Séance 08 :

Booléens

1. Découverte

(a) C.F. Sujet

(b) ***Exemples sans utiliser l'ordinateur :***

i.

```python
def tata(n, p):
    while n and n > 0:
        n = n - p
    return not(n)


val = tata(111, 3)
vap = tata(152, 90)
```

Que fait ce programme ?

Ce programme renvoie `True` si `n` est divisible par `p`, `False` sinon.

Que contiennent `val` ***et*** `vap` ***?***

`val = True`

`vap = False`

ii.

```python
def toto(L):
    while L:
        L.remove(L[-1])
        print(L)


L = [8, 3, 5, 3, 9]
toto(L)
```

Que fait ce programme ?

Ce programme supprime la première occurence du dernier élément de la liste `L` jusqu'à ce qu'elle soit vide, et affiche l'état de la liste après chaque étape :

`[8, 3, 5, 3]`

`[8, 5, 3]`

`[8]`

`[]`

2. Numéro d'une date dans l'année

(a) ***Une année est bissextile si elle est divisible par 400 ou si elle est divisible par 4 mais pas par 100. Ecrire une fonction*** `bissextile` ***qui renvoie un booléen suivant qu'une année*** `a` ***est bissextile ou non.***

```python
def bissextile(a):
    return not(a % 400) or (not(a % 4) and bool(a % 100))
```

(b) ***Donner la liste de toutes les années bissextiles de 1900 à 2020 et vérifier que la somme de ces années vaut 58 860.***

```
def sumYears(begin, end):
    A = [k for k in range(begin, end + 1) if bissextile(k)]
    return (A, sum(A))
```

En exécutant `print(sumYears(1900, 2020))` on obtient :

$$\begin{bmatrix} 1904 & 1924 & 1944 & 1964 & 1984 & 2004 \\ 1908 & 1928 & 1948 & 1968 & 1988 & 2008 \\ 1912 & 1932 & 1952 & 1972 & 1992 & 2012 \\ 1916 & 1936 & 1956 & 1976 & 1996 & 2016 \\ 1920 & 1940 & 1960 & 1980 & 2000 & 2020 \end{bmatrix}$$

`sum(A) = 58 860`

(c) ***On donne les listes des mois courts et des mois longs :***

`ML = [1, 3, 5, 7, 8, 10, 12]`

`MC = [2, 4, 5, 9, 11]`

Ecrire une fonction `jour_annee(date)` ***qui renvoie le numéro de l'année en cours.***

```
def nbOfDaysInMonth(month, isBissextile):
    if (month in MC):
        if month == 2:
            if isBissextile:
                return 29
            return 28
        else:
            return 30
    return 31


def nbOfDays(date):
    S = 0
    for i in range(1, date[1]):
        S += nbOfDaysInMonth(i, bissextile(date[2]))
    return S + date[0]
```

(d) Le jeudi 12 décembre 2019 sera un jour de pleine lune. En regardant un vieux calendrier des postes, Sophie a constaté qu'une pleine lune a eu lieu le 15 janvier 1900. Comme dans sa famille on collection les calendriers des postes depuis cette époque, Sophie a pu dénombrer 1484 pleines lunes dans les 120 calendriers de 1900 à 2019. ***Trouver avec Sophie la révolution synodique de a lune.***

```
def synodique(start, end, lunaisons):
    S = 0
    for a in range(start, end + 1):
        S += 365 + bissextile(a)
    return S / lunaisons
```

En exécutant `print(synodique(1900, 2019, 1484))`, on trouve :

La période synodique de la lune est 29.534 jours.

(e) ***La célèbre nuit du 4 au 5 août 1789 était-elle une nuit de pleine lune ?***

```
def isFullMoonMdr(date):
```

```
    # 15 janvier 1900 est une plein lune
    S = 0
    if date[1] < 1900:
        for a in range(date[2], 1900):
            S += 365 + bissextile(a)
        S = S + 15 - nbOfDays(date)
        return S % synodique(1900, 2019, 1484) < 1
```

De même, en exécutant `print(isFullMoonMdr([4, 8, 1789]))`, on trouve `False` :

La fameuse nuit du 4 au 5 août 1789 n'était pas une nuit de pleine lune.

3. Nombres premiers

(a) ***En utilisant `remove`, donner la liste des 168 nombres premiers de 1 à 1 000 en utilisant le principe du crible d'Eratosthène. Faire moins de 1 500 tours de boucle.***

```
def prime(n):
    """Returns the list of the n first n prime numbers,
    and and the number of loop turns"""

    count = 0  # loop counter

    nb = [k for k in range(2, n+1)]
    for i in range(2, n+1):
        for el in nb:
            if el % i == 0 and el != i:
                count += 1  # On compte combien de tours de boucle on fait
                nb.remove(el)
    return (count, nb)
```

Résultats :

```
== == == == = Test pour n = 1000 == == == == =
Le programme fait 831 tours(max: 1500 tours).
Temps d execution: 0.008s
Nombre de resultats: 168
Resultats: [2, 3, 5, 7, '...', 991, 997]

== == == == = Test pour n = 10000 == == == == =
Le programme fait 8770 tours(max: 20000 tours).
Temps d execution: 0.595s
Nombre de resultats: 1229
Resultats: [2, 3, 5, 7, '...', 9967, 9973]
```

4. Formule d'Euler

On donne une formule d'Euler :

$$\frac{\pi^2}{9} = \frac{5^2}{(5^2-1)}\frac{7^2}{(7^2-1)}\frac{11^2}{(11^2-1)}\frac{13^2}{(13^2-1)}\frac{17^2}{(17^2-1)}\frac{19^2}{(19^2-1)}\cdots$$

En continuant ainsi avec la liste des nombres premiers à partir de 5.

