

INF2 - Rapport du TP4

Consignes :

Travail général :

- Expliquer le problème, le raisonnement et la solution
- Libre sur la manière de faire (mais expliquer)

Exercice 1 :

- Cercles apparaissant comme des cercles
 - Pas de contraintes sur les échelles des axes
- Bonus : changer rayon des cercles

Exercice 2 :

- Avantages et inconvénients d'augmenter le nombre de points (discrétisation des données)

Exercice 3 :

- Capture d'écran pour les csv aux différentes étapes

Rendu :

Exercice 1 - Partie A

★ Explication du problème général :

Il faut tracer 9 cercles de centre (10, 10) avec un rayon allant de 1 à 9, avec un pas de 1 comme suit :

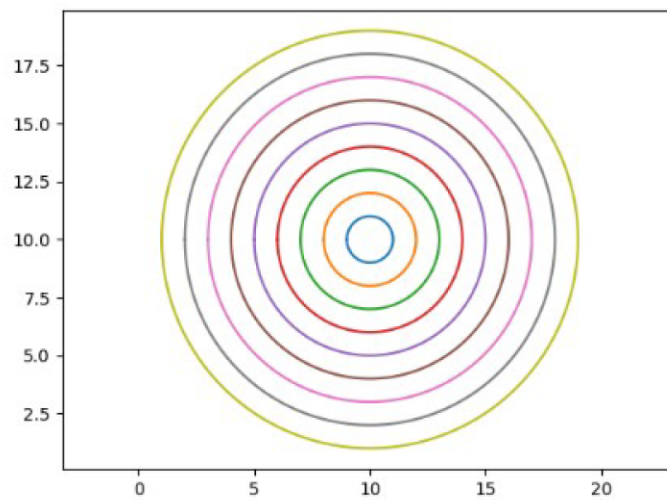


Figure 1 : Capture d'écran du graphique à réaliser

★ Raisonnement et analyse du problème :

1ère étape : Définir une fonction permettant de tracer un cercle de rayon variable

Tout d'abord, il nous semble important de réutiliser certaines formules mathématiques, notamment : $x = \text{rayon} * \cos(x)$ et $y = \text{rayon} * \sin(x)$ (x allant de 0 à 2π) pour réaliser le cercle.

Cependant, pour pouvoir générer un cercle, il faut avoir différentes valeurs de x comprises entre 0 et 2π auxquelles on appliquera les formules mathématiques (problème 1).

Résolution du problème 1 : Après recherche, la librairie Numpy pourrait nous aider. En effet, cette librairie est dotée d'une fonction `linspace`, qui permet de générer n valeurs régulièrement espacées dans l'intervalle $[0, 2\pi]$. Elle s'utilise comme suit :

```
import numpy as np
liste_valeurs_espacees = np.linspace(borne_min, borne_max,
nb_valeurs)
```

2ème étape : Concevoir une boucle pour générer les différents cercles

Afin de générer les différents cercles, l'idée que nous avons eu est d'utiliser une boucle "for" afin de faire varier le rayon de 1 à 9.

3ème étape : Affichage du graphique

Il nous semble utile de regarder la documentation de la librairie Matplotlib afin d'afficher le graphique (problème 2).

Résolution du problème 2 : Après recherche dans la documentation de la librairie Matplotlib (et plus particulièrement matplotlib.pyplot), il s'avère que pour ajouter des points sur un graphique, il faut dans un premier temps utiliser la fonction `plot(valeurs_x, valeurs_y)` pour tracer y en fonction de x. Puis, il faut afficher le graphique à l'écran, en procédant comme suit :

```
#Affichage du graphique
plt.show()
#Fermeture du graphique
plt.close()
```

4ème étape : Avoir des cercles qui ressemblent à des cercles

Après avoir généré notre premier test du programme, les cercles réalisés par la fonction n'apparaissent pas comme des cercles, mais comme des ellipses (problème 3).

