Compte rendu du projet

Widget Nubibus

Villanueva Diego

Blase Julien

Cermolacce Arnaud

Terminale G5-NSI-Villars

Sommaire :

Introduction :

Le Projet Nubibus est un travail qui nous a été confié au début de cette année scolaire. L'objectif était de réaliser un programme fonctionnel qui met en pratique nos compétences en informatique. La consigne était de créer un programme, dans n'importe quel langage de programmation, nous avions initialement eu l'idée de concevoir un petit jeu vidéo d'énigme. Néanmoins l'idée était hors de nos capacités, ce qui nous incita à nous rediriger vers une suggestion de notre professeur qui nous paraissait intéressante, un widget météo. Il s'agit d'une application qui a pour but de donner la météo. De plus, nous avons choisi de réaliser ce projet en python, un langage de programmation que nous avons abordé au cours de notre année de NSI.

Objectifs du projet :

Les objectifs du projet était simple, réaliser un widget météo, soit une fenêtre qui s'affiche sur l'ordinateur et depuis laquelle l'on peut obtenir la météo ou que l'on se situe dans le monde et qu'elle soit compréhensible dans plusieurs langues. Et nous avons choisi de réaliser ce widget en python.

Contrainte :

Les contraintes principales étaient bien sur le python. En effet, bien que nous ayons beaucoup travailler sur ce langage au lycée, il nous a tout de même fallu apprendre à utiliser certaines bibliothèques que nous n'avions pas abordées en classe, ce qui est tout à fait normal. De plus, notre manque d'expérience dans l'organisation de ce genre d'exercice a rendu la tâche un peu plus compliquée que ce que nous pensions.

Développement :

Pour commencer le projet nous avons d'abord dû décider d'un nom. Nous avons choisi "Nubibus". Ce nom provient initialement du pluriel du mot nuage en latin, ainsi "Nubibus" signifie "des nuages". Une fois le nom trouvé, nous nous sommes mis au travail.

Initialement, nous avions comme projet de réaliser l'interface et la fenêtre d'affiche en python et ensuite d'utiliser une base de données pour récupérer les données météorologiques, néanmoins cela n'a pas abouti. En effet nous n'avons pas trouvé de base de données libre ouverte à tous nous permettant d'obtenir ce que nous cherchions. Puis sur le conseil de notre professeur Monsieur Fort, qui nous a conseillé d'utiliser un API, nous avons réussi à trouver quelque chose de fonctionnel et efficace.

Pour réaliser ce projet, nous avons divisé les différentes tâches. Julien fut chargé de créer la fenêtre et l'interface du widget. Arnaud était chargé d'aider à la création de l'interface et de la fenêtre et devait faire en sorte de pouvoir traduire le widget dans plusieurs langues. Diego devait récupérer les données de l'api et permettre leur utilisation en python ainsi que leur affichage.

I) L'interface

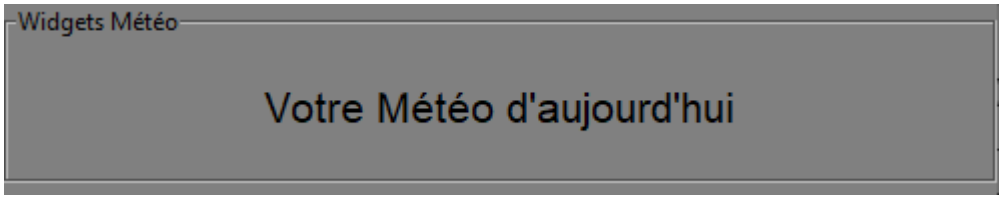
Pour réussir à obtenir le résultat souhaité, nous avons dû nous renseigner sur la bibliothèque Tkinter. Il s'agit d'une bibliothèque python qui permet de créer des fenêtres et de les modifier comme on le souhaite. Néanmoins la particularité de cette bibliothèque c'est qu'il existe une multitude de manière de l'écrire et cela peut donc créer des problèmes. Ainsi, nous avons choisi de n'en sélectionner qu'une seule et de réaliser le projet en limitant le plus possible les changements.

Ainsi, nous avons donc commencé par importer la bibliothèque tkinter, nous avons également dû importer deux autres bibliothèques en lien avec la bibliothèque tkinter, tkinter.font et tkinter.messagebox. Ces bibliothèques nous serviront à écrire dans les différents labels et canvas et à changer certains paramètres concernant la police d'écriture.

```
from tkinter import *
import tkinter.font as font
from tkinter.messagebox import *
```

Puis nous avons créé une fenêtre grâce à la commande **fenetre = Tk()** Ensuite, nous avons créé un « label frame » avec la commande LabelFrame. On assigna la lettre « l » au labelFrame puis nous avons créé un label, qui nous permet donc d'écrire dans le labelFrame, nous avons donc assigné la lettre o à ce label puis nous y avons l'entête de la fenêtre :

```
fenetre = Tk()
fenetre.title("Nubibus by DJA")
fenetre['bg']='grey'

l = LabelFrame(fenetre, text="Widgets Météo", padx=20, pady=20, bg='grey')
l.pack(fill="both", expand="yes")
o = Label(l, text="Votre Météo d'aujourd'hui", font="Arial 16", bg="grey")
o.pack()
```

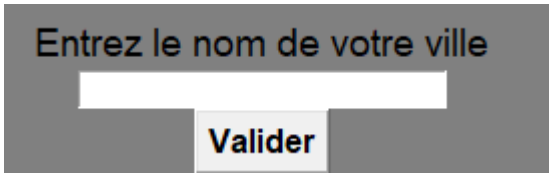


Ensuite nous avons créé un **Canvas** avec la commande **Canvas()**. Par défaut, un canvas est un encadré blanc dans lequel nous pouvons afficher/modifier des textes, des images, etc…. Ce dernier est relié à la fonction **recupere()** qui permet d'afficher les données météorologiques souhaitées.

Néanmoins pour faire fonctionner la fonction **recupere()** il faut une valeur, plus précisément un nom de ville et donc pour que l'utilisateur puisse choisir le lieux dont il souhaite obtenir les données. Nous avons donc créé un bouton avec la commande tkinter **Button()** qui nous permet d'envoyer une valeur dans la fonction **recupere()**. De plus, nous avons ajouté un Label juste au-dessus du bouton pour pouvoir inscrire la consigne.

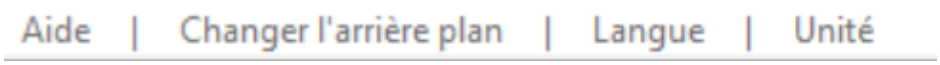
```
value = StringVar()
entree = Entry(fenetre, textvariable=value, width=30)
entree.pack()

q = font.Font(weight="bold")

bouton = Button(fenetre, text="Valider", command=recupere)
bouton['font'] = q
bouton.pack()

fenetre.mainloop()
```

Cela affiche donc:



Par la suite, nous avons ajouté **une menu barre** grâce à la commande **Menu()**. **La menu barre** est une barre qui se situe dans le haut de la fenêtre. Sur notre widget, cette barre et ses différent menu interactif nous permet de proposer des options de personnalisation du widget, incluant notamment le changement de langue, le changement de couleur, et un tutoriel qui vous explique comment faire fonctionner le widget. Pour générer les différentes fenêtres, nous avons donc généré plusieurs menus qui se regroupent dans la menubar. Ces menus sont nommés menu1, menu2 etc… et pour les faire fonctionner, on utilise la commande “menu.add_command” qui nous permet d'exécuter une fonction.

