

Saglara Orellana

May 31, 2022

Foundation of Programming: Python

Assignment 07

Rep: <https://github.com/saglarao/IntroToProg-Python-Mod07>

Page: <https://saglarao.github.io/IntroToProg-Python-Mod07/>

# Pickling and Error Handling

## Introduction

In this assignment I have written a script, that allows user to enter grocery items and its prices. Once grocery list is complete, the user can save it into a file. Once file is saved, the program can load it back again to display. I have used pickle module to create a file with grocery items in binary format and to display its contents. The script have error handling included to catch invalid user input.

## Starting the Grocery List

When the program starts running, I want it to display menu options to the user to choose from. I created a custom function called *show\_menu()*:

```
def show_menu():  
    """ Displays Menu Option to the User  
    :return: nothing  
    """  
    print('''  
        Menu of Options  
        1) Add a new grocery item  
        2) Show current list  
        3) Save Data to File  
        4) Reload Data from File  
        5) Exit Program  
        ''')  
    print() # Add an extra line for looks
```

Then I let the user to enter their menu option and capture it as *strMenuChoice* variable. Menu option 1 is to add new grocery item to the list. I assign variables *strItem* and *fltPrice* to value of custom function *input\_item\_price()*, that collects user input for new data. I want prices to be in numeric values, so I converted *fltPrice* from string input value to float type, when function was defined:

```
def input_item_price():
    """ Gets new grocery item and price

    :return: (string) of item name
    :return: (float) of item price
    """
    item = str(input('Please enter a grocery item: ')).strip()
    price = float(input('Please enter its price: ').strip())

    return item, price
```

To catch the user entering wrong data, I added try-except construct. I want prices to be entered as numbers and to not be zero or negative values. I created a custom class error *PriceError* in Exception base class with a custom message:

```
# Custom Exception Class
class PriceError(Exception):
    """Price must be valid: no negative or zero values """
    def __str__(self):
        return 'You have to pay for things!'
```

Now I can raise it in my main script body. After user input is collected for *strItem* and *strPrice*, I added the *if* statement for when price is zero or negative to raise the above created custom Exception class *PriceError*. Then the *try* block of code ends with custom function *add\_item(strItem, fltPrice, lstGroceries)*, that adds new data to the list called *lstGroceries*.

I added *ValueError* class in the first *except* statement for when user enters price as a word to show a message that they need to use numbers, as well as the *Exception* class.

```
if strMenuChoice == '1': # Adding new item to grocery list
```

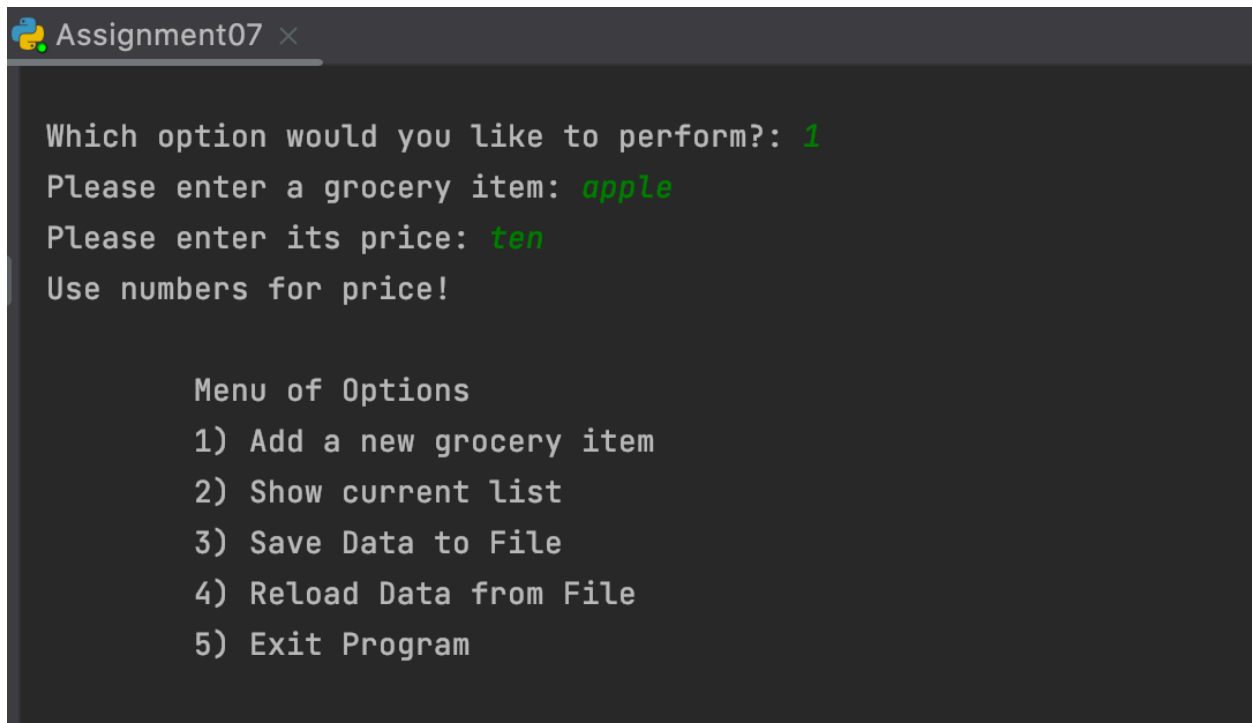
```

try:
    strItem, fltPrice = input_item_price()
    if fltPrice <= 0.0:                                # custom error condition
        raise PriceError()                            # raising custom error
    lstGroceries = add_item(strItem, fltPrice, lstGroceries)

except ValueError as e:                               # error handling if price entered as string
    print('Use numbers for price!')
except Exception as e:                               # handling errors in Exception class
    print('Invalid price value')
