

---

Mariia Kaminskaia

March 1, 2023

IT FDN 110 A Wi 23: Foundations Of Programming: Python

Assignment\_07

Link to GitHub: <https://github.com/moriia/IntroToProg-Python-Mod07>

Link to GitHub webpage: <https://moriia.github.io/IntroToProg-Python-Mod07/>

# Pickling and Exception Handling in Python

## Introduction

In this assignment, I had to do research about pickling and Exception handling in Python and create a script to demonstrate the acquired knowledge. This document summarizes the steps I performed to complete the assignment.

## Creating a script

### Binary files - general

As a part of this assignment, I had to learn and work with a binary file. Binary files are used to store data in the form of bytes, and the method of reading this file is not defined. This means that the program trying to read a binary file needs to be told how to read it. When trying to open a binary file using a normal text editor, it will be shown with unknown or unreadable characters. This is because the text editor assumes the data in text files to be encoded as text. Since the file is not encoded as text, it can not be read by the text editor (source: <https://careerkarma.com/blog/what-is-binary-file/>). In general, all files can be classified into two major formats — text and binary.

As a part of this assignment, I had to create a script that will write and read data into a binary file.

### Pickling in Python

“Pickling” is the process whereby a Python object hierarchy is converted into a byte stream, and “unpickling” is the inverse operation, whereby a byte stream (from a binary file or bytes-like

object) is converted back into an object hierarchy (source: [pickle — Python object serialization.](#)) There is a built-in pickle library in python (“pickle”), to use it firstly it should be imported (**Figure 1**).

```
7  # import ----- #
8  import pickle
9
```

**Figure 1.** Importing a pickle library.

The methods in the library can serialize a Python object into a flat byte stream, aka character stream (pickling), and transform a byte stream back into a Python object (unpickling). The character stream contains all the information necessary to reconstruct the object in another python script. Pickle is a library specific for Python use, meaning it can't be used to exchange data between applications written in different languages.

In addition to the Module 07 course video and book, the video [WHAT Is "Pickle" In Python](#) really helped me to start understanding the topic. It is a fast and clear explanation of what pickling is, and what two common methods “dump” and “load” do. After watching it I read python documentation [pickle — Python object serialization](#) with a more technical explanation of how the process works.

## Beginning of the script

As an initial step of creating a script, at the beginning of the script, I added a header about what the script is about and added a change log. Next step, I imported the built-in library “pickle” to work as a binary file, as I mentioned above. And in the last preparation step, I declared the variable I use in the script (**Figure 2**).

```
1  # ----- #
2  # Title: Assignment 07
3  # Description: Pickling and Exception handling in Python
4  # ChangeLog (Who,When,What):
5  # maria, 03.01.2023, Modified code to complete assignment 07
6  # ----- #
7  # import ----- #
8  import pickle
9
10 # Declare variables and constants ----- #
11 file_name = "pickle.dat"
12 paycheck = {} # A dictionary with 2 keys "person_name" and "salary_dollars"
13 paychecks_lst = [] # A list that acts as a 'table' of rows
14 choice_str = "" # Captures the user option selection
```

**Figure 2.** Beginning of the script file with a header, imports, and variable declaration.

---

### Main body of the script

Let's take a look at the main body of the script I wrote (Figure 3). The script is supposed to load information about paychecks (a list of items, where each item contains person\_name and person's salary). The data is stored in the file "pickle.dat", which will be unpickled and data will be displayed to the user. After it, the user will see a menu of options, and he/she will be asked which action the user wants to perform. As soon as the user choice is received, the related block of code will be executed.

```
150     print("\n")
151     print("About to load data exiting in the file ...")
152     paychecks_lst = load_data_from_the_file_and_display_it(paychecks_lst)
153
154     while True:
155
156         print_main_menu()
157         choice_str = get_user_choice()
158
159         if choice_str.lower().strip() == "1":
160             paychecks_lst = add_new_row(lst=paychecks_lst)
161             continue
162
163         elif choice_str.lower().strip() == "2":
164             edit_person_salary(lst=paychecks_lst)
165             continue
166
167         elif choice_str.lower().strip() == "3":
168             display_data(lst=paychecks_lst)
169             continue
170
171         elif choice_str.lower().strip() == "4":
172             save_data_to_the_file(lst=paychecks_lst)
173             continue
174
175         elif choice_str.lower().strip() == "5":
176             exit_program()
177
178         else:
179             try:
180                 raise ValidationErrorOfUserChoice(user_choice=choice_str)
181             except ValidationErrorOfUserChoice as e:
182                 print("Caught the exception " + repr(e) + e.message)
183             continue
```

