Introduction to REXX & CLIST

Lesson 00:

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

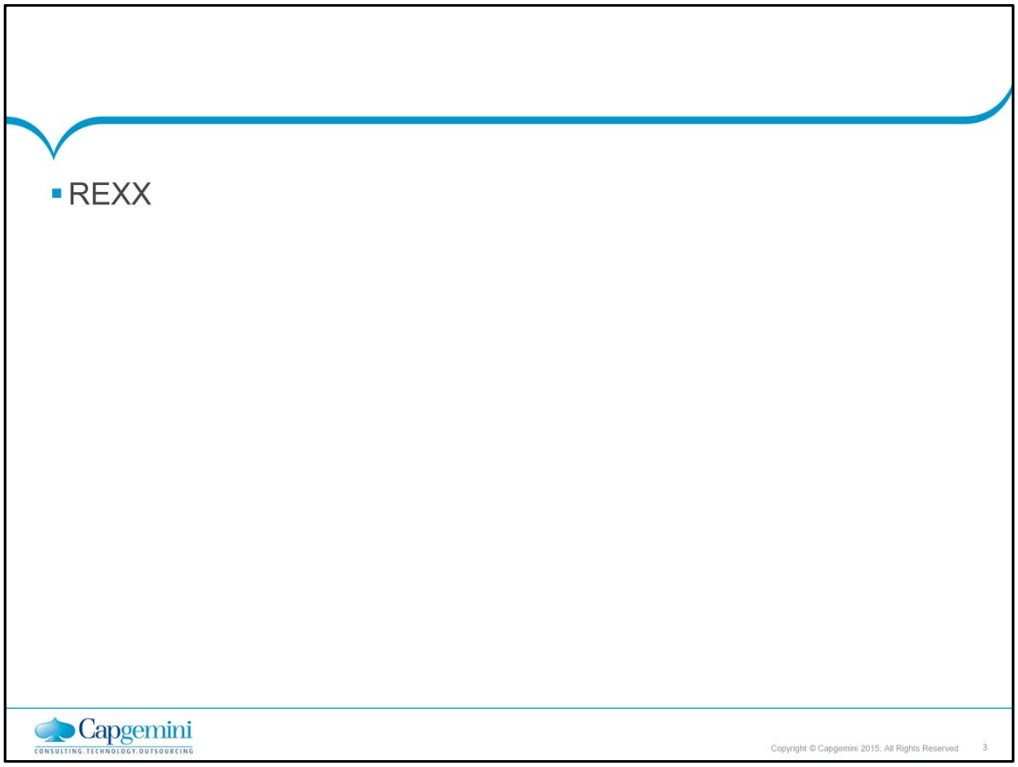
People matter, results count.

Document History

Date	Course Version No.	Software Version No.	Developer / SME	Change Record Remarks
12-Nov-2008	Draft 0.01D		Sudhir Karhadkar / Bela Kher	Original Draft
08-Jun-2009	1.0		CLS team	Review
9-Aug-2016	1.1		Veena K	Review, post the integration

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 2


REXX
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING
Copyright © Capgemini 2015. All Rights Reserved
3

Course Goals and Non Goals

- Course Goals
  - Introduction to REXX
- Course Non Goals
  - Data Stack
  - REXX in TSO/E and other MVS Address Spaces

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 4

Pre-requisites

- Good Hands-On experience on Mainframe
- Cobol
- JCL
- Well conversant with MF ISPF, TSO commands

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 5

Intended Audience

- Designers
- Developers
- Test Engineers

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 6

Day Wise Schedule

- Day 1
  - Lesson 1: Introduction and Overview
  - Lesson 2: Coding and Running a REXX Exec
  - Lesson 3: Variables and Operators
  - Lesson 4: Control Flow constructs

Capgemini
CONSULTING. TECHNOLOGY. OUTSOURCING

 7

Table of Contents

Capgemini
CONSULTING. TECHNOLOGY. OUTSOURCING

 8

Table of Contents (Contd…)

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 9

Table of Contents (Contd…)

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 10


CLIST
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING
Copyright © Capgemini 2015. All Rights Reserved
11

Course Goals and Non Goals

- Course Goals
  - Creating and Executing CLIST
- Course Non Goals
  - Using CLISTs with ISPF panels
  - Using nested CLISTs

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 12

Day Wise Schedule

- Day 1
  - Lesson 1: Introduction to CLIST
  - Lesson 2: Creating, Editing, and Executing CLISTs
  - Lesson 3: Writing CLISTs – Syntax and Conventions

Capgemini
CONSULTING. TECHNOLOGY. OUTSOURCING

 13

Table of Contents

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 14

Table of Contents

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 15

References

- IBM TSO/E REXX User's Guide
- IBM ISPF Edit and Edit Macros Manual
- IBM ISPF Services Guide
- IBM ISPF Dialog Developer's Guide and Reference

- Website:
  - http://www.theamericanprogrammer.com/programming/rexx.clist.shtml

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 16

Next Step Courses (if applicable)

- Advanced REXX
- Advanced CLIST

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 17

REXX

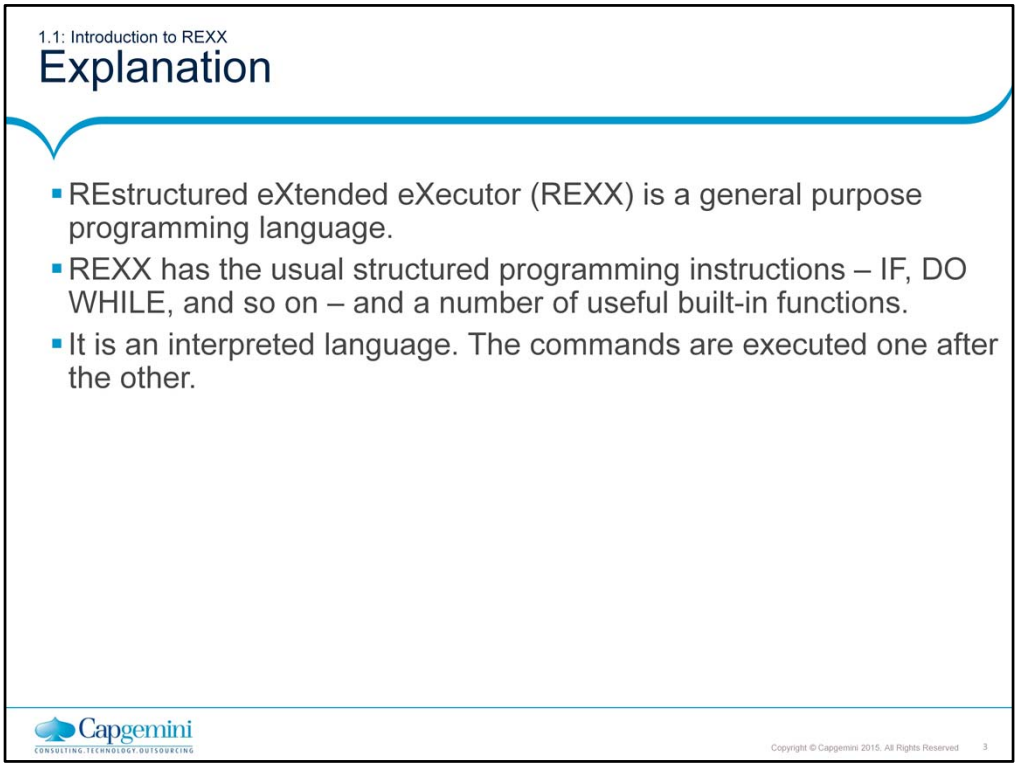
Lesson 1: Introduction and Overview

Lesson Objectives

- In this lesson, you will learn about:
  - Introduction to REXX
  - Features of REXX
  - Components of REXX

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 2

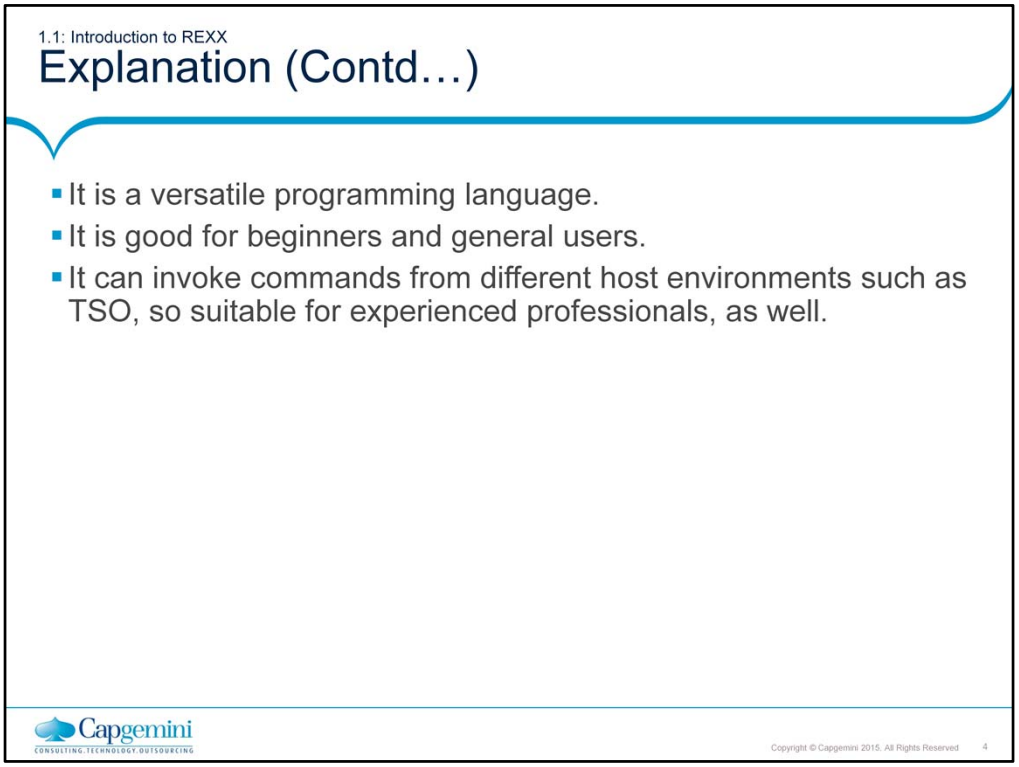


Introduction to REXX:

The language imposes no restrictions on program format. There can be more than one clause on a line, or a single clause can occupy more than one line. Indentation is allowed.

REXX is suitable for the following:

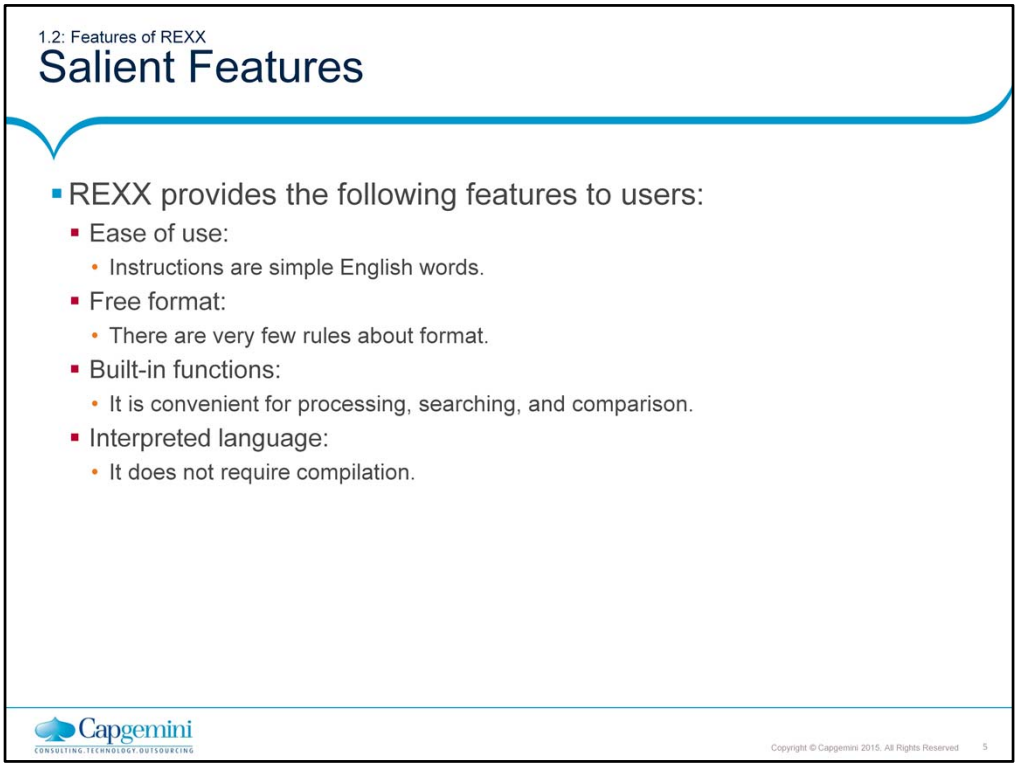
- Application front ends
- User-defined macros (such as editor subcommands)
- Command procedures
- Prototyping



Introduction to REXX (contd.):

Common programming structure, readability, and free format make it a good language for beginners and general users.

REXX language can be intermixed with commands to different host environments, thus providing powerful functions. It is also suitable for more experienced computer professionals.



Features of REXX:

REXX provides the following features to users:

Ease of use

The REXX language is easy to read and write. Many instructions are meaningful English words. Unlike some lower-level programming languages that use abbreviations, REXX instructions are common words, such as SAY, PULL, IF...THEN... ELSE..., DO... END, and EXIT.

Free format

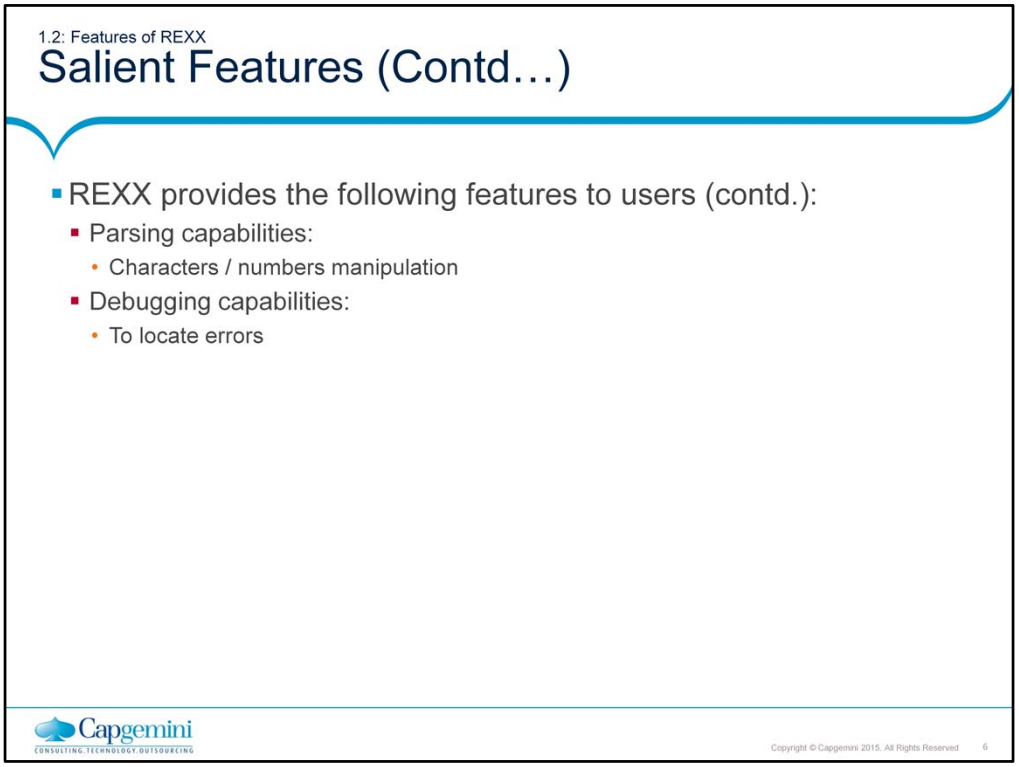
There are few rules about REXX format. Hence you need not start an instruction in a particular column, you can skip spaces in a line or skip entire lines, you can have an instruction span many lines or have multiple instructions on one line, variables need not be predefined, and you can type instructions in upper, lower, or mixed case.

Convenient built-in functions

REXX supplies built-in functions that perform various processing, searching, and comparison operations for both text and numbers. Other built-in functions provide formatting capabilities and arithmetic calculations.

Interpreted language

TSO/E implements the REXX language as an interpreted language. When a REXX exec runs, the language processor directly processes each language statement. Languages that are not interpreted must be compiled and link-edited for execution.



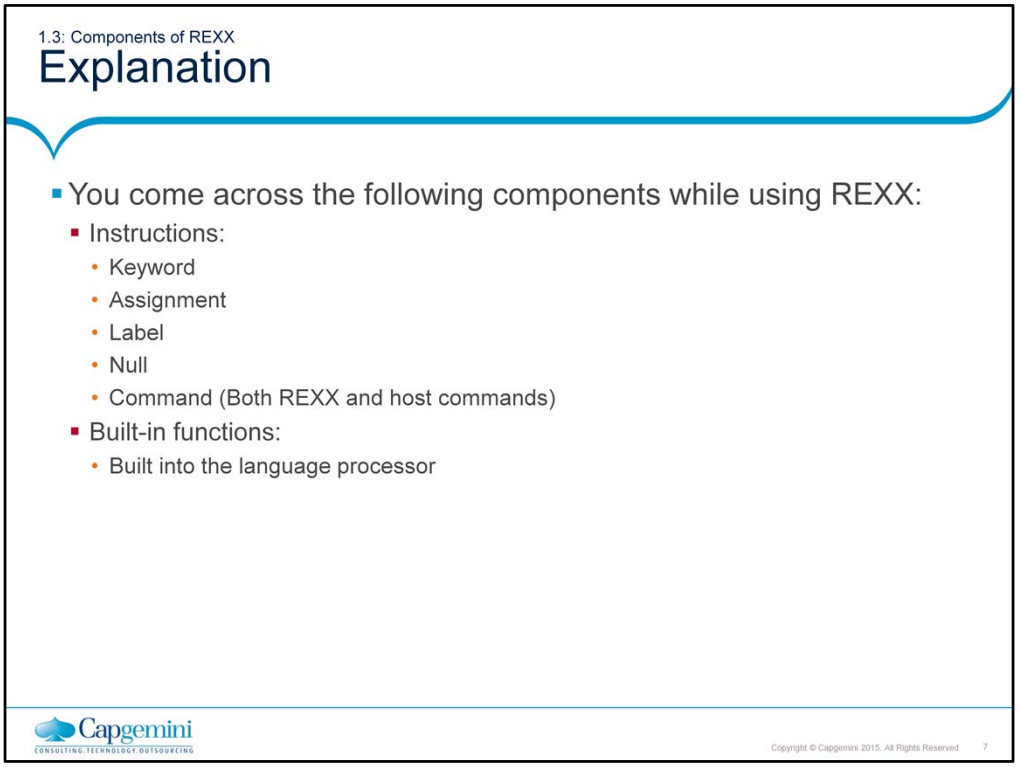
Features of REXX (contd.):

Parsing capabilities

REXX includes extensive parsing capabilities for character manipulation. This parsing capability allows you to set up a pattern to separate characters, numbers, and mixed input.

Debugging capabilities

When a REXX exec running in TSO/E encounters an error, messages describing the error are displayed on the screen. In addition, you can use the REXX TRACE instruction and the interactive debug facility to locate errors in execs.



Components of REXX:

Instructions:

Keyword:

A keyword instruction is one or more clauses, the first of which starts with a keyword that identifies the instruction. Keyword instructions control the external interfaces, the flow of control, and so on. Some keyword instructions can include nested instructions. For example:

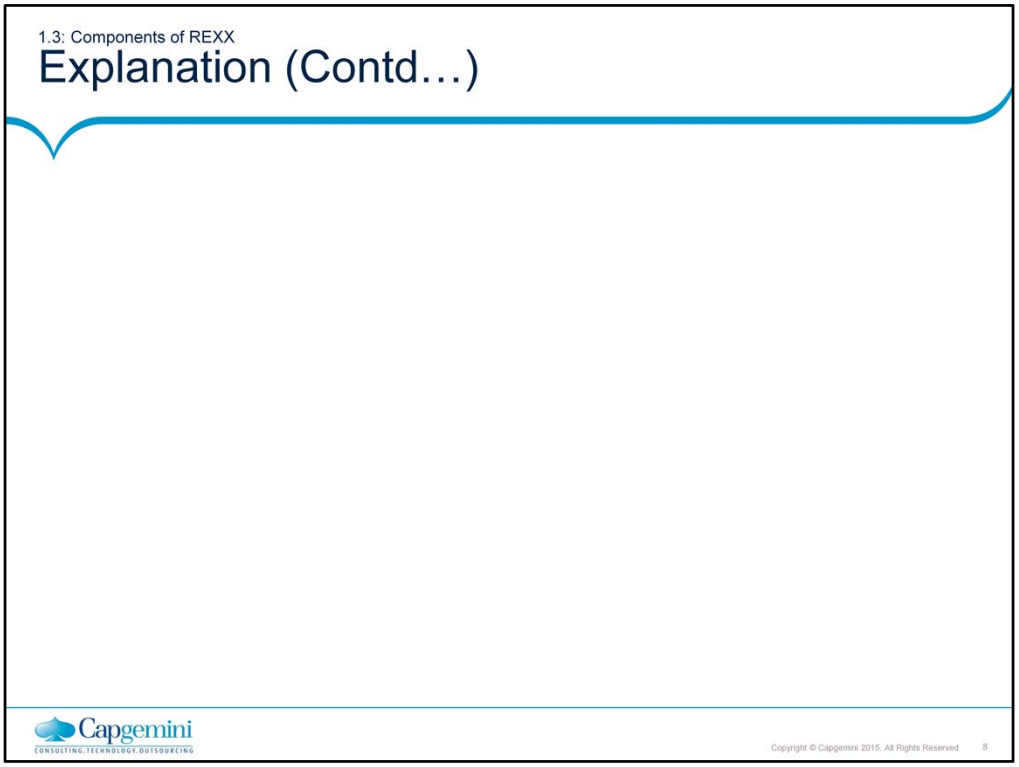
```
DO
  instruction
  instruction
END
```

Assignment:

A single clause of the form symbol=expression is an instruction known as an assignment. An assignment gives a variable a (new) value.

The result of expression becomes the new value of the variable named by the symbol. On TSO/E, if you omit expression, the variable is set to the null string. However, it is recommended that you explicitly set a variable to the null string: symbol=''.

contd.



Components of REXX (contd.):

Instructions (contd.):

Label:

A clause that consists of a single symbol followed by a colon is a label. The colon in this context implies a semicolon (clause separator), so no semicolon is required.

Labels identify the targets of CALL instructions and internal function calls.

