Basic Concepts

Adrian Streitz, Johannes Rank, Borys Levkovskyi

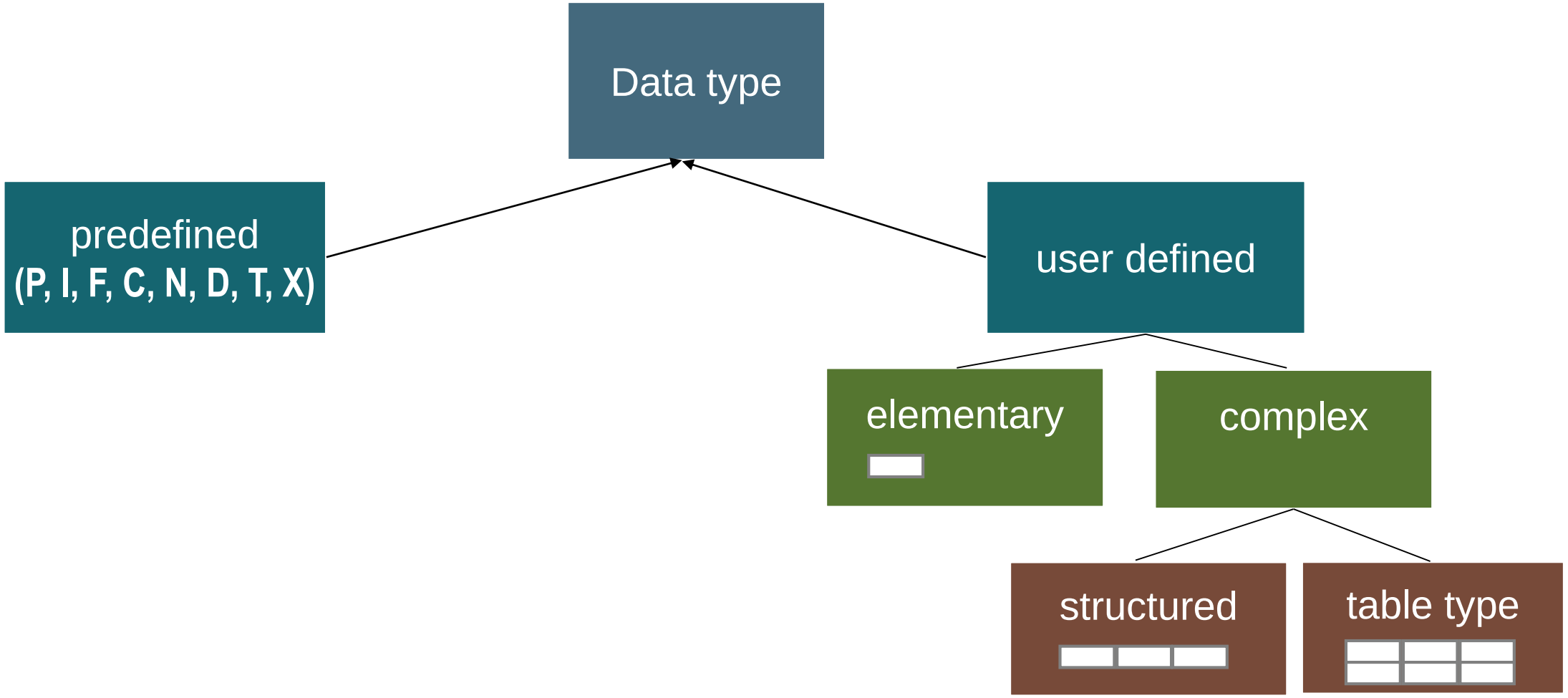
Prerequisites

- Experience with ABAP Workbench and Eclipse

Agenda

I. Data types and data declaration
II. Important instructions
III. Background processing
IV. Debugging

Data types



Source: Following SAP AG

Predefined data types in ABAP

Data type	Sense	Initial value	Possible values	Valid field length
d	Date	00000000	20161031 (31.10.2016)	Max. 8 characters
t	Time	000000	235900 (23:59 Uhr)	Max. 6 characters
i	Integer	0	987654321	Max. 4 bytes
f	Float	0	-1,860	Max. 8 bytes
String	String	„“	„Hello World!“	variable
Xstring	Byte	„“	Hexadecimal characters 0-9, A-F	variable
p	Packed number	0	1,43 (Length of p = 2)	Max. 16 bytes
n	Numerical text	00 … 0	Alphanumeric characters; Valid values are only the digits 0 to 9	Max. 65535 bytes
c	Character	<SPACE>	Alphanumeric characters	Max. 65535 bytes
x	Byte (hex)	X’0…0‘	Hexadecimal characters 0-9, A-F	Max. 65535 bytes

