

Every Box, In Order

2D array traversals



Read the whole table aloud

PAUSE AND TRY

Read every number in this table to your partner. What order did you use?

	col 0	col 1	col 2
row 0	3	1	4
row 1	2	0	5

Rows are teams. Columns are games. Now read just the scores from game 2. What changed in how you read?

By the end of class you can...

- 1 Visit every element in row-major order with nested for loops.
- 2 Visit every element in column-major order, still writing `grid[row][col]`.
- 3 Write loops for other orders: backward, one row, one column.
- 4 Traverse with nested enhanced for loops and choose each variable's type.
- 5 Explain why assigning to an enhanced for loop variable leaves the array unchanged.

A large, white, rounded square containing a large, bold, teal number '1'.

PART 1

Row-major order

Two loops, one inside the other

ROW-MAJOR PATTERN

```
for (int r = 0; r < grid.length; r++) {  
    for (int c = 0; c < grid[0].length; c++) {  
        // use grid[r][c]  
    }  
}
```

VISIT ORDER, 2-BY-3 GRID

	col 0	col 1	col 2
row 0	1st	2nd	3rd
row 1	4th	5th	6th

Piece	Job
r < grid.length	every row index
c < grid[0].length	every column index
grid[r][c]	row first, always



Row-major: a whole row is visited before any of the next row.

Printing each team's scores

SCORES.JAVA

```
int[][] games = { {3, 1, 4},  
                  {2, 0, 5} };  
for (int r = 0; r < games.length; r++) {  
    System.out.print("Team " + r + " :");  
    for (int c = 0; c < games[0].length; c++) {  
        System.out.print(" " + games[r][c]);  
    }  
    System.out.println();  
}
```

OUTPUT

```
Team 0: 3 1 4  
Team 1: 2 0 5
```



Code in the outer body, outside the inner loop, runs **once per row**.



Code in the inner body runs **once per element: $2 * 3 = 6$ times**.

Each loop gets its own length



The variable in the **first** bracket loops to **grid.length**. The one in the **second** loops to **grid[0].length**.

MIXUP.JAVA

```
int[][] games = { {3, 1, 4},  
                  {2, 0, 5} };  
for (int r = 0; r < games[0].length; r++) {  
    for (int c = 0; c < games.length; c++) {  
        System.out.print(games[r][c] + " ");  
    }  
    System.out.println();  
}
```

OUTPUT

```
3 1  
2 0  
Exception in thread "main"  
java.lang.  
ArrayIndexOutOfBoundsException:  
Index 2 out of bounds for  
length 2
```

Stars and visits

PAUSE AND TRY

What does this print? Where is each statement, and how often does it run?

```
int[][] m = new int[3][4];
int visits = 0;
for (int r = 0; r < m.length; r++) {
    System.out.print("*");
    for (int c = 0; c < m[0].length; c++) {
        visits++;
    }
}
System.out.println(" " + visits);
```

Hint: new int[3][4] has 3 rows and 4 columns.

2

PART 2

Column-major order

Swap the loops, not the brackets

BYGAME.JAVA

```
int[][] games = { {3, 1, 4},  
                  {2, 0, 5} };  
for (int c = 0; c < games[0].length; c++) {  
    System.out.print("Game " + c + " :");  
    for (int r = 0; r < games.length; r++) {  
        System.out.print(" " + games[r][c]);  
    }  
    System.out.println();  
}
```

OUTPUT

```
Game 0: 3 2  
Game 1: 1 0  
Game 2: 4 5
```

VISIT ORDER, 2-BY-3 GRID

	col 0	col 1	col 2
row 0	1st	3rd	5th
row 1	2nd	4th	6th



Column-major: a whole column is visited before the next column.

Swapping the indexes too



The order of the loops changes. The order inside the brackets **never** does: row first.

MIXUP2.JAVA

```
int[][] games = { {3, 1, 4},  
                  {2, 0, 5} };  
for (int c = 0; c < games[0].length; c++) {  
    for (int r = 0; r < games.length; r++) {  
        System.out.print(games[c][r] + " ");  
    }  
    System.out.println();  
}
```

OUTPUT

```
3 1  
2 0  
Exception in thread "main"  
java.lang.  
ArrayIndexOutOfBoundsException:  
Index 2 out of bounds for  
length 2
```

A large, white, rounded square containing the number '3' in a bold, teal, sans-serif font.

3

PART 3

Other orders and parts

Starting from the bottom-right corner

BACKWARD.JAVA

```
int[][] games = { {3, 1, 4},  
                  {2, 0, 5} };  
for (int r = games.length - 1; r >= 0; r--) {  
    for (int c = games[0].length - 1; c >= 0; c--) {  
        System.out.print(games[r][c] + " ");  
    }  
}
```

OUTPUT

5 0 2 4 1 3



Count down from **length - 1** while **>= 0**.



Starting at **games.length** asks for a row that does not exist.

One row or one column: one loop

SLICES.JAVA

```
int[][] games = { {3, 1, 4},  
                  {2, 0, 5} };  
for (int c = 0; c < games[0].length; c++) {  
    System.out.print(games[1][c] + " ");  
}  
System.out.println();  
for (int r = 0; r < games.length; r++) {  
    System.out.print(games[r][2] + " ");  
}
```

OUTPUT

```
2 0 5  
4 5
```

	col 0	col 1	col 2
row 0	3	1	4
row 1	2	0	5



Hold one index still. Loop the other, bounded by **its own** length.

One index, both brackets

PAUSE AND TRY

Which boxes does this loop visit? What does it print?

```
String[][] sq = { {"m", "x", "x"},  
                  {"x", "a", "x"},  
                  {"x", "x", "p"} };  
for (int k = 0; k < sq.length; k++) {  
    System.out.print(sq[k][k]);  
}
```

	col 0	col 1	col 2
row 0	m	x	x
row 1	x	a	x
row 2	x	x	p

A large, bold, teal-colored number '4' is centered within a white square with rounded corners. The square is slightly tilted to the right.

4

PART 4

Nested enhanced for loops

The outer variable is a whole row

MENU.JAVA

```
String[][] menu = { {"pho", "taco"},  
                    {"soup", "pie"} };  
for (String[] day : menu) {  
    for (String dish : day) {  
        System.out.print(dish + " ");  
    }  
    System.out.println("/");  
}
```

OUTPUT

```
pho taco /  
soup pie /
```

Variable	Type	Holds
menu	String[][]	the grid
day	String[]	one row
dish	String	one element



Each header drops **one pair** of brackets. Always **row-major**.

Changing the copy, not the array

COPY: NO CHANGE

```
int[][] stock = { {4, 0},  
                  {7, 2} };  
for (int[] row : stock) {  
    for (int n : row) {  
        n = 10;  
    }  
}  
System.out.println(stock[0][0]);
```

OUTPUT

4

THROUGH THE ROW: CHANGES

```
int[][] stock = { {4, 0},  
                  {7, 2} };  
for (int[] row : stock) {  
    for (int i = 0; i < row.length; i++) {  
        row[i] = 10;  
    }  
}  
System.out.println(stock[0][0]);
```

OUTPUT

10

Assigning to the **inner** enhanced for variable never changes the array. Assign through an index.

Which loop for which job?

Job	Use
Read every element, row-major	nested enhanced for, or indexed
Change elements	indexed: <code>grid[r][c] = ...</code>
Column-major or backward	indexed, with the loops you choose
One row or one column	one indexed loop
Need the row or column number	indexed

An enhanced for loop hands you **values**, not positions.

Words to know

Traversal

Visiting every element (or a chosen set) in some order.

Row-major

Across each row, then down to the next row.

Column-major

Down each column, then right to the next column.

Nested loop

A loop inside another; the inner one finishes first.

Nested enhanced for

Outer variable is a row; inner variable is an element.

Loop variable copy

Assigning to it does not change the array.

Across or down?



Explain to your partner how to turn a row-major traversal into a column-major one, and what stays the same.

SENTENCE STARTER

To go column-major, I swap the _____, but the brackets stay `grid[____][____]`.

Next: the Every Box, In Order puzzle worksheet.