Initiation à Python - leçon 3.2.5

s1
Dans cette séquence, nous allons découvrir la notion de module.

---
s2

Comme nous l'avons vu, Python dispose d'un nombre limité de commandes de base. Cependant, il est possible d'ajouter des fonctions sous la forme de modules. On utilisera le terme importer.

Un module est grossièrement un bout de code que l'on a enfermé dans un fichier. On em prisonne ainsi des fonctions et des variables ayant toutes un rapport entre elles. Ainsi, si l'on veut travailler avec les fonctionnalités prévues par le module (celles qui ont été enfermées dans le module), il n'y a qu'à importer le module selon la syntaxe suivante et utiliser ensuite toutes les fonctions et variables prévues.

Il existe un grand nombre de modules disponibles avec Python sans qu'il soit nécessaire d'installer des bibliothèques supplémentaires. Pour cette partie, nous prendrons l'exemple du module math qui contient, comme son nom l'indique, des fonctions mathématiques. Inutile de vous inquiéter, nous n'allons pas nous attarder sur le module lui-même pour coder une calculatrice scientifique, nous verrons surtout les différentes méthodes d'importation.

Ensuite, nous créerons nous-mêmes un module.

---
s3

Lorsque vous ouvrez l'interpréteur Python, les fonctionnalités du module math ne sont pas incluses. Il s'agit en effet d'un module, il vous appartient de l'importer si vous vous dites "tiens, mon programme risque d'avoir besoin de fonctions mathématiques". Nous allons voir une première syntaxe d'importation.

La syntaxe est facile à retenir : le mot-clé import, qui signifie "importer" en anglais, suivi du nom du module, ici math.
Après l'exécution de cette instruction, rien ne se passe en apparence. En réalité, Python vient d'importer le module math. Toutes les fonctions mathématiques contenues dans ce module sont maintenant accessibles. Pour appeler une fonction du module, il faut taper le nom du module suivi d'un point "." puis du nom de la fonction. C'est la même syntaxe pour appeler des variables du module.
Pourquoi ce point ?

Parce que la fonction import crée dans la mémoire un objet module. Rappelez-vous qu'en Python tout est objet. Les variables et les fonctions définies dans ce module sont les attributs et les méthodes de ce nouvel objet.

Voyons ça sur un exemple. Ici on utilise l'attribut pi de l'objet math. Dans cet autre exemple, on utilise la méthode sqrt() de l'objet math qui prend comme argument un entier ou un flottant et retourne un flottant.

Je vous propose de reproduire cet exemple.

Du point de vue des variables, quand on importe le module math, Python crée une variable math et un espace de noms dénommé "math". Quand on tape math.sqrt(), on précise à Python que l'on souhaite exécuter la variable sqrt contenue dans l'espace de noms math et qui est associée à un objet fonction.

La conséquence de l'existence de ce nouvel espace de nommage est importante : on peut créer, dans l'espace de noms principal, une autre fonction sqrt. Il n'y aura pas de conflit entre la fonction ainsi créée (et qui sera appelée grâce à l'instruction sqrt() et la fonction sqrt() du module math.

Dans certains cas, vous pourrez vouloir changer le nom de l'espace de noms dans lequel sera stocké le module importé.

Voici un exemple que je vous propose de reproduire.

---
s4

Il existe une autre méthode d'importation qui ne fonctionne pas tout à fait de la même façon. En fonction du résultat attendu, on utilise indifféremment l'une ou l'autre de ces méthodes. Reprenons notre exemple du module math. Supposons que nous ayons uniquement besoin, dans notre programme, de la fonction renvoyant la valeur absolue d'une variable. Dans ce cas, nous n'allons importer que la fonction, au lieu d'importer tout le module.
Voici la syntaxe.

Je vous propose de reproduire cet exemple.

Vous aurez remarqué qu'on ne met plus le préfixe math devant le nom de la fonction. Du point de vue des variables, Python crée une variable fabs dans l'espace de noms principal (c'est une variable qui référence un objet fonction) - c'est à dire au même niveau que toutes les autres fonctions telles que print, range, etc.. Ici, il n'y a plus d'espace de noms dénommé "math".

On peut appeler toutes les variables et fonctions d'un module en tapant * à la place du nom de la fonction à importer. Attention dans ce cas qu'il n'existe pas dans le module une fonction portant le nom d'une fonction déjà existante dans l'espace de noms principal ! Nous verrons un exemple plus loin.

Une dernière chose générale sur l'import d'un module. Si l'on utilise Python avec l'interpréteur et qu'on réinitialise celui-ci, toutes les variables sont effacées.

