

Inhaltsverzeichnis

Modul:TableTools/Doku

Dies ist die Dokumentationsseite für [Modul:TableTools](#)

This module includes a number of functions for dealing with Lua tables. It is a meta-module, meant to be called from other Lua modules, and should not be called directly from #invoke.

Inhaltsverzeichnis

- [1 Loading the module](#)
- [2 isPositiveInteger](#)
- [3 isNan](#)
- [4 shallowClone](#)
- [5 removeDuplicates](#)
- [6 numKeys](#)
- [7 affixNums](#)
- [8 numData](#)
- [9 compressSparseArray](#)
- [10 sparsePairs](#)
- [11 size](#)
- [12 keysToList](#)
- [13 sortedPairs](#)
- [14 isArray](#)
- [15 isArrayLike](#)
- [16 invert](#)
- [17 listToSet](#)
- [18 deepCopy](#)
- [19 sparseConcat](#)
- [20 length](#)
- [21 inArray](#)

Loading the module

To use any of the functions, first you must load the module.

```
local TableTools = require('Module:TableTools')
```

isPositiveInteger

```
TableTools.isPositiveInteger(value)
```

Returns true if *value* is a positive integer, and false if not. Although it doesn't operate on tables, it is included here as it is useful for determining whether a given table key is in the array part or the hash part of a table.

isNan

`TableTools.isNan(value)`

Returns true if *value* is a [NaN](#) value, and false if not. Although it doesn't operate on tables, it is included here as it is useful for determining whether a value can be a valid table key. (Lua will generate an error if a NaN value is used as a table key.)

shallowClone

`TableTools.shallowClone(t)`

Returns a clone of a table. The value returned is a new table, but all subtables and functions are shared. Metamethods are respected, but the returned table will have no metatable of its own. If you want to make a new table with no shared subtables and with metatables transferred, you can use [mw.clone](#) instead. If you want to make a new table with no shared subtables and without metatables transferred, use [deepCopy](#) with the `noMetatable` option.

removeDuplicates

`TableTools.removeDuplicates(t)`

Removes duplicate values from an array. This function is only designed to work with standard arrays: keys that are not positive integers are ignored, as are all values after the first nil value. (For arrays containing nil values, you can use [compressSparseArray](#) first.) The function tries to preserve the order of the array: the earliest non-unique value is kept, and all subsequent duplicate values are removed. For example, for the table [Vorlage:Code](#) `removeDuplicates` will return [Vorlage:Code](#).

numKeys

`TableTools.numKeys(t)`

Takes a table *t* and returns an array containing the numbers of any positive integer keys that have non-nil values, sorted in numerical order. For example, for the table [Vorlage:Code](#), `numKeys` will return [Vorlage:Code](#).

affixNums

`TableTools.affixNums(t, prefix, suffix)`

Takes a table *t* and returns an array containing the numbers of keys with the optional prefix *prefix* and the optional suffix *suffix*. For example, for the table [Vorlage:Code](#) and the prefix 'a', `affixNums` will return [Vorlage:Code](#). All characters in *prefix* and *suffix* are interpreted literally.

numData

`TableTools.numData(t, compress)`

Given a table with keys like "foo1", "bar1", "foo2", and "baz2", returns a table of subtables in the format [Vorlage:Code](#). Keys that don't end with an integer are stored in a subtable named "other". The compress option compresses the table so that it can be iterated over with `ipairs`.

compressSparseArray

`TableTools.compressSparseArray(t)`

Takes an array `t` with one or more nil values, and removes the nil values while preserving the order, so that the array can be safely traversed with `ipairs`. Any keys that are not positive integers are removed. For example, for the table [Vorlage:Code](#), `compressSparseArray` will return [Vorlage:Code](#).

sparseIpairs

`TableTools.sparseIpairs(t)`

This is an iterator function for traversing a sparse array `t`. It is similar to [ipairs](#), but will continue to iterate until the highest numerical key, whereas `ipairs` may stop after the first nil value. Any keys that are not positive integers are ignored.

Usually `sparseIpairs` is used in a generic for loop.

```
for i, v in TableTools.sparseIpairs(t) do
  -- code block
end
```

Note that `sparseIpairs` uses the [pairs](#) function in its implementation. Although some table keys appear to be ignored, all table keys are accessed when it is run.

size

`TableTools.size(t)`

Finds the size of a key/value pair table. For example, for the table [Vorlage:Code](#), `size` will return 2. The function will also work on arrays, but for arrays it is more efficient to use the `#` operator. Note that to find the table size, this function uses the [pairs](#) function to iterate through all of the table keys.

keysToList

`TableTools.keysToList(t, keySort, checked)`

Returns a list of the keys in a table, sorted using either a default comparison function or a custom `keySort` function, which follows the same rules as the `comp` function supplied to [table.sort](#). If `keySort` is false, no sorting is done. Set `checked` to true to skip the internal type checking.

sortedPairs

`TableTools.sortedPairs(t, keySort)`

Iterates through a table, with the keys sorted using the `keysToList` function. If there are only numerical keys, `sparsePairs` is probably more efficient.

isArray

`TableTools.isArray(value)`

Returns true if *value* is a table and all keys are consecutive integers starting at 1.

isArrayLike

`TableTools.isArrayLike(value)`

Returns true if *value* is iterable and all keys are consecutive integers starting at 1.

invert

`TableTools.invert(arr)`

Transposes the keys and values in an array. For example, [Vorlage:Code](#) yields [Vorlage:Code](#).

listToSet

`TableTools.listToSet(arr)`

Creates a set from the array part of the table *arr*. Indexing the set by any of the values of the array returns true. For example, [Vorlage:Code](#) yields [Vorlage:Code](#). See also [Module:Lua set](#) for more advanced ways to create a set.

deepCopy

`TableTools.deepCopy(orig, noMetatable, alreadySeen)`

Creates a copy of the table *orig*. As with `mw.clone`, all values that are not functions are duplicated and the identity of tables is preserved. If *noMetatable* is true, then the metatable (if any) is not copied. Can copy tables loaded with `mw.loadData`.

Similar to `mw.clone`, but `mw.clone` cannot copy tables loaded with `mw.loadData` and does not allow metatables *not* to be copied.

sparseConcat

`TableTools.sparseConcat(t, sep, i, j)`

Concatenates all values in the table that are indexed by a positive integer, in order. For example, [Vorlage:Code](#) yields [Vorlage:Code](#) and [Vorlage:Code](#) yields [Vorlage:Code](#).

length

TableTools.length(t, prefix)

Finds the length of an array or of a quasi-array with keys with an optional *prefix* such as "data1", "data2", etc. It uses an [exponential search](#) algorithm to find the length, so as to use as few table lookups as possible.

This algorithm is useful for arrays that use metatables (e.g. [frame.args](#)) and for quasi-arrays. For normal arrays, just use the [# operator](#), as it is implemented in [C](#) and will be quicker.

inArray

TableTools.inArray(arr, valueToFind)

Returns true if *valueToFind* is a member of the array *arr*, and false otherwise.