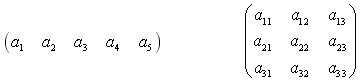
**Two Dimensions Are Better than One**

You have learned arrays can be one-dimensional. Guess what? They can also be two-dimensional. A two-dimensional array is useful when you want to maintain a table of data.

You learned about arrays in Algebra II, where they were referred to as matrices. Do the following matrices look familiar?



The subscript of each matrix element (e.g., a1, a23, etc.) identifies its relative position based on the intersection of a row and column. Arrays use index values to indicate the location of specific elements, based on their row and column.

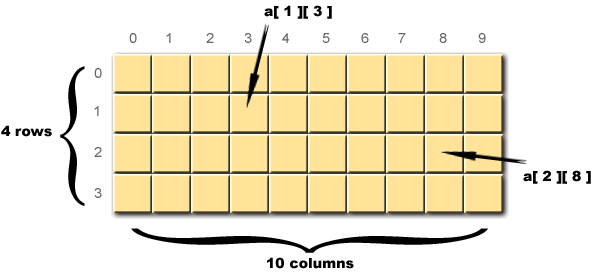
Relax, though. You will not need to do any matrix math; arrays are simply used as convenient data storage containers. If you have ever been to a sporting event or concert with ticketed seating identified by row and seat numbers, you sat in a two-dimensional array. Once again, arrays are everywhere!

### Part 1

The use of two-dimensional arrays is helpful when creating and manipulating tables data to use in calculations and to manipulate.

All the arrays we have met so far — for example, an array a of ints declared by int[] a — are *one-dimensional*. This terminology is used because it only requires one index to reference any element of a.

However, it is possible for an array to be *two-dimensional*. Such an array (also known as a *matrix*) is shown in the figure below. In this case, two indices are required in order to reference an element of the array.



The array pictured in this figure has four *rows* and 10 *columns*. The expression a[ 2 ][ 8 ] refers to the element in the row with index 2 and the column with index 8, that is, the element in the third row, ninth column.

When declaring a two-dimensional array, we must include *two* pairs of brackets, one for each index. The following statement, for example, declares a two-dimensional array a of doubles:

    double[][] a;

We can create such an array with 5 rows and 10 columns and assign it to a as follows:

    a = new double[ 5 ][ 10 ];

Alternatively, we can combine the declaration and assignment into a single line of code like this:

    double[][] a = new double[ 5 ][ 10 ];

As in the case of one-dimensional arrays, there is a shorthand method for declaring and initializing a two-dimensional array while at the same time initializing all its elements. This is as simple as specifying the contents of each row, as in this example:

    int[][] t = { { 1, 2, 3, 4 },    
                  { 5, 6, 7, 8 },    
                  { 9, 10, 11, 12 } };   
  
  
    System.out.println(  );

In discursive text, it is common to use the sequence of characters formed by the data type keyword followed by two pairs of brackets to refer to the data type of a two-dimensional array. Rather than reporting that t has been declared as a matrix of ints, for example, we could equally well say that t has been declared to have data type int[][].

If a is a two-dimensional array, the expression a.length evaluates to the number of rows the array has, and the expression a[ 0 ].length evaluates to the number of columns.

The alternative method of placing the declarative brackets immediately after the variable rather than following the data type reference is also acceptable for two-dimensional arrays, as in these examples:

double a[][];  
double m[][] = new double[ 5 ][ 10 ];  
int t[][] = { { 1, 2, 3, 4 },   
              { 5, 6, 7, 8 },   
              { 9, 10, 11, 12 } };

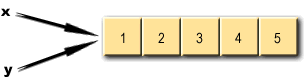
In the following code, x is initialized as a one-dimensional array of type int. Then the array x is assigned to the variable y.

    int[] x = { 1, 2, 3, 4, 5 };   
    int[] y;   
  
    y = x;

y[ 1 ] = 12;

System.out.println( x[ 1 ] );

Your experimentation in Exercise 49 should have made you suspect the following: When an array referenced by one variable is assigned to another variable, then both variables reference the *same* array. In the code above, once the assignment y = x has occurred, the situation may be depicted as follows:

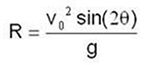


Such a suspicion is well-founded. Under such circumstances, any changes made to the elements of the array are "seen by both variables". In fact, x and y are said to be *aliases* — they are two different labels for one and the same thing. (We will have much more to say about aliases [later](https://www.eimacs.com/eimacs/mainpage?epid=E2183417918&cid=162149) in this course. They can cause all kinds of trouble unless you are very careful!)

### Part 2

Anyone who has thrown a ball intuitively understands the concept of a projectile's trajectory. Depending on the speed of the ball as it leaves the player's hand, the angle at which it is thrown, and certain complicating factors (e.g., wind speed, drag coefficient, etc.), the ball will be under thrown, on target, or overthrown. The trajectory of a rock hurled across a battlefield at a castle wall will follow a similar path. It is physics and calculus in action, so projectile motion can be modeled with a computer.

The equation for the range (R) of a projectile, where v0 is the launch speed, θ is the angle of launch, and g is the gravitational acceleration constant, is given as follows:



This formula eliminates the guesswork when computing the distance an object can be hurled down range, if its initial velocity and launch angle are known. Regardless of the object, this formula applies. The trajectories of baseballs, satellites, missiles, etc., are all subject to this simple algebraic equation. Even objects hurled with the ancient catapult follow the laws of physics.

These are exactly the kinds of calculations that had to be done by hand or with early calculators prior to World War II. Extensive tables of the different combinations of angle and speed took hours, sometimes days to prepare. Your computer, with the right program, could prepare projectile tables in a matter of seconds.