Synthèse : exercices

Ces exercices visent à réviser les points clé de l'algorithmie.

- Lors de la préparation, n'hésitez pas à envoyer un mail à votre professeur si besoin.
- Ils ont à préparer sous forme informatique à l'aide du logiciel Spyder.
- Ils seront corrigés lors du prochain TP. Vous devrez venir avec vos programmes (version informatique) qui seront analysés lors la séance.
- Chacun avancera à son rythme.

Exercice 1 : **Nombre complexe**

Créer une fonction qui prendra comme arguments le module et l'argument d'un nombre complexe. La fonction retournera la partie réelle et la partie imaginaire du nombre.
Elle affichera aussi le nombre dans le plan complexe en utilisant le module `matplotlib`.

Exercice 2 : **La muraille**

Le village dans lequel vous avez passé la nuit est en pleine effervescence au matin : encore une attaque de loups pendant la nuit !
C'est décidé, il va falloir construire une grande palissade tout autour du village. Les habitants insistent pour que cette clôture soit rectangulaire et ait une face au Nord, une au Sud, une à l'Est et une à l'Ouest, quitte à devoir travailler un peu plus que nécessaire. Ils ont maintenant besoin de votre aide pour savoir la quantité de bois dont ils vont avoir besoin pour construire cette palissade.

Partie 1 : Ce que doit faire votre programme :

Le programme doit d'abord lire un entier strictement positif correspondant au nombre de maisons. Ensuite, pour chaque maison, il doit lire la position horizontale (l'abscisse, le "x") et sa position verticale (l'ordonnée, le "y") de cette maison. Toutes les abscisses et ordonnées sont des entiers compris entre zéro et 1 million.

Le programme doit alors afficher le périmètre de la plus petite clôture rectangulaire englobant toutes les maisons. Ce rectangle doit avoir ses côtés parallèles aux axes du repère, comme montré sur l'illustration.

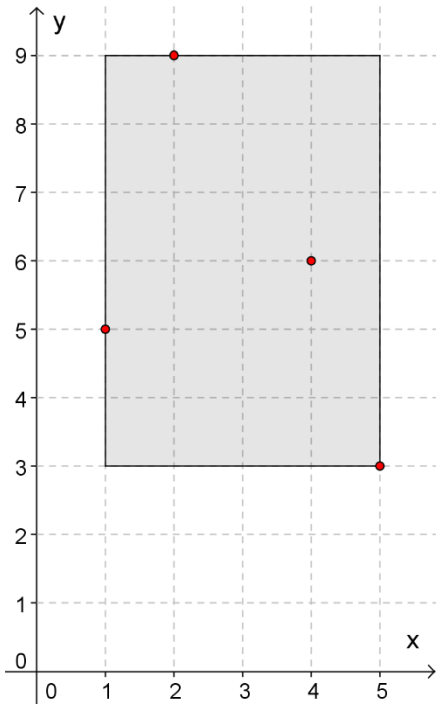
Partie 2 : Exemple

entrée :

```
4
1
5
5
3
4
6
2
9
```

sortie :

```
20
```



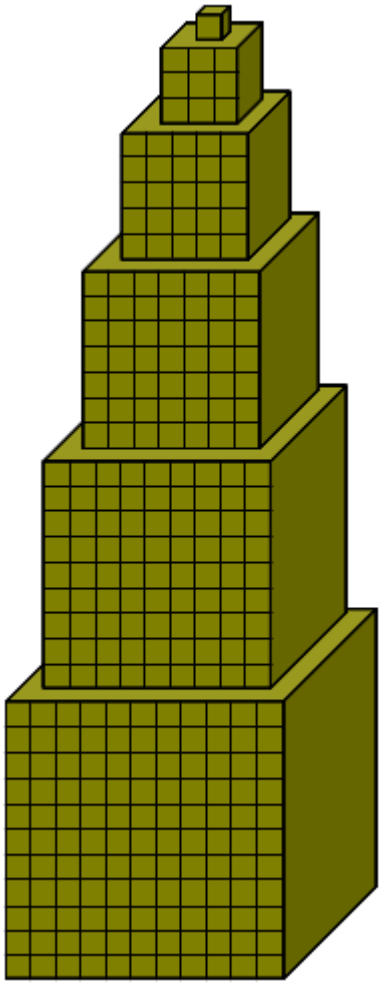
Exercice 3 : **Tri séquentiel**

La liste L du programme « exercice4.py » contient 100 nombres.

L'objectif est de ranger ces nombres par ordre croissant.

- ☞ Créer une fonction qui revoit la valeur minimale de la liste en s'appuyant sur l'algorithme présenté en cours (recherche d'un maximum).
- ☞ Compléter cette fonction pour qu'elle renvoie cette valeur minimale et la liste sans cette valeur minimale.
- ☞ En utilisant cette fonction dans un boucle, créer une nouvelle liste contenant les éléments de la liste L mais triés dans l'ordre croissant.