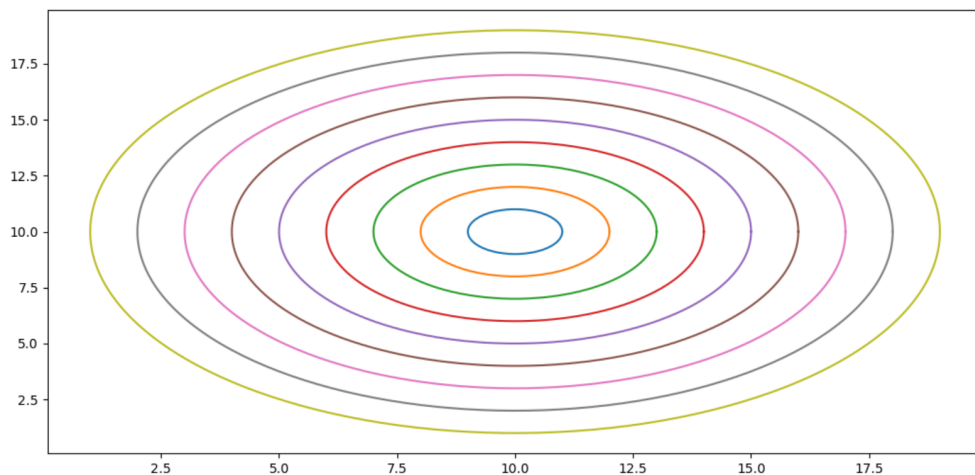


Figure 2 : Capture d'écran modélisant le problème 3

Résolution du problème 3 : Après recherche dans la documentation de la librairie Matplotlib, nous avons trouvé que pour obtenir de "vrais cercles", il faut utiliser les fonctions `subplots` et `set_aspect`, permettant de générer les axes et de leur donner la même échelle.

```
#Création de la figure et des axes
figure, axes = plt.subplots()
#Mise à la même échelle des axes
axes.set_aspect(1)
```

5ème étape (optionnel) : Echelle des axes

Egalement, après aperçu du rendu précédent, un autre problème se pose : l'échelle des axes.

Regarder la documentation de la librairie Matplotlib pourrait nous aider à résoudre ce problème (problème 4).

Résolution du problème 4 : On peut utiliser les fonctions `xticks` et `yticks` du module `matplotlib.pyplot` afin de définir les valeurs minimales et maximales pour chacun des axes.

```
#Définition des valeurs minimales et maximales des axes x et y
plt.xticks([tableau_avec_les_graduations_à_mettre])
plt.yticks([tableau_avec_les_graduations_à_mettre])

#On peut utiliser np.arange(valeur_min, valeur_max, pas) pour
générer le tableau avec les graduations
```

★ Solution apportée :

Après avoir trouvé une solution algorithmique et résolu l'ensemble des problèmes observés, nous avons pu réaliser le programme pas à pas, en commençant par la fonction `cercle` jusqu'à la gestion des graduations des axes. La solution apportée est écrite dans le fichier `Exercice 1 - Matplotlib (partie A).py`.

Exercice 1 - Partie B

★ Explication du problème général :

Il faut tracer un triangle de cercles en respectant les règles suivantes :

- sur la 1ère ligne, 1 cercle de rayon 1 centré
- sur la 2^e ligne, 2 cercles de rayon 1 centrés
- ...
- sur la 10^e ligne, 10 cercles de rayon 1 centrés

La figure à représenter est la suivante :

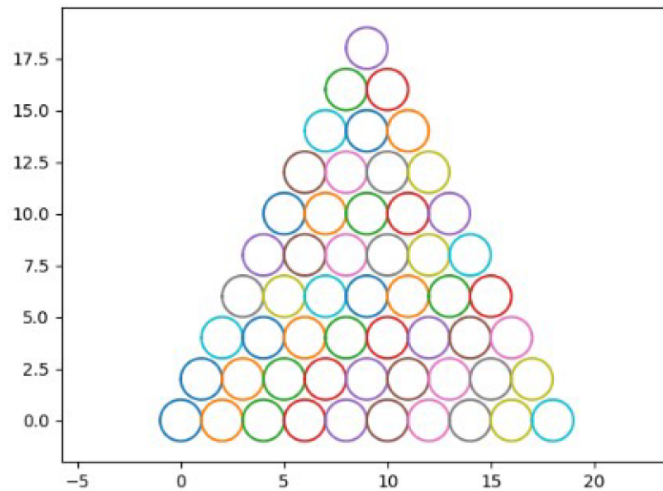


Figure 3 : Capture d'écran du graphique à réaliser

★ Raisonnement et analyse du problème :

1ère étape : Définir les éléments de la partie A que l'on peut réutiliser

Il va falloir tracer un triangle de cercles, on va donc pouvoir réutiliser la fonction `cercle` en appliquant au cas où rayon = 1.

On va pouvoir réutiliser les parties du code permettant d'obtenir de "vrais cercles" et ceux concernant la gestion des graduations sur les axes.

2ème étape : Définir une boucle pour générer l'ensemble des cercles

Après réflexion, nous nous sommes aperçus qu'il faut 2 boucles "for" pour générer ce triangle de cercles. En effet, une première boucle nous permet de parcourir les 9 lignes du graphique tandis qu'une autre imbriquée permet de générer i cercles sur la ligne i . Ainsi, nous avons obtenu le nombre de cercles souhaités pour chaque ligne.

