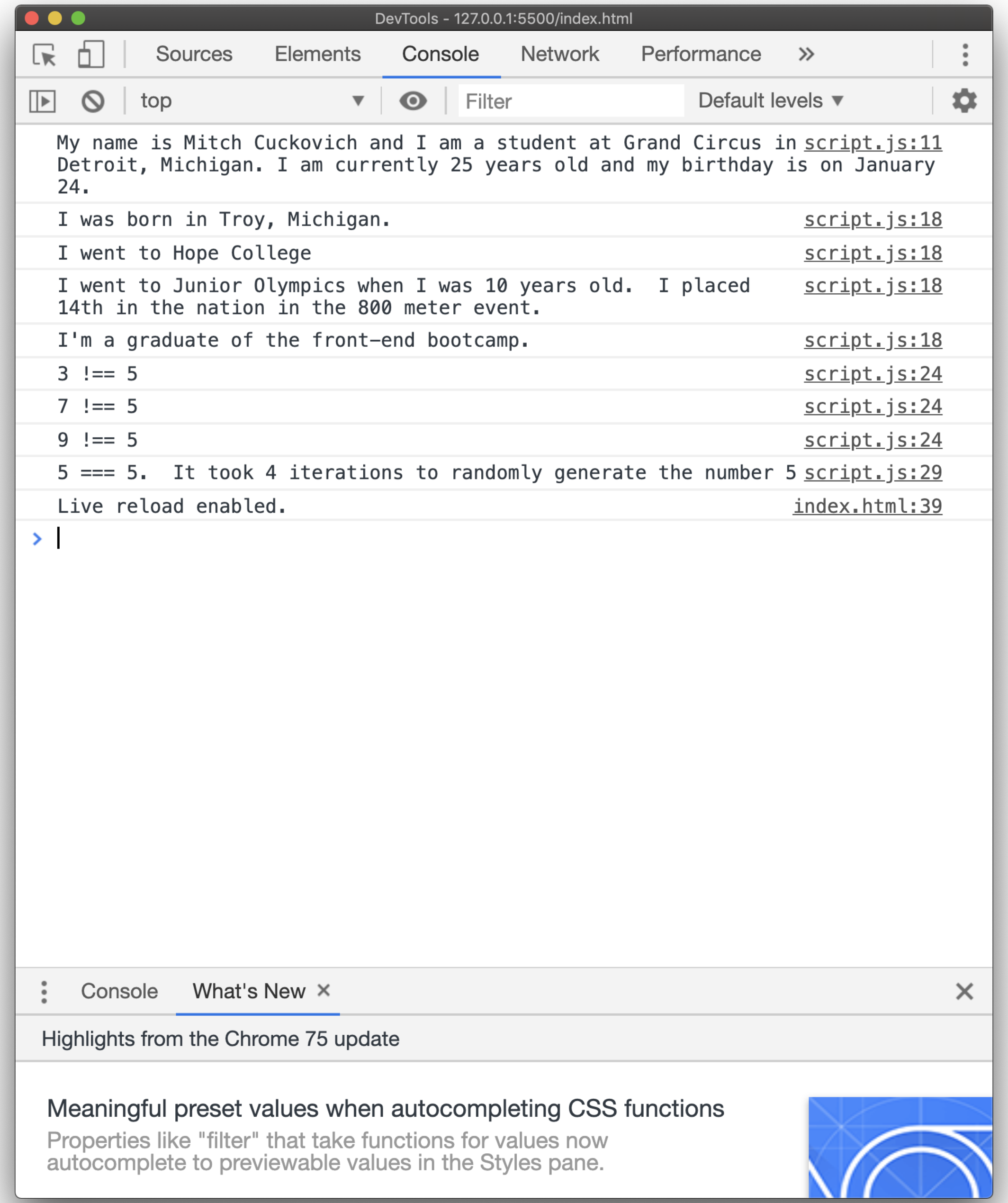
JAVASCRIPT LABS

JAVASCRIPT LAB 1 - VARIABLES, CONDITIONS & LOOPS

Task: This lab focuses on declaring and initializing variables, conditional statements, template literals, and constructing loops. The goal is to properly log statements to the console by using a mixture of the aforementioned topics. You will only need to construct an `index.html` and `script.js` file. Final output example:



```
My name is Mitch Cuckovich and I am a student at Grand Circus in  script.js:11
Detroit, Michigan. I am currently 25 years old and my birthday is on January
24.
I was born in Troy, Michigan.                                     script.js:18
I went to Hope College                                            script.js:18
I went to Junior Olympics when I was 10 years old.  I placed      script.js:18
14th in the nation in the 800 meter event.
I'm a graduate of the front-end bootcamp.                         script.js:18
3 !== 5                                                           script.js:24
7 !== 5                                                           script.js:24
9 !== 5                                                           script.js:24
5 === 5.  It took 4 iterations to randomly generate the number 5  script.js:29
Live reload enabled.                                             index.html:39
>
```

Meaningful preset values when autocompleting CSS functions
Properties like "filter" that take functions for values now autocomplete to previewable values in the Styles pane.

Build Specifications:

- Declare and initialize the following variables with appropriate values:
    - `name` (string) - Mitch Cuckovich
    - `age` (number) - 25
    - `birthday` (string) - January 24
    - `detroitGC` (boolean) - choose either true or false
    - `lifeEvents` (array with 4 items. 4 important life events)
        - "I was born in Troy, Michigan.",
        - "I went to Hope College",
        - "I went to Junior Olympics when I was 10 years old. I placed 14th in the nation in the 800 meter event.",
        - "I'm a graduate of the front-end bootcamp."
- Write an if/else statement that runs one of two console.log methods. Your console.log methods must incorporate the variables: `name`, `age`, and `birthday`.
    - If `detroitGC` is true, log the following message to the console:
        - My name is `name` and I am a student at Grand Circus in Detroit, Michigan. I am currently `age` years old and my birthday is on `birthday`.
    - else
        - My name is `name` and I am a student at Grand Circus in Grand Rapids, Michigan. I am currently `age` years old and my birthday is on `birthday`.
- Write a for loop that starts at 0 and iterates by increments of 1 while `i` is less than the length of the `lifeEvents` array. Each iteration of the loop should log a new sentence from the `lifeEvents` array. You should only have one console.log method.
- Declare and initialize a variable named `counter` to the value of 0.
- Write a while loop that loops while true.
    - Declare a variable named `randomNumber` that is initialized to a random integer between 1 and 10. Google search how to do this.
    - Write an if/else statement that has two conditions
        - If `randomNumber` is not equal to 5
            - Increment `counter`
            - Use a console.log method to say: "`randomNumber` !== 5"
        - Else
            - Increment `counter`
            - Use a console.log method to say: "5 === 5. It took `counter` iterations to randomly generate the number 5."
            - Break

Tests: Same as build specifications.

Extended Challenges:
Create an additional JavaScript file to try this...

Write a script that starts with two variables: hours and wage. Then write the code to log the total paycheck based on the hours worked and the wage ($ per hour). If the hours worked is over 40, give 150% pay for the extra hours worked. Try running it with a few different values for hours and wage. Here are some examples...

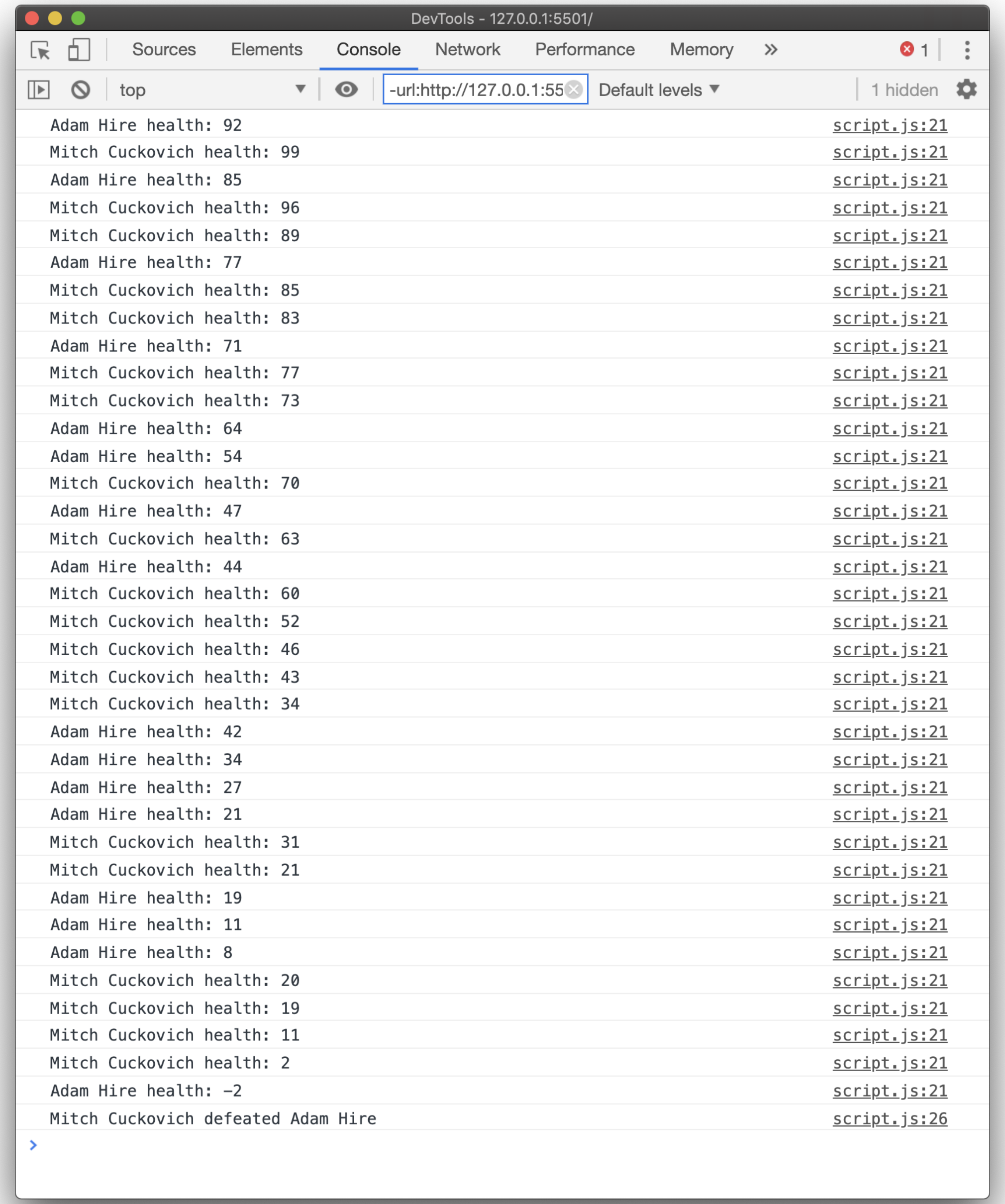
wage	hours	pay
10	20	200
10	40	400
10	50	550
12	60	840

Next, if this person got the same paycheck every week, count how many weeks it would take them to earn $1,000,000. Give the answer as a whole number. For example, with wage=10 and hours=50 it would take 1819 weeks.

JAVASCRIPT LAB 2 - GAME

Starter: This lab has starter code on GitHub Classroom. The starter includes tests you can run to automatically check some of the lab requirements.

Task: This lab will focus on three ways of writing out functions: function declaration, function expression, and arrow functions. The goal is to properly log statements to the console by using a mixture of the aforementioned topics. While this lab explicitly asks you to use certain functions, it is worth mentioning that each example `could` be written using any of the three methods for defining functions. You will only need to construct an `index.html` and `script.js` file. Final output example:



```
Adam Hire health: 92                                        script.js:21
Mitch Cuckovich health: 99                                  script.js:21
Adam Hire health: 85                                        script.js:21
Mitch Cuckovich health: 96                                  script.js:21
Mitch Cuckovich health: 89                                  script.js:21
Adam Hire health: 77                                        script.js:21
Mitch Cuckovich health: 85                                  script.js:21
Mitch Cuckovich health: 83                                  script.js:21
Adam Hire health: 71                                        script.js:21
Mitch Cuckovich health: 77                                  script.js:21
Mitch Cuckovich health: 73                                  script.js:21
Adam Hire health: 64                                        script.js:21
Adam Hire health: 54                                        script.js:21
Mitch Cuckovich health: 70                                  script.js:21
Adam Hire health: 47                                        script.js:21
Mitch Cuckovich health: 63                                  script.js:21
Adam Hire health: 44                                        script.js:21
Mitch Cuckovich health: 60                                  script.js:21
Mitch Cuckovich health: 52                                  script.js:21
Mitch Cuckovich health: 46                                  script.js:21
Mitch Cuckovich health: 43                                  script.js:21
Mitch Cuckovich health: 34                                  script.js:21
Adam Hire health: 42                                        script.js:21
Adam Hire health: 34                                        script.js:21
Adam Hire health: 27                                        script.js:21
Adam Hire health: 21                                        script.js:21
Mitch Cuckovich health: 31                                  script.js:21
Mitch Cuckovich health: 21                                  script.js:21
Adam Hire health: 19                                        script.js:21
Adam Hire health: 11                                        script.js:21
Adam Hire health: 8                                         script.js:21
Mitch Cuckovich health: 20                                  script.js:21
Mitch Cuckovich health: 19                                  script.js:21
Mitch Cuckovich health: 11                                  script.js:21
Mitch Cuckovich health: 2                                   script.js:21
Adam Hire health: -2                                        script.js:21
Mitch Cuckovich defeated Adam Hire                          script.js:26
>
```

Build Specifications:

- Declare an arrow function named `randomDamage` that has no parameters and returns a random integer between 1 and 10.
- Declare an arrow function named `chooseOption` that has two parameters named `opt1` and `opt2`. `chooseOption` does two things:
    - Declares and initializes a variable named `randNum` to either a 0 or 1, randomly.
    - Returns `opt1` if randNum is equal to 0 otherwise return `opt2` . (Use a ternary operator)
- Declare a function expression named `attackPlayer` that has one parameter named `health` which returns a number equal to `health` minus the result of the `randomDamage` function.
- Declare an arrow function named `logHealth` that has two parameters named `player` and `health` which has a console.log method to state the following message: "`player` health: `health`".
- Declare an arrow function named `logDeath` that has two parameters named `winner` and `loser` which has a console.log method to state the following message: "`winner` defeated `loser`"
- Declare an arrow function named `isDead` that has one parameter named `health` which returns a boolean value of true or false based on the following condition: `health` <= 0;
- Declare a function declaration named `fight` that has four parameters.
    - Parameters:
        - `player1` - this will hold the name of the first player
        - `player2` - this will hold the name of the second player
        - `player1Health` - this will hold the health of the first player
        - `player2Health` - this will hold the health of the second player
    - Within the `fight` function, write a while loop that loops while true
        - Declare and initialize a variable named `attacker` equal to the `chooseOption` function with `player1` and `player2` as arguments.
        - Has an if statement that is triggered when `attacker` is equal to `player1`.
            - Set `player2Health` equal to the result of `attackPlayer` with `player2Health` as its argument.
            - Calls the `logHealth` function with `player2` and `player2Health` as its arguments.
            - Call `isDead` with `player2Health` as an argument. If the result is true:
                - Call the `logDeath` function with `player1` and `player2` as arguments.
                - Break
        - Has an else statement that:
            - Sets `player1Health` equal to the `attackPlayer` function with `player1Health` as its argument.

        - Call the **`logHealth`** function with **`player1`** and **`player1Health`** as its arguments.
        - Call **`isDead`** with **`player1Health`** as an argument. If the result is true:
            - Call the **`logDeath`** function with **`player2`** and **`player1`** as arguments.
            - Break

- Lastly, call the **`fight`** function with the required four parameters. You pick the names and starting health. For example: player1: “Mitch”, player2: “Adam”, player1Health: 100, player2Health: 100.

Tests: Same as build specifications.

Extended Challenges:
Write some additional functions. Use whatever function style you like. Here are some ideas...

- printSquare: This function has a parameter for width. It logs a square shape to the console based on the width parameter. For example, given width 3, it would log:
  ```
  ###
  ###
  ###
  ```
- printTriangle: This function has a parameter for width. It logs a square shape to the console based on the width parameter. For example, given width 4, it would log:
  ```
  #
  ##
  ###
  ####
  ```
- Think of other shape methods you could write.
- getGrade: This function takes in a number parameter (0 to 100). It returns the corresponding letter grade using the scale: A=90+, B=80+, C=70+, D=60+, F=below 60. Call the function with different numbers and log the results. (Note: there should be no console.log *inside* the function.)
- prioritize: This function has two parameters, **`urgent`** and **`important`**, both boolean. It returns the priority according to this rule: urgent & important → 1, important not urgent → 2, urgent not important → 3, neither urgent nor important → 4.

JAVASCRIPT LAB 3 - STUDENT SUBMISSIONS

Task: Create an array of objects representing student submissions. Define a variety of functions for working with such an array. Also call each of the functions at least once to test it.

Build Specifications:

1. Declare a variable named `submissions` that is initialized to an array with the following objects:

name	score	date	passed
Jane	95	2020-01-24	true
Joe	77	2018-05-14	true
Jack	59	2019-07-05	false
Jill	88	2020-04-22	true

2. Declare a function named `addSubmission`
    - Parameter(s): `array`, `newName`, `newScore`, `newDate`
    - Functionality: construct an object and push it into the `array`. The object must have the same properties as the objects already in the array. Use conditional statements to set the value for the `passed` property to `true` if the score is greater than or equal to 60 and `false` otherwise.

3. Declare a function named `deleteSubmissionByIndex`
    - Parameter(s): `array`, `index`
    - Functionality: remove the object from the `array` at the specified `index` using the splice method.

4. Declare a function named `deleteSubmissionByName`
    - Parameter(s): `array`, `name`
    - Functionality: remove the object from the array that has the provided `name`. Incorporate the findIndex method and the splice method.

5. Declare a function named `editSubmission`
    - Parameter(s): `array`, `index`, `score`
    - Functionality: update an object's `score` in the `array` at the specified `index`. Use conditional statements to set the value for the `passed` property to `true` if the score is greater than or equal to 60 and `false` otherwise.

continued on the next page...

6. Declare a function named `findSubmissionByName`
   - Parameter(s): `array`, `name`
   - Functionality: return the object in the `array` that has the provided `name`. Use the find method.

7. Declare a function named `findLowestScore`
   - Parameter(s): `array`
   - Functionality: return the object in the `array` that has the lowest score. Use the forEach method to loop through the whole array.

8. Declare a function named `findAverageScore`
   - Parameter(s): `array`
   - Functionality: return the average quiz score. Use a for...of loop.

9. Declare a function named `filterPassing`
   - Parameter(s): `array`
   - Functionality: return a new array using the filter method. The filter method should find objects in the `array` that have passing scores.

10. Declare a function named `filter90AndAbove`
    - Parameter(s): `array`
    - Functionality: return a new array using the filter method. The filter method should find objects in the `array` that have scores greater than or equal to 90.

Tests: Same as build specifications.

Extended Challenges:

1. Create a function named `createRange`
   - Parameter(s): `start`, `end`
   - Functionality: construct and return an array of integers starting with the start parameter and ending at the end parameter. e.g `createRange(2, 5)` returns `[2, 3, 4, 5]`.
2. Create a function named `countElements`
   - Parameter(s): `array` (an array of strings)
   - Functionality: construct and return an object with the array values as keys and the number of times that key appears in the array as values. e.g. `countElements(['a', 'b', 'a', 'c', 'a', 'b'])` returns `{ a: 3, b: 2, c: 1 }`.

JAVASCRIPT EXERCISES

JAVASCRIPT REFRESHER EXERCISE

Skills: JavaScript variables, data types, arrays, and for loops review from Unit 1.

Task: In preparation for the in-class JavaScript lesson, try to do as much of this exercise as you can.

Setup

Create a JavaScript REPL on repl.it or create a VS code project that includes a JavaScript file.

Build Specifications

1. Create a `firstName` variable. Set it to your first name.
2. Create a `lastName` variable. Set it to your last name.
3. Create a `name` variable. Set it by concatenating firstName and lastName with a space between. (Or use a template literal.) (Don't know what concatenate means? Google "JavaScript concatenate.")
4. Log the `name` variable to the console.
5. Create a `population` variable. Set it to the population of Detroit or your favorite city.
6. Log `population` to the console.
7. Multiply the `population` by 3 and log that to the console.
8. Create a `javaScriptIsCool` variable. Set it to the boolean value `true`.
9. Create a `colors` variable. Set it to an array of the colors of the rainbow. (Each color is a string.)
10. Write a for loop that counts from 1 to 5 and logs each number to the console.
11. Write a for loop that counts from 0 to 4 and logs each number to the console.

Sample Output

```
Grant Chirpus
673104
2019312
1
2
3
4
5
0
1
2
3
4
```

CONDITIONS EXERCISE 1

Skills: Conditions

Heating/Cooling exercise:
Define two variables: **`actualTemp`** and a **`desiredTemp`**. Write conditionals to tell the heating & cooling system what to do. Log one of these three things: Run A/C, Run heat, or Standby.

Extended Challenges second exercise:
Start with two variables: **`tempCelsius`** (a number) and **`targetUnit`** (a string, either “C”, “F”, or “K”). Write a switch statement that checks the targetUnit and logs the temperature converted to that unit. Notes: K stands for Kelvin. C requires no conversion, print out the original temp.

CONDITIONS EXERCISE 2

Skills: Conditions

Task: Identify the longest of three names.

Build Specifications:
Start with three variables `name1`, `name2`, and `name3` which hold three names.

The code should output the longest of the three names. e.g., "Ada Lovelace has the longest name.". If there is a tie between any two, list both. e.g., “Charles and Brendan tie for the longest name.”. Or if all three names are the same length, output “All three names, Dorothy, Charles, and Brendan, are the same length.”

Make sure to test your code with many different combinations of lengths. There are 7 possible outcomes.

LOOPS EXERCISE 1

Skills: Conditions and loops

Counting Loops:

- Use a for loop to log numbers from 1 to 10, then another for loop to count down, logging numbers from 10 to 1.
- Repeat the exercise with a while loop.
- Repeat with a do while.
- Create this array: const numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]. Use a for ... of to loop through this array and log all the numbers. (Note: we're not asking you to also count down on this one.)

Extended Challenge / Second Exercise:

Given any string, use a loop to add padding (extra spaces) to the front to make the string ten characters long. For example, given the string "planet", log "   planet" (add four spaces), and for the string "headlamp", log " headlamp" (add two spaces).

FUNCTIONS EXERCISE 1

Skills: Functions

- Write a function named **getAreaOfCircle**. It takes a **radius** parameter. It calculates and returns the area of a circle with that radius.
- Write a function named **getCircumferenceOfCircle**. It takes a **radius** parameter. It calculates and returns the circumference of a circle with that radius.
- Write a function named **getAreaOfSquare**. It takes a **side** parameter. It calculates and returns the area of a square with that side length.
- Write a function named **getAreaOfTriangle**. It takes two parameters: **base** and **height**. It calculates and returns the area of a triangle with that base and height.

Call each of these functions and log the result to the console. (NOTE: None of these functions should use console.log within them. Instead, they must return the calculated value.)

FUNCTIONS EXERCISE 2

Skills: Hoisting / Scope / Closures / IIFE's

1 Dimensional:
A character can move forward and backward along a tightrope. They start at position 0, facing the positive direction and can move in the positive and the negative directions. They can also change direction. Use variables to keep track of their position and direction. Then define the following functions:

- `moveForward`: takes a distance parameter. Move the character forward (based on the direction they are facing) the specified distance.
- `moveBackward`: takes a distance parameter. Move the character backward (based on the direction they are facing) the specified distance.
- `turnAround`: reverse the direction the character is facing.
- `printLocation`: logs the current position to the console.

Call the functions in the following order and check the results:

1. moveForward 5
2. moveBackward 3
3. printLocation … should print 2
4. turnAround
5. moveForward 10
6. moveBackward 5
7. printLocation … should print -3

Wrap the whole program in a block or IIFE to prevent leaking global variables.

See extended challenge on next page…

Extended Challenge 2 Dimensional:
A character can move around a map in two dimensions. They start at position 0 North, 0 East, facing north and can move in any of the four cardinal directions. They can also change direction. Use variables to keep track of their position and direction. Then define the following functions:

- `moveForward`: takes a distance parameter. Move the character forward (based on the direction they are facing) the specified distance.
- `moveBackward`: takes a distance parameter. Move the character backward (based on the direction they are facing) the specified distance.
- `turnLeft`: change the direction by 90 degrees to the left.
- `turnRight`: change the direction by 90 degrees to the right.
- `printLocation`: logs the current position (N and E) to the console.

Call the functions in the following order and check the results:

1. moveForward 5
2. turnRight
3. moveForward 2
4. printLocation ... should print 5 N, 2 E
5. turnLeft
6. turnLeft
7. moveForward 7
8. turnRight
9. moveBackward 3
10. printLocation ... should print 2 N, -5 E

Wrap the whole program in a block or IIFE to prevent leaking global variables.

OBJECTS/ARRAYS EXERCISE 1

Skills: Objects, arrays, and loops

Create an array of 3 characters. Each character is an object with two properties--name and health. Create an object outside the array in a variable called opponent. The opponent also has name and health properties. (the opponent should start with more health than any of the characters.)

Create a loop that:

- Prints out the name and health of all three characters plus the opponent.
- Prompts the user to pick a character by number (1, 2, or 3).
- "Battles" the selected character against the opponent. Remove five health from both the character and the opponent. (Or you can remove a random number from each.)
- Exit the loop when the user response is null, which means that the cancel button was clicked.

Extended Challenges:

1. Do not allow a character with zero or less health to engage in battle.
2. Check for invalid input (must be number between 1 and 3).
3. Automatically end the program when the opponent health hits zero or all of the characters' health has hit zero. Display a "win" or "lose" message.

ES6 CLASSES EXERCISE 1

Skills: JavaScript Classes, properties, methods, default arguments

Overview: Complete the following exercise first using JavaScript ES6 classes.

Note that there are three general scenarios for properties.

1. properties set from constructor parameters
2. properties not set from constructor parameters. These generally start with a specific value.
3. properties set from optional constructor parameters. These generally start with a specific default value if not passed to the constructor.

Phase 1:

1. Create a class named `Room`.
2. Give it three properties set by three corresponding constructor parameters. (scenario #1)
   a. `name` (string)
   b. `length` (number)
   c. `width` (number)
3. Add two methods:
   a. `getArea`: no parameters. returns a number, the calculated rectangle area based on length and width.
   b. `getPerimeter`: no parameters. returns a number, the calculated rectangle perimeter based on length and width.
4. Below the class definition, create two instances of the Room object:
   a. `room1` - name: Sun, length: 30, width: 20
   b. `room2` - name: Green, length: 15, width: 20
5. Then console log the following for each of the two new objects: name, length, width, area, and perimeter.

Phase 2:

6. Add an `available` (boolean) property to `Room` that is not set from a constructor parameter. It always starts out as `true`. (scenario #2)
7. After creating room2, set `available` to `false`.
8. Add a console.log for the `available` property of both rooms.

Phase 3:

9. Add a `capacity` (number) property to `Room` that is an optional constructor parameter. If not specified, default to `15`.
10. Modify the new call for `room2` to set `capacity` to `18` using a constructor parameter.
11. Add a console.log for the `capacity` property of both rooms.