UNIVERSITE LIBRE DE BRUXELLES
École Polytechnique de Bruxelles

12 novembre 2012

INFO-H-100 : Introduction à la programmation
Projet 1 : Mastermind

Le but du projet est d'implémenter en *Python 3* une version simplifiée, à un joueur, du Mastermind [1]. Le jeu se déroule de la manière suivante :

- Le moteur de jeu génère au hasard une combinaison de 4 pions de couleurs **différentes**, choisies parmi les associations lettre-couleur suivantes :
  `[("r", "rouge"), ("j", "jaune"), ("v", "vert"), ("b", "bleu"), ("o", "orange"), ("c", "blanc"), ("t", "violet"), ("f", "fuchsia")]`
  Vous pouvez utiliser la fonction `shuffle` du module `random` pour générer la combinaison aléatoirement.
- Le joueur a 10 essais pour deviner la combinaison.
- À chaque essai, l'ordinateur indique :
  - le nombre de pions de la bonne couleur bien placés ;
  - le nombre de pions de la bonne couleur mais mal placés.

  L'ordinateur peut, par exemple, indiquer que deux pions sont de la bonne couleur et bien placés, un pion est de la bonne couleur et mal placé, et le joueur en déduira qu'un pion est erroné.
- Quand le joueur a deviné la combinaison ou qu'il a épuisé ses 10 essais, le programme annonce que la partie est terminée, affiche la bonne combinaison en cas d'échec, et demande au joueur s'il veut rejouer.

Si le joueur se trompe (c'est-à-dire qu'il entre une combinaison de couleurs non valide), l'ordinateur lui propose de recommencer. Le joueur peut néanmoins aussi bien entrer des lettres en majuscule qu'en minuscule.

Vous devez gérer les saisies de l'utilisateur et les tester afin d'assurer la cohérence entre la donnée récupérée et le type de données requis. Par contre, les préconditions des fonctions, qui seront spécifiées dans les *docstrings*, ne doivent pas être explicitement revérifiées au début de vos fonctions. Ces conditions sont supposées remplies lors de l'utilisation de ces fonctions (voir consignes et règles de bonne pratique).

Exemple de partie (les entrées de l'utilisateur sont soulignées) [2]

```
Il vous reste 10 essai(s)
Devinez la combinaison : rvof
Vous avez joué la combinaison : rouge vert orange fuchsia
Réponse :
Aucun pion bien placé

Il vous reste 9 essai(s)
Devinez la combinaison : rvbf
Vous avez joué la combinaison : rouge vert bleu fuchsia
Réponse :
Nombre de pions de la bonne couleur mais mal placés : 1

Il vous reste 8 essai(s)
Devinez la combinaison : CBOF
Vous avez joué la combinaison : blanc bleu orange fuchsia
Réponse :
Nombre de pions de la bonne couleur bien placés : 2

Il vous reste 7 essai(s)
Devinez la combinaison : cbtj
Vous avez joué la combinaison : blanc bleu violet jaune
Réponse :
Nombre de pions de la bonne couleur bien placés : 2
Nombre de pions de la bonne couleur mais mal placés : 2

Il vous reste 6 essai(s)
Devinez la combinaison : cbjt
Vous avez joué la combinaison : blanc bleu jaune violet
Réponse :
Nombre de pions de la bonne couleur bien placés : 4

Bravo !
Voulez-vous rejouer ? (o/n) : o
```

1. Voir `http://fr.wikipedia.org/wiki/Mastermind` pour les règles complètes.
2. Nous les avons soulignées ici pour plus de clarté ; dans votre programme, il s'agira de texte "normal".

```
Il vous reste 10 essai(s)
Devinez la combinaison : rvtf
Vous avez joué la combinaison : rouge vert violet fuchsia
Réponse :
Nombre de pions de la bonne couleur mais mal placés : 2

Il vous reste 9 essai(s)
Devinez la combinaison : jofb
Vous avez joué la combinaison : jaune orange fuchsia bleu
Réponse :
Nombre de pions de la bonne couleur bien placés : 1
Nombre de pions de la bonne couleur mais mal placés : 1

...

Il vous reste 1 essai(s)
Devinez la combinaison : TRJO
Vous avez joué la combinaison : violet rouge jaune orange
Réponse :
Nombre de pions de la bonne couleur bien placés : 1
Nombre de pions de la bonne couleur mais mal placés : 3

Vous avez écoulé vos essais... Dommage !
La bonne combinaison était : violet orange rouge jaune (TORJ)
Voulez-vous rejouer ? (o/n) : N
Au revoir !
```

Exemple d'entrées erronnées (les entrées de l'utilisateur sont soulignées)

```
Il vous reste 10 essai(s)
Devinez la combinaison : 12RG
Veuillez entrer une combinaison de couleurs existantes !
Veuillez n'utiliser que des lettres !
Devinez la combinaison : abcv
Veuillez entrer une combinaison de couleurs existantes !
Devinez la combinaison : BBBB
Veuillez entrer une combinaison de couleurs différentes !
Devinez la combinaison : b
Veuillez entrer une combinaison de longueur 4 !
Devinez la combinaison : 1
Veuillez entrer une combinaison de longueur 4 !
Veuillez entrer une combinaison de couleurs existantes !
Veuillez n'utiliser que des lettres !
Devinez la combinaison :
```

On vous demande de découper votre programme en fonctions dont voici quelques prototypes à implémenter obligatoirement :

- `mastermind()` : lance le jeu.
- `generate_guess(colors)` : génère et renvoie la combinaison à deviner.
- `valid_colors(comb, colors)` : renvoie `True` si la combinaison fait référence aux couleurs possibles, `False` sinon.
- `check_unique(comb)` : renvoie `True` si la combinaison ne contient que des couleurs différentes, `False` sinon.
- `count_well_placed(guess, comb)` : renvoie le nombre de pions de la bonne couleur bien placés.
- `count_colors(guess, comb)` : renvoie le nombre de pions de la bonne couleur mais mal placés.
- `check_win(guess, comb)` : renvoie `True` si le joueur a gagné, `False` sinon.
- `to_colors_name(comb, colors)` : prend une combinaison et renvoie une chaîne de caractères comprenant le nom complet des couleurs de la combinaison choisie.

Notez qu'à l'exception de `mastermind()`, aucune des fonctions présentées ci-dessus n'interagit avec l'utilisateur. Ce découpage n'est pas complet et vous êtes invités à définir d'autres fonctions afin de rendre votre programme et vos fonctions le plus modulaire possible. Chaque fonction doit avoir une tâche **unique** et bien définie. En particulier, une fonction ne peut jamais dépasser 10 lignes (maximum strict).

Consignes

Remise du projet

Le projet est à réaliser **seul** et à remettre **au plus tard le 10 décembre 2012 à 12h30**. Nous vous demandons de vous limiter **uniquement** à la matière vue au cours des 7 premiers TPs.

Vous devez remettre une version électronique de votre code sur un dépôt Git que vous devez créer dans le groupe suivant : `http://wit-projects.ulb.ac.be/rhodecode/INFO-H-100/2012/1`.
L'utilisation de ce système est détaillée sur le Wiki [3] et une séance d'information sera organisée le 16 novembre 2012 à 12h30 au UB4.130 (salle Socrates).

Consignes sur le code

Les critères d'évaluation sont les suivants :
- Le code **résout exactement le problème demandé sans ajout ni apport personnel**, qualité de l'algorithmique, qualité et pertinence de la découpe en fonction, syntaxe correcte, indentation parfaite, variables bien nommées.
- Le code doit également **respecter les conventions et les règles de bonne pratique** vues au cours et aux séances d'exercices (résumées sur le Wiki [4] et rappelées à la page suivante). **En particulier, vous devez être attentif à ce que chaque fonction soit correctement documentée par une *docstring***.
- Le respect des consignes.

Les étudiants-assistants sont disponibles à 12h30 dans la salle Socrate afin de répondre à vos questions.
Consultez leur horaire sur le Wiki :
`http://wit-projects.ulb.ac.be/chiliproject/projects/info-h-100/wiki/Cours`.

3. `http://wit-projects.ulb.ac.be/chiliproject/projects/info-h-100/wiki/Remise_du_projet_1_avec_git`
4. `http://wit-projects.ulb.ac.be/chiliproject/projects/info-h-100/wiki/Regle_bonne_pratique`

Règles de bonne pratique

Ces règles ont été déterminées dans un but pédagogique et/ou de bonne pratique générale en programmation.

Forme et structure

- Utilisez des noms de variables qui indiquent ce que les variables représentent.
- Utilisez une découpe intelligente en fonctions. Chaque fonction doit réaliser une tâche **unique** clairement identifiée.
- Ne faites pas de fonctions de plus de 10 lignes (maximum strict), exception faite pour des `print` successifs qui ne compteront que pour 1 ligne.
- Evitez la redondance dans votre code (copier-coller). Si cela arrive, c'est probablement qu'il manque soit une fonction, soit une boucle, soit que des tests conditionnels peuvent être regroupés.
- N'utilisez pas de variable globale sauf dans le cas où vous voulez utiliser cette variable comme une constante.
- Veillez à ce que tous les paramètres et variables d'une fonction soient utilisés dans cette fonction.

Style

- Utilisez la forme raccourcie :

```
if is_leap_year(2008):
```

  plutôt que :

```
if is_leap_year(2008) == True:
```

- Utilisez la forme :

```
return <expression booleenne>
```

  plutôt que la forme équivalente :

```
if <expression booleenne>:
    res = True
else:
    res = False
return res
```

Gestion des erreurs

- Dans le cadre de ce cours, vous pouvez supposer que le programmeur sait ce qu'il fait. Les fonctions qui documentent clairement leurs hypothèses dans leur *docstring* (soit toutes les fonctions que vous écrivez pour les projets, voir paragraphe suivant) ne doivent plus vérifier elles-mêmes que ces conditions sont bien remplies : c'est la responsabilité de l'appelant de passer des arguments corrects.
- Par contre, on ne peut jamais faire confiance à l'utilisateur du programme. Vous devrez donc toujours vérifier les entrées de l'utilisateur. Toute fonction dont le rôle est de récupérer une valeur entrée par l'utilisateur doit clairement documenter les promesses qu'elle fait sur sa valeur de retour (par exemple, toujours renvoyer un entier pair supérieur à 0), et **doit** effectivement renvoyer le bon type de valeur dans le bon domaine. Si l'utilisateur n'entre pas des données valides, la stratégie préférée sera d'afficher un message expliquant ce qui ne va pas, et de lui redemander.

Docstring

- Une fonction doit être correctement documentée par une *docstring*. C'est-à-dire pour chaque fonction : son prototype, une description succincte de la tâche qu'elle remplit, ses paramètres et leur domaine (le type et les valeurs possibles), le type de ses valeurs de retour et leur domaine et un exemple d'utilisation.
- La *docstring* est le texte qui sera affiché lorsque vous tapez `help(days_in_month)` dans l'interpréteur ; il s'agit donc de la documentation nécessaire à un programmeur pour utiliser la fonction. Ce texte ne s'adresse pas à l'utilisateur du programme (le joueur), et n'explique pas le fonctionnement interne de la fonction ; il explique uniquement comment utiliser la fonction.
  Exemple :

```
def days_in_month(month, year):
    """Calcule le nombre de jours d'un mois.

    Arguments:
        month (int) : le mois considéré, entre 1 et 12.
        year (int) : l'année considérée, strictement supérieure à 0.

    Valeurs de retour:
        int. Retourne le nombre de jours associés au mois et à l'année considérés.

    Exemples:
        >>> print(days_in_month(2, 2012))
        29
    """
    if month in [1, 3, 5, 7, 8, 10, 12]:
        nb_days = 31
    elif month in [4, 6, 9, 11]:
        nb_days = 30
    elif month == 2:
        if is_leap_year(year):
            nb_days = 29
        else:
            nb_days = 28
    return nb_days
```