Underscore.js

Prof. Dr. J. Ashkenas

04-04-2013

Contents

```
Underscore.js 1.4.4
http://underscorejs.org
(c) 2009-2013 Jeremy Ashkenas, DocumentCloud Inc.
Underscore may be freely distributed under the MIT license.

(function() {
```

Baseline setup

Establish the root object, `window` in the browser, or `global` on the server.

```
  var root = this;
```

Save the previous value of the `_` variable.

```
  var previousUnderscore = root._;
```

Establish the object that gets returned to break out of a loop iteration.

```
  var breaker = {};
```

Save bytes in the minified (but not gzipped) version:

```
  var ArrayProto = Array.prototype, ObjProto = Object.prototype, FuncProto = Function.p
```

Create quick reference variables for speed access to core prototypes.

```
  var push             = ArrayProto.push,
      slice            = ArrayProto.slice,
      concat           = ArrayProto.concat,
      toString         = ObjProto.toString,
      hasOwnProperty   = ObjProto.hasOwnProperty;
```

All **ECMAScript 5** native function implementations that we hope to use are declared here.

```
  var
    nativeForEach      = ArrayProto.forEach,
    nativeMap          = ArrayProto.map,
    nativeReduce       = ArrayProto.reduce,
    nativeReduceRight  = ArrayProto.reduceRight,
    nativeFilter       = ArrayProto.filter,
    nativeEvery        = ArrayProto.every,
    nativeSome         = ArrayProto.some,
    nativeIndexOf      = ArrayProto.indexOf,
    nativeLastIndexOf  = ArrayProto.lastIndexOf,
    nativeIsArray      = Array.isArray,
    nativeKeys         = Object.keys,
    nativeBind         = FuncProto.bind;
```

Create a safe reference to the Underscore object for use below.

```
  var _ = function(obj) {
    if (obj instanceof _) return obj;
    if (!(this instanceof _)) return new _(obj);
    this._wrapped = obj;
  };
```

Export the Underscore object for **Node.js**, with backwards-compatibility for the old `require()` API. If we're in the browser, add `_` as a global object via a string identifier, for Closure Compiler "advanced" mode.

```
if (typeof exports !== 'undefined') {
  if (typeof module !== 'undefined' && module.exports) {
    exports = module.exports = _;
  }
  exports._ = _;
} else {
  root._ = _;
}
```

Current version.

```
_.VERSION = '1.4.4';
```

Collection Functions

The cornerstone, an `each` implementation, aka `forEach`. Handles objects with the built-in `forEach`, arrays, and raw objects. Delegates to **ECMAScript 5**'s native `forEach` if available.

```
var each = _.each = _.forEach = function(obj, iterator, context) {
  if (obj == null) return;
  if (nativeForEach && obj.forEach === nativeForEach) {
    obj.forEach(iterator, context);
  } else if (obj.length === +obj.length) {
    for (var i = 0, l = obj.length; i < l; i++) {
      if (iterator.call(context, obj[i], i, obj) === breaker) return;
    }
  } else {
    for (var key in obj) {
      if (_.has(obj, key)) {
        if (iterator.call(context, obj[key], key, obj) === breaker) return;
      }
    }
  }
};
```

Return the results of applying the iterator to each element. Delegates to **ECMAScript 5**'s native `map` if available.

```
  _.map = _.collect = function(obj, iterator, context) {
    var results = [];
    if (obj == null) return results;
    if (nativeMap && obj.map === nativeMap) return obj.map(iterator, context);
    each(obj, function(value, index, list) {
      results.push(iterator.call(context, value, index, list));
    });
    return results;
  };

  var reduceError = 'Reduce of empty array with no initial value';
```

Reduce builds up a single result from a list of values, aka `inject`, or `foldl`. Delegates to **ECMAScript 5**'s native `reduce` if available.

```
  _.reduce = _.foldl = _.inject = function(obj, iterator, memo, context) {
    var initial = arguments.length > 2;
    if (obj == null) obj = [];
    if (nativeReduce && obj.reduce === nativeReduce) {
      if (context) iterator = _.bind(iterator, context);
      return initial ? obj.reduce(iterator, memo) : obj.reduce(iterator);
    }
    each(obj, function(value, index, list) {
      if (!initial) {
        memo = value;
        initial = true;
      } else {
        memo = iterator.call(context, memo, value, index, list);
      }
    });
    if (!initial) throw new TypeError(reduceError);
    return memo;
  };
```

The right-associative version of reduce, also known as `foldr`. Delegates to **ECMAScript 5**'s native `reduceRight` if available.

```
  _.reduceRight = _.foldr = function(obj, iterator, memo, context) {
    var initial = arguments.length > 2;
    if (obj == null) obj = [];
    if (nativeReduceRight && obj.reduceRight === nativeReduceRight) {
      if (context) iterator = _.bind(iterator, context);
      return initial ? obj.reduceRight(iterator, memo) : obj.reduceRight(iterator);
    }
    var length = obj.length;
    if (length !== +length) {
      var keys = _.keys(obj);
      length = keys.length;
    }
    each(obj, function(value, index, list) {
      index = keys ? keys[--length] : --length;
      if (!initial) {
        memo = obj[index];
        initial = true;
      } else {
        memo = iterator.call(context, memo, obj[index], index, list);
      }
    });
    if (!initial) throw new TypeError(reduceError);
    return memo;
  };
```

