CDAT et le langage python

II. python et les extensions de CDAT

J-Y Peterschmitt / LSCE

http://www-lsce.cea.fr/Phocea/Membres/Cours/

Un petit exemple... (1/2)

(py_jyp_ex_51.py)

Calcul de la moyenne d'hiver de variables
var(time, latitude, longitude) définies sur 12 pas de temps

```
#!/usr/bin/env python

import os, time, cdms, vcs
from os import path

rep_in = './'
rep_out = path.join('/home/scratch01', os.getlogin())
nomprefix = 'TB1_an80a0149'

varlist = ['evap', 'pluies', 'tsol']

x = vcs.init()

for nomvar in varlist:

    print 'Traitement de', nomvar

    nomfic_in = '%s_%s_cm.nc' % (nomprefix, nomvar)
    fic_in = path.join(rep_in, nomfic_in)

    nomfic_out = '%s_%s_djf.nc' % (nomprefix, nomvar)
    fic_out = path.join(rep_out, nomfic_out)

    …
```

Les 2 principaux modules, fournis par CDAT, que l'on va étudier

Définition des fichiers et variables que l'on va étudier…

Initialisation de la fenêtre où seront tracés les graphiques, le *canevas*

Détermination des noms et chemins d'accès des fichiers avec lesquels on va travailler

Un petit exemple… (2/2)

```
for nomvar in varlist:
    …
```

Suite de la boucle sur les variables…

```
f = cdms.open(fic_in)
var = f(nomvar, latitude = (-90, 90),
        longitude = (-180, 360))
 f.close()
 var.savespace(1)
```

Lecture de la variable, en spécifiant la zone d'intérêt : on va lire plus de 360° en longitude, pour pouvoir afficher tout l'Atlantique et tout le Pacifique!

```
djf = (var[11, :, :] + var[0, :, :] + var[1, :, :]) / 3
```

Calcul de la moyenne *DJF* des mois d'hiver

```
djf.id = nomvar
djf.name = nomvar
djf.units = var.units
djf.longname = nomvar + ' djf average'
```

Définition/création des *attributs* de la variable qui vient d'être calculée (on les retrouvera dans le fichier netCDF final)

```
f = cdms.open(fic_out, 'w')
f.history = 'Created by %s (%s)' % \
            (os.getlogin(), time.asctime())
f.write(djf)
f.close()
```

Ouverture d'un fichier (en écriture), auquel on associe un attribut global, et sauvegarde de la nouvelle variable

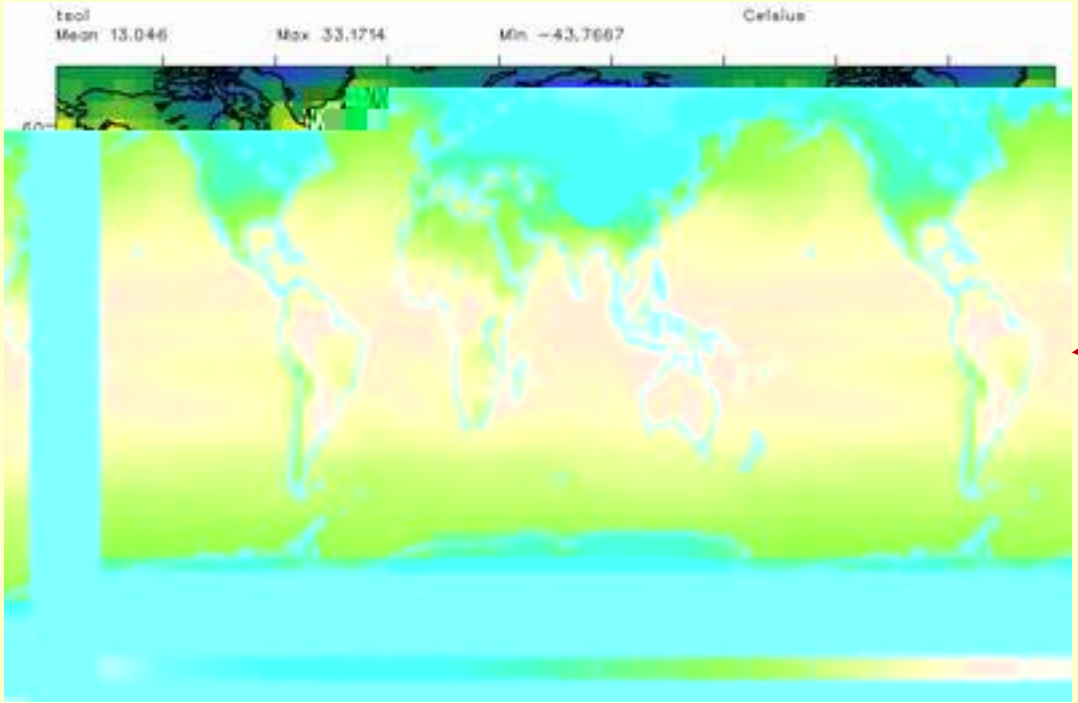
```
x.clear()
x.plot(djf)
```

Effacement du contenu du canevas et affichage de la moyenne DJF de la nouvelle variable

```
raw_input('Appuyez sur <Return> pour continuer')
```

Astuce pour économiser de la mémoire…

… et le résultat!



tsol en hiver

`ncdump –h TB1_an80a0149_tsol_djf.nc`

```
float tsol(latitude, longitude) ;                          ← djf.id
    tsol:missing_value = 1.e+20f ;
    tsol:name = "tsol" ;                                   ← djf.name
    tsol:longname = "tsol djf average" ;                   ← djf.longname
    tsol:units = "Celsius" ;                               ← djf.units

// global attributes:
  :Conventions = "CF-1.0" ;
  :history = "Created by jypeter (Tue Nov 15 17:36:29 2005)" ← f.history
```

Objectifs du cours

- Découvrir les principaux modules que CDAT rajoute au python standard
  - … en insistant beaucoup sur le module *Numeric*!
    - → Bien comprendre le fonctionnement des matrices (la *syntaxe tableaux*) permet :
      - d'éviter d'utiliser trop de boucles ***for***
      - d'obtenir de **très** bonnes performances!!

Ce cours n'est pas…

- Un cours sur python…
  - Les bases des cours précédents sont supposées (a peu près…) acquises!

    Sinon, il faut réviser

    1. *Introduction à python*
    2. *Plus loin avec python…*
- Une présentation exhaustive de tout ce que CDAT ajoute au python de base…
  - Après le tour d'horizon, il faudra aller lire la documentation et les *tutorials* disponibles, en fonction de vos besoins!
  - Vos collègues utilisateurs de python peuvent vous fournir des exemples utiles… ☺

    → Exemples de ***jypeter***, au LSCE

    /home/users/jypeter/CDAT/Progs/Devel

Qu'est-ce que CDAT?

- En résumé :
  - **C**limate **D**ata **A**nalysis **T**ools
  - CDAT rajoute des fonctionnalités au python standard
  - développé au PCMDI (USA) pour l'analyse des données AMIP, IPCC, …
- Le contenu
  - il y a des modules…
    - **`Numeric`** : librairie mathématique et calcul matriciel en *syntaxe tableau*, gestion des données masquées, …
    - **`cdms`** : entrée-sorties (grads/netCDF → netCDF), gestion des grilles, du temps, support de DODS, agrégation de fichiers via des descripteurs XML, …
    - **`vcs`** : sortie graphique de données sur grilles de type

      `v(temps, niveau, latitude, longitude)`
    - **`cdutil`**, **`cdtime`**, **`xmgrace`**, …
  - … et des programmes
    - **`vcdat`** : interface graphique de manipulation de fichiers
    - **`pyfort`** : intégration de procédures fortran dans python
    - …

Pourquoi utiliser CDAT?

- CDAT est basé sur python, un vrai langage de programmation
  - → permet de bénéficier de tout ce qui est développé dans la communauté *python*
- CDAT peut être installé (relativement) facilement sur des plateformes linux (et sur Mac OS X)
- CDAT est développé par des scientifiques de la communauté climat
  - Mêmes besoins que nous : production et analyse de gros volumes de données
  - Utilisation avec succès dans des projets de comparaison de modèles de climat : AMIP, CMIP, IPCC, …

Comment accéder à CDAT?

- Win : non disponible (pour l'instant?)
  - remarque : **`Numeric`** est disponible pour Windows dans la distribution Enthought!
- Linux et Max OS X:
  - installation en suivant les instructions standard CDAT
    - définir le chemin d'accès à CDAT (dépend du shell utilisé…)
    - taper '**`which python`**' pour vérifier
      - **`/usr/bin/python`** → python standard ☹
      - **`/votre_chemin/python`** → CDAT ☺
  - utilisation au LSCE
    - taper '**`source ~jypeter/CDAT/cdat.login`**' (pour les personnes qui utilisent le **`tcsh`** comme shell…) pour *activer* le python/CDAT installé au LSCE
      - **`…/cdat/…/python`** → CDAT ☺
  - indiquer en début de script quel python utiliser
    - **`#!/usr/bin/env python`**
- En batch ou dans un *shell* différent de **`tcsh`**

  Il faut lancer les scripts en utilisant le python de CDAT avec son chemin d'accès complet!

  ```
  /chemin_cdat/python script.py
  ```

CDAT et le langage python

Gestion des tableaux avec…

… le module **`Numeric`**

Caractéristiques des matrices

- Les objets *matrices* ou *tableaux* sont disponibles dans le module **Numeric**
  - dans ce qui suit : **`import Numeric as N`**
- Les caractéristiques d'une matrice sont :
  - le nombre de dimensions ou rang (*rank*)
  - la taille des dimensions ou forme (*shape*)
  - le type ou la précision des éléments (*typecode*)
  - le masque (*mask*) si certains éléments ne sont pas définis
  - exemple :
    `A[10]` → matrice de rang ***1*** (=vecteur), de forme ***(10,)***
- Les conventions sont les mêmes que pour le langage C :
  - pour une dimension de taille **`D`** : indices de ***0*** à ***D-1***
  - ordre des dimensions inversé par rapport au fortran!
    - fortran : **A(*dim*$_D$, …, *dim*$_2$, *dim*$_1$)**
    - C : **A[*dim*$_0$][*dim*$_1$][…][*dim*$_{D-1}$]**
    - python : **A[*dim*$_0$, *dim*$_1$, …, *dim*$_{D-1}$]**
    - Exemple :

      fortran → **`tsol(longitude, latitude, time)`**

      python → **`tsol[time, latitude, longitude]`**

Type/précision des éléments d'un tableau

- Tous les éléments d'un tableau ont le même type (pas de mélange, contrairement aux listes, …)
- Types numériques disponibles (les plus utilisés)

  typecode de **`Numeric`** ⇔ type fortran

  - **`N.Int32`** (**`'i'`** ou **`'l'`**) ⇔ **`integer*4`**

    `-2147483648` (⇔ 2^(4*8-1)) ≤ **`N.Int32`** ≤ **`2147483647`**
  - **`N.Float32`** (**`'f'`**) ⇔ **`real*4`**

    `1.17549435E-38` ≤ **`abs(N.Float32)`** ≤ **`3.40282347E+38`**

    ~7 chiffres significatifs…
  - **`N.Float64`** (**`'d'`**) ⇔ **`real*8`**

    `2.2250738585072013E-308` ≤ **`abs(N.Float64)`** ≤ **`1.7976931348623158E+308`**

    ~15 chiffres significatifs…
- Détermination du type numérique d'un tableau

  `A.typecode()` → **`'f'`** ⇔ les éléments de A sont de type **`N.Float32`**

  Taille en octets des éléments d'un tableau :

  → **`A.itemsize()`** → **`4`** (si les éléments de A sont de type **`N.Float32`**)
- Changement du type d'un tableau

  `B = A.astype(N.Float32)`

Gestion auto de la précision (1/2)

Attention! Par défaut, lors d'une opération utilisant des tableaux de **`Float32`** et des opérandes dont le type n'est pas défini explicitement, ***`Numeric`*** *passe automatiquement en* ***`Float64`***…

Exemple : **`D`**, **`J`**, **`F`** sont des tableaux de type **`N.Float32`**

`DJF = (D+J+F)/3` sera de type **`N.Float64`**!

→ gênant si on travaille sur des grosses matrices…

→ gênant si on doit écrire le résultat sous forme de **`real*4`** dans un fichier

- Solution 0 : imposer la précision *après* l'opération (ou avant l'écriture dans un fichier)

```
DJF = ((D+J+F)/3).astype(D.typecode())
```

- Solution 1 : définir *explicitement* le type des opérandes

```
trois = N.array(3, N.Float32)
DJF = (D+J+F)/trois
```

Gestion auto de la précision (2/2)

- Solution 2 : indiquer à **`Numeric`** d'essayer *d'économiser* la mémoire!
  - **`D.savespace(1)`** → activer le mode *éco*
    `D.savespace(0)` → gestion auto (par défaut)
  - **`D.spacesaver()`** → ***`1`*** si mode *éco* (***`0`*** sinon)
  - Lors d'une opération, si l'une des matrices est en mode éco, et de type **`Float32`**, le résultat sera de type **`Float32`**…

- Le plus simple, si on veut éviter des problèmes de changement de précision
  → Demander directement le mode *éco* lors de la création des tableaux de type **`Float32`**

```
N.fonction_de_creation(…, typecode=N.Float32, savespace=1)
```

- *Tableaux de **`Int32`**… Il vaut mieux ne pas utiliser savespace=1 ET convertir les constantes en réels!*

```
          I.savespace(0)          I.savespace(1)
I         array([5, 10],'i')      array([5, 10],'i')
I / 2     array([2, 5],'i')       array([2, 5],'i')
I / 2.    array([ 2.5,  5. ])     array([2, 5],'i')
I * 0.5   array([ 2.5,  5. ])     array([0, 0],'i')
```

Création d'un tableau

- A partir d'une liste

  `A = N.array(liste_de_nombres, typecode, savespace)`

  Exemple : **`A = N.array(range(10), N.Float32)`**

  ⇔ **`A = N.arrayrange(0, 10, 1, N.Float32)`**

  ⇔ **`A = N.arange(0, 10, 1, N.Float32)`**

  - Note : on peut transformer un tableau en une liste avec sa méthode **`tolist()`**.

