UNDERSCORE.JS

Collection Functions (Arrays or Objects)

_.each(list, iterator, [context]) Alias: forEach Iterates over a list of elements, yielding each in turn to an iterator function. The iterator is b
_.map(list, iterator, [context]) Alias: collect Produces a new array of values by mapping each value in list through a transformation funct
_.reduce(list, iterator, memo, [context]) Aliases: inject, foldl Also known as inject and foldl, reduce boils down a list of values into a singl
_.reduceRight(list, iterator, memo, [context]) Alias: foldr The right-associative version of reduce. Delegates to the JavaScript 1.8 versior
_.find(list, iterator, [context]) Alias: detect Looks through each value in the list, returning the first one that passes a truth test (iterator). T
_.filter(list, iterator, [context]) Alias: select Looks through each value in the list, returning an array of all the values that pass a truth test
_.where(list, properties) Looks through each value in the list, returning an array of all the values that contain all of the key-value pairs list
_.findWhere(list, properties) Looks through the list and returns the first value that matches all of the key-value pairs listed in properties.
_.reject(list, iterator, [context]) Returns the values in list without the elements that the truth test (iterator) passes. The opposite of filter.
_.every(list, iterator, [context]) Alias: all Returns true if all of the values in the list pass the iterator truth test. Delegates to the native met
_.some(list, [iterator], [context]) Alias: any Returns true if any of the values in the list pass the iterator truth test. Short-circuits and stops
_.contains(list, value) Alias: include Returns true if the value is present in the list. Uses indexOf internally, if list is an Array.
_.invoke(list, methodName, [*arguments]) Calls the method named by methodName on each value in the list. Any extra arguments passe
_.pluck(list, propertyName) A convenient version of what is perhaps the most common use-case for map: extracting a list of property va
_.max(list, [iterator], [context]) Returns the maximum value in list. If iterator is passed, it will be used on each value to generate the criter
_.min(list, [iterator], [context]) Returns the minimum value in list. If iterator is passed, it will be used on each value to generate the criteric
_.sortBy(list, iterator, [context]) Returns a sorted copy of list, ranked in ascending order by the results of running each value through iter
_.groupBy(list, iterator, [context]) Splits a collection into sets, grouped by the result of running each value through iterator. If iterator is a
_.countBy(list, iterator, [context]) Sorts a list into groups and returns a count for the number of objects in each group. Similar to groupBy
_.shuffle(list) Returns a shuffled copy of the list, using a version of the Fisher-Yates shuffle.
_.toArray(list) Converts the list (anything that can be iterated over), into a real Array. Useful for transmuting the arguments object.
_.size(list) Return the number of values in the list.

Array Functions

_.first(array, [n]) Alias: head, take Returns the first element of an array. Passing n will return the first n elements of the array.
_.initial(array, [n]) Returns everything but the last entry of the array. Especially useful on the arguments object. Pass n to exclude the last
_.last(array, [n]) Returns the last element of an array. Passing n will return the last n elements of the array.
_.rest(array, [index]) Alias: tail, drop Returns the rest of the elements in an array. Pass an index to return the values of the array from tha
_.compact(array) Returns a copy of the array with all falsy values removed. In JavaScript, false, null, 0, "", undefined and NaN are all fals
_.flatten(array, [shallow]) Flattens a nested array (the nesting can be to any depth). If you pass shallow, the array will only be flattened a
_.without(array, [*values]) Returns a copy of the array with all instances of the values removed.
_.union(*arrays) Computes the union of the passed-in arrays: the list of unique items, in order, that are present in one or more of the arra
_.intersection(*arrays) Computes the list of values that are the intersection of all the arrays. Each value in the result is present in each of
_.difference(array, *others) Similar to without, but returns the values from array that are not present in the other arrays.
_.uniq(array, [isSorted], [iterator]) Alias: unique Produces a duplicate-free version of the array, using === to test object equality. If you k
_.zip(*arrays) Merges together the values of each of the arrays with the values at the corresponding position. Useful when you have sepa
_.object(list, [values]) Converts arrays into objects. Pass either a single list of [key, value] pairs, or a list of keys, and a list of values.
_.indexOf(array, value, [isSorted]) Returns the index at which value can be found in the array, or -1 if value is not present in the array. Us
_.lastIndexOf(array, value, [fromIndex]) Returns the index of the last occurrence of value in the array, or -1 if value is not present. Uses th
_.sortedIndex(list, value, [iterator], [context]) Uses a binary search to determine the index at which the value should be inserted into the l
_.range([start], stop, [step]) A function to create flexibly-numbered lists of integers, handy for each and map loops. start, if omitted, defa

