I./ Les modules en Python :

Nous avons déjà utilisé des *modules* (ou *bibliothèques*) l'année précédente, comme le module **`math`** ou le module **`random`**. Un module Python est un ensemble de fonctions que l'on peut ajouter à son script Python. Afin de pouvoir utiliser les fonctions présentes dans un module particulier, il faut les *importer*.

Il y a plusieurs façons de le faire :

1ère méthode : **`import math`**

On importe toutes les fonctions de la bibliothèque **`math`** et on doit utiliser le préfixe **`math.`**
Exemple d'utilisation : **`a = math.sqrt(2)`**

2ème méthode : **`from math import sqrt, log, sin`**

Ici on importe seulement quelques fonctions. On n'a pas besoin d'utiliser le préfixe **`math.`**
Exemple d'utilisation : **`a = sqrt(2)`**

3ème méthode : **`from math import *`**

On importe toutes les fonctions de la bibliothèque **`math`** et on n'a pas besoin du préfixe **`math.`**
Exemple d'utilisation : **`a = sqrt(2)`**

La première méthode est la meilleure, et c'est celle qui est recommandée. En effet elle permet de savoir dans quel module se trouve une fonction particulière, et elle permet aussi d'éviter l'écrasement de fonctions homonymes. Par exemple les modules **`math`** et **`cmath`** contiennent tous les deux la fonction **`cos()`**.

Si on importe ces deux modules comme suit :

```
import math
import cmath
```

On pourra utiliser les deux fonctions en faisant : **`math.cos(x)`** et **`cmath.cos(x)`**. Si on a de plus définit une fonction **`cos()`** dans notre script, on pourra l'appeler simplement en faisant : **`cos(x)`**. Il n'y a alors aucune ambiguïté.

Si on utilise la méthode suivante :

```
from math import *
from cmath import *
```

Il y a ici une ambigüité lorsqu'on va appeler **`cos(x)`**. Est-ce la fonction du module **`math`**, ou celle du module **`cmath`** ? Ce sera en général celle du module importé en dernier. Si de plus on a définit une fonction **`cos()`** dans notre script, ce sera cette fonction qui prendra le dessus sur les autres.

Il faut donc bien faire attention, et les syntaxes « **`from`** … **`import`** … » et « **`from`** … **`import *`** » ne doivent être utilisées que lorsqu'il n'y a aucune confusion possible.

II./ Comment créer son propre module Python :

Tapez le programme suivant, et sauvegardez le sous le nom « **`salutations.py`** » dans le répertoire de votre choix :

```
# Module "salutations"
# Ce module propose plusieurs fonctions permettant de saluer l'utilisateur

def bonjour(nom):
    """
    Cette fonction affiche le message "Bonjour " suivi de la chaîne de
    caractères passée en argument.
```

```
    """
    print("Bonjour " + nom)
    return None

def salut(nom):
    """
    Cette fonction affiche le message "Salut " suivi de la chaîne de
    caractères passée en argument.
    """
    print("Salut " + nom)
    return None

def hello(nom):
    """
    Cette fonction affiche le message "Hello " suivi de la chaîne de
    caractères passée en argument.
    """
    print("Hello " + nom)
    return None
```

IMPORTANT : Vous DEVEZ taper le script en entier, y compris les commentaires. Nous verrons qu'ils sont loin d'être inutiles.

Tapez et enregistrez dans le même répertoire le fichier « **`test.py`** » suivant :

```
from salutations import *
bonjour("Maître")
salut("copain")
hello("my friend")
```

On vérifie que l'importation a rajouté trois nouvelles fonctions. Tapez maintenant dans la console Python les lignes suivantes :

```
help(bonjour)
help(salut)
help(hello)
```

Que remarque-t-on ? Les lignes de commentaires situées juste après le nom de la fonction et entourées par des triples guillemets sont appelées *docstrings*. Les docstrings sont donc des chaînes de caractères permettant de documenter la fonction. Elles sont affichées grâce à la commande **`help()`**. Les commandes Python et les fonctions des modules standards ont généralement aussi une docstring. Tapez dans la console Python :

```
help(print)
```

Puis faire :

```
import math
help(math.cos)
```

Programmation Python : créer un module chiffrement qui contient les fonctions de chiffrement de César et de Vigenère vues précédemment.

III./ Utiliser un module Python :

Vous devez être capable d'utiliser un module Python quelconque, en allant lire sa documentation. Les deux activités suivantes vous permettront de découvrir deux modules assez intéressants :

1./ Utilisation du module `pyplot` de la librairie `matplotlib` :

Un tutoriel peut être trouvé à l'adresse : https://matplotlib.org/1.2.1/users/pyplot_tutorial.html.

La première ligne du premier script de la page peut dérouter : « **`import matplotlib.pyplot as plt`** ». Elle définit un *alias*. On importe **`matplotlib.pyplot`** mais pour ne pas devoir taper ce préfixe à chaque fois que l'on appelle une fonction, on le renomme en « **`plt`** ». Pour appeler la fonction « **`plot`** », par exemple, on fera donc « **`plt.plot`** » au lieu de « **`matplotlib.pyplot.plot`** ».

Remarque : on peut utiliser des alias avec n'importe quel module. On pourrait par exemple importer notre module **`salutations`** à l'aide de « **`import salutations as slt`** ».

Programmation Python : créer un programme qui tracera la représentation graphique de F_n en fonction de n, F_n étant le $n^{ième}$ terme de la suite de Fibonacci.

Programmation Python : créer un programme qui tracera la représentation graphique des deux fonctions $y_1 = \sin x$, $y_2 = \cos x$ sur un intervalle [a, b], les valeurs de a et de b étant données par l'utilisateur. On utilisera deux couleurs différentes pour les deux courbes.

2./ Utilisation du module `turtle` :

Des informations peuvent être trouvées ici : https://docs.python.org/fr/3/library/turtle.html.

Programmation Python : créer un programme qui demande à l'utilisateur un nombre n, puis qui va tracer un polygone régulier ayant n côtés.

Ceux qui voudront aller plus loin avec le module turtle pourront regarder le tutoriel disponible à l'adresse suivante : https://zestedesavoir.com/tutoriels/944/a-la-decouverte-de-turtle/.