Planche 1
Agro-Véto 2024

Exercice sans préparation.

1. Écrire en Python une fonction `occ` qui prend en argument un nombre flottant `x` et une liste `L` et qui renvoie le nombre d'occurrences de `x` dans `L`.

2. a. Écrire en Python une fonction qui prend en argument deux entiers `m` et `n` et un flottant `p`$\in [0,1]$ (inutile de le vérifier), et qui renvoie un tableau de taille $(1,m)$ dont les coefficients sont des réalisations de variables aléatoires indépendantes suivant la loi binomiale $\mathcal{B}(n,p)$.

   b. Écrire en Python une fonction qui prend en argument une série statistique sous forme d'une liste `L` et qui renvoie un tableau de taille $(2,n)$ dont la première ligne est l'ensemble des différentes valeurs de la série statistique et dont la deuxième ligne est composée des nombres d'occurrences de chacune des valeurs.

Corrigé

Planche 2
Agro-Véto 2024

Exercice sans préparation.

1. Écrire en Python une fonction sans argument qui renvoie les résultats du lancer de deux lancers équilibrés (un vert et un rouge).

2. Il y a six joueurs numérotés de 1 à 6. Au début, ils ont chacun le même nombre de jetons.

   Lors d'une manche de jeu, on lance le dé vert. Le résultat désigne le joueur portant le numéro du dé. On fera attention qu'il lui reste des jetons. On lance ensuite le dé rouge et on calcule la somme des résultats des deux dés. Si ce résultat est le numéro d'un joueur (donc inférieur ou égal à 6), alors le joueur désigné par le dé vert lui donne un jeton, sinon il enlève son jeton du jeu.

   Écrire une fonction qui prend en argument une liste avec le nombre de jetons de chaque joueur et qui renvoie la liste des jetons restants de chaque joueur après une manche.

Corrigé

Planche 3
Agro-Véto 2024

Exercice sans préparation.

1. Une urne contient 10 boules indiscernables au toucher. Sur chacune est écrit une lettre : il y a 3 A, 3 T, 2 C et 2 G.

   On effectue n tirages avec remise. Écrire en Python une fonction qui renvoie la chaine des n caractères successivement tirés.

2. Écrire en Python une fonction boolénne `contient_avec_erreur` prenant en argument deux chaines de caractères qui renvoie `True` si, et seulement si, la première chaine de caractères contient la seconde à une erreur de caractère près. On supposera que la longueur de la première chaine est supérieure ou égale à celle de la seconde.

   Par exemple `contient_avec_erreur("ATCGG","ATT")` *devra renvoyer* `True`.

Corrigé

Planche 4
Agro-Véto 2024

Exercice sans préparation.

1. Écrire une fonction `somme` qui prend en argument une liste `L` et deux indices `i<j` et qui renvoie la somme des éléments de l'indice `i` inclus à l'indice `j` exclu.

2. Écrire une fonction `sous_liste_long` qui prend en argument une liste `L` et un entier k et qui renvoie la plus grande somme des éléments de sous-listes de `L` (d'éléments consécutifs) de longueur k.

Corrigé

Planche 5
Agro-Véto 2024

Exercice sans préparation.

1. Écrire en Python une fonction prenant en argument une liste de nombres deux à deux distincts et qui renvoie le maximum et le minimum de cette liste ainsi que leurs indices respectifs.

2. Écrire en Python une fonction prenant en argument une liste de nombres deux à deux distincts et qui renvoie le maximum et le deuxième maximum (la deuxième valeur la plus grande) de cette liste ainsi que leurs indices respectifs.

Corrigé

Planche 6
Agro-Véto 2024

Exercice sans préparation.

1. Écrire une fonction `extract` prenant en argument une liste `L` de couples (i, j) et deux entiers n et p et qui renvoie une liste de couples (i, j) - éléments de `L` - tels que $0 \leqslant i < n$ et $0 \leqslant j < p$.

   Par exemple `extract([(1,2),(5,3),(9,2)], 6, 3)` devra renvoyer `[(1,2)]`.

2. Écrire une fonction prenant en argument une matrice M, et deux indices de ligne et de colonne i et j et renvoyant la liste des éléments de M voisins de celui d'indices (de ligne et colonne) i et j.

Corrigé

Planche 7
Agro-Véto 2024

Exercice sans préparation.
On considère un nuage de points $((x_i, y_i))_{1 \leqslant i \leqslant n}$.

1. Écrire une fonction qui prend en argument les coordonnées de deux points A et B du nuage et qui renvoie la distance AB entre ces deux points.

2. Écrire une fonction qui prend argument la liste des coordonnées d'un nuage de points et qui renvoie les coordonnées du point moyen du nuage.

3. Écrire une fonction qui renvoie l'indice i du point (x_i, y_i) du nuage le plus proche du point moyen.

Corrigé

Planche 8
Agro-Véto 2024

Exercice sans préparation.

1. Écrire en Python une fonction d'argument une liste `L` qui teste si la liste `L` vérifie l'hypothèse $\mathcal{H}$ suivante

   $\mathcal{H}$: les éléments de la liste `L` sont des entiers compris (au sens large) entre 0 et $n-1$, où n désigne la longueur de `L`.

2. Écrire en Python une fonction d'argument une liste `L` vérifiant l'hypothèse $\mathcal{H}$ et qui teste si la liste `L` vérifie que l'hypothèse $\mathcal{H}'$: `L` est une permutation de $[\![0, n-1]\!]$ où n est la longueur de `L`, i.e. :

   $$\mathcal{H}' : \text{la liste } \texttt{L} \text{ contient exactement une fois chaque valeur comprise entre 0 et } n-1.$$

3. Combien de listes vérifient que `L[0] = 0` lorsqu'on permute les éléments de la liste $L = [0, \ldots, n-1]$ ($n \in \mathbb{N}^*$) ?

Corrigé

Planche 9
Agro-Véto 2024

Exercice sans préparation.
Soient a et b deux entiers naturels tels que $a < b$.

1. Écrire en Python une fonction `verif(L,a,b)` qui a pour paramètres une liste `L` d'entiers naturels et deux entiers `a` et `b` et qui renvoie `True` si tous les éléments de `L` sont dans l'intervalle $[\![a, b]\!]$ et `False` sinon.

2. Soit `L` une liste donc chaque élément est dans l'intervalle d'entiers $[\![a, b]\!]$.

   Écrire en Python une fonction `denombre(L,a,b)` qui détermine le nombre d'apparitions de chacune des valeurs possibles (entre a et b) de la liste `L` et qui renvoie le résultat dans une liste dont le premier élément représentera le nombre de a dans `L`, le deuxième le nombre de $a + 1$ dans `L`, etc.

   Par exemple, pour $a = 0$ *et* $b = 4$, `denombre([1,3,0,4,1,3,1],0,4)` *renverra* `[1,3,0,2,1,0,0]`.

Corrigé

Planche 10
Agro-Véto 2024

Exercice sans préparation.

1. Écrire une fonction Python, d'arguments d'entrée `i` et `N`, qui simule une marche aléatoire sur $\mathbb{Z}$: le marcheur démarre sur l'entier `i` et à chaque pas, il avance de 1 ou recule de 1 avec probabilité $\frac{1}{2}$. La marche s'arrête après le `N`-ème pas et la fonction renvoie l'entier sur lequel le marcheur s'est arrêté.

2. Un plateau de jeu est constitué de $n+1$ cases numérotées de 0 à n avec $n > 1$, les cases 0 et n contenant des valeurs a et b, comme ci-dessous où $n = 10$, $a = 1$ et $b = 3$:

1										3
0	1	2	3	4	5	6	7	8	9	10

Pour $i \in \{0, 1, \ldots, n\}$, on considère le jeu J_i suivant : on place un pion dans la case de numéro i et tant que l'on n'est pas dans la case 0 ou dans la case n, on lance une pièce équilibrée : si elle tombe sur face, on se déplace vers la gauche ; sinon, on se déplace vers la droite. On admet que la partie se termine presque sûrement : le gain du joueur est la valeur (a ou b) contenue dans la case finale.

Écrire une fonction qui prend en paramètres les valeurs `n`, `a`, `b` et `i`, qui simule ce jeu et qui renvoie le gain.

Corrigé

Planche 11
Agro-Véto 2024

Exercice sans préparation.

On souhaite exploiter le suivi d'une randonnée en effectuant des mesures lors de différents points de passage : latitude, longitude, altitude et temps (date). Ces données sont contenues dans une liste `coords`, où `coords[i]` désigne la liste des 4 mesures effectuées au point de passage numéro `i`. Ainsi, `coords[i][2]` renvoie par exemple l'altitude relevée au point de passage numéro `i`.

1. Écrire une fonction `temps(coords)` qui renvoie la liste des temps relevés lors de la randonnée `coords`.

2. Sans utiliser la fonction **`max`**, écrire une fonction `plus_haut(coords)` qui renvoie la liste `[latitude, longitude]` du point le plus haut lors de la randonnée `coords` (*on le supposera unique*).

Corrigé

Planche 12
Agro-Véto 2024

Exercice sans préparation.

1. Écrire une fonction Python qui simule une série de N lancers d'une pièce équilibrée, et qui renvoie la liste des résultats de ces lancers ("Pile" est codé par 1, et "Face" par 0).

2. Écrire une fonction Python qui simule une série de lancers d'une pièce équilibrée jusqu'à l'obtention de la configuration "Pile, Pile, Face", et renvoie le nombre de lancers nécessaires à l'apparition de cette configuration. A l'aide de cette fonction, évaluer le temps moyen d'attente de cette configuration.

Corrigé

Planche 13
Agro-Véto 2023

Exercice sans préparation.

1. Écrire une fonction booléenne en Python qui prend en argument une liste `L` de nombres et deux flottants a et b (tels que a $<$ b) et qui renvoie `True`, si et seulement si tous les coefficients de `L` appartiennent à l'intervalle [a, b].

2. Écrire une fonction prenant en argument un entier n, une liste `L` dont tous les éléments sont des entiers qui appartiennent a $[\![0, n]\!]$ et qui renvoie une liste `occ` telle que `occ[k]` soit le nombre de fois où apparait l'entier k dans la liste `L`.

Corrigé

Planche 14
Agro-Véto 2023

Exercice sans préparation.

Dans la liste `[6,10,17,20,8,5,4,13]`, il y a changement de monotonie aux indices 3 et 6 (on ne précise que le plus petit indice).

1. Écrire une fonction `indice_chgt` qui prend en argument une liste `L` et un indice `k` (allant de 1 à `len(L)`-2) et qui renvoie `True` s'l y a un changement de monotonie à l'indice `k` et `False` sinon.

2. Écrire une fonction `monotonie_chgt` qui prend en argument une liste `L` et qui renvoie la liste des indices auxquels on observe un changement de monotonie.

Corrigé

Planche 15
Agro-Véto 2023

Exercice sans préparation.

1. Écrire une fonction donnant le second maximum d'une liste dont les éléments sont distincts.
2. Écrire une fonction prenant en argument une liste non triée L (d'éléments deux à deux distincts) et un de ses éléments x et qui renvoie sa position (son indice) dans L si la liste était triée dans l'ordre croissant.

Corrigé

Planche 16
Agro-Véto 2023

Exercice sans préparation.
On considère une urne avec p boules numérotées de 0 à $p-1$. On réalise n tirages avec remise.

1. Écrire une fonction qui prend en argument n et p et qui renvoie la liste des numéros des boules tirées lors de la réalisation des n tirages.

2. On note S la variable aléatoire égale au nombre de numéros distincts de boules que l'on a tirées. Écrire une fonction qui simule la réalisation de la variable aléatoire S.

3. Écrire une fonction qui donne une estimation de l'espérance de S.

Corrigé

Planche 17
Agro-Véto 2023

Exercice sans préparation.

1. Écrire une fonction prenant en argument un entier naturel n non nul et que renvoie $S_n = \sum_{k=1}^{n} \frac{1}{k^2}$.

2. On admet que $S_n \underset{n\to+\infty}{\sim} \frac{\pi^2}{6}$.

   Écrire une fonction sans argument et simulant la variable aléatoire X dont la loi est donnée par :

$$\forall k \in \mathbb{N}^*,\ \mathbb{P}(X = k) = \frac{6}{\pi^2 k^2}.$$

Corrigé

Planche 18
Agro-Véto 2023

Exercice sans préparation.

1. Écrire en Python une fonction qui simule n lancers d'une pièce équilibrée et qui renvoie un couple (p, f) où p et f désignent respectivement le nombre de pile et face obtenus.

2. Écrire une fonction qui simule une succession de lancers d'une pièce équilibrée et qui renvoie le nombre de lancers nécessaires pour obtenir consécutivement "pile", "pile" et "face".

Corrigé

Planche 19
Agro-Véto 2023

Exercice sans préparation.

1. Écrire une fonction booléenne en Python qui prend en argument une liste d'entiers de longueur supérieure ou égale à 3 et qui renvoie si ses éléments sont les termes d'une suite arithmétique.

2. Reprendre la question précédente avec une suite géométrique.

Corrigé

Planche 20
Agro-Véto 2023

Exercice sans préparation.

1. Une puce se déplace sur un axe gradué de la manière suivante : à chaque instant, elle saute d'une unité vers la gauche ou vers la droite avec la même probabilité.

   Écrire une fonction qui prend en argument l'abscisse initiale i de la puce et un entier n, et qui renvoie l'abscisse finale de la puce après n sauts.

2. On suppose désormais que la puce se trouve sur une règle graduée de n centimètres et que la longueur de chaque saut est d'un centimètre.

   Écrire une fonction qui prend en argument l'abscisse initiale de la puce et qui renvoie le nombre de sauts nécessaires pour arriver au bord de la règle et l'abscisse finale.

Corrigé

Planche 21
Agro-Véto 2023

Exercice sans préparation.

1. a. Écrire une fonction renvoyant le nombre d'occurrences d'un flottant `x` d'une liste `L`.

   b. Écrire une fonction prenant en argument une liste `obs` et la liste `support` des valeurs de `obs` sans répétition et qui renvoie le nombre d'occurrences de chacune des valeurs de `support` dans `obs` au même indice.

   Par exemple, si `obs = [1,4,3,8,4,1,2,2,8,8]` *et* `support = [1,4,3,8,2]`*, alors la fonction devra renvoyer la liste* `[2,2,1,3,2]`.

2. Un étudiant propose une fonction `simule_X()` qui simule la réalisation d'une variable aléatoire X. Les valeurs prises par X sont stockées dans une liste `support`.

   Écrire une fonction qui simule un grand nombre de fois la variable aléatoire X et qui renvoie la liste des fréquences d'apparitions des valeurs de la liste `support`.

Corrigé

Planche 22
Agro-Véto 2023

Exercice sans préparation.

1. Écrire une fonction qui détermine le maximum d'une liste de nombres.
2. Écrire une fonction `histo` qui prend en argument une liste L de chiffres (entre 0 et 9) et qui renvoie une liste nb telle que pour tout $k \in [\![0, 9]\!]$, `nb[k]` soit le nombre de fois où le chiffre k apparait dans `L`.

Corrigé

Planche 23
Agro-Véto 2023

Exercice sans préparation.

1. Écrire une fonction `gene` qui prend en entrée un entier naturel non nul `n` et renvoie une chaîne de `n` caractères formée aléatoirement, de façon équiprobable, des caractères `"A"`, `"T"`, `"C"` et `"G"`.

2. Écrire une fonction `nbAC` qui prend en argument une chaîne de caractères formées des caractères `"A"`, `"T"`, `"C"` et `"G"` et qui renvoie le nombre de fois où la séquence `"AC"` est présente.

   Par exemple, `nbAC("GAGCACCCTACTTGGCGCGA")` *renverra 2.*

Corrigé

Planche 24
Agro-Véto 2023

Exercice sans préparation.
On s'intéresse dans cet exercice à des listes d'entiers. L'utilisation des fonctions **`max`** et `count` est interdite.

1. Écrire une fonction prenant en argument une liste d'entiers `L` et un entier naturel k. Cette fonction renvoie `True` si tous les entiers de cette liste sont compris entre 0 et k et `False` sinon. Par exemple, pour `L = [0, 2,0]`, la fonction renvoie `True` si $k = 2$ ou $k = 3$ et `False` si $k = 1$.

2. Écrire une fonction de mêmes arguments, où la liste est supposée ne contenir que des entiers compris entre 0 et k. Cette devra fonction renvoie l'élément le plus fréquent (le plus petit d'entre eux s'il y en a plusieurs). Par exemple, pour `L = [0, 4, 0, 1, 4]` et $k = 5$, la fonction renverra 0.

Corrigé

Planche 25
Agro-Véto 2023

Exercice sans préparation.

