L3 Lead Examiner Report 2001

January 2020

L3 Qualification in Computing

Unit 4: Software Design and Development Project

Edexcel and BTEC Qualifications

Edexcel and BTEC qualifications come from Pearson, the world's leading learning company. We provide a wide range of qualifications including academic, vocational, occupational and specific programmes for employers. For further information visit our qualifications website at http://qualifications.pearson.com/en/home.html for our BTEC qualifications.

Alternatively, you can get in touch with us using the details on our contact us page at http://qualifications.pearson.com/en/contact-us.html

If you have any subject specific questions about this specification that require the help of a subject specialist, you can speak directly to the subject team at Pearson. Their contact details can be found on this link:
http://qualifications.pearson.com/en/support/support-for-you/teachers.html

You can also use our online Ask the Expert service at https://www.edexcelonline.com
You will need an Edexcel Online username and password to access this service.

Pearson: helping people progress, everywhere

Our aim is to help everyone progress in their lives through education. We believe in every kind of learning, for all kinds of people, wherever they are in the world. We've been involved in education for over 150 years, and by working across 70 countries, in 100 languages, we have built an international reputation for our commitment to high standards and raising achievement through innovation in education. Find out more about how we can help you and your learners at: www.pearson.com/uk

January 2020
31771_2001_ER

Grade Boundaries

What is a grade boundary?

A grade boundary is where we set the level of achievement required to obtain a certain grade for the externally assessed unit. We set grade boundaries for each grade, at Distinction, Merit and Pass.

Setting grade boundaries

When we set grade boundaries, we look at the performance of every learner who took the external assessment. When we can see the full picture of performance, our experts are then able to decide where best to place the grade boundaries – this means that they decide what the lowest possible mark is for a particular grade.

When our experts set the grade boundaries, they make sure that learners receive grades which reflect their ability. Awarding grade boundaries is conducted to ensure learners achieve the grade they deserve to achieve, irrespective of variation in the external assessment.

Variations in external assessments

Each external assessment we set asks different questions and may assess different parts of the unit content outlined in the specification. It would be unfair to learners if we set the same grade boundaries for each assessment, because then it would not take accessibility into account.

Grade boundaries for this, and all other papers, are on the website via this link:

http://qualifications.pearson.com/en/support/support-topics/results-certification/grade-boundaries.html

Unit 4: Software Design and Development Project

Grade	Unclassified	N grade	Level 3		
			Pass	Merit	Distinction
Boundary Mark	0	10	21	35	50

Introduction

This was the fourth examination series for Level 3 BTEC Computing Unit 4: Software Design and Development Project.
This unit is a paper-based exam, assessed through a task-based assessment. The set task assesses learners' ability to design, create and evaluate software using Python (3.4 or a later version) or one of the C family programming languages. This unit is a mandatory unit for all learners studying the extended diploma.

The examination for this unit will always contain five activities and each one will be linked to a scenario. The scenario is clearly stated at the beginning of each assessment. The activities will test learners on different areas of the specification, and learners are expected to apply their knowledge to the scenario.

All Activities of the examination paper provide differentiation at all attainment levels and the brief is designed to escalate in difficulty so that a larger percentage of higher-grade marks depends on the skills, knowledge, understanding and application of theory.

Introduction to the Overall Performance of the Unit

The overall performance of learners was not as good compared to the previous series for this unit. It was evident that some learners were not well prepared for the rigour of this assessment.

The performance on Activity 1 was as expected with many learners picking up marks in band 2. Most of the responses used BCS symbols and had a good go at breaking down the requirements into relevant parts. Learners provided evidence of links between component parts but little evidence of handling errors within the flowcharts.

Activity 2 was of a good standard and demonstrated the learner ability to apply pseudocode design methodologies to a scenario. Learners have taken on board previous comments regarding this activity and the number of pseudocode being too close to the coding was much less compared to January and June 2019.

Activity 3 & 4 (testing) was poor again this series and resulted in most learners only accessing band 1. It is recommended that centres reinforce what a test plan consists of and the importance of testing throughout the whole design and development process. Some test plans seen did not include any test data which meant no marks could be awarded. In most cases, the testing carried out did not evidence any errors encountered which is essential for accessing higher marks.

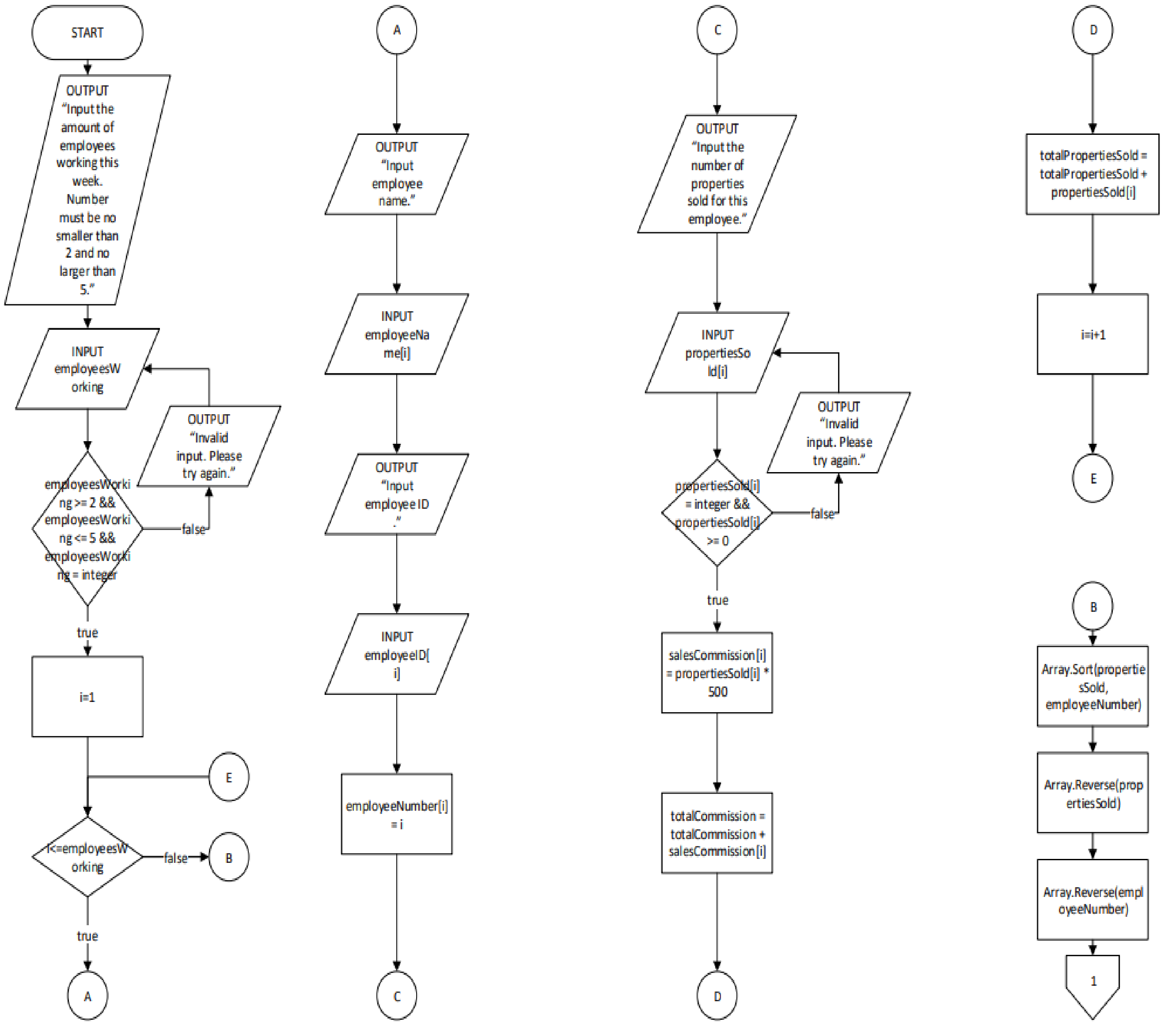
Activity 4 (Coding) was not done to a good standard by a lot of the learners. Some were awarded marks in the top mark band as they produced a working solution along with detailed comments, but most learners produced code that was mark band 2 standard at best. Some learners had not been fully prepared for coding in any programming language and only managed to produce a solution that accepted inputs.

