UNDERSCORE.JS

Karthik Raman

What is UnderscoreJS

- Javascript Utility Library
- Underscore.js is a open source component from DocumentCloud
- https://underscorejs.org/

- Complimentary to Native Javascript
  - Provides Utility methods for Arrays, Collections, Objects and Functions
  - Plays well with other libraries

- Unobtrusive
  - Delegates to built-in Native functions if present

- Flexible
  - Functional or Object Oriented Type
  - Chainable

- Templating Engine

UNDERSCORE.JS

jQuery
write less, do more.

Underscore.js to manipulate Data along with Templates

jQuery for manipulating DOM

Powerful Combination

Native Javascript Array Methods

Native Javascript Array Methods

Method
concat()
every()
fill()
filter()
find()
findIndex()
forEach()
join()
map()
pop()
push()
reduce()
reduceRight()
reverse()
shift()
slice()
some()
sort()
splice()
unshift()

Native Javascript Array Methods

Clayfin

Javascrpt_Arrray_functions.html

Method	Description
concat()	Merge two or more arrays, and returns a new array.
every()	Checks if every element in an array pass a test in a testing function.
fill()	Fill the elements in an array with a static value.
filter()	Creates a new array with all elements that pass the test in a testing function.
find()	Returns the value of the first element in an array that pass the test in a testing function.
findIndex()	Returns the index of the first element in an array that pass the test in a testing function.
forEach()	Calls a function once for each array element.
join()	Joins all elements of an array into a string.
map()	Creates a new array with the results of calling a function for each array element.
pop()	Removes the last element from an array, and returns that element.
push()	Adds one or more elements to the end of an array, and returns the array's new length.
reduce()	Reduce the values of an array to a single value (from left-to-right).
reduceRight()	Reduce the values of an array to a single value (from right-to-left).
reverse()	Reverses the order of the elements in an array.
shift()	Removes the first element from an array, and returns that element.
slice()	Selects a part of an array, and returns the new array.
some()	Checks if any of the elements in an array passes the test in a testing function.
sort()	Sorts the elements of an array.
splice()	Adds/Removes elements from an array.
unshift()	Adds new elements to the beginning of an array, and returns the array's new length.

Underscore Helper Methods

Arrays

Objects

Collections

Functions

Utility

Object-Oriented style:

```
_(arr).pluck('name')
```

Functional style:

```
_.pluck(arr, 'name')
```

Underscore
Array Helper Methods

uniq This Method removes Duplicate values from array

```
_.uniq(["Karthik","Raman","Kannan","Rajesh","Karthik",
        "Rajesh"]);
```

```
["Karthik", "Raman", "Kannan", "Rajesh"]
```

range

_.range(*Starting Number, Ending Number, Increment*)

create array with values between specified range

_.range(0, 100, 11);

[0, 11, 22, 33, 44, 55, 66, 77, 88, 99]

union union of the passed-in arrays

```
_.union([1, 2, 3], [101, 2, 1, 10], [2, 1]);
```

[1, 2, 3, 101, 10]

intersection intersection of the passed-in arrays

```
_.intersection([1, 2, 3], [101, 2, 1, 10], [2, 1]);
```

[1, 2]

difference returns the values from array that are not present in the other arrays

```
_.difference([1, 2, 3, 4, 5], [5, 2, 10]);
```

[1, 3, 4, 10]

first **_.first(array, [n])**

Returns the first element of an array. Passing n will return the first n elements of the array

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

[5]

---

initial **_.initial(array, [n])**

Returns everything but the last entry of the array. Pass n to exclude the last n elements from the result

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

[5, 4, 3, 2]

last _.last(array, [n])

Returns the last element of an array. Passing n will return the last n elements of the array

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

[1]

rest _.rest(array, [index])

Returns the rest of the elements in an array. Pass an index to return the values of the array from that index onward.

