BACCALAURÉAT

SESSION 2024

Épreuve de l'enseignement de spécialité

NUMÉRIQUE et SCIENCES INFORMATIQUES

Partie pratique

Classe Terminale de la voie générale

Sujet n°47

DURÉE DE L'ÉPREUVE : 1 heure

Le sujet comporte 4 pages numérotées de 1 / 4 à 4 / 4
Dès que le sujet vous est remis, assurez-vous qu'il est complet.

Le candidat doit traiter les 2 exercices.

EXERCICE 1 (10 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 ;
- et qui 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': 222, 'Tim': 50 })
('Alan', 222)
```

EXERCICE 2 (10 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.
- à 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.

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

    def est_vide(self):
        """Renvoie un booléen indiquant si la pile est vide."""
        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. Produit une erreur sinon.
        """
        assert 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 = ... + ...
            else:
                resultat = ...
            p.empiler(...)
    return ...
```

Exemples :

```
>>> eval_expression([2, 3, '+', 5, '*'])
25
>>> eval_expression([1, 2, '+', 3, '*'])
9
>>> eval_expression([1, 2, 3, '+', '*'])
5
```