

ARx_Func.ag

COLLABORATORS

	<i>TITLE :</i> ARx_Func.ag		
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>	<i>SIGNATURE</i>
WRITTEN BY		August 3, 2022	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

1	ARx_Func.ag	1
1.1	ARexxGuide Functions Reference	1
1.2	ARexxGuide Functions Reference ABOUT	1
1.3	ARexxGuide Functions Reference REXXSUPPORT.LIBRARY	2
1.4	ARexxGuide Functions reference INDEX to built-in & support functions	3
1.5	ARexxGuide Functions reference (1 of 12) COMPARISON	4
1.6	ARexxGuide Functions reference Comparison (1 of 7) ABBREV	5
1.7	ARexxGuide Functions reference Comparison (2 of 7) COMPARE	6
1.8	ARexxGuide Functions reference Comparison (3 of 7) FIND	6
1.9	ARexxGuide Functions reference Comparison (4 of 7) INDEX	7
1.10	ARexxGuide Functions reference Comparison (5 of 7) LASTPOS	8
1.11	ARexxGuide Functions reference Comparison (6 of 7) POS	8
1.12	ARexxGuide Functions reference Comparison (7 of 7) VERIFY	9
1.13	ARexxGuide Functions reference (2 of 12) STRING MANIPULATION	10
1.14	ARexxGuide Functions reference String (1 of 15) CENTER	11
1.15	ARexxGuide Functions reference String (2 of 15) COMPRESS	11
1.16	ARexxGuide Functions reference String (3 of 15) COPIES	12
1.17	ARexxGuide Functions reference String (4 of 15) DELSTR	12
1.18	ARexxGuide Functions reference String (5 of 15) INSERT	13
1.19	ARexxGuide Functions reference String (6 of 15) LEFT	13
1.20	ARexxGuide Functions reference String (7 of 15) LENGTH	14
1.21	ARexxGuide Functions reference String (8 of 15) OVERLAY	14
1.22	ARexxGuide Functions reference String (9 of 15) REVERSE	15
1.23	ARexxGuide Functions reference String (10 of 15) RIGHT	15
1.24	ARexxGuide Functions reference String (11 of 15) STRIP	16
1.25	ARexxGuide Functions reference String (12 of 15) SUBSTR	16
1.26	ARexxGuide Functions reference String (13 of 15) TRANSLATE	17
1.27	ARexxGuide Functions reference String (14 of 15) TRIM	18
1.28	ARexxGuide Functions reference String (15 of 15) UPPER	18
1.29	ARexxGuide Functions reference Number (9 of 9) XRANGE	19

Chapter 1

ARx_Func.ag

1.1 ARexxGuide | Functions Reference

AN AMIGAGUIDE® TO ARexx
by Robin Evans

Second edition (v2.0)

About this section

ARexx functions:

Comparison functions

FIND(), POS(), ABBREV(), etc.

String manipulation

LEFT(), SUBSTR(), etc.

Word manipulation WORD(), DELWORD(), etc.

Char/Num translation C2D(), X2C(), D2X() etc.

Number manipulation RANDOM(), MAX(), etc.

Informational DATE(), SHOW(), etc.

File input/output OPEN(), READLN(), SEEK(), etc.

File management DELETE(), EXISTS(), RENAME(), etc.

ARexx control ADDLIB(), PRAGMA(), etc.

Message ports OPENPORT(), WAITPKT(), GETARG(), etc.

Memory management IMPORT(), NEXT(), NULL(), etc.

Bit-wise operations BITAND(), BITCOMP(), etc.

All functions [INDEX]

Copyright © 1993,1994 Robin Evans. All rights reserved.

This guide is shareware . If you find it useful, please register.

