7) Built-in functions

- Built-in Function overview,
- Non SSA Built-in Functions
- TSO External Functions.

.

Resources: TSO/E REXX Reference

TSO/E REXX User's Guide

This course has been prepared by Milos Forman for MCoE needs only!

PROPRIETARY AND CONFIDENTIAL INFORMATION

These education materials and related computer software program (hereinafter referred to as the "Education Materials") is for the end user's informational purposes only and is subject to change or withdrawal by CA, Inc. at any time.

These Education Materials may not be copied, transferred, reproduced, disclosed or distributed, in whole or in part, without the prior written consent of CA. These Education Materials are proprietary information and a trade secret of CA. Title to these Education Materials remains with CA, and these Education Materials are protected by the copyright laws of the United States and international treaties. All authorized reproductions must be marked with this legend.

RESTRICTED RIGHTS LEGEND

TO THE EXTENT PERMITTED BY APPLICABLE LAW, CA PROVIDES THIS DOCUMENTATION "AS IS" WITHOUT WARRANTY OF ANY KIND, INCLUDING WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NONINFRINGEMENT. IN NO EVENT WILL CA BE LIABLE TO THE END USER OR ANY THIRD PARTY FOR ANY LOSS OR DAMAGE, DIRECT OR INDIRECT, FROM THE USE OF THIS DOCUMENTATION, INCLUDING WITHOUT LIMITATION, LOST PROFITS, BUSINESS INTERRUPTION, GOODWILL OR LOST DATA, EVEN IF CA IS EXPRESSLY ADVISED OF SUCH LOSS OR DAMAGE.

THE USE OF ANY PRODUCT REFERENCED IN THIS DOCUMENTATION AND THIS DOCUMENTATION IS GOVERNED BY THE END USER'S APPLICABLE LICENSE AGREEMENT.

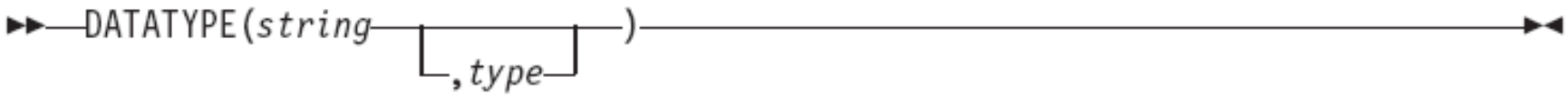
The manufacturer of this documentation is CA, Inc.

Provided with "Restricted Rights" as set forth in 48 C.F.R. Section 12.212, 48 C.F.R. Sections 52.227-19(c)(1) and (2) or DFARS Section 252.227.7013(c)(1)(ii) or applicable successor provisions.

What is a function?

- A pre-written subroutine.
- 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.

DATATYPE()



returns NUM if you specify only *string* and if *string* is a valid REXX number that can be added to 0 without error; returns CHAR if *string* is not a valid number.

If you specify *type*, returns 1 if *string* matches the type; otherwise returns 0. If *string* is null, the function returns 0 (except when *type* is X, which returns 1 for a null string). The following are valid *types*. (Only the capitalized and highlighted letter is needed; all characters following it are ignored. Note that for the hexadecimal option, you must start your string specifying the name of the option with x rather than h.)

Alphanumeric	returns 1 if *string* contains only characters from the ranges a–z, A–Z, and 0–9.
Binary	returns 1 if *string* contains only the characters 0 or 1 or both.
C	returns 1 if *string* is a mixed SBCS/DBCS string.

DATATYPE()

Dbcs	returns 1 if *string* is a DBCS-only string enclosed by SO and SI bytes.
Lowercase	returns 1 if *string* contains only characters from the range `a–z`.
Mixed case	returns 1 if *string* contains only characters from the ranges `a–z` and `A–Z`.
Number	returns 1 if *string* is a valid REXX number.
Symbol	returns 1 if *string* contains only characters that are valid in REXX symbols. (See page 10.) Note that both uppercase and lowercase alphabetics are permitted.
Uppercase	returns 1 if *string* contains only characters from the range `A–Z`.
Whole number	returns 1 if *string* is a REXX whole number under the current setting of NUMERIC DIGITS.
he**X**adecimal	returns 1 if *string* contains only characters from the ranges `a–f`, `A–F`, `0–9`, and blank (as long as blanks appear only between pairs of hexadecimal characters). Also returns 1 if *string* is a null string, which is a valid hexadecimal string.

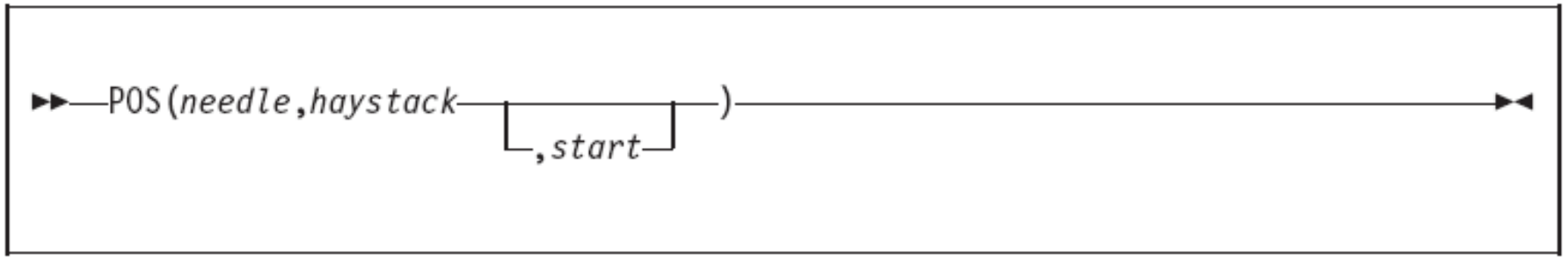
DATATYPE()

```
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 DATATYPE("1", B)
SAY
SAY DATATYPE("1")
SAY DATATYPE("A")
```

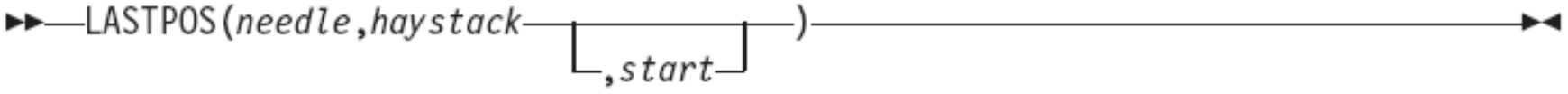
```
1
1
0
1
1
0
0
1
1
1
1
1
1

NUM
CHAR
***
```

POS(), LASTPOS()



returns the position of one string, *needle*, in another, *haystack*. (See also the INDEX and LASTPOS functions.) Returns `0` if *needle* is the null string or is not found or if *start* is greater than the length of *haystack*. By default the search starts at the first character of *haystack* (that is, the value of *start* is 1). You can override this by specifying *start* (which must be a positive whole number), the point at which the search starts.



**returns the position of the last occurrence of one string, *needle*, in another,
haystack.**

POS(), LASTPOS() examples

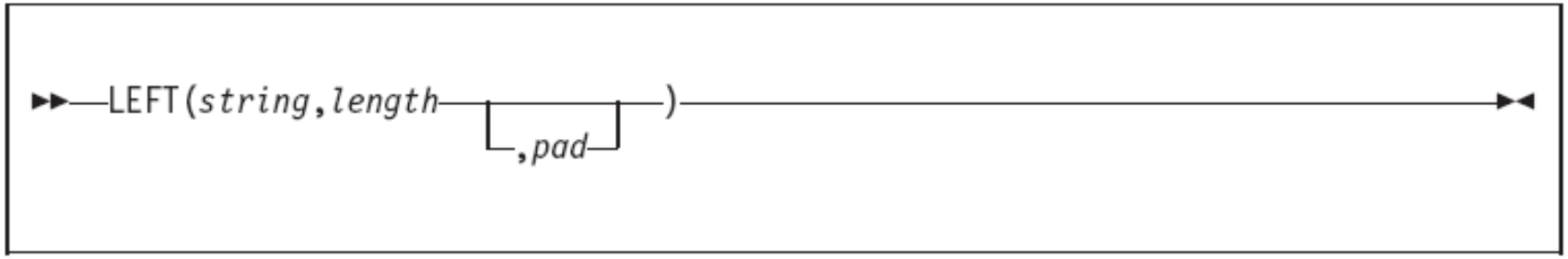
```
SAY POS(".", "CLCS.IULC00.REXX")
line = "/**************REXX****************/"
SAY POS("REXX", line)
```

```
5
15
***
```

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

```
12
25
***
```

LEFT(), RIGHT()



returns a string of length *length*, containing the leftmost *length* characters of *string*. The string returned is padded with *pad* characters (or truncated) on the right as needed. The default *pad* character is a blank. *length* must be a positive whole number or zero. The LEFT function is exactly equivalent to:

LEFT(), RIGHT() examples

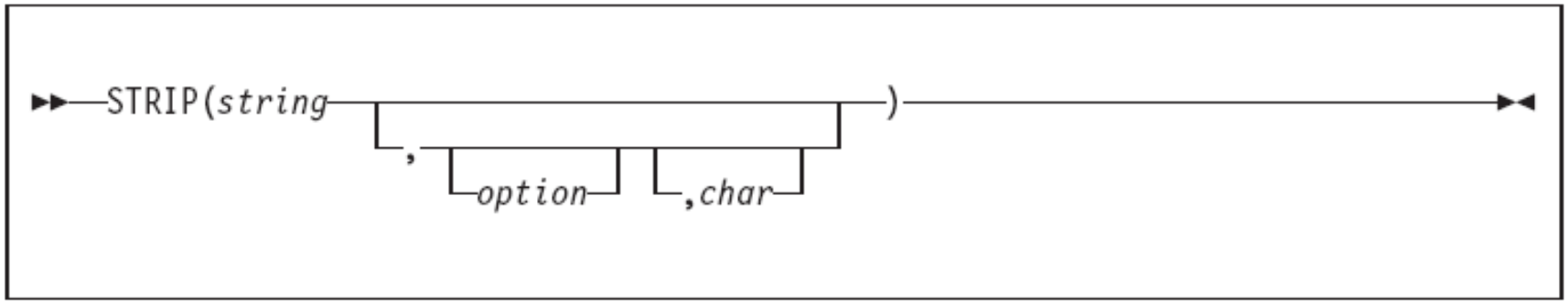
```
SAY LEFT("REXX", 2)
line = "IST510I TESTING ONLY"
SAY LEFT(line, 7)
```

```
RE
IST510I
***
```

```
SAY RIGHT("REXX", 2)
line = "IST510I TESTING ONLY"
SAY RIGHT(line, 7)
```

```
XX
NG ONLY
***
```

STRIP()

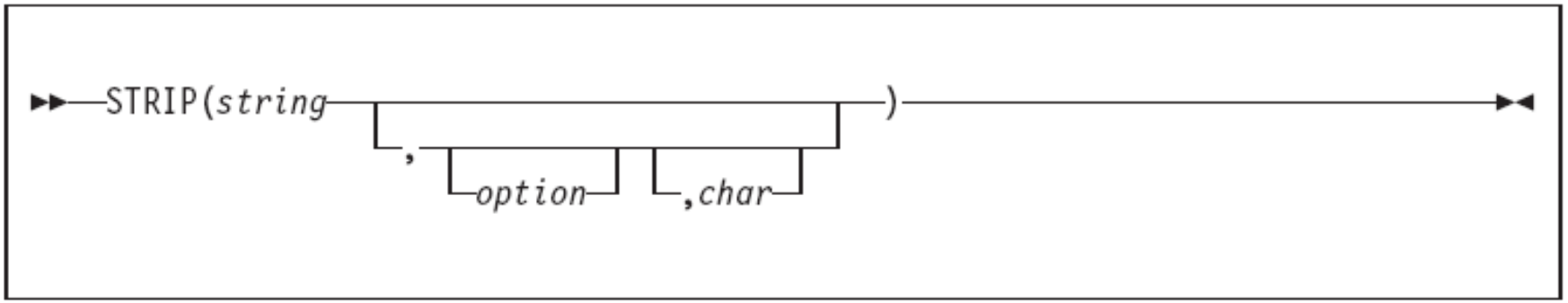


returns *string* with leading or trailing characters or both removed, based on the *option* you specify. The following are valid *option*s. (Only the capitalized and highlighted letter is needed; all characters following it are ignored.)

