Quick reference

Ref As Pseudocode

Every keyword on one page

Use this page to check a **syntax** form quickly. The topic number tells you where it is explained.

Data and assignment

Form	Example	Topic
`DECLARE <id> : <type>`	`DECLARE Total : INTEGER`	2
`CONSTANT <id> = <value>`	`CONSTANT Rate = 0.08`	2
`<id> ← <value>`	`Total ← Total + 1`	2
`DECLARE <id> : ARRAY[<l>:<u>] OF <type>`	`DECLARE Names : ARRAY[1:30] OF STRING`	6
`DECLARE <id> : ARRAY[<l1>:<u1>, <l2>:<u2>] OF <type>`	`DECLARE Board : ARRAY[1:3, 1:3] OF CHAR`	6
`TYPE <id> … ENDTYPE`	a record with `DECLARE` fields	9

Data types: `INTEGER`, `REAL`, `CHAR`, `STRING`, `BOOLEAN`, `DATE`.

Operators

Kind	Operators
arithmetic	`+ - * / DIV MOD`
relational	`= <> < > <= >=`
logic	`AND OR NOT`
strings	`&`

Input and output

Form	Example
`INPUT <id>`	`INPUT Age`
`OUTPUT <value(s)>`	`OUTPUT "Total: ", Total`

exercises.lol exercises.com.cn

Selection and loops

```
IF <condition> THEN
   <statement(s)>
ELSE
   <statement(s)>
ENDIF

CASE OF <identifier>
   <value> : <statement(s)>
   <value1> TO <value2> : <statement(s)>
   OTHERWISE : <statement(s)>
ENDCASE

FOR <id> ← <v1> TO <v2> STEP <n>
   <statement(s)>
   ↪ <statement(s)>
NEXT <id>

WHILE <condition>
   <statement(s)>
ENDWHILE

REPEAT
   <statement(s)>
UNTIL
↪ <condition>
```

Subroutines

```
PROCEDURE <id>(BYREF <p> : <type>, …)
↪ …) RETURNS <type>
   <statement(s)>
ENDPROCEDURE

CALL <id>(<arguments>)
↪ <id>(<arguments>)

FUNCTION <id>(<p> : <type>,

   <statement(s)>
   RETURN <value>
ENDFUNCTION
<variable> ←
```

Files

Form	Notes
`OPENFILE <file> FOR READ / WRITE / APPEND`	`WRITE` empties the file
`READFILE <file>, <variable>`	reads the next line
`WRITEFILE <file>, <data>`	writes one line
`EOF(<file>)`	`TRUE` when no lines are left
`CLOSEFILE <file>`	always close

exercises.lol exercises.com.cn

■ Built-in functions (from the Paper 2 insert)

Function	Returns
`LENGTH(S)`	number of characters
`LEFT(S, x)` / `RIGHT(S, x)`	first / last x characters
`MID(S, x, y)`	y characters from position x
`TO_UPPER(x)` / `TO_LOWER(x)`	x in upper / lower case
`NUM_TO_STR(x)` / `STR_TO_NUM(x)`	number to string / string to number
`IS_NUM(S)`	`TRUE` if S is a valid number
`ASC(c)` / `CHR(x)`	character code / character
`INT(x)`	whole-number part (cuts off, never rounds)
`RAND(x)`	random `REAL` from 0 up to (not including) x

Comments start with `//`. Keywords are in capitals; identifiers start with a letter.

Coming from IGCSE 0478

Many students learned IGCSE pseudocode first. The two Cambridge syllabuses use **different** rules, and a habit from one is often a mistake in the other. This table shows the 9618 form you must now use.

	IGCSE 0478	AS/A Level 9618
`THEN`	on its own line, under the `IF`	at the end of the `IF` line
constants	`CONSTANT Pi ← 3.142`	`CONSTANT Pi = 3.142`
`DIV` and `MOD`	functions: `DIV(10, 3)`	operators: `10 DIV 3`
joining strings	commas in `OUTPUT` only	the `&` operator
`OTHERWISE` in `CASE`	no colon	with a colon: `OTHERWISE :`
`CASE` values	single values, one statement each	ranges with `TO`, several statements
parameters	always by value	`BYVAL` or `BYREF`
part of a string	`SUBSTRING(S, 2, 3)`	`MID(S, 2, 3)`, also `LEFT` and `RIGHT`
upper case of a string	`UCASE(S)`	`TO_UPPER(S)`
rounding, random	`ROUND(X, 2)`, `RANDOM()`	no `ROUND`; `INT(X)` cuts off; `RAND(x)`

exercises.lol exercises.com.cn

	IGCSE 0478	AS/A Level 9618
identifiers	start with a capital; letters and digits	start with any letter; underscores allowed
data types	five	six: `DATE` is added
records	not used	`TYPE … ENDTYPE`

Here is the same small algorithm in 9618 form. It runs on this page:

```
CONSTANT Rate = 0.5
DECLARE Total : REAL
DECLARE Label : STRING
Total ← 10.0
IF 7 MOD 2 = 1 THEN
   Label ← "Odd, " & "half rate"
   OUTPUT Label, ": ", Total * Rate
ENDIF
```

The IGCSE version would put `THEN` on its own line, write `CONSTANT Rate ← 0.5` and `MOD(7, 2)`, and could not use `&`. The site's IGCSE pseudocode reference shows that form.

Writing a Paper 2 answer

Paper 2 asks you to write, complete, correct and trace pseudocode. These habits win marks.

Before you write

- **Read the whole question** and underline what must be **output** or **returned**.
- Check whether it asks for a **procedure** or a **function**, and copy the header exactly when it is given: the name, the parameters and the return type.
- Look at the **insert**: it lists every built-in function you may use, with its parameters.

While you write

- **Declare** your variables, one per line, with sensible types.
- **Initialise** totals and counters (`Total ← 0`) before the loop.
- Pick the loop that fits (topic 5): `FOR` for a known count, `WHILE` for "may be zero times", `REPEAT` for "at least once".
- **Indent** every block, and close every block: `ENDIF`, `ENDWHILE`, `NEXT`, `ENDCASE`, `ENDPROCEDURE`, `ENDFUNCTION`.
- Use the **exact** names from the question for arrays, variables and subroutines. If the question calls the array `StockLevel`, do not call it `Stock`.

- Write `CALL` for procedures, and never for functions.
- Use `←` for assignment and `=` only for comparison.

■ Tracing an algorithm

A **trace table** records the value of each variable after each step, so you can see what an algorithm does.

- Make one column for each variable, and one for the output.
- Write a new value only when it **changes**, going down one row each time.
- Work line by line. Do not guess the result from what you think the algorithm "should" do.

■ When you finish

- Read your answer as the computer would: does every loop end? Is every variable given a value before it is used?
- Test it in your head with a simple case and with an **edge case** edge case : an empty file, a value exactly on a boundary, a value that is not found.

A **dry run** with a trace table is the best way to find a logic error before the examiner does.

exercises.lol exercises.com.cn