BUT 3 Science des données
Compétence 1 : Traiter des données à des fins décisionnelles

Semestre 5 – Bases de données NoSQL (30h)

Typologie NoSQL (13,5h) – S.Abboud

Python MongoDB (16,5h) – F.Garnier

BUT 3 Science des données
Compétence 1 : Traiter des données à des fins décisionnelles

Progression proposée (F.Garnier)

Chapitre 1 : **Rappel langage Python**

Chapitre 2 : **Cartographie**

Chapitre 3 : **Python et MongoDB**

Chapitre 4 : **Programmation objet**

- 1,5 h CM
- 15 h TD

BUT Science des données Niort

Organisation du module

Modalités d'examen

*** 1 TP noté Requêtes MongoDB**

-> Ressources

- **1 Projet Python MongoDB API (+ éléments de cours de M.Abboud)**

-> Saé « Migration de données vers ou depuis un environnement NoSQL » ***mercredi 19/11 et vendredi 5/12 (11h-17h)***

BUT Science des données Niort

Chap. 1 – Rappel du langage Python

- Langage de programmation créé en 1990 par Guido Van Rossum (PB). Langage phare du domaine de la **Data Science** et du **Big Data**

- Langage scripté, disponible sous licence libre et gratuite PSFL

- Editeur au choix à l'IUT

Chap. 1 - Rappel du langage Python

- Typage fort et dynamique des données (à la volée)
- Pas de déclaration de variables
- Indentation obligatoire
- Commentaires : #
- Aide : `help()`
  `help(nomfontion)`
  `help(chaine)`

Chap. 1 - Rappel du langage Python

➢ Types de données

▪ Numérique

- Entier

```
a=4
print(a)
print(type(a))
```

test1.py

```
4
<class 'int'>
```

- Réel

```
a=2.1 #flottant ou réel
print(type(a))
```

```
<class 'float'>
```

- Complexe

```
a=1.5 + 1j
print(a.real)
print(a.imag)
print(type(a))
```

```
1.5
1.0
<class 'complex'>
```

Chap. 1 - Rappel du langage Python

➢ Types de données (suite)

▪ Chaîne de caractères

```
chaine = 'Bonjour Cnam'    # ou avec " "
print(chaine)
print(type(chaine))
```

```
Bonjour Cnam
<class 'str'>
```

▪ Booléen

```
test = (3 > 4)
print(test)
print(type(test))
```

```
False
<class 'bool'>
```

Python étant un langage sensible à la casse, l'écriture des valeurs booléennes est : `True` *et* `False`

Chap. 1 - Rappel du langage Python

➢ Types de données (suite)

- **Liste**

```
maListe = [1, 2, "cnam", True, 2.5]
print (maListe, type(maListe))
```

```
[1,2,cnam',True,2.5]
<class 'list'>
```

- **Tuple**

```
point = (1, 2)
print(point, type(point))
```

```
(1, 2)
<class 'tuple'>
```

- **Dictionnaire**

```
dico = {"p1" : 1.0, "p2" :2.0, "p3" : 3.0}
print(dico, type(dico))
```

```
{'p1': 1.0, 'p2':
2.0, 'p3': 3.0}
<class 'dict'>
```

Chap. 1 - Rappel du langage Python

➢ Opérateurs arithmétiques

Opérations	Symboles	Exemples
addition	+	2 + 5 donne 7
soustraction	-	8 - 2 donne 6
multiplication	*	6 * 7 donne 42
exponentiation (puissance)	**	5 ** 3 donne 125
division	/	7 / 2 donne 3.5
reste de division entière	%	7 % 3 donne 1
quotient de division entière	//	7 // 3 donne 2

```
print(7 * 2.) # int * float -> float
print(type(7 * 2.))
```

```
14.0
<class 'float'>
```

```
print(4/2)  # int / int -> float depuis python 3
print(type(4/2))
```

```
2.0
<class 'float'>
```

```
print(3//2)  # int // int -> int (division entière)
print(type(3//2))
```

```
1
<class 'int'>
```

Chap. 1 - Rappel du langage Python

➢ Opérateurs de comparaison

- `<`
- `>`
- `<=`
- `>=`
- `==`
- `!=`

➢ Opérateurs logiques

- `and`
- `or`
- `not`

➢ Transtypage (cast)

- `int()`
- `float()`
- `complexe()`
- `str()`
- `list()`
- `tuple()`
- `dict()`

➢ Valeur absente : `None`

➢ Suppression :

`del(nomVariable)`

Chap. 1 - Rappel du langage Python

➢ Typage dynamique

```
uneVariable = 1
print(uneVariable, type(uneVariable))
uneVariable = "uneChaine"
print(uneVariable, type(uneVariable))
```

```
1          <class 'int'>
uneChaine  <class 'str'>
```

```
var1 = "Bonjour"
var2 = 1
print(var1+var2)
```

```
TypeError: unsupported operand type(s) for
+: 'int' and 'str'
```

D'autres langages à typage dynamique faible auraient affiché : `Bonjour1`
Il n'y a donc pas de transtypage automatique en Python

Chap. 1 - Rappel du langage Python

➢ Affichage

*Paramètre **end** : Par défaut "\n"*

```
nom = "François"
a = 2
print("Bonjour",nom,"!")
print("a", a, sep = "=")
print("a=", end = "")
print(a)
```

```
Bonjour François !
a=2
a=2
```

➢ Saisie

```
prenom = str(input("Quel est ton prenom ? "))
age = int(input("Quel est ton age ? "))
taille = float(input("Quelle est ta taille ? "))
print ("Bonjour",prenom,"!")
print ("Tu as",age,"ans et tu mesures",taille,"m.")
```

Chap. 1 - Rappel du langage Python

Indentation obligatoire dans l'ensemble des structures de contrôle !