Data declaration

- **Elemental field definition:**

```
DATA f(len) TYPE <DATA TYPE>.
```

```
DATA name(20) TYPE c
```

- **Structured data object:**

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

```
DATA: BEGIN OF address,
  name   TYPE surname,
  street(30) TYPE c,
  city   TYPE spfli_type-cityfrom,
  END OF address.
```

- **Internal table:**

```
DATA itab TYPE <TABLE TYPE>.  or
DATA itab TYPE TABLE OF <STRUCTURE>.
```

```
DATA: lt_type TYPE TABLE OF int4
```

Data declaration

- **Constants:**

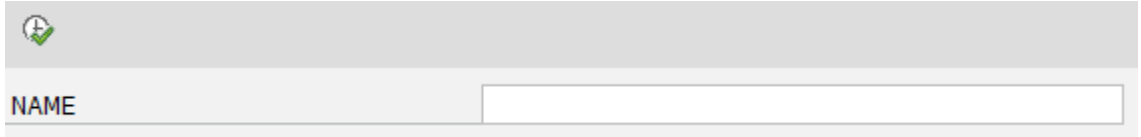
```
CONSTANTS: <C Name> TYPE <DATA TYPE>
    VALUE <value> / is INITIAL.
```

```
CONSTANTS: Org(5) TYPE C VALUE 'SAP'
```

- **Parameters:**

```
PARAMETERS: <P NAME> TYPE <DATA TYPE>.
```

```
PARAMETERS: name type string
```



Data declaration

- **Instead of defining every single data/constant/parameter:**

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

- **Use:**

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

Definition of own data types

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

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

- Definition of one's own data type:

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

Definition of own data types

- Declaration of a new structure:

```
DATA student TYPE str_student.
```

- Access to the structure:

```
WRITE student-name.
```

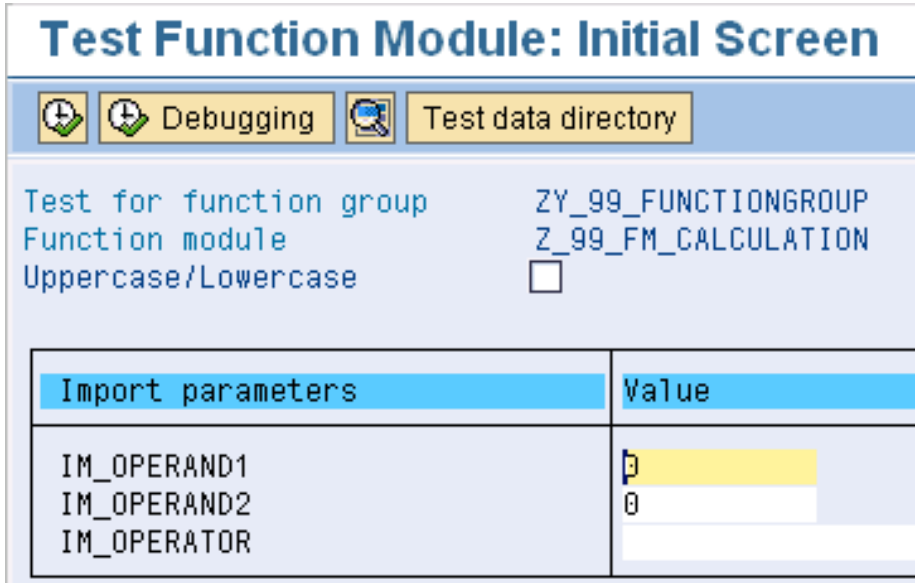
System Fields

- Structure SYST contains many system variables from the SAP system
- Structure can be viewed in Data Dictionary (SE11) via data type SYST

Field	Sense
Sy-subrc	Returncode of last instruction
Sy-date	Current date and time
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

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



Important instructions and control structures

- Data manipulation
- Data object conversion
- Control structures
  - Loops
  - Branching conditionally

