# Trabajo práctico integrador Matemática - Programación

Alumnos - Grupo nº 1 - Comisión 10

Bayurk Mara Valentina
Berrone Lanza Lina Lucia
Bonetti Kunt Daniela Sofia
Cabrera Dario Ezequiel
Benitez Carolina Anabel
Bustamante Erica

### Introducción

Para comenzar con la presentación nos gustaría aclarar que vamos a presentar tres formas de resolver el mismo ejercicio realizadas y elegidas por el equipo; en las dos primeras vamos a mostrar el código utilizando todos los conceptos aprendidos hasta el momento en programación y las probaremos en consola.

En la tercera opción se cambió el enfoque y lo que hicimos fue utilizar la IA para refactorizar y ella decida cómo corregir ese código. Decidimos abordar el trabajo de esta manera ya que al comparar las opciones que se nos ocurrieron, notamos que había tantas formas de resolver el ejercicio como personas en el grupo. En todos los casos incluimos las 6 compuertas sugeridas.

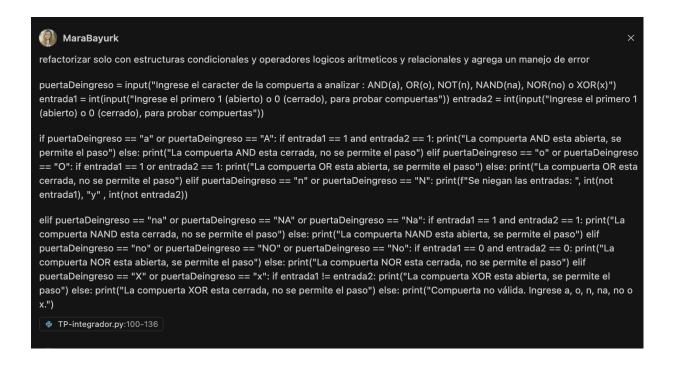
## <u>Repositorio</u>

https://github.com/MaraBayurk/TP-GRUPAL-TUPAD-G1

# Utilización de IA y Evidencias

A continuación se mostrará las diferentes capturas de las cuales han sido de apoyo a la hora de desarrollar el trabajo integrador. La misma fue utilizada para refactorizar los programas sugeridos y poder evacuar dudas sobre diferentes conceptos.

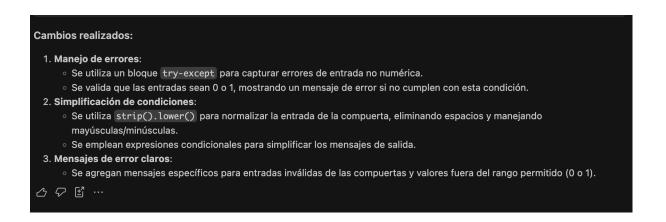
En esta captura podemos visualizar que se solicitó a la IA refactorizar el código solo con condicionales y operadores lógicos. se debió reformular la pregunta varias veces hasta llegar al resultado deseado.



En esta captura se observa cómo la IA nos ayudó a refactorizar el código, siguiendo indicaciones precisas para que utilice únicamente los contenidos vistos hasta el momento en clase

```
B GitHub Copilot
  Used 1 reference
Aquí tienes el código refactorizado utilizando únicamente estructuras condicionales, operadores lógicos, aritméticos y relacionales.
además de incluir manejo de errores para entradas inválidas:
     puertaDeingreso = input("Ingrese el caracter de la compuerta a analizar: AND(a), OR(o), NOT(n), NAND(na),
     entrada1 = int(input("Ingrese el primer valor (1 para abierto o 0 para cerrado): "))
entrada2 = int(input("Ingrese el segundo valor (1 para abierto o 0 para cerrado): "))
     if entrada1 not in [0, 1] or entrada2 not in [0, 1]:
          print("Error: Las entradas deben ser 0 o 1.")
     elif puertaDeingreso == "a":
          print("La compuerta AND está", "abierta, se permite el paso" if entrada1 and entrada2 else "cerrada, n
     elif puertaDeingreso == "o":
          print("La compuerta OR está", "abierta, se permite el paso" if entrada1 or entrada2 else "cerrada, no
     elif puertaDeingreso == "n":
          print(f"Se niegan las entradas: {int(not entrada1)} y {int(not entrada2)}")
     elif puertaDeingreso == "na":
          print("La compuerta NAND está", "abierta, se permite el paso" if not (entrada1 and entrada2) else "cer
     elif puertaDeingreso == "no":
         print("La compuerta NOR está", "abierta, se permite el paso" if not (entrada1 or entrada2) else "cerra
     elif puertaDeingreso == "x":
         print("La compuerta XOR está", "abierta, se permite el paso" if entrada1 != entrada2 else "cerrada, no
 except ValueError:
     print("Error: Entrada inválida. Asegúrese de ingresar números enteros para las entradas.")
```

En esta captura podemos ver un breve resumen de los cambios realizados por la IA. La misma detalla tanto el porqué de los cambios como el resultado generado por los mismos.



Aquí se solicita modificar la respuesta obtenida, pero sin el uso del bloque Try, de esta manera pedimos la optimización del código pero refactorizado con restricciones.

```
MaraBayurk
sin trv
TP-integrador.py:100-136
B GitHub Copilot
> Used 1 reference
Aquí tienes el código refactorizado sin usar try, pero con validaciones para manejar errores de entrada:
 puertaDeingreso = input("Ingrese el caracter de la compuerta a analizar: AND(a), OR(o), NOT(n), NAND(na), NOR(
 entrada1 = input("Ingrese el primer valor (1 para abierto o 0 para cerrado): ")
 entrada2 = input("Ingrese el segundo valor (1 para abierto o 0 para cerrado): ")
 if not (entrada1.isdigit() and entrada2.isdigit() and int(entrada1) in [0, 1] and int(entrada2) in [0, 1]):
     print("Error: Las entradas deben ser 0 o 1.")
     entrada1 = int(entrada1)
     entrada2 = int(entrada2)
     if puertaDeingreso == "a":
         print("La compuerta AND está", "abierta, se permite el paso" if entrada1 and entrada2 else "cerrada, n
     elif puertaDeingreso == "o":
         print("La compuerta OR está", "abierta, se permite el paso" if entrada1 or entrada2 else "cerrada, no
     elif puertaDeingreso == "n":
        print(f"Se niegan las entradas: {int(not entrada1)} y {int(not entrada2)}")
     elif puertaDeingreso == "na":
         print("La compuerta NAND está", "abierta, se permite el paso" if not (entrada1 and entrada2) else "cer
     elif puertaDeingreso == "no":
         print("La compuerta NOR está", "abierta, se permite el paso" if not (entrada1 or entrada2) else "cerra
     elif puertaDeingreso == "x":
         print("La compuerta XOR está", "abierta, se permite el paso" if entrada1 != entrada2 else "cerrada, no
         print("Error: Compuerta no válida. Ingrese a, o, n, na, no o x.")
```

En esta captura se muestra cómo la IA propuso una refactorización utilizando el método **isdigit**, el cual no conocíamos. Además, nos brindó una explicación clara sobre por qué lo usó y cómo funciona. Suma además un ejemplo de uso.



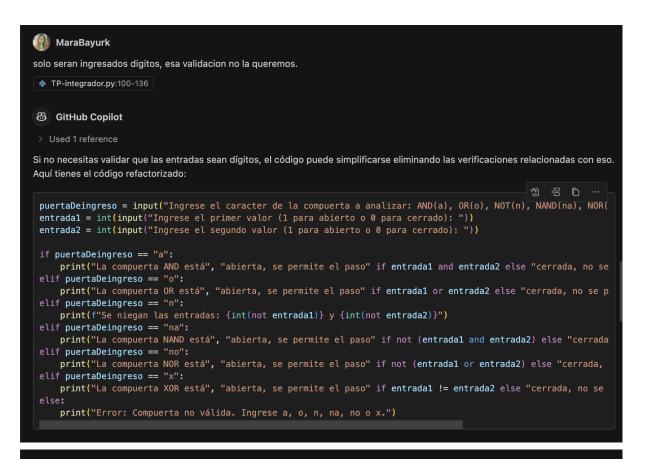
```
Uso común:

Se utiliza para validar que una entrada del usuario o una cadena contiene únicamente números antes de convertirla a un tipo numérico como  int o float. Por ejemplo:

entrada = input("Ingrese un número: ")
  if entrada.isdigit():
    numero = int(entrada)
    print(f"El número ingresado es: {numero}")
  else:
    print("Error: Debe ingresar solo números.")

♣ ♥ ♣ …
```

En esta captura le indicamos que, como solo serán ingresados dígitos, esa validación es innecesaria, por lo que le estamos dando otra restricción. Luego nos indica los cambios realizados teniendo en cuenta lo solicitado.



