Devoir surveillé d'informatique

Consignes

- La calculatrice n'est **pas autorisée**.
- On pourra toujours librement utiliser une fonction demandée à une question précédente même si cette question n'a pas été traitée.
- Veillez à présenter vos idées et vos réponses partielles même si vous ne trouvez pas la solution complète à une question.
- La clarté et la lisibilité de la rédaction et des programmes sont des éléments de notation.

❑ Exercice 1 : *Questions de cours*

1. Sur la récursivité
   a) Donner la définition d'une fonction récursive.
   b) Quel serait le comportement d'une fonction récursive dépourvue de condition d'arrêt ?
   c) Ecrire une fonction *iterative* `occurrences` qui prend en argument une chaine de caractères `s` et un caractère `c` et qui renvoie le nombre d'apparitions du caractère `c` dans la chaine `s`. Par exemples, `occurrences("un exemple",'e')` renvoie `3` et `occurrences("un exemple",'a')` renvoie `0`.
   d) Ecrire une version *récursive* de cette fonction.
   ⛑ Indication : on rappelle que si `s` une chaine de caractères alors `s[1:]` est la chaine `s` privée de son premier caractère. Par exemple `"Python"[1:]` est la chaine `"ython"`.
2. Sur le tri par sélection
   a) Expliquer brièvement le principe de l'algorithme du *tri par sélection* et donner l'évolution du contenu de la liste `[42, 28, 11, 9, 33]` lorsqu'elle est triée en utilisant cet algorithme.
   b) Ecrire en Python une fonction `echange` qui prend en argument une liste `lst` ainsi que deux entiers `i` et `j` ne renvoie rien et échange les éléments d'indice `i` et `j` de cette liste. Par exemple, si `lst = [25, 17, 34, 22, 37]` alors `echange(lst, 1, 3)` doit modifier la liste `lst` en `[25, 22, 34, 17, 37]`. On vérifiera les préconditions sur `i` et `j` à l'aide d'instructions `assert`.
   c) Ecrire en Python une fonction `indice_min_depuis` qui prend en argument une liste `lst` et un entier `i` et renvoie l'indice du plus petit élément de la liste `lst` à partir de l'indice `i`. Par exemple, si `lst = [25, 17, 34, 22, 37]` alors `indice_min_depuis(lst, 2)` doit renvoyer `3` (c'est à dire l'indice de `22` qui est l'élément minimal à partir de celui d'indice 2).
   d) Ecrire en Python une fonction `tri_selection` qui prend en argument une liste `lst` et modifie cette liste de façon à la trier par ordre croissant en utilisant l'algorithme du tri par sélection. On utilisera les fonctions `echange` et `indice_min_depuis` écrites aux questions précédentes.
   e) Un ordinateur a pris 3 secondes pour trier une liste de 250 000 éléments en utilisant un tri par sélection. Donner une estimation (aucune justification n'est demandée) du temps que prendrait cet ordinateur pour trier une liste de 1 000 000 éléments en utilisant le même algorithme.
3. Sur la représentation binaire des entiers
   a) Donner l'écriture en base 2 de $(172)_{10}$.
   b) Ecrire $(10011101)_2$ en base 10.
   c) Quel est le plus grand nombre représentable en base 2 en utilisant 10 bits ?
   d) On suppose que l'on travaille sur des entiers codés sur 8 bits en complément à 2. A quel nombre entier en base 10 correspond $(11011001)_2$?

❑ Exercice 2 : *Permutation des éléments d'une liste*

1. Le but de cette partie est de dénérer *une* permutation aléatoire des éléments d'une liste
   Le *mélange de Fischer-Yates* (aussi appelé *mélange de Knuth*) est un algorithme de mélange aléatoire des éléments d'une liste. Il consiste à parcourir la liste en partant de la fin avec un indice `i` et à échanger l'élément d'indice `i` avec un élément choisi aléatoirement parmi les éléments ayant un indice inférieur *ou égal*. Voici un exemple du déroulement de l'algorithme (crédit : Wikipedia) sur la liste `['E','L','V','I','S]` on part de la fin (donc de l'indice 4) :

Etat de la liste	Indice `i`	Choix aléatoire dans $[\![0;i]\!]$	Liste après échange
`['E','L','V','I','S']`	4	4	`['E','L','V','I','S']`
`['E','L','V','I','S']`	3	0	`['I','L','V','E','S']`
`['I','L','V','E','S']`	2	2	`['I','L','V','E','S']`
`['I','L','V','E','S']`	1	0	`['L','I','V','E','S']`

a) Donner le déroulement de l'algorithme sur la liste `['A','B','C','D','E','F']` en supposant que les choix aléatoires successifs sont : `2, 0, 3, 1, 0`.

b) Afin de générer une entier aléatoire on veut utiliser la fonction `randint` du module `math`. Que faut-il écrire en début de programme pour que cette fonction soit utilisable ?

c) Ecrire en Python une fonction *itérative*, `melange_iteratif` qui prend en argument une liste `lst` ne renvoie rien et mélange aléatoirement les éléments de la liste en utilisant l'algorithme de Fischer-Yates. On supposera déjà écrite une fonction `echange` qui prend en argument une liste ainsi que deux indices et échange les éléments situés à ces indices.

d) Ecrire en Python une fonction *récursive*, `melange_recursif` qui prend en argument une liste `lst` ne renvoie rien et mélange aléatoirement les éléments de la liste en utilisant l'algorithme de Fischer-Yates.

✪ Indication : on pourra fournir à cette fonction un paramètre entier `i` indiquant l'indice de l'élément qui sera potentiellement échangé avec l'un de ceux qui le précède.

2. Le but de cette partie est de générer la liste de *toutes* les permutations d'une liste.
On souhaite écrire une fonction `permutation` qui prend en argument une liste et renvoie la liste de toutes les permutations de cette liste. Par exemple `permutation(['I','T','C'])` doit renvoyer la liste : `[['I','T','C'], ['I','C','T'], ['T','I','C'], ['T','C','I'], ['C','T','I'], ['C','I','T']]`.
Pour cela on propose l'algorithme *récursif* suivant : on commence par générer la permutation de la sous liste commençant à l'indice 1 puis on insère le premier élément à toutes les positions possibles dans les listes obtenues. Sur l'exemple de la liste `['I','T','C']`, on génère donc les permutations de `['T','C']` c'est à dire `[['T','C'], ['C','T']]`. Puis on insère `'I'` à toutes les positions possibles dans ces listes ce qui donne :
`[['I','T','C'], ['T','I','C'], ['T','C','I'], ['I','C','T'], ['C','I','T'], ['C','T','I']]`.

a) Ecrire la fonction `insere` qui prend en argument un élément `elt` une liste `lst` et renvoie la liste de listes obtenues en insérant `elt` à toutes les positions possibles dans `lst`. Par exemple, si `lst = ['T','C']` et `elt = 'I'` alors `insere(elt, lst)` doit renvoyer `[['I','T','C'], ['T','I','C'], ['T','C','I']]`.

b) Ecrire en Python une fonction *récursive*, `permutation` qui prend en argument une liste `lst` et renvoie la liste des permutations de la liste `lst`.

✪ Indication : on pourra générer *récursivement* les permutations de `lst` privé de son premier élément puis insérer cet élément à tous les emplacements possibles grâce à la fonction précédente.

❑ **Exercice 3** : *Problème du sac à dos*

On dispose d'un sac à dos pouvant contenir un poids maximal noté P et de n objets ayant chacun un poids $(p_i)_{0\leqslant i\leqslant n-1}$ et une valeur $(v_i)_{0\leqslant i\leqslant n-1}$. On cherche à remplir le sac à dos de manière à maximiser la valeur totale des objets contenus dans le sac sans dépasser le poids maximal P. Par exemple, si on dispose des objets suivants :

— un objet de poids $p_0 = 4$ et de valeur $v_0 = 20$,

— un objet de poids $p_1 = 5$ et de valeur $v_1 = 28$,

— un objet de poids $p_2 = 6$ et de valeur $v_2 = 36$,

— un objet de poids $p_3 = 7$ et de valeur $v_3 = 50$,

et qu'on suppose que le poids maximal du sac est 10 alors un choix possible serait de prendre l'objet 3, aucun autre objet ne rentre alors dans le sac et la valeur du sac est de 50 avec un poids de 7. Une autre possibilité plus intéressante serait de choisir les objets 0 et 2, la valeur totale serait alors de 56 et le poids du sac de 10.

Dans toute la suite de l'exercice on supposera que les poids et les valeurs des objets sont fournis sous la forme d'une liste de Python contenant les tuples `(poids, valeur)` représentant les objets. Par exemple, les objets précédents seraient représentés par la liste suivante :

```
objets = [(4, 20), (5, 28), (6, 36), (7, 50)]
```

1. On considère la stratégie gloutonne suivante : on trie les objets par ordre décroissant de leur rapport valeur/poids et on les prend dans cet ordre jusqu'à ce que le poids maximal soit atteint.
   a) Vérifier qu'en appliquant cette stratégie à la liste d'objets :
   ```
   [(4, 30), (5, 34), (6, 36), (7, 49), (10, 74)]
   ```
   et un poids maximal de 10, on n'obtient pas la meilleure solution.
   b) Ecrire une fonction `glouton`, qui prend en arguments une liste d'objets qu'on suppose *déjà triée par ordre décroissant du rapport valeur/poids* et un poids maximal et qui renvoie la valeur maximale que l'on peut obtenir en appliquant la stratégie gloutonne.
   c) Montrer sur un exemple de votre choix (avec au moins 3 objets) que si on choisit de trier les objets par ordre croissant de poids (c'est à dire qu'on choisit en premier les objets les plus légers), alors la stratégie gloutonne ne donne pas non plus la solution optimale.
2. La recherche exhaustive consiste à énumérer tous les choix possibles d'objet et à calculer la valeur ainsi que le poids pour chaque choix, on retient alors le choix qui maximise la valeur du sac sans dépasser le poids maximal. On propose de représenter un choix d'objets par une liste contenant des 0 et des 1. Si le i-ème élément de la liste vaut 1 alors l'objet i est choisi, s'il vaut 0 alors l'objet i n'est pas choisi. Par exemple, pour les objets précédents, le choix de prendre uniquement l'objet 3 serait représenté par la liste `[0, 0, 0, 1]` et le choix de prendre les objets 0 et 2 serait représenté par la liste `[1, 0, 1, 0]`.
   a) Justifier rapidement que le nombre possible de choix d'objet est 2^n.
   b) Afin de générer tous les choix possibles on propose de parcourir les entiers de 0 à 2^n-1 et de convertir chaque entier en son écriture binaire. Ecrire une fonction `choix` qui prend en argument un entier `n` et un entier `k` tel que $0 \leqslant k \leqslant 2^n - 1$ et renvoie la liste des chiffres de k en base 2 sur n bits. Par exemples `choix(8,17) renvoie [0,0,0,1,0,0,0,1]` (car on utilise `8` bits et $(00010001)_2 = 17$), et `choix(8,121) renvoie [0,1, 1, 1, 1, 0, 0, 1]`
   🛟 Indication : on pourra utiliser la fonction `bin` de Python qui prend en argument un nombre entier et renvoie son écriture en base 2 précédée de `0b` sous la forme d'une chaine de caractère, par exemple `bin(17)` renvoie `"0b10001"`.
   c) Ecrire une fonction `valeur_poids` qui prend en arguments, une liste d'objets ainsi qu'un choix d'objet (sous la forme indiquée ci-dessus) et qui renvoie le poids et la valeur du sac correspondant à ce choix. Par exemple avec la liste `objets` donnée en exemple plus haut, `valeur_poids(objets, [1, 0, 1, 0])` doit renvoyer $(56, 10)$.
   d) Ecrire une fonction `recherche_exhaustive` qui prend en argument une liste d'objets et un poids maximal et qui renvoie la valeur maximale que l'on peut obtenir en appliquant la stratégie de recherche exhaustive. C'est à dire en testant tous les choix possibles d'objets et en retenant le choix de valeur maximale qui ne dépasse pas le poids maximal autorisé.