Both	removes both leading and trailing characters from *string*. This is the default.
Leading	removes leading characters from *string*.
Trailing	removes trailing characters from *string*.

The third argument, *char*, specifies the character to be removed, and the default is a blank. If you specify *char*, it must be exactly one character long.

STRIP()



returns *string* with leading or trailing characters or both removed, based on the *option* you specify. The following are valid *option*s. (Only the capitalized and highlighted letter is needed; all characters following it are ignored.)

Both	removes both leading and trailing characters from *string*. This is the default.
Leading	removes leading characters from *string*.
Trailing	removes trailing characters from *string*.

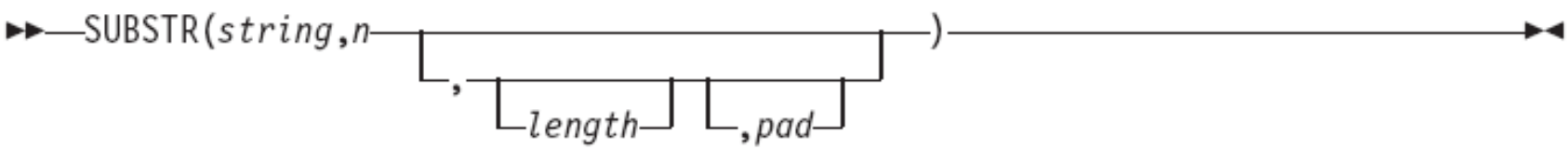
The third argument, *char*, specifies the character to be removed, and the default is a blank. If you specify *char*, it must be exactly one character long.

STRIP()

```
SAY STRIP("    REXX    ")
line = "0.120000000"
SAY STRIP(line, "T", 0)
```

```
REXX
0.12
***
```

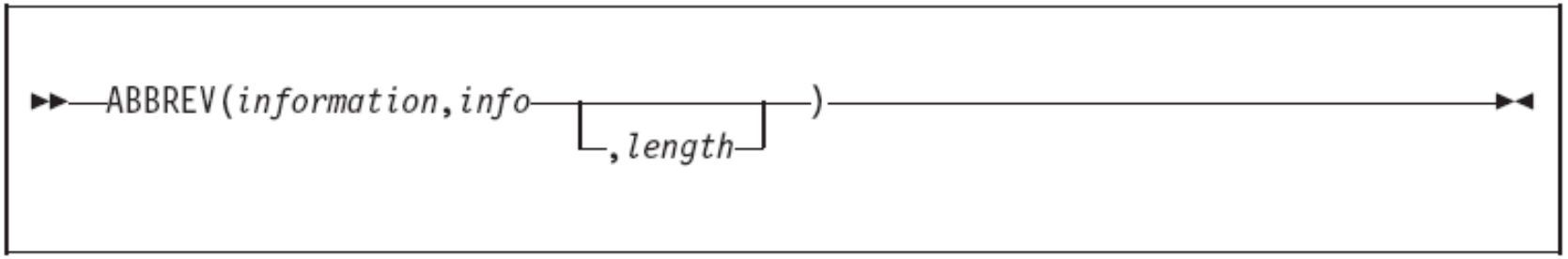
SUBSTR()




```
SAY SUBSTR("REXX PROGRAMMING", 4, 3)
line = "IST510I TESTING ONLY"
SAY SUBSTR(line, 7, 1)
```

```
X P
I
***
```

ABBREV( )



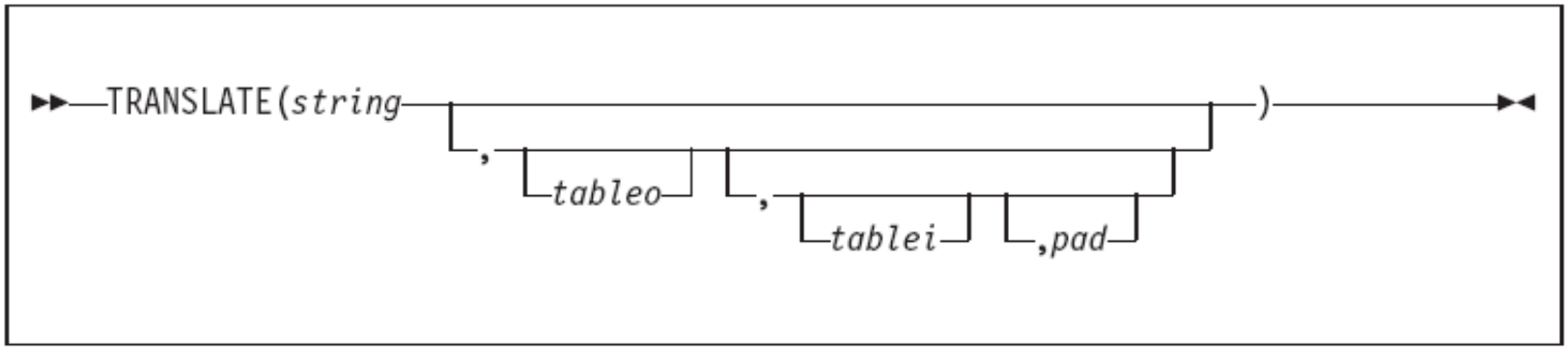
returns 1 if *info* is equal to the leading characters of *information* **and** the length of *info* is not less than *length*. Returns `0` if either of these conditions is not met.

If you specify *length*, it must be a positive whole number or zero. The default for *length* is the number of characters in *info*.

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

```
1
OK
***
```

TRANSLATE()



returns *string* with each character translated to another character or unchanged. You can also use this function to reorder the characters in *string*.

The output table is *tableo* and the input translation table is *tablei*. TRANSLATE searches *tablei* for each character in *string*. If the character is found, then the corresponding character in *tableo* is used in the result string; if there are duplicates in *tablei*, the first (leftmost) occurrence is used. If the character is not found, the original character in *string* is used. The result string is always the same length as *string*.

The tables can be of any length. If you specify neither translation table and omit *pad*, *string* is simply translated to uppercase (that is, lowercase `a`–`z` to uppercase `A`–`Z`), but, if you include *pad*, the language processor translates the entire string to *pad* characters. *tablei* defaults to `XRANGE('00'x,'FF'x)`, and *tableo* defaults to the null string and is padded with *pad* or truncated as necessary. The default *pad* is a blank.

TRANSLATE()

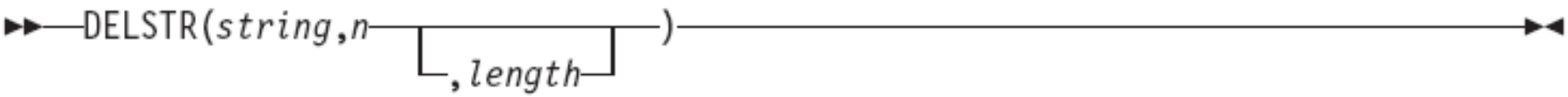
Here are some examples:

```
TRANSLATE('abcdef')                    ->    'ABCDEF'
TRANSLATE('abbc','&','b')              ->    'a&&c'
TRANSLATE('abcdef','12','ec')          ->    'ab2d1f'


TRANSLATE('abcdef','12','abcd','.')    ->    '12..ef'
TRANSLATE('APQRV',,'PR')               ->    'A Q V'
TRANSLATE('APQRV',XRANGE('00'X,'Q'))   ->    'APQ  '
TRANSLATE('4123','abcd','1234')        ->    'dabc'
```

The last example shows how to use the TRANSLATE function to reorder the characters in a string. In the example, the last character of any four-character string specified as the second argument would be moved to the beginning of the string.

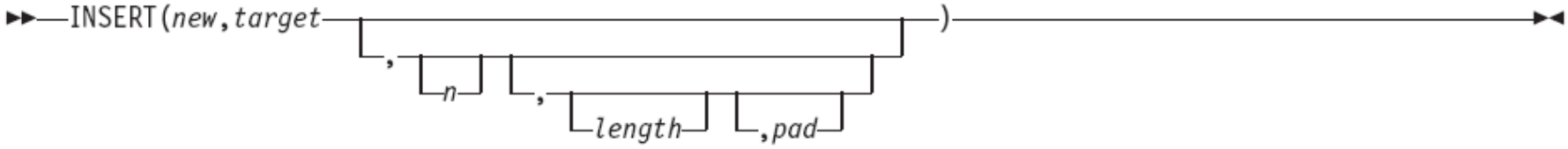
DELSTR()




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

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

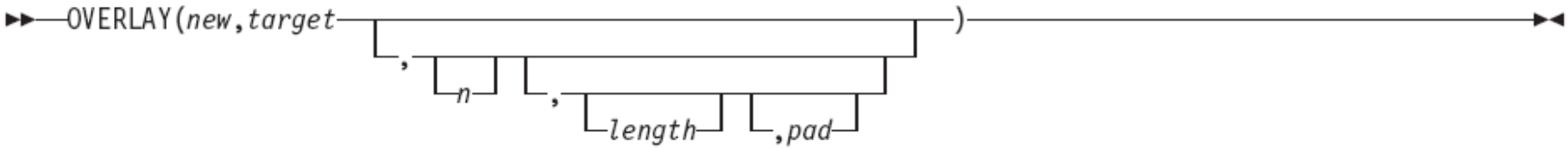
INSERT()




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

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

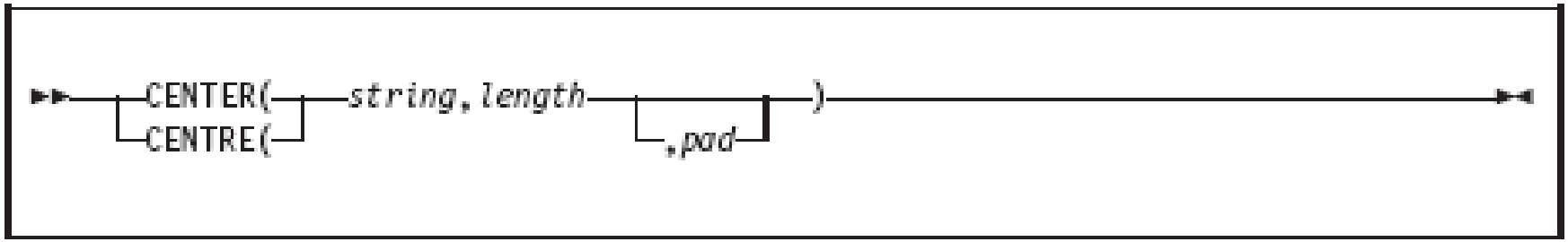
OVERLAY()




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

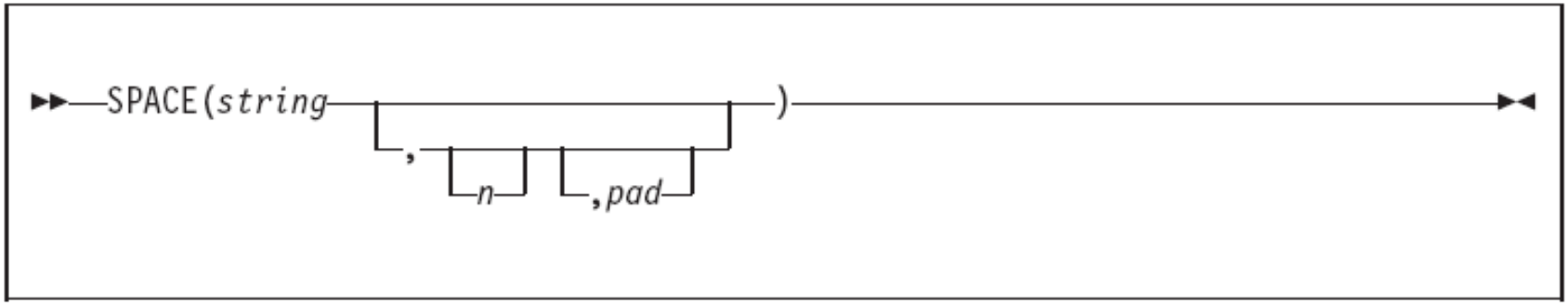
```
CLCS.IULC22.REXX
/*************REXX****************/
***
```

CENTRE()



returns a string of length *length* with *string* centered in it, with *pad* characters added as necessary to make up length. The *length* must be a positive whole number or zero. The default *pad* character is blank. If the string is longer than *length*, it is truncated at both ends to fit. If an odd number of characters are truncated or added, the right-hand end loses or gains one more character than the left-hand end.

SPACE()



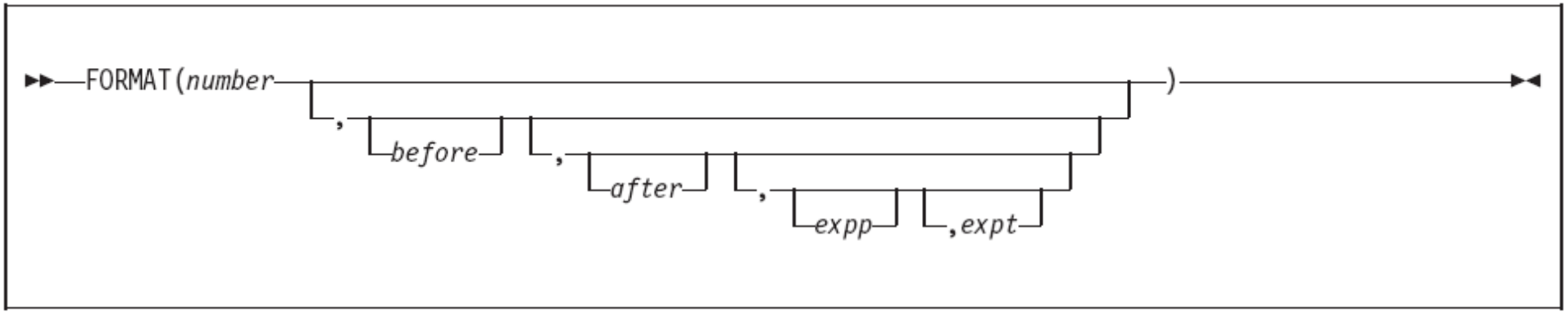
returns the blank-delimited words in *string* with *n* *pad* characters between each word. If you specify *n*, it must be a positive whole number or zero. If it is 0, all blanks are removed. Leading and trailing blanks are always removed. The default for *n* is 1, and the default *pad* character is a blank.

SPACE()

Here are some examples:

```
SPACE('abc  def  ')          ->    'abc def'
SPACE('  abc def',3)         ->    'abc   def'
SPACE('abc  def  ',1)        ->    'abc def'
SPACE('abc  def  ',0)        ->    'abcdef'
SPACE('abc  def  ',2,'+')    ->    'abc++def'
```

FORMAT()



returns *number*, rounded and formatted.

The *number* is first rounded according to standard REXX rules, just as though the operation `number+0` had been carried out. The result is precisely that of this operation if you specify only *number*. If you specify any other options, the *number* is formatted as follows.

The *before* and *after* options describe how many characters are used for the integer and decimal parts of the result, respectively. If you omit either or both of these, the number of characters used for that part is as needed.

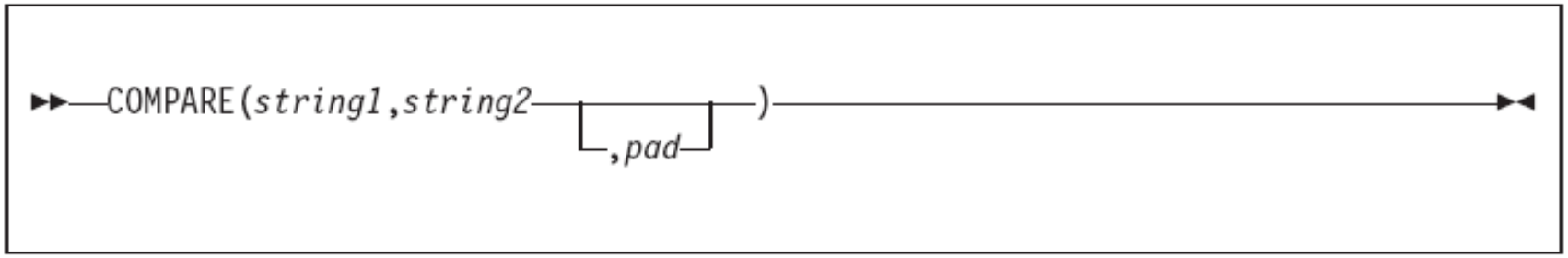
If *before* is not large enough to contain the integer part of the number (plus the sign for a negative number), an error results. If *before* is larger than needed for that part, the number is padded on the left with blanks. If *after* is not the same size as the decimal part of the number, the number is rounded (or extended with zeros) to fit. Specifying `0` causes the number to be rounded to an integer.

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
***
```

COMPARE(), COPIES()



returns 0 if the strings, *string1* and *string2*, are identical. Otherwise, returns the position of the first character that does not match. The shorter string is padded on the right with *pad* if necessary. The default *pad* character is a blank.

returns *n* concatenated copies of *string*. The *n* must be a positive whole number or zero.

Here are some examples:

```
COPIES('abc',3)     ->     'abcabcabc'
COPIES('abc',0)     ->     ''
```

LENGTH(), REVERSE()

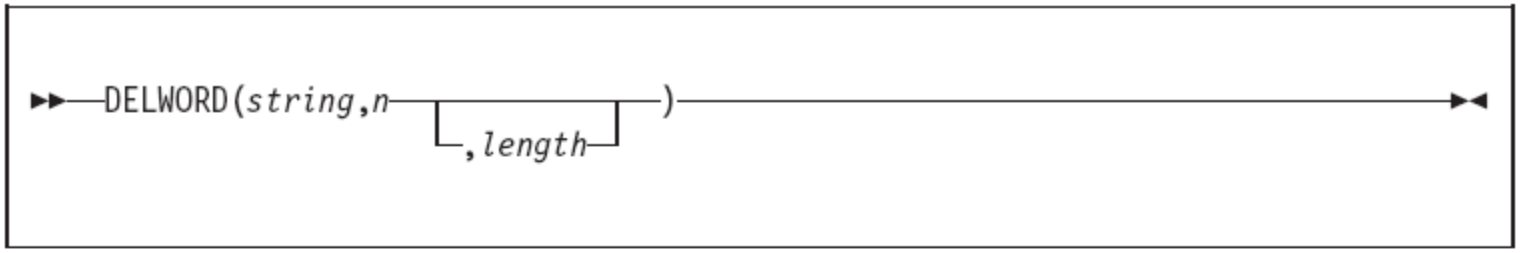
returns the length of *string*.

returns *string*, swapped end for end.

Here are some examples:

```
REVERSE('ABc.')    ->    '.cBA'
REVERSE('XYZ ')    ->    ' ZYX'
```

DELWORD()

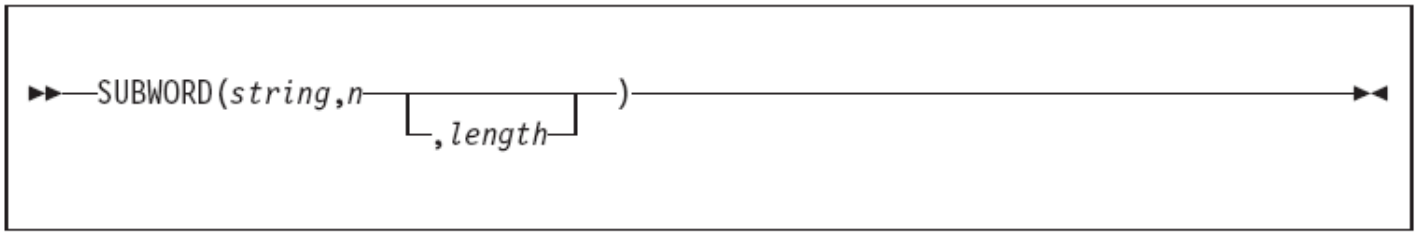


returns *string* after deleting the substring that starts at the *n*th word and is of *length* blank-delimited words. If you omit *length*, or if *length* is greater than the number of words from *n* to the end of *string*, the function deletes the remaining words in *string* (including the *n*th word). The *length* must be a positive whole number or zero. The *n* must be a positive whole number. If *n* is greater than the number of words in *string*, the function returns *string* unchanged. The string deleted includes any blanks following the final word involved but none of the blanks preceding the first word involved.

Here are some examples:

```
DELWORD('Now is the  time',2,2)  ->  'Now time'
DELWORD('Now is the time ',3)    ->  'Now is '
DELWORD('Now is the  time',5)    ->  'Now is the  time'
DELWORD('Now is   the time',3,1) ->  'Now is   time'
```

SUBWORD(), WORD()



returns the substring of *string* that starts at the *n*th word, and is up to *length* blank-delimited words. The *n* must be a positive whole number. If you omit *length*, it defaults to the number of remaining words in *string*. The returned string never has leading or trailing blanks, but includes all blanks between the selected words.

Here are some examples:

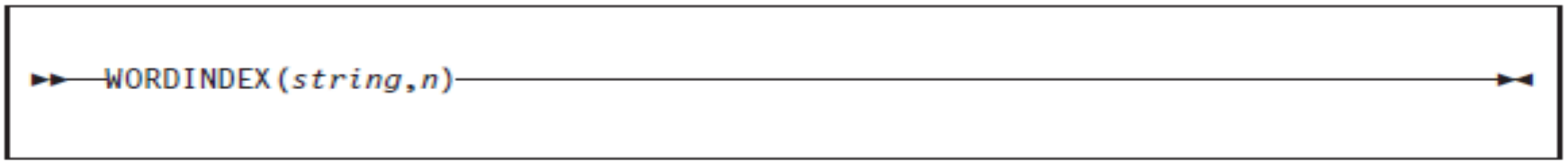
```
SUBWORD('Now is the  time',2,2)    ->    'is the'
SUBWORD('Now is the  time',3)      ->    'the  time'
SUBWORD('Now is the  time',5)      ->    ''
```

returns the *n*th blank-delimited word in *string* or returns the null string if fewer than *n* words are in *string*. The *n* must be a positive whole number. This function is exactly equivalent to SUBWORD(*string*,*n*,1).

Here are some examples:

```
WORD('Now is the time',3)    ->    'the'
WORD('Now is the time',5)    ->    ''
```

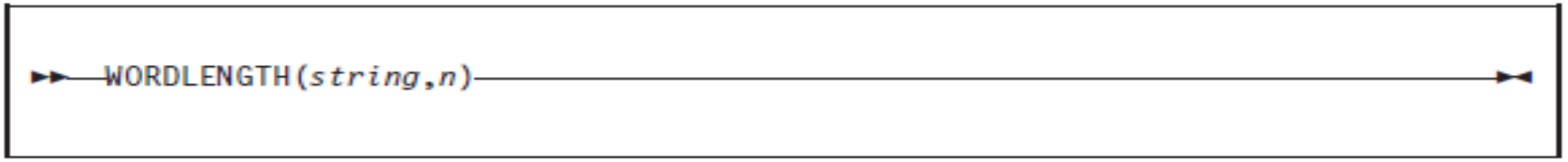
WORDINDEX(), WORDLENGTH()



returns the position of the first character in the *n*th blank-delimited word in *string* or returns 0 if fewer than *n* words are in *string*. The *n* must be a positive whole number.

Here are some examples:

```
WORDINDEX('Now is the time',3)     ->    8
WORDINDEX('Now is the time',6)     ->    0
```

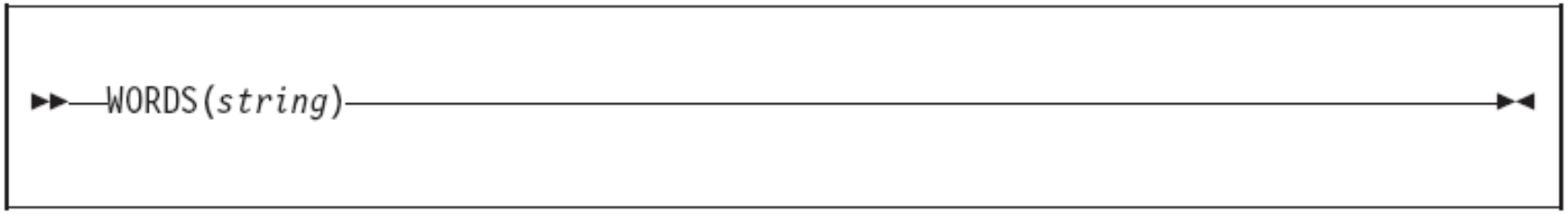


returns the length of the *n*th blank-delimited word in *string* or returns 0 if fewer than *n* words are in *string*. The *n* must be a positive whole number.

Here are some examples:

```
WORDLENGTH('Now is the time',2)        ->    2
WORDLENGTH('Now comes the time',2)     ->    5
WORDLENGTH('Now is the time',6)        ->    0
```

WORDS()



returns the number of blank-delimited words in *string*.

Here are some examples:

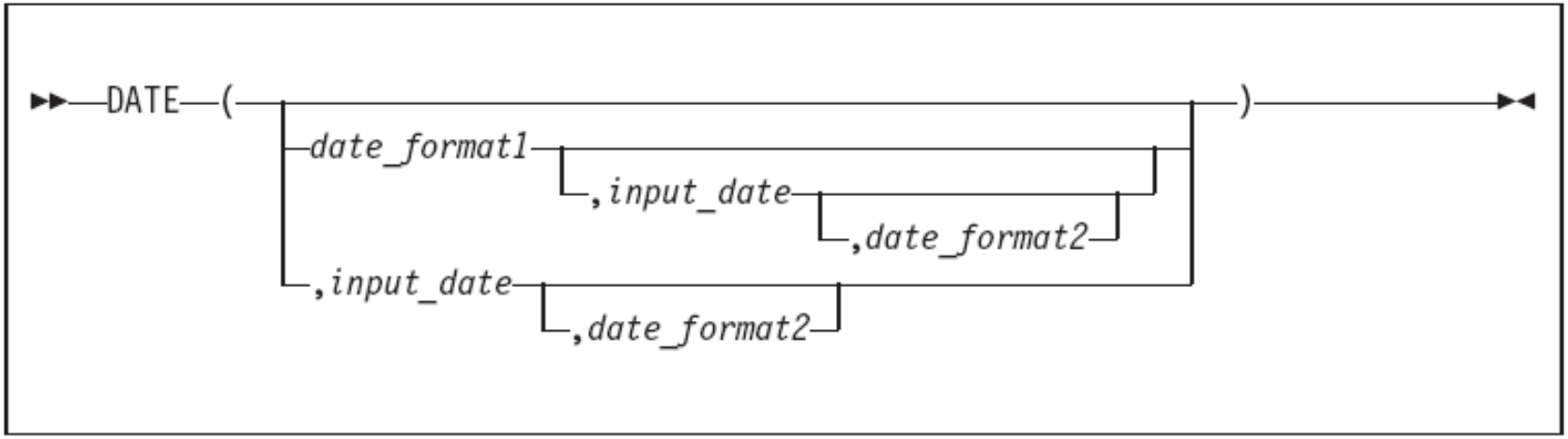
```
WORDS('Now is the time')    ->    4
WORDS(' ')                  ->    0
```

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
***
```

DATE()



returns, by default, the local date in the format: *dd mon yyyy* (day, month, year—for example, `25 Dec 2001`), with no leading zero or blank on the day. Otherwise, the string *input_date* is converted to the format specified by *date_format1*. *date_format2* can be specified to define the current format of *input_date*. The default for *date_format1* and *date_format2* is **N**ormal. *input_date* must not have a leading zero or blank.

DATE()

Base	the number of complete days (that is, not including the current day) since and including the base date, 1 January 0001, in the format: *dddddd* (no leading zeros or blanks). The expression `DATE('B')//7` returns a number in the range `0`–`6` that corresponds to the current day of the week, where `0` is Monday and `6` is Sunday. Thus, this function can be used to determine the day of the week independent of the national language in which you are working.
Century	the number of days, including the current day, since and including January 1 of the last year that is a multiple of 100 in the form: *ddddd* (no leading zeros). Example: A call to DATE(C) on March 13, 1992, returns 33675, the number of days from 1 January 1900 to 13 March 1992. Similarly, a call to DATE(C) on November 20, 2001, returns 690, the number of days from 1 January 2000 to 20 November 2001.
Days	the number of days, including the current day, so far in this year in the format: *ddd* (no leading zeros or blanks).
European	date in the format: *dd/mm/yy*
Julian	date in the format: *yyddd*.

DATE()

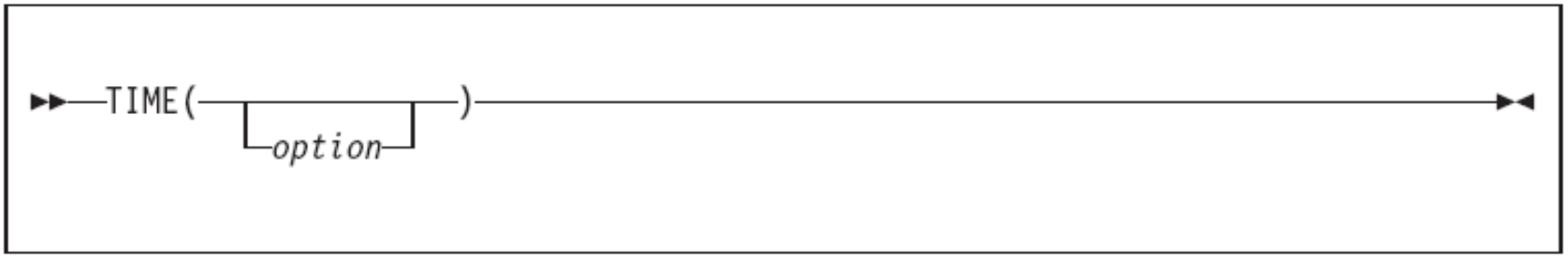
Month	full English name of the current month, in mixed case—for example, `August`. Only valid for *date_format1*.
Normal	date in the format: *dd mon yyyy*, in mixed case. **This is the default**. If the active language has an abbreviated form of the month name, then it is used—for example, `Jan`, `Feb`, and so on. If **N**ormal is specified (or allowed to default) for *date_format2*, the *input_date* must have the month (*mon*) specified in the English abbreviated form of the month name in mixed case.
Ordered	date in the format: *yy/mm/dd* (suitable for sorting, and so forth).
Standard	date in the format: *yyyymmdd* (suitable for sorting, and so forth).
Usa	date in the format: *mm/dd/yy*.
Weekday	the English name for the day of the week, in mixed case—for example, `Tuesday`. Only valid for *date_format1*.

DATE()

Here are some examples, assuming today is November 20, 2001:

```
DATE()                      ->    '20 Nov 2001'
DATE(,'20020609','S')       ->    '9 Jun 2002'
DATE('B')                   ->    '730808'
DATE('B','25 Sep 2001')     ->    '730752'
DATE('C')                   ->    '690'
DATE('E')                   ->    '20/11/01'
DATE('J')                   ->    '01324'
DATE('M')                   ->    'November'
DATE('N')                   ->    '20 Nov 2001'
DATE('N','1438','C')        ->    '8 Dec 2003'
DATE('O')                   ->    '01/11/20'
DATE('S')                   ->    '20011120'
DATE('U')                   ->    '11/20/01'
DATE('U','25 May 2001')     ->    '05/25/01'
DATE('U','25 MAY 2001')     ->    ERROR,month not in mixed case
DATE('W')                   ->    'Tuesday'
```

TIME()



returns the local time in the 24-hour clock format: hh:mm:ss (hours, minutes, and seconds) by default, for example, `04:41:37`.

You can use the following *option*s to obtain alternative formats, or to gain access to the elapsed-time clock. (Only the capitalized and highlighted letter is needed; all characters following it are ignored.)

Civil — returns the time in Civil format: hh:mmxx. The hours may take the values `1` through `12`, and the minutes the values `00` through `59`. The minutes are followed immediately by the letters `am` or `pm`. This distinguishes times in the morning (12 midnight through 11:59 a.m.—appearing as `12:00am` through `11:59am`) from noon and afternoon (12 noon through 11:59 p.m.—appearing as `12:00pm` through `11:59pm`). The hour has no leading zero. The minute field shows the current minute (rather than the nearest minute) for consistency with other TIME results.

TIME()

Elapsed	returns ssssssss.uuuuuu, the number of seconds.microseconds since the elapsed-time clock (described later) was started or reset. The number has no leading zeros or blanks, and the setting of NUMERIC DIGITS does not affect the number. The fractional part always has six digits.
Hours	returns up to two characters giving the number of hours since midnight in the format: hh (no leading zeros or blanks, except for a result of `0`).
Long	returns time in the format: hh:mm:ss.uuuuuu (uuuuuu is the fraction of seconds, in microseconds). The first eight characters of the result follow the same rules as for the Normal form, and the fractional part is always six digits.
Minutes	returns up to four characters giving the number of minutes since midnight in the format: mmmm (no leading zeros or blanks, except for a result of `0`).
Normal	returns the time in the default format hh:mm:ss, as described previously. The hours can have the values `00` through `23`, and minutes and seconds, `00` through `59`. All these are always two digits. Any fractions of seconds are ignored (times are never rounded up). **This is the default**.

