JAVASCRIPT - Day 1

Table of Content

Comments:

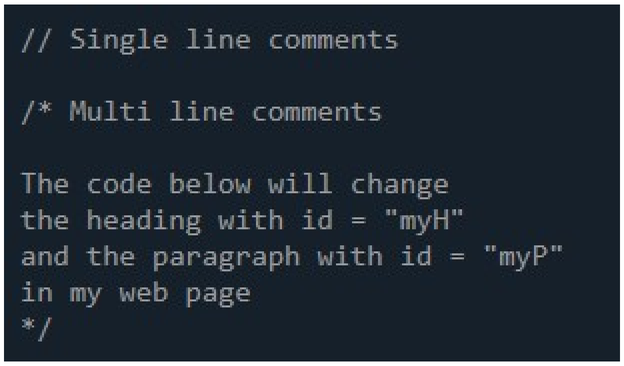
1. Desc:

JavaScript comments can be used to explain JavaScript code, and to make it more readable.

2. Usage:

JavaScript comments can also be used to prevent execution, when testing alternative code.

3. Code Snippets:



```
// Single line comments

/* Multi line comments

The code below will change
the heading with id = "myH"
and the paragraph with id = "myP"
in my web page
*/
```

4. Advantage:

Comments makes Javascript code easier to understand and maintain by explaining what the code does.

5. Disadvantage:

Oudated comments can clutter the code and cause confusion.

Variables:

1. Desc:

Variables are Containers for Storing Data.

2. Usage:

Variables are used to store data values that can be reused and manipulated throughout the program.

3. Types of Variables:

JavaScript Variables can be declared in 4 ways:

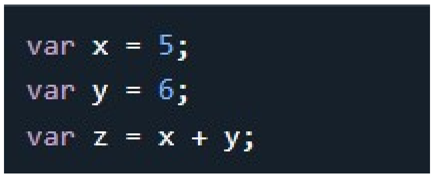
- Automatically
- Using var
- Using let
- Using const

i)Using var:

a. Desc:

Used to declare variables globally or within a function, and is hoisted to the top of its scope.

b. Code Snippets:



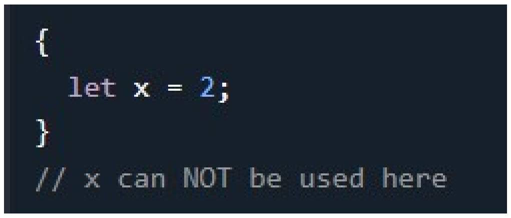
```
var x = 5;
var y = 6;
var z = x + y;
```

ii)Using let:

a. Desc:

Used to declare block-scoped variables that can be updated but not re-declared in the same block.

b. Code Snippets:



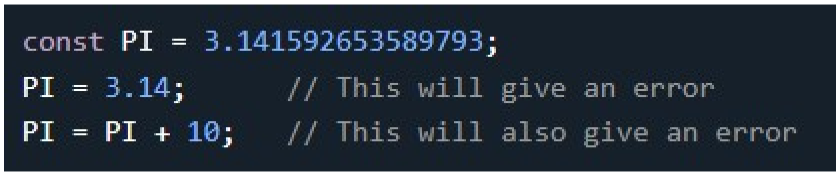
```
{
  let x = 2;
}
// x can NOT be used here
```

iii)Using const:

a. Desc:

Used to declare block-scoped variables with constant values that cannot be changed after assignment.

b. Code Snippets:



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

4. Advantage:

Variables make the code dynamic and reusable by storing and manipulating data efficiently.

5. Disadvantage:

Improper use of variables can lead to bugs, memory issues or unexpected behavior.

Identifiers

An identifier is a name used to identify variables, functions, objects, or other entities.

- **Rules for naming identifiers:**
    - Must start with a letter (a-z, A-Z), underscore (_), or dollar sign ($).
    - Ientifiers can include numbers (0-9).
    - Cannot use JavaScript reserved keywords (e.g., var, let, function).
- **Examples:**

    let myVariable; // valid

```
let $amount; // valid

let _name;  // valid

let 1name;  // invalid
```

Operators:

1. Desc:

Operators in javaScript are predefined symbols or keywords that perform specific operations on operands (values or variables) to produce a result.

2. Usage:

Operators in javascript are used to perform tasks like calculations, comparisons and assigning values.

3. Types of JavaScript Operators:

- Arithmetic Operators
- Assignment Operators
- Comparison Operators
- Logical Operators
- Bitwise Operators
- Ternary Operators
- Type Operators

i) Arithmetic Operators:

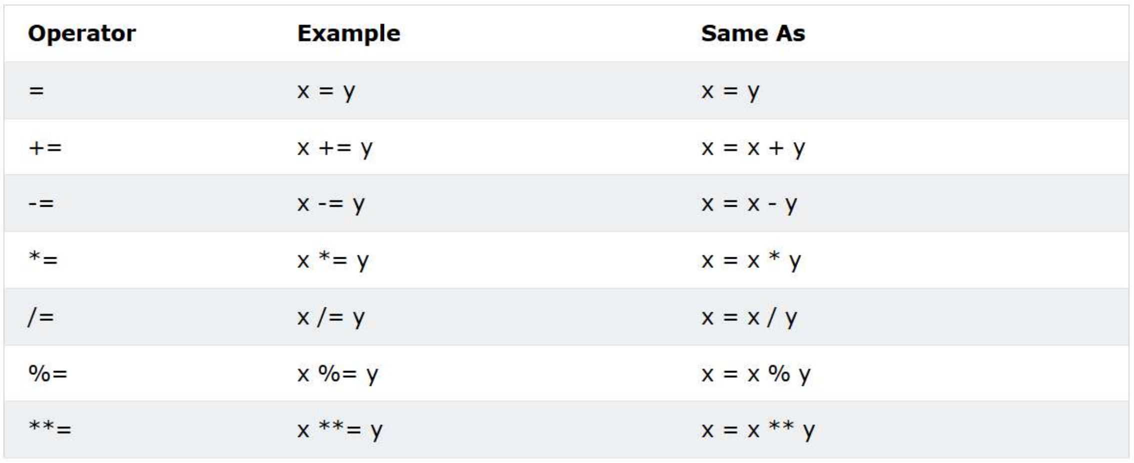
Desc: Arithmetic operators are used to perform basic mathematical operations like addition, subtraction, multiplication, division, modulus.

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
**	Exponentiation (ES2016)

/	Division
%	Modulus (Division Remainder)
++	Increment
--	Decrement

ii) Assignment Operators:

Desc: Assignment operators are used to assign values to variables, such as = for simple assignment and +=, -=, *=, /= for combined operations and assignment.



Operator	Example	Same As
=	x = y	x = y
+=	x += y	x = x + y
-=	x -= y	x = x - y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y
**=	x **= y	x = x ** y

iii) Comparison Operators:

Desc: Comparison operators are used to compare two values and return a boolean result.

Operator	Description
==	equal to
===	equal value and equal type
!=	not equal
!==	not equal value or not equal type
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to
?	ternary operator

iv) Logical Operators:

Desc: Logical operators are used to combine or invert boolean values.

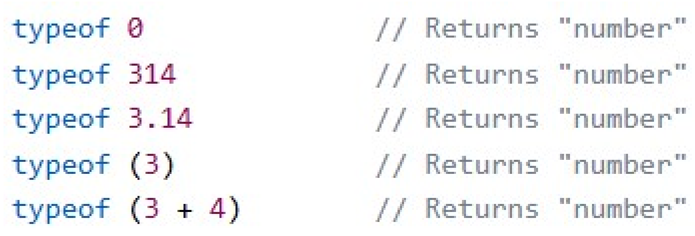
Operator	Description
&&	logical and
||	logical or
!	logical not

v) Type Operators:

Desc: Type operators are used to determine or convert data types, such as typeof and instanceof.

Operator	Description
typeof	Returns the type of a variable
instanceof	Returns true if an object is an instance of an object type

Example:



```
typeof 0                // Returns "number"
typeof 314              // Returns "number"
typeof 3.14             // Returns "number"
typeof (3)              // Returns "number"
typeof (3 + 4)          // Returns "number"
```

vi) Bitwise Operators:

Desc: Bitwise operators perform operations on binary representations of numbers.

Operator	Description	Example	Same as	Result	Decimal
&	AND	5 & 1	0101 & 0001	0001	1
|	OR	5 | 1	0101 | 0001	0101	5
~	NOT	~ 5	~0101	1010	10
^	XOR	5 ^ 1	0101 ^ 0001	0100	4
<<	left shift	5 << 1	0101 << 1	1010	10
>>	right shift	5 >> 1	0101 >> 1	0010	2
>>>	unsigned right shift	5 >>> 1	0101 >>> 1	0010	2

Data types:

1. Concept:

Without data types, a computer cannot safely solve this:

```
let x = 16 + "Volvo";
```

Does it make any sense to add "Volvo" to sixteen? Will it produce an error or will it produce a result? JavaScript will treat the example above as:

```
let x = "16" + "Volvo";
```

Note: When adding a number and a string, JavaScript will treat the number as a string.

2 Desc:

Data types define the type of value a variable can hold, such as numbers, strings, booleans, objects, and more.

3. JavaScript has 8 Datatypes

- String
- Number
- Bigint
- Boolean

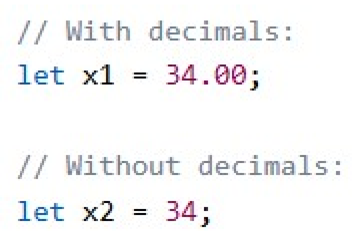
- Undefined
- Null
- Symbol
- Object

i) Number:

Desc:

All JavaScript numbers are stored as decimal numbers (floating point). Nmbers can be written with, or without decimals.

Example:



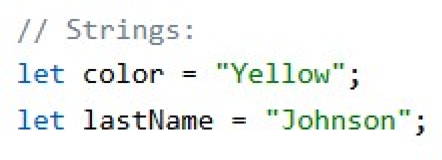
```
// With decimals:
let x1 = 34.00;

// Without decimals:
let x2 = 34;
```

ii) String:

Desc: Represents a sequence of characters, enclosed in quotes, e.g., "Hello" or 'Hi'.

Example:



```
// Strings:
let color = "Yellow";
let lastName = "Johnson";
```

iii) Bigint:

Desc:

JavaScript BigInt is a new datatype (ES2020) that can be used to store integer values that are too big to be represented by a normal JavaScript Number.

Example:

```
let x = BigInt("123456789012345678901234567890");
```

iv) Boolean:

Desc: Represents a logical value: either true or false.

Example:

```
// Booleans
let x = true;
let y = false;
```

v) Undefined:

Desc: A variable that has been declared but not assigned any value.

Example:

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

vi) Null:

Desc: Represents an intentional absence of any object value.

vii) Symbol:

Desc: Represents a unique and immutable identifier, often used as object keys.

viii) Object:

Desc:

JavaScript objects are written with curly braces {}.Object properties are written as name:value pairs, separated by commas.

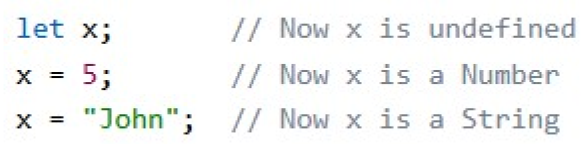
Note: The object data type can contain both built-in objects, and user defined objects.

Example:

```
const person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};
```

JavaScript Types are Dynamic:

JavaScript has dynamic types. This means that the same variable can be used to hold different data types.

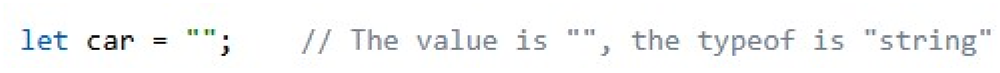


```
let x;         // Now x is undefined
x = 5;         // Now x is a Number
x = "John";    // Now x is a String
```

Empty Values:

An empty value has nothing to do with undefined. An empty string has both a legal value and a type.

Example:



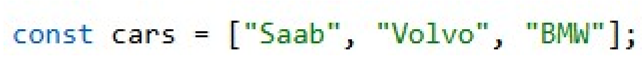
```
let car = "";    // The value is "", the typeof is "string"
```

Arrays:

1. Desc:

JavaScript array is an object that represents an ordered collections of elements where each element can be accessed using its index, starting from 0. They can store multiple data types like numbers, strings, objects, or even other arrays.

Example:



```
const cars = ["Saab", "Volvo", "BMW"];
```

2. Creating an Array:

i) Using an array literal:

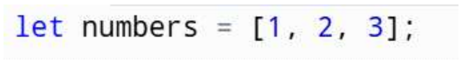
Desc:

Array literals are a way to create arrays directly by writing values inside square brackets [], making the syntax simple and readable.

Syntax:

const array_name = [item1, item2, ...];

Example:



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

ii) Using an Array object:

Desc:

An array object is created using the new Array() constructor, which builds an array as an instance of the Array class.

Example:

```
const cars = new Array("Saab", "Volvo", "BMW");
```

3. Accessing Array Elements:

Desc:

You access an array element by referring to the index number:

```
const cars = ["Saab", "Volvo", "BMW"];
let car = cars[0];  //"Saab"
```

4. Changing an Array Element:

Array elements are accessed using their index number:

Array indexes start with 0:

[0] is the first array element

[1] is the second

[2] is the third ...

Example:

```
const cars = ["Saab", "Volvo", "BMW"];
cars[0] = "Opel"; // ["Opel", "Volvo", "BMW"];
```

5. Accessing the First Array Element:

Example:

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
let fruit = fruits[0]; //Banana
```

6. Accessing the Last Array Element:

Example:

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
let fruit = fruits[fruits.length - 1]; //Mango
```

7. Array Methods:

i) Array length:

The length property returns the length (size) of an array.

Example:

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
let size = fruits.length; //4
```

ii) Array toString:

The JavaScript method toString() converts an array to a string of (comma separated) array values.

Example:

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
let result = fruits.toString();
console.log(result);
// Banana,Orange,Apple,Mango
```

iii) Array at():

The at() method returns an indexed element from an array.

Example:

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
let fruit = fruits.at(2); //Apple
```

iv) Array join():

The join() method joins all array elements into a string.

Example:

```
let fruits = ["apple", "banana", "mango"];
let result = fruits.join(" * ");
console.log(result); // apple * banana * mango
```

v) Array pop():

The pop() method removes the last element from an array. The pop() method returns the value that was "popped out".

Example:

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
let fruit = fruits.pop(); //Mango
fruits.pop(); // Banana,Orange,Apple
```

vi) Array push():

The push() method adds a new element to an array (at the end). The push() method returns the new array length.

Example:

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.push("Kiwi"); // Banana,Orange,Apple,Mango,Kiwi
let length = fruits.push("Kiwi"); //5
```

vii) Array shift():

The shift() method removes the first array element and "shifts" all other elements to a lower index. The shift() method returns the value that was "shifted out".

Example:

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.shift(); // Orange,Apple,Mango
let fruit = fruits.shift(); // Banana
```

viii) Array unshift():

The unshift() method adds a new element to an array (at the beginning), and "unshifts" older elements. The unshift() method returns the new array length.

Example:

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.unshift("Lemon"); // Lemon,Banana,Orange,Apple,Mango
let length = fruits.unshift("Lemon"); //5
```

ix) Array concat():

The concat() method creates a new array by merging (concatenating) existing arrays.

Example:

```
const myGirls = ["Cecilie", "Lone"];
const myBoys = ["Emil", "Tobias", "Linus"];
const myChildren = myGirls.concat(myBoys); // Cecilie,Lone,Emil
    ,Tobias,Linus
```

Merging an Array with Values:

```
const arr1 = ["Emil", "Tobias", "Linus"];
const myChildren = arr1.concat("Peter"); //Emil,Tobias,Linus
    ,Peter
```

Merging Three Arrays:

```
const arr1 = ["Cecilie", "Lone"];
const arr2 = ["Emil", "Tobias", "Linus"];
const arr3 = ["Robin", "Morgan"];
const myChildren = arr1.concat(arr2, arr3); //Cecilie,Lone,Emil
    ,Tobias,Linus,Robin,Morgan
```

x) Array splice():

The splice() method can be used to add new items to an array.

Example:

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.splice(2, 0, "Lemon", "Kiwi"); // Banana,Orange,Lemon
    ,Kiwi,Apple,Mango
```

Note:

The first parameter (2) defines the position where new elements should be added (spliced in).

The second parameter (0) defines how many elements should be removed.

The rest of the parameters ("Lemon" , "Kiwi") define the new elements to be added.

The splice() method returns an array with the deleted items:

Example:

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
let removed = fruits.splice(2, 2, "Lemon", "Kiwi"); // Apple
    ,Mango
```

The splice() methods can be used to remove array elements:

Example:

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.splice(0, 1); // Orange,Apple,Mango
```

xi) Array slice():

The slice() method slices out a piece of an array into a new array.

Example:

```
const fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];
const citrus = fruits.slice(1); // Orange,Lemon,Apple,Mango
```

The slice() method can take two arguments like slice(1, 3). The method then selects elements from the start argument, and up to (but not including) the end argument.

Example:

```
const fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];
const citrus = fruits.slice(1, 3);// Orange,Lemon
```

xii) Sorting an Array:

The sort() method sorts an array alphabetically:

Example:

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.sort();// Apple,Banana,Mango,Orange
```

xiii) Reversing an Array:

The reverse() method reverses the elements in an array:

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.reverse();// Mango,Apple,Orange,Banana
```

Advantages:

- Arrays allow storing multiple values in a single variable, making data handling efficient.
- They can grow or shrink in size dynamically as needed.
- Arrays are useful for iterating and processing large sets of data efficiently.

Object:

1. Desc:

An object in JavaScript is a non-primitive data type that allows you to store multiple values as properties in the form of key-value pairs, where keys are strings (or Symbols) and values can be any data type.

2. Usage:

Used to store and organize data as key-value pairs.

3. Declaration of Object and accessing the elements of Objects:

```
// Object declaration
let person = {
  name: "Alice",
  age: 25,
  city: "New York"
};
// Accessing elements
console.log(person.name);     // Output: Alice
console.log(person["age"]);   // Output: 25
```

4. Operations:

i) Adding a property to the Objects:

```
let person = { name: "Alice", age: 25 };
person.city = "New York";  // Add new property
console.log(person);
```

ii) Updating the value of property:

```
person.age = 30;  // Update age
console.log(person.age);  // Output: 30
```

iii) Deleting the property:

```
delete person.name;  // Remove 'name' property
console.log(person);
```

iv) To check whether the property exists or not:

```
console.log("city" in person);  // Output: true
console.log(person.hasOwnProperty("age"));  // Output: true
```

v) Declaration of Array of objects:

```
let students = [
  { name: "Alice", age: 20 },
  { name: "Bob", age: 22 },
  { name: "Charlie", age: 19 }
];
```

5. Advantage:

Objects allow efficient grouping of related data and functions, making code more organized and manageable.

Strings:

1. Desc:

A string is a sequence of characters enclosed in single quotes, double quotes, or backticks, used to represent textual data.

2. Usage:

Strings are used to store and manipulate text, such as names, messages, or any sequence of characters.

3. Creating and accessing String:

```
// Creating a string
let message = "Hello, World!";

// Accessing characters
console.log(message[0]);     // Output: H
console.log(message.charAt(7)); // Output: W
```

4. Methods:

i) String Length:

The length property returns the length of a string.

```
let text = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
let length = text.length; //26
```

ii) String charAt():

The charAt() method returns the character at a specified index (position) in a string.

```
let text = "HELLO WORLD";
let char = text.charAt(0); //H
```

iii) String charCodeAt():

The charCodeAt() method returns the code of the character at a specified index in a string.

```
let text = "HELLO WORLD";
let char = text.charCodeAt(0); //72
```

iv) String at():

The at() method returns the character at a specified index (position) in a string.

```
const name = "W3Schools";
let letter = name.at(2); //S
```

v) String slice():

slice() extracts a part of a string and returns the extracted part in a new string. The method takes 2 parameters: start position, and end position (end not included).

```
let text = "Apple, Banana, Kiwi";
let part = text.slice(7, 13); // Banana
```

vi) String substring():

substring() is similar to slice().

```
let str = "Apple, Banana, Kiwi";
let part = str.substring(7, 13); //Banana
```

vii) String toUpperCase():

A string is converted to upper case.

```
let text1 = "Hello World!";
let text2 = text1.toUpperCase(); //HELLO WORLD!
```

viii) String toLowerCase():

A string is converted to lower case.

```
let text1 = "Hello World!";
let text2 = text1.toLowerCase(); //hello world!
```

ix) String concat():

concat() joins two or more strings.

```
let text1 = "Hello";
let text2 = "World";
let text3 = text1.concat(" ", text2); //Hello World!
```

x) String trim():

The trim() method removes whitespace from both sides of a string.

```
let text1 = "      Hello World!      ";
let text2 = text1.trim();
```

xi) String trimStart():

The trimStart() method works like trim(), but removes whitespace only from the start of a string.

```
let text1 = "     Hello World!     ";
let text2 = text1.trimStart();
```

xii) String trimEnd():

The trimEnd() method works like trim(), but removes whitespace only from the end of a string.

```
let text1 = "     Hello World!     ";
let text2 = text1.trimEnd();
```

xiii) String padStart():

The padStart() method pads a string from the start. It pads a string with another string (multiple times) until it reaches a given length.

```
let text = "5";
let padded = text.padStart(4,"0"); //0005
```

xiv) String padEnd():

The padEnd() method pads a string from the end.

```
let text = "5";
let padded = text.padEnd(4,"0"); //5000
```

xv) String repeat():

The repeat() method returns a string with a number of copies of a string.

```
let text = "Hello world!";
let result = text.repeat(2); //Hello world!Hello world!
```

xvi) String replace():

The replace() method replaces a specified value with another value in a string.

```
let text = "Please visit Microsoft!";
let newText = text.replace("Microsoft", "Google");
//Please visit Google!
```

xvii) String split():

A string can be converted to an array with the split() method.

```
let text = "apple,banana,mango";
let fruits = text.split(",");
console.log(fruits); //["apple", "banana", "mango"]
```

Note:

```
text.split(",")    // Split on commas
text.split(" ")    // Split on spaces
text.split("|")    // Split on pipe
```

xviii) Template Strings:

Template strings (or template literals) are strings enclosed in backticks (`) that allow embedded expressions using ${}.

```
let name = "Alice";
let message = `Hello, ${name}!`;
console.log(message); //Hello, Alice!
```

Conditional Statements:

1. Desc:

The if, else, and else if, control the flow of execution by allowing you to execute different blocks of code based on whether a condition is true or false.

2. Usage:

Conditional statements are used to perform different actions based on different conditions.

3. Syntax:

```
if (condition1) {
  // code if condition1 is true
} else if (condition2) {
  // code if condition2 is true
} else {
  // code if none of the above conditions are true
}
```

Note:

- The if statement evaluates a boolean expression and executes a block of code if the expression evaluates to true.
- The if-else statement provides an alternative block of code to be executed if the expression evaluates to false.

4. Example:

```
let marks = 85;

if (marks >= 90) {
  console.log("Grade: A");
} else if (marks >= 75) {
  console.log("Grade: B");
} else {
  console.log("Grade: C");
}

//Grade: B
```

Looping Statements:

1. Desc:

Looping statements are control structures that repeatedly execute a block of code as long as a specified condition evaluates to true.

2. Usage:

Loops are used to repeat a block of code multiple times.

3. Different Kinds of Loops:

i) For Loop:

Desc: A for loop is used to repeat a block of code a specific number of times.

Syntax:

```
for (initialization; condition; update) {
  // code to be executed
}
```

Example:

```
for (let i = 1; i <= 5; i++) {
  console.log("Count: " + i);
}
```

ii) For In Loop:

Desc: The for...in loop in JavaScript is used to loop through all the keys (property names) of an object one by one.

Syntax:

```
for (key in object) {
  // code block to be executed
}
```

Example:

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

for (let key in person) {
  console.log(key + ": " + person[key]);
}
```

Output:

```
//Output
name: Alice
age: 25
city: Delhi
```

iii) For Of Loop:

Desc: The for...of loop is used to iterate over iterable objects like arrays, strings, maps, sets, etc.

Syntax:

```
for (variable of iterable) {
  // code block to be executed
}
```

Example:

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

for (let color of colors) {
  console.log(color);
}
```

```
//Output:
red
green
blue
```

While Loop:

Desc:

The while loop loops through a block of code as long as a specified condition is true.

Syntax:

```
while (condition) {
  // code block to be executed
}
```

Example:

```
let i = 1;

while (i <= 5) {
  console.log("Count: " + i);
  i++;
}
```

```
//Output:
Count: 1
Count: 2
Count: 3
Count: 4
Count: 5
```

Do While Loop:

Desc:

The Do while loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.

Syntax:

```
do {
  // code block to be executed
}
while (condition);
```

Example:

```
let i = 1;

do {
  console.log("Count: " + i);
  i++;
} while (i <= 5);
```

```
//Output
Count: 1
Count: 2
Count: 3
Count: 4
Count: 5
```