Achievement Standard: 91372\_2.45version 3

**Construct a plan for an advanced computer program for a specified task**

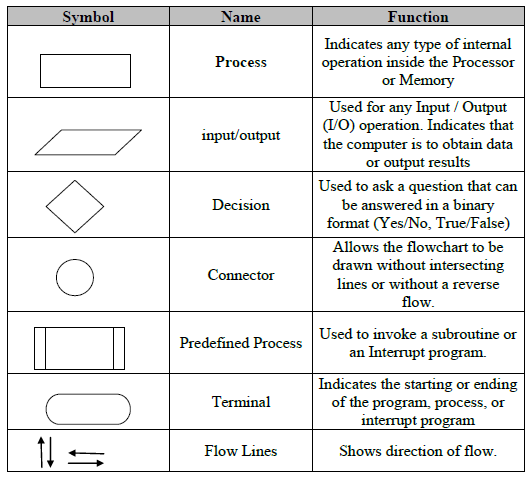
Achievement Standard: 91373\_2.46version 3

**Construct an advanced computer program for a specified task**

**Programming Assessments**

**STUDENT EVIDENCE**

**DOCUMENT**



**STUDENT NAME:** TJ Sutton **|** grade here in red

**Submission Date:** xx / xx / 2019

**Submission Number:** x

Task

You have been asked to write a plan for a program which can be used to compare the prices of various products based on their weight, volume and price.

Rather than writing down the price and weight or volume of each product and then calculating the ‘unit price’ for each item, you have decided to write a computer program to do all the hard work.

You will be assessed on…

How **successfully** your plan meets the requirements of the brief

How **clearly** you have described your plan.

The **efficiency** of your plan (eg: using functions to minimise repeated code, using variables and constants effectively and having a comprehensive test plan).

This is an individual task. You have 4 lessons (1 week) of time to complete it.

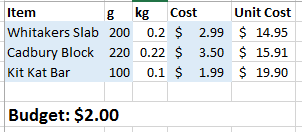
Basic Specifications

* On start-up, your program should ask the user how much money they have on-hand. You should decide on a suitable minimum amount (eg: $10.00)
* Your program should allow users to enter the details for multiple products that are being compared. For each product, the program should ask the user for…
* The product name
* The mass / volume of the product
* The price
* If users enter ‘XXX’ (or a suitable code), the program should then display
* The Item name and Unit price for each item being compared
* The average unit price
* The cheapest item
* The most expensive item
* A recommendation on which item should be purchased given the amount of money that the user has on-hand. Depending on how your program deals with prices which are more than what a user has to spend, this might not always be the product with the cheapest unit price.
* The program should then give users the option of starting over or quitting.

|  |
| --- |
| **Hint:** You can work out the Unit price by dividing the cost of an item by its volume / weight. You are allowed to use a spreadsheet to develop test data for expected values. |
| **Things to Consider**  Items can be measured in g / kg or mL / L. Your program should ensure that users enter sensible data. There are a number of ways of dealing with this issue. Below are two possible options…   * You could take a ‘no frills’ approach and tell users to enter all data to the nearest g / mL (or kg / L) * You could allow users to choose the unit and then set appropriate limits *(Think: would it make sense to allow users to enter a weight of 1000 kg? Would it make sense to allow 1000 g?)*   Will your program allow users to enter prices for goods where the amount is more than the person has to spend? Will you give users a choice so they can see which product is the best value (even if they can’t afford to buy the product at the current time). Please make a decision and then justify your choice (you could do this in the ‘steps’ part of your planning or as a comment at the top of your page). |

Possible Output Example

Below is some example test data. The test data is shaded in blue, the kg and unit cost for each item have been worked out using correct formulae. Arguably, the ‘g’ column is not required as the Unit Cost is in $ per kg.



The program should recommend that the user buy Kit Kat as they only have $2.00. Whilst Whittakers is the best value, our user does not have enough money to buy the 200g slab.

Planning Specifications : AS 91372

Develop a plan which will allow you to create a price comparison tool. Your plan must include:

• The variables that will be used and their data type

• A clear description of how the program will work – this can be presented as a series of   
 written steps and / or as a flowchart

• Reference to at least one indexed data structure (eg: a list)

• Functions with well-defined purposes. For each function, you should include a   
mini-flowchart or a series of steps showing what the function does

• A test plan which includes expected. boundary and exceptional (unexpected) input test cases

• A program based on your plan should be robust

• Your plan should not involve unnecessary duplication.