1. Écrire une fonction `somme(f,a,b,N)` qui renvoie $\displaystyle\sum_{k=0}^{N-1} f\left(a+k\frac{b-a}{N}\right)$ où f est une fonction, a et b deux réels ($a<b$) et N un entier supérieur à 1.

2. Écrire une fonction prenant en argument une fonction f, deux réels a et b, qui renvoie une valeur approchée de l'intégrale $\displaystyle\int_a^b f(t)\,\mathrm{d}t$.

   Utiliser cette fonction pour donner une valeur approchée de $\displaystyle\int_0^{+\infty} \exp\left(-t^2\right)\mathrm{d}t$ en se servant de l'égalité :

$$\int_0^{+\infty} f(t)\,\mathrm{d}t = \int_0^1 \frac{1}{(1-t)^2} f\left(\frac{t}{1-t}\right)\mathrm{d}t.$$

Corrigé

Planche 26
Agro-Véto 2023

Exercice sans préparation.
Soient $n \in \mathbb{N}$ et $(a_0, \ldots, a_n) \in \mathbb{R}^{n+1}$. On considère le polynôme $P(X) = a_0 + a_1 X + \cdots + a_n X^n$. On représente ce polynôme P en Python par la liste de ses coefficients $[a_0, \ldots, a_n]$.

1. Écrire une fonction `evalue` qui prend en argument un polynôme P et un réel (flottant) a et qui renvoie $P(a)$.
2. Écrire une fonction `deriv` qui prend en argument un polynôme P et qui renvoie P'.
3. On considère l'application linéaire f définie sur $\mathbb{R}[X]$ par $f(P)(X) = 2XP(X) - P'(X)$. Écrire une fonction f qui prend en argument un polynôme P et qui renvoie $f(P)$.

Corrigé

Planche 27
Agro-Véto 2023

Exercice sans préparation.

1. Écrire une fonction `somme_cumul` qui prend en entrée une liste `L` d'entiers et qui renvoie une liste `M` des sommes cumulées, c'est-à-dire une liste `M`, de même longueur que `L`, telle que `M[i]` $= \sum_{k=0}^{i}$ `L[k]` pour tout indice i de `L`.

2. On propose à deux joueurs A et F la résolution d'une suite infinie de problèmes numérotés $0, 1, \ldots, n, \ldots$ Ils débutent la résolution à l'instant 0 du problème numéro 0. Dès que l'un des joueurs termine la résolution du problème k, il passe au problème k + 1. Le numéro d'un problème est attribué au joueur qui le résout en premier. En cas de simultanéité de la résolution d'un problème par les deux joueurs, celui-ci n'est pas attribué.

   On souhaite écrire en Python une fonction `repartition(dureesA, dureesF)` qui, étant données les deux listes des durées de résolution des n premiers problèmes par les deux joueurs, renvoie les listes des numéros des problèmes attribués à A et F pour les n premiers problèmes.

   a. L'instruction `repartition([7,1,2,1,2],[4,2,4,3,1])` renvoie le couple `([3, 4], [0, 1])`. Quel couple renvoie l'exécution de `repartition( [3,2,5,1,1,1],[4,2,2,4,4,2])` ?

   b. Écrire la fonction recherchée.

Corrigé

Planche 28
Agro-Véto 2023
(sujet 0)

Exercice sans préparation.

1. Rappelez la définition d'une loi de Poisson.

   On note $F(\lambda, k)$ la fonction de répartition d'une loi de Poisson de paramètre λ évaluée en $k \in \mathbb{N}$.

   Programmer cette fonction en Python.

2. On définit la fonction $G : \mathbb{R}_+^* \times [0, 1[\mapsto \mathbb{R}$ comme suit :

$$G(\lambda, x) = \min\{k \mid F(\lambda, k) > x\}.$$

   On admet que G est bien définie. Programmer G en Python.

Corrigé

Planche 29
Agro-Véto 2023
(sujet 0)

Exercice sans préparation.

Dans cet exercice, on considère des listes de booléens, c'est-à-dire des listes ayant pour éléments uniquement des valeurs `True` et `False`. Par exemple les listes `[True, False, True]` et `[False, False]` sont des listes de booléens.

1. Écrire une fonction `nb_vrai` prenant en entrée une liste de booléens et renvoyant le nombre d'occurrences de la valeur `True`. Par exemple `nb_vrai([False, True, True])` doit renvoyer 2, car la valeur `True` apparait deux fois dans la liste `[False, True, True]`.

2. Écrire une fonction `bloc_vrai` prenant en entrée une liste de booléens, et renvoyant :

   - `True` si la valeur `True` est présente dans la liste et que tous les `True` sont côte-à-côte, dans le même bloc,
   - `False` sinon.

   Par exemple, `bloc_vrai([False, True, True, True, False, False])` doit renvoyer `True` car il y a un unique bloc de 3 `True`. Autre exemple, `bloc_vrai([False, True, True, True, False, False, True, True, False])` doit renvoyer `False` car il y a deux blocs de `True` : un premier de longueur 3, un second de longueur 2.

Corrigé

Planche 30
Agro-Véto 2023
(sujet 0)

Exercice sans préparation.

On considère un plateau de n cases, numérotées de 0 à $n-1$. Sur chaque case, se trouve un numéro de case (le même numéro peut être inscrit sur plusieurs cases différentes).

Par exemple, sur le plateau ci-dessous, on a écrit 2 sur la case d'indice 0, et on a écrit 4 sur la case d'indice 1 et on a écrit 3 sur la case d'indice 3.

2	4	1	3	0

On représente un plateau de taille n en machine par une liste de taille n. La k-ème élément de la liste contient le numéro inscrit sur la k-ème case. Ainsi, le plateau de l'exemple est représenté par la liste `[2,4,1,3,0]`.

1. On appelle *point fixe* une case sur laquelle est inscrit son propre numéro. Ainsi, sur l'exemple il y a un unique point fixe, la case 3 (car il est écrit 3 sur la case d'indice 3). Écrire une fonction `compter` qui prend en entrée une liste représentant un plateau et qui renvoie le nombre de points fixes (la fonction renvoie 0 s'il n'y a aucun point fixe).

2. On place un pion sur la case zéro. Ensuite, on fait avancer le pion comme suit : on regarde le numéro inscrit sur la case sur laquelle est le pion, puis on déplace le pion sur la case portant ce numéro.

   Sur l'exemple, on va déplacer le pion sur la case d'indice 2, puis la case d'indice 1, puis la case d'indice 4, puis la case d'indice 0, puis la case d'indice 1, etc.

   Écrire une fonction `avancer` qui prend en entrée la liste représentant le plateau de jeu et un entier naturel p et qui donne le numéro de la case sur laquelle arrive le pion au bout de p étapes.

   Par exemple, `avancer([2,4,1,3,0], 1)` *renvoie 2 et* `avancer([2,4,1,3,0], 3)` *renvoie 4.*

Corrigé

Planche 31
Agro-Véto 2023
(sujet 0)

Exercice sans préparation.

Une grille de Takuzu, ou Binero, est une grille carrée à n lignes et autant de colonnes avec n pair. Au départ, chaque case peut contenir 0, 1 ou être vide. En Python, une grille est codée par une liste de listes, les cases vides étant représentées par `None`. Par exemple, la grille :

0	1		0
1		0	1
0	0	1	1
1	1	0	

est codée par la liste :

```
[[0,1, None, 0],[1, None, 0, 1], [0, 0, 1, 1], [1, 1, 0, None]]
```

Dans la suite, on identifie la grille et la liste qui la représente.

1. Écrire une fonction `nb_vides` prenant en argument une grille de taille non précisée et renvoyant le nombre de cases non remplies. Par exemple, pour la grille précédente, la fonction renverra 3.

2. Écrire une fonction `test_lignes_identiques` prenant en argument une grille de taille non précisée et renvoyant un booléen testant s'il existe deux lignes identiques. Par exemple, pour la grille précédente, la fonction renverra `False`.

   On pourra se servir du test d'égalité de deux listes.

