BACCALAUREAT

SESSION 2022

Épreuve de l'enseignement de spécialité

NUMERIQUE et SCIENCES INFORMATIQUES

Partie pratique

Classe Terminale de la voie générale

Sujet n°10

DUREE DE L'EPREUVE : 1 heure

Le sujet comporte 3 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)

L'occurrence d'un caractère dans un phrase est le nombre de fois où ce caractère est présent.

Exemples :

- l'occurrence du caractère 'o' dans 'bonjour' est 2 ;
- l'occurrence du caractère 'b' dans 'Bébé' est 1 ;
- l'occurrence du caractère 'B' dans 'Bébé' est 1 ;
- l'occurrence du caractère ' ' dans 'Hello world !' est 2.

On cherche les occurrences des caractères dans une phrase. On souhaite stocker ces occurrences dans un dictionnaire dont les clefs seraient les caractères de la phrase et les valeurs l'occurrence de ces caractères.

Par exemple : avec la phrase 'Hello world !' le dictionnaire est le suivant :

```
{'H': 1,'e': 1,'l': 3,'o': 2,' ': 2,'w': 1,'r': 1,'d': 1,'!': 1}
```

(l'ordre des clefs n'ayant pas d'importance).

Écrire une fonction `occurence_lettres` avec prenant comme paramètre une variable `phrase` de type `str`. Cette fonction doit renvoyer un dictionnaire de type constitué des occurrences des caractères présents dans la phrase.

EXERCICE 2 (4 points)

La fonction `fusion` prend deux listes `L1,L2` d'entiers triées par ordre croissant et les fusionne en une liste triée `L12` qu'elle renvoie.

Le code Python de la fonction est

```
def fusion(L1,L2):
    n1 = len(L1)
    n2 = len(L2)
    L12 = [0]*(n1+n2)
    i1 = 0
    i2 = 0
    i = 0
    while i1 < n1 and ... :
        if L1[i1] < L2[i2]:
            L12[i] = ...
            i1 = ...
        else:
            L12[i] = L2[i2]
```

```
            i2 = ...
        i += 1
    while i1 < n1:
      L12[i] = ...
      i1 = i1 + 1
      i = ...
    while i2 < n2:
      L12[i] = ...
      i2 = i2 + 1
      i = ...
    return L12
```

Compléter le code.

Exemple :

```
>>> fusion([1,6,10],[0,7,8,9])
[0, 1, 6, 7, 8, 9, 10]
```

```python
from doctest import testmod


def occurrence_lettres(phrase: str) -> dict:
    """Occurrence de chaque lettre qui compose la phrase.

    Args:
        phrase (str): phrase à analyser

    Returns:
        dict: dictionnaire dont
            * les clés sont les lettres et
            * les valeurs leurs occurrences

    Tests et Exemples:
    >>> occurrence_lettres('Hello world !')
    {'H': 1, 'e': 1, 'l': 3, 'o': 2, ' ': 2, 'w': 1, 'r': 1, 'd': 1, '!': 1}
    """
    # initialisation du dictionnaire à renvoyer
    occurrences = {}

    # parcours de la phrase, lettre par lettre
    for lettre in phrase:
        if lettre in occurrences:
            # la lettre est déjà apparue donc
            # ajoute 1 à l'occurrence de la lettre
            occurrences[lettre] += 1
        else:
            # première apparition de la lettre donc
            # initialise son occurrence à 1
            occurrences[lettre] = 1

    return occurrences


assert occurrence_lettres('Hello world !') == {'H': 1, 'e': 1, 'l': 3, 'o': 2, ' ':
2, 'w': 1, 'r': 1, 'd': 1, '!': 1}
testmod()
```

```python
from doctest import testmod


def fusion(L1: list, L2: list) -> list:
    """Fusionne deux listes triées en une seule triée.

    Args:
        L1 (list): tableau trié de nombres entiers
        L2 (list): idem

    Returns:
        list: fusion des deux tableaus en argument
            en un nouveau tableau trié

    Tests et Exemples:
    >>> fusion([1,6,10],[0,7,8,9])
    [0, 1, 6, 7, 8, 9, 10]
    """
    n1 = len(L1)
    n2 = len(L2)

    # initialisation du tableau fusion
    # de taille n1 + n2
    L12 = [0]*(n1+n2)

    # initialisation des compteurs de L1 et de L2
    i1 = 0
    i2 = 0

    # initialisation du compteur de L12
    i = 0

    # première boucle qui rempli L12
    # tant que l'on n'a pas parcouru entièrement
    # l'un des deux tableaux
    while i1 < n1 and i2 < n2 :
        if L1[i1] < L2[i2]:
            # Ajoute la plus petite valeur dans L12
            # qui est celle de L1
            L12[i] = L1[i1]
            # met à jour le compteur de L1 pour traiter
            # sa valeur suivante
            i1 = i1 + 1
        else:
            # Ajoute la plus petite valeur dans L12
            # qui est celle de L2
            L12[i] = L2[i2]
            # met à jour le compteur de L2 pour traiter
            # sa valeur suivante
            i2 = i2 + 1
        i += 1

    # deuxième boucle qui finit de remplir L12 avec les
    # valeurs ordonnées de L1
    # cette boucle ne s'exécute pas si L1 est déjà
    # complètement parcouru (i1 == n1)
    while i1 < n1:
        L12[i] = L1[i1]
        i1 = i1 + 1
        i = i + 1

    # deuxième boucle qui finit de remplir L12 avec les
    # valeurs ordonnées de L2
    # cette boucle ne s'exécute pas si L2 est déjà
    # complètement parcouru (i2 == n2)
    while i2 < n2:
        L12[i] = L2[i2]
        i2 = i2 + 1
        i = i + 1

    # L12 contient désormais toutes les valeurs de L1
    # et de L2, ordonnées.
    # L'intérêt de cet algorithme est que son temps
```

```
    # d'exécution dans le pire des cas est proportionnel
    # à n1 + n2. C'est une complexité linéaire
    return L12


assert fusion([1,6,10],[0,7,8,9]) == [0, 1, 6, 7, 8, 9, 10]
testmod()
```