INSTITUTO TECNOLÓGICO SUPERIOR DEL ORIENTE DEL ESTADO DE HIDALGO

Ingeniería en Sistemas Computacionales

T1-2 Problemario

Materia:

Métodos Numéricos

Docente:

Efren Rolando Romero Leon

Fecha de Entrega:

04 de febrero de 2026.

Nombre del alumno:

Cristian Abram Vera Franco

Errores de Programación

1.- Error de Redondeo Binario

El error más común es intentar representar números decimales exactos (como `0.1`) en un sistema binario. Esto causa que operaciones simples devuelvan resultados inesperados.

- **Ejemplo:** `System.out.println(0.1 + 0.2);` imprime `0.30000000000000004` en lugar de `0.3`.
- **Causa:** Fracciones decimales no tienen una representación binaria finita.

```java
public class ErrordeRedondeoBinario
{
    public static void main(String[] args) {
        System.out.println(0.1+0.2);
    }
}
```

Versión de Python

```python
print(0.1+0.2)
```

Resultado:

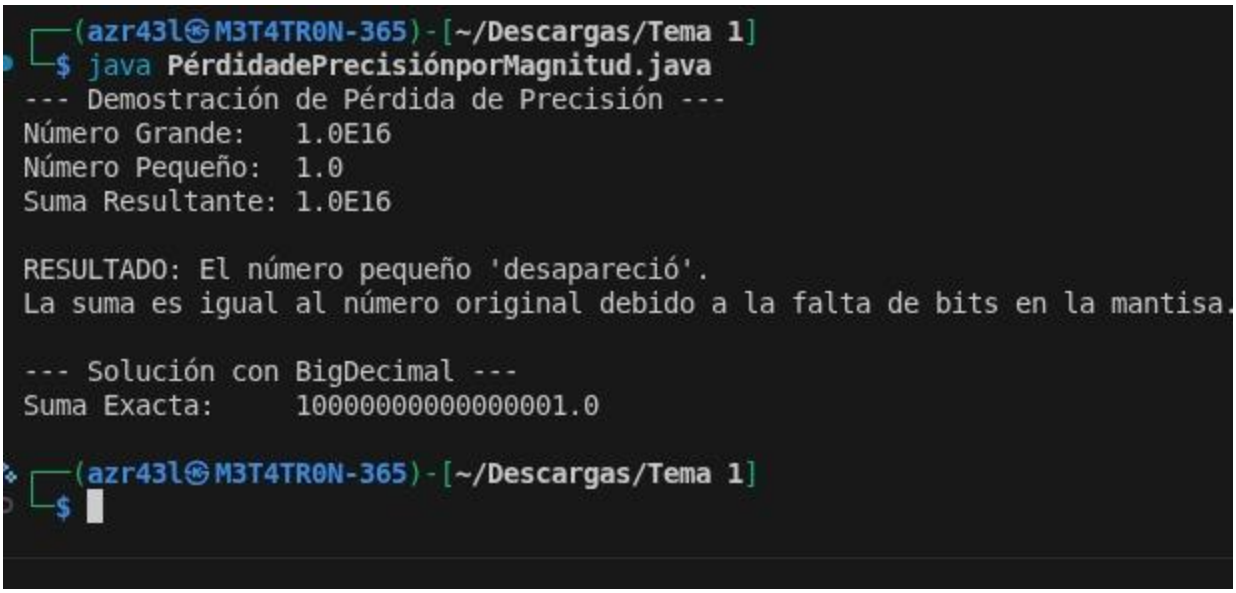
```
PS C:\Users\Abram\Downloads\Tema_1_MN\Tema 1> & C:/Users/Abram/
0.30000000000000004
PS C:\Users\Abram\Downloads\Tema_1_MN\Tema 1>
```

2. Pérdida de Precisión por Magnitud(IEEE 754)

Cuando se operan números con magnitudes muy diferentes, el número más pequeño puede "desaparecer" debido a que no cabe en la mantisa del número más grande.

