Montpellier Bio-Stat

https://groupes.renater.fr/wiki/montpellier-biostat

MUSE

MONTPELLIER UNIVERSITÉ D'EXCELLENCE

Formation Python débutant

Aubin THOMAS pour le réseau KIM Data Sciences

Institut de Génétique Humaine, CNRS, Montpellier

https://github.com/aubinthomas/Course

Plan

- Présentation de Python
- Types de données
  - Variables
  - Listes
  - Dictionnaires
  - Tableaux & matrices
- Structures de contrôle
- Structures de données avancées
  - Dataframes avec Pandas

Plan

- Présentation de Python
- Types de données
  - Variables
  - Listes
  - Dictionnaires
  - Tableaux & matrices
- Structures de contrôle
- Structures de données avancées
  - Dataframes avec Pandas

- Pas d’usage de librairies graphiques
  - Seaborn
- Pas de programmation objet
  - Python avancé
- Pas d’analyse de données
  - Scikit-learn

Présentation de Python

- Langage interprété
  - les instructions sont traduites à l'exécution par l'interpréteur. Cela impacte les performances mais permet une portabilité de code sur l'ensemble des machines pouvant faire tourner Python
- Haut niveau
  - Toutes les structures de code modernes
- Orienté objet

Présentation de Python

Pourquoi utiliser Python?

- Portabilité du code
- Structures de données modernes
- Courbe de progression rapide
- Un ensemble de librairies importantes (Math, Machine Learning, Graphiques, base de données, cartographie, Web)

Faiblesses

- Performances faibles sans l'usage de librairies spécialisées (CPU, gestion mémoire)
- Pas de multi threading

Types de données variables

- Python adopte l'affectation dynamique des variables

```
### Création de variables de typages différents
a = 1  # int
b = 2.3  # float
c = 'Cours de Python'  # string
```

```
### Fonction type pour connaître le type d'une variable
print( "a de type ", type(a) )
print( "b de type ", type(b) )
print( "c de type ", type(c) )
```

```
a de type  <class 'int'>
b de type  <class 'float'>
c de type  <class 'str'>
```

Types de données variables

- Python adopte l'affectation dynamique des variables

```
### Création de variables de typages différents
a = 1  # int
b = 2.3  # float
c = 'Cours de Python'  # string
```

```
### Fonction type pour connaître le type d'une variable
print( "a de type ", type(a) )
print( "b de type ", type(b) )
print( "c de type ", type(c) )
```

```
a de type  <class 'int'>
b de type  <class 'float'>
c de type  <class 'str'>
```

```
### Le typage est dynamique
a = a+0.5
b = 6
c = a
c = str(c)
print( "a de type ", type(a) )
print( "b de type ", type(b) )
print( c, "  c de type ", type(c) )
```

```
a de type  <class 'float'>
b de type  <class 'int'>
1.5   c de type  <class 'str'>
```

Types de données variables

- Python est un langage objet : chaque objet dispose de ses attributs et de ses fonctions internes, ou méthodes.

```
#pour lister les méthodes d'un objet, utiliser la fonction dir:
dir(a) #a est un float
```

```
['__abs__',
 '__add__',
 '__bool__',
 '__class__',
 '__delattr__',
 '__dir__',
 '__divmod__',
 '__doc__',
 '__eq__',
 '__float__',
 '__floordiv__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__getformat__',
 '__getnewargs__',
 '__gt__',
 '__hash__',
 '__init__',
 '__init_subclass__',
 '__int__',
 '__le__',
```

Types de données variables

- Python est un langage objet : chaque objet dispose de ses attributs et de ses fonctions internes, ou méthodes.

```
#pour lister les méthodes d'un objet, utiliser la fonction dir:
dir(a) #a est un float
```

```
['__abs__',
 '__add__',
 '__bool__',
 '__class__',
 '__delattr__',
 '__dir__',
 '__divmod__',
 '__doc__',
 '__eq__',
 '__float__',
 '__floordiv__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__getformat__',
 '__getnewargs__',
 '__gt__',
 '__hash__',
 '__init__',
 '__init_subclass__',
 '__int__',
 '__le__',
```

```
dir(c) # c est une chaine de caractères
```

```
['__add__',
 '__class__',
 '__contains__',
 '__delattr__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__getitem__',
 '__getnewargs__',
 '__gt__',
 '__hash__',
 '__init__',
 '__init_subclass__',
 '__iter__',
 '__le__',
 '__len__',
 '__lt__',
 '__mod__',
 '__mul__',
 '__ne__',
 '__new__',
 '__reduce__',
 '__reduce_ex__',
```

Types de données variables

- Python est un langage objet : chaque objet dispose de ses attributs et de ses fonctions internes, ou méthodes.

```
### Création de variables de typages différents
a = 1  # int
b = 2.3  # float
c = 'Cours de Python'  # string
```

```
print(c.split("de")) #split découpe une chaine de caracteres
d = c.replace("Python","R") #replace remplace une chaine de caracteres par une autre
print(d)
print(c.count('o')) #count compte le nombre d'occurences d'une sous chaine
print(c.count('ou'))
```

```
['Cours ', ' Python']
Cours de R
2
1
```

Types de données
opérateurs de variables

Python expression	Meaning
`not(a)`	not a
`a == b`	a equal b
`a != b`	a not equal b
`a & b`	a and b
`a | b`	a or b
`a >= b`	a greater equal b
`a > b`	a greater b
`a <= b`	a less equal b
`a < b`	a less b

