

Exercise walk-through

11.1.2

Built-in functions and library routines

A-Level Computer Science

11.1.2 Built-in functions and library routines

Name: _____ Class: _____ Date: _____ **Total: 51 marks**

- KEY WORDS**
- built-in functions
 - library routines
 - parameters
 - concatenates
 - program library

Objective

Learn to use the **built-in functions** and **library routines** of pseudocode: the string functions, the conversion functions and the numeric functions that the Paper 2 **insert** gives you. This is the last sheet of 11.1. You need the variables of sheet 11.1 and the operators of sheet 11.1.1.

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 Key knowledge and worked examples

Study these first. Each one shows a **method** that a question later on this sheet needs.

■ What a function is

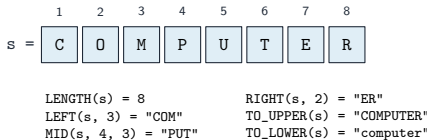
A **function** takes values (its **parameters**), does a job, and **returns** a result.
The result **replaces** the call:

`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.

■ 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**.



function	returns	example
<code>LENGTH(s)</code>	the number of characters	<code>LENGTH("Computer")</code> is 8
<code>LEFT(s, n)</code>	the first <code>n</code> characters	<code>LEFT("Computer", 3)</code> is "Com"
<code>RIGHT(s, n)</code>	the last <code>n</code> characters	<code>RIGHT("Computer", 2)</code> is "er"
<code>MID(s, start, n)</code>	<code>n</code> characters from <code>start</code>	<code>MID("Computer", 4, 3)</code> is "put"
<code>TO_UPPER(x)</code>	<code>x</code> in capital letters	<code>TO_UPPER("hi")</code> is "HI"
<code>TO_LOWER(x)</code>	<code>x</code> in small letters	<code>TO_LOWER("Hi")</code> is "hi"

■ Joining and converting

- The operator `&` **concatenates** (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(<i>x</i>)	the number <i>x</i> as a string	NUM_TO_STR(87.5) is "87.5"
STR_TO_NUM(<i>s</i>)	the string <i>s</i> as a number	STR_TO_NUM("23.45") is 23.45
IS_NUM(<i>s</i>)	TRUE if <i>s</i> is a valid number	IS_NUM("12a") is FALSE
ASC(<i>c</i>)	the ASCII code of the character <i>c</i>	ASC('A') is 65
CHR(<i>n</i>)	the character with the ASCII code <i>n</i>	CHR(66) is 'B'

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

■ Numeric functions

function	returns	example
INT(<i>x</i>)	the whole-number part of <i>x</i>	INT(27.5415) is 27
RAND(<i>n</i>)	a random real number below <i>n</i>	RAND(6) could be 4.27

RAND(6) returns a real number from 0 up to, but **not including**, 6, 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.

■ A function inside a function

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

Worked example: TO_UPPER(RIGHT("Program", 2)).

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.

■ Program libraries

A **program library** 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**

■ Key knowledge: dates use a DATE value

SETDATE(day, month, year) takes three integers in that order and returns a DATE, not a string. DAYINDEX(DateValue) gives the weekday number, where Sunday is 1; MONTH(DateValue) gives the month number. Read the insert for the exact names and types.

Worked example. Meeting $\leftarrow$ SETDATE(6, 10, 2026) stores 6 October 2026. MonthsLeft $\leftarrow$ 12 - MONTH(Meeting) stores 2.

Exercise 2.1 (a)

2.1 Name the function that does each job.

[5]

(a) returns the number of characters in a string

Exercise 2.1 (a)

2.1 Name the function that does each job.

[5]

(a) returns the number of characters in a string

(a) **LENGTH**

Exercise 2.1 (b)

2.1 Name the function that does each job.

[5]

(b) returns the last few characters of a string

Exercise 2.1 (b)

2.1 Name the function that does each job.

[5]

(b) returns the last few characters of a string

(b) **RIGHT**

Exercise 2.1 (c)

2.1 Name the function that does each job.

[5]

(c) changes a string into a number

Exercise 2.1 (c)

2.1 Name the function that does each job.

[5]

(c) changes a string into a number

(c) **STR_TO_NUM**

Exercise 2.1 (d)

2.1 Name the function that does each job.

[5]

(d) returns the ASCII code of a character

Exercise 2.1 (d)

2.1 Name the function that does each job.

[5]

(d) returns the ASCII code of a character

(d) **ASC**

Exercise 2.1 (e)

2.1 Name the function that does each job.

[5]

(e) returns the whole-number part of a real number

C	o	m	p	u	t	e	r
---	---	---	---	---	---	---	---

Exercise 2.1 (e)

2.1 Name the function that does each job.

[5]

(e) returns the whole-number part of a real number

C	o	m	p	u	t	e	r
---	---	---	---	---	---	---	---

(e) **INT**

Exercise 2.2 (a)

2.2 Give the value of each expression.

[4]

(a) `LENGTH("Computer")`

Exercise 2.2 (a)

2.2 Give the value of each expression.

[4]

(a) `LENGTH("Computer")`

(a) **8**

Exercise 2.2 (b)

2.2 Give the value of each expression.

[4]

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

Exercise 2.2 (b)

2.2 Give the value of each expression.

[4]

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

(b) `"ter"`

Exercise 2.2 (c)

2.2 Give the value of each expression.

[4]

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

Exercise 2.2 (c)

2.2 Give the value of each expression.

[4]

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

(c) "pu" – positions count from 1, so position 4 is the p

Exercise 2.2 (d)

2.2 Give the value of each expression.

[4]

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

"Computer"

string

"9.7"

string

Exercise 2.2 (d)

2.2 Give the value of each expression.

[4]

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

"Computer"

string

"9.7"

string

(d) **9** – `STR_TO_NUM` gives 9.7 and `INT` takes the whole-number part

Exercise 2.3 (a)

2.3 The variable `Word` holds "Keyboard". Work out the value of each expression. [4]

(a) `LEFT(Word, 3)`

Exercise 2.3 (a)

2.3 The variable `Word` holds "Keyboard". Work out the value of each expression. [4]

(a) `LEFT(Word, 3)`

(a) the first three characters: **"Key"**

Exercise 2.3 (b)

2.3 The variable `Word` holds "Keyboard". Work out the value of each expression. [4]

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

Exercise 2.3 (b)

2.3 The variable `Word` holds "Keyboard". Work out the value of each expression. [4]

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

(b) two characters from position 4: **"bo"**

Exercise 2.3 (c)

2.3 The variable `Word` holds "Keyboard". Work out the value of each expression. [4]

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

Exercise 2.3 (c)

2.3 The variable `Word` holds "Keyboard". Work out the value of each expression. [4]

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

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

Exercise 2.3 (d)

2.3 The variable Word holds "Keyboard". Work out the value of each expression. [4]

(d) `LENGTH(Word) DIV 3`

Word

K	e	y	b	o	a	r	d
---	---	---	---	---	---	---	---

Exercise 2.3 (d)

2.3 The variable Word holds "Keyboard". Work out the value of each expression. [4]

(d) `LENGTH(Word) DIV 3`

Word

K	e	y	b	o	a	r	d
---	---	---	---	---	---	---	---

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

Exercise 2.4 (a)

2.4 Work out the value of each expression.

[4]

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

Exercise 2.4 (a)

2.4 Work out the value of each expression.

[4]

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

(a) 'C' is two letters after 'A': $65 + 2 = \mathbf{67}$

Exercise 2.4 (b)

2.4 Work out the value of each expression.

[4]

(b) `CHR(70)`

Exercise 2.4 (b)

2.4 Work out the value of each expression.

[4]

(b) `CHR(70)`

(b) $70 - 65 = 5$ letters after 'A': 'F'

Exercise 2.4 (c)

2.4 Work out the value of each expression.

[4]

(c) "AB" & "12"

Exercise 2.4 (c)

2.4 Work out the value of each expression.

[4]

(c) "AB" & "12"

(c) & joins the strings: **"AB12"**

Exercise 2.4 (d)

2.4 Work out the value of each expression.

[4]

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

`'A'` → `65`

Exercise 2.4 (d)

2.4 Work out the value of each expression.

[4]

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

`'A'` → `65`

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

Exercises 2.5-2.6

2.5 Write the pseudocode statement that assigns to `Dice` a random integer from 1 to 6, and state why `INT` is needed. [2]



2.6 A student writes `Message ← "Total: " + Score`, where `Score` is an `INTEGER`. State what is wrong and write the correct statement. [2]

Total:

Exercises 2.5-2.6

2.5 Write the pseudocode statement that assigns to `Dice` a random integer from 1 to 6, and state why `INT` is needed. [2]



2.5

- `Dice ← INT(RAND(6)) + 1`
- `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

2.6 A student writes `Message ← "Total: " + Score`, where `Score` is an `INTEGER`. State what is wrong and write the correct statement. [2]

Total:

2.6

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

Exercise 2.7

2.7 Declare `BirthDate` with the correct type, assign 9 February 2008 using a library routine, and write a condition that tests whether it is a Thursday. [3]

09 / 02 / 2008

Exercise 2.7

2.7 Declare `BirthDate` with the correct type, assign 9 February 2008 using a library routine, and write a condition that tests whether it is a Thursday. [3]

09 / 02 / 2008

2.7

```
DECLARE BirthDate : DATE
BirthDate ← SETDATE(9, 2, 2008)
IF DAYINDEX(BirthDate) = 5 THEN
    OUTPUT "Thursday"
ENDIF
```

- DATE declaration; SETDATE with day, month, year in that order; DAYINDEX comparison with 5.

Exercise 3.1 (a)

3.1 Give the value of each expression.

[3]

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

Exercise 3.1 (a)

3.1 Give the value of each expression.

[3]

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

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

Exercise 3.1 (b)

3.1 Give the value of each expression.

[3]

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

Exercise 3.1 (b)

3.1 Give the value of each expression.

[3]

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

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

Exercise 3.1 (c)

3.1 Give the value of each expression.

[3]

(c) $\text{ASC}('Q') - \text{ASC}('A')$

P	r	o	g	r	a	m
---	---	---	---	---	---	---

C	o	m	p	u	t	e	r
---	---	---	---	---	---	---	---

Exercise 3.1 (c)

3.1 Give the value of each expression.

[3]

(c) $\text{ASC}('Q') - \text{ASC}('A')$

P	r	o	g	r	a	m
---	---	---	---	---	---	---

C	o	m	p	u	t	e	r
---	---	---	---	---	---	---	---

(c) 'Q' is 81 and 'A' is 65, so the answer is **16**

[4]

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

[4]

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

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

Exercise 3.3

3.3 Write pseudocode that reads a word and outputs it in **capitals** followed by its **length**, using built-in functions. [3]



Exercise 3.3

3.3 Write pseudocode that reads a word and outputs it in **capitals** followed by its **length**, using built-in functions. [3]



3.3

- INPUT, then TO_UPPER and LENGTH

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

Exercise 3.4

3.4 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`. [4]



Surname: Garcia
First name: Lina
Year: 12

Exercise 3.4

3.4 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`. [4]



Surname: Garcia
First name: Lina
Year: 12

3.4

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

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

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

Exercise 3.5

3.5 State **three** benefits of using a program library rather than writing the routines yourself. [3]

program library

Exercise 3.5

3.5 State **three** benefits of using a program library rather than writing the routines yourself. [3]

program library

3.5 any three:

- the routines have already been written, compiled and **tested**, so they are less likely to contain errors
- they save development time, because the code does not have to be written again
- 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

Exercise 3.6 (a)

3.6 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`.

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

[3]

Expression	Evaluates to
LEFT(<code>Code</code> , 2)	
STR_TO_NUM(MID(<code>Code</code> , 4, 4)) + 1	
LENGTH(<code>Code</code>) - 8	

Exercise 3.6 (a)

3.6 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`.

(a) Work out the value of each expression when `Code` is "TV-2025-0457". [3]

Expression	Evaluates to
<code>LEFT(Code, 2)</code>	
<code>STR_TO_NUM(MID(Code, 4, 4)) + 1</code>	
<code>LENGTH(Code) - 8</code>	

(a) `LEFT(Code, 2)` $\rightarrow$ "TV"

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

Exercise 3.6 (b)

3.6 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`.

(b) Write **one** pseudocode statement that stores the product number, as an `INTEGER`, in the variable `ProductNumber`. [3]

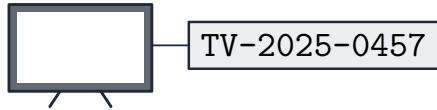


TV-2025-0457

Exercise 3.6 (b)

3.6 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`.

(b) Write **one** pseudocode statement that stores the product number, as an `INTEGER`, in the variable `ProductNumber`. [3]



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

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

Exercise 3.6 (c)

3.6 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**.

(c) 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**. [4]

Exercise 3.6 (c)

3.6 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`.

(c) 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`. [4]

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

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