PYTHON

Présenté par :

Xavier TABUTEAU

Modules

Les modules utiles

Il existe un grand nombre de modules préprogrammés qui sont fournis d'office avec Python. C'est ce qu'on appel la bibliothèque standard. Vous pouvez en trouver d'autres chez divers fournisseurs. Souvent on essaie de regrouper dans un même module des ensembles de fonctions apparentées, que l'on appelle des bibliothèques.

Math

from math import *
sqrt()
sin()
cos()
pi
…

DateTime

from datetime import date
from datetime import time
from datetime import datetime

module_math.py
module_dates.py

Modules

PyDoc

PyDoc est un module intégré à Python qui permet de consulter et de générer automatiquement de la documentation pour le code Python. Il fonctionne à partir des docstrings que l'on écrit dans les modules, classes, fonctions et méthodes.

Utilité principale

- Consulter la documentation d'un module ou d'une fonction sans quitter le terminal.
- Générer de la documentation HTML lisible dans un navigateur.
- Accéder à l'aide pour des fonctions intégrées ou du code personnalisé.

Utilisation :

pydoc nom_module ou pydoc fonction_builtin
Affiche la docstring du module on de la fonction_buitin.

pydoc nom_module.fonction
Affiche la docstring de la fonction du module nom_module.

pydoc nom_dossier ou pydoc dossier.sous_dossier….
Affiche la docstring du fichier __init__.py situé dans le dossier (s'il existe) et la liste de tout les module dans le dossier.

pydoc -w …
Génère un fichier HTML dans le dossier ou l'on se trouve au lieu d'afficher dans la console.

Les modules utiles

module_subprocess.py

Modules

subprocess

Le module subprocess en Python permet de lancer des processus externes, de communiquer avec eux et de récupérer leur sortie ou leur code de retour. Il est très utile pour exécuter des commandes système depuis un script Python.

Les fonctions principales :

run() : Exécute une commande, attend qu'elle se termine, et retourne un objet CompletedProcess.

Popen() : Exécute une commande, et continue le code python même si la commande n'est pas terminée.

check_output() : Exécute une commande et retourne la sortie (stdout).

- Options courantes :

stdin, stdout, stderr : Pour gérer les flux d'entrée/sortie.
shell=True : Exécute la commande dans un shell (attention aux failles de sécurité).
text=True (ou universal_newlines=True) : Pour avoir des chaînes de caractères au lieu de bytes.
timeout : pour définir un délai d'expiration.

Les modules utiles

Modules

sys

Ce module est utile pour interagir avec l'environnement d'exécution et manipuler les entrées/sorties ou les modules dynamiquement. Le module sys en Python fournit des fonctions et variables permettant d'interagir avec l'interpréteur Python. Voici ses principales fonctions et variables :

- Gestion des Arguments de la Ligne de Commande

  sys.argv : Liste des arguments passés au script (le premier élément est le nom du script).
  sys.exit([code]) : Termine le programme avec un code de sortie optionnel.

- Flux d'Entrée et de Sortie

  sys.stdin : Gère l'entrée standard.
  sys.stdout : Gère la sortie standard.
  sys.stderr : Gère les erreurs et les messages d'alerte.

- Informations sur l'Interpréteur Python

  sys.version : Retourne la version de Python sous forme de chaîne.
  sys.platform : Indique le système d'exploitation.
  sys.executable : Chemin de l'exécutable Python utilisé.

Les modules utiles

Modules

sys

- Gestion des Modules

sys.modules	: Dictionnaire des modules chargés.
sys.path	: Liste des chemins où Python recherche les modules.
sys.getrecursionlimit()	: Obtient la profondeur maximale de récursion.
sys.setrecursionlimit(n)	: Définit la profondeur maximale de récursion.

- Gestion de la Mémoire

sys.getsizeof(objet)	: Retourne la taille d'un objet en mémoire.
sys.maxsize	: Taille maximale d'un entier en Python.
sys.getrefcount(objet)	: Retourne le nombre de références à un objet.

module_sys.py

Modules

os

Le module os en Python permet d'interagir avec le système d'exploitation. Ce module est souvent utilisé avec shutil pour des manipulations avancées de fichiers et répertoires. Voici ses principales fonctions :

- Gestion des processus

system(command)	: Exécute une commande système.
popen(command)	: Exécute une commande et retourne son résultat.
getpid()	: Retourne l'ID du processus actuel.
getppid()	: Retourne l'ID du processus parent.

