# Python

SOCOMEC

OCTOBRE 2025

# Retour sur vos questions du premier jour



Propriété de B.S. Reproduction complète ou partielle sans autorisation préalable interdite.

# Qui contribue au développement de Python ?
# Les développeurs sont-ils rémunérés ?

## Les principaux acteurs :

### 1. Le Python Steering Council

- Un comité de 5 membres élus chaque année parmi les développeurs « core ».
- Ils décident des orientations techniques (acceptation des PEP, roadmap, etc.).
- Exemple de membres récents : Guido van Rossum (le créateur du langage), Pablo Galindo, Thomas Wouters, Barry Warsaw, etc.

### 2. Les « Core Developers » (environ 100 personnes actives)

- Ce sont les contributeurs ayant les droits d'écriture sur le dépôt officiel CPython (le cœur du langage).
- Ils relisent, testent et valident les changements proposés (Pull Requests).
- Certains travaillent pour des entreprises (Google, Microsoft, Red Hat, etc.), d'autres sont bénévoles.

### 3. Les contributeurs occasionnels (des milliers)

- Toute personne peut proposer des corrections, nouvelles fonctionnalités ou documentation via GitHub (https://github.com/python/cpython si ça vous tente !)



Propriété de B.S. Reproduction complète ou partielle sans autorisation préalable interdite.

# Qui contribue au développement de Python ?
# Les développeurs sont-ils rémunérés ?

## Les développeurs sont-ils rémunérés ?

**Pas directement par la PSF, sauf exceptions.**

**Bénévoles**

Une grande partie des développeurs Python contribuent sur leur temps libre : corrections de bugs, documentation, discussions techniques, etc.

**Salariés d'entreprises**

De nombreux contributeurs sont employés par des entreprises qui financent leurs heures de contribution (Microsoft, Google, Red Hat, IBM, Canonical, Anaconda...)

Ces entreprises utilisent Python massivement et ont donc intérêt à soutenir son évolution.

Exemple : Pablo Galindo (core dev Python 3.10/3.11) travaille chez Bloomberg et y contribue dans le cadre de son emploi.

**Financement ponctuel**

La PSF peut financer certains travaux via des bourses ("grants") pour :

- Des améliorations du compilateur, de la sécurité ou de l'infrastructure CI.
- Des événements communautaires (PyCon, sprints de développement).
- Des projets liés à la documentation ou à l'inclusion.



Propriété de B.S. Reproduction complète ou partielle sans autorisation préalable interdite.

# Qui contribue au développement de Python ?
# Les développeurs sont-ils rémunérés ?

La PSF est financée par :

- Les dons individuels.
- Les entreprises sponsors.
- Les conférences et événements.

Ces fonds servent à entretenir l'écosystème, payer quelques salariés (infrastructure, communication, gestion légale, etc.), mais pas à salarier tous les développeurs.



Propriété de B.S. Reproduction complète ou partielle sans autorisation préalable interdite.

# Qui contribue au développement de Python ?
# Les développeurs sont-ils rémunérés ?

Ce modèle fonctionne car Python a une communauté très mature et collaborative, où :

- Les décisions techniques passent par un processus ouvert (PEP – Python Enhancement Proposal),
- Le code est maintenu sur GitHub,
- Les tests et validations sont publics,
- Et chacun peut participer, du simple bug report au patch de performance.

C’est un équilibre entre :

- Un noyau de développeurs expérimentés,
- Une base mondiale de volontaires passionnés,
- Et un soutien financier indirect des grandes entreprises.



Propriété de B.S. Reproduction complète ou partielle sans autorisation préalable interdite.

# La sécurité des bibliothèques Python est-elle vérifiée avant qu'elles soient disponibles sur PyPI ?

**(Très) partiellement.**

Tout utilisateur peut publier un package après avoir créé un compte. Il n'y a pas de processus de validation centralisé pour la sécurité : PyPI ne teste pas automatiquement le code pour détecter les malwares ou vulnérabilités avant qu'il soit disponible.

