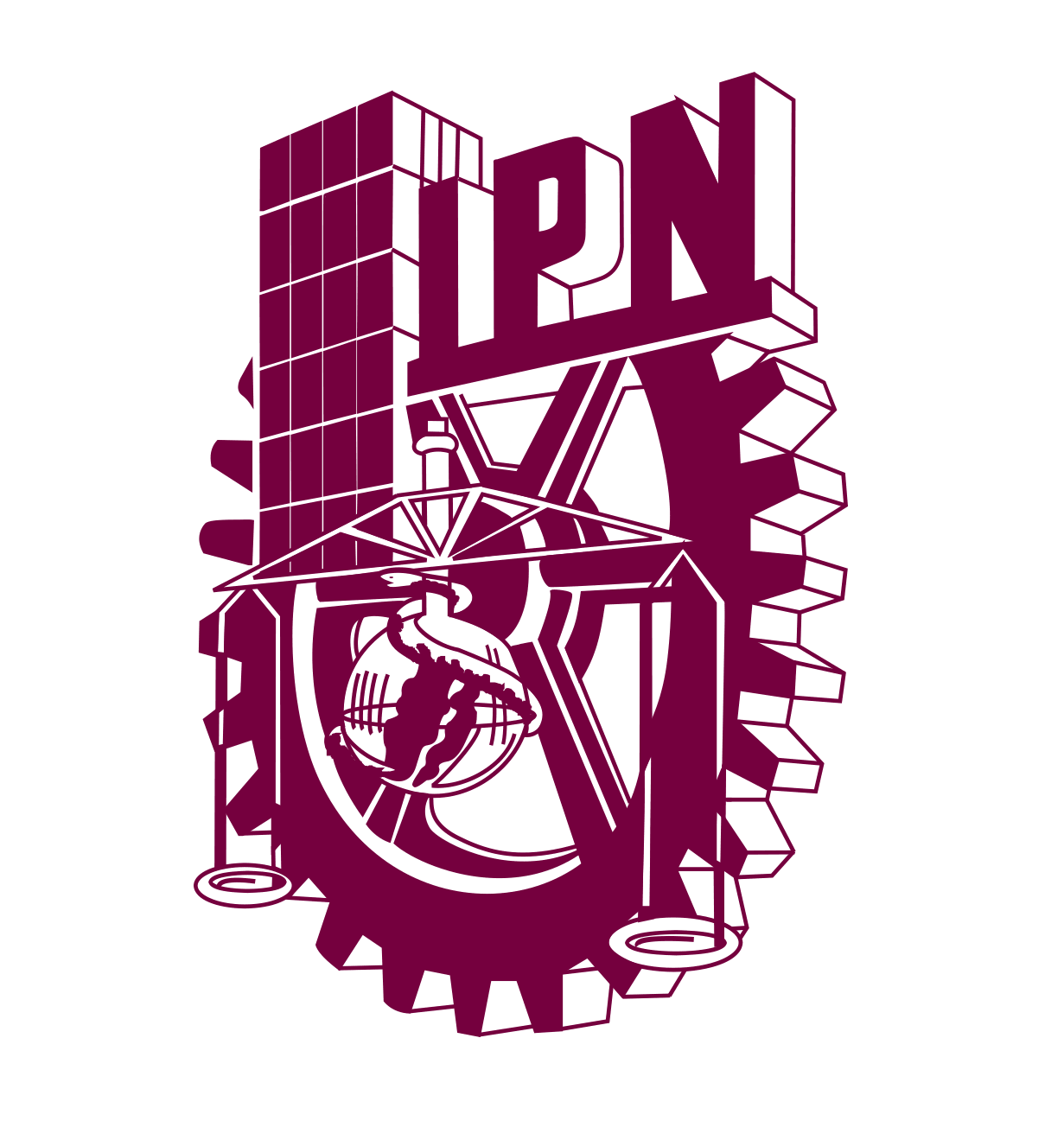
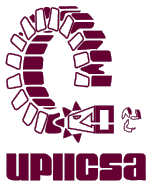
**INSTITUTO POLITÉCNICO NACIONAL**

**Unidad Profesional Interdisciplinaria de Ingeniería y Ciencias Sociales y Administrativas**

**Ingeniería de Pruebas**

**Documento Técnico: Requerimientos funcionales y no funcionales - Calculadora en Python con Tkinter**

**Autores:**

* Contreras Mosco Cristobal
* Díaz Pérez Diego
* Hernández Aguirre Ricardo
* Monroy Muñoz Angel Yael
* Salazar Rocha Any Jennifer

