```
// Using 'var' to store biography details in variables
var name = "Majid"; // Example name
var age = 30;
var profession = "Software Developer";
var isEmployed = true;
var hobbies = ["Coding", "Reading", "Traveling"];
//Creating a JS Object for biography with nested objects
var biography = {
  name: name,
  age: age,
  profession: profession,
  isEmployed: isEmployed,
  hobbies: hobbies,
  address: {
    street: "123 Main Street",
    city: "Techville",
    state: "Innovate State",
    country: "Developers Land",
  },
  degreePrograms: {
    undergraduate: "Computer Science",
    postgraduate: "Software Engineering",
    researchFocus: "Artificial Intelligence and Machine Learning",
  },
};
 // Printing specific parts of the biography on the console
console.log("Biography of " + biography.name + ":");
```

console.log("Age: " + biography.age);

console.log("Profession: " + biography.profession);

console.log("Employment Status: " + (biography.isEmployed ? "Employed" : "Unemployed"));

console.log("Hobbies: " + biography.hobbies.join(", "));

console.log("Address: " + biography.address.street + ", " + biography.address.city + ", " + biography.address.state + ", " + biography.address.country);

console.log("Degree Programs:");

console.log("- Undergraduate: " + biography.degreePrograms.undergraduate);

console.log("- Postgraduate: " + biography.degreePrograms.postgraduate);

console.log("- Research Focus: " + biography.degreePrograms.researchFocus);

Activity 1:

1. Get used to the editor, run first JS program.

2. Write biography about yourself and print on console.

3. use 'var' to store your biography in variables, use appropriate primitive types.

4. Create JS Object for your biography key-value pairs. At least 4-5 keys, nested JS Object for Address, DegreePrograms etc

e.g. { name: 'Some Name', age: 34, address: { }, degreeProgram: {} }
Print biography on console (do not print entire object as it is)

Solution:

```
var bio = {
name: 'Zaheer',
gender: 'male',
age: 33,
address: {
home: 'h#89, st98',
city: 'LA',
country: 'US'
}
}
console.log(bio.address.cit
y)
```

Activity 2:

Find the sum of all the multiples of x or y below z.
If we list all the natural numbers
below 10 that are multiples of 3 or 5,
we get 3, 5, 6 and 9. The sum of these
multiples is 23. (3+5+6+9). Find the

sum of all the multiples of x or y
below z.

```
function multiSum(z, x, y)
{
var sum=0;
for (let index = 2; index < z;
index++) {
if(index%x==0 || index%y==0){
sum+=index;
}; };
return sum; };
console.log(multiSum(10,3,5));
```

Activity 3:

Implement min and max methods which returns minimum and maximum value of supplied arguments. Implement your own algorithm to find the minimum and maximum value.
- min(4,8,1,3) // returns 1
- max(4,6,5,3,2) // returns 6

```
function min(...params) {
var min = params[0];
for (let index = 0; index < params.length; index++) {
if(min>params[index]){
min=params[index]
}
}
return min;
}
function max(...params) {
var max = params[0];
for (let index = 0; index < params.length; index++) {
if(max<params[index]){
max=params[index]
}
}
return max;
}
console.log(max(4,8,1,3));
```

Lab Task 1

Implement a program with four functions (add, subtract, multiply and divide). Each function should have different number of arguments passed.

- *First function 'add' should check the undefined arguments within the defined function.*
- *Second function 'subtract' should use the ES6 default parameter to tackle the same problem.*
- *Third function 'multiply' should use the ES6 rest parameters to multiply each argument with one another.*

 Fourth ‘divide’ should use the ‘Arguments’ object to finish the job.

LAB TASK – 01

```
// 1. Add function - Checks for undefined arguments within the function
function add(a, b) {
  if (typeof a === 'undefined' || typeof b === 'undefined') {
    console.log('Error: One or more arguments are undefined.');
    return;
  }
  console.log(`Addition Result: ${a} + ${b} =`, a + b);
}
// 2. Subtract function - Uses ES6 default parameters
function subtract(a = 0, b = 0) {
  console.log(`Subtraction Result: ${a} - ${b} =`, a - b);
}
// 3. Multiply function - Uses ES6 rest parameters
function multiply(...numbers) {
  if (numbers.length === 0) {
    console.log('No numbers provided for multiplication.');
    return;
  }
  let product = numbers.reduce((acc, num) => acc * num, 1);
  console.log(`Multiplication Result: ${numbers.join(' * ')} =`, product);
}
// 4. Divide function - Uses the 'arguments' object
function divide() {
  if (arguments.length < 2) {
```

```
    console.log('Error: At least two arguments are required for division.');
    return;
  }
  let quotient = arguments[0];
  for (let i = 1; i < arguments.length; i++) {
    if (arguments[i] === 0) {
      console.log('Error: Division by zero is not allowed.');
      return;
    }
    quotient /= arguments[i];
  }
  console.log(`Division Result: ${Array.from(arguments).join(' / ')} =`, quotient);
}
// Testing the functions
add(5, 3); // Should print: Addition Result: 5 + 3 = 8
add(5); // Should print an error message for undefined argument
subtract(10, 4); // Should print: Subtraction Result: 10 - 4 = 6
subtract(10); // Uses default parameter: Subtraction Result: 10 - 0 = 10
multiply(2, 3, 4); // Should print: Multiplication Result: 2 * 3 * 4 = 24
multiply(); // Should print a message about no numbers provided
divide(20, 5); // Should print: Division Result: 20 / 5 = 4
divide(20, 0); // Should print an error message for division by zero
divide(100); // Should print an error message about insufficient arguments
```

Lab Task 2

Implement a generic method named SolveThis() which takes a JS object. depending upon the key, it performs the operation and returns another object with the result.
For Example:

```
SolveThis({sum: [3,2,4], max: [2,4,3,5], min: [5,3,4,3]}) // returns { sum: 9, max: 5, min: 3 }
```

It should perform above implemented functions inside, such as, round, abs, ceil, floor, min, max, random etc
Hint:

```
// Create Object dynamically with dynamic keys
var res = {};
res['sum'] = 6;
res['min'] = 7;
console.log(res); // output: Object {sum: 6, min: 7}
```

LAB TASK – 02

```
// Generic method that performs operations based on keys in the input object
function SolveThis(operations) {
  // Initialize an empty result object to store the results of operations
  var res = {};
  // Iterate over each key in the operations object
  for (var key in operations) {
    if (operations.hasOwnProperty(key)) {
      // Retrieve the array of values associated with the current key
      var values = operations[key];
      // Perform operations based on the key
      switch (key) {
        case 'sum':
          res[key] = values.reduce((acc, num) => acc + num, 0);
          break;
        case 'max':
          res[key] = Math.max(...values);
          break;
        case 'min':
          res[key] = Math.min(...values);
```

```
        break;
      case 'round':
        res[key] = values.map(num => Math.round(num));
        break;
      case 'abs':
        res[key] = values.map(num => Math.abs(num));
        break;
      case 'ceil':
        res[key] = values.map(num => Math.ceil(num));
        break;
      case 'floor':
        res[key] = values.map(num => Math.floor(num));
        break;
      case 'random':
        // Generates a random value between the first and second values if provided, otherwise [0, 1)
        res[key] = Math.random() * (values[1] - values[0]) + values[0];
        break;
      default:
        res[key] = 'Operation not supported'; // Handle unsupported operations
        break;
    }
  }
}

// Return the result object with calculated values
return res;
```

```
}


// Example usage
console.log(SolveThis({ sum: [3, 2, 4], max: [2, 4, 3, 5], min: [5, 3, 4, 3] }));
// Output: { sum: 9, max: 5, min: 3 }


console.log(SolveThis({ round: [3.2, 4.7], abs: [-5, -3.4], ceil: [1.2, 3.6], floor: [3.9, 2.1] }));
// Output: { round: [3, 5], abs: [5, 3.4], ceil: [2, 4], floor: [3, 2] }


console.log(SolveThis({ random: [10, 20] }));
// Output: { random: some random number between 10 and 20 }
```

Dynamic Key Handling: The SolveThis function uses a switch statement to check the key of each entry in the input object and then applies the corresponding operation.

Operations Implemented:

sum: Adds all numbers in the array.

max: Finds the maximum value.

min: Finds the minimum value.

round: Rounds each value to the nearest integer.

abs: Returns the absolute value of each number.

ceil: Rounds each number up to the next largest integer.

floor: Rounds each number down to the nearest integer.

random: Generates a random number between two specified bounds.

Dynamic Object Creation: The results are stored in the res object using dynamic keys, and the object is returned at the end.