    Exemple d'application : création facile d'une **liste** de réels ☺

    (bornes d'intervalles, etc…)

    `N.arange(-5, 5.1, .5, N.Float32).tolist()`

    → [-5.0, -4.5, -4.0, …, 5.0]

- Tableaux pré-initialisés, de forme donnée :
  - **`A = N.ones((72, 96))`**

    → Tableau rempli avec des **`1`** (**`N.Int32`**, par défaut)
  - **`A = N.zeros((72, 96), N.Float32, 1)`**

    → Tableau rempli avec des 0. (**`N.Float32`**), avec l'attribut **`savespace`** à 1
  - **`A = N.identity(10, typecode=N.Float32)`**

    → Matrice identité, avec des **`0.`** et des **`1.`** (**`N.Float32`**)

- A partir d'opérations mathématiques combinant des tableaux existants

  Exemple : **`DJF = (D+J+F)/3`**

- …

Accès aux éléments d'un tableau

- Même convention pour les indices que pour les chaînes, listes, …
  - premier élément : indice ***0***
  - dernier élément : indice ***D-1*** ou ***-1*** (pour une dimension à ***D*** éléments)
  - fonctionnement standard des opérations d'*indexing* et de *slicing*
  - rappel : *retournement à la volée* d'une dimension : **`A[0, ::-1]`**
  - sélection de tous les éléments d'une dimension (indice *muet*) avec '**`:`**' → **`A[0, :]`**
  - les indices muets de plusieurs dimensions consécutives peuvent être remplacés par '**`...`**' (*trois points consécutifs*)

    `A[0, :, :, :]` ⇔ **`A[0, ...]`**

    `A[0, :, :, 9]` ⇔ **`A[0, ..., 9]`**

  - ne pas spécifier les indices des dernières dimensions d'un tableau est équivalent à sélectionner tous les éléments de ces dimensions…

    Pour un tableau **`A`** à 3 dimensions :

    `A[0]` ⇔ **`A[0, :]`** ⇔ **`A[0, :, :]`** ⇔ **`A[0, ...]`**

- ATTENTION! Il ne faut pas confondre la notation utilisée pour des listes imbriquées et celle pour un tableau à plusieurs dimensions!
  - liste : **`lst[3][2]`**
  - tableau : **`A[3, 2]`**

Changement de *taille*

- L'attribut **shape** donne la forme d'un tableau (tuple) :
  A (*50x64*) : **A.shape → (50, 64)**

- On peut changer la forme, du moment que l'on ne change pas le nombre total d'éléments :
  - **A2 = N.reshape(A, (3200,))**
    A2.shape → (3200,)
    Attention! **A2** n'est pas une copie de **A**!! C'est un pointeur vers le tableau **A**… avec une autre forme que **A**
  - Manipulation directe de la taille :
    A.shape = (1, 50, 64) ⇔ A.shape = (1, -1, 64)
    Astuce : une taille de dimension à **-1** est remplacée par la valeur nécessaire pour conserver le nombre d'éléments du tableau!

- Astuce d'optimisation : l'attribut ***flat*** donne accès à la version 1D d'un tableau (sans créer de nouveau tableau en mémoire)!
  c.-à-d. **tableau[]** et **tableau.flat[]** pointent sur le *même* tableau (en mémoire), ou partagent les mêmes données

  Exemple : 2 façons de positionner à 1 les deux premières lignes d'une matrice de 64 colonnes…

  A (50x64) ⇔ **A.flat** (*3200*)
  A[0:2, :] = 1 ⇔ **A.flat[0:128] = 1**

  Voir exemple d'utilisation plus loin (***reduce*** *et les extrema*)

Note : dans le cas (exceptionnel!) où **A.flat** ne marcherait pas parce que A n'est pas stocké d'un seul tenant en mémoire, on peut utiliser à la place **N.ravel(A)**…

Opérations sur les tableaux

- Les opérations se font en général *terme à terme*!
  - **`C = A * B`**

    `A` et **`B`** doivent avoir la même taille, et le résultat est le produit des éléments de **`A`** par les éléments correspondants de **`B`**
  - si les opérandes n'ont pas la même taille, Numeric *essaye* automatiquement de les rendre *conformants*.

    `B = A + 3`

    ⇔ **`B = A + 3 * N.ones(A.shape)`**

    ⇔ ***`3`*** *est équivalent à une matrice de forme* ***`(1,)`****, qui peut être étirée pour avoir la même forme que* ***`A`*** ☺

- Les opérations mathématiques standards sont disponibles :
  - **`+`**, **`-`**, **`/`**, **`**`**, **`N.absolute`**, **`N.cos`**, **`N.minimum`**, …
    - note : **`B = A + 3`** ⇔ **`B = N.add(A, 3)`**
    - attention, **`minimum`** et **`maximum`** comparent terme à terme les éléments de deux tableaux! Voir plus loin comment déterminer les extrema d'un tableau
  - **`numpy.pdf`**, I.6 (*Ufuncs* ou *Universal functions*) pour plus de détails!

Opérateur binaire… sur un seul tableau

- La méthode **reduce** d'une Ufunc, applique la fonction sur les 2 premiers éléments d'un tableau, puis sur le résultat et l'élément suivant, etc…

  Ufuncs : **add**, **subtract**, **multiply**, **divide**, **maximum**, **minimum**, …
- Exemple : **A = N.arange(2, 5, 1) → [2,3,4]**

  Produit des éléments du tableau **A** :

  N.multiply.reduce(A) → 24

  [2, 3, 4]

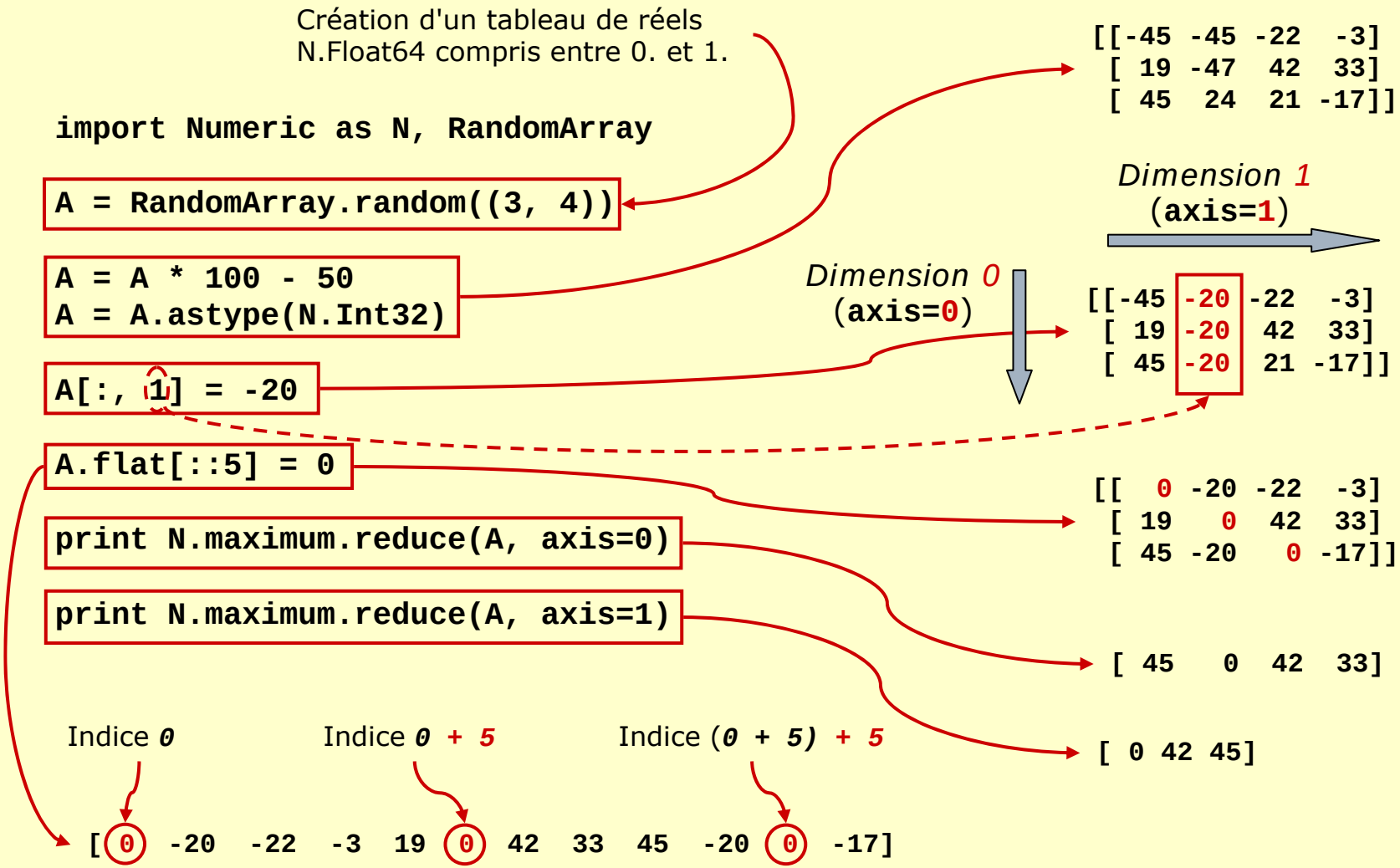
  2*3 = 6

  6*4 = 24

  Pour des tableaux à plusieurs dimensions, on peut spécifier la dimension selon laquelle la *réduction* s'effectue (dimension ***0*** par défaut → **N.*ufunc*.reduce(…, axis=0)**)
- Deux *raccourcis* (alias) sont disponibles :
  - **N.multiply.reduce(A) ⇔ N.product(A)**
  - **N.add.reduce(A) ⇔ N.sum(A)**
- Extrema d'un tableau :
  - **N.maximum.reduce(A.flat)**
  - **N.minimum.reduce(A.flat)**

Exemple : **reduce** et les extrema

(py_jyp_ex_52.py)



Fonctions de gestion des tableaux (1/3)

Tous les détails dans la partie I.8 de **numpy.pdf** (*Array functions*)

- *Copie* d'un tableau :
  B = N.array(A)
  Attention! N.asarray(A) ne fait pas de copie de A!
  N.asarray(A) ⇔ **N.array(A, copy=0)**
- Transposition :
  N.transpose(A, *axes=None*)
- *Vrai* produit de matrices :
  N.matrixmultiply(A, B)
- Produit scalaire de deux vecteurs
  N.dot(V_1, V_2)
- Moyenne d'un tableau :
  N.average(A, axis=0)
  Note : moyenne sur tout le tableau si **axis=None**
- *Indice* du plus grand/petit élément :
  N.argmax(A, *axis=-1*), N.argmin(A, *axis=-1*)
  Rappel : Pour obtenir les *extrema*, il faut utiliser:
  N.maximum.reduce(A.flat), **N.maximum.reduce(A.flat)**
- Saturation des éléments d'un tableau entre 2 seuils
  N.clip(A, Val_{min}, Val_{max})
- Vérification de l'égalité de 2 tableaux (avec une tolérance)
  N.allclose(A, B, *rtol = 1.e-5, atol = 1.e-8*)
  Renvoie 1 (True) si les 2 tableaux sont égaux, 0 sinon
  Note : **A == B** crée un tableau avec le résultat de la comparaison pour chaque élément!

Fonctions de gestion des tableaux (2/3)

- Concaténation de tableaux :

  N.concatenate((A$_0$, …, A$_{N-1}$), *axis=0*)

```
>>> un = N.ones((1, 3))   →[ [1,1,1,]]
>>> N.concatenate((un, un*2, un*3))
        [[1,1,1,]
         [2,2,2,]
         [3,3,3,]]
>>> N.concatenate((un, un*2, un*3), axis=1)
        [ [1,1,1,2,2,2,3,3,3,]]
```

- Ajout d'une dimension et répétition (plusieurs méthodes):

```
B = N.array(((1, 2), (3, 4)))          →  [[1,2,]
                                           [3,4,]]
```

  - `C = N.resize(B, (3,) + B.shape)` →

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

  `C.shape` → `(3, 2, 2)`

  Attention! Ne pas confondre **resize()** et **reshape()**

  - `C = N.concatenate((B[N.NewAxis,...], B[N.NewAxis,...], B[N.NewAxis,...]))`

    Remarque : ***N.NewAxis*** rajoute *temporairement/virtuellement* un axe au tableau **C** (évite de devoir préalablement faire un **reshape**)

  - `C = B+N.zeros((3,2,2))`

    B est rendu *conformant* à un tableau à trois dimensions, puis ajouté aux zéros…

Fonctions de gestion des tableaux (3/3)

- Choix entre les éléments de 2 tableaux :

  N.where(*condition*, A, B)

  Crée une matrice qui contient les éléments de **A** là où la condition est **True**, et de **B** sinon.

  Exemple : mise à *0* des éléments négatifs

  N.where(A>0, A, 0.)

