Exercise walk-through

# 11.1.2 Built-in functions and library routines

A-Level Computer Science

# You must be able to

- use the string functions `LENGTH`, `LEFT`, `RIGHT`, `MID`, `TO_UPPER` and `TO_LOWER`
- join strings with `&`, and convert between numbers and strings with `NUM_TO_STR` and `STR_TO_NUM`
- use `ASC`, `CHR`, `INT` and `RAND`
- work out an expression in which one function is inside another
- state the benefits of using a program library

# 1 The method

# Method — What a function is

A **function** takes values (its **parameters** 参数), does a job, and **returns** a result. The result **replaces** the call:
`Size ← LENGTH("Computer")` stores 8 in `Size`.
The exam gives this list on the insert: do not learn it by heart, but use the **exact names** and the **correct order** of the parameters.

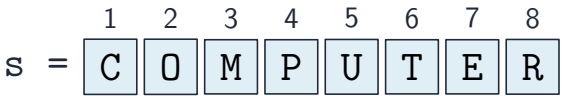
# Method — The string functions

- Positions are counted from **1**, and a space counts as a character: `LENGTH("Happy Days")` is 10.
- `TO_UPPER` and `TO_LOWER` take a `CHAR` or a `STRING`. The Pseudocode Guide also has `UCASE` and `LCASE`, which take **one `CHAR` only**.

# Method — The string functions

| function | returns | example |
|---|---|---|
| `LENGTH(s)` | the number of characters in `s` | `LENGTH("Computer")` is 8 |
| `LEFT(s, n)` | the first `n` characters | `LEFT("Computer", 3)` is `"Com"` |
| `RIGHT(s, n)` | the last `n` characters | `RIGHT("Computer", 2)` is `"er"` |
| `MID(s, start, n)` | `n` characters, beginning at position `start` | `MID("Computer", 4, 3)` is `"put"` |
| `TO_UPPER(x)` | `x` in capital letters | `TO_UPPER("hi")` is `"HI"` |
| `TO_LOWER(x)` | `x` in small letters | `TO_LOWER("Hi")` is `"hi"` |

# Method — The string functions




```
LENGTH(s) = 8      RIGHT(s, 2) = "ER"
LEFT(s, 3) = "COM" TO_UPPER(s) = "COMPUTER"
MID(s, 4, 3) = "PUT" TO_LOWER(s) = "computer"
```

# Method — Joining and converting

- The operator & **concatenates** 拼接 (joins) two strings: `"Sum" & "mer"` is `"Summer"`.
- + does **not** join strings, and a number cannot be joined to a string. Convert it first:

| function | returns | example |
|---|---|---|
| `NUM_TO_STR(x)` | the number `x` as a string | `NUM_TO_STR(87.5)` is `"87.5"` |
| `STR_TO_NUM(s)` | the string `s` as a number | `STR_TO_NUM("23.45")` is `23.45` |
| `IS_NUM(s)` | `TRUE` if `s` is a valid number | `IS_NUM("12a")` is `FALSE` |
| `ASC(c)` | the ASCII code of the character `c` | `ASC('A')` is 65 |
| `CHR(n)` | the character with the ASCII code `n` | `CHR(66)` is `'B'` |

# Method — Joining and converting

`"Total: " & Score` is an error when `Score` is an `INTEGER`. The correct statement is `"Total: " & NUM_TO_STR(Score)`.

# Method — Numeric functions

| function | returns | example |
|---|---|---|
| `INT(x)` | the whole-number part of `x` | `INT(27.5415)` is 27 |
| `RAND(n)` | a random **real** number from 0 up to, but not including, `n` | `RAND(6)` could be `4.27` |

`RAND` returns a real number, so a random **integer** needs `INT`. Dice, 1 to 6: `INT(RAND(6))` gives 0 to 5, so add 1. From `Low` to `High`: `INT(RAND(High - Low + 1)) + Low`.

# Method — A function inside a function

Work from the **inside out**: work out the inner function first, and put its result in the outer one.

1. Inner: `RIGHT("Program", 2)` is `"am"`.
2. Outer: `TO_UPPER("am")` is **`"AM"`**.

*Worked example: `LENGTH("Hello") + STR_TO_NUM("4")`.* `LENGTH("Hello")` is 5 and `STR_TO_NUM("4")` is 4, so the result is **9**.

# Method — Program libraries

A **program library** 程序库 is a collection of routines that are already written, tested and compiled. Benefits of using one:

- it **saves time**: no new code to write and test
- the routines are already **tested**, so they are reliable
- they are written by **experts**, and may do a job the programmer could not write
- a routine can be **used many times**

# 2 Practice

## Practice 2.1 ▪ 5 marks

Name the function that does each job.

