



ENSGTI
ÉCOLE D'INGÉNIEURS



Programmation Orientée Objet (Python)

Mini-Projet instrumentation en Python : partie 1

[A.Martin](#)

2026-2027



Visual Studio Code



Objectif

Ce mini-projet de 12 heures a pour objectif d'appliquer les principes de la Programmation Orientée Objet en Python à travers le pilotage d'un appareil de laboratoire polyvalent (un Rigol DHO924S), afin de bien comprendre **les relations entre classes**.

À la fin, vous saurez :

- Définir une classe mère regroupant les attributs et méthodes communs à plusieurs appareils ;
- Utiliser l'héritage pour créer une classe fille spécialisée qui étend le comportement de la classe mère ;
- Redéfinir ou compléter des méthodes héritées, et faire appel à `super()` ;
- Distinguer et implémenter les relations entre objets (composition, agrégation, association) et savoir les représenter sur un diagramme UML ;
- Structurer un projet en plusieurs fichiers (classes d'un côté, script de test de l'autre) ;
- Piloter un appareil réel via une liaison VISA (Ethernet/USB) avec des commandes SCPI ;
- Tester et valider le comportement de chaque classe par des instanciations concrètes.

Durée estimée : 8 heures

1. Partie théorique

1.1 Qu'est-ce que SCPI ?

Le protocole SCPI (Standard Commands for Programmable Instruments) est un standard qui définit un ensemble de commandes textuelles permettant de piloter à distance des appareils de mesure et de test électroniques : oscilloscopes, générateurs de signaux, alimentations, multimètres, analyseurs de spectre **indépendamment** de leur fabricant.

Les commandes sont organisées de façon hiérarchique, à la manière d'une arborescence de fichiers, avec des mots-clés séparés par des deux-points. Une commande d'action ne renvoie aucune réponse (elle règle un paramètre ou déclenche une opération), tandis qu'une commande de requête, marquée par un point d'interrogation, interroge l'appareil et attend une réponse en retour.

Un socle de commandes communes, préfixées par une étoile, est partagé par la quasi-totalité des appareils compatibles SCPI : `*IDN?` pour obtenir l'identification de l'appareil, `*RST` pour le réinitialiser, `*CLS` pour effacer la pile d'erreurs, entre autres. Au-delà de ce socle commun, chaque famille d'appareils possède ses propres commandes spécifiques à sa fonction (réglage d'une échelle sur un oscilloscope, réglage d'une fréquence sur un générateur, etc.).

Il est important de noter que SCPI ne définit que le langage des commandes elles-mêmes : le transport physique de ces commandes (Ethernet, USB, GPIB, série...) relève d'un standard distinct, le plus souvent VISA, qui s'occupe de l'établissement de la connexion et de l'acheminement des messages vers l'appareil.

L'ensemble des commandes SCPI pour le rigol DHO924S est disponible sur [E-Learn](#).

1.2 Qu'est-ce que VISA ?

Le standard VISA (Virtual Instrument Software Architecture) définit une interface de communication commune permettant de piloter des appareils de mesure et de test, quel que soit le bus physique utilisé pour les relier à l'ordinateur — USB, Ethernet (LAN), GPIB, série (RS-232), ou encore PCI.

Son rôle est de masquer les différences entre ces interfaces matérielles derrière une API unique : une fois la connexion établie, le code applicatif utilise les mêmes fonctions de base (ouvrir une session, écrire une

commande, lire une réponse, fermer la session) quel que soit le bus réellement utilisé en arrière-plan. Cela évite d'avoir à réécrire le pilotage d'un appareil selon qu'il soit branché en USB ou en Ethernet.

Chaque appareil est identifié par une chaîne de ressource VISA, qui précise l'interface et l'adresse à utiliser — par exemple `TCPIP::192.168.1.100::INSTR` pour un appareil en réseau, ou `USB0::0x1AB1::0x0517::DS1234567::INSTR` pour un appareil branché en USB.

Concrètement, VISA s'appuie sur une bibliothèque logicielle (un driver) installée sur l'ordinateur, qui peut être le pilote officiel NI-VISA (National Instruments) ou un équivalent, ou une implémentation open source en Python comme `pyvisa-py`, qui ne nécessite aucun driver externe. C'est cette bibliothèque qui assure concrètement le transport des données, tandis que le contenu des commandes échangées suit généralement un langage comme SCPI.

