**DAILY ASSESSMENT FORMAT**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date:** | **07/07/2020** | **Name:** | **Varshini MN** |
| **Course:** | **Matlab** | **USN:** | **4AL16EC089** |
| **Topic:** | **Indexing into and modifying arrays** | **Semester & Section:** | **8th B** |
| **Github Repository:** | **varshinimn-test** |  |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **FORENOON SESSION DETAILS** | | | | | |
| **Image of session** | | | | | |
| **REPORT**  **ARRAY INDEXING**  In MATLAB, there are three primary approaches to accessing array elements based on their location (index) in the array. These approaches are indexing by position, linear indexing, and logical indexing. Indexing with Element Positions The most common way is to explicitly specify the indices of the elements. For example, to access a single element of a matrix, specify the row number followed by the column number of the element.  A = [1 2 3 4; 5 6 7 8; 9 10 11 12; 13 14 15 16]  A = 4×4  1 2 3 4  5 6 7 8  9 10 11 12  13 14 15 16  e = A(3,2)  e = 10  e is the element in the 3,2 position (third row, second column) of A.  You can also reference multiple elements at a time by specifying their indices in a vector. For example, access the first and third elements of the second row of A.  r = A(2,[1 3])  r = 1×2  5 7  To access elements in a range of rows or columns, use the [colon](https://in.mathworks.com/help/matlab/ref/colon.html). For example, access the elements in the first through third row and the second through fourth column of A.  r = A(1:3,2:4)  r = 3×3  2 3 4  6 7 8  10 11 12  An alternative way to compute r is to use the keyword end to specify the second column through the last column. This approach lets you specify the last column without knowing exactly how many columns are in A.  r = A(1:3,2:end)  r = 3×3  2 3 4  6 7 8  10 11 12  If you want to access all of the rows or columns, use the colon operator by itself. For example, return the entire third column of A.  r = A(:,3)  r = 4×1  3  7  11  15  In general, you can use indexing to access elements of any array in MATLAB regardless of its data type or dimensions. For example, directly access a column of a [datetime](https://in.mathworks.com/help/matlab/ref/datetime.html) array.  t = [datetime(2018,1:5,1); datetime(2019,1:5,1)]  t = 2x5 datetime  01-Jan-2018 01-Feb-2018 01-Mar-2018 01-Apr-2018 01-May-2018  01-Jan-2019 01-Feb-2019 01-Mar-2019 01-Apr-2019 01-May-2019  march1 = t(:,3)  march1 = 2x1 datetime  01-Mar-2018  01-Mar-2019  For higher-dimensional arrays, expand the syntax to match the array dimensions. Consider a random 3-by-3-by-3 numeric array. Access the element in the second row, third column, and first sheet of the array.  A = rand(3,3,3);  e = A(2,3,1)  e = 0.5469 **INDEXING WITH A SINGLE INDEX** Another method for accessing elements of an array is to use only a single index, regardless of the size or dimensions of the array. This method is known as linear indexing. While MATLAB displays arrays according to their defined sizes and shapes, they are actually stored in memory as a single column of elements. A good way to visualize this concept is with a matrix. While the following array is displayed as a 3-by-3 matrix, MATLAB stores it as a single column made up of the columns of A appended one after the other. The stored vector contains the sequence of elements 12, 45, 33, 36, 29, 25, 91, 48, 11, and can be displayed using a single colon.  A = [12 36 91; 45 29 48; 33 25 11]  A = 3×3  12 36 91  45 29 48  33 25 11  Alinear = A(:)  Alinear = 9×1  12  45  33  36  29  25  91  48  11  While linear indexing can be less intuitive visually, it can be powerful for performing certain computations that are not dependent on the size or shape of the array. For example, you can easily sum all of the elements of A without having to provide a second argument to the sum function.  s = sum(A(:))  s = 330 **INDEXING WITH LOGICAL VALUES** Using true and false logical indicators is another useful way to index into arrays, particularly when working with conditional statements. For example, say you want to know if the elements of a matrix A are less than the corresponding elements of another matrix B. The less-than operator returns a logical array whose elements are 1 when an element in A is smaller than the corresponding element in B.  A = [1 2 6; 4 3 6]  A = 2×3  1 2 6  4 3 6  B = [0 3 7; 3 7 5]  B = 2×3  0 3 7  3 7 5  ind = A<B  ind = 2x3 logical array  0 1 1  0 1 0  MATLAB "is" functions also return logical arrays that indicate which elements of the input meet a certain condition. For example, check which elements of a [string](https://in.mathworks.com/help/matlab/ref/string.html) vector are missing using the [ismissing](https://in.mathworks.com/help/matlab/ref/ismissing.html) function.  str = ["A" "B" missing "D" "E" missing];  ind = ismissing(str)  ind = 1x6 logical array  0 0 1 0 0 1  Suppose you want to find the values of the elements that are not missing. Use the ~ operator with the index vector ind to do this.  strvals = str(~ind)  strvals = 1x4 string  "A" "B" "D" "E" | | | | | |
| **Date:** | **07/07/2020** | **Name:** | **Varshini MN** |
| **Course:** | **IOT** | **USN:** | **4AL16EC089** |
| **Topic:** | **Introduction to Big Data** | **Semester & Section:** | **8th B** |

|  |
| --- |
| **AFTERNOON SESSION DETAILS** |
| **REPORT**  **BIG DATA**  Data is information that comes from a variety of sources, such as people, pictures, text, sensors, and web sites. Data also comes from technology devices like cell phones, computers, kiosks, tablets, and cash registers. Most recently, there has been a spike in the volume of data generated by sensors. Sensors are now installed in an ever growing number of locations and objects. These include security cameras, traffic lights, intelligent cars, thermometers, and even grape vines!  Big Data is a lot of data, but what is a lot? No one has an exact number that says when data from an organization is considered “Big Data.” Here are three characteristics that indicate an organization may be dealing with Big Data:   * They have a large amount of data that increasingly requires more storage space (volume). * They have an amount of data that is growing exponentially fast (velocity). * They have data that is generated in different formats (variety).   How much data do sensors collect? Here are some estimated examples:   * Sensors in one autonomous car can generate 4,000 gigabits (Gb) of data per day. * An Airbus A380 Engine generates 1 petabyte (PB) of data on a flight from London to Singapore. * Safety sensors in mining operations can generate up to 2,4 terabits (TB) of data every minute.   **LARGE DATASETS**   * Companies do not necessarily have to generate their own Big Data. Smaller organizations might not have the sensors, the volume of customers, or the ability to generate the variety of information that could benefit their company * There are sources of free data sets available, ready to be used and analyzed by anyone willing to look for them. * Many companies of various sizes believe they have to collect their own data to see benefits from big data analytics, but it is simply not true.   **WHAT ARE THE CHALLENGES OF BIG DATA?**  IBM’s Big Data estimates conclude that “each day we create 2.5 quintillion bytes of data”. To put this into context, every minute of every day:   * We upload over 300 hours of YouTube video. * We send over 3.5 million text messages. * We stream over 86 thousand hours of Netflix video. * We like over 4 million Facebook posts. * We request over 14 million forecasts from The Weather Channel. * The rapid growth of data can be an advantage or an obstacle when it comes to achieving business goals * To be successful, enterprises must be able to easily access and manage their data assets * With this enormous amount of data being constantly created, traditional technologies and data warehouses cannot keep up with storage needs * Even with the cloud storage facilities that are available from companies like Amazon, Google, Microsoft, and many others, the security of stored data becomes a big problem * Big Data solutions must be secure, have a high fault tolerance, and use replication to ensure data does not get lost * Big Data storage is not only about storing data; it is also about managing and securing it   **FOG COMPUTING**   * Fog computing is an architecture that utilizes end-user clients or “edge” devices to do a substantial amount of the pre-processing and storage required by an organization. * Fog computing was designed to keep the data closer to the source for pre-processing * Sensor data, in particular, can be pre-processed closer to where it was collected. The information gained from that pre-processed analysis can be fed back into the companies’ systems to modify processes if required. Because the sensor data is pre-processed by end devices within the company system, communications to and from the servers and devices would be quicker. This requires less bandwidth than constantly going out to the cloud * After the data has been pre-processed, it is often shipped off for longer term storage, backup, or deeper analysis within the cloud   fog.PNG |