Concordia University
Faculty of Fine Arts
Department of Design and Computation Arts

Task 7 & 8

JSON & ARRAYS & VISUALIZE

CART 263
Creative Computation II
Sabine Rosenberg
Section B

Submitted By:
Rowan Nasser
Maia Arrais Mateo

Submission Date:
April 10th, 2025

Tasks Description

Task 2: Using map() function to add a random color to each object. The possible colors to select from: "#5d3fd3", "#a73fd3", "#d33fb5", "#d35d3f", "#d3a73f". We used map() to create a new array called irisesWithColors. Each object in the array now includes a new property called color that was randomly chosen from the possibleColor array:

```
// 2. map(): Add a random color field
let possibleColor = ["#5d3fd3", "#a73fd3", "#d33fb5", "#d35d3f", "#d3a73f"];
const irisesWithColors = irisData.map(iris => ({
    ...iris,
    color: possibleColor[Math.floor(Math.random() * possibleColor.length)]
}));
```

Task 3: Using filter() to remove all iris objects with sepalWidth >= 4.
We created a new array called filteredIrises using the filter() method:

```
// 3. filter(): Filter out sepalWidth >= 4
const filteredIrises = irisesWithColors.filter(iris => iris.sepalWidth < 4);
```

Result: The number of objects after filtering is 146

Task 4: Using reduce() to calculate the average petalLength across all iris objects.
We first calculated the total sum of all petalLength values using reduce() and then divided it by the array length to find the average:

```
// 4. reduce(): Calculate average petalLength
const totalPetalLength = irisesWithColors.reduce((sum, iris) => sum + iris.petalLength, 0);
const avgPetalLength = totalPetalLength / irisesWithColors.length;
```

Result: Average petalLength is 3.7580000000000027

Task 5: Using find() to retrieve the first iris object with petalWidth > 1.0.

```
// 5. find(): Find object with petalWidth > 1.0
const foundIris = irisesWithColors.find(iris => iris.petalWidth > 1.0);
```

Result: Snippet of found Iris

Task 6: Using some() to check if **any** object has petalLength > 10.

```
// 6. some(): Any object with petalLength > 10?
const hasPetalLengthGreaterThan10 = irisesWithColors.some(iris => iris.petalLength > 10);
```

Result: False

Task 7: Using some() to check if **any** object has petalLength === 4.2.

```
// 7. some(): Any object with petalLength == 4.2?
const hasPetalLengthEqualTo4_2 = irisesWithColors.some(iris => iris.petalLength === 4.2);
```

Result: True

Task 8: Using every() to check if **all** objects have petalWidth < 3.

```
// 8. every(): All objects with petalWidth < 3?
const allPetalWidthLessThan3 = irisesWithColors.every(iris => iris.petalWidth < 3);
```

Result: True

Task 9: Using every() to check if **all** objects have sepalWidth > 1.2.

```
// 9. every(): All objects with sepalWidth > 1.2?
const allSepalWidthGreaterThan1_2 = irisesWithColors.every(iris => iris.sepalWidth > 1.2);
```

Result: True

Task 10: Using toSorted() to sort all objects by petalWidth (smallest to largest).

```
// 10. toSorted(): Sort by petalWidth ascending
const irisesWithColorsSorted = irisesWithColors.toSorted((a, b) => a.petalWidth - b.petalWidth);
```

This sorted array was used to render the visual display.

Task 11: Visualization Summary

We created a colorful and dynamic visualization of the dataset using the irisesWithColorsSorted array.

Each iris is represented as a circle ("bubble") with:

- **Color:** randomly selected from a predefined palette
- **Size:** determined by the petalLength and petalWidth
- **Label:** the species name is displayed inside the circle
- **Position:** placed randomly across the screen to create a messy effect
- **Interactivity:** each circle Zooms in on hover and shows petal/sepal data

The visualization is built using a class named Iris, where each data object is turned into a visual element using the render() method. The interface is styled with CSS and responsive to screen size.

This approach allowed us to meet all required visualization features: creative use of color and size, animation on hover, interactivity, and dynamic layout using JavaScript DOM manipulation.

Screenshots

Figure 1 – Consol Logs

Console Log results of the running code.

Figure 2 – Running Code

Each circle represents a flower. Hovering shows petal/sepal data and enlarges the bubble.