Computing and Software Engineering

GET211

September 23, 2025

Structured Programming

- A programming paradigm emphasizing clarity, logic, and modularity.
- Introduced in the 1960s to replace unstructured, ‘goto‘-heavy code.
- A foundational paradigm for robust, scalable, and maintainable code.
- Influenced modern programming paradigms like OOP and functional programming.
- Promotes discipline in software development.

Core Principles of Structured Programming

1. **Sequence:** Linear execution of statements.
2. **Selection:** Conditional branching using structures like `if-else`.
3. **Iteration:** Loops for repetition (`for`, `while`).
4. **Modularity:** Divide programs into reusable functions or procedures.
5. **Single Entry, Single Exit:** Simplified control structures.

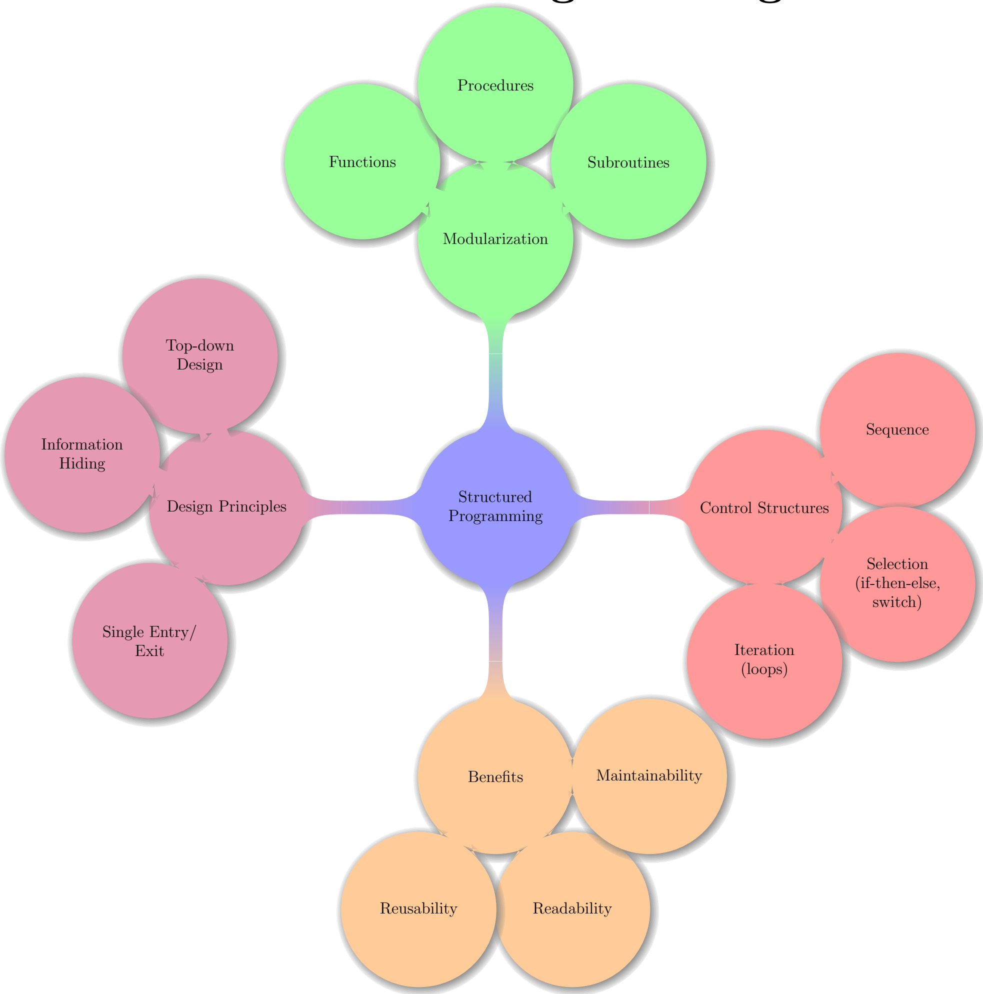
Advantages of Structured Programming

- **Readability:** Clear and logical organization of code.
- **Maintainability:** Modular design simplifies debugging and updates.
- **Reusability:** Code components can be reused in other projects.
- **Scalability:** Easy to expand programs.
- **Efficiency:** Optimized for execution with fewer errors.

Structured vs. Unstructured Programming

Aspect	**Structured Programming**	**Unstructured Programming**
Flow Control	Clear logic (loops, conditionals)	Ad-hoc (`goto` statements)
Readability	Easy to understand	Hard to follow
Maintenance	Modular and scalable	Difficult to update
Error Prone	Lower risk	High risk

Structured Programming



Function

Functions are reusable blocks of code that perform specific tasks. They enhance code organization, modularity, and efficiency in MATLAB.

Basic Structure of a Function

A MATLAB function typically consists of:

Function header: Specifies the function name, input arguments, and output arguments.

Function body: Contains the code that performs the desired computations.

```matlab
function [output1, output2, ...] = function_name(input1, input2, ...)
    % Function body
    % ...
end
```

Creating a Function

- **Create a new file:** Save the function code in a separate file with the same name as the function. For example:
  If the function is called **addNumbers**, the file must be saved as **addNumbers.m**.
  If the function is **greet**, save it as **greet.m**.
- **Define inputs and outputs:** Specify the variables that the function will receive (inputs) and return (outputs).
- **Write function body:** Implement the logic for the function's task.
- **Saving and Locating Function Files:** Save the file with the '.m' extension.
  In the same directory:
  Save the .m file in your current working directory. MATLAB will automatically recognize it.
  In a different directory:
  You can either: Add the directory to MATLAB's search path using the addpath command:

```
addpath('C:/path_to_directory')
```

Or navigate to the function's directory using cd:

```
cd 'C:/path_to_directory'
```

Function Inputs and Outputs

Input Arguments
Functions can accept multiple inputs:

```
function result = multiply(a, b)
    result = a * b;
end
```

Output Arguments Functions can return multiple outputs:

```
function [sum, product] = computeValues(a, b)
    sum = a + b;
    product = a * b;
end

[s, p] = computeValues(4, 5);
```

Default Input Values

MATLAB functions can also have default input values. Here's an example:

```
function [sum_result, diff_result] = calc_sum_diff(a,
    b)
    % Function that calculates the sum and difference
    if nargin < 2
        b = 0;  % Default value for b if not provided
    end
    sum_result = a + b;
    diff_result = a - b;
end
```

Variable Input Arguments (varargin)

MATLAB allows functions to accept a variable number of input arguments using the `varargin` keyword. Here's an example that sums all input numbers:

```
function total = sum_all_numbers(varargin)
    total = 0;
    for i = 1:length(varargin)
        total = total + varargin{i};
    end
end
```

Calling the Function with Variable Inputs

You can call the function with any number of arguments:

```
result1 = sum_all_numbers(1, 2, 3, 4);
result2 = sum_all_numbers(5, 5);
```

Variable Output Arguments (varargout)

MATLAB functions can return a variable number of output arguments using **`varargout`**. Example where the function returns the input value, its square, and its cube based on the requested outputs:

```
function varargout = power_values(x)
    varargout{1} = x;
    if nargout > 1
        varargout{2} = x^2;
    end
    if nargout > 2
        varargout{3} = x^3;
    end
end
```

Calling the Function with Variable Outputs

You can call the function and request different numbers of output arguments:

```
[output1] = power_values(3);
[output1, output2] = power_values(3);
[output1, output2, output3] = power_values(3);
```

Function

Types of Functions in MATLAB

User-Defined Functions

Custom functions written by the user and saved as ‘.m‘ files.

Example:

```
% File: addNumbers.m
function result = addNumbers(a, b)
    result = a + b;
end
```

Calling the function:

```
sum = addNumbers(5, 3);
```

Function Example

Create a function to calculate the area of a circle and name circle_area in a script:

```
function area = circle_area(radius)
    area = pi * radius^2;
end
```

Calling a Function

To use a function, you call it by its name and provide the necessary input arguments:

```
radius = 5;
result = circle_area(radius);
disp(result);
```

Anonymous Functions

An **anonymous function** in MATLAB is a function that is defined inline, without needing a separate function file. It is often used for short, simple operations.

Syntax:

```
f = @(input_arguments) expression
```

Where:

- `f` is the function handle (variable that stores the function).
- `input_arguments` are the inputs to the function.
- `expression` is the code that defines the operation performed by the function.

Anonymous Functions Examples

A simple example of an anonymous function that squares a number:

```
f = @(x) x^2;
result = f(4);
```

Here, `f` is a function that squares the input `x`.

Multiple Inputs: Anonymous functions can also take multiple inputs:

```
sumFunc = @(a, b) a + b;
result = sumFunc(3, 5);
```

The function `sumFunc` takes two inputs `a` and `b`, and returns their sum.

Multiple Outputs: You can also define anonymous functions with multiple outputs using `deal`:

```
multiOutputFunc = @(x) deal(x^2, x^3);
[square, cube] = multiOutputFunc(2);
```

This anonymous function returns both the square and the cube of the input `x`.

Function

Built-in Functions MATLAB provides a library of predefined functions like ‘sin()‘, ‘mean()‘, ‘plot()‘, etc. Example:

```
A = [1, 2, 3, 4];
average = mean(A);
```

MATLAB provides a wide variety of built-in functions that help in performing mathematical, statistical, and data manipulation tasks efficiently.

Some common built-in functions:

Category	**Function Name**	**Description**

Mathematical Functions	`sin(x)`
	`cos(x)`
	`sqrt(x)`
		Sine of angle
		Cosine of angle
		Square root of x

Statistical Functions	`mean(x)`
	`median(x)`
	`std(x)`
		Mean of array
		Median of array
		Standard deviation of array

Matrix Operations	`inv(A)`
	`eig(A)`
	`det(A)`
		Inverse of matrix A
		Eigenvalues and eigenvectors of A
		Determinant of matrix A

Plotting Functions	`plot(x, y)`
	`scatter(x, y)`
	`surf(X, Y, Z)`
		2D line plot
		Scatter plot
		3D surface plot

File I/O	`save(filename, data)`
	`load(filename)`
	`fopen(file)`
		Save data to file
		Load data from file
		Open a file for reading/writing

Utility Functions	`size(A)`
	`length(A)`
	`isnan(A)`
	`isempty(A)`
		Size of matrix A
		Length of vector A
		Check for NaN values
		Check if matrix A is empty

Nested Functions

Nested functions in MATLAB are functions defined inside other functions. They allow for:

- Access to the variables of the parent (outer) function.
- Cleaner, more organized code for logically related tasks.
- Passing inner functions as function handles to other functions.

```
function outer_function()
    % Code for outer function

    % Nested function
    function inner_function()
        % Code for inner function
    end
end
```

Example of Nested Functions

Calculates the sum of squares using a nested function:

```matlab
function sum_of_squares(arr)
    total = 0;

    for i = 1:length(arr)
        total = total + square(arr(i));  % Call the nested function
    end

    disp(['Sum of squares: ', num2str(total)]);  % Display result

    % Nested function
    function sq = square(x)
        sq = x^2;  % Return square of x
    end
end
```

Function

Scope and Workspace

- Local Variables: Variables inside functions are local and do not affect the base workspace.
- Global Variables: Declared with 'global' to share across functions.

```
global x;
x = 5;
```

- Persistent Variables: Retain their value between function calls.

```
function counter = increment()
    persistent count;
    if isempty(count)
        count = 0;
    end
    count = count + 1;
    counter = count;
end
```

Recursive Functions

A recursive function is a function that calls itself during its execution. Recursion is used when a problem can be broken down into smaller instances of the same problem.

- Base case: The simplest case for which the function doesn't call itself.
- Recursive case: The function calls itself with modified parameters.

Example: Calculating the factorial of a number.

Recursive Function

The factorial of a number n, denoted as $n!$, is defined as:

$$n! = n \times (n-1) \times (n-2) \times \cdots \times 1$$

For $n = 0$, $0! = 1$ (base case). The recursive definition of factorial:

$$n! = n \times (n-1)!$$

```matlab
function result = factorial_recursive(n)
    % Base case
    if n == 0 || n == 1
        result = 1;
    else
        % Recursive case
        result = n * factorial_recursive(n - 1);
    end
end
```

```matlab
result = factorial_recursive(5);
disp(['Factorial of 5 is: ', num2str(result)]);
```

Exercise

Write a function to check if a number is even.

Write a function to find the roots of a quadratic equation $ax^2 + bx + c = 0$. Use the quadratic formula:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Where:
- a, b, and c are the coefficients of the quadratic equation.
- The expression $b^2 - 4ac$ is called the discriminant and determines the nature of the roots.

Hints
1. Compute the discriminant: $\Delta = b^2 - 4ac$
- If $\Delta > 0$, the equation has two real and distinct roots.
- If $\Delta = 0$, the equation has exactly one real root (a repeated root).
- If $\Delta < 0$, the equation has two complex (imaginary) roots.
2. Calculate the roots using the quadratic formula:
- If the discriminant is non-negative, the roots are real and can be calculated using the quadratic formula.
- If the discriminant is negative, the roots will involve imaginary numbers.

Test:
Using the code you wrote, find the roots of the equation:
- $x^2 - 3x + 2 = 0$.
- $x^2 + 2x + 1 = 0$.
- $x^2 + 2x + 5 = 0$.

Exercise

Create a function to determine if a given number is prime. A prime number is a natural number greater than 1 that has no positive divisors other than 1 and itself.

Exercise

Create a MATLAB function that calculates the n-th Fibonacci number using a recursive approach.
The Fibonacci sequence is defined as follows:

$$F(0) = 0, \quad F(1) = 1$$

For $n \geq 2$:

$$F(n) = F(n-1) + F(n-2)$$

The function should:
1. Take an integer n as input, where $n \geq 0$.
2. Return the n-th Fibonacci number.
3. Handle base cases: $F(0) = 0$ and $F(1) = 1$.

Exercise

Create a MATLAB function that approximates the sine of an angle x using the Taylor series expansion. The Taylor series for $\sin(x)$ is:

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots$$

For this exercise, you will calculate the sine of x using the first N terms of the Taylor series expansion.

Steps:

1. Input: The function will take two inputs: x (the angle in radians) and N (the number of terms to use in the Taylor series).
2. Output: The function will return the approximation of $\sin(x)$.
3. Process: You will use the following series expansion for $\sin(x)$:

$$\sin(x) \approx x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots$$

You can write a loop to calculate the sum of the first N terms of this series. Compare the result of your function and the built-in MATLAB function.

Exercise

Write a MATLAB function called statistical_operations that accepts a list of numbers as input and performs the following tasks:
1. Sum: Calculate the sum of the input numbers.
2. Mean: Calculate the mean (average) of the input numbers.
3. Median: Calculate the median of the input numbers.
4. Standard Deviation: Calculate the standard deviation of the input numbers.
5. Maximum and Minimum: Find the maximum and minimum values in the input list.

The function should return these values as output. The function should also handle the following:
- If only one output is requested, return the sum of the numbers.
- If two outputs are requested, return the sum and mean of the numbers.
- If three outputs are requested, return the sum, mean, and median.
- If four outputs are requested, return the sum, mean, median, and standard deviation.
- If five outputs are requested, return all five values: sum, mean, median, standard deviation, and the maximum and minimum values.

Hints:
1. Use ‘nargin‘ to determine how many outputs are requested.
2. Use MATLAB’s built-in functions: sum(), mean(), median(), std(), min() and max() for minimum and maximum.
3. Use ‘varargin‘ to accept variable input arguments.
4. If no input is provided, return an error message.