

Python & Instrumentation

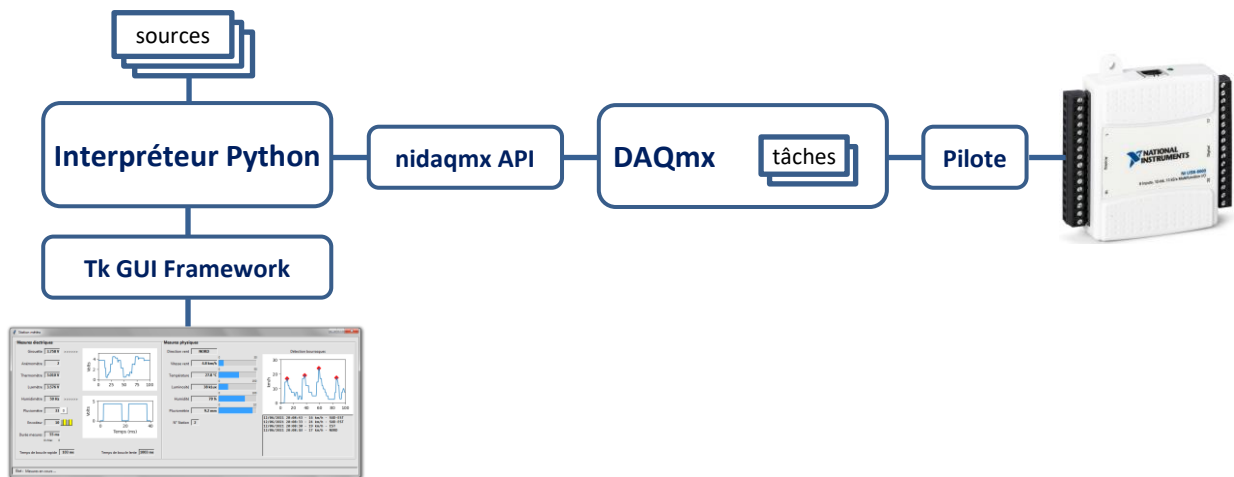
Introduction

Ce cours présente les différents concepts permettant de réaliser un logiciel en **Python** qui effectue des mesures cycliques sur du matériel **National Instruments**, et affiche les résultats par une **Interface Utilisateur Graphique** autonome.

National Instruments fournit une librairie, **nidaqmx**, qui s'interface avec les couches de gestion du matériel. Python est distribué, en standard, avec le gestionnaire d'interface graphique **Tk**.

Le fonctionnement du programme repose entièrement sur des **fichiers sources en Python** :

- gestion de l'interface utilisateur graphique (un outil permet de «dessiner l'interface» et génère des fichiers Python)
- gestion des tâches de mesure



LabVIEW vs Python

Les différences entre ces deux environnements de développement sont flagrantes :

- **LabVIEW** est un outil **payant «tout-en-un»** qui permet de créer l'interface utilisateur et le code sans autre ajout.
- **Python** est un simple **interpréteur de code**, il faut lui ajouter un **éditeur-débugueur de code**, un **éditeur d'interface utilisateur**, et plusieurs **librairies de code** ... mais l'environnement de développement Python est **gratuit** !

L'efficacité de **LabVIEW** pour créer des applications d'**instrumentation** ou de **contrôle industriel** est indéniable.

L'efficacité de **Python** est largement reconnue dans les activités d'Intelligence Artificielle pour l'analyse de données comme le **Machine Learning**, les **réseaux neuronaux**, l'**analyse d'images** ...

Bien que ces techniques soient disponibles avec d'autres langages comme le C++ (et même LabVIEW); historiquement, c'est en Python que ces techniques se sont le plus développées, et où l'on y trouve le plus de ressources ...

Pour certaines **applications de recherche** en physique ou en médical, il peut être nécessaire d'effectuer un **grand nombre de mesures**, puis d'y appliquer de **lourds traitements d'analyse**.

Réaliser l'ensemble de ces opérations en Python est pertinent, et évite de multiplier les outils de développement. Plusieurs outils intégrés (Anaconda, Jupyter ...) permettent de calculer et de visualiser en Python.

Cependant, le **contrôle des équipements de mesure** (configuration, monitoring), la réalisation des mesures en respectant des **contraintes temporelles** ... nécessitent de réaliser un **outil autonome** avec une Interface Utilisateur.

C'est l'objectif principal de ce cours.

Environnement de développement Python

L'environnement de développement est constitué des éléments suivants :

- 1) **noyau Python**, interpréteur du langage.
- 2) éditeur-débugueur de code universel **Visual Studio Code**, accompagné d'une **extension Python** spécifique.
- 3) bibliothèques de fonctions **nidaqmx**, **continuous-threading**, **matplotlib**.
- 4) éditeur d'Interface Utilisateur Graphique **PAGE**.
- 5) l'installation (gratuite) des pilotes **NI-DAQmx** suffit pour accéder au matériel, LabVIEW n'est pas indispensable.

L'installation de l'environnement est expliquée en détail dans les procédures suivantes :

- **Installation Python PC Ecole 2021** : pour configurer les ordinateurs présents dans les salles de cours
- **Installation Python Etudiant 2021** : pour une installation sur votre ordinateur personnel

Interpréteur Python

Sur les PCs de l'école une version récente de Python (3.9) est installée dans le dossier **C:\Python\Python39**.

Un interpréteur Python plus ancien (3.7) est déjà installé sur les PCs de l'école, c'est l'interpréteur par défaut.

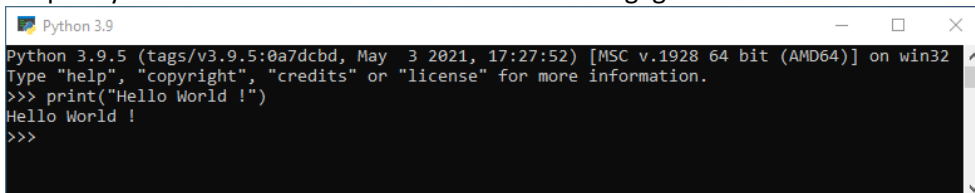
Les bibliothèques additionnelles utilisées dans ce cours ne sont pas présentes dans la version Python 3.7.

Cela pourra causer des dysfonctionnements dans Visual Studio.

Les procédures d'installation citées ci-dessus expliquent comment sélectionner l'interpréteur Python 3.9.

L'interpréteur se lance en mode console avec **C:\Python\Python39\Python.exe**.

On peut y exécuter directement les fonctions du langage.



```
Python 3.9.5 (tags/v3.9.5:0a7dcbb, May 3 2021, 17:27:52) [MSC v.1928 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> print("Hello World !")
Hello World !
>>>
```

Exemple : `print("Hello World !")`

A noter qu'il n'existe pas de commande 'cls' pour effacer la console, il faut taper la ligne suivante : `import os; os.system('cls')`

Ces instructions sont principalement utiles au début d'un programme pour 'nettoyer' le terminal avant d'y afficher des informations.

