# Iris Data Visualization Report

# Task 7 & 8

Jana Ammar, Mars Lapierre Furtado, Bianca Granata

## Objective 12: Report Summary

This report covers Objectives 2 through 10 of the Iris Data Visualization task. The code was implemented in JavaScript using the Fetch API to load data from 'iris.json', followed by various higher-order array operations. The visualization portion uses a canvas-based animated scatter plot to represent iris flowers in a creative, interactive way.

## Higher-Order Array Function Implementations

2. map(): Added a new 'color' property to each iris object by randomly selecting a color from a predefined array.

```
        // 2 - map()

        const irisesWithColors = irisData.map(iris => {

            const randomIndex = Math.floor(Math.random() * possibleColor.length);

            return { ...iris, color: possibleColor[randomIndex] };

        });

        console.log("2- Irises with Colors:", irisesWithColors);
```

3. filter(): Filtered out any iris object where sepalWidth >= 4 to create a 'filteredIrises' array.

```
        // 3 - filter()

        const filteredIrises = irisesWithColors.filter(iris => iris.sepalWidth < 4);

        console.log("3- Filtered Irises (sepalWidth < 4):", filteredIrises);
```

4. reduce(): Used reduce() to calculate the total petalLength, then computed the average.

```
        // 4 - reduce()

        const totalPetalLength = irisesWithColors.reduce((acc, iris) => acc +
iris.petalLength, 0);

        const averagePetalLength = totalPetalLength / irisesWithColors.length;

        console.log("4- Average petalLength:", averagePetalLength);
```

5. find(): Found the first iris object with petalWidth > 1.0.

```
        // 5 - find()
        const irisWithLargePetalWidth = irisesWithColors.find(iris => iris.petalWidth > 1.0);

        console.log("5- Iris with petalWidth > 1.0:", irisWithLargePetalWidth);
```

6. some(): Checked if any iris had a petalLength > 10.

```
        // 6 - some() > 10

        const hasPetalLengthGreaterThan10 = irisesWithColors.some(iris => iris.petalLength > 10);

        console.log("6- Is there an iris with petalLength > 10?", hasPetalLengthGreaterThan10);
```

7. some(): Checked if any iris had a petalLength exactly equal to 4.2.

```
        // 7 - some() == 4.2

        const hasPetalLengthEqual42 = irisesWithColors.some(iris => iris.petalLength === 4.2);

        console.log("7- Is there an iris with petalLength == 4.2?", hasPetalLengthEqual42);
```

8. every(): Checked if all iris objects had petalWidth < 3.

```
        // 8 - every() petalWidth < 3

        const allPetalWidthsLessThan3 = irisesWithColors.every(iris => iris.petalWidth < 3);

        console.log("8- Do all irises have petalWidth < 3?", allPetalWidthsLessThan3);
```

9. every(): Checked if all iris objects had sepalWidth > 1.2.

```
        // 9 - every() sepalWidth > 1.2

        const allSepalWidthsGreaterThan12 = irisesWithColors.every(iris => iris.sepalWidth > 1.2);

        console.log("9- Do all irises have sepalWidth > 1.2?", allSepalWidthsGreaterThan12);
```

10. toSorted(): Sorted the array of iris objects by petalWidth in ascending order and stored it in 'irisesWithColorsSorted'.

```
        // 10 - toSorted()

        const irisesWithColorsSorted = irisesWithColors.toSorted((a, b) => a.petalWidth - b.petalWidth);
```

```
    console.log("10- Irises sorted by petalWidth:", irisesWithColorsSorted);
```

## Visualization Summary (Objective 11)

The sorted iris dataset was visualized using a canvas-based interactive scatter plot. Each flower is represented by a colored circle, where color is randomly assigned. The X and Y axes represent sepalLength and petalLength respectively. A dropdown menu allows users to filter data points by species. When a user hovers over a data point, the species label is displayed beside it. The visualization was created using a JavaScript class 'Iris', which encapsulates each data point’s rendering logic. The canvas is animated using requestAnimationFrame().