Introduction to Computers for Engineers:

Recitation #6

Learning Objectives

- Understand the format of a 2D Matrix (aka 2D Array) in MATLAB
- Understand how to index rows and columns
- Understand how to use nested loops to help you index rows and columns
- Continue to develop an understanding of loops

Question 1: Function called “test”

- function [my_output] = test(my_input)
- [output] = test(input)

(A)

- function (my_output) = test[my_input]
- [output] = test(input)

(B)

- function (my_output) = test[my_input]
- test(input)

(C)

- function [test] = my_output(my_input)
- [output] = test(input)

(D)

Question 2: Looping arrays

- for i=1:length(array)
  - if array > 0
    - disp('hello')

(A)

- for i=1:array
  - if array(i) > 0
    - disp('hello')

(B)

- for i=1:length(array)
  - if array(i) > 0
    - disp('hello')

(C)

- for i=1:array
  - if array > 0
    - disp('hello')

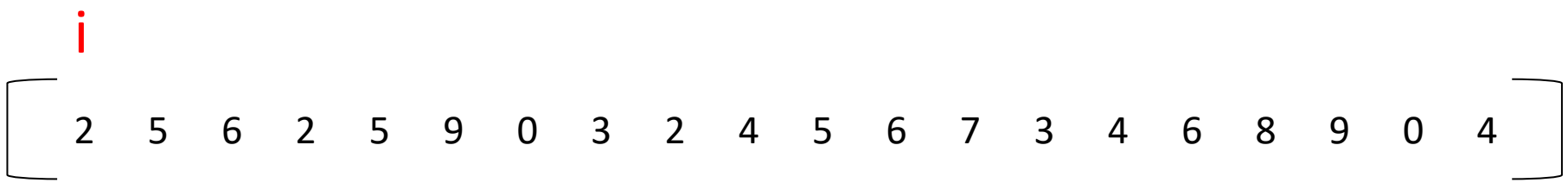
(D)

Activity 1: Matrices in MATLAB

- The objective of this activity is to introduce how to create matrices (2D arrays) in MATLAB
- 1. Choose a basic geometric shape
  - Draw it on the whiteboard
- 2. Think about how you would create that shape with **numerical elements** in a 6 row x 6 column **matrix (2D array)**
  - 1's represent the shape, 0's represent the background
  - Draw this matrix on the whiteboard
- 3. Write code that creates this matrix in MATLAB
- 4. Once the matrix is stored as a **variable**, write the command **imagesc(variable)**, where "variable" is the name of your variable to see a visualization of your shape

Loops and Arrays

```
myArray = theNumbersBelow;
for i = 1:1:length(myArray);
    myArray(i) = myArray(i);
end
```



Loops and Arrays

```
myMatrix = theNumbersBelow;
[rows, cols] = size(myMatrix);
for i = 1:1:rows
    for j = 1:1:cols
        myMatrix(i,j) = myMatrix(i,j);
    end
end
```

j

i

2	5	6	2	5	9	0	3	2	4	5	6	7	3	4	6	8	9	0	4
5	3	7	2	5	9	1	3	2	4	0	6	4	3	3	6	0	9	7	5
1	1	6	9	9	3	0	4	3	5	5	2	7	5	4	9	8	2	0	4

Activity 2: Nested Loops

- Write a function called **matrixSum** with one input variable **my_matrix** and output variable **matrix_sum**
  - my_matrix: matrix (2D array)
  - matrix_sum: double
- We want this function to take the sum of all of the elements in our input matrix.
- Procedure
  1. Write a line of code that determines the number of rows and columns in your matrix
     - [rows, cols] = size(matrix);
  2. Set up nested loops to count through indices of your matrix (i.e. every combination of row and column indices that exists)
  3. **Before your loops** create a variable that will store the sum of every number in your matrix
  4. **Inside of your loops** add value of the element at the current index to the current value of your sum variable
- **Discuss and write test cases and test your function**
  - **Come up with at least 2 different test matrices**

Activity 2: Solution

```
function [matrix_sum] = matrixSum(my_matrix)

matrix_sum = 0;

[rows, columns] = size(my_matrix);

for i=1:rows
    for j=1:columns
        matrix_sum = matrix_sum + my_matrix(i, j);

    end
end



end
```

Activity 3: Loops and Conditional Statements

- The objective of this activity is to get used to putting conditional statements inside of loops
- Write a function called **primeSum** with one input variable **my_matrix** and output variable **prime_sum**
  - my_matrix: matrix (2D array)
  - prime_sum: double
- Instead of taking the sum of every element in your matrix, we only want to take the sum of prime numbers of the matrix
- You can use the built-in function isprime to help you
  - isprime(number) returns false if number is not prime and true if the number is prime
- Discuss and test your code with different test cases.

Activity 3: Solution

```
function [prime_sum] = primeSum(my_matrix)

prime_sum = 0;

[rows, columns] = size(my_matrix);

for i=1:rows
    for j=1:columns

        element = my_matrix(i, j);
        if isprime(element)
           prime_sum = prime_sum + element;
        end

    end
end



end
```