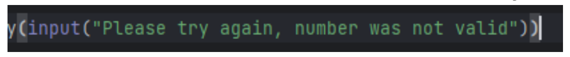
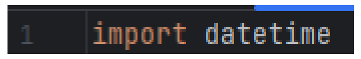
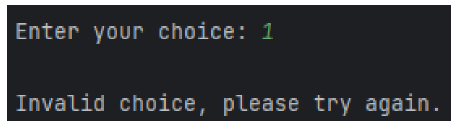
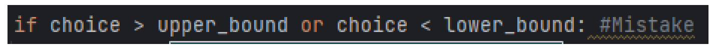
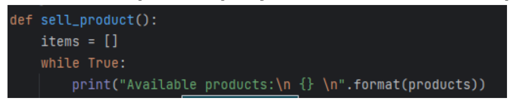
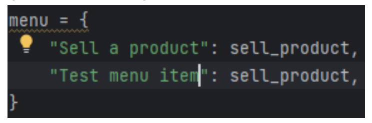
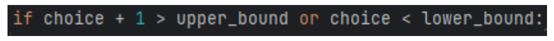
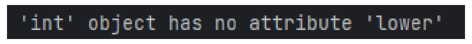
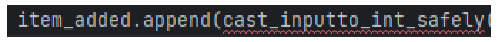
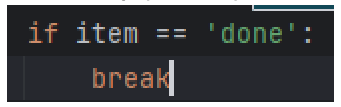
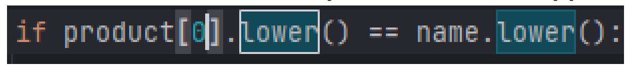
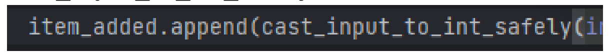
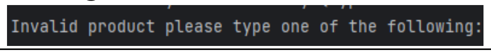
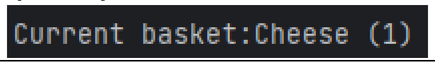
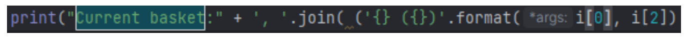
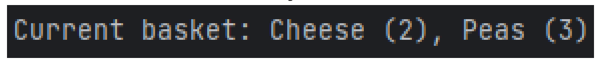
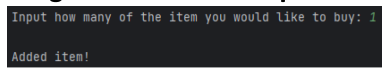
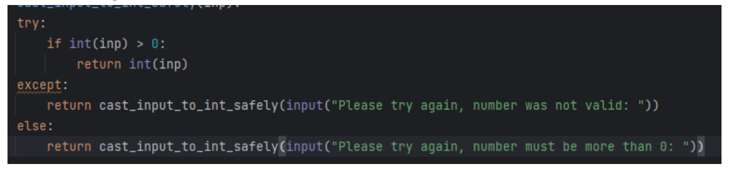
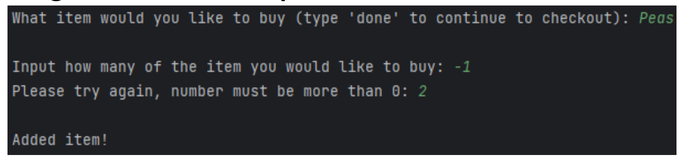
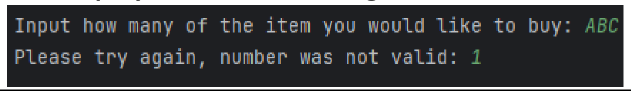
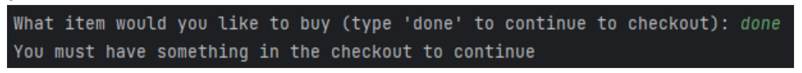
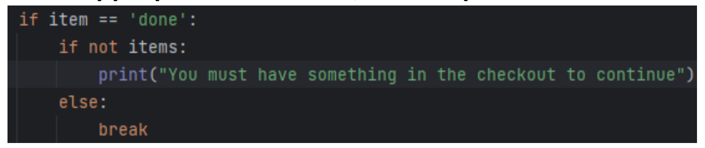
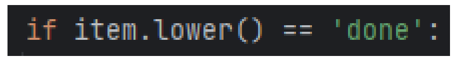
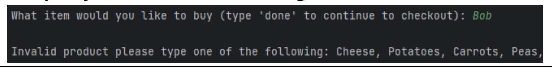
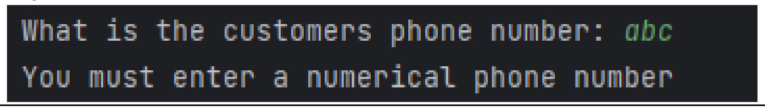
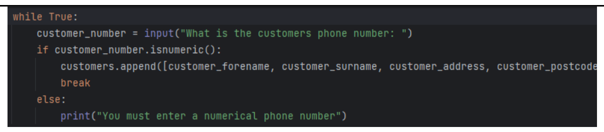
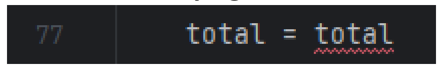
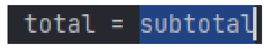
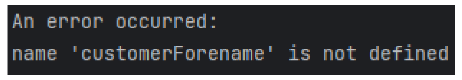
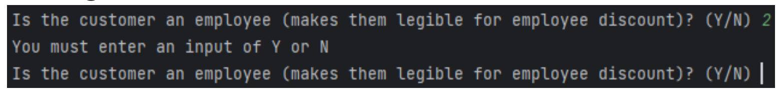
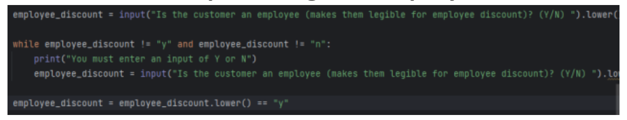
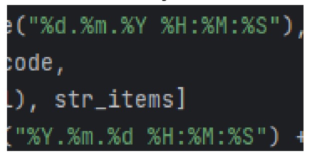
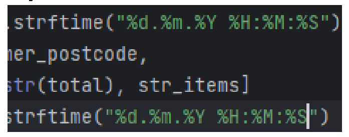
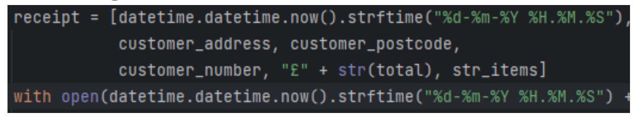
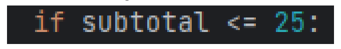
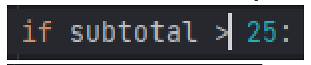
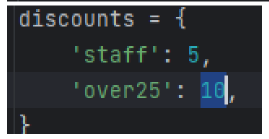
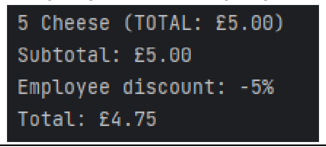
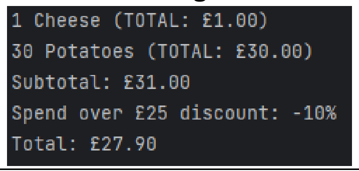
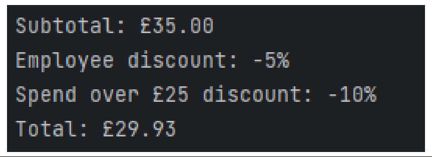
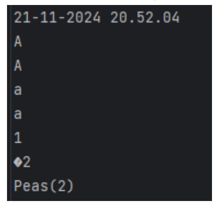
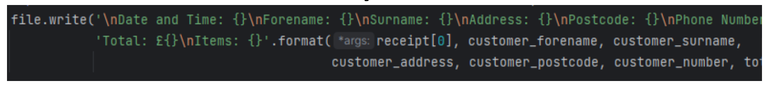
# Task 2 - Test Log

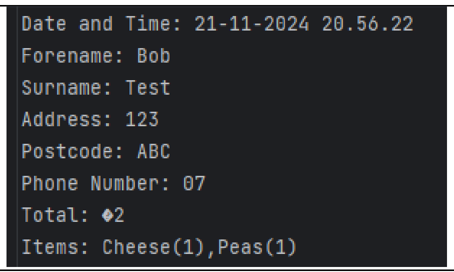
| Description of test | Test data to be used (if required) | Expected outcome | Actual outcome | Comments and intended actions |
|---|---|---|---|---|
| Attempt to run the program | | Program should run without error | Test 1:<br>Invalid SyntaxError line 18 when trying to close input<br><br>Test 2:<br>No module found 'dateime' line 1<br><br>Test 3:<br>Program ran without error | Test 1:<br>Added additional bracket to use correct syntax<br><br>`y(input("Please try again, number was not valid"))`<br><br>Test 2:<br>Fixed grammatical error when trying to identify invalid module name<br><br>`import datetime`<br><br>Test 3:<br>Program worked as expected |
| Testing Menu: Option 1 (Sell a product) | choice = 1<br>(Normal) | Program should accept the user input and should display a list of available products | Test 1:<br>Program did not accept the user input, and classed it as an invalid choice<br><br>`Enter your choice: 1`<br><br>`Invalid choice, please try again.`<br><br>Test 2:<br>Program accepted the user input, but did not display a list of available products<br><br>Test 3:<br>Program displayed list of available products | Test 1:<br>Adjusted logical operators to identify if the choice is higher than the number of available menu items, OR less than 0.<br><br>`if choice > upper_bound or choice < lower_bound: #Mistake`<br>I identified the variables by using an output, as a form of debugging, and identified the appropriate logic<br><br>Test 2:<br>Added output display to show the list of products<br><br>`def sell_product():`<br>`    items = []`<br>`    while True:`<br>`        print("Available products:\n {} \n".format(products))`<br>Test 3:<br>Program worked as expected |
