135. Calculate the Hypotenuse of a Right-Angled Triangle

The **calculateHypotenuse** function computes the length of the hypotenuse of a right-angled triangle given the lengths of its two perpendicular sides (legs). It uses the Pythagorean theorem to calculate the hypotenuse, which states that the square of the length of the hypotenuse (c) is equal to the sum of the squares of the lengths of the other two sides (a and b).

```
// Calculate the Hypotenuse of a Right-Angled Triangle
const calculateHypotenuse = (a: number, b: number): number
=> Math.sqrt(a ** 2 + b ** 2);

// Example usage:
console.log(calculateHypotenuse(3, 4));
// Output: 5
```

136. Find the Average of Odd Numbers in an Array

The **averageOfOddNumbers** function calculates the average of odd numbers within a given array. It does so by filtering the odd numbers from the array, then computing their sum and dividing by the count of odd numbers.

```
const averageOfOddNumbers = (arr: number[]): number => {
  const oddNumbers = arr.filter((num) => num % 2 !== 0);
  return oddNumbers.reduce((sum, num) => sum + num, 0) /
oddNumbers.length;
};

console.log(averageOfOddNumbers([1, 2, 3, 4, 5, 6, 7, 8, 9,
10]));
// Output: 5
```

137. Count the Letters in a String (case-insensitive)

The **countLetters** function calculates the count of each letter in a given string, considering both uppercase and lowercase versions as the same letter.

```
const countLetters = (str: string): { [key: string]: number
} => {
  const letters = str.toLowerCase().match(/[a-z]/g);
  const letterCount: { [key: string]: number } = {};
  for (const letter of letters) {
    letterCount[letter] = (letterCount[letter] || 0) + 1;
  }
  return letterCount;
};

console.log(countLetters("Hello, World!"));
// Output: { h: 1, e: 1, l: 3, o: 2, w: 1, r: 1, d: 1 }
```

138. Convert Seconds to Days, Hours, Minutes, and Seconds

The **secsToDaysHoursMinsSecs** function converts a given number of seconds into days, hours, minutes, and remaining seconds.

```
const secsToDaysHoursMinsSecs = (seconds: number): string
=> {
  const days: number = Math.floor(seconds / 86400);
  seconds %= 86400;
  const hours: number = Math.floor(seconds / 3600);
  seconds %= 3600;
  const minutes: number = Math.floor(seconds / 60);
  const remainingSecs: number = seconds % 60;
  return `${days} days, ${hours} hours, ${minutes} minutes,
and ${remainingSecs} seconds`;
};

console.log(secsToDaysHoursMinsSecs(100000));
// Output: "1 days, 3 hours, 46 minutes, and 40 seconds"
```

139. Check if a Number is a Prime Factor of Another Number

The **isPrimeFactor** function checks if a given number is a prime factor of another number.

```
const isPrimeFactor = (num: number, factor: number):
boolean =>
  num % factor === 0 && isPrime(factor);

const isPrime = (n: number): boolean => {
  if (n <= 1) return false;
  if (n <= 3) return true;

  if (n % 2 === 0 || n % 3 === 0) return false;

  for (let i = 5; i * i <= n; i += 6) {
    if (n % i === 0 || n % (i + 2) === 0) return false;
  }

  return true;
};

console.log(isPrimeFactor(20, 2));
// Output: true

console.log(isPrimeFactor(20, 3));
// Output: false
```

140. Find the Largest Prime Factor of a Number

The **largestPrimeFactor** function calculates the largest prime factor of a given number.

```
const largestPrimeFactor = (num: number): number => {
  let factor: number = 2;
  while (factor <= num) {
    if (num % factor === 0) {
      num /= factor;
    } else {
      factor++;
    }
  }
  return factor;
};

console.log(largestPrimeFactor(48));
// Output: 3
```

141. Check if a Number is a Pronic Square

The **isPronicSquare** function checks if a given number is a pronic square.

```
const isPronicSquare = (num: number): boolean =>
  Number.isInteger(Math.sqrt(num));

console.log(isPronicSquare(6)); // Output: true
console.log(isPronicSquare(20)); // Output: false
console.log(isPronicSquare(21)); // Output: true
```

