“Introduction to Python modules.”

May, 2022

Python Packaging programming

PLAN

Introduction

I. Definition et explication d’un package en Python

III.Comment installer un package Python sans pip

IV.Comment installer un package avec pip

V.Quelque package en python

Conclusion

Introduction

Nous allons découvrir une autre facette du langage Python qui en fait un langage à la fois très puissant, modulable et évolutif : l’utilisation de package. Nous allons notamment étudier le fonctionnement de quelques package prédéfinis qu’il convient de savoir manipuler.Dans cet article nous allons montré comment creer un package ,comment installer un package avec pip,comment installer un package sans pip,et enfin etudier quelque package en python

I.Definition et explication d’un package en Python

Les packages (paquets) sont des modules, mais qui peuvent contenir d’autres modules.

Un package correspond à un répertoire sur le système de fichier : il a un nom (nom du package), et contient des fichiers (les modules). Les règles de nom des packages sont donc les mêmes que pour les modules.

Avant Python 3.5, pour être un package, un répertoire devait contenir un fichier **init**.py. Ce n’est plus obligatoire aujourd’hui, mais c’est toujours utile. Quand un package est importé, c’est en fait son module **init** qui l’est.

Si nous créons un sous-répertoire mypackage dans le répertoire courant, et que nous y écrivons le fichier **init**.py suivant :

```
def myfunction():
    return None
```

Alors mypackage est utilisable comme un module contenant une fonction myfunction.

```
import mypackage

 mypackage.myfunction()
```

L’intérêt principal des packages étant tout de même de contenir plusieurs modules. On peut ainsi ajouter un fichier operations.py au répertoire mypackage.

Cela revient à disposer d'un module mypackage.operations. Mais ce module n'est par défaut pas importé dans le package : import mypackage ne donne par défaut pas accès à operations, il faudra importer explicitement ce dernier.creona un autre fichier main.py et executons le scripts

```
 import mypackage.operations
print(mypackage.operations.addition(1, 2))

from mypackage import operations
print(operations.soustraction(1, 2))

from mypackage.operations import multiplication
print(multiplication(1, 2))
```

On note aussi qu'il n'est pas nécessaire d'importer mypackage pour pouvoir importer mypackage.operations.

Pour donner accès au module operations directement en important mypackage, il est nécessaire de toucher au fichier **init**.py. Ce fichier correspondant à ce qui est chargé à l'importation du package, nous pouvons y importer mypackage.operations, ce qui le rendra directement accessible. **init**.py import mypackage.operations

Puis en console ou main.py:

```
import mypackage
 mypackage.operations.addition(1, 2)
```

Un package est un niveau d'indirection supérieur au module, mais il est aussi possible d'avoir des packages de packages, et packages de packages de packages, et plus encore : vers l'infini et au-delà !

Imports relatifs

Au sein d'un package, il est parfois usant d'avoir à en recopier le nom complet.

Dans l'exemple précédent de mypackage/**init**.py important le sous-module operations, on doit réaliser un ***import mypackage.operations**. Si le package change de nom, il faudra mettre à jour cette ligne. Et plus généralement, ça allonge parfois inutilement les lignes, et ne rend pas bien compte du fait qu'il s'agisse d'un module du même package.

Les imports relatifs permettent de résoudre ce problème. À l'intérieur d'un package, le module . référence le package courant. dans `python __init__.py` Mais en plus de cela, on a aussi par exemple .operations qui référence directement le module operations. Écrire from.operations import addition dans **init**.py permettrait de donner accès à la fonction addition directement depuis mypackage.addition.

Quand plusieurs packages sont imbriqués, il est aussi possible de référencer les packages parents avec des syntaxes similaires.

.. est le package parent ; ... le grand-parent ; etc.

Ainsi, avec ***subpackage*** un sous-répertoire de ***mypackage***, et calcul.py un fichier du répertoire ***subpackage*** : ***calcul*** peut avoir accès à la fonction ***addition*** avec le code suivant.

```
from ..operations import addition
```

import ... et from ... import ...

sont similaires mais non équivalentes dans l'évaluation des modules. Cela devient évident dans le cas des importations circulaires.

import

Prenons deux fichiers a.py et b.py qui seraient présents dans le répertoire courant.

dans le fichier a.py

```
import b

def demo():
    print(b.x)
```

dans le fichier b.py

```
import a

x = 20

def demo():
    a.demo()
```

Puis depuis une console Python :

```
>>> import b
>>> b.demo()
```

Tout fonctionne bien. Mais essayons de comprendre comment cela fonctionne derrière.

