

WEEK 07 READINGS:TROY DAVIDSON

FUNCTION PROPERTIES AND METHODS

The `call()` method can be used to set the value of `this` inside a function to an object provided as the first argument.

Memoization: What the function returns.

IMMEDIATELY INVOKED FUNCTION EXPRESSIONS.

This is easily achieved by placing parentheses at the end of the function definition (remember, we use parentheses to invoke a function). The process also has to be made into an expression, which is done by placing the whole declaration inside parentheses, as in this example:

TEMPORARY VARIABLES: There is no way to remove a variable from a scope once it's been declared. If a variable is only required temporarily, it may confuse if it's still available later in the code. It can also cause issues in your code later, such as this example:

```
let a = 1;
let b = 2;

(()=>{
  const temp = a;
  a = b;
  b = temp;
})();

a;
<< 2

b;
<< 1

console.log(temp);
<< Error: "temp is not defined"
```

INITIALIZATION CODE: An IIFE can be used to set up any initialization code that there'll be no need for again. Because the code is only run once, there's no need to create reusable, named functions, and all the variables will also be temporary. When the page loads, an IIFE will be invoked once and can set up variables, objects, and event handlers

```
(function() {  
    const name = 'Peter Parker'; // This might be obtained from a cookie in real life  
    const days = ['Sunday', 'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday'];  
    const date = new Date(), today = days[date.getDay()];  
    console.log(`Welcome back ${name}. Today is ${today}`);  
  
})();  
<< 'Welcome back Peter Parker. Today is Tuesday'
```

STRICT MODE: Be careful of using 'use strict' because you have no idea if those who also are working on the code are coding in strict mode. Best to operate strictly in a function or method, so its scope is just used in that function.

SELF-CONTAINED CODE BLOCKS: helps enclose a block of code inside its private scope.

SELF-DEFINING FUNCTIONS: Functions that define and rewrite themselves.

RECURSIVE FUNCTIONS: A recursive function invokes itself until a specific condition is met.

CALLBACKS: Callbacks can be used to facilitate event-driven asynchronous programming. Which means in programming when functions run out of order. Using events, you can trigger specific parts of your programming to be able to create a dynamic program.

Be careful of Callback hell. Too many callbacks nested within functions can make it a challenge to debug, and you are left with spaghetti code.
Example:

```

login(userName, function(error,user) {
  if(error){
    throw error;
  } else {
    getPlayerInfo(user.id, function(error,info){
      if(error){
        throw error;
      } else {
        loadGame(info, function(error,game) {
          if(error){
            throw error;
          } else {
            // code to run game
          }
        });
      }
    });
  }
});

```

PROMISES: When a promise is created, it calls an asynchronous operation and is then said to be pending. It remains in this state while the operation is taking place. A promise is created using a constructor function. This takes a function called an executor as an argument.

```

const promise = new Promise( (resolve, reject) => {
  // initialization code goes here
  if (success) {
    resolve(value);
  } else {
    reject(error);
  }
});

```

CLOSURES: You can create functions that hold other functions that allow access to the variables that are generally not within scope. As long as it is within the outer function. You can grab the variable to use with the inner function from the outer function.

GENERATORS: This creates a function but does not run the code. It returns an object.

```
function* fibonacci(a,b) {  
  let [ prev,current ] = [ a,b ];  
  while(true) {  
    [prev, current] = [current, prev + current];  
    yield current;  
  }  
}
```

The code starts by initializing the first two values of the sequence, which are provided as arguments to the function. A while loop is then used, which will continue indefinitely because it uses valid as its condition, which will always be true. Every time the iterator's next() method is called, the code inside the loop is run, and the next value is calculated by adding the previous two values together.

INTRODUCTION TO FUNCTIONAL PROGRAMMING.

Another paradigm idea like OOP. Allows code to be created strictly with just functions. Need to for objects or other coding ideas. It can be used with different styles depending on the languages you use. JavaScript excels with this idea because it is a first-class language.

Higher-order functions are a part of functional programming. Allows you to return a function within a function.

Currying: Currying is a process that involves the partial application of functions.