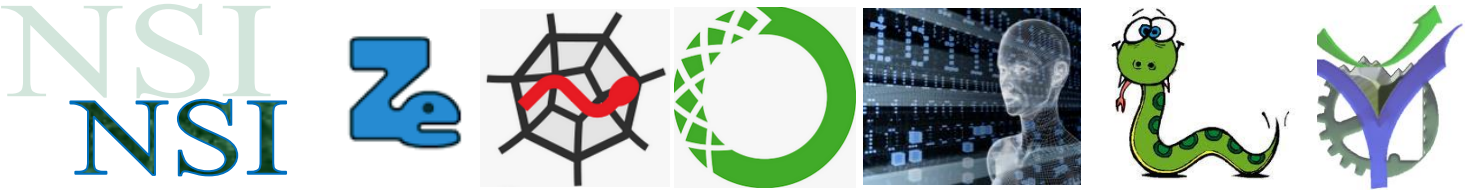


Approfondissement POO

1 Question les attributs privés, protégés, public

https://www.pierre-giraud.com/python-apprendre-programmer-cours/oriente-objet-visibilite-attribut/

1.1 Les niveaux de visibilité des membres de classe en programmation orientée objet

Dans la plupart des langages qui supportent l'orienté objet, on peut définir des "niveaux de visibilité" ou "niveaux d'accès" pour les membres des classes (c'est-à-dire pour les variables et les fonctions).

En POO, on distingue généralement trois niveaux de visibilité différents :

- Les membres privés auxquels on ne peut accéder que depuis l'intérieur de la classe ;
- Les membres protégés auxquels on ne peut accéder que depuis l'intérieur de la classe ou depuis une classe fille ;
- Les membres publics auxquels on peut accéder depuis n'importe quelle instance (ou objet) de la classe ou d'une classe fille.

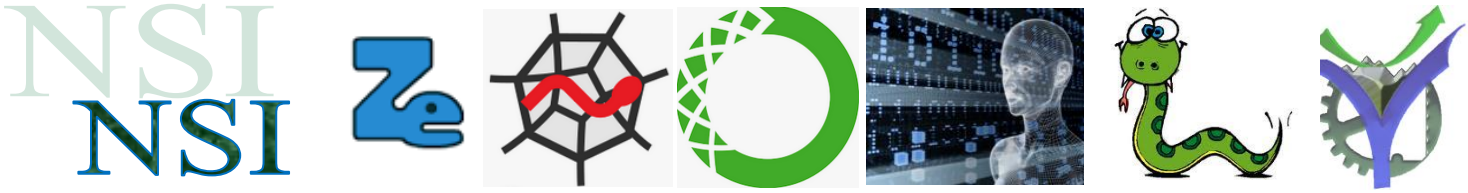
Ces niveaux de visibilité permettent de "protéger" certains membres de classes qui ne devraient pas être modifiés dans n'importe quelle situation ou depuis n'importe quel endroit.

1.2 Les niveaux de visibilité des membres de classe en Python

Python n'implémente pas directement ces concepts de visibilité des membres de classe et il est donc impossible de définir un membre comme privé ou protégé : par défaut, tous les membres de classe sont publics en Python.

En revanche, certaines conventions ont été mises en place par la communauté Python, notamment au niveau des noms des membres de classe qui servent à indiquer aux autres développeurs qu'on ne devrait accéder à tel membre que depuis l'intérieur de la classe ou à tel autre membre que depuis la classe ou une classe fille.

Attention ici : ce ne sont que des conventions qui n'ont aucun équivalent réel en Python : on va pouvoir informer d'autres développeurs du niveau de visibilité souhaité pour un membre mais tous les membres seront toujours publics en Python et il est de la responsabilité des autres développeurs de suivre nos indications ou pas.



Ces conventions sont les suivantes :

- On préfixera les noms des membres qu'on souhaite définir comme "privés" avec deux underscores comme ceci : __nom-du-membre ;
- On préfixera les noms des membres qu'on souhaite définir comme "protégés" avec un underscore comme ceci : _nom-du-membre.

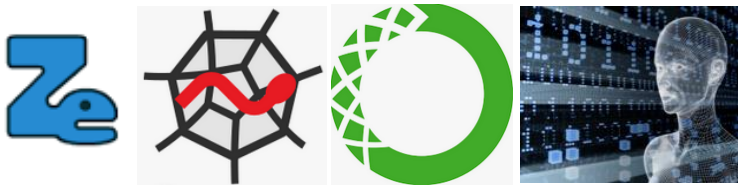
Il est à noter que ces conventions n'ont pas été adoptées par hasard. En effet, Python possède un mécanisme appelé "name mangling" qui fait que tout membre de classe commençant par deux underscores, c'est-à-dire de la forme __nom-du-membre sera remplacé textuellement lors de l'interprétation par _nom-de-classe__nom-du-membre.

Cela fait que si un développeur essaie d'utiliser un membre défini avec deux underscores tel quel, Python renverra une erreur puisqu'il préfixera le nom avec un underscore et le nom de la classe du membre.

Cette règle a été prévue par Python pour éviter les accidents de conflits entre plusieurs membres de plusieurs classes qui auraient le même nom. Elle n'empêche pas d'accéder ou de modifier un membre "privé". En effet, il suffit de préfixer le membre de la même façon que Python lors de son interprétation pour y accéder.

2 Attraper une exception

https://www.pierre-giraud.com/python-apprendre-programmer-cours/gestion-exception-try-except-else/



3 Illustration pratique

Attributs_POO_01.py

```python
class Visibilite:
    '''
    Démonstration de la non gestion par python de la protection des attributs

    public = "Variable publique = accès pour tous"
    _protected = "Variable protégée = accès depuis la classe ou d'une classe fille"
    __private  = "Variable privée = accès uniquement à l'intérieur de la classe"

    '''
    public = "Variable publique"
    _protected = "Variable protégée"
    __private  = "Variable privée"

# Création d'une instance de la classe
print("Création d'une instance de la classe Visibilite")
objet = Visibilite()
print()

# Tentative d'accès aux trois attributs directement
# L'attribut publique
print("Essai de lecture de son attribut public")
print("Lecture OK : ",objet.public)
print()

# L'attribut 'protégé'
print("Essai de lecture de son attribut 'protégé'")
print("Lecture OK : ",objet._protected)
print()

# L'attribut 'privé'
print("Essai de lecture de son attribut privé directement")
try:
    print(objet.__private)
except:
    print("Accès impossible")
print()

print("Essai de lecture de son attribut privé avec son nom complet")
# L'attribut privé avec la syntaxe de son nom
# avec objet._Visibilite
print("Lecture OK : ",objet._Visibilite__private)
```

```
Création d'une instance de la classe Visibilite

Essai de lecture de son attribut public
Lecture OK :  Variable publique

Essai de lecture de son attribut 'protégé'
Lecture OK :  Variable protégée

Essai de lecture de son attribut privé directement
Accès impossible

Essai de lecture de son attribut privé avec son nom complet
Lecture OK :  Variable privée
```