### Introduction to Arrays, Creating, Accessing - Arrays, Methods in Arrays

1. Create an array of fruits and print the elements using a for loop.

2. Access the third element of an array and print it.

3. Find the length of an array and print it.

program:

---------

// Creating an array of fruits and print the elements using a for loop

let fruits = ["Apple", "Banana", "Cherry", "strawberry", "guava"];

console.log("List of fruits:");

for (let i = 0; i < fruits.length; i++) {

console.log(fruits[i]);

}

// Access the third element of an array and print it

console.log("Third fruit in the array is:", fruits[2]); // Cherry

//Find the length of an array and print it

console.log("The length of the fruits array is:", fruits.length); // 5

### Operations: Add, Remove - push(), pop(), unshift(), shift() Methods in Arrays

1. Add an element to the end of an array using the `push()` method.

2. Remove the last element from an array using the `pop()` method.

3. Add an element to the beginning of an array using the `unshift()` method.

4. Remove the first element from an array using the `shift()` method.

program:

---------

// Add an element to the end of an array using the push() method

let array = [1, 2, 3];

console.log("Original Array:", array);

array.push(4); // Adds 4 to the end of the array

console.log("After adding the element array is:", array); // Output: [1, 2, 3, 4]

// 2. Remove the last element from an array using the pop() method

array.pop(); // Removes the last element (4)

console.log("After removing the element array is:", array); // Output: [1, 2, 3]

// 3. Add an element to the beginning of an array using the unshift() method

array.unshift(0); // Adds 0 to the beginning of the array

console.log("New array is:", array); // Output: [0, 1, 2, 3]

// 4. Remove the first element from an array using the shift() method

array.shift(); // Removes the first element (0)

console.log("New array is:", array); // Output: [1, 2, 3]

### Introduction to Objects: Creating Objects with its Properties, Accessing Properties with dot

(.) & square bracket [] representations, Appending New Properties to the Existing Objects using

dot & bracket representations

1. Create an object representing a person with properties like name, age, and city.

2. Access the person's name using both dot notation and square bracket notation.

3. Add a new property "occupation" to the person object using dot notation.

4. Add a new property "hobbies" to the person object using square bracket notation.

program:

---------

// 1. Create an object representing a person with properties like name, age, and city.

const person = {

name: 'Akshara Sanaboyina',

age: 20,

city: 'visakhapatnam'

};

// 2. Access the person's name using both dot notation and square bracket notation.

console.log(person.name); // Output: Akshara Sanaboyina

console.log(person['name']); // Output:Akshara Sanaboyina

// 3. Add a new property "occupation" to the person object using dot notation.

person.occupation = 'Software Developer';

console.log(person.occupation); // Output: Software Developer

// 4. Add a new property "hobbies" to the person object using square bracket notation.

person['hobbies'] = ['reading', 'watching movies', 'playing badminton'];

console.log(person['hobbies']); // Output:['reading', 'watching movies', 'playing badminton']

// Display the final object

console.log(person);

### Creating Objects with Properties & Methods, Calling Methods, Understanding about 'this'

keyword, Coding Exercise in Objects Declaration, Properties & Methods Accessing

1. Create a `Car` object with properties like `make`, `model`, and `year`. Add a method `drive()`

that logs a message when called.

program:

---------

const Car = {

make: 'BMW',

model: 'M3',

year: 2023,

drive: function() {

console.log(`The ${this.year} ${this.make} ${this.model} is now driving.`);

}

};

// Calling the drive method

Car.drive(); // Output: The 2023 BMW M3 is now driving.

2. Create a `Calculator` object with methods like `add`, `subtract`, `multiply`, and `divide`. Call

these methods with different numbers.

program:

----------

const Calculator = {

add: function(a, b) {

return a + b;

},

subtract: function(a, b) {

return a - b;

},

multiply: function(a, b) {

return a \* b;

},

divide: function(a, b) {

if (b === 0) {

console.log("Error: Division by zero is not allowed.");

return undefined;

}

return a / b;

}

};

// Calling the methods with different numbers

console.log(Calculator.add(7,8 )); // Output: 15

console.log(Calculator.subtract(10, 3)); // Output: 7

console.log(Calculator.multiply(10,8 )); // Output: 80

console.log(Calculator.divide(32, 4)); // Output: 8

console.log(Calculator.divide(10, 0)); // Output: Error: Division by zero is not allowed.

