Loops

Loops in JavaScript are control structures that allow you to repeat a block of code multiple times. They are essential for performing repetitive tasks, such as iterating over arrays, processing data, or implementing certain algorithms.

- **for loop**: A **for** loop is used to execute a block of code a number of times. It consists of three optional expressions enclosed in parentheses, separated by semicolons:

  for (initialization; condition; increment/decrement) {

  // code to be executed

  }

- **Initialization**: Executes once at the beginning of the loop.
- **Condition**: Evaluated for each iteration. If true, the loop continues; if false, the loop terminates.
- **Updation**: Executed after each iteration. Typically used to update the loop counter.

```
// Print numbers from 1 to 5
for (let i = 1; i <= 5; i++) {
  console.log(i);
}
```

- **while loop**: A **while** loop repeats a block of code while a specified condition is true. It has the following syntax:

  while (condition) { // code to be executed }

- The condition is evaluated before each iteration. If it returns true, the loop continues; otherwise, it stops.

```
// Print numbers from 1 to 5 using a while loop
let i = 1;
while (i <= 5) {
  console.log(i);
  i++;
}
```

□ **do...while loop**: Similar to the **while** loop, but it always executes its block of code at least once, even if the condition evaluates to false. It has the following syntax:

```
do { // code to be executed } while (condition);
```

- The block of code is executed first, then the condition is evaluated. If true, the loop continues; if false, it stops.

```
// Print numbers from 1 to 5 using a do...while loop
let j = 1;
do {
  console.log(j);
  j++;
} while (j <= 5);
```

□ **for...in loop**: Used to iterate over the properties of an object. It iterates over enumerable properties of an object, in an arbitrary order.
Syntax:

```
for (ref in strname){
  console.log(ref);//indexes
}
```

for (variable in object) { // code to be executed }

```
// Iterate over the properties of an object
const person = {
  name: "John",
  age: 30,
  gender: "male",
};
for (let prop in person) {
  console.log(prop + ": " + person[prop]);
}
```

1) **Iterates over Properties**:

- The for...in loop iterates over all enumerable properties of an object.

2) **Order Not Guaranteed**:

- The order of iteration is not guaranteed. It's generally the order in which properties were defined, but this can vary.

3) **Use with Objects**:

- Typically used for objects, not arrays, because it iterates over property names (keys) rather than values.

- **for...of loop**: Introduced in ES6, it iterates over iterable objects such as arrays, strings, maps, sets, etc.
  Syntax:

```
for (ref of strname){
  console.log(ref);//values
}
```

for (variable of iterable) { // code to be executed }

- It provides a more concise syntax compared to the traditional **for** loop for iterating over arrays and other iterable objects.

```
// Iterate over elements of an array
const numbers = [1, 2, 3, 4, 5];
for (let num of numbers) {
  console.log(num);
}
```

1) **Iterates over Values**:

- The for...of loop iterates over the values of an iterable object.
- This loop does not work with objects unless they implement the iterable protocol.

2) **Use with Arrays and Other Iterables**:

- Commonly used with arrays, strings, maps, sets, and other iterable objects.

- **Nested loops**: You can nest loops inside one another to perform more complex iterations. For example, you can use a **for** loop inside another **for** loop to iterate over a two-dimensional array.

How continue and break behaves in loops

continue and break are used to control the flow of loops, such as for, while, and do...while loops. They are not used directly within conditional statements like if, else, or switch. However, they can be used inside loops that contain conditional statements to influence the loop's behavior based on certain conditions.

continue Statement

The continue statement is used to skip the current iteration of a loop and move on to the next iteration. When the continue statement is encountered, the loop's current iteration is terminated, and control is passed to the next iteration of the loop.

```
for (let i = 0; i < 10; i++) {
  if (i % 2 === 0) {
    continue; // Skip even numbers
  }
  console.log(i); // This will only log odd numbers
}
```

In this example, the continue statement skips the even numbers, so the console.log(i) statement only logs the odd numbers from 1 to 9.

break Statement

The break statement is used to terminate the entire loop immediately. When the break statement is encountered, the loop is exited, and control is passed to the statement following the loop.

```
for (let i = 0; i < 10; i++) {
  if (i === 5) {
    break; // Exit the loop when i is 5
  }
  console.log(i); // This will log numbers 0 to 4
}
```

In this example, the break statement causes the loop to terminate when i is equal to 5, so console.log(i) only logs the numbers from 0 to 4.