## Permutaciones en Java



## Permutaciones pares en Java



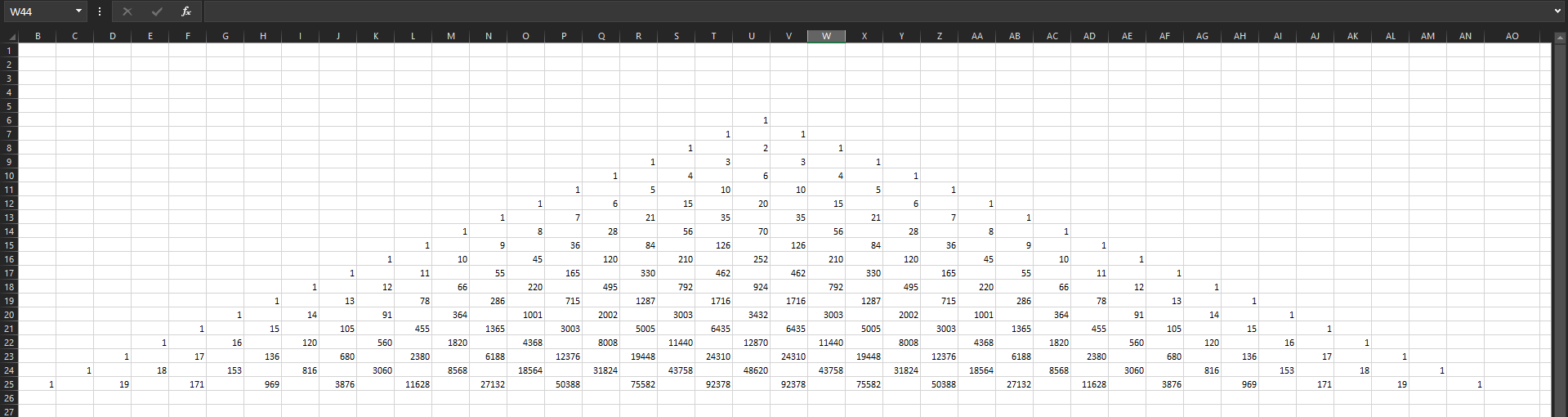
Permutaciones de Placas en Java



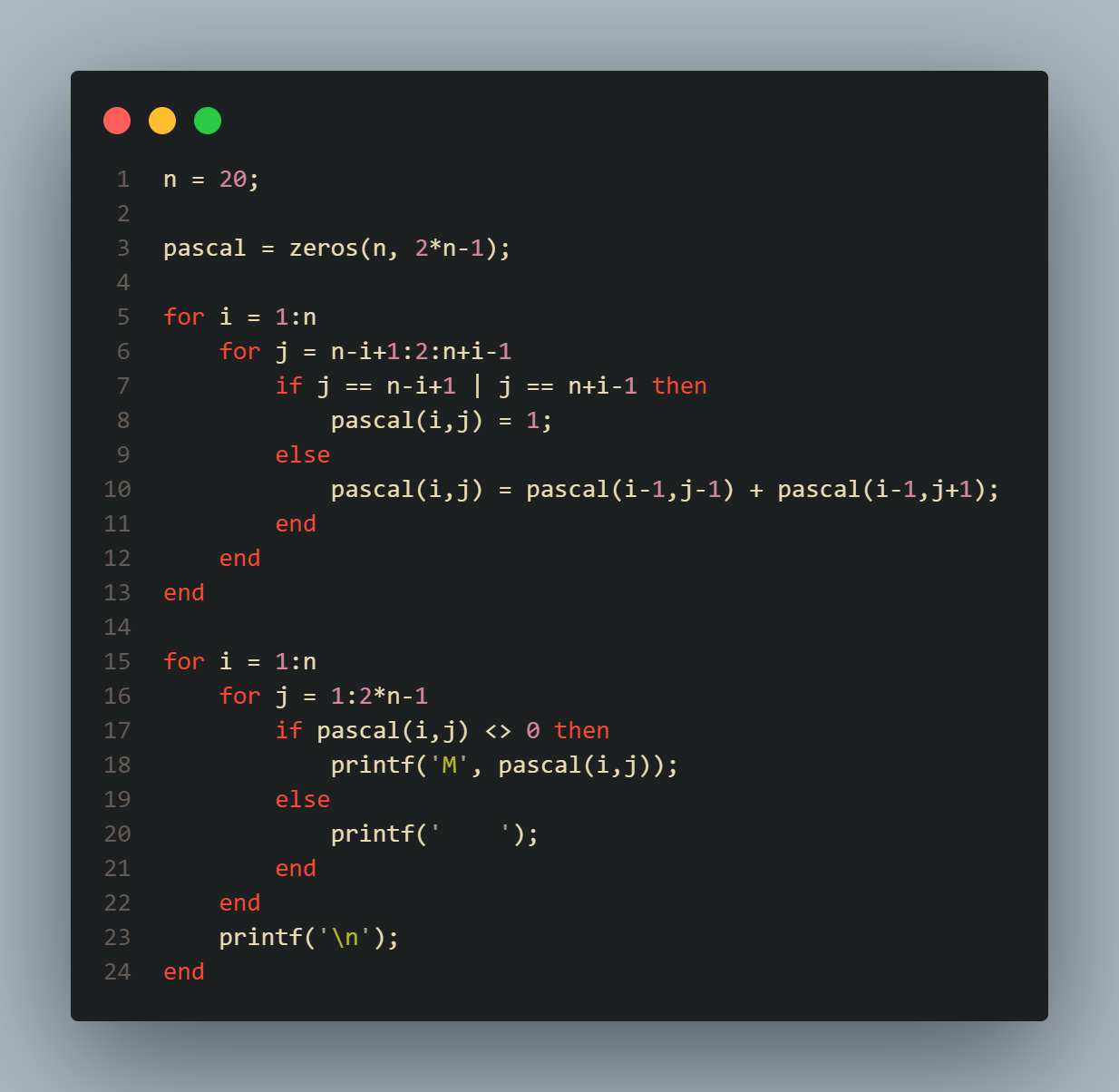
## Permutaciones en scilab



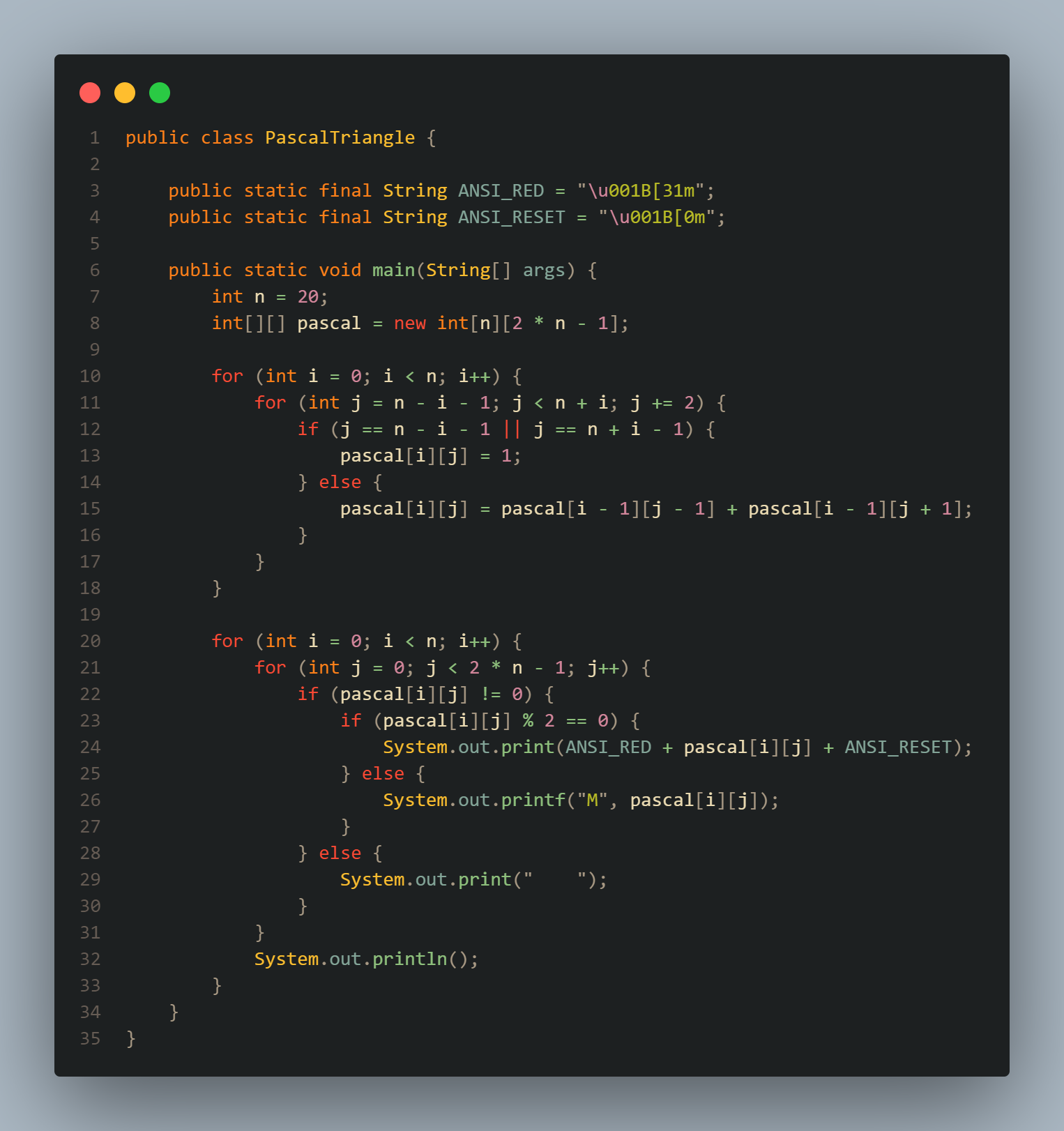
## Triángulo de Pascal en Excel



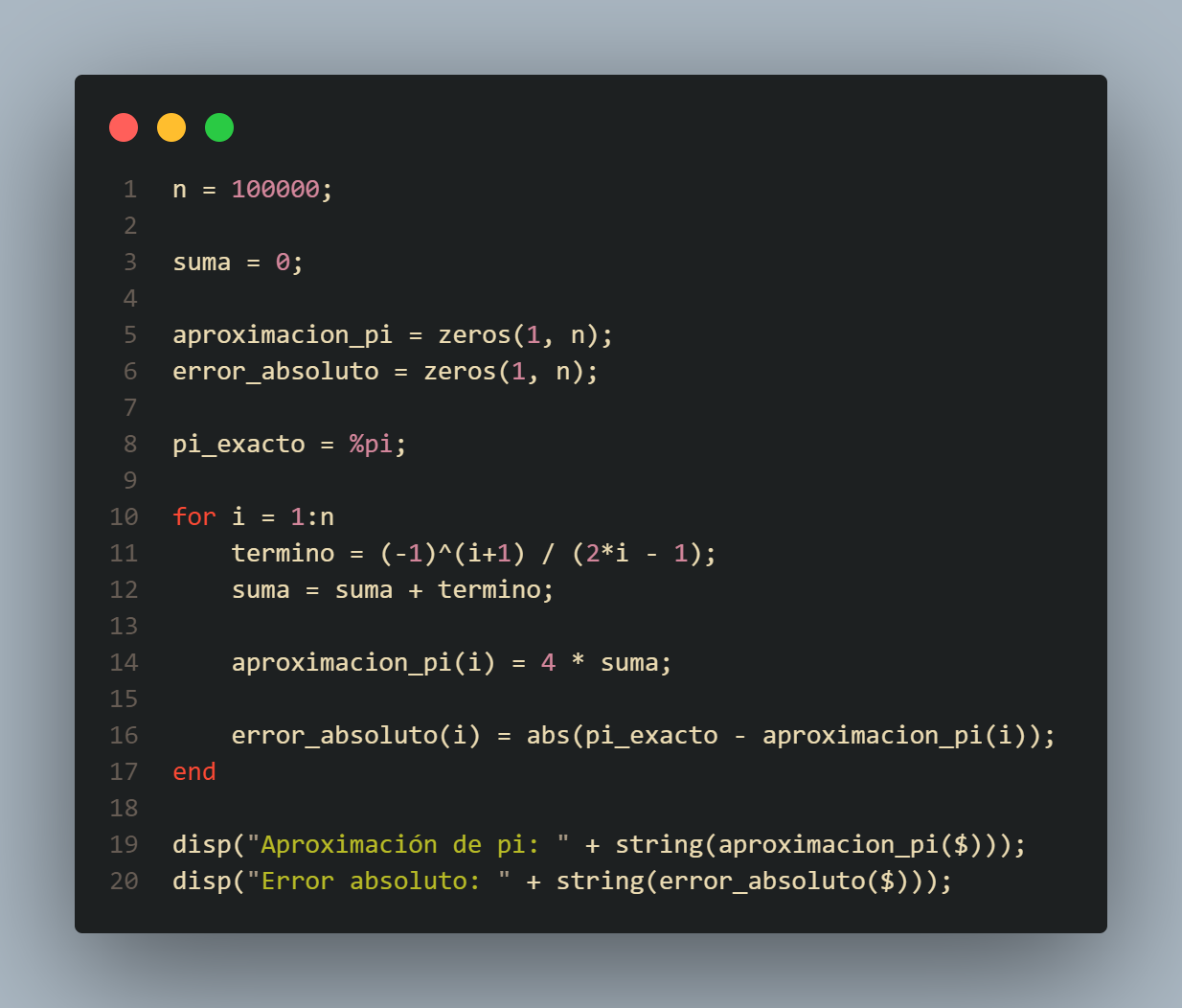
## Triángulo de Pascal en Scilab



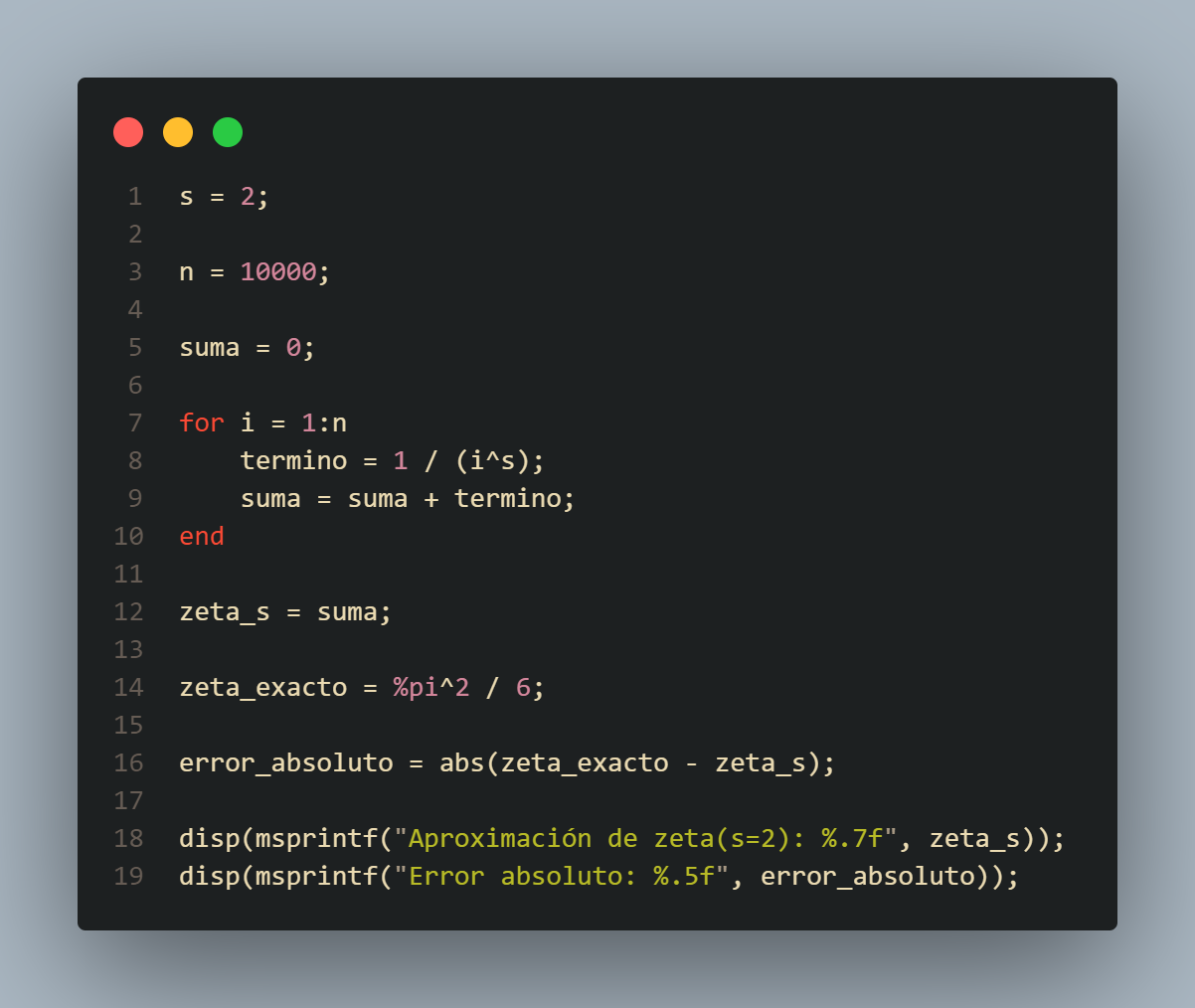
## Triángulo de Pascal con números pares de color en Java



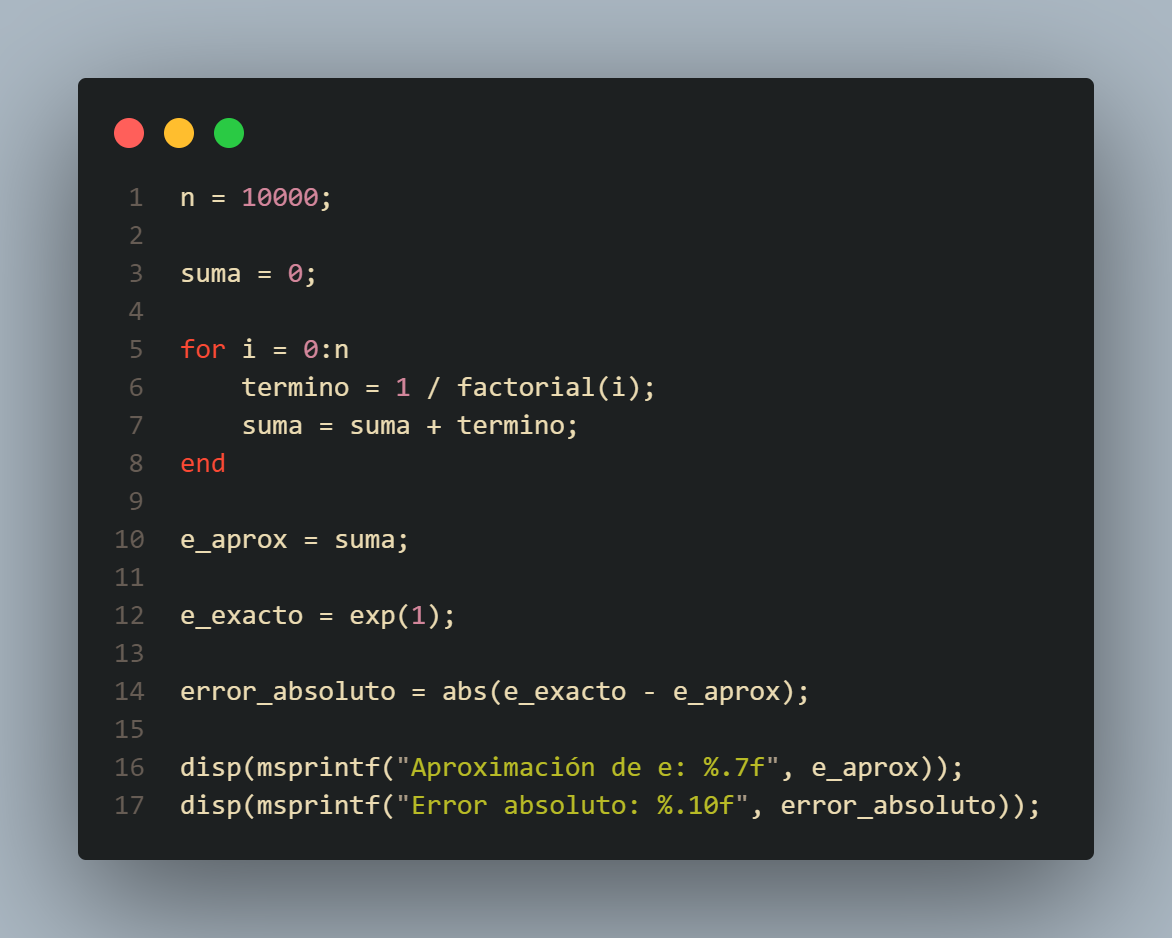
## Serie de Leibniz en Scilab



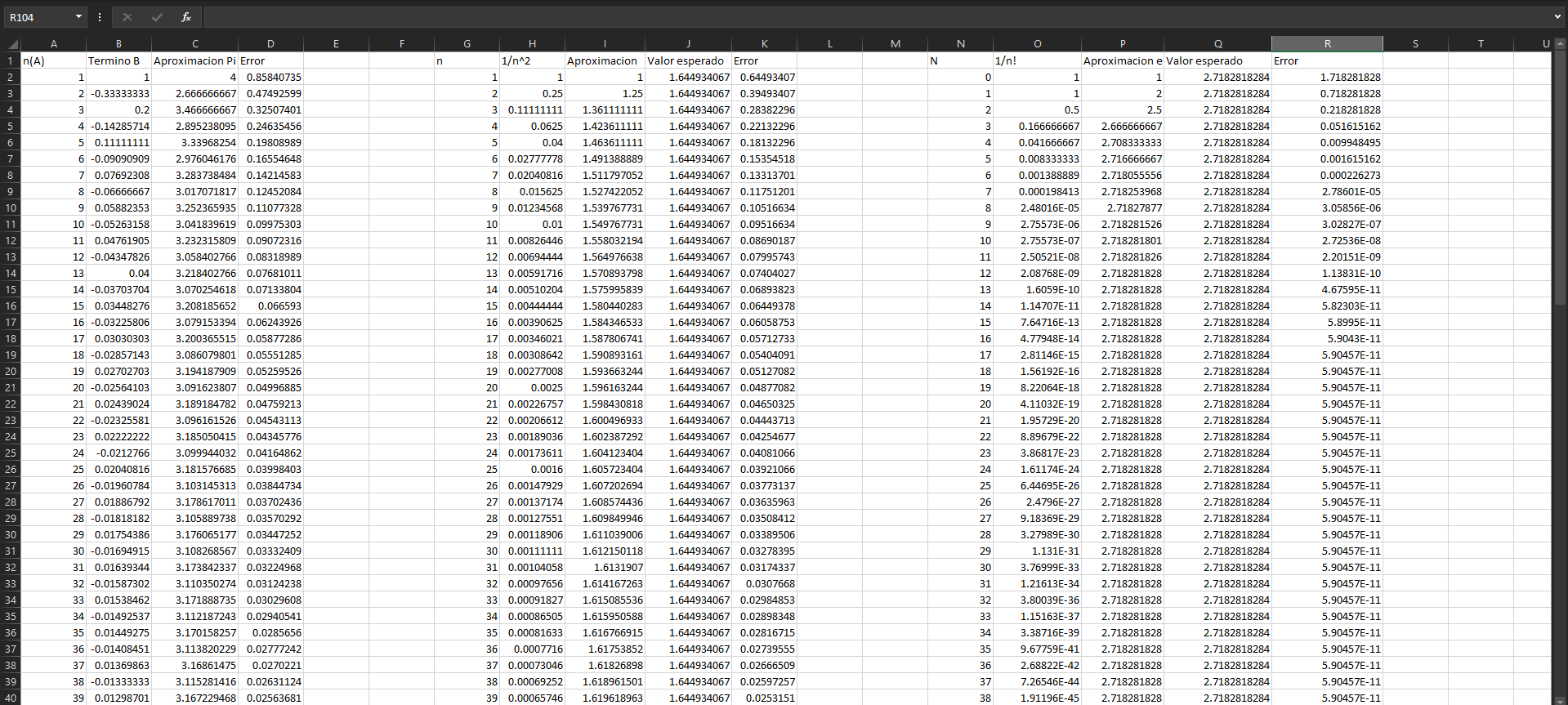
## Serie Zeta de Riemann en Scilab



## e como serie exponencial en Scilab



## Series en Excel



## Ejercicios de Sumatorias en Java

### 1



### 2



### 3



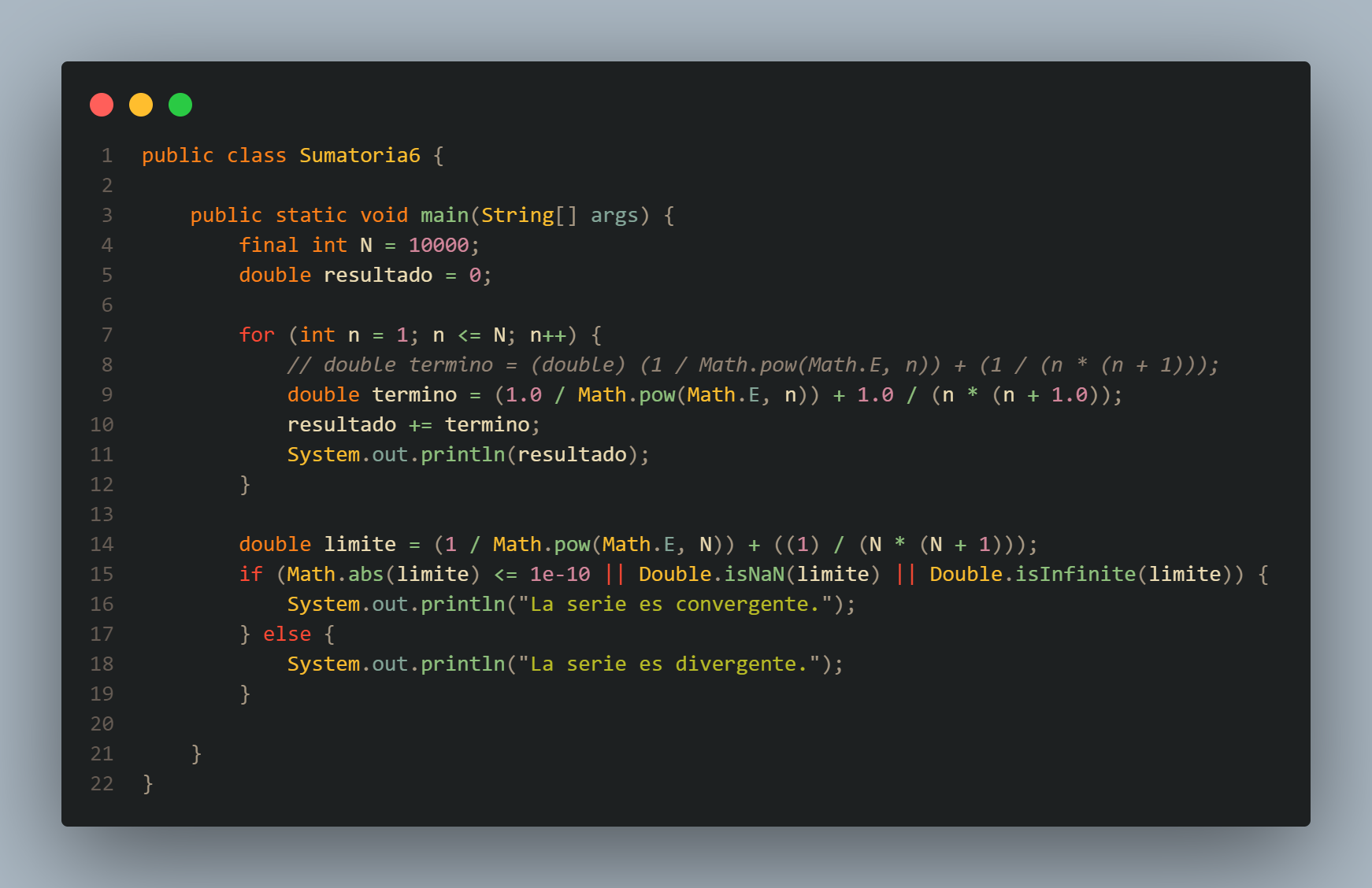
### 4



### 5



### 6



### 7



## Método del punto fijo en Scilab



### Método del punto fijo en Java

import java.util.Arrays;

public class MetodoPuntoFijo {

    public static double[] puntoFijo(FixedPointFunction g, double x0, double tol, int maxIter) {

        int iter = 0;

        double x1 = g.apply(x0);

        if (Math.abs(x1 - x0) < tol) {

            System.out.println("Iteración: " + iter + ", x: " + x1 + " (Raíz)");

            return new double[] { x1, iter };

        }

        while (iter < maxIter) {

            x1 = g.apply(x0);

            iter++;

            if (Math.abs(x1 - x0) < tol) {

                System.out.println("Iteración: " + iter + ", x: " + x1 + " (Raíz)");

                return new double[] { x1, iter };

            }

            x0 = x1;

        }

        return new double[] { x1, iter };

    }

    public static double[] recorrerVector(FixedPointFunction g, double[] xValues, double tol, int maxIter,

            Double previousRoot) {

        int startIndex = 0;

        if (previousRoot != null) {

            startIndex = Arrays.binarySearch(xValues, previousRoot) + 1;

        }

        if (startIndex >= xValues.length) {

            return new double[] { Double.NaN, maxIter };

        }

        for (int i = startIndex; i < xValues.length; i++) {

            double x0 = xValues[i];

            double[] result = puntoFijo(g, x0, tol, maxIter);

            double root = result[0];

            int iter = (int) result[1];

            if (Math.abs(g.apply(root) - root) < tol) {

                return new double[] { root, iter };

            }

        }

        return new double[] { Double.NaN, maxIter };

    }

    public static void MetodoPuntoFijo(double a, double b, double c, double d) {

        double tol = 1e-5;

        int maxIter = 20;

        double[] xValues = new double[17];

        for (int i = 0; i < xValues.length; i++) {

            xValues[i] = -6 + i;

        }

        Double previousRoot = null;

        if (a == 0 && b == 0) {

            System.out.println("Caso lineal: cx + d = 0");

            if (c != 0) {

                FixedPointFunction g1 = (x) -> -d / c;

                double[] rootIter1 = recorrerVector(g1, xValues, tol, maxIter, previousRoot);

                double root1 = rootIter1[0];

                int iter1 = (int) rootIter1[1];

                if (!Double.isNaN(root1)) {

                    System.out.println("Despeje 1: x = -d/c, Raíz aproximada: " + root1 + ", Iteraciones: " + iter1);

                    previousRoot = root1;

                }

            }

        } else if (a == 0) {

            System.out.println("Caso cuadrático: bx^2 + cx + d = 0");

            if (b != 0) {

                FixedPointFunction g1 = (x) -> (-d - b \* x \* x) / c;

                double[] rootIter1 = recorrerVector(g1, xValues, tol, maxIter, previousRoot);

                double root1 = rootIter1[0];

                int iter1 = (int) rootIter1[1];

                if (!Double.isNaN(root1)) {

                    System.out.println(

                            "Despeje 1: x = (-d - b\*x^2) / c, Raíz aproximada: " + root1 + ", Iteraciones: " + iter1);

                    previousRoot = root1;

                }

            }

            FixedPointFunction g2 = (x) -> {

                double value = (-c \* x - d) / b;

                return value < 0 ? Double.NaN : Math.sqrt(value);

            };

            double[] rootIter2 = recorrerVector(g2, xValues, tol, maxIter, previousRoot);

            double root2 = rootIter2[0];

            int iter2 = (int) rootIter2[1];

            if (!Double.isNaN(root2)) {

                System.out.println(

                        "Despeje 2: x = sqrt((-c\*x - d)/b), Raíz aproximada: " + root2 + ", Iteraciones: " + iter2);

                previousRoot = root2;

            }

        } else {

            System.out.println("Caso cúbico: ax^3 + bx^2 + cx + d = 0");

            FixedPointFunction g1 = (x) -> (-a \* x \* x \* x - b \* x \* x - d) / c;

            double[] rootIter1 = recorrerVector(g1, xValues, tol, maxIter, previousRoot);

            double root1 = rootIter1[0];

            int iter1 = (int) rootIter1[1];

            if (!Double.isNaN(root1)) {

                System.out.println("Despeje 1: x = (-a\*x^3 - b\*x^2 - d) / c, Raíz aproximada: " + root1

                        + ", Iteraciones: " + iter1);

                previousRoot = root1;

            }

            FixedPointFunction g2 = (x) -> {

                double value = (-c \* x - d - a \* x \* x \* x) / b;

                return value < 0 ? Double.NaN : Math.sqrt(value);

            };

            double[] rootIter2 = recorrerVector(g2, xValues, tol, maxIter, previousRoot);

            double root2 = rootIter2[0];

            int iter2 = (int) rootIter2[1];

            if (!Double.isNaN(root2)) {

                System.out.println("Despeje 2: x = sqrt((-c\*x - d - a\*x^3) / b), Raíz aproximada: " + root2

                        + ", Iteraciones: " + iter2);

                previousRoot = root2;

            }

            FixedPointFunction g3 = (x) -> {

                double value = (-b \* x \* x - c \* x - d) / a;

                return value < 0 ? Double.NaN : Math.cbrt(value);

            };

            double[] rootIter3 = recorrerVector(g3, xValues, tol, maxIter, previousRoot);

            double root3 = rootIter3[0];

            int iter3 = (int) rootIter3[1];

            if (!Double.isNaN(root3)) {

                System.out.println("Despeje 3: x = cbrt((-b\*x^2 - c\*x - d) / a), Raíz aproximada: " + root3

                        + ", Iteraciones: " + iter3);

            }

        }

    }

    public static void main(String[] args) {

        MetodoPuntoFijo(1, 1.11, -19.6282, 20.22744);

    }

}

@**FunctionalInterface**

interface FixedPointFunction {

    double apply(double x);

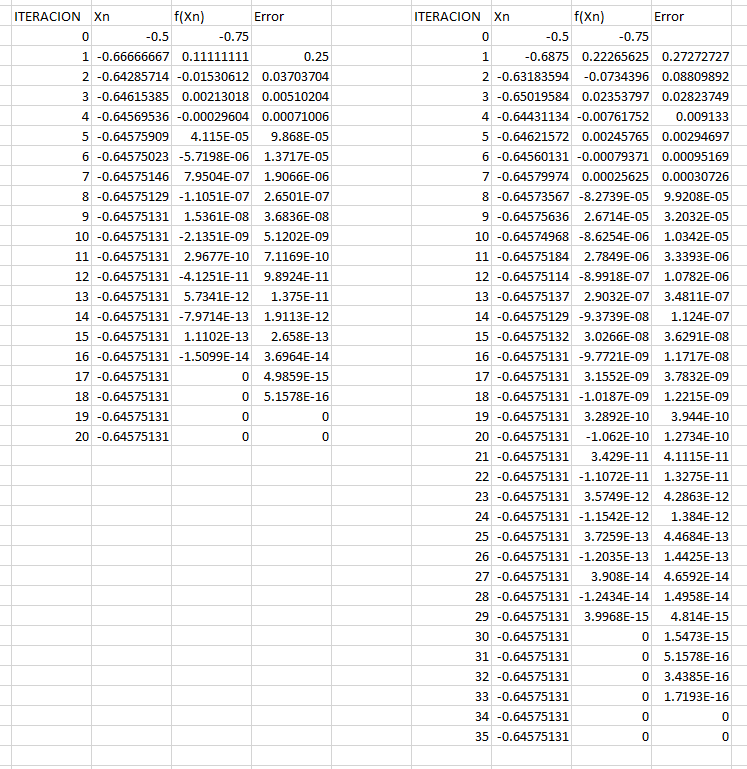
}

### Método del punto fijo en Python

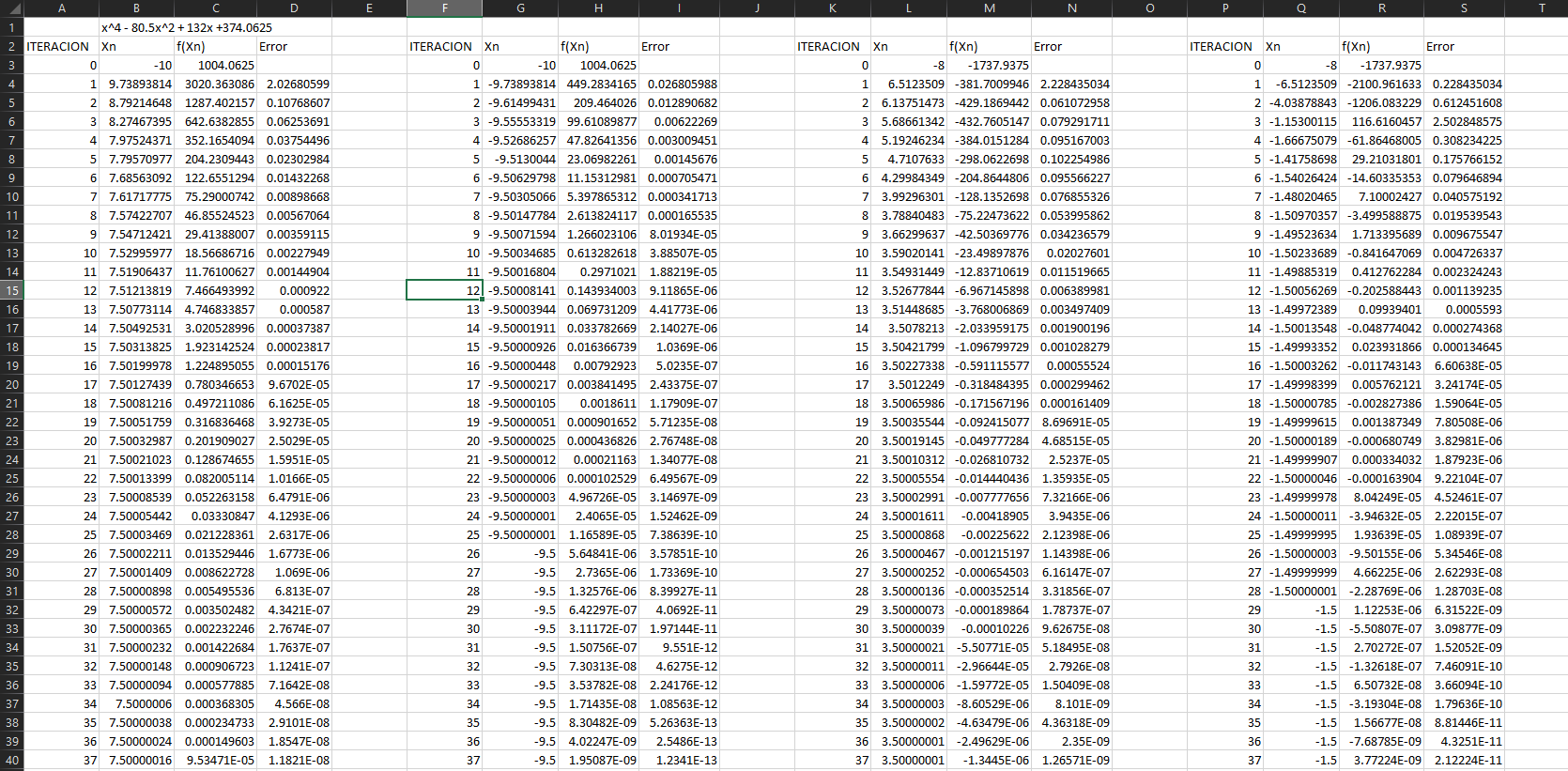


## Ejercicios del método de punto fijo en Excel

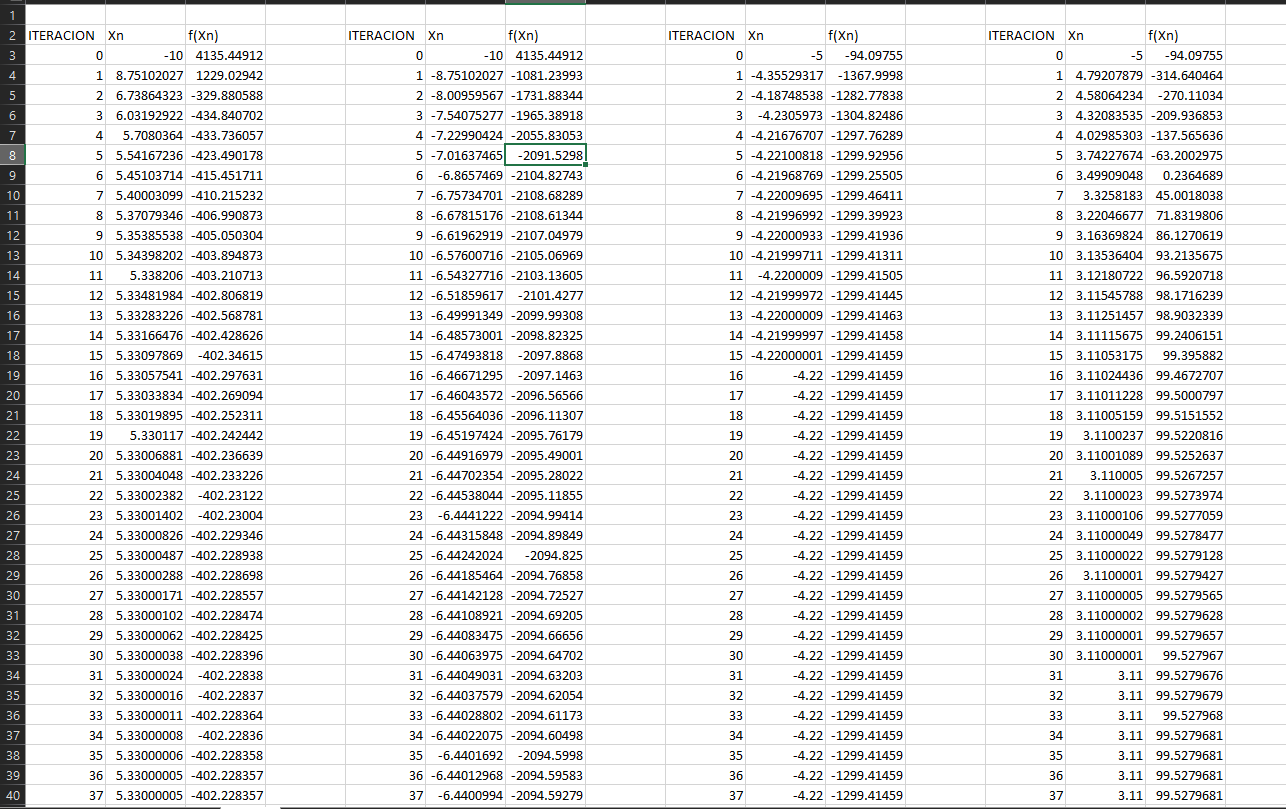
### 1



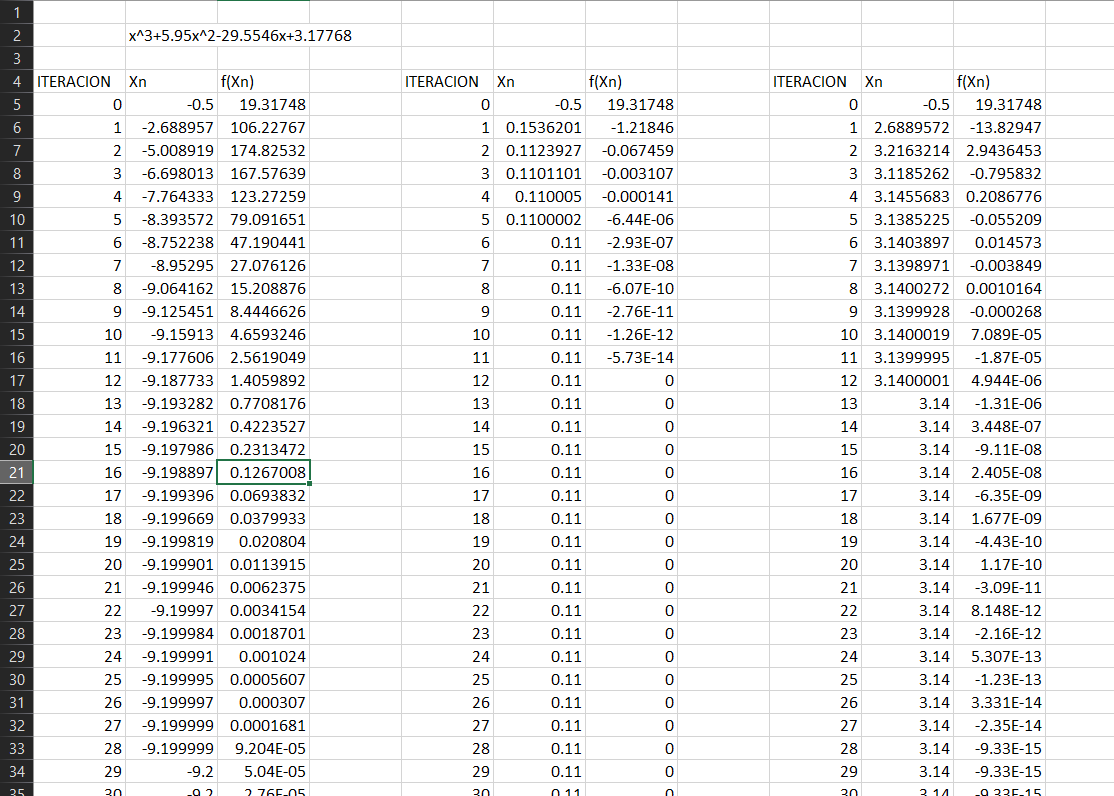
### 2



### 3



### 4



### 5

