CMSC 150
INTRODUCTION TO COMPUTING

ACKNOWLEDGEMENT: THESE SLIDES ARE ADAPTED FROM SLIDES PROVIDED WITH INTRODUCTION TO PROGRAMMING IN JAVA: AN INTERDISCIPLINARY APPROACH, SEDGEWICK AND WAYNE (PEARSON ADDISON-WESLEY 2007)

LECTURE 4

- ARRAYS
- MULTIDIMENSIONAL ARRAYS

MANY VARIABLES OF THE SAME TYPE

- Goal. 10 variables of the same type.

```
// tedious and error-prone
double a0, a1, a2, a3, a4, a5, a6, a7, a8, a9;
a0 = 0.0;
a1 = 0.0;
a2 = 0.0;
a3 = 0.0;
a4 = 0.0;
a5 = 0.0;
a6 = 0.0;
a7 = 0.0;
a8 = 0.0;
a9 = 0.0;
…
a4 = 3.0;
…
a8 = 8.0;
…
double x = a4 + a8;
```

MANY VARIABLES OF THE SAME TYPE

- Goal. 10 variables of the same type.

```
// easy alternative
double[] a = new double[10];
…
a[4] = 3.0;
…
a[8] = 8.0;
…
double x = a[4] + a[8];
```

declares, creates, and initializes
[stay tuned for details]

MANY VARIABLES OF THE SAME TYPE

- Goal. 1 million variables of the same type.

```
// scales to handle large arrays
double[] a = new double[1000000];
…
a[123456] = 3.0;
…
a[987654] = 8.0;
…
double x = a[123456] + a[987654];
```

ARRAYS

- Array. Indexed sequence of values of the same type
- Store and manipulate huge quantities of data.
- Examples.
    - 52 playing cards in a deck
    - 3 thousand undergrads at UR
    - 140 characters per Tweet
    - 4 billion nucleotides in a DNA strand
    - 50 trillion cells in the human body
    - $6.022x10^{23}$ particles in a mole

Index	Value
0	Captain America
1	Ironman
2	Thor
3	Hulk
4	Black Widow
5	Hawkeye
6	Nick Fury
7	Wannabe Avengerman

ARRAYS IN JAVA

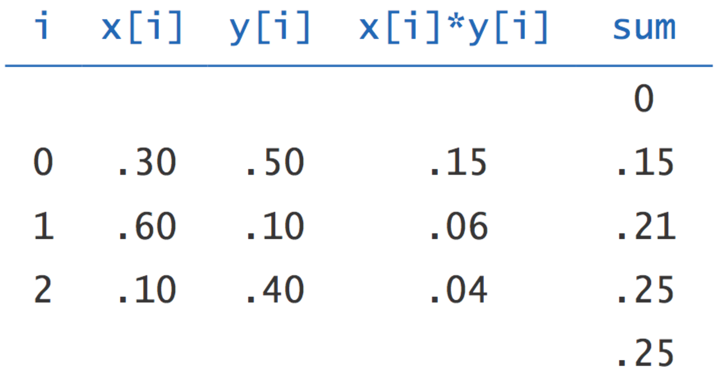
- Java has special language support for arrays.
  - To make an array: declare, create, and initialize it
  - To access entry i of array named a, use $a[i]$
  - Array indices start at 0

```
int N = 10;                 // size of array
double[] a;                 // declare the array
a = new double[N];          // create the array
for (int i = 0; i < N; i++)          // initialize the
   a[i] = 0.0;                       // array all to 0.0
OR
double[] a = new double[10]; //declare, create,
// and initialize. Default will initialize to 0 for double.
```

