ICC: Programmation

Examen intermédiaire

Sections GC & MX, 8 novembre 2019

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 **`GoForItNow`**. 20 pts

Exercice 2. Démarrage

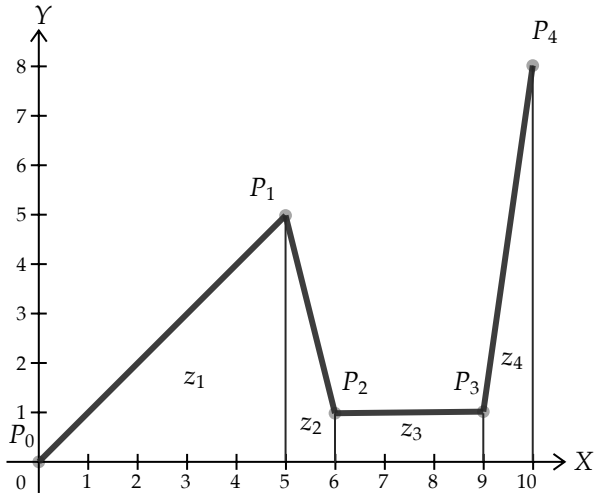
Démarrez Visual Studio Code normalement et ouvrez correctement votre *workspace*; créez un nouveau fichier Python pour l'examen intermédiaire. Allez sur la page Moodle du cours et copiez-collez le code de départ dans votre nouveau fichier. Le code, partiellement commenté, doit tourner correctement et 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. Structures de contrôles

Le code de base définit deux listes de nombres, *xs* et *ys*, toutes les deux de longueur $p = 5$. Elles définissent la représentation graphique d'une fonction mathématique obtenue en reliant linéairement les 5 points P_0 à P_4, où la coordonnée x de P_i est xs_i (en code: **`xs[i]`**) et sa coordonnée y est ys_i (donc **`ys[i]`**):



(a) Avec du code, vérifiez tout d'abord que les deux listes **`xs`** et **`ys`** aient la même longueur. Si cela est le cas, affichez: 5 pts

`Data OK`

Sinon, affichez:

`Correcting length mismatch`

et modifiez la liste la plus longue en supprimant les n derniers éléments qu'elle contient en plus par rapport à la liste la plus courte. Testez votre code en décommentant la ligne **`ys.append(1000)`** et en vérifiant que votre code supprime effectivement la valeur ainsi ajoutée indûment à **`ys`**.

Dans les deux cas, assignez une variable nommée **`p`**, qui indiquera, selon les deux listes fournies, combien de points ont été définis (ici, elle devrait valoir 5).

(b) Affichez la liste des points définis par les deux listes (si nécessaire, modifiées comme décrit au point précédent) selon ce format: 5 pts

```
Considered points: (0.0, 0.0), (5.0, 5.0), (6.0, 1.0), (9.0, 1.0), (10.0, 8.0)
```

(c) Ajoutez une fonction nommée **`is_monotonic_increasing`** qui retournera **`True`** ou **`False`** selon que la liste de nombres passée en paramètre contient des valeurs dans un ordre strictement croissant, c'est-à-dire si la valeur de chaque «case» de la liste est plus grande que toutes les précédentes et plus petite que toutes les suivantes. 8 pts

Affichez le résultat de l'appel de votre fonction avec **`xs`** et **`ys`** selon ce format:

```
xs is monotonic inscreasing: True
ys is monotonic inscreasing: False
```

(d) Ajoutez une fonction **`interpolated_y`** qui accepte un paramètre x de type **`float`** et qui retourne la valeur y obtenue par interpolation linaire entre les points définis par **`xs`** et **`ys`**. Pour cela, déterminez d'abord dans quelle zone z_i la valeur x appartient: si $xs_{i-1} \leq x \leq xs_i$, alors x appartient à z_i et la valeur interpolée sera: 9 pts

$$y = ys_{i-1} + \frac{(ys_i - ys_{i-1})(x - xs_{i-1})}{xs_i - xs_{i-1}}.$$

Si x n'appartient à aucune zone définie par la fonction, retournez la valeur spéciale obtenue par l'expression **`float("nan")`**.

(e) Avec une boucle, affichez la valeur y interpolée calculée par votre fonction pour toutes les valeurs x entre 0 et 10 (compris) par incrément de 0.1 selon ce format (ceci doit donc afficher 101 lignes en tout): 4 pts

```
x=9.5 -> y=4.5
```

(f) Pour chaque valeur x de xs, vérifiez que l'expression **`interpolated_y(x)`** calcule bien la valeur correspondante de ys. Si ce n'est pas le cas, affichez l'erreur selon le format 8 pts

```
Mismatch for x=0: y should be 0, but interpolated value is 0.1
```

(Vous ne devriez cependant pas avoir d'écart de ce type.) Si aucun écart n'est constaté, affichez une unique fois ceci:

```
No mismatch found
```

(g) Ajoutez une fonction **`areas_under_curve`** qui retourne la liste des aires des zones z_i, où l'aire de z_i est 12 pts

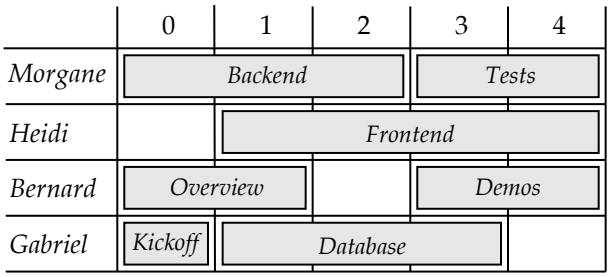
$$A(z_i) = \frac{(xs_i - xs_{i-1})(ys_i + ys_{i-1})}{2}.$$

Testez votre fonction en affichant le résultat de son appel. Calculez et affichez ensuite la somme de ces aires. Votre code devrait afficher ceci:

```
[12.5, 3.0, 3.0, 4.5]
Total area: 23.0
```

Exercice 4. Structures de données

Nous allons modéliser l'emploi du temps de 4 personnes sur un projet de 5 jours (numérotés de 0 à 4) en définissant qui fera quelle tâche quand. Voici la représentation graphique des données du code:



(a) Pour modéliser ces tâches, créez une nouvelle classe nommée **`Task`**. Elle représentera une tâche (un rectangle gris selon la représentation ci-dessus) en stockant ces données dans des champs avec les noms suivants: 6 pts

- **`person`** pour stocker le nom de la personne à qui elle a été attribuée;
- **`title`** pour stocker le titre de la tâche;
- **`start_day`** pour stocker l'index du jour où elle est censée commencer;
- **`duration`** pour stocker le nombre de jours qu'elle est censée durer;
- **`difficulty`** pour stocker un «indice de difficulté» de la tâche.

Ajoutez la méthode standard **`__init__`** pour donner les valeurs initiales respectives à ces champs. Déterminez le type approprié pour chacun. En cas de doute sur les types, suivez les types des valeurs utilisées pour créer les tâches dans la liste **`tasks`**.

(b) Ajoutez la méthode standard **`__repr__`** dans la classe **`Task`** pour faire en sorte qu'une tâche soit convertie en texte selon ces exemples: 7 pts

```
Task for Gabriel: "Kickoff" (on day 0)          si la tâche dure un seul jour;
Task for Heidi: "Frontend" (from day 1 to day 4)  sinon.
```

Dans le code de base, décommentez la définition de la variable **`tasks`** et affichez toutes les tâches sur le terminal.

(c) En parcourant la liste **`tasks`**, remplissez un **`Dict[str, List[str]]`** nommé **`occupations`** qui fait correspondre chaque prénom à une liste de 5 valeurs. La case *i* de ces listes doit contenir le nom de la tâche qui est assignée à la personne correspondante le jour *i*, ou la chaîne de caractères vide (**`""`**) s'il ou elle n'aura rien à faire. 12 pts

Affichez le contenu du dictionnaire ainsi rempli ligne par ligne selon ce format:

```
Bernard's occupation: ['Overview', 'Overview', '', 'Demos', 'Demos']
```

(d) Pour chaque personne, affichez tous les jours où elle ou il n'a rien de prévu selon ce format: 6 pts

```
Bernard has nothing to do on day 2
```

Pour ce faire, vous pouvez utiliser les informations de la variable **`occupations`** remplie au point précédent.

(e) Ajoutez du code qui détecte si, dans le planning des tâches proposé, il y a des conflicts, c'est-à-dire si, pour une personne donnée, une tâche a lieu sur un jour déjà occupé par une de ses autres tâches. 14 pts

Si votre code ne détecte aucun conflit (ce qui doit être le cas si vous n'avez pas modifié **`tasks`**), affichez:

```
No conflicts detected
```

Sinon, affichez les conflits, un par ligne, selon ce format:

```
Morgane should work on "Tests" on day 2 but is already busy
```

Pour tester un cas avec des conflits, décommentez la ligne qui contient **`tasks[-1].start_day -= 1`**, et qui a pour effet de faire travailler Morgane un jour plus tôt sur les tests et ainsi de créer un conflit de planning.

(f) On définit la *difficulté planifiée* d'une tâche comme sa difficulté (comme stockée par son champ **`difficulty`**) multipliée par sa durée (comme stockée par son champ **`duration`**). 15 pts

Écrivez du code qui trouve la personne dont la somme des difficultés planifiées des tâches est la plus grande et affichez le résultat selon ce format:

```
Morgane has the largest sum of planned difficulties: 3.7
```

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`.

Total: 136 pts