1 TP 11 - Parcours séquentiel d'un tableau ter

TD n°11 : Parcours séquentiel d'un tableau ter	Thème 4 : Langages et Programmation
EXERCICES TYPE EPREUVE PRATIQUE en Terminale	EXERCICES

Exercice n°1 :

Enoncé

Compléter la fonction `prem_match` ci-dessous qui prend en paramètres deux tableaux `tab_a` et `tab_b` de même longueur et qui :

- renvoie le plus petit indice `i` pour lequel `tab_a[i] == tab_b[i]` si cet indice existe,
- renvoie `-1` sinon.

```
def match(tab_a, tab_b):
    pass
```

Solution

```
def match(tab_a, tab_b):
    for i in range(len(tab_a)):
```

```
        if tab_a[i] == tab_b[i] :
            return i
    return -1
```

Tester votre fonction grâce au jeu de tests ci-dessous.

```
assert match([77, 5, 8, 9, 3, 9], [77, 1, 9, 8, 5, 6]) == 0
assert match([1, 77, 8, 9, 3, 9], [2, 77, 9, 8, 5, 6]) == 1

assert match([3, 9, 7, 2, 5, 4], [33, 99, 77, 2, 5, 4]) == 3
assert match([3, 9, 7, 2, 5, 4], [33, 99, 77, 22, 55, 4]) == 5

assert match([1, 5, 8, 9, 3, 9, 7, 12, 5, 4], [5, 1, 9, 8, 5, 6, 3, 2, 1, 4


assert match([6, 7, 8, 9], [1, 2, 3, 4]) == -1
assert match([], []) == -1
```

Exercice n°2 : Tableaux et extremums : plus grand dénivelé

Enoncé

En randonnée cycliste, pédestre ou à skis, on différencie les dénivelés positif et négatif. Ici nous ne ferons pas la différence.

Par exemple avec ce tableau :

```
[7, 4, 3, 6, 7, 4, 3, 1, 8]
```

on rencontre huit dénivelés lors de son parcours -3, -1, 3, 1, -3, -1, -2, 7.

Plus formellement, dans un tableau `tab` de taille `n` le dénivelé à l'indice `i` est égal à `tab[i+1] - tab[i]` (à condition que l'indice `i+1` existe).

Voici le schéma d'un tableau `tab` de taille `n` :

```
-----------------------------------------------
| indices | 0 | 1 | 2 | 3 | ... | n-2 | n-1 |
-----------------------------------------------
```

```
| valeurs | . | . | . | . | ... |  .  |  .  |
-----------------------------------------------
```

Le plus grand indice possible est donc `n-1`.

Pour avoir le droit d'écrire `tab[i+1] - tab[i]` avec un tableau `tab` de taille `n`, quelle est alors la plus grande valeur de `i` possible : `n-2`, `n-1`, `n` ou `n+1` ?

Compléter la fonction `plus_grand_denivele` ci-dessous qui prend en paramètre un tableau `tab` **de plus de deux nombres** et renvoie le plus grand dénivelé qui existe dans ce tableau.

```
def plus_grand_denivele(tab):
    pass
```

Tester votre fonction en utilisant le jeu de tests ci-dessous.

```
assert plus_grand_denivele([2, 5, 8, 3, -2, 13, 17, 18, 16, 13,
assert plus_grand_denivele([17, 18, 16, 13, 11, 5, 2, -1, -14,
assert plus_grand_denivele([22, 16, 13, 11, 5, 2, -1, -14, -19]
assert plus_grand_denivele([22, 28]) == 6
```

Compléter la fonction `plus_petit_denivele` ci-dessous qui prend en paramètre un tableau `tab` **de plus de deux nombres** et renvoie le plus petit dénivelé qui existe dans ce tableau.

```
def plus_petit_denivele(tab):
    assert len(tab) >= 2, "Le tableau doit contenir au moins de
    pass
```

Tester votre fonction modifiée en utilisant le jeu de tests ci-dessous.

```
assert plus_petit_denivele([2, 5, 8, 3, -2, 13, 17, 18, 16, 13,
assert plus_petit_denivele([17, 18, 16, 13, 11, 5, 2, -1, -14,
assert plus_petit_denivele([5, 28, 54, 103, 187, 218]) == 23
assert plus_petit_denivele([22, 28]) == 6
```

Solution

```
def plus_grand_denivele(tab):
    denivele_max = tab[1] - tab[0]

    for i in range(1, len(tab)-1) :
        if tab[i+1] - tab[i] > denivele_max :
            denivele_max = tab[i+1] - tab[i]

    return denivele_max
```

```
def plus_petit_denivele(tab):
    assert len(tab) >= 2, "Le tableau doit contenir au moins de
    denivele_min = tab[1] - tab[0]

    for i in range(1, len(tab)-1) :
        if tab[i+1] - tab[i] < denivele_min :
            denivele_min = tab[i+1] - tab[i]

    return denivele_min
```

Exercice n°3 : tableaux et accumulation : dénivelés

Enoncé

En randonnée cycliste, pédestre ou à skis, on différencie les dénivelés positif et négatif cumulés : - le dénivelé positif cumulé correspond au «nombre de mètres en hauteur» qui ont été montés du début à la fin du parcours, - le dénivelé négatif cumulé correspond au «nombre de mètres en hauteur» qui ont été descendus du début à la fin du parcours.

