# Unidad: Estructuras de datos básicas

## Arreglos, Matrices, búsqueda y ordenamiento

Introducción a la Programación

Noviembre de 2025

# Matrices: objetivo de la sesión

- Comprender la representación de una matriz como *arreglo de arreglos* y sus índices.
- Practicar recorridos dobles (por filas y por columnas).
- Implementar agregaciones básicas: suma total, por fila y por columna.
- Trabajar con diagonales en matrices cuadradas (principal y secundaria).
- Realizar búsqueda simple en 2D y una transformación útil (transpuesta).

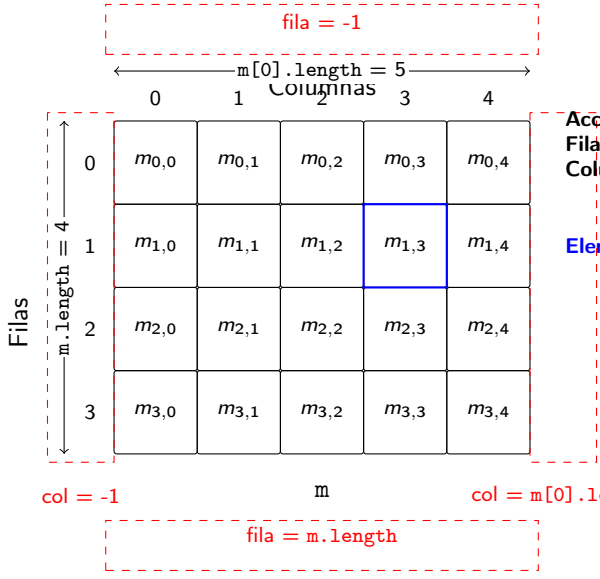
**Resultado esperado:** al finalizar, podrás leer, recorrer y operar matrices para resolver problemas sencillos en 2D.

# Matriz = arreglo de arreglos

- Una matriz puede verse como una colección de **filas**, donde cada fila es un arreglo.
- Acceso por índices: `m[i][j]` con $0 \leq i <$ `filas` y $0 \leq j <$ `columnas`.
- Dimensiones:
  - `filas = m.length`
  - `columnas = m[0].length` (*si todas las filas tienen igual longitud*)

**Idea clave:** pensamos la matriz por **filas** primero, luego por **columnas**.

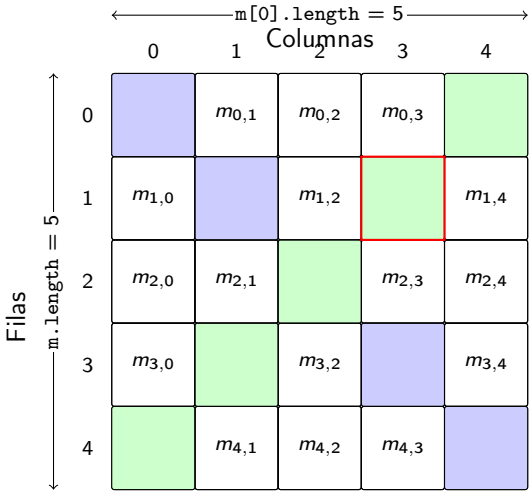
# Matriz: anatomía




**Acceso:** `m[i][j]`
**Filas:** `0` $\leq$ `i < m.length`
**Columnas:** `0` $\leq$ `j < m[0].length`

**Elemento:** `m[1][3]`

# Matriz: anatomía (diagonales)




**Acceso:** `m[i][j]`
**Principal:** azul ($i = j$)
**Secundaria:** verde ($i + j = 5 - 1$)

**Elemento:** `m[1][3]`

# Dimensiones y acceso

```
int[][] m = {
  {5, 1, 0},
  {2, 4, 7}
}; // 2 filas, 3 columnas

int filas = m.length;      // 2
int cols  = m[0].length;   // 3

int x = m[1][2];           // 7 (fila 1, col 2)
```

- **Rangos válidos:** `0 <= i < filas`, `0 <= j < cols`.
- **Precaución:** evitar asumir rectangularidad si no fue definida: ver `m[i].length`.

# Declaración y creación

- Declaración: `int[][] a;`
- Creación rectangular: `a = new int[F][C];` (todos los valores quedan en 0)
- Dimensiones:
  - `a.length` → número de filas
  - `a[0].length` → número de columnas (si es rectangular)

# Inicialización (literal) y precaución

```
int F = 3, C = 4;
int[][] a = new int[F][C]; // {0...}

int[][] b = {
  {1, 2, 3},
  {4, 5, 6}
}; // 2x3

// Precaucin: tcnicamente las filas pueden tener longitudes distintas.
int[][] c = {
  {9, 8},
  {7, 6, 5}
};
int filas = c.length;          // 2
int cols0 = c[0].length;       // 2
int cols1 = c[1].length;       // 3
```

- Si se asume rectangularidad, verificar: `for (int j = 0; j < m[i].length; j++){...}`

# Recorrido por filas (externo) y columnas (interno)

```
int[][] m = {
  {5, 1, 0},
  {2, 4, 7}
};

for (int i = 0; i < m.length; i++) {          // i recorre filas
  for (int j = 0; j < m[i].length; j++) {     // j recorre columnas de la
    fila i
    int v = m[i][j];
    // procesar v
  }
}
```

- Patrón habitual: primero filas (`i`), luego columnas (`j`).
- Usar `m[i].length` evita errores si alguna fila tiene distinta longitud.

# Recorrido por columnas (externo) y filas (interno)

```
int[][] m = {
  {5, 1, 0},
  {2, 4, 7}
};
// Si asumimos rectangularidad:
int filas = m.length;
int cols  = (filas == 0) ? 0 : m[0].length;
for (int j = 0; j < cols; j++) {      // j recorre columnas
  for (int i = 0; i < filas; i++) {   // i recorre filas
    int v = m[i][j];
    // procesar v
  }
}
```

- Útil cuando interesa operar **por columnas** (sumas por columna, máximos por columna, etc.).
- Si no es rectangular, recorrer con `for (int i=0; i<m.length; i++) if (j < m[i].length){...}`.

# Suma total de una matriz

```
int[][] m = {
  {5, 1, 0},
  {2, 4, 7}
};
int total = 0;
for (int i = 0; i < m.length; i++) {
  for (int j = 0; j < m[i].length; j++) {
    total += m[i][j];
  }
}
// usar 'total'
```

- Patrón base: doble `for` acumulando en una variable.
- Robusto ante filas con distinta longitud: usar `m[i].length`.

# Sumas por fila

```
int[][] m = {
  {5, 1, 0},
  {2, 4, 7}
};
for (int i = 0; i < m.length; i++) {
  int sumaFila = 0;
  for (int j = 0; j < m[i].length; j++) {
    sumaFila += m[i][j];
  }
  // usar 'sumaFila' para la fila i
}
```

- Un acumulador por cada fila.
- Útil para promedios por fila (dividir entre `m[i].length` si aplica).

# Sumas por columna

```
int[][] m = {
  {5, 1, 0},
  {2, 4, 7}
};
// si asumimos rectangularidad:
int filas = m.length;
int cols  = (filas == 0) ? 0 : m[0].length;
for (int j = 0; j < cols; j++) {
  int sumaCol = 0;
  for (int i = 0; i < filas; i++) {
    sumaCol += m[i][j];
  }
  // usar 'sumaCol' para la columna j
}
```

- Alternativamente, sin asumir rectangularidad:
- `for (int j=0; ; j++)` con verificación `j < m[i].length` por cada fila (más verboso).

# Diagonales en matriz cuadrada

- **Principal**: posiciones donde $i = j$.
- **Secundaria**: posiciones donde $i + j = n - 1$.
- Requiere matriz cuadrada ($n \times n$). Si no lo es, se puede usar $n = \text{mín}(\texttt{filas}, \texttt{columnas})$ para una versión parcial.

# Suma de diagonales (cuadrada)

```
int[][] m = {
  {1, 2, 3},
  {4, 5, 6},
  {7, 8, 9}
}; // 3x3

int n = m.length;            // asumir n x n
int dp = 0, ds = 0;

for (int i = 0; i < n; i++) {
  dp += m[i][i];             // principal
  ds += m[i][n - 1 - i];     // secundaria
}
// usar dp y ds
```

# Búsqueda en 2D

- Objetivo: localizar la **primera** aparición de un valor.
- Convención si no se encuentra: $(-1, -1)$.
- Patrón: doble `for` con ruptura anticipada.

# Primera aparición de un valor

```
int[][] m = {
  {5, 1, 0},
  {2, 4, 7}
};
int target = 7;
int fi = -1, fj = -1;

busca:
for (int i = 0; i < m.length; i++) {
  for (int j = 0; j < m[i].length; j++) {
    if (m[i][j] == target) { fi = i; fj = j; break busca; }
  }
}
// usar (fi, fj)
```

# Transformaciones: transpuesta e inversión por fila

- **Transpuesta**: $t[j][i] = m[i][j]$. Si $m$ es $F \times C$, entonces $t$ es $C \times F$.
- **Invertir cada fila**: intercambio de extremos hacia el centro (dos punteros).

# Transpuesta (rectangular)

```
int filas = m.length;
int cols  = (filas == 0) ? 0 : m[0].length;

int[][] t = new int[cols][filas];
for (int i = 0; i < filas; i++) {
  for (int j = 0; j < cols; j++) {
    t[j][i] = m[i][j];
  }
}
// usar t
```

# Invertir cada fila (dos punteros)

```
for (int i = 0; i < m.length; i++) {
  int j1 = 0, j2 = m[i].length - 1;
  while (j1 < j2) {
    int tmp = m[i][j1];
    m[i][j1] = m[i][j2];
    m[i][j2] = tmp;
    j1++; j2--;
  }
}
// m queda con cada fila invertida
```

# Transformaciones: transpuesta e inversión por fila

- **Transpuesta**: $t[j][i] = m[i][j]$. Si $m$ es $F \times C$, entonces $t$ es $C \times F$.
- **Invertir cada fila**: intercambio de extremos hacia el centro (dos punteros).

# Transpuesta (rectangular)

```
int filas = m.length;
int cols  = (filas == 0) ? 0 : m[0].length;

int[][] t = new int[cols][filas];
for (int i = 0; i < filas; i++) {
  for (int j = 0; j < cols; j++) {
    t[j][i] = m[i][j];
  }
}
// usar t
```

# Invertir cada fila (dos punteros)

```
for (int i = 0; i < m.length; i++) {
  int j1 = 0, j2 = m[i].length - 1;
  while (j1 < j2) {
    int tmp = m[i][j1];
    m[i][j1] = m[i][j2];
    m[i][j2] = tmp;
    j1++; j2--;
  }
}
// m queda con cada fila invertida
```

# Casos borde y errores frecuentes

- **Matriz vacía**: `m.length == 0`.
- **Filas de longitudes distintas**: usar `m[i].length` al recorrer.
- **Índices fuera de rango**: verificar $0 \leq$ `i` $<$ `m.length` y $0 \leq$ `j` $<$ `m[i].length`.
- **Confusiones típicas**: `m.length` (filas) $\neq$ `m[0].length` (columnas).

# Ejemplo A: notas por estudiante y evaluación

- Filas = estudiantes, columnas = evaluaciones.
- Calcular:
  - Promedio por estudiante (por fila).
  - Promedio por evaluación (por columna).
  - Índice del estudiante con mejor promedio.
- Patrón: doble `for`; acumuladores por fila o por columna.

# Snippet — promedios por fila y columna

```
int[][] notas = {
  {60, 70, 80},
  {50, 90, 60},
  {75, 65, 85} };
int F = notas.length;
int C = (F == 0) ? 0 : notas[0].length;
for (int i = 0; i < F; i++) {// promedio por estudiante (fila)
  int suma = 0;
  for (int j = 0; j < C; j++) suma += notas[i][j];
  double prom = (C == 0) ? 0.0 : (suma * 1.0) / C;
  // usar prom
}
for (int j = 0; j < C; j++) {// promedio por evaluacion (columna)
  int suma = 0;
  for (int i = 0; i < F; i++) suma += notas[i][j];
  double prom = (F == 0) ? 0.0 : (suma * 1.0) / F;
  // usar prom
}
```

# Ejemplo B: mapa de enteros (conteos y diagonales)

- Contar cuántas celdas cumplen $\geq k$.
- Suma total y por fila.
- Si es cuadrada: sumar diagonales.

# Snippet — conteo por condición

```
int[][] mapa = {
  {0, 2, 1, 0},
  {3, 1, 2, 2},
  {0, 0, 4, 1}
};
int k = 2;
int cnt = 0;

for (int i = 0; i < mapa.length; i++) {
  for (int j = 0; j < mapa[i].length; j++) {
    if (mapa[i][j] >= k) cnt++;
  }
}
// usar cnt
```

# Resumen de la sesión

- Matriz como **arreglo de filas**: `m[i][j]`, `m.length`, `m[i].length`.
- Recorridos: por filas y por columnas.
- Agregaciones: total, por fila y por columna.
- Diagonales en cuadradas: principal y secundaria.
- Búsqueda 2D y transformaciones básicas (transpuesta, inversión por fila).
- Casos borde: matriz vacía, longitudes distintas, rangos válidos.

# Códigos de la sesión

- Ejemplos usados: matrices (declaración, recorridos, agregaciones, diagonales, búsqueda y transformaciones).
- Abre el enlace o escanea el QR:

`https://gist.github.com/helderfernandez/0510e93caa7801a40eb42351aa05059b`