1)

#a) A estrutura de uma matriz 0-espiral é tal que seus elementos formam uma progressão aritmética de razão 0 com início em 1. Desta forma, a matriz A ∈ Z ^ n×n possui todos seus n ** 2 elementos iguais a 1.

#b)

```
def passos():
  return [
    (0, 1), (1, 0),
    (0, -1), (-1, 0)
  ]

def verifica_r_espiral(A: [[int]]) -> int:
  n, r, anterior = len(A), None, None
  if n == 0 or len(A[0]) != n:
    return r
  inicio, fim = [0, 0], [n - 1, n - 1]
  direcoes = passos()
  cont = 0
  i, j = inicio
  while inicio[0] <= fim[0] and inicio[1] <= fim[1]:
    atual = A[i][j]
    if anterior is None:
      if atual != 1:
        return None
    else:
      if r is None:
        r = atual - anterior
      elif atual != anterior + r:
        return None
    anterior = atual
    di, dj = direcoes[cont]
```

```
        proximo = (i + di, j + dj)
        if proximo[1] > fim[1] or proximo[0] > fim[0] or proximo[1] < inicio[1] or proximo[0] < inicio[0]:
            match cont:
                case 0: inicio[0] += 1
                case 1: fim[1] -= 1
                case 2: fim[0] -= 1
                case 3: inicio[1] += 1
            cont = (cont + 1) % 4
            di, dj = direcoes[cont]
        i, j = i + di, j + dj
    return r if r is not None else 0


#c)
def gera_espiral(r: int, n: int) -> list[list[int]]:
    A = [[0] * n for _ in range(n)]
    direcoes = passos()
    i = j = direcao = 0
    valor = 1
    for _ in range(n * n):
        A[i][j] = valor
        valor += r
        di, dj = direcoes[direcao]
        ni, nj = i + di, j + dj
        if 0 <= ni < n and 0 <= nj < n and A[ni][nj] == 0:
            i, j = ni, nj
        else:
            direcao = (direcao + 1) % 4
            di, dj = direcoes[direcao]
            i, j = i + di, j + dj
    return A
```

2)

#a) A matriz A é C-simétric se, e somente se, para todo 0 ≤ i,j < n vale: A[i][j] == A[k][l] tal que o k = n - i - 1 e l = n - j - 1, isso implica que A[i][j] = A[n - i - 1][n - j - 1]

#b)

```
def verifica_C_simetrica(A: [[float]]) -> bool:
  n = len(A)
  if n == 1 and len(A[0]) == 0: return True
  B = [[False] * len(A[0]) for _ in range(n)]
  for i in range(n):
    for j in range(n - i):
      k, l = n - i - 1, n - j - 1
      if B[i][j] == False and (i,j) != (k, l):
        if A[i][j] != A[k][l]: return False
        else: B[i][j] = B[k][l] = True
  return True
```

#c)

```
def verifica_D_constante(T: [[float]]) -> bool:
  n = len(T)
  for i in range(1, n):
    for j in range(1, n):
      if T[i][j] != T[i-1][j-1]: return False
  return True
```

#d) A matriz T, sendo D-constante, é também C-simétrica se, e somente se, para todo 0 ≤ i,j < n valer T[i][j] = T[j][i], isso implica que os valores nas diagonais acima da diagonal principal coincidem com os valores nas diagonais abaixo da principal, garantindo a C-simetria.

#e)

```
def cria_T(y: [float]) -> [[float]]:
```

```
    n = len(y)
    T = []
    for i in range(n):
        T.append([0] * i + y + [0] * (n - i - 1))
    return T


def z(x: [float], y: [float]) -> [float]:
    T = cria_T(y)
    n = len(x)
    m = 2 * n - 1
    z = [0] * m
    for j in range(m):
        for i in range(n):
            z[j] += x[i] * T[i][j]
    return z


#f) O produto z = x ^ T * T(y) equivale à convolução discreta entre x e y, pois cada coordenada z[k] = ∑(i = 0 à n - 1) x[i] * y[k - i], tal que y[k - i] = 0 se k - i está fora do intervalo [0, n-1]
```

3)

```
#a)
def inversa(q: [int]) -> [int]:
    n = len(q)
    inv = [0] * (n)
    for i, qi in enumerate(q):
        inv[qi - 1] = i + 1
    return inv

def sao_camaradas(p: [int], q: [int], r: [int]) -> bool:
    n = len(p)
    q_inv = inversa(q)
    for k in range(1, n + 1):
        if p[k - 1] != q[r[q_inv[k - 1] - 1] - 1]: return False
    return True

#b)
def decompor_ciclos(permut:[int]) -> (bool, [int]):
    n = len(permut)
    visitados = [False] * n
    ciclos = []
    for i in range(n):
        if not visitados[i]:
            ciclo = []
            atual = i
            while not visitados[atual]:
                visitados[atual] = True
                ciclo.append(atual + 1)
                atual = permut[atual] - 1
            idx = ciclo.index(min(ciclo))
            ciclos.append(ciclo[idx:] + ciclo[:idx])
```

```
    return sorted(ciclos, key=lambda x: (len(x), x))


def sao_camaradaveis(p: [int], r: [int]) -> (bool, list[int] | None):
    n = len(p)
    if len(r) != n: return (False, None)
    ciclos_p = decompor_ciclos(p)
    ciclos_r = decompor_ciclos(r)
    if sorted(len(c) for c in ciclos_p) != sorted(len(c) for c in ciclos_r): return (False, None)
    q = [0] * n
    for cp, cr in zip(ciclos_p, ciclos_r):
        if len(cp) != len(cr): return (False, None)
        for elem_p, elem_r in zip(cp, cr):
            q[elem_r - 1] = elem_p
    return (True, q)
```