TP6 R1

Exercice 1 :

Programmer l'exercice 1 du TD 5. Rappel de l'énoncé :

Écrire une fonction qui calcule le nombre d'années (entières) qui s'écoulent entre deux dates (jour, mois, année), la première date étant antérieure à la seconde.

Écrire une fonction qui "qualifie" une personne en fonction de son âge selon les critères suivant (on suppose que l'âge de la retraite est 65 ans) :

- *« mineure »*
- *« majeure »*
- *« vingtenaire »*
- *« trentenaire »*
- *« quadragénaire »*
- *« quinquagénaire »*
- *« sexagénaire »*
- *« retraitée »*

Plusieurs critères peuvent être combinés.

Utiliser ces fonctions dans un programme qui affiche l'âge d'une personne et sa "qualité" étant donné sa date de naissance.

Exercice 2 :

Programmer la fonction de l'exercice 2 du TD 5. Rappel de l'énoncé :

Voici une fonction récursive permettant de calculer la somme des n premiers entiers :

```
fonction sommeR(n : entier) retourne entier
debut
      si n=1 alors
            retourne 1
      sinon
            retourne n+sommeR(n-1)
      finsi
fin sommeR
```

Réécrire cette fonction de manière itérative. Comparer les deux algorithmes.

Programmer ensuite la version itérative de ce calcul.

Vous écrirez également un programme permettant d'utiliser ces fonctions.
Dans votre jeu d'essai, vous devrez faire apparaître les limites éventuelles de l'approche récursive.

Exercice 3 :

On souhaite calculer récursivement a^n, avec n un entier positif ou nul. Utiliser « l'astuce » suivante :

a^n = si n est pair alors $a^{n/2} * a^{n/2}$
sinon $a * a^{(n-1)/2} a^{(n-1)/2}$

de telle sorte que $a^{n/2}$ (ou $a^{(n-1)/2}$) ne soit calculé qu'une seule fois.

Écrivez la fonction et un programme l'utilisant.

Exercice 1

programme:

```python
def ecart (j1:int,j2:int,m1:int,m2:int,an1:int,an2:int):
    """
    fonction permettant de calculer l'écart d'âge entre deux dates
    entrée : j1,m1,an1,j2,m2,an2 : int
    sortie : age : int
    """

    age:int
    exchange_day:int
    exchange_month:int
    exchange_year:int
    if (an2<an1) or (an1==an2 and m2<m1) or (an1==an2 and m1==m2 and j2<j1):
        exchange_day=j1
        j1=j2
        j2=exchange_day
        exchange_month=m1
        m1=m2
        m2=exchange_month
        exchange_year=an1
        an1=an2
        an2=exchange_year

    age = an2 - an1
    if((m2 < m1) or ((m2 == m1) and (j2 < j1))):
        age = age - 1

    return age
```

```python
def vérifier_date(jour:int,mois:int,annee:int):···

if __name__ == "__main__":
    print("Entrez la date de naissance de la première personne :")
    j1 = int(input("Jour : "))
    m1 = int(input("Mois : "))
    an1 = int(input("Année : "))
    while not vérifier_date(j1,m1,an1):
        print("Date incorrecte.")
        print("Entrez la date de naissance de la première personne :")
        j1 = int(input("Jour : "))
        m1 = int(input("Mois : "))
        an1 = int(input("Année : "))
    print("Entrez la date ou vous voulez connaître l'écart avec la première :")
    j2 = int(input("Jour : "))
    m2 = int(input("Mois : "))
    an2 = int(input("Année : "))
    while not vérifier_date(j2,m2,an2):
        print("Date incorrecte.")
        print("Entrez la date ou vous voulez connaître l'écart avec la première :")
        j2 = int(input("Jour : "))
        m2 = int(input("Mois : "))
        an2 = int(input("Année : "))
    print("L'écart d'âge entre les deux dates est de", ecart(j1,j2,m1,m2,an1,an2), "ans.")
```

Mon programme permet de calculer l'écart d'âge entre deux dates. Je commence par demander à l'utilisateur d'entrer la première date puis la deuxième, en les divisant en jour, mois et année.

D'abord, j'appelle une fonction que j'ai créée, nommée “ecart”, qui fait le calcul de la différence en années entre les deux dates fournies. Cette fonction tient compte non seulement des années, mais aussi des mois et des jours pour vérifier si la personne la plus jeune a déjà fêté son anniversaire cette année. Si ce n'est pas le cas, je soustrais une année à l'écart.

Ensuite, j'ai une fonction de vérification, “vérifier_date”, que j'utilise pour m'assurer que les dates fournies sont valides. Si une date est incorrecte, je demande à l'utilisateur de la ressaisir.

