

11.3.1 Functions, and choosing between a procedure and a function

Name: _____ Class: _____ Date: _____ Total: 46 marks

KEY WORDS function • return value • interface • parameter
• argument

Objective

Learn to define and to use a **function** : a module that works out **one value** and **returns** it to the statement that called it. You also learn to choose between a procedure and a function, and to use the correct words for the parts of a module. Sheet 11.3 taught procedures; sheet 11.3.2 deals with passing parameters by value and by reference.

You must be able to:

- define a function with `FUNCTION ... RETURNS ... ENDFUNCTION`, and return a value with `RETURN`
- call a function inside an expression
- explain where in an algorithm it is appropriate to use a function, and where a procedure
- use the terms **header**, **interface** , **parameter** , **argument** and **return value**

1 Worked examples

Study these first. Each one shows a **method** that a question later on this sheet needs.

■ What a function is

A **function** is a module that **returns a value**. You have used built-in functions such as `LENGTH()`; now you write your own.

```
FUNCTION Square(Number : INTEGER) RETURNS INTEGER
    RETURN Number * Number
ENDFUNCTION
```

```
Result ← Square(5) + 1
OUTPUT Square(3)
```

- The header ends with `RETURNS` and the **data type** of the value that comes back.

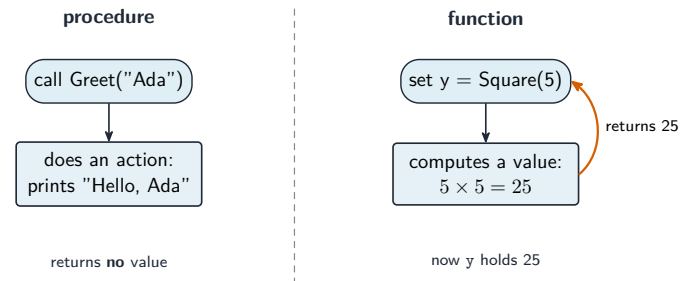
- `RETURN` gives the value back and **ends** the function.
- A function is **not** called with `CALL`. It is used **inside an expression**, and the **return value** replaces the call: `Square(5) + 1` becomes `25 + 1`, so `Result` is 26; `OUTPUT Square(3)` outputs 9.

A function that **outputs** its answer instead of returning it is wrong: `Result ← Square(5)` would have nothing to store.

■ Procedure or function?

Ask one question: **does the caller need a value back?**

- No, it only needs something done** (display a menu, print a report) → a **procedure**, called on a line of its own with `CALL`.
- Yes, it uses one value that the module works out** (a total, a grade, `TRUE` or `FALSE`) → a **function**, used inside an expression: `IF IsValid(Code) THEN`.



■ The words the mark scheme uses

```
FUNCTION Area(SideLength : REAL, Width : REAL) RETURNS REAL    ← header
    RETURN SideLength * Width                                    ← return
    ↪ value
ENDFUNCTION

Total ← Area(3.5, 2.0)                                          ← call, with
    ↪ arguments
```

- header**: the first line: the keyword, the name, the parameters and, for a function, `RETURNS` and its type
- interface**: what a caller must know: the name, the parameters with their types and order, and what is returned
- parameter**: the variable **inside** the module that receives a value: `SideLength`, `Width`
- argument**: the value that the **caller** supplies: 3.5, 2.0
- return value**: the value handed back, which replaces the call in the expression: here 7.0

The parameter is in the **definition**; the argument is in the **call**. If you say it the

other way round, you lose the mark.

■ **Writing a function that scores**

A “write a function” question is marked point by point. Write the points in this order:

1. the **header**: keyword, name, every parameter with its type, and RETURNS <type>;
2. **declare the local variables**, including a loop counter;
3. **initialise** any total, count or string **before** a loop;
4. the **loop** and the **test** inside it, using the **parameter**, not a fixed value;
5. **close** every construct: ENDIF, NEXT, ENDWHILE, ENDFUNCTION;
6. **RETURN** the result.

```
FUNCTION CountAbove(BYVAL Limit : INTEGER) RETURNS INTEGER
  DECLARE Index, Count : INTEGER
  Count ← 0
  FOR Index ← 1 TO 50
    IF Score[Index] > Limit THEN
      Count ← Count + 1
    ENDIF
  NEXT Index
  RETURN Count
ENDFUNCTION
```

..... header: name, parameters, return type

..... local variables declared with a type

..... total initialised before the loop

..... loop over every element

..... the condition, with the right boundary

..... the update, inside the IF

..... every construct closed

..... the return value, after the loop

*Worked example: write a function **IsLeap** that takes a year and returns TRUE if it is a leap year. A year is a leap year when it divides by 4, except that a century year must divide by 400.*

```
FUNCTION IsLeap(Year : INTEGER) RETURNS BOOLEAN
  IF Year MOD 400 = 0 THEN
    RETURN TRUE
  ENDIF
  IF Year MOD 100 = 0 THEN
    RETURN FALSE
  ENDIF
  IF Year MOD 4 = 0 THEN
    RETURN TRUE
  ENDIF
  RETURN FALSE
ENDFUNCTION
```

RETURN ends the function at once, so the tests are written in order of priority and no nested IF is needed. The call is then used as a value, for example as the condition of an IF: IsLeap(2028).

