

**PYTHON PROGRAMMING**

(355)

**STATE 2024**

**APPLICATION KNOWLEDGE:**

StatePython (460 points)

***TOTAL POINTS (460 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: StatePython
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.

\*It is recommended that you use relative paths rather than absolute paths to ensure that the program will run regardless of the flash drive letter. It is HIGHLY recommended that you place all of the files into one folder.

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:**

* Your program will be reading from a single text file that that has each field of the record listed line by line. Your program will read the txt file line by line and create an object based upon the field data provided. When the program reaches the word “STOP” the reading will end.
* The objects created by the text file will be stored in another object that will act as their manager.
* The goal of the program is to automatically read the txt file, create the objects, store them in the manager object, and the print the results to the console.
* Four files are provided in the contest folder: three Python and one txt.
* Txt File: the txt file for creating the objects has been provided (**PizzaText.txt**). The order of the fields is the following: Pizza Symbol, Pizza Name, Price, Active Inventory, Estimated Inventory in $’s.
* Python Files: there are three Python files in your contest folder. **Pizzas** (objects to be created), **PizzaMenu** (object that will create and store the pizzas), and **PizzaOrderControl** (reads the txt file, and creates the **PizzaMenu** object).
* The output formatting in console needs to be as close as possible to the example provided; however, your console output data from the reading the txt file and the user input and output must be exact.

**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):**

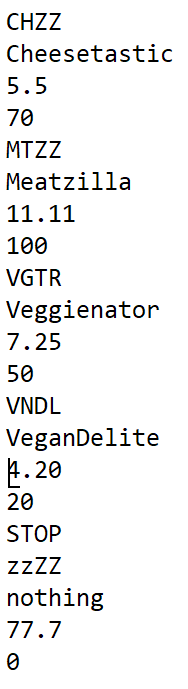
* Certain sections of your code will be graded. These gradable blocks of code can range from creating data structures, method algorithms, exception handling, and class construction.
* The grading rubric contains a section called Source Code Review: in this section are listed a description all of the graded programming concepts.
* Each gradable item must have a comment listed 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.
* 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 place 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 non-related 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)

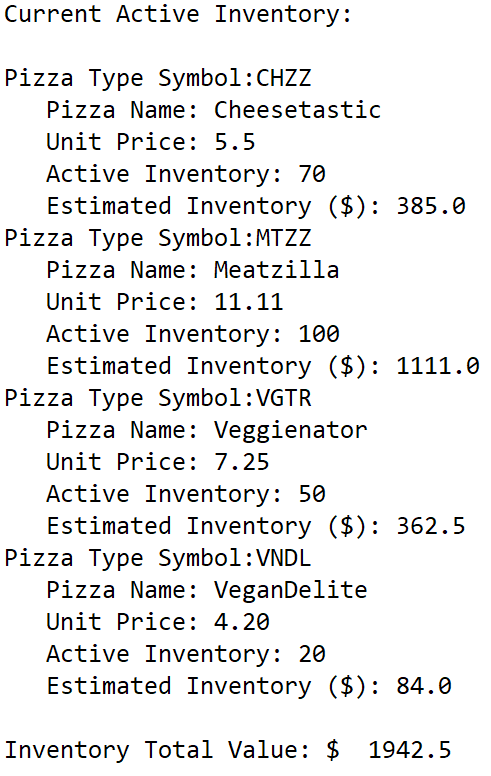
**StatePython**

In this test you will be creating a file reading program for a pizza restaurant to be used in conjunction with its store operations software that could be tied to all of its other software systems for the company. The program you will create has one goal: read the txt file immediately to create the different types of pizzas and store those objects in a central object.



**Input**

The only input into the program will be the reading of the txt file. There is no user input. An example of the input from the txt file is provided to the left

**Output**

The output from the code will be a single print job to the console. An example of the exact output from your program is provided (see the image to the right). You are required to create this print format in the Pizzas.py file. Your job will be to read the file and get the proper values for the record into the correct order and to get the output to print to console where the pattern uses a hanging indent to separate the records for readability.

**Missing File Error**

If the file is missing from the folder, output from the code should print the OS error message and the error object, following by printing the exit object, and exit the program.