- **Ejemplo:** Sumar `1.0` a un `double` extremadamente grande puede no cambiar el valor original.

```
public class PérdidadePrecisiónporMagnitud
{
     public static void main(String[] args) {
        // Un double tiene aproximadamente 15-17 dígitos significativos de
precisión
        double numeroGrande = 1.0e16; // 1 seguido de 16 ceros
        double numeroPequeno = 1.0;

        double resultado = numeroGrande + numeroPequeno;

        System.out.println("--- Demostración de Pérdida de Precisión ---");
        System.out.println("Número Grande:   " + numeroGrande);
        System.out.println("Número Pequeño:  " + numeroPequeno);
        System.out.println("Suma Resultante: " + resultado);

        // Verificación lógica
        if (resultado == numeroGrande) {
            System.out.println("\nRESULTADO: El número pequeño 'desapareció'.");
            System.out.println("La suma es igual al número original debido a la
falta de bits en la mantisa.");
        }

        // --- SOLUCIÓN USANDO BIGDECIMAL ---
        java.math.BigDecimal bdGrande = new java.math.BigDecimal("1.0e16");
        java.math.BigDecimal bdPequeno = new java.math.BigDecimal("1.0");
        java.math.BigDecimal bdResultado = bdGrande.add(bdPequeno);

        System.out.println("\n--- Solución con BigDecimal ---");
        System.out.println("Suma Exacta:     " + bdResultado.toPlainString());
    }

}
```




```
// --- SOLUCIÓN USANDO BIGDECIMAL ---
      java.math.BigDecimal bdGrande = new java.math.BigDecimal("1.0e16");
      java.math.BigDecimal bdPequeno = new java.math.BigDecimal("1.0");
      java.math.BigDecimal bdResultado = bdGrande.add(bdPequeno);

      System.out.println("\n--- Solución con BigDecimal ---");
      System.out.println("Suma Exacta:     " + bdResultado.toPlainString());
   }
}
```

Versión de Python

```python
from decimal import Decimal, getcontext

# Un float en Python tiene ~15-17 dígitos significativos
numero_grande = 1.0e16  # 1 seguido de 16 ceros
numero_pequeno = 1.0

resultado = numero_grande + numero_pequeno

print("--- Demostración de Pérdida de Precisión ---")
print("Número Grande:  ", numero_grande)
print("Número Pequeño: ", numero_pequeno)
print("Suma Resultante:", resultado)

# Verificación lógica
if resultado == numero_grande:
    print("\nRESULTADO: El número pequeño 'desapareció'.")
    print("La suma es igual al número original debido a la falta de bits en la mantisa.")
```

```
PS C:\Users\Abram\Downloads\Tema_1_MN\Tema 1> & C:/Users/Abram/AppData/Local/Microsof
--- Demostración de Pérdida de Precisión ---
Número Grande:   1e+16
Número Pequeño:  1.0
Suma Resultante: 1e+16

RESULTADO: El número pequeño 'desapareció'.
La suma es igual al número original debido a la falta de bits en la mantisa.
PS C:\Users\Abram\Downloads\Tema_1_MN\Tema 1>
```

Comparación Directa con ==

Debido a los errores de redondeo mencionados, comparar dos números `double` usando == suele fallar.

- **Solución:** Se debe usar un margen de error (épsilon).