### Iterations: For Loop, While Loop, Break & Continue Statement

1. Iterate over an array of numbers using a `for` loop and print each number.

program:

---------

const numbers = [1, 2, 3, 4, 5];

for (let i = 0; i < numbers.length; i++) {

console.log(numbers[i]);

}

2. Iterate over an array of strings using a `while` loop and print each string.

program:

---------

const strings = ["apple", "banana", "cherry"];

let index = 0;

while (index < strings.length) {

console.log(strings[index]);

index++;

}

3. Use a `for` loop to find the sum of all elements in an array.

program:

---------

const numbersToSum = [1, 2, 3, 4, 5];

let sum = 0;

for (let i = 0; i < numbersToSum.length; i++) {

sum += numbersToSum[i];

}

console.log("Sum:", sum);

4. Use a `while` loop to find the first even number in an array.

program:

---------

const mixedNumbers = [1, 3, 7, 4, 9, 6];

let i = 0;

while (i < mixedNumbers.length) {

if (mixedNumbers[i] % 2 === 0) {

console.log("First even number:", mixedNumbers[i]);

break;

}

i++;

}

5. Use a `for` loop with a `break` statement to exit the loop when a certain condition is met.

program:

---------

const moreNumbers = [2, 5, 8, 10, 12];

for (let i = 0; i < moreNumbers.length; i++) {

if (moreNumbers[i] === 10) {

console.log("Found 10, exiting loop.");

break;

}

console.log(moreNumbers[i]);

}

6. Use a `for` loop with a `continue` statement to skip certain elements in the array.

program:

---------

const skipNumbers = [1, 2, 3, 4, 5];

for (let i = 0; i < skipNumbers.length; i++) {

if (skipNumbers[i] === 3) {

continue; // Skip the number 3

}

console.log(skipNumbers[i]);

}

### Behind The Scenes: Variable Scope - Global vs Local Variables, Hoisting: Variables,

Functions

1. Demonstrate the difference between global and local variables using examples.

program:

---------

// Global Variable Example

var globalVar = "I am a global variable";

function exampleFunction() {

// Local Variable Example

var localVar = "I am a local variable";

console.log(localVar); // Output: I am a local variable

}

exampleFunction();

console.log(globalVar); // Output: I am a global variable

console.log(localVar); // Error: localVar is not defined

2. Explain the concept of hoisting and its impact on variable declaration and function calls.

3. Write code to illustrate hoisting and its consequences.

program:

---------

console.log(hoistedVar); // Output: undefined (declaration is hoisted, but not the initialization)

var hoistedVar = "I am hoisted";

console.log(hoistedVar); // Output: I am hoisted

### Hosting Coding Exercise, Understanding 'this' keyword, Normal Functions with Keyword

'this', Object's with Keyword 'this', Regular Functions vs Arrow Functions using Keyword 'this',

Understanding about 'arguments' Keyword, Objects vs Primitives

1. Create a function that demonstrates hoisting by calling a function before it's declared.

program:

---------

// Calling the function before declaration

hoistedFunction();//output: hoistedFunction is not a function

function hoistedFunction() {

console.log("This function is hoisted!");

}

2. Explain how the `this` keyword behaves in different contexts (e.g., regular functions, object

methods, arrow functions).

3. Write code to illustrate the difference between regular functions and arrow functions in terms

of `this` binding.

program:

---------

const obj = {

regularFunction: function() {

console.log("Regular Function 'this':", this); // 'this' refers to the obj

},

arrowFunction: () => {

console.log("Arrow Function 'this':", this); // 'this' is inherited from the global scope or enclosing context

}

};

obj.regularFunction(); // 'this' refers to `obj`

obj.arrowFunction(); // 'this' is inherited from the outer scope

4. Use the `arguments` keyword to access all arguments passed to a function.

program:

---------

function showArguments() {

console.log(arguments);

}

showArguments(1, 2, 3); // Outputs: [1, 2, 3]

5. Compare and contrast objects and primitives in terms of their behavior and properties.

program:

---------

// Primitive Example

let a = 10;

let b = a; // 'b' is a copy of 'a'

a = 20;

console.log(b); // Output: 10 (b is not affected by changes to a)

// Object Example

let obj1 = { name: "ANGEL" };

let obj2 = obj1; // 'obj2' is a reference to 'obj1'

obj1.name = "CUTIE";

console.log(obj2.name); // Output: "CUTIE" (obj2 is affected by changes to obj1)