Informatique PCSI
Prérequis TP 3 : modules

Il est conseillé de décomposer un programme en particulier à l'aide de fonctions. Le relecture en est facilitée et il est plus aisé de tester une à une les différentes parties. Lorsque des fonctions traitent toutes d'un même sujet, on peut les regrouper dans un fichier, un module. Différents modules peuvent être regroupés dans une bibliothèque. Un *module* peut donc être un fichier unique (extension py) qui contient des variables et des fonctions.

Modules

Création d'un module

Pour créer un module qui est un simple fichier, il suffit d'écrire les définitions de fonctions et de constantes dans un fichier dont le nom a l'extension `py`. Il est important de fournir une documentation qui indique la façon de les utiliser et ce qu'elles produisent.

Par exemple, on crée un fichier qui se nomme `aires.py`, fichier dans lequel on précise la valeur utilisée pour le nombre π et les définitions de quelques fonctions permettant de calculer des aires.

```
pi = 3.14159

def disque(rayon):
    """rayon de type float est le rayon d'un disque
    renvoie l'aire du disque"""
    return pi * rayon * rayon

def rectangle(largeur, longueur):
    """largeur et longueur sont du type float
    renvoie l'aire d'un rectangle de côtés largeur et longueur"""
    return largeur * longueur

def triangle(base, hauteur):
    """base et hauteur de type float sont la base et la hauteur
    d'un triangle
    renvoie l'aire du triangle"""
    return base * hauteur / 2
```

Importation d'un module

Attention, si le fichier `aires.py` est enregistré dans un dossier quelconque et qu'on écrit `import aires` dans un interpréteur Python, une erreur de type `ModuleNotFoundError` est alors affichée. L'interpréteur nous dit qu'il ne connaît aucun module se nommant `'aires'`. On pourrait préciser à l'interpréteur Python le chemin d'accès au fichier.

Si les fichiers sont enregistrés dans un même dossier, l'importation ne pose pas de problème.

```
import aires
print(aires.disque(5))
```

Le point entre `aires` et `disque` signifie que l'on se réfère au nom `disque` qui est défini dans le module `aires`. Lorsque le nom du module est long, on peut le renommer avec une instruction comme `import matplotlib.pyplot as plt`.

Une autre possibilité est d'écrire `from aires import disque`. Dans ce cas on peut écrire `disque(5)`. Au moment de l'importation, la fonction `disque` récupère la valeur de `pi` définie dans le fichier `aires.py` et peut l'utiliser. Mais nous n'avons pas accès à cette valeur de `pi`. La commande `print(pi)` entraîne une erreur de nom, `NameError: name 'pi' is not defined`.

On accède à la spécification d'une fonction avec la fonction `help`.

```
>>> import aires
>>> help(aires.disque)
Help on function disque in module aires:

disque(rayon)
    rayon de type float est le rayon d'un disque
    renvoie l'aire du disque
```

Exemples de modules et bibliothèques

Module math

La fonction `dir` nous permet d'explorer le contenu du module `math`.

L'instruction `import math` est obligatoire. Si elle n'est pas écrite, l'appel `dir(math)` provoque une erreur de nom : `"name 'math' is not defined"`.

Ce module contient des constantes comme π, noté `pi`, et des fonctions mathématiques. Nous pouvons obtenir des informations sur une fonction particulière à l'aide de la fonction `help`.

Dans la liste des fonctions, nous remarquons la fonction `hypot`. Qu'obtient-on avec `help` ?

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

hypot(...)
    hypot(x, y)

    Return the Euclidean distance, sqrt(x*x + y*y).
```

C'est bien la longueur de l'hypoténuse d'un triangle rectangle de petits côtés `x` et `y`.

Module random

Pour obtenir des informations sur un module, des documentations en ligne sont consultables, par exemple pour le module random : `https://docs.python.org/3/library/random.html`.

Comme précédemment, nous pouvons observer dans l'interpréteur le contenu de ce module avec les instructions `import random`, puis `dir(random)`.

Quelques fonctions qui renvoient des nombres de manière aléatoire sont souvent utilisées.