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

[4, 3, 2, 1]

compact _.compact(list)

Returns a copy of the list with all falsy values removed. In JavaScript, false, null, 0, "", undefined and NaN are all falsy.

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

[1, 2, 3]

flatten _.flatten(array, [shallow])

Flattens a nested array (the nesting can be to any depth). If you pass shallow, the array will only be flattened a single level

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

[1, 2, 3, 4]

```
_.flatten([1, [2], [3, [[4]]]], true);
```

[1, 2, 3, [[4]]]

zip _.zip(*arrays)

Merges together the values of each of the arrays with the values at the corresponding position

```
_.zip(['moe', 'larry', 'curly'], [30, 40, 50], [true, false, false]);
[["moe", 30, true], ["larry", 40, false], ["curly", 50, false]]
```

---

unzip _.unzip(*arrays)

Given an array of arrays, returns a series of new arrays, the first of which contains all of the first elements in the input arrays, the second of which contains all of the second elements, …

```
_.unzip([["moe", 30, true], ["larry", 40, false], ["curly", 50, false]]);
[['moe', 'larry', 'curly'], [30, 40, 50], [true, false, false]]
```

Underscore Collection Helper Methods

each _.each(list, iteratee, [context])

Iterates over a list of elements, yielding each in turn to an iteratee function. The iteratee is bound to the context object, if one is passed. Each invocation of iteratee is called with three arguments: (element, index, list). If list is a JavaScript object, iteratee's arguments will be (value, key, list). Returns the list for chaining

_.each() with Arrays

```
var cars = [Maruti', 'Nissan', ‘BMW'];

_.each(cars, function (element, index, list) {
    var output = 'Element: ' + element + ', ' + 'Index: ' + index + ',
' + 'List Length: ' + list.length;
    console.log(output);
});
```

```
Element: Maruti, Index: 0, List Length: 3
Element: Nissan, Index: 1, List Length: 3
Element: BMW, Index: 2, List Length: 3
```

_.each() with Objects

```
var sampleobject = {FirstKey: 'One', SecondKey: 'Two'};

_.each(sampleobject, function (value, key) {
    var output = 'The value is '
            + value + ' and the key is ' + key;
    console.log(output);
});
```

The value is One and the key is FirstKey
The value is Two and the key is SecondKey

_.each() and Arrays With Context context binds “this” object

```
var TechCompanies = {
    names: ['Google', 'Amazon', 'Apple'],
    doStuff: function (company) {
        return company + ' creates great products.';
    }
};
_.each(TechCompanies.names, function (element, index, list) {
    console.log(this.doStuff(element));
}, TechCompanies);
```

Google creates great products.
Amazon creates great products.
Apple creates great products.

_.each() and Objects With Context context binds “this” object

```
var features = {
    one: 'Auto Pilot',  two: 'Summons Feature',  three: 'Ludicrous Mode'
};
 var salesperson = {
  sellCar: function (msg) {
   return 'This Tesla has '  + msg;
    }
};
```

```
_.each(features, function (value, key)
{
    console.log(this.sellCar(value));
}, salesperson);
```

This Tesla has Auto Pilot
This Tesla has Summons Feature
This Tesla has Ludicrous Mode

map _.map(list, iteratee, [context])

Produces a new array of values by mapping each value in list through a transformation function (iteratee). The iteratee is passed three arguments: the value, then the index (or key) of the iteration, and finally a reference to the entire list.

```
_.map([1, 2, 3], function(num){ return num * 3; });
[3, 6, 9]
_.map({one: 1, two: 2, three: 3}, function(num, key){ return num * 3; });
[3, 6, 9]
_.map([[1, 2], [3, 4]], _.first);
[1, 3]
```

