Islamic University of Technology (IUT)

Organization of Islamic Cooperation (OIC)

Department of Electrical and Electronic Engineering (EEE)

Course No.	: EEE 4416
Course Name	: Simulation Lab.
Experiment No.	: 03
Experiment Name	: Introduction to Conditionals and Loops in MATLAB

Objective:

To get familiarized with the use of conditionals, iteration and loops in MATLAB.

1. RELATIONAL OPERATORS:

Sign	Description	Sign	Description
<	Less than	>=	Greater than or equal to
>	Greater than	==	Equal to
<=	Less than or equal to	~=	Not equal to

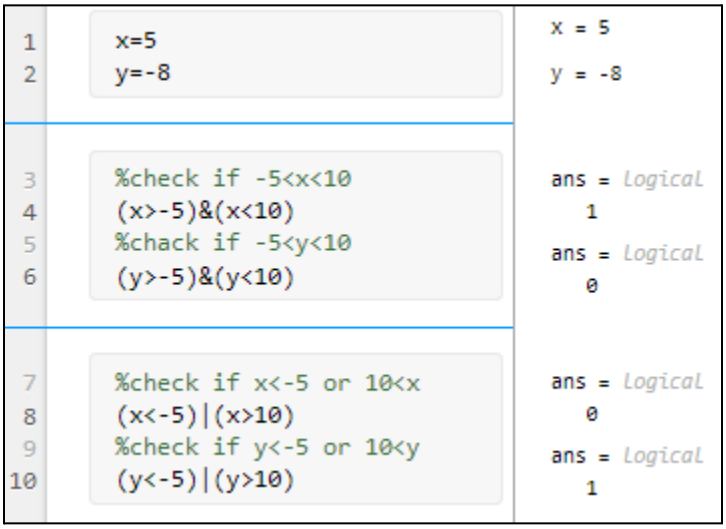
Example:

```
1  x=5
   x = 5

2  x<10
   ans = logical
      1

3  x>10
   ans = logical
      0

4  x<=10
   ans = logical
      1

5  x>=10
   ans = logical
      0

6  x==10
   ans = logical
      0

7  x~=10
   ans = logical
      1
```

2. LOGICAL OPERATORS:

Sign	Name	Description
&	And operation	Operates on two operands (A and B). If both are true, the result is true (1); otherwise the result is false (0).
|	Or operation	Operates on two operands (A and B). If either one, or both, are true, the result is true (1); otherwise (both are false) the result is false (0).

Example:



3. BUILT-IN LOGICAL FUNCTIONS:

Function	Description
all(A)	Returns 1 (true) if all elements in a vector A are true (nonzero). Returns 0 (false) if one or more elements are false (zero). If A is a matrix, it treats columns of A as vectors, and returns a vector with 1s and 0s.
any(A)	Returns 1 (true) if any element in a vector A is true (nonzero). Returns 0 (false) if all elements are false (zero). If A is a matrix, it treats columns of A as vectors, and returns a vector with 1s and 0s.
find(A)	A is a vector, and returns the indices of the nonzero elements.
find(relational operator) example: find(A>d)	If A is a vector, returns the address of the elements that are larger than d (any relational operator can be used).

Example:

```
A=[-3 11 8 5 2 -6]
A = 1×6
      -3     11     8      5      2      -6
B=[0 3 0 5; -7 0 1 4]
B = 2×4
       0      3      0      5
      -7      0      1      4
%all function
all(A)
ans = logical
   1

all(B)
ans = 1×4 logical array
   0   0   0   1
%any function
any(A)
ans = logical
   1

any(B)
ans = 1×4 logical array
   1   1   1   1
%find function
find(A)
ans = 1×6
      1      2      3      4      5      6
find(B)
ans = 5×1
      2
      3
      6
      7
      8
%find relational operation function
find(A>1)
ans = 1×4
      2      3      4      5
find(B>1)
ans = 3×1
      3
      7
      8
```

4. CONDITIONAL STATEMENTS:

i. Conditional Expression:
- Stated condition that evaluates to either true or false.

ii. Examples of the basic form of a conditional expression:
- `if a < b`
- `if c >= 5`
- `if a == b`
- `if a ~= 0`
- `if (d<h)&(x>7)`
- `if (x~=13)|(y<0)`

iii. Execution Flow:
- If the expression is true, statements following the expression are executed.
- If false, the computer skips the specified group of statements.