142. Find the Sum of the Digits of a Number

The **sumOfDigits** function calculates the sum of the digits of a given number.

```
const sumOfDigits = (num: number): number =>
  [...String(num)].reduce((sum, digit) => sum +
Number(digit), 0);

console.log(sumOfDigits(12345));
// Output: 15
```

143. Calculate the Median of an Array of Numbers

The **median** function calculates the median of an array of numbers. The median is the middle value of a dataset when it is ordered.

```
const median = (arr: number[]): number => {
  const sorted: number[] = arr.slice().sort((a, b) => a -
b);
  const mid: number = Math.floor(sorted.length / 2);
  return sorted.length % 2 === 0
    ? (sorted[mid - 1] + sorted[mid]) / 2
    : sorted[mid];
};

console.log(median([1, 3, 5, 7, 9]));
// Output: 5
```

144. Find the Greatest Common Divisor (GCD) of Two Numbers (Recursive)

The **gcd** function calculates the greatest common divisor (GCD) of two given numbers using a recursive approach.

```
const gcd = (num1: number, num2: number): number =>
  num2 === 0 ? num1 : gcd(num2, num1 % num2);

console.log(gcd(48, 18));
// Output: 6
```

145. Check if a Number is a Happy Number

The **isHappyNumber** function checks if a given number is a "happy number" or not. A happy number is a number where the sequence of repeatedly summing the squares of its digits eventually reaches the number 1.

```
const isHappyNumber = (num: number): boolean => {
  const seen: Set<number> = new Set();
  while (num !== 1 && !seen.has(num)) {
    seen.add(num);
    num = [...String(num)].reduce((sum, digit) => sum +
Number(digit) ** 2, 0);
  }
  return num === 1;
};

console.log(isHappyNumber(19));
// Output: true
console.log(isHappyNumber(4));
// Output: false
```

146. Find the First N Prime Numbers

The **firstNPrimes** function generates an array of the first n prime numbers.

```
const isPrime = (num: number): boolean => {
  if (num <= 1) return false;
  for (let i = 2; i <= Math.sqrt(num); i++) {
    if (num % i === 0) return false;
  }
  return true;
};

const firstNPrimes = (n: number): number[] => {
  const primes: number[] = [];
  let num = 2;
  while (primes.length < n) {
    if (isPrime(num)) primes.push(num);
    num++;
  }
  return primes;
};

console.log(firstNPrimes(5));
// Output: [2, 3, 5, 7, 11]
```

147. Calculate the Volume of a Sphere

The **sphereVolume** function calculates the volume of a sphere given its radius. It uses the formula (4/3) * π * r^3, where r is the radius of the sphere.

```
const sphereVolume = (radius: number): number =>
  (4 / 3) * Math.PI * radius ** 3;

console.log(sphereVolume(5));
// Output: 523.5987755982989
```

148. Find the Longest Word in a Sentence

The **longestWord** function determines the longest word in a given sentence. It does this by splitting the sentence into words using spaces as separators and then using the reduce method to compare the length of each word and keep track of the longest one.

```
const longestWord = (sentence: string): string =>
  sentence
    .split(" ")
    .reduce(
      (longest, word) => (word.length > longest.length ?
word : longest),
      ""
    );

console.log(longestWord("The quick brown fox jumped over
the lazy dog"));
// Output: "jumped"
```

149. Check if a Number is an Armstrong Number (Narcissistic Number)

The **isArmstrongNumber** function checks if a number is an Armstrong number, also known as a narcissistic number.

```
const isArmstrongNumber = (num: number): boolean => {
  const digits: number[] = [...String(num)].map(Number);
  const numDigits: number = digits.length;
  const sumOfPowers: number = digits.reduce(
    (sum, digit) => sum + digit ** numDigits,
    0
  );
  return sumOfPowers === num;
};

console.log(isArmstrongNumber(153));
// Output: true
console.log(isArmstrongNumber(370));
// Output: true
console.log(isArmstrongNumber(123));
// Output: false
```