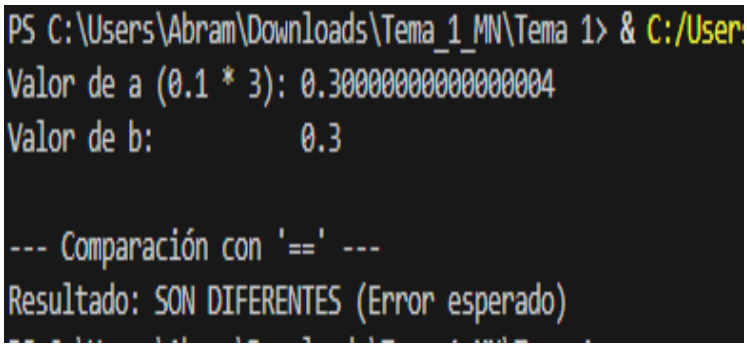
```
if (Math.abs(a - b) < 0.00001) { /* son iguales */ }
```

```
public class ComparaciónDirecta
{
    public static void main(String[] args) {
        // 1. El Problema: Error de redondeo en aritmética de punto flotante
        double a = 0.1 + 0.1 + 0.1;
        double b = 0.3;

        System.out.println("Valor de a (0.1 * 3): " + a);
        System.out.println("Valor de b:           " + b);

        // 2. Intento de comparación directa (FALLARÁ)
        System.out.println("\n--- Comparación con '==' ---");
        if (a == b) {
            System.out.println("Resultado: Son iguales");
        } else {
            System.out.println("Resultado: SON DIFERENTES (Error esperado)");
        }

        // 3. La Solución: Uso de un margen de error (Épsilon)
        double epsilon = 0.00001; // Definimos la tolerancia

        System.out.println("\n--- Comparación con Épsilon (" + epsilon + ") ---
");
        if (Math.abs(a - b) < epsilon) {
            System.out.println("Resultado: Son iguales (dentro del margen de
error)");
        } else {
            System.out.println("Resultado: Son diferentes");
        }
    }
}
```

```
┌──(azr43l㉿M3T4TR0N-365)-[~/Descargas/Tema 1]
└─$ java ComparaciónDirecta.java
Valor de a (0.1 * 3): 0.30000000000000004
Valor de b:           0.3

--- Comparación con '==' ---
Resultado: SON DIFERENTES (Error esperado)

--- Comparación con Épsilon (1.0E-5) ---
Resultado: Son iguales (dentro del margen de error)
```

Versión de Python

```python
import math

# 1. El problema: error de redondeo en punto flotante
a = 0.1 + 0.1 + 0.1
b = 0.3

print("Valor de a (0.1 * 3):", a)
print("Valor de b:          ", b)

# 2. Intento de comparación directa (FALLA)
print("\n--- Comparación con '==' ---")
if a == b:
    print("Resultado: Son iguales")
else:
    print("Resultado: SON DIFERENTES (Error esperado)")
```



4. Acumulación de Errores en Bucles

Realizar miles de operaciones aritméticas consecutivas con `double` acumula pequeños errores de redondeo que pueden resultar en una desviación significativa al final del proceso.

```
import java.math.BigDecimal;

public class AcumulacionErroresBucle{
    public static void main(String[] args) {
        int iteraciones = 1000000; // Un millón de sumas
        double incremento = 0.1;

        // 1. Acumulación usando 'double'
        double sumaDouble = 0.0;
        for (int i = 0; i < iteraciones; i++) {
            sumaDouble += incremento;
        }

        // El resultado esperado debería ser exactamente 100,000.0
        double esperado = iteraciones * incremento;

        System.out.println("--- Acumulación en Bucle (1,000,000 de iteraciones) -
--");
        System.out.println("Resultado esperado: " + esperado);
        System.out.println("Resultado double:   " + sumaDouble);
        System.out.println("Diferencia (Error): " + (sumaDouble - esperado));

        // 2. Solución usando 'BigDecimal' para precisión absoluta
        BigDecimal sumaBD = BigDecimal.ZERO;
        BigDecimal incrementoBD = new BigDecimal("0.1");

        for (int i = 0; i < iteraciones; i++) {
            sumaBD = sumaBD.add(incrementoBD);
        }

        System.out.println("\n--- Solución con BigDecimal ---");
        System.out.println("Resultado exacto:   " + sumaBD.toPlainString());

        // 3. Verificación de error
        if (sumaDouble != esperado) {
            System.out.println("\nNOTA: El error de 'double' es notable tras un
millón de sumas.");
        }
    }

}
```

```
┌──(azr43l㉿M3T4TR0N-365)-[~/Descargas/Tema 1]
└─$
┌──(azr43l㉿M3T4TR0N-365)-[~/Descargas/Tema 1]
└─$ java AcumulacionErroresBucle.java
--- Acumulación en Bucle (1,000,000 de iteraciones) ---
Resultado esperado: 100000.0
Resultado double:   100000.00000133288
Diferencia (Error): 1.3328826753422618E-6

--- Solución con BigDecimal ---
Resultado exacto:   100000.0

NOTA: El error de 'double' es notable tras un millón de sumas.

┌──(azr43l㉿M3T4TR0N-365)-[~/Descargas/Tema 1]
└─$
```

