Task 2 - Test Log

Description of test	Test data to be used (if required)	Expected outcome	Actual outcome	Comments and intended actions
Testing to make sure that the menu items are listed with the prices next to them, and making sure they are stored that way.	Lists	Expected outcome would be that when the code is run, the list showing the food options and prices show.	The list of food items are shown in the code, and they are then able to be seen for the customer.	`#list is used so that the food options are in an order and easier to see.` `#also stored as a dictionary so they can be found and stored more easily.` `Wrong statement used to separate the food options, they used a comma – ‘,’ instead of a semi-colon ‘;’, by using the semi-colon you are then able to separate the foods in the list.`
Changing ‘flag = true’ to it being recognised as ‘flag’	‘True or False’ testing, this would be making sure that something is defined as ‘True or False’ for the code.	Expected outcome would be that it would be able to be read as ‘flag = True’ as then, when you then submit the code, it knows that ‘flag’ is defined.	When I then ran the code, after changing it to ‘flag = True’, it was then able to be recognised, because it had been defined.	By changing the code as ‘flag = true’ – you might have needed to ‘define true’ in the code, so by changing ‘flag = true’ to ‘flag = True’ , the code is then able to be defined and read. `#make sure to always use ‘True’ instead of ‘true’ so that something can always be defined.`
Making sure people can enter their ‘table number’	Input testing	This would be making sure that people can enter their table number, if it has an error the first time, they can then try again.	The outcome that happened is that, people that were unable to enter the table number the first time, they could then be able to try again for a second time.	By changing the code from int(tabl_num) to `input('Please enter your table number: '), you are then able to enter the table number again. This making it easier to input.`
Making sure that you are able to enter ‘menu items’ and then end entering when you have entered all the items you want to order.	Input testing	The expected outcome would have to be that you are able to enter the food times you want to order, and then you are then able to stop entering once you have added everything you want.	When I run the code, you are able to enter the ‘menu items’ you want, and then once you have entered all the items you want, you can then enter ‘x’ when you want to and you would have then submitted all you items.	The code is fine when it has: `menu_item = input('Enter menu item. Type x if there are no more items to enter: '), because you are able to enter all the items you want, without any trouble, and then once you have finished submitting, you can then input ‘x’` `Also, when you enter a false item, this message comes up: if menu_item.isalpha() == False:` `print("Sorry you have not entered a valid menu item.")` `flag = True` `That means that the item you have entered is false, whether that is a spelling mistake, or a false item, you will get flagged for it.`

Add more rows and tables as required

Description of test	Test data to be used (if required)	Expected outcome	Actual outcome	Comments and intended actions
Making sure the customer can only order 'whole number quantity's' for each item.	**Integer Testing**	**The expected outcome would have to be that the customer, when they want to order and have to select the quantity of each item, that they can only order 'whole numbers'. For example, the customer can't order '1.5 Burgers', they would only be able to order '1 or 2 burgers' – only whole numbers.**	**The outcome when I ran it, you was only able to put in 'whole numbers' for the amount of each item you ordered. When you put '1.5 Burgers' – it would automatically be an error, and you would have to re-enter the amount for each item.**	`#When you want to order something, it has to be a WHOLE NUMBER ONLY, it can't be 1.5 burgers you are ordering.` **I also changed the code when it said that: print("Quantity must be a positive number") – just to make it more easier for the customer to read, in-case they might not understand 'positive number' they might get confused. Just to make it easier for the person ordering.**
Letting the customer enter a discount if they are valid to one.	**Integer testing**	**The expected outcome would be that the person who is ordering, at the end, before they complete their order, are able to enter a discount code, if they have one.**	**When I run the code, they are able to enter a discount code, for which they are able to select what discount they are using, and then able to enter the code so that they get the discount.**	**You have to make sure you have an input so that they can type out what discount they will be using.** **And also making sure you have a 'False' error if they enter a incorrect discount option.**
Making sure that the percentage discount the customer gets when they enter their code.	**Integer Testing**	**Expected outcome would be that they can see what discount they get, when they select one. For example, if they have a 'staff discount' – it would show they get 25% off.**	**When I ran the code, the outcome was that it showed what discount they got when they entered their code, and how much.**	`You have to edit the part of the code when it says what discounts there are. Example: print(" 2. Staff discount") – this would just show a "staff discount" and not show their actual percentage discount they get. I edited it to: print(" 2. Staff discount – 25%") – so that this shows if they have a staff discount, they get 25% off. I also did the same for the 'Early Bird Discount' – for where they get 15% discount.`
Allowing the customer to enter their full order.	**Input testing**	**The expected outcome would be that they can enter the full details of their order – with the table number, who is being served, and what items they want.**	**When I ran the code, it allowed everyone to enter their item choice, to the quantity of it, to the cost of it. Then it gave them the final total.**	**You have to make sure that you have input coded into the code above it, and then below, make sure that you get the append for 'item choice', 'quantity' and 'cost', making sure all those come up, so that the whole list is there.**

Description of test	Test data to be used (if required)	Expected outcome	Actual outcome	Comments and intended actions
Having both totals show with/without discount.	**Float test**	**Expected outcome would be that BOTH total's show, so that they know what they are paying, with or without the discount. Also, so they know how much they save.**	**When I ran the code, it showed both total's, as it should, the total without the discount, and then with the discount.**	**Had to edit the 'elif' part of the code, and to also make it line with the rest of the 'IF' statements.**
The final test, to make sure it ALL works.	**Top Down, Bottom Up.**	**To have ALL the code working properly, with all inputs in there, to submit the order as well, to also get the total's with and without the discount.**	**The code works fine, there are NO ISSUES, it works completely fine, you get to input for your orders, as well as submit what table you are at. And then get the full total for everything. Perfectly run code.**	**Had to change quite a few bits of code within it, but the code works fine. Especially with the prints, inputs, IF statements.**