Module 4

Handling DOM Events

Creating HTML Dynamically

Updating HTML Content Dynamically

Managing Web Forms with JavaScript

Functions (R)

A **function** is a code snippet that can be called by other code or by itself, or a variable that refers to the function.

When a function is called, arguments are passed to the function as input, and the function can optionally return a value.

A function in JavaScript is also an object.

A function name is an identifier included as part of a **function declaration** or **function expression**.

Function declaration (R)

The function declaration (function statement) defines a function with the specified parameters.

Syntax:

```
function name([param[, param,[..., param]]]) {
   // statements
}
```

- **name** - The function name.
- **param** (Optional) - The name of an argument to be passed to the function. Maximum number of arguments varies in different engines.
- **statements** (Optional) - The statements which comprise the body of the function.

Function declaration, cont.

By default, functions return `undefined`.

To return any other value, the function must have a `return` statement that specifies the value to return.

A simple example:

```
function calcRectArea(width, height) {
  return width * height;
}

let result = calcRectArea(7, 6)
console.log(result);
// expected output: 42
```

Function expression

The function keyword can also be used to define a function inside an expression.

A function expression is very similar to and has almost the same syntax as a function declaration.

The main difference between a function expression and a function declaration is the function name, which can be omitted in function expressions to create **anonymous functions**.

> *A function expression can be used as an IIFE (Immediately Invoked Function Expression) which runs as soon as it is defined.*

Function expression, cont.

A simple example:

```
const getRectArea = function(width, height) {
  return width * height;
};

let result = getRectArea(6, 7);
console.log(result);// expected output: 42
```

Same example as an arrow function:

```
const getRectArea2 = (width, height) => width * height;

let result2 = getRectArea2(3, 14);
console.log(result2);// expected output: 42
```

Using a function as a callback

A callback function is a function passed into another function as an argument, which is then invoked inside the outer function to complete some kind of routine or action.

HTML:

```
<button id="clickMe">I'm a button</button>
```

JS:

```
const button = document.querySelector("button#clickMe");

button.addEventListener('click', function(event) {
    console.log('button is clicked!');
    console.log(event.target); // Access event object
});
```

Codepen example, The Event Object @ W3School, The MouseEvent Object @ W3School

```
function greeting(input) {
    document.body.innerHTML = `Hello ${input}!`;
}

function processUserInput(callback) {
    var name = prompt('Please enter your name.');
    callback(name);
}

processUserInput(greeting);
```

> Notice that the greeting function is passed as an argument without the `()`, passing the whole function in as an argument.

The above example is a **synchronous** callback, as it is executed immediately.

Codepen example

Sorting using compare functions

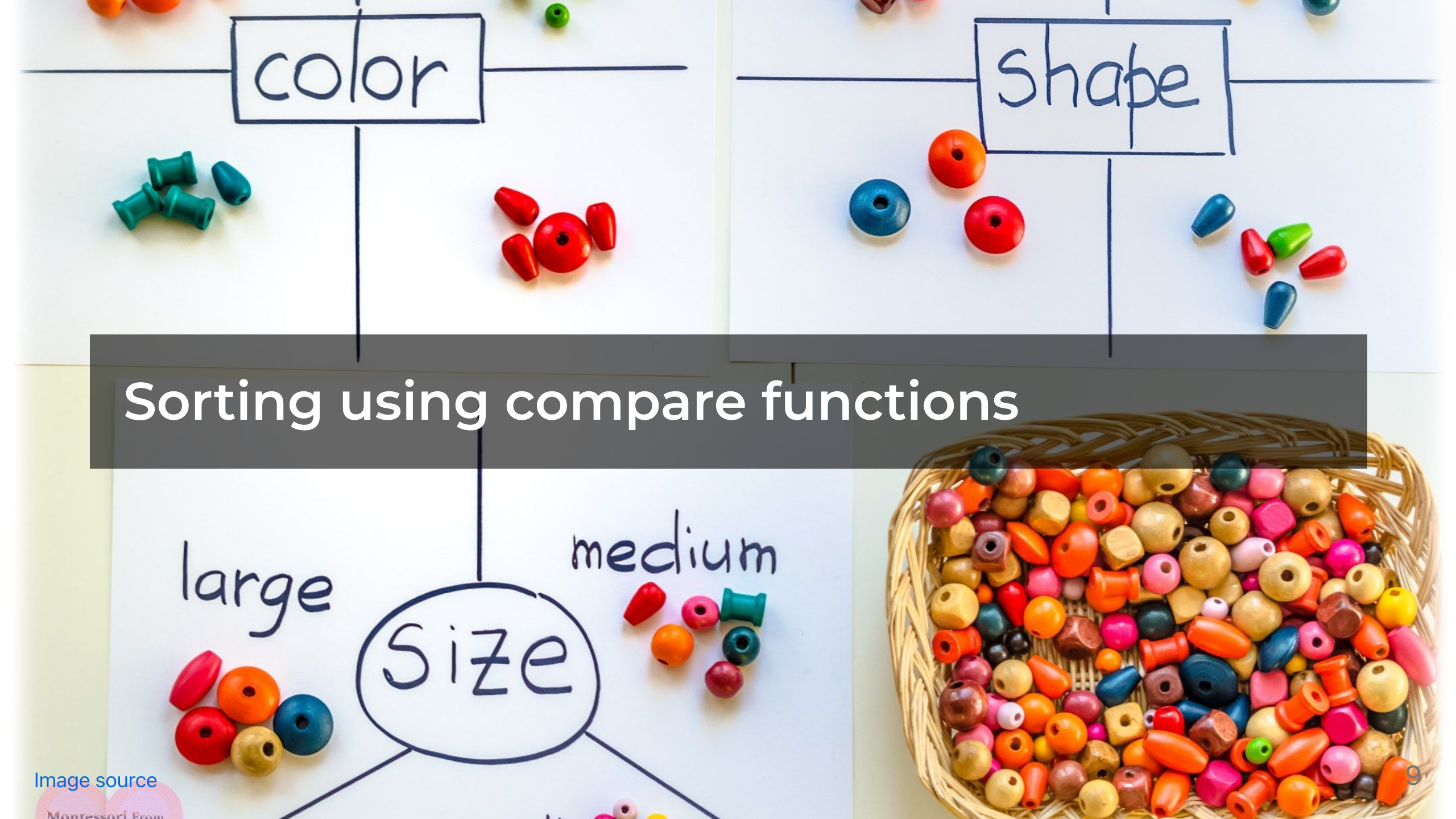

Image source

Sorting an Array

The `sort()` method sorts an array alphabetically:

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];

fruits.sort(); // Sorts the elements of fruits

console.log(fruits);
// > Array(4) [ "Apple", "Banana", "Mango", "Orange" ]
```

The `sort()` method sorts the elements of an array in place and returns the sorted array. The default sort order is **ascending**, built upon converting the elements into strings, then comparing their sequences of UTF-16 code units values.

Note that the array is sorted *in place*, and **no copy is made**.

Reversing an Array

The `reverse()` method reverses the elements in an array.

You can use it to sort an array in descending order:

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];

fruits.sort();          // First sort the elements of fruits

fruits.reverse();       // Then reverse the order of the elements

console.log(fruits);
// > Array(4) [ "Orange", "Mango", "Banana", "Apple" ]
```

Numeric Sort

By default, the `sort()` function sorts values as strings.

This works well for strings ("Apple" comes before "Banana").

However, if numbers are sorted as strings, "25" is bigger than "100", because "2" is bigger than "1".

Because of this, the `sort()` method will produce incorrect result when sorting numbers.

Numeric Sort

You can fix this by providing a **compare function**:

```
var points = [40, 100, 1, 5, 25, 10];

points.sort(function(a, b){return a - b});

console.log(points);
// Array(6) [ 1, 5, 10, 25, 40, 100 ]
```

Use the same trick to sort an array descending:

```
var points = [40, 100, 1, 5, 25, 10];

points.sort(function(a, b){return b - a});

console.log(points);
// Array(6) [ 100, 40, 25, 10, 5, 1 ]
```

The Compare Function

The purpose of the compare function is to define an alternative sort order.

The compare function should return a negative, zero, or positive value, depending on the arguments:

```
function(a, b){return a - b}
```

When the `sort()` function compares two values, it sends the values to the compare function, and sorts the values according to the returned (negative, zero, positive) value:

- If the result is **negative** a is sorted before b.
- If the result is **positive** b is sorted before a.
- If the result is 0 no changes are done with the sort order of the two values.

The Compare Function, example:

```
var points = [40, 100, 1, 5, 25, 10];

points.sort(function(a, b){return a - b});
```

The compare function compares all the values in the array, two values at a time `(a, b)`, as called from the `sort()` method.

When comparing `40` and `100`, the `sort()` method calls the compare function with the arguments (40, 100).

The compare function calculates `(a - b)`, ie. 40 - 100, and since the result reurned is negative (-60), the sort function will sort `40` as a value lower than `100`.

This is then repeated for all permutations of the items in the array.

Sorting Object Arrays

JavaScript arrays often contain objects:

```
const cars = [
  {type:"Volvo", year:2016},
  {type:"Saab", year:2001},
  {type:"BMW", year:2010}
];
```

Even if objects have properties of different data types, the `sort()` method can be used to sort the array.

The solution is to write a **compare function** to compare the property values.

Sorting Object Arrays by numeric value

Given the `cars` Array from the last slide:

```
const sortByNumericValue = (a, b) =>  a.year - b.year;

cars.sort(sortByNumericValue);

console.log(cars);
// (3) [...]
// 0: Object { type: "Saab", year: 2001 }
// 1: Object { type: "BMW", year: 2010 }
// 2: Object { type: "Volvo", year: 2016 }
```

Sorting Object Arrays by string value

Comparing string properties can be a little more complex:

```
const sortByStringValue = (a, b) => {
    var x = a.type.toLowerCase(); // Ignore case
    var y = b.type.toLowerCase(); // Ignore case
    if (x < y) { return -1; }
    if (x > y) { return 1; }
    return 0; // They are the same
}

cars.sort(sortByStringValue);

console.log(cars);
// (3) [...]
// 0: Object { type: "BMW", year: 2010 }
// 1: Object { type: "Saab", year: 2001 }
// 2: Object { type: "Volvo", year: 2016 }
```

Codepen

Sorting non-ASCII characters

For sorting strings with non-ASCII characters, i.e. strings with accented characters (e, é, è, a, ä, etc.), strings from languages other than English, use `String.localeCompare`. This function can compare those characters so they appear in the right order.

```
var items = ['réservé', 'årstid', 'premier', 'ærfugl', 'communiqué', 'café', 'Ørskog', 'adieu', 'éclair'];

items.sort();
console.log(items);
// Array(9) [ "adieu", "café", "communiqué", "premier", "réservé", "Ørskog", "årstid", "ærfugl", "éclair" ]

const sortLocaleNorwegian = function (a, b) {
  return a.localeCompare(b, 'no'); // Note the 'no' locales argument**
}

items.sort(sortLocaleNorwegian);
console.log(items);
// Array(9) [ "adieu", "café", "communiqué", "éclair", "premier", "réservé", "ærfugl", "Ørskog", "årstid" ]
```

Codepen

If "no" doesn't work, try "nb", "nn" or "nb-NO", or similar.

Example: Listing programming languages

```
const output = document.querySelector("ul#list"); // assuming an <ul id="list"> in HTML
const languages = [
    "Python", "Java", "R", "Javascript", "Swift", "C++", "C#", "PHP", "Sql", "Go"
];
output.innerHTML = listMyArray(languages);

const sortedList = languages.toSorted();
//output.innerHTML = listMyArray(sortedList);

const empty = [];
//output.innerHTML = listMyArray(empty);

function listMyArray (list) {
  let str = "";
  for (let item of list) {
    str += `<li>${item}</li>`;
  }
  if (str === "") str = "<li>No elements in list</li>";
  return str;
}
```

Codepen

Example: Listing Apple sorts

```
const output = document.querySelector("ul#list"); // assuming an <ul id="list">

// https://www.openfit.com/a-guide-to-the-most-popular-apple-varieties
const apples = [
    { sort: "Red Delicious",
    flavourProfile: `After generations of breeding for longer shelf life and cosmetic stability
     – call it vanity ripeness – the flavor has mostly been cultivated out of the Red Delicious.
    It now has thick skin, a one-note sweet taste, and an often crumbly texture` },
    { sort: "McIntosh", flavourProfile: `With soft skin and softer flesh, the McIntosh
    strikes a balance between sweet and acidic.` },
    { sort: "Golden Delicious", flavourProfile: `The meat of the apple is mild and sweet,
    the flesh is juicy, but taste-wise isn't all that different from a red delicious.` },
    { sort: "Gala", flavourProfile: `With pinkish-orange striping over a gold base, its skin
    is thin, concealing a crisp and juicy flesh that's fragrant and relatively sweet.` },
    { sort: "Granny Smith", flavourProfile: `If you're into tartness, this bitter old bird
    is your go-to. It's crisp and has juicy flesh. These apples do sweeten with storage.` },
    { sort: "Fuji", flavourProfile: `These apples are dense, crisp, and have been regarded
    by some as the sweetest of all apple varieties.` },
];

listMyApples = (apples) => {
  // console.log(apples);
  let str = "";
  for (let apple of apples) {
    str += `<li><strong>${apple.sort}</strong>: ${apple.flavourProfile}</li>`;
  }
  return str;
}

output.innerHTML = listMyApples(apples);
```

Codepen

Demos

1. **Products** - listing products using a template and `innerHTML`
2. **Restaurants** - listing restaurants using a template and `innerHTML`
3. **Big Cats** - listing and *filtering* big cats using `Array.filter()`
4. **Books** - listing and sorting using `innerHTML` and compare functions

Todos

GitHub Classroom

JS1 Lesson 4.2 Creating HTML Dynamically

JS1 Lesson 4.3 Updating HTML Content Dynamically

Mollify

Read Updating HTML Content Dynamically, and do the Lesson Task