**Figure 3.** Main Body of Script.

---

As you can see in the **Figure 3**, at the beginning of the script I load data from the file and assign the data existing in the list to the variable 'paycheck\_lst' (list of items declared at the beginning of the script). As soon as data is displayed, I display to the user a menu of options (**Figure 4**) that he/she can perform using the program. After the user sees the options, he/she is supposed to enter a value according to the existing option.

```
107 def print_main_menu():
108     print('
109         Please choose one of the following options
110         1. Add a new paycheck
111         2. Edit existing person salary
112         3. Display the data
113         4. Save data to the file
114         5. Exit program
115     ')
```

**Figure 4.** The function 'print\_main\_menu' displays options the user can perform using the program.

The options existed in the program are:

- 1. Add a new paycheck
- 2. Edit the existing person's salary
- 3. Display the data
- 4. Save data to the file
- 5. Exit program

If the user chooses option 1, he/she can add a new person and the person's salary to the list.

If the user chooses option 2, he/she can edit the existing person's salary.

If the user chooses option 3, he/she can see paychecks exist in the list.

If the user chooses option 4, he/she can save data to the file.

If the user chooses option 5, he/she exits the program.

So after a fast overview of the main body of the script, let's take a look at each function separately.

### Load data and display it from the file

As a first step of creating a script, I created a function 'load\_data\_from\_the\_file\_and\_display\_it' (**Figure 5**). I used the 'with' statement with the open() function to open a file (in my case - a binary file). Since I want to read a binary file, I have to tell the open() function the mode I want the file to

be opened. For this purpose as an argument in the `open()` function, we can view 'rb', where r - stands for reading, and 'b' - stands for binary, otherwise, without adding 'b' the function assumes I ask to open a text file.

In the end, I do not need to close the file by myself, since the 'with' statement closes the file without telling it. This is because the 'with' statement calls 2 built-in methods behind the scene – `__enter()` and `__exit()`. After the file opened I wanted to load data. The 'lst' variable is assigned to function 'load' (**Figure 5** - line 79) reads the pickled representation of an object from the file object and returns the reconstituted object ([pickle — Python object serialization](#)). If the file is empty I display the user a message that informs him about it (**Figure 5** - line 81), otherwise, each item existing in the list is displayed to the user (**Figure 5** - lines 82-87). This case is included in the try block, which helps me to test a block of code for errors. The except blocks (**Figure 5** - lines 89-95) help me to handle errors. The first exception which will be handled is for the case of "FileNotFoundError". This error tells you that you are trying to access a file or folder that does not exist. Because such a case can happen in our program I wanted to handle it, and give the user opportunity to continue using the program (**Figure 5** - lines 89-92). If there is another unexpected exception, it will be handled as well (**Figure 5** - lines 93-95).

At the end of the function, there is the finally block, it will be executed regardless of the result of the try- and except blocks, in my case, I am using it to return the list of items.

```
72 def load_data_from_the_file_and_display_it(lst):
73     try:
74         with open(file_name, 'rb') as file:
75             lst = pickle.load(file)
76             if not lst:
77                 print("There is no data, file is empty")
78             else:
79                 print("-----")
80                 print("There is next data:")
81                 for row in lst:
82                     print(row.get("person_name"), row.get("salary_dollars"))
83                 print("-----")
84     except FileNotFoundError:
85         print("No file found")
86         print("\n")
87     except Exception as e:
88         print(f"An error occurred while loading the data: {e}")
89         print("\n")
90     finally:
91         return lst
```

**Figure 5.** The function to load and display the data existing in the file

---

### Receive the user's choice

This is the simple function called in the main body script to get the user input (what option he wants to perform) and it returns the user choice to the caller (**Figure 6**).

```
117 def get_user_choice():
118     choice_entered = input("Enter your choice: ")
119     return choice_entered
```

**Figure 6.** The function 'get\_user\_choice' ask for the user input and returns it.

### Add new data

If the user chooses option 1, the function 'add\_new\_row' will be called. It should receive the parameter 'lst' that stands for a table of items already in the list. When this block of code is being executed, firstly, the user is asked for the name of the person. Secondly, users will be asked to enter a person's salary in \$. The value user enters cast to float, in case the value cannot be cast, the 'ValueError' can be received, so the function catches and handles it. If the error occurs, the user will be asked to repeat the input of the salary. After it, a dictionary row will be built with two keys - 'person\_name' and 'salary\_dollars' and it appends to the existing list of items. After the block of code is executed, an updated list of items returns to the main program (**Figure 7**).

```
19  # Adding new row to the list ----- #
20
21  def add_new_row(lst):
22      if not lst: # check in case paycheck_list does not exist
23          lst = []
24      person_name = input("Enter a name: ") # input for new username
25      while True: # loop for validation if salary entered as a digit
26          try:
27              salary_dollars = float(input("Enter a salary in $ :"))
28              break
29          except ValueError: # exception in case entered value for salary cannot be cast as a number
30              print("Error! Entered value is not a number") # Error message shown to user in case of string input
31      row = {"person_name": person_name, "salary_dollars": salary_dollars} # Build a dictionary
32      lst.append(row) # add a dictionary row to the table/list of items
33      return lst
```

**Figure 7.** The function 'add new row' which adds the new item to the existing list of items.

When the function is finished, the menu of options will be displayed again.

---

### Edit the existing person's salary

If the user chooses option 2, the function 'edit\_person\_salary' will be called. It should receive the parameter 'lst' that stands for a table of items already in the list (**Figure 8**). Since the user can enter this option, when the list of items is empty, the function checks if it is true. In case it is - there are no items in the list there will be a message prompt to the user, informing the user that the item list is empty. In case there are items in the list, there is a for loop. The for loop is looking for a first match. After finding the first match, the user is asked to enter a new salary. Salary should be cast to float, in case the user enters the invalid value, he will be asked to enter salary again. In case the person is not on the list, the user will be informed that this person does not exist in the list.

```
38 def edit_person_salary(lst):
39     if not lst:
40         print("There is no paychecks to edit")
41     else:
42         person_name = input("Enter a name: ")
43         i = 0
44         for row in lst:
45             i += 1
46             if row["person_name"].lower() == person_name.lower():
47                 while True:
48                     try:
49                         new_salary = float(input("Enter a salary in $ :"))
50                         break
51                     except ValueError:
52                         print("Error! Entered value is not a number") # Error message in case of string input
53                         row["salary_dollars"] = new_salary
54                         break
55             elif row["person_name"].lower() != person_name.lower() and i == len(lst):
56                 print("The person does not exist in the list")
57
```

**Figure 8.** The function 'edit\_person\_salary' edits the salary of the person, according to the user's choice.

When the function is finished, the menu of options will be displayed again.

### Display data

If the user chooses option 3, this function 'display data' will be called. It should receive the parameter 'lst' that stands for a table of items already in the list (**Figure 9**). In case the list of items is empty, the user will be informed that there is nothing in the list, and data to show. If there are items in the list, they will be printed for the user (**Figure 9**).

```

94 def display_data(lst):
95     if not lst:
96         print("There is no unsaved data to show")
97     else:
98         for row in lst:
99             print(row.get("person_name"), row.get("salary_dollars"))

```

**Figure 9.** The function 'display\_data' displays existing items to the user.

When the function is finished, the menu of options will be displayed again.