Return the first value which passes a truth test. Aliased as `detect`.

```
  _.find = _.detect = function(obj, iterator, context) {
    var result;
    any(obj, function(value, index, list) {
      if (iterator.call(context, value, index, list)) {
        result = value;
        return true;
      }
    });
    return result;
  };
```

Return all the elements that pass a truth test. Delegates to **ECMAScript 5**'s native `filter` if available. Aliased as `select`.

```
  _.filter = _.select = function(obj, iterator, context) {
    var results = [];
    if (obj == null) return results;
    if (nativeFilter && obj.filter === nativeFilter) return obj.filter(iterator, context);
    each(obj, function(value, index, list) {
      if (iterator.call(context, value, index, list)) results.push(value);
    });
    return results;
  };
```

Return all the elements for which a truth test fails.

```
  _.reject = function(obj, iterator, context) {
    return _.filter(obj, function(value, index, list) {
      return !iterator.call(context, value, index, list);
    }, context);
  };
```

Determine whether all of the elements match a truth test. Delegates to **ECMAScript 5**'s native `every` if available. Aliased as `all`.

```
  _.every = _.all = function(obj, iterator, context) {
    iterator || (iterator = _.identity);
    var result = true;
    if (obj == null) return result;
    if (nativeEvery && obj.every === nativeEvery) return obj.every(iterator, context);
    each(obj, function(value, index, list) {
      if (!(result = result && iterator.call(context, value, index, list))) return brea
    });
    return !!result;
  };
```

Determine if at least one element in the object matches a truth test. Delegates to **ECMAScript 5**'s native `some` if available. Aliased as `any`.

```
  var any = _.some = _.any = function(obj, iterator, context) {
    iterator || (iterator = _.identity);
    var result = false;
    if (obj == null) return result;
    if (nativeSome && obj.some === nativeSome) return obj.some(iterator, context);
    each(obj, function(value, index, list) {
      if (result || (result = iterator.call(context, value, index, list))) return break
    });
    return !!result;
  };
```

Determine if the array or object contains a given value (using `===`). Aliased as `include`.

```
  _.contains = _.include = function(obj, target) {
    if (obj == null) return false;
    if (nativeIndexOf && obj.indexOf === nativeIndexOf) return obj.indexOf(target) != -1;
    return any(obj, function(value) {
      return value === target;
    });
  };
```

Invoke a method (with arguments) on every item in a collection.

```
  _.invoke = function(obj, method) {
    var args = slice.call(arguments, 2);
    var isFunc = _.isFunction(method);
    return _.map(obj, function(value) {
      return (isFunc ? method : value[method]).apply(value, args);
    });
  };
```

Convenience version of a common use case of `map`: fetching a property.

```
  _.pluck = function(obj, key) {
    return _.map(obj, function(value){ return value[key]; });
  };
```

Convenience version of a common use case of `filter`: selecting only objects containing specific `key:value` pairs.

```
  _.where = function(obj, attrs, first) {
    if (_.isEmpty(attrs)) return first ? void 0 : [];
    return _[first ? 'find' : 'filter'](obj, function(value) {
      for (var key in attrs) {
        if (attrs[key] !== value[key]) return false;
      }
      return true;
    });
  };
```

Convenience version of a common use case of `find`: getting the first object containing specific `key:value` pairs.

```
  _.findWhere = function(obj, attrs) {
    return _.where(obj, attrs, true);
  };
```

Return the maximum element or (element-based computation). Can't optimize arrays of integers longer than 65,535 elements. See: https://bugs.webkit.org/show_bug.cgi?id=80797

```
  _.max = function(obj, iterator, context) {
    if (!iterator && _.isArray(obj) && obj[0] === +obj[0] && obj.length < 65535) {
      return Math.max.apply(Math, obj);
    }
    if (!iterator && _.isEmpty(obj)) return -Infinity;
    var result = {computed : -Infinity, value: -Infinity};
    each(obj, function(value, index, list) {
      var computed = iterator ? iterator.call(context, value, index, list) : value;
      computed >= result.computed && (result = {value : value, computed : computed});
    });
    return result.value;
  };
```

Return the minimum element (or element-based computation).

```
  _.min = function(obj, iterator, context) {
    if (!iterator && _.isArray(obj) && obj[0] === +obj[0] && obj.length < 65535) {
      return Math.min.apply(Math, obj);
    }
    if (!iterator && _.isEmpty(obj)) return Infinity;
    var result = {computed : Infinity, value: Infinity};
    each(obj, function(value, index, list) {
      var computed = iterator ? iterator.call(context, value, index, list) : value;
      computed < result.computed && (result = {value : value, computed : computed});
    });
    return result.value;
  };
```

