Exercices - Modularité - Correction

Pour chacun des 3 exercices de ce TD, vous devrez, sur papier, :

- Ecrire un module python
- Ecrire un programme principal permettant de tester votre module.
- ne pas oublier la docstring eds fonctions du odule python.

Exercice 1 Opérations mathématiques

1. Créez un module nommé `operations_mathematiques.py` en implémentant les fonctions suivantes :
   - Une fonction `addition(a, b)` qui prend deux nombres en entrée et renvoie leur somme.
   - Une fonction `soustraction(a, b)` qui prend deux nombres en entrée et renvoie leur différence.
   - Une fonction `multiplication(a, b)` qui prend deux nombres en entrée et renvoie leur produit.
   - Une fonction
   - division(a, b) qui prend deux nombres en entrée et renvoie leur quotient.
   - Une fonction `puissance(a, b)` qui prend deux nombres en entrée et renvoie a élevé à la puissance b.

```python
# operations_mathematiques.py
def addition(a, b):
    """
    Effectue l'operation d'addition de deux nombres.

    :param a: Premier nombre.
    :param b: Deuxieme nombre.
    :return: Resultat de l'addition.
    """
    return a + b

def soustraction(a, b):
    """
    Effectue l'operation de soustraction de deux nombres.

    :param a: Premier nombre (le minuend).
    :param b: Deuxieme nombre (le subtrahend).
    :return: Resultat de la soustraction.
    """
    return a - b

def multiplication(a, b):
    """
    Effectue l'operation de multiplication de deux nombres.

    :param a: Premier nombre.
    :param b: Deuxieme nombre.
    :return: Resultat de la multiplication.
    """
    return a * b

def division(a, b):
    """
    Effectue l'operation de division de deux nombres.

    :param a: Dividende.
    :param b: Diviseur (non nul).
    :return: Resultat de la division.
    """
    if b != 0:
        return a / b
    else:
        raise ValueError("Le diviseur ne peut pas etre zero.")

def puissance(a,b):
        return a**b
```

2. Créez un programme Python (par exemple, `test_operations_mathematiques.py`) dans lequel vous importez le module `operations_mathematiques` et testez les fonctions avec différents jeux de nombres. Assurez-vous de tester les cas d'addition, de soustraction, de multiplication, de division, et d'élévation à la puissance.

```
import operations_mathematiques

print(operations_mathematiques.addition(5,19))
print(operations_mathematiques.soustraction(5,19))
print(operations_mathematiques.multiplication(5,8))
print(operations_mathematiques.division(28,6))
print(operations_mathematiques.puissance(3,5))
print(operations_mathematiques.division(28,0))
```

Exercice 2 Probabilités

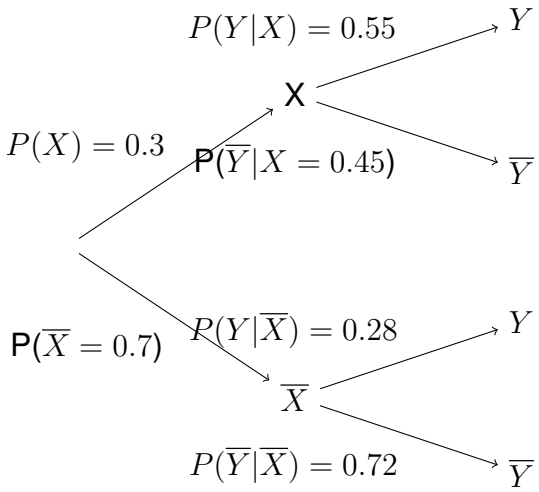
1. Créez un module nommé `probabilites.py` avec les spécifications suivantes :
   - Une fonction `probabilite_evenement(issues_favorables, issues_totales)` qui prend en entrée le nombre d'issues favorables de l'événement et le nombre total d'issue de cette expérience aléatoire que l'on considère équiprobable, et renvoie la probabilité de l'événement.
   - Une fonction `probabilite_A_inter_B(P_A, P_B_sachant_A)` qui prend en entrée les probabilités des événements A et $B|A$, et renvoie la probabilité de $p(A \cap B)$.
     On donne $P(B|A) = \dfrac{P(A \cap B)}{P(A)}$
   - Une fonction `probabilite_A_union_B(P_A, P_B_sachant_A, P_B_sachant_non_A)` qui prend en entrée les probabilités des événements A, $B|A$ et $B|\bar{A}$, et renvoie la probabilité de $p(A \cup B)$.
     On donne : $P(A \cup B) = P(A) + P(B) - P(A \cap B)$ et $P(B) = P(A)P(B|A) + P(\bar{A})P(B|\bar{A})$

```
#module probabilite

def probabilite_evenement(issues_favorables,issues_totales):
    assert issues_totales > 0, "Le nombre total d'issue doit etre > 0"
    return(issues_favorables/issues_totales)

def probabilite_A_inter_B(P_A,P_B_sachant_A):
    return P_A*P_B_sachant_A

def probabilite_A_union_B(P_A,P_B_sachant_non_A):
    return P_A + (1-P_A)*P_B_sachant_non_A
```

2. Application : Soit deux programmes informatiques x et y. On notera les événements suivants :
   - X : le programme x fonctionne.
   - $\bar{X}$: le programme x ne fonctionne pas.
   - Y : le programme y fonctionne.
   - $\bar{Y}$: le programme y ne fonctionne pas.

   Quelques données supplémentaires :
   - La probabilité que le programme x tombe en panne est de 0,3.
   - La probabilité que le programme y tombe en panne sachant que le programme x n'est pas tombé en panne est de 0,45.
   - La probabilité que le programme y ne tombe pas en panne sachant que le programme x est tombé en panne est de 0,28.

   (a) Représenter cette situation par un arbre de probabilité.



(b) Ecrire un programme principal en python qui calcule :

- la probabilité que les deux programmes ne tombent pas en panne. Quel est le résultat attendu ?

```
import probabilites

#probabilites de l'enonce
P_X = 0.3
P_nonY_sachant_X = 0.45
P_Y_sachant_nonX = 0.28

P_Y_sachant_X = 1 - P_nonY_sachant_X
P_X_inter_Y = probabilites.probabilite_A_inter_B(P_X,P_Y_sachant_X)
print(P_X_inter_Y)
#OUTPUT
0.165
```

- la probabilité que le programme x est en panne et le programme y n'est pas en panne. Quel est le résultat attendu ?

```
import probabilites

#probabilites de l'enonce
P_X = 0.3
P_nonY_sachant_X = 0.45
P_Y_sachant_nonX = 0.28

P_nonX= 1 - P_X
P_nonX_inter_Y = probabilites.probabilite_A_inter_B(P_nonX,
    P_Y_sachant_nonX)
print(P_nonX_inter_Y)
#OUTPUT
0.196
```

- la probabilité que le programme x est fonctionne ou que le programme y fonctionne. Quel est le résultat attendu ?

```
import probabilites

#probabilites de l'enonce
P_X = 0.3
P_nonY_sachant_X = 0.45
P_Y_sachant_nonX = 0.28

P_X_union_Y = probabilites.probabilite_A_union_B(P_X, P_Y_sachant_nonX)
print(P_X_union_Y)
#OUTPUT
0.496
```

- la probabilité que le programme y fonctionne. Quel est le résultat attendu ?

```
import probabilites

#probabilites de l'enonce
P_X = 0.3
P_nonY_sachant_X = 0.45
P_Y_sachant_nonX = 0.28

P_nonX= 1 - P_X
P_Y_sachant_X = 1 - P_nonY_sachant_X
P_Y =  probabilites.probabilite_A_inter_B(P_nonX,P_Y_sachant_nonX) +
    probabilites.probabilite_A_inter_B(P_X,P_Y_sachant_X)

print(P_Y)
#OUTPUT
0.361
```

Exercice 3

1. Créez un module nommé `gestion_taches.py` avec les spécifications suivantes :
   - Une fonction `ajouter_tache(tache, liste_taches)` qui prend en entrée une tâche et une liste de tâches, et renvoie la liste mise à jour avec la nouvelle tâche ajoutée.
   - Une fonction `supprimer_tache(tache, liste_taches)` qui prend en entrée une tâche et une liste de tâches, et renvoie la liste mise à jour avec la tâche supprimée.

```
#module gestion des taches

def ajouter_tache(tache,liste_tache):
    if tache not in liste_tache:
        liste_tache.append(tache)

def supprimer_tache(tache,liste_tache):
    if tache in liste_tache:
        liste_tache.remove(tache)

def afficher_taches(liste_taches):
    print(len(liste_taches)," tache(s) dans la liste")
    for t in liste_taches:
        print(t)
```

2. Programme de test : Créez un script Python (par exemple, `test_gestion_taches.py`) dans lequel vous importez le module `gestion_taches` et testez les fonctions avec différentes tâches

et listes de tâches. Assurez-vous de tester les cas d'ajout et de suppression. Utilisez un éditeur Python de votre choix pour exécuter le script de test. Vérifiez la sortie pour vous assurer que les fonctions fonctionnent correctement.

```
import gestion_taches

liste_tache = []

gestion_taches.ajouter_tache("preparer cours NSI",liste_tache)
gestion_taches.afficher_taches(liste_tache)
gestion_taches.ajouter_tache("preparer cours 3eme",liste_tache)
gestion_taches.ajouter_tache("preparer cours DNL 3eme",liste_tache)
gestion_taches.afficher_taches(liste_tache)
gestion_taches.ajouter_tache("preparer cours DNL 4eme",liste_tache)
gestion_taches.supprimer_tache("preparer cours NSI",liste_tache)
gestion_taches.afficher_taches(liste_tache)
gestion_taches.supprimer_tache("preparer cours DNL 4eme",liste_tache)
gestion_taches.supprimer_tache("preparer cours DNL 3eme",liste_tache)
gestion_taches.supprimer_tache("preparer cours 3eme",liste_tache)
gestion_taches.afficher_taches(liste_tache)
```