# 12DGT Python Documentation - Freddie’ Fast Fish

### Scenario

Freddy's Fast Fish wants to computerise their phone orders. Specifically, they want to be able to enter the Customer Details, Fish and Chip Orders, Cooked or Frozen, for Pick-up or Delivery options into a computer program and have it process and display the Delivery Details, Itemised Order, and Total Cost.

Phone orders generally consist of several kinds of Fish items.

There are 12 types of fish:  
6x $4.10 Fish: Shark, Flounder, Cod, Gurnet + 2 Extra Custom Fish  
6x $7.20 Fish: Snapper, Pink Salmon, Tuna, Smoked Marlin + 2 Extra Custom Fish

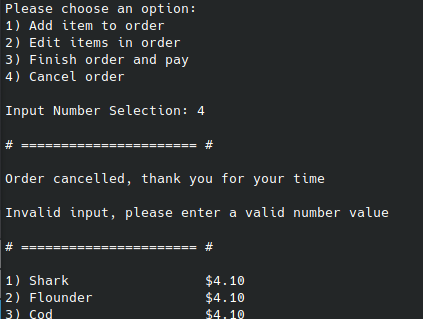
And they may also add chips (In my project they cost $3 per scoop)

+ Other details you can find in the brief document file.

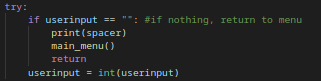
### Iteration

I have encountered multiple bugs during the coding process of this program, and have worked to fix each one. I have also changed things that I felt were not good enough for the ordering system so it worked perfectly as intended.

Cancelling Order not workingDuring my testing, I found that if you opened the add item menu, and decided you didn’t want anything more, so you didn’t input anything which returns you to the menu, and then decided to go ahead and cancel the order, it would completely ignore the exit() command and then return to the add item menu but in a loop of “Invalid Input” messages. I did a bit of testing and found that none of the exit() or quit() commands worked on it so I thought it was something wrong with another part of the code.



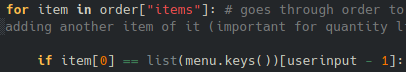
After a while of searching, I realised that the reason my code could be breaking is that it calls the return to the menu in a try/except statement, which could be why it’s returning to that part.



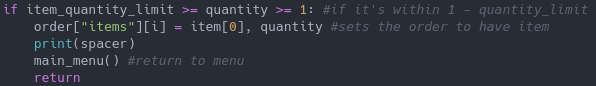
I changed it so it was outside of the loop, and it decided to work as intended.

#### Item limit did not apply when you added another of the same item

In my first iteration with the item limit, I had a problem that you could add another item of the same type which would ignore the limit (you could add 7, then another 7, then another 7 of the same item, ignoring the limit.) To fix this, it took me a while. I found that using a for loop to check the items in the order worked well to do this.



After the above code, I have the same get quantity code as the default except in this part instead of adding a new item of the same type, it edits the quantity of that item already in the order. You can see how I have done this below.



This instead of adding another item, edits the old one.

#### Frozen discount applies to non-fish items (chips)

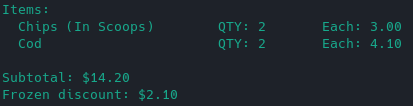
I have encountered a problem in which non-fish items (Such as chips) were having the frozen discount applied to them as well, when the brief stated that it should be applied for each fish item. There are multiple ways to do this, but the way I decided to fix this is to have a new list object which contained all of the non-fish items.



All you need to do is add the name of the item the same as it is spelt in the menu dictionary, and it will not apply the frozen discount to it. How I did this was with these four lines of code:

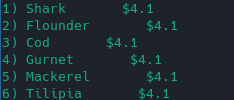


It used to count all of the items into the frozen discount, but with the first 2 lines above, you can see how it has been changed to make more sense.

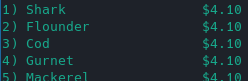


#### Items were not nicely aligned

I have had this problem where the items did not print in a nice way.



This is because there is a set number of spaces between the item and the cost. There are multiple ways to fix this. One of these ways is to add spaces to the end of each item individually so they are the right size. I did not do this, because it would not be efficient to the end users. The way I did this is to check the length of the item in characters using len(formatted\_item) and then subtracting that from 25 and multiplying the spaces by that result. I decided 25 was a good amount and that no items should be longer than that normally so it should be good.