Shuffle an array.

```
  _.shuffle = function(obj) {
    var rand;
    var index = 0;
    var shuffled = [];
    each(obj, function(value) {
      rand = _.random(index++);
      shuffled[index - 1] = shuffled[rand];
      shuffled[rand] = value;
    });
    return shuffled;
  };
```

An internal function to generate lookup iterators.

```
  var lookupIterator = function(value) {
    return _.isFunction(value) ? value : function(obj){ return obj[value]; };
  };
```

Sort the object's values by a criterion produced by an iterator.

```
  _.sortBy = function(obj, value, context) {
    var iterator = lookupIterator(value);
    return _.pluck(_.map(obj, function(value, index, list) {
      return {
        value : value,
        index : index,
        criteria : iterator.call(context, value, index, list)
      };
    }).sort(function(left, right) {
      var a = left.criteria;
      var b = right.criteria;
      if (a !== b) {
        if (a > b || a === void 0) return 1;
        if (a < b || b === void 0) return -1;
      }
      return left.index < right.index ? -1 : 1;
    }), 'value');
  };
```

An internal function used for aggregate "group by" operations.

```
  var group = function(obj, value, context, behavior) {
    var result = {};
    var iterator = lookupIterator(value == null ? _.identity : value);
    each(obj, function(value, index) {
      var key = iterator.call(context, value, index, obj);
      behavior(result, key, value);
    });
    return result;
  };
```

Groups the object's values by a criterion. Pass either a string attribute to group by, or a function that returns the criterion.

```
  _.groupBy = function(obj, value, context) {
    return group(obj, value, context, function(result, key, value) {
      (_.has(result, key) ? result[key] : (result[key] = [])).push(value);
    });
  };
```

Counts instances of an object that group by a certain criterion. Pass either a string attribute to count by, or a function that returns the criterion.

```
  _.countBy = function(obj, value, context) {
    return group(obj, value, context, function(result, key) {
      if (!_.has(result, key)) result[key] = 0;
      result[key]++;
    });
  };
```

Use a comparator function to figure out the smallest index at which an object should be inserted so as to maintain order. Uses binary search.

```
  _.sortedIndex = function(array, obj, iterator, context) {
    iterator = iterator == null ? _.identity : lookupIterator(iterator);
    var value = iterator.call(context, obj);
    var low = 0, high = array.length;
    while (low < high) {
      var mid = (low + high) >>> 1;
      iterator.call(context, array[mid]) < value ? low = mid + 1 : high = mid;
    }
    return low;
  };
```

Safely convert anything iterable into a real, live array.

```
_.toArray = function(obj) {
  if (!obj) return [];
  if (_.isArray(obj)) return slice.call(obj);
  if (obj.length === +obj.length) return _.map(obj, _.identity);
  return _.values(obj);
};
```

Return the number of elements in an object.

```
_.size = function(obj) {
  if (obj == null) return 0;
  return (obj.length === +obj.length) ? obj.length : _.keys(obj).length;
};
```

Array Functions

Get the first element of an array. Passing **n** will return the first N values in the array. Aliased as `head` and `take`. The **guard** check allows it to work with `_.map`.

```
_.first = _.head = _.take = function(array, n, guard) {
  if (array == null) return void 0;
  return (n != null) && !guard ? slice.call(array, 0, n) : array[0];
};
```

Returns everything but the last entry of the array. Especially useful on the arguments object. Passing **n** will return all the values in the array, excluding the last N. The **guard** check allows it to work with `_.map`.

```
_.initial = function(array, n, guard) {
  return slice.call(array, 0, array.length - ((n == null) || guard ? 1 : n));
};
```

Get the last element of an array. Passing **n** will return the last N values in the array. The **guard** check allows it to work with `_.map`.

```
_.last = function(array, n, guard) {
  if (array == null) return void 0;
  if ((n != null) && !guard) {
    return slice.call(array, Math.max(array.length - n, 0));
  } else {
    return array[array.length - 1];
  }
};
```

Returns everything but the first entry of the array. Aliased as `tail` and `drop`. Especially useful on the arguments object. Passing an **n** will return the rest N values in the array. The **guard** check allows it to work with `_.map`.

```
_.rest = _.tail = _.drop = function(array, n, guard) {
  return slice.call(array, (n == null) || guard ? 1 : n);
};
```

Trim out all falsy values from an array.

```
_.compact = function(array) {
  return _.filter(array, _.identity);
};
```

Internal implementation of a recursive `flatten` function.

```
var flatten = function(input, shallow, output) {
  each(input, function(value) {
    if (_.isArray(value)) {
      shallow ? push.apply(output, value) : flatten(value, shallow, output);
    } else {
      output.push(value);
    }
  });
  return output;
};
```

Return a completely flattened version of an array.

```
_.flatten = function(array, shallow) {
  return flatten(array, shallow, []);
};
```

Return a version of the array that does not contain the specified value(s).

```
  _.without = function(array) {
    return _.difference(array, slice.call(arguments, 1));
  };
```

Produce a duplicate-free version of the array. If the array has already been sorted, you have the option of using a faster algorithm. Aliased as `unique`.

```
  _.uniq = _.unique = function(array, isSorted, iterator, context) {
    if (_.isFunction(isSorted)) {
      context = iterator;
      iterator = isSorted;
      isSorted = false;
    }
    var initial = iterator ? _.map(array, iterator, context) : array;
    var results = [];
    var seen = [];
    each(initial, function(value, index) {
      if (isSorted ? (!index || seen[seen.length - 1] !== value) : !_.contains(seen, value)) {
        seen.push(value);
        results.push(array[index]);
      }
    });
    return results;
  };
```

Produce an array that contains the union: each distinct element from all of the passed-in arrays.

```
  _.union = function() {
    return _.uniq(concat.apply(ArrayProto, arguments));
  };
```

Produce an array that contains every item shared between all the passed-in arrays.

```
  _.intersection = function(array) {
    var rest = slice.call(arguments, 1);
    return _.filter(_.uniq(array), function(item) {
      return _.every(rest, function(other) {
        return _.indexOf(other, item) >= 0;
      });
    });
  };
```

Take the difference between one array and a number of other arrays. Only the elements present in just the first array will remain.

```
  _.difference = function(array) {
    var rest = concat.apply(ArrayProto, slice.call(arguments, 1));
    return _.filter(array, function(value){ return !_.contains(rest, value); });
  };
```

Zip together multiple lists into a single array – elements that share an index go together.

```
  _.zip = function() {
    var args = slice.call(arguments);
    var length = _.max(_.pluck(args, 'length'));
    var results = new Array(length);
    for (var i = 0; i < length; i++) {
      results[i] = _.pluck(args, "" + i);
    }
    return results;
  };
```

The inverse operation to `_.zip`. If given an array of pairs it returns an array of the paired elements split into two left and right element arrays, if given an array of triples it returns a three element array and so on. For example, `_.unzip` given `[['a',1],['b',2],['c',3]]` returns the array [['a','b','c'],[1,2,3]].

```
  _.unzip = function(tuples) {
      var results = [];
      _.each(tuples, function (tuple, tupleIndex) {
          _.each(tuple, function (value, itemIndex) {
              if (results.length <= itemIndex) {
                  results[itemIndex] = [];
              }
              results[itemIndex][tupleIndex] = value;
          });
      });
```

```
    return results;
  };
```

Converts lists into objects. Pass either a single array of `[key, value]` pairs, or two parallel arrays of the same length – one of keys, and one of the corresponding values.

```
  _.object = function(list, values) {
    if (list == null) return {};
    var result = {};
    for (var i = 0, l = list.length; i < l; i++) {
      if (values) {
        result[list[i]] = values[i];
      } else {
        result[list[i][0]] = list[i][1];
      }
    }
    return result;
  };
```

If the browser doesn't supply us with indexOf (I'm looking at you, **MSIE**), we need this function. Return the position of the first occurrence of an item in an array, or -1 if the item is not included in the array. Delegates to **ECMAScript 5**'s native `indexOf` if available. If the array is large and already in sort order, pass `true` for **isSorted** to use binary search.

```
  _.indexOf = function(array, item, isSorted) {
    if (array == null) return -1;
    var i = 0, l = array.length;
    if (isSorted) {
      if (typeof isSorted == 'number') {
        i = (isSorted < 0 ? Math.max(0, l + isSorted) : isSorted);
      } else {
        i = _.sortedIndex(array, item);
        return array[i] === item ? i : -1;
      }
    }
    if (nativeIndexOf && array.indexOf === nativeIndexOf) return array.indexOf(item, isSorted);
    for (; i < l; i++) if (array[i] === item) return i;
    return -1;
  };
```

Delegates to **ECMAScript 5**'s native `lastIndexOf` if available.

```
_.lastIndexOf = function(array, item, from) {
  if (array == null) return -1;
  var hasIndex = from != null;
  if (nativeLastIndexOf && array.lastIndexOf === nativeLastIndexOf) {
    return hasIndex ? array.lastIndexOf(item, from) : array.lastIndexOf(item);
  }
  var i = (hasIndex ? from : array.length);
  while (i--) if (array[i] === item) return i;
  return -1;
};
```

Generate an integer Array containing an arithmetic progression. A port of the native Python `range()` function. See the Python documentation.

```
_.range = function(start, stop, step) {
  if (arguments.length <= 1) {
    stop = start || 0;
    start = 0;
  }
  step = arguments[2] || 1;

  var len = Math.max(Math.ceil((stop - start) / step), 0);
  var idx = 0;
  var range = new Array(len);

  while(idx < len) {
    range[idx++] = start;
    start += step;
  }

  return range;
};
```

Function (ahem) Functions

Reusable constructor function for prototype setting.

```
var ctor = function(){};
```

Create a function bound to a given object (assigning `this`, and arguments, optionally). Delegates to **ECMAScript 5**'s native `Function.bind` if available.

```
  _.bind = function(func, context) {
    var args, bound;
    if (func.bind === nativeBind && nativeBind) return nativeBind.apply(func, slice.call(argument
    if (!_.isFunction(func)) throw new TypeError;
    args = slice.call(arguments, 2);
    return bound = function() {
      if (!(this instanceof bound)) return func.apply(context, args.concat(slice.call(arguments))
      ctor.prototype = func.prototype;
      var self = new ctor;
      ctor.prototype = null;
      var result = func.apply(self, args.concat(slice.call(arguments)));
      if (Object(result) === result) return result;
      return self;
    };
  };
```

Partially apply a function by creating a version that has had some of its arguments pre-filled, without changing its dynamic `this` context.

```
  _.partial = function(func) {
    var args = slice.call(arguments, 1);
    return function() {
      return func.apply(this, args.concat(slice.call(arguments)));
    };
  };
```

Bind all of an object's methods to that object. Useful for ensuring that all callbacks defined on an object belong to it.

```
  _.bindAll = function(obj) {
    var funcs = slice.call(arguments, 1);
    if (funcs.length === 0) throw new Error("bindAll must be passed function names");
    each(funcs, function(f) { obj[f] = _.bind(obj[f], obj); });
    return obj;
  };
```

Memoize an expensive function by storing its results.

```
  _.memoize = function(func, hasher) {
    var memo = {};
    hasher || (hasher = _.identity);
    return function() {
      var key = hasher.apply(this, arguments);
      return _.has(memo, key) ? memo[key] : (memo[key] = func.apply(this, arguments));
    };
  };
```

Delays a function for the given number of milliseconds, and then calls it with the arguments supplied.

```
  _.delay = function(func, wait) {
    var args = slice.call(arguments, 2);
    return setTimeout(function(){ return func.apply(null, args); }, wait);
  };
```

Defers a function, scheduling it to run after the current call stack has cleared.

```
  _.defer = function(func) {
    return _.delay.apply(_, [func, 1].concat(slice.call(arguments, 1)));
  };
```

Returns a function, that, when invoked, will only be triggered at most once during a given window of time.

```
  _.throttle = function(func, wait, immediate) {
    var context, args, timeout, result;
    var previous = 0;
    var later = function() {
      previous = new Date;
      timeout = null;
      result = func.apply(context, args);
    };
    return function() {
      var now = new Date;
      if (!previous && immediate === false) previous = now;
      var remaining = wait - (now - previous);
      context = this;
      args = arguments;
      if (remaining <= 0) {
        clearTimeout(timeout);
        timeout = null;
        previous = now;
        result = func.apply(context, args);
      } else if (!timeout) {
        timeout = setTimeout(later, remaining);
      }
      return result;
    };
  };
```

Returns a function, that, as long as it continues to be invoked, will not be triggered. The function will be called after it stops being called for N milliseconds. If `immediate` is passed, trigger the function on the leading edge, instead of the trailing.

```
  _.debounce = function(func, wait, immediate) {
    var timeout, result;
    return function() {
      var context = this, args = arguments;
      var later = function() {
        timeout = null;
        if (!immediate) result = func.apply(context, args);
      };
      var callNow = immediate && !timeout;
      clearTimeout(timeout);
      timeout = setTimeout(later, wait);
      if (callNow) result = func.apply(context, args);
      return result;
    };
  };
```

Returns a function that will be executed at most one time, no matter how often you call it. Useful for lazy initialization.

```
  _.once = function(func) {
    var ran = false, memo;
    return function() {
      if (ran) return memo;
      ran = true;
      memo = func.apply(this, arguments);
      func = null;
      return memo;
    };
  };
```

Returns the first function passed as an argument to the second, allowing you to adjust arguments, run code before and after, and conditionally execute the original function.

```
  _.wrap = function(func, wrapper) {
    return function() {
      var args = [func];
      push.apply(args, arguments);
      return wrapper.apply(this, args);
    };
  };
```

Returns a function that is the composition of a list of functions, each consuming the return value of the function that follows.

```
  _.compose = function() {
    var funcs = arguments;
    return function() {
      var args = arguments;
      for (var i = funcs.length - 1; i >= 0; i--) {
        args = [funcs[i].apply(this, args)];
      }
      return args[0];
    };
  };
```

Returns a function that will only be executed after being called N times.

```
  _.after = function(times, func) {
    if (times <= 0) return func();
    return function() {
      if (--times < 1) {
        return func.apply(this, arguments);
      }
    };
  };
```

Object Functions

Retrieve the names of an object's properties. Delegates to **ECMAScript 5**'s native `Object.keys`

```
_.keys = nativeKeys || function(obj) {
  if (obj !== Object(obj)) throw new TypeError('Invalid object');
  var keys = [];
  for (var key in obj) if (_.has(obj, key)) keys.push(key);
  return keys;
};
```

Retrieve the values of an object's properties.

```
_.values = function(obj) {
  var values = [];
  for (var key in obj) if (_.has(obj, key)) values.push(obj[key]);
  return values;
};
```

Convert an object into a list of `[key, value]` pairs.

```
_.pairs = function(obj) {
  var pairs = [];
  for (var key in obj) if (_.has(obj, key)) pairs.push([key, obj[key]]);
  return pairs;
};
```

Invert the keys and values of an object. The values must be serializable.

```
_.invert = function(obj) {
  var result = {};
  for (var key in obj) if (_.has(obj, key)) result[obj[key]] = key;
  return result;
};
```

Return a sorted list of the function names available on the object. Aliased as `methods`

```
_.functions = _.methods = function(obj) {
  var names = [];
  for (var key in obj) {
    if (_.isFunction(obj[key])) names.push(key);
  }
  return names.sort();
};
```

Extend a given object with all the properties in passed-in object(s).

```
_.extend = function(obj) {
  each(slice.call(arguments, 1), function(source) {
    if (source) {
      for (var prop in source) {
        obj[prop] = source[prop];
      }
    }
  });
  return obj;
};
```

Return a copy of the object only containing the whitelisted properties.

```
_.pick = function(obj) {
  var copy = {};
  var keys = concat.apply(ArrayProto, slice.call(arguments, 1));
  each(keys, function(key) {
    if (key in obj) copy[key] = obj[key];
  });
  return copy;
};
```

Return a copy of the object without the blacklisted properties.

```
_.omit = function(obj) {
  var copy = {};
  var keys = concat.apply(ArrayProto, slice.call(arguments, 1));
  for (var key in obj) {
    if (!_.contains(keys, key)) copy[key] = obj[key];
  }
  return copy;
};
```

Fill in a given object with default properties.

```
  _.defaults = function(obj) {
    each(slice.call(arguments, 1), function(source) {
      if (source) {
        for (var prop in source) {
          if (obj[prop] === void 0) obj[prop] = source[prop];
        }
      }
    });
    return obj;
  };
```

Create a (shallow-cloned) duplicate of an object.

```
  _.clone = function(obj) {
    if (!_.isObject(obj)) return obj;
    return _.isArray(obj) ? obj.slice() : _.extend({}, obj);
  };
```

Invokes interceptor with the obj, and then returns obj. The primary purpose of this method is to "tap into" a method chain, in order to perform operations on intermediate results within the chain.

```
  _.tap = function(obj, interceptor) {
    interceptor(obj);
    return obj;
  };
```

Internal recursive comparison function for `isEqual`.

```
  var eq = function(a, b, aStack, bStack) {
```

Identical objects are equal. `0 === -0`, but they aren't identical. See the Harmony `egal` proposal: http://wiki.ecmascript.org/doku.php?id=harmony:egal.

```
    if (a === b) return a !== 0 || 1 / a == 1 / b;
```

A strict comparison is necessary because `null == undefined`.

```
    if (a == null || b == null) return a === b;
```

Unwrap any wrapped objects.

```
    if (a instanceof _) a = a._wrapped;
    if (b instanceof _) b = b._wrapped;
```

Compare `[[Class]]` names.

```
    var className = toString.call(a);
    if (className != toString.call(b)) return false;
    switch (className) {
```

Strings, numbers, dates, and booleans are compared by value.

```
      case '[object String]':
```

Primitives and their corresponding object wrappers are equivalent; thus, `"5"` is equivalent to `new String("5")`.

```
        return a == String(b);
      case '[object Number]':
```

`NaN`s are equivalent, but non-reflexive. An `egal` comparison is performed for other numeric values.

```
        return a != +a ? b != +b : (a == 0 ? 1 / a == 1 / b : a == +b);
      case '[object Date]':
      case '[object Boolean]':
```

Coerce dates and booleans to numeric primitive values. Dates are compared by their millisecond representations. Note that invalid dates with millisecond representations of `NaN` are not equivalent.

```
        return +a == +b;
```

RegExps are compared by their source patterns and flags.

```
      case '[object RegExp]':
        return a.source == b.source &&
               a.global == b.global &&
               a.multiline == b.multiline &&
               a.ignoreCase == b.ignoreCase;
    }
    if (typeof a != 'object' || typeof b != 'object') return false;
```

Assume equality for cyclic structures. The algorithm for detecting cyclic structures is adapted from ES 5.1 section 15.12.3, abstract operation `JO`.

```
    var length = aStack.length;
    while (length--) {
```

Linear search. Performance is inversely proportional to the number of unique nested structures.

```
      if (aStack[length] == a) return bStack[length] == b;
    }
```

Add the first object to the stack of traversed objects.

```
    aStack.push(a);
    bStack.push(b);
    var size = 0, result = true;
```

Recursively compare objects and arrays.

```
    if (className == '[object Array]') {
```

Compare array lengths to determine if a deep comparison is necessary.

```
      size = a.length;
      result = size == b.length;
      if (result) {
```

Deep compare the contents, ignoring non-numeric properties.

```
        while (size--) {
          if (!(result = eq(a[size], b[size], aStack, bStack))) break;
        }
      }
    } else {
```

Objects with different constructors are not equivalent, but `Object`s from different frames are.

```
      var aCtor = a.constructor, bCtor = b.constructor;
      if (aCtor !== bCtor && !(_.isFunction(aCtor) && (aCtor instanceof aCtor) &&
                               _.isFunction(bCtor) && (bCtor instanceof bCtor))) {
        return false;
      }
```

Deep compare objects.

```
      for (var key in a) {
        if (_.has(a, key)) {
```

Count the expected number of properties.

```
          size++;
```

Deep compare each member.

```
          if (!(result = _.has(b, key) && eq(a[key], b[key], aStack, bStack))) break;
        }
      }
```

Ensure that both objects contain the same number of properties.

```
      if (result) {
        for (key in b) {
          if (_.has(b, key) && !(size--)) break;
        }
        result = !size;
      }
    }
```

Remove the first object from the stack of traversed objects.

```
    aStack.pop();
    bStack.pop();
    return result;
  };
```

Perform a deep comparison to check if two objects are equal.

```
  _.isEqual = function(a, b) {
    return eq(a, b, [], []);
  };
```

Is a given array, string, or object empty? An “empty” object has no enumerable own-properties.

```
  _.isEmpty = function(obj) {
    if (obj == null) return true;
    if (_.isArray(obj) || _.isString(obj)) return obj.length === 0;
    for (var key in obj) if (_.has(obj, key)) return false;
    return true;
  };
```

Is a given value a DOM element?

```
  _.isElement = function(obj) {
    return !!(obj && obj.nodeType === 1);
  };
```

Is a given value an array? Delegates to ECMA5's native Array.isArray

```
  _.isArray = nativeIsArray || function(obj) {
    return toString.call(obj) == '[object Array]';
  };
```

Is a given variable an object?

```
  _.isObject = function(obj) {
    return obj === Object(obj);
  };
```

Add some isType methods: isArguments, isFunction, isString, isNumber, isDate, isRegExp.

```
  each(['Arguments', 'Function', 'String', 'Number', 'Date', 'RegExp'], function(name) {
    _['is' + name] = function(obj) {
      return toString.call(obj) == '[object ' + name + ']';
    };
  });
```

Define a fallback version of the method in browsers (ahem, IE), where there isn't any inspectable "Arguments" type.

```
  if (!_.isArguments(arguments)) {
    _.isArguments = function(obj) {
      return !!(obj && _.has(obj, 'callee'));
    };
  }
```

Optimize `isFunction` if appropriate.

```
  if (typeof (/./) !== 'function') {
    _.isFunction = function(obj) {
      return typeof obj === 'function';
    };
  }
```

Is a given object a finite number?

```
  _.isFinite = function(obj) {
    return isFinite(obj) && !isNaN(parseFloat(obj));
  };
```

Is the given value `NaN`? (NaN is the only number which does not equal itself).

```
  _.isNaN = function(obj) {
    return _.isNumber(obj) && obj != +obj;
  };
```

Is a given value a boolean?

```
  _.isBoolean = function(obj) {
    return obj === true || obj === false || toString.call(obj) == '[object Boolean]';
  };
```

Is a given value equal to null?

```
  _.isNull = function(obj) {
    return obj === null;
  };
```

Is a given variable undefined?

```
  _.isUndefined = function(obj) {
    return obj === void 0;
  };
```

Shortcut function for checking if an object has a given property directly on itself (in other words, not on a prototype).

```
  _.has = function(obj, key) {
    return hasOwnProperty.call(obj, key);
  };
```

Utility Functions

Run Underscore.js in *noConflict* mode, returning the _ variable to its previous owner. Returns a reference to the Underscore object.

```
  _.noConflict = function() {
    root._ = previousUnderscore;
    return this;
  };
```

Keep the identity function around for default iterators.

```
  _.identity = function(value) {
    return value;
  };
```

Run a function **n** times.

```
  _.times = function(n, iterator, context) {
    var accum = Array(Math.max(0, n));
    for (var i = 0; i < n; i++) accum[i] = iterator.call(context, i);
    return accum;
  };
```

Return a random integer between min and max (inclusive).

```
  _.random = function(min, max) {
    if (max == null) {
      max = min;
      min = 0;
    }
    return min + Math.floor(Math.random() * (max - min + 1));
  };
```

List of HTML entities for escaping.

```
  var entityMap = {
    escape: {
      '&': '&amp;',
      '<': '&lt;',
      '>': '&gt;',
      '"': '&quot;',
      "'": '&#x27;',
      '/': '&#x2F;'
    }
  };
  entityMap.unescape = _.invert(entityMap.escape);
```

Regexes containing the keys and values listed immediately above.

```
  var entityRegexes = {
    escape:   new RegExp('[' + _.keys(entityMap.escape).join('') + ']', 'g'),
    unescape: new RegExp('(' + _.keys(entityMap.unescape).join('|') + ')', 'g')
  };
```

Functions for escaping and unescaping strings to/from HTML interpolation.

```
  _.each(['escape', 'unescape'], function(method) {
    _[method] = function(string) {
      if (string == null) return '';
      return ('' + string).replace(entityRegexes[method], function(match) {
        return entityMap[method][match];
      });
    };
  });
```

If the value of the named `property` is a function then invoke it with the `object` as context; otherwise, return it.

```
  _.result = function(object, property) {
    if (object == null) return void 0;
    var value = object[property];
    return _.isFunction(value) ? value.call(object) : value;
  };
```

Add your own custom functions to the Underscore object.

```
  _.mixin = function(obj) {
    each(_.functions(obj), function(name){
      var func = _[name] = obj[name];
      _.prototype[name] = function() {
        var args = [this._wrapped];
        push.apply(args, arguments);
        return result.call(this, func.apply(_, args));
      };
    });
  };
```

Generate a unique integer id (unique within the entire client session). Useful for temporary DOM ids.

```
  var idCounter = 0;
  _.uniqueId = function(prefix) {
    var id = ++idCounter + '';
    return prefix ? prefix + id : id;
  };
```

By default, Underscore uses ERB-style template delimiters, change the following template settings to use alternative delimiters.

```
  _.templateSettings = {
    evaluate    : /<%([\s\S]+?)%>/g,
    interpolate : /<%=([\s\S]+?)%>/g,
    escape      : /<%-([\s\S]+?)%>/g
  };
```

When customizing `templateSettings`, if you don't want to define an interpolation, evaluation or escaping regex, we need one that is guaranteed not to match.

```
var noMatch = /(.)^/;
```

Certain characters need to be escaped so that they can be put into a string literal.

```
var escapes = {
  "'":      "'",
  '\\':     '\\',
  '\r':     'r',
  '\n':     'n',
  '\t':     't',
  '\u2028': 'u2028',
  '\u2029': 'u2029'
};

var escaper = /\\|'|\r|\n|\t|\u2028|\u2029/g;
```

JavaScript micro-templating, similar to John Resig's implementation. Underscore templating handles arbitrary delimiters, preserves whitespace, and correctly escapes quotes within interpolated code.

```
_.template = function(text, data, settings) {
  var render;
  settings = _.defaults({}, settings, _.templateSettings);
```

Combine delimiters into one regular expression via alternation.

```
  var matcher = new RegExp([
    (settings.escape || noMatch).source,
    (settings.interpolate || noMatch).source,
    (settings.evaluate || noMatch).source
  ].join('|') + '|$', 'g');
```

Compile the template source, escaping string literals appropriately.

```
var index = 0;
var source = "__p+='";
text.replace(matcher, function(match, escape, interpolate, evaluate, offset) {
  source += text.slice(index, offset)
    .replace(escaper, function(match) { return '\\' + escapes[match]; });

  if (escape) {
    source += "'+\n((__t=(" + escape + "))==null?'':_.escape(__t))+\n'";
  }
  if (interpolate) {
    source += "'+\n((__t=(" + interpolate + "))==null?'':__t)+\n'";
  }
  if (evaluate) {
    source += "';\n" + evaluate + "\n__p+='";
  }
  index = offset + match.length;
  return match;
});
source += "';\n";
```

If a variable is not specified, place data values in local scope.

```
if (!settings.variable) source = 'with(obj||{}){\n' + source + '}\n';

source = "var __t,__p='',__j=Array.prototype.join," +
  "print=function(){__p+=__j.call(arguments,'');};\n" +
  source + "return __p;\n";

try {
  render = new Function(settings.variable || 'obj', '_', source);
} catch (e) {
  e.source = source;
  throw e;
}

if (data) return render(data, _);
var template = function(data) {
  return render.call(this, data, _);
};
```

Provide the compiled function source as a convenience for precompilation.

```
template.source = 'function(' + (settings.variable || 'obj') + '){\n' + source + '}';
```

```
    return template;
  };
```

Add a “chain” function, which will delegate to the wrapper.

```
  _.chain = function(obj) {
    return _(obj).chain();
  };
```

OOP

If Underscore is called as a function, it returns a wrapped object that can be used OO-style. This wrapper holds altered versions of all the underscore functions. Wrapped objects may be chained.

Helper function to continue chaining intermediate results.

```
  var result = function(obj) {
    return this._chain ? _(obj).chain() : obj;
  };
```

Add all of the Underscore functions to the wrapper object.

```
  _.mixin(_);
```

Add all mutator Array functions to the wrapper.

```
  each(['pop', 'push', 'reverse', 'shift', 'sort', 'splice', 'unshift'], function(name
    var method = ArrayProto[name];
    _.prototype[name] = function() {
      var obj = this._wrapped;
      method.apply(obj, arguments);
      if ((name == 'shift' || name == 'splice') && obj.length === 0) delete obj[0];
      return result.call(this, obj);
    };
  });
```

Add all accessor Array functions to the wrapper.

```
  each(['concat', 'join', 'slice'], function(name) {
    var method = ArrayProto[name];
    _.prototype[name] = function() {
      return result.call(this, method.apply(this._wrapped, arguments));
    };
  });

  _.extend(_.prototype, {
```

Start chaining a wrapped Underscore object.

```
    chain: function() {
      this._chain = true;
      return this;
    },
```

Extracts the result from a wrapped and chained object.

```
    value: function() {
      return this._wrapped;
    }

  });

}).call(this);
```