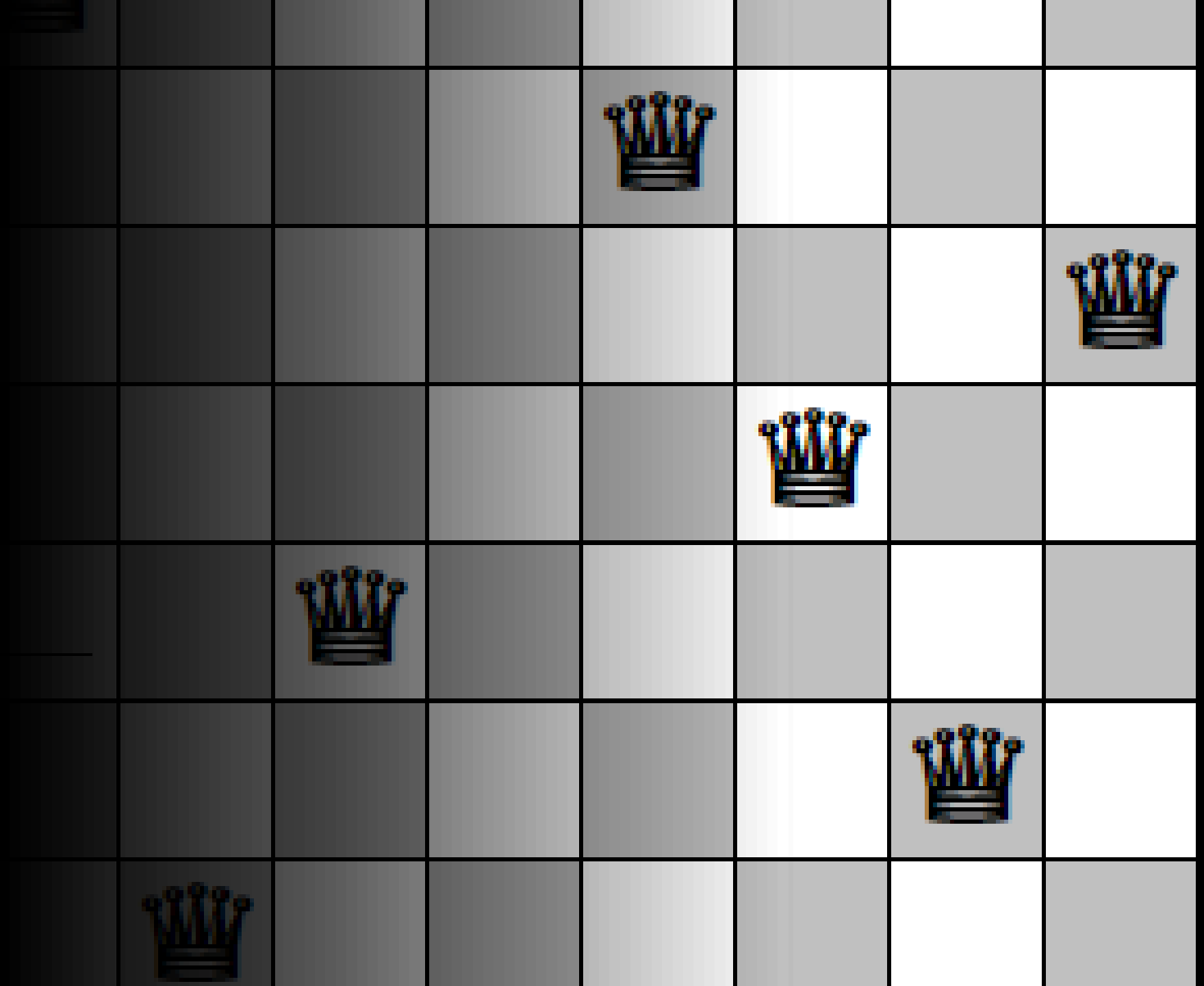
Backtracking



¿Qué es Backtracking?

