REXX

Introduction To REXX Programming

RX101/E

MVS

I - 1

--PROPRIETARY AND CONFIDENTIAL INFORMATION--

THIS MATERIAL CONTAINS, AND IS PART OF A COMPUTER SOFTWARE PROGRAM WHICH IS, PROPRIETARY AND CONFIDENTIAL INFORMATION OWNED BY COMPUTER ASSOCIATES INTERNATIONAL, INC. THE PROGRAM, INCLUDING THIS MATERIAL, MAY NOT BE DUPLICATED, DISCLOSED OR REPRODUCED IN WHOLE OR IN PART FOR ANY PURPOSE WITHOUT THE EXPRESS WRITTEN AUTHORIZATION OF COMPUTER ASSOCIATES. ALL AUTHORIZED REPRODUCTIONS MUST BE MARKED WITH THIS LEGEND.

RESTRICTED RIGHTS LEGEND

Use, duplication, or disclosure by the United States Government ("the Government") is subject to restrictions as set forth in A) subparagraph (c)(2) of the Commercial Computer Software - Restricted Rights clause at FAR 52.227-19, and/or B) subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause of DFAR 252.227-7013. This software is distributed to the Government by:

Computer Associates International, Inc.
One Computer Associates Plaza
Islandia, NY 11788-7000

Unpublished copyrighted work - all rights reserved under the copyright laws of the United States.

This material may be reproduced by or for the United States Government pursuant to the copyright license under the clause at DFAR 252.227-7013 (OCTOBER 1988).

I - 2

Call Computer Associates technical services for any information not covered in this manual or the related publications. In North America, see your Computer Associates Product Support Directory for the appropriate telephone number to call for direct support, or you may call 1-800-645-3042 or 516-342-4683 and your call will be returned as soon as possible.

Outside North America, contact your local Computer Associates technical support center for assistance.

Introduction

- RX101
  - 3 days
- Audience
  - This course is for operations personnel or application programmers with minimal experience of REXX programming
- Prerequisites
  - TSO and ISPF experience
  - Delegates should have knowledge of another programming language
- Description
  - This course is designed to give an introduction to programming with REXX, regardless of operating environment. It focuses on the structure and format of the REXX language.

I - 3

Course Outline

- Preliminary concepts
- Basic Components
- Flow Control Instructions
- Looping Instructions
- Built-in functions
- PARSE instructions

I - 4

Course Objectives

Upon completion of this course you will be able to write REXX programs using:

- Proper syntax and demonstrating the fundamentals of good programming
- REXX functions as well as DO/WHILE loops and IF/THEN/ELSE constructs
- Built-in functions such as SUBSTR, DATATYPE and POS
- Basic user-written functions

I - 5

Are we on the right course ?

I - 6

Concepts

Section 1

1 - 1

Brief History

- REstructured eXtended eXecutor
- Standard SAA procedure Language
- Use driven development
- Designed to issue commands to the operating system

1 - 2

Why Program in REXX

- Easy to read and write, using meaningful English words
- Very little punctuation
- Free format - instructions are not column dependant and can span multiple line
- Variable declaration is not required
- Built-in Functions
- Trace capabilities
- Extensive word and character manipulation

1 - 3

Where to use REXX

- Use REXX to:
  - create easy and error free tasks
  - automate repetitive tasks
  - create programs that behave like system commands
  - tailor and enhance advanced software systems

1 - 4

Simple REXX program

```
/***************************** REXX ********************************/
/* Program                                                         */
/* sample1                                                         */
/* Description                                                     */
/* REXX program to display current time.                           */
/* Author       : Michaelangelo DeParma                            */
/* Date         : 1st January 2000                                 */
/*----------------------- Amendment History -----------------------*/
/***************************************************************************/
current_time = TIME("N")
SAY "The time is : "current_time
```

```
The time is : 03:47:07
***
```

1 - 5

REXX Instruction Formats

- Instructions are scanned left to right, top to bottom
- generally one line is an instruction
- indentation is only used for readability and does not affect program execution.

1 - 6

REXX Instruction Formats

- Multiple instructions are separated by semicolons when place on the same line.

```
SAY "Hello"; SAY "Good-bye"
```

- Use a comma on the end of a line to continue a statement.

```
job_cost = cpu_charge + oi_charge + shift_charge + tape_charge,
           + print_charge
```

1 - 7

SAY Instruction

- Output
- Format

```
SAY [expression]
```

- Examples

```
SAY error_message
SAY "Enter the error message :"
```

1 - 8

PARSE PULL Instruction

- Input
- Format

```
PARSE PULL [variable]
PULL [variable]
```

- Examples

```
PARSE PULL error_message
PULL answer
```

1 - 9

SAY and PULL Instructions

- Sample Program

```
/***************************** REXX ***********************************/
/* Program                                                            */
/* sample2                                                            */
/* Description                                                        */
/* REXX program to demonstrate SAY and PULL                           */
/* Author       : Michaelangelo DeParma                               */
/* Date         : 1st January 2000                                    */
/*---------------------- Amendment History -------------------------*/
/**********************************************************************/
SAY "Please enter the error code : "
PULL error_code
SAY "The error you are requesting information"
SAY "on is : "error_code" Y/N"
PULL answer
```

1 - 10

Work Section 1.1

- 1. Write a REXX program to display a list of three names.

```
Bob Flemming
Gary Stinker
John Thomas
***
```

1 - 11

Use your dataset for all the REXX programs.

`CLCS.IULCnn.REXX`

Replace nn with your user number.

E.g. for id IULC35

`CLCS.IULC35.REXX`

Work Section 1.2

- 2. Write a REXX program to collect and name from the screen and display a welcome to the screen.

```
 Please enter your name :
bob

 Hello                                                    BOB
 Welcome to the course.
 ***
```

1 - 12

Additional Program

- Write a REXX program to collect a forename and surname and display them both to the screen on one line.

```
 Please enter your forename :
bob

 Please enter your surname :
flemming

 Welcome to the course - BOB FLEMMING
 ***
```

1 - 13

Additional program.

Write a REXX program to collect a forename and surname and display them both to the screen on one line.

Additional Program

- Write a REXX program to collect a forename and surname on one line and display them both to the screen in reverse order..

```
  Please enter your Forename and Surname :
bob flemming

 Welcome to the course - FLEMMING BOB
 ***
```

1 - 14

Basic Components

Section 2

2 - 1

Clauses

- REXX Instructions
- Tokens
  - symbols
  - operators
  - literal
  - special characters
- Blanks

```
/***************************** REXX ********************************/
/* Program                                                         */
/* sample3                                                         */
/* Description                                                     */
/* REXX program to display examples.                               */
/* Author       : Michaelangelo DeParma                            */
/* Date         : 1st January 2000                                 */
/*----------------------- Amendment History -------------------------*/
/*************************************************************************/
SAY "This is some text"
x = x+w.i*(10**(w.i-1))
x = x + w.i * (10 ** (w.i - 1))
start = "01/01/2000"; end = "31/12/2002"
```

2 - 2

Sample Program

```
/***************************** REXX ***********************************/
/* Program                                                            */
/* sample4                                                            */
/* Description                                                        */
/* REXX program to calculate payroll                                  */
/* Author       : Michaelangelo DeParma                               */
/* Date         : 1st January 2000                                    */
/*---------------------- Amendment History -------------------------*/
/**********************************************************************/
PULL pay_month                                  /* get pay_month      */
pay_roll = 0                                    /* Initialise var     */
CALL find_month                                 /* subroutine         */
SAY "Total for "pay_month" was : "pay_roll      /* Display result     */
EXIT                                            /* Leave program      */

find_month:
NOP
RETURN
```

2 - 3

Types of statements

- Null
  - blank or comments
- Label
  - label_name:
- Assignments
  - literal, numbers, symbols, operators, variables
- Keyword instructions
- Commands

2 - 4

Comments

- Begin with /* and end with */
- Can contain any sequence of characters on one or more lines
- Most REXX programs begin with a comment with the word REXX contained within the comments.
- Example

```
/***************************** REXX ********************************/
/* Program                                                         */
/* sample1                                                         */
/* Description                                                     */
/* REXX program to display current time.                           */
/* Author       : Michaelangelo DeParma                            */
/* Date         : 1st January 2000                                 */
/*----------------------- Amendment History -------------------------*/
/***********************************************************************/
current_time = TIME("N")                    /* get time            */
SAY "The time is : "current_time            /* display result      */
```

2-5

Literal Strings

- A series of characters enclosed in single or double quotes
- A literal string can contain ANY characters
- Maximum size may vary
- Continued by placing a comma after the last character on the line continuing in column 1 on next line

2 - 6

Literal Strings

- If a literal contains a ' use a "
- If a literal contains a " use a '
- Strings starting with quotes must also end with quotes
- Any string with no characters, "", is called a null string and has a length of 0

2 - 7

Literal String examples

```
SAY "12345"
SAY 'abc'
SAY "ABC"
SAY "Please enter a number"
SAY "Your name is 'Bob Flemming'"
```

```
12345
abc
ABC
Please enter a number
Your name is 'Bob Flemming'
***
```

2 - 8

Hexadecimal Strings

- Contain the characters 0-9, a-f, A-F
- Delimited by quotes
- Followed by an X or x

```
num_1 = 'ABCD'X
num_2 = "1d ec f8"X
num_3 = '123 45'x
```

2 - 9

Numbers

- Character strings that consist of one or more decimal digits
- Optionally prefixed by a plus or minus
- Can include decimal point
- can be exponential notation

```
num_1 = 29
num_2 = -94.3
num_3 = 53.312E23
num_4 = +0.5
num_5 = 0.73E-7
```

2 - 10

Test Exercise 21

- Write a REXX program using the following code. Test and identify the type of statements. Which statement is :
  - Comment
  - Literal
  - Hexadecimal
  - number
  - error

```
/* REXX EXEC */
a = "REXX EXEC"
c = 94 4F
d = '94 4F'
e = 32
f = '32'
g = "32"x
b = "REXX EXEC"x
REXX EXEC
```

2 - 11

Symbols

- A group of any characters selected from the English alphabetic and numeric characters (A-Z, a-z, 0-9) and/or from the characters .!? And _
- Symbols are used to name variables, functions, instructions, labels etc.

```
a = ALLEN
b = this is a symbol!
c = hello?
test_routine:
```

2 - 12

Simple Variables

- A variable is a name the represents a value.
- First character cannot be a number or a period
- Variable names must be less than 250 characters

```
salary = pay + benefit + bonus
job_class = "A"
msg_class = "X"
operator = current_operator_for_this_task
```

2 - 13

Simple Variables

- Variable names can only consist of the following :
  - A-Z, 0-9, a-z, @#$?!._
- Variable names are converted to upper case for comparison
- Variables are initialised to their own name in upper case.

```
SAY pay_roll
```

```
PAY_ROLL
***
```

2 - 14

Test Exercise 22

- Write a REXX program to identify which are valid statements.
  - Try and fix where possible.

```
Answer = "yes"
The-Answer = "no"
Msg.Text = "IST510I"
1ST_Class = "never"
the_worlds_largest_variable = "test"
REPLLY? = "Z NET,QUICK"
SAY reply
```

2 - 15

Compound Variables

- A technique for grouping variables
- Start with a stem which is a valid variable name followed by a period (.)
- Characters after the first period are called tail

```
CPUTIME.ASID
Stem    Tail

new_rate.shift.type.weekly
Stem     Tail  Tail Tail
```

2 - 16

Compound Variables

- Tails are treated as a series of simple variable names separated by periods
- At execution time, each variable is replaced by its current value to generate the derived name.

```
/****************************** REXX ***********************************/
/* Program                                                             */
/* test                                                                */
/* Description                                                         */
/* REXX program to test statements                                     */
/* Author       : Michaelangelo DeParma                                */
/* Date         : 1st January 2000                                     */
/*---------------------- Amendment History -------------------------*/
/*********************************************************************/
tgif = 3
day.1 = "Mon"
day.2 = "Tue"
day.3 = "Wed"
SAY day.tgif
```

2 - 17

Assigning Values to the Stem

- If a value is assigned to a stem, ALL POSSIBLE compound variables that begin with that stem also are assigned to this value.

```
name. = "Nobody"
a = 2
b = 3
name.a = "Jed"
name.b = "Jim"
SAY name.1 name.2 name.3 a b
```

```
Nobody Jed Jim 2 3
***
```

2 - 18

Arithmetic Operators

- Prefix operators
  - +, -
- Power Operators
  - **
- Multiplication and division
  - *, /, %, //
- Addition and subtraction
  - +, -
- Grouping
  - ()

2 - 19

1) Expressions in parenthesis are evaluated first

2) prefix operators - and + if present for numbers in expression

3) exponentiation **

4) Multiplication and division in this order *,/,%,//

5) Addition and Subtraction + and -

Test exercise 23

- What are the answers to the following:
  - 2+3*4
  - (2+3)*4
  - 9**2/10
  - 23%5//2**3
- Now check how REXX calculates these numbers.

2 - 20

Concatenation Operators

- Concatenation
  - blank
  - ||
  - abutted

```
a = "one"; b = "two"
total = "75"
SAY a b
SAY ab
SAY a||b
SAY total "%"
SAY total"%"
```

```
one two
AB
onetwo
75 %
75%
***
```

2 - 21

Test exercise 24

- Test the following :

```
SAY "REXX"          ||          "Course"
SAY "REXX"||"Course"
variable = "who knows"
SAY variable"the answer"
SAY "First"           "Second"
SAY variable||""||answer
```

2 - 22

Keyword Instructions

- One or more clauses, first of which starts with a keyword that identifies the instruction
- Keyword instructions control
  - external and internal interfaces
  - affect the flow of control
- Some keywords can included nested instructions
- Keywords are not case dependant.

2 - 23

Keyword Instructions

```
IF title = "Yes" THEN DO
  SAY "Hello"
END
DO WHILE title = "Yes"
   SAY "Hi"
   LEAVE
END
```

2 - 24

Work Section 2.1

- Write a REXX program to manipulate information entered from the terminal.
- Prompt the user to enter their first name. Then prompt the user to enter their last name.
- After they do this display the first and last names in the following formats on the terminal

```
first_name last_name
last_name, first_name
last_namefirst_name
first_name_lastname
```

2 - 25

What is the result from this?

Use your dataset for all the REXX programs.

CLCS.IULCnn.REX

Work Section 2.2

- Write a REXX program to prompt a user to enter their name.
- Then prompt for their gross salary.
- Then display their name and the tax they have paid at 30%

```
 Please enter your first name.
Mike
 Please enter your salary.
20000
 Your tax paid is :6000.0
 ***
```

2 - 26

What is the format of the result ?

The PARSE Instruction

Section 3

3 - 1

PARSE Instruction

- The PARSE Instruction is used to assign data from various sources to one or more variables

```
PARSE {UPPER} { ARG       }                  {template}
              { EXTERNAL  }
              { NUMERIC   }
              { PULL      }
              { SOURCE    }
              { VALUE {expression} WITH }
              { VAR name  }
              { VERSION   }
```

3 - 2

The PARSE instruction tells REXX how to assign data to one or more variables. The data to assign can be from the terminal, the data stack, or from arguments passed to a subroutine or function. The way in which REXX assigns data to a variable is governed by what is known as a 'parsing template', discussed below.

'template' is made up of alternating optional "patterns" and variable names. "patterns" are of two types: those that cause parsing to search for a matching string (variable patterns and literal patterns) and numeric patterns that supply a string position number in the data from which parsing is to extract data. Any number of "patterns" and

variables can be intermixed.

Operands

- PARSE UPPER
  - tells REXX to translate the data to be parsed to uppercase before parsing is done. Without the UPPER option, no uppercase translation is done before or after parsing.
- PARSE ARG
  - The arguments passed to the subroutine or function in which the PARSE statement is executed are parsed. This is equivalent to the operation of the REXX ARG function.
- PARSE EXTERNAL
  - REXX obtains the string to be parsed from the TSO stack, which usually gets it from the TSO terminal. PARSE PULL has the same affect as this PARSE form and is used more often.

3 - 3

Operands

- PARSE NUMERIC
  - returns the current settings for the NUMERIC options DIGITS, FUZZ, and FORM, in that order. The statement
- PARSE PULL
  - This form of the PARSE instruction makes REXX get the next string from the REXX data stack. If the stack is empty, REXX will get the string from TSO terminal.
- PARSE VALUE
  - this PARSE form parses a string under the control of the parsing template, described previously.

3 - 4

ARG

- Syntax
  - PARSE [UPPER] ARG [template]
- Parses the arguments passed to the program according to the template, optionally first translating it to uppercase
- Shortened to - ARG

```
ARG test_word
SAY "You have passed the program : "test_word
```

3 - 5

VAR

- Syntax
  - PARSE VAR name [template]

```
ARG test_words
PARSE VAR test_words first_word second_word left_overs
SAY "You have passed the program : "first_word
SAY "You have passed the program : "second_word
SAY "You have passed the program : "left_overs
```

```
You have passed the program : THIS
You have passed the program : IS
You have passed the program : A TEST OF WORDS
***
```

3 - 6

PULL

- Syntax
  - PARSE PULL [template]

```
SAY "Please enter your first name."
PARSE PULL first_name
```

3 - 7

EXTERNAL

- Syntax
  - PARSE EXTERNAL [template]

```
SAY "Please enter your first name."
PARSE EXTERNAL first_name
```

3 - 8

VALUE WITH

- Syntax
  - PARSE VALUE [expression] WITH [template]
- Use VALUE WITH to break apart long strings or expressions that need to be evaluated first

```
new_date = "12/11/2000"
PARSE VALUE new_date WITH mm "/" dd "/" yyyy
SAY mm
SAY dd
SAY yyyy
```

3 - 9

Parsing Templates

- Simplest form consists of a list of variable names
- String being parsed is split into words
- each word is assigned to a variable from left to right
- The last variable is assigned whatever is left.

```
name = "Mike Fred Bob Jones"
PARSE VALUE name WITH first_name left_overs
SAY first_name
SAY left_overs
```

3 - 10

Parsing Templates

- If there are fewer words than variables, any remaining variables are assigned the null string.
- Leading blanks are removed from each word.

```
name = "Mike"
PARSE VALUE name WITH first_name left_overs
SAY first_name
SAY left_overs
```

3 - 11

Parsing Templates

- Example - What happens here ?

```
name = "Mike Fred Jones"
PARSE VAR name new_name name
SAY name
SAY new_name
```

3 - 12

Literal Patterns

- Example

```
names = "Mike,Fred,Joe"
PARSE VAR names one "," two "," three
SAY one
SAY two
SAY three
```

```
Mike
Fred
Joe
***
```

3 - 13

Using Placeholders

- Example

```
names = "Mike,Fred,Joe"
PARSE VAR names . "," . "," last_name
SAY last_name
```

```
Joe
***
```

3 - 14

Positional Patterns

- Example

```
names = "Some text, to be, split up!"
PARSE VAR names one 10 two 20 three
SAY one
SAY two
SAY three
```

```
Some text
, to be, s
plit up!
***
```

3 - 15

Relative Positional Patterns

- Example

```
names = "0987654321"
PARSE VAR names 3 one +2 two +4 three 2 four
SAY one
SAY two
SAY three
SAY four
```

```
87
6543
21
987654321
***
```

3 - 16

Relative Positional Patterns

- The template
  - ',' -1 test_char +1
- Will
  - find the first comma in the input
  - backup one position
  - Assign one character to test_char after the comma
  - move along one position.

3 - 17

Relative Positional Patterns

- Simple using the relative column numbers relative to a literal.

```
names = "This is a list"
PARSE VAR names 1 one +1 two +1 three +1 four the_rest
SAY one
SAY two
SAY three
SAY four
SAY the_rest
```

```
T
h
i
s
is a list
***
```

3 - 18

Keyword Arguments

- Example

```
ex 'IULC00.IULC.REXX(test)' 'FILE(iulc.test), VOL(i1)'


iulc.test
i1
***
```

3 - 19

```
ARG data_list
PARSE VAR data_list "FILE(" data_set "), " "VOL(" volume ")"
SAY data_set
SAY volume

or

ARG 1 "FILE(" data_set ")" 1 "VOLSER(" volume")"
SAY data_set
SAY volume
```

Variable Patterns

- Specify a pattern by using the value of a variable instead of a fixed string number
- Place the name of the variable to be used as the pattern in parentheses
- If a +, - or = sign precedes the parentheses, the value of the variable is then used as though it were a relative column number'

```
name = "Mike"
PARSE VALUE name WITH first_name left_overs
SAY first_name
SAY left_overs
```

3 - 20

Variable Patterns example

```
data = "L/look for /1 10"
PARSE VAR data verb 2 delim +1 string (delim) rest
SAY verb
SAY delim
SAY string
SAY rest
```

```
L
/
look for
1 10
***
```

3 - 21

Work Section 3.1

- Write a REXX program to accept a name from the execution line.
- Say hello to the name.

```
ex 'clcs.iulc00.rexx(rx10131)' 'bob'



Hello bob
***
```

3 - 22

Work Section 3.2

- Write a REXX program to accept a 3 level qualified dataset from the execution line.
- Then display each section to the screen

```
ex 'CLCS.IULC00.REXX(rx10132)' 'iulc00.iulc.rexx'



Project : iulc00
Group   : iulc
Type    : rexx
***
```

3 - 23

Additional Program

If you had proceeding spaces in work section 1.2 then re-write using PARSE to remove the spaces.

Additional Program

- If you had proceeding spaces in work section 1.2 then re-write using PARSE to remove the spaces.

3 - 24

Additional Program

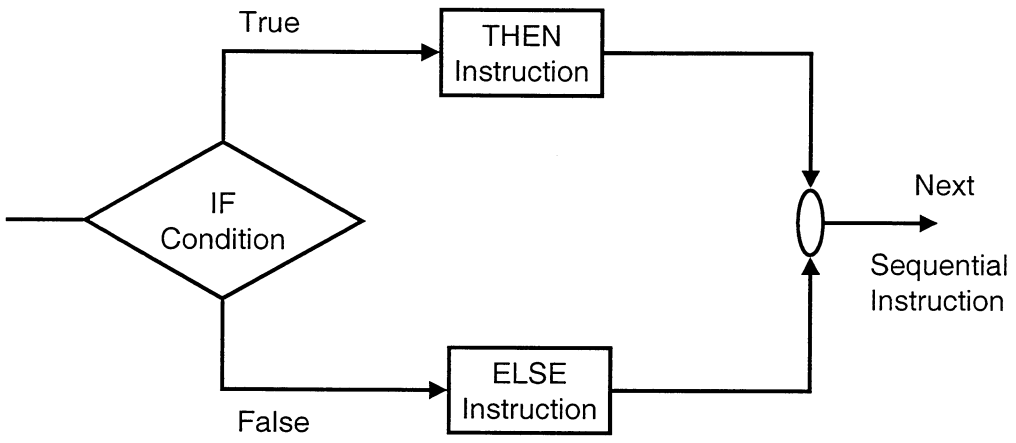
If you had proceeding spaces in work section 1.2 then re-write using PARSE to remove the spaces.

Flow Control Instructions

Section 4

4 - 1

IF/THEN/ELSE Statement



4 - 2

IF/THEN/ELSE Statement

- Format
  - IF expression THEN instruction
    - [ELSE instruction]
- Expression must evaluate to 1 or 0 (true or false)
- Every IF must have THEN and and instruction indicating what to do if the expression is true
- The ELSE clause indicates what to do if the expression is false, optional.

4 - 3

IF/THEN/ELSE Statement

- Example

```
job_name = "PAYROLL"
system = "UP"
IF job_name = "PAYROLL" THEN
   SAY "Load payroll cheques"
IF system = "UP" THEN
   SAY "System is up"
ELSE
   SAY "System is down"
```

4-4

Test Exercise 41

- Write a REXX program to test the statements below.
- Correct the code where necessary

```
test_value = 1
IF test_value = 1 SAY "Yes"
IF test_value = 1
   THEN SAY "Yes"
IF test_value = 1 THEN SAY "Yes" ELSE SAY "No"
IF test_value = 1 ELSE SAY "No"
IF test_value = 1
   THEN
      SAY "Yes"
   ELSE
      SAY "No"
```

4 - 5

Nested IF statements

- IF statements also may be nested within IF statements

```
PARSE ARG age
IF age < 65 THEN
   IF age > 21 THEN
      SAY "Over 21 and under 65"
   ELSE
      IF age >= 16 THEN
         SAY "Between 16 and 21"
      ELSE
         SAY "Under 16"
ELSE
   SAY "65 or over"
```

4 - 6

Comparative Operators

- Compare two terms and return 1 if the result is true and 0 if then result is false

- Normal comparison
  - = equal
  - \= not equal (can also use not sign, X'5F')
  - > greater than
  - < less than
  - >< greater than or less than (same as not equal)
  - >= greater than or equal to
  - <= less than or equal to
  - \< not less than
  - \> no greater than

4 - 7

== strictly equal

\== not strictly equal (can also use not sign, X'5F')

Also in 3270 :

¬= not equal

Comparative Operators

- When REXX compares two non-numeric values, it ignores leading and trailing spaces
  - " REXX " = "REXX"
    - Would evaluate as true
- When REXX compares two numeric values it ignores leading and trailing zeros.
  - 00000000012 = 12
  - 12 = 12.000
    - Would evaluate to true

4 - 8

Comparative Operators

- Strictly means that the two values must match each other.
  - 00000000000012 == 12
    - Would be false
  - " REXX " == "REXX"
    - Would evaluate as false

4 - 9

Logical Operators

- Logical operators combine two comparisons return 0 or 1.
- Types of logical operators.

&	AND
¦	OR
&&	EXCLUSIVE OR
\	NOT

4 - 10

& AND - returns a 1 (true) if both comparisons are true, and a 0 (false) otherwise - performs a logical AND operation

| OR - returns a 1 (true) if at least one comparison of several is true, and a 0 (false) otherwise - performs a logical or operation

&& EXCLUSIVE OR - returns a 1 (true) if ONLY one of a group of comparisons is true, and a 0 (false) otherwise - performs a logical exclusive OR function

\ NOT - returns the reverse logical value for an expression returns false if expression resolves to true, and true if the expression resolves to false

Multiple Logical Operators

- When multiple logical operators are used, &s are evaluated before |s.

```
test_value = 1
old_value = 2000
new_value = 3
IF test_value = 1 & (old_value = 2 | new_value = 3) THEN
   SAY "All ok"
```

4 - 11

THEN DO and ELSE DO

- This groups several statements together so that REXX will treat them as one instruction
- Often you need to execute more than one instruction in a THEN or ELSE clause

```
system_state = "UP"
IF system_state = "UP" THEN DO
   SAY "The system should be down"
   system_state = "DOWN"
END
```

4 - 12

SELECT statement

- Format
  - SELECT
    - WHEN
    - OTHERWISE
  - END

```
system_state = "UP"
SELECT
   WHEN system_state = "UP" THEN
      system_state = "DOWN"
   WHEN system_state = "DOWN" THEN
      system_state = "UP"
   WHEN system_state = "FAIL" THEN
      system_state = "DOWN"
   WHEN system_state = "WARNING" THEN
      system_state = "ERROR"
   OTHERWISE SAY "System state invalid"
END
```

4 - 13

Unlike IF with ELSE, the SELECT statement requires the OTHERWISE for all false conditions.

NOP

- Dummy instruction that has no effect
- Often used with and IF and SELECT

```
system_state = "UP"
SELECT
   WHEN system_state = "UP" THEN
      system_state = "DOWN"
   WHEN system_state = "DOWN" THEN
      system_state = "UP"
   WHEN system_state = "FAIL" THEN
      system_state = "DOWN"
   WHEN system_state = "WARNING" THEN
      system_state = "ERROR"
   OTHERWISE NOP
END
```

4 - 14

Work Section 4.1

- Re-Write the "AGE" nested IF as four separate IF statements in order to perform the same function, in a REXX program. Assign the value AGE as an argument.

```
ex 'clcs.iulc00.rexx(rx10141)' '65'

65 or over
***
```

AGE =	Result
10	
21	
65	
70	
Your age	

4 - 15

```
IF age < 65 THEN
   IF age > 21 THEN
      SAY "Over 21 and under 65"
   ELSE
      IF age >= 16 THEN
         SAY "Between 16 and 21"
      ELSE
          SAY "Under 16"
ELSE
   SAY "65 or over"
```

Enter the results in the table above.

Hint do not use any else statements.

Work Section 4.2

- Re-Write Work section 4.1 "AGE" Using the select Statement

```
ex 'clcs.iulc00.rexx(rx10131)' '65'

65 or over
***
```

AGE =	Result
10	
21	
65	
70	
Your age	

4 - 16

```
IF age < 65 THEN
   IF age > 21 THEN
      SAY "Over 21 and under 65"
   ELSE
      IF age >= 16 THEN
         SAY "Between 16 and 21"
      ELSE
          SAY "Under 16"
ELSE
   SAY "65 or over"
```

Enter the results in the table above.

