ICC: Programmation

Examen intermédiaire

Section MX, 9 novembre 2018

Veuillez déposer votre carte CAMIPRO sur votre table.
Tous les exercices sont à faire ***individuellement****.*

Vous avez droit à tout votre matériel et à un accès à Internet, mais ***toute forme de communication avec une autre personne, directe ou indirecte, par quelque moyen que ce soit, est interdite et est, le cas échéant, immédiatement sanctionnée par la note 0****.*

L'examen se réalise exclusivement sur l'infrastructure officielle de l'EPFL via les clients légers en salles d'exercice. L'usage de tout autre appareil électronique (tablette, smartphone, smartwatch, etc., ou périphérique comme clé USB, écouteurs, clavier personnel, etc.) est exclu.

Pour que le code soit considéré comme complètement correct, il faut qu'il marche comme spécifié dans la consigne ***même si les données utilisées (dans le code de base à télécharger ou générées par votre code) changent****.*

Si votre code ne compile pas ou crashe, mettez-le en commentaire. Vous pourrez quand même obtenir des points si votre idée est bonne (mais vous n'obtenez pas de points pour la description textuelle d'une idée sans écrire de code).

Dans les exemples de résultats affichés sur le terminal, les parties soulignées doivent provenir d'une variable ou d'un calcul fait dans votre code, et non d'une chaîne de caractères prédéfinie.

Merci d'attendre le feu vert du surveillant pour tourner la page.

Exercice 1. Choix multiples

Répondez au questionnaire sur la page Moodle du cours, auquel vous accéderez avec le code **`OpenSesame`**. 20 pts

Exercice 2. Démarrage

Démarrez Visual Studio Code normalement et ouvrez correctement votre *workspace*; créez un nouveau fichier Python pour l'examen intermédiaire. Allez sur la page Moodle du cours et copiez-collez le code de départ dans votre nouveau fichier. Le code, partiellement commenté, doit tourner correctement et afficher deux «sections» vides sur le terminal quand on le lance. 5 pts

Dans tout le code que vous écrirez maintenant, on vous demande d'indiquer les types uniquement pour les paramètres des fonctions et méthodes ainsi que le type de retour (y compris `None`) des fonctions et méthodes, sauf celui de la méthode spéciale `__init__`. Vous n'avez pas besoin de déclarer le type des variables locales** (mais pouvez le faire si vous le souhaitez). **Votre code doit être exécutable par Python 3.7 et ne doit pas faire appel à d'autres modules externes que ceux préimportés dans le code de base.

Merci d'insérer votre code chaque fois sous la ligne de commentaire qui correspond à la partie de l'exercice. Par exemple, tapez le code de l'exercice 3 (a) sous la ligne `### 3 (a)` du code initial, sans la supprimer.

Exercice 3. Fonctions, conditions, boucles

Pour vérifier si n, un nombre entier positif représentable par un **`int`** en Python, est un multiple de 3, on pourrait vérifier si **`n % 3 == 0`**. Mais ici, nous allons utiliser un autre système: nous allons déterminer si n est un multiple de 3 en vérifiant si la somme de ses chiffres est elle-même un multiple de 3.

(a) Ajoutez une fonction nommée **`sum_of_digits`** qui accepte un **`int`** comme paramètre unique et retourne le **`int`** résultant de l'addition des chiffres qui composent le paramètre. On part du principe que le paramètre est positif (mais pas forcément strictement). N'utilisez pas la fonction prédéfinie **`sum`** de Python. 10 pts

Testez votre fonction en lui passant successivement les valeurs **`0`**, **`3`**, **`29`**, **`999`** et **`1111`** et affichez les résultats respectifs sur le terminal.

(b) Ajoutez une fonction nommée **`sum_until_single_digit`** qui accepte aussi un **`int`** comme paramètre et qui retourne, par des appels répétés à **`sum_of_digits`**, la *somme répétée inférieure à 10 des chiffres* du paramètre. 8 pts

On définit *somme répétée inférieure à 10 des chiffres* avec cet exemple: prenons 99999. La somme de ces chiffres est $9+9+9+9+9 = 45$. Comme 45 n'est pas inférieur à 10, on additionne encore ses chiffres et on obtient $4 + 5 = 9$. Comme 9 est inférieur à 10, on définit 9 comme la *somme répétée plus petite que 10 des chiffres* de 99999.

Testez votre fonction en lui passant les mêmes valeurs de test que pour (a) et affichez les résultats respectifs sur le terminal.

