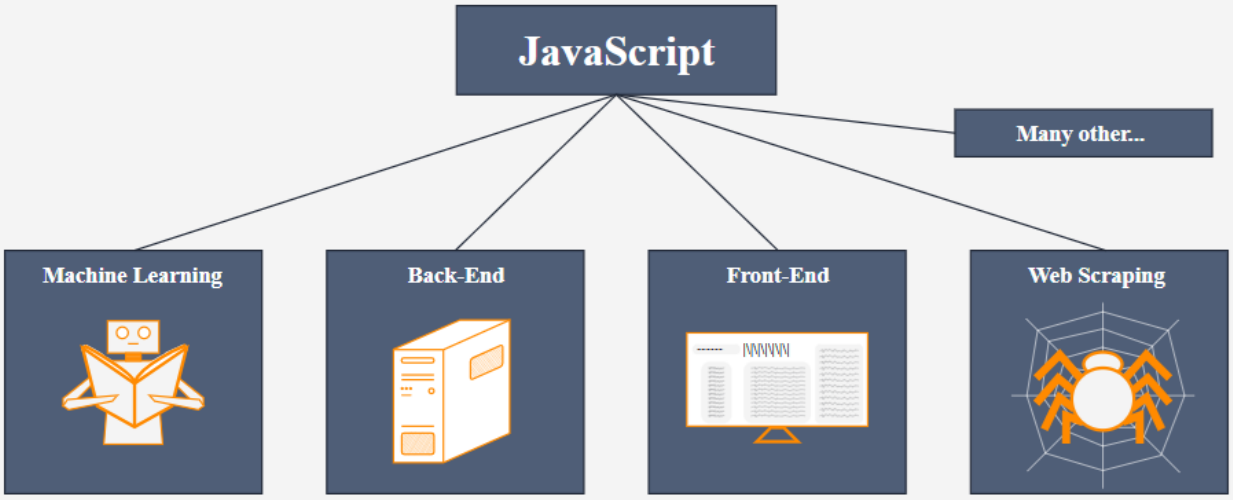
Introduction to JavaScript

Source:
https://codefinity.com/courses/v2/2cd68d01-3aa7-4429-969c-7eea851507a7/3d939211-c5c4-45d1-b569-3c63c42c6265/df740c34-f64a-46ad-ab0d-a4d0e5ba8565

Discover the fundamentals of JavaScript, including its purpose and syntax. Learn how to work with the console for output and use comments to organize and document your code. These foundational skills will prepare you for a deeper exploration of JavaScript's capabilities.

1. About JavaScript

JavaScript (JS) is one of the most widely adopted programming languages. Its applications span various domains within the IT industry: Front-End, Back-End, Machine Learning, Web Scraping, Mobile Apps, Desktop Apps, and even Games. However, its primary focus remains on Front-End development.



The Significance of Front-End Development

Browsers come equipped with an integrated JavaScript interpreter. This unique feature allows JavaScript code to execute directly within the browser, making it an indispensable tool for Front-End Development. This is the only programming language that can replace its role in this domain.

This ability to execute code in the browser empowers the creation of websites that are not just static but dynamic and highly interactive.

What is the JavaScript Interpreter?

In essence, the JavaScript interpreter serves as a program that reads your code and dutifully carries out your commands.

What is the JavaScript Interpreter?

In essence, the JavaScript interpreter serves as a program that reads your code and dutifully carries out your commands.

Salient Features

JavaScript boasts a plethora of noteworthy attributes:

- **Versatility**: JS's utility transcends various IT sectors, finding applications in Web Development, Artificial Intelligence, Mobile App development, and more;
- **Interactivity**: A standout feature of JS is its proficiency in real-time manipulation of the Document Object Model (DOM), a critical component for building dynamic and interactive user interfaces;
- **Cross-Platform Compatibility**: JS code can function seamlessly across different operating systems and devices;
- **Easy to Learn**: JS's syntax resembles other well-known programming languages, facilitating an easier learning curve;
- **Large Community**: A thriving community of developers continually contributes to the ecosystem, creating numerous open-source libraries. This results in the continuous evolution of JavaScript;
- **Fast and Efficient**: JS shines when executing calculations and processing tasks, often outperforming other interpreted programming languages quickly and efficiently.

