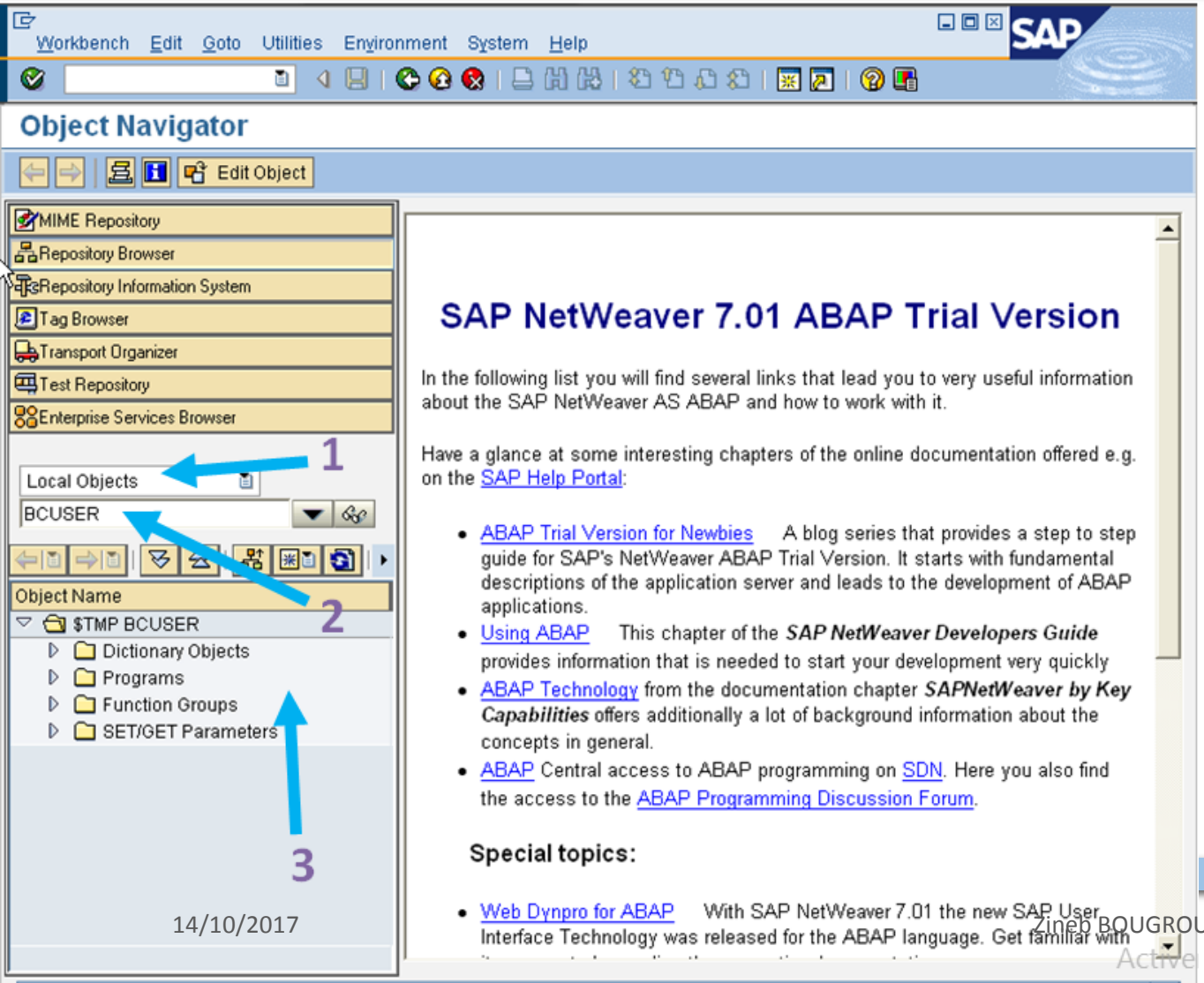
SAP
Syntaxe général

Zineb BOUGROUN

Plan

- SE80
- Hello world
- Syntaxe générale
- Conditions
- Boucles

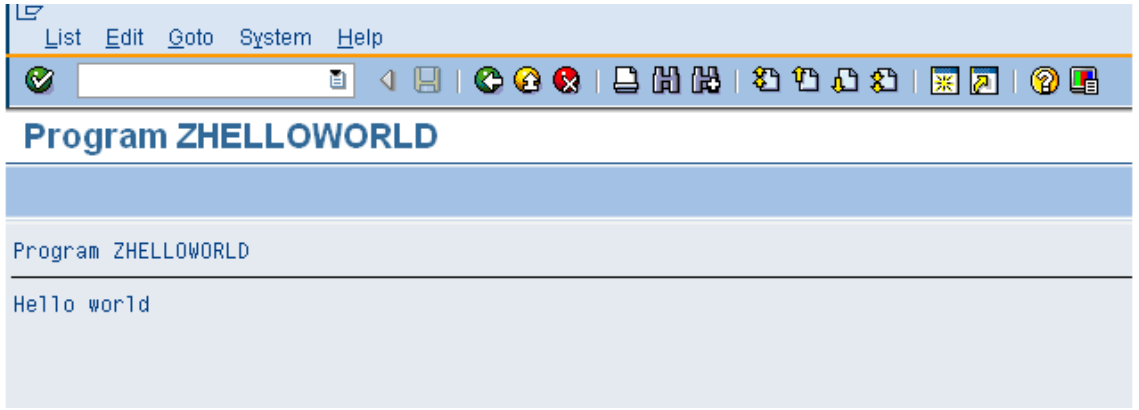
SE80



- Pour les développeurs ABAP, cette transaction est la plus importante (au sens où vous pouvez faire presque tout à partir de cette transaction).
  A partir de cette transaction, il est possible de créer, modifier, ou visualiser :
- les "programs"
- les "packages"
- les "tables"
- les "views"
- les "domains"
- les "data elements"
- ...