150. Find the Length of the Longest Word in a Sentence

The **longestWordLength** function calculates the length of the longest word in a given sentence. It splits the sentence into words using spaces as separators, and then uses the reduce method to find the maximum length among all the words.

```
const longestWordLength = (sentence: string): number =>
  sentence
    .split(" ")
    .reduce((longest, word) => Math.max(longest,
word.length), 0);

console.log(longestWordLength("The quick brown fox jumped
over the lazy dog")); // Output: 6
```

151. Check if a Number is a Strong Number

The **isStrongNumber** function checks if a number is a strong number. A strong number is a number whose sum of factorials of its digits is equal to the number itself.

```
const factorial = (num: number): number =>
  num === 0 ? 1 : num * factorial(num - 1);

const isStrongNumber = (num: number): boolean => {
  const sumOfFactorials: number = [...String(num)].reduce(
    (sum, digit) => sum + factorial(Number(digit)),
    0
  );
  return sumOfFactorials === num;
};

console.log(isStrongNumber(145));
// Output: true
console.log(isStrongNumber(123));
// Output: false
```

152. Reverse the Order of an Array

The **reverseArray** function reverses the order of elements in an array using the reverse method.

```
const reverseArray = <T>(arr: T[]): T[] =>
arr.slice().reverse();

console.log(reverseArray([1, 2, 3, 4, 5]));
// Output: [5, 4, 3, 2, 1]
```

153. Find the Area of a Rectangle

The **rectangleArea** function calculates the area of a rectangle given its length and width using the formula: length * width.

```
const rectangleArea = (length: number, width: number):
number => length * width;

console.log(rectangleArea(5, 10));
// Output: 50
```

154. Calculate the Sum of Even Numbers in an Array

The **sumOfEvenNumbers** function calculates the sum of even numbers within an array. It first filters the array to keep only the even numbers, and then uses the reduce method to compute their sum.

```
const sumOfEvenNumbers = (arr: number[]): number =>
  arr.filter((num) => num % 2 === 0).reduce((sum, num) =>
sum + num, 0);

console.log(sumOfEvenNumbers([1, 2, 3, 4, 5, 6, 7, 8, 9,
10]));
// Output: 30
```

155. Find the Greatest Common Divisor (GCD) of Two Numbers (Iterative)

The **gcdIterative** function calculates the greatest common divisor (GCD) of two numbers using an iterative approach. It employs the Euclidean algorithm to iteratively find the GCD.

```
const gcdIterative = (num1: number, num2: number): number
=> {
  while (num2 !== 0) {
    const temp: number = num2;
    num2 = num1 % num2;
    num1 = temp;
  }
  return num1;
};

console.log(gcdIterative(48, 18));
// Output: 6
```

156. Calculate the Volume of a Cylinder

The **cylinderVolume** function calculates the volume of a cylinder using the formula: π * radius^2 * height.

```
const cylinderVolume = (radius: number, height: number):
number =>
  Math.PI * radius ** 2 * height;

console.log(cylinderVolume(5, 10));
// Output: 785.3981633974483
```

157. Check if a Number is a Smith Number

The **isSmithNumber** function checks whether a given number is a Smith number.

```
const sumOfDigits = (num: number): number =>
  [...String(num)].reduce((sum, digit) => sum +
Number(digit), 0);

const sumOfPrimeFactors = (num: number): number => {
  let factor: number = 2;
  let sum: number = 0;
  while (num > 1) {
    if (num % factor === 0) {
      sum += sumOfDigits(factor);
      num /= factor;
    } else {
      factor++;
    }
  }
  return sum;
};

const isSmithNumber = (num: number): boolean =>
  sumOfDigits(num) === sumOfPrimeFactors(num);

console.log(isSmithNumber(666));
// Output: true
console.log(isSmithNumber(378));
// Output: true
console.log(isSmithNumber(123));
// Output: false
```

158. Convert Decimal Number to Octal

The **decimalToOctal** function converts a decimal (base 10) number to its octal (base 8) representation using the .toString() method with the base argument set to 8.

```
const decimalToOctal = (num: number): string =>
num.toString(8);

console.log(decimalToOctal(27));
// Output: "33"
```

159. Find the LCM of Two Numbers

The **lcm** function calculates the least common multiple (LCM) of two given numbers using the formula: (num1 * num2) / gcd(num1, num2).

```
const gcd = (num1: number, num2: number): number => {
  while (num2 !== 0) {
    const temp: number = num2;
    num2 = num1 % num2;
    num1 = temp;
  }
  return num1;
};

const lcm = (num1: number, num2: number): number =>
  (num1 * num2) / gcd(num1, num2);

console.log(lcm(24, 36));
// Output: 72
```

160. Check if a String is a Valid Phone Number (North American Format)

The **isValidPhoneNumber** function checks whether a given string is a valid phone number in the North American format "XXX-XXX-XXXX", where X represents a digit.

```
const isValidPhoneNumber = (phone: string): boolean =>
  /^\d{3}-\d{3}-\d{4}$/.test(phone);

console.log(isValidPhoneNumber("555-123-4567"));
// Output: true
console.log(isValidPhoneNumber("123-4567"));
// Output: false
```

161. Find the Sum of the First N Natural Numbers

The **sumOfNaturals** function calculates the sum of the first N natural numbers using the formula: (n * (n + 1)) / 2.

```
const sumOfNaturals = (n: number): number => (n * (n + 1))
/ 2;

console.log(sumOfNaturals(10));
// Output: 55
```

162. Check if a Number is a Perfect Number

The **isPerfectNumber** function checks whether a given number is a perfect number. A perfect number is a positive integer that is equal to the sum of its proper divisors (excluding itself).

```
const isPerfectNumber = (num: number): boolean => {
  let sum: number = 0;
  for (let i = 1; i <= num / 2; i++) {
    if (num % i === 0) sum += i;
  }
  return sum === num;
};

console.log(isPerfectNumber(28));
// Output: true
console.log(isPerfectNumber(12));
// Output: false
```

163. Find the Factors of a Number (excluding 1 and the number itself)

The **factors** function calculates the factors of a given number, excluding 1 and the number itself. It iterates through the numbers from 2 up to one less than the given number, checking if the given number is divisible by each of those numbers.

```
const factors = (num: number): number[] => {
  const result: number[] = [];
  for (let i = 2; i < num; i++) {
    if (num % i === 0) result.push(i);
  }
  return result;
};

console.log(factors(12));
// Output: [2, 3, 4, 6]
```

164. Calculate the Area of a Triangle given the Base and Height

The **triangleArea** function calculates the area of a triangle using the formula: 0.5 * base * height.

```
const triangleArea = (base: number, height: number): number
=>
  0.5 * base * height;

console.log(triangleArea(5, 10));
// Output: 25
```

165. Check if a String is a Valid Social Security Number (SSN)

The **isValidSSN** function checks whether a given string is a valid Social Security Number (SSN) in the format "XXX-XX-XXXX", where X represents a digit.

```
const isValidSSN = (ssn: string): boolean => /^\d{3}-\d{2}-
\d{4}$/.test(ssn);

console.log(isValidSSN("123-45-6789"));
// Output: true
console.log(isValidSSN("123-45-678"));
// Output: false
```

166. Generate an Array of Random Numbers within a Range

The **randomArrayInRange** function generates an array of random numbers within a specified range and of a specified length. It uses the **Array.from** method with a mapping function to create the desired array.

```
const randomArrayInRange = (
  min: number,
  max: number,
  length: number
): number[] =>
  Array.from(
    { length },
    () => Math.floor(Math.random() * (max - min + 1)) + min
  );

console.log(randomArrayInRange(1, 100, 5));
// Output: [34, 87, 19, 56, 72]
```

167. Check if a Number is a Magic Number

The **isMagicNumber** function checks whether a given number is a magic number. A magic number is a number that eventually reaches the value 1 when the sum of its digits is repeatedly calculated.

```
const isMagicNumber = (num: number): boolean => {
  let sum: number = 0;
  while (num > 0) {
    sum += num % 10;
    num = Math.floor(num / 10);
  }
  return sum === 1;
};

console.log(isMagicNumber(19));
// Output: true
console.log(isMagicNumber(123));
// Output: false
```

