1. Square Star Pattern in Javascript

```
*****
*****
*****
*****
*****
```

To create a **square star pattern** run 2 nested for loop. Execute each loop for 'n' number of times, where 'n' is number of rows/columns in the square, i.e `for(let i = 0; i < n; i++)`.

The internal loop will run for 'n' number of times which will print starts in a row and a newline at the end of the loop (`\n`) while the outer loop will run the internal loop for 'n' number of times which will print starts in a columns

Example

```
let n = 5; // row or column count
// defining an empty string
let string = "";

for(let i = 0; i < n; i++) { // external loop
  for(let j = 0; j < n; j++) { // internal loop
    string += "*";
  }
  // newline after each row
  string += "\n";
}
// printing the string
console.log(string);
```

▶ Try It Run Here

2. Hollow Square Pattern

```
*****
*   *
*   *
*   *
*****
```

Steps to create a **hollow square star pattern** are:

1. Create a variable to store the string and assign it with an empty string
2. Create a for loop to run for 'n' number of times, where 'n' is number of rows/columns in the square, i.e `for(let i = 0; i < n; i++)`
3. Inside the loop, create a for loop that prints a star (*) at the beginning and end of the line and space in between
4. Also, keep in mind that the first and last row should have only stars
5. Add a new line after each row

Example

```
let n = 5; // row or column count
// defining an empty string
let string = "";

for(let i = 0; i < n; i++) { // external loop
  for(let j = 0; j < n; j++) { // internal loop
    if(i === 0 || i === n - 1) {
      string += "*";
    }
    else {
      if(j === 0 || j === n - 1) {
        string += "*";
      }
      else {
        string += " ";
      }
    }
  }
  // newline after each row
  string += "\n";
}
// printing the string
console.log(string);
```

▶ **Try It** Run Here

3. Right Triangle Pattern in Javascript

```
    *
   **
```

```
  ***
 ****
*****
```

To create a **right triangle pattern in javascript** you will have to deal with 3 loops, 1 of which is external and 2 are internal. The external loop will execute internal loops for 'n' number of times and the internal loop will design a pattern for each row.

From the above pattern, you can see each row has a series of stars and spaces. The number of stars in a row starts from 1 preceding with 'n-1' spaces and ends with 'n' star and 0 spaces.

Create 2 internal loops, 1st print `n - i` spaces and 2nd print `i` stars, where `i` is the number of times the external loop is executed.

Example

```
let n = 5;
let string = "";
for (let i = 1; i <= n; i++) {
  // printing spaces
  for (let j = 0; j < n - i; j++) {
    string += " ";
  }
  // printing star
  for (let k = 0; k < i; k++) {
    string += "*";
  }
  string += "\n";
}
console.log(string);
```

▶ Try It Run Here

4. Left Triangle Pattern in Javascript

```
*
**
***
****
*****
```

To create the **left triangle pattern in javascript** again run 2 nested `for loop` external loop will take care of columns of pattern and the internal loop will print rows of the pattern.

You can observe from the above-shown pattern that we need to run an external loop for 'n' time while the internal loop runs for 1 time in the first execution, 2 times in the second execution, and so on till 'n' times.

You can use the value of `i` from the external loop which will increase from 1 to 'n' inside the internal loop as a condition.

Example

```
let n = 5;
let string = "";
for (let i = 1; i <= n; i++) {
  for (let j = 0; j < i; j++) {
    string += "*";
  }
  string += "\n";
}
console.log(string);
```

▶ **Try It** Run Here

5. Downward Triangle Star Pattern

```
*****
****
***
**
*
```

To create a **downward triangle star pattern** use a nested loop, it is also known as a reverse star pattern. From the above-shown pattern, you can see the number of stars decreases from 'n' to 1.

Run a `for loop` inside an external loop whose number of iterations decreases from 'n' to 1.

Example

```
let n = 5;
```

```
let string = "";
for (let i = 0; i < n; i++) {
  // printing star
  for (let k = 0; k < n - i; k++) {
    string += "*";
  }
  string += "\n";
}
console.log(string);
```

▶ **Try It** Run Here

6. Hollow Triangle Star Pattern

```
*
**
* *
*  *
*   *
******
```

Steps to create a **hollow triangle star pattern** are:

1. Run 2 nested loops, 1st for 'n' times and 2nd for 1 time for the first row, 2nd for 2 times for the second row, and so on till 'n' times
2. Print star for first and last position in each row and space for other positions
3. In the last line print star at each position

Example

```
let n = 6;
let string = "";

for (let i = 1; i <= n; i++) {
  // printing star
  for (let j = 0; j < i; j++) {
    if(i === n) {
      string += "*";
    }
```

```
    else {
      if (j == 0 || j == i - 1) {
        string += "*";
      }
      else {
        string += " ";
      }
    }
  }
  string += "\n";
}
console.log(string);
```

▶ Try It Run Here

7. Javascript Pyramid Pattern

```
    *
   ***
  *****
 *******
*********
```

The **Pyramid star pattern** is a famous star pattern, you can see the shape of the pattern above.

It uses 2 loops inside the external loop one for printing spaces and the other to print stars.

The number of spaces in a row is `n - i` in each row and the number of stars in a row is `2 * i - 1`.

Example

```
let n = 5;
let string = "";
// External loop
for (let i = 1; i <= n; i++) {
  // printing spaces
  for (let j = 1; j <= n - i; j++) {
    string += " ";
  }
  // printing star
```

```
  for (let k = 0; k < 2 * i - 1; k++) {
    string += "*";
  }
  string += "\n";
}
console.log(string);
```

▶ Try It Run Here

Also, check out alphabet patterns in JavaScript.

8. Reversed Pyramid Star Pattern

```
*********
 *******
  *****
   ***
    *
```

The **reversed pyramid star pattern** is a pyramid pattern upside-down.

It uses 2 loops inside the external loop one for printing spaces and the other to print the star. First loop prints spaces and other loop print stars. Here is the code of the reverse pyramid star pattern.

The number of spaces in a row is `i` and the number of stars is `2 * (n - i) - 1`.

Example

```
let n = 5;
let string = "";
// External loop
for (let i = 0; i < n; i++) {
  // printing spaces
  for (let j = 0; j < i; j++) {
    string += " ";
  }
  // printing star
  for (let k = 0; k < 2 * (n-i) - 1; k++) {
    string += "*";
  }
  string += "\n";
```

```
}
console.log(string);
```

▶ Try It Run Here

Stay Ahead, Learn More

- Alphabet Pattern Programs in Javascript
- Number Pattern Programs in Javascript

9. Hollow Pyramid Star Pattern

```
    *
   * *
  *   *
 *     *
*********
```

Steps to create a **hollow pyramid star pattern** are:

1. Run 2 nested loops, 1st for 'n' times and there are 2 internal loops one to print the first series of spaces and the other to print star and space series
2. Print star for first and last position in each row and space in between
3. The print star at each position in the last row

Example

```
let n = 5;
let string = "";

// External loop
for (let i = 1; i <= n; i++) {
  // printing spaces
  for (let j = 1; j <= n - i; j++) {
    string += " ";
  }
  // printing star
  for (let k = 0; k < 2 * i - 1; k++) {
    if(i === 1 || i === n) {
      string += "*";
```

```
    }
    else {
      if(k === 0 || k === 2 * i - 2) {
        string += "*";
      }
      else {
        string += " ";
      }
    }
  }
  string += "\n";
}
console.log(string);
```

▶ Try It Run Here

10. Diamond Pattern in Javascript

```
    *
   ***
  *****
 *******
*********
 *******
  *****
   ***
    *
```

The **diamond star pattern** is a combination of the pyramid and the reverse pyramid star pattern.

Observe in the above pattern it is a pyramid and a reverse pyramid together. Use the above concepts to create a diamond shape.

