

INGENIERÍA MECATRÓNICA



DI_CERO

DIEGO CERVANTES RODRÍGUEZ

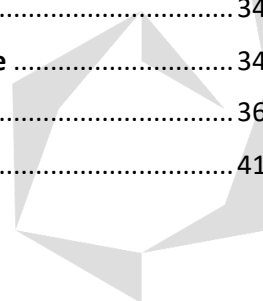
INSTRUMENTACIÓN VIRTUAL

PYTHON 3.9.7, C# & LABVIEW

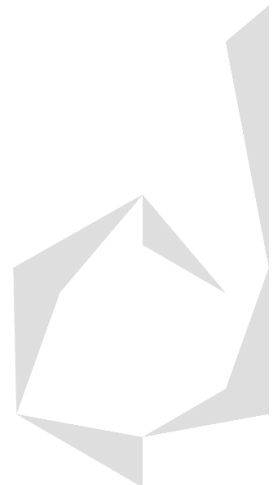
Ejercicios de Funciones y
Pruebas Unitarias (Testing)

Contenido

Teoría – Funciones :	3
Teoría – Pruebas Unitarias (Unit Testing) :	3
Dependencias:	4
Ejercicios – Funciones :	5
1.- Área de un Triángulo Arbitrario: Cálculo con las coordenadas de sus 3 vértices	5
Código Python – Visual Studio Code (Logo Azul)	6
Resultado del Código Python	8
Código C# (.NET Framework) – Visual Studio (Logo Morado)	8
Resultado del Código C#	9
2.- Calcular la Longitud de una Ruta: Cálculo con (x_0, y_0) , ..., (x_n, y_n) y Manual Testing	9
Código Python – Visual Studio Code (Logo Azul)	10
Resultado del Código Python	13
Código C# (.NET Framework) – Visual Studio (Logo Morado)	13
Resultado del Código C#	14
3.- Aproximación de π: Utilizando el programa anterior de Cálculo de Longitud de Ruta	14
Código Python – Visual Studio Code (Logo Azul)	14
Resultado del Código Python	18
Código C# (.NET Framework) – Visual Studio (Logo Morado)	18
Resultado del Código C#	20
4.- Series de Fourier: Aproximar una Función Escalonada con una suma de senos/cosenos	20
Código Python – Visual Studio Code (Logo Azul)	21
Resultado del Código Python	24
Código C# (.NET Framework) – Visual Studio (Logo Morado)	25
Resultado del Código C#	26
5.- Campana de Gauss: Implementar una función Gaussiana	26
Código Python – Visual Studio Code (Logo Azul)	28
Resultado del Código Python	32
Código C# (.NET Framework) – Visual Studio (Logo Morado)	33
Resultado del Código C#	34
6.- Pruebas Unitarias: Librerías de pruebas distintas/ Testing: Unittesting y Nose	34
Código Python – Visual Studio Code (Logo Azul)	36
Resultado del Código Python: Pruebas Unittest y Nose aprobadas	41



Resultado del Código Python: Prueba Unittest aprobada y Nose reprobada	41
Resultado del Código Python: Pruebas Unittest y Nose reprobadas, Error por usar 2 librerías ...	42
7.- Diferenciación numérica: Derivada c/ la fórmula diferencial/Testing: Assertion y Nose	42
Código Python – Visual Studio Code (Logo Azul)	43
Resultado del Código Python: Pruebas Assertion y Nose aprobadas	51
Resultado del Código Python: Prueba Assertion aprobada y Nose reprobada.....	51
Código C# (.NET Framework) – Visual Studio (Logo Morado)	52
Resultado del Código C#	53



Teoría – Funciones:

Las funciones en python se indican a través de la palabra reservada **def**, seguida del nombre de la función, un paréntesis que contiene sus distintos parámetros y dos puntos para denotar su inicio, en Python no se utilizan llaves de apertura o cierre para su sintaxis, solamente se utilizan dos puntos para indicar su inicio y tabuladores para diferenciar qué es lo que está dentro o fuera de ella.

- Las funciones son muy utilizadas en Python ya que sirven para volver modular el código, logrando así que se pueda ejecutar varias veces una misma acción sin necesidad de escribirla varias veces.
- Los distintos métodos que se utilizan en todos los lenguajes de programación para ejecutar acciones diferentes como por ejemplo lo es el método `print()`, son en realidad funciones, pero que utilizan la arquitectura de programación orientada a objetos (POO).
- Los parámetros que recibe una función son las variables con las que interactúa para realizar su acción, una función cualquiera puede o no recibir parámetros para ejecutarse.
- Una función puede o no devolver una variable que almacene su valor resultante, esto se realiza a través de la palabra reservada **return**.
- Para utilizar una función en Python se debe declarar su nombre seguido de un paréntesis en donde se le pase sus parámetros, si es que recibe algunos, además su resultado puede o no guardarse en una variable dependiendo de si retorna un valor o no.

Teoría – Pruebas Unitarias (Unit Testing):

Una prueba unitaria, también conocida como test unitario o **unit testing**, es una técnica de pruebas en el desarrollo de software que se enfoca en verificar el comportamiento y la funcionalidad de una unidad de código de manera aislada. Con “unidad de código” nos referimos a la parte más pequeña del software que se puede probar de forma independiente, como una función, un método o una clase.

Estas pruebas unitarias se centran en probar una única funcionalidad o una pequeña porción de código a la vez, sin tener en cuenta la interacción con otras partes del sistema, a esto se le llama principio de aislamiento. Para lograr esto, es común utilizar técnicas como el reemplazo de **dependencias** por objetos simulados o falsos (mocking) y el uso de configuraciones controladas.

Las pruebas unitarias generalmente siguen una estructura básica que incluye los siguientes pasos:

1. **Preparación (Setup):** Se crean los objetos necesarios para ejecutar la prueba unitaria y se configuran las condiciones iniciales para la prueba de la unidad de código (función, método o clase).
2. **Ejecución (Execution):** Se llama a la unidad de código y se le proporcionan los datos de entrada (parámetros) necesarios.
3. **Verificación (Assertion):** Se realizan aserciones para comprobar si los resultados de la unidad de código son los esperados. Si las aserciones fallan, significa que la unidad de código no se comporta como se espera y se registra un error.
 - a. **Aserción, afirmación o assertion:** Es una expresión booleana que se utiliza para verificar una condición y afirmar que debe ser verdadera en un punto específico del código. La

aserción se utiliza para asegurarse de que ciertas condiciones esperadas se cumplan durante la ejecución de un programa.

- i. Si la condición booleana es verdadera (True), la ejecución del programa continúa sin problemas.
 - ii. Si la condición booleana es falsa (False), se produce una excepción o error de aserción, lo que indica que la suposición o afirmación falló.
4. **Limpieza (Teardown):** Se realiza cualquier limpieza necesaria después de la prueba, como liberar recursos o restablecer el estado.
- a. Con algunas herramientas de testing, después de este estado se realiza un reporte del comportamiento de la unidad de código, pero esto depende enteramente de la librería o framework de testing que se esté utilizando.

Las pruebas unitarias son esenciales en el desarrollo de software porque ofrecen varios beneficios:

- **Detectar errores tempranos:** Las pruebas unitarias permiten encontrar y corregir errores de manera rápida y precisa, lo que evita que se propaguen y se conviertan en problemas más graves en etapas posteriores del desarrollo.
- **Mejorar la calidad del código:** Al probar unidades de código de forma aislada, se fomenta la escritura de código más modular, legible y mantenible.
- **Facilitar la refactorización:** Si el código continúa aprobando todas las pruebas después de una refactorización, se tiene mayor confianza en que el comportamiento del sistema no se haya visto afectado negativamente.
 - **Refactorización:** Es el proceso de modificar el código fuente de un programa para mejorar su estructura interna y hacerlo más legible, comprensible y mantenible, sin cambiar su comportamiento externo. Consiste en reorganizar, simplificar y optimizar el código existente sin agregar nuevas funcionalidades ni modificar el resultado final del programa.
- **Documentar el comportamiento esperado:** Las pruebas unitarias proporcionan una documentación del comportamiento esperado de cada unidad de código, incluyendo el tiempo de ejecución, número de fallas, etc. lo que facilita la comprensión, formas de optimización y mantenimiento del programa a largo plazo.

Dependencias

En el contexto del desarrollo de software, una **dependencia** se refiere a la relación entre dos componentes o módulos donde uno de ellos requiere del otro para funcionar correctamente. Es decir, un componente depende de otro para poder realizar su funcionalidad o para acceder a ciertos recursos o funcionalidades proporcionadas por ese otro componente.

Las dependencias pueden manifestarse de diferentes maneras en el desarrollo de software:

- **Dependencia de código:** Un componente puede depender de otro componente a través de su código fuente. Esto significa que el primer componente utiliza funciones, clases, métodos u otros elementos definidos en el segundo componente para realizar ciertas tareas.
- **Dependencia de librería:** Un componente puede depender de una librería o biblioteca, que es un conjunto de código predefinido y reutilizable. La dependencia se establece cuando el

componente utiliza las funciones, clases o métodos proporcionados por la librería para llevar a cabo ciertas operaciones.

- **Dependencia de módulo:** En algunos lenguajes de programación, el código se organiza en módulos o paquetes. Una dependencia de módulo se da cuando un módulo depende de otro módulo para acceder a su funcionalidad o recursos.
- **Dependencia de tiempo de ejecución:** Un componente puede depender de otros componentes o servicios que se deben ejecutar y estar disponibles en tiempo de ejecución. Estas dependencias se resuelven durante la ejecución del programa y pueden ser gestionadas a través de inyección de dependencias u otros mecanismos.

Las dependencias son necesarias para aprovechar la funcionalidad proporcionada por otros componentes o para cumplir con requisitos específicos.

Sin embargo, es importante gestionar adecuadamente las dependencias para evitar que se creen duplicados innecesarios o acoplamientos excesivos y así mantener la modularidad, flexibilidad y capacidad de mantenimiento del sistema. Para ello, se utilizan técnicas como la inyección de dependencias y el uso de contenedores de control para gestionar las dependencias utilizadas en un programa de manera eficiente y reducir el acoplamiento entre componentes.

Ejercicios – Funciones:

1.- Área de un Triángulo Arbitrario: Cálculo con las coordenadas de sus 3 vértices

Un triángulo arbitrario puede describirse mediante las coordenadas de sus tres vértices: (x_1, y_1) , (x_2, y_2) , (x_3, y_3) , numerados en sentido contrario a las agujas del reloj. El área del triángulo viene dada por la fórmula:

$$A = \frac{1}{2} |x_1 y_2 - x_1 y_3 - x_2 y_1 + x_2 y_3 + x_3 y_1 - x_3 y_2|$$

Escribe una función `area(vértices)` que devuelva el área de un triángulo cuyos vértices estén especificados por el argumento `vértices`, que es una lista anidada con las coordenadas de los vértices.

Pseudocódigo:

1. Por ejemplo, el cálculo del área del triángulo con las coordenadas de vértice $(0, 0)$, $(1, 0)$, $(0, 2)$ se realiza mediante la siguiente instrucción:

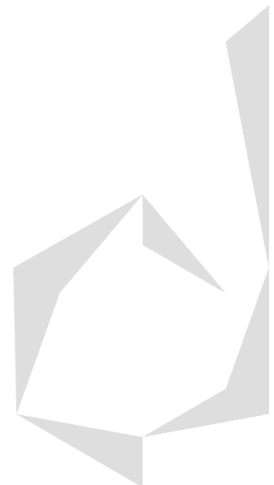
triangle1 = area([0,0], [1,0], [0,2])

2. O bien:

v1 = (0,0); v2 = (1,0); v3 = (0,2)

vertices = [v1,v2,v3]

triangle1 = area(vertices)



Código Python – Visual Studio Code (Logo Azul)

```
# -*- coding: utf-8 -*-

#En Python se introducen comentarios de una sola linea con el simbolo #.
#La primera línea de código incluida en este programa se conoce como declaración de codificación o codificación
#de caracteres. Al especificar utf-8 (caracteres Unicode) como la codificación, nos aseguramos de que el archivo
#pueda contener caracteres especiales, letras acentuadas y otros caracteres no ASCII sin problemas, garantizando
#que Python interprete correctamente esos caracteres y evite posibles errores de codificación.
#Se puede detener una ejecución con el comando [CTRL] + C puesto en consola, con el comando "cls" se borra su
#historial y en Visual Studio Code con el botón superior izquierdo de Play se corre el programa.
#Para comentar en Visual Studio Code varias líneas de código se debe pulsar:
#[CTRL] + K (VSCode queda a la espera). Después pulsa [CTRL] + C para comentar y [CTRL] + U para descomentar.

#FUNCIONES EN PYTHON: Las funciones en python se indican a través de la palabra reservada def, seguida del
#nombre de la función, un paréntesis que contiene sus distintos parámetros y dos puntos para denotar su inicio,
#en Python no se utilizan llaves de apertura o cierre para su sintaxis, solamente se utilizan dos puntos para
#indicar su inicio y tabuladores para diferenciar qué es lo que está dentro o fuera de ella.
# - Las funciones son muy utilizadas en Python ya que sirven para volver modular el código, logrando así que
#   se pueda ejecutar varias veces una misma acción sin necesidad de escribirla varias veces.
# - Los distintos métodos que se utilizan en todos los lenguajes de programación para ejecutar acciones
#   diferentes como por ejemplo lo es el método print(), son en realidad funciones, pero que utilizan una
#   arquitectura de programación orientada a objetos (POO).
# - Los parámetros que recibe una función son las variables con las que interactúa para realizar su acción,
#   una función cualquiera puede o no recibir parámetros para ejecutarse.
# - Una función puede o no devolver una variable que almacene su valor resultante, esto se realiza a través
#   de la palabra reservada "return".
# - Para utilizar una función en Python se debe declarar su nombre seguido de un paréntesis en donde se le
#   pase sus parámetros, si es que recibe algunos, además su resultado puede o no guardarse en una variable
#   dependiendo de si retorna un valor o no.

#1.- ÁREA DE UN TRIÁNGULO ARBITRARIO: Cálculo con las coordenadas de sus 3 vértices
#Un triángulo arbitrario puede describirse mediante las coordenadas de sus tres vértices: (x1, y1), (x2, y2),
#(x3, y3), numerados en sentido contrario a las agujas del reloj. El área del triángulo viene dada por la
#fórmula:
#A = (1/2)*|x1*y2 - x1*y3 - x2*y1 + x2*y3 + x3*y1 - x3*y2|
#PRIMERO SE CREARÁ UNA FUNCIÓN QUE SE USA PARA DENOTAR DE FORMA MÁS SENCILLA LA FÓRMULA SIN USAR LISTAS:
def areaTriangulo(x1,y1,x2,y2,x3,y3):
    #str(): Método que convierte un tipo de dato cualquiera en numérico decimal.
    x1 = float(x1) #v1 = (x1, y1)
    y1 = float(y1)
    x2 = float(x2) #v2 = (x2, y2)
    y2 = float(y2)
    x3 = float(x3) #v3 = (x3, y3)
    y3 = float(y3)
```

```

#abs(): Método que saca el valor absoluto de un número, volviéndolo positivo aunque sea negativo
#A = (1/2)*|x1*y2 - x1*y3 - x2*y1 + x2*y3 + x3*y1 - x3*y2|
Area = abs((1/2)*(x1*y2 - x1*y3 - x2*y1 + x2*y3 + x3*y1 - x3*y2))
return Area

#USO DE LA PRIMERA FUNCIÓN SIN LISTAS:
#input(): Método que sirve para imprimir en consola un mensaje y que luego se permita al usuario ingresar un
#valor, que será de tipo String y podrá ser almacenado en una variable.
#Coordenadas verticales y horizontales del primer vértice del triángulo
print('-----Función 1: Sin Listas-----')
x1 = input('Introduzca la coordenada x del primer vértice del triángulo: \t')    #v1 = (x1, y1)
y1 = input('Introduzca la coordenada y del primer vértice del triángulo: \t')
#Coordenadas verticales y horizontales del segundo vértice del triángulo
x2 = input('Introduzca la coordenada x del segundo vértice del triángulo: \t')    #v2 = (x2, y2)
y2 = input('Introduzca la coordenada y del segundo vértice del triángulo: \t')
#Coordenadas verticales y horizontales del tercer vértice del triángulo
x3 = input('Introduzca la coordenada x del tercer vértice del triángulo: \t')    #v3 = (x3, y3)
y3 = input('Introduzca la coordenada y del tercer vértice del triángulo: \t')

Resultado = areaTriangulo(x1,y1,x2,y2,x3,y3)

#print(): Método para imprimir un mensaje en consola y después dar un salto de línea (Enter), además si se
#quiere concatenar un mensaje (mostrar resultados de variables junto con texto estático), este se debe separar
#entre comillas, declarando los mensajes estáticos entre comillas y los nombres de variables sin comillas.
print('El área del triangulo con las coordenadas \n'
      , '(' , x1 , ',' , y1 , ')' , '(' , x2 , ',' , y2 , ')' y ' ' , '(' , x3 , ',' , y3 , ')'
      , '\nEs igual a : \n'
      , Resultado)

#LISTAS: Las listas en Python son tipos de datos estructurados, parecido a lo que son los arrays en otros
#lenguajes de programación, aunque no es el único tipo de dato agrupado que existe en Python, existen además las
#tuplas, diccionarios y numpy arrays. A las listas también se les puede llamar vectores.
#FUNCIÓN QUE UTILIZA LA FÓRMULA USANDO LISTAS:
def areaListas(v1, v2, v3):
    #A = (1/2)*|x1*y2 - x1*y3 - x2*y1 + x2*y3 + x3*y1 - x3*y2|
    a = (1/2) * abs(v1[0]*v2[1] - v1[0]*v3[1] - v2[0]*v1[1] + v2[0]*v3[1] + v3[0]*v1[1] - v3[0]*v2[1])
    return a

#USO DE LA SEGUNDA FUNCIÓN CON LISTAS:
vertice_1 = (0, 0)    #v1 = (x1, y1)
vertice_2 = (1, 0)    #v2 = (x2, y2)

```



```

vertice_3 = (0, 2) #v3 = (x3, y3)

triangle1 = areaListas(vertice_1, vertice_2, vertice_3)

#print(): Método para imprimir un mensaje en consola y después dar un salto de línea (Enter), además si se
#quiere concatenar un mensaje (mostrar resultados de variables junto con texto estático), este se debe separar
#entre comillas, declarando los mensajes estáticos entre comillas y los nombres de variables sin comillas.
print('-----Función 2: Con Listas-----')
print("El área del triángulo compuesto por los vértices \n"
      , vertice_1
      , vertice_2
      , vertice_3
      , "Es igual a: \n"
      , triangle1)

```

Resultado del Código Python

```

PROBLEMAS  SALIDA  TERMINAL  CONSOLA DE DEPURACIÓN

PS C:\Users\diego\OneDrive\Documents\Aprendiendo\Python\1.-Instrumentación Virtual> & D:/Users/diego/AppData/Local/Programs/Python/Python39/python.exe
"c:/Users/diego/OneDrive/Documents/Aprendiendo/Python/1.-Instrumentación Virtual/Instrumentación Virtual Aplicada/Tareas/Tarea 1/Ejercicio 3.11.py"
-----Función 1: Sin Listas-----
Introduzca la coordenada x del primer vértice del triángulo: 0
Introduzca la coordenada y del primer vértice del triángulo: 0
Introduzca la coordenada x del segundo vértice del triángulo: 1
Introduzca la coordenada y del segundo vértice del triángulo: 0
Introduzca la coordenada x del tercer vértice del triángulo: 0
Introduzca la coordenada y del tercer vértice del triángulo: 2
El área del triángulo con las coordenadas
( 0 , 0 ), ( 1 , 0 ) y ( 0 , 2 )
Es igual a :
1.0
-----Función 2: Con Listas-----
El área del triángulo compuesto por los vértices
(0, 0) (1, 0) (0, 2) Es igual a:
1.0
PS C:\Users\diego\OneDrive\Documents\Aprendiendo\Python\1.-Instrumentación Virtual>

```

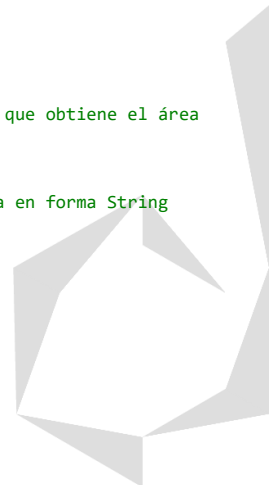
Código C# (.NET Framework) – Visual Studio (Logo Morado)

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace _9._1._Área_Triángulo_con_Vértices
{
    class AreaTrianguloArbitrario
    {
        //El código en C# se corre presionando CTRL + F5
        //EJERCICIO TAREA 3.11 ÁREA DE UN TRIÁNGULO ARBITRARIO:
        static void Main(string[] args)
        {
            //Declaración de las variables double (número decimal) para poder hacer la operación que obtiene el área
            float x1, x2, x3, y1, y2, y3;
            //Coordenadas verticales y horizontales del primer vértice del triángulo
            Console.WriteLine("Introduzca la coordenada x del primer vértice del triángulo: \t");
            x1 = float.Parse(Console.ReadLine()); //Conversión a double de lo que viene de consola en forma String
            Console.WriteLine("Introduzca la coordenada y del primer vértice del triángulo: \t");
            y1 = float.Parse(Console.ReadLine());
            //Coordenadas verticales y horizontales del segundo vértice del triángulo
            Console.WriteLine("Introduzca la coordenada x del segundo vértice del triángulo: \t");
            x2 = float.Parse(Console.ReadLine());
            Console.WriteLine("Introduzca la coordenada y del segundo vértice del triángulo: \t");
            y2 = float.Parse(Console.ReadLine());
            //Coordenadas verticales y horizontales del tercer vértice del triángulo

```



