

# Structure de données et POO

Ressource 2.08

---

Sébastien Bindel

Semestre 2

Université de Haute Alsace

<b>Objectifs</b>	étude des structures de données complexes, introduction à la programmation orientée objet, introduction aux expressions régulières.
<b>Volume horaire</b>	2 CM, 3 TD, 3 TP
<b>Évaluation</b>	une évaluation sur Moodle, un TP noté.

**Objectifs** étude des structures de données complexes,  
introduction à la programmation orientée objet,  
introduction aux expressions régulières.

**Volume horaire** 2 CM, 3 TD, 3 TP

**Évaluation** une évaluation sur Moodle,  
un TP noté.

## Équipe pédagogique

Mustafa Al Samara et Sébastien Bindel

## Données et structure de données

---

<b>Mot-clef</b>	bool
<b>Taille mémoire</b>	24 octets
<b>Valeurs possibles</b>	True ou False.

## Quelques tests sur les booléens

```
>>> import sys
>>> a = True
>>> b = False
>>> a == b
False
>>> a > b
True
>>> sys.getsizeof(bool())
24 # Est ce normal ?
```

## Quelques tests sur les entiers

```
>>> a = 3  
>>> b = 3.1  
>>> a == b  
False  
>>> b = 3.00000000000000000001  
>>> a == b  
True  
>> a << 1 # décalage à gauche (ne marche que pour les entiers !)  
6
```

<b>Mot-clef</b>	string ou byte string.
<b>Taille mémoire</b>	longueur × 1 octet.
<b>Différence</b>	un byte string contient des octets, un string contient des caractères.

## Quelques tests sur les chaînes de caractères

```
>>> a = "aeiouy"
>>> b = b"aeiouy"
>>> a == b
False
>>> for x in b: print(x, end="_")
97 101 105 111 117 121
>>> for x in a: print(x, end="_")
a e i o u y
>>> for x in a: print(ord(x), end="_")
97 101 105 111 117 121
```

## Découpage

```
>>> a = "salut,toi"  
>>> a.split(",")  
['salut', 'toi']
```

## Concaténation

```
>>> a = "salut,toi"  
>>> a+a  
'salut,toisalut,toi'
```

## Liste de caractères

```
>>> a = "salut"  
>>> a[0]  
's'  
>>> a[4]  
't'
```



**Mutable** capable d'être modifiés

**Objet** tableau et liste dynamique.

## Exemple

```
>>> a = b = [1,2,3]
>>> b.append(4)
>>> a == b
?
```

## Description

structure en deux dimensions,  
données hétérogènes,  
taille variable (ligne et colonne indexés).

## Disponibilité

bibliothèque Pandas.

## Utilité

facilite le travail des données.

Marque	Modèle	Année
BMW (S)	Série 1	2015
Mercedes	Classe A	2021

## Chargement

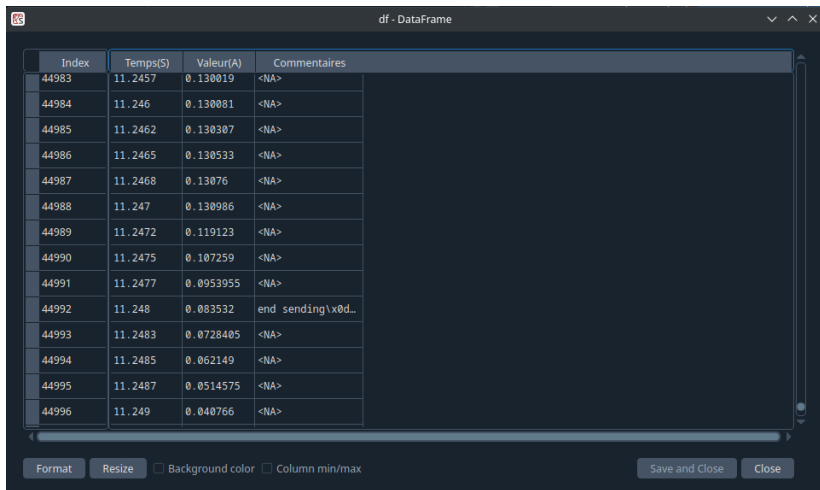
fonction `read_csv()` de Pandas,  
chemin absolu ou relatif,  
définition d'un séparateur (e.g. , tabulation)

## Données

vérifier le type de données,  
utilisation de l'attribut `dtypes`.