PyPI effectue quelques contrôles basiques au moment du téléchargement :

- Vérification du format du package (sdist, wheel).
- Validation du nom et de la version du package.
- Analyse automatisée pour détecter certains patterns de malwares connus (via des outils internes), mais ce n'est pas exhaustif.

Mais PyPI :

- Ne compile pas ni n'exécute le code pour vérifier sa sécurité.
- Ne contrôle pas la qualité ou la vulnérabilité du code.

Il y a eu plusieurs cas de typosquatting ou de packages malveillants sur PyPI. La PSF et PyPI ont mis en place :
2FA pour les mainteneurs,
Signalement et suppression rapide des packages suspects.

Les administrateurs de PyPI sont très vigilants concernant les packages malveillants évidents (typosquatting, packages conçus uniquement pour diffuser des malwares, etc.). Pour la plupart, les packages malveillants disparaissent dans les quelques jours suivant leur mise en ligne. Mais il faut vérifier soi-même qu'on télécharge bien ce qu'on pense télécharger.

Propriété de B.S. Reproduction complète ou partielle sans autorisation préalable interdite.

# La sécurité des bibliothèques Python est-elle vérifiée avant qu'elles soient disponibles sur PyPI ?

## Comment faire ?

### PyPI privé / interne

Vous hébergez vos propres packages Python sur un serveur interne ou cloud sécurisé.

Exemples :

- DevPi : serveur PyPI privé avec gestion de version, contrôle d'accès et cache PyPI.
- Artifactory / Nexus Repository : serveurs professionnels supportant PyPI, Maven, npm… Permettent de stocker vos packages internes et de proxy PyPI avec contrôle d'accès.

Avantages :

- Vous contrôlez exactement quels packages sont disponibles.
- Possibilité d'auditer et scanner avant mise à disposition.
- Permet de bloquer certains packages PyPI publics non approuvés.

### Mirrors sécurisés de PyPI

Certaines entreprises créent des miroirs internes de PyPI, synchronisés automatiquement, puis filtrés.

Exemple : bandersnatch pour créer un miroir PyPI local.

Avantages :

Accès rapide et fiable pour tous les postes.

Filtrage et audit possibles avant publication interne.



Propriété de B.S. Reproduction complète ou partielle sans autorisation préalable interdite.

# La sécurité des bibliothèques Python est-elle vérifiée avant qu'elles soient disponibles sur PyPI ?

## Comment faire ?

### Outils d'audit avant installation

Même avec PyPI public, vous pouvez renforcer la sécurité :

- pip-audit : détecte les vulnérabilités connues.
- Safety : analyse vos dépendances contre une base CVE.
- bandit : audit de sécurité statique pour votre code Python.

### Stratégie combinée recommandée

Pour un environnement critique ou industriel :

- Créer un PyPI interne ou proxy sécurisé.
- Filtrer / scanner les packages externes avant autorisation.
- Installer uniquement dans environnements virtuels isolés.
- Verrouiller les versions via requirements.txt ou poetry.lock.
- Ajouter un audit automatisé régulier des packages.



Propriété de B.S. Reproduction complète ou partielle sans autorisation préalable interdite.

# PySerial et TestStand

PySerial est la librairie standard en Python pour communiquer via des ports série (RS-232/RS-485/USB), très utile dans les environnements de test automatisés avec TestStand.

**Installation / configuration**

Identifier l'interpréteur Python utilisé par TestStand (dans TestStand : Configure → Python Adapter → Options → Python Version) et noter le chemin, par ex. C:\Python311\python.exe.

Installer PySerial avec cet interpréteur :

```
"C:\Python311\python.exe" -m pip install pyserial
```

Pour vérifier l'installation :

```
"C:\Python311\python.exe" -m pip show pyserial
```



Propriété de B.S. Reproduction complète ou partielle sans autorisation préalable interdite.

# PySerial et TestStand

**Exemple d'utilisation :**

```
import serial
import time

# Ouverture du port série
ser = serial.Serial('COM3', baudrate=9600, timeout=1)

# Envoi d'une commande
ser.write(b'*IDN?\r\n')

# Lecture de la réponse
time.sleep(0.2)
response = ser.readline().decode().strip()

# Affichage du résultat dans TestStand
print("Réponse :", response)

# Fermeture du port
ser.close()
```



Propriété de B.S. Reproduction complète ou partielle sans autorisation préalable interdite.

# IHM avec Python

## Bibliothèques GUI

- PySide6 / PyQt6 — meilleur compromis pour applicatifs industriels : riches widgets, designer (Qt Designer), bonne gestion threads/signaux. Framework graphique utilisé par PySide6 et PyQt6 : Qt.
  - PySide6 = licence LGPL (parfois préférable en entreprise).
  - PyQt6 = GPL/commercial.
- Tkinter — livré avec Python, simple, bon pour outils très légers.
- wxPython — natif look & feel Windows, stable.
- DearPyGui — rapide à prototyper, orienté GUI moderne et perf (mais moins « standard »).
- Web UI (Flask/Dash/Streamlit + navigateur) — très pratique pour remote/local UIs, nécessite bundling d'un navigateur ou accès réseau.
- Kivy — si écran tactile / multi-platforme embarquée.



Propriété de B.S. Reproduction complète ou partielle sans autorisation préalable interdite.

# IHM avec Python

## Bonnes pratiques pour une IHM de test instrumenté

- Ne pas bloquer la boucle UI. Toutes les communications série / longues opérations doivent être dans un thread / worker.
- Utiliser des files d'attente ou signaux pour envoyer données du worker vers l'UI (thread-safe).
- Timeouts et reconnexions : gère les erreurs COM (port absent, occupation).
- Logs et trace visibles depuis l'IHM (console/logfile).
- Paramétrage : interface pour choisir COM, baud, timeouts ; persist config (JSON).
- Déploiement : utiliser un environnement isolé (venv/conda) et builder (PyInstaller) pour produire un EXE.
- Intégration TestStand :
    - Option A : lancer l'application GUI séparément et laisser TestStand piloter via sockets/HTTP/fiche ou via commande CLI.
    - Option B : exécuter directement un script Python via TestStand (Adapter). Dans ce cas attention à la boucle d'événements GUI (TestStand doit lancer l'interpréteur qui démarre la boucle Qt).
    - Souvent on préfère applis séparées (plus simples à déployer) et une API locale (REST/socket/pipe) pour communication.



Propriété de B.S. Reproduction complète ou partielle sans autorisation préalable interdite.

# IHM avec Python

## Installation

```
python -m venv ts_gui_env
ts_gui_env\Scripts\activate.bat

python -m pip install pyserial pyside6
```

OU

```
python -m venv ts_gui_env
ts_gui_env\Scripts\activate.bat

python -m pip install pyqt6
```



Propriété de B.S. Reproduction complète ou partielle sans autorisation préalable interdite.

# IHM avec Python

## Packaging / déploiement

Si serial_gui.py contient un script qui ouvre un port série en arrière-plan et met à jour l'UI via signaux.

```
python -m pip install pyinstaller
pyinstaller --onefile --noconsole serial_gui.py
```

Cela produit un EXE autonome (attention aux dépendances Qt, tester en machine cible).

**Test sur poste cible** : installer un venv ou fournir l'exe + redistribuer les DLL Qt si besoin.

**Installer sur postes TestStand** : fournir l'exe et TestStand l'appelle en ligne de commande, ou configurer TestStand pour lancer directement le script Python (pointer vers le python.exe du venv).



Propriété de B.S. Reproduction complète ou partielle sans autorisation préalable interdite.

# IHM avec Python

## Intégration TestStand - patterns pratiques

Launcher : TestStand lance l'exe de l'IHM et lui envoie commandes via socket/HTTP/local REST.

Adapter direct : TestStand appelle des fonctions Python (TestStand Python Adapter) — plus simple mais parfois compliqué si UI a une boucle événementielle.

Files/DB : échange via fichiers/temp ou base de données légère (sqlite) si pas besoin de temps réel.



Propriété de B.S. Reproduction complète ou partielle sans autorisation préalable interdite.

## Un exemple de script serial_guy.py



Propriété de B.S. Reproduction complète ou partielle sans autorisation préalable interdite.