T Level Technical Qualification in Digital Production, Design and Development

Mark Scheme

Summer 2024

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

Gannt chart:

- Expect to see tasks broken down into smaller chunks where sensible, for example: o May show use of an Agile approach (or similar) o Large modules (e.g. back-end 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.
  - Hardware testing is a total value so at the higher end would expect this to be broken down and applied over time.
- There should be sensible use of concurrent and serial tasks, for example: o Logical task sequences, e.g. Infrastructure and server to be set up first and completed before modules are deployed, "Backend database" "stock and inventory" modules would be created before other modules as other modules will likely depend on data from these.
  - Showing that as one unit is being tested, development could be taking place with other team members. o Integration testing would be expected after specific modules have been unit tested. o 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. o 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: o 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.
  - One of the hardware and network technicians (Werneher Garza) is very inexperienced (an apprentice), it is unlikely that they would be able to do many tasks independently.
  - The HW staff are external contractors, it would not be sensible to allocate them to any other tasks than infrastructure, HW installation etc.
  - The cloud architect (Natalie Walterson) - Experienced as a developer in both front-end and backend, could be used in many different areas. If cloud server is chosen expect to see her working on that along with other web-based tasks.
  - Testing should be allocated to members of staff in a sensible manner – in some cases testers will be allocated as they are the developer – in some cases testing may be allocated to other member sof staff. There is no specific right solution, allocations should be sensible.

Rationale

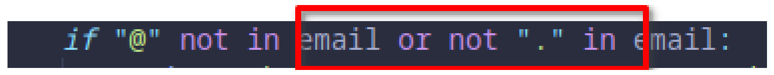
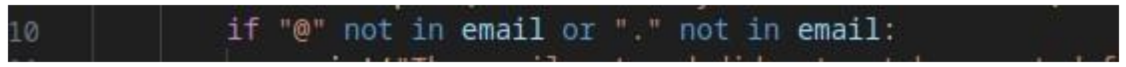
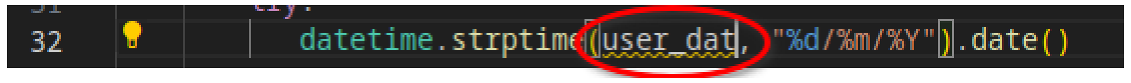
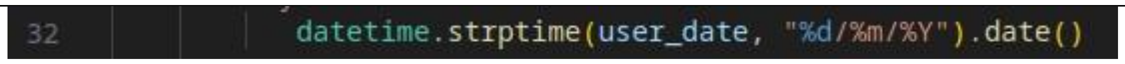
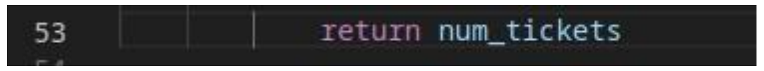
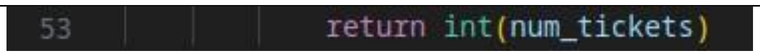
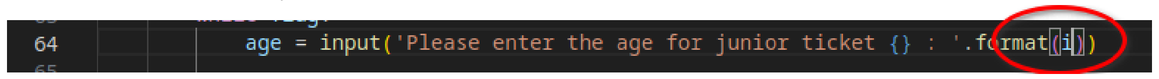
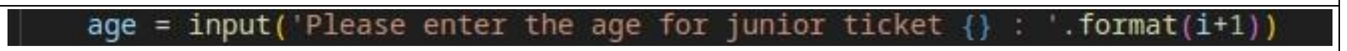
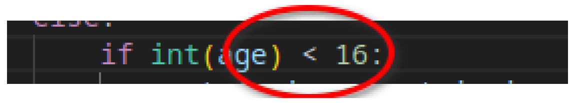
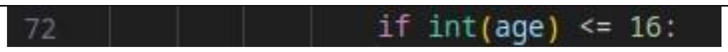
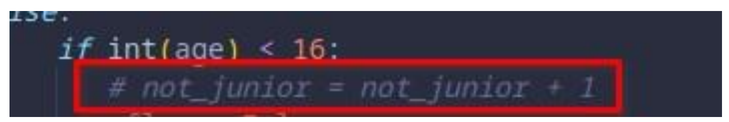
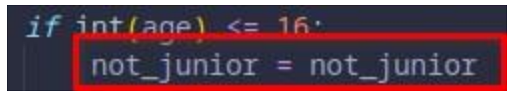
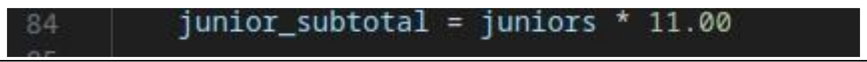
The rationale will show the reasoning for the chosen project development approach demonstrated in the Gantt chart. Points learners may consider, although some of these will vary depending on the choices learners make in terms of organisation. These include but are not limited to:

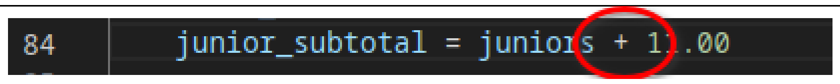
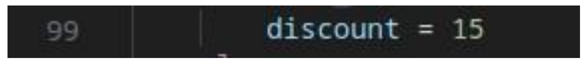
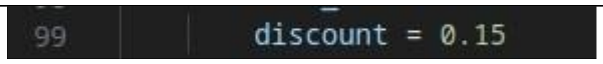
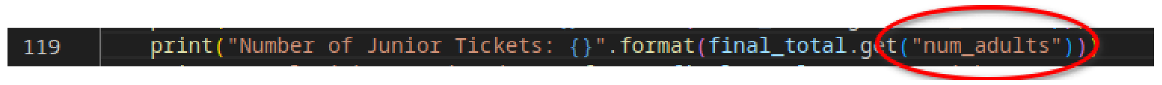
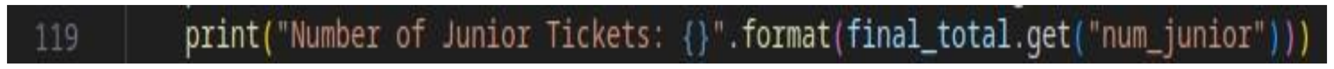
- o Staff (skills, industry expertise, experience)
- o The senior software engineer is very experienced and has experience on large scale projects, but they are new to working as team leader, so haven't proven themselves in a leadership role. Unsure how well they will manage staff/junior team members.
- o The junior software engineer (Rafe Bisset) - good level of experience and worked as a mobile app developer so suitable skills for the app module. o The web development engineer (Stanley Colt) Full-stack developer so can work on front end and back end. Very experienced in the role, but new to this company so quality of work and suitability as a team member is unknown, but potentially a very useful resource.
- o The hardware and networking technicians have clear expertise –but are external contractors so reliability and quality of work is not known. Being an external contractors, they would likely have a very strict remit in what they have been hired to do. Likely to be assigned to ONLY the HW and infrastructure tasks.
- o 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.
- Though costings vary – should show some logical methods used to calculate.
- Timescales and costs
- Learners should identify if they made the deadline or not. If a purely linear approach is used then the project could just about still meet the deadline. Using very simple calculations, the total work hours for the project is 1029.
- The requested time scale is 18 weeks - Using an efficient SDLC with concurrent tasks the deadline is easily manageable.
- Total hours for the project is simply the total work hours and does not directly relate to the number of days the project will take. There should be consideration of :
  - which jobs can only be done by specific individuals 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.
- Learners 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.
- The project is very expensive (around 400k), which will significantly impact profits in year 1 of the project.
- Learners could look to justify using less experienced members of staff to keep costings down, though mitigation (e.g. overview by the senior, more time to allow for development) and rationale should be considered.
- Risks
- Newly promoted team leader unproven at managing a team
- All hardware technicians are external contractors
- Some staff have key skills but from different industries and previous roles, as this isn't their main role this may impact on quality – quality vs meeting deadlines could be discussed by the learner.
- Potential staff absence, it is sensible to assume that there will be at least some minor staff absence over a period of 18 weeks so it would be wise to build contingency time into the project as a whole and/or the tasks assigned to each member of the team – this would be for higher level learners.
- A very small hardware team for such a large project – any slippage in time is unlikely to be able to be regained.
- Discussion of the relevant risks of decision between physical vs cloud server.
- Timescale consideration around getting the project complete 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 demonstrate 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 demonstrate 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 demonstrate 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	Description of error	Possible fix
10	Syntax/logical error – "not" in expected location in the statement.  `if "@" not in email or not "." in email:` **Identified through visual debugging** **Test data** should determine that it despite being unusual logic it still functions	 `10    if "@" not in email or "." not in email:`
32	Error in variable name of passed argument.  `32    datetime.strptime(user_dat, "%d/%m/%Y").date()` **Identified by IDE – error message**	 `32    datetime.strptime(user_date, "%d/%m/%Y").date()`
53	get_ticket_numbers function returns ticket numbers as a string instead of as an integer  `53    return num_tickets` **Identified by IDE – error message**	 `53    return int(num_tickets)`
64	Ticket number would output incorrect number (count start at 0)  `64    age = input('Please enter the age for junior ticket {} : '.format(i))` **Identified through test run / learner code check**	 `age = input('Please enter the age for junior ticket {} : '.format(i+1))`
72	Incorrect logical check for junior ticket – should be 16 and under (<=16)  `if int(age) < 16:` **Identified through test data**	 `72    if int(age) <= 16:`
73	Editing/debugging issue / poor programming practice  `if int(age) < 16:` `# not_junior = not_junior + 1` **Identified through visual debugging/logic check and test data to confirm**	 `if int(age) <= 16:` `not_junior = not_junior` Line should be either be updated to keep he not_junior variable unchanged, or removed completely.
84	Incorrect calculation/operator for junior subtotal	 `84    junior_subtotal = juniors * 11.00`