**Fecha:** [09/09/2025]

**Versión:** 1.0

**Índice**

1. Introducción
2. Objetivo
3. Alcance
4. Requisitos Funcionales
5. Requerimientos no funcionales
6. Casos de uso

6.1. División entre cero

6.2. Mostrar/Ocultar historial

6.3. Borrar último carácter

6.4. Calcular operación completa

6.5. Reiniciar calculadora

1. Conclusión

**Introducción**

La presente documentación describe los requerimientos del sistema Calculadora en Python con Tkinter. El objetivo principal es especificar de manera clara las funcionalidades que debe cumplir el software, así como las características no funcionales que aseguren su correcta operación. El sistema está diseñado para permitir al usuario realizar operaciones aritméticas básicas (suma, resta, multiplicación y división) de manera sencilla y rápida, a través de una interfaz gráfica amigable que simula una calculadora estándar.

Además, el programa incluye funciones adicionales como el manejo de historial de operaciones, validación de entradas, detección de errores comunes (por ejemplo, división entre cero) y controles de borrado.

**Objetivo**

El objetivo del sistema es proporcionar una calculadora digital con interfaz gráfica en Python (Tkinter) que combine facilidad de uso, confiabilidad y eficiencia en la ejecución de operaciones aritméticas básicas.

En detalle, el sistema busca:

1. Facilidad de uso: presentar una interfaz amigable, con botones claramente identificados y un flujo de interacción similar al de una calculadora física estándar.
2. Precisión y claridad: mostrar los resultados de manera inmediata, con redondeo automático a dos decimales cuando sea necesario, y diferenciando entre resultados enteros y decimales.
3. Seguridad en las operaciones: evitar bloqueos o errores críticos mediante validaciones que impidan entradas inválidas, operadores duplicados o intentos de división entre cero.
4. Historial accesible: ofrecer un registro de las operaciones realizadas, permitiendo al usuario revisarlas, eliminarlas o reiniciar el sistema sin perder control sobre sus cálculos.
5. Soporte a continuidad de cálculos: permitir que los resultados obtenidos puedan usarse de forma directa en nuevas operaciones, evitando que el usuario tenga que volver a ingresarlos.
6. Extensibilidad: contar con una base de código organizada y modular que facilite la incorporación de nuevas funcionalidades en versiones futuras (por ejemplo, funciones científicas o conversores).

**Alcance**

El sistema abarca las siguientes capacidades dentro de su primera versión:

1. Operaciones matemáticas: suma, resta, multiplicación y división.
2. Interfaz gráfica intuitiva: desarrollada en Tkinter, con botones para números, operadores y funciones adicionales, junto a una pantalla para mostrar la operación actual y el resultado.
3. Historial de operaciones: registro dinámico de cálculos que el usuario puede mostrar u ocultar en cualquier momento, y que se actualiza automáticamente después de cada operación.
4. Funciones de borrado:
   * C: limpia la operación actual.
   * CA: reinicia por completo la calculadora y el historial.
   * B: borra únicamente el último carácter ingresado.
5. Validaciones de entrada:
   * Restricción de máximo un punto decimal por número.
   * Reemplazo automático de operadores consecutivos.
   * Mensajes de error claros en caso de entradas incompletas.
   * Límite de 12 caracteres en la pantalla para evitar problemas visuales.
6. Manejo de errores críticos:
   * Detección de división entre cero con mensaje claro (ERROR: ÷0).
   * Prevención de operaciones incompletas (ERROR: Incompleta).
7. Compatibilidad multiplataforma: el sistema puede ejecutarse en cualquier equipo que tenga instalado Python 3 y Tkinter (Windows, Linux o macOS).

**Fuera del alcance de esta versión:**

1. Operaciones científicas (potencias, raíces, logaritmos, trigonometría).
2. Funciones de memoria avanzada (guardar y recuperar resultados múltiples).
3. Persistencia del historial tras cerrar la aplicación.
4. Ejecución en navegadores web o dispositivos móviles (limitado a escritorio con Python).

**Requerimientos Funcionales**

1. Operaciones básicas: suma (+), resta (-), multiplicación (\*) y división (/).
2. Entrada de números hasta dos decimales.
3. Diseño estándar en consola, con menú y resultado visible.
4. Historial de operaciones.
5. Botones de borrado:
   * C para borrar la última entrada.
   * CA para reiniciar la calculadora.
   * B para borrar el último carácter ingresado.
   * H para ver historial.
6. Continuación de operaciones usando = (usa el último resultado como base para seguir calculando).
7. División entre cero: el programa detecta el intento, muestra un mensaje de error (ERROR: ÷0) y no se ejecuta la operación.
8. Validaciones adicionales:
   * Evita ingresar dos puntos decimales en el mismo número.
   * Evita operadores consecutivos (los reemplaza por el último ingresado).
   * Muestra error en operaciones incompletas (ejemplo: 5+).
   * No permite más de un punto decimal en un mismo número
9. Límite de 12 caracteres en pantalla para evitar desbordes visuales.
10. Los resultados solo muestran punto decimal cuando es necesario.

**Requerimientos no funcionales posibles**

1. **Rendimiento y eficiencia**
   * El sistema debe responder a cada comando en menos de 1 segundo.
   * El historial debe poder almacenar al menos *N* operaciones sin afectar el tiempo de respuesta.
2. **Usabilidad**

La interfaz debe ser clara, mostrando siempre las opciones disponibles (menú, comandos, historial).

Los mensajes de error deben ser comprensibles (ERROR: ÷0, ERROR: Incompleta)

1. **Confiabilidad**

El sistema debe manejar adecuadamente errores de entrada (como división entre 0 o caracteres no válidos).

El historial no debe perderse a menos que se use el comando CA o se cierre la aplicación.

1. **Mantenibilidad**

El código está modularizado en métodos (agregar\_a\_operacion, realizar\_calculo, borrar\_ultimo\_caracter, etc.), lo que facilita modificaciones.

La arquitectura debe permitir agregar nuevos comandos sin reestructurar todo el sistema.

1. **Portabilidad**

El sistema debe poder ejecutarse en diferentes plataformas (ejemplo: consola, aplicación web, aplicación de escritorio).

**Casos de Uso**

**Caso de uso 1: División entre cero**

* Actor principal: Usuario.
* Objetivo: Evitar operaciones inválidas cuando se intenta dividir un número entre cero.
* Precondición: El usuario tiene la calculadora abierta e ingresa una operación con un divisor igual a 0.
* Flujo principal:
  1. El usuario ingresa una expresión (ejemplo: 8/0).
  2. El usuario presiona = para ejecutar la operación.
  3. El sistema detecta que el divisor es cero.
  4. El sistema muestra en pantalla el mensaje “ERROR: ÷0”.
  5. El sistema descarta la operación y limpia el campo de entrada.
* Postcondición: La calculadora sigue activa y lista para nuevas operaciones.
* Excepción: Si el usuario vuelve a intentar otra operación válida, se realiza normalmente.

**Caso de uso 2: Mostrar/Ocultar historial**

Actor principal: Usuario.

Objetivo: Permitir al usuario consultar las operaciones realizadas durante la sesión.

Precondición: La calculadora está abierta y en funcionamiento.

Flujo principal:

* 1. El usuario presiona el botón H.
  2. El sistema muestra el historial de operaciones en un panel lateral.
  3. El usuario puede revisar las operaciones realizadas.
  4. Si el usuario vuelve a presionar H, el sistema oculta el historial.

Postcondición: El historial permanece almacenado, aunque esté oculto, hasta que el usuario decida limpiarlo con CA.

**Caso de uso 3: Borrar último carácter**

Actor principal: Usuario.

Objetivo: Corregir errores al ingresar una operación.

Precondición: El usuario tiene una operación en curso.

Flujo principal:

* 1. El usuario presiona el botón B.
  2. El sistema elimina el último carácter ingresado en la operación actual.
  3. El sistema actualiza la pantalla para reflejar el cambio.

Postcondición: La operación queda corregida y el usuario puede continuar ingresando datos.

**Caso de uso 4: Calcular operación completa**

Actor principal: Usuario.

Objetivo: Obtener el resultado de una operación aritmética válida.

Precondición: El usuario ha ingresado una operación completa (ejemplo: 5+3\*2).

Flujo principal:

* 1. El usuario presiona el botón =.
  2. El sistema evalúa la operación siguiendo las reglas aritméticas.
  3. El sistema muestra el resultado en la pantalla.
  4. El sistema guarda la operación y su resultado en el historial.

Postcondición: El resultado puede usarse directamente en una nueva operación.

**Caso de uso 5: Reiniciar calculadora**

Actor principal: Usuario.

Objetivo: Limpiar por completo la pantalla y el historial.

Precondición: La calculadora tiene operaciones registradas o en curso.

Flujo principal:

* 1. El usuario presiona el botón CA.
  2. El sistema borra la pantalla y el historial.
  3. La calculadora vuelve a estado inicial.

Postcondición: No queda ninguna operación previa almacenada.

**Conclusiones**

El sistema de calculadora desarrollado en Python con Tkinter cumple satisfactoriamente con los requerimientos establecidos al inicio del proyecto, ofreciendo una herramienta sencilla y confiable para la realización de operaciones aritméticas básicas. La aplicación permite al usuario efectuar sumas, restas, multiplicaciones y divisiones con resultados claros y precisos, además de contar con validaciones que aseguran un manejo correcto de errores, como la detección de divisiones entre cero o entradas incompletas.