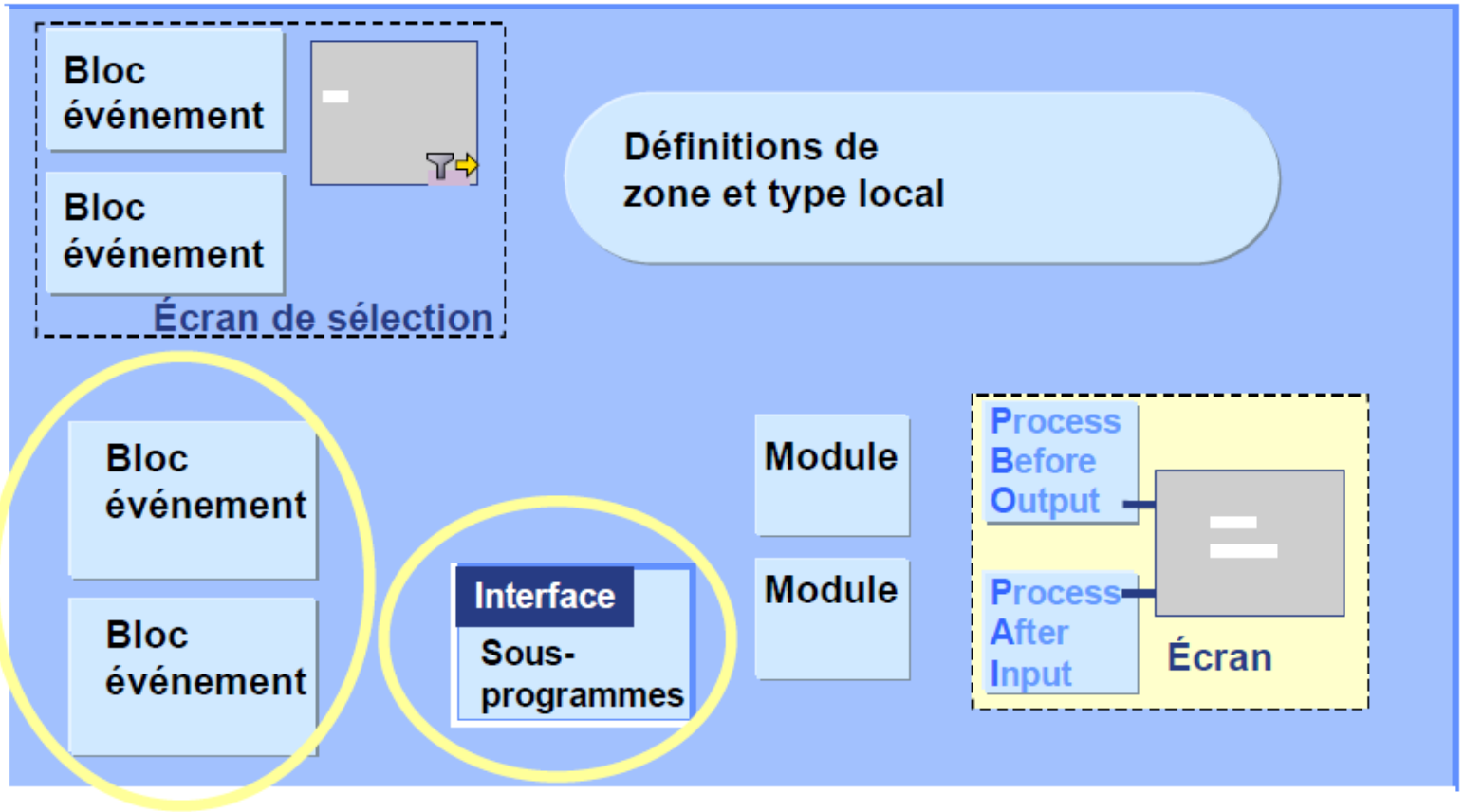
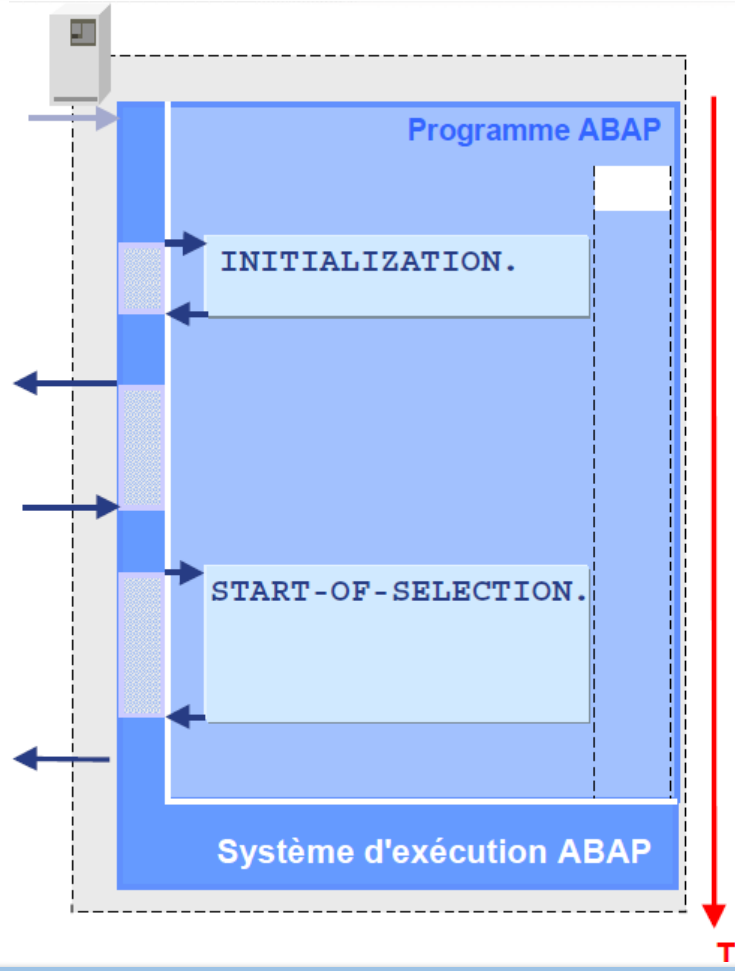
SAP
Les événements

