**1.variable declarations**

1.variable creation:

Program:

let firstname="akshara";

var lastname="sanaboyina";

const age=21;

console.log(firstname,lastname,age);

2.variable scope:

Program:

const a=10;

function outdisplay(){

    let b=20;

    console.log(a);

      function indisplay(){

       let c=30;

       console.log(a);

       console.log(b);

       console.log(c);

      }

      indisplay();

    }

outdisplay();

**2.operators**

2.1.) arithmetic operators:

Program:

let a=10

let b=20;

let c=30;

let addresult=(a+b)\*c;

let mulresult=((a\*b)+(b\*c))/(a+b);

console.log(addresult);

console.log(mulresult);

2.2)comparision opeartors:

Compare values using `==`, `!=`, `===`, `!==`, `<`, `>`, `<=`, and `>=`.

Program:

let a = 10;

let b = 5;

console.log(a==b);

console.log(a!=b);

console.log(a===b);

console.log(a!==b);

console.log(a>b);

console.log(a<b);

console.log(a>=b);

console.log(a<=b);

3)Logical opeartors:

##Combine conditions using `&&`, `||`, and `!`.

Program:

let a=30;

let b=20;

let c=10;

//&& opeartor

console.log(a>b&&a>c);//true

console.log(a>b&&a<c);//false

//|| opeartor

console.log(a>b||a>c);//true

console.log(a>b||a<c);//true

//! opeartor

let d=a>b&&a>c;

console.log(!d);//false

Data Types

1)Primitive data types

##Create variables of different primitive types (numbers, strings, booleans, null, undefined).

Program:

let numberVar = 30;

let stringVar = "Hello, World!";

let booleanVar = true;

let nullVar = null;

let undefinedVar;

console.log("numberVar value is:", numberVar, "Type:", typeof(numberVar));

console.log("stringVar value is:", stringVar, "Type:", typeof (stringVar));

console.log("booleanVar value is:", booleanVar, "Type:", typeof (booleanVar));

console.log("nullVar value is:", nullVar, "Type:", typeof (nullVar));

console.log("undefinedVar value is:", undefinedVar, "Type:", typeof (undefinedVar));

2)Object data type

##Create an object with properties and methods ,Access and modify its properties , Call its methods.

Program:

let car = {

brand: "toyoto",

model: "glanza",

year: 2022,

color: "grey",

// Method to get car details

getDetails: function() {

return `${this.year} ${this.brand} ${this.model} (${this.color})`;

},

// Method to change the color of the car

paintCar: function(newColor) {

this.color = newColor;

console.log(`The car has been painted ${newColor}.`);

}

};

// Access and modify properties

console.log("Original Car Details:");

console.log(car.getDetails());

console.log("\nModifying car properties...");

car.year = 2024; // Modifying the year

car.brand = "tata"; // Modifying the brand

console.log("\nModified Car Details:");

console.log(car.getDetails());

// Call the method to paint the car

console.log("Painting the car");

car.paintCar("Blue");

console.log("Final Car Details:");

console.log(car.getDetails());

Type conversion and coercion

1)explicit type conversion

## Use `Number()`, `String()`, and `Boolean()` to convert values between different types. Print the results to the console.

Program:

// Converting to Number

let strToNumber = Number("123");

let boolToNumber = Number(true);

let nullToNumber = Number(null);

let undefinedToNumber = Number(undefined);

console.log("String to Number:", strToNumber); // 123

console.log("Boolean to Number:", boolToNumber); // 1

console.log("Null to Number:", nullToNumber); // 0

console.log("Undefined to Number:", undefinedToNumber); // NaN

// Converting to String

let numberToString = String(123);

let boolToString = String(true);

let nullToString = String(null);

let undefinedToString = String(undefined);

console.log("Number to String:", numberToString); // "123"

console.log("Boolean to String:", boolToString); // "true"

console.log("Null to String:", nullToString); // "null"

console.log("Undefined to String:", undefinedToString); // "undefined"

// Converting to Boolean

let numberToBoolean = Boolean(123);

let StringToBoolean = Boolean("sridhar");

let nullToBoolean = Boolean(null);

let undefinedToBoolean = Boolean(undefined);

console.log("Number to Boolean:", numberToBoolean); // true

console.log("Empty String to Boolean:", emptyStringToBoolean); // true

console.log("Null to Boolean:", nullToBoolean); // false

console.log("Undefined to Boolean:", undefinedToBoolean); // false

2)Implicit type coercion

## Observe how JavaScript automatically converts values in certain contexts (e.g., arithmetic operations, comparisons).

Program:

// Implicit coercion to string

console.log('5' + 2); // "52" - Number 2 is coerced to a string and concatenated

// Implicit coercion to number

console.log('5' - 2); // 3 - String '5' is coerced to a number

// Implicit coercion to boolean

console.log(Boolean(0)); // false

console.log(Boolean(1)); // true

// Implicit coercion in comparisons

console.log('5' == 5); // true - String '5' is coerced to a number for comparison

console.log('5' === 5); // false - strict comparison checks type and value

String and template literals

1. String manipulation:

## Concatenate strings using the `+` operator , Access characters using indexing, Find the length of a string , Create substrings.

