Table of Contents

Livongo Commits

Angular

- DEV-2277
  - hypen in […] of **regular expression** has a different behavior. It has to be either beginning or ending. Ex: [.-_] is treated as all the ASCII characters from . through _ (hypen may be not included in that range), where as [._-] is treated as accept . and _ and -.
- DEV-2150
  - **JS:** unshift() prepends to the array whereas push() appends one or more elements to the array.

    ```
    var fruits = ["Banana", "Orange", "Apple", "Mango"];
    fruits.push("Kiwi");
    //Banana,Orange,Apple,Mango,Kiwi
    ```
  - Kendo dropdown UI.
- DEV-2035
  - Angular - $rootScope, $scope, $location.path()
- DEV-2059
- DEV-2347
  - **Angular - Form Validation**. To make a field invalid in Angular, you can use $setValidity().
  - C1 - setValidity usage
  - $scope.formName.fieldName.$setValidity("patternmatch",true); -- it adds a class called ng-valid-patternmatch, and ng-valid to the field
  - Other useful form validation methods -
    - $scope.formName.$setPristine(); - sets the form and all its fields to pristine state
    - $scope.formName.$setUntouched();
    -

-
- DEV-2519
  - Doing it by creating global module (Link) vs. hard-coding it (Link)
  - The tabindex attribute explicitly defines the navigation order for focusable elements (typically links and form controls) within a page (1 is first).
  - tabindex="0" - you can make elements that are not normally tab-able can be made tab-able by adding tabindex="0"
  - Elements that have identical tabindex values should be navigated in the order in which they appear in the document's (x)html mark-up
  - The keyup event occurs when a keyboard key is released. The keyup() method triggers the keyup event, or attaches a function to run when a keyup event occurs.

```
$("#careteam-span-add-member").keyup(function(event) {
        if (event.keyCode == 13) {  //enter key is pressed and released
                $("#careteam-span-add-member").click();
                 }
 });
```

- MP-1853
  - z-index: The z-index property specifies the stack order of an element. An element with greater stack order is always in front of an element with a lower stack order.
  - Note: z-index only works on positioned elements (position:absolute, position:relative, or position:fixed).
- DEV-2683, DEV-2941
  - Angular Schema Form, translate
- DEV-2397
  - If you change any model outside of the Angular context and update the binded values, you need to inform Angular of the changes by calling $apply() manually. It's like telling Angular that you are changing some models and it should fire the watchers so that your changes propagate properly.
  - using $timeout instead of $scope.apply() (Link)
    - As $scope.$apply() triggers a $digest and in angular at any point in time there can be only one $digest or $apply operation in progress. You should use $timeout
  - Lodash: Deep clone (_.clone(obj,true)) clones nested objects as well.
- DEV-1820 (C1 and C2)
  - jQuery - .hasClass(), .addClass(), .removeClass()

```
if($("#confirmPassword").hasClass("lv-input-filled") &&
$("#newPassword").hasClass("lv-input-empty"))
{
        $("#confirmPassword").addClass("ng-valid-pwd-validation");
}
```

  - **CSS:**
    - `.class1 .class2`
      class2 is a child of class1, ascendant -> descendant

```
<span class="class1"><p class="class2">Something</p></span>
```

    - `.class1.class2 (no space)`
      element must have both classes for the rule to apply.

```
<p class="class1 class2">Something</p>
```

- DEV-2450
  - CSS
    - **overflow**: The overflow property specifies what happens if content overflows an element's box.
    - overflow: scroll - The overflow is clipped, but a scroll-bar is added to see the rest of the content
    - overflow: hidden - The overflow is clipped, and the rest of the content will be invisible
    - overflow: visible - The overflow is not clipped. It renders outside the element's box. This is default
- DEV-2766
  - `string.substring(start,end)`
    - This method extracts the characters in a string between "start" and "end", not including "end" itself.
    - end - optional, if omitted, it extracts the rest of the string

```
var str = "Hello world!";
console.log(str.substring(1, 4)); //ell
console.log(str.substring(1));  //ello world!
```

- DEV-2801
  - The $.trim() function removes all newlines, spaces (including non-breaking spaces), and tabs from the beginning and end of the supplied string.

```
$.trim("    hello, how are you?    "); //"hello, how are you?"
```

  - https://learn.jquery.com/using-jquery-core/jquery-object/
  - **CSS**
    - CSS [attribute] Selector: The following example selects all <a> elements with a target attribute:

```
a[target] {
        background-color: yellow;
}
```

    - CSS [attribute=value] Selector: The following example selects all <a> elements with a target="_blank" attribute:

```
a[target=_blank] {
        background-color: yellow;
}
```

    - The CSS3 :Not() Selector: The :not(selector) selector matches every element that is NOT the specified element/selector.

```
        div *:not(p) em {...}
```

      This selects all em elements that are nested in an element other than p element, in a div. So,

      Match

```
<div>
  <strong>
    <em>...</em>
  </strong>
</div>
-------------
```

      Not a match

```
<div>
  <p>
```

```
    <em>...</em>
  </p>
</div>
-------------
```

`input:not([type=file]) {...}` - This selects all input elements not having the attribute type="file"

- Many developers prefix a $ to the name of variables that contain jQuery objects in order to help differentiate.

```
var $html = $('html');
```

- DEV-2761 (to fix misplaced menu bar in alldata.html)
- DEV-2449 (logbook hide initial message)
- DEV-3565 (Help Center - Enhanced Reg) (C1 C2, C3-lvtranslate)
- DEV-3562 (Remember me feature -> Log in) (C1)
- Positioning (C1)
  - **CSS**
    - http://learnlayout.com/margin-auto.html
    - **margin: 0 auto;** (to center an element)
      You can set the left and right margins to auto to horizontally center that element within its container. Here top and bottom are set to 0px. Note that it works only when you specify **width** for that **block** element. (button by default, is an inline-block element, so you can set width. But for margin hack to work, you've to set as dispaly:block for the button.)
- DEV-4346 (C1) (Example of using lodash library)
  - Javascript and Java - pass by value for primitive data types (byte, short, int, long - Java; in JS we've var x=10, so it depends on type of data), pass by reference for objects (arrays etc).
  - So you deep clone (_.clone(insurers, true)) below so that the original *insurers* object passed in the function parameter is intact.

```
-----------------------------------------
function normalizeInsurers(insurers) {
  var normalizedInsurers = _.clone(insurers, true);
  _.each(normalizedInsurers, function(insurer) {
     if(insurer.hasLogo) {
         insurer.logo ='<img  ng-attr-display-retina-image
ng-attr-src-normal=\'"/img/' + insurer.name + '.png"\'
ng-attr-src-retina=\'"/img/retina/' + insurer.name + '.png"\' />';
     }
     else {
         insurer.logo = insurer.name;
     }
  });
  return normalizedInsurers;
}
```

- DEV-4299 (C1, C2, C3)
  - rootScope usage - Every application has a single root scope. A property assigned with $scope cannot be used outside the controller in which it is defined whereas the a property assigned with $rootScope can be used anywhere.

