Loops

- If we are interested in both the keys and the values of an object's properties, we can use for/of with Object.entries().
- Example:

```
let pairs = "";

for(let [k, v] of Object.entries(o)) {

        pairs += k + v;

}
```

- Object.entries() returns an array of arrays, where each inner array represents a key/value pair for one property of the object.
- Strings are iterable character-by-character.

Loops

- Example

```
let frequency = {};

for(let letter of "mississippi") {

        if (frequency[letter]) {

                frequency[letter]++;

        }

        else {

                frequency[letter] = 1;

        }

}
```

How To Insert JavaScript in HTML

- In HTML, JavaScript code is inserted between <script> and </script> tags.
- Old JavaScript examples may use a type attribute: <script type="text/javascript">. The type attribute is not required.
- Scripts can be placed in the <body>, or in the <head> section of an HTML page, or in both.
- We can place any number of scripts in an HTML document.
- Scripts can also be placed in external files as well.
- External scripts are practical when the same code is used in many different web pages. JavaScript files have the file extension .js.
- To use an external script, put the name of the script file in the src (source) attribute of a <script> tag. Example: `<script src="demo.js"></script>`

for/in Loops

- A for/in loop looks a lot like a for/of loop, with the of keyword changed to in.
- While a for/of loop requires an iterable object after the of, a for/in loop works with any object after the in.
- The for/in statement loops through the property names of a specified object.
- Syntax: for (variable in object)

  statement

- To execute a for/in statement, the JavaScript interpreter first evaluates the object expression. If it evaluates to null or undefined, the interpreter skips the loop and moves on to the next statement.
- The variable in the for/in loop may be an arbitrary expression, as long as it evaluates to something suitable for the left side of an assignment.

for/in Loops

- Example:

```
let o = { x: 1, y: 2, z: 3 };

let a = [], i = 0;

for(a[i++] in o) /* empty */;
```

- The for/in loop does not actually enumerate all properties of an object.
- It does not enumerate properties whose names are symbols. And of the properties whose names are strings.
- Example:

```
for(let i in a) console.log(i);
```

Jump & Labelled Statements

- Break and Continue statement
- Any statement may be labeled by preceding it with an identifier and a colon:
  - identifier: statement
- break and continue are the only JavaScript statements that use statement labels.
- The identifier we use to label a statement can be any legal JavaScript identifier that is not a reserved word.
- The namespace for labels is different than the namespace for variables and functions, so we can use the same identifier as a statement label and as a variable or function name.
- Labeled statements may themselves be labeled.
- Example: break labelname; continue labelname;

JavaScript Output

- JavaScript can "display" data in different ways:
  - Writing into an HTML element, using innerHTML.
  - Writing into the HTML output using document.write().
  - Writing into an alert box, using window.alert().
  - Writing into the browser console, using console.log().
- Using document.write() after an HTML document is loaded, will delete all existing HTML. The document.write() method should only be used for testing.
- In JavaScript, the window object is the global scope object, that means that variables, properties, and methods by default belong to the window object.
- This also means that specifying the window keyword is optional.
- JavaScript does not have any print object or print methods. We cannot access output devices from JavaScript.
- The only exception is that we can call the window.print() method in the browser to print the content of the current window.

Defining a function

- A function definition is a regular binding where the value of the binding is a function.
- Example:

```
const square = function(x) {
return x * x;
};
console.log(square(12));
```

- A function is created with an expression that starts with the keyword function.
- Functions have a set of parameters (in this case, only x) and a body, which contains the statements that are to be executed when the function is called.
- Example:

```
const power = function(base, exponent) {
let result = 1;
for (let count = 0; count < exponent; count++) {
result *= base;
}
return result;
};
console.log(power(2, 10));
```

Bindings and Scopes

- Each binding has a scope, which is the part of the program in which the binding is visible.
- For bindings defined outside of any function or block, the scope is the whole program, these are called global.
- Bindings created for function parameters or declared inside a function can be referenced only in that function, so they are known as local bindings.
- Example:

```
let x = 10;
if (true) {
let y = 20;
var z = 30;
console.log(x + y + z);
// → 60
}
// y is not visible here
```

Functions as Values

- A function binding usually simply acts as a name for a specific piece of the program.
- Such a binding is defined once and never changed.
- A function value can do all the things that other values can do—we can use it in arbitrary expressions, not just call it.
- It is possible to store a function value in a new binding, pass it as an argument to a function, and so on.
- Example:

```
let launchMissiles = function() {
missileSystem.launch("now");
};
```

Arrow Functions

- Instead of the function keyword, it uses an arrow (=>) made up of an equal sign and a greater-than character.
- Example:

```
const power = (base, exponent) => {
let result = 1;
for (let count = 0; count < exponent;
count++) {
result *= base;
}
return result;
};
```

- The arrow comes after the list of parameters and is followed by the function's body.
- When there is only one parameter name, we can omit the parentheses around the parameter list.
- If the body is a single expression, rather than a block in braces, that expression will be returned from the function.

Arrow Functions

- Example:

```
const square1 = (x) => { return x * x; };
const square2 = x => x * x;
```

- The above two definitions of square do the same thing.
- When an arrow function has no parameters at all, its parameter list is just an empty set of parentheses.
- Example:

```
const h = () => {
console.log(“JavaScript");
};
```

Optional Arguments

- Example:

```
function power(base, exponent = 2) {
let result = 1;
for (let count = 0; count < exponent;
count++) {
result *= base;
}
return result;
}
```

- If we write an “=“ operator after a parameter, followed by an expression, the value of that expression will replace the argument when it is not given.

Closure

- The ability to treat functions as values, combined with the fact that local bindings are re-created every time a function is called, brings up an interesting question.
- What happens to local bindings when the function call that created them is no longer active?
- Example:

```
function wrapValue(n) {
let local = n;
return () => local;
}
let wrap1 = wrapValue(1);
let wrap2 = wrapValue(2);
console.log(wrap1());
console.log(wrap2());
```

- This feature—being able to reference a specific instance of a local binding in an enclosing scope—is called closure.
- A function that references bindings from local scopes around it is called a closure.

Closure

- Example:

```
function multiplier(factor) {
return number => number * factor;
}
let twice = multiplier(2);
console.log(twice(5));
```

Data sets

- JavaScript provides a data type specifically for storing sequences of values. It is called an array and is written as a list of values between square brackets, separated by commas.
- Example: let list = [2, 3, 5, 7, 11];
- A pair of square brackets will look up the element in the left-hand expression that corresponds to the index given by the expression in the brackets.
- The first index of an array is zero, not one.
- Both string and array values contain, in addition to the length property, a number of properties that hold function values.
- The push method adds values to the end of an array, and the pop method does the opposite, removing the last value in the array and returning it.

Data sets

- Example:

```
let sequence = [1, 2, 3];
sequence.push(4);
sequence.push(5);
console.log(sequence);
// → [1, 2, 3, 4, 5]
console.log(sequence.pop());
// → 5
console.log(sequence);
```

Objects

- Values of the type object are arbitrary collections of properties. One way to create an object is by using braces as an expression.
- Example:

```
let day1 = {
serial: false,
events: ["work", "touched tree", "pizza", "running"]
};
console.log(day1.serial);
day1.wolf = false;
console.log(day1.wolf);
let descriptions = {
work: "Went to work",
"touched tree": "Touched a tree"
};
```

- Inside the braces, there is a list of properties separated by commas. Each property has a name followed by a colon and a value.
- This means that braces have two meanings in JavaScript. At the start of a statement, they start a block of statements. In any other position, they describe an object.

Objects

- It is possible to assign a value to a property expression with the = operator. This will replace the property's value if it already existed or create a new property on the object if it didn't.
- The delete is a unary operator that, when applied to an object property, will remove the named property from the object.
- Example: delete day1.serial
- The binary in operator, when applied to a string and an object, tells us whether that object has a property with that name.
- Example: console.log("events" in day1)
- To find out what properties an object has, we can use the Object.keys function.
- Example: console.log(Object.keys({x: 0, y: 0, z: 2}));

Objects

- There's an Object.assign function that copies all properties from one object into another.
- Example:

```
let objectA = {a: 1, b: 2};
Object.assign(objectA, {b: 3, c: 4});
console.log(objectA);
```

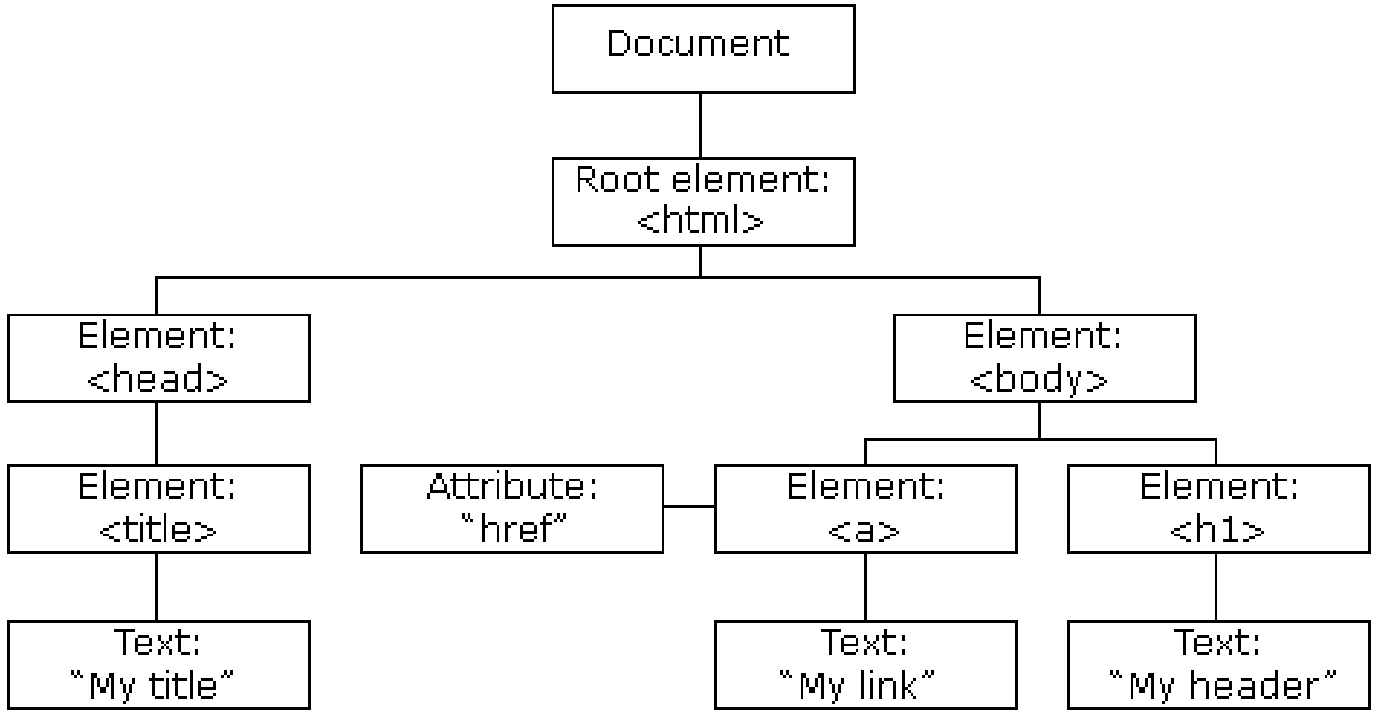
- Example:

```
let object1 = {value: 10};
let object2 = object1;
let object3 = {value: 10};
console.log(object1 == object2);
console.log(object1 == object3);
object1.value = 15;
console.log(object2.value);
console.log(object3.value);
```

- Though a const binding to an object can itself not be changed and will continue to point at the same object, the contents of that object might change.
- Example: const score = {visitors: 0, home: 0};

The HTML DOM (Document Object Model)

- When a web page is loaded, the browser creates a Document Object Model of the page.
- The HTML DOM model is constructed as a tree of Objects.



The HTML DOM (Document Object Model)

- With the object model, JavaScript gets all the power it needs to create dynamic HTML:
  - JavaScript can change all the HTML elements in the page
  - JavaScript can change all the HTML attributes in the page
  - JavaScript can change all the CSS styles in the page
  - JavaScript can remove existing HTML elements and attributes
  - JavaScript can add new HTML elements and attributes
  - JavaScript can react to all existing HTML events in the page
  - JavaScript can create new HTML events in the page

What is the HTML DOM?

- The HTML DOM is a standard object model and programming interface for HTML. It defines:
  - The HTML elements as objects
  - The properties of all HTML elements
  - The methods to access all HTML elements
  - The events for all HTML elements
- In other words: The HTML DOM is a standard for how to get, change, add, or delete HTML elements.

JavaScript - HTML DOM Methods

- HTML DOM methods are actions we can perform.
- HTML DOM properties are values (of HTML Elements) that we can set or change.
- The HTML DOM can be accessed with JavaScript (and with other programming languages).
- A property is a value that we can get or set (like changing the content of an HTML element).
- A method is an action we can do (like add or deleting an HTML element).
- Example: document.getElementById("demo").innerHTML = "Hello World!";
- The most common way to access an HTML element is to use the id of the element. The getElementById method used id="demo" to find the element.
- The innerHTML property is useful for getting or replacing the content of HTML elements.

JavaScript HTML DOM Document

- The HTML DOM document object is the owner of all other objects in our web page.
- The document object represents our web page.
- If we want to access any element in an HTML page, we always start with accessing the document object.
- Finding HTML Elements
  - document.getElementById(id): Find an element by element id
  - document.getElementsByTagName(name): Find elements by tag name
  - document.getElementsByClassName(name): Find elements by class name

JavaScript HTML DOM Document

Changing HTML Elements

- element.innerHTML = new html content [Change the inner HTML of an element]
- element.attribute = new value [Change the attribute value of an HTML element]
- element.style.property = new style [Change the style of an HTML element]
- element.setAttribute(attribute, value) [Change the attribute value of an HTML element]

Adding and Deleting Elements

- document.createElement(element): Create an HTML element
- document.removeChild(element): Remove an HTML element
- document.appendChild(element): Add an HTML element
- document.replaceChild(new, old): Replace an HTML element
- document.write(text): Write into the HTML output stream

Adding Events Handlers

- document.getElementById(id).onclick = function(){code}: Adding event handler code to an onclick event

JavaScript HTML DOM Elements

- Finding HTML Element by Id

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript HTML DOM</h2>

<p id="intro">Finding HTML Elements by Id</p>
<p>This example demonstrates the <b>getElementsById</b> method.</p>

<p id="demo"></p>

<script>
const element = document.getElementById("intro");

document.getElementById("demo").innerHTML =
"The text from the intro paragraph is: " + element.innerHTML;

</script>

</body>
</html>
```

JavaScript HTML DOM Elements

- Finding HTML Elements by Tag Name

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript HTML DOM</h2>

<p>Finding HTML Elements by Tag Name.</p>
<p>This example demonstrates the <b>getElementsByTagName</b> method.</p>

<p id="demo"></p>

<script>
const element = document.getElementsByTagName("p");

document.getElementById("demo").innerHTML = 'The text in first paragraph (index 0) is: ' + element[0].innerHTML;

</script>

</body>
</html>
```

JavaScript HTML DOM Elements

- Finding HTML Elements by Class Name

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript HTML DOM</h2>

<p>Finding HTML Elements by Class Name.</p>
<p class="intro">Hello World!</p>
<p class="intro">This example demonstrates the <b>getElementsByClassName</b> method.</p>

<p id="demo"></p>

<script>
const x = document.getElementsByClassName("intro");
document.getElementById("demo").innerHTML =
'The first paragraph (index 0) with class="intro" is: ' + x[0].innerHTML;
</script>

</body>
</html>
```

JavaScript HTML DOM Document

- Finding HTML Elements by CSS Selectors
  - If we want to find all HTML elements that match a specified CSS selector (id, class names, types, attributes, values of attributes, etc), use the querySelectorAll() method.

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript HTML DOM</h2>

<p>Finding HTML Elements by Query Selector</p>
<p class="intro">Hello World!.</p>
<p class="intro">This example demonstrates the <b>querySelectorAll</b> method.</p>

<p id="demo"></p>

<script>
const x = document.querySelectorAll("p.intro");
document.getElementById("demo").innerHTML = 'The first paragraph (index 0) with class="intro" is: ' + x[0].innerHTML;
</script>

</body>
</html>
```

JavaScript Form Validation

- It is important to validate the form submitted by the user because it can have inappropriate values. So, validation is must to authenticate user.
- Example:

```
<script>
function validateform(){
var name=document.myform.name.value;
var password=document.myform.password.value;

if (name==null || name==""){
  alert("Name can't be blank");
  return false;
}else if(password.length<6){
  alert("Password must be at least 6 characters long.");
  return false;
  }
}
</script>
<body>
<form name="myform" method="post" action="abc.jsp" onsubmit="return validateform()" >
Name: <input type="text" name="name"><br/>
Password: <input type="password" name="password"><br/>
<input type="submit" value="register">
</form>
```

JavaScript Form Validation

- JavaScript Retype Password Validation
- Example:

```
<script type="text/javascript">
function matchpass(){
var firstpassword=document.f1.password.value;
var secondpassword=document.f1.password2.value;

if(firstpassword==secondpassword){
return true;
}
else{
alert("password must be same!");
return false;
}
}
</script>

<form name="f1" action="register.jsp" onsubmit="return matchpass()">
Password:<input type="password" name="password" /><br/>
Re-enter Password:<input type="password" name="password2"/><br/>
<input type="submit">
</form>
```

