BACCALAUREAT

SESSION 2022

Épreuve de l'enseignement de spécialité

NUMERIQUE et SCIENCES INFORMATIQUES

Partie pratique

Classe Terminale de la voie générale

Sujet n°23

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)

Sur le réseau social TipTop, on s'intéresse au nombre de « like » des abonnés. Les données sont stockées dans des dictionnaires où les clés sont les pseudos et les valeurs correspondantes sont les nombres de « like » comme ci-dessous :

```
{'Bob': 102, 'Ada': 201, 'Alice': 103, 'Tim': 50}
```

Écrire une fonction `max_dico` qui :

- Prend en paramètre un dictionnaire `dico` non vide dont les clés sont des chaînes de caractères et les valeurs associées sont des entiers ;
- Renvoie un tuple dont :
  - La première valeur est la clé du dictionnaire associée à la valeur maximale ;
  - La seconde valeur est la première valeur maximale présente dans le dictionnaire.

Exemples :

```
>>> max_dico({'Bob': 102, 'Ada': 201, 'Alice': 103, 'Tim': 50})
('Ada', 201)
```

```
>>> max_dico({'Alan': 222, 'Ada': 201, 'Eve': 220, 'Tim': 50})
('Alan', 222)
```

EXERCICE 2 (4 points)

Nous avons l'habitude de noter les expressions arithmétiques avec des parenthèses comme par exemple : $(2 + 3) \times 5$.

Il existe une autre notation utilisée par certaines calculatrices, appelée notation postfixe, qui n'utilise pas de parenthèses. L'expression arithmétique précédente est alors obtenue en saisissant successivement 2, puis 3, puis l'opérateur +, puis 5, et enfin l'opérateur $\times$. On modélise cette saisie par le tableau `[2, 3, '+', 5, '*']`.

Autre exemple, la notation postfixe de $3 \times 2 + 5$ est modélisée par le tableau :

```
[3, 2, '*', 5, '+'].
```

D'une manière plus générale, la valeur associée à une expression arithmétique en notation postfixe est déterminée à l'aide d'une pile en parcourant l'expression arithmétique de gauche à droite de la façon suivante :

- Si l'élément parcouru est un nombre, on le place au sommet de la pile ;
- Si l'élément parcouru est un opérateur, on récupère les deux éléments situés au sommet de la pile et on leur applique l'opérateur. On place alors le résultat au sommet de la pile.
- A la fin du parcours, il reste alors un seul élément dans la pile qui est le résultat de l'expression arithmétique.

Dans le cadre de cet exercice, on se limitera aux opérations $\times$ et $+$.

Pour cet exercice, on dispose d'une classe `Pile` qui implémente les méthodes de base sur la structure de pile.

Compléter le script de la fonction `eval_expression` qui reçoit en paramètre une liste python représentant la notation postfixe d'une expression arithmétique et qui renvoie sa valeur associée.

Exemple :

```
>>> eval_expression([2, 3, '+', 5, '*'])
25
```

```
class Pile:
    """Classe définissant une structure de pile."""
    def __init__(self):
        self.contenu = []

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

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

    def depiler(self):
        """
        Retire et renvoie l'élément placé au sommet de la pile,
        si la pile n'est pas vide.
        """
        if not self.est_vide():
            return self.contenu.pop()


def eval_expression(tab):
    p = Pile()
    for ... in tab:
        if element != '+' ... element != '*':
            p.empiler(...)
        else:
            if element == ...:
                resultat = p.depiler() + ...
            else:
                resultat = ...
            p.empiler(...)
    return ...
```

```python
"""
Author: Pascal Padilla
Source: correction de l'exercice 1 du sujet 23 des épreuves pratiques NSI 2022
"""


from doctest import testmod


def max_dico(dico: dict) -> tuple[str, int]:
    """_summary_

    Args:
        dico (dict): _description_

    Returns:
        tuple[str, int]: _description_

    Tests et exemples:
    >>> max_dico({'Bob': 102, 'Ada': 201, 'Alice': 103, 'Tim': 50})
    ('Ada', 201)
    >>> max_dico({'Alan': 222, 'Ada': 201, 'Eve': 220, 'Tim': 50})
    ('Alan', 222)
    """
    # initialisation de valeur maximale et clé associée
    # avec des valeurs impossible
    # (un nombre négatif et un None)
    val_max = -1
    cle_max = None

    # parcours de tout le dictionnaire par couple clé/valeur
    for cle, valeur in dico.items():
        # une valeur maximale est trouvée
        if valeur > val_max:
            # mise à jour de la clé max et de sa valeur max associée
            val_max = valeur
            cle_max = cle

    # renvoie du couple clé_max et valeur_max
    return (cle_max, val_max)


# tests avec des affichages
print(max_dico({'Bob': 102, 'Ada': 201, 'Alice': 103, 'Tim': 50}))
print(max_dico({'Alan': 222, 'Ada': 201, 'Eve': 220, 'Tim': 50}))


# tests avec des assertions
assert max_dico({'Bob': 102, 'Ada': 201, 'Alice': 103, 'Tim': 50}) == ('Ada', 201)
assert max_dico({'Alan': 222, 'Ada': 201, 'Eve': 220, 'Tim': 50}) == ('Alan', 222)


# tests avec doctest
testmod()

''.join([chr(i) for i in range(10)])
```

```
"""
Author: Pascal Padilla
Source: correction de l'exercice 2 du sujet 23 des épreuves pratiques NSI 2022

Remarques:
    * classe Pile classique
"""


class Pile:
    """Classe définissant une structure de pile."""
    def __init__(self):
        self.contenu = []

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

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

    def depiler(self):
        """
        Retire et renvoie l¿élément placé au sommet de la pile,
        si la pile n¿est pas vide.
        """
        if not self.est_vide():
            return self.contenu.pop()


def eval_expression(tab):
    # création d'une pile vide
    p = Pile()

    # parcours du tableau avec 'element'
    for element in tab:
        # cas d'un nombre -> empiler au sommet de la pile
        # l'élément n'est ni un '+' ET ni un '*'
        if element != '+' and element != '*':
            p.empiler(element)

        # cas d'un opérateur
        else:
            # addition car il y a un '+' dans le résultat
            if element == '+':
                # calcul du résultat
                resultat = p.depiler() + p.depiler()
            else:
                # cas de la multiplication
                # calcul du résultat
                resultat = p.depiler() * p.depiler()

            # empiler le résultat
            p.empiler(resultat)

    # fin BOUCLE
    # tout le tableau est parcouru
    # le résultat est au sommet de la pile
    return p.depiler()


# tests avec des affichages
print(eval_expression([2, 3, '+', 5, '*']))


# tests avec des assertions
assert eval_expression([2, 3, '+', 5, '*']) == 25
```