Week 7 Reading Notes

CHAPTER 11

Function Properties and Methods

Functions have their own methods and properties.
--call() method can be used to set the value of "this" inside a function to an object as the first argument.---if the function doesn't have "this" in it, then replace the first argument with "null".
--apply()—same as call method except that it uses an array as argument. However the spread operator can now be used to split an array of values into separate parameters.

Costume Properties
You can add your own properties to functions—functionName.propertyName = "the value of the function property."

Memorization—you can put data in the cache property so that the function doesn't have to calculate an operation again…saves times.

What does it mean when you put "!" in front of a variable.

IMMEDIATELY INVOKED FUNCTION EXPRESSIONS

IIFE—an anonymous function that is invoked as soon as it's defined.—(function(){do something here;})();

Temporary Variables
Placing code that uses the temporary variable inside an IIFE will make it only available for that IIFE.

Initialization code
Because IIFE's only run once, there is no need to create any reuseable named functions…all variables are temporary.
---*you can now accomplish the same IIFY thing by placing the code in block {}

Safe Use of Strict Mode
--when you use strict mode in a whole file it can create problems when working this other's code…to avoid this, you can put "use strict" in the (function (){})
IIFE's have their own private scope.

FUNCTIONS THAT DEFINE AND REWRITE THEMSELVES

Functions can call itself, define itself, and rewrite itself.
--A function can be assigned another variable…which maintains the original function definition and not be rewritten….example was the party function and the beachParty function.
Lazy Definition Pattern is used when some initialization code is required the first time it's invoked.

Init-Time Branching

Create functions that can rewrite themselves.---you can use a if/else statement to determine which way the function should be written.

RECURSIVE FUNCTIONS

A function that invokes itself until a certain condition is met. It's a useful tool to use when iterative processes are involved.

CALLBACKS
Functions passed another function as arguments and then invoked inside the function they are to passed to.

Event-driven Asynchronous Programming.
--JS is s single-thread environment….one piece of code at at time
--code can run out of order by using callback functions. This is called asynchronoouly.
Functions will not block the resto of the program from running…called event looping.
A callback always has to wait for the current execution is invoked.

Callback Hell refers to messes of lots of callbacks.

Promises
A promise represents the future result of an asynchoronous operation.
Life Cycle of a Promise
--created—calls an asynchronous operation
--pending—remains while operation taking palce
--unsettled
--settled: once completed.
The outcome is ether resolved or rejected.

Creating a Promise
--a promise is created using a constructor function…which takes an executor as an argument.
The executor initializes the promise and starts the operations.

Dealing with a settled promise
--then() method is used to deal with the outcome.
--catch method is used to deal with a failed outcome.

Chaining Multiple Promises
The then() method can be used to chain code together sequentially.

Async Functions---the async keyword allows you to write asynchronous code as if it was synchronous.

Use “async” before the function and “await” operator will wrap the return value of the function in a promise that can then assigned to a variable.

Generalized Functions—callbacks can be used to build more generalized functions…instead of having lots of specific functions.

Functions can return functions

CLOSURES

You can’t access variables outside of its scope unless you use closures.—they can be kept alive using closures…after the function has been invoked.

A closure is a reference to a variable that was created inside the scope of another function, but is kept love and used in another part of the program.

Returning Functions
A closure is formed when the inner function is returned by the outer function, maintaining access to any variables declared inside the enclosing function.

A closure is formed when a function returns another functon that then maintains access to any variable created in the original function’s scope.

Closures not only have access to variables declared in a parent funciton’s scope, they can also change the value of these variables.

Generators are special functions used to produce iterators that maintain the state of a value. Calling a generator function doesn’t actually run any of the code in the function. It returns a generator object that is used to create an iterator that implements next() method.

Use the * in the generator functions
--function* exampleGenerator(){do something}

Generator functions employ the special yield keyword that is used to return a value. The difference between the yield and return is that yield remembers the state the return value was in.

FUNCTIONAL PROGRAMMING

JS supports functional-style programming.

Pure Functions: uses the following rules:
1—the return value of a Pure function should only depend on the values provided as arguments.

2—There are no side-effects…doesn't change values or data elsewhere.

3—referential transparency—the same argument returns the same result.

--they also need at least one argument.
--and a return value.

Pure functions help make functional programming code more concise and predictable than other programming styles.

Using "const" will help avoid creating destructive data transformations.

Pure functions are the building blocks of pructional programming.

Use function composition…combining pure functions together to complete more complex tasks

By separating pices of functionality into individual functions, we can compose more complex functions.

Higher-Order Functions
Functions that accept another function as an argument or return another function as a result.—a core tenants of functional programming.

Currying
The process that involves the partial application of functions…like when not all the arguments have been supplied. IT allows you to turn a single function into a series of funtions.

Try keyword ????—research

Chapter 13

Ajax is a technique that allows web pages to communicate at the same time with a server…pages can be updated without being reloaded.

CLIENT AND SERVERS

A client will request resources from the server…the browser is a client.
The server processes the request and sends it back ot the server.

AJAX allows the client to request resources from the servier…usually JSON..Node.js is an AJAX.

BRIEF HISTORY OF AJAX
--asynchoronous loading didn't start until some Google featuresin 2004 and 2005.

JS was considered frontend…AJAX enabled JS to send requests and receive responses from a server.

THE FETCH API

The living standard for requesting and sending data asynchronously.

Basic Usage
A global method—fetch()…one argument which is a URL
---fetch (website address)
.then(code that handles the response)
.catch (code that runs if an error is returned)

Response Interface
--properties and methods that help deal with the object returned.
Ok property o see if the response was successful
Status property—returns meaningful codes
Headers, url, redirected, type—other properties of the response object.

Redirects
The redirect method is used to redirect to another URL…but not supported in any browser.

Text Responses

The text() method takes the stream of text and is then treated as a string in JS.

File Responses
Blob() is used to read a file of raw data…like image or spreadsheet.

JSON responses
Most common format for AJAX—json() is used to deal with streaming of JSON data into a promise that resolves to a JS object.

Create Response Objects—you can create your own response objects under a constructor

Request Interface

Request objects can be used to refine control.

Request() constructor…following properties: url, method, headers, mode, cache, credientials, redirect. Example:

```
const request = new Request('https://example.com/data', {
   method: 'GET',
   mode: 'cors',
   redirect: 'follow',
   cache: 'no-cache'
});
```

Request object can be used as the parameter in fetch()

Header Interface—passes along additional information…file-type, cookie info, etc.
They can be created using an instance of a constructor function. Const headers = new Headers();

RECEIVING INFORMATION

See Exercises for more info

SENDING INFORMATION

See Exercises for more

FORM DATA
--the formData interface makes submit information in forms easier using AJAX

A formData is created using an instance from a constructor function…passed as an argument then all the data becomes serialized…reduces the amount of code needed when submitting forms.
--formDAta really helps when uploading files.

A LIVING STANDARED

The Fetch API is the living standard which means that things are always changing.—stay up to date…there is a ajax() method.

Other Notes this week….

Steps for planning a project…

1. Decide on your project
2. Check online for similar projects
3. Choose your language and tools
4. List all features and entities
--include all essential and bonus features.
--divide into essential and non-essential
--group similar features together (continually evaluate for change)
--What entities interact to make it happen…repeat for each function
--what functions, think of similarities and encapsulate.

5. Map the project architecture
6. Mark entities for setup
7. Add pseudocode to your diagram
8. Make a schedule
Wrapping up

--localStorage.setItem, localStorage.setItem, localStorage.remove ('keyName', data);
Example---local.Storage.setItem('keyName', dateItem);

To store multiple key/pairs:

Stringify when writing.
Needs to store as string in
localStorage.setItem("objectName". JSON.stringify(objectName));---to format for storage.

Parse when reading
To get it out, it has been saved as JSON format so it needs to be parsed.---fixed from the string type.
-let localData = JSON.parse(localStorage.getItem('siteData'));