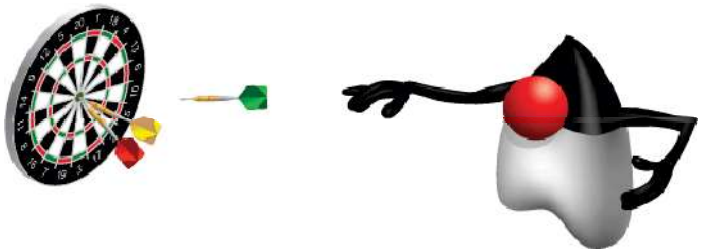
Advanced JavaScript

Objectives

After completing this lesson, you should be able to do the following:

- Define functions as value types
- Create closures and explain the variable scope
- Write JavaScript functions as modules
- Create JavaScript timers and delays to create animations in HTML



Topics

- JavaScript Functions
  - Functions as Values
  - Functions and Variable Scopes
  - Modules
- Closures
- Prototypes
- Delays and Intervals

Topics

- JavaScript Functions
  - Functions as Values
  - Functions and Variable Scopes
  - Modules
- Prototypes
- Delays and Intervals

JavaScript Functions as Values

In JavaScript, functions are another value type.

```
var sum = function(a,b){return a+b;}
```

A function can expect a function as a parameter.

```
function invoke(fn){ return fn(2,3); }
```

You can use a function as a parameter.

```
invoke(sum); // 5
```

JavaScript Function: `this` Reference

In JavaScript, all functions have a `this` reference.

```
function fun(){
  console.log(this.toString());
}
```

`this` refers to the object that is used to invoke the function.

```
var object = {value:12, fun:function(){
  console.log(this.value);
}};
object.fun() // prints 12
```

Function Methods

- `call` and `apply` methods invoke a function by setting the `this` property for the current invocation.

```
function myFunction() {
   console.log(this.myProperty);
}
var myObj = {
   myProperty : 15
};
myFunction.call(myObj);
15
myFunction.apply(myObj);
15
```

Function Methods

- `Function.prototype.toString()`:

```
>> outer.toString()

"function outer() {
    var x = "I am declared in the outer function";
    function inner()
    {
      console.log(x); } inner();
}"
```

- `Function.prototype.bind()`:

```
var myObj = {
    myVar : "Hello there!",
    greet: function () {
      innerGreet = function(name) {
       console.log(this.myVar + name);
      };
      innerGreet.bind(this," Duke")();
    }
};
```

myObj.greet() returns

"Hello there! Duke"

Function Default Value

If you want to use a default value for a function parameter, you must check the value for each parameter.

```
function x(a){
  a = a || "defaultValue";
  console.log(a);
}
x(); // defaultValue;
x(null); // defaultValue;
x("a"); // a
x(false); // defaultValue;
```

Callbacks

- Functions as parameters are commonly found as callbacks.
- Callbacks are functions that are invoked when a process is complete or some condition is met.
- You have been using callbacks with the HTML5 events!

```
window.addEventListener("load",
   function(event){ … } );
```

Callback!

Callbacks

You can even declare the callback function separately.

```
var onWindowLoad = function(event){ … };

window.addEventListener("load", onWindowLoad);
```

Use a function as a parameter.

Caution

A common mistake is invoking the function when it is used as a parameter instead of sending the function.

Wrong:

```
window.addEventListener("load",onWindowLoad());
```

Function invocation

Right:

```
window.addEventListener("load", onWindowLoad);
```

JavaScript Array Extended API

JavaScript Arrays contain many methods that use closures for array manipulation.

- `Array.sort(callback)`
  `function callback(val1, val2)`
  callback returns 0 if values are equal, a positive value if `val1` should go before `val2`, and a negative value if `val1` should go after `val2`.

JavaScript Array Extended API

- `Array.forEach(callback)`
  `function callback(value, index, array)`
  executes the callback for each element in the array.

- `Array.every(callback)`
  `function callback(value, index, array)`
  returns true if the callback returns true for all the elements in the array.

JavaScript Array Extended API

- `Array.some(callback)`
  `function callback(value, index, array)`
  returns true if the callback returns true for at least one of the elements in the array.

- `Array.filter(callback)`
  `function callback(value, index, array)`
  creates a new array with the elements that returned true when the `callback` function was applied.

JavaScript Array Extended API

- `Array.map(callback)`
  `function callback(value, index, array)` creates a new array with the results of the callback when it is called in each of the elements of the array.

- `Array.reduce(callback)`
  `function callback(previousValue, currentValue, index, array)` applies the callback function for each element in the array, and returns a single value based on the accumulated result of the callback.

Function Scope

```
function x(){
  var name="john";
  function y(){
    console.log(name);
  }
}
```

Variable can be resolved because it is declared in an outer function.

```
function outer() {
    var x = "I am declared in the outer function";
    function inner() {
        console.log(x);
    }
    inner();
};
outer();
```

Prints the content of x

Overriding Variables

You can override variables by declaring them again in inner scopes.

```
var name = "Ed";
function x(){
  var name = "John";
  console.log(name);
}
console.log(name); // Ed
x(); //John
```

Scopes and Variables

```
var name = "Ed";
function x(){
  console.log(name);
}
x(); // Ed
name = "John";
x(); // John
var name = "Peter";
x(); // Peter
```

Closures

- A function inside another function is called a closure.

```
function add(number1, number2){
  function doAdd(){
    return number1 + number2;
  }
  return doAdd();
}
add(1,1) // 2
```

- A closure has access to outer variables.

- Often callbacks are declared as closures.

Closures and Variables

If outer variables change, your closure variables will change too.

```
function trickyAdd(number1, number2){
  function doAdd(){
    return number1 + number2;
  }
  number1 += 1;
  number2 += 2;
  return doAdd();
}
trickyAdd(1,1) // 5
```

Private Scope with Closures

- Create a function that creates an object.
- Create inner functions (closures).
- Create the object that uses the defined closures.

```
function createObject(){
  var value=0;
  function sum(a,b){return a+b;};
  return {
    add2:function(val){return sum(2,val);},
    add10:function(val){return sum(10,val);},
    increment:function(val){value+=val;},
    getValue:function(){return value;}
} }
```

Private Scope with Closures

```
var object = createObject();
object.value; // undefined
object.getValue(); //0
object.increment(5);
object.getValue(); // 5
object.add2(5); // 7
object.add10(10); //20
```

Returning Functions

Functions can be used to create other functions.

```
function createIncrementByNumber(number){
  return function(x) { return number+x; }
}

var inc = createIncrementByNumber(2);
inc(3) // 5
inc(10) //12

var inc2 = createIncrementByNumber(10);
inc2(3) // 13
inc2(10) // 20
```

Topics

- JavaScript Functions
  - Functions as Values
  - Functions and Variable Scopes
- Prototypes
- Delays and Intervals

JavaScript Prototypes

- All objects in JavaScript have a prototype property.
- You can assign an object to a prototype.
- You can chain prototypes.

Prototype Functions

You can add functions to an object by using:

```
function A(){
  this.getName = function(){ return "A"; }
};
```

With prototypes, use the following syntax:

```
function A(){};
A.prototype.getName=function(){return "A";};
```

Prototype Dynamic Object Modification

Adding properties and methods to the prototype will affect all instances.

```
function A(){};
var inst1 = new A();
var inst2 = new A();
A.prototype.getName=function(){return "A";};
console.log(inst1.getName()); // A
console.log(inst2.getName()); // A
```

Prototype Chain

```
function A(){};
A.prototype.getName = function(){
  return "proto A"
};
function B(){};
B.prototype = new A();
var b = new B();
console.log(b.getName()); // proto A
```

Topics

- JavaScript Functions
  - Functions as Values
  - Functions and Variable Scopes
  - Modules
- Prototypes
- Delays and Intervals
- HTML5 Canvas

Delayed Functions

- To execute a function after a set period of time, use:

```
timeoutId = setTimeout(function, delay)
```

- Execute the function after delay milliseconds.
- Note that setTimeout is not a blocking call.

```
setTimeout(
  function(){console.log("TIMEOUT!");}
,1000);
console.log("After timeout!");
```

- This will print “After timeout!” and a second later, it will print “TIMEOUT!”

Interval Functions

To execute a function periodically in fixed intervals, use:

```
intervalId = setInterval(function, delay);
```

This will execute the function every `delay` milliseconds until it is canceled.

`setInterval` is not a blocking call.

```
setInterval(
  function(){console.log("INTERVAL!");}
,1000);
console.log("After interval!");
```

This will print “After interval!” and then every second thereafter it will print “INTERVAL!”

Canceling Intervals and Timeouts

- You can cancel intervals and timeouts.
- Both `setInterval` and `setTimeout` return an ID that you can use.

```
clearTimeout(timeoutId);
```

```
clearInterval(intervalId);
```

Summary

In this lesson, you should have learned how to:

- Define functions and use them as values
- Create closures with the appropriate variable scopes
- Create JavaScript timers and delays to create animations in HTML