PSEUDO CODE:

Pseudocode is a structured way of writing algorithms which looks very much like program code. Since it is informal, there are different ways to write pseudocode. Some commonly used conventions are listed below:

- Indent lines by space (4 space or 1 TAB while coding) to indicate that block of code is contained within previous statement.
- Use uppercase for keywords.
- Use camelCase (mixedCase) for identifiers. Or snake_case, kebab-case, PascalCase etc.)
- Things to be replaced are enclosed in angled brackets (<>).
- Use line numbers (at left side of code) if you need to reference the code. Keep it consecutive.
- Comments start from two forward slashes //. Comment continues until the end of the line.
- Use identifiers that describe the variable, procedure or function they refer to.

- Defining variables:

  `DEFINE <identifier> AS <datatype>`

  Or `DECLARE <identifier> : <data type>`

  There are five basic data types you can use:

  Char for character (a, b, c, A, B, C, #, @ etc.)

  String for collection of characters (Wade, Logan etc.)

  Integer for integers (-1, 2, 3, 253, 0 etc.)

  Real for numbers with decimals (-1.0, 2.56, 8.95 etc.)

  Boolean for Bool values. (True or False)

  You are also allowed to use DATE datatype as fundamental. Normally in `dd/mm/yyyy` format.
- Define constants:

  `CONSTANT <identifier> = <value>`
- Variable Assignments:

  `<identifier> ← <value>`

  E.g.: a ← 35
- Operations:

  You are allowed to use addition (+), subtraction (-), multiplication (*), division (/) without declaring. You can use integer division operators MOD and DIV but they need to be explained explicitly and not assumed.

  ```
  x ← a + b
  c ← c+1
  ```

- Relational operations:

  `>, <, >=, <=, = , <>` can be used and relational operations return `BOOLEAN` value.

- Logic Operators:

  `AND, OR` and `NOT` can be used and they return BOOLEAN value.

- <u>Input and Output:</u>

  ```
  OUTPUT '<Things to print here>' Or PRINT '<Things to print here>'
  OUTPUT '<Things to print here>', <identifier>
  ```

  You can use DISPLAY also.

  ```
  INPUT <identifier> Or READ <identifier>
  ```

- <u>Selection or conditional statements:</u>

  ```
  IF <Boolean expression here> THEN
        <block of code>
  ELSE
        <block of code>
  ENDIF
  ```

  You can add more conditions with ELSEIF too. (No need of THEN in this case)

  ```
  CASE OF <identifier>
        <value1> : <block of code>
        <value2> : <block of code>
        …
  OTHERWISE : <block of code>
  ENDCASE
  ```

- <u>Iteration:</u>

  ```
  FOR <identifier> = <initial value> TO <final value> STEP <increment>
        <block of code>
  NEXT

  REPEAT
        <block of code>
  UNTIL <Boolean expression>

  WHILE <condition>
        <block of code>
  ENDWHILE
  ```

- <u>Declaring arrays:</u>

  `DECLARE <identifier>: ARRAY[<l>:<u>] OF <data type>` For 1-D Array.

  `DECLARE <identifier>: ARRAY[<l1>:<u1>, <l2>:<u2>]` For 2-D Array.

- <u>Using arrays:</u>

  `<identifier>[<index>] ← <value>` For assignment, Similar for displaying

- Abstract data types:

```
TYPE <identifier1>
DECLARE <identifier2> : <data type>
DECLARE <identifier3> : <data type>
...
ENDTYPE
```

- Using custom data types:

  Example:

```
TYPE Student
      DECLARE Surname : STRING
      DECLARE FirstName : STRING
      DECLARE DateOfBirth : DATE
      DECLARE YearGroup : INTEGER
      DECLARE FormGroup : CHAR
      ENDTYPE

DECLARE Pupil1 : Student
DECLARE Pupil2 : Student
DECLARE Form : ARRAY[1:30] OF Student

Pupil1.Surname ← "Johnson"
Pupil1.Firstname ← "Leroy"
Pupil1.DateOfBirth ← 02/01/2005
Pupil1.YearGroup ← 6
Pupil1.FormGroup ← 'A'
Pupil2 ← Pupil1

FOR Index ← 1 TO 30
      Form[Index].YearGroup ← Form[Index].YearGroup + 1
ENDFOR Index
```

- String operations

  String operations like concatenation, searching and splitting can be used but should be explained clearly.

  Where functions are used to format numbers as strings for output, it should be explained.

- Random number.

  `RANDOMBETWEEN (min, max)`: generates a random integer between min and max

  `RND()`: generated a random real number between 0 and 1.

- Defining and calling procedures:

  For procedure definition:

```
PROCEDURE <identifier>
      <statements>
ENDPROCEDURE

PROCEDURE <identifier>(<param1>:<datatype>,<param2>:<datatype>...)
      <statements>
ENDPROCEDURE
```

For procedure call:

```
CALL <identifier>
CALL <identifier>(Value1,Value2...)
```

- <u>Defining and calling functions:</u>

```
FUNCTION <identifier> RETURNS <data type>
      <statements>
ENDFUNCTION

FUNCTION <identifier>(<param1>:<datatype1>,...)RETURNS <data type>
      <statements>
ENDFUNCTION
```

Function isn't called using `CALL` statement but used as part of expression.

- <u>Passing parameters by reference</u>

Example:

```
PROCEDURE SWAP(BYREF X : INTEGER, Y : INTEGER)
      Temp ← X
      X ← Y
      Y ← Temp
ENDPROCEDURE
```

If there are several parameters, they should all be passed by the same method and the `BYVALUE` or `BYREF` keyword need not be repeated. If method not specified, use passing by value.

- <u>Handling text files</u>

`OPENFILE <file identifier> FOR <File mode>`

File modes:

`READ` to read data from file.

`WRITE` to write data to a new file. Existing file will be erased.

`APPEND` to add data to existing file.

`READFILE <File Identifier>, <variable>` reads line by line

`EOF (< File Identifier>)` used to test whether the file pointer is at the end of the file.

`WRITEFILE <File identifier>, <String>`

`CLOSEFILE <File identifier>`

- Handling random files

```
OPENFILE <File identifier> FOR RANDOM
SEEK <File identifier>, <address>
GETRECORD <File identifier>, <Variable>
PUTRECORD <File identifier>, <Variable>

Example:
DECLARE Pupil : Student
DECLARE NewPupil : Student
DECLARE Position : INTEGER

NewPupil.Surname ← "Johnson"
NewPupil.Firstname ← "Leroy"
NewPupil.DateOfBirth ← 02/01/2005
NewPupil.YearGroup ← 6
NewPupil.FormGroup ← 'A'

OPENFILE StudentFile.Dat FOR RANDOM
FOR Position = 20 TO 10 STEP -1
      SEEK StudentFile.Dat, Position
      GETRECORD StudentFile.Dat, Pupil
      SEEK StudentFile.Dat, Position + 1
      PUTRECORD StudentFile.Dat, Pupil
ENDFOR

SEEK StudentFile.Dat, 10
PUTRECORD StudentFile.Dat, NewPupil

CLOSEFILE StudentFile.dat
```