Python
Modules, scripts, paquets

Nicolas Delestre

INSA
ROUEN NORMANDIE

Organisation générale d'un fichier .py

- L'encodage du fichier est obligatoirement utf-8
- Le fichier est composé de plusieurs parties
  1. Le rôle du fichier sous forme de *docstring* qui sera utilisé par help()
  2. Des métadonnées qui seront utilisées par `help()` : `__author__` (str), `__contact__` (sequence de `str`), `__credits__` (sequence de `str`), `__maintainer__` (`str`) `__email__` (`str`) `__version__` (`str`), `__copyright__` (`str`), `__date__` (`str` au format ISO 8601)
  3. Les déclarations/définitions :
     - Variables globales (en *SNAKE_CASE* majuscule)
     - Classes
     - Fonctions
     - Le code à exécuter (équivalent du *main* dans d'autres langages)

INSA ROUEN NORMANDIE

Script

- Suite de définitions et d'instructions d'un fichier `.py` qui représente un programme

```
$ python3 nom_du_script.py
```

- Grâce au *shebang* (`#!`) on peut indiquer au système d'exploitation UNIX quel interpréteur exécuté lorsque le script est utilisé avec les droits en exécution. Le première ligne du script doit alors être :

  #!/usr/bin/env python3

```
$ ./nom_du_script.py
```

- Suite de définitions et d'instructions d'un fichier `.py` qui peuvent être importées (dans un autre module ou dans un script)
- Les définitions sont propres au module
- Les instructions en dehors de toute fonction ou classe sont interprétées au premier `import` (elles servent à initialiser certaines variables globales au module)
- Un module peut être utilisé en tant que script.
  `if __name__== "__main__":` permet de conditionner l'interprétation d'instructions dans ce cas

INSA ROUEN NORMANDIE

fibo.py

```
#!/usr/bin/env python3
"""
Exemple issu du tutoriel de python: https://docs.python.org/
"""

__author__ = "Nicolas Delestre"
__contact__ = "Nicolas Delestre"
__copyright__ = "Copyleft"
__credits__ = ["Nicolas Delestre"]
__license__ = "GPL"
__version__ = "1.0.1"
__maintainer__ = "Nicolas Delestre"
__email__ = "nicolas.delestre@insa-rouen.fr"
__status__ = "Production"

def fib(n):
    """ retourne les nombres de fibonacci jusqu'au rang n """
    result = []
    a, b = 0, 1
    while b < n:
        result.append(b)
        a, b = b, a+b
    return result

if __name__ == "__main__":
    import sys
    print(fib(int(sys.argv[1])))
```

Comment importer un module ?

- `import nom_du_module` : l'utilisation d'un élément du module nécessite de préfixer cet élément par le nom du module (notation pointée)
- `from nom_du_module import element` : l'utilisation de l'élément est alors direct (sans faire référence au module)
  - l'utilisation de `*` en lieu et place de l'élément permet d'importer toutes des définitions d'un module (à éviter)
- le suffixe `as` permet de renommer localement un module importé ou l'élément du module importé

INSA ROUEN NORMANDIE

Attention à la casse des caractères dans le nommage des modules !

- La PEP8 (`https://www.python.org/dev/peps/pep-0008/`) préconise d'utiliser le *snake_case* pour nommer les modules

Comment l'interpréteur Python trouve-t-il le fichier `.py` nécessaire ?

- Lorsqu'un module est importé (par exemple `import un_module`), l'interpréteur Python recherche le fichier correspondant (pour l'exemple `un_module.py`) dans la liste des répertoires référencés par la variable `sys.path`
- `sys.path` contient
  1. le répertoire courant
  2. les répertoires présents dans la variable d'environnement `PYTHONPATH`
  3. les répertoires systèmes Python

ROUEN NORMANDIE

```
$ python3 fibo.py 10
[1, 1, 2, 3, 5, 8]
$ chmod +x fibo.py
$ ./fibo.py 10
[1, 1, 2, 3, 5, 8]
$ python3
Python 3.5.2 (default, Nov 17 2016, 17:05:23)
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import fibo
>>> fibo.fib(10)
[1, 1, 2, 3, 5, 8]
>>> import fibo as fb
>>> fb.fib(10)
[1, 1, 2, 3, 5, 8]
>>> from fibo import fib as fibonacci
>>> fibonacci(10)
[1, 1, 2, 3, 5, 8]
```

Paquet (ou *Package*) 1 / 4

- Organisation hiérarchique de paquets et de modules
- `import` permet de désigner un module en donnant le chemin des paquets (notation pointée)
- `from..import` permet d'importer
  - un module : entre le `from` et l'`import` il n'y a qu'un chemin de paquets
  - un élément d'un module : entre le `from` et l'`import` il y a un chemin de paquets se terminant par le module
- Les paquets sont représentés par :
  - un répertoire, le contenu du paquet (paquet ou module) est dans le répertoire
  - un fichier __init__.py (obligatoire) dans le répertoire, qui contient une suite d'instructions qui permet d'initialiser le paquet (exécutée lors du premier import).
    C'est ici qu'est contrôlé le `from..import *` pour un paquet (initialisation de la variable `__all__` du paquet)

Exemple issu du tutoriel Python

```
sound/                          Top-level package
      __init__.py               Initialize the sound package
      formats/                  Subpackage for file format conversions
              __init__.py
              wavread.py
              wavwrite.py
              aiffread.py
              aiffwrite.py
              auread.py
              auwrite.py
      effects/                  Subpackage for sound effects
              __init__.py
              echo.py
              surround.py
              reverse.py
      filters/                  Subpackage for filters
              __init__.py
              equalizer.py
              vocoder.py
              karaoke.py
```

```
$ cat sound/__init__.py
$ cat sound/formats/__init__.py
$ cat sound/effects/__init__.py
#!/usr/bin/env python3
__all__ = ["echo","surround","reverse"]
$ cat sound/filters/__init__.py
#!/usr/bin/env python3
__all__ = ["equalizer"]
$ cat sound/formats/wavread.py
#!/usr/bin/env python3
def wavread():
    print("wavread")
```

INSA ROUEN NORMANDIE

Exemple issu du tutoriel Python

1

```
>>> import sound.effects.echo
>>> sound.effects.echo.echo()
echo
>>> from sound.effects import echo
>>> echo.echo()
echo
>>> from sound.effects.echo import echo
>>> echo()
echo
```

2

```
>>> from sound.formats import *
>>> wavread.wavread()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'wavread' is not defined
```

3

```
>>> from sound.effects import *
>>> echo.echo()
echo
>>> surround.surround()
surround
>>> reverse.reverse()
reverse
```

4

```
>>> from sound.filters import *
>>> equalizer.equalizer()
equalizer
>>> vocoder.vocoder()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'vocoder' is not defined
```

Import relatif

- Lorsque le module d'un paquet nécessite l'import d'un paquet ou d'un module d'une autre branche du même procjet, on peut utiliser des imports relatifs (`.` pour le pauqte courant, `..` pour le paquet parent)

Exemple issu du tutoriel Python

```
sound/
      __init__.py
      formats/
              __init__.py
              ...
      effects/
              __init__.py
              echo.py
              surround.py
              ...
      filters/
              __init__.py
              equalizer.py
              ...
```

Depuis le module `surround.py`

```
from . import echo
from .. import formats
from ..filters import equalizer
```

Conclusion

Nous avons vu dans ce cours

- qu'un fichier .py se nomme un module
- qu'un module peut être un script
- que l'on peut organiser hiérarchiquement le code en paquet, qui contiennent des paquets ou des modules
- que l'instruction `import` permet de charge un module
- que l'instruction `from .. import` permet de charger un module d'un packahe ou l'élément d'un module
- que le suffixe `as` permet de renommer localelemnt un module ou l'élément d'un module

INSA ROUEN NORMANDIE