ABAP Course

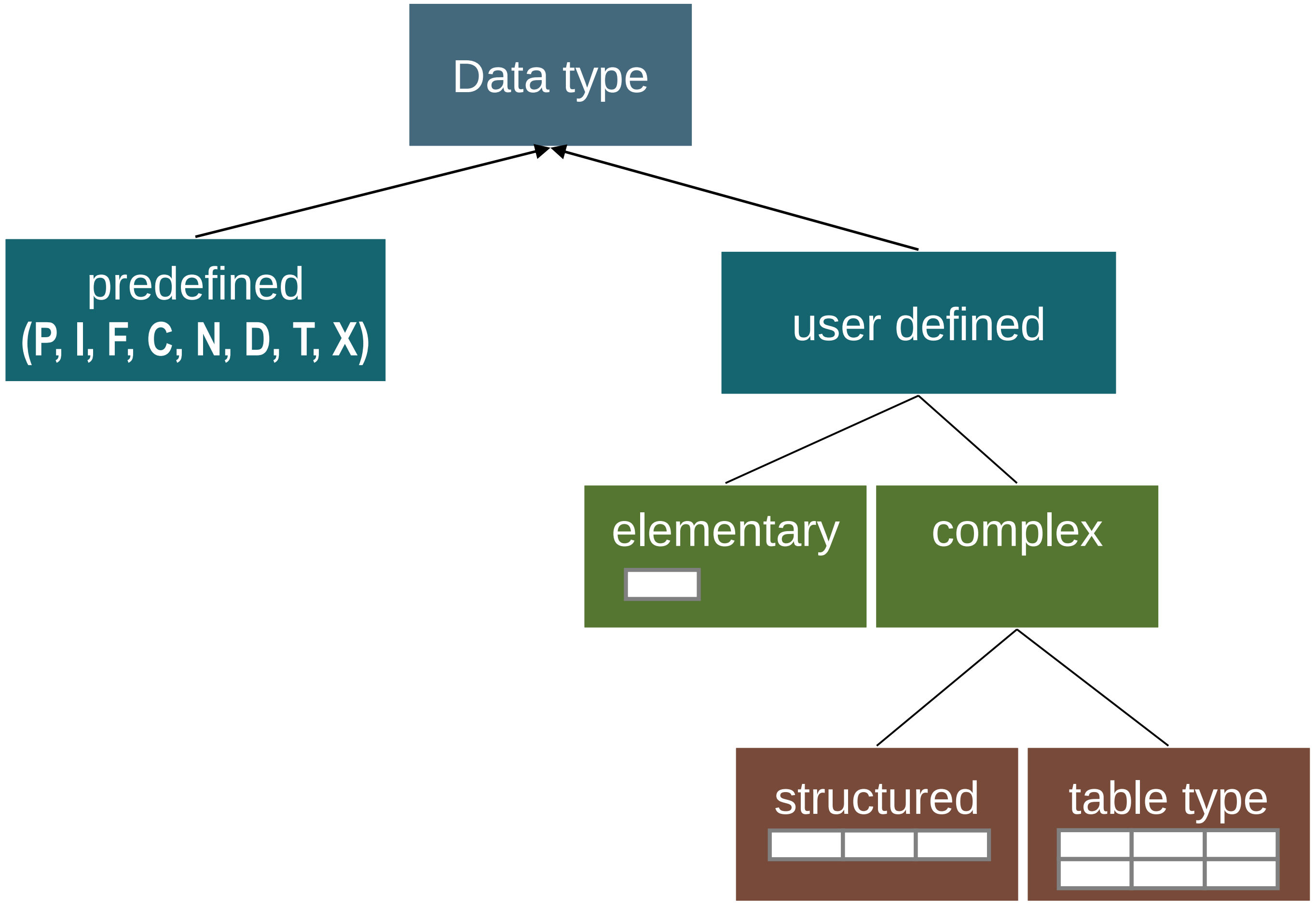
Chapter 3 – Basic concepts

Lecturer: Robert Meyer, UCC Technische Universität München
Authors: Marcus Homann, Valentin Nicolescu, André Bögelsack

Agenda

1. Data types and data declaration
2. Important instructions
3. Local modularization
4. Background processing

Data types



Source: Following SAP AG

Predefined data types in ABAP

Data type	Sense	Initial value	Values range
d	Date	00000000	
t	Time	000000	
i	Integer	0	
f	Float	0.00	
String	String		
Xstring	Byte		
p	Packed number	0	
n	Numerical text	00 … 0	Max. 65536 figures
c	Character	<SPACE>	Max. 65536 characters
x	Byte (hex)	X'00'	

Data declaration (1)

- **Elemental field definition:**

```
DATA l_var(len) TYPE <DATA TYPE>.
DATA l_var LIKE <DATA_OBJECT>.
```

- **Structured data object:**

```
DATA: BEGIN OF struc,
…
END OF struc.
```

- **Internal table:**

```
DATA l_tab TYPE <TABLE TYPE>. Or
DATA l_tab TYPE TABLE OF <STRUCTURE>.
```

- **Constants:**

```
CONSTANTS l_const TYPE <DATA TYPE> VALUE <value>.
```

Data declaration (2)

- **Parameters:**
  - Declaration of input elements
  - Syntax:

```
PARAMETERS <p>[(<length>)] [TYPE <type>|LIKE <obj>]
OBLIGATORY].
```

  - <p>: parameter name (maximal length 8)
  - OBLIGATORY: characterization as mandatory field
  - Specialized Examples: Checkbox and Radio Buttons:

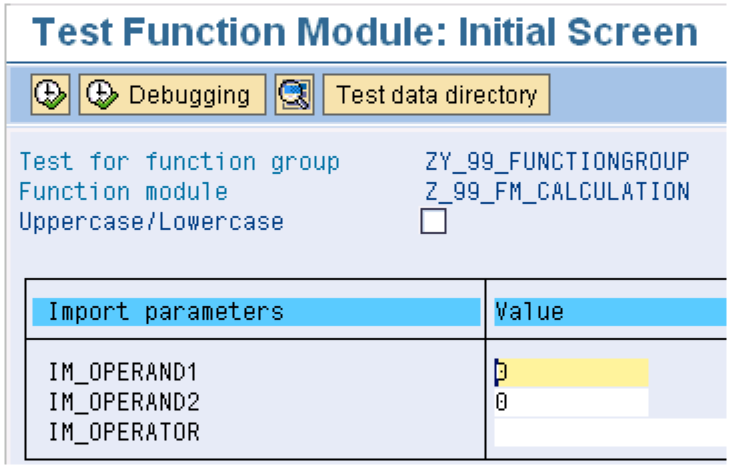
```
PARAMETERS: c1 AS CHECKBOX DEFAULT ’X’.
PARAMETERS:
 r1 RADIOBUTTON GROUP rad1 DEFAULT ’X’,
 r2 RADIOBUTTON GROUP rad1.
```

- **Object References:**

```
DATA l_object TYPE REF to <Class name>.
```

Selection screens

- Selection screens simplify interaction with user
- Selection screens always have Dynpro number 1000
- Selection screens are generated automatically when keyword `Parameters` is used in source code
- `Parameters` is also used for variable declaration



Test Function Module: Initial Screen

Test for function group	ZY_99_FUNCTIONGROUP
Function module	Z_99_FM_CALCULATION
Uppercase/Lowercase	☐

Import parameters	Value
IM_OPERAND1	0
IM_OPERAND2	0
IM_OPERATOR	

Definition of own data types

- Definition of completely new data types
- New elementary custom data types can be derived from existing ones:

```
TYPES text10 TYPE c LENGTH 10.
```

- Defining complex custom data types:

```
TYPES: BEGIN OF str_student,
name(40) TYPE c,
family_name(40) TYPE c,
id TYPE i,
END OF str_student.
```

- Declaring new structures:

```
DATA student TYPE str_student.
```

- Access to the structure by means of the hyphen (“-”) operator:

```
WRITE student-name.
```

System Values – Structure SY

- Structure **SY** contains many system variables from the SAP system
- Structure can be viewed in Data Dictionary (**SE11**) by entering data type SY

Field	Sense
sy-subrc	Return code of last instruction (0 = without errors)
sy-uname	Username of the current user
sy-host	Name of application server
sy-langu	Current system language
sy-dbsys	Name of database server
sy-tcode	Current transaction code
sy-index	Loop index
sy-client	Current client number

