

5 – Les modules Python

Certaines fonctions Python sont déjà écrites et regroupées dans des modules. Elles nécessitent d'être importées avant l'utilisation.

Pour utiliser le module *nom_module*, on ajoute en début de programme l'instruction :

```
from nom_module import *
```

Le module `math` permet d'utiliser entre autres, les fonctions `sin()`, `cos()`, la constante `pi`...

Exemple : `from math import *` ou `import math`

Ceci fonctionne si nous voulons importer tout le contenu du module, mais comment faire si nous ne voulons pas tout importer ?

Il nous est possible d'importer que ce que nous avons besoin d'utiliser, par exemple :

```
from math import pi
```

Cet import est un peu spécial, car nous importons une variable, et non une fonction. Ainsi, `print(pi)` affichera 3,1415...

```
from math import sqrt
```

Ici nous importons la fonction racine carrée. Pour l'utiliser nous pourrions par exemple faire :

```
print(sqrt(4))
```

Qui affichera 2 comme résultat.

Il est aussi possible de préfixer nos import. Ceci sera utile si le module que nous voulons importer contient plusieurs fonctions. Imaginons que nous ayons besoin du module `random`, nous allons l'appeler via :

```
import random
```

Comme ce module contient plusieurs fonction, afin d'avoir la fonction voulu il va falloir utiliser la syntaxe suivante :

```
random.randint()
```

Il est possible de préfixer nos imports afin de simplifier l'écriture et l'appel de nos imports, ainsi :

```
import random
```

deviendra

```
import random as rd
```

Puis il nous suffira d'appeler random en utilisant son préfixe, càd :

```
rd.randint()
```

Nous avons vu qu'il y avait différentes façon d'importer des modules.

From ... import * pourrait sembler une bonne idée, car nous importons tout puis n'utilisons que ce dont nous avons besoin, mais cela pourrait poser problèmes lors de la relecture (afin de modifier ou corriger le code), ou si un autre développeur souhaite jeter un oeil à votre code, car il sera quasiment impossible de savoir quelle module fournit quelle fonction. Cette façon d'importer est à bannir.

Il est donc préférable d'utiliser plutôt la notation où l'on importe le module puis l'on définit la fonction que l'on souhaite utiliser, par exemple :

```
from random import randint
```

Cela permet une meilleure visibilité et lisibilité du code, et permet de garder une trace des imports.

Pour aller plus loin :

Explication trouvée sur Developpez.net :

"Dans le cadre de *import time* tu dois faire *time.time()* (ou *time.sleep(1)*) pour accéder à ta méthode.

Si tu fais *from time import ** tu intègres les méthodes de time. Tu dois donc juste faire *time()* ou *sleep(1)*

La première méthode est la plus propre car si tu as deux imports qui comportent des méthodes avec le même nom celle importée en dernier écrase l'autre. Il me semble par exemple que Tkinter et pylab comportent tous deux des méthodes Button.

Donc si je fais

```
from Tkinter import *  
from pyvar import *
```

C'est le deuxième qui écrase le premier (Tkinter) et ils ne sont pas identiques... Imagine le résultat.

Avec *import Tkinter* cela s'écrit *Tk.Button*

Il est aussi possible d'écrire *import time as tm* (tm pour l'exemple, tu mets ce que tu veux) et donc d'utiliser *tm.time/tm.sleep* ou de n'importer que certaines méthodes *from time import sleep, time* et là c'est *sleep(1)*

Dans l'absolu *import <tonimport>* est donc plus propre."

Ici on peut voir que si 2 modules ont des fonctions qui possèdent le même nom, le dernier module appelé écrasera les autres.

