JavaScript Basics

What is Javascript?

JavaScript is a high-level, interpreted programming language that is commonly used for web development. It was created in 1995 by Brendan Eich while he was working at Netscape Communications Corporation. JavaScript is often abbreviated as JS, and it is one of the most popular programming languages in the world.

The purpose of JavaScript is to add interactivity and dynamic functionality to web pages. With JavaScript, you can add animations, respond to user actions, validate user input, and manipulate the HTML and CSS of a web page. It is also commonly used on the development side to build scalable web applications.

JavaScript Features : JavaScript has several key features that make it a popular choice for web development:

Interactivity

JavaScript allows you to add interactive elements to web pages, such as dropdown menus, pop-ups, and animations etc.

Lightweight:

JavaScript is a lightweight programming language that can be easily embedded in web pages.

Flexibility:

JavaScript can be used on the client-side (in web browsers) and the server-side (with Node.js).

Rich library support:

There are numerous JavaScript libraries and frameworks that make it easier to develop web applications.

High performance:

Modern JavaScript engines are highly optimized, making JavaScript code run very quickly in web browsers.

What is Node?

Node.js is an open-source, cross-platform JavaScript runtime environment that allows developers to run JavaScript code outside of a web browser. It is built on the V8 JavaScript engine (which is used by Google Chrome) and provides a way to write server-side applications using JavaScript.

Installation :

To install Node.js, you can download the installer from the official Node.js website (https://nodejs.org/en/). Once the installer is downloaded, run it and follow the on-screen instructions to complete the installation.

Executing :

To execute a JavaScript file with Node.js, you can use the command-line interface (CLI). Open a terminal or command prompt and navigate to the directory that contains the JavaScript file. Then, run the following command:

```
node <name of your js file>
```

This will execute the JavaScript file and output any results to the console.

console.log()

console.log() is a JavaScript function that is used to output data to the console (in web browsers or with Node.js). It takes one or more arguments and prints them to the console. For example:

```
console.log("Hello, world!");
```

This would output the string "Hello, world!" to the console.

JS variables

A variable in JavaScript is a container for storing data values. You can declare a variable using the var, let, or const keywords. For example:

```
var name = "John";
let age = 30;
const PI = 3.14;
```

In this example, name is a variable that stores a string, age is a variable that stores a number, and PI is a variable that stores a constant value.

Data Types

In the previous section, we mentioned a little bit about data types. Data or values have data types. Data types describe the characteristics of data. Data types can be divided into two:

- Primitive data types
- Non-primitive data types(Object References)

Primitive Data Types

Primitive data types in JavaScript include:

- Numbers - Integers, floats
- Strings - Any data under single quote, double quote or backtick quote
- Booleans - true or false value
- Null - empty value or no value
- Undefined - a declared variable without a value
- Symbol - Symbol is a built-in object whose constructor returns a symbol primitive — also called a Symbol value or just a Symbol — that's guaranteed to be unique.
- BigInt - BigInt is a numeric data type that allows you to represent integers with arbitrary precision.

Primitive data types are immutable(non-modifiable) data types. Once a primitive data type is created we cannot modify it.

Example:

```
let word = "JavaScript";
```

If we try to modify the string stored in variable word, JavaScript should raise an error. Any data type under a single quote, double quote, or backtick quote is a string data type.

```
word[0] = "Y";
```

This expression does not change the string stored in the variable word. So, we can say that strings are not modifiable or in other words immutable.

Primitive data types are compared by its values. Let us compare different data values. See the example below:

```
let numOne = 3;
let numTwo = 3;

console.log(numOne == numTwo); // true

let js = "JavaScript";
let py = "Python";

console.log(js == py); //false

let lightOn = true;
let lightOff = false;

console.log(lightOn == lightOff); // false
```

Booleans

A boolean data type represents one of the two values:true or false. Boolean value is either true or false. The use of these data types will be clear when you start the comparison operator. Any comparisons return a boolean value which is either true or false.

Example: Boolean Values

```
let isLightOn = true;
let isRaining = false;
let isHungry = false;
let isMarried = true;
let truValue = 4 > 3; // true
let falseValue = 4 < 3; // false
```

We agreed that boolean values are either true or false.

Undefined

If we declare a variable and if we do not assign a value, the value will be undefined. In addition to this, if a function is not returning the value, it will be undefined.

```
let firstName;
console.log(firstName); //not defined, because it is
not assigned to a value yet
```

Null

```
let empty = null;
console.log(empty); // -> null , means no value
```

Numbers

```
var a = 1;
console.log(a); // -> var holding a number data type
```

Strings

```
var a = "Geekster";
console.log(a); // -> var holding a string data type
```

Non-Primitive Data Types

Non-primitive data types are modifiable or mutable. We can modify the value of non-primitive data types after it gets created. Let us see by creating an array. An array is a list of data values in a square bracket. Arrays can contain the same or different data types. Array values are referenced by their index. In JavaScript array index starts at zero. I.e., the first element of an array is found at index zero, the second element at index one, and the third element at index two, etc.

```
let nums = [1, 2, 3];
nums[0] = 10;

console.log(nums); // [10, 2, 3]
```

As you can see, an array, which is a non-primitive data type is mutable. Non-primitive data types cannot be compared by value. Even if two non-primitive data types have the same properties and values, they are not strictly equal.

```
let nums = [1, 2, 3];
let numbers = [1, 2, 3];

console.log(nums == numbers); // false

let userOne = {
  name: "Asabeneh",
  role: "teaching",
  country: "Finland",
};

let userTwo = {
  name: "Asabeneh",
  role: "teaching",
  country: "Finland",
};

console.log(userOne == userTwo); // false
```

Rule of thumb, we do not compare non-primitive data types. Do not compare arrays, functions, or objects. Non-primitive values are referred to as reference types, because they are being compared by reference instead of value. Two objects are only strictly equal if they refer to the same underlying object.

```
let nums = [1, 2, 3];
let numbers = nums;

console.log(nums == numbers); // true

let userOne = {
  name: "Asabeneh",
  role: "teaching",
  country: "Finland",
};

let userTwo = userOne;

console.log(userOne == userTwo); // true
```

Operators

- Assignment operators

An equal sign in JavaScript is an assignment operator. It uses to assign a variable.

```
let firstName = "Asabeneh";
let country = "Finland";
```

- Arithmetic Operators

Arithmetic operators are mathematical operators.

1. Addition(+): a + b
2. Subtraction(-): a - b
3. Multiplication(*): a * b
4. Division(/): a / b
5. Modulus(%): a % b
6. Exponential(**): a ** b

```
let numOne = 4;
let numTwo = 3;
let sum = numOne + numTwo;
let diff = numOne - numTwo;
let mult = numOne * numTwo;
let div = numOne / numTwo;
let remainder = numOne % numTwo;
let powerOf = numOne ** numTwo;

console.log(sum, diff, mult, div, remainder, powerOf);
// 7,1,12,1.33,1, 64
```

```
const PI = 3.14;
let radius = 100; // length in meter

//Let us calculate area of a circle
const areaOfCircle = PI * radius * radius;
console.log(areaOfCircle); //  314 m

const gravity = 9.81; // in m/s2
let mass = 72; // in Kilogram

// Let us calculate weight of an object
const weight = mass * gravity;
console.log(weight); // 706.32 N(Newton)

const boilingPoint = 100; // temperature in oC,
boiling point of water
const bodyTemp = 37; // body temperature in oC

// Concatenating string with numbers using string
interpolation
/*
 The boiling point of water is 100 oC.
 Human body temperature is 37 oC.
 The gravity of earth is 9.81 m/s2.
 */
console.log(
  `The boiling point of water is ${boilingPoint}
oC.\nHuman body temperature is ${bodyTemp} oC.\nThe
gravity of earth is ${gravity} m / s2.`
);
```

- Comparison Operators

In programming we compare values, we use comparison operators to compare two values. We check if a value is greater or less or equal to other value.

Example: Comparison Operators

```
console.log(3 > 2); // true, because 3 is greater than
2
console.log(3 >= 2); // true, because 3 is greater
than 2
console.log(3 < 2); // false,  because 3 is greater
than 2
console.log(2 < 3); // true, because 2 is less than 3
console.log(2 <= 3); // true, because 2 is less than 3
console.log(3 == 2); // false, because 3 is not equal
to 2
console.log(3 != 2); // true, because 3 is not equal
to 2
console.log(3 == "3"); // true, compare only value
console.log(3 === "3"); // false, compare both value
and data type
console.log(3 !== "3"); // true, compare both value
and data type
console.log(3 != 3); // false, compare only value
console.log(3 !== 3); // false, compare both value and
data type
```

```
console.log(0 == false); // true, equivalent
console.log(0 === false); // false, not exactly the
same
console.log(0 == ""); // true, equivalent
console.log(0 == " "); // true, equivalent
console.log(0 === ""); // false, not exactly the same
console.log(1 == true); // true, equivalent
console.log(1 === true); // false, not exactly the
same
console.log(undefined == null); // true
console.log(undefined === null); // false
console.log(NaN == NaN); // false, not equal
console.log(NaN === NaN); // false
console.log(typeof NaN); // number
```

```
console.log("mango".length == "avocado".length); //
false
console.log("mango".length != "avocado".length); //
true
console.log("mango".length < "avocado".length); //
true
console.log("milk".length == "meat".length); // true
console.log("milk".length != "meat".length); // false
console.log("tomato".length == "potato".length); //
true
console.log("python".length > "dragon".length); //
false
```

Try to understand the above comparisons with some logic. Remembering without any logic might be difficult. JavaScript is somehow a wiered kind of programming language. JavaScript code run and give you a result but unless you are good at it may not be the desired result.
As rule of thumb, if a value is not true with == it will not be equal with ===. Using === is safer than using ==.

```
console.log(5 == "5"); // true - JavaScript coerces
the string "5" to a number and compares 5 == 5
console.log(1 == true); // true - JavaScript coerces
the boolean value true to a number (1) and compares 1
== 1
console.log(null == undefined); // true - "null" and
"undefined" are considered equal by the double equals
operator
```

```
console.log(5 === "5"); // false - The operands have
different types (number and string)
console.log(1 === true); // false - The operands have
different types (number and boolean)
console.log(null === undefined); // false - The
operands have different types (null and undefined)
```

In general, it is recommended to use the "===" operator for equality comparisons unless you explicitly need the type coercion behavior provided by the "==" operator. Using strict equality helps avoid unexpected results and makes your code more reliable and maintainable.

- Logical Operators

The following symbols are the common logical operators: &&(ampersand) , ||(pipe) and ! (negation). The && operator gets true only if the two operands are true. The || operator gets true either of the operand is true. The ! operator negates true to false and false to true.

Example:

```
// && ampersand operator example

const check = 4 > 3 && 10 > 5; // true && true -> true
const check = 4 > 3 && 10 < 5; // true && false ->
false
const check = 4 < 3 && 10 < 5; // false && false ->
false

// || pipe or operator, example

const check = 4 > 3 || 10 > 5; // true  || true ->
true
const check = 4 > 3 || 10 < 5; // true  || false ->
true
const check = 4 < 3 || 10 < 5; // false || false ->
false

//! Negation examples

let check = 4 > 3; // true
let check = !(4 > 3); //  false
let isLightOn = true;
let isLightOff = !isLightOn; // false
let isMarried = !false; // true
```

- Increment Operator

In JavaScript we use the increment operator to increase a value stored in a variable. The increment could be pre or post increment. Let us see each of them:

1) Pre-increment

```
let count = 0;
console.log(++count); // 1
console.log(count); // 1
```

2) Post-increment

```
let count = 0;
console.log(count++); // 0
console.log(count); // 1
```

- Decrement Operator

In JavaScript we use the decrement operator to decrease a value stored in a variable. The decrement could be pre or post decrement. Let us see each of them:

1) Pre-decrement

```
let count = 0;
console.log(--count); // -1
console.log(count); // -1
```

2) Post-decrement

```
let count = 0;
console.log(count--); // 0
console.log(count); // -1
```

- Ternary Operators

Ternary operator allows to write a condition. Another way to write conditionals is using ternary operators. Look at the following examples:

```
let isRaining = true;
isRaining
  ? console.log("You need a rain coat.")
  : console.log("No need for a rain coat.");
isRaining = false;

isRaining
  ? console.log("You need a rain coat.")
  : console.log("No need for a rain coat.");
```

Output:

```
You need a rain coat.
No need for a rain coat.
```

Example:

```
let number = 5;
number > 0
  ? console.log(`${number} is a positive number`)
  : console.log(`${number} is a negative number`);
number = -5;

number > 0
  ? console.log(`${number} is a positive number`)
  : console.log(`${number} is a negative number`);
```

Output:

```
5 is a positive number
-5 is a negative number
```

Comment:

In JavaScript, a single line comment can be added by using two forward slashes // .

Type Conversion in JavaScript(coercion):

Type conversion, also known as type coercion, is the process of converting a value from one data type to another. JavaScript performs implicit and explicit type conversions.

1) Implicit Type Conversion:
Implicit type conversion occurs automatically when JavaScript converts a value from one type to another without any explicit instructions from the programmer. For example, when you use the '+' operator with a string and a number, JavaScript implicitly converts the number to a string and concatenates them.

Example:

```
var num = 10;
var str = "5";
var result = num + str; // Implicitly converts num to
a string and concatenates
console.log(result); // Output: "105"
```

2) Explicit Type Conversion:
Explicit type conversion involves explicitly converting a value from one type to another using built-in functions or operators. JavaScript provides several functions and operators for explicit type conversion.

Common functions for explicit type conversion:

parseInt(): Converts a string to an integer.
parseFloat(): Converts a string to a floating-point number.
String(): Converts a value to a string.
Number(): Converts a value to a number.
Boolean(): Converts a value to a boolean.

```
var str = "10";
var num = parseInt(str); // Explicitly converts str to
an integer
console.log(num); // Output: 10

var bool = Boolean(0); // Explicitly converts 0 to a
boolean
console.log(bool); // Output: false
```

Taking input using prompt

In JavaScript, you can use the prompt() function to take input from the user via a dialog box. The prompt() function displays a dialog box with an optional message and an input field where the user can enter data.

Here's an example of using the prompt() function to take input from the user:

```
var name = prompt("Please enter your name:");
console.log("Hello, " + name + "!"); // Prints a
greeting with the entered name
```

In the above example, the prompt() function is called with the message "Please enter your name:". The user can enter their name in the input field provided by the dialog box. The value entered by the user is then assigned to the variable name. Finally, the entered name is displayed in the console with a greeting.

Note that the prompt() function returns the value entered by the user as a string. If you need to convert the input to a different data type, such as a number, you can use explicit type conversion functions like parseInt() or parseFloat().