

Distribuer un module Python

ProgFest 2022

Maxence Larose

7 mars 2022

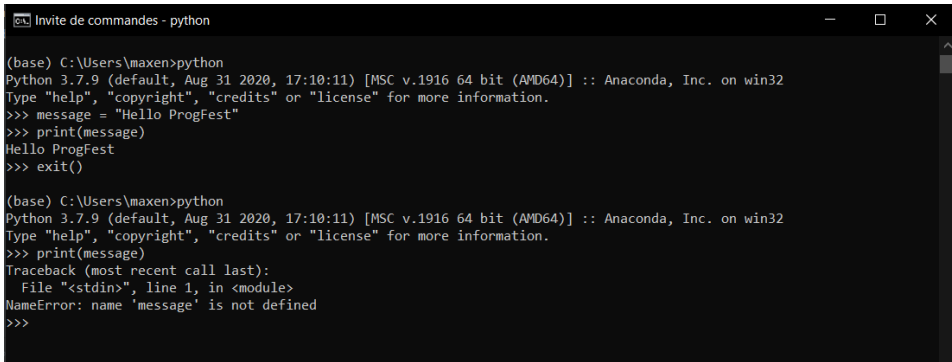


UNIVERSITÉ
LAVAL



Utiliser Python en ligne de commande

- Avantage : Rapide
- Désavantage : Impossible de créer des programmes complexes, car le *namespace* est vidé à chaque fois.



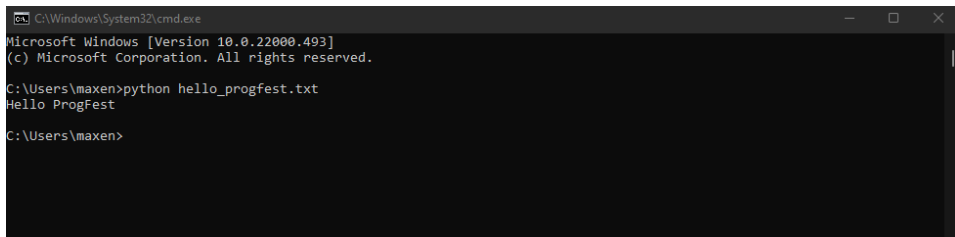
```
(base) C:\Users\maxen>python
Python 3.7.9 (default, Aug 31 2020, 17:10:11) [MSC v.1916 64 bit (AMD64)] :: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> message = "Hello ProgFest"
>>> print(message)
Hello ProgFest
>>> exit()

(base) C:\Users\maxen>python
Python 3.7.9 (default, Aug 31 2020, 17:10:11) [MSC v.1916 64 bit (AMD64)] :: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print(message)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'message' is not defined
>>>
```

Qu'est-ce qu'un script ?

Un script est une séquence d'instructions Python, contenue dans un fichier.

- Avantage : Simple et efficace
- Désavantage : Difficile à maintenir pour des programmes complexes.



```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.22000.493]
(c) Microsoft Corporation. All rights reserved.

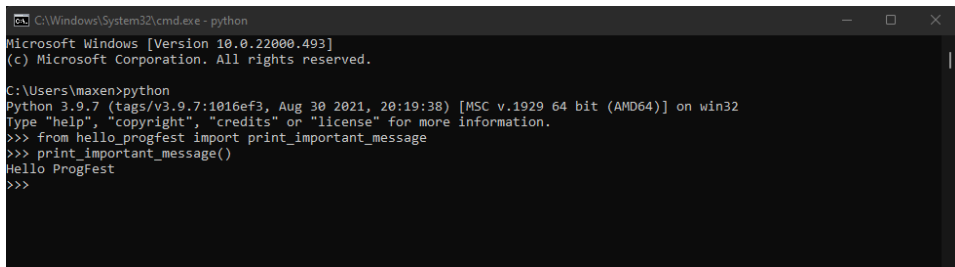
C:\Users\maxen>python hello_progfest.txt
Hello ProgFest

C:\Users\maxen>
```

Qu'est-ce qu'un module ?

Un module est un fichier avec l'extension `.py` contenant du code Python qui peut être importé dans un autre fichier Python. Le nom du fichier est le nom du module.

- Avantage : Les définitions (fonctions, classes, objets) d'un module peuvent être importées dans d'autres modules, dans des scripts ou dans le terminal.
- Désavantage : Absence de structure hiérarchique.

