EX 1:

Use 10 of Objects predefined methods and explain them.

1. Object.keys()

Explanation: Returns an array of a given object's own enumerable property names

Example:

const person = { name: 'John', age: 30, city: 'New York' };

const keys = Object.keys(person);

// Result: keys = ['name', 'age', 'city']

---

2. Object.values()

Explanation: Returns an array of a given object's own enumerable property values

Example:

const person = { name: 'John', age: 30, city: 'New York' };

const values = Object.values(person);

// Result: values = ['John', 30, 'New York']

---

3. Object.entries()

Explanation: Returns an array of a given object's own enumerable property [key, value] pairs.

Example:

const person = { name: 'John', age: 30, city: 'New York' };

const entries = Object.entries(person);

// Result: entries = [['name', 'John'], ['age', 30], ['city', 'New York']]

---

4. Object.assign()

Explanation: Copies the values of all enumerable properties from one or more source objects to a target object, and returns the target object.

Example:

```
const target = { a: 1, b: 2 };

const source = { b: 3, c: 4 };

const result = Object.assign(target, source);

// Result: target = { a: 1, b: 3, c: 4 }, result = { a: 1, b: 3, c: 4 }
```

---

5. Object.hasOwnProperty()

Explanation: Returns a boolean indicating whether the object has the specified property as its own property (not inherited).

Example:

```
const person = { name: 'John', age: 30 };

const hasName = person.hasOwnProperty('name');

// Result: hasName = true
```

---

6. Object.freeze()

Explanation: Freezes an object, preventing new properties from being added, existing properties from being removed, and values from being changed.

Example:

```
const person = { name: 'John', age: 30 };

Object.freeze(person);

person.age = 31; // This assignment will be ignored in strict mode
```

---

7. Object.seal()

Explanation: Seals an object, preventing new properties from being added and marking all existing properties as non-configurable.

Example:

```
const person = { name: 'John', age: 30 };

Object.seal(person);

person.city = 'New York'; // This assignment will be ignored
```

8. Object.create()

Explanation: Creates a new object with the specified prototype object and properties.

Example:

```
const person = { name: 'John', age: 30 };

const newPerson = Object.create(person);

newPerson.city = 'New York';

// Result: newPerson = { city: 'New York' }, newPerson inherits properties from person
```

9.Object.fromEntries()

Explanation: Transforms a list of key-value pairs into an object.

Example:

```
const entries = [['name', 'John'], ['age', 30], ['city', 'New York']];

const person = Object.fromEntries(entries);

// Result: person = { name: 'John', age: 30, city: 'New York' }
```

---

10. Object.getPrototypeOf()

Explanation: Returns the prototype of the specified object.

Example:

```
const person = { name: 'Charlie' };

const prototype = Object.getPrototypeOf(person);

// Result: prototype = Object {}
```

Ex 2 :

Use 10 of Arrays predefined methods and explain them.

1.push()

Explanation: Adds one or more elements to the end of an array and returns the new length of the array.

Example:

```
let fruits = ['apple', 'orange'];

let newLength = fruits.push('banana');

// Result: fruits = ['apple', 'orange', 'banana'], newLength = 3
```

---

2. pop()

Explanation: Removes the last element from an array and returns that element.

Example:

```
let fruits = ['apple', 'orange', 'banana'];

let removedElement = fruits.pop();

// Result: fruits = ['apple', 'orange'], removedElement = 'banana'
```

3.shift()

Explanation: Removes the first element from an array and returns that element. It also updates the length property of the array.

Example:

```
let fruits = ['apple', 'orange', 'banana'];

let shiftedElement = fruits.shift();

// Result: fruits = ['orange', 'banana'], shiftedElement = 'apple'
```

4.unshift()

Explanation: Adds one or more elements to the beginning of an array and returns the new length of the array.

Example:

```
let fruits = ['orange', 'banana'];

let newLength = fruits.unshift('apple');

// Result: fruits = ['apple', 'orange', 'banana'], newLength = 3
```

5.concat()

Explanation: Combines two or more arrays, creating a new array without modifying the original arrays.

Example:

```
let fruits = ['apple', 'orange'];

let moreFruits = ['banana', 'grape'];

let combinedFruits = fruits.concat(moreFruits);

// Result: combinedFruits = ['apple', 'orange', 'banana', 'grape']
```

---

6. indexOf()

Explanation: Returns the first index at which a given element can be found in the array, or -1 if it is not present.

Example:

```
let fruits = ['apple', 'orange', 'banana'];

let indexOfOrange = fruits.indexOf('orange');

// Result: indexOfOrange = 1
```

7.slice()

Explanation: Returns a shallow copy of a portion of an array into a new array. It takes two arguments, start and end, where the start is inclusive and the end is exclusive.

Example:

```
let fruits = ['apple', 'orange', 'banana', 'grape'];

let slicedFruits = fruits.slice(1, 3);

// Result: slicedFruits = ['orange', 'banana']
```

8.splice()

Explanation: Changes the contents of an array by removing or replacing existing elements and/or adding new elements in place.

Example:

```
let fruits = ['apple', 'orange', 'banana'];

fruits.splice(1, 1, 'grape', 'kiwi');

// Result: fruits = ['apple', 'grape', 'kiwi', 'banana']
```

---

9. reverse()

Explanation: Reverses the elements of an array in place.

Example:

```
let fruits = ['apple', 'orange', 'banana'];

fruits.reverse();

// Result: fruits = ['banana', 'orange', 'apple']
```

---

10. filter()

Explanation: Creates a new array with all elements that pass the test implemented by the provided function.

Example:

```
let numbers = [1, 2, 3, 4, 5];

let evenNumbers = numbers.filter(num => num % 2 === 0);

// Result: evenNumbers = [2, 4]
```

Ex3:

Write a nested function and explain the closure -> lexical environment.

```
function createCounter() {
// Variable in the lexical environment of createCounter
  let count = 0;

  function increment() {
    // Inner function has access to the outer function's variable 'count'
    count++;
    console.log(count);
  }
// Return the inner function, creating a closure
  return increment;
}

// Create two independent counters
const counter1 = createCounter();
const counter2 = createCounter();

// Call the counters
counter1(); // Output: 1
counter1(); // Output: 2

counter2(); // Output: 1
counter2(); // Output: 2
```

Explanation:

1.Closure:

A closure is formed when a function is defined inside another function, allowing the inner function to access the outer function's variables, parameters, and even its own parameters.

In this example, “increment()” inner Function is a closure because it's defined inside “createCounter()” outer Function.

2.Lexical Environment:

Lexical environment refers to the context in which a function is declared. It includes the variables, parameters, and functions that are in scope at the time of the function's definition.