NSI Terminale	TD/TP – Bilan structures linéaires de données et dictionnaires	Structures de données Séquence 3

Exercice 1 : QCM

1) Quelle opération ne fait pas partie de l'interface d'une pile ?
- ajouter un élément à la pile
- retirer l'élément le plus récent de la pile
- retirer l'élément le plus ancien de la pile

2) Quelle opération ne fait pas partie de l'interface d'une file ?
- ajouter un élément à la file
- retirer l'élément le plus récent de la file
- retirer l'élément le plus ancien de la file

3) Un tableau associatif permet de créer une association clé □ valeur. Pour stocker des numéros de téléphone à l'aide d'un tableau associatif, quelle solution semble préférable, dans la mesure où une personne peut avoir plusieurs numéros de téléphone ?
- la clé est le numéro de téléphone et la valeur est le nom correspondant
- la clé est le nom et la valeur est le numéro de téléphone correspondant
- la clé est le nom et la valeur est la collection des numéros de téléphone correspondants
- la clé est un simple numéro unique et la valeur est le couple nom/téléphone

4) Le type `list` utilisé dans Python correspond le mieux :
- au type abstrait liste chaînée
- au type abstrait file
- au type abstrait tableau (qui serait dynamique et l'accès à la donnée se fait à partir de l'indice)

5) La récupération d'un élément d'un objet Python de type list, **connaissant son indice** :
- nécessite un temps proportionnel au nombre d'éléments de la liste
- s'effectue en temps constant
- est impossible

6) La récupération d'un élément d'un objet Python de type list, **ne connaissant pas son indice** :
- nécessite un temps proportionnel au nombre d'éléments de la liste
- s'effectue en temps constant
- est impossible

7) Accéder à une valeur dans un dictionnaire à partir de la clé à laquelle la valeur est associée est réalisé :
- en un temps proportionnel à la taille du dictionnaire
- en un temps constant

8) Sur un objet de type `list` Python, quelles opérations sont faites en un temps indépendant de la longueur de la liste ?
- supprimer le premier élément
- supprimer le dernier élément
- ajouter un élément au début (en position 0)
- ajouter un élément à la fin

Exercice 2 :

Préciser quelle est la structure de donnée à privilégier pour chacune de ces taches :

1. Représenter un répertoire téléphonique. Dictionnaire
2. Stocker l'historique des actions effectuées dans un logiciel et disposer d'une commande Annuler. Pile
3. Envoyer des fichiers au serveur impression. File
4. On souhaite stocker un texte très long que l'on souhaite pouvoir modifier. Liste chaînée
5. Gérer le flux des personnes arrivant à la caisse d'allocations familiales ; File
6. Mise en place d'un mécanisme annuler/refaire pour un traitement de texte ; Pile
7. Lors d'une partie échec, on veut enregistrer l'ensemble des coups joués et pouvoir les consulter. Dictionnaire/Liste chaînée

Exercice 3 :

Un fabricant décide de créer des tee-shirts dont la taille peut être : XS, S, M, L, XL, XXL. À chaque taille son prix ; il adopte le principe suivant : 8 € pour la taille XS et il ajoute 2 € en passant à la taille supérieure, jusqu'au XXL.

1) Implémenter en Python ces informations dans la structure de données la mieux adaptée.
   prix = {"XS": 8,"S": 10,"M": 12,"L": 14,"XL": 16,"XLL":18}
2) Ce même fabriquant décide de changer sa façon de fixer le prix de revente des tee-shirts. Ceux dont la taille est XS sont toujours à 8 €, mais cette fois-ci, pour passer d'une taille à la suivante, il ajoute au prix de la taille inférieure la moitié de sa racine carrée. Par exemple, pour obtenir le prix des tailles S, il fait : $8+\sqrt{8}\div 2\approx 9{,}41$

Proposer un programme Python qui automatise ces calculs et les enregistre dans une structure de données adaptées.