Hello world

```
*&---------------------------------------------------------------------*
*& Report  ZHELLOWORLD
*&
*&---------------------------------------------------------------------*
*&
*&
*&---------------------------------------------------------------------*

REPORT  ZHELLOWORLD. "Le nom du programme, généré automatiquement lors de la création.

DATA v_phrase TYPE c LENGTH 20. "Une déclaration d'une variable  de type chaine

*de caractères, d'une longueur de 20.

MOVE 'Hello world' TO v_phrase. "Une affectation (le littéral 'Hello world')

*à la variable v_phrase.

WRITE v_phrase.  "Ecriture de la valeur de la variable à l'écran.
```



Les commentaires :

Une [*] en tout début de ligne commente l'ensemble de la ligne.

Code ABAP

- Une première ligne de commentaires*Une deuxième ligne de commentaires

La déclaration des variables :

En ABAP, comme en Java ou en C++, toutes les variables doivent être déclarées. Ceci se fait par le mot clé "DATA".

L'assignation :

L'assignation peut se faire par la syntaxe :
variable_1 = variable_2.
Mais il est plus courant d'utiliser la syntaxe:
MOVE variable_2 TO variable_1.

Syntaxe générale

- Une instruction se termine par un point « . »
- Une instruction peut tenir sur plusieurs lignes (seul le point marque la fin de l'instruction).
- La casse n'a pas d'importance (ni pour les mots clés ni pour les variables).
- La déclaration des variables se fait par l'instruction « **DATA : »**
- Les données doivent être déclarées
- Il existe plusieurs types de variables, entre autres :
  - Les variables : qui seront nommées v_ma_variable
  - Les structures : qui seront nommées wa_ma_structure
  - Les tables internes : qui seront nommées it_ma_table_interne
- Attention : dans la documentation, vous retrouverez souvent le terme "work area". Il s'agit en fait d'une structure.

Déclaration et assignation

- **Déclaration**
- **DATA <nom_de la_variable> TYPE <nom_du_type> <options>.**
  - Exemple : DATA v_phrase TYPE c LENGTH 20
- **Assignation :**
- L'assignation se fait de deux façons :

```
DATA v_ma_phrase1 type c length 20.
DATA v_ma_phrase2 type c length 20.

v_ma_phrase1 = 'Bonjour le monde'.
v_ma_phrase2 = v_ma_phrase2.

MOVE 'Salut la Terre' TO v_ma_phrase2.
MOVE v_ma_phrase2 TO v_ma_phrase1.
```

- Si les deux variables sont du même type, alors le MOVE est plus performant.
  Attention : respectez bien les espaces de chaque côté du [=]. En ABAP, il est important de respecter les espaces, sinon le contrôle syntaxique risque d'échouer (et la description de l'erreur et plus que floue). Pour tout type d'instruction, faites donc bien attention aux espaces.

Syntaxe générale

- **La notation ":" :**
- En ABAP, il est possible d'utiliser une syntaxe spéciale pour éviter de répéter plusieurs fois une même instruction.
  Par exemple, au lieu d'écrire :

- DATA v_variable1 type c.
  DATA v_variable2 type c.
  DATA v_variable3 type c.
  write 'Bonjour '.
  write 'le '.
  write 'monde.'.
- DATA : v_variable1 type c,
  v_variable2 type c,
  v_variable3 type c.

  write : 'Bonjour ', 'le ', 'monde.'.

Les variables système

- Ce sont des variables réservées qui sont valorisées par le système ; Elles permettent de récupérer un certains nombre d'informations liées au contexte.

- **SY-SUBRC :** Indique le bon déroulement d'une instruction (vaut 0 si OK)
- **SY-UNAME :** Contient le login de l'utilisateur
- **SY-DATUM :** Contient la date du jour
- **SY-LANGU :** Indique la langue de connexion
- …

Types

- **Déclaration de type :**
  - déclarer un nouveau type dans le dictionnaire de données.
  - déclarer des types directement dans un programme ABAP.
- Un nouveau type déclaré dans le dictionnaire de données (un **data element** donc) à l'avantage d'être visible par l'ensemble du système, et donc d'être réutilisable.
- Un nouveau type déclaré directement dans le code ABAP n'a d'existence que pour le programme dans lequel il est déclaré.

  La syntaxe est la suivante :

  TYPES PRENOM(10) TYPE C.
  TYPES NUMBER TYPE I.

  Dans le premier exemple on déclare un nouveau type qui se dénomme PRENOM, et qui consiste en une chaine de caractères d'une longueur 10.
  Dans le deuxième exemple, on déclare un nouveau type qui se dénomme NUMBER, et qui est identique au type standard I (integer).

Types

- **Utiliser les types**
- Les types décrivent les attributs
  - des zones de saisie et de sortie sur les écrans,
  - des objets de données
  - des paramètres d'interface

