Week 3 MATLAB Activities

Contents

Time required: 180 minutes

How Does Typing in a Code Tutorial Help with Learning?

Typing in a code tutorial can significantly enhance learning in several ways:

- **Active Engagement**: Typing the code yourself forces you to actively engage with the material, rather than passively reading or watching. This active participation helps reinforce the concepts being taught.
- **Muscle Memory**: Repeatedly typing code helps build muscle memory, making it easier to recall syntax and structure when you write code independently.

- **Error Handling**: When you type code, you're likely to make mistakes. Debugging these errors helps you understand common pitfalls and how to resolve them, which is a crucial skill for any programmer.
- **Understanding**: Typing out code allows you to see how different parts of the code interact. This deeper understanding can help you apply similar concepts to different problems.
- **Retention**: Studies have shown that actively doing something helps with retention. By typing out the code, you're more likely to remember the concepts and techniques.

Reading

Matlab A Practical Introduction to Programming and Problem Solving (Stormy Attaway)

Sections 2.1, 2.2

MATLAB Assignment Script

1. Create a MATLAB script named **Wk03Lastname.m**
2. Save all programs in this script.
3. Include your name and date at the top of the script file as comments.
4. Put a Section Break between each program.

Tutorial 1: Relational Operators

Relational operators are used to compare two values or expressions. The result of a relational operation is a logical value: true (1) or false (0).

The relational operators in MATLAB are:

`x == y`	x is equal to y
`x ~= y`	x is not equal to y
`x > y`	x is greater than y
`x < y`	x is less than y
`x >= y`	x is greater than or equal to y

`x <= y`	x is less than or equal to y

```
%% Relational Operator examples
clc
% Define variables
a = 5;
b = 10;

% Equal to
result = (a == b); % result is 0 (false)
disp(result);

% Not equal to
result = (a ~= b); % result is 1 (true)
disp(result);

% Greater than
result = (a > b); % result is 0 (false)
disp(result);

% Less than
result = (a < b); % result is 1 (true)
disp(result);

% Greater than or equal to
result = (a >= b); % result is 0 (false)
disp(result);

% Less than or equal to
result = (a <= b); % result is 1 (true)
disp(result);
```

Relational operators with vectors compares each element of vector1 with the corresponding element of vector2 and return a logical array.

```
%% Vector relational operators
clc
% Define two vectors
vector1 = [3, 6, 9, 12, 15];
vector2 = [5, 6, 7, 8, 9];

% Equal to
result_equal = (vector1 == vector2);
disp('Equal to (==):');
disp(result_equal);

% Not equal to
result_not_equal = (vector1 ~= vector2);
disp('Not equal to (~=):');
disp(result_not_equal);

% Greater than
result_greater = (vector1 > vector2);
disp('Greater than (>):');
disp(result_greater);

% Less than
result_less = (vector1 < vector2);
disp('Less than (<):');
disp(result_less);

% Greater than or equal to
result_greater_equal = (vector1 >= vector2);
disp('Greater than or equal to (>=):');
disp(result_greater_equal);

% Less than or equal to
result_less_equal = (vector1 <= vector2);
disp('Less than or equal to (<=):');
disp(result_less_equal);
```

Example run:

```
Equal to (==):
   0   1   0   0   0

Not equal to (~=):
   1   0   1   1   1

Greater than (>):
   0   0   1   1   1

Less than (<):
   1   0   0   0   0

Greater than or equal to (>=):
   0   1   1   1   1

Less than or equal to (<=):
   1   1   0   0   0
```

You can combine relational operators with logical operators (&, |, ~) to form complex conditions.

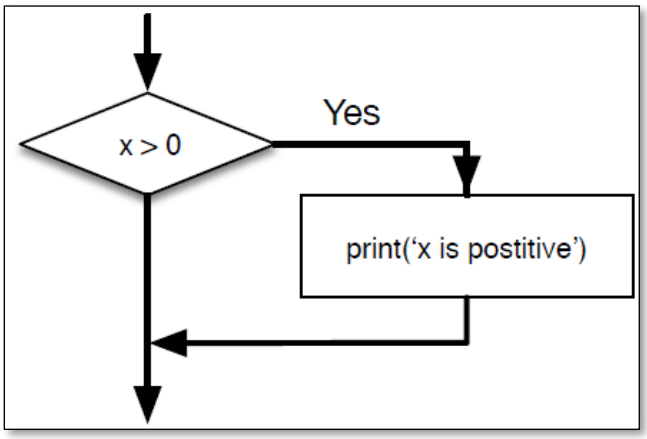
```
%% Logical Operators
clc
% Define variables
x = 15;
y = 25;

% Combine relational and logical operators
result = (x > 10) & (y < 30); % result is 1 (true)
disp(result);
result = (x < 10) | (y > 20); % result is 1 (true)
disp(result);
result = ~(x == 15); % result is 0 (false)
disp(result);
```

Tutorial 2: Decision Statements

Decision structures are essential for controlling the flow of your MATLAB programs based on conditions. By using if, else, elseif, and switch, you can create flexible and powerful scripts that respond to different inputs and scenarios.

The if Statement

The if statement executes a block of code if a specified condition is true.



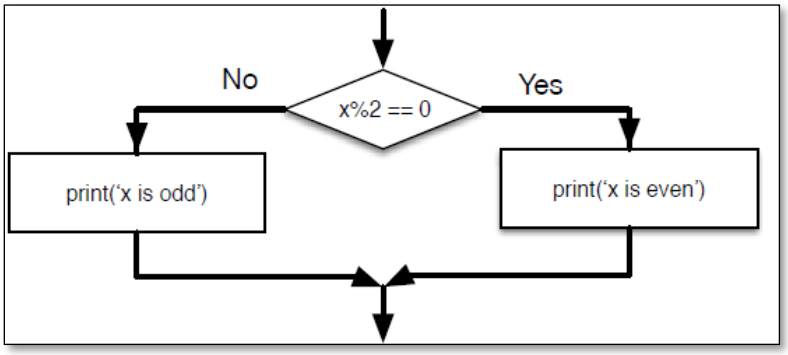

```
% Example of if statement
x = 10;
if x > 5
    disp('x is greater than 5');
end
```

Example run:

```
x is greater than 5
```

2. The else Statement

The else statement executes a block of code if the condition in the if statement is false.



This program uses the mod function to determine is a number is even or odd. If the remainder is 0, the number is even. Otherwise, it is odd.

```
% Example of if-else statement
x = input("Enter a number: ");
% mod function returns the remainder from integer division
if mod(x,  2) == 0
    fprintf('%.0f is even\n', x);
else
    fprintf('%.0f is odd\n', x);
end
```

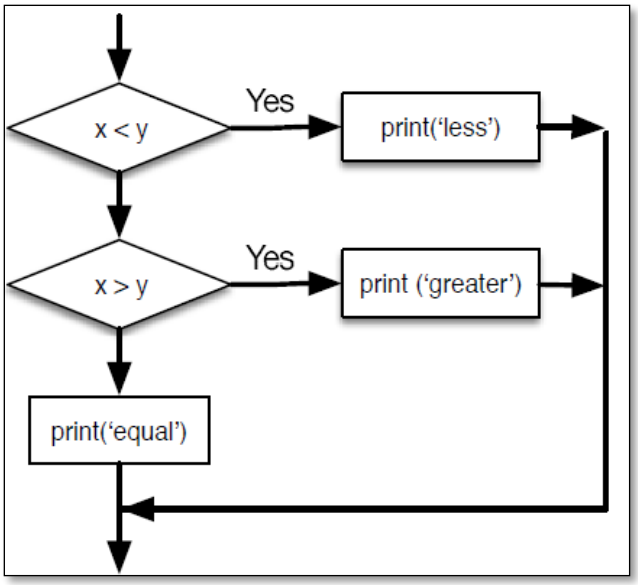
Example run:

```
Enter a number: 26
26 is even
```

```
Enter a number: 12587
12587 is odd
```

The elseif Statement

The elseif statement allows you to check multiple conditions.



