Correction
DS 4 Mathématiques

Durée : 3 heures. Calculatrices et documents non autorisés.
Les exercices et les problèmes peuvent être traités indépendamment. Dans les problèmes, même s'il est toujours possible de sauter une question, il est bon de prêter attention à la progression des questions et à la structure globale du problème, et de vérifier que les calculs sont corrects avant de passer à la suite.
L'exercice indiqué informatique contient uniquement de l'informatique et sa note sera comptée sur le bulletin dans la matière informatique.

Exercice 1

Déterminer le domaine de définition des fonctions réelles suivantes.

$$f : x \longmapsto \arccos(1+\ln(x)) \qquad g : x \longmapsto \sqrt{2e^{2x}-5e^x+3} \tag{1}$$

$$h : x \longmapsto \frac{1}{\lfloor 2x \rfloor} \qquad i : x \longmapsto \tan(3x) + \sqrt[3]{\arctan(x)} \tag{2}$$

Correction. f Il faut $x > 0$ pour que $\ln(x)$ soit défini. De plus arccos est défini sur $[-1,1]$ donc il faut, comme c'est une composition, $1+\ln(x) \in [-1,1]$. Cela donne $-1 \leqslant 1+\ln(x) \leqslant 1$ donc $-2 \leqslant \ln(x) \leqslant 0$. Et comme ln est strictement croissante, de réciproque exp, cela donne $e^{-2} \leqslant x \leqslant 1$ (ceci implique automatiquement $x > 0$ car $e^{-2} > 0$). Conclusion :

$$\mathcal{D}_f = [e^{-2}, 1]$$

g La fonction est définie si $2e^{2x}-5e^x+3 \geqslant 0$ car on prend la racine carrée de ceci. Posons $y = e^x$, cela revient à résoudre $2y^2-5y+3 \geqslant 0$. Ici $\Delta = 5^2 - 4\times 2\times 3 = 25-24 = 1$. Les racines sont $y = \frac{5\pm 1}{4}$ soit $y = 1$ et $y = \frac{3}{2}$. Donc le signe est positif si $y \leqslant 1$ ou si $y \geqslant \frac{3}{2}$. Comme $y = e^x$ alors il faut $y > 0$ et $x = \ln(y)$. Alors si $y \leqslant 1$ cela donne $x \leqslant 0$ et si $y \geqslant \frac{3}{2}$ cela donne $y \geqslant \ln(\frac{3}{2})$. Conclusion :

$$\mathcal{D}_g =]-\infty, 0] \cup [\ln(\tfrac{3}{2}), +\infty[$$

h La fonction **n'est pas** définie si $\lfloor 2x \rfloor = 0$. Mais cette condition signifie exactement $0 \leqslant 2x < 1$ soit $0 \leqslant x < \frac{1}{2}$. Conclusion :

$$\mathcal{D}_h = \mathbb{R} \setminus [0, \tfrac{1}{2}[=]-\infty, 0[\cup [\tfrac{1}{2}, +\infty[$$

i Pour que la fonction soit définie il faut que **les deux** membres soient définis. À gauche $\tan(3x)$ est défini s'il n'existe pas de $k \in \mathbb{Z}$ tel que $3x = \frac{\pi}{2} + k\pi$ c'est à dire $x = \frac{\pi}{6} + \frac{k\pi}{3}$. À droite la fonction arctan est définie sur $\mathbb{R}$ et à valeurs dans $]-\frac{\pi}{2}, \frac{\pi}{2}[$, puis on en prend la racine cubique mais celle-ci est définie sur $\mathbb{R}$ tout entier, donc le terme de droite est défini sur $\mathbb{R}$ sans problème. Conclusion :

$$\mathcal{D}_i = \mathbb{R} \setminus \{\tfrac{\pi}{6} + \tfrac{k\pi}{3} \mid k \in \mathbb{Z}\}$$

□

Exercice 2

Dans les exercices de combinatoire, comme d'habitude, on ne cherche pas à donner des valeurs numériques mais on exprime son résultat en fonction de produits, quotients, puissances, factoriels. Par contre, les réponses doivent être justifiées.
Un groupe d'élèves de BCPST s'installe pour son TP en salle informatique contenant 16 postes. On s'intéresse à la question : de combien de façons peut-on installer chacun des élèves à un poste informatique ? Dans chacune des situations suivantes :

1. Il y a 12 élèves. Certains postes seront inoccupés.

   Correction. On range les élèves à la queue. Le premier choisit son poste parmi 16. Le deuxième élève choisit son poste parmi 15. Etc. Le 12-ème élève choisit parmi 5 postes restants. Le nombre de choix possible est

   $$16 \times 15 \times \cdots \times 5 = \frac{16!}{4!}$$

   □

2. Il y a 16 élèves. Chacun a droit à sa place seul.

   Correction. Cas particulier de la méthode précédente, le résultat est directement 16!. □

