Lecture 2

1. Relational Operators

Relational operators can also work on both scalar and non-scalar data. Relational operators for arrays perform element-by-element comparisons between two arrays and return a logical array of the same size, with elements set to logical 1 (true) where the relation is true and elements set to logical 0 (false) where it is not.

The following table shows the relational operators available in MATLAB:

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

Example :Create a script file and type the following code:

```
a = 100;
b = 200;
if (a >= b)
max = a
else
max = b
end
```

When you run the file, it produces following result:

```
max =200
```

2. Logical Operators

MATLAB offers two types of logical operators and functions:

Element-wise - These operators operate on corresponding elements of logical arrays.

Short-circuit - These operators operate on scalar and logical expressions.

Element-wise logical operators operate element-by-element on logical arrays. The symbols &, |, and ~ are the logical array operators AND, OR, and NOT.

Short-circuit logical operators allow short-circuiting on logical operations. The symbols && and || are the logical short-circuit operators AND and OR.

Example :Create a script file and type the following code:

```
a = 5;
b = 20;
if ( a && b )
disp('Line 1 - Condition is true');
end
if ( a || b )
disp('Line 2 - Condition is true');
```

Decision Making

Decision making structures require that the programmer should specify one or more conditions to be evaluated or tested by the program, along with a statement or statements to be executed if the condition is determined to be true, and optionally, other statements to be executed if the condition is determined to be false.
Following is the general form of a typical decision making structure found in most of the programming languages:

1. if... end Statement

An **if ... end** statement consists of an **if** statement and a Boolean expression followed by one or more statements. It is delimited by the **end** statement.

Syntax

The syntax of an if statement in MATLAB is:

```
if <expression>
% statement(s) will execute if the boolean expression is true
<statements>
end
```

If the expression evaluates to true, then the block of code inside the if statement will be executed. If the expression evaluates to false, then the first set of code after the end statement will be executed.

Example :Create a script file and type the following code:

```
a = 10;
% check the condition using if statement
if a < 20
% if condition is true then print the following
fprintf('a is less than 20\n' );
end
fprintf('value of a is : %d\n', a);
```

When you run the file, it displays the following result:

```
a is less than 20
value of a is : 10
```

2. if...else...end Statement

An if statement can be followed by an optional else statement, which executes when the expression is false.

Syntax

The syntax of an if...else statement in MATLAB is:

```
if <expression>
% statement(s) will execute if the boolean expression is true
<statement(s)>
else
<statement(s)>
% statement(s) will execute if the boolean expression is false
end
```

If the boolean expression evaluates to true, then the if block of code will be executed, otherwise else block of code will be executed.

Example :Create a script file and type the following code:

```
a = 100;
% check the boolean condition
if a < 20
% if condition is true then print the following
fprintf('a is less than 20\n' );
else
% if condition is false then print the following
fprintf('a is not less than 20\n' );
end
fprintf('value of a is : %d\n', a);
```

When the above code is compiled and executed, it produces the following result:

```
a is not less than 20
value of a is : 100
```

3.The switch Statement

A switch block conditionally executes one set of statements from several choices. Each choice is covered by a case statement.

The **syntax** of switch statement in MATLAB is:

```
switch <switch_expression>
case <case_expression>
<statements>
case <case_expression>
<statements>
...
...
otherwise
<statements>
end
```

Example :Create a script file and type the following code in it:

```
grade = 'B';
switch(grade)
case 'A'
fprintf('Excellent!\n' );
case 'B'
fprintf('Well done\n' );
case 'C'
fprintf('Well done\n' );
case 'D'
fprintf('You passed\n' );
case 'F'
fprintf('Better try again\n' );
otherwise
fprintf('Invalid grade\n' );
end
```

When you run the file, it displays:

```
Well done
Your grade is B
```

Loop Types

There may be a situation when you need to execute a block of code several number of times. In general, statements are executed sequentially. The first statement in a function is executed first, followed by the second, and so on.
Programming languages provide various control structures that allow for more complicated execution paths.
A loop statement allows us to execute a statement or group of statements multiple times

1. The while Loop

The while loop repeatedly executes statements while condition is true.

Syntax

The syntax of a while loop in MATLAB is:

```
while <expression>
<statements>
end
```

The while loop repeatedly executes program statement(s) as long as the expression remains true.
An expression is true when the result is nonempty and contains all nonzero elements (logical or real numeric). Otherwise, the expression is false.

Example :Create a script file and type the following code:

```
a = 10;
% while loop execution
while( a < 20 )
fprintf('value of a: %d\n', a);
a = a + 1;
end
```

When you run the file, it displays the following result:

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```

2. The for Loop

A **for loop** is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times.

Syntax

The syntax of a **for loop** in MATLAB is:

```
for index = values
<program statements>
...
End
```

Example :Create a script file and type the following code:

```
for a = 10:20
fprintf('value of a: %d\n', a);
end
```

When you run the file, it displays the following result:

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
value of a: 16
value of a: 17
value of a: 18
value of a: 19
value of a: 20
```

Example :Create a script file and type the following code:

```
for a = 1.0: -0.1: 0.0
disp(a)
end
```

When you run the file, it displays the following result:

```
1
0.9000
0.8000
0.7000
0.6000
0.5000
0.4000
0.3000
0.2000
0.1000
```