1.2 ARexxGuide | Functions Reference | ABOUT

REFERENCE TO BUILT-IN AND REXXSUPPORT.LIBRARY FUNCTIONS

~~~~~

This section presents a reference to built-in functions and to the

```
functions included in
    rexxsupport.library
.
```

Each node begins with a template that shows the format of the arguments accepted by the function. The following conventions are used:

- rv = Each function is shown as part of an assignment clause to emphasize that functions are expressions. The variable name [rv] is used as an arbitrary abbreviation for 'return value'.
- <> A word or term surrounded by angle brackets should be replaced by an expression. Any form of expression that returns a value of the proper type may be used in place of this item. The replacement is often a variable, but it could also be a literal string, a number, an operation, or another function call.

The names used for the items in each template are included as mnemonic devices -- terms that may help the user remember what each expression stands for. They are not otherwise significant.

Each term is explained in more detail in the note following the template.

- [ ] Items enclosed in square brackets are optional. They may be excluded when the function is called, but the commas that separate optional items are significant. If only the second of two optional items is included, a comma must still be used as a placeholder for the omitted item as it is in the following:

```
SHOW('P',,'0a'x)
```

- { } Items enclosed in curly braces and entered in uppercase are literal values. The expression used for such an argument must return one of the values from the list.
- | A bar is used to separate a list of literal values within {} braces.
- <UC> UPPERCASE characters are used to indicate literal values that may be used as argument. The value may be entered in upper or lowercase when the instruction is actually used. Only the first letter of the option need be included. The value may be entered as any type of expression.
- >>> Three angle-braces are used in examples to indicate what the example would output if run from a shell. Those braces and the following text is not part of the code and should not be entered if the example is used.

Next: [REXXSUPPORT.LIBRARY](#) | Prev: [Function ref.](#) | Contents: [Function ref.](#)

## 1.3 ARexxGuide | Functions Reference | REXXSUPPORT.LIBRARY

Functions can be added to ARexx by means of external libraries . One such library is included with the distribution of ARexx. Called 'rexksupport.library', it should be present in the user's libs: directory after ARexx is installed.

The library adds several Amiga-specific functions that are not defined as a standard part of the REXX language. Included are memory-control functions like ALLOCMEM() , file system functions like MAKEDIR() , and interprocess-communication functions like OPENPORT() .

The functions in rexksupport.library will not automatically be available to ARexx scripts, however. They are available only if the library is explicitly added to the list of libraries through which ARexx searches to find functions.

That can be done with the ADDLIB() function or with the RXLIB command utility. Examples of loading the library are included with the description of each of those methods.

Those who frequently run ARexx programs may find it useful to add some libraries to the system during the startup sequence. Doing so doesn't take up much memory because the libraries aren't loaded until they are needed. It does assure that the library will be searched if one of its functions is used in a program.

Adding the following command to the User-Startup file will add the library name to the ARexx list, making the library available whenever it is called.

