ISN – Informatique et Sciences du Numérique

TP4 PYTHON : GESTION DE FICHIERS ET STRUCTURATION DU CODE

1 – TUPLE

*Un **tuple** est une **liste optimisée** pour **l'accès en lecture** mais qui **n'est pas modifiable**. Pour définir un tuple il faut procéder comme pour les listes mais en utilisant des parenthèses. Comme pour les listes, les différents éléments d'un tuple sont **indexés**. Leur accès est identique.*

Exercice 1

Taper dans l'interpréteur les lignes ci-dessous. **Justifier** les résultats obtenus.

```
>>>t = (1,2,3,4,5,6,7,8)
>>>print(t)
>>>print(t[0])
>>>print(t[0:5])
>>>print(t[3:])
>>>t[0] = 5
```

2 – MANIPULATION DE FICHIERS

La manipulation de fichier passe par 3 étapes : Ouverture du fichier, action sur le fichier et fermeture du fichier.

*Un fichier peut être ouvert dans **différents modes**, en fonction de l'action à réaliser :*

- *En **lecture** (mode « **r** » comme read) : **lecture seule** des données contenues dans le fichier. Si le fichier n'existe pas, un message d'erreur va apparaître.*
- *En **écriture** (mode « **w** » comme write) : **écriture de données dans un nouveau fichier**. Si le fichier qui doit être créé existe déjà, il est écrasé par le nouveau fichier.*
- *En **ajout** (mode « **a** » comme append) : **ajout de données à un fichier déjà existant**. Si le fichier à ouvrir existe déjà, l'écriture des données commence à la fin du fichier. Si le fichier n'existe pas, il est créé.*

Syntaxe et exemple

```
variable = open("Nom_fichier.extension","mode")
fic = open("niveau_1.txt","r")
```

*Le fichier manipulé doit **se trouver dans le répertoire courant du programme**, sinon le paramètre « Nom_fichier.extension » doit contenir le chemin absolu (chemin complet partant de la racine de l'arborescence des fichiers).*

*L'ouverture du fichier **crée une variable** (fic dans l'exemple) qui permettra **d'accéder au fichier**. Cette variable est un **descripteur de fichier**.*

Il existe plusieurs fonctions permettant de lire des données dans un fichier :

`readlines()` : ***lit toutes les lignes** du fichier et les places dans une **liste**.*
`readline()` : ***lit une liste** et la place dans une **chaîne de caractères**.*
`read()` : ***lit toutes les données** du fichier et les place dans une **seule chaîne de caractères**.*
`read(n)` : ***lit n caractères** et les place dans une **chaîne de caractères**.*

Pour écrire dans un fichier, il faut utiliser la fonction `write()` qui prend en paramètre la chaîne de caractère à écrire dans le fichier.

Exemple

```
fic.write("---+   +---")
```

Une fois la manipulation sur le fichier terminée, il est nécessaire de le fermer par l'intermédiaire de la fonction **`close()`**.

Exemple

```
fic.close()
```

Exercice 2

1. **Ecrire** un script permettant d'afficher le contenu du fichier « exercice2.txt » fourni. **Tester-le**. **Justifier** l'affichage obtenu.
2. **Modifier** le script afin d'obtenir un affichage sur 2 lignes sans saut de ligne supplémentaire. **Tester-le** et **vérifier** le bon fonctionnement du programme
3. **Modifier** le script afin de rajouter, nom et prénom sur une troisième ligne et la classe sur une quatrième.
4. **Tester-le** et **vérifier** le bon fonctionnement du programme en ouvrant le fichier à l'aide d'un éditeur de texte.

Une instruction permet également d'enlever les caractères superflus en fin ou début de chaînes de caractères : `strip()`.

Syntaxe

```
Chaine.strip()
```

Exercice 3

1. **Modifier** le script permettant d'afficher le contenu du fichier « exercice2.txt » fourni.
2. **Tester-le**. **Justifier** l'affichage obtenu.

3 – STRUCTURATION DU CODE : MODULES

*Il est possible d'enrichir python avec de nouvelles fonctionnalités disponibles dans des **modules**. Les fonctions fournies par un module ne sont disponibles que si le **module a été chargé**. Pour charger un module il faut utiliser l'instruction `import`. Toutes les fonctions sont alors accessibles en préfixant leur nom par le nom du module. Les modules sont généralement chargés en début de programme. Un module se présente comme un simple fichier python (extension .py).*

Exemple : fonction sqrt (racine carrée) contenue dans le module « math »

```
import math
racine = math.sqrt(4)
```

Il est également possible de structurer le code en créant des modules contenant les fonctions du programme. Ces modules seront chargés par le programme principal.

Exercice 4

1. **Ecrire** le script suivant. **Enregistrer-le** sous le nom « mon_module.py ».

```
#-------------------------------------
# Name: mon_module
#-------------------------------------
def ma_fonction() :
    print("Ok")
```

2. **Ecrire** un script permettant l'exécution de la fonction « ma_fonction ». **Enregistrer** puis **exécuter** le script. **Justifier** le résultat obtenu.

4 – LABYRINTHE

Les différents labyrinthes seront contenus dans des fichiers dont le nom aura le format suivant « niveau_N.txt » (niveau_1.txt, niveau_2.txt…). Le fichier Niveau_1.txt est fourni dans le dossier « TP4 Fichiers et modules ».

Le programme lab4.py est la nouvelle version du labyrinthe. Le script est constitué du programme et de 6 fonctions.

4.1 – Fonction charge_labyrinthe

La fonction `charge_labyrinthe` *permet de retourner le contenu du fichier « nom.txt ». Les données seront retournées sous de tuples. Pour cela il est nécessaire de convertir les listes contenues dans le fichier tuples au moyen de la fonction* `tuple(data)`.

Il est possible de définir plusieurs labyrinthes dans différents fichiers. Il est donc simple de changer de niveau en fonction du nombre de niveaux disponibles : ***niveau_1.txt***, ***niveau_2.txt****… Une simple boucle dans le programme principal permet de faire cela. A chaque tour de boucle la fonction* `charge_labyrinthe` *est appelée et permet de générer le labyrinthe courant (variable* ***niveau****) à partir des fichiers* ***niveau_1.txt***, ***niveau_2.txt****,….*

Programme permettant le changement de niveau

```
#-----------------------------------------------------------------------
# Programme principal
#-----------------------------------------------------------------------
perso = "X"
pos_perso = [1,1]
nb_niveaux = 20 # Nombre total de niveaux

# Lancement du jeu
for num_niveau in range(1,nb_niveaux + 1) : # num_niveau : de 1 à 20
    niveau = charge_labyrinthe("niveau_" + str(i))
```

Exercice 5 : Labyrinthe – Fonction charge_labyrinthe

1. **Compléter**, dans le script lab4.py, la ligne 8 afin d'ouvrir en lecture le fichier « nom.txt ».
2. **Compléter** les lignes 13 et 14 afin de lire toutes les lignes du fichier « nom.txt » et de les placer dans la liste **data**.
3. **Expliquer** le rôle des lignes 15 et 16.

4.2 – Fonction barre_score

Il est possible de changer de niveau mais le joueur doit connaître en permanence le niveau dans lequel il se trouve. Il est important de rajouter une « barre de score » qui indiquera le niveau et d'autres informations qui seront rajoutées lors des prochains TP. La fonction `barre_score` *sera appelée par la fonction « jeu ».*

Exercice 6 : Labyrinthe – Fonction barre_score

1. **Compléter** la ligne 93 permettant d'afficher le numéro du niveau courant.

4.3 – Fonction verification_deplacement

*Le changement de niveau ne doit être effectif que lorsque le personnage est sorti du niveau courant. La **sortie du labyrinthe** sera repérée par le **caractère « 0 »**. Pour détecter la présence du personnage sur la sortie, il est nécessaire de modifier la fonction* `verification_deplacement`.

*Si le personnage se trouve sur la sortie, la fonction « `verification_deplacement` » doit retourner les valeurs de **positions colonne et ligne** **[-1,-1]**.*

Exercice 7 : Labyrinthe – Fonction verification_deplacement

1. **Compléter** les lignes 47 et 48 permet retournée les valeurs [-1,-1] lorsque le déplacement du personnage lui permet d'arriver sur la case sortie.

4.4 – Fonction jeu

Dans la fonction `jeu`*, si **pos_perso** = **[-1,-1]**, il faut **afficher** « **Vous avez réussi le niveau X !** » et sortir de la fonction* `jeu` *par l'intermédiaire de l'instruction* `break`.

Exercice 8 : Labyrinthe – Fonction jeu

1. **Compléter** les lignes 108 et 109 afin d'afficher « **Vous avez réussi le niveau X !** » lorsque le personnage est arrivé sur la case sortie.
2. **Créer**, à l'aide de « Wordpad », un nouveau labyrinthe qui sera enregistré dans le fichier **niveau_2.txt**.
3. **Tester** le fonctionnement du programme et **vérifier** le passage correct de niveau.

4.5 – Structuration du script

Dans le but de structurer au mieux le code, le script va être divisé en deux fichiers. Le module « module_lab.py » qui contiendra toutes les fonctions du jeu et le fichier « jeu_lab » qui contiendra le programme principal.

Exercice 9 : module_lab et jeu_lab

1. **Créer** les fichiers « **module_lab.py** » et « **jeu_lab.py** » en séparant les fonctions et le programme principal.
2. **Exécuter** le programme principal et **vérifier** le bon fonctionnement de l'ensemble.