TP Mastermind

Préambule

Prototype des fonctions

Vous trouverez avec cet énoncé de TP un dossier `Base`, qui contient plusieurs fichiers Python (.py). Tous les prototypes des fonctions que l'on vous demande d'implémenter sont répartis dans ces différents fichiers :

```
def ma_fonction_a_implementer(argument_1 ,argument_2, ...):
    '''
    Documentation de la fonction

    :param  argument_1: information sur le premier argument
    :type  argument_1: type du premier argument
    :param  argument_2: information sur le second argument
    :type  argument_2: type du second argument
    .
    .
    .
    :return: information sur l'argument de retour
    :rtype: type de l'argument de retour
    '''

    pass
```

Vous devez remplacer l'instruction `pass` par le corps de la fonction, de telle sorte qu'elle réponde aux spécifications de l'énoncé. La documentation vous sera utile pour bien comprendre quelles entrés/sortis doivent être utilisées. Nous vous demandons de porter une attention toute particulière à la manière dont est rédigée cette documentation : il vous sera demandé dans les TP suivants d'écrire vous même cette documentation !

Test du programme

En plus des prototypes des fonctions, un `main` vous est également fourni à la fin du programme :

```
if __name__=="__main__":
    # Question x.x
    print('Test de la fonction de la question x.x')
    print('Résultat escompté : ...')
    print('Résultat obtenu :')
    resultat = ma_fonction_a_implementer(arg1, arg2)
    print(resultat)

    # Question x.x+1
    ...
```

Cette partie du programme permet de tester le comportement de vos fonctions et de vérifier qu'elles renvoient les bons résultats. Ces tests peuvent être apparentés à des **tests unitaires** (nous en reparlerons au prochain TP). Dans les prochains TP, ce sera également à vous de rédiger ces tests pour vous assurer du bon fonctionnement de votre programme.

Exécution de votre code

Il est **impératif** d'exécuter votre code régulièrement. A chaque fonction implémentée, nous vous demandons d'exécuter le fichier qui contient le code que vous avez modifié. L'exécution d'un code en python se fait simplement en appelant la commande `python3` (version 3.x de python) sur le fichier considéré. Si vous utilisez un IDE (*IDLE3* de préférence pour commencer, évitez d'utiliser *Spider* !), vous pouvez utiliser les raccourcis permettant l'exécution de votre code. Lors de l'exécution de votre code, vous verrez apparaître dans votre console les traces de votre code :

```
Test de la fonction de la question x.x
Résultat escompté :
Le bon résultats que vous devez obtenir avec votre fonction
Résultat obtenu :
Le résultat de votre fonction
```

Vous pourrez ainsi vérifier que votre code fonctionne correctement. **Nous vous incitons vivement à ajouter vos propres tests dans la section 'main' de votre programme, afin de faciliter la création de vos fonction.**

Tant que toutes les fonctions d'un fichier ne sont pas implémentées, votre programme risque de produire certaines erreurs à l'exécution. Ne vous formalisez pas avec ces erreurs, elles disparaîtront normalement au moment ou vous implémenterez les bonnes fonctions de votre programme.

Comprendre le 'main'

Lorsque vous lancez la commande `python3` sur un fichier (`mon_fichier.py`), python crée de façon automatique un certain nombre de variables qui seront accessibles dans le code de votre fichier. Ce qui nous intéresse ici, c'est la variable `__name__` du programme, qui peut prendre les valeurs suivantes :

- Si la variable est appelé dans le fichier **principal** (placé juste après la commande `python3`), `__name__` prend la valeur `"__main__"`.
- Si la variable est appelé dans un fichier que l'on importe avec la commande `import` dans le fichier principal, `__name__` aura comme valeur le nom du module qui est importé.

Grâce à cette variable, nous pouvons ainsi placer du code dans notre fichier qui sera exécuté seulement si notre fichier est considéré comme principal. Ainsi, lorsque nous aurons besoin d'utiliser les fonctions de notre fichier dans un autre code, nous pourrons importer notre fichier en tant que module est le code placé dans le bloque `if __name__=="__main__":` ne sera pas exécuté.

Mastermind

Nous nous proposons d'écrire une version du jeu Mastermind. Le but de ce jeu est de découvrir une combinaison de pions de différentes couleurs. Le nombre de tentatives (de coups) est limité. A chaque coup, le joueur reçoit des indications sur la proximité de la combinaison qu'il a proposée avec celle qu'il

doit trouver. Ces indications sont le nombre de pions bien placés et le nombre de pions présents dans la combinaison mais mal placés.

