Question	Part	Marking guidance	Total marks
01	1	**Mark is for AO2 (apply)** **D** 4; **If more than one lozenge shaded then mark is not awarded**	1
01	2	**Mark is for AO2 (apply)** **D** `'computer sciencegcse'`; **If more than one lozenge shaded then mark is not awarded**	1
01	3	**Mark is for AO2 (apply)** **C** `'sci'`; **If more than one lozenge shaded then mark is not awarded**	1
01	4	**Mark is for AO2 (apply)** **C** 101; **If more than one lozenge shaded then mark is not awarded**	1

Question	Part	Marking guidance	Total marks

02	1	**Mark is for AO3 (refine)**
		
		C#
		`string displayMessage = carReg + " is not valid";`
		
		Python
		`displayMessage = carReg + " is not valid"`
		
		VB.NET
		`Dim displayMessage As String = carReg + " is not valid"` //
		`Dim displayMessage As String = carReg & " is not valid"`
		
		I. Case
		I. Space between variable outputs
		I. Order of strings
			1

Question	Part	Marking guidance	Total marks
03	1	**Mark is for AO2 (apply)** **D** `S`; **R.** If more than one lozenge shaded	1

Question	Part	Marking guidance	Total marks
03	2	**Mark is for AO2 (apply)** **B** `2`; **R.** If more than one lozenge shaded	1

Question	Part	Marking guidance	Total marks
03	3	**Mark is for AO2 (apply)** `Sara`; **I.** Case	1

Question	Part	Marking guidance	Total marks

03	4	**2 marks for AO3 (program)**
		
		Mark A for correct identification of **2, 4**;
		Mark B for correct identification of **1**;
		
		<u>Model Answer</u>
		`var ← SUBSTRING(`**<u>`2, 4,`</u>**` name1)`
		`OUTPUT (names[`**<u>`1`</u>**`] + var)`
			2

Question	Part	Marking guidance	Total marks
04	**1**	**2 marks for AO1 (recall)** A sequence / series of steps / instructions; (that can be followed) to complete a task / to solve a problem; **A.** set of instructions / steps	2

Question	Part	Marking guidance	Total marks
04	**2**	**Mark is for AO2 (apply)** **C** `10`; **R.** if more than one lozenge shaded	1

Question	Part	Marking guidance	Total marks
04	**3**	**Mark is for AO2 (apply)** **D** `San FranciscoAlcatraz Island`; **R.** if more than one lozenge shaded	1

Question	Part	Marking guidance	Total marks
04	**4**	**Mark is for AO2 (apply)** **D** `traz`; **R.** if more than one lozenge shaded	1

Question	Part	Marking guidance	Total marks
04	**5**	**Mark is for AO2 (apply)** **C** `4`; **R.** if more than one lozenge shaded	1

Question	Part	Marking guidance	Total marks
05	1	**Mark is for AO2 (apply)** **D** 4; **R.** If more than one lozenge shaded	1
05	2	**Mark is for AO2 (apply)** **D** `'computer sciencegcse'`; **R.** If more than one lozenge shaded	1

Question	Part	Marking guidance	Total marks
06		**3 marks for AO3 (design), 4 marks for AO3 (program)** **<u>Program Design</u>** **Mark A** for the idea of inputting a character and checking if it is lower case (even if the code would not work); **Mark B** for the use of a selection construct (even if the logic is incorrect); **Mark C** for the correct, consistent use of meaningful variable names throughout (even if the code would not work); **<u>Program Logic</u>** **Mark D** for using user input correctly; **Mark E** for storing the result of user input in a variable correctly; **Mark F** for a correct expression/method that checks if the character is lowercase; **Mark G** for outputting `LOWER` and `NOT LOWER` correctly in logically separate places such as the `IF` and `ELSE` part of selection; **I.** Case of output strings for **Mark G**, but spelling must be correct. **I.** Case of program code **Maximum 6 marks** if any errors in code. **<u>Python Example 1 (fully correct)</u>** All design marks are achieved (**Marks A, B and C**) `character = input()` (**D,E**) `if (character >= 'a') and (character <= 'z'):` (**F**) `print('LOWER')` (**Part of G**) `else:` `print('NOT LOWER')` (**Part of G**) **<u>Python Example 2 (fully correct)</u>** All design marks are achieved (**Marks A, B and C**) `character = input()` (**D,E**) `if character.islower():` (**F**) `print('LOWER')` (**Part of G**) `else:` `print('NOT LOWER')` (**Part of G**)	7