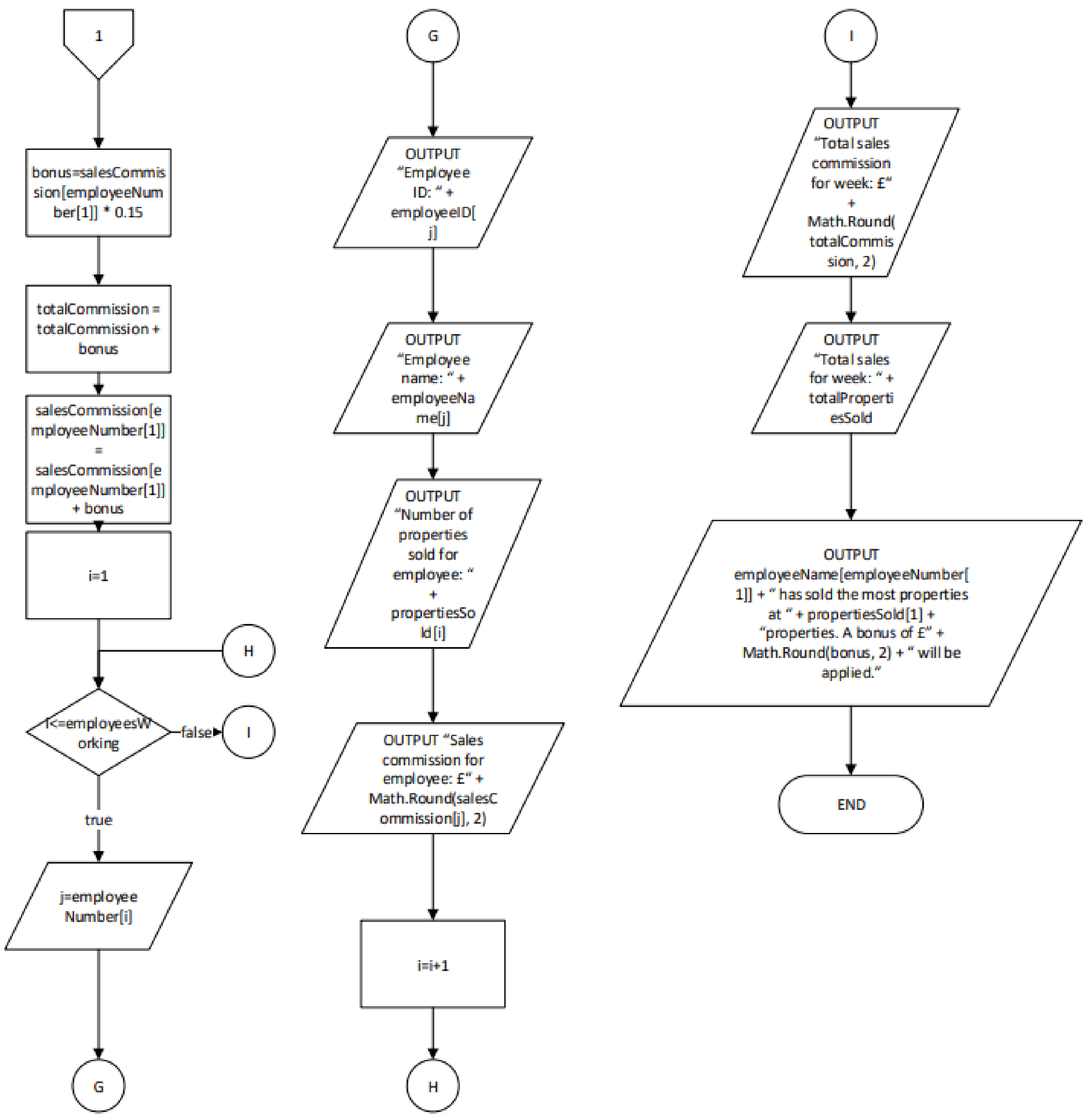
The evaluations (activity 5) were of a good standard and most learner’s accessed bands two and three. Some learners only produced a review of what they did which resulted in marks from band 1 being awarded.

Individual Questions

The following section considers each question on the paper, providing examples of learner responses and a brief commentary of why the responses gained the marks they did. This section should be considered with the live external assessment and the corresponding mark scheme.

Activity 1

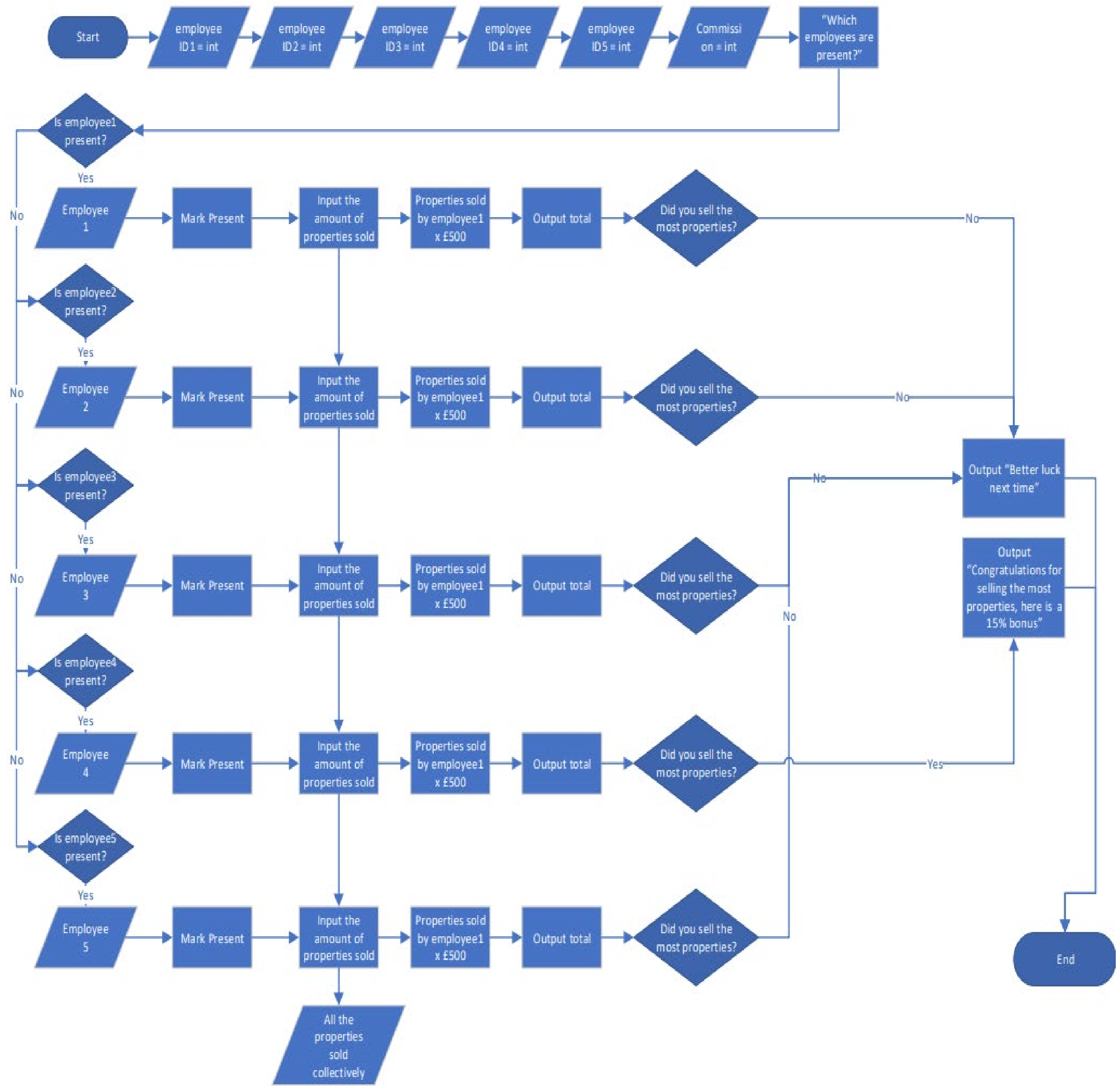

START
OUTPUT "Input the amount of employees working this week. Number must be no smaller than 2 and no larger than 5."
INPUT employeesWorking
employeesWorking >= 2 && employeesWorking <= 5 && employeesWorking = integer
false
OUTPUT "Invalid input. Please try again."
true
i=1
E
i<=employeesWorking
false
B
true
A
A
OUTPUT "Input employee name."
INPUT employeeName[i]
OUTPUT "Input employee ID."
INPUT employeeID[i]
employeeNumber[i] = i
C
C
OUTPUT "Input the number of properties sold for this employee."
INPUT propertiesSold[i]
propertiesSold[i] = integer && propertiesSold[i] >= 0
false
OUTPUT "Invalid input. Please try again."
true
salesCommission[i] = propertiesSold[i] * 500
totalCommission = totalCommission + salesCommission[i]
D
D
totalPropertiesSold = totalPropertiesSold + propertiesSold[i]
i=i+1
E
B
Array.Sort(propertiesSold, employeeNumber)
Array.Reverse(propertiesSold)
Array.Reverse(employeeNumber)
1


1
bonus=salesCommission[employeeNumber[1]] * 0.15
totalCommission = totalCommission + bonus
salesCommission[employeeNumber[1]] = salesCommission[employeeNumber[1]] + bonus
i=1
H
i<=employeesWorking
false
I
true
j=employeeNumber[i]
G
G
OUTPUT "Employee ID: " + employeeID[j]
OUTPUT "Employee name: " + employeeName[j]
OUTPUT "Number of properties sold for employee: " + propertiesSold[i]
OUTPUT "Sales commission for employee: £" + Math.Round(salesCommission[j], 2)
i=i+1
H
I
OUTPUT "Total sales commission for week: £" + Math.Round(totalCommission, 2)
OUTPUT "Total sales for week: " + totalPropertiesSold
OUTPUT employeeName[employeeNumber[1]] + " has sold the most properties at " + propertiesSold[1] + "properties. A bonus of £" + Math.Round(bonus, 2) + " will be applied."
END

The learner has produced a flow chart that addresses the specified problem.

British Computer Society (BCS) flowchart symbols have been used accurately throughout as well as the breaking down of requirements into component parts that are detailed and relevant.

