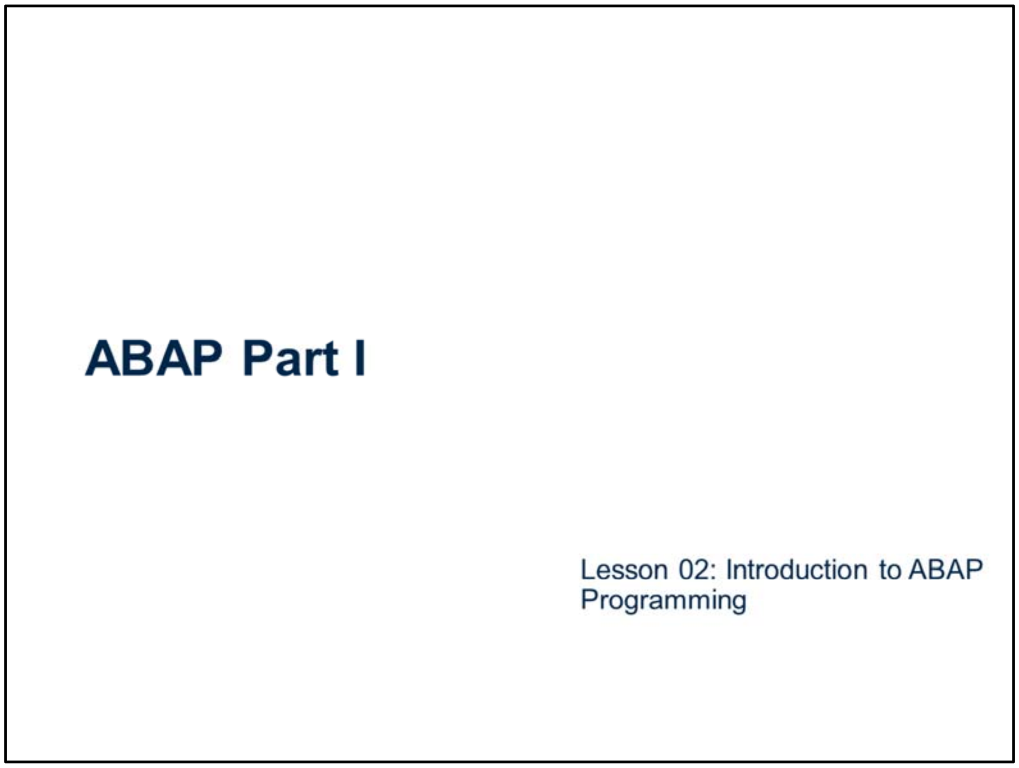
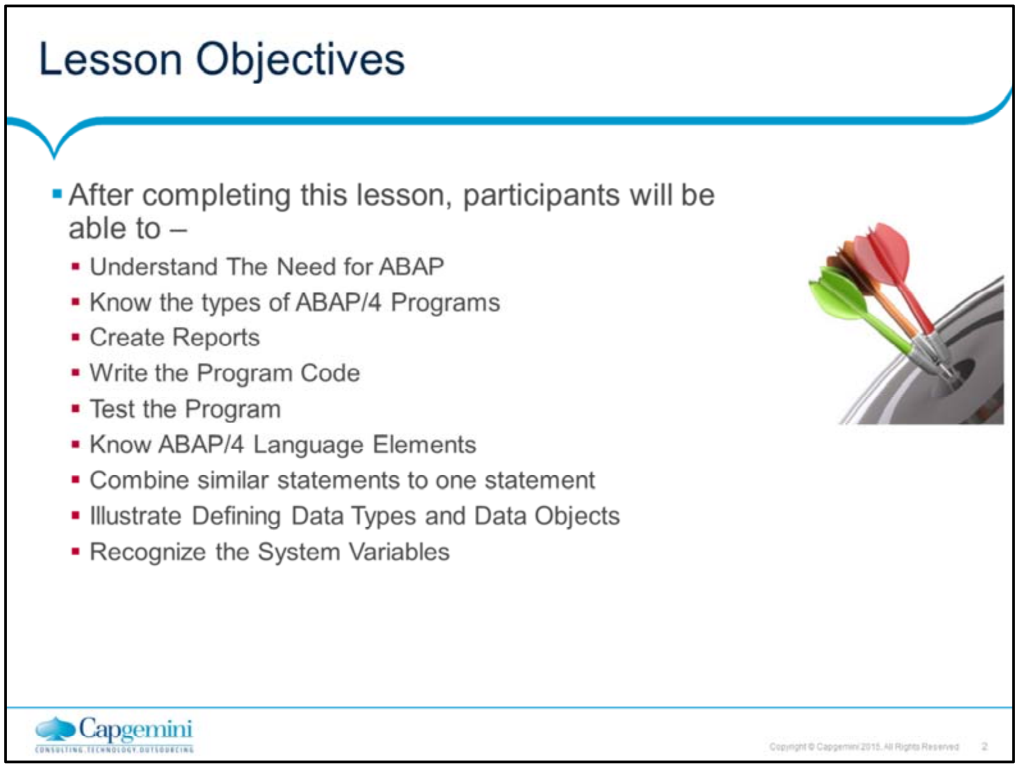


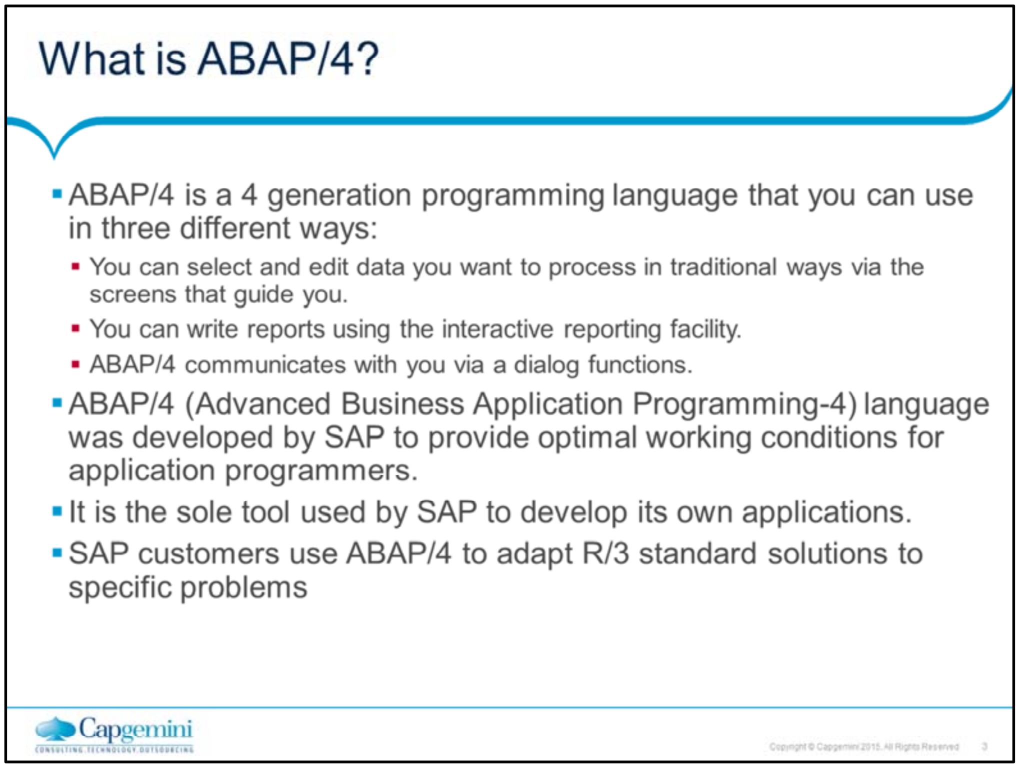
ABAP Part I

Lesson 02: Introduction to ABAP Programming



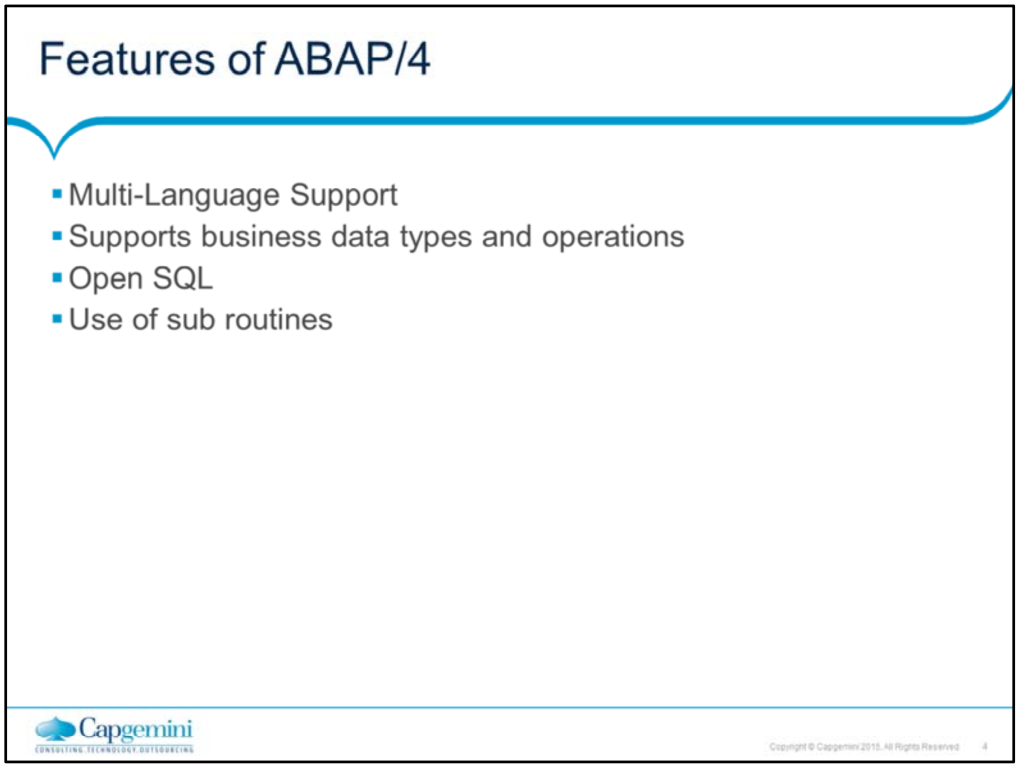
Lesson Objectives

- After completing this lesson, participants will be able to –
  - Understand The Need for ABAP
  - Know the types of ABAP/4 Programs
  - Create Reports
  - Write the Program Code
  - Test the Program
  - Know ABAP/4 Language Elements
  - Combine similar statements to one statement
  - Illustrate Defining Data Types and Data Objects
  - Recognize the System Variables



What is ABAP/4?

- ABAP/4 is a 4 generation programming language that you can use in three different ways:
  - You can select and edit data you want to process in traditional ways via the screens that guide you.
  - You can write reports using the interactive reporting facility.
  - ABAP/4 communicates with you via a dialog functions.
- ABAP/4 (Advanced Business Application Programming-4) language was developed by SAP to provide optimal working conditions for application programmers.
- It is the sole tool used by SAP to develop its own applications.
- SAP customers use ABAP/4 to adapt R/3 standard solutions to specific problems



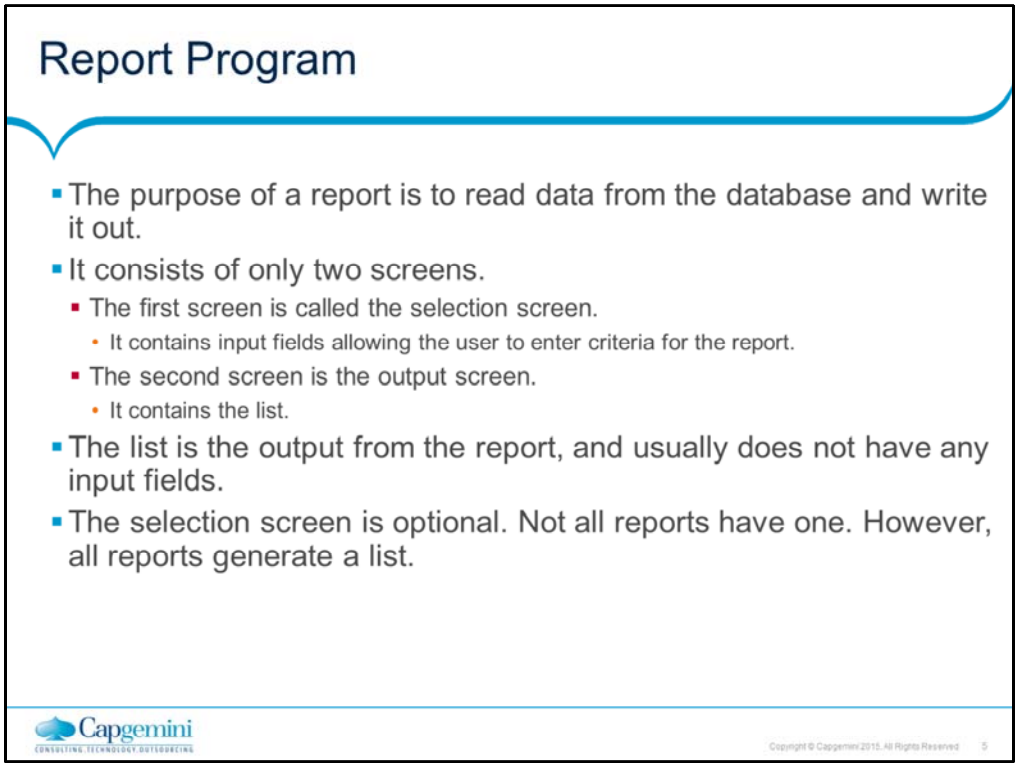
Features of ABAP/4

- Multi-Language Support
- Supports business data types and operations
- Open SQL
- Use of sub routines

Basic features of ABAP/4 are:
1) ABAP/4 contains

- Declarative elements for declaring data with various types and structures
- Operational elements for data manipulation
- Control elements for controlling the program flow
- Event elements for reacting to external events

2) Multi-Language Support : Text elements (e.g. titles, headers, and other text) used in a program are stored separately from the program code. One can change, translate, and maintain these at any time without changing the program code.
3) Open SQL : ABAP/4 contains a subset of SQL called Open SQL. With Open SQL, one can read and access database tables regardless of the database system being used.
4)Use of Subroutines: ABAP/4 allows one to define and call subroutines. The subroutines may be in the same program or in other programs. Parameters can be passed to and from subroutines in various ways.
ABAP/4 also contains a special kind of subroutine known as a *Function Module*. The Function Modules are created and maintained in a central library. Function Modules have a clearly defined data interface between the calling program and the subroutine. They can be tested in a stand-alone mode independent of the calling program.

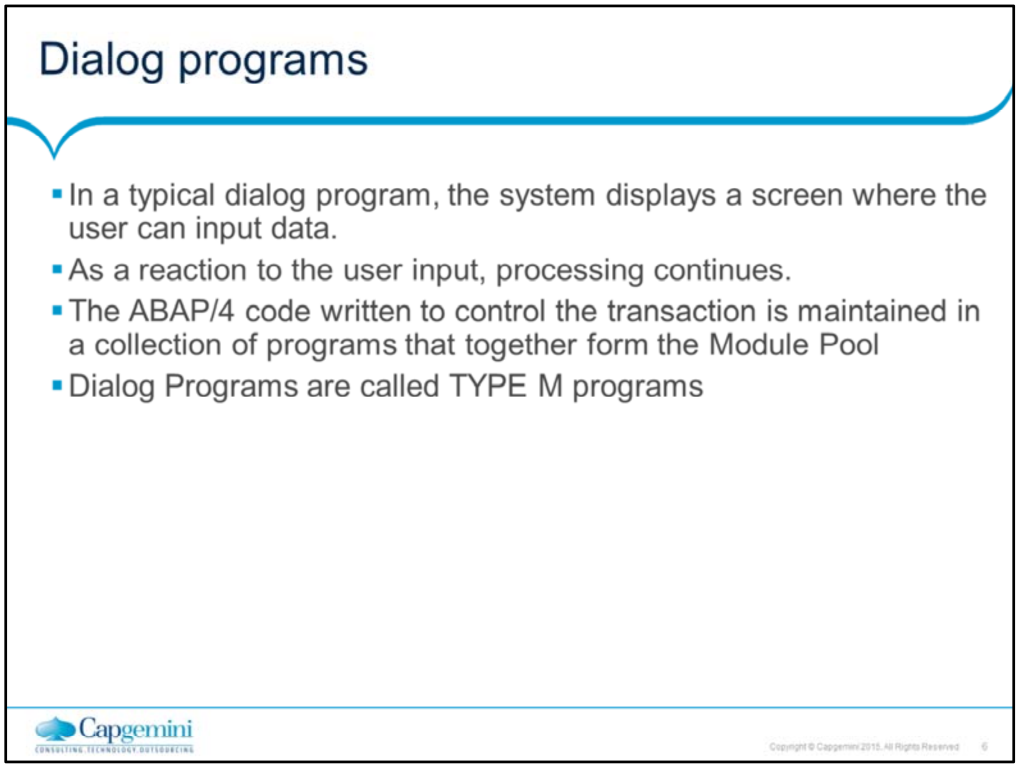


Report Program

- The purpose of a report is to read data from the database and write it out.
- It consists of only two screens.
  - The first screen is called the selection screen.
    - It contains input fields allowing the user to enter criteria for the report.
  - The second screen is the output screen.
    - It contains the list.
- The list is the output from the report, and usually does not have any input fields.
- The selection screen is optional. Not all reports have one. However, all reports generate a list.

Report Program: An ABAP/4 report is a program that retrieves data from the database, groups/filters according to different criteria and presents it on screen or as a printed list. Reports are called TYPE 1 program. A type 1 program can be started by entering its program name.

They are also known as executable programs

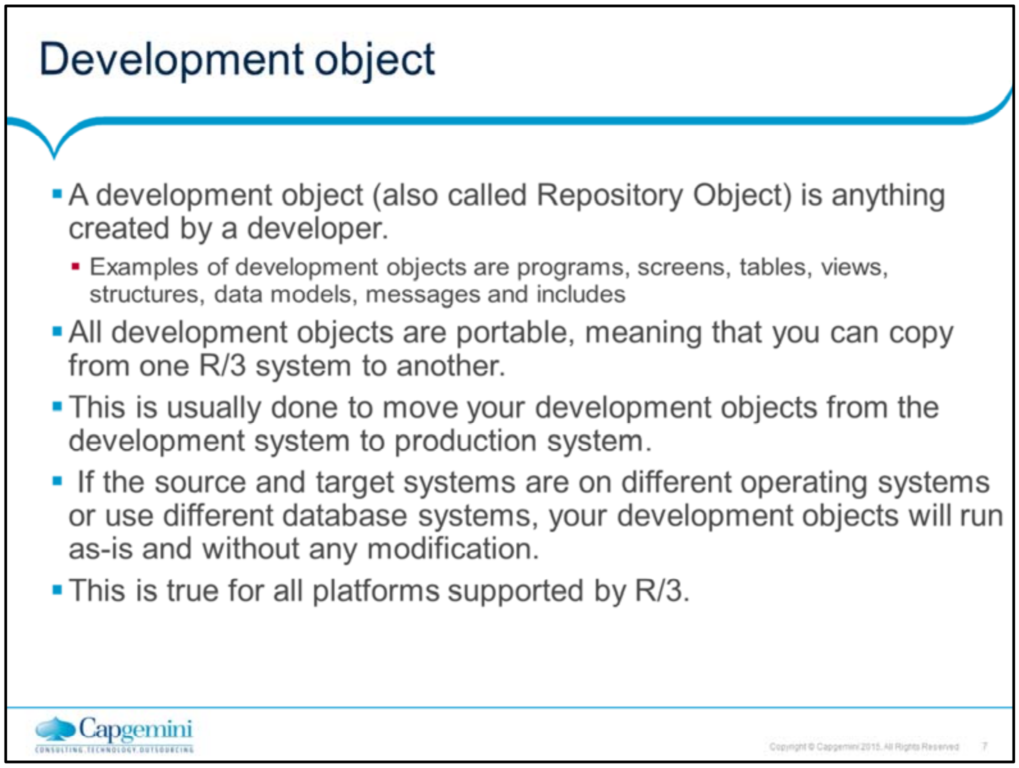


Dialog programs

- In a typical dialog program, the system displays a screen where the user can input data.
- As a reaction to the user input, processing continues.
- The ABAP/4 code written to control the transaction is maintained in a collection of programs that together form the Module Pool
- Dialog Programs are called TYPE M programs

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Type M programs can only be started using a transaction code.
A transaction code starts a screen, which consists of the screen itself and its flow logic.
Screen flow logic can call special processing blocks (dialog modules) in the corresponding ABAP/4 program.
Since type M programs contain mostly dialog modules, they are also known as module pools.

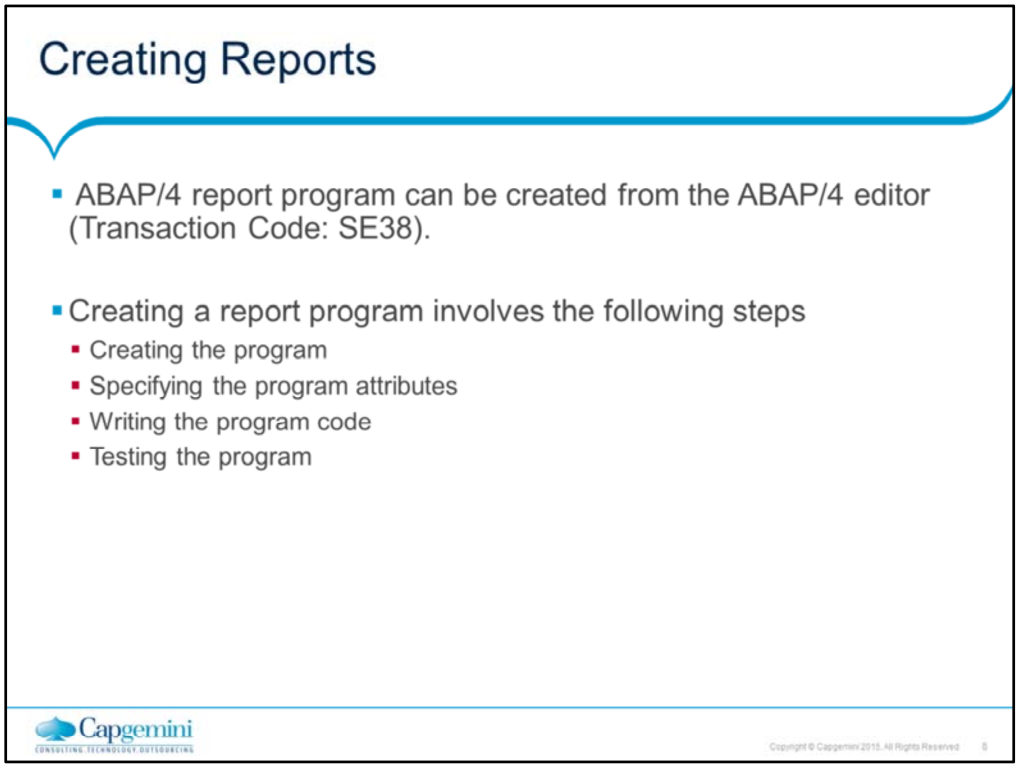


Development object

- A development object (also called Repository Object) is anything created by a developer.
  - Examples of development objects are programs, screens, tables, views, structures, data models, messages and includes
- All development objects are portable, meaning that you can copy from one R/3 system to another.
- This is usually done to move your development objects from the development system to production system.
- If the source and target systems are on different operating systems or use different database systems, your development objects will run as-is and without any modification.
- This is true for all platforms supported by R/3.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

7



Creating Reports

- ABAP/4 report program can be created from the ABAP/4 editor (Transaction Code: SE38).
- Creating a report program involves the following steps
  - Creating the program
  - Specifying the program attributes
  - Writing the program code
  - Testing the program

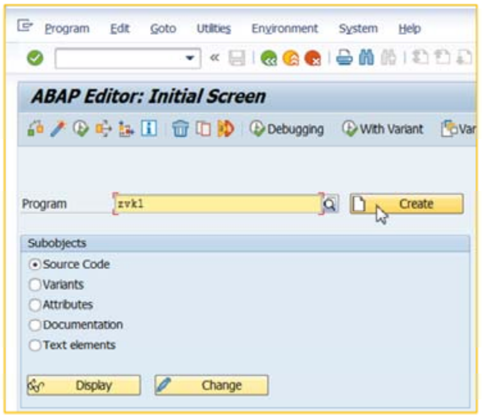
ABAP/4 reports consist of five components (sub objects):
Source code
Attributes
Text elements
Documentation
Variants

Only the source code and the attribute components are required. The rest of the components are optional.

Note that all development objects and their subobjects are stored in the R/3 system database. For example, the source code for a report is stored in database table dd010s.

Creating program

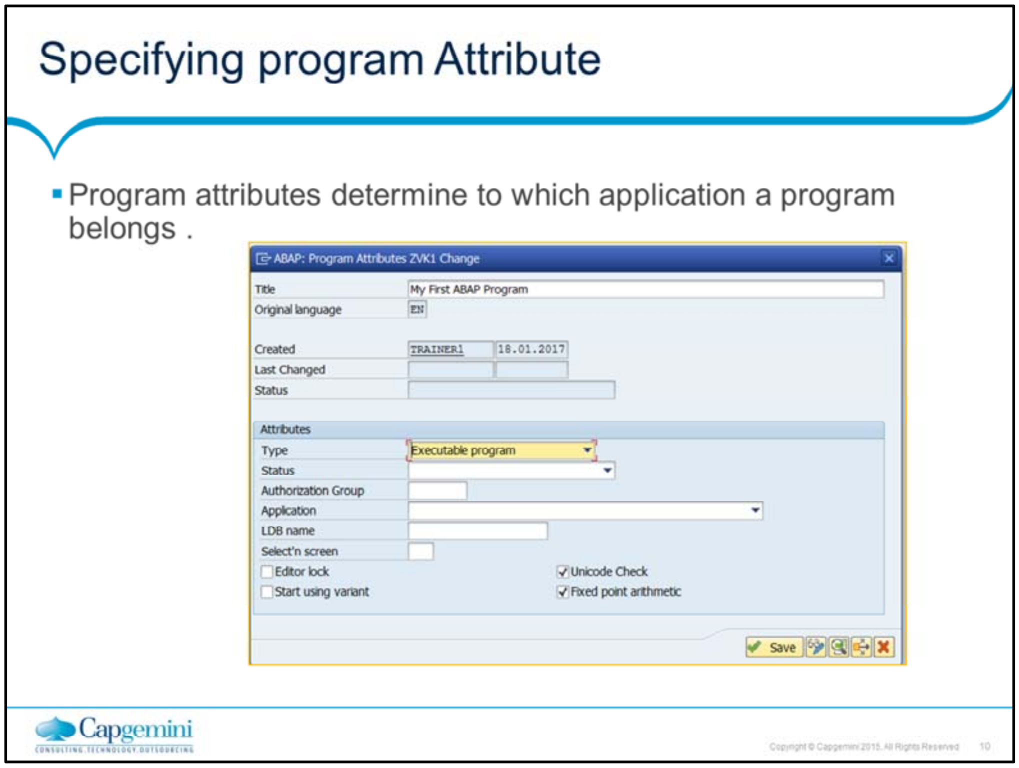
- Run transaction SE38 to go to the ABAP/4 Editor.
- This enables to create report program.