C# Example (fully correct)
All design marks are achieved (**Marks A, B and C**)

```
char character = (char)Console.Read();              (D,E)
if (Char.IsLower(character))                        (F)
{
 Console.WriteLine("LOWER");                        (Part of G)
}
else
{
 Console.WriteLine("NOT LOWER");                    (Part of G)
}
```

I. indentation in C#

VB.Net Example (fully correct)
All design marks are achieved (**Marks A, B and C**)

```
Dim character As Char
character = Console.ReadLine()                      (D,E)
If (Char.IsLower(character)) Then                   (F)
  Console.WriteLine("LOWER")                        (Part of G)
Else
  Console.WriteLine("NOT LOWER")                    (Part of G)
End If
```

I. indentation in VB.NET

Python Example 3 (partially correct – 5 marks)
All design marks are achieved (**Marks A, B and C**)

```
character = input()                                 (D,E)
if (character > 'a') or (character < 'z'):          (NOT F)
   print('NOT LOWER')                               (NOT G)
else:
   print('LOWER')                                   (NOT G)
```

Question	Part	Marking guidance	Total marks
07	1	**Mark is for AO2 (apply)** **D** `value ← LEN(film);` **R.** If more than one lozenge shaded	1

Question	Part	Marking guidance	Total marks
07	2	**Mark is for AO2 (apply)** `POSITION(film, letter);` **I.** Case **R.** Quotes	1

Question	Part	Marking guidance	Total marks
07	3	**Mark is for AO2 (apply)** **C** integer; **R.** If more than one lozenge shaded	1

Question	Part	Marking guidance	Total marks
07	4	**Mark is for AO1 (understanding)** When a value is given to a variable; // When a variable is assigned a value;	1

Question	Part	Marking guidance	Total marks
07	5	**2 marks for AO3 (program)** **Program Logic** **Mark A** for using user input and storing the result in a variable; **Mark B** for displaying `You entered` followed by the name of the film entered by the user in the appropriate place; **I.** Case **I.** Indentation **I.** Messages or no messages with input statements **I.** Gaps/spaces throughout the code, except where to do so would explicitly alter the logic of the code in a way that makes it incorrect **Maximum 1 mark** if any errors in code. **Note to examiners** In C#/VB.NET examples, explicit variable declarations are not shown. Refer to the specific language type issues section of the appropriate Marking guidance document. Any correct variable declarations in student answers should be accepted. **C# Example 1 (fully correct)** `film = Console.ReadLine();` **(A)** `Console.WriteLine("You entered " + film);` **(B)** **A.** `Write` in place of `WriteLine` **C# Example 2 (fully correct)** `film = Console.ReadLine();` **(A)** `Console.Write("You entered ");` **(Part B)** `Console.WriteLine(film);` **(Part B)** **Python Example 1 (fully correct)** `film = input()` **(A)** `print("You entered", film)` **(B)** **Python Example 2 (fully correct)** `film = input()` **(A)** `print("You entered " + film)` **(B)**	2

Python Example 3 (fully correct)

```
film = input()                                   (A)
print(f"You entered {film}")                     (B)
```

VB.NET Example 1 (fully correct)

```
film = Console.ReadLine()                        (A)
Console.WriteLine("You entered " & film)         (B)
```

A. `Write` in place of `WriteLine`

VB.NET Example 2 (fully correct)

```
film = Console.ReadLine()                        (A)
Console.WriteLine("You entered " + film)         (B)
```

A. `Write` in place of `WriteLine`

VB.NET Example 3 (fully correct)

```
film = Console.ReadLine()                      (A)
Console.Write("You entered ")                  (Part B)
Console.WriteLine(film)                        (Part B)
```

A. `Write` in place of `WriteLine`