UNIVERSITÉ JEAN MONNET SAINT-ÉTIENNE

Outils Info. TP5: Cellules

Rendu : sous forme d'archive

À la fin, vos réponses seront rendues dans une archive zip : cette archive contiendra un fichier `compte-rendu.txt` (avec votre **nom**, **groupe**, et les réponses aux questions précédées d'une apostrophe) et vos fichiers pythons.

Pour faire une **archive zip** de votre TP pour le rendu, exécutez, dans le terminal :

```
cd ~/outils-info/
zip -r tp5.zip tp5/
```

Important : mise en place

- travaillez dans un dossier dédié au TP, lui même dans `~/outils-info`,
- créez un fichier `compte-rendu.txt` pour écrire la date, vos noms et les réponses aux questions précédées d'une apostrophe,
- mais vous pouvez travaillez directement dans le dossier `tp5` (sans sous-dossier par exercice).

Exercice 1 – Introduction et téléchargement

Dans ce TP, des fichiers de base vous sont donnés et vous allez utiliser la bibliothèque qtido (récupérer la dernière version sur le site du cours). Il vous faudra donc télécharger une archive contenant les fichiers et une archive contenant la bibliothèque (`qtido.py` et un exemple pour vérifier que la bibliothèque fonctionne). Dans les fichiers donnés, certaines parties ne doivent pas être modifiées.

Q1) Téléchargez et décompressez (extrayez le contenu) les archives `ressources-tp5.zip`, ainsi que la bibliothèque graphique. Il vous sera demandé par la suite de copier certains de ces fichiers en les renommant.

Exercice 2 – Quelques Fonctions

Q2) Copiez le fichier fourni `listes.py` en tant que `ex2.py`. Attention à bien copier le fichier et à ne pas modifier le fichier d'origine.

NB: les fonctions de cet exercice ne doivent rien afficher (pas de `print`).

Q3) Écrivez une fonction `au_carre` qui reçoit une liste (contenant des nombres) et renvoie une nouvelle liste contenant les carrés des nombres.

Q4) Écrivez une fonction `au_cube` qui reçoit une liste (contenant des nombres) et renvoie une nouvelle liste contenant les cubes des nombres.

Q5) Écrivez une fonction `au_logbase` qui reçoit une base (un nombre réel) et une liste (contenant des nombres) et renvoie une nouvelle liste contenant les log_b des nombres. Rappel : $log_b(v) = \frac{log_{10}(v)}{log_{10}(b)}$

Q6) Écrivez une fonction `somme` qui reçoit une liste (contenant des nombres) et renvoie la somme des valeurs.

Q7) Écrivez une fonction `produit` qui reçoit une liste (contenant des nombres) et renvoie le produit des valeurs.

Q8) Écrivez une fonction `somme_2d` qui reçoit une liste de listes de nombres et renvoie la somme des valeurs.

Q9) Écrivez une fonction `produit_2d` qui reçoit une liste de listes de nombres et renvoie le produit des valeurs.

Structure typique d'un programme avec fonction « principale »

Un programme Python doit être proprement structuré avec 4 sections : 1) les imports des modules utilisés par le programme, 2) la définition des fonctions nécessaires, 3) la définition de la fonction principale, qui contient le code de base du programme, 4) l'appel de la fonction principale.

Voici un exemple qui trace trois carrés.

```
1
2  # 1) importation de bibliothèques
3  from qtido import *
4
5  # 2) définition des fonctions
6  def carre(fen, x, y, taille):
7      rectangle(fen, x, y, x+taille, y+taille)
8
9  # 3) définition de la fonction principale
10 def principale():
11     f = creer(600, 500)
12     carre(f, 100, 100, 100)
13     carre(f, 400, 100, 100)
14     carre(f, 225, 300, 150)
15     attendre_pendant(f, 1000)
16     exporter_image(f, "toto.png")
17
18 # 4) appel de la fonction principale
19 principale()
```

Exercice 3 – Fonctions...

Q10) Copiez le fichier fourni `inputs.py` en tant que `ex3.py`. Attention à bien copier le fichier et à ne pas modifier le fichier d'origine.

Q11) Implémentez les fonctions manquantes (`input_liste`, `en_float`, `au_carre`, `afficher_liste_multiligne`) en « devinant » à partir du programme ce qu'elles doivent faire, sans modifier la partie principale du programme. En particulier, si l'utilisateur tape `1 2 20`, l'exécution du programme doit donner l'affichage suivant :

```
1 Nombres (séparés par des espaces) : 1 2 20
2 1 - 1.0
3 2 - 4.0
4 3 - 400.0
5 Total : 405.0
```

Q12) Faites que votre programme conserve son comportement mais n'utilise pas de variables globales.

Exercice 4 – Cellules en évolution

On va créer une simulation d'évolution de cellules. Pour vous aider, une simulation (fausse) vous est donnée et il vous faudra uniquement changer les règles d'évolution.

Q13) Copiez le fichier fourni `cellules.py` en tant que `ex4.py`. Attention à bien copier le fichier et à ne pas modifier le fichier d'origine.

Q14) Assurez vous que le fichier `outils.py` est présent à coté de `ex4.py` (copiez le au besoin).

Q15) Étudiez le code du programme et lancez le et pressez la barre d'espace.

Q16) Faites que le programme conserve son comportement mais n'utilise pas de variables globales.

Les règles d'évolution des cellules se basent sur l'état courant de la cellule (vivante ou non) et le nombre de voisin (parmi les 8 voisins) qui sont vivants à ce moment.

Q17) Remplacez la « règle simple et fausse » par les règles suivantes :

- une cellule vivante survie uniquement si elle a 2 ou 3 voisins,
- une cellule naît uniquement si elle a exactement 3 voisins.

Pour valider votre programme, des fonctions d'initialisation de la grille vous sont données, chacune ayant un nom qui transmets ce qui doit se produire avec cet état initial.

Q18) Quelques idées pour aller plus loin (sans ordre précis) :

- Faire que l'âge (nombre d'itérations où la cellule a survécu) de la cellule soit stocké, bien calculé, et affiché (couleur différente selon l'âge).
- Faire que les cellules évoluent automatiquement toutes les secondes.
- Tracer des lignes verticales et horizontales pour mieux visualiser les cases.