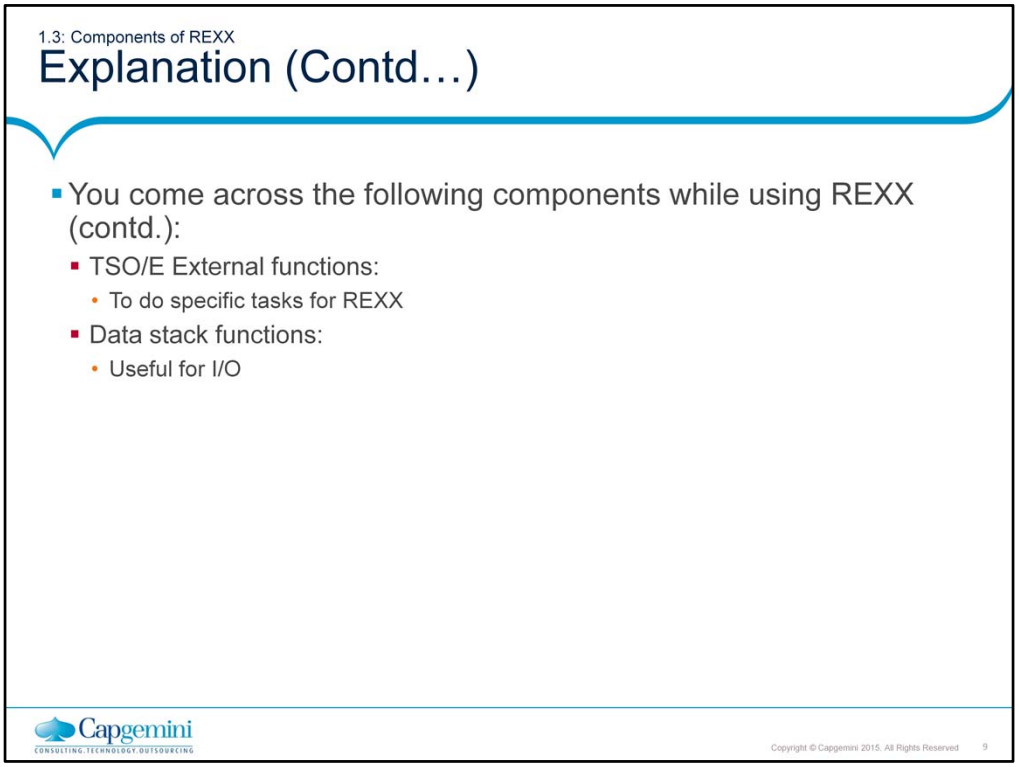
Null:

A clause consisting only of blanks or comments or both is a null clause. It is completely ignored.

Command (Both REXX and host commands):

An instruction that is not a keyword instruction, assignment, label, or null is processed as a command and is sent to a previously defined environment for processing.

contd.



Components of REXX (contd.):

Built-in functions

These functions are built into the language processor and provide convenient processing options. For example: SUBSTR(), DATE()

TSO/E External functions

These functions are provided by TSO/E and interact with the system to do specific tasks for REXX.

For example: LISTDSI, OUTTRAP

Data stack functions

A data stack can store data for I/O and other types of processing. Useful in reading and writing data into datasets.

Summary

- In this lesson, you have learnt:
  - Introduction to the concept of REXX
  - Features provided by REXX
    - Ease of Use
    - Free Format
    - Built-in functions
    - Interpreted Language
    - Parsing Capabilities
    - Debugging Capabilities
  - Components of REXX:
    - Instructions
    - Built-in Functions
    - TSO/E External Functions
    - Data stack functions

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 10

Review - Questions

- Question 1: REXX program cannot be run in interpreted mode.
  - True / False
- Question 2: REXX and JCL programs are the same.
  - True / False
- Question 3: REXX stands for ___ ___ ___.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 11

REXX

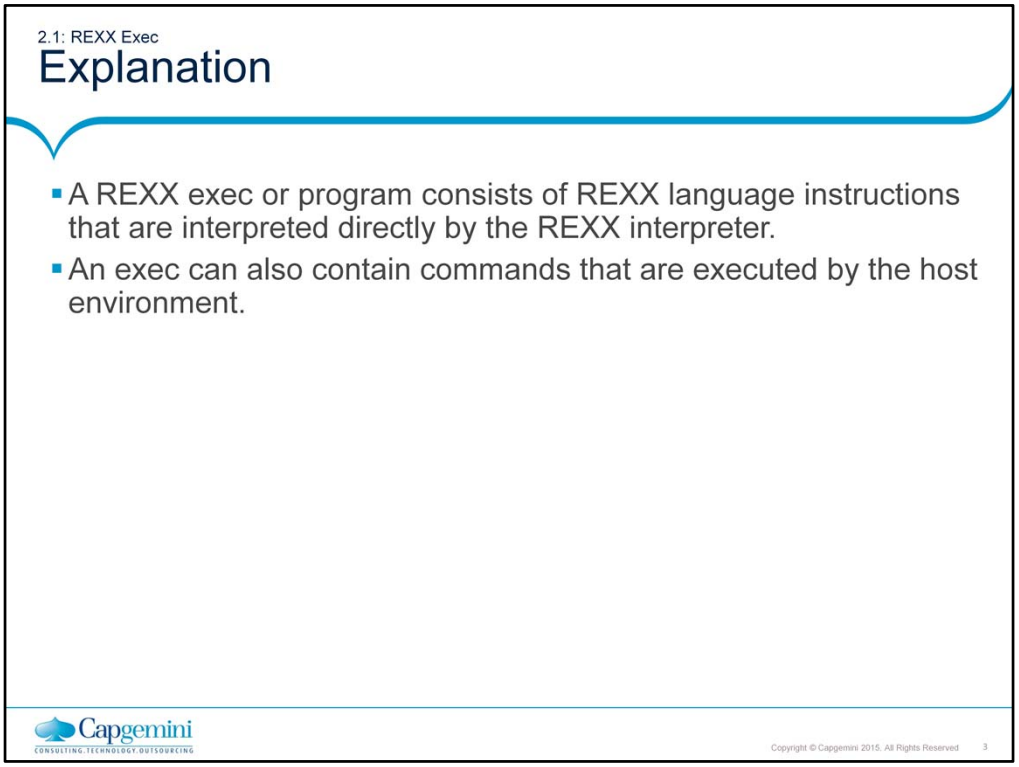
Lesson 2: Coding and Running a REXX Exec

Lesson Objectives

- In this lesson, you will learn about the following topics:
  - REXX Exec
  - Syntax of REXX instruction
  - How do you run an Exec?
  - Interpreting error messages
  - Passing data to an Exec

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 2

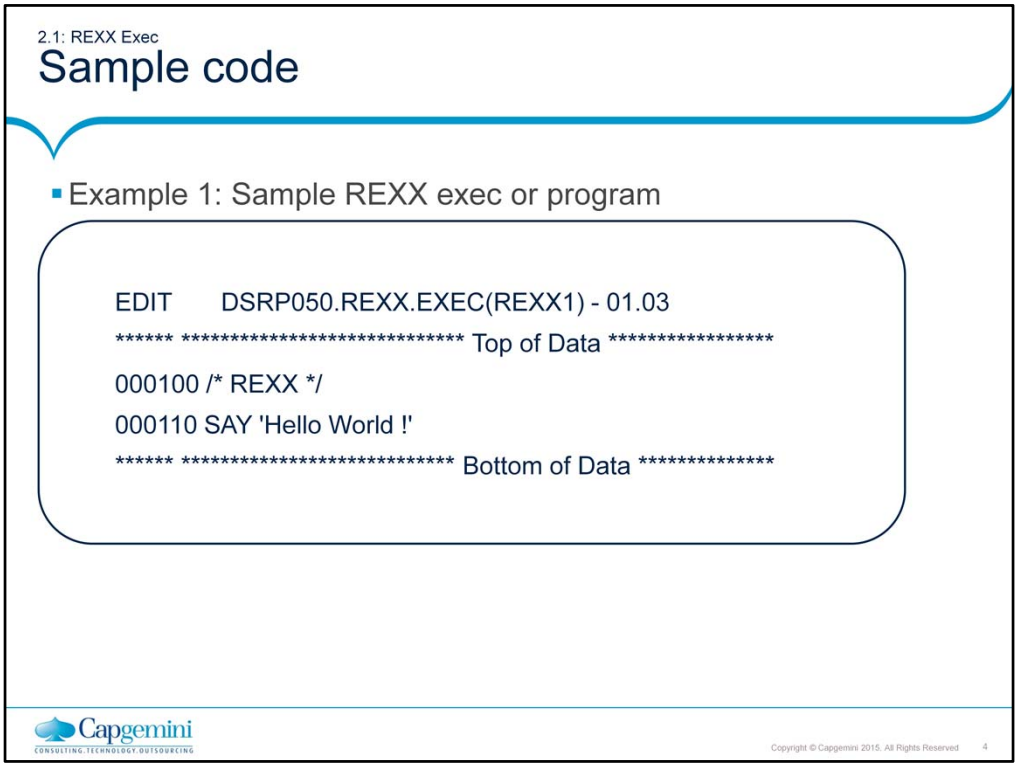


REXX Exec:

An advantage of the REXX language is its similarity to English. This similarity makes it easy to read and write a REXX exec.

For example: An exec to display a sentence on the screen uses the REXX instruction SAY followed by the sentence to be displayed within quote marks.

The sample code is shown on the next slide.



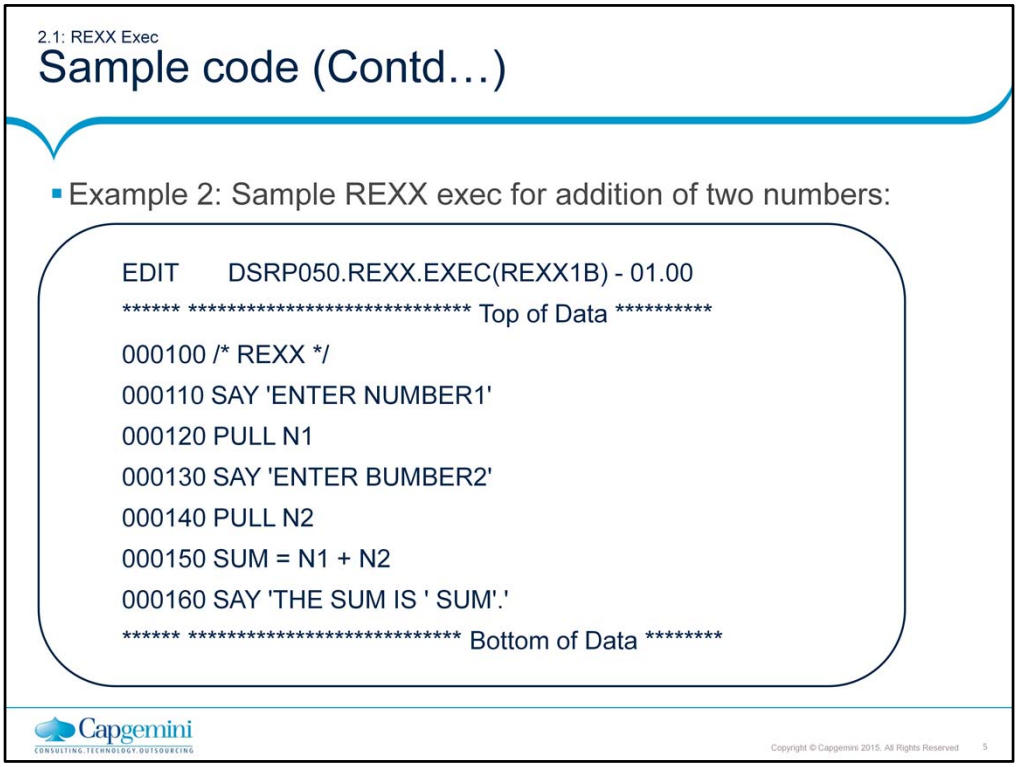
REXX Exec (contd.):

Note that this simple exec starts with a comment line to identify the program as a REXX exec. A comment begins with /* and ends with */. This is required to distinguish REXX from CLIST.

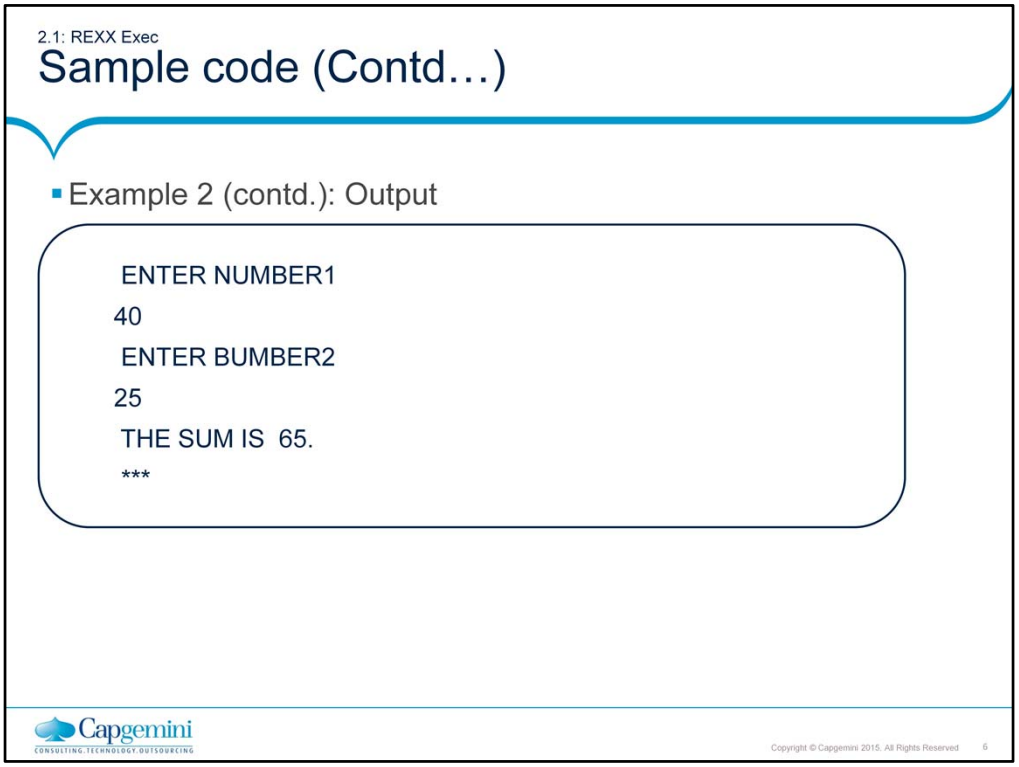
To prevent incompatibilities with CLISTs, IBM recommends that all REXX execs start with a comment that includes the characters “REXX” within the first line (line 1) of the exec. Otherwise, your REXX exec will not get executed.

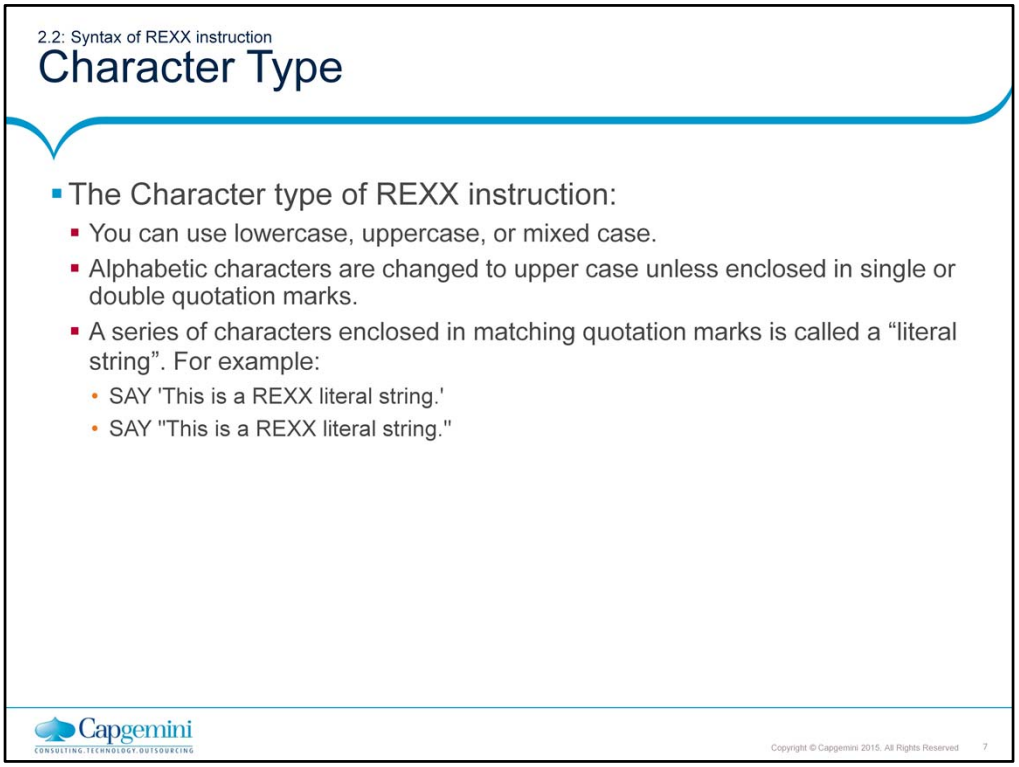
After executing the Exec shown in the slide, the following line will be displayed:

Hello World !



Note: Output of the above exec is given on next page.


2.1: REXX Exec
Sample code (Contd…)
Example 2 (contd.): Output
ENTER NUMBER1
40
ENTER BUMBER2
25
THE SUM IS 65.
***
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING
Copyright © Capgemini 2015. All Rights Reserved
6



Syntax of REXX instruction:

You cannot enclose a literal string with one each of the two types of quotation marks. The following is not a correct example of an enclosed literal string.

SAY 'This is a REXX literal string.” (Mismatching quotation marks)

Suppose you omit the quotation marks from a SAY instruction as follows:

SAY This is a REXX string.

Then the statement is displayed in uppercase on the screen as shown below:

THIS IS A REXX STRING.

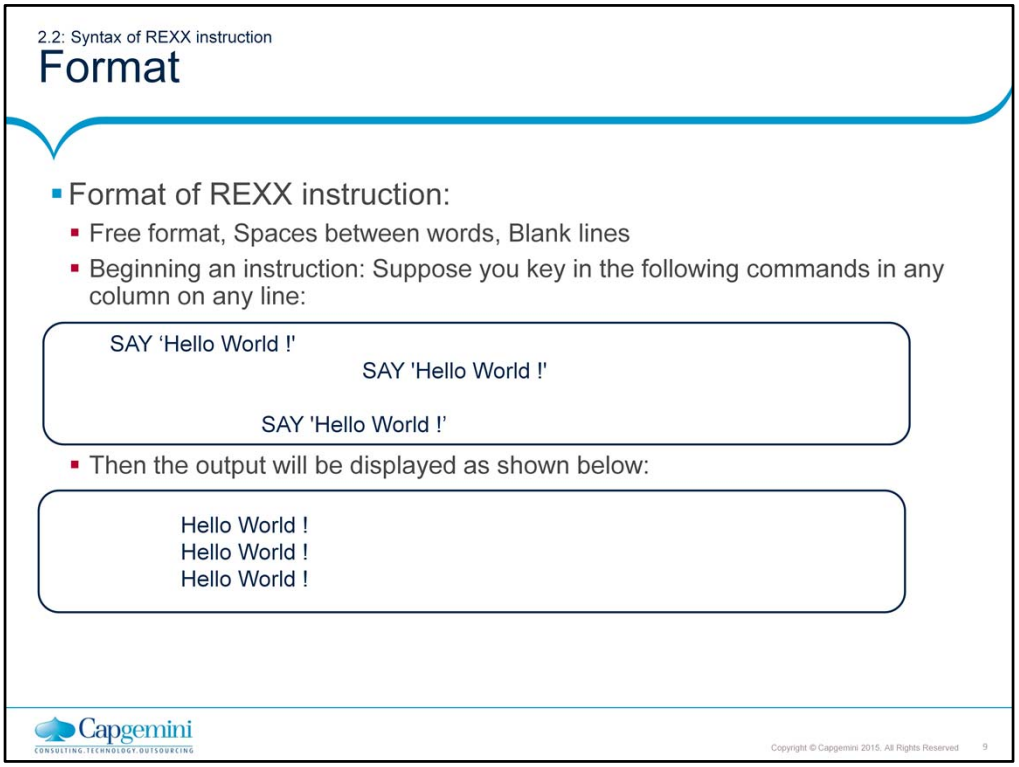
2.2: Syntax of REXX instruction

Character Type (Contd…)

- The Character type of REXX instruction (contd.)
  - If a string contains an apostrophe, then enclose the literal string in double quotation marks.
    - For example: SAY “This isn't a CLIST instruction.”
  - You can also use two single quotation marks in place of the apostrophe, because a pair of single quotation marks is processed as one.
    - For example: SAY 'This isn''t a CLIST instruction.'

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 8


2.2: Syntax of REXX instruction
Format
Format of REXX instruction:
Free format, Spaces between words, Blank lines
Beginning an instruction: Suppose you key in the following commands in any column on any line:
SAY ‘Hello World !'
SAY 'Hello World !'
SAY 'Hello World !’
Then the output will be displayed as shown below:
Hello World !
Hello World !
Hello World !
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING
Copyright © Capgemini 2015. All Rights Reserved
9

2.2: Syntax of REXX instruction

Format (Contd…)

- Format of REXX instruction:
  - Continuing an instruction: Use a comma for the next line.
  - For example:

```
SAY 'This is an extended',
'REXX literal string.'
```

  - Output is displayed as shown below:
    - This is an extended REXX literal string.
  - A space is added between “extended” and “REXX”.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 10

2.2: Syntax of REXX instruction

Format (Contd…)

- Format of REXX instruction
  - Continuing an instruction: Following two instructions are identical, and give the same result.
  - Example 1:

```
SAY 'This is',
    'a string.'
```

  - Example 2:

```
SAY 'This is' 'a string.'
```

  - Output for both samples will be as shown below:
    - This is a string.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 11

2.2: Syntax of REXX instruction

Format (Contd…)

- Format of REXX instruction:
  - Continue a literal string without adding a space.
  - Example 1:

```
SAY 'This is a string that is bro'||,
'ken in an awkward place.‘
```

  - O/P: This is a string that is broken in an awkward place.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 12

2.2: Syntax of REXX instruction

Format (Contd…)

- Format of REXX instruction:
  - Continue a literal string without adding a space (contd.):
  - Example 2:

```
SAY 'This is' ||,
'a string.'
```

  - Example 3:

```
SAY 'This is' || 'a string.'
```

    - O/P: This isa string.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 13

2.2: Syntax of REXX instruction

Format (Contd…)

- Format of REXX instruction:
  - Ending an instruction: End of the line or a semicolon
  - Example:

```
SAY 'Hi!'; say 'Hi again!'; say 'Hi for the last time!'
```

  - O/P:
    - Hi!
    - Hi again!
    - Hi for the last time!

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 14

2.2: Syntax of REXX instruction

Types

- Types of REXX instruction:
  - Let us use the following example for discussing the types:

```
000001 /*  REXX  */
000002 /*  This Exec accepts time and uses TIME command  */
000003 Game1:
000004 SAY 'What time is it?'
000005 PULL usertime                    /* Put the user's responses
into
000006                                  a variable called usertime  */
000007 IF usertime = '' '' THEN /* User did not enter the time */
000008         SAY "O.K. Game's over."
000009 ELSE
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 15

2.2: Syntax of REXX instruction

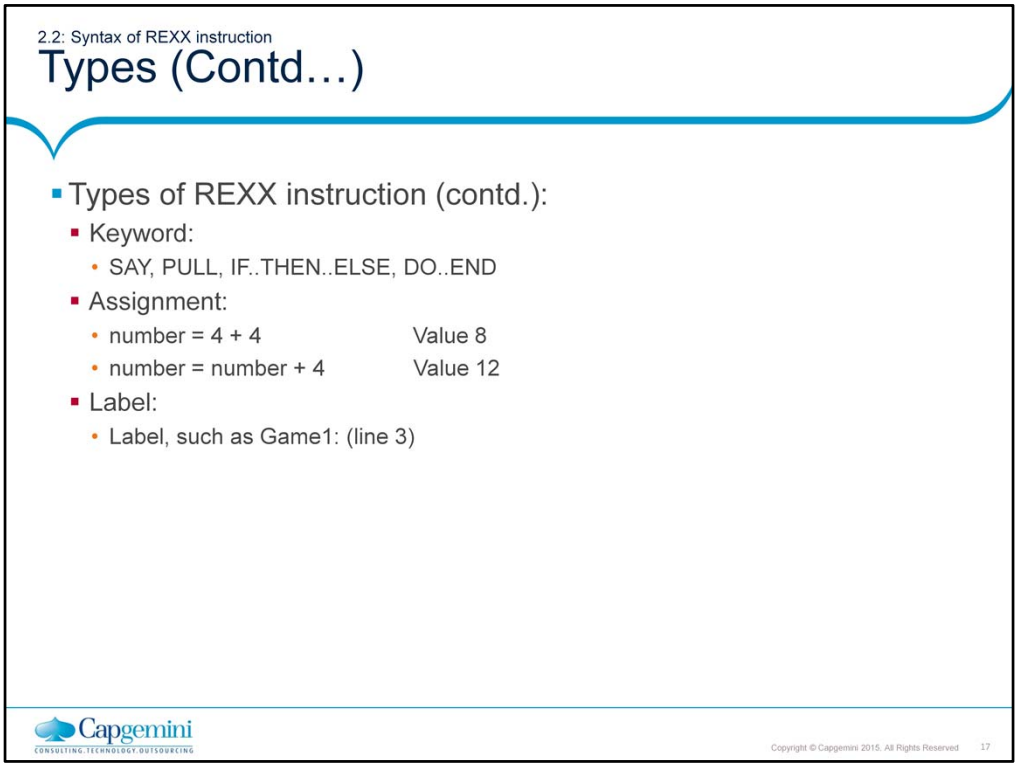
Types (Contd…)

- Types of REXX instruction (contd.):

```
000010        DO
000011 SAY "The computer says:"
000012 /* TSO System */  TIME  /* command */
000013 END
000014 EXIT
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 16



Syntax of REXX instruction (contd.):

Keyword:

A keyword instruction tells the language processor to do something. It begins with a REXX keyword that identifies what the language processor has to do.

