Inhaltsverzeichnis

Modul:List

Vorlage:Lua

This module outputs various kinds of lists. At present, it supports bulleted lists, unbulleted lists, horizontal lists, ordered lists (numbered or alphabetical), and horizontal ordered lists. It allows for easy css styling of the list or of the individual list items.

Inhaltsverzeichnis

Usage

Quick usage

Vorlage:Pre

All parameters

Vorlage:Pre

Arguments passed from parent template

Vorlage:Pre

Functions

Function name	Produces	Example output	Template using the function
		• First item	

Function name	Produces	Example output	Template using the function
`bulleted`	Bulleted lists	• Second item • Third item	Vorlage:Tlx
`unbulleted`	Unbulleted lists	• First item • Second item • Third item	Vorlage:Tlx
`horizontal`	Horizontal bulleted lists	• First item • Second item • Third item	Vorlage:Tlx
`ordered`	Ordered lists (numbered lists and alphabetical lists)	1. First item 2. Second item 3. Third item	Vorlage:Tlx
`horizontal_ordered`	Horizontal ordered lists	1. First item 2. Second item 3. Third item	Vorlage:Tlx

Parameters

- Positional parameters (`1`, `2`, `3`...) - these are the list items. If no list items are present, the module will output nothing.
- `start` - sets the start item for ordered lists. This can be a start number for numbered lists, or a start letter for alphabetical lists. Horizontal ordered lists only support numbers.
- `type` - the type of marker used in ordered lists. Possible values are "1" for numbers (the default), "A" for uppercase letters, "a" for lowercase letters, "I" for uppercase Roman numerals, and "i" for lowercase Roman numerals. Not supported in horizontal ordered lists. See also the `list_style_type` parameter.
- `list_style_type` - the type of marker used in ordered lists. This uses CSS styling, and has more types available than the `type` parameter, which uses an html attribute. Possible values are listed at MDN's list-style-type page. Support may vary by browser. `list-style-type` is an alias for this parameter.

- `class` - a custom class for the Vorlage:Tag tags surrounding the list, e.g. `plainlinks`.
- `style` - a custom css style for the Vorlage:Tag tags surrounding the list, e.g. `font-size: 90%;`.
- `list_style` - a custom css style for the list itself. The format is the same as for the Vorlage:Para parameter.
- `item_style` - a custom css style for all of the list items (the Vorlage:Tag tags). The format is the same as for the Vorlage:Para parameter.
- `item1_style`, `item2_style`, `item3_style`... - custom css styles for each of the list items. The format is the same as for the Vorlage:Para parameter.
- `item1_value`, `item2_value`, `item3_value`... - custom value for the given list item. List items following the one given will increment from the specified value. The value should be a positive integer. (Note that this option only has an effect on ordered lists.)
- `indent` - this parameter indents the list, for horizontal and horizontal ordered lists only. The value must be a number, e.g. `2`. The indent is calculated in em, and is 1.6 times the value specified. If no indent is specified, the default is zero.

Examples

Bulleted lists

Code	Result
`{{#invoke:list|bulleted|First item|Second item|Third item}}`	• First item • Second item • Third item
`{{#invoke:list|bulleted|First item|Second item|Third item|item_style=color:blue;}}`	• First item • Second item • Third item
`{{#invoke:list|bulleted|First item|Second item|Third`	• First item • Second item

Code	Result
item|item1_style=background-color:yellow;|item2_style=background-color: silver;}}	• Third item

Unbulleted lists

Code	Result
{{#invoke:list|unbulleted|First item|Second item|Third item}}	• First item • Second item • Third item
{{#invoke:list|unbulleted|First item|Second item|Third item|item_style=color:blue;}}	• First item • Second item • Third item
{{#invoke:list|unbulleted|First item|Second item|Third item|item1_style=background-color:yellow;|item2_style=background-color: silver;}}	• First item • Second item • Third item

Horizontal lists

Code	Result
	• First item • Second item

Code	Result
{{#invoke:list|horizontal|First item|Second item|Third item}}	• Third item
{{#invoke:list|horizontal|First item|Second item|Third item|indent=2}}	• First item • Second item • Third item

Ordered lists

Code	Result
{{#invoke:list|ordered|First item|Second item|Third item}}	1. First item 2. Second item 3. Third item
{{#invoke:list|ordered|First item|Second item|Third item|start=3}}	3. First item 4. Second item 5. Third item
{{#invoke:list|ordered|First item|Second item|Third item|type=i}	1. First item 2. Second item 3. Third item
{{#invoke:list|ordered|First item|Second item|Third item|list_style_type=lower-greek}}	1. First item 2. Second item 3. Third item

Horizontal ordered lists

Code	Result
	1. First item

Code	Result
{{#invoke:list|horizontal_ordered|First item|Second item|Third item}}	2. Second item 3. Third item
{{#invoke:list|horizontal_ordered|First item|Second item|Third item|start=3}}	3. First item 4. Second item 5. Third item
{{#invoke:list|horizontal_ordered|First item|Second item|Third item|indent=2}}	1. First item 2. Secon d item 3. Third item

