Chapter 6
Programming in MATLAB

Instructor: Dr. Talal Skaik
Islamic University of Gaza
Faculty of Engineering
A computer program is a sequence of computer commands. In a simple program the commands are executed one after the other in the order they are typed.

6.1 RELATIONAL AND LOGICAL OPERATORS

A relational operator compares two numbers by determining whether a comparison statement (e.g., $5 < 8$) is true or false.

- If the statement is true, it is assigned a value of 1.
- If the statement is false, it is assigned a value of 0.
### 6.1 RELATIONAL AND LOGICAL OPERATORS

**Relational operators:**

<table>
<thead>
<tr>
<th>Relational operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;</td>
<td>Less than</td>
</tr>
<tr>
<td>&gt;</td>
<td>Greater than</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Less than or equal to</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Greater than or equal to</td>
</tr>
<tr>
<td>==</td>
<td>Equal to</td>
</tr>
<tr>
<td>!=</td>
<td>Not Equal to</td>
</tr>
</tbody>
</table>
6.1 RELATIONAL AND LOGICAL OPERATORS

Relational operators:

- When two numbers are compared, the result is 1 (logical true) if the comparison, according to the relational operator, is true, and 0 (logical false) if the comparison is false.

- If two scalars are compared, the result is a scalar 1 or 0. If two arrays are compared (only arrays of the same size can be compared), the comparison is done element-by-element, and the result is a logical array of the same size with 1s and 0s according to the outcome of the comparison at each address.

- If a scalar is compared with an array, the scalar is compared with every element of the array, and the result is a logical array with 1s and 0s according to the outcome of the comparison of each element.
6.1 RELATIONAL AND LOGICAL OPERATORS

```
>> 5>8
ans =
0

>> a=5<10
Checks if 5 is smaller than 10, and assigns the answer to a.
a =
1

>> y=(6<10)+(7>8)+(5*3==60/4)

Equal to 1 since 6 is smaller than 10.
Equal to 0 since 7 is not larger than 8.
Equal to 1 since 5*3 is equal to 60/4.

y =
2
```
Checks if 5 is larger than 8.
Since the comparison is false (5 is not larger than 8) the answer is 0.
Since the comparison is true (5 is smaller than 10) the number 1 is assigned to a.
Using relational operators in math expression.
6.1 RELATIONAL AND LOGICAL OPERATORS

```matlab
>> b=[15 6 9 4 11 7 14]; c=[8 20 9 2 19 7 10];
>> d=c>=b  \text{ Checks which elements of } c \text{ are larger than or equal to elements of } b.
\text{d = }
\begin{bmatrix}
0 & 1 & 1 & 0 & 1 & 1 & 0
\end{bmatrix}
\text{Assigns 1 where an element of } c \text{ is larger than or equal to an element of } b.

>> b == c  \text{ Checks which elements of } b \text{ are equal to elements of } c.
\text{ans = }
\begin{bmatrix}
0 & 0 & 1 & 0 & 0 & 1 & 0
\end{bmatrix}

>> b~=c  \text{ Checks which elements of } b \text{ are not equal to elements of } c.
\text{ans = }
\begin{bmatrix}
1 & 1 & 0 & 1 & 1 & 0 & 1
\end{bmatrix}

>> f=b-c>0  \text{ Subtracts elements of } c \text{ from elements of } b \text{ and then checks which elements are larger than zero.}
\text{f = }
\begin{bmatrix}
1 & 0 & 0 & 1 & 0 & 0 & 1
\end{bmatrix}
```
6.1 RELATIONAL AND LOGICAL OPERATORS

Define a $3 \times 3$ matrix $A$.

\[
A = \begin{bmatrix}
2 & 9 & 4 \\
-3 & 5 & 2 \\
6 & 7 & -1
\end{bmatrix}
\]

Checks which elements in $A$ are smaller than or equal to 2. Assigns the results to matrix $B$.