A screenshot of a Windows command prompt window titled "C:\Windows\System32\cmd.exe - python". The window shows the output of running the Python interpreter. It displays the Windows version (10.0.22000.493), the Python version (3.9.7), and the path to the Python executable (C:\Users\maxen>python). The output also shows the Python version (3.9.7), the build date (Aug 30 2021, 20:19:38), and the architecture (MSC v.1929 64 bit (AMD64)) on win32. The user then enters a series of commands to import a module named "hello_progfest" and call a function named "print_important_message". The output of the function is "Hello ProgFest".

```
C:\Windows\System32\cmd.exe - python
Microsoft Windows [Version 10.0.22000.493]
(c) Microsoft Corporation. All rights reserved.

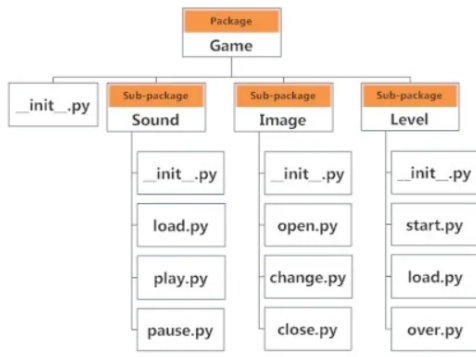
C:\Users\maxen>python
Python 3.9.7 (tags/v3.9.7:1016ef3, Aug 30 2021, 20:19:38) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> from hello_progfest import print_important_message
>>> print_important_message()
Hello ProgFest
>>>
```

Qu'est-ce qu'un package ?

Les packages sont une façon de structurer l'espace de noms des modules de Python en utilisant des noms de modules pointés. Par exemple, le nom de module `A.B` désigne un sous-module nommé `B` dans un package nommé `A`. Un package est donc simplement un répertoire, un dossier, qui doit respecté certaines contraintes dont nous discuterons.

- **Avantage :** Permet d'organiser vos modules d'une manière cohérente pour que vous et d'autres personnes puisse utiliser et réutiliser efficacement le code développé.

Qu'est-ce qu'un package ?



```
C:\Windows\System32\cmd.exe - python
Microsoft Windows [Version 10.0.22000.493]
(c) Microsoft Corporation. All rights reserved.

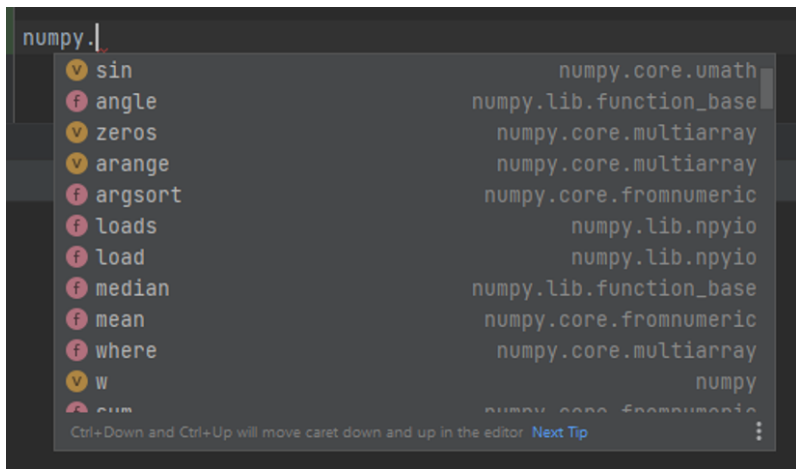
C:\Users\maxen>python
Python 3.9.7 (tags/v3.9.7:1016ef3, Aug 30 2021, 20:19:38) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> from game.sound import play
>>> play.play_theme_song()
Theme song playing...
>>>
```