The flowchart shows full coverage of inputs, outputs and processes using naming conventions appropriate to the scenario consistently. Links between component parts are complete and efficient with accurate and robust procedures for handling unexpected events. **Band 3 (10 marks).**



Start
employee ID1 = int
employee ID2 = int
employee ID3 = int
employee ID4 = int
employee ID5 = int
Commissi on = int
"Which employees are present?"
Is employee1 present?
Yes
No
Employee 1
Mark Present
Input the amount of properties sold
Properties sold by employee1 x £500
Output total
Did you sell the most properties?
No
Is employee2 present?
Yes
No
Employee 2
Mark Present
Input the amount of properties sold
Properties sold by employee1 x £500
Output total
Did you sell the most properties?
No
Is employee3 present?
Yes
No
Employee 3
Mark Present
Input the amount of properties sold
Properties sold by employee1 x £500
Output total
Did you sell the most properties?
No
No
Is employee4 present?
Yes
No
Employee 4
Mark Present
Input the amount of properties sold
Properties sold by employee1 x £500
Output total
Did you sell the most properties?
Yes
Is employee5 present?
Yes
Employee 5
Mark Present
Input the amount of properties sold
Properties sold by employee1 x £500
Output total
Did you sell the most properties?
All the properties sold collectively
Output "Better luck next time"
Output "Congratulations for selling the most properties, here is a 15% bonus"
End

The learner has addressed some aspects of the flow chart for the specified problem. British Computer Society (BCS) flowchart symbols have been used but mostly incorrect with irrelevant parts. There is some evidence of breaking down of requirements into component parts that are relevant. Links between component parts are incomplete with limited procedures for handling unexpected events.

Mark in **band 1 (3 marks)**.

Activity 2

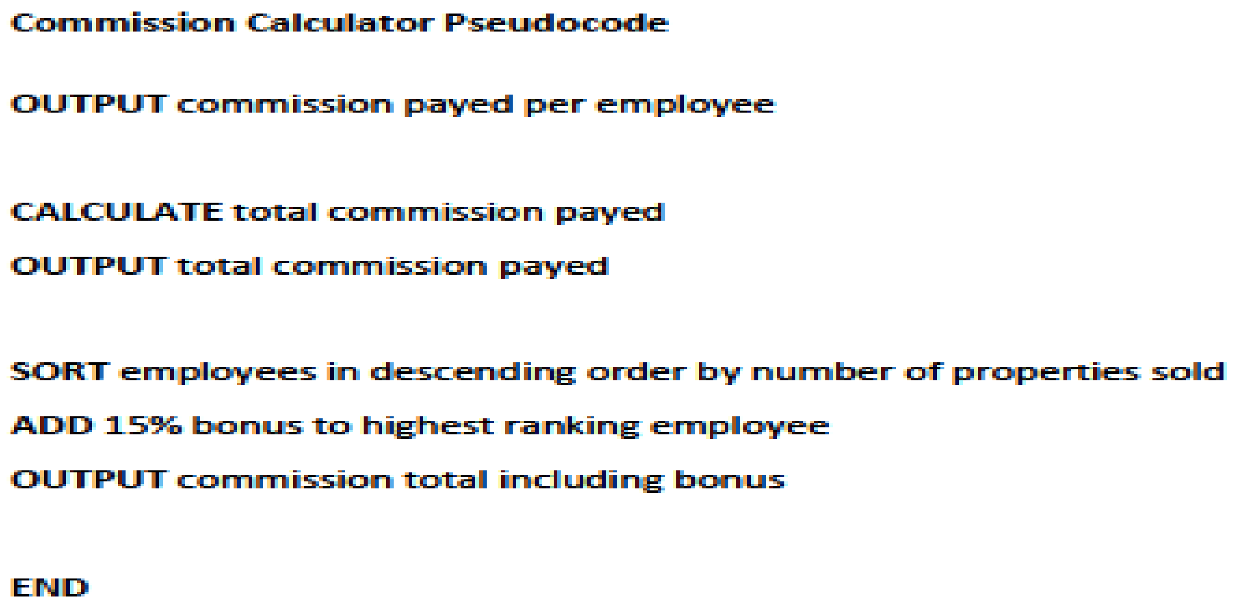
```
START

        OUTPUT "Input the amount of employees working this week. Number must be no smaller
than 2 and no larger than 5."

        employeesWorking = 0
        WHILE employeesWorking < 2 || employeesWorking > 5 || employeesWorking != int
                INPUT employeesWorking
                IF employeesWorking < 2 || employeesWorking > 5 || employeesWorking != int
                        OUTPUT "Invalid amount of employees. Please try again."
                END IF
        END WHILE

        FOR i=1; i<=employeesWorking; i++
                OUTPUT "Input employee name."
                INPUT employeeName[i]

                OUTPUT "Input employee ID."
                INPUT employeeID[i]

                employeeNumber[i] = i

                propertiesSold[i] = "placeholder"
                OUTPUT "Input the number of properties sold for this employee."
                WHILE propertiesSold[i] != int || propertiesSold < 0
                        INPUT propertiesSold[i]
                        IF propertiesSold[i] != int || propertiesSold < 0
                                OUTPUT "Invalid number of properties sold. Please try again."
                        END IF
                END WHILE
        salesCommission[i] = propertiesSold[i] * 500
        totalCommission = totalCommission + salesCommission[i]
        totalPropertiesSold = totalPropertiesSold + propertiesSold[i]

    END FOR

    Array.Sort(propertiesSold, employeeNumber)
    Array.Reverse(propertiesSold)
    Array.Reverse(employeeNumber)

    bonus=salesCommission[employeeNumber[1]] * 0.15
    totalCommission = totalCommission + bonus
    salesCommission[employeeNumber[1]] = salesCommission[employeeNumber[1]] + bonus

    FOR i=1; i<=employeesWorking; i++
        j=employeeNumber[i]
        OUTPUT "Employee ID: " + employeeID[j]
        OUTPUT "Employee name: " + employeeName[j]
        OUTPUT "Number of properties sold for employee: " + propertiesSold[i]
        OUTPUT "Sales commission for employee: £" + Math.Round(salesCommission[j], 2)
    END FOR

    OUTPUT "Total sales commission for week: £" + Math.Round(totalCommission, 2)
    OUTPUT "Total sales for week: " + totalPropertiesSold
    OUTPUT employeeName[employeeNumber[1]] + " has sold the most properties at " +
propertiesSold[1] + "properties. A bonus of £" + Math.Round(bonus, 2) + " will be applied."

END
```

The learner has produced a structure which shows appropriate and consistent use of hierarchy and indentation, providing clarity and mostly readable pseudocode. The pseudocode will provide a working solution with some minor errors. Appropriate naming conventions have been used and precise use of logical operations.

Mark in **band 3 (8 marks)**.

Commission Calculator Pseudocode

```
START
INPUT "Input week number"
IF week number is between 1 and 52
        OUTPUT "week number selected"
ELSE
        OUTPUT "Error. Please input a valid week number"

INPUT "Input number of employees working this week"
IF number of employees is between 2 and 5
        OUTPUT "Number of employees selected"
ELSE
        OUTPUT "Error. Please enter a valid number of employees"

INPUT employee names
STORE user input in the answer variable

INPUT employee ID numbers
STORE user input in the answer variable

INPUT number of properties sold per employee
STORE user input in the answer variable

OUTPUT employee list

CALCULATE total number of properties sold
OUTPUT total number of properties sold in the week

CALCULATE commission payed per employee
```



```
Commission Calculator Pseudocode

OUTPUT commission payed per employee


CALCULATE total commission payed
OUTPUT total commission payed


SORT employees in descending order by number of properties sold
ADD 15% bonus to highest ranking employee
OUTPUT commission total including bonus


END
```

The learner has produced a structure which shows some appropriate and consistent use of hierarchy and indentation, providing some clarity and readable pseudocode. However, the pseudocode will not provide a working solution as it is inefficient and does not show any calculations.

Mark in **band 1 (3 marks)**.

Activity 3

Test Plan

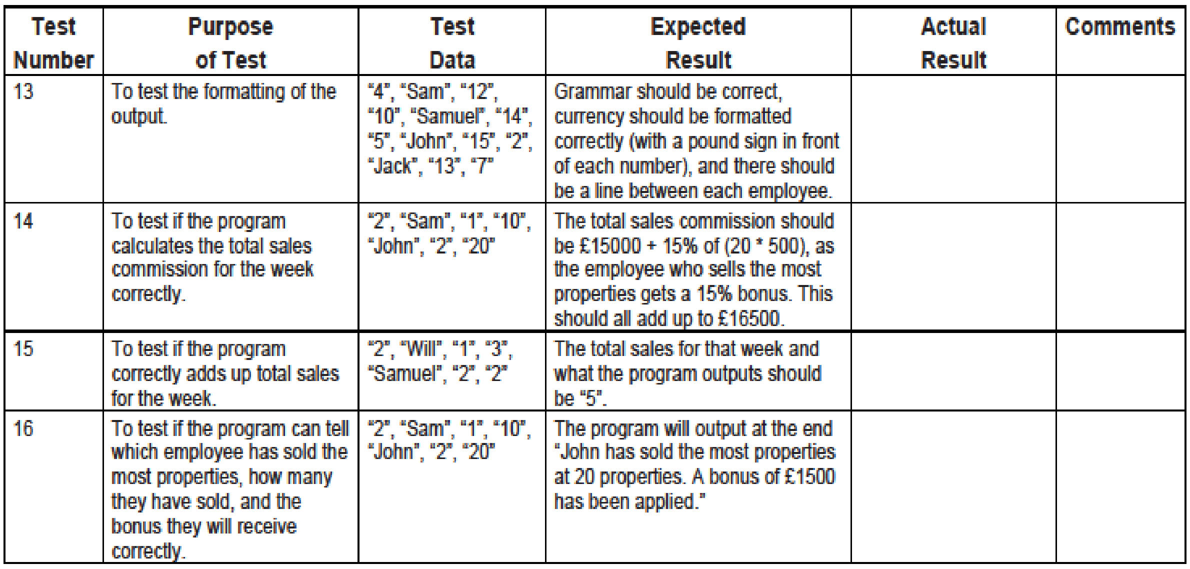
Document for Activities 3 and 4

Test Plan (add additional rows as required)

Program language the product is to be produced in (tick box for language used):

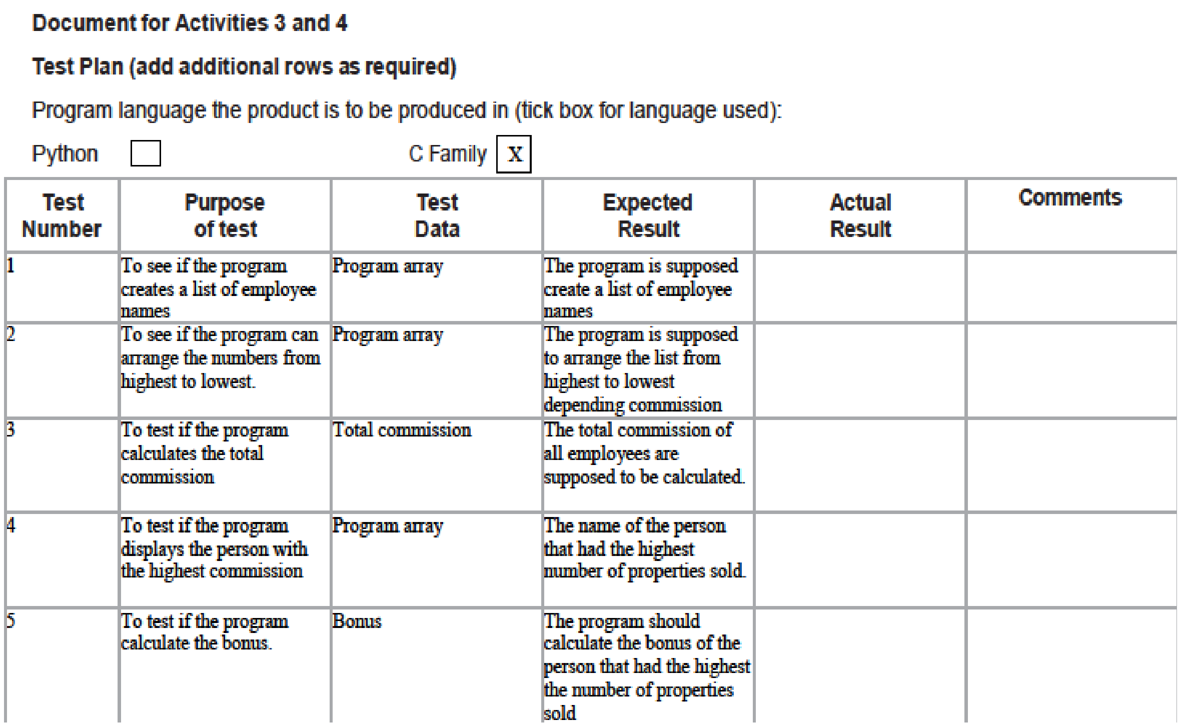
Python ☐ C Family ■

Test Number	Purpose of Test	Test Data	Expected Result	Actual Result	Comments
1	Test if the program accepts valid input for the amount of employees input.	“4”	Program accepts the input and moves on to the next field.		
2	Test if the program does not accept abnormal data in the amount of employees input.	“E$&”d#[“	Console outputs “Invalid amount of employees. Please try again.” And lets user input again, showing it handled the abnormal data accordingly.		
3	Test if the program handles the upper bounds data accordingly in the amount of employees input.	“5”	The program should accept the input as it is the largest acceptable input.		
4	Test if the program does not accept extreme data, specifically data that is above the upper bounds, in the amount of employees input.	“6”	The program should output “Invalid amount of employees. Please try again.” And allow user to input again		
5	Test if the program accepts valid input for the number of properties sold input.	“10”	The program should accept this, and later should output the sales commission for the employee as £5,000 if the employee did not sell the most properties.		
6	Test if the program does not accept abnormal data for the number of properties sold input.	“78KP*&”	The program will not accept this and will allow the user to input again, first prompting them that their input was invalid.		
7	Test if the program handles the lower bounds of the number of properties sold input.	“0”	The program will accept this as it is a valid input; an employee could have sold 0 houses.		
8	Test if the program correctly handles extreme data – specifically negative numbers.	“-16”	The program will not accept this input and will output “Invalid number of properties sold. Please try again.” And then allow the user to try again.		
9	Test if the program correctly outputs user ID for one employee.	“4884329”	The program outputs “Employee ID: 4884329”		
10	Test if the program correctly outputs employee name for one employee.	“Sam Johnson”	The program outputs “Employee name: Sam Johnson”		
11	Test if the program correctly outputs the number of properties sold for one employee.	“12”	The program outputs “Number of properties sold for employee: 12”		
12	Test if the program correctly outputs the sales commission for one employee.	“12”, and for the other employee, “5”	The program should output “Sales commission for employee: £6900” as the employee is not only given £6000 for selling 12 properties but is given an extra 15% bonus for selling the most properties.		



Test Number	Purpose of Test	Test Data	Expected Result	Actual Result	Comments
13	To test the formatting of the output.	"4", "Sam", "12", "10", "Samuel", "14", "5", "John", "15", "2", "Jack", "13", "7"	Grammar should be correct, currency should be formatted correctly (with a pound sign in front of each number), and there should be a line between each employee.		
14	To test if the program calculates the total sales commission for the week correctly.	"2", "Sam", "1", "10", "John", "2", "20"	The total sales commission should be £15000 + 15% of (20 * 500), as the employee who sells the most properties gets a 15% bonus. This should all add up to £16500.		
15	To test if the program correctly adds up total sales for the week.	"2", "Will", "1", "3", "Samuel", "2", "2"	The total sales for that week and what the program outputs should be "5".		
16	To test if the program can tell which employee has sold the most properties, how many they have sold, and the bonus they will receive correctly.	"2", "Sam", "1", "10", "John", "2", "20"	The program will output at the end "John has sold the most properties at 20 properties. A bonus of £1500 has been applied."		

The learner has produced a thorough test plan to confirm a working solution which includes a range of data. Expected results are specific and accurate based on identified test data. Mark in **band 3 (6 marks)**.



Document for Activities 3 and 4

Test Plan (add additional rows as required)

Program language the product is to be produced in (tick box for language used):

Python ☐ C Family [X]

Test Number	Purpose of test	Test Data	Expected Result	Actual Result	Comments
1	To see if the program creates a list of employee names	Program array	The program is supposed create a list of employee names		
2	To see if the program can arrange the numbers from highest to lowest.	Program array	The program is supposed to arrange the list from highest to lowest depending commission		
3	To test if the program calculates the total commission	Total commission	The total commission of all employees are supposed to be calculated.		
4	To test if the program displays the person with the highest commission	Program array	The name of the person that had the highest number of properties sold.		
5	To test if the program calculate the bonus.	Bonus	The program should calculate the bonus of the person that had the highest the number of properties sold		

The learner has produced a test plan but has no actual test data. Expected results are generic, no marks can be awarded for this test plan.

Mark in **band 0 (0 marks)**.

Activity 4 – Program

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Property_agency_commission_calculator
{
    class Program
    {
        static void Main(string[] args)
        {
            //Variables are declared, some are given a value as they are used in statements before being given a value by the user.
            int employeesWorking = 0;
            double totalCommission = 0;
            int totalPropertiesSold = 0;
            double bonus;

            Console.WriteLine("Input the amount of employees working this week. Number must be no smaller than 2 and no larger than 5.");
            //The while statement below makes it so the program will not continue while the value of the variable "employeesWorking" is smaller than 2 or bigger than 5.
            //This makes it so the user has to enter a value no smaller than 2 and no bigger than 5 in order for the program to continue.
            while (employeesWorking < 2 || employeesWorking > 5)
            {
                //This try-catch statement makes it so if the user inputs invalid data (like "3.5" or "$&^%6") the program handles the error that data gives.
        //This is because it cannot accept characters that are not numbers or
values that are decimals as the data type of the variable is integer.
        try
        {
            //The line of code below converts the string input of the user into an
integer so it can be compared to other integers and added up and multiplied by
later on in the program.
            //This line of code can create the error talked about above, however as
it is in a try catch statement that error is handled.
            employeesWorking = Convert.ToInt32(Console.ReadLine());
            if (employeesWorking < 2 || employeesWorking > 5)
            {
                Console.WriteLine("Invalid amount of employees. Please try again.");
            }
        }
        catch
        {
            Console.WriteLine("Invalid amount of employees. Please try again.");
        }
    }

    //The lines of code below create all the arrays used to store employee
details.
    //I used multiple 1D arrays as they are easier to sort. I felt they were more
suitable for this program.
    string[] employeeName = new string[employeesWorking];
    string[] employeeID = new string[employeesWorking];
    int[] employeeNumber = new int[employeesWorking];
    int[] propertiesSold = new int[employeesWorking];
    double[] salesCommission = new double[employeesWorking];
    //This for loop loops for the number of times that there are employees
working, allowing the user to input details for each employee.
```

```
for (int i = 0; i < employeesWorking; i++)
{
    //"i" is used a lot in this for loop. It is a integer variable that goes up by 1 each iteration.
    //It is often placed inside the square brackets of the arrays, as each iteration is accesses a different place in each array.
    employeeNumber[i] = i;

    Console.WriteLine("");
    Console.WriteLine("Employee number " + (employeeNumber[i] + 1));
    Console.WriteLine("Input employee name: ");
    employeeName[i] = Console.ReadLine();

