`stringlib` — library for string manipulation

Table of contents

`stringlib::contains` – **test for substring**

With `stringlib::contains` a string can be tested whether it contains another string.

Call(s):

- `stringlib::contains(string1, string2 <, option>)`

Parameters:

`string1, string2` — non empty string

Options:

`Index`	— causes the first index position at which `string2` appears in `string1` to be returned. The return value is `0` if `string2` occurs nowhere in `string1`.
`IndexList`	— causes the list of all positions at which `string2` appears in `string1` to be returned. The returned list may be empty.

Return Value: `TRUE`, if `string1` contains `string2`, otherwise `FALSE`. An integer (or a list of integers) that determines the position, if an option is given.

Related Functions: `contains`, `stringlib::pos`

Details:

- An occurrence of `string2` is not detected if overlapped by the tail of a previously detected occurrence. See Example 2.

Example 1. If called without options, `stringlib::contains` simply returns `TRUE` or `FALSE`.

```
>> stringlib::contains("abcdeabcdeabcde", "bc")

                         TRUE
```

```
>> stringlib::contains("abcdeabcdeabcde", "bc", Index)

                          1
```

```
>> stringlib::contains("abcdeabcdeabcde", "bc", IndexList)

                      [1, 6, 11]
```

Example 2. The following call does *not* return `[0,1]` because the first matching substring has not ended when the second begins.

```
>> stringlib::contains("aaa", "aa", IndexList)

                              [0]
```

Changes:

- `stringlib::contains` used to be `string::contains`.

`stringlib::format` – Formatting a string

`stringlib::format` adjusts the length of a string.

Call(s):

- `stringlib::format(string1, width, <, alignment> <, fill_char>)`

Parameters:

`string1`	—	string
`width`	—	integer that determines the length of the returned string
`alignment`	—	*`Left`*, *`Center`*, or *`Right`*
`fill_char`	—	one-character string to fill up the result string

Options:

`Left`	—	determines that the string will be aligned left
`Center`	—	determines that the string will be centered
`Right`	—	determines that the string will be aligned right

Return Value: a string of length `width` containing the given string

Related Functions: `stringlib::formatf`

Details:

- If `width` is less than the length of the given string `string1`, the substring consisting of the first `width` characters of `string1` is returned.

- If `width` exceeds the length of `string1`, the given string will be filled with the necessary number of spaces or the optional `fill_char`. These are inserted at the end in case of left alignment, or at the beginning in case of right alignment. In case of centering, the same number of filling characters is placed at the beginning and at the end, but one more is placed at the end if their total number is odd.
- If `alignment` is not given, left alignment is used by default.

Example 1. By default, a string of length 5 is adjusted to length 10 by inserting five space characters at the end.

```
>> stringlib::format("abcde", 10)

                           "abcde     "
```

In the case of centering, three spaces are inserted at the end and two at the beginning.

```
>> stringlib::format("abcde", 10, Center)

                           "  abcde   "
```

Instead of the space character, also any other character may be used as a filling character.

```
>> stringlib::format("abcde", 10, Right, ".")

                           ".....abcde"

>> stringlib::format("abcde", 10, ".")

                           "abcde....."
```

Changes:

- `stringlib::format` used to be `string::format`.

`stringlib::formatf` – Convert a floating point number to a string

`stringlib::formatf(f, d)` converts the floating point number `f` into a string after rounding it to `d` digits after the decimal point.

Call(s):

- `stringlib::formatf(f, digits <, strlength>)`

Parameters:

`f` — floating point number
`digits` — integer which determines the precision of the number
`strlength` — integer which determines the length of the returned string

Return Value: `stringlib::formatf` returns a string.

Related Functions: `stringlib::format`

Details:

- If `d` is a positive integer, a rounded fixed-point representation with `d` digits after the decimal point is returned. If `d` is zero, then a rounded fixed-point representation with one zero after the decimal point is returned. If `d` is negative, then `f` is rounded to `-d` digits before the decimal point and a fixed-point representation with one zero after the decimal point is returned.
- The representation of a negative number starts with the sign and no additional spaces. The representation of a nonnegative number starts with a single space character.
- If a third argument is specified, then the string returned consists of exactly `strlength` characters. If the converted number `f` requires less room, then it is padded on the left with spaces. If the converted number `f` requires more room, then the last characters are truncated.

Example 1. Convert the number `123.456` with two characters after the point into a string:

```
>> stringlib::formatf(123.456, 2)

                            " 123.46"
```

The same for `-123.456`:

```
>> stringlib::formatf(-123.456, 2)

                            "-123.46"
```

Convert the number `123.456` with two characters after the point into a string of the length `10`:

```
>> stringlib::formatf(123.456, 2, 10)

                          "    123.46"
```

If the string should only have the length `3`, the whole number does not fit into the string:

```
>> stringlib::formatf(123.456, 2, 3)

                                            " 12"
```

Rounding to no number after point:

```
>> stringlib::formatf(123.456, 0)

                                          " 123.0"
```

Rounding to one number in front of point:

```
>> stringlib::formatf(123.456, -1)

                                          " 120.0"
```

Changes:

- `stringlib::formatf` used to be `string::formatf`.