Program:

// Concatenate strings using the + operator

let str1 = "Hello";

let str2 = "world!";

let concatenatedString = str1 + str2;

console.log("Concatenated String:", concatenatedString); // Output: "Hello, world!"

// Access characters using indexing

let firstChar = concatenatedString[0];

let fourthChar = concatenatedString[3];

console.log("First Character:", firstChar); // Output: "H"

console.log("Last Character:", fourthChar); // Output: "!"

// Find the length of a string

let lengthOfString = concatenatedString.length;

console.log("Length of String:", lengthOfString); // Output: 11

// Create substrings

let substring1 = concatenatedString.substring(0, 5);

let substring2 = concatenatedString.substring(7);

console.log("Substring 1:", substring1); // Output: "Hello"

console.log("Substring 2:", substring2); // Output: "world!"

2)template literals

## Use template literals to embed expressions within strings.

Program:

const name = "Akshara";

const age = 20;

const study = "engineering";

// Using template literals to embed expressions and format strings

console.log(`Hello, my name is ${name}.

I am ${age} years old, and I am studying ${study}.

this is a multiline string`);

Decision making statements

1) If-else statements

##Write conditional statements using `if`, `else if`, and `else` , Execute different code blocks based on conditions.

Program:

let number = 15;

if (number > 20) {

console.log("The number is greater than 20.");

}

else if (number > 10) {

console.log("The number is greater than 10 but less than or equal to 20.");

}

else if (number > 5) {

console.log("The number is greater than 5 but less than or equal to 10.");

}

else {

console.log("The number is 5 or less.");

}

2) Switch statements:

##Create switch statements to compare a value against multiple cases. Execute different code blocks based on matching cases.

Program:

let fruit = "apple";

switch (fruit) {

case "apple":

console.log("You have an apple.");

break;

case "banana":

console.log("You have a banana.");

break;

case "orange":

console.log("You have an orange.");

break;

case "mango":

console.log("You have a mango.");

break;

default:

console.log("Fruit not found.");

}

Use strict

1)Enable strict mode:

## Add `"use strict";` at the beginning of your script. Observe how strict mode enforces stricter rules and catches errors.

Program:

"use strict";`

// Outputs: TypeError: console is not defined

// This error occurs because `console` is not defined in strict mode.

// In non-strict mode, `console` would be `undefined` and the code would still

// run, but in strict mode, it throws an error

Function call and its definition

1) Function declaration:

##Define functions with parameters. Call functions with arguments.

Program:

// Defining a function named 'greet' with a parameter 'name'

function greet(name) {

console.log('Hello' + name + '!');

}

// Defining a function named 'addNumbers' with two parameters 'a' and 'b'

function addNumbers(a, b) {

return a + b;

}

// Calling the 'greet' function with the argument 'Akshara'

greet('Akshara'); // Output: Hello Sridhar!

// Call the 'addNumbers' function with the arguments 5 and 3, and log the result

const result = addNumbers(5, 3);

console.log(result); // Output: 8

2) Function expression:

## Create functions as expressions and assign them to variables. Call these functions.

Program:

// Function expression assigned to a variable

const greet = function(name) {

return `Hello, ${name}!`;

};

// Calling the function

console.log(greet('Akshara')); // Output: Hello Akshara!

Passing parameters to functions

1) Passing arguments

## Pass different types of values as arguments to functions. Access and use these arguments within the function.

Program:

function greet(name, age) {

console.log(`Hello I am ${name}! I am ${age} years old.`);

}

greet('Akshara', 20); // Output: Hello I am Akshara! I am 20 years old.

2) Default parameters

##Set default values for function parameters. Use these default values when arguments are not provided.

function greet(name = 'world', age = 21) {

console.log(`Hello, ${name}! I am ${age} years old.`);

}

greet('guys'); // Output: Hello, guys! I am 21 years old.

Return Statement

1) Returning values

##Use the `return` statement to return values from functions. Assign returned values to variables.

function add(a, b) {

return a + b;

}

const result = add(5, 10);

console.log(result); // Output: 15

2) Chaining function calls

##Call functions that return values and pass the returned values as arguments to other functions.

function add(a, b) {

return a + b;

}

function multiply(a, b) {

return a \* b;

}

const result = multiply(add(5, 10), 2);

console.log(result); // Output: 30