	 `84 junior_subtotal = juniors + 11.00` **Identified through test data**	
99	Incorrect discount value applied  `99 discount = 15` **Identified through test data**	 `99 discount = 0.15`
119	 `119 print("Number of Junior Tickets: {}".format(final_total.get("num_adults")))` Returns the wrong dictionary item **Identified through test run / learner code check**	 `119 print("Number of Junior Tickets: {}".format(final_total.get("num_junior")))`

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 numbers of tickets to check correct discounts are applied calculations
- Testing junior age boundary/index, e.g. 15, 16 and 17 to test logic is correct.
- Using over 16 to check that customers that are over 16 get charged for adult ticket instead • Manual calculations of totals compared to program outputs to check calculations are correct.
- Checking email format check functions correctly by using entries that:
  - Include both @ and '.'
  - Include either @ or '.'
  - Include neither
- Checking the formatting of the outputted data.

Allow learners to assume email test function is working or ignoring of odd layout of **not in** statement if they have used appropriate test data which implies it is working.

For line 73 – allow learners to have left the code commented out, however they should be demonstrating that this line is not needed/functions as it is through use of test data for the age

A limited understanding of program requirements would be typified by only identifying and fixing the functional errors that would be highlighted by the IDE, but would not identify and fix logical errors. The number of errors identified is not a hurdle between Bands 2 and 3, the discriminator is the quality and appropriateness of the tests and data selected, and the level of understanding shown of the process.

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 sales for different menu items is very similar, the learner would not need to provide algorithms for each region, rather they should show how **reusable** code would be utilised 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. These may include:

- ○ Importing data from the CSV file
- ○ Taking input to select item
- ○ Iterating through data for specified data
- ○ Taking a date range specified for data
- ○ A calculation related to ‘popularity’
- ○ Generation of data frames and graphs
- ○ Aspects related to validating user input.