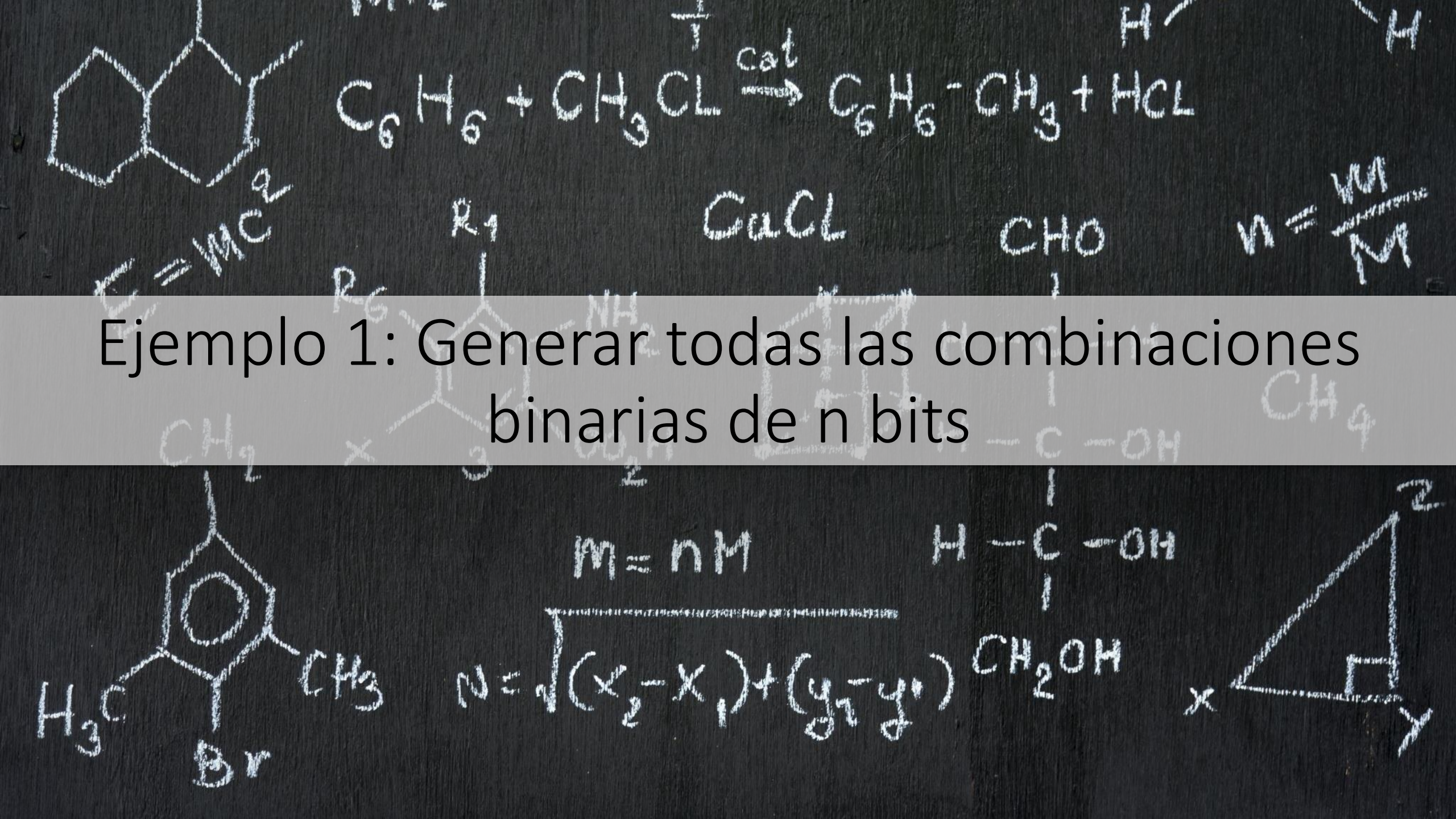
- **Backtracking** es una técnica de programación que consiste en **explorar todas las posibles soluciones a un problema** construyendo paso a paso una solución y retrocediendo (hacer *backtrack*) cuando se detecta que una elección no lleva a una solución válida.

Base del algoritmo

1. **Elegir** una opción.
2. **Probar** si esa opción es válida.
3. **Avanzar** si es válida o **retroceder** si no lo es.
4. **Repetir** hasta encontrar una o todas las soluciones.

¿Cuándo usar backtracking?

- Cuando el problema tiene:
  - Muchas combinaciones posibles.
  - Restricciones que deben cumplirse.
  - Soluciones válidas que pueden construirse progresivamente.



