Création de modules

1 - Considérations générales

1.1 Fonctions publiques ou privées

Lorsqu'on réalise un module, on distingue les fonctions publiques (celles que l'utilisateur du module aura le droit d'appeler) et les fonctions privées (celles que l'utilisateur ne pourra pas utiliser lui-même).

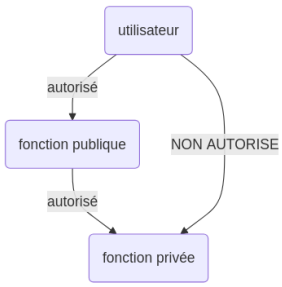

Figure 33.1 – Publiques, privées

1.2 Décomposition d'un problème

On décompose un problème en plusieurs sous-problèmes encapsulés dans des fonctions ne dépassant pas une vingtaine de lignes.

Les foncions simples travaillent sans lancer d'appel à d'autres de vos fonctions.

Les fonctions moyennes lancent des appels aux fonctions simples.

Les fonctions difficiles lancent des appels aux fonctions simples et moyennes pour parvenir à résoudre la tâche qu'on leur demande de réaliser.

1.3 Données et Interface graphique

L'une des stratégies du génie logiciele consiste à séparer les fonctions en catégories distinctes :

- celles du MODELE : elles modifient ou créent les données.
- celles de la VUE : elles réalisent un affichage et récupèrent les actions de l'utilisateur.
- celles du CONTROLEUR : elles intègrent la logique du logiciel en faisant notamment la liaison entre VUE et MODELE.

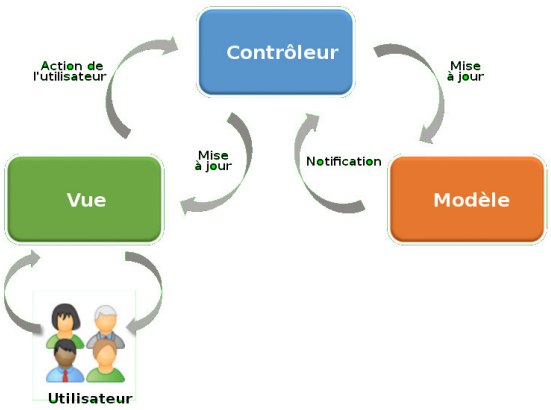

Figure 33.2 – mvc, Source Wikipédia CC BY-SA 4.0

1.4 Organisation du .py du module

Un module suit une organisation précise, comme n'importe quel programme.

- La **documentation globale** du fichier : intérêt, prototypes des fonctions publiques et quelques exemples d'utilisation initiale, un quickstart.
- Les **importations**
- Les déclarations de **CONSTANTES et variables globales**
- Les déclarations des **fonctions privées** (que l'utilisateur du module ne doit pas utiliser)
- Les déclarations des **fonctions publiques** (les fonctions librement utilisables)
- Le programme principal (si besoin)
- Le **programme de test** : situé dans la conditionnelle indiqué ci-dessous. Ces lignes s'exécutent si on lance le fichier en tant que programme principal. On y place tests et démonstrations de fonctionnement.

```
if __name__ == '__main__':
    ...
    ...
    ...
```

1.5 Désempaquetage / Unpacking

```
tup = (10, 20)
a, b = tup
# équivalent à
a = tup[0]
b = tup[1]
```

2 - Les modules en Python

2.1 Création d'un module

Un module n'est qu'un fichier Python d'extension `.py` accessible depuis un autre fichier Python à l'aide du mot-clé `import`.

2.2 Utilisation du module

Pour faire la liaison entre votre fichier Python et votre module, votre module doit être **dans le même répertoire** que votre programme Python. Pour importer un module, on utilise le mot-clé `import`. Il faut retenir qu'on importe ce qui se trouve derrière le mot-clé `import`.

Si on place le nom du module , on pourra accéder à tout ce que contient le module de cette façon :

```
import turtle
...
turtle.forward(...)
```

Si on place une fonction , on pourra accéder directement à la fonction mais pas au reste du module :

```
from turtle import forward
...
forward(...)
```

2.3 Choisir entre les deux formes

En NSI : on privilégie la forme `import module` car nous utilisons souvent plusieurs modules à la fois.

Dans d'autres circonstances : la forme `from module import fonction` permet d'écrire moins de choses puisqu'on n'a plus besoin de faire précéder la fonction par le nom du module, mais attention au risque de collisions entre plusieurs noms de fonctions identiques dans les différentes modules.

A éviter en NSI : la forme `from module import *` permet de tout utiliser directement. C'est réellement "dangereux" lorsqu'on importe plusieurs modules. Par contre, si vous n'utilisez que le module `math` par exemple, cela ne pose pas de problème bien entendu.

2.4 Noms de module et de programme

Un module est importé en utilisant simplement son nom.

Il faut donc veiller à ne JAMAIS donnez à vos propres programmes des noms correspondant à des modules qui vous connaissez (`random`, `tkinter`, `turtle`...).

3 - Compléments divers

3.1 Package

Un package est donc un **conteneur permettant de stocker des modules**. En Python, la création d'un package contenant un ou plusieurs modules est facile : il suffit

- de créer un répertoire en lui donnant un nom clair et explicite. Ce nom sera celui du package.
- de créer dans ce répertoire un fichier-python vide nommé `__init__.py` (pour cela ouvrir Thonny et enregistrer directement ce fichier à l'endroit voulu sous ce nom `__init__.py`.
- de placer dans ce répertoire le ou les fichiers des modules voulus.

```
activite
    dessiner.py
    __init__.py
```

Exemple avec cette configuration :

```
from activite.dessiner import triangle

informations_feutre = {...}
triangle(...)
```

Cela veut dire : va dans le package `activite`, trouve le module `dessiner` et ramener l'accès direct à la fonction `triangle`.

3.2 Bibliothèque

Une bibliothèque est le nom qu'on donne à un ensemble de modules, que ceux-ci soient encapsulés dans un package, ou pas. Bien entendu, dans 99.9% des cas, les modules d'une bibliothèque sont transmis sous forme de package plutôt qu'en tant que fichiers indépendants.