T Level Technical Qualification in Digital Production, Design and Development

Mark Scheme (Results)

Autumn 2022

Employer Set Project

General Marking Guidance

- All learners must receive the same treatment. Examiners must mark the first learner in exactly the same way as they mark the last.
- Mark schemes should be applied positively. Learners must be rewarded for what they have shown they can do rather than penalised for omissions.
- Examiners should mark according to the mark scheme not according to their perception of where the grade boundaries may lie.
- All marks on the mark scheme should be used appropriately.
- All the marks on the mark scheme are designed to be awarded. Examiners should always award full marks if deserved. Examiners should also be prepared to award zero marks if the learner's response is not rewardable according to the mark scheme.
- Where judgement is required, a mark scheme will provide the principles by which marks will be awarded.
- When examiners are in doubt regarding the application of the mark scheme to a learner's response, a senior examiner should be consulted.
- Crossed out work should be marked **unless** the learner has replaced it with an alternative response.
- Accept incorrect/phonetic spelling (as long as the term is recognisable) unless instructed otherwise.

Levels-Based Mark Scheme Guidance

Levels-based mark schemes (LBMS) have been designed to assess students' work holistically. They consist of two parts:

1) Indicative content
Indicative content reflects content-related points that a student might make but is not an exhaustive list. Nor is it a model answer. Students may make some or none of the points included in the indicative content as its purpose is as a guide for the relevance and expectation of the responses. Students must be credited for any appropriate response.

2) Levels-based descriptors
Each level is made up of a number of traits which when combined together articulate the quality of response that a student needs to demonstrate. The traits progress across the levels to demonstrate the different expectations of each level. When using a levels-based mark scheme, the 'best fit' approach should be used.

Applying the levels-based descriptors

Examiners should take a 'best fit' approach to determining the mark.

- Examiners should first make a holistic judgement on which level most closely matches the student's response. Students will be placed in the level that best describes their answer. Answers can display characteristics from more than one level, and where this happens markers must use any additional guidance (e.g., weighting of traits) and their professional judgement to decide which level is most appropriate.

- The mark awarded within the level will be decided based on the quality of the answer and will be modified according to how securely all traits are displayed at that level:

  - Marks will be awarded at the top of that level if the student has evidenced each of the descriptor traits securely.
  - Where the response does not securely meet all traits, the marks should be awarded based on how closely the descriptor has been met.

Task 1- Planning a project

Indicative content and marker guidance

Gantt chart:

- Expect to see tasks broken down into smaller chunks where sensible for example:
  - May show use of an Agile approach (or similar)
  - Large modules (e.g., backend database, data analysis, etc.) broken down into multiple sections of development and unit testing with logical resources being applied to tasks – look out for learners applying the same developer to test the modules – can be ok if justified through testing experience
  - When splitting tasks learners should show an understanding of how total development hours should be split between the team working on it
  - A task related to implementation of their choice of **cloud or physical servers** should be seen here – the plan should only account for one or other, not both.
- There should be sensible use of concurrent and serial tasks for example:
  - Logical task sequences, e.g., server setup/installed before module deployed, unit testing before integration testing
  - Showing that as one unit is being tested, development could be taking place with other team members
  - Integration testing would be expected after specific modules have been unit tested
  - At the higher end, expect to see consideration around the testing time for each module and how this could be split up to allow testing along with other modules
  - Possible to see an Agile approach on a granular level, e.g., per module as well as the project as a whole
  - Should show some awareness of the requirements of tasks having predecessor requirements.
- There is no single correct way organise the plan, but task orders should be sensible, for example 'create a test plan' should occur BEFORE testing commences, staff training would occur much later in the process when the system is nearer completion, etc.
- The order and implementation of the project may vary significantly depending on the SDLC approach that they are taking (check against learner rationale). For example:
  - a RAD/Agile that looks at a minimum value product (MVP). They may 'deploy' some of the modules very early on, after a short portion of development time, and then test and develop further, deploying more modules as they go
  - Or they may decide that their approach is to have most modules mostly developed and then deployed and tested later.

The rationale will show the reasoning for the chosen project development approach demonstrated in the Gantt chart. Points learners may consider, although some of these will vary depending on the choices learners make in terms of organisation. These include but are not limited to:

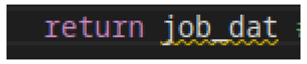
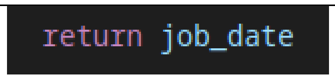
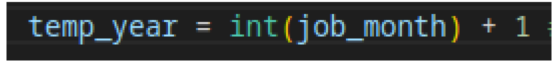
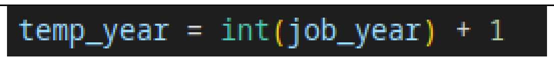
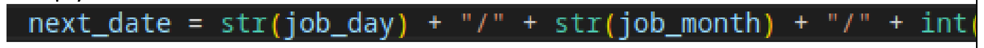
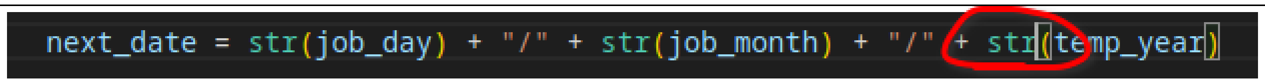
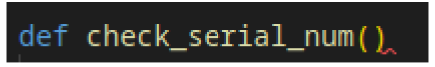
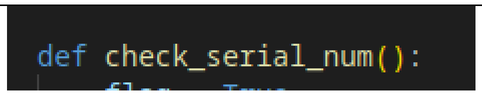
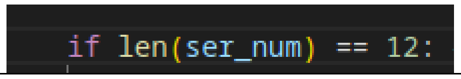
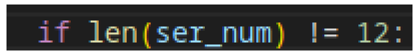
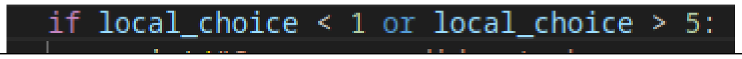
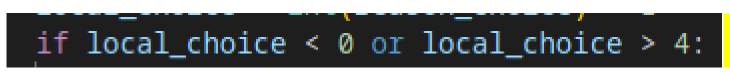
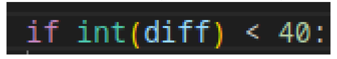
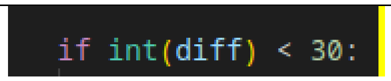
- Staff (skills, industry expertise, experience)
  - The Senior Software Engineer (Ellis Soto) has worked on large-scale projects and has experience of deployment.
  - The Junior Software Engineer (Zuzanna Misaczek) has less experience but has some clear experience around staff development and UI development. Has a wide range of skills so could be used in a flexible way.
  - Lead UI designer (Peter Evans) Very experienced in UI design, would be able to work with minimal supervision. Also has experience of training.
  - The Hardware and Networking Technician (Hou Zijin) – Experience of large-scale projects – highly specialised in Hardware and infrastructure. Would only expect to see them on these types of tasks (e.g., installation and testing of Hardware)
  - The Database Engineer (Darren Long) has lots of testing experience – as well as working on the database module, expect to see this team member testing across the range of modules. There is an option to use as part of staff training but would need to justify over the cheaper option (ZM).
  - Testing should be allocated to members of staff in a sensible manner �=� in some cases, in learner work, testers are being allocated as they are the developer – in some cases other members can be used to free up developers to start producing other modules – look to see how this is justified – the rationale.
- Resources
  - Learners should rationalise the choice of allocation of tasks to different members of staff (see above).
  - A justification of which server option they opt for – with reference to needing to employ additional staff or not, cost, etc.
  - Though costings vary – should show some logical methods used to calculate.

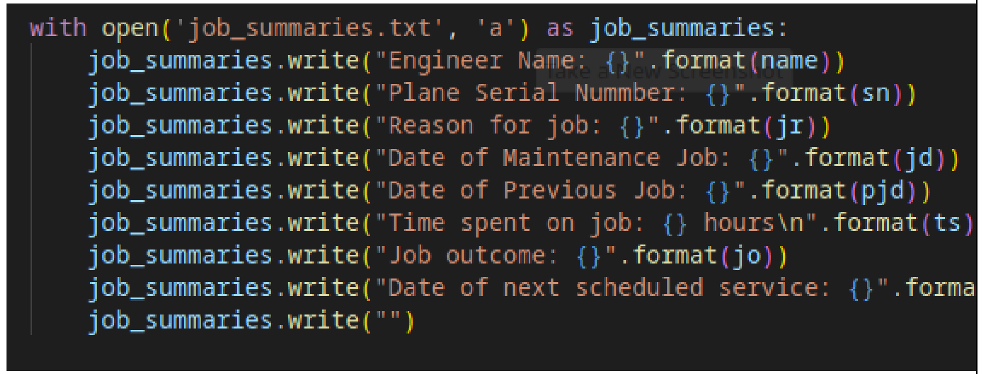
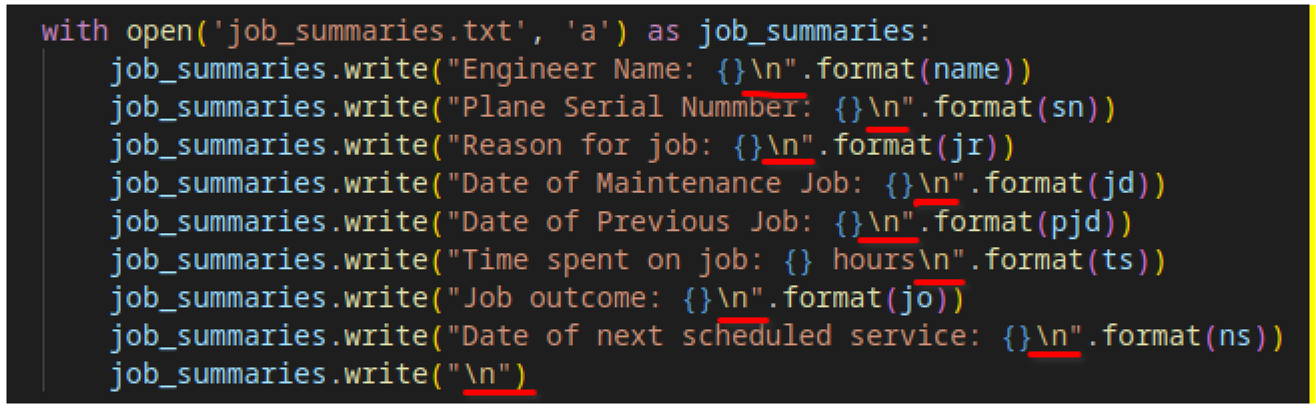
- Timescales and costs
    - Students should identify if they made the deadline or not. The number of hours for the project and available time is very close – with consideration of predecessor tasks. All staff working for the specified times per day for the duration is 2450 hours of work. The project tasks listed equal a total 862 or 877 hours depending on the server chosen.
    - "Total hours" for the project is simply the total work hours. The total hours does not directly convert to the number of days the project will take. There should be consideration of:
        - which jobs can only be done by specific individuals and how these work alongside/conflict with other tasks.
        - splitting of jobs between suitable team members
        - contingency time.
    - More contingency time may be needed on jobs performed by the junior staff members/senior members assigned to oversee work.
    - Students should discuss the cost of their solution with reference to how these costs were arrived at, and if the projected costs and increase in revenue make the project feasible – showing awareness that the project benefits will be seen over a number of years.
    - Students could look to justify using less experienced members of staff in order to keep costings down, though mitigation (e.g., overview by the senior member of staff, more time to allow for development) and rationale should be considered.
- Risks
    - Senior Software Engineer is only newly promoted to the role so may be inexperienced – this could lead to issues not being dealt with in a timely manner or may not adequately support the Junior Software Engineer.
    - Junior Software Engineer is very inexperienced and has only limited software development experience.
    - The software team is very small which means workloads may be difficult and any slip/slack would impact on timelines as there are no other developers to help.
    - Discussion of the relevant risks of a purely cloud deployed system vs the physical and the risks associated with employing an in-house technician vs paid for support from a third-party provider.
    - Timescale consideration around getting the project completed on time.

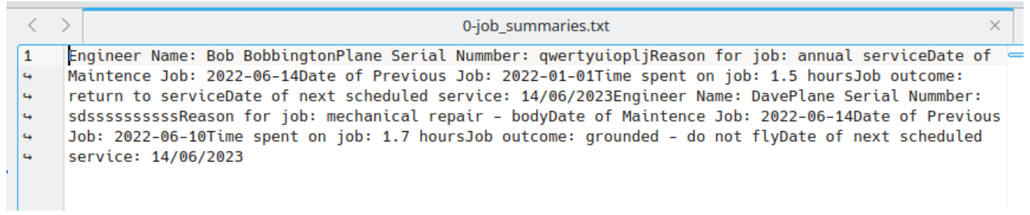
Assessment focus	Band 0	Band 1	Band 2	Band 3
	0	1-2	3-4	5-6
Gantt chart	No rewardable material.	Project tasks are somewhat organised in logical and efficient manner making some use of an appropriate SDLC model to provide some accurate prediction of the project's timescales. Resources have sometimes been assigned to project tasks effectively but there are some major and/or significant errors or omissions.	Project tasks are organised in a mostly logical and efficient manner making use of an appropriate SDLC model to provide mostly accurate predictions of the project's timescales. Resources have mostly been assigned to the project tasks effectively, but there are some minor errors or omissions.	Project tasks are organised in a thoroughly logical and efficient manner using an appropriate SDLC model to provide thoroughly accurate predictions of the project's timescales. Resources have consistently been assigned to the project tasks effectively.
	0	1	2-3	4
Resource and cost plan		Some correct resources and accurate costs have been added to the plan resulting in an estimate of limited accuracy.	Mostly correct resources and accurate costs have been added to the plan resulting in an estimate that is largely accurate.	Fully correct resources and accurate costs have been added to the plan resulting in an accurate estimate.
	0	1-3	4-6	7-9
Rationale	No rewardable material.	Rationale for project planning decisions demonstrates some effective consideration of: ● cost, risks, and benefits ● order and timing of tasks ● selection and allocation of resources ● dependencies and prerequisites.	Rationale for project planning decisions demonstrates mostly effective consideration of: ● cost, risks, and benefits ● order and timing of tasks ● selection and allocation of resources ● dependencies and prerequisites.	Rationale for project planning decisions demonstrates a thorough and perceptive consideration of: ● cost, risk, and benefits ● order and timing of tasks ● selection and allocation of resources ● dependencies and prerequisites.

Task 2- Identifying and fixing defects in existing code

Indicative content and marker guidance

Line number & Error category	Description of error	Possible fix
14 Minor Error	Error in variable name  `return job_dat`	 `return job_date`
37 Significant Error	Year for next service not calculated correctly. Adding 1 to the month instead of the year  `temp_year = int(job_month) + 1`	 `temp_year = int(job_year) + 1`
39 Minor Error	Temp year cast as int so will not concatenate  `next_date = str(job_day) + "/" + str(job_month) + "/" + int(`	 `next_date = str(job_day) + "/" + str(job_month) + "/" + str(temp_year)`
52 Minor Error	Syntax error – Missing colon at end of line  `def check_serial_num()`	 `def check_serial_num():`
59 Significant Error	Logical check for serial number length is incorrect. Should be checking for not equal to  `if len(ser_num) == 12:`	 `if len(ser_num) != 12:`
94 Significant Error	Range is incorrect for retrieving reason choice from tuple  `if local_choice < 1 or local_choice > 5:`	 `if local_choice < 0 or local_choice > 4:`
106 Significant Error	Number of days for safety check is incorrect  `if int(diff) < 40:`	 `if int(diff) < 30:`

184 – 194
/ txt file
Minor
Error
	Data is not formatted correctly when written to the txt file. All information is on a single line
	
	`with open('job_summaries.txt', 'a') as job_summaries:`
	`    job_summaries.write("Engineer Name: {}".format(name))`
	`    job_summaries.write("Plane Serial Nummber: {}".format(sn))`
	`    job_summaries.write("Reason for job: {}".format(jr))`
	`    job_summaries.write("Date of Maintenance Job: {}".format(jd))`
	`    job_summaries.write("Date of Previous Job: {}".format(pjd))`
	`    job_summaries.write("Time spent on job: {} hours\n".format(ts)`
	`    job_summaries.write("Job outcome: {}".format(jo))`
	`    job_summaries.write("Date of next scheduled service: {}".forma`
	`    job_summaries.write("")`
		Add 'newline' character to code
		
		`with open('job_summaries.txt', 'a') as job_summaries:`
		`    job_summaries.write("Engineer Name: {}\n".format(name))`
		`    job_summaries.write("Plane Serial Nummber: {}\n".format(sn))`
		`    job_summaries.write("Reason for job: {}\n".format(jr))`
		`    job_summaries.write("Date of Maintenance Job: {}\n".format(jd))`
		`    job_summaries.write("Date of Previous Job: {}\n".format(pjd))`
		`    job_summaries.write("Time spent on job: {} hours\n".format(ts))`
		`    job_summaries.write("Job outcome: {}\n".format(jo))`
		`    job_summaries.write("Date of next scheduled service: {}\n".format(ns))`
		`    job_summaries.write("\n")`

	 Engineer Name: Bob BobbingtonPlane Serial Nummber: qwertyuiopljReason for job: annual serviceDate of Maintence Job: 2022-06-14Date of Previous Job: 2022-01-01Time spent on job: 1.5 hoursJob outcome: return to serviceDate of next scheduled service: 14/06/2023Engineer Name: DavePlane Serial Nummber: sdssssssssssReason for job: mechanical repair - bodyDate of Maintence Job: 2022-06-14Date of Previous Job: 2022-06-10Time spent on job: 1.7 hoursJob outcome: grounded - do not flyDate of next scheduled service: 14/06/2023	

