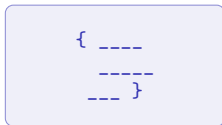


Programming Fundamentals

IGCSE Computer Science ■ Topic 8

IGCSE Computer Science



What we will cover

- 1 Variables, constants, data types
- 2 The three structures
- 3 Operators & strings
- 4 Procedures & functions
- 5 Arrays
- 6 File handling

age

16

a variable is a named box
that holds a value



Variables, constants, data types

- **variable** : a named store whose value can **change**
- **constant** : a named value that is **fixed**

Five **data types** : INTEGER, REAL, CHAR, STRING, BOOLEAN.

Pick the **smallest precise** type: a whole count is INTEGER, a price is REAL, a yes/no is BOOLEAN.

integer	whole number: 7
real	decimal: 3.5
char	one letter: 'A'
string	text: "hello"
boolean	true or false

Variables, constants, and the data types

variable

a named store whose value **can** change while the program runs

constant

a value that **never** changes, like Pi – set in one place, so it is easy to read and to alter

declare

state the name and type before use, so the computer reserves the right amount of memory

integer / real

a whole number; a number with a decimal point

char / string

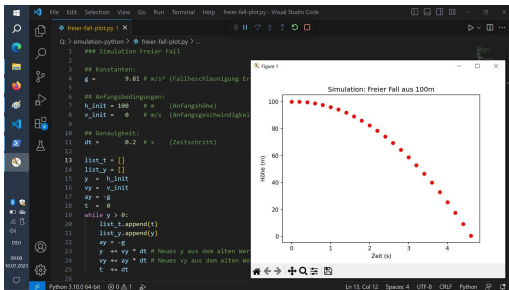
a **single** character; a sequence of them

Boolean

only TRUE or FALSE

Use **meaningful identifiers** : TotalPrice rather than t. It is a marked criterion on the practical paper, not a style preference.

Variables and constants are named stores for values



While a program runs, its variables are held in the computer's memory (RAM)

Variables and constants are named stores for values, set in the program's code

Input and output

Input reads a value from the user.

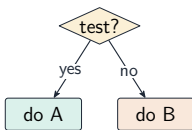
Output shows a value on the screen.

The three basic structures

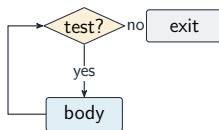
Sequence



Selection



Iteration



A **data type** says what kind of value a variable holds.

Every program is built from three control structures.

Sequence, selection, iteration

sequence

steps run **in order**, one after another

selection

chooses which steps to run, based on a condition – IF, or CASE when there are many choices

iteration

a **loop** repeats steps – and there are three kinds

count-controlled

FOR – repeats a **known** number of times

pre-condition

WHILE – tests **before** each pass, so it may run zero times

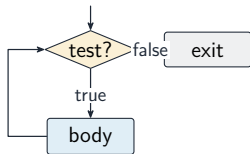
post-condition

REPEAT UNTIL – tests **after**, so it always runs at least once

Choose WHILE when the loop might not need to run at all, and REPEAT when it must run once before the test can be made.

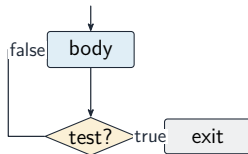
Loops and counting

WHILE (pre-condition)



tested first $\Rightarrow$ may
run **zero** times

REPEAT (post-condition)



tested last $\Rightarrow$ runs
at least once

- **totalling** – keep adding values to a running total: $\text{'total} \leftarrow \text{total} + \text{value}'$.
- **counting** – add 1 to a counter each time: $\text{'count} \leftarrow \text{count} + 1'$.

Three loops, and when each is right

Count-controlled loop

A **count-controlled loop** (FOR) runs a known number of times – when you know how many before you start.

Pre-condition loop

A **pre-condition loop** (WHILE) tests *before* the body, so it may run zero times. Use it when the data might already be finished.

Post-condition loop

A **post-condition loop** (REPEAT UNTIL) tests *after*, so it always runs at least once – right for a menu, or for validating input.

Pick the loop from the question: known count, might-not-run, or must-run-once.

Totalling and counting

```
count = count + 1    count up by 1
total = total + value add to a running total
```

```
total <- 0
```

```
count <- 0
```

```
FOR each value (4, 7, 5) value total count
```

```
total <- total + value start 0 0
```

```
count <- count + 1 4 4 1
```

```
7 11 2
```

```
5 16 3
```

total keeps a running sum;

count adds 1 each pass -

a **trace table** follows both

*A trace table follows a totalling and counting loop
pass by pass*

A counter counts up by 1; a total builds a running sum



Operators

arithmetic

+ - × /
DIV (whole),
MOD (remainder)

comparison

= ≠ < > ≤ ≥

logical

AND, OR, NOT

17 DIV 5 = 3 (whole part), 17 MOD 5 = 2 (remainder) – MOD is how you test “is it even?”

Operators, and string handling

arithmetic

add, subtract, multiply, divide; plus MOD and DIV for remainder and whole division

comparison

equal, not equal, less than, greater than, and the two combined forms

logical

AND, OR, NOT

length

how many characters a string contains

substring

take part of a string, from a starting position

upper case / lower case

change letters to capitals, or to small letters – used to compare input regardless of how it was typed

Watch the numbering: the first character may be position 0 or 1 depending on the language, and the exam paper says which.

