Please answer the following questions:

Choose which version you prefer

○
```
shuffle(obj) {
    var set = isArrayLike(obj) ? obj : _.values(obj);
    var length = set.length;
    var shuffled = Array(length);
    for (var index = 0, rand; index < length; index++) {
      rand = _.random(0, index);
      if (rand !== index) shuffled[index] =
shuffled[rand];
        shuffled[rand] = set[index];
    }
    return shuffled;
}
```

○
```
shuffle(obj) {
    var set = isArrayLike(obj) ? obj : _.values(obj);
    var l = set.length;
    var shuffled = Array(l);
    for (var i= 0, r; i< l; i++) {
      r = _.random(0, i);
      if (r!== i) shuffled[i] = shuffled[r];
        shuffled[r] = set[i];
    }
    return shuffled;
}
```

Choose which version you prefer

○
```
shuffle(obj) {
    var set = isArrayLike(obj) ? obj : _.values(obj);
    var length = set.length;
    var shuffled = Array(length);
    for (var index = 0, rand; index < length; index++) {
      rand = _.random(0, index);
      if (rand !== index) shuffled[index] =
shuffled[rand];
        shuffled[rand] = set[index];
    }
    return shuffled;
}
```

○
```
shuffle(obj) {
    var o = isArrayLike(obj) ? obj : _.values(obj);
    var l = o.length;
    var s = Array(l);
    for (var i= 0, r; i< l; i++) {
      r = _.random(0, i);
      if (r!== i) s[i] = s[r];
        s[r] = o[i];
    }
    return s;
}
```

Choose which version you prefer

○
```
shuffle(obj) {
    var set = isArrayLike(obj) ? obj : _.values(obj);
    var l = set.length;
    var shuffled = Array(l);
    for (var i= 0, r; i< l; i++) {
      r = _.random(0, i);
      if (r!== i) shuffled[i] = shuffled[r];
        shuffled[r] = set[i];
    }
    return shuffled;
}
```

○
```
shuffle(obj) {
    var o = isArrayLike(obj) ? obj : _.values(obj);
    var l = o.length;
    var s = Array(l);
    for (var i= 0, r; i< l; i++) {
      r = _.random(0, i);
      if (r!== i) s[i] = s[r];
        s[r] = o[i];
    }
    return s;
}
```

Choose which version you prefer

```
var compare = function(first,second, firstStack,
secondStack) {
    if (first == null || second == null) return first ===
second;
    if (first !== first) return second !== second;
    return deepCompare (first, second, firstStack,
secondStack);
};

var deepCompare = function(first, second, firstStack,
secondStack) {
  var className = toString.call(first);
  if (className !== toString.call(second)) return false;

   var areArrays = className === '[object Array]';
   if (!areArrays) {
    if (typeof first != 'object' || typeof second !=
'object') return false;
    var firstCtor = first.constructor, secondCtor =
second.constructor;
    if (firstCtor !== secondCtor &&
!(_.isFunction(firstCtor) && firstCtor instanceof
firstCtor &&
                    _.isFunction(secondCtor) &&
secondCtor instanceof secondCtor)
                && ('constructor' in first &&
'constructor' in second)) {
      return false;
    }
   }
```

```
var compare = function(a,b, aStack, bStack) {
    if (a == null || b == null) return a === b;
    if (a !== a) return b !== b;
    return deepCompare (a, b, aStack, bStack);
};

var deepCompare = function(a, b, aStack, bStack) {
  var className = toString.call(a);
  if (className !== toString.call(b)) return false;

   var areArrays = className === '[object Array]';
   if (!areArrays) {
    if (typeof a != 'object' || typeof b != 'object') return
false;
    var aCtor = a.constructor, bCtor = b.constructor;
    if (aCtor !== bCtor && !(_.isFunction(aCtor) &&
aCtor instanceof aCtor &&
                    _.isFunction(bCtor) && bCtor
instanceof bCtor)
                && ('constructor' in a && 'constructor'
in b)) {
      return false;
    }
   }
```

○

```
   firstStack = firstStack || [];
    secondStack = secondStack || [];
    var length = firstStack.length;
    while (length--) {
     if (firstStack[length] === first) return
secondStack[length] === second;
    }
   firstStack.push(first);
    secondStack.push(second);
   if (areArrays) {

    length = first.length;
     if (length !== second.length) return false;
    while (length--) {
      if (!compare(first[length], second[length],
firstStack, secondStack)) return false;
     }
    } else {
    var keys = _.keys(first), key;
     length = keys.length;
    if (_.keys(second).length !== length) return false;
     while (length--) {
      key = keys[length];
      if (!(_.has(second, key) && compare(first[key],
second[key], firstStack, secondStack))) return false;
     }
    }
   firstStack.pop();
    secondStack.pop();
    return true;
  };
```

```
    aStack = aStack || [];
     bStack = bStack || [];
     var length = aStack.length;
○    while (length--) {
      if (aStack[length] === a) return bStack[length] ===
b;
     }
     aStack.push(a);
     bStack.push(b);
     if (areArrays) {

      length = a.length;
       if (length !== b.length) return false;
      while (length--) {
        if (!compare(a[length], b[length], aStack,
bStack)) return false;
       }
      } else {
       var keys = _.keys(a), key;
        length = keys.length;
       if (_.keys(b).length !== length) return false;
        while (length--) {
         key = keys[length];
          if (!(_.has(b, key) && compare(a[key], b[key],
aStack, bStack))) return false;
        }
       }
      aStack.pop();
       bStack.pop();
       return true;
     };
```