TD n°10 : Parcours séquentiel d'un tableau bis	Thème 4 : Langages et Programmation
EXERCICES TYPE EPREUVE PRATIQUE en Terminale	EXERCICES

TD n°10

Exercice n°1 : Tableaux et extremums - plus grand écart

On appelle ici plus grand écart d'un tableau de nombres, la plus grande différence que l'on peut trouver entre deux valeurs **pas forcément consécutives**. Par exemple avec ce tableau :

```
[7, 4, 3, 6, -2, 4, 3, 1, 8]
```

le plus grand écart est 10 qui correspond à la différence entre -2 et 8. Et avec ce tableau :

```
[7, 4, 3, 16, 8, 4, 3, 1, -3, -7, -20, 5, 7]
```

le plus grand écart est 36 qui correspond à l'écart entre -20 et 16.

Créer la fonction `plus_grand_ecart` ci-dessous qui prend en paramètre un tableau de nombres non vide `tab` et renvoie le plus grand écart de ce tableau.

In [1]:
```
def maxi(tab):
    if len(tab)==0:
        return None
    else:
        maximum=tab[0]
        for elt in tab:
            if elt>maximum:
                maximum=elt
        return maximum

def mini(tab):
    if len(tab)==0:
        return None
    else:
        minimum=tab[0]
        for elt in tab:
            if elt<minimum:
                minimum=elt
        return minimum


def plus_grand_ecart(tab):
    return maxi(tab)-mini(tab)
```

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

In [2]:
```
assert plus_grand_ecart([7, 4, 3, 6, -2, 4, 3, 1, 8])
assert plus_grand_ecart([7, 4, 3, 16, 8, 4, 3, 1, -3, -7,
```

```
assert plus_grand_ecart([6, 6, 6, 6, 6]) == 0
assert plus_grand_ecart([7]) == 0
```

Exercice n°2 : Tableaux et extremums : plus grande puissance

On dispose de tableaux **de taille paire** dont les valeurs correspondent à des mesures effectuées aux bornes d'une résistance. Plus précisément :

- les valeurs d'indices pairs correspondent à la tension `U`,
- les valeurs d'indices impairs correspondent à l'intensité `I`.

Par exemple avec ce tableau de six valeurs :

```
[19, 6, 23, 5, 20, 4]
```

on dispose de trois mesures :

- la première avec une tension de 19 V et une intensité de 6 A (soit une puissance de 19 * 6 = 114 Watts),

- la seconde avec une tension de 23 V et une intensité de 5 A (soit une puissance de 23 * 5 = 115 Watts),
- la troisième avec une tension de 20 V et une intensité de 4 A (soit une puissance de 20 * 4 = 80 Watts).

On cherche à déterminer la puissance maximale présente dans les tableaux de mesures, c'est à dire la plus grande valeur `tab[i] * tab[i+1]` avec l'indice `i` pair.

Compléter la fonction `plus_grand_produit` ci-dessous qui :

- prend en paramètre un tableau de nombres `mesures` de taille paire non nulle,
- qui renvoie le plus grand produit de la forme `mesures[i] * mesures[i+1]` avec l'indice `i` pair.

In [3]:
```
def plus_grand_produit(mesures):
    pmax = mesures[0] * mesures[1]
    for i in range(0, len(mesures), 2) :
        if mesures[i] * mesures[i+1] > pmax :
            pmax = mesures[i] * mesures[i+1]
    return pmax

plus_grand_produit([7, 4, 6, 5, 7, 5, 6, 4, 5, 5])
```

Out[3]: 35

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

In [4]:
```
assert plus_grand_produit([7, 4, 6, 5, 7, 5, 6, 4, 5,
assert plus_grand_produit([10, 3, 9, 3, 10, 4, 9, 5, 8, 5]
assert plus_grand_produit([6, 8, 6, 8, 12, 5]) == 60
assert plus_grand_produit([8, 8, 6, 8, 12, 5]) == 64
assert plus_grand_produit([2, 1]) == 2
```

Exercice n°3 : tableaux définis en compréhension : bonus sur les notes

On dispose d'un tableau de notes comprises entre 0 et 20 et on souhaite augmenter les notes de tout le monde de deux points (en ne dépassant pas 20).
Compléter la fonction `bonus` ci-dessous qui prend en paramètre `tab` un tableau de notes et **renvoie un nouveau tableau** `nv_tab` dont les éléments sont les notes augmentées de 2 points (sans toutefois dépasser 20).