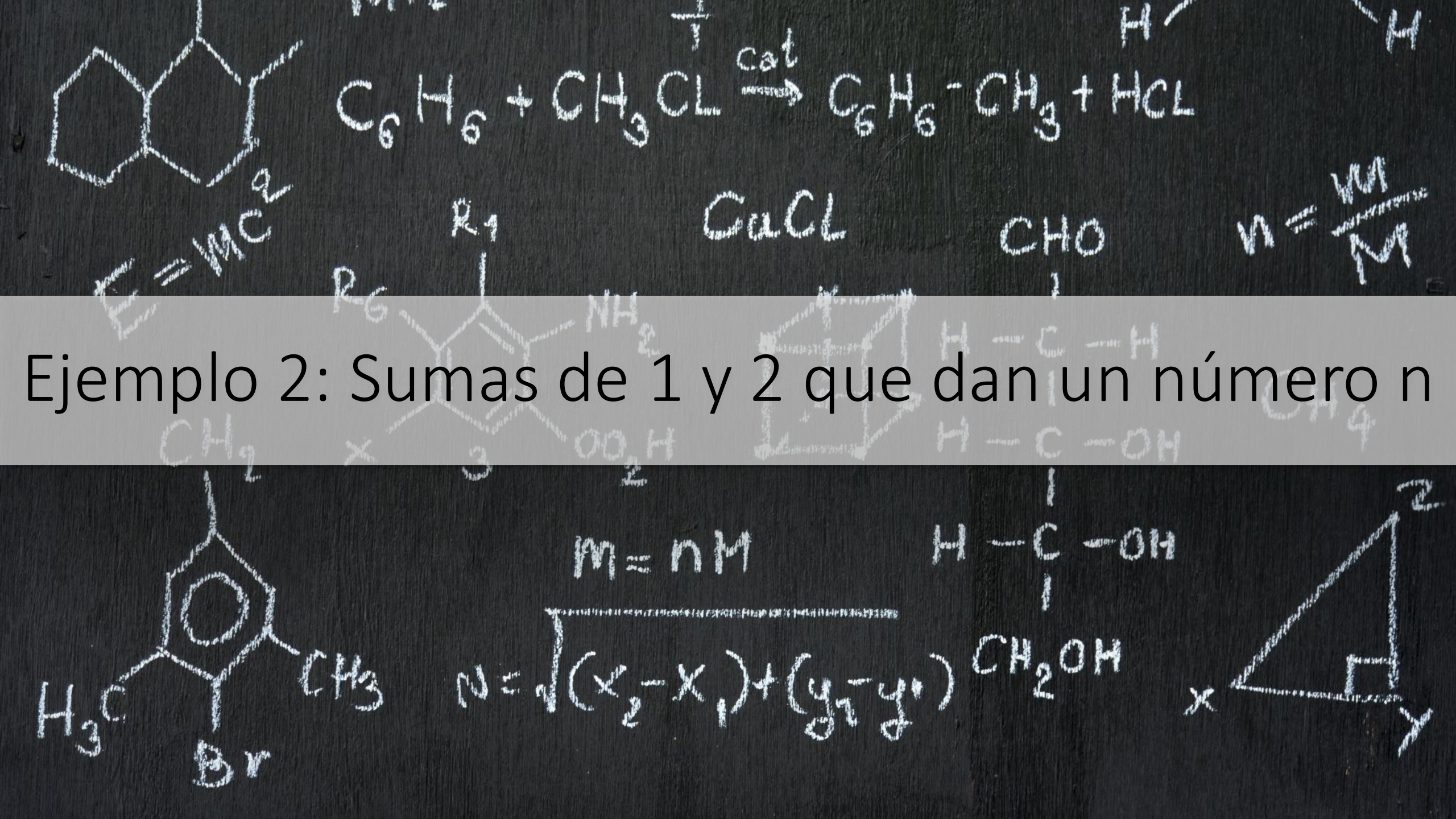
Ejemplo 1: Generar todas las combinaciones binarias de n bits

Ejemplo 1: Generar todas las combinaciones binarias de n bits

```python
def generar_binarios(n, cadena=""):
    if len(cadena) == n:
        print(cadena)
        return
    generar_binarios(n, cadena + "0")
    generar_binarios(n, cadena + "1")

# Ejemplo:
generar_binarios(3)
```

```
000
001
010
011
100
101
110
111
```