| Testing Menu: Entering number out of range | choice = 3<br>(Erroneous) | Program should not accept the input, and should display an error message | Test 1:<br>Program does not accept the user input, and displays an appropriate error message | Test 1:<br>Program worked as expected |
| Testing Menu: Entering non-numerical input | choice = ABC<br>(Erroneous) | Program should not accept the input, and should display an error message | Test 1:<br>Program does not accept the user input, and displays an appropriate error message | Test 1:<br>Program worked as expected |
| Testing Menu: Entering number out of range when there are two menu selections | choice = 3<br>(Erroneous)<br><br>`menu = {`<br>`    "Sell a product": sell_product,`<br>`    "Test menu item": sell_product,`<br>`}` | Program should not accept the user input, and should display an error message | Test 1:<br>Program IndexError line 108 when trying to check for an item that does not exist<br><br>Test 2:<br>Program does not accept the user input and displays an error message | Test 1:<br>The program identifies the choice after subtracting 1, because when looking in a list the index is 0 at the start<br><br>Due to the index being subtracted, I added 1 to the choice to identify its original value (the user's input)<br><br>`if choice + 1 > upper_bound or choice < lower_bound:`<br><br>Test 2:<br>Program worked as expected |

| | | | | |
|---|---|---|---|---|
| **Testing Option 1 (Sell product): Choosing a product available from the list** | **item = Cheese<br>(Normal)** | **Program should accept the user input as a valid product and should ask the user to enter more choices** | **Test 1:<br>Program does "accept" the choice; however it breaks from the while loop<br><br>Test 2:<br>Program resorts to the else statement  correctly, however errors when trying to "lower" an integer**<br>`'int' object has no attribute 'lower'`<br><br>**Test 3:<br>Program identifies the product correctly, but errors when trying to identify the cast_input_to_int_safely() function**<br>`item_added.append(cast_inputto_int_safely(`<br><br>**Test 4:<br>Program validates the user input and asks for additional items to add to the basket** | **Test 1:<br>Adjusted the if statement as the program was checking if the input was not equals to "done", and would break out of the while loop (line 40)**<br>`if item == 'done':`<br>`    break`<br><br>**Test 2:<br>Program compares product name to product price when running get_product(name). I adjusted the index to 0 instead of 1, so it would fetch the product name as opposed to price**<br>`if product[0].lower() == name.lower():`<br><br>**Test 3:<br>Fixed the grammatical error when trying to call the cast_input_to_int_safely function**<br>`item_added.append(cast_input_to_int_safely(i`<br><br>**Test 4:<br>Program worked as expected** |
| **Testing Option 1 (Sell product): Entering an invalid product name** | **item = Fish<br>(Erroneous)** | **Program should not accept the user input, and should display an error message** | **Test 1:<br>Program does not accept the input, and displays an error message**<br>`Invalid product please type one of the following:` | **Test 1:<br>Program worked as expected** |
| **Testing Option 1 (Sell product): Does the program display the current basket correctly?** | **item = Cheese<br>(Normal)** | **Program should display the current basket in an informative way, showing the current product names and their quantity** | **Test 1:<br>Program displays the index and the quantity, but not the name of the product<br><br>Test 2:<br>Program displays the current basket with the product name and quantity**<br>`Current basket:Cheese (1)` | **Test 1:<br>Reformatted the current basket to identify the product name by the index 0 instead of 1**<br>`print("Current basket:" + ', '.join( ('{} ({})'.format( *args: i[0], i[2])`<br>**Test 2:<br>Program worked as expected** |
| **Testing Option 1 (Sell product): Does the program display the current basket correctly when having multiple products?** | **item = Cheese<br>item = Peas<br>(Normal)** | **Program should display the current basket in an informative way, showing the current product names and their quantity** | **Test 1:<br>Program displayed all of the products and the quantity in the current basket correctly**<br>`Current basket: Cheese (2), Peas (3)` | **Test 1:<br>Program worked as expected** |
| **Testing Option 1 (Sell product): Choosing multiple products from the list** | **item = Cheese<br>item = Peas<br>(Normal)** | **Program should allow you to add multiple items to the list by repeatedly asking for an input** | **Test 1:<br>Program allowed the user to add multiple items to the list, by asking the user to enter a choice, then a quantity, and repeats itself until done** | **Test 1:<br>Program worked as expected** |
| **Testing Option 1 (Sell product): Entering a quantity amount** | **inp = 1<br>(Normal)** | **Program should allow you to enter the valid quantity amount and should move onto the next step** | **Test 1:<br>Program accepts the valid quantity amount and moves onto the next step** | **Test 1:<br>Program worked as expected**<br>`Input how many of the item you would like to buy: 1`<br><br>`Added item!` |

| | | | | |
|---|---|---|---|---|
| **Testing Option 1 (Sell product): Entering a negative quantity amount** | **inp = -1<br>(Erroneous)** | **Program should not allow the user to enter a negative quantity amount and should display an error message** | **Test 1:<br>Program accepted the negative quantity input and did not display an error message<br><br>Test 2:<br>Program did not allow the user to input a negative quantity and displayed an error message** | **Test 1:<br>Added else condition to cast_input_to_int_safely() function, so it will check if the input is less than/equals to 0 and ask for a new input**<br><br>`try:`<br>`    if int(inp) > 0:`<br>`        return int(inp)`<br>`except:`<br>`    return cast_input_to_int_safely(input("Please try again, number was not valid: "))`<br>`else:`<br>`    return cast_input_to_int_safely(input("Please try again, number must be more than 0: "))`<br>**Test 2:<br>Program worked as expected**<br><br>`What item would you like to buy (type 'done' to continue to checkout): Peas`<br><br>`Input how many of the item you would like to buy: -1`<br>`Please try again, number must be more than 0: 2`<br><br>`Added item!`<br><br>**With this new change, the function will check for any inputs being processed, and identifies if they are more than 0, meaning valid. Adds additional verification** |
| **Testing Option 1 (Sell product): Entering a non-numerical quantity amount** | **inp = ABC<br>(Erroneous)** | **Program should not allow the user to enter a non-numerical quantity and should display an error message** | **Test 1:<br>Program did not allow the user to enter a non-numerical input and displayed an error message**<br><br>`Input how many of the item you would like to buy: ABC`<br>`Please try again, number was not valid: 1` | **Test 1:<br>Program worked as expected** |