    print(e, e.__doc__, type(e), sep='\n')

```

I tested the program to add 'apple' as item and 'ten' as its price and received a message showing to use numbers for price, as it recognized error as ValueError type, and it proceeds to display menu options. Figure 01 shows the results.



```

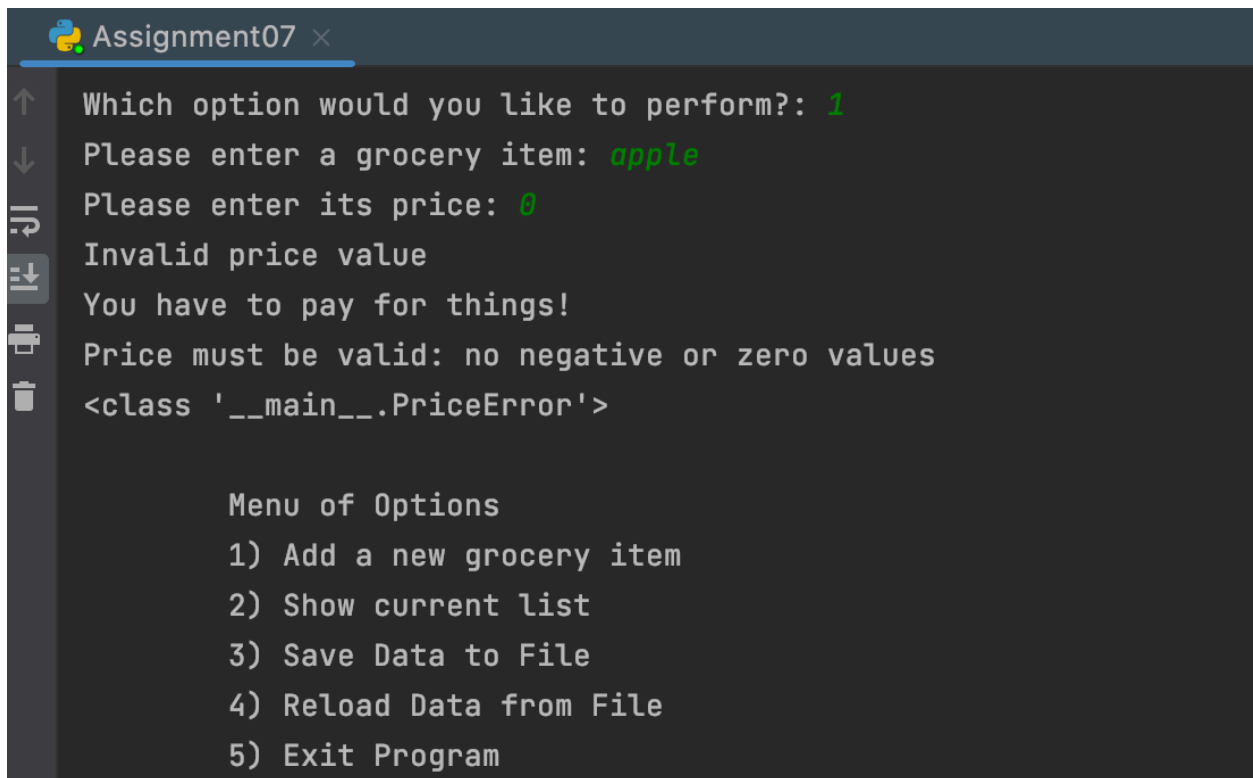
Assignment07 x
Which option would you like to perform?: 1
Please enter a grocery item: apple
Please enter its price: ten
Use numbers for price!

Menu of Options
1) Add a new grocery item
2) Show current list
3) Save Data to File
4) Reload Data from File
5) Exit Program