Exercice 4 : **Pyramide**



Les enfants de la classe de maternelle décident de construire une très grande tour à l'aide de petits cubes en bois. Ils savent exactement la forme qu'ils souhaitent pour leur tour, mais ils n'arrivent pas à savoir s'ils auront suffisamment de cubes pour la construire. Ils vous demandent de les aider à calculer le nombre de cubes nécessaires.

Ce que doit faire votre programme :

L'objectif est de construire une tour à l'aide de petits cubes en bois, sachant que la forme de cette tour consiste en un ensemble de grands cubes placés les uns au-dessus des autres. La base de la tour est un cube de taille 17×17×17, c'est-à-dire composé de 17×17×17 = 4 913 petits cubes. Sur ce cube est posé un autre cube de taille 15×15×15. Au-dessus de ce dernier se trouve un cube de 13×13×13. La tour continue ainsi jusqu'à atteindre le sommet, qui consiste en un cube de taille 1×1×1.

Exemple d'une tour de 6 étages allant de 1×1×1 cubes à 11×11×11 cubes :

Q1.

Votre programme doit calculer et afficher le nombre total de petits cubes nécessaires pour construire la pyramide. Effectuez les calculs dans le programme en y intégrant une boucle.

Q2.

Votre programme doit déterminer et afficher le nombre d'étage qu'il est possible de réaliser avec n petits cubes (n est renseigné par l'utilisateur). Effectuez les calculs dans le programme en y intégrant une boucle.

Exercice 5 : **Exercice : algorithmie sur les tableaux**

La résolution d'une grille de Sudoku est une gymnastique du cerveau qui peut ˆêtre assimilée à un décodage « correcteur d'effacement ». En effet, à partir d'une grille presque vide, il est possible (pour une grille initiale bien faite) de la compléter d'une unique manière.
L'objectif de cet exercice est de mettre en œuvre une méthode naïve permettant de compléter une grille initiale.

Notations : Pour m et n deux entiers naturels, $[m;n]$ désigne l'ensemble des entiers k tels que $m < k < n$.

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

```
L= [[0,6,0,0,0,0,2,0,5],[4,0,0,9,2,1,0,0,0],
[0,7,0,0,0,8,0,0,1],[0,0,0,0,0,5,0,0,9],
[6,4,0,0,0,0,0,7,3],[1,0,0,4,0,0,0,0,0],
[3,0,0,7,0,0,0,6,0],[0,0,0,1,4,6,0,0,2],
[2,0,6,0,0,0,0,1,0]]
```

Une grille de Sudoku est une grille de taille 9 × 9, découpée en 9 carrés de taille 3×3. Le but est de la remplir avec des chiffres de $[1;9]$, de sorte que chaque ligne, chaque colonne et chacun des 9 carres de taille 3 × 3 contienne une et une seule fois chaque entier de $[1;9]$. On dira alors que la grille est complète. En point de départ, certaines cases sont déjà remplies et on fera l'hypothèse que le Sudoku qui nous intéresse est bien écrit, c'est-à-dire qu'il possède une unique solution.
On représente en Python une grille de Sudoku par un tableau de taille 9 × 9, c'est-à-dire une liste de 9 listes de taille 9, dans laquelle les cases non remplies sont associées au chiffre 0 (voir exemple ci-dessus).
Les 9 carrés de taille 3×3 sont numérotes du haut à gauche jusqu'en bas à droite. Ainsi, sur cette grille, le carre 0, en haut et à gauche, contient les chiffres 6, 4 et 7; le carre 1, en haut et au milieu, contient les chiffres 9, 2, 1 et 8; le carre 8, en bas et à droite, contient les chiffres 6, 2 et 1.

On rappelle que les lignes du Sudoku sont alors les éléments de L accessibles par L[0],...,L[8]. L'élément de la case (i,j) est accessible par `L[i][j]`.
Remarque : on fera bien attention, dans l'ensemble de ce sujet, aux indices des tableaux. Les lignes, ainsi que les colonnes, sont indicées de 0 à 8.

Partie 1 : Vérification d'une grille complète

Si une grille de Sudoku est complète, alors pour chacune des lignes, chacune des colonnes et chacun des carrés de taille 3×3, la somme des chiffres fait 45. (La réciproque n'est pas vraie). Nous cherchons à vérifier que chaque ligne, chaque colonne et chaque carré respecte cette propriété pour vérifier la cohérence de l'ensemble de la grille.

Q1. Ecrire une fonction `ligne_complete(L,i)` qui prend une liste Sudoku `L` et un entier `i` entre 0 et 8, et renvoie `True` si la ligne `i` du Sudoku L vérifie les conditions de remplissage d'un Sudoku, et `False` sinon.

Q2. Ecrire une fonction `colonne_complete(L,i)` qui prend une liste Sudoku `L` et un entier `i` entre 0 et 8, et renvoie `True` si la colonne `i` du Sudoku L vérifie les conditions de remplissage d'un Sudoku, et `False` sinon.

Le programme définit de même la fonction `carre_complet(L,i)` pour le carre i. (on ne demande pas de l'écrire)

Q3. Ecrire une fonction `complet(L)` qui prend une liste Sudoku `L` comme argument, et qui renvoie `True` si la grille est complète, `False` sinon.

Partie 2 : Résolution d'une grille de Sodoku

Préparation des fonctions nécessaires

Q4. Ecrire une fonction `ligne(L,i)`, qui renvoie la liste des nombres qui apparaissent sur la ligne d'indice `i`.

Ainsi, avec la grille donnée dans l'énonce, on doit obtenir :

```
>>> ligne(L,0)
[6, 2, 5]
```

Le programme définit alors, de la même manière, les fonctions :

- `colonne(L,j)` qui renvoie la liste des nombres compris entre 1 et 9 qui apparaissent dans la colonne `j` ;
- `carre(L,i,j)` qui renvoie la liste des nombres compris entre 1 et 9 qui apparaissent dans le carre 3 × 3 auquel appartient la case (i,j).

(On ne demande pas d'écrire les codes associés à ces fonctions).

Ainsi, avec la grille donnée dans l'énonce, on doit obtenir :

```
>>> carre(L,4,6)
[9, 7, 3]
>>>carre(L,4,5)
[5, 4]
```

Q5. Ecrire une fonction `chiffres_ok(L,i,j)` qui renvoie la liste des chiffres que l'on peut écrire en case (i,j).

Par exemple, avec la grille initiale :

```
>>> chiffres_ok(L,4,2)
[2, 5, 8, 9]
```

On pourra, dans la suite du sujet, utiliser ces fonctions définies précédemment.

Résolution: Algorithme naïf

Naïvement, on commence par compléter les cases n'ayant qu'une seule possibilité. Nous prendrons dans la suite comme Sudoku :

```
M= [[2, 0, 0, 0, 9, 0, 3, 0, 0], [0, 1, 9, 0, 8, 0, 0, 7, 4],
[0, 0, 8, 4, 0, 0, 6, 2, 0], [5, 9, 0, 6, 2, 1, 0, 0, 0],
[0, 2, 7, 0, 0, 0, 1, 6, 0], [0, 0, 0, 5, 7, 4, 0, 9, 3],
[0, 8, 5, 0, 0, 9, 7, 0, 0], [9, 3, 0, 0, 5, 0, 8, 4, 0],
[0, 0, 2, 0, 6, 0, 0, 0, 1]]
```

Q6. A partir des fonctions précédentes, écrire une fonction `nb_possible(L,i,j)`, indiquant le nombre de chiffres possibles à la case (*i,j*).

Q7. On souhaite disposer de la fonction `un_tour(L)` qui parcourt l'ensemble des cases du Sudoku et qui complète les cases dans le cas où il n'y a qu'un chiffre possible, et renvoie, d'une part, `L` et, d'autre part, True s'il y a eu un changement, et False sinon.

Par exemple, en partant de la grille initiale M :

```
>>> un_tour(M)
[[2, 0, 0, 0, 9, 0, 3, 0, 0], [0, 1, 9, 0, 8, 0, 5, 7, 4],
[0, 0, 8, 4, 0, 0, 6, 2, 9], [5, 9, 0, 6, 2, 1, 4, 8, 7],
[0, 2, 7, 0, 3, 8, 1, 6, 5], [0, 6, 1, 5, 7, 4, 2, 9, 3],
[0, 8, 5, 0, 0, 9, 7, 3, 0], [9, 3, 6, 0, 5, 0, 8, 4, 2],
[0, 0, 2, 0, 6, 0, 9, 5, 1]] , True
```

Q8. Ecrire une fonction `complete(L)` qui exécute la fonction `un_tour` tant qu'elle modifie la liste, et renvoie `True` si la grille est complétée, et `False` sinon.

Exercice 6 : **Mots-mélés**

Le tableau donné dans le programme intitulé « exercice6.py » est stocké dans la mémoire sous forme d'un *array* formé de caractères noté `A`. Ses dimensions sont de 15 colonnes et 10 lignes.

L'objectif est de retrouver une liste de mot (notée L), qui sont écrits dans le tableau :

- horizontalement (de gauche à droite),
- verticalement (de haut en bas),
- obliquement (du haut-gauche vers le bas-droit).

Q1 : Proposer un programme Python qui indique la position de l'initiale de chaque mot écrit horizontalement.

Q2 : Compléter le programme pour qu'il indique la position de l'initiale de tous les mots et leur orientation d'écriture.

L	V	O	D	Y	T	R	O	M	P	E	T	T	E	E
B	C	X	X	P	I	M	O	R	I	L	L	E	V	Y
I	O	E	J	R	L	S	P	O	R	E	S	L	D	E
Q	K	L	P	O	Y	E	V	A	B	I	O	F	T	A
I	N	A	E	E	A	U	U	S	G	V	F	I	X	N
J	H	M	Y	T	I	A	E	R	I	A	N	P	D	N
E	L	A	M	E	L	L	E	S	O	A	R	E	N	E
L	X	X	N	X	A	H	H	V	M	T	I	I	H	A
J	W	Y	G	H	A	V	C	A	I	P	E	H	C	U
L	P	O	M	I	C	C	H	A	P	E	A	U	J	Y