Discussion 8

10/30 and 11/1

OOP - Revision to Rectangles

```
3  function rectangle(x, y, width, height) {
4    function getX() { return x; }
5    function getY() { return y; }
6    function setX(newX) { x = newX; };
7    function setY(newY) { y = newY; };
8    return {
9      getX: getX,
10     getY: getY,
11     getWidth: function() { return width; },
12     getHeight: function() { return height; },
13     moveBy: function(dx,dy) { setX(getX()+dx); setY(getY()+dy); }
14   };
15 }
16
17 let r = rectangle(2, 10, 100, 200);
18 r.moveBy(5, 8);
19 console.log(r.getX());
```

Conversion of Rectangle to Class

```
class Rectangle {
  constructor(x, y, width, height) {
   this.x = x;
   this.y = y;
   this.width = width;
   this.theta = theta;
  }
  getX() { return this.x};
  getY() { return this.y};
  setX(newX) { this.x = newX; }
  setY(newY) { this.y = newY; }
  moveBy(dx, dy) {this.x = this.x + dx; this.y = this.y + dy;}
}
```

Duck Typing: Is there a difference between them?

Exercise 1

Creates classes C1 and C2 such that

```
let a1 = [new C1, new C1];
let a2 = [new C2, new C2];
let a3 = [new C1, new C2, new C1, new C2];
let f = function(x, y) { return x + y.value(); };
console.log(a1.reduce(f, 0));
console.log(a2.reduce(f, 0));
console.log(a3.reduce(f, 0));
```

Outputs:

2

6

8

Exercise 1

```
let a1 = [new C1, new C1];
let a2 = [new C2, new C2];
let a3 = [new C1, new C2, new C1, new C2];
let f = function(x, y) { return x + y.value(); };
console.log(a1.reduce(f, 0));
console.log(a2.reduce(f, 0));
console.log(a3.reduce(f, 0));
```

```
class C1 {
  value() {
    return 1;
  }
}

class C2 {
  value() {
    return 3;
  }
}
```

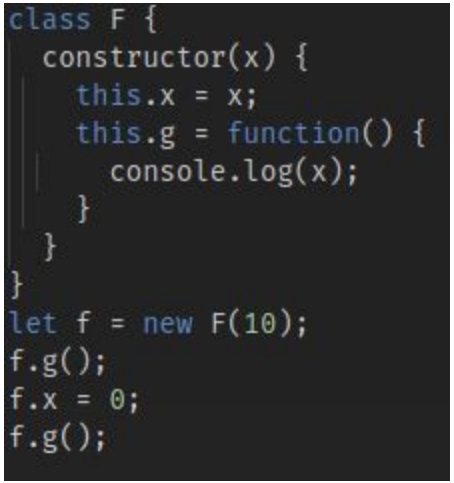
Exercise 2

What will the output be?

```
class F {
  constructor(x) {
    this.x = x;
    this.g = function() {
      console.log(x);
    }
  }
}
let f = new F(10);
f.g();
f.x = 0;
f.g();
```

Exercise 2

What will the output be?



```
class F {
  constructor(x) {
    this.x = x;
    this.g = function() {
      console.log(x);
    }
  }
}
let f = new F(10);
f.g();
f.x = 0;
f.g();
```

10

10

Exercise 3

What does this output?

```
class MyClass {
  constructor() {
    this.x = 0;
  }
  foo() {
    return {
      x: 0,
      bar() {
        this.x = 42;
      }
    };
  }
}
```

```
let c = new MyClass();
let f = c.foo();
console.log(c.x);
f.bar();
console.log(c.x);
```

Exercise 3

What does this output?

```
class MyClass {
  constructor() {
    this.x = 0;
  }
  foo() {
    return {
      x: 0,
      bar() {
        this.x = 42;
      }
    };
  }
}
```

```
let c = new MyClass();
let f = c.foo();
console.log(c.x);
f.bar();
console.log(c.x);
```

0

0

Streams

Streams

```
function memo0(f) {
  let r = { kind: "unknown" };
  return {
    get: function() {
      if (r.kind === "unknown") {
        r = { kind: "known", v: f() }
      }
      return r.v;
    },
    toString: function() {
      if (r.kind === "unknown") {
        return "<unevaluated>";
      } else {
        return r.v.toString();
      }
    }
```

```
function sone(x) {
  return snode(x, memo0(() => sempty));
}

// sappend(left: Stream<A>, right: () => Stream<A>): Stream<A>
function sappend(left, right) {
  if (left === sempty) {
    return right();
  } else {
    return snode(left.head(), memo0(() => sappend(left.tail(), right)));
  }
}
```

```
function snode(head, tail) {
  return {
    kind: 'snode',
    head: function() {return head;},
    tail: function() {return tail.get();},
    toString: function() {
      return "snode(" + head.toString() + ", " + tail.toString() + ")";
    }
  }
}
```

```
let sempty = {
  kind: "sempty",
  toString: () => 'sempty'
}
```

Binary Tree to Stream

```
function leaf(v) {
  return { kind: "leaf", value: v };
}

function node(l, r) {
  return { kind: "node", left: l, right: r};
}
```

How can we convert to stream?

treeStream (or fringeStream in the notes)

```
function treeStream(t) {
  if (t.kind == "leaf") { return sone(t.value); }
  else { return sappend(treeStream(t.left), () => treeStream(t.right)); }
}
```

Linked List to Stream

Code for linked list

```
function llnode(h, t) {
  return { kind: node, head: h, tail: t };
}

function llempty() {
  return { kind: 'empty' };
}
```

How can we convert this to a stream?

llStream

```
function llStream(ll) {
  if (ll.kind === "empty") { return sempty; }
  else { return sappend(sone(ll.head), () => llStream(ll.tail));}
}
```