Pour quitter la console, appuyer sur [Ctrl+z] puis [Entrée]

Les bibliothèques utilisées dans le cours sont visibles dans : **C:\Python\Python39\Lib\site-package** :

- **nidaqmx** accès au matériel d'acquisition NI
- **continuous-threading** gestion de threads cadencés
- **matplotlib** affichage de graphiques mathématiques

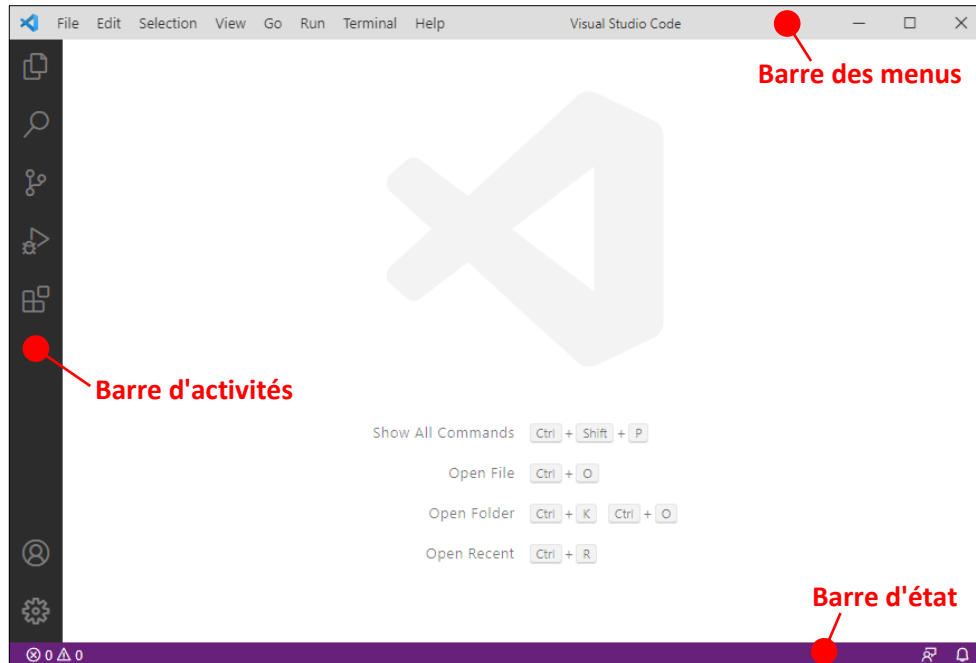
Quelques ressources pour bien commencer en Python :

- Aide en ligne, exemples et tutoriaux pour le langage Python : <https://docs.python.org/3.9>
- Les bibliothèques sur le site <https://pypi.org> disposent d'une page spécifique où l'on y trouve documentation et exemples.
- L'aide du langage et des bibliothèques sont disponibles directement par la **complétion de code** dans **Visual Studio Code**.

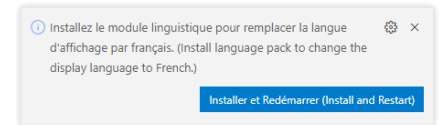
Visual Studio Code

Depuis plusieurs années, Microsoft met à disposition des outils de développement open-sources et gratuits permettant de programmer dans de nombreux langages.

Visual Studio Code, disponible pour Windows, MacOS et Linux, est un **éditeur-débugueur de code universel**, qui dispose d'une interface épurée, et permet d'utiliser la plupart des langages via une importante bibliothèque d'extensions.



Au démarrage une notification peut apparaître pour proposer d'installer la langue française. Installer ce module pour éviter que la notification réapparaisse continuellement.



Par défaut, Visual Studio Code ne gère aucun langage en particulier, il faut obligatoirement installer une extension de langage de programmation avant de pouvoir saisir un source. Voir les procédures d'installation.

Lorsque Visual Studio Code est correctement paramétré pour l'utilisation de Python, plusieurs fonctionnalités permettent de simplifier la saisie du code source.

```
import nidaqmx

with nidaqmx.Task() as task:
    task.ai_channels.add_ai_voltage_chan()
```

(method) add_ai_voltage_chan: (physical_channel, name_to_assign_to_channel="", terminal_config=TerminalConfiguration.DEFAULT, min_val=-5, max_val=5, units=VoltageUnits.VOLTS, custom_scale_name="") -> AIChannel

Creates channel(s) to measure voltage. If the measurement requires the use of internal excitation or you need excitation to scale the voltage, use the AI Custom Voltage with Excitation instance of this function.

Args:

- physical_channel (str): Specifies the names of the physical channels to use to create virtual channels. The DAQmx physical channel constant lists all physical channels on devices and modules installed in the system.
- name_to_assign_to_channel (Optional[str]): Specifies a name to assign to the virtual channel this function creates. If you do not specify a value

L'aide en ligne, sur certaines fonctionnalités de code, est parfois très complète, comme le montre l'exemple de la «**librairie nidaqmx**» ci-contre.

Seule contrainte, il faut d'abord «**importer**» une librairie avant de pouvoir l'utiliser.

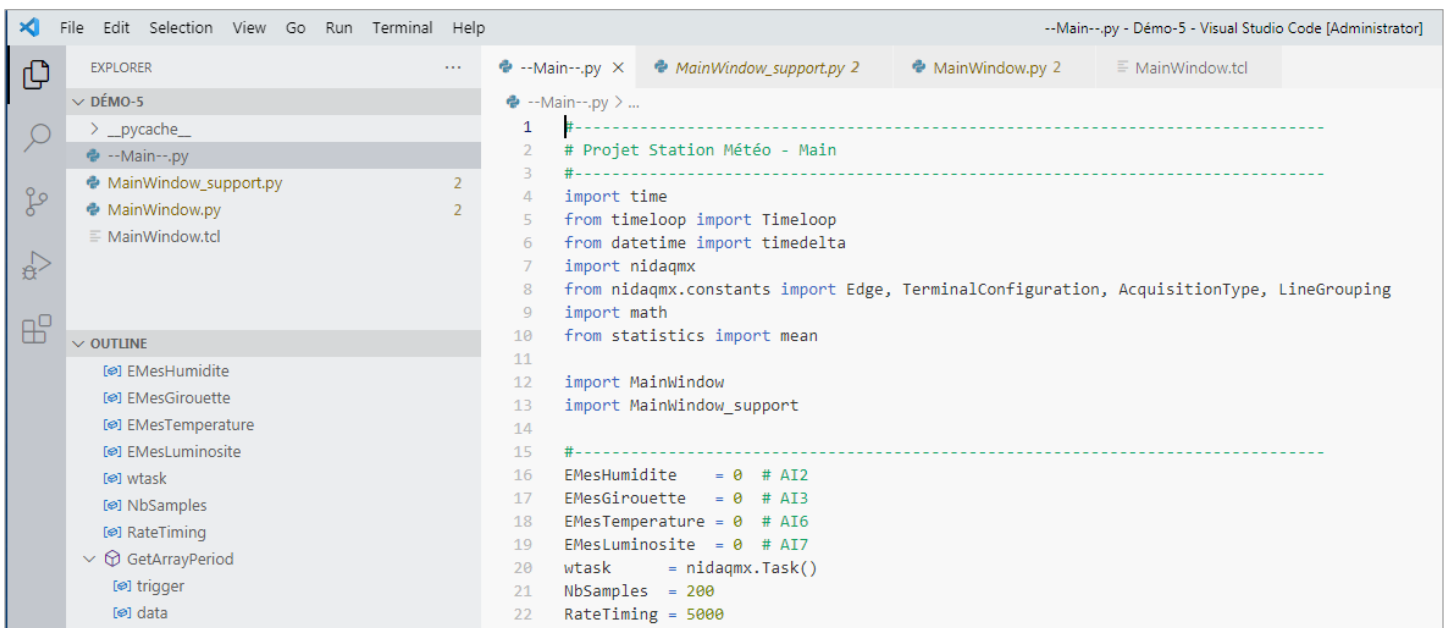
Dans l'exemple ci-dessous, la **complétion de code** permet de saisir les différents arguments d'une fonction à l'aide de *listes pop-up* qui proposent les valeurs adaptées.

L'assistant de code a ajouté automatiquement la ligne «**import TerminalConfiguration**» afin de pouvoir saisir la constante via une liste.

```
import nidaqmx
from nidaqmx.constants import TerminalConfiguration

with nidaqmx.Task() as task:
    task.ai_channels.add_ai_voltage_chan(physical_channel="dev1/ai0", terminal_config=TerminalConfiguration.
    RSE
    DEFAULT
    DIFFERENTIAL
```

La **gestion d'un projet** contenant plusieurs fichiers peut être réalisée très simplement en ouvrant le dossier contenant le projet par le menu «**File\Open Folder**».



Dans le même volet qui affiche l'explorateur de code, un **explorateur de fichiers** permet de naviguer et d'ouvrir les fichiers du projet dans des onglets (double-clic sur chaque fichier).

