**Underscore**

**Underscore** — библиотека [JavaScript](https://ru.wikipedia.org/wiki/JavaScript" \o "JavaScript), реализующая дополнительную функциональность для работы с массивами, объектами и функциями, изначально отсутствующую в javascript, но имеющую аналоги в других языках. Библиотека умеет делегировать вызовы, если какая-то функциональность реализована разработчиками браузеров.

Underscore — это невероятно удобная JavaScript-библиотека, этакий швейцарский нож для js-разработчика, набор функций-утилит, которые так привычны любителям Prototype.js (или Ruby). Однако в отличие от Prototype.js, underscore не модифицирует прототипы встроенных объектов JavaScript. Данная особенность гарантирует бесконфликтность, что позволяет замечательно сочетать этот инструмент с, например, jQuery или Backbone, или и тем и другим вместе.

Наша библиотека используется в самых современных проектах, одним из таких проектов является [пин ап](https://riskmap6.com/). Широкая линейка игровых автоматов и новейшие слотовые игры позволят вам провести время с удовольствием в пин уп. Изучайте современные фреймворки и зарабатывайте вместе с pin up.

В состав Underscore входит более 100 функций, часть из которых решает стандартные задачи, например упрощает работу с коллекциями, а часть позволяет проще делать более экзотические вещи, такие как: функциональное связывание, javascript-шаблонизация, проверки на равенство (напр. объектов), и т.д. Кроме того, Underscore умеет делегировать вызовы, т.е. если код выполняется в современном браузере, который имеет нативные реализации таких методов, как: **forEach**, **map**, **reduce**, **filter**, **every**, **some** и **indexOf**, то будут вызваны именно они.

Функциональные тесты и тесты производительности прилагаются

Также вы можете просмотреть отлично документированные [исходники](http://underscorejs.ru/docs/underscore.html).

Если вам нравится Underscore, вы можете помочь или принять участие в разработке на Underscore-contrib.

Проект хостится на GitHub. Баг-репорты и фич-реквесты можно разместить тут, на Freenode в канале #documentcloud, или в чате Gitter. Баги по русской версии документации принимаются в этом репозитории.

*Underscore - это open-source компонент DocumentCloud.*

Работа с коллекциями:

**each**\_.each(list, iteratee, [context])

Проходит по всему списку элементов, вызывая для каждого из них функцию **iteratee**, которая будет вызвана в контексте **context**, если он был передан. При каждом вызове в **iteratee** будут переданы 3 аргумента: (element, index, list). В случае, если **list** является JavaScript-объектом, то в **iteratee** будут переданы (value, key, list). Возвращает **list** для цепочного вызова.

\_.each([1, 2, 3], alert);

=> выведет через alert все цифры, по очереди...

\_.each({one: 1, two: 2, three: 3}, alert);

=> выведет через alert все пары, по очереди…

**map**\_.map(list, iteratee, [context]) *Алиас:****collect***  
Вернёт новый массив, полученный преобразованием каждого элемента **list** в функции (**iteratee**). Функция iteratee получает три аргумента: значение value, индекс index (или ключ key), и ссылку на весь список list.

\_.map([1, 2, 3], function(num){ return num \* 3; });

=> [3, 6, 9]

\_.map({one: 1, two: 2, three: 3}, function(num, key){ return num \* 3; });

=> [3, 6, 9]

\_.map([[1, 2], [3, 4]], \_.first);

=> [1, 3]

**reduce**\_.reduce(list, iteratee, [memo], [context]) *Алиасы:****inject****,****foldl***  
Также известная как **inject** или **foldl**, функция **reduce** объединяет в одно все значения из **list**. **Memo** хранит начальное состояние, и должна возвращаться функцией **iteratee** после каждого успешного шага вычислений. Функция **iteratee** получает четыре аргумента: memo, затем value и index (или key) итерации, и, последней, ссылку на весь list.