Task 1. Write the Pseudo Code for the program

Think through the process of what the program is asking, what will be needed before and during the running of it. Write out the list of steps. Don’t worry if you need to refine this one or twice.

Task 2. Draw a flowchart

As with Task 1, there are no marks for this, but a clearly detailed flowchart will assist with your coding.

Task 3. Identify the input information

Use the blank planning table – Variables. What information will the user have to enter?

Fill in the table, with the scope, data type and a brief description of each variable.

Add extra rows to the table if necessary.

Task 4. Identify the output information

What information will the program need to print out?

Add the required ‘outputs’ to your variable table.

Task 5. Identify any constants if necessary and identify indexed data structures

What data will need to be stored before the program can run?

Task 6. Determine what calculations are necessary

What data will need to be stored before the program can run?

Task 7. Develop a modular structure for your program

In your pseudocode or flowchart, include any sub-procedures/functions where required. Describe the modules or functions in terms of input and/or output where required.

Task 8. Create a set of input cases for testing the program

Create various test plans so that once the program has been created you can…

* Test that your program works for expected inputs
* Test that your program completes the calculations correctly
* Test that your program loops as expected

Task 9. Create a comprehensive set of expected, boundary and invalid input cases for testing the program (M and E)

Add to your testing plan and test data table to include boundary and invalid data input cases.

Add any additional statements to your modules where necessary.

Task 10. Refine the plan (M and E)

Make any additions to the plan so far to ensure that parameters for the modules are well chosen, each module will have a well-defined purpose within the context of the task, interaction between the modules is minimised, modules will be re-used rather than duplicated, procedural structure within each module will be efficient.

Task 11. Code your program

Refer to your flowchart and planning and code the program to answer the needs of the assessment. Refer to the marking sheet to ensure you answer required assessment criteria for each grade level.

============================================================================

Programming Specifications : AS 91373

Task 1. Review your plan

Read through your plan for the task.

Task 2. Create the program

1. Create a folder called ‘yourFullName\_Python Level 2’
2. Create a python file called ‘priceCompare\_numberX.py’
3. The number X is the version you are up to – save one each day.
4. Comment the code with:
   1. Your name
   2. Date
   3. Purpose
   4. Version
   5. Any notes

Task 3. Test your code

Test as you go. Once fully completed, use your testing tables to test inputs and boundaries.

Task 4. Update the testing and boundary tables

Paste in screenshot or make comment in the ‘Actual Outcome’ section of your testing tables.

If you make errors, then document this in the table, making comment on what happened. Then, when you get it correct, document that, commenting on what you did.

Part of the assessment is you making repeated attempts to correct errors and the teacher cannot see this unless you document your attempts as you go.

Task 5. Refine the program (M and E)

Make any additions to the program so far to ensure that you have used variables, constants, and derived values effectively so that your program is flexible and robust. Ensure that you have set out the program code concisely and documented the program with comments that explain and justify decisions about code behaviour. Ensure that the procedural structure within each module will be efficient.

EVIDENCE BEGINS HERE

Task 1. Write the Pseudo Code for the program

Task 2. Draw a flowchart

Task 3, 4, 5 Identify the input / output / process / constants information

Must test every input variable and with at least two data inputs

|  |  |  |  |
| --- | --- | --- | --- |
| **Input / Output / Process / Constants** | **Variable Name** | **Scope / Data-type** | **What this Variable is for** |
| input | cash\_on\_hand | float | *Used to record the cash the user has available to purchase products with* |
| Input | new\_product | float | *Variable that records the new product the user is adding* |
| Input | new\_mass | float | *Variable that records the mass for the new product the user is adding* |
| Input | new\_price | float | *Variable that records the price for the new product the user is adding* |
| Input | user\_input | integer | *Variable that records user input for the menu.* |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Task 6. Determine what calculations are necessary

Task 7. Develop a modular structure for your program

Task 8. Create a set of input cases for testing the program

**VALID TESTING**

**Purpose:** To test all variables work as expected. You need ***TWO*** Valid tests per input.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Test No.** | **Which input is being rested** | **Data entered to test the input** | **Reason for test** | **Expected outcome** | **Actual outcome** | **Pass / Fail** |
| 1 |  |  |  |  |  |  |
| 2 |  |  |  |  |  |
|  |  |  |  |  |  |  |
| 3 |  |  |  |  |  |
| 4 |  |  |  |  |  |  |
| 5 |  |  |  |  |  |
| 6 |  |  |  |  |  |  |