Notice the program starts with the highest number decision and works its way down.

```
% Example of if-elseif-else statement
x = 7;
if x > 10
    disp('x is greater than 10');
elseif x > 5
    disp('x is greater than 5 but less than or equal to 10');
else
    disp('x is 5 or less');
end
```

Example run:

```
x is greater than 5 but less than or equal to 10
```

Assignment 1: Grade Calculator

Write a MATLAB program that takes a student's score as input and outputs their grade based on the following criteria:

- A: 90-100
- B: 80-89
- C: 70-79
- D: 60-69
- F: Below 60

Hint: Start by determining greater than 90, and work your way down.

Example run:

```
Enter your score: 102
Your grade is: A
```

The : (Colon) Operator

The colon operator in MATLAB is a versatile tool primarily used for creating regularly spaced vectors, defining ranges, and accessing specific elements within arrays. It's represented by the colon symbol (:).

Create Regularly Spaced Vectors

To create a vector with evenly spaced values, you can use the colon operator in the following way:

```
vec = start_value : step : end_value;
```

```
>> vec = 5:1:9;
>> disp(vec)
     5     6     7     8     9
```

Access Specific Elements

```
# Retrieve the value at the specified index in the vector.
vec = [10, 20, 30, 40, 50];
element = vec(index);
```

```
>> vec = [10 20 30 40 50]
>> element = vec(3);
>> disp(element)
    30
```

Slicing

Use the colon operator to retrieve elements with a specified step within the specified range.

```
# Retrieve the values with the specified step
vec = [10, 20, 30, 40, 50];
subvec = vec(startIndex:step:endIndex);
```

```
>> vec = [10, 20, 30, 40, 50];
subvec = vec(1:2);
disp(subvec)
    10    20
```

The **end** keyword references the last element of a vector.

```
% Define a vector
vector = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100];

% Slice from the 3rd element to the end
slice1 = vector(3:end);
disp('Slice from the 3rd element to the end:');
disp(slice1);

% Slice the last 4 elements
slice2 = vector(end-3:end);
disp('Last 4 elements:');
disp(slice2);

% Slice from the beginning to the second-to-last element
slice3 = vector(1:end-1);
disp('From the beginning to the second-to-last element:');
disp(slice3);

% Slice every second element from the 2nd to the end
slice4 = vector(2:2:end);
disp('Every second element from the 2nd to the end:');
disp(slice4);
```

```
Slice from the 3rd element to the end:
    30    40    50    60    70    80    90   100

Last 4 elements:
    70    80    90   100

From the beginning to the second-to-last element:
    10    20    30    40    50    60    70    80    90

Every second element from the 2nd to the end:
    20    40    60    80   100
```

Tutorial 3: The : (Colon) Operator for Vectors

The colon operator in MATLAB is a versatile tool primarily used for creating regularly spaced vectors, defining ranges, and accessing specific elements within arrays. It's represented by the colon symbol (:). If the step value is omitted,

```
myVector = start_value : step : end_value;
```

- **startValue** - the starting value.
- **step** - the step size If step is omitted, it defaults to 1.
- **endValue** - the ending value.

Increasing Sequence 1-10

```
% Vector Declaration using the : (colon) operator
% Define the range of values from 1 to 10 with default increments of 1
% x = startValue:step:endValue;
% x = 0:10;
myVector = 1:10;
disp("Vector: ")
disp(myVector)

% Vector Operation (Element-wise multiplication)
resultVector = myVector * 3;

% Display Result
disp(resultVector);
```

- Syntax: **startValue:step:endValue**
- Example: **myVector = 1:10;** creates a vector with values 1, 2, 3, etc up to 10.

Example run:

```
Vector:
     1     2     3     4     5     6     7     8     9    10

     3     6     9    12    15    18    21    24    27    30
```

Decreasing Sequence:

```
% Decreasing sequence
decreasingVector = 8:-1:4;
disp(decreasingVector)
```

Example run:

```
8     7     6     5     4
```

Negative Values:

```
% Negative values
negativeVector = -3:-1;
disp(negativeVector)
```

Example run:

```
-3    -2    -1
```

Fractional steps:

```
% Fractional steps
fractionalStepVector = 0.5:0.5:2.5
disp(fractionalStepVector)
```

Example run:

```
0.5000    1.0000    1.5000    2.0000    2.5000
```

Reverse order:

```
% Reverse order
reverseOrderVector = 15:-2:9;
disp(reverseOrderVector)
```

Example run:

```
15    13    11     9
```

Assignment 2: Vector and Matrix Practice

1. Create the vector **vec** in three different ways: using just square brackets, using the colon operator, and using linspace:
   vec = 5 7 9 11
2. Create a variable **myend**, which stores a random integer (randint) in the inclusive range from 5 to 9. Using the colon operator, create a vector that iterates from 1 to **myend** in steps of 3.
3. Create two row vector variables. Concatenate them together to create a new row vector variable.

4. Using the colon operator and the transpose operator, create a column vector **myvec** that has the values -1 to 1 in steps of 0.5.
5. Write an expression that refers to only the elements that have odd-numbered subscripts in a vector, regardless of the length of the vector. Test your expression on vectors that have both an odd and even number of elements.

Assignment 3: Colon Operator Practice

Include comments in your code to explain each step.

1. Create a vector containing even integers from 2 to 20.
2. Generate a vector with values from 1 to 5 with an increment of 0.5.
3. Given the vector `A = [4, 7, 10, 13, 16, 19]`, extract elements from the 2nd to the 4th position.
4. Generate a vector containing the squares of numbers from 1 to 7.
5. Given the vector `B = [3, 6, 9, 12, 15, 18, 21]`, access elements at positions 2, 4, and 6.

Example run:

```
Even integers:
     2     4     6     8    10    12    14    16    18    20

Incremental values:
    1.0000    1.5000    2.0000    2.5000    3.0000    3.5000    4.0000    4.5000    5.0000

Subset:
     7    10    13

Sequence of squares:
     1     4     9    16    25    36    49

Specific elements:
     6    12    18
```

Assignment 4: Circes Circle Calculator

In the world of ancient Greek mythology, Circe, a captivating enchantress and a minor goddess of magic, held an extraordinary affinity for circles. Amid her divine responsibilities, Circe sought moments of respite and decided to fashion a circle calculator, a magical tool

embodying her fascination with circles' symbolism of eternal cycles. This calculator, pulsating with enchantment, could unveil the secrets of circles—calculating circumferences, diameters, and areas, all of which fascinated Circe.

Underneath the starlit heavens, Circe would delicately touch the calculator, unraveling mathematical mysteries and finding joy in its revelations. Her calculator transcended time, captivating both gods and mortals. Scholars sought her guidance, connecting magic with mathematics, as her artifact became an emblem of their harmonious blend, echoing her love for both mysticism and logic. Thus, Circe's circle calculator stood as an enduring testament to her dual devotion—to the enchanting realm of ancient gods and the timeless elegance of mathematical truths.

She would like you to create a MATLAB circle calculator for her to use whenever she takes a break from being a goddess.

This program will ask the user to enter the radius of a circle. Calculate and display the circle's diameter, area, and circumference.

pi is the constant for pi in MATLAB. Example: **2 pi r**

1. Display a program title for the user.
2. Ask the user for the radius of a circle.
3. Calculate the diameter. Diameter of a circle: $d = 2r$
4. Calculate the area. Area of a circle: $a = \pi r^2$
5. Calculate the circumference. Circumference of a circle: $c = 2\pi r$
6. Display the user input, diameter, area, and circumference.

Example run:

```
  ----- Circe's Circle Calculator ----
Enter the radius of the circle: 1.342
The diameter of the circle is: 2.68
The area of the circle is: 5.66
The circumference of the circle is: 8.43
```

Assignment Submission

1. Submit properly named and commented script files.
2. Attach a text file showing the successful execution of each script.

3. Attach all to the assignment in Blackboard.