

Tony Le

11/18/2019

IT FDN 100

Assignment 07

<https://github.com/TonyLe2/ITFnd100--Mod07/blob/master/docs/index.md>

Pickling and Structured Error Handling

Introduction

The assignment for this week is to research and create a script that demonstrates how pickling and structured error handling works.

Importing pickle

The first step as shown in Figure 1 is to import Pickle's methods into the script.

```
import pickle  # This imports code from another code file
```

Figure 1: Import Pickle

Declaring a class

Figure 2 establishes the variables that will be used later on in the script. We define a variable that will be used to open up our file and we assign a list to a variable to be written and read.

```
# Data ----- #  
  
file_Name = "Assignment07.txt"  
a = ['Value 1', 'Value 2', 'Value 3']
```

Figure 2: Assigning Variables

Processing the writing and reading of the data

The first part of the processing is shown in Figure 3. We first assign a variable that opens the file for writing with **fileObject**. This time, we ensure to use **'wb'** to indicate that we are writing in **binary**. We then store the data away with **pickle.dump** and close the file.

```
# Processing ----- #  
  
fileObject = open(file_Name, 'wb') # Open the file for writing  
pickle.dump(a, fileObject) # This writes the object "a" to the file named 'Assignment07.txt'  
fileObject.close() # This closes the fileObject
```

Figure 3: Write the list to file

The script in Figure 4 opens and reads the file that has been stored in binary. With the **pickle.load** command, we are able to load the stored data into a variable assigned as **b**.

```
fileObject = open(file_Name, 'rb') # Open the file to read from 'Assignment07.txt'  
b = pickle.load(fileObject) # load the object from the file into var b  
fileObject.close() # This closes the fileObject
```

Figure 4: Reading the list

Structured error handling

The presentation section is detailed in Figure 5 which also demonstrates how structured error handling works. We utilize the **try**, **except** and **else** commands in order to output a custom error message in the case it fails. In the script here, the first try will fail due to a type error and the second one will pass.

```
# Presentation ----- #  
  
try:  
    print("a is written as: " + a + " in binary file")  
    print("b is read as: " + str(b) + " from a binary file")  
except:  
    print('Error: Types do not match!\n')  
else:  
    print('No exceptions!')  
  
try:  
    print('Attempt #2: ')  
    print("a is written as: " + str(a) + " in binary file")  
    print("b is read as: " + str(b) + " from a binary file\n")  
except:  
    print('Error: Types do not match!\n')  
else:  
    print('No exceptions!')
```

