BACCALAUREAT

SESSION 2022

Épreuve de l'enseignement de spécialité

NUMERIQUE et SCIENCES INFORMATIQUES

Partie pratique

Classe Terminale de la voie générale

Sujet n°36

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)

Programmer la fonction `recherche`, prenant en paramètre un tableau non vide `tab` (type `list`) d'entiers et un entier `n`, et qui renvoie l'indice de la **dernière** occurrence de l'élément cherché. Si l'élément n'est pas présent, la fonction renvoie la longueur du tableau.

Exemples :

```
>>> recherche([5, 3],1)
2
>>> recherche([2,4],2)
0
>>> recherche([2,3,5,2,4],2)
3
```

EXERCICE 2 (4 points)

On souhaite programmer une fonction donnant la distance la plus courte entre un point de départ et une liste de points. Les points sont tous à coordonnées entières.

Les points sont donnés sous la forme d'un tuple de deux entiers.

La liste des points à traiter est donc un tableau de tuples.

On rappelle que la distance entre deux points du plan de coordonnées (x ; y) et (x' ; y') est donnée par la formule :

$$d = \sqrt{(x - x')^2 + (y - y')^2}.$$

On importe pour cela la fonction racine carrée (`sqrt`) du module `math` de Python.

On dispose d'une fonction `distance` et d'une fonction `plus_courte_distance_`:

```
from math import sqrt   # import de la fonction racine carrée

def distance(point1, point2):
    """ Calcule et renvoie la distance entre deux points. """
    return sqrt((...)**2 + (...)**2)

assert distance((1, 0), (5, 3)) == 5.0, "erreur de calcul"

def plus_courte_distance(tab, depart):
```

```
    """ Renvoie le point du tableau tab se trouvant à la plus
    courte distance du point depart."""
    point = tab[0]
    min_dist = ...
    for i in range (1, ...):
        if distance(tab[i], depart)...:
            point = ...
            min_dist = ...
    return point

assert plus_courte_distance([(7, 9), (2, 5), (5, 2)], (0, 0)) ==
(2, 5), "erreur"
```

Recopier sous Python (sans les commentaires) ces deux fonctions puis compléter leur code et ajouter une ou des déclarations (`assert`) à la fonction `distance` permettant de vérifier la ou les préconditions.

```python
from doctest import testmod

def recherche(tab: list, n: int) -> int:
    """Indice de la dernière occurence de n
    dans tab, longueur de tab sinon

    Args:
        tab (list): tableau dans lequel chercher n
        n (int): valeur à chercher

    Returns:
        int: indice de la dernière occurence de n

    Exemples:
    >>> recherche([5, 3], 1)
    2
    >>> recherche([2, 4], 2)
    0
    >>> recherche([2, 3, 5, 2, 3], 2)
    3
    """
    indice = -1
    taille = len(tab)

    for i in range(taille):
        if tab[i] == n:
            indice = i

    if indice == -1:
        return taille
    else:
        return indice


if __name__ == '__main__':
    testmod(verbose=True)
```

```python
from math import sqrt   # import de la fonction racine carrée

def distance(point1, point2):
    """ Calcule et renvoie la distance entre deux points. """
    assert isinstance(point1, tuple)
    assert len(point1) == 2
    assert isinstance(point1[0], int)
    assert isinstance(point1[1], int)
    assert isinstance(point2, tuple)
    assert len(point2) == 2
    assert isinstance(point2[0], int)
    assert isinstance(point2[1], int)
    return sqrt((point1[0]-point2[0])**2 + (point1[1]-point2[1])**2)

assert distance((1, 0), (5, 3)) == 5.0, "erreur de calcul"

def plus_courte_distance(tab, depart):
    """ Renvoie le point du tableau tab se trouvant à la plus
    courte distance du point depart."""
    point = tab[0]
    min_dist = distance(point, depart)
    for i in range (1, len(tab)):
        if distance(tab[i], depart) < min_dist:
            point = tab[i]
            min_dist = distance(tab[i], depart)
    return point

assert plus_courte_distance([(7, 9), (2, 5), (5, 2)], (0, 0)) == (2, 5), "erreur"

import unittest
from random import randint

class validation(unittest.TestCase):
    def test_distance_assert_tuple1(self):
        a, b, c, d = [randint(-100, 100) for _ in range(4)]
        self.assertRaises( AssertionError, distance, [a, b], (c, d) )
        self.assertRaises( AssertionError, distance, (a, b), [c, d] )

    def test_distance_assert_couple(self):
        a, b, c, d, e = [randint(-100, 100) for _ in range(5)]
        self.assertRaises(AssertionError, distance, (a, b, c), (d, e))
        self.assertRaises(AssertionError, distance, (a, b), (c, d, e))

    def test_distance_assert_int(self):
        a = chr(randint(65, 90))
        n1, n2, n3 = [randint(-100, 100) for _ in range(3)]
        self.assertRaises(AssertionError, distance, (a, n1), (n2, n3))
        self.assertRaises(AssertionError, distance, (n1, a), (n2, n3))
        self.assertRaises(AssertionError, distance, (n1, n2), (a, n3))
        self.assertRaises(AssertionError, distance, (n1, n2), (n3, a))

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