1. What is the difference between a generator and a function?

ANS: Both functions and generators are used in programming to define reusable code blocks, however they have various uses and unique properties:

Use:
A function is a segment of code that, when called, carries out a certain task.
After running its code block, it returns a value (if any).
Most programming languages have functions that run their whole code all at once and return the output to the caller.
They can return a single value (or none) in response to inputs passed in as input parameters.
Usually, functions continue to execute until they encounter a return statement or the function block's end.
Producer:

A generator is a unique kind of function that has the ability to halt its execution and return partial results to the caller.
It generates a series of values lazily, which means that instead of producing all of the values at once, it produces them one at a time as needed.
Yield is a keyword that generators use to return a value while continuing to execute.
They are helpful in producing lengthy value sequences without using up too much memory because they only compute values when needed.
Generators are usually employed in situations where memory efficiency is critical or where you need to iterate over a potentially infinite sequence.

2. What is the syntax of a generator?

ANS: In JavaScript, you can create generator functions using the function* syntax along with the yield keyword. Here's the basic syntax:

```
function* myGenerator() {
  // Generator logic
  yield value1;
  yield value2;
  // Additional yields as needed
}
```

With this syntax:

A generator function can be defined using the function* keyword.
The yield keyword pauses the generator's execution in order to return a value.

One or more yield statements, each of which yields a value to the caller, may be present in the generator function.

Upon using the generator function, an iterator object is returned, which can subsequently be iterated over via the next() method.

3. Are function generators iterable in JavaScript?

ANS: It is possible to make JavaScript function generators iterable. A generator function that is created with the function* syntax returns an iterable iterator object by default. You can use this iterator object in structures like for...of loops, spread syntax, and other iterable actions because it complies with JavaScript's Iterable Protocol.
In other words, function generators in JavaScript are compatible with different iteration constructions since they may be iterated over using the iterable protocol.

4. Create a generator for multiplying?

ANS:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Multiplication Generator</title>
</head>
<body>
  <script>
    function* multiplicationGenerator(start, end) {
      let result = 1;
      for (let i = start; i <= end; i++) {
        result *= i;
        yield result;
      }
    }

    // Using the generator
    const iterator = multiplicationGenerator(1, 5);
```

```
    let iterationResult = iterator.next();

    while (!iterationResult.done) {
      console.log(iterationResult.value);
      iterationResult = iterator.next();
    }
  </script>
</body>
</html>
```

OUTPUT

```
1
2
6
24
120
```

5. Print an infinite series of natural numbers using a generator.

ANS:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Infinite Series of Natural Numbers</title>
</head>
<body>
  <script>
    function* naturalNumbersGenerator() {
      let number = 1;
      while (true) {
        yield number;
```

```
            number++;
        }
    }

    // Using the generator
    const iterator = naturalNumbersGenerator();
    for (let i = 0; i < 10; i++) { // Print the first 10 numbers for demonstration
        console.log(iterator.next().value);
    }
  </script>
</body>
</html>
```

OUTPUT

1

2

3

4

5

6

7

8

9

10

6. Create a generator that can throw an exception.

ANS:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Generator with Exception</title>
</head>
<body>
  <script>
    function* generatorWithException() {
      try {
        yield 'First value';
        yield 'Second value';
        throw new Error('Exception thrown from generator');
        yield 'Third value'; // This will not be reached
      } catch (error) {
        yield `Caught exception: ${error.message}`;
      }
    }

    // Using the generator
    const iterator = generatorWithException();
    console.log(iterator.next().value); // Output: 'First value'
    console.log(iterator.next().value); // Output: 'Second value'
    console.log(iterator.next().value); // Output: 'Caught exception: Exception thrown from generator'
    console.log(iterator.next().value); // Output: undefined (generator has finished)
  </script>
</body>
</html>
```