I've provided you with a nearly, working implementation of Snake, a popular game found on many Nokia phones that is played by maneuvering the snake using the arrow keys. Unfortunately, the arrow keys don't quite work in my game yet. On top of that, the snake doesn't seem to be eating the apple when it passes over it.

Add the three snake files to a new directory called lab6 in your webpages directory. You will be working in and submitting snake.js

Part 1 (50 pts)

Finish the keydown event listener. You will notice that most of it is set up. The switch statement will be used to control what happens when specific keys are pressed. Since snake is meant to be controlled with the arrow keys, we only want to perform an action when an arrow key is pressed.

The switch statement is passed the property, event.key. Event listeners are always passed an event object that contains more specific information about that event.

The switch currently does one thing, it alerts the key that was pressed. Load the page in your browser and press some keys. You will see that each key has a specific value associated with it. Note the values for the arrow keys, as those will be the values for your cases.

Now that you have your cases, each case needs to do something different. When the left arrow is pressed, the snake should move left, etc. The game is already set up to move the snake in a specific direction specified by the variable *direction*. Direction has 4 possible values, 'down', 'up', 'left', 'right'.

Update direction to the appropriate value in each case in your switch and reload the page in your browser. You should see that the snake now responds to the arrow keys.

Part 2 (50 pts)

Now that we can control the snake, we can try to perform the objective of the game, which is to eat the ball in the middle of the play area. However, you will notice that right now, the snake seems to just go through the ball without eating it. Our game is missing the logic to detect if the snake has hit the ball or not.

If you notice, there are two constructors declared, each with their own set of properties and methods that help make the game work.

The first constructor is for an individual SnakePart. A SnakePart is made up of an HTML element, the part, and a compStyle object which represents the actual styles of the element (we want to use the computed style of the object instead of the style property because it contains the most up to date styles from the browser).

The second constructor represents the whole Snake. It's made up of an array of SnakeParts, and a length for the snake. For this assignment, the length is constant.

The Snake type also has a bunch of methods added to the prototype. One method is incomplete, isFoodEaten, which returns whether or not the head of the snake is at the same position as the food object. Complete this method, and our snake game will be complete!

A few notes:

- The position of each part is determined using the *top* and *left* style properties (Remember, we want to use the computed style, not the style property of the element to get the most up to date values).
- Top and left are returned as *strings* so we want to use a helper method, *parseInt()*, which takes a string and returns the first complete integer it can find in that string
    - For example: parseInt("20px") returns 20
- There is a variable called food declared at the top of the scope. For simplicity, this object is a SnakePart type so it also has a part and compStyle property.
- The head of the snake is always the first SnakePart in the list.