```

**Figure 01. Custom message for ValueError**

Then I want to see what the user will get, if they enter price as '0', Figure 02 shows the result.



```
Assignment07 x
Which option would you like to perform?: 1
Please enter a grocery item: apple
Please enter its price: 0
Invalid price value
You have to pay for things!
Price must be valid: no negative or zero values
<class '__main__.PriceError'>

Menu of Options
1) Add a new grocery item
2) Show current list
3) Save Data to File
4) Reload Data from File
5) Exit Program
```

**Figure 02. PriceError**

Figure 02 shows, that when the value is '0' (it will work same way, if I enter a negative number), the user gets print message ("Invalid price value") from Exception class, then it shows my custom message, doc string and its type from derived class PriceError that I defined.

Next option on the menu is to show current data, I created custom function *show\_list(list)*, that shows the data that was entered so far. I unpacked the list and print its values separating them with hyphen:

```
def show_list(list):
    """ Shows Current Grocery List
    :param list: (list) of items
    :return: nothing
    """
    print("\n***** The Current Grocery List: *****")
    for row in list:
        print(row[0], row[1], sep=' - ')
```

```
print("*****")
```

The main body of the script looks like this:

```
elif strMenuChoice == '2': # Showing current list
    show_list(lstGroceries)
```

## Saving Data to a Binary File

Menu option 3 lets user to save the data into a binary file. I have created a custom function *save\_file(file\_name, list)* to perform this task. First, I import *pickle* module inside of the function, then I open/create the file to write data and save the data to it by using *pickle.dump()* method. The return is status message, that the data was saved.

```
def save_file(file_name, list):
    """ Saves Grocery List into a binary file

    :param file_name: (string) of file name
    :param list: (list) of data to save
    :return: string
    """
    import pickle # imports pickle module

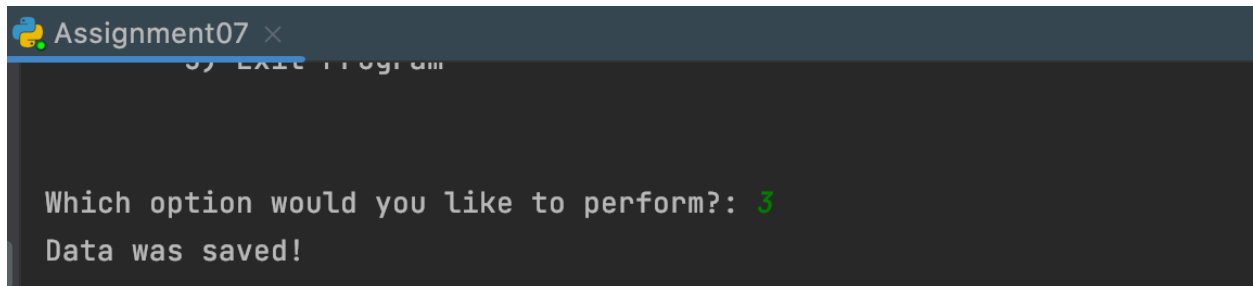
    with open(file_name, 'wb') as file: # automatically closes file when
done
        pickle.dump(list, file) # pickling data to a file

    return 'Data was saved!'
```

In the main body of the script I assign variable *strStatus* to the string value that is returned from function *save\_file*, then display *strStatus* to the user:

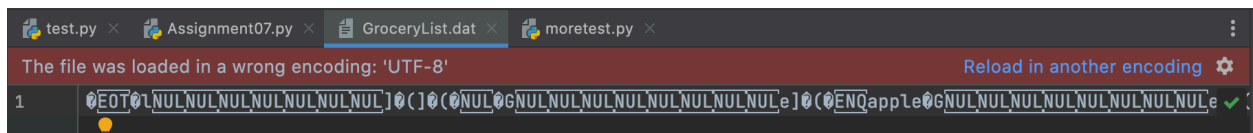
```
elif strMenuChoice == '3': # Saving data to binary file/ pickling
    strStatus = save_file(fileName, lstGroceries)
    print(strStatus)
```

Once I added couple grocery items to the list, I choose option 3 to save it into file and receive a message, that data was saved.



**Figure 03. Saving data**

New file “GroceryList.dat” appears in the project folder. When I open it, it hold binary type data, which I can’t really read.



**Figure 04. GroceryList.dat with binary data.**

## Reading Binary Data From File

Now that my data is saved, I can reload it back from file to display in the program. I have created a custom function `load_file(file_name)`. To run this function, I imported `pickle` module, then opened the file to read and used `pickle.load()` to extract the data, returning the list of data as a result.

```
def load_file(file_name):  
    """ Reads binary file  
    :param file_name: (string) of file name  
    :return: (list) of stored file data  
    """  
  
    import pickle    # imports pickle module  
  
    with open(file_name, 'rb') as file:    #automatically closes file when done  
        list = pickle.load(file)          #unpickling  
  
    return (list)
```

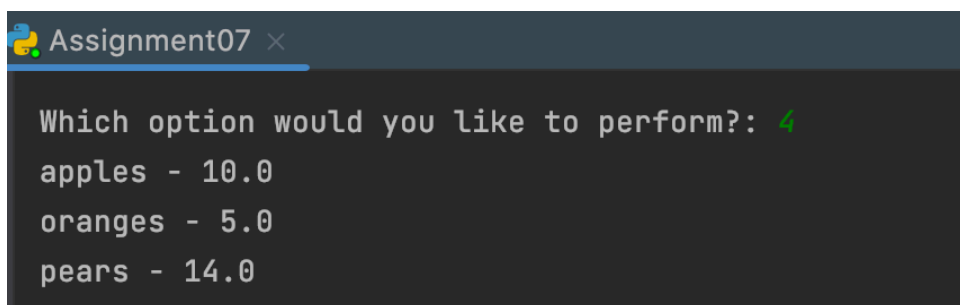
In the main body of the script I want to catch the error for when the user tries to choose menu option 4 to load file data before file is created. I used try-except and *FileNotFoundError* to capture the error and printed my custom messages: one is to say, that file does not exist and the other is to define a header for the following default class messages. I have displayed error messages differently from earlier, just to show different ways, I would most likely be more consistent otherwise.

```
elif strMenuChoice == '4':      # Unpickling file in load_file

    try:
        fileData = load_file(fileName)
        for row in fileData:
            print(row[0], row[1], sep=' - ')

    except FileNotFoundError as e: #error handling file does not exist
        print('File does not exist! Please enter data and save first\n')
        print('Built in Python error info:')
        print(e, e.__doc__, sep='\n')
        print('Python error type:', type(e))
```