Python expression	Meaning
`x + y`	sum of x and y
`x - y`	difference of x and y
`x * y`	product of x and y
`x / y`	quotient of x and y
`x // y`	floored quotient of x and y
`x % y`	remainder of x / y
`-x`	x negated
`abs(x)`	absolute value or magnitude of x
`complex(re, im)`	a complex $mathre + i \times im$
`c.conjugate()`	conjugate of the complex number c
`divmod(x, y)`	the pair `(x // y, x % y)`
`pow(x, y)`	x to the power y
`x ** y`	x to the power y

```
a = 5
b = 2
print( a==5 )
print( b!=2 )
print( (a==5) & (a>b) )
```

```
True
False
True
```

Mêmes opérateurs pour les chaines de caractères

```
A= "to"
B = "ro"
print(A==B)
print(A!=B)
print(A+B)
print(A*2+B)
```

```
False
True
toro
totoro
```

Exercice 1

1/ Ecrire un programme qui

- définit 2 variables : un rayon et une hauteur
- calcule le volume d'un cône $V = 1/3 * pi * r^2 * h$
    - en définissant pi = 3.14
    - en important au début du script la librairie math
- donne la différence entre les 2 volumes obtenus

```
>>> import math
>>> math.pi
3.141592653589793
```

Exercice 2

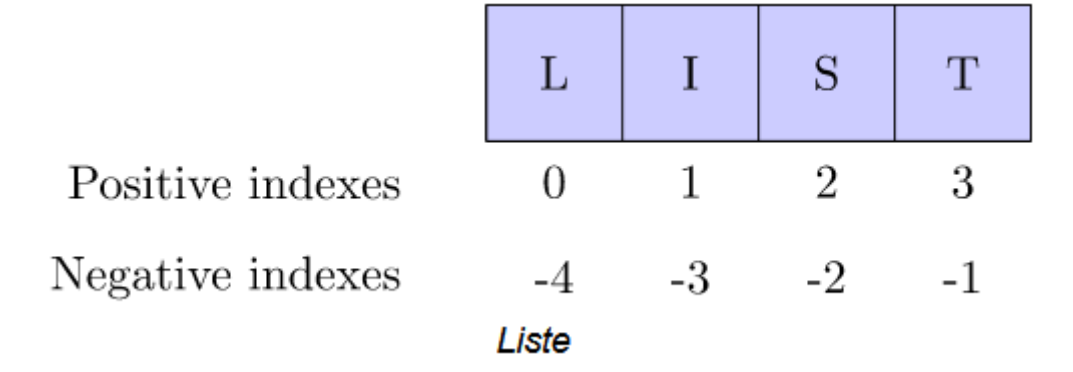
2/ initialiser 2 variables numérique A et B. Afficher le résultat des tests suivants:

- A est supérieur à B
- A est égal à B
- la partie entière de A est supérieure à la partie décimale de B

```
>>> a = 5.2
>>> b = "6.9"
>>> int(a)
5
>>> float(b)
6.9
>>>
```

Types de données
Listes

- Une liste va contenir un ensemble ordonné d'éléments dans une plage mémoire
  - Les éléments peuvent être de type différent
  - Les chaînes de caractères sont des listes



Types de données
Listes

Création, assignation de valeur

```
a = [10,2,3.6,"test"] # on peut donner directement la valeur
b = list() # ou alors déclarer son type
print("a est de type ", type(a), " et de taille ",len(a))
print("b est de type ", type(b), " et de taille ",len(b))
```

```
a est de type  <class 'list'>  et de taille  4
b est de type  <class 'list'>  et de taille  0
```

Types de données
Listes

Création, assignation de valeur

```
a = [10,2,3.6,"test"] # on peut donner directement la valeur
b = list() # ou alors déclarer son type
print("a est de type ", type(a), " et de taille ",len(a))
print("b est de type ", type(b), " et de taille ",len(b))
```

```
a est de type  <class 'list'>  et de taille  4
b est de type  <class 'list'>  et de taille  0
```

```
print(a[0])      #le premier indice est 0
print(a[2])
```

```
10
3.6
```

```
print(a[1:3])      #accéder à une plage de valeur
```

```
[2, 3.6]
```

Types de données
Listes

Ajout, suppression de valeurs

```
a.append(50)     #append permet d'ajouter en
print(a)
```

```
[10, 2, 3.6, 'test', 50]
```

```
a.pop()  #delete the last element
print(a)
a.remove(2)    #delete the element in argument
print(a)
```

```
[10, 2, 3.6, 'test']
[10, 3.6, 'test']
```

Types de données
Listes

Copie

Attention! En dehors des simples variables unitaires, les opérations sur les listes, tableaux et objets renvoient une simple référence mémoire à l'objet. Sauf les chaines de caractères!

Pour effectuer une vraie copie, utiliser l'opérateur copy

```
Bex = ['Pic', 'Saint', 'Loup']
Cex = Bex
print(Cex)
Bex[1] = "A"
Cex[2] = "Glace"

print(Bex)
print(Cex)
```

```
['Pic', 'Saint', 'Loup']
['Pic', 'A', 'Glace']
['Pic', 'A', 'Glace']
```

Types de données
Listes

Copie

Attention! En dehors des simples variables unitaires, les opérations sur les listes, tableaux et objets renvoient une simple référence mémoire à l'objet.
Pour effectuer une vraie copie, utiliser l'opérateur copy

```
Bex = ['Pic', 'Saint', 'Loup']
Cex = Bex
print(Cex)
Bex[1] = "A"
Cex[2] = "Glace"

print(Bex)
print(Cex)
```

```
['Pic', 'Saint', 'Loup']
['Pic', 'A', 'Glace']
['Pic', 'A', 'Glace']
```

```
Bex = ['Pic', 'Saint', 'Loup']
Cex = Bex.copy()
print(Cex)
Bex[1] = "A"
Cex[2] = "Glace"

print(Bex)
print(Cex)
```

```
['Pic', 'Saint', 'Loup']
['Pic', 'A', 'Loup']
['Pic', 'Saint', 'Glace']
```