Example

```
let n = 5;
let string = "";
// Upside pyramid
for (let i = 1; i <= n; i++) {
  // printing spaces
  for (let j = n; j > i; j--) {
    string += " ";
  }
```

```
  // printing star
  for (let k = 0; k < i * 2 - 1; k++) {
    string += "*";
  }
  string += "\n";
}
// downside pyramid
for (let i = 1; i <= n - 1; i++) {
  // printing spaces
  for (let j = 0; j < i; j++) {
    string += " ";
  }
  // printing star
  for (let k = (n - i) * 2 - 1; k > 0; k--) {
    string += "*";
  }
  string += "\n";
}
console.log(string);
```

▶ Try It Run Here

11. Hollow Diamond Pattern

```
    *
   * *
  *   *
 *     *
*       *
 *     *
  *   *
   * *
    *
```

A hollow diamond pattern is a simple diamond shape that has only a boundary that is made up of stars. It is a combination of the diamond and the reverse diamond pattern.

Example

```
let n = 5;
let string = "";
// Upside pyramid
// upside diamond
```

```
for (let i = 1; i <= n; i++) {
  // printing spaces
  for (let j = n; j > i; j--) {
    string += " ";
  }
  // printing star
  for (let k = 0; k < i * 2 - 1; k++) {
    if (k === 0 || k === 2 * i - 2) {
      string += "*";
    }
    else {
      string += " ";
    }
  }
  string += "\n";
}
// downside diamond
for (let i = 1; i <= n - 1; i++) {
    // printing spaces
    for (let j = 0; j < i; j++) {
      string += " ";
    }
    // printing star
    for (let k = (n - i) * 2 - 1; k >= 1; k--) {
      if (k === 1 || k === (n - i) * 2 - 1) {
        string += "*";
      }
      else {
        string += " ";
      }
    }
    string += "\n";
  }
console.log(string);
```

▶ Try It Run Here

12. Hourglass Star Pattern

```
*********
 *******
  *****
   ***
```

```
    *
   ***
  *****
 *******
*********
```

The **hourglass star pattern** is also made up of pyramid and reverse pyramid star patterns.

Observe in the above pattern it is a reverse pyramid and a pyramid together. Use the above concepts to create a diamond shape.

Example

```
let n = 5;
let string = "";
// Reversed pyramid pattern
for (let i = 0; i < n; i++) {
  // printing spaces
  for (let j = 0; j < i; j++) {
    string += " ";
  }
  // printing star
  for (let k = 0; k < (n - i) * 2 - 1; k++) {
    string += "*";
  }
  string += "\n";
}
// pyramid pattern
for (let i = 2; i <= n; i++) {
  // printing spaces
  for (let j = n; j > i; j--) {
    string += " ";
  }
  // printing star
  for (let k = 0; k < i * 2 - 1; k++) {
    string += "*";
  }
  string += "\n";
}
console.log(string);
```

▶ **Try It** Run Here

13. Right Pascal Star Pattern

```
*
**
***
****
*****
****
***
**
*
```

The **right pascal star pattern** is created using 2 nested loops.

You can observe in the above pattern it is nothing but the right triangle star pattern and **reversed triangle star pattern** together. Here is the code for the right pascal star pattern.

Example

```
let n = 5;
let string = "";
for (let i = 1; i <= n; i++) {
  for (let j = 0; j < i; j++) {
    string += "*";
  }
  string += "\n";
}
for (let i = 1; i <= n - 1; i++) {
  for (let j = 0; j < n - i; j++) {
    string += "*";
  }
  string += "\n";
}
console.log(string);
```

▶ **Try It** Run Here

14. Left Pascal Star Pattern

```
    *
   **
  ***
```

```
 ****
*****
 ****
  ***
   **
    *
```

The **left pascal star pattern** is also created using 2 nested loops.

It is the same as the right pascal star pattern just mirrored. Here is the code for the right pascal star pattern.

Example

```
let n = 5;
let string = "";
for (let i = 1; i <= n; i++) {
  for (let j = 0; j < n - i; j++) {
    string += " ";
  }
  for (let k = 0; k < i; k++) {
    string += "*";
  }
  string += "\n";
}
for (let i = 1; i <= n - 1; i++) {
  for (let j = 0; j < i; j++) {
    string += " ";
  }
  for (let k = 0; k < n - i; k++) {
    string += "*";
  }
  string += "\n";
}
console.log(string);
```

▶ **Try It** Run Here

15. Heart Star Pattern In JavaScript

```
 ***   ***
***** *****
***********
 *********
```

```
  *******
   *****
    ***
     *
```

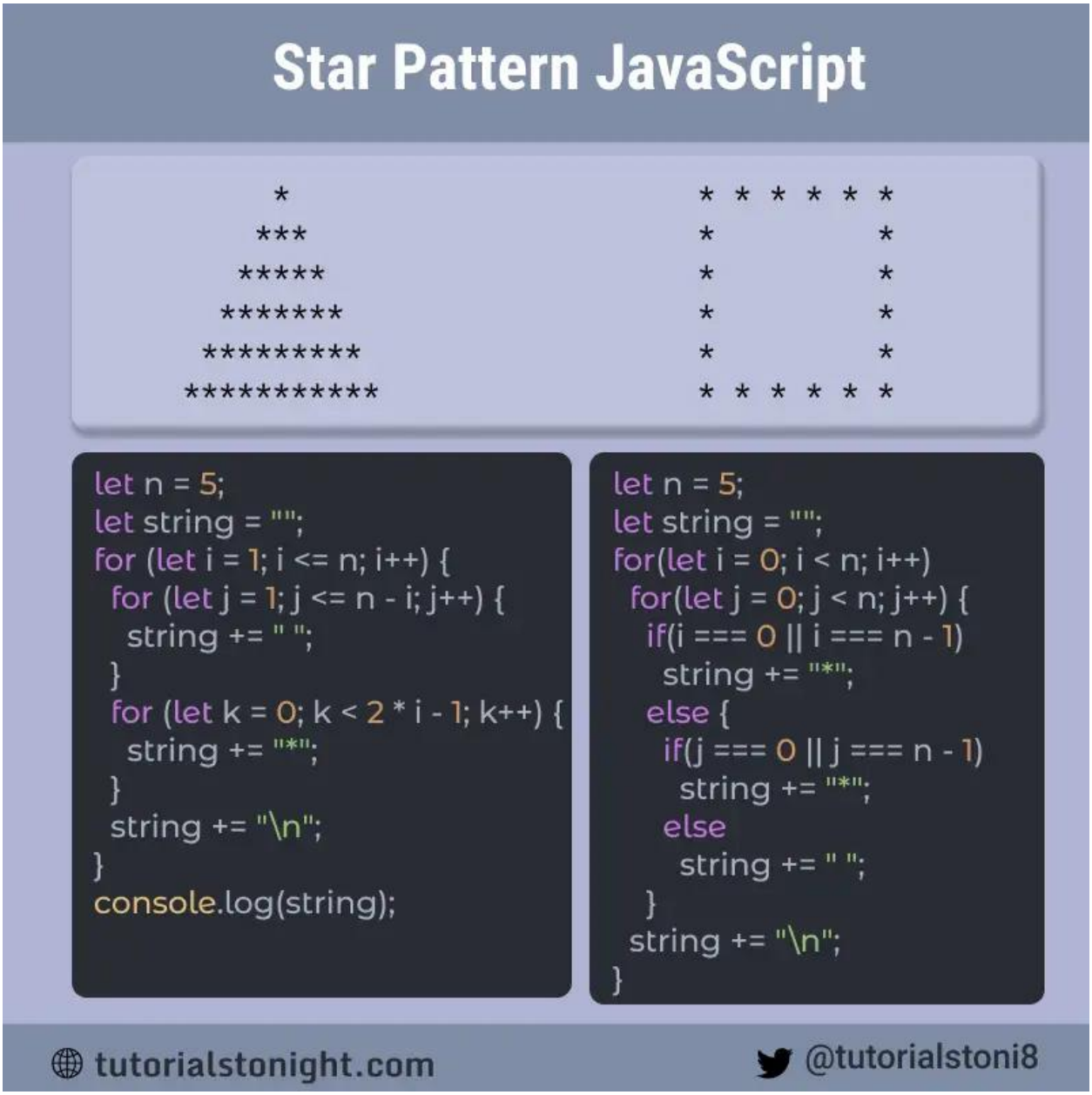
Heart star pattern is a quite complex structure to create. If you observe the pattern shown above then you can see it is made up of many other smaller structures and spaces.

