Jerry Cain
CS 106AJ

Handout #19S
October 17-18, 2018

Solution to Section #3

Portions of this handout by Eric Roberts and Patrick Young.

Solution 1: Random circles

```
/*
 * File: RandomCircles.js
 * ----------------------
 * This program draws a set of 10 circles with different sizes,
 * positions, and colors.  Each circle has a randomly chosen
 * color, a randomly chosen radius between 5 and 50 pixels,
 * and a randomly chosen position on the canvas, subject to
 * the condition that the entire circle must fit inside the
 * canvas without extending past the edge.
 */

/* Constants */

const NCIRCLES = 10;
const MIN_DIAMETER = 10;
const MAX_DIAMETER = 100;
const GWINDOW_WIDTH = 500;
const GWINDOW_HEIGHT = 300;

/* Main function */

function main() {
    let gw = GWindow(GWINDOW_WIDTH, GWINDOW_HEIGHT);
    for (let i = 0; i < NCIRCLES; i++) {
       let d = randomInteger(MIN_DIAMETER, MAX_DIAMETER);
       let x = randomInteger(0, gw.getWidth() - d);
       let y = randomInteger(0, gw.getHeight() - d);
       let circle = new GOval(x, y, d, d);
       circle.setFilled(true);
       circle.setColor(randomColor());
       gw.add(circle);
    }
}
```

Note: on some runs of the program you might not *see* 10 circles because some circles will be drawn white or be drawn on top of previously drawn circles, potentially blocking them entirely from view.

Solution 2: The Seeker and the Snitch

```
/*
 * File: CatchTheSnitch.js
 * -----------------------
 * This program draws a snitch (a yellow GOval) on the screen
 * that will jump to random locations at set intervals. Your job
 * as the seeker is to click on the snitch to catch it and win
 * the game.
 */

/* Constants */
const GWINDOW_WIDTH = 500;
const GWINDOW_HEIGHT = 300;
const TIME_STEP = 1000;
const SNITCH_DIAMETER = 25;

/* Main program */
function main() {
    let gw = GWindow(GWINDOW_WIDTH, GWINDOW_HEIGHT);
    let snitch = initializeSnitch(gw);

    /* Sets the click action handler to allow the program
     * to terminate when the user clicks on the snitch */
    let clickAction = function(e) {
       let obj = gw.getElementAt(e.getX(), e.getY());
       if (obj === snitch) {
          console.log("You win!");
          clearInterval(timer);
          gw.remove(snitch);
       }
    };
    gw.addEventListener("click", clickAction);
    let step = function() {
       snitch.setLocation(getRandomX(), getRandomY());
    };
    let timer = setInterval(step, TIME_STEP);
}

/* Initializes the snitch and adds it to the GWindow */
function initializeSnitch(gw) {
    let snitch = GOval(getRandomX(), getRandomY(),
                       SNITCH_DIAMETER, SNITCH_DIAMETER);
    snitch.setFilled(true);
    snitch.setColor("Yellow");
    gw.add(snitch);
    return snitch;
}

/* Returns a random, valid x or y location for the snitch. */
function getRandomX() {
    return randomInteger(0, GWINDOW_WIDTH - SNITCH_DIAMETER);
}
function getRandomY() {
    return randomInteger(0, GWINDOW_HEIGHT - SNITCH_DIAMETER);
}
```

Solution 3: Tracing function execution

The output of **`CalculateBill.js`** is:

```
Your total before tip is: $100.
Your final price is: $106.
```

For this problem, there are several concepts that we should understand.

(1) Parameters

Look at the following two lines:

```
let finalPrice = calculateBill(numSalads, numPizzas);
function calculateBill(numPizzas, numSalads) { ... }
```

Here, note that we actually swap **`numPizzas`** and **`numSalads`** when passing them as parameters, so inside **`calculateBill`**, **`numPizzas`** should be 4 and **`numSalads`** should be 6.

There is no connection between the names given to variables in one function, and the names of the parameters which those variables are assigned to during a function call. **When passing a variable as an argument to a function, JavaScript assigns the first argument to the first parameter, the second argument to the second parameter, and so forth, regardless of the names they are given.** Giving confusing names to parameters though, while certainly possible, constitutes bad style and should be avoided.

(2) Closure

Remembering that when you define an inner function within an existing function, we call that a closure, and **closures can access the variables of the outer function**. This means:

- In **`calculateBill`**, **`addSaladCosts`** and **`addPizzaCosts`** can both access and modify **`total`** without passing **`total`** in as a parameter.
- However, **`addTax`** and **`addTip`** both require passing **`total`** in as a parameter because they are defined outside of **`calculateBill`**.
- Finally, note that even though **`total`** is changed in **`addTax`**, it will not affect the value of **`total`** in **`calculateBill`**.