Calcul d'un polynôme 2

Soit le polynôme :

$$p_n(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$$

où $n \geq 0$ est un entier arbitraire positif ou nul. Par exemple, pour un polynôme du second degré ($n = 2$) :

$$P_2(x) = a_2 x^2 + a_1 x + a_0$$

On vous demande d'écrire une fonction nommée `polynôme` qui accepte en **entrée** les arguments suivants :

1. la valeur x pour laquelle on veut évaluer ce polynôme ;
2. la liste $[a_n, a_{n-1}, \ldots, a_0]$ des coefficients du polynôme ;

et qui retourne en **sortie** la valeur de $P_n(x)$. Par exemple, `polynôme(2, [1, 2, 3])` doit retourner `11`.

Solution de référence

Notez qu'une solution n'est généralement **pas** unique.

In [ ]:

```python
def polynôme(x, coefs):
    # initialiser une variable d'accumulation
    poly = 0
    for i, a in enumerate(reversed(coefs)):
        # ajouter chaque terme de la somme
        poly += a*x**i

    # retourner le résultat
    return poly
```

Construire une énumération

On vous demande d'écrire une fonction nommée `énumérer` qui accepte en **entrée** une **liste** de chaînes de caractères et qui produit en **sortie** une **chaîne** dans laquelles les items de la liste d'entrée sont séparés par des virgules, sauf pour le dernier qui est séparé par `'et'`.

Par exemple, `énumérer(['un', 'deux', 'trois'])` doit **retourner** la chaîne `'un, deux et trois'`.

Faites aussi en sorte que votre fonction accepte un **second** argument permettant de spécifier le séparateur pour le dernier item de l'énumération. Donnez à cet argument la valeur **par défaut** `'et'`.

Indice: utilisez la méthode join des chaînes de caratères afin de joindre facilement les items reçus. Commencez par joindre avec cette méthode les premiers items en les séparant par une virgule, puis ajoutez le dernier à la suite du séparateur.

Notez que si la liste reçue ne contient qu'un seul élément, celui-ci doit être retourné sans autre traitement, et si la liste est vide, on doit retournez une chaîne vide.

Solution de référence

Notez qu'une solution n'est généralement **pas** unique.

In [ ]:

```python
def énumérer(items, sep='et'):
    if len(items) > 1:
        # énumérer les items
        return f'{", ".join(items[:-1])} {sep} {items[-1]}'

    else:
        # retourner l'item unique ou une chaîne vide
        return items[0] if items else ''
```

Tester la présence d'un jeton

Définissez une fonction nommée `est_occupée` 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_occupée(
    (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_occupée(
    (3, 1), [
        {'jeton': 'A', 'ligne': 0, 'colonne': 2},
        {'jeton': 'B', 'ligne': 1, 'colonne': 1},
        {'jeton': 'C', 'ligne': 2, 'colonne': 0},
    ]
)
```

doit retourner `False`.

Solution de référence

Notez qu'une solution n'est généralement **pas** unique.

In [ ]:

```python
def est_occupée(indices, jetons):
    # boucler sur les jetons à la recherche des indices
    for jeton in jetons:
        if (jeton['ligne'], jeton['colonne']) == indices:
            # jeton trouvé
            break
    else:
        # jeton pas trouvé
        return False

    return True
```

Chiffrer une chaîne de caractères

Soit une chaîne de caractères de longueur arbitraire. Cette chaîne est constituée de caractères potentiellement répétés. Par exemple, la chaînes `'AAABBCDDDDEE'`.

On vous demande d'écrire une fonction nommée `chiffrer_chaîne` qui accepte en **entrée** une telle chaîne et qui retourne en **sortie** une liste de couples (c, n) où c est un caractère et n le nombre de fois que ce caractère est répété de façon consécutive. Par exemple, `chiffrer_chaîne('AAABBCDDDDEE')` doit retourner la liste :

```
[('A', 3), ('B', 2), ('C', 1), ('D', 4), ('E', 2)]
```

Dans le cas d'une chaîne vide, votre fonction doit retournez une liste vide.

Pour résoudre ce problème, vous devriez initialiser deux variables c et n à partir du premier caractère de la chaîne, puis boucler sur les autres caractères. Chaque fois que le caractère courant diffère de c, il faut ajouter un nouveau couple (c, n) à la liste résultat; sinon, ajoutez 1 à n. N'oubliez pas à la fin d'ajouter le dernier couple à la liste avant de la retourner.

Solution de référence

Notez qu'une solution n'est généralement **pas** unique.

In [ ]:

```python
def chiffrer_chaîne(chaîne):
    # initialiser la variable d'accummulation
    résultat = []

    # s'assurer que la chaîne n'est pas vide
    if not chaîne:
        return résultat

    # initialiser avec le premier caractère de la chaîne
    c, n = chaîne[0], 0

    # boucler sur le reste de la chaîne
    for x in chaîne:
        if x == c:
            # une occurence de plus pour c
            n += 1
        else:
            # ajouter un couple à la liste
            résultat.append((c, n))

            # réinitialiser avec le nouveau caractère
            c, n = x, 1
    else:
        # ajouter le dernier couple
        résultat.append((c, n))

    return résultat
```

Compter les bulletins

Écriver une fonction nommée `compter_les_bulletins` qui accepte en entrée une **liste** de bulletins de votes (de longueur arbitraire) et qui produit en sortie un rapport sous la forme d'un **dictionnaire**.

Les bulletins de vote ont la forme d'un couple `(valide, condidat)` où `candidat` est le nom d'un des candidats à l'élection, et `valide` est un booléen qui indique si oui ou nom le bulletin est valide. Par exemple, pour la liste suivante de bulletins :

```
[(True, 'Pierre'), (False, 'Jean'), (True, 'Pierre'), (True, 'Jacques')]
```

Votre fonction doit **retourner** le dictionnaire suivant :

```
{
    "nombre total de bulletins": 4,
    "bulletins invalides": 1,
    "résultats du vote": {
        "Pierre": 2,
        "Jacques": 1
    }
}
```

Notez que vous n'avez **pas** à afficher le dictionnaire, seulement le **retourner**.

Solution de référence

Notez qu'une solution n'est généralement **pas** unique.

In [ ]:

```python
def compter_les_bulletins(bulletins):
    # initialiser la variable d'accumulation pour les bulletins invalides
    invalides = 0

    # initialiser le dictionnaire pour les résultats du vote
    résultats = {}

    # boucler sur les bulletins de vote
    for valide, candidat in bulletins:
        if valide:
            # ajouter un vote pour le candidat
            résultats[candidat] = résultats.get(candidat, 0) + 1
        else:
            # incrémenter le nombre de bulletins invalide
            invalides += 1

    # retourner le rapport sur l'élection
    return {
        'nombre total de bulletins': len(bulletins),
        'bulletins invalides': invalides,
        'résultats du vote': résultats,
    }
```