Méthodologie de la programmation

Chapitre 3

La programmation modulaire avec Python

Pablo Rauzy <pr@up8.edu>
pablo.rauzy.name/teaching/mp

La programmation modulaire avec Python

Les modules

- Le code Python est organisé en *modules*.
- Vous savez déjà écrire un module Python : il s'agit simplement d'un fichier avec du code Python dedans !
- Tout ce qui est défini dans le fichier fera parti du module.
- Pour utiliser un module, il suffit de faire appel à la directive **`import`** avec le nom du fichier (sans le `.py`).

- Le fichier `foo.py` :

```
1 a = 42
2
3 def f (n):
4     print(__name__)
5     return a + n
```

- Le fichier `bar.py` :

```
1 import foo
2
3 print(foo.a) # affiche "42"
4 print(foo.f(9)) # affiche "51"
```

Modules exécutables

- Le code qui est dans le fichier importé est exécuté, du coup on ne veut pas qu'il contienne autre chose que des définitions.
- La solution fourni par Python est une variable `__name__` qui vaut `"__main__"` quand le programme est appelé directement, ou le nom du module quand le programme est importé.
- Exemple avec le fichier `name.py` :

```
1 print("Le nom du module est : " + __name__)
```

  - Si on le lance comme un programme avec la commande `python3 name.py` :
    `Le nom du module est : __main__`.
  - Si on l'importe comme une bibliothèque avec `import name` :
    `Le nom du module est : name`.

- Le fichier `fact.py` :

```
1  def fact (n):
2      result = 1
3      while n > 1:
4          result = n * result
5          n = n - 1
6      return result
7
8  if __name__ == '__main__':
9      import sys
10     print(fact(int(sys.argv[1])))
```

- On peut l'utiliser comme programme : `python3 fact.py 6`
- On peut l'utiliser comme bibliothèque : **`import`** `fact`

Le shebang

- Il est possible de rendre un script Python directement exécutable, sans avoir à le passer en argument de la commande `python3`.
- Pour cela il faut faire deux choses :
  1. marquer le fichier comme exécutable,
  2. indiquer quel interpréteur est chargé de son exécution.
- 1. se fait avec la commande `chmod` (comme “CHange MODe”).
  - `chmod +x` *`chemin-du-fichier`* rend exécutable (`+x`) le fichier donné.
- 2. se fait en mettant un *shebang* en première ligne du fichier.
  - Un shebang commence par `#!`, puis est suivi du chemin vers l’interpréteur : `#!/usr/bin/python3` par exemple.

- Le nouveau fichier `fact.py` :

```
1  #!/usr/bin/python3
2
3  def fact (n):
4      result = 1
5      while n > 1:
6          result = n * result
7          n = n - 1
8      return result
9
10 if __name__ == '__main__':
11     import sys
12     print(fact(int(sys.argv[1])))
```

- On peut l'utiliser comme programme directement : `fact.py 6`
- On peut l'utiliser comme bibliothèque : **`import`** `fact`

Différentes façons d'importer un module

- Si dans un autre script ou dans l'interpréteur on **`import fact`**, on peut maintenant utiliser la fonction `fact.fact` :

```
#!/usr/bin/python3

import fact

def print_factorial_steps (n):
    for i in range(1, n+1):
        print('×'.join([str(x) for x in range(1, i+1)])
            + ' = ' + str(fact.fact(i)))

if __name__ == '__main__':
    n = int(input('Jusqu\'où afficher factorielle ? '))
    print_factorial_steps(n)
```

Différentes façons d'importer un module

- Si dans un autre script ou dans l'interpréteur on **`import fact`**, on peut maintenant utiliser la fonction `fact.fact` :

```
1  #!/usr/bin/python3
2
3  import fact
4
5  def print_factorial_steps (n):
6      for i in range(1, n+1):
7          print('×'.join([str(x) for x in range(1, i+1)])
8              + ' = ' + str(fact.fact(i)))
9
10 if __name__ == '__main__':
11     n = int(input('Jusqu\'où afficher factorielle ? '))
12     print_factorial_steps(n)
```

- Il y a des cas comme ici où ça peut être embêtant d'écrire `fact.fact`.
- Il y a plusieurs solutions pour remédier à ce problème.

- On peut renommer le module lors de l'import :

```
1  #!/usr/bin/python3
2
3  import fact as f
4
5  def fact (x, y):
6      return x + y
7
8  def print_factorial_steps (n):
9      for i in range(1, n+1):
10         print('×'.join([str(x) for x in range(1, i+1)])
11             + ' = ' + str(f.fact(i)))
12
13 if __name__ == '__main__':
14     n = int(input('Jusqu\'où afficher factorielle ? '))
15     print_factorial_steps(n)
```

- Cela peut permettre d'éviter des conflits de noms.

from … import …

- On peut sélectivement importer directement certaines définitions :

```
#!/usr/bin/python3

from fact import fact

def print_factorial_steps (n):
    for i in range(1, n+1):
        print('×'.join([str(x) for x in range(1, i+1)])
            + ' = ' + str(fact(i)))

if __name__ == '__main__':
    n = int(input('Jusqu\'où afficher factorielle ? '))
    print_factorial_steps(n)
```

- Il est possible d'importer plusieurs définitions en les séparant par des virgules, ou directement toutes les définitions avec un `*`.

Bibliothèques et paquets

- Un ou plusieurs modules peuvent former une *bibliothèque*.
- On peut également organiser ses modules sous forme de *paquet*.

- La *bibliothèque standard* de Python contient **beaucoup** de modules.
- On a vu lors du dernier TP le module `random` qui sert à la génération de données aléatoires.
- On a aussi vu dans ce cours le module `sys`, qui a notamment le tableau `argv` avec les arguments de la ligne de commande.
- Il y a aussi des fonctions mathématiques dans `math`.
- La documentation de la bibliothèque standard est disponible en ligne : `https://docs.python.org/3/library/`.
- La version de Python installée sur les machines du Bocal est la 3.4.2.

- Le dernier TP utilisait également une bibliothèque qui s'appelle Pygame.
- Elle est distribuée sous forme de paquet.
- On reviendra sur son utilisation dans le prochaine chapitre.

Python Package Index

- Il y a énormément de bibliothèques Python, qui font toutes sortes de choses.
- Beaucoup peuvent être installée depuis PyPI (*Python Package Index*).
  `https://pypi.org/`
- L'outil `pip` est fourni avec Python est permet d'installer des paquets depuis PyPI.

→ Voyons ensemble comment l'utiliser.

Créer un paquet Python

- Un paquet Python est un dossier qui contient un fichier `__init__.py`.
- Quand on importe le paquet, c'est ce fichier est qui lu.
- À l'intérieur d'un paquet Python, on peut importer des modules en utilisant un chemin relatif :
  - `import .foo` importe `foo` dans le même dossier ;
  - `import ..bar` importe `bar` depuis le dossier parent ;
  - (bien sûr, `foo` et `bar` peuvent eux-mêmes être des paquets ou des modules).

Environnement virtuels

- Quand on travail sur un projet, c'est pratique de pouvoir installer localement l'ensemble des paquets Python dont le projet dépend dans les versions spécifiques au projet.
- Cela permet d'éviter les conflits de versions entre différents projets.
- C'est à ça que sert l'outil `venv` qui est fourni avec Python.

→ Voyons ensemble comment l'utiliser.