2. Getting Started

An introduction to JavaScript, covering the basic syntax of the language along with key terms and concepts that form the foundation for learning advanced features. The section includes hands-on tasks designed to help practice these concepts and make the content easy to remember without effort.

Summary

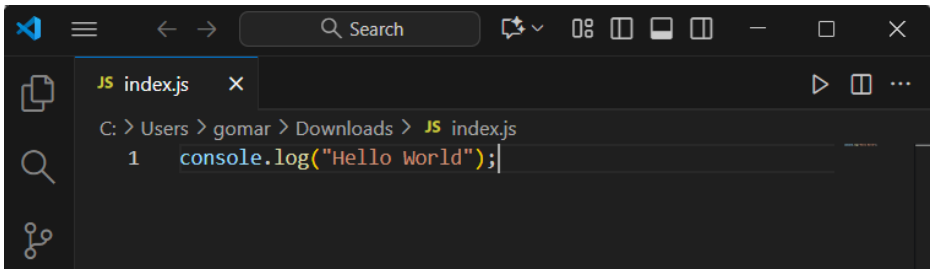
- A **statement** in programming is a single instruction that performs a specific operation or action;
- In **JavaScript**, every statement needs to be ended with a semicolon (;);
- The console.log statement can be used for displaying some output in the console;
- An example of a console.log statement is:

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

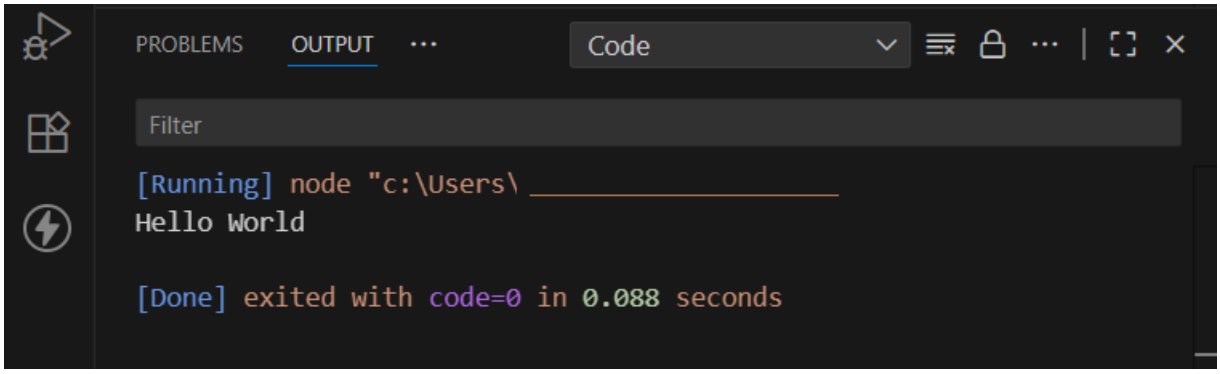
“Code Runner” library

You can use the **Code Runner** extension **Visual Studio Code** :

1. In VS Code, open the **Extensions** tab — shortcut **Ctrl + Shift + X**.
2. Search for and install the extension called **Code Runner**.
3. Open your `index.js` file or type the code in Visual Studio Code and save the file.



4. A ▶ **Run Code** button will appear in the upper right corner.
   — Click it.
5. The result will appear at the bottom in the **Output** panel:



---

3. Dealing with Numbers

Summary

- **Textual data** is always enclosed in single or double quotation marks. For example: `'Hello World'` or `"Hello World"` — both represent valid textual data;
- There are two different types of **numbers in JavaScript**:
- **Integers**;
- **Floating-point numbers**;
- **Floating-point numbers** (also known as **Floats**) are numerical values that have a decimal part. Examples: `1.234`, `24.56`, `3.1415`, etc;
- **Negative numbers** can be expressed by adding a minus (`-`) sign before the number. Examples: `-27`, `-3.14`, `-123`, etc;

- **Arithmetic operations** can be performed on numbers using the following operators:
`+`, `-`, `*`, `/`, and `**`;
- The order of arithmetic operations follows the basic rule of arithmetic (**BODMAS** or **PEMDAS** rules).

```
// Integer
console.log('Integer:', 42);

// Floating-Point Number
console.log('Floating-Point:', 3.14);

// Negative Number
console.log('Negative:', -27);

// Arithmetic Operations
console.log('Sum:', 42 + 3.14);

console.log('Product:', 42 * -27);
```

4. Outputting Multiple Values

Summary

- **Argument**: a piece of data or an expression that is passed into a statement;
- **Syntax**: the set of rules that define the correct structure of statements or code in a programming language;
- The syntax of the `console.log` statement is as follows:

```
console.log(argument1, argument2, …);
```

5. How to Use Comments in Javascript?

Summary

- **Comments**: a way to include useful notes or explanations in the code, or to temporarily ignore a piece of code;
- **Single-line comment**: used to add a comment that only takes up one line;
- The syntax of a single-line comment is:

```
// your comment text here …
```

6. Multi-Line Comments

Summary

- **Multi-line comments**: used to add comments that span over multiple lines;
- A multi-line comment starts with `/*` and ends with `*/`. Everything in between these symbols is ignored by the JavaScript engine.

```
/*
  This is a multi-line comment.
  It can span several lines.
*/
```

7. Storing Data

Definition. **Variables** are containers of data in a computer's memory. The general syntax of creating a new variable is `let variableName`.

For example, the following code declares a new variable called username:

```
let username;
```

This is known as a **Variable Declaration Statement**.

We can assign a value to a variable using the syntax `variableName = data`, where data can be some text, number, or any other valid data type.

For example:

```
let username;
username = "John Smith";
```

The statement in which we assign some value to a variable is known as an **Assignment Statement**.

We can simply use the variable name for accessing data from it:

```
let username;
username = "John Smith";
console.log(username);
```

8 Constants

Definitionю. **Constants** are similar to variables however their value, as expressed by the name, is always **constant** or **unchanged**.

We can declare a constant using the `const` keyword, and it's important to initialize it at the same point as well. Following is the syntax for constant declaration & initialization:

```
const constantName = data;
```

For example:

```
const exampleConstant = 123;
```

Changing the value of a constant results in an error:

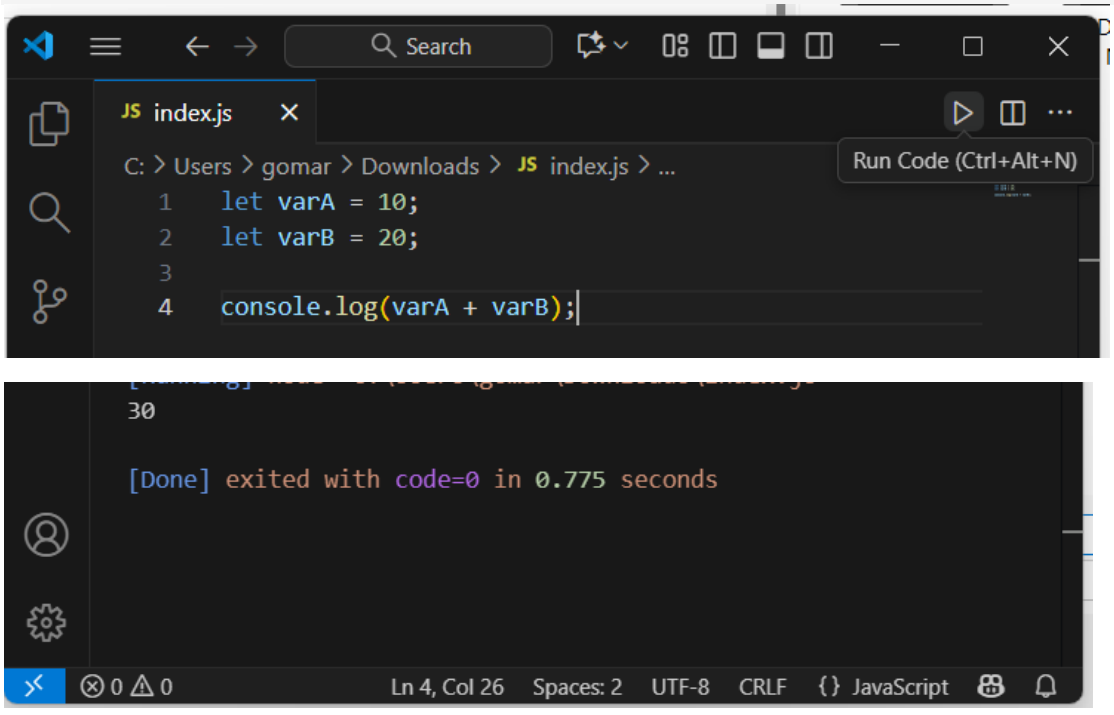
```
const exampleConstant = 123;
exampleConstant = 1234; // Error here
```

The names of the constants follow the same naming rules as the variables.

9 Performing Arithmetic On Variables

We can perform arithmetic on variables storing numbers similar to how we perform operations on raw numerical values.

For example:



10. How different Data Types Interact

Summary

- **Strings** are textual data in JavaScript. You can store them in variables like this: `let username = "Ethan";`

- You can **combine strings** using the `+` operator. This process is called **concatenation**.

Example:

```
let firstName = "John";
let lastName = "Smith";
let fullName = firstName + " " + lastName;
console.log(fullName)
```

- If you concatenate strings without adding a space manually, the result will be merged: `firstName + lastName // "JohnSmith";`
- Concatenation works only between strings. If you try to add a number, it won't behave as expected:

```
let age = 30;
let info = "Age: " + age;
console.log(info)
```

- You can **convert numbers to strings** explicitly using `String()` if needed:

  `let result = "Score: " + String(95); // "Score: 95"`.

11 Comparison Operators

Definition. **Comparison Operators**, as expressed by the name, are operators that can be used for **comparing** data.

For example, we can use the **equals to** (`==`) operator to check if two values are equal. This outputs `true` or `false` based on whether the two values are equal or not.

```
let a = 5;
let b = 10;
console.log(a == b);
```

The expression `a == b` is known as a **boolean expression** because it evaluates into a boolean value (`true` or `false`).

Note. The **equals to operator** (`==`) is not the same as the **assignment operator** (`=`).

There's a similar operator known as the **not equals to** (`!=`) operator. It simply returns `true` if the two values are not equal:

```
let a = 5;
let b = 10;
console.log(a != b);
```

There are some other operators which can be used for comparing values:

Operator	Function	Usage
===	A is strictly equal to B	A === B
!==	A is strictly not equal to B	A !== B
<	A is less than B	A < B
>	A is greater than B	A > B
<=	A is less than or equal to B	A <= B
>=	A is greater than or equal to B	A >= B

12 The `if` Statement

Definition

A **conditional statement**, also known as an **if-statement**, can be used to execute specific lines of code based on the result of a **boolean expression**.

The **syntax** of a **conditional statement** is:

```
if (boolean_expression)
{
```

```
    // some code here …
}
```

Here `boolean_expression` can be any expression which defines a **condition**. The code between the curly brackets (`{}`) is executed if the `boolean_expression` evaluates to `true`.

Note. The code enclosed between the curly brackets (`{}`) is collectively referred to as a **Code Block**.

For example:

```
let age = 18;

if(age >= 18) {
  console.log("The user is old enough to drive.");
}
```

We can have multiple **if-statements** in a program, giving us more control over which code block to execute. For example, we can further extend the previous code to output a different statement if the age is less than 18:

```
let age = 17;

if(age >= 18) {
    console.log("The user is old enough to drive.");
}

if(age < 18) {
    console.log("The user is not old enough to drive");
}
```