Calculer une série mathématique

Soit la série S définie par la formule suivante:

$$S(r, n) = \sum_{i=1}^{n} \frac{1}{(1+r)^i}$$
$$= \frac{1}{(1+r)^1} + \frac{1}{(1+r)^2} + \cdots + \frac{1}{(1+r)^{n-1}} + \frac{1}{(1+r)^n}$$

où $r \neq -1$ est une valeur réelle et $n \geq 1$ est une valeur entière.

Définissez une **fonction** Python qui calcule cette série. Nommez votre fonction `S` et faites en sorte qu'elle accepte en argument des valeurs pour r et n (dans cet ordre). Assurez-vous aussi de **toujours** retourner une valeur en **virgule flottante**. Par exemple, l'expression `S(3, 5)` doit retourner la valeur `0.3330078125`.

Contexte de l'exercice

```
1
```

Bravo!

Votre score est **100/100**.

ATTENTION: cette soumission a été effectuée **après** l'échéance, elle ne sera **pas** considérée.

Entrez votre solution dans la cellule ci-dessous

Notez bien que **seul** le contenu de cette cellule sera **évalué** par le correcteur automatique.

```python
def S(r, n):
    som = 0.
    for i in range(1, n + 1):
            som += (1/((1 + r)**i))
    return som
```

Initialiser une matrice

Définissez une fonction `matrice` qui accepte **deux** arguments ***m*** et ***n*** correspondant aux nombres de **lignes** et de **colonnes** d'une matrice. Votre fonction doit **retourner** une liste de ***m*** listes composées chacune de ***n*** zéro. Notez que $m \geq 1$ et $n \geq 1$.

Par exemple, l'expression `matrice(3, 4)` doit retourner la liste de listes suivante:

```
[[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]
```

Contrainte de réalisation: pour implanter votre solution, vous devez utiliser `for` soit sous forme d'énoncé, soit sous forme d'expression.

Finalement, notez que votre fonction ne doit faire **aucun** affichage, seulement retourner la liste de listes demandée!

Contexte de l'exercice

```
1
```

Bien que votre solution soit fonctionnelle, vous devriez tenter de la simplifier

- son nombre **d'énoncés** logiques est **3.7x** plus élevé qu'attendu (notez que *énoncé ≠ ligne*)
- assurez-vous de n'inclure **aucun** énoncé de **test** dans votre cellule de solution
- le cas échéant, déplacez-les dans la **cellule de test** prévue à cette fin

Votre score est **95/100**.

ATTENTION: cette soumission a été effectuée **après** l'échéance, elle ne sera **pas** considérée.

Entrez votre solution dans la cellule ci-dessous

Notez bien que **seul** le contenu de cette cellule sera **évalué** par le correcteur automatique.

```
def matrice(m, n):
    liste = []
    for n in range(n):
        liste.append(0)
    listef = []
    for i in range(m):
        listef.append(liste)
    return listef
```

Faites vos tests ici

Notez qu'il est **inutile** de copier votre solution dans cette cellule, car **toutes** les cellules partagent le même interpréteur python. À partir de cette cellule, faites **directement** appel aux éléments de votre solution afin de **bien** les tester.

```
matrice(3, 4)
#[[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]
```

Solution du professeur

Notez que cette solution n'est généralement **pas** unique.

```
def matrice(m, n):
    return [[0 for _ in range(n)] for _ in range(m)]
```

Afficher un damier de jetons

Définissez une **fonction** nommée `afficher_damier` qui accepte en entrée **trois** arguments:

1. le nombre de **lignes** du damier;
2. le nombre de **colonnes** du damier;
3. une liste de **tuples** (i, j, k).

où la **liste** de tuples représente des jetons actuellement présents sur le damier, avec (i, j) désignant respectivement les indices de **ligne** et de **colonne** d'un jeton, et k le symbole qui lui est associé. Votre fonction doit **retourner** une chaîne de caractères représentant le damier correspondant.

Par exemple, l'expression suivante:

```
afficher_damier(4, 10, [(0, 2, 'A'), (1, 1, 'B'), (2, 0, 'C')])
```

doit retourner la chaîne suivantes:

```
'..A.......\n.B........\nC.........\n..........\n'
```

qui une fois affichée avec `print` produirait le damier suivant de 4 lignes et 10 colonnes:

```
..A.......
.B........
C.........
..........
```

Le `.` désigne une case **vide** du damier. Vous pouvez suposer que le k des jetons est une chaîne d'**un seul** caractère.

Notez bien que votre fonction ne doit faire **aucun** affichage, seulement retourner la **chaîne** demandée. Également, faites en sorte que le **3e argument** de votre fonction ait la valeur **par défaut** d'une liste vide. Dans le cas particulier où il y aurait **plusieurs** jetons dans la même case, c'est le **dernier** de la liste qui doit être conservé.

Pour résoudre ce problème, on vous suggère les étapes suivantes:

1. Créer une matrice sous la forme d'une liste de listes.
2. Initialiser tous les éléments de votre matrice avec des `'.'`.
3. Remplacer dans votre matrice les éléments spécifiés par la liste de jetons.
4. Joindre (str.join) les éléments des lignes avec `''`.
5. Joindre les lignes jointes avec des `'\n'`.
6. Ajouter un `'\n'` pour la dernière ligne.
7. Retourner le résultat.

Contexte de l'exercice

```
1
```

Bravo!

Votre score est **100/100**.

ATTENTION: cette soumission a été effectuée **après** l'échéance, elle ne sera **pas** considérée.

