

TRABAJO INTEGRADOR: MATEMÁTICAS Y PROGRAMACIÓN

1. CARÁTULA

Título del Proyecto: Simulador de Compuertas Lógicas y Álgebra de Boole

Materia: Matemáticas y Programación

Alumno: Juan José Diaz

Fecha: 24/11/2025

2. DESCRIPCIÓN DEL PROBLEMA

Fundamentación Teórica

El presente proyecto tiene como objetivo vincular los conceptos abstractos del **Álgebra de Boole** con su aplicación práctica en la programación. El Álgebra de Boole es una estructura matemática que esquematiza las operaciones lógicas, trabajando con un conjunto binario: **0 (Falso/OFF)** y **1 (Verdadero/ON)**.

El programa desarrollado ("Simulador de Compuertas Lógicas") permite al usuario visualizar y calcular el resultado de las operaciones fundamentales que rigen la lógica digital de las computadoras:

- AND (Conjunción - Producto Lógico):** La salida es 1 solo si ambas entradas son 1 ($A \cdot B$).
- OR (Disyunción - Suma Lógica):** La salida es 1 si al menos una de las entradas es 1 ($A + B$).
- NOT (Negación - Inversor):** Invierte el valor de entrada; si entra 0, sale 1 ($\neg A$).
- XOR (O Exclusivo):** La salida es 1 solo si las entradas son diferentes ($A \oplus B$).

El software incluye validaciones para asegurar que el usuario solo ingrese datos dentro del dominio binario y utiliza representaciones gráficas (Arte ASCII) para facilitar la comprensión visual del circuito.

3. ENLACE AL VIDEO EXPLICATIVO

<https://youtu.be/TjUU3gm5jlg>

4. ANÁLISIS DE INTELIGENCIA ARTIFICIAL

Metodología y Justificación

Para el desarrollo de este proyecto se implementó una estrategia iterativa utilizando tres modelos de Inteligencia Artificial distintos, asignando roles específicos para optimizar cada fase del desarrollo:

Justificación de uso:

"Gemini se utilizó para el diseño lógico y el análisis matemático del problema, ChatGPT se empleó para generar la estructura base del código en Python, y finalmente DeepSeek se utilizó para la optimización visual y refinamiento del código (UX). Esta división de tareas permitió un flujo de trabajo más eficiente y robusto que usar una sola herramienta, aprovechando la capacidad de razonamiento de una y la capacidad de codificación de las otras."

Evidencia del Proceso

Evidencia 1: Análisis Matemático (Gemini)

FASE 2: Diseño del Algoritmo

Para que el programa sea ordenado y cumpla con la rúbrica de "código claro", le pediremos a la IA generadora que siga esta estructura:

1. **Menú Principal:** Mostrar las opciones (1. AND, 2. OR, 3. NOT, 4. XOR).
2. **Validación de Entrada:** Es crucial. Si el usuario ingresa un "2" o una letra "A", el programa debe dar error, porque el sistema binario solo acepta 0 y 1.
3. **Procesamiento:** Usar los operadores lógicos de Python (`&`, `|`, `not`, `^`) o los operadores de comparación (`and`, `or`).
4. **Visualización:** Mostrar claramente: `Entrada A: 1 | Entrada B: 0 --> Salida: 0`.

FASE 3: Acción con la IA n.º 2 (Generación de Código)

Ahora, ve a **ChatGPT** (o tu segunda IA elegida) y usa este prompt específico para la Opción B. Recuerda tomar captura de esa conversación.

Copia y pega este Prompt:

"Hola, necesito ayuda para un trabajo escolar de programación y matemáticas. Actúa como un experto en Python.

Necesito crear un script que simule una **Calculadora de Compuertas Lógicas**.

Requerimientos:

1. El programa debe mostrar un menú para elegir la operación: AND, OR, NOT, y XOR.
2. Debe pedir al usuario que ingrese los valores binarios (0 o 1). Nota: Para NOT solo se pide un valor, para las demás se piden dos.
3. **Validación:** Debe verificar que el usuario realmente ingrese 0 o 1. Si ingresa otra cosa, debe mostrar error y volver a preguntar.
4. Debe calcular el resultado usando la lógica booleana y mostrarlo en pantalla de forma clara (ej: 1 AND 0 = 0).
5. Usa un bucle `while` para que el programa siga corriendo hasta que el usuario decida salir.
6. Agrega comentarios explicando qué hace cada función para poder estudiarlo."