Types de données
Listes

Opérateurs

Les objets listes ne sont pas des tableaux! Pour cela il existe les objets Array. Les opérations numériques sont traitées comme des opérations sur chaines de caractère

```
print(a)
b = a*2
print(b)
c = a+a+a
print(c)
```

```
[10, 2, 3.6, 'test']
[10, 2, 3.6, 'test', 10, 2, 3.6, 'test']
[10, 2, 3.6, 'test', 10, 2, 3.6, 'test', 10, 2, 3.6, 'test']
```

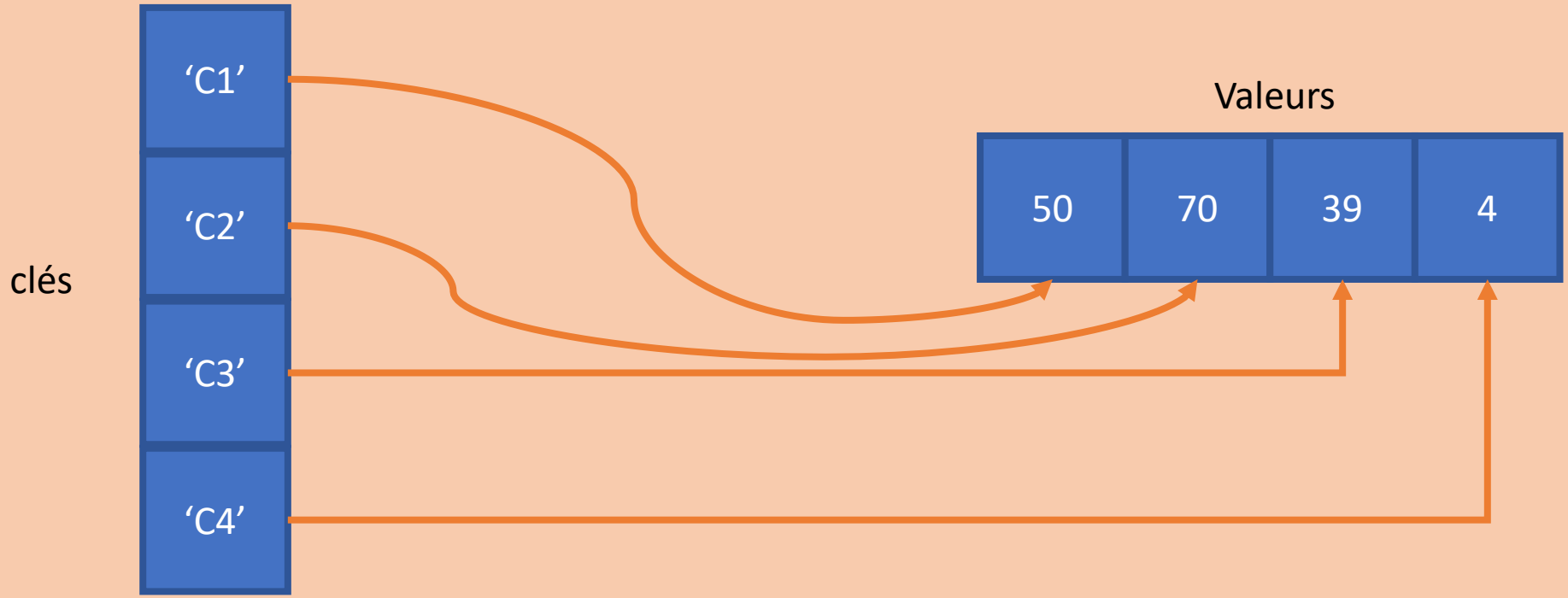
Exercice 3

- Créer une liste avec les valeurs suivantes : 1,10, 4, 50, 4, 37, 2, 4
- Ajouter le chiffre 4 une nouvelle fois
- Utiliser la fonction *dir* pour lister les méthodes disponibles pour la liste créée
    - Effectuer un tri sur la liste
    - Compter combien de 4 sont présents dans la liste

Types de données
Dictionnaires

Un dictionnaire, ou hashtable, est vu comme un tableau dont les valeurs sont accessibles par des étiquettes plutôt que des indices. En python, les étiquettes comme les valeurs peuvent être de types différents.

Les étiquettes sont par nécessité uniques



Types de données
Dictionnaires

Création, assignation de valeur

```
#Pour créer des dictionnaires vides
D1 = dict()
D2 = {}

#Pour créer des dictionnaires avec valeurs initiales
D3 = {"popA":34 , "popB":72 , 1:24, 2:47}

print(D1)
print(D2)
print(D3)
```

```
{}
{}
{'popA': 34, 'popB': 72, 1: 24, 2: 47}
```

Types de données
Dictionnaires

Création, assignation de valeur

```
#Pour ajouter des valeurs
D1['work'] = "UM"
D1['work2'] = 'INSERM'
print(D1)

D1['work'] = "INRA"  #Une étiquette ne peut apparaitre 2 fois
print(D1)
```

```
{'work': 'UM', 'work2': 'INSERM'}
{'work': 'INRA', 'work2': 'INSERM'}
```

```
D2['date'] = '15/01/2007'
D2.update(D1)# concaténer
print(D2)
```

```
{'date': '15/01/2007', 'work2': 'INSERM', 'work': 'INRA'}
```

```
del D2['work'] #supprimer une valeur si elle existe, Erreur sinon
print(D2)
```

```
{'date': '15/01/2007', 'work2': 'INSERM'}
```

Types de données
Dictionnaires

Accéder aux valeurs

```
print(D1['work']) # accéder à une valeur particulière
```

```
INRA
```

```
print(D1.values()) # obtenir les valeurs
```

```
dict_values(['INRA', 'INSERM'])
```

```
print(D1.keys()) # obtenir les clés
```

```
dict_keys(['work', 'work2'])
```

