Closures in JavaScript

Al Mazzoli

What is a closure?

- Function combined with local variables in the environment in which the function was created
- Created every time a function is created
- Example
  - add5 and add10 are closures
    - Have the same function body definition
    - But different lexical environments
      - For add5, x is 5
      - For add10, x is 10

```
function makeAdder(x) {
  return function(y) {
    return x + y;
  };
}

var add5 = makeAdder(5);
var add10 = makeAdder(10);

console.log(add5(2));  // 7
console.log(add10(2)); // 12
```

Another example

```
function makeSizer(size) {
  return function() {
    document.body.style.fontSize = size + 'px';
  };
}

var size12 = makeSizer(12);
var size14 = makeSizer(14);
var size16 = makeSizer(16);
```

- Event-based front-end code
  - Define a behavior
  - Attach behavior to user event
  - Code is attached as a callback - single function triggered by event
- https://jsfiddle.net/vnkuZ/7726/

Why is a closure?

- Lets you associate data with a function that acts on that data
  - Similar to OOP, where you can associate objects with methods
  - Can use a closure where you might use an object with a single method
    - Again, helpful in event-based development
- Can emulate private methods
  - (Methods that can only be called by other methods in the same class)
  - Way to manage global namespace
- Data hiding and encapsulation

Another example

- One lexical environment shared by three closures
- Anon function executed as soon as its defined
- privateCounter and changeBy are both private
  - Can only access through public functions
    - Value
    - Increment
    - Decrement
- Counters are independent of each other

```
var makeCounter = function() {
  var privateCounter = 0;
  function changeBy(val) {
    privateCounter += val;
  }
  return {
    increment: function() {
      changeBy(1);
    },

    decrement: function() {
      changeBy(-1);
    },

    value: function() {
      return privateCounter;
    }
  }
};

var counter1 = makeCounter();
var counter2 = makeCounter();

alert(counter1.value());  // 0.

counter1.increment();
counter1.increment();
alert(counter1.value()); // 2.

counter1.decrement();
alert(counter1.value()); // 1.
alert(counter2.value()); // 0.
```

Closures and Scope

```
// global scope
var e = 10;
function sum(a){
  return function sum2(b){
    return function sum3(c){
      // outer functions scope
      return function sum4(d){
        // local scope
        return a + b + c + d + e;
      }
    }
  }
}

var sum2 = sum(1);
var sum3 = sum2(2);
var sum4 = sum3(3);
var result = sum4(4);
console.log(result) //log 20
```

- Each closure has 3 scopes
  - Local scope
  - Outer function's scope
  - Global scope
- If the outer function is nested, the outer function's scope includes ITS outer function

Tips and tricks

- Don’t use closures unless you need to
  - Takes space, makes things slower
- Try not to put closures in loops
  - They will not do what you want

Sources

- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Closures