```

        Console.WriteLine("Introduzca la coordenada x del tercer vértice del triángulo: \t");
        x3 = float.Parse(Console.ReadLine());
        Console.WriteLine("Introduzca la coordenada y del tercer vértice del triángulo: \t");
        y3 = float.Parse(Console.ReadLine());

        double Resultado = areaTriangulo(x1, y1, x2, y2, x3, y3);
        Console.WriteLine("El área del triángulo con las coordenadas ({0}, {1}), ({2}, {3}) y ({4}, {5}) es de: {6}", x1,
            y1, x2, y2, x3, y3, Resultado);
    }

    public static float areaTriangulo(float x1, float y1, float x2, float y2, float x3, float y3)
    {
        float Area;
        /*El método abs de la clase Math saca el valor absoluto de la operación realizada en su paréntesis, que
        sirve para obtener el área de un triángulo cualquiera*/
        Area = (float)0.5 * Math.Abs(x2 * y3 - x3 * y2 - x1 * y3 + x3 * y1 + x1 * y2 - x2 * y1);
        return Area;
    }
}

```

Resultado del Código C#

```

C:\WINDOWS\system32\cmd.  X  +  v
Introduzca la coordenada x del primer vértice del triángulo: 0
Introduzca la coordenada y del primer vértice del triángulo: 0
Introduzca la coordenada x del segundo vértice del triángulo: 1
Introduzca la coordenada y del segundo vértice del triángulo: 0
Introduzca la coordenada x del tercer vértice del triángulo: 0
Introduzca la coordenada y del tercer vértice del triángulo: 2
El área del triángulo con las coordenadas (0, 0), (1, 0) y (0, 2) es de: 1
Presione una tecla para continuar . . . |

```

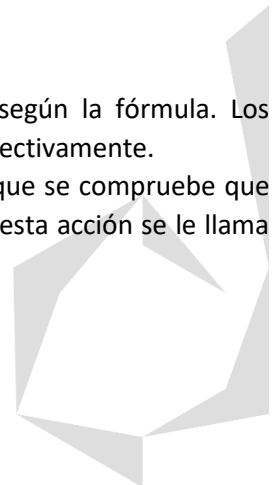
2.- Calcular la Longitud de una Ruta: Cálculo con (x_0, y_0) , ..., (x_n, y_n) y Manual Testing

Un objeto se mueve a lo largo de una trayectoria en el plano. En $n + 1$ puntos de tiempo se han registrado las correspondientes posiciones (x, y) del objeto: (x_0, y_0) , (x_1, y_1) , ..., (x_n, y_n) . La longitud total L del camino de (x_0, y_0) a (x_n, y_n) es la suma de todos los segmentos de línea individuales $((x_{i-1}, y_{i-1})$ a (x_i, y_i) , donde $i = 1, \dots, n$):

$$L = \sum_{i=1}^n \sqrt{(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2}$$

Pseudocódigo:

- Crea una función Python **pathlength(x, y)** para calcular la longitud L según la fórmula. Los argumentos (x, y) contienen las coordenadas (x_0, \dots, x_n) y (y_0, \dots, y_n) , respectivamente.
- Manual Testing:** Escribe una función de prueba **test_pathlength()** en la que se compruebe que **pathlength()** devuelve la longitud correcta en un problema de prueba. A esta acción se le llama testing.



Código Python – Visual Studio Code (Logo Azul)

```
# -*- coding: utf-8 -*-

#En Python se introducen comentarios de una sola linea con el simbolo #.
#La primera línea de código incluida en este programa se conoce como declaración de codificación o codificación
#de caracteres. Al especificar utf-8 (caracteres Unicode) como la codificación, nos aseguramos de que el archivo
#pueda contener caracteres especiales, letras acentuadas y otros caracteres no ASCII sin problemas, garantizando
#que Python interprete correctamente esos caracteres y evite posibles errores de codificación.
#Se puede detener una ejecución con el comando [CTRL] + C puesto en consola, con el comando "cls" se borra su
#historial y en Visual Studio Code con el botón superior izquierdo de Play se corre el programa.
#Para comentar en Visual Studio Code varias líneas de código se debe pulsar:
#[CTRL] + K (VSCode queda a la espera). Después pulsa [CTRL] + C para comentar y [CTRL] + U para descomentar.

#FUNCIONES EN PYTHON: Las funciones en python se indican a través de la palabra reservada def, seguida del
#nombre de la función, un paréntesis que contiene sus distintos parámetros y dos puntos para denotar su inicio,
#en Python no se utilizan llaves de apertura o cierre para su sintaxis, solamente se utilizan dos puntos para
#indicar su inicio y tabuladores para diferenciar qué es lo que está dentro o fuera de ella.
# - Las funciones son muy utilizadas en Python ya que sirven para volver modular el código, logrando así que
# se pueda ejecutar varias veces una misma acción sin necesidad de escribirla varias veces.
# - Los distintos métodos que se utilizan en todos los lenguajes de programación para ejecutar acciones
# diferentes como por ejemplo lo es el método print(), son en realidad funciones, pero que utilizan una
# arquitectura de programación orientada a objetos (POO).
# - Los parámetros que recibe una función son las variables con las que interactúa para realizar su acción,
# una función cualquiera puede o no recibir parámetros para ejecutarse.
# - Una función puede o no devolver una variable que almacene su valor resultante, esto se realiza a través
# de la palabra reservada "return".
# - Para utilizar una función en Python se debe declarar su nombre seguido de un paréntesis en donde se le
# pase sus parámetros, si es que recibe algunos, además su resultado puede o no guardarse en una variable
# dependiendo de si retorna un valor o no.

#2.- LONGITUD DE UN CAMINO: Cálculo con sus coordenadas (x1, y1), ..., (xn, yn)
#a) FUNCIÓN PATHLENGTH: Calcula la longitud de una ruta según la fórmula L.
#Un objeto se mueve a lo largo de una trayectoria en el plano. En n + 1 puntos de tiempo se han registrado las
#correspondientes posiciones (x, y) del objeto: (x0, y0), (x1, y1), ..., (xn, yn). La longitud total L del
#camino de (x0, y0) a (xn, yn) es la suma de todos los segmentos de línea individuales
#((xi-1, yi-1) a (xi, yi), donde i = 1, ..., n):
#L =  $\sum \sqrt{(x(i) - x(i-1))^2 + (y(i) - y(i-1))^2}$ ; donde i = 1,..., n.
def longitudRuta(x, y):
    Longitud = 0    #Longitud inicial = 0

    #BUCLE FOR: En Python no se utilizan llaves de apertura o cierre para la sintaxis de un bucle, solamente se
    #utilizan dos puntos para indicar su inicio y tabuladores para diferenciar qué es lo que está dentro o fuera
    #de él. Además, después de la palabra reservada "for" se declara una variable local numérica entera que solo
    #existirá dentro del bucle y será la que cuente desde 0 por default o desde el primer número indicado dentro
    #del paréntesis hasta el extremo indicado en el segundo parámetro que se encuentra dentro del paréntesis de
```

```

#la palabra reservada range() para terminar el bucle. En otras palabras, dentro del paréntesis de la palabra
#reservada "range()" se coloca el inicio y final del conteo para indicar cuantas veces se ejecutará el
#bucle.

#Es importante mencionar que los bucles for se pueden utilizar para realizar sumatorias en operaciones
#matemáticas:

#L =  $\sum (x(i)-x(i-1))^2 + (y(i)-y(i-1))^2$ ; donde  $i = 1, \dots, n$ .

#len(): Este método sirve para ver el tamaño de una lista, tupla, diccionario o numpy array.
for i in range(1, len(x)):
    #float(): Método que convierte un tipo de dato cualquiera en numérico decimal.
    #numero ** potencia = numero ^ potencia = número elevado a cierta potencia.
    #numero ** 1/2 = numero ^ 1/2 =  $\sqrt{\text{número}}$ .
    L = (((float(x[i]) - float(x[i-1]))**2)) + ((float(y[i]) - float(y[i-1]))**2))**(1/2)
    #Se utiliza una variable intermedia llamada L para calcular cada una de las longitudes y luego sumarlas
    #a la variable Longitud que almacenará el valor de la longitud total.
    Longitud += L
return Longitud

#input(): Método que sirve para imprimir en consola un mensaje y que luego se permita al usuario ingresar un
#valor, que será de tipo String y podrá ser almacenado en una variable.
#int(): Método que convierte un tipo de dato cualquiera en numérico entero.
num_coordenadas = int(input('Indique el número de coordenadas del camino (igual o mayor que 2)\n'))

#Declaración de dos listas vacías donde se almacenarán todos los puntos de las coordenadas de la ruta
x_total = []      #Puntos horizontales (x1), ..., (xn) de las coordenadas.
y_total = []      #Puntos verticales (y1), ..., (yn) de las coordenadas.
coordenadas = []  #Lista que almacenará todas las coordenadas ingresadas para imprimirlas en consola.

#CONDICIONAL IF: En Python no se utilizan llaves de apertura o cierre al utilizar condicionales, solamente se
#utilizan dos puntos para indicar el inicio del condicional y tabuladores para ver qué es lo que está dentro o
#fuera de él, ya sea para el condicional if, else if (elif) o else.
if num_coordenadas < 2:
    print('Un camino no puede tener solo una coordenada, debe ser mínimo de 2 o más')
else:
    #Dependiendo del número de coordenadas que haya introducido el usuario, se ejecutará varias veces el bucle
    #for para que el usuario las introduzca todas:
    for i in range(0, num_coordenadas):
        #input(): Método que sirve para imprimir en consola un mensaje y que luego se permita al usuario
        #ingresar un valor, que será de tipo String y podrá ser almacenado en una variable.
        #int(): Método que convierte un tipo de dato cualquiera en numérico entero.
        x_nuevo = int(input(f'Ingrese la coordenada horizontal del punto x{i}: \t'))
        y_nuevo = int(input(f'Ingrese la coordenada vertical del punto y{i}: \t'))

        #print(): Método para imprimir un mensaje en consola y después dar un salto de línea (Enter).
        print("La coordenada ", i, " es:\n", "(", x_nuevo, ", ", y_nuevo, ")")

```

```

#append(): Método que sirve para agregar valores a una lista, tupla, numpy array o diccionario.
x_total.append(x_nuevo)
y_total.append(y_nuevo)

#LISTAS ANIDADAS: Cuando dentro de la posición de una lista se encuentra otra lista interna, se le
#llama lista anidada, esto se realiza por ejemplo para categorizar datos, realizar operaciones
#matriciales, etc.

#AGREGAR LISTAS ANIDADAS CON EL MÉTODO lista.append(): Esto se debe realizar usualmente dentro de un
#bucle for, en este caso para ello primero se extraen los valores de "x_nuevo" y "y_nuevo" y luego se
#agregan a la lista de coordenadas como una lista anidada [x_nuevo, y_nuevo] con el método append().
coordenadas.append([x_nuevo, y_nuevo])

#USO DE LA FUNCIÓN longitudRuta(): Terminando el bucle for pero dentro del condicional if se utiliza la
#función que realiza el cálculo de la distancia total con todos los puntos de las coordenadas de la ruta.
Longitud_total = longitudRuta(x_total, y_total)

#print(): Método para imprimir un mensaje en consola y después dar un salto de línea (Enter), además si se
#quiere concatenar un mensaje (mostrar resultados de variables junto con texto estático), este se debe
#separar entre comillas, declarando los mensajes estáticos entre comillas y los nombres de variables sin
#comillas.
print("Las coordenadas ingresadas fueron: ", coordenadas)
print('La distancia total de la ruta con', num_coordenadas, 'coordenadas tiene una longitud de:', Longitud_total)

#TESTING: Acción realizada cuando a través de una función se comprueba el funcionamiento de otra.
#b) FUNCIÓN TEST_PATHLENGTH: Función que ejecuta la función PATHLENGTH para comprobar que se ejecuta
#correctamente, a esta acción se le llama testing.
def test_pathLenght():
    coordenada_1 = (0, 0)
    coordenada_2 = (1, 0)
    coordenada_3 = (2, 0)

    #nombreLista[index]: Para acceder a la posición de una lista se debe indicar su nombre seguido de unos
    #corchetes que indiquen la coordenada a la que se quiere acceder. Se debe tomar en cuenta que las coordenadas
    #se empiezan a contar desde 0, aunque el tamaño de la lista se cuenta desde 1.
    x_total = (coordenada_1[0], coordenada_2[0], coordenada_3[0])
    y_total = (coordenada_1[1], coordenada_2[1], coordenada_3[1])

    Longitud_total = longitudRuta(x_total, y_total)

    print('Testing: La distancia total de la ruta con', len(x_total), 'coordenadas tiene una longitud de:', Longitud_total)

#USO DE LA FUNCIÓN test_pathLenght().
test_pathLenght()

```

Resultado del Código Python

```
PROBLEMAS  SALIDA  TERMINAL  CONSOLA DE DEPURACIÓN

PS C:\Users\diego\OneDrive\Documents\Aprendiendo\Python\1.-Instrumentación Virtual> & D:/Users/diego/AppData/Local/Programs/Python/Python39/python.exe
"C:/Users/diego/OneDrive/Documents/Aprendiendo/Python/1.-Instrumentación Virtual/Instrumentación Virtual Aplicada/Tareas/Tarea 1/Tareas C#/Ej3_12.py"
Indique el número de coordenadas del camino (igual o mayor que 2)
3
Ingrese la coordenada horizontal del punto x0: 0
Ingrese la coordenada vertical del punto y0: 0
La coordenada 0 es:
(0, 0)
Ingrese la coordenada horizontal del punto x1: 1
Ingrese la coordenada vertical del punto y1: 0
La coordenada 1 es:
(1, 0)
Ingrese la coordenada horizontal del punto x2: 2
Ingrese la coordenada vertical del punto y2: 0
La coordenada 2 es:
(2, 0)
Las coordenadas ingresadas fueron: [[0, 0], [1, 0], [2, 0]]
La distancia total de la ruta con 3 coordenadas tiene una longitud de: 2.0
Testing: La distancia total de la ruta con 3 coordenadas tiene una longitud de: 2.0
PS C:\Users\diego\OneDrive\Documents\Aprendiendo\Python\1.-Instrumentación Virtual>
```

Código C# (.NET Framework) – Visual Studio (Logo Morado)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace _9._2._Longitud_de_una_Ruta
{
    class LongitudRutaArbitraria
    {
        //El código en C# se corre presionando CTRL + F5
        //EJERCICIO TAREA 3.12 LONGITUD DE UN CAMINO ARBITRARIO:

        //Función para obtener la longitud de una ruta cualquiera:
        public static double longitudRuta(float[] x, float[] y)
        {
            double Longitud = 0, L = 0;

            //L =  $\sum (x_i - x_{i-1})^2 + (y_i - y_{i-1})^2$ 
            for (int i = 1; i < x.Length; i++) //Bucle for para realizar la sumatoria de los elementos
            {
                L = Math.Sqrt(Math.Pow((x[i] - x[i - 1]), 2) + Math.Pow((y[i] - y[i - 1]), 2));
                Longitud += L;
            }
            return Longitud;
        }

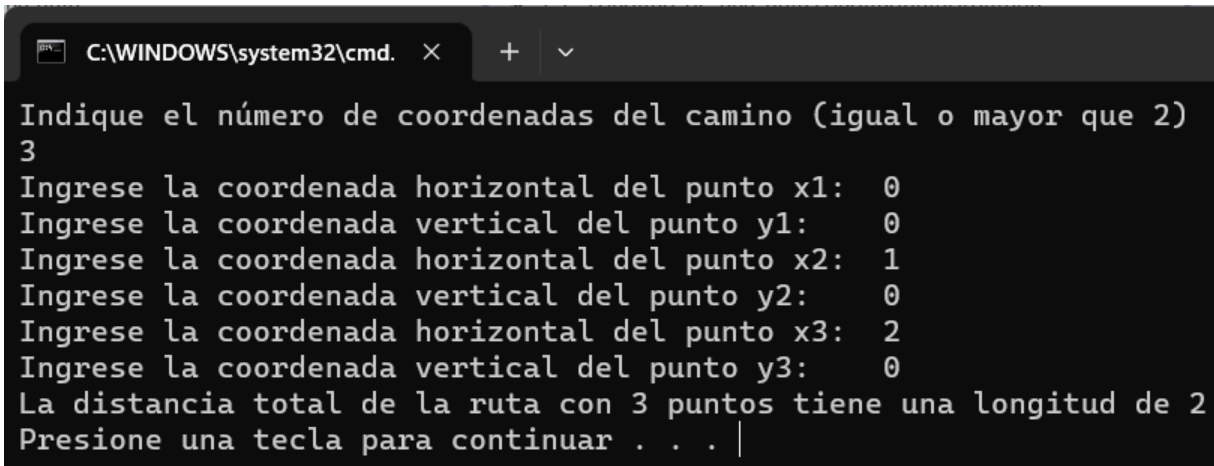
        static void Main(string[] args)
        {
            Console.WriteLine("Indique el número de coordenadas del camino (igual o mayor que 2)");
            //Conversión a int del valor proveniente de la consola en forma de String
            int num_coordenadas = Convert.ToInt32(Console.ReadLine());
            //Condicional para evaluar si el número de coordenadas es mayor o igual a 2
            if (num_coordenadas < 2)
            {
                Console.WriteLine("Un camino no puede tener solo una coordenada, deben ser mínimo más de 2");
            }
            else
            {
                //Declaración de dos arrays vacíos tipo double donde se almacenarán todos los puntos de las coordenadas
                //de la ruta
                float[] x = new float[num_coordenadas];
                float[] y = new float[num_coordenadas];
                //Dependiendo del número de coordenadas que haya introducido el usuario, se ejecutará varias veces el
                //bucle para que el usuario las introduzca todas
                for (int i = 0; i < num_coordenadas; i++)
                {
                    Console.Write("Ingrese la coordenada horizontal del punto x{0}: \t", i + 1);
                    x[i] = float.Parse(Console.ReadLine());
                    Console.Write("Ingrese la coordenada vertical del punto y{0}: \t", i + 1);
                    y[i] = float.Parse(Console.ReadLine());
                }
            }
        }
    }
}
```

```

        double Longitud_total = longitudRuta(x, y);
        Console.WriteLine("La distancia total de la ruta con {0} puntos tiene una longitud de {1}", num_coordenadas,
Longitud_total);
    }
}
}
}

```

Resultado del Código C#



```

C:\WINDOWS\system32\cmd. x + v
Indique el número de coordenadas del camino (igual o mayor que 2)
3
Ingrese la coordenada horizontal del punto x1: 0
Ingrese la coordenada vertical del punto y1: 0
Ingrese la coordenada horizontal del punto x2: 1
Ingrese la coordenada vertical del punto y2: 0
Ingrese la coordenada horizontal del punto x3: 2
Ingrese la coordenada vertical del punto y3: 0
La distancia total de la ruta con 3 puntos tiene una longitud de 2
Presione una tecla para continuar . . . |

```

3.- Aproximación de π : Utilizando el programa anterior de Cálculo de Longitud de Ruta

El valor de π es igual a la circunferencia de un círculo de radio $r = \frac{1}{2}$. Supongamos que aproximamos la forma de una circunferencia mediante un polígono que pasa por $n + 1$ puntos del círculo, la longitud de este polígono se puede hallar usando la función de Cálculo de Longitud de Ruta del ejercicio 2, calculando la ruta de $n + 1$ puntos consecutivos con coordenadas (x_i, y_i) a lo largo de una circunferencia con radio $r = \frac{1}{2}$ según las siguientes fórmulas, considerando que $n = 2^k$, donde k determina la precisión de la aproximación del polígono. A medida que la variable "k" aumente, el número de puntos utilizados y la cantidad de segmentos rectos aumenta, lo que resulta en una aproximación más precisa de la circunferencia, reduciendo exponencialmente el error entre el valor real de π y la aproximación obtenida:

$$x_i = \frac{1}{2} \cos\left(\frac{2\pi \cdot i}{n}\right), \quad y_i = \frac{1}{2} \sin\left(\frac{2\pi \cdot i}{n}\right), \quad i = 0, \dots, n$$

Llama a la función longitud de recorrido y escribe el error de la aproximación de π para $n = 2^k$, donde $k = 2, 3, \dots, 10$.