Task 9a. Create a comprehensive set of expected, boundary cases for testing the program (M and E)

**BOUNDARY TESTING**

**Purpose:** To test the limits of the code. You need **TWO** tests per boundary.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Test No.** | **Which Boundary is being rested** | **Data entered to test the input** | **Reason for test** | **Expected outcome** | **Actual outcome** | **Pass / Fail** |
| 1 |  |  |  |  |  |  |
| 2 |  |  |  |  |  |
|  |  |  |  |  |  |  |
| 3 |  |  |  |  |  |
| 4 |  |  |  |  |  |  |
| 5 |  |  |  |  |  |
| 6 |  |  |  |  |  |  |
|  |  |  |  |  |

Task 9b. Create a comprehensive set of invalid input cases for testing the program (M and E)

**INVALID TESTING**

**Purpose:** To make sure that invalid inputs will not be allowed in the code. You need TWO invalids per input.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Test No.** | **Which input is being rested** | **Data entered to test the input** | **Reason for test** | **Expected outcome** | **Actual outcome** | **Pass / Fail** |
| 1 |  |  |  |  |  |  |
| 2 |  |  |  |  |  |
|  |  |  |  |  |
| 3 |  |  |  |  |  |  |
| 4 |  |  |  |  |  |
| 5 |  |  |  |  |  |  |
| 6 |  |  |  |  |  |
|  |  |  |  |  |  |  |

**TASK CHECKLIST – 91372 - Planning**

|  |  |  |  |
| --- | --- | --- | --- |
|  | Assessment Criteria | ✓ | Evidence |
| Achieved | The plan meets the requirements of the specifications. |  |  |
| The plan uses sensible variable names. |  | Input/output/ tables |
| The plan has identified all the variable types. |  |
| The plan has identified the required inputs and outputs |  |  |
| The plan specifies where indexed structures will be used |  | Steps / variables |
| The plan has sequence (steps in order). |  | Steps |
| The plan has selection (choice of actions). |  |
| The plan has appropriate calculations / formulae |  |
| The plan has iteration (a loop). |  |
| The plan includes a test plan which  Tests that user input is valid  Tests that the calculations are correct  Tests that the program loops correctly |  | Test plan |
| Merit | Work has been done independently. |  | Teacher observation |
| The plan will lead to a program which is structured with a suitable introduction, middle and end. |  | Steps |
| The plan has well-chosen functions, actions, conditions and control structures (*least amount of coding as possible*). |  |
| The plan is flexible and robust |  |
| A program based on the plan will work for unexpected data. |  |
| The plan includes testing expected and boundary cases |  | testing table |
| Excellence | Modules have been used resulting in a well-structured, logical decomposition of the task. |  | testing table |
| Variables, constants and derived values have been specified so that the plan is flexible and robust with no unnecessary duplication. |  | Steps |
| The plan includes testing expected, boundary and exceptional cases |  | Testing tables |

**TASK CHECKLIST – 91373 - Implement (Coding)**

|  |  |  |  |
| --- | --- | --- | --- |
|  | Assessment Criteria | ✓ | Evidence |
| Achieved | Program uses sensible variable names |  | Program code |
| Program has input and output. |  |
| Program has at least one list |  |
| Program has procedures |  |
| Program has a modular structure |  |
| The plan has sequence (steps in order). |  | See the Planning Section flowchart or Pseudo Code |
| The plan has selection (choice of actions). |  |
| The plan has appropriate calculations / formulae |  |
| The plan has iteration (a loop). |  |
| Program code is set out clearly with suitable comments |  |  |
| Testing and debugging for each version of the code has been documented |  | Evidence template |
| Program works correctly on expected input cases |  |  |
| Code has been saved |  |  |
| Printout of final code has been provided |  | Code print out |
| Merit | Program was developed independently. |  | Program code |
| Variables and scope have been well chosen |  |
| Functions and parameters have been well chosen |  |
| Comments accurately describe code behaviour |  |
| Program works correctly on expected and boundary input cases. |  | Evidence template |
| Excellence | Modules are well structured and logical.   * Each module has a clear, well defined purpose * Interaction between modules is minimised * Modules are re-used rather than duplicated * The procedural structure of each module is efficient |  | Program code |
| Variables, constants and derived values ensure that the program is flexible and robust. |  |  |
| The program has been comprehensively tested and debugged |  |  |
| Student is organised. Time has been used efficiently |  |  |
| Program works correctly on expected, boundary and exceptional input cases. |  | Evidence template |