Vamos a escribir o crear juntos

+ X

Razonamiento ▾

Gemini puede cometer errores, así que verifica sus respuestas.

Evidencia 2: Generación de Código Base (ChatGPT)

✓ CALCULADORA DE COMPUERTAS LÓGICAS (Python, con comentarios)

```
python
```

Copiar código

```
# -----
# Calculadora de Compuertas Lógicas
# Permite realizar operaciones AND, OR, NOT y XOR sobre valores
# binarios (0 y 1). Incluye validaciones y menú interactivo.
# -----
```

```
def pedir_valor(mensaje):
    """
    Pide al usuario un valor binario (0 o 1).
    Si el usuario ingresa algo inválido, se vuelve a pedir.
    """
    while True:
        valor = input(mensaje)

        # Validamos si el valor es exactamente "0" o "1"
        if valor in ("0", "1"):
            return int(valor)  # lo convertimos a entero 0 o 1
        else:
            print("X Error: Debes ingresar 0 o 1.")


def calcular_and(a, b):
    """Devuelve a AND b usando operadores booleanos."""
    return a and b


def calcular_or(a, b):
    """Devuelve a OR b usando operadores booleanos."""
    return a or b


def calcular_not(a):
    """Devuelve NOT a usando operador booleano."""
    return not a


def calcular_xor(a, b):
    """Devuelve a XOR b. True cuando sólo uno es 1."""
    return (a and not b) or (not a and b)
```

↓

```
def mostrar_menu():
+ |Pregunta lo que quieras
```

0 ⏪

ChatGPT puede cometer errores. Considera verificar la información importante.

Evidencia 3: Optimización y Arte ASCII (DeepSeek)

Script Python para generar tabla de verdad

python

Copiar Descargar

```
def mostrar_arte_ascii(operacion, a, b, resultado):
    """
    MEJORA VISUAL: Muestra representaciones ASCII de las compuertas lógicas
    """
    print("\n" + "REPRESANTACIÓN GRÁFICA:".center(50, "-"))

    if operacion == "AND":
        print(""""
        [A]
        | AND |
        [B]
        """)
        print(f"    A={a}, B={b} → A AND B = {int(resultado)}")

    elif operacion == "OR":
        print(""""
        [A]
        | OR |
        [B]
        """)
        print(f"    A={a}, B={b} → A OR B = {int(resultado)}")

    elif operacion == "NOT":
        print(""""
        [A]
        | NOT |
        """)
        print(f"    A={a} → NOT A = {int(resultado)}")

    elif operacion == "XOR":
        print(""""
        [A]
        | XOR |
        [B]
        """)
        print(f"    A={a}, B={b} → A XOR B = {int(resultado)}")

    else:
        print("Operación no soportada")
```

Mensaje a DeepSeek

Pensamiento Profundo Buscar

Generated por IA, solo para referencia.