Figure 5: Structured error handling

Summary

Figure 6 shows the script in its entirety and Figure 7 shows the output of the script. The output shows that it is functioning as intended with the results of the data saved in a txt file in binary. To summarize, the script begins by assignment a value to “a” in normal text as a list. This list is then written into a file in binary and the data is stored by the **pickle.dump** method. Afterwards, the data is read in its binary form and loaded into the variable “b” by the **pickle.load** method. At this point, the list is back in its original form. The output of the script shown in Figure 7 shows the text file is stored as binary data.

```
import pickle # This imports code from another code file

# Data ----- #

file_Name = "Assignment07.txt"
a = ['Value 1','Value 2','Value 3']

# Processing ----- #

fileObject = open(file_Name,'wb') # Open the file for writing
pickle.dump(a, fileObject) # This writes the object "a" to the file named 'Assignment07.txt'
fileObject.close() # This closes the fileObject

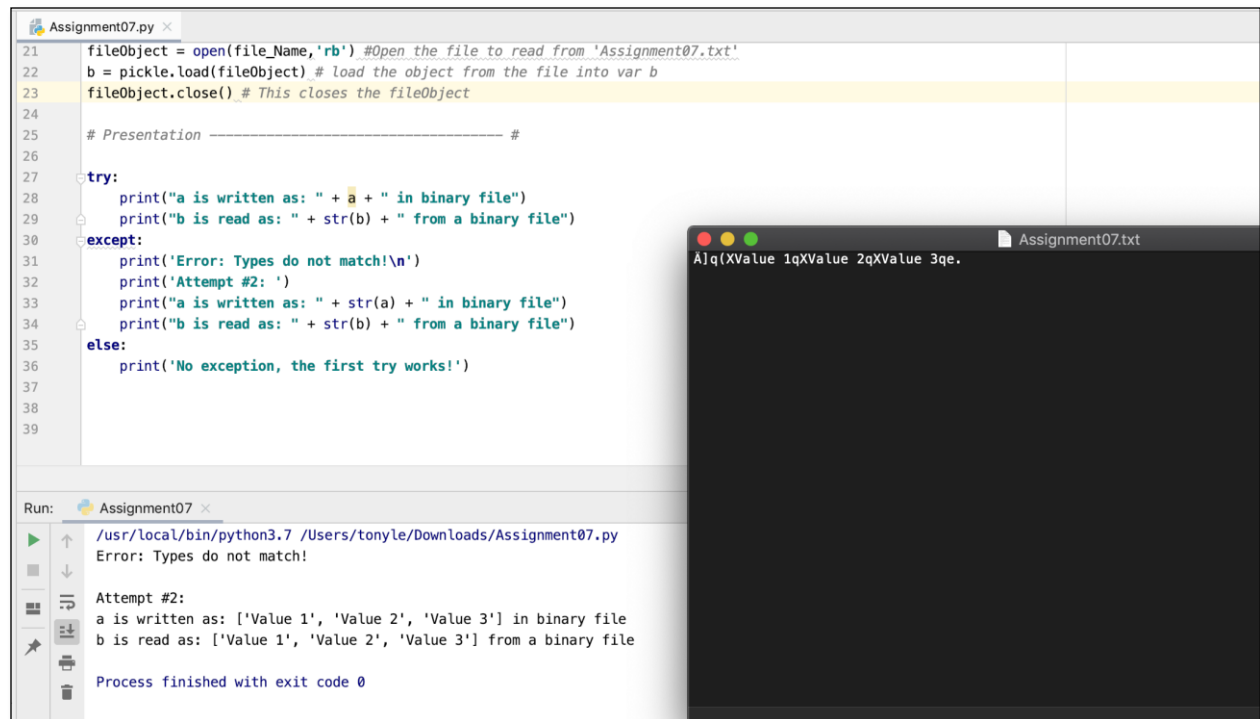
fileObject = open(file_Name,'rb') #Open the file to read from 'Assignment07.txt'
b = pickle.load(fileObject) # Load the object from the file into var b
fileObject.close() # This closes the fileObject

# Presentation ----- #

try:
    print("a is written as: " + a + " in binary file")
    print("b is read as: " + str(b) + " from a binary file")
except:
    print('Error: Types do not match!\n')
else:
    print('No exceptions!')

try:
    print('Attempt #2: ')
    print("a is written as: " + str(a) + " in binary file")
    print("b is read as: " + str(b) + " from a binary file\n")
except:
    print('Error: Types do not match!\n')
else:
    print('No exceptions!')
```

Figure 6: Full script



The image shows a Python IDE with a script named 'Assignment07.py' and its output. The script uses the pickle module to load data from a binary file. The output window shows the execution results, including an error message and the data loaded from the file.

```
21 fileObject = open(file_Name, 'rb') #Open the file to read from 'Assignment07.txt'
22 b = pickle.load(fileObject) # load the object from the file into var b
23 fileObject.close() # This closes the fileObject
24
25 # Presentation ----- #
26
27 try:
28     print("a is written as: " + a + " in binary file")
29     print("b is read as: " + str(b) + " from a binary file")
30 except:
31     print('Error: Types do not match!\n')
32     print('Attempt #2: ')
33     print("a is written as: " + str(a) + " in binary file")
34     print("b is read as: " + str(b) + " from a binary file")
35 else:
36     print('No exception, the first try works!')
```

Run: Assignment07

/usr/local/bin/python3.7 /Users/tonyle/Downloads/Assignment07.py
Error: Types do not match!

Attempt #2:
a is written as: ['Value 1', 'Value 2', 'Value 3'] in binary file
b is read as: ['Value 1', 'Value 2', 'Value 3'] from a binary file

Process finished with exit code 0

Assignment07.txt

AJq(XValue 1qXValue 2qXValue 3qe.

Figure 7: Script output showing a file stored in binary