4.1. if-end Structure:

Syntax:

```
if <expression 1>
% Executes the following statements when "expression 1" is true
<statement(s)>
end
```

Example:

```
clear all
clc
x=input("Give the Number: ")
if x>0
      disp("positive")
end
```

4.2. if-else-end Structure:

Syntax:

```
if <expression 1>
% Executes the following statements when "expression 1" is true
<statement(s)>
else
% Executes the following statements when "expression 1" is false
<statement(s)>
end
```

Example:

```
clear all
clc
x=input("Give the Number: ")
if x>0
      disp("positive")
else
      disp("non-positive")
end
```

4.3. if-elseif-else-end Structure:

Syntax:

```
if <expression 1>
% Executes the following statements when "expression 1" is true
<statement(s)>

elseif <expression 2>
% Executes the following statements when "expression 2" is true
<statement(s)>

elseif <expression 3>
% Executes the following statements when "expression 3" is true
<statement(s)>
.
.
.

else
% Executes the following statements all the above expression is false
<statement(s)>
end
```

Example:

```
clear all
clc
x=input("Give the Number: ")
if x>0
      disp("positive")
elseif x<0
      disp("negative")
else
      disp("zero")
end
```

```
clear all
clc
x=input("Give the Number: ")
if x>10
      disp("Bigger than 10")
elseif (x>0)&(x<=10)
      disp("Positive but less than or equal to 10")
elseif x==0
      disp("Zero")
elseif (x<0)&(x>=-10)
      disp("Negative but more than or equal to -10")
else
      disp("Smaller than -10")
end
```

Practice problem -1:

Write a MATLAB script that takes a student's numerical score as input and outputs their corresponding letter grade based on the following grading scale:

A: 90-100
B: 80-89
C: 70-79
D: 60-69
F: 0-59
**Additionally, include a check for invalid scores (less than 0 or greater than 100) and display an error message for such cases.

5. switch-case STATEMENT:

i. Switch-Case Structure Overview:
   - Used to direct program flow by choosing one group of commands among several possible groups.
   - Structure: switch expression, case commands (with values), optional otherwise command, and end statement.

ii. Switch Command:
   - The switch expression is compared with values next to case statements.
   - It can be a scalar, string, or a mathematical expression with pre-assigned variables.

iii. Case Commands:
   - Multiple case commands with associated values.
   - If a match is found, the group of commands following the matching case is executed.
   - Only the first matching case is executed in MATLAB, unlike C where break statements are needed.

iv. Execution Flow:
- If more than one match, only the first matching case is executed.
- If no match and an otherwise statement is present, the commands between otherwise and end are executed.
- If no match and no otherwise statement, no command group is executed.

v. Multiple Values in Case:
- Case statements can have multiple values in the form of {value1, value2, ...} (cell array).
- The case is executed if at least one of the values matches the switch expression.

Syntax:

```
switch <switch_expression>
   case <value 1>
      <statement(s)>
   case <value 2>
      <statement(s)>
   case <value 3>
      <statement(s)>
     .
     .
     .
   otherwise
      <statement(s)>
end
```

Example:

```
clear all
clc
day = input("Enter the name of the day: ","s");
switch day
   case "Monday"
      disp("Start of the work week")
   case "Tuesday"
      disp("Day 2 of the work week")
   case "Wednesday"
      disp("Day 3 of the work week")
   case "Thursday"
      disp("Day 4 of the work week")
   case "Friday"
      disp("Last day of the work week")
   otherwise
      disp("Weekend!")
end
```

Practice problem -2:

Write a MATLAB program using switch case statement that takes a student's grade as input and provides a comment based on the following grading scale:

A: Excellent
B: Good
C: Satisfactory
D: Needs Improvement
E: Barely Passed
F: Fail

**Additionally, include an error message for grades that are not within the standard A-F range.

6. for-end Loops:

In for-end loops the execution of a command, or a group of commands, is repeated a predetermined number of times.

Syntax:

```
for index = values
   <statement(s)>
      .
      .
      .
end
```

Where, **values** has one of the following 3 forms:

i. **initVal : endVal** —
   Increment the index variable from initVal to endVal by 1, and repeat the execution of statements until index is greater than endVal.

ii. **initVal : step : endVal** —
   Increment index by the value step on each iteration, or decrements index when step is negative.

iii. **valArray** —
   Create a column vector, index, from subsequent columns of array valArray on each iteration. For example, on the first iteration, index = valArray(:,1). The loop executes a maximum of n times, where n is the number of columns of valArray, given by numel(valArray(1,:)). The input valArray can be of any MATLAB® data type, including a character vector, cell array, or struct.

Example:

```
clear all
clc
for i = 1:5
      i
end
```

```
clear all
clc
x = 1:5;
for j=1:5
      s(j) = x(j)^2
      c(j) = x(j)^3
      f(j) = x(j)^4
end
s
c
f
```

```
clear all
clc
x=input("Factorial Number: ")
fact=1;
for i=1:x
      fact=fact*i;
end
fact
```

Practice problem -3:

Write a MATLAB script to generate the Fibonacci series up to the nth term using a **for** loop. The Fibonacci sequence is a series of numbers in which each number (Fibonacci number) is the sum of the two preceding ones, usually starting with 0 and 1. The Fibonacci series is defined as follows:

$F(n)=F(n-1)+F(n-2)$
where, $F(1) = 0$ and $F(2) = 1$.

**Your script should take an input 'n' from the user and display the Fibonacci series up to the nth term.

7. while-end Loops:

while-end loops are used in situations when looping is needed but the number of passes is not known in advance. In while-end loops the number of passes is not specified when the looping process starts. Instead, the looping process continues as long as a stated condition is satisfied.

i. While Loop Basics:
   - While loops in MATLAB use a conditional expression.
   - Execution depends on the truth of the condition (1 for true, 0 for false).

ii. Loop Execution:
   - If the condition is false, MATLAB skips to the end statement and continues.
   - If true, MATLAB executes statements between while and end, then checks the condition again.

iii. Loop Termination:
   - Loop continues until the condition becomes false.
   - Variables in the condition must be assigned values before the first loop iteration.

iv. Variable Updates:
   - At least one variable in the condition should get a new value within the loop.
   - Without updates, the loop may become infinite as the condition remains true.

Syntax:

```
while conditional expression
   <statement(s)>
        .
        .
        .
end
```

Example:

```
clear all
clc
i=1;
while i<=5
      i
      i = i + 1;
end
```

```
clear all
clc
x = 1:5;
j=1;
while j<=5
      s(j) = x(j)^2
      c(j) = x(j)^3
      f(j) = x(j)^4
      j=j+1;
end
s
c
f
```

```
clear all
clc
x=input("Factorial Number: ")
fact=1;
i=1;
while i<=x
      fact=fact*i;
      i=i+1;
end
fact
```

Practice problem -4:

Write a MATLAB script to generate the Fibonacci series up to the nth term using a **while** loop. The Fibonacci sequence is a series of numbers in which each number (Fibonacci number) is the sum of the two preceding ones, usually starting with 0 and 1. The Fibonacci series is defined as follows:

$F(n)=F(n-1)+F(n-2)$
where, $F(1) = 0$ and $F(2) = 1$.

**Your script should take an input 'n' from the user and display the Fibonacci series up to the nth term.

8. NESTED LOOPS AND NESTED CONDITIONAL STATEMENTS:

Loops and conditional statements can be nested within other loops or conditional statements. This means that a loop and/or a conditional statement can start (and end) within another loop or conditional statement. There is no limit to the number of loops and conditional statements that can be nested. It must be remembered, however, that each if, case, for, and while statement must have a corresponding end statement.

****Nested for-end Loop:**

In the loops shown in this Figure 1, if, for example, n = 3 and m = 4, then first k = 1 and the nested loop executes four times with h = 1, 2, 3, 4. Next k = 2 and the nested loop executes again four times with h = 1, 2, 3, 4. Finally k = 3 and the nested loop executes again four times. Every time a nested loop is typed, MATLAB automatically indents the new loop relative to the outside loop. Nested loops and conditional statements are demonstrated in the following sample problem.

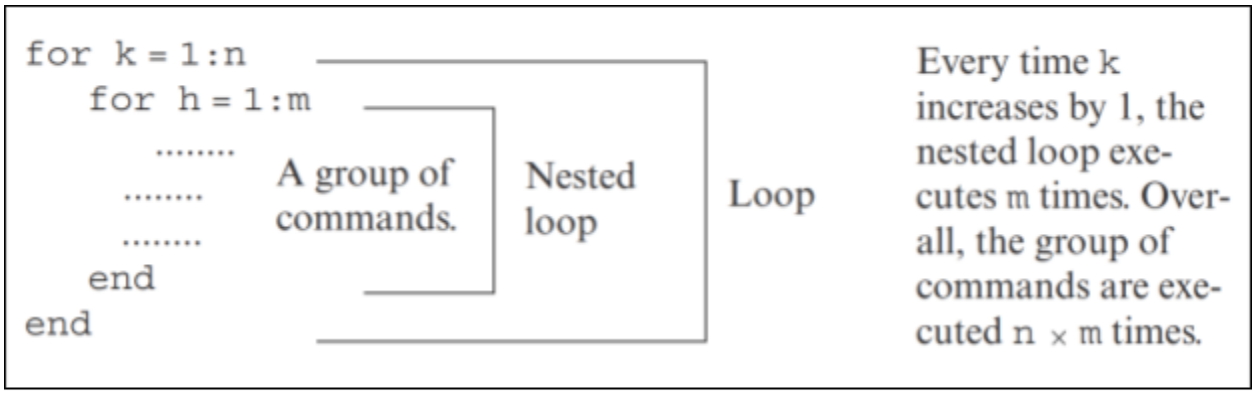


Figure 1: Structure of nested loops.

Example:

The following code computes the product of all the elements within a matrix.

```
clear all
clc
x=randi(5,[3,4])
prod=1;
for i=1:3
      for j=1:4
        prod=prod*x(i,j)
      end
end
prod
```

Practice problem -5:

Write a MATLAB code to generate the following matrix:

$$X = \begin{vmatrix} 1 & 0 & 0 & 0 \\ 1 & 2 & 0 & 0 \\ 1 & 2 & 3 & 0 \\ 1 & 2 & 3 & 4 \end{vmatrix}$$

**The above matrix is a 4 by 4 square matrix. Now, modify the code into a user defined function in which the user will insert the size of a square matrix and the corresponding matrix will be generated.

9. THE break AND continue COMMANDS:

****The break command:**

i. When inside a loop (for or while), the break command terminates the execution of the loop (the whole loop, not just the last pass). When the break command appears in a loop, MATLAB jumps to the end command of the loop and continues with the next command (it does not go back to the for command of that loop).
ii. If the break command is inside a nested loop, only the nested loop is terminated.
iii. When a break command appears outside a loop in a script or function file, it terminates the execution of the file.
iv. The break command is usually used within a conditional statement. In loops it provides a method to terminate the looping process if some condition is met

Example:

```
clear all
clc
for i=1:10
      if(i==7)
      disp("breaking the loop")
      break
      end
      i
end
```

****The continue command:**

i. The continue command can be used inside a loop (for or while) to stop the present pass and start the next pass in the looping process.

ii. The continue command is usually a part of a conditional statement. When MATLAB reaches the continue command, it does not execute the remaining commands in the loop, but skips to the end command of the loop and then starts a new pass

Example:

```
clear all
clc
for i=1:10
      if (mod(i,2)==0)
      disp("skipping even numbers")
      continue
      end
      i
end
```