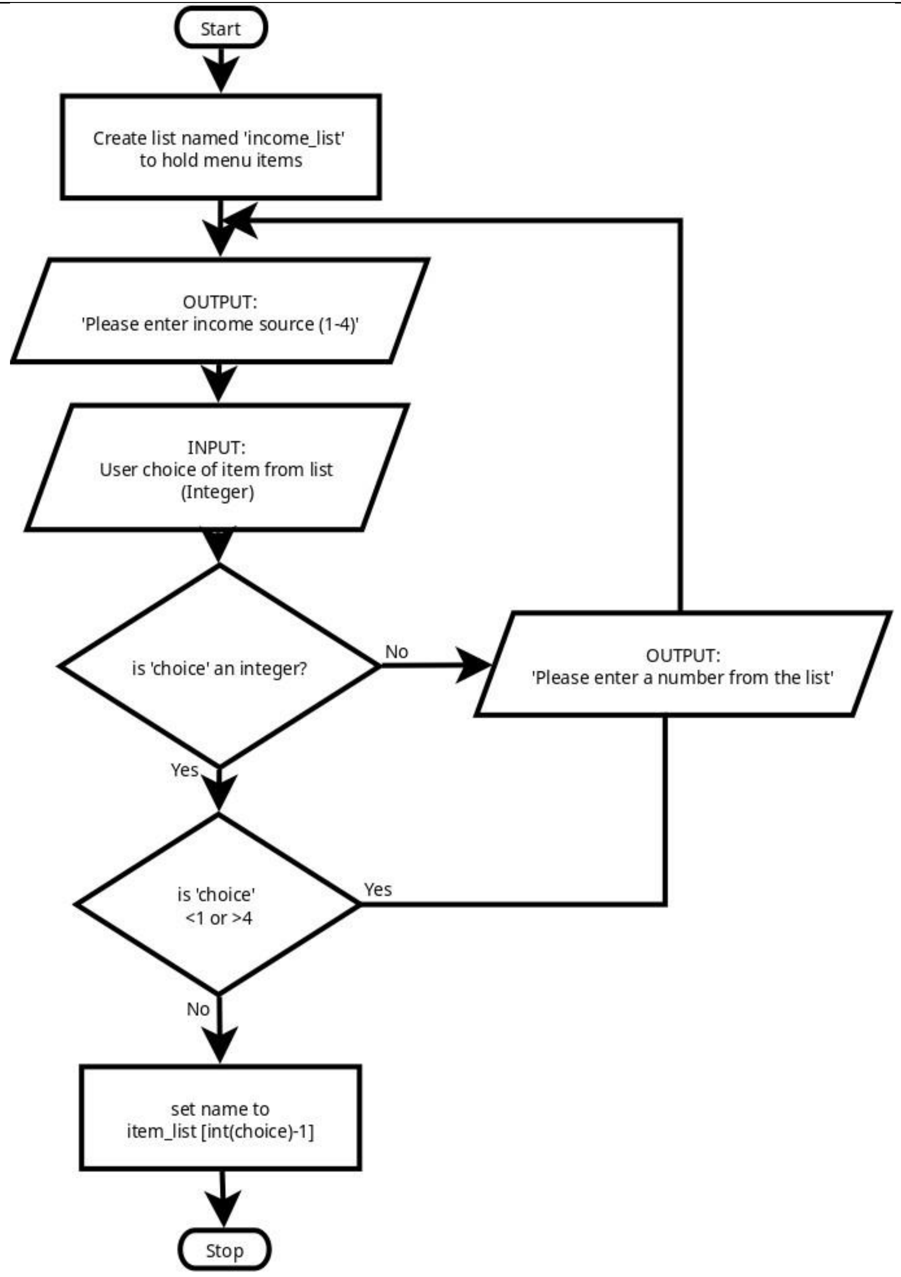
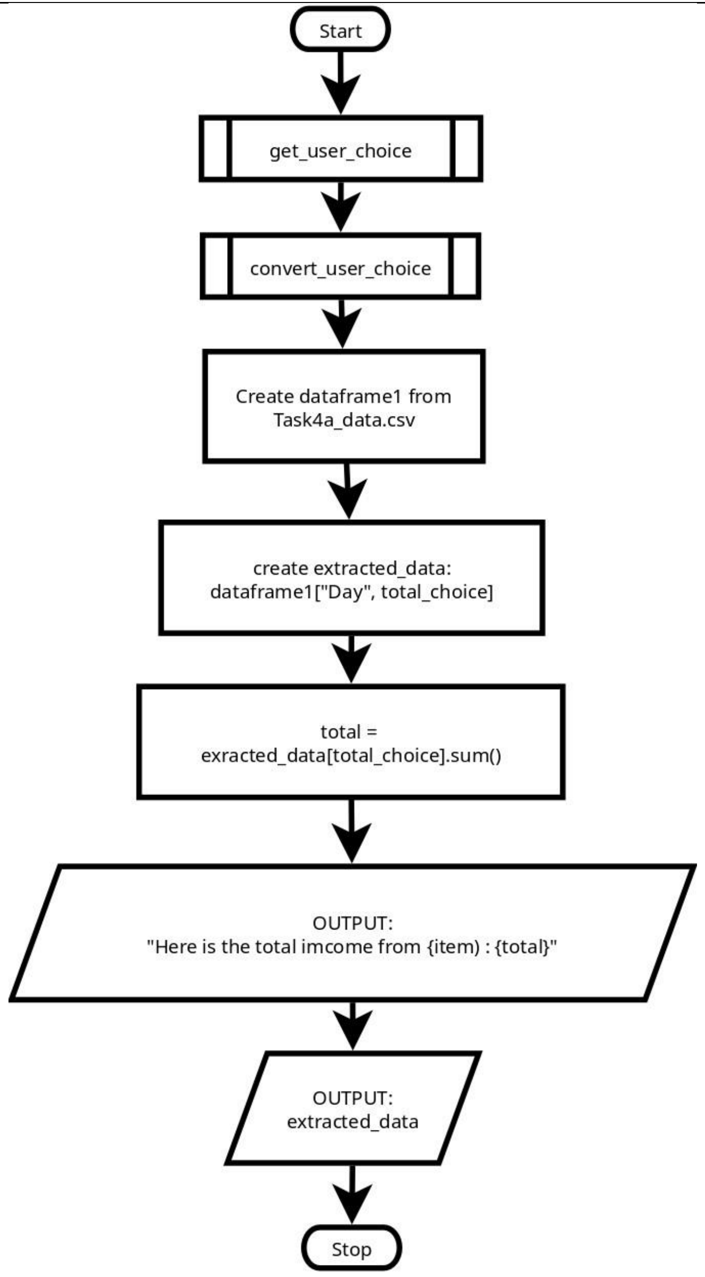
Some general characteristics of a good algorithm that may be demonstrated are:

- 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 income sources or day – type of input for user interface should 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, maybe error • 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 total based on the days selected
- Links to CSV to get items and sales
- 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 groupings
  - ○ 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 retrieve required data (e.g, the day of the week, specific income sources)
  - ○ Formatting data to appear as currency
  - ○ Consideration of ‘each week’ comparison of different days from the whole data set (e.g. total all Mondays)
- Simplification for user, e.g. choose a number from menu rather than have the user type a selection in full.

Example flow chart algorithms.

Get and validate Item choice	Calculate total
	

Start
Create list named 'income_list' to hold menu items
OUTPUT: 'Please enter income source (1-4)'
INPUT: User choice of item from list (Integer)
is 'choice' an integer?
No
OUTPUT: 'Please enter a number from the list'
Yes
is 'choice' <1 or >4
Yes
No
set name to item_list [int(choice)-1]
Stop

Start
get_user_choice
convert_user_choice
Create dataframe1 from Task4a_data.csv
create extracted_data: dataframe1["Day", total_choice]
total = exracted_data[total_choice].sum()
OUTPUT: "Here is the total imcome from {item} : {total}"
OUTPUT: extracted_data
Stop

Example detailed algorithms for repeated processes:

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

Get and validate income source choice FUNCTION

```
menu ():
        WHILE not_valid_input_flag = TRUE

                SEND list of items TO DISPLAY

        SEND '"Please enter the number of the income source you
        wish to view TO DISPLAY

        RECEIVE choice (integer) FROM KEYBOARD

        IF choice NOT integer THEN:
        SEND 'That is not  a valid choice' TO DISPLAY
        SET not_valid_input_flag TO TRUE
        ELSE IF item_choice <1 OR >4 THEN:
        SEND 'That is not  a valid choice' TO DISPLAY
        SET not_valid_input_flag TO TRUE
        ELSE:
        RETURN choice
```

Convert choice value

```
FUNCTION convert_total_menu_choice (choice):

        IF choice == "1":
                SET total_choice TO "Tickets"

        ELIF choice == "2":
                SET total_choice TO "Gift Shop"

        ELIF thoice == "3":
                SET total_choice TO= Snack Stand"

        ELSE:
                SET total_choice TO "Pictures"

        RETURN total_choice
```

Get data based on selected day and calculated total

```
IMPORT data handling library pandas

FUNCTION get_totals (total_choice):

        READ "Task3a_data.csv"
        SET dataframe TO Task3a_data.csv

        SET income TO  dataframe [["Day", total_choice]]

        SET total TO income[total_choice].sum()

        SEND  "The total income from {total_choice} was: £{total}"
        TO DISPLAY
```

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 cover the required: • inputs • processes • outputs.	Mostly detailed decomposition of the identified problems that sufficiently cover the required: • inputs • processes • outputs.	Thorough and detailed decomposition of the identified problems that comprehensively cover 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 – clearly see 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 salespeople over time but this may be interpreted in slightly different ways. Such as:
  - Higher level responses will make better use of filtering by payment source, day of the week and or income type in different combinations to give more meaningful data analysis.
  - Lower level responses may only extract based on a single criterion, e.g. a single income source or
  - 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 item names.
- User input converted from to a usable format e.g. capitalising days of the week.
- Creation of new columns in a data frame to calculate subtotals
- Numerical values are rounded to a sensible number of decimal places and currency symbol added as appropriate
- Use of visualisation, e.g. graphs to compare or show income over time.
- Helpful messages and robust input handling.

Security considerations relating to the solutions could include:

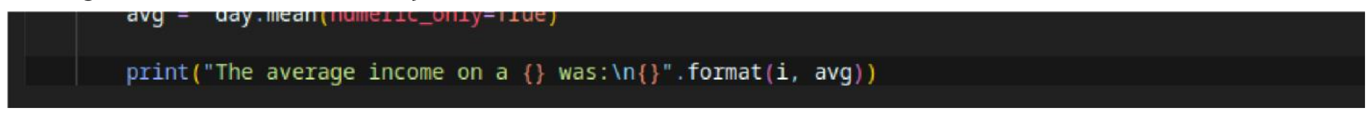
- Avoiding global variables, passing data back from functions.
- Avoiding the use of a single generated data frame to ensure security of the data – good practice to generate a new data frame within a function.
- Error handling: If the system crashes, is any data returned in the error message?

Example code snippets for parts of the solution:

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

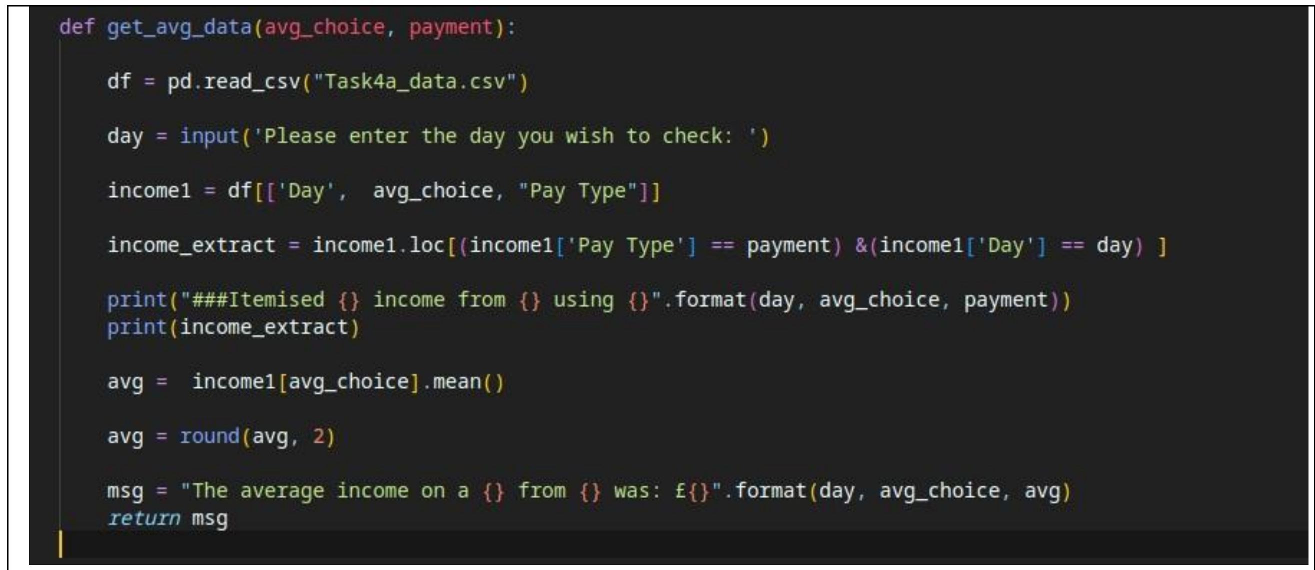
Extracting source and payment type based on a specific day, calculating average and outputting value to 2.dp by refactoring some of the provided code.