| **Testing Option 1 (Sell product): Continuing straight to checkout with no products** | **item = done<br>(Normal)** | **Program should not allow the user to continue to checkout when there are no products in the items list** | **Test 1:<br>Program allowed the user to continue to checkout, despite having no products in checkout<br><br>Test 2:<br>Program did not allow the user to continue when there are no products in the items list**<br><br>`What item would you like to buy (type 'done' to continue to checkout): done`<br>`You must have something in the checkout to continue` | **Test 1:<br>Added an if statement to check if the items list is empty using PEP 8 appropriate standards, and outputted an error**<br><br>`if item == 'done':`<br>`    if not items:`<br>`        print("You must have something in the checkout to continue")`<br>`    else:`<br>`        break`<br><br>**Test 2:<br>Program worked as expected** |
| **Testing Option 1 (Sell product): Entering alternative user input to enter checkout** | **item = DONE<br>(Extreme)** | **Program should logically allow the user to continue as the input is similar to "done". Program should take the next step** | **Test 1:<br>Program assumes this input is a product, and gives an error message for this<br><br>Test 2:<br>Program allows the user to enter DONE, as an appropriate input and takes the user to the next step** | **Test 1:<br>Modified if statement to check if the item as .lower() is the same as done**<br><br>`if item.lower() == 'done':`<br>**Test 2:<br>Program worked as expected** |
| **Testing Option 1 (Sell product): Choosing a product that is not from the list** | **item = Bob<br>(Erroneous)** | **Program should not accept the user product and should display an error message** | **Test 1:<br>Program does not accept the product input, and successfully displays an error message**<br><br>`What item would you like to buy (type 'done' to continue to checkout): Bob`<br><br>`Invalid product please type one of the following: Cheese, Potatoes, Carrots, Peas,` | **Test 1:<br>Program worked as expected** |
| **Testing Option 1: Entering non-numerical phone number** | **customer_number = ABC<br>(Erroneous)** | **Program should not accept the input as a valid phone number** | **Test 1:<br>Program accepts the non-numerical phone number and moves onto the next step without error message** | **Test 1:<br>Added while loop to check if the input is not numerical and to display an error message** |

| | | | | |
|---|---|---|---|---|
| | | and should display an error message | **Test 2:**<br>Program did not accept the invalid phone number input, and now displays an error message when a non-numerical number is inputted<br><br>`What is the customers phone number: abc`<br>`You must enter a numerical phone number` | <br>`while True:`<br>`    customer_number = input("What is the customers phone number: ")`<br>`    if customer_number.isnumeric():`<br>`        customers.append([customer_forename, customer_surname, customer_address, customer_postcode`<br>`        break`<br>`    else:`<br>`        print("You must enter a numerical phone number")`<br>**Test 2:**<br>Program worked as expected |
| **Testing Option 1: Entering employee discount** | **employee_discount = Y (Normal)** | **Program should accept the user input and should move onto the next step** | **Test 1:**<br>Program accepted the employee discount input but errored on line 77 when trying to define total with total.<br><br>`77        total = total`<br><br>**Test 2:**<br>Program accepted the input and did not error when trying to identify line 77 error with the total.<br><br>**Test 3:**<br>Program did not error when trying to identify the forename variable | **Test 1:**<br>Defined total variable as the subtotal<br><br>`total = subtotal`<br><br>**Test 2:**<br>However, the program produced an error when trying to identify 'customerForename' as an undefined variable name. I fixed this error as the convention was incorrectly named<br><br>`An error occurred:`<br>`name 'customerForename' is not defined`<br><br>**Test 3:**<br>Program worked as expected |