Types de données
Dictionnaires

Test de présence de clés et de valeurs

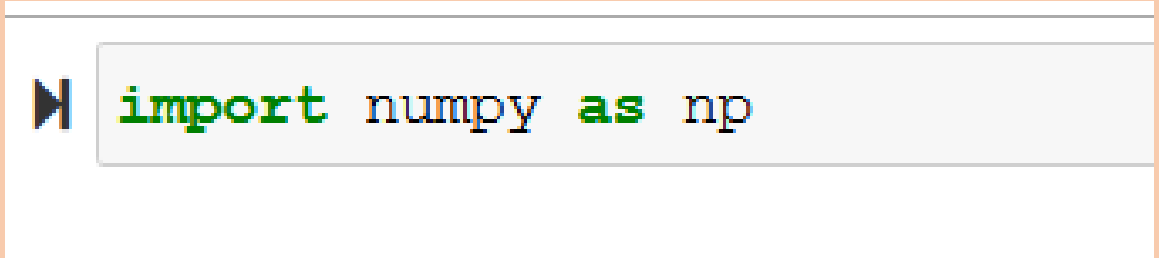
```
print(D3)

print('popA' in D3) # tester l'existence
print('popC' not in D3) # tester la non existence
print(34 in D3.values())
```

```
{'popA': 34, 'popB': 72, 1: 24, 2: 47}
True
True
True
```

Types de données
Numpy Arrays

- Les objets list et dict ne sont pas dédiés aux calculs sur de grands volumes de données, ou tout simplement le calcul matriciel.
- L'objet array qui permet l'usage de tableaux de données sous plusieurs dimensions.
- Librairie Numpy

https://numpy.org/doc/stable/reference/



Types de données
Numpy Arrays

Création

```
print("constructeur zeros")
Azeros = np.zeros( (4,5) ) #crée une matrice 5 lignes, 6 colonnes remplie de 0
print(Azeros)
print("constructeur ones")
Aones = np.ones( (4,3) ) #crée une matrice 4 lignes, 3 colonnes remplie de 1
print(Aones)
print("constructeur empty")
Aempty = np.empty((2,2)) # crée un tableau non initialisé de 2 lignes et 2 colonnes
print(Aempty)
print("constructeur full")
Afull = np.full((2, 3, 4), 14.72)# crée un tableau initialisé de 2x3x4
print(Afull)
print("constructeur identity")
Aidentity = np.identity(3) #crée une matrice identité 3x3
print(Aidentity)
print("Constructeur généraliste array")
Aarray = np.ndarray((1,6))
print(Aarray)
```

```
constructeur zeros
[[0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]]
constructeur ones
[[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]]
```

Types de données Numpy Arrays

Paramètres et typage

```
print(Aones.ndim)
print(Aones.shape)
print(Aones.dtype)
```

```
2
(4, 3)
float64
```

Types de données Numpy Arrays

Paramètres et typage

```
print(Aones.ndim)
print(Aones.shape)
print(Aones.dtype)
```

```
2
(4, 3)
float64
```

```
Aones = np.ones( (4,3) , np.int8) # créer un tableau d'entiers codés sur 8bits
print(Aones.dtype)
```

```
int8
```

```
var_tmp1 = 50.0
var_tmp2 = np.int16(50.6)
print(type(var_tmp1), type(var_tmp2))
print(var_tmp2)
```

```
<class 'float'> <class 'numpy.int16'>
50
```

Types de données Numpy Arrays

Accès aux données

```
Aones[1,0] = 5
Aones[2,2] = 7
print(Aones)
```

```
[[1 1 1]
 [5 1 1]
 [1 1 7]
 [1 1 1]]
```

```
Aones[1:4,:]  #interrogation d'ensembles
```

```
array([[5, 1, 1],
       [1, 1, 7],
       [1, 1, 1]], dtype=int8)
```

```
Aones[:,2:3]  #interrogation d'ensembles
```

```
array([[1],
       [1],
       [7],
       [1]], dtype=int8)
```

Types de données
Numpy Arrays

Opérations

```
print(Aones + 2) #ajout d'un scalaire à toutes les cases
```

```
[[3 3 3]
 [7 3 3]
 [3 3 9]
 [3 3 3]]
```

```
print(Aones * 2) #multiplication par un scalaire à toutes les cases
```

```
[[ 2  2  2]
 [10  2  2]
 [ 2  2 14]
 [ 2  2  2]]
```

```
Aones[1,0] = 5
Aones[2,2] = 7
print(Aones)
```

```
[[1 1 1]
 [5 1 1]
 [1 1 7]
 [1 1 1]]
```

Types de données
Numpy Arrays

Opérations

```
print(Aones + Aones)   #addition case par case
```

```
[[ 2  2  2]
 [10  2  2]
 [ 2  2 14]
 [ 2  2  2]]
```

```
print(Aones[:,0] + Aones[:,2])
```

```
[2 6 8 2]
```

```
print(Aones * Aones)   #multiplication case par case
```

```
[[ 1  1  1]
 [25  1  1]
 [ 1  1 49]
 [ 1  1  1]]
```

```
Aones[1,0] = 5
Aones[2,2] = 7
print(Aones)
```

```
[[1 1 1]
 [5 1 1]
 [1 1 7]
 [1 1 1]]
```