VECTOR DOT PRODUCT

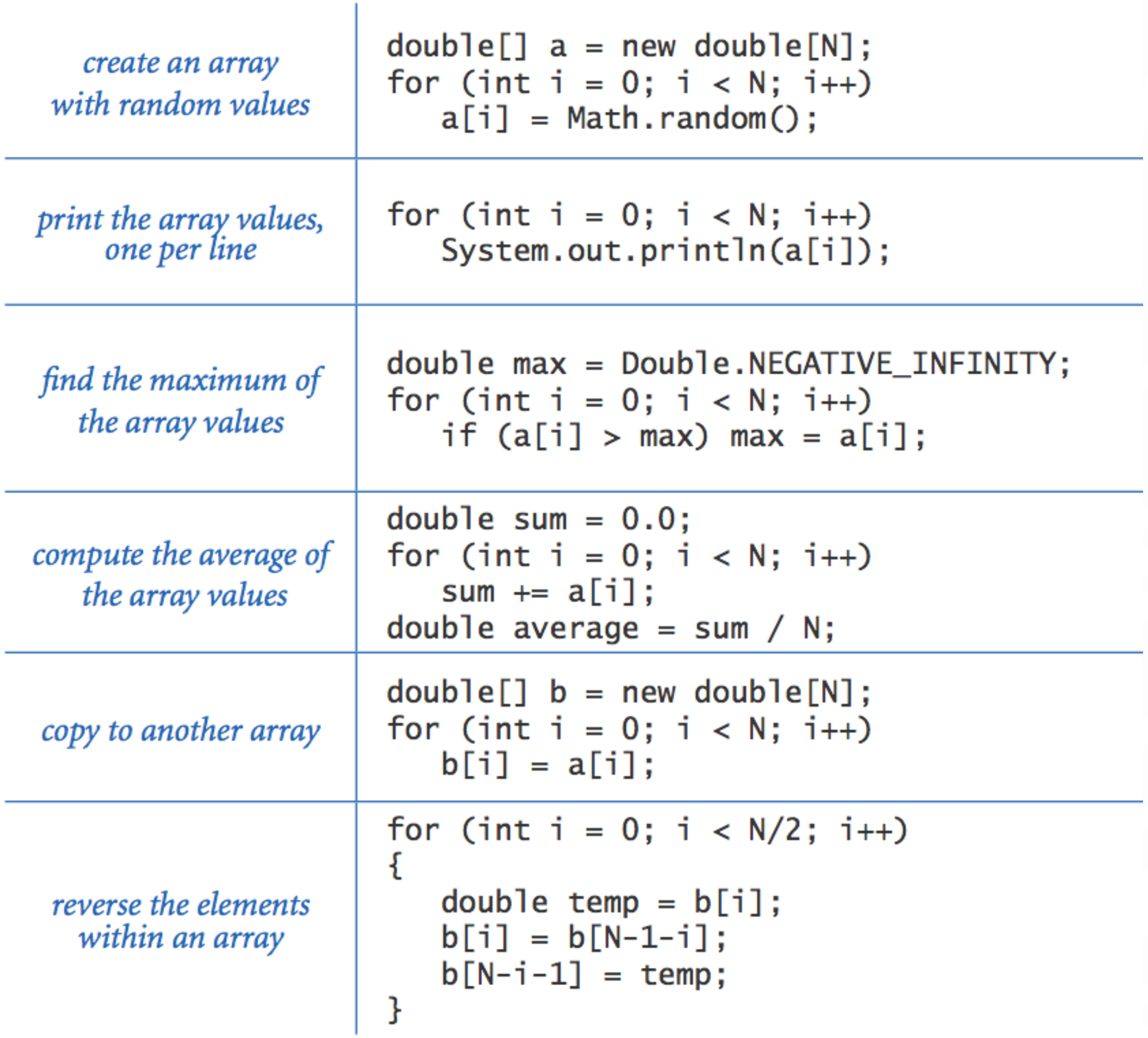
- Dot product. Given two vectors x[] and y[] of length N, their dot product is the sum of the products of their corresponding components.

```
double[] x = { 0.3, 0.6, 0.1 }; //Another way to initialize
double[] y = { 0.5, 0.1, 0.4 };
int N = x.length;
double sum = 0.0;
for (int i = 0; i < N; i++) {
   sum = sum + x[i]*y[i];
}
```



i	x[i]	y[i]	x[i]*y[i]	sum
				0
0	.30	.50	.15	.15
1	.60	.10	.06	.21
2	.10	.40	.04	.25
				.25

ARRAY-PROCESSING EXAMPLES



create an array with random values	`double[] a = new double[N];`
	`for (int i = 0; i < N; i++)`
	`   a[i] = Math.random();`

print the array values, one per line	`for (int i = 0; i < N; i++)`
	`   System.out.println(a[i]);`

find the maximum of the array values	`double max = Double.NEGATIVE_INFINITY;`
	`for (int i = 0; i < N; i++)`
	`   if (a[i] > max) max = a[i];`

compute the average of the array values	`double sum = 0.0;`
	`for (int i = 0; i < N; i++)`
	`   sum += a[i];`
	`double average = sum / N;`

copy to another array	`double[] b = new double[N];`
	`for (int i = 0; i < N; i++)`
	`   b[i] = a[i];`

reverse the elements within an array	`for (int i = 0; i < N/2; i++)`
	`{`
	`   double temp = b[i];`
	`   b[i] = b[N-1-i];`
	`   b[N-i-1] = temp;`
	`}`

EXAMPLES: DECK OF CARDS

SETTING ARRAY VALUES AT COMPILE TIME

- Ex. Print a random card.

```
String[] rank = {
  "2", "3", "4", "5", "6", "7", "8", "9",
  "10", "Jack", "Queen", "King", "Ace"
};

String[] suit = {
  "Clubs", "Diamonds", "Hearts", "Spades"
};

int i = (int) (Math.random() * 13);          // between 0 and 12
int j = (int) (Math.random() * 4);           // between 0 and 3

System.out.println(rank[i] + " of " + suit[j]);
```

SETTING ARRAY VALUES AT RUN TIME

- Ex. Create a deck of playing cards and print them out.

```
String[] deck = new String[52];             //Default initialized to “”
for (int i = 0; i < 13; i++)                //Reassign the values to something meaning full
  for (int j = 0; j < 4; j++)
    deck[4*i + j] = rank[i] + " of " + suit[j];

for (int i = 0; i < 52; i++)
  System.out.println(deck[i]);
```

- Q. In what order does it output them?

- A.

```
two of clubs
two of diamonds
two of hearts
two of spades
three of clubs
...
```

B.

```
two of clubs
three of clubs
four of clubs
five of clubs
six of clubs
...
```

SHUFFLING

- Goal. Given an array, rearrange its elements in random order.
- Shuffling algorithm.
  - In iteration i, pick random card from deck[i] through deck[N-1], with each card equally likely.
  - Exchange it with deck[i].

```java
int N = deck.length;                       //Use .length to know how many elements
for (int i = 0; i < N; i++) {              //there are.
   int r = i + (int) (Math.random() * (N-i)); //Random between i and N − 1
   String t = deck[r];                     //Swap
   deck[r] = deck[i];
   deck[i] = t;
}
```

SHUFFLING A DECK OF CARDS: PUTTING EVERYTHING TOGETHER

```
public class Deck {
  public static void main(String[] args) {
    String[] suit = { "Clubs", "Diamonds", "Hearts", "Spades" };                              //Define suits, ranks, and sizes
    String[] rank = { "2", "3", "4", "5", "6", "7", "8", "9", "10", "Jack", "Queen", "King", "Ace" };
    int SUITS = suit.length, RANKS = rank.length, N = SUITS * RANKS;

    String[] deck = new String[N];                                                            //Build deck
    for (int i = 0; i < RANKS; i++)
      for (int j = 0; j < SUITS; j++)
        deck[SUITS*i + j] = rank[i] + " of " + suit[j];

    for (int i = 0; i < N; i++) {                                                             //Shuffle
      int r = i + (int) (Math.random() * (N-i));
      String t = deck[r];
      deck[r] = deck[i];
      deck[i] = t;
    }

    for (int i = 0; i < N; i++)                                                               //Print the deck
      System.out.println(deck[i]);
  }
}
```

STRINGS REVISITED

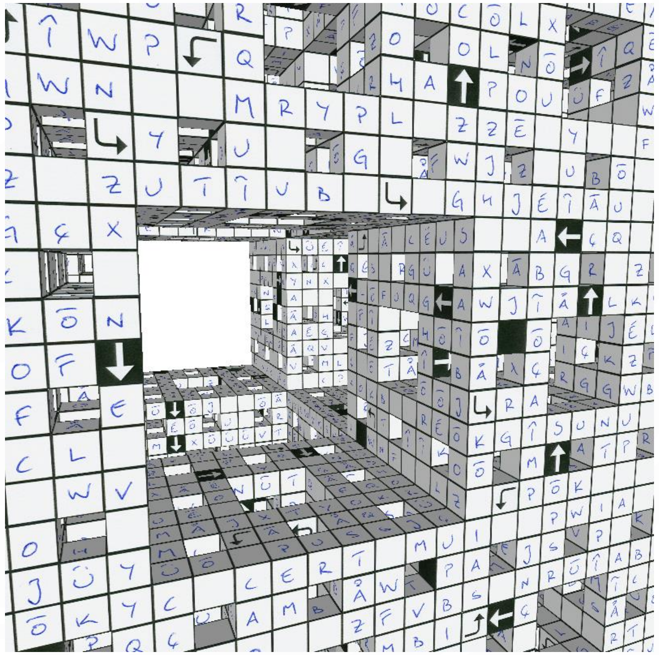
- Strings are arrays of **char**! Well sort of…
- Strings 'underneath the hood' are arrays of **char**, but we use them differently
  - Java API for String
  - Use charAt(i) instead of [i]
  - Convert to **char**[] using toCharArray() and back to a String easily

    ```
    String s = "Hello";
    char[] c = s.toCharArray();
    String s2 = new String(c);
    ```
  - Allows more, e.g., substring() which returns a portion of the String

EXERCISE – PARTNERS

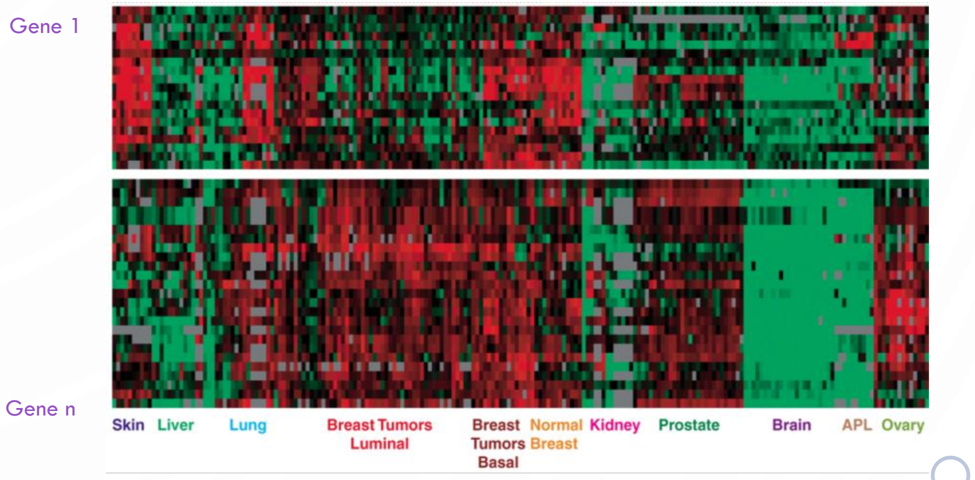
- Write an algorithm/program to find the minimum, maximum, and average of an array of doubles. Use a single loop!

MULTIDIMENSIONAL ARRAYS



TWO-DIMENSIONAL ARRAYS

- Two-dimensional arrays.
  - Table of data for each experiment and outcome.
  - Table of grades for each student and assignments.
  - Pixels in an image
- Mathematical abstraction. Matrix.
- Java abstraction. 2D array.



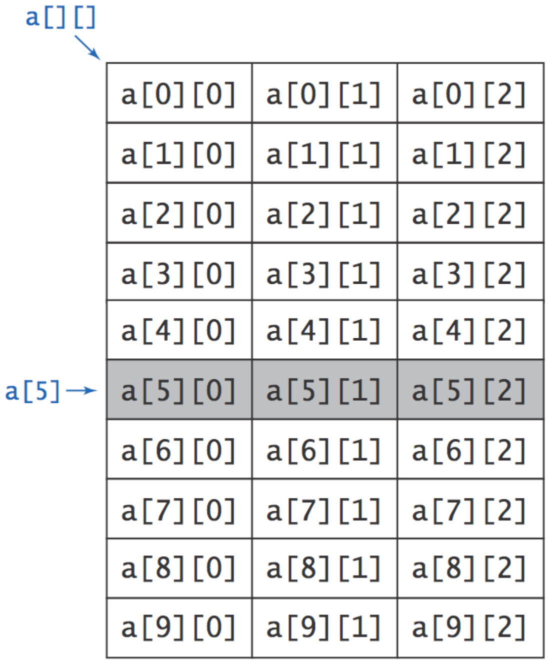
Reference: Botstein & Brown group

🟦 gene expressed
🟩 gene not expressed

TWO-DIMENSIONAL ARRAYS IN JAVA

- Array access. Use a[i][j] to access entry in row i and column j.
- Zero-based indexing. Row and column indices start at 0.

```java
int M = 10, N = 3;
double[][] a = new double[M][N];
for (int i = 0; i < M; i++)
   for (int j = 0; j < N; j++)
      a[i][j] = 0.0;
```



a[][]

a[5]

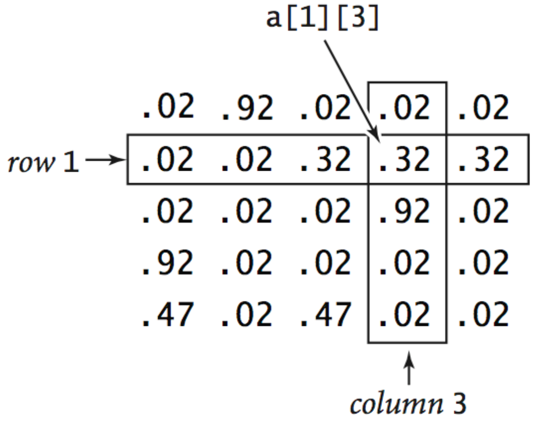
a[0][0]	a[0][1]	a[0][2]
a[1][0]	a[1][1]	a[1][2]
a[2][0]	a[2][1]	a[2][2]
a[3][0]	a[3][1]	a[3][2]
a[4][0]	a[4][1]	a[4][2]
a[5][0]	a[5][1]	a[5][2]
a[6][0]	a[6][1]	a[6][2]
a[7][0]	a[7][1]	a[7][2]
a[8][0]	a[8][1]	a[8][2]
a[9][0]	a[9][1]	a[9][2]

A 10-by-3 array

SETTING 2D ARRAY VALUES AT COMPILE TIME

- Initialize 2D array by listing values.

```
double[][] p = {
    { .02, .92, .02, .02, .02 },
    { .02, .02, .32, .32, .32 },
    { .02, .02, .02, .92, .02 },
    { .92, .02, .02, .02, .02 },
    { .47, .02, .47, .02, .02 },
};
```



a[1][3]

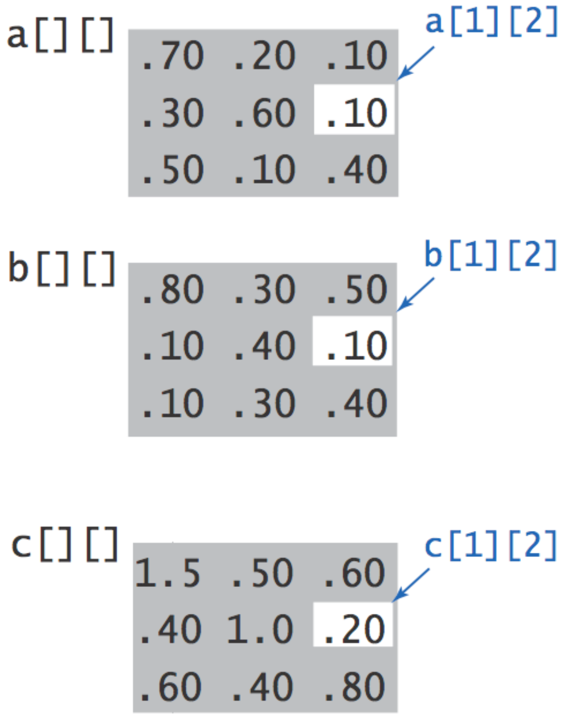
	.02	.92	.02	.02	.02
row 1	.02	.02	.32	.32	.32
	.02	.02	.02	.92	.02
	.92	.02	.02	.02	.02
	.47	.02	.47	.02	.02

column 3

MATRIX ADDITION

- Matrix addition. Given two N-by-N matrices a and b, define c to be the N-by-N matrix where $c[i][j]$ is the sum $a[i][j] + b[i][j]$

```
double[][] c = new double[N][N];
for (int i = 0; i < N; i++)
   for (int j = 0; j < N; j++)
      c[i][j] = a[i][j] + b[i][j];
```

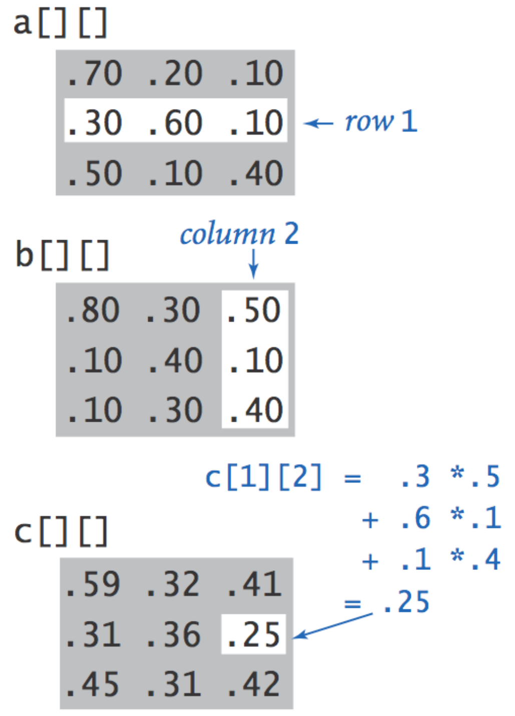


```
a[][]  .70 .20 .10   a[1][2]
       .30 .60 .10
       .50 .10 .40

b[][]  .80 .30 .50   b[1][2]
       .10 .40 .10
       .10 .30 .40

c[][]  1.5 .50 .60   c[1][2]
       .40 1.0 .20
       .60 .40 .80
```

MATRIX MULTIPLICATION

- Matrix multiplication. Given two N-by-N matrices a and b, define c to be the N-by-N matrix where $c[i][j]$ is the dot product of the ith row of $a[\,][\,]$ and the jth column of $b[\,][\,]$.

```
double[][] c = new double[N][N];
for (int i = 0; i < N; i++)
   for (int j = 0; j < N; j++)
      for (int k = 0; k < N; k++)
         c[i][j] += a[i][k] * b[k][j];
```



a[][]

```
.70 .20 .10
.30 .60 .10   ← row 1
.50 .10 .40
```

b[][]

column 2

```
.80 .30 .50
.10 .40 .10
.10 .30 .40
```

c[][]

```
.59 .32 .41
.31 .36 .25
.45 .31 .42
```

```
c[1][2] =  .3 *.5
         + .6 *.1
         + .1 *.4
         = .25
```

ODDITIES OF MULTI-DIMENSIONAL ARRAYS

- A multidimensional array is considered “ragged” if the columns do not have equal lengths

```
int N = 10;
int[][] ragged = new int[N][];
for(int i = 1; i <= N; ++i)
    ragged[i-1] = new int[i];
```

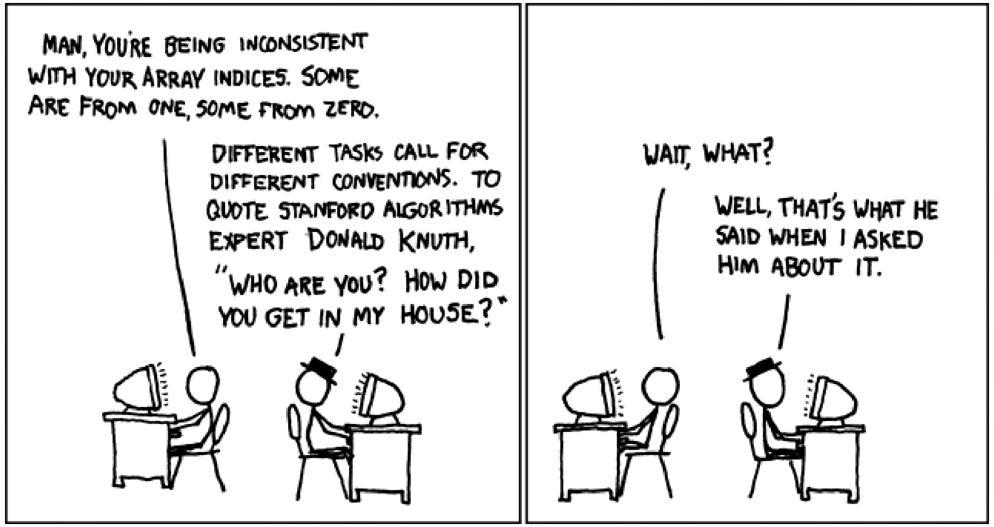
```
0
00
000
0000
00000
000000
0000000
00000000
000000000
0000000000
```

EXERCISE – PARTNERS

- Write an algorithm/program to transpose a Matrix. Transposing means that each row becomes a column in a new matrix.

SUMMARY

- Arrays.
  - Organized way to store huge quantities of data.
  - Almost as easy to use as primitive types.
  - Can directly access an element given its index.


MAN, YOU'RE BEING INCONSISTENT WITH YOUR ARRAY INDICES. SOME ARE FROM ONE, SOME FROM ZERO.
DIFFERENT TASKS CALL FOR DIFFERENT CONVENTIONS. TO QUOTE STANFORD ALGORITHMS EXPERT DONALD KNUTH, "WHO ARE YOU? HOW DID YOU GET IN MY HOUSE?"
WAIT, WHAT?
WELL, THAT'S WHAT HE SAID WHEN I ASKED HIM ABOUT IT.

http://imgs.xkcd.com/comics/donald_knuth.png