\[
B = \begin{bmatrix}
1 & 0 & 0 \\
1 & 0 & 1 \\
0 & 0 & 1
\end{bmatrix}
\]
When a logical vector is used for addressing another vector, it extracts from that vector the elements in the positions where the logical vector has 1s. For example:

```
>> r = [8 12 9 4 23 19 10]  Define a vector r.
[8 12 9 4 23 19 10]

>> s=r<=10  Checks which r elements are smaller than or equal to 10.
  A logical vector s with 1s at positions where elements of r are smaller than or equal to 10.

  s =
  1 0 1 1 0 0 1

>> t=r(s)  Use s for addresses in vector r to create vector t.
  Vector t consists of elements of r in positions where s has 1s.

  t =
  8 9 4 10

>> w=r(r<=10)  The same procedure can be done in one step.
  w =
  8 9 4 10
```
Order of precedence:
In a mathematical expression that includes relational and arithmetic operations, the arithmetic operations (+, −, *, /, \) have precedence over relational operations.

```
>> 3+4<16/2
ans =
    1
>> 3+(4<16)/2
ans =
    3.5000
```

+ and / are executed first.
The answer is 1 since 7 < 8 is true.
4 < 16 is executed first, and is equal to 1, since it is true.
3.5 is obtained from 3 + 1/2.
### Logical operators:

Logical operators in MATLAB are:

<table>
<thead>
<tr>
<th>Logical operator</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp; Example: A&amp;B</td>
<td>AND</td>
<td>Operates on two operands (A and B). If both are true, the result is true (1); otherwise the result is false (0).</td>
</tr>
<tr>
<td></td>
<td>OR</td>
<td>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).</td>
</tr>
<tr>
<td>~ Example: ~A</td>
<td>NOT</td>
<td>Operates on one operand (A). Gives the opposite of the operand; true (1) if the operand is false, and false (0) if the operand is true.</td>
</tr>
</tbody>
</table>
Logical operators have numbers as operands. A nonzero number is true, and a zero number is false.

```plaintext
>> 3 & 7
ans = 1
3 and 7 are both true (nonzero), so the outcome is 1.

>> a = 5 | 0
a = 1
5 OR 0 (assign to variable a).
1 is assigned to a since at least one number is true (nonzero).

>> ~25
ans = 0
The outcome is 0 since 25 is true (nonzero) and the opposite is false.

>> t = 25 * ((12 & 0) + (~0) + (0 | 5))
t = 50
Using logical operators in a math expression.
```
6.1 RELATIONAL AND LOGICAL OPERATORS

```
>> x=[9 3 0 11 0 15]; y=[2 0 13 -11 0 4];
>> x&y
ans =
   1  0  0  1  0  1

>> z=x|y
z =
   1  1  1  1  0  1

>> ~(x+y)
ans =
   0  0  0  1  1  0
```

The outcome is a vector with 1 in every position where both `x` and `y` are true (nonzero elements), and 0s otherwise.

The outcome is a vector with 1 in every position where either or both `x` and `y` are true (nonzero elements), and 0s otherwise.

The outcome is a vector with 0 in every position where the vector `x + y` is true (nonzero elements), and 1 in every position where `x + y` is false (zero elements).
## 6.1 RELATIONAL AND LOGICAL OPERATORS

**Order of precedence:**

<table>
<thead>
<tr>
<th>Precedence</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (highest)</td>
<td>Parentheses (if nested parentheses exist, inner ones have precedence)</td>
</tr>
<tr>
<td>2</td>
<td>Exponentiation</td>
</tr>
<tr>
<td>3</td>
<td>Logical NOT (~)</td>
</tr>
<tr>
<td>4</td>
<td>Multiplication, division</td>
</tr>
<tr>
<td>5</td>
<td>Addition, subtraction</td>
</tr>
<tr>
<td>6</td>
<td>Relational operators (&gt; ,&lt; , &gt;= , &lt;= , == , !=)</td>
</tr>
<tr>
<td>7</td>
<td>Logical AND (&amp;)</td>
</tr>
<tr>
<td>8 (lowest)</td>
<td>Logical OR (</td>
</tr>
</tbody>
</table>
6.1 RELATIONAL AND LOGICAL OPERATORS

```
>> x=-2; y=5;
Define variables x and y.
>> -5<x<-1
This inequality is correct mathematically. The answer, however, is false since MATLAB executes from left to right. -5 < x is true (-1) and then 1 < -1 is false (0).
ans =
0
>> -5<x & x<-1
The mathematically correct statement is obtained by using the logical operator &. The inequalities are executed first. Since both are true (1), the answer is 1.
ans =
1
>> ~(y<7)
y < 7 is executed first, it is true (1), and ~1 is 0.
ans =
0
>> ~y<7
~y is executed first. y is true (1) (since y is nonzero), ~1 is 0, and 0 < 7 is true (1).
ans =
1
```
6.1 RELATIONAL AND LOGICAL OPERATORS

Define variables \( x \) and \( y \).

\[
\begin{align*}
\gg \ x &= -2; \quad y = 5; \\
\gg \ \sim \ (y \geq 8) \lor (x < -1) \\
\text{ans} &= 0 \\
\gg \ \sim \ (y \geq 8) \lor (x < -1) \\
\text{ans} &= 1
\end{align*}
\]

\( y \geq 8 \) (false), and \( x < -1 \) (true) are executed first. OR is executed next (true). \( \sim \) is executed last, and gives false (0).

\( y \geq 8 \) (false), and \( x < -1 \) (true) are executed first. NOT of \( y \geq 8 \) is executed next (true). OR is executed last, and gives true (1).

**Built-in logical functions:**

MATLAB has built-in functions that are equivalent to the logical operators. These functions are:

- \( \text{and}(A, B) \) equivalent to \( A \& B \)
- \( \text{or}(A, B) \) equivalent to \( A \mid B \)
- \( \text{not}(A) \) equivalent to \( \sim A \)
# 6.1 RELATIONAL AND LOGICAL OPERATORS

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>xor(a,b)</code></td>
<td>Exclusive or. Returns true (1) if one operand is true and the other is false.</td>
<td><code>&gt;&gt; xor(7,0)</code>&lt;br&gt;<code>ans = 1</code>&lt;br&gt;<code>&gt;&gt; xor(7,-5)</code>&lt;br&gt;<code>ans = 0</code></td>
</tr>
<tr>
<td><code>all(A)</code></td>
<td>Returns 1 (true) if all elements in a vector A are true (nonzero). Returns 0 (false) if one or more elements are false (zero).&lt;br&gt;If A is a matrix, treats columns of A as vectors, and returns a vector with 1s and 0s.</td>
<td><code>&gt;&gt; A=[6 2 15 9 7 11];</code>&lt;br&gt;<code>&gt;&gt; all(A)</code>&lt;br&gt;<code>ans = 1</code>&lt;br&gt;<code>&gt;&gt; B=[6 2 15 9 0 11];</code>&lt;br&gt;<code>&gt;&gt; all(B)</code>&lt;br&gt;<code>ans = 0</code></td>
</tr>
</tbody>
</table>
### 6.1 RELATIONAL AND LOGICAL OPERATORS

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>any(A)</code></td>
<td>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, treats columns of A as vectors, and returns a vector with 1s and 0s.</td>
<td></td>
</tr>
<tr>
<td><code>find(A)</code></td>
<td>If A is a vector, returns the indices of the nonzero elements.</td>
<td></td>
</tr>
<tr>
<td><code>find(A&gt;d)</code></td>
<td>If A is a vector, returns the address of the elements that are larger than d (any relational operator can be used).</td>
<td></td>
</tr>
</tbody>
</table>

```matlab
>> A=[6 0 15 0 0 11];
>> any(A)
ans =
    1
>> B = [0 0 0 0 0 0];
>> any(B)
ans =
    0

>> A=[0 9 4 3 7 0 0 1 8];
>> find(A)
ans =
    2    3    4
         5    8    9
>> find(A>4)
ans =
    2    5    9
```
The operations of the four logical operators, and, or, xor, and not can be summarized in a truth table:

<table>
<thead>
<tr>
<th>INPUT</th>
<th>OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>AND A&amp;B</td>
</tr>
<tr>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>true</td>
<td>true</td>
</tr>
</tbody>
</table>
A conditional statement is a command that allows MATLAB to make a decision of whether to execute a group of commands that follow the conditional statement, or to skip these commands.

In a conditional statement a conditional expression is stated:

- If the expression is true, a group of commands that follow the statement are executed.
- If the expression is false, the computer skips the group.

The basic form of a conditional statement is:

```
if conditional expression consisting of relational and/or logical operators.
```
6.2 CONDITIONAL STATEMENTS

Examples:

```
if a < b
if c >= 5
if a == b
if a ~= 0
if (d<h) & (x>7)
if (x~=13) | (y<0)
```

- For every if statement there is an end statement.
- The if statement is commonly used in three structures, if-end, if-else-end, and if-elseif-else-end, which are described next.

All the variables must have assigned values.
6.2 CONDIONTIAL STATEMENTS

6.2.1 The if-end Structure

---

**Figure 6-1:** The structure of the if-end conditional statement.
Sample Problem 6-2: Calculating worker’s pay

A worker is paid according to his hourly wage up to 40 hours, and 50% more for overtime. Write a program in a script file that calculates the pay to a worker. The program asks the user to enter the number of hours and the hourly wage. The program then displays the pay.

```matlab
% 6.2 CONDITIONAL STATEMENTS

Please enter the number of hours worked
h = input('Please enter the hourly wage in $ ');
Pay = t*h;
if t>40
    Pay = Pay+(t-40)*0.5*h;
end
fprintf('The worker's pay is $ %.2f', Pay)
```
Sample Problem 6-2: Calculating worker’s pay

Application of the program (in the Command Window) for two cases is shown below (the file was saved as Workerpay):

```
>> Workerpay
Please enter the number of hours worked  35
Please enter the hourly wage in $  8
The worker’s pay is  $ 280.00
>> Workerpay
Please enter the number of hours worked  50
Please enter the hourly wage in $  10
The worker’s pay is  $ 550.00
```
6.2.2 The if-else-end Structure:

provides a means for choosing one group of commands, out of a possible two groups, for execution.

Figure 6-2: The structure of the if-else-end conditional statement.
6.2.3 The if-elseif-else-end Structure

This structure includes two conditional statements (if and elseif) that make it possible to select one out of **three** groups of commands for execution.

- Several elseif statements and associated groups of commands can be added.
- In this way more conditions can be included.

![Diagram of the if-elseif-else-end structure](image.png)

Figure 6-3: The structure of the if-elseif-else-end conditional statement.
Sample Problem 6-3: Water level in water tower

The tank in a water tower has the geometry shown in the figure (the lower part is a cylinder and the upper part is an inverted frustum of a cone). Inside the tank there is a float that indicates the level of the water. Write a MATLAB program that determines the volume of the water in the tank from the position (height $h$) of the float. The program asks the user to enter a value of $h$ in m, and as output displays the volume of the water in m$^3$. 
6.2 CONDITIONAL STATEMENTS

Sample Problem 6-3: Water level in water tower

Solution

For $0 \leq h \leq 19$ m the volume of the water is given by the volume of a cylinder with height $h$: $V = \pi 12.5^2 h$.

For $19 < h \leq 33$ m the volume of the water is given by adding the volume of a cylinder with $h = 19$ m, and the volume of the water in the cone:

$$V = \pi 12.5^2 \cdot 19 + \frac{1}{3} \pi (h - 19)(12.5^2 + 12.5 \cdot r_h + r_h^2)$$

where $r_h = 12.5 + \frac{10.5}{14}(h - 19)$. 
% The program calculates the volume of the water in the water tower.

h=input('Please enter the height of the float in meter ');
if h > 33
    disp('ERROR. The height cannot be larger than 33 m.')
elsetif h < 0
    disp('ERROR. The height cannot be a negative number.')
elsetif h <= 19
    v = pi*12.5^2*h;
    fprintf('The volume of the water is %7.3f cubic meter.\n',v)
elset else
    rh=12.5+10.5*(h-19)/14;
    v=pi*12.5^2*19+pi*(h-19)*(12.5^2+12.5*rh+rh^2)/3;
    fprintf('The volume of the water is %7.3f cubic meter.\n',v)
end
The following is the display in the Command Window when the program is used with three different values of water height.

Please enter the height of the float in meter  8
The volume of the water is 3926.991 cubic meter.

Please enter the height of the float in meter  25.7
The volume of the water is 14114.742 cubic meter.

Please enter the height of the float in meter  35
ERROR. The height cannot be larger than 33 m.
The switch-case statement is another method that can be used to direct the flow of a program. It provides a means for choosing one group of commands for execution out of several possible groups. The first line is the switch command, which has the form:

```plaintext
switch switch expression
```

Following the `switch` command are one or several `case` commands. Each has a value (can be a scalar or a string) next to it (value1, value2, etc.) and an associated group of commands below it. After the last `case` command there is an optional `otherwise` command followed by a group of commands. The last line must be an `end` statement.
6.3 THE switch-case STATEMENT

```matlab
switch switch expression
    case value1
        Group 1 of commands.
    case value2
        Group 2 of commands.
    case value3
        Group 3 of commands.
    otherwise
        Group 4 of commands.
end
```

**Figure 6-4:** The structure of a `switch-case` statement.
6.3 THE switch-case STATEMENT

- If there is more than one match, only the first matching case is executed.
- If no match is found and the `otherwise` statement (which is optional) is present, the group of commands between otherwise and end is executed.
- If no match is found and the `otherwise` statement is not present, none of the command groups is executed.

Note: In MATLAB only the first matching case is executed. After the group of commands associated with the first matching case are executed, the program skips to the end statement. This is different from the C language, where `break` statements are required.
Sample Problem 6-4: Converting units of energy

Write a program in a script file that converts a quantity of energy (work) given in units of either joule, ft-lb, cal, or eV to the equivalent quantity in different units specified by the user. The program asks the user to enter the quantity of energy, its current units, and the desired new units. The output is the quantity of energy in the new units.

The conversion factors are: $1 \text{ J} = 0.738 \text{ ft-lb} = 0.239 \text{ cal} = 6.24 \times 10^{18} \text{ eV}$.

Use the program to:

(a) Convert 150 J to ft-lb.
(b) Convert 2,800 cal to J.
(c) Convert 2.7 eV to cal.
6.3 THE switch-case STATEMENT

```matlab
Ein=input('Enter the value of the energy (work) to be converted: ');
EinUnits=input('Enter the current units (J, ft-lb, cal, or eV): ','s');
EoutUnits=input('Enter the new units (J, ft-lb, cal, or eV): ','s');
error=0;
switch EinUnits
    case 'J'
        EJ=Ein;
    case 'ft-lb'
        EJ=Ein/0.738;
    case 'cal'
        EJ=Ein/0.239;
    case 'eV'
        EJ=Ein/6.24e18;
    otherwise
        error=1;
end
```

- **Assign 0 to variable** `error`. First switch statement. Switch expression is a string with initial units.
- Each of the four `case` statements has a value (string) that corresponds to one of the initial units, and a command that converts `Ein` to units of J. (Assign the value to `EJ`).
- Assign 1 to `error` if no match is found. Possible only if initial units were typed incorrectly.
6.3 THE switch-case STATEMENT

```matlab
switch EoutUnits
    case 'J'
        Eout=EJ;
    case 'ft-lb'
        Eout=EJ*0.738;
    case 'cal'
        Eout=EJ*0.239;
    case 'eV'
        Eout=EJ*6.24e18;
    otherwise
        error=1;
end

if error
    disp('ERROR current or new units are typed incorrectly.')
else
    fprintf('E = %g %s',Eout,EoutUnits)
end
```

- **Second switch statement.** Switch expression is a string with new units.
- **Each of the four case statements.** Has a value (string) that corresponds to one of the new units, and a command that converts EJ to the new units. (Assign the value to Eout.)
- **Assign 1 to error if no match is found.** Possible only if new units were typed incorrectly.
- **If-else-end statement.**
- **If error is true (nonzero),** display an error message.
- **If error is false (zero),** display converted energy.
Converting units of energy

As an example, the script file (saved as EnergyConversion) is used next in the Command Window to make the conversion in part \(b\) of the problem statement.

```
>> EnergyConversion
Enter the value of the energy (work) to be converted: 2800
Enter the current units (J, ft-lb, cal, or eV): cal
Enter the new units (J, ft-lb, cal, or eV): J
E = 11715.5 J
```
A loop is another method to alter the flow of a computer program.

In a loop, the execution of a command, or a group of commands, is repeated several times consecutively.

Each round of execution is called a pass. In each pass at least one variable, but usually more than one, or even all the variables that are defined within the loop, are assigned new values.

MATLAB has two kinds of loops, for-end loops and while-end loops.
6.4.1 for-end Loops

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

```
for k = f:s:t
    ......  A group of MATLAB commands.
    ......  
    ......  
end
```

**Figure 6-5: The structure of a for-end loop.**
6.4 LOOPS

6.4.1 for-end Loops

\[ \text{for } k = f:s:t \]

- The loop index variable can have any variable name (usually i, j, k, m, and n are used, however, i and j should not be used if MATLAB is used with complex numbers).
- In the first pass \( k = f \) and the computer executes the commands between the `for` and `end` commands. Then, the program goes back to the `for` command for the second pass. \( k \) obtains a new value equal to \( k = f + s \), and the commands between the `for` and `end` commands are executed with the new value of \( k \). The process repeats itself until the last pass, where \( k = t \). Then the program does not go back to the `for`, but continues with the commands that follow the `end` command. For example, if \( k = 1:2:9 \), there are five loops, and the corresponding values of \( k \) are 1, 3, 5, 7, and 9.
- The increment \( s \) can be negative (i.e.; \( k = 25:-5:10 \) produces four passes with \( k = 25, 20, 15, 10 \)).
- If the increment value \( s \) is omitted, the value is 1 (default) (i.e.; \( k = 3:7 \) produces five passes with \( k = 3, 4, 5, 6, 7 \)).
The If \( f = t \), the loop is executed once.

If \( f > t \) and \( s > 0 \), or if \( f < t \) and \( s < 0 \), the loop is not executed.

If the values of \( k \), \( s \), and \( t \) are such that \( k \) cannot be equal to \( t \), then if \( s \) is positive, the last pass is the one where \( k \) has the largest value that is smaller than \( t \) (i.e., \( k = 8:10:50 \) produces five passes with \( k = 8, 18, 28, 38, 48 \)). If \( s \) is negative, the last pass is the one where \( k \) has the smallest value that is larger than \( t \).

In the for command \( k \) can also be assigned a specific value (typed as a vector). Example: for \( k = [7 \ 9 \ 8 \ 1 \ 3 \ 3 \ 5] \).

The value of \( k \) should not be redefined within the loop.

Each for command in a program must have an end command.

The value of the loop index variable \( (k) \) is not displayed automatically. It is possible to display the value in each pass (which is sometimes useful for debugging) by typing \( k \) as one of the commands in the loop.
When the loop ends, the loop index variable (k) has the value that was last assigned to it.

A simple example of a for-end loop (in a script file) is:

```matlab
for k = 1:3:10
    x = k^2
end
```

```
>> x =
    1
x =
    16
x =
    49
x =
    100
```
(a) Use a `for-end` loop in a script file to calculate the sum of the first $n$ terms of the series: $\sum_{k=1}^{n} \frac{(-1)^{k}k}{2^{k}}$. Execute the script file for $n = 4$ and $n = 20$.

(b) The function $\sin(x)$ can be written as a Taylor series by:

$$\sin x = \sum_{k=0}^{\infty} \frac{(-1)^{k}x^{2k+1}}{(2k+1)!}$$

Write a user-defined function file that calculates $\sin(x)$ by using the Taylor series. For the function name and arguments use $y = \text{Tsin}(x, n)$. The input arguments are the angle $x$ in degrees and $n$ the number of terms in the series. Use the function to calculate $\sin(150^\circ)$ using three and seven terms.
Sample Problem 6-5: Sum of a series

\[ \sum_{k=1}^{n} \frac{(-1)^k k}{2^k} \]

(a)

```python
n=input('Enter the number of terms ');
S=0;  # Setting the sum to zero.
for k=1:n
    S=S+(-1)^k*k/2^k;  # for-end loop.
end
fprintf('The sum of the series is: %f',S)
```

```
>> Exp6_5a
Enter the number of terms 4
The sum of the series is: -0.125000

>> Exp7_5a
Enter the number of terms 20
The sum of the series is: -0.222216
```
function y = Tsin(x,n) % Tsin calculates the sin using Taylor formula.
% Input arguments:
% x The angle in degrees, n number of terms.

xr=x*pi/180; % Converting the angle from degrees to radians.
y=0;
for k=0:n-1
    y=y+(-1)^k*xr^(2*k+1)/factorial(2*k+1);
end
(b)

\[
\sin x = \sum_{k=0}^{\infty} \frac{(-1)^k x^{2k+1}}{(2k+1)!}
\]

>> Tsin(150,3)  Calculating \(\sin(150^\circ)\) with three terms of Taylor series.
ans = 0.6523

>> Tsin(150,7)  Calculating \(\sin(150^\circ)\) with seven terms of Taylor series.
ans = 0.5000  The exact value is 0.5.
Sample Problem 6-6: Modify vector elements

A vector is given by $V = [5, 17, -3, 8, 0, -7, 12, 15, 20, -6, 6, 4, -7, 16]$. Write a program as a script file that doubles the elements that are positive and are divisible by 3 or 5, and, raises to the power of 3 the elements that are negative but greater than $-5$.

```matlab
V=[5, 17, -3, 8, 0, -7, 12, 15, 20, -6, 6, 4, -7, 16];
n=length(V); % Setting n to be equal to the number of elements in V.
for k=1:n
    if V(k)>0 & (rem(V(k),3)==0 | rem(V(k),5)==0)
        V(k)=2*V(k);
    elseif V(k) < 0 & V(k) > -5
        V(k)=V(k)^3;
    end
end
V =
   10  17 -27   8   0  -7  24  30  40  -6   12   4
   -8  16
```
6.4.2 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 until a stated condition is satisfied.

```
while conditional expression
    .......
    .......
    .......
    A group of MATLAB commands.
end
```

Figure 6-6: The structure of a while-end loop.
6.4.2 while-end Loops

- **For a while-end loop to execute properly:**
  - The conditional expression in the while command must include at least one variable.
  - The variables in the conditional expression must have assigned values when MATLAB executes the while command for the first time.
  - At least one of the variables in the conditional expression must be assigned a new value in the commands that are between the `while` and the `end`. Otherwise, once the looping starts it will never stop since the conditional expression will remain true.
6.4.2 while-end Loops

\[
x = 1 \quad \text{Initial value of } x \text{ is 1.}
\]

while \( x <= 15 \) \[
\begin{align*}
x &= 2 \times x \\
end
\end{align*}
\]

The next command is executed only if \( x <= 15 \).

In each pass \( x \) doubles.

When this program is executed the display in the Command Window is:

\[
x = 1 \quad \text{Initial value of } x. \\
x = 2 \quad \text{In each pass } x \text{ doubles.} \\
x = 4 \\
x = 8 \\
x = 16 \quad \text{When } x = 16, \text{ the conditional expression in the} \\
\text{while command is false and the looping stops.}
\]
The function $f(x) = e^x$ can be represented in a Taylor series by 
$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!}.$$ 

Write a program in a script file that determines $e^x$ by using the Taylor series representation. The program calculates $e^x$ by adding terms of the series and stopping when the absolute value of the term that was added last is smaller than 0.0001. Use a while-end loop, but limit the number of passes to 30. If in the 30th pass the value of the term that is added is not smaller than 0.0001, the program stops and displays a message that more than 30 terms are needed.

Use the program to calculate $e^2$, $e^{-4}$, and $e^{21}$.

The first few terms of the Taylor series are: 
$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots$$
x=input('Enter x ');

n=1; an=1; S=an;

while abs(an) >= 0.0001 & n <= 30
    an=x^n/factorial(n);
    S=S+an;
    n=n+1;
end

if n >= 30
    disp('More than 30 terms are needed')
else
    fprintf('exp(%f) = %f',x,S)
    fprintf('
The number of terms used is: %i',n)
end
Sample Problem 6-7: Taylor series representation of a function

6.4.2 while-end Loops

Enter x 2
\[ \exp(2.000000) = 7.389046 \]
The number of terms used is: 12
>> expox

Enter x -4
\[ \exp(-4.000000) = 0.018307 \]
The number of terms used is: 18
>> expox

Enter x 21

More than 30 terms are needed
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.

```
for k = 1:n
    for h = 1:m
        ........
        ........
        ........
        A group of commands.
    end
end
```

Every time \( k \) increases by 1, the nested loop executes \( m \) times. Overall, the group of commands are executed \( n \times m \) times.

Figure 6-7: Structure of nested loops.
Sample Problem 6-8: Creating a matrix with a loop

Write a program in a script file that creates an $n \times m$ matrix with elements that have the following values:

- The value of each element in the first row is the number of the column.
- The value of each element in the first column is the number of the row.
- The rest of the elements each has a value equal to the sum of the element above it and the element to the left.

When executed, the program asks the user to enter values for $n$ and $m$.

**Solution:** The program, shown next, has two loops (one nested) and a nested if-elseif-else-end structure. The elements in the matrix are assigned values row by row. The loop index variable of the first loop, $k$, is the address of the row, and the loop index variable of the second loop, $h$, is the address of the column.
n=input('Enter the number of rows ');
m=input('Enter the number of columns ');
A=[];
for k=1:n
    for h=1:m
        if k==1
            A(k,h)=h;  
        elseif h==1
            A(k,h)=k; 
        else
            A(k,h)=A(k,h-1)+A(k-1,h); 
        end
    end
end
A
6.5 NESTED LOOPS AND NESTED CONDITIONAL STATEMENTS

The program is executed in the Command Window to create a 4x5 matrix:

```
>> Chap6_exp8
Enter the number of rows 4
Enter the number of columns 5

A =

1  2  3  4  5
2  4  7 11 16
3  7 14 25 41
4 11 25 50 91
```
The **break** command:

- 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).

- If the `break` command is inside a nested loop, only the nested loop is terminated.

- When a `break` command appears outside a loop in a script or function file, it terminates the execution of the file.

- 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 — for example, if the number of loops exceeds a predetermined value.
The **continue** command:

- 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.
- 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.