{JavaScript Cheat Sheet}

Introduction to JavaScript

- Introduction

- JavaScript is the language that powers the web through dynamic behaviour
- Alongside HTML and CSS, it is the core technology that makes the Web run.
- **console.log()** → prints the message

- Comments

- **Single Line Comments** - created with forward slashes `//`.

```
// this is a single-line comment
```

- **Multi-Line Comments** - created with `/*` and `*/`.

```
/* This is a multi-line comment.
It spans multiple lines.  */
```

- Arithmetic Operators

Let's take as an example a=10 and b=20:

Operators	Description	Example
+	Addition	`a + b // Returns: 30`
-	Subtraction	`a - b // Returns: -10`
*	Multiplication	`a * b // Returns: 200`
/	Division	`b / a // Returns: 2`
%	Modulo	`a % b // Returns: 0`
**	Exponential	`a ** b // Returns:` 10^{20}
//	Floor Division	`9 // 2 // Returns: 4`

- String Interpolation

- **String Interpolation:** Inserting variables into strings using template literals (backticks `` ` ``).

```
let name = "Alice";
let greeting = `Hello, my name is ${name}.`;
console.log(greeting); // Output:  Hello, my name is Alice
```

- Assignment Operators

An assignment operator assigns a value to its left operand based on the value of its right operand.
Let's take an example **a = 10**

Operator	Description	Example
+=	Addition Assignment	`a += 3; // a is now 13`
-=	Subtraction Assignment	`a -= 5; // a is now 5`
*=	Multiplication Assignment	`a *= 2; // a is now 20`
/=	Division Assignment	`a /= 4; // a is now 2.5`

- Logical Operators

It takes two values (conditions) and returns boolean depending on the operator used.

A	B	A && B	A || B	!A
true	true	true	true	false
true	false	false	true	false
false	true	false	true	true
false	false	false	false	true

Variable and Scope

- Variable

Variables are used to store a piece of data.

- Declare Variable

- **var** → is used to declare a variable in pre-ES6 versions of JavaScript.

```
var age;
age = 23; //age assigned a
value 23
```

- **let** → declare a variable that can be reassigned.

```
let weight;
weight = 30; //assigned a value
to variable weight
```

- **const** → declare a variable with a constant value and cannot be reassigned.

```
const number = 30;
//number cannot be reassigned
```

- Scope

Scope defines where variables and functions can be accessed.

- Global scope → a value/function that can be used anywhere in the entire program.

```
var globalVar = "I am global";
function checkGlobalScope() {
    console.log(globalVar);
    // accessible here
}
console.log(globalVar); //
accessible here too
```

- Block Scope → only accessible within a {...} code block.

```
{
    let blockVar = 20;
    console.log(blockVar);
   // accessible here
}
console.log(blockVar);
// ReferenceError:  blockVar is
not defined
```

Data Types

- Primitive Data Types

Primitive Data types are the basic data types that are built-in in JavaScript.

- **Number** → represents both integer and floating-pointer numbers

```
let num1 = 42;
let num2 = 3.14
```

- **String** → represents a sequence of characters

```
let str = "hello world";
```

- **Boolean** → represents true or false values.

```
let isApple = true;
let isApple = false;
```

- **Undefined** → represents a variable that has been declared but not assigned a value.

```
let x;
console.log(x);
//output:  undefined
```

- **Null** → represents the absence of value

```
let y = null;
```

- **Symbol** → represents unique identifier.

```
let sym = Symbol("unique");
```

- **BigInit** → represents integers with arbitrary precision.

```
let bigInt =
1234567890123456789012345678901n;
```

Non-Primitive (reference) Data Types

Non-primitive data types are not built-in in JavaScript but are user-defined.

- **Objects** → represents collections of properties and methods.

```
let person = {
    firstName:  "John",
    lastName:  "Doe",
    fullName:  function() {
        return this.firstName +
        " " + this.lastName;
    }
};
console.log(person.firstName);
// Output:  John
console.log(person.fullName());
// Output:  John Doe
```

{JavaScript Cheat Sheet}

Control Flow

It determines the order in which statements are executed in a program.

- Comparison Operators

It compares two values and returns a boolean (true or false).

Operators	Description	Example
==	Equal	`5 == 5; // Output: true`
!=	Not equal	`5 != '5'; // Output: false`
===	Strict equal	`5 === 5; // Output: true`
!==	Strict not equal	`5 !== '5'; // Output: true`
>	Greater than	`5 > 3; // Output: true`
<	Less than	`5 < 3; // Output: false`
>=	Greater than or equal	`5 >= 5; // Output: true`
<=	Less than or equal	`5 <= 5; // Output: true`

- If - Else Statement

- **if** → executes a block of code within it's body if a specified condition is true.

```
let x = 10;
if (x > 5) {
    console.log("x is greater than 5");
}
// Output: x is greater than 5
```

- **else** → executes a block of code if the specified condition is false.

```
let y = 3;
if (y > 5) {
    console.log("y is greater than 5");
} else {
   console.log("y is 5 or less");
}
// Output: y is 5 or less
```

- **else if** → tests alternative conditions and execute if the condition is true.

```
let z = 7;
if (z > 10) {
    console.log("z is greater than 10");
} else if (z > 5) {
   console.log("z is greater than 5 but less than or equal to 10");
} else {
   console.log("z is 5 or less");
}
// Output: z is greater than 5 but less than or equal to 10
```

- Ternary Operators

Shorthand of **if-else statement** for binary (two conditions) decisions. If condition evaluates to truth, the first expression is executed, otherwise, the second expression is executed.

```
let a = 8;
let result = (a > 5) ? "a is greater than 5": "a is 5 or less";
console.log(result);
// Output: a is greater than 5
```

Switch and Loops

- Switch Statement

Switch statement allows to check an expression against multiple **case** clauses. If no case matches, then **default** case is executed

```
let food = "salad";
switch (food) {
   case "oyster":
     console.log("Taste of sea");
     break;
   case "pizza":
     console.log("Delicious pie");
     break;
   default:
     console.log("Enjoy your meal.);
}
//output: Enjoy your meal
```

- For Loop

Executes a block of code as long as condition is true.
It has three important instructions **initialisation, stopping condition** and **iteration**;

```
for (let i = 0; i < 4; i += 1) {
    console.log(i);
}
//output: 0, 1, 2, 3
```

- Do While Statement

Executes a block of code once, and then repeats the loop as long as the specified condition is true.

```
let i = 0;
do {
    console.log(i);
    i++;
} while (i < 5);
// Output: 0 1 2 3 4
```

- While Loop

Loops through a block of code as long as the specified condition is true.

```
let i = 0;
while (i < 5) {
console.log(i);
i++;
}
// Output: 0, 1, 2, 3, 4,
```

Functions

It is a reusable set of statements to perform a task or calculate a value

- Function Declaration

A function is declared using **`function`** keyword, **`function name`**, set of parentheses **`()`** and enclosed in set of curly braces **`{}`**

```
function greeting() {
    console.log(Hello there");
}
greeting() //calling the function
//output: Hello there
```

- Function Parameters and Arguments

Paramters are inputs to function when it is declared.
Arguments are values passed in when function is called.

```
function addNumbers(num1, num2) {
    return num1 + num2;
}
addNumbers(3, 4); //function called
//output: 7
```

- Anonymous functions

Functions that do not have name property.

```
const greetings = function() {
    console.log("Hello World");
}
greetings(); //output Hello World
```

- Arrow Functions

It was introduced in ES6 and has arrow function express instead of **`function`** keyword.

```
const sum = (num1, num2) {
    return num1 + num2;
}
console.log(sum(3, 7)) //output 10
```