➢ Conditionnelle (alternative)

```
condition1 = False
condition2 = False
if condition1:
        print("condition1 est vraie")
elif condition2:
        print("condition2 est vraie")
else:
        print("condition1 et condition2 sont fausses")
```

```
condition1 et
condition2 sont fausses
```

```
condition1 = condition2 = True
if condition1:
        if condition2:
                print("condition1 et condition2 sont vraies")
```

```
condition1 et
condition2 sont
vraies
```

Chap. 1 - Rappel du langage Python

➢ Conditionnelle (suite)

```
condition1 = True
if condition1:
        print("condition1 est vraie")

        print("toujours dans le if")
```

```
condition1
est vraie
toujours dans
le if
```

```
condition1 = False
if condition1:
        print("condition1 est vraie")
print("en dehors du if")
```

```
en dehors du
if
```

```
condition1 = condition2 = True
if condition1:
        if condition2:
        print("condition1 et condition2 sont vraies")
```

IndentationError
: expected an indented block

Chap. 1 - Rappel du langage Python

➢ Itération (boucle)

- **while** (tant que = de **0** à n itérations)

```
i = 0
while i<5:
        print(i, end="")
        i+=1
print("OK")
```

```
0 1 2 3 4 OK
```

- **while** (exprimant un répéter…jusqu'à : de **1** à n itérations)

```
while True: #on passe forcément au moins une fois
        …
        if condition: # condition du jusqu'à
                break  # on sort de la boucle
```

Chap. 1 - Rappel du langage Python

➢ Itération (suite)

- **for** (pour = nb d'itérations connu, c'est **n** !)

```
for i in [1,2,3]:
        print(i)
```

→ `1 2 3`

```
for i in range(4):
        print(i)
```

→ `0 1 2 3`

```
for i in range(5,10):
        print(i)
```

→ `5 6 7 8 9`

```
for i in range(10,5,-1):
        print(i)
```

→ `10 9 8 7 6`

```
for mot in ["big", "data", "en", "python"]:
        print(mot)
```

→ `Big data en python`

➢ Itération (suite)

▪ for (suite)

```
for lettre in "python":
        print(lettre)
```

```
p y t h o n
```

```
chaine = "python"
for index, lettre in enumerate(chaine):
        print(index,"\t",lettre)
```

```
0       p
1       y
2       t
3       h
4       o
5       n
```

```
chaine = "python" # équivaut au prg précédent
for index in range(len(chaine)):
        print(index,"\t",chaine[index])
```

```
i = 0 # équivaut au prg précédent
for lettre in chaine:
        print(i,"\t",lettre)
        i+=1
```

Chap. 1 - Rappel du langage Python

➢ Itération (suite)

▪ for (suite)

```
dico = {"p1" : 1.0, "p2" :2.0, "p3" : 3.0}
for key, value in dico.items():
        print(key, value)
```

```
p1 1.0
p2 2.0
p3 3.0
```

```
for key in dico:   # ou dico.values() pour parcours
        print(key)   # des valeurs uniquement
```

```
p1
p2
p3
```

```
maListe = list()
for i in range(5):
        maListe.append(i ** 2)
print(maListe)
```

```
[0, 1, 4, 9, 16]
```

```
maListe = [i ** 2 for i in range(5)]
print(maListe)
```

Version identique plus courte ***: list compréhension***

https://python-django.dev/page-comprehension-list-listes-python-cours-debutants

https://miamondo.org/le-langage-python/chapitre-10-la-comprehension-de-liste/

Chap. 1 - Rappel du langage Python

➢ Itération (suite)

- **for** (suite)

```
a = [1,6,0]
b = ["un", "six", "zéro"]
for i,j in zip(a,b):
        print("i=",i,"\tj=",j)
```

i

```
i= 1    j= un
i= 6    j= six
i= 0    j= zéro
```

- **for** (exprimant un tant que ou un répéter…jusqu'à)

```
maListe = [0,2,5,1,3,6,9]
for index, unChiffre in enumerate(maListe): # on recherche la valeur 5 dans la liste
        if unChiffre == 5:
                break
if unChiffre == 5:
        print("5 a été trouvé à l'index ", index)
else:
        print("5 non trouvé")
```

Chap. 1 - Rappel du langage Python

➢ Création et manipulation de **chaînes de caractères**

```
maChaine = "aujourd'hui"
print(len(maChaine))
print(maChaine[7])
print(maChaine[1:5])
print(maChaine[:5])
print(maChaine[2:])
print(maChaine[:])
print(maChaine[::-1])
print(maChaine[::2])
print(maChaine[1:5:2])
print(maChaine[5:1:2])
```

```
11
'
ujou
aujou
jourd'hui
aujourd'hui
iuh'druojua
ajudhi
uo
""
```