Function (uh, ahem) Functions

_.bind(function, object, [*arguments]) Bind a function to an object, meaning that whenever the function is called, the value of this will be
_.bindAll(object, [*methodNames]) Binds a number of methods on the object, specified by methodNames, to be run in the context of tha
_.partial(function, [*arguments]) Partially apply a function by filling in any number of its arguments, without changing its dynamic this valu
_.memoize(function, [hashFunction]) Memoizes a given function by caching the computed result. Useful for speeding up slow-running c
_.delay(function, wait, [*arguments]) Much like setTimeout, invokes function after wait milliseconds. If you pass the optional arguments,
_.defer(function, [*arguments]) Defers invoking the function until the current call stack has cleared, similar to using setTimeout with a del
_.throttle(function, wait) Creates and returns a new, throttled version of the passed function, that, when invoked repeatedly, will only act
_.debounce(function, wait, [immediate]) Creates and returns a new debounced version of the passed function that will postpone its exec
_.once(function) Creates a version of the function that can only be called one time. Repeated calls to the modified function will have no
_.after(count, function) Creates a version of the function that will only be run after first being called count times. Useful for grouping asyn
_.wrap(function, wrapper) Wraps the first function inside of the wrapper function, passing it as the first argument. This allows the wrappe
_.compose(*functions) Returns the composition of a list of functions, where each function consumes the return value of the function tha

Object Functions

_.keys(object) Retrieve all the names of the object's properties.
_.values(object) Return all of the values of the object's properties.
_.pairs(object) Convert an object into a list of [key, value] pairs.
_.invert(object) Returns a copy of the object where the keys have become the values and the values the keys. For this to work, all of you
_.functions(object) Alias: methods Returns a sorted list of the names of every method in an object — that is to say, the name of every f
_.extend(destination, *sources) Copy all of the properties in the source objects over to the destination object, and return the destination
_.pick(object, *keys) Return a copy of the object, filtered to only have values for the whitelisted keys (or array of valid keys).
_.omit(object, *keys) Return a copy of the object, filtered to omit the blacklisted keys (or array of keys).
_.defaults(object, *defaults) Fill in null and undefined properties in object with values from the defaults objects, and return the object. As
_.clone(object) Create a shallow-copied clone of the object. Any nested objects or arrays will be copied by reference, not duplicated.
_.tap(object, interceptor) Invokes interceptor with the object, and then returns object. The primary purpose of this method is to "tap into
_.has(object, key) Does the object contain the given key? Identical to object.hasOwnProperty(key), but uses a safe reference to the has
_.isEqual(object, other) Performs an optimized deep comparison between the two objects, to determine if they should be considered eq
_.isEmpty(object) Returns true if object contains no values.
_.isElement(object) Returns true if object is a DOM element.
_.isArray(object) Returns true if object is an Array.
_.isObject(value) Returns true if value is an Object. Note that JavaScript arrays and functions are objects, while (normal) strings and num
_.isArguments(object) Returns true if object is an Arguments object.
_.isFunction(object) Returns true if object is a Function.
_.isString(object) Returns true if object is a String.
_.isNumber(object) Returns true if object is a Number (including NaN).
_.isFinite(object) Returns true if object is a finite Number.
_.isBoolean(object) Returns true if object is either true or false.
_.isDate(object) Returns true if object is a Date.
_.isRegExp(object) Returns true if object is a RegExp.
_.isNaN(object) Returns true if object is NaN. Note: this is not the same as the native isNaN function, which will also return true if the var
_.isNull(object) Returns true if the value of object is null.
_.isUndefined(value) Returns true if value is undefined.

Utility Functions

.noConflict() Give control of the "" variable back to its previous owner. Returns a reference to the Underscore object.
_.identity(value) Returns the same value that is used as the argument. In math: f(x) = x This function looks useless, but is used throughou
_.times(n, iterator, [context]) Invokes the given iterator function n times. Each invocation of iterator is called with an index argument. Not
_.random(min, max) Returns a random integer between min and max, inclusive. If you only pass one argument, it will return a number be
_.mixin(object) Allows you to extend Underscore with your own utility functions. Pass a hash of {name: function} definitions to have your
_.uniqueId([prefix]) Generate a globally-unique id for client-side models or DOM elements that need one. If prefix is passed, the id will be
_.escape(string) Escapes a string for insertion into HTML, replacing &, <, >, ", ', and / characters.
_.unescape(string) The opposite of *escape*, replaces &, <, >, ", ', and / with their unescaped counterparts.
_.result(object, property) If the value of the named property is a function then invoke it; otherwise, return it.
_.template(templateString, [data], [settings]) Compiles JavaScript templates into functions that can be evaluated for rendering. Useful fo

Chaining

_.chain(obj) Returns a wrapped object. Calling methods on this object will continue to return wrapped objects until value is used.
_(obj).value() Extracts the value of a wrapped object.