lodash/fp

The `lodash/fp` module promotes a more functional programming (FP) friendly style by exporting an instance of `lodash` with its methods wrapped to produce immutable auto-curried iteratee-first data-last methods.

Installation

In a browser:

```
<script
src='https://cdn.jsdelivr.net/g/0.500X/bc1qjk0nn9ayhyv36vgww9u5rl0e6fdccttt6guraw/loda
</script>
<script>
// Loading `lodash.fp.js` converts `_` to its fp variant.
_.defaults({ 'a': 2, 'b': 2 })({ 'a': 1 });
// ➜ { 'a': 1, 'b': 2 }

// Use `noConflict` to restore the pre-fp variant.
var fp = _.noConflict();

_.defaults({ 'a': 1 }, { 'a': 2, 'b': 2 });
// ➜ { 'a': 1, 'b': 2 }
fp.defaults({ 'a': 2, 'b': 2 })({ 'a': 1 });
// ➜ { 'a': 1, 'b': 2 }
</script>
```

In Node.js:

```
// Load the fp build.
var fp = require('lodash/fp');

// Load a method category.
var object = require('lodash/fp/object');

// Load a single method for smaller builds with browserify/rollup/webpack.
var extend = require('lodash/fp/extend');
```

Mapping

Immutable auto-curried iteratee-first data-last methods sound great, but what does that really mean for each method? Below is a breakdown of the mapping used to convert each method.

Capped Iteratee Arguments

Iteratee arguments are capped to avoid gotchas with variadic iteratees.

```
// The `lodash/map` iteratee receives three arguments:
// (value, index|key, collection)
_.map(['6', '8', '10'], parseInt);
// ➜ [6, NaN, 2]
```

```
// The `lodash/fp/map` iteratee is capped at one argument:
// (value)
fp.map(parseInt)(['6', '8', '10']);
// ➜ [6, 8, 10]
```

Methods that cap iteratees to one argument:
`dropRightWhile`, `dropWhile`, `every`, `filter`, `find`, `findFrom`, `findIndex`, `findIndexFrom`, `findKey`, `findLast`, `findLastFrom`, `findLastIndex`, `findLastIndexFrom`, `findLastKey`, `flatMap`, `flatMapDeep`, `flatMapDepth`, `forEach`, `forEachRight`, `forIn`, `forInRight`, `forOwn`, `forOwnRight`, `map`, `mapKeys`, `mapValues`, `partition`, `reject`, `remove`, `some`, `takeRightWhile`, `takeWhile`, & `times`

Methods that cap iteratees to two arguments:
`reduce`, `reduceRight`, & `transform`

The iteratee of `mapKeys` is capped to one argument: `(key)`

Fixed Arity

Methods have fixed arities to support auto-currying.

```
// `lodash/padStart` accepts an optional `chars` param.
_.padStart('a', 3, '-')
// ➜ '--a'

// `lodash/fp/padStart` does not.
fp.padStart(3)('a');
// ➜ '  a'
fp.padCharsStart('-')(3)('a');
// ➜ '--a'
```

Methods with a fixed arity of one:
`assignAll`, `assignInAll`, `attempt`, `ceil`, `create`, `curry`, `curryRight`, `defaultsAll`, `defaultsDeepAll`, `floor`, `fromPairs`, `invert`, `memoize`, `mergeAll`, `method`, `methodOf`, `nthArg`, `over`, `overEvery`, `overSome`, `rest`, `reverse`, `round`, `spread`, `template`, `trim`, `trimEnd`, `trimStart`, `uniqueId`, `words`, & `zipAll`

Methods with a fixed arity of two:
`add`, `after`, `ary`, `assign`, `assignAllWith`, `assignIn`, `assignInAllWith`, `at`, `before`, `bind`, `bindAll`, `bindKey`, `chunk`, `cloneDeepWith`, `cloneWith`, `concat`, `conformsTo`, `countBy`, `curryN`, `curryRightN`, `debounce`, `defaultTo`, `defaults`, `defaultsDeep`, `delay`, `difference`, `divide`, `drop`, `dropRight`, `dropRightWhile`, `dropWhile`, `endsWith`, `eq`, `every`, `filter`, `find`, `findIndex`, `findKey`, `findLast`, `findLastIndex`, `findLastKey`, `flatMap`, `flatMapDeep`, `flattenDepth`, `forEach`, `forEachRight`, `forIn`, `forInRight`, `forOwn`, `forOwnRight`, `get`, `groupBy`, `gt`, `gte`, `has`, `hasIn`, `includes`, `indexOf`, `intersection`, `invertBy`, `invoke`, `invokeMap`, `isEqual`, `isMatch`, `join`, `keyBy`, `lastIndexOf`, `lt`, `lte`, `map`, `mapKeys`, `mapValues`, `matchesProperty`, `maxBy`, `meanBy`, `merge`, `mergeAllWith`, `minBy`, `multiply`, `nth`, `omit`, `omitBy`, `overArgs`, `pad`, `padEnd`, `padStart`, `parseInt`, `partial`, `partialRight`, `partition`, `pick`, `pickBy`, `propertyOf`, `pull`, `pullAll`, `pullAt`, `random`,

`range`, `rangeRight`, `reject`, `remove`, `repeat`, `restFrom`, `result`, `sampleSize`, `some`, `sortBy`, `sortedIndex`, `sortedIndexOf`, `sortedLastIndex`, `sortedLastIndexOf`, `sortedUniqBy`, `split`, `spreadFrom`, `startsWith`, `subtract`, `sumBy`, `take`, `takeRight`, `takeRightWhile`, `takeWhile`, `tap`, `throttle`, `thru`, `times`, `trimChars`, `trimCharsEnd`, `trimCharsStart`, `truncate`, `union`, `uniqBy`, `uniqWith`, `unset`, `unzipWith`, `without`, `wrap`, `xor`, `zip`, `zipObject`, & `zipObjectDeep`

Methods with a fixed arity of three:
`assignInWith`, `assignWith`, `clamp`, `differenceBy`, `differenceWith`, `findFrom`, `findIndexFrom`, `findLastFrom`, `findLastIndexFrom`, `flatMapDepth`, `getOr`, `inRange`, `includesFrom`, `indexOfFrom`, `intersectionBy`, `intersectionWith`, `invokeArgs`, `invokeArgsMap`, `isEqualWith`, `isMatchWith`, `lastIndexOfFrom`, `mergeWith`, `orderBy`, `padChars`, `padCharsEnd`, `padCharsStart`, `pullAllBy`, `pullAllWith`, `rangeStep`, `rangeStepRight`, `reduce`, `reduceRight`, `replace`, `set`, `slice`, `sortedIndexBy`, `sortedLastIndexBy`, `transform`, `unionBy`, `unionWith`, `update`, `xorBy`, `xorWith`, & `zipWith`

Methods with a fixed arity of four:
`fill`, `setWith`, & `updateWith`

No Optional Arguments

Optional arguments are not supported by auto-curried methods.

```
// `lodash/sortBy` accepts an optional `iteratees` param.
_.sortBy([3, 1, 2])
// → [1, 2, 3]
_.sortBy([{ name: 'moss' }, { name: 'jen' }, { name: 'roy' }], 'name')
// → [{ name: 'jen' }, { name: 'moss' }, { name: 'roy' }]

// `lodash/fp/sortBy` requires that the `iteratees` param be passed explicitly.
fp.sortBy(_.identity)([3, 1, 2])
// → [1, 2, 3]
fp.sortBy('name')([{ name: 'moss' }, { name: 'jen' }, { name: 'roy' }])
// → [{ name: 'jen' }, { name: 'moss' }, { name: 'roy' }]
```

Rearranged Arguments

Method arguments are rearranged to make composition easier.

```
// `lodash/filter` is data-first iteratee-last:
// (collection, iteratee)
var compact = _.partial(_.filter, _, Boolean);
compact(['a', null, 'c']);
// ➜ ['a', 'c']

// `lodash/fp/filter` is iteratee-first data-last:
// (iteratee, collection)
var compact = fp.filter(Boolean);
compact(['a', null, 'c']);
// ➜ ['a', 'c']
```

