# AJAX with jQuery

stands for Asynchronous Javascript And XML (XML - stands for Extensible Markup Language)

a technique for handling external data through Javascript asynchronously without reloading the entire page.

## How to use it

There are so many ways. The most flexible is using a function called $.ajax... $.ajax has been around since jQuery 1.0. All of the other AJAX functions that you can use with jQuery use this behind the scenes.

Preferred way!

$.ajax({ OPTIONS });

We could also use promises for this, instead of encapsulating the handlers (success, error etc.), we can chain functions and jQuery promises us that these will get run.

$.ajax({...}).always(function () {

// This will always run

}).done( function (data) {

// This will run on success

}).fail( function (data) {

// This will run if there is an error

});

For more information, see [here](https://learn.jquery.com/ajax/jquery-ajax-methods/) and [here](http://api.jquery.com/jquery.ajax/).

jQuery also provides lots of convenience methods if you don't need something quite as customizable as $.ajax. These include:

[$.get](http://api.jquery.com/jQuery.get/)

[$.getJSON](http://api.jquery.com/jQuery.getJSON/)

[$.getScript](http://api.jquery.com/jQuery.getScript/)

[$.post](http://api.jquery.com/jQuery.post/)

[$.load](http://api.jquery.com/load/)

They are worth getting to know.

[Check this thing out!!!](http://jqfundamentals.com/chapter/ajax-deferreds)

## Same Origin Policy

Same Origin Policy - your javascript can only talk to the current domain (the server that this site is loaded from). If you try and talk to another domain, it might respond with a 'No Cross-Origin Resource Sharing' error (or CORS).

There are three ways to deal with this:

* Allow CORS (like OMDB API)
* Support JSONP
* The site can host your code

This is called a CORS error, and is a really hard one to deal with.

# Underscore

## What is Underscore?

toolbelt

a whole range of useful utility functions that you will end up using thousands and thousands of times. It was developed by a guy called [Jeremy Ashkenas](https://twitter.com/jashkenas) (who also made Backbone and Coffeescript).

## Using it in Rails

* Add this to your Gemfile - gem 'underscore-rails'
* Run bundle
* Add this line before the require tree stuff - //= require underscore

## Common Things with Underscore

Best to download this from [here](https://github.com/ga-wolf/WDI9_Homework/tree/master/warmup_exercises/week_07/underscore-stuff), and have a muck around with it.

// ////////////////////////////////// //

// Common Methods - Arrays or Objects //

// ////////////////////////////////// //

## // Each

var arr = [ 1, 2, 3 ];

\_.each( arr, function ( element, index, entire\_array ) {

console.log( "\tElement: ", element );

console.log( "\tIndex: ", index );

console.log( "\tEntire Array: ", entire\_array );

} );

console.log("");

console.log( "UNDERSCORE EACH WITH OBJECT" );

var obj = {

four: 4,

five: 5,

nine: 9

}

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

console.log( "\tValue: ", value );

console.log( "\tKey: ", key );

console.log( "\tEntire Object: ", entire\_object );

} );

console.log("");

## // Map

// Iterates through each thing in the passed in collection and passes back an altered collection.

// \_.map( collection, iteratee\_function );

// The iteratee function receives an element, an index, and en entire collection as parameters

console.log( "UNDERSCORE MAP WITH ARRAY" );

var arr = [ 1, 2, 3 ];

var arrByFive = \_.map( arr, function ( element, index, entire\_array ) {

return element \* 5;

} );

console.log( "\tOriginal Array: ", arr );

console.log( "\tArray By Five: ", arrByFive );

console.log( "" );

console.log( "UNDERSCORE MAP WITH OBJECT" );

var obj = {

one: 1,

two: 2,

three: 3

};

var objByFive = \_.map( obj, function ( element, index, entire\_array ) {

return element \* 5;

} );

console.log( "\tOriginal Object: ", arr );

console.log( "\tObject By Five (turned into an array by map): ", objByFive );

console.log( "" );

## // Reduce

// Iterates through each thing in the passed in collection and returns a single sum.