In [5]:
```
def bonus(tab):
    nv_tab = []
    for note in tab:
        if note<18:
```

```
            nv_tab.append(note+2)
        else:
            nv_tab.append(20)
    return nv_tab

t=[12,8,5,18,19,20,14,12,11,9]

bonus(t)
```

Out[5]: [14, 10, 7, 20, 20, 20, 16, 14, 13, 11]

Exercice n°4 : tableaux et accumulation : produit des valeurs

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`.

In [6]:
```
def produit(tab):
    produit=1
```

```
    for elt in tab:
        produit*=elt
    return produit
```

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

In [7]:
```
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
```

Exercice n°5 : tableaux et accumulation : compter les sauts en hauteur

On dit que dans un tableau de nombres, il y a un «saut en hauteur» lorsqu'une valeur est

supérieure à la valeur précédente. Par exemple dans le tableau suivant :

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

il y a trois sauts en hauteur :

- entre les indices 2 et 3 (pour passer de la valeur 3 à la valeur 6),
- entre les indices 3 et 4 (pour passer de la valeur 6 à la valeur 7),
- et entre les indices 7 et 8 (pour passer de la valeur 1 à la valeur 8).

Plus formellement, dans un tableau `tab` de taille `n` on dit qu'il y a un saut en hauteur à l'indice `i` lorsque `tab[i] < tab[i+1]`.

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] < tab[i+1]` 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_sauts_en_hauteur` ci-dessous qui prend en paramètre un tableau `tab` de nombres et renvoie le nombre de sauts en hauteur présents dans le tableau `tab`.

Le parcours de `tab` sera fait par indice.

In [8]:
```
def compter_sauts_en_hauteur(tab):
    compteur = 0

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

    return compteur
```

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

In [9]:
```
tab = [7, 4, 3, 6, 7, 4, 3, 1, 8]
assert compter_sauts_en_hauteur(tab) == 3

tab = [7, 4, 3, 6, 7, 4, 3, 1, 8, 2, 5, 6, 5, 8, 1, 9, 0,
assert compter_sauts_en_hauteur(tab) == 14

tab = [1, 2, 3, 4, 5, 6, 7, 8, 9]
assert compter_sauts_en_hauteur(tab) == 8

tab = []
```

```
assert compter_sauts_en_hauteur(tab) == 0

tab = [1]
assert compter_sauts_en_hauteur(tab) == 0

tab = [7, 7, 7, 7]
assert compter_sauts_en_hauteur(tab) == 0
```

Exercice n°6 : tableaux et accumulation : compter les différences

On dispose ici de deux tableaux de même longueur. Il s'agit de calculer le nombre de différences entre les deux tableaux. Par exemple avec :

```
tab_1  = ['a', 'g', 'u', 'u', 'c', 'o', 'p', 'l'
tab_2  = ['a', 'g', 's', 'u', 'c', 'k', 'p', 'l'
```

il y a trois différences entre les deux tableaux : aux indices `2, 5` et `8`.

Compléter la fonction `compter_differences` ci-dessous qui prend en paramètre deux tableaux `tab_1` et `tab_2` de même longueur et renvoie

le nombre de différences entre les deux tableaux.

