***Javascript Assignment 18***

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

*In JavaScript, a generator function is a special type of function that can be paused and resumed, allowing you to write asynchronous code that looks synchronous. Here are some key differences between a generator function and a regular function:*

*Execution control: With a regular function, control flows from the beginning of the function to the end, and the function returns a value. With a generator function, control can be paused and resumed at any point during the function's execution using the yield keyword.*

*Multiple returns: A regular function can only return a value once, whereas a generator function can return multiple values using the yield keyword. When a yield statement is encountered, the function pauses and returns the value specified in the yield statement. The next time the function is called, it resumes execution where it left off.*

*Iteration: Generator functions can be used to create iterable objects. When a generator function is used to create an iterable object, each yield statement returns the next value in the iteration. This allows you to write concise and expressive code for iterating over collections and generating sequences.*

*Synchronous vs. asynchronous: Regular functions are synchronous by default, meaning that they execute one line of code at a time, in sequence. Generator functions can be used to write asynchronous code that appears synchronous, by pausing and resuming execution as needed.*

*Overall, generator functions provide more control over the execution flow and allow for more expressive code, especially when it comes to iterating over collections and writing asynchronous code*

1. ***What is the syntax of a generator?***

*In JavaScript, a generator function is defined using the function\* syntax. Here is the basic syntax of a generator function:*

*function\* myGenerator() {*

*// generator function body*

*}*

*The function\* keyword indicates that this is a generator function, which can be used to generate a sequence of values using the yield keyword. The function body can contain any valid JavaScript code, including loops, conditionals, and other function calls.*

*Inside a generator function, you can use the yield keyword to pause the function and return a value. Here is an example:*

*function\* myGenerator() {*

*yield 1;*

*yield 2;*

*yield 3;*

*}*

*This generator function generates a sequence of three values: 1, 2, and 3. Each time the function is called, it returns the next value in the sequence. You can use the next() method to call the generator function and get the next value:*

*const gen = myGenerator();*

*console.log(gen.next()); // { value: 1, done: false }*

*console.log(gen.next()); // { value: 2, done: false }*

*console.log(gen.next()); // { value: 3, done: false }*

*console.log(gen.next()); // { value: undefined, done: true }*

1. ***Are function generators iterable in JavaScript?***

*Yes, function generators are iterable in JavaScript. Since generators produce a sequence of values, they can be used as the basis for an iterable object.*

*In JavaScript, an iterable object is an object that can be iterated over using a for...of loop or by using the spread operator. An iterable object must have a Symbol.iterator method that returns an iterator object. An iterator object is an object with a next() method that returns the next value in the sequence.*

*When a generator function is called, it returns an iterator object. You can use this iterator object to iterate over the values generated by the generator function. Here is an example:*

*function\* myGenerator() {*

*yield 1;*

*yield 2;*

*yield 3;*

*}*

*const gen = myGenerator();*

*for (let value of gen) {*

*console.log(value);*

*}*

*// Output:*

*// 1*

*// 2*

*// 3*

1. ***Create a generator for multiplying?***

*function\* multiplier(n) {*

*let current = 1;*

*while (true) {*

*yield n \* current;*

*current++;*

*}*

*}*

*const gen = multiplier(5);*

*console.log(gen.next().value*

*console.log(gen.next().value);*

*console.log(gen.next().value);*

*console.log(gen.next().value);*

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

*Here's an example of a generator function that generates an infinite series of natural numbers:*

*function\* naturalNumbers() {*

*let n = 1;*

*while (true) {*

*yield n++;*

*}*

*}*

*const gen = naturalNumbers();*

*console.log(gen.next().value); // Output: 1*

*console.log(gen.next().value); // Output: 2*

*console.log(gen.next().value); // Output: 3*

*console.log(gen.next().value); // Output: 4*

1. ***Create a generator that can throw an exception.***

*Here's an example of a generator function that can throw an exception:*

*function\* generatorWithException() {*

*try {*

*yield 1;*

*yield 2;*

*throw new Error('Generator Exception');*

*yield 3;*

*yield 4;*

*} catch (e) {*

*console.log(e.message);*

*}*

*}*

*const gen = generatorWithException();*

*console.log(gen.next().value); // Output: 1*

*console.log(gen.next().value); // Output: 2*

*console.log(gen.next()); // Output: Error: Generator Exception*

*console.log(gen.next().value); // Output: undefined*

*console.log(gen.next().value); // Output: undefined*