Código Python – Visual Studio Code (Logo Azul)

```

# -*- coding: utf-8 -*-

#En Python se introducen comentarios de una sola línea con el símbolo #.
#La primera línea de código incluida en este programa se conoce como declaración de codificación o codificación

```

```

#de caracteres. Al especificar utf-8 (caracteres Unicode) como la codificación, nos aseguramos de que el archivo
#pueda contener caracteres especiales, letras acentuadas y otros caracteres no ASCII sin problemas, garantizando
#que Python interprete correctamente esos caracteres y evite posibles errores de codificación.

#Se puede detener una ejecución con el comando [CTRL] + C puesto en consola, con el comando "cls" se borra su
#historial y en Visual Studio Code con el botón superior izquierdo de Play se corre el programa.

#Para comentar en Visual Studio Code varias líneas de código se debe pulsar:
#[CTRL] + K (VSCode queda a la espera). Después pulsa [CTRL] + C para comentar y [CTRL] + U para descomentar.

import math #math: Librería que proporciona funciones y constantes matemáticas como seno, π, logaritmo, etc.

#FUNCIONES EN PYTHON: Las funciones en python se indican a través de la palabra reservada def, seguida del
#nombre de la función, un paréntesis que contiene sus distintos parámetros y dos puntos para denotar su inicio,
#en Python no se utilizan llaves de apertura o cierre para su sintaxis, solamente se utilizan dos puntos para
#indicar su inicio y tabuladores para diferenciar qué es lo que está dentro o fuera de ella.

# - Las funciones son muy utilizadas en Python ya que sirven para volver modular el código, logrando así que
# se pueda ejecutar varias veces una misma acción sin necesidad de escribirla varias veces.
# - Los distintos métodos que se utilizan en todos los lenguajes de programación para ejecutar acciones
# diferentes como por ejemplo lo es el método print(), son en realidad funciones, pero que utilizan una
# arquitectura de programación orientada a objetos (POO).
# - Los parámetros que recibe una función son las variables con las que interactúa para realizar su acción,
# una función cualquiera puede o no recibir parámetros para ejecutarse.
# - Una función puede o no devolver una variable que almacene su valor resultante, esto se realiza a través
# de la palabra reservada "return".
# - Para utilizar una función en Python se debe declarar su nombre seguido de un paréntesis en donde se le
# pase sus parámetros, si es que recibe algunos, además su resultado puede o no guardarse en una variable
# dependiendo de si retorna un valor o no.

#2.- LONGITUD DE UNA RUTA: Cálculo con sus coordenadas (x1, y1), ..., (xn, yn)
#FUNCIÓN PATHLENGTH: Calcula la longitud de una ruta según la fórmula L.

#Un objeto se mueve a lo largo de una trayectoria en el plano. En n + 1 puntos de tiempo se han registrado las
#correspondientes posiciones (x, y) del objeto: (x0, y0), (x1, y1), ..., (xn, yn). La longitud total L del
#camino de (x0, y0) a (xn, yn) es la suma de todos los segmentos de línea individuales

#((xi-1, yi-1) a (xi, yi), donde i = 1, ..., n):
#L =  $\sum \sqrt{(x(i) - x(i-1))^2 + (y(i) - y(i-1))^2}$ ; donde i = 1,..., n.

#3.- APROXIMACIÓN DE π: Utilizando el programa de Cálculo de Longitud de Ruta.

#El valor de π es igual a la circunferencia de un círculo de radio r = 1/2. Supongamos que aproximamos la forma
#de una circunferencia mediante un polígono que pasa por n + 1 puntos del círculo, la longitud de este polígono
#se puede hallar usando la función de Cálculo de Longitud de Ruta del ejercicio 2, calculando la ruta de n + 1
#puntos consecutivos con coordenadas (xi, yi) a lo largo de una circunferencia de radio r = 1/2 según las
#siguientes fórmulas, considerando que n = 2^k, donde k determina la precisión de la aproximación del polígono.
#A medida que la variable k aumente, el número de puntos utilizados y la cantidad de segmentos rectos aumenta,
#lo que resulta en una aproximación más precisa de la circunferencia, reduciendo exponencialmente el error entre
#el valor real de π y la aproximación obtenida:

#xi = 1/2*cos(2*π*i/n) = 1/2*cos(2*π*i/(2^k)); i = n+1 = (2^k)+1

```



```
def longitudRuta(x, y):
    Longitud = 0                #Longitud inicial = 0

    #BUCLE FOR: En Python no se utilizan llaves de apertura o cierre para la sintaxis de un bucle, solamente se
    #utilizan dos puntos para indicar su inicio y tabuladores para diferenciar qué es lo que está dentro o fuera
    #de él. Además, después de la palabra reservada "for" se declara una variable local numérica entera que solo
    #existirá dentro del bucle y será la que cuente desde 0 por default o desde el primer número indicado dentro
    #del paréntesis hasta el extremo indicado en el segundo parámetro que se encuentra dentro del paréntesis de
    #la palabra reservada range() para terminar el bucle. En otras palabras, dentro del paréntesis de la palabra
    #reservada "range()" se coloca el inicio y final del conteo para indicar cuantas veces se ejecutará el
    #bucle.

    #Es importante mencionar que los bucles for se pueden utilizar para realizar sumatorias en operaciones
    #matemáticas:
    #L =  $\sum (x(i)-x(i-1))^2 + (y(i)-y(i-1))^2$ ; donde  $i = 1, \dots, n$ .
    #len(): Este método sirve para ver el tamaño de una lista, tupla, diccionario o numpy array.
    for i in range(1, len(x)):
        #float(): Método que convierte un tipo de dato cualquiera en numérico decimal.
        #numero ** potencia = numero ^ potencia = número elevado a cierta potencia.
        #numero ** 1/2 = numero ^ 1/2 =  $\sqrt{\text{número}}$ .
        L = (((float(x[i]) - float(x[i-1]))**2)) + ((float(y[i]) - float(y[i-1]))**2))**(1/2)
        #Se utiliza una variable intermedia llamada L para calcular cada una de las longitudes y luego sumarlas
        #a la variable Longitud que almacenará el valor de la longitud total.
        Longitud += L
    return Longitud

#OBTENCIÓN DE LAS COORDENADAS DE UN POLÍGONO QUE ASEMEJA LA FORMA DE UN CÍRCULO CON RADIO  $r = 1/2$ :
#Se declaran las variables n y k para que estos sean los puntos consecutivos con los que se aproxima un polígono
#para asemejar la forma de un círculo y obtener el valor aproximado de pi, en específico con estas dos variables
#se tendrán  $n = 2^k$  puntos en el polígono para acercarse al valor del perímetro del círculo con diámetro = 1 y
#radio = 1/2.
#n =  $2^k$  puntos consecutivos que conforman el polígono:
k = (2, 3, 4, 5, 6, 7, 8, 9, 10)    #Lista k: Precisión de la aproximación de un polígono a un círculo.
n = 2                                #Base n: Puntos consecutivos para aproximar un polígono a un círculo.

#Declaración de las listas vacías donde se almacenarán los valores de cada una de las aproximaciones con  $n = 2^k$ 
#vértices para obtener el valor de pi con un varios polígonos.
error = []                          #error: Almacena el error para cada una de las aproximaciones.
coordenadas_x = []                  #x: Vector que contiene todas las coordenadas horizontales del polígono.
coordenadas_y = []                  #y: Vector que contiene todas las coordenadas verticales del polígono.
circunferencias = []                #Perímetros: Circunferencias obtenidas con las coordenadas (xi, yi).

#Bucle for each: Es un tipo de bucle for especial que sirve para recorrer todos los elementos de una lista,
#tupla o diccionario.
```

```

#En este caso es lo mismo que poner la instrucción:
# - for i in k = for i in range(0, len(k)): Instrucción que recorre los elementos de la lista k, del número 2
#   al 10.
#En programación una forma de ejecutar exponentes es por medio de ciclos for anidados:
# - El bucle for que se encuentre en la parte exterior será el exponente.
#   En este caso es k.
# - El bucle for interno será la base del exponente.
#   En este caso es n, para que de esta forma se pueda hacer que  $n = 2^k$ .
for i in k:
    #Declaración de las listas vacías que almacenarán las coordenadas x,y de cada uno de los vértices del
    #polígono que se aproxima al círculo para obtener así el valor de pi, cuyo valor es igual al perímetro del
    #círculo con diámetro = 1.
    xi = []          #xi: Coordenadas horizontales de  $i = 0, \dots, n$ .
    yi = []          #yi: Coordenadas verticales de  $i = 0, \dots, n$ .
    #El bucle se tiene que ejecutar n+1 veces, pero como  $n = 2^k$ , se debe ejecutar  $(2^k)+1$  veces.
    for j in range(0, n**i + 1):
        #Por medio de las siguientes fórmulas se describe la forma en la que se obtendrán las coordenadas
        #del polígono que se aproxima a tener la forma de un círculo.
        #append(): Método que sirve para agregar valores a una lista, tupla, numpy array o diccionario.
        #math.cos(): Método de la librería math que aplica la función coseno a lo que haya en su paréntesis.
        #math.sin(): Método de la librería math que aplica la función seno a lo que haya en su paréntesis.
        #math.pi(): Constante pi ( $\pi$ ) de la librería math.
        xi.append(math.cos(2*math.pi*j/n**i)/2) #xi =  $1/2*\cos(2*\pi*i/n) = 1/2*\cos(2*\pi*i/(2^k))$ ;  $i = n+1 = (2^k)+1$ 
        yi.append(math.sin(2*math.pi*j/n**i)/2) #yi =  $1/2*\sin(2*\pi*i/n) = 1/2*\sin(2*\pi*i/(2^k))$ ;  $i = n+1 = (2^k)+1$ 

    #Se agregan las coordenadas xi al vector que contiene todas las coordenadas horizontales del polígono
    coordenadas_x.append(xi)
    #Se agregan las coordenadas yi al vector que contiene todas las coordenadas verticales del polígono
    coordenadas_y.append(yi)

#USO DE LA FUNCIÓN longitudRuta(): Se utiliza la función que realiza el cálculo de la distancia total de una
#ruta con todos los puntos de sus coordenadas, esto se realiza una vez obtenidas todas las coordenadas del
#polígono que se aproxima a tener la misma forma que un círculo con radio  $r = 1/2$ , obteniendo así el valor
#aproximado de  $\pi$ .
#En el siguiente bucle se analizan cada una de las coordenadas guardadas en las listas "coordenadas_x" y
#"coordenadas_y" para obtener el perímetro de ese polígono usando cada una de esas coordenadas y así obtener
#las distintas aproximaciones de pi.
for i in range(0, len(k)):#El bucle for se ejecutará el mismo número de veces que las potencias de precisión k.
    #Se llena la lista de circunferencias con los distintos perímetros obtenidos con las coordenadas (xi, yi)
    #calculadas en el bucle anterior, esto se realiza utilizando la función longitudRuta().
    circunferencias.append(longitudRuta(coordenadas_x[i], coordenadas_y[i]))

    #Se calcula el error de todas las circunferencias obtenidas restando el valor de pi obtenido de la librería

```

```
#math menos el valor obtenido con función longitudRuta().
error.append(math.pi-longitudRuta(coordenadas_x[i],coordenadas_y[i]))

#print(): Método para imprimir un mensaje en consola y después dar un salto de línea (Enter), además si se
#quiere concatenar un mensaje (mostrar resultados de variables junto con texto estático), este se debe
#separar entre comillas, declarando los mensajes estáticos entre comillas y los nombres de variables sin
#comillas.
print(f'Puntos del Polígono:', n**k[i])          #Impresión en pantalla del número de (2^k)+1 puntos obtenidos del polígono.
print(f'Circunferencia = Valor de pi obtenido con la aproximación:', circunferencias[i])
print(f'Error:', error[i])          #Impresión en pantalla del error obtenido con cada circunferencia calculada.
```

Resultado del Código Python

```
PROBLEMAS  SALIDA  TERMINAL  CONSOLA DE DEPURACIÓN
PS C:\Users\diego\OneDrive\Documents\Aprendiendo\Python\1.-Instrumentación Virtual> & D:\Users\diego\AppData\Local\Programs\Python\Python39\python.exe
"c:/Users/diego/OneDrive/Documents/Aprendiendo/Python/1.-Instrumentación Virtual/Instrumentación Virtual Aplicada/Tareas/Tarea 1/Tareas C#/Ej3_13.py"
Puntos del Polígono: 4
Circunferencia = Valor de pi obtenido con la aproximación: 2.82842712474619
Error: 0.31316552884360327
Puntos del Polígono: 8
Circunferencia = Valor de pi obtenido con la aproximación: 3.061467458920718
Error: 0.0801251946690753
Puntos del Polígono: 16
Circunferencia = Valor de pi obtenido con la aproximación: 3.1214451522580515
Error: 0.02014750133174159
Puntos del Polígono: 32
Circunferencia = Valor de pi obtenido con la aproximación: 3.1365484905459406
Error: 0.005044163043852468
Puntos del Polígono: 64
Circunferencia = Valor de pi obtenido con la aproximación: 3.1403311569547543
Error: 0.001261496635038828
Puntos del Polígono: 128
Circunferencia = Valor de pi obtenido con la aproximación: 3.14127725093278
Error: 0.00031540265701313075
Puntos del Polígono: 256
Circunferencia = Valor de pi obtenido con la aproximación: 3.141513801144288
Error: 7.88524455050954e-05
Puntos del Polígono: 512
Circunferencia = Valor de pi obtenido con la aproximación: 3.141572940367109
Error: 1.9713222684014653e-05
Puntos del Polígono: 1024
Circunferencia = Valor de pi obtenido con la aproximación: 3.141587725277193
Error: 4.928312600238627e-06
PS C:\Users\diego\OneDrive\Documents\Aprendiendo\Python\1.-Instrumentación Virtual> █
```

Código C# (.NET Framework) – Visual Studio (Logo Morado)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
//Se usa la librería System.Numerics, agregada dando clic derecho en la parte de:
//Explorador de soluciones -> Referencias -> Agregar Referencia -> Nombre Librería.
using System.Numerics;

namespace _9._3._Aproximación_de_Pi
{
    class AproximacionPi
    {
        //El código en C# se corre presionando CTRL + F5
        //EJERCICIO TAREA 3.13 APROXIMACIÓN DE PI:

        //Función para obtener la longitud de una ruta cualquiera:
        public static double longitudRuta(double[] x, double[] y)
        {
            double Longitud = 0, L = 0;

            //L = ΣV(xi - xi - 1) ^ 2 + (yi - yi - 1) ^ 2
            for (int i = 1; i < x.Length; i++)//Bucle for para realizar la sumatoria de los elementos
```



```

        {
            L = Math.Sqrt(Math.Pow((x[i] - x[i - 1]), 2) + Math.Pow((y[i] - y[i - 1]), 2));
            Longitud += L;
        }
        return Longitud;
    }

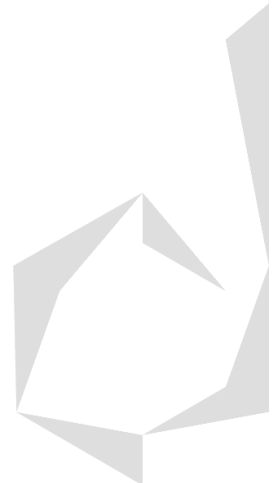
    /*Función para obtener el número de vértices del polígono con el que se pretende aproximar el valor de pi
    al crear un círculo o lo que más se parezca teniendo (n^k)+1 vértices, la función recibe 1 parámetro y
    retorna dos variables array tipo double.*/
    public static (double[], double[]) coordenadas_poligono(int num_vertices)
    {
        /*Declaración de los array vacíos que almacenarán las coordenadas x,y de cada uno de los vértices del
        polígono que se aproxime al círculo de diámetro 1 para obtener el valor de pi, que es el perímetro del
        círculo. El tamaño de los arrays se da por medio del parámetro num_vertices = (2^k)+1*/
        double[] x_p = new double[num_vertices];
        double[] y_p = new double[num_vertices];

        /*El bucle se tiene que ejecutar n + 1 veces, pero n = 2 ^ k, por lo tanto se debe ser:
        num_vertices = (n^k)+1 veces
        Pero este número de veces es el que recibe como parámetro la función.*/
        for (int i = 0; i < num_vertices; i++)
        {
            //xi = 1/2*cos(2*pi*i/n) = 1/2*cos(2*pi*i/(2^k)); i = n+1 = (2^k)+1
            x_p[i] = 0.5 * Math.Cos(2 * Math.PI * i / num_vertices);
            //yi = 1/2*sin(2*pi*i/n) = 1/2*sin(2*pi*i/(2^k)); i = n+1 = (2^k)+1
            y_p[i] = 0.5 * Math.Sin(2 * Math.PI * i / num_vertices);
        }
        return (x_p, y_p);
    }

    //Función main: Desde aquí se empieza a ejecutar todo el código
    static void Main(string[] args)
    {
        //n = base del número de vertices que tendrá el polígono que se aproxima al círculo para encontrar pi.
        int n = 2;
        /*Array k que debe almacenar todas las potencias de los n=2^k puntos en el polígono empezando desde
        2 hasta el número 10.*/
        int[] k = new int[9];
        //Declaración de los arrays vacíos para obtener las coordenadas al usar la función coordenadas_poligono
        double[] coordenadas_x = new double[k.Length];
        double[] coordenadas_y = new double[k.Length];
        double[] circunferencias = new double[k.Length];

        for (int i = 0; i < k.Length; i++)
        {
            k[i] = (int)Math.Pow(n, i + 2);
            //Console.WriteLine(n[i]);
            (coordenadas_x, coordenadas_y) = coordenadas_poligono(k[i]);
            circunferencias[i] = longitudRuta(coordenadas_x, coordenadas_y);
            Console.WriteLine("n: {0}", k[i]);
            Console.WriteLine("Circunferencia = Valor de pi obtenido con la aproximación: {0}", circunferencias[i]);
            Console.WriteLine("Error: {0}", Math.PI - circunferencias[i]);
            Console.WriteLine("");
        }
    }
}

```



Resultado del Código C#

```
C:\WINDOWS\system32\cmd. x + v
n: 4
Circunferencia = Valor de pi obtenido con la aproximación: 2.12132034355964
Error: 1.02027231003015

n: 8
Circunferencia = Valor de pi obtenido con la aproximación: 2.67878402655563
Error: 0.462808627034165

n: 16
Circunferencia = Valor de pi obtenido con la aproximación: 2.92635483024192
Error: 0.21523782334787

n: 32
Circunferencia = Valor de pi obtenido con la aproximación: 3.03853135021638
Error: 0.103061303373413

n: 64
Circunferencia = Valor de pi obtenido con la aproximación: 3.09126348262734
Error: 0.0503291709624567

n: 128
Circunferencia = Valor de pi obtenido con la aproximación: 3.11673602240987
Error: 0.0248566311799254

n: 256
Circunferencia = Valor de pi obtenido con la aproximación: 3.12924226285857
Error: 0.0123503907312243

n: 512
Circunferencia = Valor de pi obtenido con la aproximación: 3.13543705571795
Error: 0.00615559787183928

n: 1024
Circunferencia = Valor de pi obtenido con la aproximación: 3.13851976851423
Error: 0.00307288507556569

Presione una tecla para continuar . . . |
```

4.- Series de Fourier: Aproximar una Función Escalonada con una suma de senos/cosenos

A la suma de las funciones seno y/o coseno para obtener una aproximación de otra señal, se le denomina Serie de Fourier. La aproximación de una función mediante una serie de Fourier es una técnica muy importante en el arte de la ciencia y tecnología.

Consideramos la función constante a trozos descrita a continuación:

$$f(t) = \begin{cases} 1, & 0 < t < \frac{T}{2} \\ 0, & t = \frac{T}{2} \\ -1, & -\frac{T}{2} < t < T \end{cases}$$

Se busca aproximar la señal escalón $f(t)$ por medio de la suma:

$$S(t; n) = \frac{4}{\pi} \sum_{i=1}^n \frac{1}{2i-1} \left(\sin \left(\frac{2(2i-1)\pi * t}{T} \right) \right)$$

Mientras mayor sea el número de veces que se realiza la sumatoria de senos, más se aproximará a la función escalón: $S(t; n) \rightarrow f(t)$ a medida que el número de sumas tiende a infinito: $n \rightarrow \infty$.

Pseudocódigo:

- Escribe una Función Python $S(t, n, T)$ para devolver el valor de la suma de senos $S(t; n)$ que describe la serie de Fourier.
- Escribe una Función Python $f(t, T)$ para describir la función escalón $f(t)$.
- Muestra una tabla en consola que enseñe cómo varía el error $f(t) - S(t; n)$ cuando cambia el valor de las variables n y t para los casos en que:

$$n = [1, 3, 5, 10, 30, 100]$$

$$t = \alpha(T); \quad T = 2\pi; \quad \alpha = [0.01, 0.25, 0.49, 0.5, 0.99]$$

Utiliza la tabla para observar cómo la calidad de la aproximación depende del valor de α y n .

Código Python – Visual Studio Code (Logo Azul)

```
# -*- coding: utf-8 -*-

#En Python se introducen comentarios de una sola linea con el simbolo #.
#La primera línea de código incluida en este programa se conoce como declaración de codificación o codificación
#de caracteres. Al especificar utf-8 (caracteres Unicode) como la codificación, nos aseguramos de que el archivo
#pueda contener caracteres especiales, letras acentuadas y otros caracteres no ASCII sin problemas, garantizando
#que Python interprete correctamente esos caracteres y evite posibles errores de codificación.
#Se puede detener una ejecución con el comando [CTRL] + C puesto en consola, con el comando "cls" se borra su
#historial y en Visual Studio Code con el botón superior izquierdo de Play se corre el programa.
#Para comentar en Visual Studio Code varias líneas de código se debe pulsar:
#[CTRL] + K (VSCode queda a la espera). Después pulsa [CTRL] + C para comentar y [CTRL] + U para descomentar.

import math #math: Librería que proporciona funciones y constantes matemáticas como seno,  $\pi$ , logaritmo, etc.

#FUNCIONES EN PYTHON: Las funciones en python se indican a través de la palabra reservada def, seguida del
#nombre de la función, un paréntesis que contiene sus distintos parámetros y dos puntos para denotar su inicio,
#en Python no se utilizan llaves de apertura o cierre para su sintaxis, solamente se utilizan dos puntos para
#indicar su inicio y tabuladores para diferenciar qué es lo que está dentro o fuera de ella.
# - Las funciones son muy utilizadas en Python ya que sirven para volver modular el código, logrando así que
# se pueda ejecutar varias veces una misma acción sin necesidad de escribirla varias veces.
# - Los distintos métodos que se utilizan en todos los lenguajes de programación para ejecutar acciones
# diferentes como por ejemplo lo es el método print(), son en realidad funciones, pero que utilizan una
# arquitectura de programación orientada a objetos (POO).
# - Los parámetros que recibe una función son las variables con las que interactúa para realizar su acción,
# una función cualquiera puede o no recibir parámetros para ejecutarse.
# - Una función puede o no devolver una variable que almacene su valor resultante, esto se realiza a través
# de la palabra reservada "return".
# - Para utilizar una función en Python se debe declarar su nombre seguido de un paréntesis en donde se le
# pase sus parámetros, si es que recibe algunos, además su resultado puede o no guardarse en una variable
# dependiendo de si retorna un valor o no.
```

```

#4.- SERIES DE FOURIER: Aproximar una Función Escalonada con una suma de Senos/Cosenos.

#A la suma de las funciones seno y/o coseno para obtener una aproximación de otra señal, se le denomina Serie de
#Fourier (F.S.). La aproximación de una función mediante una serie de Fourier es una técnica muy importante en
#el arte de la ciencia y tecnología.

#Consideramos la función constante a trozos (escalón) descrita a continuación:

#      1,      0 < t < T/2
#  f(t) = { 0,      t = T/2
#      -1,     T/2 < t < T

#La función abarca desde t = 0 hasta t = T y se vuelve cero cuando t = T/2.