On peut faire la même chose avec un tableau de nombres. Par exemple avec ce tableau :

```
[7, 4, 3, 6, 7, 4, 3, 1, 8]
```

on rencontre trois dénivelés positifs lors de son parcours : - +3 pour passer de la valeur 3 à la valeur 6, - +1 pour passer de la valeur 6 à la

valeur 7, - +7 pour passer de la valeur 1 à la valeur 8;

soit un dénivelé positif cumulé de +11.

Et le dénivelé négatif cumulé vaut quant à lui : -3-1-3-1-2 = -10.

Plus formellement, dans un tableau `tab` de taille `n` le dénivelé à l'indice `i` est égal à `tab[i+1] - tab[i]` (à condition que l'indice `i+1` existe).

Remarque : on considérera que dans un tableau vide ou de taille 1, le dénivelé cumulé est égal à zéro.

Voici le schéma d'un tableau `tab` de taille `n` :

```
-------------------------------------------
| indices | 0 | 1 | 2 | 3 | ... | n-2 | n-1 |
-------------------------------------------
| valeurs | . | . | . | . | ... |  .  |  .  |
-------------------------------------------
```

Le plus grand indice possible est donc `n-1`.

Pour avoir le droit d'écrire `tab[i+1] - tab[i]` avec un tableau `tab` de taille `n`, quelle est alors la plus grande valeur de `i` possible : `n-2`, `n-1`, `n` ou `n+1` ?

Compléter la fonction `compter_deniveles` ci-dessous qui prend en paramètre un tableau `tab` de nombres et renvoie un 2-uplet des dénivelés positif cumulé et négatif cumulé.

Le parcours de `tab` sera fait par indice.

```
def compter_deniveles(tab):
    cumul_plus = ...
    cumul_moins = ...

    pass
```

Tester votre fonction grâce au jeu de tests ci-dessous.

```
tab = [1, 2, 3, 4, 5]
assert compter_deniveles(tab) == (4, 0)

tab = [5, 4, 3, 2, 1]
assert compter_deniveles(tab) == (0, -4)

tab = [0, 10, 0, 10]
assert compter_deniveles(tab) == (20, -10)

tab = [0, 10, 0, 10, 10, 10, 10, 0]
assert compter_deniveles(tab) == (20, -20)

tab = [3, 7, 8, 9, 1, 0, 7, 8, 9, 3, 7, 8, 9, 4, 6, 1, 0, 0, 9,
assert compter_deniveles(tab) == (66, -60)

tab = []
assert compter_deniveles(tab) == (0, 0)

tab = [5]
assert compter_deniveles(tab) == (0, 0)
```

Solution

```
def compter_deniveles(tab):
    cumul_plus = 0
    cumul_moins = 0

    for i in range(len(tab)-1):
        denivele = tab[i+1] - tab[i]
        if denivele > 0 :
            cumul_plus = cumul_plus + denivele
        else:
            cumul_moins = cumul_moins + denivele

    return cumul_plus, cumul_moins
```

Exercice n°4 : tableaux et accumulation : moyenne coefficientée

Enoncé

On dispose ici de deux tableaux de même longueur : un tableau de notes (sur 20) et un tableau de coefficients. Il s'agit de calculer la moyenne

coefficientée correspondante. Par exemple avec :

```
notes  = [12, 15, 14, 18]
coeffs = [1,  2,  1,  4 ]
```

la moyenne sera calculée ainsi :

```
(12*1 + 15*2 + 14*1 + 18*4)/(1 + 2 + 1 + 4)
```

Dans la fonction ci-dessous on appelle `num` (pour *num*érateur) ce qui correspond à `(12*1 + 15*2 + 14*1 + 18*4)` et `denom` (pour *dénom*inateur) ce qui correspond à `(1 + 2 + 1 + 4)`.

Compléter la fonction `moyenne_coefficientee` ci-dessous qui prend en paramètre deux tableaux non vides `notes` et `coeffs` et renvoie la moyenne coefficientée correspondante.

```
def moyenne_coefficientee(notes, coeffs):
    pass
```

Tester votre fonction grâce au jeu de tests ci-dessous.

```
notes = [10, 10, 20]
coeffs = [1, 3, 1]
assert moyenne_coefficientee(notes, coeffs) == 12.0

notes = [15, 17, 13, 19, 15, 11]
coeffs = [2, 3, 5, 5, 3, 8]
assert moyenne_coefficientee(notes, coeffs) == 14.3846153846153

notes = [8, 8, 8, 8, 12]
coeffs = [1, 1, 1, 1, 4]
assert moyenne_coefficientee(notes, coeffs) == 10.0
```

Solution

```
def moyenne_coefficientee(notes, coeffs):
    num = 0
    denom = 0

    for i in range(len(notes)) :
        num = num + coeffs[i] * notes[i]
        denom = denom + coeffs[i]

    moyenne = num / denom

    return moyenne
```

Exercice n°3 : tableaux et accumulation : produit des valeurs

Enoncé

Il s'agit ici de faire le produit (la multiplication) de tous les nombres présents dans le tableau.

Compléter la fonction `produit` ci-dessous qui prend en paramètre un tableau `tab` et renvoie le produit de tous les nombres présents dans le tableau `tab`.

Par convention, si le tableau est vide on considérera que le produit est égal à 1.

On réfléchira à la valeur initiale de la variable `produit`.

```
def produit(tab):
    pass
```

Tester votre fonction grâce au jeu de tests ci-dessous.

```
tab = [2, 3, 2]
assert produit(tab) == 12

tab = [1, 2, 3, 4, 5, 6]
assert produit(tab) == 720

tab = [1, 1, 1, 1, 1, 1]
assert produit(tab) == 1
```

```
tab = [1, 14, 32, 0, 15, 6]
assert produit(tab) == 0

tab = []
assert produit(tab) == 1

tab = [7]
assert produit(tab) == 7

tab = [12, 11, 3, 21, 5, 41, 4, 6, 4, 7]
assert produit(tab) == 1145612160
```

Solution

```
def produit(tab):
    prod = 1

    for elt in tab :
        prod = prod * elt

    return prod
```

Exercice n°6 :

Enoncé

Écrire une fonction Python smul à deux paramètres, un nombre et une liste de nombres, qui multiplie chaque élément de la liste par le nombre et renvoie une nouvelle liste :

```
>>> smul(2, [1, 2, 3])
[2, 4, 6]
```

```

```

Solution

```
def smul(n,tab):
    nv_tab=[]
    for elt in tab:
        nv_tab.append(n*elt)
    return nv_tab
```

Exercice n°7 :

Enoncé

Écrire une fonction Python vsom qui prend en paramètre deux listes de nombres de même longueur et qui renvoie une nouvelle liste constituée de la somme terme à terme de ces deux listes :

```
>>> vsom([1, 2, 3], [4, 5, 6])
[5, 7, 9]
```

```

```

Solution

```
def vsom(tab1,tab2):
    nv_tab=[]
    for indice in range(len(tab1)):
        nv_tab.append(tab1[indice]+tab2[indice])
    return nv_tab
```

Exercice n°8 : tableaux et mutations : écrêtage

Enoncé

On dit qu'on écrête un signal lorsqu'on limite l'amplitude du signal entre deux valeurs `a` et `b`. On peut également appliquer cela à des tableaux de valeurs. Voici par exemple un tableau `tab` que l'on a écrêté entre -150 et 150 pour donner le tableau `tab_ec` :

```
tab =    [34, 56, 89, 134, 152, 250, 87, -34, -187, -310]
tab_ec = [34, 56, 89, 134, 150, 150, 87, -34, -150, -150]
```

L'écrêtage consiste à :
Soit un nombre entier `n` ainsi que deux entiers `a` et `b` avec `a <= b` :

- `n` si `n` est compris entre `a` et `b`,
- `a` si `n` est plus petit que `a`,
- `b` si `n` est plus grand que `b`.

Ecrire une fonction ecrete_v2 qui prend en paramètres un tableau d'entiers `tab` de longueur quelconque ainsi que deux entiers `a` et `b` avec `a <= b` et **renvoie un nouveau tableau `nv_tab`** correspondant à `tab` avec toutes ses valeurs écrêtées entre `a` et `b`

```
def ecrete(tab):
    pass
```

Tester votre fonction en utilisant le jeu de tests ci-dessous.

```
assert ecrete([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13], 5

assert ecrete([-13, -4, 6, 5, 8, -3, -12, -3, 0, 6, 7], -10, -5

assert ecrete([7, 8, 3, 9, 8, 7, 2, 4, 8, 9, 0, 1, 5, 8, 8, 8,

assert ecrete([], 0, 10) == []
```

Solution

```
def ecrete(tab, a, b):
    for i in range(len(tab)):
        if tab[i]<a:
            tab[i] = a
         elif tab[i]>b:
            tab[i] = b
    return tab
```