Collection Functions

each (list, iter, [con]) *forEach*
iter: function(element, index, list)

map (list, iter, [con])
```
_.map([1, 2, 3], function(x){return x*2;})
⇒ [2, 4, 6]
```

reduce (list, iter, m, [con]) *inject, foldl*
iter: function(memo, el), *m:* memo
```
_.reduce([2, 3], function(m, i) {return m+i;}, 0) ⇒ 6
```

reduceRight (list, iter, m, [con]) *foldr*
Similar to *reduce*, but works in opposite direction.

detect (list, iter, [con])
Returns the first found value that passes a truth test (*iter*).

select (list, iter, [con]) *filter*
```
_.select([1, 2, 3], function(x) {return x<3;} )
⇒ [1, 2]
```

reject (list, iter, [con])
Opposite of select.

all (list, iter, [con]) *every*
Returns true if all of the values in the list pass the *iter* truth test.

any (list, iter, [con]) *some*
Returns true if any of the values in the list pass the *iter* truth test.

contains (list, value) *include*
Returns true if the value is present in the list. *===*

invoke (list, methodName, [*args])
Calls the method named by *methodName* on each value in the list with passed arguments (if any).

pluck (list, propertyName)
Extracting a list of property values.
```
_.pluck([{k: 1}, {k: 2}], 'k') ⇒ [1, 2]
```

max (list, [iter], [con])

min (list, [iter], [con])

sortBy (list, iter, [con])
Returns a sorted copy of list, ranked by the results of running each value through iterator.

groupBy (list, iter, [con])
Splits a collection into sets, grouped by the result of *iter*.
```
_.groupBy([1.3, 2.1, 2.4], function(x) { return Math.floor(x); } ) ⇒ { 1:
[1.3], 2: [2.1, 2.4] }
```

toArray (list) **size** (list) **shuffle** (list)

Array Functions

(will also work on the arguments object)

first (array, [n]) *head*
Returns first (first *n*) element(s) of an array.

initial (array, [n])
Returns a copy of an array excluding last (last *n*) element(s).

last (array, [n])
Returns last (last *n*) element(s) from an array.

rest (array, [n]) *tail*
Returns a copy of an array excluding first (first *n*) element(s).

compact (array)
Returns a copy of the array with all falsy *(0, false, null, undefined, "", NaN)* values removed.

flatten (array)
Flattens a nested array.
```
_.flatten([1, 2, [[3], 4]]) ⇒ [1, 2, 3, 4]
```

without (array, [*values])
Copy of the array with all passed values removed. *===*

union ([*arrays])

intersection ([*arrays])

difference (array, other)

unique (array, [isSorted], [iter]) *uniq*
Produces a duplicate-free version of the array. *===*

indexOf (array, value, [isSorted])
Returns the index at which value can be found in the array, or -1 if value is not present.

lastIndexOf (array, value)
Returns the index of the last occurrence of value in the array, or -1 if value is not present.

zip ([*arrays])
Merges together the values of each of the arrays with the values at the corresponding position.
```
_.zip( ['a', 'b', 'c'], [1, 2, 3], ['x', 'y', 'z'] )
⇒ [ ['a', 1, 'x'], ['b', 2, 'y'], ['c', 3, 'z'] ]
```

range ([start], stop, [step])
Returns a list of integers from *start* to *stop*, incremented (or decremented) by *step*, exclusive.
```
_.range(10)
⇒ [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
_.range(1, 11)
⇒ [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
_.range(0, 30, 5)
⇒ [0, 5, 10, 15, 20, 25]
_.range(0, -10, -1)
⇒ [0, -1, -2, -3, -4, -5, -6, -7, -8, -9]
_.range(0)
⇒ []
```

Function :-) Functions

bind (func, obj, [*args])
Bind a function to an object, meaning that whenever the function is called, the value of this will be the object. Optionally, bind arguments to the function to pre-fill them, also known as currying.

bindAll (func, [*methodNames])
Binds a number of methods on the object, specified by *methodNames*, to be run in the context of that object whenever they are invoked. If no methodNames are provided, all of the object's function properties will be bound to it.

memoize (func, [hashFunction])
Memoizes a given function by caching the computed result. If passed an optional *hashFunction*, it will be used to compute the hash key for storing the result, based on the arguments to the original function. The default hashFunction just uses the first argument to the memoized function as the key.

delay (func, wait, [*args])

defer (func)
Defers invoking the function until the current call stack has cleared, similar to using setTimeout with a delay of 0.

throttle (func, wait)
Returns a throttled version of the function, that, when invoked repeatedly, will only actually call the wrapped function at most once per every *wait* milliseconds.

debounce (func, wait)
Repeated calls to a debounced function will postpone it's execution until after *wait* milliseconds have elapsed.

once (func)
Creates a version of the function that can only be called one time. Repeated calls to the modified function will have no effect, returning the value from the original call.

after (count, func)
Creates a version of the function that will only be run after first being called *count* times.

wrap (func, wrapper)
Wraps the first function inside of the *wrapper* function, passing it as the first argument.

compose (*functions)
Returns the composition of a list of functions, where each function consumes the return value of the function that follows. In math terms, composing the functions *f()*, *g()*, and *h()* produces *f(g(h()))*.

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.

functions (object) *methods*
Returns a sorted list of the names of every method in an *object.*

extend (destination, *sources)
Copy all of the properties in the *source* objects over to the *destination* object.

defaults (object, *defaults)
Fill in missing properties in object with default values from the defaults objects. As soon as the property is filled, further defaults will have no effect.

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" a method chain, in order to perform operations on intermediate results within the chain.
```
_([1,2,3,200]).chain().
select(function(x) { return x%2 == 0; }).
tap(console.log).
map(function(x) { return x*x }).
value();
⇒ [2, 200]
⇒ [4, 40000]
```

isEqual (object, other)
Performs an optimized deep comparison between the two objects, to determine if they should be considered equal.

isEmpty (object)
Returns true if object contains no values.
```
_.isEmpty({}) ⇒ true
```

isElement (object)
Returns true if object is a DOM element.

isArray (object) **isArguments** (object)

isFunction (object) **isRegExp** (object)

isString (object) **isNumber** (object)
isBoolean (object) **isDate** (object)

isNull (object) **isUndefined** (object)
isUndefined (object)

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. Used as default iterator.

mixin (object)
Allows you to extend Underscore with your own utility functions.

uniqueId ([prefix])
Generate a globally-unique id for client-side models or DOM elements that need one.

template (templateString, [con])
Compiles JavaScript templates into functions that can be evaluated for rendering.

Chaining

chain ()
Returns a wrapped object. Calling methods on this object will continue to return wrapped objects until value is used.
```
var l = [{n : 'sam', age : 25}, {n : 'moe', age : 21}];
var y = _(list).chain()
.sortBy(function(s){ return s.age; })
.map(function(s){ return s.n + ' is ' + s.age; })
.first()
.value();
⇒ "moe is 21"
```

value ()
Extracts the value of a wrapped object.
```
_(obj).value()
```

Underscore.js Cheatsheet

http://documentcloud.github.com/underscore/

aveic@mail.ru

Example

example (arguments) *alias*
con: context forced for an iterator
```
some_code_examples(); _.size([1, 1]) ⇒ 2
```
A bit of description. *=== **
** === is used for test equality*