2. Travail à réaliser

2.1 Étape préparatoire

1. Ouvrez un terminal.
2. Dans votre dossier de travail «*tp_python*», créer un dossier «*projet_instrumentation*».
3. Déplacez-vous dans ce dossier et créer votre arborescence de projet.
4. Créez l'environnement virtuel «*env_projet*», activez-le et installez les bibliothèques *pyvisa* et *pyvisa-py*.

pyvisa est la bibliothèque Python qui expose l'API VISA : les objets *ResourceManager*, les méthodes *open_resource()*, *write()*, *query()*, *close()*, etc. C'est la couche avec laquelle on programme — mais à elle seule, elle ne sait pas physiquement parler à un appareil. Elle a besoin d'un backend qui effectue le travail concret de communication sur le bus (USB, Ethernet...).

pyvisa-py est un backend écrit entièrement en Python, qui réimplémente lui-même le protocole de communication (sockets TCP pour l'Ethernet, *pyusb* pour l'USB, *pyserial* pour le série...) sans dépendre d'aucun driver externe à installer.

5. Désactivez ensuite l'environnement virtuel (vous l'activerez depuis VSCode).
6. Déterminer l'adresse IP de votre Rigol DHO924S.

- Appuyer sur  puis **Utility**
- Exécutez un test de connexion avec votre appareil grâce à la commande **ping**

2.2 Compréhension de la classe *ResourceManager* du module *pyvisa*

Créez un script python qui contient les lignes suivantes :

```
import pyvisa

# 1. Ouverture : on crée un gestionnaire de ressources VISA
rm = pyvisa.ResourceManager('@py') # '@py' = backend pyvisa-py (aucun driver externe requis)
# Ouverture d'une session vers l'appareil : instrument devient un objet de classe Ressource
instrument = rm.open_resource("TCPIP::x.x.x.x::INSTR") #Remplacer avec votre adress ip
instrument.timeout = 5000 #délai d'attente en millisecondes avant erreur

# 2. Requête : on envoie une commande SCPI et on attend la réponse
```

```
reponse = instrument.query("*IDN?")
print("Identification de l'appareil :", reponse)

# 3. Fermeture : on libère la session, puis le gestionnaire de ressources
instrument.close()
rm.close()
```

2.3 Développement des classes

Étape 1 — Premier diagramme UML (avant tout code)

Avant d'écrire la moindre ligne de Python, dessinez un premier diagramme de classes UML avec un logiciel comme « draw.io » ou « plantuml » faisant apparaître au minimum ces 5 classes: *ParcAppareils*, *Appareil*, *DH0924S*, *GBF* et *Oscillo*. À ce stade, ne vous préoccupez pas encore des attributs ni des méthodes précises : l'objectif est uniquement de réfléchir aux **relations** entre ces classes — quelle classe hérite de quelle autre, quelle classe possède, agrège ou utilise quelle autre — et de choisir le bon symbole UML pour chacune (héritage, composition, agrégation, association).

Étape 2 — Choix des attributs, des méthodes et implémentation

Une fois les relations déterminées, définissez pour chaque classe les attributs et les méthodes qui vous semblent pertinents vis-à-vis du comportement réel de l'appareil. Vous êtes libres d'ajouter d'autres classes si vous jugez qu'elles clarifient votre modèle (par exemple une classe représentant une mesure, un channel, ou tout autre concept qui vous paraît utile). Implémentez ensuite l'ensemble en Python, en respectant les relations définies à l'étape 1.

Étape 3 — Diagramme UML final

En fin de projet, redessinez votre diagramme de classes UML à jour, en le faisant correspondre fidèlement au code effectivement écrit : mêmes attributs, mêmes méthodes, mêmes relations. Comparez-le à votre premier diagramme et soyez prêts à expliquer les écarts entre les deux — un modèle UML évolue naturellement à mesure que l'implémentation révèle des besoins non anticipés, et cette comparaison fait partie intégrante de l'exercice.

Résumé

Dans ce TP, vous avez appris à :

- Modéliser le fonctionnement d'une famille d'appareils de laboratoire à travers une hiérarchie de classes Python ;
- Mettre en œuvre l'héritage pour spécialiser une classe mère en classe fille ;
- Distinguer et implémenter les relations entre objets (composition, agrégation, association) ;
- Piloter un appareil réel via une liaison VISA et le protocole de commandes SCPI ;
- Faire évoluer un diagramme UML de la conception initiale à l'implémentation finale.