BACCALAUREAT

SESSION 2022

Épreuve de l'enseignement de spécialité

NUMERIQUE et SCIENCES INFORMATIQUES

Partie pratique

Classe Terminale de la voie générale

Sujet n°13

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)

On s'intéresse au problème du rendu de monnaie. On suppose qu'on dispose d'un nombre infini de billets de 5 euros, de pièces de 2 euros et de pièces de 1 euro.

Le but est d'écrire une fonction nommée `rendu` dont le paramètre est un entier positif non nul `somme_a_rendre` et qui retourne une liste de trois entiers `n1`, `n2` et `n3` qui correspondent aux nombres de billets de 5 euros (`n1`) de pièces de 2 euros (`n2`) et de pièces de 1 euro (`n3`) à rendre afin que le total rendu soit égal à `somme_a_rendre`.

On utilisera un algorithme glouton : on commencera par rendre le nombre maximal de billets de 5 euros, puis celui des pièces de 2 euros et enfin celui des pièces de 1 euros.

Exemples :

```
>>> rendu(13)
[2,1,1]
>>> rendu(64)
[12,2,0]
>>> rendu(89)
[17,2,0]
```

EXERCICE 2 (4 points)

On veut écrire une classe pour gérer une file à l'aide d'une liste chaînée. On dispose d'une classe `Maillon` permettant la création d'un maillon de la chaîne, celui-ci étant constitué d'une valeur et d'une référence au maillon suivant de la chaîne :

```
class Maillon :
    def __init__(self,v) :
        self.valeur = v
        self.suivant = None
```

Compléter la classe File suivante où l'attribut `dernier_file` contient le maillon correspondant à l'élément arrivé en dernier dans la file :

```
class File :
    def __init__(self) :
        self.dernier_file = None

    def enfile(self,element) :
        nouveau_maillon = Maillon(...)
        nouveau_maillon.suivant = self.dernier_file
        self.dernier_file = ...

    def est_vide(self) :
```

```
        return self.dernier_file == None

    def affiche(self) :
        maillon = self.dernier_file
        while maillon != ... :
            print(maillon.valeur)
            maillon = ...

    def defile(self) :
        if not self.est_vide() :
            if self.dernier_file.suivant == None :
                resultat = self.dernier_file.valeur
                self.dernier_file = None
                return resultat
            maillon = ...
            while maillon.suivant.suivant != None :
                maillon = maillon.suivant
            resultat = ...
            maillon.suivant = None
            return resultat
        return None
```

On pourra tester le fonctionnement de la classe en utilisant les commandes suivantes dans la console Python :

```
>>> F = File()
>>> F.est_vide()
True
>>> F.enfile(2)
>>> F.affiche()
2
>>> F.est_vide()
False
>>> F.enfile(5)
>>> F.enfile(7)
>>> F.affiche()
7
5
2
>>> F.defile()
2
>>> F.defile()
5
>>> F.affiche()
7
```

```python
def rendu(somme_a_rendre):
    n1, n2, n3 = 0, 0, 0
    while somme_a_rendre >= 5:
        n1 += 1
        somme_a_rendre -= 5
    while somme_a_rendre >= 2:
        n2 += 1
        somme_a_rendre -= 2
    while somme_a_rendre >= 1:
        n3 += 1
        somme_a_rendre -= 1
    return [n1, n2, n3]


import unittest

class correction(unittest.TestCase):
    def test_13(self):
        self.assertEqual(rendu(13), [2,1,1])

    def test_64(self):
        self.assertEqual(rendu(64), [12,2,0])

    def test_89(self):
        self.assertEqual(rendu(89), [17,2,0])

    def test_alea(self):
        from random import randint
        n = randint(100, 10_000)
        n1, n2, n3 = rendu(n)
        self.assertEqual(n1*5 + n2*2 + n3, n)


if __name__ == '__main__':
    unittest.main()
```

```python
class Maillon:
    def __init__(self, v, s):    # modification énoncé
        self.valeur = v
        self.suivant = s         # modification énoncé

class File:
    def __init__(self):
        self.dernier_file = None

    def enfile(self, element):
        nouveau_maillon = Maillon(element , self.dernier_file)
        self.dernier_file = nouveau_maillon

    def est_vide(self):
        return self.dernier_file == None

    def affiche(self):
        maillon = self.dernier_file
        while maillon != None:
            print(maillon.valeur)
            maillon = maillon.suivant

    def defile(self):
        if not self.est_vide():
            if self.dernier_file.suivant == None:
                resultat = self.dernier_file.valeur
                self.dernier_file = None
                return resultat
            maillon = self.dernier_file
            while maillon.suivant.suivant != None:
                maillon = maillon.suivant
            resultat = maillon.suivant.valeur
            maillon.suivant = None
            return resultat
        return None


# def validation():
#     tests = [] # valide si tous les éléments sont True
#     F = File()

#     # test_1 : tester une file vide
#     tests.append(F.est_vide() == True)

#     F.enfile(2)

#     # print_1 : '2'
#     F.affiche()

#     # test_2 : file non vide
#     tests.append(F.est_vide() == False)

#     F.enfile(5)
#     F.enfile(7)

#     # print_2 : '7\n5\n2'
#     F.affiche()

#     # test_3 : defiler
#     tests.append(F.defile() == 2)
#     # test_4 : defiler
#     tests.append(F.defile() == 5)

#     # print_3 : affichage final -> [True, True, True, True]
#     print(tests)


# validation()


import unittest

class correction(unittest.TestCase):
```

```
    def test_creation(self):
        F = File()
        self.assertEqual(F.est_vide(), True)

    def test_enfile(self):
        F = File()
        F.enfile(2)
        self.assertEqual(F.est_vide(), False)

    def test_defile(self):
        F = File()
        F.enfile(2)
        F.enfile(5)
        F.enfile(7)
        self.assertEqual(F.defile(), 2)
        self.assertEqual(F.defile(), 5)
        self.assertEqual(F.defile(), 7)
        self.assertEqual(F.est_vide(), True)


    def test_alea(self):
        from random import randint
        n = 1_000
        tab = [randint(-1000, 1000) for _ in range(n)]
        F = File()
        for x in tab:
            F.enfile(x)
            self.assertEqual(F.est_vide(), False)
        for x in tab:
            self.assertEqual(x, F.defile())
        self.assertEqual(F.est_vide(), True)


if __name__ == '__main__':
    unittest.main()
```