For example:

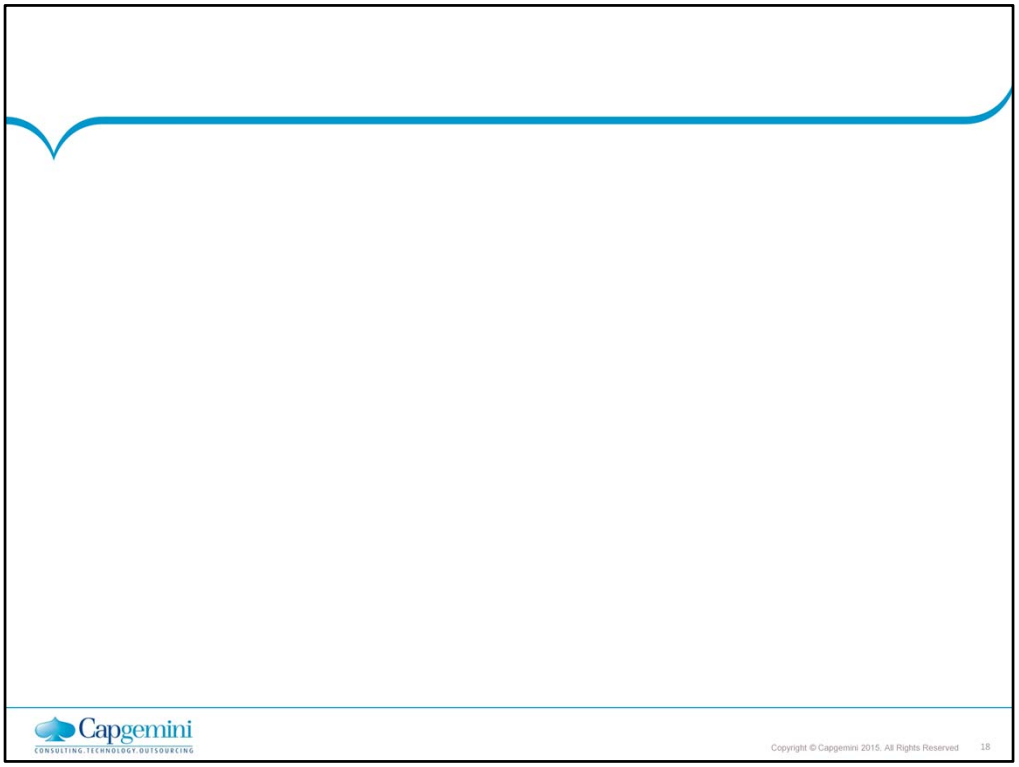
SAY (line 4) displays a string on the screen, and PULL (line 5) takes one or more words of input and puts them into the variable usertime.

IF, THEN (line 7) and ELSE (line 9) are three keywords that work together in one instruction. Each keyword forms a clause, which is a subset of an instruction.

If the expression that follows the IF keyword is TRUE, then the instruction that follows the THEN keyword is processed. Otherwise, the instruction that follows the ELSE keyword is processed.

If more than one instruction follows a THEN or an ELSE, then the instructions are preceded by a DO (line 10) and followed by an END (line 13).

The EXIT keyword (line 14) tells the language processor to end the exec. Using EXIT in this example is a convention, not a necessity, because processing ends automatically when there are no more instructions in the exec.



Syntax of REXX instruction (contd.):

Assignment:

An assignment gives a value to a variable or changes the current value of a variable. A simple assignment instruction is: number = 4

In addition to giving a variable a straightforward value, an assignment instruction can also give a variable the result of an expression. An expression is something that needs to be calculated, such as an arithmetic expression. The expression can contain numbers, variables, or both.

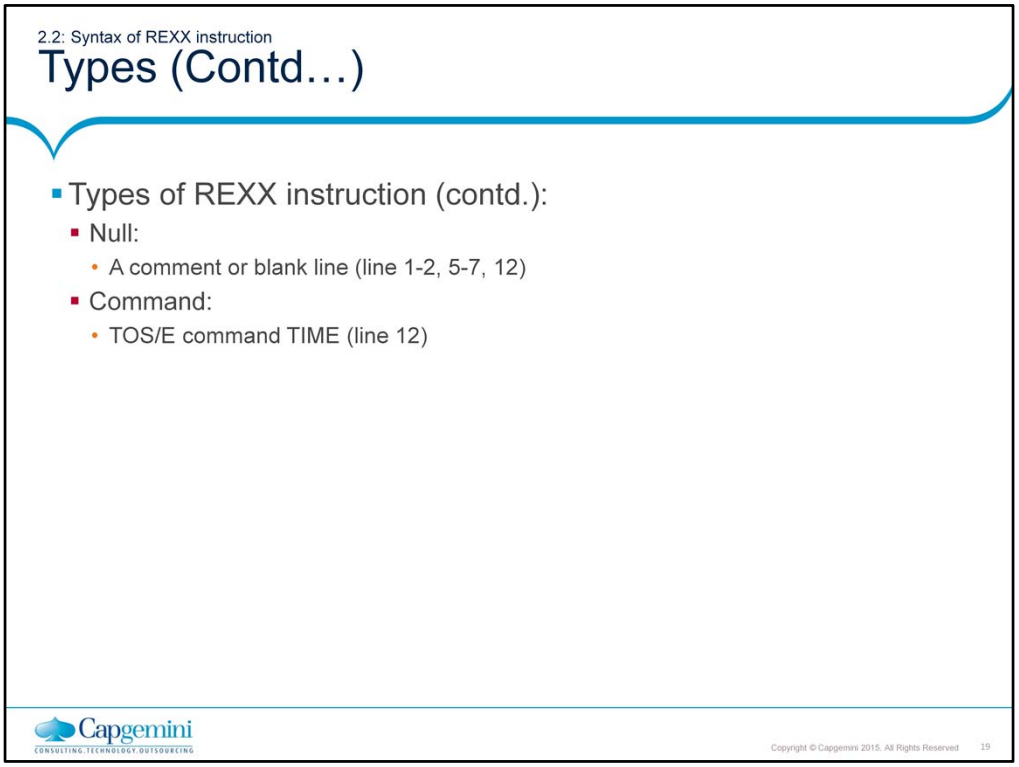
number = 4 + 4 (Value 8)

number = number + 4 (Value 12)

Label:

A label, such as Game1: (line 3), is a symbolic name followed by a colon.

A label identifies a portion of the exec and is commonly used in subroutines and functions.



Syntax of REXX instruction (contd.):

Null:

A null is a comment or a blank line, which is ignored by the language processor. However, it makes an exec easier to read. Comments (lines 1-2, 5-7, 12)

A comment begins with /* and ends with */. Comments can be on one or more lines or on part of a line. You can put information in a comment that might not be obvious to a person reading the REXX instructions.

Comments at the beginning can describe the overall purpose of the exec and perhaps list special considerations.

A comment next to an individual instruction can clarify its purpose.

REXX execs start with a comment that includes the characters “REXX” within the first line (line 1) of the exec.

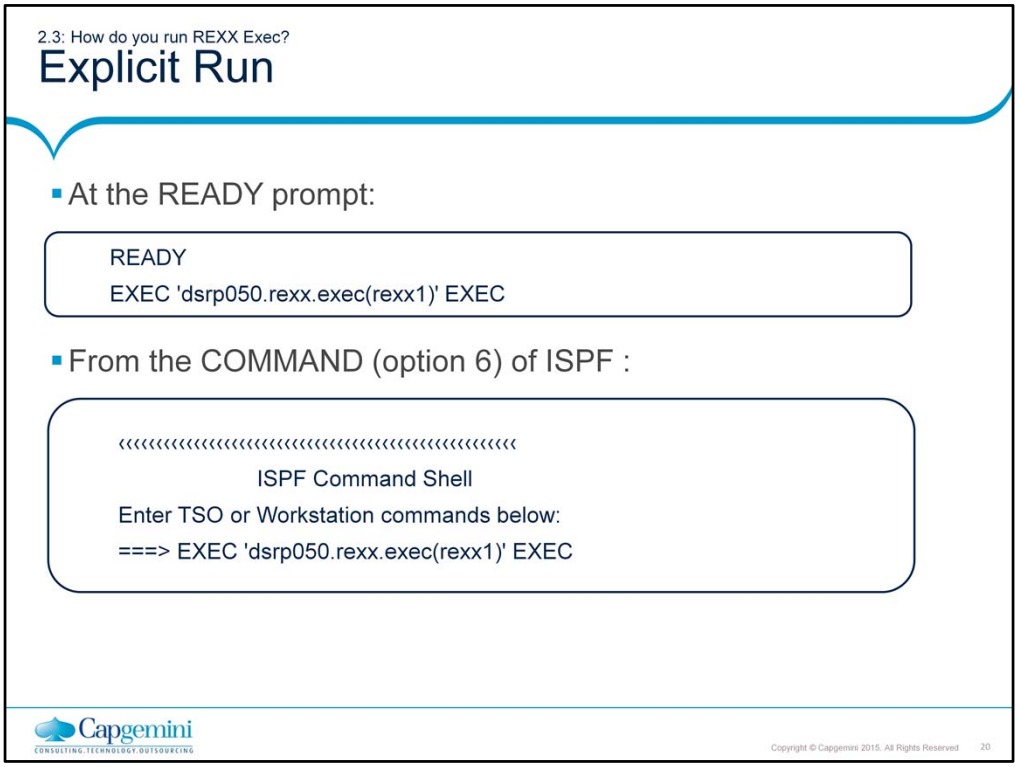
Blank lines are also NULL instructions.

Command:

An instruction that is not a keyword instruction, assignment, label, or null is processed as a command, and is sent to a previously defined environment for processing.

For example: The word "TIME" in this exec (line 12) (also shown below), though surrounded by comments, is processed as a TSO/E command.

```
/* TSO system */ TIME /* command */
```



How do you run REXX Exec – Explicit Run:

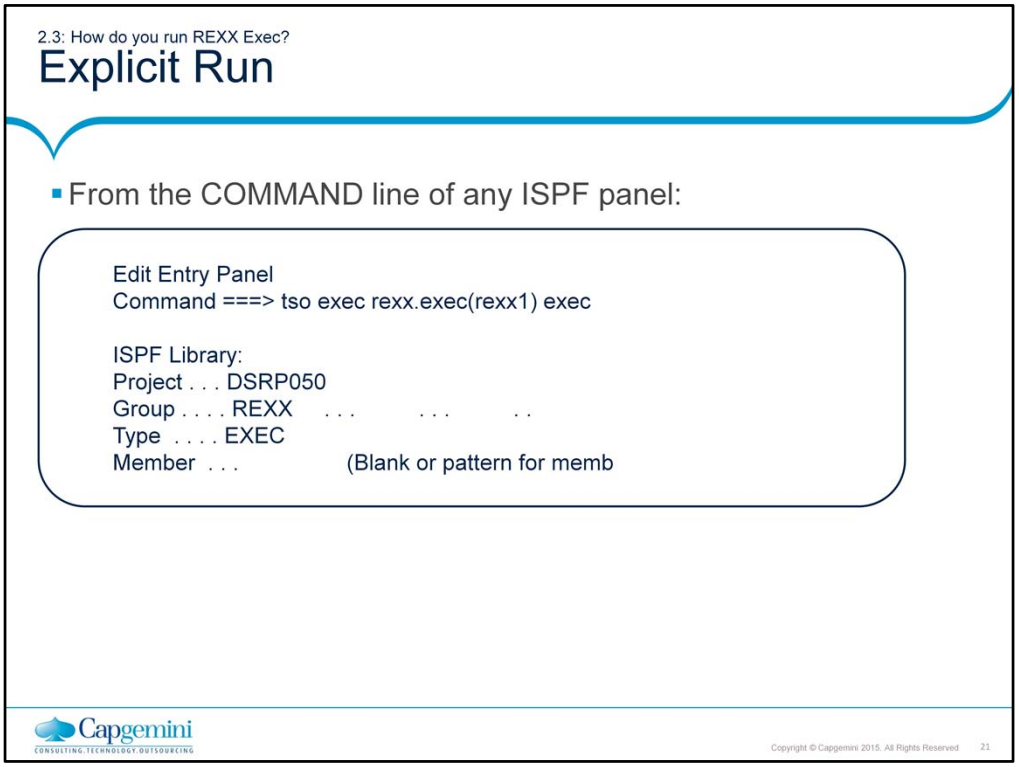
A fully-qualified data set, which appears within quotation marks can be used for the explicit run.

```
EXEC 'userid.rexx.exec(rexx1)' exec
```

A non fully-qualified data set, which has no quotation marks can eliminate your profile prefix (usually your user ID) as well as the third qualifier, exec.

```
EXEC rexx.exec(rexx1) exec /* eliminates prefix */
EXEC rexx(rexx1) exec /* eliminates prefix and exec */
```

The ending “exec” keyword is to distinguish it from CLIST.



How do you run REXX Exec – Explicit Run (contd.):

From the COMMAND line of any ISPF/PDF panel, the EXEC command is preceded by the word “tso”.

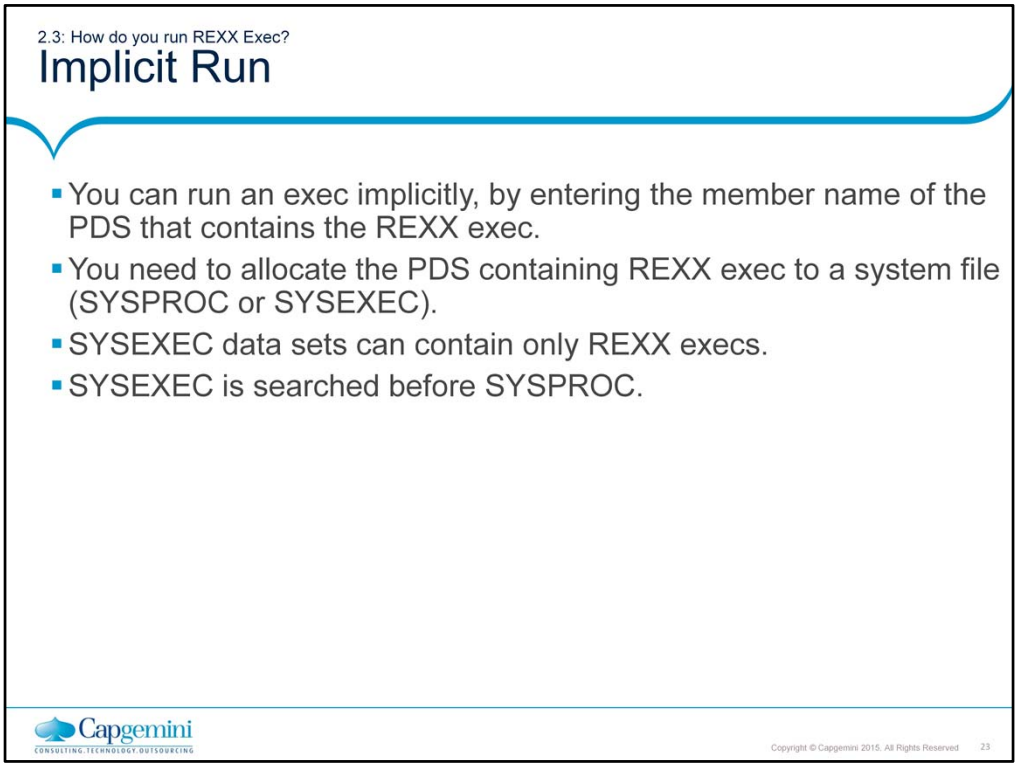
2.3: How do you run REXX Exec?

Demo

- Running REXX Exec - Explicitly

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 22



How do you run REXX Exec – Implicit Run:

Running an exec implicitly means running an exec by simply entering the member name of the data set that contains the exec. Before you can run an exec implicitly, you must allocate the PDS that contains the REXX exec to a system file (SYSPROC or SYSEXEC).

SYSPROC is a system file whose data sets can contain both CLISTs and execs. (Execs are distinguished from CLISTs by the REXX exec identifier – a comment at the beginning of the exec the first line of which includes the word "REXX".)

SYSEXEC is a system file whose data sets can contain only execs.

When both system files are available, SYSEXEC is searched before SYSPROC.

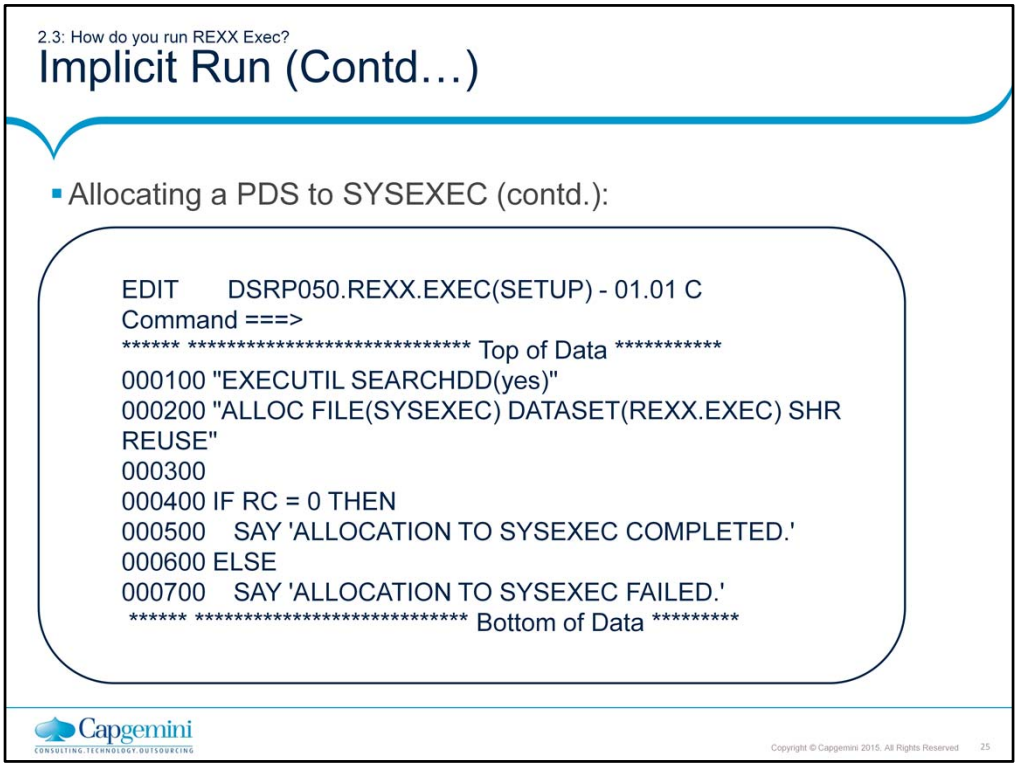
2.3: How do you run REXX Exec?

Implicit Run (Contd…)

- Allocating a PDS to SYSEXEC:
  - Create a member named SETUP in your exec PDS.
  - In SETUP member, issue ALLOCATE command to allocate your PDS to SYSEXEC.
  - The SETUP exec is shown on the next slide.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 24



How do you run REXX Exec – Implicit Run (contd.):

This exec shown in the slide is an example of how to allocate a private PDS named USERID.REXX.EXEC to SYSEXEC. Run it.

To make sure that SYSEXEC is available, the exec issues the following command:

```
EXECUTIL SEARCHDD(yes)
```

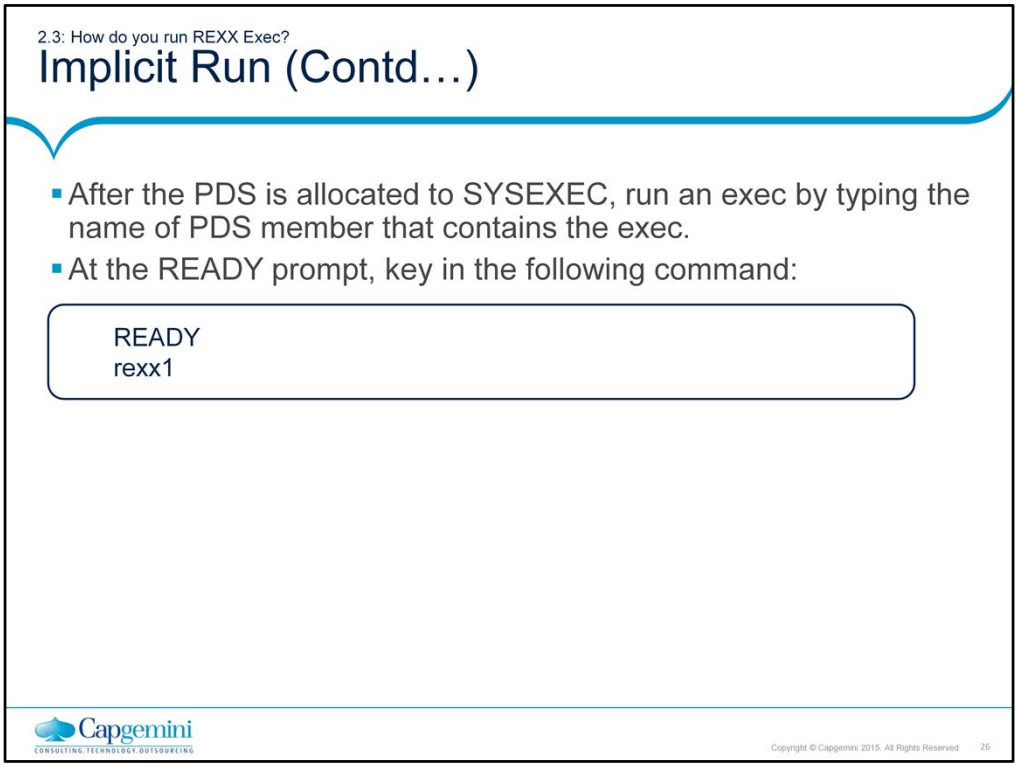
EXECUTIL SEARCHDD(yes/no)

This command specifies whether the system exec library (the default is SYSEXEC) should be searched when execs are implicitly invoked.