Additional program

- Write a REXX program to display the tax paid for each of the codes below given entered at the screen:

Tax Code as a Percent	Result
10	
20	
50	
70	
80	

```
 Please enter your salary.
12000
 Please enter your TAX band.
70
 Your tax for : 12000 : is : 8400.0
 ***
```

4 - 17

Section end

4 - 18

Debugging REXX Programs

Section 5

5 - 1

Debugging

- TRACE Instruction
  - Syntax errors
  - logic problems
  - often used when creating programs
- Error Trapping
  - abnormal situations
  - often used in production
- Dependent REXX platform
  - examples using TSO/E REXX

5 - 2

Interactive Tracing

- Before execution of a REXX program
  - EXECUTIL TS
- During execution of a REXX program
  - PA1 or ATTN
  - HI
- Commands in the REXX program
  - TRACE [?|!] option
  - EXECUTIL TS|TE|HT|RT|HI

5 - 3

TRACE Instruction

- FORMAT:
  - TRACE [trace setting]
- trace setting :
  - A
    - (All) - all REXX clauses are traced before they execute - this single option enables all of those below Call host commands are traced before execution.
  - C
    - (Commands) - all TSO/E commands are traced before they are executed and any non-zero return code returned by a TSO/E command is also traced; as an example, the 'TRACE !C' instruction causes TSO/E commands to be traced but not executed.
  - E
    - (Error) - any TSO/E command returning a non-zero return code is traced after it executes
  - I
    - (Intermediates) - all clauses are traced before they execute, and the 'intermediate' (working) results of evaluations of expressions and any substituted names are traced too

5 - 4

TRACE Instruction

- L
  - (Labels) - labels on instructions that execute are traced
- N
  - (Normal) - a TSO/E command that returns a negative return code is traced after it executes - this trace option is the default if no other is specified
- O
  - (Off) - turns off ALL tracing
- R
  - (Results) - all REXX clauses are traced before they execute, and the final results of evaluating each expression are also traced. The values that are assigned via a PULL, ARG, and PARSE instructions are traced.
- S
  - (Scan) - all clauses in the exec are traced, but not executed. Checking for missing END statements is carried out. 'TRACE S' only works if the 'TRACE S' clause is not imbedded in any other instruction, such as an INTERPRET or an internal routine.

5 - 5

TRACE Example - A

```
TRACE A
"test"
old_name = "Fred"
SAY "Please enter name : "
PARSE PULL tst_name
PARSE VAR tst_name new_name .
IF new_name = old_name THEN DO
   SAY "Hello Fred"
END
ELSE DO
   NOP
END
```

5 - 6

TRACE Example - A

```
 12 *-* "test"
       >>>   "test"
 MISSING DSNAME
     13 *-* old_name = "Fred"
     14 *-* SAY "Please enter name : "
 Please enter name :
     15 *-* PARSE PULL tst_name
Fred
     16 *-* PARSE VAR tst_name new_name .
     17 *-* IF new_name = old_name
        *-*  THEN
        *-*  DO
     18 *-*   SAY "Hello Fred"
 Hello Fred
     19 *-*  END
 ***
```

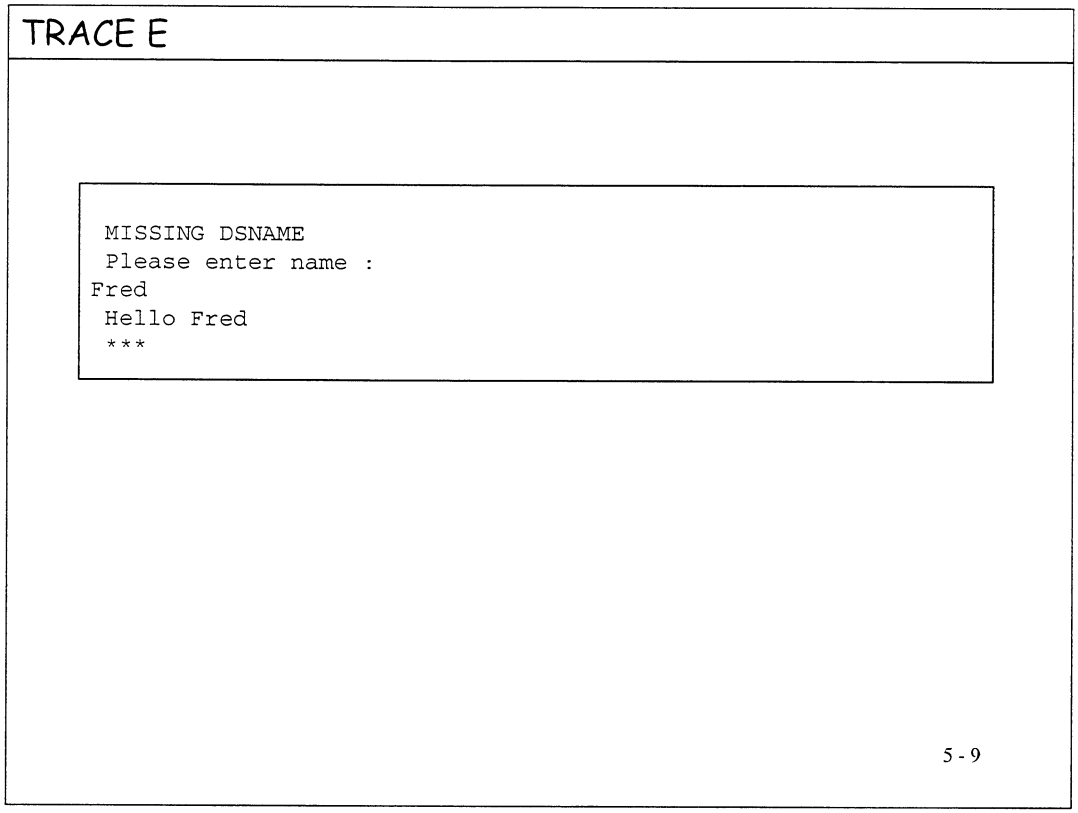
```
12 *-* : Source Code
>>>    : Result of statement
MISSING DSNAME : TSO Command
```

5-7

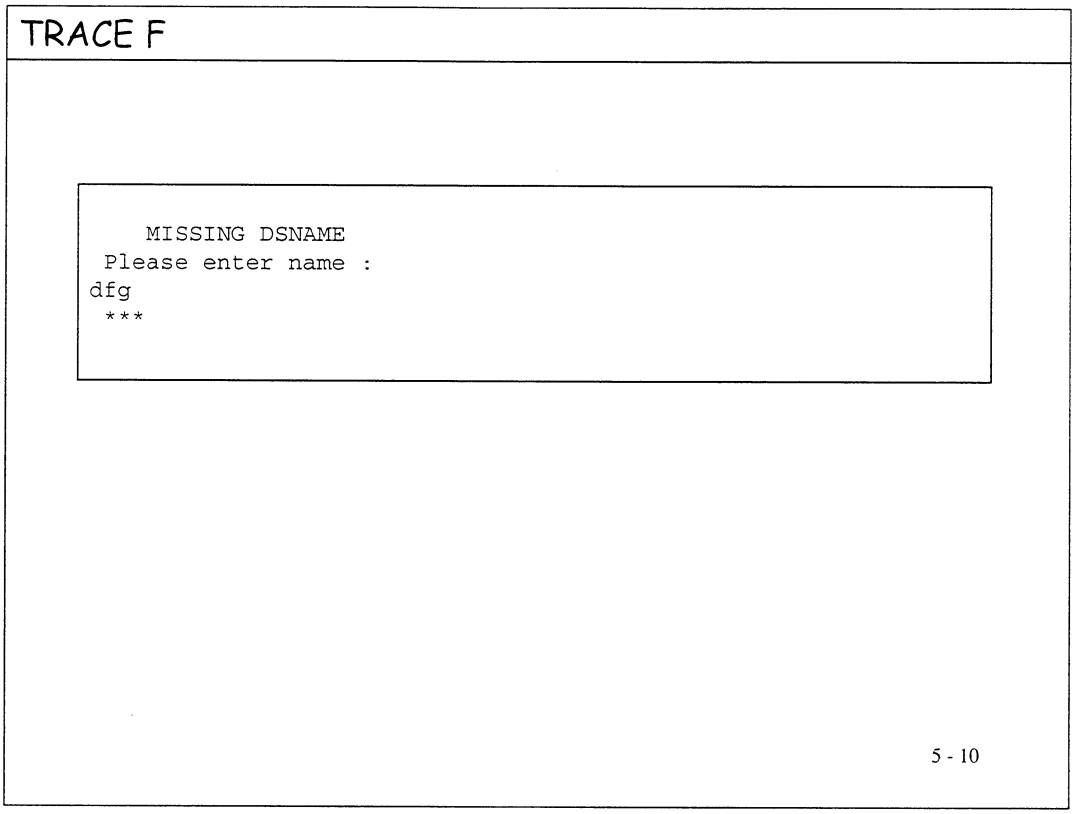
TRACE C

```
   12 *-* "test"
        >>>   "test"
 MISSING DSNAME
 Please enter name :
Fred
 Hello Fred
 ***
```

5 - 8



Traces error after execution.



F - Failure (Same as Normal)

TRACE I

```
 12 *-* "test"
        >L>   "test"
 MISSING DSNAME
     13 *-* old_name = "Fred"
        >L>   "Fred"
     14 *-* SAY "Please enter name : "
        >L>   "Please enter name : "
 Please enter name :
     15 *-* PARSE PULL tst_name
bob
        >>>   "
bob"
     16 *-* PARSE VAR tst_name new_name .
        >>>   "bob"
        >.>   ""
     17 *-* IF new_name = old_name
        >V>   "bob"
        >V>   "Fred"
        >O>   "0"
     20 *-* ELSE
        *-*  DO
     21 *-*   NOP
     22 *-*  END
 ***
```

TRACE I

```
>L>    : Literal Sting

>.>    : this line shows the value assigned to a placeholder in
         a parse template during parsing.

>V>    : this line shows the contents of a variable

>O>    : this line shows the result of an operation on two
         terms
```

5 - 12

TRACE L

```
    MISSING DSNAME
 Please enter name :
Fred
 Hello Fred
 ***
```

5 - 13

TRACE N

```
   MISSING DSNAME
 Please enter name :
Fred
 Hello Fred
 ***
```

5 - 14

Normal - Default (Same as Failure)

TRACE O

```
    MISSING DSNAME
 Please enter name :
bob
 ***
```

5 - 15

TRACE R

```
    12 *-* "test"
       >>>   "test"
 MISSING DSNAME
     13 *-* old_name = "Fred"
       >>>   "Fred"
     14 *-* SAY "Please enter name : "
       >>>   "Please enter name : "
 Please enter name :
     15 *-* PARSE PULL tst_name
bob
       >>>   "
bob"
     16 *-* PARSE VAR tst_name new_name .
       >>>   "bob"
       >.>   ""
     17 *-* IF new_name = old_name
       >>>   "0"
     20 *-* ELSE
       *-*  DO
     21 *-*   NOP
     22 *-*  END
 ***
```

5 - 16

TRACE S

```
    11 *-* TRACE S
    12 *-* "test"
    13 *-* old_name = "Fred"
    14 *-* SAY "Please enter name : "
    15 *-* PARSE PULL tst_name
    16 *-* PARSE VAR tst_name new_name .
    17 *-* IF new_name = old_name
       *-*  THEN
       *-*  DO
    18 *-*   SAY "Hello Fred"
    20 +++   ELSE
Error running TRACE, line 20: Unexpected THEN or ELSE
***
```

5 - 17

? And ! In the TRACE Instruction

- ? - Toggles interactive debugging
  - display change variables
  - execute instructions
  - used with R and I e.g TRACE ?I
- ! - Toggles host command execution inhibit
  - will not execute TSO commands
  - use with C, R and I, or by itself.

5 - 18

Error Trapping

- ? - Terminate or continue
- SIGNAL instruction
  - SIGNAL ON NOVALUE
  - SIGNAL ON SYNTAX
  - SIGNAL ON FAILURE
  - SIGNAL ON ERROR
  - SIGNAL ON HALT
- Ends with an EXIT or RETURN - may not pass back any information

5 - 19

ERROR - a TSO command ends with a non-zero return code

FAILURE - a TSO command ends with a negative return code

HALT - the terminal attention key is pressed and a HI (Halt Interpretation) command is entered

NOVALUE - an uninitialized variable is used in an expression or in a PARSE template

SYNTAX - a REXX interpretation syntax error happens

Error Trapping

- Special variables assigned in TSO/E REXX.
  - SIGL
    - variable holds line number in error
  - SOURCELINE
    - used to extract the relative line number of a line within the source for current REXX exec.
  - ERRORTEXT(RC)
    - used to extract from REXX the error message that is associated with a particular REXX error number.
  - CONDITION
    - used to retrieve the setting information for the currently trapped REXX condition.

5 - 20

Error Trapping

```
SIGNAL ON NOVALUE
IF new_name = old_name THEN DO
   SAY "Hello Fred"
END
EXIT

NOVALUE:
   SAY "Variable" CONDITION("D") has not been initialised on
   SAY "Line : "SIGL
   SAY "Source code : "SOURCELINE(SIGL)
```

```
Variable NEW_NAME HAS NOT BEEN INITIALISED ON
Line : 12
Source code : IF new_name = old_name THEN DO
***
```

5 - 21

Work Section 5.1

- Re-Write Work Section 1.2
- Test the different TRACE options.
  - A
  - C
  - E
  - F
  - I
  - L
  - N
  - O
  - R
  - S

5 - 22

Work Section 5.2

- Execute the following Program using Trace i? To see how REXX treats tails in compound variables.

```
TRACE I?
day = "Tuesday"
month.day = "May"
tuesday = "Tuesday"
SAY month.tuesday
```

5 - 23

Additional Program

Try using the TRACE() function instead

e.g.

TRACE(I)

What is the difference ?

Additional program

- Try using the TRACE() function instead
- e.g.
- TRACE(I)

- What is the difference ?

