DSA in JavaScript

DSA stands for Data Structures and Algorithms.

Data structure: - It is a way to organize and store data in a computer so that it can be accessed and modified efficiently.

Example: - a family tree.

there are two different kinds of data structures.

Primitive Data Structures are basic data structures provided by programming languages to represent single values, such as integers, floating-point numbers, characters, and Boolean.

Abstract Data Structures are higher-level data structures that are built using primitive data types and provide more complex and specialized operations. Some common examples of abstract data structures include arrays, linked lists, stacks, queues, trees, and graphs.

Algorithm: - A step-by-step procedure to solve a problem.
Example: - cooking dinner.

DSA is about finding efficient ways to store and retrieve data, to perform operations on data, and to solve specific problems.

We need a programming language for this. Here we are using JavaScript.

JavaScript

JavaScript is a scripting language you can use to make web pages interactive. JavaScript is already running in your browser on your computer. Let's understand the basics.

Comments: -

Comments are lines of code that JavaScript will intentionally ignore. It is a way to tell yourself and to other people who will later need to figure out what that code does.

There are two ways to write comments in JavaScript:

1.// This is an in-line or single line comment.

2./* this is a multi

Line comment */

Variables: -

Variables are containers which is used to store data. Variable names can be made up of numbers, letters, and $ or _, but may not contain spaces or start with a number.

Semicolon is optional but recommended.

It is done in four ways.

1.Automatically

2.Using var

3.Using let

4.Using const

Examples: -

1.Automatically

In this first example, x, y, and z are undeclared variables. They are automatically declared when first used:

```
x = 5;
y = 6;
z = x + y;
```

However, A good practice is to always declare variables before use.

2. var

```
var x = 5;
var y = 6;
var z = x + y;
```

Variables declared with the var always have global scope.

```
{
  var x = 2;
}
// x can be used here
```

Variables defined with var can be redeclared.

3.let

```
let x = 5;
let y = 6;
let z = x + y;
```

Variables declared with let have block scope.

```
{
  let x = 2;
}
// x cannot be used here
```

Variables defined with let cannot be redeclared.

4.const

```
const x = 5;
const y = 6;
const z = x + y;
```

```
const PI = 3.141592653589793;
PI = 3.14;      // This will give an error
PI = PI + 10;   // This will also give an error
```

Note: -

The var keyword was used in all JavaScript code from 1995 to 2015.

The let and const keywords were added to JavaScript in 2015 (ES6).

Always declare variables.

Always use const if the value should not be changed.

Memory in JavaScript: -

Stack: Stores function calls, local variables, and arguments. Works in a LIFO (Last In, First Out) fashion.

Heap: Stores objects, arrays, and functions. It handles dynamic memory allocation and is managed by the garbage collector.

Code Segment: Stores the actual code (global variables), which is the executable part of the program.

Data Types: -

JavaScript has 8 Datatypes

1.Numbers: (8 bytes)

let length = 16;

let weight = 7.5;

let y = 123e5; // 12300000

JavaScript numbers are always one type: double (64-bit floating point).

2.Strings: (2 bytes per character)

let color = "Yellow";

let lastName = "Johnson";

3.Booleans: (4 bytes)

let x = true;

let y = false;

4.Object: (varies)

const person = {firstName:"John", lastName:"Doe"};

The object data type can contain both built-in objects, and user defined objects:

Built-in object types can be:

objects, arrays, dates, maps, sets, promises, and more.

5.Undefined: (4 bytes)

let car; // Value is undefined, type is undefined

car = undefined; // Value is undefined, type is undefined

6.BigInt: (varies)

JavaScript BigInt is a new datatype that can be used to store integer values that are too big to be represented by a normal JavaScript Number.

let x = BigInt("123456789012345678901234567890");

7.Null: (4 bytes)

let y = null;

8.Symbol:

A Symbol is a **unique and immutable primitive value** that can be used as an identifier for object properties.

let sym1 = Symbol();

if you want to create **unique identifiers** that can't be duplicated.

Note: -

JavaScript uses the + operator for both addition and concatenation.

If you add two numbers, the result will be a number.

If you add two strings, the result will be a string concatenation.

If you add a number and a string, the result will be a string concatenation:

Operators: -

1. Arithmetic Operators

- **+ (Addition)**
  Example: 5 + 3
  Adds two operands. Result: 8
- **- (Subtraction)**
  Example: 5 - 3
  Subtracts the second operand from the first. Result: 2
- *** (Multiplication)**
  Example: 5 * 3
  Multiplies two operands. Result: 15
- **/ (Division)**
  Example: 6 / 3
  Divides the first operand by the second. Result: 2

- **% (Modulus)**

  Example: 5 % 3

  Returns the remainder of the division. Result: 2

2.Unary Operators

- **++ (Increment)**

  Example: let a = 5; a++

  Increments the operand by 1. Result: 6

- **-- (Decrement)**

  Example: let a = 5; a--

  Decrements the operand by 1. Result: 4

3. Assignment Operators

- **= (Simple assignment)**

  Example: let a = 5;

  Assigns a value to a variable.

- **+= (Add and assign)**

  Example: a += 3;

  Adds a value and assigns the result to the variable.

- **-= (Subtract and assign)**

  Example: a -= 3;

  Subtracts a value and assigns the result to the variable.

- ***= (Multiply and assign)**

  Example: a *= 3;

  Multiplies a value and assigns the result to the variable.

- **/= (Divide and assign)**

  Example: a /= 3;

  Divides a value and assigns the result to the variable.

- **%= (Modulus and assign)**

  Example: a %= 3;

  Modulus a value and assigns the result to the variable.

4. Comparison Operators

- **== (Equal to)**

  Example: 5 == 5

  Compares if two values are equal. Result: true

- **=== (Strict equal to)**

  Example: 5 === '5'

  Compares if two values are equal and of the same type. Result: false

- **!= (Not equal to)**

  Example: 5 != 3

  Compares if two values are not equal. Result: true

- **!== (Strict not equal to)**

  Example: 5 !== '5'

  Compares if two values are not equal or not of the same type. Result: true

- **> (Greater than)**

  Example: 5 > 3

  Compares if the first value is greater than the second. Result: true

- **< (Less than)**

  Example: 5 < 3

  Compares if the first value is less than the second. Result: false

- **>= (Greater than or equal to)**

  Example: 5 >= 5

  Compares if the first value is greater than or equal to the second. Result: true

- **<= (Less than or equal to)**

  Example: 5 <= 3

  Compares if the first value is less than or equal to the second. Result: false

5. String Operators

- **+ (Concatenation)**

  Example: 'Hello' + ' ' + 'World'

  Concatenates two or more strings. Result: 'Hello World'

- **+= (Concatenation assignment)**

  Example: let str = 'Hello'; str += ' World';

  Concatenates a string and assigns it to a variable. Result: 'Hello World'

6. Logical Operators

- **&& (Logical AND)**

  Example: true && false

  Returns true if both operands are true. Result: false

- **|| (Logical OR)**

  Example: true || false

  Returns true if at least one operand is true. Result: true

- **! (Logical NOT)**

  Example: !true

  Inverts the boolean value. Result: false

7. Bitwise Operators

- **& (AND)**

  Example: 5 & 3 (0101 & 0011)

  Performs a bitwise AND. Result: 1

- **| (OR)**

  Example: 5 | 3 (0101 | 0011)

  Performs a bitwise OR. Result: 7

- **^ (XOR)**

  Example: 5 ^ 3 (0101 ^ 0011)

  Performs a bitwise XOR. Result: 6

- **~ (NOT)**

  Example: ~5 (~0101)

  Performs a bitwise NOT. Result: -6

- **<< (Left shift)**

  Example: 5 << 1 (0101 << 0001)

  Shifts bits to the left. Result: 10

- **>> (Right shift)**

  Example: 5 >> 1 (0101 >> 0001)

  Shifts bits to the right. Result: 2

- **>>> (Unsigned right shift)**

  Example: -5 >>> 1 (0101 >>> 0001)

  Shifts bits to the right without sign. Result: 2147483645

8. Ternary Operators

- **condition ? expr1 : expr2**

  Example: let result = (5 > 3) ? 'Yes' : 'No';

  A shorthand for if-else. If condition is true, expr1 is executed; otherwise, expr2.

  Result: 'Yes'

9. Type Operators

- **typeof**

  Example: typeof 5

  Returns the type of the operand. Result: 'number'

- **instanceof**

  Example: 'Hello' instanceof String

  Tests if an object is an instance of a class. Result: false

  new String('Hello') instanceof String // true

Conditional Statements

1.if Statement

The if statement executes a block of code if the specified condition evaluates to true.

```
let age = 18;

if (age >= 18) {

   console.log ("You are an adult.");

}
```

Explanation: If age is 18 or more, it prints "You are an adult."

2.if-else Statement

The if-else statement executes one block of code if the condition is true, and another block if it's false.

```
let age = 16;

if (age >= 18) {

   console.log ("You are an adult.");

} else {

   console.log ("You are a minor.");

}
```

Explanation: If age is 18 or more, it prints "You are an adult"; otherwise, it prints "You are a minor."

3.else if Statement

The else if statement allows you to test multiple conditions. If the first if condition is false, it checks the else if conditions sequentially.

```
let age = 25;

if (age < 18) {

   console.log("You are a minor.");

} else if (age >= 18 && age <= 65) {

   console.log("You are an adult.");

} else {

   console.log("You are a senior.");

}
```

Explanation: It checks if the age is less than 18 (prints "minor"), between 18 and 65 (prints "adult"), or over 65 (prints "senior").

4.switch Statement

The switch statement is used to evaluate an expression against multiple cases.

```
let day = 2;

switch (day) {
   case 1:
      console.log("Monday");
      break;
   case 2:
      console.log("Tuesday");
      break;
   case 3:
      console.log("Wednesday");
      break;
   default:
      console.log("Invalid day");
}
```

Explanation: It evaluates the value of day. If day is 2, it prints "Tuesday". The break stops further checking. Default is executed when none of the other matches.

Loops

1.for Loop

The for loop runs a block of code a specific number of times, based on a condition.

```
for (let i = 0; i < 5; i++) {
   console.log(i);
}
```

Explanation: This loop prints numbers 0 through 4. The loop starts with i = 0, checks if i < 5, and increments i after each iteration.

2.while Loop

The while loop repeats a block of code as long as the specified condition is true.

```
let i = 0;

while (i < 5) {

   console.log(i);

   i++;

}
```

Explanation: This loop prints numbers 0 through 4. It continues looping as long as i < 5.

3.do-while Loop

The do-while loop is similar to the while loop, but it guarantees that the code runs at least once because the condition is checked **after** the block of code is executed.

```
let i = 0;

do {

   console.log(i);

   i++;

} while (i < 5);
```

Explanation: It works like the while loop, but the code inside the loop is executed first before the condition is checked.

4.for-in Loop

The for-in loop is used to iterate over the properties of an object.

```
let person = { name: "Alice", age: 25, city: "New York" };

for (let key in person) {

   console.log(key + ": " + person[key]);

}
```

Explanation: It loops through the properties (key) of the person object and prints the property names along with their values.

5.for-of Loop

The for-of loop is used to iterate over iterable objects like arrays or strings.

```
let colours = ["red", "green", "blue"];

for (let colour of colours) {

   console.log(colour);

}
```

Explanation: It loops over each element in the colours array and prints them one by one.