// \_.reduce( collection, iteratee\_function, starting\_value );

// The iteratee function receives a sum, a current value, and en entire collection as parameters

console.log( "UNDERSCORE REDUCE WITH ARRAY" );

var arr = [ 1, 2, 3 ];

var reducedArr = \_.reduce( arr, function ( sum, value, index, list ) {

return sum, value;

}, 0 );

console.log( "\tOriginal Array: ", arr );

console.log( "\tReduced Array (to a sum): ", reducedArr );

console.log( "" );

console.log( "UNDERSCORE REDUCE WITH OBJECT" );

var obj = {

one: 1,

two: 2,

three: 3

};

var reducedArr = \_.reduce( arr, function ( sum, value, index, list ) {

return sum, value;

}, 0 );

console.log( "\tOriginal Object: ", obj );

console.log( "\tReduced Object: ", reducedArr );

console.log( "" );

## // Find

// Iterates through each thing in the passed in collection and returns true in the passed in function.

// \_.find( collection, iteratee\_function );

// The iteratee function receives a current value as a parameter

console.log( "UNDERSCORE FIND WITH ARRAY" );

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

var firstEven = \_.find( arr, function ( num ) {

return num % 2 === 0;

} );

console.log( "\tOriginal Array: ", arr );

console.log( "\tFirst Even in the Array: ", firstEven );

console.log( "" );

console.log( "UNDERSCORE FIND WITH OBJECT" );

var obj = {

one: 1,

two: 2,

three: 3

};

var firstEven = \_.find( obj, function ( num ) {

return num % 2 === 0;

} );

console.log( "\tOriginal Object: ", arr );

console.log( "\tFirst Even Value in the Object: ", firstEven );

console.log( "" );

## // Filter

// Iterates through each thing in the passed in collection and returns everything that returns true in the passed in function.

// \_.filter( collection, iteratee\_function );

// The iteratee function receives a current value as a parameter

console.log( "UNDERSCORE FILTER WITH ARRAY" );

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

var allEvens = \_.filter( arr, function ( num ) {

return num % 2 === 0;

} );

console.log( "\tOriginal Array: ", arr );

console.log( "\tAll Evens in the Array: ", allEvens );

console.log( "" );

console.log( "UNDERSCORE FILTER WITH OBJECT" );

var obj = {

one: 1,

two: 2,

three: 3,

four: 4

};

var allEvens = \_.filter( obj, function ( num ) {

return num % 2 === 0;

} );

console.log( "\tOriginal Array: ", obj );

console.log( "\tAll Evens in the Array: ", allEvens );

console.log( "" );

## // Where

// Iterates through each thing in the passed in collection and returns everything that has the same key and values

// \_.where( collection, object );

console.log( "UNDERSCORE WHERE WITH ARRAY FILLED WITH OBJECTS" );

var books = [{

author : "Gustave Flaubert",

title : "Sentimental Education"

}, {

author : "Marie-Henri Beyle",

title : "Lucien Leuwen"

}];

var gustave = \_.where( books, { author: "Gustave Flaubert" } );

console.log( "\tOriginal Books Array: ", books );

console.log( "\tGustave Flaubert's Books: ", gustave );

console.log( "" );

## // FindWhere

// Same as where except returns the first one.

// Reject

// Iterates through each thing in the passed in collection and removes everything that returns true in the passed in function.