Program zvk1

Subobjects: [✓ Source Code]

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING



Specifying program Attribute

- Program attributes determine to which application a program belongs .

ABAP: Program Attributes ZVK1 Change

Title	My First ABAP Program
Original language	EN
Created	TRAINER1 18.01.2017
Last Changed	
Status	

Attributes

Type	Executable program
Status	
Authorization Group	
Application	
LDB name	
Select'n screen	

☐ Editor lock ✓ Unicode Check
☐ Start using variant ✓ Fixed point arithmetic

Program attributes determine to which application a program belongs and the logical database to which it is linked. Data to be entered include –
Title -The Title should ideally be descriptive of the function to be carried out by the program.
Type - The type defines the execution mode of the program.

> Value 1 declares the program as a standalone, online program that can be run independently.
> Value M, declares the program as a Module Pool. That means that the program cannot run standalone but serves as a main program for the program modules of dialog programming.
> Value I, is an Include program. Program code in an Include program can be used by many programs and helps to modularize logically related code.

Status: The program status specifies whether the program is a test program, a system program or a production program. Depending on the status, some of the utilities cannot be used. This entry is optional. For E.g.: you cannot use ABAP/4 debugging with system programs.

Application: The Application field contains the abbreviation of the application, e.g. F for Financial Accounting. This enables the system to assign the program to the appropriate business area. If the program is not application specific, the ‘*’ is used.

Authorization Group: Authorization group to which the program is assigned. The system
checks that the user belongs to an authorization group when

- o Starting or editing a program
- o Maintaining attributes
- o Using other program development utilities

•Development class/Package: The development class is important for transporting between systems. When transporting, one can combine Workbench objects assigned to one development class together. Once the attributes are saved a dialog box is displayed to input the Development class. Test program should be assigned class $TMP (or select the ‘Local Object’ button) Local Object cannot be transported from the development region to the testing region.

•Logical Databases (LDBs) from Application: This is only applicable for online programs(type 1) and determines which LDB the program uses to retrieve data from the database.

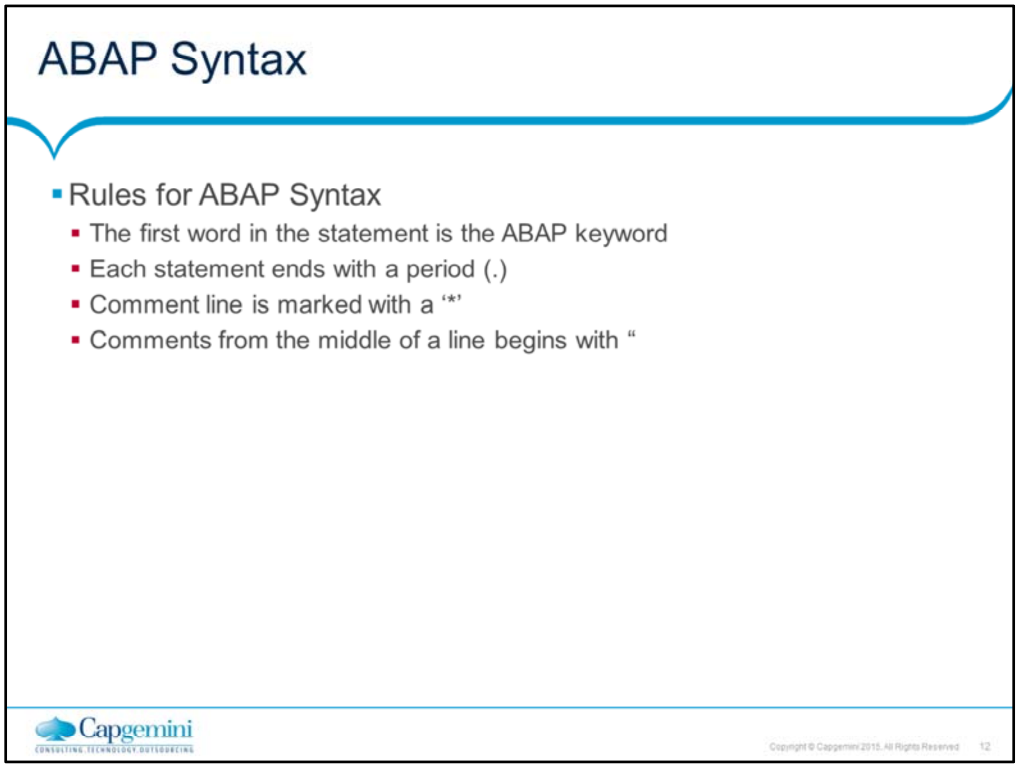
•Selection Screen Version: Is applicable to Online report using LDBs. LBDs may provide one or more selection screens on the basis of which data is retrieved. This field contains the selection screen number to be used.

•Uppercase/lowercase: If you want the ABAP/4 Editor to leave your program code as you have entered it when displaying and storing, leave this field blank. If you mark this field, all the program code (except text within quotation marks, and comments) is converted to uppercase.

•Editor lock: If this attribute is set, other users cannot modify, rename, or delete the program.

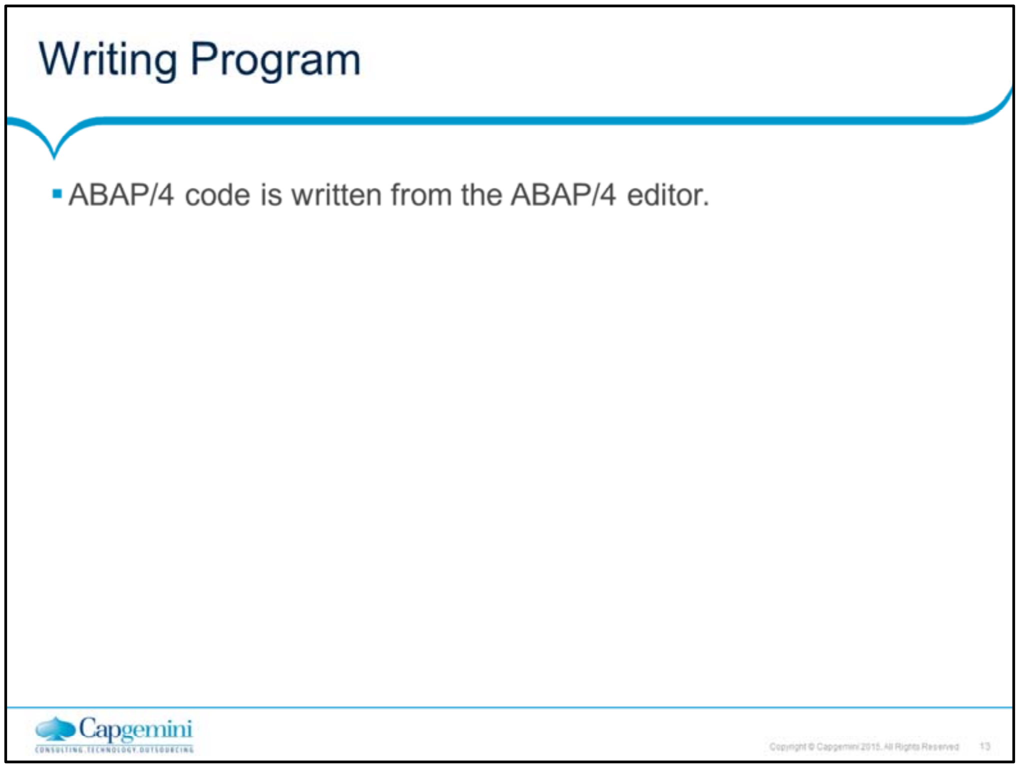
•Fixed Point Arithmetic :If this attribute is not set, the system does not recognize the decimal point for while using pack variables. Thus 3.14 would be interpreted as 314.

•Start via variant (report programs only): If this attribute is set, the user can only start your report program using a variant. Before starting the program at least one report variant must be created.



ABAP Syntax

- Rules for ABAP Syntax
  - The first word in the statement is the ABAP keyword
  - Each statement ends with a period (.)
  - Comment line is marked with a '*'
  - Comments from the middle of a line begins with "



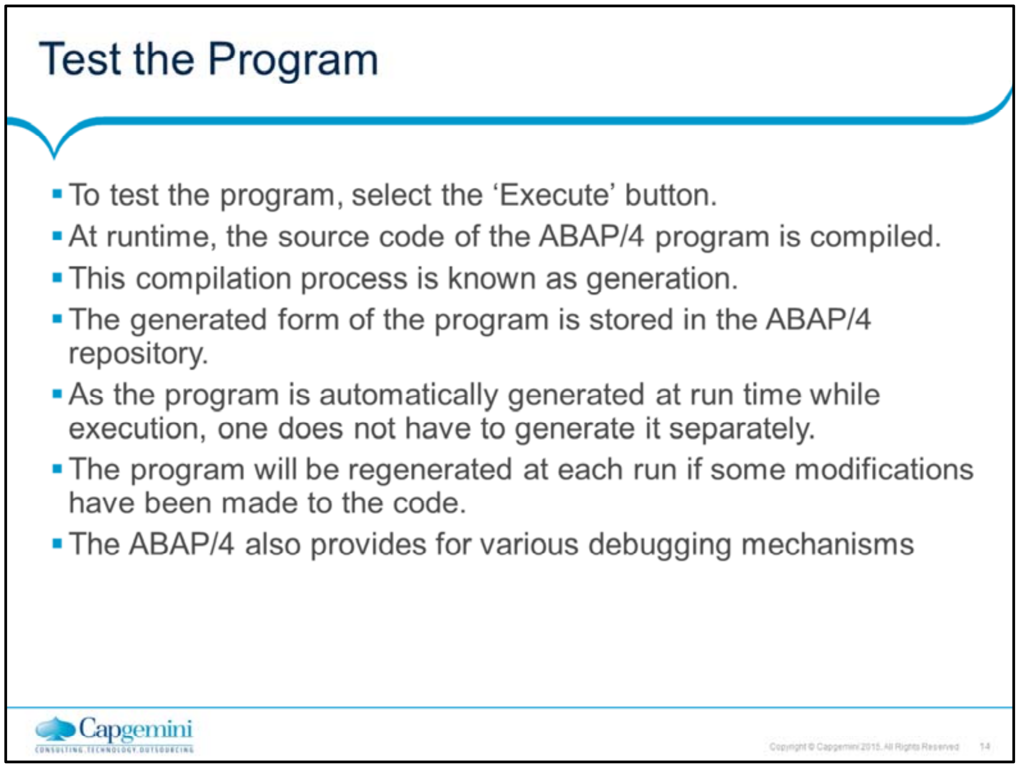
Writing Program

- ABAP/4 code is written from the ABAP/4 editor.

To go to the ABAP/4 editor one can select the ‘Source Code’ button from the Attributes screen or return to the ABAP/4 editor main screen, select ‘Source Code’ from the Object Components and select the ‘Change’ button.

The system automatically enters the first ABAP/4 statement - the REPORT statement. I.e.:
REPORT <Report Name as input in the Initial Screen>.

Once the code is written, one can check the syntax by selecting the ‘Check’ button. Error messages, if any, are displayed and the cursor is placed on the line in which the error has occurred. After the code is error free it can be saved using the ‘Save’ button.



Test the Program

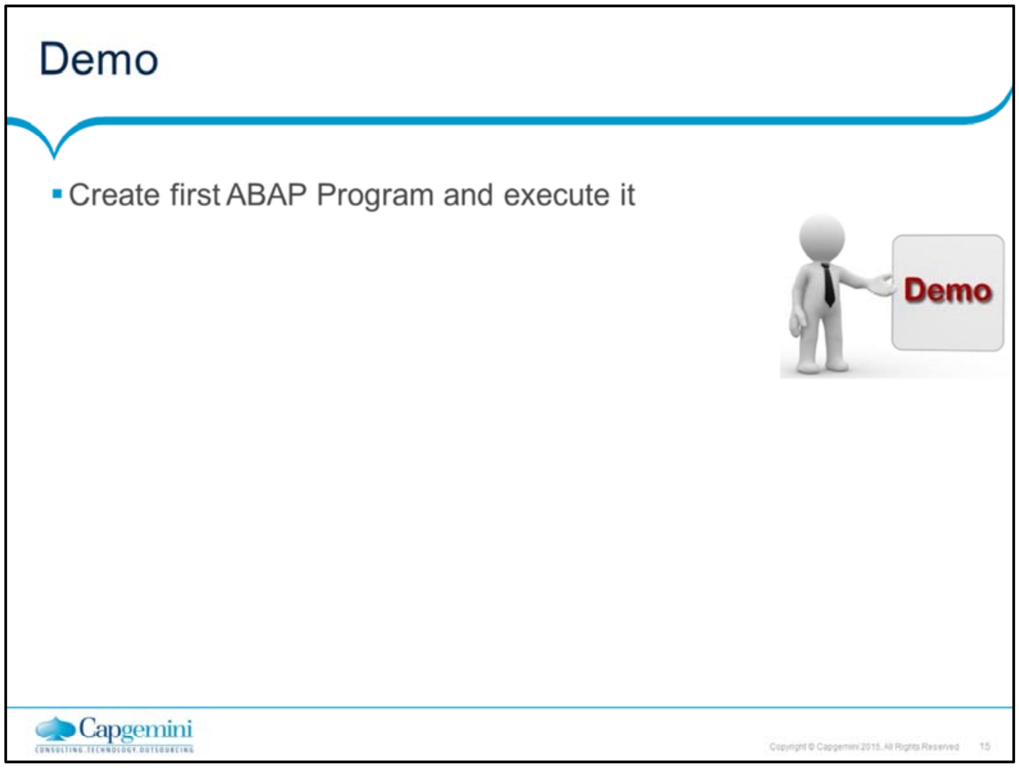
- To test the program, select the ‘Execute’ button.
- At runtime, the source code of the ABAP/4 program is compiled.
- This compilation process is known as generation.
- The generated form of the program is stored in the ABAP/4 repository.
- As the program is automatically generated at run time while execution, one does not have to generate it separately.
- The program will be regenerated at each run if some modifications have been made to the code.
- The ABAP/4 also provides for various debugging mechanisms

Capgemini CONSULTING.TECHNOLOGY.OUTSOURCING

ABAP/4 programs are interpreted; they are not compiled. The first time you execute a program, the system automatically generates a runtime object. The runtime object is pre-processed form of the source code. However, it is not an executable that you can run at the operating system level. Instead, it requires the R/3 system to interpret it. The runtime object is also known as the generated form of the program.

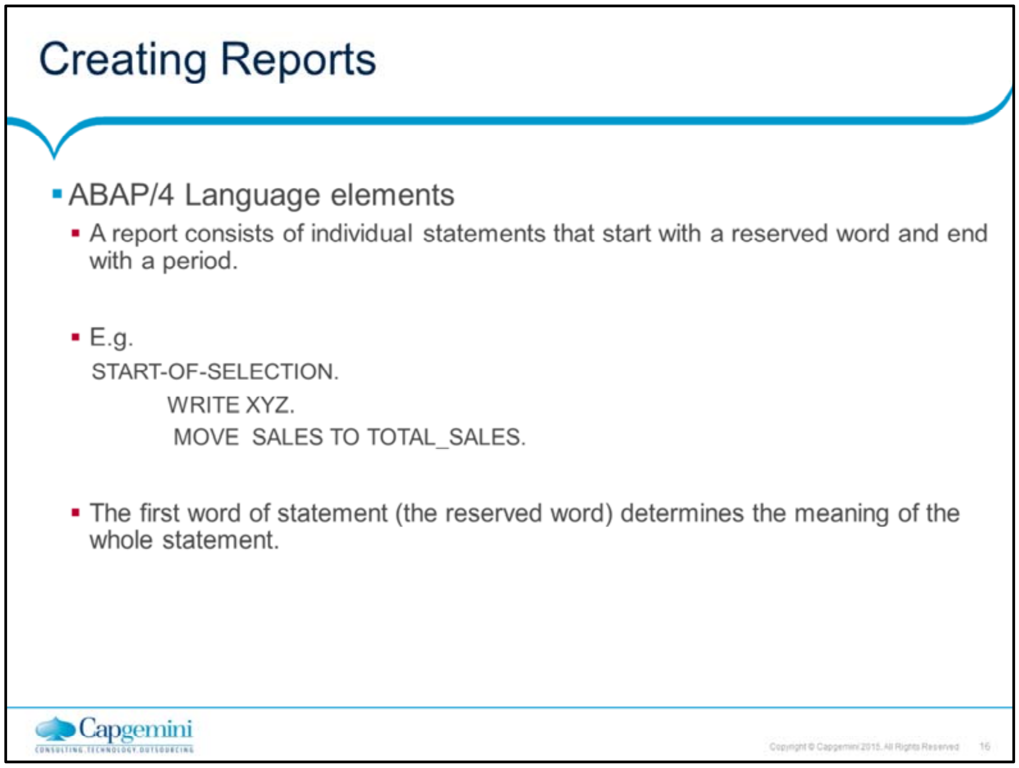
If you change the source code, the runtime object is automatically generated the next time you execute the program.

When you generate a development object, the system creates(compiles) a separate runtime object(LOAD) and stores it in the R/3 repository. This generated version is then the version that is executed(interpreted). Note that all changes to the development object become visible system-wide only when the program is activated. With inactive versions, you have a local, separate view of the R/3 repository, which provides the basis for a local runtime system. During Activation, the system creates a Run Time object. Upon execution, the system executes this Run Time Object



Demo

- Create first ABAP Program and execute it

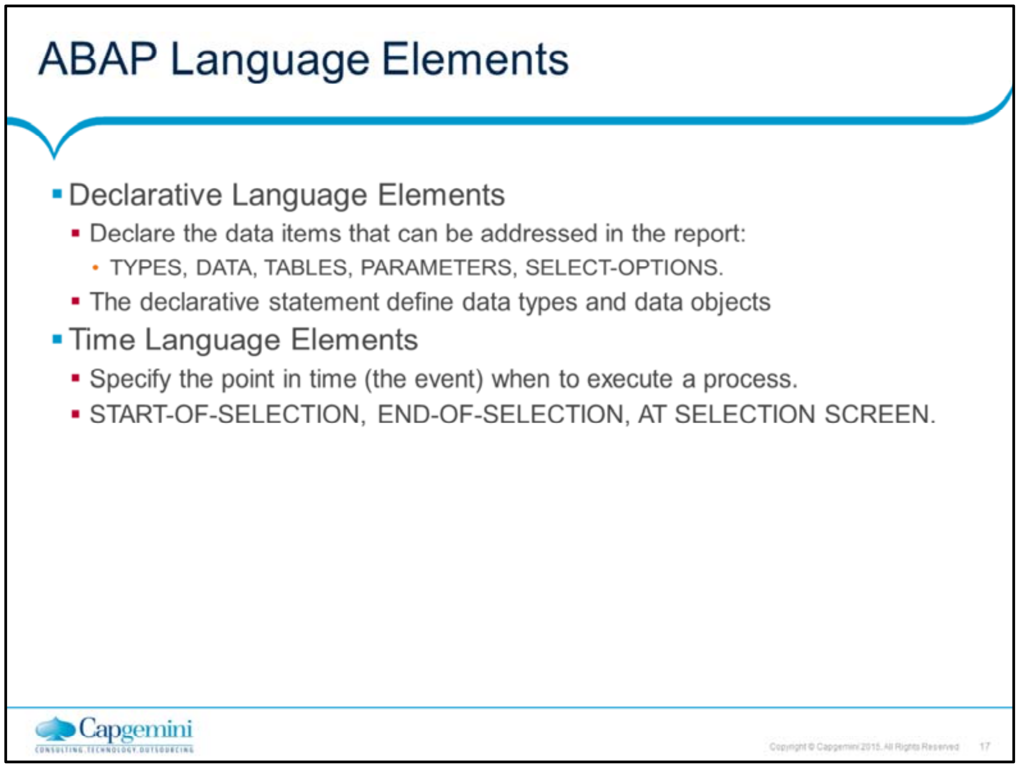


Creating Reports

- ABAP/4 Language elements
  - A report consists of individual statements that start with a reserved word and end with a period.
  - E.g.

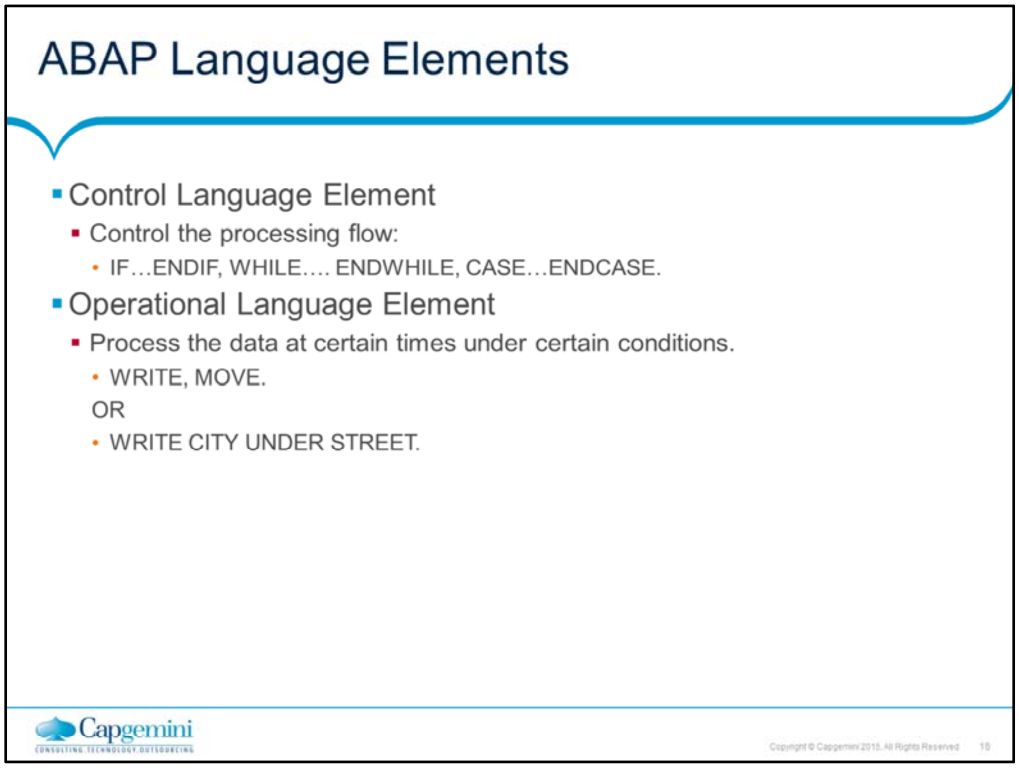
```
START-OF-SELECTION.
          WRITE XYZ.
           MOVE  SALES TO TOTAL_SALES.
```

  - The first word of statement (the reserved word) determines the meaning of the whole statement.



ABAP Language Elements

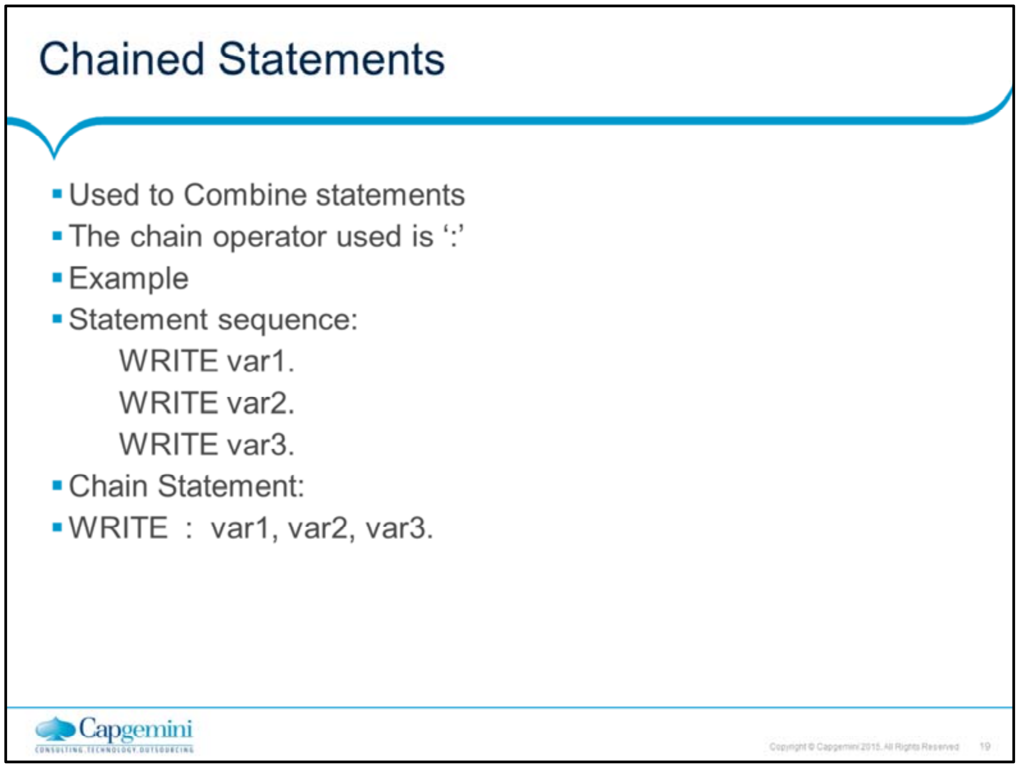
- Declarative Language Elements
  - Declare the data items that can be addressed in the report:
    - TYPES, DATA, TABLES, PARAMETERS, SELECT-OPTIONS.
  - The declarative statement define data types and data objects
- Time Language Elements
  - Specify the point in time (the event) when to execute a process.
  - START-OF-SELECTION, END-OF-SELECTION, AT SELECTION SCREEN.