Zineb BOUGROUN

Éléments possibles d'un programme ABAP



Blocs événements dans des programmes exécutables

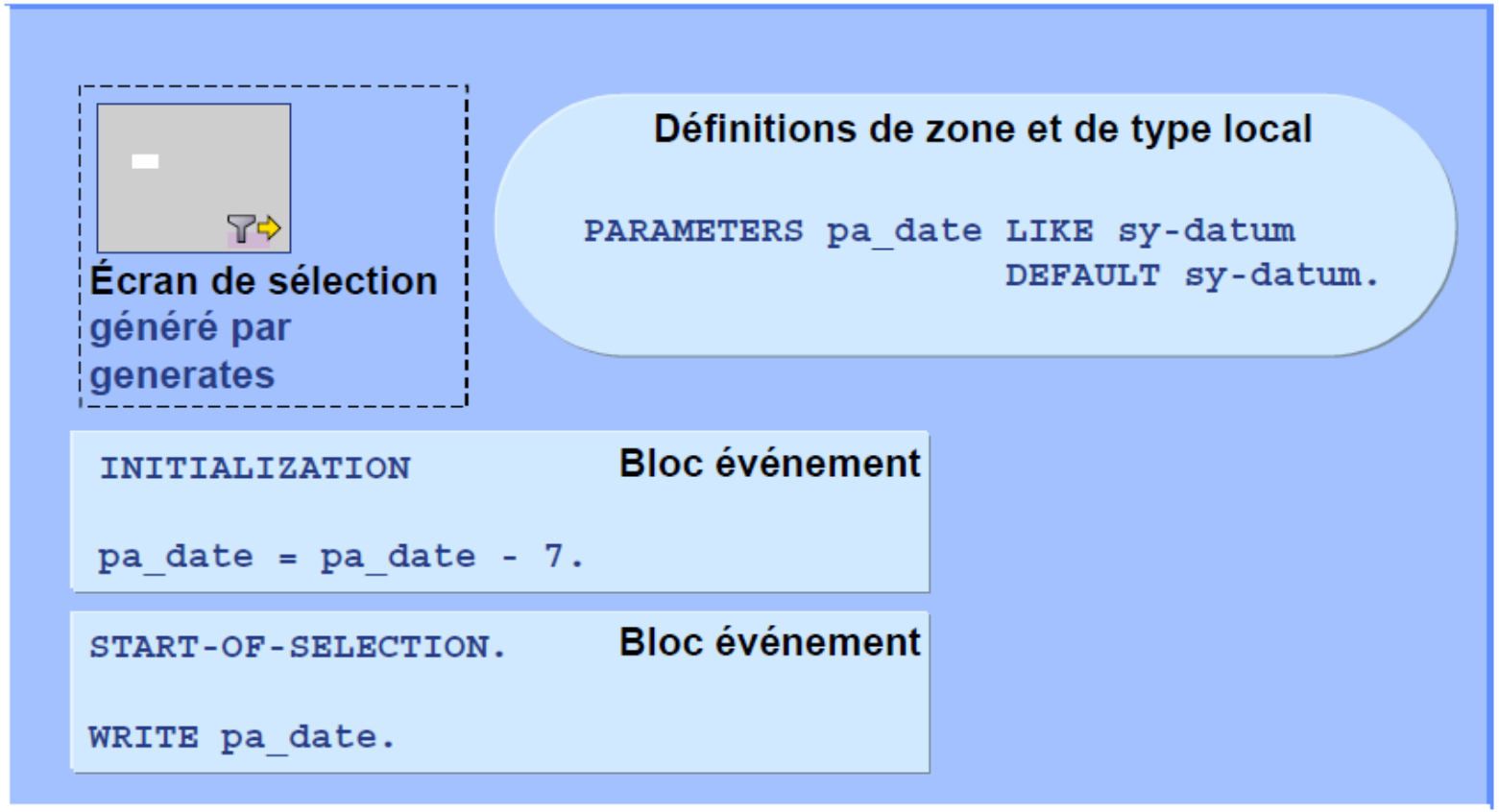


- Introduit par un mot-clé événementiel
- Un événement est délimité par le début du bloc de traitement suivant
- À différents blocs événements correspondent différentes tâches
- Le processeur ABAP détermine l'ordre de traitement des événements
- Bloc événement par défaut : START-OF-SELECTION

Événement

- Lors de l'execution d'un programme ABAP, le système déclenche successivement des événements, qui correspondent à l'état en cours du programme.
- Les différents événements appelés sont successivement :
  - **INITIALIZATION** : appelé en premier, avant l'affichage de l'écran. Après cet évènement, l'écran est affiché. Dans ce bloc, on effectue les initialisations nécessaires, et on "clear" les variables que l'on utilise (instruction CLEAR vide les données).
  - **AT-SELECTION-SCREEN** : appelé lorsque l'utilisateur valide l'écran (après avoir renseigné tous les champs obligatoire), en appuyant sur F8, ou sur l'icone Direct Processing. Dans ce bloc, par convention, on traite éventuellement la saisie de l'utilisateur pour obtenir les variables finales que l'on va utiliser dans la suite du programme.
  - **START-OF-SELECTION** : appelé après AT-SELECTION-SCREEN. Dans ce bloc, par convention, on effectue les requêtes SQL afin de récupérer l'ensemble des données que l'on va utiliser.
  - **END-OF-SELECTION** : appelé après START-OF-SELECTION. Dans ce bloc, par convention, on effectue les traitements et affichages à l'écran (affichage d'un résultat, une impression, etc...).
- On peut très bien omettre des événements, le système considérera qu'il existe mais ne contient rien.
- Un bloc évènement s'arrête au commencement du suivant.

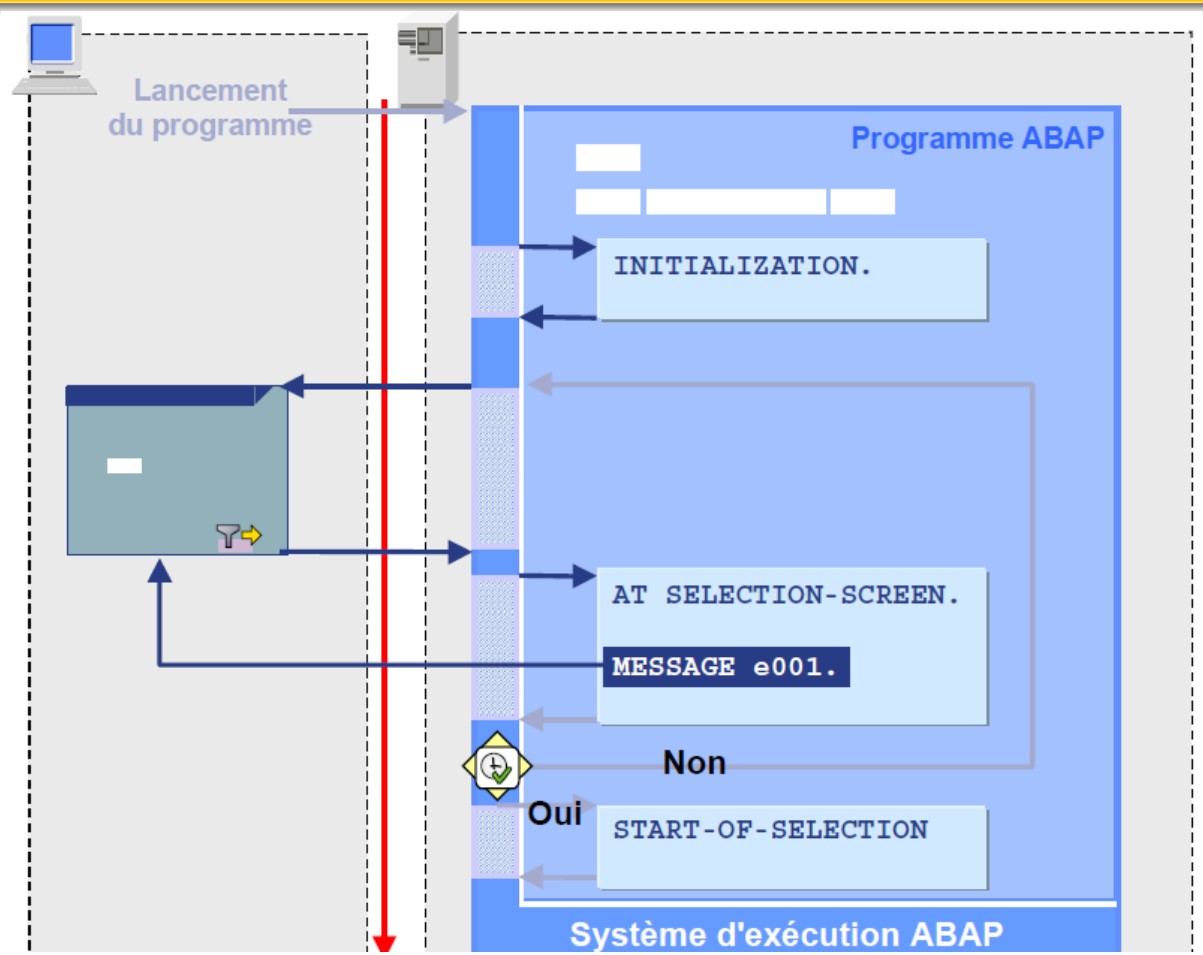
Exemple : programme ABAP comportant des blocs événements et un écran de sélection



Événement AT SELECTION-SCREEN

- Dans un programme exécutable, le système d'exécution ABAP génère un écran de sélection standard si vous avez écrit au moins une instruction **PARAMETERS ou SELECT-OPTIONS. L'écran de** sélection appartient à l'événement **AT SELECTION-SCREEN.**
- L'écran de sélection s'affiche après l'événement **INITIALIZATION.**
- Lorsque vous appuyez sur la touche Entrée ou sur une touche de fonction, cliquez sur un bouton de commande ou bien choisissez une fonction de menu, le système effectue un contrôle de type. Si les entrées ne sont pas du type adéquat, il affiche un message d'erreur et attend une nouvelle entrée. Une fois le type corrigé, il déclenche l'événement **AT SELECTION-SCREEN.**
- Le déroulement ultérieur du programme dépend de l'action utilisateur :
  - si l'utilisateur sélectionne F8 ou "Exécuter", l'événement suivant est appelé, dans ce cas, **STARTOF-SELECTION ;**
  - si l'utilisateur sélectionne toute autre fonction, l'écran de sélection se réaffiche

Événement AT SELECTION-SCREEN



Événement AT SELECTION-SCREEN

- Utilisez l'événement **AT SELECTION-SCREEN chaque fois que vous souhaitez programmer des** contrôles supplémentaires à un écran de sélection standard.
- Toute action utilisateur lance l'événement **AT SELECTION-SCREEN. Si un dialogue d'erreur se** déclenche, le système revient à l'écran de sélection et toutes les zones de saisie sont de nouveau accessibles. Un message s'affiche dans la ligne d'état.

Exemple : AT SELECTION-SCREEN

```
PARAMETERS: pa_car TYPE s_carr_id.


* Premier événement traité après avoir quitté l'écran de
AT SELECTION-SCREEN.
AT SELECTION-SCREEN.
  AUTHORITY-CHECK OBJECT 'S_CARRID'
    ID 'CARRID' FIELD pa_car
    ID 'ACTVT'  FIELD actvt_display.
  IF sy-subrc <> 0.
* Re MESSAGE e045(bc400) WITH pa_car. indiquer message dans
barre d'état
    MESSAGE e045(bc400) WITH pa_car.
  ENDIF.
```

Exemple : AT SELECTION-SCREEN

- Comme exemple de contrôle de saisie supplémentaire avec dialogue d'erreur, une zone de saisie pour l'identificateur de la compagnie aérienne doit être ajoutée au programme.
- Un contrôle d'autorisation est exécuté sur l'écran de sélection :
  - si l'utilisateur possède l'autorisation d'affichage pour la compagnie aérienne sélectionnée, le programme continue ;
  - si l'utilisateur ne possède pas l'autorisation d'affichage, l'écran de sélection se réaffiche et un message d'erreur apparaît dans la barre d'état.

Exercice

- Écrivez un programme qui permet à l'utilisateur de rentrer le code de compagnie aérienne mais ne permet l'affichage que si l'utilisateur rentre la valeur « DL » ou « MA »
- Sinon il affiche le message d'erreur de numéro 021 et d'ID BC400 stocké dans la table T100

L’événement AT LINE-SELECTION

- Dans des programmes exécutables, vous pouvez utiliser le bloc événement **AT LINE-SELECTION** pour créer des listes détaillées. Le système d'exécution ABAP :
- affiche la liste de base quand les événements appropriés sont traités (par exemple, après **STARTOF-SELECTION).**
- traite le bloc événement **AT LINE-SELECTION chaque fois que vous double-cliquez sur une** entrée.
- affiche les listes détaillées quand l'événement **AT LINE-SELECTION est traité et augmente la** valeur contenue dans **sy-lsind de un initialisé à 0**
- affiche la liste détaillée à partir du niveau précédent de la hiérarchie de liste (n-1) chaque fois que vous sélectionnez l'icône flèche verte à partir de la liste détaillée effective(n).

Syntaxe : une liste détaillée simple

```
REPORT sapbc400udd_secondary_list_a.

START-OF-SELECTION.
   WRITE: / text-001 COLOR col_heading,
          / 'sy-lsind',
             sy-lsind color 2.


AT LINE-SELECTION.
   WRITE: / text-002 COLOR col_heading.
   ULINE.
   WRITE: / 'sy-lsind',
             sy-lsind color 4.
```

Symboles de texte :

001	Listes de base
002	Listes détaillées

Exemple de listes détaillées

Liste détaillée avec interaction avec la base de donnée

Buffer de liste de base

Ligne				
...				
5	LH	0400 FRA	JFK	10:10:00
6	LH	0402 FRA	JFK	13:30:00
...	...			
11	SQ	0002 SIN	SFO	09:30:00

Ligne	wa_spfli-carrid	wa_spfli-connid
...	...	...
5	LH	0400
6	LH	0402
...	...	...
11	SQ	0002

```
REPORT sapbc400udd_example_2.
                    ...
START-OF-SELECTION.
   SELECT * FROM spfli INTO wa_spfli.
     WRITE:/ wa_spfli-carrid, wa_spfli-connid,
             wa_spfli-airpfrom, wa_spfli-airpto,
             wa_spfli-deptime.
     HIDE:   wa_spfli-carrid, wa_spfli-connid.
   ENDSELECT.
```

Zone HIDE

- Lorsque l'événement **AT LINE-SELECTION est traité, les objets de données d'un programme** contiennent les mêmes valeurs que celles affichées auparavant dans la liste de base.
- Cependant, une liste détaillée nécessite souvent des données sélectionnées dans la liste de base. Vous pouvez utiliser la zone **HIDE pour stocker certaines données de la ligne que vous avez sélectionnée** et les insérer où les utilisées pour faire d'autres traitements.
- Pour cela, utilisez le mot-clé ABAP **HIDE, suivi de la liste des objets de données voulus. Le système** mémorise automatiquement le nom et le contenu de l'objet de données, selon la position de sa ligne dans la liste.

Exemple 1 d'utilisation de la zone hide

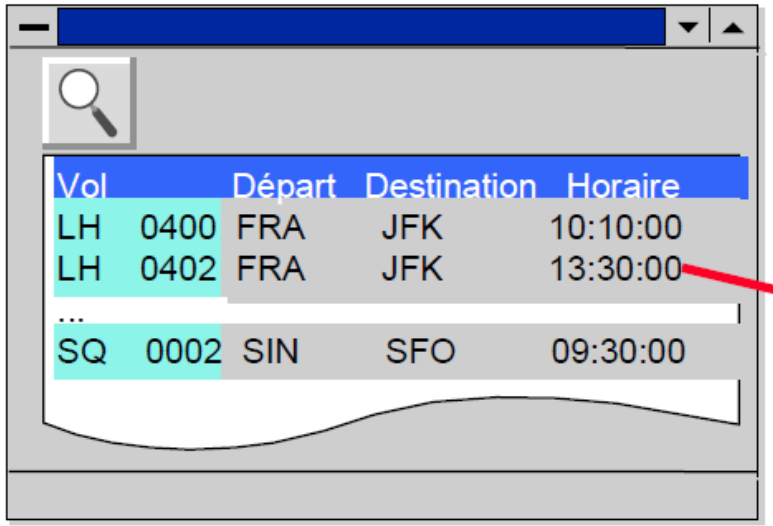
Buffer de liste de base

Ligne				
...				
5	LH	0400 FRA	JFK	10:10:00
6	LH	0402 FRA	JFK	13:30:00
...	...			
11	SQ	0002 SIN	SFO	09:30:00

Ligne	wa_spfli-carrid	wa_spfli-connid
...	...	...
5	LH	0400
6	LH	0402
...	...	...
11	SQ	0002

```
REPORT sapbc400udd_example_2.
                    ...
START-OF-SELECTION.
   SELECT * FROM spfli INTO wa_spfli.
     WRITE:/ wa_spfli-carrid, wa_spfli-connid,
             wa_spfli-airpfrom, wa_spfli-airpto,
             wa_spfli-deptime.
     HIDE:   wa_spfli-carrid, wa_spfli-connid.
   ENDSELECT.
```

Exemple 1 d'utilisation de la zone hide



Zone HIDE

Ligne	wa_spfli-carrid	wa_spfli-connid
...	...	...
5	LH	0400
6	LH	0402
...	...	...
11	SQ	0002

??	LH	0402	??	??	??	wa_spfli

```
REPORT sapbc400udd_example_2.

AT LINE-SELECTION.
     WRITE: text-001,
            wa_spfli-carrid,
            wa_spfli-connid.
```

Symboles de texte :

001	Correspondances

Exemple 2 d'utilisation de la zone hide

```
REPORT sapbc400udd_example_2.
...

AT LINE-SELECTION.
 IF sy-lsind = 1.
  WRITE: text-001,
            wa_spfli-carrid,
            wa_spfli-connid.

  SELECT fldate seatsmax seatsocc
    FROM sflight
    INTO CORRESPONDING FIELDS OF wa_sflight
    WHERE carrid = wa_spfli-carrid
    AND  connid = wa_spfli-connid.

      WRITE:/ wa_sflight-fldate,
              wa_sflight-seatsmax,
              wa_sflight-seatsocc.
  ENDSELECT.
```

Symboles de texte :

001	Correspondances

Exercice

- Écrivez un programme qui affiche tous les vols d'une compagnie aérienne donné par l'utilisateur
- Puis si l'utilisateur choisi un vol donné le programme lui affiche tous les détails du vol sélectionné y compris les date du vol sélectionné, le prix, la capacité maximal dans la classe économique….

Transp. Table: SPFLI Active
Short Description: Flight schedule

Attributes | Delivery and Maintenance | Fields | Entry help/check | Currency/Quantity Fields

Srch Help | Predefined Type

Field	Key	Ini...	Data element	Data Type	Length	Deci...	Short Description
MANDT	☑	☑	S_MANDT	CLNT	3	0	Client
CARRID	☑	☑	S_CARR_ID	CHAR	3	0	Airline Code
CONNID	☑	☑	S_CONN_ID	NUMC	4	0	Flight Connection Number
COUNTRYFR	☐	☐	LAND1	CHAR	3	0	Country Key
CITYFROM	☐	☑	S_FROM_CIT	CHAR	20	0	Departure city
AIRPFROM	☐	☑	S_FROMAIRP	CHAR	3	0	Departure airport
COUNTRYTO	☐	☐	LAND1	CHAR	3	0	Country Key
CITYTO	☐	☑	S_TO_CITY	CHAR	20	0	Arrival city
AIRPTO	☐	☑	S_TOAIRP	CHAR	3	0	Destination airport
FLTIME	☐	☑	S_FLTIME	INT4	10	0	Flight time
DEPTIME	☐	☑	S_DEP_TIME	TIMS	6	0	Departure time
ARRTIME	☐	☑	S_ARR_TIME	TIMS	6	0	Arrival time
DISTANCE	☐	☑	S_DISTANCE	QUAN	9	4	Distance
DISTID	☐	☑	S_DISTID	UNIT	3	0	Mass unit of distance (kms, miles)
FLTYPE	☐	☑	S_FLTYPE	CHAR	1	0	Flight type
PERIOD	☐	☑	S_PERIOD	INT1	3	0	Arrival n day(s) later

Transp. Table: SFLIGHT Active
Short Description: Flight

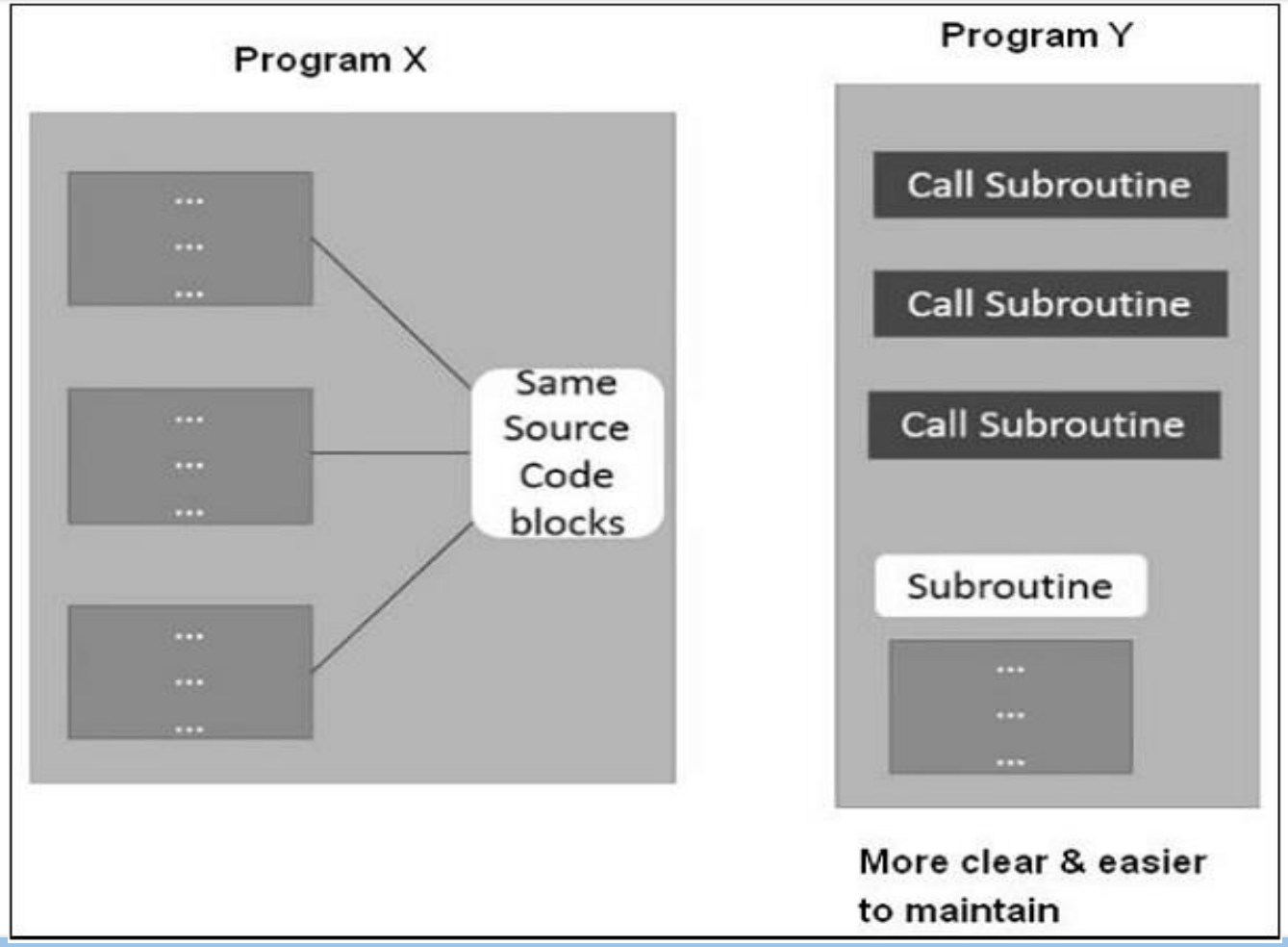
Attributes | Delivery and Maintenance | Fields | Entry help/check | Currency/Quantity Fields