// \_.reject( collection, iteratee\_function );

// The iteratee function receives a current value as a parameter

console.log( "UNDERSCORE REJECT WITH ARRAY" );

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

var odds = \_.reject( arr, function ( num ) {

return num % 2 === 0;

} );

console.log( "\tOriginal Array: ", arr );

console.log( "\tOdds Array: ", odds );

console.log( "" );

// Contains

// Iterates through each thing in the passed in collection and returns true if it has the passed in value

// \_.contain( collection, value );

console.log( "UNDERSCORE CONTAIN WITH ARRAY" );

var arr = [ 1, 2, 3 ];

var containsThree = \_.contains( arr, 3 );

console.log( "\tOriginal Array: ", arr );

console.log( "\tIt contains three? ", containsThree );

console.log( "" );

## // Pluck

// Iterates through each thing in the passed in collection and returns just the requested key

// \_.pluck( collection, key );

console.log( "UNDERSCORE PLUCK WITH ARRAY OF OBJECTS" );

var books = [{

author : "Gustave Flaubert",

title : "Sentimental Education"

}, {

author : "Marie-Henri Beyle",

title : "Lucien Leuwen"

}];

var authors = \_.pluck( books, 'author' );

console.log( "\tOriginal Books Array: ", books );

console.log( "\tAll Authors: ", authors );

console.log( "" );

## // Max

// Returns the maximum value in the array

// \_.max( collection );

console.log( "UNDERSCORE MAX WITH ARRAY" );

var arr = [ 1, 292898, 4, 232.223 ];

var maxNum = \_.max( arr );

console.log( "\tOriginal Array: ", arr );

console.log( "\tMaximum Number: ", maxNum );

console.log( "" );

console.log( "UNDERSCORE MAX WITH OBJECT" );

var people = [{

name: "Marcel",

age: Infinity

}, {

name: "Roger",

age: 34

}];

var oldestPerson = \_.max( people, function ( person ) {

return person.age;

} );

console.log( "\tOriginal Array: ", people );

console.log( "\tOldest Person: ", oldestPerson );

console.log( "" );

## // Min

// Returns the minimum value in the array

// \_.min( collection );

console.log( "UNDERSCORE MIN WITH ARRAY" );

var arr = [ 0.1, 292898, 4, 232.223 ];

var maxNum = \_.min( arr );

console.log( "\tOriginal Array: ", arr );

console.log( "\tMinimum Number: ", maxNum );

console.log( "" );

console.log( "UNDERSCORE MIN WITH OBJECT" );

var people = [{

name: "Marcel",

age: Infinity

}, {

name: "Roger",

age: 34

}];

var youngestPerson = \_.min( people, function ( person ) {

return person.age;

} );

console.log( "\tOriginal Array: ", people );

console.log( "\tYoungest Person: ", youngestPerson );

console.log( "" );

## // SortBy

// Iterates through each item in the collection and sorts them using the given function

// \_.sortBy( collection, iteratee\_function )

console.log( "UNDERSCORE SORTBY WITH ARRAY" );

var arr = [ 0.1, 292898, 4, 232.223 ];

var sortedArray = \_.sortBy( arr, function (num) {

return num;

} );

console.log( "\tOriginal Array: ", arr );

console.log( "\tSorted Array: ", sortedArray );

console.log( "" );

console.log( "UNDERSCORE SORTBY WITH OBJECT" );

var people = [{

name: "Marcel",

age: Infinity

}, {

name: "Roger",

age: 34

}];

var youngestPerson = \_.sortBy( people, "age" );

console.log( "\tOriginal Array: ", people );

console.log( "\tSorted by Age: ", youngestPerson );

console.log( "" );

## // Shuffle

// Shuffles the given collection

// \_.shuffle( collection );

console.log( "UNDERSCORE SHUFFLE WITH ARRAY" );

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

var shuffledArr = \_.shuffle( arr );

console.log( "\tOriginal Array: ", arr );

console.log( "\tShuffled Array: ", shuffledArr );

console.log( "" );

## // Sample

// Picks a number (default 1) of elements from the given collection

// \_.sample( collection, to\_return );

console.log( "UNDERSCORE SAMPLE WITH ARRAY" );

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

var sample = \_.sample( arr );

var threeSample = \_.sample( arr, 3 );

console.log( "\tOriginal Array: ", arr );

console.log( "\tSample: ", sample );

console.log( "\tThree Sampled: ", threeSample );

console.log( "" );

// /////////////////////// //

## // Common Methods - Arrays //

// /////////////////////// //

### // First

// Returns the first number of element(s) in the collection

// \_.first( collection, to\_pick );

console.log( "UNDERSCORE FIRST WITH ARRAY" );

var arr = [ 1, 2, 3, 4, 5 ];

var firstOne = \_.first( arr );

var firstThree = \_.first( arr, 3 );

console.log( "\tOriginal Array: ", arr );

console.log( "\tFirst One: ", firstOne );

console.log( "\tFirst Three: ", firstThree );

console.log( "" );

### // Last

// Returns the last number of element(s) in the collection

// \_.last( collection, to\_pick );

console.log( "UNDERSCORE LAST WITH ARRAY" );

var arr = [ 1, 2, 3, 4, 5 ];

var lastOne = \_.last( arr );

var lastThree = \_.last( arr, 3 );

console.log( "\tOriginal Array: ", arr );

console.log( "\tLast One: ", lastOne );

console.log( "\tLast Three: ", lastThree );

console.log( "" );

### // Compact

// Removes all falsey values in an array

// \_.compact( collection );

console.log( "UNDERSCORE COMPACT WITH ARRAY" );

var arr = [ 0, 1, false, 2, '', 3, undefined, NaN ];

var compactedArray = \_.compact( arr );

console.log( "\tOriginal Array: ", arr );

console.log( "\tCompacted Array: ", compactedArray );

console.log( "" );

### // Flatten

// Turns an array of arrays into one flat array ( can be specified to just flatten to one level )

// \_.flatten( arr, flatten\_to\_one\_level\_true\_or\_false );

console.log( "UNDERSCORE FLATTEN WITH ARRAY" );

var arr = [ [1], [2], [[[1]]] ];

var flattenedArray = \_.flatten( arr );

var flattenedArrayToOneLevel = \_.flatten( arr, true );

console.log( "\tOriginal Array: ", arr );

console.log( "\tFlattened Array: ", flattenedArray );

console.log( "\tFlattened Array (to one level): ", flattenedArrayToOneLevel );

console.log( "" );

### // Without

// Returns the array without the specified pieces

// \_.without( collection, remove\_this, remove\_this... );

console.log( "UNDERSCORE WITHOUT WITH ARRAY" );

var arr = [ 1, 2, 1, 0, 3, 1, 4 ];

var withoutOnes = \_.without( arr, 1 );

var withoutOnesAndTwos = \_.without( arr, 1, 2 );

console.log( "\tOriginal Array: ", arr );

console.log( "\tWithout Ones Array: ", withoutOnes );

console.log( "\tWithout Ones and Twos Array: ", withoutOnesAndTwos );

console.log( "" );

### // Union

// Returns unique values from all given arrays in an array

// \_.union( collection, collection, collection );

console.log( "UNDERSCORE UNION WITH ARRAY" );

var arr1 = [ 1, 2, 3 ];

var arr2 = [ 101, 22, 303.2 ];

var arr3 = [ 1, 2 ];

var uniqItems = \_.union( arr1, arr2, arr3 );

console.log( "\tArray 1: ", arr1, " Array 2: ", arr2, " Array 3: ", arr3 );

console.log( "\tUnique Items: ", uniqItems );

console.log( "" );

### // Intersection

// Returns values that are present in all given arrays as an array

// \_.intersection( collection, collection, collection );

console.log( "UNDERSCORE INTERSECTION WITH ARRAY" );

var arr1 = [ 1, 2, 3 ];

var arr2 = [ 101, 2, 1, 10 ];

var arr3 = [ 2, 1 ];

var intersectedItems = \_.intersection( arr1, arr2, arr3 );

console.log( "\tArray 1: ", arr1, " Array 2: ", arr2, " Array 3: ", arr3 );

console.log( "\tItems in all the Arrays: ", intersectedItems );

console.log( "" );

### // Uniq

// Returns just unique values from the given array

// \_.uniq( arr );

console.log( "UNDERSCORE UNIQ WITH ARRAY" );

var arr = [ 1, 2, 1, 4, 1, 3 ];

var uniqueItems = \_.uniq( arr );

console.log( "\tOriginal Array: ", arr );

console.log( "\tUnique Items in Array Above: ", uniqueItems );

console.log( "" );

### // Zip

// Creates an array of arrays but changes the placement of the elements. Will make an array of the first elements, then all the second elements etc. Will return undefined if there are more elements in the first array

// \_.zip( first\_array, other\_array, other\_array );

console.log( "UNDERSCORE ZIP WITH ARRAYS" );

var arr1 = [ 'moe', 'larry', 'curly' ];

var arr2 = [ 30, 40, 50 ];

var arr3 = [ true, false, false ];

var zippedArrays = \_.zip( arr1, arr2, arr3 );

console.log( "\tArray 1: ", arr1, " Array 2: ", arr2, " Array 3: ", arr3 );

console.log( "\tZipped Arrays: ", zippedArrays );

console.log( "" );

// Object

// Makes an object with the key coming from the first array and the value coming from the second array. Keeps going like that

// \_.object( arr1, arr2 );

console.log( "UNDERSCORE OBJECT WITH ARRAYS" )

var arr1 = [ "moe", "larry", "curly" ];

var arr2 = [ 30, 40, 50 ];

var createdObject = \_.object( arr1, arr2 );

console.log( "\tArray 1: ", arr1, " Array 2: ", arr2 );

console.log( "\tCreated Object: ", createdObject );

console.log( "" );

### // IndexOf

// Returns the index of the specified value. Will return -1 if it isn't present

// \_.indexOf( collection, target );

console.log( "UNDERSCORE INDEXOF WITH ARRAYS" );

var target = 2;

var arr = [ 1, 2, 3 ];

var indexOfTarget = \_.indexOf( arr, target );

console.log( "\tTarget to find index of: ", target, " Array: ", arr );

console.log( "\tIndex of Target: ", indexOfTarget );

console.log( "" );

### // Range

// Creates an array using a range

// \_.range( starting\_value, ending\_value, step );

console.log( "UNDERSCORE RANGE WITH ARRAYS" );

console.log( "\tPassing in 10 to range: ", \_.range( 10 ) );

console.log( "\tPassing in 10 and 20 to range: ", \_.range( 10, 20 ) );

console.log( "\tPassing in -1, -11, and -1 to range: ", \_.range( -1, -11, -1 ) );

console.log( "" );

////////////////////////////////

## // Common Methods - Objects //

//////////////////////////////

### // Keys

// Returns an array of keys

// \_.keys( collection );

console.log( "UNDERSCORE KEYS WITH OBJECTS" );

var obj = {

one: 1,

two: 2,

six: 6

};

var objectKeys = \_.keys( obj );

console.log( "\tOriginal Object: ", obj );

console.log( "\tObject Keys: ", objectKeys );

console.log( "" );

### // Values

// Returns an array of values

// \_.values( collection );

console.log( "UNDERSCORE VALUES WITH OBJECTS" );

var obj = {

one: 1,

two: 2,

six: 6

};

var objectKeys = \_.values( obj );

console.log( "\tOriginal Object: ", obj );

console.log( "\tObject Values: ", objectKeys );

console.log( "" );

### // Pairs

// Returns an array of arrays with the key being the first element and the value being the second element

// \_.pairs( collection );

console.log( "UNDERSCORE PAIRS WITH OBJECTS" );

var obj = {

one: 1,

two: 2,

six: 6

};

var objectKeys = \_.pairs( obj );

console.log( "\tOriginal Object: ", obj );

console.log( "\tObject Pairs in Array Form: ", objectKeys );

console.log( "" );

### // Invert

// Returns the opposite object. Keys become values

// \_.invert( collection );

console.log( "UNDERSCORE PAIRS WITH OBJECTS" );

var obj = {

one: 1,

two: 2,

six: 6

};

var objectKeys = \_.invert( obj );

console.log( "\tOriginal Object: ", obj );

console.log( "\tInverted Object: ", objectKeys );

console.log( "" );