| **Testing Option 1: Entering no employee discount** | **employee_discount = N (Normal)** | **Program should accept the user input and move onto the next step** | **Test 1:**<br>Program accepted the user input and moved onto the next step | **Test 1:**<br>Program worked as expected |
| **Testing Option 1: Entering invalid employee discount input** | **employee_discount = 2 (Erroneous)** | **Program should not accept the user input, and should display an error message** | **Test 1:**<br>Program accepts the user input and moves onto the next step without an error message<br><br>**Test 2:**<br>Program does not accept the user input and displays an error message<br><br>`Is the customer an employee (makes them legible for employee discount)? (Y/N) 2`<br>`You must enter an input of Y or N`<br>`Is the customer an employee (makes them legible for employee discount)? (Y/N)` | **Test 1:**<br>Added a while loop to check if the employee discount is not equal to any of the available options and asks the user to enter another input. Program displays an error message<br><br>`employee_discount = input("Is the customer an employee (makes them legible for employee discount)? (Y/N) ").lower(`<br><br>`while employee_discount != "y" and employee_discount != "n":`<br>`    print("You must enter an input of Y or N")`<br>`    employee_discount = input("Is the customer an employee (makes them legible for employee discount)? (Y/N) ").lo`<br><br>`employee_discount = employee_discount.lower() == "y"`<br><br>**Test 2:**<br>Program worked as expected |
| **Testing Option 1: Does the program output the correct customer details and receipt?** | **item = Cheese**<br>**inp = 2**<br><br>**customer_forename = Bob**<br>**customer_surname = Test**<br>**customer_address = 123**<br>**customer_postcode = ABC**<br>**customer_number = 07**<br>**(Normal)** | **The program should output correct customer details and an appropriate receipt. Program should not display any errors that interrupt the result or functionality of any logic** | **Test 1:**<br>Program successfully outputs the correct customer information, including the items bought (and cost), subtotal<br><br>Program error occurred when trying to open a txt file with incorrect date formats prior to being opened. The program has contradictory date formats<br><br>`e("%d.%m.%Y %H:%M:%S"),`<br>`code,`<br>`), str_items]`<br>`("%Y.%m.%d %H:%M:%S")`<br>**Test 2:**<br>Program identifies invalid argument again when trying to identify the text file | **Test 1:**<br>Updated the date format to follow the DD/MM/YYY format<br><br>`strftime("%d.%m.%Y %H:%M:%S")`<br>`ner_postcode,`<br>`str(total), str_items]`<br>`strftime("%d.%m.%Y %H:%M:%S")`<br><br>**Test 2:**<br>To mitigate this issue, I altered the format to not use colons. Files are not allowed to have colons in them, so I used the following format: DD-MM-YYYY HH.MM.SS<br><br>`receipt = [datetime.datetime.now().strftime("%d-%m-%Y %H.%M.%S"),`<br>`          customer_address, customer_postcode,`<br>`          customer_number, "£" + str(total), str_items]`<br>`with open(datetime.datetime.now().strftime("%d-%m-%Y %H.%M.%S") +` |

| | | | | |