### Chargement d'un Dataframe et vérification des types

```
df = None
df = pandas.read_csv("fichier.csv", sep=',', delimiter=None)
df.dtypes
Temps(S)      float64
Valeur(A)     float64
Commentaires  object # Aïe, on devrait avoir une chaîne de caractères
dtype: object
```



Index	Temps(S)	Valeur(A)	Commentaires
44983	11.2457	0.130019	<NA>
44984	11.246	0.130081	<NA>
44985	11.2462	0.130307	<NA>
44986	11.2465	0.130533	<NA>
44987	11.2468	0.13076	<NA>
44988	11.247	0.130986	<NA>
44989	11.2472	0.119123	<NA>
44990	11.2475	0.107259	<NA>
44991	11.2477	0.0953955	<NA>
44992	11.248	0.083532	end sending\x0d...
44993	11.2483	0.0728405	<NA>
44994	11.2485	0.062149	<NA>
44995	11.2487	0.0514575	<NA>
44996	11.249	0.040766	<NA>

## Accès à une variable

```
df['Temps(s)'] # équivalent à df['Temps(s)'][:]  
0          0.00025  
210612     52.65300  
Name: Temps(S), Length: 210613, dtype: float64
```

## Accès à la première ligne

```
>>> df.iloc[0] # utilisation d'un indice  
Temps(S)      0.0  
Valeur(A)     0.037805  
Commentaire  <NA> # Not assigned  
Name: 0, dtype: object
```

```
df.loc[0:1] # utilisation d'un label  
Temps(S)      Valeur(A)  Commentaires  
0  0.00000      0.037805      <NA>  
1  0.00025      0.037805      <NA>
```

## Problème

La variable **Commentaire** est considérée comme un objet et non une chaîne de caractères.

## Solution

Effectuer un cast sur la variable

## Transformation d'un objet en string

```
df['Commentaires'] = df['Commentaires'].astype("string")
df.dtypes
Tems(S)      float64
Valeur(A)    float64
Commentaires string # OK
```

**sauvegarde** fonction `to_csv()` du `DataFrame`,  
exportation de l'entête,  
exportation de l'index.

**Bonne pratique** sauvegarde en deux temps,  
transformation dans le format csv,  
écriture dans un fichier.

## Sauvegarde des données d'un `DataFrame`

```
df = df.to_csv(sep=',', header = True, index=False)
f = open("/tmp/data.csv", 'a') # ouverture du fichier (w: écriture, a:
    ajout)
f.write(df) # écriture dans le fichier
f.close() # fermeture du fichier
```

# Introduction à la POO

---



## Paradigme de programmation

- séquentiel,
- procédural (ce que vous faites),
- fonctionnel,
- **orienté objet**

## Concepts

- objet,
- classe,
- encapsulation,
- ...



**Définition :** un objet est une structure de données qui possède des attributs (propriétés) et des méthodes.

**Définition :** un objet est une structure de données qui possède des attributs (propriétés) et des méthodes.

## Exemple

L'objet Citron représente le fruit ayant le même nom. Dans cet exemple, l'objet Citron possède deux attributs (i) une couleur et (ii) une saveur et une méthode qui est la fonction `presser()`.

**Définition :** un objet est une structure de données qui possède des attributs (propriétés) et des méthodes.

## Exemple

L'objet Citron représente le fruit ayant le même nom. Dans cet exemple, l'objet Citron possède deux attributs (i) une couleur et (ii) une saveur et une méthode qui est la fonction `presser()`.

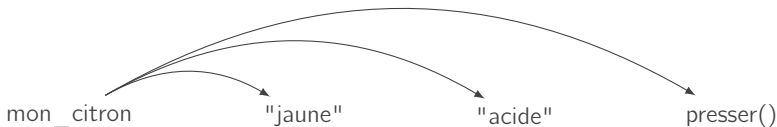


Illustration de la zone mémoire

## Accès aux attributs et aux méthodes

```
>>> mon_citron.couleur  
"jaune"  
>>> mon_citron.saveur  
"acide"  
>>> mon_citron.presser()  
10
```

## Définition

Représentation de la structure d'un objet. Un objet est donc issu d'une classe.

## Définition

Représentation de la structure d'un objet. Un objet est donc issu d'une classe.