Exercice 4 : d'une liste à un dictionnaire

Écrire en Python une fonction `list_to_dico(liste)` qui admet en argument une liste de la forme `[clé 1, valeur 1, clé 2, valeur 2, …]` et qui retourne le dictionnaire correspondant de la forme : `{clé 1 : valeur 1, clé 2 : valeur 2, …}`.

Ex :

```
>>> L = ['France', 'Paris', 'Royaune-Uni', 'Londres', 'Allemagne', 'Berlin']
>>> print(list_to_dico(L))
  {'France': 'Paris', 'Royaune-Uni': 'Londres', 'Allemagne': 'Berlin'}
```

Exercice 5 : implémentation d'un dictionnaire en POO

a) Construire une classe `dict1` qui n'aura qu'un seul attribut `Liste` : une liste vide.
Construire une méthode `ajouter` qui prend comme argument une liste [cle,valeur] et qui ajoute cette liste à l'attribut Liste.
Construire une méthode permettant l'affichage comme ci-dessous :

```
L = dict1()
L.ajouter(["cle1","val1"])
L.ajouter(["cle2","val2"])
print(L) >>>{cle1 : val1 ; cle2 : val2 ; }
```

b) Construire les méthodes supprimer, modifier, rechercher telles que :

```
L.modifier("cle1","Val1modif")
L.supprimer("cle2")
print(L) >>> {cle1 : Val1modif ; }
print(L.rechercher("cle1")) >>> Val1modif
print(L.rechercher("cle2")) >>> None
```

Exercice 6 : chiffrement

La fonction `ord(<caractère>)` renvoie la valeur numérique Unicode du caractère spécifié. Par exemple, « `ord('f')` » revoie la valeur 102.
La fonction inverse est `chr(<entier>)`. Par exemple, « `chr(102)` » revoie « f ».
Pour chiffrer un mot M, on décide de procéder ainsi : pour chaque caractère *c* de M :

- On calcule (3 × ord(*c*) – 61) % 91, puis on ajoute 33. Le résultat est noté *resultat* ;

- On trouve ensuite le caractère correspondant à *resultat* avec `chr(resultat)` : on note ce caractère c_1.
- Si *resultat* est pair, on calcule (2 × ord(*c*) – 31) % 91, puis on ajoute 33, et on trouve le caractère correspondant à ce résultat, noté c_2. Dans ce cas, le caractère *c* est chiffré en c_1 c_2 ;
- Si *resultat* est impair, on calcule (2 × ord(*c*) – 32) % 91, puis on ajoute 33, que l'on transforme en son caractère c_2, puis on calcule (3 × ord(*c*) – 60) % 91, puis on ajoute 33, que l'on transforme en son caractère c_3. Dans ce cas, le caractère *c* est chiffré en c_1 c_2 c_3.

Par exemple, pour chiffrer « P » :

- `ord('P') = 80`, donc on calcule :
  (3 × 80 – 61) % 91 = 88, puis on ajoute 33 : *resultat* = 88 + 33 = 121,
  qui est impair et qui correspond à `chr(121) = 'y'`. Donc c_1 correspond au caractère : 'y' ;
- *resultat* est impair, donc on calcule :
  (2 × 80 – 32) % 91 = 37, puis on ajoute 33 : 37 + 33 = 70,
  qui correspond à c_2 = `chr(70) = 'F'` ;
- ensuite, on calcule :
  (3 × 80 – 60) % 91 = 89, puis on ajoute 33 : 89 + 33 = 122,
  qui correspond à c_3 = `chr(122) = 'z'` ;
- « P » est donc chiffré en « `yFz` ».

Afin d'optimiser le temps de cryptage, on souhaite implémenter une structure de données nous permettant d'obtenir le cryptage de tous les caractères dont la valeur numérique Unicode est comprise entre 32 et 122 compris.

1) Implémenter la structure de données adéquate.

2) Chiffrer le message : « informatique ».