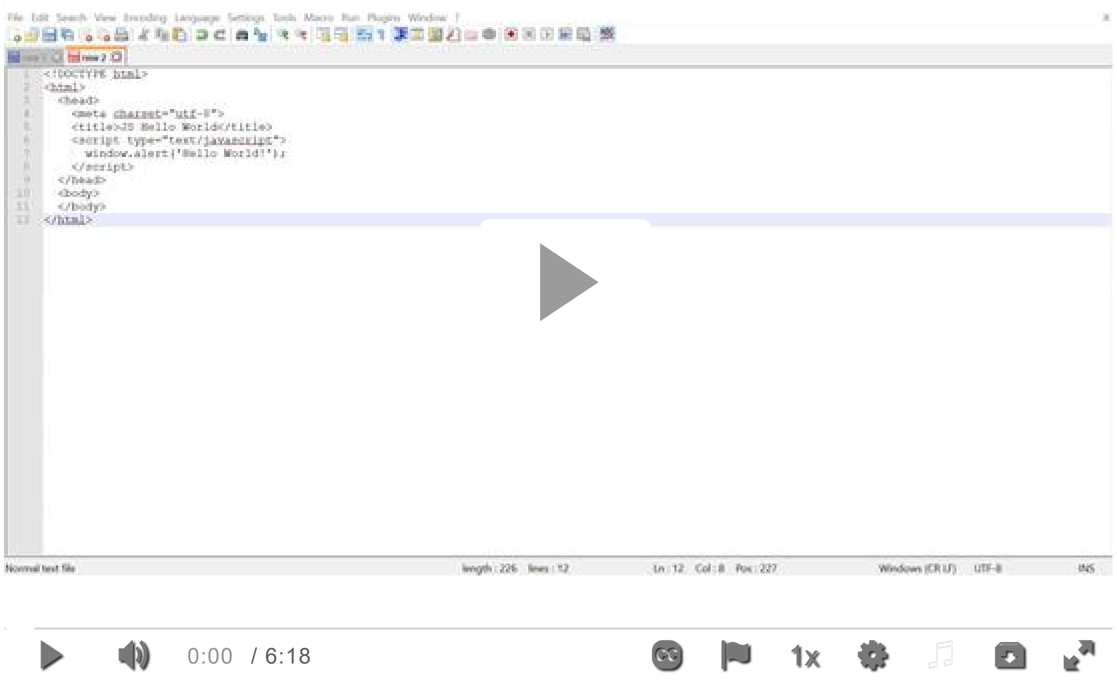
# Exploration — Introduction to JavaScript

## History

JavaScript was created in 1995 by Brendan Eich who was working for Netscape Corporation. Netscape Corporation produced the web browser Netscape Navigator which was one of the earliest commercially produced browsers. The early web consisted solely of static pages. Netscape quickly discovered the need for a programming language to run in the browser to allow dynamic behavior after a page was downloaded to the browser. JavaScript was initially named LiveScript. However, the Java programming language was created at Sun Microsystems at around the same time and was considered the "hot" programming language. So for marketing reasons, the name LiveScript was changed to JavaScript.



```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>JS Hello World</title>
    <script type="text/javascript">
      window.alert('Hello World!');
    </script>
  </head>
  <body>
  </body>
</html>
```

## Running JavaScript Programs

We will follow convention and write a `Hello World` program as our first program in JavaScript. However, as we will see, our program will differ a bit based on how we want to run it. Let us look at our `Hello World` program in the context of 3 different ways in which we can run JavaScript programs.

### Running JavaScript in a Browser

We can embed JavaScript directly in an HTML document. We will study HTML later in the course, but here is an example HTML document with JavaScript code.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>JS Hello World</title>
    <script type="text/javascript">
      window.alert('Hello World!');
    </script>
  </head>
  <body>
  </body>
</html>
```

If we save this code in a file and then open the file in a browser, the browser will pop up a window with the message `Hello World!`. Note that our JavaScript program is comprised of just the following line

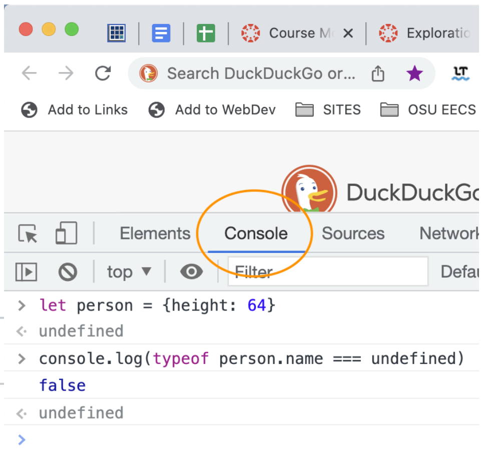
```
window.alert('Hello World!');
```

`window` is a variable in JavaScript that represents the window in which the JavaScript program is running. The function `alert` displays an alert dialog and we are passing the string `Hello World!` to be displayed in this alert. As will learn later, generally we create a separate file (or multiple files) for the JavaScript code we want to run in the browser. Then in the HTML documents in which we want this JavaScript code to be executed, we add references to these files with JavaScript code.

## Running JavaScript in the Browser Console

Modern browsers, such as Chrome, Edge, Mozilla, etc., have a JavaScript console built-in as part of developer tools provided with the browser. For example, in Chrome we can open up the developer tools by using `Ctrl-Shift-I` and then click on the 'Console' tab to view the JavaScript console. Here is a one line `Hello World!` program that we can run in this console.

```
console.log('Hello World');
```



```
> let person = {height: 64}
<· undefined
> console.log(typeof person.name === undefined)
  false
<· undefined
>
```

`console` is a variable in JavaScript that represents the browser's debugging console. We use the `log` method on `console` to print the `Hello World` message.

## Running JavaScript Using Node.js

An alternative to using a browser to run JavaScript is to use **Node.js**, also simply called **Node**. Node is a program that runs JavaScript programs similar to how browsers run JavaScript program. In fact, Node uses a JavaScript engine called V8 which is developed by Google and is used to run JavaScript in many browsers including Chrome and Edge.

If you haven't already done so, **download and install Node v16x (https://canvas.oregonstate.edu/courses/1879154/modules/items/22290737)** . Then create a file `hello_world.js` and write the following line of code in this file:

```
console.log('Hello World');
```

### Running a File Using the command node

Now we can run this file by going to the directory where the file is located and running the command

```
node hello_world.js
```

### Interactively Running JavaScript Code Using the Node REPL

In addition to using Node to run JavaScript code saved in a file, we can also use Node as a JavaScript "read-eval-print loop" or a **REPL** in which we can enter JavaScript statements. This is similar to how we run JavaScript in the console built into web browsers. To start Node's JavaScript REPL, simply run the command `node` from the command line. We then type and run JavaScript code in the REPL as shown below:

```
PS C:\Users\nauman> node
Welcome to Node.js v14.17.6.
Type ".help" for more information.
> console.log('Hello World!');
Hello World!
undefined
>
```

### Loading a File in the Node REPL

The Node REPL also supports loading a file into the current session using the `.load` command. Node will then read that file and automatically run it. The `.load` command needs to be provided the path to the file we want to load. If the file is located in the same directory from which we started the Node REPL, then the path to the file is simply the name of the file. We can load multiple files in a session by repeated use of the `.load` command. If we update a file after it has been loaded, then we can load the updated version of the file by again loading it using the `.load` command.

For example, if we start the Node REPL in the same directory where the file `hello_world.js` is located, we can load and run this file using the command `.load hello_world.js` as shown below:

```
PS C:\Users\nauman> node
Welcome to Node.js v14.17.6.
Type ".help" for more information.
```

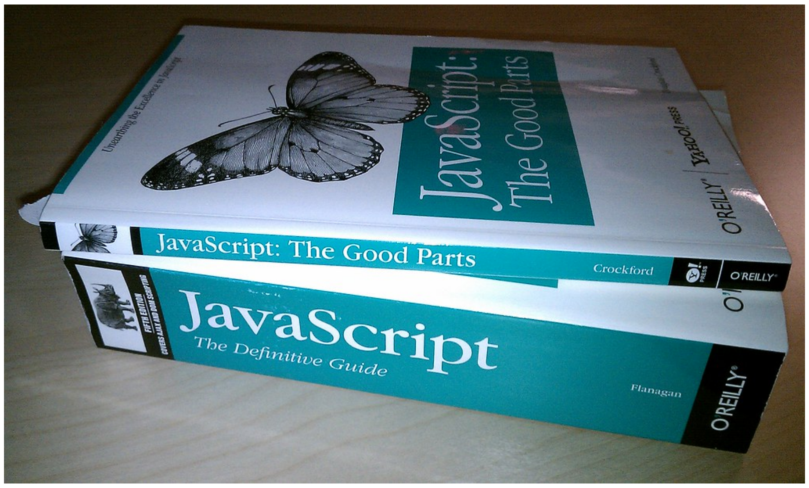
```
> .load hello_world.js
console.log('Hello World');
Hello World
undefined
>
```

Node is very widely used to develop and run server-side web applications in JavaScript. We will use Node extensively to develop our server-side programs in this course.

## The Evolution of JavaScript

The original goal of JavaScript was to serve as an easy to learn language for writing a few lines of code embedded in HTML documents. As such, it has some features that make it convenient for beginners to write code, but can also cause very strange behavior. This strange behavior has been termed JavaScript **Wat** and is hilariously described in **Wat - A lightning talk by Gary Bernhardt (https://media.oregonstate.edu/media/t/1_6ak0gvw0)** (CodeMash 2012). But the use of JavaScript continued to grow to write very rich web applications.

To help write more sophisticated programs using JavaScript, people identified patterns for writing good JavaScript programs that avoid the bad parts of JavaScript. A lot of this knowledge was described by Douglas Crockford in his book *JavaScript: The Good Parts* in 2008. This book is used in another meme about JavaScript with a picture of his short book along with the voluminous book *JavaScript: The Definitive Guide* emphasizing that JavaScript contains a lot of bad parts!



## Modern JavaScript

Delving into the history of JavaScript is incidental to our goal of learning web application development. However, knowing this history makes us aware that we must not blindly use a language feature just because JavaScript provides it. Overtime, JavaScript standards have been created to modernize the language and add features for good programming practice. Our goal in this course will be to use these modern features of JavaScript to write good, maintainable programs.

The course textbook ***Modern JavaScript for the Impatient*** **(https://www.pearson.com/us/higher-education/program/Horstmann-Modern-Java-Script-for-the-Impatient/PGM2736532.html)** is written with this goal in mind as well. The book lists the following 5 **golden rules** to follow to avoid the bad parts of JavaScript (page xvi):

1. Declare variables with `let` or `const`, not `var`.
2. Use strict mode.
3. Know your types and avoid automatic type conversion.
4. Understand prototypes, but use modern syntax for classes, constructors, and methods.
5. Don't use `this` outside constructors or methods.

The book also specifies a meta-rule **Avoid the Wat**. We don't yet have the knowledge to understand these rules, so don't worry about trying to understand them now. But over the coming modules and explorations, we will learn what these rules mean and why we should follow them.

## Summary

JavaScript is **the** language for writing client-side code for web applications. However, it is now also widely used for server-side programming. In the past, JavaScript has been much maligned for some confusing features. However, over time the language has evolved to add features for good programming practices. We will focus our study and use of JavaScript on these modern features.

## Additional Resources

Here are some references to learn more about the topics we discussed in this exploration.

- The history of JavaScript is described briefly in the **Wikipedia article (https://en.wikipedia.org/wiki/JavaScript)** and in much greater detail in the paper **JavaScript: the first 20 years (https://dl.acm.org/doi/10.1145/3386327)** by *Allen Wirfs-Brock* and *Brendan Eich* in *The Proceedings of the ACM on Programming Languages*, June 2020.
- The funny **Wat talk (https://media.oregonstate.edu/media/t/1_6ak0gvw0)** by Gary Bernhardt from 2012.
- JavaScript: The Good Parts, by *Douglas Crockford*, Yahoo Press & O'Reilly, 2008.

Image attributions

- "JavaScript: Good Parts vs. The Rest" by Ольга + Нафан is licensed under **CC BY 2.0 (http://creativecommons.org/licenses/by/2.0)**