In [ ]:
```
##########################################################
# Pandas Series
# Author : Rodrique KAFANDO
# Destination : Master FD&IA - UV-BF
# 12.07.2021
##########################################################
```

Series

- first object of pandas
- one dimensional data structure or a single column of data
- data type desn't matter in a pandas series object
- besoin d'une et une seule reference pour accéder à une donnée
- https://www.learndatasci.com/tutorials/python-pandas-tutorial-complete-introduction-for-beginners/

In [ ]:
```
# import pandas librairy
import pandas as pd
```

Create a series object from a List

In [ ]:
```
student_list = ['kabore', 'ilboudo', 'kafando', 'robgo']
```

In [ ]:
```
# default index, data, dtype
pd.Series(student_list)
```

Create a series object from a Dictionary

In [ ]:
```
# restaurant menu
menu_dic = {'Bissap': 300, 'atieke':600, 'Gonré': 6000, 'haricot':
7000}
```

In [ ]:
```
pd.Series(menu_dic)
```

In [ ]:
```
#INTRODUCE map() method
```

In [ ]:
```

```

Attributes on Series Object

In [ ]:
```
student_list = ['kabore', 'ilboudo', 'kafando', 'robgo']
```

```
pd.Series(student_list)
```

In [ ]:
```
# we need to store series object in order to acces its attributes
data = pd.Series(student_list)
```

In [ ]:
```
# attributes are directly called uppon series without (), that
make the difference with methods
data.values
```

In [ ]:
```

```

Methods on Series Object

- manipulating, adding, removing, etc.
- execute a kind of operation, while attribute does nothing

In [ ]:
```
# example
prices = [300, 500, 799.99]
p_series = pd.Series(prices)
p_series
```

In [ ]:
```
p_series.apply(lambda x: str(x))
```

In [ ]:
```
p_series.sum()
```

In [ ]:
```
p_series.product()
```

In [ ]:
```
p_series.mean()
```

Parameters and arguments

- methods has parameters
- parameters take values called arguments

In [ ]:
```
# example: level and sound are parameters and the different values
they can be assign are arguments
# level => difficult, medium, easy
# sound volume => 1 to 5
```

In [ ]:
```
student_list = ['kabore', 'ilboudo', 'kafando', 'robgo']
diner_list =['rice', 'fish', 'pizza', 'salmon']
pd.Series(diner_list, student_list) # press shit+tab to see
```

```
arguments
pd.Series(data = diner_list, index = student_list) # press
shit+tab to see arguments
```

In [ ]:
```
# pandas series with dictionary differenc ?
# we can have duplicated index
student_list = ['kabore', 'ilboudo', 'kafando', 'robgo', 'kafand']
diner_list =['rice', 'fish', 'pizza', 'salmon', 'fruits']
pd.Series(diner_list, student_list)
```

Create series from dataset with read_csv method

In [ ]:
```
# use squeeze to have a series object instead dataframe
pokemon= pd.read_csv('./pandas/pokemon.csv',  usecols =
["Pokemon"], squeeze = True)
pokemon
```

In [ ]:
```
google_stock_price =
pd.read_csv('./pandas/google_stock_price.csv')
```

In [ ]:
```

```

The .head() and .tail() methods

In [ ]:
```
google_stock_price =
pd.read_csv('./pandas/google_stock_price.csv', squeeze = True)
google_stock_price
```

In [ ]:
```
pokemon = pd.read_csv('./pandas/pokemon.csv', usecols =
["Pokemon"], squeeze = True)
pokemon
```

In [ ]:
```
# just to see what data look like
pokemon.head() # 5 first rows default from the top
```

In [ ]:
```
pokemon.tail(10) # last 5 rows from the bottom
```

Python Built-in Function

In [ ]:
```
# sorted() function
len(pokemon) # number of rows
sorted(pokemon), list(pokemon)
```

In [ ]:
```
# type()
type(pokemon)
```

In [ ]:
```
# and more build-in function
dict(pokemon)
max(google_stock_price)
min(google_stock_price)
```

More series attributes

In [ ]:
```
pokemon.values
pokemon.index
```

In [ ]:
```
pokemon.ndim
pokemon.shape
```

The .sort_values () method

In [ ]:
```
# chaining method
pokemon.sort_values().head()
```

In [ ]:
```
# from top to buttom
pokemon.sort_values(ascending = False)
```

In [ ]:
```
# numeric value
google_stock_price.sort_values(ascending = False)
google_stock_price.head(3)
```

The inplace parameter

In [ ]:
```
google_sv = google_stock_price.sort_values()
google_sv.head(3)
```

In [ ]:
```
# use inplace parameter, to overwrite stored series
google_sv.sort_values(ascending = False, inplace = True)
```

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

The .sort_index() method

In [ ]:
```
pokemon = pd.read_csv('./pandas/pokemon.csv', usecols =
```

```
["Pokemon"], squeeze = True)
```

In [ ]:
```
pokemon.sort_index()
```

In [ ]:
```

```

In [ ]:
```

```

The 'in' keyword's

- check by default on series index and not values

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

In [ ]:
```
'Bulbasaur' in pokemon
```

In [ ]:
```
4 in pokemon
```

In [ ]:
```
4 in pokemon.index
```

In [ ]:
```
'Bulbasaur' in pokemon.values
```

Extract values by index position

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

In [ ]:
```
# do it like on python list
pokemon[2]
```

In [ ]:
```
pokemon[[2, 0, 1, 3]]
```

In [ ]:
```
list(pokemon[[2, 0, 1, 3]])
```

In [ ]:
```
pokemon[90:100]
```

Extract values by index label

In [ ]:
```
# load pokemon dataset
pokemon= pd.read_csv('./pandas/pokemon.csv')
pokemon.head()
```

