	**Langages et programmation**	**NSI T^{ale}**
	Modularité	Cours/TD

*

1. Introduction

Dans cette nouvelle partie, nous allons découvrir une autre facette du langage Python qui en fait un langage à la fois très puissant, modulable et évolutif : l'utilisation de modules. Nous allons notamment étudier le fonctionnement de quelques modules prédéfinis qu'il convient de savoir manipuler.

2. Définition et cas d'utilisation des modules Python

On appelle "module" tout fichier constitué de code Python (c'est-à-dire tout fichier avec l'extension .py) importé dans un autre fichier ou script.

Les modules permettent la séparation et donc une meilleure organisation du code. En effet, il et courant dans un projet de découper son code en différents fichiers qui vont contenir des parties cohérentes du programme final pour faciliter la compréhension générale du code, la maintenance et le travail d'équipe si on travaille à plusieurs sur le projet.

En Python, on peut distinguer trois grandes catégories de module en les classant selon leur éditeur :

- Les modules standards qui ne font pas partie du langage en soi mais sont intégrés automatiquement par Python ;
- Les modules développés par des développeurs externes qu'on va pouvoir utiliser ;
- Les modules qu'on va développer nous mêmes.

Dans tous les cas, la procédure à suivre pour utiliser un module sera la même. Nous allons commencer par décrire cette procédure puis nous étudierons dans la suite de cette partie quelques modules standards qu'il convient de connaître.

3. Importer un module

Un programme Python va généralement être composé d'un script principal qui va importer différents modules (c'est-à-dire différents fichiers Python) pour pouvoir les utiliser.

Pour importer un module, on utilise la syntaxe `import nom-de-mon-module`. Pour utiliser les éléments du module dans notre script, il faudra préfixer le nom de ces éléments par le nom du module et un point. Cela permet d'éviter les conflits dans le cas où on aurait défini des éléments de même nom que ceux disponibles dans le module.

Pour comprendre comment cela fonctionne en pratique, je vous invite à ouvrir votre éditeur de texte et à créer un fichier qu'on va appeler `bonjour`. Notre fichier va contenir une variable et une fonction comme ceci :

bonjour.py

```
1   nom = "Pierre"
2
3   def disBonjour():
4       print(nom, " vous dit bonjour")
```

Ensuite, enregistrez le dans le dossier Python que vous devriez avoir créé au début de ce cours et qui devrait se trouver sur votre bureau.

Dès que tout cela est fait, on peut retourner dans le terminal ou l'invite de commande. On peut ensuite lancer l'interpréteur Python. Ici, il faut imaginer que notre terminal / invite de commande représente le script principal de notre programme. On va importer notre module dans ce script principal grâce à l'instruction `import bonjour`.

Lorsque l'interprète rencontre une instruction `import`, il importe le module s'il est présent dans le `path` (le chemin de recherche). Pour rappel, le `path` ou chemin de recherche est une liste de répertoires dans lesquels l'interpréteur cherche avant d'importer un module. Pour être tout à fait précis, lorsqu'on importe un module, l'interpréteur Python le recherche dans différents répertoires selon l'ordre suivant :

1. Le répertoire courant ;
2. Si le module est introuvable, Python recherche ensuite chaque répertoire listé dans la variable shell `PYTHONPATH` ;
3. Si tout échoue, Python vérifie le chemin par défaut. Sous UNIX, ce chemin par défaut est normalement `/usr/local/lib/python/`.

Dès que notre module est importé, on va pouvoir accéder tout simplement aux variables et fonctions de notre module depuis notre script en préfixant le nom des éléments du module par le nom du module et un point comme ceci :

Python — Python — 99×24

```
Last login: Tue Jul  2 14:12:29 on ttys000
You have mail.
Pierres-MacBook-Pro:~ pierre$ cd Desktop/Python
Pierres-MacBook-Pro:Python pierre$ python3.7
Python 3.7.3 (v3.7.3:ef4ec6ed12, Mar 25 2019, 16:52:21)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
>>> import bonjour
>>> bonjour.disBonjour()
Pierre  vous dit bonjour
>>> bonjour.nom
'Pierre'
```

4. *Créer un alias de nom pour un module*

On peut utiliser le mot clef `as` pour renommer un module ou plutôt pour créer un alias de nom. Cela peut permettre d'obtenir des scripts plus courts et plus clairs dans le cas où le nom du module est inutilement long ou s'il est peu descriptif.

```
>>> import bonjour as b
>>> b.disBonjour()
Pierre  vous dit bonjour
```

Note : on ne peut importer un module qu'une fois dans un script. Si vous testez le code ci-dessus et si vous aviez déjà importé le module précédemment, il faudra que vous quittiez l'interpréteur et que vous le relanciez pour que tout fonctionne. Pour quitter l'interpréteur, vous pouvez utiliser l'instruction `quit()`.

5. *Importer uniquement certains éléments d'un module*

Parfois, nous n'aurons besoin que de certains éléments précis dans un module. On va alors pouvoir se contenter d'importer ces éléments en particulier. Pou cela, on va utiliser l'instruction
`from nom-du-module import un-element`.

On va par exemple pouvoir choisir de n'importer que la variable `nom` ou que la fonction `disBonjour()` depuis le module `bonjour.py`.

Dans le cas où on n'importe que certains éléments depuis un module, il ne faudra pas ensuite préfixer le nom des éléments par le nom du module pour les utiliser dans notre script principal.

```
>>> from bonjour import nom
>>> nom
'Pierre'
>>> disBonjour()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'disBonjour' is not defined
>>> bonjour.nom
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'bonjour' is not defined
```

6. *Obtenir la liste des éléments d'un module*

Souvent, on importera des modules standards ou créés par d'autres développeurs. Dans ce cas là, il peut être intéressant d'obtenir rapidement la liste des éléments du module afin de voir rapidement ce qui va pouvoir nous être utile.

Pour faire cela, on peut utiliser la fonction `dir()` qui renvoie la liste de toutes les fonctions et variables d'un module.

```
>>> import bonjour
>>> dir(bonjour)
['__builtins__', '__cached__', '__doc__', '__file__', '__loader__', '__name__', '__package__', '__s
pec__', 'disBonjour', 'nom']
```

Comme vous pouvez le voir, tout fichier Python possède par défaut des éléments de configuration.

7. *Les modules Python standards*

Comme annoncé au début de cette leçon, nous importerons bien souvent des modules créés par d'autres développeurs ou des modules mis à notre disposition par Python lui même.

En effet, il existe un grand nombre de modules préconçus et prêts à l'emploi qui sont fournis d'office avec Python. Ces modules vont étendre le langage et nous permettre de réaliser toutes sortes d'opérations notamment grâce aux fonctions qu'ils nous fournissent.

Pour importer un module Python, nous allons à nouveau tout simplement utiliser une instruction `import` comme si on importait l'un de nos modules.

Les modules Python standards à connaître sont les suivants :

- Le module `cgi` ("Common Gateway Interface" ou "Interface de Passerelle Commune" en français) fournit des éléments permettant à des programmes Python de s'exécuter sur des serveurs HTTP ;
- Le module `datetime` fournit des classes pour manipuler de façon simple ou plus complexe des dates et des heures ;
- Le module `json` permet l'encodage et le décodage de données au format JSON ;
- Le module `math` fournit un ensemble de fonctions permettant de réaliser des calculs mathématiques complexes ;
- Le module `os` fournit une manière portable d'utiliser les fonctionnalités dépendantes du système d'exploitation ;
- Le module `pickle` permet de sérialiser des objets Python ;
- Le module `random` implémente des générateurs de nombres pseudo-aléatoires pour différentes distributions ;
- Le module `re` fournit des opérations sur les expressions rationnelles similaires à celles que l'on trouve dans Perl ;
- Le module `socket` fournit un accès à l'interface sockets qui correspond à un ensemble normalisé de fonctions de communication ;
- Le module `sys` fournit un accès à certaines variables système utilisées et maintenues par l'interpréteur, et à des fonctions interagissant fortement avec ce dernier ;
- Les modules `urllib.request` et `urllib.parse` permettent d'ouvrir, de lire et d'analyser des URLs.

Vous pouvez retrouver la liste complète des modules standards Python sur le site officiel.

8. *Les paquets Python*

Un paquet est tout simplement un ensemble de plusieurs modules regroupés entre eux. On va pouvoir importer des paquets de la même façon que des modules et accéder à un module ou à un élément en particulier en utilisant la syntaxe `nom-paquet.nom-module.nom-element`.

Source : https://www.pierre-giraud.com/python-apprendre-programmer-cours/module-paquet/