Types de données
Numpy Arrays

Opérations

```
print(Aones + Aones)   #addition case par case
```

```
[[ 2  2  2]
 [10  2  2]
 [ 2  2 14]
 [ 2  2  2]]
```

```
print(Aones[:,0] + Aones[:,2])
```

```
[2 6 8 2]
```

```
print(Aones * Aones)   #multiplication case par case
```

```
[[ 1  1  1]
 [25  1  1]
 [ 1  1 49]
 [ 1  1  1]]
```

```
B = Aones * Aones
C = np.ones(3)
print(B.dot(C))     #produit matriciel
```

```
[ 3. 27. 51.  3.]
```

https://numpy.org/doc/stable/reference/routines.html

Types de données Numpy Arrays

Routines logiques

```
print(np.equal(Aones,1)) # vrai pour les cases à 1
```

```
[[ True  True  True]
 [False  True  True]
 [ True  True False]
 [ True  True  True]]
```

```
np.greater_equal(Aones,2) # vrai pour les cases >=2
```

```
array([[False, False, False],
       [ True, False, False],
       [False, False,  True],
       [False, False, False]])
```

```
Aones[1,0] = 5
Aones[2,2] = 7
print(Aones)
```

```
[[1 1 1]
 [5 1 1]
 [1 1 7]
 [1 1 1]]
```

https://numpy.org/doc/stable/reference/routines.html

Types de données
Numpy Arrays
Routines statistiques

```
print(Aones)
print("\nMean selon dimension 0 : selon les lignes:")
print(Aones.mean(0))
print("\nMean selon dimension 1 : selon les colonnes:")
print(Aones.mean(1))
print("\nMean de l'ensemble des valeurs:")
print(Aones.mean())
#les notations np.mean(Aones,0)  ,  np.mean(Aones,1)  ,  np.mean(Aones) sont également valides
```

```
[[1 1 1]
 [5 1 1]
 [1 1 7]
 [1 1 1]]

Mean selon dimension 0 : selon les lignes:
[2.  1.  2.5]

Mean selon dimension 1 : selon les colonnes:
[1.         2.33333333 3.         1.        ]

Mean de l'ensemble des valeurs:
1.8333333333333333
```

```
Aones[1,0] = 5
Aones[2,2] = 7
print(Aones)
```

```
[[1 1 1]
 [5 1 1]
 [1 1 7]
 [1 1 1]]
```

https://numpy.org/doc/stable/reference/routines.html

Types de données Numpy Arrays

Routines statistiques

```
Aex = np.array([0, 1, 1, 3, 2, 1, 7])

print("\nbincounts:")
print(np.bincount(Aex))
```

```
bincounts:
[1 3 1 1 0 0 0 1]
```

```
print(Aones)
print("\nSum selon dimension 0 : selon les lignes:")
print(Aones.sum(0))
print("\nSum selon dimension 1 : selon les colonnes:")
print(Aones.sum(1))
print("\nSum de l'ensemble des valeurs:")
print(Aones.sum())
```

```
[[1 1 1]
 [5 1 1]
 [1 1 7]
 [1 1 1]]

Sum selon dimension 0 : selon les lignes:
[ 8  4 10]

Sum selon dimension 1 : selon les colonnes:
[3 7 9 3]

Sum de l'ensemble des valeurs:
22
```

Exercice 4

Créer une matrice A de taille 5x5 remplie de 7,
Créer une matrice B de taille 5x3 remplie de 1
Créer une matrice identité C de taille 5

1/ multiplier C par une valeur 5
2/ multiplier A par C, rangée dans une matrice indépendante D
3/ additionner la 5^{e} colonne de D et la 2^{e} de B
4/ donner la valeur moyenne de chaque colonne de D
5/ donner la valeur moyenne des valeurs obtenues à la question 3

Exercice 5

```
import pandas as pd
data = pd.read_csv("https://github.com/dbendet/coursera_machine_learning/raw/master/kc_house_data.csv")
data = data.to_numpy()
```

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	grade	sqft_above	sqft_basement	yr_b
0	7129300520	20141013T000000	221900.0	3	1.00	1180	5650	1.0	0	0	...	7	1180	0	1
1	6414100192	20141209T000000	538000.0	3	2.25	2570	7242	2.0	0	0	...	7	2170	400	1
2	5631500400	20150225T000000	180000.0	2	1.00	770	10000	1.0	0	0	...	6	770	0	1
3	2487200875	20141209T000000	604000.0	4	3.00	1960	5000	1.0	0	0	...	7	1050	910	1
4	1954400510	20150218T000000	510000.0	3	2.00	1680	8080	1.0	0	0	...	8	1680	0	1

1/ Combien de lignes et de colonnes?

2/ Calculer le z-score du prix (3e colonne)

La formule est la suivante: $z = (x - \mu) / \sigma$

3/ Test logique pour identifier les outliers (les 2% des valeurs les plus faibles et les plus fortes)

Les blocks Conditionnelle

si

Sinon si

Sinon

booleen

Structure générale

```
if (condition):
        instruction1
         instruction1
elif :
        instruction 2
         instruction 2
else :
        instruction3
        instruction3
```

Les blocks Conditionnelle

```
x = 10
y = 10
z = 10

# Care with the indent
if ((x==y) & (x==z)):
    print('Equality')
elif ((x <= y) & (y <= z)):
    print('Increasing order')
elif((x >= y) & (y >= z)):
    print('Decreasing order')
else:
    print('No order')
```

```
Equality
```

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

if 'a' in x:
    print('a in list')
else:
    print('a not in list')

if ('z' in x):
    print('z in list')
else:
    print('z not in list')
```

```
a in list
z not in list
```

Les blocks
Portée des variables

```
# Warning if you define some variables inside a block and not in another block
x = 10
y = 20
if(x == y):
    z = 20
else:
    w = 10
# print(z) # would crash if x != y
# print(w) # would crash if x == y
```

Exercice 6

- Ecrire un code capable de dire parmi 2 variables laquelle est la plus grande
- Ecrire un code qui prends 2 matrices 3x3 et qui rends en sortie une matrice 3x1, où chaque ligne est la moyenne par ligne maximale parmi les 2 matrices.

Exercice 7

On se fixe une pression seuil et un volume seuil pour une enceinte :

```
pSeuil = 2.3,
vSeuil = 7.41
```

Selon 2 variables pression et volume courant de l'enceinte, écrire un script qui simule le comportement suivant :

– si le volume et la pression sont supérieurs aux seuils : arrêt immédiat;

– si seule la pression est supérieure à la pression seuil : demander d'augmenter le volume de l'enceinte ;

–si seul le volume est supérieur au volume seuil : demander de diminuer le volume de l'enceinte;

–sinon déclarer que « tout va bien ».

Ce comportement sera implémenté par une alternative multiple

Les blocks
Les boucles

Structure générale

while (condition)**:**
    instruction
    instruction

for v **in** *iterable* **:**
    instruction
    instruction

Les blocks
Les boucles

```
variable = 5
while( variable > 0):
    variable = variable - 1
    print(variable)
```

```
4
3
2
1
0
```

Les blocks
Les boucles

Les iterateurs

- Listes
- Arrays
- Objet iterable (dict.keys())

range(start, stop[,step])

numpy.arange(start, stop[,step])

```
#la fonction range retourne un objet range iterable
for var_ex in range(0,3):
    print(var_ex)

print()
#la fonction numpy arange retourne un numpy array iterable
import numpy as np
for var_ex in np.arange(5,20,4):
    print(var_ex)
```

```
0
1
2

5
9
13
17
```

Exercice 8

1/ Ecrire un programme qui affiche la table de multiplication de 7

2/ Ecrire un programme qui affiche les tables de multiplication de 1 à 9

3/ Ecrire un programme qui crée un tableau 24x12

- la valeur de chaque case prend la valeur suivante:

    si l'indice de ligne est supérieur à l'indice de colonne : 5

    sinon 10

- la valeur de chaque case prend maintenant la valeur suivante

    si la valeur est supérieure à l'indice de ligne : somme des valeurs des cases de la ligne avec des indices de colonne inférieurs

    le log (np.log) de la valeur précédente sinon.

Les blocks
les fonctions

```
def carre(a):
    return a**2

def min(a,b):
    if (a>b):
        return b
    else:
        return a
```

```
a=0
b=5
while(a<8):
    if(min(a,b)==a):
        print("a² = ",carre(a))
    else:
        print("b² = ",carre(b))
        b=b+1
    a = a+1
```

```
a² =  0
a² =  1
a² =  4
a² =  9
a² =  16
a² =  25
b² =  25
b² =  36
```

Exercice 9

```
import pandas as pd
data = pd.read_csv("https://github.com/dbendet/coursera_machine_learning/raw/master/kc_house_data.csv")
data = data.to_numpy()
```

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	grade	sqft_above	sqft_basement	yr_b
0	7129300520	20141013T000000	221900.0	3	1.00	1180	5650	1.0	0	0	...	7	1180	0	1
1	6414100192	20141209T000000	538000.0	3	2.25	2570	7242	2.0	0	0	...	7	2170	400	1
2	5631500400	20150225T000000	180000.0	2	1.00	770	10000	1.0	0	0	...	6	770	0	1
3	2487200875	20141209T000000	604000.0	4	3.00	1960	5000	1.0	0	0	...	7	1050	910	1
4	1954400510	20150218T000000	510000.0	3	2.00	1680	8080	1.0	0	0	...	8	1680	0	1

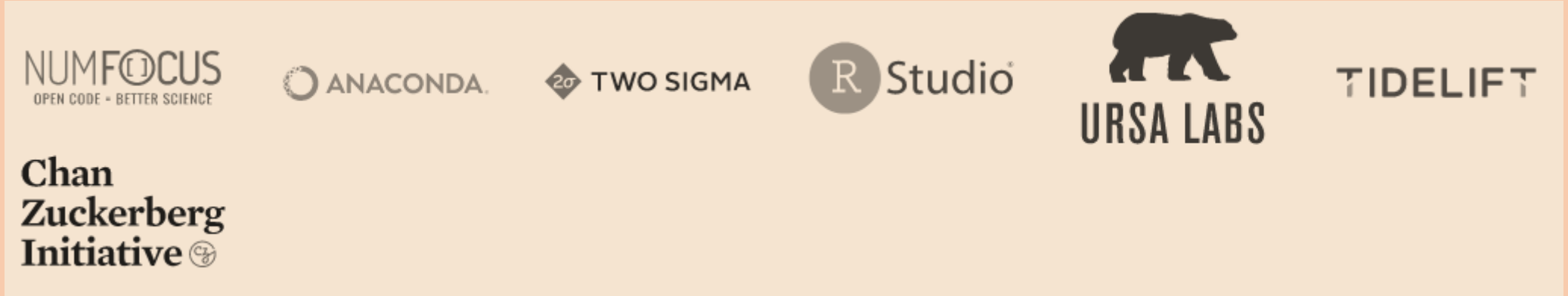
Créer un dictionnaire contenant le score maximal en fonction du nombre de chambres + SdB.

Le score sera donné par une fonction, et donné comme le prix par pieds² habitable.

Structures de données avancées pandas

- Librairie Python spécialisée dans l'analyse des données.
- Librairie ouvertement inspirée de R
  - ⇒ Un objet de type data frame
  - ⇒ Permet de réaliser de nombreuses opérations de filtrage, prétraitements, préalables à la modélisation statistique.
  - ⇒ Intégration des formats CSV, Excel, SQL, JSON, …

https://pandas.pydata.org/



Pandas
Structures

2 types de structures

- Series : 1D array, avec label, type homogène
- DataFrame : 2D, avec labels, type hétérogène

Pandas
Structures - Création

- Series

```
##################
# Création Series
s = pd.Series([1, 3, 5, np.nan, 6, 8])
s
```

```
0    1.0
1    3.0
2    5.0
3    NaN
4    6.0
5    8.0
dtype: float64
```

- Dataframe

```
##################
# Création DataFrame
df = pd.DataFrame({
    'age' :     [ 10, 22, 13, 21, 12, 11, 17],
    'section' : [ 'A', 'B', 'C', 'B', 'B', 'A', 'A'],
    'city' :    [ 'Gurgaon', 'Delhi', 'Mumbai', 'Delhi', 'Mumbai', 'Delhi', 'Mumbai'],
    'gender' :  [ 'M', 'F', 'F', 'M', 'M', 'M', 'F'],
    'favourite_color' : [ 'red', 'blue', 'yellow', 'green', 'black', 'green', 'red'],
    'salary' :  [ 0, 16000, 0, 13560, 600, 0, 750]
})
df.head(6)
```

	age	section	city	gender	favourite_color	salary
0	10	A	Gurgaon	M	red	0
1	22	B	Delhi	F	blue	16000
2	13	C	Mumbai	F	yellow	0
3	21	B	Delhi	M	green	13560
4	12	B	Mumbai	M	black	600
5	11	A	Delhi	M	green	0

Pandas
Structures - Création

- Series

```
##################
# Création Series
s = pd.Series([1, 3, 5, np.nan, 6, 8])
s
```

```
0    1.0
1    3.0
2    5.0
3    NaN
4    6.0
5    8.0
dtype: float64
```

- Dataframe

```
data = pd.read_csv("https://github.com/dbendet/coursera_machine_learning/raw/master/kc_house_data.csv")
data.head()
```

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	grade	sqft_above	sqft_basem
0	7129300520	20141013T000000	221900.0	3	1.00	1180	5650	1.0	0	0	...	7	1180	
1	6414100192	20141209T000000	538000.0	3	2.25	2570	7242	2.0	0	0	...	7	2170	
2	5631500400	20150225T000000	180000.0	2	1.00	770	10000	1.0	0	0	...	6	770	
3	2487200875	20141209T000000	604000.0	4	3.00	1960	5000	1.0	0	0	...	7	1050	
4	1954400510	20150218T000000	510000.0	3	2.00	1680	8080	1.0	0	0	...	8	1680	

5 rows × 21 columns

Pandas
Structures – Voir les données

```
################
# Viewing data
data.head() ## see the top rows
```

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	grade	sqft_above	sqft_basement	yr_built
0	7129300520	20141013T000000	221900.0	3	1.00	1180	5650	1.0	0	0	...	7	1180	0	1955
1	6414100192	20141209T000000	538000.0	3	2.25	2570	7242	2.0	0	0	...	7	2170	400	1951
2	5631500400	20150225T000000	180000.0	2	1.00	770	10000	1.0	0	0	...	6	770	0	1933
3	2487200875	20141209T000000	604000.0	4	3.00	1960	5000	1.0	0	0	...	7	1050	910	1965
4	1954400510	20150218T000000	510000.0	3	2.00	1680	8080	1.0	0	0	...	8	1680	0	1987

5 rows × 21 columns

```
data.tail()  ## see the bottom rows
```

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	grade	sqft_above	sqft_basement	yr_b
21608	263000018	20140521T000000	360000.0	3	2.50	1530	1131	3.0	0	0	...	8	1530	0	2
21609	6600060120	20150223T000000	400000.0	4	2.50	2310	5813	2.0	0	0	...	8	2310	0	2
21610	1523300141	20140623T000000	402101.0	2	0.75	1020	1350	2.0	0	0	...	7	1020	0	2
21611	291310100	20150116T000000	400000.0	3	2.50	1600	2388	2.0	0	0	...	8	1600	0	2
21612	1523300157	20141015T000000	325000.0	2	0.75	1020	1076	2.0	0	0	...	7	1020	0	2

5 rows × 21 columns

Pandas
Structures – Voir les données

```
df.index  ## see the index
```

```
RangeIndex(start=0, stop=7, step=1)
```

```
df.columns  ## see the column names
```

```
Index(['age', 'section', 'city', 'gender', 'favourite_color', 'salary'], dtype='object')
```

Pandas
Structures – Voir les données

```
df.describe()
```

	age	salary
count	7.000000	7.000000
mean	15.142857	4415.714286
std	4.879500	7121.638054
min	10.000000	0.000000
25%	11.500000	0.000000
50%	13.000000	600.000000
75%	19.000000	7155.000000
max	22.000000	16000.000000

```
df.sort_values(by = "age")
```

	age	section	city	gender	favourite_color	salary
0	10	A	Gurgaon	M	red	0
5	11	A	Delhi	M	green	0
4	12	B	Mumbai	M	black	600
2	13	C	Mumbai	F	yellow	0
6	17	A	Mumbai	F	red	750
3	21	B	Delhi	M	green	13560
1	22	B	Delhi	F	blue	16000

Pandas
Accéder aux données, sélection de colonnes

```
#### Accéder aux données
# sélectionner une colonne
df[["salary","age"]]
```

	salary	age
0	0	10
1	16000	22
2	0	13
3	13560	21
4	600	12
5	0	11
6	750	17

```
#### Accéder aux données
# sélectionner une colonne
df["salary"]
```

```
0        0
1    16000
2        0
3    13560
4      600
5        0
6      750
Name: salary, dtype: int64
```

```
# sélectionner une un sensemble de lignes
df[2:5]
```

Pandas
Accéder aux données, sélection de lignes

```
# sélectionner une un sensemble de lignes
df[2:5]
```

	age	section	city	gender	favourite_color	salary
2	13	C	Mumbai	F	yellow	0
3	21	B	Delhi	M	green	13560
4	12	B	Mumbai	M	black	600

Pandas
Accéder aux données, sélection conditionnelle simple

