11.2 Modularité

NSI terminale - JB Duthoit

11.2.1 Définition

Lorsque le code d'un programme dépasse quelques centaines de lignes, il devient difficile de le mettre au point, c'est-à-dire de corriger ses bugs, et de le maintenir, c"est-à-dire de le faire évoluer, par exemple pour intégrer de nouvelles fonctionnalités. Il est alors indispensable de le découper en plusieurs composants aussi indépendants que possible les uns des autres.
Ces composants sont appelés **modules** et un programme ainsi découpé en modules est dit **modulaire**.
On peut alors tester chaque module indépendamment, c'est le mctest unitaire puis tester les modules ensemble, c'est le **test d'intégration**.
La modularité facilite également la réutilisation : un module bien conçu peut être utilisé par d'autres programmes et mis à disposition de la communauté dans une bibliothèque de modules.

11.2.2 Créer un module

Un module définit un ensemble de fonctions, de classes d'objets et parfois de variables globales que l'on peut utiliser dans le programme principal ou dans un autre module sans avoir à connaître le détail de leur implémentation.
L'ensemble de ces définitions constitue **l'interface** de programmation du module ou **API** (Application Programming Interface).
On dit que le module exporte ces définitions. Un programme (ou un module) importe un autre module pour utiliser ses définitions.
En Python, un module correspond à un fichier contenant l'ensemble du code nécessaire à l'implémentation de son API.
L'instruction Python `import m` recherche un fichier de nom `m.py` dans le même répertoire que le fichier contenant l'instruction `import m`, puis dans un ensemble de répertoires standards contenant les bibliothèques installées sur l'ordinateur.

Si le fichier est trouvé, l'instruction crée un objet `m` dont les attributs sont les fonctions, classes et variables exportées par le module, que l'on référence par la notation pointée `m.f`.

⚠ L'instruction `from m import f` ne crée pas d'objet `m`, mais définit directement le nom `f` dans le programme qui l'importe. Dans ce cas, on n'utilisera pas la notation pointée précédente, mais directement le nom `f`.

11.2.3 Utiliser des bibliothèques de modules

Une bibliothèque est un ensemble d'un ou plusieurs modules qui sont installés sur l'ordinateur et qui permettent d'utiliser des fonctionnalités sans avoir besoin de les programmer soi-même. Python inclut une bibliothèque standard avec des modules tels que `math` ou `random` que nous avons déjà utilisés.
Il existe de nombreuses bibliothèques que l'on peut installer sur son ordinateur en plus de la bibliothèque standard.
Ces bibliothèques sont recensées dans des archives dont la plus connue est `https://pypi.org`.
Voici 3 exemples de bibliothèque très utilisés :

- `matplotlib` (affichage de graphiques),
- `PIL` (manipulation d'images)
- `numpy` (calcul numérique)
- `pandas` (manipulation de données, tableaux...)

Utiliser des bibliothèques permet de réduire le coût de développement car cela permet de réutiliser le travail fait par d'autres.
Les bibliothèques les plus répandues sont robustes car elles ont été testées par de nombreux utilisateurs. Elles sont aussi souvent optimisées pour être le plus efficace possible.

11.2.4 Spécifier et documenter un module

Pour pouvoir utiliser un module sans connaître le détail de son implémentation, il faut spécifier les fonctions, classes, méthodes et variables qu'il définit :

- la spécification d'une fonction ou d'une méthode décrit ce qu'elle fait et donne la liste de ses paramètres et leurs types, ainsi que le type de la valeur de retour ;
- la spécification d'une classe décrit son rôle et ceux de ses attributs et méthodes ;
- la spécification d'une variable globale décrit son type et son rôle.

En Python, on peut documenter chaque fonction, classe, méthode et module par une chaîne de caractères, appelée **docstring**, placée juste après sa définition.

Méthode :

1. Pour obtenir l'aide générale d'un module `m`, on entre `help(m)`.
2. Pour obtenir uniquement le docstring principal de `m`, on entre `m.__doc__`
3. `m.__dict__` permet d'obtenir un dictionnaire avec tous les fonctions de l'objet.
4. `math.acos.__doc__` permet par exemple d'obtenir l'aide de la méthode `acos`.

Exercice 11.1

```
class Point:
  """ Classe qui représente un point par ses coordonnées x, y
    """
  def __init__(self, x=0, y=0):
    """ Initialise le point aux coordonnée x, y (0,0 par dé
   faut) """
    self.abscisse = x
    self.ordonnee = y
  def translater(self, v):
    """ Ajoute aux coordonnées du point celles du point v """
    self.abscisse = self.abscisse + v.abscisse
    self.ordonnee = self.ordonnee + v.ordonnee
```

1. Comment obtenir le docstring de l'objet Point ?

Exercice 11.2

Ecrire un programme qui affiche toutes les méthodes du module `math` :

```
acos / acosh / asin / asinh / atan / atan2 / atanh / ceil / copysign
/ cos / cosh / degrees / dist / erf / erfc / exp / expm1 / fabs / fac
torial / floor / fmod / frexp / fsum / gamma / gcd / hypot / isclose
/ isfinite / isinf / isnan / isqrt / lcm / ldexp / lgamma / log / log
1p / log10 / log2 / modf / pow / radians / remainder / sin / sinh / s
qrt / tan / tanh / trunc / prod / perm / comb / nextafter / ulp / pi
/ e / tau / inf / nan /
```

Exercice 11.3

Écrire un programme qui affiche la documentation du module `math`, puis la liste des fonctions définies dans ce module et leur documentation si elle existe. On omettra les noms qui commencent par un underscore (_).

```
acos
Return the arc cosine (measured in radians) of x.

The result is between 0 and pi.

____
acosh
Return the inverse hyperbolic cosine of x.

____
asin
Return the arc sine (measured in radians) of x.

The result is between -pi/2 and pi/2.

____
asinh
Return the inverse hyperbolic sine of x.

____
atan
Return the arc tangent (measured in radians) of x.

The result is between -pi/2 and pi/2.

____
atan2
Return the arc tangent (measured in radians) of y/x.

Unlike atan(y/x), the signs of both x and y are considered.

____
atanh
Return the inverse hyperbolic tangent of x.
```

Exercice 11.4

1. À l'aide de la bibliothèque matplotlib, créer un module Python contenant deux fonctions `barres(t)` et `camembert(t)` qui affichent respectivement les données du tableau `t` sous forme de barres verticales et de camembert.
2. Ajouter à ce module une fonction graphe(x, y) qui affiche un graphe reliant les points (xi, y) des tableaux x et y.
   On utilisera la documentation (simplifiée) en français de matplotlib sur cette page Web : `http://www.python-simple.com/python-matplotlib/pyplot.php`, en particulier les rubriques « Barplot », « Pie » et « Lineplot ».

Exercice 11.5

1. À l'aide de la documentation en français de la bibliothèque Python `random` sur le site `http://www.python-simple.`
   choisir les fonctions aléatoires les plus adaptées pour implémenter les deux fonctions suivantes :
   - `pierre_papier_ciseaux()` : réalise un tirage aléatoire et désigne le gagnant selon les règles du jeu bien connu ;
   - `poker()` : tire au hasard 5 cartes à jouer et les affiche.
2. Trouver dans le site `http:// www.python-simple.com` une fonction ou une méthode permettant de compter le nombre d'occurrences d'un élément dans un tableau.
   Utiliser celle-ci pour compléter la fonction poker et afficher les paires, brelans et carrés éventuels.