JavaScript HTML DOM Document

- JavaScript Number Validation
- Example:

```
<script>
function validate(){
var num=document.myform.num.value;
if (isNaN(num)){
  document.getElementById("numloc").innerHTML="Enter Numeric value only";
  return false;
}else{
  return true;
  }
}
</script>
<form name="myform" onsubmit="return validate()" >
Number: <input type="text" name="num"><span id="numloc"></span><br/>
<input type="submit" value="submit">
</form>
```

Window prompt()

- The prompt() method displays a dialog box that prompts the user for input.
- The prompt() method returns the input value as string if the user clicks "OK", otherwise it returns null.
- Syntax: prompt(text, defaultText)
- Example:

```
let person = prompt("Please enter your name", "Harry Potter");

if (person != null) {
  document.getElementById("demo").innerHTML =  "Hello " + person + "! How are you today?";
}
```

JavaScript Cookies

- Cookies are data, stored in small text files, on your computer.
- When a web server has sent a web page to a browser, the connection is shut down, and the server forgets everything about the user.
- Cookies were invented to solve the problem "how to remember information about the user":
  - When a user visits a web page, his/her name can be stored in a cookie.
  - Next time the user visits the page, the cookie "remembers" his/her name.
- Cookies are saved in name-value pairs like: username = John Doe
- When a browser requests a web page from a server, cookies belonging to the page are added to the request. This way the server gets the necessary data to "remember" information about users.

JavaScript Cookies

- Create a Cookie with JavaScript
  - A cookie can be created like: document.cookie = "username=John Doe";
  - We can also add an expiry date (in UTC time). By default, the cookie is deleted when the browser is closed.
  - Example: document.cookie = "username=John Doe; expires=Thu, 18 Dec 2013 12:00:00 UTC";
- Read a Cookie with JavaScript
  - Cookies can be read like: let x = document.cookie;
  - document.cookie will return all cookies in one string much like: cookie1=value; cookie2=value; cookie3=value;
- Change a Cookie with JavaScript
  - We can change a cookie the same way as you create it.
  - document.cookie = "username=John Smith; expires=Thu, 18 Dec 2013 12:00:00 UTC;";
  - The old cookie is overwritten.

JavaScript Cookies

- Delete a Cookie with JavaScript
  - We don't have to specify a cookie value when we delete a cookie.
  - We just set the expires parameter to a past date.
  - Example: `document.cookie = "username=; expires=Thu, 01 Jan 1970 00:00:00;";`
- JavaScript Cookie Example
  - We will create a cookie that stores the name of a visitor.
  - The first time a visitor arrives to the web page, he/she will be asked to fill in his/her name. The name is then stored in a cookie.
  - The next time the visitor arrives at the same page, he/she will get a welcome message.
  - We will create 3 JavaScript functions:
    i. A function to set a cookie value
    ii. A function to get a cookie value
    iii. A function to check a cookie value

JavaScript Cookie Example

- A Function to Set a Cookie
- Example:

```
function setCookie(cname, cvalue, exdays) {
  const d = new Date();
  d.setTime(d.getTime() + (exdays*24*60*60*1000));
  let expires = "expires="+ d.toUTCString();
  document.cookie = cname + "=" + cvalue + ";" + expires + ";path=/";
}
```

- The value returned by toUTCString() is a string in the form Www, dd Mmm yyyy hh:mm:ss GMT

JavaScript Cookie Example

- A Function to Get a Cookie
- Example:

```
function getCookie(cname) {
  let name = cname + "=";
  let decodedCookie = decodeURIComponent(document.cookie);
  let ca = decodedCookie.split(';');
  for(let i = 0; i <ca.length; i++) {
    let c = ca[i];
    while (c.charAt(0) == ' ') {
      c = c.substring(1);
    }
    if (c.indexOf(name) == 0) {
      return c.substring(name.length, c.length);
    }
  }
  return "";
}
```

JavaScript Cookie Example

- A Function to Check a Cookie
- Example:

```
function checkCookie() {
  let username = getCookie("username");
  if (username != "") {
   alert("Welcome again " + username);
  } else {
    username = prompt("Please enter your name:", "");
    if (username != "" && username != null) {
      setCookie("username", username, 365);
    }
  }
}
```

- <body onload="checkCookie()"></body>