#### Cambios realizados:

- 1. Eliminación de validaciones de entrada:
  - $\circ$  Se asume que las entradas siempre serán números enteros válidos (0 o 1).
- 2. Simplificación de condiciones:
  - Se utiliza strip().lower() para normalizar la entrada de la compuerta, eliminando espacios y manejando mayúsculas/minúsculas.
- 3. Mensajes claros:
  - $\,\circ\,$  Los mensajes de salida se mantienen compactos y claros.
- △ ▽ 🖺 …

Aquí, le solicitamos agregar una validación que no fue tenida en cuenta por la IA en primera instancia. Y nos indica los cambios realizados.

```
MaraBayurk
a eso agrega la validacion si la entrada es un numero distinto de 1 y 0
 TP-integrador.py:100-136

    GitHub Copilot

> Used 1 reference
Aquí tienes el código actualizado con la validación para verificar si las entradas son números distintos de 1 y 0:
 puertaDeingreso = input("Ingrese el caracter de la compuerta a analizar: AND(a), OR(o), NOT(n), NAND(na), NOR(
 entrada1 = int(input("Ingrese el primer valor (1 para abierto o 0 para cerrado): "))
 entrada2 = int(input("Ingrese el segundo valor (1 para abierto o 0 para cerrado): "))
 if entrada1 not in [0, 1] or entrada2 not in [0, 1]:
     print("Error: Las entradas deben ser 0 o 1.")
     if puertaDeingreso == "a":
         print("La compuerta AND está", "abierta, se permite el paso" if entrada1 and entrada2 else "cerrada, n
     elif puertaDeingreso == "o":
         print("La compuerta OR está", "abierta, se permite el paso" if entrada1 or entrada2 else "cerrada, no
     elif puertaDeingreso == "n":
        print(f"Se niegan las entradas: {int(not entrada1)} y {int(not entrada2)}")
     elif puertaDeingreso == "na":
         print("La compuerta NAND está", "abierta, se permite el paso" if not (entrada1 and entrada2) else "cer
     elif puertaDeingreso == "no":
        print("La compuerta NOR está", "abierta, se permite el paso" if not (entrada1 or entrada2) else "cerra
     elif puertaDeingreso == "x":
        print("La compuerta XOR está", "abierta, se permite el paso" if entrada1 != entrada2 else "cerrada, no
         print("Error: Compuerta no válida. Ingrese a, o, n, na, no o x.")
```

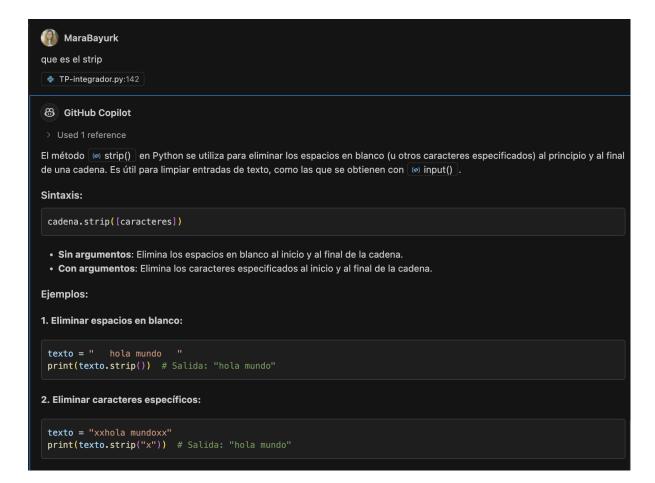
#### Cambios realizados:

- 1. Validación de entradas:
  - ∘ Se verifica que 📵 entrada1 y 📵 entrada2 estén en el rango permitido [0, 1].
  - o Si alguna de las entradas no es válida, se muestra un mensaje de error y no se ejecuta el resto del código.
- 2. Estructura compacta:
  - o Se mantiene la lógica de las compuertas sin cambios, pero ahora solo se ejecuta si las entradas son válidas.

Este enfoque asegura que el programa no procese valores fuera del rango esperado.

△ ♥ 🖺 …

En esta captura le consultamos a la IA que es el strip, un método que desconociamos, y nos dio además ejemplos del uso del mismo.



```
3. Uso con input():

Cuando un usuario ingresa texto con espacios accidentales:

entrada = input("Escribe algo: ").strip()
print(entrada)

Si el usuario escribe " hola ", el resultado será "hola".

Variantes:

• lstrip(): Elimina solo los espacios o caracteres al inicio de la cadena.

• rstrip(): Elimina solo los espacios o caracteres al final de la cadena.

Ejemplo:

texto = " hola mundo "
print(texto.lstrip()) # Salida: "hola mundo "
print(texto.rstrip()) # Salida: "hola mundo"
```

## Conclusión

Para finalizar, pudimos ver que existen múltiples maneras de resolver un mismo problema, y cada una aporta una perspectiva diferente.

A través de nuestras tres propuestas, aplicamos los conocimientos adquiridos en programación y matemática, desde estructuras condicionales hasta el uso de bucles y operadores lógicos.

Además, experimentamos con la inteligencia artificial como herramienta complementaria para mejorar y optimizar el código. Esta experiencia nos permitió no solo reforzar lo aprendido, sino también abrirnos a nuevas formas de pensar y resolver desafíos de manera más eficiente.