Convenient variable declaration

- Instead of defining every single data object by itself:

```
data a type c.
data b type i.
data c type c.
data d type i.
```

- Usage of *chain statements* is possible:

```
data: a type c, b type i, c type c, d type i.
```

Output with WRITE

- Syntax: **WRITE [/][<pos>][(<len>)] <text>.**
  - '/': Line Break; 'pos': column number; 'len': text length
- Simple Text output:

  WRITE 'Hello World'.
- Combination of Substrings and Output of Variable values:

  WRITE: 'Hello', sy-uname, /5 'Nice to see you here'.

```
Hello Max
    Nice to see you here
```

- ABAP creates no blank line by using multiple '**/**'
  - Change default setting with **SET BLANK LINES ON.**
  - Or use special command '**Skip <n>**'; n = number of blank lines

Data Manipulation and control structures

- Data manipulation
- Data object conversion
- Control structures
  - Loops
  - Branching based on conditions

Data manipulation

- Assign: `MOVE f TO g` or `g = f`
- Numeric: `ADD n TO m` or `m = m + n`
- String: `CONCATENATE, SPLIT, SEARCH, REPLACE, CONDENSE, TRANSLATE …`
- Logical:
  - For all data types: `=, <>, <, >, <=, >=`
  - For character like types: `CO (contains only), CN (contains not only), CA (contains any) …`

Control structures: branching (1)

- IF:

```
IF <logical expression>.
  <instruction 1>.
[ELSEIF <logical expression>.
  [<instruction 2>.
[ELSE.
  [<instruction 3>.
ENDIF.
```

- Example:

```
IF a > b.
  WRITE ‘a is bigger than b’.

ELSEIF a < b.
  WRITE ‘b is bigger than a’.
ELSE.
  WRITE ‘a equals b’.
ENDIF.
```

Control structures: branching (2)

- **CASE:**

```
CASE <variable name>.
  [WHEN <value 1>.
        [<instruction 1>.
  [WHEN <value 2>.
        [<instruction 2>.
  [WHEN OTHERS.
        [<instruction 3>.
ENDCASE.
```

- **Example**:

```
READ TABLE l_tab_customers INDEX 1 INTO
l_str_customer.
CASE sy-subrc.
  WHEN 0 or 2.
    WRITE: / ‘Entry found’.
  WHEN OTHERS.
    WRITE: / ‘Entry not found’.
ENDCASE.
```

Control structures: loops

- WHILE – ENDWHILE (conditional loop):

```
WHILE <logical expression>.
  <instructions>.
ENDWHILE.
```

- DO – ENDDO (count loop)

```
DO <n> TIMES.
  <instructions>
ENDDO.
```

- sy-index: returns the current loop index and refers to the current loop (in case of nested loops)

Datatype Conversion

- If it is possible to convert values from one data type to another, the SAP system does it automatically
- **Static incompatible**: between date and time
  - Is identified by compiler.
- **Dynamic incompatible**: between char ‘1234hello’ and integer
  - Is not identified by compiler → runtime error
- **Dynamic compatible**: between char ‘1234’ and integer 1234
- Exceptions can be caught by:

```
CATCH SYSTEM-EXCEPTION conversion_errors = 4.
…
ENDCATCH.
```

Modularization

- Modularization in ABAP:
  - Includes
  - FORMs (Procedures)
  - Function Groups / Function Modules
  - BAPIs

Modularization: Includes

- Outsource to external program
- The include statement is used in the main program to integrate an external program
- Instruction INCLUDE integrates external program code into the main program
- INCLUDE vs. TOP INCLUDE:
  - TOP INCLUDE also contains data declaration, which must be available in all selection screens

Local modularization: FORMs (1)

Subroutines in ABAP

- Declaration:

```
FORM <procedure name>
  USING [VALUE] <input parameter> TYPE <type>
  CHANGING [VALUE] <input/output parameter> TYPE <type>
ENDFORM.
```

- **USING** = Input Parameter
  - Formal parameter is <u>not</u> copied to actual parameter as function exist
- **CHANGING**: Output Parameter
  - Formal parameter is copied to actual parameter at function exit

- Call Forms:

```
PERFORM <procedure name> USING / CHANGING <parameter 1>
<parameter 2> <parameter n>.
```

