méthode : calcul concomitant des termes de la suite et des sommes partielles
On propose une méthode qui renvoie la même chose que `somme_racine` et qui effectue moins de calculs.

```
from math import sqrt
def somme_racine2(p, q, a0):
   """Renvoie ap + ... + aq si pour tout k, a[k+1] = sqrt(ak)"""
   s=0
   for k in range(p):
      a0 = sqrt(a0)            # cette boucle est vide si p=0
   for k in range(p, q+1):
      s += a0
      a0 = sqrt(a0)
   return s
```

2.2 Lorsque la suite (a_n) vérifie la relation de récurrence $a_{k+2} = f(a_{k+1}, a_k)$

Exemple.
On considère la suite (a_n) définie par ses deux premiers termes et par la relation $\forall k \in \mathbb{N}$, $a_{k+2} = \sqrt{a_{k+1}} + |\sin(a_k)|$.

Première méthode.
La fonction `f` de paramètres x, y renvoie $\sqrt{x} + |\sin(y)|$:

```
from math import*
def f(x,y):
   return sqrt(x)+abs(sin(y))
```

La fonction `a` de paramètres $n, a0, a1$ renvoie a_n :

```
def a(n,a0,a1):
   """Renvoie an où a[k+2] = sqrt(a[k+1])+abs(sin(ak))"""
   for k in range(n):
      a0, a1 = a1, f(a1, a0)
   return a0
```

La fonction `produit` de paramètres n, a_0, a_1 renvoie $\prod_{k=0}^{n} a_k$:

```
def produit(n,a0,a1):
   """Renvoie a0...an où a[k+2] = sqrt(a[k+1])+abs(sin(ak))"""
   pro = 1
   for k in range(n+1):
      pro *= a(k,a0,a1)
   return pro
```

Deuxième méthode : calcul concomitant des termes de la suite et des produits partiels
On propose une méthode qui renvoie la même chose que `produit` et qui effectue moins de calculs.

```
def produit2(n,a0,a1):
   """Renvoie a0...an où a[k+2] = sqrt(a[k+1])+abs(sin(ak))"""
   pro = 1
   for k in range(n+1):
      pro *= a0
      a0, a1 = a1, sqrt(a1)+abs(sin(a0))
   return pro
```

2.3 Dans les autres cas

Exemple.
On considère la suite (a_n) définie par son premier terme et par la relation de récurrence $\forall k \in \mathbb{N}$, $a_{k+1} = \dfrac{a_k}{k+1}$.

Première méthode.
La fonction `a` d'arguments $n, a0$ renvoie a_n :

```
def a(n, a0):
   """Renvoie an où a[k+1] = ak / (k+1)"""
   for k in range(n):
      a0 = a0 / (k+1)
   return a0
```

La fonction `somme` de paramètres n, a_0 renvoie $\sum_{k=0}^{n} a_k$:

```
def somme(n, a0):
   """Renvoie a0 + ... + an où a[k+1] = ak / (k+1)"""
   s=0
   for k in range(n+1):
      s += a(k, a0)
   return s
```

On peut utiliser la fonction `somme` dans un calcul comme n'importe quelle fonction mathématique de la bibliothèque `math` (exp, log, log10, cos, sin, tan, acos, asin, atan, sqrt, floor, ...).
L'instruction `somme(10,3)**5-exp(somme(4,3.9))` renvoie le nombre réel (flottant) `-2593.2555194593006` qui correspond à $\left(\sum_{k=0}^{10} u_k\right)^5 - \exp\left(\sum_{k=0}^{4} v_k\right)$ où (u_n) et (v_n) sont définies par $u_0 = 3$, $v_0 = 3,9$ et $\forall k \in \mathbb{N}$, $u_{k+1} = \dfrac{u_k}{k+1}$, $v_{k+1} = \dfrac{v_k}{k+1}$.

Deuxième méthode : calcul concomitant des termes de la suite et des sommes partielles
On propose une méthode qui renvoie la même chose que `somme` et qui effectue moins de calculs.

```
def somme2(n, a0):
   """Renvoie a0 + ... + an où a[k+1] = ak / (k+1)"""
   s = 0
   for k in range(n+1):
      s += a0
      a0 = a0 / (k+1)
   return s
```