Très logiquement, l'explorateur de projet se ferme par le menu «**File\Close Folder**».

Création et exécution sans débogage d'un programme

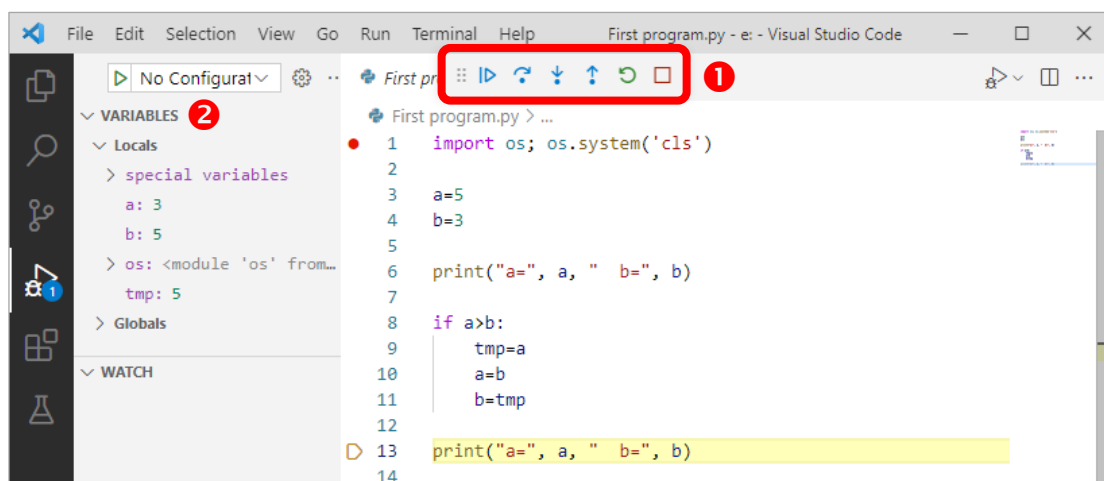
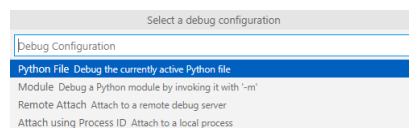
1. Créer un nouveau fichier (menu **File\New File** ou **Ctrl+N**)
2. Saisir le code, bien respecter l'indentation du code
3. Enregistrer le source (menu **File\Save** ou **Ctrl+S**)
4. Exécuter sans débogage (menu **Run\Run Without Debugging** ou **Ctrl+F5** ou bouton ▶)

Exécution avec débogage d'un programme

1. Placer un point d'arrêt dans le source (cliquer dans la marge)
2. Démarrer l'exécution avec débogage (menu **Run\Start Debugging** ou **F5** ou bouton ▶ option Debug)



Si la liste ci-contre apparaît,
sélectionner la première ligne.



Une palette d'outils de débogage apparaît. ①

Le panneau latéral à gauche affiche les variables ② au fur et à mesure de leurs créations.

Ces variables n'étant pas déclarées au préalable, l'interpréteur Python les supprime automatiquement lorsqu'elles ne sont plus utilisées.

Analyse d'un source en Python

Le codage en Python ressemble un peu au codage en C ... avec beaucoup de simplifications :

- pas de marqueur de bloc de code, c'est l'indentation du code qui définit les blocs
- pas de marqueur de fin de ligne
- pas de fonction main
- pas de déclaration obligatoire des variables
- ...

E: > Exemple.py > ...

```
1  # commentaire sur une seule ligne
2
3  """
4  Commentaire sur plusieurs lignes
5  Utilisé aussi pour générer automatiquement la documentation
6  """
7
8  import nidaqmx                                utilisation d'une librairie
9  from nidaqmx.constants import TerminalConfiguration  utilisation d'une sous-partie d'une librairie
10
11  VarBooleen : bool = False                    déclarations de variables typées et initialisées
12  VarEntier  : int  = 0
13  VarFlottant : float = 0.0
14  VarChaine  : str  = "Mon texte"
15
16  CONST_NB_MESURES = 2                        déclarations d'une constante, qui est en fait une variable
                                              (nom en MAJUSCULE pour la différencier d'une variable)
17
18  def max(a, b) :                             déclaration d'une fonction avec paramètres
19      if a >= b :
20          return a                             fin de la fonction et retour du résultat
21      if b > a :
22          return b
23
24  def acquisition() :
25      global data ← utilisation d'une variable externe à la fonction
26                  task est une variable locale
27      with nidaqmx.Task() as task:
28          task.ai_channels.add_ai_voltage_chan("dev1/ai0:1", "", TerminalConfiguration.RSE)
29          data = task.read()
30
31  def affichage() :
32      global data                                la variable data n'est pas déclarée explicitement, elle sera créée automatiquement
33
34      print("Mesures = ", data)
35      print("Max      = ", max(data[0], data[1]))
36
37  acquisition()                                partie principale du programme
38  affichage()
39
```

TERMINAL

```
Mesures = [3.0191532555790346, 2.2504040530009704]
Max      = 3.0191532555790346
```

Remarques :

Le caractère «:» marque le début d'un bloc de code. Un **trait vertical** identifie les **blocs de code**.

Une **variable non déclarée** est créée lors de sa première utilisation en écriture.

On ne peut pas lire une variable non déclarée qui n'a pas été initialisée.

Le langage Python est **case-sensitive** : une lettre en majuscule n'est pas équivalente à la même lettre en minuscule.

Indenter inutilement du code provoque une erreur.

En fin de document, plusieurs liens de **tutos** sont indiqués. Voir aussi les deux PDF «Python Cheat Sheet» joints à ce cours.

Éditeur graphique PAGE

Il existe plusieurs «frameworks» permettant de construire des interfaces graphiques en Python.

Le **framework «Tk»** est intégré en standard dans Python, son interface de programmation est «**tkinter**».

Le seul outil vraiment utilisable pour générer des interfaces graphiques avec Tk est le logiciel libre **PAGE**, disponible pour Windows, MacOS et Linux (ce logiciel s'exécute dans le RunTime Python). Ce logiciel dispose d'une documentation très complète et de quelques exemples. Il est mis à jour régulièrement.

La méthode de développement d'une interface graphique est la suivante :

- création de l'interface graphique visuellement avec l'outil PAGE;
- **génération du code Python** permettant de créer les **composants visuels** et de répondre aux **événements**;
- intégration des sources générés par l'outil dans le projet.

Il est souvent nécessaire de faire plusieurs allers-retours entre PAGE et Visual Studio pendant la mise au point.

Le framework Tk contient **peu de composants visuels** (≈25). Tk est prévu à l'origine pour être associé avec le langage de commande Tcl. Le couple **Tcl/Tk** est surtout utilisé pour créer des utilitaires d'administration système.

Exemple «Hello World !»

À partir de la fenêtre «**Widget Toolbar**» on dépose un «**Button**» et un «**Label**» sur la fenêtre, on peut ensuite les positionner librement et modifier leurs tailles.

La fenêtre «**Attribute Editor**» permet de modifier les caractéristiques des composants : «Alias», «text» ...

Un clic-droit sur le composant bouton permet d'ouvrir la boîte de dialogue «**Widget bindings**» dans laquelle on crée (menu **Insert**) l'événement «**ButtonRelease-1**» auquel on attribue le nom de fonction *callback* «**BtnSayHello**».

The screenshot displays the PAGE GUI development environment. The main window, titled 'PAGE - HelloWorld.tcl', shows a 'Hello World' application with a button labeled 'Say Hello' and a text field containing '---'. The 'Widget Toolbar' on the left lists various widgets like TButton, TCheckbutton, TCombobox, etc. The 'Attribute Editor' on the right shows the properties of the selected widget, including 'Alias = BtnSayHello' and 'text = Say Hello'. The 'Widget bindings for BtnSayHello' dialog box is open, showing the 'Insert' menu and the binding 'ButtonRelease-1' with the callback 'BtnSayHello(e)'.

Bien que le source de cet exemple soit fourni, il est nécessaire de réaliser entièrement l'exemple pour bien comprendre toutes les étapes de la création d'une application Python+GUI.

On peut ensuite générer les deux fichiers Python (voir page suivante) à partir du menu «**Gen_Python**».

HelloWorld.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
#
# GUI module generated by PAGE version 6.1
# in conjunction with Tcl version 8.6
# May 12, 2021 09:11:20 AM CEST platform: Windows NT

import sys

try:
    import Tkinter as tk
except ImportError:
    import tkinter as tk

try:
    import ttk
    py3 = False
except ImportError:
    import tkinter.ttk as ttk
    py3 = True

import HelloWorld_support

def vp_start_gui():
    '''Starting point when module is the main routine.'''
    global val, w, root
    root = tk.Tk()
    top = Main(root)
    HelloWorld_support.init(root, top)
    root.mainloop()

w = None
def create_Main(rt, *args, **kwargs):
    '''Starting point when module is imported by another module.
    Correct form of call: 'create_Main(root, *args, **kwargs)' .'''
    global w, w_win, root
    #rt = root
    root = rt
    w = tk.Toplevel(root)
    top = Main(w)
    HelloWorld_support.init(w, top, *args, **kwargs)
    return (w, top)

def destroy_Main():
    global w
    w.destroy()
    w = None

class Main:
    def __init__(self, top=None):
        '''This class configures and populates the toplevel window.
        top is the toplevel containing window.'''
        _bgcolor = '#d9d9d9' # X11 color: 'gray85'
        _fgcolor = '#000000' # X11 color: 'black'
        _compcolor = '#d9d9d9' # X11 color: 'gray85'
        _anacolor = '#d9d9d9' # X11 color: 'gray85'
        _ana2color = '#ecec' # Closest X11 color: 'gray92'

        top.geometry("286x151+270+100")
        top.minsize(116, 1)
        top.maxsize(2110, 1418)
        top.resizable(1, 1)
        top.title("Hello World")
        top.configure(background="#d9d9d9")
        top.configure(highlightbackground="#d9d9d9")
        top.configure(highlightcolor="black")

        self.BtnSayHello = tk.Button(top)
        self.BtnSayHello.place(relx=0.35, rely=0.596, height=24, width=77)
        self.BtnSayHello.configure(activebackground="#ecec")
        self.BtnSayHello.configure(activeforeground="#000000")
        self.BtnSayHello.configure(background="#d9d9d9")
        self.BtnSayHello.configure(disabledforeground="#a3a3a3")
        self.BtnSayHello.configure(font="family {Segoe UI} -size 9")
        self.BtnSayHello.configure(foreground="#000000")
        self.BtnSayHello.configure(highlightbackground="#d9d9d9")
        self.BtnSayHello.configure(highlightcolor="black")
        self.BtnSayHello.configure(pady="0")
        self.BtnSayHello.configure(text="Say Hello")
        self.BtnSayHello.bind('<ButtonRelease-1>', lambda e: HelloWorld_support.BtnSayHello(e))

        self.TextHello = tk.Label(top)
        self.TextHello.place(relx=0.245, rely=0.199, height=21, width=134)
        self.TextHello.configure(background="#d9d9d9")
        self.TextHello.configure(disabledforeground="#a3a3a3")
        self.TextHello.configure(font="family {Segoe UI} -size 9")
        self.TextHello.configure(foreground="#000000")
        self.TextHello.configure(relief="sunken")
        self.TextHello.configure(text=" - - - ")

if __name__ == '__main__':
    vp_start_gui()
```

HelloWorld_support.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
#
# Support module generated by PAGE version 6.1
# in conjunction with Tcl version 8.6
# May 12, 2021 09:11:31 AM CEST platform: Windows NT

import sys

try:
    import Tkinter as tk
except ImportError:
    import tkinter as tk

try:
    import ttk
    py3 = False
except ImportError:
    import tkinter.ttk as ttk
    py3 = True

def init(top, gui, *args, **kwargs):
    global w, top_level, root
    w = gui
    top_level = top
    root = top

def BtnSayHello(p1):
    #print('HelloWorld_support.BtnSayHello')
    #sys.stdout.flush()
    w.TextHello["text"] = "Hello World !"

def destroy_window():
    # Function which closes the window.
    global top_level
    top_level.destroy()
    top_level = None

if __name__ == '__main__':
    import HelloWorld
    HelloWorld.vp_start_gui()
```

▲ Le fichier «HelloWorld_support.py» ci-dessus contient le code de **gestion des événements** de l'interface graphique. C'est aussi le **point d'entrée** pour exécuter le programme (dernières lignes du source).

Le source de ce fichier peut être modifié, le logiciel PAGE est capable de ne mettre à jour que le code des nouveaux événements créés.

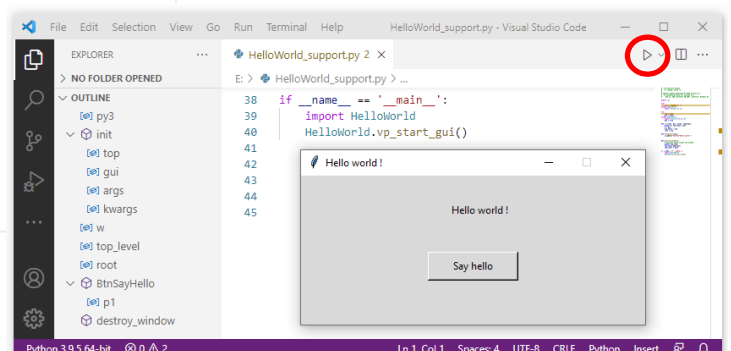
La seule modification apportée pour réaliser cet exemple est dans l'encadré rouge :

- mise en commentaire des 2 lignes ajoutées par PAGE pour tester le bouton (envoi d'un message sur la console Python);
- ajout d'une ligne de code pour afficher un message dans le label TextHello.

◀ Le fichier «HelloWorld.py» ci-contre contient le code de la création visuelle de l'interface graphique.

Le source de ce fichier ne peut pas être modifié, car à chaque évolution de l'interface graphique, son contenu est écrasé par PAGE.

La modification du source «HelloWorld_support.py» et son exécution (bouton ▶) s'effectuent dans Visual Studio Code.



ATTENTION : le logiciel PAGE a fréquemment des dysfonctionnements, sauvegardez régulièrement votre travail (Ctrl+S). En cas d'altération de la dernière sauvegarde, on peut retrouver la version précédente (nom du fichier marqué par #).

Utilisation de DAQmx en Python

National Instruments a publié une librairie de code, «**nidaqmx**», qui s'installe dans le noyau Python, et qui donne accès à toutes les fonctionnalités de la couche logicielle DAQmx à travers une interface objet.

Cette librairie est particulièrement bien documentée, mais il y a peu d'exemples d'utilisation disponibles.

Analyse d'un exemple de code NI

Bien que le source de cet exemple soit fourni, il est nécessaire de réaliser entièrement l'exemple pour bien comprendre le fonctionnement de la complétion de code.

```
F: > ai_voltage.py > ...
1  import nidaqmx                                import fonctions d'accès à DAQmx
2  from nidaqmx.constants import TerminalConfiguration  import constantes de configuration d'un terminal
3
4  with nidaqmx.Task() as task:  création de la tâche de mesure (objet task)
5      task.ai_channels.add_ai_voltage_chan("dev1/ai0:3", "", TerminalConfiguration.RSE)  config. 4 canaux mesures analogiques,
6                                                                                              AI0...AI3, en asymétrie référencé
7      print('\n4 Channels 1 Sample Read: ')
8      data = task.read()  lecture des mesures
9      print(data, "\n")
10  suppression automatique de la tâche de mesure en sortie du bloc de code "with" (suppression automatique de l'objet inutilisé)
```

TERMINAL

Copyright (C) 2009 Microsoft Corporation. Tous droits réservés.

```
4 Channels 1 Sample Read:
[2.4885995325858232, 5.094783770973304, 1.395648780277046, 1.3938510661481658]
PS F:\>
```

*AI0 connecté à la sortie 2,5V
AI1 connecté à la sortie 5V
AI2, AI3 non connectés (1,4V par défaut)*

Version alternative

La version de code précédente, très compacte, convient bien à un accès ponctuel aux mesures.

Comme avec LabVIEW, pour un accès cyclique, il faut éviter de créer/supprimer la tâche de mesure continuellement; ces deux opérations prennent jusqu'à 10 fois plus de temps que la seule opération de lecture des mesures.

Le code ci-dessous permet de séparer les 3 opérations d'accès aux mesures, comme cela se fait en LabVIEW.

```
F: > ai_voltage.py > ...
1  import nidaqmx
2  from nidaqmx.constants import TerminalConfiguration
3
4  def TaskOpen():
5      global task
6      task = nidaqmx.Task()
7      task.ai_channels.add_ai_voltage_chan("dev1/ai0:3", "", TerminalConfiguration.RSE)
8
9  def TaskRead():
10     global task
11     return task.read()
12
13  def TaskClose():
14     global task
15     task.close()
16
17  TaskOpen()
18
19  print('\n4 Channels 1 Sample Read: ')
20  print(TaskRead())
21
22  TaskClose()
```

Dans les deux exemples ci-dessus, la variable «task» n'est pas déclarée explicitement. C'est une spécificité du langage Python : les variables peuvent être créées automatiquement par l'interpréteur.

Le mot-clé «global» permet de déclarer la variable task «externe» dans chacune des 3 fonctions.

Il y a donc création d'une seule variable globale aux 3 fonctions.

L'exemple fourni mesure les temps d'exécution des deux versions de code.

Application Graphique de mesure

L'exemple présenté ci-après réalise la fusion des deux concepts présentés précédemment : créer une **Interface Utilisateur Graphique** (GUI) et effectuer des mesures via **DAQmx**.

Quelques éléments nouveaux sont mis en place :

- empêcher le redimensionnement de la fenêtre et la centrer;
- détecter automatiquement l'équipement de mesure (accès aux fonctions système de DAQmx);
- gérer les erreurs de mesure.

L'acquisition des mesures sera effectuée par le simple appui sur un bouton.

Bien que le logiciel PAGE permette de décocher les **options de redimensionnement** de la fenêtre, ces options ne sont **pas reportées dans le code source** de génération de la fenêtre. Et il n'y a **pas d'option de centrage** de la fenêtre.

Ces 2 absences sont facilement compensées en ajoutant 2 lignes à la fin de la fonction **init()** dans le fichier **TestDAQmx_support.py**

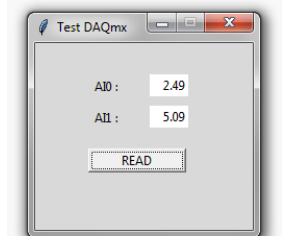
```
def init(top, gui, *args, **kwargs):  
    global w, top_level, root  
    w = gui  
    top_level = top  
    root = top  
    # new part =====  
    top.resizable(0, 0)  
    top.eval('tk:PlaceWindow . center')
```

Le code de gestion de l'événement du bouton contient le code de lecture et d'affichage des mesures, ainsi qu'une gestion simplifiée des erreurs d'accès aux mesures : si une erreur se produit dans le bloc **try**, le bloc **except** s'exécute.

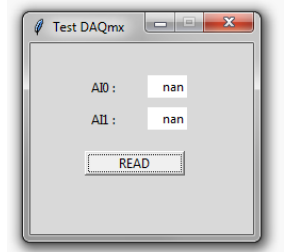
```
31 def BtnReadRelease(p1):  
32     import nidaqmx  
33     import nidaqmx.system  
34     from nidaqmx.constants import TerminalConfiguration  
35     import math  
36  
37     DevName = ""  
38     for device in nidaqmx.system.System.local().devices: boucle de lecture des devices présents dans DAQmx  
39         if device.product_type.find("USB-600") != -1:  
40             DevName = device.name  
41             break sortie de boucle dès la détection d'un device dont le  
42                                     nom de type contient "USB-600"  
43     with nidaqmx.Task() as task:  
44         try:  
45             task.ai_channels.add_ai_voltage_chan(DevName+"/ai0:1", "", TerminalConfiguration.RSE) lecture des voies AI0 et AI1  
46             ai0, ai1 = task.read() en asymétrique référencé  
47         except:  
48             ai0 = math.nan affectation de la valeur 'nan' (NotANumeric)  
49             ai1 = math.nan en cas d'erreur de mesure  
50  
51     w.AI0["text"] = "{:.2f}".format(ai0) affichage des mesures dans l'interface utilisateur  
52     w.AI1["text"] = "{:.2f}".format(ai1)
```

L'exécution du fichier TestDAQmx_support.py permet d'effectuer une mesure à chaque appui sur le bouton [READ].

(dans l'exemple ci-contre, AI0 est connecté à la sortie 2,5V, AI1 est connecté à la sortie 5V)



Lorsque le device de mesure est déconnecté, l'affichage indique 'nan' (Not A Numeric).



L'exemple fourni doit être complété (fonctions **init** et **BtnReadRelease**). Utilisez l'interpréteur Python 3.9 ou +

Acquisitions analogiques/booléennes/impulsions

Signaux analogiques périodiques et continus

L'exemple qui suit utilise la maquette de la **station météo** qui est équipée de **4 capteurs analogiques** produisant :

- **1 fréquence** : humidité

- **3 tensions continues** : girouette, température, luminosité

Pour lire la fréquence, il faut échantillonner le signal (=> tableau de N échantillons), et le faire aussi pour les 3 tensions continues (la mesure des 4 tensions analogiques doit être faite par la même tâche de mesure).

Dans l'exemple ci-dessous on ne mesure que **AI2** (connecté au capteur d'humidité) et **AI3** (connecté à la girouette). On obtient donc 2 tableaux de valeurs. Avec la fonction **mean()** on obtient la **moyenne** pour le tableau contenant une **tension continue**. Mais il n'y a aucune fonction permettant d'évaluer la période d'un signal dans un tableau.

Il est donc nécessaire de créer une fonction spécifique pour lire la **fréquence par détection des fronts**.

E: > Waveform.py > ...

Utilisez l'interpréteur Python 3.9 ou +

```
1 import nidaqmx
2 from nidaqmx.constants import TerminalConfiguration, AcquisitionType
3 import math
4 from statistics import mean
5
6
7 def GetArrayPeriod(trigger, data):
8     for i in range(0, len(data)):
9         if data[i] < trigger : break      #finding low state
10    if i == len(data)-1 : return math.nan  #'NaN' if not found
11
12
13
14    à compléter ...                        #finding first rising edge
15                                         #'NaN' if not found
16    FirstEdge = i
17
18
19
20    à compléter ...                        #finding low state
21                                         #'NaN' if not found
22
23
24
25    à compléter ...                        #finding second rising edge
26                                         #'NaN' if not found
27    SecondEdge = i
28
29    return (SecondEdge - FirstEdge)
30
31
```

Le principe esquissé dans ce source consiste à détecter alternativement les états bas et haut du signal sur une période. On retourne le nombre d'échantillons entre deux transitions bas-haut pour permettre ensuite le calcul de la période.

```
32 with nidaqmx.Task() as task:
33     NbSamples = 200
34     RateTiming = 5000
35     SampleTime = 1 / RateTiming
36     task.ai_channels.add_ai_voltage_chan(physical_channel = "dev1/ai2:3", terminal_config = TerminalConfiguration.RSE)
37     task.timing.cfg_samp_clk_timing (rate = RateTiming, sample_mode = AcquisitionType.FINITE, samps_per_chan = NbSamples)
38     Samples = task.read (number_of_samples_per_channel = NbSamples)
39
40 Frequency = 1 / (GetArrayPeriod(trigger=2, data=Samples[0]) * SampleTime)
41 Average = mean(Samples[1])
42
43 import os; os.system('cls')
44 print("Humidite : ", "{:.0f}".format(Frequency), " Hz")
45 print("Girouette : ", "{:.2f}".format(Average), " V ")
46
```

lecture AI2 et AI3, résultat dans un tableau 2D : AI2 >> Samples[0] AI3 >> Samples[1]

utilisation d'arguments "nommés" pour améliorer la lisibilité

calcul de la fréquence du signal mesuré sur AI2
calcul de la moyenne du signal mesuré sur AI3

TERMINAL

Humidite : 104 Hz
Girouette : 3.76 V

Dans cet exemple plusieurs sujets de programmation Python sont abordés : tableaux, boucle for, formatage. Les documents joints à ce cours fournissent de l'aide sur ces sujets (Python Cheat Sheet). Voir aussi les tutos dans la bibliographie.

