LOGICIEL MÉTÉO 🌤️

NATHAN BOSY | FRÉDÉRIC MARQUET

Terminale NSI

PRÉSENTATION

1. Description du projet

Le projet consiste à afficher les données météorologiques des villes souhaitées tant que l'utilisateur ne quitte pas le programme. Le logiciel fonctionne en partie grâce à une *API* qui renvoie toutes les données météorologiques de l'ensemble des villes de la planète. Cette API est reliée à de nombreux satellites tout autour du monde qui permettent de récupérer ces données. Nous avons ici utilisé *OpenWeather*, qui donne gratuitement les données en temps réel et les prévisions météorologiques. Grâce à cela, nous avons récupéré plusieurs données telles que : les températures (actuelle, minimale et maximale), les précipitations (humidité, pression), le temps (nuages, visibilité), le vent (moyenne, rafale, l'orientation) et le soleil (lever, coucher) et bien plus encore.

2. Objectif du projet

Notre objectif premier est de faire un logiciel **pratique** et **fiable**. Nous avons porté notre réflexion afin de faire un logiciel qui donne le plus d'informations possible sans que cela soit illisible, et sans faire de conversions à l'arrivée : le programme final doit être directement utilisable.

Nos premiers objectifs étaient de réussir à importer les données grâce à l'API, puis de récupérer seulement celles qui nous intéressent. Ensuite nous avons traité toutes nos données, et les avons converties dans les unités souhaitées.

L'objectif que nous n'avons pas accompli est l'interface graphique. En effet, nous souhaitions mettre une interface au programme à l'aide du module *Tkinter*. Cependant, le code du logiciel doit être complètement restructuré, ce qui prend du temps et qui laisserait inéluctablement laisser passer des failles provoquant des erreurs. Or notre but premier était d'abord d'avoir un logiciel fiable. Nous avons donc fait un compromis et **préféré la fiabilité** plutôt que la beauté.

3. Problèmes techniques et résolution

Tout au long du codage, nous avons rencontré des problèmes techniques que nous avons dû résoudre. Il y a deux sortes de problèmes : premièrement, la **gestion des erreurs**. Ensuite, nous avons des problèmes liés aux **fonctionnalités** et aux **conversions**, rendant notre logiciel plus facile à comprendre notamment. Commençons par la gestion des erreurs.

a. La gestion des erreurs

Notre plus gros souci était que le code ne marchait pas sans accès à internet. En effet, il utilise un API uniquement accessible depuis une adresse URL, mais récupère également un fichier CSV pour ensuite être traité. C'est simple : si l'utilisateur n'a pas accès à internet, le logiciel ne peut pas remplir ses critères. Nous avons donc décidé de rajouter une fonction qui empêche le logiciel de fonctionner sans accès à internet :

```
def test_connexion() :


    temp, essais = 0, 0


    while temp == 0 and essais < 3 :
        try :
            requests.get("https://google.com", timeout=5)
            temp = 1



        except ConnectionError :
            print('\n\nProblème réseau.\nTentative de reconnexion en
cours...')
            sleep(10)
            essais += 1


     assert essais != 3, ('\nNous n\'avons pas pu se connecter à
internet.\nVérifiez votre connexion et réessayez.')
```

Le reste de nos soucis venaient de l'API : et si la ville n'existait pas ? et si l'API ne fonctionne plus, que le site est inaccessible, ou alors que la station météorologique affiche une erreur ? Nous avons donc trouvé la solution : à chaque essai valide, une valeur nous affiche un code "200". Ceci veut dire que l'ensemble de ses potentiels problèmes ne se sont pas déroulés, et que les informations sont correctement parvenues. Nous avons donc rajouté quelques lignes permettant de vérifier si les données sont bien parvenues sans souci vers notre programme :

```
                                                    url     =
'https://api.openweathermap.org/data/2.5/weather?appid=25bb72e551083
279e1ba6b21ad77cc88&lang=fr&q=' + ville
            data =  requests.get(url).json()



                                    if   data['cod']   ==   200   :
# code signifiant que la ville existe
                temp = 1


               else : print('\nCette ville n\'existe pas ! Veuillez
réessayer.\n\n')
```

Le dernier problème venait des données exploitées. En effet, si nous prenions une ville sans pluie ou sans neige, le programme essayait toujours de récupérer les données et nous affichait donc un message d'erreur. De plus, certaines stations météo ne donnent pas les rafales de vent. Nous avons décidé de rajouter directement du code avant l'affichage pour vérifier si la donnée est présente ou non pour l'afficher ou ne pas l'afficher :

```
    if ('rain' in data) :
        pluie = data['rain']['1h']
        print(f'  • Pluie :          {pluie}mm/h')
```

```
    if ('snow' in data) :
        neige = data['snow']['1h']
        print(f'  • Neige :          {neige}mm/h')
```

```
if ('gust' in data['wind']) :
        rafales  = round(data['wind']['gust'] * 3.6, 1)
        print(f'  • Rafales :        {rafales}km/h')
```

b. Fonctionnalités et conversions

Premièrement, nous avons rajouté une variable qui regarde le nombre de recherches qui sont faites *(voir programme complet)*. Ainsi, lors de la première recherche, on affiche un message de bienvenue, et lors des recherches futures, nous rajoutons “nouveau” dans la recherche pour indiquer que ce n’est pas la première recherche que l’utilisateur fait.

