1. Utilisation du module Pandas

Le module `csv` utilisé précédemment se contente de lire les données structurées. Il ne fait aucun effort particulier pour analyser les données. Nous nous en sommes aperçus lorsqu'il a fallu convertir par `int()` toutes les valeurs numériques, qui étaient interprétées comme des chaînes de caractères.
La bibliothèque `pandas` est par contre spécialement conçue pour l'analyse des données (*data analysis*) : elle est donc naturellement bien plus performante.

In [ ]:
```
import pandas as pd #import du module pandas, abrégé classiquement par "pd"
```

In [ ]:
```
df = pd.read_csv('http://glassus1.free.fr/top14.csv', encoding = 'utf-8')
```

La variable est nommée classiquement `df` pour *dataframe* (que l'on peut traduire par *table de données*)

In [ ]:
```
type(df)
```

1.1 Premiers renseignements sur les fichiers de données

Que contient la variable `df` ?

In [ ]:
```
df
```

Les données sont présentées dans l'ordre originel du fichier. Il est possible d'avoir uniquement les premières lignes du fichier avec la commande `head()` et les dernières du fichier avec la commande `tail()`. Ces commandes peuvent recevoir en paramètre un nombre entier.

```
In [ ]: df.head()
```

```
In [ ]: df.tail()
```

```
In [ ]: df.head(3)
```

Pour avoir des renseignements globaux sur la structure de notre fichier, on peut utiliser la commande `df.info()`

```
In [ ]: df.info()
```

Pour accéder à une fiche particulière de joueur, on peut utiliser la fonction `loc()` :

```
In [ ]: df.loc[45]
```

1.2 Extraction de colonnes, création de graphiques

Pour créer une liste contenant uniquement les données numériques de la colonne poids, il suffit d'écrire :

```
In [ ]: poids = df['Poids']
```

Attention, la variable `poids` n'est pas une liste qui contiendrait `[122,116,112,...]` mais un type particulier à `pandas`, appelé "Series".

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

In [ ]:
```
type(poids)
```

On peut néanmoins s'en servir comme d'une liste classique.

In [ ]:
```
poids[0]
```

On voit donc que les données sont automatiquement traitées comme des nombres. Pas besoin de conversion comme avec le module `csv` !

Pour tracer notre nuage de points poids-taille, le code sera donc simplement :

In [ ]:
```
%matplotlib inline
import matplotlib.pyplot as plt
X = df['Poids']
Y = df['Taille']

plt.plot(X,Y,'ro') # r pour red, o pour un cercle. voir https://matplotlib.org/api/markers_api.html
plt.show()
```

L'interprétation numérique permet à `pandas` d'analyser automatiquement les données, avec notamment la fonction `describe()`.

In [ ]:
```
df['Taille'].describe()
```

On voit donc que les indicateurs statistiques sont proposés automatiquent. D'ailleurs, on peut très facilement tracer des boites à moustaches avec `boxplot()`.

In [ ]:
```
df.boxplot("Taille")
```

Pour les données non-numériques, la commande `describe()` n'est que peu d'utilité. Elle renseigne toutefois la valeur la plus fréquente (en statistiques, le *mode* ou *valeur modale*)

```
In [ ]: df['Poste'].describe().top
```

```
In [ ]:
```

Pour connaître par exemple la date de naissance la plus fréquente chez les joueurs du top14, on utilisera simplement :

```
In [ ]: df['Date de naissance'].describe().top
```

Qui sont les joueurs nés à cette date ?

```
In [ ]: print(df['Nom'][df['Date de naissance'] == '23/04/1993'])
```

Beaucoup plus de renseignements sont donnés par la commande `value_counts()`.

```
In [ ]: df['Taille'].value_counts()
```

1.3 Filtres et recherches

Comment créer une *dataframe* ne contenant que les joueurs de l'UBB ?

L'idée syntaxique est d'écrire à l'intérieur de `df[]` le test qui permettra le filtrage.

```
In [ ]: UBB = df[df['Equipe'] == 'Bordeaux']
```

```
In [ ]: UBB
```

1.3.1 Exercice 1

Créer une dataframe `gros` qui contient les joueurs de plus de 135 kg.

In [ ]:
```
gros = df[df['Poids'] > 135] ; gros
```

1.3.2 Exercice 2

Créer une dataframe `grand_gros` qui contient les joueurs de plus de 2m et plus de 120 kg.

In [ ]:
```
grand_gros = df[(df['Poids'] > 120) & (df['Taille'] > 200
grand_gros
```

1.3.3 Exercice 3

Trouver en une seule ligne le joueur le plus léger du Top14.

In [ ]:
```
df['Nom'][df['Poids'] == min(df['Poids'])]
print(df['Nom'][df['Poids'].idxmin])
```

1.4 Tris de données

Le tri se fait par la fonction `sort_values()` :

In [ ]:
```
newdf = df.sort_values(by=['Poids'], ascending = True)
```

In [ ]:
```
newdf.head(10)
```

In [ ]:
```

```

1.5 Rajout d'une colonne

Afin de pouvoir trier les joueurs suivant de nouveaux critères, nous allons rajouter un champ pour chaque joueur. Prenons un exemple stupide : fabriquons un nouveau champ `'Poids après les vacances'` qui contiendra le poids des joueurs augmenté de 8 kg. Ceci se fera simplement par :

In [ ]:
```
df['Poids après les vacances'] = df['Poids'] + 8
```

In [ ]:
```
df.head()
```

Pour supprimer cette colonne sans intérêt, faisons :

In [ ]:
```
del df['Poids après les vacances']
```

In [ ]:
```
df.head()
```

1.5.4 Exercice 4

1. Créer une colonne contenant l'IMC de chaque joueur
2. Créer une nouvelle dataframe contenant tous les joueurs du top14 classés par ordre d'IMC croissant.

In [ ]:
```
df['IMC'] = df['Poids'] / (df['Taille']/100)**2
df.head()
```

In [ ]:
```
imcdf = df.sort_values(by=['IMC'], ascending = True)
imcdf
```

In [ ]:
```

```