In [10]:
```
def compter_differences(tab_1, tab_2):
    compteur = 0

    for i in range(len(tab_1)) :
        if tab_1[i] != tab_2[i]:
            compteur = compteur + 1

    return compteur
```

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

In [11]:
```
tab_1  = ['a', 'g', 'u', 'u', 'c', 'o', 'p', 'l', 'm'
tab_2  = ['a', 'g', 's', 'u', 'c', 'k', 'p', 'l', 't',
assert compter_differences(tab_1, tab_2) == 3

tab_1  = ['a', 'g', 'u', 'u', 'c', 'o', 'p', 'l', 'm', 'v
tab_2  = ['a', 'g', 'u', 'u', 'c', 'o', 'p', 'l', 'm', 'v
assert compter_differences(tab_1, tab_2) == 0

tab_1  = ['a', 'g', 'u', 'u', 'c', 'o', 'p', 'l', 'm', 'v
tab_2  = ['g', 'u', 'u', 'c', 'o', 'p', 'l', 'm', 'v', 'p
assert compter_differences(tab_1, tab_2) == 10

tab_1  = ['a']
tab_2  = ['b']
assert compter_differences(tab_1, tab_2) == 1


tab_1  = []
tab_2  = []
assert compter_differences(tab_1, tab_2) == 0
```

Exercice n°7 : tableaux : frais

de gestion

Un site de petites annonces prend des frais sur les mises en vente.

- Si le prix net vendeur est inférieur ou égal à 15 euros, l'acheteur doit payer 0,50 euro de frais de gestion en plus.
- si le prix net vendeur est strictement supérieur à 15 euros, l'acheteur doit payer 1 euro de frais de gestion en plus.

Compléter la fonction `prix_acheteurs` ci-dessous qui prend en paramètre un tableau `pnv` de prix nets vendeur et **renvoie un nouveau tableau** correspondant aux prix payés par les acheteurs.

In [12]:
```
def prix_acheteurs(pnv):
    t=[]
    for elt in pnv:
        if elt<=15:
            t.append(elt+0.5)
        else:
            t.append(elt+1)
    return t
```

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

In [13]:
```
assert prix_acheteurs([14, 11, 19.99, 13.99, 7.50, 2,
```

```
assert prix_acheteurs([30, 110, 230, 350, 210]) == [31, 1
assert prix_acheteurs([3, 6, 1, 7, 9]) == [3.5, 6.5, 1.5,
assert prix_acheteurs([]) == []
assert prix_acheteurs([15]) == [15.5]
```

Exercice n°8 : Tableaux et extremums : solde maximal

On dispose d'un montant `m` de départ (par exemple `m = 17`) et d'un tableau `op` donnant les opérations bancaires successives réalisées sur un compte en banque, par exemple :

```
op = [150, -40, 18, -132, -7, -1, 29, 105]
```

qui correspond à un dépôt de 150 € puis à une dépense de 40 € puis à un dépôt de 18 € puis à trois dépenses de 132 €, 7 € et 1€ puis à deux dépôts de 29 € et 105 €.

Avec l'exemple précédent les soldes (le «solde» d'un compte bancaire est le montant présent sur le compte) successifs sont :

`17 --> 167 --> 127 --> 145 --> 13 --> 6 --> 5 --> 34 --> 139`

Ce qui correspond au tableau `tab_soldes` suivant :

```
[17, 167, 127, 145, 13, 6, 5, 34, 139]
```

On cherche, à partir de `m` et `op` à obtenir le solde maximal présent sur le compte (sur cet exemple 167 euros).

Compléter la fonction `solde_maximal` ci-dessous qui prend en paramètres un montant `m` de départ ainsi qu'un tableau `op` donnant les opérations bancaires successives et qui renvoie solde maximal présent sur le compte.

Remarque : Ici on procède comme pour un calcul de somme. On calcule le solde final du compte (variable `solde`) en vérifiant au passage si on dépasse la maximum en mémoire.

```
solde = m                                   --->
solde = solde + op[0]                       --->
solde = solde + op[1]                       --->
solde = solde + op[2]                       --->
solde = solde + op[3]                       --->
```

```
solde = solde + op[4]                                    --->
...
```

In [14]:
```python
def solde_maximal(m, op):

    solde = m
    solde_max = m

    for operation in op:
        solde = solde + operation
        if solde > solde_max :
            solde_max = solde

    return solde_max
```

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

In [15]:
```python
assert solde_maximal(17, [150, -40, 18, -132, -7, -1,
assert solde_maximal(1875, [150, -140, 148, -12, -75,
assert solde_maximal(19, [150, -140, 148, -105, 150, -140
assert solde_maximal(777, []) == 777
```

In [ ]:
```python

```