2 Practice

Now use the methods above. The first questions need one idea each.

2.1 State **one** difference between a **procedure** and a **function**. [1]

procedure

function

2.2 Study this pseudocode.

```
FUNCTION Double(Number : INTEGER) RETURNS INTEGER
  RETURN Number * 2
ENDFUNCTION

Value ← Double(4) + Double(1)
OUTPUT Value
```

4

→

Double

→

(a) State the value that is output. [1]

(b) State the **return value** of the call Double(4). [1]

2.3 Define a function Cube(x) that returns x cubed. [2]

x

→

Cube

→ x^3

2.4 Write the **header** for a function **Area** that takes two real numbers and returns a real number. [2]

REAL

REAL

→

Area

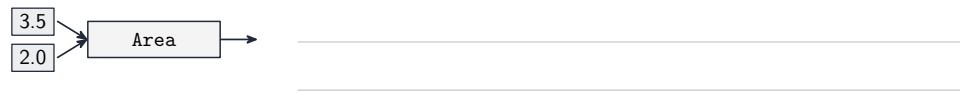
→REAL

2.5 State the difference between an **argument** and a **parameter**. [2]

```
FUNCTION Area(SideLength : REAL, Width : REAL)
```

Area(3.5, 2.0)

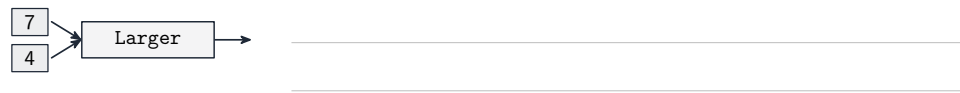
2.6 A program contains the statement `Total ← Area(3.5, 2.0)`. State the **arguments** in this call, and state what the **return value** replaces. [2]



2.7 A student writes this function. It should give back the larger of two numbers.

```
FUNCTION Larger(First : INTEGER, Second : INTEGER) RETURNS INTEGER
  IF First > Second THEN
    OUTPUT First
  ELSE
    OUTPUT Second
  ENDIF
ENDFUNCTION
```

State the error, and state how to correct it. [2]



2.8 For each task, state whether a **procedure** or a **function** is the better choice, and give a reason. [1]

(a) display a menu of options on the screen [1]

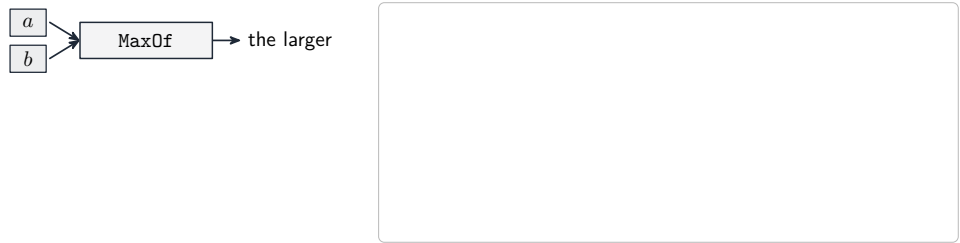


(b) work out the discount on a price [1]

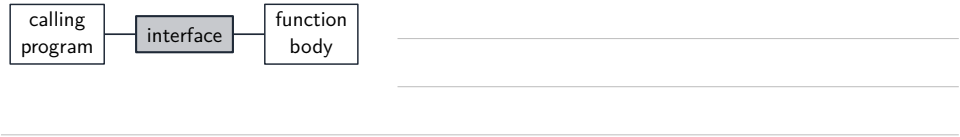
(c) exchange the values of two variables [1]

3 Exam-style questions

3.1 Write a **function** `MaxOf(a, b)` that returns the larger of two integers. [3]



3.2 Explain what is meant by the **interface** of a function. [2]



3.3 A program works out a discounted price in three different places. The same three lines of code are used each time, but on different prices and rates. The programmer decides to replace them with a function `Discount()`.

(a) Write the **header** for `Discount()`. [2]

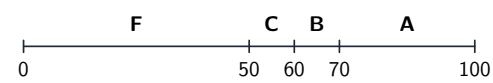


(b) Write the statement that calls `Discount()` for a price of 80.00 at a rate of 15, storing the result in `Final`. [1]

(c) State why a **function** suits this better than a procedure. [1]

3.4 A function `Grade()` takes an integer mark from 0 to 100 and returns a grade as a string: "A" for 70 or more, "B" for 60 or more, "C" for 50 or more, and "F" below 50.

Write the function, and write it so that no mark is tested more than it needs to be. [5]



3.5 Write a **function** `Initials()` that takes a person's full name as a string of two or more words separated by single spaces, and returns a string of their initials in capitals with no spaces. For example, "ada byron lovelace" returns "ABL".

You may use `LENGTH`, `MID` and `UCASE`. [6]



3.6 A cinema program works out the price of a ticket. A ticket costs 12.00. A customer under 16 pays half. On a Tuesday, every customer gets 2.00 off the price (after the half price, if there is one).

(a) Write a function `TicketPrice` that takes the customer's age and a Boolean value that is `TRUE` on a Tuesday, and returns the price of the ticket. [6]

cinema 12.00

day

(b) Write a statement that uses your function to store, in `Total`, the price of one ticket for a customer aged 12 **plus** one ticket for a customer aged 40, on a day that is not a Tuesday. [2]

(c) **Explain** why `TicketPrice` is better written as a function than as a procedure. [2]

Solutions

Each answer shows the steps, then the **result**.

2.1

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

2.2

(a) `Double(4)` is 8 and `Double(1)` is 2, so `Value` is $8 + 2 = 10$

(b) 8

2.3

- `FUNCTION ... RETURNS ... ENDFUNCTION` with `RETURN`

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

2.4

- keyword, name and both parameters with their types
- `RETURNS REAL`

```
FUNCTION Area(SideLength : REAL, Width : REAL) RETURNS REAL
```

2.5

- an **argument** is the value the caller passes in
- a **parameter** is the variable inside the subroutine that **receives** it

2.6

- the arguments are 3.5 and 2.0 – the values the caller supplies
- the return value **replaces the call** in the expression, so `Total` is assigned 7.0

2.7

- the function **outputs** the larger value, but it never **returns** a value, so the calling statement gets nothing back
- replace each `OUTPUT` with `RETURN`: `RETURN First` and `RETURN Second`

2.8

(a) a **procedure** – it performs an action and the caller needs no value back

(b) a **function** – it computes one value that the caller then uses in an expression

(c) a **procedure**, with both parameters passed `BYREF` – two values change, so there is no single value to return

3.1

- `FUNCTION ... RETURNS INTEGER`; compare; `RETURN` the larger

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

3.2

- the interface is what a caller must know in order to use the function
- its name, its parameters with their types and order, and the type of the value it returns

3.3

(a) `FUNCTION Discount(Price : REAL, Rate : REAL) RETURNS REAL`

- one mark for the keyword, name and both parameters; one for `RETURNS REAL`

(b) `Final ← Discount(80.00, 15)`

(c) the caller needs the discounted price **back** to use it, and a function's return value replaces the call inside the expression

3.4

- header with the parameter and `RETURNS STRING`
- test the **highest** boundary first
- return as soon as a test passes, so no later test runs
- the remaining boundaries in descending order
- a final `RETURN "F"` for everything below 50

```
FUNCTION Grade(Mark : INTEGER) RETURNS STRING
    IF Mark >= 70 THEN
        RETURN "A"
    ENDIF
    IF Mark >= 60 THEN
        RETURN "B"
    ENDIF
    IF Mark >= 50 THEN
        RETURN "C"
    ENDIF
    RETURN "F"
ENDFUNCTION
```

- testing from the top down means each `IF` only has to check one boundary: a mark of 65 fails the first test and passes the second, so `Mark < 70 AND Mark >= 60` is never needed

3.5

- header with the parameter and RETURNS STRING
- declare the locals, including the loop counter, and start the result as ""
- loop over the whole name
- take the first character, and every character that follows a space
- concatenate it, in capitals, onto the result
- RETURN after the loop

```
FUNCTION Initials(FullName : STRING) RETURNS STRING
  DECLARE Result : STRING
  DECLARE Index : INTEGER
  Result ← ""
  FOR Index ← 1 TO LENGTH(FullName)
    IF Index = 1 OR MID(FullName, Index - 1, 1) = " " THEN
      Result ← Result & UCASE(MID(FullName, Index, 1))
    ENDIF
  NEXT Index
  RETURN Result
ENDFUNCTION
```

3.6

(a) a header with two parameters (an INTEGER and a BOOLEAN) and RETURNS REAL

- a local variable for the price, set to 12.00
- an IF that halves the price when the age is under 16
- a second, separate IF that takes 2.00 off on a Tuesday
- the two tests in the correct order, and every IF closed
- the price returned

```
FUNCTION TicketPrice(Age : INTEGER, IsTuesday : BOOLEAN) RETURNS REAL
  DECLARE Price : REAL
  Price ← 12.00
  IF Age < 16 THEN
    Price ← Price / 2
  ENDIF
  IF IsTuesday = TRUE THEN
    Price ← Price - 2.00
  ENDIF
  RETURN Price
ENDFUNCTION
```

(b) two calls with the correct arguments, in the correct order

- the two return values added and assigned: Total ← TicketPrice(12, FALSE) + TicketPrice(40, FALSE)

(c) the module works out **one value**, the price, which the calling program needs

- a function returns this value, so the call can be used inside an expression, as in part (b); a procedure cannot be used in this way