---

s5

Voyons comment créer un module personnel.

Prenons un exemple.

Je vous propose d'ouvrir un nouveau fichier contenant la définition de deux fonctions f1 et g1 et de sauver ce fichier dans un répertoire de votre ordinateur, par exemple sous le nom mesFonctionsUsuelles.py

Ouvrez un autre fichier et sauvez ce fichier dans le même répertoire que le répertoire précédent, par exemple sous le nom testModule.py.

Que fait ce code ? On importe le module mesFonctionsUsuelles (qui se trouve au même niveau que le fichier courant testModule.py. On donne à la variable référençant l'objet module, un nom plus pratique ; on définit une fonction f2 qui utilise les fonctions f1 et g1 du module mesFonctionsUsuelles.

Testons ce fichier testModule. Attention bien entendu à ce que ce soit le fichier testModule.py qui soit actif lorsque vous choisissez le menu Run > Run Module.

C'est à vous !

---

s6

Examinons la seconde façon d'importer un module.

On conserve tel quel le module mesFonctionsUsuelles et on crée un autre fichier testModule2.py avec le code suivant dans lequel on importe l'ensemble des fonctions du module.

Pas de souci, bien entendu… tant que l'on ne s'amuse pas à définir dans le module principal par exemple une fonction g1 qui ici renvoie une expression différente de la fonction g1 du module mesFonctionsUsuelles. C'est un moindre mal, car elle pourrait renvoyer tout autre chose qu'un flottant, ce qui pourrait provoquer une erreur dans le programme. Attention donc à ces problèmes de renommage inopiné de fonctions provenant de modules que l'on a importé en bloc.

A vous de tester ces différents exercices.

---
s7

Un répertoire rassemblant plusieurs modules définit ce qu'on appelle un package.
Un package contient donc des fichiers Python et des sous-packages.

Nous allons voir comment la commande "import" permet de naviguer dans une hiérarchie de packages.

Quand on importe un module, Python doit pouvoir trouver le fichier .py correspondant. Pour cela, il cherche d'abord dans le "répertoire courant". Qu'est-ce que le répertoire courant ? Le répertoire courant c'est le répertoire du fichier que l'on est en train d'exécuter lorsque l'on sélectionne le menu Run > Run Module dans IDLE.

Dans l'exemple présenté, si on ouvre le fichier testModule3.py et si on l'exécute par le menu Run > Run Module, Python considère que le répertoire courant est le répertoire exercicesPython. L'instruction import mesFonctionsUsuelles va trouver le fichier mesFonctionsUsuelles.py qui se trouve au même niveau que le fichier testModule3.py

Si Python ne trouve pas le fichier dans le répertoire courant, il utilise une liste (appelée Search Path) indiquant les dossiers dans lesquels il doit effectuer cette recherche : Python parcourt ces dossiers dans l'ordre où ils figurent dans cette liste, et il charge le module spécifié dès qu'il le trouve.
C'est dans cette liste que l'on trouvera la localisation des packages tels que math. Notons que quand on lance un module depuis le menu Run > Run Module de l'application IDLE, le dossier où se trouve ce module est ajouté en tête du Search Path, et il devient le nouveau "répertoire courant" et devient donc le premier dossier à être exploré par Python quand il est à la recherche du code source d'un module.

Attention toute réinitialisation de IDLE supprime ce précédent répertoire courant. Une façon de s'en convaincre et de réinitialiser et de faire un test d'import depuis l'éditeur IDLE après avoir réinitialisé par Shell > Restart.

Que se passe-t-il si dans ce répertoire courant les modules sont placés dans un dossier, par exemple le dossier monPackage. La réponse est simple, il faut changer le répertoire courant au profit de ce nouveau répertoire. Ceci se fait grâce à la méthode chdir() du module os qu'il faut préalablement importer. On donne simplement le nom du répertoire.

Je vous propose de reproduire l'architecture de fichiers présentée ici et le code ci-dessous.

Que se passe-t-il si l'on souhaite utiliser des packages situés ailleurs ?
C'est le cas si l'on a développé des packages génériques qui servent à plusieurs applications.
On peut à nouveau changer le répertoire courant en donnant, éventuellement cette fois, le chemin absolu vers ce répertoire, comme dans cet exemple.

On peut changer le Search Path. Il existe des solutions pour modifier le Search Path de Python, d'une façon temporaire (jusqu'au prochain démarrage de l'interpréteur) ou permanente (notamment en modifiant la variable d'environnement PYTHONPATH ou en utilisant des fichiers avec extension pth, mais ceci dépasse le cadre de cette introduction à Python.

---
s8

Dans la leçon suivante, 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.