- Informations système

name	: Donne le nom du système d'exploitation ('posix', 'nt', etc.).
uname()	: Retourne des informations sur le système (uniquement sous Unix).
environ	: Dictionnaire contenant les variables d'environnement.
getlogin()	: Retourne le nom de l'utilisateur connecté.

Les modules utiles

Modules

os

- Gestion des fichiers et répertoires

getcwd()	: Retourne le répertoire de travail actuel.
chdir(path)	: Change le répertoire de travail.
listdir(path)	: Liste les fichiers et dossiers d'un répertoire.
mkdir(path)	: Crée un dossier.
makedirs(path)	: Crée un dossier et ses parents si nécessaires.
remove(path)	: Supprime un fichier.
rmdir(path)	: Supprime un dossier vide.
removedirs(path)	: Supprime un dossier et ses parents s'ils sont vides.

- Manipulation des chemins

path.join(path1, path2)	: Construit un chemin valide.
path.exists(path)	: Vérifie si un chemin existe.
path.isfile(path)	: Vérifie si un chemin est un fichier.
path.isdir(path)	: Vérifie si un chemin est un répertoire.
path.abspath(path)	: Donne le chemin absolu d'un fichier.
path.basename(path)	: Retourne le nom du fichier d'un chemin.
path.dirname(path)	: Retourne le dossier d'un chemin.

module_os.py

Modules

pathlib

Le module pathlib de Python v3.4 on supérieur fournit une interface orientée objet pour manipuler des chemins de fichiers et de répertoires. Ce module est particulièrement utile pour remplacer os.path avec une approche plus intuitive et plus puissante. Voici ses principales fonctions et classes :

- Création et manipulation de chemins

```
Path("chemin/vers/fichier")  : Crée un objet Path représentant un chemin (relatif ou absolu).
Path.cwd()                   : Retourne le chemin du répertoire de travail actuel.
Path.home()                  : Retourne le chemin du répertoire utilisateur.
```

- Navigation et vérifications

```
exists()    : Vérifie si le chemin existe.
is_file()   : Vérifie si le chemin est un fichier.
is_dir()    : Vérifie si le chemin est un répertoire.
```

- Opérations sur les chemins

```
name        : Nom du fichier avec extension.
stem        : Nom du fichier sans extension.
suffix      : Extension du fichier.
parent      : Répertoire parent du fichier/dossier.
parts       : Renvoie les différentes parties du chemin sous forme de tuple.
```

Les modules utiles

Modules

pathlib

- Lecture et écriture de fichiers

```
read_text()                : Lit le contenu du fichier sous forme de texte.
read_bytes()               : Lit le contenu du fichier en bytes.
write_text("contenu")      : Écrit du texte dans le fichier.
write_bytes(b"contenu")    : Écrit des bytes dans le fichier.
```

- Création et suppression

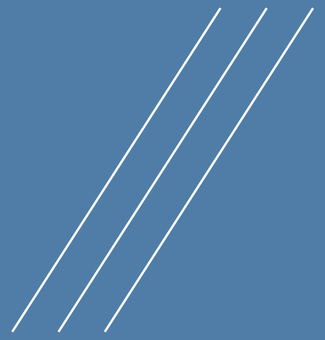
```
mkdir(parents=True, exist_ok=True) : Crée un répertoire (y compris les parents s'ils n'existent pas).
touch()                            : Crée un fichier vide.
unlink()                           : Supprime un fichier.
rmdir()                            : Supprime un répertoire vide.
```

- Gestion des fichiers et répertoires

```
iterdir()                  : Itère sur les éléments d'un répertoire.
glob("*.txt")              : Recherche des fichiers correspondant à un motif.
rename("nouveau_nom")      : Renomme le fichier ou le répertoire.
replace("nouveau_chemin") : Déplace le fichier en écrasant s'il existe déjà.
```

module_pathlib.py

PYTHON



Présenté par

Xavier TABUTEAU