BACCALAUREAT

SESSION 2022

Épreuve de l'enseignement de spécialité

NUMERIQUE et SCIENCES INFORMATIQUES

Partie pratique

Classe Terminale de la voie générale

Sujet n°19

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 `multiplication` prenant en paramètres deux nombres entiers `n1` et `n2`, et qui renvoie le produit de ces deux nombres.

Les seules opérations autorisées sont l'addition et la soustraction.

Exemples :

```
>>> multiplication(3,5)
15
>>> multiplication(-4,-8)
32
>>> multiplication(-2,6)
-12
>>> multiplication(-2,0)
0
```

EXERCICE 2 (4 points)

Soit `T` un tableau non vide d'entiers triés dans l'ordre croissant et `n` un entier.

La fonction `chercher`, donnée à la page suivante, doit renvoyer un indice où la valeur `n` apparaît éventuellement dans `T`, et `None` sinon.

Les paramètres de la fonction sont :

- `T`, le tableau dans lequel s'effectue la recherche ;
- `n`, l'entier à chercher dans le tableau ;
- `i`, l'indice de début de la partie du tableau où s'effectue la recherche ;
- `j`, l'indice de fin de la partie du tableau où s'effectue la recherche.

La fonction `chercher` est une fonction récursive basée sur le principe « diviser pour régner ».

Le code de la fonction commence par vérifier si `0 <= i et j < len(T)`. Si cette condition n'est pas vérifiée, elle affiche `"Erreur"` puis renvoie `None`.

Recopier et compléter le code de la fonction `chercher` proposée ci-dessous :

```
def chercher(T,n,i,j):
    if i < 0 or ??? :
        print("Erreur")
        return None
    if i > j :
        return None
    m = (i+j) // ???
    if T[m] < ??? :
        return chercher(T, n, ??? , ???)
    elif ??? :
        return chercher(T, n, ??? , ??? )
    else :
        return ???
```

L'exécution du code doit donner :

```
>>> chercher([1,5,6,6,9,12],7,0,10)
Erreur
>>> chercher([1,5,6,6,9,12],7,0,5)
>>> chercher([1,5,6,6,9,12],9,0,5)
4
>>> chercher([1,5,6,6,9,12],6,0,5)
2
```

```
"""
Author: Pascal Padilla
Source: correction de l'exercice 1
    * du sujet 19 des épreuves pratiques NSI 2022
    * du sujet 21 des épreuves pratiques NSI 2022
"""


from doctest import testmod


# version itérative
def multiplication(n1: int, n2: int) -> int:
    """Renvoie le résultat de la multiplication de n1 par n2
    (avec n1 et n2 entiers relatifs).
    Petit défi : fonction programmée uniquement avec
                 les opérateurs + et -.

    Args:
        n1 (int): premier facteur
        n2 (int): deuxième facteur

    Returns:
        int: résultat de n1 × n2

    Tests et exemples:
    >>> multiplication(3,5)
    15
    >>> multiplication(-4,-8)
    32
    >>> multiplication(-2,6)
    -12
    >>> multiplication(-2,0)
    0
    """
    # résultat du produit initialisé à 0
    # car si aucune boucle ne s'exécute (n1 ¿ 0)
    # alors le résultat reste correct
    produit = 0

    # cas de n1 positif:
    #   faire n1 additions successives pour obtenir "n1 fois n2"
    if n1 >= 0:
        for i in range(n1):
            produit = produit + n2

    # cas de n1 négatif:
    #   faire n1 soustractions successives pour obtenir "n1 fois n2"
    else:
        for i in range(-n1):
            produit = produit - n2

    return produit


# version récursive (pour se faire plaisir ;) )
def multiplication_r(n1: int, n2: int) -> int:
    """Renvoie le résultat de la multiplication de n1 par n2
    (avec n1 et n2 entiers relatifs).
    Petit défi : fonction programmée uniquement avec
                 les opérateurs + et -.

    Args:
        n1 (int): premier facteur
        n2 (int): deuxième facteur

    Returns:
        int: résultat de n1 × n2

    Tests et exemples:
    >>> multiplication_r(3,5)
    15
    >>> multiplication_r(-4,-8)
```

```
    32
    >>> multiplication_r(-2,6)
    -12
    >>> multiplication_r(-2,0)
    0
    """
    # l'idée est de remarquer pour n1 POSITIF:
    # ¿ n1 × n2 = n2 + (n1 - 1) × n2
    # ce qui s'écrit de façon fonctionnelle:
    # ¿ multiplication(n1, n2) = n2 + multiplication(n1-1, n2)
    #
    # pour le cas n1 NÉGATIF:
    # ¿ n1 × n2 = (n1 + 1) × n2 - n2
    #
    # cette remarque permet une définition récursive de la multiplication
    #   ¿ avec n1 qui diminue de 1 à chaque appel récursif
    #   ¿ jusqu'à atteindre le cas de base 1

    # cas de bases (1) et (2):
    if n1 == 0 or n2 == 0:
        return 0

    # cas de base (3)
    if n1 == 1:
        return n2

    # cas positif:
    if n1 > 1:
        return n2 + multiplication(n1 - 1, n2)
    # cas négatif
    else:
        return multiplication(n1 + 1, n2) - n2


# tests avec des affichages
print(multiplication(3,5))
print(multiplication(-4,-8))
print(multiplication(-2,6))
print(multiplication(-2,0))


# tests de la fonction avec doctest
testmod()
```

```
"""
Author: Pascal Padilla
Source: correction de l'exercice 2 du sujet 19 des épreuves pratiques NSI 2022

Remarques :
    * quelques corrections d'espaces après les virgules
"""


def chercher(T, n, i, j):
    # cas de débordement de la zone de recherche
    # ¿ début de la zone (i) avant 0
    # ¿ fin de la zone (j) après le dernier élément
    if i < 0 or j >= len(T) :
        print("Erreur")
        return None

    # la zone de recherche est vide car i "croise" j
    if i > j :
        return None

    # indice médian de la zone de recherche
    m = (i + j) // 2

    # cas où la valeur médiane est plus petit que
    # la valeur recherchée n
    # et donc on va chercher dans la zone au dessus de médiane
    # (on met à jour le début de la zone de recherche i)
    if T[m] < n :
        return chercher(T, n, m + 1 , j)

    # cas de la valeur médiane plus grande que n
    # et donc on recherche dans la zone avant la médiane
    # et donc on met à jour la fin de la zone j
    elif n < T[m] :
        return chercher(T, n, i , m - 1)

    # la valeur médiane est égale à n : trouvé !
    # et donc on renvoie l'indice correspondant
    else :
        return m


# tests avec des assertions:
assert chercher([1,5,6,6,9,12],7,0,10) == None
assert chercher([1,5,6,6,9,12],7,0,5) == None
assert chercher([1,5,6,6,9,12],9,0,5) == 4
assert chercher([1,5,6,6,9,12],6,0,5) == 2


# tests avec des affichages:
print(chercher([1,5,6,6,9,12],7,0,10))
print(chercher([1,5,6,6,9,12],7,0,5))
print(chercher([1,5,6,6,9,12],9,0,5))
print(chercher([1,5,6,6,9,12],6,0,5))
```