Module 12 - Day 2: Introduction to Lodash

Lesson 1: Getting Started with Lodash

Source and article:
https://javascript.plainenglish.io/lodash-methods-to-make-your-life-a-breeze-3f5aae11f7

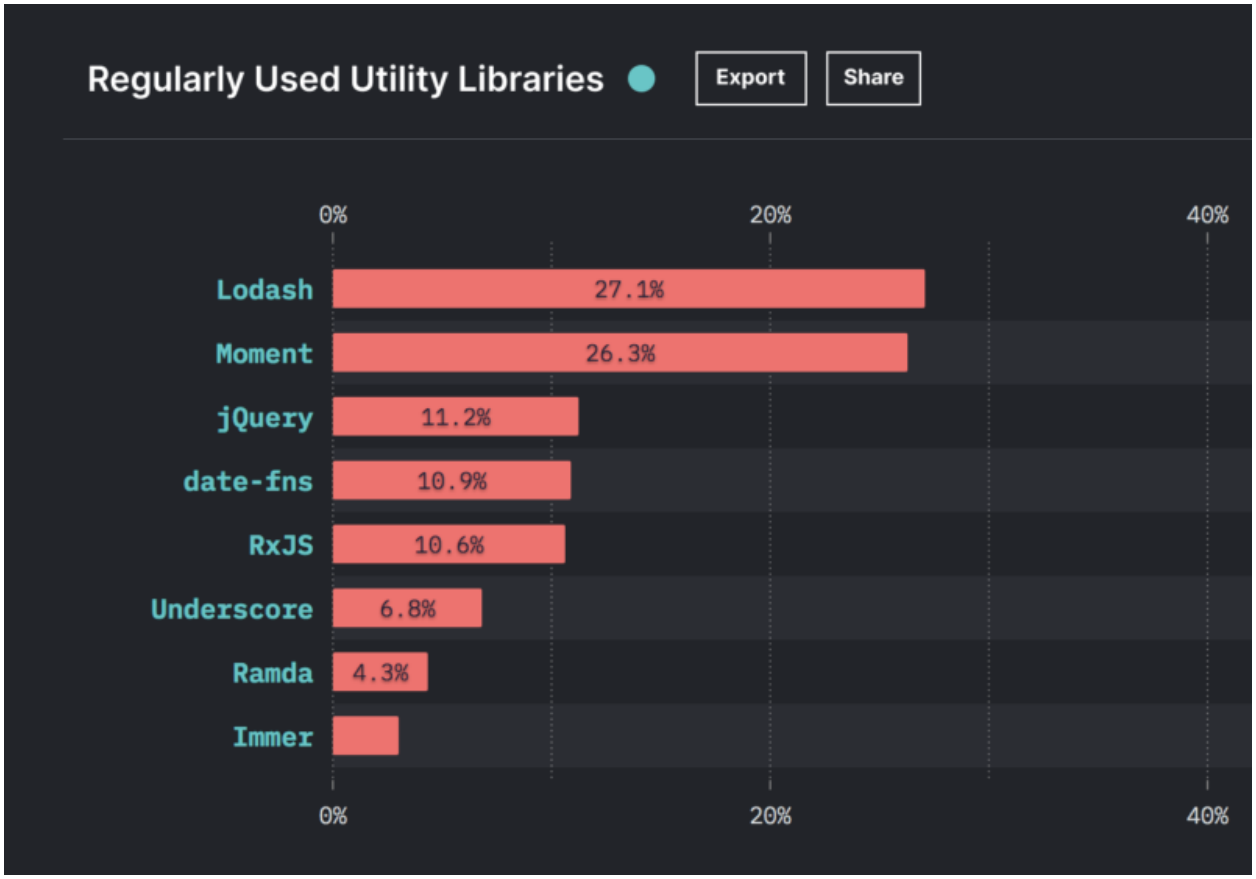
Introduction

Lodash is a modern JavaScript utility library that provides helpful methods for tasks such as working with arrays, numbers, objects, strings, and more. It's like a swiss army knife for your code, providing you with a vast range of tools to solve common programming problems in a more readable and efficient way.

Why use Lodash?

1. **Consistency**: Lodash functions behave consistently, regardless of the browser you're working in, which is not always the case with native JavaScript functions.
2. **Readability**: Lodash methods can often be more readable and expressive than native JavaScript, particularly when working with complex data structures.

3. **Performance**: Lodash methods are generally optimized for performance.
4. **Modularity**: You can import just the Lodash functions you need, which can help keep your codebase lean.



(According to the State of Javascript 2019 Survey results, Source: https://externlabs.com/blogs/10-lodash-functions-everyone-should-know/)

Using Lodash with Other Libraries

Lodash can be used in conjunction with other libraries like jQuery, React, Angular, etc. It doesn't conflict with these libraries but rather complements them by providing additional utility functions.

Lodash vs jQuery

While both Lodash and jQuery are utility libraries, they serve different purposes.

- **jQuery** is primarily a DOM manipulation library. It's great for adding interactive behavior to your web pages, handling events, creating animations, and developing AJAX applications.
- **Lodash**, on the other hand, is an all-purpose utility library. It provides a wealth of functions for manipulating and working with data structures like arrays, objects, and strings. It doesn't include any DOM manipulation functions.

Whether you use Lodash or jQuery depends on the problem you're trying to solve. If you need to manipulate the DOM, jQuery is the tool for the job. If you're working with data structures, Lodash can make your life much easier.

Getting Started with Lodash: CDN

To start using Lodash in your project, you can include it via a Content Delivery Network (CDN). This is the quickest and easiest way to get started:

```
<script src="https://cdn.jsdelivr.net/npm/lodash@4.17.21/lodash.min.js"></script>
```

Lodash Syntax Overview

Lodash functions can be called in a couple of different ways. Here's a simple example of using the `_.map` function to transform an array:

```
var array = [1, 2, 3];

var newArray = _.map(array, function(num) {

   return num * 3;

});

console.log(newArray); // Outputs: [3, 6, 9]
```

In this example, `_.map` is a function provided by Lodash that takes an array and a function as arguments. It applies the function to each item in the array and returns a new array with the results.

Module 12 - Day 2: Lodash Deep Dive Part 1

Lesson 2: Exploring Lodash Array and Object Methods

Introduction

Lodash provides numerous array and object methods that can make your life as a developer much easier. In this lesson, we'll deep dive into some of these methods and understand how they can be used in real-world scenarios.

Array Methods

1. **compact(array)**: This method creates an array with all falsey values removed. The values false, null, 0, "", undefined, and NaN are falsey in JavaScript.

   Use Case: When you want to remove all falsey values from an array.

   ```
   _.compact([0, 1, false, 2, '', 3]);

   // => [1, 2, 3]
   ```

2. **flattenDeep(array)**: Recursively flattens array.

   Use Case: When you have an array of arrays (or deeper nested arrays) and you want to have all the values in a single flat array.

   ```
   _.flattenDeep([1, [2, [3, [4]], 5]]);

   // => [1, 2, 3, 4, 5]
   ```

3. **difference(array, [values])**: Creates an array of array values not included in the other given arrays.

   Use Case: When you want to find values that are in one array but not in others.

   ```
   _.difference([2, 1], [2, 3]);
   ```

```
// => [1]
```

4. **intersection([arrays])**: Creates an array of unique values that are included in all given arrays.

   Use Case: When you want to find common values between arrays.

```
_.intersection([2, 1], [2, 3]);

// => [2]
```

5. **reverse(array)**: Reverses array so that the first element becomes the last, the second element becomes the second to last, and so on.

   Use Case: When you want to reverse the order of elements in an array.

```
_.reverse([1, 2, 3]);

// => [3, 2, 1]
```

6. **sortBy(collection, [iteratees])**: Creates an array of elements, sorted in ascending order by the results of running each element in a collection thru each iteratee.

   Use Case: When you want to sort an array of objects based on one or more properties.

```
var users = [
  { 'user': 'fred', 'age': 48 },
  { 'user': 'barney', 'age': 36 },
  { 'user': 'fred', 'age': 40 },
  { 'user': 'barney', 'age': 34 }
];


_.sortBy(users, ['user', 'age']);

/*
```

```
=> [

{ 'user': 'barney', 'age': 34 },

{ 'user': 'barney', 'age': 36 },

{ 'user': 'fred', 'age': 40 },

{ 'user': 'fred', 'age': 48 }

]

*/
```

7. **sortedIndex(array, value)**: Uses a binary search to determine the lowest index at which value should be inserted into array in order to maintain its sort order.

   Use Case: When you want to find the index at which a value should be inserted into a sorted array.

```
_.sortedIndex([30, 50], 40);

// => 1
```

The `splice()` method adds and/or removes array elements.

The `splice()` method overwrites the original array.

Syntax

```
array.splice(index, howmany, item1, ....., itemX)
```

Parameters

Parameter	Description
index	Required. The position to add/remove items. Negative value defines the position from the end of the array.
howmany	Optional. Number of items to be removed.
item1, ..., *itemX*	Optional. New elements(s) to be added.

```
(Snippet of the splice method that is used in the Codio example)
```

8. **uniq(array)**: Creates a duplicate-free version of an array.

   Use Case: When you want to remove duplicate values from an array.

```
_.uniq([2, 1, 2]);

// => [2, 1]
```

9. **uniqBy(array, [iteratee])**: This method is like _.uniq except that it accepts iteratee which is invoked for each element in an array to generate the criterion by which uniqueness is computed.

   Use Case: When you want to remove duplicate values from an array based on a specific property or computed value.

```
_.uniqBy([{ 'x': 1 }, { 'x': 2 }, { 'x': 1 }], 'x');

// => [{ 'x': 1 }, { 'x': 2 }]
```

Object Methods

1. **pickBy(object, [predicate])**: Creates an object composed of the object properties predicate returns truthy for. The predicate is invoked with two arguments: (value, key).

   The **_.pickBy()** method is used to return a copy of the object that composed of the object properties predicate returns truthy for.

   Syntax:

```
_.pickBy( object, predicate )
```

   Parameters: This method accepts two parameters as mentioned above and described below:

   - **object:** This parameter holds the source object.
   - **predicate:** This parameter holds the function that is invoked for every property. It is an optional value.

Source: https://www.geeksforgeeks.org/lodash-_-pickby-method/#

Use Case: When you want to create a new object that only includes properties that satisfy a certain condition.

```
var object = { 'a': 1, 'b': '2', 'c': 3 };


_.pickBy(object, _.isNumber);

// => { 'a': 1, 'c': 3 }
```

2. **omitBy(object, [predicate])**: The opposite of `_.pickBy`; this method creates an object composed of the object properties predicate returns falsey for.

   Use Case: When you want to create a new object that excludes properties that satisfy a certain condition.

```
var object = { 'a': 1, 'b': '2', 'c': 3 };


_.omitBy(object, _.isNumber);

// => { 'b': '2' }
```

3. **merge(object, [sources])**: This method is used to merge two or more objects. This method is useful for merge objects with the same properties.

   Use Case: When you want to merge properties of several objects into one, with later sources overriding properties of earlier ones.

```
var object = {

  'a': [{ 'b': 2 }, { 'd': 4 }]

};


var other = {

  'a': [{ 'c': 3 }, { 'e': 5 }]

};
```

```
_.merge(object, other);

// => { 'a': [{ 'b': 2, 'c': 3 }, { 'd': 4, 'e': 5 }] }
```

Module 12 - Day 2: Lodash Deep Dive Part 2

Lesson 3: String and Other Useful Lodash Methods

Introduction

In this lesson, we continue to explore more Lodash methods, focusing on string manipulation and other useful functions. We'll look at practical examples to illustrate the usefulness of each method.

String Methods

1. **capitalize(string)**: Converts the first character of `string` to upper case and the remaining to lower case.

   Use Case: When you want to capitalize the first letter of a string, like a name or title.

   ```
   _.capitalize('FRED');

   // => 'Fred'
   ```

2. **truncate(string, [options])**: Truncates `string` if it's longer than the given maximum string length.

   Use Case: When you want to shorten a string, such as a long title or description, to a specific length.

   ```
   _.truncate('hello world', { 'length': 5 });

   // => 'he...'
   ```

3. **camelCase(string)**: Converts `string` to camel case.

   Use Case: When you want to convert a string to camel case, for example, when generating variable or function names from user input.

   ```
   _.camelCase('Foo Bar');

   // => 'fooBar'
   ```

4. **escape(string)**: Converts the characters "&", "<", ">", "", and "" in `string` to their corresponding HTML entities.

   Use Case: When you want to escape HTML characters in a string to prevent XSS attacks.

   ```
   _.escape('fred, barney, & pebbles');

   // => 'fred, barney, &amp; pebbles'
   ```

5. **pad(string, [length=0], [chars=' '])**: Pads `string` on the left and right sides if it's shorter than `length`.