reduce **_.reduce(list, iterator, memo, [context])**

Also known as **inject**, **reduce** boils down a list of values into a single value. Memo is the initial state of the reduction, and each successive step of it should be returned by iterator

```
var values = [1, 2, 3, 4, 5];
 var sum = _.reduce(values, function (memo, number) {
    console.log('Calculating: ' + memo + ' + ' + number);
    return memo + number;
});
console.log('Total :' + sum);
```

```
Calculating: 1 + 2
Calculating: 3 + 3
Calculating: 6 + 4
Calculating: 10 + 5
Total : 15
```

```
var data = {
    items: [
        {item: 'Soup', price: 49},
        {item: 'Roti', price: 35},
        {item: 'Gravy', price: 55},
        {item: 'Ice Cream', price: 20}
    ]
};
var total = _.reduce(data.items,
        function (memo, value) {
            return memo + value.price;
        }, 0);

log('Total Price : Rs. ' + total);
```

Initial Value set as 0

Total Price : Rs. 159

reduceRight **_.reduceRight(list, iterator, memo, [context])**

The right-associative version of **reduce**

```
var values = [1, 2, 3, 4, 5];
 var sum = _.reduceRight(values, function (memo, number) {
    console.log('Calculating: ' + memo + ' + ' + number);
    return memo + number;
});
console.log('Total :' + sum);
```

```
Calculating: 5 + 4
Calculating: 9 + 3
Calculating: 12 + 2
Calculating: 14 + 1
Total : 15
```

every _.every(list, [predicate], [context])

Returns *true* if all of the values in the **list** pass the **predicate** test.

```
_.every([2, 4, 5], function(num) { return num % 2 == 0; });
```

false

```
var people = [
        {name: 'Karthik', car: true},
        {name: 'Raman', car: true},
        {name: 'Uma', car: false}
    ],
    hasCar = function (value) {
        return (value.car === true);
    };
    console.log(_.every(people, hasCar));
```

false

some _.some(list, [predicate], [context])

Returns *true* if any of the values in the **list** pass the **predicate** test

```
_.some([2, 4, 5], function(num) { return num % 2 == 0; });
```

true

```
var people = [
        {name: 'Karthik', car: true},
        {name: 'Raman', car: true},
        {name: 'Uma', car: false}
    ],
    hasCar = function (value) {
        return (value.car === true);
    };
    console.log(_.some(people, hasCar));
```

true

contains _.contains(list, value, [fromIndex])

Returns true if the value is present in the list

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

   true
```

where _.where(list, properties)

Looks through each value in the **list**, returning an array of all the values that matches the key-value pairs listed in prorpeties

```
var people = [
        {name: 'Karthik', car: true},
        {name: 'Raman', car: true},
        {name: 'Uma', car: false}
    ],
    hasCar = function (value) {
        return (value.car === true);
    };
    console.log(_.where(people, hasCar));
```

```
[{name: 'Karthik', car: true},
{name: 'Raman', car: true}]
```

findWhere _.findWhere(list, properties)

Looks through each value in the **list**, returning the first value that matches the key-value pairs listed in prorpeties

```
var people = [
        {name: 'Karthik', car: true},
        {name: 'Raman', car: true},
        {name: 'Uma', car: false}
    ],
    hasCar = function (value) {
        return (value.car === true);
    };
    console.log(_.findWhere(people, hasCar));
```

[{name: 'Karthik', car: true}]

sortBy **_.sortBy(list, iteratee, [context])**

Returns a sorted copy of list, ranked in ascending order by the results of running each value through iteratee

```
var array = _.sortBy([2, 3, 1], function(num) {
        return num;
        });
```

[1, 2, 3]

```
var values = ['Underscore', 'jQuery', 'Angular', 'React', 'Backbone'];
 var lengthSorter = function (value) {
    return value.length;
};
var sortedValues = _.sortBy(values, lengthSorter);
```

[React, jQuery, Angular, Backbone, Underscore]

```
var arr = [
        {name: 'Karthik', id: 2},
        {name: 'Raman', id: 1},
        {name: 'Kannan', id: 3}
    ];
    console.log(_.sortBy(arr, 'id'));
```

```
[{name: 'Raman', id: 1},
 {name: 'Karthik', id: 2},
 {name: 'Kannan', id: 3} ]
```

```
var arr = [
        {name: 'Karthik', id: 2},
        {name: 'Raman', id: 1},
        {name: 'Kannan', id: 3}
    ];
    console.log(_.sortBy(arr, 'id').reverse());
```

```
[{name: 'Kannan', id: 3} ,
 {name: 'Karthik', id: 2},
 {name: 'Raman', id: 1} ]
```

find **_.find(list, predicate, [context])**

Looks through each value in the list, returning the first one that passes a predicate test

```
var even = _.find([1, 2, 3, 4, 5, 6], function(num){ return num % 2 == 0;
});
```

2

```
var values = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
var greaterThan7 = function (value) {
    return value > 7;
};
_.find(values, greaterThan7)
```

8

```
var websites = [{
    name: 'Google',
    type: 'Search Engine'
}, {
    name: 'Bing',
    type: 'Search Engine'
}, {
    name: 'Facebook',
    type: 'Social Network'
}, {
    name: 'Twitter',
    type: 'Social Network'
}, {
    name: 'Amazon',
    type: 'Shopping Site'
}];
```

```
var getWebsitesCountPerType = function () {
  var websitesCountPerType = [];
  websites.forEach(function (website) {
    var currentTypeCount = _.find(websitesCountPerType,
        function (typeCount) {
          return typeCount.type === website.type;
        });
    if (currentTypeCount) {
      currentTypeCount.count += 1;
    }
    else {
      websitesCountPerType.push({
        type: website.type,
        count: 1
      });
    }
  });
  return websitesCountPerType;
};
```

websitesCountPerType

type	count
Search Engine	2
Social Network	2
Shopping Site	1

filter _.filter(list, predicate, [context])

Looks through each value in the list, returning an array of all the values that pass a predicate test

```
var evens = _.filter([1, 2, 3, 4, 5, 6], function(num){ return num % 2 == 0;
});
```

[2, 4, 6]

```
var values = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
var greaterThan7 = function (value) {
    return value > 7;
};
_.filter(values, greaterThan7)
```

[8, 9, 10]

```
var values = [
            {phone: 'iPhone 11 Pro', price: 100000},
            {phone: 'Samsung Galaxy 10', price: 60000},
            {phone: 'One Plus 7',price: 49000},
            {pnone: 'Oppo F11 Pro', price: 18000}
        ];
 greaterThan50000 = function (value) {
            return value.price > 50000;
        },

_.each(_.filter(values, greaterThan50000),
        function (value) {
            console.log(value.phone');
        });
```

iPhone 11 Pro
Samsung Galaxy 10

reject _.reject(list, predicate, [context])

Returns the values in list without the elements that passes the predicate test (**opposite of filter**)

```
var odds = _.reject([1, 2, 3, 4, 5, 6], function(num){ return num % 2 == 0; });
```

[1, 3, 5]

```
var values = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
var greaterThan7 = function (value) {
    return value > 7;
};
_.reject(values, greaterThan7)
```

[1, 2, 3, 4, 5, 6, 7]

```
var values = [
            {phone: 'iPhone 11 Pro', price: 100000},
            {phone: 'Samsung Galaxy 10', price: 60000},
            {phone: 'One Plus 7',price: 49000},
            {pnone: 'Oppo F11 Pro', price: 18000}
        ];
 greaterThan50000 = function (value) {
            return value.price > 50000;
        },

_.each(_.reject(values, greaterThan50000),
        function (value) {
            console.log(value.phone');
        });
```

One Plus 7
Oppo F11 Pro

max _.max(list, [iteratee], [context])

Returns the maximum value in **list**

```
var odds = _.max([1, 2, 3, 4, 5, 6]);
```

[6]

min _.min(list, [iteratee], [context])

Returns the minmum value in **list**