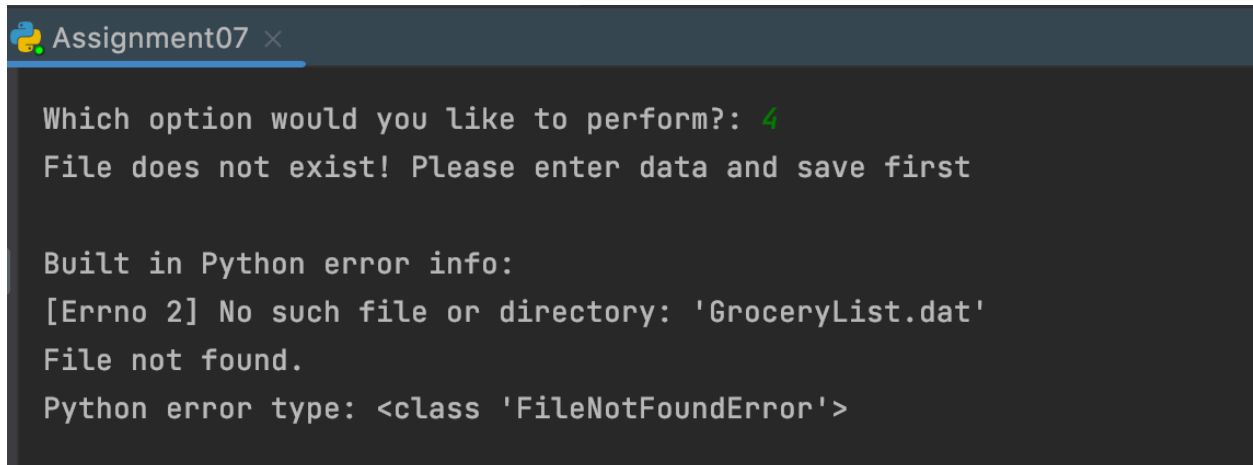
Since I already had file saved, I tested the unpickling the data first to make sure, it loads and displays correctly. Figure 05 shows the results.



```
Assignment07 x
Which option would you like to perform?: 4
apples - 10.0
oranges - 5.0
pears - 14.0
```

**Figure 05. Loading data from file.**

Next I deleted the file to see an error message. Figure 06 shows what program does, if the file doesn't exist and I try to choose option 4 of the menu.

A screenshot of a terminal window with a dark background. The title bar at the top shows a Python logo icon followed by the text 'Assignment07' and a close button icon. The terminal displays the following text: 'Which option would you like to perform?: 4' (where '4' is green), 'File does not exist! Please enter data and save first', a blank line, 'Built in Python error info:', '[Errno 2] No such file or directory: 'GroceryList.dat'', 'File not found.', and 'Python error type: <class 'FileNotFoundError'>'.

```
Assignment07 ×  
Which option would you like to perform?: 4  
File does not exist! Please enter data and save first  
  
Built in Python error info:  
[Errno 2] No such file or directory: 'GroceryList.dat'  
File not found.  
Python error type: <class 'FileNotFoundError'>
```

**Figure 06. *FileNotFoundError***

## Summary

The script lets the user to add items to the grocery list, then saves it as a binary data into a file, that can be later extracted back to display by using pickle module. While collecting user input, I added custom and regular error handling to catch incorrect value, display more user friendly message, as well as have the program to continue to run instead of shutting down.