Chapter 18

How to create and use closures,
IIFEs,
the module pattern, and plugins

Objectives

- How to use closures
- The Slide Show application
- How to use immediately invoked function
- How to work with the module pattern
- The Slide Show application with the module pattern
- How to use the module pattern to create jQuery plugins
- A Blackjack application that uses a Blackjack plugin

How to use closures

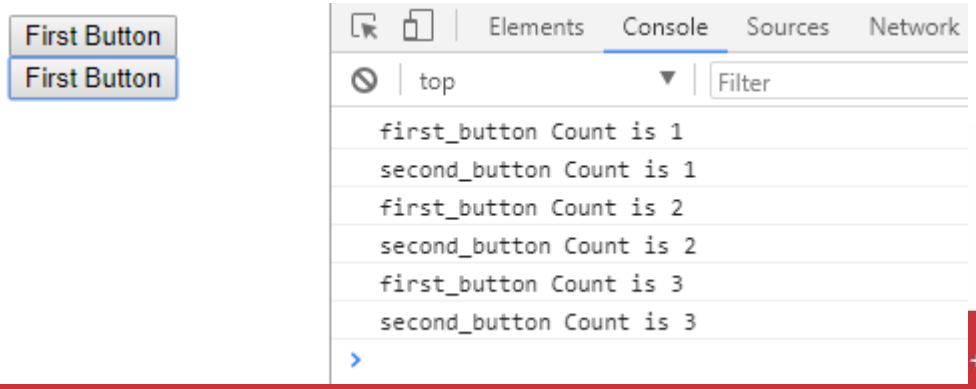
Closures in JavaScript

- Closure is a function which is created in another function (parent function). It can access the object in parent function even the parent function has finished executing and is out of scope.
- Clouse are a powerful feature of JavaScript language and they are common used in many framework.

How closures work

- An example that illustrates a closure

```
9   $( document ).ready(function(){
10      var createClickCounter = function(){
11          var counter =0; //outer scope variable with clickCounter function
12
13          var clickCounter = function(){  //Clousure function
14              counter++;
15
16              console.log(this.id + " Count is " + counter);
17          };
18          return clickCounter;
19      }
20      $("#first_button").click(createClickCounter());
21      $("#second_button").click(createClickCounter());
22  });
```



Result in console when you click on buttons

How closures work (cont.)

- When we click a button, look at return statement in line 18, it call clickCounter function. In this time the createClickCounter was finish but clickCounter function still can access counter variable in this function.
- Look at the result in console, it can separate the counter variable private for each event handler. It is advanced of closure.

How to use closures to create private state

- A function that creates a closure with private state

```
var getSlideShow =function(){
    var timer, play = true, speed = 2000;
    var nodes = { image: null, caption: null };
    var img = {catche: [], counter: 0 };

    var stopSlideShow = function(){ ... };
    var displayNextImage = function(){ ... };

    return{
        loadImages: function(slides) { ... },
        startSlideShow: function(){ ... },
        togglePlay: function(e){ ... }
    };
};
```

- Code that create and uses the slideshow object

```
var slideShow = getSlideShow();
slideShow.loadImages(slides);
```

How to work with the this keyword in closures

- Although an inner function has access to the variables in the outer function that contains it, each function has its own *this* keyword.
- If an inner function needs access to the outer function's *this* keyword, the outer function can pass it use the *bind()* method of the inner function.
- Sometimes, an inner function needs access to both the outer function's *this* keyword and it's own *this* keyword. The outer function can store the value of *this* in a variable.

How to work with the this keyword in closures (cont.)

- How to use the bind() method to set the value of an inner function's *this* keyword

```
var tax = {
    rate: 0.075;
    calc: function(sub){
        return (sub + (sub + this.rate)).toFixed(2);
    },
    displayFullPrice: function(subtotal){

        return function(){
            alert("Full price = $" + this.calc(subtotal));
        }.bind(this);
    }
};
```

- Attach the inner function as an event handler for a button

```
$("#camera").click(tax.displayFullPrice(125));
//Display "Full price =$134.38" when button is clicked
```

How to work with the this keyword in closures (cont.)

