Cours Apl 07 - Fonctions de comparaisons, Booléens

Nous entrons ici dans le monde des Booléens, ou si vous préférez, des oui et des non.
Le oui est représenté par le nombre 1 et le non par 0.

- **Les fonctions** `< > ≤ ≥` ne peuvent s'appliquer qu'à des nombres.
Elles font des comparaisons terme à terme entre 2 objets a priori de même taille.
Exemple :

```
      5 7 3 < 3 5 7
0 0 1
      9 > 3 8 20 12 9
1 1 0 0 0
```

- **Les fonctions = et** `≠` acceptent indifféremment des arguments numériques ou alphabétiques.
Exemple :

```
      'aabc' = 'papa'
0 1 0 0
```

Seul le 2ème "a" de chaque vecteur correspondait.

```
      'aabc' ≠ 'papa'
1 0 1 1
```

- **La fonction** `∈` cherche si les éléments de G existent dans D, quel que soit leur emplacement et leur nombre. De même `∈`, ne travaillant pas terme à terme, n'a aucune exigence concernant les structures de G et D. Par contre le résultat aura la structure de G.

Exemple :

```
      'papa' ∈ 'chapeau'
1 1 1 1
```

En effet le "p" et le "a" de papa se retrouvent bien dans chapeau.

```
      (2 3 ⍴ ⍳ 6) ∈ 2 4 6
0 1 0
1 0 1
```

- Une autre fonction intéressante est **l'appartient souligné :** `⍷`
Il cherche les emplacements du vecteur G dans l'objet D de structure quelconque et en marque le début d'un 1. La structure du résultat est celle de D.
Exemple :

```
      'pa' ⍷ 'papa est parti patauger'
1 0 1 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0
```

A chaque fois que le vecteur 'pa' est détecté, on place un 1 à l'emplacement du 'p'.

- La dernière fonction de comparaison est la fonction **"identique" :** ≡
Contrairement à toutes les précédentes, elle ne travaille pas sur le contenu des arguments mais sur eux dans leur globalité et rend donc un seul booléen pour dire si les objets sont identiques ou pas.
Exemples :

```
      'papa'≡ 1 2 2
0
      'abc' ≡ 'acb'
0
      1 ≡ ,1
0
```

Dans ce dernier exemple on aurait pu s'attendre à ce qu'Apl réponde oui, mais il compare aussi la structure des objets. Or 1 est un scalaire alors que ,1 est un vecteur. Ils ne sont donc pas identiques.

- **La fonction not : ~** ,utilisée en mode monadique, retourne le contraire de ce qu'on lui soumet.
Exemple :

```
      ~ 0 0 1 1 0
1 1 0 0 1
```

- Combiner les conditions

Il est fréquent de devoir combiner plusieurs tests.
Par exemple pour savoir si un ou plusieurs nombres sont compris dans un intervalle, on doit tester si ils sont supérieurs à la borne inférieure et inférieurs à la borne supérieure.

- Il existe principalement 2 fonctions Apl pour cela : **le "et" :** ∧ **et le "ou" :** ∨
Exemple : `Vtest ← 8 12 4 2 9 15`
Je veux connaître les nombres compris entre 8 et 14, bornes incluses.

```
      (Vtest ≥ 8) ∧ Vtest ≤ 14
1 1 0 0 1 0
```

Décomposons :
En remplaçant chaque test par son résultat intermédiaire, on obtient :
(1 1 0 0 1 1) ∧ 1 1 1 1 1 0

- Applications des booléens

- Exécution conditionnelle : L'utilisation la plus commune dans tous les langages est celle qui consiste à dire "Si oui alors je fais ci, sinon je fais ça".
Cette possibilité est utilisée dans l'écriture des fonctions utilisateurs (programmes), que nous étudierons plus loin.

- On peut également **utiliser les booléens en tant que nombres** pour modifier la valeur d'un objet.
Exemple : on veut mettre à zéro tous les nombres inférieurs à 5

```
      Ex ← 3 8 5 12 2 ¯3
      (~Ex < 5) × Ex
0 8 5 12 0 0
```

Décomposons :

Ex < 5 Inférieur à 5 ?
1 0 0 0 1 1

~ 1 0 0 0 1 1 Le contraire
0 1 1 1 0 0

0 1 1 1 0 0 x Ex
0 8 5 12 0 0

- L'utilisation la plus originale mais pas la moins efficace est la "**compression**". La fonction utilisée dans ce cas est le **/**
Exemple :

```
      1 1 0 0 1 / 'abcde'
'abe'
```

On peut également l'appliquer à une matrice : 1 1 0 0 / 3 4 ⍴⍳ 4
1 2
1 2
1 2

On peut aussi compresser tout un objet avec un scalaire :
0 / 1 2 3 4 rend un vecteur de dimension 0.