The test plan/log should also contain inclusion of tests that show how a range of aspects of the program have been tested, including areas of the program that appear to be coded correctly have been tested to ensure outputs are correct and the program is robust. These may include (but not limited to):

- Using different dates to test the 30-day safety criteria
- Using different dates to test the annual service +1 year calculation
- Different lengths of plane serial number with consideration of testing boundaries for logic (extreme date), i.e., 11, 12 and 13 characters long
- Non-numerical values for menu options
- Numbers for menu options that sit outside of the given range and at the boundaries of the range
- Checking menu options return the correct item to test indexing
- Checking the formatting of the outputted text file.

A limited understanding of program requirements would be typified by only identifying and fixing the minor errors that would be highlighted by the IDE, but would not identify and fix significant errors.

The number of errors identified is not a hurdle between Bands 2 and 3 for the first two assessment focus, the discriminator is the quality and appropriateness of the tests and data selected, and the level of understanding shown of the **process**.

Note – The suggested fixes are for guidance only, credit alternative solutions that are logically correct and would produce the correct/required outcomes. For example, the error in line 59 could be corrected by altering the Boolean flag value that is checked rather than the operator itself.

Assessment focus	Band 0	Band 1	Band 2	Band 3
	0	1-2	3-5	6-8
Use of testing to identify defects	No rewardable material.	Tests selected show a basic understanding of the identified program requirements. Test log includes some appropriate test data. Testing has identified some error in the code provided.	Tests selected show a good understanding of the identified program requirements. Test log includes a good range of normal, erroneous and extreme data. Testing has identified most errors in the code provided.	Tests selected show a thorough and detailed understanding of the identified program requirements. Test log includes a comprehensive range of normal, erroneous and extreme data. Testing has comprehensively identified the errors in the code provided.
		1	**2-3**	**4**
Understanding of the testing process		Test log shows a basic understanding of how errors/problems were identified and how they were rectified.	Test log shows a good understanding of how errors/problems were identified and how they were rectified.	Test log shows a thorough and detailed understanding of how errors/problems were identified and how they were rectified.
		1-3	**4-6**	**7-9**
The solution		Code has some functionality, but significant errors still persist. Changes made apply some precise logic and programming structures which would result in some correct outcomes.	Code is mostly functional, but some minor errors still persist. Changes made apply mostly precise logic and programming structures which would result in mostly correct outcomes.	Code is fully functional. Changes to code apply fully precise logic and programming structures throughout which would result in consistently correct outcomes.

Task 3- Designing a solution

Indicative content and marker guidance

General guidance

- Algorithm designs should demonstrate decomposition of the problem into simpler and more understood primitives.
- The design should provide high level coverage of the process as well as identify **reusable components.**
- Detailed algorithms (pseudocode) do not need to be provided for ALL repeated processes. For example, if the process for calculating number of passengers for each airport is very similar, the learner would not need to provide algorithms for each airport, rather they should show how **reusable** code and may provide some additional annotation to explain the process as necessary.
- Good decomposition will show all the necessary processes and sub-processes that make up the main problem.
- A decomposition diagram is not required, decomposition should be demonstrated through detail and process break down in the flowcharts or pseudocode

Some general characteristics of a good algorithm that may be demonstrated are:

- the steps are clearly defined
- each step is uniquely defined and should depend on the input and the result of the preceding steps
- receives input, selection of airports and dates, input coding for usability may also be considered
- produces appropriate type of output (e.g., screen display, return value or return list), which results are required, what happens if no results can be computed (e.g., an error message)
- sensible naming conventions for variables and processes
- use of key words, symbols, hierarchies, and structures as appropriate to the chosen method to represent the algorithm (i.e., pseudocode or flow chart).

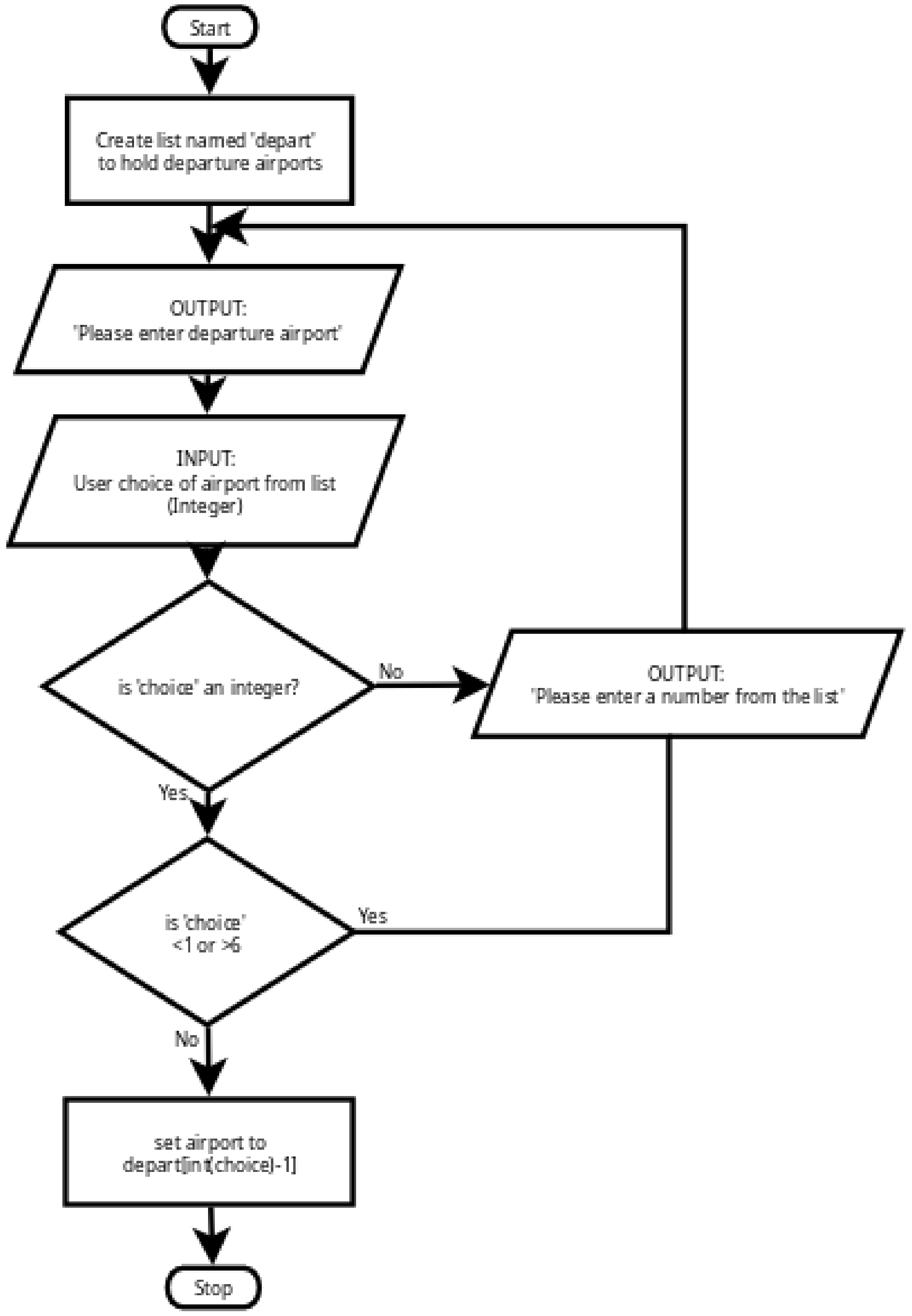
Scenario specific characteristics may include:

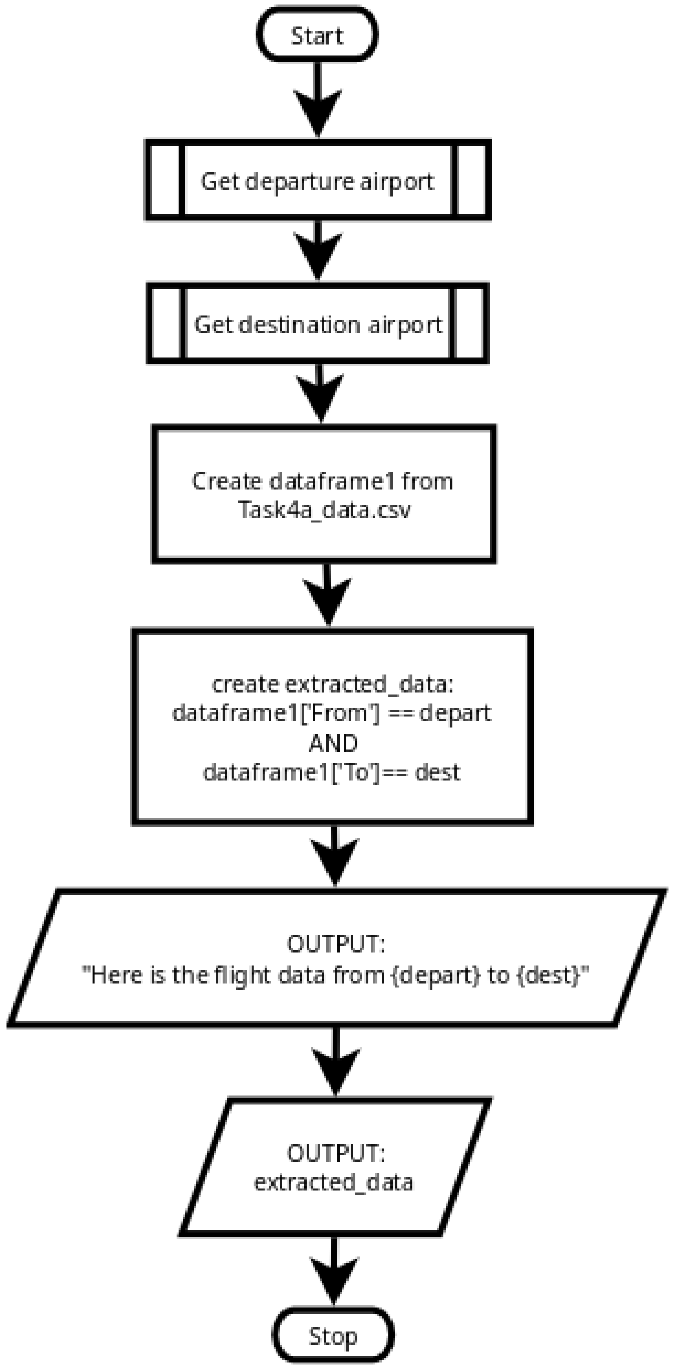
- suitable logic and calculations to define date range
- suitable calculations to calculate numbers of passengers, e.g., average passengers within a specific range, total numbers, etc.
- links to CSV to get destination, departure, and date with a date range, etc.
- sensible use of CSV or run-time data structure (e.g., data frame) to hold different parts of the data for processing, e.g.
  - list to hold menu options, or predefined date ranges, airport code
  - new data frame to hold data for the selected date range to aid calculations
- understanding of given data such as:
  - use of header row in CSV to locate required data
  - need to convert dates to a usable date format

simplification for user, e.g., choose a number from menu rather than type the region in full.

