Introduction to JavaScript

Learning Objective

Introduction

JavaScript is a high-level, interpreted, and dynamic programming language used for client-side web development, creating interactive web pages, and implementing complex logic on the front end.

Focus: Basics of JavaScript, Variables, Data types, Special types in JavaScript

Prerequisites: VS Code IDE with Node.js installed

Theme

Imagine you are browsing an online shopping website, and you add an item to your cart. Without refreshing the page, the website immediately updates the cart icon to show the number of items you have added. This is made possible by JavaScript, which allows for dynamic updates to web pages without requiring a full page refresh. JavaScript is also responsible for many interactive website features, such as dropdown menus, pop-ups, and sliders.

Primary Goals

- Understand the basics of JavaScript
- Learn about JavaScript variables and syntax

Introduction to JavaScript

Social media platforms extensively use JavaScript to provide a seamless user experience. One example is Facebook which uses JavaScript for various features on its platform. When you first log in, Facebook uses JavaScript to load your personalized news feed, which includes posts from your friends, pages you follow, and

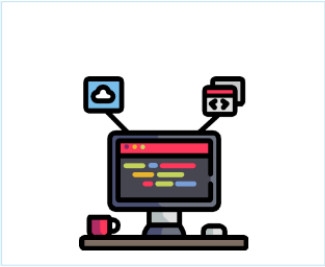
advertisements. Scrolling through the news feed triggers additional JavaScript code, which dynamically loads more posts as you reach the end of the current batch.

Facebook's chat feature also uses JavaScript to provide real-time updates of your conversations, with new messages appearing on the page without needing a manual refresh. The platform also uses JavaScript to handle interactions within posts, such as liking and commenting, display pop-up notifications, and handle user authentication.

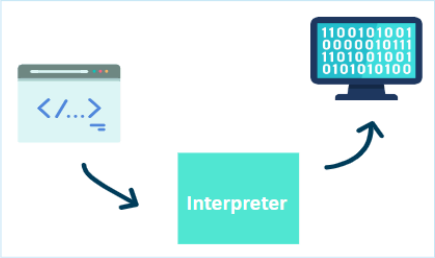
Do you know What JavaScript is?

JavaScript is an interpreted, high-level programming language frequently used to develop dynamic web pages.

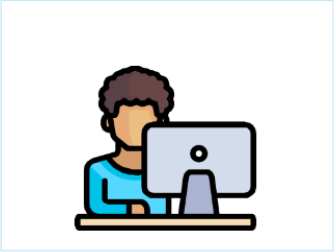
- JavaScript is used when a web page has features that go beyond just displaying information, such as updates, alerts, and actions triggered by button clicks.



Interactive Web Pages



Interpreted Language



Runs on the Client's System

Why Dynamically Typed Language?

In the world of programming, dynamically-typed languages stand out as the ones where the interpreter assigns the variable's type at runtime based on its value at the moment. This feature allows for greater flexibility in coding, as well as faster prototyping and testing.

It's unnecessary to declare the variable type when declaring a variable explicitly.

Need for JavaScript

Limitations and Constraints of the Webpage Created with HTML & CSS:

- This simple static HTML currently lacks responsiveness but includes images, text, buttons, and other UI features.
- The website needs to be utilized from a business or a customer standpoint.

JavaScript is useful because it provides the following advantages:

- Our system is lightning-fast because it runs code locally instead of contacting the server.
- Create highly responsive interfaces and elevate the user experience with ease.
- JavaScript does not require a compilation step. Instead, an interpreter in the browser reads over the JavaScript code, interprets each line, and runs it.
- Enhance functionality on the fly without server latency, providing a seamless experience for your users.

What makes them so well-loved by programmers?

- As of today, there are 1.9 billion websites on the Internet, with 95% of them utilizing JavaScript in some capacity.
- To improve the functionality of certain websites and web applications, try disabling JavaScript access on your web browser. JavaScript is responsible for managing the interactivity of the web, and turning it off may cause some sites to appear less engaging.
- Additionally, developers appreciate it for its user-friendly nature, making it easy to learn and become proficient in. Even with no prior coding experience, one can create stunning projects with the knowledge of JavaScript. For businesses, this translates to a reliable pool of competent JavaScript developers always at the ready.

Other than simplicity, JavaScript is also known for its speed and versatility. It can be used for the following programming projects:

- Add interactive elements to your website
- Develop web and mobile applications
- Create web servers
- Develop games

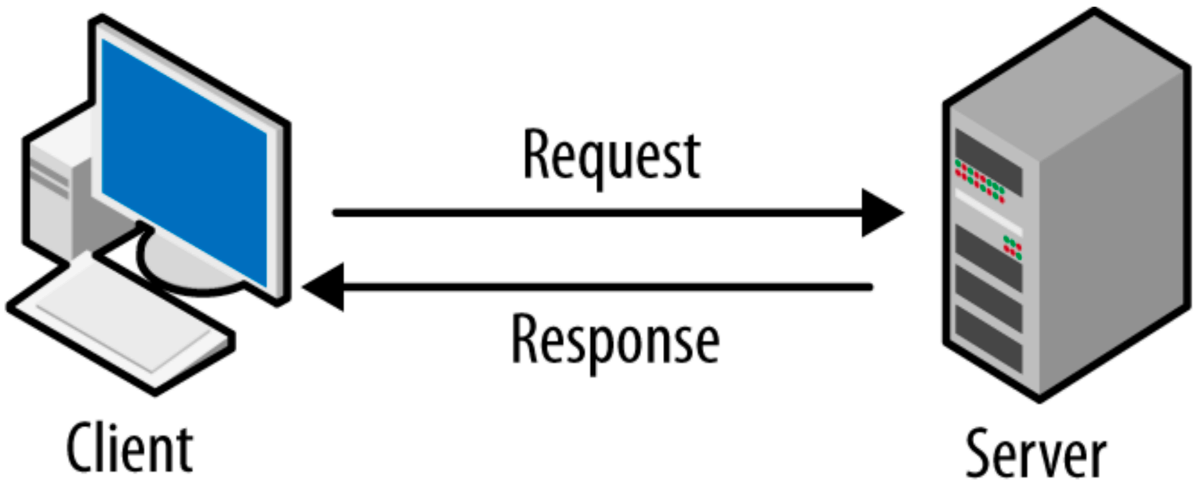
Why is JavaScript Popular?

1. Versatility: JavaScript is primarily used for front-end development, but it can also be used for server-side programming with Node.js. This flexibility allows developers to work across the full stack with a single language.
2. Browser compatibility: JavaScript is supported by all modern web browsers, making it an essential tool for creating interactive and dynamic websites that work across different platforms and devices.
3. Ease of learning: JavaScript has a relatively simple syntax and is considered easier to learn compared to some other programming languages.
4. Large community and ecosystem: JavaScript has a massive community of developers, which means there are abundant resources, tutorials, and tools available for learning and problem-solving. Additionally, there are numerous libraries and frameworks, such as React, Angular, and Vue.js, that make development faster and more efficient.
5. Regular updates and improvements: JavaScript is continuously evolving, with new features being added to the language regularly through the ECMAScript standard. This ensures that JavaScript stays up-to-date with modern programming practices and techniques.

6. Asynchronous programming: JavaScript supports asynchronous programming, which is essential for handling multiple tasks simultaneously, such as handling user input or fetching data from a server without blocking the main thread.

Role of JavaScript in Web Development and UX

Client-Server Cycle



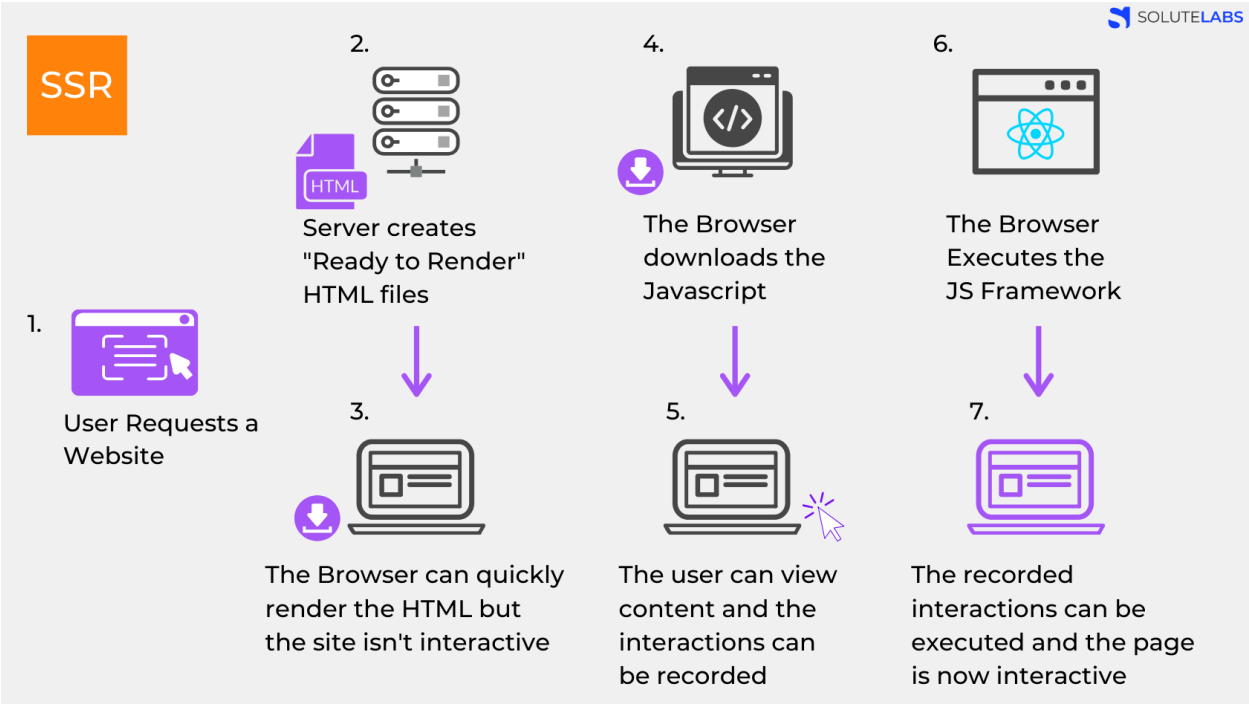
With the help of JavaScript frameworks, it is now possible to render dynamic content directly from the browser by requesting only the necessary content. In this scenario, the server only serves the required base HTML wrapper.

This transformation provided visitors with a seamless user experience due to the minimal loading time of the web page. The page did not reload once loaded, further enhancing the experience.

Server-side Rendering

As discussed earlier, the traditional method of rendering dynamic web content involves the following steps:

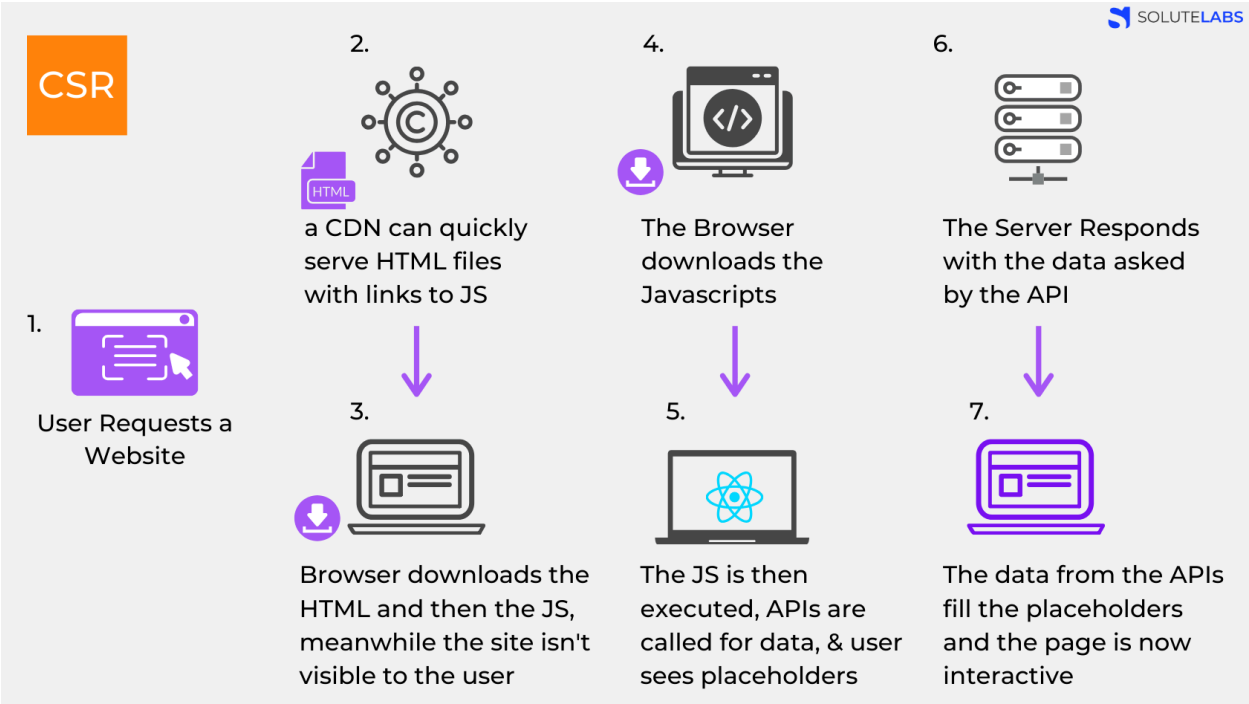
- The user sends a request to a website (usually through a browser).
- The server checks the resource, compiles, and prepares the HTML content by going through server-side scripts within the page.
- After compilation, the HTML is transmitted to the client's web browser to be rendered and displayed.
- The browser downloads the HTML and displays the site to the end user.
- The browser also downloads JavaScript (JS), and as it executes the JS, it makes the page interactive.



Client-side Rendering

The typical process for rendering a web page in a client-side rendering scenario is as follows:

- The user initiates a request to a website, typically through a browser.
- Instead of a server, a Content Delivery Network (CDN) can be utilized to provide static HTML, CSS, and supporting files to the user.
- The browser downloads the HTML, followed by JavaScript. Meanwhile, the user observes a loading symbol.
- Once the browser fetches the JavaScript, it sends requests to the server via AJAX to retrieve dynamic content and processes it to produce the final content.
- When the server responds, the final content is rendered using DOM processing within the user's browser.



Usability of JavaScript in Front-end

A front-end web developer creates the code and markup that is displayed by a web browser when you visit a website. In other words, they determine what you see on a webpage.

- To build a webpage, there are three main components to consider: HTML, CSS, and JavaScript. While HTML provides the structure and content of a site, CSS (Cascading Style Sheets) adds visual appeal, and JavaScript enables interactivity.
- These components work in harmony to create an effective website. This blog post will focus on JavaScript and its practical applications.

Interactivity

JavaScript is a powerful tool for websites. It enables you to create interactive UI components like image sliders, pop-ups, mega menus, form validations, tabs, accordions, and more.

AJAX

You might be thinking of the cleaning product, but AJAX stands for Asynchronous JavaScript and XML in this case. It enables a webpage to communicate with the web server in the background without requiring a page reload.

Cross browser compatibility and Standards compliance

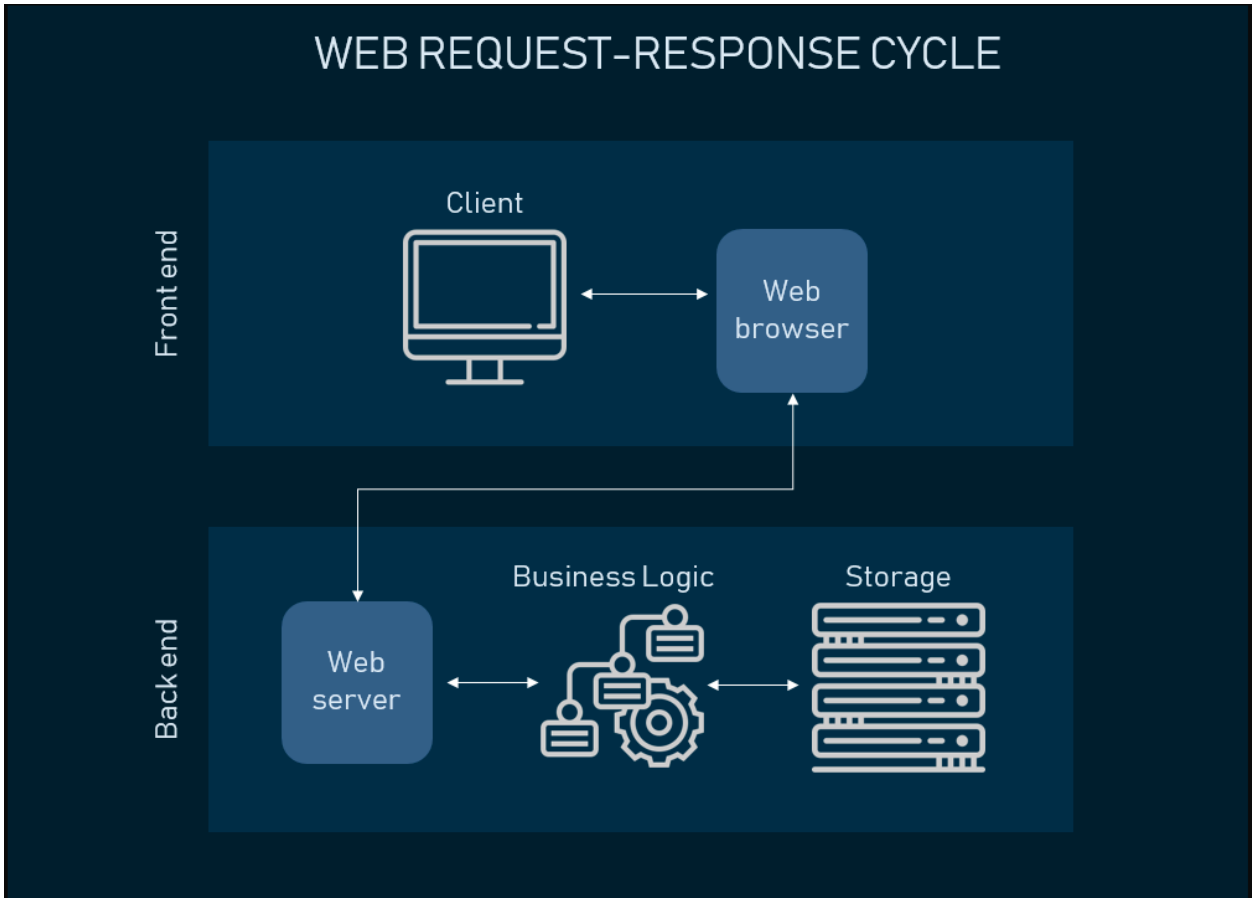
Many different web browsers are available, including Firefox, Chrome, Safari, Opera, and Internet Explorer (versions 10, 11, and Edge), which are compatible with different operating systems and devices. Unfortunately, each browser has its own unique bugs and quirks, and while W3C compliance standards are improving, there are still times when a front-end web developer needs to use JavaScript to resolve issues.

Plugins

There are several plugins that run on JavaScript. If you have visited a website with banner ads, chat support, suggested content, forms, or social sharing, a third-party JavaScript plugin likely powers it. Typically, these plugins have configurable options that require additional setup to function properly. It’s important to comprehend these options. Plugins are generally designed to be easily added to a site with minimal changes.

Frameworks

- A JavaScript framework is essential for rendering complex dynamic interactions on a webpage. It is particularly useful for multi-step form-fills, where certain steps depend on previously entered information and certain data is populated based on previous inputs. A framework can make accomplishing this task easier, and problems can arise quickly.
- Using a JavaScript framework resolves these issues and enables you to complete your form-fill seamlessly, satisfying your clients. While dozens of frameworks are available, the most popular ones (as of this writing) are Google's Angular, Facebook's React, and the open-source Vue.js.
- JavaScript is a fundamental tool for front-end web developers. Without it, dynamic web applications like image carousels and partial page reload that retain your position would not exist.



Usability of JavaScript in the Back-end

The rise in popularity of Node.js has led to an increased usage of JavaScript as a backend language. It is important to grasp the language's basics and general rules to begin working with JavaScript in the backend.

JavaScript Engine

Each web browser has its own JavaScript engine that supports JavaScript scripts so that they can function properly. The main task of a JavaScript engine is to take

JavaScript code and convert it into optimized code, fast that can be interpreted by a browser. Here are the names of the JavaScript engines used in some of the most popular web browsers:

- Chrome: V8
- Firefox: SpiderMonkey
- Safari: JavaScript Core
- Microsoft Edge/Internet Explorer: Chakra/ChakraCore.

Interaction: alert, prompt, confirm

In JavaScript, `alert`, `prompt`, and `confirm` are three built-in functions that allow for simple interaction with users through dialog boxes in the web browser. These dialog boxes are modal, meaning they halt the execution of the script until the user interacts with them.

Here's an overview of each function and its purpose:

1. `alert`: The `alert` function displays a simple message box with a provided message. It is commonly used to show informational messages or to get the user's attention. The message is displayed as a modal dialog box, and the user must click the OK button to dismiss it.
    1. Example:
    2. ```
       alert("Hello, World!");
       ```
2. `prompt`: The `prompt` function displays a dialog box that allows the user to enter input. It takes two arguments: the message to be displayed as a prompt and an optional default value for the input field. The function returns the text entered by the user as a string or `null` if the user cancels the dialog.
    1. Example:

```
const name = prompt("Please enter your name:", "John Doe");
if (name !== null) {
  console.log("Hello, " + name + "!");
```

    2. }
3. `confirm`: The `confirm` function displays a dialog box with a message and two buttons: OK and Cancel. It is typically used to prompt the user for a yes-or-no decision. The function returns `true` if the user clicks OK and `false` if the user clicks Cancel.
    1. Example:

```
const result = confirm("Are you sure you want to delete this item?");
```

    2. ```
       console.log(result);
       ```

These functions are synchronous, meaning that they halt the execution of JavaScript code until the user interacts with the dialog box. While they are useful for simple interactions, they can sometimes interrupt the user experience and are not suitable for

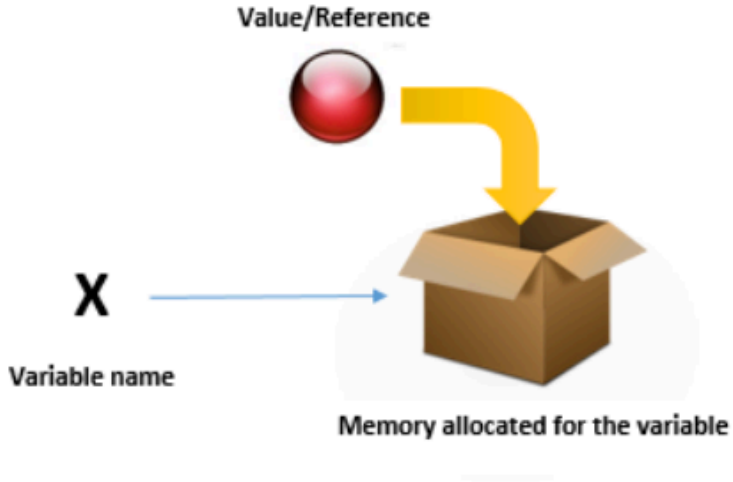
complex user interfaces. In those cases, more advanced techniques using HTML, CSS, and JavaScript event handling may be employed.

Variables

What is a Variable?

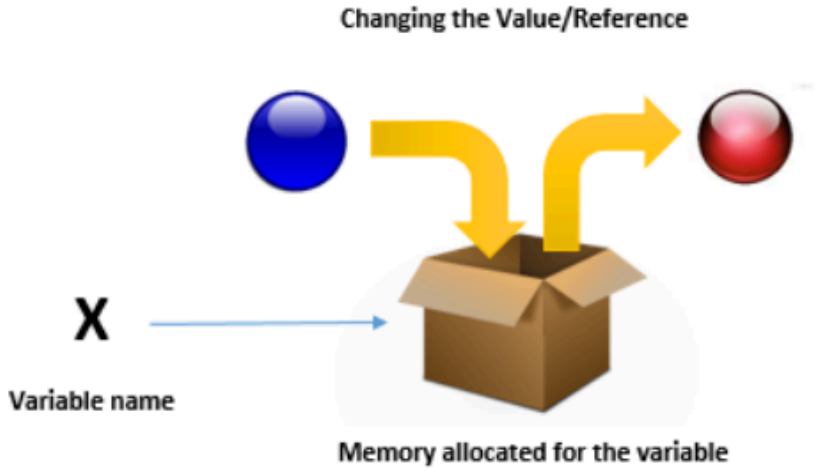
It is a name given (assigned) to a memory location that serves as a temporary data container. Variables are reserved memory locations used to store values.

x = red



Now setting the new value;

x = blue



JavaScript Variable Scope

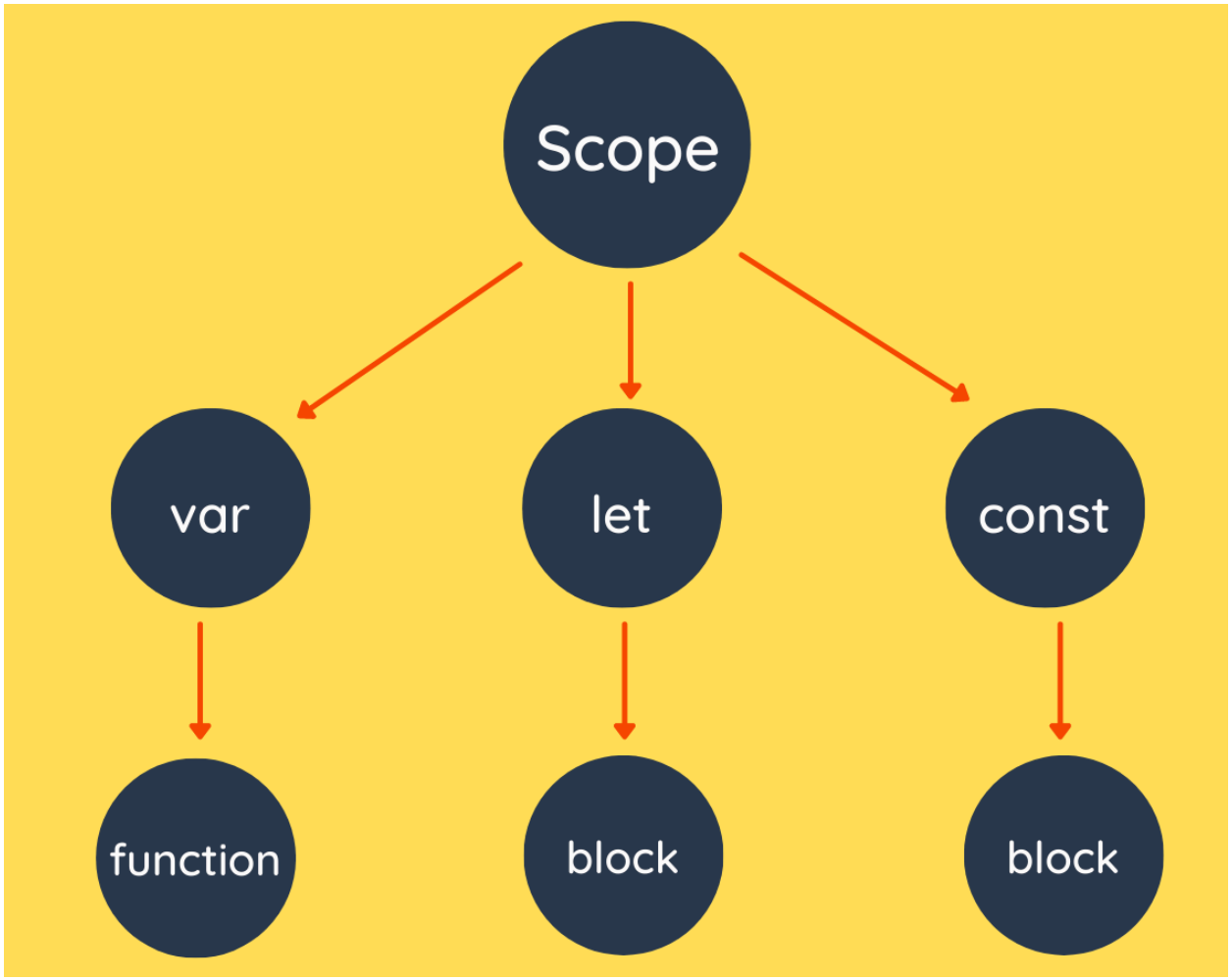
It refers to where it is visible and can be used in your code.

- Global Variables: A global variable can be defined anywhere in your JavaScript code and has a global scope.
- Local Variables: The scope of a local variable is limited to the function in which it is defined, meaning it is not visible outside of that function. Similarly, function parameters are always local to that specific function.

How to declare variables in JavaScript?

JavaScript has three ways to declare variables:

- var (declares mutable variables used before ES6)
- let (declares mutable variables)
- const (declares immutable variables)



“var” Keyword in JavaScript

var is a reserved keyword that is followed by a reference variable name, and the name defined after the keyword can then be used as a pointer to the data in memory.

var is the oldest method used as a variable declaration in JavaScript. Let's declare a variable and initialize it by assigning a value to it with the help of the assignment operator (=):

```
// Declaration and initialization
```

```
var name = "John Doe";
```

Alternatively,

```
// Declaration
var name;
// Initialization

name = "John Doe";
```

Issues with “var”

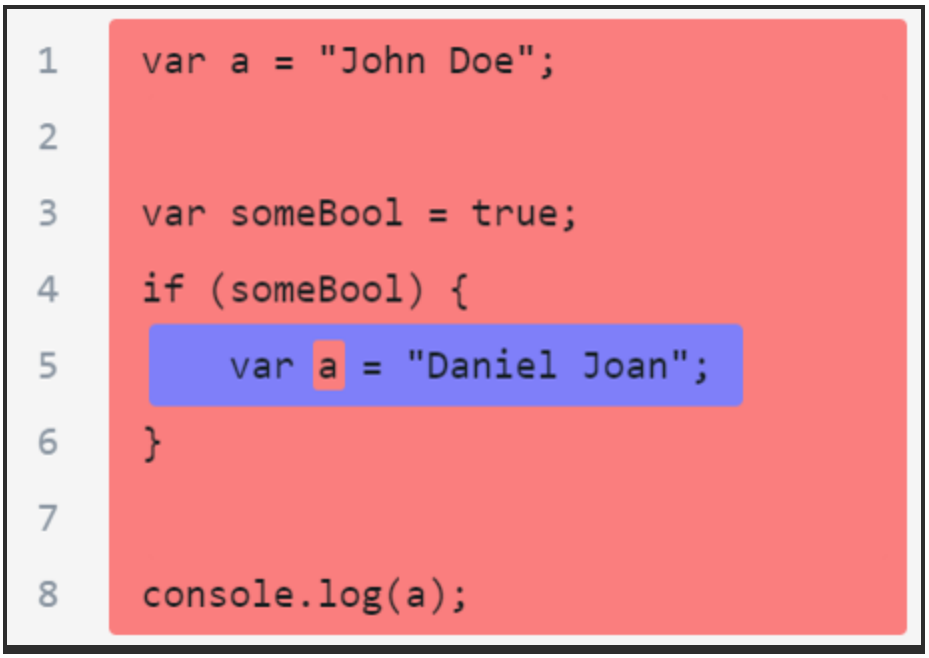
`var` are not block-scoped.