- How to use a variable to store the value of an outer function's *this* keyword

```
var tax = {
    rate: 0.075;
    calc: function(sub){
        return (sub + (sub + this.rate)).toFixed(2);
    },
    displayFullPrice: function(subtotal){

        var that = this;
        return function(){
            alert("Full price for " + this.id + "= $"
                    + this.calc(subtotal));
        }
    }
};
```

- Attach the inner function as an event handler for a button

```
$("#camera").click(tax.displayFullPrice(125));
//Display "Full price for camera =$134.38" when button is clicked
```

The Slide Show application

The Slide Show application

- The User Interface

The Slide Show application

- The HTML code

```
<head>
    <title>Fishing Slide Show</title>
    <link rel="stylesheet" href="slideshow.css"/>
    <script src="https://code.jquery.com/jquery-3.1.1.min.js"></script>
    <script src="library_slideshow.js"></script>
    <script src="main.js"></script>
</head>

<body>
    <main>
        <h1>Fishing Slide Show</h1>
        <p><input type="button" id="play_pause" value="Pause"></p>
        <p><img src="images/release.jpg" id="image" alt=""></p>
        <p><span id="caption">Catch and Release</span></p>
    </main>
</body>
```

The Slide Show application

- The JavaScript code – library_slideshow.js file

```
var createSlideshow = function() {
    // private variables and functions
    var timer, play = true, speed = 2000;
    var nodes = { image: null, caption: null };
    var img = { cache: [], counter: 0 };

    var stopSlideShow = function() { clearInterval( timer ); };
    var displayNextImage = function() {
        img.counter = ++img.counter % img.cache.length;
        var image = img.cache[img.counter];
        nodes.image.attr("src", image.src);
        nodes.caption.text( image.title );
    };
```

The Slide Show application

- The JavaScript code – library_slideshow.js file

```
    // public methods that have access to private variables and functions
    return {
        loadImages: function(slides) {
            var image;
            for ( var i = 0; i < slides.length; i++ ) {
                image = new Image();
                image.src = "images/" + slides[i].href;
                image.title = slides[i].title;
                img.cache.push( image );
            }
            return this;
        },
        startSlideShow: function() {
            if (arguments.length === 2) {
                nodes.image = arguments[0];
                nodes.caption = arguments[1];
            }
            timer = setInterval(displayNextImage, speed);
            return this;
        },
        createToggleHandler: function() {
            var me = this;        // store 'this', which is the object literal
            return function() {
                // 'this' is the clicked button; 'me' is the object literal
                if ( play ) { stopSlideShow(); } else { me.startSlideShow(); }
                this.value = (play) ? "Resume" : "Pause";
                play = ! play;    // toggle play flag
            };
        }
    };
};
```

The Slide Show application

- The JavaScript code – main.js file

```
$( document ).ready(function() {
    // create the slideshow object
    var slideshow = createSlideshow();

    var slides = [
        {href:"release.jpg", title:"Catch and Release"},
        {href:"deer.jpg", title:"Deer at Play"},
        {href:"hero.jpg", title:"The Big One!"},
        {href:"bison.jpg", title:"Roaming Bison"}
    ];

    $("#play_pause").click( slideshow.createToggleHandler() );

    slideshow.loadImages(slides).startSlideShow( $("#image"), $("#caption") );
});
```

How to use immediately invoked function expressions (IIFEs)

IIFEs introduction

- An Immediately invoked function expression (IIFEs) is a function that is define and invoked all at once.
- IIFEs are frequently used in JavaScript application.
- Some benefits of using an IIFE
  - Helps you keep variables and functions out of global scope.
  - Helps keep variables and functions from conflicting with other variables and functions that have the same name.
  - Can be use to create private state.

How to code an IIFE

- A function expression that is defined and then invoke

```
var sayHello = function(){ //define function
    console.log(“Hello”);
}
sayHello();          //invoke function
```

- An Immediately invoked function expression

```
(function(){ //define and invoke function
    console.log(“Hello”);
})();
```

- Two ways to code the parentheses of an IIFE

```
(function(){ console.log(“Hello”);})();
(function(){ console.log(“Hello”);}());
```

How to code an IIFE(cont.)

- An IIFE that keeps a variable from conflicting with another variable of the same name

```
var today = "Today is the first day of the rest of your life.";
//This today variable in the main function and store a String object

today = today + (function(){
    var today = new Date(); //This variable in IIFE and store a Date object
    if(today.getDay()===0 || today.getDay()===6){
        return "Plus, it's the weekend!";
    }else{
        return "";}
})();   //Invoke the IIFE

alert(today);   //Display the String object
```

How to use an IIFE to solve the closure loop problem

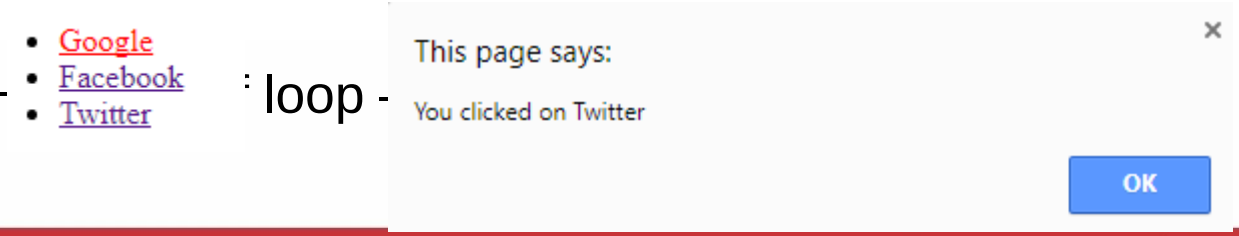
- How this affects closures in a loop
  - Assigning click event handles in a loop

```
$( document ).ready(function(){
    var topSites = ["Google", "Facebook", "Twitter"];
    var links = $("#top_sites").find("a");
    for(var i in topSites){
        $(links[i]).text(topSites[i]);

        $(links[i]).click(function(){
            alert("You clicked on " + topSites[i]);
        })
    }

});
```

  - loop -



How to use an IIFE to solve the closure loop problem (cont.)

- How to fix the loop problem by calling a function that returns a function

```
$( document ).ready(function(){
    var topSites = ["Google", "Facebook", "Twitter"];
    var links = $("#top_sites").find("a");

    var createHandler = function(name){
        return function() {alert("You clicked on " + name);};
    };

    for(var i in topSites){
        $(links[i]).text(topSites[i]);
        $(links[i]).click( createHandler(topSites[i]) );
    }
});
```

How to use an IIFE to solve the closure loop problem (cont.)

- How to fix the loop problem by using an IIFE

```
$( document ).ready(function(){
    var topSites = ["Google", "Facebook", "Twitter"];
    var links = $("#top_sites").find("a");

    for(var i in topSites){
        $(links[i]).text(topSites[i]);
        $(links[i]).click(
            (function(name){
                return function(){
                    alert("You clicked on " + name);
                };
            })(topSites[i]) //Value for 'name' parameter passed to the IIFE
        );
    }
});
```

How to work with the module pattern

The module pattern introduction

- The module pattern use an IIFE to create a single instance of the object, or module, that's returned by the function.
- That way, you get the benefits of an object literal while still having the private state of a closure.
- Module pattern makes it easy to argument an object, you can split an object among multiple file. This is helpful when you have a large code.
- Several third-party library use module pattern, including jQuery.

How to use module pattern

- A module pattern that creates a single slideshow object with private state

```
var slideshow = (function(){
    var timer, play = true;
    var nodes = {image: null, caption: null};
    var img = {cache: [], counter: 0};

    var stopSlideShow = function(){ ... }
    var displayNextImage = function(){ ... }

    //public properties and methods
    return{
        speed: 2000;
        loadImages: function(slide) { ... },
        startSlideShow: function() { ... },
        createToggleHandler: function() { ... }
    }
})(); //Invoke the IIFE to create the object
```

How to use module pattern (cont.)

- How to create a namespace
  - An object literal that's used as a namespace

```
var myapp = {};
```

  - How to ensure you don't override an existing namespace

```
var myapp = myapp || {};
```

  - A single slideshow object added to the myapp namespace

```
myapp.slideshow = (function(){
    ….
})();  //Invoke the IIFE to create the object
```

How to augment a module and use accessor properties

- To *augment* a module, you use an IIFE, and you import a module to be augmented by passing it as an argument when you invoke the IIFE.

How to augment a module and use accessor properties (cont.)

- An example that uses an IIFE to augment the slideshow object

```
(function(mod){
    mod.changeSpeed = function(speed){
        var newSpeed = parseInt(speed);

        this.speed = (isNaN(newSpeed) || newSpeed < 500) ? 2000: newSpeed;

        return this;
    };
})(myapp.slideshow);  //invoke IIFE; import the module to be augmented
```

- An example that uses the slideshow object's new method

```
$("#change_speed").click(function(){
    var ms = prompt("Enter slideshow speed in milliseconds.","2000");
    myapp.slideshow.changeSpeed(ms).startSlideShow();
});
```

How to augment a module and use accessor properties (cont.)

- How to use an accessor property to provide limited access to a private variable

```
myapp.slideshow = (function(){
    var speed = 2000, ...
    var prototype = { ... };

    var properties = {
        interval: {
            get: function(){ return speed; },
            set: function( s){
                speed = (isNaN(s) || s<500 ) ? 2000 : s;
            }
        }
    };

    return Object.create( prototype, properties );

})();        //Invoke IIFE
```

The Slide Show application with the module pattern

The Slide Show application

- The User Interface

The Slide Show application

- The HTML code

```
<head>
    <title>Fishing Slide Show</title>
    <link rel="stylesheet" href="slideshow.css"/>
    <script src="https://code.jquery.com/jquery-3.1.1.min.js"></script>
    <script src="library_slideshow.js"></script>
    <script src="library_slideshow_enhancements.js"></script>
    <script src="main.js"></script>
</head>
<body>
    <main>
        <h1>Fishing Slide Show</h1>
        <p><input type="button" id="play_pause" value="Pause">
           <input type="button" id="change_speed" value="Change Speed">
           <input type="button" id="view_slides" value="View Slides"></p>
        <p><img src="images/release.jpg" id="image" alt=""></p>
        <p><span id="caption">Catch and Release</span></p>
    </main>
</body>
```

The Slide Show application

- The library_slideshow_enhancements.js file

```
(function(mod) {
    mod.changeSpeed = function(speed) {
        this.interval = parseInt(speed);
        return this; // return 'this' so can be chained
    };
    mod.displaySlides = function() {
        var slides = this.images.map( function( current ) {
            var pieces = current.src.split("/");
            return pieces[pieces.length - 1]; // return last array element
        });
        return slides.join(", ");
    };
})(myapp.slideshow); // invoke IIFE; import the object to be augmented
```

The Slide Show application

- The JavaScript code - main.js file

```
$( document ).ready(function() {
    // create the slideshow object
    var slideshow = myapp.slideshow; // use an alias to make code shorter

    var slides = [
        {href:"release.jpg", title:"Catch and Release"},
        {href:"deer.jpg", title:"Deer at Play"},
        {href:"hero.jpg", title:"The Big One!"},
        {href:"bison.jpg", title:"Roaming Bison"}
    ];

    $("#play_pause").click( slideshow.createToggleHandler() );
    $("#change_speed").click( function() {
        var ms = prompt( "Current speed is "
            + slideshow.interval + " milliseconds.\n"
            + "Please enter a new speed in milliseconds."
        , 2000 );
        slideshow.changeSpeed(ms).startSlideShow();
    });
    $("#view_slides").click( function() {
        alert( slideshow.displaySlides() );
    });

    slideshow.loadImages(slides).startSlideShow( $("#image"), $("#caption") );
});
```

How to use the module pattern to create jQuery plugins

How to use the module pattern to create jQuery plugins

- The jQuery library uses the module pattern, so you use IIFEs to create plugins that augment that library.
- jQuery provides an API for creating plugins.

The structure of a plugin

- The module pattern of a jQuery plugin
  - An IIFE that imports the jQuery object and adds a method to its prototype

```
(function($){
    $.fn.pluginName = function(){
        //The code for the plugin
    };
})(jQuery);
```

  - A method that returns the this keyword so it can be chained

```
(function($){
    $.fn.pluginName = function(){
        //The code for the plugin
        return this;
    };
})(jQuery);
```

The structure of a plugin (cont.)

- The module pattern of a jQuery plugin
  - An method that uses the each() method so it iterates all selected element

```
(function($){
    $.fn.pluginName = function(){
        this.each(function(){
            //The code for the plugin
        });
        return this;
    };
})(jQuery);
```

  - A method that returns the this keyword so it can be chained

```
(function($){
    $.fn.pluginName = function(){
        return this.each(function(){
            //The code for the plugin
        });
    };
})(jQuery);
```

- Naming conversions for plugin files
  - jquery.pluginName.js

How to code a plugin that highlights the items in a menu

- The User Interface

- The HTML code

```
<body>
    <h1>The Highlight Menu plugin</h1>
    <nav>
        <ul>
            <li><a href="index.html">Home</a></li>
            <li><a href="faq.html">FAQs</a></li>
            <li><a href="contact.html">Contact Us</a></li>
        </ul>
    </nav>
    <main>
    </main>
</body>
```

How to code a plugin that highlights the items in a menu (cont.)

- The highlightMenu plugin in jquery.highlightmenu.js file

```
(function($){
    $.fn.highlightMenu = function() {
        return this.each(function() {
            var items = $(this).find("a");
            items.mouseover(function() {
                $(this).addClass("mouseover");
            });
            items.mouseout(function() {
                $(this).removeClass("mouseover");
            });
        });
    };
})(jQuery);
```

- jQuery that uses the highlightMenu plugin

```
$(document).ready(function() {
    $("nav").highlightMenu();
}); // end ready
```

How to add options to a plugin

- To provide options for plugin, you code a parameter for the plugin that will receive the options that the user can set.
- The highlightMenu plugin with options

```
(function($){
    $.fn.highlightMenu = function(options) {
        var o = $.extend({
            "mouseoverClass"   : "mouseover",
            "mouseoutClass"    : "mouseout",
            "useMouseout"      : true
        }, options);

        return this.each(function() {
            var items = $(this).find("a");
            items.mouseover(function() {
                $(this).addClass(o.mouseoverClass);
                if (o.useMouseout) {
                    $(this).removeClass(o.mouseoutClass);
                }
            });
            items.mouseout(function() {
                $(this).removeClass(o.mouseoverClass);
                if (o.useMouseout) {
                    $(this).addClass(o.mouseoutClass);
                }
            });
        });
    };
})(jQuery);
```

How to add options to a plugin (cont.)

- jQuery that uses the hightlightMenu plugin and sets one of its options

```
$(document).ready(function() {
    $("nav").highlightMenu({
        useMouseOut: true
    });
}); // end ready
```

A Blackjack application that uses a blackjack plugin (Page 576 - 583)

Summary

- Closure is a function which is created in another function (parent function). It can access the object in parent function even the parent function has finished executing and is out of scope.
- Although an inner function has access to the variables in the outer function that contains it, each function has its own *this* keyword.
- If an inner function needs access to the outer function's *this* keyword, the outer function can pass it use the *bind()* method of the inner function
- An Immediately invoked function expression (IIFEs) is a function that is define and invoked all at once.

Summary

- The module pattern use an IIFE to create a single instance of the object, or module, that's returned by the function.
- To *augment* a module, you use an IIFE, and you import a module to be augmented by passing it as an argument when you invoke the IIFE.
- The jQuery library uses the module pattern, so you use IIFEs to create plugins that augment that library.
- jQuery provides an API for creating plugins.

The End.