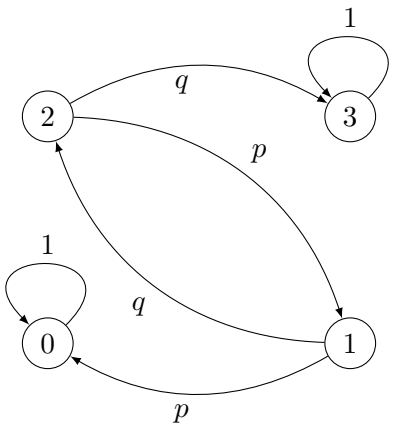
Corrigé

Planche 32
Agro-Véto 2023
(sujet 0)

Exercice sans préparation.
Soit $p \in]0,1[$. Une puce peut occuper quatre positions numérotées de 0 à 3. À chaque instant $t = n$, où $n \in \mathbb{N}$, elle se déplace (ou pas). On suppose que :

- 0 et 3 sont des puits: si la puce s'y trouve, elle y reste définitivement ;
- si la puce est en 1, alors elle saute à la position 0 à l'instant suivant avec la probabilité p ou à la position 2 avec la probabilité $q = 1 - p$;
- si la puce est en 2, alors elle saute en 1 avec la probabilité p ou en 3 avec la probabilité q.

À $t = 0$, la puce se trouve en 1. Les conditions de déplacement peuvent être schématisées comme suit :



On suppose que la bibliothèque `random` a été importée avec l'instruction `import random`. On rappelle que pour un nombre `a` entre 0 et 1, l'expression `random.random()<a` vaut `True` avec la probabilité `a`.

1. Écrire un programme saut prenant en argument le paramètre p et la position de la puce à l'instant n et renvoyant la position de la puce à l'instant suivant $n + 1$.
2. Écrire un programme position prenant en argument le paramètre p et un entier naturel n non nul et renvoyant la liste des n premières positions de la puce.

Corrigé

Planche 33
Agro-Véto 2023
(sujet 0)

Exercice sans préparation.

1. Écrire une fonction `strict_croissante` prenant une liste d'entiers en paramètre et renvoyant `True` si elle est strictement croissante et `False` sinon.

   Par exemple, l'instruction `strict_croissante([1, 7, 99])` *renvoie* `True`*, mais* `strict_croissante([1, 7, 5])` *et* `strict_croissante([1, 7, 7])` *renvoient* `False`*.*

2. Écrire une fonction `strict_monotone` prenant une liste d'entiers en paramètre et renvoyant `True` si elle est strictement monotone et `False` sinon.

   Par exemple, `strict_monotone([5, 2, 1])` *renvoie* `True`*, mais* `strict_monotone([1, 2, 1])` *renvoie* `False`*.*

Corrigé

Planche 34
Agro-Véto 2022

Exercice sans préparation.
Soit N un entier naturel non nul. On cherche à trier une liste `L` d'entiers naturels strictement inférieurs à N.

1. Écrire une fonction `comptage`, d'arguments `L` et N, renvoyant une liste `P` dont le k-ème élément désigne le nombre d'occurences de l'entier k dans la liste `L`.

2. Utiliser la liste `P` pour en déduire une fonction `tri`, d'arguments `L` et N, renvoyant la liste `L` triée dans l'ordre croissant.

Corrigé

Planche 35
Agro-Véto 2022

Exercice sans préparation.

1. Écrire une fonction récursive `factoriel_rec` prenant en argument un entier naturel n et renvoyant $n!$.
2. Écrire une fonction itérative `factoriel_iter` prenant en argument un entier naturel n et renvoyant $n!$.

Corrigé

Planche 36
Agro-Véto 2022

Exercice sans préparation.

On modélise une permutation des entiers de 1 à n par une liste de n nombres distincts, compris entre 1 et n.

Compléter le code ci-dessous afin d'écrire une fonction qui prend en argument un entier naturel non nul n et qui renvoie une liste de listes représentant la liste des permutations des entiers de 1 à n.

Par exemple, `permutations(3)` devra renvoyer `[[3,2,1],[2,3,1],[2,1,3],[3,1,2],[1,3,2],[1,2,3]]`.

```
def permutations(n):
    if n == 1:
        return ...
    else:
        L = permutations(n-1)
        L2 = ...
        for p in L:
            for k in range(...):
                p2 = p[...] + ... + p[...]
                L2.append(...)
    return L2
```

Corrigé

Planche 37
Agro-Véto 2022

Exercice sans préparation.

Le but de cette partie est de trouver le nombre maximal de 0 contigus (c'est-à-dire figurant dans des cases consécutives) dans un tableau unidimensionnel.

On utilisera le tableau suivant comme exemple :

`i`	0	1	2	3	4	5	6	7	8	9	10	11	12
`tab[i]`	0	0	1	1	0	0	0	1	0	0	0	1	1

1. Écrire une fonction `nombreZeros(t,i)`, prenant en paramètres une liste `t` (de longueur n), et un indice i compris entre 0 et $n-1$, et renvoyant le nombre de zéros consécutifs à partir de l'élément d'indice i.

2. Écrire une fonction `nombreZerosMax` qui prend en argument une liste `t` et qui renvoie le nombre maximal de zéros consécutifs dans `t`. *On prendra soin d'écrire l'algorithme le plus efficace possible.*

Corrigé

Planche 38
Agro-Véto 2022

Exercice sans préparation.
On considère le code ci-dessous

```
def d(n):
    L=[1]
    for nombre in range(2,n+1):
        if n%nombre==0:
            L.append(nombre)
    return L
```

1. Quel est le résultat de `d(4)` ? de `d(10)` ? Que fait la fonction `d` ?
2. On dit qu'un entier k est un diviseur non trivial de n si k divise n et si k diffère de 1 et n. Écrire une fonction `somme` qui renvoie la somme des carrés des diviseurs non triviaux de l'entier passé en argument.
3. Écrire la suite des instructions permettant d'afficher tous les nombres entiers inférieurs à 1000 et égaux à la somme des carrés de leurs diviseurs non-triviaux.

Corrigé

Planche 39
Agro-Véto 2022

Exercice sans préparation.

On suppose qu'on dispose d'une fonction `alphabet` qui renvoie une chaîne de 26 caractères composées des 26 lettres (minuscules) de l'alphabet français, dans l'ordre alphabétique.

On souhaite coder une chaîne de caractères par décalage de n lettres. Par exemple, pour $n = 3$, `"a"` devient `"d"`, `"b"` devient `"e"`, ... `"w"` devient `"z"`, `"x"` devient `"a"`, `"y"` devient `"b"` et `"z"` devient `"c"`.

1. Écrire une fonction `decalage` qui prend en argument un entier `n` et qui renvoie une chaîne de caractères contenant les 26 lettres de l'alphabet (en minuscule) décalés de n.

2. En déduire une fonction `codage` qui prend en argument une chaîne de caractères et un entier naturel n qui code par décalage la chaîne de caractères en décalant chaque caractères de n.

Corrigé

Planche 40
Agro-Véto 2022

Exercice sans préparation.

1. On considère l'entier $n = 1234$. Quel est le quotient, noté q, dans la division euclidienne de n par 10 ? Quel est le reste ? Que se passe-t-il si on recommence la division par 10 à partir de q?

2. Écrire une fonction `sommecube`, d'argument n, renvoyant la somme des cubes des chiffres de l'écriture décimale du nombre n.

3. Écrire un script listant tous les nombres entiers inférieurs à 1000 et égaux à la somme des cubes de leurs chiffres.

Corrigé

Planche 41
Agro-Véto 2022

Exercice sans préparation.

1. Écrire une fonction qui renvoie la liste des indices du maximum d'une liste d'entiers.
2. Écrire une fonction qui renvoie la valeur de la deuxième plus grande valeur d'une liste d'entiers deux-à-deux distincts (on suppose que la liste est de longueur au plus 2).

Corrigé

Planche 42
Agro-Véto 2022

Exercice sans préparation.

1. Écrire une fonction booléenne qui vérifie si une matrice (représentée par un array) passée en argument est diagonale.
2. Écrire une fonction booléenne qui vérifie si une matrice (représentée par un array) passée en argument est carrée et si elle contient tous les entiers de 1 à n^2, où n est son nombre de lignes et de colonnes.

Corrigé

Planche 43
Agro-Véto 2022

Exercice sans préparation.

Othello est un jeu de société qui se joue à deux joueurs sur un plateau de 8×8 cases. L'objectif du jeu est de capturer le plus grand nombre possible de pions de l'adversaire en les retournant.

Voici les règles de retournement des pions dans le jeu Othello :

- À son tour, chaque joueur place un pion de sa couleur sur une case vide du plateau. Le pion doit être placé de manière à encadrer un ou plusieurs pions adverses entre le pion nouvellement placé et un autre pion de sa couleur déjà présent sur le plateau.
- Les pions adverses encadrés entre les deux pions de la même couleur sont retournés et deviennent des pions de la couleur du joueur actif.
- Les pions peuvent être capturés horizontalement, verticalement ou en diagonale.
- Il est obligatoire de jouer un coup qui permet de retourner au moins un pion adverse. Si un joueur ne peut pas jouer un tel coup, il doit passer son tour.

