Gestion de Projet Big Data & Développement d'applications Big Data

EDAH Kodjo
Consultant Systèmes d’Information, Big-Data

Objectifs

- Comprendre la notion et les spécificités du Big Data
- Connaître les technologies de l’écosystème Hadoop
- Connaître le langage python et utiliser les librairies de machine learning
- Savoir utiliser les outils de visualisation des données (Dataviz)

Partie 3 : Python

Les bases de python

- Présentation de Python
- Les types et les opérations de base
- Les structures de contrôle
- Les fonctions
- Les fichiers
- Les classes
- Les exceptions
- Les modules

Présentation de Python

- Développé en 1989 par Guido van Rossum
- Open source
- Portable (Windows, linux Mac OS)
- Orienté objet
- Dynamique
- Extensible
- Support pour l'intégration d'autre langage

Présentation de Python

- Langage de programmation généraliste, interprété et open source
- Version 2.7 (2012) et 3.8 (2019)
- Multitudes de frameworks et packages:
  - Web: Django, flask
  - Calcul scientifique: Numpy, Scipy
  - Analyse de données et machine learning: pandas, scikit-learn
- Langage pour application Data/Analytics par excellence

Outils

- Distribution Anaconda
  - 100+ Packages inclus
  - Gestion d'environnement virtuel (reproduction)

- IPython (Shell) et Projet Jupyter
  - Exécution interactive
  - Visualisation intégrée
  - Support de Markdown
  - Sauvegarde de l'état à travers un Kernel
  - Exportation en HTML

Affichage : la fonction print

- La fonction print()

```
print("Hello world!")
```

```
Hello world!
```

- Écriture formatée

```
x = 32
nom = "John"
print("{} a {} ans".format(nom, x))
```

```
John a 32 ans
```

Variables

- Commence avec A-Z a-z ou _
- Autres caractères lettres, chiffres, ou _
- Sensible à la casse
- Pas de longueur précise (raisonnablement)
- Pas les mots réservés au langage
- Assignations :

  <nom_variable> = <expression>

- Convention de nommage : **snake_case**

```
age = 10

print(age)
```

```
10
```

```
age, nom_etudiant, prenom_etudiant  = 10, "Hubert" , "Charles"

print(age, nom_etudiant, prenom_etudiant)
```

```
10 Hubert Charles
```

Structure de données

- Toutes les valeurs en Python sont des objets
- Nombres : int, long, float, complex
- Booléan : bool
- String

```
age   = 10
string_var = "Mon texte"
print(type(age))
print(type(string_var))
```

```
<class 'int'>
<class 'str'>
```

Les opérations

- Arithmétique : +, -, *, /, %, **, //
- Comparaison : ==, !=, >, <, >=, <=
- Logique : and, or, not
- Assignations: +=, -=, /=, *=, %=, **=, //=
- Bitwise: &, |, ^, ~,>>, <<

```
age  = 10
ma_chaine= "Iris school"
print(age >= 5 + 5 )
print(len(ma_chaine) > 25 / 2 )
```

```
True
False
```

Structure de contrôle - if else

```
if <expr>:
    <statement(s)>
elif <expr>:
    <statement(s)>
elif <expr>:
    <statement(s)>
    ...
else:
    <statement(s)>
```

```
nom_etudiant = 'John'
if (nom_etudiant == 'Fred'):
  print('Hello Fred')
elif (nom_etudiant == 'Xander'):
  print('Hello Xander')
else :
  print('Qui es-tu ?')
```

- Attention à indentation

Structure de contrôle boucle - for

- C’est une itération sur une séquence d'élément: iterator
- on peut utiliser break pour interrompre
- on peut utiliser continue pour ignorer la suite de la boucle

```
nom_etudiant = "Fredy"
for lettre in nom_etudiant:
  print(lettre)
```

```
F
r
e
d
y
```

```
nom_etudiant = "Fredy"
for lettre in nom_etudiant:
  print(lettre)
  if(lettre == "e"):
    break
```

```
F
r
e
```

```
nom_etudiant = "Fredy"
for lettre in nom_etudiant:
  if(lettre == "e"):
    continue
  print(lettre)
```

```
F
r
d
y
```

Structures de contrôle - while

```
while <expr> :
    <statement(s)>
```

```
i = 0
while i < 3:
    print("i =" , i)
    i = i +1
```

```
i = 0
i = 1
i = 2
```

Listes

- Une liste est une structure de données qui contient une série de valeurs
- Python autorise la construction de liste contenant des valeurs de types différents Accès par indice
- Concaténation simple avec “+”
- **append(x), remove(x), insert(i,x)**
- Les fonctions range() et list()

```
list(range(10))
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
liste_present=["Charles","Carine"]
liste_absent=["Kodjo","Victor", "Charles"]
liste_total=liste_present + liste_absent
print(liste_total)
print(liste_total[3])
```

```
['Charles', 'Carine', 'Kodjo', 'Victor', 'Charles']
Victor
```

```
liste_poly=["Charles","Carine",12,True]
print(liste_poly[3])
print(liste_poly[5])
```

```
True
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-30-2d233b6f98a0> in <module>()
      1 liste_poly=["Charles","Carine",12,True]
      2 print(liste_poly[3])
----> 3 print(liste_poly[5])

IndexError: list index out of range
```

Les tuples

- Un tuple est une structure de données qui contient une série de valeurs et qui reste immutable après la création
- Les tuples peuvent contenir différents types de valeurs
- Comme les listes on accède aux éléments avec l'index

```
# La liste vide  se créé avec []
# Le tuple vide  se créé avec ()
emptyTuple = ()
```

```
# Tuple avec des élements
etudiant_present = ("John",
                    "Sarah",
                    {"nom" : "Ahmed", "present" : True},
                    ["M1-Iris", "L3-Iris"])

for elt in etudiant_present :
  print(elt)
```

Déballage de séquence (Sequence unpacking)

```
#Déballage de séquence
facture = ("Lait", 2, 100, "Leclerc");
produit, quantite, prix, magasin = facture

print("* Le produit {} est acheté à {}".format(facture[0], facture[3]))
print("*** Le produit {} est acheté à {}".format(produit, magasin))
```

```
* Le produit Lait est acheté à Leclerc
*** Le produit Lait est acheté à Leclerc
```

Dictionnaires

- Ensemble non ordonné de paires clés,valeurs
- Les objets sont accessibles via leur clé (pas de notion d'indice)
- Le type de clé permis : String, Nombre
- tuple (non modifiable)
- Les clés sont hachées => lecture et recherche en O(1)
- Pour supprimer une entrée : del

```
dict_1 = dict()
dict_1["nom"]="Iris"
dict_1["age"]=16
dict_1
```

```
{'age': 16, 'nom': 'Iris'}
```

```
dict_2 = {"nom" : "Iris", "age" : 16}

dict_2
```

```
{'age': 16, 'nom': 'Iris'}
```

```
del dict_1["age"]
```

Fonctions

```
def nom_fonction( arguments ):
    """
     La description de ma fonction
    """
    expresion
    ....

    return [expression]
```

```
def afficheur(argument1,argument2):
    """
    This is the second line of the docstring.
    """
    print(argument1)
    return argument2

print(afficheur("Nom","John"))
```

```
Nom
John
```

Bon sens

- 4 espaces pour l'indentation (pas de tab)
- Ne pas mixer les tabs et les espaces
- Longueur maximale d'une ligne fixée à 79 caractères
- Sauter plus qu'une ligne pour séparer les fonctions de haut niveau et une seule ligne pour séparer les méthodes dans une classe
- Quand c'est possible, ajouter des commentaires sur une seule ligne
- Utiliser des espaces entre une expression et les déclarations

Quelques règles


```
import this
"""The Zen of Python, by Tim Peters. (poster by Joachim Jablon)"""

Beautiful is better than ugly.
Explicit is better than impl..
Simple is better than cσмplεx.
Cσмplεx is better than cOmp1|c@ted.
Flat is better than nested.
S p a r s e  is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
raise PythonicError("Errors should never pass silently.")
# Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
# Although that way may not be obvious at first unless you're Dutch.
Now is better than ...                  never.
Although never is often better thanrightnow.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

Pandas

- Pandas est une librairie python qui permet de manipuler facilement des données à analyser :
  - manipuler des tableaux de données

```
s = pandas.Series([1, 2, 5, 7]) : série numérique entière.
s = pandas.Series([1.3, 2, 5.3, 7]) : série numérique flottante.
s = pandas.Series([1, 2, 5, 7], dtype = float) : série numérique flottante
```

- On peut utiliser comme type les types numpy, par exemple :

```
s = pandas.Series([1, 2, 5, 7], dtype = numpy.int8)
```

POO : programmation orientée object

- Python est un langage permettant la programmation orientée objet

- Rappel
  - Encapsulation
  - Héritage
  - Polymorphisme
  - Composition
  - Classe, Object, variable/méthode de classe, attributs, méthodes …..

POO : programmation orientée object

- Classe

Nom de la classe

Les attributs

Les méthodes

POO : programmation orientée object

- Classe

```
class Personne:
    "Définition d'une personne"

    # Constructor
    def __init__(self, nom, prenom, age = 18, taille=170):
        self.nom = nom
        self.prenom = prenom
        self.age = age
        self.taille = taille

    # Afficher une personne
    def afficher(self):
      print("Nom : {} , Prenom {}, age = {}, taille {}"
      .format(self.nom, self.prenom, self.age, self.taille))
```

POO : programmation orientée object

- Objet : instance de classe

```
etudiant = Personne("John", "Doe", taille=175)
etudiant.afficher()
```

```
Nom : John , Prenom Doe, age = 18, taille 175
```

POO : programmation orientée object

- Héritage

```
class Etudiant(Personne):

  def __init__(self,nom, prenom, univ, age = 18, taille=170):
    Personne.__init__(self, nom, prenom, age, taille )
    self.univ = univ

  def afficher(self):
    super().afficher()
    print("L'universite de l 'etudiant est {} ".format(self.univ))
```

POO : programmation orientée object

- Héritage

```
class Etudiant(Personne):

  def __init__(self,nom, prenom, univ, age = 18, taille=170):
    Personne.__init__(self, nom, prenom, age, taille )
    self.univ = univ

  def afficher(self):
    super().afficher()
    print("L'universite de l 'etudiant est {} ".format(self.univ))
```

POO : programmation orientée object

- Héritage

```
etudiant = Etudiant("Jobn", "Doe", "Iris")

etudiant.afficher()

print(isinstance(etudiant, Etudiant))
print(isinstance(etudiant, Personne))
print(issubclass(Personne, Etudiant))
print(issubclass(Etudiant, Personne))
```

```
Nom : Jobn , Prenom Doe, age = 18, taille 170
L'universite de l 'etudiant est Iris
True
True
False
True
```

POO : programmation orientée object

- Attributs de classe

```
class Etudiant():
  listUniv = []
  def __init__(self,nom):
    self.nom = nom
    self.etudList = []

  def addUniv(self, univ):
    Etudiant.listUniv.append(univ)
    self.etudList.append(univ)
```

```
etudiantLicence = Etudiant("John")
etudiantLicence.addUniv("Sorbonne")

print(etudiantLicence.etudList)
print(Etudiant.listUniv)

etudiantMaster = Etudiant("Sarah")
etudiantMaster.addUniv("Iris")

print(etudiantMaster.etudList)
print(Etudiant.listUniv)
```

```
['Sorbonne']
['Sorbonne']
['Iris']
['Sorbonne', 'Iris']
```

Pandas : dataFrames

- Un dataframe se comporte comme un dictionnaire dont les clés sont les noms des colonnes et les valeurs sont des séries.

```
import numpy
import pandas
numpy_array = numpy.array([[1, 2, 3, 4], [5, 6, 7, 8], [19, 9, 9, 10], [2, 25, 6, 10]])
df = pandas.DataFrame(numpy_array, columns = ['Col1', 'Col2', 'Col3', 'Col4'])
print(df)
df.describe()
```

```
   Col1  Col2  Col3  Col4
0     1     2     3     4
1     5     6     7     8
2    19     9     9    10
3     2    25     6    10
```

Pandas : dataFrames

- On peut facilement lire et écrire ces dataframes à partir ou vers un fichier.

```
df = pandas.read_csv('chemin_de_mon_fichier.csv')
```

```
df.to_csv('new_file.csv', sep = '\t')
```

Pandas

- On peut facilement tracer des graphes à partir de ces DataFrames grâce à matplotlib.

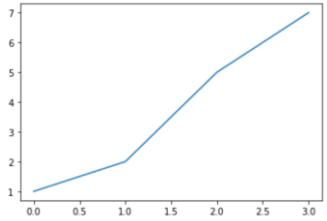
```
DataFrame.plot(x=None, y=None, kind='line', ax=None, subplots=False, sharex=None, sharey=False, layout=None,
figsize=None, use_index=True, title=None, grid=None, legend=True, style=None, logx=False, logy=False,
loglog=False, xticks=None, yticks=None, xlim=None, ylim=None, rot=None, fontsize=None, colormap=None,
table=False, yerr=None, xerr=None, secondary_y=False, sort_columns=False, **kwds) ¶
```

[source]

Make plots of DataFrame using matplotlib / pylab.

- Pour utiliser pandas : import pandas

```
In [16]: s = pd.Series([1, 2, 5, 7])
         s.plot()
Out[16]: <AxesSubplot:>
```



Questions ?

Merci