(a) ***A quel nombre premier k faut-il s'arrêter dans cette formule pour avoir $\frac{\pi^2}{9}$ à 10^{-4} près ?***

- J'espère qu'il faut $k \leq 1229$. Ainsi je peux utiliser la liste précédente, et boucler en faisant augmenter k à chaque fois.
  Je créer donc une première fonction qui ressemble beaucoup à `prime` qui renvoie la liste des nombres premiers inférieurs à `n` :

```python
def listeNombresPremiers(n):
    nb = [k for k in range(2, n+1)]
    for i in range(2, n+1):
        for el in nb:
            if el % i == 0 and el != i:
                nb.remove(el)
    return nb
```

❏ Ensuite je crée la fonction qui renvoie le terme général de la formule d'Euler ci-dessus :

```python
def Euler(n):
    return (n**2)/(n**2-1)
```

❏ Enfin je calcule chaque terme du produit en allant un rang plus loin à chaque fois, puis je compare les rangs k et $k-1$. Si la différence est inférieure à 10^{-4}, on est bon :

```python
def picarre(pres):
    k, eps, P, oP, oeps = 0, 1, 1, 1, 1

    E = listeNombresPremiers(10000)

    while (eps > 10 ** (-pres)):
        oP = P
        P *= Euler(E[k])
        oeps = eps
        eps = abs(oP - P)
        k += 1

    return (k - 1, oeps)
```

Remarque : je renvoie $k-1$ et non k car pour $k=32$, on est à 10^{-5} et avec k=31 on est tout juste au dessus de 10^{-4}

❏ J'exécute `print(picarre(4))` Avec k = 31, on trouve une précision `eps = 0.00010185214032998324` soit une précision à 10^{-4} près. D'où :

$$\boxed{k = 31}$$

(b) ***Idem pour avoir $\frac{\pi^2}{9}$ à 10^{-5} près ?***

J'exécute `print(picarre(5))` et trouve, pour $k=79$, une précision `eps = 1.0226025770831981e-05` soit une précision à 10^{-5} près. Donc :

$$\boxed{\text{Pour avoir } \frac{\pi^2}{9} \text{ à } 10^{-5} \text{ près, il faut } k = 79.}$$

(c) ***En prenant 13 valeurs différentes d'erreurs égales à $\frac{1}{M}$ où M est équirépartie entre 50 et 10^5, vérifier que dans la formule d'Euler, l'erreur est de l'ordre de $\frac{1}{kln(k)}$ où k est le dernier nombre premier utilisé dans le produit.***

❏ On commence par importer les librairies, et aussi par calculer la liste des 1 229 premiers nombres premiers, ça sera fait :

```python
import numpy as np
import matplotlib.pylab as plt

E = listeNombresPremiers(10000)
```

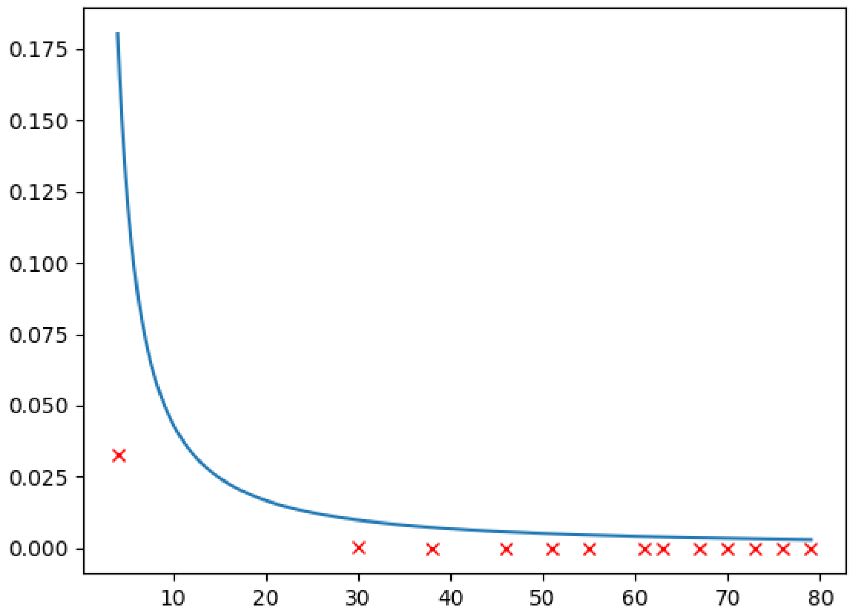
❏ Bien-sûr, il faut un peu modifier la fonction `picarre` précédente :

```python
def picarre(pres):
    k, eps, P, oP, oeps = 0, 1, 1, 1, 1

    while (eps > pres):
        oP = P
        P *= Euler(E[k])
        oeps = eps
        eps = abs(oP - P)
        k += 1

    return (k-1, oeps)
```

❏ Enfin, on code ce qui est demandé :

```python
def pres():
    # On definit la liste de precisions
    M = np.linspace(50, 10**5, 13)

    K, Eps = [], []

    # On recupere tous les k et eps pour ces 13 precisions differentes
    for m in M:
        k, eps = picarre(1/m)
        K.append(k)
        Eps.append(eps)

    # On calcule la fonction 1/kln(k)
    kMaxi = max(K)
    kMini = min(K)
    X = np.linspace(kMini, kMaxi, 1000)
    Y = 1/(X * np.log(X))

    # On trace tout ca
    plt.plot(X, Y)
    plt.plot(K, Eps, 'rx')
    plt.show()
```

❏ Et on obtient ça :



Mais ce n'est pas très concluant... J'ai essayé de retourner k et non $k-1$ dans la fonction `picarre`, mais ça ne change pas grand chose. . .