Mémento Python 3 pour le calcul scientifique

Version AM–1.0

Forme inspirée initialement du mémento de Laurent Pointal, disponible ici : https://perso.limsi.fr/pointal/python:memento

Cette version sur l'E.N.T. Arts et Métiers :
https://savoir.ensam.eu/moodle/course/view.php?id=1428

Aide
F1

`dir(nom)` *liste des noms des méthodes et attributs de* ***nom***
`help(nom)` *aide sur l'objet* ***nom***
`help("nom_module.nom")` *aide sur l'objet* `nom` *du module* `nom_module`

Types de base

Entier, décimal, complexe, booléen, rien

objets non mutables

`int` `783` `0` *zéro* `-192` `0b010` *binaire* `0o642` *octal* `0xF3` *hexadécimal*

`float` `9.23` `0.0` `-1.7e-6` (-1,7×10^{-6})

`complex` `1j` `0j` `2+3j` `1.3-3.5e2j`

`bool` `True` `False`

`NoneType` `None` (une seule valeur : « rien »)

Identificateurs

Noms d'objets, de fonctions, de modules, de classes, etc.

`a`...`zA`...`Z_` suivi de `a`...`zA`...`Z_0`...`9`

- accents possibles mais à éviter
- mots clés du langage interdits
- distinction casse min/MAJ

☺ `a toto x7 y_max BigOne`
☹ ~~`8y and for`~~

Affectation/nommage

Symbole : `=`

affectation ⇔ **association** d'un *nom* à un objet

`nom_objet = <expression>`

1) *évaluation de l'expression de droite pour créer un objet*
2) *nommage de l'objet créé*

```
x = 1.2 + 8 + sin(y)
```

Affectations multiples

<n noms> `=` *<itérable de taille n>*

```
u,v,w = 1j,"a",None
```

`a,b = b,a` *échange de valeurs*

Affectations combinée avec une opération ✧

`x ✧= c` *équivaut à :* `x = x✧c`

Suppression d'un nom

`del x` *l'objet associé disparaît seulement s'il n'a plus de nom, par le mécanisme du « ramasse-miettes »*

Conteneurs : opérations génériques

`len(c)` `min(c)` `max(c)` `sum(c)`
nom `in c` → booléen, test de présence dans `c` d'un élément identique (*comparaison* `==`) à ***nom***
nom `not in c` → *booléen, test d'absence*
`c1 + c2` → *concaténation*
`c * 5` → *5 répétitions* (`c+c+c+c+c`)
`c.index(`***nom***`)` → *position du premier élément identique à* ***nom***
`c.index(`***nom , idx***`)` → *position du premier élément identique à* ***nom*** *à partir de la position* ***idx***
`c.count(`***nom***`)` → *nombre d'occurrences*

Opérations sur listes

modification « en place » de la liste `L` *originale ces méthodes ne renvoient rien en général*

`L.append(`***nom***`)` *ajout d'un élément à la fin*
`L.extend(`***itérable***`)` *ajout d'un itérable converti en liste à la fin*
`L.insert(`***idx , nom***`)` *insertion d'un élément à la position* ***idx***
`L.remove(`***nom***`)` *suppression du premier élément identique (comparaison* `==`*) à* ***nom***
`L.pop()` *renvoie et supprime le dernier élément*
`L.pop(`***idx***`)` *renvoie et supprime l'élément à la position* ***idx***
`L.sort()` *ordonne la liste (ordre croissant)*
`L.sort(reverse=True)` *ordonne la liste par ordre décroissant*
`L.reverse()` *renversement de la liste*
`L.clear()` *vide la liste*

Objets itérables

- **Conteneurs numérotés** (*listes, tuples, chaînes de caractères*)

`list`	`[1,5,9]`	`["abc"]`	`[]`	`["x",-1j,["a",False]]`
`tuple`	`(1,5,9)`	`("abc",)`	`()`	`11,"y",[2-1j,True]`
`str`	`"abc"`	`"z"` *Singleton*	`""` *Objet vide*	

