function hallarDivisores(numero) {

numero = numero; //Se recibe el número al cual se le hallaran los divisores

var resultados = []; //Se crea un arreglo en el que se añadirán los divisores

for (let i = 0; i <= numero; i++) {

//Se hace un ciclo para todos los números menores que el recibido

if (numero % i == 0) {

//Se comprueba si el contador del ciclo es divisor de el numero recibido

resultados.push(" " + i); //Se agrega el numero al arreglo

}

}

imprimir(numero + " es divisible entre " + resultados); //Se imprime

return resultados; //Retorna los divisores

}

function hallarPrimos(numero) {

if (numero <= 1) {

//Si el numero es menor o igual a uno, por definición, no es primo

imprimir(

"Su numero es menor o igual a 1, se requiere un entero positivo mayor a 1."

);

} else if (numero > 2) {

var divisores; //Se crea un arreglo

divisores = hallarDivisores(numero); //En el arreglo se agregan los elementos obtenidos del método hallarDivisores

var toPrint = ""; // Se crea un string que contendrá el texto a imprimir

if (divisores.length == 2) {

//Si el numero de divisores es exactamente 2

toPrint = "El numero " + numero + " es primo"; //Significa que es primo

} else if (divisores.length > 2) {

//El numero de divisores es mayor a 2, es decir que es compuesto

var x = 2; // Se inicializa un contador en 2

var y = numero; // Se almacena el numero en cuestión

toPrint = "El numero " + y + " es compuesto y se puede expresar como: "; //Se agrega a la variable el texto a imprimir

while (y > 1) {

// Mientras la variable y sea mayor a 1

if (y % x == 0) {

// Si el residuo de la división y/x es 0

y = y / x; //Se toma el cociente de la división, y se reemplaza por este en la variable y

toPrint += x + "x"; //Se agrega x y el signo "x" al texto a imprimir

//Si el mismo numero (x) sigue dividiendo a y se toma el mismo

} else {

//Si el valor del contador (x) no divide a y se le aumenta uno

x++;

}

}

if (toPrint.lastIndexOf("x")) {

toPrint = toPrint.slice(0, -1); //Se elimina el ultimo símbolo "x"

}

}

imprimir(toPrint); //Se imprime la variable

}

}

function hallarMCD(numero1, numero2) {

var x = Math.max(numero1, numero2); //Se le asigna a x el mayor de los dos números ingresados

var y = Math.min(numero1, numero2); //Se le asigna a y el menor de los dos números ingresados

x = Math.abs(x); //Se toma el valor absoluto del numero

y = Math.abs(y); //Se toma el valor absoluto del numero

var residuo; //Se crea una variable residuo

var toPrint; //Se crea una variable que contendrá el texto a imprimir

// Algoritmo de Euclides

do {

residuo = y; // Se le asigna a la variable "residuo" el valor de y

y = x % y; // Se le asigna a y el residuo de la división

x = residuo; //Se le asigna a x el valor anterior de y, ahora llamado residuo

} while (y != 0); //Mientras que y sea diferente a 0

if (x == 1) {

//Si el ultimo valor de x (el MCD) es 1 (es decir, cuando y sea 0) quiere decir que los números son primos relativos

toPrint =

"Los números " + numero1 + " y " + numero2 + " son primos relativos"; //Se agrega el texto a imprimir

} else {

// El MCD es diferente de 1

toPrint =

"El Máximo común Divisor entre " +

numero1 +

" y " +

numero2 +

" es: " +

x; //Se agrega el texto a imprimir

}

imprimir2(toPrint); //Se imprime

return x; //Se retorna el MCD

}

function hallarMCM(numero1, numero2) {

var q = hallarMCD(numero1, numero2); //Se halla el MCD de los dos números, a través de la función hallarMCD

var x = Math.max(numero1, numero2); //Se le asigna a x el mayor de los dos números ingresados

var y = Math.min(numero1, numero2); //Se le asigna a y el menor de los dos números ingresados

x = Math.abs(x); //Se toma el valor absoluto de x

y = Math.abs(y); //Se toma el valor absoluto de y

var MCM = (x \* y) / q; //El mínimo común múltiplo es equivalente a multiplicar los dos números y dividirlos entre el MCD

imprimir2("El Mínimo común Múltiplo entre " + x + " y " + y + " es: " + MCM); //Se imprime

}

function isPrimo(numero) {

if (numero <= 1) {

return false; // Si el numero es menor a uno o igual, por definición, no es primo

