

**PYTHON PROGRAMMING**

(355)

**REGIONAL 2024**

**APPLICATION KNOWLEDGE:**

RegionalPython (470 points)

***TOTAL POINTS (470 points)***

**Test Time: 90 minutes**

**GENERAL GUIDELINES:**

*Failure to adhere to any of the following rules will result in disqualification:*

1. Member must hand in this test booklet and all printouts if any. Failure to do so will result in disqualification.
2. No equipment, supplies, or materials other than those specified for this event are allowed in the testing area. No previous BPA tests and/or sample tests (handwritten, photocopied, or keyed) are allowed in the testing area.
3. Electronic devices will be monitored according to ACT standards.

You will have ninety (90) minutes to complete your work.

Your name and/or school name should *not* appear on work you submit for grading.

1. Create a folder on the flash drive provided using your contestant number as the name of the folder.
2. Copy your entire solution/project into this folder. The project folder for you has already been provided: RegionalPython
3. Submit your entire solution/project so that the graders may open your project to review the source code.
4. Ensure that the files required to run your program are present and will execute on the flash drive provided.
5. You will need to use a local Python IDE to complete this exam. No online interpreters for Python are allowed.

\*Note that the flash drive letter may *not* be the same when the program is graded as it was when you created the program.

The graders will *not* compile or alter your source code to correct for this.   
Submissions that do *not* contain source code will *not* be graded.

**Assumptions to make when taking this assessment:**

* The goal of the program is to create an order entry form, and printout a list of all the data that was entered.
* One Python file is provided in the contest folder (PizzaOrder.py).
* The user will enter in the following information
  + The number of pizza items to create (range is 1 to 10 inclusive).
  + Three letter (or number) menu code.
  + Pizza name.
  + Total inventory to keep for that pizza.
* The price of the pizza item will be randomly generated in the Pizza object when it is created; the value should be in US dollars. The price should range from $10 to $20 inclusive
* The Total Inventory Cost is calculated in the pizza object (price \* inventory count); the value should be in US dollars.
* The output in console needs to be as close as possible to the example provided. NOTE: in the sample only one pizza is created due to limited space.
* The final listing should be vertical, with a space between each pizza item.

**Development Standards:**

* Your Code must use a consistent variable naming convention.
* All subroutines (if any), functions (if any), and methods (if any) must be documented with comments explaining the purpose of the method, the input parameters (if any), and the output (if any). Readability is a goal of good code.

**Commenting for Source Code Review (see the rubric):**

* Certain sections of your code will be graded. These gradable blocks of code can range from creating data structures, method algorithms, exception handling, and class construction.
* The grading rubric contains a section called Source Code Review: in this section is a list of descriptions of all the graded programming concepts.
* Each gradable item must have a comment placed at its beginning, and the comment must be prefixed with the comment flag. The flag helps the graders easily locate the code to increase the effectiveness of grading.
* The flag will always use this naming convention: **SC#** (NOTE: the # symbol will be replaced with sequential numbering, i.e. **SC1, SC2, SC3**, etc.
* Unless stated otherwise, NO explanation in the comment with the flag is required, only the comment flag; however, any information placed in the comment could help the grader better understand and avoid any costly errors.
* The comment flag needs to be placed in close proximity to the block of code it represents.
* If a comment flag is not present, you will not receive credit.
* In this example the Source Code Review has a gradable section of code for printing to the console (remember these are generic examples):
  + SC12: ***print*** method in the ***main*** classis printing the correct object \_\_\_\_ 10 pts
  + The user will place the code above the method call:

***#SC12 printing the car object***

***print(car)***

**RegionalPython**

In this test you will be creating an entry level order form for a startup frozen pizza company. The form will allow the employee to determine how many pizza types (i.e., cheese, pepperoni, veggie, etc,) to create. When creating pizza types, the employee will be required to create a three letter/number menu code, the pizza name, and how many units of the pizza type to keep in its freezer. This is a test software, so the program will randomly create a price ($10-$20) for the pizza. Once all the data entry requirements are completed, the program will print all of the newly created pizzas to the console.

**Input**

All input into the ordering software will be done via console entry. The data entry by the employee will cause the software to construct pizza objects in the code which will be stored in a simple data structure in the driver section. The input into the software will need to be regulated to ensure that the employee is not entering values greater or less than the required parameters, or improper values in the wrong areas; for example, when the program prompts for how many pizzas to create, the user should not be allowed to enter letters or symbols.

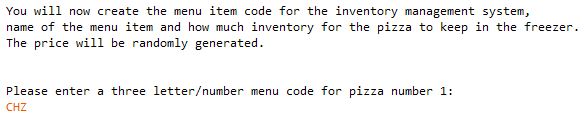
**Output**

There will be a system of coordinated output in conjunction with the data entry. Once the employee finishes entering all the required values, the program will automatically generate the final list of all created items.

**STEP 1 Item Quantality Entry**

Image shows STEP 1 Item Quantality Entry Program prompts user how many pizza types they want to create, after entry the program confirms the number.

**STEP 2 Three Letter Code Entry**

Program prompts user to create a three-character menu code.

