

**Exercice 2 (Algorithmique, structures de données et gestion de processus - 6 points)**

On cherche à créer une application de type *liste de tâches* à faire pour aider Alice à planifier sa journée. Pour cela Alice saisit les informations concernant chacune des tâches qu'elle doit effectuer : elle indique un nom pour la tâche, ainsi que la durée qu'elle estime nécessaire afin de la réaliser. On représente une tâche saisie par Alice à l'aide d'un objet de type `Tache`, muni de quatre attributs :

- \* le `numero` de la tâche, saisi par Alice ;
- \* le `nom` de la tâche, saisi par Alice ;
- \* la `duree` (un entier exprimé en minute) nécessaire à la réalisation de la tâche saisie par Alice ;
- \* la `duree_restante` (un entier exprimé en minute) avant la fin de la tâche. Cet attribut sera initialisé avec la durée totale nécessaire à la réalisation de la tâche.

Avancer de  $n$  minutes ( $n$  entier positif) dans une tâche consiste à diminuer de  $n$  la durée restante de cette tâche. Une tâche est terminée si la durée restante est négative ou nulle.

Lors de la phase de planification de ses tâches (aucune d'entre elles n'est commencée), Alice liste les tâches suivantes qui doivent être effectuées :

Numéro	Nom	Durée	Durée restante
1	Répondre aux e-mails	45	45
2	Ranger ma chambre	60	60
3	Réviser la NSI	90	90
4	S'entraîner aux échecs	30	30
5	Apprendre le vocabulaire de chinois	30	30
6	Lire Fondation	60	60
7	Ecrire ma lettre au Père Noël	20	20

On dispose de la classe `Tache` ci-dessous pour représenter les tâches :

```

1 class Tache:
2     def __init__(self, numero, nom, duree):
3         self.numero = numero
4         self.nom = nom
5         self.duree_initiale = duree
6         self.duree_restante = duree
7
8     def __repr__(self):
9         return '<t'+str(self.numero)+'>'

```

1. Donner le code Python qui permet d'instancier deux variables `tache1` et `tache2` représentant les tâches suivantes :

- \* tâche numéro 1 : Répondre aux e-mails. Durée estimée : 45 minutes.
- \* tâche numéro 2 : Ranger ma chambre. Durée estimée : 60 minutes.

On supposera dans la suite que les variables `tache1`, `tache2`, ..., `tache7` représentent les tâches établies par Alice lors de la phase de planification.

La méthode `__repr__` renvoie une représentation de l'instance sous forme d'une chaîne de caractères. La fonction `print` utilise cette méthode. Ainsi on a :

```
>>> print(tache1)
<t1>
```

2. Recopier et compléter le code de la méthode `avancer` de la classe `Tache` qui permet d'avancer la tâche `self` de  $n$  minutes.

```

1 def avancer(self, n):
2     ...

```

3. Recopier et compléter le code de la méthode `est_terminee` de la classe `Tache` qui renvoie `True` si la tâche est terminée, ou `False` sinon.

```

1 def est_terminee(self) :
2     ...

```

Afin d'aider Alice à planifier sa journée, on lui propose d'associer à chacune des tâches une priorité. La priorité d'une tâche est représentée par un entier de la manière suivante : 1 est la priorité minimale et, plus le nombre est grand, plus la tâche associée est prioritaire.

Pour stocker toutes les tâches à effectuer, on utilise une file, dans laquelle les éléments sont des tuples `(tache, priorite)`. Les éléments stockés dans la file doivent respecter les deux conditions ci-après.

- \* Condition 1 : les éléments sont rangés par ordre décroissant de priorité. L'élément de priorité maximale se trouve au début de la file, l'élément le moins prioritaire se trouve à la fin de la file.
- \* Condition 2 : parmi les éléments de même priorité, les éléments sont rangés dans l'ordre dans lequel ils ont été insérés dans la file. Ainsi, le premier élément de priorité  $p$  inséré se trouve devant les éléments de même priorité  $p$  insérés plus tard.

Par exemple, si la file de tâches `f` est la file :

```
[debut] (<t3>, 4) (<t1>, 3) (<t2>, 3) (<t4>, 1) (<t5>, 1) [fin]
```

cela signifie que :

- \* la tâche de priorité maximale est la tâche numéro 3;
- \* les deux tâches à exécuter en priorité après la tâche numéro 3 sont les tâches numéro 1 et numéro 2. La tâche numéro 1 a été ajoutée à la file des tâches à traiter avant la tâche numéro 2;
- \* il n'y a pas de tâche de priorité 2;
- \* les tâches les moins prioritaires de la file sont les tâches numéro 4 et numéro 5. La tâche numéro 4 a été ajoutée avant la tâche numéro 5.

4. Représenter l'état de la file `f` lorsqu'on lui ajoute successivement la tâche numéro 6 avec la priorité 2, puis la tâche numéro 7 avec la priorité 4 en respectant les conditions 1 et 2 décrites ci-dessus.