Entrez votre solution dans la cellule ci-dessous

Notez bien que **seul** le contenu de cette cellule sera **évalué** par le correcteur automatique.

```python
def afficher_damier(m, n, jetons=[]):
    damier = []
    for _ in range(m):
        contenu_ligne = []
        for _ in range(n):
            contenu_ligne += '.'
        damier.append(contenu_ligne)
    for i, j, k in jetons:
        damier[i][j] = k
    #Pour montrer à quoi la liste du damier est supposée de ressembler
    print(damier)
    #On retourne la liste en joignant tout pour former une seule chaine de caractère.
    #C'est aussi là qu'on rajouterait les lignes de début et de fin.
    return '\n'.join(''.join(z for z in ligne) for ligne in damier)+'\n'

#tester notre fct
x = afficher_damier(4, 10, [(0, 2, 'A'), (1, 1, 'B'), (2, 0, 'C')])
print(x)
```

Solution du professeur

Notez que cette solution n'est généralement **pas** unique.

```python
def afficher_damier(m, n, jetons=[]):
    damier = [['.' for _ in range(n)] for _ in range(m)]
    for i, j, k in jetons:
        damier[i][j] = k
    return '\n'.join(''.join(k for k in ligne) for ligne in damier)+'\n'
x = afficher_damier(4, 10, [(0, 2, 'A'), (1, 1, 'B'), (2, 0, 'C')])
print(x)
```

Décoder un damier

Définissez une fonction nommée `decoder_damier` qui accepte en argument un damier représenté sous la forme d'une **chaîne** de caractères, et qui **retourne** un tuple (m, n, k), où m et n sont respectivement les nombres de **lignes** et de **colonnes** du damier, et k est un dictionnaire des jetons qui se trouvent sur ce damier. Un jeton est défini comme tout caractère différent de `'.'`, ce dernier symbolisant une case **vide** du damier. Le dictionnaire retourné doit contenir des associations `clé: valeur` où la clé est un **couple** (i, j) des indices de ligne et de colonne de la case du jeton dont le symbole est **valeur**.

Par exemple, l'expression suivante:

```
decoder_damier('..A.......\n.B........\nC.........\n..........\n')
```

doit retourner le tuple suivant:

```
(4, 10, {(0, 2): 'A', (1, 1): 'B', (2, 0): 'C'})
```

c'est-à-dire un damier de 4 lignes par 10 colonnes, avec des jetons A, B et C respectivement dans les cases $(0, 2)$, $(1, 1)$ et $(2, 0)$.

Notez bien que votre fonction ne doit faire **aucun** affichage, seulement retourner le **tuple** demandé. Vous pouvez supposer que la chaîne reçue en argument ne contient aucun espace blanc, que les lignes sont séparées par des `\n` et qu'elles comportent toutes exactement le même nombre de caractères. Vous pouvez aussi supposer que $m \geq 1$ et $n \geq 1$.

Pour résoudre ce problème, on vous suggère de:

1. Découper la chaîne en ses différentes lignes grâce à str.splitlines.
2. Initialiser un dictionnaire vide pour les jetons.
3. Boucler sur les lignes du damier (pensez à enumerate).
4. Boucler sur les caractères de chaque ligne.
5. Si un caractère diffère de `'.'`, alors l'ajouter au dictionnaire de jetons.
6. Retourner le tuple demandé.

Contexte de l'exercice

```
1
```

Bravo!

Votre score est **100/100**.

ATTENTION: cette soumission a été effectuée **après** l'échéance, elle ne sera **pas** considérée.

Entrez votre solution dans la cellule ci-dessous

Notez bien que **seul** le contenu de cette cellule sera **évalué** par le correcteur automatique.

```python
def decoder_damier(chn):
    dico = {}
    ligne = chn.splitlines()
    for i, n in enumerate(ligne):
        for p, case in enumerate(n):
            if case != '.':
                dico[i, p] = f'{case}'
    return (len(ligne), len(ligne[0]), dico)
```

Tester la présence d'un jeton

Définissez une fonction nommée `est_occupee` qui accepte en entrée **deux** arguments:

1. un **couple** (i, j) spécifiant respectivement les indices de **ligne** et de **colonne** d'une case d'un damier;
2. une **liste** de **dictionnaires** décrivant l'ensemble des jetons actuellement présents sur le damier;

où chaque **dictionnaire** de la liste contient exactement les **trois** clés suivantes:

1. `'jeton'` associé au symbole du jeton;
2. `'ligne'` associé à l'indice de ligne du jeton;
3. `'colonne'` associé à l'indice de colonne du jeton.

Votre fonction doit **retourner** un booléen indiquant si oui ou non la case (i, j) du damier est actuellement occupée par un jeton.

Par exemple, l'expression suivante:

```
est_occupee(
    (1, 1), [
        {'jeton': 'A', 'ligne': 0, 'colonne': 2},
        {'jeton': 'B', 'ligne': 1, 'colonne': 1},
        {'jeton': 'C', 'ligne': 2, 'colonne': 0},
    ]
)
```

doit retourner `True`, alors que l'expression:

```
est_occupee(
    (3, 1), [
        {'jeton': 'A', 'ligne': 0, 'colonne': 2},
        {'jeton': 'B', 'ligne': 1, 'colonne': 1},
        {'jeton': 'C', 'ligne': 2, 'colonne': 0},
    ]
)
```

doit retourner `False`.

Contexte de l'exercice

```
1
```

Bravo!

Votre score est **100/100**.

ATTENTION: cette soumission a été effectuée **après** l'échéance, elle ne sera **pas** considérée.

Entrez votre solution dans la cellule ci-dessous

Notez bien que **seul** le contenu de cette cellule sera **évalué** par le correcteur automatique.

```python
def est_occupee(couple, liste):
    for i in liste:
        if couple == (i['ligne'], i['colonne']):
            return True
    else:
        return False
```