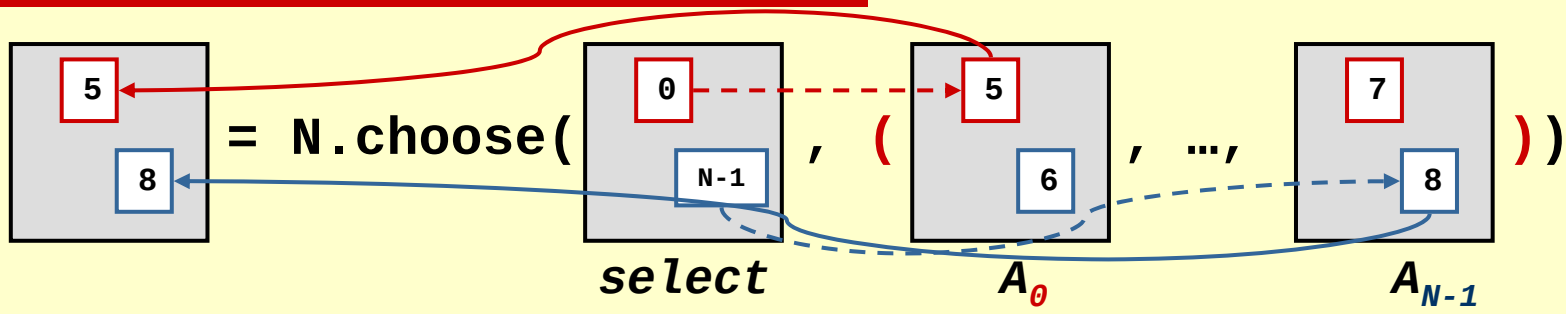
- Sélection dans plusieurs tableaux :

  N.choose(select, (A_0, A_1, …, A_{N-1}))

  Permet de créer un tableau **A** de même taille que A_i (et que **select**), dont chaque élément est choisi en fonction de l'indice spécifié à la même position dans **select**

  C'est une généralisation à plusieurs tableaux de **N.where(),** mais il faut faire attention à l'ordre des paramètres! → voir exemple suivant

choose et **where** de plus près...



- Mise à zéro des éléments négatifs d'une matrice

```
>>> A = N.array(((-1, 2, -1), (2, -1, 2), (-1, 2, -1)))
                                          [[-1  2 -1]
                                     A =   [ 2 -1  2]
                                           [-1  2 -1]]
>>> A > 0                          →      [[ 0  1  0]
                                           [ 1  0  1]
                                           [ 0  1  0]]
>>> positif = N.greater(A, 0)      ← équivalent à 'A > 0'
>>> N.allclose(A>0, positif)       → 1    (les deux méthodes/matrices sont
                                           bien identiques!)
>>> N.where(A>0, A, 0)             →      [[ 0  2  0]
                                           [ 2  0  2]
                                           [ 0  2  0]]
>>> N.choose(positif, (0, A))      →      même résultat que
                                          N.where(A>0, A, 0)
```

- On pourrait aussi utiliser, pour ne pas avoir d'éléments inférieurs à zéro :

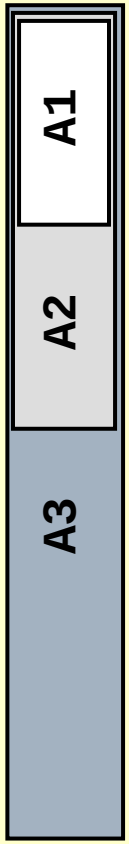
```
N.clip(A, 0, 1000)
```

Les tableaux et la mémoire… (1/3)

- Les éléments d'un tableau de **`Numeric`** sont (normalement) rangés de façon contigüe en mémoire
- Place occupée en mémoire :
  - produit de la taille des dimensions * nb d'octets par élément
  - **`N.product(A.shape) * A.itemsize()`**
- *C'est l'indice de la dernière dimension (la plus à droite) qui varie le plus vite…*
  - **`A[`*`dim`*$_0$**`,` *`dim`*$_1$**`,` *`dim`*$_2$**`]`** ← *dim*$_0$ * *dim*$_1$ * *dim*$_2$ *éléments*
  - **`A.flat[d`**$_f$ **`=` *`dim`*$_0$`*`*`dim`*$_1$`*`*`dim`*$_2$**`]`** ← *dim*$_0$ * *dim*$_1$ * *dim*$_2$ *éléments*
  - Correspondance des indices entre **`A.flat`** et **`A`** :
    `A.flat[k*`*`dim`*$_1$`*`*`dim`*$_2$ **`+` *`li`*`*`*`dim`*$_2$ **`+`** *`co`*`]` ⇔ **`A[k, li, co]`**
- Fonctions de conversion entre un tableau ***`A`*** et une chaîne ***`ram`*** (c.-à-d. une suite d'octets, en mémoire ou dans un fichier) :
  - **`ram = A.tostring()`**
  - **`B = N.fromstring(ram, A.typecode())`**
    `B.shape = A.shape`

Les tableaux et la mémoire… (2/3)

```
A1 → array( [0,   1,  0],'i')
A2 → array([[ 0,  1,  0],
            [16,  0,  0]],'i')
A3 → array([[[  0,   1,   0],
             [ 16,   0,   0]],
            [[  0,   0, 256],
             [  0, 512,   0]]],'i')
```



Valeur	Valeur (hexadecimal/ *little endian*)	*Adresse* (décalage en octets)	A1[co] (3,)	A2[li, co] (2, 3)	A3[k, li, co] (2, 2, 3)	Index A3 k*2*3 + li*3 + co
0	\00\00\00\00	0	A1[0]	[0,0]	[0,0,0]	0
1	\01\00\00\00	4	A1[1]	[0,1]	[0,0,1]	1
0	\00\00\00\00	8	A1[2]	[0,2]	[0,0,2]	2
16	\10\00\00\00	12		[1,0]	[0,1,0]	3 + 0 = 3
0	\00\00\00\00	16		[1,1]	[0,1,1]	3 + 1 = 4
0	\00\00\00\00	20		[1,2]	[0,1,2]	3 + 2 = 5
0	\00\00\00\00	24			[1,0,0]	6 + 0 + 0 = 6
0	\00\00\00\00	28			[1,0,1]	7
256	\00\01\00\00	32			[1,0,2]	8
0	\00\00\00\00	36			[1,1,0]	9
512	\00\02\00\00	40			[1,1,1]	10
0	\00\00\00\00	44			[1,1,2]	11

Les tableaux et la mémoire… (3/3)

- Création de **A1**, **A2** et **A3**
  - `A1 = N.array((0, 1, 0), N.Int32)`
  - `A2 = N.array(((0, 1, 0), (16, 0, 0)), N.Int32)`
  - `A3 = N.array((((0, 1, 0), (16, 0, 0)),`
    `((0, 0, 256), (0, 512, 0))), N.Int32)`
- Etude du contenu de **A2**

```
A2 → array([[ 0,  1,  0],
            [16,  0,  0]],'i')
    ram2 = A2.tostring()
    ram2[0    :4]    → '\x00\x00\x00\x00'      (print 0x00 → 0)
    ram2[0+4*1:4+4*1] → '\x01\x00\x00\x00'      (print 0x01 → 1)
    ram2[0+4*3:4+4*3] → '\x10\x00\x00\x00'      (print 0x10 → 16)
```

- Note : si on veut vraiment que les 3 tableaux soient au même endroit en mémoire…

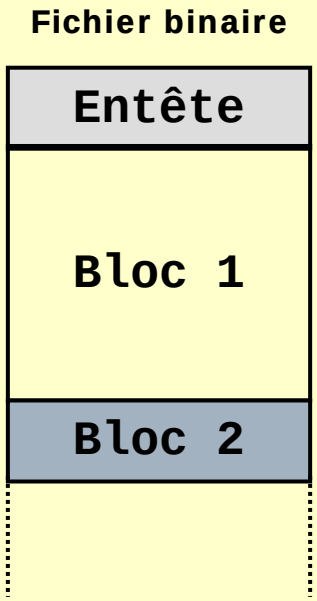
  `A2bis = A3[0, :, :]` ou `A2bis = A3[0, ...]`

  `A1bis = A3[0, 0, :]` ou `A1bis = A2bis[0, :]`

  `A1bis[0] = -99` → modifie la valeur de `A2bis[0, 0]` et `A3[0, 0, 0]` !
- Note : *little endian* et *big endian* …
  - *little* : `'\x00\x02\x00\x00'` ⇔ *big* : `'\x00\x00\x02\x00'`
  - `print 0x00000200` → `512` (rappel : $512 = 2*2^8 + 0*2^4 + 0$)

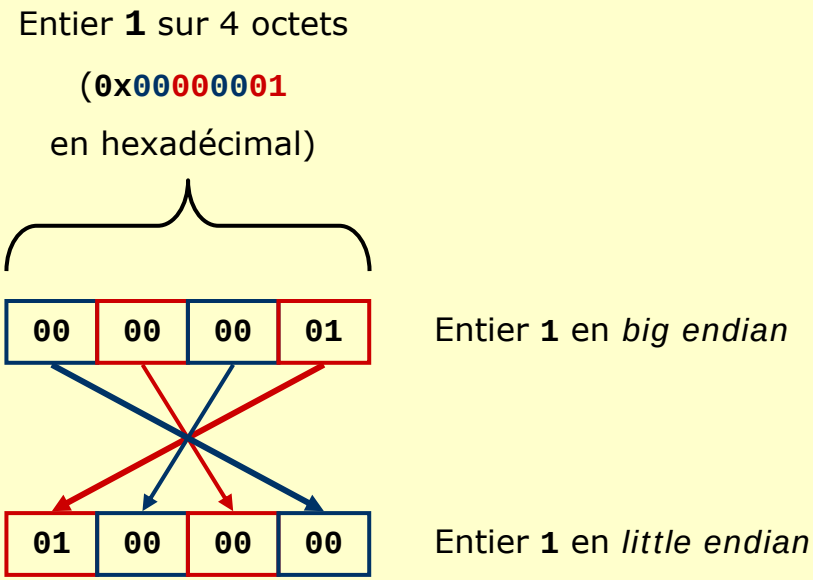
Les fichiers binaires

- Qu'est-ce qu'un fichier binaire (cas général)?
  - pas un fichier texte ☺
    → Pas manipulable avec un éditeur de texte (vi, emacs, …)
  - entête (éventuel) et suite de blocs de données binaires et/ou texte
- Pourquoi utiliser des fichiers binaires?
  - pas pratique à lire… ☹ Sauf si on utilise un format *auto-documenté* (netCDF, …)
  - cela prend moins de place de stocker un nombre en binaire qu'avec des caractères…
    - Entier sur 4 octets : +/- 2^{31} → **-2147483648** **2147483647**
    - Equivalent avec du texte → 11 caractères! 10 caractères
  - la lecture/écriture (par des programmes…) de fichiers binaires est beaucoup plus rapide que celle de fichiers texte



Manipulation de fichiers binaires

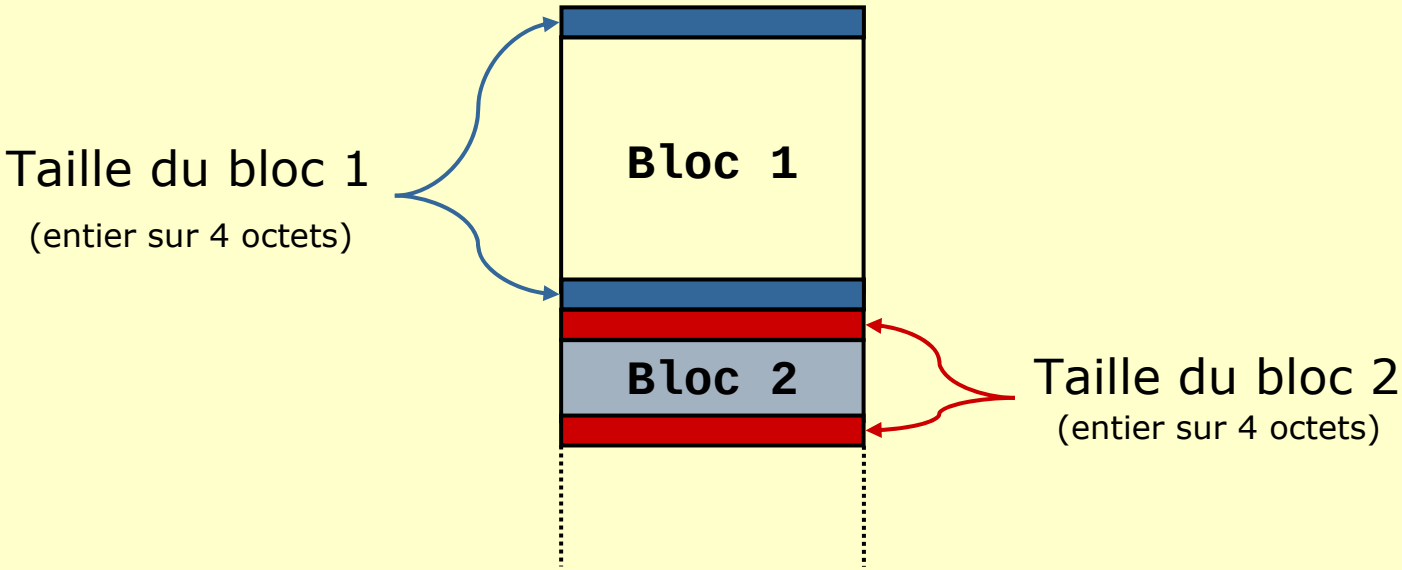
- Comment utiliser un fichier binaire?
  - il faut que le format du fichier soit documenté en détail…
  - …ou disposer du source d'un programme permettant de lire/écrire ce format
  - il faut aussi connaître le type de processeur sur lequel le fichier a été généré pour connaître l'ordre de stockage des octets dans le fichier
    - Intel/AMD, Alpha (DEC) → binaire *little endian*
    - Tous les autres processeurs → binaire *big endian*
    - On passe du *big endian* au *little endian* en retournant l'ordre des octets qui constituent chaque nombre



Fichiers fortran *unformatted* à accès séquentiel…

Fichier binaire fortran *unformatted* à accès séquentiel

- Chaque opération **`WRITE(unit)`** ajoute un bloc de données au fichier (bloc de $\mathbf{nb_i}$ octets)
- Chacun des blocs de données est précédé ET suivi par l'entier $\mathbf{nb_i}$ codé sur 4 octets



Exemple de programme fortran (py_jyp_ex_53.f90)