The complete code of the heart star pattern is as follows.

Example

```
var n = 6;
var str = "";
for (let i = n / 2; i < n; i += 2) {
  // print first spaces
  for (let j = 1; j < n - i; j += 2) {
    str += " ";
  }
  // print first stars
  for (let j = 1; j < i + 1; j++) {
    str += "*";
  }
  // print second spaces
  for (let j = 1; j < n - i + 1; j++) {
    str += " ";
  }
  // print second stars
  for (let j = 1; j < i + 1; j++) {
    str += "*";
  }
  str += "\n";
}
// lower part
// inverted pyramid
for (let i = n; i > 0; i--) {
  for (let j = 0; j < n - i; j++) {
    str += " ";
  }
  for (let j = 1; j < i * 2; j++) {
    str += "*";
  }
  str += "\n";
}
console.log(str);
```

▶ **Try It** Run Here



We will use the ASCII value of a letter to loop through the alphabet.

ASCII value is a number that represents the character. For example, the letter A is represented by 65, Z is 90, and so on.

We will also use String.fromCharCode() to convert the number to a letter.

The following example is to loop through the alphabet and print the letter on the screen.

Example

```
for(let i = 65; i <= 90; i++){
  console.log(String.fromCharCode(i));
}
```

Run Here

Print Alphabet Pattern In Javascript

Here we have discussed 12 different alphabet patterns in detail with their programs in javascript.

Alphabet Pattern 1:

```
A
A B
A B C
A B C D
A B C D E
```

To create above pattern run 2 nested `for loop`. Run external loop for 'N' number of times, where 'N' is number of rows in the square, i.e `for(let i = 0;i < N;i++)`.

The internal loop will run for 1 time in the first iteration of external code, 2 times in the second iteration, and so on.

In each the iteration of internal loop add 65 to variable "j" and convert it to a character using `String.fromCharCode()` (65 is ASCII value of 'A').

Example

```
let n = 5; // you can take input from prompt or change the value
```

```
let string = "";
// External loop
for (let i = 1; i <= n; i++) {
  // printing characters
  for (let j = 0; j < i; j++) {
    string += String.fromCharCode(j + 65);
  }
  string += "\n";
}
console.log(string);
```

▶ Try It Run Here

Alphabet Pattern 2:

```
A
B B
C C C
D D D D
E E E E E
```

Pattern2 is similar to pattern1, only difference is in characters. You can see pattern2, characters are changing only in the next iteration of the external loop. So you can simply use 'i - 1' while creating characters instead of 'j' (*'i - 1' because 'i' starts from 1*).

Example

```
let n = 5;
let string = "";
// External loop
for (let i = 1; i <= n; i++) {
  // printing characters
  for (let j = 0; j < i; j++) {
    string += String.fromCharCode((i - 1) + 65);
  }
  string += "\n";
}
console.log(string);
```

▶ Try It Run Here

Alphabet Pattern 3:

```
A
B C
D E F
G H I J
K L M N O
```

This pattern is the same as pattern1 only difference is that the character is changing in each and every iteration. To achieve this you can simply create a random variable ('count' in the below example) and increase it in every iteration and use it to get the character.

Example

```
let n = 5; // you can take input using prompt or change the value
let string = "";
let count = 0;
// External loop
for (let i = 1; i <= n; i++) {
  for (let j = 0; j < i; j++) {
    string += String.fromCharCode(count + 65);
    count++; // increment cause next alphabet
  }
  string += "\n";
}
console.log(string);
```

▶ **Try It** Run Here

Alphabet Pattern 4:

```
ABCDE
ABCD
ABC
AB
A
```

In this pattern just control the internal loop such as it runs for 'N' times in the first iteration of the external loop, 'N - 1' times in the second iteration, and so on. To get this set initialization variable (j) less than `'n - i + 1'`.

Now use the initialization variable of the internal loop for character increment.

Example

```
let n = 5; // you can take input using prompt or change the value
let string = "";
// External loop
for (let i = 1; i <= n; i++) {
  for (let j = 0; j < n - i + 1; j++) {
    string += String.fromCharCode(j + 65);
  }
  string += "\n";
}
console.log(string);
```

▶ Try It Run Here

Alphabet Pattern 5:

```
EDCBA
EDCB
EDC
ED
E
```

This pattern is the same as pattern4 only difference is that instead of starting the character being 'A' it is the character at ASCII value 'N - 1 + 65', where 'N' is the height of the pattern.

To achieve this use `'N - 1 - j'` for character creation and add 65 to it (65 is ASCII value of A).

Example

```
let n = 5; // you can take input using prompt or change the value
let string = "";
// External loop
for (let i = 1; i <= n; i++) {
  for (let j = 0; j < n - i + 1; j++) {
    string += String.fromCharCode((n - 1 - j) + 65);
  }
  string += "\n";
}
```

```
console.log(string);
```

▶ Try It Run Here

Alphabet Pattern 6:

```
EDCBA
DCBA
CBA
BA
A
```

This pattern is the same as pattern4 only difference is that in pattern4 starting character is 'A' but in this pattern ending character is 'A' in each row.

To achieve this use `'N - i - j'` for character creation and add 65 to it (65 is ASCII value of A).

Example

```
let n = 5; // you can take input using prompt or change the value
let string = "";
// External loop
for (let i = 1; i <= n; i++) {
  for (let j = 0; j < n - i + 1; j++) {
    string += String.fromCharCode((n - i - j) + 65);
  }
  string += "\n";
}
console.log(string);
```

▶ Try It Run Here

Alphabet Pattern 7: Pyramid patten

```
    A
   ABC
  ABCDE
 ABCDEFG
ABCDEFGHI
```

This is a pyramid-shaped pattern using the alphabet, we have created the pyramid pattern using stars in the last section. Using the same technique create the pattern and instead of printing stars, print alphabets using `String.fromCharCode`.

Example

```
let n = 5;
let string = "";
// External loop
for (let i = 1; i <= n; i++) {
  // creating spaces
  for (let j = 0; j < n - i; j++) {
    string += " ";
  }
  // creating alphabets
  for (let k = 0; k < 2 * i - 1; k++) {
    string += String.fromCharCode(k + 65);
  }
  string += "\n";
}
console.log(string);
```

▶ **Try It** Run Here

Stay Ahead, Learn More

- Star Pattern Programs in Javascript
- Number Pattern Programs in Javascript

Alphabet Pattern 8: Pyramid pattern

```
    A
   BCD
  EFGHI
 JKLMNOP
QRSTUVWXY
```

This pattern is the same as pattern7 just alphabets are increasing in each and every iteration.

To keep track of this create a variable and increment it in every iteration of the internal loop and use this variable to create alphabets.

Example

```
let n = 5;
let string = "";
let count = 0;
// External loop
for (let i = 1; i <= n; i++) {
  // creating spaces
  for (let j = 0; j < n - i; j++) {
    string += " ";
  }
  // creating alphabets
  for (let k = 0; k < 2 * i - 1; k++) {
    string += String.fromCharCode(count + 65);
    count++;
  }
  string += "\n";
}
console.log(string);
```

▶ **Try It** Run Here

Alphabet Pattern 9: Reverse Pyramid Pattern

```
ABCDEFGHI
 ABCDEFG
  ABCDE
   ABC
    A
```

This is a reverse pyramid pattern using alphabets. Just control the formation of spaces and the creation of stars in reverse order. See the code below to understand.

Compare codes of pyramid and reverse pyramid for better understanding.

Example

```
let n = 5;
```

```
let string = "";
// External loop
for (let i = 1; i <= n; i++) {
  // creating spaces
  for (let j = 1; j < i; j++) {
    string += " ";
  }
  // creating alphabets
  for (let k = 0; k < 2 * (n - i + 1) - 1; k++) {
    string += String.fromCharCode(k + 65);
  }
  string += "\n";
}
console.log(string);
```

▶ **Try It** Run Here

Alphabet Pattern 10: Diamond Pattern

```
    A
   ABC
  ABCDE
 ABCDEFG
ABCDEFGHI
 ABCDEFG
  ABCDE
   ABC
    A
```

The diamond pattern is a combination of the pyramid and reverse pyramid alphabet patterns.

Example

```
let n = 5;
let string = "";
// Pyramid
for (let i = 1; i <= n; i++) {
  for (let j = 1; j < n - i + 1; j++) {
    string += " ";
  }
  for (let k = 0; k < 2 * i - 1; k++) {
    string += String.fromCharCode(k + 65);
  }
```

```
  string += "\n";
}
// Reverse Pyramid
for (let i = 1; i <= n - 1; i++) {
  for (let j = 1; j < i + 1; j++) {
    string += " ";
  }
  for (let k = 0; k < 2 * (n - i) - 1; k++) {
    string += String.fromCharCode(k + 65);
  }
  string += "\n";
}
console.log(string);
```

▶ **Try It** Run Here

Alphabet Pattern 11: Hourglass Pattern

```
ABCDEFGHI
 ABCDEFG
  ABCDE
   ABC
    A
   ABC
  ABCDE
 ABCDEFG
ABCDEFGHI
```

Hourglass pattern is a combination of the reverse pyramid and pyramid alphabet patterns.

Example

```
let n = 5;
let string = "";
// Reverse Pyramid
for (let i = 1; i <= n; i++) {
  for (let j = 1; j < i; j++) {
    string += " ";
  }
  for (let k = 0; k < 2 * (n - i + 1) - 1; k++) {
    string += String.fromCharCode(k + 65);
  }
  string += "\n";
```

```
}
// Pyramid
for (let i = 1; i <= n - 1; i++) {
  for (let j = 1; j < n - i; j++) {
    string += " ";
  }
  for (let k = 0; k < 2 * (i + 1) - 1; k++) {
    string += String.fromCharCode(k + 65);
  }
  string += "\n";
}
console.log(string);
```

▶ **Try It** Run Here

Alphabet Pattern 12: Pascal Pattern

```
A
AB
ABC
ABCD
ABCDE
ABCD
ABC
AB
A
```

Pascal pattern is the same as a diamond pattern just remove spaces and change conditions in internal loops.

Example

```
let n = 5;
let string = "";
// Pyramid
for (let i = 1; i <= n; i++) {
  for (let k = 0; k < i; k++) {
    string += String.fromCharCode(k + 65);
  }
  string += "\n";
}
// Reverse Pyramid
for (let i = 1; i <= n - 1; i++) {
  for (let k = 0; k < n - i; k++) {
```

```
    string += String.fromCharCode(k + 65);
  }
  string += "\n";
}
console.log(string);
```

▶ Try It Run Here

Javascript Array Methods List

Here is a list of javascript array methods. We will go in detail with each method further in the section.

Click on any method to jump to the section.

1. concat()
2. copyWithIn()
3. entries()
4. every()
5. fill()
6. filter()
7. find()
8. findIndex()
9. forEach()
10. Array.from()
11. includes()
12. indexOf()
13. isArray()
14. join()
15. lastIndexOf()
16. map()
17. push()
18. pop()
19. shift()
20. unshift()
21. reduce()
22. reverse()

23. slice()
24. some()
25. sort()
26. splice()
27. toString()
28. values()

Now let's look at each String method in detail.

1. Javascript concat Array

The `concat()` array method in javascript is used to merge 2 or more arrays or values into a single array.

The `concat()` method does not change the original array but returns a new array by merging them.

Syntax:

```
var newArray = arr.concat(arg1, arg2, ...)
```

Here the arguments of the method can be an array and/or any other data type.

Example

```
const numbers = [1, 2, 3];
const characters = ['a', 'b', 'c'];
const booleans = [true, false];

// single argument
console.log(numbers.concat(characters));
// multiple argument
console.log(numbers.concat(characters, booleans));
// passing values
console.log(numbers.concat(characters, 24, "John", {x: 12, y: 13}));
```

▶ Try It Run Here

Alternate method:

Apart from concat() method, you can also spread operator (`...`) to merge multiple arrays and values into a single array.

Here is the above example using the spread operator.

Example

```
const numbers = [1, 2, 3];
const characters = ['a', 'b', 'c'];
const booleans = [true, false];

console.log([...numbers, ...characters]);
// multiple argument
console.log([...numbers, ...characters, ...booleans]);
// passing values
console.log([...numbers, 24, "John"]);
```

Run Here

2. copyWithIn Method

The `copyWithIn()` method in javascript copies a part of the same array within the same calling array.

This method modifies elements of the original array but the length of the array is not modified.

Syntax:

```
arr.copyWithIn(targetIndex);

arr.copyWithIn(targetIndex, startIndex);

arr.copyWithIn(targetIndex, startIndex, endIndex);
```

The `copyWithIn()` method takes three arguments:

- **targetIndex**: It defines the target index from where a copy of the element should starts
- **startIndex** (optional): It defines the index from where the method starts copying elements. The default value is 0.

- **endIndex** (optional): It defines the index at which method stops copying the element. The default value is the last index.

Example

```
//since array is modified we took 3 array with same elements
const arr1 = ['a', 'b', 'c', 'd', 'e', 1, 2, 3];
const arr2 = ['a', 'b', 'c', 'd', 'e', 1, 2, 3];
const arr3 = ['a', 'b', 'c', 'd', 'e', 1, 2, 3];

// copy elements from index 0 to last index to index 2
console.log(arr1.copyWithin(2));
// copy elements from index 4 to last index to index 2
console.log(arr2.copyWithin(2, 4));
// copy elements from index 4 to index 6 to index 2
console.log(arr3.copyWithin(2, 4, 6));
```

▶ **Try It** Run Here

When you use the negative index, the method starts counting from the end of the array. i.e -1 is the last index, -2 is the second last index, and so on.

Here is an example of using a negative index.

Example

```
const arr1 = ['a', 'b', 'c', 'd', 'e', 1, 2, 3];
const arr2 = ['a', 'b', 'c', 'd', 'e', 1, 2, 3];

console.log(arr1.copyWithin(2, -3));
console.log(arr2.copyWithin(-5, -3, -1));
```

Run Here

3. entries Method In Javascript

The `entries()` method in javascript returns a new **Array Iterator** object that contains the key/value pairs for each index in the array.

The `entries()` method does not change the original array but returns a new array iterator object.

This iterator can be used to get the next element in the array by calling the `next()` method on the iterator object.

Syntax:

```
arr.entries()
```

Here the `entries()` method does not take any argument.

Example

```
const arr = ["a", "b", "c", "d", "e"];
const iterator1 = arr.entries();
console.log(iterator1.next());
console.log(iterator1.next().value);
console.log(iterator1.next().value);

// using for...of loop
const iterator2 = arr.entries();
for (const [index, value] of iterator2) {
  console.log(index, value)
}
```

▶ **Try It** Run Here

4. every() Method

The `every()` method in javascript executes a function for each element in the array and returns true if the function returns true for all elements.

The original array is not modified by this method.

Syntax:

```
arr.every(callback(currentValue, index, arr), thisArg)
```

You can define the callback function within the method or you can define it outside the method.

The callback function takes three arguments:

- **currentValue**: It defines the current element in the array.

- **index** (optional): It defines the index of the current element in the array.
- **arr** (optional): It defines the array.

Let's see an example where we are checking if all the elements in the array are even.

Example

```
const arr1 = [1, 2, 3, 4, 5];
const arr2 = [22, 42, 86, 100, 4];

function isEven(num) {
  return num % 2 === 0;
}

console.log(arr1.every(isEven));
console.log(arr2.every(isEven));
```

Run Here

5. fill Method In Javascript

The `fill()` method in javascript returns a modified array by filling a specified index with the specified value.

The original array is not modified by the `fill()` method.

Syntax:

```
arr.fill(value);

arr.fill(value, start);

arr.fill(value, start, end);
```

The `fill()` method accepts 3 arguments:

- **value**: Value to be filled
- **start**(optional): Index from where the filling has to start. Its default value is 0.
- **end**(optional): Index where the filling is to be stopped.

Example

```
const arr = ["a", "a", "a", "a", "a", "a", "a"];

// fill the whole array with "b"
console.log(arr.fill("b"));

// fill the array from index 2 to last with "c"
console.log(arr.fill("c", 2));

// fill the array from index 4 to index 6 with "d"
console.log(arr.fill("d", 4, 6));
```

▶ **Try It** Run Here

This method can be used when you create an array of a specified size and you want to fill the array with a particular value.

Example

```
const arr = new Array(10).fill("a");
console.log(arr);
```

Run Here

6. filter Method In Javascript

The `filter()` method in javascript is used to create a new array with the elements of the same array if elements pass a certain condition.

The `filter()` methods accept a callback function as an argument.

The callback function returns `true` or `false` after checking some conditions over each and every element. If `true` is returned then that element is added to the new array, else discarded.

Finally, a new array is returned with those added elements.

Syntax:

```
arr.filter(callback(currentValue, index, arr), thisArg)
```

The argument `currentValue` is necessary and `index` and `arr` are optional.

Example

```
const arr = [10, 12, 5, 15, 2, 32, 20, -5, 23];

// create a new array with all the elements greater than 10
console.log(arr.filter((element) => element > 10));

// array with only even numbers
console.log(arr.filter((element) => element % 2 === 0));
```

▶ Try It Run Here

7. find Method In Javascript

The `find()` method in javascript is returned the first value of the array elements which satisfies the provided condition.

The `find()` method accepts a callback function where you can define the condition.

The callback function returns `true` or `false` after checking some conditions over each and every element. If `true` is returned then the element is returned, else the next element is checked.

Finally, the first element which satisfies the condition is returned.

Syntax:

```
arr.find(callback(currentValue, index, arr), thisArg)
```

Let's see an example where we are finding the first element which is greater than 10.

Example

```
const arr = [1, 10, 2, 25, 5, 15];

// method return the first element which is greater than 10
console.log(arr.find((element) => element > 10));
```

▶ Try It Run Here

If nothing is found then `undefined` is returned.

The `find()` method does not return the index of the element which satisfies the condition to get index use `findIndex()` method.

8. findIndex Method In Javascript

The `findIndex()` method is the same as `find()` but it returns the index value of the element, not the value itself.

If no such element exists then returns -1.

The `findIndex()` method accepts a callback function where you can define the condition.

Syntax:

```
arr.findIndex(callback(currentValue, index, arr), thisArg)
```

Let's find the index of first element which is greater than 10.

Example

```
const arr = [1, 10, 2, 25, 5, 15];

// method return the index of first element which is greater than 10
console.log(arr.findIndex((element) => element > 10));
```

▶ Try It Run Here

9. forEach Method In Javascript

The `forEach()` method in javascript is used to iterate over each and every element of the array and execute a function for all elements.

It accepts a callback function as an argument.

Syntax:

```
arr.forEach(callback(currentValue, index, arr), thisArg)
```

The callback function accepts three arguments:

- `currentValue` is the current element of the array.
- The `index` is the index of the current element.
- `arr` is the array itself.

In the below example we are looping through the array and printing the elements.

Example

```
const arr = [10, 12, 37, 24, 65];

function print(element, index) {
  console.log(element + " is at index " + index);
}

// using forEach method
arr.forEach(print);
```

▶ **Try It** Run Here

 # Printing sum of the square of all elements of the array

Example

```
const arr = [1, 2, 3, 4, 5];

var sum = 0;
function square(element) {
  sum += element * element;
}

arr.forEach(square);
console.log("sum = " + sum);
```

Run Here

10. Array.from Method In Javascript

The `Array.from()` method in javascript is used to convert an array like object to an array.

What do we mean by an array like object?

An array like object is an object which has a length property. Example: string, arguments, NodeList, HTMLCollection, etc.

To convert these objects to a proper array pass the object as an argument to the `Array.from()` method.

Syntax:

```
Array.from(arrayLike)
```

The `Array.from()` method returns a new array and does not change the original array.

Example

```
const alphabets = "abcdefghijklmnopqrstuvwxyz";

// converting string to array
const arr = Array.from(alphabets);
console.log(arr);

const obj = {
  0: "a",
  1: "b",
  2: "c",
  length: 3
};
// converting object to array
const arr2 = Array.from(obj);
console.log(arr2);
```

Run Here

11. includes Method In Javascript

The `includes()` method in javascript determines whether certain values exist in the array or not. If the value exists in the array then the method returns `true` else return `false`.

Syntax:

```
arr.includes(searchElement, fromIndex)
```

The function accepts 2 arguments:

- `searchElement` is the value to be searched.
- `fromIndex` is the index from which the search starts.

Example

```
const fruit = ["mango", "banana", "apple", "orange", "watermelon"]
console.log(fruit.includes("apple")); // true

const alphabets = ["a", "b", "c", "d", "e"];
console.log(alphabets.includes("a")); // true
console.log(alphabets.includes("a", 1)); // false
```

▶ **Try It** Run Here

12. indexOf Method In Javascript

The `indexOf()` method in javascript returns the first index of the element passed as an argument. If the value does not exist then returns -1.

Syntax:

```
arr.indexOf(searchElement, fromIndex)
```

The function accepts 2 arguments:

- `searchElement` is the value to be searched.
- `fromIndex` is the index from which the search starts.

Example

```
const arr = ["a", "b", "c", "d", "e"];

// get the index of "c"
console.log(arr.indexOf("c"));
```

▶ **Try It** Run Here

What is difference between `indexOf()` and `findIndex()`?

The `indexOf()` method accepts a direct value as an argument whereas the `findIndex()` method accepts a callback function as an argument and gives more flexibility to the user.

13. isArray Method In Javascript

The `isArray()` method in javascript returns `true` if the passed argument is an array else returns `false`.

It is used to check whether the passed argument is an array or not.

Syntax:

```
Array.isArray(arr)
```

Example

```
const arr = ["a", "b", "c", "d", "e"];

// check if array
console.log(Array.isArray(arr));

console.log(Array.isArray("abc")); // false
```

Run Here

14. join Method In Javascript

The `join()` method in javascript joins all elements of the array as a string separated by commas or by some specified separator and returns it as a string.

The function accepts 1 argument which is a separator. It is optional and its default value is a comma (",").

Syntax:

```
arr.join(separator)
```

This method is used to join the elements of the array as a string.

Example

```
const fruit = ["mango", "banana", "apple", "orange", "watermelon"];

// default separator (,)
console.log(fruit.join());
// blank separator
console.log(fruit.join(''));
// custom separator
console.log(fruit.join('-'));
```

▶ **Try It** Run Here

15. lastIndexOf Method In Javascript

The `lastIndexOf()` method returns the last index of the element passed as an argument in the method. If the value does not exist then returns -1.

Syntax:

```
arr.lastIndexOf(searchElement, fromIndex)
```

The method accepts 2 arguments:

- `searchElement` is the value to be searched.
- `fromIndex` (optional) is the index from which the search starts.

Example

```
const arr = ["b", "d", "i", "b", "d", "f", "i", "b", "d", "g", "i"];

// get the index of 'd'
console.log(arr.lastIndexOf('d'));
// get the index of 'g'
console.log(arr.lastIndexOf('g'));
```

Run Here

16. map Method In Javascript

The `map()` method in javascript is one of the **most useful array method**, it returns a new array by filling the array with the results of the callback function running for each element of the array.

You have to pass a callback function as an argument to the `map()` method. The callback function is called for each element of the array and the return value of the callback function is stored in the new array.

Syntax:

```
arr.map(callback(currentValue, index, array), thisArg)
```

The callback function in the map method accepts 3 arguments:

- `currentValue` is the current element of the array.
- `index` (optional) is the index of the current element of the array.
- `array` (optional) is the array that is being traversed.

Example

```
const num = [1, 2, 3, 4, 5];

// multiply each element by its index
const newArray = num.map((element, index) => element * index);
console.log(newArray);
```

▶ **Try It** Run Here

17. push Method In Javascript

The `push()` method in javascript is used to add one or more elements to the end of the calling array.

After adding the elements to the array, the method returns the new length of the array.

Syntax:

```
arr.push(value1, value2, ...)
```

This method accepts 1 or more arguments.

Example

```
const fruit = ["mango", "banana"];

// add elements to the end of the array
fruit.push("apple");
console.log(fruit);
// add 2 new elements to the end
fruit.push("orange", "watermelon");
console.log(fruit);
```

▶ Try It Run Here

18. pop Method In Javascript

The `pop()` method in javascript removes the last element from the array and returns the removed elements.

If the array is empty, the method returns undefined.

Syntax:

```
arr.pop()
```

Here is an example of the `pop()` method:

Example

```
const fruit = ["mango", "banana", "apple", "orange", "watermelon"];
fruit.pop();
console.log("After 1 pop => " + fruit);
fruit.pop();
console.log("After 2 pop => " + fruit);
```

▶ Try It Run Here

19. shift Method In Javascript

Just like the `pop()` method `shift()` method also removes the element from the array but instead of removing the last element, it removes the first element.

If the array is empty, the method returns undefined.

Syntax:

```
arr.shift()
```

Here is an example of the `shift()` method:

Example

```
const fruit = ["mango", "banana", "apple", "orange", "watermelon"];
fruit.shift();
console.log("After 1 shift => " + fruit);
fruit.shift();
console.log("After 2 shift => " + fruit);
```

▶ Try It Run Here

20. unshift Method In Javascript

Like `push()` method `unshift()` method also adds the element to the array but instead of adding it to the ending of the array, it adds it to the beginning.

This method returns the new length of the array.

Syntax:

```
arr.unshift(value1, value2, ...)
```

This method accepts 1 or more arguments.

Example

```
const fruit = ["orange", "watermelon"];

// add elements to the beginning of the array
fruit.unshift("apple");
console.log("After 1st unshift => " + fruit);
// add 2 new elements to the beginning
```

```
fruit.shift("mango", "banana");
console.log("After 2nd unshift => " + fruit);
```

▶ Try It Run Here

21. reduce Method In Javascript

The `reduce()` method in javascript reduce the array element to a single value, using a reducer function.

It is a little bit tricky to understand the `reduce()` method. Let's see how it works.

Syntax:

```
arr.reduce(callback(accumulator, currentValue,
currentIndex, array))
```

The reducer callback function accepts the following arguments:

- **accumulator**: It stores return value of previously returned in last execution or initial value in the first iteration. Default initial value is 0.
- **currentValue**: It is the current element being executed.
- **index**: index value of the current element. It is optional.

The callback function seems to be executed for each element but it starts execution from the second element. The first element is the initial value.

At the start, the accumulator becomes the value at index 0 and the current value is the value at index 1.

In the second iteration, the accumulator becomes the value returned by the callback function in the previous iteration and the current value is the value at index 2.

Example

```
const num = [1, 2, 3, 4, 5];

// sum of all elements
const sum = num.reduce((acc, curr) => acc + curr);
```

```
console.log(sum);
```

▶ Try It Run Here

Here is another example to find the biggest number in the array:

Example

```
const num = [24, 53, 78, 91, 12];

// find biggest number
const max = num.reduce((acc, curr) => Math.max(acc, curr));
console.log(max);
```

Run Here

22. reverse Method In Javascript

The `reverse()` method in javascript reverse the array elements from first to last.

```
const num = [1, 2, 3, 4, 5];
console.log(num.reverse());
```

▶ Try It Run Here

23. slice Method In Javascript

The `slice()` method in javascript returns a new array by copying the calling array's elements in new the array.

The `slice()` method accepts 2 arguments:

- **start**: It is starting position to copy the element. It is optional, its default value is 0.
- **end**: It is the ending position to copy the element. It is optional, its default value is the last index of the array.

```
const num1 = [1, 2, 3, 4, 5];
const num2 = num1.slice();
const num3 = num1.slice(2);
```

```
const num4 = num1.slice(2,4);
console.log(num2);
console.log(num3);
console.log(num4);
```

▶ Try It Run Here

24. some Method In Javascript

The `some()` method in javascript determines whether at least one element in the array passes certain conditions by providing a callback function.

If any of array element passes the test then it returns `true`, else returns `false`.

The callback function's first argument is the current element and the second element is the index of the element which is also optional.

```
const num = [1, 2, 3, 4, 5];
console.log(num.some((element) => element > 3));
```

▶ Try It Run Here

25. sort Method In Javascript

The `sort()` method in javascript sort the array element in ascending order according to the character's Unicode value. Means it convert its element to string and sort it by comparing the string in UTF-16 code.

The `sort()` function accepts a compare function which defines some sorting order. This compare function has 2 arguments, 'firstElement': the first element for comparison and 'secondElement': these second element for comparison.

```
const fruit = ["mango", "banana", "apple", "orange", "watermelon"];
console.log(fruit.sort());
const num = [2, 321, 100, 1310, 43];
console.log(num.sort());
console.log(num.sort((firstElement, secondElement)=>{
```

```
  return firstElement - secondElement;
}));
```

▶ Try It Run Here

26. splice Method In Javascript

The `splice()` method in javascript changes the array element by replacing or removing the array element from the array. It can also add elements to the desired index in the array either by keeping or removing the rest elements of the array.

```
arr.splice(index, deleteCount, replaceBy)
```

```
const day = ["monday", "wednesday", "friday", "sunday"];
day.splice(1, 0, "tuesday");
console.log(day);
day.splice(4, 1, "saturday");
console.log(day);
day.splice(3, 2);
console.log(day);
```

▶ Try It Run Here

27. toString Method In Javascript

The `toString()` method in javascript returns a string representing the elements of the array.

```
const arr = [1, "star", 34, true, 23];
console.log(arr.toString());
```

▶ Try It Run Here

28. values Method In Javascript

The `values()` method in javascript returns a new array iterator that contains values of each index in the array.

```
const array1 = [1, "star", 34, true, 23];
const iterator = array1.values();
for (const value of iterator) {
  console.log(value);
}
```

▶ Try It Run Here

Javascript String Methods List

JavaScript has a large number of string methods all for different purposes. We have listed some of the most frequently use string methods for you.

Here is the list of string methods with examples. Click on any method to jump to the section.

1. charAt()
2. charCodeAt()
3. concat()
4. endsWith()
5. includes()
6. indexOf()
7. lastIndexOf()
8. match()
9. matchAll()
10. repeat()
11. replace()
12. replaceAll()
13. search()
14. slice()
15. split()
16. startsWith()
17. substr()
18. substring()
19. toLowerCase()
20. toUpperCase()
21. toString()
22. trim()

23.valueOf()

let's now look at each String methods in detail.

1. charAt in javascript

The charAt() string method in javascript returns a character at the specific index of the given string.

The `charAt()` method takes the index value as an argument and returns the corresponding character from the calling string.

An index value of a string starts with 0, which means the first character has an index value of 0, the second character has an index value of 1, and so on.

Example

```
const str = "Hello World";

// using charAt() method
console.log(str.charAt(1));
console.log(str.charAt(6));
```

▶ **Try It** Run Here

The default index value in the charAt method is 0.

If the index value is out of range then it returns an empty string ('').

If the index can not be converted to an integer then this method **returns the first character** of the string.

Example

```
const str = "Hello World";
//default index value = 0, return first character
console.log(str.charAt());

// index out of range, return empty string
console.log(str.charAt(13));
```

```
// index can't be converted to an integer, return first character
console.log(str.charAt('a'));
```
Run Here

Altenative way to charAt() method

You can consider the string as an array of characters and call out any character of the string by its index value, as all the characters are a member of the array.

Example
```
const str = "Hello World";

// using square bracket notation
console.log(str[0]);
console.log(str[1]);
console.log(str[13]); // index out of range, return empty string
```
Run Here

2. charCodeAt in javascript

The charCodeAt string method in javascript returns the Unicode value (between `0` and `65535`) of the character present at the given index of the string. Example Unicode value of 'A' is 65.

The `charCodeAt()` method takes an `index` as an argument and returns the Unicode value of the character present at that index value.

Example
```
const str = "Hello World";

// using charCodeAt() method
console.log(str.charCodeAt(0));
console.log(str.charCodeAt(1));
```
▶ **Try It** Run Here

The default value of the index is 0. If the index value is not given or it can't be converted to a string the 0 is used by default.

If the index value is out of range then this method returns `NaN`.

Example

```
const str = "Hello World";

//default index value = 0
console.log(str.charCodeAt());
console.log(str.charCodeAt("x"));
// Out of range returns NaN
console.log(str.charCodeAt(13));
```

Run Here

3. concat in javascript

The concat string method in javascript concatenates the passed string in the method to the calling string and returns the concatenated string as a new string.

The `concat()` method can take any number of strings as an argument.

If the passed argument is not a string then it converts the argument to string and then concatenates it.

Example

```
const firstName = "Hello";
const lastname = "World";

// using concat method
let result1 = firstName.concat(" ", lastname);
console.log(result1);
// multiple arguments
let result2 = "Learning".concat(" ", "to", " ", "code.");
console.log(result2);
```

▶ **Try It** Run Here

Note: for concatenation of string it is suggested to use assignment operators (`+` or `+=`) instead of the `concat` method.

Example

```
const firstName = "Hello";
```

```
const lastname = "World"
console.log(firstName + " " + lastname);
console.log("Learning" + " " +  "to" + " " + "code.");
```
Run Here

4. endsWith in javascript

The endsWith string method in javascript is used to determines whether the string ends with a specified substring or not.

If it ends with a specified string then it returns `true`, else returns `false`.

The substring to be checked for the ending is passed as the first argument in the method.

Example

```
const question = "What is DOM?";

// using endsWith method
// checks whether the string ends with "?"
console.log(question.endsWith("?"));
```
▶ Try It Run Here

There is also a second argument in the method which is optional. It specifies a length value for the string. If you pass the length value is 5 then it will see only the first 5 characters of the string.

Please note that `endsWith()` method is **case sensitive**.

Example

```
const question = "What is DOM?";

// check only first 5 characters
console.log(question.endsWith("?", 5));
```
Run Here

5. String includes in javascript

The includes string method in javascript is used determines whether a given substring is present in the calling string or not.

If the string is present then the method returns `true`, if not substring is not present then the method returns `false`.

The matching of `includes()` method for the string is case-sensitive.

Example

```
const sentence = "Carbon emission is increasing Day by day.";

// check if the string contains words
console.log(sentence.includes("day")); // true
console.log(sentence.includes("Day")); // true
console.log(sentence.includes("DAY")); // false
```

▶ Try It Run Here

The includes also accept an optional second argument which determines the position from where the method starts searching within the string

The default value of the second argument is 0.

Example

```
const sentence = "Carbon emission is increasing day by day.";

// checking from the index 20
console.log(sentence.includes("is", 20));
// checking from the index 2
console.log(sentence.includes("is", 2));
// checking from the index 5
console.log(sentence.includes("emission", 5));
```

Run Here

6. indexOf in JavaScript

The indexOf string method in javascript is used to get the index of the first occurrence of a specified value in a string.

If the character or substring is not present then it returns -1. The search string is case-sensitive.

Example

```
const sentence = "Carbon emission is increasing day by day";

// get 1st index of 'day'
console.log(sentence.indexOf("day"));
console.log(sentence.indexOf("Carbon"));
// case-sensitive, returns -1
console.log(sentence.indexOf("carbon"));
```

▶ Try It Run Here

If no search string is provided to the method then default search string is 'undefined'. Example `'undefined'.indexOf()` will return 0.

There is a second argument that is optional that defines the start index from where the method starts to search for the character or substring.

Example

```
const sentence = "Carbon emission is increasing day by day";

// no substring provided to search
console.log(sentence.indexOf());
console.log('undefined'.indexOf());
// start searching from index 40
console.log(sentence.indexOf("day", 40));
```

Run Here

7. String lastIndexOf JavaScript

The `lastIndexOf()` string method in javascript searches the last occurrence of a substring within a string. The method starts searching from the end of the string for efficiency.

If the substring is not in the given string then it returns -1.

Example

```
const sentence = "to do or not to do";

// finding index of last "to"
console.log(sentence.lastIndexOf("to"));
```

▶ Try It Run Here

If nothing is passed in the method then the default search string is 'undefined'.

Example `'undefined'.lastIndexOf()` will return 0, because there is no argument and now the lastIndexOf method will search for `undefined` which is at index 0.

Example

```
const sentence = "to do or not to do";

// nothing to search, so will search for 'undefined'
console.log(sentence.lastIndexOf());
console.log('undefined'.lastIndexOf());
```

Run Here

There is an optional second argument that is fromIndex. It defines the index from where the method starts to search for the substring from the end of the string.

Its default value is +Infinity. If value of fromIndex is greater than the length of the string then whole string is searched.

Example

```
const sentence = "to do or not to do";

// searching from index 10
console.log(sentence.lastIndexOf("do", 10));
console.log(sentence.lastIndexOf("do", -2));
```

▶ Try It Run Here

8. match in javascript

The match() string method in javascript uses a regular expression to match a series of characters within the calling string.

The method returns the output as an array of string with matched character or strings as its element.

If the parameter passed is not a regular expression then it is implicitly converted to **RegExp** by using `new RegExp(regexp)`.

Example

```
const series = "bdWg2AdjgH4du5jUgT";

// match all capital letters and numbers
console.log(series.match(/[A-Z0-9]/g));
```

▶ **Try It** Run Here

If the used regular expression does not have a **g** flag then the method returns only the first complete match and its capturing group.

If nothing matches within the string then the method returns `null`.

Example

```
const string = "to do or not to do";

// no g flag
console.log(string.match(/do/));
// no match found, return null
console.log(string.match(/dog/));
```

Run Here

9. matchAll in javascript

The matchAll() string method in javascript is an improved variant of the match() method.

In the match method when the regular expression is used without the **g** flag then it works fine but with the **g** flag, it only returns the matches and not its capturing group.

But the matchAll() method finds all the matching string with its capturing group against the regular expression.

Instead of returning an array the matchAll() method returns an iterator to all results matching the string. You will have to explicitly convert it to array using sperad operator (`[...]`) or using Array.from() method.

Note: It is compulsory to use `g` flag with the matchAll() method.

Example

```
const series = "to do or not to do";

// matching "do" and capturing group
const array = [...series.matchAll(/d(o)/g)];
console.log(array);
console.log(array[0])
```

▶ **Try It** Run Here

10. repeat in javascript

The repeat() method concatenates a passed string by a specified number of times and return it as a new string.

A number of times string is to be repeated is passed as an argument in the method, where the number lies between `0` and `+Infinity`.

If a decimal value is passed then it will be converted to an Integer when passed.

The method reports an error for a negative number.

Example

```
const str = "Tick tock, ";

// repeat the string by 2 times
console.log(str.repeat(2));
// converts the decimal value to integer
console.log(str.repeat(3.5));
// repeat 0 times
console.log(str.repeat(0));
```

▶ Try It Run Here

11. replace in javascript

The replace method selects one or all matches from a string and replace it with a `replacement string` and return it as new string.

To find the match method use `string` or `regular expression`. When a string is passed as an argument then it select only the first match while when a regular expression is passed then it selects all the matches.

Example

```
const str = "Carbon emission is increasing day by day.";

// using string to match
console.log(str.replace("day", "year"));
// using regular expression to match
console.log(str.replace(/day/g, "year"));
```

▶ Try It Run Here

Further, the replacement can be a string to be replaced in thr string or a function to be called for each match.

Here is an example that use function in the replace method to be called for each match.

Example

```
const str = "Carbon emission is increasing day by day.";

// using string to match
str.replace("day", foundMatch);
// using regular expression to match
str.replace(/day/g, foundMatch);

function foundMatch(match, index, string) {
  console.log(`match = ${match}, index = ${index}`);
}
```

Run Here

12. replaceAll in javascript

The replaceAll method returns a new string after replacing all occurrences of matched pattern with a replacement string.

Unlike replace() method it replaces all the occurrences whether the given pattern is a string or a regular expression.

The replacement can be a string to be replaced or a function to be called for each match.

Note: While using regular expression in the replaceAll() method using a global flag (**g**) is compulsory with it otherwise it will throw a 'TypeError'.

Example

```
const str = "Carbon emission is increasing day by day.";

// select all match using both string or regular expression
console.log(str.replaceAll("day", "year"));
console.log(str.replaceAll(/day/g, "month"));
```

▶ Try It Run Here

13. search in javascript

The search string method in javascript is used to determine whether a pattern exists within the calling string or not, if it exists then the method returns the index value of the first match within the string.

The search method uses regex to search for a pattern in a string, if a string is passed to search for then the method implicitly convert it to regex.

Example

```
const str = "kjhBfdbAjdbj";
console.log(str.search(/[A-Z]/g));
```

▶ Try It Run Here

14. slice in javascript

The slice string method in javascript extracts a part of the string and returns it as a new string.

```
str.slice( startIndex, [, endIndex])
```

The `slice()` takes 2 arguments, the first argument is the start index for the slicing string and the second argument is the end of the slicing string, where the second argument is optional.

The default value of `endIndex` is `str.length` When the second argument is not passed then the string is sliced from 'startIndex' to the end of the string.

The `slice()` method also accepts negative value, where -1 represents the last index

Example

```
const sentence = "Carbon emission is increasing day by day";
console.log(sentence.slice(5, 15));
console.log(sentence.slice(5));
console.log(sentence.slice(-15, -5));
```

▶ Try It Run Here

15. split in javascript

The split string method in javascript divides the given string intro substring and returns an array of substrings.

The method takes an argument which is a pattern to be used for dividing the string.

```
str.split(pattern, [, limit])
```

- If the pattern is an empty string (`''`) then the method split the string at each character
- If the pattern is a string (`' '`) then the method split the string at each space
- If the pattern can be a regular expression. `'\n'` splits the string at each new line

The *limit* defines the maximum number of substrings to be returned. If it is 0 then an empty array ([]) is returned.

Example

```
const sentence = "Carbon emission is increasing day by day";

// no pattern -> return whole string in array
console.log(sentence.split());
// split at each space
console.log(sentence.split(''));
// split at each space
console.log(sentence.split(' '));
// split at each 'is'
console.log(sentence.split('is'));
```

▶ **Try It** Run Here

16. startsWith in javascript

The startsWith string method in javascript determines whether a string starts with some given substring or not. If it starts with the desired string then it returns `true` else return `false`.

The search string is passed as the first argument to the method. There is also an optional argument that defines the position from where the method should start checking.

The startsWith method is case-sensitive.

Example

```
const sentence = "Carbon emission is increasing day by day";
console.log(sentence.startsWith("Car"));
```

```
console.log(sentence.startsWith("carbon")); // return false case-sensitive
console.log(sentence.startsWith("bon", 3));
```

▶ Try It Run Here

17. substr in javascript

The substr() string method in javascript is used to extract a substring from a string. It returns a part of the string, starting at a specific index and ending after a given number of characters.

```
str.substr(startIndex, [, length])
```

- `startIndex` - It specifies the index value from where the substring starts
- `length` - It defines number of characters to be extracted

If the value of `startIndex` is negative then the index is counted from the end of the string in opposite direction. If it is `NaN` then it is treated as 0.

Example

```
const sentence = "Carbon emission is increasing day by day";

// start at index 10, cut 6 characters
console.log(sentence.substr(10, 6));
// start at index 10, cut all characters
console.log(sentence.substr(10));

// negative index
console.log(sentence.substr(-4, 3));
```

▶ Try It Run Here

18. substring in javascript

The substring() method extracts a part of string between 2 given index values. It returns a part of the string, starting at a specific index and ending after a given number of characters.

```
str.substring(startIndex, endIndex)
```

- `startIndex` - It specifies the index value from where the substring starts
- `endIndex` - It specifies the index value from where the substring ends

If `endIndex` is not specified then it is treated as the last character of the string.

If the value of `startIndex` is greater than the value of `endIndex` then value of these two variables is swapped.

Example

```
const sentence = "Carbon emission is increasing day by day";

// extracting string
// start at index 10, cut all characters
console.log(sentence.substring(10));
// start at index 10, and end at index 20
console.log(sentence.substring(10, 20));

// startIndex > endIndex
console.log(sentence.substring(20, 10));
```

▶ **Try It** Run Here

19. toLowerCase in javascript

The `toLowerCase()` string method in javascript converts the case of a string to lowercase and returns it as a new string.

Example

```
const sentence = "CARBON emission IS INCREASING DAY BY DAY";
console.log(sentence.toLowerCase());
```

▶ **Try It** Run Here

20. toUpperCase in javascript

The toUpperCase string method in javascript returns a new string by converting the calling string to uppercase.

Example

```
const sentence = "carbon emission is increasing day by day";
console.log(sentence.toUpperCase());
```

▶ Try It Run Here

21. toString in javascript

The `toString()` string method in javascript returns a string representing the specified object.

This method also convert numbers to strings in a different number system. For example you can convert a number to a string in binary system by using `toString(2)`, octal system by using `toString(8)` and hexadecimal system by using `toString(16)`, etc.

Example

```
const str = new String("hello World!");
console.log(str.toString(str));
const num = 20;
console.log(num.toString(2));
```

▶ Try It Run Here

22. trim in javascript

The `trim()` string method in javascript removes whitespaces from both ends of the string. Whitespaces are space, tabs, newline, etc.

Example

```
const str = "    TutorialsTonight    ";
console.log(str.trim());
```

▶ Try It Run Here

23. valueOf in javascript

The `valueOf()` string method in javascript returns the primitive value of a **String object**.

Example

```
const str = new String("hello world");
console.log(str.valueOf(str));
```

▶ **Try It** Run Here