Initiation à Python - leçon 3.3

s1
Dans cette séquence, nous allons découvrir le moyen de prendre en compte des données extérieures provenant du clavier ou d'un fichier. Nous allons également voir comment afficher des données à l'écran ou dans des fichiers.

---
s2

Pour les problèmes qui nous intéressent, les interactions d'un utilisateur avec un programme se feront par l'attente de données entrées au clavier, et par l'affichage de résultats ou de messages à l'écran. Ce n'est pas vrai pour une application web ou un jeu.

La syntaxe pour que le programme affiche un message demandant à l'utilisateur d'entrer une donnée est la suivante.

Python affiche le message (sans passer à la ligne) et attend que l'utilisateur valide par la touche "Entrée" une réponse au clavier, puis il crée un objet de type chaîne de caractères et le référence par la variable ma_variable.

Voyons ça sur un exemple.

Notez bien que c'est un objet chaîne de caractères qui est créé. Si cette chaîne de caractères est en fait un entier ou un flottant, il faudra bien entendu la convertir pour pouvoir l'utiliser en tant que telle. Ceci se fait grâce aux fonctions int ou float.

---
s3

Un mot sur les opérateurs de formatage qui sont utiles pour définir la façon dont vont être affichées les variables.
Si l'on souhaite afficher par exemple un réel avec une certaine précision (nombre de chiffres avant et après la virgule) ou bien un entier avec un texte explicatif, il est nécessaire de procéder à un formatage et de mettre en place un gabarit

Dans ce premier exemple, on affiche une valeur entière précédée de la chaîne "x=". Le gabarit "x=" suivi d'un entier est appliqué à la variable entière 2

Voici un gabarit permettant d'afficher la même valeur 2 comme un réel.

Voici comment afficher les deux valeurs d'un tuple.

---
s4

Une autre façon de procéder est d'utiliser la méthode format() de l'objet chaîne de caractères.

On déclare une variable gabarit comme une chaîne de caractères au sein de laquelle on prévoit des emplacements pour des variables. Ces emplacements sont des paires d'accolades. Ce gabarit est formaté grâce à des données qui sont passées en argument de la méthode format et vont occuper ici les emplacements dans l'ordre dans lequel elles sont fournies.

Une façon un peu plus élaborée consiste à donner un numéro à ces emplacements - ici de 0 à 2. Dans cet exemple, le résultat est affiché par un flottant avec 2 décimales.

Je vous propose de reproduire les exemples présentés.

---
s5

Souvent, on désire lire des données présentes dans un fichier présent sur le disque dur, sur une clé usb, etc. ; de la même façon, on peut vouloir écrire des données sur un fichier.

Ce fichier peut être accessible en lecture seule, ou en lecture et écriture.

Python propose un objet file pour gérer les interactions avec les fichiers. Que ce soit en mode lecture ou en mode écriture, nous verrons que Python doit d'abord ouvrir le fichier et, quand le traitement est fini (lecture ou écriture dans ce fichier) doit le fermer afin de libérer les ressources utilisées par la création de l'objet file. On peut laisser Python se charger lui-même de fermer le fichier quand tout est terminé.

Selon le système d'exploitation (Windows, Linux, Mac OSX, etc.), la notion de fichier est différente. Cependant l'objet file offre des méthodes qui permettent de masquer ces différences. Dans tous les cas, un fichier possède un nom et réside (ou est créé) dans un dossier. Pour accéder à ce fichier, il faut donc indiquer son nom et le chemin d'accès à son dossier.

Comme pour le chargement des modules, si l'on indique seulement le nom, Python va chercher dans le "répertoire courant", c'est-à-dire le répertoire contenant le programme Python en cours d'exécution (ou si l'on travaille en mode interactif, le répertoire du dernier programme utilisé).

Voyons cela sur un exemple.

