JavaScript and Node.js

Web Programming

Suriya Natsupakpong, PhD

Institute of Field Robotics (FIBO)

King Mongkut’s University of Technology Thonburi (KMUTT)

JavaScript

- It is released in 1995, JavaScript used to add interactive elements to web pages: button clicks, hover states, form validation, etc.
- Today, JavaScript got more robust with DHTML and AJAX.
- With Node.js, JavaScript has become a real software language that's used to build fullstack applications. JavaScript is everywhere.
- European Computer Manufacturers Association (ECMA) committee is to manage and prioritize these proposals to decide what's included in each spec.

JavaScript

- JavaScript code is embedded in an HTML web page and is interpreted by the web browser.
- When JavaScript code is embedded in an HTML document, it needs to be contained, or encapsulated, in a script element.
- The JavaScript is typed between the opening <script> tag and the closing </script> tag.
- The impact on our scripts is that they will execute wherever they are located in the document.
- The script element can be coded within the head element or the body element.

```
<script>
      // code here
</script>


<script scr="app.js"></script>
```

JavaScript vs Java

JavaScript	Java
JavaScript is used for front-end web development (for example field level validation in a HTML form).	Java is used as a back-end language within a web environment.
Interpreted (not compiled) by the client.	Compiled bytecodes downloaded from the server, executed on the client.
Object-oriented. No distinction between types of objects. Inheritance is through the prototype mechanism, and properties and methods can be added to any object dynamically.	Class-based. Objects are divided into classes and instances with all inheritance through the class hierarchy. Classes and instances cannot have properties or methods added dynamically.
Variable data types are not declared (loose typing).	Variable data types must be declared as Java maintains strong type checking.
Cannot automatically write to hard disk.	Cannot automatically write to hard disk.

https://www.w3resource.com/javascript/javascript.php

JavaScript : Fundamental Rules

- Case-sensitivity
- Semicolons
- White space
- Comments
- JavaScript is a “weakly typed” language.
- Primitive types:
  - Numbers : a set of all possible number values, “not a number” (NaN), “positive infinity” (Infinity), “negative infinity” (-Infinity)
  - Strings : Any set of characters—letters, numbers, symbols, and so on—between a set of double or single quotes.
  - undefined : the type for anything that isn’t predefined by JavaScript.
  - null : non-value, something that has been defined, but has no inherent value.
  - Booleans : true and false.

```
// comment this line

/*
   multi-line comment
*/
```

JavaScript Syntax

```javascript
// Two slashes start single-line comments
let x; // declaring a variable
x = 3 + y; // assigning a value to the variable `x`
foo(x, y); // calling function `foo` with parameters `x` and `y`
obj.bar(3); // calling method `bar` of object `obj`
// A conditional statement
if (x === 0) {
  // Is `x` equal to zero?
  x = 123;
}
// Defining function `baz` with parameters `a` and `b`
function baz(a, b) {
  return a + b;
}
```

```javascript
let x;
if (y >= 0) {
  x = y;
} else {
  x = -y;
}
```

```javascript
let x = y >= 0 ? y : -y;
```

```javascript
myFunction(y >= 0 ? y : -y);
```

A single equals sign (=) is used to assign a value to a variable.

A triple equals sign (===) is used to compare two values

JavaScript Reserved Words

arguments	break	case	catch	class
const	continue	debugger	default	delete
do	else	enum	export	extends
False	finally	for	function	if
implements	import	in	instanceof	interface
let	new	null	package	private
protected	public	return	static	super
Switch	this	throw	true	try
typeof	var	void	while	

Infinity
NaN
undefined

Operators

- Addtition operator (+)
- Subtraction operator (-)
- Division operator (/)
- Remainder operator (%)
- Multiplication operator (*)
- Exponentiation operator (**)

Comparison Operators

- < means "less than"
- <= means "minus than, or equal to"
- > means "greater than"
- >= means "greater than, or equal to“
- === checks for equality
- !== checks for inequality
- == checks for equality
- != checks for inequality

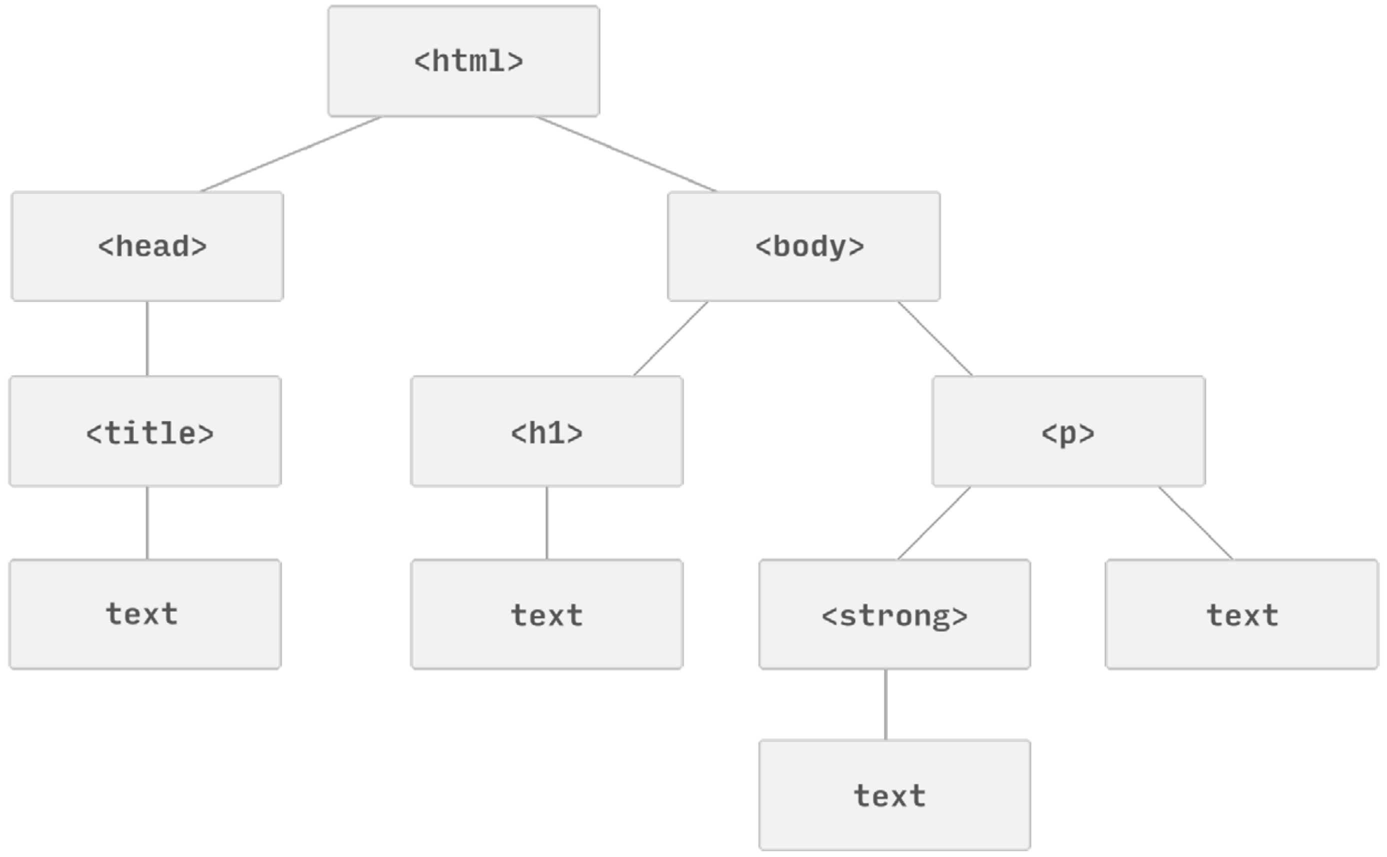
Equality Operators: === Versus ==

Strict equality (===) and strict inequality (!==) consider only values that have the same type to be equal. Normal (or “lenient”) equality (==) and inequality (!=) try to convert values of different types before comparing them as with strict (in)equality.

Assignment Operators

x = value	Assigns to a variable x that has previously been declared
let x = value	Combines a variable declaration with an assignment
obj.propKey = value	Sets a property
obj['propKey'] = value	Sets a property
arr[index] = value	Sets an array element
myvar op= value	myvar = myvar op value

Document Object Model (DOM)



Object in JavaScirpt

Document object model (DOM)

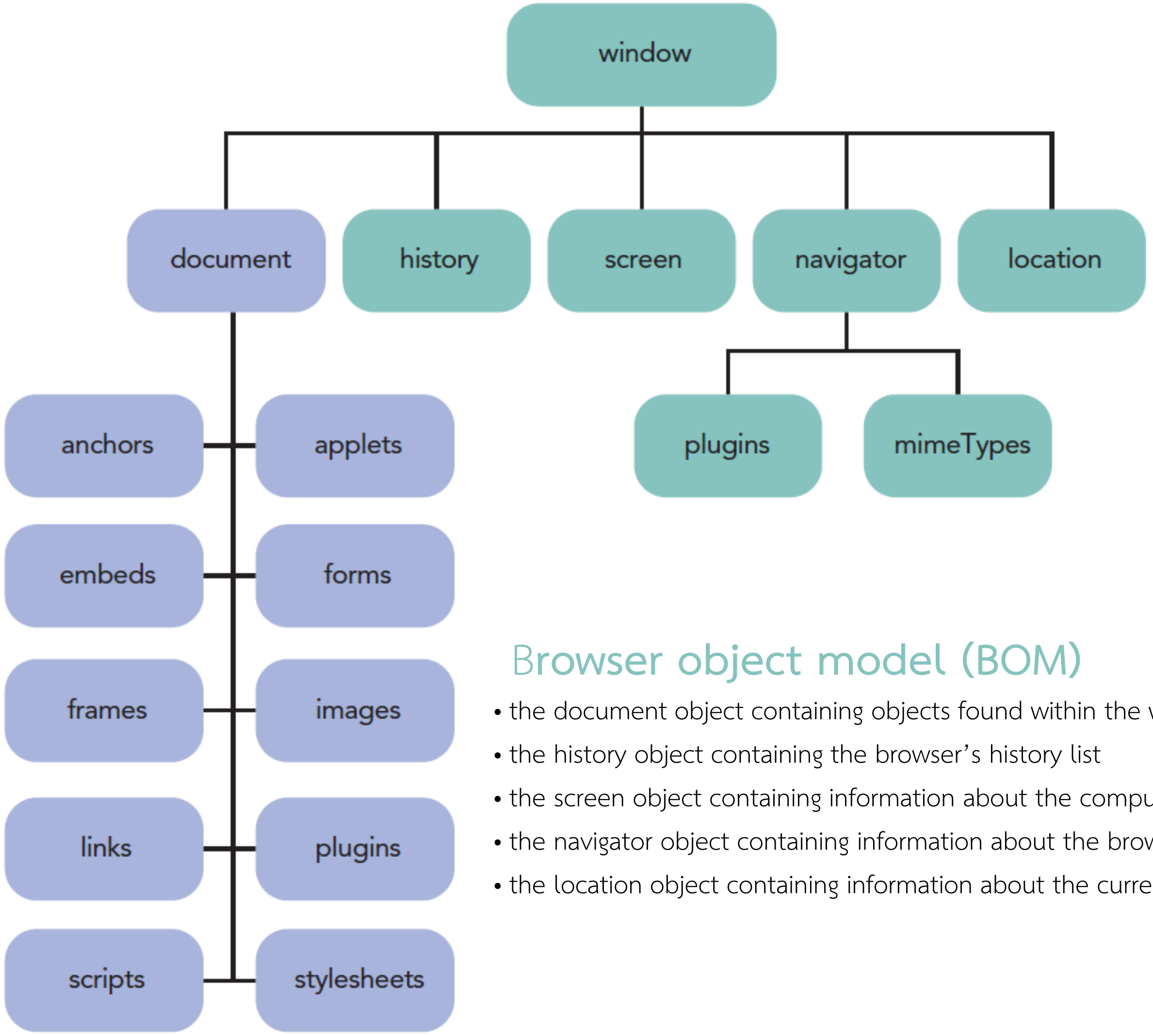
Object Collection	References
document.anchors	All elements marked with the `<a>` tag
document.applets	All `applet` elements
document.embeds	All `embed` elements
document.forms	All web forms
document.frames	All `frame` elements
document.images	All inline images
document.links	All hypertext links
document.plugins	All plug-ins supported by the browser
document.scripts	All `script` elements
document.styleSheets	All `stylesheet` elements

document.getElementsByTagName(tag)

document.getElementsByClassName(class)

document.getElementsByName(name)

document.getElementById(id)



Browser object model (BOM)

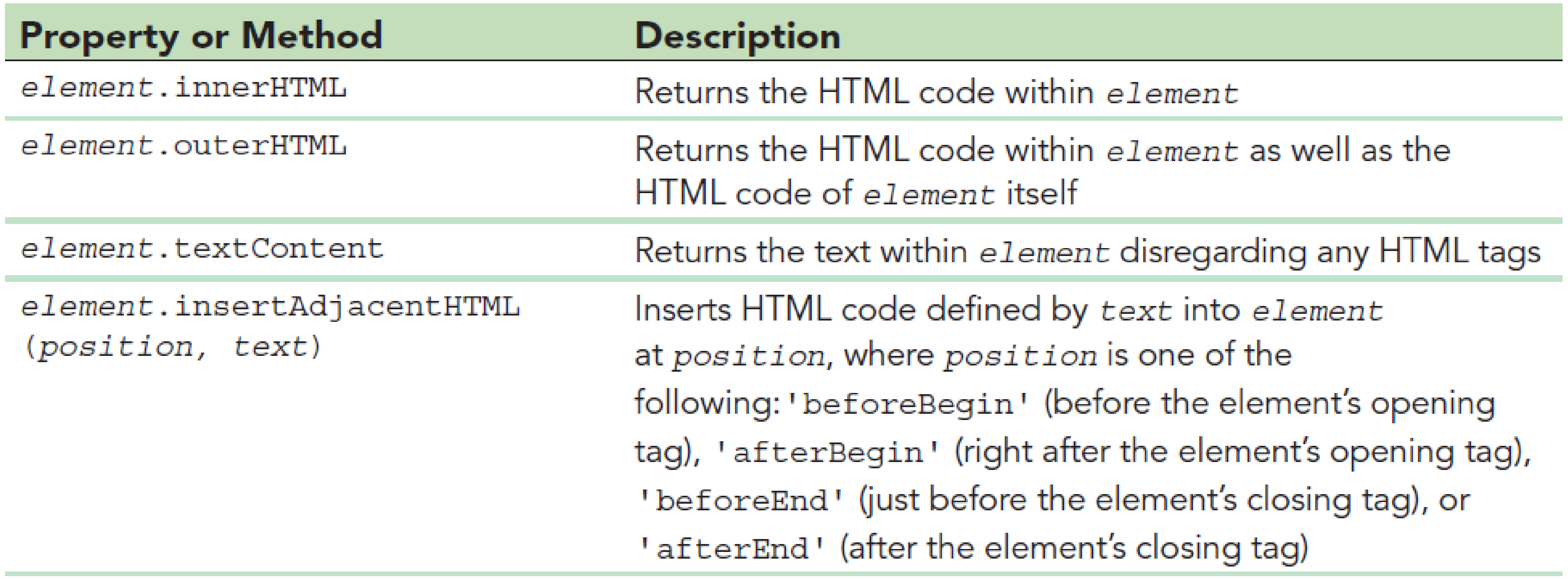
- the document object containing objects found within the web page document
- the history object containing the browser's history list
- the screen object containing information about the computer screen
- the navigator object containing information about the browser application
- the location object containing information about the current URL

Document Object Model (DOM)

```
document.write("text to be written to the document");
```

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>JavaScript Practice</title>
    <meta charset="utf-8" />
  </head>
  <body>
    <h1>Using JavaScript</h1>
    <script>
      document.write("<p>Using document.write to add text</p>");
      document.write("<h2>Notice that we can add HTML tags too!</h2>");
    </script>
    <h3>
      This document was last modified on:
      <script>
        document.write(document.lastModified);
      </script>
    </h3>
  </body>
</html>
```

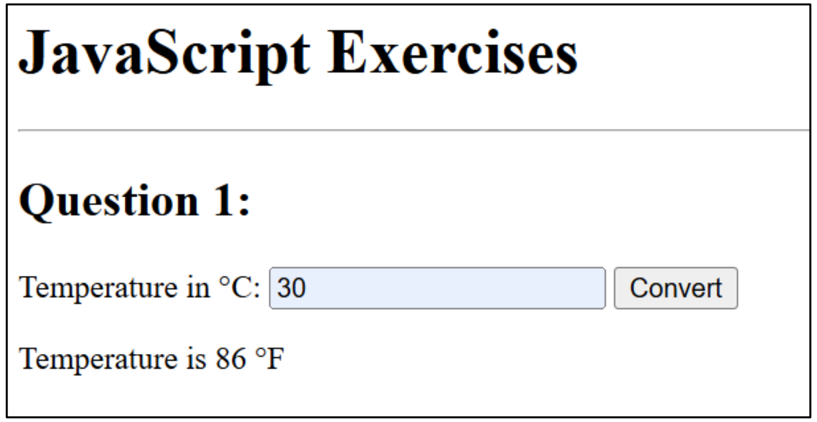
Properties and Method to Insert Content



Property or Method	Description
element.innerHTML	Returns the HTML code within *element*
element.outerHTML	Returns the HTML code within *element* as well as the HTML code of *element* itself
element.textContent	Returns the text within *element* disregarding any HTML tags
element.insertAdjacentHTML (*position*, *text*)	Inserts HTML code defined by *text* into *element* at *position*, where *position* is one of the following: 'beforeBegin' (before the element's opening tag), 'afterBegin' (right after the element's opening tag), 'beforeEnd' (just before the element's closing tag), or 'afterEnd' (after the element's closing tag)

Example

```html
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <script src="exercise.js"></script>
    <title>Web Programming Exercises</title>
  </head>
  <body>
    <h1>JavaScript Exercises</h1>
    <hr />
    <h2>Question 1:</h2>
    <form id="fun1">
      Temperature in °C: <input type="text" id="celsius"  />
      <input type="button" onclick="CToF()" value="Convert" />
    </form>
    <p>Temperature is <span id="ans1"></span></p>
  </body>
</html>
```



JavaScript Exercises

Question 1:

Temperature in °C: 30 Convert

Temperature is 86 °F

```javascript
function CToF() {
  let tempC = document.getElementById("celsius").value;
  document.getElementById("ans1").innerHTML = (tempC * 9) / 5 + 32 + " °F";
}
```

Conditionals

```
if (true) {
  //do something
}
```

```
if (true) {
  //do something
} else {
  //do something else
}
```

```
if (a === true) {
  //do something
} else if (b === true) {
  //do something else
} else {
  //fallback
}
```

Loops

```
const list = ["a", "b", "c"];
let i = 0;
while (i < list.length) {
  console.log(list[i]); //value
  console.log(i); //index
  i = i + 1;
}
```

```
const list = ["a", "b", "c"];
let i = 0;
do {
  console.log(list[i]); //value
  console.log(i); //index
  i = i + 1;
} while (i < list.length);
```

```
const list = ["a", "b", "c"];
for (let i = 0; i < list.length; i++) {
  console.log(list[i]); //value
  console.log(i); //index
}
```

```
const list = ["a", "b", "c"];
for (const value of list) {
  console.log(value); //value
}
```

```
while (true) {
  if (somethingIsTrue) break;
}
```

```
while (true) {
  if (somethingIsTrue) continue;
  //do something else
}
```

Collecting Variable Values Using a Prompt

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>JavaScript Practice</title>
    <meta charset="utf-8" />
  </head>
  <body>
    <h1>Using JavaScript</h1>
    <script>
      var userName;
      userName = prompt("Please enter your name");
      document.write("<h2>Hello " + userName + "</h2>");
      var userColor;
      userColor = prompt("Please type the color name blue or red");
      document.bgColor = userColor;
    </script>
  </body>
</html>
```

Control Structure Example

```html
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>JavaScript Practice</title>
    <meta charset="utf-8" />
  </head>
  <body>
    <h1>Using JavaScript</h1>
    <script>
      var quantity;
      quantity = prompt("Type a quantity greater than 0");
      if (quantity <= 0) {
        document.write("<p>Quantity is not greater than 0.</p>");
        document.write("<p>Please refresh the web page.</p>");
      } else {
        document.write("<p>Quantity is greater than 0.</p>");
      }
    </script>
  </body>
</html>
```

Loop Examples

```html
<!DOCTYPE html>
<html>
  <body>
    <p>Learning JavaScript While Loops</p>
    <p id="demo7"></p>
    <script>
      let msg = "";
      let YY = 0;
      while (YY < 10) {
        msg += "<br>The next digit is " + YY;
        YY++;
      }
      document.getElementById("demo7").innerHTML = msg;
    </script>
  </body>
</html>
```

```html
<!DOCTYPE html>
<html>
  <body>
    <p id="forloop-demo"></p>
    <script>
      const carsforsale = ["Honda", "Toyota", "Vezel", "Fiat", "Volkswagen"];
      let msg = "";
      for (let YY = 0; YY < carsforsale.length; YY++) {
          msg += carsforsale[YY] + "<br>";
      }
      document.getElementById("forloop-demo").innerHTML = msg;
    </script>
  </body>
</html>
```

Creating Functions

- Function declarations

```
function logCompliment() {
     console.log("You're doing great!");
}

logCompliment();
```

```
// Invoking the function before it's declared
hey();
// Function Declaration
function hey() {
     alert("hey!");
}
```

- Function expressions

```
const logCompliment = function() {
     console.log("You're doing great!");
};

logCompliment();
```

```
// Invoking the function before it's declared
hey();
// Function Expression
const hey = function() {
     alert("hey!");
};
```

```
TypeError: hey is not a function
```

Creating Functions

- Passing arguments

```
const logCompliment = function(firstName, message) {
    console.log(`${firstName}: ${message}`);
};
logCompliment("Molly", "You're so cool");
```

- Function returns

```
const createCompliment = function(firstName, message) {
    return `${firstName}: ${message}`;
};
createCompliment("Molly", "You're so cool");
```

Creating Functions

- Default parameters

```
function logActivity(name = "Shane McConkey", activity = "skiing") {
    console.log(`${name} loves ${activity}`);
}
```

```
const defaultPerson = {
    name: {
        first: "Shane",
        last: "McConkey"
    },
    favActivity: "skiing"
};
function logActivity(person = defaultPerson) {
    console.log(`${person.name.first} loves ${person.favActivity}`);
}
```

Arrow Functions

- A useful new feature of ES6. With arrow functions, you can create functions without using the function keyword.

```
// Typical function
const lordify = function(firstName, land) {
         return `${firstName} of ${land}`;
};

// Arrow Function
const lordify = (firstName, land) => `${firstName} of ${land}`;

console.log(lordify("Don", "Piscataway")); // Don of Piscataway
console.log(lordify("Todd", "Schenectady")); // Todd of Schenectady
```

- Returning objects

```
const person = (firstName, lastName) => ({
     first: firstName,
     last: lastName
});
console.log(person("Flad", "Hanson"));
```

Functions

```
function getData() {
  // do something
}
getData()

function getData(color) {
  //do something
}
getData('black')

function getData((color = "black", age = 25) {
  //do something
}
getData('green', 24)

function getData1() {
  // do something
  return "hi!";
}
let result = getData1();

function getData2() {
  return ["FIBO", 30];
}
let [name, age] = getData2();
```

Arrow Functions

```
() => {
  //...
};

let getData = function getData() {
  //...
};

let getData = function () {
  //...
};

getData();

let getData = () => {
  //...
};

const getData = () => console.log("hi!");

const getData = (param1, param2) => console.log(param1, param2);

const getData = (color = "black", age = 2) => {
  //do something
};
```

Function Example

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>JavaScript Practice</title>
    <meta charset="utf-8" />
    <script>
      function promptQuantity() {
        var quantity;
        quantity = prompt("Please type a quantity greater than 0");
        if (quantity <= 0) {
          alert("Quantity is not greater than 0.");
        } else {
          alert("Thank you for entering a quantity greater than 0.");
        } // end if
      } // end function promptQuantity
    </script>
  </head>
  <body>
    <h1>Using JavaScript</h1>
    <button onclick="promptQuantity();">Click to enter quantity</button>
  </body>
</html>
```

Objects and Properties

- An object can contain multiple values as properties.
- Each property is made of up a key/value pair. The “key” in “key/value” is a string we define that points to a value—as with naming a variable, we want our keys to have names that are predictable, flexible, and easy to understand.
- Defining an objects
  - New keyword

```
const car = new Object();
const car = Object.create();
```

  - Object literal notation

```
const car = {};
```

```
const car = {
  color: "blue",
};
car.color = "yellow";
car["color"] = "red";


const car = {
  brand: {
    name: "Ford",
  },
  color: "blue",
};
car.brand.name;
car["brand"]["name"];


const car = {
  brand: "Ford",
  model: "Fiesta",
  start: function () {
    console.log(`Started ${this.brand} ${this.model}`);
  },
};
car.start();
```

Classes

```
class Person {
  constructor(name) {
    this.name = name;
  }
  hello() {
    return "Hello, I am " + this.name + ".";
  }
}

const sealab = new Person("sealab");
sealab.hello(); //'Hello, I am sealab.'
```

```
class Programmer extends Person {
  hello() {
    return super.hello() + ". I am also a programmer.";
  }
}
const sealab = new Programmer();
sealab.hello(); //'Hello, I am undefined.. I am also a programmer.'
```

```
class Vacation {
  constructor(destination, length) {
    this.destination = destination;
    this.length = length;
  }
  print() {
    console.log(`${this.destination} will take ${this.length} days.`);
  }
}
const trip = new Vacation("Santiago, Chile", 7);
trip.print(); // Chile will take 7 days.
```

```
class Expedition extends Vacation {
  constructor(destination, length, gear) {
    super(destination, length);
    this.gear = gear;
  }
  print() {
    super.print();
    console.log(`Bring your ${this.gear.join(" and your ")}`);
  }
}
const trip = new Expedition("Mt. Whitney", 3, [
  "sunglasses",
  "prayer flags",
  "camera",
]);
trip.print();
```

Asynchronous Programming and Callbacks

- The callback function is executed asynchronously.

```
const doSomething = (callback) => {
  //do things
  //do things
  const result = /* .. */ callback(result);
};
```

- The promise means completing it successfully, rejecting a promise means ending it with an error.

```
const doSomething = new Promise((resolve, reject) => {
  //some code
  const success = /* ... */
  if (success) {
    resolve("ok");
  } else {
    reject("this error occurred");
  }
})
```

- Async functions are a higher-level abstraction over promises.

```
const doSomething = async () => {
  const data = await getData();
  console.log(data);
};
```

```
const getFirstUserData = async () => {
  // get users list
  const response = await fetch("/users.json");
  // parse JSON
  const users = await response.json();
  // pick first user
  const user = users[0];
  // get user data
  const userResponse = await fetch(`/users/${user.name}`);
  // parse JSON
  const userData = await userResponse.json();
  return userData;
};
getFirstUserData();
```

Alert Message Box

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>JavaScript Practice</title>
    <meta charset="utf-8" />
  </head>
  <body>
    <h1>Using JavaScript</h1>
    <script>
      alert("Welcome to my web page!");
    </script>
    <h2>When does this display?</h2>
  </body>
</html>
```

Event Handlers and Alert Messages

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>JavaScript Practice</title>
    <meta charset="utf-8" />
  </head>
  <body>
    <h1>Using JavaScript</h1>
    <ul>
      <li>
        <a href="#" onmouseover="alert('You moused over');">Mouseover test</a>
      </li>
      <li>
        <a href="#" onmouseout="alert('You moused out');">Mouseout test</a>
      </li>
    </ul>
  </body>
</html>
```

addEventListener Method

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>JavaScript Practice</title>
    <meta charset="utf-8" />
  </head>
  <body>
    <h1>Using JavaScript</h1>
    <button id="myB">Click to enter quantity</button>
    <script>
      var myButton = document.getElementById("myB");
      myButton.addEventListener("click", promptQuantity);
      function promptQuantity() {
        var quantity;
        quantity = prompt("Please type a quantity greater than 0");
        if (quantity <= 0) {
          alert("Quantity is not greater than 0.");
        } else {
          alert("Thank you for entering a quantity greater than 0.");
        } // end if
      } // end function promptQuantity
    </script>
  </body>
</html>
```

Form Handling

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>JavaScript Practice</title>
    <meta charset="utf-8" />
    <style>
      input {
        display: block;
        margin-bottom: 1em;
      }
      label {
        float: left;
        width: 5em;
        padding-right: 1em;
        text-align: right;
      }
      input[type="submit"] {
        margin-left: 7em;
      }
    </style>
  </head>
```

```
  <body>
    <h1>JavaScript Form Handling</h1>
    <form
      method="post"
      action="https://fibo.kmutt.ac.th"
      onsubmit="return validateForm();"
    >
      <label for="userName">Name: </label>
      <input type="text" name="userName" id="userName" />
      <label for="userAge">Age: </label>
      <input type="text" name="userAge" id="userAge" />
      <input type="submit" value="Send information" />
    </form>
  </body>
</html>
```

```
    <script>
      function validateForm() {
        if (document.forms[0].userName.value == "") {
          alert("Name field cannot be empty.");
          return false;
        } // end if
        if (document.forms[0].userAge.value < 18) {
          alert("Age is less than 18. You are not an adult.");
          return false;
        } // end if
        alert("Name and age are valid.");
        return true;
      } // end function validateForm
    </script>
```

Node.js

- Node.js is an open-source and cross-platform JavaScript runtime environment.
- Node.js runs the V8 JavaScript engine, the core of Google Chrome, outside of the browser. This allows Node.js to be very performant.
- A Node.js app runs in a single process, without creating a new thread for every request. Node.js provides a set of asynchronous I/O primitives in its standard library that prevent JavaScript code from blocking.
- Run node application: `node app.js`
- Restart the application automatically, nodemon module is used by install: `npm i -g nodemon`
- Run the application using nodemon followed by application file name: `nodemon app.js`
- Set environment variables of the node application: `USER_ID=239482 USER_KEY=foobar node app.js`
- An example to accesses the USER_ID and USER_KEY environment variables:

```
process.env.USER_ID // "239482"
process.env.USER_KEY // "foobar"
```

- Create an .env file in the root directory of your project

```
# .env file
USER_ID="239482"
USER_KEY="foobar"
NODE_ENV="development"
```

```
require('dotenv').config()
process.env.USER_ID // "239482"
process.env.USER_KEY // "foobar"
process.env.NODE_ENV // "development"
```

An Example Node.js Application

```javascript
const http = require("http")
const hostname = "127.0.0.1"
const port = 3000
const server = http.createServer((req, res) => {
  res.statusCode = 200
  res.setHeader("Content-Type", "text/plain")
  res.end("Hello World\n")
})
server.listen(port, hostname, () => {
  console.log(`Server running at http://${hostname}:${port}/`)
})
```

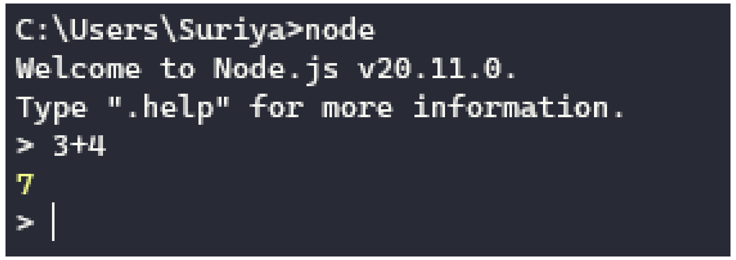
indicate a successful response

- Express
- Meteor
- koa
- Next.js
- Micro
- Socket.io
- SvelteKit
- Remix
- Fastify

Where to Host a Nodejs App

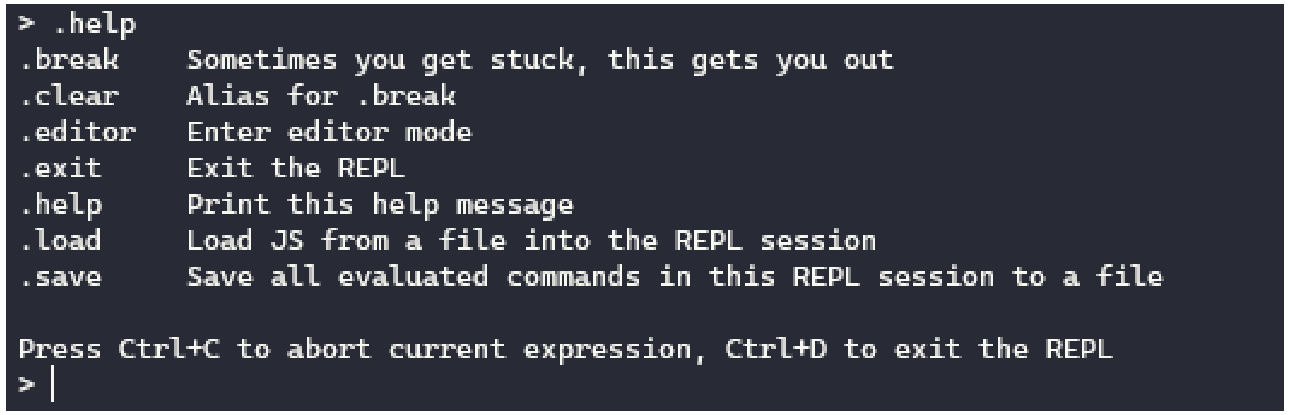
- local tunnel: https://github.com/localtunnel/localtunnel
- ngrok: https://ngrok.com/
- Glitch: https://glitch.com/
- Codepen: https://codepen.io/
- Serverless: publish your apps as functions on AWS Lambda, Azure, Google Cloud
- PAAS: Platform As A Service
    - Vercel: https://vercel.com/
    - Nanobox
    - Heroku
    - Microsoft Azure
    - Google Cloud Platform
- Virtual Private Server

Node.js REPL



```
C:\Users\Suriya>node
Welcome to Node.js v20.11.0.
Type ".help" for more information.
> 3+4
7
> |
```

- REPL stands for Read Evaluate Print Loop, and it is a programming language environment.
- The Node.js *Read-Eval-Print-Loop* (**REPL**) is an interactive shell that processes Node.js expressions. The shell **reads** JavaScript code the user enters, **eval**uates the result of interpreting the line of code, **prints** the result to the user, and **loops** until the user signals to quit.
- Start a REPL session: `node`
- Get access to the history of the previous lines of code executed: up arrow key
- Dot commands: .help, .editor, .break, .clear, .load, .save, .exit
- Exploring JavaScript objects: type JavaScript class, then '.' and press 'tab'
- Explore global objects: global. and pressing 'tab'
- Print the result of the last operation: '_'
- Use the tab to autocomplete



```
> .help
.break    Sometimes you get stuck, this gets you out
.clear    Alias for .break
.editor   Enter editor mode
.exit     Exit the REPL
.help     Print this help message
.load     Load JS from a file into the REPL session
.save     Save all evaluated commands in this REPL session to a file

Press Ctrl+C to abort current expression, Ctrl+D to exit the REPL
> |
```

Arguments from the Command Line

- Passing any number of arguments when invoking a Node.js application: `node app.js arg1 arg2`
- Iterate over all the arguments and get the additional arguments by creating a new array:

```
process.argv.forEach((val, index) => {
  console.log(`${index}: ${val}`)
})

const args = process.argv.slice(2)
```

```
node app.js name=fibo
```

- Using the minimist library, which helps dealing with arguments: `npm install minimist`
- Use double dashes before each argument name: `npm app.js --name=fibo`

```
const args = require('minimist')(process.argv.slice(2))
console.log(args.name)
```

NPM

- Use **npm** to install a variety of packages
- A **package.json** file describes the project and all its dependencies.
- Using this command to initialize the project from scratch and create a package.json file :

```
npm init -y
```

```
{
  "name": "myapp",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

- To install your own dependencies with npm:

```
npm install package-name
```

- To remove a package with npm:

```
npm remove package-name
```

YARN

- An alternative to npm is Yarn. It was released in 2016 by Facebook in collaboration with Exponent, Google, and Tilde. The project helps Facebook and other companies manage their dependencies reliably.
- Yarn is used in production by Facebook and is included in projects like React, React Native, and Create React App.
- First, install Yarn globally with npm:

```
npm install -g yarn
```

- To install your own dependencies with yarn:

```
yarn add package-name
```

- To remove a package with yarn:

```
yarn remove package-name
```

nodemon

- A tool that helps develop Node.js based applications by automatically restarting the node application when file changes in the directory are detected.
- Install

```
npm install -g nodemon
```

- Uses

```
nodemon app.js
```

Bootstrap

- Bootstrap is an open source frontend framework maintained by Twitter for developing responsive websites and web applications. It includes HTML, CSS, and JavaScript code to build user interface components. It's a fast and easy way to develop a powerful mobile-first user interface.

```
npm i bootstrap@5.3.2
```

- Bootstrap uses HTML5 as the means to create content, and Bootstrap adds a large number of CSS classes that can be assigned to those elements. In addition to this, JavaScript code has been provided to deliver specific functionality.

active	clearfix	fixed	justify-content	order	success
alert	close	float	lead	P	tab
align	Col	font	list	page	table
badge	container	form	m	position	tooltip
bg	custom	h	mark	progress	visible
blockquote	d	has	modal	rounded	w
border	Display	initialism	nav	row	was-validated
btn	drop	input	next	shadow	
card	embed	invalid-feedback	no	sr-only	
carousel	fade	is	Offset	stretched	

Web Frameworks: Express

- Write handlers for requests with different HTTP verbs at different URL paths (routes).
- Integrate with "view" rendering engines in order to generate responses by inserting data into templates.
- Set common web application settings like the port to use for connecting, and the location of templates that are used for rendering the response.
- Add additional request processing "middleware" at any point within the request handling pipeline.

- Install Express

```
npm install express
```

Hello World - Express

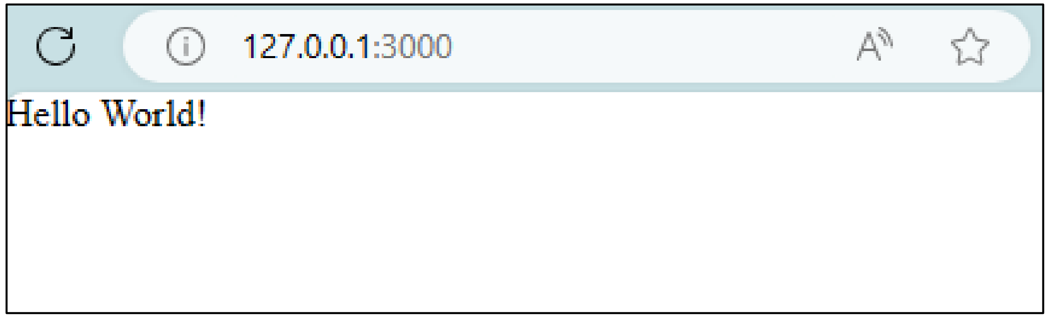
app.js

```
const express = require("express");
const app = express();
const port = 3000;

app.get("/", function (req, res) {
  res.send("Hello World!");
});

app.listen(port, function () {
  console.log(`Example app listening on port ${port}!`);
});
```

node app.js



Hello World!

- Methods in Express

```
checkout(), copy(), delete(), get(), head(), lock(), merge(), mkactivity(), mkcol(), move(), m-
search(), notify(), options(), patch(), post(), purge(), put(), report(), search(), subscribe(),
trace(), unlock(), unsubscribe()
```

Serving Static Files

- Use the line below to serve images, CSS files, and JavaScript files from a directory named 'public' at the same level as where you call node:

```
app.use(express.static("public"));
```

- Any files in the public directory are served by adding their filename (relative to the base "public" directory) to the base URL. So for example:

```
http://localhost:3000/images/dog.jpg
http://localhost:3000/css/style.css
http://localhost:3000/js/app.js
http://localhost:3000/about.html
```

- Create a virtual prefix for your static URLs

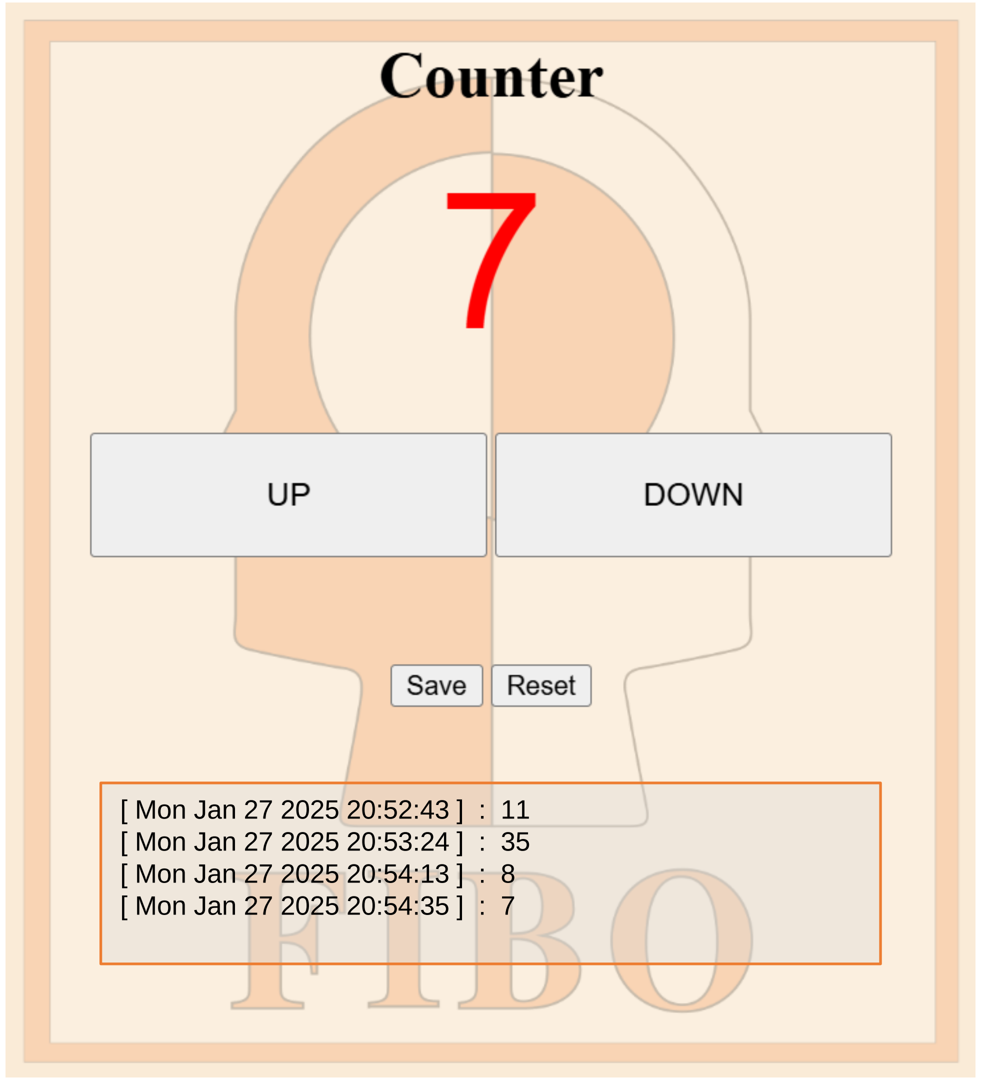
```
app.use("/media", express.static("public"));
```

Practice More

- https://www.freecodecamp.org/learn
- https://exercism.org/
- https://www.codementor.io/projects/javascript
- https://www.codingame.com/
- https://www.warriorjs.com/

Fun 03: Counter

- ให้สร้างเว็บไซต์สำหรับนับจำนวนผู้ใช้บริการ โดยมีคุณสมบัติดังนี้
  - ตัวเลขแสดงผลจำนวนผู้ใช้บริการ
  - ปุ่มเพิ่มจำนวน
  - ปุ่มลงจำนวน
  - ปุ่มบันทึก เมื่อกดแล้วจะบันทึกผลจำนวนผู้ใช้บริการ พร้อมวันเวลาที่บันทึก
  - ปุ่มรีเซ็ตให้จำนวนเป็น 0
  - ส่วนแสดงผลการบันทึกข้อมูล



```
[ Mon Jan 27 2025 20:52:43 ]  :  11
[ Mon Jan 27 2025 20:53:24 ]  :  35
[ Mon Jan 27 2025 20:54:13 ]  :  8
[ Mon Jan 27 2025 20:54:35 ]  :  7
```