```
rxlib rexksupport.library 0 -30 0
```

#### Compatibility issues:

The functions in rexksupport.library are, by their nature, system-specific. They are ARexx extensions to the language. All REXX implementations are able to load external libraries, but the language definition makes no attempt to define what can or should be included in the libraries.

Next: [Function ref.](#) | Prev: [About section](#) | Contents: [Function ref.](#)

## 1.4 ARexxGuide | Functions reference | INDEX to built-in & support functions

|        |          |           |        |          |
|--------|----------|-----------|--------|----------|
|        | ABBREV   |           |        |          |
|        | ABS      | ADDRESS   | ADDLIB | ALLOCMEM |
| ARG    | BADDR    | B2C       | BITAND | BITCHG   |
| BITCLR | BITCOMP  | BITOR     | BITSET | BITTST   |
| BITXOR | C2B      | C2D       | C2X    |          |
|        | CENTER   |           |        |          |
|        | CLOSE    | CLOSEPORT |        |          |
|        | COMPARE  |           |        |          |
|        | COMPRESS |           |        |          |
|        | COPIES   |           |        |          |

|          |           |            |        |           |   |
|----------|-----------|------------|--------|-----------|---|
|          | D2C       | DATATYPE   | DATE   | DELAY     | ↔ |
|          | DELETE    |            |        |           |   |
|          | DELSTR    |            |        |           |   |
| EXISTS   | DELWORD   | DIGITS     | EOF    | ERRORTEXT |   |
|          | EXPORT    |            |        |           |   |
|          | FIND      |            |        |           |   |
|          | FORBID    | FORM       |        |           |   |
| FREEMEM  | FREESPACE | FUZZ       | GETARG | GETCLIP   |   |
| GETPKT   | GETSPACE  | HASH       | IMPORT |           |   |
|          | INDEX     |            |        |           |   |
|          | INSERT    |            |        |           |   |
|          | LASTPOS   |            |        |           |   |
|          | LEFT      |            |        |           |   |
|          | LENGTH    |            |        |           |   |
|          | LINES     |            |        |           |   |
| MAKEDIR  | MAX       | MIN        | NEXT   | NULL      |   |
| OFFSET   | OPEN      | OPENPORT   |        |           |   |
|          | OVERLAY   |            |        |           |   |
|          | PERMIT    |            |        |           |   |
|          | POS       |            |        |           |   |
|          | PRAGMA    | RANDOM     | RANDU  | READCH    |   |
| READLN   | REMLIB    | RENAME     | REPLY  |           |   |
|          | REVERSE   |            |        |           |   |
|          | RIGHT     |            |        |           |   |
|          | SEEK      | SETCLIP    | SHOW   | SHOWDIR   |   |
| SHOWLIST | SIGN      | SOURCELINE | SPACE  | STATEF    |   |
| STORAGE  |           |            |        |           |   |
|          | STRIP     |            |        |           |   |
|          | SUBSTR    |            |        |           |   |
|          | SUBWORD   | SYMBOL     |        |           |   |
| TIME     | TRACE     |            |        |           |   |
|          | TRANSLATE |            |        |           |   |
|          | TRIM      |            |        |           |   |
|          | TRUNC     |            |        |           |   |
| TYPEPKT  |           |            |        |           |   |
|          | UPPER     |            |        |           |   |
|          | VALUE     |            |        |           |   |
|          | VERIFY    |            |        |           |   |
|          | WAITPKT   |            |        |           |   |
| WORD     | WORDINDEX | WORDLENGTH | WORDS  | WRITECH   |   |
| WRITELN  | X2C       |            |        |           |   |
|          | XRANGE    |            |        |           |   |

## 1.5 ARexxGuide | Functions reference (1 of 12) | COMPARISON

```

ABBREV
(<longstring>,<shortstring>,[<length>])

COMPARE
(<string1>,<string2>,[<padchar>])

FIND
(<haystack>,<needle>)

INDEX
(<haystack>,<needle>,[<startpos>])

LASTPOS
(<needle>,<haystack>,[<startpos>])

POS
(<needle>,<haystack>,[<startpos>])

VERIFY
(<string>,<reference>,[{'NOMATCH'|'MATCH'}],[<startpos>])

```

Related functions:

```

BITCOMP
BITTST
DATATYPE

```

Also see [Bit manipulation functions](#)

Comparisons of one type or another are one of the most frequent tasks of any program. Comparisons allow a program to branch off to different code based on different conditions. Comparison operators give ARexx the standard tools for matching strings, but these functions extend the power of the operators, allowing quick checks for a substring (what Cowlishaw so elegantly calls a 'needle') in a string (the 'haystack'), or for a word or phrase within a string of words.

The external library package `RexxDosSupport.library`, by Hartmut Goebel, includes functions, `ParsePattern()` and `MatchPattern()`, that use pattern-match routines supplied by the operating system. The routines are, of course, system-specific and non-portable but can be useful when case-insensitive matching is needed or wild-cards must be used in a pattern.

Next: [String functions](#) | Prev: [BITXOR\(\)](#) | Contents: [Function reference](#)

## 1.6 ARexxGuide | Functions reference | Comparison (1 of 7) | ABBREV

```

rv = ABBREV(<longstring>,<shortstring>,[<length>])
rv is boolean value

```

Returns 1 if `<shortstring>` is equal to the leading characters of `<longstring>`. If `<length>` is specified, then `<shortstring>` must also be at least that long. The comparison is case-sensitive. If `<length>` is not



specified, an empty string will always match <longstring>.

Returns 0 if either condition is not met.

Examples:

```
say abbrev('Waldorf','Waldo');          >>> 1
say abbrev('Waldorf','WALDO');          >>> 0
say abbrev(
    upper('Waldorf')
    , 'WALDO');          >>> 1
say abbrev('YES', 'Y')                  >>> 1
say abbrev('YES', '')                   >>> 1
say abbrev('YES', '', 1)                 >>> 0
```

Also see

COMPARE

LEFT

Technique note: Read one file, write to another  
 Extract file name from full spec  
 Data scratchpad with PUSH & QUEUE

Next: COMPARE() | Prev: Comparison func. | Contents: Comparison func.

## 1.7 ARexxGuide | Functions reference | Comparison (2 of 7) | COMPARE

```
rv = COMPARE(<string1>,<string2>,[<padchar>])
rv is a number
```

The result is 0 if both strings are identical. If they aren't, the number returned is the position of the first character where the strings differ. The shorter string is padded with <padchar> before the comparison.

The default pad character is a blank.

Examples:

```
say compare('The first','The only');    >>> 5
say compare('worldwide','wordwide');    >>> 4
say compare('foo','f');                  >>> 2
say compare('foo','f','o');              >>> 0
```

Also see

ABBREV

VERIFY

Next: FIND() | Prev: ABBREV() | Contents: Comparison functions

## 1.8 ARexxGuide | Functions reference | Comparison (3 of 7) | FIND

```

        rv = FIND(<haystack>, <needle>)
rv is a number

```

Locates the blank-delimited word or words <needle> within the string <haystack> and returns the word position of the first match, or 0 if there is no match.

The search is case sensitive.

Examples:

```

say find('Tied to Godot?', 'dot');      >>> 0
say find('Tied to Godot?', 'to');      >>> 2

```

Also see

INDEX

POS

WORDINDEX

Compatibility issues:

In the standard language definition, this function is not defined, but a similar function called WORDPOS() is defined in TRL2 . It takes arguments in the reverse order. WORDPOS() accepts an optional third argument that specifies the word at which the search should begin.

To maintain compatibility, the following user function could be used instead of FIND().

```
/* WordPos() user function */
```

WordPos:

```

if arg(3, 'o') | ~datatype(arg(3), 'N') then
    return find(arg(2), arg(1))
else do
    wpSub = find(subword(arg(2), arg(3)), arg(1))
    if wpSub > 0 then
        return arg(3) + wpSub - 1
    else
        return 0
end

```

Next: INDEX() | Prev: COMPARE() | Contents: Comparison functions

## 1.9 ARexxGuide | Functions reference | Comparison (4 of 7) | INDEX

```

        rv = INDEX(<haystack>, <needle>, [<startpos>])
rv is a number

```

The result is the character position within the string <haystack> of the the first occurrence of the string <needle> or 0 if a match isn't found.

If <startpos> is specified, then the search proceeds from that position in <haystack>.

The search is case sensitive.

Examples:

```
say index('Tied to Godot?','dot');    >>> 11
say index('Tied to Godot?','to');     >>> 6
```

Also see

FIND

POS

Compatibility issues:

This function was supported in early versions of IBM's REXX, but is no longer included in the standard language definition. The POS() function should be used instead.

Next: LASTPOS() | Prev: FIND() | Contents: Comparison functions

## 1.10 ARexxGuide | Functions reference | Comparison (5 of 7) | LASTPOS

```
rv = LASTPOS(<needle>,<haystack>,[<startpos>])
rv is a number
```

The result is the character position within the string <haystack> of the the last occurrence of the string <needle> or 0 if a match isn't found.

If <startpos> is specified, then the search proceeds backwards from that position in <haystack>.

Examples:

```
say lastpos('eak','growing weaker and weaker');    >>> 21
say lastpos('eak','growing weaker and weaker',20); >>> 10
```

Also see

POS

Technique note: Extract file name from full spec  
WordWrap() user function

Next: POS() | Prev: INDEX() | Contents: Comparison functions

## 1.11 ARexxGuide | Functions reference | Comparison (6 of 7) | POS