Conteneurs hétérogènes

Objets non mutables

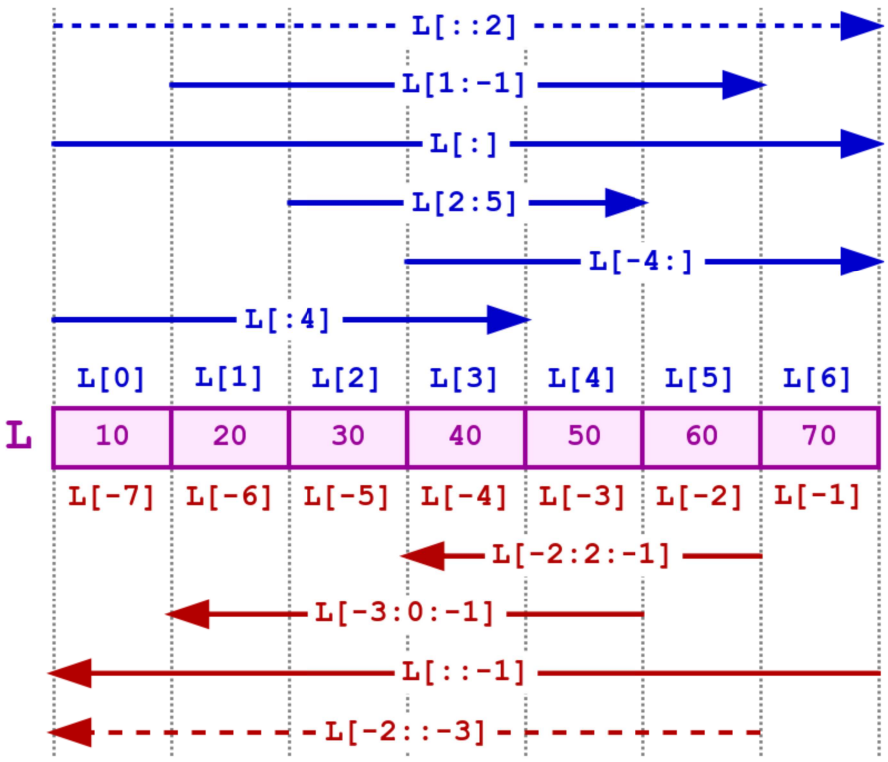
expression juste avec des virgules → `tuple`

Nombre d'éléments
`len(`*objet*`)` donne : `3` `1` `0` `3`

- **Itérateurs** (*objets destinés à être parcourus par* `in`)

`range(n)` : pour parcourir les *n* premiers entiers naturels, de 0 à *n*-1 inclus.
`range(n,m)` : pour parcourir les entiers naturels de *n* inclus à *m* exclus par pas de 1.
`range(n,m,p)` : pour parcourir les entiers naturels de *n* inclus à *m* exclus par pas de *p*.
`reversed(`*itérable*`)` : pour parcourir un objet itérable à l'envers.
`enumerate(`*itérable*`)` : pour parcourir un objet itérable en ayant accès à la numérotation.
`zip(`*itérable1 , itérable2 , ...*`)` : pour parcourir en parallèle plusieurs objets itérables.

Parcours de conteneurs numérotés



index à partir de 0

- Accès à chaque **élément** par `L[`***index***`]`

`L[0]` → `10` ⇒ *le premier*
`L[1]` → `20` ⇒ *le deuxième*
`L[-1]` → `70` ⇒ *le dernier*
`L[-2]` → `60` ⇒ *l'avant-dernier*

- Accès à une **partie** par `L[`***début inclus : fin exclue : pas***`]`

`L[2:5]` → `[30,40,50]` ⇒ *indices 2,3 et 4*
`L[:4]` → `[10,20,30,40]` ⇒ *les 4 premiers*
`L[-4:]` → `[40,50,60,70]` ⇒ *les 4 derniers*
`L[::2]` → `[10,30,50,70]` ⇒ *de 2 en 2*
`L[:]` *tous : copie superficielle du conteneur*
`L[::-1]` *tous, de droite à gauche*
`L[-2::-3]` → `[60,30]` ⇒ *de -3 en -3 en partant de l'avant-dernier*

Sur les listes *(conteneurs mutables),* ***suppression*** *d'un élément ou d'une partie par* `del`*, et* ***remplacement*** *par* `=`

`del L[4]` effet sur la liste `L` similaire à `L.pop(4)` → `L` *devient* `[10,20,30,40,60,70]`
`del L[1::2]` suppression des éléments d'indices impairs → `L` *devient* `[10,30,50,70]`
`L[4] = 99` → `L` *devient* `[10,20,30,40,99,60,70]`
`L[1::2] = "abc"` itérable ayant le même nombre d'éléments que la partie à remplacer, sauf si le pas vaut 1 → `L` *devient* `[10,"a",30,"b",50,"c",70]`
`L[1:-1] = range(2)` → `L` *devient* `[10,0,1,70]`

Chaînes de caractères

Caractères spéciaux :
`"\n"` *retour à la ligne*
`"\t"` *tabulation*
`"\\"` *« backslash \ »*
`"\""` *ou* `'"'` *guillemet "*
`"'"` *ou* `'\''` *apostrophe '*

Exemple :
```
ch = "X\tY\tZ\n1\t2\t3"
print(ch)  affiche :  X   Y   Z
                      1   2   3
print(repr(ch))  affiche :
'X\tY\tZ\n1\t2\t3'
```

`r"dossier\sd\nom.py"` → `'dossier\\sd\\nom.py'`
Le préfixe `r` *signifie ``raw string'' (tous les caractères sont considérés comme de vrais caractères)*

Méthodes sur les chaînes

Une chaîne n'est pas modifiable ; ces méthodes renvoient en général une nouvelle chaîne ou un autre objet

```
"nomfic.txt".replace(".txt",".png") → 'nomfic.png'
"b-a-ba".replace("a","eu") → 'b-eu-beu'   remplacement de toutes les occurrences
" \tUne phrase.\n ".strip() → 'Une phrase.'   nettoyage début et fin
"des  mots\tespacés".split() → ['des','mots','espacés']
"1.2,4e-2,-8.2,2.3".split(",") → ['1.2','4e-2','-8.2','2.3']
" ; ".join(["1.2","4e-2","-8.2","2.3"]) → '1.2 ; 4e-2 ; -8.2 ; 2.3'
ch.lower() minuscules, ch.upper() majuscules, ch.title(), ch.swapcase()
```
Recherche de position : `find` *similaire à* `index` *mais renvoie* `-1` *en cas d'absence, au lieu de soulever une erreur*
```
"image.png".endswith(".txt") → False
"essai001.txt".startswith("essai") → True
```

Formatage

La méthode `format` ***sur une chaîne contenant*** `"{<numéro>:<format>}"` ***(accolades)***

```
"{} ~ {}".format("pi",3.14) → 'pi ~ 3.14'          ordre et formats par défaut
"{1:} -> {0:}{1:}".format(3,"B") → 'B -> 3B'       ordre, répétition
"essai_{:04d}.txt".format(12) → 'essai_0012.txt'   entier, 4 chiffres, complété par des 0
"L : {:.3f} m".format(0.01) → 'L : 0.010 m'         décimal, 3 chiffres après la virgule
"m : {:.2e} kg".format(0.012) → 'm : 1.20e-02 kg'   scientifique, 2 chiffres après la virgule
```

Mémento Python 3 pour le calcul scientifique

ARTS ET MÉTIERS ParisTech

Blocs d'instructions

```
instruction parente :
    bloc d'instructions 1…
    ⋮
    instruction parente :
        bloc d'instructions 2…
        ⋮
instruction suivant le bloc 1
⋮
```

(indentation)

Symbole `:` puis indentation (4 espaces en général)

Instruction conditionnelle

```
if booléen1 :
    bloc d'instructions 1…
    ⋮
elif booléen2 :
    bloc d'instructions 2…
    ⋮
else :
    dernier bloc…
    ⋮
```

Blocs `else` et `elif` facultatifs.

`if`/`elif` *x* : si *x* n'est pas un booléen équivaut en Python à `if`/`elif bool(`*x*`)` : (voir conversions).

Définition de fonction

Une fonction fait des actions et renvoie un ou plusieurs objets, ou ne renvoie rien.

```
def nom_fct(x,y,z=0,a=None) :
    bloc d'instructions…
    ⋮
    if a is None :
        ⋮
    else :
        ⋮
    return r0,r1,…,rk
```

`x` et `y` : *arguments positionnels, obligatoires*
`z` et `a` : *arguments optionnels avec des valeurs par défaut, nommés*

Plusieurs `return` *possibles (interruptions)*
Une absence de `return` signifie qu'à la fin, `return None` *(rien n'est renvoyé)*

Appel(s) de la fonction

Autant de noms que d'objets renvoyés

```
a0,a1,…,ak = nom_fct(-1,2)
b0,b1,…,bk = nom_fct(3.2,-1.5,a="spline")
```

`True`/`False` Logique booléenne

- ***Opérations booléennes***
  - `not A` « *non A* »
  - `A and B` « *A et B* »
  - `A or B` « *A ou B* »
  - `(not A) and (B or C)` *exemple*
- ***Opérateurs renvoyant un booléen***
  - *nom1* `is` *nom2* *2 noms du même objet ?*
  - *nom1* `==` *nom2* *valeurs identiques ?*
  - Autres comparateurs : `<` `>` `<=` `>=` `!=` (≠)
  - *nom_objet* `in` *nom_iterable* — *l'itérable* ***nom_iterable*** *contient-il un objet de valeur identique à celle de* ***nom_objet*** *?*

Conversions

`bool(`*x*`)` → `False` *pour* *x* : `None`, `0` (`int`), `0.0` (`float`), `0j` (`complex`), *itérable vide*
→ `True` *pour* *x* : *valeur numérique non nulle, itérable non vide*

`int("15")` → `15`
`int("15",7)` → `12` *(base 7)*
`int(-15.56)` → `-15` *(troncature)*
`round(-15.56)` → `-16` *(arrondi)*
`float(-15)` → `-15.0`
`float("-2e-3")` → `-0.002`
`complex("2-3j")` → `(2-3j)`
`complex(2,-3)` → `(2-3j)`
`list(`*x*`)` *Conversion d'un itérable en liste* exemple : `list(range(12,-1,-1))`
`sorted(`*x*`)` *Conversion d'un itérable en liste ordonnée (ordre croissant)*
`sorted(x,reverse=True)` *Conversion d'un itérable en liste ordonnée (ordre décroissant)*
`tuple(`*x*`)` *Conversion en tuple*
`"{}".format(`*x*`)` *Conversion en chaîne de caractères*
`ord("A")` → `65` ; `chr(65)` → `'A'`

Mathématiques

- ***Opérations***
  - `+` `-` `*` `/`
  - `**` *puissance* `2**10` → `1024`
  - `//` *quotient de la division euclidienne*
  - `%` *reste de la division euclidienne*
- ***Fonctions intrinsèques***
  - `abs(`*x*`)` *valeur absolue / module*
  - `round(`*x*`,`*n*`)` *arrondi du* `float` *x à* *n* *chiffres après la virgule*
  - `pow(a,b)` *équivalent à* `a**b`
  - `pow(`*a*`,`*b*`,`*p*`)` *reste de la division euclidienne de* a^b *par* *p*
  - *z*`.real` → *partie réelle de z*
  - *z*`.imag` → *partie imaginaire de z*
  - *z*`.conjugate()` → *conjugué de z*

`sys`

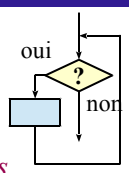
```
import sys
```

`sys.path` → *liste des chemins des dossiers contenant des modules Python*
`sys.path.append(`*chemin*`)` *Ajout du* ***chemin absolu*** *d'un dossier contenant des modules*
`sys.platform` → *nom du système d'exploitation*

Boucle conditionnelle

Bloc d'instructions répété tant que ***condition*** *est vraie*

```
while condition :
    instructions…
    ⋮
    (valeurs impliquées dans condition modifiées)
    ⋮
```



Attention aux boucles sans fin !

Contrôle de boucle

`break` *sortie immédiate*
`continue` *itération suivante*

Exemple

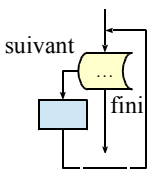
```
from random import randint
somme,nombre = 0,0
while somme < 100 :
    nombre += 1
    somme += randint(1,10)
print(nombre,";",somme)
```

Le nombre d'itérations n'est pas connu à l'avance

Boucle par itérations

Bloc d'instructions répété ***pour chaque*** *élément de l'****itérable****, désigné par* ***nom***

```
for nom in itérable :
    instructions…
    ⋮
```



Variantes avec parcours en parallèle

```
for a,b in itérable :
    bloc d'instructions
```
Itérations sur des couples

```
for numéro,nom in enumerate(itérable) :
    bloc d'instructions
```
Numérotation en parallèle, à partir de `0`

```
for numéro,nom in enumerate(itérable,d) :
    bloc d'instructions
```
Numérotation en parallèle, à partir de *d*

```
for e1,e2,… in zip(itérable1,itérable2,…) :
    bloc d'instructions
```
Parcours en parallèle de plusieurs itérables ; s'arrête dès qu'on arrive à la fin de l'un d'entre eux

Liste en compréhension

- ***Inconditionnelle / conditionnelle***

```
L = [ f(e) for e in itérable ]
L = [ f(e) for e in itérable if b(e)]
```

Fichiers texte

N'est indiquée ici que l'ouverture avec fermeture automatique, au format normalisé UTF-8.

Le « ***chemin*** » d'un fichier est une chaîne de caractères (voir module `os` ci-dessous)

- ***Lecture intégrale d'un seul bloc***

```
with open(chemin,"r",encoding="utf8") as f:
    texte = f.read()
```

- ***Lecture ligne par ligne***

```
with open(chemin,"r",encoding="utf8") as f:
    lignes = f.readlines()
(Nettoyage éventuel des débuts et fins de lignes)
lignes = [c.strip() for c in lignes]
```

- ***Écriture dans un fichier***

```
with open(chemin,"w",encoding="utf8") as f:
    f.write(début)…
    ⋮
    f.write(suite)…
    ⋮
    f.write(fin)
```

Gestion basique d'exceptions

```
try :
    bloc à essayer
except :
    bloc exécuté en cas d'erreur
```

Affichage

```
x,y = -1.2,0.3
print("Pt",2,"(",x,",",y+4,")")
```
→ `Pt 2 = ( -1.2 , 4.3 )`

Un espace est inséré à la place de chaque virgule séparant deux objets consécutifs. Pour mieux maîtriser l'affichage, utiliser la méthode de formatage `str.format`

Saisie

```
s = input("Choix ? ")
```

`input` *renvoie toujours une* ***chaîne de caractères*** *; la convertir si besoin vers le type désiré*

Importation de modules

Module `mon_mod` ⇔ Fichier `mon_mod.py`

- ***Importation d'objets par leurs noms***

```
from mon_mod import nom1,nom2
```

- ***Importation avec renommage***

```
from mon_mod import nom1 as n1
```

- ***Importation du module complet***

```
import mon_mod
    ⋮
… mon_mod.nom1 …
```

- ***Importation du module complet avec renommage***

```
import mon_mod as mm
    ⋮
… mm.nom1 …
```