} else if (numero > 1) {

// Si el numero es mayor a uno

var divisores = hallarDivisores(numero); // Utilizando el método hallarDivisores se guardan los divisores

if (divisores.length == 2) {

// Si el arreglo divisores tiene exactamente dos divisores

return true; // Es primo

} else {

// Si tiene un numero diferente de divisores

return false; // Significa que es compuesto

}

}

}

function Conjetura(i) {

var toPrint = ""; //Se crea una variable con el texto a imprimir

for (let n = 2; n <= i; n++) {

//Se hace un ciclo para todos los números mayores a 2(el primo mas pequeño) y menores al número dado

var aux = i - n; //Se despeja de la suma uno de los primos, quedando (P1 = i - P2), siendo P2 = n

if (isPrimo(n) && isPrimo(aux)) {

// Se confirma que los dos números sean primos

toPrint = "El numero " + i + " es igual a " + n + " + " + aux + "<br>"; //Se almacena para imprimir

return toPrint; //Se retorna

}

}

return toPrint; //Se retorna (vacío)

}

function CicloConjetura(tamaño) {

var toPrint = ""; //Se crea una variable que contendrá el texto a imprimir

var sum; //Se crea una variable sum que contendrá cada suma

for (let i = 2; i <= tamaño \* 2 + 2; i += 2) {

//Se hace un ciclo para la cantidad de números digitada

sum = Conjetura(i); //Se obtiene la suma de cada número y se guarda en la variable sum

toPrint += sum; //Se agrega cada suma al texto a imprimir

}

imprimir3(toPrint); //Se imprime

}

function relaciones(universalS, relacionS) {

var relacion; //Se crea arreglo de la relacion

var universal; //Se crea arreglo de la relacion

relacionS = relacionS.replace(/\s+/g, ""); //Se eliminan los espacios

universalS = universalS.replace(/\s+/g, ""); //Se eliminan los espacios

relacion = relacionS.split(";"); //Se separa por punto y coma

universal = universalS.split(";"); //Se separa por punto y coma

resultado.innerHTML = ""; //Se vacían los resultados de otros puntos

resultado2.innerHTML = ""; //

resultado3.innerHTML = ""; //

resultado4.innerHTML = ""; //

//Comprobar que todos los elementos pertenecen al conjunto A

for (let j = 0; j < relacion.length; j++) {

//Ciclo que recorre toda la relacion

const element = relacion[j].split(","); //Separa el elemento en los puntos

for (let q = 0; q < element.length; q++) {

//Se recorren los puntos de cada elemento

const x = element[q];

if (!universal.includes(x)) {

//Si el elemento no pertenece al universal

resultado4.innerHTML =

"<br/>El elemento " + x + " no pertenece al conjunto A"; //Imprime

}

}

}

var PrCruz = [];

var Reflexiva = [];

for (let i = 0; i < universal.length; i++) {

//Ciclo que recorre toda la relacion

const e1 = universal[i]; //Almacena el valor 1

for (let j = 0; j < universal.length; j++) {

//Ciclo que recorre de nuevo toda la relacion

const e2 = universal[j]; //Almacena el valor 2

if (e1 != "," && e2 != "," && (e1 != ";" && e2 != ";")) {

//No tiene en cuenta comas ni punto y coma

// Halla el producto cruz

PrCruz.push(e1 + "," + e2); //Une todos los puntos con todos y los agrega al arreglo PrCruz

//Halla los elementos que debe contener para ser Reflexiva

if (e1 == e2) {

//Si ambos puntos son iguales

Reflexiva.push(e1 + "," + e2); //Los une y los agrega al arreglo Reflexiva

}

}

}

}

//Reflexiva

var auxRef = true; //Variable auxiliar relaciones reflexivas

var faltRef = []; //Conjunto de parejas que faltan para que sea reflexiva

for (let i = 0; i < Reflexiva.length; i++) {

//Recorre todo el arreglo Reflexiva(Que contiene las parejas necesarias para que sea reflexiva)

if (!relacion.includes(Reflexiva[i])) {

//Si no incluye a todos

auxRef = auxRef && false; //Deja de ser reflexiva

faltRef.push(Reflexiva[i]); //Agrega las relaciones faltantes a un arreglo

} else {

auxRef = auxRef && true; //Mantiene el valor

}

}

if (auxRef == true) {

//Si la relacion es reflexiva

resultado4.innerHTML += "<br/>La relacion es reflexiva"; //Imprime

} else {

//Si no es reflexiva

resultado4.innerHTML +=

"<br/>La relacion no es reflexiva ya que las parejas {"; //Imprime

for (let i = 0; i < faltRef.length; i++) {

//Ciclo que recorre todos las parejas faltantes

resultado4.innerHTML += "(" + faltRef[i] + "); "; //Imprime todas las parejas

}

resultado4.innerHTML += "} no pertenecen a la relacion"; //Imprime

resultado4.innerHTML = resultado4.innerHTML.replace("; }", "}"); //Elimina el "; " final

