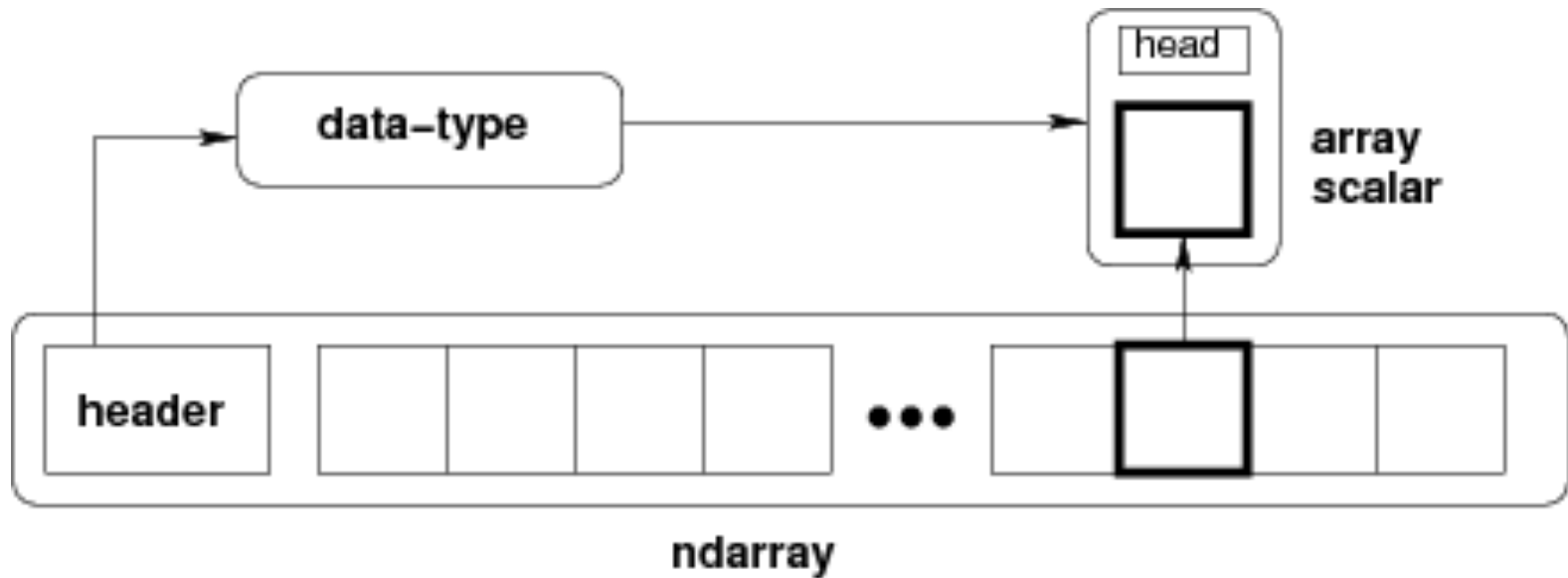


Révision

NumPy Array



Tableaux NumPy

CREATION SIMPLE

```
>>> a = array([0,1,2,3])
>>> a
array([0, 1, 2, 3])
```

VERIFIER LE TYPE

```
>>> type(a)
<type 'array'>
```

'TYPE' DES ELEMENTS

```
>>> a.dtype
dtype('int32')
```

OCTETS PAR ELEMENT

```
>>> a.itemsize # par element
4
```

SHAPE

```
# shape renvoie un tuple
# avec la longueur du tableau
# le long de chaque dimension
>>> a.shape
(4,)
>>> shape(a)
(4,)
```

SIZE

```
# size renvoie le nb
# d'elements dans le tableau
>>> a.size
4
>>> size(a)
4
```

Tableaux NumPy

OCTETS UTILISES EN MEMOIRE

```
>>> a.nbytes  
12
```

NB DE DIMENSIONS

```
>>> a.ndim  
1
```

COPIE DE TABLEAUX

```
>>> b = a.copy()  
>>> b  
array([0, 1, 2, 3])
```

CONVERSION EN LISTE

```
>>> a.tolist()  
[0, 1, 2, 3]
```

Pour des tableaux 1D list
fonctionne aussi mais #
lentement

```
>>> list(a)  
[0, 1, 2, 3]
```

Indexation

INDEXATION

```
>>> a[0]
0
>>> a[0] = 10
>>> a
[10, 1, 2, 3]
```

REEMPLISSAGE

```
>>> a.fill(0)
>>> a
[0, 0, 0, 0]
```

Aussi, mais c'est plus lent

```
>>> a[:] = 1
>>> a
[1, 1, 1, 1]
```



ATTENTION AU TYPE

```
>>> a.dtype
dtype('int32')
```

```
.
>>> a[0] = 10.6
>>> a
[10, 1, 2, 3]
```

```
>>> a.fill(-4.8)
>>> a
[-4, -4, -4, -4]
```

Tableaux Multi-Dimensionnels

```
>>> a = array([[ 0, 1, 2, 3],  
               [10,11,12,13]])
```


```
>>> a  
array([[ 0, 1, 2, 3],  
       [10,11,12,13]])
```

```
>>> a.shape  
(2, 4)  
>>> shape(a)  
(2, 4)
```

```
>>> a.size  
8  
>>> size(a)  
8
```

```
>>> a.ndim  
2
```

```
>>> a[1,3]  
13
```



colonne
ligne

```
>>> a[1,3] = -1  
>>> a  
array([[ 0, 1, 2, 3],  
       [10,11,12,-1]])
```

```
>>> a[1]  
array([10, 11, 12, -1])
```

Slicing (Vues)

```
>>> a[0,3:5]
array([3, 4])
```

```
>>> a[4:,4:]
array([[44, 45],
       [54, 55]])
```

```
>>> a[:,2]
array([2, 12, 22, 32, 42, 52])
```

```
>>> a[2::2,::2]
array([[20, 22, 24],
       [40, 42, 44]])
```

0	1	2	3	4	5
10	11	12	13	14	15
20	21	22	23	24	25
30	31	32	33	34	35
40	41	42	43	44	45
50	51	52	53	54	55

Slicing (Vues)

Le modèle permet une "indexation simple" (entiers et tranches) dans le tableau pour avoir une **vue** des mêmes données.

```
>>> b = a[:, ::2]
>>> b[0,1] = 100
>>> print a
[[ 1.    2.  100.]]
 [ 4.    5.    6.]]
>>> c =
a[:, ::2].copy()
>>> c[1,0] = 500
>>> print a
[[ 1.    2.  100.]]
 [ 4.    5.    6.]]
```

Autre utilisation de Vue

```
>>> b = a.view('i8')
>>> [hex(val.item()) for
val in b.flat]
['0x3FF0000000000000L',
 '0x4000000000000000L',
 '0x4059000000000000L',
 '0x4010000000000000L',
 '0x4014000000000000L',
 '0x4018000000000000L']
```


Slices (Tranches) sont des références

Les slices sont des références à la mémoire du tableau original. La modification des valeurs dans une tranche modifie également le tableau original.

```
>>> a = array((0,1,2,3,4))
```

```
# Créer une tranche contenant uniquement  
# le dernier élément de a
```

```
>>> b = a[2:4]
```

```
>>> b[0] = 10
```

```
# changer b a changé a!
```

```
>>> a
```

```
array([ 1,  2, 10,  3,  4])
```

Fancy Indexing

INDEXATION PAR LA POSITION

```
>>> a = arange(0,80,10)
```

```
# indexation avancée
```

```
>>> y = a[[1, 2, -3]]
```

```
>>> print y
```

```
[10 20 50]
```

```
# avec take
```

```
>>> y = take(a, [1,2,-3])
```

```
>>> print y
```

```
[10 20 50]
```

INDEXING AVEC DES BOOLEES

```
>>> mask = array([0,1,1,0,0,1,0,0],  
...              dtype=bool)
```

```
# indexation avancée
```

```
>>> y = a[mask]
```

```
>>> print y
```

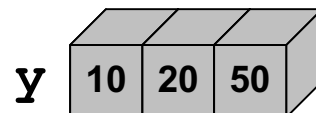
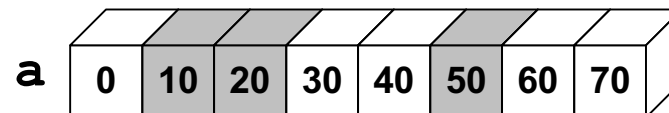
```
[10,20,50]
```

```
# avec compress
```

```
>>> y = compress(mask, a)
```

```
>>> print y
```

```
[10,20,50]
```



Indexation avancée en 2D

```
>>> a[(0,1,2,3,4), (1,2,3,4,5)]  
array([ 1, 12, 23, 34, 45])
```

```
>>> a[3:,[0, 2, 5]]  
array([[30, 32, 35],  
       [40, 42, 45]],  
       [50, 52, 55]])
```

```
>>> mask = array([1,0,1,0,0,1],  
                  dtype=bool)
```

```
>>> a[mask,2]  
array([2,22,52])
```

0	1	2	3	4	5
10	11	12	13	14	15
20	21	22	23	24	25
30	31	32	33	34	35
40	41	42	43	44	45
50	51	52	53	54	55



Contrairement au découpage en tranches, l'indexation fantaisie crée des copies au lieu de vues dans les tableaux originaux.

Opération sur les tableaux

```
>>> a = array([[1,2,3],
               [4,5,6]], float)
# La somme par défaut s'applique à
# toutes les valeurs du tableau.
>>> sum(a)
21.
```

```
# Ajouter le mot-clé "axis" pour
# effectuer la somme le long de
# l'axe 0.
```

```
>>> sum(a, axis=0)
array([5., 7., 9.])
```

```
# Ajouter le mot-clé "axis" pour
# effectuer la somme le long du
# dernier axe.
```

```
>>> sum(a, axis=-1)
array([6., 15.])
```

```
# La méthode a.sum() par
# défaut effectue la somme de
# toutes les valeurs du tableau.
>>> a.sum()
21.
```

```
# effectuer la somme le long d'un
# axe spécifique.
```

```
>>> a.sum(axis=0)
array([5., 7., 9.])
```

```
# produit le long des colonnes.
```

```
>>> a.prod(axis=0)
array([ 4., 10., 18.])
```

```
# forme fonctionnelle.
```

```
>>> prod(a, axis=0)
array([ 4., 10., 18.])
```

Min/Max

MIN

```
>>> a = array([2.,3.,0.,1.])
>>> a.min(axis=0)
0.
# Utilisez amin() de NumPy au lieu
# de min() intégré de Python pour
# des opérations plus rapides sur
# les tableaux multidimensionnels.
>>> amin(a, axis=0)
0.
```

ARGMIN

```
# Trouver l'index de la valeur
# minimale.
>>> a.argmin(axis=0)
2
# forme fonctionnelle.
>>> argmin(a, axis=0)
2
```

MAX

```
>>> a = array([2.,1.,0.,3.])
>>> a.max(axis=0)
3.
```

```
# forme fonctionnelle.
>>> amax(a, axis=0)
3.
```

ARGMAX

```
# Find index of maximum value.
>>> a.argmax(axis=0)
1
# forme fonctionnelle.
>>> argmax(a, axis=0)
1
```

Analyse statistique

MEAN

```
>>> a = array([[1,2,3],
               [4,5,6]], float)

# La valeur moyenne de chaque
# colonne
>>> a.mean(axis=0)
array([ 2.5,  3.5,  4.5])
>>> mean(a, axis=0)
array([ 2.5,  3.5,  4.5])
>>> average(a, axis=0)
array([ 2.5,  3.5,  4.5])
# La fonction average peut
# également calculer une moyenne
# pondérée
>>> average(a, weights=[1,2],
...          axis=0)
array([ 3.,  4.,  5.] )
```

=> moyenne pondérée de chaque colonne, en pondérant les valeurs de la première ligne par 1 et celles de la deuxième ligne par 2.

STANDARD DEV./VARIANCE

Standard Deviation

```
>>> a.std(axis=0)
array([ 1.5,  1.5,  1.5])
```

Variance

```
>>> a.var(axis=0)
array([2.25, 2.25, 2.25])
>>> var(a, axis=0)
array([2.25, 2.25, 2.25])
```

$$(1*1 + 4*2) / (1 + 2) = 3.$$

$$(2*1 + 5*2) / (1 + 2) = 4.$$

$$(3*1 + 6*2) / (1 + 2) = 5$$

Autres opérations

CLIP

Limiter les valeurs à une plage

```
>>> a = array([[1,2,3],  
               [4,5,6]], float)
```

Valeurs < 3 sont mises à 3.

Valeurs > 5 sont mises à 5.

```
>>> a.clip(3,5)
```

```
>>> a
```

```
array([[ 3.,  3.,  3.],  
       [ 4.,  5.,  5.]])
```

Arrondi se fait toujours à
l'entier pair le plus proche en
cas d'égalité

ROUND

Arrondir les valeurs dans un
tableau. NumPy arrondit au plus
proche, donc 1.5 et 2.5 sont
tous deux arrondis à 2

```
>>> a = array([1.35, 2.5, 1.5])
```

```
>>> a.round()
```

```
array([ 1.,  2.,  2.])
```

Arrondi au 1er décimal.

```
>>> a.round(decimals=1)
```

```
array([ 1.4,  2.5,  1.5])
```

Méthodes et Attributs

ATTRIBUTS DE BASE

`a.dtype` - Type numérique des éléments du tableau. `float32`, `uint8`, etc.
`a.shape` - Forme du tableau. `(m,n,o,...)` `a.size` - Nombre d'éléments dans tout le tableau.
`a.itemsize` - Nombre d'octets utilisés par un seul élément dans le tableau.
`a.nbytes` - Nombre d'octets utilisés par l'ensemble du tableau (données seulement).
`a.ndim` - Nombre de dimensions dans le tableau.

OPÉRATIONS DE FORME

`a.flat` - Un itérateur pour parcourir le tableau comme s'il était en 1D.
`a.flatten()` - Renvoie une copie en 1D d'un tableau multidimensionnel.
`a.ravel()` - Identique à `flatten()`, mais renvoie une "vue" si possible.
`a.resize(new_size)` - Change la taille/forme d'un tableau sur place.
`a.swapaxes(axis1, axis2)` - Échange l'ordre de deux axes dans un tableau.
`a.transpose(*axes)` - Échange l'ordre de n'importe quel nombre d'axes de tableau. `a.T` - Raccourci pour `a.transpose()`
`a.squeeze()` - Supprime les dimensions de longueur=1 d'un tableau.

Méthodes et attributs

REEMPLISSAGE ET COPIE

`a.copy()` - Renvoie une copie du tableau.

`a.fill(value)` - Remplit le tableau avec une valeur scalaire.

CONVERSION / COERCITION

`a.tolist()` - Convertit le tableau en listes imbriquées de valeurs.

`a.tostring()` - Copie brute de la mémoire du tableau dans une chaîne Python.

`a.astype(dtype)` - Renvoie le tableau contraint au dtype donné.

`a.byteswap(False)` - Convertit l'ordre des octets (big <-> little endian).

NOMBRES COMPLEXES

`a.real` - Renvoie la partie réelle du tableau.

`a.imag` - Renvoie la partie imaginaire du tableau.

`a.conjugate()` - Renvoie le conjugué complexe du tableau.

`a.conj()` - Renvoie le conjugué complexe d'un tableau. (identique à `conjugate()`)

Méthodes et attributs

SAUVEGARDE

`a.dump(file)` - Stocke les données du tableau sous forme binaire dans le fichier donné.

`a.dumps()` - Renvoie le pickle binaire du tableau sous forme de chaîne.

`a.tofile(fid, sep="", format="%s")` - Sortie ascii formatée vers un fichier.

RECHERCHE / TRI

`a.nonzero()` - Renvoie les indices de tous les éléments non nuls dans un.

`a.sort(axis=-1)` - Trie en place les éléments du tableau le long de l'axe.

`a.argsort(axis=-1)` - Renvoie les indices pour l'ordre de tri des éléments le long de l'axe.

`a.searchsorted(b)` - Renvoie l'indice où les éléments de `b` iraient dans `a`.

OPÉRATIONS MATHÉMATIQUES ÉLÉMENTAIRES

`a.clip(low, high)` - Limite les valeurs du tableau à la plage spécifiée.

`a.round(decimals=0)` - Arrondit au nombre spécifié de chiffres.

`a.cumsum(axis=None)` - Somme cumulative des éléments le long de l'axe.

`a.cumprod(axis=None)` - Produit cumulatif des éléments le long de l'axe.

Méthodes et attributs

MÉTHODES DE RÉDUCTION

Toutes les méthodes suivantes "réduisent" la taille du tableau d'une dimension en effectuant une opération le long de l'axe spécifié. Si l'axe est None, l'opération est effectuée sur l'ensemble du tableau.

- `a.sum(axis=None)` - Somme des valeurs le long de l'axe.
- `a.prod(axis=None)` - Produit de toutes les valeurs le long de l'axe.
- `a.min(axis=None)` - Trouve la valeur minimale le long de l'axe.
- `a.max(axis=None)` - Trouve la valeur maximale le long de l'axe.
- `a.argmin(axis=None)` - Trouve l'indice de la valeur minimale le long de l'axe.
- `a.argmax(axis=None)` - Trouve l'indice de la valeur maximale le long de l'axe.
- `a.ptp(axis=None)` - Calcule `a.max(axis) - a.min(axis)`
- `a.mean(axis=None)` - Trouve la valeur moyenne (moyenne) le long de l'axe.
- `a.std(axis=None)` - Trouve l'écart-type le long de l'axe.
- `a.var(axis=None)` - Trouve la variance le long de l'axe.
- `a.any(axis=None)` - Vrai si une valeur le long de l'axe est non nulle. (ou)
- `a.all(axis=None)` - Vrai si toutes les valeurs le long de l'axe sont non nulles. (et)

Autres opérations

CALCULS SIMPLES SUR LES TABLEAUX

```
>>> a = array([1,2,3,4])
>>> b = array([2,3,4,5])
>>> a + b
array([3, 5, 7, 9])
```

Constantes Numpy :



pi = 3.14159265359
e = 2.71828182846

FONCTIONS MATHÉMATIQUES

```
# Créer un tableau de 0 à 10
>>> x = arange(11.)

# Multiplier l'intégralité du
# tableau par une valeur scalaire
>>> a = (2*pi)/10.
>>> a
0.62831853071795862
>>> a*x
array([ 0., 0.628, ..., 6.283])

# Opérations en place
>>> x *= a
>>> x
array([ 0., 0.628, ..., 6.283])

# app. fonction au tableau.
>>> y = sin(x)
```

Opérateurs binaires

$a + b \rightarrow \text{add}(a,b)$
 $a - b \rightarrow \text{subtract}(a,b)$
 $a \% b \rightarrow \text{remainder}(a,b)$

$a * b \rightarrow \text{multiply}(a,b)$
 $a / b \rightarrow \text{divide}(a,b)$
 $a ** b \rightarrow \text{power}(a,b)$

MULTIPLIER PAR UN SCALAIRE

```
>>> a = array((1,2))
>>> a*3.
array([3., 6.] )
```

ADDITION ÉLÉMENT PAR ÉLÉMENT

```
>>> a = array([1,2])
>>> b = array([3,4])
>>> a + b
array([4, 6])
```

ADDITION EN UTILISANT UNE FONCTION D'OPÉRATEUR

```
>>> add(a,b)
array([4, 6])
```



OPÉRATION EN PLACE

```
# Écraser le contenu de a.
# Économise la création du tableau
# surcharge
>>> add(a,b,a) # a += b. résultat c
array([4, 6])
>>> a
array([4, 6])
```

Opérateurs logiques

equal	(==)
greater_equal	(>=)
logical_and	
logical_not	

not_equal	(!=)
less	(<)
logical_or	

greater	(>)
less_equal	(<=)
logical_xor	

2D EXAMPLE

```
>>> a = array((1,2,3,4), (2,3,4,5))
>>> b = array((1,2,5,4), (1,3,4,5))
>>> a == b
array([[True, True, False, True],
       [False, True, True, True]])
# fonction équivalente
>>> equal(a,b)
array([[True, True, False, True],
       [False, True, True, True]])
```

Opérateurs binaires

<code>bitwise_and</code>	<code>(&)</code>	<code>invert</code>	<code>(~)</code>	<code>right_shift(a,shifts)</code>
<code>bitwise_or</code>	<code>()</code>	<code>bitwise_xor</code>		<code>left_shift(a,shifts)</code>

BITWISE EXAMPLES

```
>>> a = array((1,2,4,8))
>>> b = array((16,32,64,128))
>>> bitwise_or(a,b)
array([ 17,  34,  68, 136])
```

inversion de bits

```
>>> a = array((1,2,3,4), uint8)
>>> invert(a)
array([254, 253, 252, 251], dtype=uint8)
```

Opération de décalage à gauche

```
>>> left_shift(a,3)
array([ 8, 16, 24, 32], dtype=uint8)
```

Trigonométrie et autres fonctions

TRIGONOMETRIQUE

<code>sin(x)</code>	<code>sinh(x)</code>
<code>cos(x)</code>	<code>cosh(x)</code>
<code>arccos(x)</code>	<code>arccosh(x)</code>
<code>arctan(x)</code>	<code>arctanh(x)</code>
<code>arcsin(x)</code>	<code>arcsinh(x)</code>
<code>arctan2(x,y)</code>	

AUTRES

<code>exp(x)</code>	<code>log(x)</code>
<code>log10(x)</code>	<code>sqrt(x)</code>
<code>absolute(x)</code>	<code>conjugate(x)</code>
<code>negative(x)</code>	<code>ceil(x)</code>
<code>floor(x)</code>	<code>fabs(x)</code>
<code>hypot(x,y)</code>	<code>fmod(x,y)</code>
<code>maximum(x,y)</code>	<code>minimum(x,y)</code>

'hypoténuse de deux nombres, →
`hypot(x,y)`

Calcul de la distance élément
par élément en utilisant $\sqrt{x^2 + y^2}$

Broadcasting : Diffusion

Lorsqu'il y a plusieurs entrées, elles doivent toutes être "diffusables" vers la même forme.

- Tous les tableaux sont promus au même nombre de dimensions (en préfixant des 1 à la forme).
- Toutes les dimensions de longueur 1 sont étendues selon les autres entrées avec des longueurs non unitaires dans cette dimension.

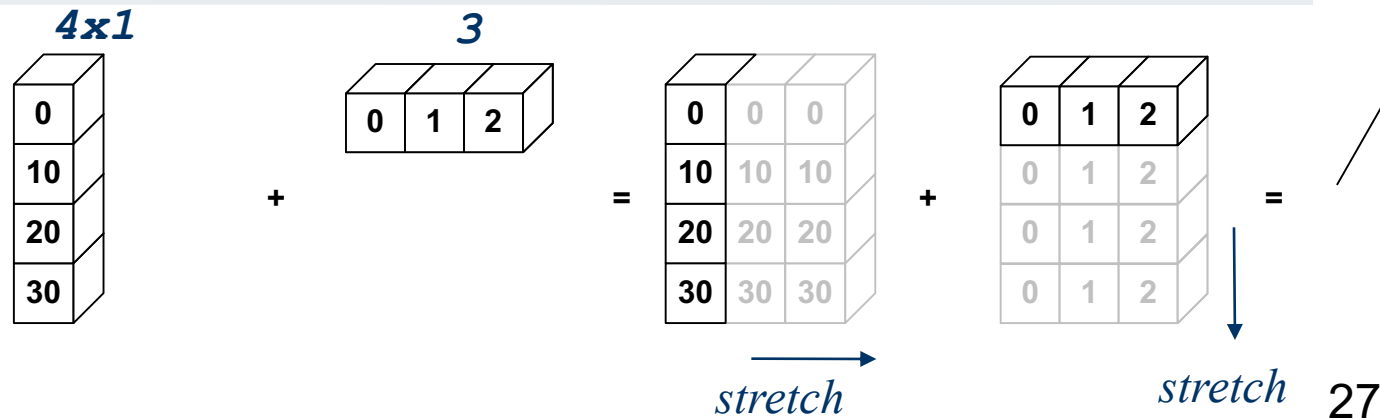
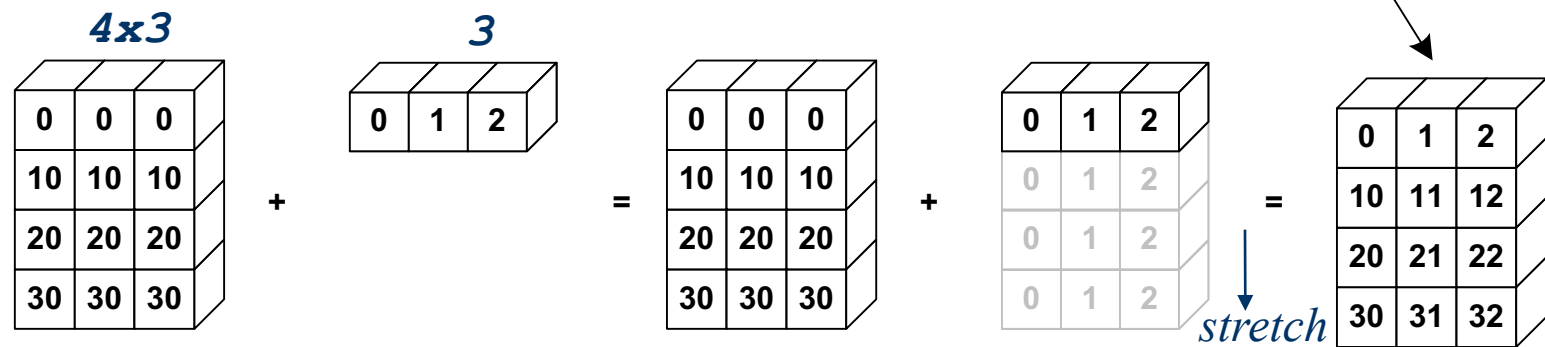
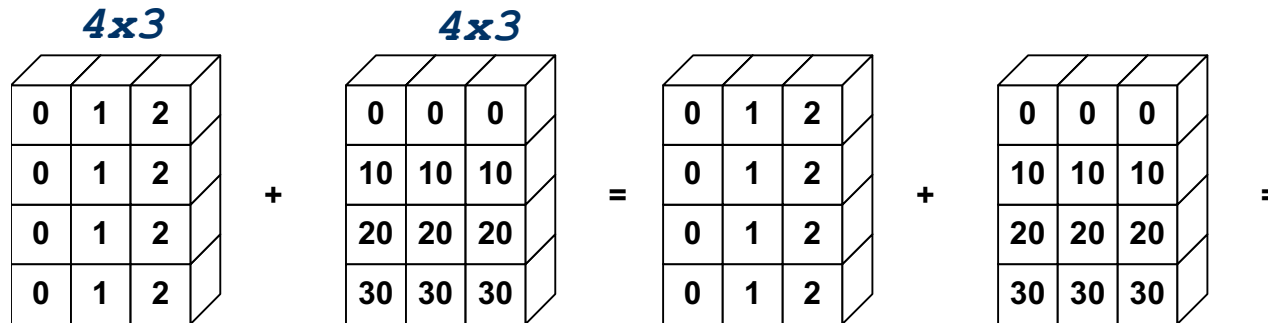
```
>>> x = [1,2,3,4];  
>>> y =  
[[10],[20],[30]]  
>>> print N.add(x,y)  
[[11 12 13 14]  
 [21 22 23 24]  
 [31 32 33 34]]  
>>> x = array(x)  
>>> y = array(y)  
>>> print x+y  
[[11 12 13 14]  
 [21 22 23 24]  
 [31 32 33 34]]
```

x a une forme (4,), la fonction universelle la voit comme ayant une forme (1, 4).

y a une forme (3, 1).

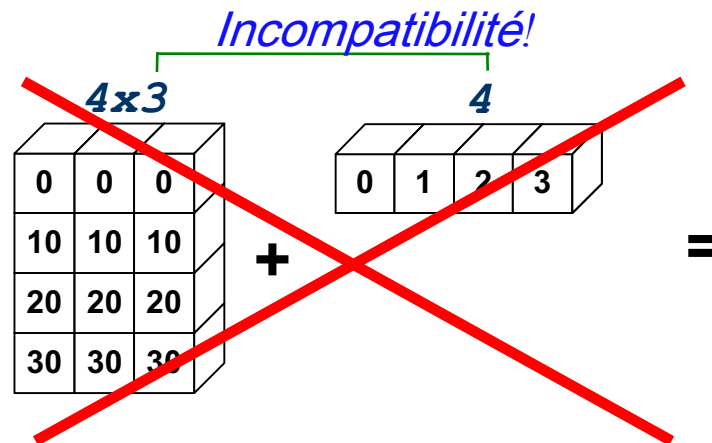
Le résultat de la fonction universelle a une forme (3, 4).

Broadcasting : Diffusion



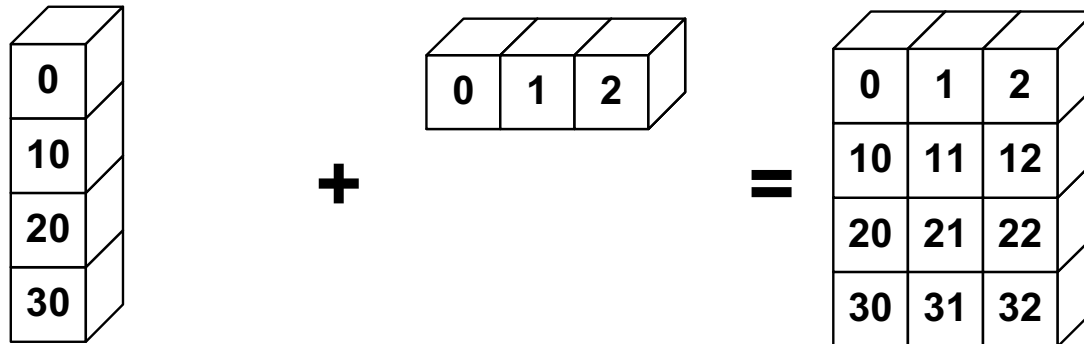
Règles Broadcasting

Les axes finaux des deux tableaux doivent soit être de taille 1, soit avoir la même taille pour que la diffusion se produise. Sinon, **une exception "ValueError: frames are not aligned" est levée.**



Broadcasting

```
>>> a = array((0,10,20,30))  
>>> b = array((0,1,2))  
>>> y = a[:, None] + b
```



Autres fonctions

Les opérateurs mathématiques, comparatifs, logiques et bit à bit qui prennent deux arguments (opérateurs binaires) ont des méthodes spéciales qui fonctionnent sur les tableaux :

`op.reduce(a,axis=0)`

`op.accumulate(a,axis=0)`

`op.outer(a,b)`

`op.reduceat(a,indices)`

```
# plt.axis([0, 10, 0, 35]) : Cela configure les
    limites des axes dans un graphique en utilisant
    matplotlib. Plus précisément, cela configure
    les limites de l'axe des x de 0 à 10 et les
    limites de l'axe des y de 0 à 35.

# plt.plot(x, y) trace une ligne reliant les
    points donnés par les coordonnées (x, y),
    tandis que plt.scatter(x, y) trace des points
    individuels aux coordonnées (x, y) sans les
    relier par une ligne. En d'autres termes,
    plt.plot() est utilisé pour tracer des courbes
    ou des lignes, tandis que plt.scatter() est
    utilisé pour tracer des nuages de points.
```