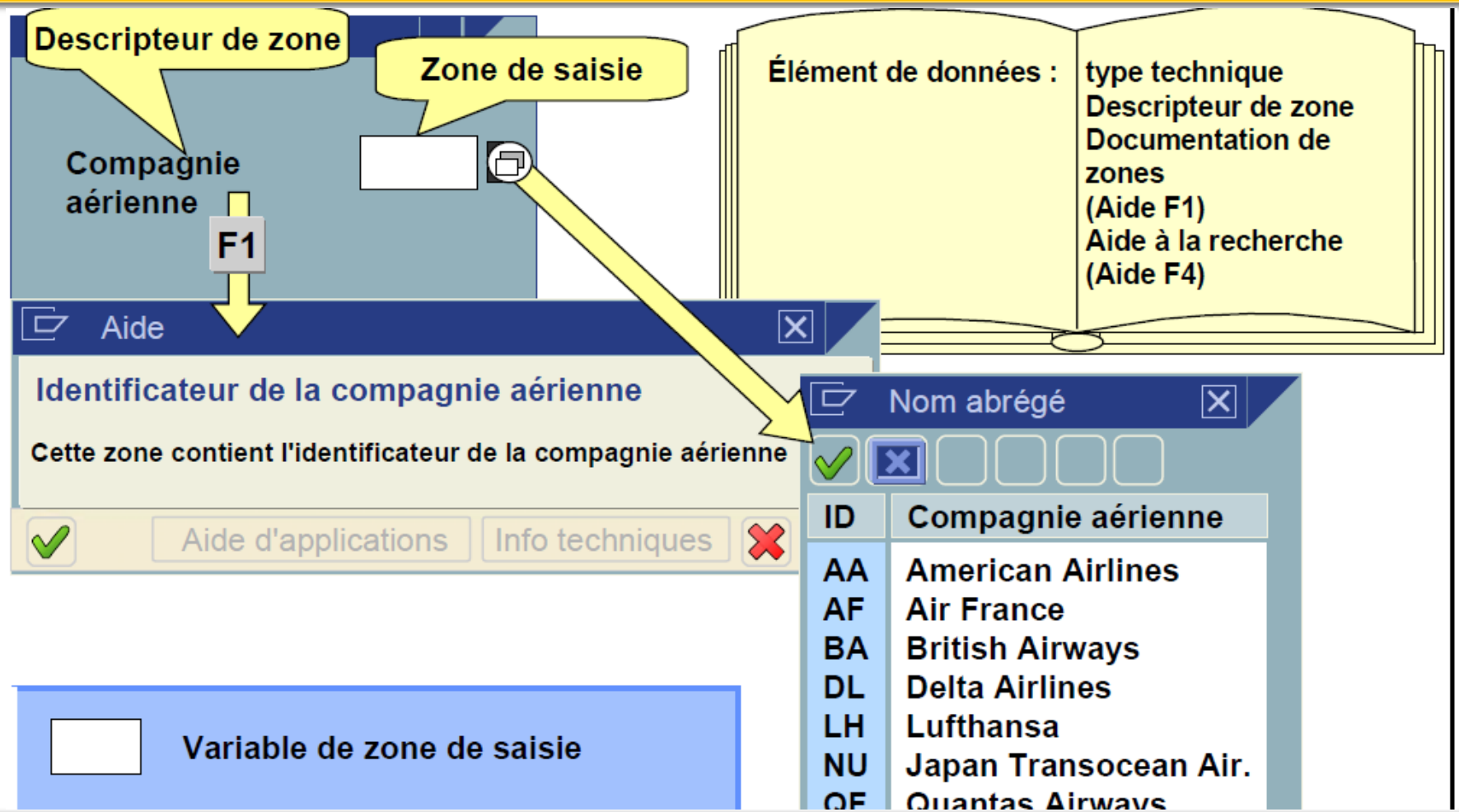
Types

- **Faire référence aux tables :**
- À tout moment, il est possible d'effectuer une requête SQL (OpenSQL) directement dans le code. Il est aussi possible de faire référence aux types des champs d'une base de données, pour nous éviter à aller chercher le « data element » correspondant à tel ou tel champ de la table.
  Exemple :
  Nous voulons déclarer une variable pour stocker l'id d'une compagnie aérienne. Notre système SAP possède la table SFLIGHT, qui possède justement une colonne pour les id des compagnies aériennes. Il s'agit du champ CARRID (en anglais, compagnie aérienne se dit CARRIER, donc CARRID signifie CARRIER ID).
  Au lieu de chercher à retrouver le « data element » correspondant à cet ID, nous allons réutiliser la donnée.
  L'instruction DATA nous permet de faire référence à un champ d'une table, nous pouvons donc écrire :
- **DATA v_carrid TYPE sflight-carrid.**
- Cette instruction signifie que nous déclarons la variable v_carrid, qui est du même type que la colonne CARRID de la table SFLIGHT.

