Procedures and function

Starter

"Have you ever faced a problem that seemed overwhelming to solve all at once?" or "Think about how you organize your tasks when working on a big project. What strategies do you use?"

Facilitate a brief discussion on problem-solving strategies. Introduce the idea that breaking down complex problems into smaller, more manageable parts is a common approach in various fields, including programming.

What is a procedure?

A procedure, often referred to as a subroutine or function named which a named sequence of instructions or statements that perform a specific task or operation.

Procedures are used to break down a program into smaller, manageable pieces, making it easier to understand, debug, and maintain. They promote code reusability and modularity by allowing you to encapsulate a particular functionality or a series of steps within a named block of code.

A procedure typically consists of a

* header contains the procedure's name and any input parameters it accepts.
* body. contains the instructions or statements that define the procedure's functionality.

Once a procedure is defined, it can be called or invoked from other parts of the program to execute the instructions contained within it.

A procedure has two parts:

**The Definition** is where the code section is separated from the rest of the code and given a name. Defining a procedure does nothing until the code is called.

The pseudocode syntax

A procedure with no parameters is defined as follows:

PROCEDURE <identifier>

<statements>

ENDPROCEDURE

Procedures with parameters

A procedure with parameters is defined as

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

<statements>

ENDPROCEDURE

**The Call:** We run a procedure by calling it by its name.

Procedures should be called as follows:

With one parameter

CALL <identifier>

Example

PROCEDURE hello\_world

    print("Nice to meet you world")

ENDPROCEDURE

CALL hello\_world

With one parameter

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

Example

PROCEDURE add\_numbers(num1: INTEGER, num2: INTEGER):

sum := num1 + num2 // Calculate the sum

OUTPUT "The sum of " + num1 + " and " + num2 + " is " + sum // Display the result

ENDPROCEDURE

CALL add\_numbers (10, 5)

Output The sum of 10 and 5 is 15.

Where is it most appropriate to use a procedure in a program.

Procedures are commonly used in the construction of algorithms when there is a need to perform a specific task or operation repeatedly or when you want to break down a larger problem into smaller, more manageable parts.

1. Repeated tasks: If there is a particular task that needs to be performed multiple times within the algorithm, it is a good candidate for a procedure. By defining a procedure for the task, you can avoid duplicating the same code in multiple places and improve the readability and maintainability of your algorithm. Instead, you can call the procedure whenever the task needs to be executed.

2. When the algorithm becomes complex and difficult to understand as a whole, it is beneficial to break it down into smaller, more modular parts using procedures. Each procedure can handle a specific subtask or subproblem, making the algorithm more manageable and easier to comprehend. This modular approach improves code organization, readability, and maintainability.

4. Abstraction: Procedures can be used to abstract complex functionality behind a simple interface. By defining a procedure that performs a complex task, you can hide the implementation details and provide a high-level interface to the rest of the algorithm. This abstraction improves the readability and comprehensibility of the algorithm, as the details of the complex task are encapsulated within the procedure.

example

Greet Users: Create a procedure named GREET that takes a user's name as a parameter and prints a greeting message. Call this procedure to greet three different users.

Calculate Area: Define a procedure named CALCULATE\_AREA that calculates and prints the area of a rectangle. The procedure should take the length and width as parameters. Call the procedure with different sets of length and width values.

Factorial Calculation: Write a procedure named FACTORIAL to calculate the factorial of a given number. Call the procedure to find the factorial of three different numbers.

Generate Multiplication Table: Create a procedure named MULTIPLICATION\_TABLE that takes an integer as a parameter and prints the multiplication table for that number up to 10. Call the procedure with different numbers.

Check Prime Number: Define a procedure named IS\_PRIME that takes an integer as a parameter and prints whether the number is prime or not. Call the procedure with different numbers.

Temperature Conversion: Write a procedure named CONVERT\_TO\_CELSIUS that takes a temperature in Fahrenheit as a parameter and prints the equivalent temperature in Celsius. Call the procedure with different Fahrenheit temperatures.

Calculate Simple Interest: Create a procedure named SIMPLE\_INTEREST that calculates and prints the simple interest. The procedure should take the principal amount, interest rate, and time as parameters. Call the procedure with different values.

Display Pattern: Define a procedure named DISPLAY\_PATTERN that takes an integer as a parameter and prints a pattern. For example, if the parameter is 5, the pattern could be:

FUNCTIONS

Functions are blocks of organized, reusable code that perform a specific task. They are designed to be modular, allowing code to be divided into smaller, manageable units.

Like procedures, functions consist of a set of instructions executed sequentially.

A function with no parameters is defined as follows:

FUNCTION <identifier> RETURNS <data type>

<statements>

ENDFUNCTION