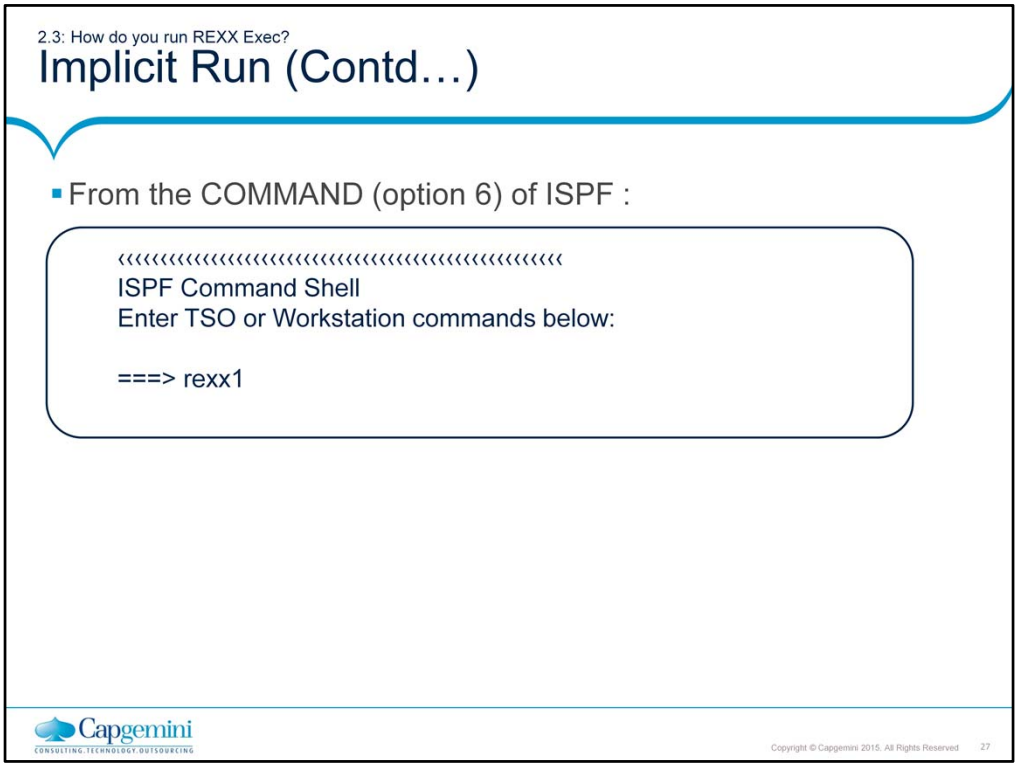
YES indicates that the system exec library (SYSEXEC) is searched, and if the exec is not found, SYSPROC is then searched.

NO indicates that SYSPROC only is searched.

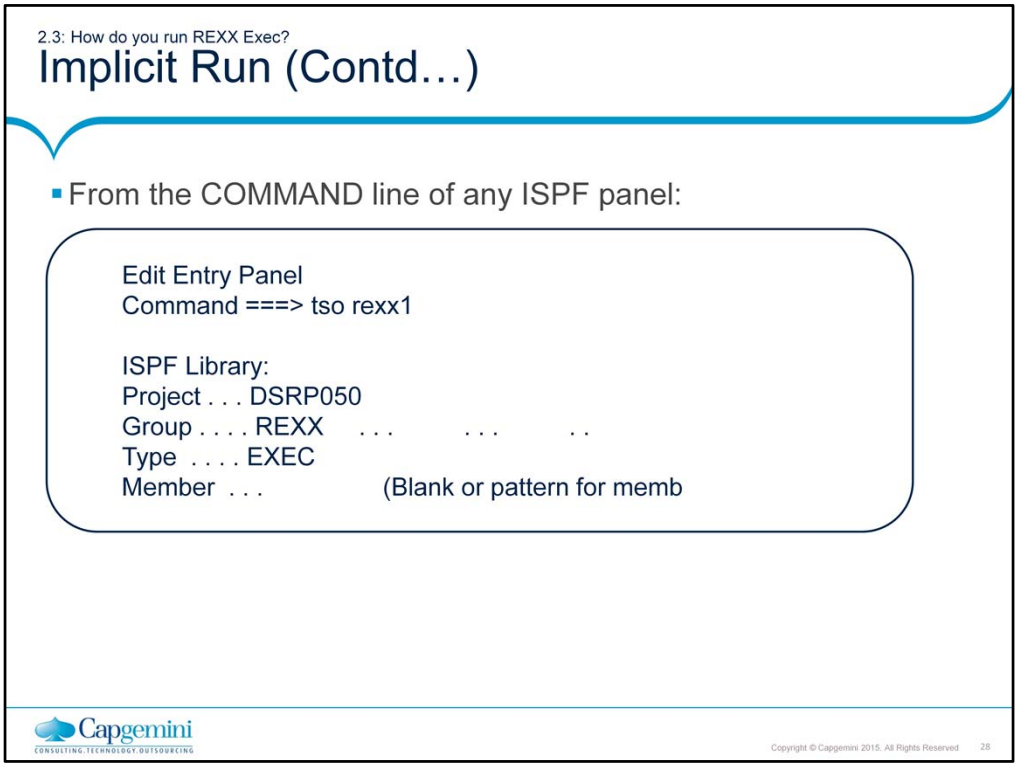
After the ALLOCATE command executes, a message indicates whether the command was successful or not.



Note: EXEC keyword is not required. Only PDS member name is mentioned.



Note: EXEC keyword is not required. Only PDS member name is mentioned.



How do you run REXX Exec – Implicit Run (contd.):

From the COMMAND line of any ISPF/PDF panel:

The PDS member name is preceded by the word “tso”. EXEC keyword is not required.

To reduce the search time for an exec that is executed implicitly and to differentiate it from a TSO/E command, precede the member name with a %:

```
READY
%rexx1
```

When a member name is preceded by %, TSO/E searches a limited number of system files for the name, thus reducing the search time.

Without the %, TSO/E searches several files before it searches SYSEXEC and SYSPROC to ensure that the name entered is not a TSO/E command.

2.3: How do you run REXX Exec?

Demo

- Running REXX Exec - Implicitly

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 29

2.4: Interpreting Error Messages

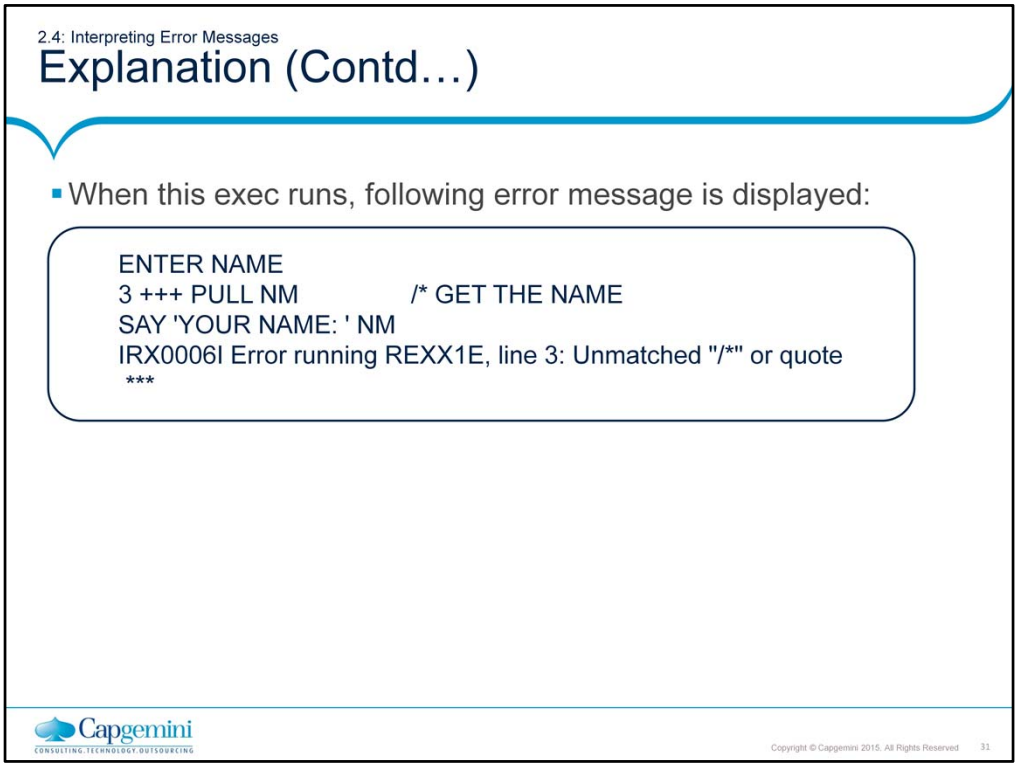
Explanation

- While running an exec that contains an error, an error message is displayed with corresponding line of exec.
- Error messages result from Syntax errors and Computation errors.
- Example of an exec with syntax error:

```
000100 /* REXX */
000110 SAY 'ENTER NAME'
000120 PULL NM              /* GET THE NAME
000130 SAY 'YOUR NAME: ' NM
****** ***************************** Bottom of Data **
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 30



Interpreting Error messages:

The exec runs until it detects the error, namely a missing */ at the end of the comment.

The SAY instruction displays the string (“Enter Name”), however it doesn’t wait for the user’s response because the next line of the exec contains the syntax error.

The exec ends and the language processor displays error messages.

The first error message begins with the line number of the statement where the error was detected, followed by three pluses (+++) and the contents of the statement (as shown below).

```
3 +++ PULL NM              /* GET THE NAME
SAY 'YOUR NAME: ' NM
```

The second error message begins with the message number followed by a message containing the exec name, line where the error was found, and an explanation of the error.

```
IRX0006I Error running REXX1E, line 3: Unmatched "/*" or
quote
```

2.4: Interpreting Error Messages

Demo

- Interpreting Error Messages

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 32

2.5: Passing data to an Exec

Explanation

- Terminal interaction (via keyboard):
  - PULL instruction to receive input

```
000100 /* REXX */
000110 SAY 'ENTER NUMBER1'
000120 PULL N1
000130 SAY 'ENTER NUMBER2'
000140 PULL N2
000150 SUM = N1 + N2
000160 SAY 'THE SUM IS ' SUM'.'
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 33

2.5: Passing data to an Exec

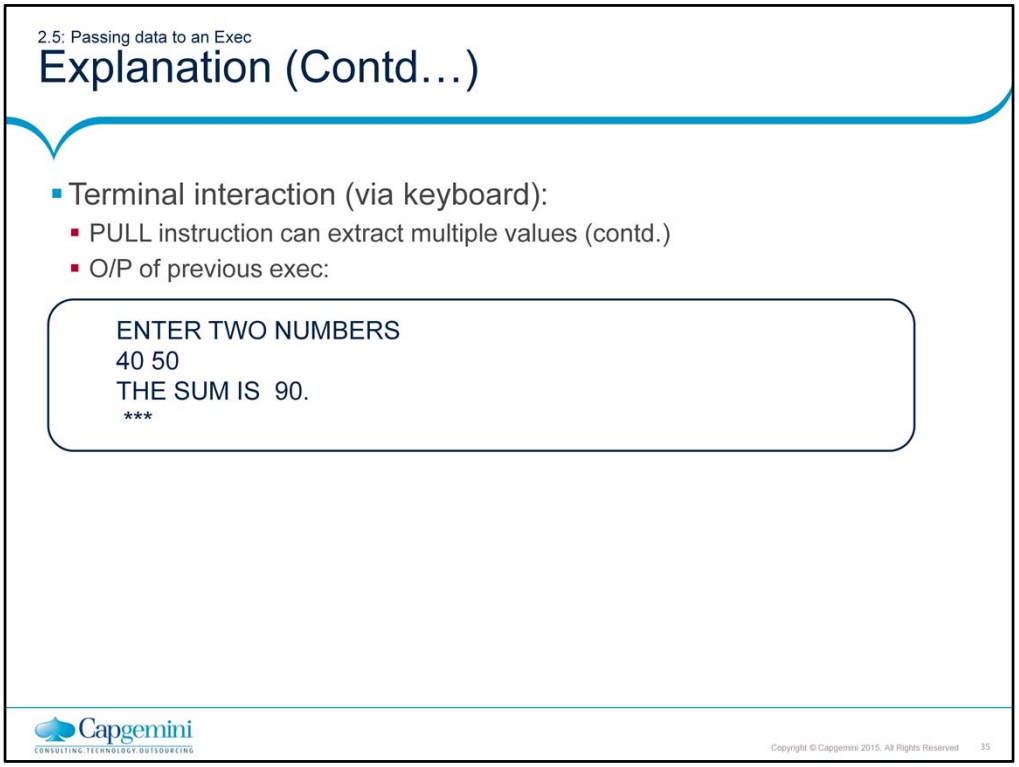
Explanation (Contd…)

- Terminal interaction (via keyboard):
  - PULL instruction can extract multiple values

```
000100 /* REXX */
000110 SAY 'ENTER TWO NUMBERS'
000120 PULL N1 N2
000150 SUM = N1 + N2
000160 SAY 'THE SUM IS ' SUM'.' .
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 34



Note: The PULL instruction can extract more than one value at a time from the terminal by separating a line of input, as shown in the above slide.

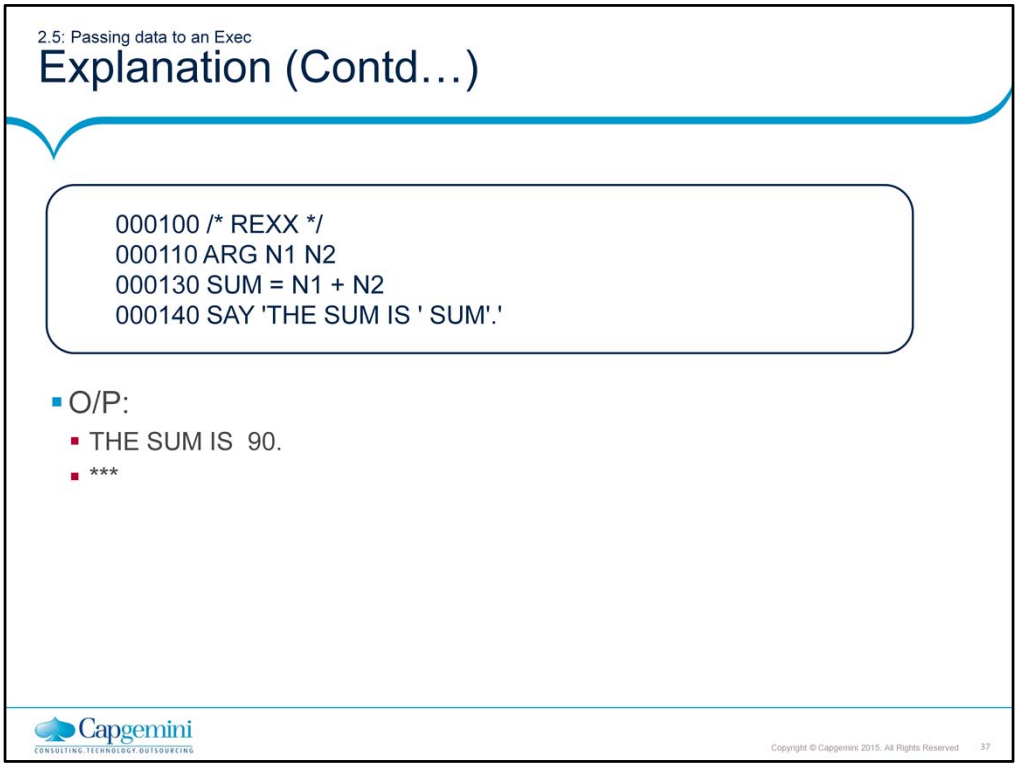
2.5: Passing data to an Exec

Explanation (Contd…)

- By specifying input values when invoking the exec:
  - Explicit: EXEC rexx.exec(rexx2C) '50 40' EXEC
  - Implicit: rexx2c 50 40
- These values are called arguments.
- The exec “rexx2c" uses the ARG instruction to assign the input to variables.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

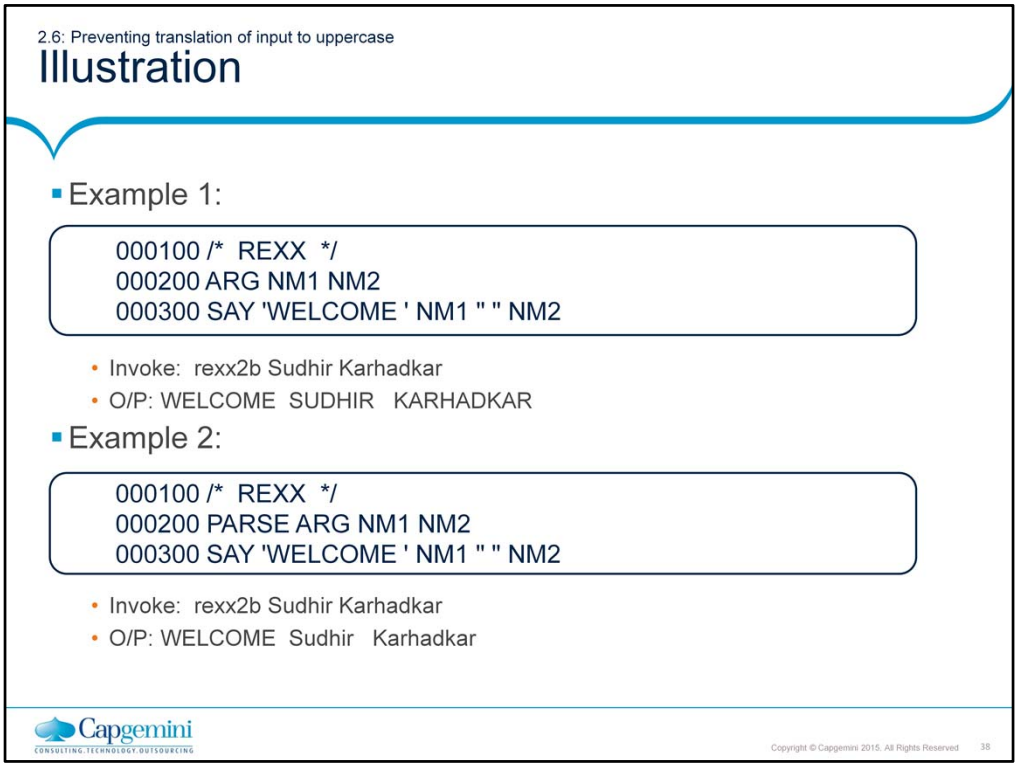
 36



Note:

ARG assigns the first number, namely 50, to N1, and the second number, namely 40, to N2.

If the number of values is fewer or more than the number of variable names after the PULL or the ARG instruction, errors can occur.



Note: Like the PULL instruction, the ARG instruction changes alphabetic characters to uppercase. To prevent translation to uppercase, precede ARG with PARSE as shown in the above slide.

2.6: Preventing translation of input to uppercase

Demo

- Passing data to an exec
- Preventing translation of input to uppercase

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 39

2.6: Preventing translation of input to uppercase

Lab

- Lab-book Lab-1: Coding and Running a REXX Exec

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 40

2.6: Preventing translation of input to uppercase

Summary

- In this lesson, we have learnt the following:
  - The concept of REXX Exec.
  - Syntax of REXX instruction.
  - The method of running an Exec.
  - The method of interpreting error messages.
  - The method of passing data to an Exec.

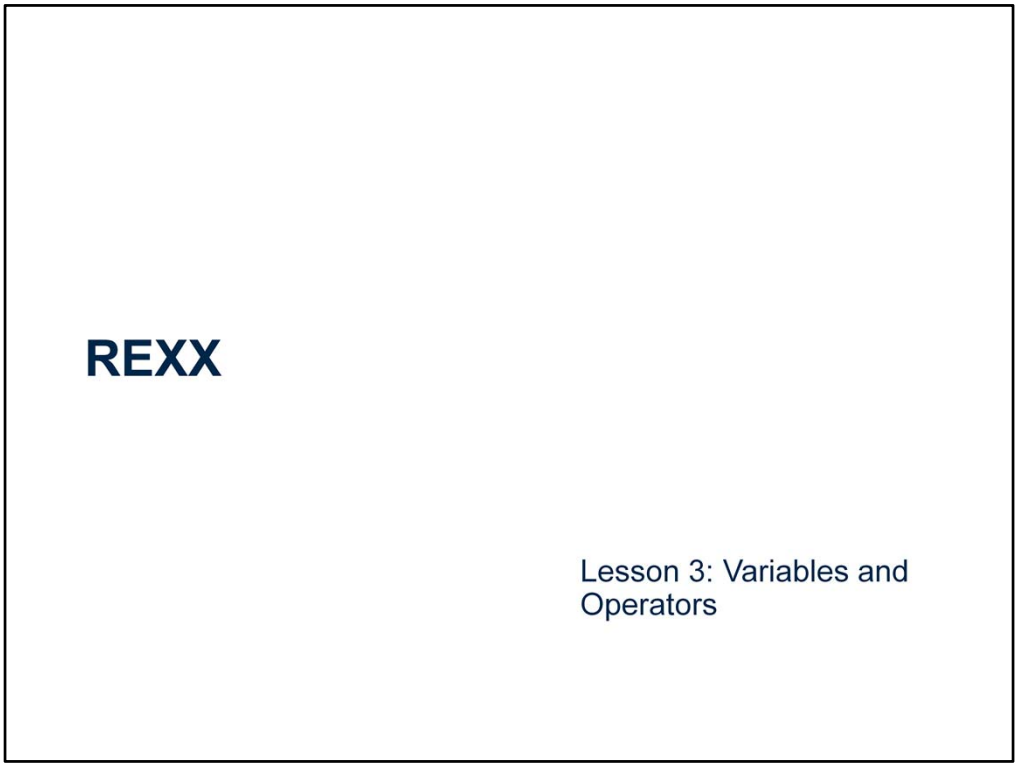
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 41

2.6: Preventing translation of input to uppercase

Review - Questions

- Question 1: For running REXX exec implicitly, PDS containing REXX execs is to be allocated to which system file?
  - Option 1: SYSRUN
  - Option 2: STEPLIB
  - Option 3: SYSEXEC
- Question 2: PULL instruction can extract multiple values.
  - True / False

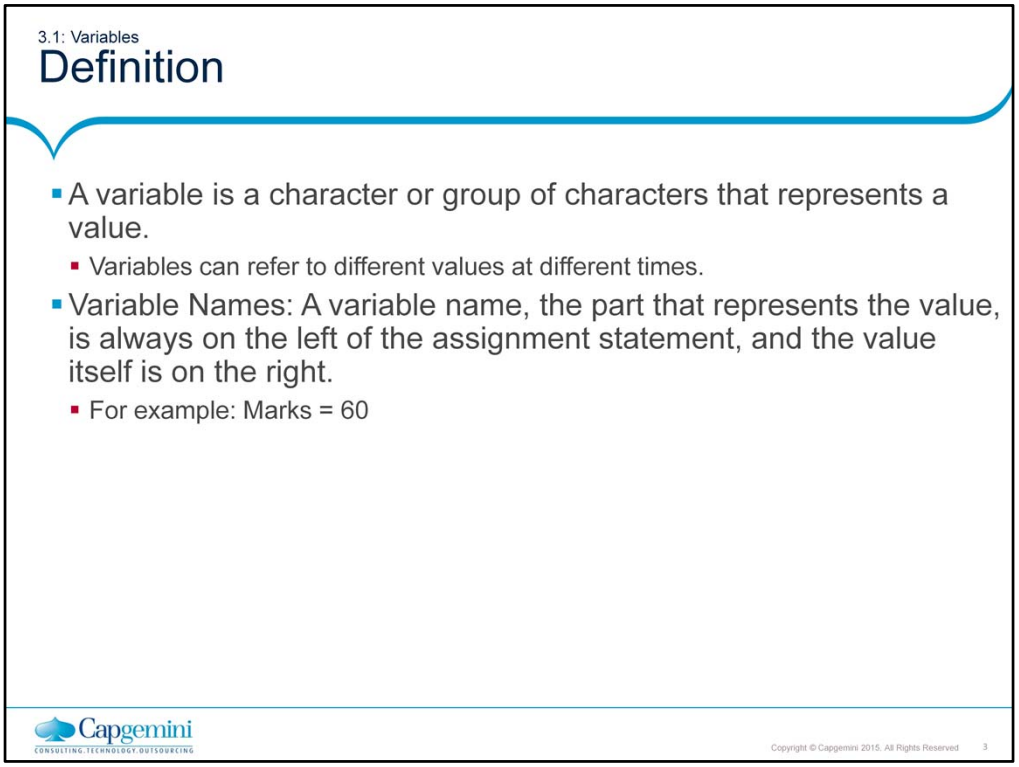

REXX
Lesson 3: Variables and Operators

Lesson Objectives

- In this lesson, you will learn the following topics:
  - Variables
  - Arithmetic operators
  - Comparison operators
  - Logical operators
  - Concatenation operator
  - Priority of operators

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 2


3.1: Variables
Definition
A variable is a character or group of characters that represents a value.
Variables can refer to different values at different times.
Variable Names: A variable name, the part that represents the value, is always on the left of the assignment statement, and the value itself is on the right.
For example: Marks = 60
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING
Copyright © Capgemini 2015. All Rights Reserved 3

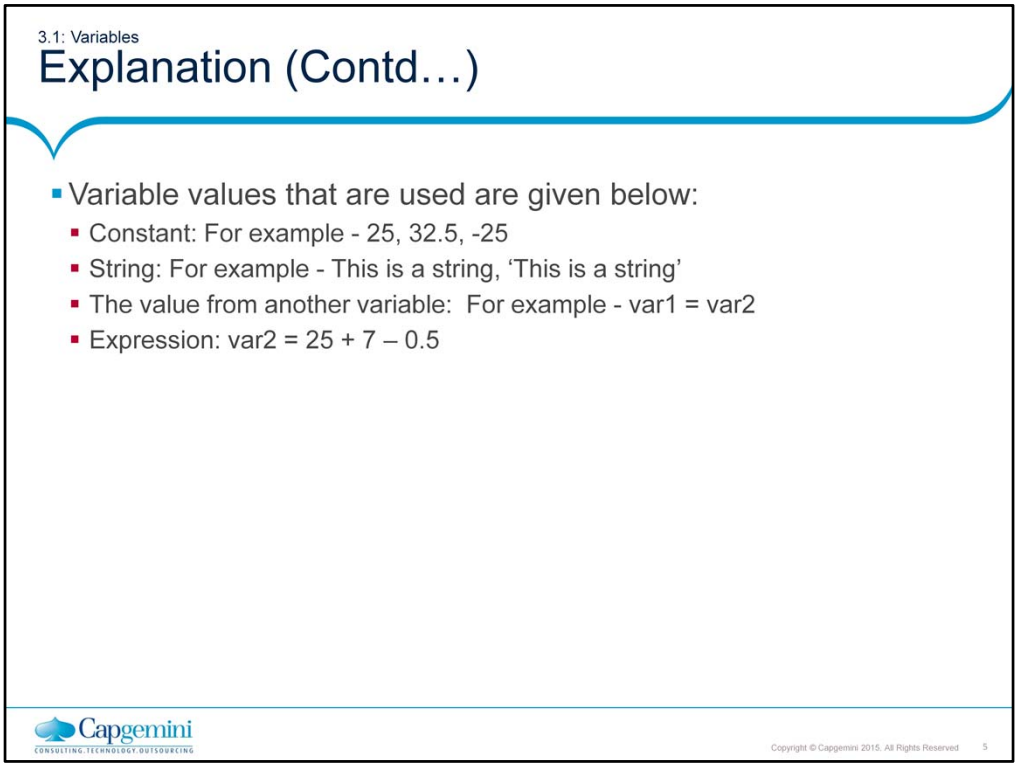
3.1: Variables

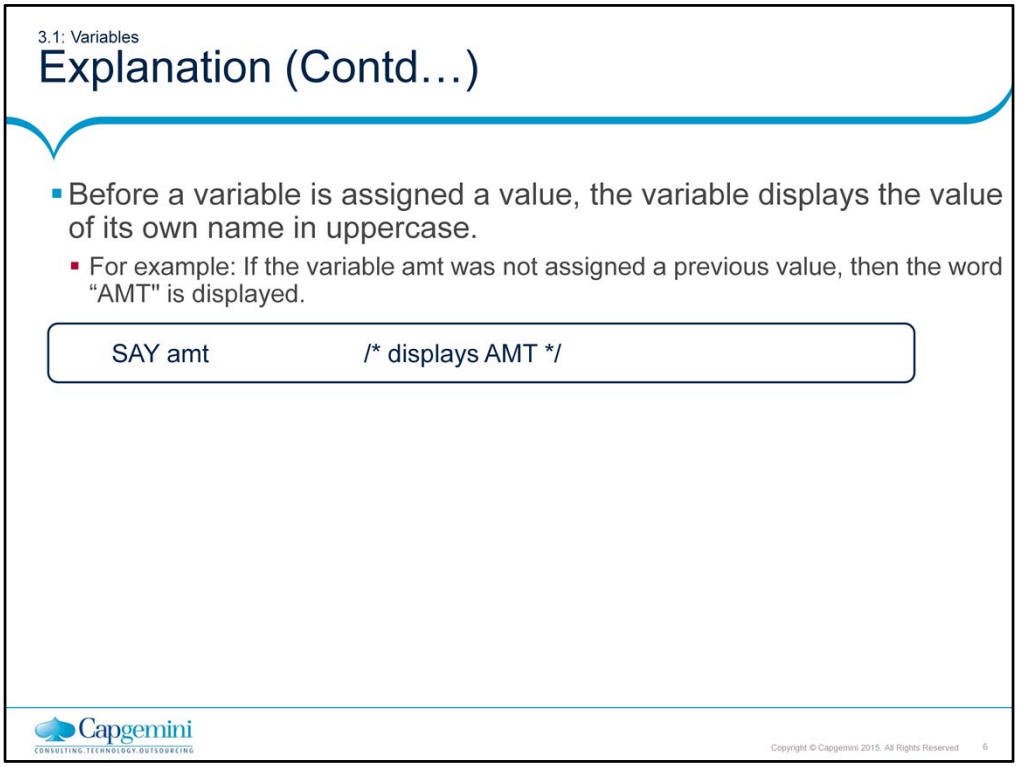
Explanation

- Variable Names consist of A .. Z, a .. z, 0 .. 9, @ # $ ¢ ? ! . _
- Restrictions on the variable name are given below:
  - The first character cannot be 0 thru 9 or a period (.)
  - The variable name cannot exceed 250 bytes.
  - The variable name should not be RC, SIGL, or RESULT, which are REXX special variables.
- Examples of variable names are given below:
  - ANSWER, B, Word3, total_amount, ?98B

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 4


3.1: Variables
Explanation (Contd…)
Variable values that are used are given below:
Constant: For example - 25, 32.5, -25
String: For example - This is a string, 'This is a string'
The value from another variable: For example - var1 = var2
Expression: var2 = 25 + 7 – 0.5
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING
Copyright © Capgemini 2015. All Rights Reserved 5


3.1: Variables
Explanation (Contd…)
Before a variable is assigned a value, the variable displays the value of its own name in uppercase.
For example: If the variable amt was not assigned a previous value, then the word “AMT" is displayed.
SAY amt /* displays AMT */
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING
Copyright © Capgemini 2015. All Rights Reserved 6

3.2: Expressions

Definition

- An expression is something that needs to be calculated and consists of numbers, variables, or strings, and one or more operators.
- The operators determine the kind of calculation to be done on the numbers, variables, and strings.
  - There are four types of operators:
    - arithmetic
    - comparison
    - logical, and
    - concatenation

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 7

3.3: Arithmetic operators

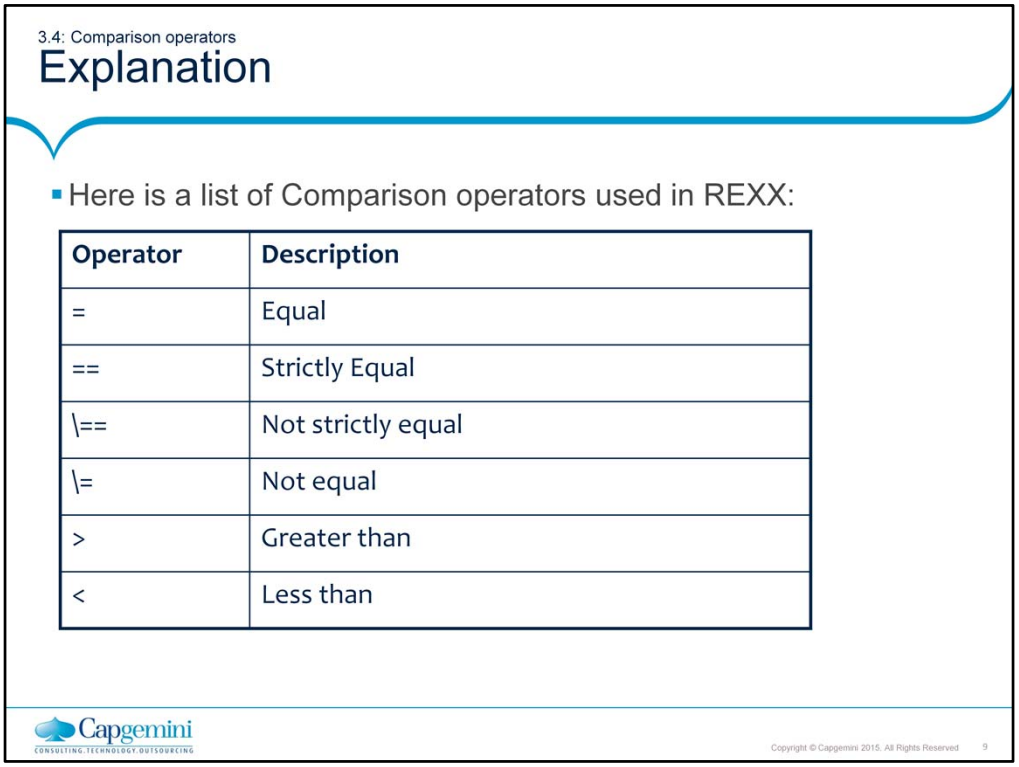
Explanation

- Here is the list of Arithmetic operators used in REXX:

Operator	Description
+	Add
-	Subtract
*	Multiply
/	Divide
%	Divide and return a whole number with out a remainder
//	Divide and return the remainder only
**	Raise a number to a whole number power
-number	Negate the number
+number	Add the number to 0

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 8



Comparison operators:

Expressions that use comparison operators do not return a number value as do the arithmetic expressions. Comparison expressions return either a TRUE or FALSE response in terms of 1 or 0 as follows:

1 True

0 False

The Strictly Equal and Equal Operators:

When two expressions are strictly equal, everything including the blanks and case (when the expressions are characters) is exactly the same.

When two expressions are equal, they are resolved to be the same.

The following expressions are all TRUE.

