# Chapitre 4
# Instructions basiques ABAP

## 1. Variables et constantes

Comme dans tout langage de programmation, l'ABAP contient des variables et des constantes. Par définition, une variable est un symbole informatique associant un nom à une valeur qui peut varier durant l'exécution du programme. Cette définition s'applique également à une constante, à la différence près que sa valeur est fixée dès le début et ne changera jamais au cours de l'exécution du programme.

On retrouve sur SAP les types de variables suivants :

**Types de variables ABAP**

| Type | Description | Longueur par défaut | Valeur par défaut | Exemple |
|---|---|---|---|---|
| c | Chaîne de caractères alphanumériques | 1 | '' | 'ABC012' |
| n | Numérique | 1 | 0 | 5 |
| d | Date | 8 | 00000000 | 20090412 |
| t | Heure | 6 | 000000 | 134523 |
| x | Hexadécimal | 1 | X'0' | 65AF |
| i | Entier | 4 | 0 | 5 |
| p | Nombre à virgule | 8 | 0 | 5,6 |
| f | Format scientifique | 8 | 0 | 2,2 E+209 |

| Type | Description | Longueur par défaut | Valeur par défaut | Exemple |
|---|---|---|---|---|
| string | Texte long | Variable | '' | N'importe quelle chaîne de caractères |
| xstring | String d'hexadécimal | Variable | | N'importe quelle chaîne hexadécimale |

Remarque

*Le format date est de type AnnéeMoisJour (AAAAMMJJ), pour un affichage plus adéquat, il faudra toujours modifier la variable* `date` *(voir un exemple dans la section Opérations sur variable texte de ce chapitre).*

Si le type entier **i** et le type numérique **n** sont comparés, il apparaît qu'ils sont sensiblement les mêmes. En fait, pour apporter un peu plus de précisions, le type entier **i** est, comme son nom l'indique, une chaîne numérique de nombres entiers alors que le type numérique **n** est aussi une chaîne numérique mais stockée sous forme de caractères, ce qui est pratique lors d'un travail avec des instructions sur des variables texte comme le **CONCATENATE** (cf. section Opérations sur variable texte de ce chapitre).

Maintenant que les différents types de variables ont été vus, il reste à savoir comment elles sont déclarées en ABAP. Pour cela, on utilise l'instruction `DATA` puis le nom de la variable (libre), sa longueur entre parenthèses, l'instruction de référence (qui peut être égale à `TYPE`, `LIKE`... comme dans l'exemple ci-dessous) puis son type :

```
DATA v_name(10) TYPE c.
DATA v_date     TYPE d.
DATA v_hour     TYPE t.
DATA v_year(4)  TYPE n.
```

Pour que ce soit plus lisible, l'instruction **`DATA`** peut être suivie de deux points ':' pour ajouter plusieurs paramètres à sa composante. Ainsi, il apparaîtra :

```
DATA: v_name(20) TYPE c,
      v_date     TYPE d,
      v_hour     TYPE t,
      v_year(4)  TYPE n.
```

**Remarque**

*Chaque fin de ligne se termine par une virgule (`,`) afin de séparer les paramètres composant le **`DATA`** mais cela ne signifie pas que ce soit la fin de l'instruction. Comme déjà évoqué dans le chapitre Premiers pas sur SAP - Premier programme ABAP, la fin d'une instruction en ABAP se termine toujours par un point (`.`).*

Quatre variables ont été créées : la variable `V_NAME` de type chaîne de caractères avec une longueur de dix positions, `V_DATE` de type date et donc d'une longueur fixe de huit positions, `V_HOUR` de type heure et donc aussi avec une longueur fixe mais de six positions, et enfin la variable `V_YEAR` de type numérique avec une longueur de huit positions.

**Remarque**

*La nomenclature choisie ici commence par `v_` mais est totalement libre. D'un point de vue personnel, les variables utilisées commencent toujours par `GV_` pour des variables globales, `LV_` pour celles en locales, `PV_` pour une variable en paramètre... Ainsi un simple coup d'œil sur le nom indique très rapidement à quel niveau la variable a été déclarée.*

Comme cité un peu plus haut, l'instruction de référence peut être égale à **`TYPE`** ou **`LIKE`**. Pour comprendre la différence entre les deux, voici un exemple :

```
DATA: v_name(10) TYPE c,
      v_name2    LIKE v_name.
```

La variable `V_NAME` est déclarée avec un type chaîne de caractères et de longueur 10 et `V_NAME2` quant à elle prend comme référence la variable `V_NAME`. Ainsi, **`TYPE`** va pointer directement vers un type spécifique alors que le **`LIKE`** va en prendre indirectement une référence. Dans une programmation objet (cf. chapitre Les classes), le **`TYPE`** est à privilégier.

Concernant les constantes, c'est exactement le même principe sauf que l'instruction commencera par **`CONSTANTS`** et devra comporter obligatoirement une valeur fixe avec **`VALUE`**.

```
CONSTANTS : c_yes(1)    TYPE c VALUE 'X',
            c_answer(2) TYPE i VALUE '42',
            c_pi        TYPE p DECIMALS 2 VALUE '3.14'.
```

Trois constantes ont été déclarées : la première `C_YES` de type chaîne de caractères alphanumériques de longueur d'une seule position aura comme valeur `'X'`, la deuxième `C_ANSWER` est un type entier d'une longueur de deux positions et d'une valeur de `'42'`, et enfin la dernière `C_PI` est un nombre avec virgule d'une valeur de `'3,14'`. Il est à noter qu'en ABAP, le point (`.`) est utilisé pour les décimales.

**Remarque**

*Le type `p` doit être accompagné par l'instruction **`DECIMALS`** qui va définir le nombre de chiffres après la virgule, sinon la variable associée sera considérée comme un nombre entier.*

Il existe aussi des variables système en arrière-plan, accessibles par tous les programmes, et dans les mandants, stockées dans la table **SY**.

Elles informent du statut global du système comme :

| | |
|---|---|
| Le mandant de connexion | `SY-MANDT` |
| Le nom de l'utilisateur | `SY-UNAME` |
| La date | `SY-DATUM` |
| L'heure | `SY-UZEIT` |

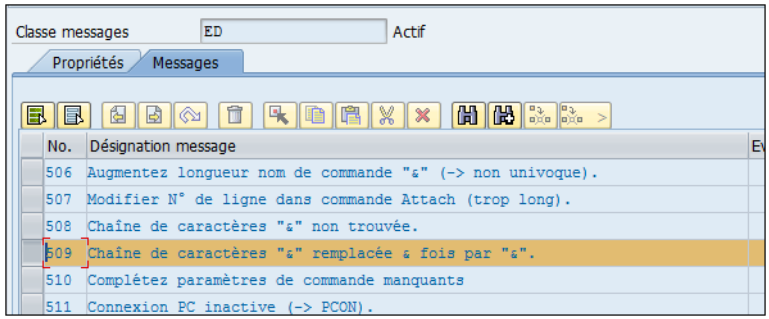
Ainsi que des informations plus spécifiques au programme exécuté, notamment :

| | |
|---|---|
| Code retour d'un traitement | `SY-SUBRC` |
| Statut du message après exécution d'une opération spécifique | `SY-MSGTY` |
| Le numéro de la ligne d'une table lue lors d'une boucle sur table interne (cf. chapitre Les tables internes - Organisation et Lecture) | `SY-TABIX` |
| Le numéro d'un index utilisé également dans une boucle de type `DO` ou `WHILE` (voir un peu plus loin la section sur Les boucles) | `SY-INDEX` |