When you declare a variable within a code block, using curly braces (`{}`), its scope "flows out" of the block!

```
var a = "John Doe";

var someBool = true;
if (someBool) {
  var a = "Daniel Joan";
}

console.log(a);
```



```
var a = "John Doe";

var someBool = true;
if (someBool) {
    var a = "Daniel Joan";
}

console.log(a);
```

The `a` that points to "John Doe" is global, and the `a` that points to "Daniel Joan" is defined within a block. However, when we try printing the `a` that's within scope, we run into:

```
Daniel Joan
```

The variable `var` is not block-scoped. Even though we might assume that we have defined a local variable `name` that points to `Daniel Joan`, we have just overwritten the `name` variable that points to `John Doe`.

Visualize this code here

“let” Keyword in JavaScript

The `let` keyword was introduced with `ES6` and has since become the preferred method for declaring variables. It is considered an improvement over var declarations and is block-scoped, meaning that the variables can only be accessed within the immediate block. This circumvents the main issue that can arise from using var.

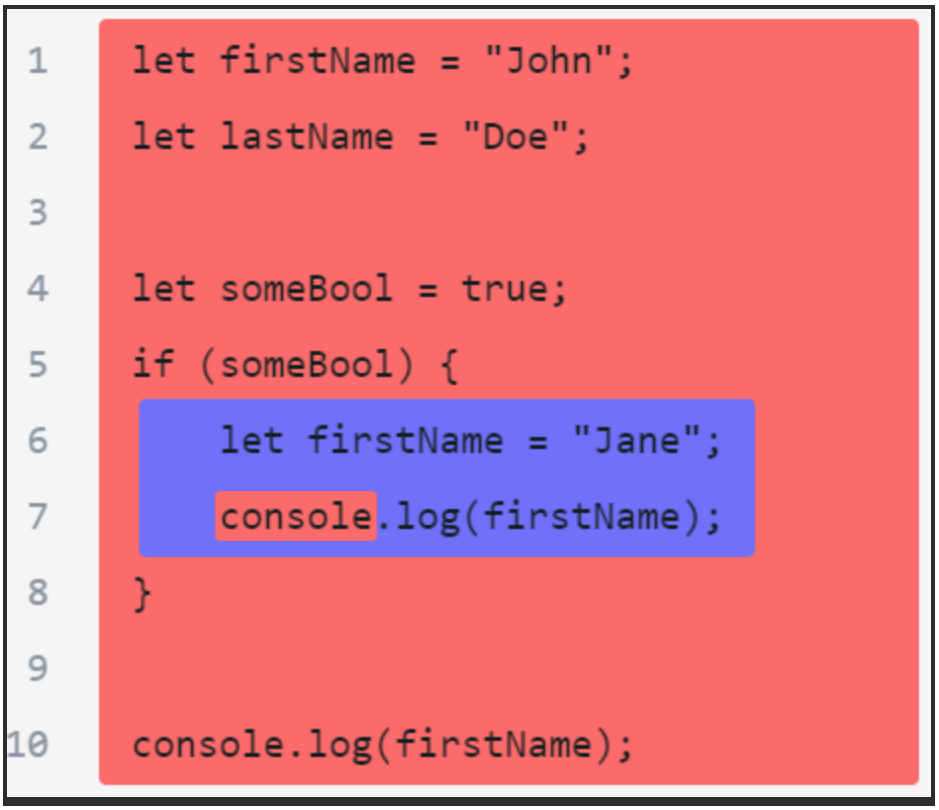
Scope of ”let”

A variable defined with the `let` keyword is only accessible within the block or function in which it is defined:

```
let firstName = "John";
let lastName = "Doe";

let someBool = true;
if(someBool){
    let firstName = "Jane";
    console.log(firstName);
}
```

```
console.log(firstName);
```



```
let firstName = "John";
let lastName = "Doe";

let someBool = true;
if (someBool) {
    let firstName = "Jane";
    console.log(firstName);
}

console.log(firstName);
```

This time, the `firstName` referring to "Jane" and the `firstName` referring to "John" no longer overlap! As a result, the code produces:

```
Jane
```

```
John
```

The `firstName` declared within the block is only available within the block, while the `firstName` declared outside the block is available globally. Since they have different scopes, both instances of `firstName` are treated as different variable references.

[Visualize this code here]

“const” Keyword in JavaScript

The `const` declaration was added in ES6 along with `let`, and it functions similarly to `let`. `const` is used to point to data in memory that holds values that should not change, as the name suggests. Once a `const` variable is assigned, it cannot be reassigned to a different object in memory.

```
const names = "John";
```

```
const names = "Jane";
```

This results in the following:

```
Uncaught SyntaxError: Identifier 'names' has already been declared
```

Scope of const

The `const` keyword defines a variable whose scope is limited to the block defined by curly braces like `let` declarations. However, `const` variables cannot be updated or re-declared, meaning their values remain constant within the scope.

```
const name = "John"
name = "Doe";
```

```
// Uncaught TypeError: Assignment to constant variable.*
```

To sum it up, there are a few key practices to keep in mind when deciding between different options, aside from the obvious need to avoid bugs:

- const is the most recommended choice, followed by let, while var should be avoided.
- Use let if the value it points to is expected to change over time.
- Use const for global, constant values.
- When importing libraries, it's best to use const.

Activity: Code Practice

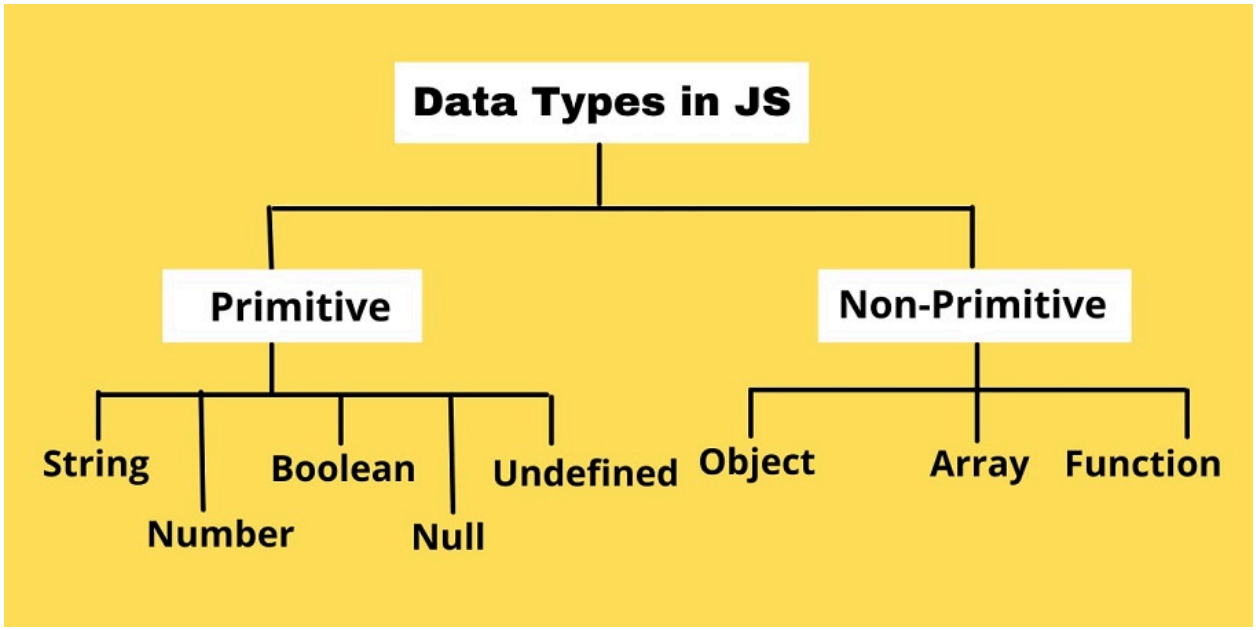
1. Write a program that asks the user to enter their name, age, and favorite color.
2. Store each of these values in a separate variable.
3. Then, using string interpolation, print a message to the console that says, `Your name is {name}, you are {age} years old, and your favorite color is {color}"`
4. Be sure to use the variables you created in the message.

Solution

Data Types

As a beginner in JavaScript, one of the first things you learn is about data types. These are the basic building blocks that you use to create any program. You can think of them as different kinds of containers that hold different kinds of information.

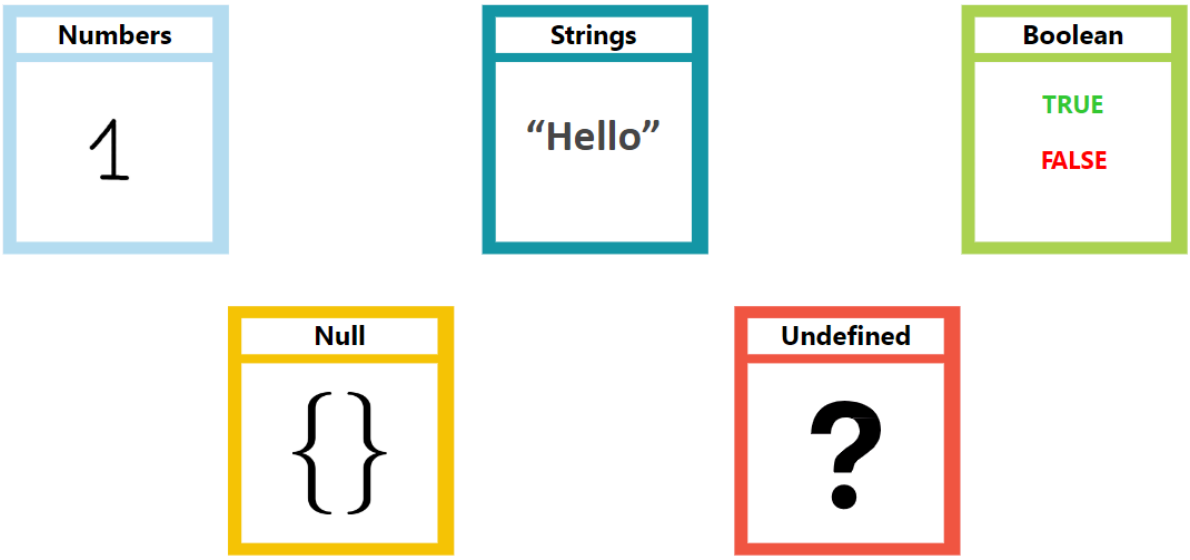
Imagine you're building a house with different rooms. Each room has a specific purpose and contains different items. In JavaScript, each data type is like a room with a specific purpose. The string data type is like a room for storing text, while the number data type is like a room for storing numeric values.



Concept of Data Types

In programming, data types are an essential concept. Having knowledge about the type of a variable is crucial in order to perform operations on it.

There are 5 types of primitive data types in JavaScript :



There are different types of non-primitive data types in JavaScript :

- Reference data types, unlike primitive data types, are dynamic in nature. That is, they do not have a fixed size.
- Most of them are considered objects and therefore have methods.

- JavaScript reference types can contain complex values such as arrays, objects, and functions. Since these values cannot fit within the fixed memory allocated to a variable, the in-memory value of a reference type is the reference to the value's location in memory, represented by a memory address.
- Reference types are also known as complex types or container types.

1. Array: An Array is a collection of values or elements, which can be of any data type that is stored in a contiguous memory location and accessed using an index.
    1. `var names = [ “Mayank”, “Shubham”, “Amrita”]; // Array of strings`

```
var ages = [ 30, 29, 33 ] ; // Array of numbers
```

2. Objects: An object refers to a set of properties consisting of a key and a corresponding value. These values can include primitive data types, functions, or even other objects.

```
var student = {
  roll_no: 34,
  name: "Mayank",
  age: 27,
  city: "Delhi"


};
```

3. Functions: A function is a block of code used to perform a specific task or action and can be reused in different parts of a program.

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

console.log(sum(3, 4)); // 7
```

```
console.log(sum(5, 6)); //11
```

Understanding Special Types in JavaScript

JavaScript Type Conversions

JavaScript Type Conversion is the process of converting one data type to another.

In JavaScript, there are two types of conversions: Implicit and Explicit.

1. Implicit Type Conversion in JavaScript
    1. Implicit Type Conversion is the automatic conversion of data types done by JavaScript during expressions evaluation. This type of conversion occurs when JavaScript expects one data type but gets another. For example, when a string is concatenated with a number, JavaScript implicitly converts the number to a string.
    Example 1: Numeric string used with + gives string type

```
let numStr = "10";
let result = numStr + 5;
```

    2. `console.log(typeof result); // string`
    3. Example 2: Implicit Conversion to Number

```
let numStr = "10";
let result1 = numStr - 5;
let result2 = numStr / 2;
let result3 = numStr * 3;
```

```
console.log(typeof result1); // number
console.log(typeof result2); // number
```

4. `console.log(typeof result3); // number`
5. Example 3: If a Boolean is used, true is 1, false is 0

```
let bool = true;
let result = bool + 5;
console.log(typeof result); // number
```

6. `console.log(result); // 6`
7. Note that JavaScript considers 0 as false and all non-zero numbers as true. If true is converted to a number, then the result is always 1.
   Example 4: null is zero when used with the number

```
let nullVal = null;
let result = nullVal + 5;
console.log(typeof result); // number
```

8. `console.log(result); // 5`

2. Explicit Conversion in JavaScript
    1. Explicit Type Conversion is the developer's manual conversion of one data type to another using built-in conversion functions. It is also known as typecasting.Convert to Number Explicitly
        1. In JavaScript, you can utilize `Number()` to convert numeric strings and Boolean values into numbers.

```
let result;

// string to number
```

```
result = Number('324');
console.log(result); // 324

result = Number('324^(e-1)')
console.log(result); // 32.4

// boolean to number
result = Number(true);
console.log(result); // 1

result = Number(false);
```

2. `console.log(result); // 0`

   `NaN` will be the result if a string is not a valid number.

```
let result;
result = Number('hello');
console.log(result); // NaN

result = Number(undefined);
console.log(result); // NaN

result = Number(NaN);
```

3. `console.log(result); // NaN`

   You can also generate numbers from strings using `parseInt()`, `parseFloat()`, unary operator `+`, and `Math.floor()`.

```
let result;
result = parseInt('20.01');
console.log(result); // 20

result = parseFloat('20.01');
console.log(result); // 20.01

result = +'20.01';
console.log(result); // 20.01
```

```
result = Math.floor('20.01');
```

2. `console.log(result); // 20` Convert to String Explicitly
    1. In JavaScript, you can convert non-string data types to strings using either `String()` or `toString()`.

```
//number to string
let result;
result = String(325);
console.log(result);  // "325"

result = String(2 + 5);
console.log(result); // "7"

//other data types to string
result = String(undefined);
console.log(result); // "undefined"

result = String(null);
console.log(result); // "null"

result = String(true);
console.log(result); // "true"

result = String(NaN);
console.log(result); // "NaN"

//using toString()
result = (324).toString();
console.log(result); // "324"

result = true.toString();
console.log(result); // "true"

result = String(false);
```

3. `console.log(result); // "false"`
   Note: `String()` takes `null` and `undefined` and converts them

to string. However, `toString()` gives errors when `null is passed`.Convert to Boolean Explicitly

1. You can use Boolean() to convert other data types to a Boolean in JavaScript. The conversion rules state that `undefined`, `null`, `0`, `NaN`, and `''` will convert to `false`.

```
let result;
result = Boolean('');
console.log(result); // false

result = Boolean(0);
console.log(result); // false

result = Boolean(undefined);
console.log(result); // false

result = Boolean(null);
console.log(result); // false

result = Boolean(NaN);
```

2. `console.log(result); // false`

   All other values give `true`.

```
result = Boolean(324);
console.log(result); // true

result = Boolean('hello');
console.log(result); // true

result = Boolean(' ');
```

4. `console.log(result); // true`

> According to the Stack Overflow Developer Survey 2022, JavaScript is the most commonly used language among professional developers, with over 67.9% of respondents using it.

Summary

What did we learn?

- JavaScript is a programming language that is interpreted, high-level, and dynamic and is utilized to develop interactive and dynamic web pages and applications.
- It is essential for adding interactivity and dynamic elements to websites, allowing developers to create complex and responsive user interfaces, validate user input, and communicate with back-end systems.
- JavaScript is popular due to its versatility, ability to run on various platforms, large and active community, and rich library and tools ecosystem.
- It powers the internet by enabling dynamic, interactive, and engaging website user experiences, allowing for real-time updates and communication with back-end systems, and making it possible to create complex and responsive web applications.
- Cross-browser compatibility and standards compliance are crucial aspects of web development. JavaScript developers must be aware of compatibility issues

and write code that follows web standards to ensure their websites are accessible and functional for all users.

- Variables in JavaScript are containers that hold a value and can be referenced by a name. They can be declared using the "var", "let", or "const" keywords and store different data types.
- Variables can have different scopes, including global, local, and block-level scopes, depending on where they are defined in the code.

Shortcomings & Challenges

- Overemphasizing syntax and neglecting core concepts such as object-oriented programming, asynchronous programming, and error handling.
- Not addressing performance and security considerations, such as memory leaks and cross-site scripting.

Best practices to follow

- Writing readable, maintainable, and organized code using good naming conventions, indentation, and comments.
- Following coding standards and using linting tools to detect errors and enforce best practices.
- Handling errors and exceptions gracefully with try-catch blocks and appropriate error messages.

Enhance your knowledge

Supercharge your knowledge by exploring the resources provided! 📚

- Symbols in JavaScript: https://thecodebarbarian.com/a-practical-guide-to-symbols-in-javascript.html
- What is JavaScript?: https://www.guru99.com/introduction-to-javascript.html
- An Introduction to JavaScript: https://javascript.info/intro