On obtient donc:



Cette **menu bar** utilise plusieur sous-fonction tel que def bleu() qui permet le changement de couleur du widget en bleu ou encore def alert() qui permet d’afficher le tutoriel d’utilisation

```
def bleu():
    """
        Permet de changer le theme du widget avec la couleur bleue
    """
    l.config(bg="blue", fg="white")
    k.config(bg="blue", fg="white")
    o.config(bg="blue", fg="white")
    fenetre['bg']='blue'
```

.

cette définition change le bg (soient la couleur) des label l et o et du canvas k grâce à la commande .config, il change également la couleur de la police du texte inscrits dans o/k/l en blanc pour une meilleur visibilité. On change également le paramètre bg de la fenêtre pour changer la couleur général du widget

II) L'API

Tout d'abord, il nous fallait trouver une base de données qu'on pourrait exploiter pour notre widget, mais n'étant que 3 débutants, on a cherché sans rien trouvé jusqu'à ce que notre professeur adorée nous guida vers l'API (**Interface de programmation**) qui facilite la création de logiciels d'application;
*L'**API** crée un langage universel pour faire communiquer le client et le serveur. On parle d'intégration ou de systèmes intégrés lorsque plusieurs systèmes sont reliés par une **API**. Ce type de système est interopérable et permet aux différentes applications d'échanger entre elles.*
donc pour faire court, **L'API** est une sorte de base de données facile et pratique à utiliser.

J'ai donc cherché un site météorologique sur lequel **L'API** serait gratuit car la plupart des sites qui offrent ce genre de données sont payants avec un abonnement. J'ai fini par trouvé un site "***OpenWeatherMap***" qui propose des services gratuits et payants pour les données météorologiques.

J'ai donc choisi un **API** qui donne la météo actuelle et qui contient:

- Accès aux données météorologiques actuelles plus de 200000 villes.
- Des données météorologiques provenant de différentes sources telles que des modèles météorologiques mondiaux et locaux, des satellites, des radars et un vaste réseau de stations météorologiques.
- Formats JSON, XML et HTML.
- Inclus dans les abonnements gratuits et payants.
- Avec plusieurs fonctionnalités qu'on pourrait ajouter nous mêmes par la suite.

Suivant le choix de **L'API** et la création d'un compte OpenWeatherMap on crée une **clé API** pour que ça soit notre chemin et c'est cette clé qui va nous permettre d'avoir accès aux données demandés.

Ensuite d'avoir trouvé notre bonheur on doit appeler les donnés du site vers notre widget.
Le site est juste parfait a ce niveau, il nous explique tout ce qu'on peut faire et comment on doit faire, même si on doit quand même réfléchir par nous mêmes.
On utilise donc cette commande pour appeler les donnés:

```
complete_api_link =
"https://api.openweathermap.org/data/2.5/weather?q="+localisation+"&app
id="+user_api+"&lang="+langage
```

Ici, On envoie une demande au site les données météorologiques de "*Localisation*" qui sera la ville que l'utilisateur va demander suivi de "*user_api*" qui sera notre clé pour avoir accès aux données et on a rajouté une option qui est la langue.

Donc le code pour accéder à la base de données est :

```
    user_api = "29b27df27f95328236439092ed43a1c3"
    localisation = f
    langage= "fr"

    complete_api_link =
"https://api.openweathermap.org/data/2.5/weather?q="+localisation+"&app
id="+user_api+"&lang="+langage
    api_link = requests.get(complete_api_link)
    api_data = api_link.json()
```

Enfin, grâce à ce code le site nous renvoie les données demandés et on doit maintenant les trier et les afficher dans notre interface Tkinter.

```
        temp_city = ((api_data['main']['temp']) - 273.15)
        weather_desc = api_data['weather'][0]['description']
        hmdt = api_data['main']['humidity']
        wind_spd = api_data['wind']['speed']
        date_time = datetime.now().strftime("%d %b %Y | %I:%M:%S %p")
        city_country= api_data['sys']['country']
```

C'est ici qu'on récupère les données et on commence à les trier.
par exemple:

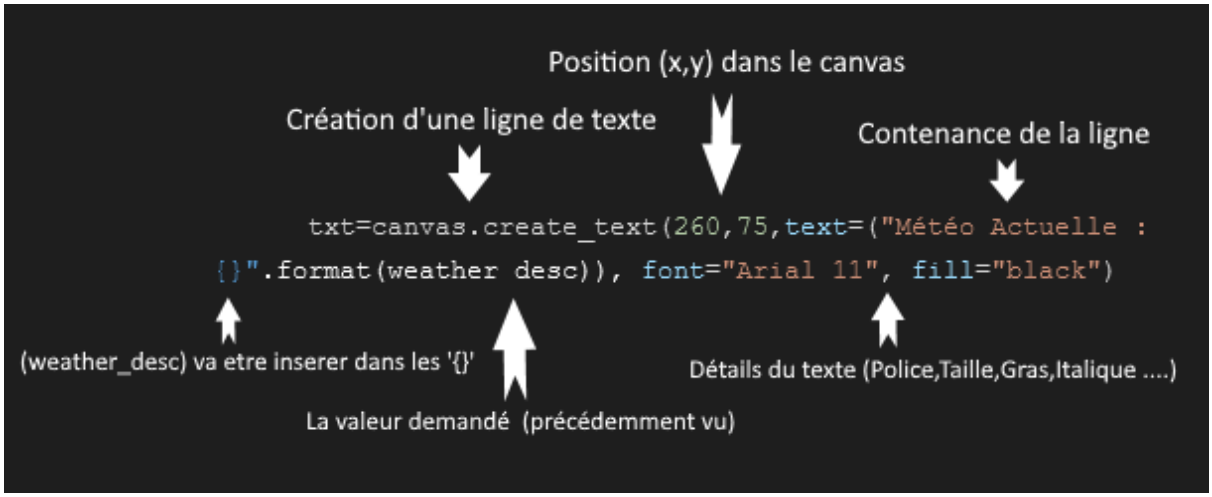
-temp_city est la température de la ville qui nous est donnée en Kelvin qu'on transforme en C°.
-weather_desc est la description de la météo.
-hmdt est le pourcentage d'humidité.
-wind_spd est la vitesse du vent en km/h.
-date_time est la date et l'horaire a laquel les données ont été demandé au widget.
-city_country est le pays originaire de la ville demandé.

Maintenant que les donnés reçu sont triés et simplifier on dois les afficher dans l'interface et pour ce faire on va utiliser ces commandes:

```
      txt=canvas.create_text(175,12, text=(city_country),
font=('Arial', 12,'bold'), fill="black")
```

```
      txt=canvas.create_text(175,30, text=("|  {}  | {}
|".format(localisation.upper(), date_time)), font="Arial 12",
fill="black")

      txt=canvas.create_text(179,54,text="{:.2f} C°".format(temp_city),
font=('Arial', 15,'bold'), fill="black")

      txt=canvas.create_text(260,75,text=("Météo Actuelle :
{}".format(weather desc)), font="Arial 11", fill="black")

      txt=canvas.create_text(90,75,text=("Humidité Acutelle :
{}%".format(hmdt)), font="Arial 11", fill="black")

      txt=canvas.create_text(170,90,text=("Vitesse du Vent actuelle :
{}km/h".format(wind_spd)), font="Arial 11", fill="black")
```

Enfin, on va maintenant afficher les données à l'utilisateur.
Par exemple:



Maintenant, on va s'attaquer aux bugs qu'on pourrait rencontrer :

-Si la ville n'existait pas.
-Si l'utilisateur n'a pas écrit une ville.

```
if api_data['cod'] == '404':

        txt=canvas.create_text(170,35,text="Ville ''{}''
Introuvable,".format(localisation), font=('Arial', 13,'bold'),
fill="#ff3552")
```

```
        txt=canvas.create_text(170,60,text="Vérifiez le Nom ou essayer
une autre Ville".format(localisation), font=('Arial', 10,'bold'),
fill="#ff3552")
```

*En informatique, l'erreur **404** est un code du protocole de communication HTTP sur le réseau internet pour signaler un incident. Ce code est renvoyé par un serveur HTTP pour indiquer qu'aucune ressource, généralement une page web, n'a été trouvée à l'adresse demandée.*

C'est donc de ce code d'erreur qu'on va pouvoir réparer le bug si une ville est introuvable; Si le code ne trouve pas la ville, il va envoyer le code 404 ce qui voudrait dire pour le code d'afficher la ligne de texte ci-dessus.