#Se busca aproximar la señal escalón f(t) por medio de la suma:

#  S(t,n) = (4/π)*Σ(1/(2i-1))*sin((2*(2i-1)*π*t)/T); donde i = 1,..., n.

#Mientras mayor sea el número de veces que se realiza la sumatoria de senos, más se aproximará a la función
#escalón: S(t;n) → f(t) a medida que el número de sumas tiende a infinito: n → ∞.

#a) Escribe una Función Python S(t, n, T) para devolver el valor de la suma de senos S(t;n) que describe la
#serie de Fourier.

#BUCLE FOR: En Python no se utilizan llaves de apertura o cierre para la sintaxis de un bucle, solamente se
#utilizan dos puntos para indicar su inicio y tabuladores para diferenciar qué es lo que está dentro o fuera de
#él. Además, después de la palabra reservada "for" se declara una variable local numérica entera que solo
#existirá dentro del bucle y será la que cuente desde 0 por default o desde el primer número indicado dentro del
#paréntesis hasta el extremo indicado en el segundo parámetro que se encuentra dentro del paréntesis de la
#palabra reservada range() para terminar el bucle. En otras palabras, dentro del paréntesis de la palabra
#reservada "range()" se coloca el inicio y final del conteo para indicar cuantas veces se ejecutará el bucle.
#Es importante mencionar que los bucles for se pueden utilizar para realizar sumatorias en operaciones
#matemáticas:

#S(t, n, T) = (4/π)*Σ(1/(2i-1))*sin((2*(2i-1)*π*t)/T); donde i = 1,..., n.
def S(t, n, T):
    serieFourier = 0    #Variable que guarda el valor final de la serie de Fourier S(t,n,T).
    sumatoria = 0       #Variable que guarda el valor temporal de la sumatoria previa a obtener la F.S.
    #Se ejecutó el bucle for hasta n+1 porque ese valor no lo va a tocar, logrando así que la variable i valga:
    #i = 1,..., n.
    for i in range(1, n+1):
        #Dentro del bucle for solo se obtiene el resultado de la sumatoria.
        #sumatoria = Σ(1/(2i-1))*sin((2*(2i-1)*π*t)/T)
        sumatoria = sumatoria + 1.0/(2*i-1)*math.sin((2*((2*i)-1)*math.pi*t)/T)
    #Fuera del bucle for se realiza el resto de la operación, que en este caso es multiplicar la sumatoria
    #por 4/π.
    #S(t, n, T) = (4/π)*sumatoria = (4/π)*Σ(1/(2i-1))*sin((2*(2i-1)*π*t)/T)
    serieFourier = (4/math.pi)*sumatoria
    return serieFourier

#b) Escribe una Función Python f(t,T) para describir la función escalón f(t).

#CONDICIONAL ELSE IF: En Python no se utilizan llaves de apertura o cierre para la sintaxis de un condicional,
#solamente se utilizan dos puntos para indicar su inicio y tabuladores para diferenciar qué es lo que está

```

```

#dentro o fuera de él.

#Los condicionales else if en Python se indican con la palabra reservada elif y lo que hacen es declarar varias
#condiciones que estén conectadas entre sí, para que de esta manera se puedan evaluar todas sus posibilidades:
#
#         1,          0 < t < T/2
#   f(t) = { 0,          t = T/2
#
#         -1,        T/2 < t < T
#
#La función abarca desde t = 0 hasta t = T y se vuelve cero cuando t = T/2.
#Comparador con y sin Tolerancia: Cuando se compara de forma muy rigida dos valores decimales entre sí, puede
#existir un error al realizar su comparación, por esta razón es que es recomendable indicar una tolerancia y
#realizar la comparación respecto a ella, para así evitar errores.
#La comparación "diferente a" con tolerancia se efectúa al realizar una resta entre los valores comparados,
#indicando que si el resultado de dicha resta es menor a la tolerancia declarada, los valores son distintos.
def f(t, T):
    if (0 < t < (T/2)):
        senal_escalon = 1          #f(t) = 1, cuando 0 < t < T/2
        #Comparador con tolerancia: En este caso se está dando una tolerancia de 1e-8 = 0.00000001
        #abs(): Método que saca el valor absoluto de un número, volviéndolo positivo aunque sea originalmente
        #negativo.
        #El método abs() saca el valor absoluto de la comparación para que esta sea más precisa, de esta forma la
        #tolerancia aplica hacia ambos lados.
        #Si la comparación da como resultado False es porque t no está ni cerca de ser igual a T/2, osea: t != T/2,
        #Si da True es porque se encuentra dentro del rango de la tolerancia o es exactamente igual, osea: t == T/2.
    elif (abs(t-(T/2)) < 1e-8):
        senal_escalon = 0          #f(t) = 0, cuando t = T/2
    elif ((T/2) < t < T):
        senal_escalon = -1         #f(t) = -1, cuando T/2 < t < T
    else:
        #Caso de cuando el vector t (tiempo) se sale del rango 0 < t < T:
        print ("El vector t se sale del rango del tiempo, debe estar en el intervalo 0 < t < T, intenta de nuevo")
        senal_escalon = 0
    return senal_escalon

#Número de las sumas de senos que se realizará en la serie de Fourier S(t, n, T) para aproximarse a la función
#escalón f(t, T).
numSumasFourier = [1, 3, 5, 10, 30, 100]
#Periodo de la función senoidal = 2π
T = 2*math.pi          #T = 2*π
#Factor alfa de multiplicación para obtener un tiempo en específico respecto al periodo T.
α = [0.01, 0.25, 0.49, 0.5, 0.99]

#Bucle for each de una sola línea para rellenar una lista con una operación específica:
#Bucle for each en una sola línea: [instrucción      for  variable_local  in  lista_a_recorrer]
#Bucle for en una sola línea:      [instrucción      for  variable_local  in  range(inicio, final)]
tiempo = [alpha*T for alpha in α] #t = α*T

```



```
#c) Muestra una tabla en consola que enseñe cómo varía el error  $f(t) - S(t;n)$  cuando cambia el valor de las variables
#n (número de sumas de fourier) y t (tiempo) para los casos en que:
# n = [1, 3, 5, 10, 30, 100]; t =  $\alpha * T$ ; T =  $2\pi$ ;  $\alpha = [0.01, 0.25, 0.49, 0.5, 0.99]$ 
#BUCLE FOR EACH: Es un tipo especial de bucle for que sirve para recorrer todos los elementos de una lista,
#tupla o diccionario, su sintaxis es la siguiente:
# - for i in lista:
#La instrucción del Bucle for each es equivalente a poner la instrucción:
# - for i in range(0, len(lista)):
for n in numSumasFourier:
    print ("Número de sumas de Fourier: ", n)
    print ("1.- t = ",  $\alpha[0]$ , " $2\pi \cdot t/T$  = ",  $f(\text{tiempo}[0], T)$ , " $S(t, n, T)$  = ",  $S(\text{tiempo}[0], n, T)$ , " $\text{Error} = ", 100 \cdot \text{abs}(S(\text{tiempo}[0], n, T) - f(\text{tiempo}[0], T))$ )
    print ("2.- t = ",  $\alpha[1]$ , " $2\pi \cdot t/T$  = ",  $f(\text{tiempo}[1], T)$ , " $S(t, n, T)$  = ",  $S(\text{tiempo}[1], n, T)$ , " $\text{Error} = ", 100 \cdot \text{abs}(S(\text{tiempo}[1], n, T) - f(\text{tiempo}[1], T))$ )
    print ("3.- t = ",  $\alpha[2]$ , " $2\pi \cdot t/T$  = ",  $f(\text{tiempo}[2], T)$ , " $S(t, n, T)$  = ",  $S(\text{tiempo}[2], n, T)$ , " $\text{Error} = ", 100 \cdot \text{abs}(S(\text{tiempo}[2], n, T) - f(\text{tiempo}[2], T))$ )
    print ("4.- t = ",  $\alpha[3]$ , " $2\pi \cdot t/T$  = ",  $f(\text{tiempo}[3], T)$ , " $S(t, n, T)$  = ",  $S(\text{tiempo}[3], n, T)$ , " $\text{Error} = ", 100 \cdot \text{abs}(S(\text{tiempo}[3], n, T) - f(\text{tiempo}[3], T))$ )
    print ("5.- t = ",  $\alpha[4]$ , " $2\pi \cdot t/T$  = ",  $f(\text{tiempo}[4], T)$ , " $S(t, n, T)$  = ",  $S(\text{tiempo}[4], n, T)$ , " $\text{Error} = ", 100 \cdot \text{abs}(S(\text{tiempo}[4], n, T) - f(\text{tiempo}[4], T))$ )
```

Resultado del Código Python

```
PROBLEMAS SALIDA TERMINAL CONSOLA DE DEPURACIÓN
PS C:\Users\diego\OneDrive\Documents\Aprendiendo\Python\1.-Instrumentación Virtual> & D:/Users/diego/AppData/Local/Programs/Python/Python39/python.exe
"C:/Users/diego/OneDrive/Documents/Aprendiendo/Python/1.-Instrumentación Virtual/Instrumentación Virtual Aplicada/Tareas/Tarea 1/Tareas C#/Ej3_15.py"
Número de sumas de Fourier: 1
1.- t = 0.01 * 2π f(t, T) = 1 S(t, n, T) = 0.0799473724991873 Error = 92.00526275008127
2.- t = 0.25 * 2π f(t, T) = 1 S(t, n, T) = 1.2732395447351628 Error = 27.323954473516277
3.- t = 0.49 * 2π f(t, T) = 1 S(t, n, T) = 0.07994737249918757 Error = 92.00526275008124
4.- t = 0.5 * 2π f(t, T) = 0 S(t, n, T) = 1.5592687330077502e-16 Error = 1.55926873300775e-14
5.- t = 0.99 * 2π f(t, T) = -1 S(t, n, T) = -0.07994737249918717 Error = 92.00526275008129
Número de sumas de Fourier: 3
1.- t = 0.01 * 2π f(t, T) = 1 S(t, n, T) = 0.2381650038375327 Error = 76.18349961624673
2.- t = 0.25 * 2π f(t, T) = 1 S(t, n, T) = 1.1034742721038078 Error = 10.347427210380777
3.- t = 0.49 * 2π f(t, T) = 1 S(t, n, T) = 0.23816500383753397 Error = 76.18349961624659
4.- t = 0.5 * 2π f(t, T) = 0 S(t, n, T) = 9.201261745993586e-16 Error = 9.201261745993585e-14
5.- t = 0.99 * 2π f(t, T) = -1 S(t, n, T) = -0.23816500383753314 Error = 76.18349961624669
Número de sumas de Fourier: 5
1.- t = 0.01 * 2π f(t, T) = 1 S(t, n, T) = 0.39141456468866714 Error = 60.85854353113329
2.- t = 0.25 * 2π f(t, T) = 1 S(t, n, T) = 1.0630539690963425 Error = 6.305396909634253
3.- t = 0.49 * 2π f(t, T) = 1 S(t, n, T) = 0.39141456468866836 Error = 60.85854353113316
4.- t = 0.5 * 2π f(t, T) = 0 S(t, n, T) = 1.2319799212009088e-15 Error = 1.2319799212009089e-13
5.- t = 0.99 * 2π f(t, T) = -1 S(t, n, T) = -0.3914145646886675 Error = 60.85854353113325
Número de sumas de Fourier: 10
1.- t = 0.01 * 2π f(t, T) = 1 S(t, n, T) = 0.7332025651234054 Error = 26.679743487659458
2.- t = 0.25 * 2π f(t, T) = 1 S(t, n, T) = 0.9682476228907636 Error = 3.1752377109236396
3.- t = 0.49 * 2π f(t, T) = 1 S(t, n, T) = 0.7332025651234072 Error = 26.67974348765928
4.- t = 0.5 * 2π f(t, T) = 0 S(t, n, T) = 2.3484340907064398e-15 Error = 2.3484340907064397e-13
5.- t = 0.99 * 2π f(t, T) = -1 S(t, n, T) = -0.7332025651234055 Error = 26.679743487659447
Número de sumas de Fourier: 30
1.- t = 0.01 * 2π f(t, T) = 1 S(t, n, T) = 1.144816107779649 Error = 14.481610777964903
2.- t = 0.25 * 2π f(t, T) = 1 S(t, n, T) = 0.9893926136945961 Error = 1.060738630540392
3.- t = 0.49 * 2π f(t, T) = 1 S(t, n, T) = 1.1448161077796466 Error = 14.48161077796466
4.- t = 0.5 * 2π f(t, T) = 0 S(t, n, T) = 5.902289380275787e-15 Error = 5.902289380275787e-13
5.- t = 0.99 * 2π f(t, T) = -1 S(t, n, T) = -1.1448161077796506 Error = 14.48161077796506
Número de sumas de Fourier: 100
1.- t = 0.01 * 2π f(t, T) = 1 S(t, n, T) = 0.9499059916589789 Error = 5.009400834102107
2.- t = 0.25 * 2π f(t, T) = 1 S(t, n, T) = 0.9968169807056898 Error = 0.31830192943101965
3.- t = 0.49 * 2π f(t, T) = 1 S(t, n, T) = 0.9499059916589752 Error = 5.009400834102484
4.- t = 0.5 * 2π f(t, T) = 0 S(t, n, T) = 1.6088777474263388e-14 Error = 1.6088777474263388e-12
5.- t = 0.99 * 2π f(t, T) = -1 S(t, n, T) = -0.9499059916589805 Error = 5.009400834101951
PS C:\Users\diego\OneDrive\Documents\Aprendiendo\Python\1.-Instrumentación Virtual>
```

Código C# (.NET Framework) – Visual Studio (Logo Morado)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
//Se usa la librería System.Numerics, agregada dando clic derecho en la parte de:
//Explorador de soluciones -> Referencias -> Agregar Referencia -> Nombre Librería.
using System.Numerics;

namespace _9._4._Series_de_Fourier___Escalón
{
    class Aproximacion_Escalon_Suma_Senos
    {
        //El código en C# se corre presionando CTRL + F5
        //EJERCICIO TAREA 3.15 PROXIMAR UNA FUNCIÓN ESCALÓN AL SUMAR SEÑALES SENOIDALES:

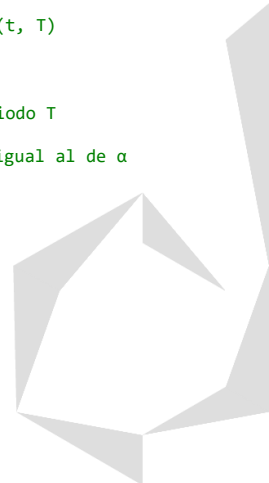
        //Funciones para crear ambas expresiones matemáticas:
        //S(t, n, T) = (4/π)*Σ(1/(2i-1))*sin((2*(2i-1)*πt)/(T))
        static double S(double t, int n, double T)
        {
            double suma_senos = 0;
            for (int i = 1; i < n + 1; i++)
            {
                //S(t, n, T) = Σ(1/(2i-1))*sin((2*(2i-1)*πt)/(T))
                suma_senos = suma_senos + 1.0 / (2 * i - 1) * Math.Sin((2 * ((2 * i) - 1) * Math.PI * t) / T);
            }
            //S(t, n, T) = (4/π)*Σ(1/(2i-1))*sin((2*(2i-1)*πt)/(T))
            suma_senos = suma_senos * (4 / Math.PI);
            return suma_senos;
        }

        //f(t, T) = 1, 0, -1 que va de t=0 a t=T y se vuelve cero cuando t = T/2
        static int f(double t, double T)
        {
            int senal_escalon;
            if (0 < t && t < T / 2) //0<t<T/2
            {
                senal_escalon = 1;
            }
            /*Comparador con tolerancia, el método Math.Abs() saca el valor absoluto de una resta y si el resultado es
            menor a el número decimal descrito, el resultado es erróneo y en este caso t es igual a T/2, osea: t == T/2*/
            else if (Math.Abs(t - T / 2) < 1E-8) //t=T/2
            {
                senal_escalon = 0;
            }
            else if (T / 2 < t && t < T) //T/2<t<T
            {
                senal_escalon = -1;
            }
            else
            {
                Console.WriteLine("El vector t se sale del rango del tiempo, debe estar en el intervalo 0<t<T, intenta de nuevo");
                senal_escalon = 0;
            }
            return senal_escalon;
        }

        //Función main: Desde aquí se empieza a ejecutar todo el código
        static void Main(string[] args)
        {
            //Número de interacciones que se realizará a la función S(t, n, T) para acercarse a f(t, T)
            int[] iteraciones_S = { 1, 3, 5, 10, 30, 100 };
            //Periodo de la función senoidal = 2π
            double T = 2 * Math.PI;
            //Factor alfa de multiplicación para obtener un tiempo en específico respecto al periodo T
            double[] α = { 0.01, 0.25, 0.49, 0.5, 0.99 };
            //En C# al crear un array siempre de debe indicar su tamaño, el tamaño del array es igual al de α
            double[] tiempo = new double[α.Length];

            //Bucle for para rellenar el array del tiempo con la operación indicada:
            for (int i = 0; i < α.Length; i++)
            {
                //t = α*T
                tiempo[i] = α[i] * T;
            }

            //bucle for each: Recorre todos los elementos del array nombrado en el bucle
        }
    }
}
```



```

foreach (int n in iteraciones_S)
{
    Console.WriteLine("Iteración: {0}", n);
    Console.WriteLine("1.- t = {0}*2π \tf(t, T) = {1}\tS(t, n, T) = {2}\tError = {3}", α[0], f(tiempo[0], T),
S(tiempo[0], n, T), 100 * Math.Abs(S(tiempo[0], n, T) - f(tiempo[0], T)));
    Console.WriteLine("2.- t = {0}*2π \tf(t, T) = {1}\tS(t, n, T) = {2}\tError = {3}", α[1], f(tiempo[1], T),
S(tiempo[1], n, T), 100 * Math.Abs(S(tiempo[1], n, T) - f(tiempo[1], T)));
    Console.WriteLine("3.- t = {0}*2π \tf(t, T) = {1}\tS(t, n, T) = {2}\tError = {3}", α[2], f(tiempo[2], T),
S(tiempo[2], n, T), 100 * Math.Abs(S(tiempo[2], n, T) - f(tiempo[2], T)));
    Console.WriteLine("4.- t = {0}*2π \tf(t, T) = {1}\tS(t, n, T) = {2}\tError = {3}", α[3], f(tiempo[3], T),
S(tiempo[3], n, T), 100 * Math.Abs(S(tiempo[3], n, T) - f(tiempo[3], T)));
    Console.WriteLine("5.- t = {0}*2π \tf(t, T) = {1}\tS(t, n, T) = {2}\tError = {3}", α[4], f(tiempo[4], T),
S(tiempo[4], n, T), 100 * Math.Abs(S(tiempo[4], n, T) - f(tiempo[4], T)));
    Console.WriteLine();
}
}
}
}

```

Resultado del Código C#

```

C:\WINDOWS\system32\cmd. x + v
Iteración: 1
1.- t = 0.01*2π      f(t, T) = 1      S(t, n, T) = 0.0799473724991873      Error = %92.0052627500813
2.- t = 0.25*2π      f(t, T) = 1      S(t, n, T) = 1.27323954473516      Error = %27.3239544735163
3.- t = 0.49*2π      f(t, T) = 1      S(t, n, T) = 0.0799473724991876      Error = %92.0052627500812
4.- t = 0.5*2π       f(t, T) = 0      S(t, n, T) = 1.55921723642059E-16      Error = %1.55921723642059E-14
5.- t = 0.99*2π      f(t, T) = -1     S(t, n, T) = -0.0799473724991872      Error = %92.0052627500813

Iteración: 3
1.- t = 0.01*2π      f(t, T) = 1      S(t, n, T) = 0.238165003837533      Error = %76.1834996162467
2.- t = 0.25*2π      f(t, T) = 1      S(t, n, T) = 1.10347427210381      Error = %10.3474272103808
3.- t = 0.49*2π      f(t, T) = 1      S(t, n, T) = 0.238165003837534      Error = %76.1834996162466
4.- t = 0.5*2π       f(t, T) = 0      S(t, n, T) = 9.20110725623211E-16      Error = %9.20110725623211E-14
5.- t = 0.99*2π      f(t, T) = -1     S(t, n, T) = -0.238165003837533      Error = %76.1834996162467

Iteración: 5
1.- t = 0.01*2π      f(t, T) = 1      S(t, n, T) = 0.391414564688667      Error = %60.8585435311333
2.- t = 0.25*2π      f(t, T) = 1      S(t, n, T) = 1.06305396909634      Error = %6.30539690963425
3.- t = 0.49*2π      f(t, T) = 1      S(t, n, T) = 0.391414564688668      Error = %60.8585435311332
4.- t = 0.5*2π       f(t, T) = 0      S(t, n, T) = 1.23195417290733E-15      Error = %1.23195417290733E-13
5.- t = 0.99*2π      f(t, T) = -1     S(t, n, T) = -0.391414564688667      Error = %60.8585435311333

Iteración: 10
1.- t = 0.01*2π      f(t, T) = 1      S(t, n, T) = 0.733202565123405      Error = %26.6797434876595
2.- t = 0.25*2π      f(t, T) = 1      S(t, n, T) = 0.968247622890764      Error = %3.17523771092364
3.- t = 0.49*2π      f(t, T) = 1      S(t, n, T) = 0.733202565123407      Error = %26.6797434876593
4.- t = 0.5*2π       f(t, T) = 0      S(t, n, T) = 2.34838259411928E-15      Error = %2.34838259411928E-13
5.- t = 0.99*2π      f(t, T) = -1     S(t, n, T) = -0.733202565123406      Error = %26.6797434876594

Iteración: 30
1.- t = 0.01*2π      f(t, T) = 1      S(t, n, T) = 1.14481610777965      Error = %14.4816107779649
2.- t = 0.25*2π      f(t, T) = 1      S(t, n, T) = 0.989392613694596      Error = %1.06073863054039
3.- t = 0.49*2π      f(t, T) = 1      S(t, n, T) = 1.14481610777965      Error = %14.4816107779647
4.- t = 0.5*2π       f(t, T) = 0      S(t, n, T) = 5.90213489051431E-15      Error = %5.90213489051431E-13
5.- t = 0.99*2π      f(t, T) = -1     S(t, n, T) = -1.14481610777965      Error = %14.4816107779651

Iteración: 100
1.- t = 0.01*2π      f(t, T) = 1      S(t, n, T) = 0.949905991658979      Error = %5.00940083410211
2.- t = 0.25*2π      f(t, T) = 1      S(t, n, T) = 0.99681698070569      Error = %0.31830192943102
3.- t = 0.49*2π      f(t, T) = 1      S(t, n, T) = 0.949905991658975      Error = %5.00940083410248
4.- t = 0.5*2π       f(t, T) = 0      S(t, n, T) = 1.60882625083918E-14      Error = %1.60882625083918E-12

```

5.- Campana de Gauss: Implementar una función Gaussiana

La campana de Gauss o también llamada distribución normal es una **representación gráfica que muestra una distribución de datos en torno a un valor central llamado media**, con la Campana de Gauss es posible establecer una serie de parámetros que ayudan a predecir y racionalizar lo que aparentemente son resultados aleatorios, obteniendo una versión más clara y visual de la distribución

de un conjunto de números. Esta herramienta se utiliza para representar la dispersión de los datos y su tendencia, con el fin de detectar patrones o comportamientos en diferentes situaciones, por lo cual es muy utilizada en estadística, probabilidad, filtros, visión artificial, etc. En su expresión más sencilla:

- a = Representa el valor más alto (amplitud) de la campana de Gauss.
 - Mientras menor sea el valor de σ , mayor será la amplitud de la función.

$$a = \frac{1}{\sigma\sqrt{2\pi}}$$

- b = Es la posición del centro de la campana.
 - $b = \mu = m$: Esta variable es llamada media.
 - La función es simétrica respecto a la media μ .
 - Su valor máximo se encuentra en la media μ .
- c = Controla el ancho de la campana de Gauss (desviación estándar).
 - $c^2 = \sigma^2 = s^2$: Esta variable es llamada varianza.
 - En las coordenadas de $\mu - \sigma$ y $\mu + \sigma$ se presentan los puntos de inflexión de la curva.