Average income from each day of the week:

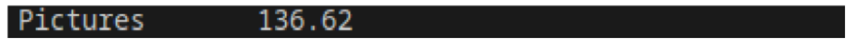


```
        avg =  day.mean(numeric_only=True)

        print("The average income on a {} was:\n{}".format(i, avg))
```

EXAMPLE OUTPUT



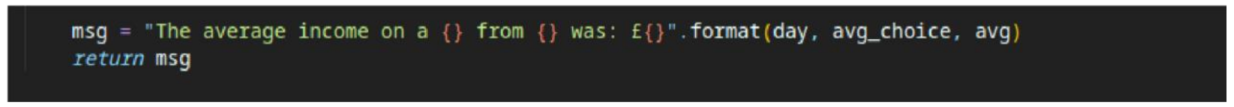
```
def get_avg_data(avg_choice, payment):

    df = pd.read_csv("Task4a_data.csv")

    day = input('Please enter the day you wish to check: ')

    income1 = df[['Day',  avg_choice, "Pay Type"]]

    income_extract = income1.loc[(income1['Pay Type'] == payment) &(income1['Day'] == day) ]

    print("###Itemised {} income from {} using {}".format(day, avg_choice, payment))
    print(income_extract)

    avg =  income1[avg_choice].mean()

    avg = round(avg, 2)

    msg = "The average income on a {} from {} was: £{}".format(day, avg_choice, avg)
    return msg
```



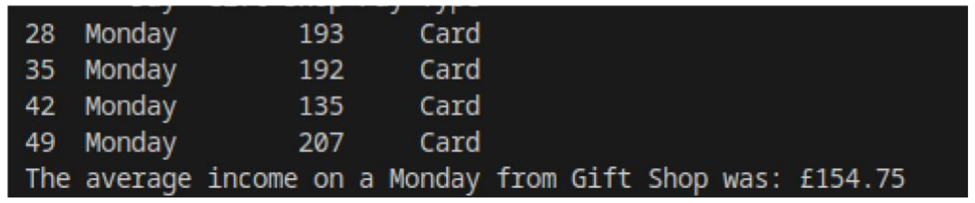
```
Pictures        136.62
```

Calculating average income on specific days and chosen payment type

Example OUTPUT:



```
    msg = "The average income on a {} from {} was: £{}".format(day, avg_choice, avg)
    return msg
```



```
28  Monday         193      Card
35  Monday         192      Card
42  Monday         135      Card
49  Monday         207      Card
The average income on a Monday from Gift Shop was: £154.75
```

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 code may lack efficiency and 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 totals, averages, days of the week, payment types in different time frames.
- Rounding vs truncation for calculations (e.g. average income) and consideration of data types to specify number of decimal places for the calculation output.
- How the brief was interpreted and integrated into the functionality of the program, e.g. were different income sources and payment types compared, was the user able to choose options from a list?
- How existing code was integrated.
- Choice of libraries/functions to get required data.
- How trends, patterns have been calculated, e.g. have user inputs for dates been considered, how has the information been grouped/sorted?
- 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.