JavaScript and Node.js Tutorial

Saravanan Vijayakumaran

Department of Electrical Engineering
Indian Institute of Technology Bombay

March 11, 2026

JavaScript and Node.js

- JavaScript
  - A language created in 1995 to add interactivity to webpages
  - First shipped as part of the Netscape Navigator browser
  - Supported by all browsers today
  - Has little connection to Java; name chosen for marketing reasons
- Node.js is a JavaScript runtime released in 2009
  - Runtime = Execution environment
  - Allows running JavaScript programs outside the browser
  - Enabled the creation of many web frameworks to write server applications using JavaScript instead of PHP
  - Basis for many popular frontend frameworks like React, Vue
- Our needs
  - Learn enough JavaScript and Node.js to use Hardhat
  - Hardhat is a Node.js package for Ethereum smart contract development

JavaScript Tutorial

Hello World

- Using the browser
  - Open Developer Tools in a browser
  - Type the following in the Console tab

```
console.log("Hello, world!")
```

- Using Node.js runtime
  - Install Node.js using instructions given here (`nvm` method is recommended for Linux/macOS)
  - Create file `hello.js` containing the same string you typed above
  - Run `node hello` in a terminal

Variables

- Variables can be declared using the `let` keyword

```
let x = 5;
let y;
y = 'Hello';
```

- Statements have a optional semicolon at the end
  - Semicolons are useful when we want to place multiple statements on a single line
- Variable declaration and initialization can be done separately
  - Uninitialized variables evaluate to `undefined`
- The type of variables is automatically inferred
- The same variable can be reassigned to a value of a different type (dynamic typing)

```
let x = 5;
let y = 'Hello';
x = y;
```

- Immutable variables can be declared using the `const` keyword

```
const z = 5;
```

Data Types

- JavaScript has 8 basic data types
  - `number`: integers in the range $\pm(2^{53} - 1)$ or floating-point numbers, `Infinity`, `-Infinity`, `NaN`
  - `bigint`: arbitrary-precision integers; denoted with an `n` suffix
  - `string`: a string of zero or more characters
  - `boolean`: `true` or `false`
  - `undefined`: unassigned variables
  - `null`: can be used to set a variable to an unknown value
  - `object`: Complex data structures
  - `symbol`: Unique identifiers that can be used as keys in objects
- The `typeof` operator returns the data type of a variable

Control Flow

- `if` statement

```
if (x % 2 == 0) {
    console.log("Even")
} else if (x % 3 == 0) {
    console.log("Odd and divisible by 3")
} else {
    console.log("Not divisible by 2 or 3")
}
```

- Conditional operator `?`

```
let accessAllowed = (age > 18) ? true : false;
```

- `switch` statement

```
switch (a) {
  case 3:
    console.log( 'Too small' );
    break;
  case 5:
    console.log( 'Too big' );
    break;
  default:
    console.log( "I don't know such values" );
}
```

Loops

- `while` loop

```
while (condition) {
  // code
}
```

- `for` loop

```
for (let i = 0; i < 3; i++ ) {
  console.log(i);
}
```

Functions

- Funcion declaration

```
function sum(a, b) {
  return a + b;
}
```

  - The types of arguments are not specified
  - The types of return values type are not specified
- Functions can return different types depending on its input

```
function getHostel(strName, num) {
    if (strName) {
      return `Hostel No. ${num}`;
    } else {
      return num
    }
}
```

- **Aside:** Strings enclosed in backticks can be used for variable interpolation

Functions as Values

- Functions can be passed as arguments to other functions

```
function showMessage(predicate, yes, no) {
  if (predicate) yes()
  else no();
}

function yesFunction() {
  console.log("Predicate is true.");
}

function noFunction() {
  console.log("Predicate is false.");
}

showMessage(true, yesFunction, noFunction);
```

- Useful for defining callbacks

Arrow Functions

- Functions can also be defined using arrow notation

```
let func = (arg1, arg2,.., argN) => expression;
```