Tracking/maintenance category

- Vorlage:Clc

See also

- Module:Separated entries

```
-- This module outputs different kinds of lists. At the moment, bulleted,
-- unbulleted, horizontal, ordered, and horizontal ordered lists are supported.

local libUtil = require('libraryUtil')
local checkType = libUtil.checkType
local mTableTools = require('Module:TableTools')

local p = {}

local listTypes = {
        ['bulleted'] = true,
        ['unbulleted'] = true,
        ['horizontal'] = true,
        ['ordered'] = true,
        ['horizontal_ordered'] = true
}

function p.makeListData(listType, args)
        -- Constructs a data table to be passed to p.renderList.
        local data = {}

        -- Classes
        data.classes = {}
        if listType == 'horizontal' or listType == 'horizontal_ordered' then
                table.insert(data.classes, 'hlist hlist-separated')
        elseif listType == 'unbulleted' then
                table.insert(data.classes, 'plainlist')
        end
        table.insert(data.classes, args.class)

        -- Main div style
```

```
	data.style = args.style

	-- Indent for horizontal lists
	if listType == 'horizontal' or listType == 'horizontal_ordered' then
		local indent = tonumber(args.indent)
		indent = indent and indent * 1.6 or 0
		if indent > 0 then
			data.marginLeft = indent .. 'em'
		end
	end

	-- List style types for ordered lists
	-- This could be "1, 2, 3", "a, b, c", or a number of others. The list st
	-- type is either set by the "type" attribute or the "list-style-type" CS
	-- property.
	if listType == 'ordered' or listType == 'horizontal_ordered' then
		data.listStyleType = args.list_style_type or args['list-style-typ
		data.type = args['type']

		-- Detect invalid type attributes and attempt to convert them to
		-- list-style-type CSS properties.
		if data.type
			and not data.listStyleType
			and not tostring(data.type):find('^%s*[1AaIi]%s*$')
		then
			data.listStyleType = data.type
			data.type = nil
		end
	end

	-- List tag type
	if listType == 'ordered' or listType == 'horizontal_ordered' then
		data.listTag = 'ol'
	else
		data.listTag = 'ul'
	end

	-- Start number for ordered lists
	data.start = args.start
	if listType == 'horizontal_ordered' then
		-- Apply fix to get start numbers working with horizontal ordered
		local startNum = tonumber(data.start)
		if startNum then
			data.counterReset = 'listitem ' .. tostring(startNum - 1)
		end
	end

	-- List style
	 -- ul_style and ol_style are included for backwards compatibility. No
	 -- distinction is made for ordered or unordered lists.
	data.listStyle = args.list_style

	-- List items
	-- li_style is included for backwards compatibility. item_style was inclu
	-- to be easier to understand for non-coders.
	data.itemStyle = args.item_style or args.li_style
	data.items = {}
	for i, num in ipairs(mTableTools.numKeys(args)) do
		local item = {}
		item.content = args[num]
		item.style = args['item' .. tostring(num) .. '_style']
			or args['item_style' .. tostring(num)]
		item.value = args['item' .. tostring(num) .. '_value']
			or args['item_value' .. tostring(num)]
```

```
                table.insert(data.items, item)
        end

        return data
end

function p.renderList(data)
        -- Renders the list HTML.

        -- Return the blank string if there are no list items.
        if type(data.items) ~= 'table' or #data.items < 1 then
                return ''
        end

        -- Render the main div tag.
        local root = mw.html.create('div')
        for i, class in ipairs(data.classes or {}) do
                root:addClass(class)
        end
        root:css{['margin-left'] = data.marginLeft}
        if data.style then
                root:cssText(data.style)
        end

        -- Render the list tag.
        local list = root:tag(data.listTag or 'ul')
        list
                :attr{start = data.start, type = data.type}
                :css{
                        ['counter-reset'] = data.counterReset,
                        ['list-style-type'] = data.listStyleType
                }
        if data.listStyle then
                list:cssText(data.listStyle)
        end

        -- Render the list items
        for i, t in ipairs(data.items or {}) do
                local item = list:tag('li')
                if data.itemStyle then
                        item:cssText(data.itemStyle)
                end
                if t.style then
                        item:cssText(t.style)
                end
                item
                        :attr{value = t.value}
                        :wikitext(t.content)
        end

        return tostring(root)
end

function p.renderTrackingCategories(args)
        local isDeprecated = false -- Tracks deprecated parameters.
        for k, v in pairs(args) do
                k = tostring(k)
                if k:find('^item_style%d+$') or k:find('^item_value%d+$') then
                        isDeprecated = true
                        break
                end
        end
        local ret = ''
        if isDeprecated then
```

```
                ret = ret .. '[[Category:List templates with deprecated parameter
        end
        return ret
end

function p.makeList(listType, args)
        if not listType or not listTypes[listType] then
                error(string.format(
                        "bad argument #1 to 'makeList' ('%s' is not a valid list
                        tostring(listType)
                ), 2)
        end
        checkType('makeList', 2, args, 'table')
        local data = p.makeListData(listType, args)
        local list = p.renderList(data)
        local trackingCategories = p.renderTrackingCategories(args)
        return list .. trackingCategories
end

for listType in pairs(listTypes) do
        p[listType] = function (frame)
                local mArguments = require('Module:Arguments')
                local origArgs = mArguments.getArgs(frame, {
                        valueFunc = function (key, value)
                        if not value or not mw.ustring.find(value, '%S') then ret
                        if mw.ustring.find(value, '^%s*[%*#;:]') then
                                return value
                        else
                                return value:match('^%s*(.-)%s*$')
                        end
                        return nil
                end
                })
                -- Copy all the arguments to a new table, for faster indexing.
                local args = {}
                for k, v in pairs(origArgs) do
                        args[k] = v
                end
                return p.makeList(listType, args)
        end
end

return p
```