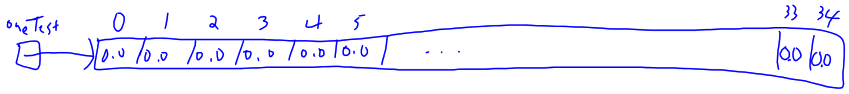
7.3 Multi-dimensional Arrays

Suppose we needed to store student results on tests. A single test could be declared by:

```
float[] oneTest = new float[35];
```

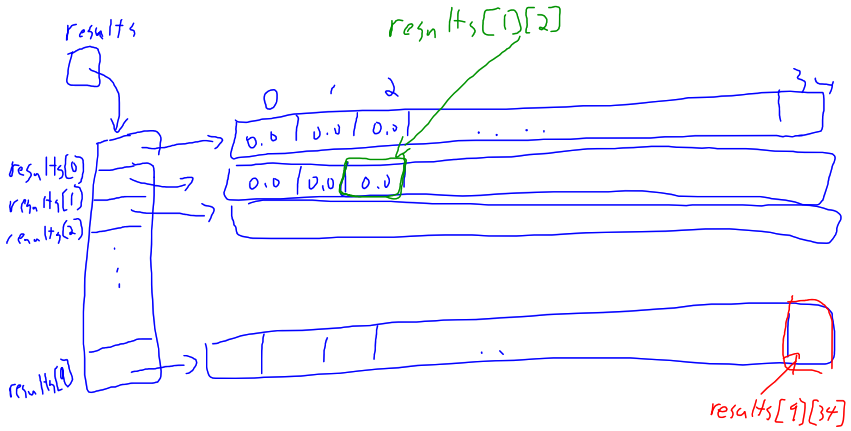


If we need results for the whole year (say 10 tests) we can use a 2-D array:

```
float[][] results = new float[10][35];
```

rows columns

A 2-D array can also be thought of as an array of arrays. To do a whole year we need an array of those "oneTest"s



`results[0]` is a reference to a 1-D array (the first row of the 2-D array).

Any individual element can be referred to by specifying a row and a column. This array will have 350 elements (10 x 35). As before each element will be initialized to 0.0 when the array is created.

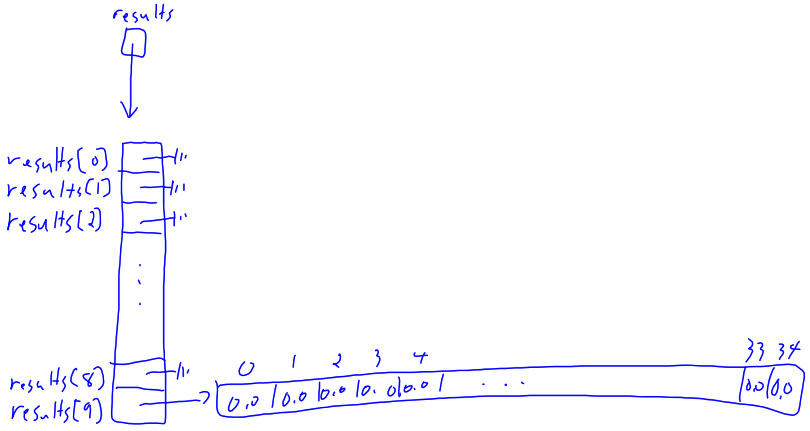
We could also create the array in steps. We would need to do that if each row didn't have the same number of columns (a 2-D array does not necessarily need to be rectangular as in other languages like Turing). Let's look at how we could have created the results array in steps (although we don't need to since it is rectangular):

```
float[][] results - new float[10][];
```

Each reference to a float array is initialized to `null`. Then we could create the last (10th) row:

```
results[9] = new float[35]; // the row can be thought of as a 1-D array
```

The array would look like this:



To create the whole array like the original we could do this:

```
for (int i = 0; i < results.length; i++)
   results[i] = new float[35];
```

We didn't need to make all the rows have 35 elements when we do it this way. We could have made each row have a different number.

To add up all the values in the `results` array:

```
float sum = 0;
for (int row = 0; row < results.length; row++)
   for (int col = 0; col < results[0].length; col++)
      sum += results[row][col];
```

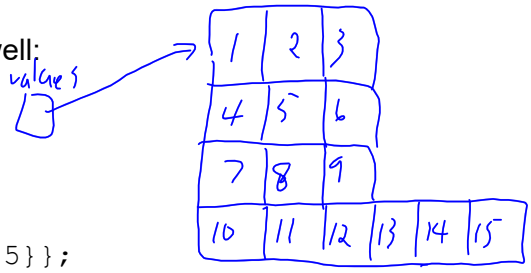
`results.length` gives the number of rows (see diagram on previous page)

`results[0].length` gives the number of columns in the first row. If `results` wasn't a rectangular array, then in the for loop above we would have said

`col < results[row].length` instead of `col < results[0].length` since all the rows wouldn't have had the same number of columns

We can initialize a 2-D array when it is declared as well:

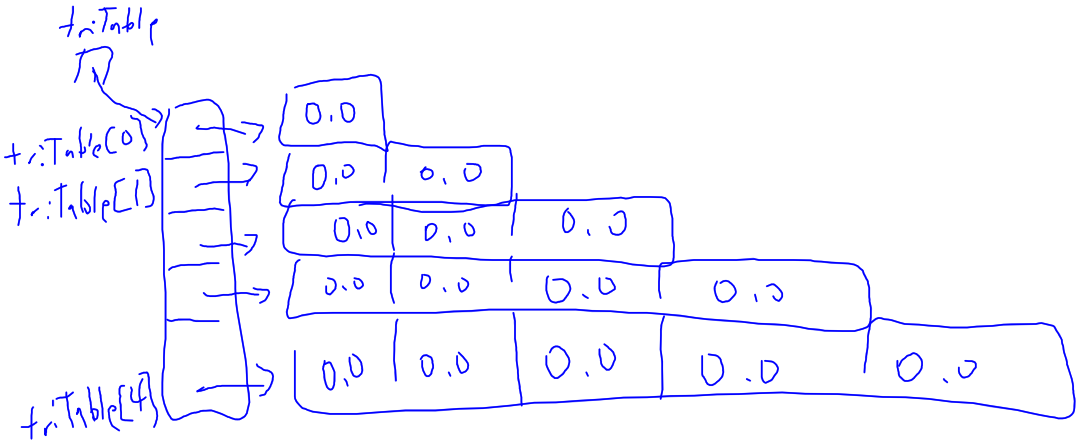
```
int[][]values = {{1, 2, 3},
                 {4, 5, 6},
                 {7, 8, 9},
                 {10, 11, 12, 13, 14, 15}};
```



You don't have to indent it like this, but it makes it easier to keep track of the rows and columns. This is an example of an array that is not rectangular.

Example 3 from the text: Another array that is not rectangular

```
float[][] triTable = new float[5][];
for (int row = 0; row < triTable.length; row++)
   triTable[row] =  new float[row + 1];
```



If you don't specify all dimensions when you create an array you must folllow certain rules. The only place you can have blank dimensions are at the end, not the beginning or the middle. Also you can have as many dimensions as you want (3-D, 4-D, 5-D...) although you rarely see more that 3-D and even that is not nearly as common as 1-D and 2-D. Here are some declarations, with some of them being illegal:

```
float [][][] a = new float[1][][];
int[][][] b = new int[4][][6];
String[][][] c = new String[][4][6];
boolean [][][]d = new boolean[10][8][];
char[][][]e = new char[][][9];
```

illegal, can't have blank dimension in middle

illegal, can't have blank dimension in front

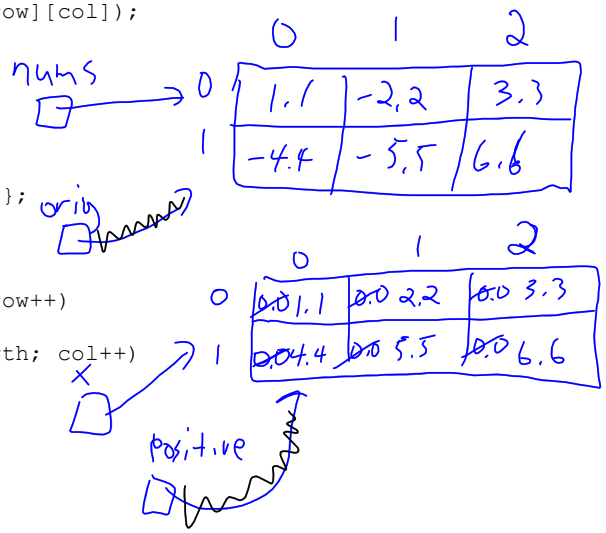
illegal,

You can make methods that return multi-dimensional arrays and that take them as parameters. Here is an example of a method that takes a 2-D array as a parameter, and replaces any negative values with the absolute value of that element. It also shows how to call the method and how you can print a 2-D array in a way that shows its structure.

```
float[][] makePositive(float[][] orig)
{
  float[][] positive = new float[orig.length][orig[0].length];
  // this is assuming it is a rectangular array
  // positive will be the same size as orig
  for (int row = 0; row < orig.length; row++)
    for (int col = 0; col < orig[0].length; col++)
      positive[row][col] = abs(orig[row][col]);
  return positive;
}

void setup()
{
  float[][] nums = {{1.1, -2.2, 3.3},
                    {-4.4, -5.5, 6.6}};
  float[][] x;
  x = makePositive(nums);

  for (int row = 0; row < x.length; row++)
  {
    for (int col = 0; col < x[0].length; col++)
      print(x[row][col] + " ");
    println();
  }
}
```



The output would be:

```
1.1 2.2 3.3

4.4 5.5 6.6
```

Read section 7.3 in the textbook. Do exercise 7.3 #1-8 (I'll post solutions on Wednesday)

Look for 7.4 on Wednesday.