Ecriture d'un tableau dans un fichier binaire, de 3 façons différentes

```
PROGRAM py_jyp_ex_53

  INTEGER, PARAMETER :: nblignes = 2, nbcolonnes = 3
  INTEGER i, j
  REAL*4, DIMENSION(nbcolonnes, nblignes) :: a

  a(:, :) = 0.
  a(1, :) = 1.
  a(3, :) = 3.
  DO j = 1, nblignes
    WRITE(*, *) a(:, j)
  ENDDO

  OPEN(10, file='a_brut.dat', form='unformatted')
  WRITE(10) a
  CLOSE(10)

  OPEN(10, file='a_entete.dat', form='unformatted')
  WRITE(10) 'a_entete.dat', nbcolonnes, nblignes, a
  CLOSE(10)

  OPEN(10, file='a_entete_multwrite.dat', form='unformatted')
  WRITE(10) 'a_entete.dat'
  WRITE(10) nbcolonnes
  WRITE(10) nblignes
  WRITE(10) a
  CLOSE(10)

END PROGRAM py_jyp_ex_53
```

```
1.000000      0.0000000E+00   3.000000
1.000000      0.0000000E+00   3.000000
```

Initialisation et affichage du tableau à écrire dans les fichiers

Ecriture *simple* du tableau **a**

Ecriture en une seule fois de **a**, précédé par un entête

Ecriture du fichier binaire avec entête, en plusieurs fois

Analyse des 3 fichiers binaires (1/2)

- Commande **od** (*object dump*)
  - od –I fichier → affichage des entiers sur 4 octets
  - od –f fichier → affichage des réels sur 4 octets
  - od –c fichier → affichage des caractères
- Commande **strings**
  - affichage uniquement des caractères visibles

24 octets (*tableau* ***a***) + 2 entiers sur 4 octets (binaire fortran…) = 32 octets

24 octets + 12 octets (*'**a_entete.dat**'*) + 2 entiers sur 4 octets (***nbcolonnes, nblignes***) + 2 entiers sur 4 octets = 32 octets

```
> ls -l a_*.dat
       32 a_brut.dat
       52 a_entete.dat
       76 a_entete_multwrite.dat
```

2 * 3 points * 4 octets = 24 octets

```
> od -f a_brut.dat
0000000   3.363116e-44   1.000000e+00   0.000000e+00   3.000000e+00
0000020   1.000000e+00   0.000000e+00   3.000000e+00   3.363116e-44

> od -l a_brut.dat
0000000           24  1065353216            0  1077936128
0000020  1065353216            0  1077936128           24
```

Analyse des 3 fichiers binaires (2/2)

```
> od -f a_entete.dat
0000000   6.165713e-44   1.774684e+28   7.213306e+22   7.142936e+31
0000020   4.203895e-45   2.802597e-45   1.000000e+00   0.000000e+00
0000040   3.000000e+00   1.000000e+00   0.000000e+00   3.000000e+00
0000060   6.165713e-44

> od -l a_entete.dat
0000000          44  1852137313  1702126964  1952539694
0000020           3           2  1065353216           0
0000040  1077936128  1065353216           0  1077936128
0000060          44

> od -c a_entete.dat
0000000   ,  \0  \0  \0   a   _   e   n   t   e   t   e   .   d   a   t
0000020 003  \0  \0  \0 002  \0  \0  \0  \0  \0 200   ?  \0  \0  \0  \0
0000040  \0  \0   @   @  \0  \0 200   ?  \0  \0  \0  \0  \0  \0   @   @
0000060   ,  \0  \0  \0

> strings a_entete.dat
a_entete.dat
```

```
> od -f a_entete_multwrite.dat
0000000   1.681558e-44   1.774684e+28   7.213306e+22   7.142936e+31
0000020   1.681558e-44   5.605194e-45   4.203895e-45   5.605194e-45
0000040   5.605194e-45   2.802597e-45   5.605194e-45   3.363116e-44
0000060   1.000000e+00   0.000000e+00   3.000000e+00   1.000000e+00
0000100   0.000000e+00   3.000000e+00   3.363116e-44

> od -l a_entete_multwrite.dat
0000000          12  1852137313  1702126964  1952539694
0000020          12           4           3           4
0000040           4           2           4          24
0000060  1065353216           0  1077936128  1065353216
0000100           0  1077936128          24
```

Lecture de a_brut.dat

(py_jyp_ex_53_a_brut.py)

*Lecture du fichier binaire **a_brut.dat***

```
#!/usr/bin/env python

import Numeric as N

f = open('a_brut.dat', 'rb')

donnee_brute = f.read(4)
longueur_octets = N.fromstring(donnee_brute, N.Int32)
print longueur_octets
longueur_octets = longueur_octets[0]
print longueur_octets

donnee_brute = f.read(longueur_octets)
A = N.fromstring(donnee_brute, N.Float32)
print A
A.shape = (2, 3)
print A

donnee_brute = f.read(4)
longueur_octets = N.fromstring(donnee_brute, N.Int32)
print longueur_octets

f.close()
```

Ouverture en mode *lecture binaire*

Lecture des 4 premiers octets, et conversion en un nombre entier

Lecture du tableau

On donne sa forme au tableau. Attention, on est obligé de connaître la forme du tableau, qui n'est pas indiquée dans le fichier, et il faut permuter l'ordre des dimensions par rapport au fortran!

```
[24]
24
[ 1.  0.  3.  1.  0.  3.]
[[ 1.  0.  3.]
 [ 1.  0.  3.]]
[24]
```

Lecture de a_entete.dat

(py_jyp_ex_53_a_entete.py)

Lecture du fichier binaire ***a_entete.dat***

```
#!/usr/bin/env python

import Numeric as N

f = open('a_entete.dat', 'rb')

longueur_octets = N.fromstring(f.read(4), N.Int32)[0]
print longueur_octets

nom = f.read(12)
print nom

nbcolonnes = N.fromstring(f.read(4), N.Int32)[0]
nblignes = N.fromstring(f.read(4), N.Int32)[0]
print nbcolonnes, nblignes

A = N.fromstring(f.read(nbcolonnes*nblignes*4), N.Float32)
A.shape = (nblignes, nbcolonnes)
print A

f.close()
```

Lecture des 4 premiers octets, et conversion en un nombre entier

Lecture de l'entete (nom, nombre de lignes et de colonnes). Note, on est obligé de connaître la longueur du nom de fichier dans l'entête! Ce serait plus pratique que le nom soit contenu dans un champ de longueur fixe, complété par des espaces ☺

Lecture du tableau

```
44
a_entete.dat
3 2
[[ 1.  0.  3.]
 [ 1.  0.  3.]]
```

Remarques sur la lecture binaire

- Binaire fortran *unformatted* séquentiel : si on connaît *à priori* la structure du fichier et la taille des blocs, on n'a pas besoin de l'information de longueur de début/fin de bloc

  Il suffit de la lire pour la sauter (**`f.read(4)`**)

- Cas général du binaire (non fortran) : pas d'information de longueur à lire

  c.-à-d. les programmes de lecture sont les mêmes, sans instructions **`f.read(4)`**

- Fichier binaire généré sur une machine qui n'a pas le même *endian* que la machine sur laquelle on exécute le script

  Il suffit de changer l'ordre des octets après la lecture!

```
N.fromstring(f.read(taille), N.Float32).byteswapped()
```

Ecriture binaire en python (py_jyp_ex_54_a_brut.py)

Ecriture en python d'un fichier binaire identique au fichier a_brut.dat généré par **py_jyp_ex_53.f90**

```
#!/usr/bin/env python

import Numeric as N

nbcolonnes, nblignes = 3, 2
a = N.zeros((nblignes, nbcolonnes), N.Float32)
a[:, 0] = 1.
a[:, 2] = 3.
print a

f = open('a_brut_python.dat', 'wb')

longueur_octets = N.array(nblignes * nbcolonnes * a.itemsize(),
                          N.Int32)

f.write(longueur_octets.tostring())
f.write(a.tostring())
f.write(longueur_octets.tostring())

f.close()

# The end
```

```
[[ 1.  0.  3.]
 [ 1.  0.  3.]]
```

Initialisation du tableau

Ouverture en mode *écriture binaire*

Ecriture des objets, après les avoir convertis en une suite d'octets…

Note : le fichier a le même type d'endian que la machine sur laquelle on exécute le script

Valeurs manquantes → tableaux masqués!

- Pourquoi des *masques*?
  - valeur *masquée* ⇔ valeur manquante (*missing value*) ⇔ valeur non définie
  - objectifs :
    - pouvoir continuer à travailler avec des matrices dont toutes les valeurs ne sont pas définies
    - masquer temporairement des parties de matrices, pour qu'elles ne soient pas modifiées par les opérations qui suivront → sélection de zones
- Les fonctionnalités standard (de **`Numeric`**) sont disponibles dans le module *Masked Array*

  `import MA` (au lieu de **`import Numeric`**)

  `N.zeros(shape)` → **`MA.zeros(shape)`**
- Les opérations sont optimisées, mais un peu moins rapides qu'avec des tableaux Numeric…

Comment se présente un tableau masqué?

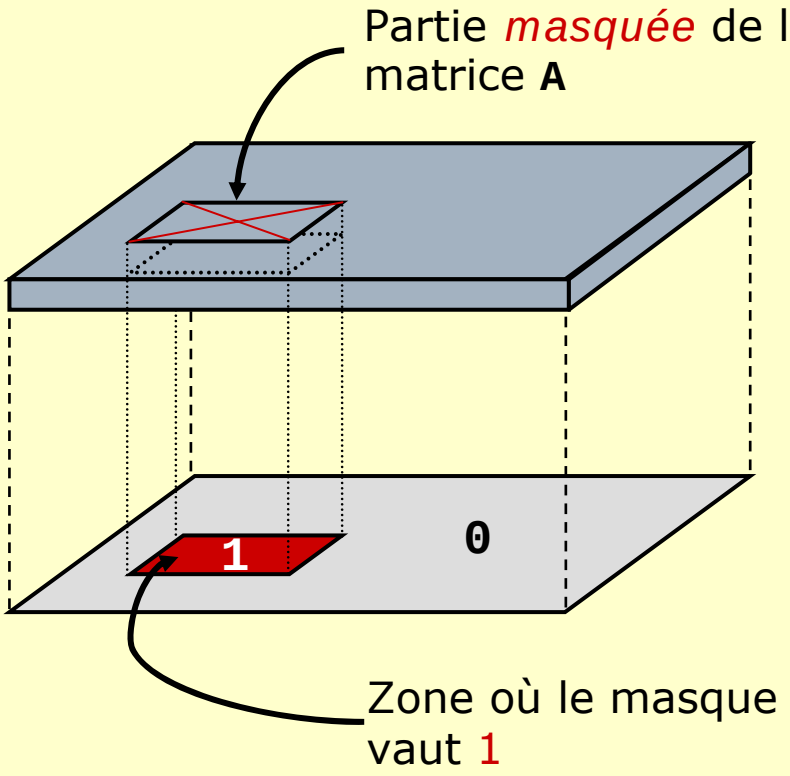
Un tableau masqué est la combinaison :

- d'un tableau **`Numeric`** *normal*
- d'un tableau de même forme définissant le masque
  (***`1`*** là où les éléments sont masqués, ***`0`*** sinon)
  - Note : la taille du *tableau de masque* est *petite*
    (1 octet par élément ⇔ **`N.Int8`** ⇔ **`'1'`**)
- d'une valeur de remplissage (*fill value*)

`A` ***`n*m`***

Masque de **`A`**

→ **`A.mask()`** ***`n*m`***



Travailler avec un tableau masqué (1/2)

- Description détaillée de MA dans la partie II.20 de **`numpy.pdf`** (*Masked arrays*)
- Tout ce qui était faisable avec **`Numeric`** le reste avec **`MA`**
- Dans une opération combinant des opérandes masqués, le résultat est masqué comme les opérandes (union des masques)
- Récupération du masque :

  `m = A.mask()`

  Le masque ***`m`*** vaut ***`None`*** si **`A`** ne contient pas d'élément masqué (plutôt que d'avoir un masque rempli de zéros…)
- Nombre d'éléments **`non`** masqués :

  `nb = A.count()`

  - astuce : *comptage* du nombre de valeurs strictement positives d'un tableau. On masque les valeurs négatives ou nulles et on compte le nombre de points non masqués! ☺

```
Apos = MA.masked_less_equal(A, 0.)
nb_pos = Apos.count()
```

Travailler avec un tableau masqué (2/2)

- Suppression des éléments masqués :
  `A_ok = A.compressed()`
  On récupère un tableau **`Numeric`** 1D (sans masque…) de **`A.count()`** éléments
  - astuce : **`compressed()`** peut permettre d'accélérer des opérations en réduisant le nombre de points à traiter, ou en évitant de devoir faire des boucles avec des tests ☺
    Exemple : somme des éléments strictement positifs d'un tableau
    ```
    Apos = MA.masked_less_equal(A, 0.)
    sum_pos = Numeric.sum(Apos.compressed())
    ```
- Détermination/positionnement de la valeur de remplissage
  - **`A.fill_value()`**
  - **`A.set_fill_value(valeur)`**
    Note : valeur de remplissage par défaut :
    - Tableaux de **`Int32`** : **`0`**
    - Tableaux de **`Float32`** et **`Float64`** : **`1.0e+20`**
- Suppression du masque :
  Permet de passer un objet de **`MA`** vers **`Numeric`**
  - **`B = A.filled(`*`valeur`*`)`**
    Tous les éléments masqués de **`A`** sont remplacés par **`valeur`**, ou par *fill_value*, si **`valeur`** n'est pas spécifiée
  - astuce : mise à zéro des valeurs comprises dans un intervalle (attention, ne marche correctement que sur un tableau qui ne contient pas déjà des valeurs masquées!)
    ```
    B = MA.masked_inside(A, borne_inf, borne_sup).filled(0)
    ```
- La méthode **`unmask()`** (pour gagner un peu de place mémoire):
  - si un tableau **`A`** de **`MA`** est associé à un tableau de masque dont toutes les valeurs sont à zéro, **`unmask()`** remplace ce tableau par **`None`**, et **`A`** est toujours un objet de **`MA`**.
  - si **`A`** contient des valeurs masquées, il ne se passe rien…
  - **`A.unmask()`** est une méthode *in-place*! **`A.unmask()`** ne retourne pas de valeur, et le masque de **`A`** est directement modifié

Comment masquer des éléments?

- Masquage d'éléments dont on connaît les indices :
  Il suffit de leur donner la valeur **`MA.masked`**!
  ```
  A[:, 1] = MA.masked
  ```
- Masquage en fonction des valeurs :
  - Masquage d'éléments plus grands qu'un seuil :
    ```
    B = MA.masked_greater(A, seuil)
    ```
  - **`masked_greater_equal`**, ***`masked_equal`***, **`masked_not_equal`**, **`masked_less`**, **`masked_less_equal`**
    - Pour masquer des **`N.Float`*`xx`*** égaux à une certaine valeur, il vaut mieux utiliser **`masked_values`** que **`masked_equal`**
  - Masquage à l'intérieur/extérieur d'un intervalle :
    ```
    masked_inside(A, borne1, borne2)
    masked_outside(A, borne1, borne2)
    ```
- Masquage en fonction d'une condition :
  ```
  B = MA.masked_where(condition, A)
  ```
  Le tableau **`B`** est égal au tableau **`A`**, masqué là où les éléments de **`condition`** valent **`True`** (**`1`**)… et là où **`A`** est déjà masqué.

Manipulation des masques

- On peut manipuler des masques, puis masquer un tableau avec le masque ***m*** résultant :
  - **`B = MA.array(A, m, copy=1)`**
  - Si on n'a pas besoin de copier le tableau :
    `A = MA.masked_array(A, m)`
- Récupération d'un masque : **`m = A.mask()`**
- Les masques sont des tableaux de **`0`** et de **`1`**, entre lesquels on peut faire des *opérations logiques* standard :
  - Union de deux masques :
    `m = MA.mask_or(m1, m2)`
    ou **`m = N.logical_or(m1, m2)`**
  - Autres opérations logiques :
    `logical_and(m1, m2)`, **`logical_not(m1)`**, **`logical_xor(m1, m2)`**

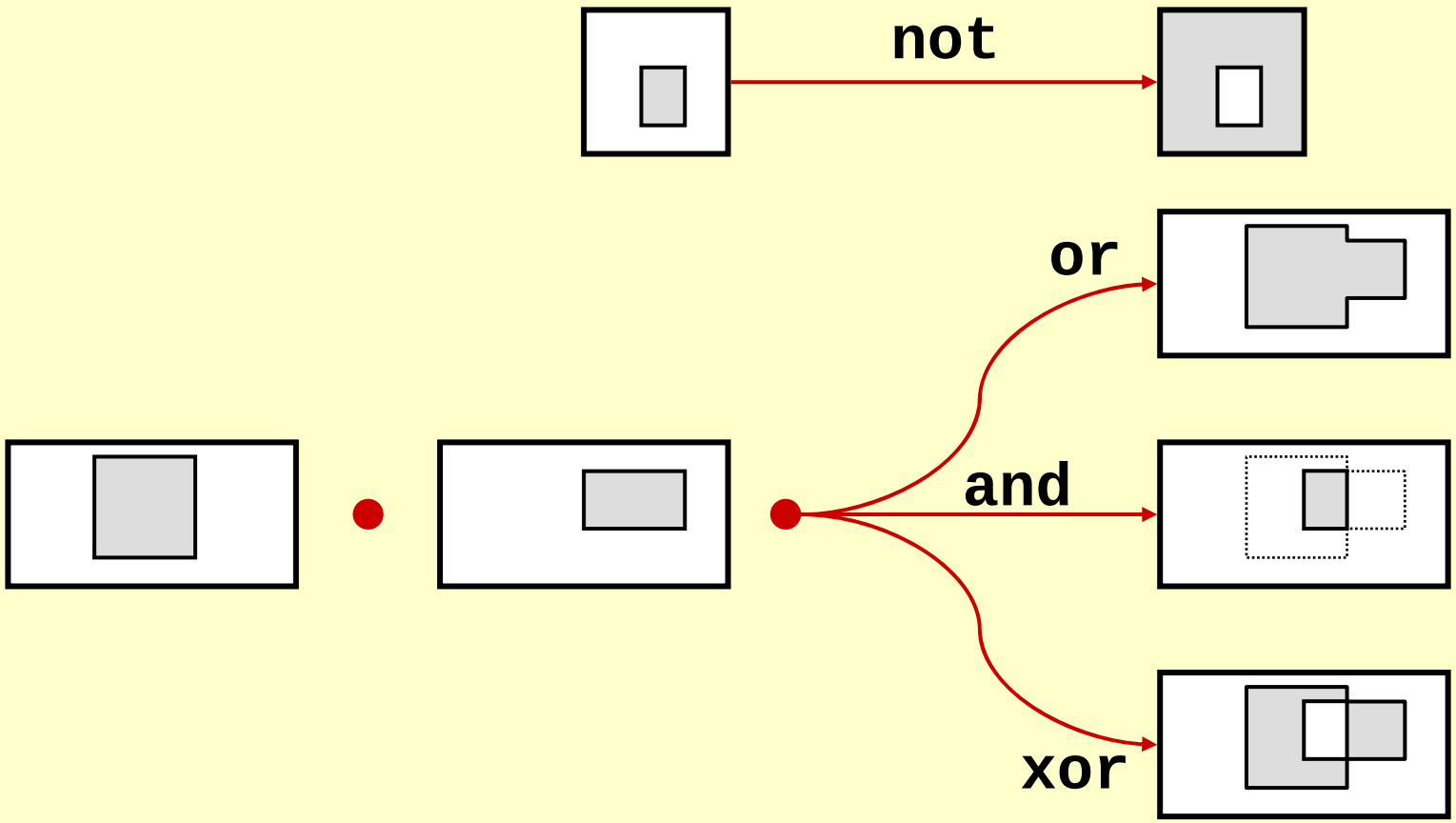
XOR	0	1
0	0	1
1	1	0

- Note : création d'un masque vide, d'une forme donnée (ex : la forme de **`A`**)
  `m = MA.make_mask_none(A.shape)`

Opérations logiques sur les masques

Elément non masqué **0**

Elément masqué **1**



Les masques, par l'exemple

(py_jyp_ex_55.py)

```
import Numeric as N, MA
```

```
[[-999,  -1,   2,]
 [  -1,   2,-999,]
 [  -1,   2,-999,]]
```

```
A = N.array(((-999, -1, 2), (-1, 2, -999), (-1, 2, -999)), N.Int32)
```

```
B = MA.masked_values(A, -999)
```

```
[[-- ,-1 ,2 ,]
 [-1 ,2 ,-- ,]
 [-1 ,2 ,-- ,]]
```

```
print B.count()
```

```
6
```

```
B[1,1] = MA.masked
```

```
[[-- ,-1 ,2 ,]
 [-1 ,-- ,-- ,]
 [-1 ,2 ,-- ,]]
```

```
print B.mask()
```

```
[[1,0,0,]
 [0,1,1,]
 [0,0,1,]]
```

```
print B.compressed()
```

```
[-1, 2,-1,-1, 2,]
```

```
B += 10
```

```
C = B.filled(-10)
```

```
[[-10,  9, 12,]
 [  9,-10,-10,]
 [  9, 12,-10,]]
```

```
m = MA.make_mask_none(C.shape)
m[:, 1] = 1
```

```
[[0,1,0,]
 [0,1,0,]
 [0,1,0,]]
```

```
C = MA.masked_array(C, MA.mask_or(m, B.mask()))
```

```
[[-- ,-- ,12 ,]
 [9 ,-- ,-- ,]
 [9 ,-- ,-- ,]]
```

Améliorer les performances… (1/2)

- **Considérations générales** :
  - si le programme est bien écrit, les performances seront normalement très acceptables
  - il faut (souvent) choisir entre avoir un programme rapide qui consomme beaucoup de mémoire, ou un programme un peu plus lent
    → Il vaut mieux essayer de bien gérer la mémoire
  - il faut que le programme reste compréhensible, même après optimisation!
    → Ne pas hésiter à mettre des commentaires, à indiquer les indices muets (':'), bien choisir les noms des variables, etc… ☺

- Du bon sens dans la gestion de la mémoire!
  - ne pas utiliser des tableaux plus grands que nécessaire (i.e. est-ce nécessaire de charger tout un tableau en mémoire, ou peut-on le traiter progressivement?)
  - effacer explicitement les gros tableaux qui ne servent plus avec **`del(tableau)`**
  - Ne pas oublier que certaines opérations nécessitent la création de tableaux intermédiaires pour les calculs…
    - Utiliser (éventuellement) des opérations *in-place* (mais cela peut nuire à la lisibilité d'un programme…)
      `A = A – 1` ➔ `N.subtract(A, 1, A)` ⇔ `A -= 1`
- Toujours utiliser la *syntaxe tableau* plutôt que des boucles

  a = N.arange(10) → **`[0,1,2,3,4,5,6,7,8,9,]`**
  - Eviter les boucles explicites

```
b = N.zeros((9,))
for i in range(len(a)-1):
    b[i] = a[i] - a[i+1]
```

  - Bonne approche :

```
b = a[:-1]-a[1:] → [-1,-1,-1,-1,-1,-1,-1,-1,-1,]
```

Améliorer les performances… (2/2)

- Utiliser des masques, des **choose** et des **where** plutôt que des boucles avec des opérations conditionnelles…

```
A = N.array(((0, -1, 2), (-1, 2, -3), (-1, 2, 3)), N.Int32)
```

  - Mauvaise approche pour calculer la moyenne des éléments positifs ou nuls de A :

```
tot = nb = 0
for j in range(3):
    for i in range(3):
        val = A[j, i]
        if val >= 0:
            tot += val
            nb += 1
print float(tot)/nb
```

→ 1.8

  - Bonne approche :

```
B = MA.masked_less(A, 0)
tot = MA.sum(B.flat)
nb = B.count()
print float(tot)/nb
```

  - Encore mieux!

```
MA.average(B.flat)
```

```
A →  [[ 0,-1, 2,]
      [-1, 2,-3,]
      [-1, 2, 3,]]
```

```
B →  [[0 ,-- ,2 ,]
      [-- ,2 ,-- ,]
      [-- ,2 ,3 ,]]
```

- Eviter les masques, lorsqu'ils ne sont pas nécessaires

  → les opérations sur des tableaux de **MA** sont un peu plus lentes que celles sur les tableaux de **Numeric**…

Attentions! Erreurs à éviter (1/…)

- Si **A** est un tableau et **B=A**, **B** est une *référence* à la même zone mémoire que **A**! Pour que **B** soit effectivement une *copie* de **A**, utiliser :
  - **`B = N.array(A)`**
  - Le comportement pointeur peut être utile si A est un tableau à plusieurs dimensions et que l'on veut temporairement le considérer comme un tableau à 1 dimension!
    - B = A.flat
- Les 3 lignes suivantes sont équivalentes et créent une *référence* au tableau A, pas une copie!
  - **`B = A`**
  - **`B = N.asarray(A)`**
  - **`B = N.array(A, copy=0)`**
- L'objet à convertir en tableau doit être le premier paramètre de **`N.array`**
  - **`N.array(1, 2, 3)`** → ☹
  - **`N.array((1, 2, 3))`** → OK!

Attentions! Erreurs à éviter (2/…)

- **`len(A)`** donne la taille de la première dimension (**`len(A)` ⇔ `A.shape[0]`**), pas le nombre d'éléments du tableau!

  nombre d'éléments d'un tableau :

  `len(A.flat)`

  ou **`Numeric.product(A.shape)`**

- Ne faut pas oublier de parenthèses dans certaines fonctions
  - **`N.ones((o, n, m))`**
  - **`N.concatenate((`A_0`, …, `A_{N-1}`))`**
  - **`N.choose(select, (`A_0`, `A_1`, …, `A_{N-1}`))`**

Trucs et astuces de **`Numeric`** (1/…)

- Conversion d'un tableau en une liste
  - **`A.tolist()`** (c'est la réciproque de **`N.array(liste)`**)
    - **`N.arange(0, 0.6, 0.1).tolist()`**
    - → **`[0.0, 0.1, 0.2, 0.3, 0.4, 0.5]`**

Trucs et astuces de **Numeric** (2/…)

- Indices des éléments d'un tableau satisfaisant à une condition (pour les utilisateurs de ***where*** dans IDL, par exemple ☺)
  - **`N.nonzero(A_1d)`**
    → donne les indices des éléments non nuls d'une matrice A à une dimension
  - **`A == valeur`** (ou autre condition)
    → renvoie une matrice avec des **`1`** là où la condition est satisfaite (et **`0`** sinon)
  - **`ind_flat = N.nonzero((A == valeur).flat)`**
    → renvoie les indices 1d où la condition est satisfaite!
    - Si **`A`** a 1 dimension : **`ind_flat`** est le tableau des indices!
    - Si **`A`** a 2 dimensions : on utilise la *division entière* (**`a<b ➔ a/b = 0`**), le modulo, et le fait que : **`indice = ligne * nb_col + colonne`**

      ```
      tab_ligne   = ind_flat / nb_col
      tab_colonne = ind_flat % nb_col
      ```
  - exemple :

    ```
    >>> A = N.array(((1, 2, 3, 4), (3, 4, 1, 2), (4, 1, 2, 3)),
                    N.Float32)
          A →          [[ 1.,  2.,  3.,  4.],
                        [ 3.,  4.,  1.,  2.],
                        [ 4.,  1.,  2.,  3.]],'f')
    >>> ind_flat = Numeric.nonzero((A==2.).flat)
    >>> lignes   = ind_flat / A.shape[1]      → [0, 1, 2]
    >>> colonnes = ind_flat % A.shape[1]      → [1, 3, 2]
    ```
  - note : on peut aussi faire la même chose avec des masques

Trucs et astuces de **MA** (1/...)

- Indices des éléments d'un tableau satisfaisant à une condition (pour les utilisateurs de ***where*** dans IDL, qui ont besoin des indices explicites)

```
A = N.array(((1, 2, 3, 4), (3, 4, 1, 2), (4, 1, 2, 3)), N.Float32)
            A →             [[ 1.,  2.,  3.,  4.],
                             [ 3.,  4.,  1.,  2.],
                             [ 4.,  1.,  2.,  3.]],'f')
```

1. on masque les éléments auxquels on ne s'intéresse pas

```
A_mask = MA.masked_not_equal(A, 2.).mask()→          [[1,0,1,1,]
                                                      [1,1,1,0,]
                                                      [1,1,0,1,]]
```

2. on applique ce masque à un tableau qui contient tous les indices des cases

```
A_indices = N.indices(A.shape)              →        [[[0,0,0,0,]
                                                      [1,1,1,1,]
                                                      [2,2,2,2,]]
                                                     [[0,1,2,3,]
                                                      [0,1,2,3,]
                                                      [0,1,2,3,]]]
nb_dims = N.rank(A)
A_indices_masked = MA.masked_array(A_indices, N.resize(A_mask, (nb_dims,) + A.shape))
```

3. on supprime les indices masqués

```
A_indices_compressed = A_indices_masked.compressed()
                                            →        [0,1,2,1,3,2,]
A_indices_compressed.shape = (nb_dims, -1) →         [[0,1,2,]
                                                      [1,3,2,]]
lignes   = A_indices_compressed[0]          →        [0,1,2,]
colonnes = A_indices_compressed[1]          →        [1,3,2,]
```

CDAT et le langage python

Gestion des variables et des Entrées/Sorties avec…

… le module **`cdms`**

(et les modules **`MV`**, **`cdtime`**, **`cdutil`**, **`genutil`**, **`…`**)

A quoi sert **cdms**?

- Le module **cdms** permet très facilement de :
  - lire des fichiers de données au format NetCDF et grads/grib (avec un fichier '**.ctl**' approprié)

    Note : les fichiers NetCDF peuvent être sur un serveur OPeNDAP/DODS distant!
    - Accès aux variables
    - Accès aux métadonnées des variables
      - nom, unités et autres attributs
      - axes
  - créer des fichiers NetCDF (*uniquement NetCDF…*)
  - modifier des variables dans des fichiers NetCDF existants
  - *changement de grilles, grilles irrégulières*, …
  - …

- Documentation détaillée dans **cdms.pdf**
  - Et résumé dans **cdms_quick_start.pdf**

Utiliser la documentation de **cdms**…

- C'est plus un manuel de référence que d'utilisation…
  → Il faut commencer par lire cette présentation et **cdms_quick_start.pdf** ☺
- Les sections du manuel de **cdms** sont organisées en fonctions des objets créés par **cdms**
  - 1 Introduction et exemples
  - 2.3 Les méthodes de **cdms** (**cdms.xxx()**, fonctions de création d'axes et des grilles, …)
  - 2.5 Les axes (création) → voir aussi 2.3
  - 2.6 Les objets fichiers (lecture et écriture des variables)
  - 2.9 Le module **MV** (la même chose que **Numeric** et **MA**…)
  - 2.10 Les grilles → voir aussi 2.3
  - 2.11 Les variables (création et utilisation)
    - Table 2.34 Création (**cdms.createVariable()**)
    - Table 2.35 Utilisation des variables (et récupération des axes)
    - 2.11.1 Les sélecteurs (comment spécifier les *zones* à lire)
  - 3 Le module **cdtime** (gestion/conversion du temps)
  - …

Qu'est-ce qu'un fichier NetCDF?

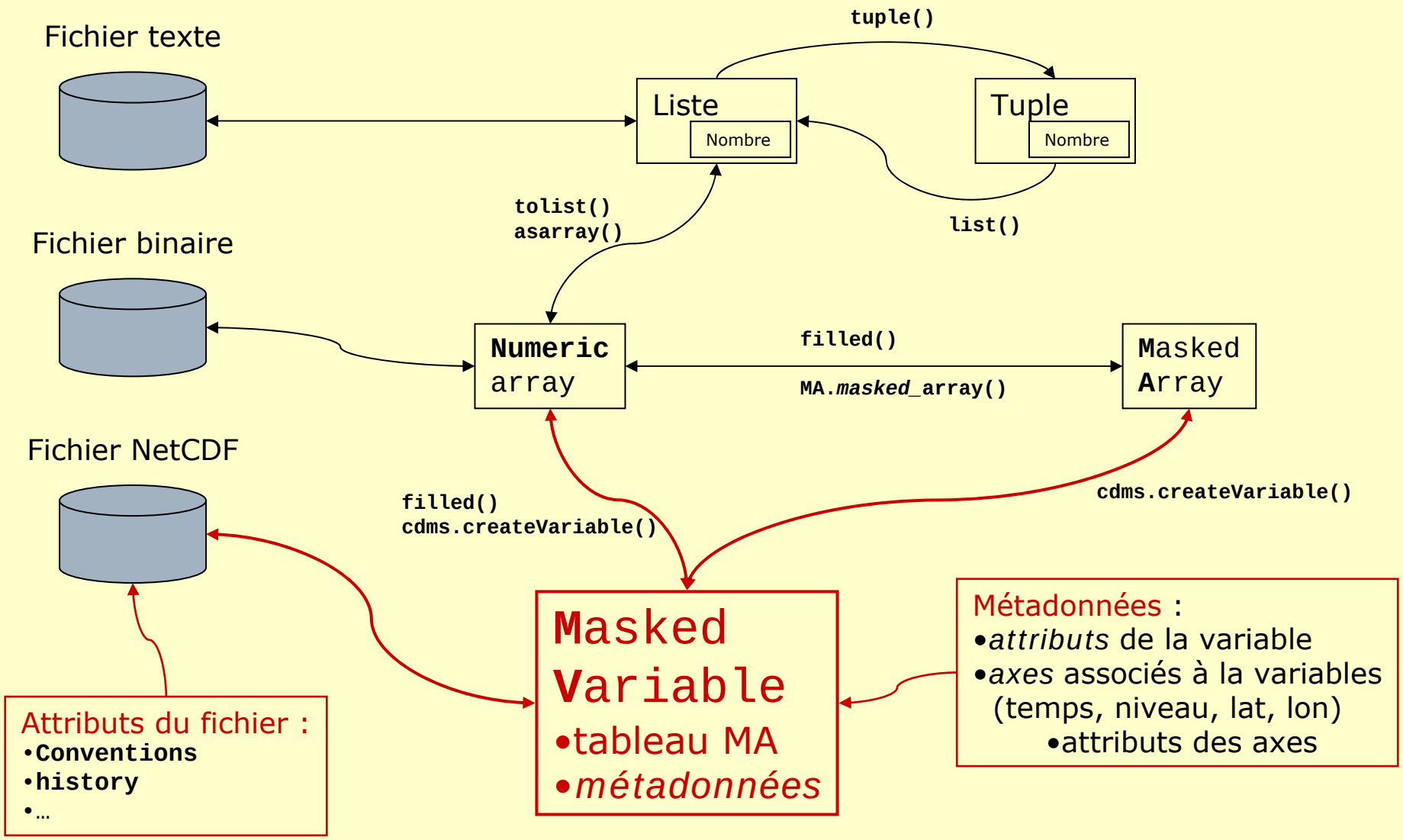
- C'est un fichier binaire qui :
  - est au format NetCDF (lu/écrit avec la librairie NetCDF)… ☺
    - fichier auto-documenté → pas besoin d'avoir de connaissances *à-priori* sur le fichier pour pouvoir utiliser son contenu!
    - contenu lisible sur n'importe quel type de processeur…
      norme XDR → pas de problème de *little/big endian*!
    - Accès de base au contenu avec :
      `ncdump [-h] fichier.nc | more`
  - suit éventuellement la convention CF
    - ***C****limate and* ***F****orecast* (anciennement convention *GDT*)
    - objectif : faciliter l'échange et la comparaison des données, faciliter la compréhension des données par les programmes et les utilisateurs
    - convention : les variables et les axes doivent avoir un attribut **`units`**, il est recommandé d'utiliser des noms standards pour les variables, et de leur rajouter un attribut **`standard_name`**, …

      Exemple : variable **`rls`**

      `units` → **`W m-2`**

      `standard_name` → **`surface_net_downward_longwave_flux`**
- Plus de renseignements :
  - http://www.unidata.ucar.edu/software/netcdf/
  - http://www.cgd.ucar.edu/cms/eaton/cf-metadata/
    http://www.cgd.ucar.edu/cms/eaton/cf-metadata/daresburytalk.pdf

Tout ça pour arriver aux... variables **MV**!



Opérations sur les fichiers

- Ouverture d'un fichier :
  `fic = cdms.open('nom_fichier', mode)`
  Modes d'accès :
  - **`'r'`** = lecture seule (par défaut)
  - **`'w'`** = écriture (création d'un fichier)
  - **`'a'`** ou **`'r+'`** = mise à jour d'un fichier (avec des *file variables*, ou pour rajouter des pas de temps)

  Fichiers OPeNDAP/DODS (accès à des fichiers distants en *lecture*) :
  `fic = cdms.open('http://nom_serveur/cgi-bin/nph-dods/chemin/nom_fichier.nc')`
- Fermeture : **`fic.close()`**
  Attention! Ne jamais oublier de **`fermer`** les fichiers ouverts avec cdms!
- Lecture d'attributs globaux, en mode **`'r'`**
  `print fic.Conventions, fic.history`
- Création d'attributs en mode **`'w'`** (et **`'r+'`**?)
  `fic.nom_attribut = valeur` ☺
  Exemple : **`fic.history = 'Created by %s (%s)' % (user, date)`**
- Informations sur le contenu :
  - **`fic.attributes.keys()`** → liste des attributs globaux du fichier
  - **`fic.listvariables()`** → contenu du fichier
  - **`fic.axes.keys()`** → liste des axes
    `lon_ax = fic.getAxis('longitude')`
    ou **`lon_ax = fic.axes['longitude']`**
- Ecriture d'une variable dans un fichier : **`fic.write(variable_MV)`**
  Note : voir aussi l'ajout de pas de temps à un fichier, et l'utilisation d'une *file variable* pour la modification d'une variable directement dans un fichier

Lecture d'une variable

- Une variable est un objet du module **MV** (**M**asked **V**ariables)
- Chargement d'une variable en mémoire :

  var = fic(nom_var *[, sélection]*)

  - Par défaut, chargement de toute la variable (attention aux *grosses* variables!), avec les mêmes axes que dans le fichier
  - On peut aussi utiliser un *pointeur* (*file variable*) sur une variable dans le fichier!
- Spécification (restriction à un intervalle) des axes :
  - un axe pour lequel on ne spécifie rien sera lu entièrement
  - il est possible de spécifier ce que l'on veut lire pour un ou plusieurs axes

    var = fic('tas', time=slice(2, 5), latitude=(-90, 90), longitude=(-180, 180))
    - spécification en coordonnées :

      nom_axe=(coord_début, coord_fin [, 'co'])
      - voir plus loin pour des détails sur la gestion du temps
      - voir la doc de **MapIntervalExt** dans **cdms.pdf** (table 2.10 de la section 2.5) pour plus de détails sur la façon de spécifier une zone (**'co'** et ses variantes…)
    - spécification en indices :

      … en n'oubliant pas que les indices commencent à **0**!

      nom_axe=slice(indice_début, indice_fin_exclu, pas)

      Note : retournement d'un axe avec **slice(None, None, -1)** ☺
    - suppression des dimensions de taille 1 (dim *unitaires*) :

      Il faut rajouter un mot-clé : **squeeze=1**

      var = fic('tas', time=slice(2, 3)) → **var.shape = (1, 50, 64)**

      var = fic('tas', time=slice(2, 3), squeeze=1) → **var.shape = (50, 64)**

Les *sélecteurs* de zone

Autres façons d'écrire :

```
var = fic('tas', time=slice(2, 5), latitude=(-90, 90),
    longitude=(-180, 180))
```

- Les sélecteurs permettent de créer des zones prédéfinies, utilisables pour lire une variable :

```
from cdms.selectors import Selector
tsel = Selector(time=slice(2, 5))
zonesel = Selector(latitude=(-90, 90), longitude=(-180, 180))
```

→ **`var = fic('tas', tsel, zonesel)`**

- Les sélecteurs peuvent être combinés!

```
combisel = tsel & zonesel
```

→ **`var = fic('tas', combisel)`**

- Remarque : les noms des axes…
  - On peut spécifier un axe en donnant son *nom générique* (ex : **`time`**) ou son nom exact dans le fichier (ex : **`time_counter`**)!
  - Si on utilise le nom générique, **`cdms`** arrivera à reconnaître l'axe dans le fichier si son nom est suffisamment standardOn peut aider **`cdms`** en lui indiquant des synonymes pour les noms des axes

```
cdms.axis.latitude_aliases.append('y1')
```