---
s6

La première fonction à mettre en oeuvre pour lire ou écrire dans un fichier est la fonction open

La fonction open() prend comme argument obligatoire le nom du fichier.
L'argument optionnel mode permet de définir si l'on est en mode lecture indiqué par la lettre 'r' (et qui est le mode par défaut), en mode écriture par la lettre 'w' ou en mode ajout par la lettre 'a'.
L'argument optionnel buffering contient le nombre de lignes à stocker dans le buffer. Le buffer est une zone mémoire de taille limitée qui sert à stocker des données (généralement de façon temporaire).

Commençons par un exemple de lecture.

Première chose à faire, télécharger le fichier présent djinns.txt sur la page du cours et le déplacer dans le répertoire de travail. Dans les exercices qui vont suivre, nous supposerons que les fichiers utilisés en mode lecture ou écriture sont dans le “dossier courant”. Ces fichiers seront donc simplement désignés par leur nom. Mais si on veut un contrôle précis sur les dossiers, on importera le module os qui fournit quelques fonctions utiles que nous avons déjà introduites os.chdir("chemin") permet de changer le dossier de travail, os.getcwd() renvoie le dossier de travail actuel.

Dans l'éditeur interactif IDLE, ouvrez un fichier Python et sauvez-le dans le même répertoire que le fichier texte.

Reproduisez les lignes suivantes et exécutez ce fichier. Que fait Python ? Deux choses
- la création d’un objet de type file qui représente le fichier physique djinns.txt
- la création d'une variable monFichier qui référence l'objet file

Attention, il ne faut pas confondre les objets de type file tels qu’ils sont créés et manipulés par Python avec la concrétisation “physique” de ces fichiers sur le disque. Les objets file sont des abstractions (on parle de fichier logique) permettant de désigner commodément ces fichiers “physiques”.

L'argument mode n'est pas spécifié. Python utilise ici le mode par défaut, le mode lecture.

Dans ce deuxième exemple, on utilise la méthode readline qui lit la ligne courante. Cela signifie que Python garde en mémoire le numéro de la dernière ligne lue.

Dans ce troisième exemple, on utilise la méthode readlines qui renvoie un objet list contenant toutes les lignes du fichier lu. On peut ensuite parcourir cet objet list de façon classique. Nous constatons au passage que chaque ligne est terminée par les caractères de fin de ligne “\n” (antislash n).

Dans ce dernier exemple, on effectue une lecture séquentielle des lignes grâce à une boucle for.

---
s7

Voyons à présent comment écrire dans un fichier. Dans les modes écriture ou ajout, le fichier sera créé s'il n'existe pas.

Commençons par le mode écriture.

Dans l'éditeur interactif IDLE, ouvrez un fichier Python et sauvez-le par exemple sous le nom test_ecriture.py

Reproduisez les lignes suivantes et exécutez ce fichier. Que fait Python ? Deux choses
- la création d'un objet de type file qui représente un fichier physique test.txt qui vient d'être créé
- la création d'une variable test qui référence l'objet file

Comme premier exercice, je vous propose d'écrire les deux premières lignes d'un poème de Blaise Cendrars dans le fichier grâce à la méthode write dont l'argument est une chaîne de caractères. Si l'on exécute ce programme, on constate que Python a bien écrit dans le fichier, mais sans retour à la ligne.
On peut introduire ce retour à la ligne en terminant les deux chaînes de caractères par le caractère de fin de ligne "\n" (antislash n).

Si l'on réexécute le programme, on constate que le contenu du fichier précédent est écrasé.

---
s8

Voyons le mode ajout.

Dans l'éditeur interactif IDLE, ouvrez un fichier Python et sauvez-le par exemple sous le nom test_ajout.py

Reproduisez les lignes suivantes et exécutez ce fichier. Si l'on exécute ce programme, on constate que Python a bien écrit dans le fichier, mais à la suite du texte existant.

Si l'on réexécute le programme, on constate que la nouvelle ligne est écrite deux fois.

---
s9

Nous vous proposons maintenant de passer à la dernière partie de ce cours qui sera l’occasion d’introduire quelques compléments utiles.