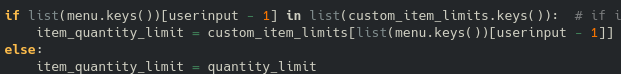
#### Custom Item Limits

I decided to add custom item limits if you wanted to change the item limit on individual items. This was because I wanted to give chips its own limit, so I decided to make it a useful feature that multiple things can have custom item limits if the user wants them.

I started by making a dictionary object, called custom\_item\_limits, and added chips and the custom quantity.

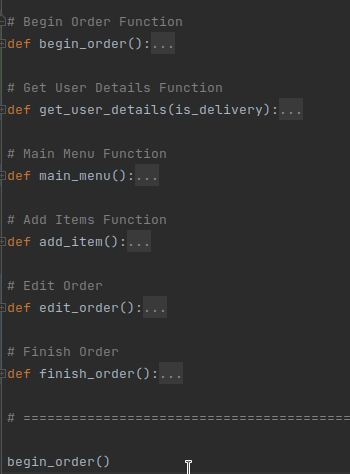


Now since I have this, I can go ahead and continue the coding for them. In this case, I just needed to change the add item quantity section



This does the trick, just before the checking if it’s in the limit.

#### Adding Restarting Sequence



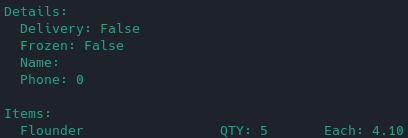
I had a few bugs when creating the restart order option for the program.

I had a few possible ways I could restart the order. A very easy way to do this would be to place the whole thing into a while loop. While this is functional, this is not the best method to do this, as you can use functions for everything and then call said functions multiple times. I changed my get user details code into a function and then changed the begin order code into a function so I could call the begin order function multiple times, restarting the order when called.

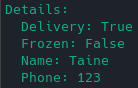
On the left you can see the compacted view of my code after making this change. Begin order contains all of the code required to start the order, like calling get\_user\_details() and then calling the main menu function. It also resets the order variable.

#### Fixing Restart Break

Restarting the order was breaking when restarting the order. It did not save the details of the new order, just the items.



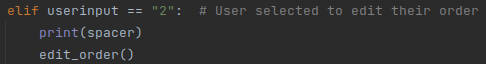
I fixed this by adding a line of code “global order” before resetting the order variable within the start order function. This is important because it sets the order variable to be a global variable.



This fixed the issue.

#### Editing Order Function

I decided that I would like to incoperate an “edit order” option into the menu which allows you to delete items from the order. This is particularly useful when a customer does not want a certain item anymore and so the order doesn’t have to be restarted. I added another menu choice:



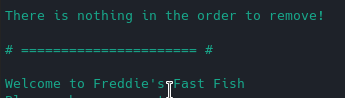
And then started the edit\_order() function. For the edit order function, I used a for loop which prints every item from the order in a numbered list. I would then get user input on which item they wanted to remove, and if said input was blank (“”) it would return to the menu. If it was not blank, it checks if it is an integer value, and then if it is valid it will remove it from the order.

#### Output Barely Readable / A mess

The output was a mess before I upgraded my code, as there was no whitespace in between the print statements. This one was quite an easy fix as I just created a spacer string which I could print which would nicely break up the walls of text.

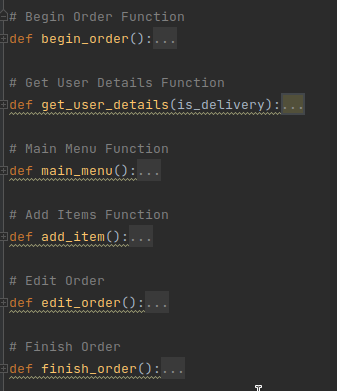
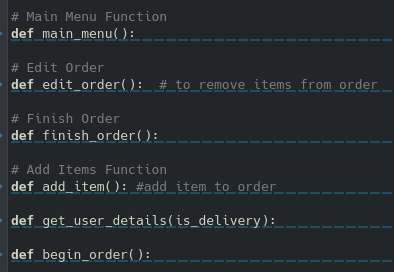


This worked nicely.



#### Code structure not making sense

This part of iteration is not for the end user, it is for anyone who decides to work on the code in the future. While I was developing this code, I placed the functions in the order I had implemented them. For reference, on the left is my original structure vs my updated structure of the code.



The right is in the order it would logically appear, based on where they are in the main menu, and the logical flow of the program. Before changing this, this was difficult to work with as I did not know where each part of the code was. This will help the next person to develop it and myself as I change it.

### Testing

I have created a spreadsheet in which I have tested each of my inputs to see if anything would break if inputted the incorrect values. It can be found in this project directory at ./Testing.xlsx