"Ville ''{}'' Introuvable,Vérifiez le Nom ou essayez une autre Ville".


```
if api_data['cod'] == '400':
            txt=canvas.create_text(170,35,text="Veuillez Insérez une
ville".format(localisation), font=('Arial', 15,'bold'), fill="#ff3552")

txt=canvas.create_text(170,70,text="S.V.P".format(localisation),
font=('Arial', 13,'bold'), fill="#ff3552")
```

*Le code de statut de réponse HTTP **400 Bad Request** indique que le serveur ne peut pas comprendre la requête en raison d'une syntaxe invalide. Le client ne devrait pas répéter la requête sans modification.*

Quand on n'insère pas de nom dans la case le nom d'erreur que reçoit le programme est 400 ce qui va le traduire par les lignes de textes ci-dessus.

***"Veuillez Insérez une ville
SVP ".***

Maintenant qu'on a le cœur de notre projet il faut que tout le reste du programme reçoivent ses données et pour ce faire, nous avons créé la fonction ***recupere()***. Cette fonction peut être considérée comme le cœur du widget, elle effectue trois actions :

La première, est de créer un pont entre l'application et le site en question.

La deuxième, est de savoir reconnaître 2 types d'erreurs courantes et de les corriger en informant l'utilisateur de sa faute.

La troisième, est de récupérer les données météo et pouvoir les afficher a l'utilisateur.

III) Le changement de langue et d'unité

On commence par intégrer une variable qui prendra différentes valeurs selon l'option choisi (on prendra ici l'exemple de l'unité de température)

```
mesureTemp=-1
```

Plus loin on associe a chaque valeur de la variable une fonction qui convertit en l'unité sélectionnée.

```
def change_mesure(var):
    global mesureTemp
    mesureTemp=var

def to_Fahrenheit(tempKelvin):
  return (tempKelvin-273.15)*(9/5)+32

def to_Celsius(tempKelvin):
  return tempKelvin - 273.15
```

Ici la température étant donnée en Kelvin on la convertit ou non.

Ensuite on crée le menu en question:

```
menu7 = Menu(menubar, tearoff=0, bg="white", fg="black")
menubar.add_cascade(label="Unité", menu=menu7)
menu7.add_command(label="Celcius (défaut)", command= lambda:change_mesure(-1))
menu7.add_command(label="Farenheit", command=lambda:change_mesure(1))
menu7.add_command(label="Kelvin", command=lambda:change_mesure(0))
```

ici lambda est ajouté car les fonction ne font pas partie du canvas contrairement à celles pour modifier sa couleur

```
  else:
      if(mesureTemp == -1) :
          temp_city = (to_Celsius(api_data['main']['temp']))
      else:
          if(mesureTemp == 0) :
              temp_city = (api_data['main']['temp'])
          else:
              temp_city = (to_Fahrenheit(api_data['main']['temp']))
      weather_desc = api_data['weather'][0]['description']
```

Enfin ici on associe les différentes valeurs de la variable mesureTemp a l'affichage de l'API.

Pour changer les langues on procède de façon similaire en créant une nouvelle variable :

```
langage=-1
```

```
def change_langue(var1):
    global langage
    langage=var1
```