    - using $rootScope vs. factory. (C2)
        - with $rootScope, say, you save $rootScope.firstName = “xyz”; Later in a different page’s view, you have {{firstName}}, it may create confusion if you have a property with the same name firstName in its controller.
        - C1 (DEV-4481 - ineligibleuser - hide continue to MP in confirmation page.)
- DEV-4297 (C1)
    - Custom directive Angular for format zip code
- DEV-4439 (C1) - jQuery to detect keyUp enter event
- DEV-4372 (Laura commit - C1) - To learn
- DEV-4599 (C1, C2) - Terms and Conditions
    - input id should match label id, for to click anywhere on label resulting in checking.
- DEV-4561 (C1) - Keeping “Not Listed” as the last option
    - Lodash (underscore similar) JS library

      Ex: _.isUndefined(), _.isEmpty(), _.filter(), _.each()

```
var obj;
_.isUndefined(obj) - true //since obj is just declared, not defined
_.isEmpty(obj) - true  // mind you it is true here even if it's not defined
------------------------------
var obj = {}; //now it's defined
_.isUndefined(obj) - false
_.isEmpty(obj) - true
obj = {name: "keera"};
_.isUndefined(obj) - false
_.isEmpty(obj) - false
------------------------------
```

    - JS has function level scope (unlike Java, .net having block level scope)
    - JS - pass by value vs. pass by reference (http://stackoverflow.com/questions/518000/is-javascript-a-pass-by-reference-or-pass-by-value-language)

```
var fruits = ['apple','grapes'];
var c = 10;
function modify(myFruits, myC) {
   myFruits.push('banana');
   myC = 11;
}
modify(fruits,c);
console.log(fruits); // ["apple", "grapes", "banana"]
console.log(c);  //10
```

```
function normalizeInsurers(insurers) {
     var normalizedInsurers = _.clone(insurers, true); //*** deep cloning as you don't want to change
    the original 'insurers' object. Refer above example.

     var notListedInsurer;
    //_.isUndefined(notListedInsurer) true
    //_.isEmpty(notListedInsurer) true
     _.each(normalizedInsurers, function(insurer) {
         insurer.logo = insurer.name;
         if(insurer.name === "Not Listed") {
```

```
            notListedInsurer = insurer;
            //_.isUndefined(notListedInsurer) false
          //_.isEmpty(notListedInsurer) false
        }
    });

    //Keep "Not Listed" as the bottom most option in the insurers list
    if(!_.isUndefined(notListedInsurer) &&  !_.isEmpty(notListedInsurer)) //second check is optional
here
    {
        normalizedInsurers = _.filter(normalizedInsurers,  function(obj) {
          return (obj.name !== "Not Listed");
        });
        normalizedInsurers.push(notListedInsurer);
    }
    return normalizedInsurers;
}
```

- DEV-4567 (C1, C2)
- DEV-4763 (C1)
    - display: block
- DEV-4786
-

CSS:

- element1 + element2 selector (**adjacent sibling selector**) - selects element2 that is immediately after element1 (link)

```
.health-profile.step input#diabetesManagement+span {
    display: block;
    margin-top: -22px;
    margin-left: 22px;
}
```

Selects the below DOM elements

```
<div class="health-profile step">
   <input id="diabetesManagement" .../>
   <span>....</span>
   <span>....</span>
   <input …. />
   <span>....</span>
</div>
```

- element1 ~ element2 selector (**general sibling selector**): selects element2 that is followed by element1 and within the same parent (doesn't have to be immediately after)

```
        .health-profile input#diabetesManagement_value~span {
    display: block;
    margin-top: -22px;
    margin-left: 22px;
}
```

Selects the below HTML

```
<div class="health-profile">
```

```
    <input id="diabetesManagement_value" .../>
    <span>....</span>
    <span>....</span>
    <input …. />
    <span>....</span>
</div>
```

- DEV-4786 (C1)
  - **jQuery:**

```
$(".step.health-profile-step #diabetesManagement .radio-custom-label
span").each(function() {
     //selects text before :
     var text = $(this).text().split(':')[0];
     var boldText = '<b>' + text + '</b>';
     $(this).html($(this).html().replace(text, boldText));
});
```

$(selector).each() -- When called it iterates over the DOM elements that are part of the jQuery object

```
<ul>
 <li>foo</li>
 <li>bar</li>
</ul>

$( "li" ).each(function( index ) {
 console.log( index + ": " + $( this ).text() );
});
//output
0: foo
1: bar
```

$(this).html() -- returns “Just getting started: I’m starting a new routine (e.g., I’ve been recently diagnosed or I have a new medication regimen.)” which is the html content inside the selected span -- Just getting….

.text(): The text() method sets or returns the text content of the selected elements. When this method is used to return content, it returns the text content of all matched elements (HTML markup will be removed). When used to set content (ex: .text(“abc”)), it *replaces* the existing content for the selected DOM element.
To set or return the innerHTML (text + HTML markup) of the selected elements, use the **html()** method.
* Both .text() and .html() return string.

```
<p>Her score is <strong>100.</strong></p>
```

`$("p").text()` -- Her score is 100.
`$("p").html()` -- Her score is 100.
`$("p").text("What is <strong>Geeta's</strong> score?")` -- now it's rendered in the browser as -- What is Geeta's score?
`$("p").html("What is <strong>Geeta's</strong> score?")` -- now it's rendered in the browser as -- What is **Geeta's** score?

javascript String replace(): The replace() method searches a string for a specified value, or a regular expression, and returns a new string where the specified values are replaced.

```
var txt = "Mr Blue has a blue house and a blue car";
var res = txt.replace(/blue/gi, "red");
//gi -- global case-insensitive
```

Result - Mr red has a red house and a red car

http://stackoverflow.com/questions/4886319/replace-text-in-html-page-with-jquery

Another way of doing the same:

```
$(".step.health-profile-step #diabetesManagement .radio-custom-label
span").each(function() {
 var text = $(this).text();
 var textFront = text.split(':')[0];
 var boldTextFront = '<b>' + textFront + '</b>';
 var textBack = text.split(':')[1];
 $(this).replaceWith(boldTextFront + ": " + textBack);
});
```

jQuery replaceWith() - The .replaceWith() method removes content from the DOM and inserts new content in its place with a single call.

