\_.map *=* \_.collect *=* function(obj, iteratee, context) {

iteratee *=* cb(iteratee, context);

var keys *=* *!*isArrayLike(obj) *&&* \_.keys(obj),

length *=* (keys *||* obj).length,

results *=* Array(length);

*for* (var index *=* 0; index *<* length; index*++*) {

var currentKey *=* keys *?* keys[index] *:* index;

results[index] *=* iteratee(obj[currentKey], currentKey, obj);

}

*return* results;

};

cb(iteratee, context);

this creates a callback function with context

ex :

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

=> [3, 6, 9]

Value is the passed function , context is “undefined” in this case

var cb *=* function(value, context, argCount) {

*if* (value *==* null) *return* \_.identity;

*if* (\_.isFunction(value)) *return* optimizeCb(value, context, argCount);

*if* (\_.isObject(value)) *return* \_.matcher(value);

*return* \_.property(value);

};

\_.keys(obj),

\_.keys *=* function(obj) {

*if* (*!*\_.isObject(obj)) *return* [];

*if* (nativeKeys) *return* nativeKeys(obj);

var keys *=* [];

*for* (var key *in* obj) *if* (\_.has(obj, key)) keys.push(key);

*// Ahem, IE < 9.*

*if* (hasEnumBug) collectNonEnumProps(obj, keys);

*return* keys;

};

results[index] *=* iteratee(obj[currentKey], currentKey, obj);

2) reduce

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

=> 6

Concat

log([4, 5].concat([2, 3]).concat([0, 1]));

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

3) find

\_.find(['a', 'b', 3, 'd'], \_.isNumber); //=> 3

4) problem with terniary operator

var arr1 *=* [1, 2, 3, 4, 5, 6];

var find *=* function(arr, fn) {

var res *=* [];

arr.forEach(function(element) {

res.push(fn(element) *?* element *:* "");

});

*return* res;

};

var even *=* find([1, 2, 3, 4, 5, 6], function(num) {

*return* num *%* 2 *==* 0;

});

log(even);

problem : [ '', 2, '', 4, '', 6 ]

solution :

*if* (fn(element)) res.push(element);

5)partition

var pass *=* [], fail *=* [];

\_.partition *=* function(obj, predicate, context) {

predicate *=* cb(predicate, context);

var pass *=* [], fail *=* [];

\_.each(obj, function(value, key, obj) {

(predicate(value, key, obj) *?* pass *:* fail).push(value);

});

*return* [pass, fail];

};

6)

\_.without([1, 2, 1, 0, 3, 1, 4], 0, 1);

=> [2, 3, 4]

arguments = [1, 2, 1, 0, 3, 1, 4], 0, 1

slice.call(arguments, 1) // [0, 1]

\_.without *=* function(array) {

//[1, 2, 1, 0, 3, 1, 4] is array

// arguments is [1, 2, 1, 0, 3, 1, 4], 0, 1

*return* \_.difference(array, slice.call(arguments, 1));

};

\_.difference *=* function(array) {

var rest *=* flatten(arguments, true, true, 1);

*return* \_.filter(array, function(value){

*return* *!*\_.contains(rest, value);

});

};