Question number	Answer	Additional guidance	Mark
2	The following assessment objectives are assessed: • AO2.1a • AO2.1b • AO3.1 • AO3.2a • AO3.2b • AO3.2c Award marks as shown. • Add ':' at end of the line: `if (choice == 'y'):` (1) • Add missing ')' before ':' in the line: `for num in range(5, -1, -1):` (1) • Add missing " before end bracket in the line: `print("Goodbye")` (1) • Printing a suitable question for the user based on context, i.e. "Do you want me to sing?" (1) • Accept user input of 'y' and 'n' (1) • Changing the variable name 'x' to a more meaningful name (1) such as 'choice' throughout the code • Addition of comment indicating reverse stepping (1) • One mark each for insertion of white space to aid readability, up to a maximum of two marks (2) • Correct output for 'y' (count down 5 to 0 and then Goodbye) and correct output for 'n' (Goodbye) (1)		**(10)**

```
# ------------------------------------------------------------------
# Global variables
# ------------------------------------------------------------------

# ===>  Change the identifier x to a more meaningful name
choice = ""

# ------------------------------------------------------------------
# Main program
# ------------------------------------------------------------------
# ===> Display a suitable question to the user
print ("Would you like me to sing?")

# ===> Accept the user's input (no validation required)
choice = input("Choose 'y' for yes and 'n' for no")

if (choice == 'y'):
    # ===> Add a comment to explain the effect of the last -1 in this call
    # Counting backwards by using step as -1
    for num in range(5, -1, -1):
        print(num, "green bottles sitting on the wall")

print("Goodbye")

```

Question number	Answer	Additional guidance	Mark
3	The following assessment objectives are assessed: • AO2.1b • AO3.1 • AO3.2a • AO3.2b • AO3.2c Award marks as shown. • Fixing runtime error by coercion of input to 'int' (1) • Fixing errors by using modulus (1) • Use of at least one appropriate 'if' statement in the solution (1) • Adding validation for input numbers using: o relational operator (<=20) (1) o relational operator (>=1) (1) o correct Boolean operator (and/or) (1) • Corrects output message for even numbers and odd numbers (1) Levels-based mark scheme to a maximum of 6, from: • Solution design (3) • Functionality (3)	• Fixing error with odd numbers can be done in several different ways (see examples) • Award any accurate tests for validation range Considerations: • 6.1.6 Using test data to evaluate a program, such as extreme data [a character], normal data [1...20] and boundary data [0, 21] • 6.2.2 Appropriate use of sequencing, selection and repetition • 6.1.1 Use analysis to solve problems • 6.1.6 Use logical reasoning to evaluate efficiency (i.e. reduce tests)	**(13)**

Solution design (levels-based mark scheme)

0	1	2	3	Max.
No rewardable material	• There has been little attempt to decompose the problem. • Some of the component parts of the problem can be seen in the solution, although this will not be complete. • Some parts of the logic are clear and appropriate to the problem. • The use of variables and data structures, appropriate to the problem, is limited. • The choice of programming constructs, appropriate to the problem, is limited.	• There has been some attempt to decompose the problem. • Most of the component parts of the problem can be seen in the solution. • Most parts of the logic are clear and appropriate to the problem. • The use of variables and data structures is mostly appropriate. • The choice of programming constructs is mostly appropriate to the problem.	• The problem has been decomposed clearly into component parts. • The component parts of the problem can be seen clearly in the solution. • The logic is clear and appropriate to the problem. • The choice of variables and data structures is appropriate to the problem. • The choice of programming constructs is accurate and appropriate to the problem.	**3**

Functionality (levels-based mark scheme)

0	1	2	3	Max.
No rewardable material	**Functionality (when the code is run)** • The component parts of the program are incorrect or incomplete, providing a program of limited functionality that meets some of the given requirements. • Program outputs are of limited accuracy and/or provide limited information. • Program responds predictably to some of the anticipated input. • Solution is not robust and may crash on anticipated or provided input.	**Functionality (when the code is run)** • The component parts of the program are complete, providing a functional program that meets most of the stated requirements. • Program outputs are mostly accurate and informative. • Program responds predictably to most of the anticipated input. • Solution may not be robust within the constraints of the problem.	**Functionality (when the code is run)** • The component parts of the program are complete, providing a functional program that fully meets the given requirements. • Program outputs are accurate, informative, and suitable for the user. • Program responds predictably to anticipated input. • Solution is robust within the constraints of the problem.	**3**

```
# ------------------------------------------------------------------------
# Global variables
# ------------------------------------------------------------------------
num = 0

# ------------------------------------------------------------------------
# Main program
# ------------------------------------------------------------------------

# ---------- Solution 1 ----------
# Coerce the input from string to integer
num = int (input ("Please enter a number (1 ... 20)"))

# Check for valid input numbers
if (num >=1 and num <= 20):
    if (num % 2 == 0):
        print (num, "is even")
    else:
        # If num is not even, then it must be odd
        print (num, "is odd")
else:
    print ("Invalid input")      # Error message for bad input

```

```
24  # ---------- Solution 2 ----------
25  num = int (input ("Please enter a number (1 ... 20)"))
26
27  if (num < 1):
28      print ("Bad input")
29  elif (num > 20):
30      print ("Bad input")
31  elif (num % 2 == 0):
32      print (num, "is even")
33  else:
34      print (num, "is odd")
35
36  # ---------- Solution 3 ----------
37  num = int (input ("Please enter a number (1 ... 20)"))
38
39  if (num >= 1) and (num <= 20):
40      if (num % 2 == 0):
41          print (num, " is even")
42      elif (num %2 != 0):
43          # Extra check for odd
44          print (num, " is odd")
45  else:
46      print ("Bad input")      # Error message for the user
47
48  # ---------- Solution 4 ----------
49  num = int (input ("Please enter a number (1 ... 20)"))
50
51  # Keeps looping until a good input is identified
52  while (num < 1) or (num > 20):
53      print ("Invalid input")
54      num = int (input ("Please enter a number (1 ...20)"))
55
56  if (num % 2 == 0):
57      print (num, "is even")
58  else:
59      print (num, "is odd")
60
```

Question number	Answer	Additional guidance	Mark
4	The following assessment objectives are assessed: • AO2.1b • AO3.1 • AO3.2a • AO3.2b • AO3.2c Award marks as shown. • Use of comments, white space and layout to aid readability (1) • Initial input done outside loop, to handle first entry is '0' (1) • Repetition (while) used as outermost loop (1) • 'elif (year > 13)' is placed later in the logic than 'if (year < 1)' (1) • 'elif (year < 12)' is placed later in the logic than 'elif (year < 7)' (1) • Accepting next round of input done inside loop (1) • Validation messages match validation tests: o Year too small (1) o Year too big (1) • Institution messages match tests: o Primary (1) o Secondary (1) o College (1) • Correct outputs for each set of test data: o 0 = exiting (1) o 1 and 6 = Primary (1) o 7 and 11 = Secondary (1) o 12 = College (1)		**(15)**

```
# ------------------------------------------------------------------
# Global variables
# ------------------------------------------------------------------
year = 0            # Do not move this line
strYear = ""        # Do not move this line

# ------------------------------------------------------------------
# Main program
# ------------------------------------------------------------------

# Put the lines into the correct order to solve the problem.
# A user types in a year group.  The program indicates which stage
# of education the year group belongs to.  The program loops until
# the user enters 0.
# Example:
# Input                 Output
# ---------------------------------------
# 0                     Exits program
# 1, 2, 3, 4, 5, 6      Primary
# 7, 8, 9, 10, 11       Secondary
# 12, 13                College

# ---------- Solution 1 ----------
# Prime the loop, just in case the first entry is '0'
strYear = input ("Enter year group (1 to 13, 0 to exit)")
year = int (strYear)

# Keep looping until user wants to stop
while (year != 0):
    # Validate input as a real year group
    if (year < 1):
        print ("Year too small")
    elif (year > 13):
        print ("Year too big")
    elif (year < 7):
        print ("Primary")
    elif (year < 12):
        print ("Secondary")
    else:
        print ("College")

    # Get a new input before going to top of loop
    strYear = input("Enter year group (1 to 13, 0 to exit)")
    year = int(strYear)

```

```
# ---------- Solution 2 ----------
# Prime the loop, just in case the first entry is '0'
strYear = input ("Enter year group (1 to 13, 0 to exit)")
year = int (strYear)

# Keep looping until user wants to stop
while (year != 0):
    # Validate input as a real year group
    if (year < 1):
        print ("Year too small")
    elif (year < 7):
        print ("Primary")
    elif (year < 12):
        print ("Secondary")
    elif (year > 13):
        print("Year too big")
    else:
        print ("College")

    # Get a new input before going to top of loop
    strYear = input("Enter year group (1 to 13, 0 to exit)")
    year = int(strYear)

```

Question number	Answer	Additional guidance	Mark
5	The following assessment objectives are assessed: • AO2.1b • AO3.1 • AO3.2a • AO3.2b • AO3.2c Award marks as shown. • Import of math library (1) • Two parameters in first line of subprogram definition (1) with names 'pRadius' and 'pHeight', in any order (1) • Accurate translation of the formula to code (1) • Use of math.pi constant in formula translation (1) • Two passed-in parameters ('pRadius' and 'pHeight') used in the calculation (1) • Assignment of calculation to 'theVolume' (1) • One return statement with 'theVolume' in brackets (1) • Parameters in call to subprogram are 'baseRadius' and 'coneHeight', in any order (1) • Order of parameters matches order in first line of subprogram definition (1) • Capture of returned value in main program, in 'coneVolume' (1) • Format volume to three decimal places for outputting only (1) Levels-based mark scheme to a maximum of 3, from: • Functionality (3)	Considerations: • 6.1.1 Be able to use decomposition to analyse requirements • 6.1.2 Be able to write in a high-level language • 6.6.1 Be able to perform generalisations • Default printing will drop trailing 0s, even if rounded, so string formatting should be used	**(15)**

Functionality (levels-based mark scheme)

0	1	2	3	Max.
No rewardable material	**Functionality (when the code is run)** • The component parts of the program are incorrect or incomplete, providing a program of limited functionality that meets some of the given requirements. • Program outputs are of limited accuracy and/or provide limited information. • Program responds predictably to some of the anticipated input. • Solution is not robust and may crash on anticipated or provided input.	**Functionality (when the code is run)** • The component parts of the program are complete, providing a functional program that meets most of the stated requirements. • Program outputs are mostly accurate and informative. • Program responds predictably to most of the anticipated input. • Solution may not be robust within the constraints of the problem.	**Functionality (when the code is run)** • The component parts of the program are complete, providing a functional program that fully meets the given requirements. • Program outputs are accurate, informative, and suitable for the user. • Program responds predictably to anticipated input. • Solution is robust within the constraints of the problem.	**3**

```
# ---------------------------------------------------------------------
# Import libraries
# ---------------------------------------------------------------------
# ===> Import a library to use Pi
import math

# ---------------------------------------------------------------------
# Global variables
# ---------------------------------------------------------------------

# Hard coded for testing
coneHeight = 10.7
baseRadius = 1.2
coneVolume = 0.0

# ---------------------------------------------------------------------
# Subprograms
# ---------------------------------------------------------------------
# ===> Add parameters inside the brackets
def calcVolume (pRadius, pHeight):

    print ("The radius is:", pRadius)
    print ("The height is:", pHeight)

    # ===> Complete the calculation for the volume
    theVolume = 1/3 * math.pi * math.pow (pRadius, 2) * pHeight
    # theVolume = 1/3 * math.pi * pRadius**2 * pHeight
    # theVolume = 1/3 * math.pi * pRadius * pRadius * pHeight

    print ("The volume is:", theVolume)

    # ===> Return the volume to the caller
    return (theVolume)

# ---------------------------------------------------------------------
# Main program
# ---------------------------------------------------------------------

# ===> Call the subprogram, passing parameters,
#      and catch the returned value in the correct variable
coneVolume = calcVolume (baseRadius, coneHeight)

# ===> Print the total volume to three decimal places using string.format()
# ===> by completing the pattern inside the { }
print ("{:.3f}".format(coneVolume))

```

Question number	Answer	Additional guidance	Mark
6	The following assessment objectives are assessed: • AO2.1b • AO3.1 • AO3.2a • AO3.2b • AO3.2c Award marks as shown. Points-based mark scheme: **Inputs** • Accepts and responds to user input (1) • Validation with range check using relational operators >=1000, <=9999 (1) **Process** • Use of library subprograms len() (1) to work with any number of users in the list • Use of Boolean (1) to stop loop when found or passed over • Use of 2-dimensional indexing (1) in user list **Outputs** • Display of appropriate messages (1) Levels-based mark scheme to a maximum of 9, from: • Solution design (3) • Good programming practices (3) • Functionality (3)	Considerations: • 6.1.1 Use decomposition and abstraction to analyse a problem (inputs, outputs, processing, initialisation, design) • 6.6.1 Decompose into subproblems • 6.1.2 Write in a high-level language • 6.2.2 Use sequencing and selection components	**(15)**

Solution design (levels-based mark scheme)

0	1	2	3	Max.
No rewardable material	• There has been little attempt to decompose the problem. • Some of the component parts of the problem can be seen in the solution, although this will not be complete. • Some parts of the logic are clear and appropriate to the problem. • The use of variables and data structures, appropriate to the problem, is limited. • The choice of programming constructs, appropriate to the problem, is limited.	• There has been some attempt to decompose the problem. • Most of the component parts of the problem can be seen in the solution. • Most parts of the logic are clear and appropriate to the problem. • The use of variables and data structures is mostly appropriate. • The choice of programming constructs is mostly appropriate to the problem.	• The problem has been decomposed clearly into component parts. • The component parts of the problem can be seen clearly in the solution. • The logic is clear and appropriate to the problem. • The choice of variables and data structures is appropriate to the problem. • The choice of programming constructs is accurate and appropriate to the problem.	**3**

Good programming practices (levels-based mark scheme)

0	1	2	3	Max.
No rewardable material	• There has been little attempt to lay out the code into identifiable sections to aid readability. • Some use of meaningful variable names. • Limited or excessive commenting. • Parts of the code are clear, with limited use of appropriate spacing and indentation.	• There has been some attempt to lay out the code to aid readability, although sections may still be mixed. • Uses mostly meaningful variable names. • Some use of appropriate commenting, although may be excessive. • Code is mostly clear, with some use of appropriate white space to aid readability.	• Layout of code is effective in separating sections, e.g. putting all variables together, putting all subprograms together as appropriate. • Meaningful variable names and subprogram interfaces are used where appropriate. • Effective commenting is used to explain logic of code blocks. • Code is clear, with good use of white space to aid readability.	**3**

Functionality (levels-based mark scheme)

0	1	2	3	Max.
No rewardable material	**Functionality (when the code is run)** • The component parts of the program are incorrect or incomplete, providing a program of limited functionality that meets some of the given requirements. • Program outputs are of limited accuracy and/or provide limited information. • Program responds predictably to some of the anticipated input. • Solution is not robust and may crash on anticipated or provided input.	**Functionality (when the code is run)** • The component parts of the program are complete, providing a functional program that meets most of the stated requirements. • Program outputs are mostly accurate and informative. • Program responds predictably to most of the anticipated input. • Solution may not be robust within the constraints of the problem.	**Functionality (when the code is run)** • The component parts of the program are complete, providing a functional program that fully meets the given requirements. • Program outputs are accurate, informative, and suitable for the user. • Program responds predictably to anticipated input. • Solution is robust within the constraints of the problem.	**3**

```
# ------------------------------------------------------------------
# Global variables
# ------------------------------------------------------------------

# User Number, Last Name, First Name, Login Name, Passcode
userList = [[110,"Cashin","Bonnie","Cae110",7005],
            [101,"Cheruit","Madeleine","Che101",1507],
            [103,"Chanel","Coco","Cho103",7333],
            [107,"Gres","Madame","Gre107",3054],
            [114,"Hamnett","Katharine","Hae114",4807],
            [118,"Herrera","Carolina","Hea118",5567],
            [111,"Hulanicki","Barbara","Hua111",5125],
            [116,"Johnson","Betsey","Joy116",8869],
            [104,"Lanvin","Jeanne","Lae104",8580],
            [109,"McCardell","Claire","Mce109",5991],
            [102,"Paquin","Jeanne","Pae102",6495],
            [112,"Quant","Mary","Quy112",9028],
            [113,"Rykiel","Sonia","Rya113",1177],
            [105,"Schiaparelli","Elsa","Sca105",2980],
            [108,"Schlee","Valentina","Sca108",6801],
            [106,"Vionnet","Madeleine","Vie106",9042],
            [117,"Von Furstenberg","Diane","Voe117",2553],
            [119,"Wang","Vera","Waa119",2004],
            [115,"Westwood","Vivienne","Wee115",7806]]

inID = ""                # String
inPass = 0               # Integer
found = False            # Haven't found the record yet
passed = False           # Haven't gone past where it should be
index = 0                # The current record being looked at

```

```
# ------------------------------------------------------------------
# Main program
# ------------------------------------------------------------------

# Get user login name
inID = input ("Enter your user login name, type X to exit.")

# Get user passcode
inPass = int (input ("Enter your four digit passcode"))

# Check if passcode is valid
if (inPass >= 1000 and inPass <= 9999):
    # Look through userList to find matching set
    while (found == False and passed == False and index < len(userList)):
        # If both parts match (authenticated), display welcome message
        if (userList[index][3] == inID and userList[index][4] == inPass):
            found = True
            print ("Welcome", userList[index][2], userList[index][1])
        # Check if have passed over where it should be in the list
        elif (userList[index][3] > inID):
            passed = True               # Stops looping
        else:
            index = index + 1           # Look at next entry
    # If not found or passed, display "Invalid Login Credentials"
    if (found == False):
        print ("Invalid Login Credentials")
else:
    print ("Passcode must be four digits long")

```