11 Programming – Part 2

Name: _____ Class: _____ Date: _____

Vocabulary

Write each word three times.

English	**Write it 3 times** (English)		
operators			
library routines			
concatenation			
nested			

Recall

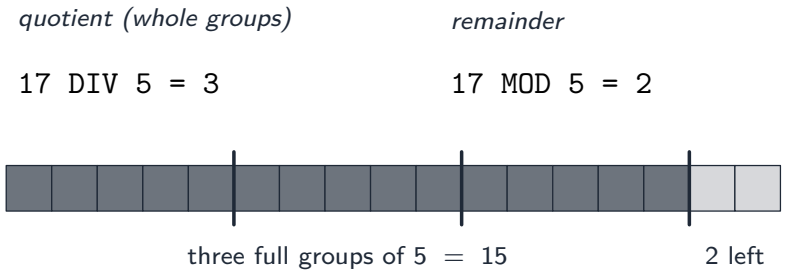
In **Part 1** you used variables and constants, the three constructs (selection, iteration), arrays, and subroutines (procedures, functions, parameters, scope). Part 2 adds two more operators and ready-made functions, the `CASE` statement, and writing clean, efficient code.

New ideas

Read these first. Each idea has a short worked example. Then do the Practice.

■ 1 Operators and library routines

Besides $+ - \times \div$, two useful **operators** work on whole numbers: `DIV` (integer division – `7 DIV 2 = 3`) and `MOD` (the remainder – `7 MOD 2 = 1`).

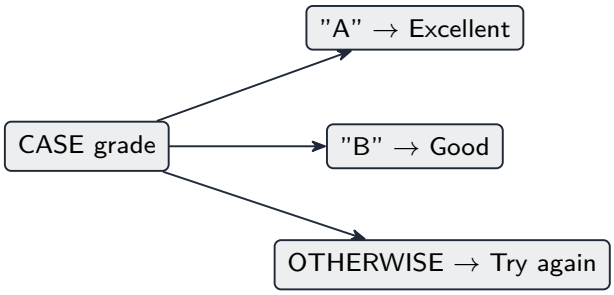


Many jobs already have ready-made **library routines** , so you don't write them yourself – string functions like `LENGTH(s)`, `LEFT(s, n)`, `MID(s, start, len)`, and joining two strings (**concatenation**) with `&`.

Worked example. `23 DIV 10 = 2` and `23 MOD 10 = 3`, so 23 is "2 tens and 3 ones"; `LEFT("HELLO", 2)` gives `"HE"`.

2 The CASE statement

To test one value against many options, a deep **nested** `IF` is hard to read; a `CASE` is cleaner:




```
CASE OF grade
    "A": OUTPUT "Excellent"
    "B": OUTPUT "Good"
    OTHERWISE: OUTPUT "Try again"
ENDCASE
```

Worked example. With `grade = "B"`, the `CASE` above matches the `"B"` branch and outputs "Good".

3 Writing efficient, readable code

- compute a value **once before a loop** if it doesn't change inside it;
- **exit a loop early** when the answer is found;
- use **meaningful names** (`numberOfPupils`, not `n`) and comment the *intent*, not the mechanics.

slower

```
FOR i = 1 TO n
  x = slow()  // every pass
  use(x, i)
NEXT i
```

faster

```
x = slow()  // once
FOR i = 1 TO n
  use(x, i)
NEXT i
```

Worked example. If a loop runs `tax ← price * 0.2` every pass but `price` never changes inside it, move that line above the loop so it runs once, not a thousand times.

Watch out: `DIV` throws away the remainder – `7 DIV 2` is 3, not 3.5. Use `MOD` when you want the remainder.

Practice

11.1 Find (a) `17 DIV 5` and (b) `17 MOD 5`. [2]

11.2 Find (a) `LENGTH("HELLO")` and (b) `LEFT("HELLO", 2)`. [2]

11.3 State when a `CASE` statement is a better choice than a nested `IF`. [1]

11.4 State the condition (using `MOD`) that is true when `n` is even. [1]

11.5 State one way to make a loop run more efficiently. [1]

11.6 Explain *why* `n MOD 2 = 0` is a correct test for whether `n` is even. [2]

11.7 Challenge. Using `DIV` and `MOD`, write pseudocode that inputs a number of seconds and outputs the whole minutes and the leftover seconds. [3]