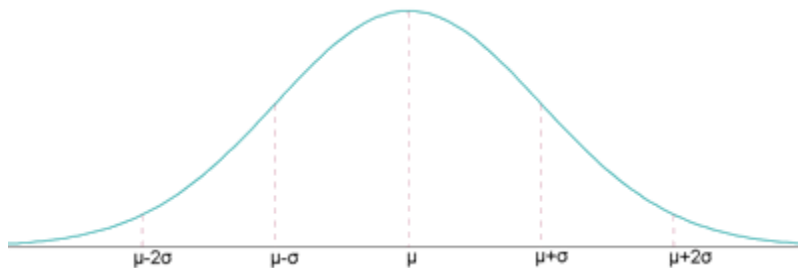
$$f(x) = (a)e^{\left[\frac{-(x-b)^2}{2c^2}\right]} = \left(\frac{1}{\sigma\sqrt{2\pi}}\right)e^{\left[\frac{-(x-\mu)^2}{2\sigma^2}\right]} = \left(\frac{1}{\sigma\sqrt{2\pi}}\right)e^{\left[-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right]}$$

- El 100% de la probabilidad equivale al área encerrada bajo la curva:

$$f(\mu - \sigma < x < \mu + \sigma) = 0.6826 = 68.26 \%$$

$$f(\mu - 2\sigma < x < \mu + 2\sigma) = 0.954 = 95.4 \%$$

$$f(\mu - 3\sigma < x < \mu + 3\sigma) = 0.997 = 99.7 \%$$



Pseudocódigo:

- Escribe una función en Python llamada **gauss(x, m = 0, s = 1)** para calcular el resultado de una función Gaussiana, que es descrita por la siguiente función:

$$f(x) = \frac{1}{s\sqrt{2\pi}}e^{\left[-\frac{1}{2}\left(\frac{x-m}{s}\right)^2\right]} = \frac{1}{\sigma\sqrt{2\pi}}e^{\left[-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right]}$$

- Grafica la función de Gauss con n valores de x y $f(x)$ uniformemente espaciados con intervalos de $[m - 5s, m + 5s]$. Elija m , s y n libremente.
- Escribe una tabla bien formateada.

Código Python – Visual Studio Code (Logo Azul)

```
# -*- coding: utf-8 -*-

#En Python se introducen comentarios de una sola linea con el simbolo #.
#La primera línea de código incluida en este programa se conoce como declaración de codificación o codificación
#de caracteres. Al especificar utf-8 (caracteres Unicode) como la codificación, nos aseguramos de que el archivo
#pueda contener caracteres especiales, letras acentuadas y otros caracteres no ASCII sin problemas, garantizando
#que Python interprete correctamente esos caracteres y evite posibles errores de codificación.
#Se puede detener una ejecución con el comando [CTRL] + C puesto en consola, con el comando "cls" se borra su
#historial y en Visual Studio Code con el botón superior izquierdo de Play se corre el programa.
#Para comentar en Visual Studio Code varias líneas de código se debe pulsar:
#[CTRL] + K (VSCode queda a la espera). Después pulsa [CTRL] + C para comentar y [CTRL] + U para descomentar.

import math #math: Librería que proporciona funciones y constantes matemáticas como seno,  $\pi$ , logaritmo, etc.
import numpy as np #numpy: Librería que realiza operaciones matemáticas complejas (matriciales).
import tabulate #tabulate: Librería que permite crear tablas con varios formatos conocidos.
import matplotlib.pyplot as plt #matplotlib: Librería de graficación matemática.

#FUNCIONES EN PYTHON: Las funciones en python se indican a través de la palabra reservada def, seguida del
#nombre de la función, un paréntesis que contiene sus distintos parámetros y dos puntos para denotar su inicio,
#en Python no se utilizan llaves de apertura o cierre para su sintaxis, solamente se utilizan dos puntos para
#indicar su inicio y tabuladores para diferenciar qué es lo que está dentro o fuera de ella.
# - Las funciones son muy utilizadas en Python ya que sirven para volver modular el código, logrando así que
# se pueda ejecutar varias veces una misma acción sin necesidad de escribirla varias veces.
# - Los distintos métodos que se utilizan en todos los lenguajes de programación para ejecutar acciones
# diferentes como por ejemplo lo es el método print(), son en realidad funciones, pero que utilizan una
# arquitectura de programación orientada a objetos (POO).
# - Los parámetros que recibe una función son las variables con las que interactúa para realizar su acción,
# una función cualquiera puede o no recibir parámetros para ejecutarse.
# - Una función puede o no devolver una variable que almacene su valor resultante, esto se realiza a través
# de la palabra reservada "return".
# - Para utilizar una función en Python se debe declarar su nombre seguido de un paréntesis en donde se le
# pase sus parámetros, si es que recibe algunos, además su resultado puede o no guardarse en una variable
# dependiendo de si retorna un valor o no.

#5.- CAMPANA DE GAUSS: La campana de Gauss o también llamada distribución normal es una representación gráfica
#que muestra una distribución de datos en torno a un valor central llamado media, con la Campana de Gauss es
#posible establecer una serie de parámetros que ayudan a predecir y racionalizar lo que aparentemente son
#resultados aleatorios, obteniendo una versión más clara y visual de la distribución de un conjunto de números.
#Esta herramienta se utiliza para representar la dispersión de los datos y su tendencia, con el fin de detectar
#patrones o comportamientos en diferentes situaciones, por lo cual es muy utilizada en estadística,
#probabilidad, filtros, visión artificial, etc. En su expresión más sencilla:
# - a = Representa el valor más alto (amplitud) de la campana de Gauss.
# - Mientras menor sea el valor de o, mayor será la amplitud de la función.
# - b = Es la posición del centro de la campana.
```

```

# - b = μ = m: Esta variable es llamada media.
# - La función es simétrica respecto a la media μ.
# - Su valor máximo se encuentra en la media μ.
# - c = Controla el ancho de la campana de Gauss (desviación estándar).
# - c^2 = σ^2 = s^2: Esta variable es llamada varianza.
# - En las coordenadas de μ - σ y μ + σ se presentan los puntos de inflexión de la curva.
#G(x) = (a)e^[-(x-b)^2/(2c^2)] = (1/(σ√2π))e^[-(x-μ)^2/(2σ^2)] = (1/(s√2π))e^[-1/2*((x-m)/s)^2].

#a) Escribe una Función en Python llamada gauss(x, m = 0, s = 1) para calcular el resultado de una función
#Gaussiana.

#La función gaussiana recibe 3 parámetros:
# - x: Eje horizontal.
# - m: Media, es el punto central de la función.
# - s: Varianza, dicta el ancho de la función.
#G(x) = (1/s√2π)*exp(-1/2*((x-m)/s)^2)

def gauss(x, m , s ):
    resultado = []          #Lista vacía que almacena el resultado de la función Gaussiana.
    #BUCLE FOR EACH: Es un tipo especial de bucle for que sirve para recorrer todos los elementos de una lista,
    #tupla o diccionario, su sintaxis es la siguiente:
    # - for i in lista:
    #La instrucción del Bucle for each es equivalente a poner la instrucción:
    # - for i in range(0, len(lista)):
    #Es importante mencionar que al usar el bucle for each, la variable del Bucle ahora será manejada como si
    #fuera una lista en sí a la cual se le están accediendo todos sus valores, por lo cual ahora usar esta
    #variable corresponde a haber accedido a cada valor de la lista que recorra el bucle:
    # - i = lista[i]
    for i in x:              #Bucle for each que recorre todos los puntos del eje x.
        var_temporal = []    #Variable intermedia que almacena cada coordenada (x, G) de la función Gauss.
        #math.sqrt(x) = √x = x**1/2: Método que saca la raíz cuadrada de un número.
        #math.pi(): Constante pi (π) de la librería math.
        #math.exp(x) = e^x = e**x: Método que devuelve el valor exponencial de un número con base "e".
        #G(x) = (1/s√2π)*exp((-1/2)*((x-m)/s)^2)
        G = (1/(s*math.sqrt(2*math.pi)))*(math.exp((-1/2)*((i-m)/s)**2))
        #append(): Método que sirve para agregar valores a una lista, tupla, numpy array o diccionario.
        #Dentro de la lista de variable temporal primero se agregan los valores del eje x y luego los resultados
        #de la función Gaussiana correspondiente para cada uno.
        #Colocar primero .append(i) y luego .append(G) no es lo mismo a poner .append([i, G]), ya que con la
        #segunda instrucción se estará creando una lista interna innecesaria que estará causando problemas al
        #intentar crear la tabla de datos con la librería tabulate.
        var_temporal.append(i)
        var_temporal.append(G)
    #Al final el resultado de la variable temporal que contiene los valores de x y G se agrega a la lista de
    #resultado, para que de esta manera se cree una lista anidada que contenga las coordenadas de todos los
    #puntos de la función Gaussiana.
    #Otra forma de lograr lo mismo sin haber usado una variable intermedia fue haber aplicado el método

```

```

        #.append([i, G]) directamente al vector resultado.
        resultado.append(var_temporal)

    print("Coordenadas del vector Var temporal:\n", var_temporal, "\n")
    print("Vector Resultado G(x):\n", resultado, "\n")
    return resultado

#Ahora se debe crear el rango de valores para los cuales se aplicará la función Gauss, que esta estará en un
#intervalo de: [m - 5s, m + 5s]

m = 0                #m = Media, indica el punto central de la función.
s = 1                #s = Varianza, mientras mayor sea, función más ancha, pero reduce su amplitud.
n = 30               #n = Número de valores que conforman la curva de Gauss.
r = (abs(m-5*s) + abs(m+5*s))/(n-1) #r = intervalo = [m-5s, m+5s]/(n-1)

#numpy.arange(): El método arange sirve para crear un listado de elementos indicando su punto inicial, punto
#final e intervalo de valores; np.arange(punto_inicial, punto_final, intervalo).
xi = np.arange((m-5*s), (m+ 5*s) + r, r)

#USO DE LA FUNCIÓN gauss(): Creación de la función Gauss a través del eje horizontal x, el valor de media (m) y
#de varianza (s).
fg = gauss(xi, m, s)    #Aplicación de la función Gauss

#BUCLE FOR EACH: Es un tipo especial de bucle for que sirve para recorrer todos los elementos de una lista,
#tupla o diccionario, su sintaxis es la siguiente:
# - for i in lista:

#La instrucción del Bucle for each es equivalente a poner la instrucción:
# - for i in range(0, len(lista)):

#Es importante mencionar que al usar el bucle for each, la variable del Bucle ahora será manejada como si fuera
#una lista en sí a la cual se le están accediendo todos sus valores, por lo cual ahora usar esta variable
#corresponde a haber accedido a cada valor de la lista que recorra el bucle:
# - i = lista[i]

x_graph = []          #Lista vacía que almacenará el eje horizontal para graficar la función Gauss.
y_graph = []          #Lista vacía que almacenará el eje vertical para graficar la función Gauss.
for j in fg:          #Bucle for each que recorre todas las coordenadas de la función Gauss.
    #append(): Método que sirve para agregar valores a una lista, tupla, numpy array o diccionario.
    x_graph.append(j[0])    #Acceso al índice 0 (x) de la lista anidada en cada coordenada del vector fg.
    y_graph.append(j[1])    #Acceso al índice 1 (y) de la lista anidada en cada coordenada del vector fg.

#print(): Método para imprimir un mensaje en consola y después dar un salto de línea (Enter), además si se
#quiere concatenar un mensaje (mostrar resultados de variables junto con texto estático), este se debe separar
#entre comillas, declarando los mensajes estáticos entre comillas y los nombres de variables sin comillas.
print("x:", x_graph, "\n")    #Impresión en consola del vector x.
print("y:", y_graph, "\n")    #Impresión en consola del vector y.
print("\n")

#GRAFICAR DATOS EN PYTHON:
#matplotlib.figure(): Método usado para crear la ventana de graficación (objeto pyplot).
fig = plt.figure()#Creación del objeto pyplot.

```

```

#matplotlib.axes(): Método usado para crear la rejilla en la ventana de graficación.
ax = plt.axes()#Se crea un objeto ax perteneciente de la clase axes().
#axes.plot(): Método usado para crear la gráfica en la rejilla previamente creada en la ventana de graficación,
#indicando como primer parámetro su eje horizontal, luego su eje vertical y finalmente el estilo de la gráfica:
# - Colores:          C1: color naranja, r: color rojo, b: color azul, g: verde, c: cyan, m: morado,
#   y: amarillo, k: negro, w: blanco.
# - Tipo de marcadores: o: círculos, +: símbolos de más, .; puntos, v: Triángulo hacia abajo, h: Hexágono, etc.
# - Tipo de Líneas:   -: sólida, --: punteada (líneas), :: punteada (puntos), -.: línea y punto, 'or': Nada.
ax.plot(x_graph, y_graph, 'w1:')#w1: significa w: color blanco, 1: tri_down :: línea punteada (puntos).
#axes.set_xlabel(): Método para indicar el texto que aparece en el eje x.
ax.set_xlabel(r'$x$')#Texto que aparece en el eje horizontal
#axes.set_ylabel(): Método para indicar el texto que aparece en el eje y.
ax.set_ylabel(r'$y = G(x, m, s)$')#Texto que aparece en el eje vertical
#axes.set_facecolor(): Método para indicar el color de fondo de la gráfica, el cual puede ser indicado por los
#mismos colores previamente mencionados en el método plot() o se pueden usar los siguientes con el código xkcd:
# - Colores: xkcd:aqua, xkcd:aquamarine, xkcd:azure, xkcd:beige, etc. Los colores se obtienen de este link:
#https://matplotlib.org/stable/tutorials/colors/colors.html
ax.set_facecolor('xkcd:crimson')
#matplotlib.show(): Método para mostrar la gráfica creada.
plt.show()

#CREAR UNA TABLA CON LA LIBRERÍA TABULATE: La tabla se mostrará hasta que cierre la ventana de la gráfica.
#tabulate.tabulate(): Método que sirve para crear una lista con los valores de un vector, dicho vector debe
#contener listas anidadas en cada una de sus posiciones para que estas sean agrupadas en cada fila de la lista.
#Las tablas creadas con el método tabulate aceptan los siguientes parámetros:
# - tabular_data: Este parámetro especifica los datos a mostrar. Puede ser una lista de listas, una lista de
#   diccionarios, una lista de tuplas, o una estructura de datos similar.
# - headers: Es un parámetro opcional que se utiliza para especificar los encabezados de las columnas de la
#   tabla. Puede ser una lista de cadenas que representen los nombres de las columnas.
# - tablefmt: Es un parámetro opcional que indica el formato deseado para la tabla generada.
#   - "plain": Sin formato adicional.
#   - "simple": Formato simple con líneas horizontales.
#   - "grid": Formato con bordes de cuadrícula.
#   - "pipe": Formato con delimitadores de tubería.
#   - "orgtbl": Formato de tabla org.
# - numalign: Este parámetro se utiliza para alinear los números en las columnas de la tabla. Puede tener los
#   siguientes valores:
#   - "left": Alinea los números a la izquierda.
#   - "center": Centra los números en la columna.
#   - "right": Alinea los números a la derecha.
# - stralign: Este parámetro se utiliza para alinear las cadenas de texto en las columnas de la tabla. Puede
#   tener los siguientes valores:
#   - "left": Alinea las cadenas de texto a la izquierda.
#   - "center": Centra las cadenas de texto en la columna.
#   - "right": Alinea las cadenas de texto a la derecha.

```



```
# - missingval: Este parámetro se utiliza para establecer el valor a mostrar cuando los datos son nulos o no están
# disponibles. Puede ser cualquier valor válido, como una cadena de texto, un número, etc. Cuando se encuentre un
# valor nulo en los datos, se mostrará el valor especificado en missingval.

print(tabulate.tabulate(tabular_data = fg, headers = ['x','f(x)'], tablefmt = "grid", numalign = "center", missingval =
"invalid_data"))
```

Resultado del Código Python

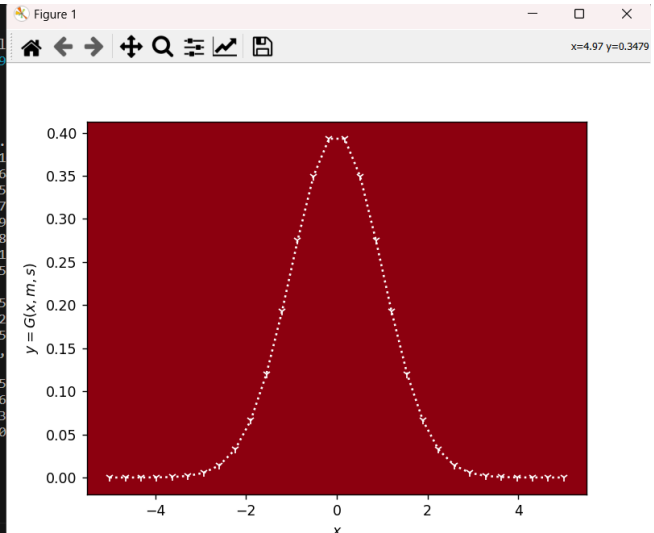
```
PROBLEMAS SALIDA TERMINAL CONSOLA DE DEPURACIÓN

PS C:\Users\diego\OneDrive\Documents\Aprendiendo\Python\1.-Instrumentación Virtual>
"c:\Users\diego\OneDrive\Documents\Aprendiendo\Python\1.-Instrumentación Virtual\9
Coordenadas del vector Var temporal:
[5.000000000000007, 1.4867195147342452e-06]

Vector Resultado G(x):
[[-5.0, 1.4867195147342977e-06], [-4.655172413793103, 7.85583779599305e-06], [-4.
5353192547026454], [-3.620689655172413, 0.0005678601849840233], [-3.27586206896551
, [-2.5862068965517224, 0.01407758322283598], [-2.2413793103448256, 0.032360098606
2, 0.11968853725031257], [-1.2068965517241352, 0.1925810496954935], [-0.8620689655
4], [-0.17241379310344485, 0.39305654728944145], [0.17241379310345195, 0.393056547
55, 0.27512768393363507], [1.2068965517241423, 0.19258104969549186], [1.5517241379
], [2.2413793103448327, 0.03236009860644958], [2.5862068965517295, 0.0140775832228
3, 0.001864849957905257], [3.62068965517242, 0.0005678601849840087], [3.965517241
899e-05], [4.65517241379311, 7.855837795992785e-06], [5.000000000000007, 1.4867195

x: [-5.0, -4.655172413793103, -4.310344827586206, -3.9655172413793096, -3.62068965
-2.2413793103448256, -1.8965517241379288, -1.551724137931032, -1.2068965517241352
7241379310345195, 0.5172413793103487, 0.8620689655172455, 1.2068965517241423, 1.55
7295, 2.9310344827586263, 3.275862068965523, 3.62068965517242, 3.9655172413793167,

y: [1.4867195147342977e-06, 7.85583779599305e-06, 3.685663874863011e-05, 0.0001535
585803353717, 0.01407758322283598, 0.0323600986064501, 0.06604673018611557, 0.1196
22534, 0.39305654728944145, 0.393056547289441, 0.34899145007225213, 0.275127683933
0.03236009860644958, 0.014077583222835725, 0.005437585803353606, 0.00186484995790
99e-05, 7.855837795992785e-06, 1.4867195147342452e-06]
```



