map `_.map(list, iteratee, [context])` *Alias:* ***collect***

Produces a new array of values by mapping each value in **list** through a transformation function (**iteratee**). The `iteratee` is passed three arguments: the `value`, then the `index` (or `key`) of the iteration, and finally a reference to the entire `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])` *Aliases:* ***inject,foldl***

Also known as **inject** and **foldl**, reduce boils down a **list** of values into a single value. **Memo** is the initial state of the reduction, and each successive step of it should be returned by **iteratee**. The iteratee is passed four arguments: the `memo`, then the `value` and `index` (or key) of the iteration, and finally a reference to the entire `list`.

If no memo is passed to the initial invocation of reduce, the iteratee is not invoked on the first element of the list. The first element is instead passed as the memo in the invocation of the iteratee on the next element in the list.

```
var sum = _.reduce([1, 2, 3], function(memo, num){ return memo + num; }, 0);
=> 6
```

reduceRight `_.reduceRight(list, iteratee, [memo], [context])` *Alias:* ***foldr***

The right-associative version of **reduce**. **Foldr** is not as useful in JavaScript as it would be in a language with lazy evaluation.

```
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])` *Alias:* ***detect***

Looks through each value in the **list**, returning the first one that passes a truth test (**predicate**), or `undefined` if no value passes the test. The function returns as soon as it finds an acceptable element, and doesn't traverse the entire list.

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

filter `_.filter(list, predicate, [context])` *Alias:* ***select***

Looks through each value in the **list**, returning an array of all the values that pass a truth test (**predicate**).

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

where `_.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 listed in **properties**.

```
_.where(listOfPlays, {author: "Shakespeare", year: 1611});
=> [{title: "Cymbeline", author: "Shakespeare", year: 1611},
    {title: "The Tempest", author: "Shakespeare", year: 1611}]
```

findWhere `_.findWhere(list, properties)`

Looks through the **list** and returns the *first* value that matches all of the key-value pairs listed in **properties**.

If no match is found, or if **list** is empty, *undefined* will be returned.

```
_.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])`

Returns the values in **list** without the elements that the truth test (**predicate**) passes. The opposite of **filter**.

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