Chapitre 12. Autres langages, modules et bibliothèques (Partie 2)

Table des matières

Remplissez le jupyter notebook suivant en vous aidant de votre livre de Première NSI de Serge BAYS .

- Pour répondre, double-cliquez sur **Réponse** et complétez la zone en-dessous. Puis cliquez sur le bouton *Exécuter*.
- **Important : pour fermer votre jupyter notebook, cliquez sur :**

Fichier / Créer une nouvelle sauvegarde

puis sur :

Fichier / Fermer et Arrêter

- Ecrivez ci-dessous votre prénom et votre nom :

Réponse :

Chapitre 12. Autres langages, modules et bibliothèques

2. Modules et bibliothèques

2.1 Faire un module "mesfonctions"

Lisez le paragraphe **Modules et bibliothèques** du milieu de la p. 21 au milieu de la p. 22

Nous allons créer un module contenant quelques fonctions.

Copier les lignes ci-dessous dans un fichier spyder dont le nom sera mesfonctions.py Vous l'enregistrerez dans le même dossier que ce fichier jupyternotebook.

```
"""
Fichier contenant mes fonctions
"""

def maximum(a,b):
    """ la fonction renvoie le maximum des deux nombres

    paramètres
    a : float : un des deux nombres à comparer
    b : float : l'autre nombre à comparer

    Return
    max : float : la valeur la plus petite entre a et b

    """

    if a > b:
        max = a
    else :
        max = b
    return max

def minimum(a,b):
    """ la fonction renvoie le minimum des deux nombres

    paramètres
    a : float : un des deux nombres à comparer
    b : float : l'autre nombre à comparer

    Return
    min : float : la valeur la plus petite entre a et b

    """
    if a > b:
        min = b
    else :
        min = a
    return min
```

pour voir si cela fonctionne Exécutez le programme ci-dessous

In [ ]:
```
import mesfonctions

mesfonctions.minimum(4,8)
```

1) Quelle différence y a-t-il entre :

- import mesfonctions

et

- from mesfonctions import maximum ?

Réponse :

2) créez dans votre module quatre fonctions

- La fonction *moyenne_pond* renvoyant la moyenne de nombres se trouvant dans une liste [[x1, n1], [x2, n2], ..., [xk, nk]] contenant les nombres x_i et l'effectif n_i associé.
- La fonction *mediane_pond* renvoyant la médiane d'une liste contenant les nombres et l'effectif associé
- La fonction *max_pond* renvoyant la ou les valeurs, sous la forme d'une liste, ayant l'effectif le plus grand
- La fonction *min_pond* renvoyant la ou les valeurs, sous la forme d'une liste, ayant l'effectif le plus petit

- Ecrivez dans le fichier mesfonctions.py le code de vos quatre fonctions.
- Puis testez le bon fonctionnement en exécutant la cellule ci-dessous.

In [ ]:
```
#Pour tester vos fonctions qui auront ét ajoutées au fichier mesfonctions.py
from mesfonctions import *

# Voici les tests avec la liste
L = [[2, 5], [3, 1], [4, 6], [5, 3], [7, 4], [8, 5], [9, 2], [12, 4]]

# Vous devez obtenir
# moyenne :  6.2
# valeur(s) ayant l'effectif le plus grand :  [4]
# Médiane :  6.0
# valeur(s) ayant l'effectif le plus petit :  [3]

print("moyenne : ", moyenne_pond(L))
print("valeur(s) ayant l'effectif le plus grand : ", max_pond(L))
print("Médiane : ", mediane_pond(L))
print("valeur(s) ayant l'effectif le plus petit : ", min_pond(L))
```

- Vous joindrez votre fichier mesfonctions.py à ce fichier jupyternotebook pour la correction.

2.2 Module math

Lisez le paragraphe **Module math** du milieu de la p. 22 au bas de la p. 23

3) Que permet la fonction *dir* ?

Réponse :

4) Appliquer cette fonction à votre module en tapant et exécutant cette instruction dans la ligne ci-dessous.

In [ ]:
```
import mesfonctions

dir(mesfonctions)
```

5) Que permet la fonction *help* ?

Réponse :

6) Appliquer cette aide à la fonction mediane_pond de votre module mesfonctions. Qu'allez-vous saisir ?

In [ ]:
```
#  Ecrire ci-dessous les instructions à saisir pour répondre ensuite à la question
```

Réponse :

7) Donner la description de la fonction *sqrt* mais avant vous donnerez les instructions à saisir.

In [ ]:
```
#  Ecrire ci-dessous les instructions à saisir pour répondre ensuite à la question
```

Réponse :

8) Donner la description de la fonction *fmod*. Donner avant les instructions à saisir

In [ ]:
```
#Ecrire ci-dessous les instructions à saisir pour répondre ensuite à la question
```

Réponse :

9) Quelle est la différence entre *fmod* et % ?

In [ ]:
```
#  Exécutez le code ci-dessous et répondez ensuite à la question.

print("Le reste de la division de 9 par 4 est ", 9%4)
print("Le reste de la division de -9 par 4 est ", -9%4)

import math
print("Le reste de la division de 9 par 4 est ", math.fmod(9, 4))
print("Le reste de la division de -9 par 4 est ", math.fmod(-9, 4))
```

Réponse :

10) Que fait la fonction *gcd* ? Ecrire ensuite une fonction qui renvoie la fraction simplifiée (s'il est possible de la simplifier). Seront passés en paramètre le numérateur et le dénominateur. La fonction renverra, s'il y a simplification, un tuple (numérateur, dénominateur)

In [ ]:
```
import math
dir(math)

help(math.gcd)
```

Réponse :

In [ ]:
```
#  Réponse

from math import gcd

def simplification(num : int , denom : int):
    .....    #    A compléter


#Pour tester
simplification(121, 44)
```

2.3 Module random

Lisez le paragraphe **Module random** du bas de la p. 23 au haut de la p. 25

11) La fonction *randomize* est-elle une fonction du module random ? Comment avez-vous trouvé la réponse (hors recherches sur le web) ?

In [ ]:
```
import random
dir(random)
```

Réponse :

12) Ecrire une fonction qui simule 100 lancers de deux dés dont les faces sont numérotées de 1 à 6 et qui renvoie une table contenant pour chaque somme l'effectif correspondant.
Le nombre de lancers est entré comme paramètre de cette fonction
N'oubliez pas la docstring.

Dans un permier temps, lisez l'aide sur la fonction random.randint

In [ ]:
```
import random

help(random.randint)
```

In [ ]:
```
from random import *

def repartition(nombre : int): #Compléter la fonction
    """




    """

    ........


#Pour tester
resultats = repartition(100)
print(resultats)
for i in range(0, 11):
    print(i+2,"--", resultats[i][1])
```

2.4 Module Turtle

Lisez le paragraphe **Module Turtle** p. 25

13) Que permet le module Turtle ?

Réponse :

turtle est un module de Python permettant de faire avancer une « pointe de crayon » portée par une « tortue » sur une « feuille ».
Cette feuille est munie d'un repère dont l'origine a pour coordonnées (0, 0) situé au centre.
Il est possible de dire à la tortue d'aller à un point de coordonnées (-30, 150) par exemple. Il est possible de lui dire si elle doit lever la pointe de crayon (donc elle avancera sans tracer de trait) ou de baisser la pointe de crayon (donc elle avancera en traçant). Il est possible de lui dire d'avancer de 25 pixels avec forward(25) ou de reculer de 10 pixels avec backward(10). Il est possible de lui dire de tourner à gauche d'un angle de 45° avec left(45) ou de tourner à droite d'un angle de 30° avec right(30) etc...

Un programme devra toujours débuter par : **from turtle import ***

Les principales fonctions du module turtle sont :

Mouvement de la Tortue :

`forward(d)` -> Avancer d'une distance d (en pixels)
`backward(d)` -> Reculer d'une distance d (en pixels)
`goto(x,y)` -> Positionner la tortue au point de coordonnées (x ; y)
`setx(x)` -> Définit la première coordonnée de la tortue à x, en laissant la deuxième coordonnée inchangée.
`sety(y)` -> Définit la deuxième coordonnée de la tortue à y, en laissant la première coordonnée inchangée.
`xcor()` -> Renvoie la coordonnée x de la tortue.
`ycor()` -> Renvoie la coordonnée y de la tortue.
`color(couleur)` -> Couleur peut être une chaîne prédéfinie ('red', 'blue', 'green', etc.)
`left(a)` -> Fait pivoter la tortue d'un angle a degrés vers la gauche
`right(a)` -> Fait pivoter la tortue d'un angle de a degrés vers la droite
`circle(r)` -> trace un cercle de rayon r, le point de départ de la tortue appartient au cercle (attention il n'est pas centré sur la position de la tortue)
`circle(r,s)` -> trace un arc de cercle correspondant à s degrés
`dot(d,c)` -> dessine un disque de diamètre d et de couleur c là où est la tortue
`setheading(a)` -> permet de fixer un cap absolu à la tortue (a est en degrés)
`screensize(width, height, fond)` -> permet de définir la taille du canevas. *width* : nombre entier positif, nouvelle largeur du canevas en pixels, *height* : nombre entier positif, nouvelle hauteur du canevas, en pixels, *fond* : chaîne de caractères indiquant la couleur de fond

Contrôle du stylo :

`up()` -> lève le stylo
`down()` -> repose le stylo
`width(épaisseur)` -> Choisir l'épaisseur du tracé (en pixels)
`reset()` -> nettoie la fenêtre de dessin, réinitialise la tortue ; elle est située alors au centre de l'écran de dessin tournée vers la droite.
`color(c)` -> la couleur par défaut est le noir, on peut la changer en mettant une couleur prédéfinie c : 'red','green', 'blue', 'yellow', . . .
`backward(taille)` -> recule de `taille` pixels, où `taille` doit être déclaré avant
`fillcolor(c)` -> Remplit une figure fermée à l'aide de la couleur demandée c.
les balises `begin_fill()` et `end_fill()` permettent de commencer et de terminer le remplissage d'une figure géométrique.
`pencolor((1,0,1))` -> met la couleur du stylo en *magenta*
`pencolor((0.1,0,0.1))` -> met la couleur du stylo en *magenta* très foncé
`speed(v)` -> "fastest" accélère le tracé et "slowest" le ralenti
`circle(r)` -> trace un cercle de rayon `r` , où `r` doit être déclaré avant
`write(texte)` -> texte doit être une chaîne de caractères délimitée avec des " ou des '

Vous pouvez aussi aller voir la documentationofficielle python : http://docs.python.org/3.2/library/turtle.html (http://docs.python.org/3.2/library/turtle.html)

- Exécutez la cellule ci-dessous pour voit les fonctions du module Turtle :

In [ ]:
```
import turtle
dir(turtle)
```

Voici trois exemples.

- Vous verrez que la tortue exécute un dessin dans une fenêtre 'Python Graphic Turtle' séparée.
- Avant le code Turtle, il y a un bout de code nommé "boilerplate". Le terme "boilerplate" vient de l'industrie de la presse américaine et signifie littéralement "histoires prêtes à l'emploi" pour désigner des parties standards prêtes à être réutilisées pour un autre usage.
- Ce "bout de code boilerplate" permet de **gérer l'erreur 'Terminator'** qui est relevée au lancement du programme.
- Après le code Turtle, la procédure *exitonclick()* du module *turtle* est exécutée. Elle permet de fermer par un simple clic dans la fenêtre 'Python Graphic Turtle' après la fin du dessin.
  Cela **évite le plantage du "programme noyau" de Python**, à la fermeture de cette fenêtre.

In [ ]:

```
# Importation du module turtle
from turtle import *


#  Code boilerplate
try:
    reset()
except Terminator:
    pass

# ----------------- Code Turtle ICI ---------

def exemple():
    goto(-100,-100)
    clear()
    fillcolor("green")
    begin_fill()
    forward(200)
    left(120)
    pencolor((1,0.5,0))
    forward(400)
    end_fill()


speed("slowest")
exemple()


# ---------------------------------------------

exitonclick()   #  Permet de fermer la fenêtre 'Python Turtle Graphics' par
                #  un clic dedans, lorsque le dessin est terminé.
```

In [ ]:
```
# Importation du module turtle
from turtle import *


#  Code boilerplate
try:
    reset()
except Terminator:
    pass

# ----------------- Code Turtle ICI ---------

a=0
down()
while a<12:
    a=a+1
    forward(150)
    left(150)
up()



# -------------------------------------------

exitonclick()  #  Permet de fermer la fenêtre 'Python Turtle Graphics' par
               #  un clic dedans, lorsque le dessin est terminé.
```

In [ ]:
```
# Importation du module turtle
from turtle import *


#  Code boilerplate
try:
    reset()
except Terminator:
    pass

# ----------------- Code Turtle ICI ---------

color('red', 'yellow')
begin_fill()
forward(200)
left(170)
while abs(pos()) > 1:
    forward(200)
    left(170)
end_fill()


# -------------------------------------------

exitonclick()   #  Permet de fermer la fenêtre 'Python Turtle Graphics' par
                #  un clic dedans, lorsque le dessin est terminé.
```

14) Ecrire une procédure *polygone* permettant de tracer un polygone régulier dont la longueur des côtés et le nombre de côtés sont passés en paramètre.

In [ ]:
```
# Importation du module turtle
import turtle as tu
from math import sin, tan, pi


#  Code boilerplate
try:
    tu.reset()
except Terminator:
    pass

# ----------------- Code Turtle ICI ---------

#Le centre du polygone sera l'origine du repère

def polygone(long, nbcotes) :
    """




    """

    ..........

tu.reset()
polygone(100,5)


# -------------------------------------------

tu.exitonclick()    #  Permet de fermer la fenêtre 'Python Turtle Graphics' par
                    #  un clic dedans, lorsque le dessin est terminé.
```

15) Ecrire les procédures suivantes en respectant la docstring :

In [ ]:
```
##  Exercice 2

from turtle import *


#  Code boilerplate
try:
    reset()
except Terminator:
    pass


# ----------------- Code Turtle ICI ---------

def triangle_haut(long):
    """
    Cette procédure dessine un triangle équilatéral dont les côtés sont de longueur n
et qui a la
    pointe vers le haut.
    paramètre
    ---------
    long : entier : Longueur en pixels des côtés

    Retourne
    ---------
    Rien c'est une procédure
    """

    ...........


def triangle_bas(long):
    """
    Cette procédure dessine un triangle équilatéral dont les côtés sont de longueur n
et qui a la
    pointe vers le bas.
    paramètre
    ---------
    long : entier : Longueur en pixels des côtés

    Retourne
    ---------
    Rien c'est une procédure
    """

    ...........



def triangle_oriente(long, angle):
    """
    Cette procédure dessine un triangle équilatéral dont les côtés sont de longueur n
et
    d'une orientation bien déterminées. Cet angle est déterminé par rapport à l'orient
ation initiale vers la droite
    et est dans le sens contraire des aiguilles d'une montre.
    Le triangle de référence étant le triangle avec la pointe vers le haut.
    paramètre
    ---------
```

```
    n : entier : Longueur en pixels des côtés

    Retourne
    ---------
    Rien c'est une procédure
    """

    ..........



# -------------------------------------------

reset()
up()
n = 150
screensize(600, 600)
goto(-n, -n)
triangle_haut(n)
goto(n, n)
triangle_bas(n)
goto(-n, -n)
triangle_oriente(n, 30)

exitonclick()   #  Permet de fermer la fenêtre 'Python Turtle Graphics' par
                #  un clic dedans, lorsque le dessin est terminé.
```

16) Ecrire les procédures suivantes en respectant la docstring :

In [ ]:
```
##  Exercice 3

from turtle import *

#  Code boilerplate
try:
    reset()
except Terminator:
    pass


# ----------------- Code Turtle ICI ---------

def carre(long):
    """
    Cette procédure trace un carre de côté long. Il est préférable que la tortue termi
ne son dessin là où
    elle a démarré et avec la même orientation.

    paramètre
    ---------
    long : entier : Longueur en pixels des côtés du carré

    Retourne
    ---------
    Rien c'est une procédure
    """

    ...........



def ligne_de_carres(long, nb):
    """
    Cette procédure trace n carrés sur une ligne chaque carré étant de coté long (on u
tilisera la fonction carre).

    paramètre
    ---------
    long : entier : Longueur en pixels des côtés des carrés
    nb : entier : nombre de carrés


    Retourne
    ---------
    Rien c'est une procédure
    """


    ...........



def carre_croissant(long, nb):
    """
    Cette procédure trace une ligne de carrés, le premier carré étant de côté long, le
suivant de taille 1,25 fois
    la taille du carré qui le précède ; les carrés seront espacés la première fois de
long/4 puis cette distance sera
```

```
    multipliée aussi par 1,25 à chaque fois.
    Remarque : On utilisera la fonction carre mais pas ligne_de_carres.

    paramètre
    ---------
    long : entier : Longueur en pixels des côtés du premier carré
    nb : entier : nombre de carrés

    Retourne
    ---------
    Rien c'est une procédure
    """


    ...........



#  ---------------- tests ----------------

reset()
up()
n = 40

goto(-300, 200)
carre(n)

goto(-300, 0)
ligne_de_carres(n, 8)

goto(-300, -200)
carre_croissant(n, 7)


exitonclick()    #  Permet de fermer la fenêtre 'Python Turtle Graphics' par
                 #  un clic dedans, lorsque le dessin est terminé.
```

2.5 Bibliothèque Matplotlib

Lisez le paragraphe **Module Matplotlib** p. 26

Affichage de graphiques

Une bibliothèque Python (library en anglais) est un ensemble de fonctions, de constantes, de types et de modules.

La bibliothèque standard de Python contient les modules *math* et *random* parmi beaucoup d'autres.

La bibliothèque Matplotlib contient le module *pyplot* utilisé pour le tracé de courbes.

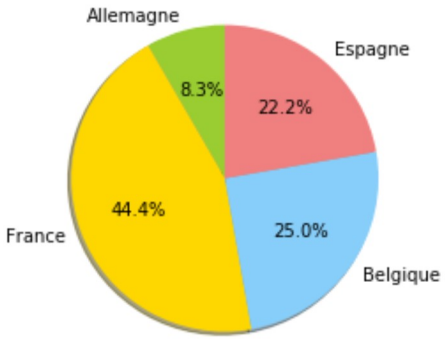
N'hésitez pas à utiliser l'aide interactive disponible dans pyplot, en tapant plt(, puis la touche Tab, pour obtenir la syntaxe et les différentes options possibles.

In [ ]:
```
import matplotlib.pyplot as plt

plt(
```

Voici, ci-dessous, un exemple de tracé d'un diagramme circulaire disponible par la fonction pie du module *pyplot*.
La liste et le paramètre colors ne sont pas obligatoires. Si vous ne les mettez pas, les couleurs seront choisies par la fonction de traçage.

Voici un exemple de diagramme circulaire :

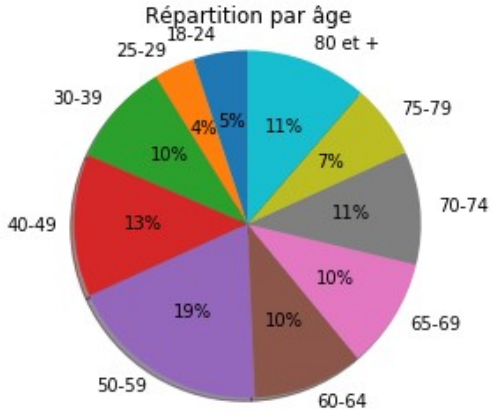
In [2]:
```
#   La ligne qui suit est nécessaire quand on veut utiliser matplotlib dans un jupyter notebook.
%matplotlib inline

import matplotlib.pyplot as plt

labels = ['Allemagne', 'France', 'Belgique', 'Espagne']
sizes = [15, 80, 45, 40]
colors = ['yellowgreen', 'gold', 'lightskyblue', 'lightcoral']

plt.pie(sizes, labels=labels, colors=colors,
        autopct='%1.1f%%', shadow=True, startangle=90)

#plt.savefig('diag_circulaire.png')  #Si vous voulez sauvegarder le graphique
plt.show()
```



Les données ainsi que la structure des fichiers concernant la covid sont accessibles en bas de page sur ce site : https://www.data.gouv.fr/fr/datasets/donnees-relatives-aux-personnes-vaccinees-contre-la-covid-19-1/ (https://www.data.gouv.fr/fr/datasets/donnees-relatives-aux-personnes-vaccinees-contre-la-covid-19-1/)

Nous allons utiliser les données concernant la répartition par âge des personnes vaccinées en France

Voilà ce que nous voulons obtenir (ce diagramme correspond à la situation au 7 juin 2021. (Le programme résupérant les dernières données vos données seront plus récentes).



la table des données a pour première ligne : ['fra', 'clage_vacsi', 'jour', 'n_tot_dose1', 'n_tot_complet', 'pop', 'couv_tot_dose1', 'couv_tot_complet']

17) A quoi cela correspond-il ?

Réponse :

18) A quoi correspond 'jour' ? Quel est le type des données correspondant à ce champ ?

Réponse :

Voici le programme permettant de récupérer les données sur le site du gouvernement et de créer la table total_ligne

In [ ]:
```
import requests

#  Evolution vaccination par classe d'age.
#  Récupératon du fichier mis en ligne ce jour.
vaccin_age1_2 = requests.get('https://www.data.gouv.fr/fr/datasets/r/dc103057-d933-4e4b-bdbf-36d312af9ca9')

champ=""
ligne=[]
total_ligne=[]
for i in range(len(vaccin_age1_2.text)):
    if vaccin_age1_2.text[i]!="\n":     #  Si je ne suis pas en fin de ligne.
        if vaccin_age1_2.text[i]==";": #  Si j'arrive à la fin d'un champ de données.
            ligne.append(champ)         #  Ajout du champ dans la liste ligne.
            champ=""                    #  Le champ étant noté, on repart à 'vide' pour le champ suivant.
        else:
            champ = champ + vaccin_age1_2.text[i]  #  On ajoute le caractère au contenu du champ.
    else:
        ligne.append(champ)             #  Nous sommes en fin de ligne, donc on ajoute le
                                        #  dernier champ dans la liste ligne.
        total_ligne.append(ligne)       #  On ajoute la ligne à la table total_ligne.
        ligne=[]                        #  La ligne étant complète on réinitialise pour la suivante.
        champ=""                        #  idem le champs est à réinitialiser.
```

In [ ]:
```
print(total_ligne)
```

19) Que contient total_ligne[i][1]) ? (i étant un entier supérieur à 1)

Réponse :

Voici une fonction qui sera appelée lors de la création des graphiques

In [ ]:
```
def lib_etiquettes(nombre):
    '''
    renvoie l'étiquette correspondant au nombre
    9 : 0-9
    17 : 10-17
    24 : 18-24
    29 : 25-29
    39 : 30-39
    49 : 40-49
    59 : 50-59
    64 : 60-64
    69 : 65-69
    74 : 70-74
    79 : 75-79
    80 : 80 et +

    paramètres
    nombre : entier : nombre correspondant à la classe d'age

    retourne
    l'étiquette correspondante
    '''
    #Initialisation du dictionnaire contenant les libellés des classes d'âges.
    dico = {9 : '0-9' , 17 : '10-17', 24 : '18-24', 29 : '25-29', 39 : '30-39', 49 : '
40-49', 59 : '50-59', 64 : '60-64', 69 : '65-69',
            74 : '70-74', 79 : '75-79', 80 : '80 et +'}

    return dico[nombre]    #Retourne l'étiquette correspondant au champs âge.
```

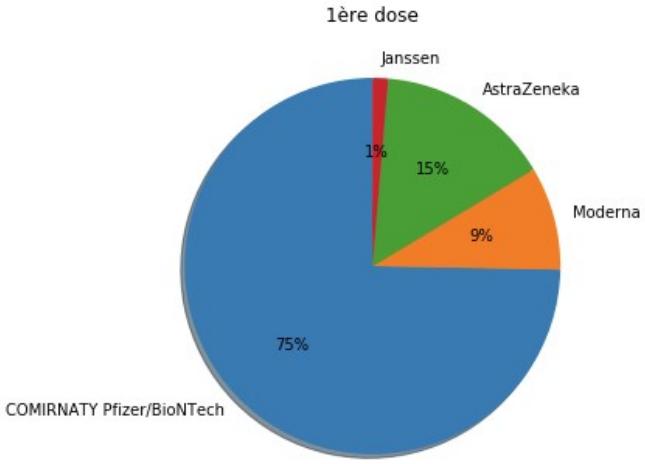
20) Expliquer cette instruction : lib_etiquettes(int(total_ligne[i][1])). (i est un nombre entier plus grand que 1)

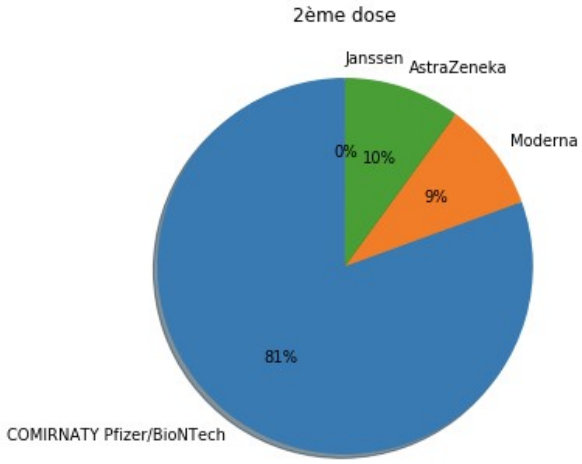
Réponse :

21) Compléter le programme pour que le diagramme circulaire s'affiche.
La première chose à faire sera de bien comprendre la stucture de la table total_ligne

In [ ]:
```
#   La ligne qui suit est nécessaire quand on veut utiliser matplotlib dans un jupyter notebook.
%matplotlib inline

import matplotlib.pyplot as plt

#Répartition des doses par âge
nombre_doses = []  #Initialisation des doses par catégories d'âges
etiquettes = []    #Initialmisation des étiquettes utiles

for i in range(1,len(total_ligne)-1):      #On parcours les 'sous'listes
    .....   #A compléter, récupérer les étiquettes en utilisant lib_etiquette et les ajouter à la liste des étiquettes
    .....  #A compléter, récupérer les données correspondants à la classe d'âge correspondant

plt.title ('Répartition par âge')
plt.pie(nombre_doses, labels=etiquettes, autopct='%1.f%%', shadow=True, startangle=90)

plt.savefig('repartition_au_'+total_ligne[1][2]+'.png')
plt.show()
```

On veut maintenant afficher la répartition des vaccins en fonction de leurs noms.
On va afficher deux diagrammes circulaires, un pour la répartition pour la première dose et un autre pour la seconde dose





Attention ce graphique est celui au 7 juin

In [ ]:
```
#Récupération des données

#evolution vaccination par type
#1 : COMIRNATY Pfizer/BioNTech
#2 : Moderna
#3 : AstraZeneka
#4 : Janssen
vaccin_type = requests.get('https://www.data.gouv.fr/fr/datasets/r/b8d4eb4c-d0ae-4af6-bb23-0e39f70262bd')
#print(vaccin_dose1_2.text)

champ=""
ligne=[]
total_ligne=[]
for i in range(len(vaccin_type.text)):
    if vaccin_type.text[i]!="\n":
        if vaccin_type.text[i]==";":
            ligne.append(champ)
            champ=""
        else:
            champ = champ + vaccin_type.text[i]
    else:
        ligne.append(champ)
        total_ligne.append(ligne)
        ligne=[]
        champ=""
```

22) Quels sont les noms des champs ?

Réponse :

23) A quoi correspondent les quatres premières lignes de données à partir des lignes d'index 1 ?

Réponse :

24) A quoi correspond la ligne de données d'index 5 ?

Réponse :

Explicaton de la fonction subplot

Paramètres m,n,p

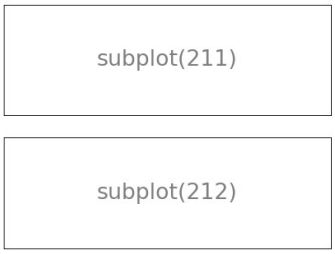
```
entiers positifs
```

mnp

```
un entier avec mnp comme notation décimale
```

Description

subplot(m,n,p) ou subplot(mnp) divise la fenêtre graphique courante en une matrice m x n matrice de sous fenêtres et sélectionne la p-ième sous-fenêtre comme emplacement de dessin par défaut. Le numéro de la sous-fenêtre est compté ligne par ligne, c'est à dire que l'emplacement (i,j) de la matrice porte le numéro (i-1)*n + j.

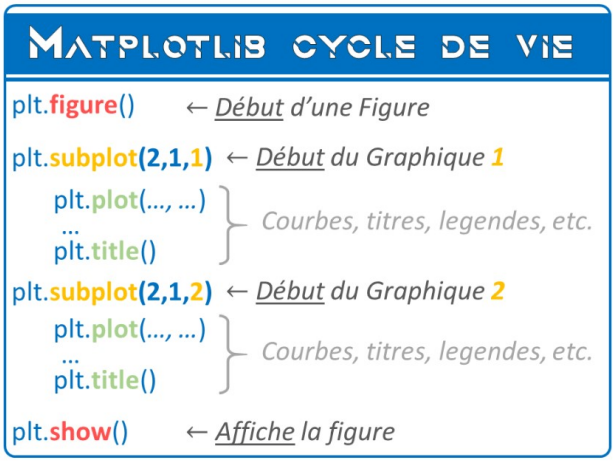


organisation en une colonne et deux lignes

organisation en deux colonnes et deux lignes

organisation en deux colonne et une lignes

et voici une explication en terme de programmation

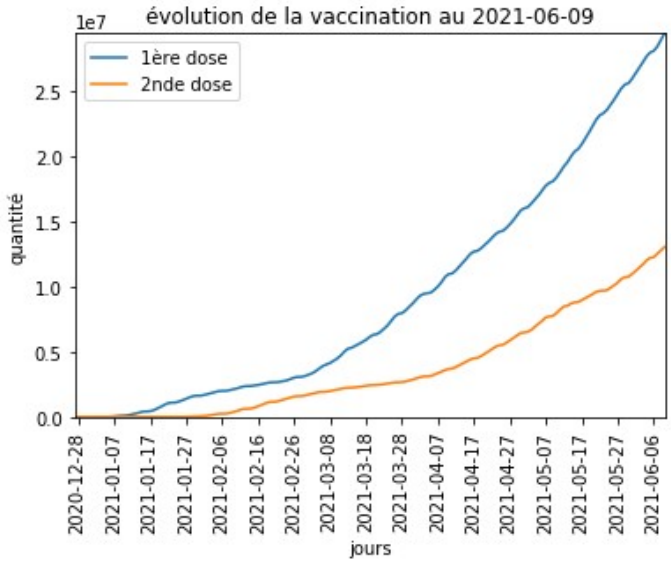


25) Compléter le programme pour qu'il affiche deux diagrammes circulaires

In [ ]:
```
#   La ligne qui suit est nécessaire quand on veut utiliser matplotlib dans un jupyter notebook.
%matplotlib inline

import requests, csv

import matplotlib.pyplot as plt



#Répartition au jour donné de la répartition par vaccins

etiquettes = ['COMIRNATY Pfizer/BioNTech','Moderna','AstraZeneka','Janssen'] #définir les étiquettes
tot_dose_1 = [] #intialisation de la liste des quantités par vaccins pour la première dose dans l'ordre d'apparaition dans la table
tot_dose_2 = [] #intialisation de la liste des quantités par vaccins pour la seconde dose dans l'ordre d'apparaition dans la table
for i in range(1,len(total_ligne)-1):  #Parcours de la table des données quel les données sur les 4 vaccins
    .....  #ajouter à tot_dose_1 la quantité de doses injectées pour le vaccin codé i
    .....  #ajouter à tot_dose_2 la quantité de doses injectées pour le vaccin codé i

# création de la fenêtre graphique 1
plt.figure(1,figsize=(12,10))
#définition des marges
plt.gcf().subplots_adjust(left = 0.125, bottom = 0.2, right = 1,
                          top = 0.9, wspace = 0.5, hspace = 0)

plt.title ('répartition par type de vaccin')
plt.subplot(1,2,1)
#partage la figure en 2 x 1 emplacements de graphes (2 lignes et 1 colonne)
#sélectionne le 1er emplacement pour les instructions graphiques suivantes.
#Les numéros des graphes sont comptés par ligne.
plt.title ('1ère dose')
plt.pie(tot_dose_1, labels=etiquettes, autopct='%1.0f%%', shadow=True, startangle=90)
plt.subplot(1,2,2)
plt.title ('2ème dose')
plt.pie(tot_dose_2, labels=etiquettes, autopct='%1.0f%%', shadow=True, startangle=90)

#Sauvegarder l'image dans un fichier dont le nom sera de la forme "repartition_par_type_de_vaccin_au_ .... .png"
#Sur les pointillés apparaîtra la date des données exemple : repartition_par_type_de_vaccin_au_2021-06-08.png
plt.savefig('repartition_par_type_de_vaccin_au_'+total_ligne[1][2]+'.png',orientation='landscape')
plt.show()
```

On veut maintenant voir l'évolution des personnes vaccinées.

24) Compléter le programme pour afficher ce graphique.



Attention ce graphique est celui au 9 juin

In [ ]:
```
#   La ligne qui suit est nécessaire quand on veut utiliser matplotlib dans un jupyter
notebook.
%matplotlib inline
import requests, csv
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
import datetime as dt

##répartition des vaccins par type
vaccin_total = requests.get('https://www.data.gouv.fr/fr/datasets/r/b273cf3b-e9de-437c
-af55-eda5979e92fc')

champ=""
ligne=[]
total_ligne=[]
for i in range(len(vaccin_total.text)):
    if vaccin_total.text[i]!="\n":
        if vaccin_total.text[i]==";":
            ligne.append(champ)
            champ=""
        else:
            champ = champ + vaccin_total.text[i]
    else:
        ligne.append(champ)
        total_ligne.append(ligne)
        ligne=[]
        champ=""


#..............

etiquettes = []
abscisses = []
tot_dose_1 = []
tot_dose_2 = []

x = 0   #Utile si l'on veut afficher une abscsse type 'rang' à la place de la date
for i in range(1,len(total_ligne)):
    if (total_ligne[i][1] == '0'):
        tot_dose_1.append(int(total_ligne[i][5]))
        tot_dose_2.append(int(total_ligne[i][6]))
        etiquettes.append(total_ligne[i][2])
        abscisses.append(x)    #Utile si l'on veut afficher une abscsse type 'rang' à
la place de la date
        x = x + 1              #Utile si l'on veut afficher une abscsse type 'rang' à
la place de la date


plt.figure(figsize=(6,4))

ax = plt.axes()  #  Création d'un objet de la classe Axes

# Taille du graphe
xmin = min(etiquettes)  #si on veut le rang il faut remplacer étiquette par abscisses
ymin = 0
xmax = max(etiquettes)   #si on veut le rang il faut remplacer étiquette par abscisses
ymax = max(tot_dose_1+ tot_dose_2)

plt.xticks(rotation = 'vertical')
```

```
# Affichage des segments.
ax.plot(etiquettes, tot_dose_1,label="1ère dose") # Affichage des segments
ax.plot(etiquettes, tot_dose_2,label="2nde dose") # Affichage des segments
ax.set(xlim=(xmin, xmax), ylim=(ymin, ymax))


ax.xaxis.set_ticks(range(1,len(etiquettes),10))  #Affiche une étiquette toutes les 10 étiquettes

plt.title("évolution de la vaccination au "+total_ligne[-1][2])
plt.xlabel("jours")
plt.ylabel("quantité")
plt.legend()
#plt.gcf().subplots_adjust(left = 0, bottom = -0.5, right = 1, top = 0, wspace = 1, hspace = 0)

plt.savefig('evolution_'+total_ligne[-1][2]+'.png', bbox_inches = 'tight')

plt.show()
```

25) Quels instructions saisir pour avoir de l'aide sur la fonction linspace du module numpy sans recherche sur le web ?

Réponse :

Il faut saisir : .....

In [ ]:
```
from numpy import linspace
help(linspace)
```

26) Que va renvoyer linspace(-3, 3, 20) ?

Réponse :

27) Compléter le programme ci-dessous pour qu'il affiche la représentation graphique de la fonction cosinus entre -3 et 3 en affichant 40 points

In [ ]:

```
#   La ligne qui suit est nécessaire quand on veut utiliser matplotlib dans un jupyter notebook.
%matplotlib inline

import matplotlib.pyplot as plt
from math import *  #A ne pas mettre à l'élève de compléter
from numpy import *

y = ...
plt.plot(... , ... ),'.')
plt.axhline(0,color="black",lw=.5)
plt.show()
```