    Console.WriteLine("Input employee ID.");
    employeeID[i] = Console.ReadLine();

    Console.WriteLine("Input the number of properties sold for this employee.");
    //propertiesSold[i] is set to -1 as it would by default be set to 0, therefore the while loop would be skipped.
    propertiesSold[i] = -1;
    //This while loop loops until the user inputs a value greater or equal to 0.
    while (propertiesSold[i] < 0)
    {
        //Like the previous try catch statement, this is in place to handle the errors caused by invalid data being entered.
        try
        {
            propertiesSold[i] = Convert.ToInt32(Console.ReadLine());
            if (propertiesSold[i] < 0)
            {
```

```
                            Console.WriteLine("Invalid number of properties sold. Please try
again.");
                        }
                    }
                    catch
                    {
                        Console.WriteLine("Invalid number of properties sold. Please try
again.");
                    }
                }

                //The lines of code below:
                //Calculate the sales commission for each employee,
                //Add each employee's commission to the total commission,
                //Add each employee's number of properties sold to the total number of
properties sold.
                salesCommission[i] = propertiesSold[i] * 500;
                totalCommission += salesCommission[i];
                totalPropertiesSold += propertiesSold[i];
            }

            //These lines of code sort the number of properties sold from low to high,
as well as sorting the employee numbers array by that propertiesSold array.
            //This means the employeeNumber array now has the order (from low to
high) of each employee by the amount of properties they sold.
            //Both arrays are then reversed to order them both from high to low.
            Array.Sort(propertiesSold, employeeNumber);
            Array.Reverse(propertiesSold);
            Array.Reverse(employeeNumber);

            //The lines of code below calculate the bonus that the employee that sold
the most properties will receive.
```

```
    //It also adds the employee's bonus to the total commission and to their own commission.
    bonus = salesCommission[employeeNumber[0]] * 0.15;
    totalCommission += bonus;
    salesCommission[employeeNumber[0]] += bonus;


    //In the lines below where currency is being displayed, I used ".ToString("c2")" to format the currency correctly. .ToString converts the variable to a string.
    //"c2" converts the variable into a currency, adding a pound sign before and displaying two decimal points as required.


    //This loop's purpose is to output the details of each employee, including their ID, name, the number of properties they sold, and the sales commission they will receive.
    for (int i = 0; i < employeesWorking; i++)
    {
        //For every iteration, the line below assigns the employee number for the employees from the most properties sold to the lowest.
        //This is extremely useful as it means I can reference each employee's details in order of the amount of properties sold descending without having to sort the other arrays.
        int j = employeeNumber[i];
        Console.WriteLine("");
        Console.WriteLine("Employee ID: " + employeeID[j]);
        Console.WriteLine("Employee name: " + employeeName[j]);
        Console.WriteLine("Number of properties sold for employee: " + propertiesSold[i]);
        //The line of code below
        Console.WriteLine("Sales commission for employee: " + salesCommission[j].ToString("c2"), 2);
    }
```

```
            //The lines of code below display the total commission for the week, and the total sales for the week.

            //Also, the bottom line displays the name, amount of properties they sold, and they bonus they received, of the employee that sold the most properties.

            Console.WriteLine("Total sales commission for week: " + totalCommission.ToString("c2"));

            Console.WriteLine("Total sales for week: " + totalPropertiesSold);

            //Below you can see employeeNumber[0] being used inside the square brackets of the employeeName array and propertiesSold[0] being used.

            //"employeeNumber[0]" references the position in the employeeName array of the employee that has sold the most properties.

            //"propertiesSold[0]" references the amount of properties the user that has sold the most amount of properties has sold.

            Console.WriteLine(employeeName[employeeNumber[0]] + " has sold the most properties at " + propertiesSold[0] + " properties. A bonus of " + bonus.ToString("c2") + " has been applied.");


            Console.WriteLine("");

            //The line of code below prompts the user that they can press enter to exit the program.

            Console.WriteLine("Press enter to exit this program.");

            //If the user presses enter, the line of code below makes it so that the program will end, as it tries to go onto the next line of code, realises there is no more code, and therefore terminates the program.

            Console.ReadLine();

        }

    }

}
```

The learner has produced a program that fully meets all the requirements. Accurate syntax and indentation have been used throughout the code and commenting is consistently clear and informative. Program outputs are accurate and informative, validation and other checks have been used which are all accurate, resulting in a robust program being created.

Mark in **band 4 (24 marks)**.

```
print("Welcome to commission calculator version 1.0 ")
print("")

#This is the commission earned by an employee for each property sold.
propComm = int(500)

#This is the field in which employee information will be stored
employees = []

#This is the field in which the total properties sold will be stored
totalProp = int()

# variables:
# empNum - number of employees working this week
# empID - individual employee ID number
# empName - individual employee name
# propSold - number of properties sold by employee
# totalComm - total commission paid out
# empComm - commission earned per employee
# empRank - employee's rank by properties sold

def week_num():
    weekNum = int(input("please enter week number: "))
    print("")
    if (weekNum >= 1) and (weekNum <=52):
        print ("week selected")
        print ("")
    else:
        print ("Please select a valid week number")
        print("")
        week_num()
week_num()

def main():
    totalProp = 0
    empNum = int(input("Please enter number of employees working this week: "))
    print("")
    if (empNum >= 2):
        print("Number of employees selected")
        print("")
    elif (empNum <2) or (empNum >5):
        print ("Please enter a valid number of employees")
        print("")
#~~~ add loop to above function so that an invalid week number loops back to
beginning ~~~
    for i in range(0,empNum):
        empID = str(input("Employee ID: "))
        empName = str(input("Employee Name: "))
        propSold = int(input("Properties Sold: "))
        commPaid = propComm * propSold

        employees.append([empID, empName, propSold, commPaid])
    empIndex = int()
    for row in employees:
        totalProp = totalProp + int(employees[empIndex][2])
        empIndex = empIndex + 1

main()
```

The learner has produced a program that meets some of the requirements. The program accepts an input for week number, number of employees working, name, ID and properties sold. Once these inputs have been entered the program ends. Some validation has been used but not effectively. Outputs are accurate and mostly informative. Some accurate syntax and indentation used along with some logical structure. No error handling has been used. Commenting of the code has not been done so this can only get a mark in mark band 1.

Mark in **band 1 (6 marks).**

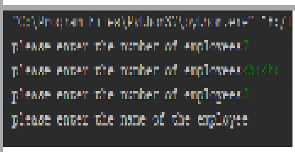
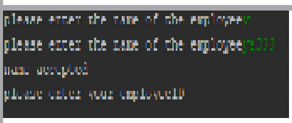
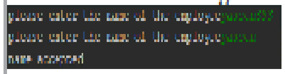
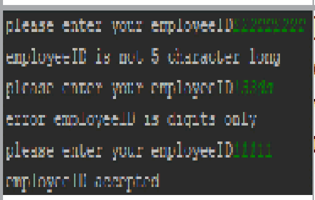
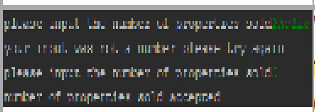
Activity 4 – Testing

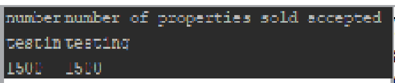
Document for Activities 3 and 4

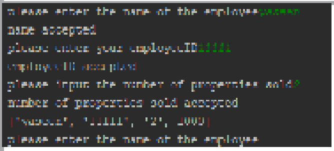
Test Plan (add additional rows as required)

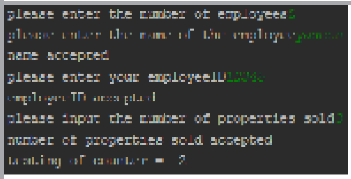
Program language the product is to be produced in (tick box for language used):

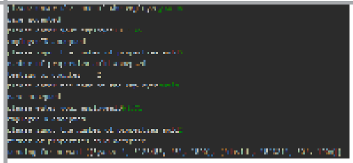
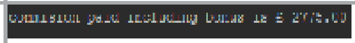
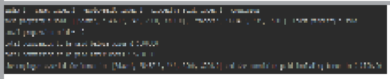
Python yes

Test Number	Purpose of test	Test Data	Expected Result	Actual Result	Comments
1	To check if the input for the number of employees is a digit and not alphanumeric and is within 2 to 5.	3 4hj4hj 7	The code should recognise the alpha characters and the input is not within range and ask the user to input again.	 "C:\Program Files\Python37\python.exe" "E:/ please enter the number of employees7 please enter the number of employees4hj4hj please enter the number of employees3 please enter the name of the employee	Here the result was as expected the number of employees could not be input unless it was a digit or within a range of 2 and 5
2	To check if the name input does not contain numbers and is equal to or above 2 characters long	Y Yas333 Yaseen	The code should recognise both the numbers in the second test data and the length of the first and ask the user to input the name again.	 please enter the name of the employeeY please enter the name of the employeeYas333 name accepted please enter your employeeID First test failed allowed digits  please enter the name of the employee[illegible] please enter the name of the employeeyaseen name accepted After fixing only alpha characters are allowed	The if condition checking if name was a digit would allow both numbers and letters so I changed it to only look for alpha characters
3	To check if the employee ID is made of digits and if the length of it is 5	11111 133ds 22222222222222	The code should recognise both invalid inputs and ask to user to enter the ID number once again	 please enter your employeeID2222222222 employeeID is not 5 character long please enter your employeeID133ds error employeeID is digits only please enter your employeeID11111 employeeID accepted	Here the test worked as expected no invalid input was permitted only 5-digit numbers.
4	To check if the number of properties sold is a digit.	2 2hello	The code will detect a invalid data type and ask the user to input the data once again.	 please input the number of properties sold2hello your input was not a number please try again please input the number of properties sold2 number of properties sold accepted	The validation confirmed that alphanumerical characters where input and asked the user to input it again.

5	The purpose of this test is to see if the employee commission is calculated correctly	3 properties
		
		Properties that are sold will
			The testing print statement will show the accurate calculation which would be 1500	
				`number number of properties sold accepted`
				`testin testing`
				`1500 1500`
					The calculation wa accurate and the correct commissio was calculated

6	To check if the employee data is being correctly appended to their own list	The employee name, ID, commission and properties sold will be appended	The testing print statement will show a list containing all the of the employee's information	
				`please enter the name of the employeeyaseen`
				`name accepted`
				`please enter your employeeID11111`
				`employee ID accepted`
				`please input the number of properties sold2`
				`number of properties sold accepted`
				`['yaseen', '11111', '2', 1000]`
				`please enter the name of the employee`
					Here the list for employee1 has bee output showing tha the data is being appended correctly

7	Check if the global count variable is incrementing by 1 each time an employee's information is input	calculation no test data	A testing print statement will show the count will be incremented from its original value of 1 to 2.	
				`please enter the number of employees2`
				`please enter the name of the employeeyaseen`
				`name accepted`
				`please enter your employeeID12345`
				`employee ID accepted`
				`please input the number of properties sold3`
				`number of properties sold accepted`
				`testing of counter = 2`
					After the first employee data is input the counter is incremented by 1

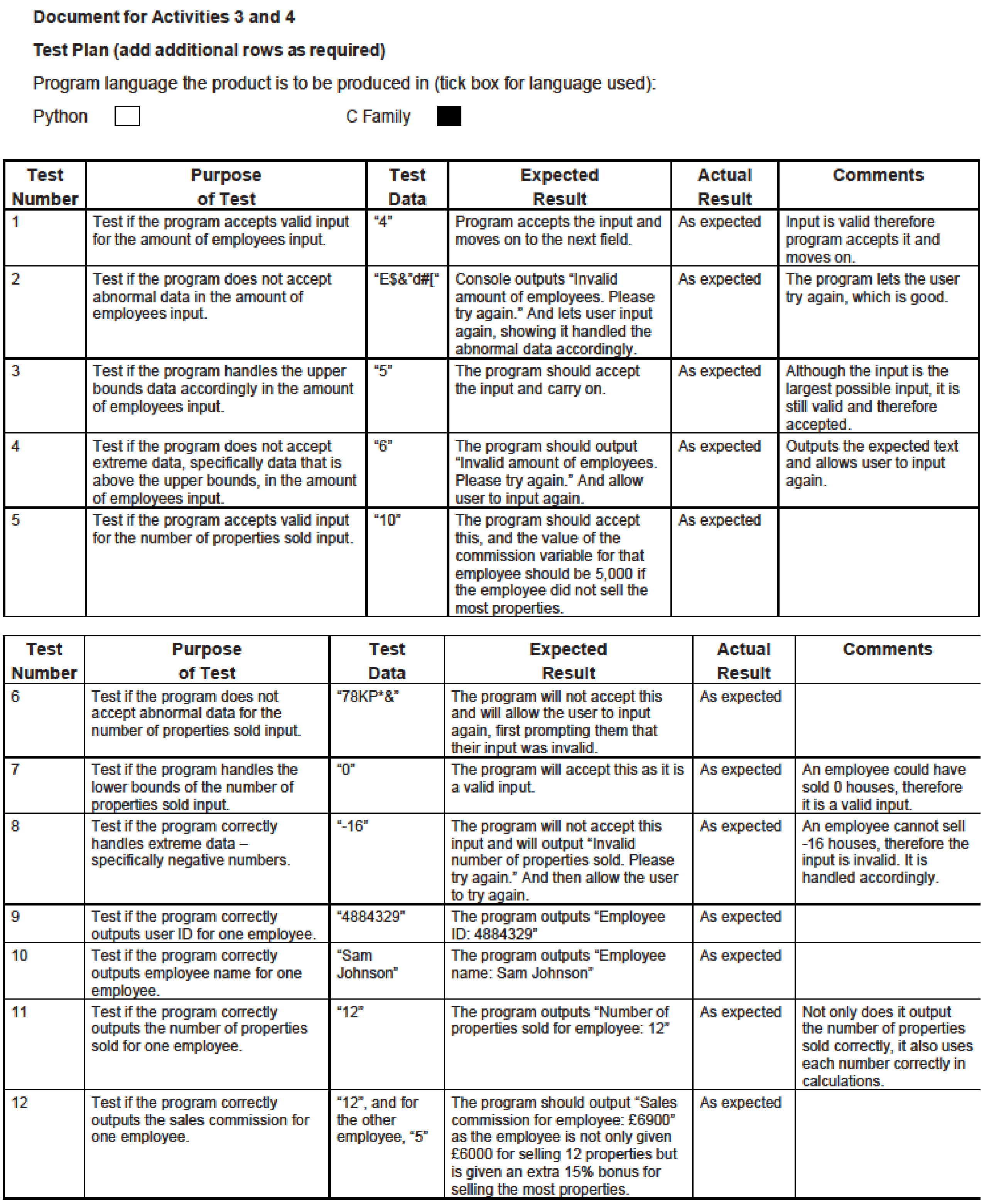
8	To check if the list is being sorted correctly from the highest amount of properties sold to the lowest.	There is no test data it is a test to ensure that a function to sort data is accurate.	The 2D array of employee lists is output with the 3rd index (properties) being shown from high to low	 [illegible]	After both employ are input the sorted list contains both their information a yaseen is first as he sold the most property's
9	To check if the bonus commission is correctly calculated	The employee will have sold 3 properties	The commission will be a 15% bonus on top of his 1500 commission which is 2775	 `commission paid including bonus is £ 2775.00`	Here the output bonus is accurate the test was a succ
10	Check to see if all information is being output to user.	Previous inputs and calculations	The data is correctly output to the user.	 [illegible]	All the informatio output and easy to read.

The learner has produced some evidence of an iterative development process that identifies and resolves some basic errors. Comments show understanding of the basic errors and how they were fixed.

Mark in **band 2 (3 marks).**

Document for Activities 3 and 4

Test Plan (add additional rows as required)



Document for Activities 3 and 4

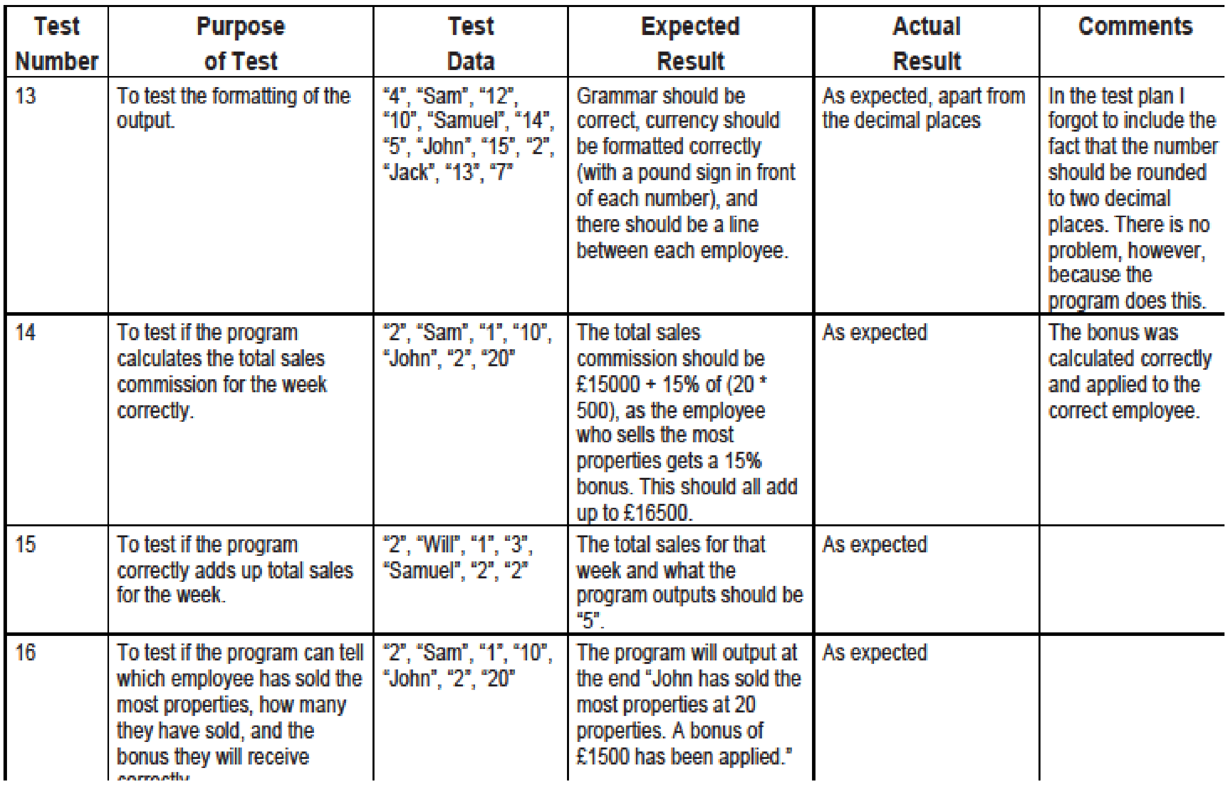
Test Plan (add additional rows as required)

Program language the product is to be produced in (tick box for language used):

Python ☐ C Family ■

Test Number	Purpose of Test	Test Data	Expected Result	Actual Result	Comments
1	Test if the program accepts valid input for the amount of employees input.	"4"	Program accepts the input and moves on to the next field.	As expected	Input is valid therefore program accepts it and moves on.
2	Test if the program does not accept abnormal data in the amount of employees input.	"E$&"d#["	Console outputs "Invalid amount of employees. Please try again." And lets user input again, showing it handled the abnormal data accordingly.	As expected	The program lets the user try again, which is good.
3	Test if the program handles the upper bounds data accordingly in the amount of employees input.	"5"	The program should accept the input and carry on.	As expected	Although the input is the largest possible input, it is still valid and therefore accepted.
4	Test if the program does not accept extreme data, specifically data that is above the upper bounds, in the amount of employees input.	"6"	The program should output "Invalid amount of employees. Please try again." And allow user to input again.	As expected	Outputs the expected text and allows user to input again.
5	Test if the program accepts valid input for the number of properties sold input.	"10"	The program should accept this, and the value of the commission variable for that employee should be 5,000 if the employee did not sell the most properties.	As expected	

Test Number	Purpose of Test	Test Data	Expected Result	Actual Result	Comments
6	Test if the program does not accept abnormal data for the number of properties sold input.	"78KP*&"	The program will not accept this and will allow the user to input again, first prompting them that their input was invalid.	As expected	
7	Test if the program handles the lower bounds of the number of properties sold input.	"0"	The program will accept this as it is a valid input.	As expected	An employee could have sold 0 houses, therefore it is a valid input.
8	Test if the program correctly handles extreme data – specifically negative numbers.	"-16"	The program will not accept this input and will output "Invalid number of properties sold. Please try again." And then allow the user to try again.	As expected	An employee cannot sell -16 houses, therefore the input is invalid. It is handled accordingly.
9	Test if the program correctly outputs user ID for one employee.	"4884329"	The program outputs "Employee ID: 4884329"	As expected	
10	Test if the program correctly outputs employee name for one employee.	"Sam Johnson"	The program outputs "Employee name: Sam Johnson"	As expected	
11	Test if the program correctly outputs the number of properties sold for one employee.	"12"	The program outputs "Number of properties sold for employee: 12"	As expected	Not only does it output the number of properties sold correctly, it also uses each number correctly in calculations.
12	Test if the program correctly outputs the sales commission for one employee.	"12", and for the other employee, "5"	The program should output "Sales commission for employee: £6900" as the employee is not only given £6000 for selling 12 properties but is given an extra 15% bonus for selling the most properties.	As expected	



Test Number	Purpose of Test	Test Data	Expected Result	Actual Result	Comments
13	To test the formatting of the output.	"4", "Sam", "12", "10", "Samuel", "14", "5", "John", "15", "2", "Jack", "13", "7"	Grammar should be correct, currency should be formatted correctly (with a pound sign in front of each number), and there should be a line between each employee.	As expected, apart from the decimal places	In the test plan I forgot to include the fact that the number should be rounded to two decimal places. There is no problem, however, because the program does this.
14	To test if the program calculates the total sales commission for the week correctly.	"2", "Sam", "1", "10", "John", "2", "20"	The total sales commission should be £15000 + 15% of (20 * 500), as the employee who sells the most properties gets a 15% bonus. This should all add up to £16500.	As expected	The bonus was calculated correctly and applied to the correct employee.
15	To test if the program correctly adds up total sales for the week.	"2", "Will", "1", "3", "Samuel", "2", "2"	The total sales for that week and what the program outputs should be "5".	As expected	
16	To test if the program can tell which employee has sold the most properties, how many they have sold, and the bonus they will receive	"2", "Sam", "1", "10", "John", "2", "20"	The program will output at the end "John has sold the most properties at 20 properties. A bonus of £1500 has been applied."	As expected	

Testing shows evidence of a limited or linear development process, with minimal identification and resolution of errors.

To get into mark band 2 there must be evidence of errors and how they have been solved.

Mark in **band 1 (2 marks)**.

Activity 5

Evaluation

I was employed by a local property agency to write a program that will help its Sales Manager by calculating the amount of commission paid to employees and which employee will receive a bonus. I feel my solution meets the requirements of the scenario.

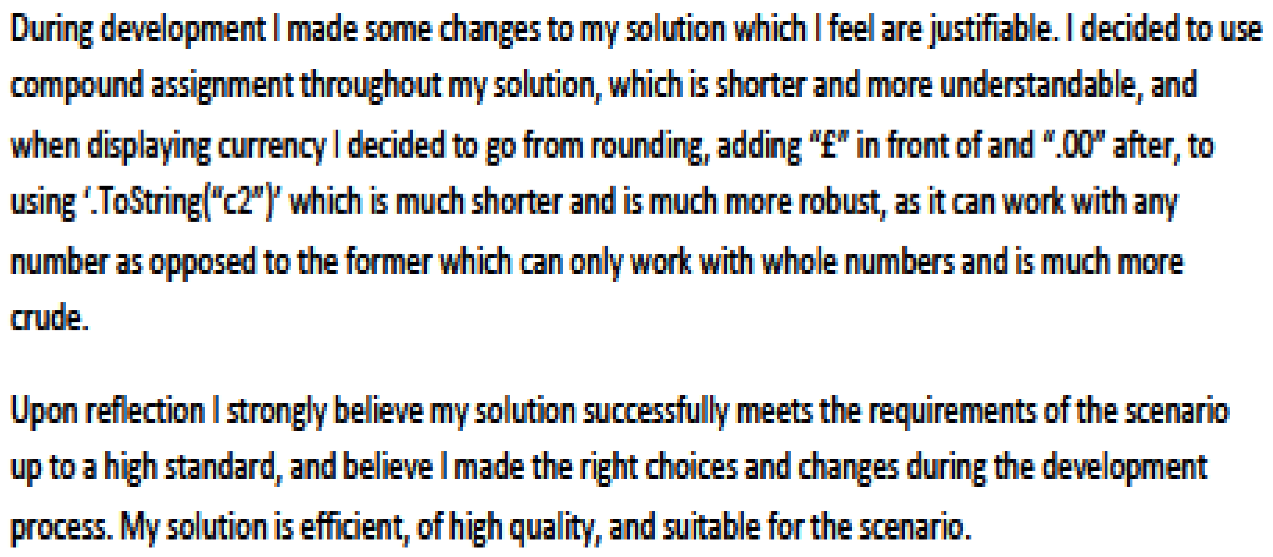
It allows the user to input the name, ID number and number of properties sold for each employee working; it calculates the sales commission paid to each employee, the total commission the company must pay, and the total number of properties sold by the company; it ranks the employees in order of the number of properties they sold from the highest number to the lowest number, and applies a 15% bonus to the commission of the employee with the most sales, taking this into account when calculating the total amount of commission the company will need to pay.

In terms of quality and performance, I believe my solution is efficient and robust, handling all types of data the user can enter accordingly. I have ensured my solution is user friendly, incorporating spacing in both the code and the program itself so as to help users clearly identify each user, or to make it clear to them when to enter data, when they have entered invalid data, and what to do next to name some examples.

When storing the details of each user I felt I had two viable options; 2D arrays and 1D arrays. After careful consideration I decided to use 1D arrays, as I strongly believe them to be better suited for the scenario; 1D arrays are much easier to sort, allowing the developer to use method groups such as "Array.Sort" and "Array.Reverse" to efficiently sort and reverse arrays in single lines of code. Had I used 2D arrays I would had to have written much more code. This, the use of 2D arrays, would have been appropriate in the circumstance that I was developing a solution for a larger company with tens or hundreds of employees working at once instead of 2 to 5. I believe this is the case because a 2D array would be more robust when handling a lot more data as in the circumstance, however, for this smaller company, I feel 1D arrays are more appropriate as they are simpler and can easily handle small amounts of data.

My program recorded the number of properties sold as whole numbers, as I use the integer data type, and the output is all formatted correctly, including correct grammar. The employees and their details are displayed in order of number of properties sold, from most to least. The sales commission for each employee and the total for the week are both correctly displayed as currency to two decimal places, the total number of properties sold in the week is also displayed. The name and number of properties sold for the employee who has sold the most, including the bonus amount to be applied to that employee, is all displayed – the bonus being displayed as currency to two decimal places.

I feel my code is formatted correctly – I utilised indentations, camel case, compound assignment and comments to make my code easy to read and understand. Expanding more on readability, I followed coding etiquette by breaking up my code with blank lines, sectioning areas of code to make it more readable.



During development I made some changes to my solution which I feel are justifiable. I decided to use compound assignment throughout my solution, which is shorter and more understandable, and when displaying currency I decided to go from rounding, adding "£" in front of and ".00" after, to using '.ToString("c2")' which is much shorter and is much more robust, as it can work with any number as opposed to the former which can only work with whole numbers and is much more crude.

Upon reflection I strongly believe my solution successfully meets the requirements of the scenario up to a high standard, and believe I made the right choices and changes during the development process. My solution is efficient, of high quality, and suitable for the scenario.

The learner has demonstrated a mostly accurate and detailed understanding of technical concepts. Valid and mostly supported justification of coding conventions used, and the learner has made logical links between aspects of the solution and the requirements of the scenario.

Valid and mostly supported judgements of the quality and performance of the program. Accurate technical vocabulary used to support arguments.

Mark in **band 3** (**9 marks**).

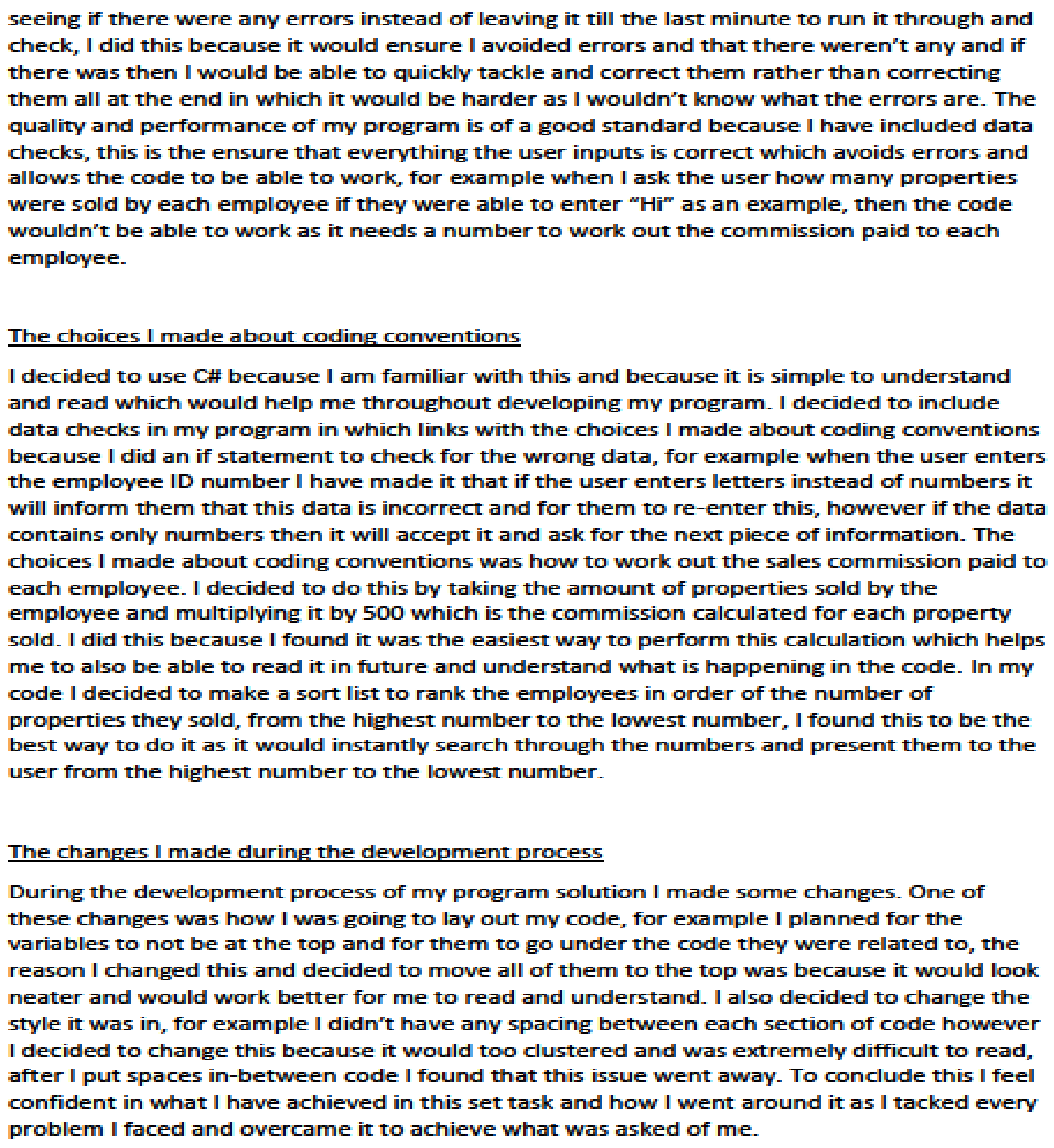
Evaluation

How well my solution meets the requirements of the scenario

My solution meets the requirements of the scenario very well because I have created a fully functional program which can be used by the local property agency to help its sales manager. This is because the user can enter the employee name, employee ID number and number of properties sold for each employee working to then get all of the data and information they require, this is because I have done all of the code so all they need to do is enter this minimal information to get the details the agency is after. My solution meets the requirements of the scenario because this is details such as calculating the sales commission paid to each employee, calculating the total commission the company must pay, calculating the total number of properties sold by the company, ranking the employees in order of the number of properties they sold, from the highest number to the lowest number, and applying a 15% bonus to the commission of the employee with the most sales. My solution also meets the requirements of the scenario because I successfully created a good flowchart, this helped me with my code because it allowed me to understand what was being asked of me as I got to write it down and visually see how it was going to work out before I started the code. I ensured my flowchart was clear so I could get a good understanding from it by avoiding making it look clustered. I also made for to keep my flowchart simplistic but without restricting all of the detail from it so it was easy to read and wasn't too overwhelming. My solution also meets the requirements of the scenario because I successfully created a good piece of pseudocode, this helped me with my code for the same reason as my flowchart as it further allowed me to understand what was being asked of me as it was a way of double checking all the content was there and that I knew what I was going to do in my code. My flowchart and pseudocode helped my solution meet the brief as if I didn't do this and went straight onto the code there could've been something I missed however this allowed me to ensure I read what was being asked thoroughly at least twice avoiding mistakes and errors.

The quality and performance of my program

My program is of a high quality and performance because in my code I included indents in my code, this was because I wanted to ensure it was easy to read to prevent any confusion. Not only this, but it also allowed my code to look efficient and neat as it wasn't all in a single block together. The quality and performance of my program is of a high standard because I included comments throughout my code, this ensured that my program was easy to read as every step I took was explained thoroughly, this also ensured that I was able to understand the program myself so I knew what I had and hadn't done so I knew what was next to avoid redoing something and creating errors throughout the code. I ensured to have my variables placed at the top of the code to have it all neatly laced together, I also did this because it would be easier to locate all of my variables if I needed to check something instead of searching the whole code to look for one specific piece of code. I ensured the quality and performance of my program was of a high standard by constantly running my program and



seeing if there were any errors instead of leaving it till the last minute to run it through and check, I did this because it would ensure I avoided errors and that there weren't any and if there was then I would be able to quickly tackle and correct them rather than correcting them all at the end in which it would be harder as I wouldn't know what the errors are. The quality and performance of my program is of a good standard because I have included data checks, this is the ensure that everything the user inputs is correct which avoids errors and allows the code to be able to work, for example when I ask the user how many properties were sold by each employee if they were able to enter "Hi" as an example, then the code wouldn't be able to work as it needs a number to work out the commission paid to each employee.

<u>The choices I made about coding conventions</u>

I decided to use C# because I am familiar with this and because it is simple to understand and read which would help me throughout developing my program. I decided to include data checks in my program in which links with the choices I made about coding conventions because I did an if statement to check for the wrong data, for example when the user enters the employee ID number I have made it that if the user enters letters instead of numbers it will inform them that this data is incorrect and for them to re-enter this, however if the data contains only numbers then it will accept it and ask for the next piece of information. The choices I made about coding conventions was how to work out the sales commission paid to each employee. I decided to do this by taking the amount of properties sold by the employee and multiplying it by 500 which is the commission calculated for each property sold. I did this because I found it was the easiest way to perform this calculation which helps me to also be able to read it in future and understand what is happening in the code. In my code I decided to make a sort list to rank the employees in order of the number of properties they sold, from the highest number to the lowest number, I found this to be the best way to do it as it would instantly search through the numbers and present them to the user from the highest number to the lowest number.

<u>The changes I made during the development process</u>

During the development process of my program solution I made some changes. One of these changes was how I was going to lay out my code, for example I planned for the variables to not be at the top and for them to go under the code they were related to, the reason I changed this and decided to move all of them to the top was because it would look neater and would work better for me to read and understand. I also decided to change the style it was in, for example I didn't have any spacing between each section of code however I decided to change this because it would too clustered and was extremely difficult to read, after I put spaces in-between code I found that this issue went away. To conclude this I feel confident in what I have achieved in this set task and how I went around it as I tacked every problem I faced and overcame it to achieve what was asked of me.

The learner has demonstrated superficial understanding of relevant technical concepts. There is unsupported justification of changes made during the development process and limited justification of coding conventions supported. Limited judgements about the quality and performance of the program keeps this evaluation in **mark band 1 (3 marks)**.

Summary

Based on performance in this examination series, learners are offered the following advice:

- Apply their knowledge to as many different scenarios as possible. The exam paper will always contain 5 activities which always be the same just the scenario would be different and therefore this will prepare learners to be able to provide answers to the given context under exam conditions.
- Use standard naming conventions throughout the design process and clearly demonstrate this in the flowchart and pseudocode.
- Pseudocode needs to be a detailed yet readable description of what a computer program must do, expressed in a natural language rather than in a programming language if top marks are to be achieved.
- Develop a better understanding of the testing process. Test plan must include normal, abnormal and extreme data. Testing must address errors encountered and how these were overcome. The testing must be iterative, document tests when code is being developed as this will give a true reflection of the development.
- Ensure the Program uses accurate validation and error checking procedures throughout, resulting in a robust program that minimises errors and handles unexpected events. This will enhance the completed solution and allow the higher mark bands to be accessed. Programs must address most requirements to gain higher marks.
- The evaluation needs to include a fully supported justification of changes made during the development process, as well as a fully supported justification of coding conventions selected if higher mark bands are to be accessed.

Llywodraeth Cynulliad Cymru
Welsh Assembly Government

Rewarding Learning