```
'WORD' = word          /* returns 1 */
'word ' \== word       /* returns 1 */
'word' == 'word'       /* returns 1 */
```

3.4: Comparison operators

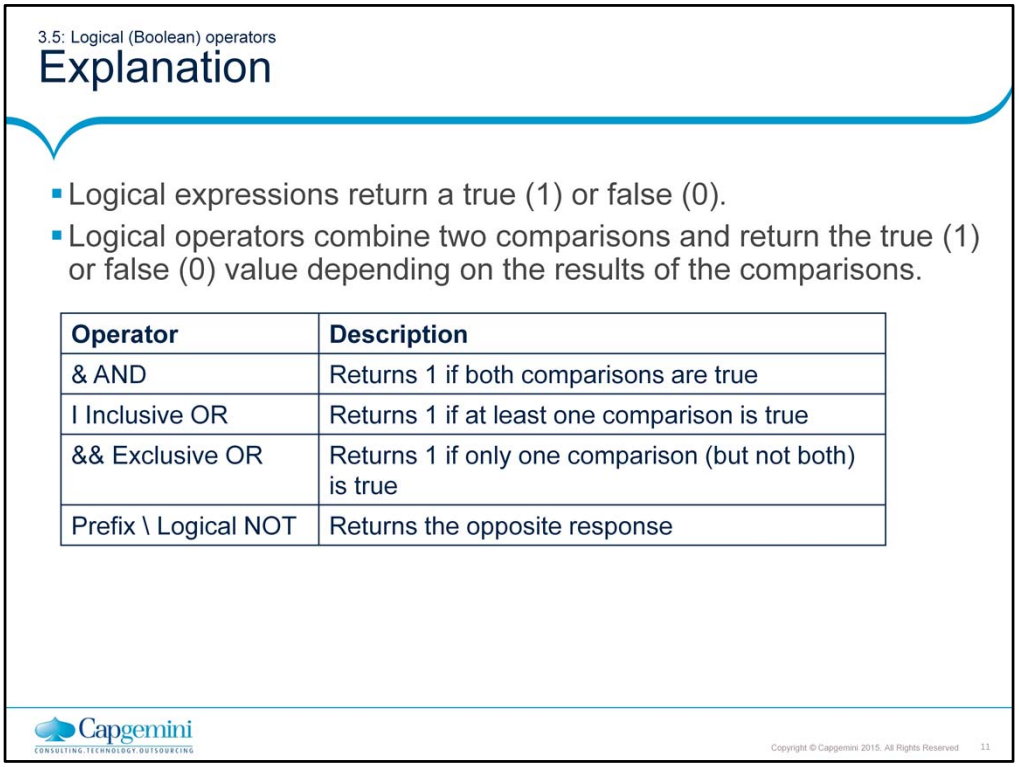
Explanation (Contd…)

- The not character, namely "¬", is synonymous with the backslash ("\"). The two characters may be used interchangeably.

Operator	Description
><	Greater than or less than (same as not equal)
>=	Greater than or equal to
\<	Not less than
<=	Less than or equal to
\>	Not greater than

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 10



Logical (Boolean) operators:

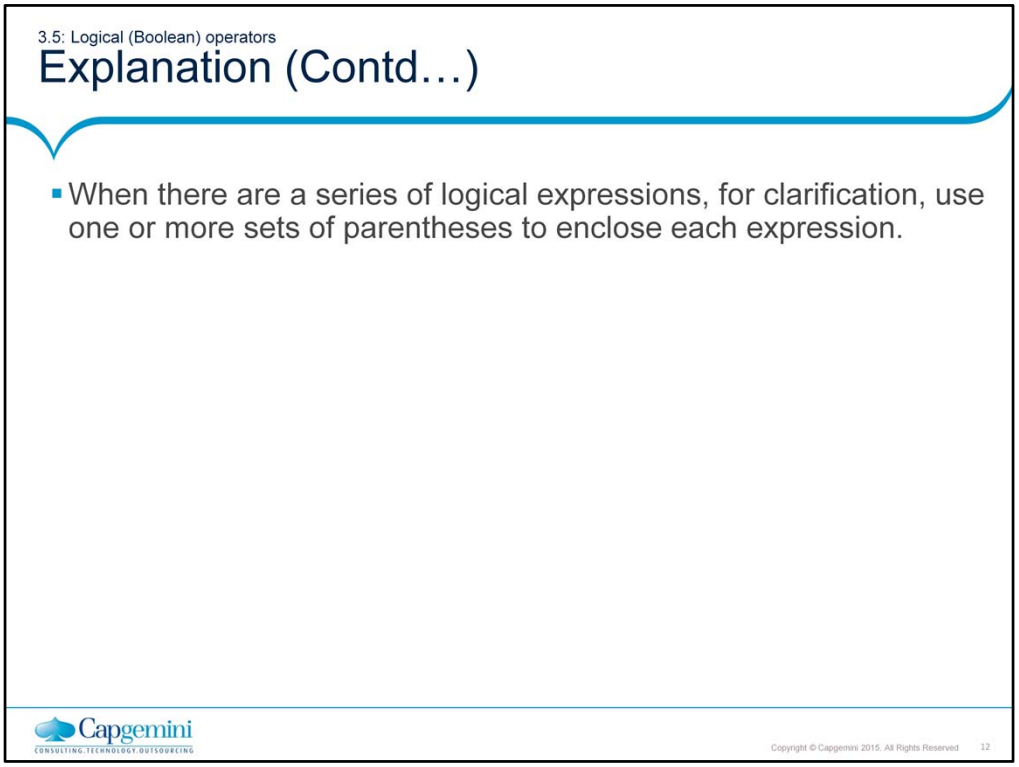
Example:

```
/****REXX ****/
PARSE ARG season snowing broken_leg
IF ((season = 'winter') | (snowing ='yes')) & (broken_leg ='no')
THEN
        SAY 'Go skiing.'
ELSE
        SAY 'Stay home.'
```

When arguments passed to this example are "spring yes no", then the output is as shown below:

- - - - - - -

Go skiing.



Logical (Boolean) operators:

& AND:

```
(4>2) & (a=a)  /* true, so result is 1*/
(2>4) & (a=a)  /* false, so result is 0*/
```

|Inclusive OR:

```
(4>2) | (5=3)  /* at least one is true, so result is 1*/
(2>4) | (5=3)  /* neither one is true, so result is 0*/
```

|Exclusive OR:

```
(4>2) && (5=3)  /* only  one is true, so result is 1*/
(2<4) && (5=5)  /* both are true, so result is 0*/
```

Prefix \ Logical NOT:

```
\0        /* opposite of zero, so result is 1*/
\(4>2)    /* opposite of true, so result is 0*/
```

3.6: Concatenation operators

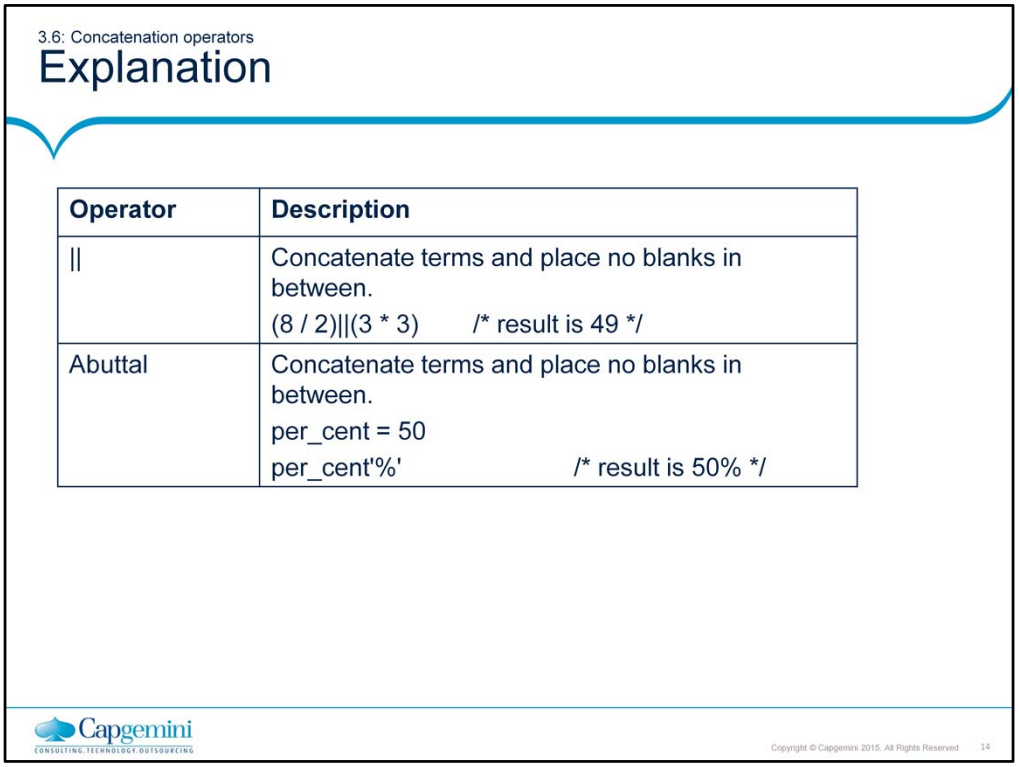
Definition

- Concatenation operators combine two terms into one. The terms can be strings, variables, expressions, or constants.
- Concatenation can be used in formatting output.

Operator	Description
blank	Concatenate terms and place one blank in between. Terms that are separated by more than one blank default to one blank when read. SAY hi sudhir /* result is HI SUDHIR */

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 13



Concatenation operators:

Example:

```
/**** REXX ****/
sport = 'base'
equipment = 'ball'
column = '     '
cost = 5
SAY  sport||equipment  column  '$' cost
```

- - - - - - - - - - -

The output is:

baseball $ 5

3.7: Priority of operators

Explanation

Operator	Description
\ - +	Prefix operators
**	Power (exponential)
* / % //	Multiply and divide
+ -	Add and subtract
Blank || abuttal	Concatenation operators
== = >< etc.	Comparison operators
&	Logical AND
| &&	Inclusive OR and exclusive OR

Capgemini
CONSULTING. TECHNOLOGY. OUTSOURCING

 15

Demo

- Demonstration of the use of Comparison, Logical and Concatenation operators

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 16

Lab

- Lab-book Lab-2: Operators

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 17

Summary

- In this lesson you have learnt the following concepts:
  - Variables
  - Arithmetic operators
  - Comparison operators
  - Logical operators
  - Concatenation operator
  - Priority of operators

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 18

Review – Questions

- Question 1: Arithmetic operator // does following:
  - Option 1: Divides and returns division as a whole number
  - Option 2: Divides and returns the remainder only
  - Option 3: Divides two times
- Question 2: Multiply / Divide has higher priority over Add /subtract.
  - True / False
- Question 3: Comparison expressions return either ___ or ___ .

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 19

CLIST

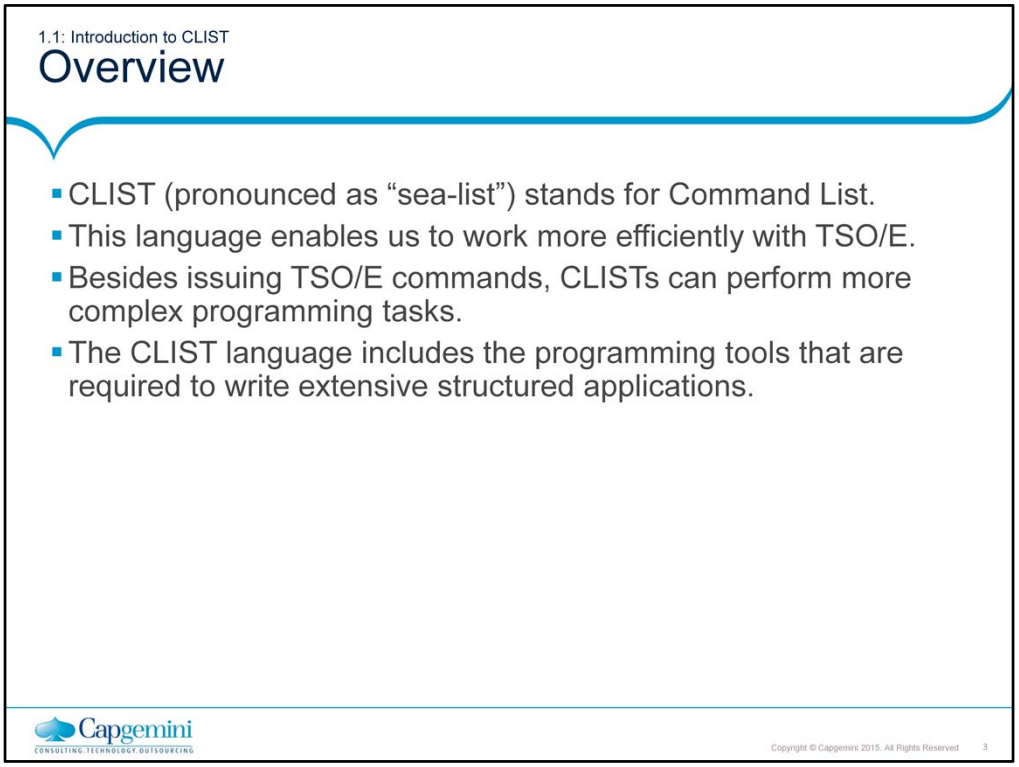
Lesson 1: Introduction to CLIST

Lesson Objectives

- To understand the following topics:
  - Features of CLIST language
  - Categories of CLIST

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 2


1.1: Introduction to CLIST
Overview
CLIST (pronounced as “sea-list”) stands for Command List.
This language enables us to work more efficiently with TSO/E.
Besides issuing TSO/E commands, CLISTs can perform more complex programming tasks.
The CLIST language includes the programming tools that are required to write extensive structured applications.
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING
Copyright © Capgemini 2015. All Rights Reserved 3

1.1: Introduction to CLIST

Overview

- CLISTs can perform any number of complex tasks from displaying a series of full-screen panels to managing programs written in other languages.
- It is an interpretive language.
- Like programs in other high-level interpretive languages, CLISTs are easy to write and test.
- You do not have to compile and link-edit them.
- To test a CLIST, you execute it, correct errors if any, and re-execute it.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 4

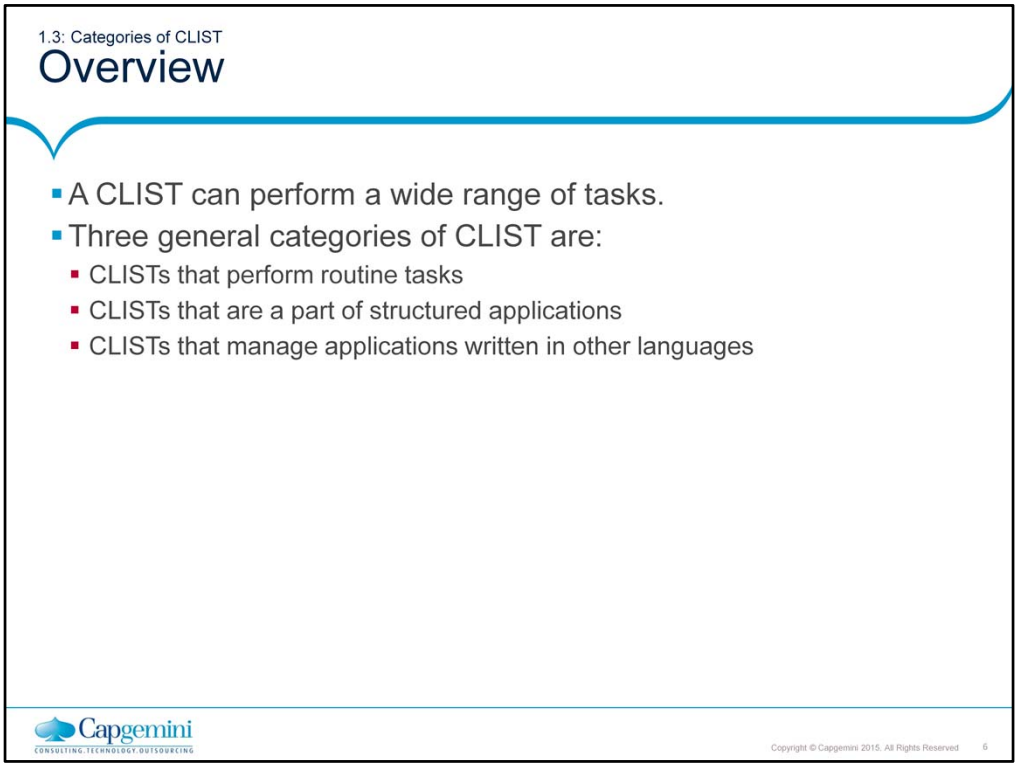
1.2: Features of CLIST

Salient Features

- The CLIST language provides a wide range of programming functions. Its features include:
  - an extensive set of “arithmetic operators” and “logical operators” for processing numeric data
  - “string-handling functions” for processing character data
  - “CLIST statements” that let the user:
    - structure your programs
    - perform I/O
    - define and modify variables, and
    - handle errors and attention interrupts

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 5


1.3: Categories of CLIST
Overview
A CLIST can perform a wide range of tasks.
Three general categories of CLIST are:
CLISTs that perform routine tasks
CLISTs that are a part of structured applications
CLISTs that manage applications written in other languages
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING
Copyright © Capgemini 2015. All Rights Reserved 6

1.3: Categories of CLIST

Explanation

- As a user of TSO/E, you probably perform certain tasks on a regular basis.
  - These tasks may involve, entering TSO/E commands:
    - to check on the status of datasets
    - to allocate datasets for particular programs, and
    - to print files
- You can write CLISTs that significantly reduce the amount of time that you have to spend on these routine tasks.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 7

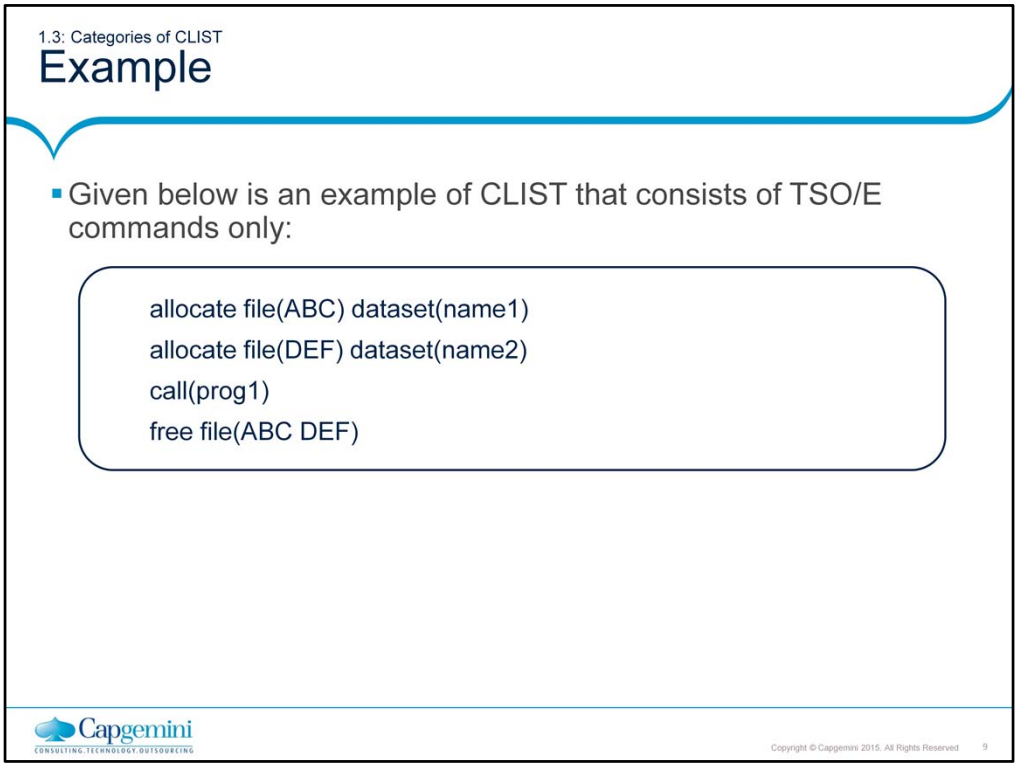
1.3: Categories of CLIST

Explanation

- By grouping together the instructions required to complete a task into a CLIST, you can reduce:
  - the time
  - the number of key strokes and
  - the errors involved in performing the task
- Thus using CLIST, you can increase your productivity.
  - Such a CLIST can consist of:
    - only TSO/E commands, or
    - combination of TSO/E commands, JCL statements, or CLIST statements.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 8



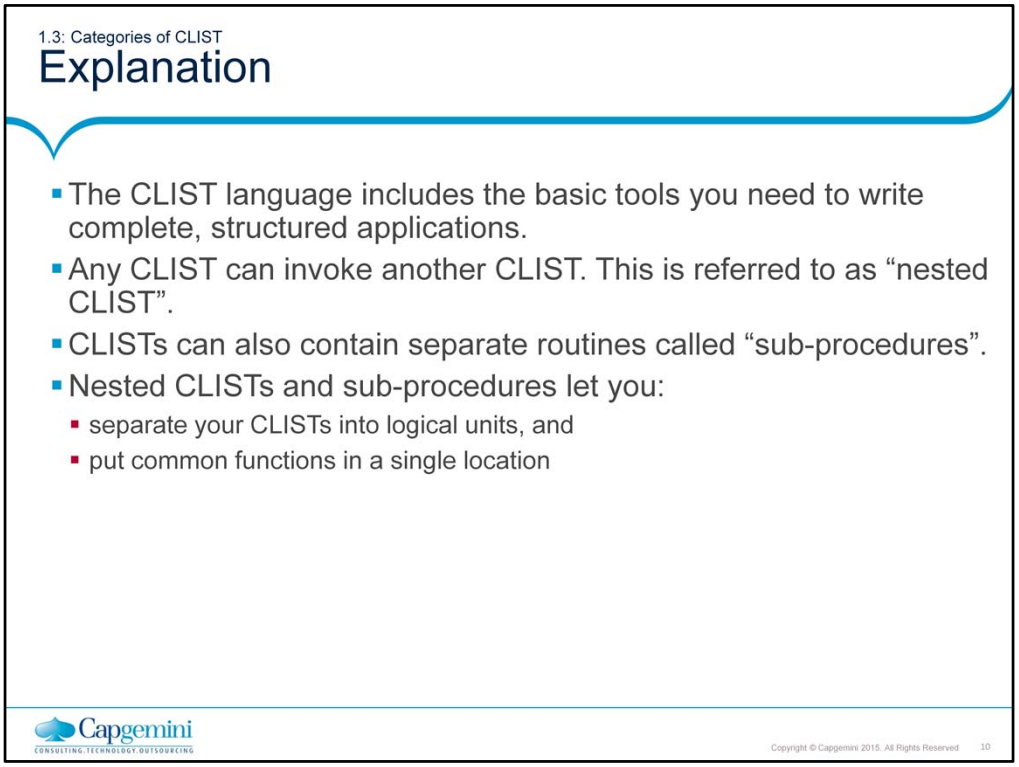
Example:

The CLIST shown in the slide issues TSO/E commands:

- to allocate files for a program
- to call the program, and
- to free the files when the program is finished

Whenever you wanted to perform these related tasks, you can simply execute the CLIST instead of retyping the commands.

If tasks require specific input from a user, you can obtain the input in a CLIST by using CLIST statements, or TSO/E commands to prompt the user for the input.



CLISTs that are a part of structured applications:

Specific CLIST statements let you:

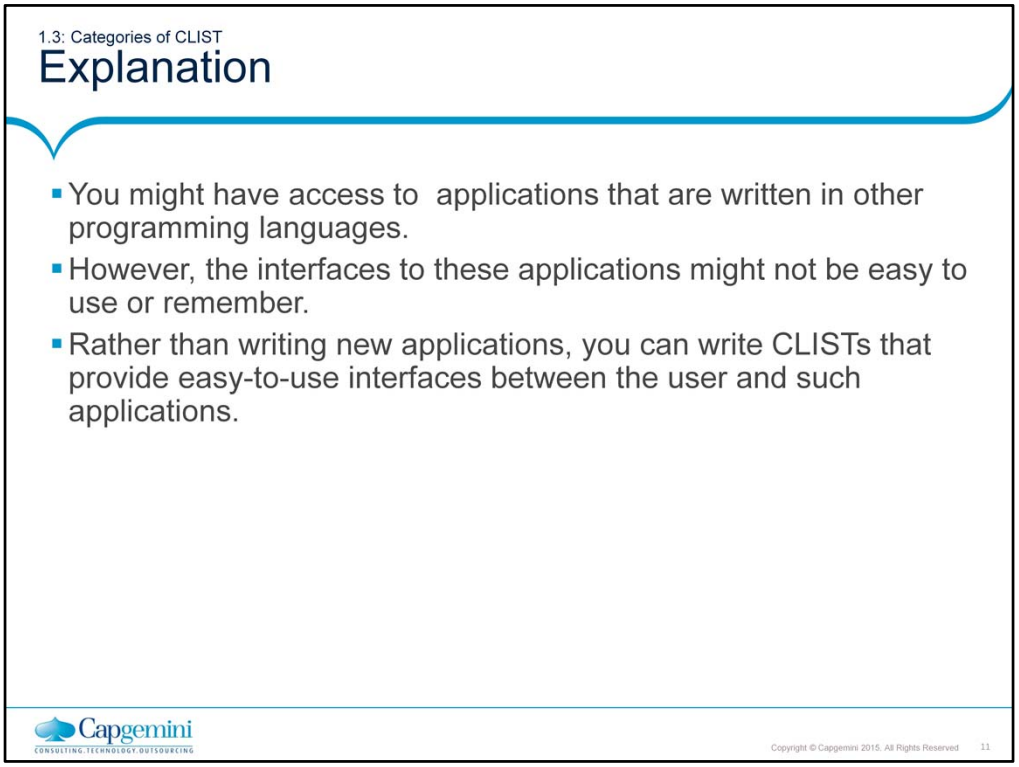
- Define common data for sub-procedures and nested CLISTs
- Restrict data to certain sub-procedures and CLISTs
- Pass specific data to a sub-procedure or nested CLIST

For interactive applications, CLISTs can issue commands of the Interactive System Productivity Facility (ISPF) to display full-screen panels.

Conversely, ISPF panels can invoke CLISTs, based on input that a user types on the panel.

- When the user changes a value on a panel, the change applies to the value in the CLIST that displayed the panel.

With ISPF, the CLISTs can manage extensive panel-driven dialogs.



CLISTs that manage applications written in other languages:

A CLIST can send messages to, and receive messages from the terminal to determine what the user wants to do.

Then based on this information, the CLIST can set up the environment, and issue the commands required to invoke the program that performs the requested tasks.

Summary

- In this lesson, you have learnt about:
  - the features of CLIST language
  - the Categories of CLIST language

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 12

Review Questions

- Question 1: A CLIST which invokes another CLIST is called as:
  - Option 1: Function
  - Option 2: Routine
  - Option 3: Nested CLIST
  - Option 4: Structured CLIST

- Question 2: CLIST programs have to be compiled.
  - True / False
- Question 2: CLISTs can contain separate routines called ___.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 13

CLIST

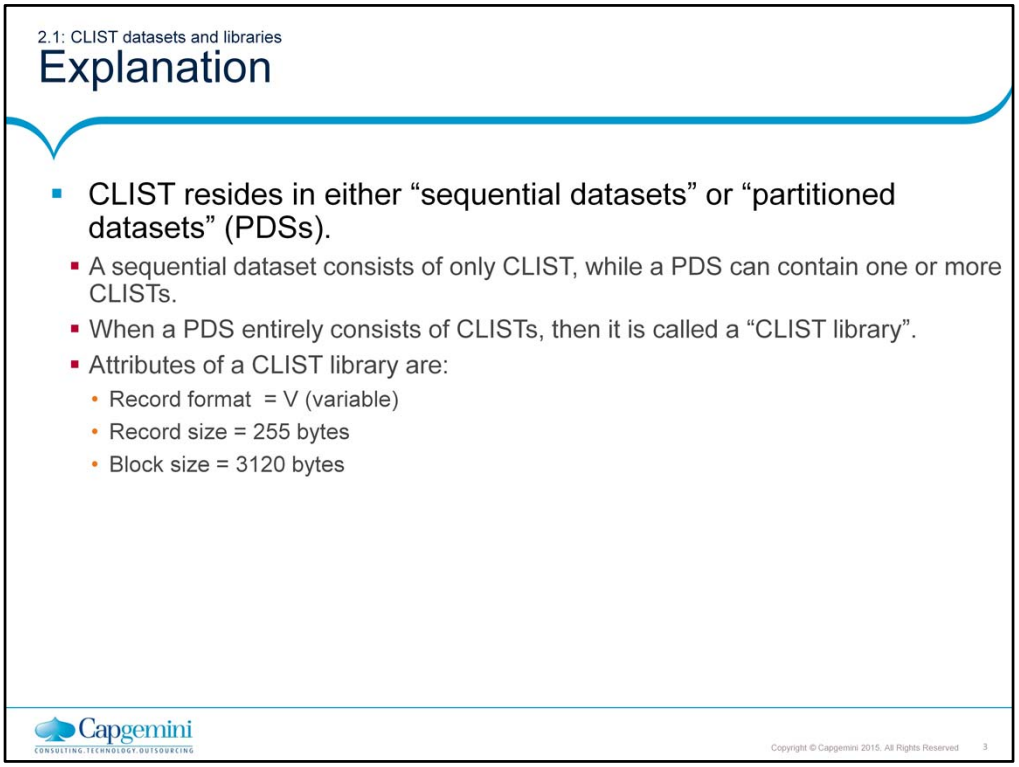
Lesson 2: Creating, Editing and Executing CLISTs

Lesson Objectives

- To understand the following topics:
  - Creating, editing, and executing CLIST
  - Passing parameters to CLIST
  - Using the ALTLIB command

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 2



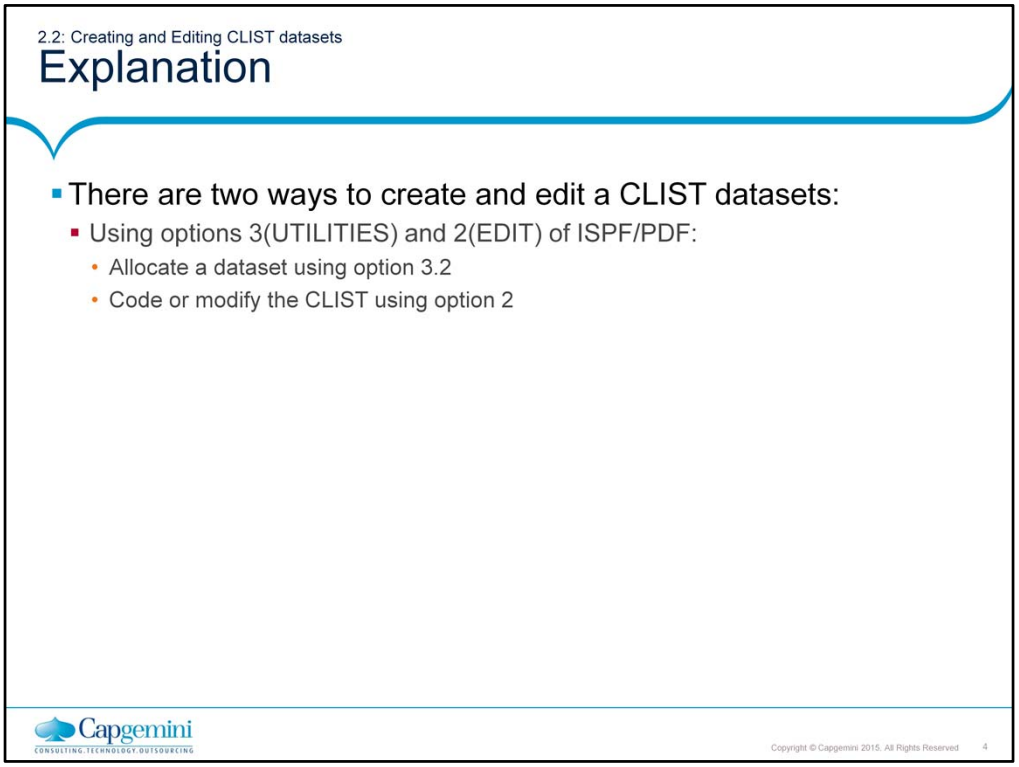
CLIST datasets and libraries:

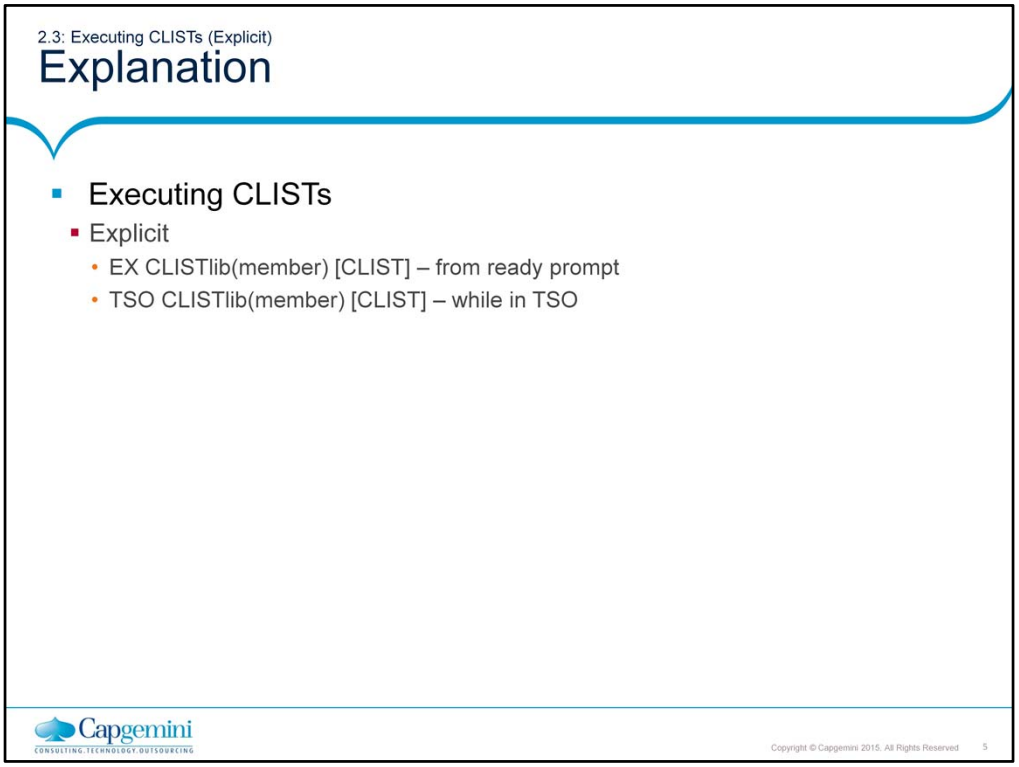
CLIST libraries make CLISTs easy to maintain and execute.

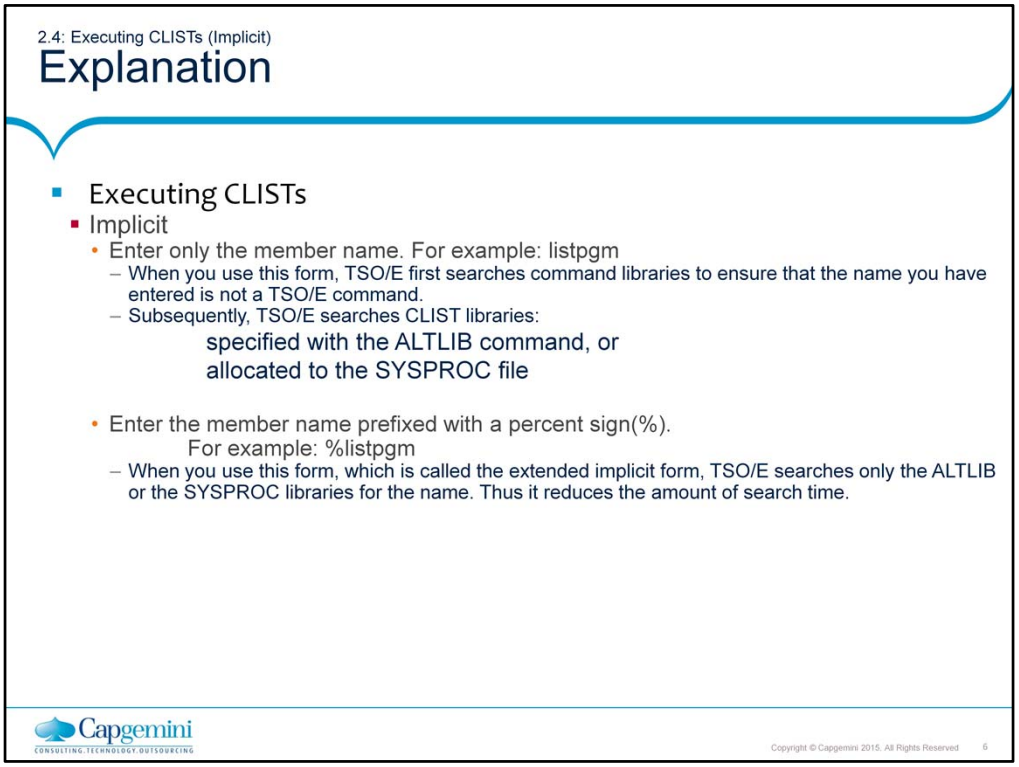
Your installation can keep commonly used CLISTs in a “system CLIST library”, and you can keep your own CLISTs in a “private CLIST library”.

If you allocate a CLIST library to the file SYSPROC, or specify the library on the ALTLIB command, you can implicitly execute the CLISTs by simply typing their member names.

Implicit execution, frees you from having to code the name of the CLIST library on an EXEC command.


2.2: Creating and Editing CLIST datasets
Explanation
There are two ways to create and edit a CLIST datasets:
Using options 3(UTILITIES) and 2(EDIT) of ISPF/PDF:
Allocate a dataset using option 3.2
Code or modify the CLIST using option 2
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING
Copyright © Capgemini 2015. All Rights Reserved 4


2.3: Executing CLISTs (Explicit)
Explanation
Executing CLISTs
Explicit
EX CLISTlib(member) [CLIST] – from ready prompt
TSO CLISTlib(member) [CLIST] – while in TSO
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING
Copyright © Capgemini 2015. All Rights Reserved 5


2.4: Executing CLISTs (Implicit)
Explanation
Executing CLISTs
Implicit
Enter only the member name. For example: listpgm
When you use this form, TSO/E first searches command libraries to ensure that the name you have entered is not a TSO/E command.
Subsequently, TSO/E searches CLIST libraries:
specified with the ALTLIB command, or
allocated to the SYSPROC file
Enter the member name prefixed with a percent sign(%).
For example: %listpgm
When you use this form, which is called the extended implicit form, TSO/E searches only the ALTLIB or the SYSPROC libraries for the name. Thus it reduces the amount of search time.
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING
Copyright © Capgemini 2015. All Rights Reserved
6

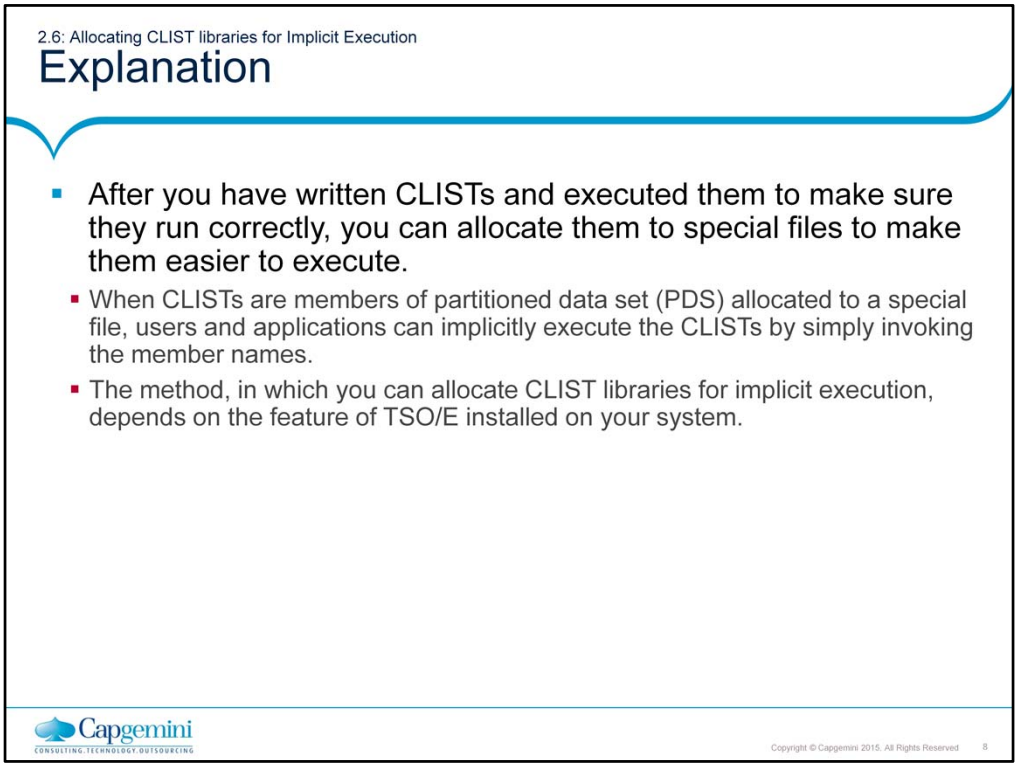
2.5: Passing Parameters to a CLIST

Explanation

- To pass parameters to a CLIST, include them on the EXEC command or sub-command as follows:
  - For the explicit form, pass parameters in single quotes.
    For example: clistname ‘parm1 parm2’
  - For the implicit or the extended implicit form, omit the quotes.
    For example: %clistname parm1 parm2

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 7

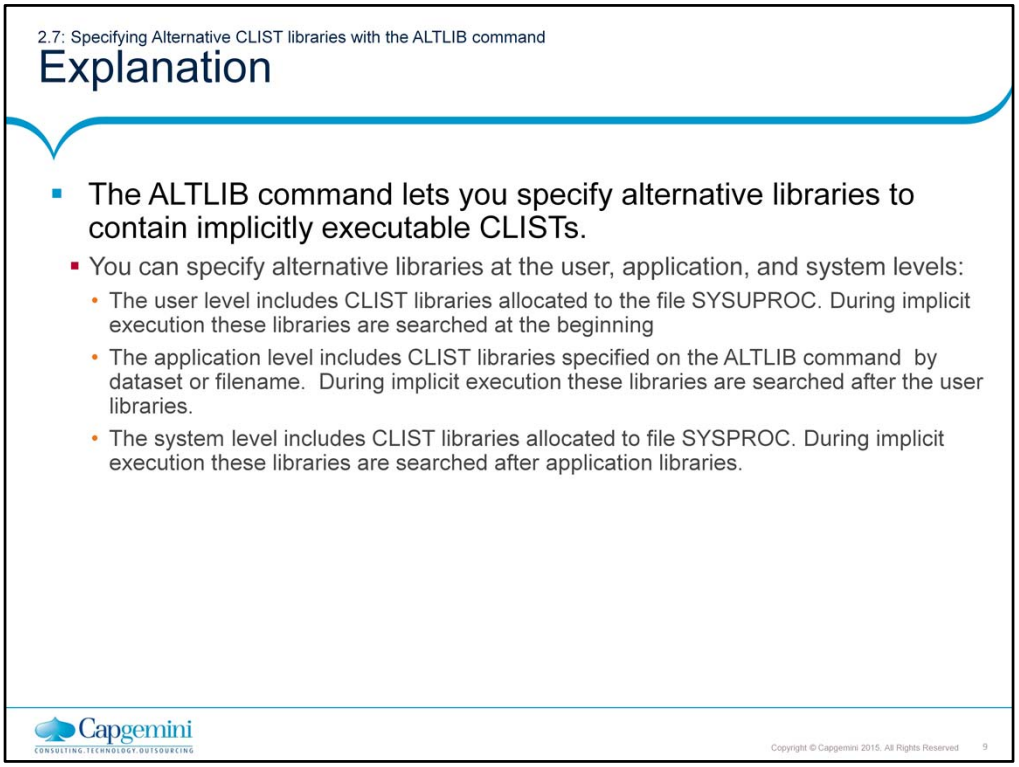


Allocating CLIST Libraries for Implicit Execution:

The ALTLIB command gives you more flexibility in specifying CLIST libraries for implicit execution.

With ALTLIB, a user or ISPF application can easily activate and deactivate CLIST libraries for implicit execution as the need arises.

This flexibility can result in less search time when fewer CLISTs are activated for implicit execution at the same time.



Specifying Alternative CLIST libraries with the ALTLIB command:

Examples of the ALTLIB Command:

In the following example, an application issues the ALTLIB command to allow implicit execution of CLISTs in the data set NEW.CLIB, to be searched ahead of SYSPROC.