Modularization: FORMs (2)

- ‘VALUE’ keyword:
  - ‘VALUE’ denoted -> call by Value
    - formal parameter has own memory → if it is changed, the actual parameter is not changed
  - ‘VALUE’ not denoted -> call by reference
    - Formal and actual parameter point to same memory
    - USING and CHANGING have same behavior with call by reference

Modularization: Function modules (1)

- Outsources functionality to external module
- Function modules are not allowed to access global variables → export variables when calling function module
- More than 100,000 function modules available
- Function modules are organized in function groups
- Function modules can be remote accessible
- Function groups may have own TOP include
- Declared with Function Builder (transaction **SE37**)

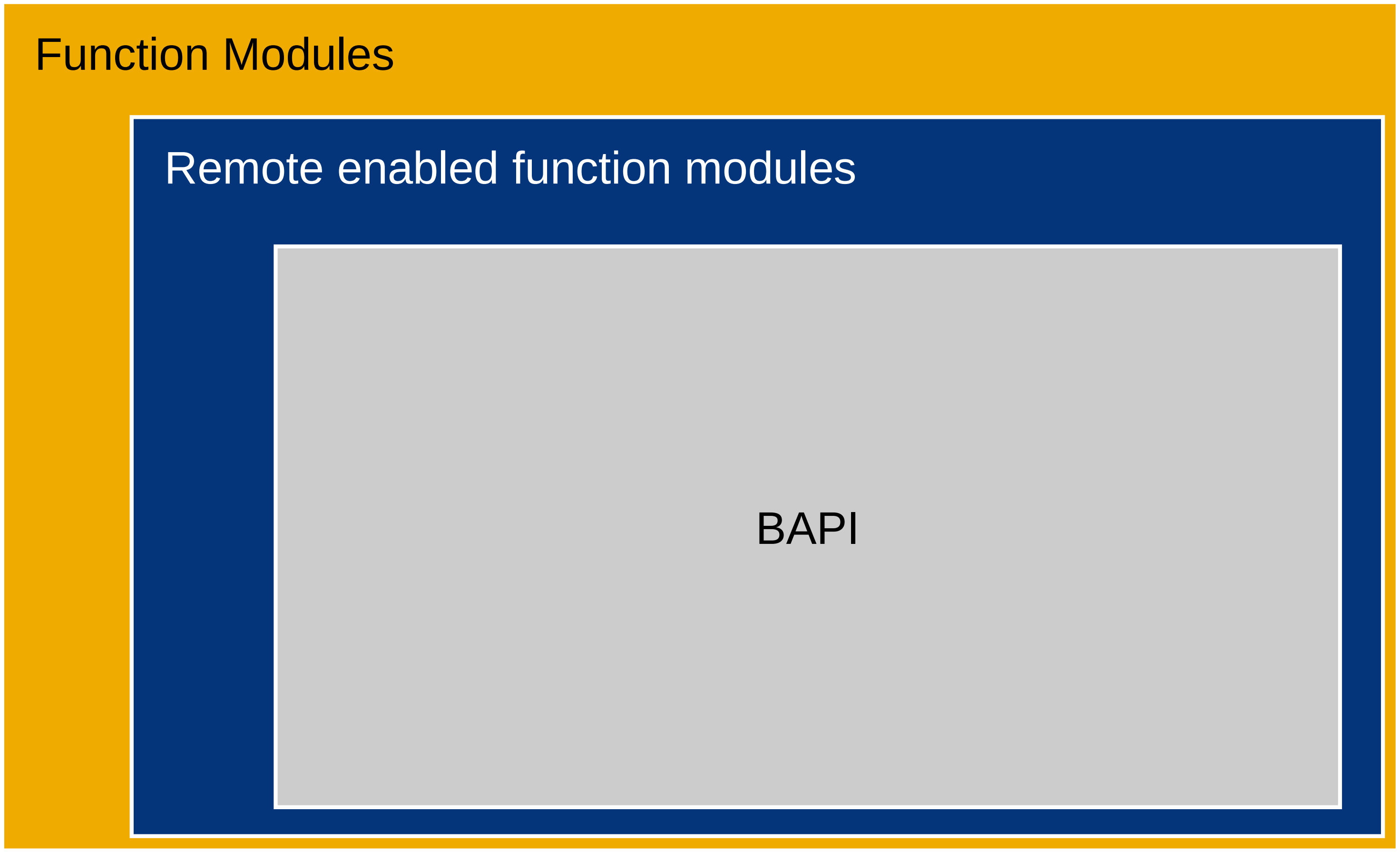
Modularization: Function modules (2)

- Call Syntax:

```
CALL Function ‘function name’
                   [EXPORTING par1 = var1]
                   [IMPORTING par2 = var2]
                   [CHANGING par3 = var3].
```

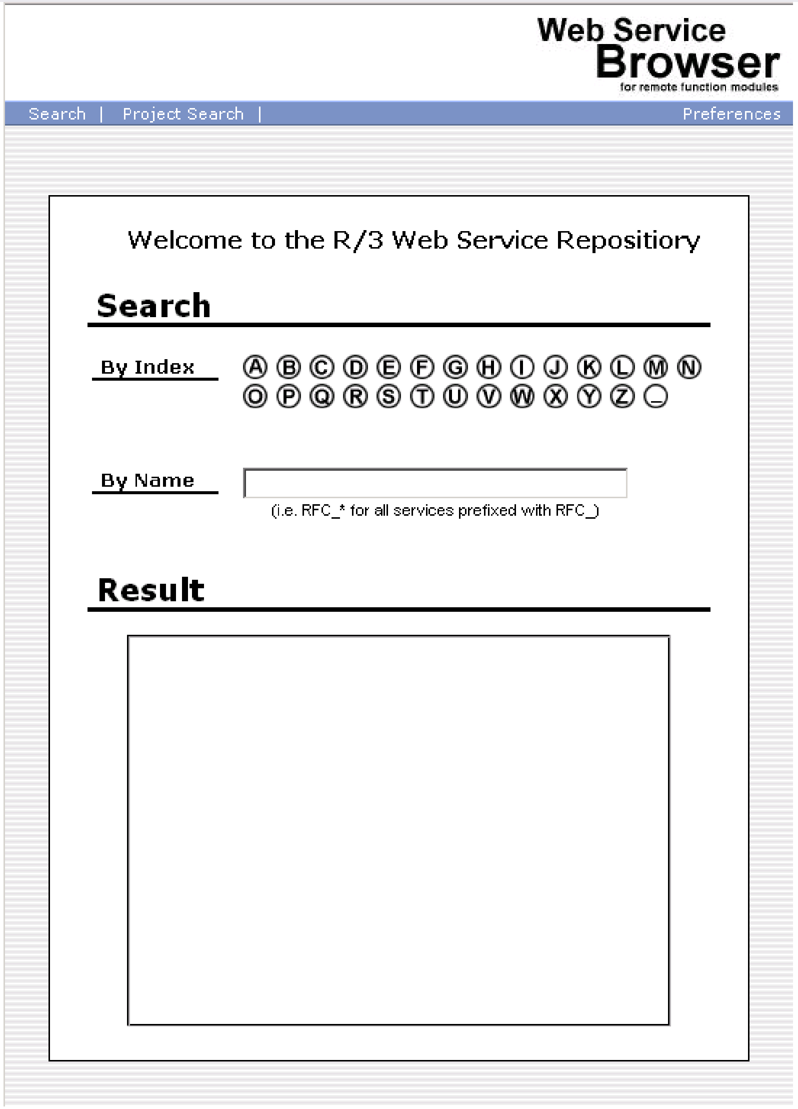
- IMPORTING = Input Parameter
- EXPORTING = Output Parameter
- CHANGING = Parameters that are changed during function execution

Modularization: Function modules (3)



Modularization: Function modules

- Introduction of Web Services as of WebAS 6.20
- Web service browser available under: http://<host>:<port>/sap/bc/bsp/sap/webservicebrowser/search.html
- <host> and <port> can be obtained from transaction **SMICM**
- <port> signifies the AS ABAP's ICM HTTP(S) port


Web Service
Browser
for remote function modules
Search | Project Search |
Preferences
By Index
A B C D E F G H I J K L M N
O P Q R S T U V W X Y Z _
By Name

Welcome to the R/3 Web Service Repository

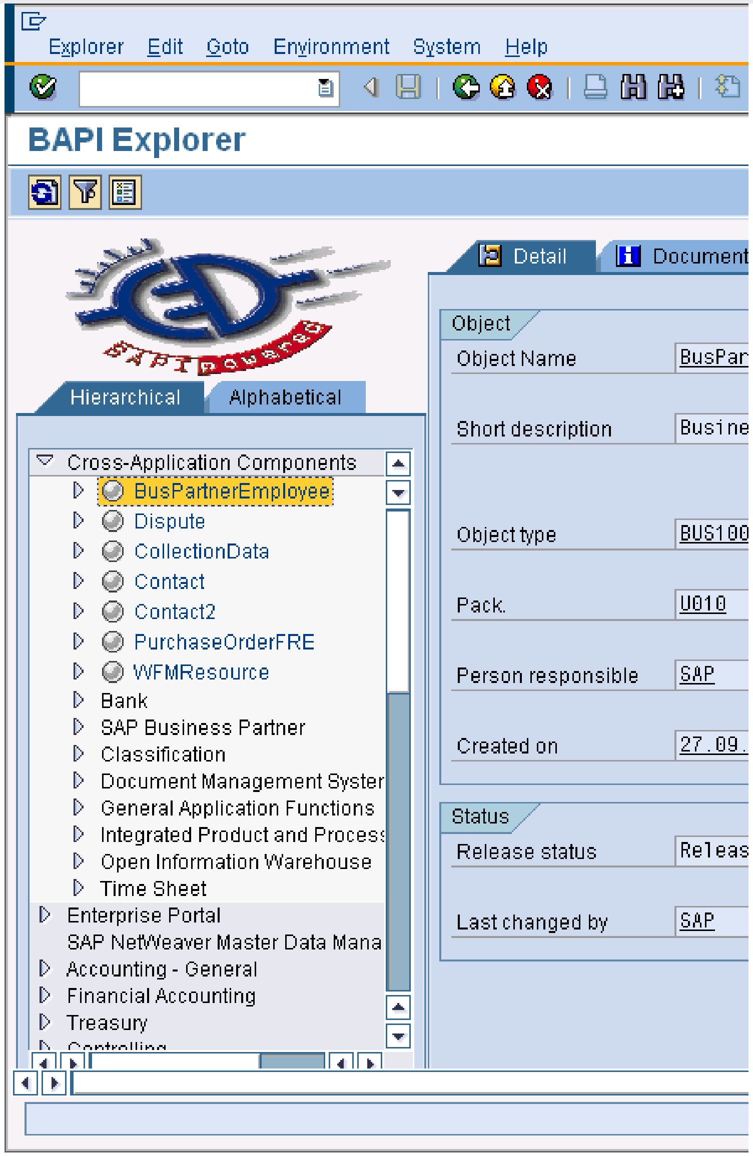
Search

(i.e. RFC_* for all services prefixed with RFC_)

Result

Modularization: BAPI's (1)

- BAPI = Business Application Programming Interface
- RFC enabled function modules
- Overview about all BAPI can be obtained from BAPI explorer (transaction **BAPI**)



Cross-Application Components
- BusPartnerEmployee
- Dispute
- CollectionData
- Contact
- Contact2
- PurchaseOrderFRE
- WFMResource
- Bank
- SAP Business Partner
- Classification
- Document Management Syster
- General Application Functions
- Integrated Product and Process
- Open Information Warehouse
- Time Sheet

Enterprise Portal
SAP NetWeaver Master Data Mana
Accounting - General
Financial Accounting
Treasury

Object

Object Name	BusPar
Short description	Busine
Object type	BUS100
Pack.	U010
Person responsible	SAP
Created on	27.09.

Status

Release status	Releas
Last changed by	SAP

Modularization: BAPI’s (2)

- Usage of BAPIs:
  - BAPIs provide you the functionality of a SAP transaction → be sure to be familiar with the SAP transaction
  - Search for the appropriate BAPI and read the documentation carefully
  - Test the BAPI using the Function Builder
  - Use the BAPI
- Possible problems:
  - Pay attention to the data types and mandatory data

Background processing

- Usual programs use dialog work processes
- Long running programs should always run in the background
- All ABAP programs can be scheduled as background jobs in transaction **SM36**
- For ABAP programs with a user interface you can predefine the user input by using variants