# BACCALAUREAT

## SESSION 2022

---

## Épreuve de l'enseignement de spécialité

# NUMERIQUE et SCIENCES INFORMATIQUES

# Partie pratique

## Classe Terminale de la voie générale

---

## Sujet n°24

---

DUREE DE L'EPREUVE : 1 heure

**Le sujet comporte 2 pages numérotées de 1 / 3 à 3 / 3**
**Dès que le sujet vous est remis, assurez-vous qu'il est complet.**

*Le candidat doit traiter les 2 exercices.*

## EXERCICE 1 (4 points)

Écrire la fonction `maxliste`, prenant en paramètre un tableau non vide de nombres `tab` (type `list`) et renvoyant le plus grand élément de ce tableau.

Exemples :

```
 >>> maxliste([98, 12, 104, 23, 131, 9])
131
>>> maxliste([-27, 24, -3, 15])
24
```

## EXERCICE 2 (4 points)

On dispose de chaînes de caractères contenant uniquement des parenthèses ouvrantes et fermantes.

Un parenthésage est correct si :

- le nombre de parenthèses ouvrantes de la chaîne est égal au nombre de parenthèses fermantes.
- en parcourant la chaîne de gauche à droite, le nombre de parenthèses déjà ouvertes doit être, à tout moment, supérieur ou égal au nombre de parenthèses déjà fermées.

Ainsi, `"((()())(()))"` est un parenthésage correct.
Les parenthésages `"())(()"` et `"(())(()"` sont, eux, incorrects.

On dispose du code de la classe `Pile` suivant :

```
class Pile:
    """Classe définissant une pile"""
    def __init__(self, valeurs=[]):
        self.valeurs = valeurs

    def est_vide(self):
        """Renvoie True si la pile est vide, False sinon"""
        return self.valeurs == []

    def empiler(self, c):
        """Place l'élément c au sommet de la pile"""
        self.valeurs.append(c)

    def depiler(self):
        """
        Supprime l'élément placé au sommet de la pile, à condition qu'elle
        soit non vide
        """
        if self.est_vide() == False:
```

```
            self.valeurs.pop()
```

On souhaite programmer une fonction `parenthesage` qui prend en paramètre une chaîne `ch` de parenthèses et renvoie `True` si la chaîne est bien parenthésée et `False` sinon.
Cette fonction utilise une pile et suit le principe suivant : en parcourant la chaîne de gauche à droite, si on trouve une parenthèse ouvrante, on l'empile au sommet de la pile et si on trouve une parenthèse fermante, on dépile (si possible !) la parenthèse ouvrante stockée au sommet de la pile.
La chaîne est alors bien parenthésée si, à la fin du parcours, la pile est vide.
Elle est, par contre, mal parenthésée :
- si dans le parcours, on trouve une parenthèse fermante, alors que la pile est vide ;
- ou si, à la fin du parcours, la pile n'est pas vide.

```
def parenthesage (ch):
    """Renvoie True si la chaîne ch est bien parenthésée et False sinon"""
    p = Pile()
    for c in ch:
        if c == ...:
            p.empiler(c)
        elif c == ...:
            if p.est_vide():
                return ...
        else:
            ...
    return p.est_vide()

assert parenthesage("((()())(()))") == True
assert parenthesage("())(()") == False
assert parenthesage("(())(()") == False
```

Compléter le code de la fonction `parenthesage`.

```python
"""
Author: Pascal Padilla
Source: correction de l'exercice 1 du sujet 24 des épreuves pratiques NSI 2022
"""


from doctest import testmod


def maxliste(tab: list[int]) -> int:
    """Renvoie le plus grand élément d'un tableau.

    Args:
        tab (list[int]): tableau d'entiers relatifs

    Returns:
        int: valeur maximale du tableau

    Tests et Exemples:
    >>> maxliste([98, 12, 104, 23, 131, 9])
    131
    >>> maxliste([-27, 24, -3, 15])
    24
    """
    # BOUCLE qui va parcourir tout le tableau
    # à partir de la 2ème case
    # -> invariant:
    #   * valeur_max est la plus grande valeur de la
    #   portion tab[0..i-1] du tableau
    # -> initialisation:
    #   * valeur_max: première valeur du tableau
    #   * i: 1 pour commencer à la 2ème case
    valeur_max = tab[0]

    taille = len(tab)
    for i in range(1, taille):
        # maintient de l'invariant si la valeur
        # courante est plus grande que val_max
        if tab[i] > valeur_max:
            valeur_max = tab[i]

    return valeur_max


# Tests avec des affichages
print(maxliste([98, 12, 104, 23, 131, 9]))
print(maxliste([-27, 24, -3, 15]))


# Tests avec des assertions:
assert maxliste([98, 12, 104, 23, 131, 9]) == 131
assert maxliste([-27, 24, -3, 15]) == 24


# Tests avec doctest
testmod()
```

```
"""
Author: Pascal Padilla
Source: correction de l'exercice 2 du sujet 24 des épreuves pratiques NSI 2022

Remarques:
* ne JAMAIS mettre un tableau VIDE comme paramètre par défaut
d'une fonction.
Ici le bug est dans la définition de __init__ de la Pile:
    def __init__(self, valeurs = []):...
Ça fait des erreurs assez complexes à trouver...

Exemple d'erreur de fou :
>>> parenthesage("((()())(()))")
True
>>> parenthesage("((()())(()))")
False
;)
... et oui, c'est dingue !
"""


class Pile:
    """ Classe définissant une pile """
    # modification d'énoncé sinon c'est FAUX...
    # def __init__(self, valeurs=[]):
    #     self.valeurs = valeurs

    def __init__(self):     # suppression de la valeur par défaut
        self.valeurs = []   # initialisation avec une pile vide

    def est_vide(self):
        """Renvoie True si la pile est vide, False sinon"""
        return self.valeurs == []

    def empiler(self, c):
        """Place l¿élément c au sommet de la pile"""
        self.valeurs.append(c)

    def depiler(self):
        """Supprime l¿élément placé au sommet de la pile, à condition qu¿elle soit n
on vide"""
        if self.est_vide() == False:
            self.valeurs.pop()


def parenthesage (ch):
    """Renvoie True si la chaîne ch est bien parenthésée et False sinon"""
    # création d'une pile vide
    p = Pile()

    # parcours de la chaîne, caractère par caractère
    for c in ch:
        # empiler une parenthèse ouvrante
        if c == '(':
            p.empiler(c)

        # cas des parenthèses fermantes
        elif c == ')':
            # mauvais parenthésage avec une fermante alors
            # que la pile est vide
            if p.est_vide():
                return False

            # dépiler la parenthèse ouvrante au sommet de la pile
            else:
                p.depiler()

    # parenthésage correct <=> pile vide à la fin
    return p.est_vide()


assert parenthesage("((()())(()))") == True
assert parenthesage("())(()") == False
```

```
assert parenthesage("(()))(()") == False
```