Advanced Javascript

Day 11- Arrow functions, Closures

Arrow Functions

( ) =>

Another way of writing functions. So, what’s the difference?

Arrow functions are a simpler and more compact way of writing functions in JavaScript. Two factors influenced the introduction of arrow functions:

1. The need for shorter functions
2. Behavior of the this keyword.

```
double = (x) => {
    return x*2
    }
```

is the same as

```
function double(x){
    return x*2
   }
```

The new arrow functions also ensures that 'this' is handled better.

Arrow functions do not have their own this keyword property.

How do functions behave inside objects currently?

```
var myobj = {
    i: 10,
    b: () => console.log(this.i, this), //arrow function
    c: function() {
      console.log(this.i, this);
    } //Normal function
  }

  myobj.b(); // prints undefined, Window {...} (or the global object)
   myobj.c(); // prints 10, Object {...}
```

Closures

Let’s revisit functional execution context and local variable

Closure

A closure gives you access to an outer function’s scope from an inner function. In JavaScript, closures are created every time a function is created, at function creation time.

What would a nested function need?

- There must be a nested function (a function inside another function).
- This nested function has to refer to a variable defined inside the enclosing function.
- The enclosing function must return the nested function.

```
function sum(x){
    return function(y){
        return x + y
    }
  }
```

The function sum, returns an inner function.

But after it returns the inner function, it can be seen that the variables inside require x, from the outer function sum.

It should throw a ReferenceError, but it does not.

This is happening as Javascript creates a closure.

Currying

Currying is a technique of evaluating function with multiple arguments, into sequence of functions with single argument.