Signaux booléens

La maquette de la station météo est équipée de **2 capteurs booléens** qui produisent :

- des **impulsions lentes**, contacteur du pluviomètre à bascule;
- un **mot de 4 bits**, encodeur rotatif.

Dans l'exemple ci-dessous, le **port P0** est lu, soit sous la forme de bits séparés, soit sous la forme d'un octet.
Lors de la mesure réalisée, toutes les lignes étaient à 1 sauf P0.0.

F: > Digital.py > ...

Utilisez l'interpréteur Python 3.9 ou +

```
1 import nidaqmx
2 from nidaqmx.constants import LineGrouping
3
4 with nidaqmx.Task() as task:
5     task.di_channels.add_di_chan(lines = "Dev1/port0/line0:7", line_grouping = LineGrouping.CHAN_PER_LINE)
6
7     data = task.read()
8     print("8 bits read: ", data)
9
10 with nidaqmx.Task() as task:
11     task.di_channels.add_di_chan(lines = "Dev1/port0/line0:7", line_grouping = LineGrouping.CHAN_FOR_ALL_LINES)
12
13     data = task.read()
14     print("1 byte read: ", data)
15
```

TERMINAL

```
8 bits read: [False, True, True, True, True, True, True, True]
1 byte read: 254
PS F:\
```

Ce source sera modifié pour effectuer la lecture des **4 bits du port P1 (encodeur) de la station météo**.

Signaux impulsionnels

La maquette de la station météo est équipée d'un capteur de **compte-tours** qui fournit un nombre d'impulsions/sec.

Dans l'exemple ci-dessous, le compteur sur l'**entrée PFI0**, connecté à un signal à 1KHz, est actionné pendant une seconde avant d'être lu, et fourni donc directement la fréquence en Hz.

F: > Counter.py > ...

```
1 import nidaqmx
2 from nidaqmx.constants import Edge
3 import time
4
5
6 with nidaqmx.Task() as task:
7     task.ci_channels.add_ci_count_edges_chan(counter = "dev1/ctr0", edge = Edge.FALLING).ci_count_edges_term = "/dev1/pfi0"
8     task.start()
9     time.sleep(1)
10    data = task.read()
11    print("Frequency= ", data , " Hz")
12
```

TERMINAL

```
Windows PowerShell
Copyright (C) 2009 Microsoft Corporation. Tous droits réservés.

Frequency= 1000 Hz
PS F:\
```

Ce source sera utilisé pour lire l'**anémomètre de la station météo**.

Boucles temporisées

La librairie **continuous_threading** facilite la mise en place d'un thread répétitif dont la **cadence** peut être définie. Dans l'exemple ci-dessous on crée deux boucles :

- une boucle rapide à **100ms**, associée à la variable **objet tf**, qui exécute la **fonction TimedLoopFast**
- une boucle lente à **1sec**, associée à la variable **objet ts**, qui exécute la **fonction TimedLoopSlow**

F: > TimedLoop.py > ...

Utilisez l'interpréteur Python 3.9 ou +

```
1 import continuous_threading
2 import time
3
4 def TimedLoopFast():
5     print("TimedLoop Fast")
6
7 def TimedLoopSlow():
8     print("TimedLoop Slow")
9
10 tf = continuous_threading.PeriodicThread(0.1, TimedLoopFast)
11 ts = continuous_threading.PeriodicThread(1.0, TimedLoopSlow)
12
13 print("Begin")
14 tf.start()
15 ts.start()
16
17 time.sleep(2)
18
19 tf.join()
20 ts.join()
21 print("End")
22
```

La méthode **start** démarre le thread.

La méthode **join** attend que le thread termine son traitement en cours, puis l'arrête.

L'exécution de ce source très simple permet d'afficher les appels répétitifs des deux boucles pendant 2 secondes.

TERMINAL

[illegible]

Graphes

Matplotlib

L'affichage des données sous la forme de graphes est essentiel en instrumentation.

Le «framework graphique Tk» ne contient aucun composant pour réaliser cette fonction; mais une librairie, très utilisée, permet l'affichage de graphes par programmation : **matplotlib**.

L'affichage «graphique» d'un signal périodique (comme celui fourni par le capteur d'humidité de la **station météo**) peut être facilement réalisé avec la librairie matplotlib en mode console.

Le code ci-dessous, réalise l'acquisition du capteur d'humidité sur l'entrée **AI2**, comme déjà vu précédemment.

E: > Matplotlib.py > ...

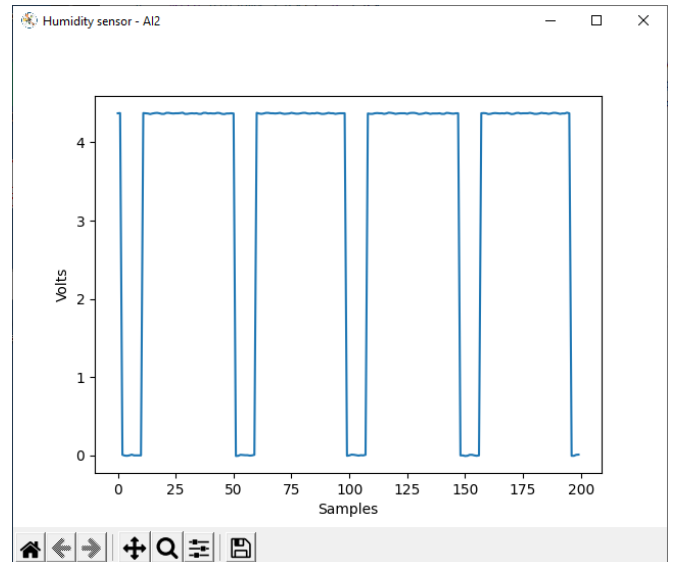
```
1 import nidaqmx
2 from nidaqmx.constants import TerminalConfiguration, AcquisitionType
3
4 with nidaqmx.Task() as task:
5     NbSamples = 200
6     RateTiming = 5000
7     task.ai_channels.add_ai_voltage_chan(physical_channel = "dev1/ai2", terminal_config = TerminalConfiguration.RSE)
8     task.timing.cfg_samp_clk_timing (rate = RateTiming, sample_mode = AcquisitionType.FINITE, samps_per_chan = NbSamples)
9     Samples = task.read (number_of_samples_per_channel = NbSamples)
10
11 import matplotlib.pyplot as plt
12 plt.figure("Humidity sensor - AI2")
13 plt.xlabel("Samples")
14 plt.ylabel("Volts")
15 plt.plot (Samples)
16 plt.show ()
17
```

Utilisez l'interpréteur Python 3.9 ou +

Quelques lignes suffisent, à la fin du code, pour afficher le signal dans une fenêtre.

La librairie «matplotlib» est essentiellement conçue pour visualiser et analyser des courbes dans le cadre d'une utilisation mathématique et scientifique.