3. Il y a 17 élèves. Deux élèves seront donc nécessairement en binôme sur un même poste.

   Correction. On va d'abord choisir lesquels des deux élèves sont en binôme. Comme il y a 17 élèves le nombre de choix possible est $\binom{17}{2} = \frac{17\times16}{2}$. Ensuite on leur fait choisir leur poste en premier (pourquoi pas), il y a 16 choix. Puis on fait choisir leurs postes aux 15 élèves restants parmi les 15 postes restants... en résumé : pour chaque choix du binôme, pour chaque choix du poste sur lequel se place le binôme, il y aura encore 15 élèves à placer parmi 15 postes. Le nombre total de choix est donc

$$\binom{17}{2} \times 16 \times 15! = \frac{17\times16}{2} \times 16 \times 15! = \frac{17\times16\times16\times15!}{2} = \frac{16\times17!}{2}$$

   Un autre point de vue est le suivant : on range les élèves à la queue, chacun des 16 premiers choisit un poste comme dans la question précédente, soit 16! choix possible. Puis le dernier élève choisit au hasard une place parmi ces 16 là, et celle où il s'installera sera alors automatiquement un binôme. Le total de choix devient $16 \times 16!$. Mais alors :
   — Cela impose que c'est le 17-ème élève dans la queue qui est en binôme, et il n'y a pas de raison pour cela. Il faut donc multiplier par les 17 choix possibles pour l'élève qui choisit en dernier, soit un total de choix de $17 \times 16 \times 16! = 16 \times 17!$.
   — Chaque configuration est alors comptée deux fois : celle ou le dernier élève (disons qu'il s'appelle A) s'installe et forme donc un binôme avec B qui avait choisi sa place tout seul avant, et celle où au même poste c'est A qui s'était installé d'abord et que B a choisi aussi en tant que 17-ème élève. Il faut donc diviser par 2.
   On retrouve le même résultat. □

4. Il y a bien 17 élèves mais un élève nommé X ne veut pas être en binôme.

   Correction. On fait s'asseoir X d'abord, il a 16 postes possibles. Puis il reste 16 élèves à installer à 15 postes, en formant parmi eux un binôme. C'est donc le résultat de la question précédente (mais avec un élève et un poste de moins) qui donne donc $\frac{15\times16!}{2}$. Donc notre résultat maintenant est :

$$16 \times \frac{15\times16!}{2} = \frac{16\times15\times16!}{2}$$

   Autre point de vue : on forme un binôme parmi les 16 élèves autres que X, soit $\binom{16}{2}$ possibilités. Puis on choisit la place de chacun, soit au total $\binom{16}{2} \times 16!$. □

5. Il y a 17 élèves, et les élèves nommés X et Y peuvent éventuellement être en binôme mais certainement pas ensemble.

   Correction. On fait s'installer d'abord X puis Y (pourquoi pas), donc X a 16 postes possibles et comme Y ne se met pas en binôme avec lui il a 15 postes possibles. Il faut ensuite compter séparément :
   (a) Si ni X ni Y ne font partie d'un binôme : cela revient à installer 15 élèves sur les 14 postes restants en formant parmi eux un binôme. Le nombre de choix est donc (encore une fois comme dans les questions précédentes) $\frac{14\times15!}{2}$.
   (b) Si c'est X qui fait partie d'un binôme (et il n'y a qu'un seul binôme) : il faut choisir un élève parmi les 15 qui sera avec X, puis installer 14 élèves restants sur les 14 autres postes. Le nombre de choix est donc $15 \times 14!$ qui est aussi égal à 15! (on installe 15 élève sur les 15 places qui sont toutes sauf celle où est déjà installé Y).
   (c) De même si c'est Y et non pas X qui est en binôme, et on trouve encore 15!.

   Au total on trouve donc (ce sont 3 cas disjoints)

$$16 \times 15 \times \left(\frac{14\times15!}{2} + 15! + 15!\right) = \frac{16\times15\times15!}{2}(14+4) = \frac{18\times15\times16!}{2} = 15\times9\times16!$$

   Autre point de vue : il y a un total de $\binom{17}{2}$ binômes possibles, et parmi ceux-là un seul est composé de X et Y ensemble, il y en a donc $\binom{17}{2} - 1$ possibles où X et Y ne sont pas ensemble. Puis on fait choisir sa place à tout le monde, donnant $\left(\binom{17}{2} - 1\right) \times 16!$. On trouve que $15 \times 9 = \binom{17}{2} - 1 = 135$, c'est bien le même résultat. □

Exercice 3

Montrer que l'application suivante est bijective et donner sa bijection réciproque.

$$\begin{aligned}\varphi : \mathbb{R}^3 &\longrightarrow \mathbb{R}^3\\ (x,y,z) &\longmapsto (x+y+3z, x-y+2z, x+2y-z)\end{aligned} \tag{3}$$

Correction. Soit $(a,b,c) \in \mathbb{R}^3$, cela revient à montrer qu'il existe une et une seule solution à l'équation $\varphi(x,y,z) = (a,b,c)$ puis, si c'est bien le cas, exprimer (x,y,z) en fonction de (a,b,c). Cela revient à étudier puis résoudre le système linéaire suivant :

$$(S) : \begin{cases} x + y + 3z = a \\ x - y + 2z = b \\ x + 2y - z = c \end{cases}$$

On commence donc l'application du pivot de Gauss, d'abord avec $L_2 \leftarrow L_2 - L_1$ et $L_3 \leftarrow L_3 - L_1$ pour éliminer x dans les lignes L_2 et L_3 :

$$(S) \underset{\substack{L_2 \leftarrow L_2 - L_1 \\ L_3 \leftarrow L_3 - L_1}}{\iff} \begin{cases} x + y + 3z = a \\ -2y - z = b - a \\ y - 4z = c - a \end{cases} \underset{\substack{L_2 \leftarrow -L_2 \\ L_2 \leftrightarrow L_3}}{\iff} \begin{cases} x + y + 3z = a \\ y - 4z = c - a \\ 2y + z = a - b \end{cases}$$

Le pivot se poursuit avec $L_3 \leftarrow L_3 - 2L_2$ pour éliminer y dans L_3 :

$$(S) \underset{L_3 \leftarrow 2L_2}{\iff} \begin{cases} x + y + 3z = a \\ y - 4z = c - a \\ 9z = (a-b) - 2(c-a) = 3a - b - 2c \end{cases}$$

À ce stade le système est échelonné : il est de rang 3 en 3 variables, il n'y a pas de condition de compatibilité ni d'inconnue auxiliaire, donc il existe une unique solution $(x,y,z) \in \mathbb{R}^3$ sans condition sur $(a,b,c) \in \mathbb{R}^3$. Donc l'application φ est bijective. Il reste à exprimer (x,y,z) en fonction de (a,b,c), en remontant le système échelonné. On trouve d'abord :

$$L_3 : z = \frac{3a-b-2c}{9}$$

puis en remplaçant

$$L_2 : y = (c-a) + 4z = \frac{9(c-a) + 4(3a-b-2c)}{9} = \frac{3a-4b+c}{9}$$

et enfin

$$L_1 : x = a - y - 3z = \frac{9a - (3a-4b+c) - 3(3a-b-2c)}{9} = \frac{-3a+7b+5c}{9}$$

On en déduit que l'application φ a pour réciproque

$$\begin{aligned}\varphi^{-1} : \mathbb{R}^3 &\longrightarrow \mathbb{R}^3\\ (a,b,c) &\longmapsto \left(\frac{-3a+7b+5c}{9}, \frac{3a-4b+c}{9}, \frac{3a-b-2c}{9}\right)\end{aligned}$$

□

Exercice 4 (informatique)

On cherche à écrire des fonctions Python pour étudier les anagrammes. Pour simplifier, on s'intéresse uniquement aux anagrammes de nombres écrits avec les chiffres de 0 à 9, qu'on représente par la liste de leurs chiffres. Ainsi le nombre 154 est représenté par la liste `[1, 5, 4]`, et un exemple d'anagramme est le nombre 451 représenté par `[4, 5, 1]`. On considère qu'un nombre peut très bien avoir pour premier chiffre 0, ainsi le nombre 2023 représenté par `[2, 0, 2, 3]` a bien 0223 pour anagramme, représenté par `[0, 2, 2, 3]` ; cela ne fait que simplifier le problème.

1. Écrire une fonction `compte(L, x)` qui compte combien de fois le chiffre `x` apparait dans la liste `L` représentant un nombre. En particulier la fonction retourne simplement 0 si le chiffre n'apparait pas.

   Correction. Il suffit d'écrire une fonction comprenant une variable `c`, avec une boucle, et la variable est initialisée à 0 et augmente de 1 chaque fois que le terme `L[i]` est égal à `x`.

   ```
   def compte(L, x):
       c = 0
       for i in range(len(L)):
           if L[i] == x:
               c = c + 1
       return c
   ```

Ici on peut aussi directement itérer sur les éléments :

```
...
for y in L:
    if y == x:
....
```

□

2. Écrire une fonction `compte_tout(L)` qui retourne une **liste** `C` de longueur 10, où `C[i]` est égal au nombre de fois où apparait le chiffre `i` dans la liste `L`.