On modélise un plateau par un array de 8 lignes par 8 colonnes dont les coefficients - `0`, `1` et -1 - représentent respectivement une case vide, un pion blanc ou un pion noir.

Compléter le code de la fonction booléenne `coup_valide` qui prend en argument un array modélisant un plateau de jeu à un tour donné, un numéro `couleur` de joueur (-1 ou 1) et deux entiers i et j. La fonction devra renvoyer `True` si, et seulement si, le joueur `couleur` peut placer un pion à la ligne `i`, colonne `j` du plateau.

```
def coup_valide(plateau, couleur, i, j):
    if plateau[i,j] != ... :
        return ...
    compteur = ...
    for (di,dj) in [(-1,-1),(-1,0),(-1,1),(0,-1),(0,1),(1,-1),(1,0),(1,1)]:
        x,y = i+di, j+dj
        while 0<= ... < 8 and 0 <= ... < 8 and plateau[x,y] == ... :
            x, y = ... , ...
            compteur = ...
        if ... and ... and plateau[x,y] == ... and compteur ... :
            return True
    return ...
```

Corrigé

Planche 44
Agro-Véto 2022

Exercice sans préparation.

On considère des graphes non orientés à n sommets dont les sommets sont étiquetés par les entiers de 0 à $n-1$. On modélise ces graphes par leurs listes d'adjacence.

Par exemple, dans le graphe représenté par `[[1,2,4],[0],[0,4],[4],[0,2,3]]`*, les voisins du sommet 2 sont 0 et 4.*

1. Pour un graphe non orienté G, on note S la liste de ses sommets. Pour un sous-ensemble non vide X de sommets tel que $X \neq S$, on appelle **coupe** de G par X l'ensemble des arêtes ayant exactement une extrémité dans X et une dans $S \setminus X$. Une arête sera représentée par la liste de ses deux extrémités (dans un ordre quelconque), et une coupe sera représentée par la liste des arêtes qui la composent.

   Par exemple, dans le graphe G représenté par `[[1,2,4],[0],[0,4],[4],[0,2,3]]`, la coupe définie par $X = \{0, 2\}$ est représentée par `[[0, 1], [0, 4], [2, 4]]`. Écrire une fonction coupe prenant en argument un graphe G et un ensemble de sommets X (représenté par une liste), et qui renvoie la liste des arêtes de la coupe de G par X.

2. Écrire une fonction `voisins_de_voisins` prenant en argument un graphe non orienté représenté par sa liste d'adjacence et renvoyant une liste (sans répétition) de tous les couples de points situés exactement à deux arêtes l'un de l'autre.

Corrigé

Corrigé de l'exercice de la planche 1

1.
```
def occ(x,L):
    nb_x = 0
    for y in L:
        if x==y:
            nb_x += 1
    return nb_x
```

2. a.
```
import random as rd

def simulebinom(n,p):
    s = 0
    for _ in range(n):
        if rd.random() < p:
            s += 1
    return s


def seriebinom(m,n,p):
    L = []
    for _ in range(m):
        L.append(simulebinom(n,p))
    return L
```

b. La méthode la plus esthétique (et efficace) est d'utiliser un dictionnaire :

```
import numpy as np

def occ_serie(L):
    dico = {}
    for x in L:
        if x in dico:
            dico[x] += 1
        else:
            dico[x] = 1
    tab = [[],[]]
    for x,nb_x in dico.items():
        tab[0].append(x)
        tab[1].append(nb_x)
    return np.array(tab)
```

```
def occ_serie(L):
    dico = {}
    for x in L:
        if x not in dico:
            dico[x]=1
        else:
            dico[x] += 1
    tab = [list(dico.keys()),list(dico.values())]
    return np.array(tab)
```

Retour à la planche 1

Corrigé de l'exercice de la planche 2

1.
```python
import random as rd

def desVR():
    return (rd.randin(1,6),rd.randin(1,6))
```

2.
```python
def manche(nbjetons):
    v,r = desVR()
    if nbjetons[v-1] == 0:
        return nbjetons
    elif v+r <= 6:
        nbjetons[v+r-1] += 1
    nbjetons[v-1] -= 1
    return nbjetons
```

Retour à la planche 2

Corrigé de l'exercice de la planche 3

1.
```python
import random as rd

def chaine_tirages(n):
    urne = "A"*3 + "T"*3 + "C"*2 + "G"*2
    ch = ""
    for _ in range(n):
        ch += rd.choice(urne)
    return ch
```

2. On commence par écrire une fonction qui cherche la seconde chaine à un indice i fixé dans la première, en comptant le nombre d'erreurs de caractères. On peut alors écrire la deuxième fonction parcourant tous les indices possibles où peut commencer la deuxième chaine dans la première.

```python
def contient_avec_erreur_indice_i(i,ch1,ch2):
    nb_erreurs = 0
    for j in range(len(ch2)):
        if i+j < len(ch1) and ch1[i+j] != ch2[j]:
            nb_erreurs += 1
        if nb_erreurs == 2:
            return False
    return True

def contient_avec_erreur(ch1,ch2):
    for i in range(len(ch1)-len(ch2)+1):
        if contient_avec_erreur_indice_i(i,ch1,ch2):
            return True
    return False
```

Retour à la planche 3

Corrigé de l'exercice de la planche 4

1.
```
def somme(L,i,j):
    s = 0
    for k in range(i,j):
        s += L[k]
    return s
```

2.
```
def sous_liste_long(L,k):
    s_max = somme(L,0,k)
    for i in range(1,len(L)-k):
        s = somme(L,i,i+k)
        if s > s_max:
            s_max = s
    return s_max
```

Retour à la planche 4

Corrigé de l'exercice de la planche 5

1.
```
def minmax(L):
    i_min, i_max = 0, 0
    for i in range(1,len(L)):
        if L[i] < L[i_min]:
            i_min = i
        elif L[i] > L[i_max]:
            i_max = i
     return L[i_max], i_max, L[i_min], i_min
```

2.
```
def minmax(L):
    i_max, i_2max = 0, 1
    if L[0] < L[1]:
        i_max, i_2max = 1, 0
    for i in range(2,len(L)):
        if L[i] > L[i_max]:
            i_max, i_2max = i, i_max
        elif L[i] > L[i_2max]:
            i_2max = i
    return L[i_max], i_max, L[i_2max], i_2max
```

Retour à la planche 5

Corrigé de l'exercice de la planche 6

1.
```
def extract(L,n,p):
    return [(i,j) for (i,j) in L if (0<=i<n and 0 <= j <p)]
```

2.
```
def voisins(M,i,j):
    L = []
    n,p = np.shape(M)
    for u in range(max(0,i-1),1+min(n-1,i+1)):
        for v in range(max(0,j-1),1+min(n-1,j+1)):
            if abs(u-i) + abs(v-j) > 0:
                L.append(M[u,v])
    return L
```

Retour à la planche 6

Corrigé de l'exercice de la planche 7

1.
```
def d(A,B):
    return ((A[0]-B[0])**2 + (A[1]-B[1])**2)**0.5
```

2.
```
def point_moyen(nuage):
    s_x, s_y, n = 0, 0, len(nuage)
    for k in range(n):
        x,y = nuage[k]
        s_x += x
        s_y += y
    return (s_x/n,s_y/n)
```

3.
```
def i_plus_proche_pm(nuage):
    pm = point_moyen(nuage)
    i_min = 1
    for i in range(2,len(nuage)):
        if d(nuage[i],pm) < d(nuage[i_min],pm):
            i_min = i
    return i_min
```

Retour à la planche 7

Corrigé de l'exercice de la planche 8

1.
```
def testeH(L):
    n = len(L)
    for x in L:
        if x != int(x) or not(0 <= x < n): # ou type(x) != int
            return False
    return True
```

2.
```
def testeHprime(L):
    n = len(L)
    nb_occ = [0]*n
    for x in L:
        if int(x) == x and 0 <= x < n and nb_occ[x] == 0: # ou type(x) = int
            nb_occ[x] = 1
        else:
            return False
    return True
```

3. Il suffit de considérer tous les permutations de la sous-liste $[\![1, n-1]\!]$: il y en a $(n-1)!$.

Retour à la planche 8

Corrigé de l'exercice de la planche 9

1.
```
def verif(L,a,b):
    for x in L:
        if x != int(x) or not(a <= x <= b):
            return False
    return True
```

2.
```
def denombre(L,a,b):
    n = len(L)
    occ = [0]*n
    for x in L:
        occ[x-a] += 1
    return occ
```

Retour à la planche 9

Corrigé de l'exercice de la planche 10

1.
```python
import random as rd

def marche_aleatoire(i,N):
    x = i
    for _ in range(N):
        x += rd.choice([-1,1]) # ou x += 2*rd.randint(0,1) - 1
    return x
```

2.
```python
def gain(n,a,b,i):
    x = i
    while 0 < x < n-1:
        x += rd.choice([-1,1])
    if x == 0:
        return a
    else:
        return b
```

Retour à la planche 10

Corrigé de l'exercice de la planche 11

1.
```
def temps(coords):
    return [coords[k][3] for k in len(coords)]
```

2.
```
def plus_haut(coords):
    k_max = 0
    for k in range(1,len(coords)):
        if coords[k][2] > coords[k_max][2]:
            k_max = k
    return coords[k_max][:2]
```

Retour à la planche 11

Corrigé de l'exercice de la planche 12

1.
```python
def simule_lancers(N):
    return [rd.randint(0,1) for k in range(N)]
```

2.
```python
def simule_config():
    L = [rd.randint(0,1) for k in range(3)]
    while L[-3:] != [1,1,0]:
        L.append(rd.randint(0,1))
    return L
```

On utilise ensuite la loi faible des grands nombres pour estimer le temps d'attente de cette configuration :

```python
N = 10000
s = 0
for _ in range(N):
    s += len(simule_config())
print(s/N)
```

Retour à la planche 12

Corrigé de l'exercice de la planche 13

1.
```
def intervalle(L,a,b):
    for x in L:
        if x < a or x > b:
            return False
    return True
```

2. On construit une liste de $n+1$ valeurs, initialement des zéros. À chaque fois, qu'on rencontre une valeur x dans L, on ajoute 1 à son nombre d'apparitions (occurrences) dans la liste `occ`.

```
def nb_occurrences(L,n):
    occ = [0] * (n+1)
    for x in L:
        occ[x] += 1
    return occ
```

Retour à la planche 13

Corrigé de l'exercice de la planche 14

1. On remarque qu'il y a changement de monotonie à l'indice k si, et seulement si, `L[k]-L[k-1]` et `L[k+1]-L[k]` sont de signes opposés.

```
def indice_chgt(L,k):
    return (L[k]-L[k-1])*(L[k+1]-L[k]) < 0
```

2. On propose une première version (non optimisée)

```
def monotonie_chgt(L):
    indices = []
    for k in range(1,len(L)-1):
        if indice_chgt(L,k):
            indices.append(k)
    return indices
```

puis une seconde plus compacte

```
def monotonie_chgt(L):
    return [k for k in range(1,len(L)-1) if indice_chgt(L,k)]
```

et enfin une version un peu plus optimisée (en évitant de calculer plusieurs fois `L[k+1]-L[k]`.

```
def monotonie_chgt(L):
    ancienne_diff = L[1]-L[0]
    indices = []
    for k in range(1,len(L)-1):
        nouvelle_diff = L[k+1]-L[k]
        if nouvelle_diff * ancienne_diff < 0:
            indices.append(k)
        ancienne_diff = nouvelle_diff
    return indices
```

Retour à la planche 14

Corrigé de l'exercice de la planche 15

1.
```
def second_max(L):
    if L[0] > L[1]:
        max1,max2 = L[0], L[1]
    else:
            max1,max2 = L[1], L[0]
    for k in range(2,len(L)):
        if L[k] > max1:
            max1, max2 = L[k], max1
        elif L[k] > max2:
            max2 = L[k]
    return max2
```

2. Il suffit de compter combien d'éléments de la liste sont strictement inférieurs à x.

```
def indice(L,x):
    nb_inf = 0
    for y in L:
        if y < x:
            nb_inf += 1
    return nb_inf
```

Retour à la planche 15

Corrigé de l'exercice de la planche 16

1.
```python
import random as rd

def liste(n,p):
    L = []
    for k in range(n):
        L.append(rd.randint(0,p-1))
    return L
```

2.
```python
def nb_boules(n,p):
    L = []
    for k in range(n):
        num = rd.randint(0,p-1)
        if num not in L:
            L.append(num)
    return len(L)
```

3. On utilise la loi des grands nombres pour approcher $\mathbb{E}(S)$:

```python
def esperance_S(n,p):
    N = 1000
    s = 0
    for i in range(N)
        s += nb_boules(n,p)
    return s/N
```

Retour à la planche 16

Corrigé de l'exercice de la planche 17

1.
```
def somme(n):
    s = 0
    for k in range(1,n+1):
        s += 1/(k**2)
    return s
```

2. On utilise la méthode d'inversion : si U est une variable aléatoire suivant la loi uniforme sur $[0,1[$ et si F désigne la fonction de répartition de X, alors :

$$\forall k \in \mathbb{N}^*,\ \mathbb{P}(X=k) = \mathbb{P}(F(k-1) < U \leqslant F(k)).$$

```
import numpy as np
import random as rd

def simule_X():
    x = rd.random()
    s = 0
    k = 1
    while s < x:
        s += 6/((np.pi*k)**2)
        k += 1
    return k-1
```

Retour à la planche 17

Corrigé de l'exercice de la planche 18

1.
```python
import random as rd

def simule_lancers(n):
    p = 0
    for k in range(n):
        if rd.random() < 1/2:
            p += 1
     return (p,n-p)
```

2. L'idée est de stoker la liste des résultats des lancers (le résultat des deux derniers résultats suffirait). On modélise par 0 l'obtention de "pile" et 1 celle de "face".

```python
def nb_lancers():
    lancers = [rd.randint(0,1) for k in range(3)] # 0 pour pile, 1 pour face
    while lancers[-3:] != [0,0,1]:
        lancers.append(rd.randint(0,1))
    return len(lancers)
```

Retour à la planche 18

Corrigé de l'exercice de la planche 19

1. On calcule la différence entre les deux premiers termes de la liste, correspondant à la raison de la suite si elle était arithmétique.

```
def arithmetique(u):
    r = u[1] - u[0]
    for k in range(1,len(u)-1):
        if u[k+1] - u[k] != r:
            return False
    return True
```

2. On adapte la même idée qu'à la question précédente, en distinguant le cas où le premier terme est nul.

```
def geometrique(v):
    if v[0] == 0:
        q = 0
    else:
        q = v[1]/v[0]
    for k in range(0,len(v)-1):
        if v[k+1] != q*v[k]:
            return False
    return True
```

Retour à la planche 19

Corrigé de l'exercice de la planche 20

1.
```python
import random as rd

def sauts(i,n):
    L = [-1,1]
    for k in range(n):
        i += rd.choice(L)
    return i
```

2.
```python
def bord(i):
    L = [-1,1]
    nb_sauts = 0
    while 0 < i < n:
        i += rd.choice(L)
        nb_sauts += 1
    return (nb_sauts,i)
```

Retour à la planche 20

Corrigé de l'exercice de la planche 21

1. a.
```
def occ(x,L):
    s = 0
    for y in L:
        if x == y:
            s += 1
    return s
```

   b.
```
def list_occ(obs,support):
    return [occ(x,obs) for x in support]
```

2.
```
def liste_freq(support)
    N = 1000
    freq = [0]*len(support)
    for k in range(N):
        x = simuleX()
        for i in range(len(support)):
            if x == support[i]:
                freq[i] += 1/N
     return freq
```

Retour à la planche 21

Corrigé de l'exercice de la planche 22

1.
```
def maxi(L):
    m = L[0]
    for x in L[1:]:
        if x > m:
            m = x
    return m
```

2.
```
def histo(L):
    nb = [0] * 10
    for x in L:
        nb[x] += 1
    return nb
```

Retour à la planche 22

Corrigé de l'exercice de la planche 23

1. Un classique :

```
def gene(n):
    L = "ATCG" # ou L = ["A","T","C","G"]
    ch = ""
    for k in range(n):
        ch += rd.choice(L)
    return ch
```

2. On propose un code (non optimisé) qui utilise le test d'égalité de chaînes de caractères.

```
def nbAC(chaine):
    s = 0
    n = len(chaine)
    for k in range(n-1):
        if chaine[k:k+2] == "AC":
            s += 1
    return s
```