- Example

```
let sum = (a, b) => a + b;
```

- Rewriting earlier example with arrow functions

```
function showMessage(predicate, yes, no) {
  if (predicate) yes()
  else no();
}

showMessage(
  true,
  () => console.log("Predicate is true."),
  () => console.log("Predicate is false.")
);
```

Arrays

- List of possibly differently typed values

```
let arr = [1, 2, 'Hello']
```

- The *i*th element can be accessed as `array[i]`
- Elements can added and removed at the end of the array using `push/pop`

```
arr.push(1)
arr.push('Hello')
arr.pop()
```

- Elements can added and removed at the beginning of the array using `unshift/shift`

```
let fruits = ["Orange", "Pear"];

fruits.unshift('Apple');

console.log( fruits ); // Apple, Orange, Pear
```

Objects

- Keyed collections of data; key-value pairs are called properties

```
let user = {
  name: "John",
  age: 30
};
```

- Property values can be accessed using key names in two ways

```
console.log(user.name);
console.log(user["age"])
```

- Properties can be added by assignment, removed using `delete`, and checked for existence using `in`

```
user.isAdmin = true
delete user.age
console.log("age" in user)
```

- We can iterate over keys using a `for` loop

```
for (let k in user) {
    console.log(k, user[k])
}
```

Promises

- Many JavaScript functions are asynchronous
  - Results of the function are not immediately available
- Example: `fetch` gets the contents at a URL

```
let result = fetch("https://example.com/")
console.log(result)
```

- The above code prints `Promise { <pending> }`
- A `Promise` will eventually contain the return value or an error
- We can wait for the asynchronous operation to complete using the `then` method

```
result
    .then((response) => response.text())
    .then(console.log)
    .catch((e) => {
        console.log("Something went wrong!!!!")
        console.log(e)
    })
```

Async/await

- `async/await` is a better syntax for working with promises
- The `async` keyword indicates that the function returns a `Promise`

```
async function f() {
  return 1;
}

f().then(console.log);
```

- We can use `await` inside `async` functions to wait for a promise to be fulfilled

```
async function getText(result) {
    let response = await result;
    let text = await response.text();
    console.log(text)
}

let result = fetch("https://example.com/")
getText(result)
```

Node.js

Node.js

- Enables JavaScript use outside the browser
- Has a large collection of open-source packages
- Enter Node.js REPL using `node` command
- Run scripts using `node script-name.js`
- `npm` = Node package manager
- Install packages using `npm install package-name`
  - Package files are installed in `node_modules` subdirectory of current directory
  - For global installation, use `npm install -g`
  - `package.json` and `package-lock.json` files contain details of installed packages

Example: Accepting Command-Line Input

- The `inquirer` package can be used to accept input from the command line
- Run `npm install inquirer`
- Contents of `package.json`

```
{
  "dependencies": {
    "inquirer": "^9.2.15"
  }
}
```

- Add a property with key `type` and value `module`

```
{
  "type": "module",
  "dependencies": {
    "inquirer": "^9.2.15"
  }
}
```

  - This is a hack for demo purposes; not the default workflow

Example: Accepting Command-Line Input

Put the following code in main.js and run with "node main"

```
import inquirer from "inquirer";

const questions = [
  {
    name: 'name',
    type: 'input',
    message: "What's your name?",
  },
  {
    name: 'program',
    type: 'select',
    choices: ["BTech", "Dual Degree", "MTech", "IDDP", "PhD"
        ],
    message: "Which program are you in?",
  },
];

inquirer.prompt(questions).then(answers => {
  console.log(`Name: ${answers.name}`);
  console.log(`Program: ${answers.program}`);
});
```

References

- JavaScript Wikipedia page `https://en.wikipedia.org/wiki/JavaScript`
- Node.js Wikipedia page `https://en.wikipedia.org/wiki/Node.js`
- JavaScript Tutorial `https://javascript.info/`
- JavaScript for NodeJS
- fetch `https://developer.mozilla.org/en-US/docs/Web/API/fetch`
- Response of fetch `https://developer.mozilla.org/en-US/docs/Web/API/Response`
- Accepting command-line input in Node.js