- DEV-4525 (C1, C2, C3, C4, C5) - Cross browser compatibility - **Safari auto-fill hack** - Assuming that feature is enabled in browser preferences (which is by default enabled) - Safari is ignoring autocomplete="off" and is autofilling username and pwd fields by detecting them. Tried the following fixes:
  - Didn't work
    - Add autocomplete="off" to all input fields which fixed the problem of browser auto-filling in Chrome and IE, but Safari's auto-fill feature is still auto-populating.
    - Add autocomplete="off" to all input fields as well as the form
    - Make the input field readonly and removed the attribute when the field is on focus or mouse entered using jQuery. You can attach events (focus click mouseover dbclick keyup keypress etc…) using **on** jquery method to the selected DOM elements or you can directly use event methods like (keydown() focus() click() etc…). The keydown() method triggers the keydown event, or attaches a function to run when a keydown event occurs.

```
$("#account-input-email, #account-input-password").on("focus  mouseover", function() {
      $(this).removeAttr("readonly");
});
```

    - Make the input field readonly and removed the attribute when the field is on focus or mouse entered using HTML DOM events

```
<input id="account-input-email"
      name="email"
      type="text"
      placeholder="{{placeholders.email}}"
      required
      form-input
```

```
                readonly
                onfocus="this.removeAttribute('readonly')"
                onmouseenter="this.removeAttribute('readonly')"
                autocomplete="off"
                ng-model="account.email"
                ng-change="onChangeEmail()"
              ng-pattern="/^[A-Z0-9._+%-]+@[A-Z0-9][A-Z0-9._-]*[A-Z0-9]\.[A-Z]{2,}$/i"
            />
```

        - Fake input fields: Safari somehow figures out what is the password field (may be detecting type=”password”) and identified username field as the the input field (type=email or text) which is close to the pwd field. Then it autofills both of them. To avoid it, have fake input fields to bluff Safari so that it autofills fake fields and not the real ones.

```
<!-- Adding a fake input to get around Safari's auto-fill feature -->
<input id="fake-pwd" type="password" value="Safari auto-fill hack" style="display:
none;" />
```

    - **Solutions worked**
        - Fake input fields with removing display:none - If the element has style display:none, it will be removed from the normal flow of page (not from the DOM) (visibility: hidden - hides the element but leaves the space as is.), and in Safari, it is not auto-filling. So here is the fix that worked:

```
<!-- Adding a fake input to get around Safari's auto-fill feature -->
<input id="fake-email" type="text" value="Safari auto-fill hack" class="novisible"
/>
.account .novisible {
  opacity: 0;
  position: absolute;
  z-index: -1
}
```

        One problem:

        When display:none is used, as the element is removed from the normal flow of page, allowing other elements to fill in, tab does not go over the hidden element during navigation, but when opacity:0 fix is used, it hides the element but, the element still takes up place and is included when you navigate using tab. To get around this, use tabindex=-1 for this element so it will be skipped from tab navigation.

- DEV-5424 (C1, C2): use of ng-class and $rootScope

```
ng-class="{'spanishMode': currentLanguage==='es'}"
```

Here if $rootScope.currentLanguage === es, then spanishMode class will be dynamically added to the class attribute as a value, and if not, removed.

CSS media queries

```
@media (max-width: 900px) {  //apply to viewport width 900px and below
   nav .navbar-info {
       font-size: 0.925em;
   }
}
```

- DEV-5431 (C1) - Cross browser compatibility
    - navigator.userAgent (to detect end user device environment - iPad/iPod/iPhone, Safari or Chrome)
    -
- DEV-5462 (C1) - Cross browser compatibility, npm install
    - placeholder attribute not supported in IE9.
    - Fixed this by using a jquery placeholder library, by installing it using npm. First it has to be mentioned as a dependency in package.json file and when you hit in the terminal npm install, all the dependencies in that file would be installed. If you hit `npm install --save jquery-placeholder` it will be first saved as an entry in package.json and will be installed under node_modules directory. Then you have to manually copy the installed js file and place it in your script files directory (common/extJS/..) and reference it from template file.
    -
- DEV-5235 (C1)
    - Angular $broadcast an event, catch the event $on, custom directives
      ```
      $scope.currentLanguage = 'es'; //can also take value of 'en'
      ng-class="currentLanguage" same as ng-class="{'es': currentLanguage === 'es'}"
      ```
- DEV-5836 (C1)
    - To disable a div, add the following CSS
      ```
      pointer-events: none;
      cursor: default;
      ```
- DEV-6157 (C1) - bootstrap modal hide/show
  ```
  $("#final-modal-registration-error").on('shown.bs.modal', function() {
    //Check if API registration error modal is open as well and hide it
    if($("#api-registration-error").hasClass("in")) {
      $("#api-registration-error").modal('hide');
    }
  });
  ```
- DEV-5856 (C1) - angular scroll to an element on a page
- (C1) - moment, JS date

```
//Add 5 days from today excluding weekends
    function addDays(date, numberOfDays) {
      var count = 0, day = date.getDate(), month = date.getMonth(), year = date.getFullYear(),
weekday;
      while (count < numberOfDays) {
        day++;
        weekday = new Date(year, month, day).getDay();
        if(weekday === 0 || weekday === 6) {
          continue;
        }
        count++;
      }
      date.setDate(day);
      return date;
    }

   moment( addDays(new Date(), 5) ).format('MM/DD/YYYY');
```

- **Nested controllers**: (Stackoverflow link)

```
//Accessing parent scope from child controller
```

View:

```
<div ng-controller="ParentCtrl">
       <div ng-controller="ChildCtrl">
              {{firstname}}  // Derick
              {{$parent.firstname}}  // John
              {{lastname}}  // Ray
              {{$parent.lastname} }  // Ray
      </div>
</div>
```

Controller:

```
var app = angular.module('myApp', []);
app.controller('ParentCtrl', function($scope) {
     $scope.firstname = "John";
     $scope.lastname = "Ray";
});
app.controller('ChildCtrl', function($scope) {
       $scope.firstname = "Derick";
       console.log($scope.firstname, $scope.$parent.firstname, $scope.lastname);
      // Derick John Ray
       //child controller will inherit all scope variables of parent
});
```

- MW (C1)
    - How to get query string variables from the request? (request parameters): you can send params in this.get(), which you can be accessed like req.query.something in node controller. Note that using req.params.something would give something form the URL path (not after the ?)
- $watch, $on, $emit, $broadcast
    - TODO
- Coach refacotring (C1) DEV-7374
    - Pay note to using callbacks vs ($scope.$watch() and $(timeout)
- S
- S
- S
-

C# .net

- https://jira.livongo.com/browse/DEV-3180 (https://github.com/livongo/admin/pull/789/files)
    - Returns user application preferences for a group of user PIDs\
- DEV-3855 https://jira.livongo.com/browse/DEV-3989 (https://github.com/livongo/admin/pull/811)
- We can filter out application before requesting data
    - **CSS:**

        - Inline (ex: span) vs. block (ex: div, p) elements (http://www.w3schools.com/html/html_blocks.asp)
        - **display: inline-block;** (no need to use clear: left for the next element that comes in new line)
        - To make three divs (div1, div2 and div3) side by side and a div4 in the next line, you have two options - 1) use **float: left** for div1, div2 and div3 and use **clear: left** for div4. 2) user **display: inline-block** for div1, div2 and div3
        - **overflow: auto** - If overflow is clipped, a scroll-bar should be added to see the rest of the content
- https://jira.livongo.com/browse/DEV-4059
- DEV-4017 (https://github.com/livongo/admin/pull/872)
- DEV-4273 (C1, https://github.com/livongo/admin/pull/916)
    - **Use of StringBuilder vs. String:** .In the .NET Framework, a string is immutable; it cannot be modified in place. The C# + concatenation operator builds a new string and causes reduced performance when it concatenates large amounts of text.
    - However, the .NET Framework includes a StringBuilder class that is optimized for string concatenation
- DEV-4078

LEARNINGS

JavaScript

- **Variable scope in JS - JS declarations are hoisted**

```
var x = 5;
(function () {
        console.log(x); //undefined
        var x = 10;
        console.log(x); //10
})();
```

This will print out **undefined** and **10** rather than 5 and 10 since *JavaScript always moves variable declarations (not initializations) to the top of the scope*, making the code equivalent to:

```
var x = 5;
(function () {
        var x;
        console.log(x);
        x = 10;
        console.log(x);
})();
```

http://stackoverflow.com/questions/500431/what-is-the-scope-of-variables-in-javascript

- **JavaScript has function-level scope**

```
<script>
var x=1;
if(x)
{
```

```
        var x=3;
        console.log(x); //3 - overwrites x=1
}
console.log(x); //3
</script>
```

This is unlike rest of of C and its family which have block level scope.

- JS variable scope - http://www.programmerinterview.com/index.php/javascript/javascript-how-var-works/

```
------
var x = 10; //this has global scope, so available even inside test()
function test() {
    console.log(x); // 10
}
test();
------
var x = 10; //this has global scope
function test( )
{
   console.log(x);  //undefined (only JS declarations are hoisted (both in functions and
blocks), not the definition/value/assignments
   var x = 11;      //local variable x declaration is hoisted (moved to top of function),
so undefined printed.
   console.log(x);  //11
}
test( );
console.log(x);  //10 - refers to global variable
------
var x = 10; //this has global scope
function test( )
{
   console.log(x);  //10 - global value of x is printed
   x = 11;          //refers to global variable x as var is not used and global variable x
value will be replaced from 10 to 11
   console.log(x);  //11
}
test( );
console.log(x);  //11
-------
var x = 10; //this has global scope
function test( )
{
   console.log(x);  //10 - global value of x is printed
   y = 11;          //implied global variable (it is not found by JS when searched up the
scope chain)
   console.log(y);  //11
}
test( );
console.log(y);  //11
-------
var x = 10; //this has global scope
```

```
function test( )
{
   console.log(y);  //control breaks here as y is not found, results in console error
"Uncaught ReferenceError: y is not defined"
   y = 11;
   console.log(y);
}
test( );
console.log(y);
```

- The function declaration or statement defines a function with the specified parameters.
- **Function declaration vs. Function Expression:** difference lies in how the browser loads them into the execution context. **Function declaration and variable declarations are always hoisted** (moved to top of execution context), but their assignment expressions are not.

```
console.log(test);  //prints the whole test function as below.
                    // function test() { return "hi"; }
console.log(test()); //bye - runs test() and as it returns "bye", prints it.
console.log(sayHello); //undefined - JS variable declarations are hoisted (mind - not an
error here. Ex; var t; console.log(t) //undefined).
console.log(sayHello()); //Console error - At this time, as definitions or initializations
are not hoisted, so cannot find function here and results in error.
var sayHello = function(){  // Function expression
   return "hi";
};
function test(){  // Function declaration
  return "bye";
}
console.log(sayHello);  //execution stops at the above console error, but if it continues,
would have printed the sayHello function
console.log(sayHello()); //hi
```

- -- for superscript, for subscript
- The content between the opening <option> and closing </option> tags is what the browsers will display in a drop-down list. However, the value of the value attribute is what will be sent to the server when a form is submitted. Note: If the value attribute is not specified, the content will be passed as a value instead.

```
  <select name="cars">
    <option value="volvo">Volvo XC90</option>
    <option value="saab">Saab 95</option>
    <option value="mercedes">Mercedes SLK</option>
    <option value="audi">Audi TT</option>
  </select>
```

- `===` vs. `==`: The == operator will compare for equality after doing any necessary type conversions. The === operator will not do the conversion, so if two values are not the same type === will simply return false. (link)
    a. `0 == '0' //true`
    b. `0 === '0' //false`
- In JS, you can create objects via 1) object initializer or 2) constructor function

```
<div id="prev"></div>
<script>
```

```
//1) Creating object via object initializer
var car = {
	model : "Maruti",
	color : "white",
	start : function(speed){  //note the usage of this to access object properties
		document.getElementById("prev").innerHTML = this.model + " " + speed;
	}
}
car.start(10);
car.cost = 1000;
console.log(car); //Object {model: "Maruti", color: "white", cost: 1000, function
start()...}

//2) Creating object via constructor function
function Bike(model,color,cost)
{
	this.model = model;
	this.color = color;
	this.cost = cost;
	this.displayBike = function(){
		var result = "A beautiful " + this.model + " bike is worth " +   this.cost;
		console.log(result);
	}
}

var myBike = new Bike("Hero Honda", "black", 25000);
myBike.displayBike(); //A beautiful Hero Honda bike is worth 25000

//3) creating object via constructor function - another variant via function expressions
(Link)

function displayCycle()
{
	var result = "A beautiful " + this.model + "cycle is worth " + this.cost;
	console.log(result);
}

var Cycle = function (model,color,cost)
{
	this.model = model;
	this.color = color;
	this.cost = cost;
	this.displayCycle = displayCycle;
}

var myCycle = new Cycle ("Schwinn", "red", 1000);
myCycle.year = 1996;
console.log(myCycle);
myCycle.displayCycle(); //A beautiful Schwinncycle is worth 1000

</script>
```

```
</html>
```

- *********

```
var animal = ["cat","dog","goat","pig","rat"];
var tempArray = animal;
tempArray.reverse();
alert(animal); // displays reversed array
```

Arrays are just one kind of *OBJECT*. And you *NEVER* can actually have a variable in JavaScript that *contains* an object. You can only have *references* to objects.
The only things in JS that are not objects are boolean and numeric values. Primitive types. Strings in many ways act as if they are primitive types, but that's because strings are immutable (you can't really change a string object). **Arrays and other objects are mutable,** so changes are seen by all referring variables.

- `JS call() and apply():` They both are used to execute a function in the context or scope of the first argument passed. They both can be called only on other functions.

```
var person1 = {name: 'Marvin', age: 42, size: '2xM'};
var person2 = {name: 'Zaphod', age: 42000000000, size: '1xS'};
var sayHello = function(){
       alert('Hello, ' + this.name);
};
var sayGoodbye = function(){
       alert('Goodbye, ' + this.name);
};

sayHello();  //Hello, undefined
sayGoodbye();  //Goodbye undefined

sayHello.call(person1);  //Hello, Marvin

sayGoodbye.call(person2); //Goodbye, Zaphod

sayHello.apply(person1);

sayGoodbye.apply(person2);
```

- **this**: In most cases, the value of *this* is determined by how a function is called.

```
       function test() {
           console.log(this); //prints the global window object
       }
       test();
       ------------
       myapp = {};
       myapp.name = "harsha"
       myapp.age = 32;
       myapp.race = "Asian";
       myapp.foo = function() {
          console.log(this); //points to myapp object and prints it - Object {name:
       "harsha", age: 32, race: "Asian", foo: function() {}};
          console.log(this.age); //32
       }
       myapp.foo();
       ------------
```

```
document.getElementById("output").addEventListener("mouseover", function(){
   console.log(this); //prints selected element
});
```

- **Printing contents of Object in JavaScript**

```
var myObj = {
          name: "geeta",
          place: "gudivada",
          education: "btech",
          age: 23,
          salaryPerHour: 45.5,
          getMonthlySalary: function() {
               return this.salaryPerHour*8*22;
          }
     };
     console.log(myObj.education); //btech
     console.log(myObj["education"]); //btech  -- mind you, "" is important
     console.log(Object.keys(myObj)); //["name", "place", "education", "age",
"salaryPerHour", "getMonthlySalary"]
     //The JavaScript for/in statement loops through the properties of an object
     for(var myKey in myObj)
     {
          console.log("key: " + myKey + ", value: " + myObj[myKey]);
     }
         key: name, value: geeta
         key: place, value: gudivada
         key: education, value: btech
         key: age, value: 23
         key: salaryPerHour, value: 45.5
         key: getMonthlySalary, value: function () {
                   return this.salaryPerHour*8*22;
              }

     console.log(myObj.getMonthlySalary()); //8008
```

- **Revealing Module Pattern Example:**

Involves concepts of closure (inner function having access to outer function environment), self invoking anonymous function, arrays

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <title>JavaScript Testing</title>
   <script>
       var myModule = (function (){

           var animal = ["cat","dog","goat","pig","rat"];
           var i = 0;

           function reverseArrayPrivate() {
```

```
            //to keep our original animal array intact, we copy our animal array into a new
array
            var newArray = Array.prototype.slice.call(animal,2);  //mind you, it's not
slice(), fun.call(arg1,..) -> call() executes fun in the context of arg1 (obj). Other args in
call() are passed to fun.
            //The above is as good as (but a more efficient one) - var newArray =
animal.slice(2); which slices the array from 2 index till end (array starts with index 0)
            console.log("new array :", JSON.stringify(newArray), "original array intact :",
JSON.stringify(animal));
            //new array : ["goat","pig","rat"] original array intact :
["cat","dog","goat","pig","rat"]
        }

        function searchRegExPrivate(pattern) {
            var ans = "";
            for(i=0;i<animal.length;i++)
            {
                if(animal[i].toString().search(pattern) !== -1) {
                    ans = ans + animal[i].toString() + " ";
                }
            }
            console.log(ans);  //cat goat rat
        }

        function reverseArrayPublic() {
            reverseArrayPrivate();
        }

        function searchRegExPublic(regEx) {
            searchRegExPrivate(regEx);
        }

        //we return an object, returning public functions. As this is wrapped in self-invoking
    function (which executes itself automatically), this is returned and assigned to the
    variable myModule.
        return {
            findReverse : reverseArrayPublic,
            search : searchRegExPublic
        };
    })();

    myModule.findReverse();

    var regExInput = /at/gi;
    myModule.search(regExInput);

  </script>
</head>
<body>
</body>
</html>
```

Also, refer this - http://stackoverflow.com/questions/2024888/a-simple-question-on-jquery-closure

- **“use strict”;**
  - Dynamically typed languages - JS, python, PHP - You don’t have to declare variables before using them.

```
myVariable = 5;
myVaraible = myVariable + 2; //mind the intentional typo, but myVaraible is created as a implied global variable
console.log(myVariable); //5
--------------
"use strict"; //mandates declaring variables before using them, throws error here
myVaraible = myVariable + 2;
console.log(myVariable);
```

  - Function declarations and variable declarations are always hoisted.

```
displayError();  //alerts “Please define the array”. Ideally, function has to be reached only when array is not defined or null.
var index = 10, arr = ["banana","orange","cantaloupe"];
if (arr==null || arr==undefined) {
     function displayError() {
       alert("Please define the array");
   }
}
```

    Executing the above in strict mode, would result in console error - Uncaught ReferenceError: displayError is not defined
  - s
  - s
  -
- setInterval() and setTimeout(): Example program which prints out a message on the screen continuously

```
<script>
window.addEventListener("load", function(){
-----------------------------------------------
Method 1
   function printMessage() {
       //some other stuff
       document.getElementById("message").innerHTML += "<b>Printing message</b><br>";
   }
   setInterval(printMessage,1000);
-----------------------------------------------
//drawback of Method 1: If printMessage() takes more than 1 sec to execute, then the behavior of the program goes wrong as setInterval keeps executing that method for every 1 sec.
Method 2 avoids that by printing message only after the stuff before that line is executed.
Method 2
   (function printMessage(){
       //some other stuff
       document.getElementById("message").innerHTML += "<b>Printing message</b><br>";
       setTimeout(printMessage,1000);
   })();
});
```

```
</script>
```

- **Wrapping the entire content of a JavaScript source file in a function block:**
  This is an increasingly common practice, employed by many popular JavaScript libraries (jQuery, Node.js, etc.). This technique creates a closure around the entire contents of the file which, perhaps most importantly, *creates a private namespace* and thereby helps avoid potential name clashes between different JavaScript modules and libraries.
  A closure contains a function and a reference to the environment in which the function was created. It is basically an inner function which has access to the environment of the outer function.
  Another feature of this technique is to allow for an *easily referenceable (presumably shorter) alias for a global variable.* This is often used, for example, in jQuery plugins. jQuery allows you to disable the $ reference to the jQuery namespace, using jQuery.noConflict(). If this has been done, your code can still use $ employing this closure technique, as follows:

```
(function($)
{
       /* jQuery plugin code referencing $ */
       $.fn.function_name = function(){};
 })(jQuery);
```

  $ (received argument) will be used as an alias for jQuery (passed argument). Other advantage of doing so is for performance. When jQuery (or $) usage is encounter inside the self invoking anonymous function, JS checks if the variable exists in the local scope, if not, then higer level scopes is checked until widow scope is reached. If we provide l*ocal scoping to the global variable* by passing jQuery in to the parenthesis, it helps in reducing the amount of overhead of looking up the $ variable.

- Coding convention: use curly brace at the end of the line in javascript, rather than on the beginning of a new line.
  It is not just for stylistic preference, there's a behavior of javascript which argues for the above convention.

```
Ex:
function foo() {
  return
  {
      bar: "hello"
  };
}
console.log(foo()); //undefined  -- ideally, it should print the whole object
```

  Reason: The reason for this has to do with the fact that semicolons are technically optional in JavaScript (although omitting them is generally really bad form). As a result, when the line containing the return statement (with nothing else on the line) is encountered in foo2(), a semicolon is automatically inserted immediately after the return statement.
  No error is thrown since the remainder of the code is perfectly valid, even though it doesn't ever get invoked or do anything (it is simply an unused code block that defines a property bar which is equal to the string "hello").

- **Native methods:**

Question: Define a repeatify function on the String object. The function accepts an integer that specifies how many times the string has to be repeated. The function returns the string repeated the number of times specified. For example:

```
console.log('hello'.repeatify(3));
```

Should print hellohellohello.

Answer

A possible implementation is shown below:

```
<!DOCTYPE html>
<html lang="en">
 <body>
   <p id="show"></p>
   <script>
       String.prototype.repeatify = String.prototype.repeatify || function(times) {
         var str = "";
         for(var i=0; i <times; i++) {
            str += this;
         }
         return str;
       }
       document.getElementById("show").innerHTML = "hello".repeatify(6); //Note to call
the method below the function as the assignment function expressions are not hoisted
    </script>
  </body>
</html>
```

The question tests the knowledge of the developer about inheritance in JavaScript and the prototype property. It also verifies that the developer is able to extend native data type functionalities (although this should not be done).

Another important point here is to demonstrate that you are aware about how to not override possible already defined functions. This is done by testing that the function didn't exist before defining your own:

```
String.prototype.repeatify = String.prototype.repeatify || function(times) {/* code here
*/};
```

This technique is particularly useful when you are asked to shim a JavaScript function.

- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/new
- http://www.toptal.com/javascript/10-most-common-javascript-mistakes
    - https://developer.mozilla.org/en-US/docs/Web/API/Document/createDocumentFragment
        - Since the document fragment is in memory and not part of the main DOM tree, appending children to it does not cause page reflow (computation of element's position and geometry). Consequently, using document fragments often results in better performance.
    -
- http://www.toptal.com/javascript#hiring-guide
- let - let statement gives you block level scope in javascript. Also, you cannot use a variable before it is declared using let. Curretnly ES6 (next version of JS) supports let (but only in strict mode.)

```
"use strict";
console.log(i); //Uncaught ReferenceError: i is not defined
let i=5;
```

```
        console.log("value of i outside the loop is "+i);

        "use strict";
        for(let i=0;i<5;i++) {
           console.log(i);
        }
        console.log("value of i outside the loop is "+i); //Uncaught ReferenceError: i is
        not defined
```

- **by value vs. by reference in JS:**

All objects interact by reference in JS and primitive types (number, string, boolean, undefined, null, symbol) interact by value.

```
  ---------------------------
      var a=3, b;  //value 3 is created in memory at a location and 'a' holds the address of
that location (a reference to that location).
      b=a; //Since 'a' and 'b' are primitve types, a duplicate of a's value (3) is created in a
different location, and now 'b' points to that new location.
      a=5;
      console.log(a); //5
      console.log(b): //3
  ---------------------------
      var obj1 = {
            firstName: "Hari",
            lastName: "Narayan"
      }; //this object is created in a location, and obj1 holds the address of this location.
      var obj2 = obj1; //Since objects in JS interact by reference, obj2 holds the same
reference as obj1, and now both obj2 and obj1 point to the same location.
      obj2.lastName = "Nandan";
      console.log(obj1.lastName); // Nandan

      //Objects are mutable whereas primitive types are not.
      //Also, passing objects to a function will be by reference, and not by value.

      function changeName(obj) {
            obj.lastName: "Kishore"
      }
      changeName(obj1); //now obj1.lastName = Kishore, obj2.lastName = Kishore.

      //Note: equals operator sets up new memory space (new address).
      var obj2 = {lastName: "Diya"}
      obj2.lastName; //Diya
      obj1.lastName; //Kishore
```

- splice(start, deleteCount, [elem1, elem2, …]); //to add, remove elements from any index
  - Start - start index specifying where to modify the array.
  - deleteCount - specifies how many elements to be removed. If 0, then no elements removed, and in this case, you need to specify elements to be added.
- **Array .sort():** Note that by default, the sort() function sorts values as strings. Because of this, the sort() method will produce incorrect result when sorting numbers. You can fix this by providing a compare function:

```
var points = [40, 100, 1, 5, 25, 10];
points.sort(function(a, b){return a-b});
```

- Regex - positive look ahead - link
- **Window.location**

```
https://coach.local.livongo.com:3001/abcd?from=meter#/
window.location is a object
{
        hash: "#/"
        host:"coach.local.livongo.com:3001"
        hostname:"coach.local.livongo.com"
        href:"https://coach.local.livongo.com:3001/abcd?from=meter#/"
        search: "?from=meter"
        pathname: "/abcd"
}

https://my.local.livongo.com:3001/#/reset-password?token=1826zKK9
Window.location {
        hash:"#/reset-password?token=1826zKK9"
        host:"my.local.livongo.com:3001"
        hostname:"my.local.livongo.com"
        href:"https://my.local.livongo.com:3001/#/reset-password?token=1826zKK
        9"
        pathname:"/"
        search: ""
}
```

- **Regular Expression:**
  - `"English begins with en and ends with sh".match(/en/)`
  - `"English begins with en and ends with sh".match(/en/g)  //global modifier`
  - `"English begins with en and ends with sh".match(/en/gi)  //case insensitive modifier`
  - `"English begins with en and ends with sh".match(/(en|English)/g) // OR - |`
  - `"English begins with en and ends with sh".match(/\b(en|English)\b/g)  // \b metacharacter - word boundary match (whole word match)`
  - `"English begins with en and ends with sh".match(/[en]/)  //[abc]- Find any character between the brackets, [^abc] - Find any character not between the brackets`
  - `"English begins with en and ends with sh".match(/[en]/i)`
  - `"English begins with en and ends with sh".match(/[en]/gi)`
- F
- f
- s

---

jQuery

- jQuery .html (.innerHTML() of JS) vs .text (.innerText() of JS) : `jQuery.html()` treats the string as HTML, `jQuery.text()` treats the content as text.

```
<body>
	<div id="div1"></div>
	<div id="div2"></div>
</body>
<script type="text/javascript">
	$(function(){ //shorter method for the document ready event
		$("#div1").html('<a href="example.html">Link</a><b>hello</b>');
		$("#div2").text('<a href="example.html">Link</a><b>hello</b>');
	});
</script>
```

Output:
Linkhello
Linkhello

- jQuery .each(): The each() method specifies a function to run for each matched element.
`$(selector).each(function(index,element))`

- jQuery Object: https://learn.jquery.com/using-jquery-core/jquery-object/ - jQuery wrapped DOM element vs. native DOM element. Also Link. It is very important to make the distinction between jQuery object and native DOM elements. Native DOM methods and properties are not present on the jQuery object, and vice versa.

```
	var firstParaNativeDom = $("p").get(0);  //selecting only the first p element on the page.
	var secondParaNativeDom = $("p")[1];
	var firstP = $("p").eq(0); //Now firstP (recommended to use variable name as $firstP to better recognize jQuery object vs. native DOM element) is a jQuery object containing only the first p element on the page.
```

Both firstParaNativeDom and secondParaNativeDom contain native DOM element and calling like firstParaNativeDom.text() will return console error as they are not jQuery objects.

- **jQuery Performance**: Caching your selections (to some variable)
Every time you do select a particular element like `$("input[type=checkbox][value=1]")`, jQuery Traverses through all Elements of DOM every time for selecting that element.

Bad way

```
$(".myList").click( function() {
   $(".myList").hide();
});
```

Good way

```
var $myList  = $(".myList");  //good way of naming convention for jquery variables to better differentiate from other variables
$myList.click( function() {
	$myList.hide();  // No need to re-select .myList since we cached it
});
```

- jQuery end(): http://stackoverflow.com/questions/1745814/what-does-the-end-function-do-in-jquery

- If you apply any destructive operation like find(), it modifies the jQuery object, with the matched elements in find(). end() reverts back
- http://stackoverflow.com/questions/21151044/how-to-clear-all-input-fields-in-bootstrap-modal-when-clicking-data-dismiss-butt

```
$('#modal1').on('hidden.bs.modal', function (e) {
       $(this)
         .find("input,textarea,select")
           .val('')
           .end()
         .find("input[type=checkbox], input[type=radio]")
           .prop("checked", "")
           .end();
});
```

- Trigger multiple events - you can write your own custom function in jquery (link)

HTML & CSS

- **CSS Unit - em (relative unit):**
  - When used to specify font-size: em unit refers to the font size of the parent element.

```
<body>
  <h1>Hello</h1>
</body>

CSS:
h1 {
  font-size: 1.5em; //means 1.5 times the font-size of body. If not font-size for body in not specified in the stylesheet, the browser will look in the user's preferences
}
```

  - For others: Relative to the font-size of the element (2em means 2 times the size of the current font).

```
h1 {
  line-height: 2em; //(twice the font-size of h1)
  margin: 1.5em;
}
```

- **Eric Meyer's reset:** CSS resets take every common HTML element with a predefined style and provide one unified style for all browsers. These resets generally involve removing any sizing, margins, paddings, or additional styles and toning these values down. Because CSS cascades from top to bottom, our reset needs to be at the very top of our style sheet.
- **Semantics**: Emphasizes using proper element to give meaning and structure to the content of the page. http://learn.shayhowe.com/html-css/getting-to-know-html/ For the longest time the structure of a web page was built using divisions. The problem was that divisions provide no semantic value, and it was fairly difficult to determine the intention of these divisions. Fortunately HTML5 introduced new structurally based elements, including the <header>, <nav>, <article>, <section>, <aside>, and<footer> elements.
- A selector's specificity weight, along with its placement in the cascade, identifies how its styles will be rendered.

```
#food {
  background: green;
}
p {
  background: orange;
}
```

Although the type selector comes after the ID selector in the cascade, the ID selector takes precedence over the type selector because it has a higher specificity weight; consequently the paragraph will appear with a green background. At times styles may not appear on elements as intended, and chances are the specificity weights of our selectors are breaking the cascade.

Specificity weight calculation:

Ex: .hotdog p.mustard, had two class selectors and one type selector. Combined, the selector has a specificity weight value of 0-2-1. The 0 in the first column is for zero ID selectors, the 2 in the second column is for two class selectors, and the 1 in the last column is for one type selector.

- Layering Styles with Multiple Classes is a good practise. The higher our specificity weights rise, the more likely our cascade is to break.One way to keep the specificity weights of our selectors low is to be as modular as possible, sharing similar styles from element to element.

```
<a class="btn btn-danger">...</a>
<a class="btn btn-success">...</a>
```

- Please keep in mind that inline-level elements will not accept the *width* and *height* properties. Block and inline-block elements will, however, accept the *width* and *height* properties and their corresponding values.
- <img> is an "inline block" element. This means that they flow inline like text, but also have a width and height like block elements.
- **Vendor prefixes:** Vendor prefixes help older browsers to support recently developed CSS features.The most common *vendor prefixes* are outlined here:

  Mozilla Firefox: -moz-
  Microsoft Internet Explorer: -ms-
  Webkit (Google Chrome and Apple Safari): -webkit-

  Some of the CSS properties that require vendor prefixes are: gradient backgrounds, box-sizing, animation & transition.
- **New CSS features:**
  - Advanced CSS selectors
    - *Pseudo classes (single colon)*: they define a special state of an element. Ex: hyperlink (:link,:visited,:hover,:active), :enabled, :disabled, :read-only, :read-write, :requied, :optional, :not(selector), :first-child, :last-child, :nth-child(n)
    - *Pseudo elements (double colon):* style specified parts of the element. Ex: ::first-line, ::first-word, ::before, ::after, ::selection
    - Attribute selectors: Ex: a[target="_blank"],[class^="top"] (class attribute value begins with top), [class$="top"] (ends),[class*="top"] (has top in it)
  - Animations and transitions
  - Gradient backgrounds (background: linear-gradient(red, black)), radial-gradient(color_top, color_bottom)) - solves the problem of attaching two different images using photo-editing softwares to have a gradient like feel. Note that background is a short-hand property, you can also use background-image for above (and even for specifying multiple backgrounds.)
  - Multiple backgrounds

```
background:url("https://s3-us-west-2.amazonaws.com/s.cdpn.io/29841/tick_1.svg")
20px 50% no-repeat, linear-gradient(red, black);
```

First value - foremost, last value - rearmost

- box-sizing, border-radius, border-image properties
- 3D CSS
- cal()
- CSS counters

- The **position** property identifies *how* an element is positioned on a page and whether or not it will appear within the normal flow of a document. This is used in conjunction with the box offset properties—top, right, bottom, and left.
  - Relative positioning:
    - Element with a position value of relative will appear within the normal flow of a document, and the original space and position of the relatively positioned element will be preserved.
    - Relatively positioned elements are moved in relation to their original position.
  - Absolute positioning:
    - Element with a position value of absolute will not appear within the normal flow of a document, and the original space and position of the absolutely positioned element will not be preserved.
    - Additionally, absolutely positioned elements are moved in relation to their closest relatively positioned parent element. Should a relatively positioned parent element not exist, the absolutely positioned element will be positioned in relation to the <body> element.
  - A fixed position element is positioned relative to the viewport, or the browser window itself.
- z-index:
  - The z-index property specifies the stack order of elements.
  - An element with a higher z-index/stack order is always in front of an element with a lower z-index/stack order.
  - Z-index will only work on elements that have a specified position (position:absolute, position:relative, or position:fixed).
- text-align: center;
  - Aligns the inner content of a block element to the center. *It affects text, all inline or inline-block elements in that container.*
- Webkit-appearance: none - to remove platform specific native styling

```
.form-question .form-question-input input,
.form-question .form-question-input textarea {
   /* Remove First and then add later*/
   -webkit-appearance: none;
   -moz-appearance: none;
   appearance: none;
}
.form-question .form-question-input select,
.form-question .form-question-input input {
   padding: 1em 3em 1em 1em;
   border: 1px solid #D1D9DB;
   border-radius: 1px;
   box-shadow: none;
   outline: 0 !important;
```