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)` …
  - For byte like types: `BYTE-CO`, `BYTE-CN`, `BYTE-CA` …
  - For bit patterns: `O (Ones)`, `Z (Zeros)`, `M (Mixed)`

Data object conversion

- If it is possible to migrate values from one data type to another, the SAP system does it automatically
- Static incompatible: between date and time
- Dynamic incompatible: between char ‘1234hello’ and integer
- Dynamic compatible: between char ‘1234’ and integer 1234
- Exceptions can be caught

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

Control structures: loops

- WHILE – ENDWHILE:

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

- DO – ENDDO

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

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

Control structures: branching

- IF:

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

Control structures: branching

- CASE:

```
CASE <data object>.
      [WHEN <value 1>.
        [<instruction 1>.
      [WHEN <value 2>.
        [<instruction 2>.
      [WHEN OTHERS.
        [<instruction 3>.
ENDCASE.
```

Comments

- Commenting by the preceding sign ´*´:

```
*********************************
* PROGRAM ZTEST *
* WRITTEN BY SAP, 26.08.2015 *
* LAST CHANGED BY SAP, 26.08.2016 *
* TASK: PERFORMING TESTS *
*********************************
```

only valid if at the beggining of the line

use **CTRL + 7** in Eclipse to comment automatically

- Commenting by ‘ “ ‘ at *any position* in the line:

```
write: 'TEST'. "Comment
```

Background processing vs. Foreground 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 TA SM36
- For ABAP programs with a user interface you can predefine the user input by using variants

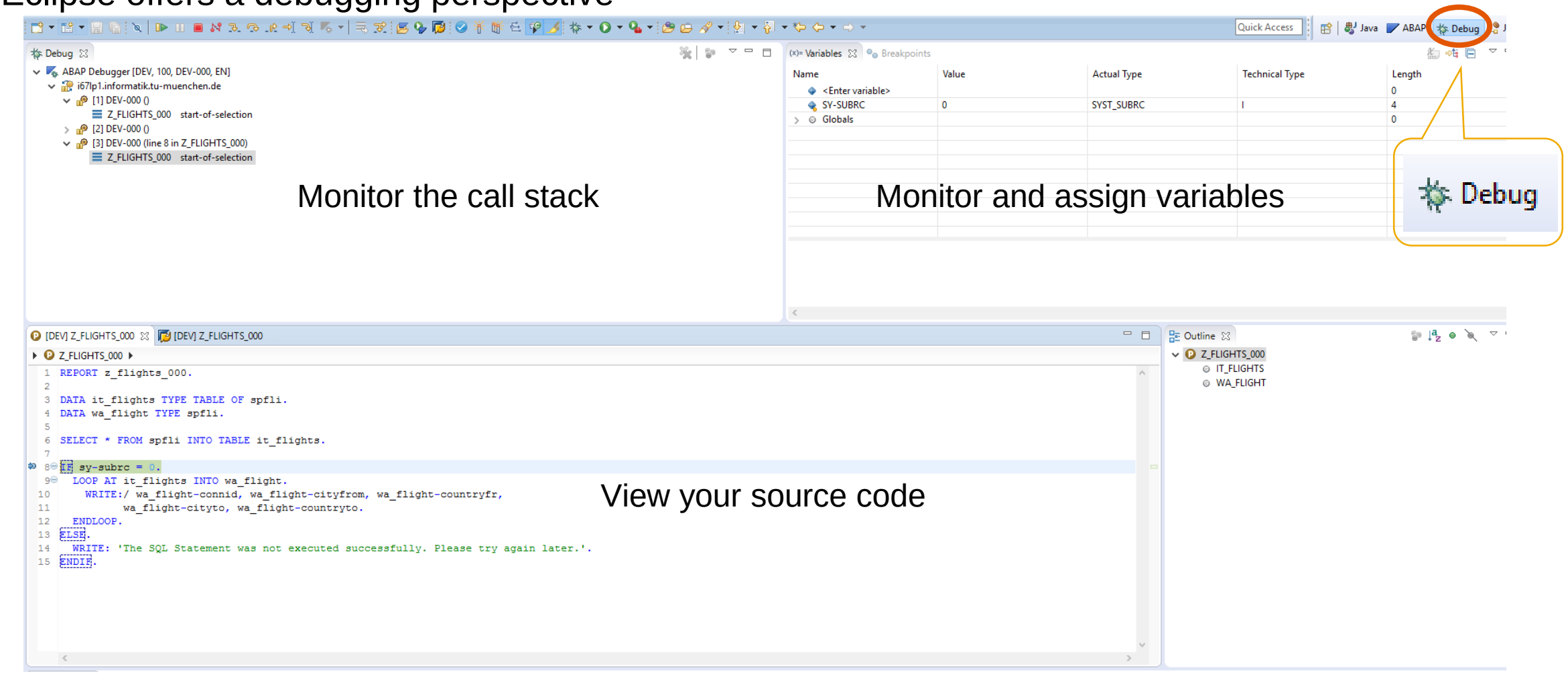
Why debugging?