5 - 24

Looping Instructions

Section 6

6 - 1

DO

```
loop_counter = 0
DO 5
   loop_counter = loop_counter + 1
   SAY loop_counter
END
```

```
1
2
3
4
5
***
```

6 - 2

DO FOREVER

```
DO FOREVER
   SAY "Your name please : "
   PARSE UPPER EXTERNAL name
   PARSE VAR name fore_name .
   IF fore_name = "BOB" THEN DO
      EXIT
   END
END
```

```
 Your name please :
sdfgfd
 Your name please :
fg
 Your name please :
BOB
 ***
```

6 - 3

Test Exercise 61

- Write a REXX program to loop 5 times and show even numbers to the screen only.

```
Count is : 2
Count is : 4
***
```

6 - 4

DO count = 1 TO n BY n

```
DO loop_counter = 1 TO 5 BY 1
   SAY loop_counter
END
```

```
1
2
3
4
5
***
```

6 - 5

Test Exercise 62

- Write a REXX program to count to 10 showing only every third number.

```
1
4
7
10
***
```

6 - 6

DO WHILE

```
loop_counter = 0
DO WHILE loop_counter < 5
   loop_counter = loop_counter + 1
   SAY loop_counter
END
```

```
1
2
3
4
5
***
```

6 - 7

DO UNTIL

```
loop_counter = 0
DO UNTIL loop_counter < 5
   loop_counter = loop_counter + 1
   SAY loop_counter
END
```

```
1
***
```

6 - 8

ITERATE

```
DO loop_counter = 1 TO 5
   IF loop_counter = 3 THEN DO
      ITERATE
   END
   SAY loop_counter
END
```

```
1
2
4
5
***
```

6 - 9

LEAVE

```
DO loop_counter = 1 TO 5
   IF loop_counter = 3 THEN DO
      LEAVE
   END
   SAY loop_counter
END
```

```
1
2
***
```

6 - 10

Looping with Compound variables

```
DO loop_counter = 1 TO 3
   PARSE UPPER EXTERNAL new_name
   full_name.loop_counter = new_name
END
DO test_counter = 3 TO 1 BY -1
   SAY full_name.test_counter
END
```

```
BOB
GARY
FRED
FRED
GARY
BOB
***
```

6 - 11

Work Section 6.1

- Write a REXX program which will:
  - ask for 10 numbers from the user
  - assign the numbers to compound variables
  - output each variable and it's value
  - output the average of the 10 variables

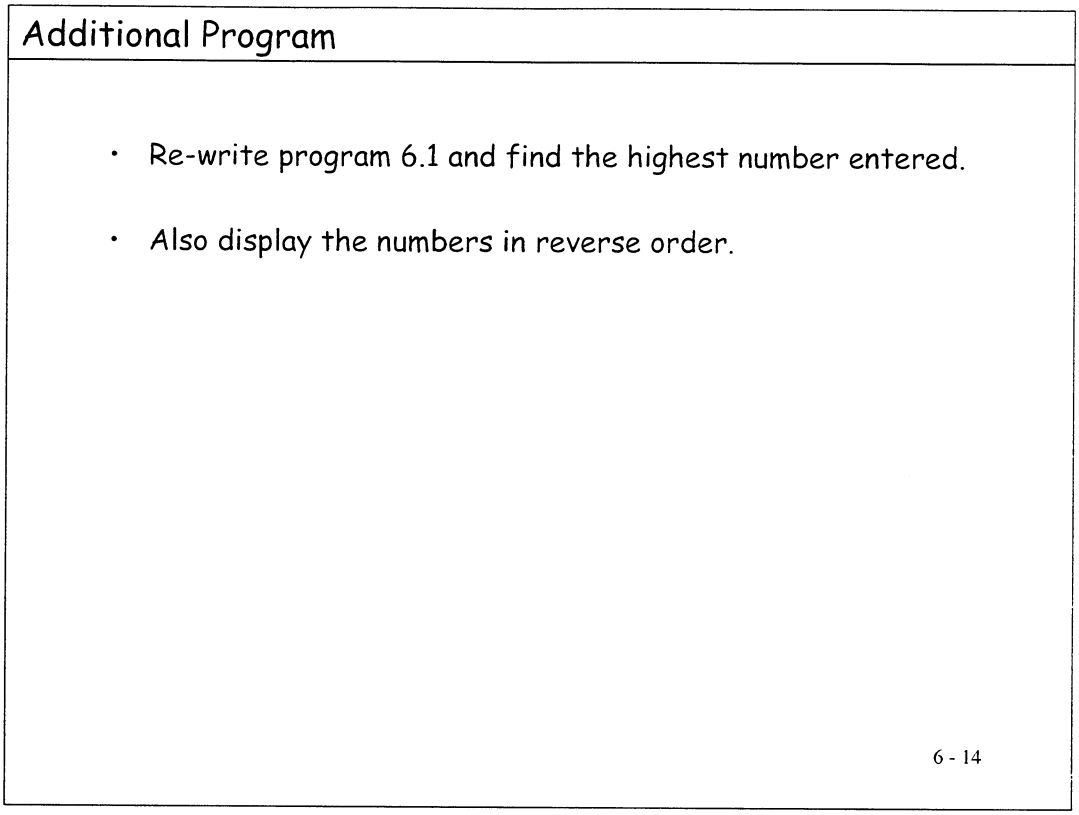
```
 Please enter a number:
1
 Please enter a number:
2
 Please enter a number:
3
 Please enter a number:
4
 Please enter a number:
5
                                                          1
                                                          2
                                                          3
                                                          4
                                                          5
 Average = 3
 ***
```

6 - 12

Work Section 6.2

- Re-write Work section 2.1 to allow the user to enter their name as many times as they want.
- Stop the program if they don't enter any more names.

6 - 13



Additional program

Re-write program 2.1 and find the highest number entered.

Functions

Section 7

7 - 1

What is a function?

- A pre-written sub-routine.
- A Function returns a value.
- The function name is suffixed with brackets, which are used for any arguments.
- REXX has a number of supplied functions.

7 - 2

DATATYPE

```
SAY DATATYPE("AA", A)
SAY DATATYPE("1", B)
SAY DATATYPE("A", L)
SAY DATATYPE("Aa", M)
SAY DATATYPE("1", N)
SAY DATATYPE("a", U)
SAY DATATYPE("1.2", W)
SAY DATATYPE("1", X)
SAY DATATYPE("1", B)
SAY DATATYPE("1", B)
SAY DATATYPE("1", B)
SAY DATATYPE("1", B)
SAY DATATYPE("1", B)
SAY
SAY DATATYPE("1")
SAY DATATYPE("A")
```

```
1
1
0
1
1
0
0
1
1
1
1
1
1

NUM
CHAR
***
```

7 - 3

A for Alphanumeric, returns a 1 if 'string' contains upper or lower case letters or the numbers 0 through 9.

N for Numeric, returns a 1 if 'string' is a valid number acceptable to REXX

W for Whole number, returns a 1 if 'string' is a whole number acceptable to REXX under the current setting of NUMERIC DIGITS.

L for Lowercase, returns a 1 if 'string' contains ONLY lowercase letters in the range A-Z.

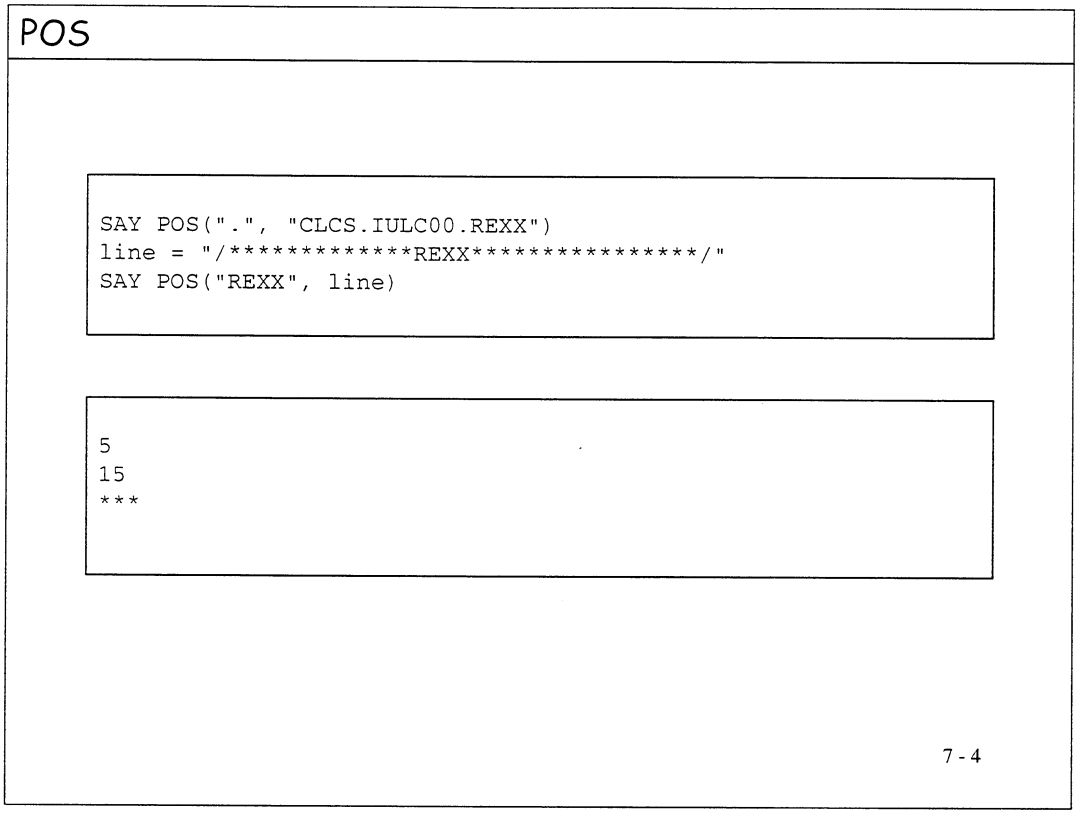
U for Uppercase, returns a 1 if 'string' contains ONLY uppercase letters in the range A-Z.

M for Mixed case, returns a 1 if 'string' contains either upper or lowercase letters in the range A-Z.

S for Symbol, returns a 1 if 'string' contains only the characters that are valid for forming REXX symbols.

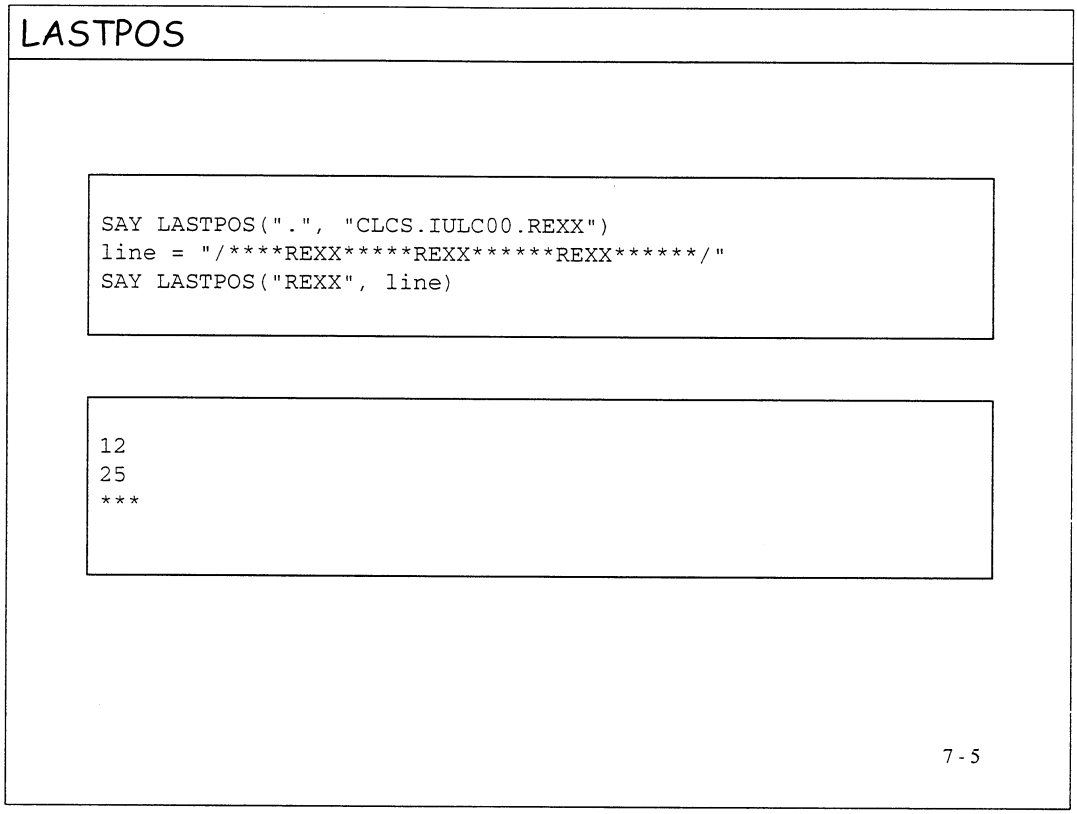
B for Bitstring, returns a 1 if 'string' contains ONLY 0's and 1's.

X for Hexadecimal, returns a 1 if 'string' contains valid hexadecimal characters in the ranges a-f, A-F, 0-9, or blank. A 1 is also returned if 'string' is a null (zero length) string.



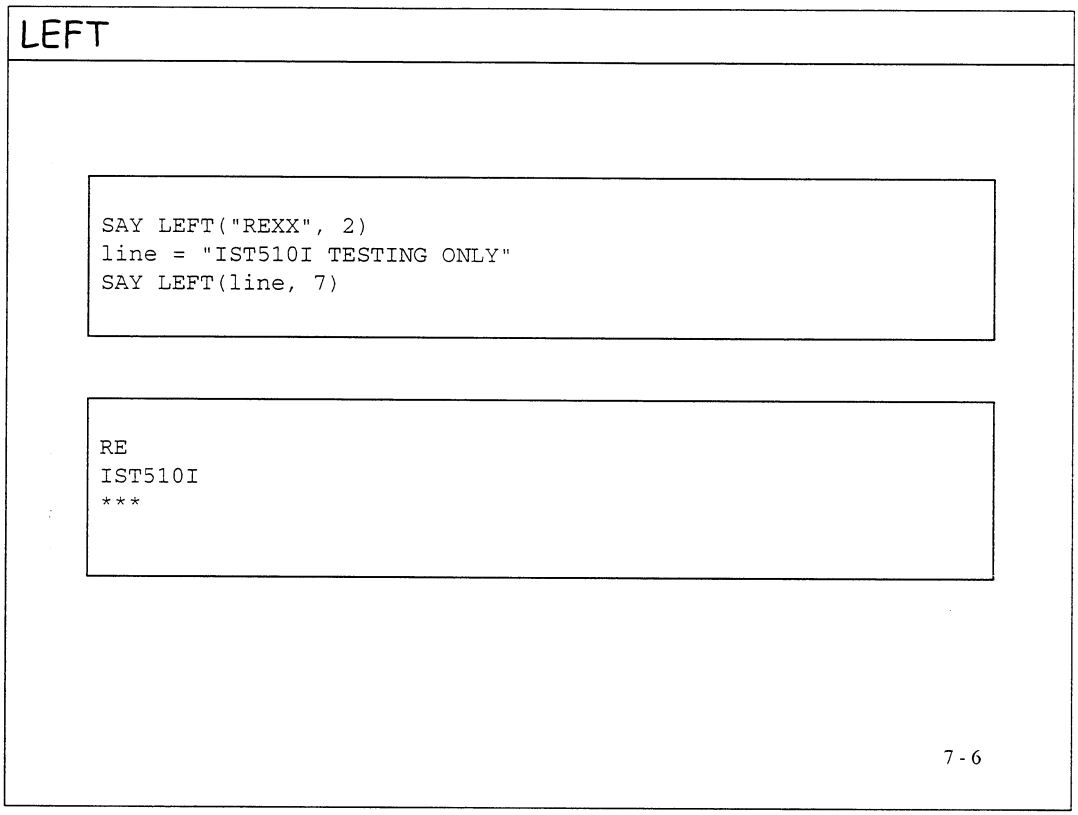
POS(substring,string{,startloc})

Returns the position of one string, in another.



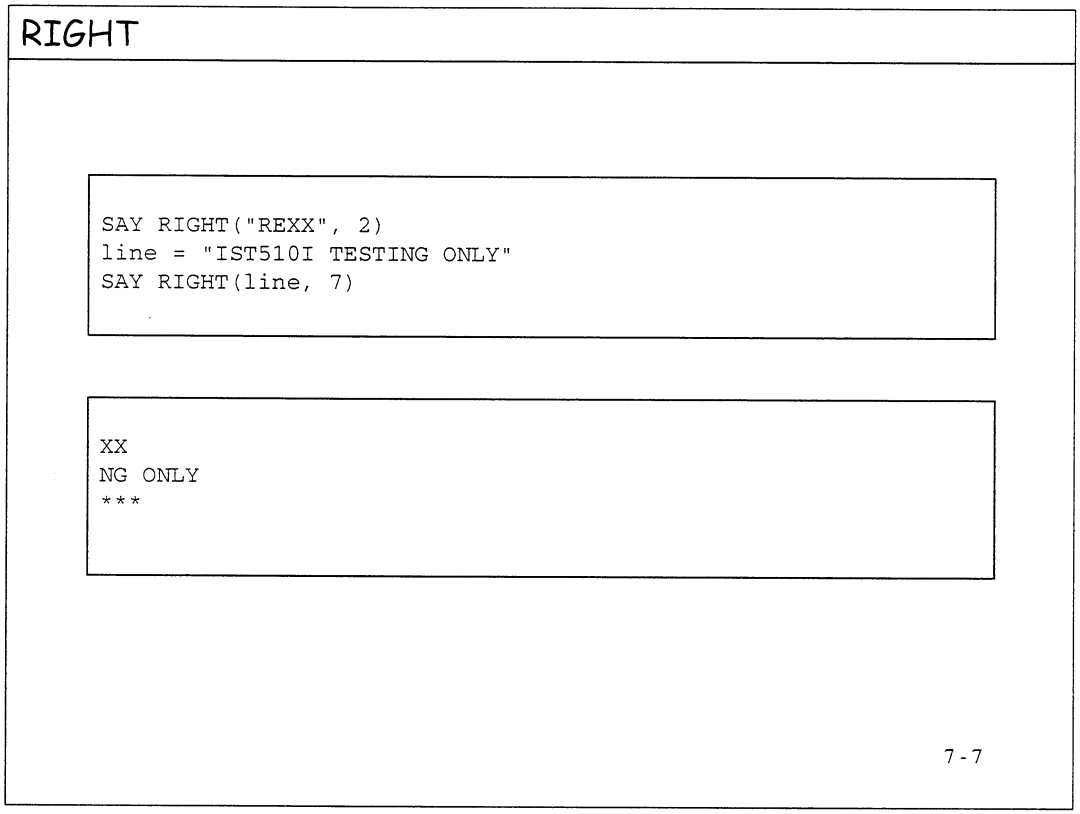
LASTPOS(substring,string{,start})

Returns the position of the last occurrence of one string in another.



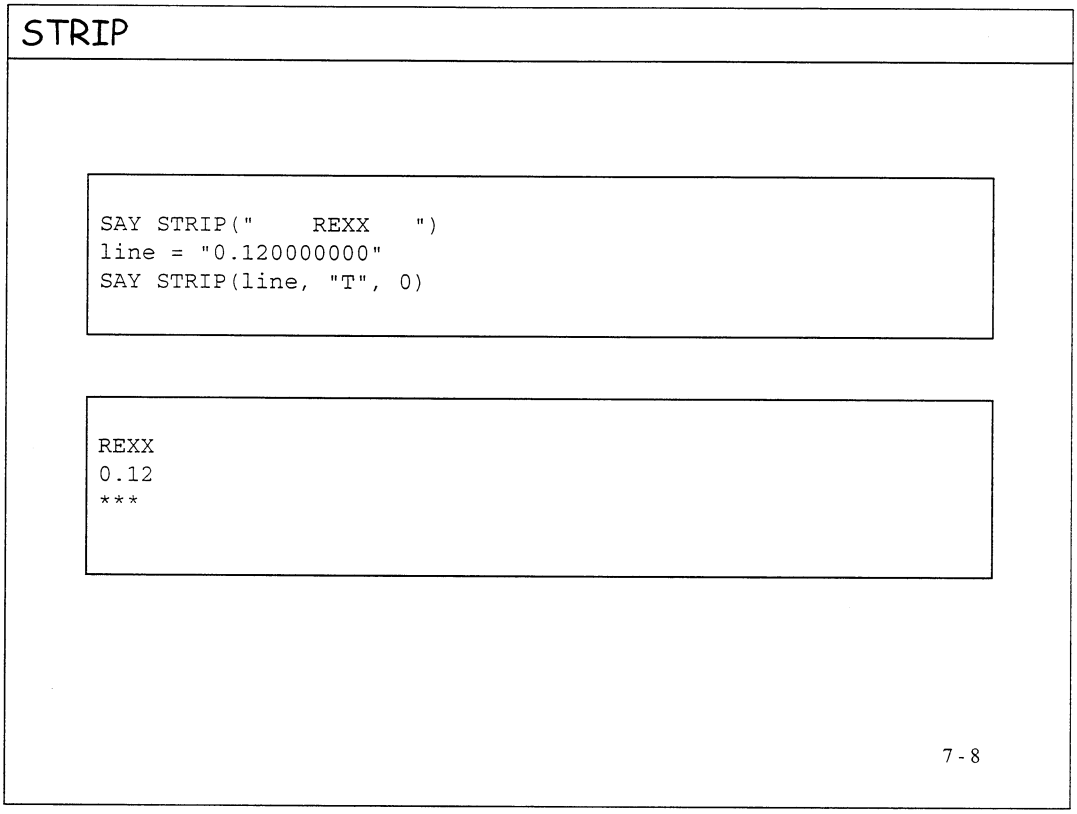
`LEFT(string,length{,pad})`

Returns a string of length, containing the leftmost length.



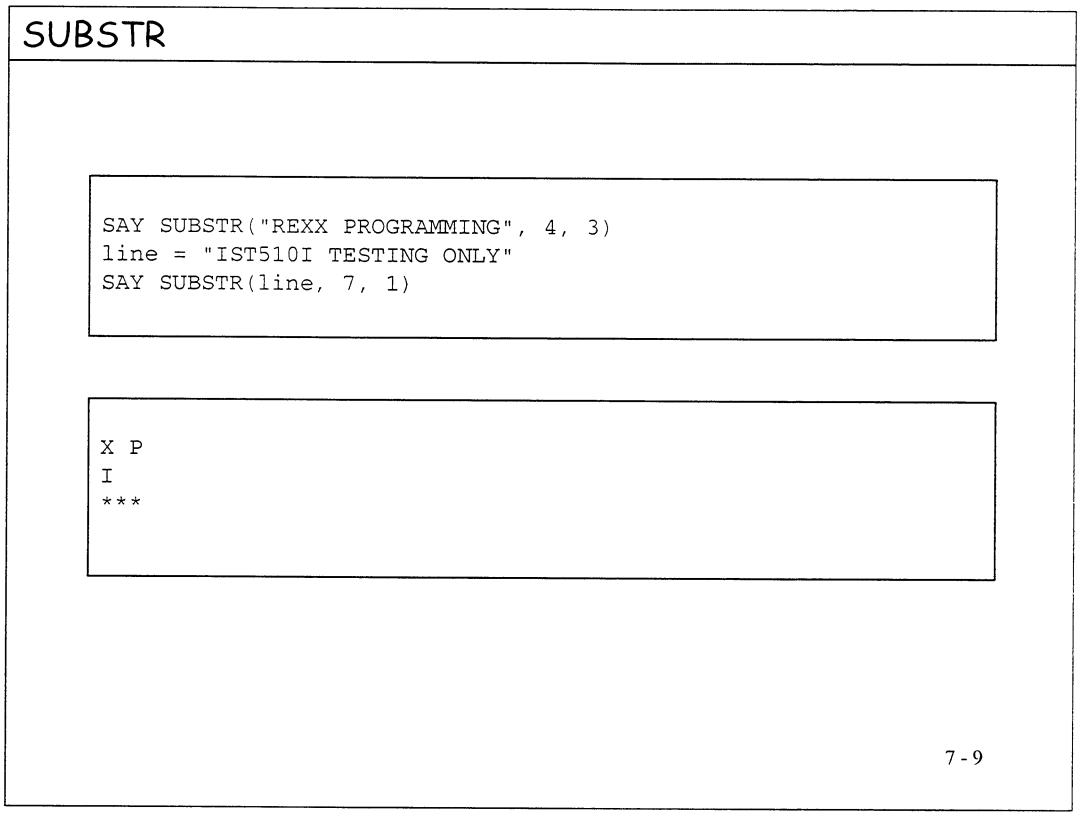
RIGHT(string,length{,pad})

Returns a string of length, containing the rightmost length.



STRIP(string{,{option}{,char}})

Returns string with leading or trailing characters or both removed.



SUBSTR(string,n{,{length}{,pad}})

Returns the substring of string that begins at the nth character and is of length

ABBREV

```
SAY ABBREV("REXX PROGRAMMING", "REXX P")
line = "CON"
IF ABBREV("CONFIRM", line, 1) = 1 THEN DO
   SAY "OK"
END
```

```
1
OK
***
```

7 - 10

ABBREV(string,prefix{,length})

Returns 1 if info is equal to the leading characters of information.

TRANSLATE

```
SAY translate("23.12.2000", "/", ".")
line = "CLCS.IULC00.REXX"
SAY TRANSLATE(line, " ", ".")
```

```
23/12/2000
CLCS IULC00 REXX
***
```

7 - 11

TRANSLATE(string{,{outtab}{,{intab}{,pad}}})

Returns string with each character translated to another character.

VERIFY

```
SAY VERIFY("CLCS.IULC00.REXX", "CLSIU0REX.ZYT")
SAY VERIFY("CLCS.IULC00.REXX", "Y")
line = "Mike DeParma"
SAY VERIFY(line, "MIKE", "N")
SAY VERIFY(line, "MIKE", "M")
```

```
0
1
2
1
***
```

7 - 12

`VERIFY(string1,string2{,{'opt'}{,begin}})`

Returns a number that, by default, indicates whether string is composed only of characters from reference.

DELSTR

```
SAY DELSTR("CLCS.IULC00.REXX", 6, 6)
line = "/*************REXX***************/"
SAY DELSTR(line, 15, 4)
```

```
CLCS..REXX
/**************************/
***
```

7 - 13

```
DELSTR(string,n---------------)
                  +-,length-+
```

Returns string after deleting the substring that begins at the nth character and is of length characters.

INSERT

```
SAY INSERT("IULC00", "CLCS..REXX", 5)
line = "/*****************************/"
SAY INSERT("REXX", line, 15)
```

```
CLCS.IULC00.REXX
/**************REXX**************/
***
```

7 - 14

INSERT(ins,target{,{n}{,{length}{,pad}}})

Inserts the string new, padded or truncated to length length, into the string target after the nth character.

OVERLAY

```
SAY OVERLAY("IULC22", "CLCS.IULC00.REXX", 6)
line = "/*************TEST***************/"
SAY OVERLAY("REXX", line, 15)
```

```
CLCS.IULC22.REXX
/*************REXX***************/
***
```

7 - 15

OVERLAY(new,target(,{n}{,{length}{,pad}}))

Returns the string target, which, starting at the nth character, is overlaid with the string new, padded or truncated to length length.

CENTRE

```
SAY CENTRE("CLCS.IULC00.REXX", 50)
line = "REXX"
SAY CENTRE(line, 79, "*")
```

```
           CLCS.IULC00.REXX
************************REXX*****************************
***
```

7 - 16

`CENTRE(string,length{,pad})`

Returns a string of length length with string centered in it, with pad characters added as necessary to make up length.

SPACE

```
SAY SPACE("This     is     the     REXX Course", 1)
line = "CLCS     IULCC00  REXX"
SAY SPACE(line, 1, ".")
SAY SPACE("A B C D E F G", 0)
SAY SPACE("A B C", 5)
```

```
This is the REXX Course
CLCS.IULCC00.REXX
ABCDEFG
A     B     C
***
```

7 - 17

`SPACE(string{,{n}{,pad}})`

Returns the blank-delimited words in string with n pad characters between each word.