```
rv = POS(<needle>,<haystack>,[<startpos>])
rv is a number
```

The result is the character position within the string <haystack> of the the first occurrence of the string <needle> or 0 if a match isn't found.

If <startpos> is specified, then the search proceeds forward from that position in <haystack>.

The search is case sensitive.

Examples:

```
say pos('eak','growing weaker and weaker')    >>> 10
say pos('eak','growing weaker and weaker',11)  >>> 21
```

Also see

LASTPOS

VERIFY

Technique note: Extract file name from full spec  
Using the clip list

Next: VERIFY() | Prev: LASTPOS() | Contents: Comparison functions

## 1.12 ARexxGuide | Functions reference | Comparison (7 of 7) | VERIFY

```
rv = VERIFY(<string>, <reference>, ['Match'], [<startpos>])
rv is a number
```

Checks for the presence in <string> of any characters that appear in <reference> -- a list of characters which may be entered in any order.

If the 'MATCH' option is omitted (or if any other value is used as an argument), then the function returns 0 when all characters in <string> are contained in <reference>. If a character in <string> is not included in <reference> the return is a positive integer that indicates the position of the first character in <string> that does not match a character in <reference>.

The 'M' (match) option will cause the function to return the position of the first character in <string> that matches a character in <reference>. It returns 0 if none of the characters in <string> match a character in <reference>.

If <startpos> is specified, the search will begin at that character position in <string>.

Examples:

```
say verify('#789-ABD', '1234567890ABCD-#')    >>> 0
say verify('#432-cfo', '1234567890ABCD-#')    >>> 6
say verify('FileName', '.*/*?`#%', 'm')       >>> 0
say verify('File*NAME', '.*/*?`#%', 'm')       >>> 5
say verify('File*NAME', '.*/*?`#%', 'm', 6)    >>> 0
say verify('t:foo/file', ':/', 'm')           >>> 2
say verify('a', 'AEIOUaeiou')                 >>> 0
say verify('vowel', 'AEIOUaeiou', 'm')        >>> 2
say verify('vowel', 'AEIOUaeiou', 'm', 3)     >>> 4
```

Also see DATATYPE

POS

Technique note: Check unique datatypes

Divide a word at non-space char.

Next: Comparison functions | Prev: POS() | Contents: Comparison functions

## 1.13 ARexxGuide | Functions reference (2 of 12) | STRING MANIPULATION

CENTER  
(<string>, <length>, [<padchar>])

COMPRESS  
(<string>, [<list>])

COPIES  
(<string>, <number>)

DELSTR  
(<string>, <number>, [<length>])

INSERT  
(<new string>, <old string>, <startpos>, [<length>], [<padchar>])

LEFT  
(<string>, <length>, [<padchar>])

LENGTH  
(<string>)

OVERLAY  
(<new string>, <old string>, [<startpos>], [<length>], [<padchar>])

REVERSE  
(<string>)

RIGHT  
(<string>, <length>, [<padchar>])

STRIP  
(<string>, [{'B'|'L'|'T'}], [<list>])

SUBSTR  
(<string>, <startpos>, [<length>], [<padchar>])

TRANSLATE  
(<string>, [<output table>], [<input table>], [<padchar>])

TRIM  
(<string>)

UPPER  
(<string>)

XRANGE  
([<start>, [<end>])

---

Also see [Word manipulation functions](#)  
[Number manipulation functions](#)  
[PARSE instruction](#)

Nearly any change one might contemplate for a string can be made with one, or a combination of these functions, or one of the closely-allied word manipulation functions. They'll cut chunks out of a string -- `LEFT()`, `RIGHT()`, `SUBSTR()`, `DELSTR()`; or remove only certain characters -- `STRIP()`, `TRIM()`, `COMPRESS()`; or add to the string -- `OVERLAY()`, `INSERT()`, `COPIES()`, `CENTER()`; or transform it in subtle and wonderful ways -- `TRANSLATE()`, `REVERSE()`.

Next: [Word functions](#) | Prev: [FuncList](#) | Contents: [Function reference](#)

## 1.14 ARexxGuide | Functions reference | String (1 of 15) | CENTER