3ème étape : Centrer chaque ligne de cercles

Cependant, un problème se pose : il faut centrer chaque ligne de cercles afin d'obtenir le rendu. Pour cela, on change les centres des cercles lors de l'appel de la fonction `cercle`.

Plus généralement, on remarque que :

- sur la ligne 0 (la base du graphique), il y a 9 cercles en tout, on note que le premier cercle a pour centre (0, 0) et le cercle suivant sur la même ligne est espacé de 2 fois le rayon du cercle (c'est-à-dire de 2 unités selon l'axe des abscisses)
- sur la ligne i , située $2*i$ graduations au-dessus de la ligne 0, il y a $9-i$ cercles, le premier cercle de cette ligne a pour centre (i , $2*i$) et le cercle suivant sur la même ligne est espacé de 2 fois le rayon du cercle (c'est-à-dire de 2 unités selon l'axe des abscisses)

On en déduit le code suivant pour générer les lignes de cercles :

```
#Boucle pour générer les 10 lignes de cercles de rayon 1
for i in range(10):
    #Boucle pour créer les 10-i cercles de rayon 1 par ligne
    for j in range(10-i):
        cercle(i + 2 * j, 2 * i, 1)
```

★ Solution apportée :

Après avoir réutilisé les éléments utiles de la partie A, nous avons généré à l'aide de 2 boucles imbriquées le triangle de cercles demandé. La solution apportée est écrite dans le fichier Exercice 1 - Matplotlib (partie B).py.

Exercice 2

★ Explication du problème général :

On doit écrire un programme qui stocke et affiche les valeurs de x et cos(x) allant de -5 à 5 avec une incrémentation de 1.

★ Raisonnement et analyse du problème :

Partie A : Ecrire la fonction écrire() qui permet de stocker les valeurs de x et cos(x) allant de -5 à 5 avec une incrémentation de 1 dans un fichier math.csv

1ère étape : Générer les valeurs de x et cos(x) demandées

Pour générer les valeurs entre -5 et 5, la solution nous ayant semblé la plus efficace est de créer un tableau comme suit :

```
abscisse = [x for x in range(-5, 6)]
```

Pour générer un tableau avec les valeurs de cos(x), nous avons utilisé la fonction cos() du module numpy comme suit :

```
ordonnee = [numpy.cos(x) for x in abscisse]
```

2ème étape : Créer un dataframe avec les abscisses et ordonnées

Afin de pouvoir écrire dans le fichier math.csv, nous avons choisi d'utiliser la librairie pandas afin de générer un dataframe associé aux données comme vu en cours magistraux. Nous souhaitons écrire dans une première colonne les valeurs de x et dans une deuxième colonne les valeurs de cos(x).

Pour cela nous instancions un objet de la classe DataFrame (issu de Numpy) comme suit :

```
donnees = pd.DataFrame({'x': abscisse, 'cosinus': ordonnee},
                        columns=['x', 'cosinus'])
```

3ème étape : Ecriture des données dans le fichier

Afin d'écrire les données dans le fichier math.csv, il faut exporter les données au format csv grâce à la fonction to_csv de la classe DataFrame du module pandas comme suit :

```
donnees.to_csv(math.csv, index=None, header=True,
               encoding='utf-8', sep=';')
```

Nous avons choisi d'utiliser comme séparateur de données le point virgule, l'encodage en utf-8 et d'afficher le nom des colonnes pour améliorer la lisibilité du fichier.

★ Solution apportée :

Après avoir déterminé une solution algorithmique, nous avons écrit la fonction écrire() dans le fichier Exercice 2 - Lecture et écriture.py, qui nous a donné le résultat suivant dans le fichier math.csv :

	x	sinus
1	-5	0.28366218546322625
2	-4	-0.6536436208636119
3	-3	-0.9899924966004454
4	-2	-0.4161468365471424
5	-1	0.5403023058681398
6	0	1.0
7	1	0.5403023058681398
8	2	-0.4161468365471424
9	3	-0.9899924966004454
10	4	-0.6536436208636119
11	5	0.28366218546322625

Figure 4 : Fichier math.csv après écriture

Partie B : Ecrire la fonction lire() qui permet de lire le fichier math.csv et d'afficher la courbe associée grâce à la librairie matplotlib.

1ère étape : Extraire les données du fichier pour les exploiter avec Python

Afin d'extraire les données du fichier pour les rendre exploitables par Python, on utilise la fonction read_csv de la classe DataFrame du module comme suit :

```
donnees = pd.read_csv(fichier, usecols=['x', 'cosinus'], sep=';')
```

On utilise uniquement les colonnes x et cosinus du fichier math.csv écrit précédemment et on définit comme séparateur de données le point virgule (car utilisé lors de l'écriture du fichier math.csv). Ces données pourront ensuite être affichées.

Si on affiche les données dans la console, on obtient le résultat suivant :

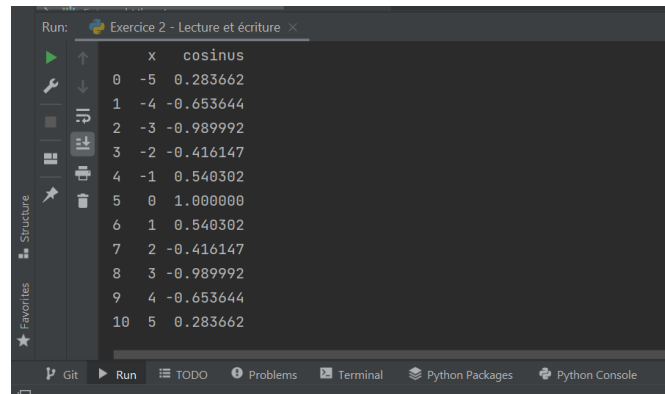


Figure 5 : Affichage des données extraites du fichier math.csv dans la console

2ème étape : Séparer les données correspondant aux abscisses x et aux ordonnées cos(x)

Afin d'extraire les données correspondant à une colonne précise du fichier, on écrit `donnees.nom_de_la_colonne`.

Ainsi, appliqué à notre fichier math.csv et ses colonnes, cela fonctionne comme suit :

```
#Extraction du fichier des valeurs de x
abscisse = donnees.x
#Extraction du fichier des valeurs de cosinus
ordonnee = donnees.cosinus
```

3ème étape : Création de la courbe et affichage du graphique

On ajoute les points sur le graphique grâce à la fonction `plot(valeurs_x, valeurs_y)` comme expliqué à la question 1, puis on utilise les fonctions `show()` et `close()` du module `matplotlib.pyplot` pour afficher à l'écran le résultat. A nouveau, on redéfinit l'échelle des axes afin d'obtenir un rendu visuel agréable pour l'utilisateur.

★ Solution apportée :

Ainsi, nous avons écrit la fonction `lire()` dans le fichier Exercice 2 - Lire et écrire.py. Nous avons le résultat suivant :

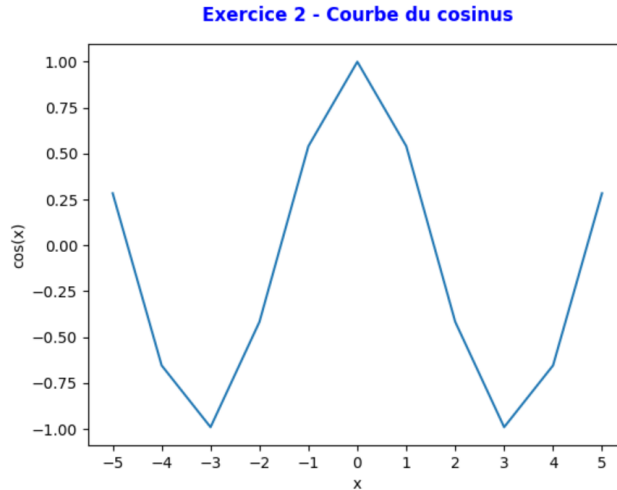


Figure 6 : Tracé de la fonction cosinus à partir des données du fichier math.csv

Exercice 3

★ Explication du problème général :

On souhaite créer un système de gestion relatif à un concours. Ce système doit permettre à l'utilisateur d'ajouter des candidats et renvoie un diagramme en camembert récapitulant les pourcentages de candidats admis, recalés et ajournés.

★ Raisonnement et analyse du problème :

1ère étape : Définir la fonction permettant d'ajouter des candidats dans le fichier candidats.txt

Tout d'abord, il faut ouvrir le fichier candidats.txt en mode "ajout", afin de laisser la possibilité à l'utilisateur d'ajouter l'ensemble des candidats en plusieurs fois s'il le souhaite.

Pour cela, on procède comme suit :

```
#Ouverture du fichier candidats.txt en ajout([a]ppend)
with open("candidats.txt", "a") as candidats:
```

Ensuite, il faut créer une boucle pour que l'utilisateur puisse ajouter autant de candidats que possible. Ne connaissant pas le nombre d'itérations, nous ferons une boucle "while".

Puis, on demande à l'utilisateur de saisir les informations du candidat à ajouter (NCIN, nom, prénom, âge et décision). On teste les valeurs rentrées (ex : âge > 0, décision correcte) puis si les données sont correctes, il faut écrire une ligne relative au candidat dans le fichier concours.txt.

Pour cela, on utilise la méthode write() comme suit :

```
#Ecriture des informations dans le fichier
candidats.write(f"{ncin};{nom};{prenom};{age};{decision}\n")
```

2ème étape : Définir la fonction admis() permettant de stocker les informations des candidats admis dans le fichier admis.txt

Il faut à la fois lire les données du fichier concours.txt et écrire dans le fichier admis.txt. On ouvre donc le fichier concours.txt en lecture et admis.txt en écriture comme suit :

```
#Ouverture du fichier candidats.txt en lecture([r]ead)
#Ouverture du fichier admis.txt en écriture([w]rite)
with open("candidats.txt", "r") as candidats:
    with open("admis.txt", "w") as admis:
```

Puis pour chaque candidat du fichier concours.txt (utilisation d'une boucle for pour parcourir chaque ligne du fichier), il faut extraire ses informations afin de savoir s'il est admis ou non.

Pour cela, on supprime de la ligne associée au candidat les caractères non imprimables ("n" pour saut de ligne à la fin de chaque ligne) grâce à la méthode strip().

Puis en remarquant que chaque donnée du candidat (ex : NCIN, nom, ...) est séparé par un point virgule, on peut scinder la chaîne de caractère en une liste de valeurs grâce à la méthode split() comme suit :

```
#Enlève le \n à la fin de la chaîne
donnees = candidat.strip('\n')
#Séparation des différentes données dans un tableau
donnees = donnees.split(';')
#On obtient alors donnees = [ncin, nom, prenom, age, decision]
```

Ensuite, si la donnée d'indice 4 (celle correspondant à la décision) est "admis", alors on ajoute la ligne relative au candidat dans le fichier admis.txt avec la méthode write comme suit :

```
admis.write(candidat)
```

3ème étape : Définir la fonction attente() permettant de stocker les informations des candidats en attente dans le fichier attente.txt

Un candidat est dit "en attente" s'il est admis, mais est âgé de 30 ans ou plus.

Plutôt que de parcourir le fichier concours.txt et regarder si le candidat est admis et est âgé de 30 ans ou plus, afin de gagner en temps de calcul, on va utiliser le fichier admis.txt et regarder si l'âge du candidat admis est supérieur ou égal à 30.

On ouvre donc les fichiers admis.txt en lecture et attente.txt en écriture comme expliqué précédemment, c'est-à-dire :

```
with open("admis.txt", "r") as admis:
    with open("attente.txt", "w") as attente:
```

Puis on extrait les données relatives au candidat comme précédemment (voir 2ème étape).

Enfin, on teste si la donnée d'indice 3 (correspondant à l'âge) est supérieur ou égal à 30. Pour cela, il faut d'abord convertir la donnée d'indice 3 en entier car pour le moment c'est une chaîne de caractères puisque issue d'un fichier texte. On procède donc comme suit :

```
#Si son âge est supérieur ou égal à 30, on le place sur liste
d'attente
if int(donnees[3]) >= 30:
    attente.write(candidat)
```

4ème étape : Définir la fonction statistiques() qui retourne le pourcentage de candidats pour une décision placée en paramètre

Tout d'abord, on regarde si la décision placée en paramètre est valide, c'est-à-dire, si l'utilisateur a saisi admis, refusé ou ajourné.

```
#Test si la décision placée en paramètre existe
if dec not in ('admis', 'refusé', 'ajourné'):
    raise ValueError("Décision invalide.")
```

Puis on ouvre en lecture le fichier concours.txt et on parcourt chaque ligne du fichier (on incrémente un compteur à chaque ligne pour compter le nombre total de candidats.

Ensuite, on extrait les données comme précédemment et on teste si la donnée d'indice 4 correspondant à la décision est identique à celle placée en paramètres. Si ce sont les mêmes, alors on incrémente le compteur correspondant à la décision, comme suit :

```
#Si la décision correspond à celle placée en paramètre
if donnees[4] == dec:
    #On incrémente le nombre de 1
    nb_dec += 1
```

Enfin, on retourne le pourcentage de candidats ayant eu la décision placée en paramètres comme suit :

```
#Calcul du pourcentage
pourcentage = nb_dec / nb_candidats * 100
#Retourne le pourcentage de candidats pour la décision
return pourcentage
```

5ème étape : Définir la fonction `trace_camembert()` qui trace le diagramme en camembert les résultats au concours par décision

Tout d'abord, il faut récupérer les pourcentages de chacune des décisions grâce à la fonction créée précédemment. Pour cela, on procède comme suit :

```
#Récupération des pourcentages pour créer le camembert
pourcentage_admis = statistiques('admis')
pourcentage_refuse = statistiques('refusé')
pourcentage_ajourne = statistiques('ajourné')
```

On crée une liste comportant ces pourcentages :

```
donnees = [pourcentage_admis, pourcentage_refuse,
pourcentage_ajourne]
```

Pour générer un diagramme en camembert, d'après la documentation de la bibliothèque matplotlib, il faut utiliser la fonction `pie` du module `matplotlib.pyplot` comme suit :

```
plt.pie(donnees)
```

On peut éventuellement, afin d'améliorer le rendu visuel, rajouter un titre au graphique, placer des étiquettes sur les données, ajouter une légende et faire en sorte d'obtenir un camembert "plein". D'où le code suivant, qui est une amélioration de celui présenté ci-dessus :

```
#Etiquettes pour les données du graphique avec les valeurs
etiquettes = [f"Admis\n({pourcentage_admis:2.2f}%)",
f"Refusé\n({pourcentage_refuse:2.2f}%)",
f"Ajourné\n({pourcentage_ajourne:2.2f}%)" ]

#Création d'un tableau avec les couleurs
couleurs = ['orange', 'purple', 'blue']

#Création du camembert (attribut normalize pour avoir cercle plein)
plt.pie(donnees, labels=etiquettes, colors=couleurs,
labeldistance=1.15, normalize=True)
```

```
#Ajout d'un titre bleu en gras
plt.title("Exercice 3 - Résultat au concours", fontweight='bold',
color='b', pad='15')

#Mise en place de la légende
plt.legend(['Admis', 'Refusé', 'Ajourné'])
```

Puis on affiche et ferme le camembert avec les méthodes `show()` et `close()` comme suit :

```
#Affichage du camembert
plt.show()
#Fermeture du graphique
plt.close()
```

6ème étape : Définir la fonction `supprimer()` qui supprime du fichier `admis.txt` les candidats en attente

On sait que l'on ne peut pas supprimer réellement une ligne d'un fichier texte avec Python. L'idée que nous avons eu est d'ouvrir une première fois le fichier `admis.txt` en lecture afin de récupérer les données de l'ensemble des candidats admis. Pour cela, on utilise la méthode `readlines()` comme suit :

```
#Ouverture du fichier admis.txt en lecture
with open('admis.txt', 'r') as admis:
    #Récupération de l'ensemble des candidats dans un tableau
    candidats = admis.readlines()
```

Puis on rouvre le fichier en écriture afin d'écraser le fichier `admis.txt`. On a alors un fichier vide. On parcourt ensuite le tableau contenant les candidats défini ci-dessus. On extrait les données du candidat sous forme d'un tableau. Si la donnée 3 du candidat (correspondant à son âge), convertie en entier comme précédemment, est inférieure à 30, alors on l'écrit dans le fichier `admis.txt`.

On a ainsi le code suivant :

```
with open('admis.txt', 'w') as admis:
    #Pour chaque candidat admis
    for candidat in candidats:
        # Extraction des informations relatives au candidat
        donnees = candidat.strip('\n')
        donnees = donnees.split(';')
        #Si son âge est supérieur ou égal à 30, on ne le ré-ajoute
        pas au fichier des admis
        if int(donnees[3]) <= 30:
            admis.write(candidat)
```

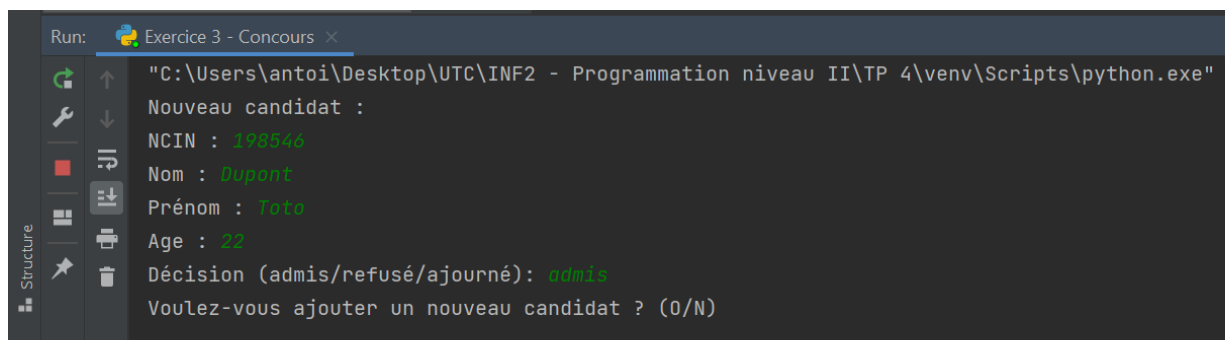
7ème étape : Concevoir le programme principal

On construit le programme principal en appelant successivement les différentes fonctions réaliser. Dans un premier temps, on demande à l'utilisateur de saisir les données des candidats, puis on sépare dans 2 fichiers différents les candidats admis et en attente. Enfin, on affiche le camembert relatif au concours et on supprime du fichier admis.txt les candidats en attente.

```
#Programme principal
def main():
    #Saisie des candidats
    saisir()
    #Création du fichier avec les admis
    admis()
    #Création du fichier avec les admis en attente
    attente()
    #Affichage du camembert
    trace_camembert()
    #Suppression des personnes âgées de plus de 30 ans au fichier
    des admis
    supprimer()
```

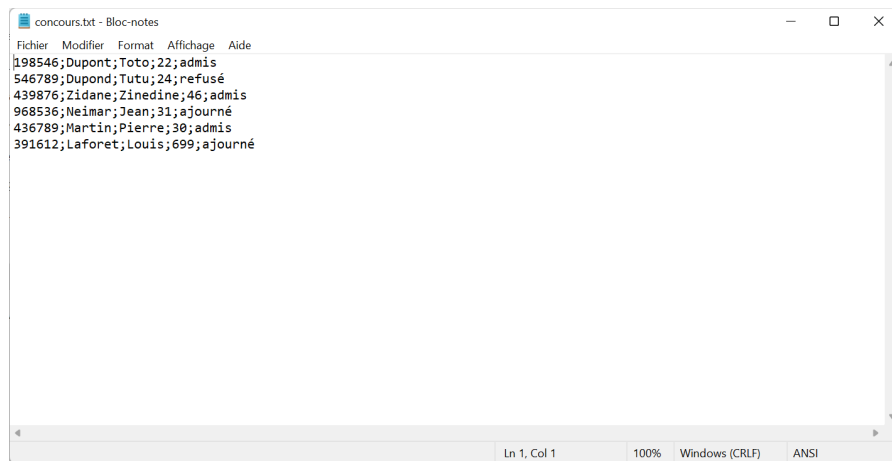
★ Solution apportée :

Ainsi, nous avons conçu pas à pas le programme demandé. En voici des captures d'écran des résultats obtenus :



```
Run: Exercice 3 - Concours x
"C:\Users\anto\ Desktop\UTC\INF2 - Programmation niveau II\TP 4\venv\Scripts\python.exe"
Nouveau candidat :
NCIN : 198546
Nom : Dupont
Prénom : Toto
Age : 22
Décision (admis/refusé/ajourné): admis
Voulez-vous ajouter un nouveau candidat ? (O/N)
```

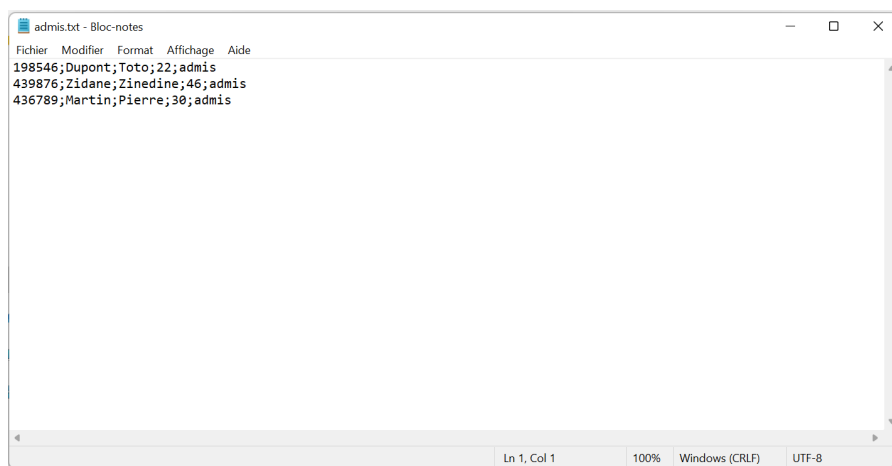
Figure 7 : Saisie des candidats par l'utilisateur (fonction saisie)



```
Fichier  Modifier  Format  Affichage  Aide
198546;Dupont;Toto;22;admis
546789;Dupond;Tutu;24;refusé
439876;Zidane;Zinedine;46;admis
968536;Neymar;Jean;31;ajourné
436789;Martin;Pierre;30;admis
391612;Laforet;Louis;699;ajourné

Ln 1, Col 1    100%  Windows (CRLF)  ANSI
```

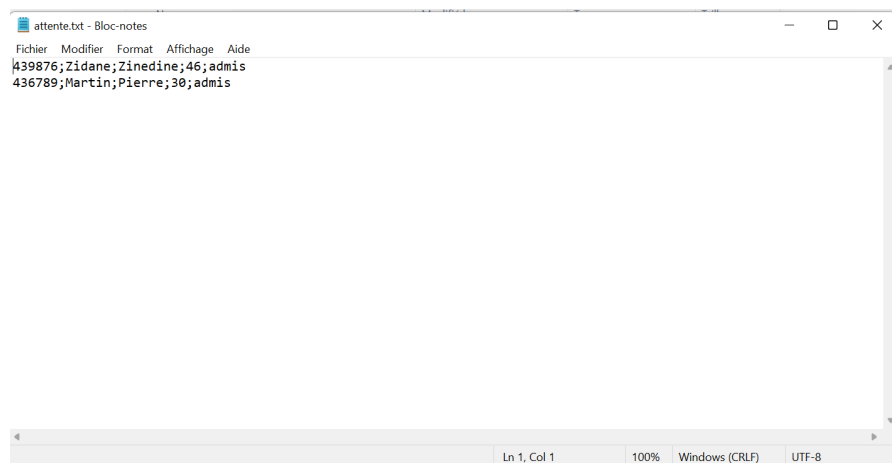
Figure 8 : Fichier concours.txt après saisie des candidats par l'utilisateur



```
Fichier  Modifier  Format  Affichage  Aide
198546;Dupont;Toto;22;admis
439876;Zidane;Zinedine;46;admis
436789;Martin;Pierre;30;admis

Ln 1, Col 1    100%  Windows (CRLF)  UTF-8
```

Figure 9 : Fichier admis.txt après appel de la fonction admis()
(A cette étape, les candidats de plus de 30 ans sont toujours inscrits dans ce fichier)



```
Fichier  Modifier  Format  Affichage  Aide
439876;Zidane;Zinedine;46;admis
436789;Martin;Pierre;30;admis

Ln 1, Col 1    100%  Windows (CRLF)  UTF-8
```

Figure 10 : Fichier attente.txt après appel de la fonction attente()

Exercice 3 - Résultat au concours

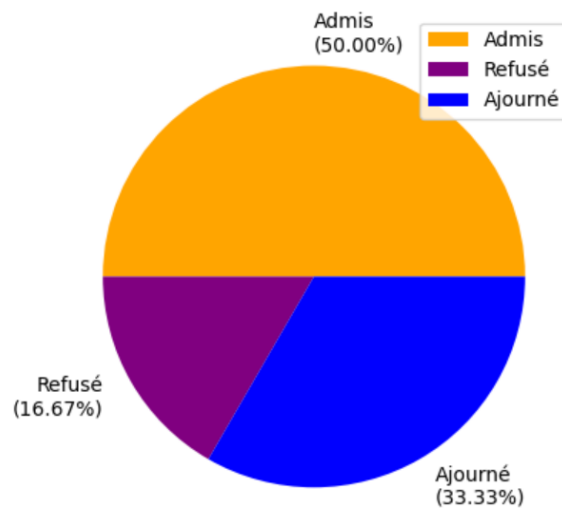


Figure 11 : Résultat au concours sous forme de diagramme (fonction `tracer_camembert`)

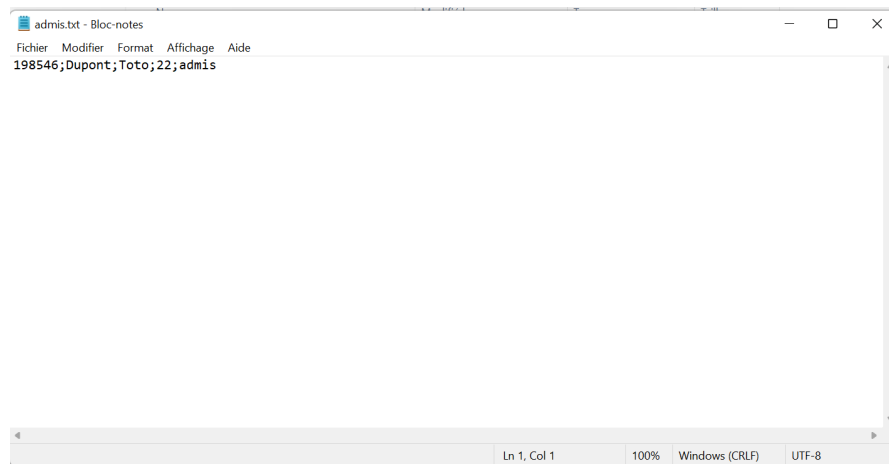


Figure 12 : Fichier `admis.txt` après appel de la fonction `supprimer()`
(Les admis de 30 ans et plus ont été supprimés)