Ensuite, une partie de notre travail s’est concentrée sur des **conversions d’unités**. En effet, nous avons dû convertir des mètres en kilomètres (division par 1000), des secondes en heures (division de 3600 car une heure = 3600 secondes).

De plus, les températures nous sont données en kelvin. Or nous voulons les afficher en degrés celsius. Ainsi nous avons rajouté 273,15 à chaque données (température du 0 absolu).

Enfin la pression est donnée en hPa. Pour faciliter la compréhension, nous l’avons converti en ATM (1 ATM = l’atmosphère terrestre moyenne au niveau de la mer). Ainsi, nous avons divisé les données par 1013,25 (= 1 ATM).

Nous avons également ajouté un système d'emoji : en fonction du temps, un emoji différent est affiché. Pour se faire, nous avons pris un code puis utilisé une fonction afin de le transformer en texte contenant un emoji. Le numéro correspond au temps actuel, "d" pour jour et "n" pour nuit :

```
def code_emoji(code) :
    if   code == '01d' :
        return "☀️"
    elif code == '01n' :
        return "🌚"
    elif code == '02d' or code == '02n' :
        return "🌤️"
     elif code == '03d' or code == '04d' or  code == '03n' or  code
== '04n':
        return "☁️"
    elif code == '09d' or code == '09n' :
        return "🌧️"
    elif code == '10d' or code == '10n' :
        return "🌦️"
    elif code == '11d' or code == '11n' :
        return "⛈️"
    elif code == '13d' or code == '13n' :
        return "🌨️"
    elif code == '50d' or code == '50n' :
        return "🌫️"
```

L'un des problèmes techniques que nous avons rencontrés, est de déterminer l'heure actuelle du lever et coucher du soleil. En effet, l'API nous renvoie l'heure mais en *Unix*. une mesure du temps basée sur le nombre de secondes écoulées depuis le 1er janvier 1970. Afin de convertir cela en heure, nous avons importé le module *datetime* qui convertit l'unix en heure. Nous avons également converti les heures en fonction du pays (*UTC*)

```
temps                          =  datetime.utcfromtimestamp(data['dt']  +
data['timezone']).strftime('%Hh%M')

UTC                                    =   round(data['timezone']/3600)
# diviser par le nombre de sec dans une heure


    if UTC >= 0 :

                                       UTC_texte      =      '+'+str(UTC)
# rajouter "+" si l'UTC est positif
    else :

        UTC_texte = str(UTC)
```

```
lever_soleil                                                          =
datetime.utcfromtimestamp(data['sys']['sunrise']                      +
data['timezone']).strftime('%Hh%M')

                          coucher_soleil                              =
datetime.utcfromtimestamp(data['sys']['sunset']                       +
data['timezone']).strftime('%Hh%M')
```

Un autre problème était de convertir les degrés de rotation en informations "Est, Ouest, Nord, Sud, etc..." Pour ce faire, nous avons divisé l'angle par 8 et récupérer le reste d'une division euclidienne et pouvoir récupérer l'orientation :

```
def direction(degré) :


  orientation = ['Nord',
                 'Nord-Est',
                   'Est',
                 'Sud-Est',
                   'Sud',
                'Sud-Ouest',
                  'Ouest',
                'Nord-Ouest']


  x = round(degré / (360 / len(orientation)))     # divise l'angle
                                  par 8


   return orientation[x % len(orientation)]       # retourne la
bonne orientation en prenant le reste une division euclidienne
                          (entre 0 et 7)
```

Enfin, notre dernier problème était que les pays étaient affichés en code ISO-3611 de type Alpha 2 (exemple : FR pour la France, DE pour l'Allemagne). Pour résoudre ce problème, nous récupérons au début du code un fichier CSV contenant toutes les informations ISO-3611 et les types d'affichage des pays :

```
test_connexion()
# vérification d'accès à internet

data_pays                                                    =
p.read_csv('https://www.data.gouv.fr/fr/datasets/r/4cafbbf6-9f90-418
4-b7e3-d23d6509e77b') # récupère le fichier csv data.gouv.fr
```

