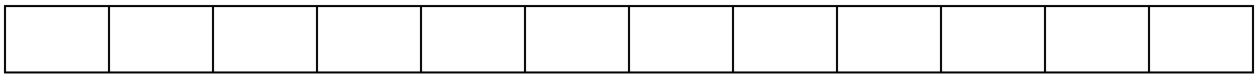
# April 13th Workshop Summary

Data Structures Part 2

## 1D Array Review:

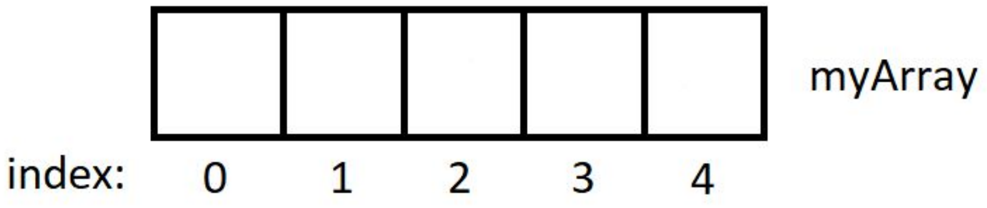


- You can think of 1D array as a sequence of slots saved in your computer memory
- Every slot has a specific index, and you can get the data you want based on this index

**Example:**
Let's start with creating an array of length 5, filled with integers, named myArray.
We can write the following:

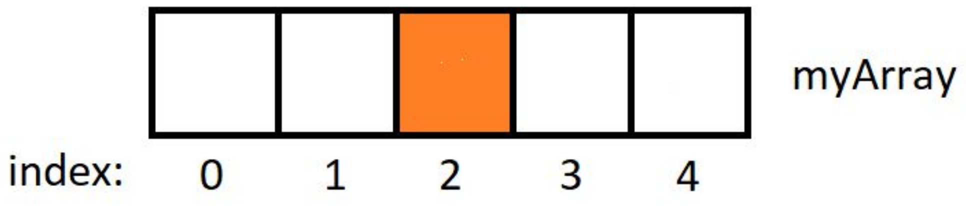
```
int myArray[] = new int[5];
```

Then we would create an array looking like:



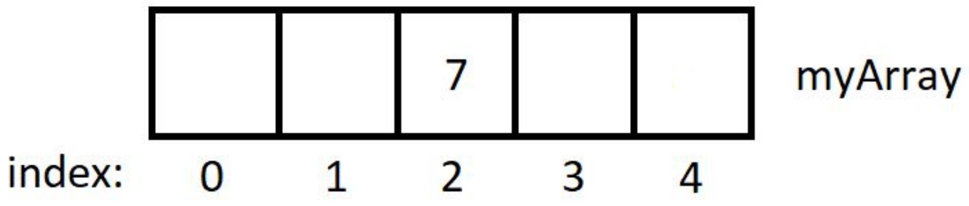
This array has a length of 5, with indices from 0 to 4.

To access a specific slot in the array, we can use the index of this slot.
For example, if we want to access the 3rd slot in myArray, we say: `myArray[2];` This will give us this slot:



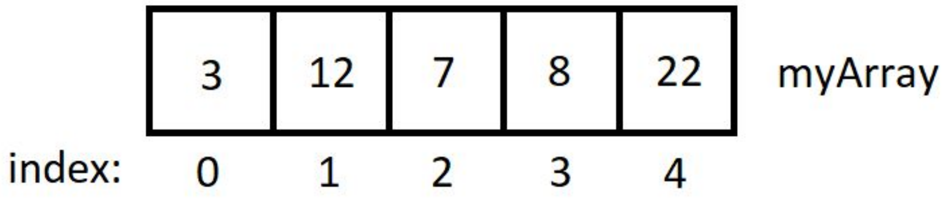
Now we can try to fill the array with numbers;
Let's fill the orange slot with the number 7, we can say:
`myArray[2] = 7;`

Then we'll get an array looking like this:



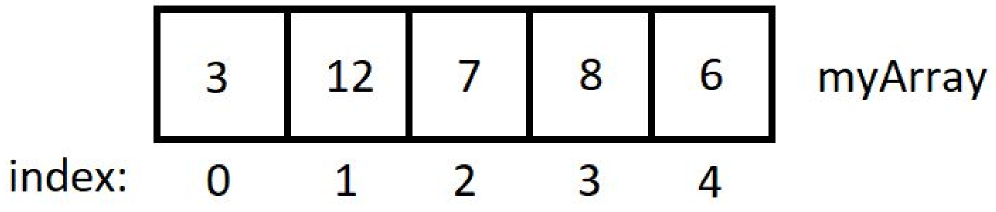
Let's proceed to fill the array with more numbers:
```
myArray[0] = 3;
myArray[1] = 12;
myArray[3] = 8;
myArray[4] = 22;
```

And we'll end up with the following array



**In conclusion, to access a certain slot in the array, we need just need to say:**
**<name-of-array> [ <slot index> ];**

Let's change the 5th slot of the array to the number 6! We can write:
`myArray[4] = 6;`
Then our array would look like:



## Reviewing for loops:

What for loops do is that it can repeatedly do something, some number of times, so you don't have to type all that code yourself.

For example, if I want to access the first, second, third… to 100th slot of the array, and if I don't use for loop, then I would have to type:

```
myLongArray[0];
myLongArray[1];
myLongArray[2];
myLongArray[3];
…
myLongArray[99];
```

But we see that we are doing the EXACT same thing; the only difference is we are changing the index of the array by 1 every time!

So here's how for loops work:

**Example of a for loop:**

```
for(int i = 0; i < 100; i++){
        // do something
}
```

This chunk above, is doing the following things:

1. Creating an integer i, and assign i to 0;
   ```
   for(int i = 0; i < 100; i++){
         // do something
   }
   ```
2. Checking if i is smaller than 100
   ```
   for(int i = 0; i < 100; i++){
         // do something
   }
   ```
3. If i is smaller than 100, we <do something>; otherwise, we skip the whole thing
   ```
   for(int i = 0; i < 100; i++){
         // do something
   }
   ```
4. After we <do something>, we increment i by 1; now i = 1;
   ```
   for(int i = 0; i < 100; i++){
         // do something
   ```

```
        }
```

5. Go back to step 2, check if i (which right now equal to 1), is smaller than 100
```
        for(int i = 0; i < 100; i++){
                // do something
        }
```
6. We <do something>
```
        for(int i = 0; i < 100; i++){
                // do something
        }
```
7. We increment i by 1; now i =2.
```
        for(int i = 0; i < 100; i++){
                // do something
        }
        …
        …
        …
```
Now let's pretend our i has been incremented to 100. We then check if i is smaller than 100. But 100 is NOT smaller than 100, therefore we stop executing the loop!

- We noticed that when we are going through our for loop, we are increasing i by 1 every single time. So our 'i' would equal: 0, 1, 2, 3, 4, 5, …, 99.

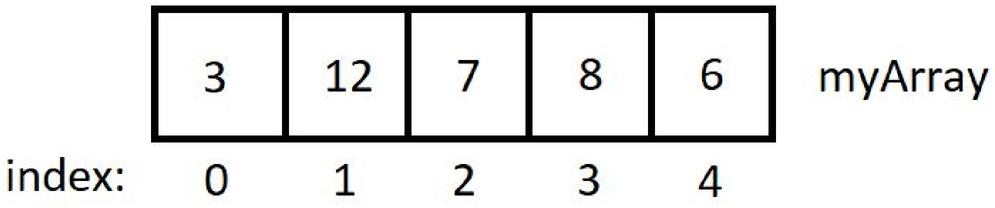
- We also notice that when we are trying to go through our array, we are also increasing our index by 1 every single time. We go from index: 0, 1, 2, 3, 4, 5, …, 99.

- We can access the array by saying: myLongArray[<index>]

- Therefore if we make our index = i, we can go through our entire array since i changes from 0, 1, … , 99.

## Example problem #1:

Print out every single element in myArray using a for loop.
Here's a picture of myArray:



To do this, we need to know a few things:

1. How to print out something?
2. How do you access an element in the array?
3. How can you go to every single element in the array?
4. How can we use a for loop?