```
df[df['salary']>0]
```

	age	section	city	gender	favourite_color	salary
1	22	B	Delhi	F	blue	16000
3	21	B	Delhi	M	green	13560
4	12	B	Mumbai	M	black	600
6	17	A	Mumbai	F	red	750

```
df[(df['salary']>0) & (df['age']>18)]
```

	age	section	city	gender	favourite_color	salary
1	22	B	Delhi	F	blue	16000
3	21	B	Delhi	M	green	13560

Pandas
Accéder aux données, opérateur isin

```
df[df['favourite_color'].isin(['blue','red','green'])]
```

	age	section	city	gender	favourite_color	salary
0	10	A	Gurgaon	M	red	0
1	22	B	Delhi	F	blue	16000
3	21	B	Delhi	M	green	13560
5	11	A	Delhi	M	green	0
6	17	A	Mumbai	F	red	750

Pandas
Ajout de colonne

```
df['salary_upgrade'] = df['salary']*1.3+200
df
```

	age	section	city	gender	favourite_color	salary	salary_upgrade
0	10	A	Gurgaon	M	red	0	200.0
1	22	B	Delhi	F	blue	16000	21000.0
2	13	C	Mumbai	F	yellow	0	200.0
3	21	B	Delhi	M	green	13560	17828.0
4	12	B	Mumbai	M	black	600	980.0
5	11	A	Delhi	M	green	0	200.0
6	17	A	Mumbai	F	red	750	1175.0

Pandas
Opérateurs loc & iloc

- **loc** est basé sur les labels.
  - Il sera utilisé dans le cas où on spécifie les noms de lignes et les noms de colonnes auxquelles nous vons accéder
- **iloc** est basé sur les index entiers
  - Il sera utilisé si on spécifie le numéro de la ligne et/ou le numéro de la colonne

Pandas
Opérateurs loc

Trouver toutes les lignes selon des conditions sur une colonne

```
df.loc[df.age>20]
```

Run this cell

	age	section	city	gender	favourite_color	salary
1	22	B	Delhi	F	blue	16000
3	21	B	Delhi	M	green	13560
6	17	A	Mumbai	F	red	750

```
df.loc[(df.age>20) | (df.salary>500)]
```

	age	section	city	gender	favourite_color	salary
1	22	B	Delhi	F	blue	16000
3	21	B	Delhi	M	green	13560
4	12	B	Mumbai	M	black	600
6	17	A	Mumbai	F	red	750

Pandas
Opérateurs loc

Trouver un ensemble de lignes

```
df.loc[1:3]
```

	age	section	city	gender	favourite_color	salary
1	22	B	Delhi	F	blue	16000
2	13	C	Mumbai	F	yellow	0
3	21	B	Delhi	M	green	13560

ATTENTION!!!! Il ne s'agit pas des lignes 1 à 3
Il s'agit des indexs!
Si indexs non chiffrés, alors pas possible d'utiliser cette notation.
Si les indexs sont dans le désordre (suite à un sort par exemple), 1:3 ne fonctionne plus.

Pandas
Opérateurs loc

Trouver un ensemble de lignes

```
df = df.sort_values(by='age')
df
```

	age	section	city	gender	favourite_color	salary
0	10	A	Gurgaon	M	red	0
5	11	A	Delhi	M	green	0
4	12	B	Mumbai	M	black	600
2	13	NM	Mumbai	F	yellow	14250
6	17	NM	Mumbai	F	red	14250
3	21	B	Delhi	M	green	13560
1	22	NM	Delhi	F	blue	14250

```
df.loc[1]
```

```
age                   22
section               NM
city               Delhi
gender                F
favourite_color     blue
salary             14250
Name: 1, dtype: object
```

ATTENTION!!!! Il ne s'agit pas des lignes 1 à 3
Il s'agit des indexs!
Si indexs non chiffrés, alors pas possible d'utiliser cette notation.
Si les indexs sont dans le désordre (suite à un sort par exemple), 1:3 ne fonctionne plus.

Pandas
Opérateurs loc

Sélectionner des colonnes

```
df.loc[df.age>12,['city', 'gender']]
```

	city	gender
1	Delhi	F
2	Mumbai	F
3	Delhi	M
6	Mumbai	F

Pandas
Opérateurs loc

Update des valeurs

```
#update 1 column
df.loc[(df.age> 12)&(df.gender=='F'),['section']] = "NM"
df
```

	age	section	city	gender	favourite_color	salary
0	10	A	Gurgaon	M	red	0
1	22	NM	Delhi	F	blue	14250
2	13	NM	Mumbai	F	yellow	14250
3	21	B	Delhi	M	green	13560
4	12	B	Mumbai	M	black	600
5	11	A	Delhi	M	green	0
6	17	NM	Mumbai	F	red	14250

Pandas
Opérateurs loc

Update des valeurs

```
#update 2 columns
df.loc[(df.age> 12)&(df.gender=='F'),['section','salary']] = ["NM",14250]
df
```

	age	section	city	gender	favourite_color	salary
0	10	A	Gurgaon	M	red	0
1	22	NM	Delhi	F	blue	14250
2	13	NM	Mumbai	F	yellow	14250
3	21	B	Delhi	M	green	13560
4	12	B	Mumbai	M	black	600
5	11	A	Delhi	M	green	0
6	17	NM	Mumbai	F	red	14250

Exercice 10

- A partir du code de l'exercice 5, sans transformer le dataframe en matrice numpy
  - Ajouter une colonne price2, égale à 80% du prix.
  - Ajouter à price2
    - 300$ si il y a plus de 2 chambres
    - 500$ si il y a plus de 2 chambres et 2 salles de bain
- Il y a un respect de la notations de numpy
  - Essayer la méthode mean(), median(), sum() sur le dataframe
  - Calculer le Z-score de price