Dans la version du jeu que nous allons proposer, les couleurs sont au nombre de 8. Dans notre version initiale du jeu, **il ne sera pas possible d'avoir une couleur en double dans une combinaison**. A chaque couleur identifiée par une lettre, nous associerons son nom et un code en utilisant des dictionnaires :

```
codes_couleurs = {"R":0, "B":1, "J":2, "V":3, "N":4, "M":5, "F":6, "O":7}
couleurs_codes = {0:"R", 1:"B", 2:"J", 3:"V", 4:"N", 5:"M", 6:"F", 7:"O"}
noms_couleurs = {"R":"Rouge", "B":"Bleu", "J":"Jaune", "V":"Vert", "N":"
    Noir", "M":"Marron", "F":"Fushia", "O":"Orange"}
```

Interaction

Les fonctions à implémenter se trouvent dans le fichier `interaction.py`.

1. Implémenter la fonction `affiche_indications(nb_places, nb_couleurs, proposition)` qui permet un affichage basique de l'état du jeu. La proposition sera passée à la fonction sous forme de liste d'entier. On se servira des dictionnaires pour obtenir les couleurs correspondantes aux codes.
2. Compléter la fonction `affiche_jolies_indications(nb_places, nb_couleurs, proposition)` qui permet un affichage plus agréable. Nous souhaitons un affichage de la forme : pions mal placés - rappel proposition - pions bien placés, comme ceci :

```
 *  *  *  *  RBJV
       *  *  BRJV  *  *
```

Qui se lira : "4 pions mal placés pour la combinaison RBJV et 0 pions bien placés" pour la première ligne et "2 pions mal placés pour la combinaison BRJV et 2 pions bien placés"

3. Écrivez une fonction permettant la saisie d'une proposition par le joueur. Cette saisie aura la forme d'une chaîne de caractères constituée des lettres représentant les couleurs, et votre fonction renverra la liste contenant le code correspondant. La fonction `input` en python permet de récupérer une chaîne de caractère écrite dans le terminal.
4. Nous allons maintenant ajouter une étape de vérification lors de la saisie de l'utilisateur. Implémenter la fonction `saisie_verif_joueur(nb_pions)` qui vérifie que l'utilisateur indique bien le bon nombre (nb_pions) de pions à placer, que ces pions sont de ont des couleurs autorisées et qui gérera le cas ou la saisie est faite en lettres minuscules plutôt que majuscules.

Mécanique de jeu

Les fonctions à implémenter se trouvent dans le fichier `mecanique.py`.

5. Écrivez une fonction `verification_position(solution, proposition)` qui retourne le nombre de pions bien placés. La solution et la proposition sont fournies sous forme de liste d'entiers de même taille.
6. Écrivez une fonction `verification_couleur(solution, proposition)` qui retourne le nombre de pions présents dans la combinaison, mais mals placés.
7. Écrivez une fonction `verification(solution, proposition)` qui retourne le nombre de pions bien placés et le nombre de pions présents dans la combinaison, mais mals placés. **Attention**, les pions comptabilisés comme bien placés ne devront pas aussi être compté dans le nombre de pions mal placés !

Interface de jeu

Les fonctions à implémenter se trouvent dans le fichier `interface.py`.

8. Écrivez la méthode permettant de jouer. Cette méthode aura comme paramètres la combinaison à découvrir (liste d'entiers) et le nombre de coups autorisés pour découvrir la bonne combinaison. Vous réutiliserez les fonctions précédemment implémentées.
9. Afin de pouvoir jouer contre un ordinateur, prévoyez une fonction générant de manière aléatoire une combinaison de pions d'une taille donnée. Cette combinaison sera retournée sous forme d'une liste d'entiers.

Aller plus loin

Dans cette partie, vous modifierez les fonctions déjà existantes et créerez de nouvelle fonctions à votre convenance.

10. Complétez le jeu en permettant au joueur de choisir le niveau difficulté : nombre de couleurs utilisées, nombre de pions de la combinaison et nombre de coups autorisés pour découvrir la bonne combinaison.
11. Ajoutez la possibilité d'avoir des couleurs en double dans une combinaison.
12. Implémentez une version du jeu complètement autonome en incorporant un algorithme permettant de deviner la combinaison du mastermind.