Если никакого значения **memo** не было передано при вызове **reduce**, функция **iteratee** не будет вызвана на первом элементе списка. Вместо этого, первый элемент будет использован в качестве начального состояния.

var sum = \_.reduce([1, 2, 3], function(memo, num){ return memo + num; }, 0);

=> 6

**reduceRight**\_.reduceRight(list, iteratee, memo, [context]) *Алиас:****foldr***  
Правоассоциативная версия **reduce**. Однако **foldr** не так полезен в JavaScript, как в языках с ленивыми вычислениями.

var list = [[0, 1], [2, 3], [4, 5]];

var flat = \_.reduceRight(list, function(a, b) { return a.concat(b); }, []);

=> [4, 5, 2, 3, 0, 1]

**find**\_.find(list, predicate, [context]) *Алиас:****detect***  
Вызывает для каждого элемента **list** функцию сравнения **predicate**, возвращая первый элемент, для которого **predicate** вернула *true*, или *undefined*, если ни один элемент не подошёл. Выполнение функции при этом прерывается, т.е. вызовов **iterator** для оставшейся части **list** не будет.

var even = \_.find([1, 2, 3, 4, 5, 6], function(num){ return num % 2 == 0; });

=> 2

**filter**\_.filter(list, predicate, [context]) *Алиас:****select***  
Проходит через каждое значение **list**, возвращая массив всех значений, для которых **predicate** вернул *true*.

var evens = \_.filter([1, 2, 3, 4, 5, 6], function(num){ return num % 2 == 0; });

=> [2, 4, 6]

**where**\_.where(list, properties)  
Вернёт массив из элементов **list**, для которых совпадают значения для соответсвующих ключей, перечисленные в **properties**.

\_.where(listOfPlays, {author: "Shakespeare", year: 1611});

=> [{title: "Cymbeline", author: "Shakespeare", year: 1611},

{title: "The Tempest", author: "Shakespeare", year: 1611}]

**findWhere**\_.findWhere(list, properties)  
Сработает аналогично **where**, но вернёт только первый элемент, соответствующий заданным **properties**.

Если совпадений не найдено или **list** пустой, то вернёт *undefined*.

\_.findWhere(publicServicePulitzers, {newsroom: "The New York Times"});

=> {year: 1918, newsroom: "The New York Times",

reason: "For its public service in publishing in full so many official reports,

documents and speeches by European statesmen relating to the progress and

conduct of the war."}

**reject**\_.reject(list, predicate, [context])  
Возвращает массив, содержащий все значения **list**, за исключением элементов, для которых функция **predicate** вернула значение *true*. Т.е. **reject** является «антонимом» **filter**.

var odds = \_.reject([1, 2, 3, 4, 5, 6], function(num){ return num % 2 == 0; });

=> [1, 3, 5]

**every**\_.every(list, [predicate], [context]) *Алиас:****all***  
Вернёт *true*, если для каждого значения из **list** **predicate** вернёт *true*. Прерывает обход списка как только обнаружит несоответствие.

\_.every([2, 4, 5], function(num) { return num % 2 == 0; });

=> false

**some**\_.some(list, [predicate], [context]) *Алиас:****any***  
Вернёт *true*, если хотя бы для одного значения из **list** **predicate** вернёт *true*. После нахождения первого удовлетворяющего условию элемента выполнение функции прерывается.

\_.some([null, 0, 'yes', false]);

=> true

**contains**\_.contains(list, value, [fromIndex]) *Алиас:****includes***  
Вернёт *true*, если в **list** содержится элемент, эквивалентный **value**. Для сравнения используется оператор ===. Если **list** является массивом, будет вызван метод **indexOf**. Используйте **fromIndex**, чтобы начать поиск с конкретного индекса.

\_.contains([1, 2, 3], 3);

=> true

**invoke**\_.invoke(list, methodName, \*arguments)  
Вызовет метод **methodName** на каждом значении **list**. Все дополнительные аргументы, если они переданы в **invoke**, будут также переданы при вызове **methodName**.

\_.invoke([[5, 1, 7], [3, 2, 1]], 'sort');

=> [[1, 5, 7], [1, 2, 3]]

**pluck**\_.pluck(list, propertyName)  
Краткий алиас для одного из наиболее частых применений функции **map**: вернёт массив, состоящий из значений свойства **propertyName** каждого из элементов **list**.

var stooges = [{name: 'moe', age: 40}, {name: 'larry', age: 50}, {name: 'curly', age: 60}];

\_.pluck(stooges, 'name');

=> ["moe", "larry", "curly"]

**max**\_.max(list, [iteratee], [context])  
Вернёт максимальное значение из **list**. Если был передан **iteratee**, то он будет использован для генерация критерия, по которому будет проведено сравнение. Если список пуст, будет возвращено *-Infinity*. Поэтому используйте [isEmpty](http://underscorejs.ru/" \l "isEmpty) для проверки. Не-числовые элементы **list** будут пропущены.

var stooges = [{name: 'moe', age: 40}, {name: 'larry', age: 50}, {name: 'curly', age: 60}];

\_.max(stooges, function(stooge){ return stooge.age; });

=> {name: 'curly', age: 60};

**min**\_.min(list, [iteratee], [context])  
Вернёт минимальное значение из **list**. Если был передан **iteratee**, то он будет использован для генерации критерия, по которому будет проведено сравнение. Если список пуст, будет возвращено *Infinity*. Поэтому используйте [isEmpty](http://underscorejs.ru/" \l "isEmpty) для проверки. Не-числовые элементы **list** будут пропущены.

var numbers = [10, 5, 100, 2, 1000];

\_.min(numbers);

=> 2

**sortBy**\_.sortBy(list, iteratee, [context])  
Возвращает копию списка, отсортированную по возрастанию значений, которые возвращает функция **iteratee**, вызываемая на каждом элементе списка. Параметр **iteratee** может также быть названием поля, по которому должна вестить сортировка (например length).

\_.sortBy([1, 2, 3, 4, 5, 6], function(num){ return Math.sin(num); });

=> [5, 4, 6, 3, 1, 2]

var stooges = [{name: 'moe', age: 40}, {name: 'larry', age: 50}, {name: 'curly', age: 60}];

\_.sortBy(stooges, 'name');

=> [{name: 'curly', age: 60}, {name: 'larry', age: 50}, {name: 'moe', age: 40}];

**groupBy**\_.groupBy(list, iteratee, [context])  
Разделяет коллекцию на группы, объединённые по результатам вызова функции **iteratee** на каждом элементе коллекции. Если параметр **iteratee** содержит строку с названием поля, коллекция будет сгруппирована по значениям этого поля в элементах.

\_.groupBy([1.3, 2.1, 2.4], function(num){ return Math.floor(num); });

=> {1: [1.3], 2: [2.1, 2.4]}

\_.groupBy(['one', 'two', 'three'], 'length');

=> {3: ["one", "two"], 5: ["three"]}

**indexBy**\_.indexBy(list, iteratee, [context])  
На вход получает коллекцию **list** и **iteratee** - функцию, которая возвращает ключ для каждого элемента коллекции (либо имя свойства), вернёт объект из пар индекс : соответствующий ему объект. Т.е. работает аналогично [groupBy](http://underscorejs.ru/" \l "groupBy), но для случаев, когда индексы уникальны.

var stooges = [{name: 'moe', age: 40}, {name: 'larry', age: 50}, {name: 'curly', age: 60}];

\_.indexBy(stooges, 'age');

=> {

"40": {name: 'moe', age: 40},

"50": {name: 'larry', age: 50},

"60": {name: 'curly', age: 60}

}

**countBy**\_.countBy(list, iteratee, [context])  
Сортирует коллекцию в группы и возвращает количество объектов в каждой группе. Т.е. работает аналогично groupBy, но возвращает количество элементов вместо самой группы.

\_.countBy([1, 2, 3, 4, 5], function(num) {

return num % 2 == 0 ? 'even': 'odd';

});

=> {odd: 3, even: 2}

**shuffle**\_.shuffle(list)  
Вернет перемешанную копию **list**, используя метод случайного перемешивания Фишера-Ятса.

\_.shuffle([1, 2, 3, 4, 5, 6]);

=> [4, 1, 6, 3, 5, 2]

**sample**\_.sample(list, [n])  
Возвращает **n** случайных элементов из **list**. Если аргумент **n** не был передан, вернёт один элемент.

\_.sample([1, 2, 3, 4, 5, 6]);

=> 4

\_.sample([1, 2, 3, 4, 5, 6], 3);

=> [1, 6, 2]

**toArray**\_.toArray(list)  
Сконвертирует **list** в массив. Удобен, например, для преобразования объекта **arguments**.

(function(){ return \_.toArray(arguments).slice(1); })(1, 2, 3, 4);

=> [2, 3, 4]

**size**\_.size(list)  
Вернёт количество элементов в **list**.

\_.size({one: 1, two: 2, three: 3});

=> 3

**partition**\_.partition(array, predicate)  
Разобъёт массив **array** на две части: одна - для элементов которой функция **predicate** вернёт **true**, и другая - для всех остальных.

\_.partition([0, 1, 2, 3, 4, 5], isOdd);

=> [[1, 3, 5], [0, 2, 4]]

Работа с объектами:

**keys**\_.keys(object)  
Возвращает все ключи объекта. (В случае массива - индексы.)

\_.keys({one: 1, two: 2, three: 3});

=> ["one", "two", "three"]

**allKeys**\_.allKeys(object)  
Возвращает названия *всех* собственных и унаследованных полей объекта.

function Stooge(name) {

this.name = name;

}

Stooge.prototype.silly = true;

\_.allKeys(new Stooge("Moe"));

=> ["name", "silly"]

**values**\_.values(object)  
Возвращает *значения* всех собственных полей объекта.

\_.values({one: 1, two: 2, three: 3});

=> [1, 2, 3]

**mapObject**\_.mapObject(object, iteratee, [context])  
Как [map](http://underscorejs.ru/#map), только для объектов. Преобразует поочерёдно значение каждого поля объекта.

\_.mapObject({start: 5, end: 12}, function(val, key) {

return val + 5;

});

=> {start: 10, end: 17}

**pairs**\_.pairs(object)  
Возвращает список пар [key, value] (ключ - значение) объекта. Действие противоположно функции [object](http://underscorejs.ru/#object).

\_.pairs({one: 1, two: 2, three: 3});

=> [["one", 1], ["two", 2], ["three", 3]]

**invert**\_.invert(object)  
Вернёт копию **object** где ключи — значения, а значения — ключи. Чтобы это заработало, нужно, чтобы все значения свойств объекта могли быть уникально сериализованы в строки.

\_.invert({Moe: "Moses", Larry: "Louis", Curly: "Jerome"});

=> {Moses: "Moe", Louis: "Larry", Jerome: "Curly"};

**create**\_.create(prototype, props)  
Создаёт новый объект с заданным прототипом, опционально добавляя **props** в качестве *собственных* полей объекта. Своего рода Object.create, но без лишней возни с дескрипторами.

var moe = \_.create(Stooge.prototype, {name: "Moe"});

**functions**\_.functions(object) *Алиас:****methods***  
Возвращает отсортированный массив с названиями всех методов объекта.

\_.functions(\_);

=> ["all", "any", "bind", "bindAll", "clone", "compact", "compose" ...

**findKey**\_.findKey(object, predicate, [context])  
Похоже на [\_.findIndex](http://underscorejs.ru/#findIndex), но для ключей объекта. Возвращает ключ *key*, для которого **predicate** возвращает *true*. Или *undefined* если такой ключ не был найден.

**extend**\_.extend(destination, \*sources)  
Скопирует все свойства из объектов **source** в объект **destination**. Все вложенные объекты или массивы будут скопированы "по ссылке" (т.е. не создавая копий). Если объекты **source** имеют одноименные свойства, то значения каждого будет затёрто значениями из следующего.

\_.extend({name: 'moe'}, {age: 50});

=> {name: 'moe', age: 50}

**extendOwn**\_.extendOwn(destination, \*sources) *Алиас:****assign***  
Как **extend**, но копирует только *собственные* значения объекта.

**pick**\_.pick(object, \*keys)  
Возвращает копию объекта, содержащую только указанные ключи. Вместо ключей, можно указать функцию, определяющую, какие ключи оставить.

\_.pick({name: 'moe', age: 50, userid: 'moe1'}, 'name', 'age');

=> {name: 'moe', age: 50}

\_.pick({name: 'moe', age: 50, userid: 'moe1'}, function(value, key, object) {

return \_.isNumber(value);

});

=> {age: 50}

**omit**\_.omit(object, \*keys)  
Возвращает копию объекта без указанных ключей (или массива ключей). Вместо ключей, можно передать функцию, которая будет определять, какие ключи отбросить.

\_.omit({name: 'moe', age: 50, userid: 'moe1'}, 'userid');

=> {name: 'moe', age: 50}

\_.omit({name: 'moe', age: 50, userid: 'moe1'}, function(value, key, object) {

return \_.isNumber(value);

});

=> {name: 'moe', userid: 'moe1'}

**defaults**\_.defaults(object, \*defaults)  
Проинициализирует неопределённые (undefined) свойства объета значениями одноимённых свойств из **defaults**. Если же какие-то свойства объекта уже определены, то они не будут изменены.

var iceCream = {flavor: "chocolate"};

\_.defaults(iceCream, {flavor: "vanilla", sprinkles: "lots"});

=> {flavor: "chocolate", sprinkles: "lots"}

**clone**\_.clone(object)  
Вернёт клон **object**. Любые, входящие в его состав массивы или объекты будут скопированы по ссылке, а *не* продублированы.

\_.clone({name: 'moe'});

=> {name: 'moe'};

**tap**\_.tap(object, interceptor)  
Вызовет **interceptor** с **object** в качестве аргумента и затем вернёт **object**. Основная задача данного метода — возможность «вклиниться» в цепочку вызовов для того, чтобы произвести некие операции с промежуточными итогами.

\_.chain([1,2,3,200])

.filter(function(num) { return num % 2 == 0; })

.tap(alert)

.map(function(num) { return num \* num })

.value();

=> // [2, 200] (alerted)

=> [4, 40000]

**has**\_.has(object, key)  
Проверяет, содержит ли объект указанный ключ. Идентично object.hasOwnProperty(key), но использует безопасную ссылку на функцию hasOwnProperty, на случай, если функция была случайно затёрта.

\_.has({a: 1, b: 2, c: 3}, "b");

=> true

**property**\_.property(key)  
Возвращает функцию, которая вернёт ключ **key** любого переданого её объекта.

var stooge = {name: 'moe'};

'moe' === \_.property('name')(stooge);

=> true

**propertyOf**\_.propertyOf(object)  
Обратно по действию функции \_.property. Берёт объект, и возвращает функцию, которая будет возвращать значение любого переданного ключа, для этого объекта.

var stooge = {name: 'moe'};

\_.propertyOf(stooge)('name');

=> 'moe'

**matcher**\_.matcher(attrs) *Алиас:****matches***  
Вернёт логическую функцию, с помощью которой можно будет определить, обладает ли объект такими же полями-значениями как в **attrs**.

var ready = \_.matcher({selected: true, visible: true});

var readyToGoList = \_.filter(list, ready);

**isEqual**\_.isEqual(object, other)  
Выполняет оптимизированное "глубокое" сравнение двух объектов, чтобы определить их эквивалентность.

var stooge = {name: 'moe', luckyNumbers: [13, 27, 34]};

var clone = {name: 'moe', luckyNumbers: [13, 27, 34]};

stooge == clone;

=> false

\_.isEqual(stooge, clone);

=> true

**isMatch**\_.isMatch(object, properties)  
Проверяет, содержатся ли ключи-значения **properties** в объекте.

var stooge = {name: 'moe', age: 32};

\_.isMatch(stooge, {age: 32});

=> true

**isEmpty**\_.isEmpty(object)  
Вернёт *true* если коллекция **object** не содержит ни одного значения. Для строк и массивов \_.isEmpty проверяет длину на равенство *0*.

\_.isEmpty([1, 2, 3]);

=> false

\_.isEmpty({});

=> true

**isElement**\_.isElement(object)  
Вовращает *true* если **object** - элемент DOM.

\_.isElement(jQuery('body')[0]);

=> true

**isArray**\_.isArray(object)  
Возвращает *true* если **object** - это массив.

(function(){ return \_.isArray(arguments); })();

=> false

\_.isArray([1,2,3]);

=> true

**isObject**\_.isObject(value)  
Возвращает *true* если **value** - это объект. Примите во внимание, что массивы и функции в JavaScript это тоже объекты, тогда как строки и числа - нет.

\_.isObject({});

=> true

\_.isObject(1);

=> false

**isArguments**\_.isArguments(object)  
Возвращает *true* если **object** это объект Arguments.

(function(){ return \_.isArguments(arguments); })(1, 2, 3);

=> true

\_.isArguments([1,2,3]);

=> false

**isFunction**\_.isFunction(object)  
Возвращает *true* если **object** это функция.

\_.isFunction(alert);

=> true

**isString**\_.isString(object)  
Возвращает *true* если **object** это строка.

\_.isString("moe");

=> true

**isNumber**\_.isNumber(object)  
Возвращает *true* если **object** это число (включая NaN).

\_.isNumber(8.4 \* 5);

=> true

**isFinite**\_.isFinite(object)  
Возвращает *true* если **object** это конечное число.

\_.isFinite(-101);

=> true

\_.isFinite(-Infinity);

=> false

**isBoolean**\_.isBoolean(object)  
Возвращает *true* если **object** это *true* или *false*.

\_.isBoolean(null);

=> false

**isDate**\_.isDate(object)  
Возвращает *true* если **object** является датой (Date).

\_.isDate(new Date());

=> true

**isRegExp**\_.isRegExp(object)  
Возвращает *true* если **object** это регулярное выражение (RegExp).

\_.isRegExp(/moe/);

=> true

**isError**\_.isError(object)  
Возвращает *true* если **object** это объект унаследованный от Error.

try {

throw new TypeError("Example");

} catch (o\_O) {

\_.isError(o\_O);

}

=> true

**isSymbol**\_.isSymbol(object)  
Возвращает *true* если **object** это символ (Symbol).

\_.isSymbol(Symbol());

=> true

**isNaN**\_.isNaN(object)  
Возвращает *true* если **object** это *NaN*.  
Примечание: это не тоже самое, что нативная функция **isNaN**, которая также возвращает *true* для многих не числовых значений, таких как undefined.

\_.isNaN(NaN);

=> true

isNaN(undefined);

=> true

\_.isNaN(undefined);

=> false

**isNull**\_.isNull(object)  
Возвращает *true* если значение **object** равно *null*.

\_.isNull(null);

=> true

\_.isNull(undefined);

=> false

**isUndefined**\_.isUndefined(value)  
Возвращает *true* если значение **value** равно *undefined*.

\_.isUndefined(window.missingVariable);

=> true

**Объектно-Ориентированный стиль**

Вы можете использовать Underscore как в функциональном, так и в объектно-ориентированном стиле. Эти две строки кода выполняют одну и ту же операцию: удвоение чисел в списке.

\_.map([1, 2, 3], function(n){ return n \* 2; });

\_([1, 2, 3]).map(function(n){ return n \* 2; });

**Цепочные вызовы**

Вызов chain приведёт к тому, что все последующие вызовы будут возвращать "обёрнутые" объекты. Когда вы закончите вычисления, вызовите value чтобы получить финальный результат. Вот пример объединения в одну "цепь" вызовов **map/flatten/reduce**, для того, чтобы подсчитать, сколько раз встречается каждое слово в стихотворении:

var lyrics = [

{line: 1, words: "I'm a lumberjack and I'm okay"},

{line: 2, words: "I sleep all night and I work all day"},

{line: 3, words: "He's a lumberjack and he's okay"},

{line: 4, words: "He sleeps all night and he works all day"}

];

\_.chain(lyrics)

.map(function(line) { return line.words.split(' '); })

.flatten()

.reduce(function(counts, word) {

counts[word] = (counts[word] || 0) + 1;

return counts;

}, {})

.value();

=> {lumberjack: 2, all: 4, night: 2 ... }

Кроме того, цепочка, сделанная из объекта Underscore, обладает всеми методами стандартного объекта Array. А это значит, что вы можете воспользоваться в цепочке методом reverse или push, и затем продолжить вычисления.

**chain**\_.chain(obj)  
Возвращает "обёрнутый" объект. Вызовы функций на нём также будут возвращать "обёрнутые" объекты, до тех пор, пока не будет вызван метод value.

var stooges = [{name: 'curly', age: 25}, {name: 'moe', age: 21}, {name: 'larry', age: 23}];

var youngest = \_.chain(stooges)

.sortBy(function(stooge){ return stooge.age; })

.map(function(stooge){ return stooge.name + ' is ' + stooge.age; })

.first()

.value();

=> "moe is 21"

**value**\_.chain(obj).value()  
Извлекает значение из "обёрнутого" объекта.

\_.chain([1, 2, 3]).reverse().value();

=> [3, 2, 1]

Работа с коллекциями

Метод each()

Служит для перебора всех элементов коллекции и применения функции к каждому из элементов. Код ниже пробежится по всем элементам коллекции a1 и выведет каждый из этих элементов в консоль браузера:

var a1 = [ 1, 2, 3 ];

each( a1, function (el) { console.log(el) });

Метод find()

С помощью этого метода можно искать элементы в коллекциях\массивах. Метод возвращает первый элемент, который удовлетворяет условию, заданному в функции:

var findExample = [ 1, 2, 3, 4, 5, 6, 7, 8, 9 ];

var findResult = find( findExample, function (el) {

return el % 2 == 0;

});

Метод filter()

Метод для поиска элементов массива по какому-то условию. Возвращает новый массив:

var filterExample = [ 1,2,3,4,5,6,7,8,9 ];

var filterResult = filter( filterExample, function (el) {

return el % 3 === 0

Работа с массивами

Метод first()

Этот метод возвращает первый элемент массива - все просто и понятно:

var a1 = [ 1, 2, 3 ];

var firstEl = first(a1);

Метод object()

Этот метод преобразует несколько массивов в один объект:

var names = [ 'Peter', 'Mary', 'John' ];

var ages = [ 30, 20, 40 ];

var peoples = object( names, ages );

peoples = {Peter: 30, Mary: 20, John: 40};

Работа с функциями

Метод bind()

Этот метод “привязывает” исполнение функции только к определенному объекту. Чтобы было понятнее, начнем из далека, из классики.

Создадим три объекта, у каждого из которых будет ключ name:

var someObj1 = {};

var someObj2 = {};

var someObj3 = {};

someObj1.name = 'Peter';

someObj2.name = 'Mary';

someObj3.name = 'John';

Работа с объектами

Метод keys()

Преобразует ключи объекта в массив:

var fruits = { apple: 10, melon: 20, potato: 30, tomato: 50 };

var fruitSorts = keys( fruits );

Метод values()

Преобразует значения ключей объекта в массив:

var fruitsvalues = values( fruits );

Метод pairs()

Преобразует пары ключ-значение объекта в массив, состоящий из подмассивов:

var fruitsPairs = pairs( fruits );

Метод invert()

Инвертирует пару ключ:значение в объекте:

var fruitsInvert = invert( fruits );

Метод pick()

Вернет новый объект, в котором будут только указанные ключи:

var fruitsPicked = pick( fruits, [ 'melon', 'tomato' ] );

Метод clone()

Возвращает полную копию оригинального объекта:

var man = { weight: 80, age: 30, height: 180, gender: 'male' };

var manDouble = clone( man );

man.age = 32;

manDouble.gender = 'female';

console.log( man );

console.log( manDouble );

Утилиты

Метод random()

Возвращает случайное число из диапазона min - max ( включительно нижнюю и верхнюю границы )

var rnd = random( 0, 255 );

Метод now()

Возвращает текущее время:

var currTime = now();

console.log( currTime );

Метод times()

Запускаем функцию на исполнение три раза:

times(3, function () {

console.log( 'Holla!' );

});