Program | Assessment schedule: Digital Technologies 91372

|  |  |  |
| --- | --- | --- |
| **Evidence/Judgements for Achievement** | **Evidence/Judgements for Achievement with Merit** | **Evidence/Judgments for Achievement with Excellence** |
| The student has constructed a plan for an advanced computer programme | The student has skilfully constructed a plan for an advanced computer programme | The student has efficiently constructed a plan for an advanced computer |
| The student has constructed a plan for an advanced computer program with some **teacher guidance.**   * A main module has been planned with calls to other modules that have a procedural structure for a sub task. * The modules could include parameters where needed | The student has **independently** constructed a plan for an advanced computer programme.  *See bullet points for achieved* | The student has constructed a plan for an **efficient** programme. The task has been broken down into several logical modules.   * A main module has been planned with calls to several other modules that have a distinct purpose.   *For example, the program includes functions to*   * *Validate numeric data (ie: check that weight, volume and price are numbers above zero)*. * *Ensure the item name is not left blank* * *Convert from g to kg / mL to L* * There is little code duplication between modules. |
| The plan specifies variables, their scopes and data types  *Some examples may include:*   * *username, global, string* * *budget, global, float* * *mass\_volume, global, float* * *product\_name, global, string* * *product\_unit, global, string* * *product\_amount, global float* * *price, global float* * *valid, local within data validation functions, boolean* * *response, local within data validation functions* | The plan specifies **well-chosen** scopes for the variables  *See achieved for possible variable name examples. Variable names will be sensible and all in lowercase.* | The plan specifies variables, constants, and derived values effectively so as to **maximise the flexibility and robustness** of the plan  *See achieved for possible variable name examples. Variable names will be sensible and all in lowercase. An excellence plan should have a relatively high number of local variables as functions should be used to minimise code repetition and maximise flexibility.* |
| The plan specifies an indexed data structure  *An indexed structure could be a list to hold each item’s name, weight, cost and unit cost.* | The plan specifies a **well-chosen** indexed data structure  *An indexed structure could be a list to hold each item’s name, weight, cost and unit cost.* | The plan specifies multiple indexed structures / a nested list if appropriate  *The plan includes a \*temporary\* ‘Item’ list which is used to store information about each item. Each item is then appended to a larger ‘Entries’ list which can be sorted by the cheapest item.* |
| The plan specifies a modular structure for the program, including details of the procedural structures of the modules  The task has been broken down into a main module and at least one other module. The main module calls the other module that has a procedural structure for a sub task.  *For example: The main module asks for the username and uses a ‘not\_blank’ function to check that the name is not left blank. This function is also used to check that the name of items being compared is not left blank.* | The plan specifies **well-chosen parameters** for the modules  Student has accurately planned the parameters that will be passed into their modules:  *For example:*   * *The ‘not\_blank’ function takes in a question as a parameter and returns a response string.* * *The ‘num\_check’ function takes in a question, a minimum value and a maximum value and returns a valid float* | The plan specifies modules (including their procedural structures) that constitute a well-structured logical decomposition of the task  Students has minimised code repetition in their plan. The functions that they have planned to use might include:   * *A not\_blank function for checking that user name and item name are not blank* * *A yes / no function for validating responses to yes / no questions* * *A number checking function* * *A conversion function (to convert g to kg / mL to L)* |
| The plan specifies a set of expected input cases for testing the program  *For example:*  *Username: Joe Bloggs*  *Budget: $10.00*  *Items as follows:*  *Whitakers Dark Chocolate, 250g, $3.50*  *Cadbury Dark Chocolate, 200g, $2.99*  *Kit kat, 50g, $1.50* | The plan specifies a set of **expected** and **boundary** input cases for testing the program  *See achieved for expected case*  *Should have planned cases for boundary values*  *Budget: $1.99, $2.00*  *Cost of $0.00, $0.01, $100.00, $100.01*  *Weight / volume of: 0, 1g, 20kg, 20.1kg* | The plan specifies a comprehensive set of **expected**, **boundary** and **exceptional** input cases for testing the program  *See achieved / merit and…*  *Should have invalid inputs*  *Name / item name <blank>*  *Cost: -$1.00*  *Weight: -1kg, “hello world”* |