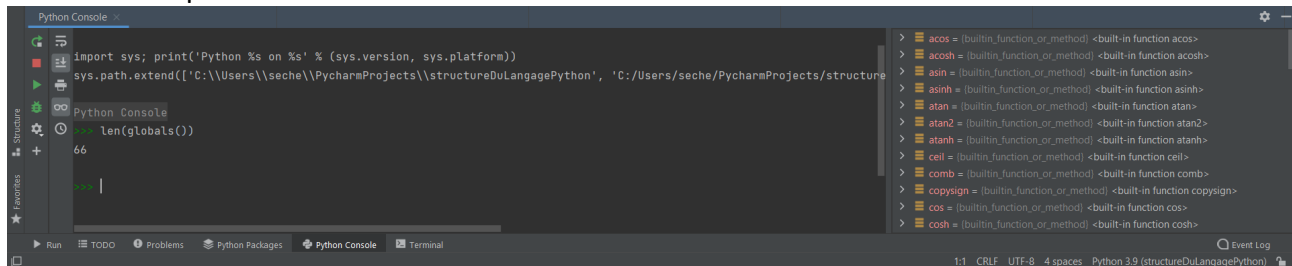
Une 2° explication trouvée sur un autre site (les 2 explications se complètent).

Même si, par exemple, `from math import *` est à bannir, il est tout à fait correct d'utiliser `import math`, je dirais même que c'est plus propre.

Tout ce passe au niveau de ce qui est importé et la façon dont c'est importé, dans les globals().

Différence entre *from math import ** et *import math* : (en console)

`from math import *` :

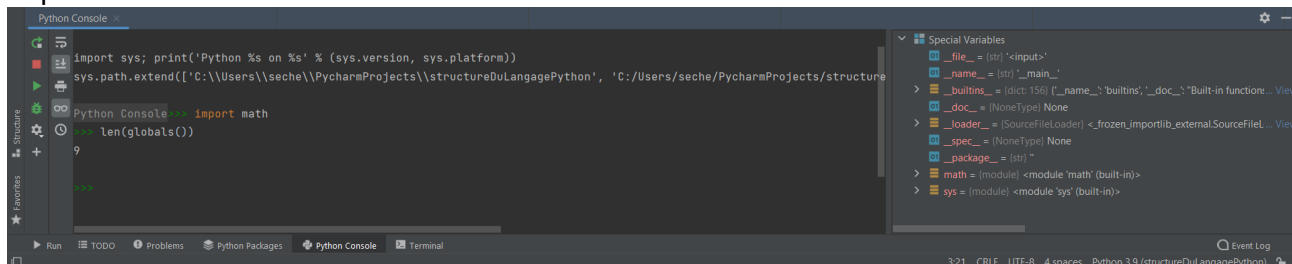


The screenshot shows a Python IDE with a console window. The code executed is:

```
import sys; print('Python %s on %s' % (sys.version, sys.platform))
sys.path.extend(['C:\\Users\\seche\\PycharmProjects\\structureDuLangagePython', 'C:/Users/seche/PycharmProjects/structureDuLangagePython'])
len(globals())
```

The output of `len(globals())` is 66. On the right side of the IDE, a list of imported functions is displayed, including `acos`, `acosh`, `asin`, `asinh`, `atan`, `atan2`, `atanh`, `ceil`, `comb`, `copysign`, `cos`, and `cosh`.

`import math` :



The screenshot shows a Python IDE with a console window. The code executed is:

```
import sys; print('Python %s on %s' % (sys.version, sys.platform))
sys.path.extend(['C:\\Users\\seche\\PycharmProjects\\structureDuLangagePython', 'C:/Users/seche/PycharmProjects/structureDuLangagePython'])
import math
len(globals())
```

The output of `len(globals())` is 9. On the right side of the IDE, a list of special variables is displayed, including `_file_`, `_name_`, `_builtins_`, `_loader_`, `_spec_`, `_package_`, `math`, and `sys`.

Ici, tout ce passe dans le namespace de Python (comme cela dépasse le cadre de la 1° NSI, nous n'irons pas plus loin, mais je vous encourage à vous renseigner dessus si cela vous interesse, ou tout simplement si cela pique votre curiosité).