JavaScript Closures

Closures

Functions can "look" outwards

```
var a = 10;

function print() {
  console.log(a);
}

print();
```

a is in this case a "free variable" for the **print** function

Closures

```
var a = 1;

function print(c) {
  var b = 2;

  function actuallyPrint(d) {
    var e = 5;
    console.log(a, b, c, d, e);
  }

  actuallyPrint(4);
}

print(3);
```

We can have closures in closures

(nested functions)

Closures

```
var a = 10;

function makePrinter() {
  var b = 20;

  function actuallyPrint() {
    console.log(a, b);
  }

  return actuallyPrint;
}

var print = makePrinter();
print();
```

We're using a function to create a closure context for the function we're returning

Closures

```
var a = 10;

function makePrinter() {
  var b = 20;

  return function () {
    console.log(a, b);
  }
}

var print = makePrinter();
print();
```

We can also return a function directly

Closures

Used to create private members

```
function makeCounter() {
  var count = 0;

  return function getNext() {
    return count++;
  }
}

var counter = makeCounter();
counter(); // 0
counter(); // 1
counter(); // 2
```

Factory function

Closures

```
function makeCounter() {
  var count = 0;

  function increment() {
    count++;
  }

  function decrement() {
    count--;
  }

  function getValue() {
    return count;
  }

  return {
    increment: increment,
    decrement: decrement,
    getValue: getValue,
  };
}

var counter = makeCounter();
counter.getValue(); // => 0
counter.increment();
counter.increment();
counter.getValue(); // => 2
```

Closures

```
function makeCounter() {
  var count = 0;

  return {
    increment: function () {
      count++;
    },
    decrement: function () {
      count--;
    },
    getValue: function () {
      return count;
    }
  };
}
```