Example: Sum of Natural Numbers Using Recursion

```
// program to find the sum of natural numbers using recursion

function sum(num) {
    if(num > 0) {
        return num + sum(num - 1);
    }
    else {
        return num;
    }
 }

// take input from the user
const number = parseInt(prompt('Enter a positive integer: '));

const result = sum(number);

// display the result
console.log(`The sum is ${result}`);
```

Run Code

Output

```
Enter a positive integer: 5
The sum is 15
```

In the above program, the user is prompted to enter a number.

Then the `sum()` function is called by passing the parameter (here **5**) that the user entered.

- If the number is greater than **0**, the function calls itself by decreasing the number by **1**.
- This process continues until the number is **1**. When the number reaches **0**, the program stops.
- If the user enters a negative number, the negative number is returned and the program stops.

Here,

```
sum(5) returns 5 + sum(4)

sum(4) returns 5 + 4 + sum(3)

sum(3) returns 5 + 4 + 3 + sum(2)

sum(2) returns 5 + 4 + 3 + 2 + sum(1)

sum(1) returns 5 + 4 + 3 + 2 + 1 + sum(0)
```

```
sum(0) returns 5 + 4 + 3 + 2 + 1 + 0
```

JavaScript Program to Guess a Random Number

Example: Program to Guess a Number

```
// program where the user has to guess a number generated by a program

function guessNumber() {

    // generating a random integer from 1 to 10
    const random = Math.floor(Math.random() * 10) + 1;

    // take input from the user
    let number = parseInt(prompt('Guess a number from 1 to 10: '));

    // take the input until the guess is correct
    while(number !== random) {
        number = parseInt(prompt('Guess a number from 1 to 10: '));
    }

    // check if the guess is correct
    if(number == random) {
        console.log('You guessed the correct number.');
    }

  }

// call the function
guessNumber();
```

Run Code

Output

```
Guess a number from 1 to 10: 1
Guess a number from 1 to 10: 8
Guess a number from 1 to 10: 5
Guess a number from 1 to 10: 4
You guessed the correct number.
```

> **Note**: You will get different output values each time you run the program because each time a different number is generated.

In the above program, the `guessNumber()` function is created where a random number from **1** to **10** is generated using `Math.random()` function.

To learn more about how to generate a random number, visit JavaScript Generate Random Number.

- The user is prompted to guess a number from **1** to **10**.
- The `parseInt()` converts the numeric string value to an integer value.
- The `while` loop is used to take input from the user until the user guesses the correct answer.
- The `if...else` statement is used to check the condition. The equal to `==` operator is used to check if the guess was correct.

```
if(number == random)
```

Example: Find Factorial Using Recursion

```
// program to find the factorial of a number
function factorial(x) {

    // if number is 0
    if (x == 0) {
        return 1;
    }

    // if number is positive
    else {
        return x * factorial(x - 1);
    }
}

// take input from the user
const num = prompt('Enter a positive number: ');

// calling factorial() if num is positive
if (num >= 0) {
    const result = factorial(num);
    console.log(`The factorial of ${num} is ${result}`);
}
else {
    console.log('Enter a positive number.');
```

```
}
Run Code
```

Output

```
Enter a positive number: 4
The factorial of 4 is 24
```

In the above program, the user is prompted to enter a number.

When the user enters a negative number, a message `Enter a positive number.` is shown.
When the user enters a positive number or **0**, the function `factorial(num)` gets called.

- If the user enters the number **0**, the program will return **1**.
- If the user enters a number greater than **0**, the program will recursively call itself by decreasing the number.
- This process continues until the number becomes 1. Then when the number reaches 0, 1 is returned.