(c) Ajoutez une fonction nommée **`is_multiple_of_3`** qui accepte aussi un **`int`** comme paramètre et qui retourne une valeur indiquant si oui ou non son argument est un multiple de 3. Ceci doit se faire en appelant la fonction **`sum_until_single_digit`** et en vérifiant si la valeur retournée par celle-ci est 0, 3, 6 ou 9. 8 pts

Testez votre fonction comme en (a) et (b) avec les mêmes valeurs de test et affichez les résultats.

(d) Estimez, par génération aléatoire, la probabilité d'obtenir un multiple de 3. Pour ce faire, générez 1000 fois un **`int`** aléatoire entre 0 (compris) et 99 (non compris) avec l'expression **`randrange(0, 99)`** et comptez combien de fois vous obtenez un multiple de 3 tel que vérifié par un appel à votre fonction **`is_multiple_of_3`**. Affichez le résultat (qui doit en moyenne tourner autour de 0.333, et qui changera légèrement à chaque exécution) sur le terminal selon ce format: 9 pts

`Sample probability to get a multiple of 3: 0.353`

(e) Écrivez une fonction **one_digit_multiples_of** qui accepte un **int** comme paramètre et qui retourne une **List[int]**, potentiellement vide, de tous les multiples de ce paramètre inférieurs à 10, en omettant la valeur triviale 0. 9 pts

(f) En une ligne de code, définissez une variable **multiples** et assignez-lui comme valeur une liste contenant: comme premier élément, la liste de tous les multiples de 1 inférieurs à 10; comme deuxième élément, la liste de tous les multiples de 2 inférieurs à 10; comme troisième élément, la liste de tous les multiples de 3 inférieurs à 10; etc., jusqu'à 9. Utilisez pour cela une expression avec une compréhension de liste faisant appel à la fonction **one_digit_multiples_of**. (Attention à ne pas créer de boucle infinie en générant une liste infinie des multiples de 0.) 6 pts

Affichez ensuite le contenu de cette variable sur le terminal. Votre code devrait afficher ceci:

```
[[1, 2, 3, 4, 5, 6, 7, 8, 9], [2, 4, 6, 8], [3, 6, 9], [4, 8], [5], [6], [7], [8], [9]]
```

Exercice 4. Classes, champs

Nous allons modéliser des saisons de séries télévisées et manipuler des données qui les définissent.

(a) Pour modéliser une saison d'une série, créez une nouvelle classe nommée **SeriesSeason**. Elle modélisera une saison d'une série en stockant ces données dans des champs avec les noms suivants: 8 pts

- **series_name** pour stocker le nom de la série;
- **season_num** pour stocker le numéro de la saison (la première sera numérotée 1, puis 2, etc.);
- **start_year** pour stocker l'année où le premier épisode de cette saison a été diffusé;
- **num_episodes** pour stocker le nombre d'épisodes dans cette saison.

Ajoutez la méthode standard **__init__** pour donner les valeurs initiales respectives à ces champs. Déterminez le type approprié pour chacun.

(b) Ajoutez la méthode standard **__repr__** dans la classe **SeriesSeason** pour faire en sorte qu'une saison soit convertie en texte selon cet exemple: 5 pts

```
"The Expanse", season 1 (2015), 10 episodes
```

Dans le code de base, décommentez la définition de la variable **seasons** et affichez toutes les saisons sur le terminal.

(c) En parcourant la liste **seasons**, remplissez un **Dict[str, int]** qui fait correspondre chaque nom de série au nombre total d'épisodes diffusés pour de cette série, obtenu en additionnant le nombre d'épisodes pour chacune des saisons de la série. Affichez le résultat selon ce format (l'ordre des séries n'est pas considéré comme important): 8 pts

```
3 series are defined:
- "The Expanse" has 36 episodes
- "Game of Thrones" has 73 episodes
- "Westworld" has 20 episodes
```

(d) Ajoutez du code qui vérifie s'il existe dans les données fournies deux saisons d'une même série avec le même numéro de saison. Si c'est le cas, affichez-le selon ce format: 12 pts

```
There are multiple mentions of season 4 of "Game of Thrones"
```

(e) Ajoutez du code pour détecter si une saison n d'une série donnée commence avant les saisons 1 à $n-1$. Vous devriez détecter 3 incohérences dans les données fournies. Affichez-les selon ce format: 12 pts

```
Season 2 of "Westworld" reportedly started in 2014, which is earlier than season 1 (2016)
```

N'oubliez pas de rendre votre fichier .py sur Moodle.
Vérifiez que votre fichier ait bien été uploadé correctement et qu'il ait la bonne extension (.py et non .pyc ou autre).

Total: 120 pts