# Spécialité NSI en terminale
# Exercices 3

Dans cet exercice, on s'interdit l'usage de la fonction **min** préprogrammée en Python et permettant d'obtenir directement le minimum d'une liste donnée en argument. En revanche, on se donne la fonction `mini` suivante, écrite en Python.

```
1  def mini(t):
2      '''Calcule le minimum d'un tableau d'entiers ou de flottants.'''
3      if len(t) == 0:
4          return None
5      p = t[0]
6      for i in range(len(t)):
7          if t[i] <= p:
8              p = t[i]
9      return p
```

**Q1.** Expliquer le déroulement pas à pas, (en précisant l'évolution de la valeur des variables) lors de l'appel `mini([6,2,15,2,15])`, puis donner la valeur renvoyée.

**Q2.** Prouver que lorsque `t` est une liste non vide d'entiers ou de flottants, `mini(t)` renvoie la valeur minimale des éléments de `t`. Utiliser un invariant de boucle précis.

**Q3.** Evaluer le coût en temps de l'appel `mini(t)`, exprimé en fonction du nombre $n$ d'éléments de la liste `t`.

**Q4.** Proposer une modification de la fonction `mini` pour que la valeur renvoyée soit le maximum et non le minimum. On pourra utiliser la numérotation des lignes pour préciser le lieu d'éventuelles modifications et ainsi éviter de réécrire toute la fonction.

On souhaite récupérer non plus le minimum d'une liste mais la (une) position dans le tableau où le minimum est atteint. Dans l'exemple vu plus haut, il y a deux positions où ce minimum est atteint : 1 et 3.

**Q5.** Expliquer le principe d'une fonction réalisant cette opération et en particulier le rôle des variables manipulées.

**Q6.** *Cette question ne sera lue que si la question précédente a été traitée.*
Ecrire une fonction `position_mini` réalisant effectivement cette opération.

**Q7.** Préciser l'indice renvoyé si le minimum est présent plusieurs fois dans la liste. Proposer une modification permettant de changer ce comportement.

On souhaite maintenant déterminer la valeur minimale d'un tableau bidimensionnel d'entiers/de flottants.
Un appel de cette fonction pourrait être :

```
>>> mini2D([[10, 3, 15], [5, 13, 10]])
3
```

**Q8.** Expliquer le principe d'une fonction réalisant ce travail.

**Q9.** Programmer effectivement cette fonction (on supposera les listes internes de taille non nulle).

**Q10.** Evaluer la complexité temporelle de cette fonction.

On souhaite, partant d'une liste constituée de listes à deux éléments [chaîne, entier], déterminer la/une chaîne pour laquelle l'entier/le flottant associé est minimal :

```
>>> chaine_mini([['Tokyo', 7000], ['Paris', 6000], ['Londres', 8000]])
'Paris'
```

**Q11.** Ecrire une fonction `chaine_mini` réalisant effectivement cette opération.

**Q12.** Ecrire enfin une fonction `majores_par` prenant en entrée une liste `t` d'entiers/de flottants ainsi qu'un entier/flottant `seuil` et renvoyant le nombre d'éléments de `t` majorés au sens strict par `seuil` :

```
>>> majores_par([12, -5, 10, 9], 10)
2
```

Précision : un nombre $x$ est majoré au sens strict pas un nombre $s$ si $x < s$.