   Correction. On a besoin d'une liste de taille 10 remplie de zéros, et le plus pratique est donc de l'initialiser avec la syntaxe `C = [0] * 10`. La façon naïve est simplement d'utiliser la fonction précédente :

   ```
   def compte_tout(L):
       C = [0] * 10
       for i in range(10):
           C[i] = compte(L, i)
       return C
   ```

   mais on peut aussi être plus efficace : on parcourt une seule fois toute la liste `L`, et si `L[j]` est égal à `i` alors on peut augmenter `C[i]` de 1 comme dans le question précédente :

   ```
   def compte_tout(L):
       C = [0] * 10
       for j in range(len(L)):
           i = L[j]
           C[i] = C[i] + 1
       return C
   ```

   et idem ici on peut directement itérer sur les éléments de `L`. □

3. Écrire une fonction **récursive** `factoriel(n)` qui prend en argument un entier n positif et retourne le nombre $n!$.

   Correction. Question de cours ! Sans commentaires.

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

   □

4. Compléter la fonction suivante pour qu'elle retourne le nombre d'anagrammes qu'on peut former à partir du nombre représenté par la liste `L`.

   ```
   def nombre_anagrammes(L):
       C = compte_tout(L)
       n = len(L)
       numerateur = ...
       denominateur = ...
       for i in range(10):
           ...
       return numerateur // denominateur # division en nombres entiers
   ```

   Correction. Le nombre d'anagrammes sera un quotient de $n!$ (où n est la longueur de la liste, donc le nombre de chiffres du nombre considéré) et il faut diviser par un produit : quand un chiffre apparait plusieurs fois, on divise par la factorielle du nombre de fois où il apparait. En fait, comme $0! = 1$ et $1! = 1$, le dénominateur est simplement le produit sur les chiffres i de la factorielle du nombre de fois que le chiffre i apparait dans le nombre, compté ici dans `C[i]`. Cela donne donc tout simplement le programme suivant, où `numerateur` vaut d'un seul coup $n!$ et où `denominateur` est une variable accumulatrice pour le produit :

```
def nombre_anagrammes(L):
    C = compte_tout(L)
    n = len(L)
    numerateur = factoriel(n)
    denominateur = 1
    for i in range(10):
         denominateur = denominateur * factoriel(C[i])
    return numerateur // denominateur
```

Éventuellement, si on ne s'intéresse qu'aux chiffres qui apparaissent plusieurs fois (ou si la fonction `factoriel(n)` qu'on a écrit ne traite par correctement les cas $n = 0$ ou $n = 1$) on obtient à l'intérieur de la boucle

```
for i in range(10):
    if C[i] >= 2:
        denominateur = denominateur * factoriel(C[i])
```

□

5. On souhaite écrire une fonction qui teste si un anagramme donné vérifie la condition qu'il n'y a jamais deux chiffres consécutifs égaux. C'est une fonction qui doit renvoyer la valeur booléenne `True` s'il **n'y a pas** de chiffres consécutifs égaux, et qui doit renvoyer `False` sinon.
   Un élève de BCPST1A écrit la fonction suivante :

```
def non_consecutifs_1A(L):
    for i in range(len(L)):
        if L[i] != L[i+1]: # les deux chiffres consécutifs sont différents
            return True # alors c'est vrai
        else:
            return False # sinon ils sont égaux : c'est donc faux
```

   (a) Quel(s) problème(s) pose cette fonction ? Que renvoie `non_consecutifs_1A([5, 3, 3, 1])` ?

   (b) La corriger et écrire une fonction `non_consecutifs_1B(L)` qui renvoie bien `True` si et seulement si il n'y a pas deux chiffres consécutifs égaux.

   Correction. La fonction teste bien si les deux chiffres consécutifs sont différents. Or, si c'est le cas, à cause du `return`, la fonction s'arrête… éventuellement dès le début : `non_consecutifs_1A([5, 3, 3, 1])` renvoie `True` car au départ $i = 0$, elle teste donc que 5 est différent de 3, et s'arrête. En fait, si la condition `L[i] != L[i+1]` est vérifiée, cela ne signifie pas qu'il faut renvoyer `True` mais il faut simplement continuer la boucle, c'est seulement dans le cas contraire qu'il faut s'arrêter ! Il faut donc penser à l'envers et le `return True` est **hors** de la boucle. C'est l'erreur classique qu'ont fait tous les élèves au début, en fait les 1B…
   Au passage, il y a un problème d'indices dans le `range` : puisqu'on écrit `L[i+1]` il faut que `i` s'arrête un cran avant. La fonction peut provoquer des erreurs du type `index out of range`.
   Bref, la fonction corrigée est :

```
def non_consecutifs_1B(L):
    for i in range(len(L)-1): # attention, on va écrire L[i+1]
        if L[i] == L[i+1]: # on a trouvé deux chiffres consécutifs égaux
            return False # fini ! c'est faux
    # sinon on est arrivé au bout de la boucle sans trouver de contre-exemple
    return True # donc c'est vrai
```

