ALGO
PYTHON

Durée : 4h

- 1 - Au hasard, combien de tirages ?

Objectifs :
- Créer un tableau de 24 nombres aléatoires tous différents compris en 1 et 24.
- Se rendre compte du temps que ça prend.

Le hasard en Python :

```
from random import *
randrange(0,100)
```
→ entre 0 (inclu) et 100 (exclu)

Première version : vous avez le droit d'utiliser l'opéateur « in » de Python :

```
if value in array:
    doSomething
```

Deuxième version : vous n'avez pas le droit d'utiliser l'opérateur « in ». Faites de l'algo

Mesures :

Dans une des deux versions, comptez le nombre de fois où un tirage de nombre aléatoire a été effectué. ça fait beaucoup non ?

Dans chaque version, changez le nombre de valeurs (10000 valeurs entre 1 et 10000). Mesurez le temps de chaque version.
Augmentez les valeurs indépendamment. Il y a un risque de bug. Lequel ? Comment l'éviter ?

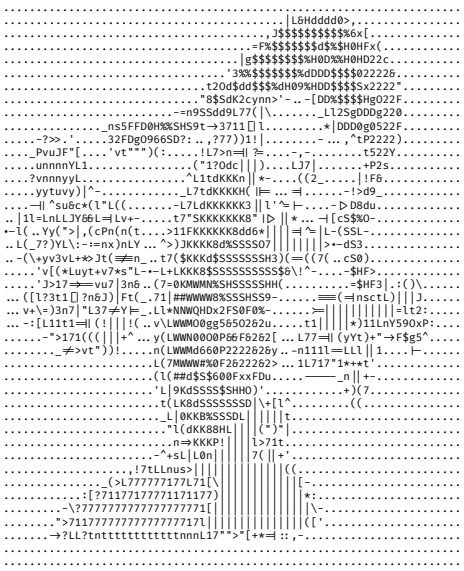
```
import time

before  = time.time()
```

Votre code ici !

```
after  = time.time()
print(after-before)
```

- 2 - Pixel Art

Objectif : convertir une image en ASCII art



Suggestion de déroulement :

1. Voyez comment l'image est structurée
2. Parcourez la (double boucle)
3. Pour chaque bloc de pixels, calculez la moyenne de luminosité à partir des composantes RGB
4. Trouvez le caractère qui convient (cf liste ci-dessous)
5. Gérez les retours à la ligne

Testez sur différentes images (pas trop grosses)
Testez avec différentes tailles de blocs de pixels
Visualisez les fichiers générés avec une police à chasse fixe (caractères tous de même taille)

Généralisez → Fonction avec nom de fichier, taille des blocs, ...
Pour aller plus loin : générez de l'HTML avec des couleurs, du CSS inline… bien lourd

Pour charger une image sous forme de tableau :

```
from PIL import Image
import numpy as np

image = np.asarray(Image.open(inFilename), dtype='int32')
```

Faites un « **print** » pour voir à quoi ça ressemble

Pour enregistrer un fichier texte :

```
f = open(outFilename, "w")
a = f.write(myString)
f.close()
```

Un peu de données utiles : Le nombre de pixels par caractère. Chaque tuple contient le code ASCII et le nombre de pixels (densité du caractères) :

```
pixelsPerChar  =  [(32,0),(33,37),(34,51),(35,98),(36,87),(37,88),(38,71),(39,25),(40,42),
(41,48),(42,44),(43,46),(44,26),(45,24),(46,15),(47,44),(48,80),(49,49),(50,72),(51,69),
(52,71),(53,74),(54,84),(55,50),(56,93),(57,83),(58,30),(59,44),(60,50),(61,39),(62,52),
(63,56),(64,98),(65,83),(66,92),(67,71),(68,78),(69,87),(70,77),(71,83),(72,85),(73,52),
(74,62),(75,94),(76,55),(77,105),(78,100),(79,82),(80,73),(81,102),(82,88),(83,86),
(84,62),(85,82),(86,74),(87,109),(88,88),(89,67),(90,74),(91,47),(92,43),(93,47),(94,32),
(95,28),(96,15),(97,78),(98,88),(99,66),(100,91),(101,82),(102,62),(103,81),(104,72),
(105,42),(106,51),(107,77),(108,45),(109,87),(110,60),(111,67),(112,80),(113,85),
(114,50),(115,70),(116,58),(117,64),(118,61),(119,78),(120,75),(121,65),(122,61),
(123,49),(124,36),(125,48),(126,28),(127,0),(128,0),(129,0),(130,0),(131,0),(132,0),
(133,0),(134,0),(135,0),(136,0),(137,0),(138,0),(139,0),(140,0),(141,0),(142,0),(143,0),
(144,0),(145,0),(146,0),(147,0),(148,0),(149,0),(150,0),(151,0),(152,0),(153,0),(154,0),
(155,0),(156,0),(157,0),(158,0),(159,0),(160,0),(161,0),(162,0),(163,0),(164,0),(165,0),
(166,0),(167,0),(168,0),(169,0),(170,0),(171,0),(172,0),(173,0),(174,0),(175,0),(176,0),
(177,0),(178,0),(179,0),(180,0),(181,0),(182,0),(183,0),(184,0),(185,0),(186,0),(187,0),
(188,0),(189,0),(190,0),(191,0),(192,0),(193,0),(194,0),(195,0),(196,0),(197,0),(198,0),
(199,0),(200,0),(201,0),(202,0),(203,0),(204,0),(205,0),(206,0),(207,0),(208,0),(209,0),
(210,0),(211,0),(212,0),(213,0),(214,0),(215,0),(216,0),(217,0),(218,0),(219,0),(220,0),
(221,0),(222,0),(223,0),(224,0),(225,0),(226,0),(227,0),(228,0),(229,0),(230,0),(231,0),
(232,0),(233,0),(234,0),(235,0),(236,0),(237,0),(238,0),(239,0),(240,0),(241,0),(242,0),
(243,0),(244,0),(245,0),(246,0),(247,0),(248,0),(249,0),(250,0),(251,0),(252,0),(253,0),
(254,0)]
pixelsPerChar.sort(key=lambda y: y[1])
maxPixelsPerChar = 109
```