EXPLICATIONS :

EXO 8 : Factorielle

Le code que vous avez fourni est une implémentation en Python de la fonction factorielle, une fonction mathématique classique qui calcule le produit de tous les entiers positifs de 1 à n. Voici une explication détaillée du code :

Fonction `factorielle(n)`

```
python

def factorielle(n):
    """
    Calcule la factorielle d'un entier n.

    Args:
    - n : un entier

    Returns:
    - La factorielle de n
    """
    if n == 0:
        return 1
    else:
        return n * factorielle(n - 1)
```

Cette fonction calcule la factorielle d'un entier `n` en utilisant une approche récursive. Voici comment cela fonctionne :

- **Si `n` est égal à zéro, cela signifie que la factorielle de zéro est 1, donc la fonction renvoie 1.**
- **Sinon (c'est-à-dire si `n` est différent de zéro), la fonction renvoie `n` multiplié par la factorielle de `n - 1`. Cette opération est répétée récursivement jusqu'à ce que `n` devienne zéro.**

Test de la fonction avec contrôle de sécurité

```
python

while True:
    try:
        entier = int(input("Entrez un entier positif : "))
        if entier < 0:
            raise ValueError("Entrez un entier positif.")
        print("La factorielle de", entier, "est", factorielle(entier))
        break
    except ValueError as e:
        print(e)
```

- **Cette partie du code utilise une boucle `while True` pour s'assurer que l'utilisateur fournit un entier positif. Si l'entrée de l'utilisateur n'est pas un entier positif, une exception `ValueError` est levée avec un message d'erreur.**

- **Si l'entrée de l'utilisateur est valide, la fonction `factorielle` est appelée avec l'entier fourni, et le résultat est affiché.**

En résumé, ce code calcule la factorielle d'un nombre entier positif donné par l'utilisateur, en utilisant une approche récursive pour effectuer le calcul. Il inclut également une gestion d'erreur pour s'assurer que l'entrée de l'utilisateur est valide.

En Python, `raise` est un mot-clé utilisé pour déclencher une exception explicite à un endroit spécifique dans votre code. Lorsque vous utilisez `raise`, vous indiquez essentiellement que quelque chose d'inattendu s'est produit et que vous souhaitez lever une exception pour signaler cette situation.

Voici comment vous pouvez utiliser `raise` :

1. **Avec un type d'exception intégré :** Vous pouvez simplement écrire `raise` suivi du nom de l'exception que vous souhaitez lever. Par exemple :

   python

- `raise ValueError("Ce n'est pas une valeur valide.")`

- **Avec une instance d'exception :** Vous pouvez également lever une exception en fournissant une instance d'exception que vous avez créée. Par exemple :

python

2.
```
my_exception = ValueError("Ce n'est pas une valeur valide.")
raise my_exception
```

L'utilisation de `raise` est courante dans la gestion des erreurs et des exceptions. Lorsqu'une exception est levée, elle interrompt l'exécution normale du programme et recherche un gestionnaire d'exceptions approprié pour la gérer. Si aucun gestionnaire n'est trouvé, le programme s'arrête et affiche une trace de la pile (traceback) qui indique où l'exception a été levée.

Dans le code Python que vous avez fourni :

python

```
except ValueError as e:
    print(e)
```

Cela fait partie d'une structure `try-except` qui est utilisée pour gérer les exceptions dans le code. Voici ce que cela signifie :

- **`except ValueError as e:` : Cela indique que le bloc de code suivant va gérer toute exception de type `ValueError`. Les exceptions sont des erreurs qui se produisent lors de l'exécution du code.**
    - **`ValueError` est un type d'exception en Python qui est levé lorsque la fonction a reçu un argument du type correct mais d'une valeur inappropriée.**

- `as e` : Cela assigne l'objet de l'exception à la variable `e`. Cela permet d'accéder aux informations sur l'exception, telles que son message d'erreur.

- `print(e)`: C'est le bloc d'instructions qui s'exécute si une exception de type `ValueError` est levée. Il imprime simplement le message d'erreur associé à l'exception `e`.

En résumé, le bloc de code `except ValueError as e:` capture les erreurs de type `ValueError` qui peuvent survenir dans le bloc `try`, puis imprime le message d'erreur associé à cette exception. Cela permet au programme de gérer gracieusement les erreurs et de fournir des informations utiles sur ce qui s'est mal passé.

