Statistiques pour la SAé 2.04
Cours 0 + TP0

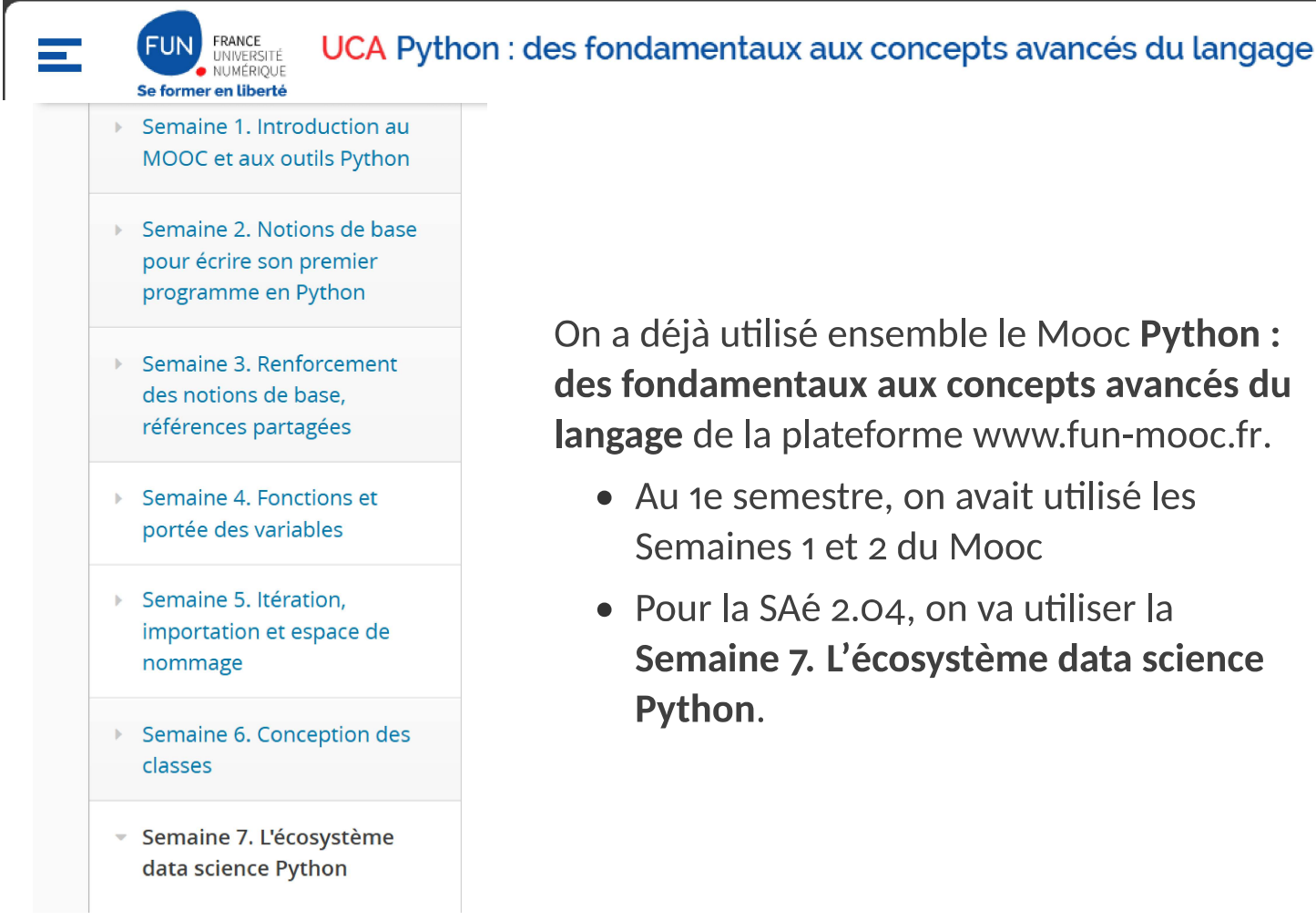
Structures Python
Listes, Numpy.Array et
Pandas.DataFrame

Statistique pour la SAé 2.04

Tiphaine Jézéquel

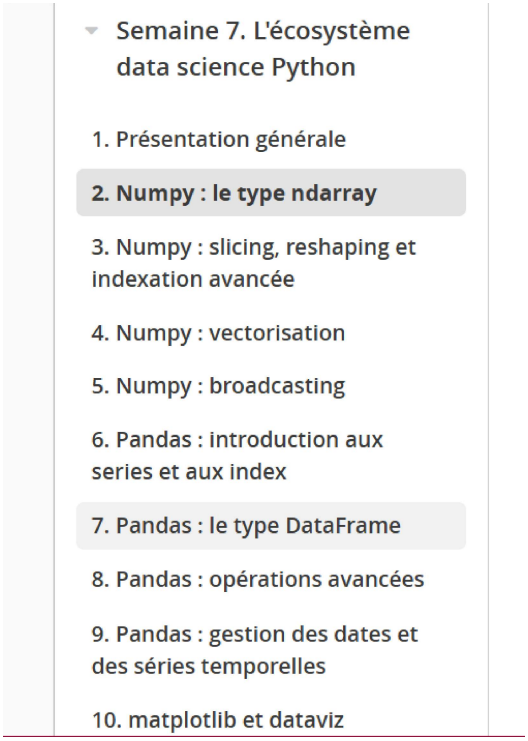
2024-2025

Le Mooc sur Python pour la SAé 2.04



On a déjà utilisé ensemble le Mooc **Python : des fondamentaux aux concepts avancés du langage** de la plateforme www.fun-mooc.fr.

- Au 1e semestre, on avait utilisé les Semaines 1 et 2 du Mooc
- Pour la SAé 2.04, on va utiliser la **Semaine 7. L'écosystème data science Python**.



Plan

1. Listes vs Numpy.Array
2. La structure Pandas.DataFrame

1. Listes *vs* Numpy.Array

Numpy.Array : une structure supplémentaire sur certaines listes

Structure qui s'applique sur

- des listes d'objets de même type

```
In [2]: liste1=[2,3,5]

In [3]: liste1
Out[3]: [2, 3, 5]

In [4]: array1=np.array(liste1)

In [5]: array1
Out[5]: array([2, 3, 5])
```

- ou des listes de listes de même longueur

```
In [9]: liste3=[[2,1],[3,5]]

In [10]: liste3
Out[10]: [[2, 1], [3, 5]]

In [11]: np.array(liste3)
Out[11]:
array([[2, 1],
       [3, 5]])
```

↪ ne s'applique pas sur une liste de listes de longueurs différentes :

```
In [6]: liste2=[[2],[3,5]]

In [7]: liste2
Out[7]: [[2], [3, 5]]

In [8]: array2=np.array(liste2)
C:\Users\tiphaine\AppData\Local\Temp/ipykernel_
VisibleDeprecationWarning: Creating an ndarray
```

Retour simple au type liste

```
In [17]: array1.tolist()
Out[17]: [2, 3, 5]
```

```
In [18]: array3.tolist()
Out[18]: [[2, 1], [3, 5]]
```

Points communs : slicing, mutabilité

Accéder à un élement

- liste ou np.array à 1 dimension :

```
In [22]: liste1[0]
Out[22]: 2

In [23]: array1[0]
Out[23]: 2
```

- liste de listes ou np.array à 2 dimensions :

```
In [24]: liste3[0][1]
Out[24]: 1

In [25]: array3[0][1]
Out[25]: 1

In [26]: array3[0,1]
Out[26]: 1
```

Modifier un élement

- liste ou np.array à 1 dimension :

```
In [27]: liste1
Out[27]: [2, 3, 5]

In [28]: liste1[0]=10

In [29]: liste1
Out[29]: [10, 3, 5]
```

- liste de listes ou np.array à 2 dimensions :

```
In [31]: array3[0,1]=10

In [32]: array3
Out[32]:
array([[ 2, 10],
       [ 3,  5]])
```

Différences : les opérations + et * sur les listes et les np.array

Opérations +, * sur les listes

Pour les listes, + et * se réfèrent à la concaténation :

```
In [34]: liste1
Out[34]: [10, 3, 5]

In [35]: liste1+liste1
Out[35]: [10, 3, 5, 10, 3, 5]
```

```
In [36]: liste1*2
Out[36]: [10, 3, 5, 10, 3, 5]

In [37]: liste1*3
Out[37]: [10, 3, 5, 10, 3, 5, 10, 3, 5]
```

Opérations +, * sur les np.array

Pour les np.array, + et * sont les opérations numériques sur les éléments :

```
In [38]: array1
Out[38]: array([2, 3, 5])

In [39]: array1+array1
Out[39]: array([ 4,  6, 10])
```

```
In [40]: array1*2
Out[40]: array([ 4,  6, 10])

In [41]: array1*3
Out[41]: array([ 6,  9, 15])
```