### // Pick

// Returns an object with just the passed in keys (white listing)

// \_.pick( collection, key\_to\_keep, key\_to\_keep );

console.log( "UNDERSCORE PICK WITH OBJECTS" );

var obj = {

name: "Moe",

age: 50,

userID: 142423

}

var whiteListedObject = \_.pick( obj, "name", "age" );

var functionWhiteListedObject = \_.pick( obj, function ( value, key, object ) {

return \_.isNumber( value ); // Returns true if it is a number

} );

console.log( "\tOriginal Object: ", obj );

console.log( "\tWhite Listed Object (passing in keys): ", whiteListedObject );

console.log( "\tWhite Listed Object (passing in a function that returns a boolean): ", functionWhiteListedObject );

console.log( "" );

### // Omit

// Returns an object without the passed in keys (black listing)

// \_.omit( collection, key\_to\_remove, key\_to\_remove );

console.log( "UNDERSCORE OMIT WITH OBJECTS" );

var obj = {

name: "Moe",

age: 50,

userID: 142423

}

var omittedKeys = \_.omit( obj, 'name' );

var omittedKeysWithFunction = \_.omit( obj, function ( value, key, object ) {

return \_.isNumber( value );

} );

console.log( "\tOriginal Object: ", obj );

console.log( "\tOmitted Keys (specified with key): ", omittedKeys );

console.log( "\tOmitted Keys (specified by a function): ", omittedKeysWithFunction );

console.log( "" );

### // Has

// Returns true if the object has the specified key

// \_.has( collection, key );

console.log( "UNDERSCORE HAS WITH OBJECTS" );

var obj = {

name: "Moe",

age: 50,

userID: 142423

}

var hasName = \_.has( obj, "name" );

console.log( "\tOriginal Object: ", obj );

console.log( "\tDid it have the name key? ", hasName );

console.log( "" );

// //////////////// //

## // Common Functions //

// //////////////// //

## // Delay

// Calls a function after a specified amount of time

// \_.delay( function, time\_in\_ms );

console.log( "UNDERSCORE DELAY WITH FUNCTION" );

var toCall = function () {

console.log( "\tDelayed function Called." )

}

\_.delay( toCall, 1000 );

console.log( "" );

### // Throttle

// Says that the specified function can be called only every so often (will call it straight away when it reads this line). Throttle guarantees that the given function actually runs

// \_.throttle( function, time\_in\_ms );

console.log( "UNDERSCORE THROTTLE WITH FUNCTION" );

var showScrollTop = function () {

var scrollTop = $( window ).scrollTop();

console.log( "\tScroll Top is: ", scrollTop );

}

var throttledShowScrollTop = \_.throttle( showScrollTop, 100 );

$( window ).scroll( throttledShowScrollTop );

console.log( "" );

// This function can only call every 100 milliseconds but will call straight away when defined

### // Debounce

// Says that the specified function can be called only every so often (won't call straight away)

// \_.debounce( function, time\_in\_ms );

console.log( "UNDERSCORE DEBOUNCE WITH FUNCTION" );

var calculateLayout = function () {

var windowWidth = $( window ).width();

console.log( "\tWindow Width is: ", windowWidth );

}

var debouncedCalculateLayout = \_.debounce( calculateLayout, 300 );

$(window).resize( debouncedCalculateLayout );

// This function can only call every 300 ms but won't call straight away

### // Once

// Says that the specified function can be called only once

// \_.once( function );

console.log( "" );

console.log( "UNDERSCORE ONCE WITH FUNCTION" );

var createApplication = function () {

console.log( "\tCreate Application called." );

}

var initialize = \_.once( createApplication );

initialize(); // This will call

initialize(); // This won't call

console.log( "" );

### // Times

// Call the passed in function a specified amount of times (receives an index as a parameter)

// \_.times( num\_of\_times, function );

console.log( "UNDERSCORE TIMES WITH FUNCTION" );

\_.times( 3, function ( index ) {

console.log( "\tIndex: ", index );

