PARTE A:

El código de la parte A tiene las siguientes funciones:

1. **str_a_lista**:
    - Pasa de una cadena de caracteres a una lista, ignorando los espacios en blanco.
    - Esto facilita el procesamiento de la fórmula, ya que se trabaja con una lista de caracteres en lugar de una cadena.
2. **es_formula_valida**:
    - Verifica si la fórmula es válida según tres criterios:
        - Si el carácter es una letra (átomo).
        - Todos los átomos en la fórmula están definidos en el diccionario valores.
        - Los paréntesis están balanceados.
3. **evaluar_formula**:
    - Evalúa una fórmula lógica recursivamente, teniendo en cuenta los operadores ! (negación), & (conjunción) y | (disyunción).
    - Devuelve True o False dependiendo del valor de verdad de la fórmula.
4. **main**:
    - Lee la entrada, procesa las fórmulas y muestra los resultados.

PARTE B:

El código de la Parte B tiene las siguientes funciones:

1. **generar_combinaciones:**
    - Genera todas las combinaciones posibles de valores de verdad para los átomos de una fórmula.
    - Esto permite evaluar la fórmula en todas las combinaciones posibles de sus átomos.
2. **tipo_formula:**
    - Determina si una fórmula es una tautología**,** contradicción o contingencia**.**
3. **Integración en main:**
    - Lee las fórmulas de la Parte B, obtiene sus átomos, genera las combinaciones y determina su tipo utilizando tipo_formula.

Explicación más detallada Parte A:

Primero, tomo “sys” de la librería de Python, e importo stdin (standard input), que me permite leer datos de entrada. Luego convierto valores en un diccionario vacío.

1. **str_a_lista:**
    - En esta función, toma una cadena de caracteres (strings) que es la fórmula y set que es un conjunto, en este caso, llamado ignorar, y va a devolver una lista de caracteres (strings).

      Hice una lista vacía llamada ans, para almacenar los caracteres después del proceso. Hago un ciclo por cada carácter en la cadena de caracteres, y si el carácter NO está en el conjunto ignorar, agrego el carácter a la lista ans. Y al final hago un return a la lista ans.

2. **es_formula_valida:**
    - En esta función, lo que hice fue pasarle la formula en forma de lista, y que me lo devuelva como un booleano para verificar si la fórmula es válida o no. Tomo la variable “valida” como True, y recorro cada carácter que este en la formula con un for. Si ese carácter es una átomo devuelve True, y si no devuelve False (eso es lo que hace .isalpha).

        - Si el carácter SI es una átomo, y NO está en valores, la formula NO es válida. Para que sea válida es que el carácter sea un átomo y este en valores.

      Después, inicializo un contador en 0, y recorro cada carácter en la formula:

        - Si el carácter es un paréntesis abierto “(“, le sumo 1 al contador y si es un paréntesis cerrado “)”, le resto 1 al contador.
        - Si en algún momento el contador es menor que 0, la formula NO es válida, ya que se estaría cerrando un paréntesis sin primero abrir uno, lo cual esto invalida la formula.

      Y por último, hago un return dependiendo de la condición; solo si la fórmula es válida y contador == 0, devuelve True. En algún otro caso la formula devuelve False.

3. **evaluar_formula**:

   En esta función lo que hice fue pasarle la fórmula en forma de lista, junto con los índices de inicio y fin como unos enteros, y devolver un valor booleano (True o False) que representa el valor de verdad de la fórmula evaluada.

Hice una variable llamada “resultado” y le di el valor de False.

- Si inicio == fin, quiere decir que la fórmula es un átomo y devuelvo el valor de ese átomo que está en el diccionario, porque “formula[inicio]” es el átomo, y “valores[formula[inicio]]” es el valor que tiene el átomo en el diccionario “valores”.

- Si el inicio de la fórmula es igual a “!”, que en mi caso significa negación, el resultado es un llamado recursivo a la misma fórmula pero con un not antes, es decir “not evaluar_formula(formula, inicio + 1, fin)”. Esto evalúa la formula negada desde inicio + 1, porque ya sabemos que el primer carácter es la negación, y ese ya lo hicimos, por lo que el inicio pasa a ser un carácter a la derecha y le fin sigue siendo el mismo.

- Si el inicio de la fórmula es el carácter “(“ y el fin es “)”, esto significa que la formula esta entre paréntesis y lo que hace es buscar el operador principal de la siguiente manera:

    - Se inicializa un contador en 0, i en inicio + 1 porque ya sabemos que el primero es un “(“, y inicializamos una variable llamada “operadorPrincipal” en -1 (que lo que va a ser es tomar la posición del operador principal, además este no puede ser 0 porque tomaría el valor del primer carácter de la formula, lo cual no seria correcto). Mientras que i sea menor que fin y el operadorPrincipal sea -1 se hace lo siguiente:.
        - Si formula[i], ósea el carácter que sigue es “(“, suma 1 al contador, y si es “)”, resta 1 al contador.
        - Si la formula[i] está en {‘&’, ‘|’} (que son los operadores and, y or respectivamente), y el contador de los paréntesis es igual a 0, entonces la variable operadorPrincipal pasa a ser i, que es la posición del operador, pero si el carácter no es un operador o el contador no es 0 el operador principal sigue en -1.

Cada vez que hace todo este proceso se le suma 1 a i, hasta que encuentra cual es el operador principal, (si no se encuentra, es un átomo o la formula está mal formulada).