`stringlib::pos` – Position of a substring

`stringlib::pos` returns the position of a substring in a string.

Call(s):

- `stringlib::pos(string1, string2 <, pos>)`

Parameters:

`string1, string2`	—	non empty string
`pos`	—	integer that determines the first position to search

Return Value: An integer that determines the position or `FAIL`.

Related Functions: `stringlib::contains`, `length`

Details:

- The third optional argument must be less than the length of `string1`.
- If `string1` does not contain `string2`, then `FAIL` will be returned.

Example 1. In case of several occurrences of the substring, the position of the first is returned.

```
>> stringlib::pos("abcdeabcdeabcde", "bc")

                               1
```

Example 2. If a starting point for the search is given, `stringlib::pos` returns the first position at which the substring occurs after that starting point.

```
>> stringlib::pos("abcdeabcdeabcde", "bc", 5)

                               6
```

Example 3. The result is `FAIL` if the substring does not occur at all or after the given starting point.

```
>> stringlib::pos("abcdeabcdeabcde", "bc", 12)

                             FAIL
```

Changes:

- `stringlib::pos` used to be `string::pos`.

`stringlib::remove` – Delete substrings

With `stringlib::remove` a substring can be deleted from another string.

Call(s):

- `stringlib::remove(string1, string2 <, option>)`

Parameters:

`string1, string2` — non empty string

Options:

`First` — determines that only the first appearance of `string2` in `string1` will be deleted

Return Value: the given string without the deleted parts

Related Functions: `delete`, `stringlib::subs`, `stringlib::subsop`

Details:

- After a `string2` has been found, the search for further occurrences of it continues after its last letter; hence only the first of several overlapping occurrences is detected. See Example 3.

Example 1. By default, out of several occurrences of the given substring *all* are removed.

```
>> stringlib::remove("abcdeabcdeabcde", "bc")

                            "adeadeade"
```

Example 2. Using the option `First` causes `stringlib::remove` to remove only the first occurrence of the given substring.

```
>> stringlib::remove("abcdeabcdeabcde", "bc", First)

                          "adeabcdeabcde"
```

Example 3. In the following example, the given substring occurs twice, where both instances of it do overlap. Only the first occurrence is removed.

```
>> stringlib::remove("aaa", "aa")

                                "a"
```

Changes:

- `stringlib::remove` used to be `string::delete`.

`stringlib::subs` – Substitution in a string

`stringlib::subs` substitutes a substring by another string.

Call(s):

```
stringlib::subs(string, substring = replacement,
                <First>)
```

Parameters:

`string` — non empty string
`substring` — non empty string that should be replaced
`replacement` — any string that replaced `substring`

Options:

`First` — determines, that only the first appearance of `substring` in `string` will be replaced

Return Value: the given string with `substring` replaced by `replacement`

Related Functions: `subs`, `stringlib::subsop`, `stringlib::pos`, `stringlib::remove`

Details:

- By default, every occurrence of the string `substring` in `string` is replaced by `replacement`. The option *`First`* causes only the first appearance of `substring` to be replaced.
- The result is not searched again for instances of `substring`. See Example 3.
- Among several overlapping occurrences of `substring`, the leftmost one is replaced.

Example 1. The string `replacement` may be empty.

```
>> stringlib::subs("abcdeabcdeabcde", "bc" = "")

                        "adeadeade"
```

Example 2. Every `substring` is replaced unless the option *`First`* is given.

```
>> stringlib::subs("abcdeabcdeabcde", "bc" = "xxx")

                   "axxxdeaxxxdeaxxxde"

>> stringlib::subs("abcdeabcdeabcde", "bc" = "xxx", First)

                    "axxxdeabcdeabcde"
```

Example 3. The substitution may produce a new instance of `substring`, but this one is not replaced.

```
>> stringlib::subs("aab", "ab"="b")

                                    "ab"
```

Changes:

- `stringlib::subs` used to be `string::subs`.

`stringlib::subsop` – Substitution in a string

`stringlib::subsop` removes one or more characters at a given position and inserts another substring at that position instead.

Call(s):

- `stringlib::subsop(string, index = replacement)`

Parameters:

`string`	—	non empty string
`index`	—	integer or range of integers that determines the chars to be replaced
`replacement`	—	any string to replace the given char or range

Return Value: the given string with the replacement

Related Functions: `subsop`, `stringlib::pos`, `stringlib::remove`, `stringlib::subs`

Details:

- The char with index `index` in `string` (if `index` is an integer) or the range of chars (if `index` is a range of integers) is removed. Instead `replacement` is inserted at that position. The inserted string need not have the same length.

Example 1. Delete the first character:

```
>> stringlib::subsop("abcdeabcdeabcde", 0 = "")

                        "bcdeabcdeabcde"
```

The 2nd to 3rd character will be replaced by `"xxx"`:

```
>> stringlib::subsop("abcdeabcdeabcde", 1..2 = "xxx")

                       "axxxdeabcdeabcde"
```

Delete the characters 2 to 11:

```
>> stringlib::subsop("abcdeabcdeabcde", 1..10 = "")

                             "abcde"
```

Changes:

- `stringlib::subsop` used to be `string::subsop`.