168. Check if a String is a Valid IPv4 Address

The **isValidIPv4** function checks whether a given string represents a valid IPv4 address.

```
const isValidIPv4 = (ip: string): boolean =>
  /^(?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-
9]?)\.){3}(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)$/.test(
    ip
  );

console.log(isValidIPv4("192.168.1.1"));
// Output: true
console.log(isValidIPv4("256.0.0.1"));
// Output: false
```

169. Convert Decimal Number to Hexadecimal

The **decimalToHex** function converts a decimal (base 10) number to its equivalent hexadecimal (base 16) representation using the built-in toString method with a radix of 16.

```
const decimalToHex = (num: number): string =>
num.toString(16);

console.log(decimalToHex(255));
// Output: "ff"
```

170. Check if a String is a Valid Date (YYYY-MM-DD Format)

The **isValidDate** function checks whether a given string is a valid date in the format "YYYY-MM-DD".

```
const isValidDate = (date: string): boolean => /^\d{4}-\d{2}-\d{2}$/.test(date);

console.log(isValidDate("2023-08-02"));
// Output: true
console.log(isValidDate("02-08-2023"));
// Output: false
```

171. Find the Smallest Common Multiple of an Array of Numbers

gcd function calculates the greatest common divisor using the Euclidean algorithm. The lcmArray function then calculates the least common multiple (LCM) of an array of numbers by reducing the array and applying the formula (lcm * num) / gcd(lcm, num).

```
const gcd = (num1: number, num2: number): number => {
  while (num2 !== 0) {
    const temp = num2;
    num2 = num1 % num2;
    num1 = temp;
  }
  return num1;
};

const lcmArray = (arr: number[]): number =>
  arr.reduce((lcm, num) => (lcm * num) / gcd(lcm, num), 1);

console.log(lcmArray([2, 3, 4, 5]));
// Output: 60
```

172. Check if a String is a Valid Password (At least 8 characters, with a digit and special character)

The **isValidPassword** function uses a regular expression to validate a password. The regular expression requires that the password contains at least one letter ([A-Za-z]), one digit (\d), and one special character ([@$!%*?&]).

```
const isValidPassword = (password: string): boolean =>
/^(?=.*[A-Za-z])(?=.*\d)(?=.*[@$!%*?&])[A-Za-
z\d@$!%*?&]{8,}$/.test(password);

console.log(isValidPassword("P@ssw0rd"));
// Output: true
console.log(isValidPassword("password123"));
// Output: false
```

173. Find the Nth Fibonacci Number

The **fibonacci** function calculates the Nth Fibonacci number using an iterative approach.

```
const fibonacci = (n: number): number => {
  if (n === 1) return 0;
  if (n === 2) return 1;
  let prev1 = 0;
  let prev2 = 1;
  let result: number;
  for (let i = 3; i <= n; i++) {
    result = prev1 + prev2;
    prev1 = prev2;
    prev2 = result;
  }
  return result!;
};

console.log(fibonacci(7));
// Output: 8
```

174. Check if a Number is a Deficient Number

The **isDeficientNumber** function determines whether a given number is a deficient number.

```
const sumOfProperDivisors = (num: number): number => {
  let sum = 0;
  for (let i = 1; i < num; i++) {
    if (num % i === 0) {
      sum += i;
    }
  }
  return sum;
};

const isDeficientNumber = (num: number): boolean =>
  num > sumOfProperDivisors(num);

console.log(isDeficientNumber(10));
// Output: true
console.log(isDeficientNumber(28));
// Output: false
```

175. Calculate the Distance between Two Points in 2D

The **distanceBetweenPoints** function calculates the Euclidean distance between two points in a 2D plane. Given the coordinates of two points (x1, y1) and (x2, y2), it uses the formula sqrt((x2 - x1)^2 + (y2 - y1)^2) to compute the distance between them.

```
const distanceBetweenPoints = (
  x1: number,
  y1: number,
  x2: number,
  y2: number
): number => Math.sqrt((x2 - x1) ** 2 + (y2 - y1) ** 2);

console.log(distanceBetweenPoints(0, 0, 3, 4));
// Output: 5
```