

Introduction to Javascript

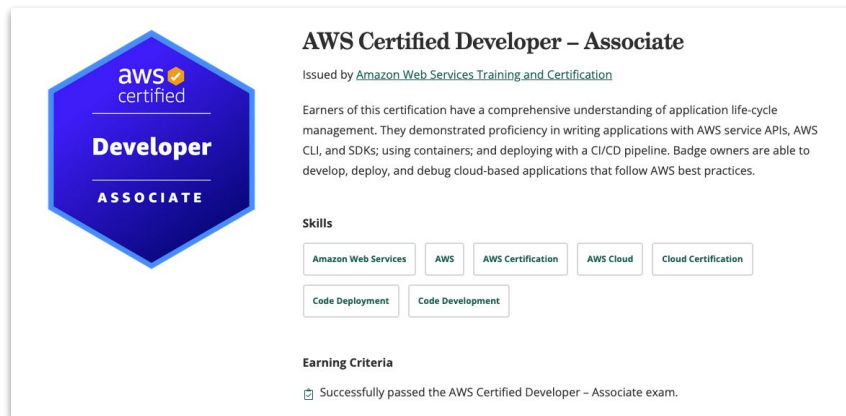
Daniyal Nagori

About Instructor

- Chief Executive Officer - PIAIC
- Director at Panacloud
- Chief Technical Officer - TravelclubIQ



About Instructor





History of Javascript

Creation At Netscape

Creation at Netscape

The first web browser with a graphical user interface, Mosaic, was released in 1993. Accessible to non-technical people, it played a prominent role in the rapid growth of the nascent World Wide Web. The lead developers of Mosaic then founded the Netscape corporation, which released a more polished browser, Netscape Navigator, in 1994. This quickly became the most-used.



Creation at Netscape

During these formative years of the Web, web pages could only be static, lacking the capability for dynamic behavior after the page was loaded in the browser. There was a desire in the flourishing web development scene to remove this limitation, so in 1995, Netscape decided to add a scripting language to Navigator. They pursued two routes to achieve this: collaborating with Sun Microsystems to embed the Java programming language, while also hiring Brendan Eich to embed the Scheme language.

A yellow square logo with the letters 'JS' in a bold, black, sans-serif font, representing JavaScript.

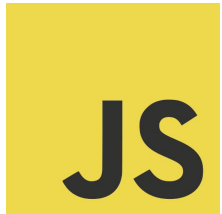
Creation at Netscape

Netscape management soon decided that the best option was for Eich to devise a new language, with syntax similar to Java and less like Scheme or other extant scripting languages. Although the new language and its interpreter implementation were called LiveScript when first shipped as part of a Navigator beta in September 1995, the name was changed to JavaScript for the official release in December.

The logo consists of a solid yellow square. Inside the square, the letters 'JS' are written in a bold, black, sans-serif font. The 'J' and 'S' are connected, with the 'S' having a slightly larger, more stylized appearance.

Creation at Netscape

The choice of the JavaScript name has caused confusion, implying that it is directly related to Java. At the time, the dot-com boom had begun and Java was the hot new language, so Eich considered the JavaScript name a marketing ploy by Netscape.



Adoption At Microsoft

Adoption by Microsoft

Microsoft debuted Internet Explorer in 1995, leading to a browser war with Netscape. On the JavaScript front, Microsoft reverse-engineered the Navigator interpreter to create its own, called JScript.



Adoption by Microsoft

JScript was first released in 1996, alongside initial support for CSS and extensions to HTML. Each of these implementations was noticeably different from their counterparts in Navigator. These differences made it difficult for developers to make their websites work well in both browsers, leading to widespread use of "best viewed in Netscape" and "best viewed in Internet Explorer" logos for several years

A yellow square logo with the letters 'JS' in a bold, black, sans-serif font.

The Rise of JScript

The rise of JScript

In November 1996, Netscape submitted JavaScript to Ecma International, as the starting point for a standard specification that all browser vendors could conform to. This led to the official release of the first ECMAScript language specification in June 1997.

The standards process continued for a few years, with the release of ECMAScript 2 in June 1998 and ECMAScript 3 in December 1999. Work on ECMAScript 4 began in 2000.

A yellow square logo with the letters 'JS' in a bold, black, sans-serif font.

The rise of JScript

Meanwhile, Microsoft gained an increasingly dominant position in the browser market. By the early 2000s, Internet Explorer's market share reached 95%. This meant that JScript became the de facto standard for client-side scripting on the Web.

Microsoft initially participated in the standards process and implemented some proposals in its JScript language, but eventually it stopped collaborating on Ecma work. Thus ECMAScript 4 was mothballed.

A yellow square logo with the letters 'JS' in a bold, black, sans-serif font.

Growth and standardization

Growth and standardization

During the period of Internet Explorer dominance in the early 2000s, client-side scripting was stagnant. This started to change in 2004, when the successor of Netscape, Mozilla, released the Firefox browser. Firefox was well received by many, taking significant market share from Internet Explorer.



Growth and standardization

In 2005, Mozilla joined ECMA International, and work started on the ECMAScript for XML (E4X) standard. This led to Mozilla working jointly with Macromedia (later acquired by Adobe Systems), who were implementing E4X in their ActionScript 3 language, which was based on an ECMAScript 4 draft. The goal became standardizing ActionScript 3 as the new ECMAScript 4. To this end, Adobe Systems released the Tamarin implementation as an open source project. However, Tamarin and ActionScript 3 were too different from established client-side scripting, and without cooperation from Microsoft, ECMAScript 4 never reached fruition.

A yellow square containing the letters 'JS' in a bold, black, sans-serif font, representing JavaScript.

Growth and standardization

Meanwhile, very important developments were occurring in open-source communities not affiliated with ECMA work. In 2005, Jesse James Garrett released a white paper in which he coined the term Ajax and described a set of technologies, of which JavaScript was the backbone, to create web applications where data can be loaded in the background, avoiding the need for full page reloads. This sparked a renaissance period of JavaScript, spearheaded by open-source libraries and the communities that formed around them. Many new libraries were created, including jQuery, Prototype, Dojo Toolkit, and MooTools.

A yellow square logo with the letters 'JS' in a bold, black, sans-serif font.

Growth and standardization

Google debuted its Chrome browser in 2008, with the V8 JavaScript engine that was faster than its competition. The key innovation was just-in-time compilation (JIT), so other browser vendors needed to overhaul their engines for JIT.

In July 2008, these disparate parties came together for a conference in Oslo. This led to the eventual agreement in early 2009 to combine all relevant work and drive the language forward. The result was the ECMAScript 5 standard, released in December 2009.

A yellow square logo with the letters 'JS' in a bold, black, sans-serif font.



Reaching maturity

Reaching maturity

Ambitious work on the language continued for several years, culminating in an extensive collection of additions and refinements being formalized with the publication of ECMAScript 6 in 2015.

The creation of Node.js in 2009 by Ryan Dahl sparked a significant increase in the usage of JavaScript outside of web browsers. Node combines the V8 engine, an event loop, and I/O APIs, thereby providing a stand-alone JavaScript runtime system. As of 2018, Node had been used by millions of developers, and npm had the most modules of any package manager in the world.

A yellow square logo with the letters 'JS' in a bold, black, sans-serif font.

Reaching maturity

The ECMAScript draft specification is currently maintained openly on GitHub, and editions are produced via regular annual snapshots. Potential revisions to the language are vetted through a comprehensive proposal process. Now, instead of edition numbers, developers check the status of upcoming features individually.

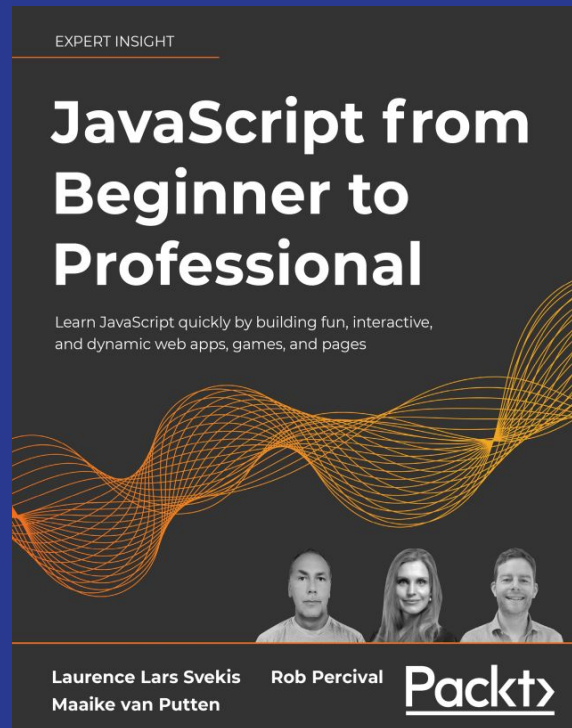


Reaching maturity

The current JavaScript ecosystem has many libraries and frameworks, established programming practices, and substantial usage of JavaScript outside of web browsers. Plus, with the rise of single-page applications and other JavaScript-heavy websites, several transpilers have been created to aid the development process.

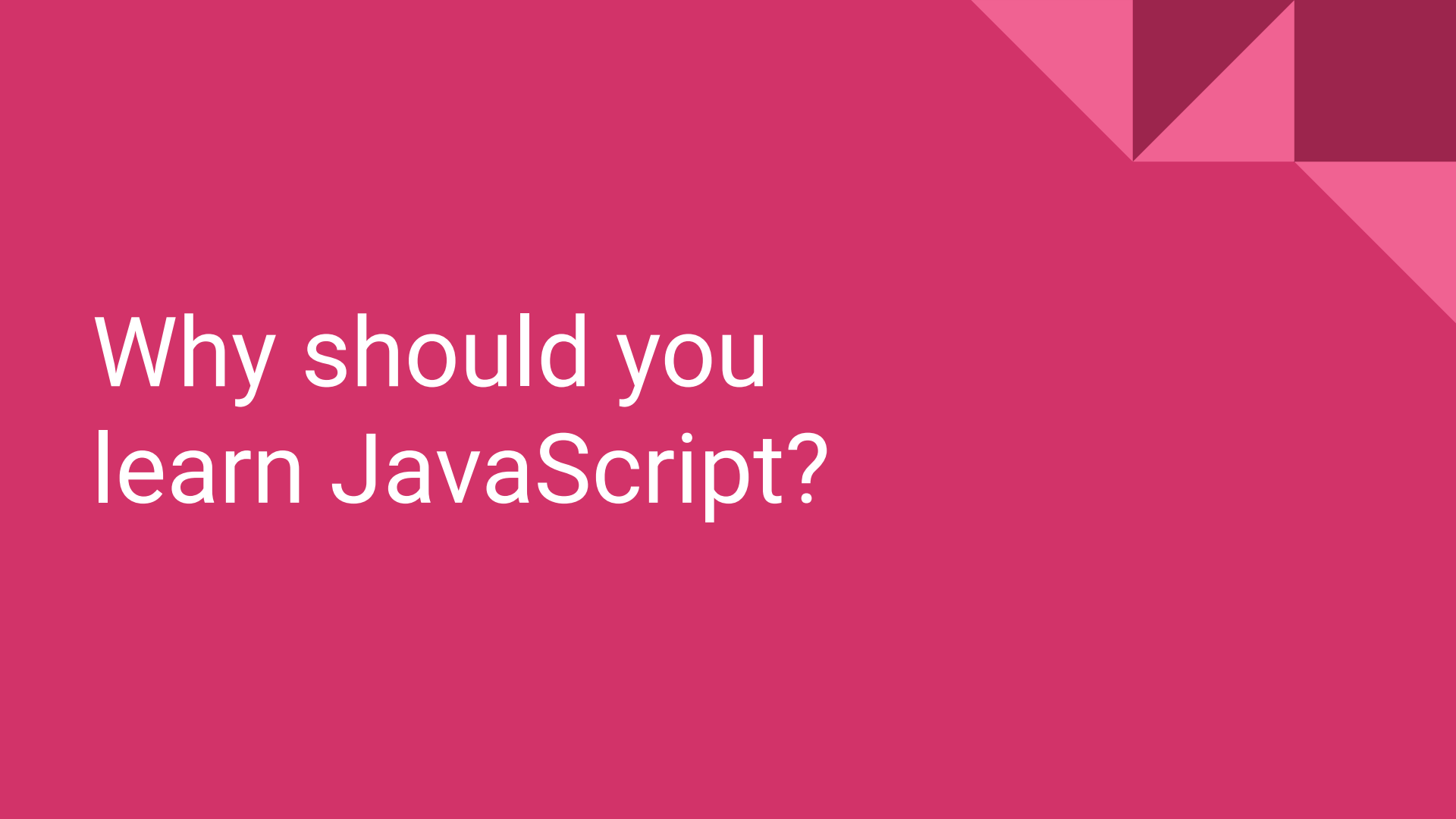


Reference Book: “JavaScript from Beginner to Professional”



Getting Started with JavaScript

Chapter 1

The background is a solid pink color. In the top right corner, there is a decorative pattern of overlapping triangles in various shades of pink and magenta, creating a modern, geometric look.


Why should you
learn JavaScript?

Why should you learn JavaScript?

- JavaScript originates from 1995, and is often considered the most widely used programming language.
- JavaScript is the language that web browsers support and understand.
- You have everything you need to interpret it already installed on your computer if you have a web browser and text editor.
- JavaScript is a great programming language for beginners
- Most advanced software developers will know at least some JavaScript because they will have run into it at some point.



Why should you learn JavaScript?

- You can start building really cool apps using JavaScript sooner than you could imagine.
 - JavaScript can be used for programming for the web browser, but also the logic layer of code that we cannot see (such as communication with the database) of an application can be programmed in JavaScript, along with games, automation scripts, and a plethora of other purposes.
 - JavaScript can also be used for different programming styles, by which we mean ways to structure and write code unlike other languages which restrict the programming paradigm such as **OOP** or **FP**
- 

Why should you learn JavaScript?

- There are a ton of libraries and frameworks you can use once you get the basics of JavaScript down.
- These libraries and frameworks will really enhance your software life and make it a lot easier and possible to get more done in less time.
 - Examples of these great libraries and frameworks include React, Vue.js, jQuery, Angular, and Node.js.
- There won't be a problem for which you cannot find a solution on the internet.




Setting up your environment

Setting up your environment

- There are many ways in which you can set up a JavaScript coding environment. Such as:
 - Integrated Development Environment (IDE). Example: VS Code, Sublime Text, Atom, etc.
 - Web browser. Example: Chrome, Firefox, etc.
 - Online editor (optional). Example: StackBlitz, Replit, etc.





How does the
browser understand
JavaScript?

How does the browser understand JavaScript?

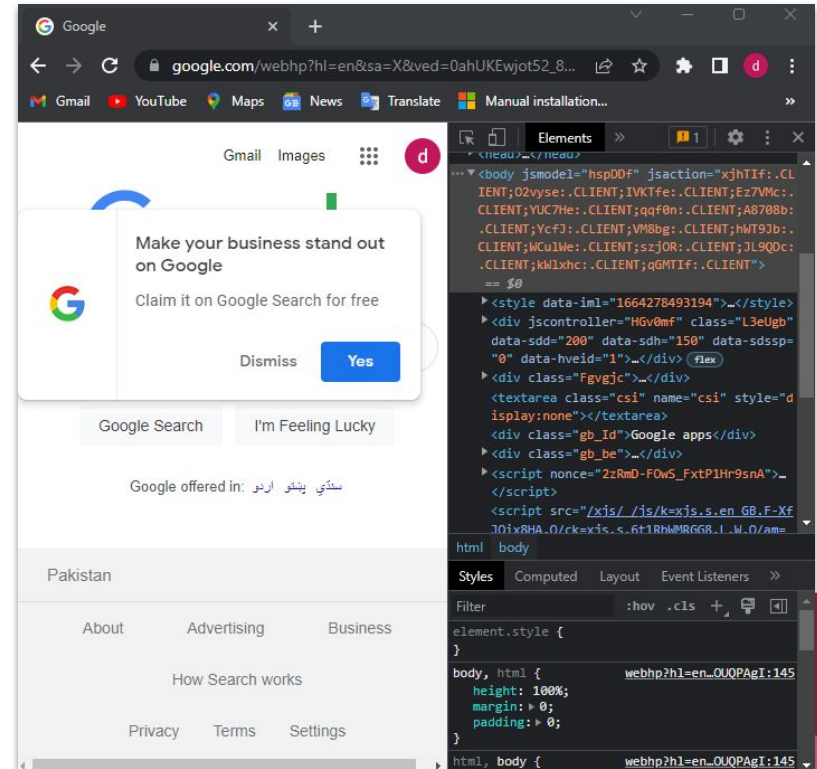
- JavaScript is an interpreted language, which means that the computer understands it while running it. Some languages get processed before running, this is called compiling, but not JavaScript. The computer can just interpret JavaScript on the fly. The "engine" that understands JavaScript will be called the interpreter here.
- Browsers use **ECMAScript** to support JavaScript (in addition to some other topics such as **Document Object Model (DOM)**).



Using the browser console

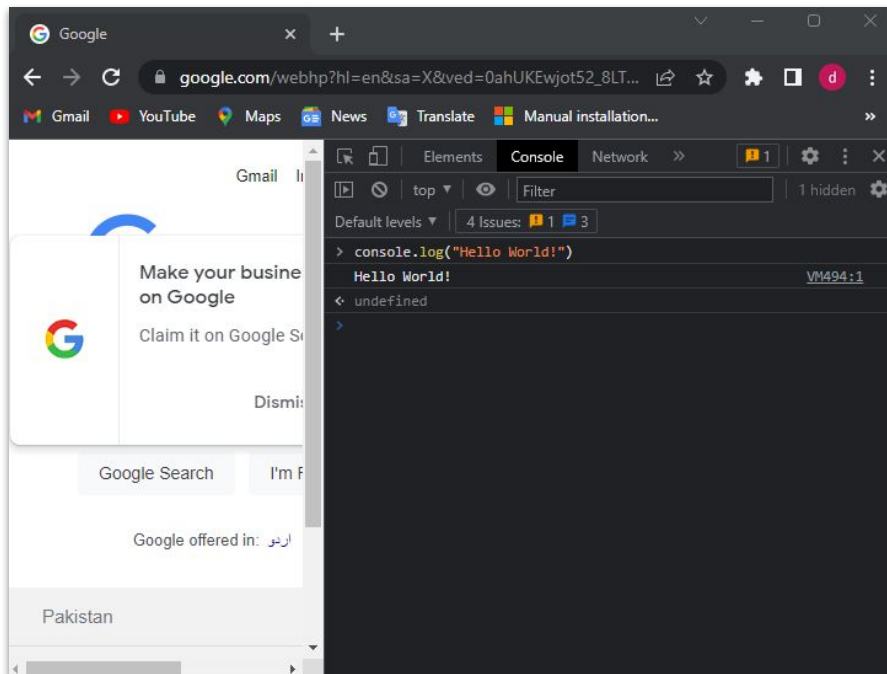
Using the browser console

- If you hit F12 on a Windows computer while you are in the web browser, or you right-click and select Inspect on macOS systems, you will see a screen appear, similar to the one in the following screenshot.
- There are multiple tabs in this panel for inspecting the page.



Using the browser console

- The console is used by developers to log what is going on and do any debugging.
- Click on the **console** tab and type `console.log("Hello world!");` And press Enter key.



Adding JavaScript to a web page

Adding JavaScript to a web page

- There are two ways to link JavaScript to a web page.
 - The first way is to type the JavaScript directly in the HTML between two <script> tags.

```
<html>  
  <script type="text/javascript">  
    alert("Hello World!");  
  </script>  
</html>
```

- The second way is to create a file with extension of .js and link it to our web page.

```
<html>  
  <script type="text/javascript" src="hello_world.js"></script>  
</html>
```



Writing JavaScript code

Writing JavaScript code

- Formatting code:
 - Code needs to be formatted well. If you have a long file with many lines of code and you didn't stick to a few basic formatting rules, it is going to be hard to understand what you've written.
 - The two most important rules for formatting the code are indentations and semicolons.

```
1 var variable1 = 1; var variable2 = 2; var variable3 = 3
2 if (variable1 === 1) {
3   console.log(variable1)
4   if (variable2 === 1) {
5     console.log(variable2)
6   } else {
7     console.log(variable3)
8   }
9 }
```

```
1 var variable1 = 1;
2 var variable2 = 2;
3 var variable3 = 3;
4
5 if (variable1 === 1) {
6   console.log(variable1);
7   if (variable2 === 1) {
8     console.log(variable2);
9   } else {
10    console.log(variable3);
11  }
12 }
```

Writing JavaScript code

- Code comments:

- With comments, you can tell the interpreter to ignore some lines of the file. They won't get executed if they are comments.
- Adding comments to specific parts of the code to explain what is happening or why a certain choice has been made.

```
1  // This line is commented out and the interpreter will ignore it.  
2  
3  /**  
4   * this is a multi-line comment.  
5   * comments to specific parts of the code to explain what is happening  
6   */  
7
```



JavaScript Essentials

Chapter 2

Variables

Variables

- Variable means anything that can vary.
- A JavaScript variable is simply **a name of storage location**.
- A variable must have a unique name.



Variables

- Variables are values in your code that can represent different values each time the code runs.
- The first time you create a variable, you declare it. And you need a special word for that: `let` , `var` , Or `const` .

Example: `let firstname = "Ali";`

- The commonly used naming conventions used for **variables** are camel-case.

Example: `let firstName = "Ali";`



Variables Scope

- LOCAL
- GLOBAL



Variables Names

- A variable name can't contain any spaces
- A variable name can contain only letters, numbers, dollar signs, and underscores.
- The first character must be a letter, or an underscore (-), or a dollar sign (\$).
- Subsequent characters may be letters, digits, underscores, or dollar signs.
- Numbers are not allowed as the first character of variable.

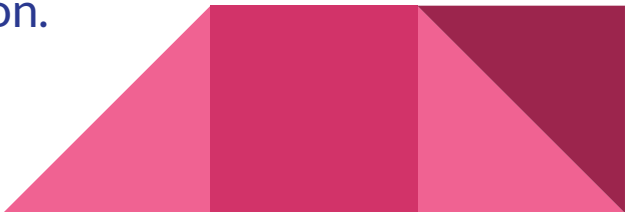


Comments

- Single line Javascript comments **start with two forward slashes (//)**.
- All text after the two forward slashes until the end of a line makes up a comment
- Even when there are forward slashes in the commented text.
- Multi-line Comments
- Multi-line comments start with `/*` and end with `*/`.
- Any text between `/*` and `*/` will be ignored by JavaScript.



Statements

- A computer program is a list of "instructions" to be "executed" by a computer.
 - In a programming language, these programming instructions are called statements.
 - A JavaScript program is a list of programming statements.
 - JavaScript applications consist of statements with an appropriate syntax. A single statement may span multiple lines. Multiple statements may occur on a single line if each statement is separated by a semicolon.
- 


Let, Var, Const

Hoisting of var

Hoisting is a JavaScript mechanism where variables and function declarations are moved to the top of their scope before code execution.

Hoisting of let

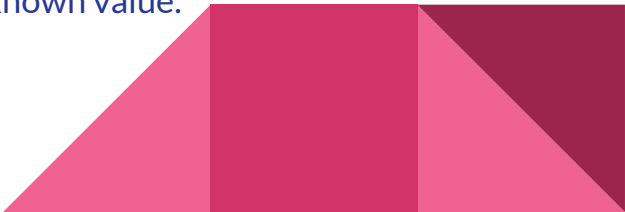
Just like `var`, `let` declarations are hoisted to the top. Unlike `var` which is initialized as `undefined`, the `let` keyword is not initialized. So if you try to use a `let` variable before declaration, you'll get a Reference Error.



Primitive data types

Primitive data types

- String
 - A string is used to store a text value.
Example: `let firstName = "Ali";`
- Number
 - A number is used to store a numeric value.
Example: `let score = 25;`
- Boolean
 - A boolean is used to store a value that is either `true` or `false`.
Example: `let isMarried = false;`
- Undefined
 - An undefined type is either when it has not been defined or it has not been assigned a value.
Example: `let unassigned;`
- Null
 - null is a special value for saying that a variable is empty or has an unknown value.
Example: `let empty = null;`



Template Literals

A new and fast way to deal with strings is **Template Literals** or **Template String**.

How we were dealing with strings before ?

```
var myName = "daniyal" ;
```

```
var hello = "Hello " + myName ;
```

```
console.log(hello); //Hello daniyal
```



Template Literals

What is Template literals ?

As we mentioned before , it's a way to deal with strings and specially dynamic strings ; so you don't need to think more about what's the next quote to use single or double.

How to use Template literals

It uses a `backticks` to write string within it.



Analyzing and modifying data types

Analyzing and modifying data types

- You can check the type of a variable by entering **typeof**.

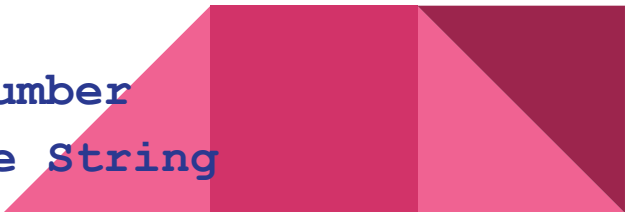
Example:

```
let testVariable = 1;  
console.log(typeof testVariable);
```

- The variables in JavaScript can change types. Sometimes JavaScript does this automatically.

Example:

```
let v1 = 2;  
let v2 = "2";  
console.log(v1 * v2); // 4 ← Type Number  
console.log(v1 + v2); // "22" ← Type String
```



Analyzing and modifying data types

- There are three conversion methods:
 - `String()` ← converts to string type
 - `Number()` ← converts to number type
 - `Boolean()` ← converts to boolean type



Operators

Operators

- Arithmetic operators:

- Addition

Example:

- ```
let n1 = 1;
let n2 = 2;
console.log(n1 + n2); // 3
```
  - ```
let str1 = "1";  
let str2 = "2";  
console.log(str1 + str2); // "12"
```



Operators

- Arithmetic operators:

- Subtraction

Example:

- ```
let n1 = 5;
let n2 = 2;
console.log(n1 - n2); // 3
```

- Multiplication

Example:

- ```
let n1 = 5;  
let n2 = 2;  
console.log(n1 * n2); // 10
```



Operators

- Arithmetic operators:

- Division

Example:

- `let n1 = 4;`
`let n2 = 2;`
`console.log(n1 / n2); // 2`

- Exponentiation

Example:

- `let n1 = 2;`
`let n2 = 2;`
`console.log(n1 ** n2); // 4`



Operators

- Arithmetic operators:

- Modulus

Example:

- `let n1 = 10;`
`let n2 = 3;`
`console.log(n1 % n2); // 1`



Operators

- Assignment operators:
 - Assignment operators are used to assign values to variables.

Example:

- ```
let n = 5;
console.log(n); // 5
n += 5;
console.log(n); // 10
n -= 5;
console.log(n); // 5
```





# Operators

- Comparison operators:

- Comparison operators are used to compare values of variables.

Example:

- ```
let n = 5;
console.log(n == 5); // true
console.log(n === 5); // true
console.log(n != 5); // false
console.log(n > 8); // false
console.log(n < 8); // true
console.log(n >= 8); // false
console.log(n <= 8); // true
```

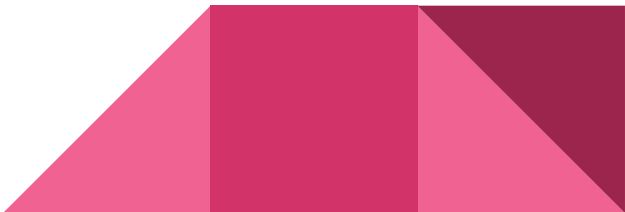


Operators

- Logical operators:
 - Logical operators are used to combine multiple conditions in one.

Example:

```
■ let n = 5;
  console.log(n >= 5 && n < 10); // true
  console.log(n > 5 && n < 10); // false
  console.log(n >= 5 || n < 10); // true
  console.log(n > 5 || n < 10); // true
  console.log(!(n < 10)); // false
  console.log(!(n > 10)); // true
```





Thank You