Langage Python A3 TD6

En principe tous les modules utilisés dans ce TD font partie de la distribution anaconda. Du coup directement accessible sous spyder.

Exercice 1 : Donnée infoclimat

Il s'agit de tracer les températures à Paris sur une semaine via le webservice du site https://www.infoclimat.fr/
Infoclimat est une association à but non lucratif, essayez de ne pas "bombarder" le site par des requêtes répétées. Astuce spyder, favorisez le recours à **run current cell** et **run selection** pour ne pas exécuter votre script complet à chaque essai.

Infoclimat vous permet de récupérer les données de prévisions météo de Paris à 7 jours en JSON, XML, CSV. **Nous allons utiliser JSON** via l'API Infoclimat, qui nous retournera les prévisions détaillées pour Paris.
Le lien de l'API JSON et la description des données sont disponible ici:
https://www.infoclimat.fr/api-previsions-meteo.html?id=2988507&cntry=FR

a-Importer les modules suivant:

```
import json
import requests
```

b-Récupérez le lien de l'API JSON est affecté le à la variable url

```
url="http://www.infoclimat.fr/public-api/gfs/json?_ll=48.8534...
```

c-Ecrivez et testez le code suivant, et observez bien l'utilisation de json

```
try:
    api_request=requests.get(url)
    data=json.loads(api_request.content)
except Exception as e:
    data="Error..."
```

d-Vérifiez le type(data), il s'agit bien d'un dictionnaire, Explorez data. (for k in data, for k, v in..)
e-Vérifiez data des clés suivantes : request_state, request_key, message, model_run, source
f-Supprimez-les de votre dictionnaire.

g-Explorez la valeur pour une clé (texte représentant une date) et **vérifiez** bien qu'il s'agit d'un dictionnaire également.
Comme par exemple:

```
In [152]: print(data['2021-04-06 14:00:00'])
{'temperature': {'2m': 279, 'sol': 278, '500hPa': -0.1, '850hPa': -0.1},
'pression': {'niveau_de_la_mer': 101170}, 'pluie': 0.6, 'pluie_convective': 0.6,
'humidite': {'2m': 47.7}, 'vent_moyen': {'10m': 19.5}, 'vent_rafales': {'10m':
23.9}, 'vent_direction': {'10m': 310}, 'iso_zero': 534, 'risque_neige': 'oui',
'cape': 0, 'nebulosite': {'haute': 0, 'moyenne': 0, 'basse': 0, 'totale': 59}}
```

h-Comment accéder à la température à 2m à cette date? La température au sol?
i-Comment accéder à l'humidité?

j-Importer datetime du module datetime

```
from datetime import datetime
```

Pour convertir une chaine de caractère de cette forme '2021-04-06 14:00:00' en un objet datetime nous pouvons utiliser la fonction datetime.strptime comme par exemple:

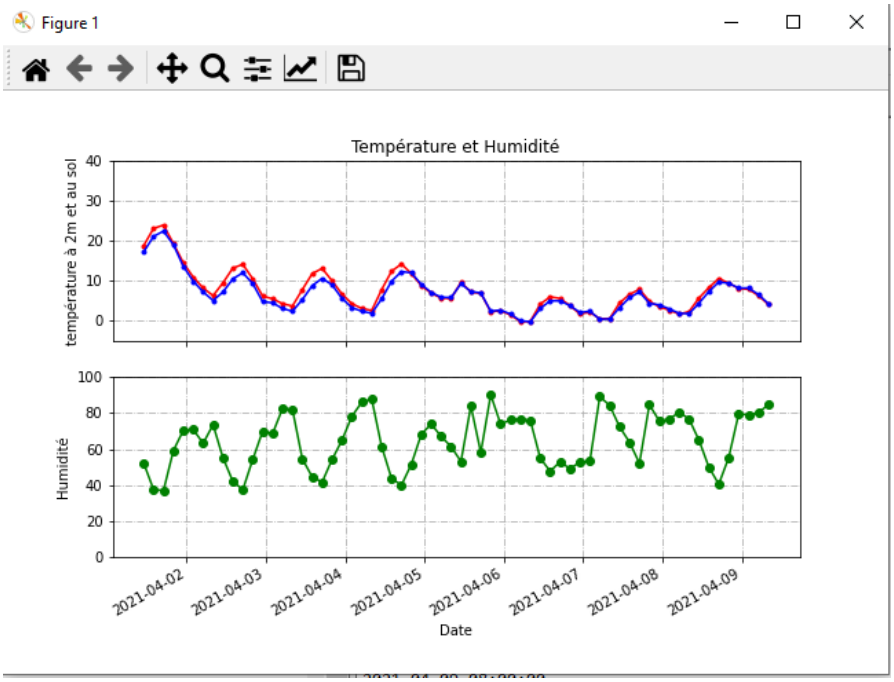
```
dt=datetime.strptime("2021-04-01 05:00:00", "%Y-%m-%d %H:%M:%S")
```