Enfin, dans la partie principale de mon programme, je guide l'utilisateur pour qu'il entre d'abord la première date, puis la deuxième. Après avoir vérifié que les deux dates sont correctes, j'affiche la différence d'âge en années grâce au résultat de la fonction `ecart`.

En résumé, mon programme me permet de calculer l'écart d'âge exact entre deux dates, en prenant en compte les anniversaires non encore passés dans l'année en cours.

jeux d’essais:

```
Entrez la date de naissance de la première personne :
Jour : 23
Mois : 455
Année : 78
Date incorrecte.
Entrez la date de naissance de la première personne :
Jour : 4
Mois : 5
Année : 4865
Entrez la date ou vous voulez connaître l'écart avec la première :
Jour : 45
Mois : 4
Année : 54
Date incorrecte.
Entrez la date ou vous voulez connaître l'écart avec la première :
Jour : 6
Mois : 4
Année : 7894
L'écart d'âge entre les deux dates est de 3028 ans.
```

```
Entrez la date de naissance de la première personne :
Jour : 3
Mois : 3
Année : 6584
Entrez la date ou vous voulez connaître l'écart avec la première :
Jour : 4
Mois : 4
Année : 1952
L'écart d'âge entre les deux dates est de 4631 ans.
```

Exercice 2

programme:

```python
def sommeR(n:int):
    """
    fonction permettant de calculer la somme des n premiers entiers
    entrée : n : int
    sortie : somme : int
    """
    if (n==1):
        return 1
    else:
        return n + sommeR(n-1)

if __name__ == "__main__":
    n = int(input("Entrez un entier n : "))
    while n < 0:
        print("Entrez un entier positif.")
        n = int(input("Entrez un entier n : "))
    print("La somme des",n,"premiers entiers est de",sommeR(n))
```

J'ai conçu ce programme Python pour calculer la somme des n premiers nombres entiers positifs.
Comment ça marche ?

- ★ Fonction sommer(n): C'est le cœur de mon programme. Elle prend un nombre entier n en entrée et renvoie la somme de tous les nombres de 1 jusqu'à n. J'ai utilisé la récursivité pour implémenter cette fonction :

- Si n est égal à 1, la fonction renvoie directement 1 (cas de base).
- Sinon, elle renvoie la somme de n et du résultat de l'appel récursif sur n-1.

★ Partie principale: C'est là que l'utilisateur interagit avec le programme.

- Je demande à l'utilisateur de saisir un nombre entier n.
- Si n est négatif, je redemande à l'utilisateur d'entrer un nombre positif, car ma fonction sommer est conçue pour les nombres positifs.
- Une fois un nombre positif saisi, j'appelle la fonction sommer avec n en argument et j'affiche le résultat à l'écran.

Pourquoi j'ai choisi la récursivité ? La récursivité est un concept élégant en programmation qui consiste à définir une fonction en termes d'elle-même. Dans ce cas, elle me permet d'exprimer de manière concise le calcul de la somme, en le décomposant en un problème plus petit à chaque appel récursif.
Exemple: Si tu entres 5, le programme va calculer :

★ sommeR(5) = 5 + sommeR(4)
★ sommeR(4) = 4 + sommeR(3)
★ ...
★ sommeR(1) = 1

Et ainsi de suite, jusqu'à ce que tous les appels récursifs soient résolus et que le résultat final soit obtenu.
En résumé, ce programme offre une solution simple et efficace pour calculer la somme des n premiers entiers positifs en utilisant la récursivité. Il est un bon exemple de l'utilisation de fonctions et de la logique conditionnelle en Python."

jeux d’essais:

```
Entrez un entier n : -36
Entrez un entier positif.
Entrez un entier n : 5
La somme des 5 premiers entiers est de 15
PS C:\Users\Moi\Desktop\autres\cours\2024-2025\s1\r1.01\tp6> & C:/Users/Moi/AppData/Local/Prog
1/tp6/ex2.py
Entrez un entier n : 1654
Traceback (most recent call last):
  File "c:\Users\Moi\Desktop\autres\cours\2024-2025\s1\r1.01\tp6\ex2.py", line 17, in <module>
    print("La somme des",n,"premiers entiers est de",sommeR(n))
  File "c:\Users\Moi\Desktop\autres\cours\2024-2025\s1\r1.01\tp6\ex2.py", line 10, in sommeR
    return n + sommeR(n-1)
  File "c:\Users\Moi\Desktop\autres\cours\2024-2025\s1\r1.01\tp6\ex2.py", line 10, in sommeR
    return n + sommeR(n-1)
  File "c:\Users\Moi\Desktop\autres\cours\2024-2025\s1\r1.01\tp6\ex2.py", line 10, in sommeR
    return n + sommeR(n-1)
  [Previous line repeated 995 more times]
  File "c:\Users\Moi\Desktop\autres\cours\2024-2025\s1\r1.01\tp6\ex2.py", line 7, in sommeR
    if (n==1):
RecursionError: maximum recursion depth exceeded in comparison
PS C:\Users\Moi\Desktop\autres\cours\2024-2025\s1\r1.01\tp6>
```

