# CH 4 : Types de données en **Scilab**

## I. Les types de données en informatique

### I.1. Définition

En informatique, chaque donnée est associée à un **type de données**. Les types de données sont à comprendre comme une classification des données. Formellement, un type de données est défini par :

- un ensemble des valeurs : toutes les valeurs qui ont le même type.
- un ensemble d'opérateurs : les opérateurs qui permettent de manipuler et de combiner ces valeurs.

La définition peut sembler un peu vague. Ainsi, afin d'appréhender cette notion, nous allons détailler les principaux types de données qui sont utilisés dans **Scilab**. On pourra noter que nous avons déjà rencontré la plupart de ces types de données dans les chapitres précédents, sans toutefois les nommer précisément.

### I.2. La fonction `typeof`

La fonction `typeof` permet d'associer chaque objet à son type. Par exemple, on peut se demander quel est le type associé au réel `1`.

```
--> typeof(1)
    ans  =
        constant
```

Commençons donc par détailler le type `constant`.

## II. Le type `constant`

Nous avons mentionné dans le « CH 1 : Découverte de **Scilab** » que les nombres représentables en **Scilab** sont régis par la norme IEEE-754 (voir http://fr.wikipedia.org/wiki/IEEE_754). Tous ces réels sont regroupés dans un même type de données, nommé `constant`.

Rappelons les principaux opérateurs permettant de manipuler et de combiner les objets de type `constant`, déjà présentés dans le premier chapitre.

| | |
|---|---|
| `+` | opérateur d'addition |
| `-` | opérateur de soustraction |
| `*` | opérateur de multiplication |
| `/` | opérateur de division |
| `^` | opérateur puissance : permet de calculer $x^y$ |
| `**` | opérateur puissance : `x**y` est l'équivalent de `x^y` |

Il faut comprendre que tous ces opérateurs ont la même spécification, à savoir `op : constant` $\times$ `constant` $\rightarrow$ `constant`. Autrement dit, chacune de ces fonctions prend en entrée deux objets de type `constant` et les combinent pour obtenir un nouvel objet de type `constant`. Ceci n'est pas très surprenant car correspond à la manipulation des réels en mathématiques : si l'on somme, soustrait, multiplie, ou divise deux réels, on obtient un réel.

Il est à noter que la fonction `typeof` peut être utilisée pour :

- estimer le type de données d'une valeur.

```
--> typeof(3.4)
    ans  =
        constant
```

- estimer le type de données d'une expression.

```
--> typeof(5+2*3/4)
    ans  =
        constant
```

- estimer le type de données d'une variable.

```
--> typeof(%pi)
    ans  =
        constant
--> a = 5; typeof(a)
    ans  =
        constant
```

Le type d'une variable est par définition le type de la valeur qu'elle contient.

## III. Le type `boolean`

Le type `boolean` (*booléen* en français) ne regroupe que deux éléments : `%f` et `%t`. La variable `%t` représente la valeur de vérité *vrai* et `%f` représente la valeur de vérité *faux*.

Rappelons les principaux opérateurs permettant de manipuler et de combiner les objets de type `boolean`.

| | |
|---|---|
| `\|` | opérateur `OU` de **Scilab** |
| `&` | opérateur `ET` de **Scilab** |
| $\sim$ | opérateur `NON` de **Scilab** |

Les deux premiers opérateurs | et & possèdent la même spécification, à savoir `op : boolean` $\times$ `boolean` $\rightarrow$ `boolean`. L'opérateur $\sim$ a pour spécification $\sim$ `: boolean` $\rightarrow$ `boolean`. Ceci signifie que la disjonction (l'opérateur |) et la conjonction (l'opérateur &) de deux `boolean` est un `boolean`, et que la négation (l'opérateur $\sim$) d'un `boolean` est un `boolean`.

La fonction `typeof` permet d'illustrer le fonctionnement du type `boolean`.

```
--> typeof(%t)
    ans  =
        boolean
```

On peut notamment tester les spécifications des opérateurs `|`, `&` et $\sim$.

```
--> typeof(%t & %f), typeof(%t | %f), typeof(~%f)
    ans  =
        boolean
    ans  =
        boolean
    ans  =
        boolean
```

Ces résultats ne sont pas surprenants puisque `%t & %f` vaut `%f`, que `%t | %f` vaut `%t` et que $\sim$`%f` vaut `%t`.

