Why Use JavaScript?

JavaScript is a language that can be used in a variety of ways. You can use it for webpages, desktop applications, Backend Development, and other areas of development. What makes it special for web development is how JavaScript allows for the developer to add dynamic content such as menus and dropdowns so that the user can navigate through the website easily. JavaScript's design also allows for browser capability meaning that the browser works properly on any browser version on any device. JavaScript can also be used for desktop applications, with the help of a software framework called Electron the developer is able to use a module called Browser Module which lets them customize their API, can control the application menus, or alert users with dialogue. Electron also has first-class support with the Mac app store (macOS) and Microsoft Store (Windows), and Snap Store (Linux). Another major use for JavaScript is backend development, with JavaScript's ability to create efficient server-side scripts it can automate tasks on the server and reduces processing time. JavaScript has a runtime environment called Node.js which was introduced in 2009 and it allows JavaScript to be ran outside of a web browser. Node is very lightweight and efficient so it is popular for backend development, but there are 2 other popular frameworks that can be used namely Koa.js and Express.js. JavaScript has a lot to give to backend developers since it has an asynchronous nature, is scalable, and speedy. Having an asynchronous nature means that the program does not wait for one task to complete before moving onto the next, this is necessary for backend development because it allows for the server to take multiple requests at once without having to wait for each one to move onto the next one. The scalability part revolves around the system's ability to handle increased loads. JavaScript is event-driven, because of this JavaScript is very scalable and can handle large number of events/requests without slowing down. And backend development also requires speed. As JavaScript is compiled and executed on the server, it doesn't need to be downloaded and interpreted on the browser. One more thing that JavaScript is useful for is Game development. With its wide spread use in web browsers and other platforms, JavaScript enables game developers to create interactive and engaging games. Games built with JavaScript may include user input, physics, sound, and graphics. Using JavaScript developers can create the games across a wide variety of devices.

The History behind JavaScript

In the 90's Netscape navigator was one of the biggest web browsers at the time. Netscape wanted their website to be more dynamic and interactive along with being able to update and interact with others on a webpage without having to refresh the page. When they initially began developing this they wanted to use Java however after getting a little into it

they realized that it deserves it's own language that they can modify to work better for what they want. Brendan Eich made Mocha in 10 days in 1995. Mocha was later changed to LiveScript but then finally changed to JavaScript. In 1996 JavaScript was proposed to be standardized by ECMA international, this led to ECMA-262 or ECMAScript. This led to JavaScript being implemented into many websites to make the more interactive and dynamic beginning in 1997. This allowed websites to make improvements and be more interactive. One such example is the implementation of YouTubes old popup annotations . These annotations would pop up on screen either providing information about the video or a link to another video. You would get rid of these by pressing the X in the corner removing it from the screen in real time no refresh needed. While JavaScript was created for Netscape navigator JavaScript would be used in internet explore which came out the same year JavaScript was made. JavaScript is easy to use so many websites use it to the point where about 95% of websites use JavaScript in some way.

How It Works

The Big Picture:

JavaScript is an interpreted language capable of ensuring compatibility between various environments independent of specific browsers or operating systems. Its integration into the ECMAScript standard allows developers to create web interfaces with insurance that compatibility will be reliable. JavaScript supports three very important paradigms: procedural, functional, and object oriented, allowing it to be flexible and operate in multiple different areas outside of its original purpose of web development.

The Paradigms:

There are several distinct advantages that JavaScript possesses due to its multi paradigm nature. Each of these advantages help JavaScript operate in the various types of projects it's used in. Some of these advantages and how they applied in our programs are listed below.

The first paradigm is procedural. An example of an advantage is how JavaScript's procedural nature allows scripts, attached to HTML web pages, to execute without requiring them to be attached to a greater JavaScript program. This freedom allows programmers to implement smaller sections of code throughout a webpage without having to implement everything into a larger JavaScript program. If needed, JavaScript code can even be written directly inside of HTML files. The procedural ability of JavaScript was very useful when working on the complex program. I was able to attach a JavaScript file to an

HTML element inside product.html. This property allowed me to organize my code in a way that made it modular and easy to change, without relying on a singular main method like in other languages such as Java. Something that separates JavaScript from other programming languages capable of procedural programming, is its ability to also support OOP and functional programming styles.

JavaScript is also capable of supporting functional programming techniques. Functions can be stored in variables, higher order functions can be used, and function composition can be used to build more complex functions with smaller functions. JavaScript also supports many more functional programming techniques. An example of a functional technique used in one of our programs is the arrow function.

```
interval = setInterval(() => advanceQueue(queue), 5 * 60 * 1000);
```

This code uses the arrow function to return a reference to advanceQueue() itself as a parameter for setInterval() instead of the result of advanceQueue(). While this uses a functional programming technique, this code is not purely functional. There was not a specific use case within our programs that called for the use of more advanced functional programming techniques. While JavaScript can be used to program in a functional way, it does not strictly enforce the “no side effects” philosophy of languages that have functional programming as their primary focus.

The last paradigm that we will discuss is JavaScript’s Object-Oriented capabilities. Object-Oriented programming is very powerful when building complex applications with many moving parts. JavaScript supports OOP with a form of inheritance that is different from other languages that we are familiar with. This type of inheritance is called prototypal inheritance. Prototypal inheritance allows a “prototype” of an object to be defined and allows other objects to inherit the methods and properties of the prototype. The specifics of this type of inheritance will be further discussed within the Design section of the report. While we did not use inheritance within our programs, we have used objects. Within our complex program, we used CustomerQueue, Customer, and Product objects. Being able to implement these complex data types as objects allowed us to encapsulate and modularize our code. However, since we did not rely on inheritance, it may have been possible to implement these objects with just functions instead. JavaScript’s unique type of inheritance, and other language characteristics will be talked about next in the Design section.

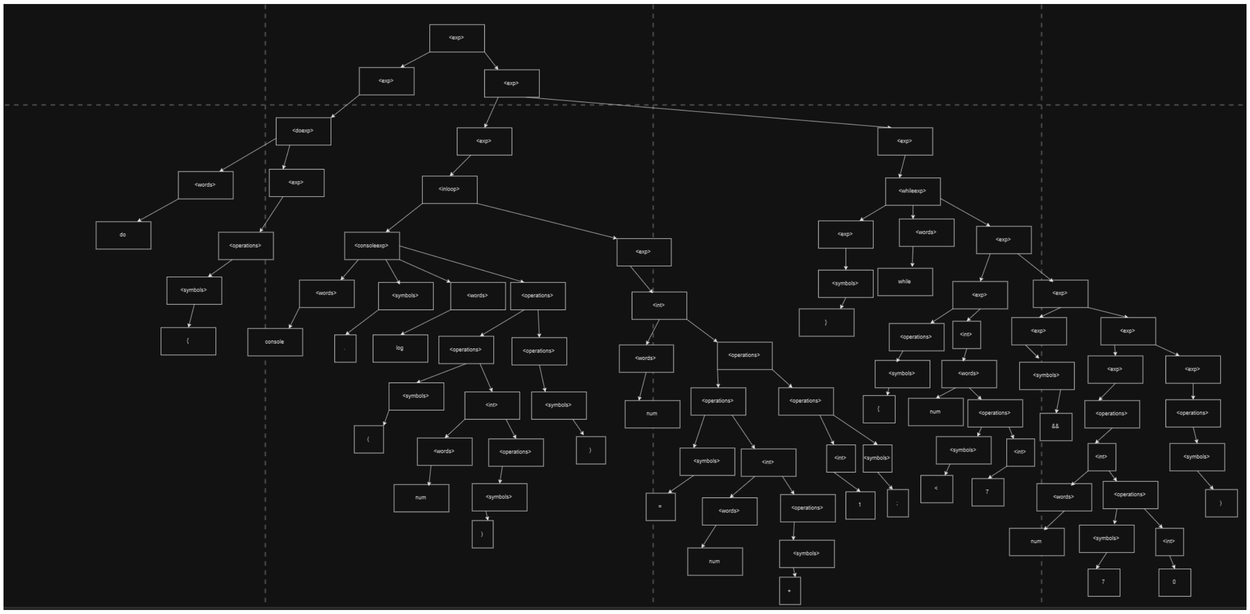
The Design:

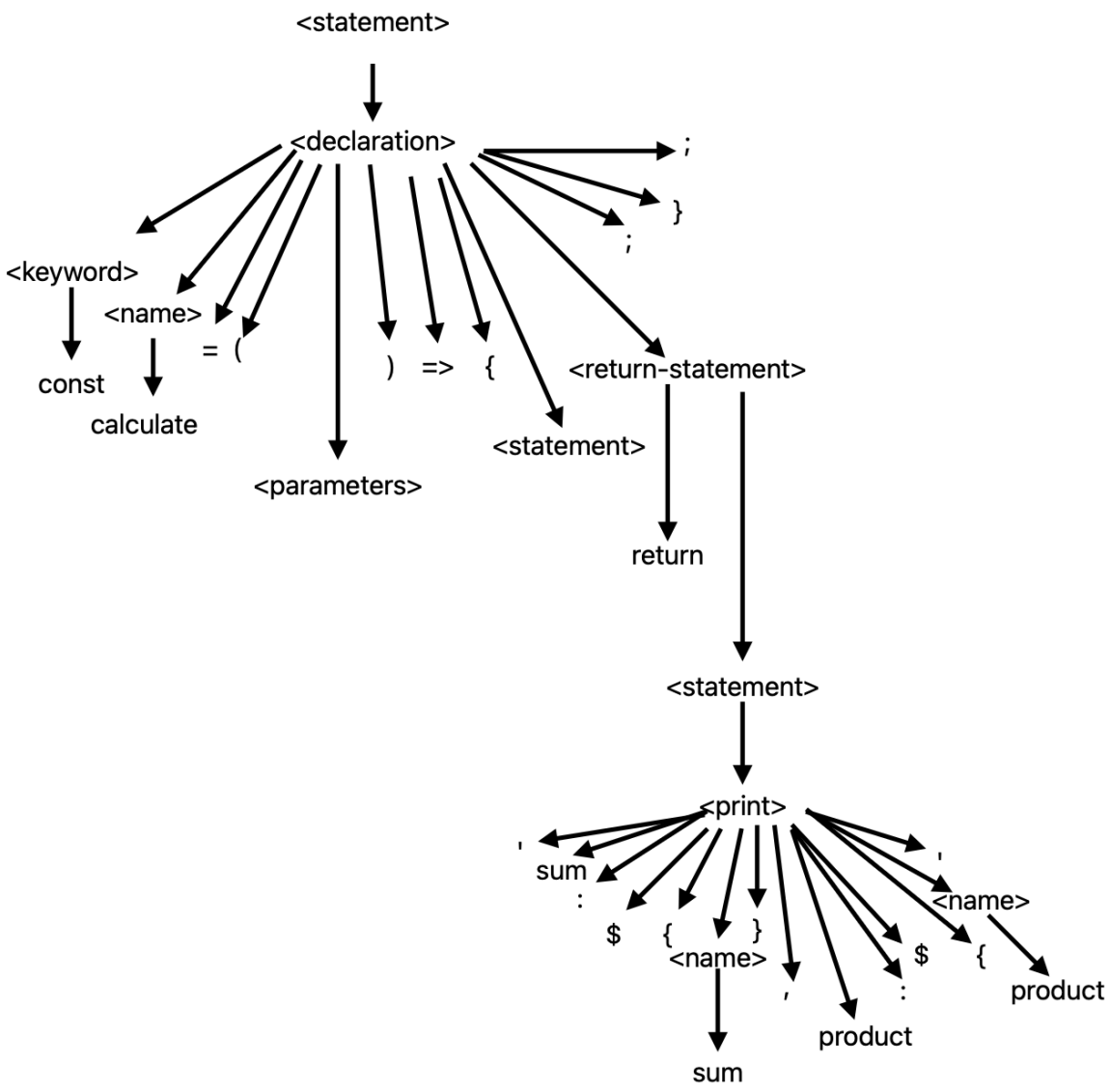
JavaScript's design is a dynamically typed language which means that the values for variables aren't given until runtime based on the values at the time. And JavaScript is also a weak type checking language which means that it recognizes different data types but doesn't use them too strictly. The interpreter chooses when to convert data when reasonable, if we have two lines "let x = 14;" and "let y = '14';" if we try to compare x == y then the return will be true because the interpreters recognizes both are 14 but if we do something like x === y then the return value is false because x is an integer and y is a string. But because of the conversions JavaScript can sometimes run into extremes. JavaScript is also a language that supports Prototypal inheritance. That means that the object that is created can gain more attributes based off of another object which would be the "prototype". The two operations that make this possible are Object.setPrototypeOf() and __proto__. Object.setPrototypeOf() works by having 2 parameters, the first being an object then the second parameter being the prototype. And the second option __proto__ is used like "obj.__proto__ = prototype;". Both of these options do the same thing essentially, they take the attributes from the prototype object and then puts them into the object that we are assigning those values to.

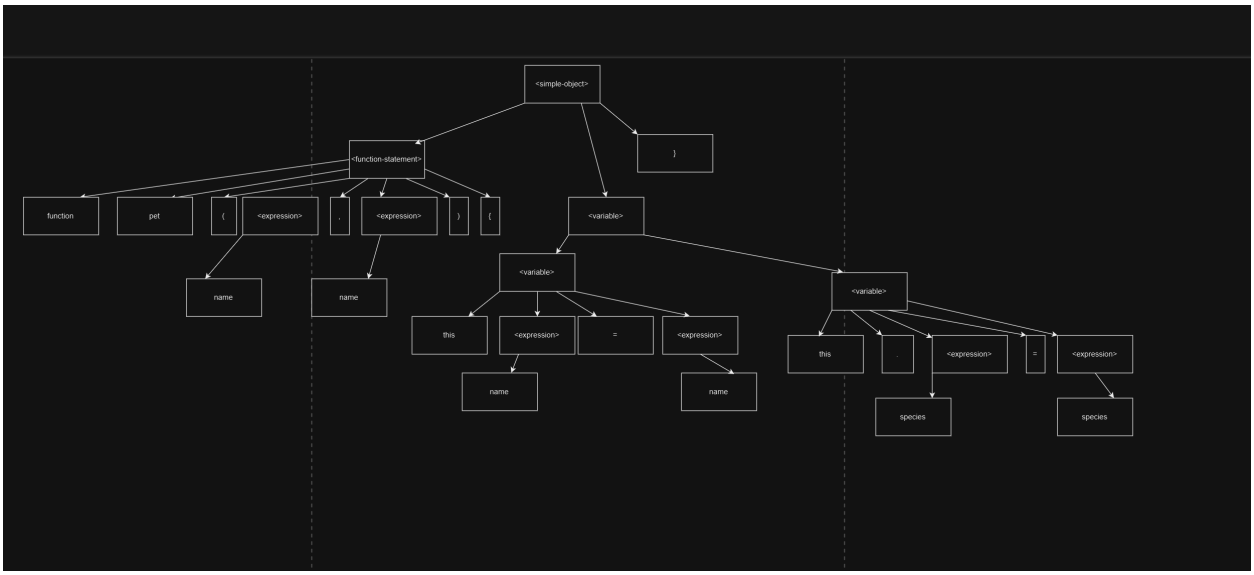
Scoping

Another design of JavaScript are the 3 different scopes that are apart of JavaScript. The first being the block scope which are scopes inside things like if statements, then the function scope for things inside the function they are defined in, and lastly the global scope for things that are defined in the program outside of functions and if blocks. In the. Block scope if a variable is defined either the keyword const or let then the variable can't be accessed from outside the block, but if it is defined with the keyword var that means that the variable can be used outside the block. But for something like the function scope, no matter what the variable is defined with it can't be accessed from outside its function. But for global scope no matter what the variable is defined with, the variable can be accessed from anywhere inside the program, in other functions and any block.

Parse trees:




<statement>
<declaration>
;
}
;
<keyword>
<name>
= (
) => {
<return-statement>
const
calculate
<parameters>
<statement>
return
<statement>
<print>
'
sum
:
$ { }
<name>
,
'
<name>
$ {
:
product
product
sum



Summary

In summary, this project demonstrated the usage of JavaScript to develop a functional and interactive application. Throughout the development process, we applied essential JavaScript principles such as variables, functions, event handling, and DOM manipulation. The project also allowed us to learn debugging, testing, and improving the user experience. Overall, the programs produced successfully serve their intended purpose, with potential future additions such as new capabilities, improved error handling, or a mobile device interface. Overall, the project strengthened our grasp of JavaScript.

Source Cited

Electron. (n.d.). *Electronjs.org*. https://www.electronjs.org/

How to create a desktop app using JavaScript? (2024, October 8). *Geeks for Geeks*. https://www.geeksforgeeks.org/how-to-create-a-desktop-app-using-javascript/

JavaScript: Weak typing. (n.d.). *shared.nav.book*. https://code-basics.com/languages/javascript/lessons/data-types-weak-typing

JavaScript history. (n.d.). *W3 Schools*. https://www.w3schools.com/js/js_history.asp

Maricheva, A. (2023, March 10). A trip back in time: The history of

JavaScript. *SoftTeco*. https://softteco.com/blog/history-of-javascript

Mbathi, C. (n.d.). How to use JavaScript for backend development. *Turing.com*. https://www.turing.com/kb/javascript-for-backend-development

MDN contributors. (2025, March 6). *Dynamic typing*. mdn web docs. https://developer.mozilla.org/en-US/docs/Glossary/Dynamic_typing

MDN contributors. (2025, April 2). *Inheritance and the prototype chain*. mdn web docs. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Inheritance_and_the_prototype_chain

MDN contributors. (2025, April 2). *JavaScript*. mdn web docs. https://developer.mozilla.org/en-US/docs/Web/JavaScript

Prototype inheritance in JavaScript. (2024, December 6). *Geeks for Geeks*. https://www.geeksforgeeks.org/prototype-inheritance-in-javascript/

Sakhniuk, M. (2022, November 4). Two programming paradigms in JavaScript? *JSQ*. https://iq.js.org/questions/javascript/javascript-programming-paradigms

Williams, M. (2024, August 6). What is JavaScript used for? *ComputerScience.org*. https://www.computerscience.org/bootcamps/guides/javascript-uses/