```
var odds = _.min([1, 2, 3, 4, 5, 6]);
```

[1]

```
var values = [
            {phone: 'iPhone 11 Pro', price: 100000},
            {phone: 'Samsung Galaxy 10', price: 60000},
            {phone: ‘One Plus 7',price: 49000},
            {pnone: ‘Oppo F11 Pro', price: 18000}
        ];

 _.max(values, function (value) {
            return value.price;
        });
```

{phone: 'iPhone 11 Pro', price: 100000}

countBy **_.countBy(list, iteratee, [context])**

Sorts a list into groups and returns a count for the number of objects in each group

```
_.countBy([1, 2, 3, 4, 5], function(num) {
  return num % 2 == 0 ? 'even': 'odd';
});
```

{odd: 3, even: 2}

groupBy _.groupBy(list, iteratee, [context])
Splits a collection into sets, grouped by the result of running each value through iteratee

```
_.groupBy(['one', 'two', 'three'], 'length');
```

```
{3: ["one", "two"], 5: ["three"]}
```

```
_.groupBy([1.3, 2.1, 2.4], function(num){ return Math.floor(num); });
```

```
{1: [1.3], 2: [2.1, 2.4]}
```

```
var websites = [{
    name: 'Google',
    type: 'Search Engine'
}, {
    name: 'Bing',
    type: 'Search Engine'
}, {
    name: 'Facebook',
    type: 'Social Network'
}, {
    name: 'Twitter',
    type: 'Social Network'
}, {
    name: 'Amazon',
    type: 'Shopping Site'
}];
```

```
_.groupBy(websites , ‘type')
```

```
{
Search Engine  :  [ { name: 'Google', type: 'Search Engine'},
                    { name: 'Bing', type: 'Search Engine'} ],

Shopping Site   :  [{ name: 'Amazon', type: 'Shopping Site‘}],

Social Network :  [{ name: 'Facebook ', type: 'Social Network‘},
                    { name: 'Twitter ', type: ' 'Social Network‘}]
}
```

partition _.partition(list, predicate)

Split list into two arrays: one whose elements all satisfy predicate and one whose elements all do not satisfy predicate

```
_. partition([1, 2, 3, 4, 5], function(num) {
                return num % 2 == 0 ;
});
```

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

pluck _.pluck(list, propertyName)

extracting a list of property values

```
var websites = [{
    name: 'Google',
    type: 'Search Engine'
}, {
    name: 'Bing',
    type: 'Search Engine'
}, {
    name: 'Facebook',
    type: 'Social Network'
}, {
    name: 'Twitter',
    type: 'Social Network'
];
```

```
_.pluck(websites , ‘name')
```

['Google ', 'Bing ', 'Facebook ‘, 'Twitter’]

invoke _.invoke(list, methodName, *arguments)

Calls the method named by methodName on each value in the list. Any extra arguments passed to invoke will be forwarded on to the method invocation

```
_.invoke([[5, 1, 7], [3, 2, 1]], 'sort');
```

[[1, 5, 7], [1, 2, 3]]

```
_.invoke([[5, 1, 7], [3, 2, 1]], ‘join’, ‘ # ’);
```

[[5 # 1 # 7], [3 # 2 # 1]]

Underscore Functions Helper Methods

bind _.bind(function, object, *arguments)

Bind a function to an object, meaning that whenever the function is called, the value of this will be the object

```
function Developer(skill) {
  this.skill = skill;
  this.greet = function(){
    console.log(this.skill + ' rocks!');
  }
}
var dev = new Developer('Javascript');
var func = _.bind(dev.greet, dev);
func();
```

Javacript rocks!

bindAll _.bindAll(object, *methodNames)

Binds a number of methods on the object, specified by methodNames, to be run in the context of that object whenever they are invoked