   Use Case: When you want to align text output or generate fixed-width strings.

   ```
   _.pad('abc', 8);

   // => '  abc   '
   ```

6. **words(string, [pattern])**: Splits `string` into an array of its words.

   Use Case: When you want to analyze or manipulate individual words from a text string.

   ```
   _.words('fred, barney, & pebbles');

   // => ['fred', 'barney', 'pebbles']
   ```

7. **template(string, [options])**: Creates a compiled template function that can interpolate data properties in "interpolate" delimiters.

   Use Case: When you need to generate text strings from data, like generating a personalized greeting for an email.

   ```
   var compiled = _.template('hello <%= user %>!');

   compiled({ 'user': 'fred' });
   ```

```
// => 'hello fred!'
```

8. **replace(string, pattern, replacement)**: Replaces matches for `pattern` in `string` with `replacement`.

   Use Case: When you need to find and replace certain patterns in a string.

```
_.replace('Hi Fred', 'Fred', 'Barney');

// => 'Hi Barney'
```

Other Useful Methods

1. **partition(collection, predicate)**: Creates an array of elements split into two groups, the first of which contains elements `predicate` returns truthy for, the second of which contains elements `predicate` returns falsey for.

   Use Case: When you want to divide a collection into two arrays based on a predicate function.

```
var users = [
  { 'user': 'barney', 'age': 36, 'active': false },
  { 'user': 'fred', 'age': 40, 'active': true },
  { 'user': 'pebbles', 'age': 1, 'active': false }
];


_.partition(users, 'active');
/*
=> [
[{ 'user': 'fred', 'age': 40, 'active': true }],
[
  { 'user': 'barney', 'age': 36, 'active': false },
  { 'user': 'pebbles', 'age': 1, 'active': false }
]
```

```
]

*/
```

2. **debounce(func, [wait=0], [options])**: Creates a debounced function that delays invoking `func` until after `wait` milliseconds have elapsed since the last time the debounced function was invoked.

   Use Case: When you want to limit the rate at which a function can fire, such as in event handlers like scrolling, resizing, user input, etc.

```
// Avoid costly calculations while the window size is in flux.

jQuery(window).on('resize', _.debounce(calculateLayout, 150));
```

3. **orderBy(collection, [iteratees=[_.identity]], [orders])**: This method is like `_.sortBy` except that it allows specifying the sort orders of the `iteratees` to sort by.

   Use Case: When you want to sort a collection based on multiple properties with different sort orders.

```
var users = [

  { 'user': 'fred', 'age': 48 },

  { 'user': 'barney', 'age': 34 },

  { 'user': 'fred', 'age': 40 },

  { 'user': 'barney', 'age': 36 }

];



// Sort by `user` in ascending order and by `age` in descending order.

_.orderBy(users, ['user', 'age'], ['asc', 'desc']);

/*

=> [

{ 'user': 'barney', 'age': 36 },

{ 'user': 'barney', 'age': 34 },

{ 'user': 'fred', 'age': 48 },
```

```
{ 'user': 'fred', 'age': 40 }

]

*/
```

4. **cloneDeep(value)**: This method is like `_.clone`, except that it recursively clones `value`.

   Use Case: When you want to create a deep copy of a complex object or array.

```
var objects = [{ 'a': 1 }, { 'b': 2 }];


var deep = _.cloneDeep(objects);

console.log(deep[0] === objects[0]);

// => false
```

5. **isEqual(value, other)**: Performs a deep comparison between two values to determine if they are equivalent.

   Use Case: When you want to check if two complex objects or arrays are equal in value.

```
var object = { 'a': 1 };

var other = { 'a': 1 };


_.isEqual(object, other);

// => true
```

6. **attempt(func, [args])**: Attempts to invoke `func`, returning either the result or the caught error object.

   Use Case: When you want to try to execute a function and handle any errors that may occur.

```
// The following example attempts to invoke a function that might throw an
error.

var result = _.attempt(function(arg) {
```

```
  return arg.split(' ');
}, 'hello world');


if (_.isError(result)) {
  console.log('Caught an error.');
} else {
  console.log(result);
}
// => ['hello', 'world']
```

Resources

1. Lodash Official Documentation: The official Lodash documentation is a great resource to learn about all the available methods in Lodash.
2. Lodash GitHub Repository: The GitHub repo of Lodash is also a good resource to check out. It includes all the source code, and you can also find discussions and issues related to the library.
3. What is Lodash and How it Works (for beginners)
4. Lodash in 2022: Necessary or Obsolete?

Codepen Examples

Day 2, Lesson 2

Array Methods Examples

Object Methods Examples

Day 2, Lesson 3

String Methods Examples

AEL Guides

AEL 2.1 Guide

1. To remove duplicates from an array, you can use the `_.uniq()` function in Lodash. Here's a basic implementation:

```
function removeDuplicates(array) {

    return _.uniq(array);

   }
```

2. To transform the values of a map in Lodash, you can use the `_.map()` function. Here's a basic implementation:

```
function transformMapValues(array) {

    return _.map(array, function(num) {

        // transform the value here

        return num * 2;

    });

   }
```

3. To find unique elements across several sets, you can use the `_.difference()` function multiple times. Here's a basic implementation:

```
function findUniqueElements(sets) {

    return _.difference(_.difference(sets[0], sets[1]), sets[2]);

   }
```

4. To merge two or more objects in Lodash, you can use the `_.merge()` function. Here's a basic implementation:

```
function mergeObjects(objects) {
```

```
 return _.merge({}, ...objects);
}
```

AEL 3.1 Guide

1. To group objects based on a specific property, you can use the `_.groupBy()` function. You would need to transform the 'year' property to decade before grouping. Here's a basic implementation:

```
function groupByDecade(books) {

    return _.groupBy(books, function(book) {

        // return the decade of the book's year

        return Math.floor(book.year / 10) * 10;

    });

   }
```

2. To filter out objects based on a condition, you can use the `_.reject()` function. Here's a basic implementation:

```
function filterBySize(animals, maxSize) {

    return _.reject(animals, function(animal) {

        // reject the animal if its size is larger than maxSize

        return animal.size > maxSize;

    });

   }
```

3. To try invoking a function and catch any errors that occur, you can use the `_.attempt()` function. Here's a basic implementation:

```
function add(a, b) {

    return a + b;

}


function tryAdd(a, b) {
```

```
    var result = _.attempt(add, a, b);


    if (_.isError(result)) {
        // handle the error
        console.error(result);
        return null;
    }


    return result;
}
```

These are just basic implementations. Depending on the exact requirements of the assignment and the specific details of the data you're working with, you may need to adjust these implementations to fit your needs.

Additional Notes

How can you connect this to your learning journey and potential career path?

Here's how learning Lodash can benefit you in the real world:

1. **Improving Code Readability and Maintainability**: Lodash offers many utility functions that can make your code more readable and easier to understand. This is especially important in a team setting where multiple developers are working on the same codebase. Code that is easy to read is also easier to maintain and debug, which can save a lot of time and effort in the long run.

2. **Web Development Careers**: Lodash is a tool that's often used in web development, especially in JavaScript development roles. This includes front-end development, back-end development (Node.js), and full-stack development. By mastering Lodash, you will be better prepared for these types of roles.

3. **Efficiency and Performance**: Some Lodash functions are optimized for performance. When dealing with large datasets, using Lodash functions can sometimes be more efficient than native JavaScript functions. This is particularly relevant in data-intensive roles or when working on performance-critical applications.

4. **Cross-Browser Compatibility**: Lodash functions have consistent behavior across different JavaScript environments, which is not always the case with native JavaScript methods. This is essential when creating web applications that need to work across a variety of browsers.

5. **Real-World Use Cases**: Here are a few examples of how Lodash might be used in a real-world setting:

   - A front-end developer might use Lodash's array and object manipulation methods when working with complex state objects in a React or Vue.js application.

   - A back-end developer might use Lodash's utility functions when processing data from a database or an API.

- A full-stack developer might use Lodash on both the front-end and the back-end, providing consistency across the entire application.

In terms of career paths, mastering Lodash and JavaScript in general can open up opportunities in a variety of areas, including:

- Web Development

- Full-Stack Development

- Back-End Development (Node.js)

- Front-End Development

- Software Engineering

- Data Analysis (JavaScript is increasingly being used in this field)

- Technical Writing (for those who enjoy explaining technical concepts)

While Lodash is a powerful tool, it's just one of many libraries and frameworks in the JavaScript ecosystem. Being adaptable and continually learning new technologies is key to a successful career in web development and programming.