Chapter 11

-First-class objects – can have properties and methods themselves

-call() method, length method, can create your own

-Memoization

-can save in a cache so we can retrieve previous results later

```
                function square(x){
    square.cache = square.cache || {};
    if (!square.cache[x]) {
        square.cache[x] = x*x;
    }
    return square.cache[x]
}
```

```
function square(x){
    square.cache = square.cache || {};
    if (!square.cache[x]) {
        square.cache[x] = x*x;
    }
    return square.cache[x]
}
```

-Immediately Invoked Function Expressions

-To immediately invoke a function, place (); parenthesis at the end of the function and then turn the function into an expression by wrapping the whole function in parenthesis.

-Temporary Variables

-make sure you keep variables inside the scope needed and don't make them global unless you'll actually need them later. IIFE's help to do this.

- Initialization Code

-use IIFE to create code that will only need to be run once and then the variables can go away (could also potentially be achieved by just using block scope

-Safe Use of Strict Mode

-putting 'use strict' at the beginning of a file may cause problems especially if using someone else's code. If you put it in a IIFE it is only used that one time.

-**Functions that define ad rewrite themselves**

-a function can assign itself to a new variable runs once and then will only run as it is redefined.

-be careful because you can lose properties when it redefines

-used mostly when you need an initialization just one time and then the function can define itself as it really is.

Init-Time Branching

-can check for browser support in an if statement first, and if so run the function. Else run something that all browsers support.

Recursive Functions

-function that invokes itself. MUST HAVE AN END SO IT DOESN"T RUN FOREVER (used in previous python course)

Callbacks – pass function as argument and then invoke the function inside

Event-driven Asynchronous Programming

-when an event happens (click, focus, keyup) and the code runs out of order based on when that event occurred.

-can also set functions based on amount of time. But has to wait for current stack to be complete before executing.

Callback Hell

-using a lot of asynchronous methods can result in spaghetti code and make it hard to find what you need.

Promise – something to be called asynchronously in the future.

Promise Life Cycle

-resolved or rejected – worked or didn't work

-Super Promise – the pending phase of a promise

Creating a Promise

-constructor function (resolve, reject) =>

Settled Promise

-the then portion of the if can happen

Chaining Multiple Promise

-

Async Functions – preceded by async, lets you just write your asynchronous code like it was synchronous.

Generalized Function – functions can have functions as callbacks in the parameters

Functions that Return Functions – function can return itself so you can change the value in a variable

Closures

Function Scope – location of variable determines where it can be used. Inside block is local scope, outside block global scope

Var, const, let also help determine scope

Returning Functions – a closure happens when the inner function that is returned has access to the variables in the outer function

Generators – used to produce iterators that keep the state of a value

-use * after the word function* name(){}

-does not run any of the code in the function, just stores it as an object

-use yield keyword (same as return, but is remembered the next time it's called)

-can step through a sequence like Fibonacci number using .next()

Functional Programming – when a language only uses functions

Pure Function – 1 – return value only relies on values as arguments, not values from somewhere else

2- No side-effects. Doesn't change data anywhere else. Non-destructive

3 – Given the same argument, you'll always get the same result

(must have at least one argument and return a result)

Using const will help you not change variables but return new values

Higher-Order Functions –

Accept function as argument or return function as result. Or both

Currying –

Partial application of functions