Technique du ***slicing*** *: possibilité de préciser un pas d'avancement (`step`)*

Documentation sur le Slicing : https://docs.python.org/release/3.6.2/library/functions.html?highlight=slice#slice

Chap. 1 - Rappel du langage Python

➢ Création et manipulation de **chaînes de caractères : fonctions associées**

```
maChaine = "aujourd'hui"
print(maChaine.upper())
print(maChaine.lower())
print(maChaine.capitalize())
print(maChaine.find("j"))
print(maChaine.replace("jourd'hui"," matin"))
print(maChaine.count("u"))
print(maChaine.split("'"))
```

```
AUJOURD'HUI
aujourd'hui
Aujourd'hui
2
au matin
3
['aujourd', 'hui']
```

Documentation sur les chaînes : https://docs.python.org/3/library/string.html

Chap. 1 - Rappel du langage Python

➢ Création et manipulation de **listes**

Listes : similaires aux chaînes de caractères sauf que les éléments peuvent être de n'importe quel type.

```
maListe = [1, 2, "cnam", True, 2.5]
print(maListe[1:3])
```

→ `[2, 'cnam']`

```
maListe = [ [1,"gea",True], [2,"stid",False], [3,"stid",True] ]
print(maListe[1])
```

→ `[2,"stid",False]`

```
maListe0 = list()
maListe1 = list(range(10,30,2))
maListe2 = list(range(-10,10))
maListe3 = list(range(10,-1,-1))
print(maListe0,maListe1, maListe2, maListe3)
```

→

```
[]
[10, 12, 14, 16, 18, 20, 22, 24, 26, 28]
[-10, -9, -8, -7, -6, -5, -4, -3, -2, -1, 0,
1, 2, 3, 4, 5, 6, 7, 8, 9]
[10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
```

Chap. 1 - Rappel du langage Python

➢ Création et manipulation de **listes** (suite)

```
chaine = "cnam"
maListe = list(chaine)
print(maListe)
```

```
['c','n','a','m']
```

```
print(sorted(maListe))
# OU
maListe.sort()
print(maListe)
# Attention :
print(maListe.sort())
```

```
['a','c','m','n']

['a','c','m','n']

None
```

```
chaine = "cnam"
maListe = list(chaine)
maListe.reverse()
print(maListe)
maListe.sort(reverse=True)
print(maListe)
```

```
['m', 'a', 'n', 'c']

['n', 'm', 'c', 'a']
```

Chap. 1 - Rappel du langage Python

➢ Création et manipulation de **listes** (suite)

```
maListe = []  # Création d'une liste vide Ou maListe = list()
maListe.append("A")  # Ajout d'un élément
maListe.append("A")
maListe.append("b")
print(maListe)
print(maListe.count("A"))
print(len(maListe))
maListe[0] = "a"
maListe[1:3] = ["b","c"]
print(maListe)
maListe.insert(0,"a")
maListe.insert(1,"b")
print(maListe)
maListe.remove("a")  # Suppression uniquement du premier "a"
print(maListe)
del(maListe[2])      # maListe.pop() ôte le dernier élément
print(maListe)
```

Attention,
print(maListe.count())
provoque une erreur, voir
help(list.count)

```
['A', 'A', 'b']
2
3


['a', 'b', 'c']


['a', 'b', 'a', 'b', 'c']

['b', 'a', 'b', 'c']

['b', 'a', 'c']
```

Chap. 1 - Rappel du langage Python

➢ Création et manipulation de **listes** (suite)

```
maListe = ['b', 'c', 'b', 'c']
if "b" in maListe:   # teste la présence d'un élément
        print(maListe.index("b") )   # index du 1er élément trouvé
else:
        print("pas de b")
```

```
0
```

```
l1 = [1,2,3]
l2 = [4,5,6]
print(l1 + l2) # Concaténation
l1.append(l2)
print(l1)
print(l2 * 2)  # Duplication
```

```
[1, 2, 3, 4, 5, 6]

[1, 2, 3, [4, 5, 6]]
[4, 5, 6, 4, 5, 6]
```

Chap. 1 - Rappel du langage Python

➢ Création et manipulation de **listes** (suite)

```
maListe1 = [3,1,9,7]
maListe2 = [x**2 for x in maListe1]
print(maListe2)
maListe3 = [x**2 for x in maListe1 if x>4]
print(maListe3)
```

*Mécanisme de **list compréhension** valable également sur les chaînes et les tuples*

```
[9, 1, 81, 49]

[81, 49]
```

```
maListe1 = [1,2,3,4]
maListe2 = maListe1     # Copie de liste
print(maListe2)
maListe1[0] = 5
print(maListe1,maListe2)
maListe2[1] = 0
print(maListe1,maListe2)
# pour éviter ce problème de passage de référence
maListe2 = maListe1.copy()
maListe1[0] = -1
print(maListe1,maListe2)
```

```
[1,2,3,4]

[5,2,3,4] [5,2,3,4]

[5,0,3,4] [5,0,3,4]

[-1,0,3,4] [5,0,3,4]
```

***help(list)** pour en savoir davantage.*

➢ Création et manipulation de **tuples**

Contrairement aux chaînes, listes et dictionnaires, **les tuples ne sont pas modifiables !**
La syntaxe de création est la suivante : `monTuple = (…, …, …)` OU `monTuple = …, …, …`

```
point = (1, 2)
print(point[0])
x, y = point
print(x,y)
point[0] = 2
```

```
1


1 2
TypeError: 'tuple' object does not support
item assignment
```

Il est possible d'utiliser les mêmes outils d'indexation en séquence vu pour les chaînes.