Luego, si el operador principal es diferente de -1, es decir, ya tiene la posición del operador y teniendo en cuenta que “formula[operadorPrincipal]” devuelve el carácter que este en la posición de i que es el operador principal. Y que el resultado lo separo en dos partes para que sea mucho más fácil, en la primera parte antes del operador principal y en la segunda parte después del operador principal, y esas dos fórmulas las uno mediante un and o el or según lo necesite.

Es decir la función evalua_formula() recibe tres parámetros que son formula, inicio y fin. Tomando en cuenta esto, en la primera parte es evaluar_formula(formula, inicio + 1, operadorPrincipal - 1), lo que quiere decir esta parte, es que se evalué la fórmula que se pasa en el primer parámetro, desde inicio + 1 (que es después del paréntesis) hasta el operador principal – 1 (que es antes del operador). Y la segunda parte es evaluar_formula(formula, operadorPrincipal + 1, fin - 1), lo que quiere decir que evalúa la formula desde el operador principal + 1 (es decir a la derecha de esa posición) hasta el fin – 1 (ya que fin es el carácter “)”).

Entonces:

- Si es == ‘&’, la variable resultado es la primera parte and (operador) la segunda parte.
- Si es == ‘|’, la variable resultado es la primera parte or (operador) la segunda parte.

Y por último hago un return a la variable resultado que tomara el valor de True o False.

4. **main**:
    - En esta función, con la variable “numeroAtomos”, leo los átomos que se ingresen, hice un ciclo con for en el rango de numeroAtomos, defino nombreAtomo, valor, que es el átomo (p,q,r…) con su respectivo valor (1, 0 en string), luego convierto valor a entero (se pasa primero como string porque es lo que el usuario entra, entonces toca convertirlo), y después a booleano. Y guarda el átomo y el valor en el diccionario “valores”.
    - Luego, Con la variable “numeroFormulas_A”, leo el número de fórmulas que se quieren evaluar y lo paso a un número entero. Hice un ciclo con for en el rango de numeroFormulas_A, paso la formula a una lista haciendo uso de la función anteriormente definida str_a_lista(), después, llamo a la función de es_formula_valida de la siguiente manera, si no es válida la formula, hago un print(“-1”), y si la fórmula es válida, resultado es igual a la función evaluar_formula(formula, 0, len(formula) – 1), que es desde el primer carácter, hasta el último carácter.
    - Si resultado después de hacer la función evaluar_formula es True devuelve un 1, y si es False devuelve 0.

Explicación más detallada Parte B:

1. **generar_combinaciones:**

   En esta función, se toman 4 parámetros, una lista llamada átomos (que solo tiene átomos), un entero llamado posición que es la posición de la lista de los átomos, un diccionario llamado valorActual que almacena la combinación actual de valores de verdad, y combinaciones que es una lista que almacena todas las combinaciones.

   - Si la posición actual es igual a la longitud de la lista atomos, significa que ya se completó una combinación, entonces se agrega a la lista "combinaciones". Se usa el .copy() para hacer una copia independiente del diccionario "valorActual", porque se van a estar modificando los valores de los átomos para obtener todas las combinaciones posibles.
   - Al átomo actual se le asigna el valor de True, luego se llama recursivamente a la función generar_combinaciones con el siguiente átomo (posición + 1), y se le da un valor. Luego, verifica, si no hay más átomos, guarda esa combinación en la lista combinaciones. Para el segundo átomo le da el otro valor y vuelve a hacer lo mismo. Después, vamos al primer átomo y le damos el valor que nos falta, y hacemos el mismo proceso, así es como se obtiene todas las combinaciones posibles.

2. **tipo_formula:**

   En esta función, se reciben dos parámetros, la lista de la formula y la lista de átomos. Esta función devuelve un entero porque si es Tautología devuelve 1, si es Contradicción devuelve 0 y si es Contingencia devuelve -1.

   - Crea la lista llamada "combinaciones" que se usa en la anterior función, y genera todas las combinaciones posibles.
   - Se crea otra lista, llamada "resultados", en la cual se hace un ciclo por cada combinación en la lista de combinaciones, luego por cada átomo, valor en combinación.items() (el .ítems devuelve la llave-valor del diccionario, en este caso átomo, valor que es como yo lo definí), y le asigna a cada átomo su respectivo valor.
   - Luego, se llama a la función evaluar_formula para evaluar la fórmula con los valores actuales, lo que va a devolver True o False, y esto se va a agregar a la lista llamada "resultado".
     - Si todos los resultados son True, retorna 1, que significa que es una Tautología.
     - Si ningún elemento es True, retorna 0, que significa que es una Contradicción.

- Y si no es ninguna de esas dos, retorna -1, que significa que es una Contingencia (unas verdaderas y otras falsas).

3. **main:**
    - En esta función, con la variable “numeroFormulas_B”, leo el número de fórmulas que se quieren evaluar y lo paso a un número entero. Hice un ciclo con for en el rango de numeroFormulas_A, paso la formula a una lista haciendo uso de la función anteriormente definida str_a_lista().
    - Creo una lista vacía llamada atomos, y hago un ciclo por cada carácter en la formula, si el carácter es un átomo, entonces agrego el carácter a la lista “atomos”.
    - Luego, atomos_formula, es igual a list(set(atomos)). Esto lo que hace es que la lista “atomos” la convierto en un conjunto, porque los conjuntos no permiten elementos duplicados. Después ese conjunto lo convierto en una lista de nuevo.
    - Por último, la variable “tipo” es igual a la función tipo_formula, el cual se le pasan los parámetros de la formula y atomos_formula que se acabó de definir. Hago un print(tipo), que lo que va a hacer es ir a la función tipo_formula, ejecutarla y retornarme los valores correspondientes para la Tautología, Contradicción y Contingencia.