Eric Roberts and Jerry Cain
CS 106J

Handout #13
April 17, 2017

Assignment #2—Simple JavaScript Programs

Due: Monday, April 24

Your job in this assignment is to write programs to solve each of these problems.

Problem 1 (Chapter 2, exercise 6, page 75)

It is a beautiful thing, the destruction of words.
—Syme in George Orwell's *1984*

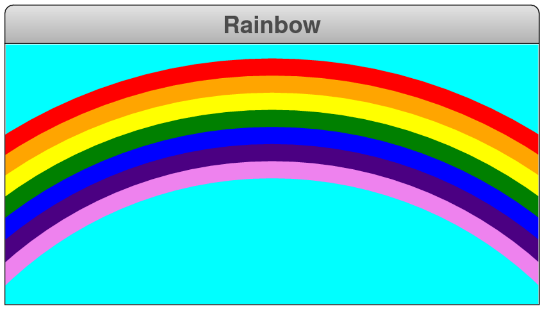
In Orwell's novel, Syme and his colleagues at the Ministry of Truth are engaged in simplifying English into a more regular language called *Newspeak.* As Orwell describes in his appendix entitled "The Principles of Newspeak," words can take a variety of prefixes to eliminate the need for the massive number of words we have in English. For example, Orwell writes

> Any word—this again applied in principle to every word in the language—could be negatived by adding the affix *un-*, or could be strengthened by the affix *plus-*, or, for still greater emphasis, *doubleplus-*. Thus, for example, *uncold* meant "warm," while *pluscold* and *doublepluscold* meant, respectively, "very cold" and "superlatively cold."

Create a file called `Newspeak.js` that defines three functions—`negate`, `intensify`, and `reinforce`—that take a string and add the prefixes `"un"`, `"plus"`, and `"double"` to that string, respectively. As an example, calling `reinforce(intensify(negate("bad")))` returns `"doubleplusungood"`.

Problem 2 (Chapter 2, exercise 10, page 76)

Use the `GObject` hierarchy to draw a rainbow that looks something like this:



Starting at the top, the seven bands in the rainbow are red, orange, yellow, green, blue, indigo, and violet, respectively; cyan makes a lovely color for the sky. Remember that this chapter defines only the `GRect`, `GOval`, and `GLine` classes and does not include a graphical object that represents an arc. It will help to think outside the box, in a more literal sense than usual.

Rather than specify the exact dimensions of each circle (and there are indeed circles here), play around with their sizes and positioning until you get something that matches your aesthetic sensibilities. The only things we'll be concerned about are:

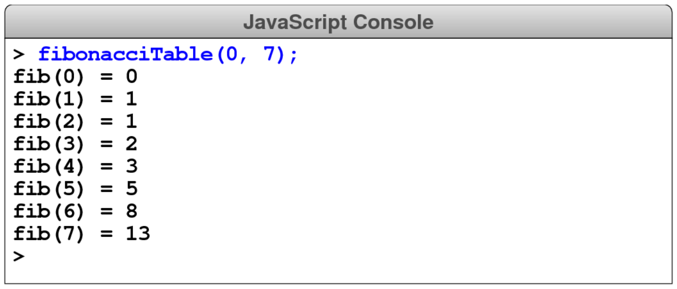
- The top of the arc should not be off the screen.
- Each of the arcs in the rainbow should get clipped along the sides of the window, and not along the bottom.

Problem 3 (adapted from Chapter 3, exercise 10, page 110)

In mathematics, there is a famous sequence of numbers called the ***Fibonacci sequence*** after the thirteenth-century Italian mathematician Leonardo Fibonacci. The first two terms in this sequence are 0 and 1, and every subsequent term is the sum of the preceding two. Thus the first several terms in the Fibonacci sequence are as follows:

$$
\begin{aligned}
F_0 &= 0 \\
F_1 &= 1 \\
F_2 &= 1 \quad (0 + 1) \\
F_3 &= 2 \quad (1 + 1) \\
F_4 &= 3 \quad (1 + 2) \\
F_5 &= 5 \quad (2 + 3) \\
F_6 &= 8 \quad (3 + 5) \\
F_7 &= 13 \quad (5 + 8)
\end{aligned}
$$

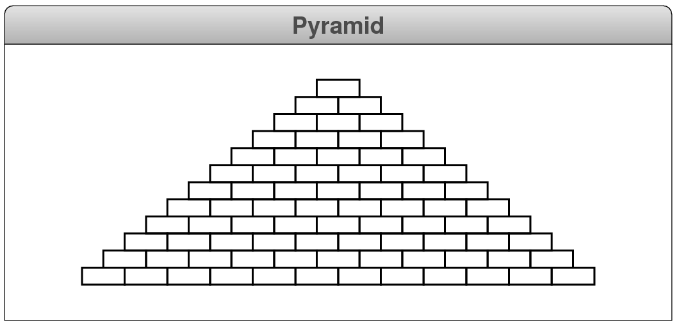
Write a function **`fib(n)`** that returns the **`n`**th Fibonacci number. Using the function **`factorialTable`** on page 94 as a model, write a function **`fibonacciTable(min, max)`** that uses **`console.log`** to display the terms of the Fibonacci sequence between the indices **`min`** and **`max`**. A sample run of your program might look like this:



```
> fibonacciTable(0, 7);
fib(0) = 0
fib(1) = 1
fib(2) = 1
fib(3) = 2
fib(4) = 3
fib(5) = 5
fib(6) = 8
fib(7) = 13
>
```

Problem 4 (Chapter 3, exercise 13, page 111)

Write a program that displays a pyramid on the graphics window. The pyramid consists of bricks arranged in horizontal rows, arranged so that the number of bricks in each row decreases by one as you move upward, as shown in the following sample run:



The pyramid should be centered in the window both horizontally and vertically. Your program should also use the following constants to make the program easier to change:

`BRICK_WIDTH`	The width of each brick
`BRICK_HEIGHT`	The height of each brick
`BRICKS_IN_BASE`	The number of bricks in the base

Problem 5

Eric’s “Once upon a time” story on holism and reductionism included a passage from Douglas Hofstadter’s Pulitzer-prize-winning book *Gödel, Escher, Bach*. Hofstadter’s book contains many interesting mathematical puzzles, many of which can be expressed in the form of computer programs. Of these, most require programming skills well beyond the second week of CS 106J. In Chapter XII, Hofstadter mentions a wonderful problem that is well within the scope of the control statements from Chapter 3. The problem can be expressed as follows:

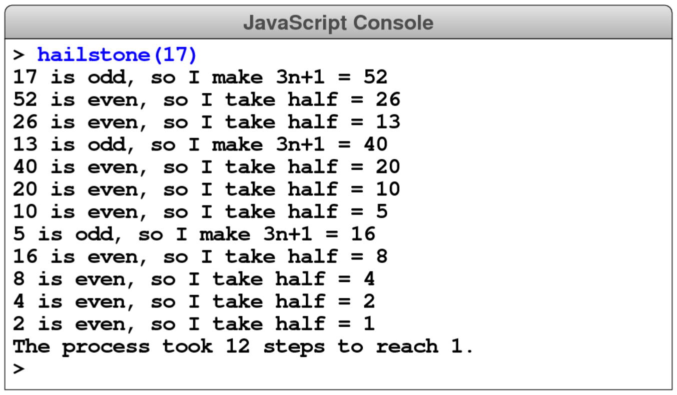
> Pick some positive integer and call it *n*.
> If *n* is even, divide it by two.
> If *n* is odd, multiply it by three and add one.
> Continue this process until *n* is equal to one.

On page 401 of the Vintage edition, Hofstadter illustrates this process with the following example, starting with the number 15:

15	is odd, so I make $3n+1$:	46
46	is even, so I take half:	23
23	is odd, so I make $3n+1$:	70
70	is even, so I take half:	35
35	is odd, so I make $3n+1$:	106
106	is even, so I take half:	53
53	is odd, so I make $3n+1$:	160
160	is even, so I take half:	80
80	is even, so I take half:	40
40	is even, so I take half:	20
20	is even, so I take half:	10
10	is even, so I take half:	5
5	is odd, so I make $3n+1$:	16
16	is even, so I take half:	8
8	is even, so I take half:	4
4	is even, so I take half:	2
2	is even, so I take half:	1

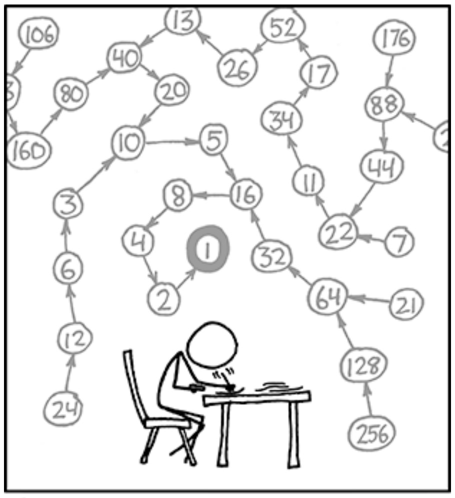
As you can see from this example, the numbers go up and down, but eventually—at least for all numbers that have ever been tried—comes down to end in 1. In some respects, this process is reminiscent of the formation of hailstones, which get carried upward by the winds over and over again before they finally descend to the ground. Because of this analogy, this sequence of numbers is usually called the ***Hailstone sequence,*** although it goes by many other names as well.

Write a function **`hailstone`** that takes an integer and then uses **`console.log`** to display the Hailstone sequence for that number, just as in Hofstadter's book, followed by a line showing the number of steps taken to reach 1. For example, your program should be able to produce a sample run that looks like this:



```
> hailstone(17)
17 is odd, so I make 3n+1 = 52
52 is even, so I take half = 26
26 is even, so I take half = 13
13 is odd, so I make 3n+1 = 40
40 is even, so I take half = 20
20 is even, so I take half = 10
10 is even, so I take half = 5
5 is odd, so I make 3n+1 = 16
16 is even, so I take half = 8
8 is even, so I take half = 4
4 is even, so I take half = 2
2 is even, so I take half = 1
The process took 12 steps to reach 1.
>
```

The fascinating thing about this problem is that no one has yet been able to prove that it always stops. The number of steps in the process can certainly get very large. How many steps, for example, does your program take when *n* is 27? The conjecture that this process always terminates is called the ***Collatz conjecture,*** and appears in the following *XKCD* cartoon by Randall Munroe:



THE COLLATZ CONJECTURE STATES THAT IF YOU PICK A NUMBER, AND IF IT'S EVEN DIVIDE IT BY TWO AND IF IT'S ODD MULTIPLY IT BY THREE AND ADD ONE, AND YOU REPEAT THIS PROCEDURE LONG ENOUGH, EVENTUALLY YOUR FRIENDS WILL STOP CALLING TO SEE IF YOU WANT TO HANG OUT.