Tout d'abord, nous importons ***b*** et commençons ainsi l'évaluation du module. Dès que l'évaluation commence, le module est ajouté aux modules importés. Ce cache nous permet d'éviter d'importer plusieurs fois le même module : s'il est présent, nous le retournons simplement plutôt que de l'évaluer à nouveau.

Ainsi, ***b*** commence à être évalué. Sa première ligne est celle qui importe le module ***a***. L'interpréteur commence alors à évaluer le module ***a***.

L'interpréteur essaie d'abord d'importer ***b***.***b*** est déjà présent dans les modules importés, même s'il est actuellement vide, donc l'importation se termine correctement. Le reste du module ***a*** est évalué, puis le reste du module ***b*** est également évalué.

from ... import

Remplaçons maintenant le contenu du fichier a.py par le suivant.

```
from b import x
def demo():
  print(x)
```

Ici, si nous ré-exécutons le code précédent depuis la console, ça pose problème !

ImportError: cannot import name 'x' from partially initialized module 'b' (most likely due to a circular import)

Pourquoi cette différence ?

Au début du module a, nous demandons maintenant d'extraire la variable x du module b. Cependant, ce dernier module n'est pas encore complètement chargé (il est actuellement bloqué sur la ligne import a, jusqu'à ce que a soit complètement évalué), il ne contient donc pas de valeur x pour le moment.

Ceci est également dû au fait que les noms de variables utilisés dans les fonctions ne sont pas résolus avant l'exécution. La ligne print(b.x) n'est donc pas un problème, puisque lorsque la fonction est appelée, b aura été complètement chargée.

On peut reproduire l'erreur, sans import de ..., si on essaie d'utiliser x directement à partir du module a.

II.Comment installer un package Python sans pip

La pratique la plus courante d'installation de bibliothèques externes dans votre système consiste à utiliser la commande Python pip . Cependant, il existe une autre méthode pour installer manuellement les bibliothèques Python sans utiliser la commande pip .

Dans cet article, nous allons expliquer comment installer manuellement un package python.

Vous trouverez ci-dessous l'approche étape par étape pour installer manuellement une bibliothèque.

Étape 1: téléchargement des fichiers

- Allez sur le site recherchez le package que vous souhaitez installer
- Dans le menu sur le côté gauche, cliquez sur le bouton de téléchargement des fichiers.

Étape 2: Décompressez les fichiers téléchargés s'ils sont compressés à l'aide d'un logiciel de décompression.

Étape 3: Remplacez le répertoire de travail actuel par le fichier contenant Setup.py à l'aide de la commande cd.

Étape 4: Lisez attentivement les instructions d'installation et installez comme indiqué.

Étape 5: Après avoir remplacé le répertoire de travail actuel par le fichier contenant Setup.py, tapez la commande suivante:

```
python Setup.py install
```

Voilà, vous êtes maintenant prêt à utiliser la bibliothèque installée et vous pouvez importer cette bibliothèque dans votre programme python.

III.Comment installer un package avec pip

Le langage Python propose un nombre considérable de librairies. L'abondance est telle que des équipes de développeurs ont décidé de créer des gestionnaires de paquets pour faciliter la gestion de toutes ces librairies. Le gestionnaire de paquets PIP est devenu l'outil le plus utilisé pour gérer les librairies de son projet, à tel point qu'il est désormais intégré par défaut à Python. Si ce n'est pas le cas de votre installation, vous pouvez également le faire manuellement comme nous l'avons vue précédemment.