FORMAT

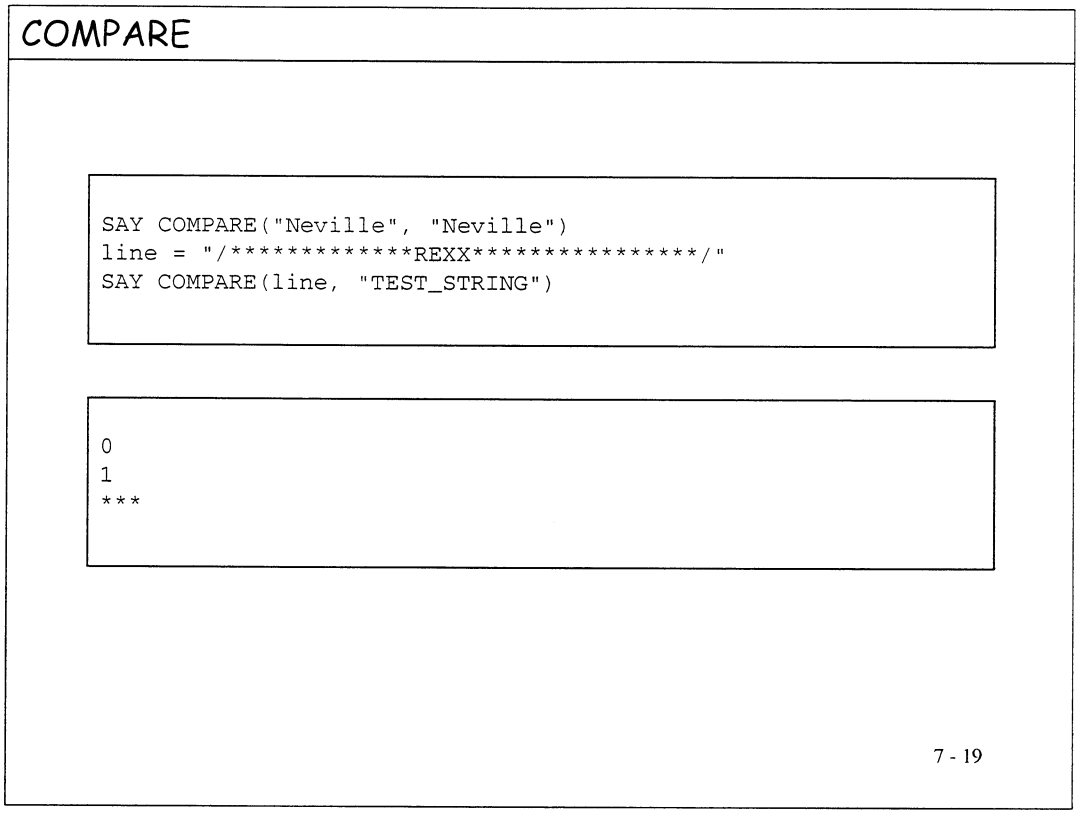
```
SAY FORMAT("12000", 10)
line = "3.5"
SAY FORMAT(line, 10)
SAY FORMAT("124.5656", 10, 2)
SAY FORMAT("17591.73",,,2,2)
```

```
     12000
         3.5
       124.57
1.759173E+04
***
```

7 - 18

FORMAT(number{,{before}}{,{after}})

Returns number, rounded and formatted.



`COMPARE(string1,string2(,pad))`

Returns 0 if the strings, string1 and string2, are identical. Otherwise, returns the position of the first character that does not match.

COPIES

```
SAY COPIES("REXX ", 5)
line = "GREAT"
SAY COPIES(line, 3)
```

```
REXX REXX REXX REXX REXX
GREATGREATGREAT
***
```

7 - 20

`COPIES(string,n)`

Returns n concatenated copies of string.

LENGTH

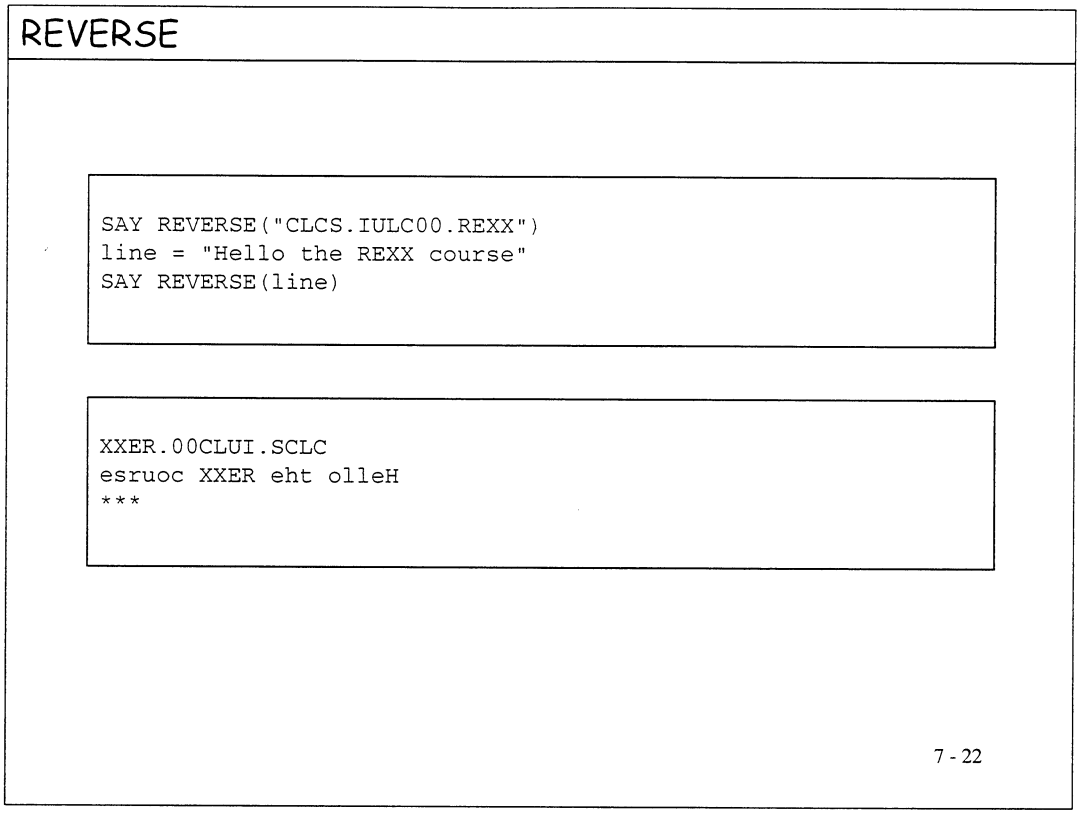
```
SAY LENGTH("CLCS.IULC00.REXX")
line = "/**************REXX***************/"
SAY LENGTH(line)
```

```
16
35
***
```

7 - 21

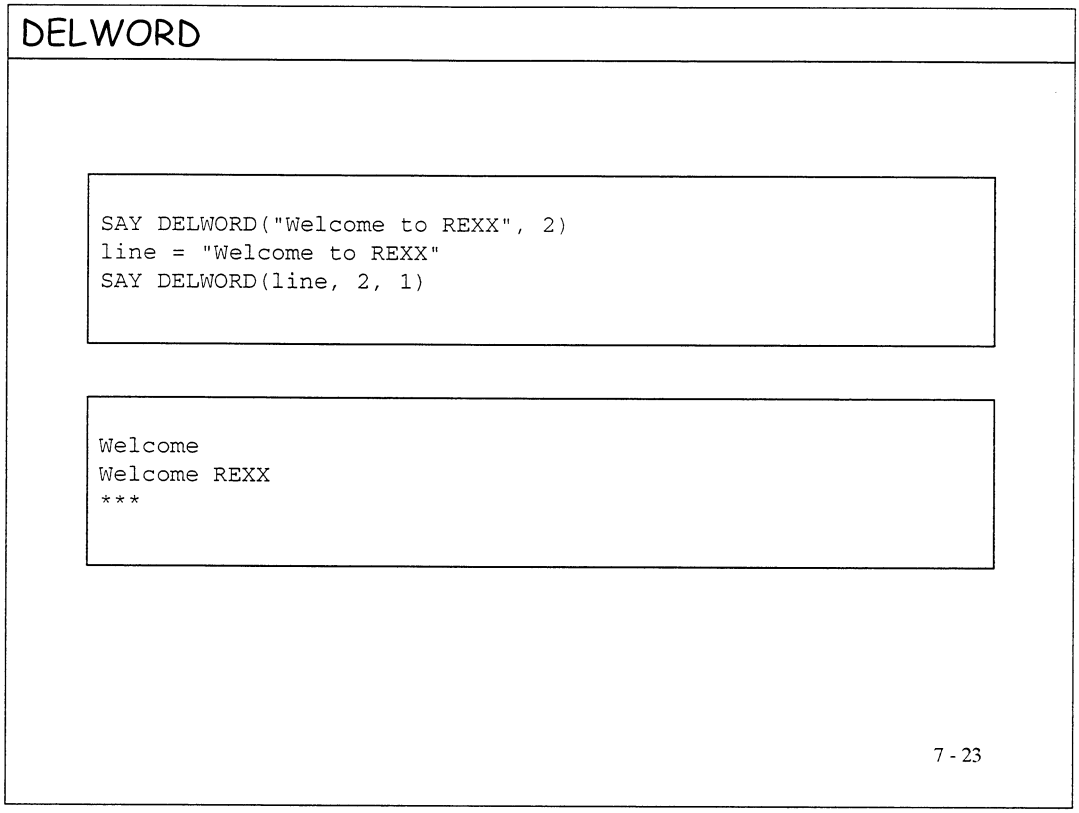
`LENGTH(string)`

Returns the length of string.



REVERSE(string)

Returns string, swapped end for end.



DELWORD(string,n{,length})

Returns string after deleting the substring that starts at the nth word and is of length blank-delimited words.

SUBWORD

```
SAY SUBWORD("Welcome to REXX programmers", 2)
line = "Welcome to REXX programmers"
SAY SUBWORD(line, 2, 2)
```

```
to REXX programmers
to REXX
***
```

7 - 24

SUBWORD(string,n{,length})

Returns the substring of string that starts at the nth word, and is up to length blank-delimited words.

WORD

```
SAY WORD("Welcome to REXX programmers", 2)
line = "Welcome to REXX programmers"
SAY WORD(line, 3)
```

```
to
REXX
***
```

7 - 25

`WORD(string,n)`

Returns the nth blank-delimited word in string or returns the null string if fewer than n words are in string.

WORDINDEX

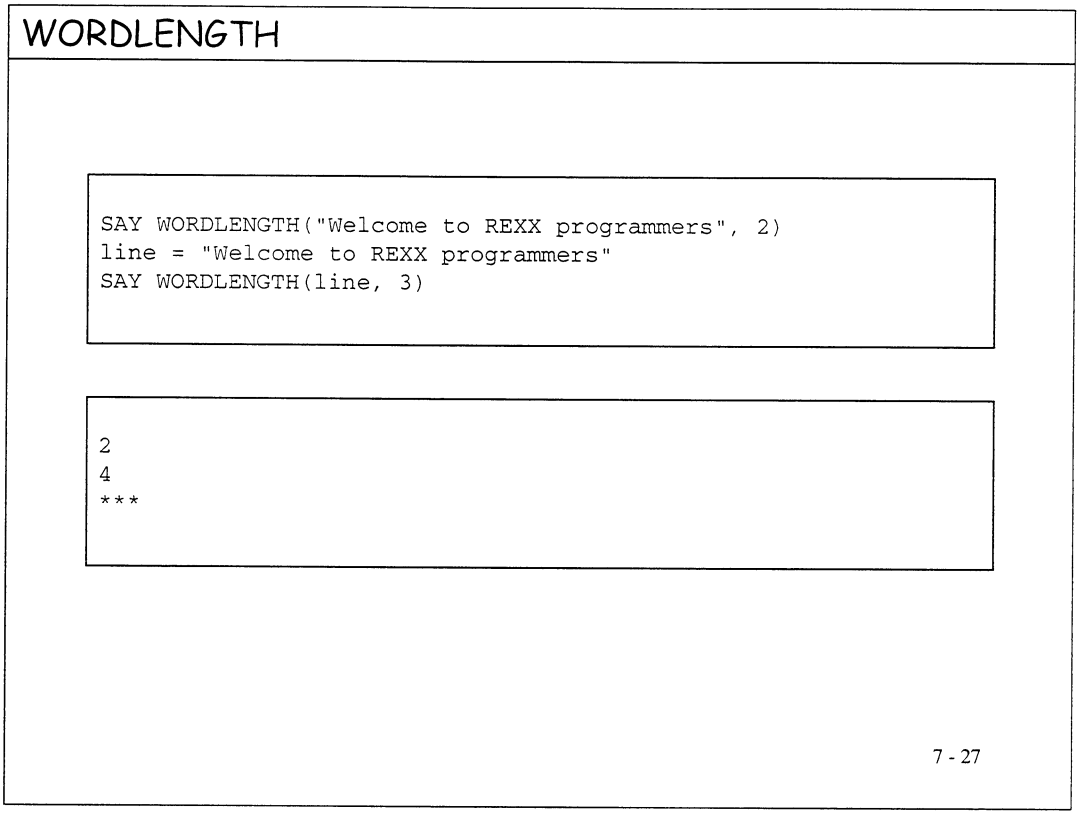
```
SAY WORDINDEX("Welcome to REXX programmers", 3)
line = "Welcome to REXX programmers"
SAY WORDINDEX(line, 4)
```

```
12
17
***
```

7 - 26

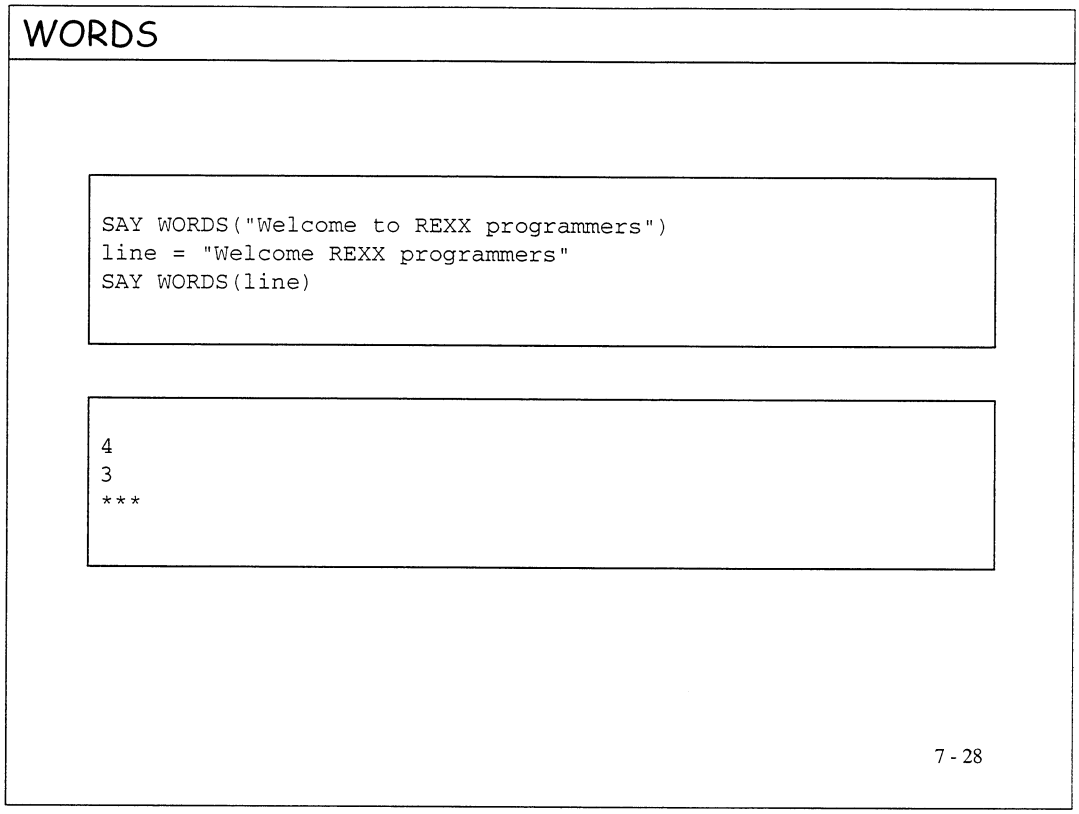
WORDINDEX(string,n)

Returns the position of the first character in the nth blank-delimited word in string or returns 0 if fewer than n words are in string.



WORDLENGTH(string,n)

Returns the length of the nth blank-delimited word in string or returns 0 if fewer than n words are in string.



`WORDS(string)`

Returns the number of blank-delimited words in string.

Arithmetic Functions

```
SAY ABS(-32)
SAY ABS(32)
SAY MIN(234, 3245, 3, 234)
SAY MAX(234, 3245, 3, 234)
SAY RANDOM(1, 49)
SAY SIGN(-32)
SAY TRUNC(213.1487876, 2)
```

```
32
32
3
3245
9
-1
213.14
***
```

7 - 29

ABS(number)

MIN(number{,number}...)

MAX(number{,number}...)

RANDOM({min}{,{max}{,seed}})

SIGN(number)

TRUNC(number{,n})

DATE

```
SAY DATE()
SAY DATE("B")
SAY DATE("C")
SAY DATE("D")
SAY DATE("E")
SAY DATE("J")
SAY DATE("M")
SAY DATE("O")
SAY DATE("S")
SAY DATE("U")
SAY DATE("W")
```

```
3 Feb 2000
730152
34
34
03/02/00
00034
February
00/02/03
20000203
02/03/00
Thursday
***
```

7 - 30

C as in Century, returns number of days into the current century in the form 'ddddd' with leading zeros stripped.

D for Days, returns number of days into the current year in the form 'ddd' with leading zeros stripped.

U returns date in USA format, 'mm/dd/yy'

E for European dates, returns the date in the form 'dd/mm/yy'

J returns a Julian date in the form 'yyddd'

M returns the full name of the current month (e.g., December)

O as in 'Ordered' date, returns the date in the form 'yy/mm/dd', allowing sorting in chronological date sequence

S returns an alternative sort-format date in the form 'yyyymmdd'

W returns the day of the week (e.g. 'Tuesday', 'Sunday', etc.)

TIME

```
SAY TIME()
SAY TIME("E")
SAY TIME("H")
SAY TIME("L")
SAY TIME("M")
SAY TIME("R")
SAY TIME("S")
```

```
05:18:59
0
5
05:18:59.066418
318
0.000432
19139
***
```

7 - 31

E - Causes TIME to return the elapsed time since a previous TIME('R') in seconds and microseconds.

R - Causes TIME to return the elapsed time since a previous TIME('R') in seconds and microseconds.

H - Causes TIME to return the number of hours since midnight on the same day in the form 'hh', with no leading zeros

L - Causes TIME to return the current time of day in the form hh:mm:ss.uuuuuu, where 'uuuuuu' is a fractional number of seconds, in microseconds (millionths of seconds)

M - Causes TIME to return the current time of day as the number of minutes since midnight on the same day in the form mmmm, with no leading zeros

S - Causes TIME to return the current time of day as the number of seconds since midnight on the same day in the form sssss, with no leading zeros

Work Section 7.1

- Write a REXX program which will:
  - Format a title in the the centre of the screen and underlined.
  - Show today's date in the format : mm/dd/yy
  - Show the time in the format : hh:mm:ss
  - Show the date in the format DD-MM-YY

```
                    Function Program
                    ================

The american formatted date is : 02/03/00
This program was executed at : 05:42:57
The european date : 03-02-00
***
```

7 - 32

Work Section 7.2

- Write REXX program to prompt for a Name and check that is your name, the program can accept any way of writing your name. If it is not your name loop round until your name is entered or the word "STOP"
  - E.g. FRED SMITH or FRED or FRED S
- Ask for a selection of 4 numbers.
  - Show the highest number
  - Show the lowest Number

7 - 33

```
 Please enter your name :
mick smith
 Please enter your name :
mike de
 Please enter four numbers
1
2
3
6
 The highest number is : 6
 The lowest number is : 1
 ***
```

Additional Program

- Write a REXX program to play a guess the number game.

```
                          Number game
                          ============

 Please Guess the number (1-100) :
50
 Too high
 Please Guess the number (1-100) :
25
 Too high
 Please Guess the number (1-100) :
10
 Too high
 Please Guess the number (1-100) :
5
 Too low
 Please Guess the number (1-100) :
7
 Hurrah you guessed the number in 5 guesses.
 ***
```

7 - 34

Subroutines and Functions

Section 8

8 - 1

What are they?

- Sections of a program they perform specific tasks.
- Can be branch to from anywhere in the program
- Can be inside or outside the program
- Created as a routine or a function

8 - 2

N.B.

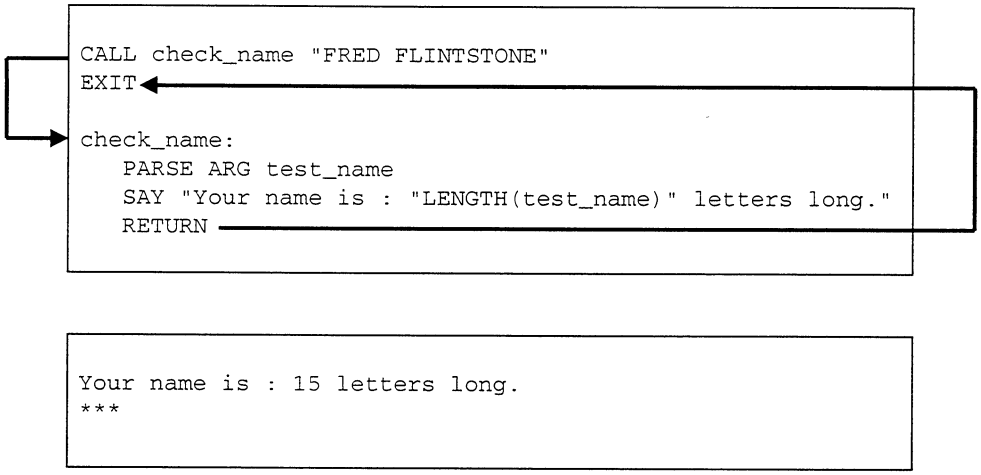
Functions should always return a value.

CALL and Commas

CALL with	PARSE With	Result
CALL subrt 1 2 3	ARG first second third	First = "1" Second = "2" Third = "3"
CALL subrt 1 2 3	ARG first, second, third	First = "1 2 3" Second = "" Third = ""
CALL subrt 1, 2, 3	ARG first second third	First = "1" Second = "" Third = ""
CALL subrt 1, 2, 3	ARG first, second, third	First = "1" Second = "2" Third = "3"

8 - 3

Internal Sample Program




```
Your name is : 15 letters long.
***
```

8 - 4

External Sample Program

```
CALL chckname "FRED FLINTSTONE"
```

```
/********************** REXX ***************************/
/* Program                                             */
/* chckname                                            */
/* Arguments                                           */
/* one argument which is the string to be checked      */
/* Description                                         */
/* REXX routine to return the length of a string       */
/* Author       : Michaelangelo DeParma                */
/* Date         : 3rd February 2000                    */
/*---------------- Amendment History -------------------*/
/*****************************************************/
PARSE ARG test_name, rubbish
SAY "Your name is : "LENGTH(test_name)" letters long."
RETURN
```

8-5

The executing program calls the member (in this example) in the same dataset called chckname.

Internal Sample Function

```
ARG nm1 nm2 nm3 unused
check_name = name_check(nm1 nm2 nm3)
IF check_name = 1 THEN DO
   SAY "All the names are in this department."
END
ELSE DO
   SAY "NOT all the names are in this department."
END
EXIT

name_check:
   ARG name1 name2 name3
   IF name1 = "FRED" & name2 = "BOB" & name3 = "JANE" THEN DO
      name_result = 1
   END
   ELSE DO
      name_result = 0
   END
   RETURN name_result
```

8 - 6

Internal Sample Function

```
ex 'crone90.crone.rx101(sample9)' 'fred bob jane'



All the names are in this department.
***
```

8 - 7

External Sample Function

```
ARG nm1 nm2 nm3 unused
check_name = namechek(nm1 nm2 nm3)
IF check_name = 1 THEN DO
   SAY "All the names are in this department."
END
ELSE DO
   SAY "NOT all the names are in this department."
END
```

8 - 8

Work Section 8.1

- Write a REXX program which add a series of numbers that are passed to the subroutine

```
CALL addup 1 2 3 4 5 6 7 8
```

```
The total of : 1 2 3 4 5 6 7 8
is : 36
***
```

8 - 9

Work Section 8.2

- Re-write work section 8.1 as an external function.

```
total = addup(1 2 3 4 5 6 7 8)
SAY "The total is : "total
```

```
The total is : 36
***
```

8 - 10

Additional Program

- Write an external REXX range function to check if a number is within a given range.

```
low = 1
high = 100
number_check = RANGECHK(low high number)
```

```
 Please enter the number you wish to check :
123
 The number is out of range 1 to 100
 ***
```

8 - 11

Additional Program

- Create a reverse word function which will reverse the order of words passed to it.

```
SAY "Please enter your list of words : "
PARSE PULL list_of_words
list_of_words = STRIP(list_of_words)
SAY "The list is : "||REVWORD(list_of_words)
```

```
  Please enter your list of words :
one two three
 The list is :  THREE TWO ONE
 ***
```

8 - 12

Work Section 1

1.

```
/****************************** REXX ***********************************/
/* Program                                                             */
/* rx10111                                                             */
/* Description                                                         */
/* REXX program to display a list of names                             */
/* Author       : Michaelangelo DeParma                                */
/* Date         : 1st January 2000                                     */
/*----------------------- Amendment History -------------------------*/
/*********************************************************************/
SAY "Bob Flemming"
SAY "Gary Stinker"
SAY "John Thomas"
```

2.

```
/****************************** REXX ***********************************/
/* Program                                                             */
/* rx10112                                                             */
/* Description                                                         */
/* REXX program to welcome a name to the screen                        */
/* Author       : Michaelangelo DeParma                                */
/* Date         : 1st January 2000                                     */
/*----------------------- Amendment History -------------------------*/
/*********************************************************************/
SAY "Please enter your name :"
PULL your_name
SAY
SAY "Hello "your_name
SAY "Welcome to the course."
```

Additional Program

```
/****************************** REXX ***********************************/
/* Program                                                             */
/* rx10113                                                             */
/* Description                                                         */
/* REXX program to welcome a name to the screen                        */
/* Author      : Michaelangelo DeParma                                 */
/* Date        : 1st January 2000                                      */
/*----------------------- Amendment History -------------------------*/
/***********************************************************************/
SAY "Please enter your forename : "
PULL fore_name
SAY
SAY "Please enter your surname : "
PULL sur_name
SAY
SAY "Welcome to the course - "fore_name sur_name
```

Addition Program 2

```
/****************************** REXX ***********************************/
/* Program                                                             */
/* rx10114                                                             */
/* Description                                                         */
/* REXX program to welcome a name to the screen                        */
/* Author      : Michaelangelo DeParma                                 */
/* Date        : 1st January 2000                                      */
/*----------------------- Amendment History -------------------------*/
/***********************************************************************/
CLEAR
SAY "Please enter your Forename and Surname :"
PULL fore_name sur_name
SAY
SAY "Welcome to the course - "sur_name fore_name
```

Test21

```
/****************************** REXX ***********************************/
/* Program                                                             */
/* test21                                                              */
/* Description                                                         */
/* REXX program to test statements                                     */
/* Author       : Michaelangelo DeParma                                */
/* Date         : 1st January 2000                                     */
/*---------------------- Amendment History -------------------------*/
/*********************************************************************/
CLEAR
/* REXX EXEC */
a = "REXX EXEC"
c = 94 4F
d = '94 4F'
e = 32
f = '32'
g = "32"x
b = "REXX EXEC"x
REXX EXEC
```

Test22

```
/***************************** REXX ***********************************/
/* Program                                                            */
/* test22                                                             */
/* Description                                                        */
/* REXX program to test statements                                    */
/* Author       : Michaelangelo DeParma                               */
/* Date         : 1st January 2000                                    */
/*----------------------- Amendment History -------------------------*/
/**********************************************************************/
CLEAR
Answer = "yes"
The-Answer = "no"
Msg.Text = "IST510I"
1ST_Class = "never"
the_worlds_largest_variable = "test"
REPLLY? = "Z NET,QUICK"
SAY reply
```

Work Section 2

1.

```
/****************************** REXX ***********************************/
/* Program                                                             */
/* rx10121                                                             */
/* Description                                                         */
/* REXX program to check concatenation.                                */
/* Author       : Michaelangelo DeParma                                */
/* Date         : 1st January 2000                                     */
/*---------------------- Amendment History -------------------------*/
/*********************************************************************/
CLEAR
SAY "Please enter your first name."
PARSE PULL first_name
SAY "Please enter your last name."
PARSE PULL last_name
SAY first_name last_name
SAY last_name"," first_name
SAY last_name||first_name
SAY first_name||last_name
```

2.

```
/****************************** REXX ***********************************/
/* Program                                                             */
/* rx10122                                                             */
/* Description                                                         */
/* REXX program to calculate TAX.                                      */
/* Author       : Michaelangelo DeParma                                */
/* Date         : 1st January 2000                                     */
/*---------------------- Amendment History -------------------------*/
/*********************************************************************/
CLEAR
SAY "Please enter your first name."
PARSE PULL first_name
SAY "Please enter your salary."
PARSE PULL salary
tax = .3 * salary
SAY "Your tax paid is :"tax
```

Work Section 3

1.

```
/****************************** REXX ***********************************/
/* Program                                                             */
/* rx10131                                                             */
/* Description                                                         */
/* REXX program to accept an argument.                                 */
/* Author       : Michaelangelo DeParma                                */
/* Date         : 1st January 2000                                     */
/*---------------------- Amendment History -------------------------*/
/*********************************************************************/
CLEAR
PARSE ARG name
SAY "Hello "name
```

2.

```
/****************************** REXX ***********************************/
/* Program                                                             */
/* rx10132                                                             */
/* Description                                                         */
/* REXX program to accept an argument and split the dataset name.      */
/* Author       : Michaelangelo DeParma                                */
/* Date         : 1st January 2000                                     */
/*---------------------- Amendment History -------------------------*/
/*********************************************************************/
CLEAR
PARSE ARG data_set
PARSE VAR data_set project "." group "." type
SAY "Project : "project
SAY "Group   : "group
SAY "Type    : "type
```