|---|---|---|---|---|
| | | | **Test 3:**<br>**Program correctly outputs customer details, and an appropriate receipt. Program did not display any errors, however the spent over £25 discount was applied incorrectly, but also with an incorrect percent off**<br><br>`if subtotal <= 25:`<br><br>**Test 4:**<br>**Program outputs correct details and customer discount based on subtotal** | **Test 3:**<br>**Updated if statement to check if the subtotal is more than 25 and not less than/equals to. Updated discounts**<br><br>`if subtotal > 25:`<br><br>`discounts = {`<br>`    'staff': 5,`<br>`    'over25': 10,`<br>`}`<br>**Test 4:**<br>**Program worked as expected** |
| **Testing Option 1: Does the program calculate the correct employee discount** | **employee_discount = Y (Normal)** | **Program should calculate 5% off the total order for employees and should display the correct outputs** | **Test 1:**<br>**Program correctly calculated 5% off the total order for an employee and displayed the following**<br><br>`5 Cheese (TOTAL: £5.00)`<br>`Subtotal: £5.00`<br>`Employee discount: -5%`<br>`Total: £4.75` | **Test 1:**<br>**Program worked as expected** |
| **Testing Option 1: Does the program calculate the correct customer discount for orders more than £25** | | **Program should calculate 10% off the total order for customers that spend more than £25 and should display the outcome for final total** | **Test 1:**<br>**Program correctly calculated 10% off the total order, when the subtotal was higher than 25 and outputted the final total**<br><br>`1 Cheese (TOTAL: £1.00)`<br>`30 Potatoes (TOTAL: £30.00)`<br>`Subtotal: £31.00`<br>`Spend over £25 discount: -10%`<br>`Total: £27.90` | **Test 1:**<br>**Program worked as expected** |
| **Testing Option 1: Does the program give a discount to employees if the conditions for a customer discount are met?** | | **Program, optionally, can give both discounts and should calculate the result. The program should calculate the employee discount first and the over 25 discounts after** | **Test 1:**<br>**Program calculates the discounts correctly, with the employee discount being calculated first, followed by the over £25 discount**<br><br>`Subtotal: £35.00`<br>`Employee discount: -5%`<br>`Spend over £25 discount: -10%`<br>`Total: £29.93` | **Test 1:**<br>**Program worked as expected** |
| **Testing Option 1: Does the program write the receipt to a new file correctly?** | | **Program should create and append to a new text file with the receipt details in an informative way** | **Test 1:**<br>**Program successfully writes to a new text file with the receipt information, however it is not readable in its current format**<br><br>`21-11-2024 20.52.04`<br>`A`<br>`A`<br>`a`<br>`a`<br>`1`<br>`�2`<br>`Peas(2)`<br><br>**Test 2:**<br>**Program outputs the information in a more formatted, readable way** | **Test 1:**<br>**Changed file.write to make use of .format() to process the information in a readable way**<br><br>`file.write('\nDate and Time: {}\nForename: {}\nSurname: {}\nAddress: {}\nPostcode: {}\nPhone Numbe`<br>`          'Total: £{}\nItems: {}'.format( *args: receipt[0], customer_forename, customer_surname,`<br>`                                      customer_address, customer_postcode, customer_number, to`<br><br>**Test 2:**<br>**Program worked as expected** |

|  |  |  |  |  |
|---|---|---|---|---|
|  |  |  | <br>```<br>Date and Time: 21-11-2024 20.56.22<br>Forename: Bob<br>Surname: Test<br>Address: 123<br>Postcode: ABC<br>Phone Number: 07<br>Total: �2<br>Items: Cheese(1),Peas(1)<br>``` |  |