```
#---------------------------------------------------------------'
#Installer :
# - Anaconda 2.3.0
# - Jet Brains PyCharm Community Edition 4.0.3
# - Pyodbc
#   o https://code.google.com/archive/p/pyodbc/downloads
#   o 3.0.7 32-bit Windows Installer for Python 2.7
#--------------------------------------------------------------


####################################
### Fonction dans Cobra ###
### Moocs Youtube Data School ###
####################################

# 1. Import
import pandas as pd

# 2. Read Table
    # csv
    orders = pd.read_table('data/chip.csv')
    # url
    orders = pd.read_table('http://bit.ly/chiporders')
    # Separateur personnalisé
    orders = pd.read_table('http://bit.ly/movieusers'
        , sep = '|')
    # with No header
    orders = pd.read_table('http://bit.ly/movieusers', sep = '|'
        , header = None)
    # Définir le nom des colonnes
    user_cols = ['user_Id', 'age', 'gender', 'occupation',
        'zip_code']
    orders = pd.read_table('http://bit.ly/movieusers', sep =
        '|', header = None
        , names = user_cols)
    # Remplacer le nom des colonnes
    orders = pd.read_table('http://bit.ly/movieusers'
        , header = 0, names = user_cols)


# 3. read_csv
    # Use the ',' as default sep
    orders = pd.read_table('http://bit.ly/movieusers', sep =
        ',')
    # ou
    orders = pd.read_csv('http://bit.ly/movieusers')

    # COBRA
```

```
        extract = pd.read_csv(filePath, header=0)


# 4. dataFrame
    # Les objets issues de read_csv ou read_table sont des
        dataframe
    type (orders)                  --> pandas.core.frame.dataFrame

    # Read dataFrame
    orders ['City']
    orders.City                    # Same (dont work if there is
        a space)

    # Type = Series
    type(orders ['City'])          --> pandas.core.Series.Series

    # Creation de colonnes / Series
    orders ['Location'] = orders ['City'] + ', ' + orders
        ['State']

    # Decrire un dataFrame
    oders.shape     --> (960, 6)    #rows, columns
    orders.describe
        --> Count
            mean
            std
            min
            25%
            ...
            max
    orders.dtypes       --> type des colonnes

    # column name
        orders.columns
            --> Index ([u'City', u'Colors Reported'...], dtype
                = 'object')

        orders.rename(columns = {'Colors
            Reported':'Colors_Reported'}, inplace = True)
            #Dico with key = old name and value = new name
            #Inplace = affect the dataFrame
        # or
        orders.columns = ['col1', 'col2', 'col3'...]
    # plus rapide
    orders.columns = orders.columns.str.replace(' ','_')


    # drop
    ufo.drop('Colors reported', inplace=True, axis = 1)
    # Drop rows (the 2 first)
    ufo.drop([0, 1], inplace=True, axis = 0)
        # axis = 0 --> row
        # axis = 1 --> column
        #Inplace = affect the dataFrame (update l'existante)

        # COBRA
        mat.drop(['IDX CLOSE', 'col 2', 'col 3'],
            inplace=True, axis=1)

    # Sort
    movies['title'].sort_values(ascending = False)
        --> Series, ne change pas le dataFrame sous-jacent
    movies.sort_values('title', ascending = True)
        --> Sort le dataFrame sous-jacent
    movies.sort_values(['title', 'duration'], ascending = True)
        --> sort sur plusieurs critères

        # COBRA (old version)
        mat = mat.sort(['DATE'], ascending=[1])


    # Filter
    movies[movies.duration >= 200]
        # COBRA -- WHERE
        mat['INDEX CLOSE'] = mat['INDEX
            CLOSE'].where(mat['INDEX CLOSE'].notnull(),
            mat['IDX CLOSE'])

    # loc / iloc
    movies.loc[movies.duration >= 200, 'genre']
        # Trier sur les lignes en duration et sélectionner
            uniquement la colonne Genre

        # COBRA
        mat_synth = pd.DataFrame()
        mat_synth.loc[0, 'NAME'] = mat.head(1)['NAME'].iloc[0]
        mat_synth.loc[0, 'ISIN'] = mat.head(1)['ISIN'].iloc[0]





# 5.Series
    # Convertir une liste en Series
        ser_bl_duration = pd.Series(list_Bt_duration)
```

```
# Créer une liste de Bool à partir d'une Series
    list_Bt_duration = movies.duration >= 200




# COBRA
--cnxn = pyodbc.connect('DRIVER={SQL Server};SERVER=' +
    server + ';DATABASE=' + db1)
--cursor = cnxn.cursor()
--cursor.execute("SELECT TOP 1 str_IsinLongName "
--               "FROM V_Ref_FundIsinPListing "
--               "WHERE str_Isin = '" + isin + "' ")
extract = pd.DataFrame.from_records(cursor.fetchall(),
    columns=['NAME'])
name = str(extract.head(1)['NAME'].iloc[0])




# boucle sur dataframe
for i, d_param in all_etf.iterrows():




# shift
pricesEtf['ETF RETURN'] = pricesEtf['ETF ADJ'] /
    pricesEtf['ETF ADJ'].shift(1) - 1

# merge
mat = pd.merge(mat, pricesIdx, how='left', on=['DATE',
    'INDEX'])



# ExcelWriter
writer = pd.ExcelWriter(str.replace(fileFolder + '\\' +
    fileName, '\\\\', '\\'), datetime_format='yyyy-mm-dd')
wb = writer.book
wb.formats[0].font_size = 9
mat_details.to_excel(writer, sheet_name='Details',
    index=False, startrow=2, startcol=0)
mat_synth.to_excel(writer, sheet_name='Synthesis',
    index=False, startrow=2, startcol=0)
ws_details = writer.sheets['Details']
ws_synth = writer.sheets['Synthesis']
```

```
# NUMPY
    import numpy as np
    # isnan
        np.isnan
        # Je l'utilise pour savoir si une valeur dans un
            dataFrame is null
        (", NULL" if np.isnan(d_param['ETF PERF']) else ", " +
            str(d_param['ETF PERF']) + " ") +





#-----------------------------------------------------------
# Exo & Fonction classique
#-----------------------------------------------------------

# Volatilite
def Vol (Price):
    tab = pd.dataFrame()
    tab = price
    tab['return'] = tab['price'] / tab['price'].shift(1) -1
    return tab['return'].std(ddof = 1) * math.sqrt(252)
```