□

6. Que teste la fonction suivante ?

```
def tres_serieux(L, M):
    A = compte_tout(L)
    B = compte_tout(M)
    for i in range(10):
        if A[i] != B[i]:
            return False
    return True
```

   Correction. La boucle teste à chaque itération si `L` a autant de fois le chiffre i que `M`. Si non, elle s'arrête en renvoyant `False`. Mais si elle arrive au bout de la boucle sans avoir été arrêtée, c'est à dire si pour chaque valeur de i entre 0

et 9 les listes `L` et `M` contiennent autant de fois le chiffre i, alors elle renvoie `True`. Bref, la fonction teste si les deux nombres représentés par les listes `L` et `M` contiennent le même nombre de fois chaque chiffre.

... Ce qui signifie en fait exactement que ces deux nombres sont des anagrammes l'un de l'autre ! Puisqu'on inclut qu'un compte de 0 pour le chiffre i signifie que le chiffre i n'apparait pas dans le nombre, deux nombres sont anagrammes simplement quand les comptes de chaque chiffres sont égaux, sans avoir à tester par de nouvelles boucles si chaque chiffre de l'un est présent dans l'autre etc. Donc la fonction renvoie `True` si `L` et `M` sont anagrammes l'un de l'autre, et `False` sinon. Et ça c'est du très sérieux ! □

Problème

Pour tout entier $n \geqslant 1$ et tout entier $p \geqslant 1$, on cherche à dénombrer les applications surjectives de $[\![1, n]\!]$ dans $[\![1, p]\!]$. Il s'agit aussi du nombre d'applications surjectives de n'importe quel ensemble à n éléments dans un ensemble à p éléments. On le note $S_{n,p}$.

Partie 1

1. En listant simplement les applications de $[\![1, n]\!]$ dans $[\![1, p]\!]$, déterminer les valeurs de $S_{1,1}$, $S_{1,2}$, $S_{2,1}$ et $S_{2,2}$.

   Correction. (a) $n = 1$, $p = 1$: il n'y a qu'une seule application qui est $1 \mapsto 1$ et celle-ci est surjective. Donc $S_{1,1} = 1$.

   (b) $n = 1$, $p = 2$: il y a deux applications. qui sont $1 \mapsto 1$ et $1 \mapsto 2$. Or aucune n'est surjective : la première n'atteint pas la valeur 2, et la deuxième n'atteint pas la valeur 1. Donc $S_{1,2} = 0$.

   (c) $n = 2$, $p = 1$: il y a une seule application, qui envoie 1 et 2 sur 1. Celle-ci est surjective. Donc $S_{2,1} = 1$.

   (d) $n = 2$, $p = 2$: il y a 4 applications, à lister :

   $$\varphi_1 : 1 \mapsto 1, 2 \mapsto 1 \qquad \varphi_2 : 1 \mapsto 1, 2 \mapsto 2$$
   $$\varphi_3 : 1 \mapsto 2, 2 \mapsto 1 \qquad \varphi_4 : 1 \mapsto 2, 2 \mapsto 2$$

   Ainsi écrites, φ_2 et φ_3 sont surjectives, mais pas φ_1 (ne prend pas la valeur 1) ni φ_4 (ne prend pas la valeur 2). Donc $S_{2,2} = 2$.

   □

2. Que vaut $S_{n,p}$ si $p > n$?

   Correction. Il s'agit d'applications surjectives d'un ensemble vers un ensemble **plus gros** et donc il n'y en a pas. Ainsi $S_{n,p} = 0$ dans ce cas. □

3. Que vaut $S_{n,p}$ si $p = n$?

   Correction. Dans ce cas les ensembles de départ et d'arrivée sont les mêmes, égaux à $[\![1, n]\!]$. Une application φ de cet ensemble dans lui-même est donc surjective si et seulement si elle est bijective. Donc il s'agit tout simplement du nombre de permutations de $[\![1, n]\!]$, qui est $n!$. □

4. Déterminer $S_{n,1}$.

   Correction. Il n'y a qu'une seule application de $[\![1, n]\!]$ dans $[\![1, 1]\!]$ car ce dernier est le singleton $\{1\}$, et cette application est constante. Elle est aussi surjective, car 1 est bien l'image de n'importe lequel des éléments de départ. Bref, $S_{n,1} = 1$. □

5. (a) Soit $n \geqslant 1$. Quelles sont les applications de $[\![1, n]\!]$ dans $[\![1, 2]\!]$ qui ne sont pas surjectives ?