pas beaucoup de depth

Exercice 3

programme:

```
fonction récursivité_puissance_n(nombre:entier ,a:entier) → renvoi entier
début
      si (n=0) alors
            retourne 1
      sinon
            retourne a*récursivité_puissance_n(n-1,a)
      finsi

fin récursivité_puissance_n
```

```
fonction recurs_a_n(nombre:entier ,a:entier) → renvoi entier
début
      avec: res:entier
      si (a mod 2 =0) alors
            res←a^(n/2).a^(n/2)
            retourne res
      sinon
            res←a.a^((n−1)/2).a^(n−1/2)
            retourne res
fin recurs_a_n
```

```python
import time

def récursive_puissance_n(n: int, a: int) -> int:
    """
    Fonction permettant de calculer a^n
    entrée : n, a : int
    sortie : res : int
    """
    if n == 0:
        return 1
    else:
        return a * récursive_puissance_n(n - 1, a)

def recurs_a_n(a: int, n: int) -> int:
    """
    Fonction optimisée pour calculer a^n
    entrée : a, n : int
    sortie : res : int
    """
    if a % 2 == 0:
        res = pow(a, n // 2) * pow(a, n // 2)
        return res
    else:
        res = pow(a, (n - 1) // 2) * pow(a, (n - 1) // 2) * a
        return res
```

```python
if __name__ == "__main__":
    a = int(input("Entrez un entier a : "))
    n = int(input("Entrez un entier n : "))

    while n < 0:
        print("Entrez un entier positif.")
        n = int(input("Entrez un entier n : "))

    # Mesurer le temps pour récursive_puissance_n
    start_time = time.time()
    result1 = récursive_puissance_n(n, a)
    end_time = time.time()
    time_recursive = end_time - start_time

    # Mesurer le temps pour recurs_a_n
    start_time1 = time.time()
    result2 = recurs_a_n(a, n)
    end_time1 = time.time()
    time_recurs_a_n = end_time1 - start_time1

    print(f"La puissance de {a} à la puissance {n} est de {result1}")
    print(f"Temps d'exécution de récursive_puissance_n : {time_recursive:.6f} secondes")

    print(f"La puissance de {a} à la puissance {n} est de {result2}")
    print(f"Temps d'exécution de recurs_a_n : {time_recurs_a_n:.6f} secondes")
```

J'ai créé deux fonctions Python pour calculer la puissance d'un nombre a élevé à la puissance n.

Première fonction : récursive_puissance_n Cette fonction utilise une approche récursive simple pour calculer la puissance. Elle fonctionne en multipliant a par lui-même n fois. Bien que cette méthode soit facile à comprendre, elle peut devenir très lente pour de grandes valeurs de n en raison du grand nombre d'appels récursifs.

Deuxième fonction : recurs_a_n Cette fonction est une version optimisée de la première. Elle exploite une propriété mathématique de l'exponentiation :

- Si n est pair, alors a^n = (a^(n/2))^2.
- Si n est impair, alors a^n = a * (a^((n-1)/2))^2. En utilisant cette propriété, nous réduisons considérablement le nombre d'opérations à effectuer, ce qui améliore significativement les performances pour de grandes valeurs de n.

Partie principale du programme Dans la partie principale, je demande à l'utilisateur de saisir les valeurs de a et n. Je m'assure que n est positif. Ensuite, j'appelle les deux fonctions pour calculer la puissance et je mesure le temps d'exécution de chacune. Enfin, j'affiche les résultats, y compris les temps d'exécution, pour comparer les performances des deux méthodes.
Pourquoi utiliser deux fonctions ? J'ai créé deux fonctions pour illustrer la différence de performance entre une approche récursive naïve et une approche optimisée. Cela permet de visualiser l'impact des optimisations sur le temps d'exécution.
En résumé, ce programme compare deux méthodes pour calculer une puissance : une méthode récursive simple mais moins efficace et une méthode optimisée qui exploite une propriété mathématique. Les résultats montrent généralement que la méthode optimisée est beaucoup plus rapide, surtout pour de grandes valeurs de n.

jeux d’essais:

```
Entrez un entier a : 6
Entrez un entier n : 88
La puissance de 6 à la puissance 88 est de 3001304322877741810638664917749763569090004369058583134069827211100 16
Temps d'exécution de récursive_puissance_n : 0.001000 secondes
La puissance de 6 à la puissance 88 est de 3001304322877741810638664917749763569090004369058583134069827211100 16
Temps d'exécution de recurs_a_n : 0.000000 secondes
```