ABAP Language Elements

- Control Language Element
  - Control the processing flow:
    - IF…ENDIF, WHILE…. ENDWHILE, CASE…ENDCASE.
- Operational Language Element
  - Process the data at certain times under certain conditions.
    - WRITE, MOVE.

    OR
    - WRITE CITY UNDER STREET.

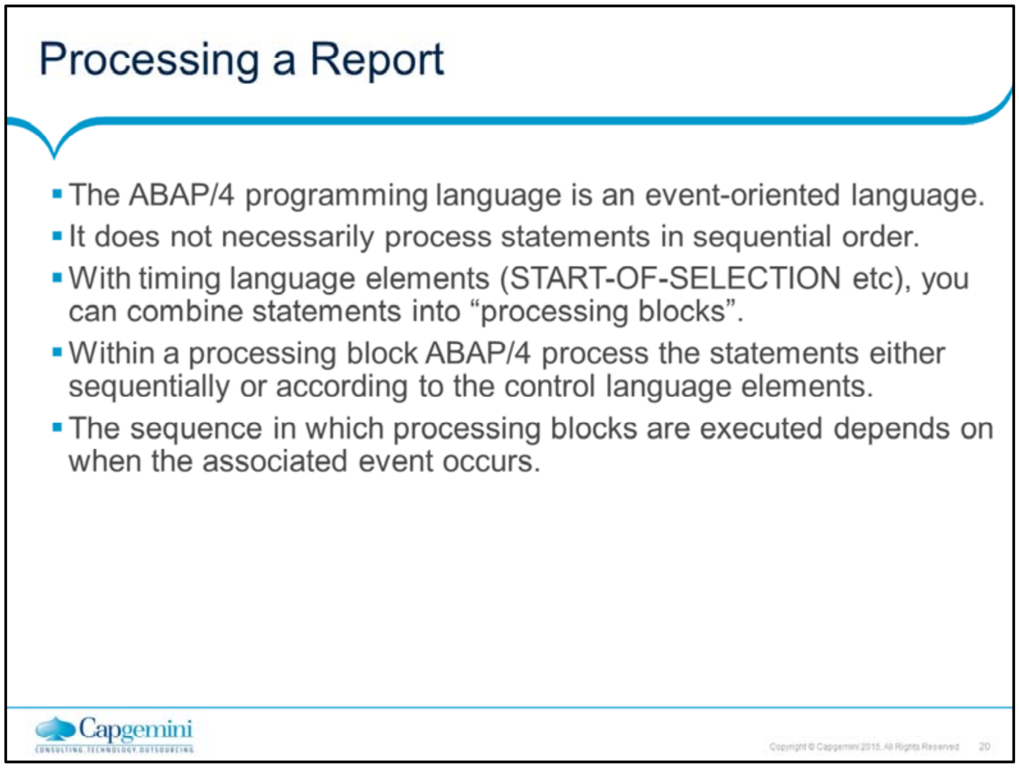


Chained Statements

- Used to Combine statements
- The chain operator used is ':'
- Example
- Statement sequence:

```
WRITE var1.
WRITE var2.
WRITE var3.
```

- Chain Statement:
- WRITE : var1, var2, var3.

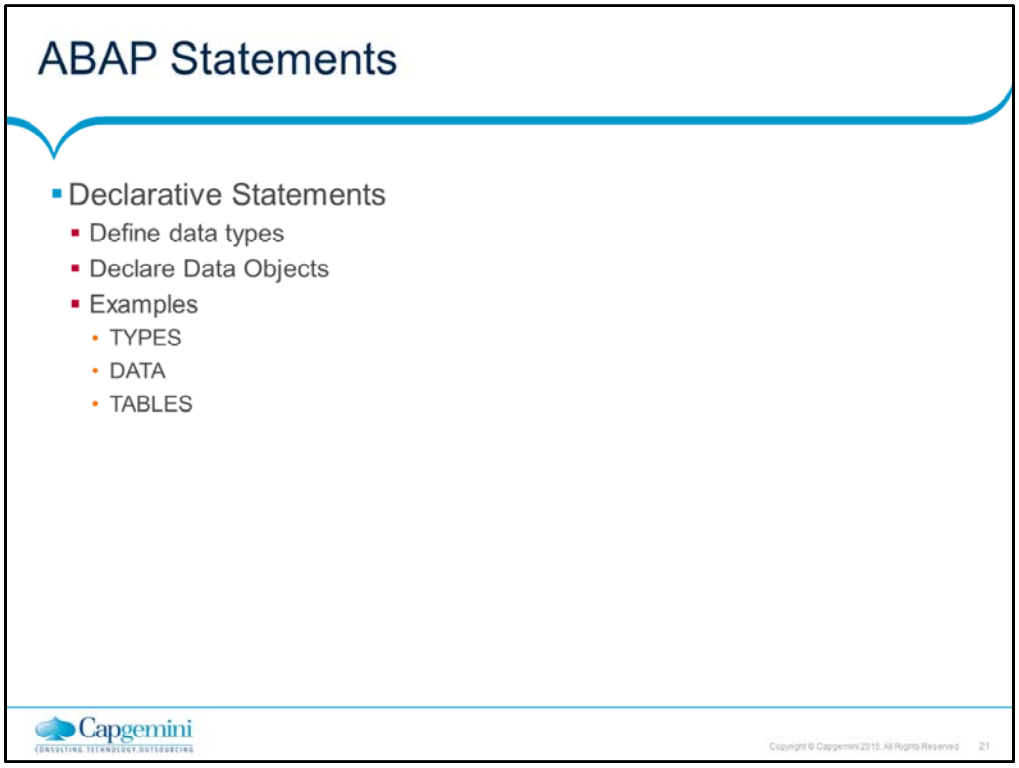


Processing a Report

- The ABAP/4 programming language is an event-oriented language.
- It does not necessarily process statements in sequential order.
- With timing language elements (START-OF-SELECTION etc), you can combine statements into "processing blocks".
- Within a processing block ABAP/4 process the statements either sequentially or according to the control language elements.
- The sequence in which processing blocks are executed depends on when the associated event occurs.

```
e.g.
report demo.
<declaration section>

at selection-screen.        " processing block
start-of-selection.         " processing block.

        if (   ).
             -----
            -------
        endif.

end-of-selection.
      write: 'the sum is ',  total.
```

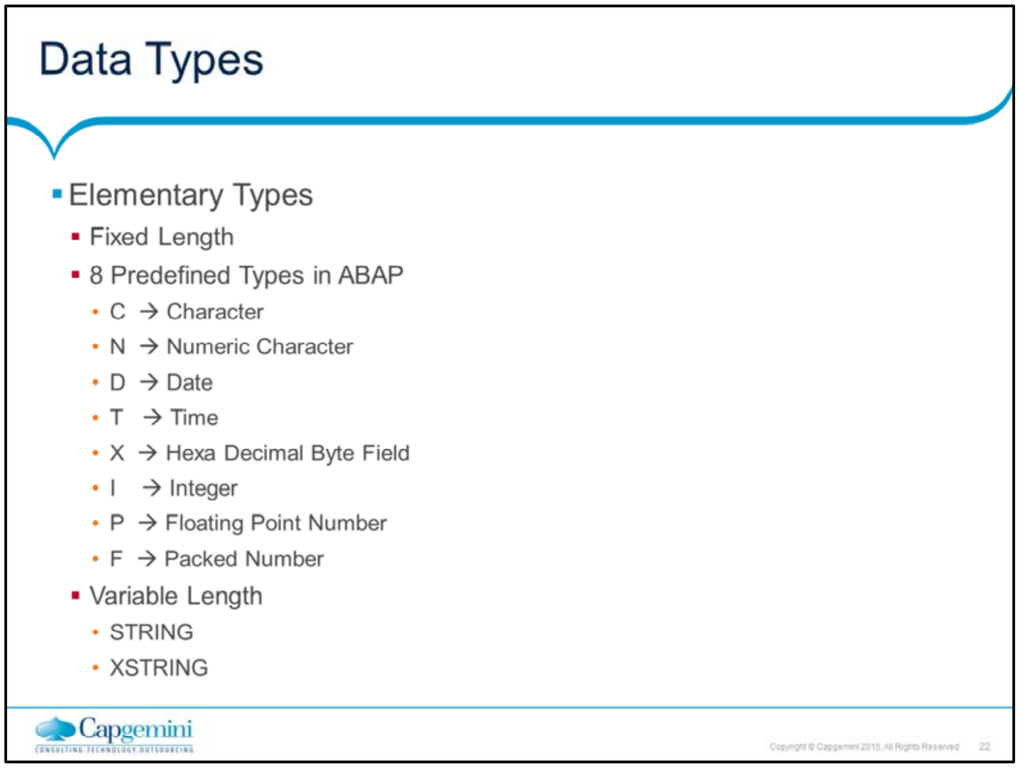
A timing language element always introduces a new processing block. Note that there is no particular language element to mark the end of processing block. A new event or the definition of a subroutine (FORM…ENDFORM) ends the preceding processing block. For reasons of clarity you should list processing blocks in their order of execution.



ABAP Statements

- Declarative Statements
  - Define data types
  - Declare Data Objects
  - Examples
    - TYPES
    - DATA
    - TABLES

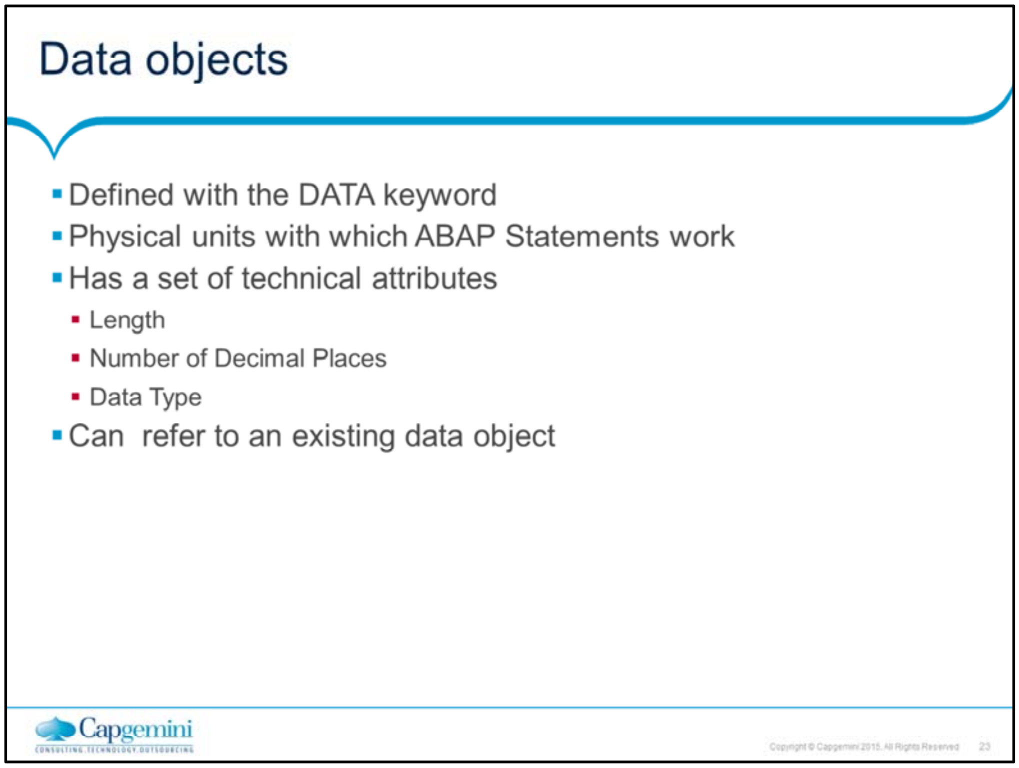
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING



Data Types

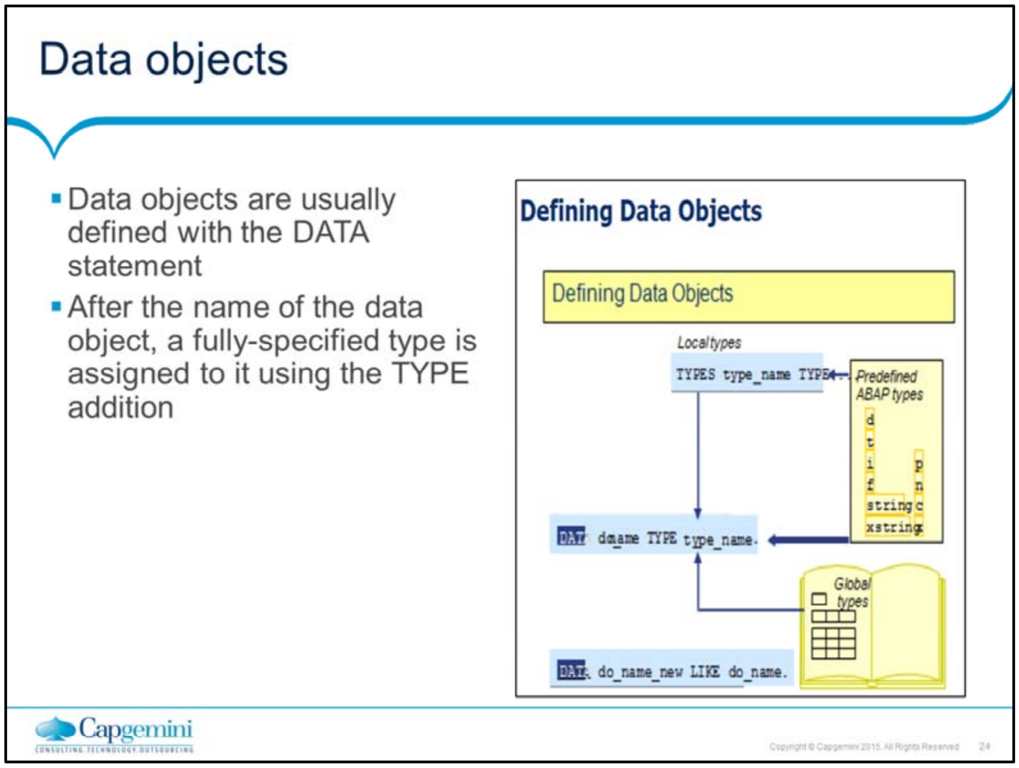
- Elementary Types
  - Fixed Length
  - 8 Predefined Types in ABAP
    - C → Character
    - N → Numeric Character
    - D → Date
    - T → Time
    - X → Hexa Decimal Byte Field
    - I → Integer
    - P → Floating Point Number
    - F → Packed Number
  - Variable Length
    - STRING
    - XSTRING

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING



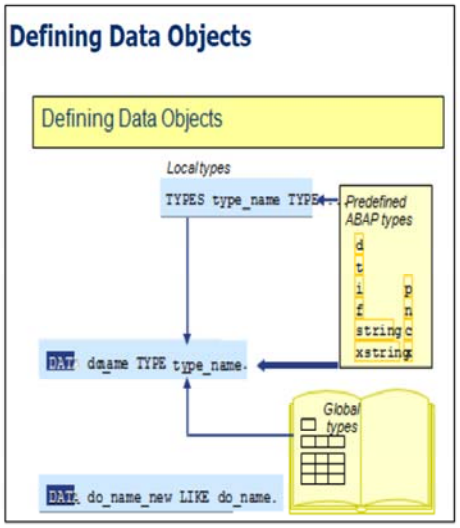
Data objects

- Defined with the DATA keyword
- Physical units with which ABAP Statements work
- Has a set of technical attributes
  - Length
  - Number of Decimal Places
  - Data Type
- Can refer to an existing data object



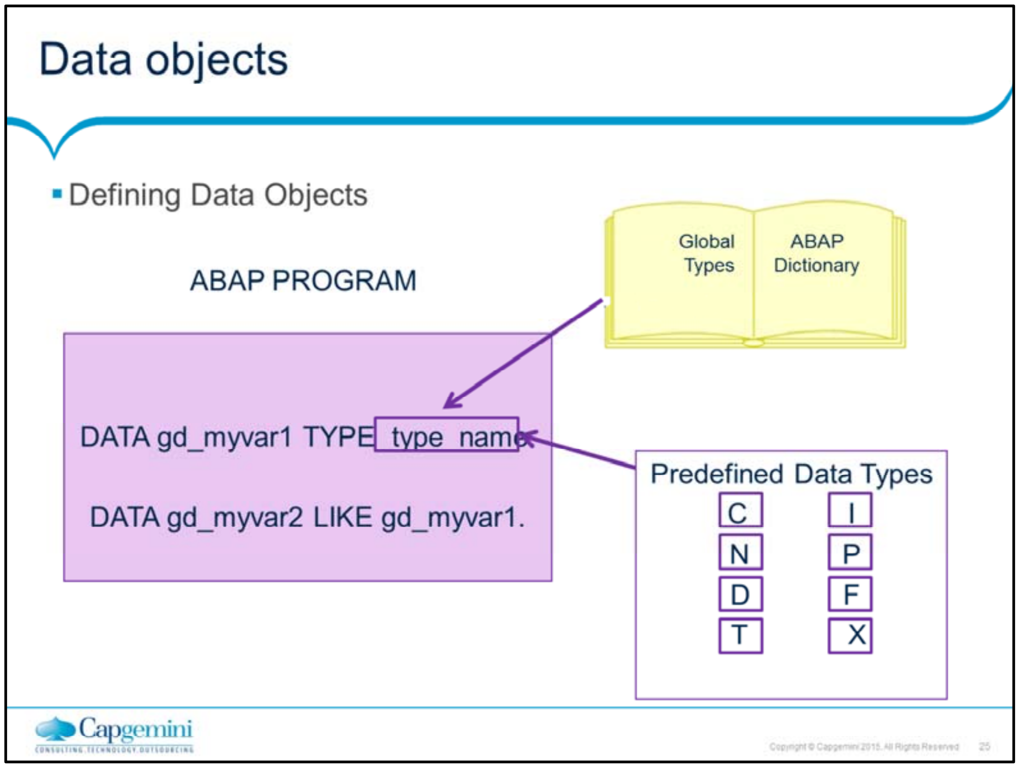
Data objects

- Data objects are usually defined with the DATA statement
- After the name of the data object, a fully-specified type is assigned to it using the TYPE addition



Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

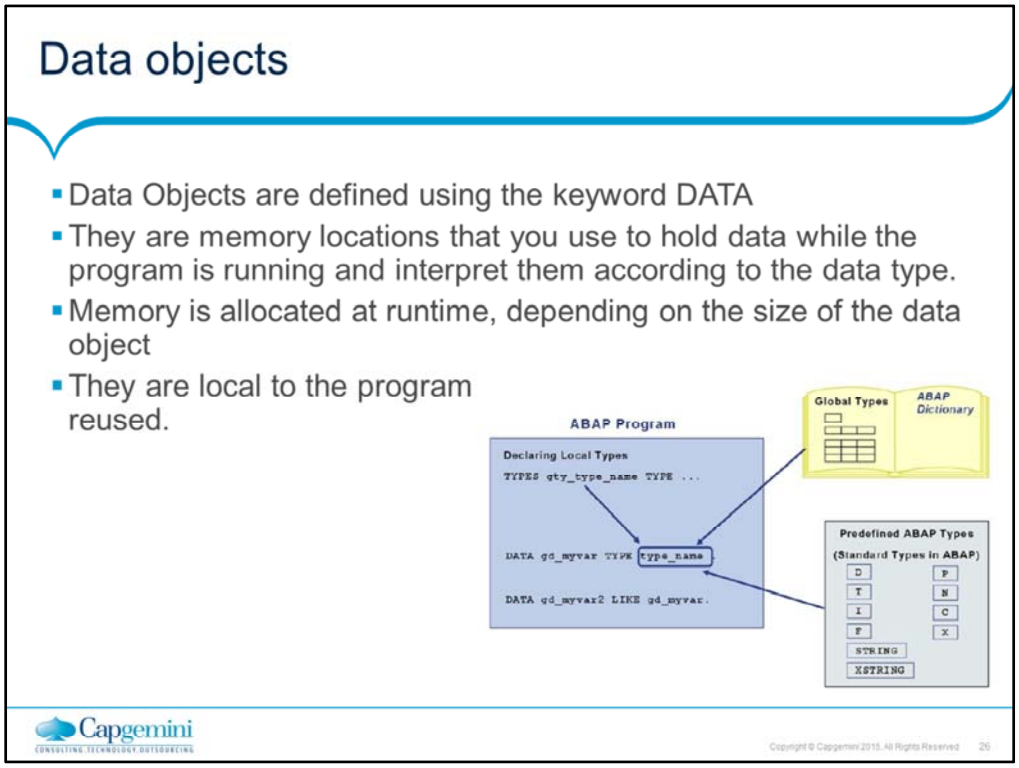
24



Data objects

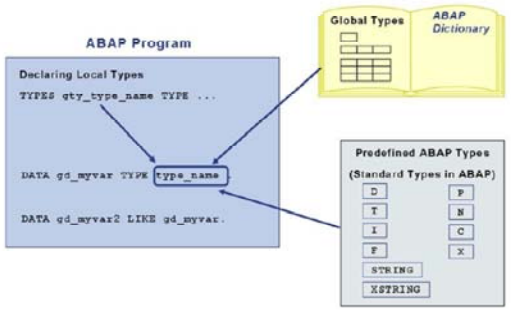
- Defining Data Objects

ABAP PROGRAM
DATA gd_myvar1 TYPE type_name
DATA gd_myvar2 LIKE gd_myvar1.
Global Types
ABAP Dictionary
Predefined Data Types
C
I
N
P
D
F
T
X
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING
Copyright © Capgemini 2015. All Rights Reserved
25



Data objects

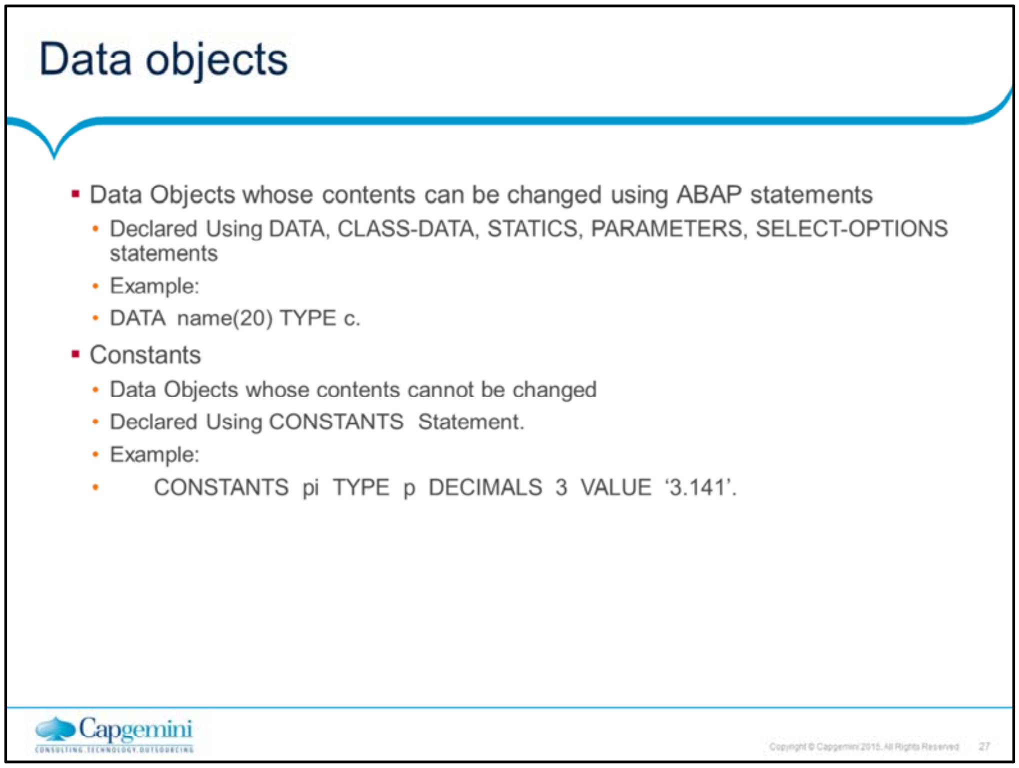
- Data Objects are defined using the keyword DATA
- They are memory locations that you use to hold data while the program is running and interpret them according to the data type.
- Memory is allocated at runtime, depending on the size of the data object
- They are local to the program reused.



```
TYPES gty_type_name TYPE ...

DATA gd_myvar TYPE type_name

DATA gd_myvar2 LIKE gd_myvar.
```

Capgemini

26

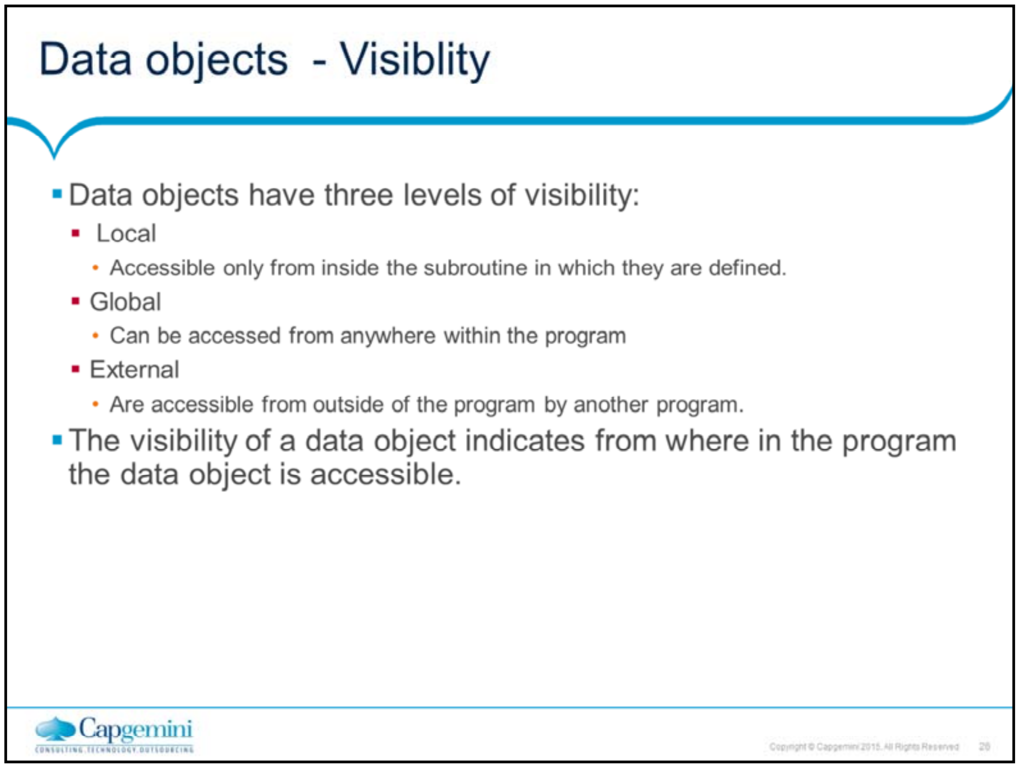


Data objects

- Data Objects whose contents can be changed using ABAP statements
  - Declared Using DATA, CLASS-DATA, STATICS, PARAMETERS, SELECT-OPTIONS statements
  - Example:
  - DATA name(20) TYPE c.
- Constants
  - Data Objects whose contents cannot be changed
  - Declared Using CONSTANTS Statement.
  - Example:
  - CONSTANTS pi TYPE p DECIMALS 3 VALUE ‘3.141’.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

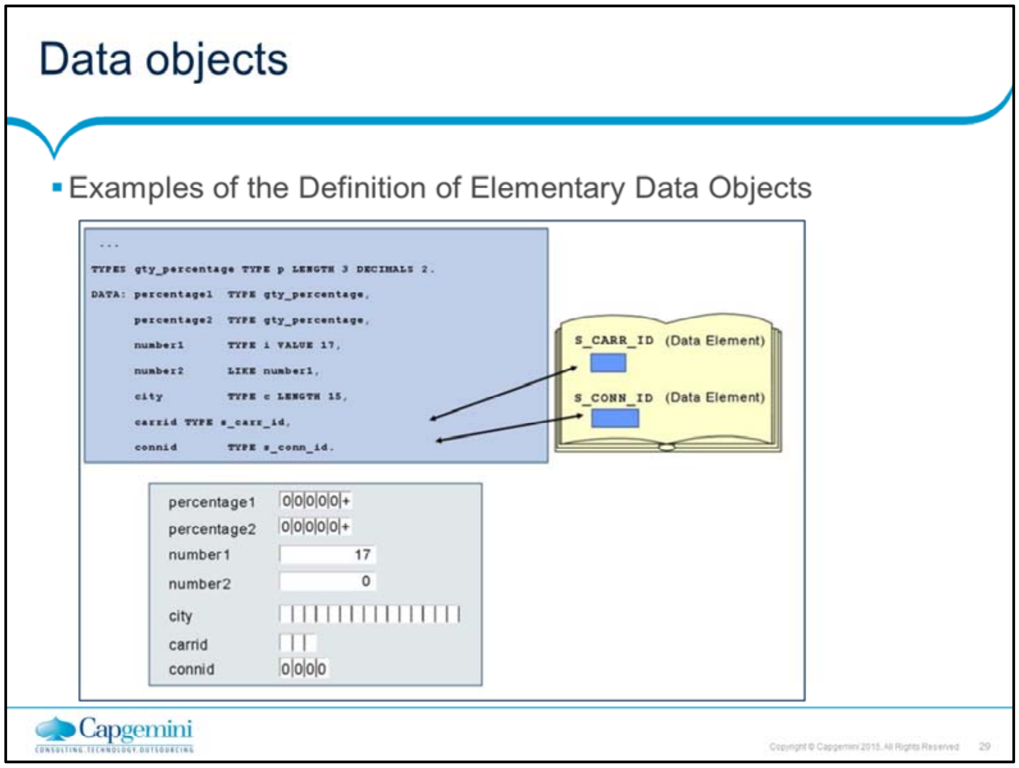
When the program starts, the memory allocation for each data object occurs in the roll area of the program. While the program is running, you can read the contents of a non-modifiable data object or put data into a modifiable data object and then retrieve it. When the program ends, the system frees the memory for all data objects and their contents are lost.



Data objects - Visiblity

- Data objects have three levels of visibility:
  - Local
    - Accessible only from inside the subroutine in which they are defined.
  - Global
    - Can be accessed from anywhere within the program
  - External
    - Are accessible from outside of the program by another program.
- The visibility of a data object indicates from where in the program the data object is accessible.

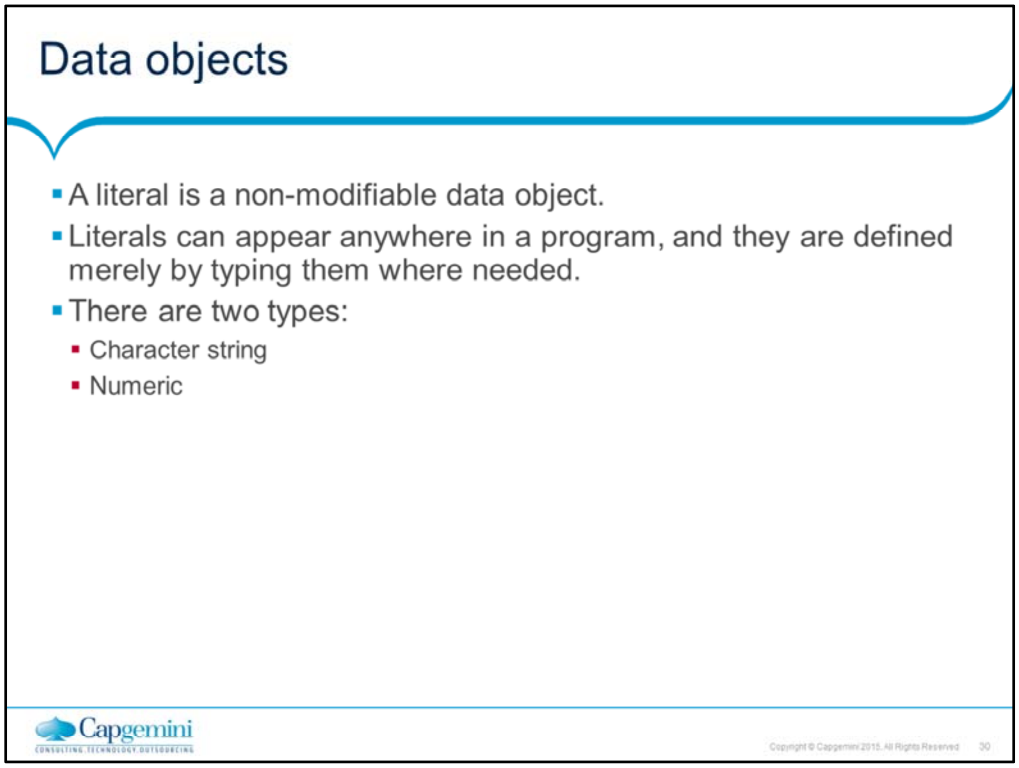
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING



Data objects

- Examples of the Definition of Elementary Data Objects

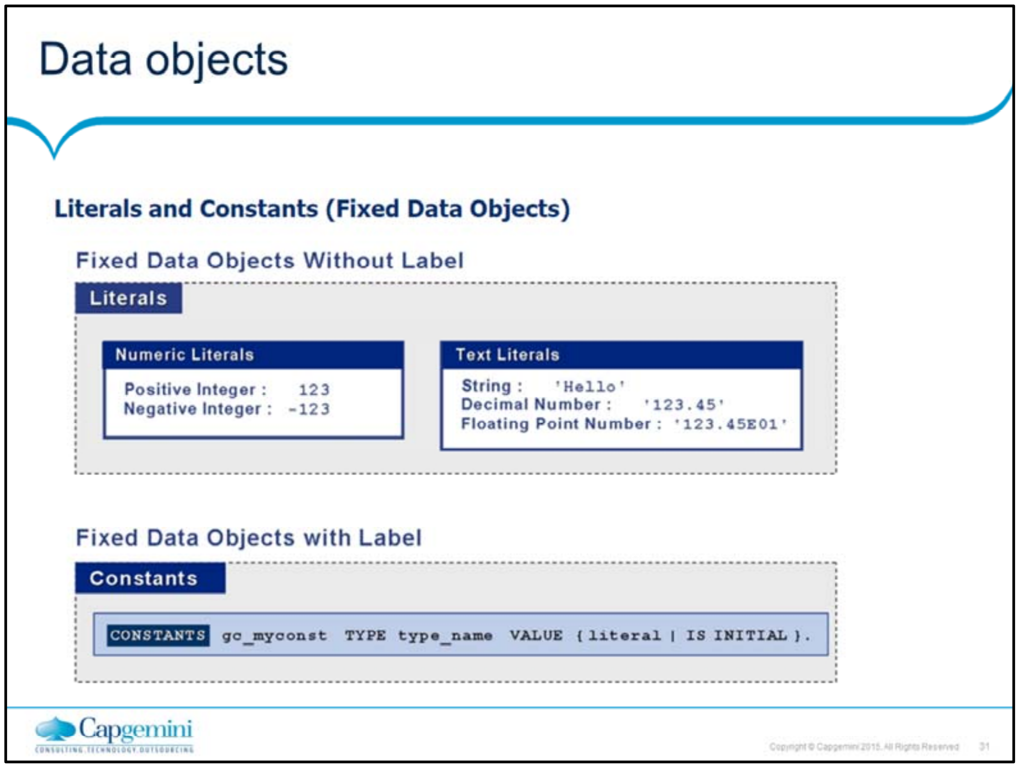
```
...
TYPES gty_percentage TYPE p LENGTH 3 DECIMALS 2.
DATA: percentage1  TYPE gty_percentage,
      percentage2  TYPE gty_percentage,
      number1      TYPE i VALUE 17,
      number2      LIKE number1,
      city         TYPE c LENGTH 15,
      carrid TYPE s_carr_id,
      connid       TYPE s_conn_id.
```



Data objects

- A literal is a non-modifiable data object.
- Literals can appear anywhere in a program, and they are defined merely by typing them where needed.
- There are two types:
  - Character string
  - Numeric

Character literals are sequences of alphanumeric characters in the source code of an ABAP program enclosed in single quotation marks or backquotes. Character literals enclosed in quotation marks have the predefined ABAP type C and are described as **text field literals**. Literals enclosed in backquotes have the ABAP type STRING and are described as **string literals.**
Examples of text field literals:
'Antony Smith'
'69190 Walldorf'
Examples of string literals:
`Anton Schmitt `
`69190 Walldorf `
Number literals
Numeric literals are sequences of digits which may contain a plus or minus sign



Data objects

Literals and Constants (Fixed Data Objects)

Fixed Data Objects Without Label

Literals

Numeric Literals

Positive Integer :	123
Negative Integer :	-123

Text Literals

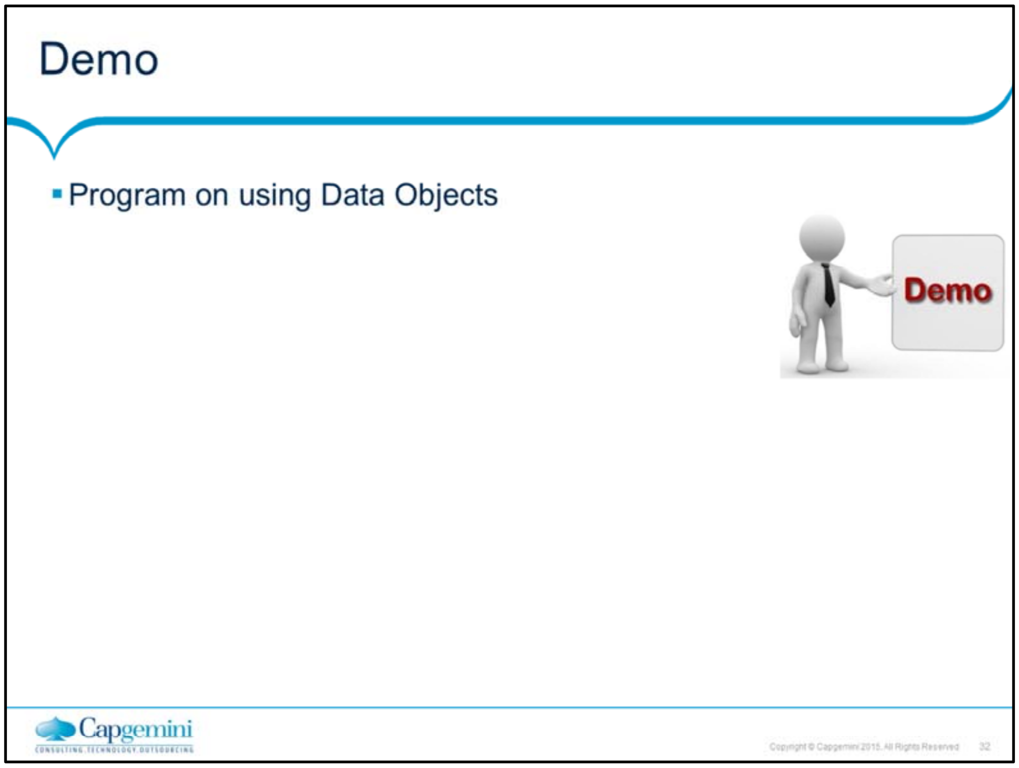
String :	'Hello'
Decimal Number :	'123.45'
Floating Point Number :	'123.45E01'

Fixed Data Objects with Label

Constants

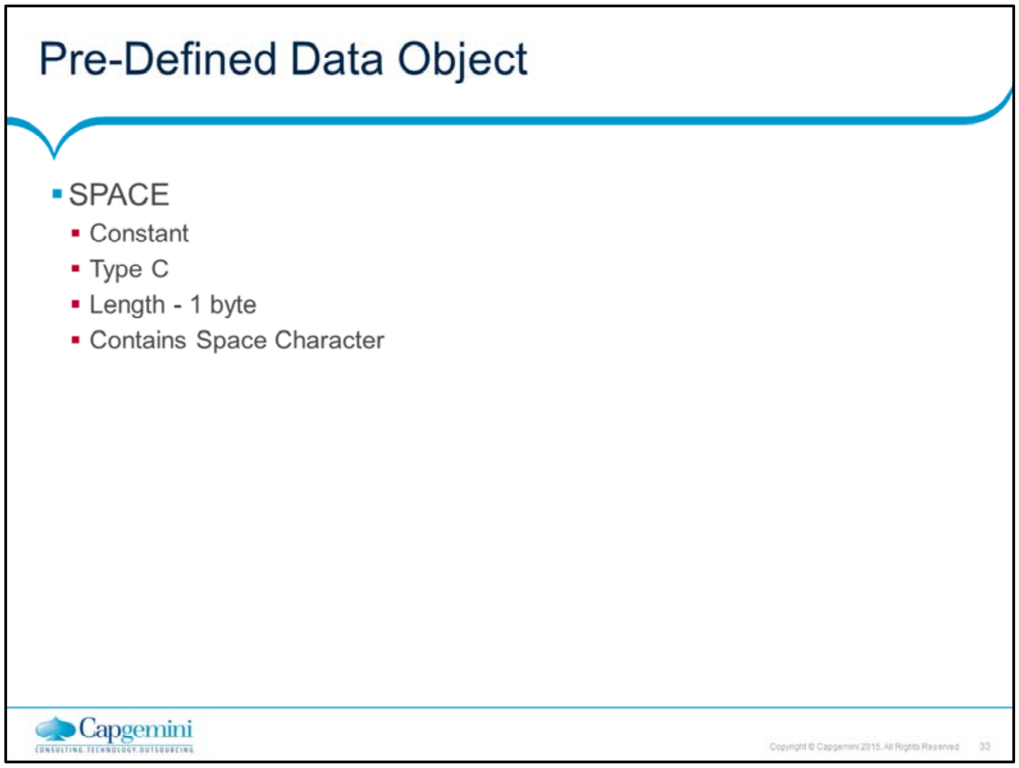
```
CONSTANTS gc_myconst TYPE type_name VALUE {literal | IS INITIAL }.
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING



Demo

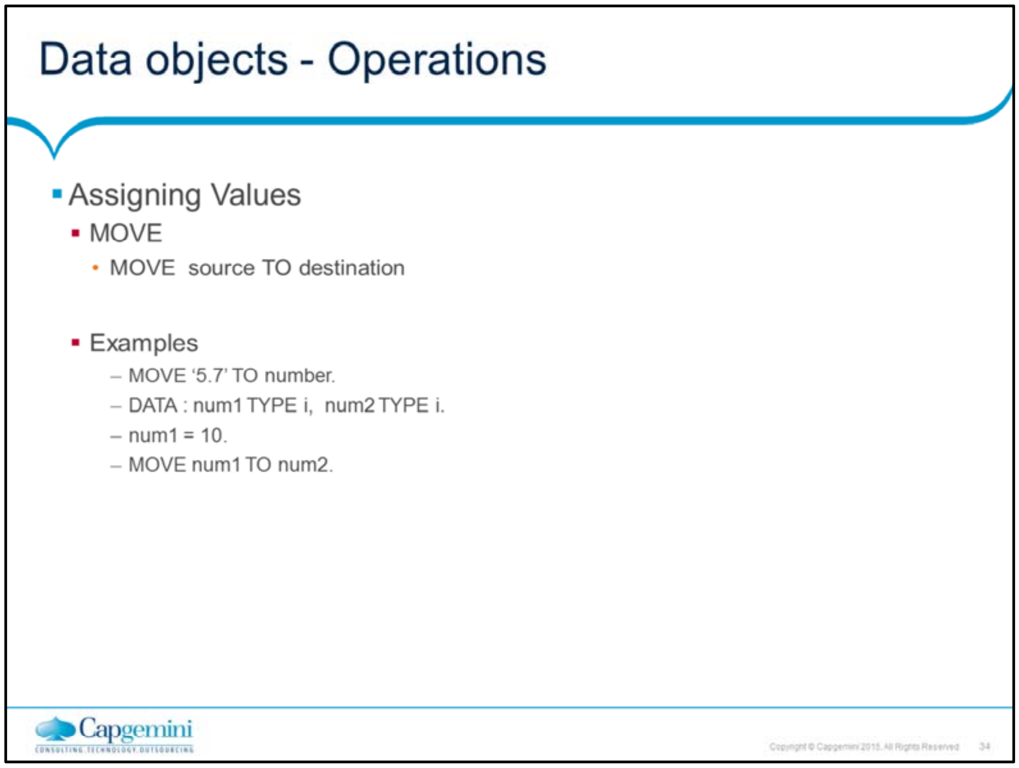
- Program on using Data Objects



Pre-Defined Data Object

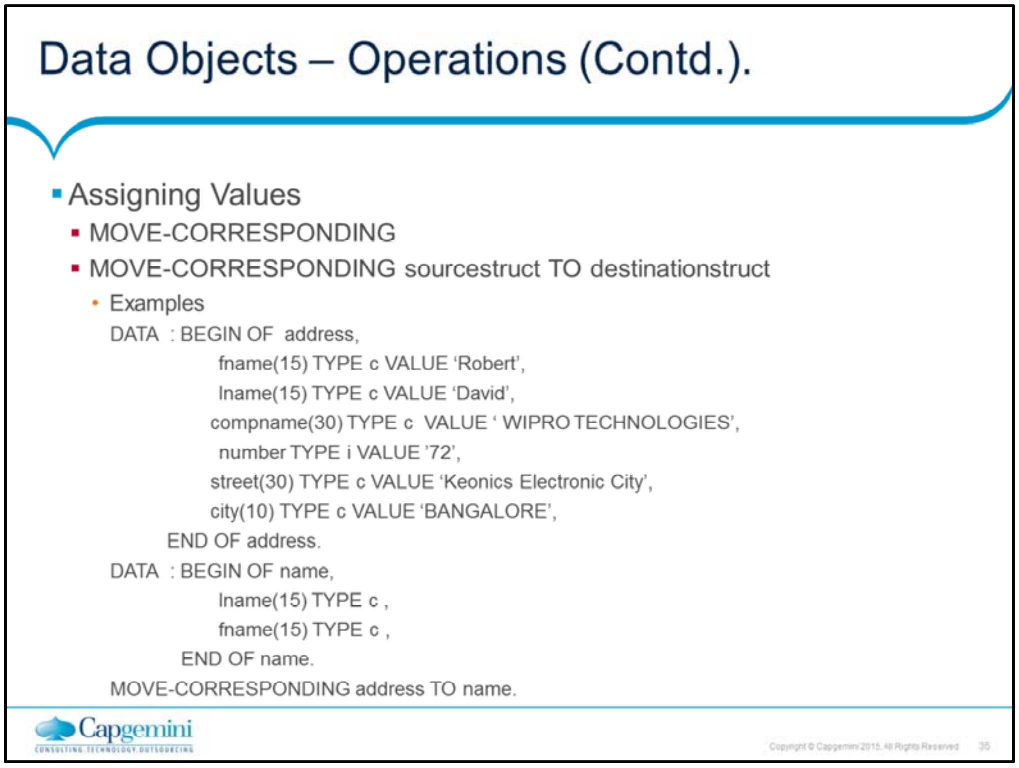
- SPACE
  - Constant
  - Type C
  - Length - 1 byte
  - Contains Space Character

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING



Data objects - Operations

- Assigning Values
  - MOVE
    - MOVE source TO destination
  - Examples
    - MOVE ‘5.7’ TO number.
    - DATA : num1 TYPE i, num2 TYPE i.
    - num1 = 10.
    - MOVE num1 TO num2.

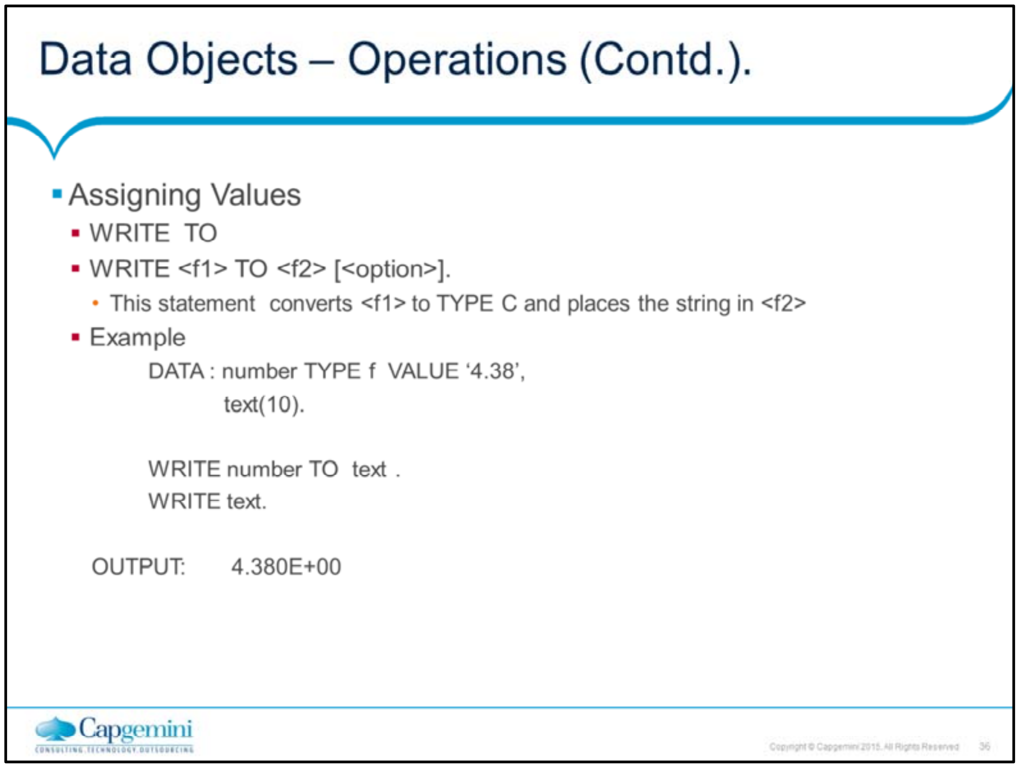


Data Objects – Operations (Contd.).

- Assigning Values
  - MOVE-CORRESPONDING
  - MOVE-CORRESPONDING sourcestruct TO destinationstruct
    - Examples