To print out a statement, we say:

```
System.out.println(<stuff to print out>);
```

To access an element in the array, we say:

```
myArray[<slot index>];
```

To go to every single element in the array, we say:

```
myArray[0], myArray[1], myArray[2], myArray[3], myArray[4];
```

How can we use a for loop?
Since in a for loop, i changes from 0 to 1 to 2 to … a certain number you assign it to, we can just say:

```
myArray[i];
```

Since i = 0, and then 1, and then 2, and then 3, and then 4

**Solution:**

```
for(int i = 0; i < 5; i++){
      System.out.println(myArray[i]);
}
```

This will print out:

```
3
12
7
8
6
```

## Example problem #2:

Find the sum of myArray using a for loop.

To find the sum, we got to add the first element, the second element, the third element, the forth element and the fifth element of the array together.

In math, we can say:
sum = 1st element + 2nd element + 3rd element + 4th element + 5th element;

We know how to access the 1st element of the array, which is: myArray[0];
Similarly, 2nd element of the array would be myArray[1] …

So we can also say:
sum = myArray[0] + myArray[1] + myArray[2] + myArray[3] + myArray[4]

We see that again, our indices are changing by 1. It goes from 0, 1, 2, … , 4. So we are thinking, can we use a for loop??????????

We know that to find the sum of all 5 elements, we can say it is:
myArray[4] + sum of the previous 4 elements.

We know that to find the sum of the first 4 elements, we can say it is:
myArray[3] + sum of the previous 3 elements.

We know that to find the sum of the previous 3 elements, we can say it is:
myArray[2] + sum of the previous 2 elements.

We know that to find the sum of the previous 2 elements, we can say it is:
myArray[1] + sum of the previous element

The sum of the previous element, is really just the previous element itself + 0:
myArray[0] + 0

We know that for loops are used when we need to repeatedly do something and look, we are repeatedly doing something! What are we repeatedly doing?

We are repeatedly saying:
sum = current element + sum (i.e. sum of all 5 elements = myArray[4] + sum of 4 elements)

And our current element is changing; it goes from myArray[0] to myArray[1] to myArray[2]... and knowing from previous experience, we can replace all of that as myArray[i], and put myArray[i] into a for loop!

So we can say something like:
sum = myArray[i] + sum; and shove this into a for loop.

But one last thing! The computer doesn't know what 'sum' is. We need to somehow store the result of the sum into a variable. Before we started adding anything, our sum = 0;

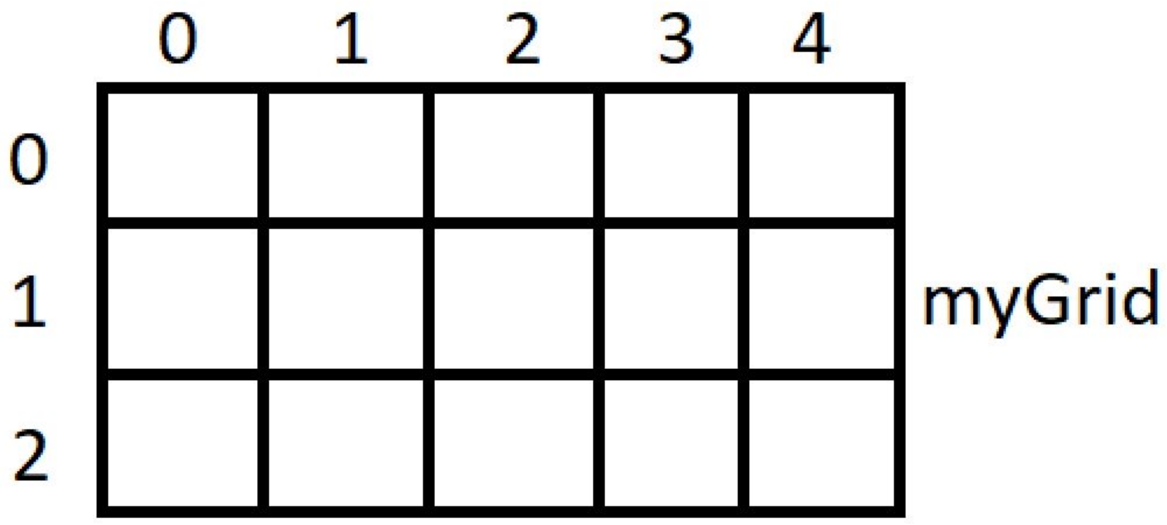
**Solution:**

```
int sum = 0;

for(int i = 0; i < 5; i++){
        sum = myArray[i] + sum;
}

System.out.println(sum);
```

**NOTE:** we should NOT write “int sum = 0” inside our loop, because if that’s the case, everytime the loop is run, sum would become 0 again!

