**MATHS ASSIGNMENT**

**1). Create a function that finds the maximum range of a triangle's third edge, where the side lengths are all integers.**

**Examples**

**nextEdge(8, 10) ➞ 17**

**nextEdge(5, 7) ➞ 11**

**nextEdge(9, 2) ➞ 10**

**Notes**

**● (side1 + side2) - 1 = maximum range of third edge.**

**● The side lengths of the triangle are positive integers.**

**● Don't forget to return the result.**

function nextEdge(side1,side2) {

    let maxrange = (side1 + side2) - 1;

    return maxrange;

}

let rangeOfThirdEdge = nextEdge(8,10);

console.log(rangeOfThirdEdge);

**OUTPUT**:

PS C:\Users\ADMIN\Documents\c\JS\_CODE> node assign\_15.js

17

**==================================================================================**

**2). The right shift operation is similar to floor division by powers of two. Write a function that mimics (without the use of >>) the right shift operator and returns the result from the two given integers. Try to solve this challenge by recursion.**

const rightShift = (first, second) => {

    let value = Math.floor(first/Math.pow(2, second));

    return value;

};

console.log(rightShift(20,3));

**PS C:\Users\ADMIN\Documents\c\JS\_CODE> node tryyy.js**

**2**

**==================================================================================**

**3). Create a function that takes numbers b and m as arguments and returns the second derivative of the function f(x)=x^b +x\* (e^(b\*m)) with respect to x evaluated at x=m, where b and m are constants.**

function derivative(b,m) {

    let x, e;

    let eq= x^b +x\* (e^(b\*m))

    let firstder = b\*(x^(b-1)) +  [0 + e^(b\*m)];

     x = m;

    let secder = b\*(b-1)\*(x^(b-2));

    return secder;

}

console.log(derivative(2,3));

**Output:**

**PS C:\Users\ADMIN\Documents\c\JS\_CODE> node derivative.js**

**6**

**==================================================================================**

**4). This Triangular Number Sequence is generated from a pattern of dots that form a triangle. The first 5 numbers of the sequence, or dots, are:**

**1, 3, 6, 10, 15**

**Examples**

**triangle(1) ➞ 1**

**triangle(6) ➞ 56 (1+3+6+10+15+21)**

function tri(num) {

    let sum = 0;

    for(let j =1;j <= num; j++) {

        sum += j;

    }

    return sum;

}

//console.log(tri(6));

function triangle(num) {

    let total = 0, temp, sum=0;

    for(let  i = 1; i <= num; i++) {

        sum += tri(i) ;

    }

    return sum;

}

console.log(triangle(6));

**OUTPUT:**

**PS C:\Users\ADMIN\Documents\c\JS\_CODE> node assign\_15.js**

**56**

**==================================================================================**

**5). Given a total due and an array representing the amount of change in your pocket, determine whether or not you are able to pay for the item. Change will always be represented in the following order: quarters, dimes, nickels, pennies.**

**To illustrate: changeEnough([25, 20, 5, 0], 4.25) should yield true, since having 25 quarters, 20 dimes, 5 nickels and 0 pennies gives you 6.25 + 2 + .25 + 0 = 8.50.**

**Examples**

**changeEnough([2, 100, 0, 0], 14.11) ➞ false**

**changeEnough([0, 0, 20, 5], 0.75) ➞ true**

**changeEnough([30, 40, 20, 5], 12.55) ➞ true**

**Notes**

**● quarter: 25 cents / $0.25**

**● dime: 10 cents / $0.10**

**● nickel: 5 cents / $0.05**

**● penny: 1 cent / $0.01**

let quarter =0.25

let dime = 0.10

let nickel = 0.05

let penny = 0.01

let sum = 0;

function changeEnough(arr1, multiple) {

    const arr2 = [quarter, dime, nickel, penny];

    const aa = arr1.map((x,i) => arr2[i] \* x ).reduce((sum, num) => sum+num);

    //console.log(aa);

    if (aa % multiple == 0) {

        console.log("true");

    }

    else {

        console.log("false");

    }

}

changeEnough([25, 20, 5, 0], 4.25);

changeEnough([2, 100, 0, 0], 14.11);

changeEnough([0, 0, 20, 5], 0.75);

changeEnough([30, 40, 20, 5], 12.55);

**OUTPUT:  
PS C:\Users\ADMIN\Documents\c\JS\_CODE> node tryyy.js**

**true**

**false**

**false**

**true**