EXO 9 : Tri de liste

Le code que vous avez fourni est une implémentation en Python d'une fonction `tri_croissant(liste)` qui trie une liste de nombres par ordre croissant. Voici une explication détaillée du code :

Fonction `tri_croissant(liste)`

```
python

def tri_croissant(liste):
    """
    Trie une liste de nombres par ordre croissant.

    Args:
    - liste : une liste de nombres

    Returns:
    - Une nouvelle liste contenant les mêmes éléments triés par ordre croissant
    """
    try:
        sorted_list = sorted(liste)
        return sorted_list
    except TypeError:
        print("Erreur : La liste contient des éléments non comparables.")
        return []
```

- Cette fonction prend en entrée une liste de nombres.
- À l'intérieur de la fonction, il y a un bloc `try-except`. Le bloc `try` essaie de trier la liste donnée par l'utilisateur en utilisant la fonction `sorted()` de Python, qui renvoie une nouvelle liste triée. Si cette opération réussit, la liste triée est retournée.
- Si la tentative de tri échoue en raison d'une incompatibilité de types dans la liste (par exemple, si la liste contient des éléments qui ne peuvent pas être comparés entre eux), une exception `TypeError` est levée. Dans ce cas, le bloc `except` est exécuté, qui imprime un message d'erreur indiquant que la liste contient des éléments non comparables, puis retourne une liste vide `[]`.

Test de la fonction

```
python

while True:
    try:
        elements = input("Entrez les nombres séparés par des espaces :
").split()
        elements = [float(element) for element in elements]
        sorted_elements = tri_croissant(elements)
        print("Liste triée par ordre croissant :", sorted_elements)
        break
    except ValueError:
        print("Erreur : Veuillez entrer des nombres valides.")
```

- **Cette partie du code permet de tester la fonction `tri_croissant`. Elle utilise une boucle `while True` pour s'assurer que l'utilisateur fournit une liste valide de nombres.**
- **L'utilisateur est invité à entrer des nombres séparés par des espaces. La méthode `split()` est utilisée pour séparer les nombres et les convertir en une liste.**
- **Ensuite, une liste de nombres est créée en convertissant chaque élément de la liste en nombre flottant.**
- **La fonction `tri_croissant()` est appelée avec la liste de nombres fournie par l'utilisateur.**
- **Si l'utilisateur entre des valeurs qui ne peuvent pas être converties en nombres, une exception `ValueError` est levée. Dans ce cas, un message d'erreur est affiché, et l'utilisateur est invité à entrer à nouveau des nombres valides.**

En résumé, ce code permet de trier une liste de nombres donnée par l'utilisateur par ordre croissant en utilisant la fonction `sorted()` de Python. Il inclut également une gestion d'erreur pour traiter les cas où les éléments de la liste ne peuvent pas être triés en raison d'incompatibilités de types.

EXO 10 : Vérification de la primalité

Le code que vous avez fourni est une implémentation en Python d'une fonction `est_premier(n)` qui vérifie si un entier donné est un nombre premier ou non. Voici une explication détaillée du code :

Fonction `est_premier(n)`

```
python

def est_premier(n):
    """
    Vérifie si un entier est premier.

    Args:
    - n : un entier

    Returns:
    - True si l'entier est premier, False sinon
```

```
    """
    if n <= 1:
        return False
    elif n <= 3:
        return True
    elif n % 2 == 0 or n % 3 == 0:
        return False
    i = 5
    while i * i <= n:
        if n % i == 0 or n % (i + 2) == 0:
            return False
        i += 6
    return True
```

- **Cette fonction prend en entrée un entier `n` et retourne `True` s'il est premier et `False` sinon.**
- **Le test commence par vérifier si `n` est inférieur ou égal à 1. Si c'est le cas, `n` ne peut pas être premier, donc la fonction retourne `False`.**
- **Ensuite, il y a des conditions spéciales pour les petits nombres premiers. Si `n` est égal à 2 ou 3, la fonction retourne `True`.**
- **Ensuite, le code vérifie si `n` est divisible par 2 ou 3. Si c'est le cas, il ne peut pas être premier, donc la fonction retourne `False`.**
- **Ensuite, la fonction utilise un algorithme de test de primalité qui évite de tester tous les diviseurs potentiels de `n`. Cet algorithme utilise le fait que tous les nombres premiers supérieurs à 3 peuvent être écrits sous la forme 6k ± 1.**
- **La boucle `while` vérifie si `n` est divisible par l'un des nombres de la forme 6k ± 1 jusqu'à ce que le carré de `i` dépasse `n`.**
- **Si `n` est divisible par l'un de ces nombres, la fonction retourne `False`.**
- **Si aucun des tests de divisibilité n'échoue, alors `n` est premier, et la fonction retourne `True`.**

Test de la fonction

```
python

while True:
    try:
        entier = int(input("Entrez un entier positif : "))
        if entier < 0:
            raise ValueError("Entrez un entier positif.")
        if est_premier(entier):
            print(f"L'entier {entier} est premier.")
        else:
            print(f"L'entier {entier} n'est pas premier.")
        break
    except ValueError as e:
        print(e)
```

- **Cette partie du code permet de tester la fonction `est_premier()`.**
- **L'utilisateur est invité à entrer un entier positif.**
- **Si l'entier est négatif, une exception `ValueError` est levée.**
- **Sinon, la fonction `est_premier()` est appelée avec l'entier fourni.**

- **En fonction du résultat retourné par la fonction, le programme imprime si l'entier est premier ou non.**

En résumé, ce code utilise un algorithme efficace pour déterminer si un entier donné est premier ou non, et fournit une interface utilisateur pour tester cette fonctionnalité en entrant des entiers positifs.