Et ensuite, nous utilisons une fonction qui récupère la bonne ligne où se trouve le code Alpha 2 et récupérons le nom entier du pays en français bien sûr !

```
def nom_pays(code, data) :


      ligne  =  data[data[" Code  alpha2"]  ==  code]        #  retient
seulement la ligne du pays ("FR")


    return ligne.values[0][3]                          # retourne le nom
associé au code ("France")
```

CODE EN ENTIER (AVEC COMMENTAIRES)

```
'''
                    LOGICIEL MÉTÉO
affiche les données météorologiques de la ville souhaitée.

                       créé par
                   Frédéric MARQUET
                          &
                    Nathan BOSY


                       v0.4.0
'''


import requests
```

```
from requests.exceptions import ConnectionError


from datetime import datetime

from time import sleep


import pandas as p









'''

Fonction qui permet de vérifier si on est connecté à internet.

'''


def test_connexion() :


    temp, essais = 0, 0


    while temp == 0 and essais < 3 :

        try :

            requests.get("https://google.com", timeout=5)

            temp = 1



        except ConnectionError :

            print('\n\nProblème réseau.\nTentative de reconnexion en cours...')

            sleep(10)

            essais += 1
```

```
        assert essais != 3, ('\nNous n\'avons pas pu se connecter à internet.\nVérifiez
votre connexion et réessayez.')









nbr_recherches = 0


test_connexion()
# vérification d'accès à internet

data_pays                                                                    =
p.read_csv('https://www.data.gouv.fr/fr/datasets/r/4cafbbf6-9f90-4184-b7e3-d23d6509e77b'
) # récupère le fichier csv data.gouv.fr







def data(nbr_recherches) :



    '''
    Programme qui demande et vérifie si la ville qui est entrée existe,
    puis après on récupère toutes ses données dans un variable.
    '''


    nouvelle_recherche = ''


    if nbr_recherches == 0 :
        print('\n                    LOGICIEL MÉTÉO',
                '\nAffiche les données météorologiques de la ville souhaitée.'        #
message de bienvenue (:
```

```
            '\n              (entrer \"q\" pour quitter)')

    temp = 0
    while temp == 0 :

            if nbr_recherches >= 1 :
                  nouvelle_recherche = 'nouvelle '                                  #
pour changer le texte en recherchant

              ville = input(f'\nVeuillez entrer le nom de la {nouvelle_recherche}ville :
')

               assert ville != 'q', ('\nMerci d\'avoir utilisé nos services !')      #
pour quitter le programme



              #ville = 'Béziers'                                                    #
test plus rapide
              #ville = ville.replace(' ','')                                       ->
pas possible (contre-exemple : New York)



              test_connexion()                                                      #
vérification d'accès à internet

                                                                          url       =
'https://api.openweathermap.org/data/2.5/weather?appid=25bb72e551083279e1ba6b21ad77cc88&
lang=fr&q=' + ville
            data =  requests.get(url).json()



              if data['cod'] == 200 :                                               #
code signifiant que la ville existe
                temp = 1
```

```
            else : print('\nCette ville n\'existe pas ! Veuillez réessayer.\n\n')



    #print(data)                                                                    #
test pour avoir toutes les données









    '''

    Récupération de toutes les données puis conversion avec les bonnes unités.

    '''



    pays                = nom_pays(data['sys']['country'], data_pays)              #
exemple : convertion "FR" en "France"

    description         = data['weather'][0]['description']

    emoji               = data['weather'][0]['icon']

    temperature         = round(data['main']['temp'] - 273.15, 1)                  #
convertion kelvin en degrés celsus

         temps                              =  datetime.utcfromtimestamp(data['dt']  +
data['timezone']).strftime('%Hh%M')









    UTC                 = round(data['timezone']/3600)                             #
diviser par le nombre de sec dans une heure



    if UTC >= 0 :

         UTC_texte = '+'+str(UTC)                                                  #
rajouter "+" si l'UTC est positif
```

```
    else :
        UTC_texte = str(UTC)



    temperature_min    = round(data['main']['temp_min'] - 273.15, 1)
    temperature_max    = round(data['main']['temp_max'] - 273.15, 1)
    ressenti           = round(data['main']['feels_like'] - 273.15, 1)

    humidite           = data['main']['humidity']
    pression             = round(data['main']['pressure']/1013.25, 3)                  #
convertion hP en ATM



    nuages             = data['clouds']['all']
    visibilite           = round(data['visibility']/1000, 1)                           #
convertion m en degrés km

    vent               = round(data['wind']['speed'] * 3.6, 1)
    orientation_vent    = direction(data['wind']['deg'])                               #
pour calculer la direction du vent

        lever_soleil              =  datetime.utcfromtimestamp(data['sys']['sunrise']  +
data['timezone']).strftime('%Hh%M')
        coucher_soleil          =  datetime.utcfromtimestamp(data['sys']['sunset']    +
data['timezone']).strftime('%Hh%M')



    '''
    Affichage dans la console des données extraites avec les convertions.
```

```
    '''


    print(f'\n\n\n\nDONNÉES DE LA VILLE DE {ville.upper()}, {pays.upper()}',

                    f'\n\n{description.capitalize()}     -   ',   code_emoji(emoji),   f'   -
{temperature}°C',
          f'\ndonnées de {temps} (UTC{UTC_texte})',


           '\n\n\nTEMPÉRATURES',
          f'\n   • Minimum :       {temperature_min}°C',
          f'\n   • Maximum :       {temperature_max}°C',
          f'\n   • Ressenti :      {ressenti}°C',



           '\n\nPRÉCIPITATIONS')




    if ('rain' in data) :
        pluie = data['rain']['1h']
        print(f'   • Pluie :         {pluie}mm/h')


    if ('snow' in data) :
        neige = data['snow']['1h']
        print(f'   • Neige :         {neige}mm/h')




    print(f'   • Humidité :      {humidite}%',
        f'\n   • Pression :      {pression} ATM'
```

```
          '\n\nTEMPS',
          f'\n  • Nuages :         {nuages}%',
          f'\n  • Visibilité :     {visibilite}km'



          '\n\nVENT',
          f'\n  • Moyenne :        {vent}km/h')



    if ('gust' in data['wind']) :
        rafales  = round(data['wind']['gust'] * 3.6, 1)
        print(f'  • Rafales :        {rafales}km/h')



    print(f'  • Orientation :    {orientation_vent}',



          '\n\nSOLEIL',
          f'\n  • Lever :          {lever_soleil}',
          f'\n  • Coucher :        {coucher_soleil}\n\n\n')



'''
Fonction qui convertit un code donné par un emoji.
```

```
'''

def code_emoji(code) :

    if   code == '01d' :
        return "🌞"

    elif code == '01n' :
        return "🌚"

    elif code == '02d' or code == '02n' :
        return "⛅"

    elif code == '03d' or code == '04d' or  code == '03n' or  code == '04n':
        return "☁️"

    elif code == '09d' or code == '09n' :
        return "🌧️"

    elif code == '10d' or code == '10n' :
        return "🌦️"

    elif code == '11d' or code == '11n' :
        return "⛈️"

    elif code == '13d' or code == '13n' :
        return "🌨️"

    elif code == '50d' or code == '50n' :
        return "🌫️"
```

```
'''
Fonction qui permet de convertir un angle donné en orientation.
'''

def direction(degré) :

    orientation = ['Nord',
                   'Nord-Est',
                   'Est',
                   'Sud-Est',
                   'Sud',
                   'Sud-Ouest',
                   'Ouest',
                   'Nord-Ouest']

    x = round(degré / (360 / len(orientation)))    # divise l'angle par 8

     return orientation[x % len(orientation)]        # retourne la bonne orientation en
prenant le reste une division euclidienne (entre 0 et 7)




'''
Fonction qui permet de convertir un code ISO-3611 de type Alpha 2 en nom
'''
```

```
def nom_pays(code, data) :

    ligne = data[data[" Code alpha2"] == code]      # retient seulement la ligne du pays
("FR")

    return ligne.values[0][3]                          # retourne le nom associé au code
("France")









'''
Boucle pour lancer le programme en boucle jusqu'à ce que l'utilisateur quitte.
'''

while True :
    data(nbr_recherches)
    nbr_recherches += 1









'''
Exemple de données qui arrvient après la demande :

    //////////

    {'coord': {'lon': 3.0833,
               'lat': 43.5},
```

```
'weather': [{'id': 804,
             'main': 'Clouds',
             'description': 'couvert',            FAIT
             'icon': '04d'}],                     FAIT

'base': 'stations',

'main': {'temp': 295.33,                          FAIT
         'feels_like': 295.12,                    FAIT
         'temp_min': 291.47,                      FAIT
         'temp_max': 296.61,                      FAIT
         'pressure': 1023,                        FAIT
         'humidity': 58,                          FAIT
         'sea_level': 1023,
         'grnd_level': 984},

'visibility': 10000,                              FAIT

'wind': {'speed': 2.54,                           FAIT
         'deg': 325,                              FAIT
         'gust': 3.71},                           FAIT

'clouds': {'all': 100},                           FAIT

'dt': 1664730576,                                 FAIT

'sys': {'type': 1,
        'id': 6519,
        'country': 'FR',                          FAIT
        'sunrise': 1664689553,                    FAIT
```

```
        'sunset': 1664731685},                    FAIT

 'timezone': 7200,                                FAIT

 'id': 3032832,

 'name': 'Béziers',                               FAIT

 'cod': 200}                                      FAIT

//////////

+ 'snow' et 'rain'                                FAIT

'''
```