```
var buttonView = {
  label  : 'underscore',
  onClick: function(){ console.log('clicked: ' + this.label); },
  onHover: function(){ console.log('hovering: ' + this.label); }
};
_.bindAll(buttonView, 'onClick', 'onHover');
// When the button is clicked, this.label will have the correct value.
jQuery('#underscore_button').on('click', buttonView.onClick);
```

partial _.partial(function, *arguments)

Partially apply a function by filling in any number of its arguments, without changing its dynamic this value

```
var subtract = function(a, b) { return b - a; };
sub5 = _.partial(subtract, 5);
sub5(20);
```

15

```
// using PlaceHoders
var cube = _.partial(Math.pow, _, 3);
var x = cube(4);     // Evaluates Math.pow(4, 3)
```

64

```
var mult = function (a,b,c,d,e) { return a*b*c*d*e;}
var m = _.partial(mult , 1 , _ , _ , _ , 5 );
m ( 2 , 3 , 4 )
```

120

wrap _.wrap(function, wrapper)

Wraps the first function inside of the wrapper function, passing it as the first argument. This allows the wrapper to execute code before and after the function runs

```
var hello = function(name) { return "hello: " + name; };
hello = _.wrap(hello, function(func) {
  return “<before> " + func(“Karthik") + “ <after>";
});
hello();
```

‘<before> hello: Karthik <after>'

compose _.compose(*functions)

Returns the composition of a list of functions, where each function consumes the return value of the function that follows. In math terms, composing the functions f(), g(), and h() produces f(g(h()))

```
var greet   = function(name){ return "hello: " + name; };
var exclaim  = function(statement){ return statement.toUpperCase() + "!"; };
var welcome = _.compose(greet, exclaim);
welcome('karthik');
```

'hello: KARTHIK !'

once _.once(function)

Creates a version of the function that can only be called one time. Repeated calls to the modified function will have no effect

