**DAILY ASSESSMENT FORMAT**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date:** | **07/07/2020** | **Name:** | **Nayanashree k s** |
| **Course:** | **Matlab** | **USN:** | **4AL16EC042** |
| **Topic:** | **Indexing into and modifying arrays** | **Semester & Section:** | **8 A** |
| **Github Repository:** | **Nayana\_online** |  |  |

|  |
| --- |
| **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  For more information on working with multidimensional arrays, see [Multidimensional Arrays](https://in.mathworks.com/help/matlab/math/multidimensional-arrays.html). 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  For example, the 3,2 element of A is 25, and you can access it using the syntax A(3,2). You can also access this element using the syntax A(6), since 25 is sixth element of the stored vector sequence.  e = A(3,2)  e = 25  elinear = A(6)  elinear = 25  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  The [sub2ind](https://in.mathworks.com/help/matlab/ref/sub2ind.html) and [ind2sub](https://in.mathworks.com/help/matlab/ref/ind2sub.html) functions help to convert between original array indices and their linear version. For example, compute the linear index of the 3,2 element of A.  linearidx = sub2ind(size(A),3,2)  linearidx = 6  Convert from the linear index back to its row and column form.  [row,col] = ind2sub(size(A),6)  row = 3  col = 2 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  Now that you know the locations of the elements meeting the condition, you can inspect the individual values using ind as the index array. MATLAB matches the locations of the value 1 in ind to the corresponding elements of A and B, and lists their values in a column vector.  Avals = A(ind)  Avals = 3×1  2  3  6  Bvals = B(ind)  Bvals = 3×1  3  7  7  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:** | **7/7/2020** | **Name:** | **Nayanashree K S** | |
| **Course:** | **Cisco** | **USN:** | **4al16ec042** | |
| **Topic:** | **Everything becomes programmable**  **Everything generates data** | **Semester & Section:** | **8 A** | |
| **AFTERNOON SESSION DETAILS** | | | |
| **REPORT** **System Software, Application Software, and Computer Languages** There are two common types of computer software: system software and application software.  Application software programs are created to accomplish a certain task or collection of tasks. For example, Cisco Packet Tracer is a network simulation program that allows users to model complex networks and ask “what if” questions about network behavior.  System software works between the computer hardware and the application program. It is the system software that controls the computer hardware and allows the application programs to function. Common examples of system software include Linux, Apple OSX, and Microsoft Windows.  Both system software and application software are created using a programming language. A programming language is a formal language designed to create programs that communicate instructions to computer hardware. These programs implement algorithms which are self-contained, step-by-step sets of operations to be performed.  Some computer languages compile their programs into a set of machine-language instructions. C++ is an example of a compiled computer language. Others interpret these instructions directly without first compiling them into machine language. Python is an example of an interpreted programming language. An example of Python code is shown in the figure.  When the programming language is determined and the process is diagrammed in a flowchart, program creation can begin. Most computer languages use similar program structures. **Programming Variables** Programming languages utilize variables as dynamic buckets to hold phrases, numbers, or other important information that can be used in coding. Instead of repeating specific values in numerous places throughout the code, a variable can be used. Variables can hold the result of a calculation, the result of a database query, or some other value. This means that the same code will function using different pieces of data without having to be rewritten.  For instance “x + y = z” is an example of a programming expression. In this expression, x, y and z are variables which can represent characters, character strings, numeric values or memory addresses.  A variable can refer to a value. For instance the expression “a = 10” associates the value 10 to variable a.  A variable can also represent a memory location. The expression “a = 10” represents that the value 10 is stored in some location of the computer memory, which is referred to as ‘a’.  Variables can be classified into two categories:   * **Local Variables** - These are variables that are within the scope of a program / function / procedure. * **Global Variables** - These are variables that are in the scope for the time of the program’s execution. They can be retrieved by any part of the program.   Variables allow programmers to quickly create a wide range of simple or complex programs which tell the computer to behave in a pre-defined fashion.  The image contains icons of percentage signs, addition signs, numbers, email envelopes and gear. **What is Blockly?** Blockly is a visual programming tool created to help beginners understand the concepts of programming. By using a number of block types, Blockly allows a user to create a program without entering any lines of code. This is shown in Figure 1.  Blockly implements visual programming by assigning different programming structures to colored blocks. The blocks also contain slots and spaces to allow programmers to enter values required by the structure. Programmers can connect programming structures together by dragging and attaching the appropriate blocks. Programming structures such as conditionals, loops, and variables are all available for use.  Creating a new variable in Blockly is a simple matter of dragging the variable block onto the work space and filling in the value slot. It is also possible to change the contents of a variable as the program is being executed.  Figure 2 shows a Blockly variable.  Blockly also supports functions. Similar to the variables, Blockly has specific blocks to represent functions. Also similar to variables, programmers simply select and drag function blocks to the work space and fill in the required slots.  Notice in Figures 1 and 2 that the variable block and the print on screen block both have a bevel tab on the bottom and a slot on the top. This means that the two blocks can be snapped together to create a program sequence. Blockly will execute the block on the top first, then move on to the block below it.  Other blocks are available such as an IF THEN block, a WHILE block and a FOR block. There are also blocks specifically for sensors and actuators.  Blockly can be used to translate the block-based code into Python or JavaScript. This is very useful to beginner programmers. | | | |
|  | | | |