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.

```
"""
Author: Pascal Padilla
Source: correction de l'exercice 1 du sujet 36 des épreuves pratiques NSI 2022

Remarques:
    * /!\ attention ici c'est la occurrence qui est recherchée !
"""


def recherche(tab, n):
    """ Recherche la valeur de la dernière occurrence
    de n dans le tableau tab.

    Args:
        tab (list): tableau non vide de nb entiers
        n (int) : nombre entier à rechercher

    Returns:
        int : indice de la dernière valeur recherchée ou
              -1 si absent du tableau

    Tests et Exemples:
    >>> recherche([5, 3], 1)
    2
    >>> recherche([2, 4], 2)
    0
    >>> recherche([2, 3, 5, 2, 4], 2)
    3
    """
    longueur = len(tab)

    # BOUCLE
    # -> invariant: i_n est l'indice de la dernière apparition de n
    # dans la zone tab[0 .. i-1] ou la longueur du tableau si n n'y est pas
    i_n = longueur

    # -> condition d'arrêt: après la boucle i <- n
    for i in range(longueur):
        if tab[i] == n:
            i_n = i

    # fin BOUCLE
    # tout le tableau est parcouru
    return i_n


# vérification avec des assertions
assert recherche([5, 3], 1) == 2
assert recherche([2, 4], 2) == 0
assert recherche([2, 3, 5, 2, 4], 2) == 3

# vérification avec des affichages
print(recherche([5, 3], 1))
print(recherche([2, 4], 2))
print(recherche([2, 3, 5, 2, 4], 2))

# vérification avec doctest
from doctest import testmod
testmod()
```

```
"""
Author: Pascal Padilla
Source: correction de l'exercice 2 du sujet 36 des épreuves pratiques NSI 2022

Remarques:
    * 8 assertions ajoutées (dans la fonction `distance`)


"""


from math import sqrt   # import de la fonction racine carrée

def distance(point1, point2):
    """ Calcule et renvoie la distance entre deux points. """
    # ajout des assertions pour point1
    assert isinstance(point1, tuple)    # point1 est bien un tuple
    assert len(point1) == 2             #        de taille 2 (un couple)
    assert isinstance(point1[0], int)   # la première coordonnée de point1 est un nb
 entier
    assert isinstance(point1[1], int)   # la deuxième coordonnée de point1 est un nb
 entier

    # ajout des assertions pour point2
    assert isinstance(point2, tuple)    # idem
    assert len(point2) == 2             # idem
    assert isinstance(point2[0], int)   # idem
    assert isinstance(point2[1], int)   # idem

    # calcul puis renvoie de la distance
    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."""
    # BOUCLE
    # -> invariant: `point` est le point le plus proche de `depart`
    #               parmis tous les points de tab[0 .. i-1]
    # -> invariant: min_dist est la distance entre `depart` et `point`
    # -> initialisation: i <- 1
    # -> initialisation: point et min_dist invariants pour la zone
    #                    tab[0..0] du tableau
    point = tab[0]
    min_dist = distance(point, depart)
    # -> condition d'arrêt: après la boucle i <- dernier indice de `tab`
    for i in range (1, len(tab)):
        # maintient de l'invariant si le point courant est plus proche
        # de `depart` que `point`
        if distance(tab[i], depart) < min_dist:
            # mise à jour de `point` et de la distance minimale associée
            point = tab[i]
            min_dist = distance(tab[i], depart)

    # fin BOUCLE: parmis tous les points de tab, `point` est le plus proche
    return point

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