➢ Création et manipulation de **dictionnaires**

Ils servent à stocker des données sous la forme clé-valeur.
La syntaxe est la suivante : `{clé1 : valeur1, clé2 : valeur2, …}`

```
dico1 = {"p1" : 1.0, "p2" :2.0, "p3" : 3.0}
dico2 = dict(p1=1.0, p2=2.0, p3=3.0) # autre possibilité
# Les dictionnaires ne sont pas ordonnés !
print(dico1["p1"])
dico1["p1"] = "A"
dico1["p2"] = "B"
dico1["p4"] = "D"   # Ajout
print(dico1)
del(dico1["p1"])    # Suppression clé : Ou del dico1["p1"] sans les ()
print(dico1)
if "p1" in dico1:   # Test de présence
…
print(dico1["p6"])
```

```
1.0

{'p1': 'A',
'p2': 'B',
'p3': 3.0,
'p4': 'D'}

{'p2': 'B',
'p3': 3.0,
'p4': 'D'}

KeyError:'p6'
```

Chap. 1 - Rappel du langage Python

➢ Création et manipulation de **dictionnaires** (suite)

Contenu plus complexe :

```
monDico = {"nom": "Garnier", "prénom": "François",
"langage": ["JAVA", "Python", "SQL", "VBA"],
"département": {"nom":"STID", "lieu":"PUN"}}
print(monDico,len(monDico)
print(monDico["prénom"])
print(monDico["langage"])
print(monDico["langage"][1])
print(monDico["département"])
print(monDico["département"]["nom"])
print(monDico.get("prénom"))
print(monDico.keys())
print(monDico.values())
```

```
{'nom': 'Garnier', 'prénom': 'François',
'langage': ['JAVA', 'Python', 'SQL', 'VBA'],
'département': {'nom': 'STID', 'lieu': 'PUN'}}

4
François
['JAVA', 'Python', 'SQL', 'VBA']

Python
{'nom': 'STID', 'lieu': 'PUN'}

STID
François
dict_keys(['nom', 'prénom', 'langage',
'département'])
dict_values(['Garnier', 'François', ['JAVA',
'Python', 'SQL', 'VBA'], {'nom': 'STID', 'lieu':
'PUN'}])
```

➢ Création et manipulation de **dictionnaires** (suite)

```
monDico.popitem()  #enlève le dernier item
print(monDico)
monDico.pop("nom")
print(monDico)

monDico["type"] = "PRAG" #Ajout item
print(monDico)

monDico2 = monDico.copy() # non liés

fruits = ["pommes", "bananes", "poires", "oranges"]
nombres = [5, 2, 10, 4]
print({fruits[i]:nombres[i] for i in range(4)})
# OU
print(dict(zip(fruits, nombres)))
```

```
{'nom': 'Garnier', 'prénom': 'François',
'langage': ['JAVA', 'Python', 'SQL', 'VBA']}

{'prénom': 'François', 'langage': ['JAVA',
'Python', 'SQL', 'VBA']}

{'prénom': 'François', 'langage': ['JAVA',
'Python', 'SQL', 'VBA'], 'type': 'PRAG'}

{'pommes': 5, 'bananes': 2, 'poires': 10,
'oranges': 4}
```

Chap. 1 - Rappel du langage Python

➢ Modules et imports

```
import math # pour utiliser les fonctions cos et pi
x = math.cos (2 * math.pi)
print(x)
```

→ 1.0

```
from math import cos, pi # from math import * : autre possibilité
x = cos (2 * pi)
print(x)
```

→ 1.0

```
import math as m
print(m.cos(1))
```

```
import fractions
a = fractions.Fraction(2,3)
b = fractions.Fraction(1,2)
print (a + b)
```

→ 7/6

- *help peut aussi être utilisée sur des modules, vous obtiendrez une description de l'ensemble des fonctions du module :* ***help(math)***
- *Possibilité de créer ses propres modules contenant des fonctions à écrire*