Quelques conséquences de ces différences

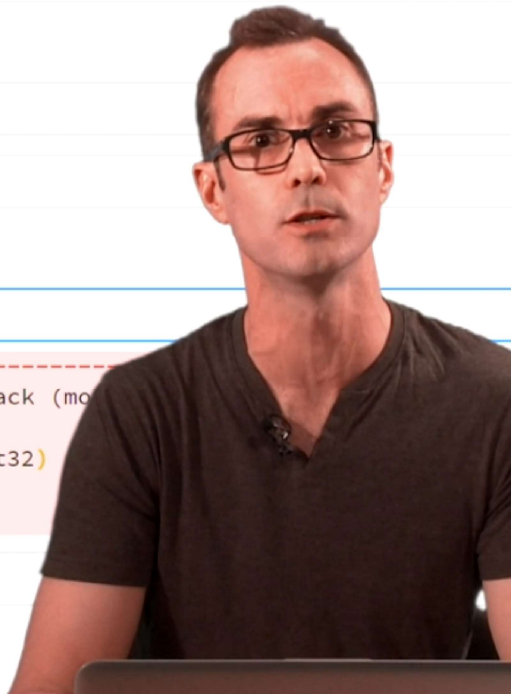
Création d'une liste / d'un numpy.array

```
In [54]: liste4=[]
    ...: for i in range(4):
    ...:     liste4 += [i]

In [55]: liste4
Out[55]: [0, 1, 2, 3]
```

```
In [56]: array4=np.zeros(4)

In [57]: array4
Out[57]: array([0., 0., 0., 0.])

In [58]: for i in range(4):
    ...:     array4[i] = i

In [59]: array4
Out[59]: array([0., 1., 2., 3.])
```

- Les opérations numériques, et donc les fonctions de statistiques, s'utilisent de manière + sûre sur des np.array.
  ↪ On verra la semaine prochaine les fonctions numpy pour la moyenne, la variance etc. , qui s'appliquent à des np.array.
- La concaténation de np.array est + compliquée, il faut utiliser la fonction `np.concatenate`.

```
In [16]: np.array([1, 4.5, 129], dtype=np.int8)

Out[16]: array([   1,    4, -127], dtype=int8)

In [17]: a = np.array([1, 2, np.nan])

In [18]: a.dtype

Out[18]: dtype('float64')

In [19]: a = np.array([1, 2, np.nan], dtype=np.int32)

---------------------------------------------------------------
ValueError                                Traceback (mo
<ipython-input-19-eee848514b22> in <module>()
----> 1 a = np.array([1, 2, np.nan], dtype=np.int32)

ValueError: cannot convert float NaN to integer

In [ ]:
```



2. La structure Pandas.DataFrame

C'est une structure qui se rajoute sur un np.array pour créer un tableau avec des noms de lignes et de colonnes :

	Maths	Prog	Com
Albert	1	2	3
Maria	12	13	10
Zoe	15	12	13

De même que les `array` sont dans la librairie `Numpy`, la structure `dataframe` se trouve dans une librairie Python : la librairie Pandas.

```
import pandas as pd
```

Créer un DataFrame à partir d'un fichier .csv

Pour un fichier `MonFichier.csv` se trouvant dans le même dossier que le fichier Python utilisé :

```
MonDataFrame=pd.read_csv("MonFichier.csv")
```

De np.array à DataFrame et inversement

Pour créer un DataFrame à partir d'un np.array, il faut créer la liste des noms des lignes, et la liste des noms des colonnes, puis les combiner ainsi :

```
In [71]: array5
Out[71]:
array([[ 1,  2,  3],
       [12, 13, 10],
       [15, 12, 13]])

In [72]: noms_lignes=['Albert','Maria','Zoe']

In [73]: noms_colonnes=['Maths','Prog','Com']

In [74]: df5=pd.DataFrame(data=array5,index=noms_lignes,columns=noms_colonnes)

In [75]: df5
Out[75]:
        Maths  Prog  Com
Albert      1     2    3
Maria      12    13   10
Zoe        15    12   13
```

Pour revenir à la structure np.array à partir d'un DataFrame, c'est simple :

```
In [76]: array5=df5.to_numpy()

In [77]: array5
Out[77]:
array([[ 1,  2,  3],
       [12, 13, 10],
       [15, 12, 13]])
```