Here,

```
factorial(4) returns 4 * factorial(3)

factorial(3) returns 4 * 3 * factorial(2)

factorial(2) returns 4 * 3 * 2 * factorial(1)

factorial(1) returns 4 * 3 * 2 * 1 * factorial(0)

factorial(0) returns 4 * 3 * 2 * 1 * 1
```

Example: Shuffle Deck of Cards

```
// program to shuffle the deck of cards

// declare card elements
const suits = ["Spades", "Diamonds", "Club", "Heart"];
const values = [
  "Ace",
  "2",
  "3",
  "4",
  "5",
  "6",
```

```
  "7",
  "8",
  "9",
  "10",
  "Jack",
  "Queen",
  "King",
];

// empty array to contain cards
let deck = [];

// create a deck of cards
for (let i = 0; i < suits.length; i++) {
    for (let x = 0; x < values.length; x++) {
        let card = { Value: values[x], Suit: suits[i] };
        deck.push(card);
    }
}

// shuffle the cards
for (let i = deck.length - 1; i > 0; i--) {
    let j = Math.floor(Math.random() * i);
    let temp = deck[i];
    deck[i] = deck[j];
    deck[j] = temp;
}

console.log('The first five cards are:');

// display 5 results
for (let i = 0; i < 5; i++) {
    console.log(`${deck[i].Value} of ${deck[i].Suit}`)
}
```
Run Code

Output

```
The first five cards are:
4 of Club
5 of Diamonds
Jack of Diamonds
2 of Club
4 of Spades
```

In the above program, the `suits` and `values` variables contain the elements of a card.

The nested `for` loop is used to create a deck of cards.

- We need to create a deck of cards containing each `suits` with all the `values`. So the first `for` loop iterates over all the `suits` and the second `for` loop iterates over the `values`. Then, the elements are created and added to the `deck` array.
- The array elements are stored as an object as:

```
[{Value: "Ace", Suit: "Spades"},{Value: "2", Suit: "Spades"}.....]
```

The second `for` loop is used to shuffle the deck of cards.

- `Math.random()` generates a random number.
- `Math.floor()` returns the number by decreasing the value to the nearest integer value.
- A random number is generated between **0** and **51** and two card positions are swapped.

The third `for` loop is used to display the first five cards in the new deck.

JavaScript Program to Create Objects in Different Ways

Example 1: Using object literal

```
// program to create JavaScript object using object literal
const person = {
    name: 'John',
    age: 20,
    hobbies: ['reading', 'games', 'coding'],
    greet: function() {
        console.log('Hello everyone.');
    },
    score: {
        maths: 90,
        science: 80
    }
};

console.log(typeof person); // object

// accessing the object value
console.log(person.name);
console.log(person.hobbies[0]);
person.greet();
console.log(person.score.maths);
```

Run Code

Output

```
object
John
reading
Hello everyone.
90
```

In this program, we have created an object named **person**.

You can create an object using an object literal. An object literal uses `{ }` to create an object directly.

An object is created with a **key:value** pair.

You can also define functions, arrays and even objects inside of an object. You can access the value of the object using dot `.` notation.

The syntax for creating an object using instance of an object is:

```
const objectName = new Object();
```

Example 2: Create an Object using Instance of Object Directly

```
// program to create JavaScript object using instance of an object
const person = new Object ( {
    name: 'John',
    age: 20,
    hobbies: ['reading', 'games', 'coding'],
    greet: function() {
        console.log('Hello everyone.');
    },
    score: {
        maths: 90,
        science: 80
    }
});

console.log(typeof person); // object

// accessing the object value
console.log(person.name);
console.log(person.hobbies[0]);
person.greet();
console.log(person.score.maths);
```

Run Code

Output

```
object
John
reading
Hello everyone.
90
```

Here, the `new` keyword is used with the `Object()` instance to create an object.

Example 3: Create an object using Constructor Function

```
// program to create JavaScript object using instance of an object

function Person() {
    this.name = 'John',
    this.age = 20,
    this.hobbies = ['reading', 'games', 'coding'],
    this.greet = function() {
        console.log('Hello everyone.');
    },
    this.score = {
        maths: 90,
        science: 80
    }

}

const person = new Person();

console.log(typeof person); // object

// accessing the object value
console.log(person.name);
console.log(person.hobbies[0]);
person.greet();
console.log(person.score.maths);
```

Run Code

Output

```
object
John
reading
Hello everyone.
```

```
90
```

In the above example, the `Person()` constructor function is used to create an object using the `new` keyword.

`new Person()` creates a new object.

JavaScript Program to Remove a Property from an Object

An object is written in a **key/value** pair. The **key/value** pair is called a property. For example,

```
const student = {
    name: 'John',
    age: 22
}
```

Here, `name: 'John'` and `age: 22` are the two properties of a student object.

Example: Remove a Property From an Object

```
// program to remove a property from an object

// creating an object
const student = {
    name: 'John',
    age: 20,
    hobbies: ['reading', 'games', 'coding'],
    greet: function() {
        console.log('Hello everyone.');
    },
    score: {
        maths: 90,
        science: 80
    }
};

// deleting a property from an object
```

```
delete student.greet;
delete student['score'];

console.log(student);
```
Run Code

Output

```
{
  age: 20,
  hobbies: ["reading", "games", "coding"],
  name: "John"
}
```

In the above program, the `delete` operator is used to remove a property from an object.

You can use the `delete` operator with `.` or `[ ]` to remove the property from an object.

> **Note**: You should not use the delete operator on predefined JavaScript object properties.

Example 1: Check if Key Exists in Object Using in Operator

```
// program to check if a key exists

const person = {
    id: 1,
    name: 'John',
    age: 23
}

// check if key exists
const hasKey = 'name' in person;

if(hasKey) {
    console.log('The key exists.');
}
else {
    console.log('The key does not exist.');
}
```
Run Code

Output

```
The key exists.
```

In the above program, the `in` operator is used to check if a key exists in an object.
The `in` operator returns `true` if the specified key is in the object, otherwise it returns `false`.

Example 2: Check if Key Exists in Object Using hasOwnProperty()

```
// program to check if a key exists

const person = {
    id: 1,
    name: 'John',
    age: 23
}

//check if key exists
const hasKey = person.hasOwnProperty('name');

if(hasKey) {
    console.log('The key exists.');
}
else {
    console.log('The key does not exist.');
}
```

Run Code

Output

```
The key exists.
```

In the above program, the `hasOwnProperty()` method is used to check if a key exists in an object. The `hasOwnProperty()` method returns `true` if the specified key is in the object, otherwise it returns `false`.

JavaScript Program to Clone a JS Object

A JavaScript object is a complex data type that can contain various data types. For example,

```
const person = {
    name: 'John',
    age: 21,
}
```

Here, `person` is an object. Now, you can't clone an object by doing something like this.

```
const copy = person;
console.log(copy); // {name: "John", age: 21}
```

In the above program, the `copy` variable has the same value as the `person` object. However, if you change the value of the `copy` object, the value in the `person` object will also change. For example,

```
copy.name = 'Peter';
console.log(copy.name); // Peter
console.log(person.name); // Peter
```

The change is seen in both objects because objects are **reference types**. And both <code>copy</code> and <code>person</code> are pointing to the same object.

Example 1. Clone the Object Using Object.assign()

```
// program to clone the object

// declaring object
const person = {
    name: 'John',
    age: 21,
}

// cloning the object
const clonePerson = Object.assign({}, person);

console.log(clonePerson);

// changing the value of clonePerson
clonePerson.name = 'Peter';

console.log(clonePerson.name);
console.log(person.name);
```

Run Code

Output

```
{name: "John", age: 21}
Peter
John
```

The `Object.assign()` method is part of the **ES6** standard. The `Object.assign()` method performs deep copy and copies all the properties from one or more objects.

> **Note**: The empty `{}` as the first argument ensures that you don't change the original object.

Example 2: Clone the Object Using Spread Syntax

```
// program to clone the object
// declaring object
const person = {
    name: 'John',
    age: 21,
}

// cloning the object
const clonePerson = { ... person}

console.log(clonePerson);

// changing the value of clonePerson
clonePerson.name = 'Peter';

console.log(clonePerson.name);
console.log(person.name);
```

Run Code

Output

```
{name: "John", age: 21}
Peter
John
```

The spread syntax ... was introduced in the later version(ES6).

The spread syntax can be used to make a shallow copy of an object. This means it will copy the object. However, the deeper objects are referenced. For example,

```
const person = {
    name: 'John',
    age: 21,
```

```
    // the inner objects will change in the shallow copy
    marks: { math: 66, english: 73}
}

// cloning the object
const clonePerson = { ... person}

console.log(clonePerson); // {name: "John", age: 21, marks: {…}}

// changing the value of clonePerson
clonePerson.marks.math = 100;

console.log(clonePerson.marks.math); // 100
console.log(person.marks.math); // 100
```
Run Code

Here, when the inner object value `math` is changed to **100** of `clonePerson` object, the value of the `math` key of the `person` object also changes.

Example 3: Clone the Object Using JSON.parse()

```
// program to clone the object
// declaring object
const person = {
    name: 'John',
    age: 21,
}

// cloning the object
const clonePerson = JSON.parse(JSON.stringify(person));

console.log(clonePerson);

// changing the value of clonePerson
clonePerson.name = 'Peter';

console.log(clonePerson.name);
console.log(person.name);
```
Run Code

Output

```
{name: "John", age: 21}
Peter
```

```
John
```

In the above program, the `JSON.parse()` method is used to clone an object.

> **Note**: `JSON.parse()` only works with `Number` and `String` object literal. It does not work with an object literal with `function` or `symbol` properties.

Example 1: Loop Through Object Using for...in

```
// program to loop through an object using for...in loop

const student = {
    name: 'John',
    age: 20,
    hobbies: ['reading', 'games', 'coding'],
};

// using for...in
for (let key in student) {
    let value;

    // get the value
    value = student[key];

    console.log(key + " - " +  value);
}
```

Run Code

Output

```
name - John
age - 20
hobbies - ["reading", "games", "coding"]
```

In the above example, the `for...in` loop is used to loop through the `student` object.
The value of each key is accessed by using `student[key]`.

> **Note**: The `for...in` loop will also count inherited properties.

For example,

```
const student = {
    name: 'John',
    age: 20,
    hobbies: ['reading', 'games', 'coding'],
```

```
};

const person = {
    gender: 'male'
}

// inheriting property
student.__proto__ = person;

for (let key in student) {
    let value;

    // get the value
    value = student[key];

    console.log(key + " - " +  value);
}
```

Run Code

Output

```
name - John
age - 20
hobbies - ["reading", "games", "coding"]
gender - male
```

If you want, you can only loop through the object's own property by using the `hasOwnProperty()` method.

```
if (student.hasOwnProperty(key)) {
    ++count:
}
```

Example 2: Loop Through Object Using Object.entries and for...of

```
// program to loop through an object using for...in loop

const student = {
    name: 'John',
    age: 20,
    hobbies: ['reading', 'games', 'coding'],
};

// using Object.entries
// using for...of loop
```

```
for (let [key, value] of Object.entries(student)) {
    console.log(key + " - " +  value);
}
```
Run Code

Output

```
name - John
age - 20
hobbies - ["reading", "games", "coding"]
```

In the above program, the object is looped using the `Object.entries()` method and the `for...of` loop.

The `Object.entries()` method returns an array of a given object's key/value pairs.

The `for...of` loop is used to loop through an array.

Example 1: Merge Property of Two Objects Using Object.assign()

```
// program to merge property of two objects

// object 1
const person = {
    name: 'Jack',
    age:26
}

// object 2
const student = {
    gender: 'male'
}

// merge two objects
const newObj = Object.assign(person, student);

console.log(newObj);
```
Run Code

Output

```
{
    name: "Jack",
    age: 26,
    gender: "male"
}
```

In the above example, two objects are merged into one using the `Object.assign()` method.

The `Object.assign()` method returns an object by copying the values of all enumerable properties from one or more source objects.

Example 2: Merge Property of Two Objects Using Spread Operator

```
// program to merge property of two objects

// object 1
const person = {
    name: 'Jack',
    age:26
}

// object 2
const student = {
    gender: 'male'
}

// merge two objects
const newObj = {...person, ...student};

console.log(newObj);
```

Run Code

Output

```
{
    name: "Jack",
    age: 26,
    gender: "male"
}
```

In the above example, two objects are merged together using the spread operator ....

> **Note**: In both the above examples, if the two objects have the same key, then the second object's key overwrites the first object's key.

Example 1: Count the Number of Key in an Object Using for...in

```
// program to count the number of keys/properties in an object

const student = {
    name: 'John',
    age: 20,
    hobbies: ['reading', 'games', 'coding'],
};

let count = 0;

// loop through each key/value
for(let key in student) {

    // increase the count
    ++count;
}

console.log(count);
```
Run Code

Output

```
3
```

The above program counts the number of keys/properties in an object using the `for...in` loop.

The `count` variable is initially **0**. Then, the `for...in` loop increases the count by **1** for every key/value in an object.

> **Note**: While using the `for...in` loop, it will also count inherited properties.

For example,

```
const student = {
    name: 'John',
    age: 20,
    hobbies: ['reading', 'games', 'coding'],
};

const person = {
    gender: 'male'
}

student.__proto__ = person;

let count = 0;

for(let key in student) {
```

```
    // increase the count
    ++count;
}

console.log(count); // 4
```
Run Code

If you only want to loop through the object's own property, you can use the `hasOwnProperty()` method.

```
if (student.hasOwnProperty(key)) {
    ++count:
}
```

Example 2: Count the Number of Key in an Object Using Object.key()

```
// program to count the number of keys/properties in an object

const student = {
    name: 'John',
    age: 20,
    hobbies: ['reading', 'games', 'coding'],
};

// count the key/value
const result = Object.keys(student).length;

console.log(result);
```
Run Code

Output

```
3
```

In the above program, the `Object.keys()` method and the `length` property are used to count the number of keys in an object.

The `Object.keys()` method returns an array of a given object's own enumerable property names i.e. `["name", "age", "hobbies"]`.

The `length` property returns the length of the array.

Example 1: Add Key/Value Pair to an Object Using Dot Notation

```
// program to add a key/value pair to an object

const person = {
    name: 'Monica',
    age: 22,
    gender: 'female'
}

// add a key/value pair
person.height = 5.4;

console.log(person);
```

Run Code

Output

```
{
    name: "Monica",
    age: 22,
    gender: "female",
    height: 5.4
}
```

In the above example, we add the new property `height` to the `person` object using the dot notation . i.e. `person.height = 5.4;`.

Example 2: Add Key/Value Pair to an Object Using Square Bracket Notation

```
// program to add a key/value pair to an object

const person = {
    name: 'Monica',
    age: 22,
    gender: 'female'
}

// add a key/value pair
person['height'] = 5.4;

console.log(person);
```

Run Code

Output

```
{
    name: "Monica",
    age: 22,
    gender: "female",
    height: 5.4
}
```

In the above example, we add the new property `height` to the `person` object using the square bracket notation `[]` i.e. `person['height'] = 5.4;`.

JavaScript Program to Convert Objects to Strings

Example 1: Convert Object to String Using JSON.stringify()

```
// program to convert an object to a string

const person = {
    name: 'Jack',
    age: 27
}

const result =  JSON.stringify(person);

console.log(result);
console.log(typeof result);
```

Run Code

Output

```
{"name":"Jack","age":27}
string
```

In the above example, the `JSON.stringify()` method is used to convert an object to a string. The `typeof` operator gives the data type of the `result` variable.

Example 2: Convert Object to String Using String()

```
// program to convert an object to a string

const person = {
    name: 'Jack',
    age: 27
}

const result1 = String(person);
const result2 = String(person['name']);

console.log(result1);
console.log(result2);

console.log(typeof result1);
```

Run Code

Output

```
[object Object]
Jack
string
```

In the above example, the `String()` function converts the value of an object to a string.

When using the `String()` function on an `Object`, the converted result will give `[object Object]`.

The `typeof` operator gives the data type of the `result` variable.