- Sooner or later even professional programmers have to deal with software bugs
- Often programmers assume certain preconditions like:
  - Variable x = 16
  - In an if-statement the else-branch gets executed
- How to verify that the preconditions are met? → Debugging
- Debugging helps to answer following questions:
  - What is the bug / software fault?
  - Where does the bug occur?
  - How is it caused?

→ It helps getting a better understanding about software bugs

Debugging ABAP in Eclipse

- Eclipse offers a debugging perspective

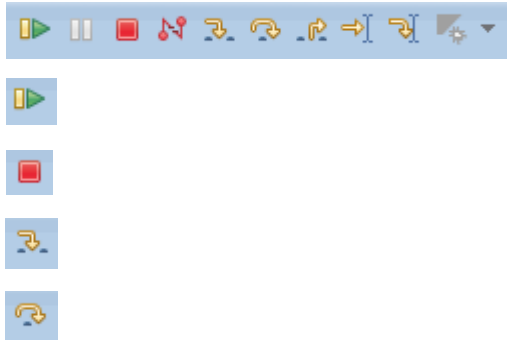


Working with the Eclipse Debugger

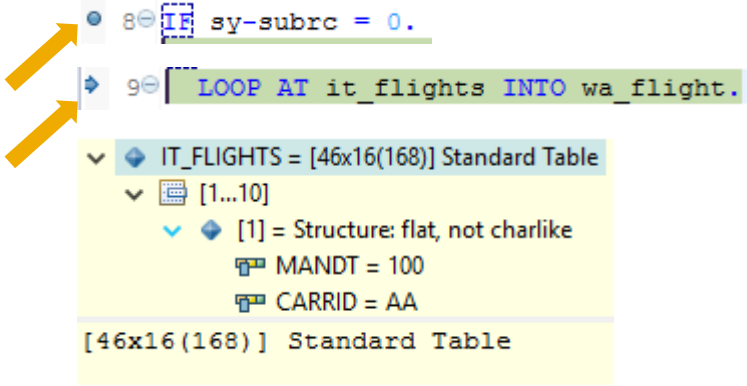
1. Set a breakpoint in your source code

```
8 IF sy-subrc = 0.
```

2. Control the debugger
   - Run the program again – it will stop at the next breakpoint
   - Terminate the program
   - Step deeper into the source code
   - Do one more step (step over)



3. Elements of the debugger
   - Breakpoint
   - Current execution marker
   - View contents of variable (e.g. by mouseover)



Now you know how to debug your ABAP program in Eclipse.

To consolidate your knowledge, you can do task 4 of the Basic Concepts exercise.

Check your knowledge

Check your knowledge

- Which are valid data types in ABAP.

  ☐ Integer (i) ☐ Double (d) ☐ Numerical text (n) ☐ Long (l)

- Following code works flawless (explain your answer!).

```
DATA intvalue TYPE i.
intvalue = 'ABC123'.
```

  ☐ True ☐ False

- What to do first when doing an ABAP exercise and your program produces an unexpected output:

  ☐ Ask your instructor ☐ Do some research on Google ☐ Use the debugger

Solution

Check your knowledge

- Which are valid data types in ABAP.

  ☒ Integer (i) ☐ Double (d) ☒ Numerical text (n) ☐ Long (l)

- Following code works flawless (explain your answer!).

```
DATA intvalue TYPE i.
intvalue = 'ABC123'.
```

  ☐ True ☒ False

See slide 16. To correct the code either change the type of the variable to ‘c’ or set the value to ‘123’

- What to do first when doing an ABAP exercise and your program produces an unexpected output:

  ☐ Ask your instructor ☐ So some research on Google ☒ Use the debugger

© 2018 SAP SE or an SAP affiliate company. All rights reserved.