Chapter 8

Forms

The <form> element contains controls such as input fields ,buttons, drop down menus etc. JS is now more frequently used from processing information before it is sent to the server. Values are pre-entered, entered by the user, or populated by JS.

--<form name = 'search' action= '/search'>. What does the '/' do?
Atrributres—name, action
<input>, <button>

Document.forms returns all the HTML collection of forms in the document…use index notiation to access a particular one.

Ways to call forms:

```
const form = document.forms[0];
const form = document.getElementsByTagname('form')[0];
const form = document.forms.formName;
```

to create form object…

```
const [input,button, otherFields] = form.elements;
```

To call a field within a form: const input = form.inputFieldName

Propetie and Methods:
--form.submit()—html attributes with "type = 'submit'"
--form.reset()
--form.action—can be sent to a difference url to processed

Form Events:
--focus
--blur
--change

Submit:
--add a js submit event listener.

```
const form = document.forms['search'];
```

form.addEventListener ('submit', search, false);--why false?—false refers to how the event it performed..false is the default. It starts with the most inner element and works out. "true" starts with the outer most element and works in.

```
function search() {
   alert(' Form Submitted');
}
```

Retrieving and Changing values in a Form
--value property is the text inside the field.-- definedVariable.value
--to set the value you can definedVariable.value = “new value here”

addEventListener(event, function, useCapture);---use capture is for the true or false…false is bubbling.
textContent vs innerHTML vs .createTextNode

placeholder attribute in HTML leaves a message in the field.
--autofocus attribute—gives focus to an element when the page loads.

Form Controls:
<input>
<select>--drop downlist
<textarea>--longer text entry
<button> for submitting or resetting forms

Input Fields
Default is “text”—short text. You can use the “

Attributes
--value attriute to set a default value.
--password
--name
--id

-type = “checkbox”. True or false.
--checked attribute to default to true.
hero.powers = [...form.powers].filter(box => box.checked).map(box => box.value);
[…form.powers]—turns it into an array ,filter returns only checked into “box”
.map()—replaces the check boxes with values.

Radio Buttons:
Only one can be checked.
--checked attribute

Hidden Input Field:
Send information the user has already submitted or settings.

File Input Fields: type = “file”—to upload a file.

Other Types of input fields:
Number, tel, color

Select Drop-Down List—needs “multiple”attribute for more than one option.

```
<select >
        <option value = “optionValue”></option>
        <option value = “optionValue”></option>
</select>
```

To access the text from the option:
form.city.options[form.city.selectedIndex].text

Text Areas:
<textarea>
--rows attribute
Cols attribute
Put default value between tags.

Buttons:
Type = “submit” or “reset”—reset is not good practice because everything is lost.
Disable attribute to disable
I Need a Hero!

Form Validation

Checking to see if all the required information is entered and correctly.
--the validation from JS is used to improve the user experience. The server side should still be used for security purposes.

There is a validation API that some browsers support:
Attributes
--required

Disabling the Submit Button
--use the disable attribute.

Chapter 12

Object-Oriented Programming

Encapsulation—keep everything inside an object including methods, so the outside world can’t see them.

Polymorphism—various objects can share the same methods, be also have the ability to override shared methods with a more specific implementation.

Inheritance—an exsisting object then inherit all it’s exsisitng methods and properties…with the ability to improve and add new methods and properties.

Classes are new in javascript.—they are created blueprints that help you create more objects following the blueprint---known as prototype language.

Class-based language vs prototype-based language?
Definition of literals?

Constructor Functions
An alternative way to create an object.---a function with "this" property
--instance of---a method that will tell us if a new object was created from the constructor function.
--Array object constructors need more than one value or it doesn't create the array.

Class Declarations—are newer
--class Dice {
    Constructor(sides=6) {
    }

    Method() {
    Do something
    }
}

Vs

Constructor Function
 Const Dice= function(sides = 6) {
    This.sides = sides;
    This.roll = function() {
        So something
    }
}

Class is preferred to constructor because it is more succinct, easier to read
--th constructor property returns the constructor function,
--can be used to make a copy without naming the constructor. Const newObject = new otherObject.constructor(value);

Static Methods

Methods for classes that are called by the class directly and not by the instances of.
They are called from the original class "Dice" and not redDice or greenDice.

Prototypal Inheritance

--all the objects created from a class inherit the same methods and properties.
--use the .prototype property to add new properties or classes that weren't originally added or are entirely new.

Afterwards it will be added in all new class objects.
--.this is always used to refer to the instance that the method is calling.

To find prototype property—Object.getPrototypeOf(objectName) or objectName.constrcutor.prototype. Can return Boolean…
Turtle.prototype.isPrototypeOf(raph)<<true

--hasOwnProperty()—check if method is own property or inherited from prototype—boolean

Prototypes are live. If you add a new method, all instances inherit it..even the created ones.
Can't overwrite prototypes to empty…unless constructor function is used.
--You can overwrite instances with properties or methods.
--JS first looks for own property…then goes up the chain looking for inherited.

Public and Private Methods
Create private properties so users can't access them and change them. Use functions to access and change the private properties
Private Methods

I**nheritance**
The Prototype Chain…new clases have prototypes but so do the objects. use
Object.getPrototypeOf(Object.getPrototypeOf(classObjectName))
<< {}

Null means the end of the chain

The Object Constructor
All objects inherit the prototype of the Object() constructor function…the last stop in the chain.
Enumerable means the method and properties show up in a for..in loop
propertyIsEnumerable()--Object.prototype.propertyIsEnumerable();
--best for built-in methods to nonenumberale so they don't show up and custom methods to be enumerable.

Keep classes simple and add sub-classes with extends….class subClassObjectName *extends* ObjectName {}

Polymorphism
The same method can be implemented in different ways depending on which object.

Primitive values don't have their own methods.
Every object has the toString() method…inherited from Object.prototype…can be changed to more descriptive.

Adding Methods to Built-in Objects

You can add more methods to built in objects called monkey-patching. It is very powerful but frowned on by the js community. Can be used to add methods not supported in some browsers.
Before using, check for built in methods first…be careful.

Use extend to subclass a built class and create your own.

Property Attributes and Descriptors

Objects are a collection of key-valued pairs…properties have attributes that are stored in property descriptor. Attributes, value(undefined by default), writable (changeable or not) enumerable (will show up in for-in loop), configurable (can change attributes or not.)

Oject.getOwnPropertyDescriptor(me, 'name);
Object.defineProperty(propertyItem1, propertyItem2, { attributes listed here})

Getters and Setters

Can have get() or set()…they are for property descriptors instead of attribute values.

--get() and set()
They help control the getting and setting properties in classes.
Give lots more power in controlling the way property assignment works.

Creating Objects from Other Objects

Objects can be created using other objects are blueprints to avoid clases.
Object() constructor function has method "create"

--const newObjectName = Object.create(originalObjectName);…new object inherits all the properties and methods from the original. The original is the prototype. Can add new properties. -objectName.newPropertyName = "New Property String"; or use create too.

It's a prototype and is live.

Can create super object prototypes…attaching an object prototype to another object prototype so it has both.

Const newObjectName = Object.create(originalObjectName);
newObjectName.change= function….

The new object has the properties of the original and the properties of the new.

Object Prototype Chain---happens when creating prototypes from objects…when using Object.create. The objects have all the properties and methods of the linked objects.

Mixins

A way of adding properties and methods of some objects to another object without using inheritance. Use the Object.assign() method…are problems for arrays and nested objects.

Shallow vs hard---shallow copies change with the original…hard do not.

Deep copies—assign properties from one object to another.

```
Create own mixin property
function mixin(target,...objects) {
   for (const object of objects) {
   if(typeof object === 'object') {
      for (const key of Object.keys(object)) {
         if (typeof object[key] === 'object') {
         target[key] = Array.isArray(object[key]) ? [] : {};
         mixin(target[key],object[key]);
         } else {
         Object.assign(target,object);
         }
      }
      }
   }
   return target;
}
```

--can add in properties all at once.
--can also use a copy method to make an exact deep copy of an object.

```
Copy function
function copy(target) {
   const object =  Object.create(Object.getPrototypeOf(target));
   mixin(object,target);
   return object;
}
```

Target is the object to be copied. Mixin function is then used to add properties.

Factory Functions-a function that can be used to return an object.

```
function createSuperhuman(...mixins) {
   const object = copy(Superhuman);
   return mixin(object,...mixins);
}
```

Can use as an object literal---const hulk = createSuperhuman({name: 'Hulk", realName: 'nameHere', etc});

Use with Modular functionality…attach other objects to original object.

Chaining Functions

If a method contains "this" it can be chained together to form a sequence. Code can be on the same line and makes it concise.

Binding this

It allows us to create generalized methods that refer to properties specific to a particular object. Be aware of a certain problem when a function is nested inside another function, which can often happen when using methods in objects, especially ones that accept callback functions.

Set that to this const that=this; and use that in the nested function.
--bind()
--use for-of Instead of forEach()…it does not require a nexted function.
--use arrow funtions. They remain bound without "this"

Borrowing Methods from Prototypes

Use call() method to borrow another object properties or methods.

Composition Over Inheritance

Many times you can inherit much more than you need. This is called the Gorilla Banana problem. To solve this…use composition over inheritance. Create small objects that describes singles tasks or behaviors and use them for building blocks for more complex objects.
--keep inheritance chains short.
—just borrow a method… not inherit the whole object
—or put the method in its own object.

CHAPTER 15—Modular JavaScript

A module is a self-contained piece of code that provides functions and methods that can be used in other files and by other modules.

Help with public APIs and keep implementation hidden.

Coupling…hard coupling vs loosely coupling. Both scenarios would probably use connect() …loose coupling is more flexible and less restrictions.

ES6 Modules
Code in self-contained files.
--Don't need to use strict…can't opt out of strict.
--has global scope
--top level is undefined instead of global object
--can't use HTML style comments in modules

The files uses keyword export to specify values or functions that are made available to others.

--export const objectName = value;
--import {objectName" from './filename.js';
To export function. Export {value, value}
--import {value, value} from './filename.js';

*--wildcard. -import * as stats from './filename.js';…stats becomes namespace
--export default objectName to export a single function.
--import objectName from '.filename.js';

Node.js Modules
Use node REPL