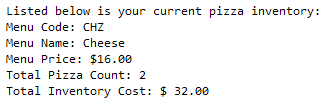
**STEP 3 Pizza Name Entry**

Image shows STEP 3 Pizza Name EntryProgram gives feedback from STEP 2 and asks user to enter in name of each pizza item. In this example there is only 1 pizza item.

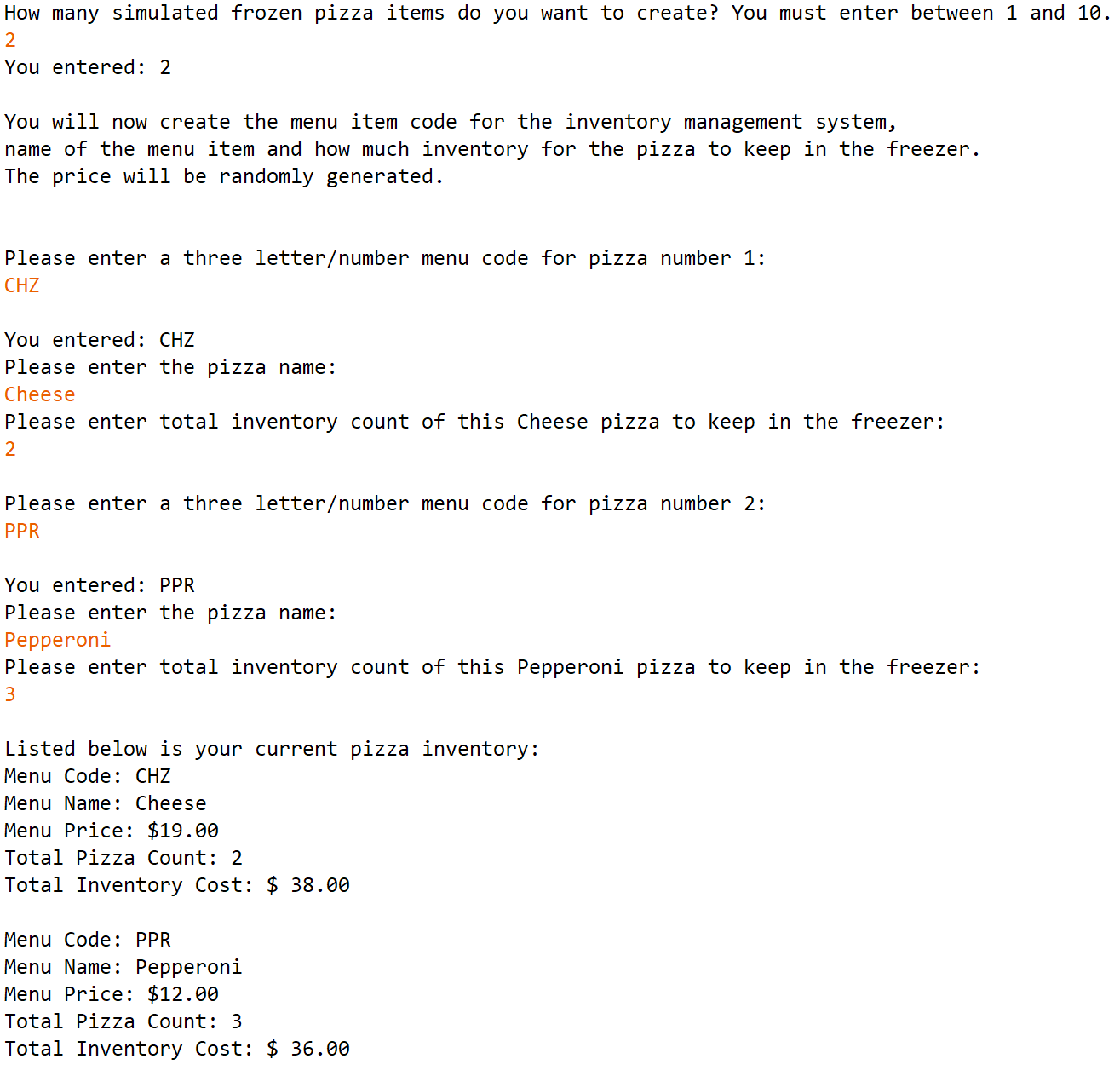
**STEP 4 Storage Quantity**

Program asks user to enter quantity to keep in the freezer.

**NOTE:** if the user chooses to make more than one pizza item; the program will repeat data entry sections of STEPS 2-4 until all data for the number of pizza items is created. In STEP 5 the program will print out all data.

**STEP 5 Output of Data Entry 1 Item**

Program gives feedback with all information entered by the user. A sample for a single item entry is provided with the image on the right.



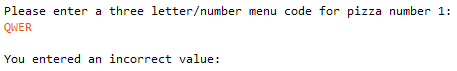
**STEP 5 Output of Data Entry 2 Items**

Program gives feedback with all information entered by the user. A sample for two item entries is provided with the image on the left.

**STEP 1 Data Value Error**

Image shows STEP 1 Range ViolationImage shows STEP 1 Data Value ErrorThe program detects if non whole numbers are entered.