Versión de Python

```python
from decimal import Decimal, getcontext

iteraciones = 1_000_000  # Un millón de sumas
incremento = 0.1

# 1. Acumulación usando 'float'
suma_float = 0.0
for _ in range(iteraciones):
    suma_float += incremento

# Resultado esperado
esperado = iteraciones * incremento

print("--- Acumulación en Bucle (1,000,000 de iteraciones) ---")
print("Resultado esperado:", esperado)
print("Resultado float:   ", suma_float)
print("Diferencia (Error):", suma_float - esperado)
```

r

```
PS C:\Users\Abram\Downloads\Tema_1_MN\Tema 1> & C:/Users/Abram/App
--- Acumulación en Bucle (1,000,000 de iteraciones) ---
Resultado esperado: 100000.0
Resultado float:    100000.00000133288
Diferencia (Error): 1.3328826753422618e-06
PS C:\Users\Abram\Downloads\Tema_1_MN\Tema 1>
```

Cancelación por Resta (Loss of Significance)

Ocurre cuando restas dos números muy cercanos entre sí. La mayoría de los dígitos significativos se cancelan, dejando solo el error de redondeo como el "resultado" aparente.

java

```java
public class CancelacionPorResta {
    public static void main(String[] args) {
        // Dos números muy grandes y muy cercanos
        double x = 1234567890.1234567;
        double y = 1234567890.1234560;

        // El resultado esperado es 0.0000007
        double resultado = x - y;

        System.out.println("Resultado real: " + resultado);
        // En 2026 notarás que el resultado es 0.0 o un valor basura
        // debido a que los últimos dígitos se perdieron al almacenar x e y.
    }
}
```

```
┌──(azr43l㉿M3T4TR0N-365)-[~/Descargas/Tema 1]
└─$ java CancelaciónporResta.java
Resultado real: 7.1525573730468 75E-7
```

Versión de Python

```python
# Dos números muy grandes y muy cercanos
x = 1234567890.1234567
y = 1234567890.1234560

# El resultado esperado es 0.0000007
resultado = x - y

print("Resultado real:", resultado)
```

```
PS C:\Users\Abram\Downloads\Tema_1_MN\Tema 1> & C:/Users/Abram
Resultado real: 7.1525573730468 75e-07
PS C:\Users\Abram\Downloads\Tema_1_MN\Tema 1>
```

Desbordamiento Silencioso (Overflow)

A diferencia de los errores de decimales, este ocurre con tipos enteros (`int`, `long`). Si superas el valor máximo, Java no lanza una excepción; simplemente "da la vuelta" al número más pequeño (negativo), lo que destruye la exactitud del cálculo.

java

```java
public class DesbordamientoEntero {
    public static void main(String[] args) {
        int max = Integer.MAX_VALUE; // 2,147,483,647
        int resultado = max + 1;

        System.out.println("Máximo int: " + max);
        System.out.println("Máximo + 1: " + resultado); // Imprime -
2147483648

        // Solución: Usar Math.addExact para lanzar una excepción si ocurre
        try {
            Math.addExact(max, 1);
        } catch (ArithmeticException e) {
            System.out.println("Error detectado: ¡Desbordamiento!");
        }
    }
}
```

```
┌──(azr43l㉿M3T4TR0N-365)-[~/Descargas/Tema 1]
└─$ java DesbordamientoSilencioso.java
Máximo int: 2147483647
Máximo + 1: -2147483648
Error detectado: ¡Desbordamiento!
```