Pour comprendre l'intérêt du type de données `boolean`, il faut introduire les opérateurs de comparaison.

| | |
|---|---|
| `>` | permet de **tester** la supériorité stricte |
| `<` | permet de **tester** l'infériorité stricte |
| `>=` | permet de **tester** la supériorité large |
| `<=` | permet de **tester** l'infériorité large |
| `==` | permet de **tester** l'égalité |

Ces opérateurs ont pour spécification `op : constant` $\times$ `constant` $\rightarrow$ `boolean`. Ce ne sont pas des opérateurs combinant des `boolean` mais qui permettent de créer des `boolean` en combinant des objets de type `constant`. Ils seront utilisés pour créer des conditionnelles (*cf* CH 7).

```
--> typeof(3 <= %e)
    ans  =
        boolean
--> x = (12 == 3.5*(11-9)+5), typeof(x)
    x  =
        V
    ans  =
        boolean
```

### Conversion implicite de type

Nous avons vu précédemment que l'opérateur `+` a pour spécification : `constant` $\times$ `constant` $\rightarrow$ `constant`. Cela signifie notamment que cet opérateur ne peut être utilisé, a priori, qu'avec des objets de type `constant`. En réalité, il est possible d'utiliser la fonction `+` avec des objets de type `boolean`. Pour ce faire, une **conversion de types** est opérée. Le booléen `%t` est remplacé par `1` (et donc converti vers le type `constant`) et le booléen `%f` est remplacé par `0`. **Scilab** utilise ici la bijection naturelle qui existe entre l'ensemble {`%f`, `%t`} et l'ensemble $\{0, 1\}$. Ce mécanisme a lieu sans que l'on ait à faire appel à une fonction de conversion. On dira donc qu'il y a **conversion implicite** de type. Via cette conversion, toute fonction utilisant des objets de type `constant` peut être utilisée avec des objets de type `boolean`.

```
--> ((8+%t)**%f)*(26-%t)
    ans  =
        25.
```

## IV. Le type `string`

Le type `string` regroupe l'ensemble des chaînes de caractères *i.e.* **tous** les éléments qui s'écrivent entre guillemets (`" "` ou `' '`).

Il n'y a qu'un opérateur combinant les chaînes de caractère. Il s'agit de l'opérateur de concaténation qui prend en paramètre deux chaînes de caractère et en crée une nouvelle en les mettant bout à bout.

| | |
|---|---|
| `+` | opérateur de concaténation |

L'opérateur de concaténation est noté grâce au symbole d'addition `+` et a pour spécification `+ : string` $\times$ `string` $\rightarrow$ `string`. Ceci semble en contradiction avec la présentation précédente, décrivant l'opérateur `+` comme un opérateur permettant de combiner seulement des objets de type `constant`. Expliquons ce phénomène.

### Surcharge d'un opérateur

En **Scilab**, il est possible de **surcharger** la définition d'un opérateur *i.e.* de définir le comportement de cet opérateur pour des objets dont le type n'est pas celui défini dans sa spécification. En l'occurence, l'opérateur `+` a pour spécification `+ : constant` $\times$ `constant` $\rightarrow$ `constant` mais est surchargé pour pouvoir être utilisé avec des chaînes de caractères.

```
--> typeof("Bonjour à tous !")
    ans  =
        string

--> typeof("5+3*2"), typeof("%t | %f")
    ans  =
        string
    ans  =
        string
```

☞ Tout élément encadré par des guillemets est de type `string` et sa valeur est la chaîne de caractères entre ces guillements.

```
--> a = "je vais" + "bien" + "et vous ?" , typeof(a)
    a  =
        je vaisbienetvous ?
    ans  =
        string

--> b = "je vais" + " bien " + "et vous ?"
    b  =
        je vais bien et vous ?
```

☞ L'opérateur `+` permet de coller deux chaînes de caractères. On fera attention à la gestion des espaces qui sont considérés comme des caractères à part entière.

```
-->"j'ai 20 ans"
  !--error 276
Opérateur, virgule ou point-virgule manquant.

-->"j''ai 20 ans"
    ans  =
        j'ai 20 ans
```

☞ Si l'on souhaite écrire une phrase avec une apostrophe, il faudra doubler le symbole et ce afin de supprimer toute ambigüité (le symbole ' est aussi utilisé pour créer des chaînes de caractères).

