3.2 Modules

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.

3.2.1 Qu'est ce qu'un module ?

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.

Remarque 5 :

1. Les modules sont des programmes `Python` qui contiennent des fonctions que l'on est amené à réutiliser souvent *(on les appelle aussi **bibliothèques** ou **librairies**)*. Ce sont des « boîtes à outils » qui vont nous être très utiles.
2. Nous connaissons déjà un module que nous avons importé lors de la résolution d'une équation du second degré : le module `math`
3. Les modules permettent la séparation et donc une meilleure organisation du code. En effet, il est 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.

   Ainsi le module `math` contient-il les fonctions mathématiques
4. Il y a beaucoup de modules en `Python`. 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, comme le module `math` ;
   ⇒ Les modules développés par des développeurs externes qu'on va pouvoir utiliser, comme le module `matplotlib` ;
   ⇒ Les modules qu'on va développer nous mêmes.
5. Lorsque nous ouvrons l'interpréteur `Python`, les fonctionnalités du module `math` ne sont pas incluses ; il faudra les importer. Dans tous les cas, la procédure à suivre pour utiliser un mode 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 connaitre.

3.2.2 Importer un module : la méthode *import*

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.

Remarque 6 :

Préfixer la fonction par le nom du module permet d'éviter les conflits dans le cas où on aurait défini des éléments de même nom que ceux disponibles dans le module. On parle **d'espace de nommage**

Exemple 7 :

Nous allons procéder à des exemples

1. On importe le module `math`

   ```
   import math
   ```

   La syntaxe est facile à retenir : le mot-clé `import`, qui signifie « importer » en anglais, suivi du nom du module, ici, `math`

2. Après l'exécution de cette instruction, rien ne se passe....*en apparence*. En réalité, `Python` vient d'importer tout le module `math`. Toutes les fonctions mathématiques contenues dans le 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. Voyons un exemple :

```
>>> math.sqrt(16)
4.0
>>> math.sqrt(20)
4.47213595499958
>>>
```

   La fonction `sqrt` du module `math` renvoie la racine carrée du nombre inséré en argument

Remarque 7 :

1. Comment connaître toutes les fonctions existantes dans le module `math` ?
   C'est une fonction, la fonction `help` qui nous donnera l'information

```
help (math)
```

   Ou encore aller chercher la documentation sur le net dans la documentation Python
2. Il est aussi possible de se documenter sur une fonction bien précise en utilisant la fonction `help`

```
>>> help(math.sqrt)
Help on built-in function sqrt in module math:

sqrt(x, /)
    Return the square root of x.

>>>
```

3.2.3 Espace de nom

Un espace de noms est un système permettant de s'assurer que tous les noms d'un programme sont uniques et peuvent être utilisés sans aucun conflit.
Il s'agit de regrouper certaines fonctions ou variables sous un préfixe spécifique. Les espaces de noms assurent que chaque nom utilisé dans un programme est unique.

Exemple 8 :

1. En vérité, lorsque vous tapez `import math`, cela crée **un espace de noms dénommé « math »**, contenant les variables et fonctions du module `math`.
   Quand vous tapez `math.sqrt(25)`, vous précisez à `Python` que vous souhaitez exécuter la fonction `sqrt` corrtenue dans 1'espace do noms `math`.
   Cela signifie que vous pouvez avoir, dans l'espace de noms principal, une autre fonction `sqrt` que vous avez définie vous-même.
   Il n'y aura donc pas de conflit entre, d'une part, la fonction `sqrt` que nous avons créée et que vous appellerez grâce à l'instruction `sqrt` et, d'autre part, la fonction `sqrt` du module `math` que vous apellerez grâce à l'instruction `math.sqrt`
2. **Un exemple concret**
   On considère le (tout) petit script `Python` ci-après

```
import math
a = 5
b= 3
```

   Dans l'espace de nom principal, celui que l'on utilise depuis le début et qui ne nécessite pas de préfixe, on trouve :

⇒ La variable `a`
⇒ La variable `b`
⇒ Le module `math` qui se trouve dans un espace de noms également appelé `math` dans lequel on trouve
  ⋆ La fonction `sqrt`
  ⋆ La variable `pi`
  ⋆ Et bien d'autres fonctions et variables

3. L'intérêt des modules est de stocker à part variables et fonctions, dans un espace de nom, sans risque de conflit avec les variables ou fonctions que nous avons créées.
   Dans certains cas, nous pourrons modifier le nom de l'espace dans lequel le module importé sera stocké.

3.2.4 Créer un alias de nom pour un module

On peut utiliser le mot clef as pour renommer un module et créer un alias de nom.

Exemple 9 :

1. ```
   import math as mathematiques
   ```

   En faisant cet alias, nous importons le module `math` en spécifiant à `Python` de **l'héberger dans un espace de noms** appelé `mathematiques` au lieu de `math`. On peut penser que cela pourra permettre de mieux contrôler les espaces de noms des modules que nous importons.

2. ```
   >>> import random as rand
   >>> rand.randint(1, 10)
   6
   >>> rand.uniform(1, 3)
   2.643472616544236
   ```

   Dans cet exemple, les fonctions du module `random` sont accessibles via l'alias `rand`.

Remarque 8 :

1. 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.
2. 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.

3.2.5 Importer uniquement certains éléments d'un module

Si nous ne voulons n'importer que certains éléments dans un module, nous utilisons, pour cela l'instruction from nom-du-module import un-element.

Exemple 10 :

```
>>> from math import fabs
>>> fabs(-10.5)
10.5
>>> fabs(45)
45.0
>>>
```

Dans cet exemple, nous n'avons importé que la fonction `fabs` qui est la fonction valeur absolue.

Remarque 9 :

1. Lorsque nous importons uniquement une fonction, vous remarquez que nous ne mettons pas le préfixe `math.`
   En fait, par cette méthode, on met la fonction `fabs` dans l'espace de nom principal, au même niveau que la fonction `print` ou les variables que nous aurons créées.
2. Par la méthode :

   ```
   >>> from math import *
   >>> fabs(-10.5)
   10.5
   >>> sqrt(49)
   7.0
   >>>
   ```

   nous importons **toutes les fonctions** du module `math` dans l'espace de nommage principal

Exemple 11 :

Dans le script ci-après, nous importons 2 modules : les modules `math` et `random`.
Nous faisons calculer, dans ce script l'aire d'un cercle de rayon R où R est un entier défini de manière aléatoire entre 1 et 10

```
#-*-coding:Latin-1 -*



#algo Fonction Aire d'un cercle
#Exemple d'importation de modules

import math
import random

R = random.randint(1,10) #Fonction randint du domaine de nommage random
                         #Nous prenons un cercle de rayon un nombre entier aléatoire
                         #pris entre 1 et 10
S = math.pi *(R**2)

print("Le rayon est R= ",R,"et la surface est S= ",S)
```

Nous obtenons comme résultat :

```
Le rayon est R=  10 et la surface est S=  314.1592653589793
>>>
Le rayon est R=  3 et la surface est S=  28.274333882308138
>>>
Le rayon est R=  6 et la surface est S=  113.09733552923255
>>>
```

3.2.6 Créer son propre module

Il est tout à fait possible de créer ses propres modules. Comment ?

1. On crée un fichier, par exemple `module.py` dans lequel nous mettons une ou deux fonctions
2. Dans un autre fichier exécutable, par exemple `test.py`, nous importons, ou tout le module, ou un fonction issue du module

Exemple 12 :

Exemple par un exercice résolu

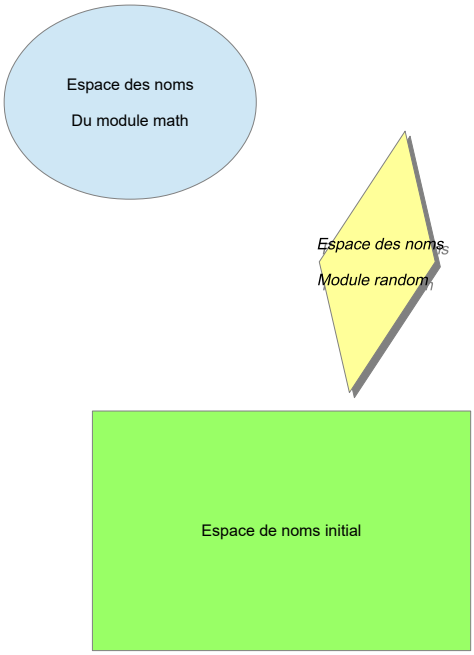

FIGURE 3.1 – Une modélisation des espaces de noms

Ecrire l'algorithme de calcul de maximum d'un ensemble de Max notes, le nombre de notes étant donné par l'opérateur. Le programme principal appelle une fonction qui se charge de vérifier la validité de la note (note comprise entre 0 et 20).

1. Nous commençons par créer une fonction et « l'enfermer » dans un module, c'est à dire un fichier que nous avons appelé `Essai_Module`

```
#-*-coding:Latin-1 -*

#algo controle de valeur

def  controle(x):
    f= (x>=0) and (x<= 20)
    return f
```

`f` est une variable booléenne qui n'a que 2 valeurs `True` ou `False`
Il est clair que nous pourrions réutiliser ce module pour un autre script `Python`. On appelle cela une **factorisation** du code.

2. Dans un second temps, nous écrivons le programme principal dans lequel nous importons le module nouvellement créé en lui donnant un alias.

```
#-*-coding:Latin-1 -*

#algo Note Maximale Entre 0 et 20
#Cet algorithme donne la meilleure des Max notes comprises entre 0 et 20

import Essai_Module as controle

Max =int(input("Combien de notes? "))
note= int(input("Donner une note entière entre 0 et 20 "))
while controle.controle(note) == False : #on utilise une fonction du module
  print("Vous avez fait une erreur")
```

```
    note= int(input("On recommence: donner une note entière entre 0 et 20 "))
maximum =note
compteur = 1
while compteur < Max :
    note= int(input("Donner une note "))
    while controle.controle(note) == False : #on utilise une nouvelle fois
                                             # une fonction du module
        print("Vous avez fait une erreur")
        note= int(input("Donner une note entre 0 et 20 "))
    if note>=maximum :
        maximum=note
    compteur += 1

print("La note maximale est ",maximum)
```

3.2.7 Les packages

Les packages sont des dossiers
Dans ces dossiers, on peut y trouver des fichiers (ou modules) ou d'autres dossiers (des packages)
C'est donc un mode de regroupement de modules