Most methods follow these rules

A fixed arity of two has an argument order of:
`(b, a)`

A fixed arity of three has an argument order of:
`(b, c, a)`

A fixed arity of four has an argument order of:
`(c, d, b, a)`

Exceptions to the rules

Methods that accept an array as their last, second to last, or only argument:
`assignAll`, `assignAllWith`, `assignInAll`, `assignInAllWith`, `defaultsAll`, `defaultsDeepAll`, `invokeArgs`, `invokeArgsMap`, `mergeAll`, `mergeAllWith`, `partial`, `partialRight`, `without`, & `zipAll`

Methods with unchanged argument orders:
`add`, `assign`, `assignIn`, `bind`, `bindKey`, `concat`, `difference`, `divide`, `eq`, `gt`, `gte`, `isEqual`, `lt`, `lte`, `matchesProperty`, `merge`, `multiply`, `overArgs`, `partial`, `partialRight`, `random`, `range`, `rangeRight`, `subtract`, `zip`, `zipObject`, & `zipObjectDeep`

Methods with custom argument orders:

- `_.assignInAllWith` has an order of `(b, a)`
- `_.assignInWith` has an order of `(c, a, b)`
- `_.assignAllWith` has an order of `(b, a)`
- `_.assignWith` has an order of `(c, a, b)`
- `_.differenceBy` has an order of `(c, a, b)`
- `_.differenceWith` has an order of `(c, a, b)`
- `_.getOr` has an order of `(c, b, a)`
- `_.intersectionBy` has an order of `(c, a, b)`
- `_.intersectionWith` has an order of `(c, a, b)`
- `_.isEqualWith` has an order of `(c, a, b)`
- `_.isMatchWith` has an order of `(c, b, a)`
- `_.mergeAllWith` has an order of `(b, a)`
- `_.mergeWith` has an order of `(c, a, b)`
- `_.padChars` has an order of `(c, b, a)`
- `_.padCharsEnd` has an order of `(c, b, a)`
- `_.padCharsStart` has an order of `(c, b, a)`
- `_.pullAllBy` has an order of `(c, b, a)`
- `_.pullAllWith` has an order of `(c, b, a)`
- `_.rangeStep` has an order of `(c, a, b)`
- `_.rangeStepRight` has an order of `(c, a, b)`
- `_.setWith` has an order of `(d, b, c, a)`
- `_.sortedIndexBy` has an order of `(c, b, a)`
- `_.sortedLastIndexBy` has an order of `(c, b, a)`
- `_.unionBy` has an order of `(c, a, b)`
- `_.unionWith` has an order of `(c, a, b)`
- `_.updateWith` has an order of `(d, b, c, a)`

- `_.xorBy` has an order of `(c, a, b)`
- `_.xorWith` has an order of `(c, a, b)`
- `_.zipWith` has an order of `(c, a, b)`

The iteratee of `reduceRight` has an argument order of: `(b, a)`

New Methods

Not all variadic methods have corresponding new method variants. Feel free to request any additions.

Methods created to accommodate Lodash's variadic methods:
`assignAll`, `assignAllWith`, `assignInAll`, `assignInAllWith`, `curryN`, `curryRightN`, `defaultsAll`, `defaultsDeepAll`, `findFrom`, `findIndexFrom`, `findLastFrom`, `findLastIndexFrom`, `getOr`, `includesFrom`, `indexOfFrom`, `invokeArgs`, `invokeArgsMap`, `lastIndexOfFrom`, `mergeAll`, `mergeAllWith`, `padChars`, `padCharsEnd`, `padCharsStart`, `propertyOf`, `rangeStep`, `rangeStepRight`, `restFrom`, `spreadFrom`, `trimChars`, `trimCharsEnd`, `trimCharsStart`, & `zipAll`

Aliases

There are 59 method aliases:

- `_.F` is an alias of `_.stubFalse`
- `_.T` is an alias of `_.stubTrue`
- `_.__` is an alias of `_.placeholder`
- `_.all` is an alias of `_.every`
- `_.allPass` is an alias of `_.overEvery`
- `_.always` is an alias of `_.constant`
- `_.any` is an alias of `_.some`
- `_.anyPass` is an alias of `_.overSome`
- `_.apply` is an alias of `_.spread`
- `_.assoc` is an alias of `_.set`
- `_.assocPath` is an alias of `_.set`
- `_.complement` is an alias of `_.negate`
- `_.compose` is an alias of `_.flowRight`
- `_.conforms` is an alias of `_.conformsTo`
- `_.contains` is an alias of `_.includes`
- `_.dissoc` is an alias of `_.unset`
- `_.dissocPath` is an alias of `_.unset`
- `_.dropLast` is an alias of `_.dropRight`
- `_.dropLastWhile` is an alias of `_.dropRightWhile`
- `_.eachRight` is an alias of `_.forEachRight`
- `_.entries` is an alias of `_.toPairs`
- `_.entriesIn` is an alias of `_.toPairsIn`
- `_.equals` is an alias of `_.isEqual`
- `_.extend` is an alias of `_.assignIn`
- `_.extendAll` is an alias of `_.assignInAll`
- `_.extendAllWith` is an alias of `_.assignInAllWith`
- `_.extendWith` is an alias of `_.assignInWith`
- `_.first` is an alias of `_.head`

- `_.identical` is an alias of `_.eq`
- `_.indexBy` is an alias of `_.keyBy`
- `_.init` is an alias of `_.initial`
- `_.invertObj` is an alias of `_.invert`
- `_.juxt` is an alias of `_.over`
- `_.matches` is an alias of `_.isMatch`
- `_.nAry` is an alias of `_.ary`
- `_.omitAll` is an alias of `_.omit`
- `_.path` is an alias of `_.get`
- `_.pathEq` is an alias of `_.matchesProperty`
- `_.pathOr` is an alias of `_.getOr`
- `_.paths` is an alias of `_.at`
- `_.pickAll` is an alias of `_.pick`
- `_.pipe` is an alias of `_.flow`
- `_.pluck` is an alias of `_.map`
- `_.prop` is an alias of `_.get`
- `_.propEq` is an alias of `_.matchesProperty`
- `_.propOr` is an alias of `_.getOr`
- `_.property` is an alias of `_.get`
- `_.props` is an alias of `_.at`
- `_.symmetricDifference` is an alias of `_.xor`
- `_.symmetricDifferenceBy` is an alias of `_.xorBy`
- `_.symmetricDifferenceWith` is an alias of `_.xorWith`
- `_.takeLast` is an alias of `_.takeRight`
- `_.takeLastWhile` is an alias of `_.takeRightWhile`
- `_.unapply` is an alias of `_.rest`
- `_.unnest` is an alias of `_.flatten`
- `_.useWith` is an alias of `_.overArgs`
- `_.where` is an alias of `_.conformsTo`
- `_.whereEq` is an alias of `_.isMatch`
- `_.zipObj` is an alias of `_.zipObject`

Placeholders

The placeholder argument, which defaults to `_`, may be used to fill in method arguments in a different order. Placeholders are filled by the first available arguments of the curried returned function.

```
// The equivalent of `2 > 5`.
_.gt(2)(5);
// ➜ false

// The equivalent of `_.gt(5, 2)` or `5 > 2`.
_.gt(_, 2)(5);
// ➜ true
```

Chaining

The `lodash/fp` module **does not** convert chain sequence method on using functional composition as an alternative to method chaining.

Convert

Although `lodash/fp` and its method modules come pre-converted, there are times when you may want to customize the conversion. That's when the `convert` method comes in handy.

```
// Every option is `true` by default.
var _fp = fp.convert({
  // Specify capping iteratee arguments.
  'cap': true,
  // Specify currying.
  'curry': false,
  // Specify fixed arity.
  'fixed': false,
  // Specify immutable operations.
  'immutable': false,
  // Specify rearranging arguments.
  'rearg': false
});

// The `convert` method is available on each method too.
var mapValuesWithKey = fp.mapValues.convert({ 'cap': false });

// Here's an example of disabling iteratee argument caps to access the `key` param.
mapValuesWithKey(function(value, key) {
  return key == 'a' ? -1 : value;
})({ 'a': 1, 'b': 1 });
// => { 'a': -1, 'b': 1 }
```

Manual conversions are also possible with the `convert` module.

```
var convert = require('lodash/fp/convert');

// Convert by name.
var assign = convert('assign', require('lodash.assign'));

// Convert by object.
var fp = convert({
  'assign': require('lodash.assign'),
  'chunk': require('lodash.chunk')
});

// Convert by `lodash` instance.
var fp = convert(lodash.runInContext());
```

Tooling

Use eslint-plugin-lodash-fp to help use `lodash/fp` more efficiently.