**Remarque**

*Le statut de message `SY-MSGTY` sera toujours accompagné d'une classe de message (`SY-MSGID`), et de son numéro (`SY-MSGNO`), avec les paramètres (`SY-MSGV1`, `SY-MSGV2`, `SY-MSGV3`, et `SY-MSGV4`) consultables via la transaction SE91.*

Exemple avec la classe de message ED, numéro de message 509



Ce message comporte trois paramètres symbolisés par le caractère '&'. Ainsi la valeur du premier paramètre sera stockée dans la variable système SY-MSGV1, le deuxième dans SY-MSGV2, et le troisième dans SY-MSGV3 (le quatrième sera vide car non utilisé). Aussi, si les valeurs sont définies de la façon suivante :

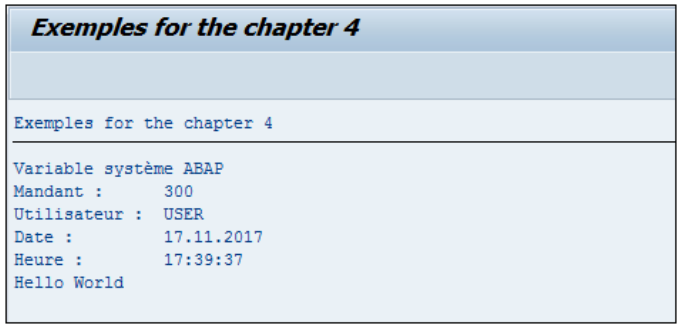
- SY-MSGTY = 'I'
- SY-MSGID = 'ED'
- SY-MSGNO = '509'
- SY-MSGV1 = 'Hello'
- SY-MSGV2 = '3'
- SY-MSGV3 = 'Bye'

SAP aura toutes les informations pour construire un message d'information (I) comportant le texte : Chaîne de caractères "Hello" remplacée 3 fois par "Bye".

Pour illustrer ce chapitre un petit programme pour afficher quelques informations du système et une constante contenant la valeur 'Hello world'.

```
CONSTANTS : c_text(11)  TYPE c VALUE 'Hello World'.

WRITE:/ 'Variable système ABAP'.
WRITE:/ 'Mandant :     ', sy-mandt.
WRITE:/ 'Utilisateur : ', sy-uname.
WRITE:/ 'Date :        ', sy-datum.
WRITE:/ 'Heure :       ', sy-uzeit.
WRITE / c_text.
```



Remarque

*Ici aussi, on peut utiliser les deux points (':') avec l'instruction* ***WRITE*** *qui contiendra ainsi une série de paramètres. Elle est suivie de la barre '/' signifiant une nouvelle ligne, puis d'un texte lui-même suivi d'une virgule, et enfin une variable. Ainsi, comme pour l'exemple du* `DATA` *en début de chapitre, il aurait été possible d'écrire ce code de la manière suivante :*

```
CONSTANTS : c_text(11)  TYPE c VALUE 'Hello World'.

WRITE:/ 'Variable système ABAP',
      / 'Mandant :     ', sy-mandt,
      / 'Utilisateur : ', sy-uname,
      / 'Date :        ', sy-datum,
      / 'Heure :       ', sy-uzeit,
      / c_text.
```

## 2. Opérations arithmétiques

Comme dans tout langage de programmation, les variables de types numériques (entier, avec décimales...) peuvent être utilisées dans des opérations arithmétiques.

Tout d'abord, pour assigner une valeur à une variable, les instructions **MOVE** ou égal (**=**) sont utilisées.

Exemple

```
DATA: v_a(2) TYPE i,
      v_b(2) TYPE i,
      v_c(2) TYPE i,
      v_d(2) TYPE i.

v_a = 3.
v_b = v_a.
MOVE 5   TO v_c.
MOVE v_c TO v_d.

WRITE:/ 'Valeur de v_a : ', v_a,
      / 'Valeur de v_b : ', v_b,
      / 'Valeur de v_c : ', v_c,
      / 'Valeur de v_d : ', v_d.
```