```
ALTLIB ACTIVATE APPLICATION(CLIST) DATASET(new.clib)
```

The application can also allow searching for any private CLISTs, which the user has allocated to the file SYSUPROC, with the following command:

```
ALTLIB ACTIVATE USER(CLIST)
```

To display the active libraries in their current search order, use the DISPLAY operand as follows:

```
ALTLIB DISPLAY
```

2.8: Example

Illustration

- Here is an example of ALTLIB command:
  - altlib activate application(clist) dataset('DSRP035.VANDANA.CLIST')
  - Subsequently, you can execute a CLIST from the command prompt as tso exec 'DSRP035.VANDANA.clist(addnum)'

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 10

2.9: Workout on CLISTs

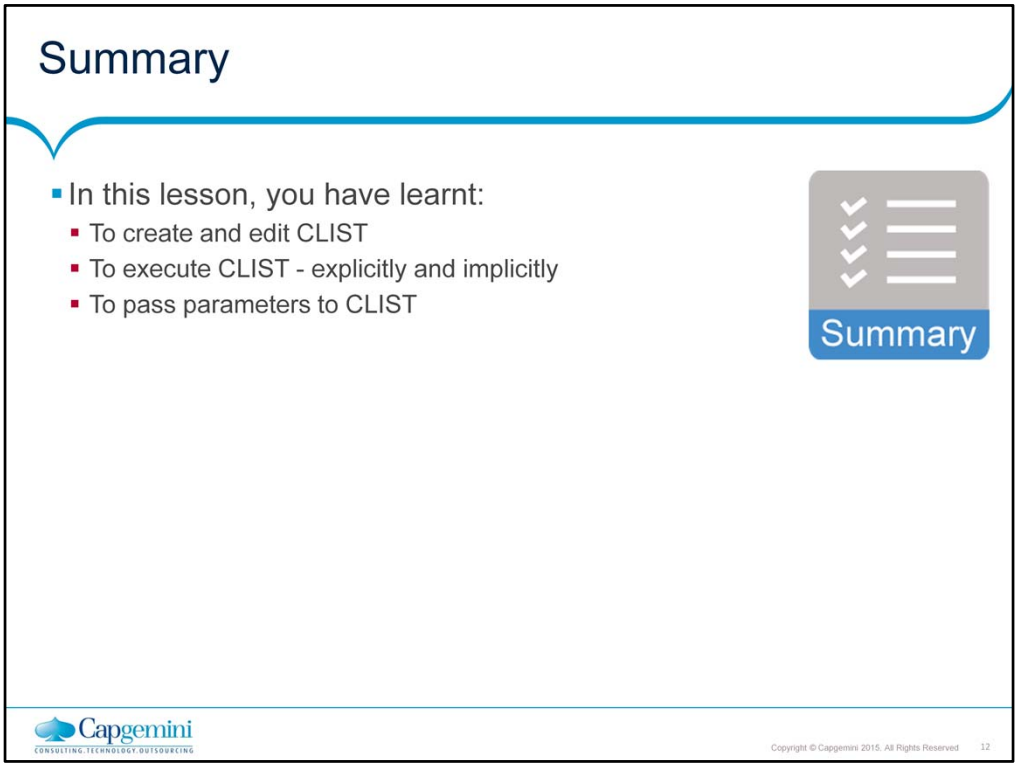
Lab

- Lab session

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 11

Add the notes here.



Add the notes here.

Review Question

- Question 1: To execute implicitly a CLIST, the CLIST name may be
  - preceded by:
  - Option 1: &
  - Option 2: *
  - Option 3: %
  - Option 4: #
- Question 2: CLISTs can reside only in a PDS.
  - True/False

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 13

Add the notes here.

CLIST

Lesson 3: Writing CLISTs – Syntax and Conventions

Lesson Objectives

- To understand the following topics:
  Syntax rules regarding writing CLIST statements
  - Operators and Expressions in CLIST
    - Text to come here (Arial 16, Normal)
  - Text to come here (Arial 14, Normal)
  - Text to come here (Arial 12, Normal)

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 2

3.1: Overview of CLIST statements

Explanation

- CLIST statements set controls, assign values to variables, monitor the conditions under which CLISTs execute and perform I/O
- CLIST statements execute both in the command and sub-command environment (under the TSO/E EXEC command and the EXEC sub-command of TSO/E EDIT).
- They fall into different categories (discussed on the next slide).

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 3

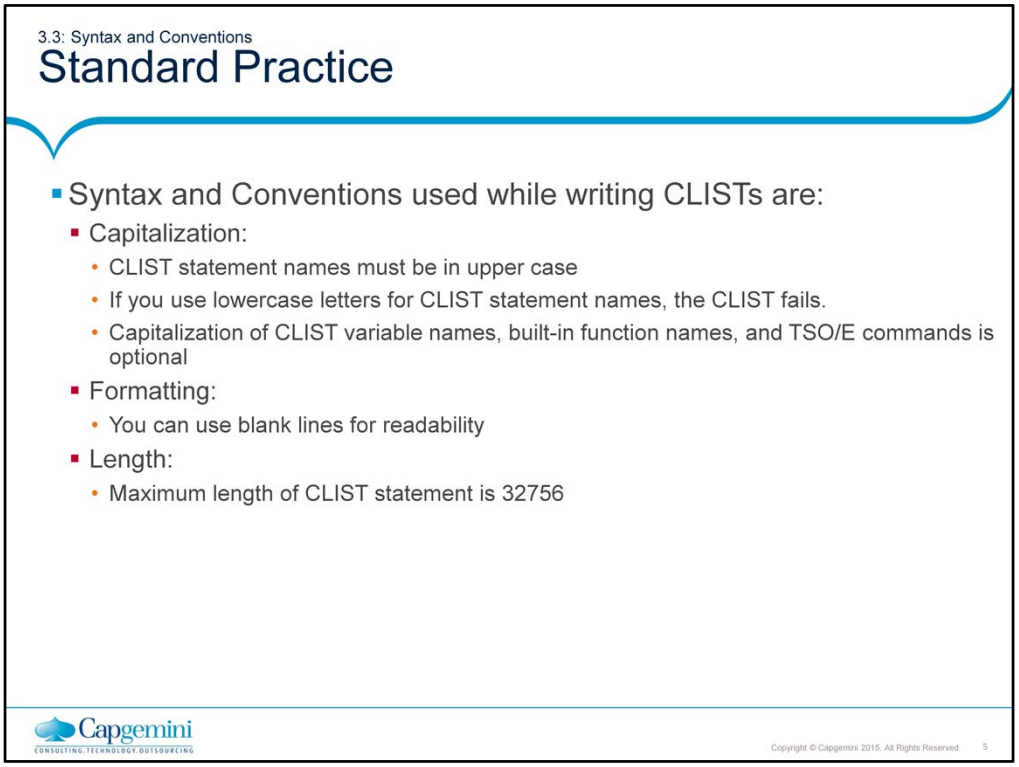
3.2: CLIST Statement Categories

Explanation

Control	Assignment	Conditional	I/O
ATTN	READ	DO	CLOSFILE
CONTROL	READDVAL	IF-THEN-ELSE	GETFILE
END-DATA	SET	SELECT	OPENFILE
DATA- PROMPT	LISTDSI		PUTFILE
ERROR, EXIT, GLOBAL, GOTO, NGLOBAL, PROC			
RETURN, SYSCALL, SYSREF, TERMIN, WRITE, WRITENR			

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 4


3.3: Syntax and Conventions
Standard Practice
Syntax and Conventions used while writing CLISTs are:
Capitalization:
CLIST statement names must be in upper case
If you use lowercase letters for CLIST statement names, the CLIST fails.
Capitalization of CLIST variable names, built-in function names, and TSO/E commands is optional
Formatting:
You can use blank lines for readability
Length:
Maximum length of CLIST statement is 32756
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING
Copyright © Capgemini 2015. All Rights Reserved 5

Standard Practices

- Comments:
  - Comments can be included on a line by itself
  - Comments can be included before, in the middle of, or after a CLIST statement or TSO/E command
  - For example 1:

```
/*comment*/
```

  - For example 2:

```
/*get return code*/  SET RC = &LASTCC
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 6

Standard Practices

- If you include a comment after a CLIST statement or TSO/E command, or on a line by itself, then the closing comment delimiter is not needed.
- For example:

```
alloc file(in) data(accounts.data) shr /* Input data set
```

- The first comment in CLIST should not contain the string REXX. Then CLIST treats it as REXX EXEC.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 7

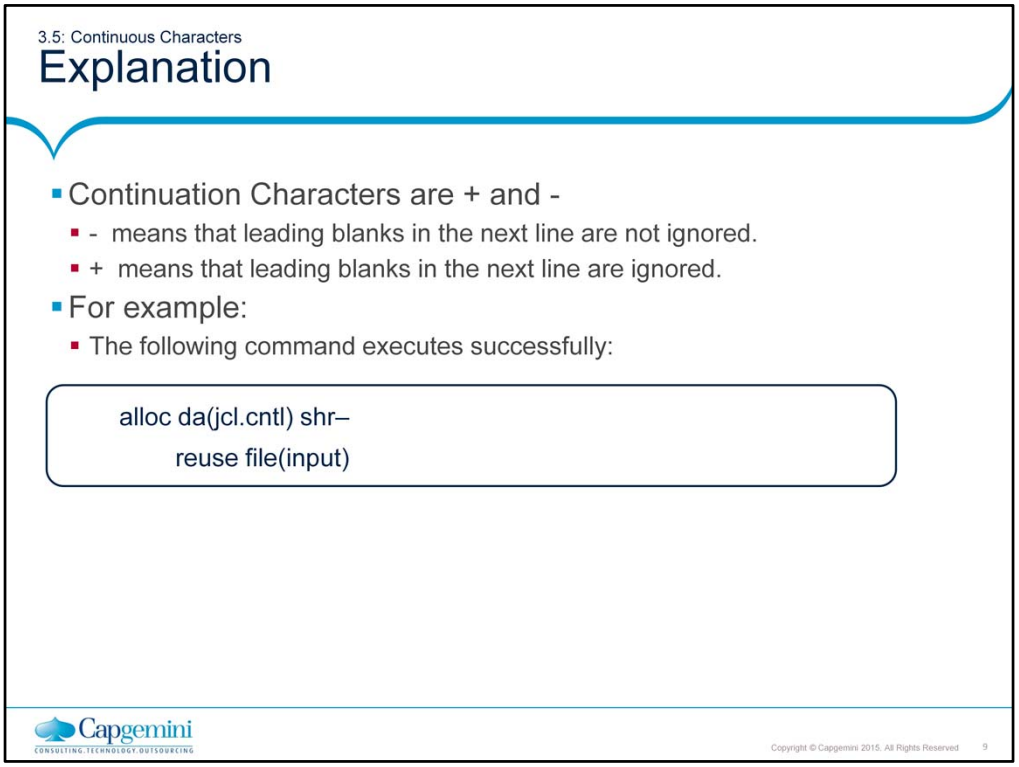
3.4: Delimiters

Explanation

- Operands are variables or data that provide information to be used in processing the statement
- Operands include one or more blanks between CLIST statement and its first operand.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

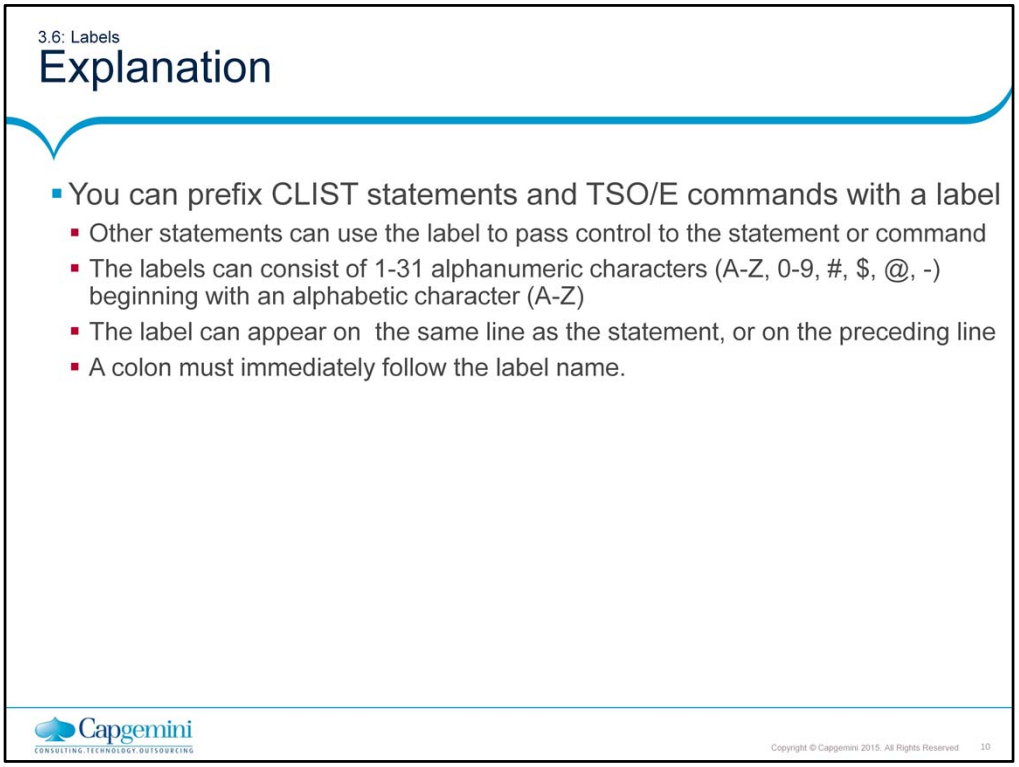
 8



Continuation Characters:

In case you substitute a plus sign for the hyphen in the example shown in the slide, the command fails. This is because, when the lines are joined logically, there is no blank between the end of the shr keyword and the beginning of the reuse keyword.

You will have to insert a blank before the plus sign for correct execution.