Retour à la planche 23

Corrigé de l'exercice de la planche 24

1.
```python
def verif(L,k):
    for x in L:
        if x < 0 or x > k:
            return False
    return True
```

```python
def majoritaire(L,k):
    occurrence = [0 for i in range(k+1)]
    i_max = 0
    nb_max = 0
    for x in L:
        occurrence[x] += 1
        if occurrence[x] > nb_max:
            i_max = x
            nb_max = occurrence[x]
        elif occurrence[x] == nb_max and x < i_max:
            i_max = x
    return i_max
```

Retour à la planche 24

Corrigé de l'exercice de la planche 25

2.
```python
def somme(f,a,b,N):
    s = 0
    h = (b-a)/N
    for k in range(N):
        s += f(a+k*h)
    return s
```

2. On applique la méthode des rectangles qui utilise la convergence des sommes de Riemann.

```python
def approx_integrale(f,a,b):
    N = 1000
    return (b-a)/N * somme(f,a,b,N)
```

L'égalité de l'énoncé s'écrit :

$$\int_0^{+\infty} \exp\left(-t^2\right) \mathrm{d}t = \int_0^1 \frac{1}{(1-t)^2} \exp\left(-\frac{t^2}{(1-t)^2}\right) \mathrm{d}t.$$

Le code ci-dessous fournit donc une approximation `x` de l'intégrale $\displaystyle\int_0^{+\infty} \exp\left(-t^2\right) \mathrm{d}t = \frac{\sqrt{\pi}}{2}$ (en utilisant une densité de la loi normale $\mathcal{N}\left(0, \frac{1}{\sqrt{2}}\right)$:

```python
import numpy as np

def f(t):
    return np.exp(-(t/(1-t))**2)/(1-t)**2

x = approx_integrale(f,0,1)
```

Retour à la planche 25

Corrigé de l'exercice de la planche 26

1. On propose une version itérative et une version récursive :

```
def evalue(P,a):
    s = 0
    for k in range(len(P)):
        s += P[k]*a**k
    return s

def evalue2(P,a):
    if len(P)==1:
        return P[0]
    else:
        return P[0] + a * eval2(P[1:],a)
```

2. On distingue le cas où P est nul. Dans le cas contraire, si $P(X) = \sum_{i=0}^{n} a_i X^i$, alors :

$$P'(X) = \sum_{i=1}^{n} i a_i X^{i-1} = \sum_{k=0}^{n-1} (k+1) a_{k+1} X^k.$$

On en déduit le code ci-dessous :

```
def deriv(P):
    if len(P)==1:
        return [0]
    return [(k+1)*P[k+1] for k in range(len(P)-1)]
```

3.

```
def f(P):
    if P == [0]:
        return P
    else:
        n = len(P)
        res = [0]*(n+1)
        XP = [0]+P
        P_prime = deriv(P)
        for k in range(n-1):
            res[k] = 2*XP[k] - P_prime[k]
        res[n-1] = 2*XP[n-1]
        res[n] = 2*XP[n]
        return res
```

Retour à la planche 26

Corrigé de l'exercice de la planche 27

1.
```python
def somme_cumul(L):
    M = [L[0]]
    for k in range(1,len(L)):
        M.append(M[k-1] + L[k])
    return M
```

2. a. L'exécution de `repartition( [3,2,5,1,1,1],[4,2,2,4,4,2])` renvoie le couple : `([0, 1, 3, 4, 5], [2])`.

   b.
```python
def repartition(dureesA,dureesF):
    n = len(dureesA)
    cumulA,cumulF = somme_cumul(dureesA),somme_cumul(dureesF)
    A,F = [],[]
    for k in range(n):
        if cumulA[k] < cumulF[k]:
            A.append(k)
        elif cumulF[k] < cumulA[k]:
            F.append(k)
    return A, F
```

Retour à la planche 27

Corrigé de l'exercice de la planche 28

1. On dit qu'une variable aléatoire réelle X suit la loi de Poisson de paramètre $\lambda > 0$ si :

$$\forall k \in \mathbb{N},\ \mathbb{P}(X = k) = \frac{\lambda^k}{k!} e^{-\lambda}.$$

On implémente la fonction `F` en remarquant que :

$$\forall n \in \mathbb{N},\ F(\lambda, n) = \sum_{k=0}^{n} \mathbb{P}(X = k) = e^{-\lambda} \left(1 + \sum_{k=1}^{n} \frac{\lambda^k}{k!}\right).$$

Pour éviter de calculer λ^k et $k!$ quand on connait λ^{k-1} et $(k-1)!$, on utilise la relation : $\dfrac{\lambda^k}{k!} = \dfrac{\lambda}{k} \times \dfrac{\lambda^{k-1}}{(k-1)!}$.

```
import numpy as np

def F(lambd,n):
    somme = 1
    produit = 1
    for k in range(1,n+1):
        produit *= lambd/k
        somme += produit
    return somme * np.exp(-lambd)
```

2. On calcule successivement $F(\lambda, k)$ jusqu'à dépasser strictement x. Pour éviter de recalculer des termes déjà calculés, on n'appelle pas la fonction de la question précédente, mais on somme successivement les $\mathbb{P}(X = k)$.

```
def G(lambd,x):
    f = np.exp(-lambd)
    if f > x:
        return 0
    k = 0
    produit = np.exp(-lambd)
    while f <= x:
        k+= 1
        produit *= lambd/k
        f += produit
    return k
```

Retour à la planche 28

Corrigé de l'exercice de la planche 29

1. On parcourt (par élément ici) une liste de booléens et on ajoute 1 à notre compteur dès qu'on rencontre `True`.

```
def nb_vrai(liste):
    s = 0
    for x in liste:
        if x:
            s += 1
    return s
```

On peut aussi profiter du fait que les booléens sont en réalité un sous-type du type **`int`** (entiers) en Python. Les booléens `True` et `False` sont respectivement égaux à 1 et 0.

```
def nb_vrai(liste):
    s = 0
    for x in liste:
        s += x
    return s
```

2. Une idée est de parcourir la liste en retenant les indices du premier et du dernier `True` (s'ils existent) et de compter le nombre de `True`. La différence d'indices entre le premier et le dernier `True` indique alors si tous les `True` sont consécutifs.

```
def bloc_vrai(L):
    premier_True = -1
    compteur = 0
    for k in range(len(L)):
        if L[k]:
            compteur += 1
            dernier_True = k
            if premier_True == -1:
                premier_True = k
     return dernier_True - premier_True + 1 == compteur
```

Retour à la planche 29

Corrigé de l'exercice de la planche 30

1.
```
def compter(plateau):
    points_fixes = 0
    for k in range(len(plateau)):
        if plateau[k] == k:
            points_fixes += 1
    return points_fixes
```

2.
```
def avancer(plateau,p):
    k = 0
    for i in range(p):
        k = plateau[k]
    return k
```

Retour à la planche 30

Corrigé de l'exercice de la planche 31

1.
```
def nb_vides(grille):
    compteur = 0
    for i in range(len(grille)):
        for j in range(len(grille[0])):
            if grille[i][j] == None:
                compteur += 1
    return compteur
```

2. Il suffit de considérer tous les couples de numéros de lignes (i, j) où $0 \leqslant i < j \leqslant n-1$.

```
def test_lignes_identiques(grille):
    n = len(grille)
    for i in range(n-1):
        for j in range(i+1,n):
            if grille[i] == grille[j]:
                return True
    return False
```

Retour à la planche 31

Corrigé de l'exercice de la planche 32

1.
```
def deplacement(p, position):
    u = random.random()
    if position in [0,3]:
        return position
    elif u < p:
        return position - 1
    else:
        return position + 1
```

2.
```
def trajet(p,n):
    positions = [1]
    for k in range(n-1):
        positions.append(deplacement(p,positions[-1]))
    return positions
```

Retour à la planche 32

Corrigé de l'exercice de la planche 33

1.
```python
def strict_croissante(liste):
    n = len(liste)
    for k in range(n-1):
        if liste[k] >= liste[k+1]:
            return False
    return True
```

2.
```python
def strict_monotone(liste):
    n = len(liste)
    if liste[0] < liste[1]:
        return strict_croissante(liste)
    else:
        return strict_croissante(liste[::-1])
```

Retour à la planche 33

Corrigé de l'exercice de la planche 34

1. On construit une liste `P` initialement remplie de N zéros. On parcourt la liste `L` par élément (un parcours par indice est aussi possible); pour chaque valeur `v` lue dans `L`, on ajoute 1 au nombre `P[v]`, représentant le nombre de `v` lus dans `L`.

```
def comptage(L,N):
    P = [0 for k in range(N)]
    for v in L:
        P[v] += 1
    return P
```

2. On construit une liste en ajoutant chaque entier de 0 à $N-1$ autant de fois qu'il apparait dans `L`.

```
def tri(L,N):
    P = comptage(L,N)
    L2 = []
    for k in range(N):
        L2 += [k] * P[k]
    return L2
```

Retour à la planche 34

Corrigé de l'exercice de la planche 35

1. On utilise la définition par récurrence : $0! = 1$ et $n! = n(n-1)!$ pour tout $n \geqslant 1$.

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

2. On utilise la formule explicite : $0! = 0$ (par convention) et $\forall n \in \mathbb{N}^*,\ n! = 1 \times 2 \times 3 \times \cdots \times (n-1) \times n$.

```
def factoriel_iter(n):
    prod = 1
    for k in range(2,n+1): # on ne rentre pas dans cette boucle si n=0
        prod*=k
    return prod
```

Retour à la planche 35

Corrigé de l'exercice de la planche 36

On remarque que la fonction est récursive.

Le cas de base est celui où la liste ne contient qu'un seul élément.

Dans le cas contraire, pour générer la liste des permutations des entiers de 1 à n, on génère celle des entiers de 1 à $n-1$, notée L. Dans chaque permutation dans L, on insère n à toutes les positions possibles.

```
def permutations(n):
    if n == 1:
        return [[1]]
    else:
        L = permutations(n-1)
        L2 = []
        for p in L:
            for k in range(n):
                p2 = p[:n] + [n] + p[n:]
                L2.append(p2)
    return L2
```

Retour à la planche 36

Corrigé de l'exercice de la planche 37

1. On pourrait écrire une boucle conditionnelle (boucle **while**) mais il faudrait vérifier qu'on ne sort pas du tableau. On exploite ici le fait que l'instruction **return** arrête l'exécution de la fonction. On ne doit pas oublier de renvoyer le nombre de zéros en fin de boucle (dans le cas où on n'aurait jamais lu un seul 1).

```
def nombreZerosDroite(i, tab):
    nb = 0
    for k in range(i,len(tab)):
        if tab[i] == 0:
            nb +=1
        else:
            return nb
     return nb
```

2. À chaque fois qu'on lit un zéro, on utilise la fonction précédente pour compter le nombre `nb` de zéros contigus à partir de cette case. Pour éviter de relire des zéros déjà lus par la fonction `nombreZerosDroite`, on effectue un décalage d'indice de `nb+1` (le nombre de zéros consécutifs lus sans oublier le "un" qui suit (éventuellement).

```
def nombreZerosMax(tab):
    nbmax = 0
    i = 0
    while i < len(tab):
        nb = nombreZerosDroite(i, tab)
        if nb > nbmax:
            nbmax = nb
        i += nb + 1
    return nbmax
```

Retour à la planche 37

Corrigé de l'exercice de la planche 38

1. On trouve que `d(4) = [1,2,4]` et `d(10) = [1,2,5,10]`. La fonction `d` renvoie la liste des diviseurs de l'entier passé en argument.
2.
```
def somme(n):
    s = 0
    L=[1]
    for nombre in range(2,n):
        if n%nombre==0:
            s += nombre**2
    return s
```
3.
```
L = []
N=1000
for n in range(1,N+1):
    if somme(n)==n:
        L.append(n)
print(L)
```

On peut aussi écrire (pour les esthètes !) :

```
print([n for n in range(1,N+1) if somme(n)==n])
```

Retour à la planche 38

Corrigé de l'exercice de la planche 39

1.
```
def decalage(n):
    chaine = alphabet()
    return chaine[n:] + chaine[:n]
```

2.
```
def codage(chaine,n):
    alphaB = alphabet()
    alphaBdecale = decalage(n)
    crypt = {}
    for k in range(26):
        crypt[alphaB[k]] = alphaBdecale[k]
    chaine2 = ""
    for car in chaine:
        chaine2 += crypt[car]
    return chaine2
```

Retour à la planche 39

Corrigé de l'exercice de la planche 40

1. Le quotient de 1234 par 10 est 123 et son reste est 4. Le quotient de 123 par 10 est 12, et son reste est 3.
2.
```
def sommecube(n):
    somme
    while n >0:
        somme += (n%10)**3
        n = n//10
    return somme
```

3. On propose deux solutions.
```
# première solution
liste = []
for n in range(1001):
    if n==sommecube(n):
        liste.append(n)
print(liste)

# deuxième solution
print([n for n in range(1001) if n==sommecube(n)])
```

Retour à la planche 40

Corrigé de l'exercice de la planche 41

1. On propose deux versions : l'une en retenant la valeur du maximum et l'autre sans.

```
def indicesMax(L):
    maxi = L[0]
    indices_max = [0]
    for k in range(1,len(L)):
        if L[k] == maxi:
            indices_max.append(k)
       elif L[k] > maxi:
           maxi = L[k]
           indices_max = [k]
    return indices_max

def indicesMax(L):
    indices_max = [0]
    for k in range(1,len(L)):
        if L[k] == L[indices_max[0]]:
            indices_max.append(k)
       elif L[k] > indices_max[0]:
           indices_max = [k]
    return indices_max
```

2.

```
def deuxiemeMax(L):
    max1, max2 = L[0], L[1]
    if max2 > max1:
        max1, max2 = max2, max1
    for k in range(2,len(L)):
        if L[k] > max1:
            max1, max2 = L[k], max1
        elif max1 < L[k] < max2:
            max2 = L[k]
    return max2
```

Retour à la planche 41

Corrigé de l'exercice de la planche 42

1.
```
def diagonale(M):
    n, _ = np.shape(M)
    for i in range(n):
        for j in range(n):
            if i != j and M[i,j] != 0:
                return False
    return True
```

2. Après avoir déterminé si la matrice est carrée, on construit une liste `L` du nombre d'occurrences de chaque entier de 1 à n^2 de sorte que `L[k]` soit le nombre d'apparitions de `k+1` (attention au décalage d'indice) dans M.

```
def test(M):
    n, p = np.shape(M)
    if n != p:
        return False
    L = [0]*(n**2)
    for i in range(n):
        for j in range(n):
            L[M[i,j]-1] += 1 # attention au décalage d'indice
            if L[M[i,j]-1] > 1:
                return False
     return True
```

Retour à la planche 42

Corrigé de l'exercice de la planche 43

On vérifie initialement que la case à jouer est libre.

On considère ensuite l'une des 8 directions possibles partant de la case (i, j) - le couple (di, dj) un vecteur de directions. On se déplace dans la direction (di, dj) tant que la case considérée est sur le plateau et prise par l'adversaire. En sortie de boucle, si la case est valide et au joueur s'apprêtant à jouer, le coup est jouable. Si on n'a jamais renvoyé `True`, c'est qu'on ne peut retourner un pion adverse dans aucune des huit directions.

```
def coup_valide(plateau, couleur, i, j):
    if plateau[i,j] != 0:
        return False
    compteur = 0
    for (di,dj) in [(-1,-1),(-1,0),(-1,1),(0,-1),(0,1),(1,-1),(1,0),(1,1)]:
        x,y = i+di, j+dj
        while 0 <= x < 8 and 0 <= y < 8 and plateau[x,y] == -couleur:
            x, y = x+di, y+dj
            compteur += 1
        if 0 <= x < 8 and 0 <= y < 8 and plateau[x,y] == couleur and compteur > 0:
            return True
    return False
```

Retour à la planche 43

Corrigé de l'exercice de la planche 44

1.
```
def coupe(graphe,X):
    n = len(graphe)
    L = []
    for i in X:
        for j in graphe[i]:
            if j not in X:
                L.append([i,j])
    return L
```

2. On parcourt chaque sommet `i` du graphe. Pour chacun de ses voisins `j`, on détermine parmi les voisins `k` de `j`, les sommets qui sont à une distance différente de 0 et 1, i.e. les sommets qui sont à la fois distincts du sommet `i` lui-même et qui ne sont pas voisins de `i`.

```
def voisins_de_voisins(graphe):
    n = len(graphe)
    L = []
    for i in range(n):
        for j in graphe[i]:
            for k in graphe[j]:
                if i!=k and k not in graphe[i]:
                    L.append((i,k))
    return L
```

Pour éviter les doublons, on peut remplacer le test `i!=k` par `i<k`.

Retour à la planche 44