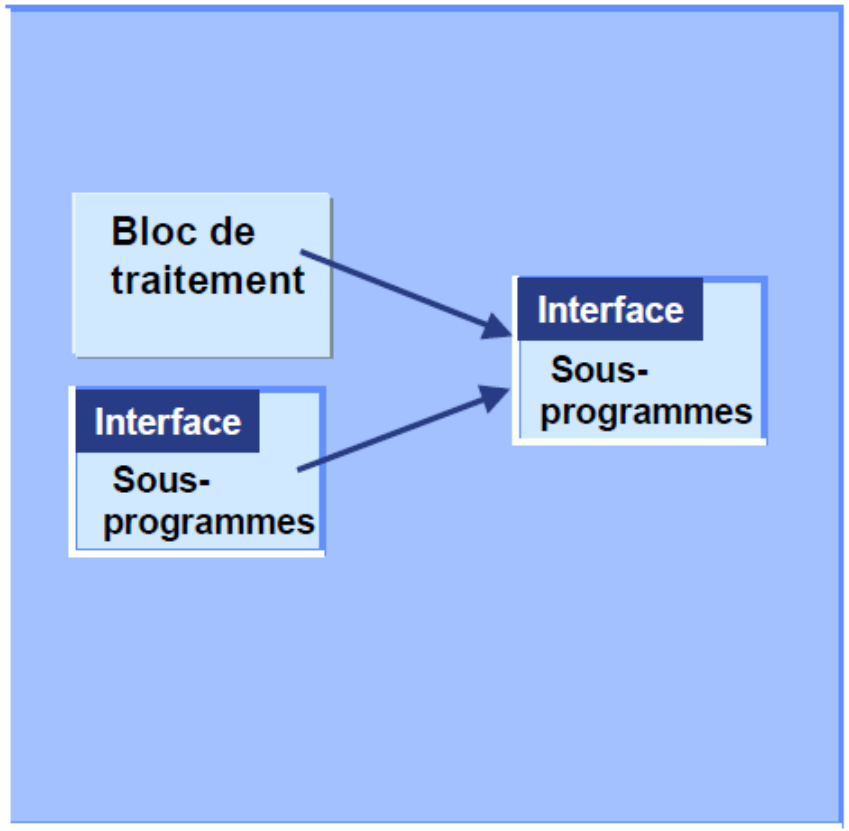
Srch Help | Predefined Type

Field	Key	Ini...	Data element	Data Type	Length	Deci...	Short Description
MANDT	☑	☑	S_MANDT	CLNT	3	0	Client
CARRID	☑	☑	S_CARR_ID	CHAR	3	0	Airline Code
CONNID	☑	☑	S_CONN_ID	NUMC	4	0	Flight Connection Number
FLDATE	☑	☑	S_DATE	DATS	8	0	Flight date
PRICE	☐	☑	S_PRICE	CURR	15	2	Airfare
CURRENCY	☐	☑	S_CURRCODE	CUKY	5	0	Local currency of airline
PLANETYPE	☐	☑	S_PLANETYE	CHAR	10	0	Aircraft Type
SEATSMAX	☐	☑	S_SEATSMAX	INT4	10	0	Maximum capacity in economy class
SEATSOCC	☐	☑	S_SEATSOCC	INT4	10	0	Occupied seats in economy class
PAYMENTSUM	☐	☑	S_SUM	CURR	17	2	Total of current bookings
SEATSMAX_B	☐	☑	S_SMAX_B	INT4	10	0	Maximum capacity in business class
SEATSOCC_B	☐	☑	S_SOCC_B	INT4	10	0	Occupied seats in business class
SEATSMAX_F	☐	☑	S_SMAX_F	INT4	10	0	Maximum capacity in first class
SEATSOCC_F	☐	☑	S_SOCC_F	INT4	10	0	Occupied seats in first class

Les sous programmes

- Afin d'éliminer les redondances qui peuvent exister dans un seul programme nous créons des sous programmes





- Fonctions encapsulées
- Programmes mieux structurés qui peuvent être lus et compris plus facilement
- Gestion de programmes plus facile
- Peut être réutilisé de manière locale dans le programme
- Bloc de traitement avec interface

Les sous programmes

- Pour appeler un sous programme nous utilisons

PERFORM subroutine_name [parameters] .