Example flow chart algorithms

Get departure airport


Start
Create list named 'depart' to hold departure airports
OUTPUT: 'Please enter departure airport'
INPUT: User choice of airport from list (Integer)
is 'choice' an integer?
No
OUTPUT: 'Please enter a number from the list'
Yes
is 'choice' <1 or >6
Yes
No
set airport to depart[int(choice)-1]
Stop


Start
Get departure airport
Get destination airport
Create dataframe1 from Task4a_data.csv
create extracted_data: dataframe1['From'] == depart AND dataframe1['To']== dest
OUTPUT: "Here is the flight data from {depart} to {dest}"
OUTPUT: extracted_data
Stop

Example detailed pseudocode algorithms for repeated processes:

Note – these are intended to be indicative of the types of algorithms that may be presented. These **do not** show all processes. Accept any responses that provide logically correct outcomes/solutions

Get and validate departure airport choice

```
WHILE not_valid_input_flag = TRUE
    SEND ‘"Please enter departure airport” TO DISPLAY
    SET depart TO [<list of departure airports>]
    RECEIVE depart_choice (integer) FROM KEYBOARD
    IF depart_choice choice NOT integer THEN:
        SEND ‘That is not  a valid choice’ TO DISPLAY
        SET not_valid_input_flag TO TRUE
    ELSE IF depart_choice <1 OR >6 THEN:
        SEND ‘That is not  a valid choice’ TO DISPLAY
        SET not_valid_input_flag TO TRUE
    ELSE:
        SET airport_name TO depart [int(depart_choice)-1]
    RETURN airport_name
```

Get and validate start date – CAST to date format

```
    SET non_valid_flag TO TRUE
    while flag == TRUE
        SEND ‘'Please enter end date for your time range DD/MM/YYYY') TO DISPLAY
        RECEIVE input (STRING) FROM KEYBOARD
        try:
            SET input(STRING) TO input(DATETIME)
        except:
            SEND "Sorry, you did not enter a valid date" TO DISPLAY
            flag = True
        else:
            SET end_date TO input(DATETIME)
            RETURN end_date
```

Get data based on number of days of data required

```
FUNCTION get_data( depart, dest,days)
BEGIN FUNCTION
    data_frame1 = pd.read_csv("Task4a_data.csv")
    extracted_data = data_frame1[From’] == depart  AND data_frame1[To’] == dest
    extract_days = extracted_data[: , -days: ]
    SEND ("We have found these flights that match your criteria:") TO DISPLAY
    RETURN extract_days
END FUNCTION
```

Assessment focus	Band 0	Band 1	Band 2	Band 3
	0	1-2	3-5	6-8
Decomposition of the problem	No rewardable Material.	Basic or superficial decomposition of the identified problems that superficially covers the required: • inputs • processes • outputs.	Mostly detailed decomposition of the identified problems that sufficiently covers the required: • inputs • processes • outputs.	Thorough and detailed decomposition of the identified problems that comprehensively covers the required: • inputs • processes • outputs.
		1-2	3-4	5-6
Application of logical thinking and conventions		Algorithms would produce some correct outcomes as a result of: • some precise logic • some appropriate structure and sequence which is likely to be inefficient. Some use of accepted conventions.	Algorithms would produce mostly correct outcomes as a result of: • mostly precise logic • appropriate structure and sequence but which may lack efficiency. Mostly appropriate use of accepted conventions though some minor inconsistencies may still exist.	Algorithms would produce correct outcomes as a result of: • precise logic • appropriate and efficient structure and sequence. Appropriate and consistent use of accepted conventions.
		1	2	3
Communication of the design		Superficial communication of the design uses technical language which is only sometimes appropriate for the audience.	Adequate communication of the design uses technical language which is mostly appropriate for the audience.	Effective communication of the design uses technical language which is appropriate for the audience.

Task 4a- Developing the solution

Indicative content and marker guidance

The solution

- Provides a developed coded solution that utilises the given code and adds the additional functionality as stated in the requirements.
- Integration of existing code may include:
  - ‘import’ function to pull code as a whole in when needed (note given code and learner code will need to be in the same folder)
  - integration into learner’s own code base as a function.
- The solution will be well structured and modular in nature – with clear subsections, this may be separated modules or the use of procedures, functions, or classes.
- Code will be annotated to aid future maintenance of the system.
- Data/information should be output in a meaningful way to the user. This may include use of a data frame, graph and/or text-based summary.
- Output data should show consideration of different **airports** over time, but this may be interpreted in slightly different ways. Such as:
  - higher level responses will make better use of patterns and trends over time and allow the user to select airports and AM/PM flights, as well as dates in different combinations to give more meaningful data analysis
  - lower level responses may only extract information based on a single criterion, e.g., a single departure airport, or calculate using only the number of previous days (as with starting code base), rather than providing options for specific date ranges.
  - learners make use of tabulated formats or graphs to display this information in a clearer way
  - lower level responses may extract some meaningful information but may only output in an unsorted fashion or may not give the user sufficient choice.
- There are different ways that this task can be interpreted so the characteristics of higher level responses will show a greater discrimination of the data to be extracted.
- Code should be robust, typical errors that may be accounted for in this scenario include negative values, non-numerical characters, entering a choice not provided in the menu.

Possible areas included that contribute to ‘user experience’:

- Outputs are meaningful and make sense to the user, e.g., outputs are accompanied by meaningful text to contextualise them
- Simplification of input processes e.g., use of numbers in a menu rather than writing the full airport codes
- User input converted from string to a usable date format
- Potential issue of difference between input date format and date format of imported data solved (e.g., “dayfirst = True”)
- Creation of new columns in a data frame to calculate averages, totals, etc.
- Use of visualisation, e.g., bar graphs to passenger numbers over time
- Helpful messages and robust input handling.

