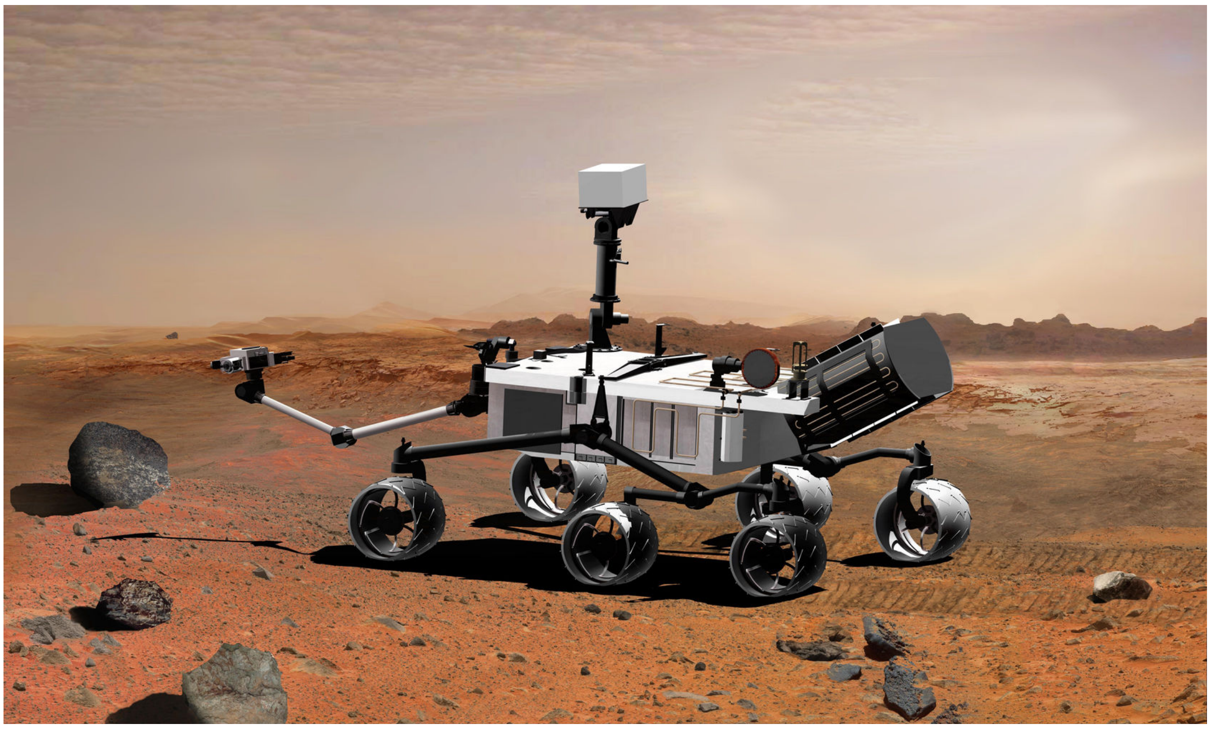
Exercise - Mars Rover Kata

We are sending a rover to Mars and we need to program its movements so that we can send it commands from Earth. At EOI, we've been tasked with developing the code for doing so, and we figured this would be a job for the junior developers.



Basic Information

Our Mars Rover is kind of dumb. By that, we mean it can't move and turn at the same time. This means that if the rover wants to move to the left, it's first move must be a turn. Its next move will then be a step forward.

We will discuss this concept in more detail as we progress through the exercise.

In addition, our rover is on a test mission. NASA has placed the rover on a 10x10 grid to make sure all is well before we ship it off to mars.

At a high level, what we will do in the exercise is the following:

• Create a function to turn the rover.

• Create a function to move the rover forwards or backwards based on its direction.

• Create a function to receive a list of commands and move based off of those commands.

Setup

In this exercise, we can use the Chrome Developer Console to take advantage of global functions and objects.

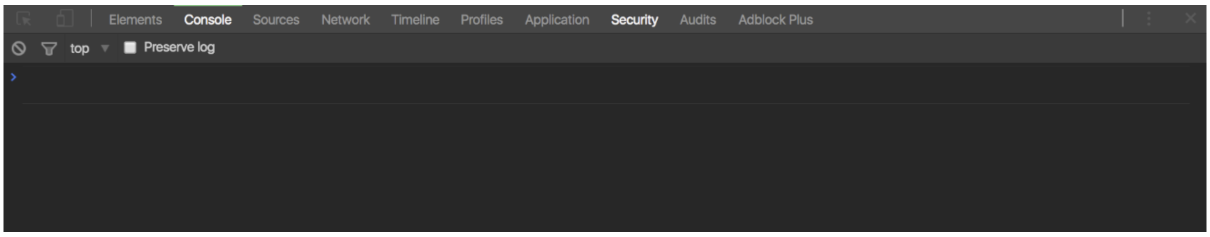
First, get the repo and unzip the files.

Open mars-rover-v2/javascripts/app.js in **SUBLIME.**

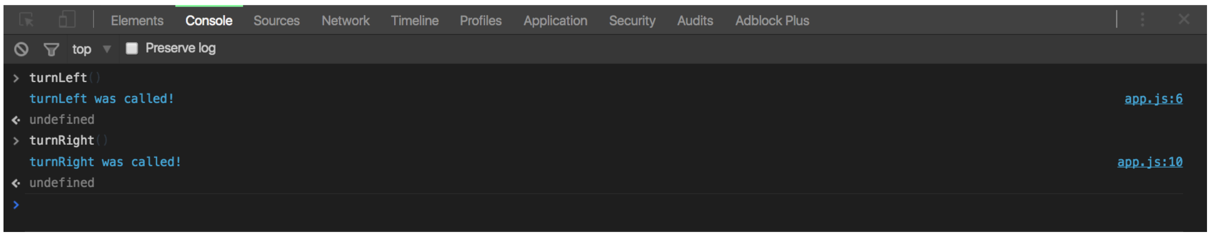
Then, open `mars-rover-v2/index.html` in Google Chrome. You can do this by navigating to the directory in your terminal and typing `open index.html`, or by dragging the HTML file to your browser.

With the HTML file open in Chrome, open your developer console. On Linux and Windows this is done with `ctrl + shift + j`. On Macs, this is done with `cmd + opt + j`.

After you have completed this, your page should look similar to this:



We can actually run the functions and access globally scoped variables using the console:



```
> turnLeft()
  turnLeft was called!                                   app.js:6
<· undefined
> turnRight()
  turnRight was called!                                  app.js:10
<· undefined
>
```

You're going to be using this to test out your rover and run the functions we will build in each step.

Iteration 1 | The Rover Object

Create an object to represent the rover. This object will have only one property for now: the `direction`.

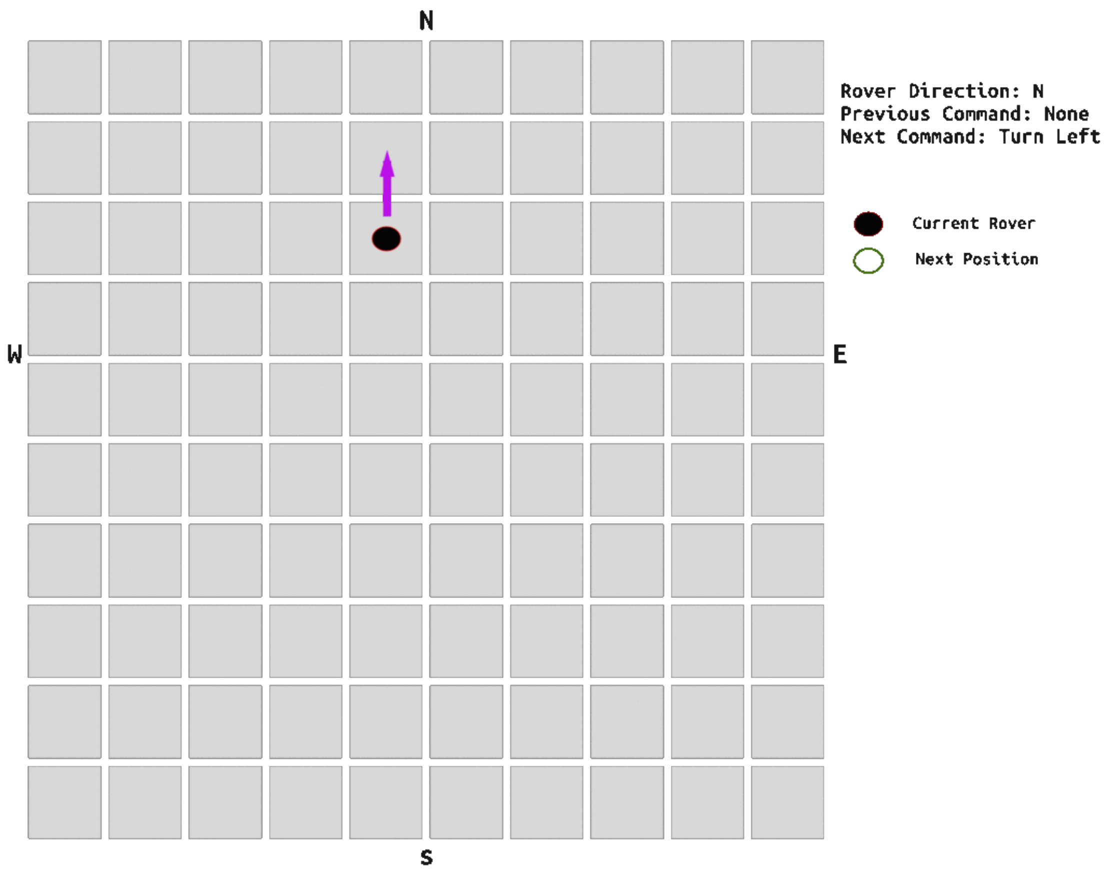
The direction property can contain one of four values: `"N"`, `"S"`, `"E"`, or `"W"`. The rover's default direction will be `"N"` (north).

Iteration 2 | Turning the Rover

The rover has a direction attribute. We've already provided functions called `turnLeft` and `turnRight` that receive a rover object as an argument. Your job is to turn the rover in the appropriate direction based off of its current direction.

Examples

- **Rover is facing North** and turns `left` => Rover is now facing **West**
- **Rover is facing West** and turns `left` => Rover is now facing **South**
- **Rover is facing North** and turns `right` => Rover is now facing **East**.



Suggested Approach

Begin with the a switch statement. Based off of which function is being called, turn the rover in the correct direction based off of its current direction.

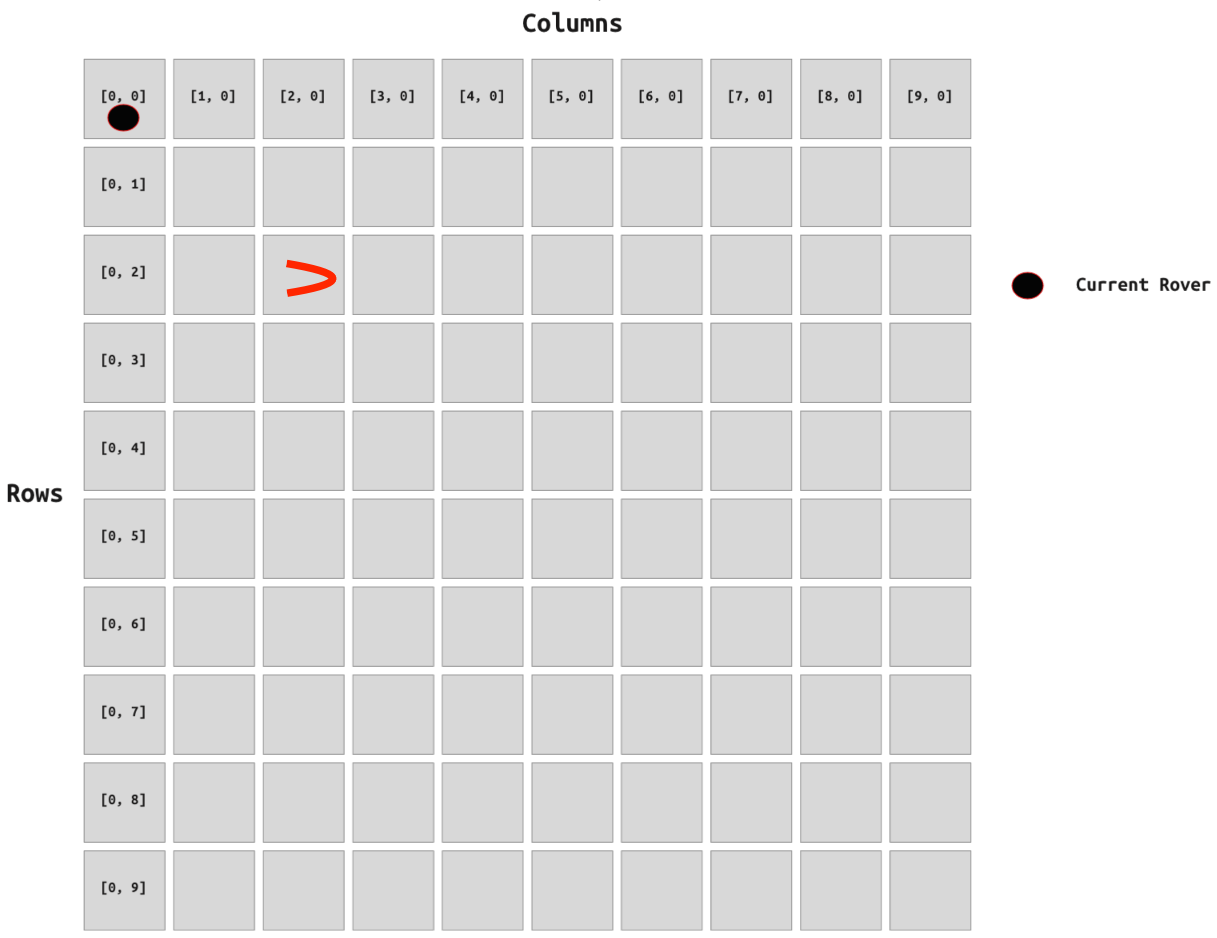
Test these functions using the Chrome Console to make sure they work. Remember, the `console.log()` function can be used to log anything. You can use this to see which direction your rover is currently facing.

Iteration 3 | Moving the Rover

The Rover Object's `position`

In order to move the rover around, we have to keep track of the rover's position.

Positions can be represented as a pair of coordinates, `x` and `y`. Add two properties to your rover called `x` and `y`. Their default values will both be 0.



Moving Forward

Once the rover has a position, it's time to move it.

How to Move

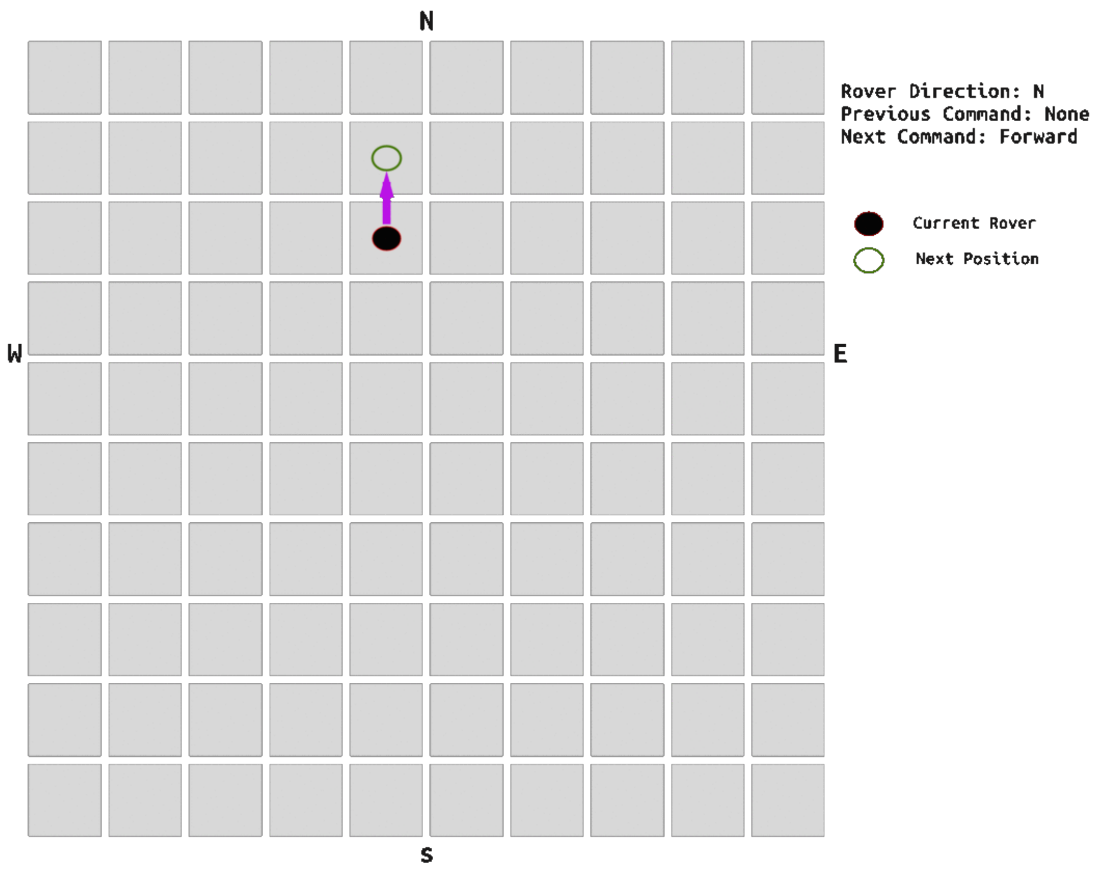
Moving forward is a function of the rover's current direction, and the movement forward.

For instance, if the Rover is facing **west** and moves forward, we would *decrease* the Rover's `x` by 1.

If the rover is facing **north** and moves forward, we would *decrease* the rover's `y` by 1.

If the rover is facing **south** and moves forward, we would *increase* the `y` by 1.

Fill in this logic in the `moveForward` function. After each movement, `console.log` the rover's coordinates so you can see where it is positioned.



Iteration 4 | Commands

Create a function that receives a list of commands. These commands will be the first letter of either `(f)orward`, `(r)ight`, or `(l)eft`.

To test it, use the string: `'rffrfflfrff'`.

Suggested Approach

Use a loop to iterate over the string. Inside of this loop, write a switch or if statement to call the correct function. For instance, if the letter is `f`, you're going to want to call the `goForward`function.

Iteration 5 | Tracking

We want to know where our rover has been. Create a property on the rover object that contains the coordinates of the places it has been. Call this property `travelLog`.

After each move, push the coordinates of the previous space to the `travelLog` array. After the rover has finished its moves, print out all of the spaces the rover has traveled over.

Bonus | Enforce Boundaries

At some point you may have accidentally run our rover off of the grid. If you recall, our grid is 10x10.

Make sure your rover doesn't accidentally roam off the map!

Bonus | Other Suggested Features

If you found the first few iterations of the exercise easy, try implementing the following features:

• **Moving Backwards**: Create another function called `moveBackward` that will move the rover back. This will be very similiar to the `moveForward` function.

• **Validate Inputs**: If we enter a letter into our inputs that isn't a rover command, nothing happens. For example `ffzzy` would simply move forward twice. Add validation so that the inputs must be `f`, `b`, `r`, or `l`.

The following require you to actually create a grid for the rover to move around on. In code, these grids are often represented as two dimensional arrays.

• **Obstacles** - Create obstacles for the rover. If the rover's next move would run it into an obstacle, stop it from moving forward and report the obstacle as found with `console.log`.

• **Other Rovers** - Add additional rovers to the map. Have them take turns moving. If one rover is going to run into the other, you should stop the rover and `console.log` a message saying so.

/Happy coding! ;)