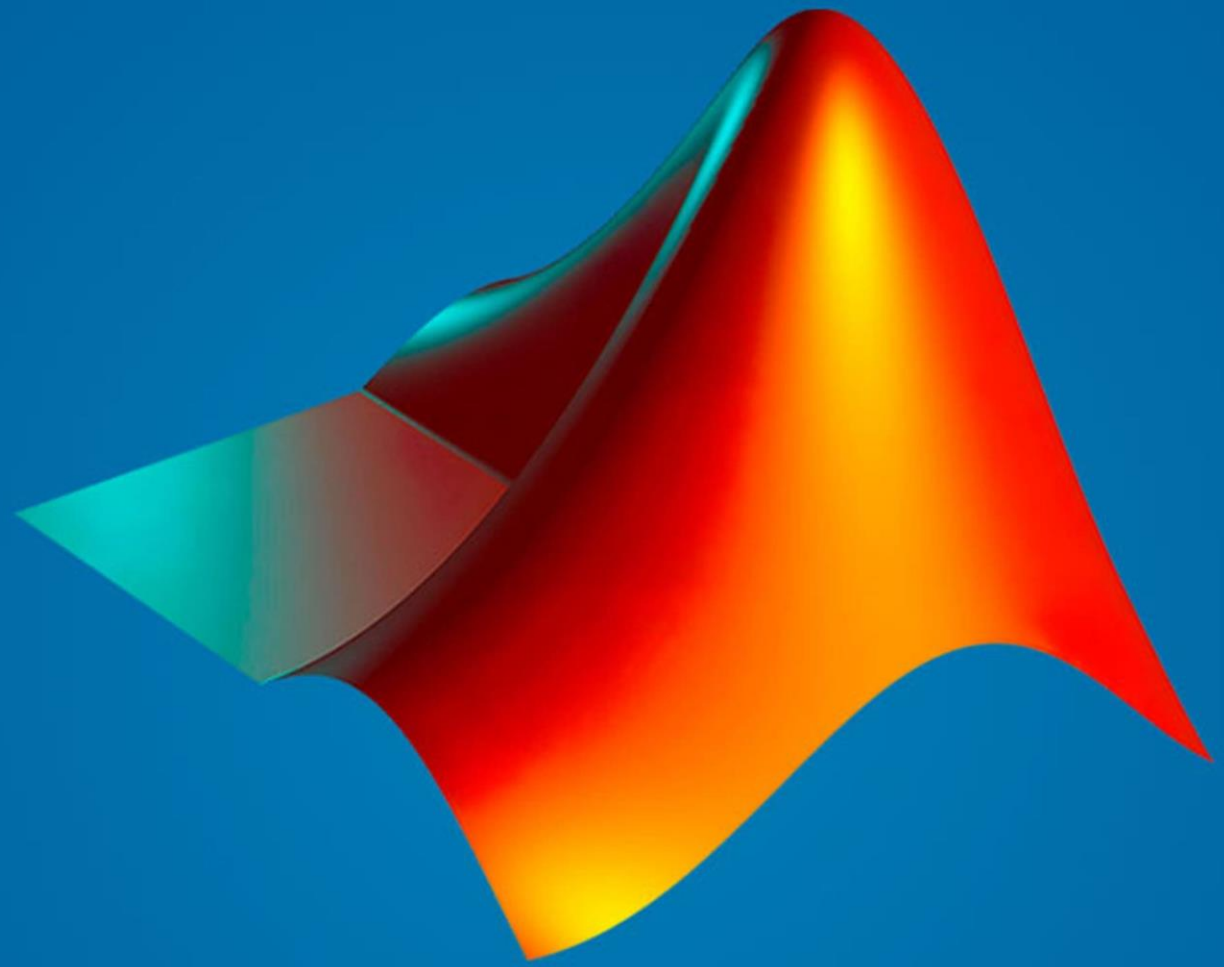


MATLAB

CONTROL STATEMENTS

The if statement allows conditional execution based on a condition.

```
if condition
    % code to execute if condition is true
elseif another_condition
    % code to execute if another_condition is true
else
    % code to execute if none of the above conditions are true
end
```

```
x = 5;
if x > 0  %The condition is checked first.
    disp('x is positive'); %If true, it prints "x is positive".
elseif x < 0 %If true, it prints "x is negative".
    disp('x is negative');
else   %If neither is true, it prints "x is zero".
    disp('x is zero');
end
```

Output

x is positive

LOOPS

MATLAB provides several types of loops, including for loop and while loop.

```
for index = start_value:end_value %The loop starts with the keyword
    % and followed by a iterable variable.
    % code to execute in each iteration
end %the loop end with end keyword
```

```
% For loop example
for i = 1:5  %The loop starts with the keyword and followed by a loop
             %variable. i=1:5 means that i will take values from 1 to 5.
    disp(i);  % disp() function prints the current value of i
end
```

Output

```
1
2
3
4
5
```

LOOPS

MATLAB provides several types of loops, including for loop and while loop.

```
while condition
    % code to execute while condition is true
end
```

```
% While loop example
i = 1;
while i <= 5  %The loop starts with the keyword and followed by condition
              % in this case it is i<=5.
    disp(i);  % Display the value of i
    i = i + 1;  % Increment i
end
```

Output

```
1
2
3
4
5
```

FUNCTIONS

A function in MATLAB is defined using the keyword function, followed by the output variables, the function name, and the input arguments.

```
function [output1, output2, ..., outputN] = functionName(input1, input2, ..., inputM)
    % Function description (optional comment)

    % Code to compute the outputs based on the inputs
    % Example:
    output1 = input1 + input2;

    % Return the outputs
end
```

FUNCTIONS

Here's a simple example that defines a function to calculate the area of a rectangle:

```
% This function(rectangleArea) calculates the area of a rectangle.
% We create the function like this way,
% or we can save it in a different file called rectangleArea

function area = rectangleArea(length, width)  %The function (rectangleArea)
% takes two inputs(length, width) and returns output area
    area = length * width; %Function operation
end

area=rectangleArea(4,5); %Function call
disp(area) % disp() function, print the area
```

FUNCTIONS

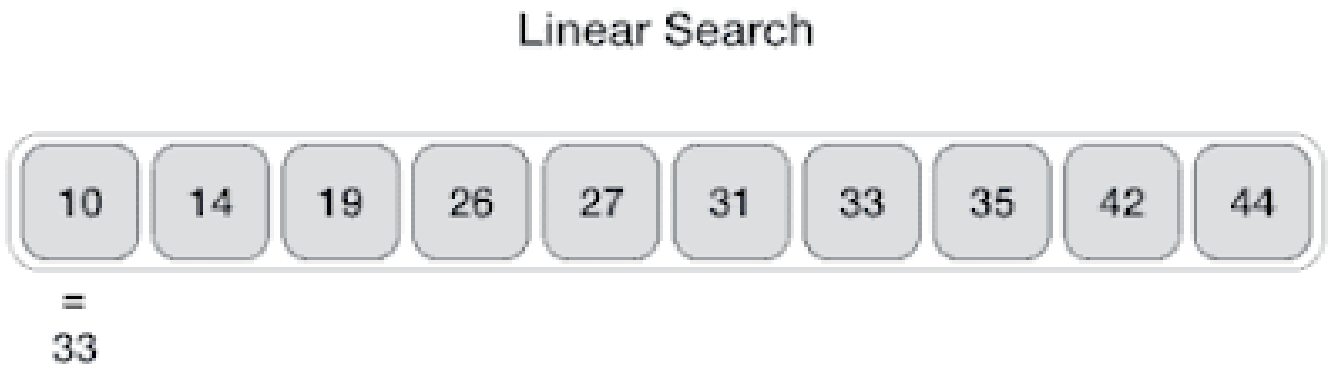
```
%Multiple Outputs Example.
% This function calculates both the area and the perimeter of a rectangle
%It has two inputs (length, width) and two outputs(area, perimeter)
function [area, perimeter] = rectangleProps(length, width)
    area = length * width;
    perimeter = 2 * (length + width);
end

%Calling function
[area, perimeter] = rectangleProps(5, 10);

disp(area); %print area
disp(perimeter); %print perimeter
```

Successive Search

Successive Search, also known as Linear Search or Sequential Search, is a basic search algorithm used to find the location of a specific element in a list or array by checking each element one by one.



Steps of the SS:

- Start from the first element of the array.
- Compare each element to the target value.
- If a match is found, return the index or position.
- If the target is not found, continue until the last element.

```
% This function performs a linear search for the target in the array arr.
% If found, it returns the index; otherwise, it returns -1.
function index = successive_search(arr, target)
    % Initialize index to -1, which means the target is not found
    index = -1;
    % Loop through each element of the array
    for i = 1:length(arr) % length(arr) Gets the length of the array
        if arr(i) == target %i. value equals to target
            index = i; % If found, store the index
            break;     % break the loop because the target is found
        end
    end
end

% Example usage:
array = [15, 3, 8, 23, 42, 7, 50, 45, 1, -10, -5];
targetValue = -10;

% Call the successive_search function
resultIndex = successive_search(array, targetValue);

% Display the result
if resultIndex == -1
    fprintf('Element not found in the array.\n');
else
    fprintf('Element found at index: %d\n', resultIndex);
end
```

Struct

Struct is a data type that allows you to group related data using named fields.

Each field in a structure can contain data of varying types and sizes.

Fields in a structure are accessed by their names.

Structures can contain different data types in different fields, such as arrays, matrices, strings, cells or even other structures.

Struct

```
%struct
%You can create a structure in MATLAB using the dot notation to define fields.
%This creates an empty person struct.
person.fname=[]; %First field fname
person.lname=[]; %Second field lname
person.grades={}; %Third field grades, it's a cell array.
person.totalgrade=[]; %Last field totalgrade
```

Struct

```
%Assigning Values to the Structure:
%first element of person (person(1))
person(1).fname="Elif";
person(1).lname="Elif";
person(1).grades{1}=95; %grades{1} stores Midterm grade
person(1).grades{2}=95; %grades{2} stores Final grade
%totalgrade is still empty, because nothing assigned


%Second elements of person (person(2))
person(2).fname="Emre";
person(2).lname="Emre";
person(2).grades{1}=85; %Midterm grade
person(2).grades{2}=95; %Final grade
%totalgrade is still empty, because nothing assigned
```

Struct

```
%display the first student's(person(1)) details,
%including the midterm and final grades:
%In disp() function to combine text and variables, we need to
% convert numeric values to strings before using disp.
%num2str() convert numeric value to string
 disp(['Name: ', person(1).fname]);
 disp(['Last name: ', person(1).lname]);
 disp(['Midterm Grade: ', num2str(person(1).grades{1})]);
 disp(['Final Grade: ', num2str(person(1).grades{2})]);
```

Struct

```
%Calculating and Displaying the Total Grade for Each Student:
for i=1:length(person) %length(person) gets the size of struct
    % Calculate total grade with 40% weight for midterm and 60% for final
    totalGrade=((person(i).grades{1}*40)/100)+((person(i).grades{2}*60)/100);
    %display the details
    disp(['Name: ', person(i).fname]);
    disp(['Last name: ', person(i).lname]);
    person(i).totalgrade=totalGrade;
    disp(['Total Grade: ', num2str(person(i).totalgrade)]);
end
```

Struct

```
%We can add new fields to an existing structure
%by assigning a value to a new field name:
%Adding gpa field to the person structure
person(1).gpa=3.51;
person(2).gpa=3.90;
```

```
%To remove a field from a structure, we can use the rmfield function
% Removes the field 'totalgrade' from the structure
person = rmfield(person, 'totalgrade');
```

%rmfield output must be assigend back to the structure

Struct

```
%We can extract data from the same field across all elements of a
% structure array using the dot notation with array indexing.

% Extracting the 'gpa' field from all structures in the array
all_gbas = [person.gpa];  % Returns a numeric array

% Extracting the 'grades' field from all structures
all_grades = {person.grades};  % Returns a cell array
```