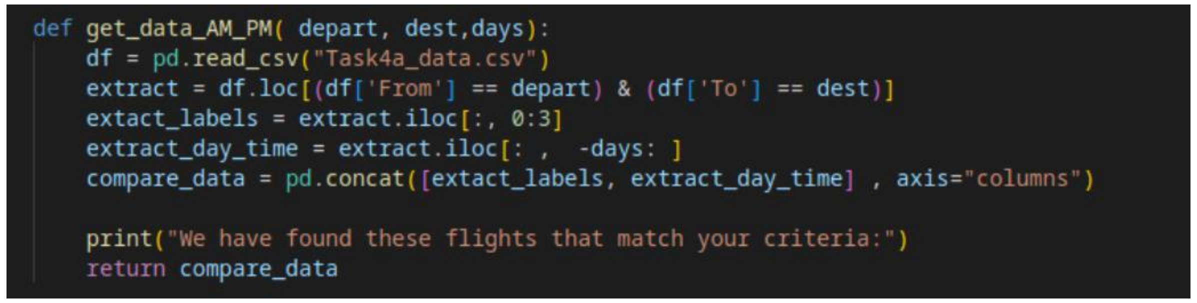
Security consideration relating to the solutions could include:

- Avoiding global variables, passing data back from functions
- Avoiding the use of a single generated data frame to ensure security of the data – good practice to generate a new data frame within a function
- Error handling: If the system crashes, is any data returned in the error message?

Example code snippets for parts of the solution:

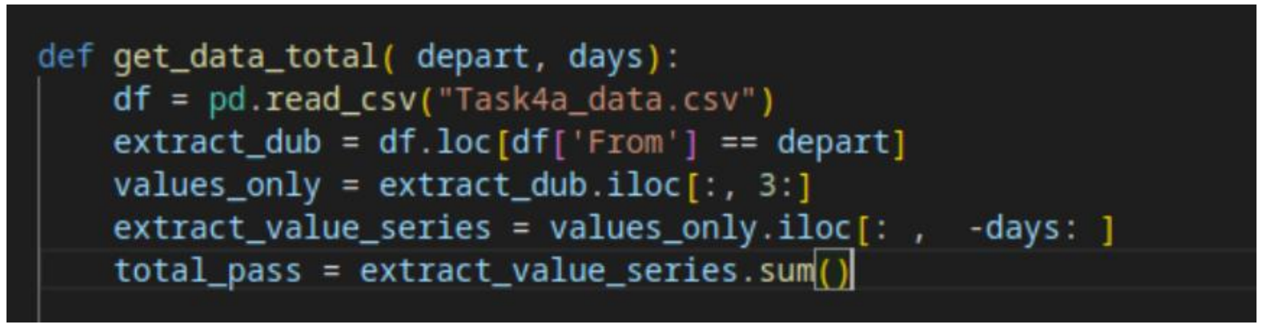
Note- these are intended to be indicative of the types of responses which may be presented. These do not show all processes. Accept any response that provides logically correct outcomes/ solution.

Extracting data from the csv file to gain AM/PM flights for a specific airport



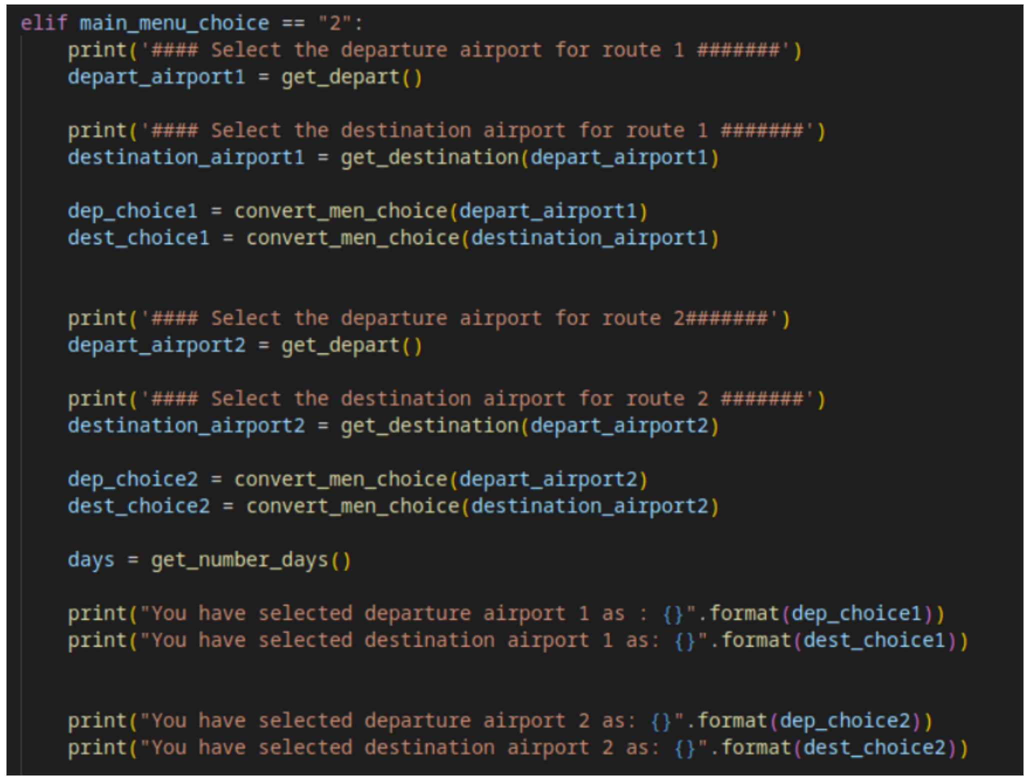
```python
def get_data_AM_PM( depart, dest,days):
    df = pd.read_csv("Task4a_data.csv")
    extract = df.loc[(df['From'] == depart) & (df['To'] == dest)]
    extact_labels = extract.iloc[:, 0:3]
    extract_day_time = extract.iloc[: ,  -days: ]
    compare_data = pd.concat([extact_labels, extract_day_time] , axis="columns")

    print("We have found these flights that match your criteria:")
    return compare_data
```