Additional Program

```
/****************************** REXX ***********************************/
/* Program                                                             */
/* rx10133                                                             */
/* Description                                                         */
/* REXX program to welcome a name to the screen                        */
/* Author       : Michaelangelo DeParma                                */
/* Date         : 1st January 2000                                     */
/*---------------------- Amendment History -------------------------*/
/*********************************************************************/
CLEAR
SAY "Please enter your name :"
PULL name
PARSE VAR name your_name .
SAY
SAY "Hello "your_name
SAY "Welcome to the course."
```

Test 41

```
test_value = 1
IF test_value = 1 THEN
   SAY "Yes"
IF test_value = 1 THEN
   SAY "Yes"
IF test_value = 1 THEN
   SAY "Yes"
ELSE
   SAY "No"
IF test_value = 1 THEN
   SAY "Yes"
ELSE
   SAY "No"
IF test_value = 1
   THEN
      SAY "Yes"
   ELSE
      SAY "No"
```

Work Section 4

1.

```
/****************************** REXX ***********************************/
/* Program                                                             */
/* rx10141                                                             */
/* Description                                                         */
/* REXX program to test IF statements                                  */
/* Author      : Michaelangelo DeParma                                 */
/* Date        : 1st January 2000                                      */
/*---------------------- Amendment History -------------------------*/
/*********************************************************************/
CLEAR
PARSE ARG age
IF age > 64 THEN
   SAY "65 or over"
IF age < 65 & age > 21 THEN
   SAY "Over 21 and under 65"
IF age >= 16 & age <= 21 THEN
   SAY "Between 16 and 21"
IF age < 16 THEN
SAY "Under 16"
```

2.

```
/****************************** REXX ***********************************/
/* Program                                                             */
/* rx10142                                                             */
/* Description                                                         */
/* REXX program to test IF statements                                  */
/* Author      : Michaelangelo DeParma                                 */
/* Date        : 1st January 2000                                      */
/*---------------------- Amendment History -------------------------*/
/*********************************************************************/
CLEAR
PARSE ARG age
SELECT
   WHEN age < 16 THEN SAY "Under 16"
   WHEN age <=21 THEN SAY "Between 16 and 21"
   WHEN age < 65 THEN SAY "Over 21 and under 65"
   OTHERWISE SAY "65 or over"
END
```

Additional Program

```
/****************************** REXX ***********************************/
/* Program                                                             */
/* rx10143                                                             */
/* Description                                                         */
/* REXX program to test IF statements                                  */
/* Author       : Michaelangelo DeParma                                */
/* Date         : 1st January 2000                                     */
/*----------------------- Amendment History -------------------------*/
/***********************************************************************/
CLEAR
SAY "Please enter your salary."
PARSE UPPER PULL salary un_used
SAY "Please enter your TAX band."
PARSE UPPER PULL tax un_used
SELECT
   WHEN tax = 10 THEN DO
      tax_paid = salary * 0.1
   END
   WHEN tax = 20 THEN DO
      tax_paid = salary * 0.2
   END
   WHEN tax = 50 THEN DO
      tax_paid = salary * 0.5
   END
   WHEN tax = 70 THEN DO
      tax_paid = salary * 0.7
   END
   WHEN tax = 80 THEN DO
      tax_paid = salary * 0.8
   END
   OTHERWISE DO
      SAY "Incorrect tax code : "tax
      EXIT
   END
END
SAY "Your tax for :" salary ": is :" tax_paid
```

Work Section 5

1.

```
/***************************** REXX ***********************************/
/* Program                                                            */
/* rx10151                                                            */
/* Description                                                        */
/* REXX program to welcome a name to the screen and test TRACE        */
/* Author       : Michaelangelo DeParma                               */
/* Date         : 2nd February 2000                                   */
/*----------------------- Amendment History -------------------------*/
/**********************************************************************/
CLEAR
TRACE A
SAY "Please enter your name :"
PULL your_name
SAY
SAY "Hello "your_name
SAY "Welcome to the course."
```

2.

```
/***************************** REXX ***********************************/
/* Program                                                            */
/* rx10152                                                            */
/* Description                                                        */
/* REXX program to welcome a name to the screen and test TRACE        */
/* Author       : Michaelangelo DeParma                               */
/* Date         : 2nd February 2000                                   */
/*----------------------- Amendment History -------------------------*/
/**********************************************************************/
CLEAR
TRACE I?
day = "Tuesday"
month.day = "May"
tuesday = "Tuesday"
SAY month.Tuesday
```

Additional Program

```
/***************************** REXX ***********************************/
/* Program                                                            */
/* rx10153                                                            */
/* Description                                                        */
/* REXX program to welcome a name to the screen and test TRACE        */
/* Author      : Michaelangelo DeParma                                */
/* Date        : 2nd February 2000                                    */
/*----------------------- Amendment History -------------------------*/
/**********************************************************************/
CLEAR
TRACE(I)
Day = "Tuesday"
Month.day = "May"
Tuesday = "Tuesday"
SAY month.Tuesday
```

Test 61

```
/***************************** REXX ***********************************/
/* Program                                                            */
/* test61                                                             */
/* Description                                                        */
/* REXX program to test do statements                                 */
/* Author       : Michaelangelo DeParma                               */
/* Date         : 2nd February 2000                                   */
/*----------------------- Amendment History -------------------------*/
/**********************************************************************/
CLEAR
Loop_counter = 0
DO 5
   loop_counter = loop_counter + 1
   even_result = loop_counter // 2
   IF even_result = 0 THEN DO
      SAY "Count is : "loop_counter
   END
END
```

Test 62

```
/***************************** REXX ***********************************/
/* Program                                                            */
/* test62                                                             */
/* Description                                                        */
/* REXX program to test DO statements                                 */
/* Author       : Michaelangelo DeParma                               */
/* Date         : 2nd February 2000                                   */
/*----------------------- Amendment History -------------------------*/
/**********************************************************************/
CLEAR
DO loop_counter = 1 TO 10 BY 3
   SAY loop_counter
END
```

Work Section 6

1.

```
/****************************** REXX ***********************************/
/* Program                                                             */
/* rx10161                                                             */
/* Description                                                         */
/* REXX program to test looping.                                       */
/* Author      : Michaelangelo DeParma                                 */
/* Date        : 2nd February 2000                                     */
/*---------------------- Amendment History -------------------------*/
/***********************************************************************/
CLEAR
Total = 0
DO loop_counter = 1 TO 5
   SAY "Please enter a number:"
   PARSE PULL number.loop_counter
   total = total + number.loop_counter
END
DO new_counter = 1 TO 5
   SAY number.new_counter
END
Average = total / 5
SAY "Average = "average
```

1.a.

```
/***************************** REXX ***********************************/
/* Program                                                            */
/* rx10161a                                                           */
/* Description                                                        */
/* REXX program to test looping.                                      */
/* Author      : Michaelangelo DeParma                                */
/* Date        : 2nd February 2000                                    */
/*----------------------- Amendment History -------------------------*/
/**********************************************************************/
CLEAR
Total = 0
DO loop_counter = 1 TO 5
   SAY "Please enter a number:"
   PARSE PULL number.loop_counter
   IF highest < number.loop_counter THEN DO
      highest = number.loop_counter
   END
   total = total + number.loop_counter
END
DO new_counter = 1 TO 5
   SAY number.new_counter
END
Average = total / 5
SAY "Average = "average
SAY "Highest = "highest
```

2.

```
/***************************** REXX ***********************************/
/* Program                                                            */
/* rx10162                                                            */
/* Description                                                        */
/* REXX program to loop round work section 2.1                        */
/* Author       : Michaelangelo DeParma                               */
/* Date         : 2nd February 2000                                   */
/*----------------------- Amendment History -------------------------*/
/**********************************************************************/
CLEAR
DO FOREVER
   SAY "Please enter your first name."
   PARSE PULL first_name
   IF first_name = "" THEN DO
      EXIT
   END
   SAY "Please enter your last name."
   PARSE PULL last_name
   SAY first_name last_name
   SAY last_name"," first_name
   SAY last_name||first_name
   SAY first_name||last_name
END
```

Work Section 7

1.

```
/****************************** REXX ***********************************/
/* Program                                                             */
/* rx10171                                                             */
/* Description                                                         */
/* REXX program to test the use of functions.                          */
/* Author      : Michaelangelo DeParma                                 */
/* Date        : 3rd February 2000                                     */
/*---------------------- Amendment History -------------------------*/
/*********************************************************************/
CLEAR
Program_title = CENTRE("Function Program", 79)
Under_line = CENTRE("==================", 79)
SAY program_title
SAY under_line
SAY
SAY "The american formatted date is : "DATE("U")
SAY "This program was executed at : "TIME()
SAY "The european date : "TRANSLATE(DATE("E"), "-", "/")
```

2.

```
/***************************** REXX ***********************************/
/* Program                                                            */
/* rx10172                                                            */
/* Description                                                        */
/* REXX program to test the use of functions.                         */
/* Author      : Michaelangelo DeParma                                */
/* Date        : 3rd February 2000                                    */
/*---------------------- Amendment History -------------------------*/
/**********************************************************************/
CLEAR
DO FOREVER
   SAY "Please enter your name :"
   PARSE UPPER PULL name
   PARSE VAR name full_name .
   IF ABBREV("MIKE DEPARMA", full_name, 4) = 0 THEN DO
      ITERATE
   END
   ELSE DO
      LEAVE
   END
END
SAY "Please enter four numbers"
PARSE PULL one_num
PARSE PULL two_num
PARSE PULL three_num
PARSE PULL four_num
SAY "The highest number is : "MAX(one_num, two_num, three_num,
four_num)
SAY "The lowest number is : "MIN(one_num, two_num, three_num, four_num)
```

Additional Program.

```
/***************************** REXX ***********************************/
/* Program                                                            */
/* rx10172a                                                           */
/* Description                                                        */
/* REXX program to test the use of functions.                         */
/* Author      : Michaelangelo DeParma                                */
/* Date        : 3rd February 2000                                    */
/*---------------------- Amendment History -------------------------*/
/**********************************************************************/
CLEAR
Game_number = RANDOM(1, 100)
go_counter = 0
program_title = CENTRE("Number game", 79)
under_line = CENTRE("=============", 79)
SAY program_title
SAY under_line
SAY
DO FOREVER
   SAY "Please Guess the number (1-100) :"
   PARSE PULL player_guess
   go_counter = go_counter + 1
   IF player_guess = game_number THEN DO
      LEAVE
   END
   ELSE DO
      IF player_guess > game_number THEN DO
         SAY "Too high"
      END
      ELSE DO
         SAY "Too low"
      END
   END
END
SAY
SAY "Hurrah you guessed the number in "go_counter" guesses."
```

Work Section 8

1.

```
/***************************** REXX ********************************/
/* Program                                                         */
/* rx10181                                                         */
/* Description                                                     */
/* REXX program to test the use of site written routines           */
/* Author      : Michaelangelo DeParma                             */
/* Date        : 3rd February 2000                                 */
/*----------------------- Amendment History ----------------------*/
/***********************************************************************/
CLEAR
CALL addup 1 2 3 4 5 6 7 8
EXIT

addup:
   ARG numbers
   number_count = WORDS(numbers)
   total = 0
   DO loop_counter = 1 TO number_count
      total = total + WORD(numbers, loop_counter)
   END
   SAY "The total of : "numbers
   SAY "is : "total
   RETURN
```

2.

```
/****************************** REXX ***********************************/
/* Program                                                             */
/* rx10182                                                             */
/* Description                                                         */
/* REXX program to test the use of site written routines               */
/* Author       : Michaelangelo DeParma                                */
/* Date         : 3rd February 2000                                    */
/*---------------------- Amendment History -------------------------*/
/***********************************************************************/
CLEAR
total = addup(1 2 3 4 5 6 7 8)
SAY "The total is : "total
```

```
/****************************** REXX ***********************************/
/* Program                                                             */
/* addup                                                               */
/* Arguments                                                           */
/* All the numbers to added up.                                        */
/* Description                                                         */
/* REXX program to test the use of site written routines               */
/* Author       : Michaelangelo DeParma                                */
/* Date         : 3rd February 2000                                    */
/*---------------------- Amendment History -------------------------*/
/***********************************************************************/
ARG numbers
number_count = WORDS(numbers)
total = 0
DO loop_counter = 1 TO number_count
  total = total + WORD(numbers, loop_counter)
END
RETURN total
```

Additional Program

```
/****************************** REXX ***********************************/
/* Program                                                             */
/* rx10182a                                                            */
/* Description                                                         */
/* REXX program to test the use of site written routines               */
/* Author      : Michaelangelo DeParma                                 */
/* Date        : 3rd February 2000                                     */
/*---------------------- Amendment History -------------------------*/
/*********************************************************************/
CLEAR
low = 1
high = 100
SAY "Please enter the number you wish to check : "
PARSE PULL num
PARSE VAR num number .
number_check = RANGECHK(low high number)
IF number_check = 1 THEN DO
   SAY "The number is in the range "low" to "high
END
ELSE DO
   SAY "The number is out of range "low" to "high
END
```

```
/****************************** REXX ***********************************/
/* Program                                                             */
/* rangechk                                                            */
/* Arguments                                                           */
/* low - smallest number, high the highest number and                  */
/* the number to check.                                                */
/* Description                                                         */
/* REXX program to test the use of site written routines               */
/* Author      : Michaelangelo DeParma                                 */
/* Date        : 3rd February 2000                                     */
/*---------------------- Amendment History -------------------------*/
/*********************************************************************/
ARG low high number
IF number >= low & number <= high THEN DO
   check_result = 1
END
ELSE DO
   check_result = 0
END
RETURN check_result
```

Additional Program

```
/****************************** REXX ***********************************/
/* Program                                                             */
/* rx10184                                                             */
/* Description                                                         */
/* REXX program to test the use of REVWORD function                    */
/* Author      : Michaelangelo DeParma                                 */
/* Date        : 3rd February 2000                                     */
/*---------------------- Amendment History -------------------------*/
/*********************************************************************/
CLEAR
SAY "Please enter your list of words : "
PARSE PULL list_of_words
list_of_words = STRIP(list_of_words)
SAY "The list is : "||REVWORD(list_of_words)
```

```
/****************************** REXX ***********************************/
/* Program                                                             */
/* revword                                                             */
/* Arguments                                                           */
/* a word list.                                                        */
/* Description                                                         */
/* REXX program to reverse a list of words.                            */
/* Author      : Michaelangelo DeParma                                 */
/* Date        : 3rd February 2000                                     */
/*---------------------- Amendment History -------------------------*/
/*********************************************************************/
ARG word_list
new_list = ""
no_of_words = WORDS(word_list)
DO loop_counter = no_of_words TO 1 BY -1
   new_list = new_list||" "||WORD(word_list, loop_counter)
END
RETURN new_list
```