On suppose déjà définies les méthodes suivantes pour la classe `File` :

- \* `File()` : crée et renvoie un objet de type `File`, vide.
- \* `enfiler(self, e)` : ajoute l'élément `e` à la fin de la file `f`.
- \* `defiler(self)` : renvoie, en le supprimant de la file, le premier élément de la file si cela est possible.
- \* `examiner(self)` : renvoie, sans le supprimer de la file, le premier élément de la file si cela est possible.
- \* `est_vide(self)` : renvoie `True` si la file est vide, ou `False` sinon.

5. En repartant de la file `f` suivante :

```
[debut] (<t3>, 4) (<t1>, 3) (<t2>, 3) (<t4>, 1) (<t5>, 1) [fin]
```

donner la valeur de `f.defiler()[0]`, et représenter le contenu de la file `f` après l'exécution de cette instruction.

6. En repartant de la file `f` suivante :

```
[debut] (<t3>, 4) (<t1>, 3) (<t2>, 3) (<t4>, 1) (<t5>, 1) [fin]
```

donner la valeur de `f.examiner()[1]`, et représenter le contenu de la file `f` après l'exécution de cette instruction.

On souhaite écrire une fonction `ajouter_file_prio` qui prend en paramètres :

- \* une file `f` dont les éléments sont des tuples `(tache, priorite)` respectant les deux conditions de l'énoncé;
- \* une tâche `t`;
- \* la priorité `p` de la tâche `t`;

et qui ajoute le tuple `(t, p)` à la bonne position dans la file `f`.

On utilise une file auxiliaire `f_aux` que l'on remplit en défilant les éléments en début de file `f` tant que la priorité du premier élément de la file est supérieure ou égale à `p`. Puis on enfiler l'élément `(t, p)` dans la file auxiliaire. On défile ensuite tous les éléments restants de `f` dans `f_aux` et enfin on enfiler dans `f` tous les éléments de `f_aux`.

7. Recopier et compléter le code de la fonction `ajouter_file_prio`.

```

1 def ajouter_file_prio(f, t, p):
2     f_aux = File()
3     while ...:
4         ...
5         ...enfiler(...)
6     while not ...:
7         ...
8     while not ...:
9         ...

```

8. Donner le coût d'exécution temporel dans le pire des cas de la fonction `ajouter_file_prio`, en fonction du nombre  $m$  d'éléments de la file  $f$ .

Une fois qu'Alice a entré les tâches qu'elle doit effectuer, leur durée estimée, ainsi que la priorité à laquelle elle doit les effectuer, l'application lui propose un planning en utilisant la technique dite Pomodoro :

- \* la tâche à effectuer est la tâche qui se trouve en tête de file;
- \* on défile cette tâche de la file des tâches à effectuer;
- \* on avance cette tâche de 25 minutes;
- \* si cette tâche n'est pas terminée, on rajoute cette tâche dans la file des tâches à effectuer, avec la même priorité qu'initialement (en utilisant la fonction `ajouter_file_prio`);
- \* si cette tâche se termine au cours des 25 minutes, alors Alice attend la fin des 25 minutes en se reposant;
- \* on continue ces étapes tant que la file des tâches à effectuer n'est pas vide.

On rappelle les tâches à effectuer ci-dessous, classées par ordre de priorité. On considérera que les tâches sont ajoutées à la file de priorité dans l'ordre du tableau ci-dessous :

Numéro	Nom	Durée	Priorité
3	Réviser la NSI	90	4
7	Ecrire ma lettre au Père Noël	20	4
1	Répondre aux e-mails	45	3
2	Ranger ma chambre	60	3
6	Lire Fondation	60	2
4	S'entraîner aux échecs	30	1
5	Apprendre le vocabulaire de chinois	30	1

9. Indiquer pour chaque bloc de 25 minutes la tâche qui avance, en suivant le modèle proposé, jusqu'à la fin de toutes les tâches. On fera particulièrement attention au cas où la tâche n'est pas terminée : celle-ci est rajoutée à la file des tâches à effectuer (dont elle avait été supprimée) avec la même priorité qu'initialement, en respectant les conditions 1 et 2 de l'énoncé.

10. Ecrire le code d'une fonction `planning` qui prend en paramètre une file de priorité  $f$  dont les éléments sont des tuples  $(tache, prio)$ , et qui renvoie une liste de tâches, dans l'ordre dans lequel elles vont être effectuées par tranche de 25 minutes avec la méthode Pomodoro. Par exemple, si `tache1`, `tache2` et `tache3` sont les tâches numéro 1, numéro 2 et numéro 3, alors le programme suivant :

```

1 file = File()
2 for t, p in [(tache1, 3), (tache2, 3), (tache3, 4)]:
3     ajouter_file_prio(file, t, p)
4 print(planning(file))

```

produit l'affichage :

[<t3>, <t3>, <t3>, <t3>, <t1>, <t2>, <t1>, <t2>, <t2>]