5. CÓDIGO FUENTE (PYTHON)

Github: <https://github.com/diazjuanjo/UTN-TUPaD-MAT-TPI>

A continuación se adjunta el código completo desarrollado para el simulador:

```
# -----
# TRABAJO INTEGRADOR: MATEMÁTICAS Y PROGRAMACIÓN
# TEMA: Simulador de Compuertas Lógicas
# ALUMNO: Díaz Juan José
#
# DESCRIPCIÓN:
# Este programa permite simular el comportamiento de las compuertas
# lógicas fundamentales (AND, OR, NOT, XOR) utilizando Álgebra de Boole.
# Incluye representaciones gráficas en ASCII y validación de entradas.
# -----

import os

def limpiar_pantalla():
    # Para Windows
    if os.name == 'nt':
        _ = os.system('cls')
    # Para macOS y Linux (el nombre es 'posix')
    else:
        _ = os.system('clear')

def pedir_valor(mensaje):
    """
    Pide al usuario un valor binario (0 o 1).
    Si el usuario ingresa algo inválido, se vuelve a pedir hasta que sea correcto.
    """
    while True:
        valor = input(mensaje)
        # Validamos si el valor es exactamente "0" o "1"
        if valor in ("0", "1"):
            return int(valor)
        else:
            print("X Error: Debes ingresar 0 o 1 (Sistema Binario).")

def calcular_and(a, b):
    """Simula compuerta AND: Salida 1 solo si ambas entradas son 1."""
    return a and b
```

```

def calcular_or(a, b):
    """Simula compuerta OR: Salida 1 si al menos una entrada es 1."""
    return a or b

def calcular_not(a):
    """Simula compuerta NOT: Invierte el valor de entrada."""
    return not a

def calcular_xor(a, b):
    """Simula compuerta XOR: Salida 1 si las entradas son diferentes."""
    # Equivalente lógico: (A and not B) or (not A and B)
    return (a and not b) or (not a and b)

def mostrar_menu():
    """Muestra el menú interactivo en consola."""
    print("\n" + "=" * 50)
    print("⊗ SIMULADOR DE ÁLGEBRA DE BOOLE ⊗")
    print("=" * 50)
    print("1. AND (Conjunción - A ∧ B)")
    print("2. OR (Disyunción - A ∨ B)")
    print("3. NOT (Negación - ¬A)")
    print("4. XOR (O Exclusivo - A ⊕ B)")
    print("5. Salir")
    print("—" * 50)
    return input("Elige una opción (1-5): ")

def obtener_datos_operacion(opcion):
    """
    Maneja la lógica de pedir datos según la compuerta elegida.
    Optimiza el código evitando repetición de inputs.
    """
    operaciones = {
        '1': {'nombre': 'AND', 'entradas': 2, 'funcion': calcular_and},
        '2': {'nombre': 'OR', 'entradas': 2, 'funcion': calcular_or},
        '3': {'nombre': 'NOT', 'entradas': 1, 'funcion': calcular_not},
        '4': {'nombre': 'XOR', 'entradas': 2, 'funcion': calcular_xor}
    }
    if opcion not in operaciones:
        return None, None, None, None
    info = operaciones[opcion]
    # Lógica para pedir 1 o 2 valores según corresponda
    if info['entradas'] == 1:
        a = pedir_valor("Ingresa el valor (0 o 1): ")
        b = None
    else:
        a = pedir_valor("Ingresa el primer valor (0 o 1): ")
        b = pedir_valor("Ingresa el segundo valor (0 o 1): ")
    resultado = info['funcion'](a, b)
    return info['nombre'], resultado, a, b

def mostrar_arte_ascii(operacion, a, b, resultado):
    """
    Dibuja la compuerta lógica en la consola para visualización educativa.
    """
    print("\n" + "⊗ ESQUEMA LÓGICO:" .center(50, "-"))

```

```

res_int = int(resultado) # Convertir True/False a 1/0 para visualización
if operacion == "AND":
    print(f"""
    [ {a} ---| |
      | AND |--- {res_int}
    {b} ---| |
    """)

elif operacion == "OR":
    print(f"""
    [ {a} ---| |
      | OR |--- {res_int}
    {b} ---| |
    """)

elif operacion == "NOT":
    print(f"""
    [ | |
    {a} ---| NOT |--- {res_int}
    | |
    """)

elif operacion == "XOR":
    print(f"""
    [ {a} ---| |
      | XOR |--- {res_int}
    {b} ---| |
    """)

def mostrar_resultado_detallado(operacion, a, b, resultado):
    """Imprime el resultado final combinando matemáticas y programación."""
    limpiar_pantalla()
    print("\n" + " RESULTADO MATEMÁTICO:".center(50, "="))
    res_int = int(resultado)
    if operacion == "NOT":
        print(f" Expresión: ¬{a} = {res_int}")
    else:
        print(f" Expresión: {a} {operacion} {b} = {res_int}")
    mostrar_arte_ascii(operacion, a, b, resultado)
    print("-" * 50)

# -----
# FUNCIÓN PRINCIPAL (Entry Point)
# -----


def main():
    limpiar_pantalla()
    print(" INICIANDO SISTEMA DE LÓGICA DIGITAL...")
    while True:
        opcion = mostrar_menu()

    if opcion == "5":

```

```
print("\n👋 Fin de la simulación.")  
break  
nombre_op, resultado, a, b = obtener_datos_operacion(opcion)  
if nombre_op is None:  
    limpiar_pantalla()  
    print("❌ Opción inválida. Intenta nuevamente.")  
    continue  
mostrar_resultado_detallado(nombre_op, a, b, resultado)  
input("\n➡ Presiona Enter para continuar...")  
limpiar_pantalla()  
  
if __name__ == "__main__":  
    main()
```