Exercise walk-through

Structured Programming ▪ 11.3

eXercise

You must be able to

- define and **call** a **procedure** and a **function**
- use **parameters**, including **by value** and **by reference**
- use the terms argument, parameter, return value, **local**/**global**

Method — Procedure vs function

A **procedure** does an action and returns **nothing**; a **function** returns a **value** used in an expression.

```
PROCEDURE Greet(Name : STRING)
    OUTPUT "Hello, ", Name
ENDPROCEDURE
CALL Greet("Ada")

FUNCTION Square(x : INTEGER) RETURNS INTEGER
    RETURN x * x
ENDFUNCTION
Result ← Square(5) + 1      // Result = 26
```

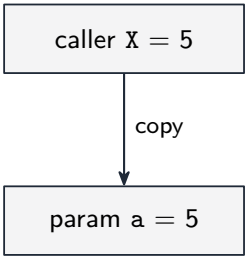
Method — Parameters: by value or by reference

A **parameter** is the variable that **receives** input; the value the caller supplies is the **argument**. Swap must use **BYREF** so the new values stay in the caller:

```
PROCEDURE Swap(BYREF a : INTEGER, BYREF b : INTEGER)
    DECLARE temp : INTEGER
    temp ← a
    a ← b
    b ← temp
ENDPROCEDURE
```

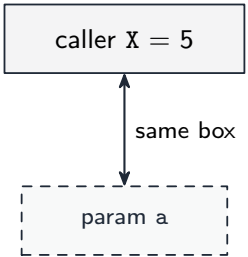
Method — Parameters: by value or by reference

pass by value (a copy)



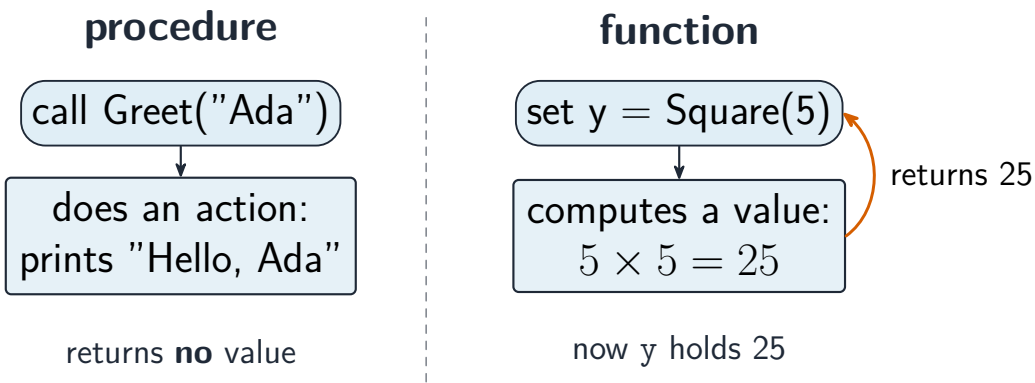
change a → X stays 5

pass by reference (BYREF)



change a → X changes too

Method — Parameters: by value or by reference



Procedures and functions structure a program

Method — Local, global, and when to use a subroutine

A **local** variable lives only inside its subroutine; a **global** is visible everywhere (prefer locals). Use a subroutine when logic **repeats**, has a **clear purpose**, or to **test** a part on its own.

Practice 2.1 ▪ 1 mark

State **one** difference between a **procedure** and a **function**.

Solution 2.1

Worked steps

A **function returns a value** (used in an expression); a **procedure does not** return a value (it performs an action) B1.

Practice 2.2 ▪ 2 marks

Define a procedure `Greet(Name)` that outputs `"Hello, "` followed by the name.

Solution 2.2

Method: Procedure vs function

Worked steps

```
PROCEDURE Greet(Name : STRING)
    OUTPUT "Hello, ", Name
ENDPROCEDURE
```

Solution 2.2

M1 A1 *'PROCEDURE ... ENDPROCEDURE' with a parameter, correct 'OUTPUT'*

Practice 2.3 ▪ 2 marks

Define a function `Cube(x)` that returns `x` cubed.

Solution 2.3

Method: Procedure vs function

Worked steps

```
FUNCTION Cube(x : INTEGER) RETURNS INTEGER
    RETURN x * x * x
ENDFUNCTION
```

Solution 2.3

M1 A1 *'FUNCTION ... RETURNS ... ENDFUNCTION'*, *'RETURN x * x * x'*

Practice 2.4 ▪ 2 marks

State the difference between an **argument** and a **parameter**.

Solution 2.4

Method: Parameters: by value or by reference

Worked steps

An **argument** is the value the caller passes in; a **parameter** is the variable inside the subroutine that **receives** it B1 B1.

Exam-style 3.1 ▪ 3 marks

Write a **function** `MaxOf(a, b)` that returns the larger of two integers.

Solution 3.1

Method: Procedure vs function

Worked steps

```
FUNCTION MaxOf(a : INTEGER, b : INTEGER) RETURNS INTEGER
    IF a > b THEN
        RETURN a
    ELSE
        RETURN b
    ENDIF
ENDFUNCTION
```

Solution 3.1

M1 M1 A1 *'FUNCTION ... RETURNS INTEGER', compare 'a' and 'b', 'RETURN' the larger in both cases*

Exam-style 3.2 ▪ 4 marks

Explain **pass by value** and **pass by reference**, and state which one a `Swap` procedure must use, and why.

Solution 3.2

Method: Parameters: by value or by reference

Worked steps

By value — the routine gets a **copy**, so changes inside it do **not** affect the caller B1. **By reference** — the routine shares the caller's actual variable, so changes **do** affect it B1. `Swap` must use **pass by reference (`BYREF`)** B1 so the swapped values remain in the caller's variables after the call B1.

Exam-style 3.3 ▪ 2 marks

State **two** benefits of using subroutines in a large program.

Solution 3.3

Worked steps

Any two: avoids repeating code; easier to read/maintain; can be tested on its own; supports decomposition/teamwork B1 B1.

Exam-style 3.4 ▪ 1 mark

State what is meant by a **local variable**.

Solution 3.4

Method: Local, global, and when to use a subroutine

Worked steps

A variable declared **inside** a subroutine that exists **only while that subroutine runs** (and is not visible outside it) B1.