PROBLEMAS	SALIDA	TERMINAL	CONSOLA DE DEPURACIÓN
x	f(x)		
-5	1.48672e-06		
-4.65517	7.85584e-06		
-4.31034	3.68566e-05		
-3.96552	0.000153532		
-3.62069	0.00056786		
-3.27586	0.00186485		
-2.93103	0.00543759		
-2.58621	0.0140776		
-2.24138	0.0323601		
-1.89655	0.0660467		
-1.55172	0.119689		
-1.2069	0.192581		
-0.862069	0.275128		
-0.517241	0.348991		
-0.172414	0.393057		
0.172414	0.393057		
0.517241	0.348991		
0.862069	0.275128		
1.2069	0.192581		

```
+-----+-----+
| -0.862069 | 0.275128 |
+-----+-----+
| -0.517241 | 0.348991 |
+-----+-----+
| -0.172414 | 0.393057 |
+-----+-----+
| 0.172414 | 0.393057 |
+-----+-----+
| 0.517241 | 0.348991 |
+-----+-----+
| 0.862069 | 0.275128 |
+-----+-----+
| 1.2069 | 0.192581 |
+-----+-----+
| 1.55172 | 0.119689 |
+-----+-----+
| 1.89655 | 0.0660467 |
+-----+-----+
| 2.24138 | 0.0323601 |
+-----+-----+
| 2.58621 | 0.0140776 |
+-----+-----+
| 2.93103 | 0.00543759 |
+-----+-----+
| 3.27586 | 0.00186485 |
+-----+-----+
| 3.62069 | 0.00056786 |
+-----+-----+
| 3.96552 | 0.000153532 |
+-----+-----+
| 4.31034 | 3.68566e-05 |
+-----+-----+
| 4.65517 | 7.85584e-06 |
+-----+-----+
| 5 | 1.48672e-06 |
+-----+-----+
PS C:\Users\diego\OneDrive\Documents\Aprendiendo\Python\1.-Instrumentación Virtual>
```

Lín. 183, col. 137 Espacios: 4 UTF-8 CRLF Python

Código C# (.NET Framework) – Visual Studio (Logo Morado)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace _9._5._Campana_de_Gauss
{
    class FuncionGaussiana
    {
        //El código en C# se corre presionando CTRL + F5
        //EJERCICIO TAREA 3.16 FUNCIÓN GAUSSIANA:

        static double gauss(double x, int m, double s)
        {
            double gauss = 0;
            gauss = Math.Exp(Math.Pow(((x - m) / s), 2) / (-2)) / (Math.Sqrt(2 * Math.PI) * s);
            return gauss;
        }

        //Función main: Desde aquí se empieza a ejecutar todo el código
        static void Main(string[] args)
        {
            //Rango de valores para los cuales se aplicará la función Gauss, esta estará en un intervalo de: [m-5s, m+5s]
            int m = 0;
            double s = 2.2360;
            //número de valores
            int n = 30;
            //Número de elementos a los cuales se aplicará la función Gaussiana
            double[] x = new double[n + 1];
            double[] fg = new double[n + 1];

            //Número de elementos a los cuales se aplicará la función Gaussiana
            double r = (Math.Abs(m - 5 * s) + Math.Abs(m + 5 * s)); //[m-5s, m+5s]
            double xi = r / n; //intervalo [m-5s, m+5s]/(n-1)

            //Creación de los valores con punto inicial m-5s, punto final m+5s e intervalo [m-5s, m+5s]/(n-1)
            for (int i = 0; i < n + 1; i++)
            {
                if (i == 0)
                {
```

```

        x[i] = m - 5 * s;
    }
    else
    {
        x[i] = x[i - 1] + xi;
    }
    fg[i] = gauss(x[i], m, s); //Aplicación de la función gauss
}

/*Bucle for para crear la tabla con los valores de x y su resultado f(x) con la función Gauss e
imprimirla en consola*/
for (int i = 0; i < n + 1; i++)
{
    Console.WriteLine(" x = {0:F5}\t\t-----\t\tf(x) = {1:F5}\n", x[i], fg[i]);
}
}
}
}
}

```

Resultado del Código C#

```

C:\WINDOWS\system32\cmd. x + v
x = -5.00000 ----- f(x) = 0.00000
x = -4.66667 ----- f(x) = 0.00001
x = -4.33333 ----- f(x) = 0.00003
x = -4.00000 ----- f(x) = 0.00013
x = -3.66667 ----- f(x) = 0.00048
x = -3.33333 ----- f(x) = 0.00154
x = -3.00000 ----- f(x) = 0.00443
x = -2.66667 ----- f(x) = 0.01140
x = -2.33333 ----- f(x) = 0.02622
x = -2.00000 ----- f(x) = 0.05399
x = -1.66667 ----- f(x) = 0.09948
x = -1.33333 ----- f(x) = 0.16401
x = -1.00000 ----- f(x) = 0.24197
x = -0.66667 ----- f(x) = 0.31945
x = -0.33333 ----- f(x) = 0.37738
x = 0.00000 ----- f(x) = 0.39894
x = 0.33333 ----- f(x) = 0.37738
x = 0.66667 ----- f(x) = 0.31945
x = 1.00000 ----- f(x) = 0.24197
x = 1.33333 ----- f(x) = 0.16401
x = 1.66667 ----- f(x) = 0.09948
x = 2.00000 ----- f(x) = 0.05399
x = 2.33333 ----- f(x) = 0.02622
x = 2.66667 ----- f(x) = 0.01140
x = 3.00000 ----- f(x) = 0.00443
x = 3.33333 ----- f(x) = 0.00154
x = 3.66667 ----- f(x) = 0.00048
x = 4.00000 ----- f(x) = 0.00013
x = 4.33333 ----- f(x) = 0.00003
x = 4.66667 ----- f(x) = 0.00001
x = 5.00000 ----- f(x) = 0.00000
Presione una tecla para continuar . . . |

```

6.- Pruebas Unitarias: Librerías de pruebas distintas/Testing: Unittesting y Nose

Usar dos librerías distintas para realizar pruebas unitarias en un mismo programa puede resultar complicado y propenso a errores, ya que los frameworks de pruebas están diseñados para proporcionar una estructura coherente y consistente para las pruebas, por lo que mezclar dos o más frameworks puede generar inconsistencias y conflictos. Si se desea utilizar librerías distintas para diferentes pruebas en un mismo programa de Python, es recomendable separar las pruebas en módulos o archivos diferentes, asignando a cada módulo a un framework específico con su propio conjunto de configuraciones y variables, para que no se vean afectadas las pruebas del otro módulo.

Se realizará un programa que utilice dos de las librerías de testing más famosas en Python, una es la librería **nose** y la otra es la librería **unittest**, identificando el error generado al usar ambas.

- a) **Unittest Testing:** Siguiendo las convenciones de la librería de pruebas **unittest**, realiza y ejecuta pruebas automatizadas a través de una función llamada **test_algo()**. La librería **unittest** está muy enfocada a la programación orientada a objetos (POO).
- a. **unittest:** Es una biblioteca de Python que se utiliza para escribir y ejecutar pruebas unitarias. Proporciona una forma estándar de organizar, ejecutar y verificar las pruebas en el código Python.
 - i. **Estructura de pruebas organizada:** Proporciona una estructura organizada para escribir pruebas. Se pueden definir conjuntos de pruebas, casos de prueba y métodos de prueba individuales, aunque es importante remarcar que su programación es algo complicada ya que está completamente basada en la POO, por lo cual se debe crear una clase para declarar su código.
 - ii. **Escritura de pruebas:** Permite escribir pruebas de manera clara y legible utilizando métodos y aserciones proporcionados por la librería **unittest**, facilitando así la forma en la que se especifican las condiciones de prueba y las expectativas de los resultados.
 - iii. **Descubrimiento y ejecución automática de pruebas:** Los métodos de la librería **unittest** tienen la capacidad de descubrir automáticamente los casos de prueba y métodos de prueba en el código para luego ejecutarlos. Esto significa que no se necesita llamar manualmente cada prueba, lo que facilita la ejecución de todas las pruebas de manera eficiente.
 - iv. **Aserciones integradas:** Proporciona un conjunto de aserciones integradas que permiten verificar automáticamente los resultados de las pruebas. Estas aserciones incluyen comparaciones numéricas, verificación de igualdad, comprobación de existencia, entre otros.
 - v. **Informes de resultados:** Al terminar las pruebas se generan informes detallados sobre los resultados obtenidos, lo que facilita la identificación de pruebas exitosas y fallidas, permitiendo una depuración más sencilla.
- b) **Nose Testing:** Siguiendo las convenciones de la librería de pruebas **nose**, que extiende y mejora las capacidades de la librería **unittest**, realiza y ejecuta pruebas automatizadas a través de una función llamada **test_otra_cosa()**. La librería **unittest** está muy enfocada a la programación orientada a objetos (POO), mientras que la librería **nose** no, por lo cual es más sencilla de utilizar, esa es una de sus grandes ventajas.
- a. **nose (también conocido como nose2):** Es otro marco de prueba para Python que se basa en la librería **unittest** pero proporciona una sintaxis más sencilla y algunas características adicionales. Al igual que **pytest**, **nose** también admite la automatización de pruebas unitarias, la ejecución paralela y la parametrización de pruebas. Además, permite la escritura de pruebas más concisas y ofrece una amplia gama de complementos y extensiones para personalizar y mejorar el proceso de las pruebas.
 - i. **Facilidad en la estructura de pruebas:** Proporciona una estructura para escribir pruebas de forma más sencilla a comparación de las pruebas escritas con la librería **unittest**, ya que su programación no está completamente basada en la POO, por lo cual sus funciones de pruebas se declaran de forma usual.
 - ii. **Plugins y extensiones:** Admite la integración de plugins y extensiones personalizadas, lo que permite ampliar las funcionalidades del marco de

pruebas. Hay una amplia variedad de plugins disponibles que pueden agregar características como generación de informes HTML, cobertura de código, pruebas de rendimiento, etc.

- iii. **Ejecución más rápida:** Mejora la velocidad de ejecución de las pruebas en comparación con la librería **unittest**.
- iv. **Mayor flexibilidad:** Ofrece una sintaxis más flexible para la escritura de pruebas y aserciones. También admite el uso de decoradores en las pruebas.

Código Python – Visual Studio Code (Logo Azul)

```
# -*- coding: utf-8 -*-

#En Python se introducen comentarios de una sola linea con el simbolo #.
#La primera línea de código incluida en este programa se conoce como declaración de codificación o codificación
#de caracteres. Al especificar utf-8 (caracteres Unicode) como la codificación, nos aseguramos de que el archivo
#pueda contener caracteres especiales, letras acentuadas y otros caracteres no ASCII sin problemas, garantizando
#que Python interprete correctamente esos caracteres y evite posibles errores de codificación.
#Se puede detener una ejecución con el comando [CTRL] + C puesto en consola, con el comando "cls" se borra su
#historial y en Visual Studio Code con el botón superior izquierdo de Play se corre el programa.
#Para comentar en Visual Studio Code varias líneas de código se debe pulsar:
#[CTRL] + K (VSCode queda a la espera). Después pulsa [CTRL] + C para comentar y [CTRL] + U para descomentar.

#PRUEBAS UNITARIAS:

import unittest #unittest: Librería orientada a objetos para ejecutar y analizar pruebas unitarias.
import nose #nose: Extensión de la librería unittest, simplificando su código, entre otras mejoras adicionales.
#Usar dos librerías distintas para realizar pruebas unitarias en un mismo programa puede resultar complicado y
#propenso a errores, ya que los frameworks de pruebas están diseñados para proporcionar una estructura
#coherente y consistente para las pruebas, por lo que mezclar dos o más frameworks puede generar inconsistencias
#y conflictos. Si se desea utilizar librerías distintas para diferentes pruebas en un mismo programa, es
#recomendable separar las pruebas en módulos o archivos diferentes, asignando a cada módulo a un framework
#específico con su propio conjunto de configuraciones y variables, para que no se vean afectadas las pruebas
#del otro módulo.

#Se realizará un programa que utilice dos de las librerías de testing más famosas en Python, una es la librería
#nose y la otra es la librería unittest, identificando el error generado al usar ambas.

#ERROR AL USAR DOS FRAMEWORKS EN EL MISMO ARCHIVO: En el siguiente archivo se verifica el resultado obtenido al
#sumar dos valores distintos:

# - Prueba unittest: Se suman dos valores numéricos y se comprueba que su suma sea igual a 12.
# - El error causado por utilizar las dos librerías unittest y nose en el mismo archivo es que al forzar
# un error en la prueba unittest, se lanzará una excepción tanto para nose como para unittest, cuando en
# realidad solo se debería lanzar para unittest, no para nose.
# - Prueba nose: Se concatenan dos valores string y se comprueba el resultado diga "HolaMundo".
# - Al forzar un error en la prueba nose, si se generará una excepción correcta en nose, y no se generará
# una para unittest, así es como debería ejecutarse el archivo.

#Por lo tanto, el error es obtenido cuando falla la prueba de unittest, ya que erróneamente arroja una excepción
```

```

#para nose también.

#CONDICIÓN DE LAS PRUEBAS UNITTEST Y NOSE:
#mi_funcion(): Función propia que realiza una suma de dos variables distintas, definiendo así la operación que
#quieren comprobar las pruebas unitarias unittest y nose.
def mi_funcion(a, b):
    return a + b

#PRUEBAS UNITARIAS AUTOMÁTICAS CON LA LIBRERÍA UNITTEST: La librería unittest está muy enfocada a la
#programación orientada a objetos (POO), por lo cual, para utilizarla es necesario crear una clase propia que
#herede de la clase TestCase, perteneciente a la librería unittest.
class MyTestCase(unittest.TestCase):
    #setUp(): Método que pertenece a las herramientas incluidas con la librería unittest, este sirve para
    #indicar los atributos que se utilizan al realizar las pruebas unitarias y se ejecuta antes de cada prueba
    #para configurar su estado inicial.
    def setUp(self):
        #ATRIBUTOS DE LA CLASE: Palabra reservada self.
        #Configuración inicial para cada prueba unittest, con la palabra reservada self se está accediendo a la
        #instancia de la clase actual, osea la clase MyTestCase, para así asignar como atributo los siguientes
        #valores:
        self.parametro3 = 10
        self.parametro4 = 3    #Si la variable es diferente a 2, ocurre un error en unittest y nose.
    #tearDown(): Método que pertenece a las herramientas incluidas con la librería unittest, este sirve para
    #limpiar o reiniciar el valor de los atributos utilizados al realizar pruebas unitarias.
    def tearDown(self):
        #ATRIBUTOS DE LA CLASE: Palabra reservada self.
        #Limpieza después de cada prueba unittest, con la palabra reservada self se está accediendo a la
        #instancia de la clase actual, osea la clase MyTestCase, para así asignar como atributo los siguientes
        #valores:
        self.parametro3 = None
        self.parametro4 = None

#PRUEBA UNITARIA PERSONALIZADA UNITTEST:
#test_algo(): Las funciones que realizan pruebas unitarias a través de la librería unittest se deben
#encontrar dentro de una clase propia que herede de TestCase. Dentro de la función se declaran los métodos
#que realizarán pruebas unitarias personalizadas. Es muy importante mencionar que el nombre de todas las
#funciones que se creen con este fin deben empezar con la palabra "test", para que todas puedan ser
#ejecutadas automáticamente cuando en el método main se corran las pruebas unitarias unittest.
def test_algo(self):
    # Acceder a los parámetros configurados en setUp()
    print("Test Unittest")
    resultado = mi_funcion(self.parametro3, self.parametro4)

```

```

self.assertEqual(resultado, 12)

#PRUEBAS UNITARIAS AUTOMÁTICAS CON LA LIBRERÍA NOSE: Una de las mejoras de la librería nose en comparación con
#unittest, es que su programación es más sencilla, ya que no se debe utilizar una clase para su ejecución, sus
#funciones se declaran de forma normal y se ejecutan de forma manual.
#configInicialNose(): Función propia que indica los atributos a utilizar en las pruebas unitarias nose, este se
#ejecuta antes de cada prueba para configurar su estado inicial, así como lo hacía el método setUp() de la
#librería unittest, pero a diferencia de ese método, todo este proceso se ejecuta de forma manual.
def configInicialNose():
    #MODIFICADORES DE ACCESO: Palabra reservada global.
    #Configuración inicial para cada prueba nose, con la palabra reservada global se está accediendo al
    #modificador de acceso de una variable, para así lograr que se pueda acceder a ella desde fuera de la
    #función o método:
    global parametro1, parametro2
    parametro1 = "Hola"
    parametro2 = "Mundo"          #Si la variable es diferente a "Mundo", ocurre un error en nose.
#limpiezaVariables(): Función propia que limpia o reinicia el valor de los atributos utilizados al realizar
#pruebas unitarias nose, este se ejecuta después de cada prueba para limpiar los valores asignados previamente
#en su estado inicial, así como lo hacía el método tearDown() de la librería unittest, pero a diferencia de ese
#método, todo este proceso se ejecuta de forma manual.
def limpiezaVariables():
    #MODIFICADORES DE ACCESO: Palabra reservada global.
    #Limpieza después de cada prueba nose, con la palabra reservada global se está accediendo al modificador de
    #acceso de una variable, para así lograr que se pueda acceder a ella desde fuera de la función o método:
    global parametro1, parametro2
    parametro1 = None
    parametro2 = None

#PRUEBA UNITARIA PERSONALIZADA NOSE:
#test_algo(): Las funciones que realizan pruebas unitarias a través de la librería nose deben declarar los
#métodos que realizan sus pruebas unitarias personalizadas. Es muy importante mencionar que el nombre de todas
#las funciones que se creen con este fin deben empezar con la palabra "test", para que todas puedan ser
#ejecutadas automáticamente cuando en el método main se corran las pruebas unitarias nose, de la misma forma
#como se declaran las funciones que realizan pruebas unitarias unittest.
def test_otra_cosa():
    print("Test Nose")
    # Configuración inicial para cada prueba en nose
    configInicialNose()
    # Acceder a los parámetros configurados en setUp()
    resultado = mi_funcion(parametro1, parametro2)
    assert resultado == "HolaMundo"
    limpiezaVariables()

```

```

#__name__ == '__main__': Método main, esta función es super importante ya que sirve para instanciar las clases del
#programa y ejecutar sus métodos, en python pueden existir varios métodos main en un solo programa, aunque no es
#una buena práctica.

#Ejecutar las pruebas dentro del método main permite ejecutarlas automáticamente cuando el archivo se ejecute
#como archivo principal, mientras que ejecutar las pruebas fuera del método main proporciona más control sobre
#cuándo y cómo se ejecutan las pruebas, lo que puede ser útil en escenarios más complejos o personalizados.
if __name__ == "__main__":
    #MANEJO DE EXCEPCIONES: Es una parte de código que se conforma de dos partes, try y except:
    # - Primero se ejecuta el código que haya dentro del try y si es que llegara a ocurrir una excepción durante
    #   su ejecución, el programa brinca al código del except
    # - En la parte de código donde se encuentra la palabra reservada except, se ejecuta cierta acción cuando
    #   ocurra el error.
    #Se utiliza esta arquitectura de código cuando se quiera efectuar una acción donde se espera que pueda
    #ocurrir un error durante su ejecución y en este caso se usa para intentar manejar de forma separada las
    #excepciones causadas en las pruebas unittest y nose, aunque solo funciona parcialmente.
    #Ejecutar pruebas unittest:
    print("-----Unit Test Testing-----")
    try:
        #EJECUCIÓN DE PRUEBAS UNITARIAS UNITTEST DE FORMA AUTOMATIZADA:
        #Creación de objeto o instancia de la clase TestSuite(), perteneciente a la librería unittest.
        suite = unittest.TestSuite()
        #TestSuite es un objeto que permite agrupar y organizar todas las pruebas unitarias creadas a partir de
        #la librería unittest. Proporcionando varios métodos que permiten construir el conjunto de pruebas de
        #manera flexible y escalable:
        # - addTest(test): Agrega una prueba individual al conjunto de pruebas.
        #
        # - addTests(tests): Agrega varias pruebas al conjunto de pruebas.
        #
        # - addTestSuite(test_suite): Agrega otro objeto TestSuite al conjunto de pruebas.
        #
        # - countTestCases(): Retorna el número total de casos de prueba en el conjunto.
        # - tests(): Retorna una lista de todas las pruebas en el conjunto, en el orden en que fueron agregadas.
        # - debug(): Ejecuta las pruebas en el conjunto en modo de depuración.
        # - getTestCaseNames(testcase_class): Retorna una lista de los nombres de los casos de prueba en una
        #   clase de caso de prueba dada.
        # - __iter__(): Retorna un iterador para recorrer todas las pruebas en el conjunto.
        # - __str__(): Retorna una representación de cadena legible del conjunto de pruebas.
        suite.addTest(MyTestCase('test_algo'))
        #Creación de objeto o instancia de la clase TextTestRunner(), perteneciente a la librería unittest.
        #TestSuite es un objeto que se utiliza para ejecutar pruebas unitarias y generar un informe de
        #resultados en formato de texto que se muestre en consola:
        # - run(result): Ejecuta las pruebas del conjunto result y genera informes que indican si cada prueba ha

```



```

# tenido éxito o ha fallado, además de mostrar en cuánto tiempo fue ejecutada cada prueba. Dichos
# informes se presentan en un formato legible para los humanos en consola o en un archivo de texto y
# se conforman de la siguiente estructura:
#
#     - Nombre de la prueba: Se muestra información sobre cada prueba individual ejecutada, como
#       el nombre de la prueba y la clase que la está ejecutando, normalmente esto se muestra
#       solamente cuando ocurre una excepción.
#
#           test_nombre_de_prueba          (clase_de_prueba)
#
#           -----
#
#           Mensaje de la excepción.
#
#           -----
#
#     - Estado de la prueba: Muestra si la prueba ha pasado correctamente o ha fallado. Si una
#       prueba falla, se mostrará un mensaje de error específico para cada prueba.
#
#           OK:      La prueba fue aprobada.
#
#           F o E: La prueba ha fallado y arrojado una excepción.
#
#     - Tiempo de ejecución: Indica la cantidad total de pruebas que se ejecutaron y el tiempo que
#       tomó en ejecutarse.
#
#           -----
#
#           Ran      "numero_pruebas"      tests in      "tiempo" s
#
#     - Estadísticas globales: Después de mostrar los detalles de cada prueba, se presenta un
#       resumen general de los resultados, mostrando la misma información de arriba pero ahora
#       para la suma de todas las pruebas.
#
# - runTest(test): Ejecuta una prueba individual especificada dentro del conjunto.
unittest.TextTestRunner().run(suite)

#Para identificar el tipo de excepción que ha ocurrido y utilizarlo en la instrucción except, se puede
#utilizar la clase Exception, que es una clase incorporada en Python utilizada para describir todos los
#tipos de excepciones, luego de colocar el nombre de la clase Exception se usa la palabra reservada "as"
#seguida de un nombre de variable, esto nos permitirá acceder a la instancia de la excepción y
#utilizarla dentro del except, aunque esto al ejecutar pruebas unitarias no funciona de nada, ya que estas
#líneas de código son ignoradas y el error es mostrado siguiendo las convenciones de las librerías de
#pruebas unittest y/o nose.
except Exception as e:
    print("Ocurrió el siguiente tipo de error: ", type(e).__name__)
    print("Error en unittest:", e)

#Ejecutar pruebas nose con métodos de configuración inicial y limpieza implementados:
print("\n\n-----Nose Testing-----")
try:
    #nose.runmodule(): Método que busca y ejecuta automáticamente todas las pruebas unitarias que utilicen
    #la librería de pruebas nose de Python, para ello el nombre de todas las funciones que ejecuten pruebas
    #unitarias con nose deben empezar con la palabra "test", además genera un informe de resultados en la
    #salida estándar, ya sea en consola o en un archivo de texto. Dichos informes se presentan en un formato
    #legible para los humanos en consola o en un archivo de texto y se conforman de la siguiente estructura:
    #
    # - Nombre de la prueba: Indica el nombre de la prueba unitaria que se está ejecutando.
    #
    #     test_nombre_de_prueba          (clase_de_prueba)
    #
    #     -----

```

```

#     Mensaje de la excepción.
#
# -----
# - Estado de la prueba: Muestra si la prueba ha pasado correctamente o ha fallado. Si una prueba falla,
#   se mostrará un mensaje de error específico para cada prueba.
#
#     OK:   La prueba fue aprobada.
#
#     F o E: La prueba ha fallado y arrojado una excepción.
#
#     - Tiempo de ejecución: Indica la cantidad total de pruebas que se ejecutaron y el tiempo que
#       tomó en ejecutarse.
#
# -----
#
#     Ran          "numero_pruebas"      tests in    "tiempo"  s
#
# - Estadísticas globales: Después de mostrar los detalles de cada prueba, se presenta un resumen
#   general de los resultados, mostrando la misma información de arriba pero ahora para la suma de todas
#   las pruebas.
nose.runmodule()
except Exception as e:
    print("Error en nose:", e)

```

Resultado del Código Python: Pruebas *Unittest* y *Nose* aprobadas

```

PROBLEMAS  SALIDA  TERMINAL  CONSOLA DE DEPURACIÓN

PS C:\Users\diego\OneDrive\Documents\Aprendiendo\Python\1.-Instrumentación Virtual> & D:/Users/diego/AppData/Local/Programs/Python/Python39/python.exe
"c:/Users/diego/OneDrive/Documents/Aprendiendo/Python/1.-Instrumentación Virtual/Instrumentación Virtual Aplicada/Tareas/Tarea 1/Tareas C#/Ej_18.py"

-----Unit Test Testing-----
Test Unittest
.
-----
Ran 1 test in 0.000s

OK

-----Nose Testing-----
..
-----
Ran 2 tests in 0.000s

OK
PS C:\Users\diego\OneDrive\Documents\Aprendiendo\Python\1.-Instrumentación Virtual>

```

Resultado del Código Python: Prueba *Unittest* aprobada y *Nose* reprobada

```

PROBLEMAS  SALIDA  TERMINAL  CONSOLA DE DEPURACIÓN

PS C:\Users\diego\OneDrive\Documents\Aprendiendo\Python\1.-Instrumentación Virtual> & D:/Users/diego/AppData/Local/Programs/Python/Python39/python.exe
"c:/Users/diego/OneDrive/Documents/Aprendiendo/Python/1.-Instrumentación Virtual/Instrumentación Virtual Aplicada/Tareas/Tarea 1/Tareas C#/Ej_18.py"

-----Unit Test Testing-----
Test Unittest
.
-----
Ran 1 test in 0.000s

OK

-----Nose Testing-----
.F
-----
FAIL: __main__.test_otra_cosa
-----
Traceback (most recent call last):
  File "D:/Users/diego/AppData/Local/Programs/Python/Python39/lib/site-packages/nose/case.py", line 198, in runTest
    self.test(*self.arg)
  File "c:/Users/diego/OneDrive/Documents/Aprendiendo/Python/1.-Instrumentación Virtual/Instrumentación Virtual Aplicada/Tareas/Tarea 1/Tareas C#/Ej_18
.py", line 123, in test_otra_cosa
    assert resultado == "HolaMundo"
AssertionError:
----- >> begin captured stdout << -----
Test Nose
----- >> end captured stdout << -----
-----
Ran 2 tests in 0.001s

FAILED (failures=1)
PS C:\Users\diego\OneDrive\Documents\Aprendiendo\Python\1.-Instrumentación Virtual>

```

Resultado del Código Python: Pruebas *Unittest* y *Nose* reprobadas, Error por usar 2 librerías

```
PROBLEMAS  SALIDA  TERMINAL  CONSOLA DE DEPURACIÓN

PS C:\Users\diego\OneDrive\Documents\Aprendiendo\Python\1.-Instrumentación Virtual> & D:/Users/diego/AppData/Local/Programs/Python/Python39/python.exe
"c:/Users/diego/OneDrive/Documents/Aprendiendo/Python/1.-Instrumentación Virtual/Instrumentación Virtual Aplicada/Tareas/Tarea 1/Tareas C#/Ej_18.py"
-----Unit Test Testing-----
Test Unittest
F
=====
FAIL: test_algo (__main__.MyTestCase)
-----
Traceback (most recent call last):
  File "c:/Users/diego/OneDrive/Documents/Aprendiendo/Python/1.-Instrumentación Virtual/Instrumentación Virtual Aplicada/Tareas/Tarea 1/Tareas C#/Ej_18
.py", line 81, in test_algo
    self.assertEqual(resultado, 12)
AssertionError: 13 != 12

-----
Ran 1 test in 0.001s

FAILED (failures=1)

-----Nose Testing-----
F.
=====
FAIL: test_algo (__main__.MyTestCase)
-----
Traceback (most recent call last):
  File "c:/Users/diego/OneDrive/Documents/Aprendiendo/Python/1.-Instrumentación Virtual/Instrumentación Virtual Aplicada/Tareas/Tarea 1/Tareas C#/Ej_18
.py", line 81, in test_algo
    self.assertEqual(resultado, 12)
AssertionError: 13 != 12
----- >> begin captured stdout << -----
Test Unittest
----- >> end captured stdout << -----
-----
Ran 2 tests in 0.001s

FAILED (failures=1)
PS C:\Users\diego\OneDrive\Documents\Aprendiendo\Python\1.-Instrumentación Virtual> |
```

7.- Diferenciación numérica: Derivada c/ la fórmula diferencial/Testing: Assertion y Nose

La siguiente fórmula general puede utilizarse para hallar una derivada aproximada de una función matemática $f(x)$ si h es lo suficientemente pequeña, además se debe comprobar que se puede realizar dos pruebas unitarias distintas en un mismo archivo, siempre y cuando no se utilicen dos librerías diferentes:

$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h}$$

Pseudocódigo:

- Escribe una función **diff(f, x, h = 1e-5)** que devuelva la aproximación de la derivada de una función matemática representada por una función **f(x)**.
- Aplica la función **diff()** para diferenciar las siguientes ecuaciones, que están declaradas por medio de funciones anónimas dentro del código Python, luego resta el resultado obtenido en la diferencial menos el que se debió haber obtenido al realizar la derivada de la función al ser evaluada en el punto de x indicado debajo, con esta operación se calcula el error manualmente:

$$f(x) = e^x; \quad x = 0$$

$$f(x) = e^{-2x^2}; \quad x = 0$$

$$f(x) = \cos x; \quad x = 2\pi$$

$$f(x) = \ln x; \quad x = 1$$



- c) **Assertion Testing:** Escribe una función `assertionTesting()` que verifique la implementación de la función `diff()` aplicada a las funciones del inciso b). En este ejemplo no se está utilizando ninguna biblioteca de pruebas como `unittest` o `nose`, en su lugar se está realizando una prueba básica y manual utilizando la declaración `assert`.
- d) **Nose Testing:** Siguiendo las convenciones de la librería de pruebas `nose`, que extiende y mejora las capacidades de la librería `unittest`, realiza y ejecuta pruebas automatizadas a través de una función llamada `test_nose_application()`. La librería `unittest` está muy enfocada a la programación orientada a objetos (POO), mientras que la librería `nose` no, por lo cual es más sencilla de utilizar, esa es una de sus grandes ventajas.
 - a. **nose (también conocido como nose2):** Es otro marco de prueba para Python que se basa en la librería `unittest` pero proporciona una sintaxis más sencilla y algunas características adicionales. Al igual que `pytest`, `nose` también admite la automatización de pruebas unitarias, la ejecución paralela y la parametrización de pruebas. Además, permite la escritura de pruebas más concisas y ofrece una amplia gama de complementos y extensiones para personalizar y mejorar el proceso de las pruebas.
 - i. **Facilidad en la estructura de pruebas:** Proporciona una estructura para escribir pruebas de forma más sencilla a comparación de las pruebas escritas con la librería `unittest`, ya que su programación no está completamente basada en la POO, por lo cual sus funciones de pruebas se declaran de forma usual.
 - ii. **Plugins y extensiones:** Admite la integración de plugins y extensiones personalizadas, lo que permite ampliar las funcionalidades del marco de pruebas. Hay una amplia variedad de plugins disponibles que pueden agregar características como generación de informes HTML, cobertura de código, pruebas de rendimiento, etc.
 - iii. **Ejecución más rápida:** Mejora la velocidad de ejecución de las pruebas en comparación con la librería `unittest`.
 - iv. **Mayor flexibilidad:** Ofrece una sintaxis más flexible para la escritura de pruebas y aserciones. También admite el uso de decoradores en las pruebas.

Utiliza $h = 0,01$ en cada caso, escribe el error y señala la diferencia entre la derivada exacta y el resultado obtenido con la diferenciación de las funciones. Reúne estos cuatro ejemplos en una función llamada `application()`.

Código Python – Visual Studio Code (Logo Azul)

```
# -*- coding: utf-8 -*-

#En Python se introducen comentarios de una sola linea con el simbolo #.
#La primera línea de código incluida en este programa se conoce como declaración de codificación o codificación
#de caracteres. Al especificar utf-8 (caracteres Unicode) como la codificación, nos aseguramos de que el archivo
#pueda contener caracteres especiales, letras acentuadas y otros caracteres no ASCII sin problemas, garantizando
#que Python interprete correctamente esos caracteres y evite posibles errores de codificación.
#Se puede detener una ejecución con el comando [CTRL] + C puesto en consola, con el comando "cls" se borra su
#historial y en Visual Studio Code con el botón superior izquierdo de Play se corre el programa.
#Para comentar en Visual Studio Code varias líneas de código se debe pulsar:
```

```

#[CTRL] + K (VSCode queda a la espera). Después pulsa [CTRL] + C para comentar y [CTRL] + U para descomentar.

import numpy as np #numpy: Librería que realiza operaciones matemáticas complejas (matriciales).
import tabulate #tabulate: Librería que permite crear tablas con varios formatos conocidos.
import nose.tools #nose: Extensión de la librería unittest, brindando características y mejoras adicionales.
#Usar dos librerías distintas para realizar pruebas unitarias en un mismo programa puede resultar complicado y
#propenso a errores, ya que los frameworks de pruebas están diseñados para proporcionar una estructura
#coherente y consistente para las pruebas, por lo que mezclar dos o más frameworks puede generar inconsistencias
#y conflictos. Si se desea utilizar librerías distintas para diferentes pruebas en un mismo programa, es
#recomendable separar las pruebas en módulos o archivos diferentes, asignando a cada módulo a un framework
#específico con su propio conjunto de configuraciones y variables, para que no se vean afectadas las pruebas
#del otro módulo.

#6.- DIFERENCIACIÓN NUMÉRICA: Derivada con la fórmula diferencial - Assertion, Unittesting y Nose Testing.
#La fórmula general puede utilizarse para hallar una derivada aproximada de una función matemática f(x)
#si h es lo suficientemente pequeña:
#f'(x) ≈ (f(x+h)-f(x-h))/2*h

#a) Escribe una función diff(f, x, h = 1e-5) que devuelva una aproximación de la derivada de una función
#matemática representada por una función f(x).
def diff(f, x, h = 1e-5):
    #El 2 del numerador se expresa con un valor decimal para que se entienda que el valor que devuelve df es
    #float, osea un número decimal.
    df = (f(x+h) - f(x-h))/(2.0*h) #f'(x) ≈ (f(x+h)-f(x-h))/2*h
    return df

#b) Aplica la función diff() para diferenciar las siguientes ecuaciones, que están declaradas por medio de
#funciones anónimas dentro del código Python, luego resta el resultado obtenido en la diferencial menos el
#que se debió haber obtenido al realizar la derivada de la función al ser evaluada en el punto de x indicado
#debajo, con esta operación se calcula el error manualmente:
# 1.- f(x) = e^x;          x = 0;          f'(0) = 1*e^x = 1*e^0 = 1
# 2.- f(x) = e^(-2x^2);    x = 0;          f'(0) = -4x*e^(-2x^2) = -4(0)*e^(-2(0)^2) = 0*e^0 = 0
# 3.- f(x) = cos(x);      x = 2π;         f'(0) = -sin(x) = -sin(0) = -sin(0) = 0
# 4.- f(x) = ln(x);       x = 1;          f'(0) = 1/x = 1/1 = 1

#Función Anónima: En este tipo de funciones se utiliza la palabra reservada "lambda" para crear funciones
#sin nombre, mejor conocidas como "funciones lambda" o "expresiones lambda", estas son declaradas por medio
#de la siguiente sintaxis:
# variable = lambda parámetros: operación_función
#numpy.exp(x) = e^x = e**x: Método que devuelve el valor exponencial de un número con base "e".
#numpy.cos(x): Calcula la función coseno aplicada a una variable x, esperando que su valor esté en radianes.
#numpy.log(x): Calcula el logaritmo natural (logaritmo en base e) ln de un valor o una matriz de valores.
f1 = lambda x: np.exp(x)          #1.- f(x) = e^x

```

```

f2 = lambda x: np.exp(-2.0*(x**2))          #2.-  $f(x) = e^{-2x^2}$ 
f3 = lambda x: np.cos(x)                   #3.-  $f(x) = \cos(x)$ 
f4 = lambda x: np.log(x)                   #4.-  $f(x) = \ln(x)$ 

#DIFERENCIACIÓN DE LAS FUNCIONES:
x = [0, 0, 2*np.pi, 1] #Puntos de x donde se evalúan las diferenciales de las funciones f1, f2, f3 y f4.
resultado_derivada = [1, 0, 0, 1] #Resultados exactos de todas las derivadas evaluadas en los puntos de x.
resultado_diferencial = [] #Lista que guardará el resultado de las derivadas hechas con la diferencial.
#append(): Método que sirve para agregar valores a una lista, tupla, numpy array o diccionario.
#Se aplica la función diferencial a cada función para aproximar su derivada y esto se realiza evaluando cada
#función en su respectivo valor de x, que es distinto para cada una, además se declara que el valor de
#dx = h, se cambie al que estaba indicado por default a ser dx = h = 0.01.
dx = 0.01
resultado_diferencial.append(diff(f1, x[0], dx))
resultado_diferencial.append(diff(f2, x[1], dx))
resultado_diferencial.append(diff(f3, x[2], dx))
resultado_diferencial.append(diff(f4, x[3], dx))

#c) Assertion Testing: Escribe una función assertionTesting() que verifique la implementación de la función
#diff() aplicada a las funciones del inciso b). En este ejemplo no se está utilizando ninguna biblioteca de
#pruebas como unittest o nose, en su lugar se está realizando una prueba básica y manual utilizando la
#declaración assert.
def assertionTesting(umbral_assertion_testing):
    #IMPRIMIR LOS RESULTADOS EN UNA TABLA Y CALCULAR EL ERROR OBTENIDO EN ELLAS CON LA RESTA INDICADA EN C):
    error = [] #Error calculado al restar el resultado de: diferencial_f(x) - derivada_f(x).
    nombre_funciones = ['e^x', 'e^(-2x^2)', 'cos(x)', 'ln(x)'] #Nombre de las columnas en la tabla de funciones.
    #Lista vacía donde se almacenarán el nombre de la función, el valor x con el que se realizó la derivada, su
    #resultado y el error que se obtuvo al restar ambas para poder mandar la matriz al método tabulate y mostrar
    #la tabla en consola.
    tabla = []

    #CÁLCULO DEL ERROR PARA CADA UNA DE LAS FUNCIONES: Para ello se resta el valor obtenido de la diferencial
    #menos el valor que debió haber obtenido al aplicar su derivada, después lo que se hace es sacar el valor
    #absoluto de este valor, ya que no puede existir errores negativos.
    #abs(): Método que saca el valor absoluto de un número, volviéndolo positivo aunque sea originalmente
    #negativo.
    error.append(abs(resultado_diferencial[0] - resultado_derivada[0]))
    error.append(abs(resultado_diferencial[1] - resultado_derivada[1]))
    error.append(abs(resultado_diferencial[2] - resultado_derivada[2]))
    error.append(abs(resultado_diferencial[3] - resultado_derivada[3]))

    #BUCLE FOR: En Python no se utilizan llaves de apertura o cierre para la sintaxis de un bucle, solamente se
    #utilizan dos puntos para indicar su inicio y tabuladores para diferenciar qué es lo que está dentro o fuera

```

```

#de él. Además, después de la palabra reservada "for" se declara una variable local numérica entera que solo
#existirá dentro del bucle y será la que cuente desde 0 por default o desde el primer número indicado dentro
#del paréntesis hasta el extremo indicado en el segundo parámetro que se encuentra dentro del paréntesis de
#la palabra reservada range() para terminar el bucle. En otras palabras, dentro del paréntesis de la palabra
#reservada "range()" se coloca el inicio y final del conteo para indicar cuantas veces se ejecutará el
#bucle.

#En específico este bucle for se utiliza para crear la matriz que contenga todos los datos a mostrarse en la
#tabla creada con la librería tabulate, otra forma de lograr lo mismo hubiera sido utilizar una variable
#intermedia a la que se le haya aplicado el método var_temp.append(i) y luego var_temp.append(j) para crear
#una sublista al aplicar finalmente el método matriz.append(var_temp).

#Aunque en vez de hacer todo eso se puede aplicar directamente el método matriz.append([i, j]) y así crear
#las sublistas.

for i in range(0, len(nombre_funciones)):
    tabla.append([nombre_funciones[i], x[i], resultado_derivada[i] , resultado_diferencial[i], error[i]])

#CREAR UNA TABLA CON LA LIBRERÍA TABULATE: La tabla se mostrará hasta que cierre la ventana de la gráfica.
#tabulate.tabulate(): Método que sirve para crear una lista con los valores de un vector, dicho vector debe
#contener listas anidadas en cada una de sus posiciones para que estas sean agrupadas en cada fila de la
#lista.

#Las tablas creadas con el método tabulate aceptan los siguientes parámetros:

# - tabular_data: Este parámetro especifica los datos a mostrar. Puede ser una lista de listas, una lista de
# diccionarios, una lista de tuplas, o una estructura de datos similar.

# - headers: Es un parámetro opcional que se utiliza para especificar los encabezados de las columnas de la
# tabla. Puede ser una lista de cadenas que representen los nombres de las columnas.

# - tablefmt: Es un parámetro opcional que indica el formato deseado para la tabla generada.

#     - "plain": Sin formato adicional.
#     - "simple": Formato simple con líneas horizontales.
#     - "grid": Formato con bordes de cuadrícula.
#     - "pipe": Formato con delimitadores de tubería.
#     - "orgtbl": Formato de tabla org.

# - numalign: Este parámetro se utiliza para alinear los números en las columnas de la tabla. Puede tener
# los siguientes valores:
#     - "left": Alinea los números a la izquierda.
#     - "center": Centra los números en la columna.
#     - "right": Alinea los números a la derecha.

# - stralign: Este parámetro se utiliza para alinear las cadenas de texto en las columnas de la tabla. Puede
# tener los siguientes valores:
#     - "left": Alinea las cadenas de texto a la izquierda.
#     - "center": Centra las cadenas de texto en la columna.
#     - "right": Alinea las cadenas de texto a la derecha.