   Correction. Soit une application de $[\![1, n]\!]$ dans $[\![1, 2]\!]$ prend à la fois les valeurs 1 et 2. Soit il y en a une de ces deux là qu'elle ne prend pas. Si elle ne prend pas 1 alors c'est en fait une application constante à la valeur 2 (il n'y a qu'une seule telle application constante). Et de même si elle ne prend pas 2 alors c'est une application constante à la valeur 1. Bref, il n'y a que deux applications dans ce cas qui ne sont pas surjectives. □

   (b) En déduire la valeur de $S_{n,2}$ pour tout entier $n \geqslant 1$.

   Correction. Le nombre d'applications de $[\![1, n]\!]$ dans $[\![1, 2]\!]$ est exactement 2^n. Parmi toutes ces applications seules 2 ne sont pas surjectives. Donc $S_{n,2} = 2^n - 2$. □

Partie 2

Le but est de démontrer la relation de récurrence (4), que l'on pourra éventuellement admettre pour passer à la partie suivante.
On fixe un couple d'entiers (n,p) tels que $n \geqslant p \geqslant 2$ et on désigne par $\mathscr{S}$ l'ensemble des surjections de $[\![1,n]\!]$ dans $[\![1,p]\!]$. On exprimera les réponses aux questions 6b et 7b en fonction de p, $S_{n-1,p}$ et $S_{n-1,p-1}$.

6. (a) Soit $\varphi \in \mathscr{S}$. On pose $\beta = \varphi(1)$ et on suppose que β admet un seul antécédent par φ. Justifier que l'application $\psi_1 : [\![2,n]\!] \to [\![1,p]\!] \setminus \{\beta\}$, $x \mapsto \varphi(x)$ est surjective.

Correction. D'abord cette application est bien définie car si $x \in [\![2,n]\!]$ alors $x \neq 1$ donc $\varphi(x) \neq \beta$ (car β n'a qu'un seul antécédent) donc ψ_1 (obtenue par restriction de φ) est bien à valeurs dans $[\![1,p]\!] \setminus \{\beta\}$.
Soit alors $y \in [\![1,p]\!] \setminus \{\beta\}$. Alors comme φ est surjective, y admet un antécédent x dans $[\![1,n]\!]$ par φ. Il s'agit de montrer que $x \neq 1$. Mais $\varphi(1) = \beta$ et $y \neq \beta$. Bref, x est un antécédent de y par ψ_1. □

(b) En déduire une expression du cardinal de $\mathscr{S}_1 = \{\varphi \in \mathscr{S} \mid \text{Card}\{x \in [\![1,n]\!] \mid \varphi(x) = \varphi(1)\} = 1\}$.

Correction. Il s'agit exactement des applications $\varphi \in \mathscr{S}$ telles que $\varphi(1)$ admet un unique antécédent. Comme on l'a vu, une fois fixé $\varphi(1)$, cela détermine uniquement une application surjective d'un ensemble à $n-1$ éléments vers un ensemble à $p-1$ éléments. Et il y a p choix possibles pour $\varphi(1)$ (les p éléments de $[\![1,p]\!]$). Dans l'autre sens, étant donné l'un des éléments $\beta \in [\![1,p]\!]$ et une application $\psi_1 : [\![2,n]\!] \to [\![1,p]\!] \setminus \{\beta\}$ on peut définir une application $\varphi \in \mathscr{S}$ qui étend ψ en posant $\varphi(1) = \beta$, et alors $\varphi \in \mathscr{S}_1$. Bref :

$$\text{Card}(\mathscr{S}_1) = p \times S_{n-1,p-1}$$

□

7. (a) Soit $\varphi \in \mathscr{S}$. On pose $\beta = \varphi(1)$ et on suppose que β admet au moins deux antécédents par φ. Justifier que l'application $\psi_2 : [\![2,n]\!] \to [\![1,p]\!]$, $x \mapsto \varphi(x)$ est surjective.

Correction. L'application ψ_2 est bien définie car c'est simplement la restriction à $[\![2,n]\!]$ de φ. Soit alors $y \in [\![1,p]\!]$. Comme φ est surjective, y admet au moins un antécédent x par φ et cet antécédent est dans $[\![1,n]\!]$. Mais si jamais $x = 1$ alors c'est que $y = \beta$ et il y a **un autre** antécédent, disons $x' \neq 1$ avec $\varphi(x') = y$. Bref, soit x est déjà dans $[\![2,n]\!]$ soit on choisit x' qui, lui, est dedans. Et ceci démontre que ψ_2 est surjective. □

(b) En déduire une expression du cardinal de $\mathscr{S}_2 = \{\varphi \in \mathscr{S} \mid \text{Card}\{x \in [\![1,n]\!] \mid \varphi(x) = \varphi(1)\} \geqslant 2\}$.

Correction. Il s'agit exactement des applications $\varphi \in \mathscr{S}$ telles que $\varphi(1)$ admet au moins deux antécédents. Comme juste avant, une fois fixé $\beta \in [\![1,p]\!]$ (p choix possibles) alors φ défini une application ψ_2 surjective entre un ensemble à $n-1$ éléments et un ensemble à p éléments. Réciproquement, partant d'une application surjective $\psi : [\![2,n]\!] \to [\![1,p]\!]$ et du choix d'un élément $\beta \in [\![1,p]\!]$ on peut bien compléter ψ en une application $\varphi \in \mathscr{S}$ en posant $\varphi(1) = \beta$. Bref, on trouve

$$\text{Card}(\mathscr{S}_2) = p \times S_{n-1,p}$$

□

8. Montrer que :

$$S_{n,p} = p(S_{n-1,p} + S_{n-1,p-1}). \tag{4}$$

Correction. Bien sûr, $\mathscr{S} = \mathscr{S}_1 \cup \mathscr{S}_2$ et cette union est disjointe, car une application $\varphi \in \mathscr{S}$ vérifie soit la condition d'être dans $\mathscr{S}_1$ ($\varphi(1)$ admet un unique antécédent) soit celle d'être dans $\mathscr{S}_2$ (φ_1 admet au moins deux antécédents). Cela implique donc $\text{Card}(\mathscr{S}) = \text{Card}(\mathscr{S}_1) + \text{Card}(\mathscr{S}_2)$ d'où

$$S_{n,p} = p \times S_{n-1,p-1} + \times S_{n-1,p} = p(S_{n-1,p-1} + S_{n-1,p})$$

□

Partie 3

On applique maintenant la relation (4).

9. Écrire une fonction Python **récursive** `surjections(n, p)` qui calcule $S_{n,p}$ par cette formule de récurrence, pour tous $n \geqslant 1$ et $p \geqslant 1$.

Correction. Bien sûr, il y aura une ligne `return surjections(n-1, p-1) + surjections(n-1, p)`. Mais il faut faire un petit peu attention à la condition d'arrêt pour ne pas produire de boucle infinie. Si n et p sont bien tous les deux supérieurs à 1 alors la récurrence peut s'arrêter soit si $n = 1$ (alors $S_{1,p} = 1$ si $p = 1$ et 0 sinon) soit si $p = 1$ (alors $S_{n,1} = 1$ toujours). Donc attention à la logique et l'ordre de test des conditions ! On voit qu'il est plus facile de tester p d'abord.

```
def surjections(n, p):
    if p == 1:
        return 1
    elif n == 1: # ici n=1 et p>1
        return 0
    else:
        return surjections(n-1, p-1) + surjections(n-1, p)
```

□

10. Pensez-vous que la fonction ainsi écrite va être rapide à calculer pour l'ordinateur ? Justifier.

Correction. Non ce n'est pas très efficace car il y un un phénomène d'explosion de la pile d'appels : par exemples `surjections(5, 3)` va appeler `surjections(4, 2)` et `surjections(4, 3)` pour calculer son résultat ; puis d'un côté `surjections(4, 2)` va appeler `surjections(3, 2)` et `surjections(3, 1)`, alors que de l'autre côté `surjections(4, 3)` va appeler `surjections(3, 2)` et `surjections(3, 3)` ; ici `surjections(3, 2)` est appelé deux fois, tout ce qui en descend va être appelé encore inutilement de multiples fois... Avec des nombres de l'ordre de 30 la fonction peut rapidement provoquer trop d'appels et le programme sera lent ou ne terminera pas. □

11. Démontrer la formule, pour $n \geqslant 1$ et $p \geqslant 1$

$$S_{n,p} = (-1)^p \sum_{k=1}^{p} (-1)^k \binom{p}{k} k^n. \tag{5}$$

soigneusement par récurrence en posant l'hypothèse

$$P(n) : \qquad \forall p \geqslant 1, \quad S_{n,p} = (-1)^p \sum_{k=1}^{p} (-1)^k \binom{p}{k} k^n \tag{6}$$

Correction. D'abord $P(n)$ est donnée, avec $n \geqslant 1$. Pour $P(1)$ il s'agit de montrer :

$$P(1) : \qquad \forall p \geqslant 1, \quad S_{1,p} = (-1)^p \sum_{k=1}^{p} (-1)^k \binom{p}{k} k$$

Or on a vu que $S_{1,p} = 1$ si $p = 1$ et 0 sinon. Si $p = 1$ la formule ci-dessus donne simplement 1 (car $(-1)^p = 1$, et sous la somme $k = 1$ donc le terme vaut -1, donc il reste $-(-1)$), donc c'est vrai pour $p = 1$. Sinon, c'est une formule du type pion qui permet de simplifier d'abord

$$\sum_{k=1}^{p} (-1)^k \binom{p}{k} k = \sum_{k=1}^{p} (-1)^k \binom{p-1}{k-1} p = p \sum_{k=1}^{p} (-1)^k \binom{p-1}{k-1}$$

puis un changement d'indice de type simple décalage, disons $\ell = k - 1$, donne

$$\sum_{k=1}^{p} (-1)^k \binom{p-1}{k-1} = \sum_{\ell=0}^{p-1} (-1)^{\ell+1} \binom{p-1}{\ell} = -\sum_{\ell=0}^{p-1} (-1)^{\ell} \binom{p-1}{\ell}$$

et là on reconnait avec le binôme de Newton $(1-1)^p$ qui fait bien 0 si $p > 1$. Donc :

$$(-1)^p \sum_{k=1}^{p} (-1)^k \binom{p}{k} k^n = (-1)^p \times p \times -(1-1)^p = 0$$

et ceci termine l'initialisation de la récurrence.

Pour l'hérédité maintenant : soit $n \geqslant 1$, supposons $P(n)$. Montrons $P(n+1)$. Soit $p \geqslant 1$.

Il faut d'abord distinguer le cas où $p = 1$, alors $S_{n+1,p} = 1$ et exactement comme tout à l'heure la somme de droite vaut 1 si $p = 1$. Donc $P(n+1)$ est vérifiée si $p = 1$. Sinon, on utilise la formule de récurrence (4) et l'hypothèse de récurrence $P(n)$:

$$S_{n+1,p} = p(S_{n-1,p-1} + S_{n-1,p}) = p \times \left((-1)^{p-1} \sum_{k=1}^{p-1} (-1)^k \binom{p-1}{k} k^n + (-1)^p \sum_{k=1}^{p} (-1)^k \binom{p}{k} k^n \right)$$

On voit qu'on veut faire apparaitre une formule de type Pascal pour regrouper les deux sommes. Mais tenant compte du fait que $(-1)^{p-1} = -(-1)^p$ on trouve en regroupant :

$$S_{n+1,p} = p \times (-1)^p \times \left(-\sum_{k=1}^{p-1} (-1)^k \binom{p-1}{k} k^n + \sum_{k=1}^{p} (-1)^k \binom{p}{k} k^n \right)$$

Dans la somme de gauche on rajoute artificiellement un terme nul pour aller jusqu'à $k = p$, et donc on peut tout regrouper :

$$S_{n+1,p} = p \times (-1)^p \times \sum_{k=1}^{p} \left((-1)^k \times k^n \times \left(-\binom{p-1}{k} + \binom{p}{k} \right) \right)$$

Mais alors la formule de Pascal $\binom{p-1}{k} + \binom{p-1}{k-1} = \binom{p}{k}$ donne ici sous la somme $\binom{p}{k} - \binom{p-1}{k} = \binom{p-1}{k-1}$ et donc

$$S_{n+1,p} = p \times (-1)^p \times \sum_{k=1}^{p} \left((-1)^k \times k^n \times \binom{p-1}{k-1} \right)$$

Il s'agit alors de faire rentrer le p sous la somme et d'appliquer la formule du pion : $p\binom{p-1}{k-1} = k\binom{p}{k}$ qui donne donc

$$S_{n+1,p} = (-1)^p \times \sum_{k=1}^{p} \left((-1)^k \times k^n \times k \times \binom{p}{k} \right)$$

Puis tenant compte de $k^n \times k = k^{n+1}$, c'est fini. □

12. En déduire une fonction Python itérative (avec des boucles, mais pas de récursivité) `surjections2(n, p)` qui calcule $S_{n,p}$. On supposera qu'on dispose déjà d'une fonction `binome(k, n)` qui calcule le coefficient binomial $\binom{n}{k}$.

Correction. Ce n'est plus qu'un calcul de somme.

```
def surjections2(n, p):
    s = 0
    for k in range(1, p+1):
        s = s + (-1)**k * binome(k, p) * k**n
    return (-1)**p * s
```

Un poil plus efficace informatiquement (un calcul de puissance est en général une opération lourde), ne pas calculer des $(-1)^k$ ni de $(-1)^p$ mais tester la parité pour savoir s'il faut additionner ou soustraire.

```
def surjections2(n, p):
    s = 0
    for k in range(1, p+1):
        x = binome(k, p) * k**n
        if k % 2 == 0:
            s = s + x
        else:
            s = s - x
    if p % 2 == 0:
        return s
    else:
        return -s
```

□