1. SAE15 : TD2-Traiter des données en Python

Ce TD est à faire en binôme et vous guidera dans la manipulation en Python de données numériques.

Pour aller plus loin, il existe le module `Python numpy` qui permet d'accélérer et de faciliter les traitements des données numériques.

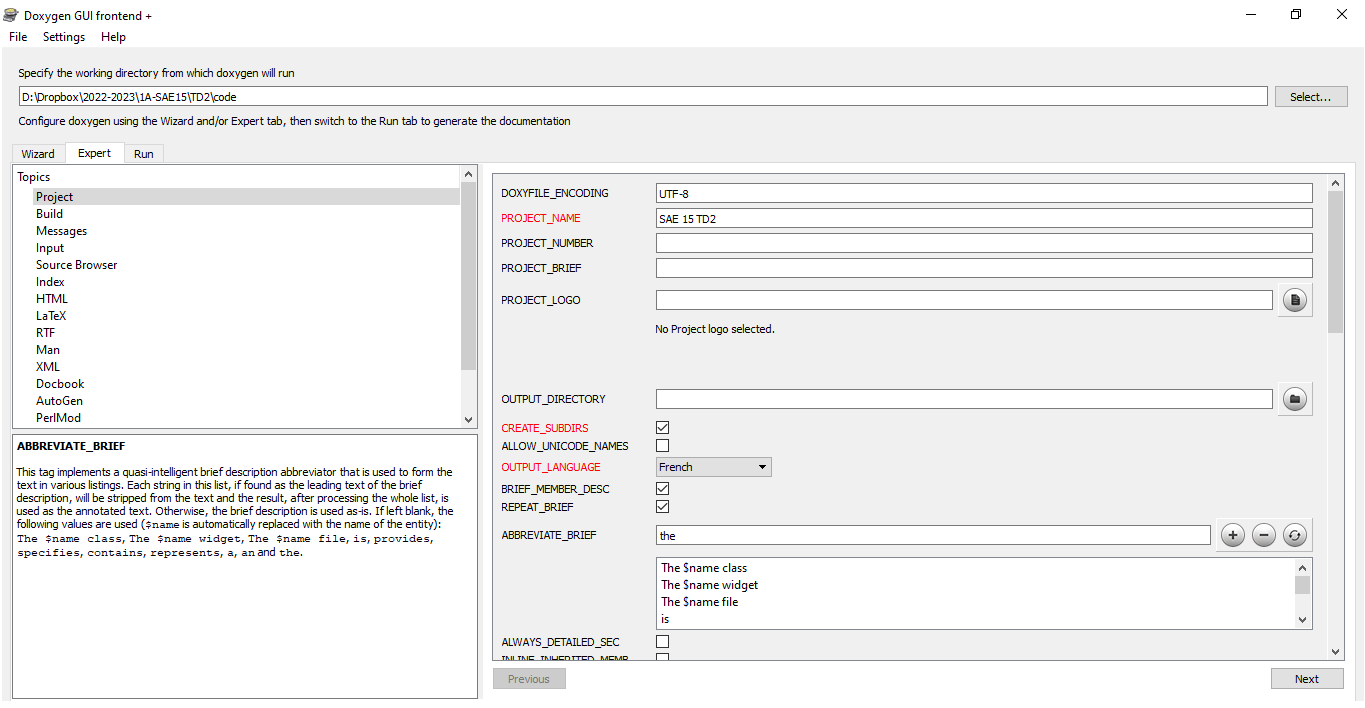
Nous allons nous concentrer sur la réalisation d'algorithmes pour extraire des données (importer des données `CSV` vers un programme `Python`) et exploiter ces données (calculer des moyennes, ...).

2. Prenez soin de bien documenter votre code

Plusieurs outils existent pour générer des documentations techniques à partir de code source. La documentation de votre code est indispensable. Il faut documenter les différentes fonctions mais aussi les points importants de votre code.

En bonus, vous pourrez utilisez Doxygen (https://www.doxygen.nl/index.html) pour générer la documentation des modules que vous écrirez. Ce logiciel est disponible pour toutes les plateformes `Windows`, `Linux`, `OSX` et peut être utilisé en mode commande depuis un terminal ou bien à travers son interface graphique (`Doxywizard`). La figure suivante illustre l'interface graphique de cet outil à travers lequel les informations suivantes doivent être fournies :

- nom d'un projet de documentation,
- répertoire contenant les sources du code à analyser,
- répertoire de destination de la documentation,
- format de la documentation (`rtf`, `html`, `latex`, `etc`.), html sera privilégié.



Pour que `Doxygen` puisse analyser vos commentaires, il faut qu'ils respectent un certain format. Voici un exemple montrant comment documenter les fonctions et procédures de vos modules ainsi que les autres

modules utilisés, le principal étant la documentation des prototypes des fonctions/procédures. Pour qu'un commentaire soit pris en compte par Doxygen, il suffit de commencer le commentaire par le code suivant : """! Des commentaires peuvent tout d'abord être ajoutés au début du fichier contenant le module pour décrire :

- son objectif à travail une description introduite par le tag @brief,
- des sections (tag @section) pour décrire l'auteur, les modifications, les todo-listes, etc.

Chaque variable peut également être décrite de nouveau en la précédant du code : """! Le plus important est de commenter les fonctions et procédures avec une description textuelle du rôle de la fonction/procédure et une description de chaque paramètre et de l'éventuelle retour.

- la description de la fonction est introduite par le tag @brief,
- la description d'un paramètre suit le tag @param où apparaît le type, le nom et la description du paramètre,
- la description du retour @return où apparaît le type et la description du résultat de la fonction.

```
"""! @brief Traitement de données TD2 SAE15 2023-2024 1-Version avec liste de
liste de 2 éléments

@package td2_1
Package td2_1

@file td2_1.py
1-Version avec la liste de liste de 2 éléments

@version 1.0

@section td2_1 TODO
- faire les assert

@section author_libraries_devises Author(s)
- Crée par Philippe Durand le 01/07/2022.
- Copyright (c) 2023 IUT de Lannion. All rights reserved.

"""

def valeurMin(valeurs : list  ) -> int:
    """!
    Fonction qui calcule la valeur min d'une liste,
    @param valeurs list la liste des valeurs
    @return int la valeur min
    """
```

SAE 15 TD2

[Lien utile pour Doxygen] http://tvaira.free.fr/projets/activites/activite-documentation-doxygen.html

3. Récupération des données en Python de la taille d'un dossier

Faites un premier fichier td2_1.py

3.1 Lecture depuis un fichier CSV et import dans une liste

A partir du fichier `resTailleDossier.csv` généré lors du TD1 Vous allez désormais écrire une fonction python `lectureFichier` qui retournera la liste contenant les données. Un élément de la liste sera une sous liste d'entiers `[timestamp, donnée]`.

```
[['5', '1656509158'], ['2', '1656509239'], ['3', '1656509240'], ['4',
'1656509241'], ['5', '1656509242']]
```

Vous trouverez ci-dessous un rappel du code pour charger dans une liste l'ensemble des données présentes dans le fichier `resTailleDossier.csv`.

```
with open(fichier, newline='') as csvfile :
        datareader = csv.reader(csvfile, delimiter= ',', quotechar='|')
        for ligne in datareader :
            #print(ligne)
```

Testez votre fonction dans la partie principale du programme (main)

3.2 Transposition de la liste

Afin de faire des statistiques et plus tard de l'affichage, il est utile de définir une fonction `transposer` qui va permettre d'obtenir la liste de chaque élément dans une sous liste. Le nombre de sous listes dépendra du nombre d'éléments collectés (deux dans l'exemple présenté)

```
[[5, 2, 3, 4, 5], [1656509158, 1656509239, 1656509240, 1656509241, 1656509242]]
```

Testez votre fonction dans la partie principale du programme (main)

3.3 Calcul de statistiques

Dans cette seconde partie, vous allez définir quelques fonctions pour calculer des statistiques sur une liste de données (taille fichier ici). Voici quelques exemples de statistiques utiles :

- la valeur minimum
- la valeur maximum
- la moyenne arithmétique
- la valeur médiane
- la variance
- l'écart type
- et les trois bornes délimitant les quartiles.

Aide : Pour identifier la valeur médiane et les bornes des quartiles, il faut au préalable trier le tableau par ordre croissant. Utiliser la fonction sorted comme illustré ci-dessous :

```
#pour trier la liste dans l'ordre croissant
listetriee : List = sorted(listeatrier)

[5, 2, 3, 4, 5] => [2, 2, 3, 4,5, 5]
```

Testez vos fonctions dans la partie principale du programme (main)

4. Récupération des données en Python des tailles des fichiers d'un dossier

Faites un fichier td2_2.py

4.1 Lecture depuis un fichier CSV et import dans un dictionnaire

A partir du fichier `resTaillesFichiers.csv` généré lors du TD1 Vous allez désormais écrire une fonction python `csvToDict` qui retournera un dictionnaire qui associera une clé à chaque nom de fichier. Chaque clé sera liée à une liste qui contiendra ici des sous listes d'entiers `[ donnée,timestamp]`.

```
def csvToDict(f : str) -> dict[str,list[list[int]]]:
    """!
    Fonction chargeant les données stockées dans un fichier csv avec un séparateur
`,` et retournant un dictionnaire où les clés sont les différents noms de fichiers
décrits et les valeurs [taille en ko, timestamp] sont stockées dans une liste de
listes
    @param fichier (str): Le nom du fichier à charger
    @return dict[str,list]: les clés sont les noms des fichiers observés et les
valeurs une liste de dimension 2 [[taille,timestamp]]
    """
```

Pour construire l'élaboration de cette fonction, vous aurez besoin des éléments techniques suivants :

- initialiser une liste de listes lorsque vous rencontrez pour la première fois un nom de fichier (`f` avec ses valeurs `[taille1, temps1]`):

```
liste : list = [[ ??? , ??? ]] # on crée une liste à deux dimensions
dictionnaire[cle] =  liste
#Attention à bien stocker des valeurs numériques dans votre liste
```

- ajouter une nouvelle entrée dans la liste des valeurs pour un fichier déjà connu (càd. qui possède déjà une clé dans le dictionnaire à construire).

```
dictionnaire[cle].append([???, ???]) # on crée une liste simple
```

```
Exemple de fichier CSV

f1.txt,0,1637092438
f2.txt,0,1637092438
f3.txt,0,1637092438
f1.txt,1000,1637092439
f2.txt,1000,1637092439
f3.txt,3000,1637092439

#Exemple de dictionnaire à produire
{
    'f1.txt': [[0, 1637092438], [1000, 1637092439]],
    'f2.txt': [[0, 1637092438], [1000, 1637092439]],
    'f3.txt': [[0, 1637092438], [3000, 1637092439]]
}
```

Testez votre fonction dans la partie principale du programme (main) en affichant le dictionnaire obtenu.

4.2 Observation de l'évolution de la taille des fichiers

Les données manipulées décrivent l'évolution de la taille de fichiers selon un intervalle de temps constant (une minute logiquement : 60 secondes). Vous allez construire deux fonctions qui prennent toutes les deux en paramètre une liste des données d'un fichier et qui retournent respectivement les informations suivantes :

- l'augmentation moyenne de la taille du fichier entre deux captations,

```
def augmentationMoyenne(valeurs : list[int]) -> float:
    """!
    Fonction qui calcule l'augmentation moyenne de la taille d'un fichier entre
deux captations
    @params: valeurs (list): les valeurs successives de taille d'un fichier
    @returns: float la valeur de l'augmentation moyenne dans l'unité de saisie
    """
```

- la plus grande augmentation de taille lors de la période d'évaluation retournée sous forme d'un couple (augmentation, timestamp)

```
def plusGrandeAugmentation(valeurs : list[list[int]]) -> tuple[float,int]:
    """calcule la plus grande augmentation de taille lors de la période
d'évaluation retournée sous forme d'un couple (augmentation, timestamp).
    @params:valeurs (list): les valeurs successives de taille d'un fichier et son
timestamp
    @returns: tuple qui contient la valeur de l'augmentationMax et le timestamp
(augmentationMax, timestamp).
    """
```

Testez vos fonctions dans la partie principale du programme (main).

4.3 Détection de valeurs limites

Vous allez maintenant construire une nouvelle fonction qui prend cette fois-ci deux paramètres : une liste des données d'un fichier et une taille limite exprimée en kilooctets. La fonction retourne la date (i.e. le timestamp) à partir duquel le fichier a atteint la taille limite.

```
def limiteAtteinte(valeurs : list[list[int]], limite : int) -> int or None:
    """!
    Détermine la date où la taille limite a été atteinte
    @params:valeurs (list): les valeurs successives de taille d'un fichier et son
timestamp
    @params:limite (int) une taille limite exprimée en kilooctets
    @returns: (int) la date au format timestamp
    """
```

Faites un test depuis votre programme principal en affichant pour chaque fichier observé, la date à laquelle il a dépassé une taille plausible dans les données manipulées.

Au lieu d'afficher un timestamp au format EPOCH, vous utiliserez l'instruction ci-dessous pour obtenir une date plus lisible :

```
datetime.utcfromtimestamp(tpsLimite).strftime('%d-%m-%Y %H:%M:%S')
```

Affichage final :
Complétez le programme principal de votre script pour afficher pour chaque fichier les informations suivantes :

- le nom du fichier
- l'augmentation moyenne de sa taille à chaque captation et la date de la plus grande augmentation (avec l'augmentation elle-même),
- la date à laquelle le fichier a dépassé un seuil pertinent

```
Augmentation moyenne du fichier f1.txt 1000
Augmentation Max du fichier f1.txt (1000, 1637092439)
Limite 4000 atteinte du fichier f1.txt au temps 16-11-2021 19:54:23
Augmentation moyenne du fichier f2.txt 3200
Augmentation Max du fichier f2.txt (8000, 1637092463)
Limite 4000 atteinte du fichier f2.txt au temps 16-11-2021 19:54:22
Augmentation moyenne du fichier f3.txt 300
Augmentation Max du fichier f3.txt (1000, 1637092463)
```

5. Regroupement de données (Travail personnel pour votre SAé)

A faire dans le fichier td2_1.py

Dans la liste construite lors de la question précédente, vous devez avoir un élément pour chaque minute d'observation de la taille de votre répertoire personnel.

Vous allez désormais construire une fonction `obtenirDonneesParDuree` de regroupement de ces données selon une durée. La fonction va prendre en paramètre la liste des données et cette durée exprimée en minute. Cela permettra par exemple de regrouper les informations par tranches de 5 minutes ou d'une heure. Vous aurez besoin du module datetime pour récupérer une date à partir du temps epoch.

Voici un exemple qui définit deux dates et calcule leur différence en minutes:

```
from datetime import datetime
#un timestamp
d = datetime.fromtimestamp(1634920610)
d2 = datetime.fromtimestamp(1634920915)
#différence entre les deux dates en minuts
minutes_diff = (d2 - d).total_seconds() / 60
```

Les données présentes dans la liste passée en paramètre seront donc regroupées par tranche de `X` minutes (où X est la valeur second paramètre `duree` de la fonction).

```
def obtenirDonneesParDuree(liste : list[list[int]], duree : int) ->
list[list[int]] :
    """!
    Fonction qui retourne une liste à deux dimensions  ,
    @param liste List la liste des valeurs ex : [['5', '1656509238'], ['2',
'1656509339'], ['3', '1656509440'], ['4', '1656509541'], ['5', '1656509642']]
    @param duree int la durée d'observation depuis le début
    @return list la liste contient des listes des valeurs sur la durée
    exemple de retour [['5', '1656509238'], ['2', '1656509339'], ['3',
'1656509440']]
    """
```

Testez votre fonction dans la partie principale du programme (main).