Structure d'un package : Les fichiers `__init__.py`

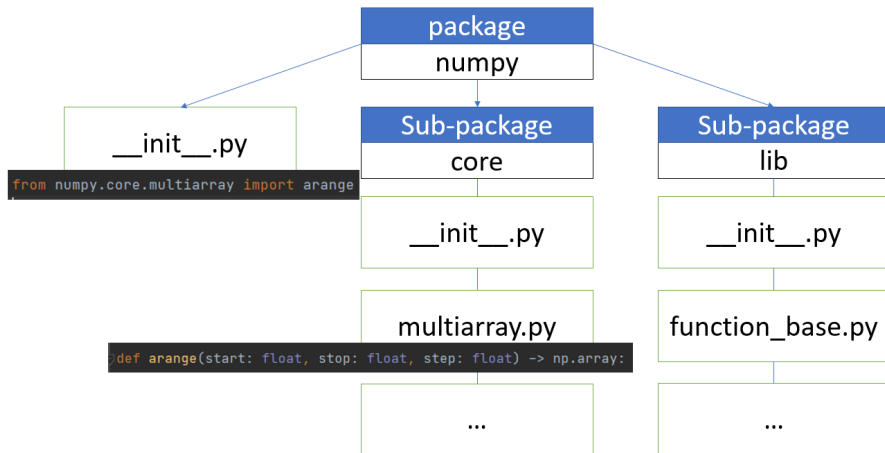
Chaque package Python est un répertoire qui DOIT contenir un fichier spécial appelé `__init__.py`. Ce fichier indique que le répertoire qu'il contient est un package Python et qu'il peut donc être importé de la même manière qu'un module. Il est généralement vide, mais il peut être utilisé pour exporter des modules du package sous un nom plus pratique, contenir des fonctions, etc.

Si vous supprimez le fichier `__init__.py`, Python ne recherchera plus pour des sous-modules dans ce répertoire, et les tentatives d'importation du module échoueront.

Structure d'un package : Les sous-modules

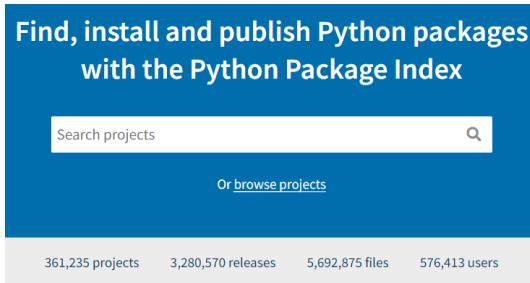


Structure d'un package : Les sous-modules



Qu'est-ce que PyPI ?

PyPI est LE répertoire de packages pour les développeurs Python. Il existe également TestPyPI pour essayer les outils de distribution sans affecter le monde réel.



Structure d'un projet

Un projet bien fait se doit d'avoir une structure claire. Il faut donc réfléchir à savoir quels sont les fichiers que nous avons et comment les organiser correctement.

Une des structures recommandées est présentée ici.

```
project_name
├── .github
│   └── workflows
│       └── main.yml
├── docs
│   ├── make.bat
│   ├── Makefile
│   ├── README.md
│   ├── conf.py
│   └── index.rst
├── examples
│   └── example.py
├── package_name
│   ├── __init__.py
│   └── main.py
├── tests
│   └── __init__.py
├── .gitignore
├── .readthedocs.yml
├── LICENSE.txt
├── MANIFEST.in
├── README.md
├── requirements.txt
└── setup.py
```

Le fichier setup.py est utilisé pour construire une *source distribution*, ou plus communément une **sdist**. Celle-ci contient tout le code source et toutes les meta-données du package. Par défaut, la distribution sera générée sous la forme d'un fichier tar gzipé.

Structure d'un projet : Le fichier setup.py

```
import setuptools

setuptools.setup(
    name="package_name",
    version="0.0.1",
    author="Example Author",
    author_email="author@example.com",
    description="A small example package",
    long_description=open("README.md", "r", encoding="utf-8").read(),
    long_description_content_type="text/markdown",
    url="https://github.com/pypa/sampleproject",
    classifiers=[
        "Programming Language :: Python :: 3",
        "License :: OSI Approved :: MIT License",
        "Operating System :: OS Independent",
    ],
    packages=setuptools.find_packages(),
    install_requires=['matplotlib>=3', 'numpy'],
    python_requires=">=3.6",
)
```

Structure d'un projet : Le fichier MANIFEST.in

Un fichier MANIFEST.in est constitué de commandes, une par ligne, spécifiant à setuptools d'ajouter ou de supprimer un ensemble de fichiers de la sdist.

```
1 include README.md  
2 include examples/
```

- requirements.txt → Spécifier les packages nécessaires pour travailler au développement de votre package.

- requirements.txt → Spécifier les packages nécessaires pour travailler au développement de votre package.
- README.md → Documentation qui apparaît entre autres sur GitHub.

- requirements.txt → Spécifier les packages nécessaires pour travailler au développement de votre package.
- README.md → Documentation qui apparaît entre autres sur GitHub.
- LICENSE.txt → Votre projet a besoin d'une licence open source, sinon personne ne pourra l'utiliser.

- `package_name` → Package contenant tout le code source.

Structure d'un projet : Les dossiers

- `package_name` → Package contenant tout le code source.
- `exemples` → Dossier contenant plusieurs exemples d'utilisation du package.

Structure d'un projet : Les dossiers

- `package_name` → Package contenant tout le code source.
- `exemples` → Dossier contenant plusieurs exemples d'utilisation du package.
- `tests` → Package contenant tous les tests unitaires.

- `package_name` → Package contenant tout le code source.
- `exemples` → Dossier contenant plusieurs exemples d'utilisation du package.
- `tests` → Package contenant tous les tests unitaires.
- `docs` → Dossier contenant l'ensemble des fichiers liés à la documentation Sphinx du code.

Structure d'un projet : Les dossiers

- `package_name` → Package contenant tout le code source.
- `exemples` → Dossier contenant plusieurs exemples d'utilisation du package.
- `tests` → Package contenant tous les tests unitaires.
- `docs` → Dossier contenant l'ensemble des fichiers liés à la documentation Sphinx du code.
- `.github` → Dossier contenant l'ensemble des workflows utiles pour l'intégration continue du package.

Sphinx fournit le meilleur générateur de documentation automatique en Python. Il utilise principalement le docstring des différentes classes et fonction du package pour créer la documentation.

Vous pouvez héberger gratuitement votre documentation sur la plateforme Read the Docs. En fait, Read the Docs est considéré comme une plateforme de documentation continue pour Sphinx. En pratique, cela signifie que votre documentation est construite, testée et mise à jour à chaque release de votre package.

spike2py

Search docs

Installation

Tutorial

How To Guides

Topic Guides

☐ Reference Guide

trial.TrialInfo

trial.Trial

trial.load

channels.ChannellInfo

channels.Channel

channels.Event

channels.Keyboard

channels.Waveform

channels.Wavemark

sig_proc.SignalProcessing

plot.plot_channel

plot.plot_trial

read.read

Help

License

» Reference Guide

[View page source](#)

Reference Guide

trial.TrialInfo

```
class spike2py.trial.TrialInfo(file: pathlib.Path = None, channels: List[str] = None, name: str = None, subject_id: str = None, path_save_figures: pathlib.Path = None, path_save_trial: pathlib.Path = None) \[source\]
```

Information about trial

See `spike2py.trial.Trial` parameters for details.

trial.Trial

```
class spike2py.trial.Trial(trial_info: spike2py.trial.TrialInfo) \[source\]
```

Class for experimental trial recorded using Spike2

Parameters: `trial_info` (*NamedTuple*) –

`file : pathlib.Path, str`

Absolute path to data file. Only .mat files supported

`channels : List[str]`

List of channel names, as they appeared in the original .smr file Example: ['biceps', 'triceps', 'torque'] If not included, all channels will be processed

`name : str`

Descriptive name for trial; defaults to filename

`subject_id : str`

C'est essentiellement une méthode de développement de logiciel qui consiste à intégrer régulièrement les modifications de code à un répertoire centralisé (le main), suite à quoi des opérations de création et de test sont automatiquement menées. On ne veut pas d'énorme merge complexe.

En pratique :

- Des tests automatisés sur les push fait sur le répertoire principal.
- Rejet des push si les tests de CI échouent.

- flake8 → Enforces pep8 and other formatting standards

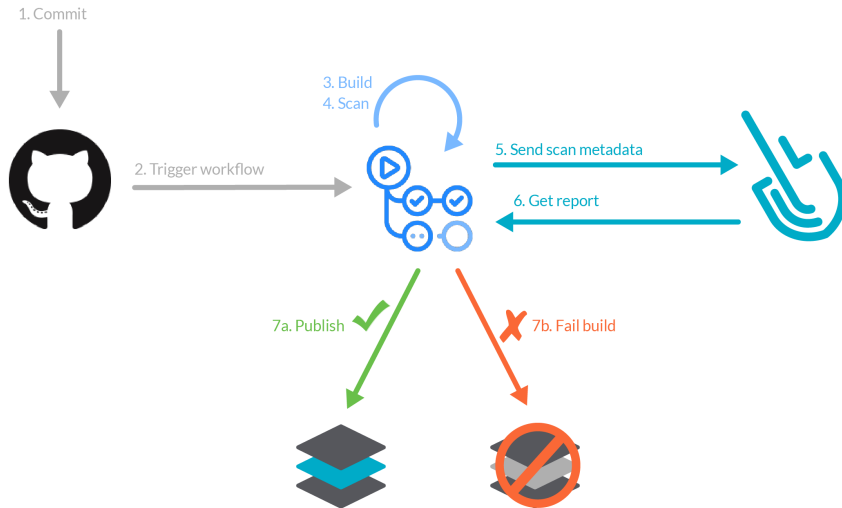
Exemple de « test »

- flake8 → Enforces pep8 and other formatting standards
- mypy → Type checker

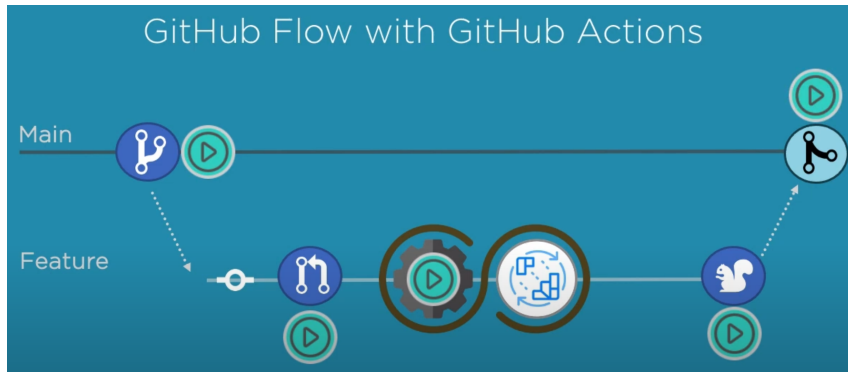
- flake8 → Enforces pep8 and other formatting standards
- mypy → Type checker
- isort → Sort imports alphabetically

- flake8 → Enforces pep8 and other formatting standards
- mypy → Type checker
- isort → Sort imports alphabetically
- pytest → Testing framework for python

Outil d'intégration continue : GitHub Actions



Outil d'intégration continue : GitHub Actions Flow



GitHub Events

on: pull_request_comment

on: delete

on: release

on: deployment

on: pro

on: check_suite

on: pull_request

on: pull_request_review

on: scheduled

on: issue_comment

on: push

on: page_build

on: check_run

on: mile

GitHub WorkFlows

```
name: learn-github-actions
```

```
on: [push]
```

```
jobs:
```

```
  check-bats-version:
```

```
    runs-on: ubuntu-latest
```

```
    steps:
```

- uses: actions/checkout@v2
- uses: actions/setup-node@v1
- run: npm install -g bats
- run: bats -v

◀ Event

◀ Job

◀ Steps

◀ Actions/commands

Le versionnage se fait suivant MAJOR.MINOR.PATCH (e.g. python 3.9.8).

Incrémenter :

- La version MAJOR lorsque vous effectuez des changements d'API incompatibles.
- La version MINOR lorsque vous ajoutez des fonctionnalités d'une manière rétrocompatible.
- La version PATCH lorsque vous apportez des corrections de bogues rétrocompatibles.

v1.22.2

Latest

Compare



charris released this Feb 03, 2022

· 856 commits to main since this release



v1.22.2



f6dddc



NumPy 1.22.2 Release Notes

The NumPy 1.22.2 is maintenance release that fixes bugs discovered after the 1.22.1 release. Notable fixes are:

- Several build related fixes for downstream projects and other platforms.
- Various Annotation fixes/additions.
- Numpy wheels for Windows will use the 1.41 tool chain, fixing downstream link problems for projects using NumPy provided libraries on Windows.
- Deal with [CVE-2021-41495](#) complaint.

The Python versions supported for this release are 3.8-3.10.

Distribuer sur PyPI

Useful commands to publish a package on PyPI or TestPyPI

Create the package source distribution

```
pip install twine wheel build
```

```
python -m build
```

Upload to PyPI

Make sure to register to [PyPI](#) first.

```
python -m twine check dist/*
```

```
python -m twine upload dist/*
```

Upload to TestPyPI

Make sure to register to [TestPyPI](#) first.

```
python -m twine upload --repository testpypi dist/*
```

`https://github.com/MaxenceLarose/ProgFest-PackageDistributionIntroduction`