**1. isInteger Function**

javascript

// Checks if a number is an integer using bitwise operators

function isInteger(n) {

return (n ^ 0) === n;

}

This function uses the XOR (^) bitwise operator to check if a given number is an integer. If the number is already an integer, XORing it with 0 won't change its value.

**2. even Function**

javascript

// Returns an array of even numbers from 2 to 20

function even() {

return [2, 4, 6, 8, 10, 12, 14, 16, 18, 20];

}

This function simply returns an array of even numbers from 2 to 20 (inclusive).

**3. sumTo Function**

javascript

// Calculates the sum of numbers up to a given value using a loop

function sumTo(n) {

let sum = 0;

for (let i = 1; i <= n; i++) sum += i;

return sum;

}

This function calculates the sum of numbers up to the given value n using a loop.

**4. recSumTo Function**

javascript

// Calculates the sum of numbers up to a given value using recursion

function recSumTo(n) {

if (n <= 0) return 0;

return n + recSumTo(n - 1);

}

This function calculates the sum of numbers up to the given value n using recursion.

**5. factorial Function**

javascript

// Calculates the factorial of a given number

function factorial(n) {

let result = 1;

for (let i = 1; i <= n; i++) result \*= i;

return result;

}

This function calculates the factorial of a given number using a loop.

**6. isBinary Function**

javascript

// Checks if a number is a power of two

function isBinary(n) {

return (n != 0) && ((n & (n - 1)) == 0);

}

This function checks if a given number is a power of two using bitwise operations.

**7. fibonacci Function**

javascript

// Calculates the N-th Fibonacci number using recursion

function fibonacci(n) {

if (n <= 0) return 0;

else if (n == 1) return 1;

return fibonacci(n - 1) + fibonacci(n - 2);

}

This function calculates the N-th Fibonacci number using recursion.

**8. getOperationFn Function**

javascript

// Returns a function that performs a given operation on its argument

function getOperationFn(initialValue, operatorFn) {

if (operatorFn == null) return () => initialValue;

return (number) => {

initialValue = operatorFn(initialValue, number);

return initialValue;

}

}

This function returns a function that performs a given operation on its argument. If no operation is provided, it returns a function that always returns the initial value.

**9. sequence Function**

javascript

// Returns a function that generates an arithmetic sequence

function sequence(start = 0, step = 1) {

start -= step;

return () => {

start += step;

return start;

};

}

This function returns a function that generates an arithmetic sequence with a specified starting value (start) and step size (step).

**10. deepEqual Function**

javascript

// Checks if two objects are deeply equal

function deepEqual(firstObject, secondObject) {

// Implementation to check deep equality

}

This function checks if two objects are deeply equal, considering nested properties.

**11. Module Export**

javascript

if (typeof module === 'object') {

module.exports = {

isInteger,

even,

sumTo,

recSumTo,

factorial,

isBinary,

fibonacci,

getOperationFn,

sequence,

deepEqual,

};

}

This block exports the functions as a module if the code is running in a Node.js environment.

1. Can you explain how the isBinary function checks if a number is a power of two?
2. Explain the difference between the sumTo and recSumTo functions.
3. How would you optimize the fibonacci function for better performance?
4. What is the purpose of the getOperationFn function, and when would you use it?
5. Describe the purpose of the deepEqual function and provide an example of its usage.

**1. How does the isBinary function check if a number is a power of two?**

The isBinary function checks if a number is a power of two using bitwise operators. The key insight here is that binary representation of a power of two has only one '1' bit. The expression (n & (n - 1)) removes the rightmost '1' bit from the binary representation of n. If the result is 0, it means n had only one '1' bit, making it a power of two.

**2. Explain the difference between the sumTo and recSumTo functions.**

* The sumTo function calculates the sum of numbers up to a given value (n) using a loop. It initializes a variable sum to zero and then iterates from 1 to n, adding each number to the sum.
* The recSumTo function, on the other hand, uses recursion to calculate the sum. It has a base case: if n is less than or equal to 0, it returns 0. Otherwise, it adds n to the result of the recursive call with n-1. It effectively sums up the numbers from n to 1 recursively.

**3. How would you optimize the fibonacci function for better performance?**

The fibonacci function, as implemented, has an exponential time complexity due to redundant calculations. To optimize it, you could use memoization to store and reuse previously computed Fibonacci values. This avoids redundant recursive calls and significantly improves the performance, especially for larger values of n.

**4. What is the purpose of the getOperationFn function, and when would you use it?**

The getOperationFn function is a higher-order function that returns a function based on an initial value and an operation function. It is useful in scenarios where you want to create and use a function that performs a specific operation with a stored state. The returned function can be called multiple times with new values, and it will update and return the accumulated result based on the provided operation function.

**5. Describe the purpose of the deepEqual function and provide an example of its usage.**

The deepEqual function checks if two values or objects are deeply equal, meaning their values are the same or, if objects, their properties are deeply equal. It performs a recursive comparison for nested objects.

Example of usage:

javascript

const obj1 = { arr: [22, 33], text: 'text' };

const obj2 = { arr: [22, 33], text: 'text' };

const obj3 = { arr: [22, 3], text: 'text2' };

console.log(core.deepEqual(obj1, obj2)); // true

console.log(core.deepEqual(obj1, obj3)); // false

In this example, deepEqual returns true for obj1 and obj2 because their properties and nested array values are equal. It returns false for obj1 and obj3 because the nested array values are different