Utilisation d'une variable

- Informations sur une variable (en mode intéractif) :
  `var.info()`
- Manipulation numérique :
  Tout se passe comme pour une variable de **`MA`**! ☺
  - opérations de *slicing* et *indexing* standards (**`var[]`**)
  - pour conserver les métadonnées d'une variable (les axes, en particulier), il faut utiliser les fonctions de **`MV`** :
    `import cdms` ou **`import MV`**
    `MA.average()` ⇔ **`cdms.MV.average()`** ⇔ **`MV.average()`**
    Attention, pour calculer une moyenne *pondérée* par la surface des mailles, il faut utiliser **`cdutil.averager()`** !
- Liste des attributs d'une variables : **`var.attributes.keys()`**
- Lecture d'un attribut : **`var.`*`nom_attribut`***
  `u = var.units` (ou **`var.attributes['units']`**)
- Modification/création d'un attribut :
  `var.nom_attribut = valeur`
- Copie d'une variable :
  `B = A[…]` ou **`B = A.clone()`**
- Restriction de la zone *à posteriori* (après lecture) :
  - en plus du slicing/indexing (**`var[]`**), il est possible de restreindre la zone sur laquelle la variable est définie
    `var_NH = var(latitude=(0, 90))`
  - note : on peut aussi faire un **`squeeze=1`** à posteriori! ☺

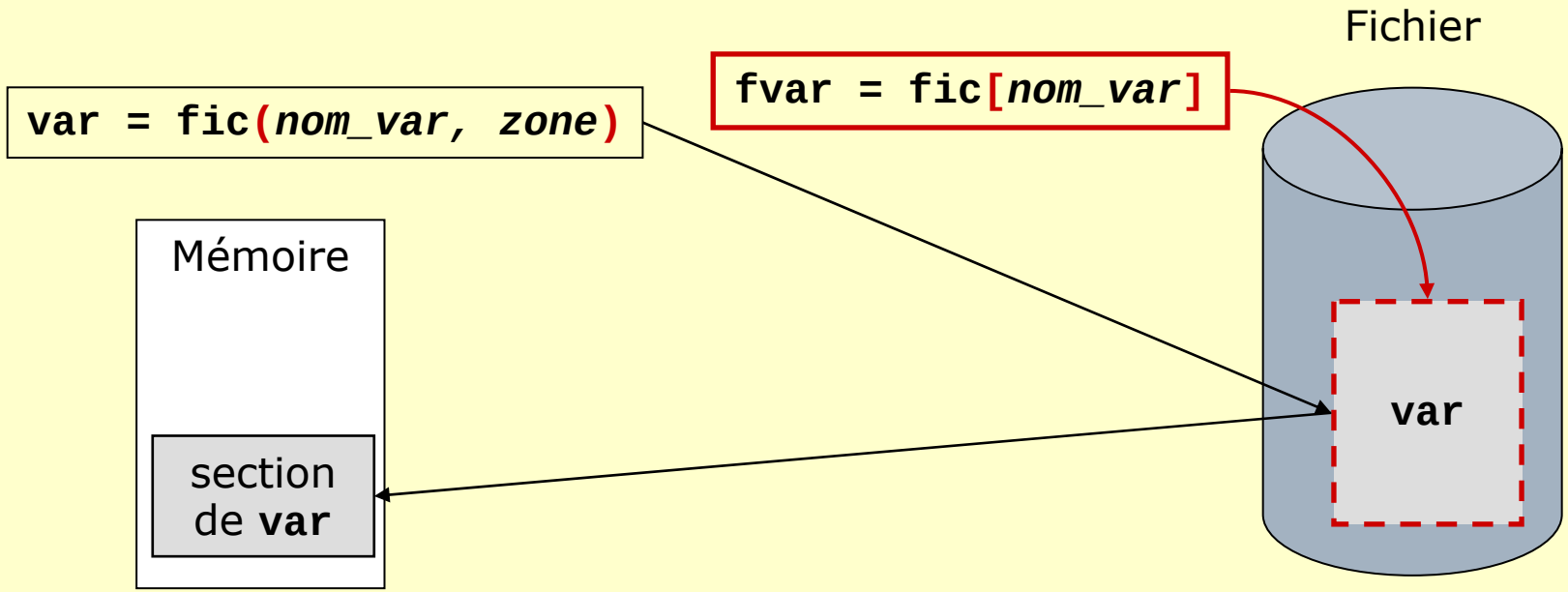
Les *file variables* (1/2)

Une *file variable* est un *pointeur* sur une variables *dans* un fichier

`fvar = fic[`*`nom_var`*`]`

- La variable n'occupe pas de place en mémoire
  (tant que l'on ne fait pas d'opération dessus)
- On ne peut pas spécifier de zone (on pointe sur *toute* la variable dans le fichier)
- Les valeurs de la variables sont chargées en mémoire lors d'une opération de slicing/indexing
  → Chargement uniquement des valeurs demandées

- Note : dans **`cdms.pdf`**, les variables qui ne sont pas des *file variables* sont appelées des *transient variables*… ☺



Les *file variables* (2/2)

- Modification d'une variable *dans* un fichier existant :
  - utilisation principale des *file variables*!
  - il faut ouvrir le fichier en mode '**r+**' (lecture/écriture)
  - l'écriture se fait automatiquement quand on assigne une valeur à la *file variable* :

    `fvar.shape` → **`(12, 50, 64)`**

    `fvar[:, 10:15, 31:40] = 0.` → mise à zéro d'une zone *dans* le fichier, pour tous les pas de temps
  - astuce : on peut utiliser **`mapIntervalExt`** pour déterminer les indices de la zone à modifier!

- Erreurs à éviter :
  - il vaut mieux travailler sur une copie du fichier à modifier, pour mettre au point le script! ☺

    → **`shutil.copy(fic, destination)`**
  - ne pas fermer le fichier tant que l'on a besoin d'une variable en cours d'utilisation (ou d'un axe de cette variable)
  - ne pas oublier de fermer le fichier à la fin!
  - ne pas se tromper dans l'ecriture!
    - **`fvar[…] = nouvelles_valeurs`** → OK ☺
      - **`nouvelles_valeurs`** doit avoir le même *typecode* que fvar!
      - **`nouvelles_valeurs`** doit avoir une *shape* compatible avec la zone sélectionnée de fvar!
    - **`fvar = nouvelles_valeurs`** → KO ☹

      → cette opération détruit le pointeur et crée une nouvelle variable **`fvar`**…

Récupération des axes d'une variable

- Noms des axes et liste des *objets axes* (dans le même ordre que pour la variable) :
  ```
  var.getAxisIds()
  var.getAxisList()
  ```

- Récupération d'un axe spécifique :
  - **`latax = var.get`*`Latitude`*`()`**
    → même chose avec **`Time`**, **`Level`** et **`Longitude`**
  - **`var.getAxis(`*`numéro_de_l_axe`*`)`**
    → **`var.getAxis(var.getAxisIndex('pres_nivs'))`**

- Informations sur un axe :
  - **`print latax`**
    ```
    latax
       id: latitude
       Designated a latitude axis.
       units:  degrees_north
       Length: 46
       First:  -90.0
       Last:   90.0
       Other axis attributes:
          axis: Y
    ```
  - **`latax.id`**, **`latax.units`**, **`latax.axis`**, **`len(latax)`**, **`latax.typecode()`**, **`latax[:]`** (valeurs des coordonnées)
  - Détermination du type d'un axe avec les méthodes **`axe.isTime()`**, **`isLevel()`**, **`isLatitude()`**, **`isLongitude()`**

- Bornes d'un axe : **`latax.`*`getBounds()`***
  - récupération des bornes, dans un tableau **`(len(latax), 2)`**
  - les bornes sont définies dans une variable dont le nom est indiqué dans l'attribut **`latax.bounds`** (norme CF)
    `ncdump –h fichier.nc` → **`float latitude(latitude) ;`**
    `latitude:bounds = "bounds_latitude" ;`
    `latitude:units = "degrees_north" ;`
  - Si les bornes des axes ne sont pas définies dans le fichier NetCDF, les bornes seront automatiquement générées, en fonction du mode de : **`cdms.setAutoBounds(mode)`**
    - **`2`** ou **`'grid'`** (par défaut) : génération des bornes des axes reconnus comme des latitudes et longitudes
    - **`1`** ou **`'on'`** : génération des bornes pour tous les axes (y compris le temps)
    - **`0`** ou **`'off'`** : pas de génération des bornes
  - Note : il faut avoir des bornes correctes pour calculer des bonnes moyennes pondérées (avec **`cdutil.averager()`**)
    - moyennes spatiales : bornes des latitudes et longitudes
    - moyennes temportelles : bornes de l'axe des temps

- Grille (latitude, longitude) associée à la variable : **`grille = var.getGrid()`**

Création des axes (1/2)

- Copie d'un axe : **`copie_axe = axe.clone()`**
- Création d'un axe :
  `axe = cdms.createAxis(`*`valeurs`*` [, `*`bornes`*`])`
  - **`valeurs`** est un vecteur de valeurs strictement croissantes
  - **`bornes`** est un tableau optionnel (mais recommandé!) de forme **`(len(valeurs), 2)`**
  - Spécification des bons attributs :
    - Temps :
      ```
      axe.designateTime(calendar=cdms.calendrier)
      axe.id = 'time'
      axe.units = 'days since 1900-1-1'
      ```
      Types de calendriers :
      - **`MixedCalendar`** (par défaut) : Julian avant 1582-10-15, Gregorian ensuite…
      - **`NoLeapCalendar`** : années de 365 jours
      - **`Calendar360`** : années de 360 jours
      - **`JulianCalendar`** et **`GregorianCalendar`**

      Il faut que le type de calendrier soit bien défini, pour que les fonctions de conversion de temps donnent des bons résultats!

      Note : **`ferret`** semble avoir des problèmes avec le calendrier **`MixedCalendar`** (qui génère un attribut '**`calendar = "proleptic_gregorian" ;`**' dans le fichier netCDF)… Dans ce cas, il faut changer la valeur de l'attribut **`calendar`** de l'axe temps après l'appel à **`designateTime`**() et avant l'écriture dans un fichier!
      → **`axe.calendar = 'Gregorian'`**
    - Niveau :
      ```
      axe.designateLevel()
      axe.id = 'plev'
      axe.units = Pa
      ```
    - Latitude :
      ```
      axe.designateLatitude()
      axe.id = 'lat'
      axe.units = 'degrees_north'
      ```
    - Longitude :
      ```
      axe.designateLongitude()
      axe.id = 'lon'
      axe.units = 'degrees_east'
      ```
- Modification des valeurs des coordonnées d'un axe existant :
  `axe.assignValue(`*`valeurs`*`)` (mise à jour des bornes?)
- Remplacement d'un axe d'une variable :
  `var.setAxis(`*`index_de_l_axe, nouvel_objet_axe`*`)`
  ou **`var.setAxisList([`**axe_0**`, `**axe_1**`, …, `**axe_{N-1}**`])`** ← remplacement de tous les axes

Création des axes (2/2)

- Création d'axes réguliers :
  - **cdms.create*Uniform*LatitudeAxis(lat_deb, nb_lat, lat_inc)**
  - **cdms.create*Uniform*LongitudeAxis(lon_deb, nb_lon, lon_inc)**

- Création d'axes des latitudes spéciaux :
  - **latax = cdms.create*EqualArea*Axis(nb_lat)**
    → axe régulier en sinus de la latitude (type LMD4/LMD5…)
  - **latax = cdms.create*Gaussian*Axis(nb_lat)**
    → grilles de type *Txx* et *Rxx*

- Création d'une grille latitude/longitude :
  - **cdms.create*Uniform*Grid(lat_deb, nb_lat, lat_inc, lon_deb, nb_lon, lon_inc)**
    → **grille_lmdz = cdms.createUniformGrid(90, 46, 4, -180, 72, 5)**
    lat_lmdz = grille_lmdz.getLatitude()
    lon_lmdz = grille_lmdz.getLongitude()
  - **cdms.create*Gaussian*Grid(nb_lat, lon_deb)**
  - à partir d'axes existants : **cdms.create*Rect*Grid(latax, lonax)**

- Récupération des *poids* des cases (en latitude et longitude)
  poids_dims = grille.getWeights() → tuple de 2 éléments
  → **poids_dims[0] : poids selon les latitudes**
  → **poids_dims[1] : poids selon les longitudes**
  Poids de la grille :
  → **poids = N.matrixmultiply(poids_dims[0][:, N.NewAxis], poids_dims[1][N.NewAxis, :])**

Gestion de l'axe des temps (2/3)

- Manipulation des dates :
  Note : le calendrier utilisé par défaut est le ***MixedCalendar***…
  - ajout, soustraction d'une durée :
    - **ta1.add(1.5, cdtime.Day)** → **2007-3-1 2:0:0.0**
    - **tr1.sub(3600, cdtime.Seconds)** → **1.00 hours since 2007-2-27 14:00:00**
  - comparaison :
    - **ta1.cmp(tr1)** → **-1** (**ta1** est antérieur à **tr1**)
    - **tr1.cmp(ta1)** → **1**
  - conversion temps absolu⇔relatif :
    - **tr1.tocomp()** → **2007-2-27 16:0:0.0**
    - **tr_test_fevrier = cdtime.reltime(2, 'days since 2004-2-28')**
      - **tr_test_fevrier.tocomp(cdtime.MixedCalendar)**
        2004-3-1 0:0:0.0
      - **tr_test_fevrier.tocomp(cdtime.NoLeapCalendar)**
        2004-3-2 0:0:0.0
      - **tr_test_fevrier.tocomp(cdtime.Calendar360)**
        2004-2-30 0:0:0.0
    - **ta.torel('hours since 2004-1-1')**
      - **27686.00 hours since 2004-1-1**
    - **ta.torel('hours since 2004-1-1', cdtime.NoLeapCalendar)**
      - **27662.00 hours since 2004-1-1**

- Manipulation d'un axe des temps :
  - rappel : affichage des valeurs brutes avec **timax[:]**
  - affichage des valeurs sous forme (de liste) de temps absolu ou relatif :
    timax.asComponentTime() et **timax.asRelativeTime()**

Gestion de l'axe des temps (3/3)

- Conversion d'unités relatives
  - exemple : **timax → id: time**

```
Designated a time axis.
units:  days since 1987-1-2 0:0
Length: 5
First:  0.0
Last:   4.0
```