## 2D Arrays:

2D arrays are like a grid (think of battleship!).
The following is a 3 by 5 array that is also indexed!



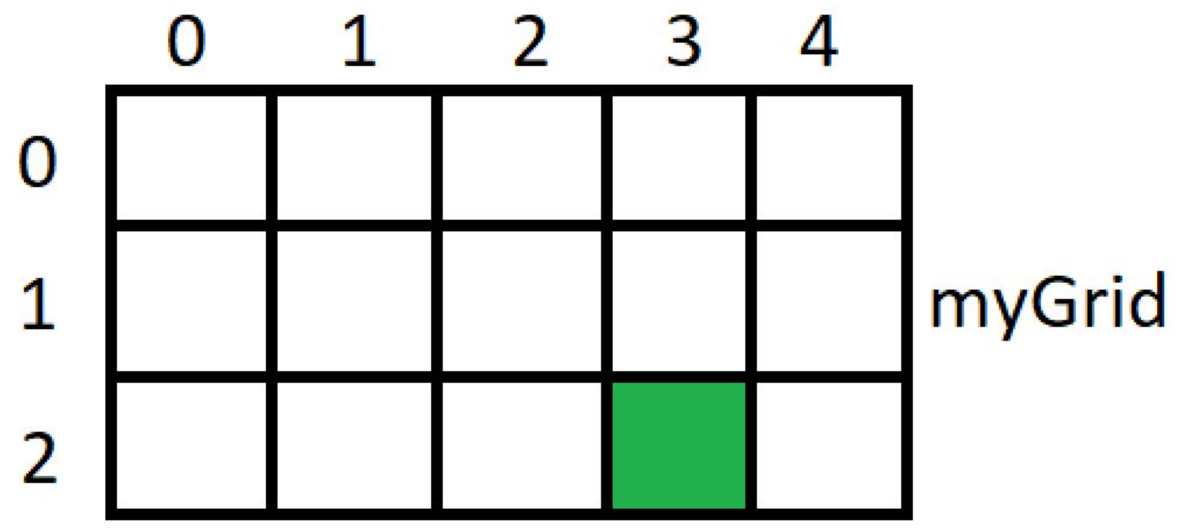
2D arrays are just like 1D arrays, except we need to account for another dimension, so we need to think of the grid as ROWs and COLUMNs.

Recall that to create a 1D array like myArray, we say: int myArray[] = new int[5];

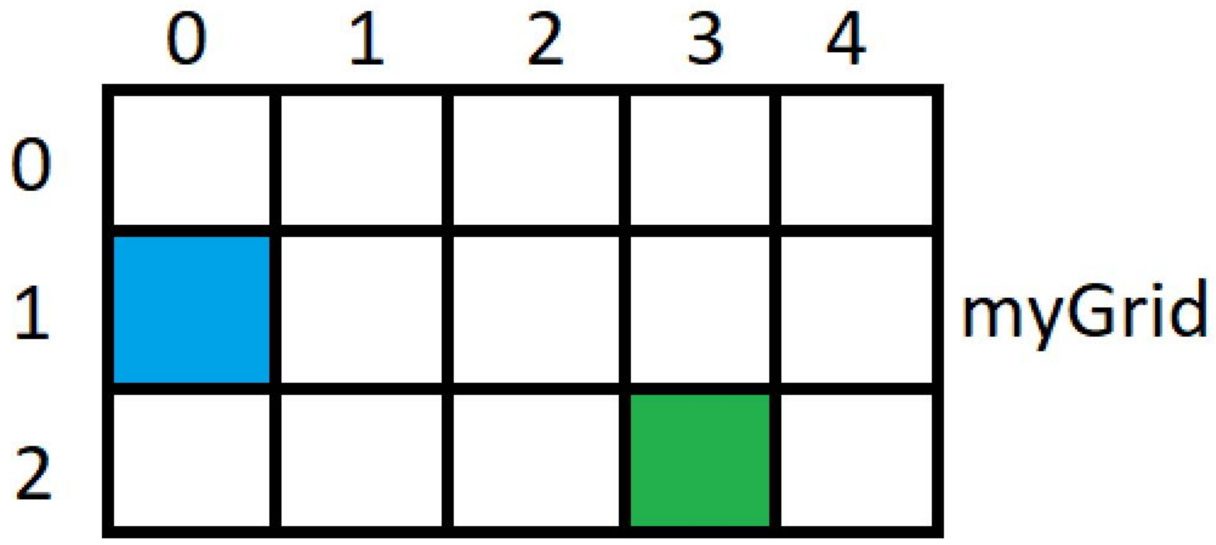
To create a 2D array, of integers, like the one shown above, we can tell the computer:

```
int myGrid[][] = new int[3][5];
```

This would create a grid with 3 rows and 5 columns. NOTE that we also write ROWs first, and then COLUMNS.
In order to access a specific slot in myGrid, we can also tell the computer which row AND THEN which column the slot is in.



To access the green cell like the above, we can say:
`myGrid[2][3];`

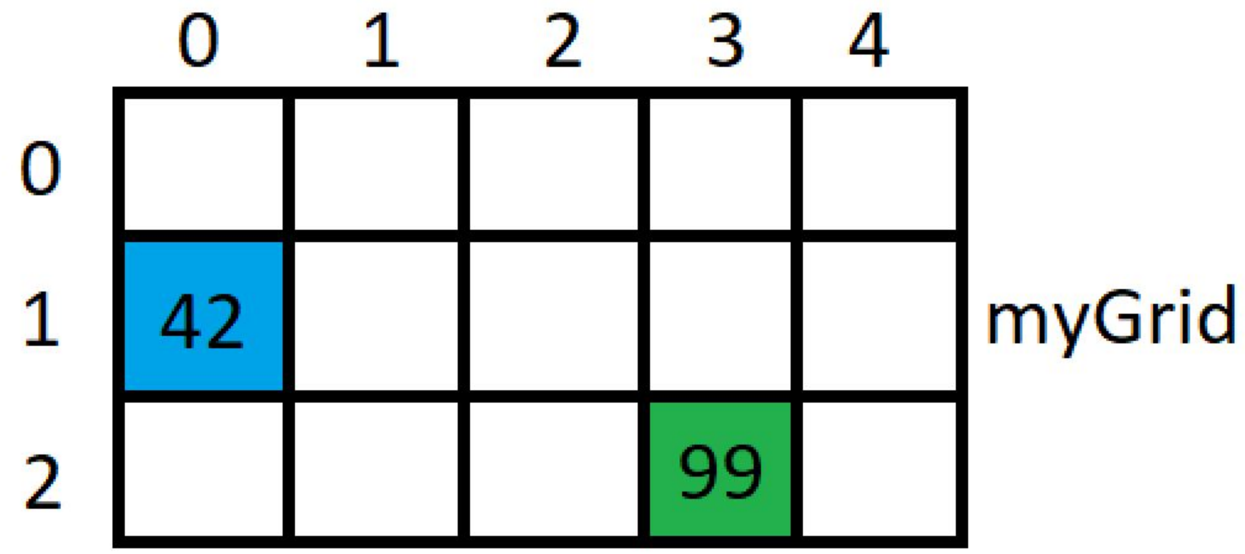


To access the blue cell like the above, we can say:
`myGrid[1][0];`

Let’s assign our blue cell the number 42, and our green cell the number 99, we can say:

```
myGrid[1][0] = 42;
myGrid[2][3] = 99;
```

Then our grid would look like:

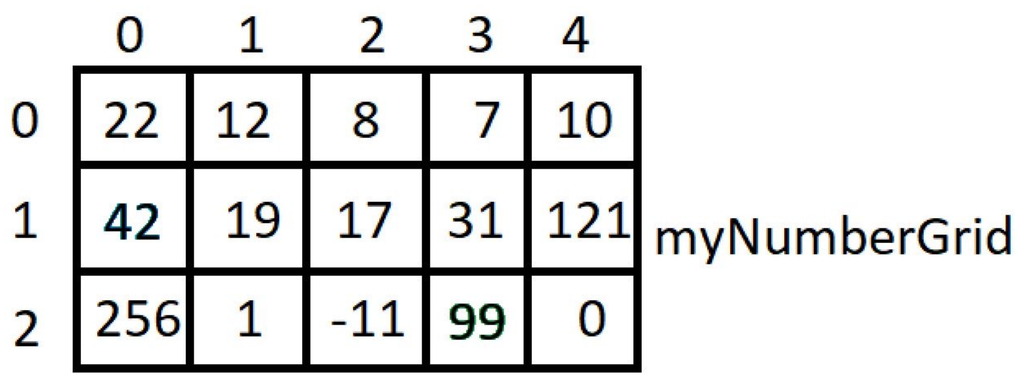


| | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| 0 | | | | | |
| 1 | 42 | | | | |
| 2 | | | | 99 | |

**NOTE: Before you proceed with the following section, be sure that you have read the review of for loops section!**

### Example problem #1:

Given the following grid, myNumberGrid, print out all of the numbers from this grid, from left to right, top to bottom. (i.e. print out: 22, 12, 8, 7, 10, 42, 19 …)



| | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| 0 | 22 | 12 | 8 | 7 | 10 |
| 1 | 42 | 19 | 17 | 31 | 121 |
| 2 | 256 | 1 | -11 | 99 | 0 |

myNumberGrid

We know that we can access every single grid by its row and column.
What we can do is we can print out:

```
myNumberGrid[0][0]; myNumberGrid[0][1]; myNumberGrid[0][2];
myNumberGrid[0][3]; myNumberGrid[0][4];

myNumberGrid[1][0]; myNumberGrid[1][1]; myNumberGrid[1][2];
myNumberGrid[1][3]; myNumberGrid[1][4];

myNumberGrid[2][0]; myNumberGrid[2][1]; myNumberGrid[2][2];
myNumberGrid[2][3]; myNumberGrid[2][4];
```

But if we have a ton of these grids, we cannot just type all of them out.
We also noticed that hey, in every single section, our column index change by 1;
And in every section, our row index change by 1 as well!

Let's take first section in red, we see that the indeces are:
[0][0], [0][1], [0][2], [0][3], [0][4].

Also recall from the for-loop review section, we know that the integer "i" would always increment by 1, it goes from 0 to 1 to 2 to 3 to…

Therefore, just printing this one row out, we can say: (just to spice things up a little, let's use the letter 'j' instead of 'i', it's the same thing!)

```
for (int j = 0; j < 5; j++){
      System.out.print(myNumberGrid[0][j]);
}
```

If we want to print all three rows out, we can say:

```
for (int j = 0; j < 5; j++) {
      System.out.print(myNumberGrid[0][j]);
}

System.out.print("\n") // this just prints a new line

for (int j = 0; j < 5; j++) {
      System.out.print(myNumberGrid[1][j]);
}

System.out.print("\n") // this just prints a new line
```

```
for (int j = 0; j < 5; j++){
      System.out.print(myNumberGrid[2][j]);
}

System.out.print("\n") // this just prints a new line
```

BUT WAIT A MINUTE! We are repeating things again! The only thing that changes is our row number that's highlighted in blue.
But if we look at the row number, we see that it increments from 0, to 1, to 2; Can we shove that into a for loop again?

The answer is yes we can!  Since we are doing the exact same thing (writing those for loops 3 times), let's just shove that thing (the for loops) into another for loop!

**Solution:**

```
For (int i = 0; i < 3; i++) {
      for (int j = 0; j < 5; j++) {
            System.out.print(myNumberGrid[i][j]);
      }
      System.out.print("\n") // this just prints a new line
}
```

This will print out:

```
22128710
42191731121
2561-11990
```

Since we did not put a space between the numbers. But you can always add a space behind the part where you print out the numbers!

## Challenge problem:

Add all of the numbers up in myGrid using nested for loops!

## Additional Problems:

1. Using a for loop, print 100 “*”

2. Create a character array of the same length as the length of your name. Fill this character array with the letters of your name. Using a for loop, print out the letters of this character array.

3. You are creating mines to combat your enemies. Create a character 2D array (a grid) of 10 rows and 12 columns. You have 10 mines, which are represented as “*”. Put these mines in any cell you want.

4. Oopsies! You forgot how many mines you have put in your battlefield. Now you need to go through every single cell in your battlefield to count how many mines you have put. Use a nested for loop to do that!