➢ Création de Fonctions

Une fonction est définie en Python par la syntaxe suivante : **`def`** `nomFonction(param1,param2,…)` **`:`**
Une procédure est une fonction sans clause `return`.
L'indentation doit être respectée.

```
def maProcedure1():
        print("proc1")  # pas de return ici
                        # équivaut à une procédure

# Programme principal (appelant)
maProcedure1()      # Appel de procédure
```

proc1

```
def maProcedure2(chaine):
        """
        Affichage d'une chaîne et de sa longueur
        """
        print(chaine," est de longueur ", len(chaine))
help(maProcedure2)
```

maProcedure2(chaine)
Affichage d'une chaîne et de sa longueur

Chap. 1 - Rappel du langage Python

➢ Création de Fonctions (suite)

```
maProcedure2("cnam")
maProcedure2([1,2,3])
print(maProcedure2("cnam")) # ne retourne rien
```

```
cnam  est de longueur  4
[1, 2, 3]  est de longueur  3
cnam  est de longueur  4
None
```

```
def carre(x):
        """
        Fonctionne que retourne le carré de x
        """
        return x ** 2
print(carre(4))    # Appel de fonction
```

```
16
```

```
def puissance(x:int)->tuple:
        """
        Retourne les 3 premières puissances de x
        """
        return x ** 2, x ** 3, x ** 4
monTuple = puissance(4)
print(monTuple, type(monTuple), len(monTuple), monTuple[1])
x2, x3, x4 = puissance(3)
print(x2, x3)
```

```
(16, 64, 256)
<class 'tuple'>
3
64


9 27
```

➢ Création de Fonctions (suite)

```
def fonction3(x, p=2, debug=False):
        if debug:
                print("évalue ma fonction avec x =", x, " et l'exposant p =",p)
        return x ** p

print(fonction3(5)) # les 2ème et 3ème paramètres sont optionnels
print(fonction3(5,3))
print(fonction3(5,debug=True))
print(fonction3(p=3, debug=True,x=7)) # n'importe quel ordre possible
```

```
25

125

évalue ma fonction avec x = 5 et l'exposant p = 2
25

évalue ma fonction avec x = 7 et l'exposant p = 3
343
```

➢ Création de Fonctions (suite)

Passage par référence des paramètres (plus facile pour contrôler les allocations mémoires). Attention aux effets de bords produit en Python : le paramètre peut être modifié dans la fonction s'il est de type mutable :

```
def fonction4(maListe):
        maListe.append(4)
        return None # C'est donc une procédure
maListe = [1, 2, 3]
fonction4(maListe)  # Appel de procédure
print(maListe)
```

```
[1, 2, 3, 4]
```

Evitez de modifier la variable originale au sein de la fonction, passer plutôt par une copie de la variable :

```
def fonction4(maListe):
        nouvelleListe = list(maListe)  # trantypage pas obligatoire
        nouvelleListe.append(4)
        return nouvelleListe    # C'est donc une fonction !
maListe = [1, 2, 3]
maListe = fonction4(maListe)  # Appel de fonction
print(maListe)
```

```
[1, 2, 3, 4]
```

➢ Pureté d'une fonction

Une fonction est dite **pure** si elle n'a pas d'effet de bords lors de son appel.

Dans la fonction `f1()` définie ci-dessous, la variable `a` réceptrice de l'affectation est locale et son affectation ne change pas la valeur de la variable `a` globale. `f1()` sera donc considérée comme **pure**.

```
def f1(a):
        a = [b**2 for b in a]
        return a
a = list(range(5))
print(a)
print(f1(a))
print(a)
```

```
[0, 1, 2, 3, 4]
[0, 1, 4, 9, 16]
[0, 1, 2, 3, 4]
```

➢ Pureté d'une fonction (suite)

La variable `a` globale est modifiée suite à l'appel de la fonction. `f2()` est considérée **impure**.

```
def f2(a):
        for i in a:
                a[i] = a[i] ** 2
        return a
a = list(range(5))
print(a)
print(f2(a))
print(a)
```

```
[0, 1, 2, 3, 4]
[0, 1, 4, 9, 16]
[0, 1, 4, 9, 16]
```

➢ Gestion des erreurs

Bloc `try…exept…finally`

```
def somme(var):
        try:
                resultat = sum(var)
        except:
                print("Erreur, somme impossible !")
                resultat = None
        finally:
                return resultat

nb1 = somme([1,4,8])
print(nb1)
nb2 = somme([1,4,"huit"])
print(nb2)
```

```
13
Erreur, somme
impossible !
None
```

➢ Fonctions sur les listes

Lors de l'utilisation de listes, il est possible d'utiliser des fonctions *high order functions*, qui prennent en paramètre une fonction et une liste (ou un tuple)

```
def carre(x):
        return x ** 2
maListe = [12, 7, 2, 6, 11]
maListe = list(map(carre,maListe))
print(maListe)
```

```
[144, 49, 4, 36, 121]
```

La fonction **map** retourne un objet `map` qu'il faut transtypé en `list`

```
def pair(x):
        return x % 2 == 0
maListe = [1,2,3,4,5,6,7,8]
maListe = list(filter(pair,maListe))
print(maListe)
```

```
[2, 4, 6, 8]
```

La fonction **filter** retourne un objet `map` qu'il faut transtypé en `list`

➢ Fonctions sur les listes (suite)

La fonction **reduce** réduit une liste **en une valeur** en rapport avec la fonction en paramètre (reduce appartient au module `functools`) :

```
from functools import *

def somme(x,y):
        return x + y

maListe = [12, 7, 2, 6, 11]
print(reduce(somme,maListe))

print(reduce(somme,maListe,100))   # avec valeur initialisée à 100

print(sum(maListe))  # évidemment plus simple dans ce cas
```

```
38

138

38
```

Cette fonction **reduce** regroupe ici deux éléments de liste ensemble, puis regroupe le résultat précédent avec l'élément suivant, et ainsi de suite, jusqu'à épuisement de la liste.

➢ Fonctions anonymes

En python, il est courant de passer des fonctions en paramètre au sein d'autres fonctions. Dans ce cadre, il existe un mécanisme évitant de déclarer la fonction au préalable : les **fonctions anonymes** déclarées via l'opérateur `lambda`. L'inconvénient est que la fonction doit tenir sur une ligne et que l'on ne peut pas la réutiliser sans la réécrire.

```
maListe = [12, 7, 2, 6, 11]
maListe = list(map(lambda x: x**2,maListe))
print(maListe)
```

```
[144, 49, 4, 36, 121]
```

```
maListe = [12, 7, 2, 6, 11]
maListe = list(map(lambda x: x**2 if x > 7 else -(x**2),maListe))
print(maListe)
```

```
[144, -49, -4, -36, 121]
```

➢ Lecture d'un fichier

```
fichier = open("1-Iris.txt")    # Jeu de données Iris de Fisher*
toutesLesLignes = fichier.readlines()    # variable de type str
fichier.close()
print(toutesLesLignes[0:3])
print(len(toutesLesLignes))
```

```
['"Sepal Length"\t"Sepal Width"\t"Petal Length"\t"Petal Width"\t"Species"\n',
    '5.1\t3.5\t1.4\t0.2\tsetosa\n',
    '4.9\t3\t1.4\t0.2\tsetosa\n']
151
```

* *Les données de ce fichier sont célèbres. Elles ont été collectées par Edgar Anderson. Le fichier donne les mesures en centimètres des variables suivantes :*

— longueur du sépale (Sepal.Length),
— largeur du sépale (Sepal.Width),
— longueur du pétale (Petal.Length),
— largeur du pétale (Petal.Width)

*pour trois espèces d'iris qui sont les : 1. Iris setosa, 2. Iris versicolor et 3. Iris virginica. Sir R.A. Fisher a utilisé ces données pour construire des combinaisons linéaires des variables permettant de séparer au mieux les trois espèces d'iris. Ce jeu de données contient **150 iris**.*

https://fr.wikipedia.org/wiki/Iris_de_Fisher

➢ Lecture d'un fichier (ligne à ligne)

```
from csv import *   # pour le reader ou avec module pandas (chap.2)
fichier = open("1-Iris.txt", "r")

# Création du lecteur
lecteur = reader(fichier)

# Affichage de chaque ligne du fichier
for ligne in lecteur:
        tabLigne = ligne[0].split("\t")   # éclate les données séparées par une tabulation
        print(tabLigne[0])     # on aura tous les "Sepal length"
fichier.close()
```

➢ DataFrame utilisation

Soit le fichier `pourboires_cours.csv` décrivant les pourboires donnés à un serveur sur une période :

	A	B	C	D	E	F	G
1	Montant_commande	pourboire	sexe	fumeur	jour_semaine	heure	nbPersonnes
2	16.99	1.01	Femme	Non	Dimanche	Diner	2
3	10.34	1.66	Homme	Non	Dimanche	Diner	3
4	21.01	3.5	Homme	Non	Dimanche	Diner	3
5	23.68	3.31	Homme	Non	Dimanche	Diner	2
6	24.59	3.61	Femme	Non	Dimanche	Diner	4
7	25.29	4.71	Homme	Non	Dimanche	Diner	4
8	8.77	2.0	Homme	Non	Dimanche	Diner	2
9	26.88	3.12	Homme	Non	Dimanche	Diner	4

```
import pandas

df = pandas.read_csv("pourboires_cours.csv", header=0, sep=";")

# df = DataFrame - header = 0 permet d'indiquer que la première ligne est une ligne d'en-tête
```

➢ DataFrame utilisation

Nom de la méthode (Procédure, fonction ou propriété)	Résultat
`df.info()`	Noms, types des colonnes, et nbvaleurs non nulles
`df.head()`	Les 5 premières lignes
`df.tail()`	Les 5 dernières lignes
`df.columns` OU `df.index`	Liste des colonnes ou des index de ligne
`df.shape`	Taille du DataFrame (nbLignes, nbColonnes)
`len(df)` OU `len(df.columns)`	Nb de lignes ou nb de colonnes
`df.describe()`	Stats descriptives sur toutes les colonnes quanti
`df["pourboire"].describe()` `#df.pourboire.describe()`	Stats descriptives sur une colonnes quanti
`df["pourboire"].mean()` `#possible avec max, min, count, sum, prod, median, std, var, mad …`	Moyenne d'une colonne numérique
`df.mean(axis = 1)`	Moyenne des colonnes numériques de la ligne 1

Chap. 1 - Rappel du langage Python

➢ DataFrame utilisation

`df.corr()`	Renvoie une matrice de corrélation pour chaque colonne quantit entre elles. Plus la valeur se rapproche de 1, plus la corrélation est forte.
`df2 = df.loc[:,['Montant_commande', 'sexe']]`	Renvoie un nouveau DataFrame avec toutes lignes et deux colonnes
`df['Montant_commande'][0:3]`	Renvoie les 3 premières valeurs de la colonne
`df.iloc[0]  OU  df.loc["nomIndex1"])` `df.iloc[1:3,2:4] # 2 lignes, 2 colonnes`	Renvoie la première ligne Renvoie les lignes 1 et 2 pour les colonnes 2 et 3
`df2 = df[df["nbPersonnes"] > 2]`	Renvoie un df avec les lignes sélectionnées
`df2=df[df['jour_semaine'].isin(["Jeudi", "Vendredi"])]`	Renvoie un df avec les lignes sélectionnées
`df2 = df[(df["nbPersonnes"] > 2) & (df["sexe"]=="Femme")]`	Double condition
`df2 = df.query("nbPersonnes > 2 and sexe =='Femme'")` `#une recherche peut se faire également avec numpy.where`	Même résultat que la ligne d'au-dessus !
`df.jour_semaine.unique()` `df['jour_semaine'].unique()`	Liste des valeurs sans doublons pour la colonne
`df.groupby(['sexe']).size())`	Equivalent du group by sql ici avec la taille (count) : Nb d'hommes et de femmes
`df.groupby(['sexe']).mean())`	Moyenne par sexe pour les colonnes quantitatives
`df.to_csv("fichier.csv",sep=";", encoding="utf-16-be",index=False)`	Ecriture du df dans un fichier csv

Chap. 1 - Rappel du langage Python

➢ DataFrame modification

Nom de la méthode (Procédure ou fonction)	Cas d'utilisation
`df["pourboire"] = df["pourboire"].round(1)`	Arrondir une colonne numérique

`df=df.sort_values(by = 'Montant_commande')`
`df=df.sort_values(by = 'pourboire', ascending = False)`
`df=df.sort_values(by = ['nbPersonnes', 'pourboire'], ascending = [True, False])`
	Tri du df par ordre croissant par défaut

`df=df.rename(columns = {'sexe': 's', 'nbPersonnes': 'nb'})`	Renomme des colonnes
`df=df.assign(Montant_total_clt = df['Montant_commande'] + df['pourboire'])`	Rajoute une colonne
`df=df.insert(0, 'nouvelleCol', listeValeurs)`	Insertion d'une colonne à une position donnée
`df.drop(columns=['sexe', 'nbPersonnes'], inplace = True)` `df=df.filter(["Montant_commande","pourboire", "fumeur","jour_semaine","heure"], axis=1)`	Suppression de colonnes (2 possibilités) : - Drop : celles que l'on enlève - Filter : celles que l'on garde
`df.drop(index = [0, 3], inplace = True)`	Suppression de lignes 0 et 3

`df['nbPersonnes'][df['nbPersonnes']<=2] = 0`
*Préférable d'utiliser une fonction **anonyme** :*
`df['nbPersonnes'] = df['nbPersonnes'].apply( lambda x: 0 if x <= 2 else x)`
	Modification des valeurs d'une colonne.
	
	Applique une fonction pour toutes les lignes du df

`df.dropna(inplace = True)`	Supprime les lignes qui ont au moins une valeur nulle
`df.fillna(0)`	Toutes les valeurs NaN sont remplacées par des 0
`df['pourboire'].fillna(0, inplace = True)`	Remplace les NaN par des 0 dans la colonne
`df = df.replace("Diner","Soir")` `# applicable aussi à une colonne`	Remplace des valeurs

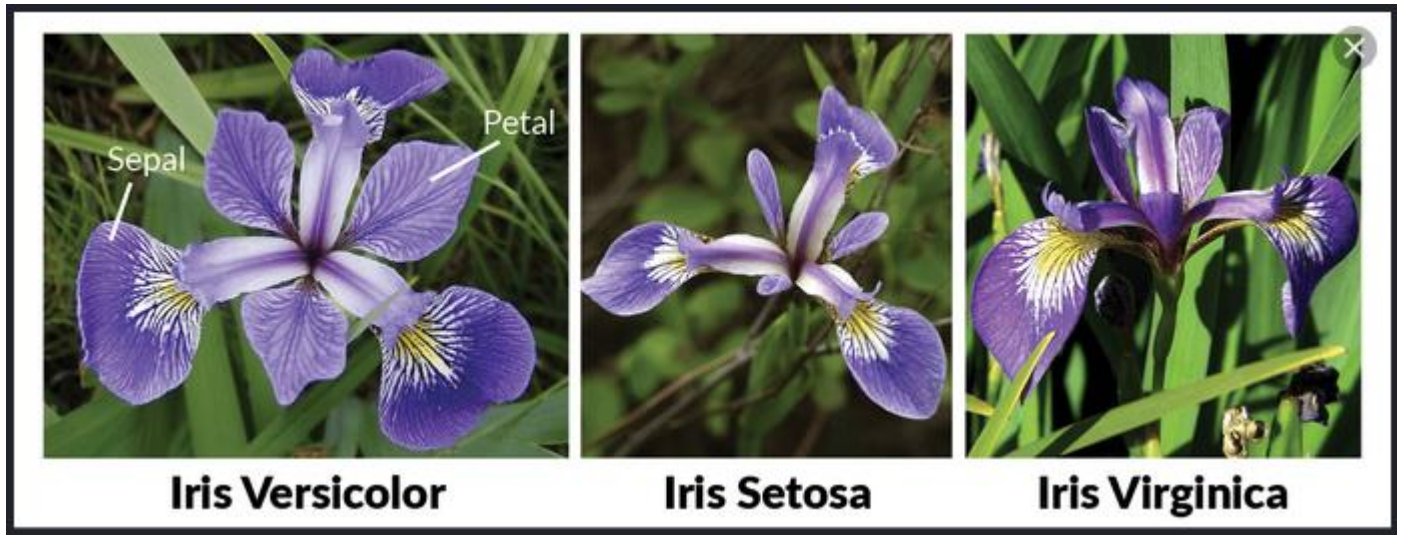
`df.astype({'pourboire': float,'nbPersonnes': int})`
`df['date'] = df['date'].astype(str)`
`df['date'] = pandas.to_datetime(df["date"])`
	Transtyper des colonnes

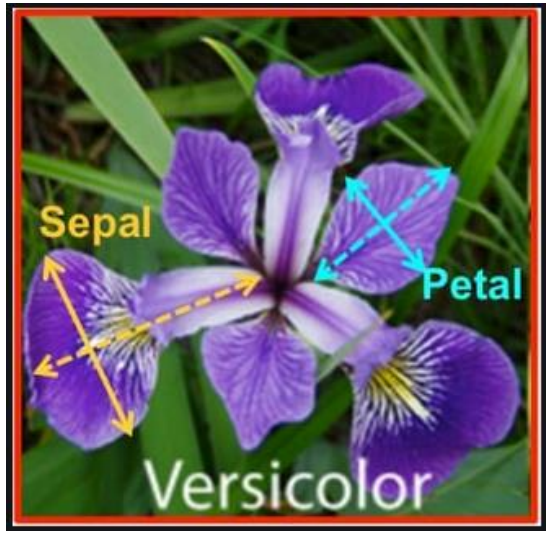
`df1.append(df2)`	Ajoute les lignes de df1 à la fin de df2 (le nombre de colonnes doit être identique)
`pandas.concat([df1, df2], axis=1)`	Ajoute les colonnes de df1 à la fin de df2 (Le nombre de lignes doit être identique)

➢ DataFrame modification

`df1.join(df2, on=col1, how='inner')`	Créer une jointure des df1 et df2 par la colonne col1 qui ont des valeurs identiques.
`df1.merge(df2, left_on='lkey', right_on='rkey', how='outer')`	Fusion des DataFrame
`df['latitude']=df['gps'].apply( lambda x: x[0]) df['Longitude']=df['gps'].apply( lambda x: x[1]) df.drop(columns = ['gps'],inplace=True)`	Remplacement d'une colonne de type complexe en autant de colonne que de valeurs dans la colonne complexe
`df =df.drop_duplicates()`	Suppression de doublons sur toute la ligne
`df =df.drop_duplicates(subset = ['A', 'B']) :`	Suppression de doublons sur colonnes A et B
`df = pd.DataFrame({ "Nom" : liste_nom, "Prenom" : liste_place, "DateNaissance" : liste_point, "Salaire" : liste_salaire })`	Création d'un DataFrame avec des listes de valeurs.

EXERCICE


Sepal
Petal
Iris Versicolor
Iris Setosa
Iris Virginica


Sepal
Petal
Versicolor

➢ Exercice 1

1.Télécharger le fichier `"1-Iris.txt"`, importer et afficher les lignes dans une unique variable chaîne.

2. Créer une liste composée de 151 listes à partir de cette variable chaîne. Chaque sous-liste contiendra les 5 informations présentes sur chaque ligne.

Il faudra donc supprimer les guillemets de la 1ère ligne d'en-tête (« " ») , le caractère de fin de ligne ("\n") et enfin les « \t ». Utiliser « replace » entre autre… !

3. Créer une fonction `strToFloat(…)` permettant de transformer une chaîne en réel :

- Si cette chaîne n'est pas transformable, la renvoyer telle quelle
- Exemples : `strToFloat("1.234") => 1.234` et `strToFloat("cnam") => "cnam"`

4. En utilisant la liste de la question 2, créer une liste de 150 dictionnaires, chaque dictionnaire sera un iris, avec les 5 champs nommés.

Les noms des variables sont sur la première ligne de la liste précédemment créée. Idéalement programme à écrire en une fois avec des "list compréhension" et l'utilisation de la fonction écrite à la question 3 pour transtyper les valeurs lues.

5. Calculer pour chaque iris le rapport entre la surface d'un pétale et la surface d'un sépale. La surface de chacun (pétale, sépale) est le produit de la largeur et de la longueur. Cette information sera un nouveau champ du dictionnaire

6. Créer une liste ne contenant que les *iris setosa*.

7. Calculer la moyenne des 4 variables initiales (longueur, largeur pétale et sépale).

8. Calculer la moyenne des 4 variables initiales pour chaque espèce.

Rappel du langage PYTHON

Documentation du langage python 3 : https://docs.python.org/3/reference/

Bibliothèque standard Python : https://docs.python.org/3/library/