

Exercise walk-through

11.3.1

Functions, and choosing between a procedure and a function

A-Level Computer Science

Functions

A **function** returns a value, which becomes part of an expression:

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



Use a **procedure** when the subroutine performs an action; use a **function** when it computes a value for the caller.

Where each is appropriate

The syllabus asks *where in the construction of an algorithm* each one fits.

A procedure

where the **same group of steps** is needed at several points: validate an input, print a menu, swap two values. The steps are written once and **CALL**ed by name.

A function

where a **single value** must be calculated and then used in an expression, such as a total, a TRUE/FALSE result or the larger of two numbers.

The **return value** replaces the call, which is why a function can sit inside a condition: `IF IsValid(Code) THEN`

Terminology

Term	Meaning
definition	the PROCEDURE ... ENDPROCEDURE (or function) block
call	where it is invoked
argument	a value passed in
parameter	the variable that receives it
return value	what a function passes back
procedure / function header	the first line, giving the name and parameters: PROCEDURE Name(params) or FUNCTION Name(params) RETURNS type
interface, or signature	name + parameters + return type: what a caller must know to use it

Worked example: describe the header

Describe each term used in this header:

```
FUNCTION Pass2(Count : INTEGER) RETURNS BOOLEAN
```

Term	Meaning
FUNCTION	a subroutine that returns a value
Pass2	the identifier used to call it
Count	the parameter: the identifier that receives the argument passed in
INTEGER	the data type of the parameter
RETURNS BOOLEAN	the data type of the value the function returns

In **PROCEDURE** MyProc(Count : **INTEGER**, Message : **STRING**) both identifiers are **parameters**, used inside like local variables.

Converting a procedure into a function

- 1 change **PROCEDURE** to **FUNCTION** and add **RETURNS** with the type
- 2 replace the **OUTPUT**, or the **BYREF** parameter that carried the result out, with a **RETURN** statement
- 3 change every call so that the returned value is used:

Result $\leftarrow$ Unpack(Text) instead of **CALL** Unpack(Text, Result)

For a “write the header” question, write the **whole line**: **FUNCTION**
Calculate(Expression : STRING) RETURNS INTEGER.

Exercise 2.1

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

[1]

procedure

function

Exercise 2.1

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

[1]

procedure

function

2.1

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

Exercise 2.2 (a)

2.2 Study this pseudocode.

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

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



(a) State the value that is output.

[1]

Exercise 2.2 (a)

2.2 Study this pseudocode.

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

```
Value  $\leftarrow$  Double(4) + Double(1)  
OUTPUT Value
```



(a) State the value that is output.

[1]

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

Exercise 2.2 (b)

2.2 Study this pseudocode.

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

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



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

[1]

Exercise 2.2 (b)

2.2 Study this pseudocode.

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

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



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

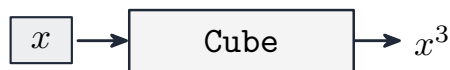
[1]

(b) 8

Exercise 2.3

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

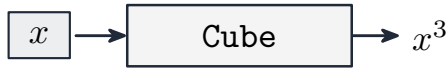
[2]



Exercise 2.3

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

[2]



2.3

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

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

Exercise 2.4

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



Exercise 2.4

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



2.4

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

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

Exercises 2.5-2.6

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

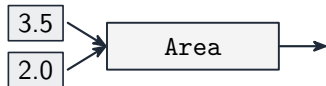
[2]

```
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]



Exercises 2.5-2.6

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

[2]

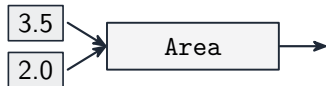
```
FUNCTION Area(SideLength : REAL, Width : REAL)
```

```
Area(3.5, 2.0)
```

2.5

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

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]



2.6

- 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

Exercise 2.7

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]



Exercise 2.7

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]



2.7

- 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

Exercise 2.8

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

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

(a)  (b)  (c)  _____

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

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

Exercise 2.8

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

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

(a)  (b)  (c) 

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

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

2.8

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

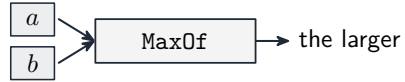
(b) a **function** – it computes one value that the caller then uses in an expression

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

Exercise 3.1

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

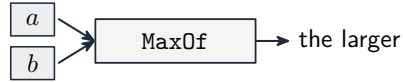
[3]



Exercise 3.1

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

[3]



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
```

Exercise 3.2

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

[2]



Exercise 3.2

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

[2]



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

Exercise 3.3 (a)

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]



Exercise 3.3 (a)

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]



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

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

Exercise 3.3 (b)

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()`.

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

[1]

Exercise 3.3 (b)

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()`.

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

(b) `Final ← Discount(80.00, 15)`

Exercise 3.3 (c)

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()`.

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

[1]

Exercise 3.3 (c)

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()`.

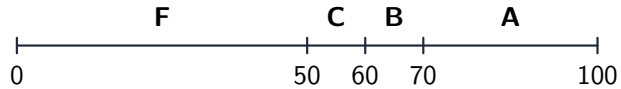
(c) State why a **function** suits this better than a procedure. [1]

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

Exercise 3.4

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.

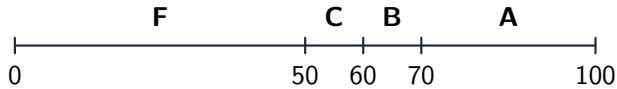
Write the function, and write it so that no mark is tested more than it needs to be. [5]



Exercise 3.4

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]



3.4

- header with the parameter and RETURNS STRING

Full solution 3.4 (1/2)

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"
```

Full solution 3.4 (2/2)

```
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

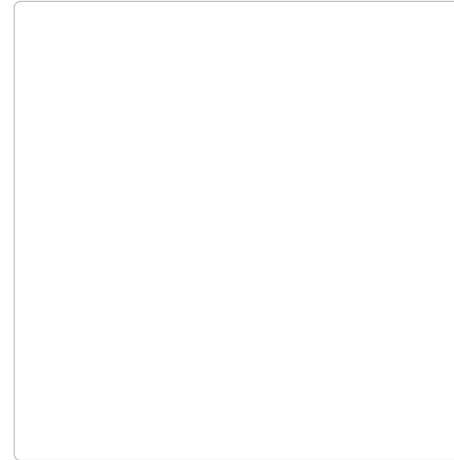
Exercise 3.5

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]

`"ada byron lovelace"` → `Initials` → `"ABL"`



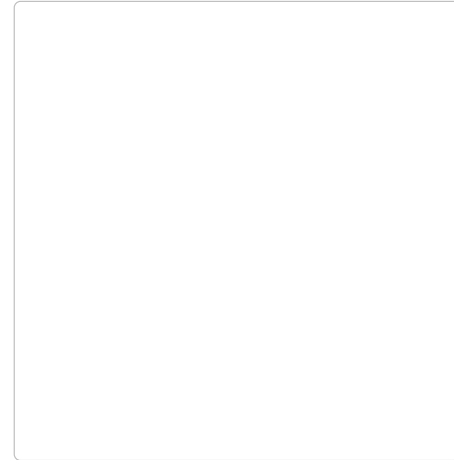
Exercise 3.5

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]

`"ada byron lovelace"` → `Initials` → `"ABL"`



3.5

- header with the parameter and `RETURNS STRING`
- declare the locals, including the loop counter, and start the result as `" "`

Full solution on next slide

Full solution 3.5

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
```

Exercise 3.6

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]

Exercise 3.6

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]

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

Full solution 3.6 (1/2)

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
```

Full solution 3.6 (2/2)

```
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