Le résultat s'affiche dans une fenêtre indépendante (distincte de l'interface utilisateur de l'application) et sans que son contenu puisse être mis à jour dynamiquement à la façon d'un oscilloscope.



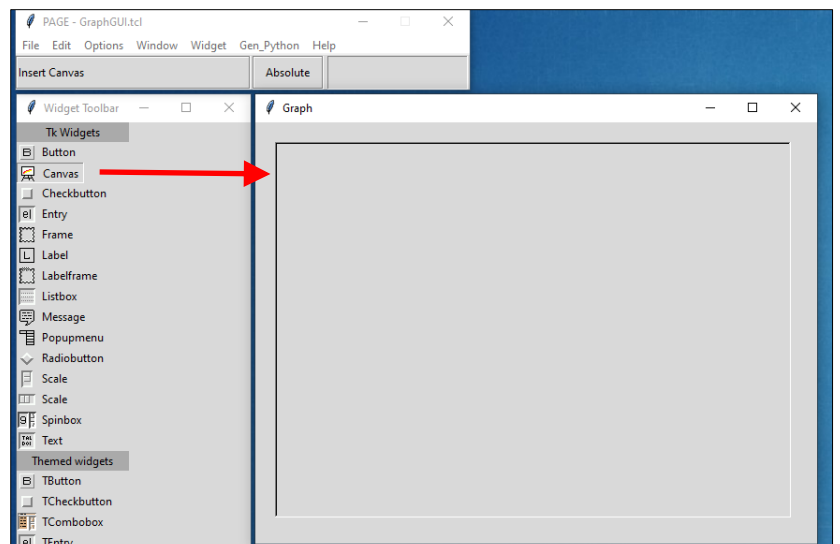
Graphe XY GUI

Bien que ce soit assez peu documenté (un seul exemple sans explication dans la documentation), il est possible d'intégrer un **graphe matplotlib dans une interface utilisateur** et de le mettre à jour **cycliquement**, à la manière d'un oscilloscope.

Il faut créer avec PAGE une interface utilisateur, très simple, puisque ne contenant qu'un seul composant de type **canvas** (Alias=Canvas1).

Ce composant est un container qui offre une surface sur laquelle il est possible de dessiner.

Le principe consiste à détourner la fonction de tracé de courbe de matplotlib vers la surface du composant Canvas1.



Le code de traçage du graphe est à peine plus compliqué, il est organisé en deux fonctions :

```
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
from matplotlib.figure import Figure

def GraphInit():
    global GraphSubPlot, GraphCanvas

    fig = Figure(dpi=100)
    fig.set_tight_layout({'pad' : 1})
    GraphSubPlot = fig.add_subplot()

    GraphCanvas = FigureCanvasTkAgg(fig, master=root)
    GraphCanvas.get_tk_widget().place(in_=w.Canvas1, relwidth=1, relheight=1)

def GraphPlot(ax, ay):
    global GraphSubPlot, GraphCanvas

    GraphSubPlot.clear()
    GraphSubPlot.plot(ax, ay)
    GraphSubPlot.set(xlabel="Time (ms)", ylabel="Volts", autoscaley_on=False, ylim=(-0.1, 5.2))
    GraphCanvas.draw()
```



composant de dessin de la courbe créé dans PAGE

L'acquisition doit être lancée cycliquement, on ne peut pas ouvrir/lire/fermer la voie de mesure à chaque appel. C'est donc un modèle de code en 3 fonctions qu'il faut mettre en place :

```
import nidaqmx
from nidaqmx.constants import TerminalConfiguration, AcquisitionType

def DAQmxOpen():
    global WaveformNbSamples, WaveformX, WaveformTask

    WaveformNbSamples = 200
    RateTiming = 5000
    SampleTime = 1 / RateTiming

    WaveformX = [0.0] * WaveformNbSamples
    for i in range(0, len(WaveformX)): WaveformX[i] = (SampleTime * i) * 1000

    WaveformTask = nidaqmx.Task()
    WaveformTask.ai_channels.add_ai_voltage_chan(physical_channel = "dev1/ai2", terminal_config = TerminalConfiguration.RSE)
    WaveformTask.timing.cfg_samp_clk_timing (rate = RateTiming, sample_mode = AcquisitionType.FINITE, samps_per_chan = WaveformNbSamples)

def DAQmxRead():
    global WaveformTask, WaveformNbSamples, WaveformY

    WaveformY = WaveformTask.read(number_of_samples_per_channel = WaveformNbSamples)

def DAQmxClose():
    global WaveformTask

    if WaveformTask is not None: WaveformTask.close()
    WaveformTask = None
```

La gestion de la boucle se décline en trois fonctions qui appellent les fonctions d'acquisition et de tracé de la courbe :

```
import concurrent.futures
import threading

def TimedLoopStart():
    global th

    DAQmxOpen()
    GraphInit()

    th = concurrent.futures.ThreadPoolExecutor(max_workers=1)
    th.submit(TimedLoopJob)
    th.start()

def TimedLoopJob(): # 100ms loop
    global WaveformX, WaveformY

    DAQmxRead()
    GraphPlot(WaveformX, WaveformY)

def TimedLoopStop():
    global th

    th.shutdown(wait=True)
    DAQmxClose()
```


Les fonctions décrites précédemment sont insérées dans le fichier **GraphGUI_support.py** généré par PAGE. Ainsi que l'appel des deux fonctions de gestion du thread d'acquisition/affichage des mesures :

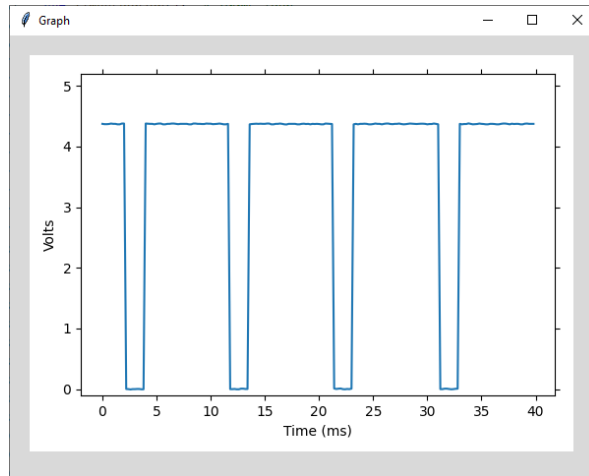
TimedLoopStart() et **TimedLoopStop()**.

```
def init(top, gui, *args, **kwargs):
    global w, top_level, root
    w = gui
    top_level = top
    root = top
    TimedLoopStart()

def close(p1):
    TimedLoopStop()

def destroy_window():
    # Function which closes the window.
    global top_level
    top_level.destroy()
    top_level = None

if __name__ == '__main__':
    import GraphGUI
    GraphGUI.vp_start_gui()
```



Utilisez l'interpréteur Python 3.9 ou +

Le thread de gestion de l'interface utilisateur démarre le thread de la boucle temporisée par la fonction **init()**. L'arrêt du thread de la boucle temporisée (qui effectue des affichages) par la fonction **destroy_window()** arriverait trop tardivement : les éléments visuels de l'interface utilisateur ont déjà été supprimés de la mémoire. C'est la raison de l'ajout d'un événement **close** dans PAGE, déclenché par le bouton de fermeture de la fenêtre.

Graphe déroulant GUI

L'affichage «graphique» d'un signal qui se déroule lentement dans le temps utilise le même principe que précédemment, mais cette fois-ci c'est une mesure isolée qui est envoyée à la fonction d'affichage et qui maintient à jour un **tableau d'historique des mesures**.

L'interface utilisateur est absolument identique à l'exemple précédent.