On crée ensuite une fonction pour la définir

Puis on crée un nouveau menu bar dans lequel chaque langue modifie la valeur de la variable:

```
menu5 = Menu(menubar, tearoff=0, bg="white", fg="black")
menubar.add_cascade(label="Langue Des Infos", menu=menu5)
menu5.add_command(label="Français (défaut)", command=lambda:change_langue(-1))
menu5.add_command(label="Anglais", command=lambda:change_langue(0))
menu5.add_command(label="Russe", command=lambda:change_langue(1))
menu5.add_command(label="Chinois", command=lambda:change_langue(2))
menu5.add_command(label="Arabe", command=lambda:change_langue(3))
menu5.add_command(label="Italien", command=lambda:change_langue(4))
menu5.add_command(label="Espagnol", command=lambda:change_langue(5))
menu5.add_command(label="Allemand", command=lambda:change_langue(6))
```

Ensuite on change la clé en fonction de la valeur de la variable sélectionnée:

```
if langage==-1:
    complete_api_link = "https://api.openweathermap.org/data/2.5/weather?q="+localisation+"&appid="+user_api+"&lang=fr"
if langage==0:
    complete_api_link = "https://api.openweathermap.org/data/2.5/weather?q="+localisation+"&appid="+user_api+"&lang=en"
if langage==1:
    complete_api_link = "https://api.openweathermap.org/data/2.5/weather?q="+localisation+"&appid="+user_api+"&lang=ru"
if langage==2:
    complete_api_link = "https://api.openweathermap.org/data/2.5/weather?q="+localisation+"&appid="+user_api+"&lang=zh_cn"
if langage==3:
    complete_api_link = "https://api.openweathermap.org/data/2.5/weather?q="+localisation+"&appid="+user_api+"&lang=ar"
if langage==4:
    complete_api_link = "https://api.openweathermap.org/data/2.5/weather?q="+localisation+"&appid="+user_api+"&lang=it"
if langage==5:
    complete_api_link = "https://api.openweathermap.org/data/2.5/weather?q="+localisation+"&appid="+user_api+"&lang=es"
if langage==6:
    complete_api_link = "https://api.openweathermap.org/data/2.5/weather?q="+localisation+"&appid="+user_api+"&lang=de"

api_link = requests.get(complete_api_link)
```

Enfin on change la langue des informations écrites dans le widget(ici en russe):

```
if langage==1:
    txt=canvas.create_text(150,15, text=(" - {}  || {}".format(localisation.upper(), date_time)), font="Arial 12", fill="black")
    txt=canvas.create_text(140,30,text="Текущая температура: {:.2f} градусы°".format(temp_city), font="Arial 11", fill="black")
    txt=canvas.create_text(140,45,text=("Текущая погода :",weather_desc,), font="Arial 11", fill="black")
    txt=canvas.create_text(140,60,text=("Текущая влажность :",hmdt, '%'), font="Arial 11", fill="black")
    txt=canvas.create_text(140,75,text=("Текущая скорость ветра :",wind_spd ,'km/h'), font="Arial 11", fill="black")
    canvas.pack(side=TOP, padx=50, pady=50)
```

NB : Il est important de noter que TOUS les changements concernant la langue et/ou l'unité de mesure de température seront effectués seulement après avoir cliqué une nouvelle fois sur valider.

Conclusion :

Pour conclure, nous pouvons dire que nos objectifs ont été validés, en effet, le programme est fonctionnel, sans bug et nous fournit donc les données météorologiques souhaitées. De plus, nous avons pu également ajouter certaines options de personnalisation tel que le changement de couleur du widget qui n'était pas initialement prévu, et une petite référence à notre idée initale, le jeux vidéo, en incorporant un "easter-egg", comme cela se faisait et se fais encore dans certains jeux vidéo. Cet affichage secret contient uniquement les noms des développeurs, soient nos noms.
Ce projet nous aura aussi permis d'approfondir nos compétences dans le travail d'équipe et également d'apprendre de nouvelle bibliothèque python tel que tkinter.

Logiciels Utilisés:

Visual Studio Code

SPYDER

Site Utilisé:

OpenWeather

Nos plus sincères remerciements à notre prof d'NSI pour nous avoir aidé et supporté durant cette année et pour pouvoir lui proposer quelque chose qui nous fasse plaisir et qui nous a plu a créé.

Cordialement,

Vos élèves préférés de Dominique Villars.