}

//Simétrica

var auxSim = true; //Variable auxiliar relaciones Simétricas

var auxSimAsim = true; //Variable auxiliar relaciones Simétricas y antisimétricas

var faltSim = []; //Conjunto de las parejas faltantes para que sea simétrica

for (let i = 0; i < relacion.length; i++) {

//Ciclo que recorre todas las parejas de la relacion

const element = relacion[i]; //element sera la pareja actual

var elementA = element.split(","); //Se la pareja separa por coma

var simetria = elementA[1] + "," + elementA[0]; //Se invierte la pareja

if (relacion.includes(simetria)) {

//Se verifica que la pareja invertida también pertenece a la relacion

auxSim = auxSim && true; //Se mantiene el valor

if (element == simetria) {

//Si ambas parejas son iguales (es decir, x=y)

auxSimAsim = auxSimAsim && true; //La relacion es simétrica y antisimétrica

} else {

auxSimAsim = auxSimAsim && false; //La relacion es solamente simétrica

}

} else {

auxSim = auxSim && false; //La relacion se vuelve antisimétrica

faltSim.push(simetria); //Almacena la pareja por la cual no es simétrica

}

}

if (auxSim == true && auxSimAsim == true) {

//Si la relacion es simétrica y antisimétrica

resultado4.innerHTML += "<br/>La relacion es simétrica y antisimétrica"; //Imprime

} else if (auxSim == true) {

//Si la relacion es simétrica

resultado4.innerHTML += "<br/>La relacion es simétrica"; //Imprime

} else {

//Si la relacion es antisimétrica

resultado4.innerHTML +=

"<br/> La relacion es Antisimétrica ya que las parejas {"; //Imprime

for (let i = 0; i < faltSim.length; i++) {

//Ciclo para todas las parejas por las cuales la relacion es antisimétrica

resultado4.innerHTML += "(" + faltSim[i] + "); "; //Imprime

}

resultado4.innerHTML += "} no pertenecen a la relacion"; //Imprime

resultado4.innerHTML = resultado4.innerHTML.replace("; }", "}"); //Elimina el "; " final

}

// Transitiva

var auxTran = true; //Variable auxiliar relacion transitiva

var buscado;

var buscadoNE = [];

for (let i = 0; i < relacion.length; i++) {

//Ciclo que recorre todas las parejas

const element1 = relacion[i]; // Almacena la pareja actual

var element1A = element1.split(","); // Separa la pareja actual

for (let j = 0; j < relacion.length; j++) {

//Ciclo que recorre nuevamente todas la parejas

const element2 = relacion[j]; //Almacena la segunda pareja

var element2A = element2.split(","); //Separa la segunda pareja

if (element1A[1] == element2A[0]) {

//Compara si el segundo elemento de la P1 es igual al primero de la P2

buscado = element1A[0] + "," + element2A[1]; //Hace una nueva pareja con el primer elemento de la P1 y el segundo de la P2

if (relacion.includes(buscado)) {

//Revisa si la relacion incluye esa nueva pareja

auxTran = auxTran && true; //Lo cual significa que la relacion mantiene su transitividad

} else {

//Si la relacion no incluye la nueva pareja

auxTran = auxTran && false; //Significa que la relacion ya no es transitiva

buscadoNE.push(buscado); //Agrega la nueva pareja al arreglo buscadoNE

}

}

}

}

if (auxTran == true) {

//La relacion es transitiva

resultado4.innerHTML += "<br/>La relacion es transitiva"; //Imprime

} else if (auxTran == false) {

//La relacion no es transitiva

resultado4.innerHTML +=

"<br/>La relacion no es transitiva ya que las parejas {"; //Imprime

for (let i = 0; i < buscadoNE.length; i++) {

//Ciclo para las parejas por las cuales la relacion no es simétrica

resultado4.innerHTML += "(" + buscadoNE[i] + "); "; //Imprime

}

resultado4.innerHTML += "} no pertenecen a la relacion"; //Imprime

resultado4.innerHTML = resultado4.innerHTML.replace("; }", "}"); //Elimina el "; " final

}

// Equivalencia u orden

if (auxRef == true && auxSimAsim == true && auxTran == true) {

//Si la relacion es Reflexiva, Simétrica, Antisimétrica y Transitiva

resultado4.innerHTML += "<br/> La relacion es de Equivalencia y de Orden"; //Quiere decir que es de Equivalencia y de Orden

} else if (auxRef == true && auxSim == true && auxTran == true) {

//Si la relacion es Reflexiva, Simétrica y Transitiva

resultado4.innerHTML += "<br/> La relacion es de Equivalencia"; //Quiere decir que la relacion es de Equivalencia

} else if (auxRef == true && auxSim == true && auxTran == true) {

//Si la relacion es Reflexiva, Antisimética y Transitiva

resultado4.innerHTML += "<br/> La relacion es de Orden"; //Quiere decir que la relacion es de Orden

} else {

//Si la relacion no es Reflexiva, Simétrica, Antisimétrica o Transitiva quiere decir que la relacion no es

resultado4.innerHTML +=

"<br/> La relacion no es de Orden ni de Equivalencia"; // ni de Orden ni de Equivalencia

}

}