Versión de Python

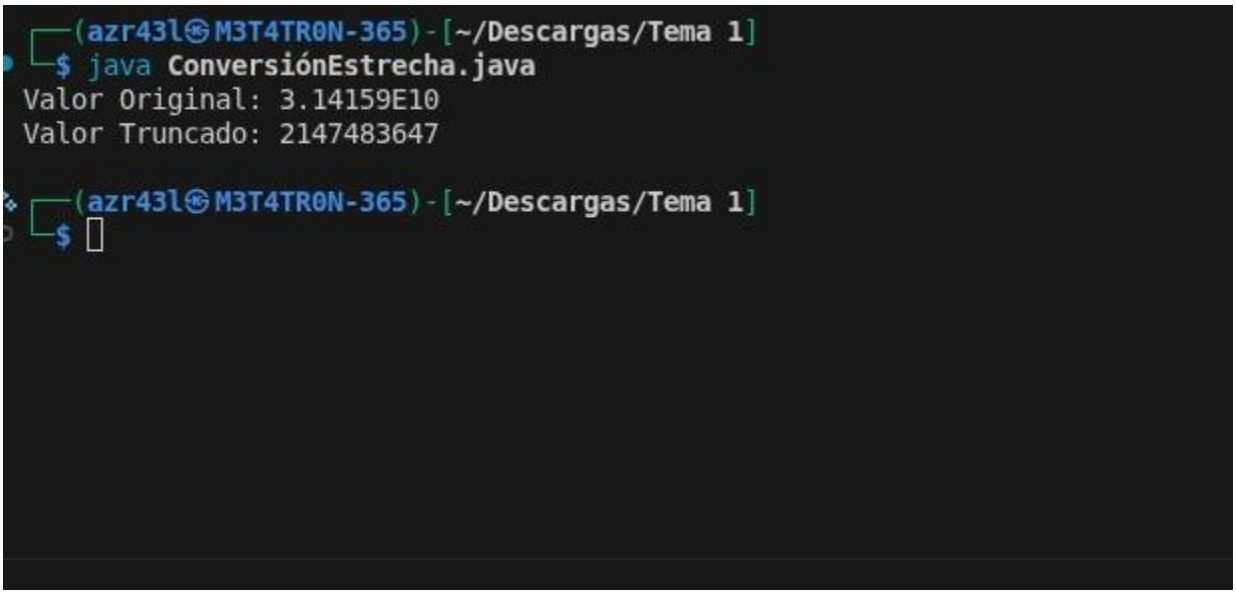
NO hay desbordamiento

Conversión Estrecha (Narrowing Primitive Conversion)

Al convertir un tipo de mayor capacidad a uno menor (de `double` a `int` o de `long` a `int`), Java simplemente trunca los bits sobrantes, lo que puede cambiar drásticamente el valor original sin previo aviso.

Java

```java
public class ErrorConversion {
    public static void main(String[] args) {
        double valorDouble = 3.14159e10; // Un número muy grande
        int valorInt = (int) valorDouble; // Casting explícito

        System.out.println("Valor Original: " + valorDouble);
        System.out.println("Valor Truncado: " + valorInt);
        // Imprime 2147483647 (el máximo int) porque el double no cabe.
    }
}
```



Se salvan los demas lenguajes o APP

R: no

Veamos

EXCEL

La operación =`100,39 - 100,35` puede no dar exactamente `0,04` en la memoria interna de Excel, aunque la celda parezca mostrarlo correctamente.

Pruebalo

Si usas ese resultado en un `=SI(A1=0,04; "OK"; "ERROR")`, podría devolver "ERROR".

Violación de la Propiedad Asociativa

En matemáticas puras, (a + b) + c siempre es igual a a + (b + c). En la computación de punto flotante, esto no es verdad. El orden de las operaciones altera el resultado debido a cuándo se aplica el redondeo.

```python
# Definimos tres números
a = 1e30  # Un número enorme
b = -1e30 # El mismo número enorme pero negativo
c = 1.0   # Un número pequeño

# Caso 1: Sumamos los grandes primero
# (1e30 - 1e30) = 0.0, luego 0.0 + 1.0 = 1.0
resultado_1 = (a + b) + c

# Caso 2: Sumamos el negativo grande con el pequeño primero
# (-1e30 + 1.0) sigue siendo -1e30 porque el 1.0 se pierde por falta de precisión
# Luego 1e30 + (-1e30) = 0.0
resultado_2 = a + (b + c)

print(f"Resultado 1: {resultado_1}") # Imprime 1.0
print(f"Resultado 2: {resultado_2}") # Imprime 0.0 (¡Error!)
print(f"¿Son iguales?: {resultado_1 == resultado_2}")
```

```
PS C:\Users\Abram\Downloads\Tema_1_MN\Tema 1> & C:/Users/Abram/A
0.py"
Resultado 1: 1.0
Resultado 2: 0.0
¿Son iguales?: False
PS C:\Users\Abram\Downloads\Tema_1_MN\Tema 1>
```

Subdesbordamiento (Floating Point Underflow)

Existe el **Underflow** en flotantes. Ocurre cuando un número es tan pequeño (tan cercano a cero) que la computadora ya no puede representarlo y simplemente lo convierte en **cero absoluto**, perdiendo toda la información de su magnitud.

```python
# El flotante positivo más pequeño posible antes de llegar a cero
x = 1e-300
y = 1e-100

print(f"Valor inicial de x: {x}")

# Dividimos x por un número muy grande
resultado = x / y
resultado_extremo = x / 1e100

print(f"División normal: {resultado}")
# Aquí ocurre el Underflow: el resultado debería ser 1e-400,
# pero es demasiado pequeño para Python (float de 64 bits)
print(f"Underflow (se vuelve cero): {resultado_extremo}")
```

```
PS C:\Users\Abram\Downloads\Tema_1_MN\Tema 1> & C:/Us
0.py"
Valor inicial de x: 1e-300
División normal: 1e-200
Underflow (se vuelve cero): 0.0
PS C:\Users\Abram\Downloads\Tema_1_MN\Tema 1>
```

La Trampa de NaN (Not a Number)

NaN es un valor especial estándar (IEEE 754) que resulta de operaciones imposibles (como 0/0 o inf - inf). El error común aquí es intentar comparar NaN. Por definición, **NaN no es igual a nada, ni siquiera a sí mismo**.

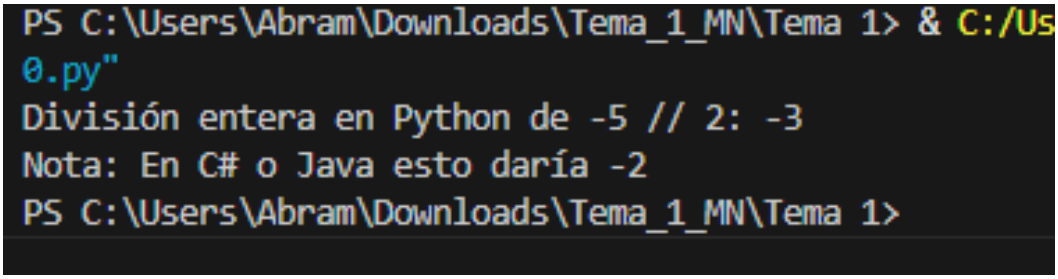
```python
import math

# Generamos un NaN
val_nan = float('nan')

print(f"Valor: {val_nan}")

# Intento de comparación lógica estándar
if val_nan == val_nan:
    print("Son iguales")
else:
    print("¡Error de lógica! NaN no es igual a NaN")

# Forma correcta de verificar
if math.isnan(val_nan):
    print("Forma correcta: Se detectó un NaN")
```

```
PS C:\Users\Abram\Downloads\Tema_1_MN\Tema 1> & C:
0.py"
Valor: nan
¡Error de lógica! NaN no es igual a NaN
Forma correcta: Se detectó un NaN
PS C:\Users\Abram\Downloads\Tema_1_MN\Tema 1>
```

División de Enteros Negativos (Diferencia de Comportamiento)

Este es un error lógico-aritmético. En matemáticas, la división puede ser ambigua con negativos. Python usa "floor division" (redondea hacia $-\infty$), mientras que lenguajes como C, Java o C# usan "truncation" (redondean hacia 0). Si migras algoritmos de C# a Python, esto romperá tu código.

```python
# En Java/C#: -5 / 2 daría -2 (quita el decimal .5)
# En Python: -5 // 2 da -3 (baja al siguiente entero menor)

numerador = -5
denominador = 2

resultado = numerador // denominador

print(f"División entera en Python de -5 // 2: {resultado}")
print("Nota: En C# o Java esto daría -2")
```



Comparación entre Decimal y Float

Un error nuevo es intentar compararlos directamente o mezclarlos sin cuidado. Un float ya trae "basura" de precisión, y al convertirlo a Decimal, esa basura se hace visible.

```python
from decimal import Decimal

# Un float simple
f = 1.1
# Un Decimal creado desde string (precisión perfecta)
d_string = Decimal("1.1")
# Un Decimal creado desde el float (hereda el error del float)
d_float = Decimal(f)

print(f"Float original: {f}")
print(f"Decimal desde string: {d_string}")
print(f"Decimal desde float:  {d_float}") # ¡Mira todos los decimales extra!

# Error de comparación
if d_string == d_float:
    print("Son iguales")
else:
    print("Son diferentes (El Decimal desde float cargó el error binario)")
```

```
PS C:\Users\Abram\Downloads\Tema_1_MN\Tema 1> & C:/Users/Abram/AppData/Local/
0.py"
Float original: 1.1
Decimal desde string: 1.1
Decimal desde float:  1.100000000000000088817841970012523233890533447265625
Son diferentes (El Decimal desde float cargó el error binario)
PS C:\Users\Abram\Downloads\Tema_1_MN\Tema 1>
```

Error Absoluto y relativo

El error **absoluto** es una medida de la diferencia entre el valor verdadero y el valor aproximado de un cálculo numérico. Representa la magnitud del desvío o la imprecisión en el resultado obtenido. Se calcula restando el valor verdadero al valor aproximado. El error absoluto puede ser positivo o negativo, dependiendo de si el valor aproximado está por encima o por debajo del valor verdadero. Cuanto menor sea el valor del error absoluto, mayor será la precisión de la aproximación.

$$|Valor_{real} - Valor_{aproximado}|$$

El error **relativo** es una medida relativa de la precisión de un cálculo numérico. Se calcula dividiendo el error absoluto entre el valor verdadero y se expresa generalmente como un valor decimal o porcentual. El error relativo permite comparar la magnitud del error con respecto al tamaño del valor verdadero. Proporciona una forma de evaluar la exactitud relativa de una aproximación y es particularmente útil cuando se trabaja con números de diferentes magnitudes. Un error relativo más pequeño indica una aproximación más precisa.

$$\frac{E_a}{|Valor_{real}|} \times 100\%$$

Ejercicios realizados en clase:

Doña Laura

Se ha estimado que en el monedero de doña Laura tenia 160 monedas de oro, pero al contarlo su nieta se dan cuenta que solamente hay 156.

Calcular el erro absoluto y el error relativo

(156-160) = 4

(4/160) = 2.5 * 100 = %2.5

valor absoluto = 4

valor relativo = %2.5

CNC

El margen de error de una máquina CNC normalmente se expresa como una tolerancia del tipo ±(algún valor en mm o pulgadas); a partir de esa tolerancia puedes obtener el error absoluto y el error relativo de la máquina para una dimensión concreta.

