Eric Roberts and Jerry Cain
CS 106J

Handout #14S
April 19, 2017

Solutions for Section #2

1. Bug Squash

```
function isEricRoberts(name) {
      if (name = "Eric Roberts"){
            return true;
      }
      return false;
}
isEricRoberts("Jerry Cain"); ➔ should return false

BUG: The = operator is used for assignment. The === is the equality
operator we meant to use here. When we mix these up, name gets assigned
to "Eric Roberts" and by default this will always return true.
```

```
function climbHooverTower() {
      var currentFloor = 1;
      while (currentFloor <= 12) {
            console.log("On floor " + currentFloor + ". ");
            currentFloor + 1;
      }
}
climbHooverTower(); ➔ should output "On floor 1. / ... / On floor 12."
Note: the / represents a line break, used here just to save space.


BUG: We forget to assign currentFloor back to itself when incrementing
it. Fixes include currentFloor++ or currentFloor += 1. In this buggy
code, currentFloor remains at 1, which means currentFloor never reaches
12. As a result, we end up with an infinite loop. Note that if your
programs hangs or freezes when you run it, an infinite loop is a likely
culprit.
```

```
function countToTen() {
      var counts = "";
      for (var i = 1; i < 10; i++) {
            counts += i + " ";
      }
      console.log(counts);
}
countToTen(); ➔ should output "1 2 3 4 5 6 7 8 9 10 "


BUG: Programmers often mix up the comparison operators < and <=. In
this case, since we actually want to count to 10, we want to use <=.
Always be careful with these and make sure you are coding what you are
thinking.
```

2. Squares

```
/*
 * File: Squares.js
 * --------------------
 * This program draws two squares. One is centered in a Gwindow
 * and the other is printed out as a string in the console. We make
 * sure to define constants instead of using magic numbers.
 */
import "graphics";

/* Constants for the GWindow and the square's dimensions */
const GWINDOW_WIDTH = 500;
const GWINDOW_HEIGHT = 200;
const SQUARE_LENGTH = 15;

/* Creates a GWindow and adds a filled square centered in the window */
function graphicalSquare() {
      var gw = GWindow(GWINDOW_WIDTH, GWINDOW_HEIGHT);
      var x_center = gw.getWidth() / 2;
      var y_center = gw.getHeight() / 2;
      gw.add(createCenteredSquare(x_center, y_center));
}

/* Creates a square centered at the location x_center and y_center.
 * Remember that the origin of GObjects is in the top left corner.
 */
function createCenteredSquare(x_center, y_center) {
      var offset = SQUARE_LENGTH / 2;
      var square = GRect(x_center - offset, y_center - offset,
                         SQUARE_LENGTH, SQUARE_LENGTH);
      square.setFilled(true);
      return square;
}

/* Creates a single string representing a square by concatenating
 * SQUARE_LENGTH rows with SQUARE_LENGTH x's in each row. */
function asciiSquare() {
      var square = "";
      for (var i = 0; i < SQUARE_LENGTH; i++) {
            var row = "";
            for (var j = 0; j < SQUARE_LENGTH; j++) {
                  row += "x";
            }
            square += row + "\n";
```

```
      }
      return square;
}

/* Creates and outputs both the graphical square and ASCII square */
function main() {
      graphicalSquare();
      console.log(asciiSquare());
}
```

3. Drawing a face

```
/*
 * File: RobotFace.js
 * --------------------
 * This program draws a robot face using GRects and GOvals.
 * We make sure to define constants at the top of our program instead
 * of using magic numbers. We also write the program in terms of
 * reusable and general methods drawRectangle and drawCircle.
 */
import "graphics";

/* Constants for the drawing */
const HEAD_WIDTH = 150;
const HEAD_HEIGHT = 250;
const EYE_RADIUS = 10;
const MOUTH_WIDTH = 60;
const MOUTH_HEIGHT = 20;
const GWINDOW_WIDTH = 500;
const GWINDOW_HEIGHT = 200;

/* Draw the entire face centered in the graphics window */
function main(){
   var gw = GWindow(GWINDOW_WIDTH, GWINDOW_HEIGHT);
   var cx = gw.getWidth()/2;
   var cy = gw.getHeight()/2;
   addHead(cx - HEAD_WIDTH/2, cy - HEAD_HEIGHT/2, gw);
   addEye(cx - HEAD_WIDTH/4, cy - HEAD_HEIGHT/4, gw);
   addEye(cx + HEAD_WIDTH/4, cy - HEAD_HEIGHT/4, gw);
   addMouth(cx - MOUTH_WIDTH/2, cy + HEAD_HEIGHT/4, gw);
}

/*
 * Add a head with top left at position x,y. Adding a head consists
 * of drawing a rectangle with the given width, height, and color.
 */
function addHead(x, y, gw){
   drawRectangle(x, y, HEAD_WIDTH, HEAD_HEIGHT, "Grey", gw);
}

/*
 * Add an eye centered at cx, cy. Adding an eye consists of drawing
 * a circle with the given radius and color.
 */
```

```
function addEye(cx, cy, gw){
   drawCircle(cx, cy, EYE_RADIUS, "Yellow", gw);
}

/*
 * Add a mouth with top left at x,y. Adding a mouth consists of
 * drawing a rectangle with given width, height and color.
 */
function addMouth(x, y, gw){
   drawRectangle(x,y, MOUTH_WIDTH, MOUTH_HEIGHT, "White", gw);
}

/*
 * This function draws a general rectangle with its top left
 * at position x,y with a specified width, height and color.
 */
function drawRectangle(x, y, width, height, color, gw){
   var rect = GRect(x, y, width, height);
   rect.setFilled(true);
   rect.setColor(color);
   gw.add(rect);
}

/*
 * This method draws a general circle centered at (cx,cy),
 * with a given radius r and a Color c.
 */
function drawCircle(cx, cy, r, color, gw){
   var x = cx - r;
   var y = cy - r;
   var circle = GOval(2 * r, 2 * r);
   circle.setColor(color);
   circle.setFilled(true);
   gw.add(circle, x, y);
}
```

Style Focus for Section 2:

Always Use Constants: Our code should never contain "magic numbers," meaning numbers we use in our code that don't have a clear meaning. For example don't just have "7," say **`DAYS_IN_WEEK`**. Instead of "10," we write **`EYE_RADIUS`**. Well-named constants make it clear what the purpose of the variable is, and also reduce errors. If someone wants to change the **`EYE_RADIUS`**, they can modify its value everywhere in the program by only changing it once. If we just wrote "10," they would have to go searching through the code to find all the places we use this value. The only numbers we don't need to turn into constants are the numbers 0, 1 and sometimes 2.

General and Reusable Functions: It is important to write methods that are general and reusable. If you find yourself copying and pasting code, this is probably a sign that you should have a more general method to accomplish this task. However, figuring out how to write general and reusable functions is an art, and is quite challenging. Look for similarities in your code, or ask yourself how you can use parameters.