Project #1 – ATL Cookies

ATL Cookies (ATLC) is a bakery located close to GA Tech, known for late-night deliveries of hot cookies. A big part of its regular clientele are GA Tech students pulling all-night study sessions.

You will program an ordering system for ATLC. Please only address the following requirements.

Process

The following process is reflected in the sample dialog between example customers and the system (see the provided starter Notebook). You can therefore also review that dialog for additional clarity on any of the below requirements.

When a customer comes into the store, they are presented with a menu display (see Table 1):

Menu Items	Sugar Cookie	Chocolate Cookie	Peanut Cookie	Menu Item Price
Sugar Snack	6			6.0
Six Pack	2	2	2	12.0
Delicious Duo	3	3		9.0
Burdell Bundle	3	3	3	30.0

Table 1: **Menu Items Table** (showing components of each menu item and menu item price)

The **Menu Items Table** is generated using the Python module *prettytable*. Addendum 2 contains instructions on how to install this module in Anaconda.

The menu item information comes from the *menu* dictionary (see Figure 1). The menu prices shown in the table are calculated (as a function of menu item make-up and the respective cookie prices).

Once a customer has inspected the menu, they will be ready to place an order. For simplicity, we will assume a customer can place only a single order from the menu at a time (one order per interactive session).

As cookies are only baked once a day, there may be insufficient stock to fulfill an order. If this is the case, the order should not be taken, and the system will ask the customer to revise their initial order based on the available quantity. For example, if a customer wants to order 10 Six Packs and there are only enough cookies on hand to satisfy 5 Six Packs orders, the system will ask the customer to revise their order but limit the order quantity to 5 Six Packs. To calculate the level of available stock, refer to Addendum 1. A critical point is when HBC runs out of stock of a particular cookie – after that, they will no longer be able to take any orders containing that cookie type until the next day when a new batch of cookies is baked.

After an order has been accepted, the system will inform the customer of the total price and the order details will be recorded. Inspect the *OrderRec* entries in Figure 1 to see what information must be recorded.

Data Structures

Your program features the following initial data structures (see Figure 1):

```
# menu options in a nested dict
menu = {  'Delicious Duo' : {'Sugar' : 3, 'Chocolate' : 3, 'Peanut' : 0 },
              'Six Pack' : {'Sugar' : 2, 'Chocolate' : 2, 'Peanut' : 2 },
           'Sugar Snack' : {'Sugar' : 6, 'Chocolate' : 0, 'Peanut' : 0 },
        'Burdell Bundle' : {'Sugar' : 5, 'Chocolate' : 5, 'Peanut' : 5 } }

# individual prices for cookies in a dict
cookiePrices = {'Sugar' : 1.00, 'Chocolate' : 2.00, 'Peanut' : 3.00}

# current cookie inventory in a dict
cookieInv    = {'Sugar' : 50,   'Chocolate' : 35,   'Peanut' : 20}

# recorded past orders in a nested dict (in reality, would be in a DBMS)
'Legend: SS => Sugar Snack ; SP => Six Pack; DD => Delicious Duo; BB => Burdell Bundle'
'Content:  tranCode, orderQty, orderDateTime, tranCost'
orderRec = { 'SS1' : {5, '2022-02-26 13:52', 30.00 },
             'SS2' : {2, '2022-02-26 15:13', 12.00 },
             'DD1' : {3, '2022-02-26 15:32', 27.00 },
             'SP1' : {1, '2022-02-26 15:43', 12.00 },
             'DD2' : {6, '2022-02-26 16:03', 42.00 },
             'BB1' : {2, '2022-02-26 16:12', 30.00 },
             'DD3' : {4, '2022-02-26 16:14', 36.00 }
           }
```

Figure 1: Initial Data Structures

Data about the menu items (see Table 1) are stored in a nested dictionary (see Figure 1). When creating your application, you can assume that no new menu items will be added or deleted – but it is possible the composition of each menu item may be adjusted over time (but there is no need to include specific code to enact this).

When creating your application, you should assume the prices of cookies can change (but will not exceed $2.50). Cookies are not sold individually.

The current inventory of cookies is stored in a dictionary. You can make up the initial stock amounts (rather than use the ones shown in Figure 1), but stock amounts should not exceed 50 units of each type of cookie. As cookies are sold the stock amounts will be updated.

When an order is completed it will be recorded in the *orderRec* dictionary. The key to this dictionary is the *tranCode* field. You will notice the encoding of this field reflects the menu item ordered and also the transaction number (starting from 1). When adding a new record to the *orderRec* dictionary you will need to search the existing order transactions to work out the last recorded transaction number applicable to that menu item type (as the tranCode recorded for the current order must have the next number).

Program Design

The order-taking dialog between the system and customer – including displaying the initial menu – will be controlled by code outside any function (something akin to a main function – but there is no need to include an actual main function in your code because we have not emphasized this concept, and even without a main function, the Python interpreter will automatically execute all your source code, starting at the top line).

You must have the following functions (and arguments).

Function	Description
displayMenu()	Displays the table of menu items.
getMenuPrice(menuItem)	Calculates the total price for a specific menu item.
maxUnitsCanOrder(menuItem)	Returns the max number of units that can be ordered given current inventory. See Addendum 1.
decreaseInv(menuItem , orderQty)	Updates inventory to reflect sales of cookies
recordOrder(menuItem, orderQty, tranCost)	Record completed orders
processOrder(menuItem, orderQty)	Process completed orders: decrease inventory; calculate transaction cost; print price; record transaction. Coordinates other functions to minimize code redundancy.

Testing

You have been given four testing scenarios:

1. There are enough cookies in inventory to satisfy the order.
2. There are insufficient cookies in inventory to satisfy the order and the customer accepts the suggested revised order quantity.
3. There are insufficient cookies in inventory to satisfy the order and the customer rejects the suggested revised order quantity.
4. There are no cookies in inventory to satisfy the order and the order-taking process is terminated.

Generate output reflecting the testing scenarios by copying your code to another cell and executing your program reflecting each test scenario. Then delete the copies of your code to reduce the clutter (to help to grade). **So, please include the generated output of each of the 4 testing scenarios within your submission** – but excluding the statement: "*watermark- do not include this output statement in your testing code output*"

Other Requirements

- You cannot hard-code values that can be sourced from one of the provided data structures (i.e., do not include magic values), nor the order of elements within data structures. Exceptions are:
    - You can hard-code menu items and their respective quantities within the pretty table, but not the menu item price.
    - You can hard-code keys referring to cookie types (e.g., Sugar) in the following functions:
        - **getMenuPrice()**
        - **maxUnitsCanOrder()**
        - **decreaseInv()**
    - In **recordOrder()** you can hard-code the menu items to associate them with their respective transaction code header characters (e.g., DD).
- Include all your code within a single Jupyter cell.
- You cannot use the keyword **global**.
- Functions should be located after the data structures, and the order of functions should be the same as the order of functions listed in the function definition table.
- Include substantial program comments to assist a reviewer in quickly understanding your code.
- Do not capture possible exceptions using try/except statements. However, for **maxUnitsCanOrder()** you will need to include multiple try/except statements to capture the possible division-by-zero error (see.

Submission

Submit a Notebook with your code. At the top of the notebook add your name/s.

Final Comments

This assignment stresses procedural programming. A more 'scalable' design would likely also reflect: an object-oriented approach (do not declare any classes), use of databases to store persistent data structures, and the use of web technologies to provide an advanced interface to the application.

Addendum 1: 'adequate inventory' logic

An order cannot be taken if there is insufficient cookie inventory for any cookie type that is needed for a particular menu item. For example: if you have on-hand 100 units of both Sugar and Peanut cookies, but 0 units for Choc cookies, you will still not be able to see any Delicious Duo, Six Pack & Burdell Bundle menu items. An approach to implementation of this logic (reflecting various stock scenarios mocked up in Excel) is shown below. Note: things get a little more complex when there is no stock on hand for a particular cookie, or when a customer orders a menu item that does not contain all cookie types (to illustrate this, the cookie mix for Six Pack was assumed to change to include no chocolate cookies – see scenario #3 below).

		Sugar	Choc	Peanut
Assume=>	SixPack	2	2	2
Assume=>	Inventory	15	19	17
	[ ]	7.5	9.5	8.50
Min =>	7.5	7.5	9.5	8.50
Can offer=>	7			

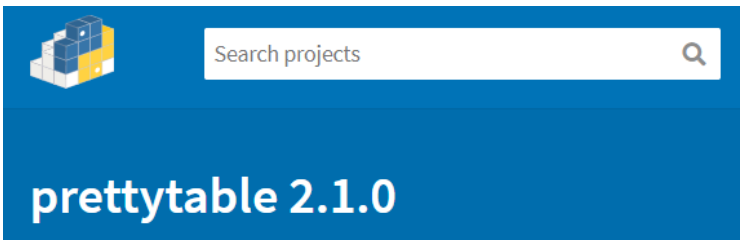
		Sugar	Choc	Peanut
Assume=>	SixPack	2	2	2
Assume=>	Inventory	10	19	17
	[ ]	5	9.5	8.50
Min =>	5.0	5	9.5	8.50
Can offer=>	5			

		Sugar	Choc	Peanut
Assume=>	SixPack	2	0	2
Assume=>	Inventory	10	19	17
	[ ]	5	#DIV/0!	8.50
Min =>	5.0	5		8.50
Can offer=>	5			

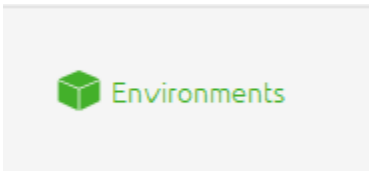
		Sugar	Choc	Peanut
Assume=>	SixPack	2	2	2
Assume=>	Inventory	10	0	17
	[ ]	5	0	8.50
Min =>	0.0	5	0	8.50
Can offer=>	0			

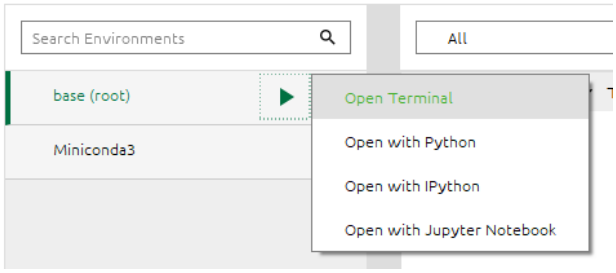
Addendum 2: prettytable

The prettytable module is not pre-installed in Anaconda.



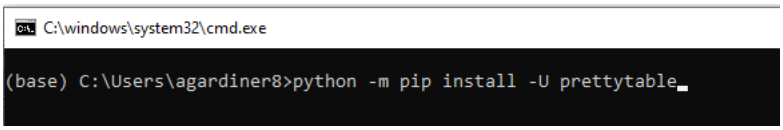
To install this module within Anaconda, open a terminal within Environments.





At the Terminal prompt, execute the following command:

> python -m pip install -U prettytable



Success looks like: