JavaScript

CS5610: Web Development

Joydeep Mitra

Pre-class Activity

- Fork the repo https://github.com/CSE-316-Software-Development/learn-js
  - Create a new branch with the name: *today's date(MMDDYY)*
  - Enter your full name in README.md
  - Push to the branch when done.
- Useful git commands:
  - Create new local branch
    ```
    $ git checkout -b <branch-name>
    ```
  - Push new branch to remote
    ```
    $ git push origin <branch-name>
    ```
  - Push modified/new files to remote branch
    ```
    $ git add <path/to/file>
    $ git commit -m "useful message"
    $ git push
    ```

Introduction

- JavaScript can access and change HTML elements and attributes.
  - The **getElementById()** method finds an HTML element by id and changes its attributes (e.g., *innerHTML, textContent, style,* etc.).
- JavaScript output commands:
  - Write to an HTML page
    document.write()
  - Write to an alert box
    window.alert()
  - Write to browser console
    console.log()
  - Write to a printing device
    window.print()
- Let's look at *learn-js/basic.html.*

Where To Put JavaScript Code

- JavaScript is placed between **<script>** and **</script>** tags in HTML.
- The **<script>** and **</script>** tags can be in the **<head>** or **<body>**.
- JavaScript code is often written in functions.
  - A JavaScript function is a block code that can be executed when the function is called.
  - Functions are called when an event occurs (e.g., button click).
  - More on functions later!

External JavaScript

- Often, we may want to separate HTML and JavaScript to improve readability and performance (faster page loads).
  - An external file (without <script> tags):
    - `<script src="/path/to/myScript.js"></script>`
  - JavaScript a URL:
    - `<script src="https://cs5500.northeastern.edu/js/myScript.js"></script>`
- Let's see example in *learn-js/where2putjs.html*.

For You to Do

- Open *quiz/where2putjs.html* in a browser and inspect the console.
  - Do you see an error? Why?
  - Fix the error.

Statements

- What is a JavaScript program?
  - a sequence of instructions or statements.
- What are statements?
  - composed of values, operators, expressions, keywords, and comments.
  - semicolons separate statements (optional but recommended).
  - can be grouped together in code blocks, inside curly bracket {..}.
  - statements in a code block are executed together (e.g., functions).

Variables

- Variables are containers for storing values.
- JavaScript variables are declared using var, let, const, and nothing.

```
var x = 5;
var y = 6;
var z = x + y;
```

```
let x = 5;
let y = 6;
let z = x + y;
```

```
x = 5;
y = 6;
z = x + y;
```

- When declaring variables always use var, let, or const.
  - All JavaScript code between 1995 and 2015 use var.
  - let and const were added to JavaScript in 2015.
  - To run JavaScript in older browser, you must use var.
  - More on their differences later!

Variable Rules

- All variables have a unique identifier.
  - Can contain letter, digits, underscores, and dollar signs.
  - Must begin with a letter.
  - Are case sensitive.
  - Reserved words cannot be used as identifier names.
- The = operator is used as an assignment operator. E.g.,
  x = x + 5
- Variables can hold values of many data types such as numbers and strings. Why?
  - JavaScript is dynamically typed.
    - `Pros => Flexibility for the developer.`
    - `Cons => Potential type errors at runtime.`

Variables

- Creating a variable in JavaScript is called “declaring” a variable.
- Declaring a variable using var or let without a value makes the variable *undefined.* E.g.,
  ```
  var x; // x is undefined
  ```
- You can declare multiple variables at the same time. E.g.,

  -
    ```
    let person = "John Doe", carName = "Volvo", price = 200;
    ```
- Re-declaring a variable with `var` will not lose its value.

  -
    ```
    var carName = "Volvo";
    var carName; // carName = “Volvo”
    ```

Variables

- You can write arithmetic expressions with JavaScript variables. E.g.,
  ```
  let x = 5 + 2 + 3;
  ```
- Strings can be concatenated with the + operator. E.g.,
  ```
  let x = "John" + " " + "Doe";   // x = "John Doe"
  ```
- If types are mixed, then we get unexpected behavior. E.g.,
  ```
  let x = "5" + 2 + 3;
  let x = 2 + 3 + "5";
  ```
- `JavaScript is weakly typed.`

Variables: `let` vs. `var`

- Variables declared with `let` cannot be re-declared.

```
let x = "John Doe";

let x = 0;   // SyntaxError: 'x' has already been declared
```

- Variables declared with `var` can be re-declared.

```
var x = "John Doe";

var x = 0;
```

Variables: The `let` Keyword

- Variables declared with `let` inside a block {..} **cannot** be accessed from outside the block.

```
{
  let x = 2;
}
// Can x be used here? // No
```

- Variables declared with `var` inside a block {..} **can** be accessed from outside the block.

```
{
  var x = 2;
}
// x CAN be used here
```

Variables: The let Keyword

- Redeclaring a variable with let inside a block {..} will not redeclare it outside the block.

```
let x = 10;
// Here x is 10

{
let x = 2;
// Here x is 2
}
// What is x here?  // x is 10
```

- Redeclaring a variable with var inside a block {..} will redeclare it outside the block.

```
var x = 10;
// Here x is 10

{
var x = 2;
// Here x is 2
}

// What is x here? // x is 2
```

Variables: The `let` Keyword

- Using a `let` variable before it is declared will result in a reference error.

```
carName = "Saab"; // reference error
let carName = "Volvo";
```

- Variables declared with `var` can be used any time as such variables are **hoisted** to the top.

```
carName = "Volvo";    // this is OK
var carName;
```

Variables: The `const` Keyword

- A `const` variable cannot be reassigned.

```
const PI = 3.141592653589793;
PI = 3.14;              // This will give an error
PI = PI + 10;          // This will also give an error
```

- A `const` variable must be initialized when its declared.

```
const PI = 3.14159265359;  //correct

//incorrect
const PI;
PI = 3.14159265359;
```

Variables: The `const` Keyword

- As a general rule, we always use `const` unless we know the value will change. E.g.,
  - Declaring a new array, a new object, a new function, a new regular expression.
- The keyword `const` does not declare a constant value; it **declares a constant reference** to a value.
- *Therefore, we cannot reassign a constant array or object but we can change elements in a constant array or properties of a constant object.*

```
// We can create a constant array:
const cars = ["Saab", "Volvo", "BMW"];

// Is this allowed? // Yes. changes
cars[0] = "Toyota"; // an element:

// Is this allowed? // Yes.
cars.push("Audi");  // adds an element:

// Is this allowed? // ERROR
cars = ["Toyota", "Volvo", "Audi"];
```

Variables: The `const` Keyword

- Just like `let`, redeclaring a variable using const in a block does not redeclare it outside the block.

```
const x = 10;
// Here x is 10

{
const x = 2;
// Here x is 2
}

//What is x?

 Here x is 10
```

Variables: The const Keyword

- Redeclaring an existing let , var, or const variable using const, in the same scope, is not allowed.

```
var x = 2;      // Allowed
const x = 2;    // Not allowed

{
let x = 2;      // Allowed
const x = 2;    // Not allowed
}

{
const x = 2;    // Allowed
const x = 2;    // Not allowed
}
```

Variables: The `const` Keyword

- Reassigning an existing `const`, in the same scope, is not allowed.

```
const x = 2;
x = 2;                  // Not allowed
var x = 2;              // Not allowed
let x = 2;              // Not allowed
const x = 2;            // Not allowed

{
  const x = 2;          // Allowed
  x = 2;                // Not allowed
  var x = 2;            // Not allowed
  let x = 2;            // Not allowed
  const x = 2;          // Not allowed
}
```

Variables: The `const` Keyword

- Using a `const` variable before it is declared will result in a **reference error**.

```
alert (carName);        // this is an error
const carName = "Volvo";
```

Arithmetic Operators

- Arithmetic operators work on numbers, variables, literals, or a combination of them.
  - Addition (+), Subtraction (-), Multiplication (*), Exponentiation (**)
  - Division (/), Modulus (%), Increment (++), Decrement(--)
- Operator precedence and associativity are crucial for evaluating arithmetic expressions.
- Check MDN docs for more information https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Operator_Precedence

For you To Do

- Identify and fix all errors in:
  - *quiz/vars1.html*
  - *quiz/vars2.html*
  - *quiz/vars3.html*

Data Types

- JavaScript has **primitive types** and **objects** (e.g. Arrays).
- Primitive types are *immutable*:
  - String, Number, Boolean (**true** and **false**).
  - Everything else is *mutable.*
- What is *immutability*?
  - Structure cannot be modified after they are created.
- So, can you modify a string after it is created?

Objects

- In JavaScript objects are containers for multiple data values.
- Objects have properties of the form *name : value*.
- They are declared using **const**.

```
const person = {
  firstName: "John",
  lastName: "Doe",
  age: 50,
  eyeColor: "blue"
};
```

- Two ways to access objects:
  - `objectName[“propertyName”]` or
  - `objectName.propertyName`.

Object

- A method is a function defined in an object.

```
const person = {
  firstName: "John",
  lastName : "Doe",
  id       : 5566,
  fullName : function() {
    return this.firstName + " " + this.lastName;
  }
};
```

  - `this` refers to the object in which the function is defined.
  - Object methods can be accessed as `objectName.methodName()`

```
    name = person.fullName();
```

Let’s look at the example in *learn-html/whatisthis.html*

Functions

- A JavaScript function is a block of code that gets executed when the function is invoked.
  - defined with the **function** then a *name* and *optional parameters* inside parenthesis.

```
function name(parameter1, parameter2, parameter3) {
  // code to be executed
}
```

  - must end with a **return** statement which indicates the value that gets returned.
  - *no return statement* returns *undefined* from the function.

Functions

- Functions are *first-class citizens*, i.e., they can be used as expressions.
  - Let’s see example in *learn-js/first-class-funcs.html*

- Variables declared in a function can be used only inside the function.

```
// code here can NOT use carName
function myFunction() {
  let carName = "Volvo";
  // code here CAN use carName
}
// code here can NOT use carName
```

Functions

- Functions may or may not have parameters.
    - Data types are not specified.
    - Parameter types are not checked.
    - No. of arguments received are not checked.
- If a functions is called with missing arguments, then the parameter is assigned *undefined*.
- Therefore, it is prudent, in some cases, to have default values for parameters.

```
function myFunction(x, y) {
  if (y === undefined) {
    y = 2;
  }
}
```

```
function myFunction(x, y = 2) {
  // function code
}
```

Functions

- Code in a function is executed when it is
  - Invoked
  - Called
  - Applied
- Invoking a function is the same as ”calling” a function with arguments and the object in which the function is defined.
- The default object is “window”.
- However, “calling” in JavaScript has other connotations.

Functions

- The `call()` method is used to call functions with different objects.
    - Enables reusability.
- It takes an object as an argument and is used to ”invoke” methods in another object.
- The method is evaluated in the context of the object passed as argument to `call()`.

```
const person = {
  fullName: function() {
    return this.firstName + " " + this.lastName;
  }
}
const person1 = {
  firstName:"John",
  lastName: "Doe"
}
const person2 = {
  firstName:"Mary",
  lastName: "Doe"
}

// What will this return?
person.fullName.call(person1);
```

Functions

- The `call()` method accepts arguments.

```
const person = {
  fullName: function(city, country) {
    return this.firstName + " " + this.lastName + "," + city + "," + country;
  }
}

const person1 = {
  firstName:"John",
  lastName: "Doe"
}

person.fullName.call(person1, "Oslo", "Norway");
```

Functions

- Just like functions can be called, functions can be applied using `apply()`.
- The only difference between `apply()` and `call()` is that `apply()` takes an array as argument as opposed to a list of comma-separated arguments.

```
const person = {
  fullName: function(city, country) {
    return this.firstName + " " + this.lastName + "," + city + "," +
country;
  }
}

const person1 = {
  firstName:"John",
  lastName: "Doe"
}

person.fullName.apply(person1, ["Oslo", "Norway"]);
```

For You to Do

- Define a JavaScript function in *quiz/scripts/logger.js* called *logMsg()* that can be used to log an error message for any object that contains the property *errMsg*.

Arrow Functions

- Arrow functions are functions without names.
- They provide syntax to succinctly define short functions.
- They always return a value.

```
// a function that takes two parameters and returns a number.
let myFunction = (a, b) => a * b;
```

this Keyword in Arrow Functions

- Arrow functions do not have their own **this** binding.
  - They inherit **this** from the surrounding lexical scope.
  - So, don’t use them as methods.
- Let’s look at the example in *learn-js/scripts/arrow.js*

For You to Do

- *quiz/arrow.html* has a bug, which leads to displaying undefined in the screen. Identify the error and fix it.

Function Closures

- What is a closure?
  - A function + surrounding state (lexical environment).
  - They are used in the context of nested functions.
  - Hence, closures are useful to specify callback functions that are invoked due to an event.
- Let’s explore the example in *learn-js/scripts/simple-closure.js*

For You to Do

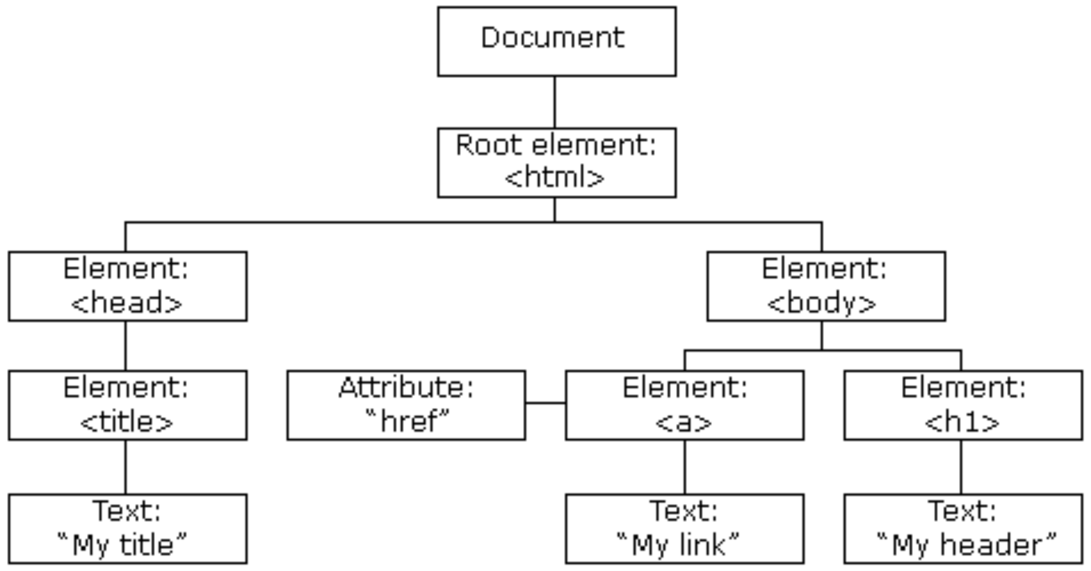
- *quiz/closure.html* has an error in the *onclick* event handler. Fix the error such that the text color changes to the color specified in the text box.

Pre-class Activity

- Fork/Sync the repo https://github.com/CSE-316-Software-Development/learn-js
  - Create a new branch with the name: *today's date(MMDDYY)*
  - Enter your full name in README.md
  - Push to the branch when done.
- Useful git commands:
  - Create new local branch
    ```
    $ git checkout –b <branch-name>
    ```
  - Push new branch to remote
    ```
    $ git push origin <branch-name>
    ```
  - Push modified/new files to remote branch
    ```
    $ git add <path/to/file>
    $ git commit –m "useful message"
    $ git push
    ```

The HTML Document Object Model

- The HTML Document Object Model (DOM) a tree-like structure of all the elements in an HTML page.
- We can use JavaScript to modify, add, and delete objects and their properties in the DOM.
  - content of HTML elements.
  - CSS style elements.
  - React to DOM events.



source: w3schools.org

Finding DOM Elements

- Find Element By Id.

```
const element = document.getElementById("intro"); // gets tag with id=“intro”
```

- Find Element By Tag name.

```
const x = document.getElementById("main");
const y = x.getElementsByTagName("p");   // get all <p> tags in tag with id=“main”
```

- Find Element By class.

```
const x = document.getElementsByClassName("intro");
```

- Find first matching element by selectors.

```
// returns an <input name=“login” in a div with class user-panel main
const x = document.querySelector("div.user-panel.main input[name='login']");
```

Read more https://developer.mozilla.org/en-US/docs/Web/API/Document/querySelector

Working With DOM Elements

- What can we do with DOM elements?
  - Change HTML content in them.

```
document.getElementById(id).innerHTML = new HTML
document.getElementById(id).textContent = new text (more secure for strings)
```

  - Change the value of an attribute.

```
document.getElementById(id).attribute = new value
```

  - Change HTML style.

```
document.getElementById(id).style.property = new style
```

Reacting to DOM Events

- We can execute JavaScript code when an event occurs (e.g., click) by adding code to DOM event attributes.
- We can assign events to HTML elements using JavaScript.
- Let’s explore the example in *learn-js/reactDOM.html*

DOM Events

- There are many DOM events.
  - Focus events, mouse events, keyboard events, etc.
- For the complete reference read https://developer.mozilla.org/en-US/docs/Web/Events

DOM EventListener

- The `addEventListener()` method is used to attach an event handler to a DOM element.
  - Adds a handler to the existing list of handlers.
  - An element can have multiple handlers.
  - Event handlers can be of the same type (e.g., two “click” event handlers).
  - An event handler can be removed using the `removeEventListener()` method.
- Let’s explore the example in *learn-js/addEvent.html*

For You to Do

- Fix the error in *quiz/addEvent.js* such that all event handlers display the expected message when *quiz/addEvent.html* is rendered.

Navigating The DOM

- Recall every tag in the DOM is a node.
- A node has children nodes, which can be accessed by using the property `childNodes[nodenumber]`
- A parent node can be accessed using the property `parentNode`
- A node has a value `nodeValue` and a type `nodeType`
- A node can have the following types:
  - Element node (code 1).
  - Text node (code 3).
  - Comment node (code 8).

Navigating The DOM

- Let’s say we want to access the text in the <td> tags from the following HTML:

```
<div id="root">
      <table>
        <tr>
          <td> Cell 1 </td>
          <td> Cell 2 </td>
          <td> Cell 3 </td>
        </tr>
      </table>
      <button onclick="logCells()">Log Cells</button>
</div>
```

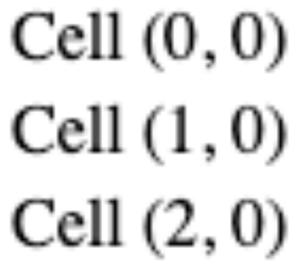
- *learn-js/navigatedom.html* shows how to navigate the DOM.

Manipulating The DOM

- We can add create new elements using the Elements API.
  - E.g., `document.createElement("p")` creates a <p> element
- We can add elements to an existing element node.
  - E.g., `element.appendChild(node)` adds *node* as a child of *element*.
- We can add attributes to an element node.
  - E.g.,
    ```
    button.addEventListener("click", function() {
            foo();
    })
    ```
    adds an *onclick* listener to a button element.
- For more operations of the Elements API refer https://developer.mozilla.org/en-US/docs/Web/API/Element

Manipulating The DOM

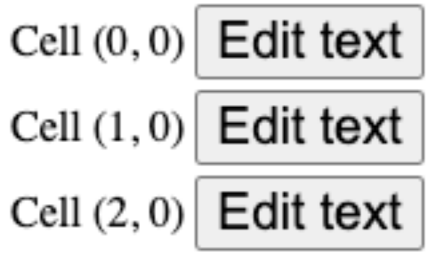
- Let’s look at the example in *learn-js/manipulateDOM.html*
  - Clicking on the *Add Table* button must add following table dynamically to the page.



Cell (0, 0)
Cell (1, 0)
Cell (2, 0)

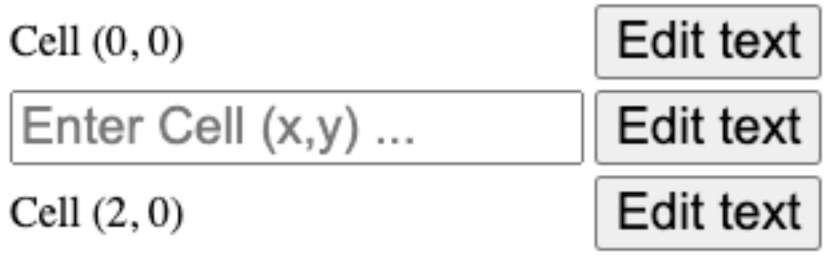
For You to Do

- In *quiz/scripts/manipulatedom.js*, change the file such that when *Add Table* is clicked then the following table is generated and added.



Cell (0, 0)	Edit text
Cell (1, 0)	Edit text
Cell (2, 0)	Edit text

- Clicking *Edit text* must change the table and display a new table as follows:



Cell (0, 0)	Edit text
Enter Cell (x,y) ...	Edit text
Cell (2, 0)	Edit text

Additional Reading

- MDN Docs DOM guides:
  - https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction

Back to Strings!

- String Extraction using `slice`:
  - `slice(start, end)`: extracts from position `start` to `end-1`.
  - `slice(end)`: extracts from position `start` to end of string.
  - If parameter is negative, starts from end of string.
  - If parameters are out of bounds, empty string is returned.

```
let str = "Welcome to strings!";
let y = str.slice(7, 13);    // ' to st'
let z = str.slice(7);        // ' to strings!'
let u = str.slice(-13, -7);  // 'e to s'
```

- **Does String Extraction modify the original string?**

More String Methods

- Searching a string with `indexOf`().
  - `indexOf(t)`: returns the index of the 1st occurrence of `t`.
  - `indexOf(t, n)`: returns the index of the 1st occurrence of `t` starting from `n`.
  - Returns -1 if search string not found.

```
let str = "the people found them!";
str.indexOf("the");           // 0
str.indexOf("the", 5);        // 17
```

Replacing String Methods

- Replacing strings with other strings:
  - `replace(t,s)`: replaces the 1st occurrence of `t` with `s`.
  - If `t` is not found, original string is kept intact.
  - `replace(/rexp/g,s)`: replaces all strings that match `rexp` with `s`.
  - `/rexp/` is a regular expression; `g` indicates global.

```
let x = "Welcome to strings and strings!";
let y = x.replace("strings", "js strings"); // Welcome to js strings and strings!
let z = x.replace(/strings/g, "js strings"); // Welcome to js strings and js strings!
```

Regular Expressions

Modifiers	Meaning
i	Case-sensitive match
g	Global match
m	Multiline match

Anchors	Meaning
^	Begins with
$	Ends with

```
let text = "Learn JavaScript";
let n = text.search(/javascript/i);         // returns 6

let text = "Learn Java and JavaScript";
let n = text.match(/Java/g);         // Java, JavaSript

let text = "This\nis it!";
let n = /^is/m.test(text);         // true
```

Regular Expressions

Patterns	Meaning
[abc]	Any characters within []
[0-9]	Any digits within the range
(x|y)	Either x or y.
\s	Whitespace
\d	Digits

Patterns	Meaning
^	Begins with
$	Ends with
p*	0 or more occurrence
p+	1 or more occurrence
p?	At most 1 occurrence

```
let text = "Learn JavaScript and java";
let n = text.match(/(J|j)ava/g);            // returns JavaScript, java

let text = " Learn java and ECMAScript 15 or higher";
let n = text.match(/[1-9]+/);       // 15

let text = "New York\nCalifornia\nTexas";
let n = text.match(/^[Nc]/img);            // N,C
```

Additional References

- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_expressions

For You to Do

- Modify the JavaScript *hideEmail()* in *quiz/hideEmail-q.js* such that it anonymizes the username(local part) of a valid email address, that is, replace the local part with * characters. An email address is valid if
  - the local part contains alphanumeric characters or ‘_’,
  - uses ’@’ to separate the local part with the domain name, and
  - the domain name ends with *northeastern.edu*.
- Test the function by loading *quiz/hideEmail-q.html*

Arrays

- An array holds a list of heterogenous values. We use the syntax `const array_name = [item1, item2, ...];` to create an array.

```
const year = ["2022", "2021", 2020];
```

- Array indices start at 0; [0] is the 1st element, [1] the 2nd, …
- For a const array, the array variable cannot be re-declared but its elements can be changed.

```
const languages = ["Javascript", "Python", "Ruby"];;
languages[0] = "Rust";
```

- An array is an *Object* not a primitive type.

Useful Array Methods

- Construct a delimited string from an array using `join()`.

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
document.getElementById("demo").innerHTML = fruits.join(" * ");
// Result : Banana * Orange * Apple * Mango
```

- Remove an element from an array using `pop()`.

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
let fruit = fruits.pop();    // fruit = "Mango"
```

- Append an element to end of array using `push()`.

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
let length = fruits.push("Kiwi");
```

- Reverse an array using `reverse()`.

```
const fruits = ["Orange", "Apple", "Mango"];
fruits.reverse();    //fruits = ["Mango", "Apple", "Orange"]
```

Useful Array Methods

- Popping an element from the start of array using `shift()`.

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.shift();  // returns "Banana"
document.getElementById("demo").innerHTML = fruits;
// Result : ["Orange", "Apple", "Mango"]
```

- Add an element to start of array using `unshift()`.

```
const fruits = ["Orange", "Apple", "Mango"];
fruits.unshift("Apple"); // returns new length: 4
document.getElementById("demo").innerHTML = fruits;
// Result : ["Apple ", "Orange", "Apple", "Mango"]
```

Sorting Arrays

- Sorting an array is as easy as calling `sort()` on the array.
- But we may want to define the sorting order.
- We need to invoke `sort()` with a compare function as argument.
- The compare function takes two arguments `a` and `b`.
  - If result of comparison is negative `a` is placed before `b`.
  - If result of comparison is positive `b` is placed before `a`.

```
const points = [40, 100, 1, 5, 25, 10];
points.sort(function(a, b){return a - b});  // points = [1,5,10,25,40,100]
```

```
const points = [40, 100, 1, 5, 25, 10];
points.sort(function(a, b){return b - a});  // points = [100,40,25,10,5,1]
```

Sorting Arrays

- Defining compare functions are necessary when we are sorting object arrays. E.g.,

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

- Sort this array by year.

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

// Result: Saab 2001, BMW 2010, Volvo 2016

For You to Do

- *quiz/array1-q.html* is a page with a search box. When a user enters text in the box, the text gets added to a list and the list is displayed, sorted by the length of the text. Further, if the list has 5 or more items at any point, then the first item added to the list will be removed.
    - Complete the script in *quiz/scripts/array1-q.js* to exhibit the above behavior.

Iterating Arrays

- `forEach()` calls a **function** on every element in an array.
  - The function takes *value*, *index*, and the *array* as arguments respectively.
  - The function can also be defined with one argument as in most cases only *value* is relevant.
- Let’s see an example in *learn-js/scripts/forEach.js*

Iterating Arrays

- `map()` applies a function to every element in an array and returns a new array (**shallow copy**).
  - The function takes *value*, *index*, and the *array* as arguments respectively.
  - The function can also be defined with one argument as in most cases only *value* is relevant.
- Let’s see an example in *learn-js/scripts/map.js*

Iterating Arrays

- `filter()` applies a boolean function to every element in an array and returns a new array (**shallow copy**) with elements for which the function returns true.
  - The function takes *value*, *index*, and the *array* as arguments respectively.
  - The function can also be defined with one argument as in most cases only *value* is relevant.
- Let’s see an example in *learn-js/scripts/filter*

Additional Array Reference

- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array

For You to Do

- The file *quiz/scripts/map-deep.js* creates a copy of a given matrix by sorting each row in increasing order. But changing the given array also changes the copy, and vice-versa.
  - Run the script and test this claim.
  - Why does it happen?
  - Can you fix it?

Handling Errors

- Runtime errors may happen when executing JavaScript.
- We can handle errors and provide meaningful feedback.
  - Encapsulate any code that may cause errors in **try** and handle them in **catch**.
  - Define a **finally** block to execute code regardless of errors (e.g., close a file).
  - Use **throw** to define custom errors

```
try {
  Block of code to try
}
catch(err) {
  Block of code to handle errors
}
```

Common Runtime Error

- ReferenceError
  - Reference variable before declarartion.
- SyntaxError
  - Using undefined constructs.
- TypeError
  - Expression with incompatible types.
- For more information on Error see https://developer.mozilla.org/en-US/docs/web/javascript/reference/global_objects/error
- Let's see an example in *learn-js/exception-eg.html*

For You to Do

- Complete the function `calculateSquareRoots()` *quiz/exception.html* such that it console logs the array of numbers and the square root of each number in the array. Also, console log an error message if a runtime exception occurs due to square root on a negative number.

Classes

- A JavaScript class is a template for JavaScript objects.
  - has a *constructor* to initialize fields and methods that can change the state of the object; called when an instance of class is created.
  - Fields in a class can be made *private* by declaring them at the beginning of a class with ‘#’.
- Classes can inherit from a parent class using **extends**
  - All the methods of the parent class are available to the child class.
  - Child class must user **super()** in derived constructor.

Classes

- To read and write class fields, JavaScript encourages defining *getter* and *setter* methods.

```
class Person {
  constructor(name, year) {
    this._name = name;
    this._birthYear = year;
  }
  get name() {
    return this._name;
  }
  set name(n) {
    this._name = n;
  }
}
```

```
let p = new Person("Gal", 1984);
p.name = “Hal”;
document.getElementById("demo").innerHTML = p.name;
```

Class Example

- Let’s look at an example in *learn-js/scripts/stack-q.js*
    - It has a parent and a child class implementing a stack like structure.

For You to Do

- Define the *persons* property in *quiz/scripts/stack-q1.js* such that it can only be accessed by get and set methods. Also, define the get and set methods.

Modules

- JavaScript programs can be divided into modules.
- Module features can be *exported* so other programs can use them.
- This is done by placing the **export** before each item that needs to be exported.

```
export const name = 'square';

export function draw(ctx, length, x, y, color) {
  ctx.fillStyle = color;
  ctx.fillRect(x, y, length, length);

  return {
    length: length,
    x: x,
    y: y,
    color: color
  };
}
```

Modules

- An alternative way to export features is to use a *single export* at the end of the module.

```
export { name, draw };
```

- Other scripts can *import* exported features from a module.

```
import { name, draw } from ‘/path/to/module.js';
```

- If there is only one feature to export from a module, we can use the **export default** keyword.

```
export default class Square {
   constructor(ctx, listId, length, x, y, color) { ... }
   draw() { ... } ...
}
```

```
// in the script
import Square from ‘/path/to/module.js';
```

Modules

- JavaScript modules can be applied to HTML using the *type="module"* attribute.

```
<script type="module" src="main.js"></script>
```

- Importing modules with a *file:// URL* will lead to **C**ross **O**rigin **R**esource **S**haring (CORS) Error due to security requirements.
- Modules need to be imported through a server.

Additional References

- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Modules
- https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS