Procedures and functions

Ref As Pseudocode

PROCEDURE and CALL

A **subroutine** is a named block of code that does one job. You write it once and use it as often as you need. Pseudocode has two kinds: procedures and functions.

A **procedure** carries out a task. It does not give a value back.

```
PROCEDURE <identifier>()
   <statement(s)>
ENDPROCEDURE

PROCEDURE <identifier>(<param1> : <data type>, <param2> : <data type>
↪  ...)
   <statement(s)>
ENDPROCEDURE
```

You run a procedure with `CALL`:

```
CALL <identifier>()
CALL <identifier>(<value1>, <value2> ...)
```

```
PROCEDURE DrawLine(LineLength : INTEGER, Symbol : CHAR)
   DECLARE Line : STRING
   DECLARE i : INTEGER
   Line ← ""
   FOR i ← 1 TO LineLength
      Line ← Line & Symbol
   NEXT i
   OUTPUT Line
ENDPROCEDURE

PROCEDURE ShowTitle()
   CALL DrawLine(12, '=')
   OUTPUT "  MAIN MENU"
   CALL DrawLine(12, '=')
ENDPROCEDURE

CALL ShowTitle()
CALL DrawLine(5, '*')
```

- A **parameter** is a variable in the header that receives a value. `LineLength` and `Symbol` are the parameters of `DrawLine`.

- The value you send in the `CALL` is called an **argument** . The arguments must be in the **same order** and of the **right types** as the parameters.
- A procedure with no parameters still has empty brackets: `ShowTitle()`.
- A `CALL` is a complete statement on its own line.

■ Why use subroutines?

- The same code is not written again and again.
- Each subroutine can be written and tested on its own.
- The main program becomes short and easy to read.

Common mistakes

- Calling a procedure without `CALL`: write `CALL DrawLine(5, '*')`, not just `DrawLine(5, '*')`.
- Leaving out the data type of a parameter in the header.
- Forgetting `ENDPROCEDURE`.

FUNCTION and RETURN

A **function** works like a procedure, but it **returns a single value** to the place where it was used. The header says the data type of that value.

```
FUNCTION <identifier>(<param1> : <data type> ...) RETURNS <data type>
   <statement(s)>
ENDFUNCTION
```

```
FUNCTION Max(Number1 : INTEGER, Number2 : INTEGER) RETURNS INTEGER
   IF Number1 > Number2 THEN
      RETURN Number1
   ELSE
      RETURN Number2
   ENDIF
ENDFUNCTION

DECLARE Distance : INTEGER
Distance ← 8
OUTPUT "Penalty fine = ", Max(10, Distance * 2)
```

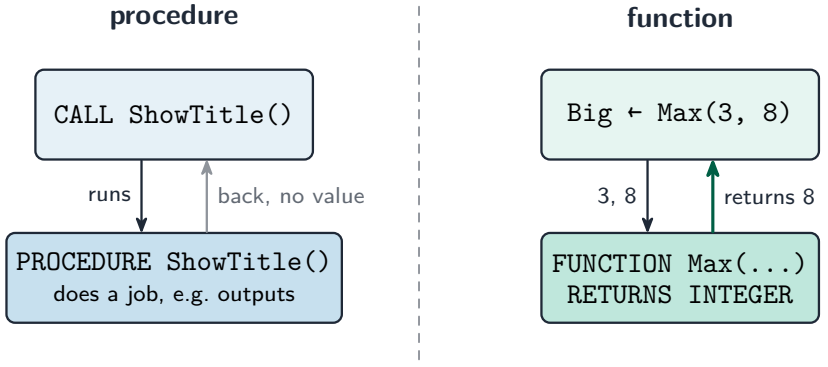
- `RETURN` gives the value back. When `RETURN` runs, the function **stops at once**; any lines after it are skipped.
- A function call is **not** a statement on its own. Use it **inside an expression**: in an assignment, an `OUTPUT`, a condition, or as an argument.
- **Never** use `CALL` with a function.

```
FUNCTION IsEven(Number : INTEGER) RETURNS BOOLEAN
   RETURN Number MOD 2 = 0
ENDFUNCTION

DECLARE Value : INTEGER
FOR Value ← 1 TO 4
   IF IsEven(Value) THEN
      OUTPUT Value, " is even"
   ENDIF
NEXT Value
```

Procedure or function?

- The subroutine works out **one value** that the caller needs: a **function**.
- It **does** something (outputs, changes an array, writes a file) and has no single answer: a **procedure**.



A procedure is called to do a job; a function returns a value to use

Common mistakes

- `CALL Max(3, 4)`. A function is used in an expression: `Biggest ← Max(3, 4)`.
- Forgetting `RETURNS <data type>` in the header, or forgetting the `RETURN` statement in the body.
- Outputting the answer inside the function instead of returning it, when the question asks for a function that **returns** a value.

BYVAL and BYREF

There are two ways to pass a value to a parameter.

- **By value** (`BYVAL`): the subroutine gets a **copy** of the value. If it changes the parameter, the caller's variable does **not** change.
- **By reference** (`BYREF`): the subroutine gets the caller's **own variable**. A change inside the subroutine **is** seen by the caller after the call.

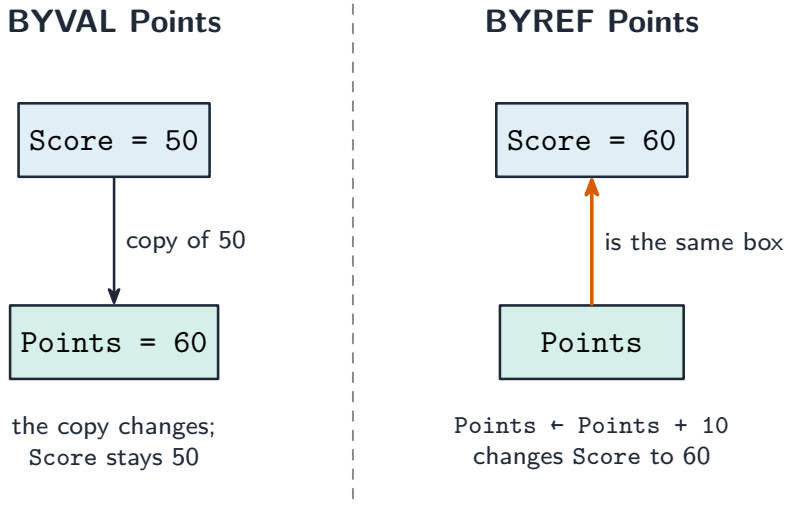
Write the keyword before the parameter in the header. If you write neither, the parameter is passed **by value**.

```
PROCEDURE AddBonus(BYVAL Points : INTEGER)
   Points ← Points + 10
ENDPROCEDURE

PROCEDURE AddBonusRef(BYREF Points : INTEGER)
   Points ← Points + 10
ENDPROCEDURE

DECLARE Score : INTEGER
Score ← 50
CALL AddBonus(Score)
OUTPUT "After BYVAL: ", Score
CALL AddBonusRef(Score)
OUTPUT "After BYREF: ", Score
```

The first `OUTPUT` shows 50 (only the copy changed). The second shows 60.



BYVAL gives the procedure a copy; BYREF gives it the caller's own variable

One keyword for several parameters

If several parameters in a row use the same method, write the keyword once. Here **both** `X` and `Y` are `BYREF`:

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

DECLARE A : INTEGER
DECLARE B : INTEGER
A ← 3
B ← 8
CALL Swap(A, B)
OUTPUT "A = ", A, ", B = ", B
```

When to use BYREF

- A procedure must give back **more than one** result.
- A procedure must change the caller's variable, as `Swap` does.

Do not pass parameters by reference to a **function**. A function gives back its one value with `RETURN`.

Common mistakes

- Expecting a change to a plain (`BYVAL`) parameter to be seen outside. It is a copy.
- Using `BYREF` in a `FUNCTION` header.
- Writing `BYREF` in an IGCSE 0478 answer. That syllabus does not have it.

Local and global variables

The **scope** of a variable is the part of the program where it can be used.

- A **local variable** is declared **inside** a subroutine. It exists only while that subroutine runs, and only that subroutine can use it. Parameters are local too.
- A **global variable** is declared in the **main program**, outside every subroutine. Any part of the program can use it.

exercises.lol exercises.com.cn

```
DECLARE Total : INTEGER

PROCEDURE AddToTotal(Amount : INTEGER)
   DECLARE Doubled : INTEGER
   Doubled ← Amount * 2
   Total ← Total + Doubled
ENDPROCEDURE

Total ← 0
CALL AddToTotal(5)
CALL AddToTotal(3)
OUTPUT "Total: ", Total
```

- `Total` is **global**: the procedure can change it, and the main program sees the result, 16.
- `Doubled` and `Amount` are **local** to `AddToTotal`. The main program cannot use them.

Prefer local variables

- A subroutine that uses only its parameters and local variables can be tested and reused on its own.
- Two subroutines can both use a local variable called `i` without any clash.
- A global that many subroutines change makes errors hard to find.

So pass data in through **parameters**, give results back with `RETURN` (or `BYREF`), and keep globals for the few values that really belong to the whole program.

Practice

1. Write a function `Area(Width : REAL, Height : REAL) RETURNS REAL`, then use it to output the area of a 3.5 by 2.0 rectangle.

2. Write a procedure `MinMax` that takes three integers and gives back the smallest and the largest through two `BYREF` parameters.

<details> <summary>Model answer</summary>

```
FUNCTION Area(Width : REAL, Height : REAL) RETURNS REAL
   RETURN Width * Height
ENDFUNCTION

OUTPUT "Area: ", Area(3.5, 2.0)
```

```
PROCEDURE MinMax(A : INTEGER, B : INTEGER, C : INTEGER, BYREF Smallest :
↪  INTEGER, Largest : INTEGER)
   Smallest ← A
   Largest ← A
   IF B < Smallest THEN
      Smallest ← B
   ENDIF
   IF C < Smallest THEN
      Smallest ← C
   ENDIF
   IF B > Largest THEN
      Largest ← B
   ENDIF
   IF C > Largest THEN
      Largest ← C
   ENDIF
ENDPROCEDURE

DECLARE Low : INTEGER
DECLARE High : INTEGER
CALL MinMax(7, 2, 9, Low, High)
OUTPUT "Smallest ", Low, ", largest ", High
```

`A`, `B` and `C` are passed by value (the default). `BYREF` before `Smallest` also covers `Largest`, because the keyword is not repeated.

</details>

exercises.lol exercises.com.cn