ET Piles, Files, POO, algorithmique

Durée : 1h

Rappel : choisir **UN** exercice parmi les deux suivants.

Exercice 1 : files, programmation Python

Cet exercice porte sur la notion de file et sur la programmation en Python du programme de Terminale.

Rappel : une file est une structure de données abstraite fondée sur le principe « premier arrivé, premier servi. »

1. Laquelle de ces deux situations est associée à une structure de file ?

Situation 1 : « *Je cuisine des crêpes. Dès qu'une crêpe est faite, je la place sur un plat. Chaque nouvelle crêpe est placée sur la crêpe précédente. Quand je vais manger une de ces crêpes, je commencerai par la crêpe située en haut de mon tas.* »

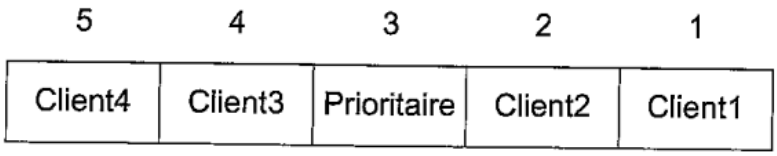
Situation 2 : « *Je dispose d'une imprimante placée en réseau dans ma salle de classe équipée d'ordinateurs, tous en réseau. Tous les élèves présents ont accès à cette imprimante, via le réseau. A la fin de la séance, les élèves envoient leur production à l'impression. Les documents sont imprimés dans l'ordre d'arrivée* ».

On modélise la gestion de l'attente à une caisse de supermarché. Les clients sont associés à une **File**. Les personnes prioritaires passeront devant les autres clients sans attendre. Nous ne tenons pas compte dans cet exercice de graduation dans les « priorités ». Nous ne tenons pas compte de personnes arrivant ensemble en caisse : il y aura toujours un des deux clients arrivé avant l'autre. On appelle 1ère personne dans la queue, la première personne qui est juste derrière le client en train de payer ses articles en caisse. En d'autres termes, le client qui règle ses articles ne compte plus, puisqu'il n'attend plus dans la queue.

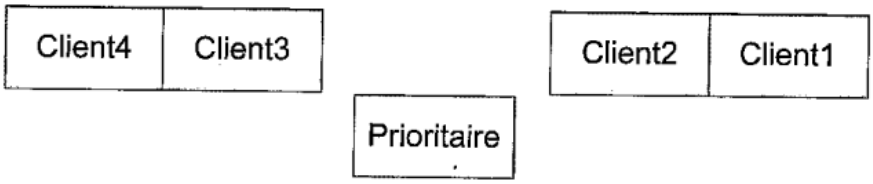
Voici les règles appliquées :

- La 1ère personne arrivée se place dans la queue.
- Le contrôleur « relations clients » du supermarché vérifie les priorités « clients ».
- Si une personne dispose d'un accès prioritaire, elle passe en position 1, et de ce fait, tout le reste des clients dans l'attente rétrograde d'une place.
- Si deux personnes sont prioritaires, la 2ème arrivée se placera derrière la 1ère arrivée « Prioritaire » et ainsi de suite avec tout nouveau prioritaire.

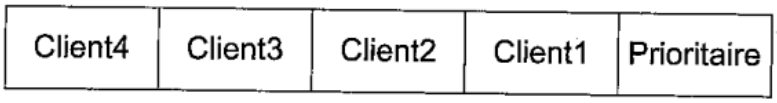
Exemple : À un instant t, la file est dans l'état ci-dessous :



La réorganisation grâce au contrôleur « relations clients » se met en place : « Client1 » et « Client2 » font un pas de côté, de même pour les personnes derrière « Prioritaire », en respectant leur ordre d'arrivée.



Le « Prioritaire » s'avance et se retrouve en position 1. Puis la file finale se réorganise.



Nous utiliserons uniquement les quatre fonctions primitives suivantes pour la suite des questions.

Structure de données abstraite: **File**
Utilise: Élément, Booléen
Opérations : • creer_file_vide : Ø → File creer_file_vide() renvoie une file vide • est_vide : File → Booléen est_vide (File) renvoie True si File est vide, False sinon • enfiler: File, Élément → Rien enfiler(File, element) ajoute element dans la file File • defiler: File → Élément defiler(File) renvoie l'élément en tête de la file File tout en le retirant de la file

On suppose que le contenu de la file **F** est le suivant:

Queue *Tête*

Client4	Prioritaire	Client3	Client2	Client1

2.a) On considère la file **V** définie dans le code ci-dessous.

Quel sera le contenu de **V**, **F** et de la variable val à la suite de ces instructions Python?

```
1   V = creer_file_vide()
2   val = defiler(F)
3   while  not est_vide (F) and val != 'Prioritaire'
4         enfiler (V, val)
5         val = defiler(F)
```

On considère la fonction **longueur_file**, écrite en Python, ci-dessous. Le but de cette fonction est de renvoyer le nombre d'éléments d'une file donnée en paramètre. À la fin du programme, la file **F** doit avoir retrouvé son état d'origine.

2.b) Compléter le programme ci-dessous.

```
1     def longueur_file(F) :
2           V= creer_file_vide()
3           n= 0
4           while not est_vide(F) :
5              n = …
6              val = defiler(F)
7              enfiler(V,val)
8           while not est_vide(V) :
9              …
10             …
11          return n
```

2.c) Écrire une fonction **compter_prio** qui prend en paramètre une file **F** . Cette fonction renvoie le nombre de personnes prioritaires dans la file d'attente, à l'instant t. La file **F** doit être identique à celle du départ en fin d'éxécution de la fonction.

Exercice 2 : piles, programmation Python

La notation polonaise inverse (NPI) permet d'écrire des expressions de calculs numériques sans utiliser de parenthèse. Cette manière de présenter les calculs a été utilisée dans des calculatrices de bureau dès la fin des années 1960. La NPI est une forme d'écriture d'expressions algébriques qui se distingue par la position relative que prennent les nombres et leurs opérations.

Par exemple :

Notation classique	Notation NPI
3+9	3 9 +
8×(3+5)	8 3 5 + ×
(17+5)×4	17 5 + 4 ×

L'expression est lue et évaluée de la gauche vers la droite en mettant à jour une pile.

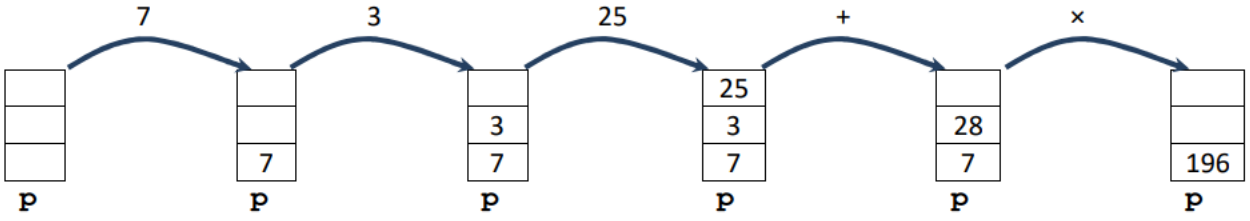
- Les nombres sont empilés dans l'ordre de la lecture.
- Dès la lecture d'un opérateur (+, -, ×, /), les deux nombres au sommet de la pile sont dépilés et remplacés par le résultat de l'opération effectuée avec ces deux nombres. Ce résultat est ensuite empilé au sommet de la pile.

A la fin de la lecture, la valeur au sommet est renvoyée.

Exemple : l'expression `7 3 25 + ×` qui correspond au calcul `7×(3+25)` s'évalue à 196 comme le montrent les états successifs de la pile créée, nommée `p` :

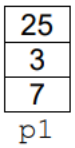
- On empile la valeur 7.
- On empile la valeur 3.
- On empile la valeur 25.
- On remplace les deux nombres du sommet de la pile (25 et 3) par leur somme 28.
- On remplace les deux nombres du sommet de la pile (28 et 7) par leur produit 196.

Schéma descriptif des différentes étapes d'exécution.



1. En vous inspirant de l'exemple ci-dessus, dessiner le schéma descriptif de ce que donne l'évaluation par la NPI de l'expression `12 4 5 × +` .

2. On dispose de la pile suivante nommée `p1` :



On rappelle ci-dessous les primitives de la structure de pile (LIFO : Last In First out) :

Fonction	Description
`pile_vide()`	Créé et renvoie une nouvelle pile vide
`empiler(p, e)`	Place l'élément **`e`** au sommet de la pile **`p`**.
`depiler(p)`	Supprime et renvoie l'élément se trouvant au sommet de **`p`**.
`est_vide(p)`	Renvoie un booléen indiquant si **`p`** est vide ou non.

On dispose aussi de la fonction suivante, qui prend en paramètre une pile `p` :

```
def top(p) :
    x = depiler(p)
    empiler(p, x)
    return x
```

On exécute la ligne suivante `temp = top(p1)` :

a. Quelle valeur contient la variable `temp` après cette exécution ?

b. Représenter la pile `p1` après cette exécution.

3. En utilisant uniquement les 4 primitives d'une pile, écrire en langage Python la fonction `addition(p)` qui prend en paramètre une pile p d'au moins deux éléments et qui remplace les deux nombres du sommet d'une pile p par leur somme. Remarque : cette fonction ne renvoie rien, mais la pile p est modifiée.

4. On considère que l'on dispose également d'une fonction `multiplication(p)` qui remplace les deux nombres du sommet d'une pile p par leur produit (on ne demande pas d'écrire cette fonction). Recopier et compléter, en n'utilisant que les primitives d'une pile et les deux fonctions `addition` et `multiplication`, la suite d'instructions (ci-dessous) qui réalise le calcul (3+5)×7 dont l'écriture en NPI est : 3 5 + 7 ×

```
p=pile_vide()
empiler(p,3)
```