Le Mans Université, Master 1 économie

Python – TP1 : Bases du langage – 4h

I Utilisation de Spyder

Exercice 1 : Cellules

Entrez les deux lignes suivantes dans un document tp1.py et séparez les de quelques lignes.

```
# %% Exerice 1, partie 1
# %% Exerice 1, partie 2
# %% Exerice 1, partie 3
```

Vous verrez apparaître des cellules dans votre document.

Dans la seconde cellule, affectez 1 à la variable a et affichez la variable a. Exécutez la cellule (ctrl + entrée). Dans la troisième cellule, augmentez de 1 la valeur se trouvant dans a et affichez la nouvelle valeur. Exécutez la cellule. Exécutez de nouveau la cellule. Dans la première cellule, affichez a. Tirez-en vos conclusions. Remarquez l'alerte donnée à cette ligne car vous appelez une variable qui, si l'on exécute les lignes dans un ordre normal, n'a jamais été assignée.

Pour éviter les interactions indésirables entre les variables de différents exercices, vous pouvez taper dans la console, avant de lancer l'exercice, la commande :

```
%reset
```

Dans la pratique, vous créerez au moins **une cellule par exercice**. Ces cellules seront très utiles lors de rendus de devoirs afin d'avoir, en un seul document, vos réponses à tous les exercices.

Remarque : On aurait pu avoir le même comportement (voire des cellule plus faciles à utiliser) en utilisant un *Notebook* mais nous aurions eu à disposition moins d'outils de debug.

Exercice 2 : Outils de debug

1. Créez une variable a et lui affecter la valeur 1. Créer une variable b et lui affecter la valeur 2.

   Commenter le code suivant en détaillant ce qu'il calcule :

   ```
   while a+b < 200:
       temp=a
       a=a+b
       b=temp*b
   ```

   En sortie de boucle, affichez a puis affichez b.

2. Lancez le script en mode débogage (ctrl + F5). Rien ne se passe mais vous constatez que l'invite de commande dans la console affiche « ipdb> ». Cela signifie que le débogueur est activé.
3. Utilisez plusieurs fois « Exécuter la ligne en cours » (ctrl + F10) et suivez l'évolution du programme en regardant la console et l'explorateur de variables.

   On constate que le débogage commence à la première cellule du document. Il n'y a malheureusement pas moyen de déboguer une cellule précise. Il y a deux options pour y remédier partiellement :

   — Ouvrir un nouveau document dans lequel on ne met que le code à déboguer. Ici, on copierait le contenu de la seconde cellule.

   — Placer un point d'arrêt (double-clic dans la marge) au début de la cellule, en face d'un ligne où il y a du code (pas un commentaire). Le lancement du debug exécute alors le code jusqu'à ce point d'arrêt.

   Essayez les deux méthodes. L'utilisation du débogueur doit devenir assez naturelle pour vous dès que le code ne se comporte pas comme on l'attend.

II Structures de données

Exercice 3 : Tuples

Affectez aux variables a, b et c les valeurs 1, 2 et 3 en une seule instruction. Intervertissez les valeurs contenues dans a et b **sans** passer par une troisième variable.

N'hésitez pas à prendre quelques minutes pour vous documenter, au passage, sur le rôle des tuples dans Python.

Exercice 4 : Listes 1

Créez une liste *etuPren* contenant les prénoms Benjamin, Olivia, Mohamed.

Créez leur liste d'âges *etuAge* contenant leurs âges respectifs : 21, 24, 22.

Ajoutez Maria en début de liste des prénoms.

Ajoutez l'âge 23 (de Maria) à la fin de la liste des âges.

Affichez à l'écran combien d'étudiants sont dans la liste.

Les positions des prénoms et des âges ne correspondent plus. Changer la liste des prénoms afin de les faire correspondre aux âges.

Afficher sur une même ligne le message suivant : « le nombre d'étudiants est x » , où x est le nombre d'étudiants (remplacé par le programme).

Exercice 5 : Listes 2

Créez, en une seule instruction, une liste l d'entiers composée 50 fois de suite de la séquence de valeurs 1, 2, 3, 4. Concaténez [10, 9, 8] et la liste l pour former s.

Affectez à la variable a la 5-ième valeur de la liste.

Placez dans une variable $l2$ la liste composée des 10 premières valeurs de s. Même chose avec les 10 dernières dans une liste $l3$.

Exercice 6 : Listes 3

Créez en une seule ligne la liste des 50 premiers multiples de 7. Vous pourrez vous intéresser à l'utilisation de **for** dans la définition d'une liste.

Exercice 7 : Ensembles

Placez dans deux variables les ensembles $e1 = \{1, 3, 8, 11\}$ et $e2 = \{2, 3, 3, 3, 4, 8, 9\}$.

Affichez $e2$. Combien d'éléments $e2$ comporte-t-il ?

Affichez, sans faire de boucle, les éléments communs à $e1$ et $e2$.

Affichez, sans faire de boucle, les éléments qui sont soit dans $e1$ soit dans $e2$ (mais pas les deux).

Affichez **True** si $e1$ est égal à $e2$ et **False** sinon.

Exercice 8 : Boucle for et ensemble

Écrivez un programme Python affichant tous les nombres de 0 à 60 sauf 3, 6, 7, 14, 19, 21 et 57.

Utilisez **for**, **in** et un ensemble.

Exercice 9 : Appartenance ?

Créez une liste d'entiers. Affectez à la variable a une valeur entière de votre choix.

En une ligne, vérifiez si a se trouve dans la liste. Le résultat sera True ou False.

III Input

Exercice 10 : Pair ou impair ?

Écrivez un programme pour déterminer si un nombre donné par l'utilisateur est pair ou impair, afficher un message approprié à l'utilisateur.

Exercice 11 : Produit ou 100

Écrivez un programme Python qui renverra le produit de deux entiers demandés à l'utilisateur. Toutefois, si le produit est compris entre 10 et 100, il renverra 100.

Exercice 12 : Première fonction

Écrivez une fonction calculant la distance entre les points de coordonnées (x_1, y_1) et (x_2, y_2).

IV Boucles : for et while

Exercice 13 : Itérateur VS liste

Tapez le code

```
a = range(8)
b = [0, 1, 2, 3, 4, 5, 6, 7]
```

Faites une boucle **for** sur i **in** a pour afficher les i. Idem en remplaçant a par b.

Comparez les résultats.

Est-ce que $a == b$?

Tester les instructions :

```
c = list(range(8))
```

Que vaut alors c ?

Exercice 14 : Entiers dans un intervalle

Écrivez un programme demandant deux réels $a < b$ et affichant tous les entiers compris dans l'intervalle $[a, b]$.

On pourra importer la librairie math à l'aide de la commande

```
import math
```

et se servir des fonctions **ceil** et **floor**.

Exercice 15 : Devinette aléatoire

Écrivez un programme Python qui génère un entier choisi uniformément parmi les entiers de 0 à 9. Vous pouvez pour cela utiliser l'instruction **help** sur le module **random**.

L'utilisateur doit ensuite deviner l'entier. Tant qu'il fait une mauvaise proposition, le programme lui indique si sa valeur est trop grande ou trop petite et l'invite à essayer à nouveau.

Exercice 16 : Bing Bang

Écrivez un programme Python qui affiche les nombres entiers de 1 à 100. Pour les multiples de trois afficher "Bing" au lieu du nombre et pour les multiples de cinq "Bang". Pour les nombres qui sont des multiples de trois et cinq imprimer "BingBang".

Vous pourrez proposer deux variantes : une où l'on crée d'abord la liste qu'on affiche ensuite, et l'autre où l'on affiche sans mémoriser.

V Fonctions

Exercice 17 : Nombre premier

Écrivez une fonction **IsPrime(n)**, qui renvoie **True** si n est un nombre premier, sinon **False**.

Exercice 18 : Factorielle

Écrire une fonction factorielle **CalFact(n)** retournant la valeur de la factorielle d'un entier n donné en argument, sans utiliser la récursivité (l'appel de la fonction à elle-même).

Exercice 19 : Liste des nombres premiers

Écrivez une fonction qui établit la liste des nombres premiers de compris entre 1 et 10^6 et la retourne en sortie. Ce programme utilisera ladite liste.

Conseil : commencez avec la liste des premiers inférieurs à 100... On ne sait jamais !

Exercice 20 : Somme des impair récursive

Écrivez une fonction Python pour calculer la somme d'une liste de nombres en utilisant la récursivité (la fonction s'appelle elle-même).

Utiliser cette fonction pour afficher la somme des nombres impairs de 1 à 100.

Exercice 21 : Factorielle (bis) Écrire une fonction factorielle **CalFactR(n)** retournant $n!$ pour un entier n donné en argument en utilisant la récursivité.

VI Exercices de synthèse

Exercice 22 : Dénombrement guidé On souhaite faire du dénombrement sur des entiers à 4 chiffres (0000 et 0001 en font partie). L'objectif est de compter combien il existe de nombres à 4 chiffres tels que l'une de ces deux conditions est validée :

- la somme des deux chiffres de gauche est égale au produit des deux chiffres de droite
- le produit des 4 chiffres est égal au nombre

Nous allons procéder en plusieurs étapes.

1. Écrire une fonction chiffre(nombre, puissance) qui renvoie le chiffre correspondant à la puissance de 10 dans le nombre. Par exemple, chiffre(2748, 2) renverrait 7. En effet, $10^2 = 100$ et le chiffre des centaines est 7. chiffre(2748, 0) renvoie 8 et chiffre(2748, 3) renvoie 2.
2. Ecrire une fonction estValide(nombre) qui détermine si le nombre doit être compté ou pas. Il est valide (retourne True) s'il s'agit d'un entier positif à 4 chiffres et s'il

vérifie une deux deux conditions énoncées ci-dessus. Sinon, la fonction renvoie False.

3. Écrire une boucle parcourant tous les entiers à 4 chiffres. Dans cette boucle, chaque nombre est testé pour savoir s'il vérifie une des deux conditions et un compteur est entretenu pour obtenir le résultat final. Afficher tous les nombres valides.

Exercice 23 : Dénombrement en autonomie Combien existe-t-il de nombres de 7 chiffres composés seulement de 0 et de 1 tels que deux 1 ne sont jamais consécutifs ?