} );

console.log( "" );

### // Random

// Generates a random value between 0 and the passed in number if just one value is passed in. Or between the first and second values. Best to be explicit and pass in 0 if necessary

// \_.random( starting\_point, ending\_point );

console.log( "UNDERSCORE RANDOM WITH FUNCTION" );

var upToOneHundred = \_.random( 100 );

var betweenOneHundredAndTwoHundred = \_.random( 100, 200 );

var betweenOneHundredAndMinusTwoHundred = \_.random( 100, -200 );

var betweenMinusOneHundredAndMinusTwoHundred = \_.random( -100, -200 );

console.log( "\tUp to 100: ", upToOneHundred );

console.log( "\tBetween 100 and 200: ", betweenOneHundredAndTwoHundred );

console.log( "\tBetween 100 and -200: ", betweenOneHundredAndMinusTwoHundred );

console.log( "\tBetween -100 and -200: ", betweenMinusOneHundredAndMinusTwoHundred );

console.log( "" );

// ///////////////// //

## // Predicate Methods // (TRue or False)

// ///////////////// //

## // isEqual

// Checks whether two collections are equal

// \_.isEqual( collection\_one, collection\_two );

console.log( "UNDERSCORE ISEQUAL WITH COLLECTION" );

var arr1 = [ 0, 1, 2 ];

var arr2 = [ 0, 1, 2 ];

var returnedEquals = arr1 === arr2; // Returns false

var returned = \_.isEqual( arr1, arr2 ); // Returns true

console.log( "\tThing 1: ", arr1, " Thing 2: ", arr2 );

console.log( "\tThing 1 and Thing 2 compared with three equals: ", returnedEquals );

console.log( "\tThing 1 and Thing 2 compared with isEqual: ", returned );

console.log( "" );

## // isMatch

// Tells you if the keys and values in properties are contained in object.

// \_.isMatch( collection, obj );

console.log( "UNDERSCORE ISMATCH WITH OBJECT" );

var obj = {

name: "Roger"

};

var matched = \_.isMatch( obj, { name: "Roger" } ); // Returns true

console.log( "\tObject: ", obj );

console.log( "\tObject has a name: ", matched );

console.log( "" );

## // isEmpty

// Returns true if there is nothing in the array or the object

// \_.isEmpty( thing );

console.log( "UNDERSCORE ISEMPTY WITH COLLECTION" );

var emptyArr = [];

var notEmptyObj = {

name: "Roger"

};

var emptyArrMethod = \_.isEmpty( emptyArr ); // Returns true

var filledObjMethod = \_.isEmpty( notEmptyObj ); // Returns false

console.log( "\tArray is: ", emptyArr, ". Is it empty? ", emptyArrMethod );

console.log( "\tObject is: ", notEmptyObj, ". Is it empty? ", filledObjMethod );

console.log( "" );

## // isArray

// Returns true if it is an array

// \_.isArray( thing );

console.log( "UNDERSCORE ISARRAY WITH THING" );

var arr = [];

var obj = {};

var arrMethod = \_.isArray( arr ); // Returns true

var objMethod = \_.isArray( obj ); // Returns false

console.log( "\tThing 1: ", arr, ". Is it an array? ", arrMethod );

console.log( "\tThing 2: ", obj, ". Is it an array? ", objMethod );

console.log( "" );

## // isObject

// Returns true if it is an object

// \_.isObject( thing );

console.log( "UNDERSCORE ISOBJECT WITH THING" );

var arr = [];

var obj = {};

var arrMethod = \_.isObject( arr ); // Returns false

var objMethod = \_.isObject( obj ); // Returns true

console.log( "\tThing 1: ", arr, ". Is it an object? ", arrMethod );

console.log( "\tThing 2: ", obj, ". Is it an object? ", objMethod );

console.log( "" );