## Exemple

La classe Citron possède deux attributs qui sont des chaînes de caractères et une méthode qui retourne un entier.

**Objectif** construction d'un objet, création d'instances  
initialiser les paramètres.

**Mot clef** `__init__()`

**Paramètres** valeurs des attributs,  
`self` qui est l'objet lui même.

## Déclaration du constructeur de l'objet Citron

```
class Citron:
    def __init__(self, sav, col):
        self.saveur=sav
        self.couleur=col
```



## Main.py

```
import sys
import Voiture as myVoiture

def main() -> int:
    citron_jaune = myCitron.Citron("très_acide","jaune")
    print(citron_jaune.couleur) # affiche jaune
    citron_vert = myCitron.Citron("acide","vert")
    print(citron_vert.couleur) # affiche vert
    return (0)

if __name__ == "__main__":
    sys.exit(main())
```

**Objectif**      Instancier un objet sans paramètres.

**Méthode**      Paramètres avec des valeurs par défaut.

**Valeurs**      None de préférence.

## Constructeur avec des valeurs par défaut

```
class Citron:
    def __init__(self, sav=None, col=None):
        self.saveur=sav
        self.couleur=col
# Dans la fonction main
citron_industriel = Citron()
```



<b>attribut publique</b>	accessible en dehors de la classe, à éviter sauf cas exceptionnel.
<b>attribut privé</b>	non accessible en dehors de la classe, respect du principe de l'encapsulation,
<b>Différenciation</b>	le __ avant le nom de l'attribut.

## Classe Citron

```
class Citron:
    def __init__(self, sav=None, col=None):
        self.saveur=sav
        self.couleur=col
```

## Main.py

```
import sys
import Citron as myCitron

def main() -> int:
    monCitron = myCitron.Citron()
    monCitron.couleur = "rouge"
    return (0)

if __name__ == "__main__":
    sys.exit(main())
```

## Classe Citron

```
class Citron:
    def __init__(self, sav=None, col=None):
        self.__saveur=sav
        self.__couleur=col
```

## Main.py

```
import sys
import Citron as myCitron

def main() -> int:
    monCitron = myCitron.Citron()
    monCitron.couleur = "rouge" # AttributeError
    monCitron.__couleur = "rouge" # AttributeError
    return (0)

if __name__ == "__main__":
    sys.exit(main())
```

- Objectif** permettre l'accès aux attributs (lecture/écriture)
- Comment** Définition de fonctions
- Convention** `get__nom` de l'attribut pour la lecture,  
`set__nom` de l'attribut pour l'écriture.

## Classe Citron

```
class Citron:
    def __init__(self, sav=None, col=None):
        self.__saveur=sav
        self.__couleur=col

    def get_saveur(self) -> string :
        return self.__saveur

    def set_couleur(self, new_color):
        self.__couleur = new_color
```

## Main.py

```
import sys
import Citron as myCitron

def main() -> int:
    monCitron = myCitron.Citron()
    monCitron.set_couleur("rouge")
    print(monCitron.get_saveur()) # Retourne None, c.f constructeur vide
    return (0)

if __name__ == "__main__":
    sys.exit(main())
```

**Objectif** détruire proprement les séquences mutables et dictionnaires  
fonction appelée par le ramasse miette.

**Mot clef** `__del__()`

**Paramètres** `self` qui est l'objet lui même.

## Classe Citron

```
class Citron:
    def __init__(self, sav=None, col=None):
        self.__saveur=sav
        self.__couleur=col

    def __del__(self):
        del self.__saveur
        del self.__couleur
```



**Objectif**            personnaliser l'affichage d'un objet.

**Mot clef**            `__str__(self)`

## Classe Citron

```
class Citron:
    def __init__(self, sav=None, col=None):
        self.__saveur=sav
        self.__couleur=col

    def __str__(self):
        return str(self.__saveur)+"⌵"+str(self.__couleur)
```

## Sans redéfinition de la fonction str

```
import Citron as myCitron

def main() -> int:
    monCitron = myCitron.Citron()
    monCitron.set_couleur("rouge")
    print(monCitron) # retourne <__main__.Citron object at 0x7f79dc332350>
    return (0)
```

## Avec redéfinition de la fonction str

```
import Citron as myCitron

def main() -> int:
    monCitron = myCitron.Citron()
    monCitron.set_couleur("rouge")
    print(monCitron) # retourne None rouge
    return (0)
```