La compression est également utilisée pour faire des affectations partielles.
Exemple : soit `phrase←'binjiur tintin'`
On veut remplacer les i par des o.
Pour marquer les i, on écrit : `phrase = 'i'`
Pour les afficher : `(phrase = 'i')/phrase`
Pour les remplacer par des o :

```
((phrase = 'i')/phrase)←'o'

      phrase
'bonjour tonton'
```

Travaux pratiques :

0. Chargez votre Ws de travaux pratiques :
)load c:\Mes documents\pratique-apl

1. Affichez un booléen indiquant les positions de Coeff inférieures à 100.

2. En une seule expression, transformez 1 en 1000 et 0 en 1 (vous aurez besoin de + et x).
Essayez avec le vecteur 1 0

Affichez Coeff en multipliant par 1000 ses éléments inférieurs à 100 et en laissant les autres dans leur état initial.

3. Affichez le vecteur des nombres impairs de Mnum3.

4. Créez le vecteur suivant : Vtruc←10?100
En utilisant un booléen, modifiez les valeurs inférieures à 50 en leur ajoutant 100.

5. Affichez une matrice booléenne indiquant quels nombres de Mnum2 sont égaux à 10 ou 20.

6. Affichez le vecteur des nombres de Mnum2 non égaux à 30

7. Affichez les nombres de Coeff inférieurs à 100 et supérieurs à 1.

8. Affichez les nombres de Coeff pairs et inférieurs à 100.
- Affichez les nombres de Coeff pairs ou inférieurs à 100.

9. Affichez les éléments de Vtest qui, multipliés par 10, seront supérieurs à 100.

10. Parmi les éléments de Vtest, affichez les multiples de 4

11. Sauvez votre travail :
)save

Solutions :

1. Affichez un booléen indiquant les positions de Coeff inférieures à 100.

```
Coeff<100
```

2. En une seule expression, transformez 1 en 1000 et 0 en 1 (vous aurez besoin de + et x). Essayez avec le vecteur 1 0

```
1+999×1 0
```

Affichez Coeff en multipliant par 1000 ses éléments inférieurs à 100 et en laissant les autres dans leur état initial.

```
Coeff × 1+999 × Coeff < 100
```

Si on décompose, `Coeff < 100` nous donne un booléen qu'on transforme en 1000 et en 1 via `1+999 ×`
Il suffit donc de multiplier Coeff par ce vecteur pour obtenir le résultat final.

3. Affichez le vecteur des nombres impairs de Mnum3.

```
(,~2|Mnum3)/,Mnum3
```

`2|Mnum3` nous donne le reste de la division par 2 de chaque élément de Mnum3.
Ce reste est égal à 0 dans le cas d'un nombre pair et à 1 pour un nombre impair.
~ inverse le sens du faux booléen ainsi obtenu et donne donc oui pour pair et non pour impair.
Le booléen obtenu est le suivant :
0 1 0 1
0 1 0 1
0 1 0 1

On ne peut pas demander à Apl de restituer une matrice filtrée par un masque booléen car l'objet restitué serait une sorte de gruyère qu'Apl est incapable de gérer. On n'est plus sûr de toujours avoir un objet à 2 dimensions.
En revanche, un vecteur filtré par un autre vecteur rendra toujours un vecteur. C'est pourquoi on linéarise à la fois le booléen et Mnum3 : `(,booléen)/,Matrice`

4. Créez le vecteur suivant : Vtruc←10?100
En utilisant un booléen, modifiez les valeurs inférieures à 50 en leur ajoutant 100.

```
Vtruc←10?100
((Vtruc<50)/Vtruc)←100+((Vtruc<50)/Vtruc)
```

ou en affection modifiée (chapitre 4)

```
((Vtruc<50)/Vtruc)+←100
```

5. Affichez une matrice booléenne indiquant quels nombres de Mnum2 sont égaux à 10 ou 20.

```
Mnum2 ∈ 10 20
```

6. Affichez le vecteur des nombres de Mnum2 non égaux à 30

```
Mnum2 ≠ 30
```

7. Affichez les nombres de Coeff inférieurs à 100 et supérieurs à 1.

```
((Coeff < 100) ∧ Coeff > 1)/Coeff
```

8. Affichez les nombres de Coeff pairs et inférieurs à 100.

```
((~2|Coeff)∧Coeff<100)/Coeff
```

- Affichez les nombres de Coeff pairs ou inférieurs à 100.

```
((~2|Coeff)∨Coeff<100)/Coeff
```

9. Affichez les éléments de Vtest qui, multipliés par 10, seront supérieurs à 100.

```
(100<Vtest×10)/Vtest
```

10. Parmi les éléments de Vtest, affichez les multiples de 4

```
(0=4|Vtest)/Vtest
```