### Save data to the file

If the user chooses option 4, this function 'display data' will be called. It should receive the parameter 'lst' that stands for a table of items already in the list (**Figure 10**). Here I am opening a binary file as a file using the function open() and 'with' statement, which closes the file after finishing. I passed to function argument 'wb', which stands for the mode, 'w'- writing (rewrite existing data) in the file, and 'b' - stands for binary, otherwise without adding 'b' the function assumes I ask to open a text file. After Data is saved, there is a message which informs the user that the action is completed. If any exception inherited from the build-in 'Exception' class occurred, the user will be notified about it.

```

61 def save_data_to_the_file(lst):
62     try:
63         with open(file_name, 'wb') as file: # writing data to the file by erasing previously existing
64             pickle.dump(lst, file)
65             print("Data saved!")
66     except Exception as e: # exception in case error will happen while writing
67         print(f"An error occurred while saving the data: {e}")

```

**Figure 10.** Function 'save\_data\_to\_the\_file' writes data to the file.

When the function is finished, the menu of options will be displayed again.

### Exit the program

If the user chooses option 5, this function 'exit\_program' will be called. The program will be finished and closed (**Figure 11**).

```

102 def exit_program():
103     print("Goodbye!")
104     exit()

```

**Figure 11.** The function 'exit\_program' closes the program.

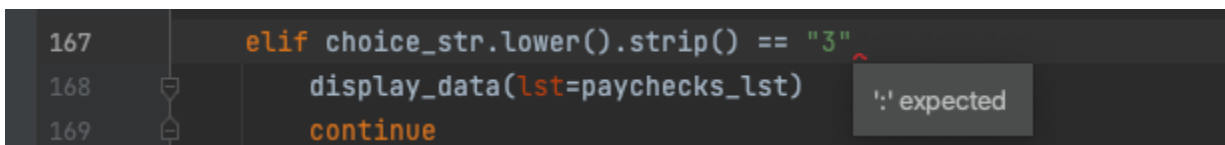


---

## Exceptions

As a part of the assignment, I had to research how to handle Exceptions in Python and about exceptions in general. There are mainly three kinds of errors in Python: syntax errors, exceptions, and logical errors. I liked this article [Error and Exception Handling in Python: Fundamentals for Data Scientists](#) it was an easy-to-read and clear article with a simple explanation of the most common errors and how to handle exceptions that are raised at runtime. The following info, I took from the [Error and Exception Handling in Python: Fundamentals for Data Scientists](#) article.

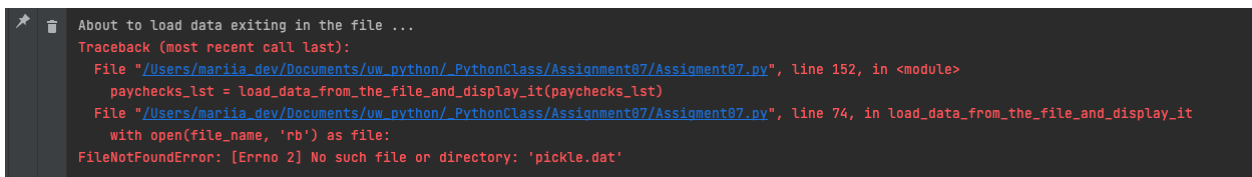
**Syntax errors** - missing symbols (comma, bracket, colon), misspelling a keyword, and having incorrect indentation are common syntax errors in Python (**Figure 12**).



```
167         elif choice_str.lower().strip() == "3"
168             display_data(lst=paychecks_lst)
169             continue
```

**Figure 12.** An example of syntax error.

**Exceptions** happen at run time. When Python cannot execute the requested action, it terminates the code and raises an error message. Trying to read from a file that does not exist, performing operations with incompatible types of variables, and more are exceptions that raise an error in Python (**Figure 13**).



```
About to load data exiting in the file ...
Traceback (most recent call last):
  File "/Users/mariia_dev/Documents/uw_python/_PythonClass/Assignment07/Assignment07.py", line 152, in <module>
    paychecks_lst = load_data_from_the_file_and_display_it(paychecks_lst)
  File "/Users/mariia_dev/Documents/uw_python/_PythonClass/Assignment07/Assignment07.py", line 74, in load_data_from_the_file_and_display_it
    with open(file_name, 'rb') as file:
FileNotFoundError: [Errno 2] No such file or directory: 'pickle.dat'
```

**Figure 13.** FileNotFoundError - example for the exception.

**Logical errors** are the most difficult errors to fix as they don't crash the code and there is any error message. For example using incorrect variable names, code that is not reflecting the algorithm logic properly, and making mistakes on boolean operators will result in logical errors.