In [ ]:
```

```

```
# change default index
# load pokemon dataset
pokemon= pd.read_csv('./pandas/pokemon.csv', index_col =
'Pokemon')
pokemon.head()
```

In [ ]:
```
# change default index,
# load pokemon dataset and use the name as index
pokemon= pd.read_csv('./pandas/pokemon.csv', index_col =
'Pokemon', squeeze = True)
pokemon.head(3)
```

In [ ]:
```
# extract by index position
pokemon[[2,4]]
```

In [ ]:
```
# by index label
pokemon['Bulbasaur']
pokemon['Charmeleon']
```

In [ ]:
```
# à la difference des index, la dernière valeure sera incluse dans
la sortie de la requete
pokemon['Charmeleon':'Rattata']
```

In [ ]:
```
# with a step value
pokemon['Charmeleon':'Rattata':3]
```

In [ ]:
```
# introduce keyword error...
pokemon['error']
```

In [ ]:
```
# introduce keyword error...
pokemon[['Charmeleon','error']]
```

In [ ]:
```
# introduce keyword error..., solve it by reindex method
pokemon.reindex(index = ['Charmeleon','error'])
```

In [ ]:
```
pokemon['error'] = 'test'
```

In [ ]:
```
# assign and check
pokemon[['Charmeleon','error']]
```

In [ ]:
```
pokemon['error']
```

In [ ]:
```
```

The .get() method on series

- en cas de valeur inexistante, .get() ne renvoit pas de message d'error si default = None

In [ ]:
```
# let's use our pokemon data
pokemon.get(2)
```

In [ ]:
```
# inexistant label with get()
pokemon.get('python')
```

In [ ]:
```
# use defaut parameter to return a message
pokemon.get('python', default = 'value does not exist')
```

In [ ]:
```
# use defaut parameter to return a message
pokemon.get(['python','Charmeleon'], default = 'value does not
exist')
```

Math method on series Object

In [ ]:
```
google_stock_price =
pd.read_csv('./pandas/google_stock_price.csv', squeeze = True)
```

In [ ]:
```
# do not include missing value
pokemon.count()
```

In [ ]:
```
len(pokemon)
```

In [ ]:
```
# show truth
pokemon.reindex(index = ['Charmeleon','Nanvalue'])
```

In [ ]:
```
pokemon.count()
```

In [ ]:
```
len(pokemon)
```

In [ ]:
```
pokemon.reindex(index = ['mynan'])
```

In [ ]:
```
import numpy
pokemon['mynan'] = numpy.nan
```

```
pokemon['mynan2'] = numpy.nan
pokemon['mynan3'] = numpy.nan
```

In [ ]:
```
pokemon.count()
```

In [ ]:
```
len(pokemon)
```

In [ ]:
```
# sum() method
google_stock_price.sum()
```

In [ ]:
```
google_stock_price.mean()
```

In [ ]:
```
google_stock_price.std()
google_stock_price.min()
google_stock_price.max()
```

In [ ]:
```
google_stock_price.describe()
```

The .idexmax() and idxmin() methods on series

In [ ]:
```
google_stock_price.min()
```

In [ ]:
```
google_stock_price.max()
```

In [ ]:
```
google_stock_price.idxmin()
# check
```

In [ ]:
```
google_stock_price[google_stock_price.idxmin()]
```

In [ ]:
```

```

In [ ]:
```
google_stock_price.idxmax()
```

The .value_counts() method on series

- obtenir la somme de chaque valeur unique dans une series

In [ ]:
```
pokemon.head(5)
```

In [ ]:
```
pokemon.value_counts()
pokemon.value_counts(ascending = True)
```

In [ ]:
```
pokemon.value_counts().sum()
```

In [ ]:
```
pokemon.count()
```

THe .apply() method on series

- permet d'utiliser une function specific sue chaque valeur unique de la serie

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

In [ ]:
```
# let's apply a classification method on the dataset
def classif_val(number):
    if number < 200:
        return 'OK'
    elif number >= 200 and number < 700:
        return 'Perfect'
    else:
        return 'Incredible'
```

In [ ]:
```
google_stock_price.apply(classif_val).tail(5)
```

In [ ]:
```
# use lamda function
google_stock_price.apply(lambda price : classif_val(price))
## use simple operator
google_stock_price.apply(lambda price : price +10)
```

The .map() method on series

In [ ]:
```
## let's use pokemon data
pokemon_names = pd.read_csv('./pandas/pokemon.csv', usecols =
['Pokemon'], squeeze = True)
pokemon.head(5)
```

In [ ]:
```
pokemon_types = pd.read_csv('./pandas/pokemon.csv', index_col =
'Pokemon', squeeze = True)
pokemon_types.head(5)
```

In [ ]:
```
# use map
pokemon_names.map(pokemon_types).head(3)
```

In [ ]:
```
pokemon_types_d = pd.read_csv('./pandas/pokemon.csv', index_col =
'Pokemon', squeeze = True).to_dict()
```

In [ ]:
```
# with two differents data typ (series and dictionary)
pokemon_names.map(pokemon_types_d).head(3)
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: