# ABAP Course

Chapter 3: Basic concepts

**Content**
The third chapter is about the basic concepts in ABAP programs. You will focus on data declaration as well as data manipulation. Furthermore you will create a program which uses user inputs and a function module. In the last section you will use a BAPI to use external functionality which is not included into your program.

**Prerequisites**
Before starting the exercises you should successfully proceed through chapter 2.

**Motivation**
This chapter explains the basic concepts of ABAP programs. It focuses on the structure of data, the data manipulation and the functions. Furthermore you will learn how to use external program and functions in your programs.

**Lecture notes**
The fundamental understanding of the ABAP development in the SAP system is a prerequisite for the students. Students do not use the developments from chapter 2, which means they can go on without touching chapter 2. Students can go on with their account from chapter 1.

- **Product**: All
- **Level**: Beginner
- **Focus**: Programming
- **Version**: 1.0
- **Author**: UCC Technische Universität München

**Task 1: Login into the SAP system**

**Short description**: Use SAPGui to login into the SAP system with your username and password

*Login*

Start the SAPGui and login into the development system using the provided account and password. Please refer to chapter 1 for your username and your password.

**Task 2: Data declaration and manipulation**

**Short description:** Use data declaration to declare a new data structure for your variables and use the debugger to get information about the variables during runtime

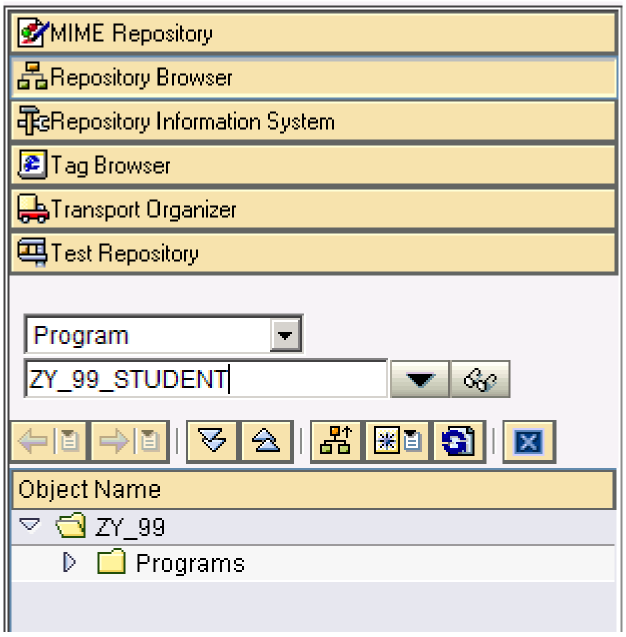
Please start the Object Navigator from the SAP Easy Access Menu by using the following path:

*Menu path*

**Tools • ABAP Workbench • Overview • Object Navigator**.

You may also use the transaction code **SE80** for direct access.

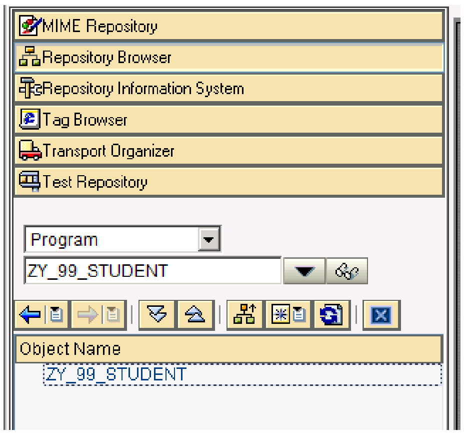
Create a new program which is named **ZY_##_STUDENT**. This can be done by selecting '**PROGRAM**' from the dropdown list, type in your program name and hit '**ENTER**':



*No TOP include*

The SAP system will ask you if you want to create the new program. Please answer yes to this question. Deselect the TOP include checkbox in the next pop-up. For the program attributes it is mandatory to select '**T Test Program**' as the program's status. In the next step the system asks you for a package. Please select the package you created in chapter 1, which is named ZY_##. The last step is the assignment to a transport request. Please use the transport request you created in chapter 2. The correct transport request should be in the input field already.

After you have answered the last question the new program is created and you have to open it by double clicking on the entry in the left section.



***str_student***

Now you create a new data structure called '**str_student**'. This is done by using the '**TYPE**' instruction. The new structure includes data about students such as name, family name and the student's id. Data like name and family name are character types **(c)** whereas the student's id is a numeric type **(n)**. You restrict the length of the name and family name to maximum 40 characters and the student's id should not exceed 10 numbers:

```
*&---------------------------------------------------------------------*
*& Report  ZY_99_STUDENT
*&
*&---------------------------------------------------------------------*
*&
*&
*&---------------------------------------------------------------------*

REPORT  zy_99_student.

TYPES: BEGIN OF str_student,
        name(40) TYPE c,
        family_name(40) TYPE c,
        students_id(10) TYPE n,
       END OF str_student.

DATA student TYPE str_student.
```

***Hint***

**Hint:**
Use the Pretty Printer to check for syntax errors very quickly and to use uppercase for key words.

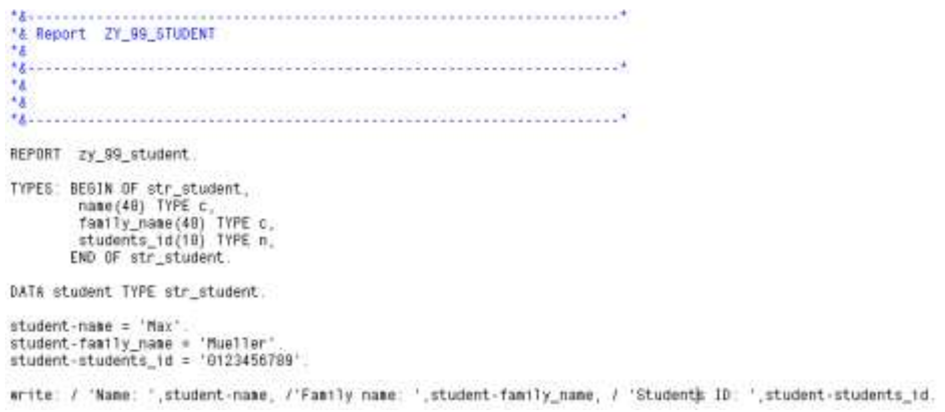
***Assign values***

In the next step you want to use the new structure to store some data inside. Therefore you have to assign values to the structure. This can be done by using the name of the structure and the field names. It is mandatory to use spaces between the structure's name and the value.

```
student-name = 'Max'.
student-family_name = 'Mueller'.
student-students_id = '0123456789'.
```

***WRITE /***

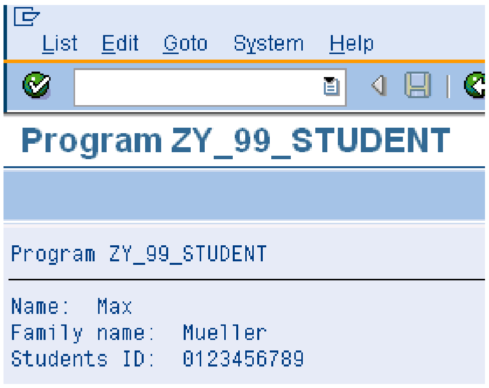
Finally you want to prove if the variable declaration was successful. Therefore we use the write instruction to write out the value of each structure field. This is done by using the '**WRITE /**' instruction. The slash at the end of the write instruction indicates that a new line should be opened after every output. So your complete source code should look similar to this:



```
*&---------------------------------------------------------------------*
*& Report  ZY_99_STUDENT
*&
*&---------------------------------------------------------------------*
*&
*&
*&---------------------------------------------------------------------*

REPORT  zy_99_student.

TYPES: BEGIN OF str_student,
         name(40) TYPE c,
         family_name(40) TYPE c,
         students_id(10) TYPE n,
       END OF str_student.

DATA student TYPE str_student.

student-name = 'Max'.
student-family_name = 'Mueller'.
student-students_id = '0123456789'.

write: / 'Name: ',student-name, /'Family name: ',student-family_name, / 'Students ID: ',student-students_id.
```

*Save and activate*

The result can be viewed after you saved and activated your program. It should look similar to this:



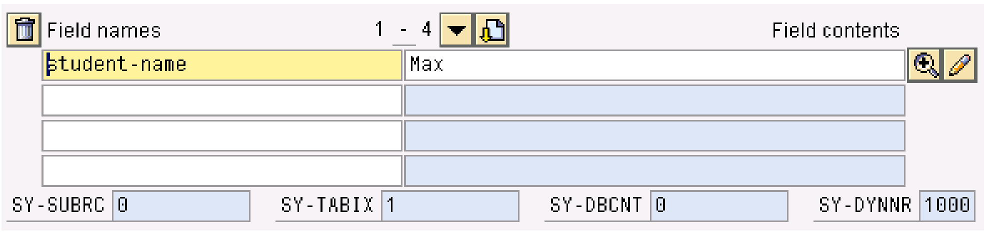
```
Program ZY_99_STUDENT

Program ZY_99_STUDENT

Name:  Max
Family name:  Mueller
Students ID:  0123456789
```

In chapter 2 we introduced the debugger. Now we want to use it to determine the variable values during runtime. So please start your program again using the debugger. You can do so by using the menu path:

*Start debugger*

**Program • Test • Debugging**

As soon as you start the debugger you will jump into the debugger screen and see the current instruction the debugger is working on. Use the '**Single step**' button (F5) to work on the first instructions. You will see that the debugger stops at the instruction `student-name = 'Max'`. This means the data structure as well as the variables were already created and now we assign values to the variable. Double click on the variable student-name and then use the '**Single step**' button again. After this step the debugger stopped at the next instruction. But now we can see the current value of the variable `student-name` in the lower section of the debugger.



| Field names | Field contents |
|---|---|
| student-name | Max |
| | |
| | |
| | |

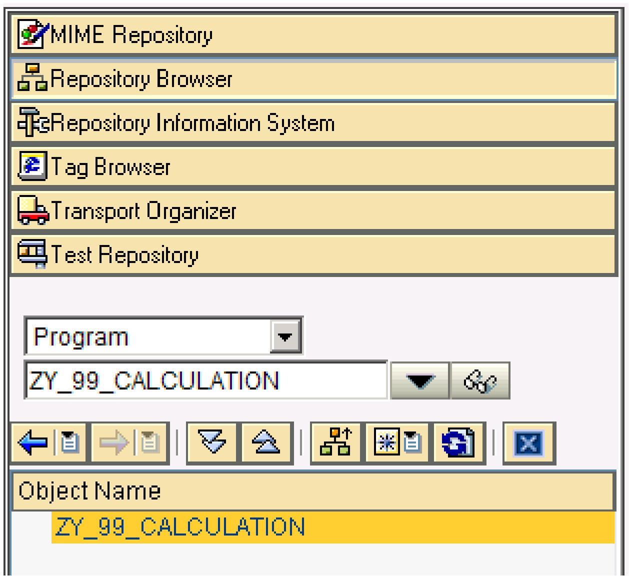
```
SY-SUBRC 0    SY-TABIX 1    SY-DBCNT 0    SY-DYNNR 1000
```

You may proceed now and check all the other variable values step-by-step.

**Task 3: User input and control structures**

**Short description:** Use selection screens and control structure to create a simple calculation program

Create a new program which is named **ZY_##_CALCULATION**. This can be done by selecting '**PROGRAM**' from the dropdown list, type in your program name and hit '**ENTER**':



The SAP system will ask you if you want to create the new program. Please answer yes to this question. Deselect the TOP include checkbox in the next pop-up. For the program attributes it is mandatory to select '**T Test Program**' as the program's status. In the next step the system asks you for a package. Please select the package you created in chapter 1, which is named **ZY_##**. The last step is the assignment to a transport request. Please use the transport request you created in chapter 2. The correct transport request should be in the input field already.

***No TOP include***

***PARAMETERS***

The function of the new program is the calculation of two operands by choosing an operand. Therefore you create a selection screen for the user input now. This is done by using the **PARAMETERS** instruction. This instruction is used to handle the user input. Both operands are integer (**i**) variables whereas the operator is char (**c**) type with the length of 1. Furthermore we need a variable for the result which is an integer type (**i**).

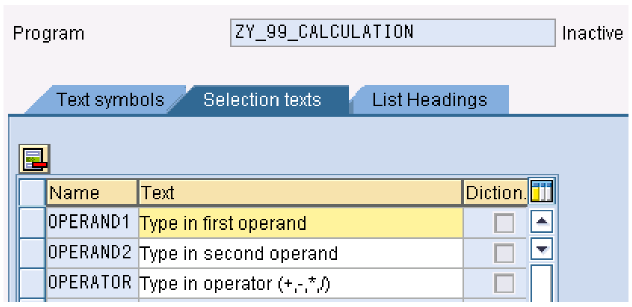
```
*&---------------------------------------------------------------------*
*& Report  ZY_99_CALCULATION
*&
*&---------------------------------------------------------------------*
*&
*&
*&---------------------------------------------------------------------*

REPORT  zy_99_calculation.

PARAMETERS: operand1 TYPE i, operand2 TYPE i, operator TYPE c.
DATA result TYPE i.
```

Please check, save and activate your program now. When testing your program you will see a little user interface which is generated automatically. The user can type in the operand1, operand1 and the operator. Besides the SAP system automatically generates the '**Execute**' button (F8). Unfortunately you have not defined any actions yet and this is why nothing happens when clicking on the '**Execute**' button.

***Changing screen names***

In the next step you change the screen names of operand1, operand2 and operator variables. Please use the following menu path to change the screen names:
**Goto • Text Elements • Selection Texts**

Now as we want to give the user a little help when using our program we change the screen names of our variables to more descriptive ones.



Program ZY_99_CALCULATION Inactive

| Name | Text | Diction. |
|---|---|---|
| OPERAND1 | Type in first operand | ☐ |
| OPERAND2 | Type in second operand | ☐ |
| OPERATOR | Type in operator (+,-,*,/) | ☐ |

After you have changed the screen names go back to your source code and save, activate and test your program again to see the results.

***CASE instruction***

In the last step we want to implement the calculation. Your program uses the operator to add, subtract, multiply or divide both operands - which calculation is used depends on the user input. In your program you will use a '**CASE**' instruction. Your source code should look similar to this:

```
*&---------------------------------------------------------------------*
*& Report  ZY_99_CALCULATION
*&
*&---------------------------------------------------------------------*
*&
*&
*&---------------------------------------------------------------------*

REPORT  zy_99_calculation.

PARAMETERS: operand1 TYPE i, operand2 TYPE i, operator TYPE c.
DATA result TYPE i.

CASE operator.
  WHEN '+'.
    result = operand1 + operand2.
  WHEN '-'.
    result = operand1 - operand2.
  WHEN '*'.
    result = operand1 * operand2.
  WHEN '/'.
    result = operand1 / operand2.
ENDCASE.

WRITE: operand1, ' ', operator, ' ', operand2, '=', result.
```

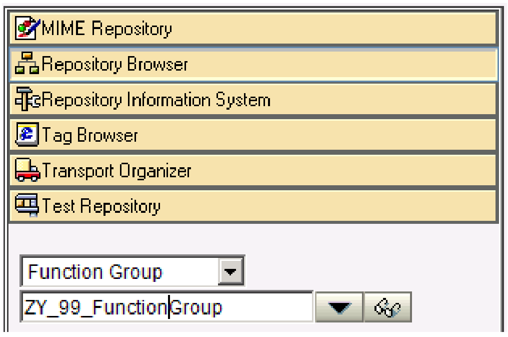
***Activate everything!***

Save, check and activate your new program. If the SAP systems shows you a user dialog when you try to activate the new program, please select all items in the dialog to be activated. This is because not only the new program but also the new selection texts have to be activated.