Error absoluto:diferencia entre el valor real (o nominal del plano) y el valor medido o fabricado.

$$E_t = \frac{error\ verdadero}{valor\ verdadero}$$

Error relativo:cociente entre el error absoluto y el valor real (o nominal), normalmente en porcentaje.

$$E_t = \frac{error\ verdadero}{valor\ verdadero}\ 100\%$$

|

Margen de error típico de una CNC (tolerancia)

● Servicios CNC genéricos suelen trabajar, si no se especifica nada, con

tolerancias estándar alrededor de ±0,1 mm.

● Máquinas CNC de alta precisión pueden llegar a ±0,0025 mm (±0,0001").

● Guías comunes indican tolerancias típicas como:

- ● Torno CNC: ≈ ±0,005" (≈ ±0,13 mm).
- ● Fresado CNC 3 ejes: valores típicos del orden de ±0,005" a ±0,01"

  dependiendo del proceso.

Supón que la máquina tiene una tolerancia típica de ±0,1 mm al mecanizar una longitud nominal de 50 mm.

1. Tolerancia de la máquina (margen de error): ±0,1 mm.

2. Error absoluto máximo: 0,1 mm (es la desviación permitida respecto al valor nominal).

3. Error relativo=0.1/ 50×100%=0,2%

En contexto de una CNC, si quieres hablar del “margen” propio de la máquina sin aún medir nada, su error absoluto máximo es simplemente su tolerancia:

- Si la CNC tiene tolerancia de ±0,1 mm, el error absoluto máximo de la máquina es 0,1 mm.

Supón una cota nominal de 50 mm y una CNC con tolerancia ±0,1 mm:

- Error absoluto máximo de la máquina: 0,1 mm (ese es su margen).
- Si una pieza sale en 49,94 mm, entonces:
    - Error absoluto real de esa pieza:
    - |50−49,94|=0,06 mm

Ejemplo 1: El problema de la escala (El Puente vs. El Remache)

Este ejemplo demuestra por qué el error absoluto por sí solo es engañoso. Imagina que tienes un error de medición de **1 cm** en dos situaciones diferentes.

Situación A: Midiendo un puente

- **Valor Real:** 10,000 cm 100 m
- **Valor Medido:** $10,001 cm
- **E_a:** |10000 - 10001| = 1 cm
- **E_r:** $\frac{1}{10000} = 0.0001 \rightarrow 0.01\%$
    - *Conclusión:* Un error despreciable. El puente sigue siendo funcional.

Situación B: Midiendo un remache (tornillo)

- Valor Real: 3 cm
- Valor Medido: 4cm
- $Ea: |3 - 4| = 1\ cm$
- $Er: \frac{1}{3} = 0.333 \rightarrow 33.3\%$
    - *Conclusión:* Un error catastrófico. El remache no entrará en el agujero.

Ejemplo 2: Aproximación Numérica (Cálculo de π)

Supongamos que en un programa de Python usas una aproximación simple de π en lugar de la librería math.

- **Valor Real** (π): $\mathbf{3.141592}$ ....
- **Valor Aproximado:** 3.14

1. **Error Absoluto:**

   $$| 3.141592 - 3.14 | = 0.001592$$

   (Te faltaron 0.001592 unidades)

2. **Error Relativo:**

   $$\frac{0.001592}{3.141592} \approx 0.000506 \rightarrow 0.05\%$$

   (Tu cálculo tiene una precisión del 99.95%)

Ejemplo 3: Finanzas (Contabilidad)

Imagina que proyectaste un presupuesto para tu agencia de marketing.

- **Ingreso Real:** $50,000$
- **Ingreso Pronosticado:** $48,500$

1. **Error Absoluto:**

$$|50000 – 48500| = \$1,500$$

(Te equivocaste por mil quinientos pesos)

2. **Error Relativo:**

$$\frac{1500}{50000} = 0.03 \rightarrow 3\%$$