## // isFunction

// Returns true if it is a function

// \_.isFunction( thing );

console.log( "UNDERSCORE ISFUNCTION WITH THING" );

var myFunc = function () {};

var arr = [];

var funcMethod = \_.isFunction( myFunc ); // Returns true

var arrMethod = \_.isFunction( arr ); // Returns false

console.log( "\tThing 1: ", myFunc, ". Is it a function? ", funcMethod );

console.log( "\tThing 2: ", arr, ". Is it a function? ", arrMethod );

console.log( "" );

## // isString

// Returns true if it is a string

// \_.isString( thing );

console.log( "UNDERSCORE ISSTRING WITH THING" );

var myFunc = function () {};

var str = "";

var funcMethod = \_.isFunction( myFunc ); // Returns false

var strMethod = \_.isFunction( str ); // Returns true

console.log( "\tThing 1: ", myFunc, ". Is it a string? ", funcMethod );

console.log( "\tThing 2: ", str, ". Is it a string? ", strMethod );

console.log( "" );

## // isNumber

// Returns true if it is a number

// \_.isNumber( thing );

console.log( "UNDERSCORE ISNUMBER WITH THING" );

var myFunc = function () {};

var num = 1;

var funcMethod = \_.isNumber( myFunc ); // Returns false

var numMethod = \_.isNumber( num ); // Returns true

console.log( "\tThing 1: ", myFunc, ". Is it a number? ", funcMethod );

console.log( "\tThing 2: ", num, ". Is it a number? ", numMethod );

console.log( "" );

## // isFinite

// Returns true if it is a finite value

// \_.isFinite( thing );

console.log( "UNDERSCORE ISFINITE WITH COLLECTION" );

var infiniteThing = -Infinity;

var finiteThing = 1;

var infiniteMethod = \_.isFinite( infiniteThing ); // Returns false

var finiteMethod = \_.isFinite( finiteThing ); // Returns true

console.log( "\tThing 1: ", infiniteThing, ". Is it finite? ", infiniteMethod );

console.log( "\tThing 2: ", finiteThing, ". Is it finite? ", finiteMethod );

console.log( "" );

## // isBoolean

// Returns true if it is a boolean value

// \_.isBoolean( thing );

console.log( "UNDERSCORE ISBOOLEAN WITH COLLECTION" );

var isTrue = true;

var str = "";

var trueMethod = \_.isBoolean( isTrue ); // Returns true

var strMethod = \_.isBoolean( str ); // Returns false

console.log( "\tThing 1: ", isTrue, ". Is it a function? ", trueMethod );

console.log( "\tThing 2: ", str, ". Is it a function? ", strMethod );

console.log( "" );

## // isNaN

// Returns true if it is NaN

// \_.isNaN( thing );

console.log( "UNDERSCORE ISNAN WITH COLLECTION" );

var str = "";

var nope = NaN;

var strMethod = \_.isNaN( str ); // Returns false

var nanMethod = \_.isNaN( nope ); // Returns true

console.log( "\tThing 1: ", str, ". Is it NaN? ", strMethod );

console.log( "\tThing 2: ", nope, ". Is it NaN? ", nanMethod );

console.log( "" );

## // isNull

// Returns true if it is null

// \_.isNull( thing );

console.log( "UNDERSCORE ISNULL WITH COLLECTION" );

var str = "";

var nope = null;

var strMethod = \_.isNull( str ); // Returns false

var nullMethod = \_.isNull( nope ); // Returns true

console.log( "\tThing 1: ", str, ". Is it null? ", strMethod );

console.log( "\tThing 2: ", nope, ". Is it null? ", nullMethod );

console.log( "" );

## // isUndefined

// Returns true if it is undefined

// \_.isUndefined( thing );

console.log( "UNDERSCORE ISUNDEFINED WITH COLLECTION" );

var str = "";

var nope = undefined;

var strMethod = \_.isUndefined( str ); // Returns false

var undefinedMethod = \_.isUndefined( nope ); // Returns true

console.log( "\tThing 1: ", str, ". Is it undefined? ", strMethod );

console.log( "\tThing 2: ", nope, ". Is it undefined? ", undefinedMethod );

console.log( "" );