### Conversion explicite de type

En **Scilab**, il est possible de convertir des objets de certains types vers un autre type. On s'intéresse essentiellement à la fonction `string`[1] qui a pour spécification `string : constant → string` et permet de convertir tout objet de type `constant` en un objet de type `string`.

```
-->a = string(5), typeof(a)
    a  =
        5
    ans  =
        string
```

Cette fonctionnalité nous sera très utile, notamment lorsque nous voulons concaténer une chaîne de caractères avec un objet de type `constant`.

```
-->x = 20;
-->"j''ai " + x + " ans"
  !--error 144
Opération non définie pour les opérandes donnés.

-->"j''ai " + string(x) + " ans"
    ans  =
        j'ai 20 ans
```

En fait, l'opérateur `string` est aussi surchargé. Ce qui permet notamment de convertir en chaînes de caractères des objets de type `boolean`.

```
-->string(%t)
    ans  =
        string
```

Nous pouvons maintenant expliquer l'utilisation de l'opérateur `string` dans le chapitre précédent. La valeur de la variable `xPlus` (de type `constant`) est convertie en un objet de type `string` afin d'être concaténée à la phrase `"La plus grande ..."`. Pour illustrer la possibilité de concaténer en une fois plusieurs chaînes de caractères, nous rajoutons une phrase à l'affichage final.

```
// Test de la fonction racP avec demande des entrées
// et affichage des sorties
a = input("Entrez la valeur du coefficient a : ");
b = input("Entrez la valeur du coefficient b : ");
c = input("Entrez la valeur du coefficient c : ");
[xPlus, xMoins] = racP(a, b, c);
disp("Voici les deux racines du polynôme " + string(a) +
"X^2 + " + string(b) + "X + " + string(c))
disp("La plus grande racine vaut " + string(xPlus));
disp("La plus petite racine vaut " + string(xMoins));
```

Ce qui permet d'obtenir, lors de l'exécution, l'affichage suivant.

```
-->exec('/Info/Exemple_cours/racPDispInp.sce', -1)
Entrez la valeur du coefficient a : 3
Entrez la valeur du coefficient b : 5
Entrez la valeur du coefficient c : 2

  Voici les deux racines du polynôme 3X^2 + 5X + 2

  La plus grande racine vaut -0.6666667

  La plus petite racine vaut -1.
```

[1] Malheureusement, la fonction `string` porte le même nom que le type de données `string` ...

## V. Le type `function`

Il existe plusieurs autres types de données en **Scilab**, qu'il n'est pas utile de connaître à notre niveau. Nous discutons ici brièvement de la gestion des fonctions dans le simple but d'étoffer notre culture informatique.

En **Scilab**, les fonctions sont des objets qui sont séparés en deux catégories. Les fonctions prédéfinies telles que `log`, `sqrt`, `abs` sont regroupées dans le type `fptr`. Les fonctions définies par l'utilisateur à l'aide de la construction `function ...endfunction` sont regroupées dans le type `function`[2].

```
--> typeof(sqrt), typeof(racP)
 ans  =
     fptr
 ans  =
     function
```

Les fonctions étant des objets à part entière, il est notamment possible d'affecter la valeur d'une fonction à une variable ...

```
--> h = sqrt; h(9)
 ans  =
     3.
```

... ou encore de prendre des fonctions comme paramètres d'une fonction.

```
// Cette fonction prend en paramètre les fonctions f et g
// et l'élément x et calcule la composée f o g au point x
function y = composeePoint(f, g, x)
    y = f(g(x));
endfunction
```

```
--> composeePoint(exp, log, 5)
 ans  =
     5.
```

## VI. Les erreurs de type

En primaire, vous avez appris que l'on n'additionne pas les choux et les carottes. L'idée derrière cette maxime est qu'il ne faut pas mélanger les différentes unités de mesure. On ne peut additionner des mètres et des centimètres (sauf à tout convertir en centimètres !) ; des poids et des tailles ; des vitesses et des longueurs. Cette remarque peut être généralisée : il faut respecter les spécifications des opérateurs. Par exemple, on rencontre parfois l'erreur (ou plutôt l'horreur) consistant à écrire : $\sqrt{25} \Leftrightarrow 5$. Ceci n'a pas de sens. En effet, $\sqrt{25}$ et 5 sont des nombres et l'opérateur d'équivalence sert uniquement à combiner des propositions.

En informatique, les opérateurs sont définis pour des objets de certains types. Utiliser un opérateur avec des objets qu'il ne prend pas en charge correspond à effectuer une erreur de type. En **Scilab**, ce genre d'erreurs sera relevé par l'interpréteur du langage.

```
--> log("36")
  !--error 246
Fonction non définie pour le type d'argument donné.

--> 3+"5"
  !--error 144
Opération non définie pour les opérandes donnés.

--> 2*sqrt
  !--error 144
Opération non définie pour les opérandes donnés.
```

☞ Le fait de travailler avec des données typées permet de mettre à jour certaines erreurs. Les types de données font donc partie intégrante des mécanismes de sécurité permettant de protéger le code.

[2] On peut encore une fois déplorer la collision entre le nom d'un type de données et un mot clé du langage.