Image shows Missing File Error

**Requirements**

1. Your contestant number must appear as a comment at the top of the **PizzaOrderControl.py** source code file.
2. You will be creating all of your code in the provided Python files. The only other file in the folder will be the **PizzasText.txt** file which should be left alone.
3. You will be programming all three Python files. The files have basic elements included. The files cannot be combined into a single file.
4. Code elements and methods in the **PizzaOrderControl.py** class:
   1. Program this class to read the PizzasText.txt file and store the information as Pizza objects stored into a PizzaMenu object.
   2. Print the entire inventory to the console.
   3. If the file is not found: The code must use exception handling, print out the error object, print out the exit commands, and properly exit the program.
5. Code elements and methods in the helper class **PizzaMenu.py** class:
   1. Program this class to be constructed as an object that can store Pizza objects
   2. ***createPizza*** method will create Pizza objects with appropriate parameters and add it to a data structure.
   3. ***getInventoryTVL*** method gets the total value of all the pizza types stored in the data structure. (NOTE: the actual calculation will be performed in the pizza object (price \* inventory)). This method will return a calculated value used in the final output printed to console.
   4. ***\_\_str\_\_*** method: will return the completed required output to **PizzaOrderControl**. HINT: you might consider traversing your data structure to get all of the desired output for each pizza object in this location.
6. Code elements and methods in the **Pizzas.py** class:
   1. Create the constructor using the supplied variables.
   2. ***getValue*** method: returns a value calculated from the pizza price and inventory count for that item.
   3. ***\_\_str\_\_*** method: returns the desired output.
   4. ***printPizzaType*** method: returns the printed string from the ***\_\_str\_\_*** method. This method will be called from

Your application will be graded on the following criteria:

**Solution and Project**

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

The fourrequiredfiles are all in one folder \_\_\_\_ 10 pts

**Program Execution**

The **PizzaOrderController** class runs and reads the input file \_\_\_\_ 30 pts

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

Program displays the OS error err object message if the file cannot be found \_\_\_\_ 20 pts

Program displays the exit command message and exit the program if the file

cannot be found \_\_\_\_ 20 pts

Output format matches sample \_\_\_\_ 30 pts

The program quits reading from the text file at the word “STOP” from

txt file and does not print extra symbols \_\_\_\_ 30 pts

Estimated Inventory value matches the output for each item (no partial credit) \_\_\_\_ 30 pts

Inventory Total Value matches the desired output \_\_\_\_ 30 pts

**Subtotal\_\_\_\_\_\_/210**

**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: **PizzaOrderController** imported proper class to construct correct object \_\_\_\_ 10 pts

SC2: **PizzaOrderController** constructs the PizzaMenu object \_\_\_\_ 10 pts

SC3: **PizzaOrderController** code to read the txt file line by line with

proper variable assignment \_\_\_\_ 20 pts

SC4: **PizzaOrderController** uses ***try/except*** to catch missing file and prints

the exception object message \_\_\_\_ 30 pts

SC5: **PizzaOrderController** prints out the exit command message and exits \_\_\_\_ 20 pts

SC6: **PizzaMenu** constructor with data structure attribute \_\_\_\_ 10 pts

SC7: **PizzaMenu** method ***createPizza*** creates Pizzas objects and adds

to data structure designed to hold Pizzas objects \_\_\_\_ 20 pts

SC8: **PizzaMenu** method ***getInventoryTVL*** gets the sum of all the

Estimated Inventory values of the pizza objects \_\_\_\_ 20 pts

SC9: **PizzaMenu** method ***\_\_str\_\_*** gets the all the string output of the

pizza objects by using their ***printPizzaType*** method \_\_\_\_ 20 pts

SC10: ***PizzaMenu*** method ***\_\_str\_\_*** prints the total value of inventory \_\_\_\_ 20 pts

SC11: ***PizzaMenu*** method ***\_\_str\_\_*** returns the complete output of the list \_\_\_\_ 20 pts

SC12: ***Pizzas*** constructor assigns proper values to instance variables \_\_\_\_ 10 pts

SC13: ***Pizzas*** method ***\_\_str\_\_*** completes output of the object and returns \_\_\_\_ 20 pts

SC14: ***Pizzas*** method ***getValue*** calculates and returns the inventory value \_\_\_\_ 10 pts

SC15: ***Pizzas*** method ***printPizzaType*** calls the \_\_str\_\_ method and returns

the proper output for the \_\_\_\_ 10 pts

**Subtotal\_\_\_\_\_\_/250**

**Total Points**: **\_\_\_\_\_\_\_/ 460pts**