Comme dans l'exemple précédent, la gestion du graphe est assurée par seulement 2 fonctions :

```
#-- graph functions -----
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
from matplotlib.figure import Figure

def ChartInit():
    global ChartSubPlot, ChartCanvas, ChartNbPoints, ChartHistoric

    ChartNbPoints = 100
    ChartHistoric = []

    fig = Figure(dpi=100)
    fig.set_tight_layout({'pad' : 1})
    ChartSubPlot = fig.add_subplot()

    ChartCanvas = FigureCanvasTkAgg(fig, master=root)
    ChartCanvas.get_tk_widget().place(in_=w.Canvas1, relwidth=1, relheight=1)

def ChartPlot(value):
    global ChartSubPlot, ChartCanvas, ChartNbPoints, ChartHistoric

    if len(ChartHistoric) >= ChartNbPoints:
        ChartHistoric.pop(0)
        ChartHistoric.append(value)

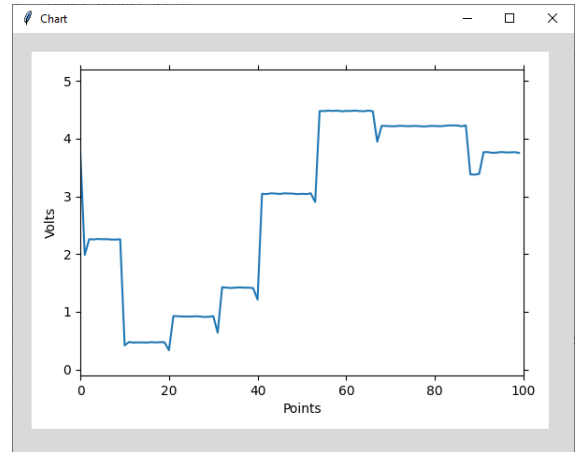
    ChartSubPlot.clear()
    ChartSubPlot.plot(ChartHistoric)
    ChartSubPlot.set(xlabel="Points", ylabel="Volts", autoscalex_on=False, xlim=(0, ChartNbPoints), autoscaley_on=False, ylim=(-0.1, 5.2))
    ChartCanvas.draw()
```

Au démarrage, le tableau d'historique est vide; les points de mesure sont ajoutés, successivement, à la fin du tableau. Lorsque le nombre maximum de points à afficher est atteint, le point au début du tableau est supprimé avant d'en ajouter un nouveau à la fin. C'est ce qui provoque le «défilement» de la courbe.

L'acquisition des mesures et la gestion de la boucle temporisée sont très similaires à l'exemple précédent.

Au final, le fichier **ChartGUI_support.py** ressemble beaucoup au fichier GraphGUI_support.py.

L'exécution de ce programme permet d'afficher la courbe de variation de la **girouette de la station météo** (entrée **AI3**).



Utilisez l'interpréteur Python 3.9 ou +

Gestion des erreurs

La librairie nidaqmx génère, comme en LabVIEW, une erreur lorsqu'un accès au matériel ne peut être réalisé.

La gestion des **erreurs en Python**, appelées **exceptions**, est réalisée avec la structure **try ... except**. La génération d'une exception est produite avec l'instruction **raise**.

Le source ci-dessous réalise une simple lecture de la voie **AI0**.

```
import nidaqmx
import nidaqmx.system
from nidaqmx.task import Task
from nidaqmx.constants import TerminalConfiguration

vTask : Task = None

import os; os.system('cls')

try:
    DevName = ""
    for device in nidaqmx.system.System.local().devices:
        if device.product_type.find("USB-600") != -1:
            DevName = device.name
            break
    if DevName == "": raise(nidaqmx.DaqError("Device auto-search failed", -200220, "No task"))

    vTask = nidaqmx.Task("VoltTask")
    vTask.ai_channels.add_ai_voltage_chan (physical_channel = DevName+"/ai0", terminal_config = TerminalConfiguration.RSE)

    print("Volt = ", vTask.read())

    vTask.close()
except nidaqmx.DaqError as e:
    print(e.__str__())
    if vTask != None: vTask.close()
```

Lorsqu'une erreur se produit dans le bloc de code **try**, l'exécution du bloc s'arrête, et le bloc **except** est exécuté.

Si la recherche du nom du device échoue, aucune instruction ne provoque d'erreur.

Afin de déclencher un seul type de traitement d'erreur, on «provoque» une erreur avec l'instruction **raise**.

Exécuter ce source dans les situations suivantes :

- | | |
|--|---|
| • mesure avec le boîtier connecté | affichage d'une tension de 1,4V (AI0 n'est pas connectée) |
| • mesure d'une voie inexistante (AI9) | affichage d'une erreur générée par la librairie nidaqmx |
| • mesure avec le boîtier USB déconnecté | affichage de l'erreur générée par l'instruction raise |

Gestion des fichiers

L'écriture cyclique de données dans un fichier nécessite, comme pour les acquisitions, un **traitement en 3 phases**. De même, les opérations sur les fichiers pouvant générer des erreurs, les **exceptions** doivent être traitées.

Dans l'exemple ci-dessous, **file** est la variable objet créée lors de l'ouverture du fichier, et utilisée par les opérations d'écriture et de fermeture du fichier.

```
import os; os.system('cls')

#-- open file -----
try:
    file = open('e:\MyFile.txt', 'a')
except IOError:
    file = None
    print("Open Error")

#-- write file -----
try:
    if (file != None):
        file.write("Hello world !\n")
        file.flush
except IOError:
    print("Write Error")

#-- close file -----
try:
    if (file != None):
        file.close()
        file = None
except IOError:
    print("Close Error")
```

Exécuter ce source dans les situations suivantes :

- utiliser un chemin de fichier inexistant
- chemin correct + attribut 'a' (append)
- attribut 'r' (read)

affiche 'Open Error'

fichier créé avec 1 ligne, puis ajout 1 ligne à chaque exécution

affiche 'Write Error'

Bibliographie

Python

- Python Wiki : <https://wiki.python.org/moin/>
Python Documentation : <https://docs.python.org/3/contents.html>

IDE

- Visual Studio Code : <https://code.visualstudio.com/>
Python Tools for VS : <https://docs.microsoft.com/fr-fr/visualstudio/python/overview-of-python-tools-for-visual-studio>
Getting Started With Python : <https://code.visualstudio.com/docs/python/python-tutorial>

GUI

- PAGE, Tk GUI Generator : <http://sourceforge.net/projects/page/>
Using PAGE : <http://page.sourceforge.net/html/use.html>
Tkinter : <https://wiki.python.org/moin/TkInter>
Tkinter Documentation : <https://tkdocs.com/tutorial/index.html>

Librairies

- Python Package : <https://pypi.org/>
NumPy : <https://numpy.org/doc/1.20/user/index.html>
SciPy : <https://docs.scipy.org/doc/scipy/reference/>
Matplotlib : <https://matplotlib.org/stable/users/index.html>
Continuous Threading : <https://pypi.org/project/continuous-threading/>

NI-DAQmx

- Control Device with NI-DAQmx : <https://knowledge.ni.com/KnowledgeArticleDetails?id=kA00Z0000019Pf1SAE&l=fr-FR>
Ressources Python : <https://www.ni.com/fr-fr/support/documentation/supplemental...>
NI-DAQmx Python Doc : <https://nidaqmx-python.readthedocs.io/en/latest/index.html>

Tutoriaux, ressources

- Python Doctor (FR) : <https://python.doctor>
Développez Python (FR) : <https://python.developpez.com/tutoriels/cours-python-uni-paris7/>
Tutorials Point : <https://www.tutorialspoint.com/python/index.htm>
w3schools : https://www.w3schools.com/python/python_intro.asp
Real Python : <https://realpython.com/>
Python Guides : <https://pythonguides.com/learn-python/>
Stack Exchange : <https://stackapps.com/questions/tagged/python>
Stack Overflow : <https://stackoverflow.com/questions/tagged/python>

Quick Python References

- <https://quickref.me/python>
<https://www.cs.put.poznan.pl/csobaniec/software/python/py-grc.html>
<https://perso.limsi.fr/poitala/media/python:cours:mementopython3-english.pdf>
http://sixthresearcher.com/wp-content/uploads/2016/12/Python3_reference_cheat_sheet.pdf