TP 3: Classe `FileDePriorite`

L'objectif de ce TP est de développer une classe `FileDePriorite` du module `file_de_priorite.py`. Une fonction `cle`, initialisée par défaut à l'identité, sera utilisée pour comparer les éléments de cette file. On considère qu'un élément $e1$ est prioritaire sur un autre élément $e2$ si $cle(e1) < cle(e2)$. Si $e1 == e2$ et que $e1$ a été enfilé avant $e2$ alors $e1$ sortira avant $e2$.

Cette classe proposera les méthodes suivantes :

— `__init__(self, elements: Sequence[T] =(), cle: Callable[[T], Any]=lambda e:e)` qui permet d'initialiser la file avec des éléments ;
— `est_vide(self) -> bool` qui permet de savoir si la file est vide ;
— `enfiler(self, element: T)` qui permet d'enfiler un élément dans la file. La complexité en temps dans le pire des cas doit être au plus en n (où $log_2(n)$), mais pas en $n \times log_2(n)$.
— `defiler(self) -> T` qui permet de défiler l'élément le plus prioritaire, on obtient alors cet élément. La complexité en temps est en $O(1)$;
— `element(self) -> T` qui permet d'obtenir l'élément le plus prioritaire sans le défiler. La complexité en temps est en $O(1)$.

De plus :

— on peut parcourir tous les éléments de la file du plus prioritaire au moins prioritaire ;
— la représentation informelle commencera par `"-> "` suivi des éléments de la file de priorité (à la fin celui sera prêt à être défilé) séparé par une virgule suivi d'un espace et finira par `"->"`

Voici quelques exemples d'utilisation de cette classe :

```
In [1]: from file_de_priorite import FileDePriorite

In [2]: f=FileDePriorite((3,9,1,4))

In [3]: f.est_vide()
Out[3]: False

In [4]: f.element()
Out[4]: 1

In [5]: f.enfiler(2)

In [6]: print(f.defiler())
1

In [7]: f.element()
Out[7]: 2

In [8]: for e in f: print(e)
2
3
```

```
4
9

In [9]: print(f)
-> 9, 4, 3, 2 ->
```