Si vous récupérez et installez Python sur le site officiel, le gestionnaire de paquets PIP est déjà intégré à l'installeur. L'utilisation dépend du système d'exploitation que vous possédez. Si vous travaillez avec un système basé sur Unix comme Linux ou Mac, vous devez utiliser la commande "python3…". L'argument "-m" indique que l'on souhaite appeler un module, ici ce sera "pip". La commande "install" installer un paquet.

```
  python3.9 -m pip install [le_paquet_a_installer]
```

Si vous travaillez avec un système de la famille Windows, alors la commande "py" remplace "python3…". Le reste de la commande ne change pas.

```
py -m pip install [le_paquet_a_installer]
```

Il peut cependant arriver que PIP ne soit pas installé avec le langage Python. Dans ce cas, vous obtiendrez le message d'erreur "No module named pip". Il existe une commande Python qui vérifie que PIP est installé et, si ce n'est pas le cas, récupère et installe le module.

```
python3.9 -m ensurepip --default-pip
```

Une autre solution consiste à télécharger sur le site des développeurs de PIP le script d'installation du gestionnaire de paquets. Voici la commande pour le télécharger avec Linux ou l'outil Wget pour Windows :

```
wget https://bootstrap.pypa.io/get-pip.py
```

Vous pouvez lancer l'installation avec la commande suivante. Vous devez posséder les droits d'administration pour installer PIP avec ce script.

```
sudo python3.6 get-pip.py
```

V.Quelque package en python

Lorsque vous travaillez avec des projets, vous pouvez rencontrer des scénarios où vous ne pourrez pas résoudre avec le codage standard d'un langage de programmation. Nous avons besoin de packages et de modules pour surmonter ces problèmes.Dans cet article nous allons etudié quelques packages et modules en python.

BeautifulSoup4 – web scraping

BeautifulSoup ,le package est utilisée pour le web scraping. C'est un module pratique avec lequel travailler. Même les débutants peuvent commencer à l'utiliser en utilisant le docs.

Vous pouvez installer BeautifulSoup en tapant la commande suivante dans le terminal / ligne de commande.

```
  pip install beautifulsoup4
```

```
## importing bs4, requests modules
import bs4
import requests

## initialiser  url
url = "https://www.consumerreports.org/cro/a-to-z-index/products/index.htm"

## obtenir la réponse de la page en utilisant la méthode get du module requests
page = requests.get(url)

## stocker le contenu de la page dans une variable
html = page.content

## création d'un objet BeautifulSoup
soup = bs4.BeautifulSoup(html, "lxml")

## voir la classe ou l'id de la balise qui contient les noms et les liens
div_class = "crux-body-copy"

## récupérer tous les divs en utilisant la méthode find_all
div_tags = soup.find_all("div", class_=div_class)## trouver les divs qui ont la classe mentionnée

## nous verrons toutes les balises avec une balise qui a un nom et un lien à l'intérieur de la div.
for tag in div_tags:
    print(tag)
```

Science des données et apprentissage automatique

Numpy est utilisé pour le calcul scientifique.

Il nous permet de travailler sur des tableaux multidimensionnels. L'implémentation des tableaux n'existe pas en Python. Les développeurs utilisent principalement numpy dans les projets d'apprentissage automatique. C'est un package facile à apprendre et open-source. Presque tous les ingénieurs en apprentissage machine ou les data scientists utilisent ce module pour les calculs mathématiques complexes.

Exécutez la commande suivante pour installer le numpy module.

```
pip install numpy
```

Pandas

Pandas est un module d'analyse de données. Nous pouvons filtrer les données plus efficacement en utilisant le pandas bibliothèque. Il propose différents types de structures de données pratiques à utiliser. Il fournit également la gestion de fichiers avec différents formats de fichiers.

Installez le module à l'aide de la commande suivante.

```
pip install pandas
```

Matplotlib

matplotlib est une bibliothèque de traçage de graphes 2D. Vous pouvez visualiser les données en utilisant matplotlib.

Il génere des images des figures dans différents formats. il trace différents types de diagrammes tels que des graphiques à barres, des graphiques d'erreur, des histogrammes, des nuages de points, etc., vous pouvez installer le matplotlib en utilisant la commande suivante.

```
pip install matplotlib
```

Conclusion

Cet article nous a permis de mettre en évidence toute la complexité qui se cache derrière un package . En partant de d'installation d'un package avec ou sans pip et l'etude des packages existant.