Introduction to Computer Programming

- Week 7

- *Eng. Sylvain Manirakiza* -

Sun - Nov 9th, 2025

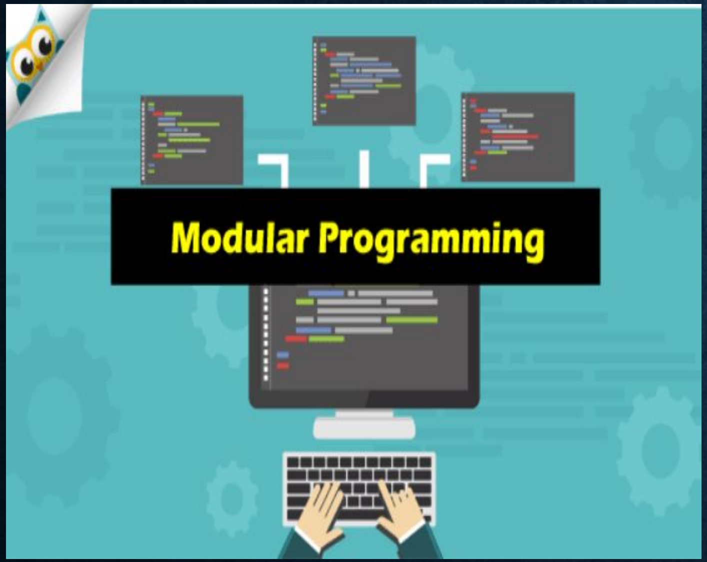
Chap 4. Modular Programming

- Week 7

- Eng. Sylvain Manirakiza -

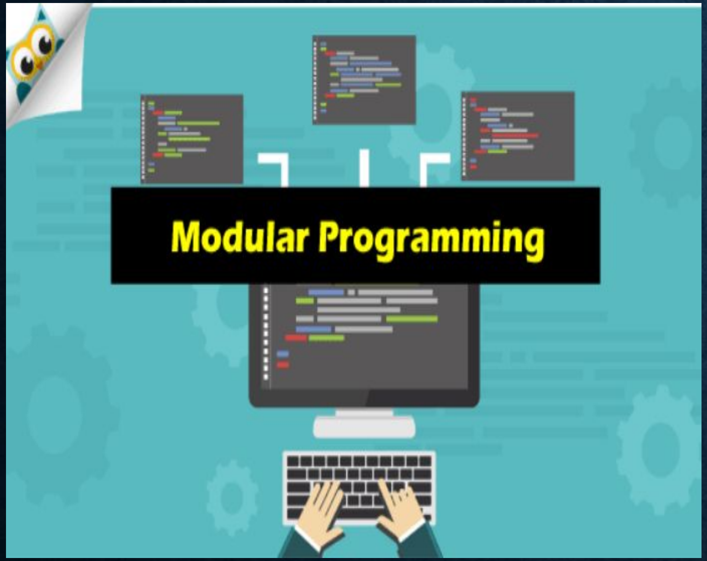
Sun - Nov 9th, 2025

Modular Programming



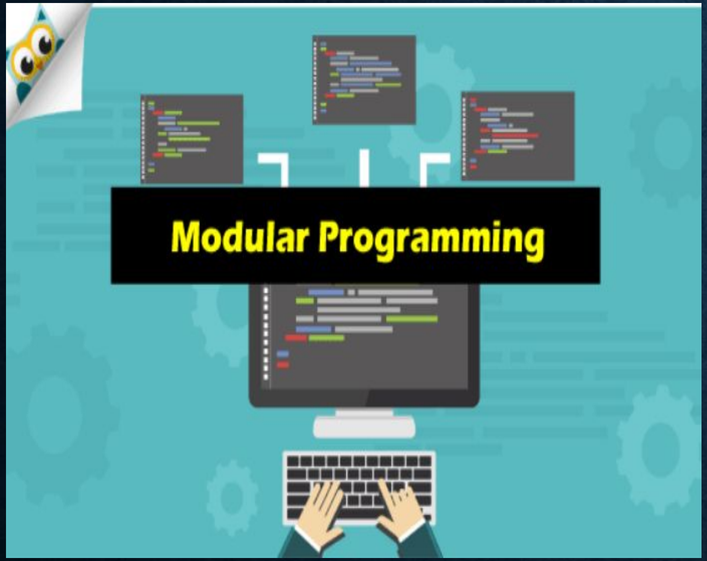
The concept of modular programming originated in the 1960s to help users. Programmers began to divide the more extensive programs into smaller parts. Though the concept of modular programming is decades old, it is the most convenient programming method.

Definition



Modular programming is defined as a software design technique that focuses on separating the program functionality into independent, interchangeable methods/modules. Each of them contains everything needed to execute only one aspect of functionality.

Definition

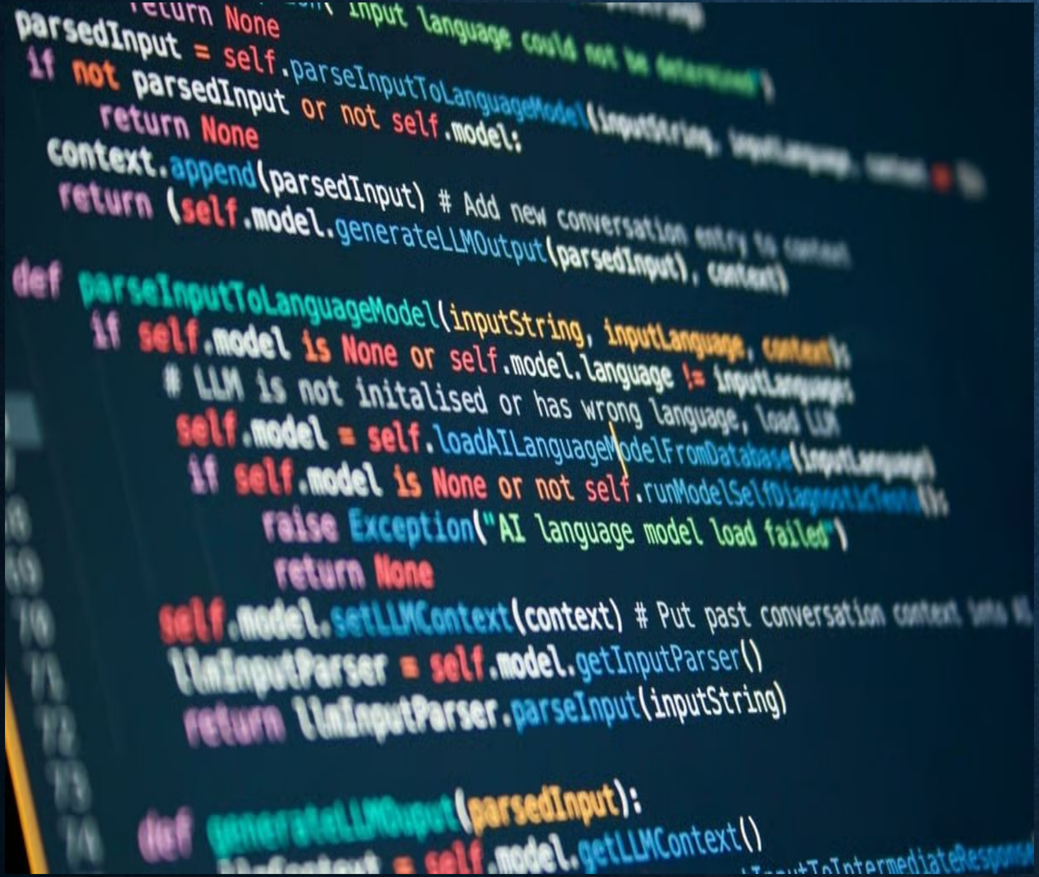


- Modular Programming is a programming approach that breaks a large program into smaller, reusable modules or functions.
- Each module performs a specific task and works independently.

Modular Programming

Modularity is all about making blocks, and each block is made with the help of other blocks. Every block in itself is solid and testable and can be stacked together to create an entire application. Therefore, thinking about the concept of modularity is also like building the whole architecture of the application.

Examples of modular programming languages



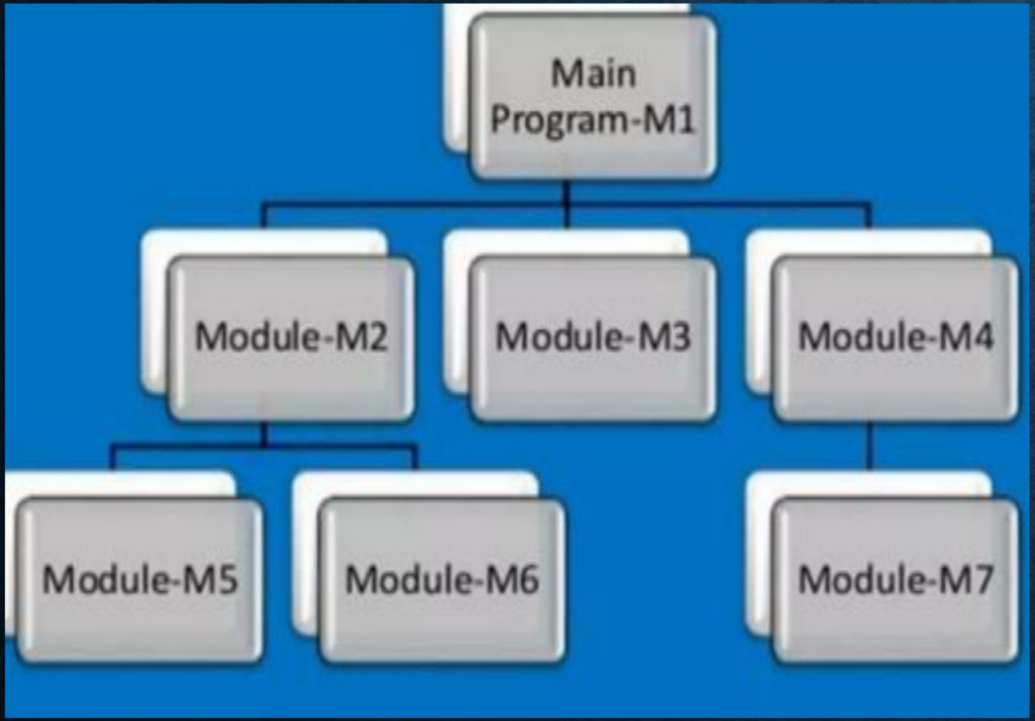
```
                                    input language could not be determined")
        return None
    parsedInput = self.parseInputToLanguageModel(inputString, inputLanguage, context)
    if not parsedInput or not self.model:
        return None
    context.append(parsedInput) # Add new conversation entry to context
    return (self.model.generateLLMOutput(parsedInput), context)

def parseInputToLanguageModel(inputString, inputLanguage, context):
    if self.model is None or self.model.language != inputLanguage:
        # LLM is not initalised or has wrong language, load LLM
        self.model = self.loadAILanguageModelFromDatabase(inputLanguage)
        if self.model is None or not self.runModelSelfDiagnosticChecks():
            raise Exception("AI language model load failed")
            return None
    self.model.setLLMContext(context) # Put past conversation context into
    llmInputParser = self.model.getInputParser()
    return llmInputParser.parseInput(inputString)

def generateLLMOutput(parsedInput):
    llmContext = self.model.getLLMContext()
```

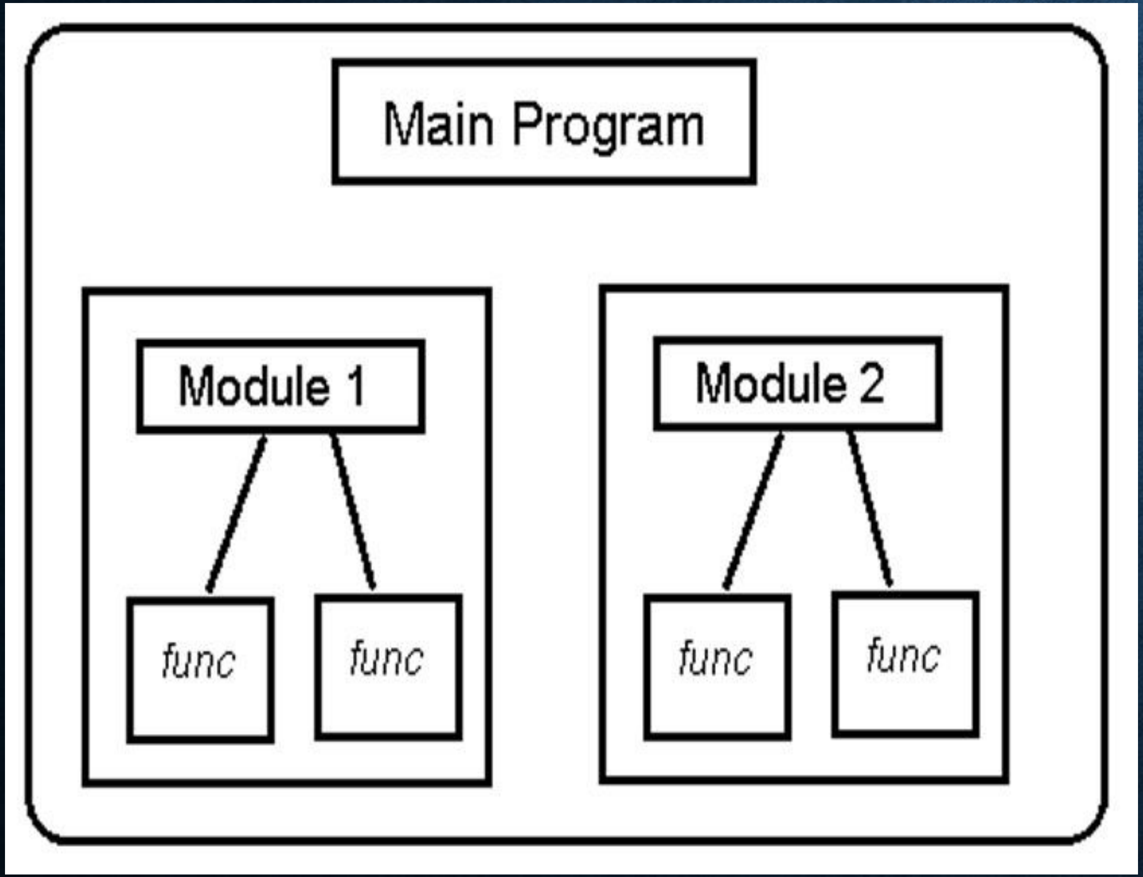
All the object-oriented programming languages like C++, Java, etc., are modular programming languages.

Structure of Modular Programming



1. Main Program → Calls different modules.
2. Predefined Functions → Built-in functions (e.g., print(), sqrt()).
3. User-Defined Functions → Custom functions created by the programmer.
4. Recursion → A function calling itself.

What is a Module?



A module is defined as a part of a software program that contains one or more routines. When we merge one or more modules, it makes up a program. Modules are implemented in the program through interfaces. The introduction of modularity allowed programmers to reuse prewritten code with new applications. Modules are created and merged with compilers, in which each module performs a business or routine operation within the program.

Example of Module

```
[Python]

import math  # Importing the math module

print(math.sqrt(25))  # Output: 5.0
print(math.pi)  # Output: 3.141592653589793
```

- math.sqrt(25): Uses the sqrt() function from the **math module** to find the square root of 25.
- math.pi: Uses the pi constant from the math module.

How to Declare a Function in Programming?

A function declaration is the first step in defining a function. It tells the program:

- The function name
- The parameters (inputs)
- The return type (if applicable)

General Function Declaration Format

```
Function function_name(parameter1, parameter2, ...)
  Statements (function body)
  Return (optional)
End Function
```

Example - Module Structure

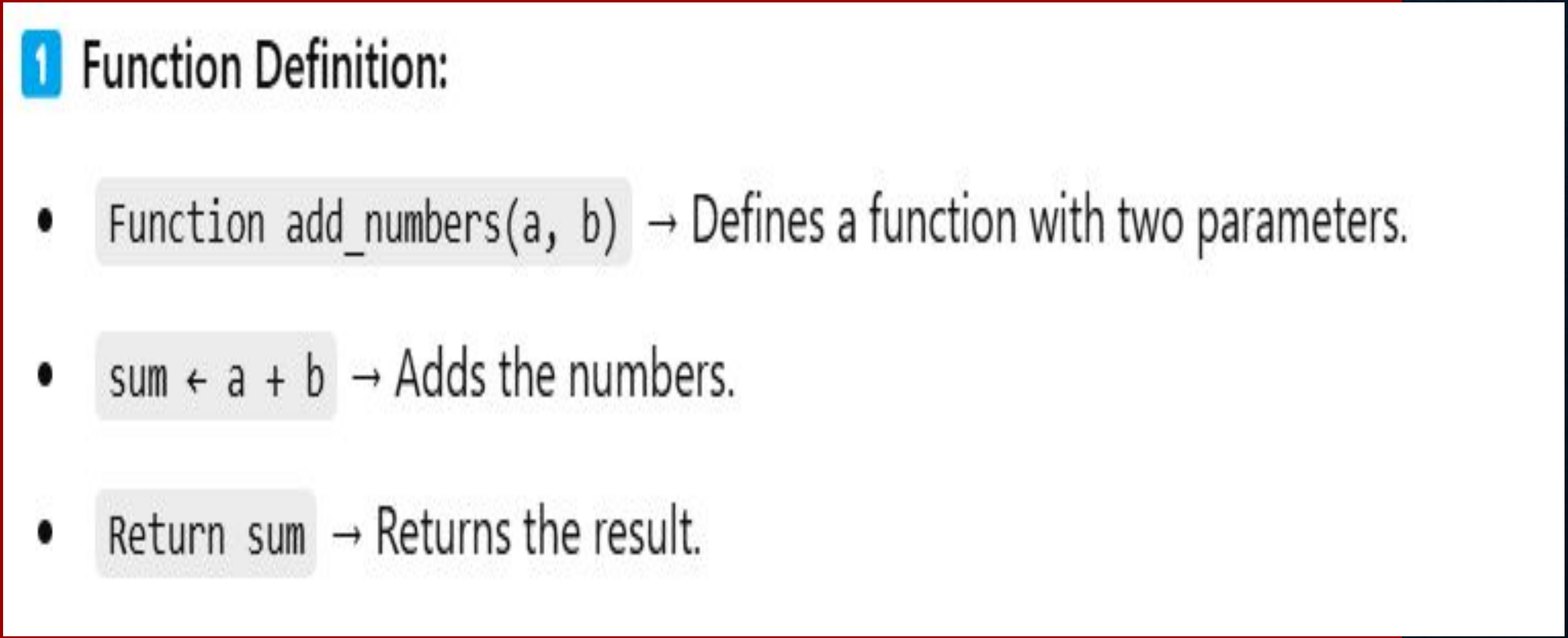
```
Function add_numbers(a, b)
  sum ← a + b
  Return sum
End Function

Start
  Write "Enter first number: "
  Read num1
  Write "Enter second number: "
  Read num2

  result ← add_numbers(num1, num2)

  Write "Sum: ", result
End
```

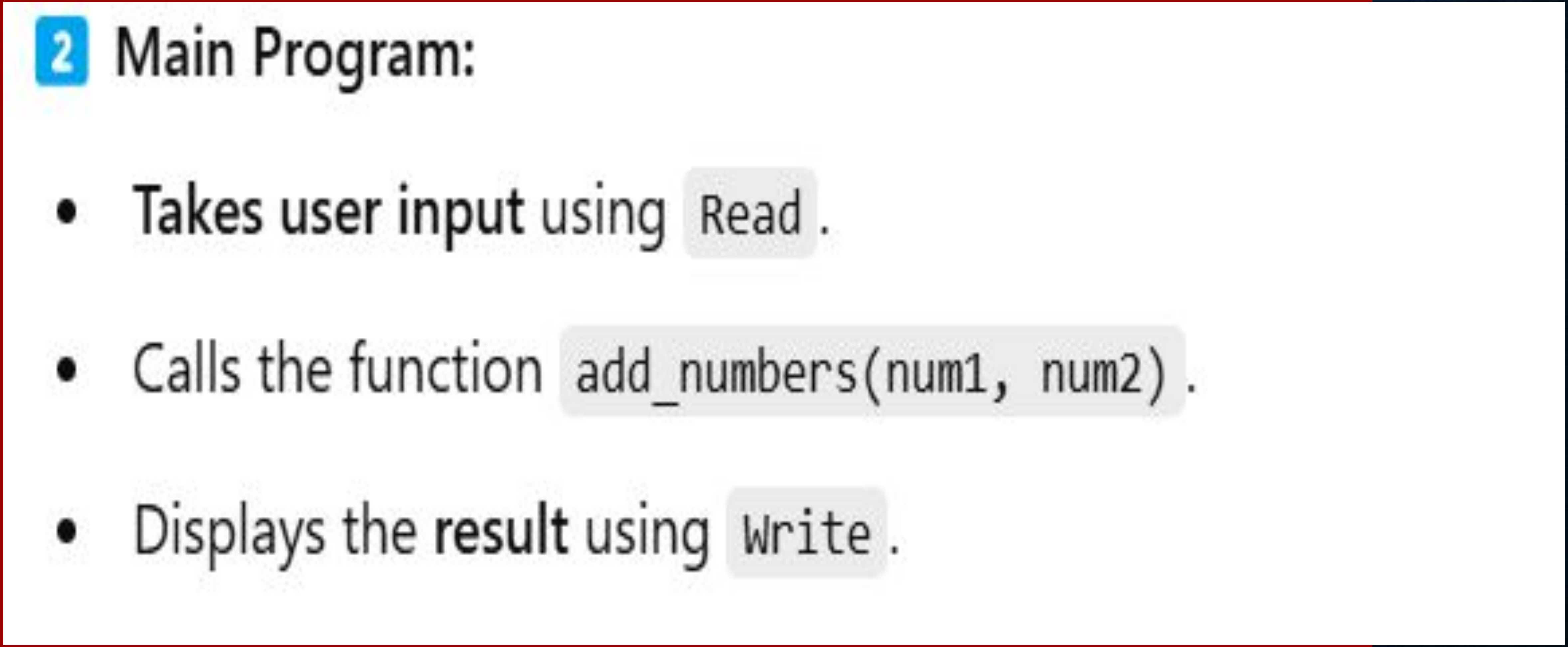
Structure Breakdown



1 Function Definition:

- `Function add_numbers(a, b)` → Defines a function with two parameters.
- `sum ← a + b` → Adds the numbers.
- `Return sum` → Returns the result.

Structure Breakdown



2 Main Program:

- **Takes user input** using `Read`.
- Calls the function `add_numbers(num1, num2)`.
- Displays the **result** using `Write`.

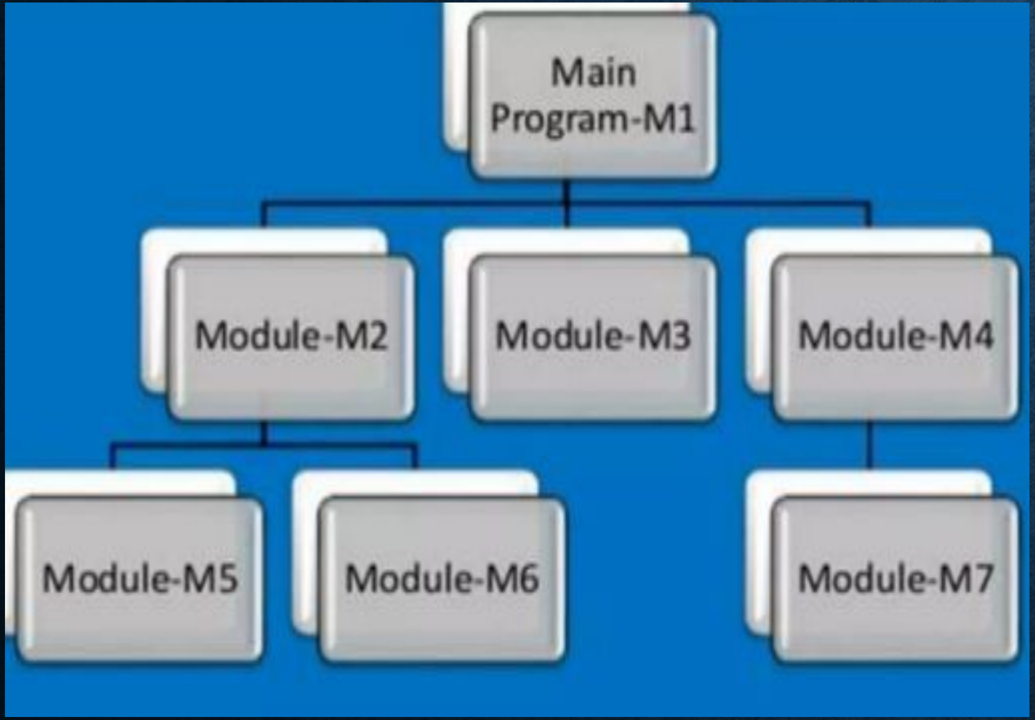
End goal

```
Enter first number: 5
Enter second number: 3
Sum: 8
```

Key Features of Modular Programming

- Divides a program into smaller parts
- Each module has a specific purpose
- Reduces code redundancy
- Easier debugging and maintenance

Advantages of Modular Programming



1. Improves Readability – Code is easier to understand.
2. Reusability – Functions can be used multiple times.
3. Easy Debugging – Errors are easier to track and fix.
4. Scalability – Code can be expanded easily.

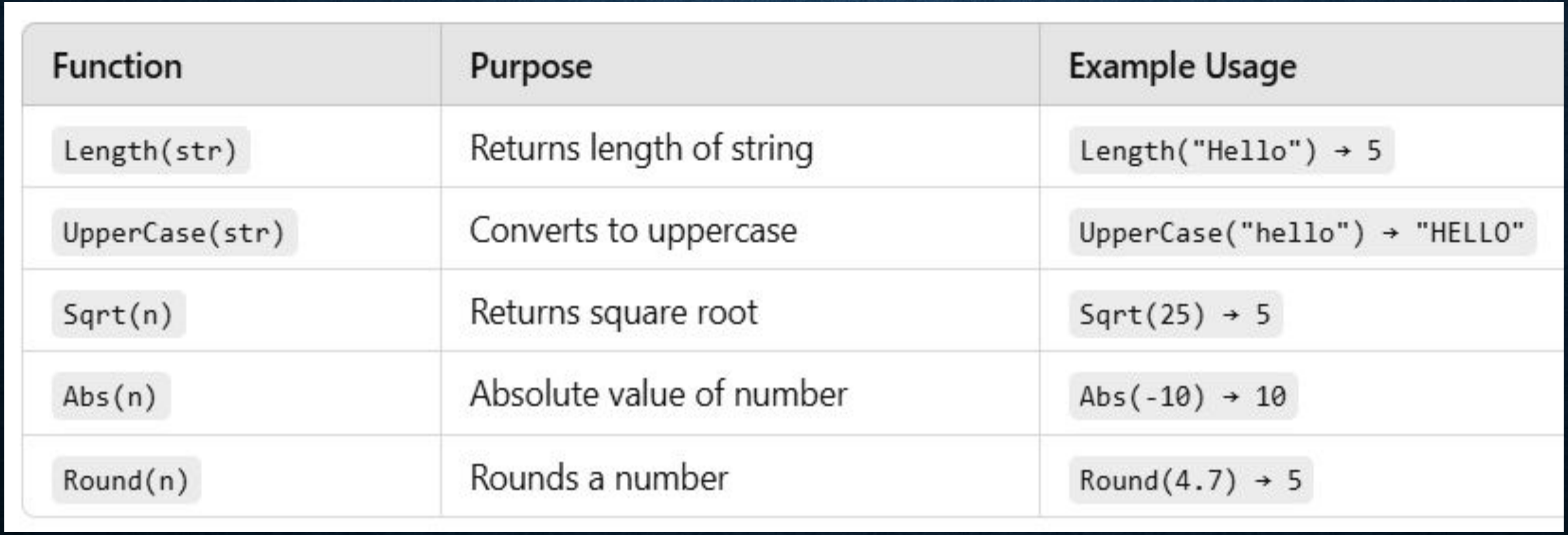
Applications/Real-World Examples:

1. Banking Systems → Separate modules for transactions, loans, and customer accounts.
2. E-commerce Websites → Different modules for orders, payments, and inventory.

1. Predefined Functions (Built-in Functions)

1. Definition Predefined functions are built-in functions available in a programming language.

2. Common Predefined Functions



Function	Purpose	Example Usage
`Length(str)`	Returns length of string	`Length("Hello") → 5`
`UpperCase(str)`	Converts to uppercase	`UpperCase("hello") → "HELLO"`
`Sqrt(n)`	Returns square root	`Sqrt(25) → 5`
`Abs(n)`	Absolute value of number	`Abs(-10) → 10`
`Round(n)`	Rounds a number	`Round(4.7) → 5`

Example: Using Predefined Functions

```
Start
  name ← "programming"
  Write UpperCase(name)  // Output: PROGRAMMING
  num ← -9
  Write Abs(num)         // Output: 9
  Write Sqrt(16)         // Output: 4
End
```

=> **Predefined functions simplify programming tasks.**

2. User-Defined Functions

1. Definition
   - User-defined functions are functions created by the programmer.
   - Can accept parameters & return values.

2. Example

```
Function multiply(a, b)
  Return a * b
End Function

Start
  num1 ← 4
  num2 ← 3
  result ← multiply(num1, num2)
  Write "Product: ", result
End
```

>> Custom functions reduce redundancy and make code efficient.

3. Recursion in Functions

1. Definition
   - Recursion occurs when a function calls itself.
   - Used in problems like factorial, Fibonacci, and tree traversal.

2. Example

```
Function factorial(n)
  If n = 1 Then
    Return 1
  Else
    Return n * factorial(n - 1)
  End If
End Function

Start
  num ← 5
  Write factorial(num)
End
```

Output: **120** **Since**

$5! = 5 \times 4 \times 3 \times 2 \times 1$

>> Recursion breaks complex problems into smaller subproblems.

Exercices

Exercices - Modular pseudocode

1. Write a Function to Calculate the Area of a Rectangle and displays the result.
2. Write a function is_even(n) that checks if a number is even. The main program should take input from the user and call the function.
3. Write a modular pseudocode program that converts Celsius to Fahrenheit using a function.
4. Write a recursive function that calculates the factorial of a number.
5. Write a function that takes three numbers and returns the largest.

Work on all questions as a group and submit concise answers using the assignment submission Form here - Before the next session!

End