```
>>> help(random.randint)
Help on method randint in module random:

randint(a, b) method of random.Random instance
    Return random integer in range [a, b], including both end points.
```

La fonction `randint` renvoie un entier compris entre `a` et `b`. Nous sommes informés que les bornes `a` et `b` sont incluses alors que la borne `b` est exclue avec `randrange(a, b)`.

Une autre fonction souvent utilisée est `random` qui renvoie un nombre de type float compris entre 0 et 1, la borne 1 étant exclue.

Module Turtle

Le module `turtle` permet de dessiner en programmant. On commence par tout importer. Puis on utilise des instructions simples comme `forward(x)` pour avancer de `x` pixels, donc tracer un segment de longueur `x` pixels, `left(d)` pour tourner à gauche de `d` degrés.

```
from turtle import *

def triangle(pixel):
    for i in range(3):
        forward(pixel)
        left(120)

triangle(200)
```

Nous exécutons ce programme et obtenons dans la fenêtre de dessin un triangle équilatéral.

Voici quelques fonctions dont la signification est presque évidente :
`forward(distance)`, `backward(distance)`, pour avancer ou reculer ;
`left(angle)`, `right(angle)`, pour tourner à gauche ou à droite ;
`reset()`, pour effacer ;
`goto(x, y)`, `up()`, `down()`, pour aller à un point donné, lever ou baisser le crayon ;
`color('couleur')`, `width(épaisseur)`, pour la couleur et l'épaisseur du trait ;
`write('texte')`, pour afficher du texte ;
`begin_fill()`, `end_fill()`, pour remplir une figure fermée avec une couleur.

Le point central a pour coordonnées $(0;0)$. Le programme est écrit dans un fichier puis exécuté et on observe le résultat. L'utilisation de l'interpréteur est intéressante pour tester des instructions.

```
>>> from turtle import *
>>> forward(100)
>>> goto(-80,150)
```

Le résultat de chaque instruction est immédiatement visible sur la figure.

On trouve de nombreux exemples sur internet et une documentation est disponible à l'adresse : `https://docs.python.org/fr/3.8/library/turtle.html`.

Bibliothèques

Une bibliothèque est en général une collection de modules. Il en existe un très grand nombre en Python qui viennent compléter suivant les besoins la bibliothèque standard qui contient les modules `math` et `random` parmi beaucoup d'autres.

La bibliothèque **Matplotlib** en particulier est utilisée pour tout ce qui concerne les graphiques. Elle contient le module `pyplot` utilisé pour le tracé de courbes. Il est aussi possible de tracer des histogrammes et toutes sortes de représentations graphiques.

Une documentation précise et de très nombreux exemples d'utilisation se trouvent facilement par une recherche sur Internet.

Le principe pour tracer une courbe

Le tracé d'une courbe s'obtient avec des points dont on précise les coordonnées dans deux listes. Ce tracé nécessite donc la constitution de deux listes, la liste des abscisses et celle des ordonnées.

Voici une syntaxe minimale pour obtenir un tracé :

```
import matplotlib.pyplot as plt

def f(x):  # une fonction
    return x ** 2 - 5 * x + 1

x = [i * 0.01 for i in range(101)] # la liste des abscisses
y = [f(u) for u in x] # la liste des ordonnées

plt.plot(x, y) # construction de la courbe
plt.grid() # un quadrillage facultatif
plt.show() # affichage de la courbe
```

La fonction `plot` construit la courbe, la fonction `show` l'affiche à l'écran. Nous pouvons ainsi obtenir plusieurs courbes sur la même figure.

```
def f(x):  # une fonction
    return x ** 2 - 3 * x + 2

def g(x):  # une fonction
    return - x ** 2 + x + 1

x = [1 + i * 0.01 for i in range(101)]
y1 = [f(u) for u in x]
y2 = [g(u) for u in x]

plt.plot(x, y1) # construction de la courbe 1
plt.plot(x, y2) # construction de la courbe 2
plt.show() # affichage de la courbe
```

Pour avoir une idée de l'étendue des possibilités de représentation graphique, il suffit de regarder les exemples présentés à l'adresse `https://matplotlib.org/gallery/index.html`.