**STEP 1 Range Violation:** The program detects if user enters values beyond 1-10 items.

**STEP 2 Range Violation:** user cannot enter in more than three characters for symbol.

**STEP 5 Data Value Error:** the program detects if non whole numbers are entered.

Image shows STEP 5 Data Value Error

**Requirements**

1. Your contestant number must appear as a comment at the top of the ***PizzaOrder.py*** source code file.
2. You will be programming all of the ***Pizzas*** class which can be created as object. In addition, you will be coding mainly in the section called the ***DRIVER SECTION***, which will run the user input gathering, create the pizza objects, and print the final list.
3. Code elements and methods in the ***Pizzas*** class:
   1. Program a constructor with the following parameters: MenuCode, MenuName, PizzaCount.
   2. The price of the pizza will be randomly generated. The price needs to have NO decimal value; however, the price will be formatted to US currency with two ‘0’ (zeros) in the cents section. For example, $24.00.
   3. The total inventory cost for the pizza needs to be formatted for US currency.
   4. ***getValue*** method: calculates the total inventory cost (price \* pizza count).
   5. ***toString*** method: returns a string of information about the pizza object in the required format at seen in the output example.
4. Code elements and methods in the ***DRIVER SECTION***:
   1. This section will be designated with the name in comments
   2. There are no required methods in this section. You can create methods if you choose; any methods you create will need to be commented.
   3. This section of the code will control the user input.
   4. Exception handling will need to be performed on only numerical entries by the user. The two numerical entries will be number of pizza types to create, and how many of each pizza type to store in the inventory.
   5. Entries for the pizza code need to check that the entry is exactly three.
   6. Entries for the pizza name has no parameters that need to be checked.
   7. Any entry errors need an error message and return the user back to the location where they need to enter in the proper amount.
   8. This section will also be required to print out all of the pizza objects that are created.

Your application will be graded on the following criteria:

**Solution and Project**

The project is present on the flash drive \_\_\_\_ 20 pts

The requiredfile is in one folder; the folder name is your contestant ID \_\_\_\_ 10 pts

**Program Execution**

Code copied to USB drive and the program runs from USB \_\_\_\_ 30 pts

*If the program does not execute, then the remaining items in this section receive a score of zero.*

STEP 1: Quantity of simulated pizza items question: accepts a positive whole number

entry, with feedback statement showing how much was entered \_\_\_\_ 10 pts

STEP 1: Quantity of simulated pizza items question: does not accepts out of

range entry gives feedback; forces user to reenter amount \_\_\_\_ 20 pts

STEP 1: Quantity of simulated pizza items question: does not accepts letters

gives feedback; forces user to reenter amount \_\_\_\_ 30 pts

STEP 2: Three letter menu code entry: accepts any three letters, number or

symbol and with feedback statement showing what was entered \_\_\_\_ 20 pts

STEP 2: Three letter menu code entry: does not accepts out of range entry

gives feedback; forces user to reenter amount \_\_\_\_ 20 pts

STEP 3: Pizza name entry: accepts any entry, and shows newly created pizza

name in the total inventory count question \_\_\_\_ 10 pts

STEP 4: Total inventory count: accepts a positive whole number \_\_\_\_ 10 pts

STEP 4: Total inventory count: does not accepts letters; forces user to

reenter amount \_\_\_\_ 30 pts

STEP 5: All pizza objects print with exact output as provided (no partial credit) \_\_\_\_ 30 pts

**Subtotal\_\_\_\_\_\_/240**

**Source Code Review**

*NOTE: you must place the comment flag in front of the comment in your code in order to get credit. The comment flag will precede the explanation. For example, if the flag is SC1, your comment must read as “#SC1…” in front of the part of the code being reviewed. Code must work to get credit.*

SC1: **Pizzas** constructor uses the three required parameters \_\_\_\_ 10 pts

SC2: **Pizzas** creates a random price between 10 and 20 (inclusive) \_\_\_\_ 10 pts

SC3: **Pizzas** method ***getValue*** returns correct calculated value of pizza items \_\_\_\_ 10 pts

SC4: **Pizzas** method ***toString*** returns string with correct object attributes \_\_\_\_ 10 pts

SC5: **Pizzas** price and total value formatted to US currency (no partial credit) \_\_\_\_ 20 pts

SC6: **DRIVER SECTION** uses ***try/except*** to catch entry error for pizza items \_\_\_\_ 40 pts

SC7: **DRIVER SECTION** code to check for rangeentry error for pizza items \_\_\_\_ 20 pts

SC8: **DRIVER SECTION** code to check for pizza symbol size \_\_\_\_ 20 pts

SC9: **DRIVER SECTION** code to enter in pizza name \_\_\_\_ 10 pts

SC10: **DRIVER SECTION** uses ***try/except*** to catch entry error inventory count \_\_\_\_ 40 pts

SC11: **DRIVER SECTION** construct pizza objects and place into data structure \_\_\_\_ 20 pts

SC12: **DRIVER SECTION** code to print all pizza objects from data structure \_\_\_\_ 20 pts

**Subtotal\_\_\_\_\_\_/230**

**Total Points: \_\_\_\_\_\_470pts**