```
DATA  : BEGIN OF  address,
              fname(15) TYPE c VALUE ‘Robert’,
              lname(15) TYPE c VALUE ‘David’,
             compname(30) TYPE c  VALUE ‘ WIPRO TECHNOLOGIES’,
              number TYPE i VALUE ’72’,
             street(30) TYPE c VALUE ‘Keonics Electronic City’,
             city(10) TYPE c VALUE ‘BANGALORE’,
        END OF address.
DATA  : BEGIN OF name,
              lname(15) TYPE c ,
              fname(15) TYPE c ,
         END OF name.
MOVE-CORRESPONDING address TO name.
```

35

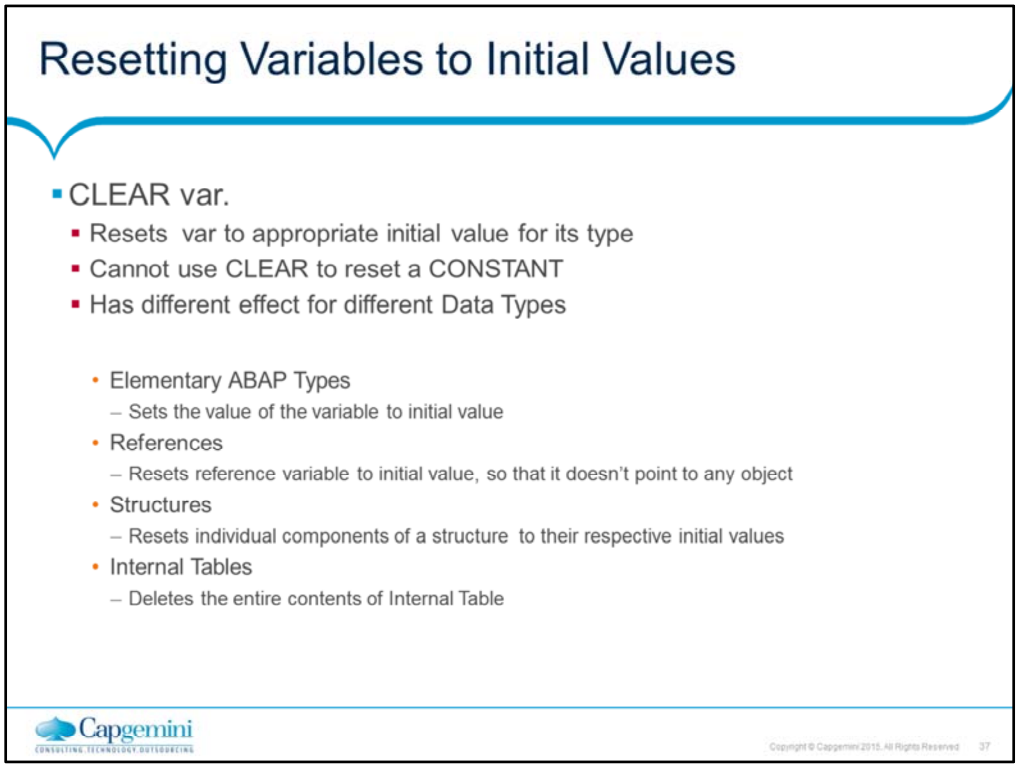


Data Objects – Operations (Contd.).

- Assigning Values
  - WRITE TO
  - WRITE <f1> TO <f2> [<option>].
    - This statement converts <f1> to TYPE C and places the string in <f2>
  - Example

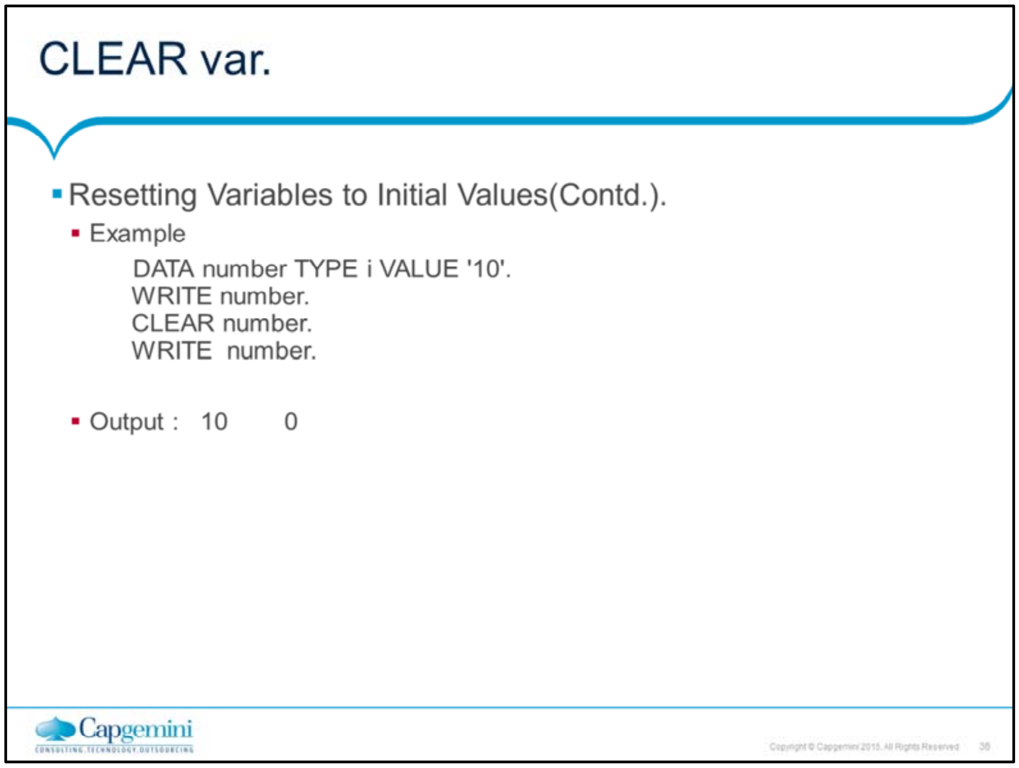
```
DATA : number TYPE f  VALUE '4.38',
         text(10).

WRITE number TO  text .
WRITE text.
```

```
OUTPUT:      4.380E+00
```



Resetting Variables to Initial Values

- CLEAR var.
  - Resets var to appropriate initial value for its type
  - Cannot use CLEAR to reset a CONSTANT
  - Has different effect for different Data Types
    - Elementary ABAP Types
      – Sets the value of the variable to initial value
    - References
      – Resets reference variable to initial value, so that it doesn't point to any object
    - Structures
      – Resets individual components of a structure to their respective initial values
    - Internal Tables
      – Deletes the entire contents of Internal Table



CLEAR var.

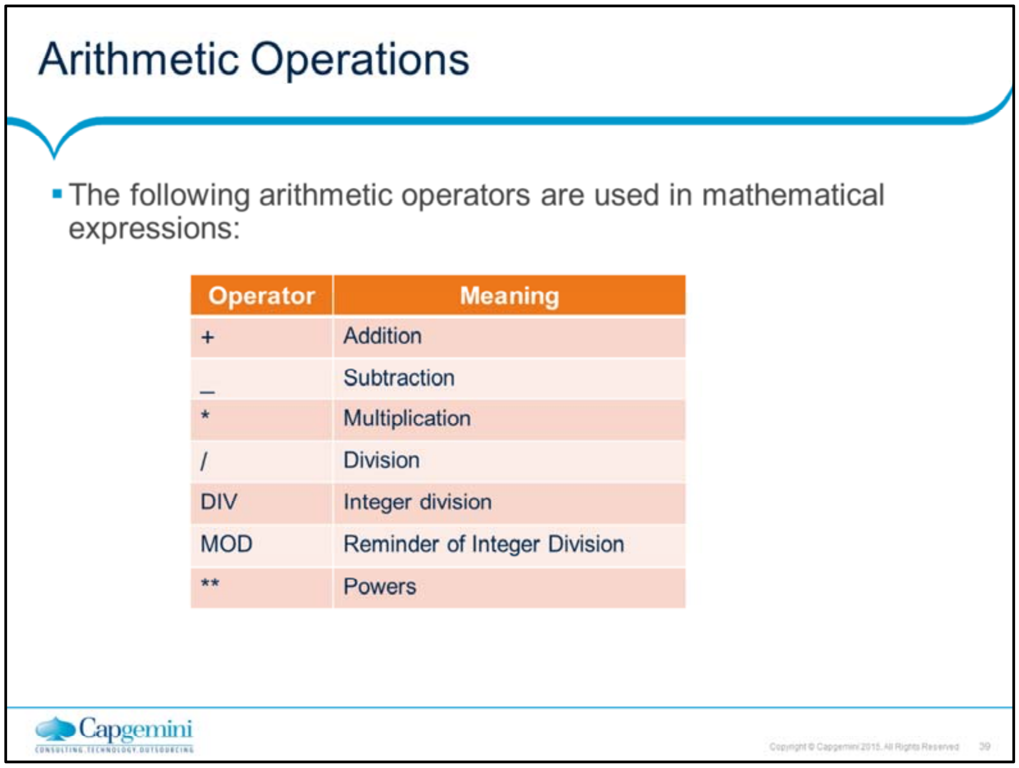
- Resetting Variables to Initial Values(Contd.).
  - Example

```
DATA number TYPE i VALUE '10'.
WRITE number.
CLEAR number.
WRITE  number.
```

  - Output :   10   0

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

38

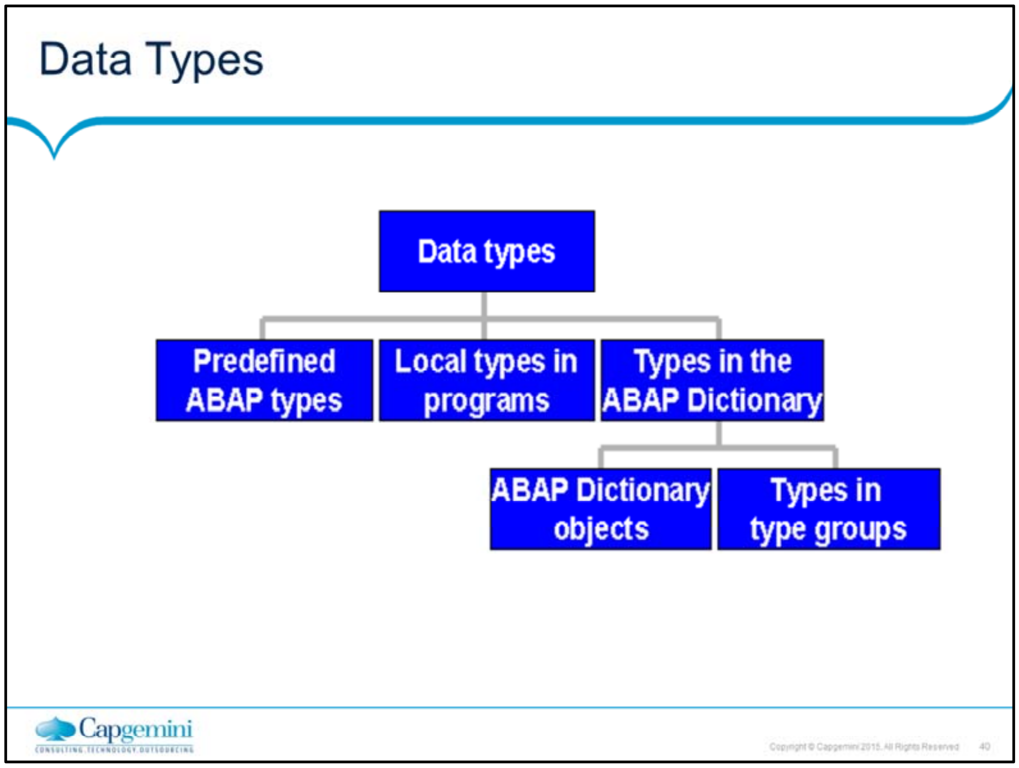


Arithmetic Operations

- The following arithmetic operators are used in mathematical expressions:

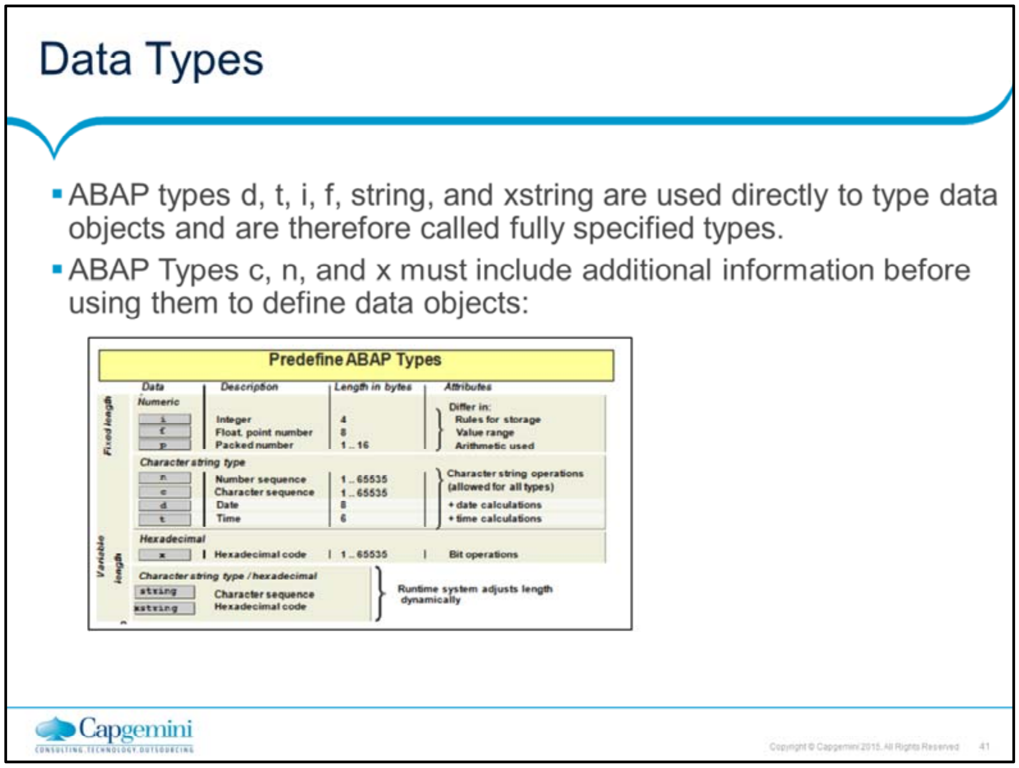
Operator	Meaning
+	Addition
_	Subtraction
*	Multiplication
/	Division
DIV	Integer division
MOD	Reminder of Integer Division
**	Powers

Capgemini CONSULTING.TECHNOLOGY.OUTSOURCING



Data Types

Data types
Predefined ABAP types
Local types in programs
Types in the ABAP Dictionary
ABAP Dictionary objects
Types in type groups



Data Types

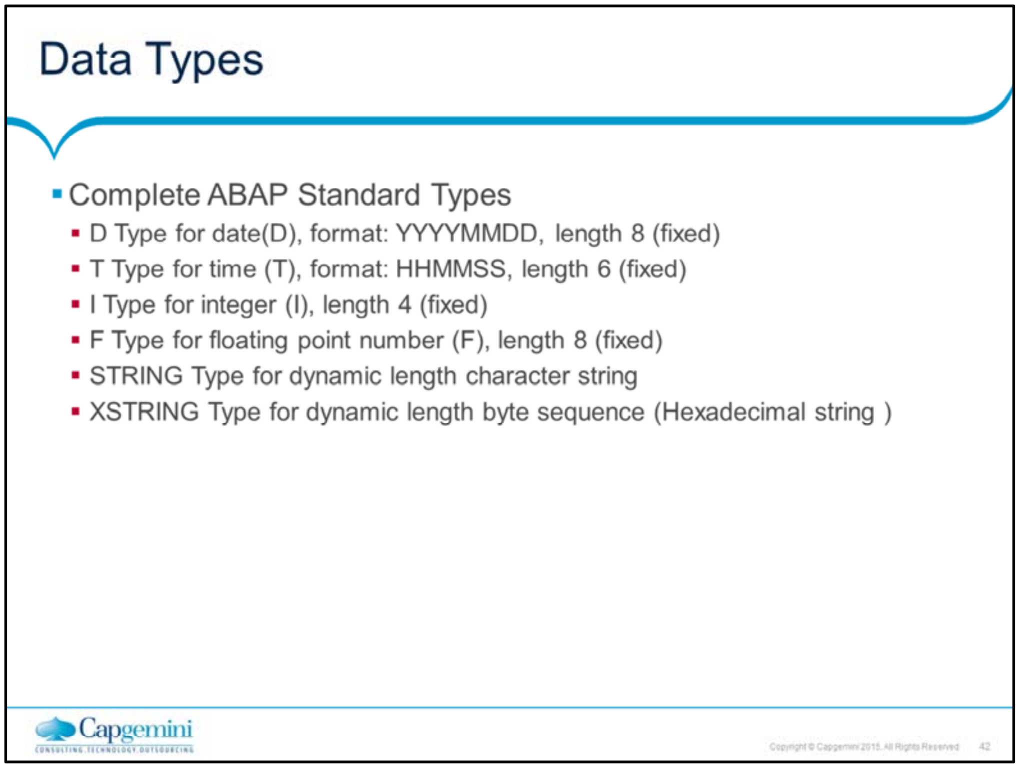
- ABAP types d, t, i, f, string, and xstring are used directly to type data objects and are therefore called fully specified types.
- ABAP Types c, n, and x must include additional information before using them to define data objects:

Predefine ABAP Types

	Data	Description	Length in bytes	Attributes
Fixed length	*Numeric*			Differ in:
	i	Integer	4	Rules for storage
	f	Float. point number	8	Value range
	p	Packed number	1 .. 16	Arithmetic used
	Character string type			
	n	Number sequence	1 .. 65535	Character string operations (allowed for all types)
	c	Character sequence	1 .. 65535	
	d	Date	8	+ date calculations
	t	Time	6	+ time calculations
	Hexadecimal			
	x	Hexadecimal code	1 .. 65535	Bit operations
Variable length	*Character string type / hexadecimal*			Runtime system adjusts length dynamically
	string	Character sequence		
	xstring	Hexadecimal code		

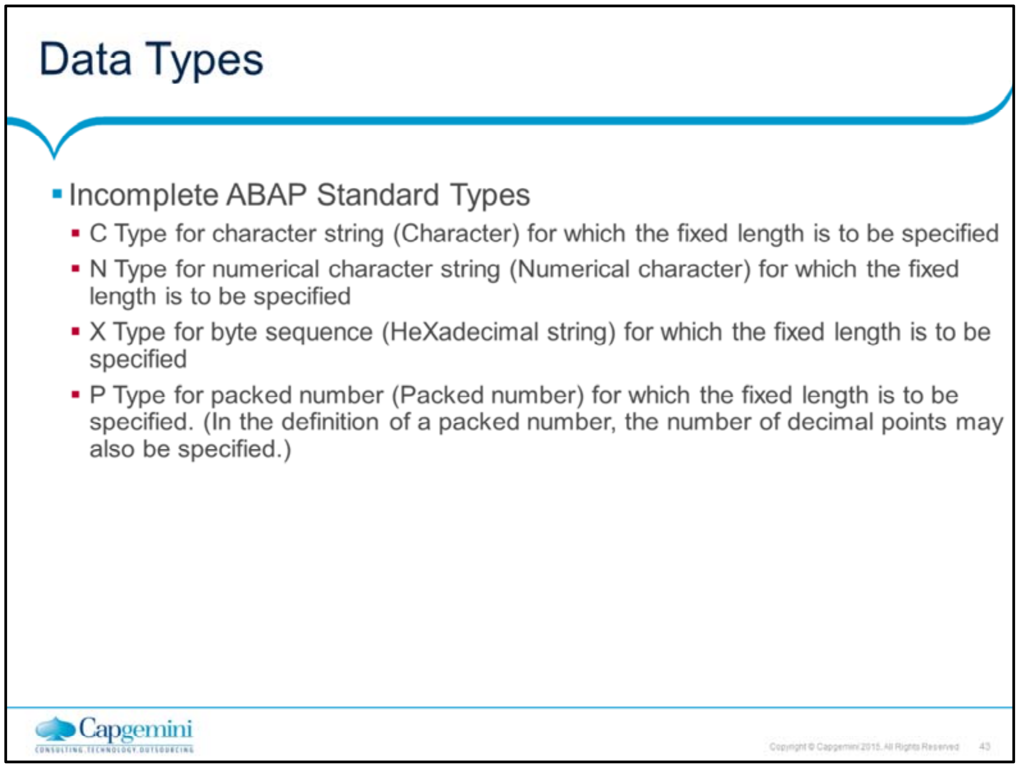
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

41



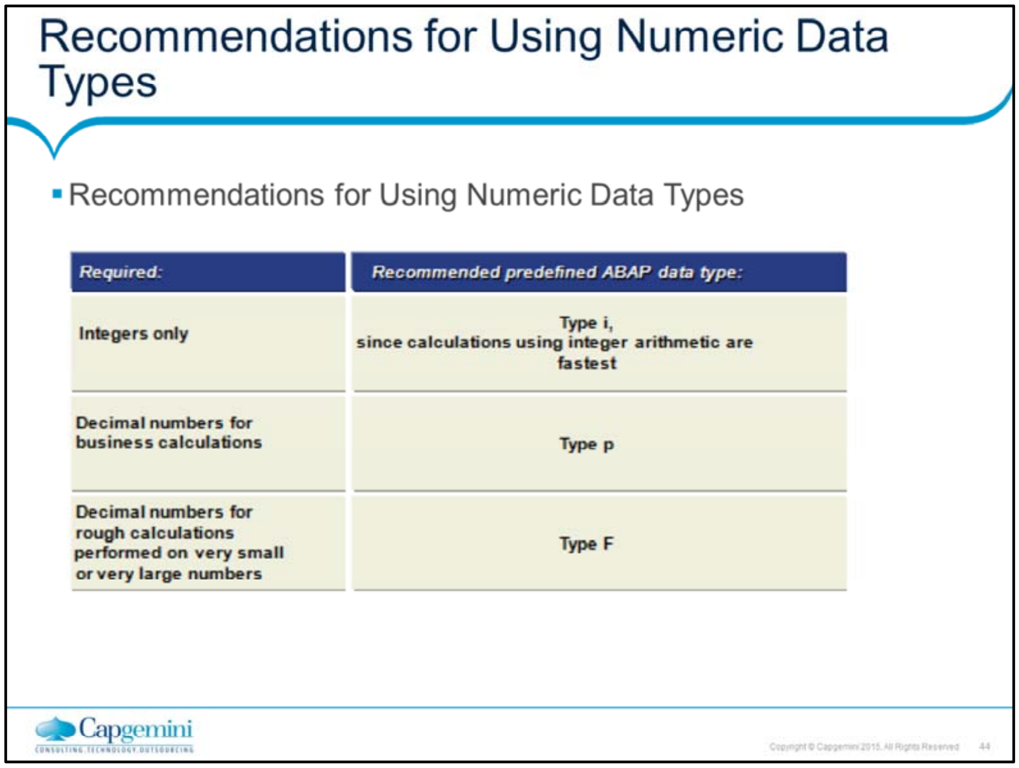
Data Types

- Complete ABAP Standard Types
  - D Type for date(D), format: YYYYMMDD, length 8 (fixed)
  - T Type for time (T), format: HHMMSS, length 6 (fixed)
  - I Type for integer (I), length 4 (fixed)
  - F Type for floating point number (F), length 8 (fixed)
  - STRING Type for dynamic length character string
  - XSTRING Type for dynamic length byte sequence (Hexadecimal string)



Data Types

- Incomplete ABAP Standard Types
  - C Type for character string (Character) for which the fixed length is to be specified
  - N Type for numerical character string (Numerical character) for which the fixed length is to be specified
  - X Type for byte sequence (HeXadecimal string) for which the fixed length is to be specified
  - P Type for packed number (Packed number) for which the fixed length is to be specified. (In the definition of a packed number, the number of decimal points may also be specified.)

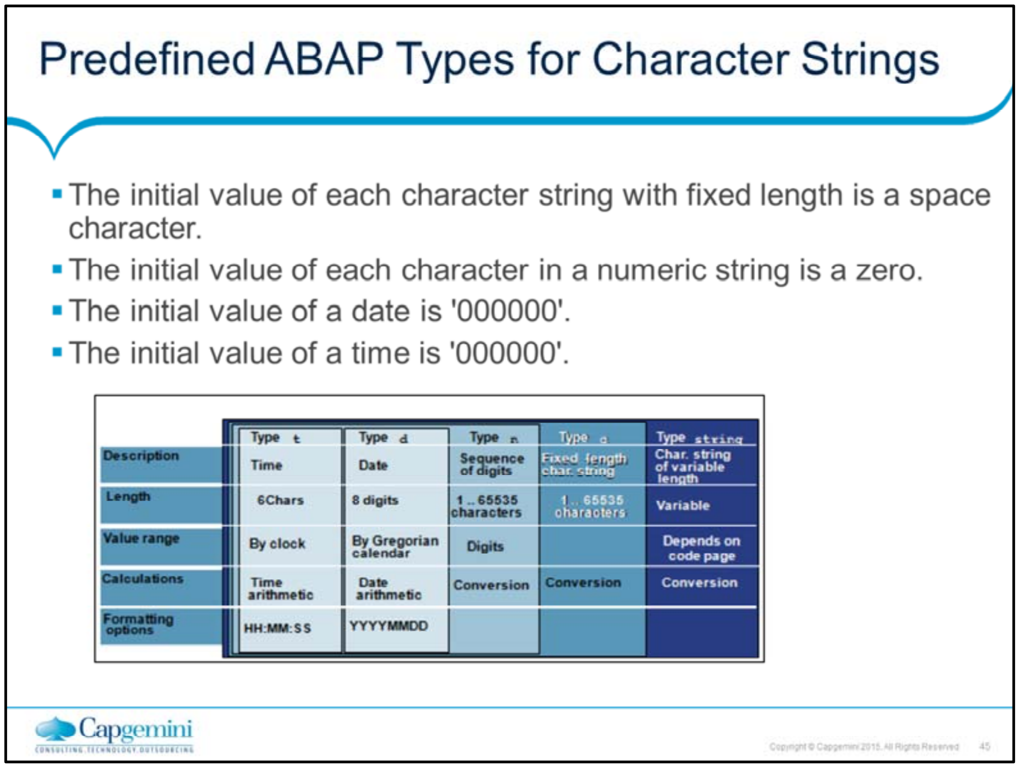


Recommendations for Using Numeric Data Types

- Recommendations for Using Numeric Data Types

Required:	*Recommended predefined ABAP data type:*
Integers only	Type i, since calculations using integer arithmetic are fastest
Decimal numbers for business calculations	Type p
Decimal numbers for rough calculations performed on very small or very large numbers	Type F

44

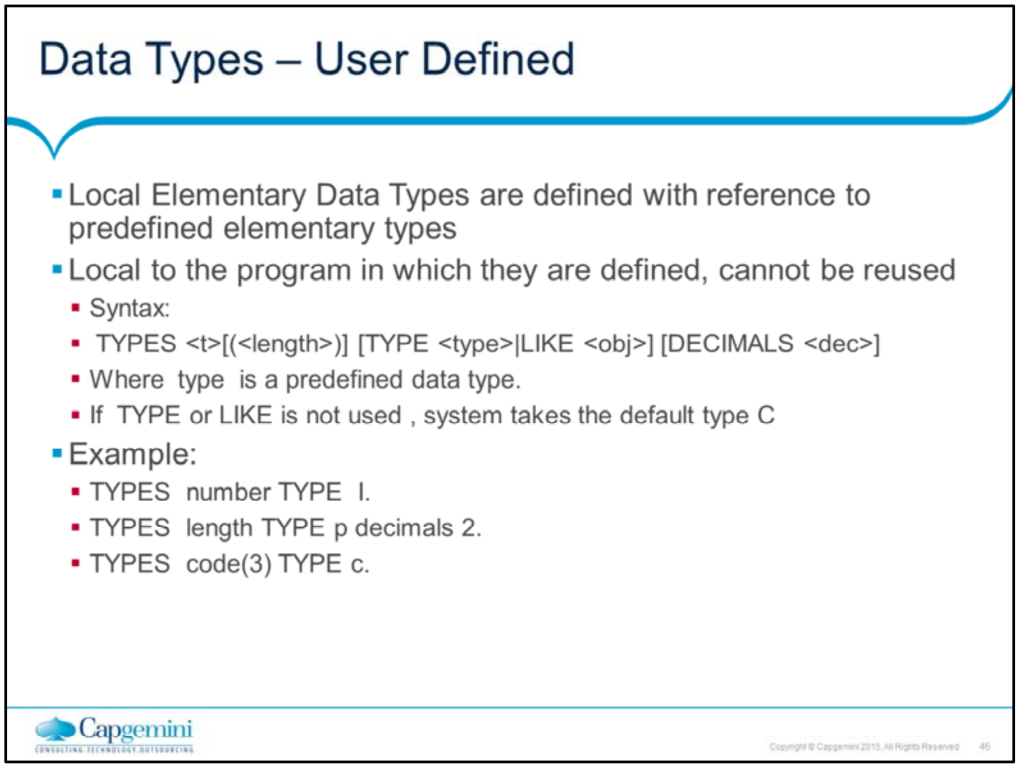


Predefined ABAP Types for Character Strings

- The initial value of each character string with fixed length is a space character.
- The initial value of each character in a numeric string is a zero.
- The initial value of a date is '000000'.
- The initial value of a time is '000000'.

	Type t	Type d	Type n	Type c	Type string
Description	Time	Date	Sequence of digits	Fixed length char. string	Char. string of variable length
Length	6Chars	8 digits	1 .. 65535 characters	1 .. 65535 characters	Variable
Value range	By clock	By Gregorian calendar	Digits		Depends on code page
Calculations	Time arithmetic	Date arithmetic	Conversion	Conversion	Conversion
Formatting options	HH:MM:SS	YYYYMMDD			

45

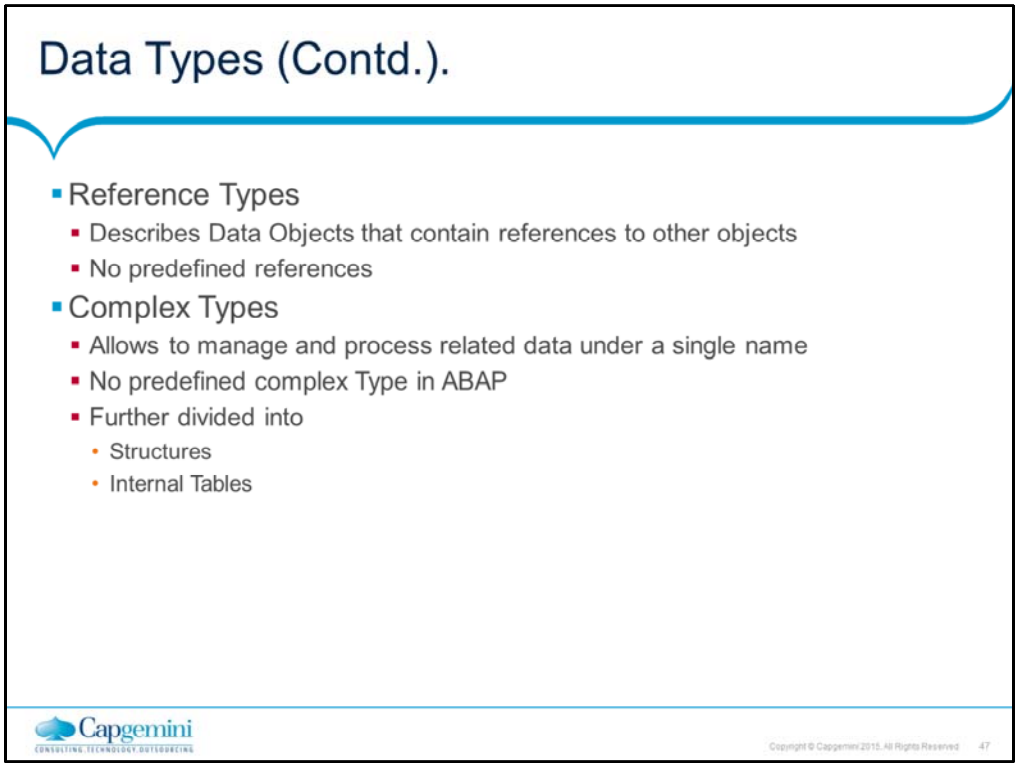


Data Types – User Defined

- Local Elementary Data Types are defined with reference to predefined elementary types
- Local to the program in which they are defined, cannot be reused
  - Syntax:
  - TYPES <t>[(<length>)] [TYPE <type>|LIKE <obj>] [DECIMALS <dec>]
  - Where type is a predefined data type.
  - If TYPE or LIKE is not used , system takes the default type C
- Example:
  - TYPES number TYPE I.
  - TYPES length TYPE p decimals 2.
  - TYPES code(3) TYPE c.

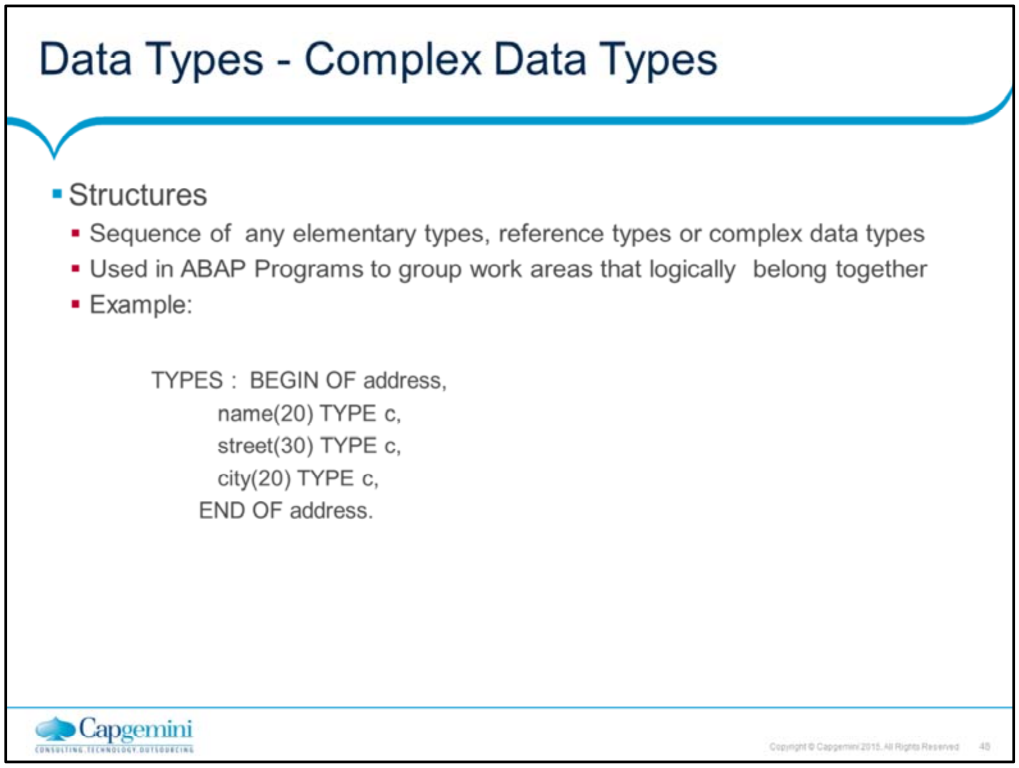
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

46



Data Types (Contd.).

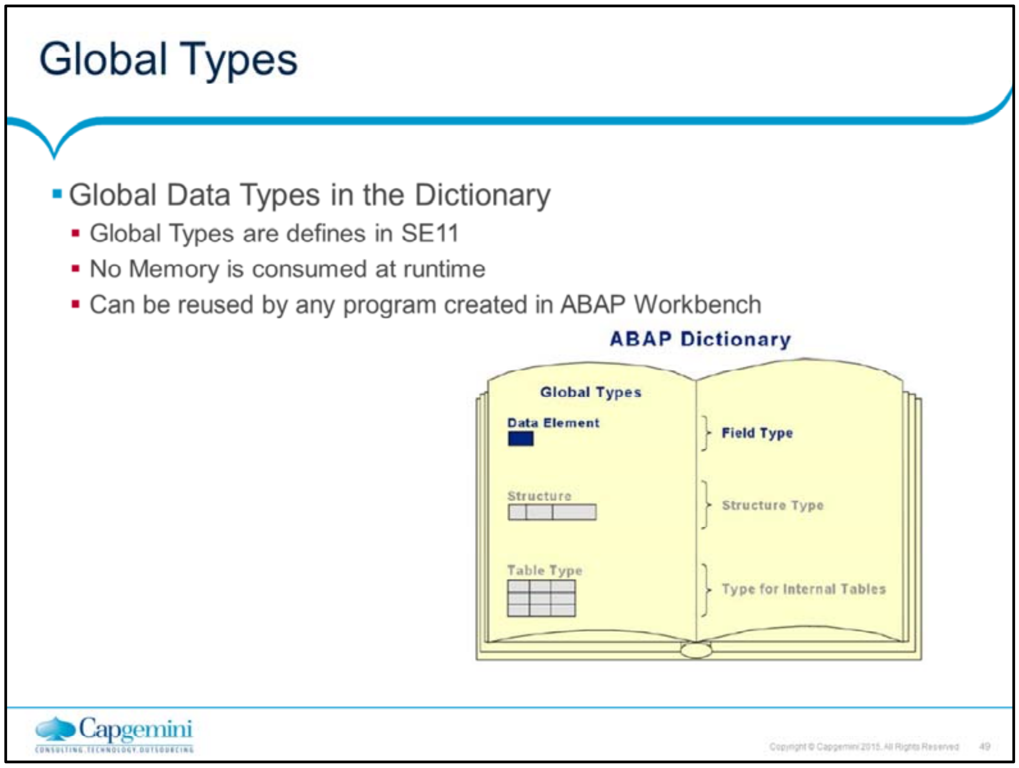
- Reference Types
  - Describes Data Objects that contain references to other objects
  - No predefined references
- Complex Types
  - Allows to manage and process related data under a single name
  - No predefined complex Type in ABAP
  - Further divided into
    - Structures
    - Internal Tables



Data Types - Complex Data Types

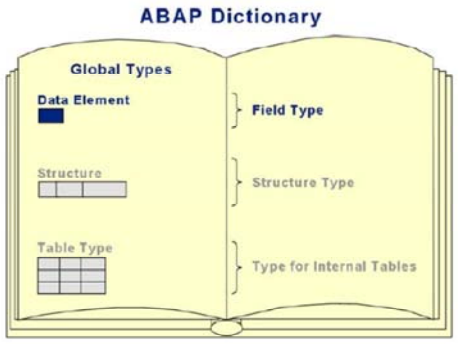
- Structures
  - Sequence of any elementary types, reference types or complex data types
  - Used in ABAP Programs to group work areas that logically belong together
  - Example:

```
TYPES : BEGIN OF address,
        name(20) TYPE c,
        street(30) TYPE c,
        city(20) TYPE c,
      END OF address.
```

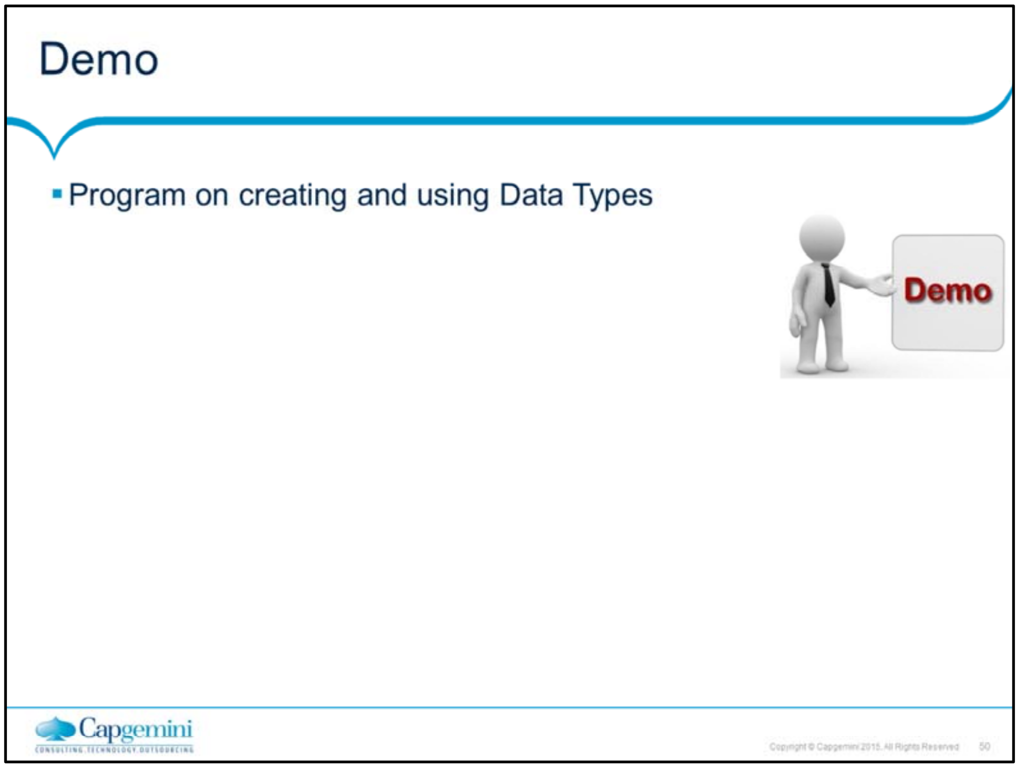


Global Types

- Global Data Types in the Dictionary
  - Global Types are defines in SE11
  - No Memory is consumed at runtime
  - Can be reused by any program created in ABAP Workbench



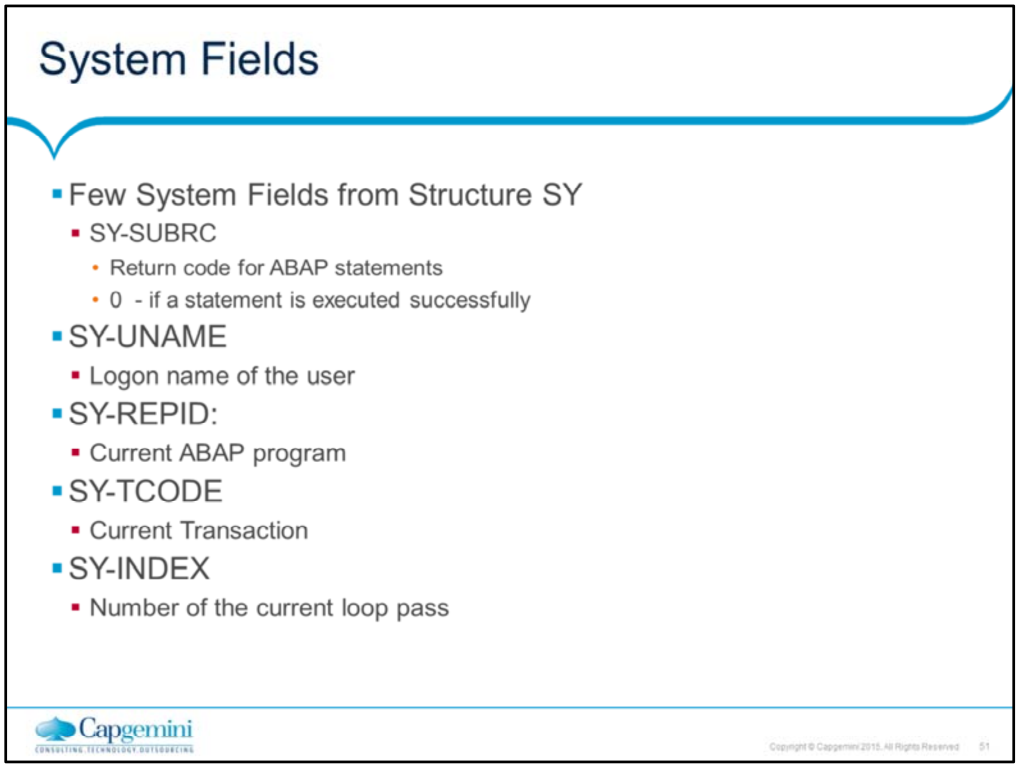
49



Demo

- Program on creating and using Data Types

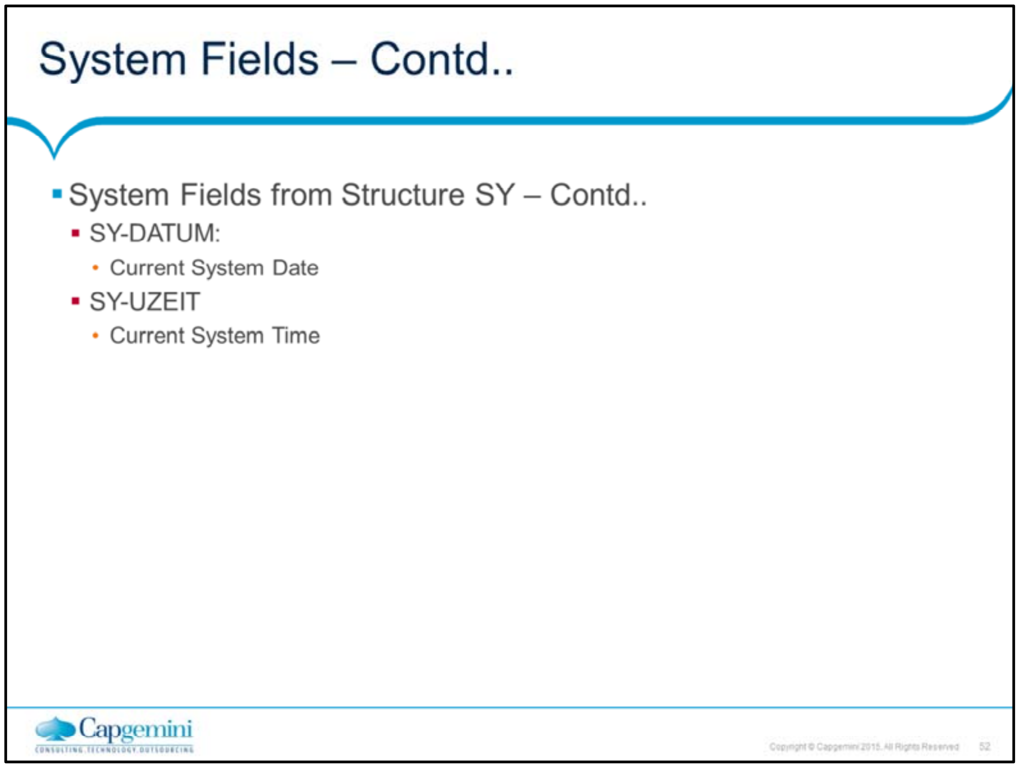
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING



System Fields

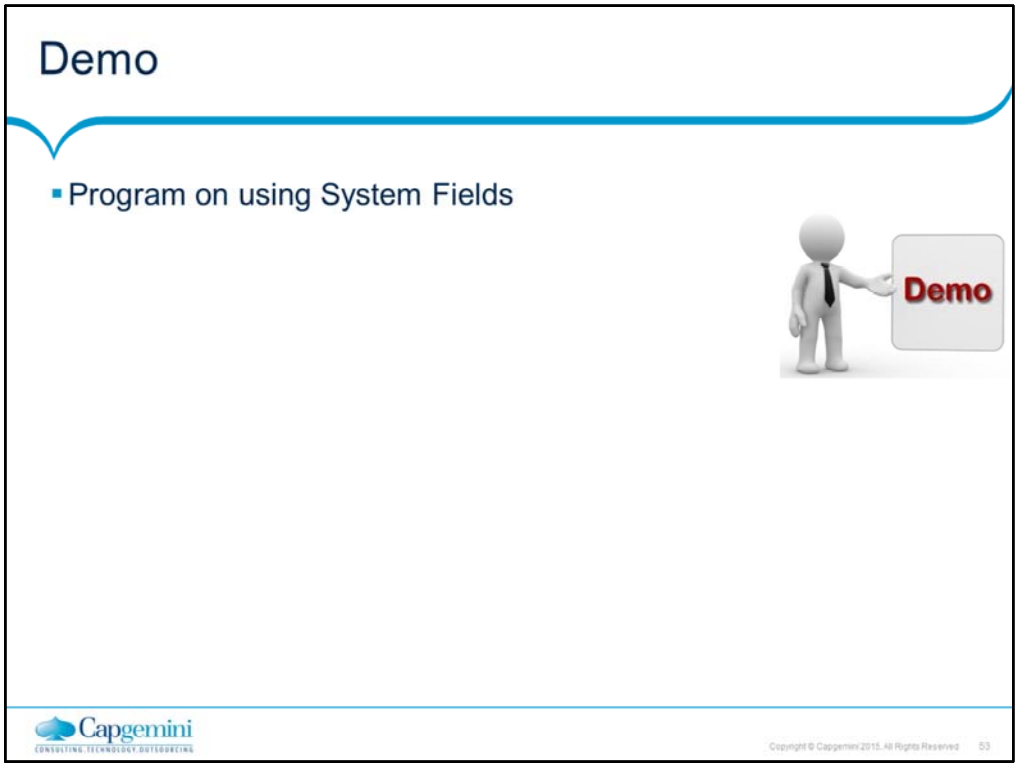
- Few System Fields from Structure SY
  - SY-SUBRC
    - Return code for ABAP statements
    - 0 - if a statement is executed successfully
- SY-UNAME
  - Logon name of the user
- SY-REPID:
  - Current ABAP program
- SY-TCODE
  - Current Transaction
- SY-INDEX
  - Number of the current loop pass

Capgemini CONSULTING.TECHNOLOGY.OUTSOURCING



System Fields – Contd..

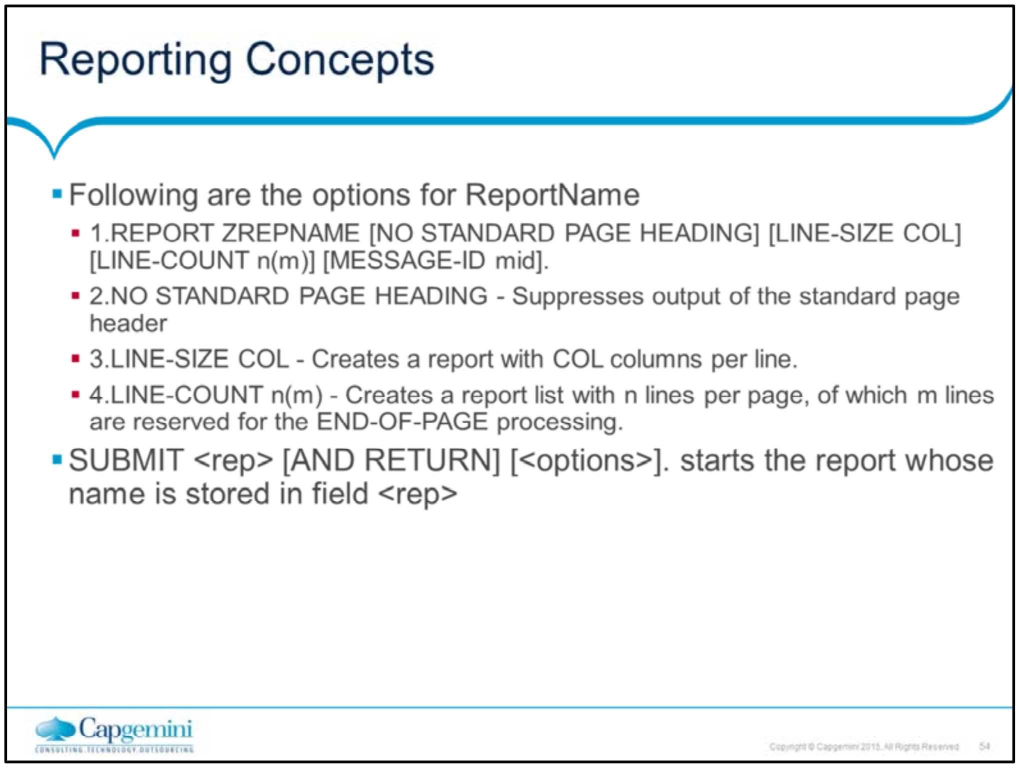
- System Fields from Structure SY – Contd..
  - SY-DATUM:
    - Current System Date
  - SY-UZEIT
    - Current System Time



Demo

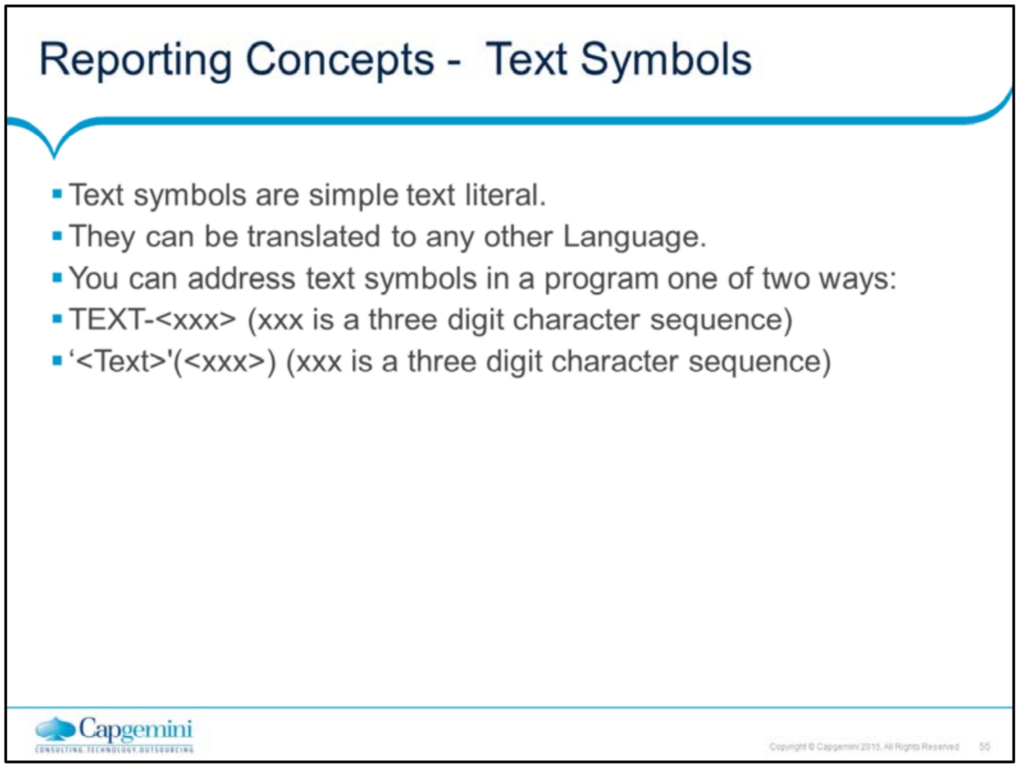
- Program on using System Fields

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING



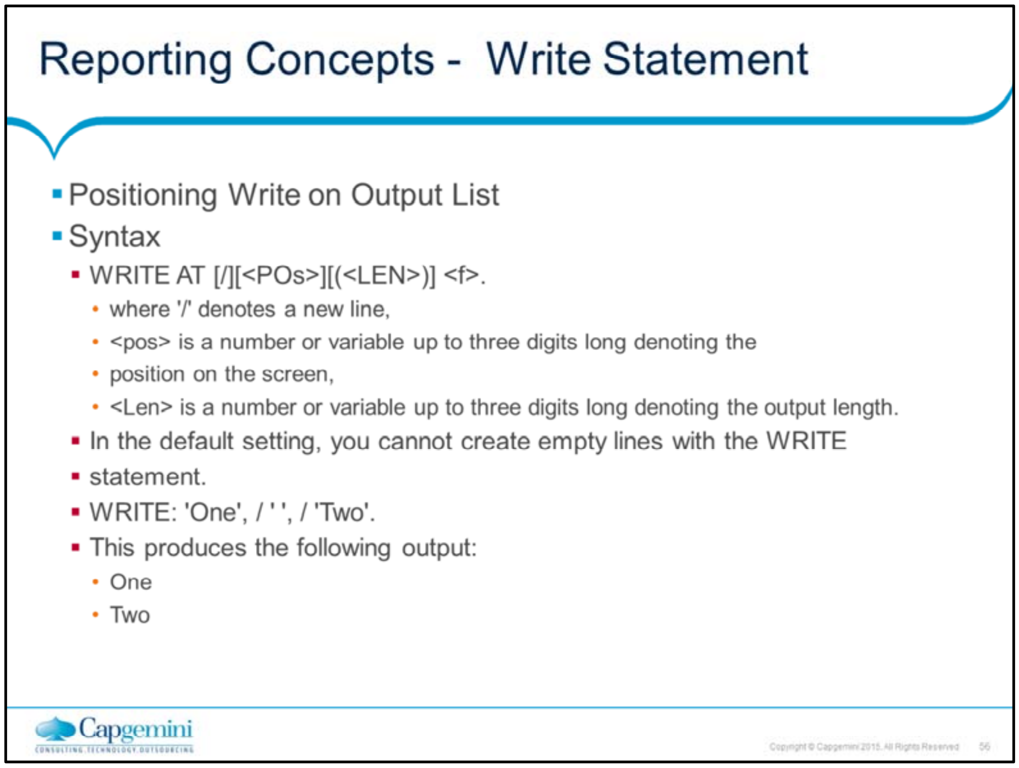
Reporting Concepts

- Following are the options for ReportName
  - 1.REPORT ZREPNAME [NO STANDARD PAGE HEADING] [LINE-SIZE COL] [LINE-COUNT n(m)] [MESSAGE-ID mid].
  - 2.NO STANDARD PAGE HEADING - Suppresses output of the standard page header
  - 3.LINE-SIZE COL - Creates a report with COL columns per line.
  - 4.LINE-COUNT n(m) - Creates a report list with n lines per page, of which m lines are reserved for the END-OF-PAGE processing.
- SUBMIT <rep> [AND RETURN] [<options>]. starts the report whose name is stored in field <rep>



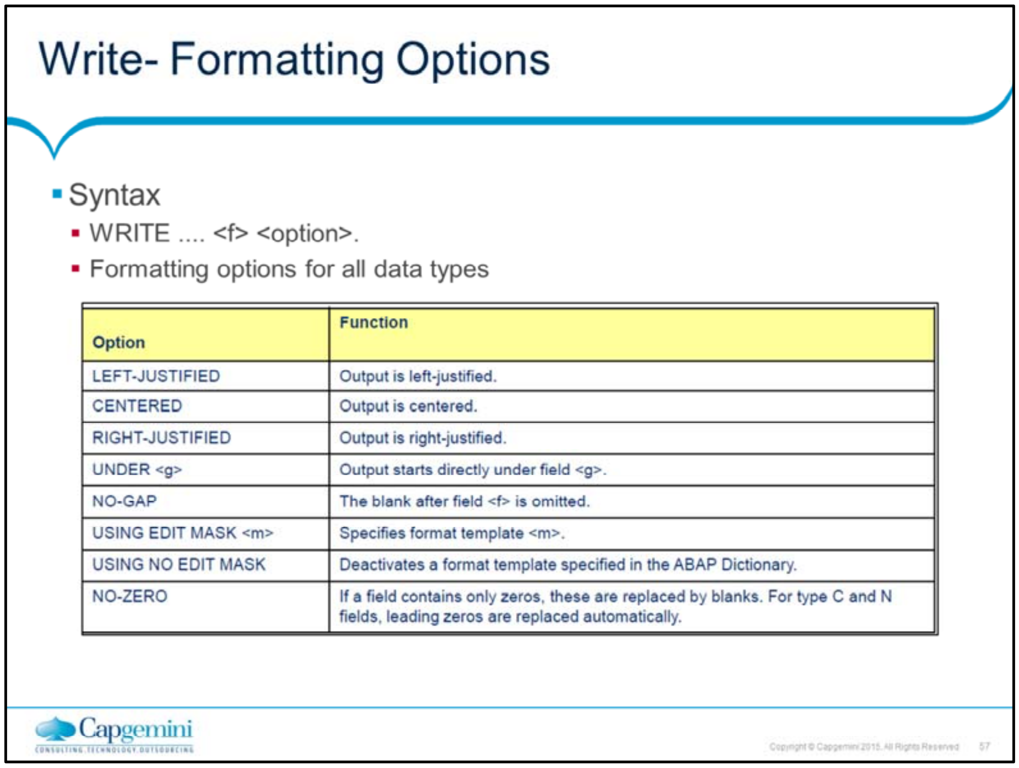
Reporting Concepts - Text Symbols

- Text symbols are simple text literal.
- They can be translated to any other Language.
- You can address text symbols in a program one of two ways:
- TEXT-<xxx> (xxx is a three digit character sequence)
- ‘<Text>'(<xxx>) (xxx is a three digit character sequence)



Reporting Concepts - Write Statement

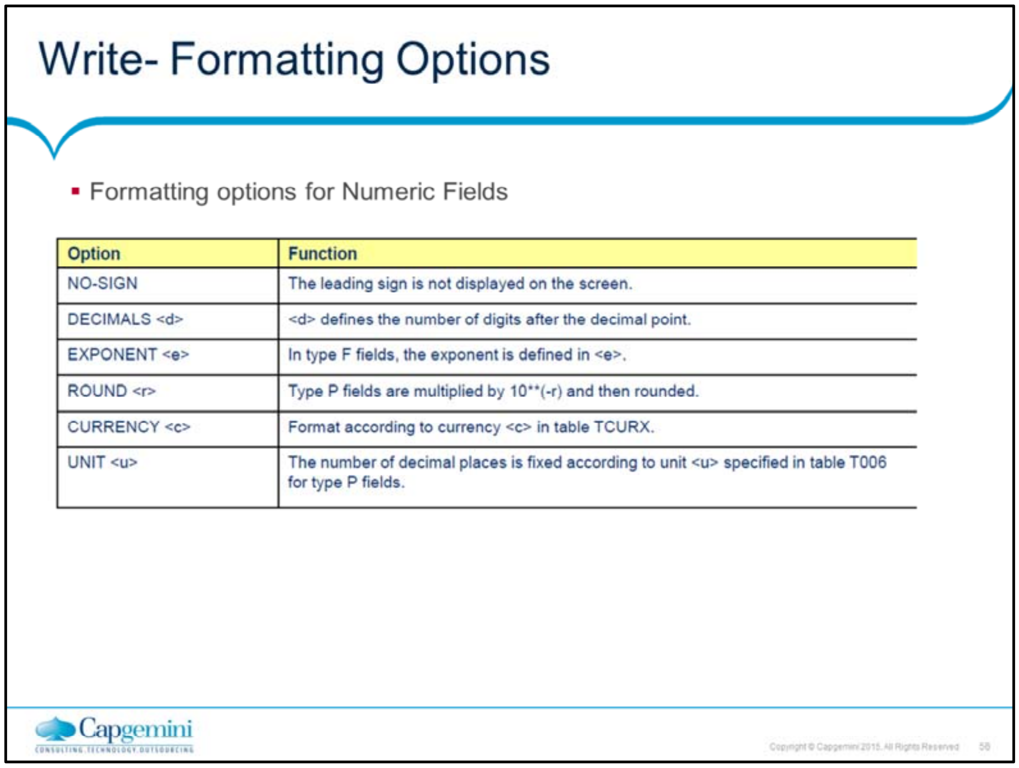
- Positioning Write on Output List
- Syntax
  - WRITE AT [/][<POs>][(<LEN>)] <f>.
    - where '/' denotes a new line,
    - <pos> is a number or variable up to three digits long denoting the
    - position on the screen,
    - <Len> is a number or variable up to three digits long denoting the output length.
  - In the default setting, you cannot create empty lines with the WRITE
  - statement.
  - WRITE: 'One', / ' ', / 'Two'.
  - This produces the following output:
    - One
    - Two



Write- Formatting Options

- Syntax
  - WRITE .... <f> <option>.
  - Formatting options for all data types

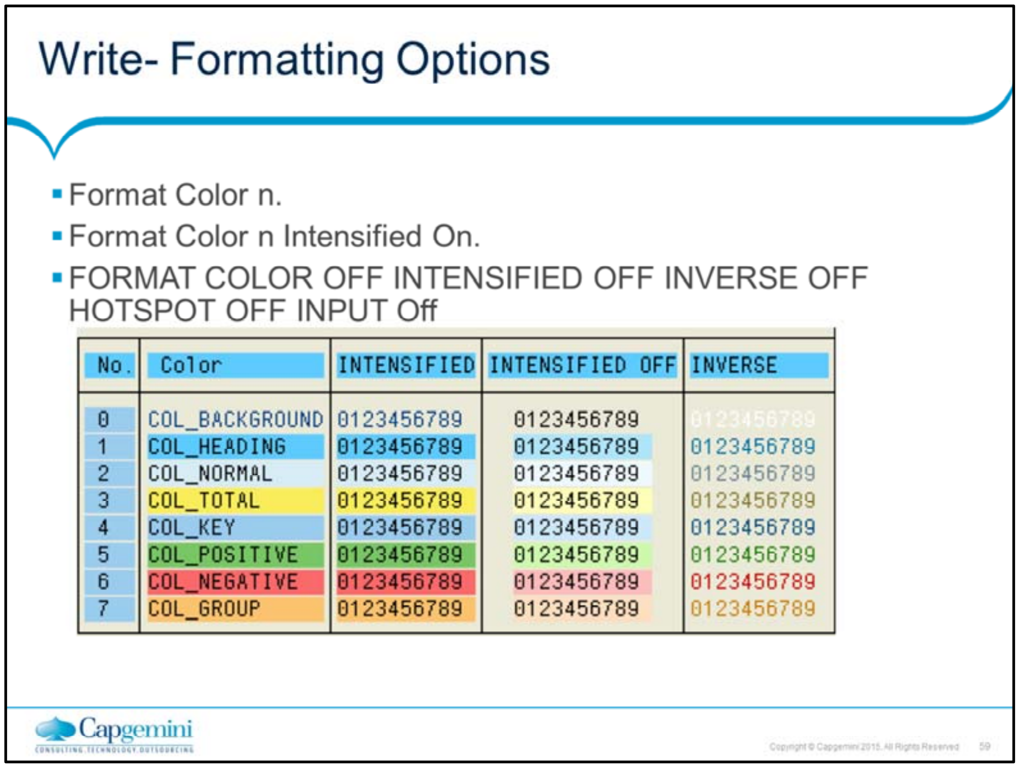
Option	Function
LEFT-JUSTIFIED	Output is left-justified.
CENTERED	Output is centered.
RIGHT-JUSTIFIED	Output is right-justified.
UNDER <g>	Output starts directly under field <g>.
NO-GAP	The blank after field <f> is omitted.
USING EDIT MASK <m>	Specifies format template <m>.
USING NO EDIT MASK	Deactivates a format template specified in the ABAP Dictionary.
NO-ZERO	If a field contains only zeros, these are replaced by blanks. For type C and N fields, leading zeros are replaced automatically.



Write- Formatting Options

- Formatting options for Numeric Fields

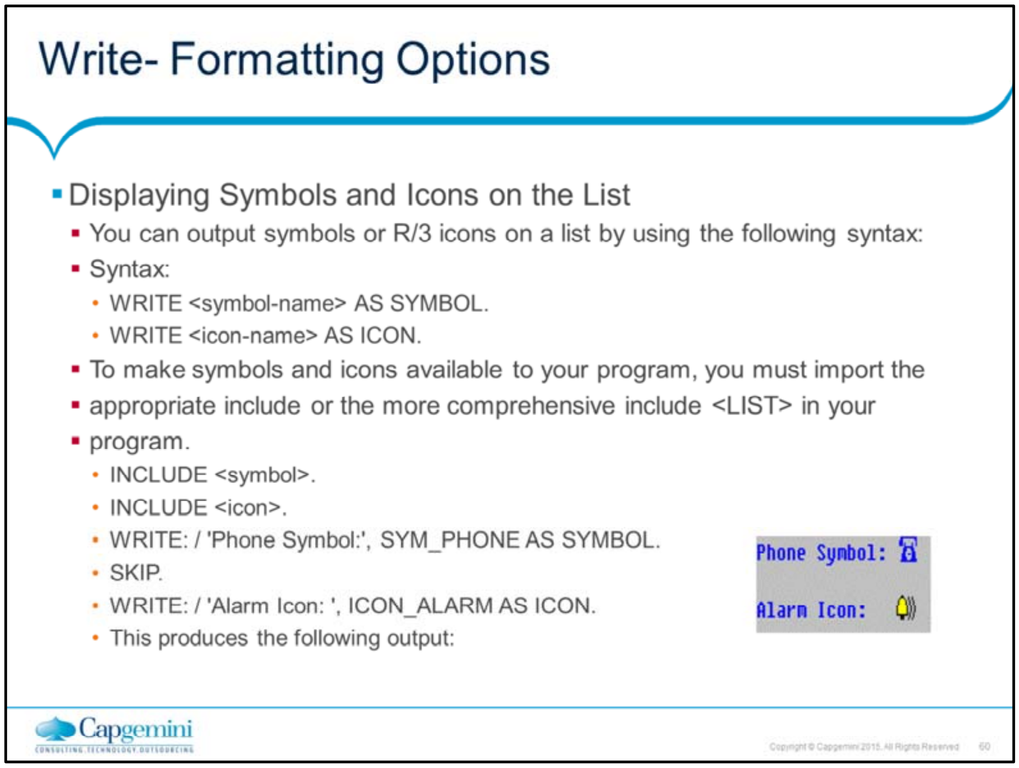
Option	Function
NO-SIGN	The leading sign is not displayed on the screen.
DECIMALS <d>	<d> defines the number of digits after the decimal point.
EXPONENT <e>	In type F fields, the exponent is defined in <e>.
ROUND <r>	Type P fields are multiplied by 10**(-r) and then rounded.
CURRENCY <c>	Format according to currency <c> in table TCURX.
UNIT <u>	The number of decimal places is fixed according to unit <u> specified in table T006 for type P fields.



Write- Formatting Options

- Format Color n.
- Format Color n Intensified On.
- FORMAT COLOR OFF INTENSIFIED OFF INVERSE OFF HOTSPOT OFF INPUT Off

No.	Color	INTENSIFIED	INTENSIFIED OFF	INVERSE
0	COL_BACKGROUND	0123456789	0123456789	0123456789
1	COL_HEADING	0123456789	0123456789	0123456789
2	COL_NORMAL	0123456789	0123456789	0123456789
3	COL_TOTAL	0123456789	0123456789	0123456789
4	COL_KEY	0123456789	0123456789	0123456789
5	COL_POSITIVE	0123456789	0123456789	0123456789
6	COL_NEGATIVE	0123456789	0123456789	0123456789
7	COL_GROUP	0123456789	0123456789	0123456789

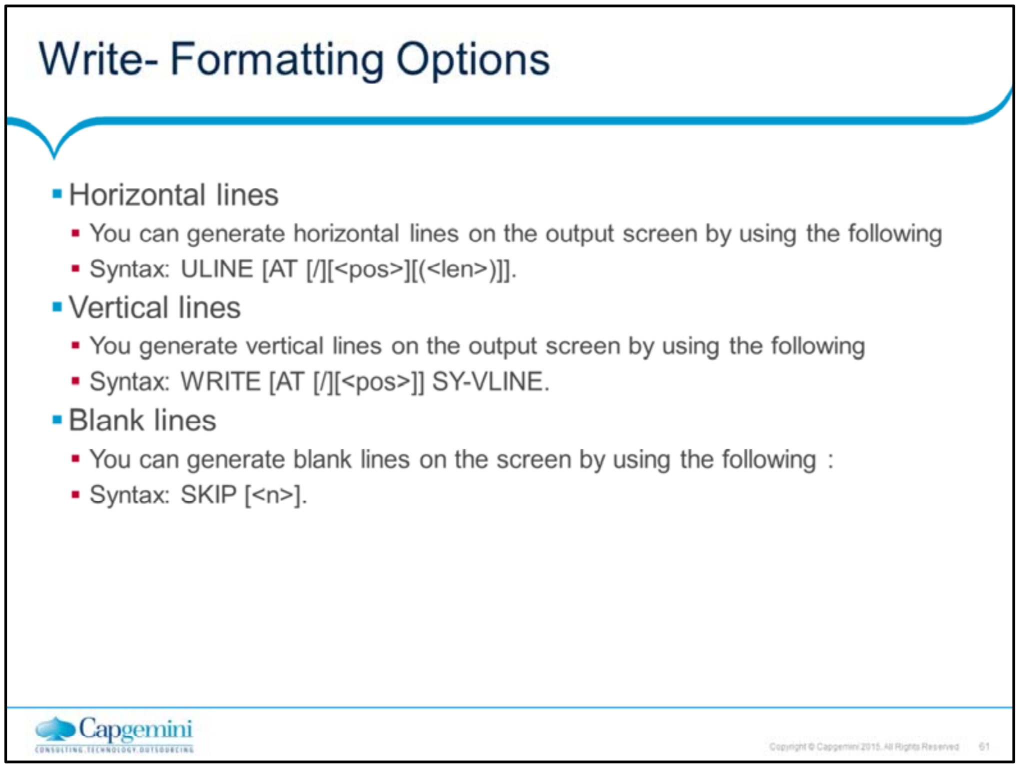


Write- Formatting Options

- Displaying Symbols and Icons on the List
  - You can output symbols or R/3 icons on a list by using the following syntax:
  - Syntax:
    - WRITE <symbol-name> AS SYMBOL.
    - WRITE <icon-name> AS ICON.
  - To make symbols and icons available to your program, you must import the
  - appropriate include or the more comprehensive include <LIST> in your
  - program.
    - INCLUDE <symbol>.
    - INCLUDE <icon>.
    - WRITE: / 'Phone Symbol:', SYM_PHONE AS SYMBOL.
    - SKIP.
    - WRITE: / 'Alarm Icon: ', ICON_ALARM AS ICON.
    - This produces the following output:

```
Phone Symbol:

Alarm Icon:
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

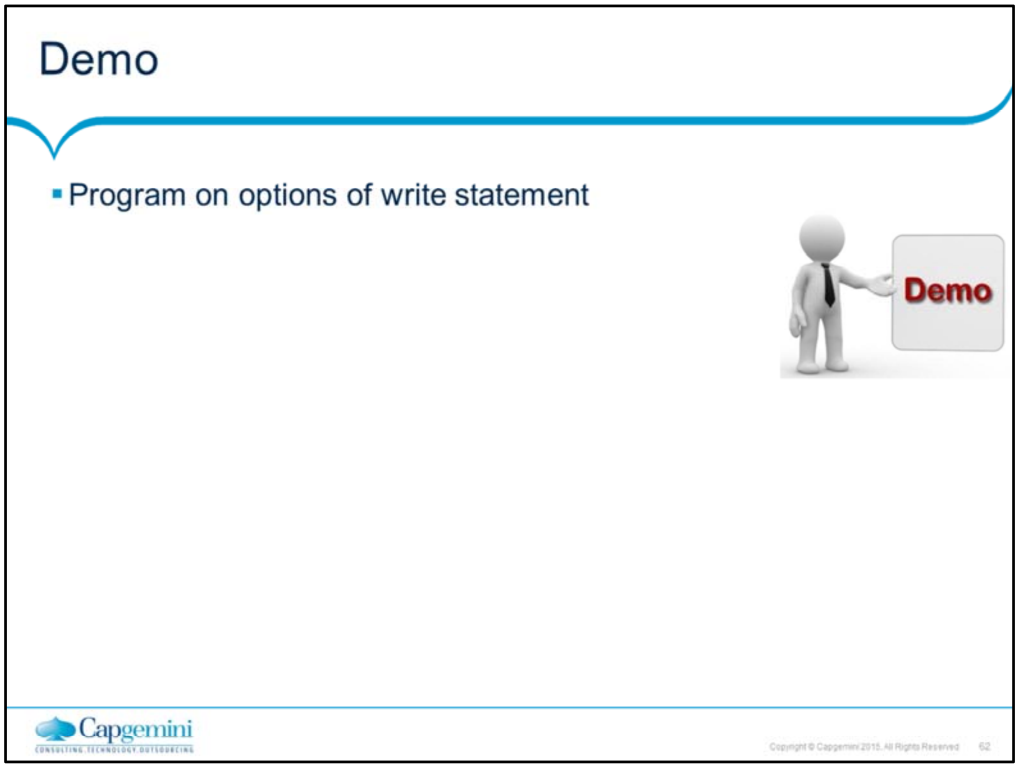


Write- Formatting Options

- Horizontal lines
  - You can generate horizontal lines on the output screen by using the following
  - Syntax: ULINE [AT [/][<pos>][(<len>)]].
- Vertical lines
  - You generate vertical lines on the output screen by using the following
  - Syntax: WRITE [AT [/][<pos>]] SY-VLINE.
- Blank lines
  - You can generate blank lines on the screen by using the following :
  - Syntax: SKIP [<n>].