**Task 4: Calculation as a function module**

**Short description:** Use a function module to separate the calculation from the main program

In the previous task you calculated the result in the main program. In this task we want to separate the calculation from the main program and want to use a function module instead. Therefore create a new function group called '**ZY_##_FUNCTIONGROUP**'. This can be done easily from the dropdown menu choosing '**Function Group**' and typing in the name of your new function group.



Function Group

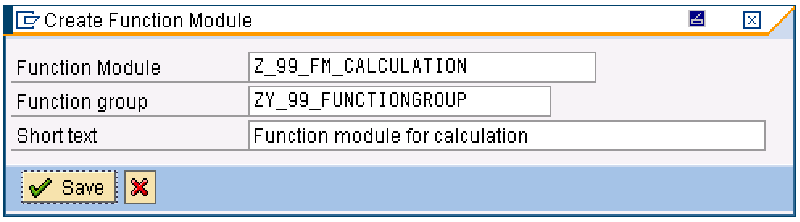
ZY_99_FunctionGroup

Maintain the short text, assign the function group to the package **ZY_##** and use the transport request which was also used when creating the package. In the next step you create a new function module by right clicking on your function group in the navigation tree and choosing the following menu path from the context menu:

**Create • Function Module**

Please create a new function module which is named **Z_##_FM_CALCULATION**. Note that the function module does not start with a ZZ_ but with a single Z_.

***Z_ instead of ZZ_!***

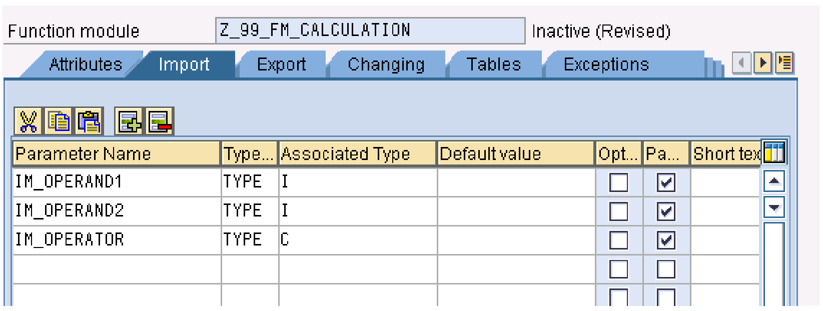


Create Function Module

| | |
|---|---|
| Function Module | Z_99_FM_CALCULATION |
| Function group | ZY_99_FUNCTIONGROUP |
| Short text | Function module for calculation |

The SAP system brings you directly to the '**Import'** tab of the new function module. Here you define all the parameters which should be imported by the function module. In our scenario we want to import three parameters: **IM_OPERAND1**, **IM_OPERAND2** and **IM_OPERATOR**. Besides you have to define the parameters data type and the pass value.

***Import***

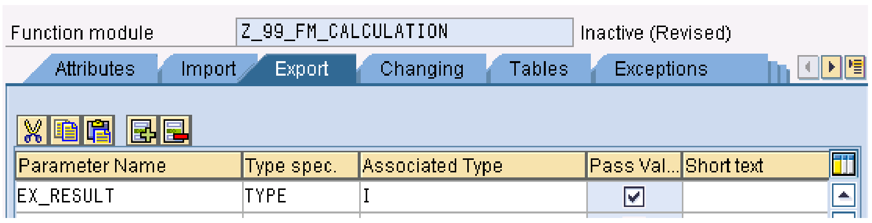


Function module Z_99_FM_CALCULATION Inactive (Revised)

| Parameter Name | Type... | Associated Type | Default value | Opt... | Pa... | Short tex |
|---|---|---|---|---|---|---|
| IM_OPERAND1 | TYPE | I | | ☐ | ☑ | |
| IM_OPERAND2 | TYPE | I | | ☐ | ☑ | |
| IM_OPERATOR | TYPE | C | | ☐ | ☑ | |
| | | | | ☐ | ☐ | |
| | | | | ☐ | ☐ | |

Now you have to switch to the '**Export'** tab. Here you have to define all variables to be exported by the function module. Please define the variable **EX_RESULT** to be exported.

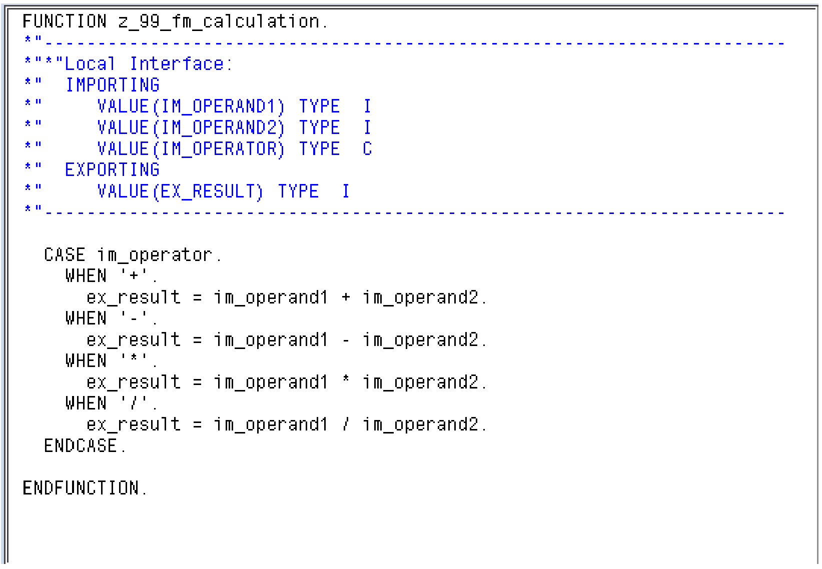
***Export***



Function module Z_99_FM_CALCULATION Inactive (Revised)

| Parameter Name | Type spec. | Associated Type | Pass Val... | Short text |
|---|---|---|---|---|
| EX_RESULT | TYPE | I | ☑ | |

Now switch to the tab '**Source code**' where the interface definition was already created by the system. If you change the interface definition here it will not effect the importing and exporting parameters as this is just text. In the source code of your function module you implement the new ‘**CASE**’ instruction whereas you have to replace the old parameter names (e.g. operand1) with the new ones (IM_OPERAND1). Your new source code should look similar to this:

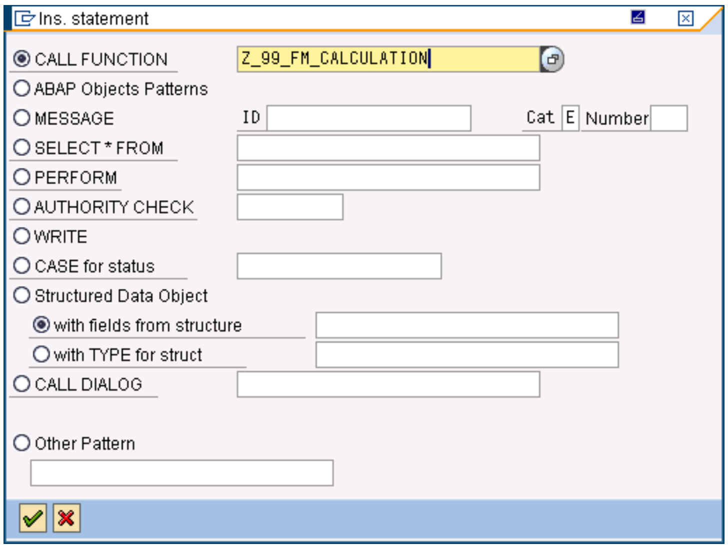


```
FUNCTION z_99_fm_calculation.
*"----------------------------------------------------------------------
*"*"Local Interface:
*"  IMPORTING
*"     VALUE(IM_OPERAND1) TYPE  I
*"     VALUE(IM_OPERAND2) TYPE  I
*"     VALUE(IM_OPERATOR) TYPE  C
*"  EXPORTING
*"     VALUE(EX_RESULT) TYPE  I
*"----------------------------------------------------------------------

  CASE im_operator.
    WHEN '+'.
      ex_result = im_operand1 + im_operand2.
    WHEN '-'.
      ex_result = im_operand1 - im_operand2.
    WHEN '*'.
      ex_result = im_operand1 * im_operand2.
    WHEN '/'.
      ex_result = im_operand1 / im_operand2.
  ENDCASE.

ENDFUNCTION.
```

Do not forget to save, check and activate the function module/group. You may test the new function module directly.

***Call Function***

In this step we want to replace the case instruction in your program by a function call. Obviously the function to be called is your new created function Z_##_FM_CALCULATION. Therefore switch back to your program **ZY_##_CALCULATION** by pressing the ‘**Back**’ button (F3). Delete the entire case instruction from your program and press the ‘**Pattern**’ button.



Ins. statement

(●) CALL FUNCTION   Z_99_FM_CALCULATION
( ) ABAP Objects Patterns
( ) MESSAGE   ID   Cat E   Number
( ) SELECT * FROM
( ) PERFORM
( ) AUTHORITY CHECK
( ) WRITE
( ) CASE for status
( ) Structured Data Object
  (●) with fields from structure
  ( ) with TYPE for struct
( ) CALL DIALOG

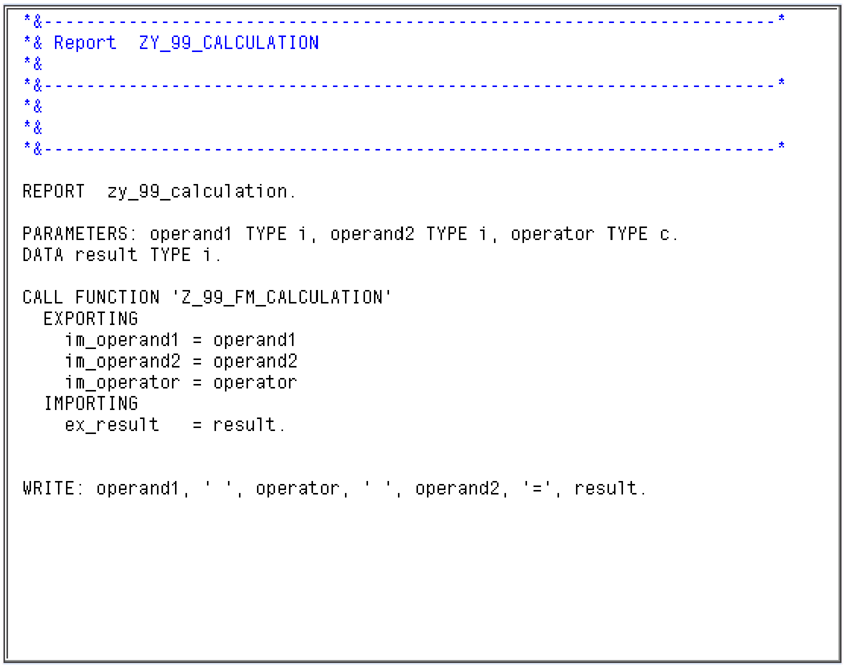
( ) Other Pattern

In the input field type in your function module name and then press ‘**Enter**’. After pressing ‘**ENTER**’ the system automatically inserts a new function call and a kind of skeleton. You see the importing parameters as well as the exporting parameters. All you have to do now is assign the variables to the importing and exporting parameters. Moreover, you have to uncomment the importing lines.

***Importing vs. Exporting***

**Hint:**
When you call a function inside your programs the exporting parameters always describe parameters, which are exported to the external function module. The importing parameters are parameters, which are imported to the current program.

Your new source code should look similar to this:



```
*&---------------------------------------------------------------------*
*& Report  ZY_99_CALCULATION
*&
*&---------------------------------------------------------------------*
*&
*&
*&---------------------------------------------------------------------*

REPORT  zy_99_calculation.

PARAMETERS: operand1 TYPE i, operand2 TYPE i, operator TYPE c.
DATA result TYPE i.

CALL FUNCTION 'Z_99_FM_CALCULATION'
  EXPORTING
    im_operand1 = operand1
    im_operand2 = operand2
    im_operator = operator
  IMPORTING
    ex_result   = result.


WRITE: operand1, ' ', operator, ' ', operand2, '=', result.
```

Save, check and activate your program. Now you can test the program with your new function call.