```python
#EJERCICIO1

def imprimir_matriz(m):
    for fila in m:
        for elemento in fila:
            print(elemento)

m = [[1, 2, 3],
     [4, 5, 6],
     [7, 8, 9]]

imprimir_matriz(m)

#EJERCICIO 2 Suma de Matrices #
def suma_elementos_matriz(m):
    suma = 0
    for fila in m:
        for elemento in fila:
            suma += elemento
    return suma

m = [[1, 2, 3],
     [2, 2, 2],
     [1, 2, 3]]

m2 = [[1, 2, 3],
      [2, 2, 2],
      [1, 2, 3],
      [1, 2, 3]]

        # Calcular y mostrar la suma de la primera matriz
resultado1 = suma_elementos_matriz(m)
print("La suma de los elementos en la primera matriz es:", resultado1)

        # Calcular y mostrar la suma de la segunda matriz
resultado2 = suma_elementos_matriz(m2)
print("La suma de los elementos en la segunda matriz es:", resultado2)

        # Verificar las sumas usando assert
assert resultado1 == 18
assert resultado2 == 24


#EJERCICIO 3 n_diagonal
def n_diagonal(m, n):
    rows, cols = len(m), len(m[0])  # Obtenemos el numero de filas y columnas de la matriz

    # Recorremos la matriz y verificamos si hay mas de 'n' elementos distintos de cero fuera de la diagonal
    for i in range(rows):
        for j in range(cols):
            if i != j and m[i][j] != 0:
                if abs(i - j) > n:  # Verificamos si la posicion esta mas lejos de la diagonal de lo permitido
                    return False
    return True

# Ejemplos de uso
m_diagonal = [[1, 0, 0],
              [0, 7, 0],
              [0, 0, 3]]
assert n_diagonal(m_diagonal, 0) == True

m_no_diagonal1 = [[1, 0, 2],
                  [0, 7, 0],
                  [0, 0, 3]]
assert n_diagonal(m_no_diagonal1, 1) == True

m_no_diagonal2 = [[1, 0, 2],
                  [0, 7, 0],
                  [0, 3, 3]]
assert n_diagonal(m_no_diagonal2, 2) == True

m_no_diagonal3 = [[0, 0, 2],
                  [0, 7, 0],
                  [2, 3, 3]]
assert n_diagonal(m_no_diagonal3, 2) == False

#EJERCICIO 4 MAX-MIN
def max_min(m):
    if not m:
        return None, None

    # Inicializamos el minimo y el maximo con el primer elemento de la matriz
    min_elemento = max_elemento = m[0][0]

    # Recorremos la matriz para encontrar el minimo y el maximo
    for fila in m:
```

```
        for elemento in fila:
            if elemento < min_elemento:
                min_elemento = elemento
            if elemento > max_elemento:
                max_elemento = elemento

    return max_elemento, min_elemento


m = [[1, 0, 2],
     [0, 7, 0],
     [0, 0, 3]]

m_1_elemento = [[10]]

assert max_min(m) == (7, 0)
assert max_min(m_1_elemento) == (10, 10)

#EJERCICIO 5 Matrices bonitas
def bonita(m):
    n = len(m)


    for i in range(n):
        for j in range(n):
            if i != j and m[i][j] != 'b':
                return False


    for i in range(n):
        if m[i][i] != 'a':
            return False


    for i in range(n):
        for j in range(i + 1, n):
            if m[i][j] != 'c':
                return False

    return True


m_bonita = [['a', 'b', 'b', 'b'],
            ['c', 'a', 'b', 'b'],
            ['c', 'c', 'a', 'b'],
            ['c', 'c', 'c', 'a']]

m_no_bonita = [['a', 'b', 'b', 'b'],
               ['c', 'a', 'b', 'a'],
               ['c', 'c', 'a', 'b'],
               ['c', 'c', 'c', 'a']]

print(bonita(m_bonita))
print(bonita(m_no_bonita))

#EJERCICIO 6 TRANSPUERTA
def es_transpuesta(m1, m2):
    n = len(m1)

    # Comparamos los elementos de m1 con los elementos de m2 en posiciones intercambiadas
    for i in range(n):
        for j in range(n):
            if m1[i][j] != m2[j][i]:
                return False

    return True

mat1 = [[2, 0, 0, 0],
        [3, 2, 0, 0],
        [3, 3, 2, 0],
        [3, 3, 3, 2]]

mat2 = [[2, 3, 3, 3],
        [0, 2, 3, 3],
        [0, 0, 2, 3],
        [0, 0, 0, 2]]

mat3 = [[1, 1, 1, 0],
        [1, 1, 0, 0],
        [1, 0, 0, 0],
        [0, 0, 0, 0]]

mat4 = [[0, 0, 0, 0],
        [0, 0, 0, 1],
        [0, 0, 1, 1],
        [0, 1, 1, 1]]
```

```python
assert es_transpuesta(mat1, mat2)
assert not es_transpuesta(mat3, mat4)


#EJERCICIO 7 SUMA DE MATRICES

def crear_matriz_zeros(n_filas, n_columnas):
    m = []
    for i in range(n_filas):
        m.append([0] * n_columnas)
    return m

def suma_matrices(m1, m2):
    if len(m1) != len(m2) or len(m1[0]) != len(m2[0]):
        raise ValueError("Las matrices deben tener el mismo tamaNo para sumarse.")

    filas, columnas = len(m1), len(m1[0])

    resultado = crear_matriz_zeros(filas, columnas)

    for i in range(filas):
        for j in range(columnas):
            resultado[i][j] = m1[i][j] + m2[i][j]

    return resultado

mat1 = [[2, 0, 0, 0],
        [3, 2, 0, 0],
        [3, 3, 2, 0],
        [3, 3, 3, 2]]

mat2 = [[2, 3, 3, 3],
        [0, 2, 3, 3],
        [0, 0, 2, 3],
        [0, 0, 0, 2]]

resultado = suma_matrices(mat1, mat2)

for fila in resultado:
    print(fila)

#EJERCICIO 8 SUMA DE BORDES
def suma_bordes(m):
    if not m:
        return 0

    filas = len(m)
    columnas = len(m[0])

    suma = 0

    for i in range(filas):
        for j in range(columnas):
            if i == 0 or i == filas - 1 or j == 0 or j == columnas - 1:
                suma += m[i][j]

    return suma

# Ejemplos de uso
mat1 = [[2, 0, 8, 0],
        [3, 2, 0, 6],
        [3, 3, 2, 0],
        [3, 3, 3, 2]]

assert suma_bordes(mat1) == 33

mat2 = [[1, 2, 3, 4],
        [3, 2, 0, 6],
        [3, 3, 2, 0],
        [3, 3, 3, 2],
        [3, 3, 3, 2]]

assert suma_bordes(mat2) == 38

#EJERCICIO 9 SOLO
def solo_1(m):
    filas = len(m)
    columnas = len(m[0])

    # Verificar cada fila
    for fila in m:
        if fila.count(1) != 1:
            return False

    # Verificar cada columna
    for j in range(columnas):
        columna = [m[i][j] for i in range(filas)]
```

```
        if columna.count(1) != 1:
            return False

    return True


m1 = [[0, 0, 1],
      [1, 0, 0],
      [0, 1, 0]]
assert solo_1(m1)

m2 = [[0, 0, 1],
      [1, 0, 0],
      [0, 0, 1]]
assert not solo_1(m2)

m3 = [[0, 1],
      [1, 0]]
assert solo_1(m3)

m4 = [[0, 0],
      [1, 0]]
assert not solo_1(m4)

m5 = [[1, 1, 0],
      [0, 0, 0],
      [0, 0, 1]]
assert not solo_1(m5)
```