Showing total passengers for a specific departure airport



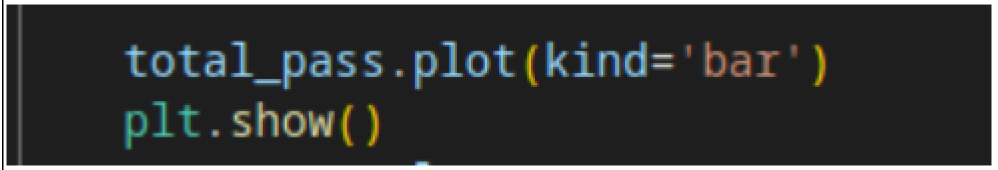
```python
def get_data_total( depart, days):
    df = pd.read_csv("Task4a_data.csv")
    extract_dub = df.loc[df['From'] == depart]
    values_only = extract_dub.iloc[:, 3:]
    extract_value_series = values_only.iloc[: ,  -days: ]
    total_pass = extract_value_series.sum()
```

Refactoring of given code to expand functionality



```python
elif main_menu_choice == "2":
    print('#### Select the departure airport for route 1 #######')
    depart_airport1 = get_depart()

    print('#### Select the destination airport for route 1 #######')
    destination_airport1 = get_destination(depart_airport1)

    dep_choice1 = convert_men_choice(depart_airport1)
    dest_choice1 = convert_men_choice(destination_airport1)


    print('#### Select the departure airport for route 2#######')
    depart_airport2 = get_depart()

    print('#### Select the destination airport for route 2 #######')
    destination_airport2 = get_destination(depart_airport2)

    dep_choice2 = convert_men_choice(depart_airport2)
    dest_choice2 = convert_men_choice(destination_airport2)

    days = get_number_days()

    print("You have selected departure airport 1 as : {}".format(dep_choice1))
    print("You have selected destination airport 1 as: {}".format(dest_choice1))


    print("You have selected departure airport 2 as: {}".format(dep_choice2))
    print("You have selected destination airport 2 as: {}".format(dest_choice2))
```

Use of matplotlib to produce a bar graph from a subset of the data (to accompany the printed table)



```python
    total_pass.plot(kind='bar')
    plt.show()
```

Assessment focus	Band 0	Band 1	Band 2	Band 3	Band 4
	0	1-2	3-4	5-6	
Functionality	No rewardable material	The solution implements code with some functionality, but some major errors still persist.	The solution implements mostly functional code, but some minor errors still persist.	The solution implements functional and efficient code throughout.	
		1	2	3	
Logic and programming structures		The code uses some precise logic and programming structures which would result in some correct outcomes.	The code uses mostly precise logic and programming structures which would result in sufficiently correct outcomes.	The code uses precise logic and programming structures throughout which would result in consistently correct outcomes.	
		1	2	3	
Robustness		The code handles some common user errors	The code handles most common user errors	The code thoroughly handles common, and most unexpected, user errors	
		1-2	3-4	5-6	
Security		The code mitigates against some common vulnerabilities as a result of some effective application of secure coding practices.	The code mitigates against most relevant vulnerabilities through mostly effective application of secure coding practices	The code thoroughly mitigates against relevant vulnerabilities through effective application of secure coding practices	
		1-2	3-4	5-6	7-8
Code organisation		The code is partially maintainable by a third party but would present significant difficulties through the use of: • inconsistent naming conventions • limited logical organisation • limited informative commenting.	The code is partially maintainable by a third party but would present some minor difficulties through the use of: • some consistent naming conventions • some logical organisation • some informative commenting.	The code is maintainable by a third party and would present only a few minor difficulties through the use of: • mostly consistent naming conventions • mostly logical organisation • mostly informative commenting.	The code is easily maintainable by a third party through the use of: • consistent and appropriate naming conventions • fully logical organisation • highly informative commenting.
		1-2	3-4	5-6	7-8
User experience		Basic user experience is provided through limited effective use of: • input handling • user guidance and error messages • outputs.	Adequate user experience is provided through somewhat effective use of: • input handling • user guidance and error messages outputs.	Good user experience is provided through mostly effective use of: • input handling • user guidance and error messages • outputs.	Excellent user experience is provided through consistently effective use of: • input handling • user guidance and error messages • outputs.

Task 4b- Reflective evaluation

Indicative content and marker guidance

Indicative content will vary according to the approach learners have taken in Task 4a and the effectiveness of the solution they created.

Generic features of effective evaluations are likely to include:

- the extent to which the solution meets the:
  - requirements of the Set Task Brief
  - needs of the users
- a justification of how the solution could be further developed/enhanced
- specific examples from the solution to support points made
- contextualisation of any points made and explaining what they did and justifying why.

Contextulisation for this scenario may include:

- How they solved issues with date formats vs strings
- Choice of data output (e.g., text, table, graph type) – which is most suitable for showing number of passengers over a period of time, staff sales data
- Rounding vs truncation for calculations (e.g., average passenger numbers) and consideration of data types to specify number of decimal places for the calculation output
- How the brief was interpreted – such as how ‘two selected routes’ was integrated into the functionality of the program, e.g., were different routes compared, was the user able to choose routes from a list?
- How existing code was integrated
- Choice of libraries/functions to get required data
- How data was extracted (e.g., use directly form the csv file, use of subsets of data, how the choice of data frame or datafile impacted on search/extraction)
- Use of variables (global vs local, passing data between functions)
- Input error handling, e.g., why they might have excluded text or negative values, menu options, etc.

Assessment focus	Band 0	Band 1	Band 2	Band 3
	0	1-2	3-4	5-6
Programming outcomes	No rewardable material.	Judgements reached are somewhat supported showing a superficial or basic understanding of how well the solution met the: ● requirements of the Set Task Brief ● needs of the users.	Judgements reached are mostly well supported showing a good understanding of how well the solution met the: ● requirements of the Set Task Brief ● needs of the users.	Judgements reached are comprehensively well supported showing a thorough and detailed understanding of how well the solution met the: ● requirements of the Set Task Brief ● needs of the users.
		1	2	3
Future Developments		A superficial or simplistic rationale is provided for what future developments should be implemented.	A good rationale which is reasonably well supported is provided for what future developments should be implemented.	A convincing and well-supported rationale is provided for what future developments should be implemented.