```
}
```

- **Page Layout**: using float vs. inline-block
  - http://designshack.net/articles/css/whats-the-deal-with-display-inline-block/ (whitespace issue)
  - http://www.w3schools.com/css/css_rwd_mediaqueries.asp (page layout using % width)
-

Others

- **To console.log objects in Chrome developer tools (source)**
  Chrome developer tools has this feature called 'live update' which can be confusing. When you print an object over various points in time, you don't see changes in the object in a linear fashion, instead the aggregate final value is updated lively across all points, which may cause confusion. To get around this, use:
  ```
  console.log(JSON.stringify(data));
  ```
- **XSS attack (Cross site scripting):**
  One type of security injection mechanism where attacker/hacker injects malicious javascript into the website which is run on other unsuspecting users browsers who visit the website.
- **S**
- **S**
- **s**

C#

1. DEV-3180 - User group app prefererences

The current preference names and possible values are:

Name	Description	Possible Values
lang	Language	ISO 639-1 2-letter code, UPPERCASE
locale	Locale	ISO 3166-1 alpha-2 country code, UPPERCASE
bgUnits	BG units	0=mg/dL, 1=mmol/L
wtUnits	Weight units	0=pounds, 1=kilograms
distUnits	Distance units	0=miles, 1=kilometers
prefMeth	Preferred method of communication	PHONE=phone, SMS=text message, ALL=either phone or text, UNSELECTED=not yet selected
optIn	Is user opted in?	0=no, 1=yes

smsOptOut	Has user opted out of SMS?	0=no, 1=yes

Old code:

```
 public string GetPreferenceValue(string prefName, string prefValue)
  {
         if (prefName == "smsOptOut" || prefName == "optIn")
           return (Convert.ToInt16(prefValue) == 0) ? "no" : "yes";
         else if (prefName == "prefMeth")
         {
           if (prefValue == "PHONE")
             return "phone";
           else if (prefValue == "SMS")
             return "text message";
           else if (prefValue == "ALL")
             return "either phone or text";
           else if (prefValue == "UNSELECTED")
             return "not yet selected";
         }
         else if (prefName == "distUnits")
           return (Convert.ToInt16(prefValue) == 0) ? "miles" : "kilometers";
         else if (prefName == "wtUnits")
           return (Convert.ToInt16(prefValue) == 0) ? "pounds" : "kilograms";
         else if (prefName == "bgUnits")
           return (Convert.ToInt16(prefValue) == 0) ? "mg/dL" : "mmol/L";
         return prefValue;
}
```

Refactored code:

```
  public string GetPreferenceValue(string prefName, string prefValue)
  {
    Dictionary<string, string> dict;
    string val = "";
    if (Global.PreferenceValue.TryGetValue(prefName, out dict))
    {
      if(dict.TryGetValue(prefValue, out val))
      {
        return val;
      }
    }
    return prefValue;
  }

public static Dictionary<string, Dictionary<string, string>> PreferenceValue = new
Dictionary<string, Dictionary<string, string>>
    {
        {
            "lang", new Dictionary<string, string>
            {
                {"EN", "english" },
```

```
            {"ES", "spanish" }
        }
    },
    {
        "bgUnits", new Dictionary<string, string>
        {
            {"0", "mg/dL" },
            {"1", "mmol/L" }
        }
    },
    {
        "wtUnits", new Dictionary<string, string>
        {
            {"0", "pounds" },
            {"1", "kilograms" }
        }
    },
    {
        "distUnits", new Dictionary<string, string>
        {
            {"0", "miles" },
            {"1", "kilometers" }
        }
    },
    {
        "prefMeth", new Dictionary<string, string>
        {
            {"PHONE", "phone" },
            {"SMS", "text message" },
            {"ALL", "either phone or text" },
            {"UNSELECTED", "not yet selected" }
        }
    },
    {
        "optIn", new Dictionary<string, string>
        {
            {"0", "no" },
            {"1", "yes" }
        }
    },
    {
        "smsOptOut", new Dictionary<string, string>
        {
            {"0", "no" },
            {"1", "yes" }
        }
    }
};
```

- DEV-4078 (C1) - BG Alerts History

Udemy Courses Notes

Learning Wish List

Tech

1. Lodash:
    - _.isUndefined
    - _.isEmpty
    - _.clone
    - _.map
    - _.find
    - _.getPath (from /extJS/lodash/_.object.selectors.js
2. Custom Directives Angular
3. Services/Factory Angular
4. Routing Angular

Non-Tech

1. Statistical analysis

Major projects (Livongo)

1. cigna.livongo.com - Customized registration flow for Cigna (DEV-5548)
2. Admin portal
3. Timed lockout after multiple failed login attempts (DEV-5180)
4. ng-idle
5. ES translation
6. Request scheduling through acuity (DEV-6922)
7. coach.livongo.com (DEV-7374)
8. schedule.livongo.com (DEV-6866)
9. MP coaching support

Cross-browser fixes and hacks

How can you detect broswer type?

- Using navigator.userAgent
- Using conditional commetns - in index (global scope) file, declare a variable and you can use that variable in your code to check browsers

```
<!--[if lte IE 9]>
<script type="text/javascript">
```

```
 // Hack for IE8/9 CORS - should be removed
 window.isIElte9 = true;

</script>
<![endif]-->
```

The above runs only in IE9.
In your code -

```
if (!_.isUndefined(window.isIElte9) && window.isIElte9) {
...
}
```

-

1. Placeholder color in IE vs other browsers - Input placeholder CSS different in IE10, 11 (https://github.com/livongo/memberPortal/commit/ea4c03ce9abc505ccbb590f2d13de61de6c06dce)

http://stackoverflow.com/questions/22199047/placeholder-css-not-being-applied-in-ie-11

It is important we write rule sets differently for each vendor-prefix. If we combine them using comma, then if one of them is not understood by the browser, then the whole is discarded.

```
::-webkit-input-placeholder { /* Chrome, Safari, Opera */
   font-weight: 600;
   font-size: .875em;
   color: #ADB0B2;
}

::-moz-placeholder {  /* Firefox */
   font-weight: 600;
   font-size: .875em;
   color: #ADB0B2;
}

:-ms-input-placeholder { /* IE10–11 */
   font-weight: 600;
   font-size: .875em;
   color: #ADB0B2;
}

::-ms-input-placeholder { /* Edge */
   font-weight: 600;
   font-size: .875em;
   color: #ADB0B2;
}
```

2. DEV-8083 (Apparently, older IE has issues displaying web-fonts (in our case bootstrap's glyphicon). So, replaced bootstrap's glyphicon with a standard image.)
3. DEV-8252 (glyphicon not showing up in IE)
   @media all and (-ms-high-contrast: none), (-ms-high-contrast: active) {

```
    /* IE10+ CSS styles go here */
    .form-question .form-question-input input {
        padding: 1em 3em 1em 1em;
    }
}
```

4. DEV-8475
   http://perrymitchell.net/article/xdomainrequest-cors-ie9/
   IE9 CORS
   For IE9, you make API calls via
   https://coach.integration.livongo.com/api/v1/coach/appointment/types
   Normally, you would do like
   https://mw.integration.livongo.com/v1/coach/appointment/types
5. DEV-8562 - [Coach] placeholders are empty in IE9, IE8 (used html5 jQuery placeholder plugin polyfill) - it runs only when browsers natively do not support placeholders (https://github.com/mathiasbynens/jquery-placeholder)
6.
7. S
8. S