Labels:

For example:

label: IF A=…

Or

label: +

IF A=…

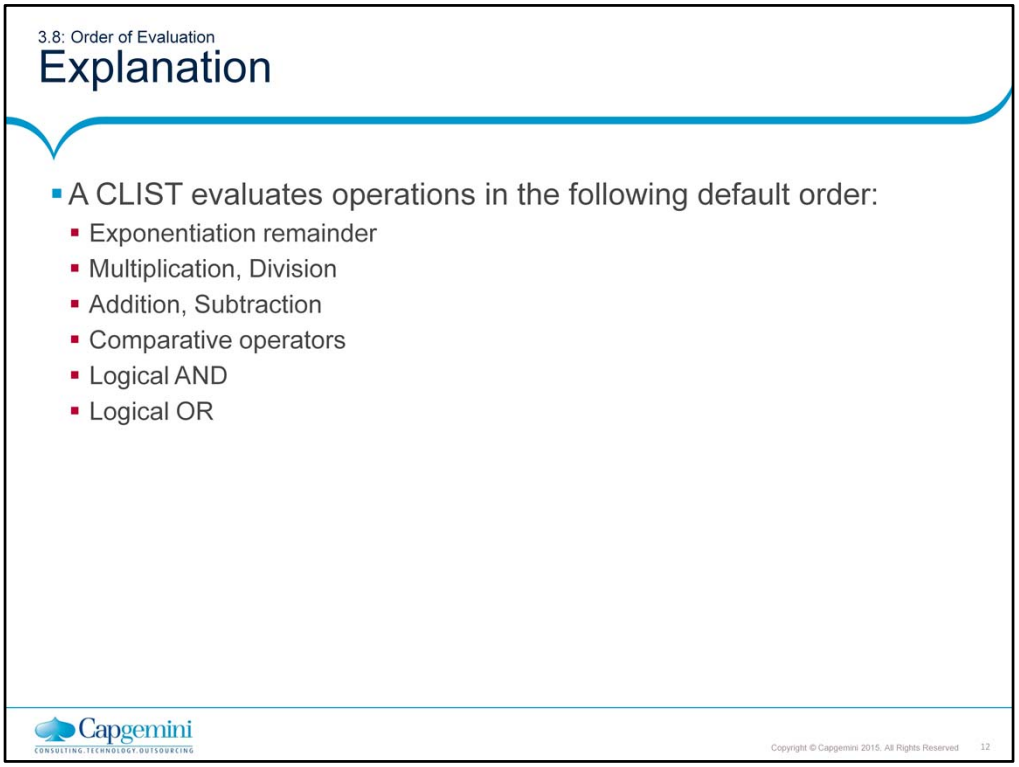
3.7: Operators

Categories

- Various categories of Operators are:
  - Arithmetic operators:
    - +, -, *, /, **, //, ()
  - Comparative operators:
    - EQ, NE, LT, GT, LE, GE, NG, NL
  - Logical operators:
    - AND, &&
    - OR, |

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 11



Order of Evaluation:

You can override the default order by placing parentheses around the operations you want to be executed first in the order.

For example: Without any parentheses, the following example performs multiplication, division, and then addition. The statement sets X to the value 24.

```
SET X = 4+5*8/2
```

By placing parentheses around 4+5, you indicate to the CLIST that it should perform addition first, and then proceed with the default order (multiplication, and then division). The following statement sets X to the value 36.

```
SET X = (4+5)*8/2
```

3.9: Workout with CLISTs

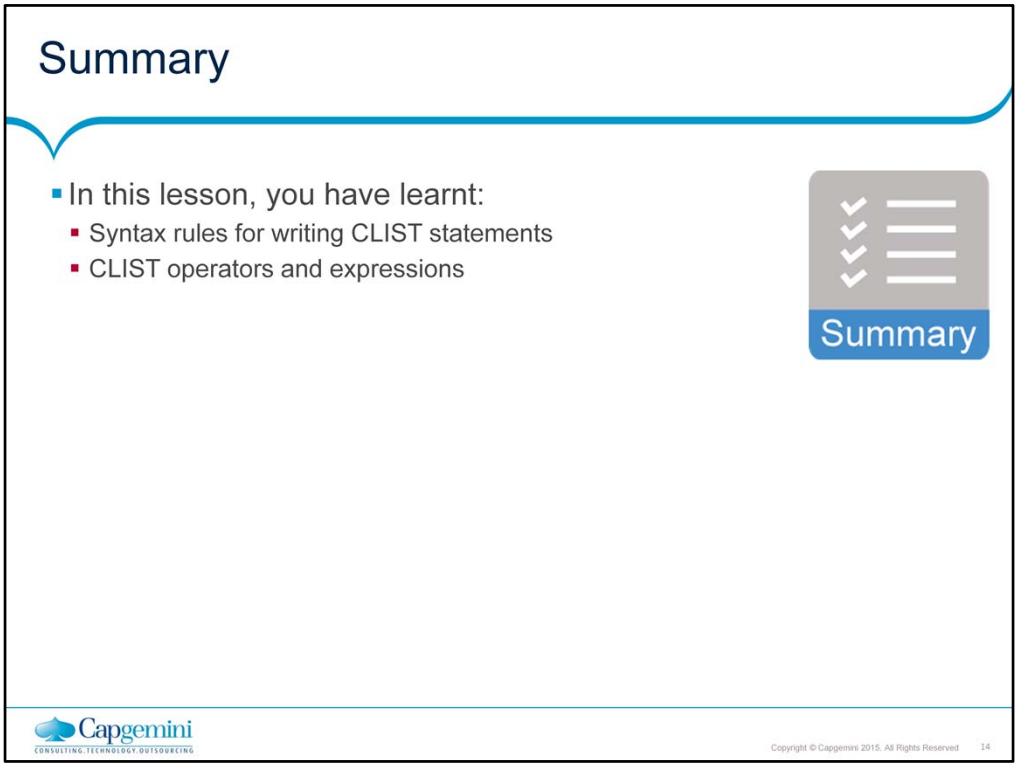
Lab

- Lab Session

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 13

Add the notes here.



Add the notes here.

Review – Questions

- Question 1: Which of the following is a valid continuation symbol?
  - Option 1: +
  - Option 2: *
  - Option 3: &
  - Option 4: #
- Question 2: All CLIST variable names must be capitalized.
  - True / False

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 15

Review – Questions

- Question 3: A ___ must immediately follow the label name.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 16

Introduction to CLIST and REXX

Lab Book

Document Revision History

Date	Revision No.	Author	Summary of Changes
17-Dec-2008	1.0	Sudhir Karhadkar	Original
11-Jun-2009	1.1	CLS Team	Review
Mar-2013	2	Rajita D	Revamped

Table of Contents

Getting Started

Overview

This lab book is a guided tour for learning REXX. It comprises solved examples and 'To Do' assignments. You can follow the steps provided in the solved examples and work out the 'To Do' assignments given.

Setup Checklist for REXX

Here is what is expected on your machine in order for the lab to work.

Minimum System Requirements

- Intel Pentium 90 or higher
- Microsoft Windows 95, 98, or NT 4.0, 2k, XP.
- Memory: 32MB of RAM (64MB or more recommended)
- Internet Explorer 6.0 or higher

Please ensure that the following is done:

- PASSPORT PC-TO-HOST (mainframe terminal simulator) is installed
- Connectivity to the Mainframe
- Availability of REXX on the mainframe

Instructions

- Create a directory by your emp-code in drive D. In this directory, create and edit your source REXX programs, then upload it to the mainframe.

Learning More (Bibliography)

- IBM REXX User's Guide
- IBM ISPF Edit and Edit Macros Manual
- IBM ISPF Services Guide
- IBM ISPF Dialog Developer's Guide and Reference

Lab 1. Coding and Running a REXX Exec

Goals	• Understand how to write simple REXX Exec. • Learn to run Exec – Explicitly and Implicitly • Passing data to an Exec
Time	60 minutes

1.1: Simple REXX Exec

First create a PDS *Loginid*.REXX.EXEC. Specifications: RECFM FB, LRECL 80, Block-size as 800.

The source REXX programs i.e. exec are stored as members in this PDS.

Solution:

Step 1: Write the following code as a member in the above PDS.

Loginid.REXX.EXEC(REXX1)

```
/* REXX */
SAY 'Hello World !'
EXIT
```

Example 1: Simple REXX code

1.2: How to run REXX Exec – Explicitly

Step 1: Go to the COMMAND (Option 6) of ISPF.

Step 2: Type (on Option 6 screen) -- EXEC 'dsrp050.rexx.exec(rexx1)' EXEC

OR

From the COMMAND line of any ISPF panel (say Edit Entry Panel), Type –
TSO EXEC rexx.exec(rexx1) EXEC

Output is:

```
Hello World !
```

Example 2: Output

1.3A: Displaying string literal

Step 1: Type the following code. Check how the strings are displayed when enclosed in single quotes, double quotes, and not enclosed in quotes.
Let the member name be dsrp050.rexx.exec(rexx1A)

```
/****REXX****/
SAY 'This is a REXX literal string.'
SAY "This is a REXX literal string."
SAY This is a REXX literal string.
EXIT
```

Example 3: REXX literal string

Step 2: Run the Exec explicitly as explained above.

Observe the output and note your findings.

1.3B: Displaying string literal with apostrophe

Step 1: Type the following code. Check how the strings with apostrophe are displayed – apostrophe or two single quotes are used.
Let the member name be dsrp050.rexx.exec(rexx1B)

```
/****REXX****/
SAY “This isn't a CLIST instruction.”
SAY 'This isn''t a CLIST instruction.'
EXIT
```

Example 4: Sample code

Step 2: Run the Exec explicitly.

Observe the output.

1.4A: Format: Continuing an instruction

Step 1: Type the following code. Check how the instruction is continued to the next line by using comma.
Let the member name be dsrp050.rexx.exec(rexx1C)

```
/****REXX****/
SAY 'This is an extended',
```



```
     'REXX literal string.'
EXIT
```

Example 5: Sample code

Step 2: Run the Exec explicitly.

Observe the output.

Output is:

```
This is an extended REXX literal string.
```

Example 6: Output

A space is added between “extended” and “REXX”.

1.4B: Format: Continuing an instruction

Step 1: Type the following code. Check how the instruction is continued on next line using comma.
Let the member name be dsrp050.rexx.exec(rexx1D)

```
/****REXX****/
SAY 'This is',
     'a string.'
SAY 'This is' 'a string.‘
EXIT
```

Example 7: Sample code

Step 2: Run the Exec explicitly.

Observe the output and note your findings.

1.4C: Format: Continuing an instruction

Step 1: Type the following code. Check how to continue a literal string without adding a space.
Let the member name be dsrp050.rexx.exec(rexx1E)

```
/****REXX****/
SAY 'This is a string that is bro'||,
      'ken in an awkward place.‘
EXIT
```

Example 8: Sample code

Step 2: Run the Exec explicitly.

Observe the output and verify it is as follows.

Output is:

```
This is a string that is broken in an awkward place.
```

Example 9: Output

1.4D: Try another program.

Step 1: Type the following code. Check how to continue a literal string without adding a space.
Let the member name be dsrp050.rexx.exec(rexx1F)

```
/****REXX****/
SAY 'This is' ||,
      'a string.'
SAY 'This is' || 'a string.'
EXIT
```

Example 10: Sample code

Step 2: Run the Exec explicitly.

Observe the output and verify if it is as follows.

Output is:

```
This isa string.
```

Example 11: Output

1.4E: Invoking TSO command

Step 1: Type the following code. REXX exec can invoke TSO command; it can accept data from user using PULL instruction. This program executes TIME command of TSO. Let the member name be dsrp050.rexx.exec(rexx1G).

```
/****REXX****/
/*  This Exec accepts time and uses TIME command  */
Game1:
SAY 'What time is it?'
PULL usertime            /* Put the user's responses into
                         a variable called usertime  */
IF usertime = '' '' THEN       /* User did not enter the time */
  SAY "O.K. Game's over."
ELSE
   DO
     SAY "The computer says:"
     /* TSO System */  TIME  /* command */
  END
EXIT
```

Example 12: TSO command

Step 2: Run the Exec explicitly.

Observe the output.

1.4F: (To do)

Write a REXX exec which accepts three numbers from the user. Display the sum of the numbers. The exec should prompt the user to enter the number. (To do)

1.4G: (To do)

Write a REXX exec which accepts the user name and displays welcome message for the user. (To do)

1.5A: How to run REXX Exec – Implicitly

The PDS containing REXX exec *Loginid*.REXX.EXEC needs to be allocated to ddname SYSEXEC.

Step 1: Create the member SETUP in *Loginid*.REXX.EXEC as follows.

Loginid.REXX.EXEC(SETUP)

```
"EXECUTIL SEARCHDD(yes)"
"ALLOC FILE(SYSEXEC) DATASET(REXX.EXEC) SHR REUSE"

IF RC = 0 THEN
  SAY 'ALLOCATION TO SYSEXEC COMPLETED.'
ELSE
  SAY 'ALLOCATION TO SYSEXEC FAILED.'
```

Example 13: Sample code

Step 2: Run this exec explicitly.

Now, the system exec library SYSEXEC will be searched when execs are implicitly invoked. This happens due to the statement EXECUTIL SEARCHDD(yes).

Step 3: Go to the COMMAND (Option 6) of ISPF.

Step 4: Type (on Option 6 screen) -- rexx1

OR

From the COMMAND line of any ISPF panel (say Edit Entry Panel), Type – **TSO** rexx1

To reduce the search time for an exec that is executed implicitly and to differentiate it from a TSO/E command, precede the member name with a %:

TSO %rexx1 or
%rexx1 (from Option 6)

1.5B: Run all previous examples implicitly. Note the difference.

1.6: Interpreting Error Messages

While running an exec that contains an error, an error message is displayed with corresponding line of exec

Step 1: Type the following code (which contains an error).

```
/* REXX */
SAY 'ENTER NAME'
PULL NM              /* GET THE NAME
SAY 'YOUR NAME: ' NM
```

Example 14: Sample code

Step 2: Execute the exec either explicitly or implicitly. Analyze the error message.

The code has an error. The exec runs until it detects the error, a missing */ at the end of the comment. The exec ends with an error message as given below.

```
ENTER NAME
3 +++ PULL NM              /* GET THE NAME
SAY 'YOUR NAME: ' NM
IRX0006I Error running REXX1E, line 3: Unmatched "/*" or quote
***
```

Example 15: Error message

The first line of error message begins with the line number of the statement where the error was detected, followed by three pluses (+++) and the contents of the statement. The second line of error message begins with the message number followed by a message containing the exec name, line where the error was found, and an explanation of the error.

1.7A: Passing data to an exec: PULL instruction

The exec displays the message to the user to enter the data. Then PULL instruction accepts the data from the user into the variables.
Please refer to the assignments 1.4E, 1.4F, 1.4G.

1.7B: Passing data to an exec: PULL instruction for multiple values

The PULL instruction can extract multiple values on a single statement.

Step 1: Type the following code.

```
/* REXX */
SAY 'ENTER TWO NUMBERS'
PULL N1 N2
SUM = N1 + N2
SAY 'THE SUM IS ' SUM'.'
```

Example 16: Sample code

Step 2: Run the exec either explicitly or implicitly.
Output is:

```
ENTER TWO NUMBERS
40 50
 THE SUM IS  90.
 ***
```

Example 17: Output

1.7C: Specifying input values (numbers) when invoking the exec

Step 1: Type the following code.

```
/* REXX */
ARG N1 N2
SUM = N1 + N2
SAY 'THE SUM IS ' SUM'.'
```

Example 18: Sample code

Step 2: Execute this exec.
Explicit: EXEC rexx.exec(rexx17C) '50 40' EXEC
Implicit: rexx17c 50 40

These values are called *arguments*.

Output is:

```
        THE SUM IS  90.
        ***
```

Example 19: Output

1.7D: Specifying input values (string) when invoking the exec

Step 1: Type the following code. It uses ARG instruction.

```
/*  REXX  */
ARG NM1 NM2
SAY 'WELCOME ' NM1 " " NM2
```

Example 20: Sample code

Step 2: Execute this exec.
Implicit: rexx17d Sachin Tendulkar

Output is:

```
WELCOME  SACHIN   TENDULKAR
         ***
```

Example 21: Output

Observe that the name is displayed in capitals, and not as entered by the user.

1.7E: Specifying input values (string) when invoking the exec

Step 1: Type the following code. It uses PARSE ARG instruction.

```
/*  REXX  */
PARSE ARG NM1 NM2
SAY 'WELCOME ' NM1 " " NM2
```

Example 22: Sample code

Step 2: Execute this exec.
Implicit: rexx17e Sachin Tendulkar

Output is:

```
WELCOME  Sachin   Tendulkar
         ***
```

Example 23: Output

Observe that the name is displayed as entered by the user.

Lab 2. Operators

Goals	• Learn how Arithmetic, Comparison and Logical operators work • Understand Operator priority
Time	45 minutes

2.1: Using operators

Step 1: Type the following code.

```
/****REXX ****/
PARSE ARG season snowing broken_leg
IF ((season = 'winter') | (snowing ='yes')) & (broken_leg ='no')  THEN
                    SAY 'Go skiing.'
ELSE
                    SAY 'Stay home.‘
EXIT
```

Example 24: Sample code

Step 2: Run the exec. Arguments passed are "spring yes no".
Run the exec. Arguments passed are "spring no no".
Run the exec with different combinations of arguments.

2.2: Using comparison and logical operators

Step 1: Type the following code**.**

```
/****REXX ****/

SAY (4>2) & (a=a)       /* true, so result is 1*/
SAY (2>4) & (a=a)       /* false, so result is 0*/
SAY (4>2) | (5=3)       /* at least one is true, so result is 1*/
SAY (2>4) | (5=3)       /* neither one is true, so result is 0*/
SAY (4>2) && (5=3)      /* only  one is true, so result is 1*/
SAY (2<4) && (5=5)      /* both are true, so result is 0*/
SAY \0                  /* opposite of zero, so result is 1*/
SAY \(4>2)              /* opposite of true, so result is 0*/ EXIT
```

Example 25: Sample code

Step 2: Run the exec and observe the output.

2.3: Using concatenation operators

Step 1: Type the following code.

```
/****REXX ****/
sport = 'base'
equipment = 'ball'
column = ‘     ’
cost = 5
SAY  sport||equipment  column  '$' cost
EXIT
```

Example 26: Sample code

Step 2: Run the exec and observe the output.

Lab 3. Control Flow Constructs

Goals	• Learn how to use IF..THEN..ELSE and nested IF .. statements • Learn how to code simple loops and conditional loops • Understand LEAVE instruction
Time	90 minutes

3.1: IF .. THEN .. ELSE

Step 1: Type the following code. Accept the variables “weather” and “tenniscourt” in the following exec as command line arguments.

```
        IF weather = ‘fine’ THEN
          DO
                SAY 'What a lovely day!'
                IF tenniscourt = ‘free’ THEN
                      SAY 'Shall we play tennis?'
                ELSE NOP
          END
        ELSE
          SAY 'Shall we take our raincoats?‘
```

Example 27: Sample code

Step 2: Run the exec with different values of arguments.

3.2: Loop using control variable

Step 1: Type the following code.

```
        /****REXX****/
        DO number = 1 TO 10  BY 2
                SAY 'Loop '  number
                SAY 'Hello ..!!'
        END
        SAY ‘Came out of loop when number reached ' number
EXIT
```

Example 28: Sample code

Step 2: Run the exec and observe the output.

3.3: Leave instruction

Step 1: Type the following code.

```
        /****REXX****/
        DO number = 1 TO 5
                SAY 'Loop '  number
                 IF number > 2 THEN
                    LEAVE
                 ELSE
                    SAY 'Hello ..!!'
        END
        SAY ‘Came out of loop when number reached ' number
EXIT
```

Example 29: Sample code

Step 2: Run the exec and observe the output.

3.4: Nested DO Loops

Step 1: Type the following code.

```
        /****REXX****/
        DO outer = 1 TO 2
    DO inner = 1 TO 2
                IF inner > 1 THEN
                   LEAVE inner
                ELSE
                   SAY 'HIP'
    END
    SAY 'HURRAH'
       END
EXIT
```

Example 30: Sample code

Step 2: Run the exec and observe the output.

3.5: (To do)

Write a loop as DO NUM = 10 To 8 …… END. Put a SAY instruction inside the loop. Observe what happens to the loop. Display the value of NUM after the loop. Note your conclusion.

3.6: (To do)

Accept a password from the user. If it equals to “PATNI123”, then display “Password is correct”, else display “Wrong password”. If user enters incorrect password three times, then display “Not allowed to continue”. Otherwise for correct password, display the number of attempt, as First/Second/Third, as the case may be.

3.7: (To do)

Write two nested loops. The outer loop should be DO outer = 1 To 3 ….. END. The inner loop should be DO inner = 1 TO 3 ….. END. For every iteration of the outer loop, only one iteration of the inner loop should be performed. Control variable value for inner loop has to be DO inner = 1 TO 3. Put appropriate SAY instructions in the loops.

3.8: (To do)

Accept “employee-name” and “salary” from the user. Compute the tax as follows.

If the salary is less than 50001, then tax is zero
If the salary is greater than 50000and less than 60001,
    then the tax is 10% of (salary – 50000)
If the salary is greater than 60000 and less than 100001,
    then the tax is 1000 + 20% of (salary – 60000)
If the salary is greater than 100000, then the tax is 9000 + 30% of (salary – 100000)

Form a string of name, salary, tax, and then display the string.

Repeat this till the user says no more employees.

Lab 4. Functions and Subroutines

Goals	• Learn how to use built-in and user defined functions • Learn how to code subroutines • Learn how to pass and receive data from/to functions, subroutines
Time	90 minutes

4.1: Coding of a function

Step 1: Type the following code. The function accepts three numbers and returns the maximum of it.

```
/****REXX****/
PARSE ARG number1, number2, number3
IF number1 > number2 THEN
   IF number1 > number3 THEN
      greatest = number1
   ELSE
      greatest = number3
ELSE
   IF number2 > number3 THEN
      greatest = number2
   ELSE
      greatest = number3
RETURN greatest
```

Example 31: Sample code

Step 2: Insert the lines in the beginning to invoke this function. Run it with different integer values.

4.2: Built-in functions

Step 1: Type the following code.

```
/****REXX****/
SAY 'ENTER DATA'
PARSE PULL VAR1
SAY DATATYPE(VAR1)
SAY LENGTH(VAR1)
SAY SUBSTR(VAR1,2,8)
```

Example 32: Sample code

Step 2: Run it with different input values.

4.3: (To do)

Use built-in functions of MAX, MIN, RANDOM, ABS, COMPARE, LENGTH, SUBSTR, DELSTR, USERID.

4.4: Passing data to a subroutine using variables

Step 1: Type the following code.

```
/****REXX****/
number1 = 5
number2 = 10
CALL sub1
SAY answer
EXIT

sub1:
answer = number1 + number2
RETURN
```

Example 33: Sample code

Step 2: Run the exec.

4.5: Passing data to a subroutine: Problems of sharing variables

Step 1: Type the following code.

```
/****REXX****/
number1 = 5
number2 = 10
DO i = 1 TO 5
   CALL sub2
   SAY answer
END
EXIT

sub2:
DO i = 1 TO 5
   answer = number1 + number2
   number1 = number2
   number2 = answer
END
RETURN
```

Example 34: Sample code

Step 2: Run the exec and analyze the output.

4.6: Passing data to a subroutine: PROCEDURE instruction

Step 1: Type the following code.

```
/****REXX****/
number1 = 10
CALL sub3
SAY number1 number2
EXIT


sub3: PROCEDURE
number1 = 7
number2 = 5
RETURN
```

Example 35: Sample code

Step 2: Run the exec and analyze the output. Please note your findings.

4.7: Passing data to a subroutine: PROCEDURE EXPOSE instruction

Step 1: Type the following code.

```
/****REXX****/
number1 = 10
CALL sub4
SAY number1 number2
EXIT

sub4: PROCEDURE EXPOSE number1
number1 = 7
number2 = 5
RETURN
```

Example 36: Sample code

Step 2: Run the exec and analyze the output. Please note your findings.

4.8: Passing data to a subroutine by using arguments

Step 1: Type the following code.

```
/****REXX****/
PARSE ARG long wide
CALL perimeter long, wide
SAY 'The perimeter is ' RESULT
EXIT

perimeter:
ARG length, width
perim = 2 * length + 2 * width
RETURN perim
```

Example 37: Sample code

Step 2: Run the exec accepting parameter values from command line.

4.9: Passing data to a function (To do)

Repeat assignments 4.4 thru 4.8 by using Functions
.

Lab 5. CLIST Assignments

1.1: Write a CLIST program to accept two numbers and display the sum of the two numbers.

1.2: Write a CLIST program to accept a number, check whether it is a prime number, and display appropriate message.

1.3: Write a CLIST program to accept a four digit year, display whether it is a leap year or not. Display appropriate messages for the given conditions.

a. Year accepted not a four digit numeric

b. Year is a leap year

c. Year is not a leap year

1.4: Write a CLIST program to accept AM or PM. If the user enters "AM" the message should be displayed as "Good Morning". If the user enters "PM" the message should be displayed as "Good Evening".

