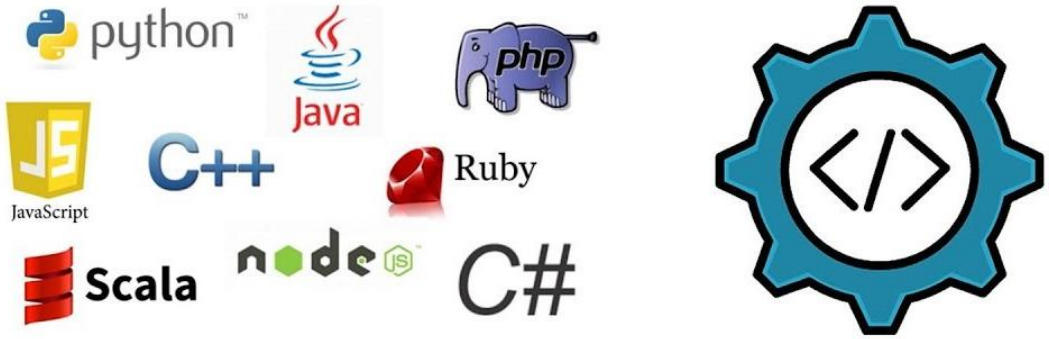
pseudocode



Prepared By - Nirmala Kumar Sahu

INDEX

Pseudocode

Introduction

- Pseudocode is a method of describing algorithms or problem-solving steps in a structured but informal way. It combines natural language with programming-like constructs to outline the logic and flow of a program or process without requiring strict syntax rules from any specific programming language.
- Pseudocode is a technique used to describe the distinct steps of an algorithm in a manner that's easy to understand for anyone with basic programming knowledge.

Key Characteristics of Pseudocode

- **Language-Independent:** It is not written in any particular programming language, making it accessible and easy to understand for anyone familiar with programming concepts.
- **Focuses on Logic:** Pseudocode emphasizes the logic and sequence of operations rather than syntax.
- **Readable:** It uses simple, plain English and basic constructs like IF, FOR, and WHILE.
- **Non-Executable:** Unlike actual code, pseudocode cannot be compiled or run on a computer.

Why Use Pseudocode?

- **Simplifies Complex Logic:** Helps break down problems into manageable steps.
- **Communication Tool:** Facilitates understanding among team members, even those not familiar with specific programming languages.
- **Planning:** Serves as a blueprint for writing actual code.
- **Debugging:** Allows testing of logic before implementation.

Example of Pseudocode

```
START
INPUT number
IF number % 2 == 0 THEN
   OUTPUT "The number is even."
ELSE
   OUTPUT "The number is odd."
ENDIF
END
```

Constructs of Pseudocode

The constructs of pseudocode represent the fundamental building blocks needed to describe algorithms. These constructs mirror core programming concepts but are expressed in a simplified and natural language format.

1. Input and Output: Used to represent data being received or displayed.

- **Input**: Represented using INPUT or READ.
  Syntax:

```
INPUT number
```

- **Output:** Represented using OUTPUT, PRINT, or DISPLAY.
  Syntax:

```
OUTPUT "The result is ", result
```

2. Variables and Assignment: Used to store and manipulate data.

- **Variable Declaration and Assignment:**
  Syntax:

```
SET sum = 0
```

- **Update Variable:**
  Syntax:

```
SET count = count + 1
```

3. Decision-Making (Conditional Statements): Used to handle branching logic.

- **IF-ELSE:**
  Syntax:

```
IF age >= 18 THEN
   OUTPUT "You are eligible to vote."
ELSE
   OUTPUT "You are not eligible to vote."
ENDIF
```

- **Nested IF:**
  Syntax:

```
IF score >= 90 THEN
   OUTPUT "Grade: A"
ELSE IF score >= 80 THEN
   OUTPUT "Grade: B"
ELSE
```

```
    OUTPUT "Grade: C"
ENDIF
```

4. Repetition (Loops): Used to perform repeated actions.

- **FOR Loop:**

  Syntax:

```
FOR i = 1 TO 10
    OUTPUT i
ENDFOR
```

- **WHILE Loop:**

  Syntax:

```
WHILE count < 5
    SET count = count + 1
ENDWHILE
```

5. Functions and Procedures: Used to represent reusable blocks of logic.

- **Function Definition:**

  Syntax:

```
FUNCTION addNumbers(a, b)
    RETURN a + b
ENDFUNCTION
```

- **Calling a Function:**

  Syntax:

```
SET result = addNumbers(5, 10)
```

6. Comments: Optional explanations to clarify the logic (helpful for complex algorithms).

Syntax:

```
// Calculate the factorial of a number
INPUT number
```

7. Flow Control: Defines the sequence and flow of the algorithm.

- **START and END:** Denote the beginning and end of the pseudocode.

  Syntax:

```
START
// Steps of the algorithm
END
```

Example Combining Constructs

Problem: Sum of First N Numbers

Solution:

```
START
INPUT N
SET sum = 0
FOR i = 1 TO N
   SET sum = sum + i
ENDFOR
OUTPUT "The sum is ", sum
END
```

Extra Constructs in Pseudocode

1. Error Handling: Used to manage unexpected or invalid inputs.

- **Try-Catch Equivalent:**

  Syntax:

```
TRY
   INPUT number
   IF number < 0 THEN
      THROW "Negative numbers are not allowed"
   ENDIF
CATCH error
   OUTPUT error
ENDTRY
```

2. Data Structures: Used for organizing and manipulating data effectively.

- **Lists/Arrays:**

  Syntax:

```
SET numbers = [1, 2, 3, 4, 5]
```

- **Adding to a List:**

  Syntax:

```
APPEND 6 TO numbers
```

- **Accessing Elements:**

  Syntax:

```
OUTPUT numbers[2]  // Outputs the third element
```

- **Dictionaries/Key-Value Pairs:**

  Syntax:

```
SET student = {"name": "Alice", "age": 20}
OUTPUT student["name"]  // Outputs "Alice"
```

3. Advanced Loops: Used for more complex iterations.

- **Do-While Loop (Ensures the block runs at least once):**

  Syntax:

```
DO
  INPUT number
WHILE number < 0
```

- **Foreach Loop (Iterating over a collection):**

  Syntax:

```
FOREACH item IN numbers
  OUTPUT item
ENDFOREACH
```

4. Parallel and Concurrent Processing: Used for algorithms that involve simultaneous execution.

- **Parallel Execution:**

  Syntax:

```
PARALLEL EXECUTE
  TASK 1: Read file
  TASK 2: Process data
  TASK 3: Write output
END PARALLEL
```

- **Wait for Tasks to Complete:**

  Syntax:

```
WAIT UNTIL all tasks are complete
```

5. Modularity and Reusability: Used to structure code into reusable comp.

- **Subroutines:**

  Syntax:

```
SUBROUTINE greetUser()
  OUTPUT "Hello, User!"
ENDSUBROUTINE
```

- **Calling Subroutines:**

  Syntax:

```
CALL greetUser()
```

6. Logical Operators: Used to simplify decision-making.

- **AND/OR/NOT:**

  Syntax:

```
IF age >= 18 AND hasID THEN
  OUTPUT "Access Granted"
ELSE
  OUTPUT "Access Denied"
ENDIF
```

7. Recursion: Used to solve problems where a function calls itself.

Syntax:

```
FUNCTION factorial(n)
  IF n == 0 THEN
    RETURN 1
  ELSE
    RETURN n * factorial(n - 1)
  ENDIF
ENDFUNCTION
```

8. Sorting and Searching Constructs: Used to describe sorting or searching operations.

- **Sorting:**

  Syntax:

```
FUNCTION bubbleSort(list)
  FOR i = 1 TO LENGTH(list) - 1
    FOR j = 1 TO LENGTH(list) - i
      IF list[j] > list[j + 1] THEN
        SWAP list[j] WITH list[j + 1]
      ENDIF
    ENDFOR
  ENDFOR
  RETURN list
ENDFUNCTION
```

- **Searching:**

  Syntax:

```
FUNCTION binarySearch(list, target)
  SET low = 0
  SET high = LENGTH(list) - 1
  WHILE low <= high
```

```
        SET mid = (low + high) // 2
        IF list[mid] == target THEN
          RETURN mid
        ELSE IF list[mid] < target THEN
          SET low = mid + 1
        ELSE
          SET high = mid - 1
        ENDIF
      ENDWHILE
      RETURN -1  // Target not found
    ENDFUNCTION
```

9. Specialized Constructs: Used for domain-specific algorithms.

- **State Transitions:**

  Syntax:

```
    SWITCH state
      CASE "START":
        OUTPUT "Game is starting."
      CASE "PLAY":
        OUTPUT "Game in progress."
      CASE "END":
        OUTPUT "Game over."
      DEFAULT:
        OUTPUT "Invalid state."
    ENDSWITCH
```

- **Priority Queues:**

  Syntax:

```
    INSERT task WITH priority HIGH INTO queue
```

10. Invoking Classes: Classes in pseudocode represent objects with properties and methods. Invoking a class involves creating an object and calling its methods.

- **Defining a Class:**

  Syntax:

```
    CLASS ClassName
      // Properties
      PROPERTY propertyName
```

```
    // Methods
    FUNCTION methodName(parameters)
      // Operations
      RETURN result
    ENDFUNCTION
  ENDCLASS
```

- **Creating an Object:**
  Syntax:
  ```
  SET objectName = NEW ClassName()
  ```

- **Accessing Properties and Methods:**
  Syntax:
  ```
  SET objectName.propertyName = value
  SET result = objectName.methodName(arguments)
  ```

Example: Using a Class for a Circle

```
CLASS Circle
  PROPERTY radius

  FUNCTION setRadius(r)
    SET radius = r
  ENDFUNCTION

  FUNCTION calculateArea()
    RETURN 3.14 * radius * radius
  ENDFUNCTION
ENDCLASS

START
SET circle1 = NEW Circle()
CALL circle1.setRadius(5)
SET area = circle1.calculateArea()
OUTPUT "The area of the circle is ", area
END
```

---------- The END ----------