TIME()

Reset — returns ssssssss.uuuuuu, the number of seconds.microseconds since the elapsed-time clock (described later) was started or reset and also resets the elapsed-time clock to zero. The number has no leading zeros or blanks, and the setting of NUMERIC DIGITS does not affect the number. The fractional part always has six digits.

Seconds — returns up to five characters giving the number of seconds since midnight in the format: sssss (no leading zeros or blanks, except for a result of 0).

Here are some examples, assuming that the time is 4:54 p.m.:

```
TIME()       ->   '16:54:22'
TIME('C')    ->   '4:54pm'
TIME('H')    ->   '16'
TIME('L')    ->   '16:54:22.123456'   /* Perhaps */
TIME('M')    ->   '1014'            /* 54 + 60*16 */
TIME('N')    ->   '16:54:22'
TIME('S')    ->   '60862'  /* 22 + 60*(54+60*16) */
```

TSO/E External functions

In addition to the built-in functions, TSO/E provides external functions that you can use to do specific tasks:

- GETMSG - returns in variables a system message issued during an extended console session. It also returns in variables associated information about the message.

- LISTDSI - returns in variables the data set attributes of a specified data set.

- MSG - controls the display of TSO/E messages. The function returns the previous setting of MSG (ON/OFF).

- MVSVAR - uses specific argument values to return information about MVS, TSO/E, and the current session.

- OUTTRAP - traps lines of TSO/E command output into a specified series of variables. The function call returns the variable name specified.

TSO/E External functions

- PROMPT - sets the prompt option ON/OFF for TSO/E interactive commands. The function returns the previous setting of prompt.

- SETLANG - retrieves and optionally changes the language in which REXX messages are displayed. The function returns the previous language setting.

- STORAGE - retrieves and optionally changes the value in a storage address. Carefully!

- SYSCPUS - returns in a stem variable information about all CPUs that are on-line.

- SYSDSN - returns OK if the specified data set exists; otherwise, it returns an appropriate error message.

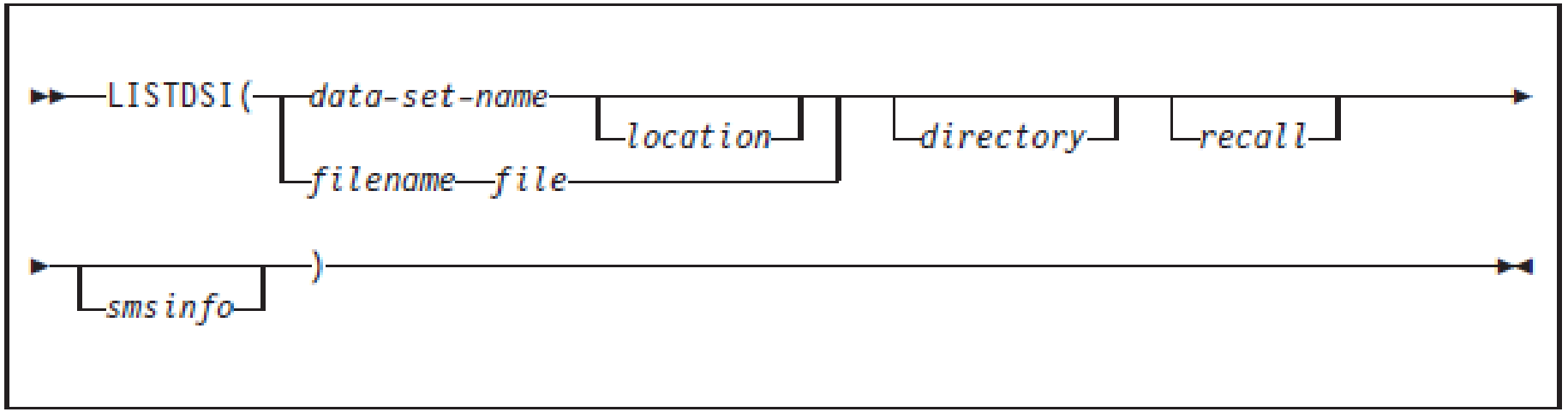
- SYSVAR - uses specific argument values to return information about the user, terminal, language, exec, system, and console session.

TSO/E External functions

LISTDSI

You can use the LISTDSI (List Dataset Information) function to retrieve detailed information about a data set's attributes.

LISTDSI does not support tape datasets. LISTDSI supports GDG data sets when using absolute generation names, but does not support relative GDG names. LISTDSI does not support HFS data sets.



TSO/E External functions

MVSVAR

MVSVAR returns information about MVS, TSO/E, and the current session, such as the symbolic name of the MVS system, or the security label of the TSO/E session.

The MVSVAR function is available **in any MVS address space.**

TSO/E External functions

SYSCPUS

SYSCPUS places, in a stem variable, information about those CPUs that are on-line.

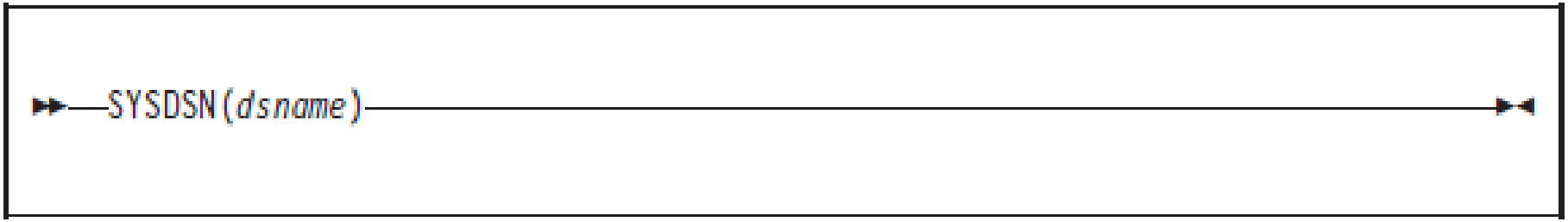
The SYSCPUS function runs **in any MVS address space.**

TSO/E External functions

SYSDSN

SYSDSN returns whether the specified data set exists and is available for use. The **dsname** can be the name of any cataloged data set or cataloged PDS with a member name. Additionally, if you specify a member of a PDS, SYSDSN checks to see if you have access to the data set.

SYSDSN does not support tape datasets. SYSDSN supports generation data group (GDG) data sets when using absolute generation names, but does not support relative GDG names.



TSO/E External functions

SYSVAR

The SYSVAR function retrieves information about MVS, TSO/E, and the current session, such as levels of software available, your logon procedure, and your user ID.

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
***
```

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

Work section 7.2 (output)

```
 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) Built-in functions

- Built-in Function overview,
- Non SSA Built-in Functions
- TSO External Functions.

.

Resources: TSO/E REXX Reference

TSO/E REXX User's Guide

This course has been prepared by Milos Forman for MCoE needs only!

There is only subset of functions. See the resources for the others.

PROPRIETARY AND CONFIDENTIAL INFORMATION

These education materials and related computer software program (hereinafter referred to as the "Education Materials") is for the end user's informational purposes only and is subject to change or withdrawal by CA, Inc. at any time.
These Education Materials may not be copied, transferred, reproduced, disclosed or distributed, in whole or in part, without the prior written consent of CA. These Education Materials are proprietary information and a trade secret of CA. Title to these Education Materials remains with CA, and these Education Materials are protected by the copyright laws of the United States and international treaties. All authorized reproductions must be marked with this legend.

RESTRICTED RIGHTS LEGEND

TO THE EXTENT PERMITTED BY APPLICABLE LAW, CA PROVIDES THIS DOCUMENTATION "AS IS" WITHOUT WARRANTY OF ANY KIND, INCLUDING WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NONINFRINGEMENT. IN NO EVENT WILL CA BE LIABLE TO THE END USER OR ANY THIRD PARTY FOR ANY LOSS OR DAMAGE, DIRECT OR INDIRECT, FROM THE USE OF THIS DOCUMENTATION, INCLUDING WITHOUT LIMITATION, LOST PROFITS, BUSINESS INTERRUPTION, GOODWILL OR LOST DATA, EVEN IF CA IS EXPRESSLY ADVISED OF SUCH LOSS OR DAMAGE.
THE USE OF ANY PRODUCT REFERENCED IN THIS DOCUMENTATION AND THIS DOCUMENTATION IS GOVERNED BY THE END USER'S APPLICABLE LICENSE AGREEMENT.
The manufacturer of this documentation is CA, Inc.
Provided with "Restricted Rights" as set forth in 48 C.F.R. Section 12.212, 48 C.F.R. Sections 52.227-19(c)(1) and (2) or DFARS Section 252.227.7013(c)(1)(ii) or applicable successor provisions.

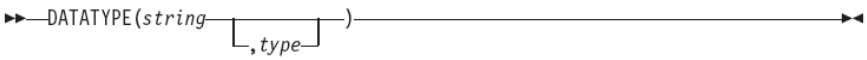
What is a function?

- A pre-written subroutine.
- 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.

This course has been prepared by Milos Forman for MCoE needs only!

Let us look to some of them.

DATATYPE()



returns NUM if you specify only *string* and if *string* is a valid REXX number that can be added to 0 without error; returns CHAR if *string* is not a valid number.

If you specify *type*, returns 1 if *string* matches the type; otherwise returns 0. If *string* is null, the function returns 0 (except when *type* is X, which returns 1 for a null string). The following are valid *types*. (Only the capitalized and highlighted letter is needed; all characters following it are ignored. Note that for the hexadecimal option, you must start your string specifying the name of the option with x rather than h.)

Alphanumeric	returns 1 if *string* contains only characters from the ranges a–z, A–Z, and 0–9.
Binary	returns 1 if *string* contains only the characters 0 or 1 or both.
C	returns 1 if *string* is a mixed SBCS/DBCS string.

4

datatype() - tests the meaning or type of characters in a string:

say datatype(“AA”,A)

Where AA is the string and A is the type. In this case it returns 1, because the string matches the type (alphanumeric).

DATATYPE()

Dbcs	returns 1 if *string* is a DBCS-only string enclosed by SO and SI bytes.
Lowercase	returns 1 if *string* contains only characters from the range `a–z`.
Mixed case	returns 1 if *string* contains only characters from the ranges `a–z` and `A–Z`.
Number	returns 1 if *string* is a valid REXX number.
Symbol	returns 1 if *string* contains only characters that are valid in REXX symbols. (See page 10.) Note that both uppercase and lowercase alphabetics are permitted.
Uppercase	returns 1 if *string* contains only characters from the range `A–Z`.
Whole number	returns 1 if *string* is a REXX whole number under the current setting of NUMERIC DIGITS.
he**X**adecimal	returns 1 if *string* contains only characters from the ranges `a–f`, `A–F`, `0–9`, and blank (as long as blanks appear only between pairs of hexadecimal characters). Also returns 1 if *string* is a null string, which is a valid hexadecimal string.

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")
```

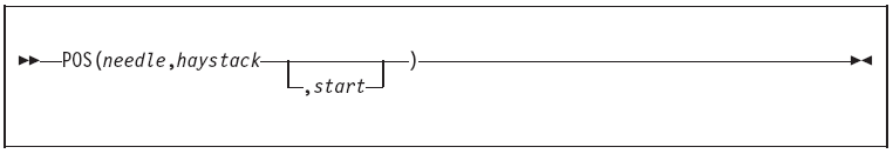
```
1
1
0
1
1
0
0
1
1
1
1
1
1

NUM
CHAR
***
```

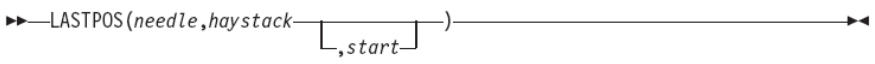
6

See ‘MCOE.REXA.REXX(BUILTFUN)’

POS(), LASTPOS()



returns the position of one string, *needle*, in another, *haystack*. (See also the INDEX and LASTPOS functions.) Returns `0` if *needle* is the null string or is not found or if *start* is greater than the length of *haystack*. By default the search starts at the first character of *haystack* (that is, the value of *start* is `1`). You can override this by specifying *start* (which must be a positive whole number), the point at which the search starts.



returns the position of the last occurrence of one string, *needle*, in another, *haystack*.

7

pos() - returns the position of one string, in another.

lastpos() - returns the position of the last occurrence of one string in another.

POS(), LASTPOS() examples

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

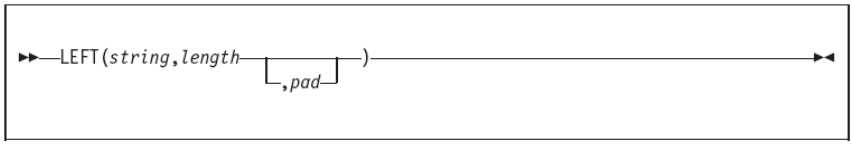
```
5
15
***
```

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

```
12
25
***
```


See ‘MCOE.REXA.REXX(BUILTFUN)’

LEFT(), RIGHT()



returns a string of length *length*, containing the leftmost *length* characters of *string*. The string returned is padded with *pad* characters (or truncated) on the right as needed. The default *pad* character is a blank. *length* must be a positive whole number or zero. The LEFT function is exactly equivalent to:

9

left() - returns a string of length, containing the leftmost length.

right() - returns a string of length, containing the rightmost length.

LEFT(), RIGHT() examples

```
SAY LEFT("REXX", 2)
line = "IST510I TESTING ONLY"
SAY LEFT(line, 7)
```

```
RE
IST510I
***
```

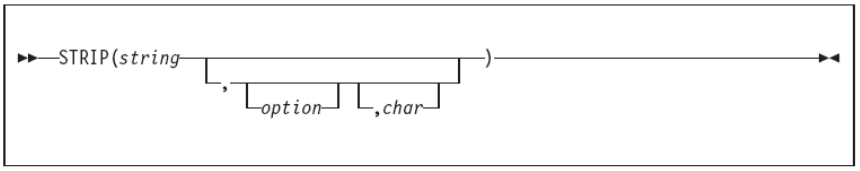
```
SAY RIGHT("REXX", 2)
line = "IST510I TESTING ONLY"
SAY RIGHT(line, 7)
```

```
XX
NG ONLY
***
```

10

See ‘MCOE.REXA.REXX(BUILTFUN)’

STRIP()



returns *string* with leading or trailing characters or both removed, based on the *option* you specify. The following are valid *option*s. (Only the capitalized and highlighted letter is needed; all characters following it are ignored.)

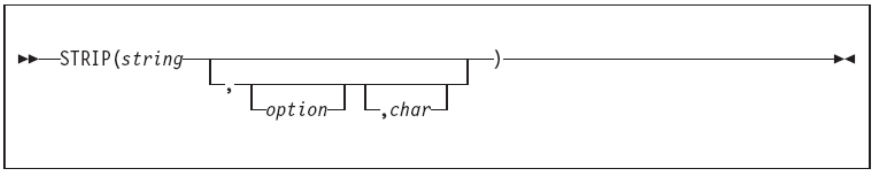
Both	removes both leading and trailing characters from *string*. This is the default.
Leading	removes leading characters from *string*.
Trailing	removes trailing characters from *string*.

The third argument, *char*, specifies the character to be removed, and the default is a blank. If you specify *char*, it must be exactly one character long.

11

strip() – returns string with leading or trailing characters or both removed.

STRIP()



returns *string* with leading or trailing characters or both removed, based on the *option* you specify. The following are valid *option*s. (Only the capitalized and highlighted letter is needed; all characters following it are ignored.)

Both	removes both leading and trailing characters from *string*. This is the default.
Leading	removes leading characters from *string*.
Trailing	removes trailing characters from *string*.

The third argument, *char*, specifies the character to be removed, and the default is a blank. If you specify *char*, it must be exactly one character long.

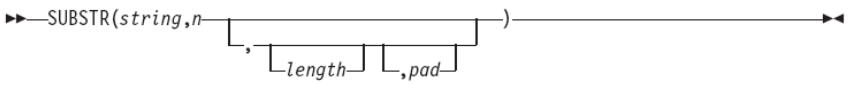
STRIP()

```
SAY STRIP("    REXX    ")
line = "0.120000000"
SAY STRIP(line, "T", 0)
```

```
REXX
0.12
***
```


See ‘MCOE.REXA.REXX(BUILTFUN)’

SUBSTR()




```
SAY SUBSTR("REXX PROGRAMMING", 4, 3)
line = "IST510I TESTING ONLY"
SAY SUBSTR(line, 7, 1)
```

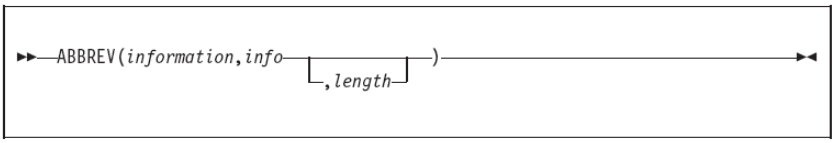
```
X P
I
***
```

14

substr() - returns the substring of string that begins at the nth character and is of length.

See ‘MCOE.REXA.REXX(BUILTFUN)’

ABBREV()



returns 1 if *info* is equal to the leading characters of *information* **and** the length of *info* is not less than *length*. Returns 0 if either of these conditions is not met.

If you specify *length*, it must be a positive whole number or zero. The default for *length* is the number of characters in *info*.

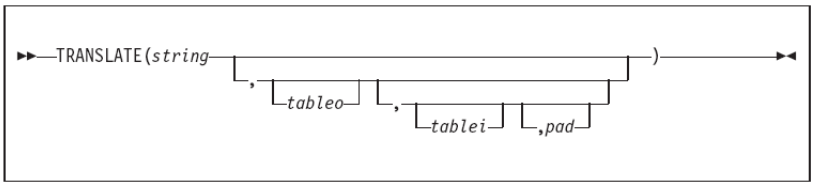
```
SAY ABBREV("REXX PROGRAMMING", "REXX P")
line = "CON"
IF ABBREV("CONFIRM", line, 1) = 1 THEN DO
   SAY "OK"
END
```

```
1
OK
***
```

abbrev() – returns 1 if info is equal to the leading characters of information, and the length of prefix is not less than length.

See ‘MCOE.REXA.REXX(BUILTFUN)’

TRANSLATE()



returns *string* with each character translated to another character or unchanged. You can also use this function to reorder the characters in *string*.

The output table is *tableo* and the input translation table is *tablei*. TRANSLATE searches *tablei* for each character in *string*. If the character is found, then the corresponding character in *tableo* is used in the result string; if there are duplicates in *tablei*, the first (leftmost) occurrence is used. If the character is not found, the original character in *string* is used. The result string is always the same length as *string*.

The tables can be of any length. If you specify neither translation table and omit *pad*, *string* is simply translated to uppercase (that is, lowercase `a–z` to uppercase `A–Z`), but, if you include *pad*, the language processor translates the entire string to *pad* characters. *tablei* defaults to `XRANGE('00'x,'FF'x)`, and *tableo* defaults to the null string and is padded with *pad* or truncated as necessary. The default *pad* is a blank.

translate() – returns string with each character translated to another character.

TRANSLATE()

Here are some examples:

```
TRANSLATE('abcdef')                    ->    'ABCDEF'
TRANSLATE('abbc','&','b')              ->    'a&&c'
TRANSLATE('abcdef','12','ec')          ->    'ab2d1f'


TRANSLATE('abcdef','12','abcd','.')    ->    '12..ef'
TRANSLATE('APQRV',,'PR')               ->    'A Q V'
TRANSLATE('APQRV',XRANGE('00'X,'Q'))   ->    'APQ  '
TRANSLATE('4123','abcd','1234')        ->    'dabc'
```

The last example shows how to use the TRANSLATE function to reorder the characters in a string. In the example, the last character of any four-character string specified as the second argument would be moved to the beginning of the string.

17

See 'MCOE.REXA.REXX(BUILTFUN)'

DELSTR()

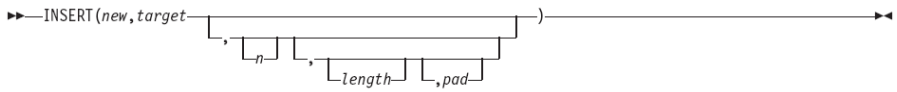

```
SAY DELSTR("CLCS.IULC00.REXX", 6, 6)
line = "/*************REXX***************/"
SAY DELSTR(line, 15, 4)
```

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


delstr() – returns string after deleting the substring that begins at the nth character and is of length characters.

See ‘MCOE.REXA.REXX(BUILTFUN)’

INSERT()



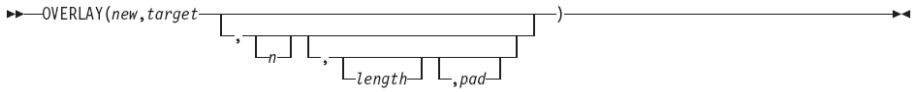

```
SAY INSERT("IULC00", "CLCS..REXX", 5)
line = "/*****************************/"
SAY INSERT("REXX", line, 15)
```

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


insert() – inserts the string new, padded or truncated to length length, into the string target after the nth character.

See ‘MCOE.REXA.REXX(BUILTFUN)’

OVERLAY()




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

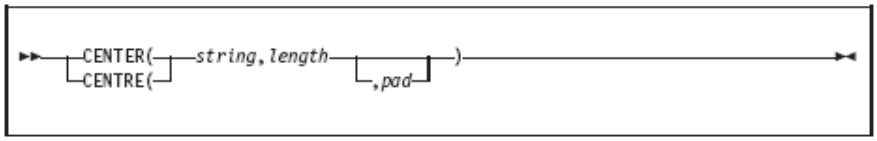
```
CLCS.IULC22.REXX
/*************REXX***************/
***
```

20

overlay() – returns the string target, which, starting at the nth character, is overlaid with the string new, padded or truncated to length length.

See ‘MCOE.REXA.REXX(BUILTFUN)’

CENTRE()



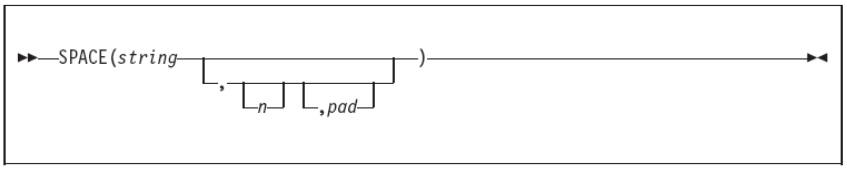
returns a string of length *length* with *string* centered in it, with *pad* characters added as necessary to make up length. The *length* must be a positive whole number or zero. The default *pad* character is blank. If the string is longer than *length*, it is truncated at both ends to fit. If an odd number of characters are truncated or added, the right-hand end loses or gains one more character than the left-hand end.

21

center() – returns a string of length length with string centered in it, with pad characters added as necessary to make up length.

See ‘MCOE.REXA.REXX(BUILTFUN)’

SPACE()



returns the blank-delimited words in *string* with *n* *pad* characters between each word. If you specify *n*, it must be a positive whole number or zero. If it is 0, all blanks are removed. Leading and trailing blanks are always removed. The default for *n* is 1, and the default *pad* character is a blank.

22

space() – returns the blank-delimited words in string with n pad characters between each word.

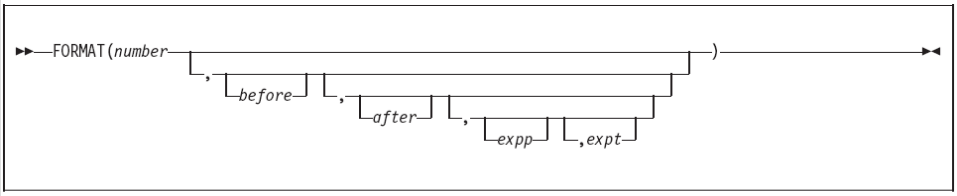
SPACE()

Here are some examples:

```
SPACE('abc  def  ')          ->    'abc def'
SPACE('  abc def',3)         ->    'abc   def'
SPACE('abc  def  ',1)        ->    'abc def'
SPACE('abc  def  ',0)        ->    'abcdef'
SPACE('abc  def  ',2,'+')    ->    'abc++def'
```


See ‘MCOE.REXA.REXX(BUILTFUN)’

FORMAT()



returns *number*, rounded and formatted.

The *number* is first rounded according to standard REXX rules, just as though the operation `number+0` had been carried out. The result is precisely that of this operation if you specify only *number*. If you specify any other options, the *number* is formatted as follows.

The *before* and *after* options describe how many characters are used for the integer and decimal parts of the result, respectively. If you omit either or both of these, the number of characters used for that part is as needed.

If *before* is not large enough to contain the integer part of the number (plus the sign for a negative number), an error results. If *before* is larger than needed for that part, the number is padded on the left with blanks. If *after* is not the same size as the decimal part of the number, the number is rounded (or extended with zeros) to fit. Specifying `0` causes the number to be rounded to an integer.

24

format() – returns number, rounded and formatted.

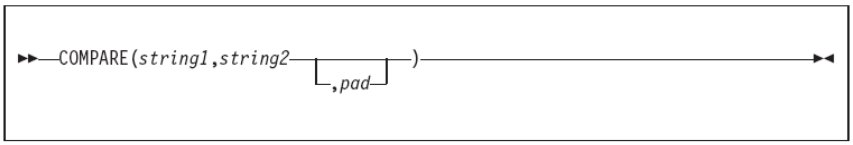
FORMAT()

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

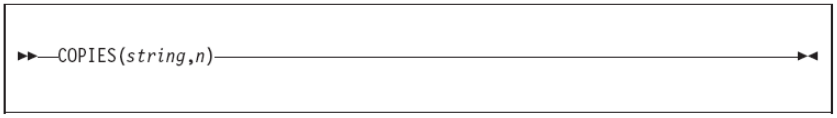
```
     12000
         3.5
       124.57
1.759173E+04
***
```


See ‘MCOE.REXA.REXX(BUILTFUN)’

COMPARE(), COPIES()



returns 0 if the strings, *string1* and *string2*, are identical. Otherwise, returns the position of the first character that does not match. The shorter string is padded on the right with *pad* if necessary. The default *pad* character is a blank.



returns *n* concatenated copies of *string*. The *n* must be a positive whole number or zero.

Here are some examples:

```
COPIES('abc',3)     ->     'abcabcabc'
COPIES('abc',0)     ->     ''
```

26

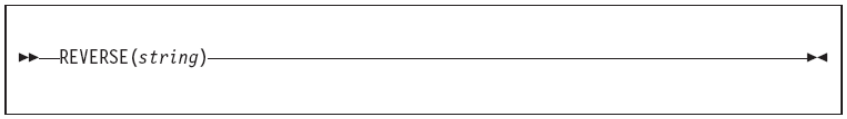
compare() – returns 0 if the strings, string 1 and string 2, are identical. Otherwise, returns the position of the first character that does not match.

copies() – returns n concatenated copies of string.

See ‘MCOE.REXA.REXX(BUILTFUN)’

LENGTH(), REVERSE()

returns the length of *string*.



returns *string*, swapped end for end.

Here are some examples:

```
REVERSE('ABc.')    ->    '.cBA'
REVERSE('XYZ ')    ->    ' ZYX'
```

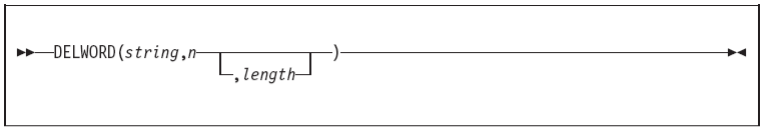
27

length() – returns the length of string.

reverse() – returns string, swapped end for end.

See ‘MCOE.REXA.REXX(BUILTFUN)’

DELWORD()



returns *string* after deleting the substring that starts at the *n*th word and is of *length* blank-delimited words. If you omit *length*, or if *length* is greater than the number of words from *n* to the end of *string*, the function deletes the remaining words in *string* (including the *n*th word). The *length* must be a positive whole number or zero. The *n* must be a positive whole number. If *n* is greater than the number of words in *string*, the function returns *string* unchanged. The string deleted includes any blanks following the final word involved but none of the blanks preceding the first word involved.

Here are some examples:

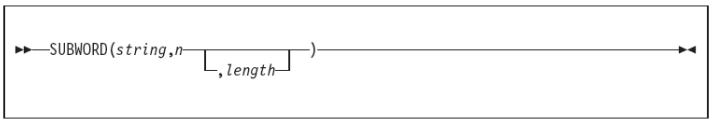
```
DELWORD('Now is the  time',2,2)  ->  'Now time'
DELWORD('Now is the time ',3)    ->  'Now is '
DELWORD('Now is the  time',5)    ->  'Now is the  time'
DELWORD('Now is   the time',3,1) ->  'Now is   time'
```

28

delword() – returns string after deleting the substring that starts at the nth word and is of length blank-delimited words.

See ‘MCOE.REXA.REXX(BUILTFUN)’

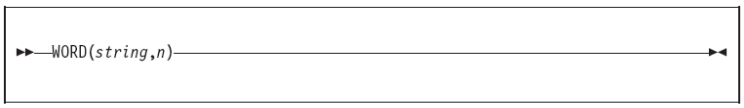
SUBWORD(), WORD()



returns the substring of *string* that starts at the *n*th word, and is up to *length* blank-delimited words. The *n* must be a positive whole number. If you omit *length*, it defaults to the number of remaining words in *string*. The returned string never has leading or trailing blanks, but includes all blanks between the selected words.

Here are some examples:

```
SUBWORD('Now is the  time',2,2)    ->    'is the'
SUBWORD('Now is the  time',3)      ->    'the  time'
SUBWORD('Now is the  time',5)      ->    ''
```



returns the *n*th blank-delimited word in *string* or returns the null string if fewer than *n* words are in *string*. The *n* must be a positive whole number. This function is exactly equivalent to SUBWORD(*string*,*n*,1).

Here are some examples:

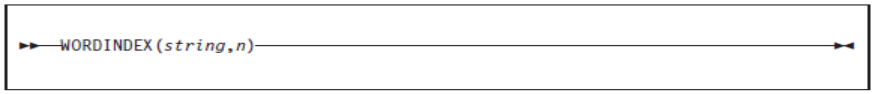
```
WORD('Now is the time',3)    ->    'the'
WORD('Now is the time',5)    ->    ''
```

subword() – returns the substring that starts at the nth word, and is up to length blank-delimited words.

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

See ‘MCOE.REXA.REXX(BUILTFUN)’

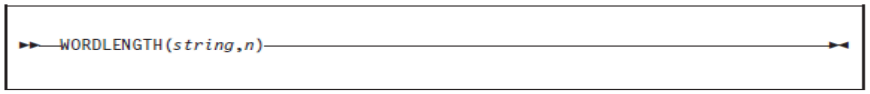
WORDINDEX(), WORDLENGTH()



returns the position of the first character in the *n*th blank-delimited word in *string* or returns 0 if fewer than *n* words are in *string*. The *n* must be a positive whole number.

Here are some examples:

```
WORDINDEX('Now is the time',3)    ->    8
WORDINDEX('Now is the time',6)    ->    0
```



returns the length of the *n*th blank-delimited word in *string* or returns 0 if fewer than *n* words are in *string*. The *n* must be a positive whole number.

Here are some examples:

```
WORDLENGTH('Now is the time',2)       ->    2
WORDLENGTH('Now comes the time',2)    ->    5
WORDLENGTH('Now is the time',6)       ->    0
```

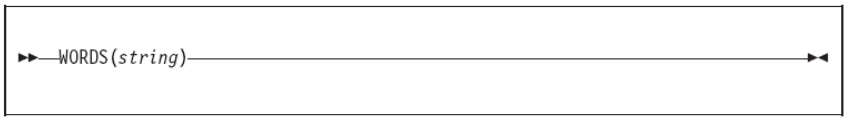
30

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

See ‘MCOE.REXA.REXX(BUILTFUN)’

WORDS()



returns the number of blank-delimited words in *string*.

Here are some examples:

```
WORDS('Now is the time')    ->    4
WORDS(' ')                  ->    0
```

31

words() – returns the number of blank-delimited words in string.

See ‘MCOE.REXA.REXX(BUILTFUN)’

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
***
```

32

abs(number) – returns absolut value of number.

min(number) – returns smallest number from the list.

max(number) – returns largest number from the list.

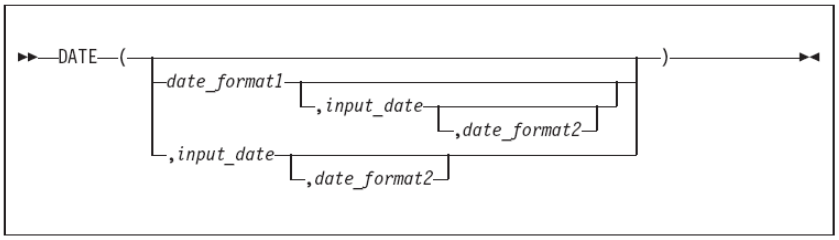
random(number) – returns quasi random number from the range.

sign(number) – returns sign of number.

trunc(number) – returns integer part of number.

See ‘MCOE.REXA.REXX(BUILTFUN)’

DATE()



returns, by default, the local date in the format: *dd mon yyyy* (day, month, year—for example, `25 Dec 2001`), with no leading zero or blank on the day. Otherwise, the string *input_date* is converted to the format specified by *date_format1*. *date_format2* can be specified to define the current format of *input_date*. The default for *date_format1* and *date_format2* is **N**ormal. *input_date* must not have a leading zero or blank.

date() – returns the local date.

DATE()

Base	the number of complete days (that is, not including the current day) since and including the base date, 1 January 0001, in the format: *dddddd* (no leading zeros or blanks). The expression `DATE('B')//7` returns a number in the range `0`–`6` that corresponds to the current day of the week, where `0` is Monday and `6` is Sunday. Thus, this function can be used to determine the day of the week independent of the national language in which you are working.
Century	the number of days, including the current day, since and including January 1 of the last year that is a multiple of 100 in the form: *ddddd* (no leading zeros). Example: A call to DATE(C) on March 13, 1992, returns 33675, the number of days from 1 January 1900 to 13 March 1992. Similarly, a call to DATE(C) on November 20, 2001, returns 690, the number of days from 1 January 2000 to 20 November 2001.
Days	the number of days, including the current day, so far in this year in the format: *ddd* (no leading zeros or blanks).
European	date in the format: *dd/mm/yy*
Julian	date in the format: *yyddd*.

DATE()

Month	full English name of the current month, in mixed case—for example, `August`. Only valid for *date_format1*.
Normal	date in the format: *dd mon yyyy*, in mixed case. **This is the default**. If the active language has an abbreviated form of the month name, then it is used—for example, `Jan`, `Feb`, and so on. If **N**ormal is specified (or allowed to default) for *date_format2*, the *input_date* must have the month (*mon*) specified in the English abbreviated form of the month name in mixed case.
Ordered	date in the format: *yy/mm/dd* (suitable for sorting, and so forth).
Standard	date in the format: *yyyymmdd* (suitable for sorting, and so forth).
Usa	date in the format: *mm/dd/yy*.
Weekday	the English name for the day of the week, in mixed case—for example, `Tuesday`. Only valid for *date_format1*.

DATE()

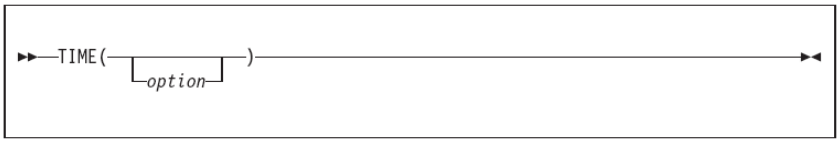
Here are some examples, assuming today is November 20, 2001:

```
DATE()                      ->    '20 Nov 2001'
DATE(,'20020609','S')       ->    '9 Jun 2002'
DATE('B')                   ->    '730808'
DATE('B','25 Sep 2001')     ->    '730752'
DATE('C')                   ->    '690'
DATE('E')                   ->    '20/11/01'
DATE('J')                   ->    '01324'
DATE('M')                   ->    'November'
DATE('N')                   ->    '20 Nov 2001'
DATE('N','1438','C')        ->    '8 Dec 2003'
DATE('O')                   ->    '01/11/20'
DATE('S')                   ->    '20011120'
DATE('U')                   ->    '11/20/01'
DATE('U','25 May 2001')     ->    '05/25/01'
DATE('U','25 MAY 2001')     ->    ERROR,month not in mixed case
DATE('W')                   ->    'Tuesday'
```

36

See ‘MCOE.REXA.REXX(BUILTFUN)’

TIME()



returns the local time in the 24-hour clock format: hh:mm:ss (hours, minutes, and seconds) by default, for example, `04:41:37`.

You can use the following *option*s to obtain alternative formats, or to gain access to the elapsed-time clock. (Only the capitalized and highlighted letter is needed; all characters following it are ignored.)

Civil	returns the time in Civil format: hh:mmxx. The hours may take the values `1` through `12`, and the minutes the values `00` through `59`. The minutes are followed immediately by the letters `am` or `pm`. This distinguishes times in the morning (12 midnight through 11:59 a.m.—appearing as `12:00am` through `11:59am`) from noon and afternoon (12 noon through 11:59 p.m.—appearing as `12:00pm` through `11:59pm`). The hour has no leading zero. The minute field shows the current minute (rather than the nearest minute) for consistency with other TIME results.

37

time() – returns the local time.

TIME()

Elapsed	returns ssssssss.uuuuuu, the number of seconds.microseconds since the elapsed-time clock (described later) was started or reset. The number has no leading zeros or blanks, and the setting of NUMERIC DIGITS does not affect the number. The fractional part always has six digits.
Hours	returns up to two characters giving the number of hours since midnight in the format: hh (no leading zeros or blanks, except for a result of `0`).
Long	returns time in the format: hh:mm:ss.uuuuuu (uuuuuu is the fraction of seconds, in microseconds). The first eight characters of the result follow the same rules as for the Normal form, and the fractional part is always six digits.
Minutes	returns up to four characters giving the number of minutes since midnight in the format: mmmm (no leading zeros or blanks, except for a result of `0`).
Normal	returns the time in the default format hh:mm:ss, as described previously. The hours can have the values `00` through `23`, and minutes and seconds, `00` through `59`. All these are always two digits. Any fractions of seconds are ignored (times are never rounded up). **This is the default**.
3	

TIME()

Reset — returns sssssssss.uuuuuu, the number of seconds.microseconds since the elapsed-time clock (described later) was started or reset and also resets the elapsed-time clock to zero. The number has no leading zeros or blanks, and the setting of NUMERIC DIGITS does not affect the number. The fractional part always has six digits.

Seconds — returns up to five characters giving the number of seconds since midnight in the format: sssss (no leading zeros or blanks, except for a result of 0).

Here are some examples, assuming that the time is 4:54 p.m.:

```
TIME()       ->   '16:54:22'
TIME('C')    ->   '4:54pm'
TIME('H')    ->   '16'
TIME('L')    ->   '16:54:22.123456'   /* Perhaps */
TIME('M')    ->   '1014'           /* 54 + 60*16 */
TIME('N')    ->   '16:54:22'
TIME('S')    ->   '60862'  /* 22 + 60*(54+60*16) */
```

39

See ‘MCOE.REXA.REXX(BUILTFUN)’

TSO/E External functions

In addition to the built-in functions, TSO/E provides external functions that you can use to do specific tasks:

- GETMSG - returns in variables a system message issued during an extended console session. It also returns in variables associated information about the message.
- LISTDSI - returns in variables the data set attributes of a specified data set.
- MSG - controls the display of TSO/E messages. The function returns the previous setting of MSG (ON/OFF).
- MVSVAR - uses specific argument values to return information about MVS, TSO/E, and the current session.
- OUTTRAP - traps lines of TSO/E command output into a specified series of variables. The function call returns the variable name specified.

TSO/E External functions

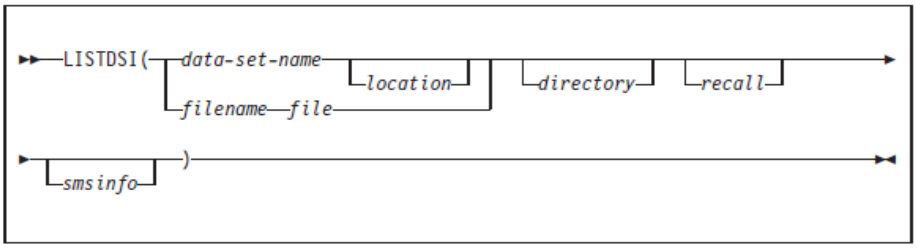
- PROMPT - sets the prompt option ON/OFF for TSO/E interactive commands. The function returns the previous setting of prompt.
- SETLANG - retrieves and optionally changes the language in which REXX messages are displayed. The function returns the previous language setting.
- STORAGE - retrieves and optionally changes the value in a storage address. Carefully!
- SYSCPUS - returns in a stem variable information about all CPUs that are on-line.
- SYSDSN - returns OK if the specified data set exists; otherwise, it returns an appropriate error message.
- SYSVAR - uses specific argument values to return information about the user, terminal, language, exec, system, and console session.

TSO/E External functions

LISTDSI

You can use the LISTDSI (List Dataset Information) function to retrieve detailed information about a data set's attributes.

LISTDSI does not support tape datasets. LISTDSI supports GDG data sets when using absolute generation names, but does not support relative GDG names. LISTDSI does not support HFS data sets.



42

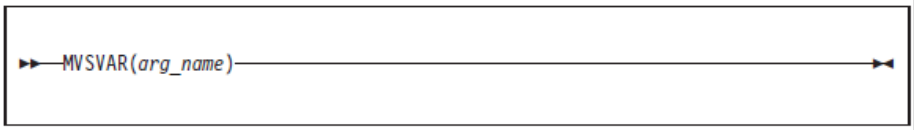
See 'MCOE.REXA.REXX(DSNINFO)'

TSO/E External functions

MVSVAR

MVSVAR returns information about MVS, TSO/E, and the current session, such as the symbolic name of the MVS system, or the security label of the TSO/E session.

The MVSVAR function is available **in any MVS address space.**



43

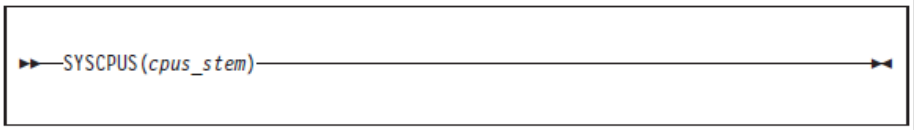
See ‘MCOE.REXA.REXX(MVSINFO)’

TSO/E External functions

SYSCPUS

SYSCPUS places, in a stem variable, information about those CPUs that are on-line.

The SYSCPUS function runs **in any MVS address space.**



On a z990 machine or later, all CPU numbers are identical; therefore, SYSCPUS returns the same value for all CPUs.

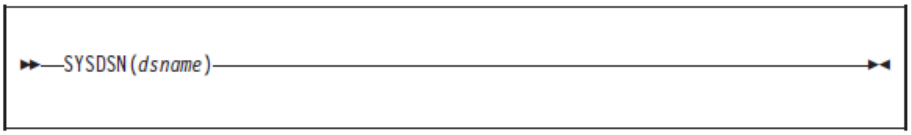
See ‘MCOE.REXA.REXX(CPUINFO)’

TSO/E External functions

SYSDSN

SYSDSN returns whether the specified data set exists and is available for use. The **dsname** can be the name of any cataloged data set or cataloged PDS with a member name. Additionally, if you specify a member of a PDS, SYSDSN checks to see if you have access to the data set.

SYSDSN does not support tape datasets. SYSDSN supports generation data group (GDG) data sets when using absolute generation names, but does not support relative GDG names.



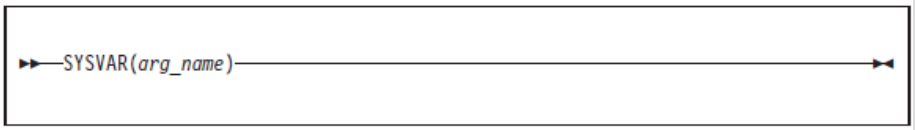
45

See ‘MCOE.REXA.REXX(DSNINFO)’

TSO/E External functions

SYSVAR

The SYSVAR function retrieves information about MVS, TSO/E, and the current session, such as levels of software available, your logon procedure, and your user ID.



46

SYSVAR function is very similar to MVSVAR but there is different set of arguments.

See ‘MCOE.REXA.REXX(SYSINFO)’

See more examples in „TSO REXX Users Guide“

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
***
```

47

You should use nested functions **translate** and **date**.

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

48

It checks only the first name – FN.

Work section 7.2 (output)

```
 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.
 ***
```