Éléments de syntaxe en vrac

I Fichiers

❦ Définition. *Fichier*

Un *fichier* est un ensemble de données cohérentes stockées sur l'ordinateur. Un fichier possède un *nom* et une *extension*. L'extension détermine le type de données contenues par le fichier.

— `nadine.wav`, `albumine.mp3` contiennent de la musique
— `maline.jpg`, `heroine.png`, `insuline.bmp` contiennent des images
— `raspoutine.py` contient du code python
— `bakounine.txt` contient du texte
— …

❦ Définition. *Arborescence de Fichier*

Un fichier est contenu par un *dossiers*, dit *parent*. Le dossier parent peut lui même être contenu par un autre dossier, et ainsi de suite jusqu'à la *racine* qui n'est contenue par personne. Un dossier peut contenir plusieurs fichiers ou dossiers.

Ceci formant une *arborescence*.

Sous Windows las racine est le nom du disque (souvent `C:/`) qui contient le fichier. Sous Linux et Max c'est tout simplement le dossier `/`.

❦ Définition. *Chemin Absolu*

Le chemin *absolu* d'un fichier est constitué de la séquence des dossiers à traverser *depuis la racine* jusqu'au dossier parent, suivi du nom de fichier et de son extension.

Exemple :

— `C:\Users\nadine\Documents\informatique\tp5\tp5.py`
— `/home/nadine/informatique/tp5/tp5.py`

❦ Définition. *Chemin Absolu*

Le chemin *absolu* d'un fichier est constitué de la séquence des dossiers à traverser *depuis le dossier courant* jusqu'au dossier parent, suivi du nom de fichier et de son extension.

Exemple :

— `informatique\tp5\tp5.py` depuis le dossier `Documents`

— `informatique/tp5/tp5.py` depuis le dossier utilisateur `nadine`.

En Python on peut créer un *objet fichier* grâce à la fonction `open` en spécifiant son chemin et le mode d'ouverture. Attention, pour donner le chemin on ajoutera un `r` avant la chaîne de caractère pour pouvoir utiliser le caractère \ sans le dédoubler.

```
file = open(r"informatique/tp5/tp5.py","r")
```

Mode d'ouverture	Description
r	Ouvre un fichier en lecture seule. Il est impossible de modifier le fichier. Le pointeur inter
r+	Ouvre un fichier en lecture et en écriture. Le pointeur interne est placé au début du fichier
a	Ouvre un fichier en écriture seule en conservant les données existantes. Le pointeur intern
a+	Ouvre un fichier en lecture et en écriture en conservant les données existantes. Le pointeu
w	Ouvre un fichier en écriture seule. Si le fichier existe, les informations existantes seront su
w+	Ouvre un fichier en lecture et en écriture. Si le fichier existe, les informations existantes se

L'objet fichier permet de manipuler le fichier dont le chemin a été donné en paramètre. Avec les droits adéquats :

— Lecture du fichier dans une chaîne de caractère : `s = file.read()`
— Lecture du fichier dans une liste de chaînes de caractère. Chaque élément de la liste correspond à une ligne du fichier : `l = file.readlines()`
— Lecture d'une ligne du fichier dans une chaînes de caractère : `l = file.readline()`
— Écriture d'une chaîne de caractère dans un fichier `n = file.write(s)`. La fonction renvoie le nombre de caractères écrits.

En plus de séparer le fichier en une liste de lignes, on peut le séparer en une liste de mots. Un mot est un ensemble de caractère différent d'un espace. On peut séparer les mots d'une chaîne de caractère pour obtenir une liste de mot grâce à la fonction `split`.

```
l = s.split()
```

Après utilisation (lecture, écriture ou les deux), le fichier doit être fermé :

```
file.close()
```

II Modules

❦ Définition. *Module*

> Un module est un fichier de code Python qui contient des fonctions et des objets Python. Pour pouvoir utiliser ces fonctions et objets, il faut *importer* le module par son nom : **import** `module`.

On peut renommer le module lors de l'importation à l'aide de la syntaxe **import** `module` **as** `alias`.

```
import numpy as np # Importe le module numpy et le renomme np
```

Les fonctions et objets du modules sont alors accessible en précisant le nom, ou l'alias, du module avant la fonction :

```
import nadine
import numpy as np
x = nadine.fonction(y)
t = np.full((12,12),0.0)
```

On peut importer une ou plusieurs fonctions ou objets particulier du module.

```
from nadine import gagarine # Importe la fonction gagarine du module nadine
# Importe les fonctions full, array et matric du module numpy
from numpy import full, array, matrix
```

On peut alors utiliser directement ces fonctions sans préciser le module :

```
x = gagarine(y)
t = full((12,12),0.0)
t2 = array([1,2,3])
```

Une très mauvaise pratique consiste à importer toutes les fonction d'un module, sans importer le module en lui même :

```
from numpy import *
```

III Assertions

❦ Définition. *Assertion*

Une *assertion* est une condition qui si elle est fausse provoque une erreur d'exécution.

On peut écrire une assertion en Python grâce à la syntaxe suivante **assert** `condition`.

Exemples :

```
assert x > 0
assert len(l) > 0 and x != 0
assert len(l1) == len(l2)
```