Chapter 17

How to create and use own objects

Objectives

- Basic skills for working with objects
- The Miles Per Gallon application
- How to create and call constructors
- The Trips application
- How to create a factory function
- Advanced skills for working with objects
- The Task List application

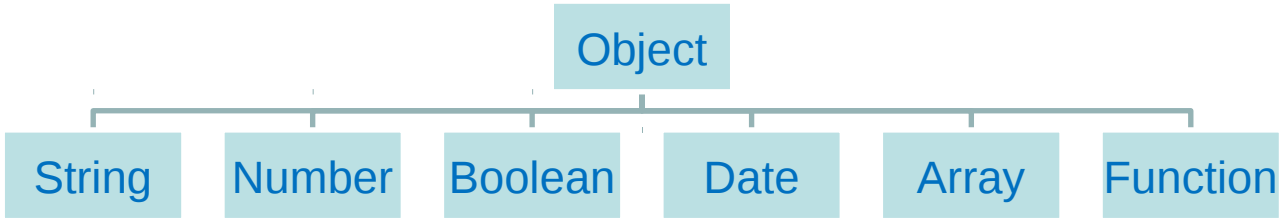
Basic skills for working with objects

Introduction to Object in JavaScript

- JavaScript permit you create an object base on the built-in class like Object, String, Number, Boolean, Date, Array, Function … They called Native Object Types.
- There are two level hierarchy of object types. The top level consists of the Object object type. The next level consist types like String, Number, Boolean, Date, Array, Function…

How to create and use the native object types

- The JavaScript hierarchy of the native object types



- The syntax for creating a new object of a native type

```
var variableName = new ObjectType(arguments);
```

  - Example create a new object of the Date type

```
var today = new Date();
```

How to create and use the native object types - Example

- Create a new object of a native type with literal values
  - Create a new object of the String type

```
var lastName = “Hopper”; //Same as new String(“Hopper”)
```

  - Create a new object of the Number type

```
var taxRate = .0875;   //Same as new Number(.0875)
```

  - Create a new object of the Boolean type

```
var validFlag = true;  //Same as new Boolean(true)
```

  - Create a new object of the Array type

```
var tasks = [];  //Same as new Array()
```

How to create and use the native object types – Example (Cont.)

- Create a new object of a native type with literal values
  - Create a new object of the Function type

```
var isValue = function(){…};  //Same as new Function('..')
```

  - Create a new object of the Object type

```
var invoice = {};
```

- Use the properties and methods of the native object types
  - Use the length property of a String object

```
length = lastName.length;  //Same as lastName["length"]
```

  - Use the toFixed method of a Number object

```
formattedRate = taxRate.toFixed(4);
                //Same as taxRate["toFixed"](4)l
```

How to create your own objects with object literals

- Object literals it mean this object is created by a literal value.
- When you create an object literal, you can add value properties by coding pairs of property names and values that are separated by colons.

How to create your own objects with object literals (cont.)

- How to initialize a new object with properties and methods
  - Example 1: initialize a new object with one property

```
var invoice = { taxRate: 0.0875 };
```

  - Example 2: initialize a new object with one method

```
var invoice = {
   getTotal: function(subtotal, salesTax){
      return subtotal + salesTax;
   }
};
```

How to create your own objects with object literals (cont.)

- How to initialize a new object with properties and methods
  - Example 3: initialize a new object with properties and methods

```
var invoice = {
    taxRate: 0.0875,
    getSaleTax: function(subtotal){
        return subtotal * this.taxRate;
    },
    getTotal: function(subtotal, salesTax){
        return subtotal + this.getSaleTax(subtotal);
    }
};
```

How to create your own objects with object literals (cont.)

- Refer to the properties and methods of an object

```
alert(invoice.taxRate); //display 0.0875
var saleTax=invoice.getSalesTax(100); //saleTax=8.75
var invoiceTotal=invoice.getTotal(100);
                    //invoiceTotal=108.75
```

- Nested objects and refer to nested properties and methods
  - How to nest one object within another

```
var invoice = {
    terms:{
        taxRate: 0.0875;
        dueDays: 30
    }
};
```

  - Refer to the properties and methods of nested object

```
alert(invoice.terms.taxRate);     //Display 0.0875
```

How to extend or modify and object

- Once an object created, you can add new properties and method to it. Also you can change the value of an existing property and change a method.

- How to add properties and methods to an object

```
    var invoice = {};
invoice.taxRate = 0.0875;  //add a property
invoice.getSalesTax = function(subtotal){
    return (subtotal * this.taxRate);
}                  //add a method
```

How to extend or modify and object (cont.)

- How to modify the value of a property

```
invoice.taxRate = 0.095;
```

- How to remove a property from an object

```
delete invoice.taxRate;
```

- Two variables that refer to the same object

```
today = new Date();
now = today;
```

JavaScript libraries introduction

- A JavaScript library is an external file that contains related functions, objects or both.
- You can create a JavaScript your self or use of third party like jQuery.
- To use A JavaScript in your web page, you must include it to your web page by using script element.

JavaScript libraries introduction (cont.)

- The benefits of JavaScript libraries:
  - They let you group similar functionality in a single file.
  - They make your code easier to understand, maintain and reuse.
  - The encourage the separation of concern.

How to create and use JavaScript libraries

- An example of a simple library file(library_mpg.js)

```
var mpg = {
    miles: 0,
    gallons: 0,
    calculate: function(){
        return this.miles/this.gallons;
    }
};
```

How to create and use JavaScript libraries

- How to include and use JavaScript library
  - Include a JavaScript library in to your web page

```
<head>
    <title>Calculate MPG</title>
    <link rel="stylesheet" href="mpg.css">
    <script src="https://code.jquery.com/jquery-3.1.1.min.js"></script>
    <script type="text/javascript" src="library.js"></script>
    <script type="text/javascript" src="main.js"></script>
</head>
```

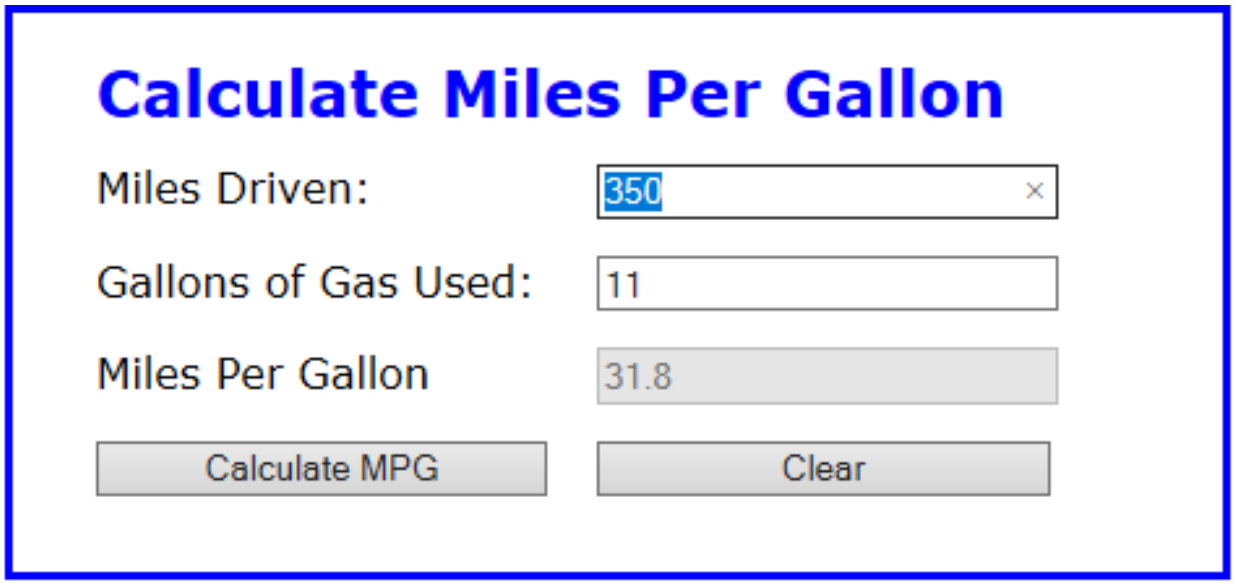
  - Use a function in JavaScript library

```
$( document ).ready(function() {
    $("#calculate").click( function() {
        mpg.miles = parseFloat( $("#miles").val() );
        mpg.gallons = parseFloat( $("#gallons").val() );
        if ( !mpg.isValid() ) {
            alert("Both entries must be numeric and greater than zero");
        } else {
            $("#mpg").val( mpg.calculate() );
            $("#miles").select();
        }
    });
});
```

The Miles Per Gallon application

The Miles Per Gallon application

- The User Interface



The Miles Per Gallon application

- The HTML code

```
<head>
    <title>Calculate MPG</title>
    <link rel="stylesheet" href="mpg.css">
    <script src="https://code.jquery.com/jquery-3.1.1.min.js"></script>
    <script type="text/javascript" src="library.js"></script>
    <script type="text/javascript" src="main.js"></script>
</head>

<body>
    <main>
        <h1>Calculate Miles Per Gallon</h1>
        <label for="miles">Miles Driven:</label>
            <input type="text" id="miles"><br>
        <label for="gallons">Gallons of Gas Used:</label>
            <input type="text" id="gallons"><br>
        <label for="mpg">Miles Per Gallon</label>
            <input type="text" id="mpg" disabled><br>

        <input type="button" id="calculate" value="Calculate MPG">
        <input type="button" id="clear" value="Clear">
    </main>
</body>
```

The Miles Per Gallon application

- The JavaScript code – The libeary_mpg.js file

```
var mpg = {
    miles: 0,
    gallons: 0,
    isValid: function() {
        if ( isNaN(this.miles) || isNaN(this.gallons) ) {
            return false;
        } else if ( this.miles <= 0 || this.gallons <= 0 ) {
            return false;
        } else {
            return true;
        }
    },
    calculate: function() {
        var mpg = this.miles / this.gallons;
        return mpg.toFixed(1);
    }
};
```

The Miles Per Gallon application

- The JavaScript code – The main.js file

```
$( document ).ready(function() {
    $("#calculate").click( function() {
        mpg.miles = parseFloat( $("#miles").val() );
        mpg.gallons = parseFloat( $("#gallons").val() );
        if ( !mpg.isValid() ) {
            alert("Both entries must be numeric and greater than zero");
        } else {
            $("#mpg").val( mpg.calculate() );
            $("#miles").select();
        }
    });

    $("#clear").click( function() {
        $("#miles").val( "" );
        $("#gallons").val( "" );
        $("#mpg").val( "" );

        $("#miles").focus();
    });

    $("#miles").focus();
});
```

How to create and call constructors

How to create your own object types with constructor functions

- A **constructor function**(or constructor) is a special kind of function that creates an object type.
- If you want to able to create multiple instances of yours owner object types, you can code constructor function for the object to be created.
- You can code the methods for a constructor on the ***prototype object*** of the object type. That way, all the instances of the project type that are created by the constructor function share the same methods.

How to create your own object types with constructor functions (cont.)

- How to use a constructor function to create an Invoice object type.
  - Code a constructor with no parameter

```
var invoice = function(){
   this.subtotal;
   this.taxRate;
};
```

  - Code a constructor with two parameters

```
var invoice = function(suntotal, taxRate){
   this.subtotal;
   this.taxRate;
};
```

How to create your own object types with constructor functions (cont.)

- How to add methods to the Invoice object type

```
//add the getTaxAmount method to the Invoice prototype
   invoice.prototype.getTaxAmount: function(){
      return(suntotal * this.taxRate);
   };

//add the getInvoiceTota method to the Invoice prototype
   invoice.prototype.getInvoiceTotal: function(){
       return subtotal + this.getTaxAmout(subtotal);
   };
```

How to create your own object types with constructor functions (cont.)

- How to create instances of the Invoice object type

```
var invoice1 = new Invoice(1000.00, 0.075);
var invoice2 = new Invoice(100.00, 0.0875);
```

  - How to access a new object's properties

```
alert(invoice1.subtotal);  //display 1000.00
alert(invoice2.taxRate);   //display 0.0875
```

  - How to access a new object's methods

```
invoice1.getInvoiceTotal();   //return 1075.00
invoice2.getTaxAmount();   //return 8.75
```

What else you should know about prototypes

- Creates two instances of the Date object type

```
var taxDay = new Date(“4/17/2017”);
var xmas = new Date(“12/25/2017”);

alert(taxDay.toDateString());    //display Mon Apr 17 2017
alert(xmas.toDateString());   //display Mon Dec 25 2017
```

- Add a method to the prototype object of the Date object type

```
Date.prototype.toNumericDateString = function(){
    var m = this.getMonth() + 1;
    var d = this.getDate();
    var y = this.getFullYear();
    return m + “/” + d + “/” + y;
};
alert(taxDay.toNumericDateString()); //display 4/17/2017
alert(xmas.toNumericDateString());   //display 12/25/2017
```

What else you should know about prototypes (cont.)

- Add an own property to an instances of the Date object type

```
taxDay.hasExtension = true;

alert(taxDay.hasExtension);   //display true
alert(xmas. hasExtension);    //display undefined
```

- Uses an own property to override a prototype property

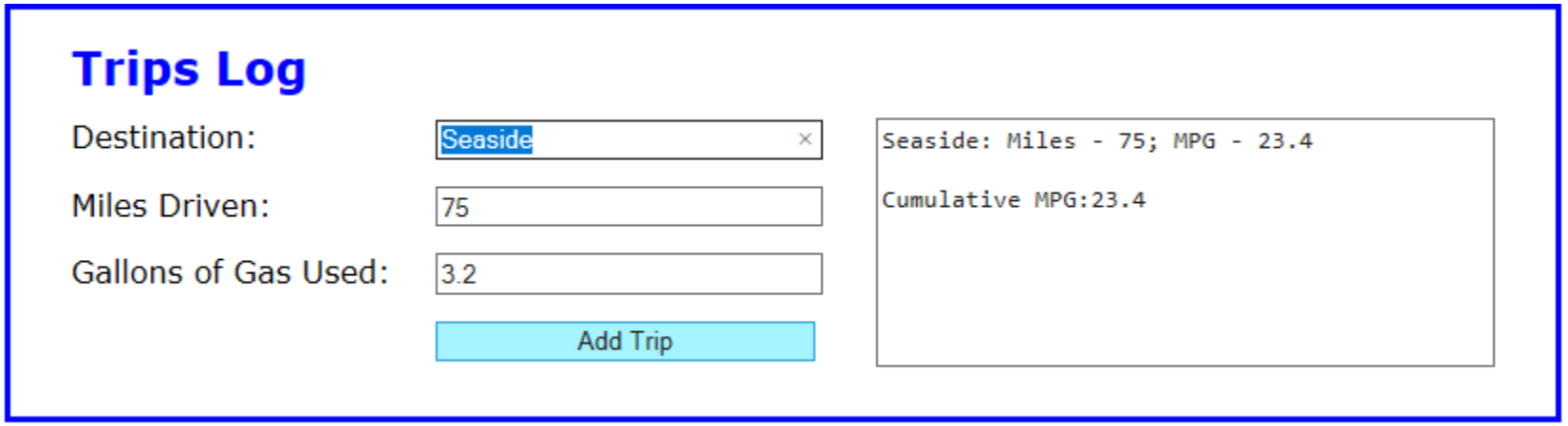
```
xmas.toDateString = function(){
    return “It’s Christmas Day”;
};

alert(taxDay.toDateString()); //display Mon Apr 17 2017
alert(xmas.toDateString());   //display It’s Christmas Day
```

The Trips application

The Trips application

- The User Interface



The Trips application

- The HTML code

```
<main>
    <h1>Trips Log</h1>
    <div id="trips">
        <textarea id="trip_list" rows="8" cols="42"></textarea>
    </div>
    <label for="destination">Destination:</label>
        <input type="text" id="destination"><br>
    <label for="miles">Miles Driven:</label>
        <input type="text" id="miles"><br>
    <label for="gallons">Gallons of Gas Used:</label>
        <input type="text" id="gallons"><br>

    <label> </label>
    <input type="button" id="add_trip" value="Add Trip">
</main>
```

The Trips application

- The JavaScript code – The libeary_trip.js file

```
var Trip = function(destination, miles, gallons) {
    this.destination = destination;
    this.miles = parseFloat( miles );
    this.gallons = parseFloat( gallons );
};
Trip.prototype.isValid = function() {
    if ( this.destination === "" || isNaN(this.miles) || isNaN(this.gallons) ) {
        return false;
    } else if ( this.miles <= 0 || this.gallons <= 0 ) {
        return false;
    } else {
        return true;
    }
};
Trip.prototype.calculateMpg = function() {
    return this.miles / this.gallons;
};
Trip.prototype.toString = function() {
    var mpg = this.calculateMpg().toFixed(1);
    return this.destination + ": Miles - " + this.miles + "; MPG - " +  mpg;
};
```

The Trips application

- The JavaScript code – The main.js file

```
$( document ).ready(function() {
    var trips = [];

    var displayTrips = function() {
        var displayString = "", mpgTotal = 0;

        for (var i in trips) {
            displayString += trips[i].toString() + "\n";
            mpgTotal += trips[i].calculateMpg();
        }
        var cumulativeMpg = mpgTotal / trips.length;
        displayString += "\nCumulative MPG:" + cumulativeMpg.toFixed(1);

        $("#trip_list").val( displayString );
        $("#destination").select();
    };

    $("#add_trip").click( function() {
        var trip = new Trip(
            $("#destination").val(), $("#miles").val(), $("#gallons").val() );
        if ( !trip.isValid() ) {
            alert("Please complete all fields. "
                + "Miles and gallons must be numeric and greater than zero.");
        } else {
           trips.push( trip );
           displayTrips();
        }
    });

    $("#destination").focus();
});
```

How to create a factory function

Factory function introduction

- A **factory function** use the *create() method* to create new objects.

- The create() method of the Object object type

Method	Description
`create(prototype, properties)`	Create a new object.

- How to use create() method to create a new object

```
var obj = Object.create(Object.prototype);
```

How to use the create() method of Object object

- A custom prototype object with one method

```
var invoicePrototype = {
   getTotal: function(){
   return this.subtotal + (
         this.subtotal + this.taxRate);
   }
};
```

  - Code that uses the custom prototype object with create() method

```
var invoice = Object.create(invoicePrototype);
invoice.suntotal = 100;
invoice.taxRate = 0.075;
alert(invoice.getTotal()); //display 107.50
```

How to use the create() method of Object object (cont.)

- A factory function that uses the create() method to create an object

```
var getInvoice = function(subtotal, taxRate){
   var invoicePrototype = {
    getTotal: function(){
      return this.subtotal +(this.subtotal + this.taxRate);
    }
   }
};
```

  - Code that uses the factory function

```
var invoice = getInvoice(100,0.075);
alert(invoice.getTotal());    //display 107.50
```

The Trips application with a factory function

- The JavaScript code – The libeary_trip.js file

```
var getTrip = function(destination, miles, gallons) {
    var tripPrototype = {
        isValid: function() {
            if ( this.destination === "" || isNaN(this.miles) || isNaN(this.gallons) ) {
                return false;
            } else if ( this.miles <= 0 || this.gallons <= 0 ) {
                return false;
            } else {
                return true;
            }
        },
        calculateMpg: function() {
            return this.miles / this.gallons;
        },
        toString: function() {
            var mpg = this.calculateMpg().toFixed(1);
            return this.destination + ": Miles - "
                + this.miles + "; MPG - " +  mpg;
        }
    };

    var trip = Object.create( tripPrototype );
    trip.destination = destination;
    trip.miles = parseFloat( miles );
    trip.gallons = parseFloat( gallons );
    return trip;
};
```

The Trips application with a factory function

- The JavaScript code – The main.js file

```
$( document ).ready(function() {
    var trips = [];

    var displayTrips = function() {
        var displayString = "", mpgTotal = 0;

        for (var i in trips) {
            displayString += trips[i].toString() + "\n";
            mpgTotal += trips[i].calculateMpg();
        }
        var cumulativeMpg = mpgTotal / trips.length;
        displayString += "\nCumulative MPG:" + cumulativeMpg.toFixed(1);

        $("#trip_list").val( displayString );
        $("#destination").select();
    };

    $("#add_trip").click( function() {
        var trip = getTrip(
            $("#destination").val(), $("#miles").val(), $("#gallons").val() );
        if ( !trip.isValid() ) {
            alert("Please complete all fields. "
                + "Miles and gallons must be numeric and greater than zero.");
        } else {
           trips.push( trip );
           displayTrips();
        }
    });

    $("#destination").focus();
});
```

Advanced skills for working with objects

How to use the arguments property of a Function object

- In JavaScript, all functions are objects, and all of the arguments passed to a function are stored in the arguments property of the Functions object.
- If a function uses the arguments property to get the arguments that are passed to it, the calling statement can pass more or fewer arguments than the parameter list specifies.

How to use the arguments property of a Function object (cont.)

- A function that use arguments property to get an argument

```
var isEven = function(){
   return arguments[0] % 2 ===0;
};
isEven(6)   //return true
```

- How to determine the number of arguments that have been passed

```
var countArgs = function(){
   alert("Number of arguments: " + arguments.lengh);
}
countArgs(1,"Text", true);
         //display "Number of arguments: 3"
```

How to use the arguments property of a Function object (cont.)

- An invoice constructor that accepts a variable number of arguments

```
var Invoice = function(taxRate){
    this.items =[];
    this.taxRate = taxRate;
    if(arguments.length >1){
        for(var i=1; i<arguments.length; i++){
            this.items.push(arguments[i]);
        }
    }
};
```

How to use the arguments property of a Function object (cont.)

- An Invoice method that provide a default value for an argument

```
Invoice.prototype.listItems = function(separator){
    var sep =(arguments.length ===0)? ”,”:separator;
    return this.items.join( sep );
};
```

- An instance of Invoice with tax rate and 3 item codes

```
var invoice = new Invoice(0.07, “ABC1”, “ABC2”, “ABC3”);
alert(invoice.listItems());   //display ABC1, ABC2, ABC3
alert(invoice.listItems(“ | ”));
                //display ABC1 | ABC2 | ABC3
```

How to create cascading methods

- A **cascading method** is a method of an object that can be chained with another methods.

- Example:

```
tasklist.load().display( $(“tasks”));
```

- To do that, the method must return a reference to the original object by using the this keyword.

How to create cascading methods (cont.)

- Two methods that modify an object but don’t return the object
  - The load and display methods of the tasklist object

```
var tasklist ={
    load: function(){
        //load code goes here
    },
    display: function(div){
        //display code goes here
    }
}
```

  - Chaining these method calls won’t work

```
tasklist.load().display( $(“task”);  //TypeError
```

  - The methods must be called one at a time

```
tasklist.load();
 tasklist. display( $(“task”);
```

How to create cascading methods (cont.)

- Two methods that modify an object and then return the object
  - The load and display methods of the tasklist object

```
var tasklist ={
    load: function(){
        //load code goes here
        return this;
    },
    display: function(div){
        //display code goes here
        return this;
    }
}
```

  - Chaining these method calls does work

```
tasklist.load().display( $("task");
```

How to inherit methods from another object

- **Inheritance** lets you share base functionality among several object.
- This let you create a base object with common methods and then **extend** it with more specialized methods.
- You can inherit the methods of an object by using **constructor** or using the **Object.create()** method.

How to inherit methods from another object (cont.)

- How to inherit methods using constructors
  - A constructor function that creates a Percent object

```
var Percent = function(){
    this.rate;
};
Percent.prototype.getPercent = function(subtotal){
    return subtotal * this.rate;
}
```

  - A constructor for a Commission object that inherits the Percent object

```
var Commission = function(){
    this.rate;
    this.isSplit;
};
Commission.prototype = new Percent(); //Inherit
Commission.prototype.calculateCommission = function(subtotal){
    var percent = this.getPercent(subtotal);
    return (this.isSplit) ? percent/2 : percent;
};
```

How to inherit methods from another object (cont.)

- How to inherit methods using constructors
  - An instance of the Commission object type

```
var commission = new Commission();
commission.rate = 0.085;
commission.isSplit = true;
alert(commission.getPercent(140));  //display 11.9
alert(commission.calculateCommission(140));
                    //display 5.95
```

How to inherit methods from another object (cont.)

- How to inherit methods using the Object.create() method
  - A percentPrototype object

```
var percentPrototype = {
    getPercent: function(subtotal){
        return subtotal * this.rate;
    }
}
```

  - A commissionPrototype object that inherits the percentPrototype object

```
var commissionPrototype =Object.create(percentPrototype);
                        //inherit
commissionPrototype.calculateCommission = function(subtotal){
    var percent = this.getPercent(subtotal);
    return (this.isSplit) ? Percent / 2 : percent;
};
```

How to inherit methods from another object (cont.)

- How to inherit methods using the Object.create() method
  - An instance of the Commission object type

```
var commission = Object.create(commissionPrototype);
commission.rate = 0.07;
commission.isSplit = false;
alert(commission.getPercent(100));   //display 9.8
alert(commission.calculateCommission(100));
                        //display 9.8
```

How to use the **this** keyword

- You've been using **this keyword** within the methods of an object to **refer to that object**
- However, value of this keyword depends on how a function is invoke

- The value of the this keyword depends on how the function is invoked

How function is invoked	Value of the this keyword
`Normal function call`	Undefined in strict mode. The Window object in non-strict mode.
`As a method of an object`	The object that contains the function.
`As an event handler`	The object that raised the event.

How to use the **this** keyword (cont.)

- Methods of a function for specifying the value of this

Method	Description
`call(thisArg[, arg1]…)`	Lets you specify the value of *this* at the time you invoke a function.
`apply(thisArg, argArray)`	Lets you specify the value of *this* at the time you invoke a function.
`bind(thisArg[, arg1]…)`	Lets you specify the value of *this* at the time you code or assign a function.

How to use the **this** keyword (cont.)

- How to use the call() method to borrow method from Array object

```
var displayArguments = function(){
    var display = Array.prototype.join.call(arguments,” ”);
    alert(display);
}
displayArguments(“Michael”, “R”, “Murach”);
                  //display Michael R Murach
```

- An apply() method that produces the same results

```
var display =Array.prototype.join.call(arguments,[“ ”]);
```

How to use the **this** keyword (cont.)

- How to use the bind() method to override *this* in an event handler

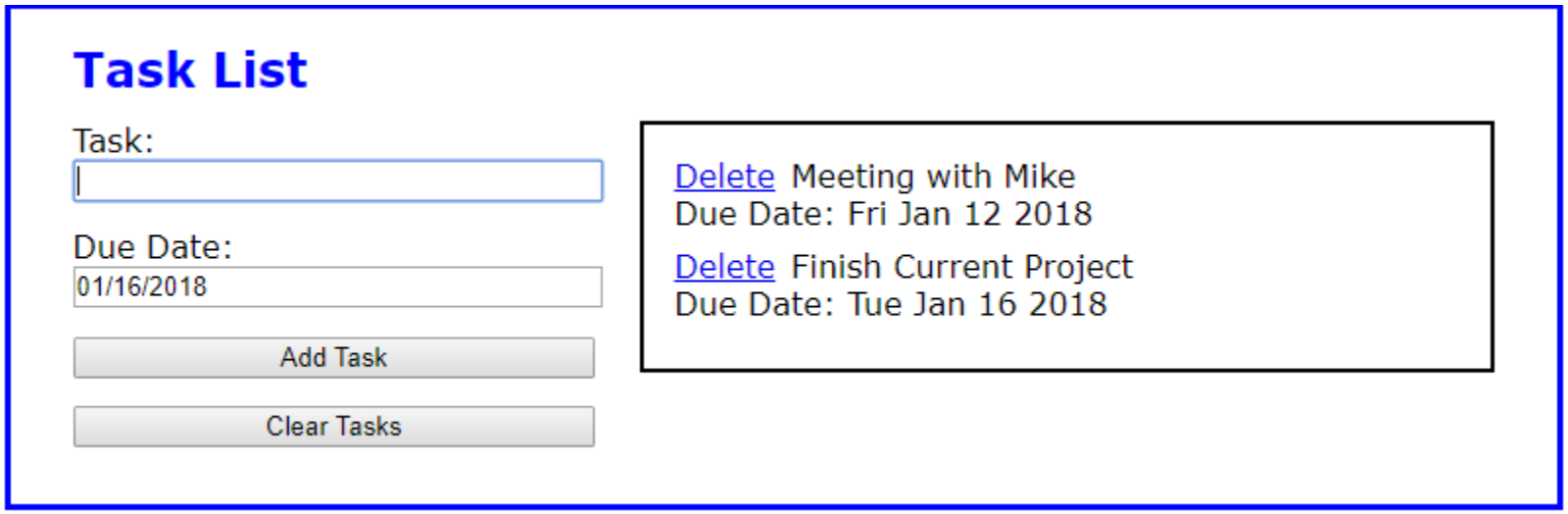
```
$(document).ready(function(){
   var saleTax = {
    taxRate” 0.08,
    calculate: function(){
       var amount = parseFloat($(“amount”).val());
       alert(“Sales tax= $” + amount * this.totalRate);
    }
   }
});

//The bind method makes sure that ‘this’ is the saleTax object
$(“sales_tax”).click(
   salesTax.calculate.bind( saleTax );
);
```

The Task List application

The Task List application

- The user interface



The Task List application

- The HTML code

```
<head>
    <title>Task List</title>
    <link rel="stylesheet" href="//code.jquery.com/ui/1.12.1/themes/base/jquery-ui.css">
    <link rel="stylesheet" href="task_list.css">
    <script src="https://code.jquery.com/jquery-3.1.1.min.js"></script>
    <script src="https://code.jquery.com/ui/1.12.1/jquery-ui.js"></script>
    <script src="library_task.js"></script>
    <script src="library_storage.js"></script>
    <script src="library_tasklist.js"></script>
    <script src="task_list.js"></script>
</head>

<body>
    <main>
        <h1>Task List</h1>
        <div id="tasks"></div>
        <label for="task">Task:</label><br>
            <input type="text" name="task" id="task"><br>
        <label for="due_date">Due Date:</label><br>
            <input type="text" name="due_date" id="due_date"><br>
        <input type="button" name="add_task" id="add_task" value="Add Task"><br>
        <input type="button" name="clear_tasks" id="clear_tasks" value="Clear Tasks">
        <div class="clear"></div>
    </main>
</body>
```

The Task List application

- The JavaScript code – The library_task.js file

```
var Task = function(task, dueDate) {
    this.text = task;
    if (arguments.length === 1) {
        this.dueDate = new Date();
        this.dueDate.setMonth( this.dueDate.getMonth() + 1 );
    } else {
        this.dueDate = new Date( dueDate );
    }
};
Task.prototype.isValid = function() {
    if (this.text === "") { return false; }
    var dt = new Date();
    if (this.dueDate.getTime() <= dt.getTime() ) {
        return false;
    }
    return true;
};
Task.prototype.toString = function() {
    return this.text + "<br>Due Date: " + this.dueDate.toDateString();
};
```

The Task List application

- The JavaScript code – The library_storage.js file

```
var getLocalStorage = function(key) {
    var prototype = {
        get: function() {
            return localStorage.getItem(this.key) || "";
        },
        set:function(str) {
            localStorage.setItem(this.key, str);
        },
        clear: function() {
            localStorage.setItem(this.key, "");
        }
    };

    var storage = Object.create( prototype );
    storage.key = key;
    return storage;
};
```

The Task List application

- The JavaScript code – The library_storage.js file

```
var getTaskStorage = function(key) {
    var prototype = getLocalStorage(key);

    prototype.retrieveTasks = function() {
        var str = this.get();
        if (str.length === 0) {
            return [];
        } else {
            var interim = str.split( "|" );
            // convert each interim string to a Task object
            return interim.map( function( current ) {
                var t = current.split( "~~" );
                return new Task( t[0], t[1] );
            });
        }
    };
    prototype.storeTasks = function(tasks) {
        if (!Array.isArray(tasks)) {
            this.set( "" );
        } else {
            // convert each Task object to an interim string
            var interim = tasks.map( function( current ) {
                return current.text + "~~" + current.dueDate.toDateString();
            });
            this.set( interim.join( "|" ) );
        }
    };

    var storage = Object.create(prototype);
    storage.key = key;
    return storage;
};
```

The Task List application

- The JavaScript code – The library_tasklist.js file

```
var tasklist = {
    tasks: [],
    storage: getTaskStorage("tasks_17"),
    load: function() {
        this.tasks = this.storage.retrieveTasks();
        return this;
    },
    save: function() {
        this.storage.storeTasks(this.tasks);
        return this;
    },
    sort: function() {
        this.tasks.sort( function(task1, task2) {
            if ( task1.dueDate < task2.dueDate ) { return -1; }
            else if ( task1.dueDate > task2.dueDate ) { return 1; }
            else { return 0; }
        });
        return this;
    },
    add: function(task) {
        this.tasks.push(task);
        return this;
    },
    delete: function(i) {
        this.sort();
        this.tasks.splice(i, 1);
        return this;
    },
    clear: function() {
        this.storage.clear();
        return this;
    },
```

The Task List application

- The JavaScript code – The library_tasklist.js file

```
    display: function(div) {
        this.sort();

        //create and load html string from sorted array
        var html = "";
        for (var i in this.tasks) {
            html = html.concat("<p>");
            html = html.concat("<a href='#' title='", i, "'>Delete</a>");
            html = html.concat(this.tasks[i].toString());
            html = html.concat("</p>");
        }
        div.html(html);

        // add onclick event handler to each <a> tag just added to div
        div.find("a").each(function() {
            $(this).on("click", function(evt){
                tasklist.load().delete(this.title).save().display(div);
                evt.preventDefault();
                $("input:first").focus();
            });
        });
        return this;
    }
};
```

The Task List application

- The JavaScript code – The main.js file

```
$( document ).ready(function() {
    $("#add_task").click( function() {
        if ( $("#due_date").val() === "" ) {
            var newTask = new Task( $("#task").val() );
        } else {
            var newTask = new Task( $("#task").val(), $("#due_date").val() );
        }

        if ( newTask.isValid() ) {
            tasklist.load().add(newTask).save().display( $("#tasks") );
            $("#task").val("");
        } else {
            alert("Please enter a task and a future due date.");
        }
        $("#task").focus();
    });

    $("#clear_tasks").click( function() {
        tasklist.clear();
        $("#tasks").html("");
        $("#task").val("");
        $("#due_date").val("");
        $("#task").focus();
    });

    $("#due_date").datepicker({
        changeMonth: true,
        changeYear: true
    });

    tasklist.load().display( $("#tasks") );
    $("#task").focus();
});
```

Summary

- JavaScript permit you create an object base on the built-in class like Object, String, Number, Boolean, Date, Array, Function … They called Native Object Types.
- A JavaScript library is an external file that contains related functions, objects or both. You can create a JavaScript your self or use of third party like jQuery.
- To use A JavaScript in your web page, you must include it to your web page by using script element.
- A **constructor function**(or constructor) is a special kind of function that creates an object type. If you want to able to create multiple instances of yours owner object types, you can code constructor function for the object to be created.

Summary

- A **factory function** use the *create() method* to create new objects.
- In JavaScript, all functions are objects, and all of the arguments passed to a function are stored in the arguments property of the Functions object.
- A **cascading method** is a method of an object that can be chained with another methods.
- You've been using **this keyword** within the methods of an object to **refer to that object**. However, value of this keyword depends on how a function is invoke.

The End.