- Pour créer un sous programme nous utilisons

FORM subroutine_name [parameters] .
ENDFORM.

Les sous programmes

```
PERFORM sub_display.
WRITE:/ 'After Perform'.
*&---------------------* *& Form sub_display *&----------------*
FORM sub_display.
WRITE:/ 'Inside Subroutine'.
ENDFORM. " sub_display
```

```
ABAP Demo Program
-----------------------------------

Inside Subroutine
After Perform
```

Sous programme où sous routine

- Les sous-programmes peuvent appeler d'autres sous-routines et peuvent également appeler eux-mêmes. Une fois le sous-programme terminé, le contrôle retourne à l'instruction suivante après l'instruction PERFORM.

- Nous pouvons terminer un sous-programme en utilisant l'instruction EXIT ou CHECK.

- L'instruction EXIT peut être utilisée pour terminer un sous-programme inconditionnellement. Le contrôle retourne à l'instruction suivante après l'instruction PERFORM.

Exemples

```
PERFORM sub_display.
WRITE:/ 'After Perform Statement'.

*&---------------------------------------------------------------------*
*&      Form  sub_display
*&---------------------------------------------------------------------*
FORM sub_display.
  WRITE:/ 'Before Exit Statement'.
  EXIT.
  WRITE:/ 'After Exit Statement'.  " This will not be executed
ENDFORM.                    " sub_display
```

```
ABAP Demo Program

Before Exit Statement
After Perform Statement
```

```
DATA: flag TYPE c.
DO 2 TIMES.
  PERFORM sub_display.
ENDDO.
WRITE:/ 'After Perform Statement'.

*&---------------------------------------------------------------------*
*&      Form  sub_display
*&---------------------------------------------------------------------*
FORM sub_display.
  WRITE:/ 'Before Check Statement'.

  CHECK flag NE 'X'.
  WRITE:/ 'Check Passed'.
  flag = 'X'.
ENDFORM.                    " sub_display
```

```
ABAP Demo Program

Before Check Statement
Check Passed
Before Check Statement
After Perform Statement
```

Sous programme syntaxe générale

```
FORM subr
[USING   p1 TYPE type
                        p2 LIKE field
                                ...
                        VALUE(p3) TYPE type
                        VALUE(p4) LIKE field ...  ]
 [CHANGING {
            {VALUE(p1) TYPE type}|
            {p1 TYPE type}|{p1 LIKE field}  ...   } ]
 ...
ENDFORM.
```

Exemple de syntaxe : interface du sous-programme

```
*&---------------------------------------------------------------------*
*&      Form  FILL_ITAB
*&---------------------------------------------------------------------*
*       Filling internal table with records of  sflight  with
*       carrid = p_carrid, calculate percentage
*----------------------------------------------------------------------*
*    -->P_carrid     carrier id
*    <--P_IT_FLIGHT  internal standard table with line type sbc400focc
*----------------------------------------------------------------------*
FORM fill_itab USING    value( p_carrid) TYPE s_carr_id
               CHANGING p_it_flight      TYPE sbc400_t_sbc400focc.


 ...

ENDFORM.                                " FILL_ITAB
```

Les sous-programmes

- Vous devez déterminer le périmètre fonctionnel d'un sous-programme et examiner son nom, son interface et les commentaires. Si le sous-programme contient la fonctionnalité requise, vous avez ensuite besoin des informations suivantes pour être en mesure d'appeler le sous-programme :
- **nom du sous-programme**
- **les paramètres d'interface auxquels il a accès (uniquement en mode lecture) : les paramètres** sont répertoriés après l'option **USING. Le type et l'ordre des paramètres d'interface sont** importants.
- **les paramètres d'interface qu'il modifie : les paramètres sont répertoriés après l'option CHANGING. Le type et l'ordre des paramètres d'interface sont importants.**

Les sous-programmes : paramètres

- Les paramètres de l'interface sont dits "**formels" ; les paramètres transmis au sous-programme sont** dits "**effectifs".**
- Le nombre de paramètres effectifs doit être identique à celui des paramètres formels
- Lorsque vous appelez un sous-programme par **PERFORM, le système vérifie si le type des** paramètres effectifs de l'instruction **PERFORM est compatible avec celui des paramètres formels.** Différents types de contrôles sont exécutés pour différents types.
- **Contrôle complet du type :**
  - **TYPE D, F, I, T ou <type dictionnaire>. Ces types sont entièrement spécifiés. Le s**ystème contrôle si le type de données du paramètre effectif est identique à celui du paramètre formel dans son intégralité.
- **Contrôle partiel par types génériques :**
  - **TYPE C, N, P ou X : le système contrôle si le paramètre effectif est de type C, N, P ou X. La longueur du paramètre et le nombre de décimales dans l'option DECIMALS (type P) sont** transférés du paramètre effectif vers le paramètre formel.
  - **TYPE <type générique du dictionnaire > : le paramètre formel hérite de toutes** les informations non spécifiées concernant un type générique du Dictionnaire, depuis un paramètre formel.