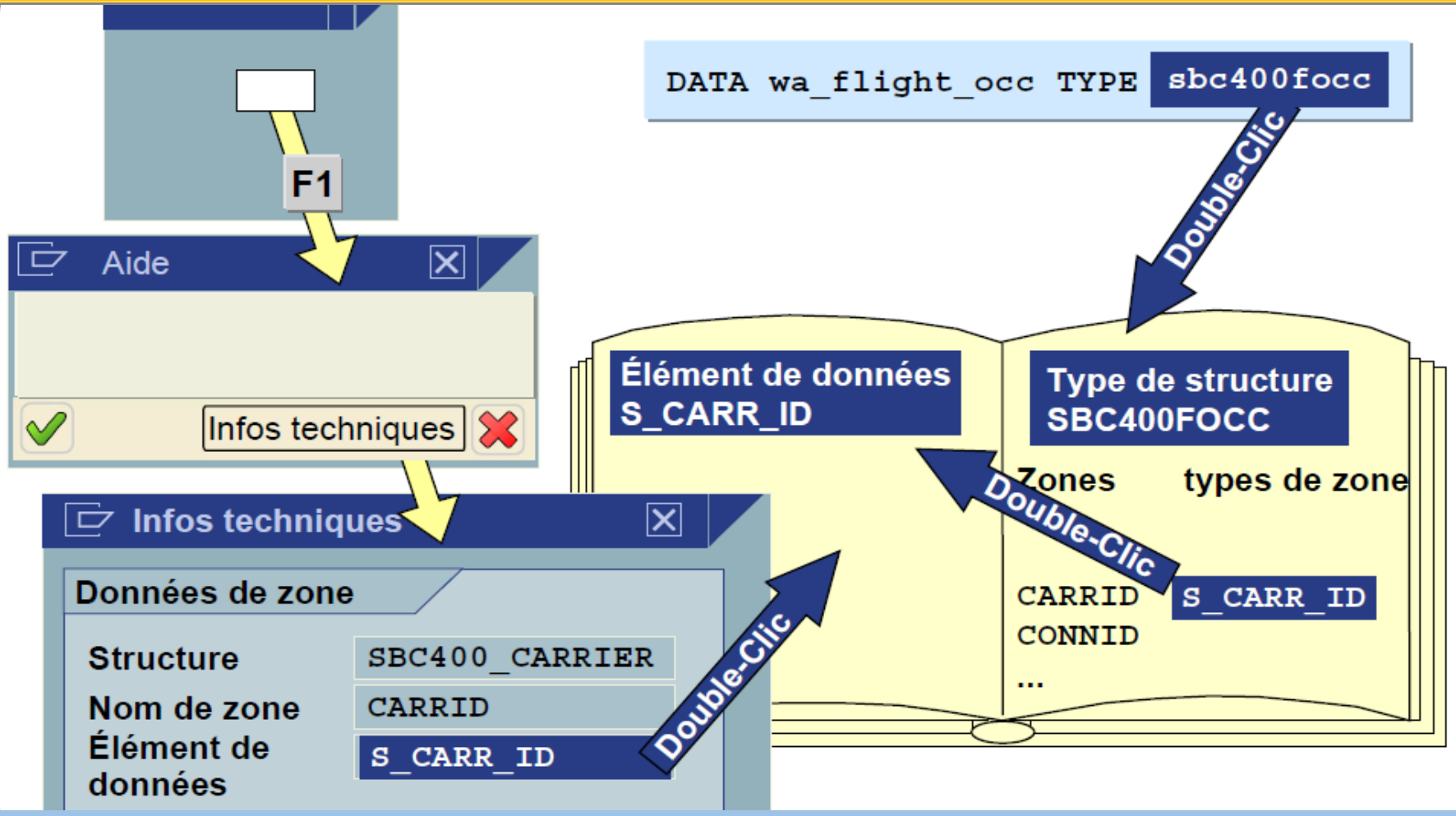
Types de données locaux et généraux

- **Les types locaux sont utilisés dans des programmes**
  - si seuls des attributs techniques sont nécessaires et qu'aucun attribut sémantique n'est requis et
  - si les types sont uniquement utilisés localement dans un programme.
- **Types globaux (= types du dictionnaire ABAP) sont utilisés**
  - si vous souhaitez utiliser les types de manière externe (par exemple, pour définir les paramètres d'interface des fonctions générales ou les objets de données du programme qui servent d'interface à la base de données ou au serveur de présentation),
  - si vous souhaitez également des informations sémantiques (par exemple, sur les écrans avec des zones de saisie et de sortie).

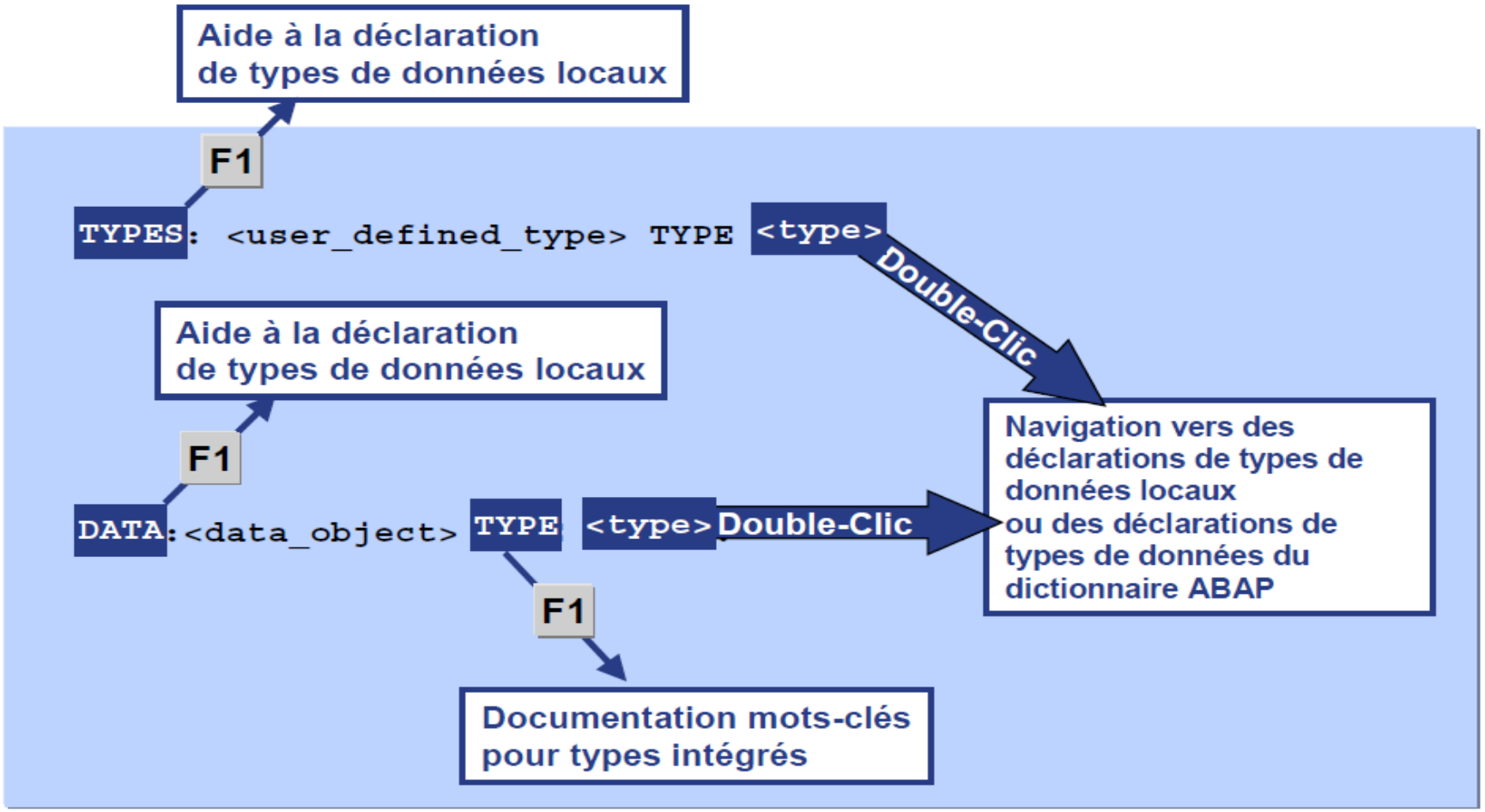
Utilisation des types élémentaires du dictionnaire



Recherche des types du dictionnaire ABAP



Types de données locaux dans les programmes



Types de données locaux dans les programmes

Lorsque vous définissez des types ou des variables simples, vous pouvez vous référer à un type prédéfini.

Les types numériques

- **Type I :** Nombre entier signé ; valeurs comprises entre-2147483648 et 2147483647.

  DATA nom_variable TYPE I.

- **Type P :** 15 caractères

  DATA nom_variable(5) TYPE p DECIMALS 2.

- **Type F :** Format décimal scientifique ; valeurs comprises entre 2.2...E-308 et 1.7...E+308

  DATA nomvariable type f.

-

Types de données locaux dans les programmes

Les types caractères

- **Type C :** Définit une chaîne de caractères ; Elle peut avoir jusqu'à 65535 caractères.

  DATA nom_variable(10) TYPE C. Déclare une chaîne de 10 caractères.

- **Type N :** Correspond aux chaînes numériques , elle peut contenir jusqu'à 65535 caractères.

  La valeur numérique stockée est précédée par des 0.

  DATA nom_variable(10) TYPE N. Déclare une chaîne numérique de 10 caractères.

- **Type STRING :** Chaine de caractère sans limite de longueur

  Data nom_variable TYPE STRING.

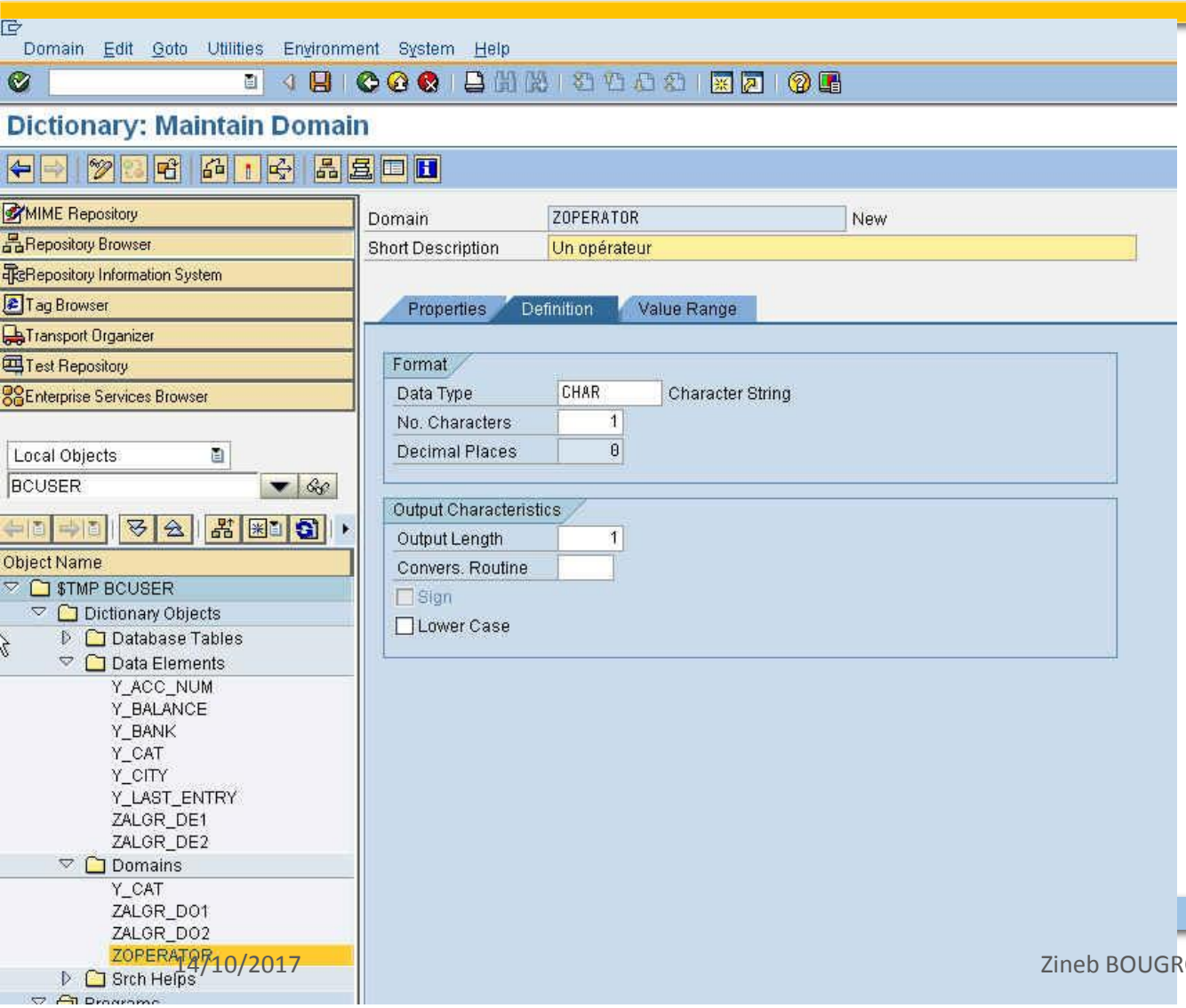
Les types dates

- **D Date (YYYYMMDD)**
- **T Heure (HHMMSS)**

Domain et Data Element

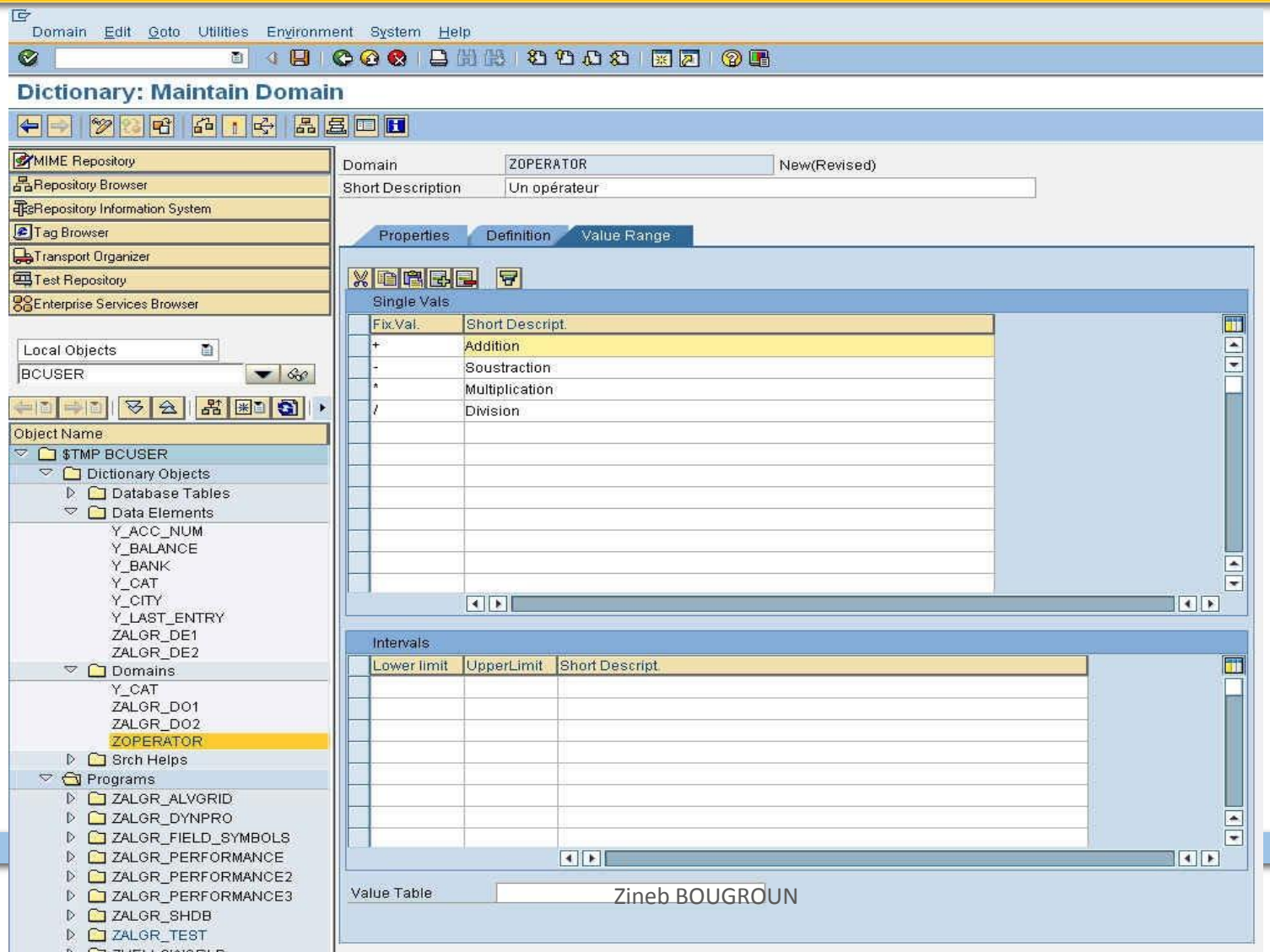
- **Domain :**
- C'est la définition technique d'un type de donnée.
- Par exemple, si nous utilisons un type pour définir un opérateur, et que ce type n'existe pas.
- Nous pouvons représenter un opérateur par les caractères [+ - * /] dans notre programme.
- Nous creyons donc un "Domain" qui nous indique qu'un opérateur est une chaine de caractères de longueur 1 qui peut prendre uniquement les valeurs [+ - * /].

Domain et Data Element



- Création de domaine
  - Data type est le type primitif (CHAR signifie "Character String").
  - No. Characters est la longeur de la chaine.
  - Output lenght est la longueur qui doit être réservé pour l'affichage.

Domain et Data Element



Domain et Data Element

- **Data Element :**
- Un Data Element est un type fonctionnel, c'est à dire une surcouche d'un type technique (le "Domain").
- Dans notre exemple, le domaine "ZOPERATOR" n'est pas utilisable dans du code ABAP.
- **On ne peut pas écrire par exemple :**

```
*&---------------------------------------------------------------------*
*& Report  ZALGR_TEST
*&
*&---------------------------------------------------------------------*
*&
*&
*&---------------------------------------------------------------------*
REPORT  zalgr_test.

DATA :
    v_operator TYPE zoperator.
```

Domain et Data Element

- En ABAP, on ne code qu'à partir des types fonctionnels, et non des domaines.
- un data element sert à définir les libellés associé à ce type (ce libellé se retrouvera lors des affichages à l'écran), il sert à définir l'aide en ligne, et il sert à porter le "sens fonctionnel" de la donnée.
- Un domaine (définition technique d'un type), peut par exemple signifier que l'on a besoin d'une chaine de caractères d'une longueur de 8.
- Ce même domaine peut être utilisé par 2 data elements différents, par exemple :
- un premier data element qui représente un numéro d'abonné.
- un autre data element qui représente un numéro de plaque d'immatriculation.
- Même si ces deux types sont identiques au sens technique (même domaine : chaine de caractères de longueur 8), dans un programme le sens fonctionnel est différent. On a donc deux "types fonctionnels" différents, qui sont les "data elements", tous les deux associés au même domaine.

Domain et Data Element

Data Element Edit Goto Utilities Environment System Help

Dictionary: Maintain Data Element

Documentation | Supplementary Documentation

MIME Repository
Repository Browser
Repository Information System
Tag Browser
Transport Organizer
Test Repository
Enterprise Services Browser

Local Objects
BCUSER

Object Name
- $TMP BCUSER
  - Dictionary Objects
    - Database Tables
    - Data Elements
      - Y_ACC_NUM
      - Y_BALANCE
      - Y_BANK
      - Y_CAT
      - Y_CITY
      - Y_LAST_ENTRY
      - ZALGR_DE1
      - ZALGR_DE2
      - Z_OPERATOR
    - Domains
      - Y_CAT
      - ZALGR_DO1
      - ZALGR_DO2
      - ZOPERATOR
    - Srch Helps
    - Programs
      - ZALGR_ALVGRID

Data element: Z_OPERATOR — Active
Short Description: Un opérateur

Attributes | Data Type | Further Characteristics | Field Label

- Elementary Type
  - Domain: ZOPERATOR — Un opérateur
    - Data Type: CHAR — Character String
    - Length: 1 — Decimal Places: 0
  - Predefined Type
    - Data Type:
    - Length: 0 — Decimal Places: 0
  - Reference Type
    - Name of Ref. Type:
  - Reference to Predefined Type
    - Data Type:
    - Length: 0 — Decimal Places: 0

Data Element Edit Goto Utilities Environment System Help

Dictionary: Maintain Data Element

Documentation | Supplementary Documentation

MIME Repository
Repository Browser
Repository Information System
Tag Browser
Transport Organizer
Test Repository
Enterprise Services Browser

Local Objects
BCUSER

Object Name
- $TMP BCUSER
  - Dictionary Objects
    - Database Tables
    - Data Elements
      - Y_ACC_NUM
      - Y_BALANCE
      - Y_BANK
      - Y_CAT
      - Y_CITY
      - Y_LAST_ENTRY
      - ZALGR_DE1
      - ZALGR_DE2
      - Z_OPERATOR
    - Domains
      - Y_CAT
      - ZALGR_DO1
      - ZALGR_DO2
      - ZOPERATOR
    - Srch Helps

Data element: Z_OPERATOR — Active
Short Description: Un opérateur

Attributes | Data Type | Further Characteristics | Field Label

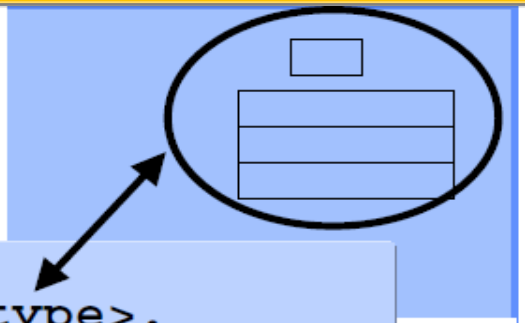
	Length	Field Label
Short	8	Operator
Medium	20	The classic operator
Long	33	The classic operator : + - * /
Heading	33	The classic operator : + - * /

Domain et Data Element

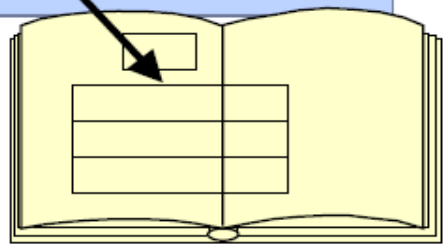
```
*&---------------------------------------------------------------------*
*& Report  ZALGR_TEST
*&
*&---------------------------------------------------------------------*
*&
*&
*&---------------------------------------------------------------------*
REPORT  zalgr_test.

DATA :
    v_operator TYPE z_operator.
```

- **Retenez donc bien les points suivants :**
- Un "Domain" est la définition technique d'un type. Par exemple une chaine de caractères d'une longueur 10.
- Un "Data Element" est associé à un seul "Domain".
- Un "Domain" peut être utilisé par n "Data Elements".
- Un "Data Element" contient la définition des libellés et de l'aide à la recherche.
- Un programme ABAP fait toujours référence à des "Data Element", jamais à des "Domain"."

Objets de données



```
DATA: <varname> TYPE <user-defined-type>.
                     <ABAP-dictionary-type>.
```



```
DATA: <varname> LIKE <data-object>.
```

Différence entre TYPE et LIKE

- TYPE:
  - Il alloue la mémoire pendant l'exécution (type d'objet).
  - Il améliore les performances.
  - Il est utilisé lorsque le lien d'objet défini par l'utilisateur avec le type de données système SAP.
  - Il fait référence aux types de données définis par l'utilisateur
  - Il affecte un type de données directement à l'objet de données lors de la déclaration.
- LIKE:
  - alloue immédiatement la mémoire
  - C'est lorsque l'objet de données est lié à l'autre objet de données.
  - fait référence au type de données existant de l'objet de données
- LIKE, vous attribuez le type de données d'un autre objet à l'objet de données déclarant. Le type de données est référencé indirectement.

Objets de données

- Règles de nommage des objets de données :
  - un nom peut comporter 30 caractères maximum (lettres, nombres, ou symboles).
  - les symboles suivants NE SONT PAS autorisés : ( ) +. , :
  - SPACE est une zone prédéfinie.

Objets de données

```
CONSTANTS <constants> TYPE <type> VALUE <literal>.
```

```
CONSTANTS: PI       TYPE P DECIMALS 4 VALUE '3.1415',
           BOSS(4) TYPE C VALUE 'Hugo'.
```

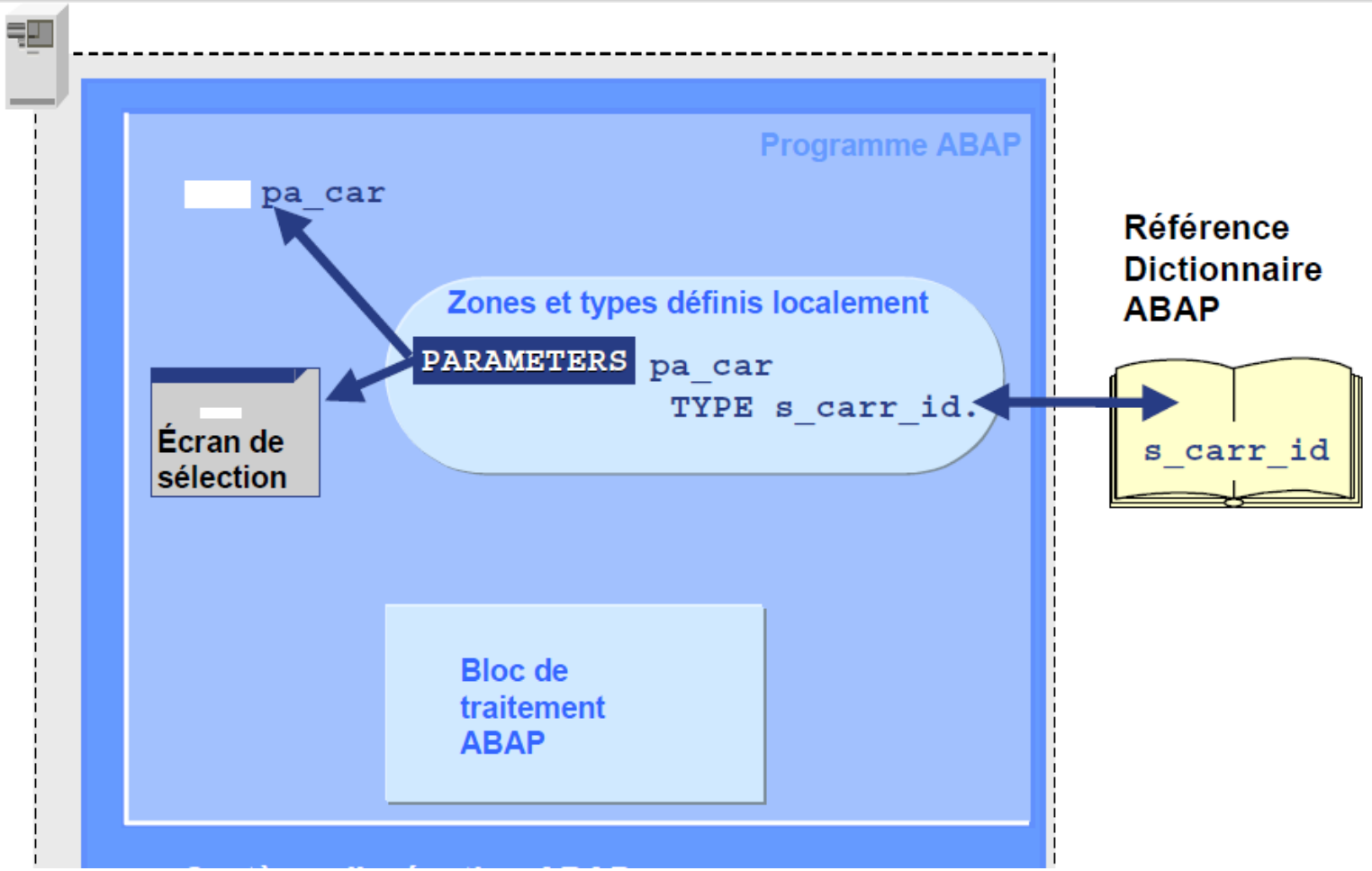
Littéral numérique

715, -431
Nombre entier (un signe moins est placé en face de nombres négatifs)

Littéral alphanumérique

'abc', '2.17', '1.213E15', '0FFF', 'A''B'
Séquence de caractères terminée par des guillemets simples, 255 caractères max.

Instruction PARAMETERS



Instruction PARAMETERS

- Dans un programme exécutable, une seule instruction **PARAMETERS suffit pour générer un écran de** sélection standard.
- L'instruction

  PARAMETERS <nom> TYPE <type>

  comme l'instruction

  PARAMETERS <nom> LIKE <objet de données>

  génèrent toutes deux une zone de saisie simple sur l'écran de sélection, et un objet de données **<nom> du type spécifié.**
- Si l'utilisateur saisit une valeur et choisit "Exécuter", cette valeur est placée dans l'objet de données interne **<nom> du programme. Le système n'autorise que les entrées de type approprié.**

Condition

- **IF/ELSEIF/ELSE/ENDIF :**

```
IF ( 1 eq 2 ).
    write : '1 égal 2'.    skip 1.
ELSEIF ( 1 lt 0 ).
    write : '1 inférieur ou égal à 0'.
   skip 1.
ELSE
    write : 'aucun des deux'.
      skip 1.
ENDIF.
```

- Evitez les ELSEIF, ce n'est pas très lisible. Il faut mieux imbriquer un autre IF/ELSE dans un IF, ou bien utiliser un CASE (voir plus bas) s'il y a beaucoup de possibilités.

Condition

- **CASE / ENDCASE :**
- data v_operateur(1) type c value '*'.

```
CASE v_operateur.
    WHEN '-'.
       write : 'v_operateur vaut -'.
    WHEN '+'.
       write : 'v_operateur vaut +'.
    WHEN OTHERS.
       write : 'v_operateur vaut ni + ni -'.
ENDCASE.
```

Évaluation du contenu d'une zone

```
CASE <data object1>.
  WHEN <data object2>.
       Instructions
  WHEN <data object4> OR <data object5>.
       Instructions
  WHEN OTHERS.
       Instructions
ENDCASE.
```

```
IF <logical expression>.
       Instructions
ELSEIF <logical expression>.
       Instructions
ELSEIF <logical expression>.
       Instructions
ELSE.
       Instructions
ENDIF.
```

Boucle

- **DO / ENDDO :**

```
DO 3 TIMES.
    WRITE: 'hello'.
    SKIP 1.
ENDDO.
```

L'entier indiquant le nombre de passages à effectuer doit être de type i.

- **WHILE / ENDWHILE :**

```
DATA compteur TYPE i VALUE 0.
WHILE ( compteur LT 3 ).
    WRITE : compteur.
    ADD 1 TO compteur.
ENDWHILE.
```

Exercices

- Écrire un programme qui affiche les nombres de 1 à 10.
- Écrire un programme qui calcule la somme des entiers de 1 à 100.
- Écrire un programme qui saisit deux nombres entiers positifs et calcule le premier à la puissance du second. Il affiche le résultat.