Ejemplo 2: Sumas de 1 y 2 que dan un número n

Ejemplo 2: Sumas de 1 y 2 que dan un número n

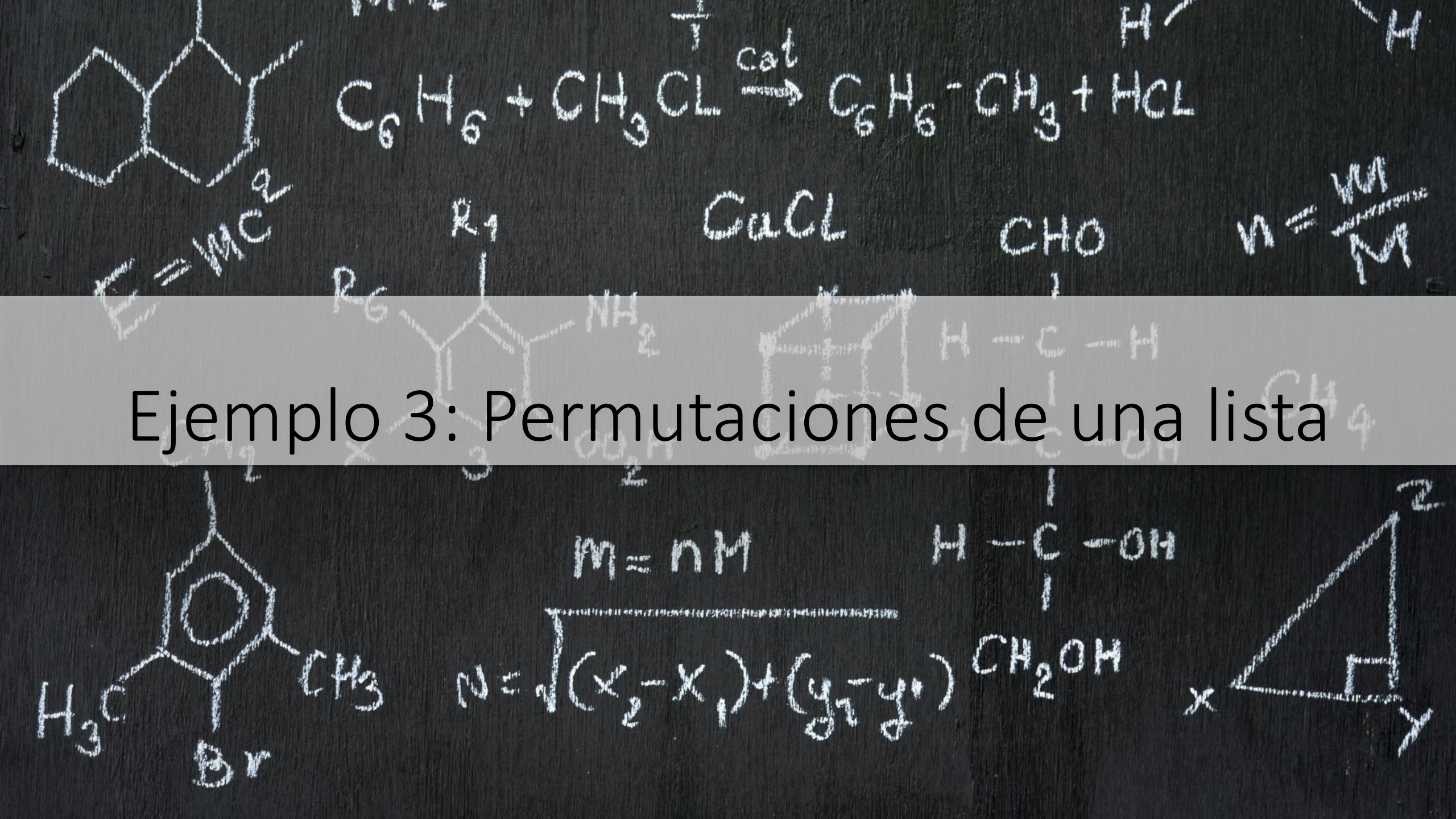
```python
def sumar_con_1_y_2(n, combinacion=[]):
    if sum(combinacion) == n:
        print(combinacion)
        return
    if sum(combinacion) > n:
        return  # nos pasamos, hay que retroceder

    # Opción 1: agregar un 1
    sumar_con_1_y_2(n, combinacion + [1])

    # Opción 2: agregar un 2
    sumar_con_1_y_2(n, combinacion + [2])

# Ejemplo:
sumar_con_1_y_2(4)
```

```
[1, 1, 1, 1]
[1, 1, 2]
[1, 2, 1]
[2, 1, 1]
[2, 2]
```



Ejemplo 3: Permutaciones de una lista

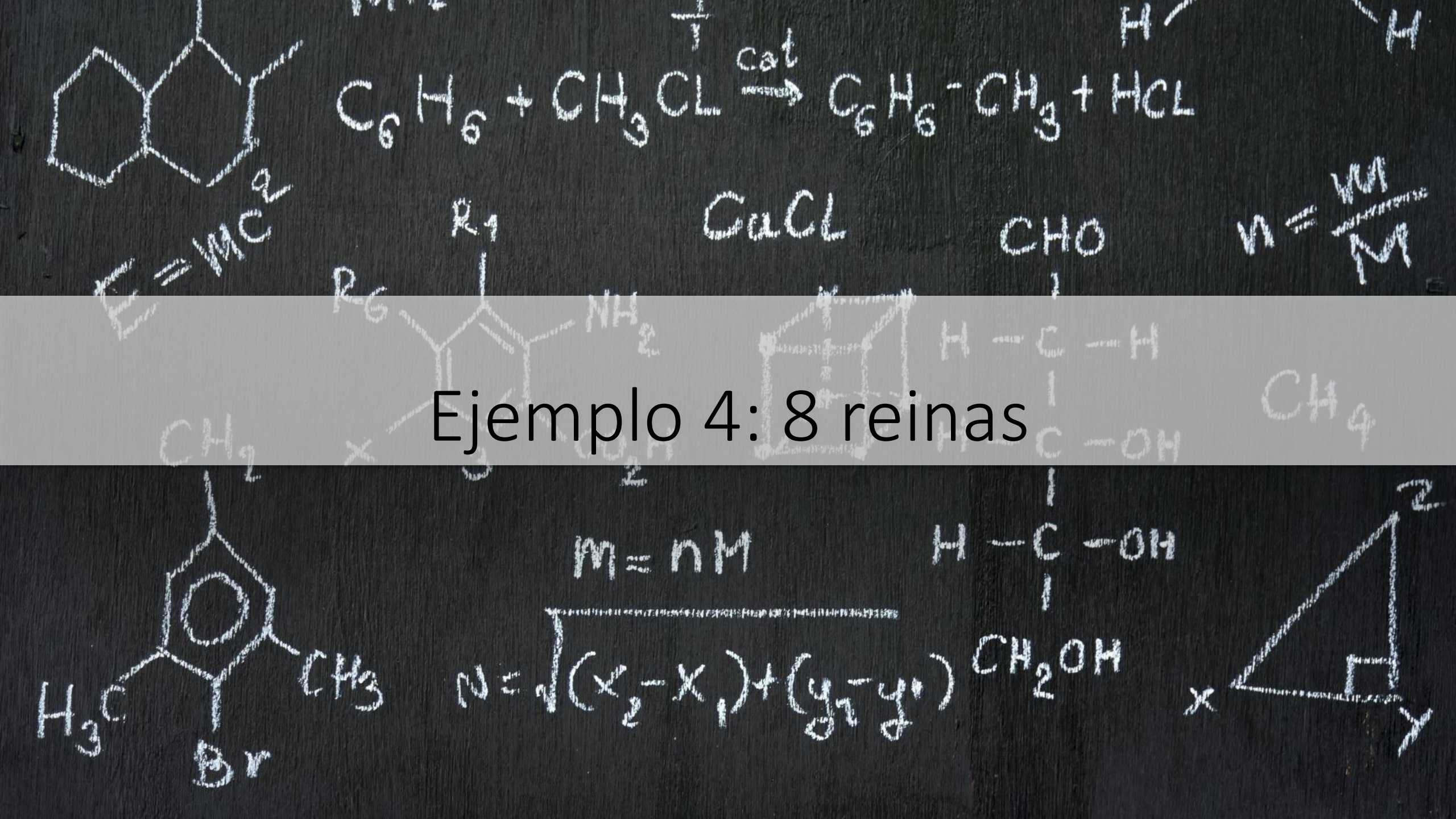
Ejemplo 3. Permutaciones de una lista

```python
def permutar(lista, camino=[]):
    if not lista:
        print(camino)
        return

    for i in range(len(lista)):
        # Elegimos el elemento i
        nuevo_elemento = lista[i]
        # Generamos una nueva lista sin ese elemento
        resto = lista[:i] + lista[i+1:]
        # Exploramos con esa nueva elección
        permutar(resto, camino + [nuevo_elemento])

# Ejemplo:
permutar([1, 2, 3])
```

```
[1, 2, 3]
[1, 3, 2]
[2, 1, 3]
[2, 3, 1]
[3, 1, 2]
[3, 2, 1]
```