The three families of operators

arithmetic

`+` `-` `*` `/`
`MOD` `DIV`

give a
number

relational

`=` `<>` `<` `>`
`<=` `>=`

compare →
true or false

logical

`AND` `OR`
`NOT`

join conditions →
true or false

The three families of operators: arithmetic, relational and logical

String handling

- `LENGTH(s)` – number of characters
- `SUBSTRING(s, start, n)` – part of a string
- `UCASE` / `LCASE` – change case
- `&` – **concatenation** (join)

String positions count from **1**, not 0. And a **nested** statement is just one structure **inside** another – an IF within a FOR.

Nested statements

A **nested statement** is one control structure placed **inside** another.

You will not have to write more than three levels of nesting.



Procedures and functions

- **procedure** : a named block that **does an action**, returns nothing
- **function** : a named block that **returns a value**

Library routines are ready-made subroutines (ROUND, RANDOM).

A **function** call sits inside an expression (it gives a value back); a **procedure** call is a statement on its own.

Procedures, functions, and where a variable lives

procedure	a named block of code that does something and returns nothing
function	a named block that returns a value , so it can be used inside an expression
parameter	a value passed in, so the same routine works on different data
local variable	exists only inside its routine – safer, because nothing else can change it
global variable	visible everywhere – convenient, and a common source of hard-to-find bugs
why subroutines	write once, use many times; and each piece can be tested on its own

Prefer local variables and pass what you need as parameters. That is exactly what makes a program **maintainable**, which the paper marks explicitly.

Writing a maintainable program

meaningful names

score, not x

comments

explain *why*,
not the obvious

subroutines

split the work into
named blocks

Maintainable code is code the **next person** (or you, next month) can read and change without fear.



Arrays

An **array** holds many values of the **same type** under one name, picked out by an **index** .

A 1-D array is a list (`Marks[3]`); a 2-D array is a grid (`Grid[2,3]`) – perfect for a FOR loop over every element.

Arrays, in detail

index	1	2	3	4	5
scores	90	75	60	88	50

↑
`scores[2]` is 75

1-D: a list, one index

	column		
	1	2	3
1			
2			
3			

← `grid[2,3]`

2-D: a table, [row, column]

An **array** holds many values of the **same type** under one name, picked out by an **index** – one index for a list, two for a table.

One- and two-dimensional arrays

1D array

a single **list** – one index picks an element, as in `Names[3]`

2D array

a **table** of rows and columns – **two** indexes, as in `Grid[2,5]`

why use one

one name for many values, so a loop can process them all

traversing a 1D array

one FOR loop over the indexes

traversing a 2D array

nested loops – the outer over rows, the inner over columns

the caution

going outside the declared bounds is an error the language will report

An array plus a loop is the pattern behind every totalling, counting, searching and sorting question on the paper.

Arrays in one and two dimensions

1D array

A **one-dimensional (1D) array** is a list: one index, `scores[3]`, and every element the same data type.

2D array

A **two-dimensional (2D) array** is a table: two indices, `grid[row, col]` – a seating plan, a game board, a set of results per student.

Traversing

One loop walks a 1D array; nested loops walk a 2D one, the outer over rows and the inner over columns.

Why use one

One name for many values, so a loop can process them all – which is impossible with separate variables.

Say what the index *means* before you write the loop: position in a list, or row and column in a table.

File handling

Store data on disk so it survives when the program ends:

- `OPENFILE` for `READ` / `WRITE`
- `READFILE` / `WRITEFILE`
- `CLOSEFILE` when done

Variables live in RAM and vanish at the end; a **file** keeps the data between runs. Always `CLOSEFILE` – unsaved writes are lost otherwise.

Worked example – total and average of 5 marks

A program must read 5 marks, then output the total and the average.

```
total ← 0
FOR i ← 1 TO 5
    INPUT mark
    total ← total + mark
NEXT i
average ← total / 5
OUTPUT total, average
```

Two lines carry the marks.
total ← 0 must sit **before** the loop – inside it, the total resets on every pass – and the division must sit **after** it, or the average is taken of an unfinished total.

Exam tips

- A **variable** can change while the program runs; a **constant** is fixed. Learn the five data types: integer, real, char, string, Boolean.
- Every program is built from three structures: **sequence**, **selection** (IF / CASE), and **iteration** (loops).
- A **WHILE** loop tests *before* the body, so it may run zero times; a **REPEAT ... UNTIL** loop tests *after*, so it always runs at least once.
- MOD gives the remainder and DIV the whole-number part of a division: 17 MOD 5 is 2, 17 DIV 5 is 3.
- A **function** returns a value; a **procedure** does not. A **local** variable works only inside its block; a **global** one works anywhere.



Topic 8 – key takeaways

1 Structures

sequence, selection, iteration

2 Loops

FOR known; WHILE before; REPEAT after

3 Subroutines

procedure acts; function returns a value

4 Arrays & files

array = same type by index; file survives runs

Check yourself

- 1 Which loop may run zero times?
- 2 What is $17 \text{ MOD } 5$?
- 3 Procedure or function: which returns a value?
- 4 From which index do string positions count?

```
{  
  _  
  _  
}
```

Answers 1. WHILE 2. 2 3. a function 4. 1