## Exception Handling

The purpose of exception handling is to catch the error. It prevents the code execution from failures and uncontrolled stops.

### Try-except

From what I saw, for now, I assume that the try-except block is the most common method to handle exceptions.

Try block has code to be tested and monitored for the exceptions, at the same time except the block contains code for the case if a specific exception occurs. If no exception occurs then this section is skipped and the try-except statement is concluded. I used it several times in my code (while loading data or editing existing data). Try block stops when the first exception occurs. Try-except blocks may have several except blocks to handle several different types of exceptions.

Additionally, in the try-except block, it is possible to add 'else' or 'finally'. The difference is that the 'else' block is executed only if there is no exception occurring in a try block. However, 'finally' is always executed.

## Raising Exceptions

If there is a condition that happens and should be caught, it is possible to raise exceptions with a 'raise' keyword. It is possible to define a custom exception, that will happen if specific conditions are met. In my script, I defined a new exception "ValidationErrorOfUserChoice". To do so I created a new class ValidationErrorOfUserChoice (**Figure 14**), which inherits from the built-in Exception class. It overwrites the parent constructor with a specific message, and as a result, it returns a string of messages.

```
123 # define Python user-defined exceptions
124
125 class ValidationErrorOfUserChoice(Exception):
126     """Exception raised for case user entered wrong value.
127
128     Attribute:
129         user_choice -- users input
130         Message explain of the error
131     """
132
133     def __init__(self, user_choice):
134         # Call the base class constructor with the parameters it needs
135         self.user_choice = user_choice
136         self.message = f"\n -----! Error ! ----- \n" \
137             f"Entered value - {user_choice} is not correct.\n" \
138             f" Please choose a valid option.\n" \
139             f"Valid options are 1,2,3,4,5 and 6" \
140             f"\n -----! Error ! ----- \n"
141
142     def __str__(self):
143         return self.message
```

**Figure 14.** ValidationErrorOfUserChoice custom made an exception.

Since I did not want that after raising the exception, the code will be stopped. I surrounded it with a try-except block (**Figure 15**). In case a user entered an invalid choice, the exception will be raised and caught, and the error will be displayed to the user (**Figure 16**).

```

178         else:
179             try:
180                 raise ValidationErrorOfUserChoice(user_choice=choice_str)
181             except ValidationErrorOfUserChoice as e:
182                 print("Caught the exception " + repr(e) + e.message)
183             continue

```

**Figure 15.** Raise and catch ValidationErrorOfUserChoice extension.

```

5. Exit program

Enter your choice: ff
Caught the exception ValidationErrorOfUserChoice()
-----! Error ! -----
Entered value - ff is not correct.
Please choose a valid option.
Valid options are 1,2,3,4,5 and 6
-----! Error ! -----

Please choose one of the following options:
1. Add a new paycheck
2. Edit existing person salary
3. Display the data
4. Save data to the file
5. Exit program

Enter your choice: |