```
rv = CENTER(<string>,<length>,[<padchar>])
rv is a string
```

The function name may be spelled `CENTRE` or `CENTER`.

The result is a string of `<length>` characters with `<string>` centered in it. The `<padchar>` is used to fill out the left and right sides of the string. The default pad character is a blank.

Example:

```
say ['center('Title',20)'];      >>> [      Title      ]
say center('Title',22,'*');    >>> *****Title*****
```

Also see [SPACE](#)

[COPIES](#)  
Next: [COMPRESS\(\)](#) | Prev: [String functions](#) | Contents: [String](#) ↔  
[functions](#)

## 1.15 ARexxGuide | Functions reference | String (2 of 15) | COMPRESS

```
rv = COMPRESS(<string>,[<list>])
rv is a string
```

Removes any of the characters contained in `<list>` from `<string>`. The default character for `<list>` is a blank, so this function will remove all blanks if only `<string>` is specified.

Examples:

```
say compress('$1,045','$,%');    >>> 1045
say compress('Call me Ismael.');
```

Also see

TRANSLATE

STRIP

SPACE

Technique note: CountChar() user function

Compatibility issues:

This function is an extension that is not defined in TRL2 . Although a function of this name might be included in other REXX implementations, there is no assurance that it will be.

Next: COPIES() | Prev: CENTER() | Contents: String functions

## 1.16 ARexxGuide | Functions reference | String (3 of 15) | COPIES

```
rv = COPIES(<string>,<number>)
rv is a string
```

The result is a new string composed of <string> concatenated with itself <number> times.

Example:

```
say copies('xo',6);          >>> xoxoxoxoxo
```

Also see

XRANGE

CENTER

Technique note: Format a table of information  
AddComma() user function

Next: DELSTR() | Prev: COMPRESS() | Contents: String functions

## 1.17 ARexxGuide | Functions reference | String (4 of 15) | DELSTR

```
rv = DELSTR(<string>,<number>,[<length>])
rv is a string
```

Deletes a portion of <string> of <length> characters beginning at the <number> character position. The new string is returned. If <number> is greater than the length of <string> then <string> is returned unchanged.

If <length> is omitted, all characters beginning at position <number> are deleted.

Example:

```
say delstr('indifference',3,3);    >>> inference
```

Also see DELWORD

```

RIGHT

SUBSTR

INSERT

OVERLAY
Next: INSERT() | Prev: COPIES() | Contents: String functions

```

## 1.18 ARexxGuide | Functions reference | String (5 of 15) | INSERT

```

rv = INSERT(<newstr>, <oldstr>, [<startpos>], [<length>],[<padchar <←
>])
rv is a string

```

<newstr> is inserted into <oldstr> beginning at <startpos>, the character-count position. <newstr> will be padded with <padchar> or truncated to <length> characters.

If <startpos> is greater than the length of <oldstr> then <padchar> will be added to the end of <oldstr> before the new string is added. If <startpos> is 0 or is omitted, then <newstr> will be padded to <length> and then added to the start of <oldstr>

The default length is the length of <newstr>. The default pad character is a blank.

Example:

```

say insert('always behaved like','I have a pig.',7,20)
>>> I have always behaved like a pig.

```

Also see

```

OVERLAY

DELSTR
Technique note: WordWrap() user function

```

Next: LEFT() | Prev: DELSTR() | Contents: String functions

## 1.19 ARexxGuide | Functions reference | String (6 of 15) | LEFT

```

rv = LEFT(<string>,<length>,[<padchar>])
rv is a string

```

The result is a string of <length> characters made up of the leftmost characters in <string>. If <length> is greater than the length of <string>, then the string returned is filled out on the right with <padchar> -- a quick way to left-justify a string.



The default pad character is a blank.

Example:

```
say left('never to stop saying',13);    >>> never to stop
say left('Widget', 12)'|'              >>> Widget      |
say left('No', 4, '!')                 >>> No!!
```

Also see

RIGHT

SUBSTR

ABBREV

Technique note: Formatting tables  
 Extract file name from full spec  
 Determine library version number

Next: LENGTH() | Prev: INSERT() | Contents: String functions

## 1.20 ARexxGuide | Functions reference | String (7 of 15) | LENGTH

```
rv = LENGTH(<string>)
rv is a number
```

The result is the number of characters in <string>.

Example:

```
say length('never to stop saying');    >>> 20
```

Technique note: CountChar() user function  
 AddComma() user function  
 WordWrap() user function  
 Check unique datatypes

Next: OVERLAY() | Prev: LEFT() | Contents: String functions

## 1.21 ARexxGuide | Functions reference | String (8 of 15) | OVERLAY

```
rv = OVERLAY(<newstr>, <oldstr>,[<startpos>], [<length>],[<padchar ←
>])
rv is a string
```

Replaces the characters of <oldstr> starting at position <startpos> with the characters of <newstr>. The default starting position is the beginning of <oldstr>.

If <length> is not specified, all of the characters from <newstr> will be overlaid on <oldstr>. If <length> is specified, then <newstr> will either be truncated to that length or expanded to <length> using <padchar> to fill out the string.

The default pad character is a blank.

Examples:

```
say overlay('12', 'abcdefg', 3, 4, '*')    >>> ab12**g
say overlay('abc', '12345678', 4, 2)      >>> 123ab678
say overlay('think of it',,
  'the less I concentrate the more certain I am', 12)
  >>> the less I think of it the more certain I am
```

Also see

INSERT

DELSTR

The third example above uses the comma continuation character to ↵  
turn

two lines of text into one program line.

Next: REVERSE() | Prev: LENGTH() | Contents: String functions

## 1.22 ARexxGuide | Functions reference | String (9 of 15) | REVERSE

```
rv = REVERSE(<string>)
rv is a string
```

The result is <string> flipped end for end.

Example:

```
say reverse('chameleon');    >>> noelemahc
```

Also see

LASTPOS

Technique note: Add commas to a number

Next: RIGHT() | Prev: OVERLAY() | Contents: String functions

## 1.23 ARexxGuide | Functions reference | String (10 of 15) | RIGHT

```
rv = RIGHT(<string>,<length>,[<padchar>])
rv is a string
```

The result is a string of <length> characters made up the rightmost characters in <string>. If <length> is greater than the length of <string>, then the result is filled out on the left with <padchar> -- a quick way to right-justify a string.

The default pad character is a blank.

Example:

```
say right('never to stop saying',11);    >>> stop saying
say '$'right(4.50, 6)                    >>> $ 4.50
```

```
say '$'right(123.99, 6)          >>> $123.99
say right('Whoa', 6, 'W')      >>> WWWhoa
```

Also see

LEFT

SUBSTR

DELSTR

Technique note: Formatting tables  
Determine library version number

Next: STRIP() | Prev: REVERSE() | Contents: String functions

## 1.24 ARexxGuide | Functions reference | String (11 of 15) | STRIP

```
rv = STRIP(<string>, [{'B'|'L'|'T'}], [<list>])
rv is a string
```

Removes spaces (by default) or any character in <list> from the leading, trailing, or both ends (specified by the option used as the second argument) of <string>. The default option is 'B'.

Example:

```
say '|'|strip(' understand ')'|';          >>> |understand|
say '|'|strip(' understand ',L)'|';        >>> |understand |
say '|'|strip('___understand___',T,'_')'|'; >>> |___understand|
say strip('understand',, 'dnu')           >>> ersta
```

The examples use the abuttal concatenation operator to add the character '|' to the beginning and end of the string returned by STRIP().

Also see

COMPRESS

TRIM

Technique note: AddComma() user function

Compatibility issues:

Standard REXX accepts only a single character where ARexx accepts a <list> of characters to be stripped. Using a multiple-character list will cause an error in most other implementations of the language.

Next: SUBSTR() | Prev: RIGHT() | Contents: String functions

## 1.25 ARexxGuide | Functions reference | String (12 of 15) | SUBSTR

```
rv = SUBSTR(<string>, <startpos>, [<length>], [<padchar>])
rv is a string
```

The result is a string of <length> characters made up the characters in <string> beginning at <startpos>.

If <length> is not specified, then all of the string to the right of <startpos> will be returned. If the argument is specified, the returned string will have <length> characters, filled out, if necessary, with <padchar>.

The default pad character is a blank.

Example:

```
say substr('indifference',3,3)      >>> dif
say substr('No way',4,5,'!')      >>> way!!
```

Also see

```
LEFT
RIGHT
DELSTR
SUBWORD
TRUNC
```

```
Technique note: Format() user function
                WordWrap() user function
                Extract file name from full spec
```

Next: TRANSLATE() | Prev: STRIP() | Contents: String functions

## 1.26 ARexxGuide | Functions reference | String (13 of 15) | TRANSLATE

```
rv = TRANSLATE(<string>, [<output table>], [<input table>], [< ←
                        padchar>])
rv is a string
```

Any character in <string> that also appears in the <input table> is converted to the corresponding character in the <output table> or to the <padchar> if there isn't a corresponding character in the <output table>.

If neither table is supplied, then the <string> is converted to upper case, just as it would be by

```
UPPER(<string>)
```

.

The default pad character is a blank.

Examples:

```
say translate('abcdef', '123456', 'abcdef') >>> 123456
say translate('abcdef', '123456', 'defabc') >>> 456123
say translate('abcdef', '1234', 'defabc','*') >>> 4**123
say translate('UNNAMABLE', xrange('a','z'), xrange('A','Z'))
>>> unnamable
```

Translate a string to lowercase

~~~~~

<string> can be translated to lowercase with the following function:

```
string = translate(string, 'abcdefghijklmnopqrstuvwxy',,
                   'ABCDEFGHIJKLMNopQRSTUVWXYZ')
```

If one isn't worried about incompatibility that would arise from non-ASCII character sets, then the list of characters can be replaced by these calls to the

```
XRANGE()
function:
```

```
string = translate(string, xrange('a','z'), xrange('A', 'Z'))
```

The function BITOR(<string>) will also translate the alphabetic characters in <string> to lowercase characters, but it will shift the ASCII characters between 91 and 95 { [\] ^ _} to characters 123 through 127.

Also see

```
COMPRESS
Next: TRIM() | Prev: SUBSTR() | Contents: String functions
```

1.27 ARexxGuide | Functions reference | String (14 of 15) | TRIM

```
rv = TRIM(<string>)
rv is a string
```

The result is <string> with the trailing blanks removed.

Example:

```
say '|trim(' understand ')'|'; >>> | understand|
```

Also see

```
STRIP
Compatibility issues:
```

This function is an extension that is not defined in TRL2 . Although a function of this name might be included in other REXX implementations, there is no assurance that it will be.

```
Next: UPPER() | Prev: TRANSLATE() | Contents: String functions
```

1.28 ARexxGuide | Functions reference | String (15 of 15) | UPPER

```
rv = UPPER(<string>)
rv is a string
```

The result is <string> translated to all uppercase characters.

Example:

```
say upper('Waldorf')      >>> WALDORF
```

Also see

[TRANSLATE](#)

Includes note on translating a string to lowercase.

Technique note: [Data scratchpad with PUSH & QUEUE](#)

Compatibility issues:

This function is an extension that is not defined in TRL2 . Although a function of this name might be included in other REXX implementations, there is no assurance that it will be. The `TRANSLATE(<string>)` function, without other options, does the same thing.

Next: [XRANGE\(\)](#) | Prev: [TRIM\(\)](#) | Contents: [String functions](#)

1.29 ARexxGuide | Functions reference | Number (9 of 9) | XRANGE

```
rv = XRANGE([<start>, [<end>]])
rv is a string
```

The result is a string comprised of all the characters between and including <start> and <end>.

The output of the function is a character string. Use the `c2x()` function, for example, to convert the output to hexadecimal number format.

Examples:

```
say xrange('a','g');      >>> abcdefg
say xrange(1,8);          >>> 12345678
say c2x(xrange('c'x,'14'x)); >>> 0C0D0E0F1011121314
```

Also see

[COPIES](#)

Technique note: [Check unique datatypes](#)

Next: [String functions](#) | Prev: [UPPER\(\)](#) | Contents: [String functions](#)