```
var initialize = _.once(createApplication);
initialize();
initialize();
```

after **_.after(count, function)**

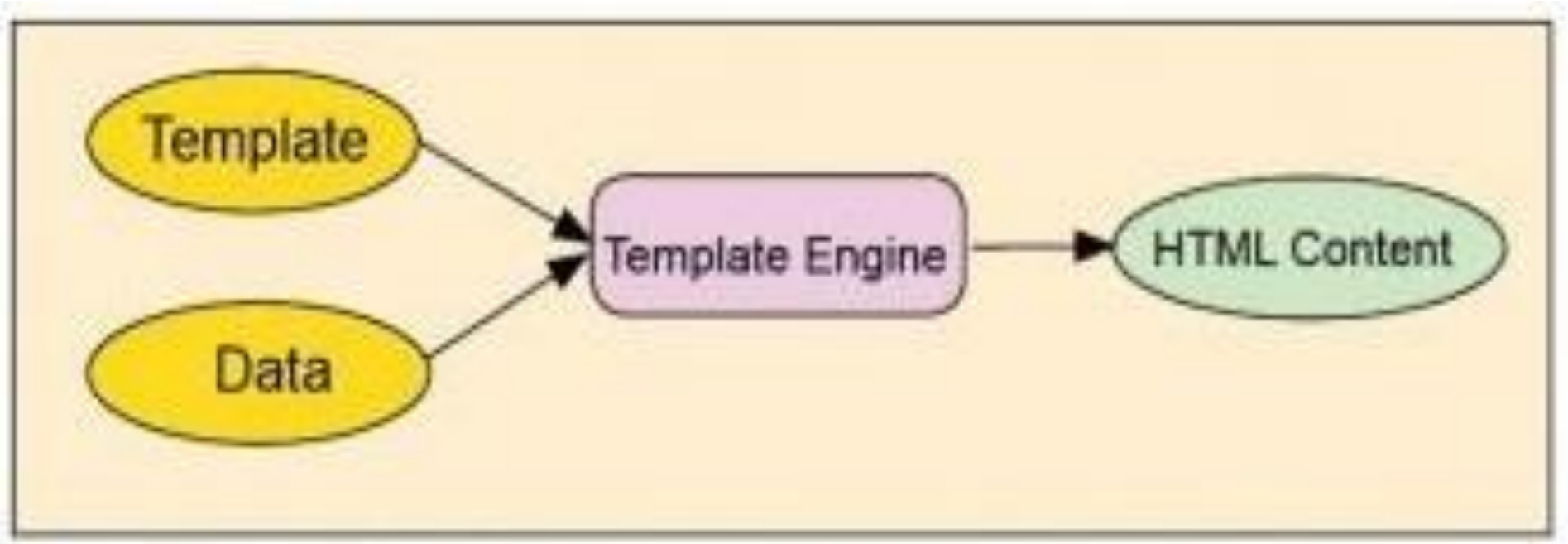
Creates a version of the function that will only be run after being called **count** times.

Useful for grouping asynchronous responses, where you want to be sure that all the async calls have finished, before proceeding

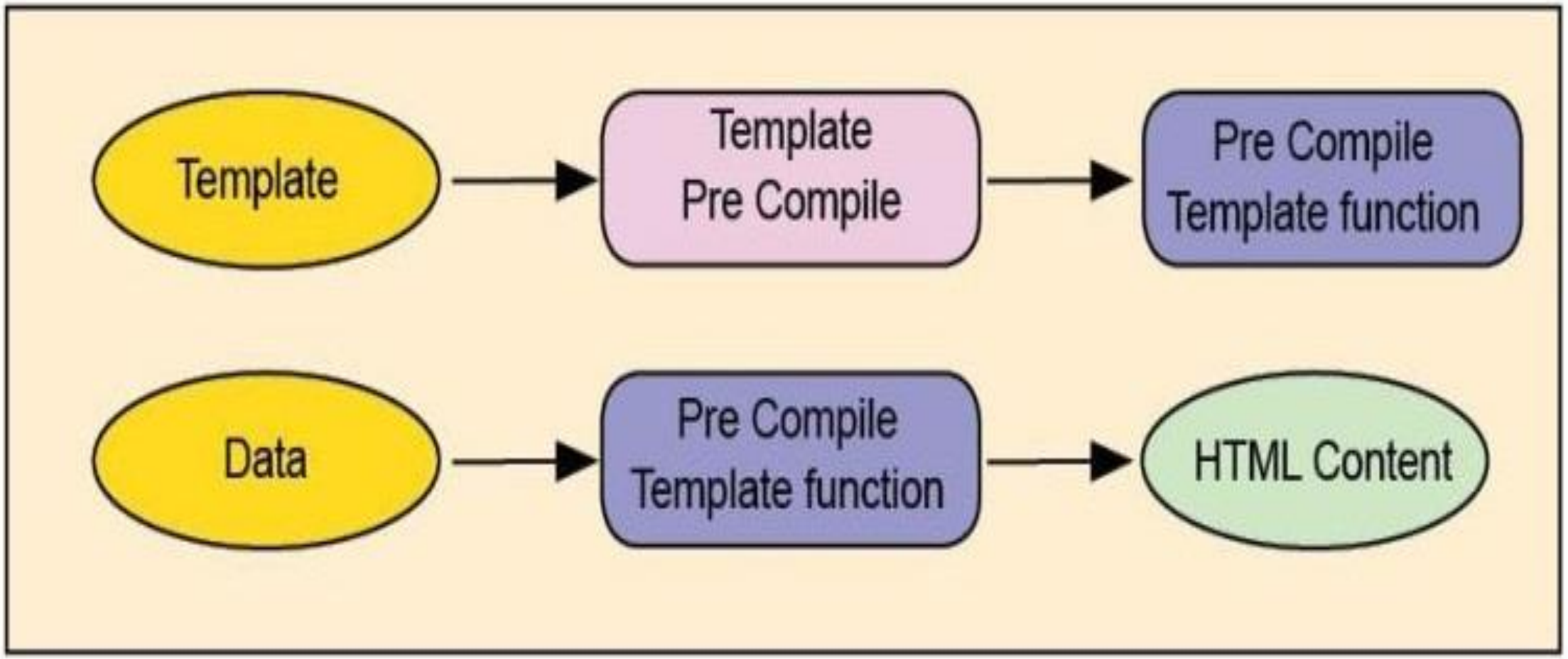
```
var renderNotes = _.after(notes.length, render);
_.each(notes, function(note) {
  note.asyncSave({success: renderNotes});
});
// renderNotes is run once, after all notes have saved.
```

Underscore Templating Engine

Decouple Javascript and HTML
Code Maintainability



Template Precompilation



template function has the following signature:

```
_.template ( templateString , data? , settings? )
```

templateString - holds the template

data - optional parameter which will be inserted into the template

If data is omitted, the template will be compiled as a function to which data can be applied later

settings - to override the global settings

```
_.template("Hello <%=user%> !!!", { user: "< Karthik >" })
```

Hello < Karthik > !!!

```
var templ =  _.template("Hello <%=user%> !!!")
```

Compile into a function

```
templ({ user: "< Karthik >" })
```

Insert data into template

Hello < Karthik > !!!

interpolate **<%= %>**

```
_.template("Hello <%=user%> !!!", { user: "< Karthik >" })
```

Hello < Karthik > !!!

Insert the result of an expression

data objects are all available as variables

No escaping happens, values are inserted verbatim

escape <%- %>

```
_.template("Hello <%-user%> !!!", { user: "< Karthik >" })
```

```
Hello &lt; Karthik &gt; !!!
```

Insert the result of an expression, but escape the following characters & < > " ' /

& - &	< - <	> - >
" - "	' - '	/ - /

evaluate <% %> Loops and conditions

```
var templ = _.template(
      "Users: <%_.forEach(users, function (u,index) {%>"
      + "<%if (index>0) {%>, <%}%>"
      + "<%=u%>"
      + "<%})%>"
   );
```

```
templ({ users: [ "Karthik", "Kannan", "Rajesh" ]})
```

Users: Karthik, Kannan, Rajesh

refer to the properties of the data via object, instead of accessing them as variables. variable holding the data object is obj by default

```
_.template("Hello <%=user%> !!!", { user: "< Karthik >" })

_.template("Hello <%=obj.user%> !!!", { user: "< Karthik >"})

_.template(" <%if (obj.title) {%>Title: <%=title%><%}%>",
                                        { title: “underscore"})

_.template(" <%=data.title%>", { title: “underscore"}, {
variable: "data" })                    // obj changed as data
```

Global Settings for Templates

```
_.templateSettings = {
    interpolate : /\{\{(.+?)\}\}/g
  };
```

Mustache-style curly braces instead of angle brackets

```
_.template("Hello {{user}} !!!", { user: "< Karthik >" })
```

Hello < Karthik > !!!

Keep the template content as a string and assign it to a variable

Keep the template contents in a text file (tpl)
include it by using the <script type="text/template" > tag

Use a modular script loader like RequireJS

Template-1.html

Template-2.html

Clayfin

Template - Logic

```
<table>
<% _.each(items, function(item, key, list) { %>
<tr>
<td> <%= item.id %></td>
<td><%= item.name %></td >
</tr>
 <% });%>
</table >
```

Underscore.js

Template - Logicless

```
<table>
{{#each items}}
<tr>
<td>{{item.id }}</td>
<td>{{item.name }}</td >
</tr>
{{/each}}
</table >
```

Handlebars.js
Mustache.js

Underscore

Mustache

HandleBars

EJS

Jade

Pug

Squirelly

dust

jsRender

Marko

Good Libraries **reduces the burden**

UNDERSCORE.JS

Thank You

www.clayfin.com