```

**Figure 16.** Example of error message showing after extension happened.

Run the code in the PyCharm and Terminal

After the script was finished I run in the script pycharm (**Figure 17**).

```

Please choose one of the following options:
1. Add a new paycheck
2. Edit existing person salary
3. Display the data
4. Save data to the file
5. Exit program

Enter your choice: 1
Enter a name: john
Enter a salary in $ :1000

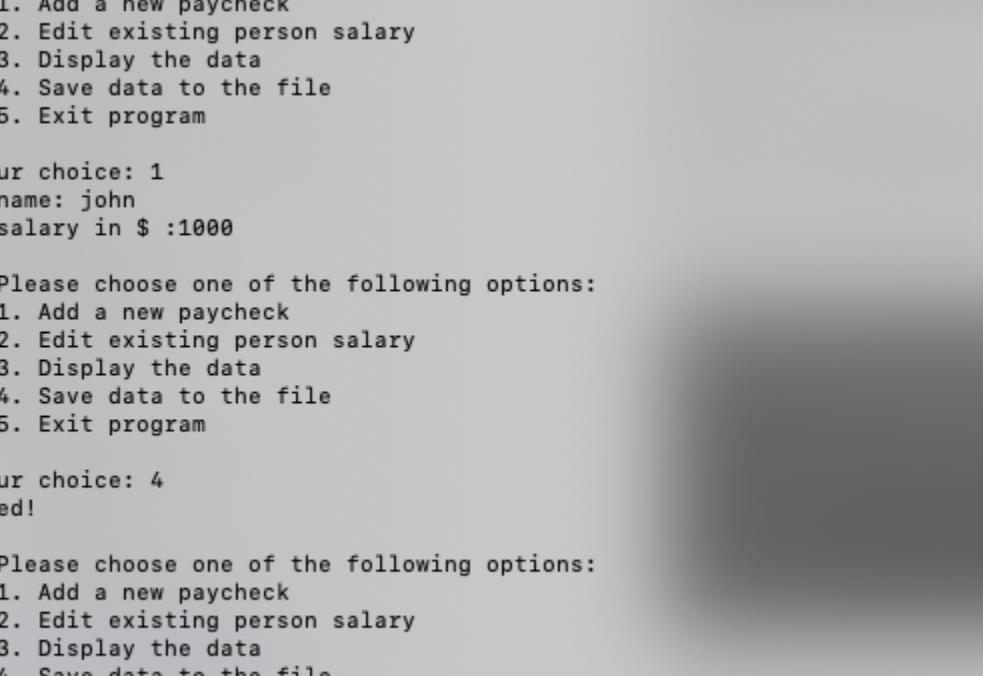
Please choose one of the following options:
1. Add a new paycheck
2. Edit existing person salary
3. Display the data
4. Save data to the file
5. Exit program

Enter your choice: 4
Data saved!

```

**Figure 17.** Result of the running script in the PyCharm.

After I run the program in the terminal (**Figure 18**).



```
mariia_dev — Assignment07.py — 86x29
Please choose one of the following options:
1. Add a new paycheck
2. Edit existing person salary
3. Display the data
4. Save data to the file
5. Exit program

Enter your choice: 1
Enter a name: john
Enter a salary in $ :1000

Please choose one of the following options:
1. Add a new paycheck
2. Edit existing person salary
3. Display the data
4. Save data to the file
5. Exit program

Enter your choice: 4
Data saved!

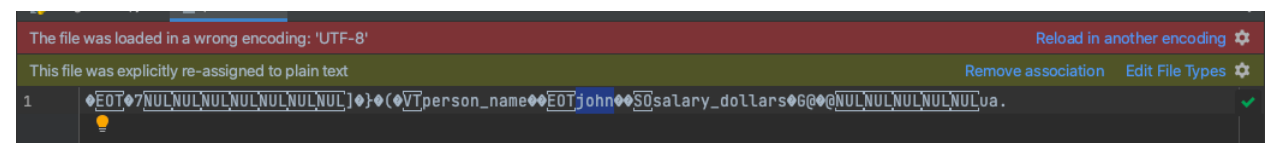
Please choose one of the following options:
1. Add a new paycheck
2. Edit existing person salary
3. Display the data
4. Save data to the file
5. Exit program

Enter your choice: 1
```

**Figure 18.** Result of the running script in Terminal

Check data is saved in the file

After the script was run I checked the file, where the data was supposed to be saved (**Figure 19**).



**Figure 19.** Result of running the script, the data is saved in the file.

## Summary

While completing the assignment I learned about the difference between binary and text files. How to read and write when working with binary files. I learned about pickling in python and that it can be dangerous, so I have been careful and unpickle only trusted files. I read and summarize different ways how to handle exceptions and tried to create my exception. Also at the end, I

---

learned about markdown, a lightweight markup language that describes how text should look on a page. Using it I created a simple webpage in GitHub, which contains this description.