k-En s'appuyant sur la Compréhension des listes calculez les listes suivantes: lesDates, lesTempA2m, lesTempAuSol, lesHumidites
lesDates: liste contenant toutes les dates ayant servi de clé dans data (attention il s'agit des objets datetime.datetime et non pas des str.
lesTempA2m: liste contenant toutes les températures à 2m (en °C et non pas en kelvin)
lesTempAuSol: liste contenant toutes les températures au sol (en °C et non pas en kelvin)
lesHumiditesA2m: liste des humidités à 2m

l-Importez matplotlib.pyplot

```
import matplotlib.pyplot as plt
```

m-Tracez lesTempA2m, lesTempAuSol et lesHumiditesA2m en fonction de lesDates



Astuce: *Pour afficher le plot dans une fenêtre séparée sous spyder :*
Tapez ***%matplotlib auto*** *dans l'interpréteur python, pour rechanger taper* ***%matplotlib inline***

Exercice 2 : Décorateur

Un décorateur est une fonction qui modifie le comportement des autres fonctions, en particulier en ajoutant des instructions avant et/ou après la fonction et même conditionner l'appel de la fonction ou le limiter.

```
def mon_decorateur(fonction):
    def inner(*param, **param2):
        print("Action avant .............. ")
        fonction(*param, **param2)
        print("Action après ..............")
    return inner

#pour appliquer le décorateur à une fonction, il suffit d'ajouter
# @mon_decorateur juste avant la méthode
@mon_decorateur
def Affichage(v):
    print("Execution des instructions", v)
```

Le décorateur peut prendre en compte les paramètres de la fonction initiale en "forwardant" les paramètres de inner à la fonction décorée.
Créez une liste l=[1,2,3,4] et appelez la méthode Affichage(l) et observez le résultat.

Créez un décorateur, qui vous permet de calculer le temps d'exécution pour toute méthode décorée avec.
Pour mesurer le temps d'exécution d'une portion de code, il suffit simplement d'utiliser la fonction time.time() qui renvoie le temps CPU en secondes. La différence entre 2 de ces différents temps donnera le temps d'exécution de la portion de code encadrée.

Pour ceux qui aiment aller plus loin: *Python inclut un profileur appelé cProfile . Il donne non seulement le temps d'exécution total, mais également le temps de chaque fonction séparément, et vous indique combien de fois chaque fonction a été appelée, ce qui permet de déterminer facilement où vous devez effectuer des optimisations.*

```
import cProfile
cProfile.run('foo()')
```

Ci-dessous un décorateur utilisant cprofile pour "profiler" toute fonction décorée avec.

```python
import cProfile, pstats, io

def profile(fnc):
    """A decorator that uses cProfile to profile a function"""
    def inner(*args, **kwargs):
        pr = cProfile.Profile()
        pr.enable()
        retval = fnc(*args, **kwargs)
        pr.disable()
        s = io.StringIO()
        sortby = 'cumulative'
        ps = pstats.Stats(pr, stream=s).sort_stats(sortby)
        ps.print_stats()
        print(s.getvalue())
        return retval
    return inner

@profile
def TriFusion(myList):
```

Exercice 3 : GUI : tkinter

Commencez par tester et bien analyser ce code (en particulier la documentation de pack et ces options)

```python
import tkinter
#n'oubliez pas d'explorer tkinter vite fait
#dir(tkinter)
from tkinter import *
fenetre=Tk()
#créer un widget Label
monlabel=Label(fenetre, text="premier code tkinter")
#empaqueter le widget dans la fenetre
monlabel.pack()                    # defaults to side = "top"
#monlabel.pack(side="left")
#monlabel.pack(expand=1)
# lancer la fenetre
fenetre.mainloop()
```

Tester ensuite ce code (en particulier la création d'une entrée text, le bouton qui permet de modifier un label et la disposition des widget via grid

```
from tkinter import *
fenetre=Tk()
fenetre.title('deuxieme fenetre')
#fenetre.iconbitmap('iconedeprog.ico')
fenetre.geometry("450x150")
#créer un widget Label
monlabel1=Label(fenetre, text="label1")
monlabel2=Label(fenetre, text="label2")
monlabel3=Label(fenetre, text="label3")
monlabel4=Label(fenetre, text="label4")

#empaqueter le widget dans la fenetre ave grid
monlabel1.grid(row=1, column=0)
monlabel2.grid(row=1, column=1)
monlabel3.grid(row=2, column=2)
monlabel4.grid(row=3, column=0,columnspan=3)

entreeNom=Entry(fenetre, width=50, bg="grey", fg="blue")
def myClick():
    monlabel4.config(text="Hello " + entreeNom.get(), fg="red")


entreeNom.grid(row=0,column=0, columnspan=3)
#Ajouter un boutton pour quiter
monbouton=Button(fenetre,text="changer", padx=10, pady=10, command=myClick).grid(row=4, column=4)
```

Exercice 4 : GUI : tkinter : Calculatrice simple

Completer le code du fichier TD6_calculatrice2complete.py disponible sur DVO pour finaliser une calculatrice simple.