Ejemplo 4: 8 reinas

Ejemplo 4: 8 reinas

```python
def es_seguro(tablero, fila, col):
    for i in range(fila):
        if tablero[i] == col or \
           tablero[i] - i == col - fila or \
           tablero[i] + i == col + fila:
            return False
    return True

def resolver_reinas(tablero, fila):
    if fila == len(tablero):
        imprimir_tablero(tablero)
        return True  # ¡Encontramos una solución!

    for col in range(8):
        if es_seguro(tablero, fila, col):
            tablero[fila] = col
            if resolver_reinas(tablero, fila + 1):
                return True  # Propagar hacia arriba para detener todo
            tablero[fila] = -1  # backtrack
    return False  # No se encontró solución en esta rama
```

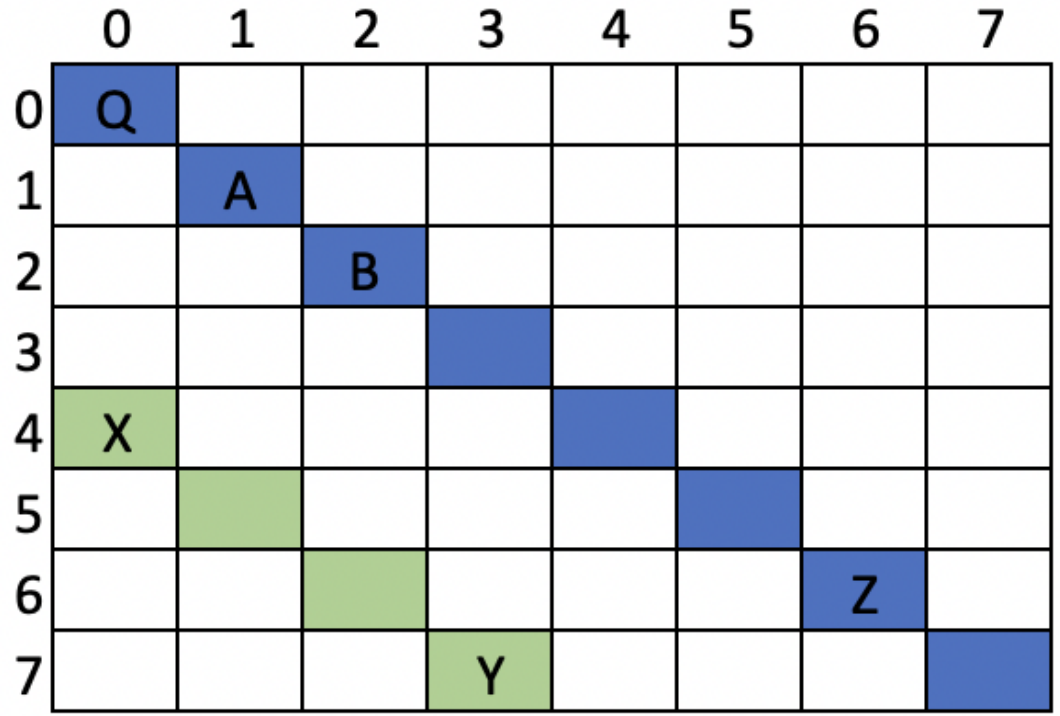
```python
def imprimir_tablero(tablero):
    for i in range(8):
        fila = ["."] * 8
        fila[tablero[i]] = "Q"
        print(" ".join(fila))
    print()

# Inicia con todas las posiciones vacías
tablero = [-1] * 8
resolver_reinas(tablero, 0)
```

Resultado:

92 soluciones

Algoritmo es_seguro



Si se resta la fila menos la columna de dos celdas, y el resultado es cero, implica que están en la misma diagonal

	fila		col		
A	1	-	1	=	0
B	2	-	2	=	0
Z	6	-	6	=	0
Q	0	-	0	=	0
X	4	-	0	=	4
Y	7	-	3	=	4