# Practice 2.1 (a)

returns the number of characters in a string

# Solution 2.1 (a)

Method: Joining and converting — p.8

**Worked steps**

`LENGTH` B1

# Practice 2.1 (b)

returns the last few characters of a string

# Solution 2.1 (b)

**Worked steps**

`RIGHT` B1

# Practice 2.1 (c)

changes a string into a number

# Solution 2.1 (c)

**Worked steps**

`STR_TO_NUM` B1

# Practice 2.1 (d)

returns the ASCII code of a character

# Solution 2.1 (d)

**Worked steps**

`ASC` B1

# Practice 2.1 (e)

returns the whole-number part of a real number

# Solution 2.1 (e)

**Worked steps**

`INT` B1

## Practice 2.2 ▪ 4 marks

Give the value of each expression.

# Practice 2.2 (a)

```
LENGTH("Computer")
```

# Solution 2.2 (a)

Method: The string functions — p.5

**Worked steps**

8 B1

# Practice 2.2 (b)

```
RIGHT("Computer", 3)
```

# Solution 2.2 (b)

**Worked steps**

`"ter"` B1

# Practice 2.2 (c)

```
MID("Computer", 4, 2)
```

# Solution 2.2 (c)

**Worked steps**

"pu" – positions count from 1, so position 4 is the p B1

# Practice 2.2 (d)

```
INT(STR_TO_NUM("9.7"))
```

# Solution 2.2 (d)

**Worked steps**

**9** – `STR_TO_NUM` gives $9.7$ and `INT` takes the whole-number part B1

## Practice 2.3 ▪ 4 marks

The variable `Word` holds `"Keyboard"`. Work out the value of each expression.

# Practice 2.3 (a)

```
LEFT(Word, 3)
```

# Solution 2.3 (a)

**Worked steps**

the first three characters: `"Key"` B1

# Practice 2.3 (b)

```
MID(Word, 4, 2)
```

# Solution 2.3 (b)

**Worked steps**

two characters from position 4: `"bo"` B1

# Practice 2.3 (c)

```
TO_UPPER(RIGHT(Word, 5))
```

# Solution 2.3 (c)

**Worked steps**

the inner function first: `RIGHT(Word, 5)` is `"board"`, so the result is **`"BOARD"`**

B1

# Practice 2.3 (d)

```
LENGTH(Word) DIV 3
```

# Solution 2.3 (d)

**Worked steps**

`LENGTH(Word)` is 8, and `8 DIV 3` is **2** B1

## Practice 2.4 ▪ 4 marks

Work out the value of each expression.

# Practice 2.4 (a)

`ASC('C')`, given that `ASC('A')` is 65

# Solution 2.4 (a)

**Worked steps**

‘`C'` is two letters after ‘`A'`: $65 + 2 = \mathbf{67}$ B1

# Practice 2.4 (b)

```
CHR(70)
```

# Solution 2.4 (b)

**Worked steps**

$70 - 65 = 5$ letters after `'A'`: `'F'` B1

# Practice 2.4 (c)

```
"AB" & "12"
```

# Solution 2.4 (c)

**Worked steps**

`&` joins the strings: `"AB12"` B1

# Practice 2.4 (d)

```
STR_TO_NUM("12") + 3
```

# Solution 2.4 (d)

**Worked steps**

the string becomes the number 12, and $12 + 3 = \mathbf{15}$ B1

## Practice 2.5 ▪ 2 marks

Write the pseudocode statement that assigns to `Dice` a random integer from 1 to 6, and state why `INT` is needed.

# Solution 2.5

Method: Numeric functions — p.10

**Worked steps**

- `Dice ← INT(RAND(6)) + 1` B1
- `RAND` returns a **real** from 0 up to but not including 6, so `INT` is needed to make it a whole number; the `+ 1` shifts the range from 0-5 to 1-6 B1

## Practice 2.6 ▪ 2 marks

A student writes `Message ← "Total: " + Score`, where `Score` is an `INTEGER`. State what is wrong and write the correct statement.

# Solution 2.6

Method: Joining and converting — p.8

**Worked steps**

- + cannot join a string to an integer: the two are different types B1
- `Message ← "Total: " & NUM_TO_STR(Score)` – & concatenates, and the number must be converted to a string first B1

# 3 Exam-style

## Exam-style 3.1 ▪ 3 marks

Give the value of each expression.

# Exam-style 3.1 (a)

```
TO_UPPER(RIGHT("Program", 2))
```

# Solution 3.1 (a)

**Worked steps**

the inner call runs first: `RIGHT("Program", 2)` is `"am"`, so the answer is `"AM"` B1

# Exam-style 3.1 (b)

```
LENGTH(MID("Computer", 3, 4))
```

# Solution 3.1 (b)

**Worked steps**

`MID("Computer", 3, 4)` is `"mput"`, so `LENGTH` of it is `4` B1

# Exam-style 3.1 (c)

```
ASC(‘Q') - ASC(‘A')
```

# Solution 3.1 (c)

**Worked steps**

‘`Q'` is 81 and ‘`A'` is 65, so the answer is **16** B1

# Exam-style 3.2 ▪ 4 marks

Complete each expression so that it evaluates to the value shown. Use only functions and operators from the insert.

| Expression | Evaluates to |
|---|---|
| `.....  ("Saturday", 3)` | `"Sat"` |
| `.....  ('D')` | `68` |
| `2 * .....  ("27")` | `54` |
| `MID("Notebook", ..... , .....)` | `"book"` |

# Solution 3.2

Method: The string functions — p.5

## Worked steps

- the first three characters → `LEFT` B1
- a character to its code → `ASC` B1
- the string `"27"` must become the number 27 → `STR_TO_NUM` B1
- `"book"` starts at position 5 and has 4 characters → `5, 4` B1

## Exam-style 3.3 ▪ 3 marks

Write pseudocode that reads a word and outputs it in **capitals** followed by its **length**, using built-in functions.

# Solution 3.3

Method: The string functions — p.5

## Worked steps

- INPUT, then TO_UPPER and LENGTH M1 M1 A1

```
OUTPUT "Enter a word:"
INPUT Word
OUTPUT TO_UPPER(Word)
OUTPUT "Length: ", LENGTH(Word)
```

## Exam-style 3.4 ▪ 4 marks

A school builds each student's username from their surname, their first name and their year group: the **first three letters of the surname in capitals**, then the **first letter of the first name**, then the **year group**.
Write a single pseudocode statement that assigns this username to `Username`, using the variables `Surname`, `FirstName` and `Year`, where `Year` is an `INTEGER`.

# Solution 3.4

## Worked steps

- `LEFT(Surname, 3)` for the first three letters **B1**
- `TO_UPPER(...)` around it: it accepts a whole string, whereas `UCASE` takes one `CHAR` and is not on the insert **B1**
- `LEFT(FirstName, 1)` for the initial **B1**
- `NUM_TO_STR(Year)`, and the three parts joined with `&` **B1**

```
Username ← TO_UPPER(LEFT(Surname, 3)) & LEFT(FirstName, 1) & NUM_TO_STR(Year)
```

## Solution 3.4

- for `Surname "Chen"`, `FirstName "Yuwei"` and `Year 12` this gives `"CHEY12"`

## Exam-style 3.5 ▪ 3 marks

State **three** benefits of using a program library rather than writing the routines yourself.

# Solution 3.5

Method: Program libraries — p.12

**Worked steps**

any three:

- the routines have already been written, compiled and **tested**, so they are less likely to contain errors B1
- they save development time, because the code does not have to be written again B1
- they may do things the programmer could not write themselves, such as complex statistics or graphics; they are written by experts; and a routine with a fixed interface can be called from anywhere in the program, and reused across many programs B1

# Exam-style 3.6 ▪ 10 marks

A shop gives every product a code such as `"TV-2025-0457"`. The code has three parts: two letters for the type of product, the year the product was first sold, and a product number. The code is stored in the variable `Code`.

# Exam-style 3.6 (a) ▪ 3 marks

Work out the value of each expression when `Code` is `"TV-2025-0457"`.

| Expression | Evaluates to |
|---|---|
| `LEFT(Code, 2)` | |
| `STR_TO_NUM(MID(Code, 4, 4)) + 1` | |
| `LENGTH(Code) - 8` | |

# Solution 3.6 (a)

Method: The string functions — p.5

**Worked steps**

`LEFT(Code, 2)` → `"TV"` B1

- `MID(Code, 4, 4)` is "2025", which becomes the number 2025, and $2025 + 1 = \mathbf{2026}$ B1
- `LENGTH(Code)` is 12, and $12 - 8 = \mathbf{4}$ B1

## Exam-style 3.6 (b) ▪ 3 marks

Write **one** pseudocode statement that stores the product number, as an `INTEGER`, in the variable `ProductNumber`.

# Solution 3.6 (b)

**Worked steps**

the last four characters hold the product number: `RIGHT(Code, 4)` (or `MID(Code, 9, 4)`) B1

- they are converted with `STR_TO_NUM` B1
- the result is assigned to `ProductNumber`: `ProductNumber ← STR_TO_NUM(RIGHT(Code, 4))` B1

# Exam-style 3.6 (c) ▪ 4 marks

The shop makes a short label from the code: the two letters in small letters, then the last two digits of the year. For `"TV-2025-0457"` the label is `"tv25"`. Write **one** pseudocode statement that stores the label in the variable `Label`.

# Solution 3.6 (c)

**Worked steps**

the two letters: `LEFT(Code, 2)` B1

- made small with `TO_LOWER` B1
- the last two digits of the year: `MID(Code, 6, 2)` B1
- the two parts joined with `&` and assigned: `Label ← TO_LOWER(LEFT(Code, 2)) & MID(Code, 6, 2)` B1