Programme utilisé comme module

- ***Bloc-Test*** *(non lu en cas d'utilisation du programme* `mon_mod.py` *en tant que module)*

```
if __name__ == "__main__" :
    Bloc d'instructions
    ⋮
```

Quelques modules internes de Python (The Python Standard Library)

`os`

```
import os
```

`os.getcwd()` → ***Chemin absolu*** *du « répertoire de travail »* (working directory), *à partir duquel on peut donner des* ***chemins relatifs****.*

Chemin absolu : chaîne commençant par une lettre majuscule suivie de `":"` *(Windows), ou par* `"/"` *(autre)*

Chemin relatif par rapport au répertoire de travail ***wd*** *:*
nom de fichier ⇔ fichier dans ***wd***
`"."` ⇔ ***wd*** ; `".."` ⇔ *père de* ***wd***
`"../.."` ⇔ *grand-père de* ***wd***
`"sous-dossier/image.png"`

Le séparateur `"/"` *fonctionne pour tous les systèmes, au contraire du* `\` (`"\\"`)

`os.listdir(`*chemin*`)` → *liste des sous-dossiers et fichiers du dossier désigné par* ***chemin****.*

`os.path.isfile(`*chemin*`)` → Booléen : est-ce un fichier ?
`os.path.isdir(`*chemin*`)` → Booléen : est-ce un dossier ?

```
for sdp,Lsd,Lnf in os.walk(chemin) :
```
Parcourt récursivement chaque sous-dossier, de chemin relatif `sdp`*, dont la liste des sous-dossiers est* `Lsd` *et celle des fichiers est* `Lnf`

`time`

```
from time import time
debut = time()
⋮ (instructions)
duree = time() - debut
```

Évaluation d'une durée d'exécution, en secondes

Mémento Python 3 pour le calcul scientifique

ARTS ET MÉTIERS ParisTech

Aide numpy/scipy

`np.info(`*nom_de_la_fonction*`)`

Fonctions mathématiques

```
import numpy as np
```

En calcul scientifique, il est préférable d'utiliser les fonctions de **numpy**, *au lieu de celles des modules basiques* **math** *et* **cmath**, *puisque* ***les fonctions de numpy sont vectorisées*** *: elle s'appliquent aussi bien à des scalaires (*`float`, `complex`*) qu'à des vecteurs, matrices, tableaux, avec des durées de calculs minimisées.*

`np.pi, np.e` → *Constantes* π *et e*

`np.abs, np.sqrt, np.exp, np.log, np.log10, np.log2` → `abs` *, racine carrée, exponentielle, logarithmes népérien, décimal, en base 2*

`np.cos, np.sin, np.tan` → *Fonctions trigonométriques (angles en radians)*

`np.degrees, np.radians` → *Conversion radian→degré, degré→radian*

`np.arccos, np.arcsin` → *Fonctions trigonométriques réciproques*

`np.arctan2(y,x)` → *Angle dans*]–π,π]

`np.cosh, np.sinh, np.tanh` *(trigonométrie hyperbolique)*

`np.arcsinh, np.arccosh, np.arctanh`

Tableaux numpy.ndarray : généralités

Un tableau **T** *de type* `numpy.ndarray` *(« n-dimensional array »)* *est un conteneur homogène dont les valeurs sont stockées en mémoire de façon séquentielle.*

`T.ndim` → *« dimension* ***d*** *» = nombre d'indices (1 pour un vecteur, 2 pour une matrice)*

`T.shape` → *« forme » = plages de variation des indices, regroupées en* `tuple` $(n_0, n_1, \dots, n_{d-1})$ *: le premier indice varie de* `0` *à* n_0-1 *, le deuxième de* `0` *à* n_1-1 *, etc.*

`T.size` → nombre d'éléments, valant $n_0 \times n_1 \times \cdots \times n_{d-1}$

`T.dtype` → *type des données contenues dans le tableau (*`np.bool, nb.int32, np.uint8, np.float, np.complex, np.unicode`*, etc.)*

générateurs

shp *est la forme du tableau créé,* ***data_type*** *le type de données contenues dans le tableau (*`np.float` *si l'option* `dtype` *n'est pas utilisée)*

`T = np.empty(`*shp*`,dtype=`*data_type*`)` → *pas d'initialisation*

`T = np.zeros(`*shp*`,dtype=`*data_type*`)` → *tout à* `0`/`False`

`T = np.ones(`*shp*`,dtype=`*data_type*`)` → *tout à* `1`/`True`

- ***Tableaux de même forme que*** **T** *(même type de données que* **T** *si ce n'est pas spécifié) :*

`S = np.empty_like(T,dtype=`*data_type*`)`

`S = np.zeros_like(T,dtype=`*data_type*`)`

`S = np.ones_like(T,dtype=`*data_type*`)`

Vecteurs

- ***Un vecteur*** **V** ***est un tableau à un seul indice***
- *Comme pour les listes,* `V[`*i*`]` *est le (*$i+1$*)*^{ème} *coefficient, et l'on peut extraire des sous-vecteurs par :* `V[:2]`, `V[-3:]`, `V[::-1]`, etc.

Si `c` est un nombre, les opérations `c*V, V/c, V+c, V-c, V//c, V%c, V**c` *se font sur chaque coefficient*

Si `U` est un vecteur de même dimension que `V`, les opérations `U+V, U-V, U*V, U/V, U//V, U%V, U**V` *sont des* ***opérations terme à terme***

- ***Produit scalaire*** **:** `U.dot(V)` *ou* `np.dot(U,V)` *ou* `U@V`

générateurs

`np.linspace(`*a*`,`*b*`,`*n*`)` → ***n*** *valeurs régulièrement espacées de* ***a*** *à* ***b*** *(bornes incluses)*

`np.arange(`x_{min}`,`x_{max}`,`*dx*`)` → *de* x_{min} *inclus à* x_{max} ***exclu*** *par pas de* ***dx***

Statistiques

Sans l'option `axis`, *un tableau est considéré comme une simple séquence de valeurs*

`T.max(), T.min(), T.sum()`

`T.argmax(), T.argmin()` *indices séquentiels des extremums*

`T.sum(axis=`*d*`)` → *sommes sur le (*d*-1)-ème indice*

`T.mean(),T.std(),T.std(ddof=1)` *moyenne, écart-type*

`V = np.unique(T)` *valeurs distinctes, sans ou avec les effectifs*

`V,N = np.unique(T,return_counts=True)`

`np.cov(T), np.corrcoef(T)` *matrices de* ***covariance*** *et de* ***corrélation*** *;* `T` *est un tableau* $k \times n$ *qui représente* n *répétitions du tirage d'un vecteur de dimension* k *; ces matrices sont* $k \times k$.

Modules random et numpy.random — Tirages pseudo-aléatoires

```
import random
```

`random.random()` → *Valeur flottante dans l'intervalle* [0,1[*(loi uniforme)*

`random.randint(`*a*`,`*b*`)` → *Valeur entière entre* ***a*** *inclus et* ***b*** *inclus (équiprobabilité)*

`random.choice(`*L*`)` → *Un élément de la liste* ***L*** *(équiprobabilité)*

`random.shuffle(`*L*`)` → `None`, *mélange la liste* ***L*** *« **en place** »*

```
import numpy.random as rd
```

`rd.rand(`$n_0, \dots, n_{d-1}$`)` → *Tableau de forme* $(n_0, \dots, n_{d-1})$*, de flottants dans l'intervalle* [0,1[*(loi uniforme)*

`rd.randint(`*a*`,`*b*`,`*shp*`)` → Tableau de forme ***shp***, d'entiers entre ***a*** inclus et ***b*** exclu (équiprobabilité)

`rd.randint(`*n*`,size=`*d*`)` → Vecteur de dimension ***d***, d'entiers entre `0` et ***n*** `-1` (équiprobabilité)

`rd.choice(`*Omega*`,`*n*`,p=`*probas*`)` → *Tirage avec remise d'un échantillon de taille* ***n*** *dans* ***Omega***, *avec les probabilités* ***probas***

`rd.choice(`*Omega*`,`*n*`,replace=False)` → *Tirage sans remise d'un échantillon de taille* ***n*** *dans* ***Omega*** *(équiprobabilité)*

`rd.normal(`*m*`,`*s*`,`*shp*`)` → *Tableau de forme* ***shp*** *de flottants tirés selon une loi normale de moyenne* ***m*** *et d'écart-type* ***s***

`rd.uniform(`*a*`,`*b*`,`*shp*`)` → *Tableau de forme* ***shp*** *de flottants tirés selon une loi uniforme sur l'intervalle* [***a*** , ***b*** [

Conversions

Le passage maîtrisé `list` ↔ `ndarray` *permet de bénéficier des avantages des 2 types*

`T = np.array(L)` → *Liste en tableau, type de données automatique*

`T = np.array(L,dtype=`*data_type*`)` → *Idem, type spécifié*

`L = T.tolist()` → *Tableau en liste*

`new_T = T.astype(`*data_type*`)` → *Conversion des données*

`S = T.flatten()` → *Conversion en vecteur (la séquence des données telles qu'elles sont stockées en mémoire)*

`np.unravel_index(`n_s`,T.shape)` *donne la position dans le tableau* **T** *à partir de l'index séquentiel* n_s *(indice dans* **S**)

Matrices

générateurs

`np.eye(`*n*`)` → *matrice identité d'ordre* ***n***

`np.eye(`*n*`,k=`*d*`)` → *matrice carrée d'ordre* ***n*** *avec des* `1` *décalés de* ***d*** *vers la droite par rapport à la diagonale*

`np.diag(V)` → *matrice diagonale dont la diagonale est le vecteur* `V`

- ***Une matrice*** **M** ***est un tableau à deux indices***
- `M[`*i*`,`*j*`]` *est le coefficient de la (*$i+1$*)-ième ligne et (*$j+1$*)-ième colonne*
- `M[`*i*`,:]` *est la (*$i+1$*)-ième ligne,* `M[:,`*j*`]` *la (*$j+1$*)-ième colonne,* `M[`*i*`:`*i+h*`,`*j*`:`*j+l*`]` *une sous-matrice* $h \times l$
- ***Opérations : voir Vecteurs***
- ***Produit matriciel*** **:** `M.dot(V)` *ou* `np.dot(M,V)` *ou* `M@V`

`M.transpose(),M.trace()` → *transposée, trace*

Matrices carrées *uniquement (algèbre linéaire) :*

```
import numpy.linalg as la        ("Linear algebra")
```

`la.det(M), la.inv(M)` → *déterminant, inverse*

`vp = la.eigvals(M)` → `vp` *vecteur des valeurs propres*

`vp,P = la.eig(M)` → `P` *matrice de passage*

`la.matrix_rank(M), la.matrix_power(M,`*p*`)`

`X = la.solve(M,V)` → *Vecteur solution de* **M X** = **V**

Tableaux booléens, comparaison,tri

`B = (T==1.0)`

`B = (abs(T)<=1.0)` → `B` *est un tableau de booléens, de même forme que* `T`

`B = (T>0)*(T<1)` *Par exemple* `B*np.sin(np.pi*T)` *renverra un tableau de* $\sin(\pi x)$ *pour tous les coefficients* x *dans*]0,1[*et de 0 pour les autres*

`B.any(), B.all()` → *booléen « Au moins un* `True` *», « Que des* `True` *»*

`indices = np.where(B)` → ***tuple*** *de vecteurs d'indices donnant les positions des* `True`

`T[indices]` → *extraction séquentielle des valeurs*

`T.clip(`v_{min}`,`v_{max}`)` → *tableau dans lequel les valeurs ont été ramenées entre* v_{min} *et* v_{max}

Intégration numérique

```
import scipy.integrate as spi
```

`spi.odeint(F,Y0,Vt)` → *renvoie une solution numérique du problème de Cauchy* $\mathbf{Y}'(t) = \mathbf{F}(\mathbf{Y}(t),t)$ *, où* $\mathbf{Y}(t)$ *est un vecteur d'ordre n, avec la condition initiale* $\mathbf{Y}(t_0) =$ `Y0`*, pour les valeurs de t dans le vecteur* `Vt` *commençant par* t_0*, sous forme d'une matrice* $n \times k$

`spi.quad(f,a,b)[0]` → *renvoie une évaluation numérique de l'intégrale :* $\int_a^b f(t)\,dt$

Mémento Python 3 pour le calcul scientifique

Graphiques Matplotlib

```
import matplotlib.pyplot as plt
```

plt.figure(*mon_titre*, figsize=(*W*,*H*)) *crée ou sélectionne une figure dont la barre de titre contient **mon_titre** et dont la taille est **W**×**H** (en inches, uniquement lors de la création de la figure)*

plt.plot(X,Y,*dir_abrg*) *trace le nuage de points d'abscisses dans* **X** *et d'ordonnées dans* **Y** *; **dir_abrg** est une chaîne de caractères qui contient une couleur ("r"-ed, "g"-reen, "b"-lue, "c"-yan, "y"-ellow, "m"-agenta, "k" black), une marque (voir ci-dessous) et un type de ligne ("" pas de ligne, "-" plain, "--" dashed, ":" dotted, ...)*

- **Options courantes :**

linewidth=... *épaisseur du trait (0 pour aucun trait)* **label=**... *étiquette pour la légende*
color=... *couleur du trait :* **(*r*,*g*,*b*)** *ou* **"m","c",...** **dashes=**... *style de pointillé (liste de longueurs)*
marker=... *forme de la marque :*

"o" "." "*" ">" "<" "^" "v" "s" "d" "D" "p" "h" "H" "8" "1" "2" "3" "4" "+" "x"

markersize=... *taille de la marque* **markeredgewidth=**... *épaisseur du contour*
markeredgecolor=... *couleur du contour* **markerfacecolor=**... *couleur de l'intérieur*

plt.axis("equal"), plt.grid() *repère orthonormé, quadrillage*

plt.xlim(a,b), plt.ylim(a,b) *plages d'affichage ; si a > b, inversion de l'axe*

plt.xlabel(*axe_x*,size=*s*,color=(*r*,*g*,*b*)), plt.ylabel(*axe_y*,...) *étiquettes sur les axes, en réglant la taille **s** et la couleur de la police de caractères (**r**, **g** et **b** dans [0,1])*

plt.legend(loc="best",fontsize=*s*) *affichage des labels des "plot" en légende*

plt.show() *affichage des différentes figures et remise à zéro*

plt.twinx() *bascule sur une deuxième échelle des ordonnées apparaissant à droite du graphique*

plt.xticks(Xt), plt.yticks(Yt) *réglage des graduations des axes*

plt.subplot(nbL,nbC,numero) *début de tracé dans un graphique situé dans un **tableau de graphiques** à* **nbL** *lignes,* **nbC** *colonnes ;* **numero** *est le numéro séquentiel du graphique dans le tableau (voir ci-contre).*

plt.subplots_adjust(left=*L*,right=*R*,bottom=*B*,top=*T*,wspace=*W*,hspace=*H*) *ajustement des marges (voir ci-contre)*

plt.title(*Titre_du_graphique*) *rajout d'un titre au graphique en cours de tracé*

plt.suptitle(*Titre_général*) *rajout d'un titre à la fenêtre de graphiques*

plt.text(*x*,*y*,*texte*,fontdict=*dico*,horizontalalignment=*HA*,verticalalignment=*HV*) *tracé du texte **texte** à la position (**x**,**y**), avec réglage des alignements (**HA**∈{"center","left","right"}, **HV**∈{"center","top","bottom"}) ; **dico** est un dictionnaire (voir ci-contre)*

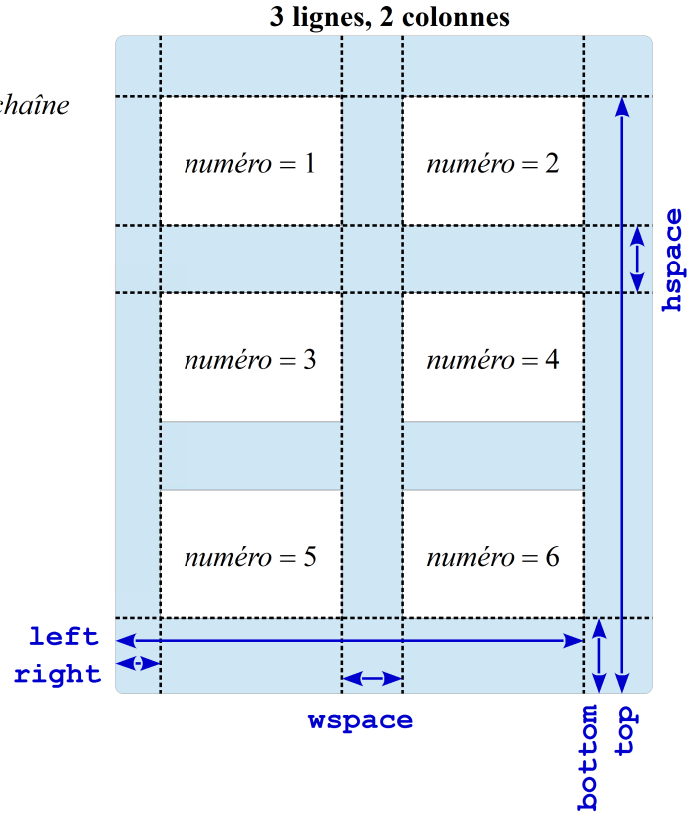
plt.axis("off") *suppression des axes et du cadre*

plt.imshow(T,interpolation="none",extent=(*gauche*,*droite*,*bas*,*haut*)) *tracé d'une image pixélisée non lissée à partir d'un tableau* **T** *(n_L×n_C×4 format **RGBA**, n_L×n_C×3 format **RGB**) ; l'option* **extent** *permet de régler la plage correspondant à l'image ((–0.5, n_C–0.5, n_L–0.5, –0.5) par défaut)*

plt.imshow(T,interpolation="none",vmin=v_{min},vmax=v_{max},cmap=*palette*,extent=(*gauche*,*droite*,*bas*,*haut*)) *tracé d'une image pixélisée non lissée à partir d'un tableau* **T** *rectangulaire n_L×n_C correspondant à des niveaux de gris sur la plage [v_{min}, v_{max}], avec la palette de couleurs **palette** : voir* **https://matplotlib.org/examples/color/colormaps_reference.html**)

plt.colorbar(schrink=*c*) *Affichage de l'échelle des couleurs du tracé précédent sur la plage [v_{min}, v_{max}]*

3 lignes, 2 colonnes



Dictionnaires

Un dictionnaire **D**, *de type* **dict** *(type itérable), se présente sous la forme :*

```
{clef_0:valeur_0,clef_1:valeur_1,...}
```

On peut accéder aux clefs par **D.keys()**, *aux valeurs par* **D.values()**, *et obtenir une liste de couples par* **D.items()**. *On peut extraire une valeur par sa clef :* **D[new_key]**, *ou compléter le dictionnaire par :* **D[new_key] = new_val.**

Parcours :

```
for key,val in D.items() :
    (bloc d'instructions)
```

Exemple pour la fonction **plt.text** :

```
{ "family" : "Courier New",
  "weight" : "bold",
  "style" : "normal",
  "size" : 18,
  "color" : (0.0,0.5,0.8) }
```

Exemple de tracé avec **plt.imshow** *et* **plt.colorbar**, *avec la palette* **"gist_heat"**