Exemple de syntaxe : appel des sous-programmes

```
PARAMETERS pa_car TYPE s_carr_id.
DATA: it_flight TYPE sbc400_t_sbc400focc.
      :
      :
PERFORM fill_itab USING     pa_car
                  CHANGING it_flight .
      :
      :
```

Paramètre de saisie

Paramètres formels

```
FORM fill_itab
     USING     VALUE(p_carrid)   TYPE s_carr_id
     CHANGING  p_it_flight       TYPE sbc400_t_sbc400focc.

DATA ls_flight TYPE sbc400focc. " local structure

SELECT carrid connid fldate seatsmax seatsocc FROM sflight
       INTO CORRESPONDING FIELDS OF TABLE p_it_flight
       WHERE carrid = p_carrid.
...
ENDFORM.
```

Une erreur de syntaxe s'affiche si les types de paramètres effectifs ne sont pas compatibles avec la manière dont le paramètre formel est typé

USING et CHANGING

- Lorsque vous utilisez USING la valeur que vous assignez comme une valeur Formal n'effectuera pas la valeur réelle.
- Mais lorsque nous utilisons CHANGEMENT - La valeur Formel va écraser la valeur réelle.
- La valeur formelle est la valeur que vous assignez dans le sous-programme.
- Il crée la copie du VARIABLE.
- La valeur réelle est la valeur que vous affectez en dehors du sous-programme.

Passage de paramètres par référence

```
REPORT form_tech_param_reference.

DATA: num1 TYPE i,
 num2 TYPE i,
 sum TYPE i.
num1 = 2. num2 = 4.
PERFORM addit USING num1 num2 CHANGING sum.
num1 = 7. num2 = 11.
PERFORM addit USING num1 num2 CHANGING sum.

FORM addit
  USING add_num1  TYPE any
  add_num2  TYPE any
  CHANGING add_sum TYPE any.

                  add_sum = add_num1 + add_num2.
                 PERFORM out USING add_num1 add_num2 add_sum.

ENDFORM.

FORM out
  USING out_num1 TYPE any
  out_num2 TYPE any
  out_sum TYPE any.

                  WRITE: / 'la somme de ', out_num1, 'et', out_num2, 'est', out_sum.

ENDFORM.
```

Somme de 2 et 4 est 6
Somme de of 7 et 11 est 18

Passage de paramètres par référence et par valeur

```
REPORT demo_mod_tech_example_2.

DATA: num TYPE i VALUE 5,
 fac TYPE i VALUE 0.

PERFORM fact USING num CHANGING fac.

        WRITE: / 'le factoriel de ', num, 'est', fac.

FORM fact
  USING value(f_num) TYPE i
  CHANGING f_fact TYPE i.

        f_fact = 1.
        WHILE f_num GE 1.
                f_fact = f_fact * f_num.
               f_num = f_num - 1.
         ENDWHILE.

ENDFORM.
```

```
le factoriel de  5 est 120
```

Passage de paramètres par valeur

```
REPORT demo_mod_tech_example_3.
DATA: op1 TYPE i,
 op2 TYPE i,
 res TYPE i.
op1 = 3.
op2 = 4.

PERFORM multip USING op1 op2 CHANGING res.

WRITE: / ‘après le sous programme:',
  / 'RES=', res.

FORM multip
  USING value(op1) TYPE any
  value(op2) TYPE any
  CHANGING value(r) TYPE any.
                r = op1 * op2.
                 WRITE: / ‘Dans le sous programme:',
                / 'R=', r, 'RES=', res.
ENDFORM.
```

R= 12 RES= 0
après le sous programme :
RES= 12

Appel par valeur et par référence

- **Appel par référence : l'adresse du paramètre effectif est transmis. Dans le sous-programme, la** variable est adressée en utilisant le nom du paramètre formel. Les modifications ont un effet immédiat sur la variable globale. Si seul le nom du paramètre formel est spécifié dans l'interface du sous-programme, le paramètre est alors appelé par référence.
- **Appel par valeur : lorsque le sous-programme est appelé, une variable locale est créée avec le nom** du paramètre formel et la valeur du paramètre effectif est copiée vers le paramètre formel. Deux types d'appel par valeur existent :
  - **Appel par valeur : le paramètre formel est transmis dans l'interface après la clause USING** comprenant l'option **VALUE(<nom paramètre>). Lorsque le sous-programme est appelé, le** paramètre effectif est copié vers le paramètre formel. Les modifications apportées au paramètre formel affectent uniquement la copie locale, et non le paramètre effectif.
  - **Appel par valeur : le paramètre formel est transmis dans l'interface après la clause CHANGING** comprenant l'option **VALUE(<nom paramètre>). Lorsque le sous-programme est appelé, le** paramètre effectif est copié vers le paramètre formel. Les modifications initialement effectuées sur le paramètre formel affectent uniquement la copie locale. Lorsque l'instruction **ENDFORM est** atteinte, la valeur du paramètre formel est recopiée dans le paramètre effectif.

Exercices

- Construisez un type structure appelé « line » qui contient deux champs de type entier.
- Créer un subroutine qui retourne la puissance de deux des 7 premiers chiffres (1,2,3,…7) (sachant que n^2 en abap est calculé par n ** 2 et que sy-index donne l'indexe de l'itération commençant par 1))
- Créer un subroutine qui affiche le résultat retournée par le premier subroutine
- Appeler les deux subroutine dans un programme principale

Solution

```
REPORT  ZROUTINE.
DATA: BEGIN OF line,
  col1 TYPE i,
  col2 TYPE i,
END OF line.
DATA itab LIKE STANDARD TABLE OF line.

PERFORM fill CHANGING itab.
PERFORM out USING itab.


*&---------------------------------------------------------------------*
*&      Form  FILL
*&---------------------------------------------------------------------*
FORM fill CHANGING f_itab LIKE itab.
  DATA f_line LIKE LINE OF f_itab.
  DO 7 TIMES.
    f_line-col1 = sy-index.
    f_line-col2 = sy-index ** 2.
    APPEND f_line TO f_itab.
  ENDDO.
ENDFORM.                    "FILL


*&---------------------------------------------------------------------*
*&      Form  OUT
*&---------------------------------------------------------------------*
FORM out USING value(f_itab) LIKE itab.
  DATA f_line LIKE LINE OF f_itab.
  LOOP AT f_itab INTO f_line.
    WRITE: / f_line-col1, f_line-col2.
  ENDLOOP.
ENDFORM.                    "OUT
```