Plan | Assessment schedule: Digital Technologies 91373

|  |  |  |
| --- | --- | --- |
| Evidence/Judgements for Achievement | Evidence/Judgements for Achievement with Merit | Evidence/Judgements for Achievement with Excellence |
| Construct an advanced computer program for a specified task. | Skilfully construct an advanced computer program for a specified task. | Efficiently construct an advanced computer program for a specified task. |
| With some guidance, the student has:   * implemented a plan for an advanced program in a suitable programming language   All the specifications outlined in the task have been met.   * *The program asks for the users’ name and budget* * *It asks if the item is being sold by weight or volume* * *It prompts the user to enter the name, weight and cost of items until a suitable exit code (such as XXX) is entered* * *It checks that the username is not blank* * *It checks that the weight / volume of the item is less between 0.1 kg / L and 20kg / 20L* * *It checks that the budget is $2.00 or more* * *It checks that the cost of each item is between 0 and $100* * *It correctly calculates and displays the unit price for each item* * *It correctly calculates and displays the cheapest item, the most expensive item and the average unit cost for the items being compared* * *It allows the user to make multiple comparisons (ie: it loops if required)* | The student has:   * **independently** implemented a plan for an advanced program in a suitable programming language that uses well-chosen scopes for variables, and well-chosen parameters for modules   *See achieved and…*   * *The program should ask the user for the units for each item and convert to kg / L if necessary* * *The program recommends the cheapest item (by unit cost) provided that the price is not more than the budget price*   *Eg: given the items below and a budget of $2.00, the program should recommend that the user purchase ‘Kit Kat’ as they don’t have enough money to buy the Whittakers slab (even though it is cheaper weight for weight).* | The student has:   * constructed an advanced program where the modules (including their procedural structures) constitute a well-structured logical decomposition of the task * used variables, constants, and derived values effectively so as to increase the flexibility and robustness of the program   *See Merit*  *In addition to meeting the task specifications, the code should be efficient and robust.*  *For example, one function is used to validate all numeric data. The number checking function might have the following parameters:*   * *question* * *low number (lowest allowed value)* * *high (highest allowed value)* * *error (error message to be displayed if invalid data is entered)* |
| * set out the program code clearly and documented the program with comments * *The student has written well formatted code with easy to read comments.* * *Functions generally have comments at the start which clearly state their purpose* | * documented the program with variable and module names and comments that accurately **describe** code function and behaviour   *For example the comment above the number checking function might look like this….*  *# Checks that input is a number between a minimum and maximum possible value* | * set out the program code **concisely** and documented the program with comments that **explain** and **justify** decisions   *For example the comment above start of the while loop for repeating the code might read*  *# Uses <enter> to loop or any key to quit (this makes it impossible for user to enter an invalid response and is faster than having to type ‘yes’ / ‘no’).* |
| * tested and debugged the program to ensure it works on a sample of **expected** input cases.   *For example shows expected and actual output for the following:*  *:*  *Username: Joe Bloggs*  *Budget: $10.00*  *Items as follows:*  *Whitakers Dark Chocolate, 250g, $3.50*  *Cadbury Dark Chocolate, 200g, $2.99*  *Kit kat, 50g, $1.50* | * tested and debugged the program in an **organised** way to ensure it works on inputs that include both **expected** and **boundary** cases.   *As for achieved and shows expected and actual output for the following boundary cases:*  *Budget: $1.99, $2.00*  *Cost of $0.00, $0.01, $100.00, $100.01*  *Weight / volume of: 0, 0.1kg (ie: 100g), 20kg, 20.1kg* | * **comprehensively** tested and debugged the program in an organised and time‑effective way to ensure the program is correct on **expected**, **boundary** and **invalid** inputs.   *As for merit and shows expected and actual output for the following boundary cases:*  *Should have invalid inputs*  *Name / item name <blank>*  *Cost: -$1.00 <ie: ‘$’ included as a string>*  *Weight: -1kg, <ie: ‘kg’ included as a string>*  *“hello world”* |