# - missingval: Este parámetro se utiliza para establecer el valor a mostrar cuando los datos son nulos o no
# están disponibles. Puede ser cualquier valor válido, como una cadena de texto, un número, etc. Cuando se
# encuentre un valor nulo en los datos, se mostrará el valor especificado en missingval.

print(tabulate.tabulate(tabla, headers = ['f(x)', 'Valor de x', 'Derivada evaluada en x', 'Diferencial evaluada en
x', 'Error'], tablefmt = "grid", numalign = "center"))

```

```

#PRUEBAS UNITARIAS AUTOMÁTICAS CON LA DECLARACIÓN ASSERT (ASSERTION TESTING):
#{index:formato}.format(variable): El método format() permite asignar un formato a un string, para ello
#primero se debe usar un especificador de formato, que sigue la siguiente sintaxis:
# - {caracteres_que_se_quieren_afectar:formato}
#   - {index:}: Para indicar los caracteres que se quieren afectar tenemos que tener en cuenta que las
#     posiciones de un string se manejan igual que las de una lista, tupla o diccionario, por lo cual se
#     empiezan a contar desde 0 si es que solamente se quiere afectar un solo string, si es que se
#     quieren afectar todos, no se pone nada.
#   - {:formato}: Se pueden agregar modificadores de formato para controlar la precisión decimal, el
#     relleno, la alineación y otros aspectos del formato. Algunos modificadores comunes incluyen:
#     - {:.0e}: El número se mostrará en notación científica con 0 dígitos después del punto decimal,
#       para ello primero se debe convertir a entero con el método float().
#     - {:.5f}: El número se mostrará en formato decimal con 5 dígitos después del punto, para ello
#       primero se debe convertir a entero con el método float().
#     - {:b}: El número se mostrará en formato binario, para ello primero se debe convertir a entero
#       con el método int().
#     - {:d}: El número se mostrará en formato de número decimal, para ello primero se debe convertir
#       a entero con el método int().
#     - {:o}: El número se mostrará en formato octal, para ello primero se debe convertir a entero con
#       el método int().
#     - {:x}: El número se mostrará en formato hexadecimal con minúsculas, para ello primero se debe
#       convertir a entero con el método int().
#     - {:X}: El número se mostrará en formato hexadecimal con mayúsculas, para ello primero se debe
#       convertir a entero con el método int().

notacion_cientifica = "{:.1e}".format(umbral_assertion_testing)
print("\n\n-----Assertion Testing-----")
print("Para pasar las pruebas de Assertion Testing el error debe ser menor o igual a: ", umbral_assertion_testing, " =
", notacion_cientifica)

#assert: La palabra reservada assert en Python se utiliza como una declaración de aserción. Su propósito es
#verificar si una condición dada es verdadera. Si la condición es falsa, se genera una excepción llamada
#AssertionError, la sintaxis de la operación de aserción o afirmación es la siguiente:
# - assert      condición,      mensaje_de_error
#   - condición: Es una expresión que se evalúa como verdadera o falsa.
#   - mensaje_de_error: Es un string opcional que se mostrará si la condición es falsa, si se llegara a
#     mostrar este mensaje cuando la condición es falsa, el programa terminaría su ejecución en esa
#     línea de código.
#     Este mensaje puede proporcionar información sobre el error o la aserción que falló.

assert (error[0] <= umbral_assertion_testing), "Error muy grande al aproximar la derivada de e^x por medio de su diferencial"
assert (error[1] <= umbral_assertion_testing), "Error muy grande al aproximar la derivada de e^(-2x^2) por medio de su
diferencial"

assert (error[2] <= umbral_assertion_testing), "Error muy grande al aproximar la derivada de cos(x) por medio de su
diferencial"

assert (error[3] <= umbral_assertion_testing), "Error muy grande al aproximar la derivada de ln(x) por medio de su
diferencial"

```



```

    print("Todas las pruebas pasaron correctamente.")

#Llamada a la función assertionTesting().
#Si la tolerancia se vuelve igual o menor a 1e-5, la prueba unitaria falla y se lanza una excepción.
umbralAssertion = 1e-4          #Tolerancia de comparación del Assertion Testing
assertionTesting(umbralAssertion)

#d) Nose Testing: Siguiendo las convenciones de la librería de pruebas nose, que extiende y mejora las
#capacidades de la librería unittest, realiza y ejecuta pruebas automatizadas a través de la función
#test_nose_application(). La librería unittest está muy enfocada a la programación orientada a objetos (POO),
#mientras que la librería nose no, por lo cuál es más sencilla de utilizar, esa es una de sus grandes ventajas.
#Si la tolerancia se vuelve igual o menor a 1e-4, la prueba unitaria falla y se lanza una excepción.

#PRUEBAS UNITARIAS AUTOMÁTICAS CON LA LIBRERÍA NOSE: Una de las mejoras de la librería nose en comparación con
#unittest, es que su programación es más sencilla, ya que no se debe utilizar una clase para su ejecución, sus
#funciones se declaran de forma normal y se ejecutan de forma manual.
#configInicialNose(): Función propia que indica los atributos a utilizar en las pruebas unitarias nose, este se
#ejecuta antes de cada prueba para configurar su estado inicial, así como lo hacía el método setUp() de la
#librería unittest, pero a diferencia de ese método, todo este proceso se ejecuta de forma manual.
def configInicialNose():
    #MODIFICADORES DE ACCESO: Palabra reservada global.
    #Configuración inicial para cada prueba nose, con la palabra reservada global se está accediendo al
    #modificador de acceso de una variable, para así lograr que se pueda acceder a ella desde fuera de la
    #función o método:
    global umbral_nose_testing
    umbral_nose_testing = 1e-5          #Tolerancia de comparación del Nose Testing

#limpiezaVariables(): Función propia que limpia o reinicia el valor de los atributos utilizados al realizar
#pruebas unitarias nose, este se ejecuta después de cada prueba para limpiar los valores asignados previamente
#en su estado inicial, así como lo hacía el método tearDown() de la librería unittest, pero a diferencia de ese
#método, todo este proceso se ejecuta de forma manual.
def limpiezaVariables():
    #MODIFICADORES DE ACCESO: Palabra reservada global.
    #Limpieza después de cada prueba nose, con la palabra reservada global se está accediendo al modificador de
    #acceso de una variable, para así lograr que se pueda acceder a ella desde fuera de la función o método:
    global umbral_nose_testing
    umbral_nose_testing = None          #Limpieza de la variable que almacena la tolerancia.

#test_nose_application(): Las funciones que realizan pruebas unitarias a través de la librería nose no se deben
#encontrar dentro de una clase, demostrando así su optimización en comparación con el código que utiliza la
#librería unittest al generar pruebas unitarias personalizadas. Aunque también es muy importante mencionar que
#el nombre de todas las funciones que se creen con este fin deben empezar con la palabra "test", de igual manera
#como se realiza al crear pruebas unitarias con la librería unittest.
def test_nose_application():

```

```

#nose.tools.assert_almost_equal(): Método que compara dos valores numéricos y verifica si están "casi"
#iguales dentro de una tolerancia dada. Esto es útil cuando se trabaja con cálculos numéricos y se espera
#que los resultados sean aproximados debido a errores de redondeo u otras limitaciones, el método recibe
#los siguientes parámetros:
# - nose.tools.assert_almost_equal(valor_real, valor_esperado, delta = Tolerancia)
#   - Si la diferencia entre los valores es menor a la tolerancia, la afirmación se considera verdadera
#     y la ejecución continúa sin problemas.
#   - Si la diferencia es mayor o igual a la tolerancia, se genera una excepción AssertionError,
#     indicando que los valores no están "casi" iguales.
nose.tools.assert_almost_equal(resultado_diferencial[0], resultado_derivada[0], delta = umbral_nose_testing)
nose.tools.assert_almost_equal(resultado_diferencial[1], resultado_derivada[1], delta = umbral_nose_testing)
nose.tools.assert_almost_equal(resultado_diferencial[2], resultado_derivada[2], delta = umbral_nose_testing)
nose.tools.assert_almost_equal(resultado_diferencial[3], resultado_derivada[3], delta = umbral_nose_testing)
limpiezaVariables() #Limpieza de variables para las pruebas nose.

#__name__ == '__main__': Método main, esta función es super importante ya que sirve para instanciar las clases del
#programa y ejecutar sus métodos, en python pueden existir varios métodos main en un solo programa, aunque no es
#una buena práctica.
#Ejecutar las pruebas dentro del método main permite ejecutarlas automáticamente cuando el archivo se ejecute
#como archivo principal, mientras que ejecutar las pruebas fuera del método main proporciona más control sobre
#cuándo y cómo se ejecutan las pruebas, lo que puede ser útil en escenarios más complejos o personalizados.
if __name__ == "__main__":
    #MANEJO DE EXCEPCIONES: Es una parte de código que se conforma de dos partes, try y except:
    # - Primero se ejecuta el código que haya dentro del try y si es que llegara a ocurrir una excepción durante
    #   su ejecución, el programa brinca al código del except
    # - En la parte de código donde se encuentra la palabra reservada except, se ejecuta cierta acción cuando
    #   ocurra el error.
    #Se utiliza esta arquitectura de código cuando se quiera efectuar una acción donde se espera que pueda
    #ocurrir un error durante su ejecución.
    #EJECUTAR LAS PRUEBAS UNITARIAS NOSE:
    try:
        configInicialNose() #Configuración inicial para cada prueba nose.
        #{index:formato}.format(variable): El método format() permite asignar un formato a un string, para ello
        #primero se debe usar un especificador de formato, que sigue la siguiente sintaxis:
        # - {caracteres_que_se_quieren_afectar:formato}
        #   - {index:}: Para indicar los caracteres que se quieren afectar tenemos que tener en cuenta que
        #     las posiciones de un string se manejan igual que las de una lista, tupla o diccionario, por lo
        #     cual se empiezan a contar desde 0 si es que solamente se quiere afectar un solo string, si es
        #     que se quieren afectar todos, no se pone nada.
        #   - {:formato}: Se pueden agregar modificadores de formato para controlar la precisión decimal, el
        #     relleno, la alineación y otros aspectos del formato. Algunos modificadores comunes incluyen:
        #     - {:.0e}: El número se mostrará en notación científica con 0 dígitos después del punto
        #     decimal, para ello primero se debe convertir a entero con el método float().

```

```

#         - {:.5f}: El número se mostrará en formato decimal con 5 dígitos después del punto, para ello
#             primero se debe convertir a entero con el método float().
#         - {:b}: El número se mostrará en formato binario, para ello primero se debe convertir a
#             entero con el método int().
#         - {:d}: El número se mostrará en formato de número decimal, para ello primero se debe
#             convertir a entero con el método int().
#         - {:b}: El número se mostrará en formato octal, para ello primero se debe convertir a entero
#             con el método int().
#         - {:x}: El número se mostrará en formato hexadecimal con minúsculas, para ello primero se
#             debe convertir a entero con el método int().
#         - {:X}: El número se mostrará en formato hexadecimal con mayúsculas, para ello primero se
#             debe convertir a entero con el método int().

notacion_cientifica_nose = "{:.1e}".format(umbral_nose_testing)
print("\n\n-----Nose
Testing-----")

print("Para pasar las pruebas de Nose Testing el error debe ser menor o igual a: ", umbral_nose_testing, " = ",
notacion_cientifica_nose)

#nose.runmodule(): Método que busca y ejecuta automáticamente todas las pruebas unitarias que utilicen
#la librería de pruebas nose de Python, para ello el nombre de todas las funciones que ejecuten pruebas
#unitarias con nose deben empezar con la palabra "test", además genera un informe de resultados en la
#salida estándar, ya sea en consola o en un archivo de texto. Dichos informes se presentan en un formato
#legible para los humanos en consola o en un archivo de texto y se conforman de la siguiente estructura:
# - Nombre de la prueba: Indica el nombre de la prueba unitaria que se está ejecutando.
#         test_nombre_de_prueba          (clase_de_prueba)
#         -----
#         Mensaje de la excepción.
#         -----
# - Estado de la prueba: Muestra si la prueba ha pasado correctamente o ha fallado. Si una prueba falla,
#     se mostrará un mensaje de error específico para cada prueba.
#         OK:      La prueba fue aprobada.
#         F o E: La prueba ha fallado y arrojado una excepción.
#         - Tiempo de ejecución: Indica la cantidad total de pruebas que se ejecutaron y el tiempo que
#             tomó en ejecutarse.
#         -----
#         Ran      "numero_pruebas"      tests in      "tiempo" s
# - Estadísticas globales: Después de mostrar los detalles de cada prueba, se presenta un resumen
#     general de los resultados, mostrando la misma información de arriba pero ahora para la suma de todas
#     las pruebas.
nose.runmodule()
except Exception as error:

#type(clase).__name__: Esta instrucción no es un método, sino una expresión que se utiliza para obtener
#el nombre de la clase de un objeto en Python, donde type(error) devuelve el tipo de excepción en este
#caso ya que error es un objeto de una clase de excepción.

# - __name__: Es un atributo especial en Python que se utiliza para obtener el nombre de la clase del
#     objeto.

```

```
print("Ocurrió el siguiente tipo de error: ", type(error).__name__)
print("Error en node:", error)
```

Resultado del Código Python: Pruebas *Assertion* y *Nose* aprobadas

```
PROBLEMAS  SALIDA  TERMINAL  CONSOLA DE DEPURACIÓN

PS C:\Users\diego\OneDrive\Documents\Aprendiendo\Python\1.-Instrumentación Virtual> & D:/Users/diego/AppData/Local/Programs/Python/Python39/python.exe
"c:/Users/diego/OneDrive/Documents/Aprendiendo/Python/1.-Instrumentación Virtual/Instrumentación Virtual Aplicada/Tareas/Tarea 1/Tareas C#/centered_diff2.py"

+-----+-----+-----+-----+-----+
| f(x)   | Valor de x | Derivada evaluada en x | Diferencial evaluada en x | Error |
+-----+-----+-----+-----+-----+
| e^x    | 0          | 1                     | 1.00002                   | 1.66667e-05 |
+-----+-----+-----+-----+-----+
| e^(-2x^2) | 0        | 0                     | 0                           | 0        |
+-----+-----+-----+-----+-----+
| cos(x)  | 6.28319   | 0                     | 0                           | 0        |
+-----+-----+-----+-----+-----+
| ln(x)   | 1          | 1                     | 1.00003                   | 3.33353e-05 |
+-----+-----+-----+-----+-----+

-----Assertion Testing-----
Para pasar las pruebas de Assertion Testing el error debe ser menor o igual a: 0.0001 = 1.0e-04
Todas las pruebas pasaron correctamente.

-----Nose Testing-----
Para pasar las pruebas de Nose Testing el error debe ser menor o igual a: 0.0001 = 1.0e-04
.
-----
Ran 1 test in 0.000s

OK
PS C:\Users\diego\OneDrive\Documents\Aprendiendo\Python\1.-Instrumentación Virtual>
```

Resultado del Código Python: Prueba *Assertion* aprobada y *Nose* reprobada

```
PROBLEMAS  SALIDA  TERMINAL  CONSOLA DE DEPURACIÓN

PS C:\Users\diego\OneDrive\Documents\Aprendiendo\Python\1.-Instrumentación Virtual> & D:/Users/diego/AppData/Local/Programs/Python/Python39/python.exe
"c:/Users/diego/OneDrive/Documents/Aprendiendo/Python/1.-Instrumentación Virtual/Instrumentación Virtual Aplicada/Tareas/Tarea 1/Tareas C#/centered_diff2.py"

+-----+-----+-----+-----+-----+
| f(x)   | Valor de x | Derivada evaluada en x | Diferencial evaluada en x | Error |
+-----+-----+-----+-----+-----+
| e^x    | 0          | 1                     | 1.00002                   | 1.66667e-05 |
+-----+-----+-----+-----+-----+
| e^(-2x^2) | 0        | 0                     | 0                           | 0        |
+-----+-----+-----+-----+-----+
| cos(x)  | 6.28319   | 0                     | 0                           | 0        |
+-----+-----+-----+-----+-----+
| ln(x)   | 1          | 1                     | 1.00003                   | 3.33353e-05 |
+-----+-----+-----+-----+-----+

-----Assertion Testing-----
Para pasar las pruebas de Assertion Testing el error debe ser menor o igual a: 0.0001 = 1.0e-04
Todas las pruebas pasaron correctamente.

-----Nose Testing-----
Para pasar las pruebas de Nose Testing el error debe ser menor o igual a: 1e-05 = 1.0e-05
F
=====
FAIL: __main__.test_nose_application
-----
Traceback (most recent call last):
  File "D:/Users/diego/AppData/Local/Programs/Python/Python39/lib/site-packages/nose/case.py", line 198, in runTest
    self.test(*self.args)
  File "c:/Users/diego/OneDrive/Documents/Aprendiendo/Python/1.-Instrumentación Virtual/Instrumentación Virtual Aplicada/Tareas/Tarea 1/Tareas C#/centered_diff2.py", line 240, in test_nose_application
    nose.tools.assert_almost_equal(resultado_diferencial[0], resultado_derivada[0], delta = umbral_nose_testing)
AssertionError: 1.0000166667499921 != 1 within 1e-05 delta (1.6666749992122476e-05 difference)

-----
Ran 1 test in 0.001s

FAILED (failures=1)
PS C:\Users\diego\OneDrive\Documents\Aprendiendo\Python\1.-Instrumentación Virtual>
```

Código C# (.NET Framework) – Visual Studio (Logo Morado)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
//Se usa la librería MathNet.Numerics, para ello se debe seleccionar la siguiente opción:
//Explorador de soluciones -> Clic Derecho en Referencias -> Administrar Paquetes NuGet -> Examinar -> MathNet.Numerics ->
//Instalar.
//Explorador de soluciones -> Clic Derecho en Referencias -> Administrar Paquetes NuGet -> Examinar -> NUnit -> Instalar.
//Explorador de soluciones -> Referencias -> Agregar Referencia -> Nombre Librería.
using MathNet.Numerics;
using NUnit.Framework;

namespace _9._7._Derivada_Diferencial_y_Testing
{
    class DerivadaConDefinicionDiferencial
    {
        static void Main(string[] args)
        {
            double umbralAssertion = 1e-4;
            double umbralNoseTesting = 1e-4;

            diff((x) => Math.Exp(x), 0, 0.01);
            diff((x) => Math.Exp(-2.0 * Math.Pow(x, 2)), 0, 0.01);
            diff((x) => Math.Cos(x), 2 * Math.PI, 0.01);
            diff((x) => Math.Log(x), 1, 0.01);

            assertionTesting(umbralAssertion);
            noseApplication(umbralNoseTesting);
        }

        static double diff(Func<double, double> f, double x, double h = 1e-5)
        {
            double df = (f(x + h) - f(x - h)) / (2.0 * h);
            return df;
        }

        static void assertionTesting(double umbralAssertionTesting)
        {
            double[] resultadoDerivada = { 1, 0, 0, 1 };
            double[] x = { 0, 0, 2 * Math.PI, 1 };
            double[] resultadoDiferencial = new double[4];
            double dx = 0.01;
            double[] error = new double[4];
            string[] nombreFunciones = { "e^x", "e^(-2x^2)", "cos(x)", "ln(x)" };
            string[][] tabla = new string[nombreFunciones.Length][];

            resultadoDiferencial[0] = diff((xx) => Math.Exp(xx), x[0], dx);
            resultadoDiferencial[1] = diff((xx) => Math.Exp(-2.0 * Math.Pow(xx, 2)), x[1], dx);
            resultadoDiferencial[2] = diff((xx) => Math.Cos(xx), x[2], dx);
            resultadoDiferencial[3] = diff((xx) => Math.Log(xx), x[3], dx);

            for (int i = 0; i < nombreFunciones.Length; i++)
            {
                error[i] = Math.Abs(resultadoDiferencial[i] - resultadoDerivada[i]);
                tabla[i] = new string[] { nombreFunciones[i], x[i].ToString(), resultadoDerivada[i].ToString(),
                resultadoDiferencial[i].ToString(), error[i].ToString() };
            }

            Console.WriteLine("f(x)\t\tValor de x\tDerivada evaluada en x\tDiferencial evaluada en x\tError");
            foreach (string[] row in tabla)
            {
                Console.WriteLine(string.Join("\t\t", row));
            }

            string notacionCientifica = umbralAssertionTesting.ToString("0.0e+0");
            Console.WriteLine("\n\n-----Assertion
            Testing-----");
            Console.WriteLine("Para pasar las pruebas de Assertion Testing el error debe ser menor o igual a: " +
            umbralAssertionTesting + " = " + notacionCientifica);
            Assert.LessOrEqual(error[0], umbralAssertionTesting, "Error muy grande al aproximar la derivada de e^x por medio
            de su diferencial");
            Assert.LessOrEqual(error[1], umbralAssertionTesting, "Error muy grande al aproximar la derivada de e^(-2x^2) por
            medio de su diferencial");
            Assert.LessOrEqual(error[2], umbralAssertionTesting, "Error muy grande al aproximar la derivada de cos(x) por medio
            de su diferencial");
            Assert.LessOrEqual(error[3], umbralAssertionTesting, "Error muy grande al aproximar la derivada de ln(x) por medio
```

```

        de su diferencial");
        Console.WriteLine("Todas las pruebas pasaron correctamente.");
    }

    static void noseApplication(double umbralNoseTesting)
    {
        double[] resultadoDerivada = { 1, 0, 0, 1 };
        double[] resultadoDiferencial = { diff((x) => Math.Exp(x), 0, 0.01), diff((x) => Math.Exp(-2.0 * Math.Pow(x, 2)),
        0, 0.01), diff((x) => Math.Cos(x), 2 * Math.PI, 0.01), diff((x) => Math.Log(x), 1, 0.01) };

        Console.WriteLine("\n\n-----Nose
        Testing-----");
        Console.WriteLine("Para pasar las pruebas de Nose Testing el error debe ser menor o igual a: " +
        umbralNoseTesting.ToString("0.0e+0"));
        Assert.AreEqual(resultadoDiferencial[0], resultadoDerivada[0], umbralNoseTesting, "Error muy grande al aproximar
        la derivada");
        Assert.AreEqual(resultadoDiferencial[1], resultadoDerivada[1], umbralNoseTesting, "Error muy grande al aproximar
        la derivada");
        Assert.AreEqual(resultadoDiferencial[2], resultadoDerivada[2], umbralNoseTesting, "Error muy grande al aproximar
        la derivada");
        Assert.AreEqual(resultadoDiferencial[3], resultadoDerivada[3], umbralNoseTesting, "Error muy grande al aproximar
        la derivada");
        Console.WriteLine("Todas las pruebas pasaron correctamente.");
    }
}
}

```

Resultado del Código C#

```

C:\WINDOWS\system32\cmd. x + v
f(x)      Valor de x      Derivada evaluada en x      Diferencial evaluada en x      Error
e^x        0              1              1.00001666674999              1.66667499921225E-05
e^(-2x^2)  0              0              0              0
cos(x)     6.28318530717959  0              0              0
ln(x)      1              1              1.00003333333348              3.3335333477158E-05

-----Assertion Testing-----
Para pasar las pruebas de Assertion Testing el error debe ser menor o igual a: 0.0001 = 1.0e-4
Todas las pruebas pasaron correctamente.

-----Nose Testing-----
Para pasar las pruebas de Nose Testing el error debe ser menor o igual a: 1.0e-4
Todas las pruebas pasaron correctamente.
Presione una tecla para continuar . . . |

```