  - **timax[:]** → **[0., 1., 2., 3., 4.,]**
  - **timax.asComponentTime()** → **[1987-1-2 0:0:0.0, 1987-1-3 0:0:0.0, 1987-1-4 0:0:0.0, 1987-1-5 0:0:0.0, 1987-1-6 0:0:0.0]**
  - **timax.asComponentTime()[0].*torel*('hours since 1987-1-1 12:0:0')**
    → **12.00 hours since 1987-1-1 12:0:0**
  - **timax.asComponentTime()[-1].torel('hours since 1987-1-1 12:0:0').*value***
    → **108.0**

- Création de bornes correctes (ou correction des bornes existantes) :
  - par défaut, si les bornes ne sont pas définies, et si on a un réglage **cdms.setAutoBounds(1)**, les bornes seront définies au milieu de deux pas de temps consécutifs! → on ne veut en général pas ça!
  - correction des bornes avec :
    cdutil.setTimeBoundsDaily(*variable_ou_axe*) (et aussi **Monthly** et **Yearly**)
    Les bornes sont positionnées au début des intervalles considérés, et au début de l'intervalle suivant (exemple : du début d'une journée au début de la journée suivante)
    cdutil.setTimeBoundsDaily(timax) (avec le même axe **timax** que ci-dessus)
    timax.getBounds() → **[[0., 1.,], [1., 2.,], …, [4., 5.,]]**
  - dans le cas où on a plusieurs données par jour, il faut utiliser le paramètre optionnel frequency…
    cdutil.setTimeBoundsDaily(*variable_ou_axe, frequency=freq*) :
    - données toutes les 12 heures : **frequency=2**
    - données toutes les 6 heures : **frequency=4**
    - données toutes les heures : **frequency=24**

Exemple : fonction de conversion de temps relatif

```
def changeTimeAxisUnits(old_axis, new_units):
    import Numeric, cdms, cdtime

    # Get the calendar, if defined
    # (default to cdtime.DefaultCalendar otherwise, which
    # is supposed to be cdtime.MixedCalendar...)
    try:
        cal = old_axis.calendar
    except:
        cal = cdtime.DefaultCalendar

    # Determine the new values of the time coordinates
    old_comptime_list = old_axis.asComponentTime(cal)
    new_vals_list = [t.torel(new_units, cal).value
                     for t in old_comptime_list]
    new_vals = Numeric.asarray(new_vals_list, old_axis.typecode())

    # Determine the new values of the bounds,
    # if the bounds are defined, then create the axis
    old_bounds = old_axis.getBounds()
    if old_bounds:
        old_units = old_axis.units
        new_bounds_list = [cdtime.reltime(b, old_units).torel(new_units,
                                                              cal).value
                           for b in old_bounds.flat.tolist()]
        new_bounds = Numeric.asarray(new_bounds_list, old_bounds.typecode())
        # Bounds array must have a shape of (len(old_axis), 2)
        new_bounds.shape = (-1, 2)
        new_axis = cdms.createAxis(new_vals, new_bounds)
    else:
        new_axis = cdms.createAxis(new_vals)

    # Define the attributes for the new axis
    new_axis.designateTime(cal)
    new_axis.id = old_axis.id
    new_axis.units = new_units

    return new_axis
```

Création d'une variable

- Par copie ou *slicing/indexing* d'une variable existante

  `v_nouvelle = v[0, …]`

- Par opération mathématique sur une ou plusieurs variables existantes

  `v_nouvelle = v - 273.15`

  - les variables doivent être définies sur les mêmes axes (et la nouvelle variable aura aussi les mêmes axes)
  - le *typecode* de la nouvelle variable dépend des opérandes et du mode de gestion de la précision
  - la nouvelle variable a un nom par défaut (***`variable_nn`***), et pas d'attributs…

    Mise à jour du nom et des attributs :

    ```
    v_nouvelle.id = 'nouveau_nom'
    v_nouvelle.units = 'nouvelles unités'
    ```

- En combinant divers éléments :

  ```
  var = cdms.createVariable(tableau_ou_var, mask=m,
                            axes=(axe0, …, axeN-1), id=nom_var)
  var.units = 'unités'
  var.autre_attribut = valeur …
  ```

  Notes :

  - le nom de la variable (**`id`**), le masque, les axes et les attributs seront les mêmes que ceux de **`tableau_ou_var`**, si **`tableau_ou_var`** est une variable **`MV`**, et que l'on ne fournit pas de paramètres de remplacement.
  - Plus de détails dans la table 2.34 (p 96) de la section 2.11 … ☺

Attentions! Erreurs à éviter (1/…)

- Le module **cdms** n'est disponible que si CDAT a été installé!

  Si on a l'erreur suivante, c'est que l'on n'a pas accès à la bonne version de python/CDAT

  ```
  ImportError: No module named cdms
  ```

- Mise à zéro (ou autre) d'une variable :
  - **var[…] = 0.** → toutes les valeurs de **var** sont mises à zéro ☺
  - **var = 0.** → le tableau **var** est détruit et remplacé par un réel valant zéro! ☹
- BUG **cdms**! Création d'un axe de taille 1 (1 pas de temps, par exemple) :

  ```
  axval = Numeric.array((0.,), Numeric.Float32)
  ```

  - Ne pas utiliser :

    ```
    axe = cdms.createAxis(axval)
    ou  axe = cdms.createAxis(0.)
    ```

  - Utiliser :

    ```
    axe = cdms.createAxis((axval,))
    ou  axe = cdms.createAxis((0.,))
    ```

Trucs et astuces de **cdms, …** (1/…)

- On peut faire directement un :

  `import MV`

  pour pouvoir taper **`MV.fonction()`** au lieu de **`cdms.MV.fonction()`**
- Transformation d'une variable **`MV`** en un tableau de **`Numeric`** : **`var.filled()`**
- Extrema d'une variable :

  `MV.minimum.reduce(var.flat)`,
  `MV.maximum.reduce(var.flat)`

  ou **`genutil.minmax(var)`**
- Changement d'unités :

  Si on n'a pas de mémoire ou que l'on est vraiment fatigué! ☺

```
from genutil import udunits
    lapse_rate = udunits(1., 'degC km-1')
        lapse_rate .value → 1.
        lapse_rate.units → 'degC km-1'
    lapse_rate.to('K m-1') → udunits(0.27415,"K m-1")
    facteur, offset = lapse_rate.how('K m-1')
        → (0.001, 0.27315)
    lapse_rate - udunits(0.27415,"K m-1")
        udunits(0.0,"degC km-1")
```

Liste des unités connues : **`lapse_rate.available_units()`**

Trucs et astuces de **cdms, …** (2/…)

- Rajouter des pas de temps à une variable dans un fichier existant!
  - le fichier doit être ouvert en mode '**a**' (ou '**r+**' : dans ce mode, génération d'une *exception* à l'ouverture, si le fichier n'existe pas déjà)
  - le fichier doit déjà contenir une variable **var1(time1$_{fic1}$, lon1, lat1)** dont la première dimension (indice **0**…) est le temps, et est considérée comme ***UNLIMITED*** par netCDF. On suppose que **time1$_{fic1}$** est défini avec l'unité **unit1** et le calendrier **calendar1**

    Exemple : **fic1** contient **var1**, défini sur un axe de N_{fic1} pas de temps.

    Vf1$_{min}$ <= **time1$_{fic1}$** <= **Vf1$_{max}$**
  - On suppose que les pas de temps à rajouter à **var1** dans le fichier sont contenus dans la variable **var1** en mémoire : **var1(time1, lon1, lat1)**

    où **time1** a 2 pas de temps [**V1 unit1**, **V2 unit1**] (et **V1** < **V2**)

    et **time1** est défini sur le même calendrier **calendar1** que celui de **fic1**
  - si on rajoute des pas de temps à **var1**, et que le fichier contient d'autres variables définies sur l'axe des temps (**var$_i$(time1, …)**), ces variables sont aussi étendues, en remplissant l'extension avec la valeur *missing value*
  - **fic1.write(var1)** :
    - *Remplacement* des *valeurs* de **var1** pour les pas de temps **V1** et **V2** si **V1** et **V2** sont 2 pas de temps *consécutifs* et *existants* de **time1$_{fic1}$**
    - *Remplacement* de **V1** et *ajout* de **V2** si **V1**==**Vf1$_{max}$** (rappel : **V1** < **V2**)
    - *Ajout* de **V1** et **V2** si **V1**>**Vf1$_{max}$**
  - **fic1.write(var1, index=idx)** :
    - Si **0** <= **idx** <= **N$_{fic1}$-1** : *remplacement* (et éventuellement *ajout*) des pas de temps d'indice **idx** et **idx+1** par **V1** et **V2** et les valeurs de **var1**
    - Si **idx** > **N$_{fic1}$-1** : *ajout* des 2 pas de temps de **var1**
    - Attention! Avec ce mode d'écriture, **cdms** ne vérifie pas la cohérence des valeurs de **V1** et **V2** avec les valeurs de **time1$_{fic1}$** …

      Exemple : si **time1$_{fic1}$=[15, 45, 75, 105, 135] days since 1-1-1** (**calendar360**)

      et **V1=0**, **V2=10**, **idx=2**

      on obtient un axe mis à jour : **[15, 45, 0, 10, 135]** !

Trucs et astuces de **cdms**, … (3/…)

- Détermination des indices d'une zone en fonction de ses coordonnées, avec **MapIntervalExt**… → exemple pour les latitudes
  - définition de la zone :
    zonelat = (lat_deb, lat_fin, 'cc')
    Note : voir la doc de **MapIntervalExt** dans **cdms.pdf** (table 2.10 de la section 2.5) pour plus de détails sur la façon de spécifier une zone (**'cc'** et ses variantes…)
  - récupération des *indices* et du *pas* :
    - **latax = var.getLatitude()**
      ind_deb, ind_fin, pas = latax.MapIntervalExt(zonelat)
      - Si **pas=1** : **lat_deb** et **lat_fin** sont orientés dans le même sens que les latitudes dans le fichier
      - Si **pas=-1** : sens inverse
      - Exemple : **lat_deb<lat_fin** et les latitudes vont de +90° à -90° → **pas=-1**
    - Les indices récupérés sont tels que :
      var = fic('nom', latitude=zonelat) ⇔ **var = fic('nom', latitude=slice(ind_deb, ind_fin, pas))**
    - Utilité : la connaissance des indices permet de faire des opérations de *slicing* pour étudier/modifier une zone d'intérêt!
      - … en particulier pour travailler avec des *files variables*!
  - exemple :
    - **latax[:]** → **[-90.,-86., … , 10., 14., 18., 22., 26., 30., … , 90.]**
    - **latax.mapIntervalExt((11, 52, 'cc'))** → **(26, 36, 1)**
    - **latax[26:36:1]** → **[14., 18., 22., 26., 30., 34., 38., 42., 46., 50.]**
    - **latax[26]** → **14.0** **latax[35]** → **50.0**
  - longitudes : si on n'obtient pas **0**<= **ind_deb** < **ind_fin** <= **len(lonax)**, la zone demandée a necessité la prise en compte de la périodicité de 360°, et il faut envisager un traitement particulier!
    lonax[:] → [0., 5., 10., …, 345., 350., 355.] (72 longitudes)
    - **lonax.mapIntervalExt((-10, 10, 'cc'))** → **(-2, 3, 1)**
    - **lonax.mapIntervalExt((340, 380, 'cc'))** → **(68, 77, 1)**

cdutil et **genutil**

Ces modules sont décrits dans
cdat_utilities.pdf

→ lire la doc avant de faire des moyennes et des statistiques! ☺

- cdutil.averager …

CDAT et le langage python

Gestion du graphisme avec…

… le module **vcs**

Graphisme avec vcs

Autres modules (*contrib*)

- Ce sont des modules supplémentaires distribués avec CDAT
- La liste à jour est disponible sur

  http://www-pcmdi.llnl.gov/software-portal/cdat/contrib

Liens utiles

- Site officiel de CDAT

  http://www-pcmdi.llnl.gov/software-portal/cdat/
  - Liste des contribs (modules optionnels) disponibles dans CDAT

    http://www-pcmdi.llnl.gov/software-portal/cdat/contrib
  - Récupération des sources sur :
    - Sourceforge : http://sourceforge.net/projects/cdat/
    - Le serveur subversion du PCMDI :

      http://www-pcmdi.llnl.gov/software-portal/cdat/support/svn
- Python/CDAT for Earth Scientists: Tips and Examples

  http://www.johnny-lin.com/cdat_tips/
- Table de correspondance entre python, IDL et MATLAB/Octave

  http://37mm.no/mpy/idl-python-xref.pdf
- Plein de scripts de JYP au LSCE

  http://boulimix/~jypeter/CDAT/Progs/Devel/

  http://www-lscei.cea.fr/Dokuwiki/doku.php?id=cdat:exemples

Documentation recommandée

- Sur le site officiel

  http://www-pcmdi.llnl.gov/software-portal/cdat/manuals
  http://www-pcmdi.llnl.gov/software-portal/cdat/quick_reference

- Au LSCE
  - Doc :

    http://boulimix/~jypeter/CDAT/Doc/CDAT
    ou **/home/users/jypeter/CDAT/Doc/CDAT**
    et **/home/users/jypeter/CDAT/Doc/utile.txt**
    - **numpy.pdf** : Numeric
    - **cdms.pdf**, **cdms_quick_start.pdf** : cdms
    - **vcs_quick_ref.pdf** : vcs
  - Exemples (*tutorials*) :

    /home/share/unix_files/cdat/tutorials
  - Exemples de ***jypeter***

    /home/users/jypeter/CDAT/Progs/Devel
    http://www-lscei.cea.fr/Dokuwiki/doku.php?id=cdat:exemples

Des questions?

- …

A faire…

- détails sur cdutil, genutil
- Regridding
- Grilles irrégulières
- Agréger plusieurs fichiers netCDF en un seul jeu de données (*dataset*)
- vcs…

- …