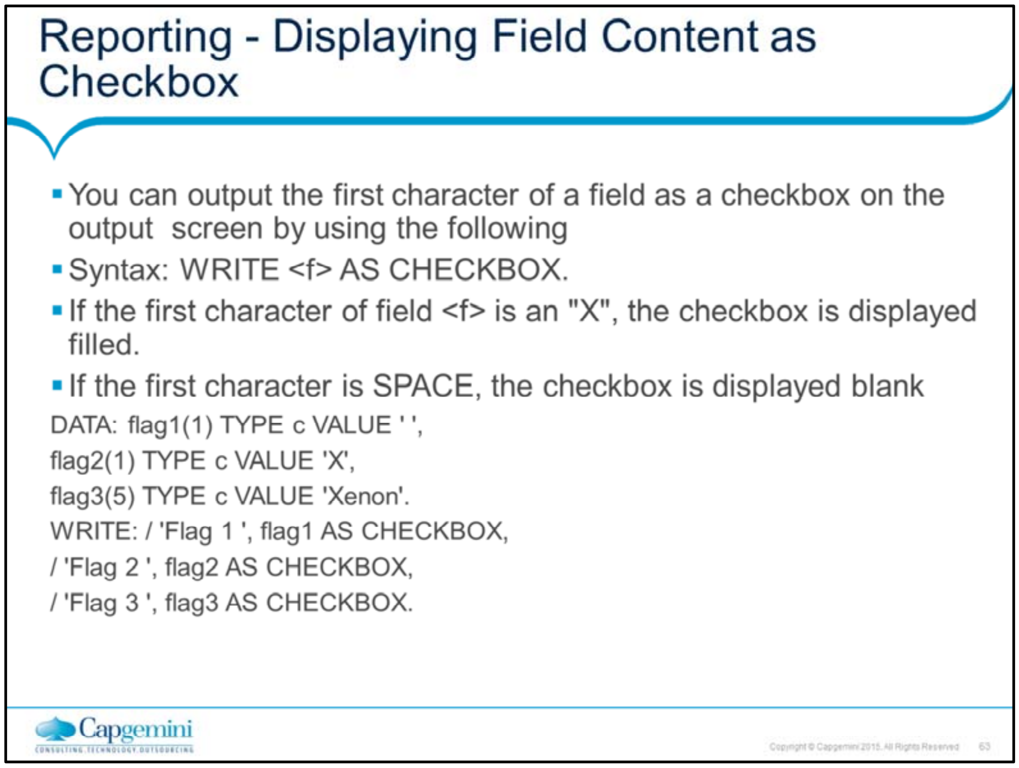
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

61



Demo

- Program on options of write statement

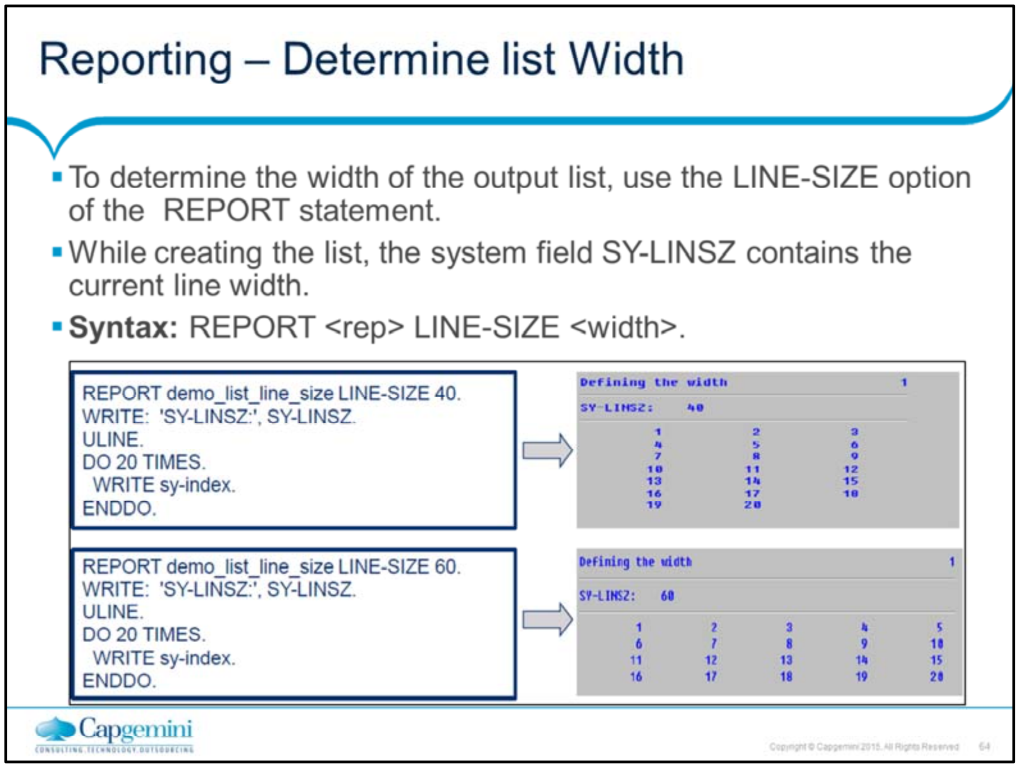


Reporting - Displaying Field Content as Checkbox

- You can output the first character of a field as a checkbox on the output screen by using the following
- Syntax: WRITE <f> AS CHECKBOX.
- If the first character of field <f> is an "X", the checkbox is displayed filled.
- If the first character is SPACE, the checkbox is displayed blank

```
DATA: flag1(1) TYPE c VALUE ' ',
flag2(1) TYPE c VALUE 'X',
flag3(5) TYPE c VALUE 'Xenon'.
WRITE: / 'Flag 1 ', flag1 AS CHECKBOX,
/ 'Flag 2 ', flag2 AS CHECKBOX,
/ 'Flag 3 ', flag3 AS CHECKBOX.
```

63



Reporting – Determine list Width

- To determine the width of the output list, use the LINE-SIZE option of the REPORT statement.
- While creating the list, the system field SY-LINSZ contains the current line width.
- **Syntax:** REPORT <rep> LINE-SIZE <width>.

```
REPORT demo_list_line_size LINE-SIZE 40.
WRITE:  'SY-LINSZ:', SY-LINSZ.
ULINE.
DO 20 TIMES.
  WRITE sy-index.
ENDDO.
```

```
Defining the width                   1
SY-LINSZ:    40
          1          2          3
          4          5          6
          7          8          9
         10         11         12
         13         14         15
         16         17         18
         19         20
```

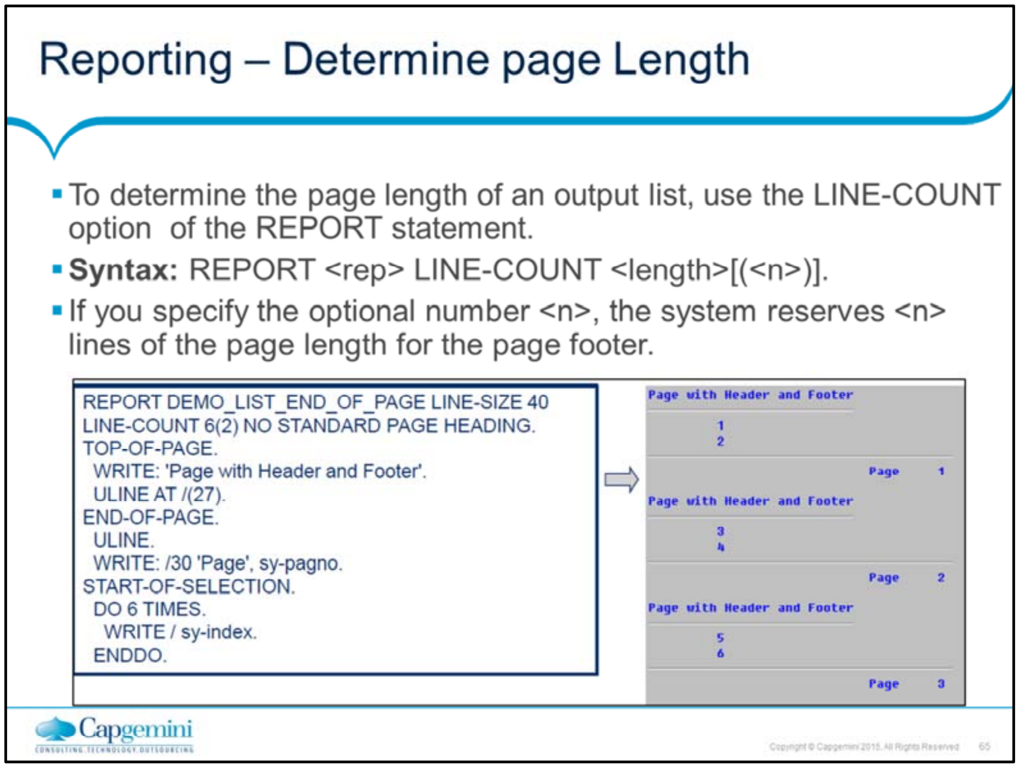
```
REPORT demo_list_line_size LINE-SIZE 60.
WRITE:  'SY-LINSZ:', SY-LINSZ.
ULINE.
DO 20 TIMES.
  WRITE sy-index.
ENDDO.
```

```
Defining the width                                         1
SY-LINSZ:    60
          1          2          3          4          5
          6          7          8          9         10
         11         12         13         14         15
         16         17         18         19         20
```

Capgemini

CONSULTING.TECHNOLOGY.OUTSOURCING

64



Reporting – Determine page Length

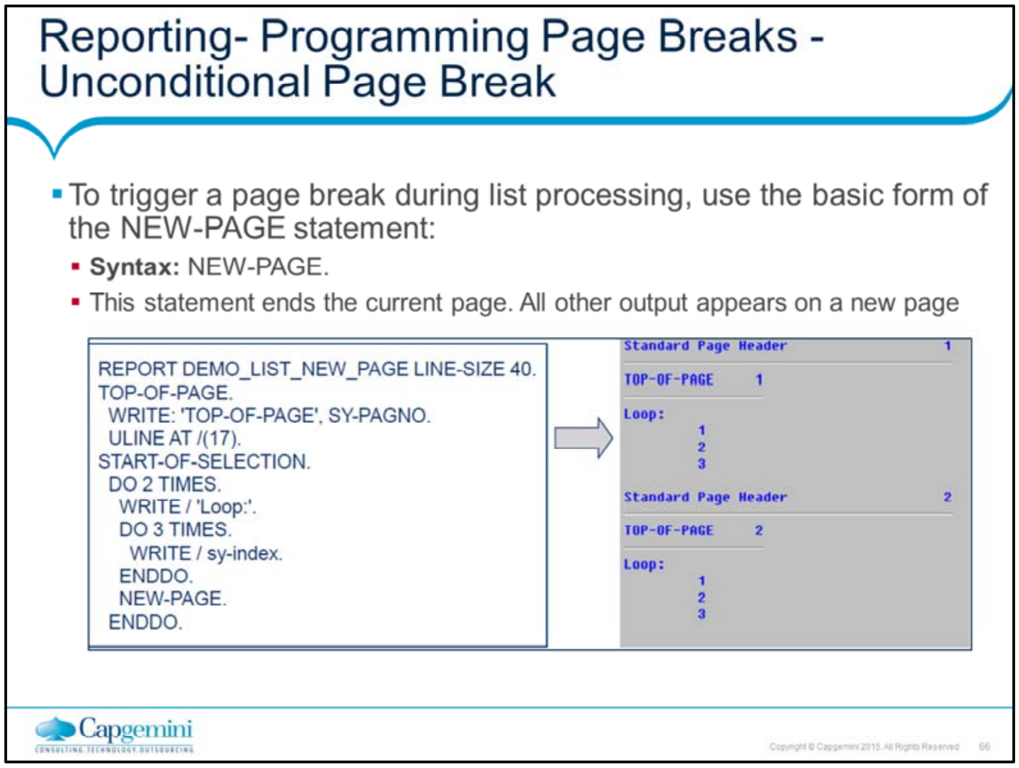
- To determine the page length of an output list, use the LINE-COUNT option of the REPORT statement.
- **Syntax:** REPORT <rep> LINE-COUNT <length>[(<n>)].
- If you specify the optional number <n>, the system reserves <n> lines of the page length for the page footer.

```
REPORT DEMO_LIST_END_OF_PAGE LINE-SIZE 40
LINE-COUNT 6(2) NO STANDARD PAGE HEADING.
TOP-OF-PAGE.
 WRITE: 'Page with Header and Footer'.
 ULINE AT /(27).
END-OF-PAGE.
 ULINE.
 WRITE: /30 'Page', sy-pagno.
START-OF-SELECTION.
 DO 6 TIMES.
   WRITE / sy-index.
 ENDDO.
```

```
Page with Header and Footer
---------------------------
          1
          2
----------------------------------------
                             Page     1
Page with Header and Footer
---------------------------
          3
          4
----------------------------------------
                             Page     2
Page with Header and Footer
---------------------------
          5
          6
----------------------------------------
                             Page     3
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

65



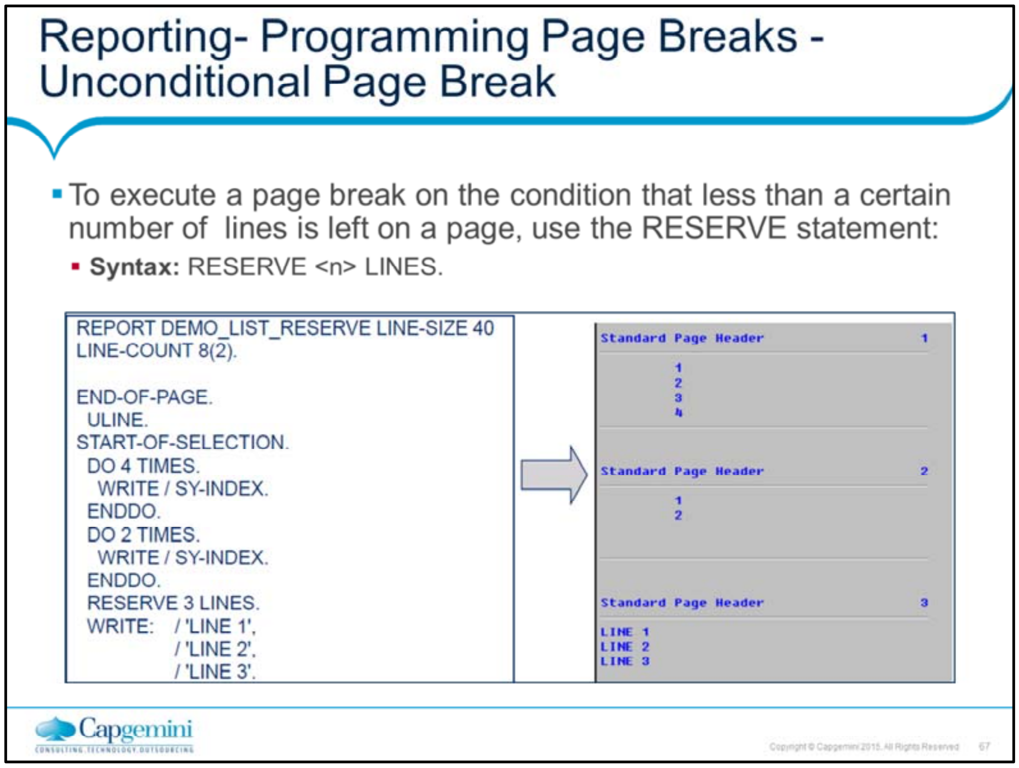
Reporting- Programming Page Breaks - Unconditional Page Break

- To trigger a page break during list processing, use the basic form of the NEW-PAGE statement:
  - **Syntax:** NEW-PAGE.
  - This statement ends the current page. All other output appears on a new page

```
REPORT DEMO_LIST_NEW_PAGE LINE-SIZE 40.
TOP-OF-PAGE.
 WRITE: 'TOP-OF-PAGE', SY-PAGNO.
 ULINE AT /(17).
START-OF-SELECTION.
 DO 2 TIMES.
   WRITE / 'Loop:'.
   DO 3 TIMES.
     WRITE / sy-index.
   ENDDO.
   NEW-PAGE.
 ENDDO.
```

```
Standard Page Header                    1

TOP-OF-PAGE      1

Loop:
          1
          2
          3

Standard Page Header                    2

TOP-OF-PAGE      2

Loop:
          1
          2
          3
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING



Reporting- Programming Page Breaks - Unconditional Page Break

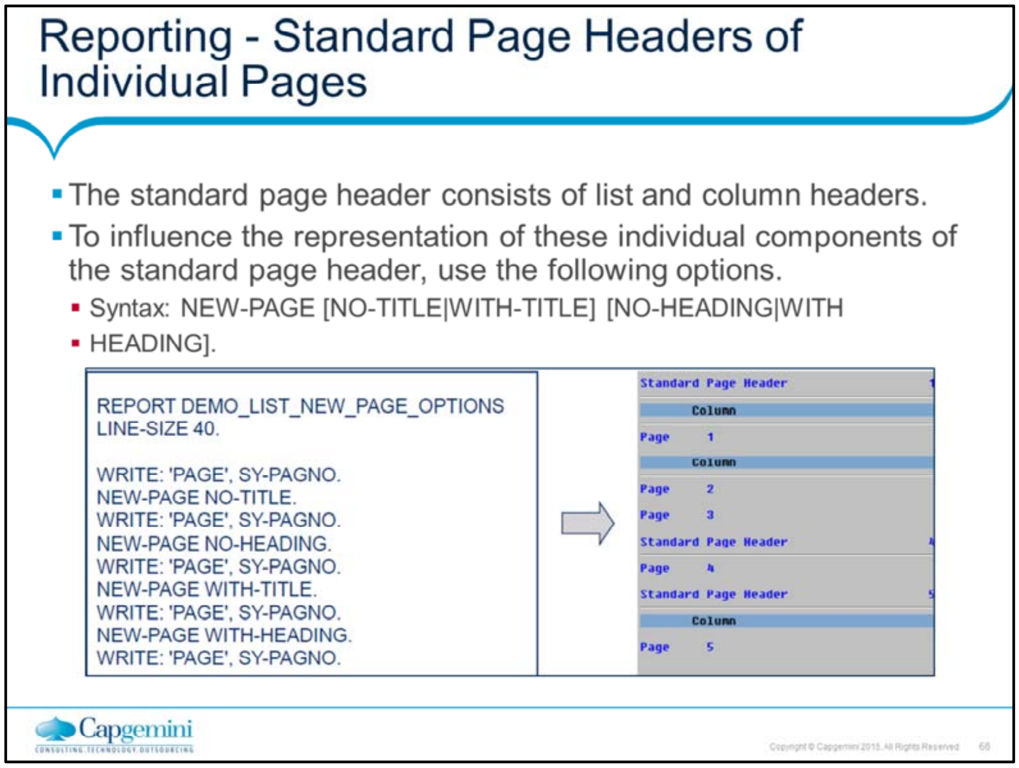
- To execute a page break on the condition that less than a certain number of lines is left on a page, use the RESERVE statement:
  - **Syntax:** RESERVE <n> LINES.

```
REPORT DEMO_LIST_RESERVE LINE-SIZE 40
LINE-COUNT 8(2).

END-OF-PAGE.
  ULINE.
START-OF-SELECTION.
  DO 4 TIMES.
    WRITE / SY-INDEX.
  ENDDO.
  DO 2 TIMES.
    WRITE / SY-INDEX.
  ENDDO.
  RESERVE 3 LINES.
  WRITE:    / 'LINE 1',
            / 'LINE 2',
            / 'LINE 3'.
```

```
Standard Page Header                   1
-----------------------------------------
          1
          2
          3
          4
-----------------------------------------

Standard Page Header                   2
-----------------------------------------
          1
          2

-----------------------------------------

Standard Page Header                   3
-----------------------------------------
LINE 1
LINE 2
LINE 3
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

67



Reporting - Standard Page Headers of Individual Pages

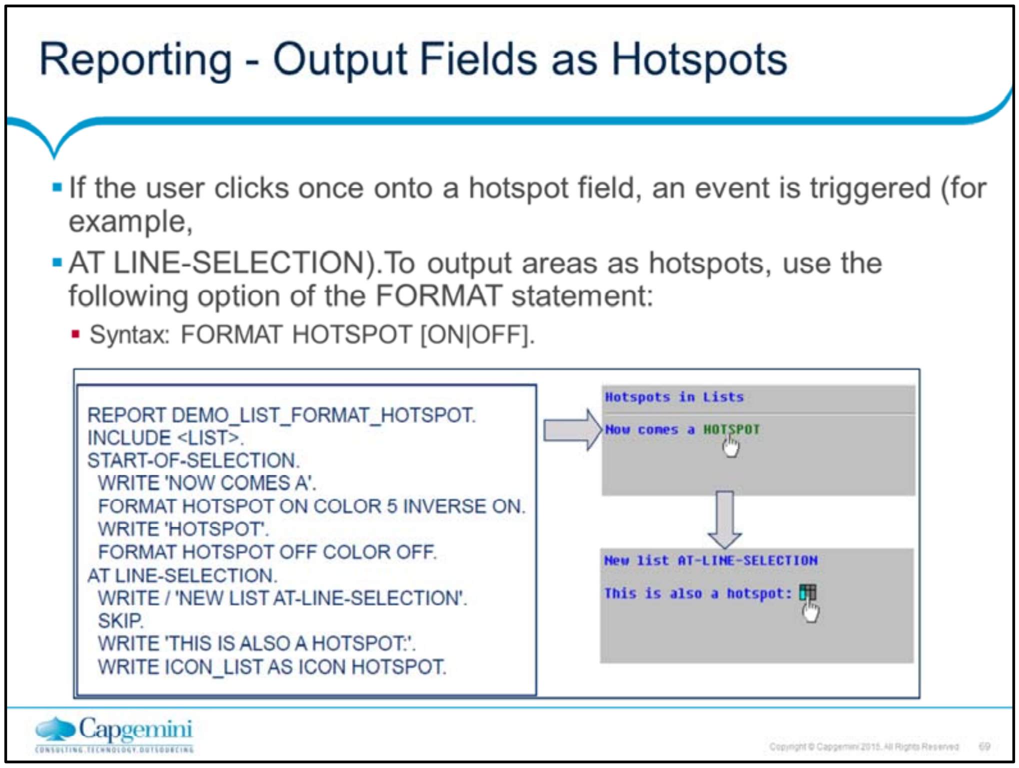
- The standard page header consists of list and column headers.
- To influence the representation of these individual components of the standard page header, use the following options.
  - Syntax: NEW-PAGE [NO-TITLE|WITH-TITLE] [NO-HEADING|WITH
  - HEADING].

```
REPORT DEMO_LIST_NEW_PAGE_OPTIONS
LINE-SIZE 40.

WRITE: 'PAGE', SY-PAGNO.
NEW-PAGE NO-TITLE.
WRITE: 'PAGE', SY-PAGNO.
NEW-PAGE NO-HEADING.
WRITE: 'PAGE', SY-PAGNO.
NEW-PAGE WITH-TITLE.
WRITE: 'PAGE', SY-PAGNO.
NEW-PAGE WITH-HEADING.
WRITE: 'PAGE', SY-PAGNO.
```

```
Standard Page Header                   1
        Column
Page     1
        Column
Page     2
Page     3
Standard Page Header                   4
Page     4
Standard Page Header                   5
        Column
Page     5
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

68



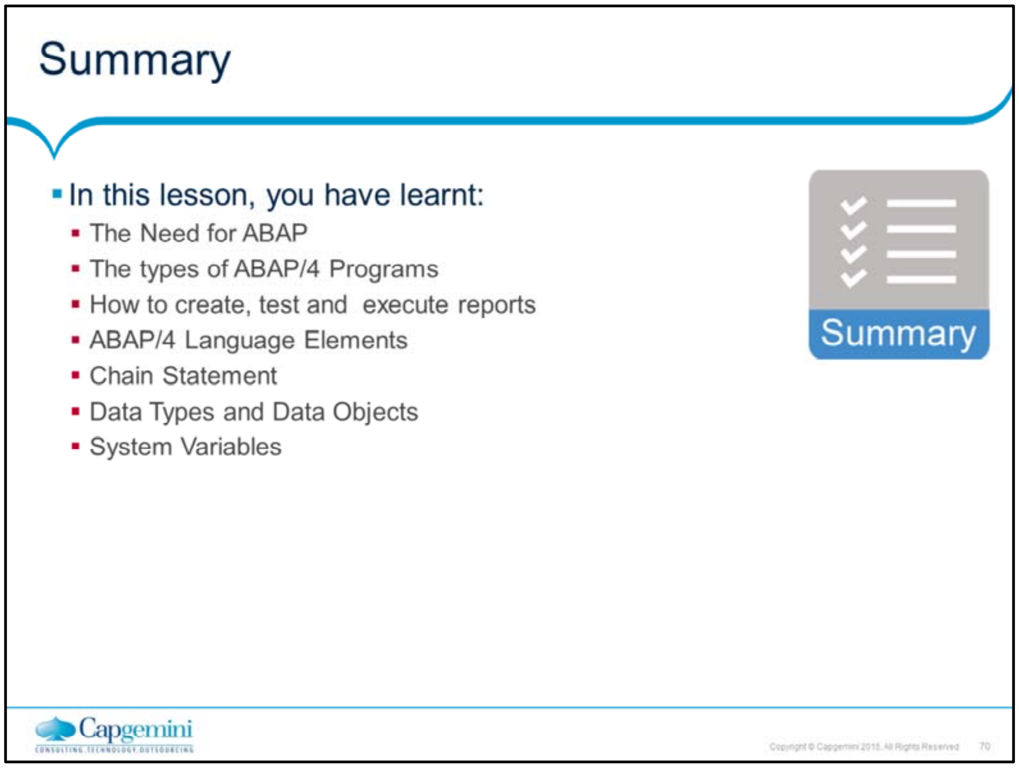
Reporting - Output Fields as Hotspots

- If the user clicks once onto a hotspot field, an event is triggered (for example,
- AT LINE-SELECTION).To output areas as hotspots, use the following option of the FORMAT statement:
  - Syntax: FORMAT HOTSPOT [ON|OFF].

```
REPORT DEMO_LIST_FORMAT_HOTSPOT.
INCLUDE <LIST>.
START-OF-SELECTION.
  WRITE 'NOW COMES A'.
  FORMAT HOTSPOT ON COLOR 5 INVERSE ON.
  WRITE 'HOTSPOT'.
  FORMAT HOTSPOT OFF COLOR OFF.
AT LINE-SELECTION.
  WRITE / 'NEW LIST AT-LINE-SELECTION'.
  SKIP.
  WRITE 'THIS IS ALSO A HOTSPOT:'.
  WRITE ICON_LIST AS ICON HOTSPOT.
```

```
Hotspots in Lists

Now comes a HOTSPOT
```

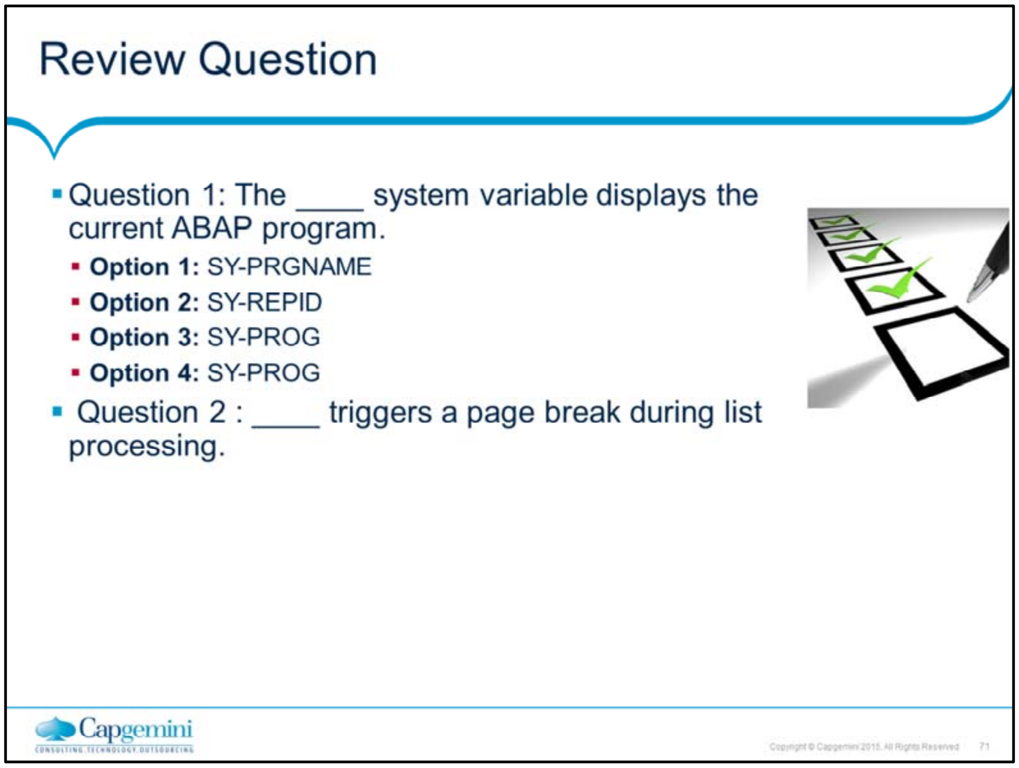
```
New list AT-LINE-SELECTION

This is also a hotspot:
```



Summary

- In this lesson, you have learnt:
  - The Need for ABAP
  - The types of ABAP/4 Programs
  - How to create, test and execute reports
  - ABAP/4 Language Elements
  - Chain Statement
  - Data Types and Data Objects
  - System Variables

Summary



Review Question

- Question 1: The _____ system variable displays the current ABAP program.
  - **Option 1:** SY-PRGNAME
  - **Option 2:** SY-REPID
  - **Option 3:** SY-PROG
  - **Option 4:** SY-PROG
- Question 2 : _____ triggers a page break during list processing.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING