

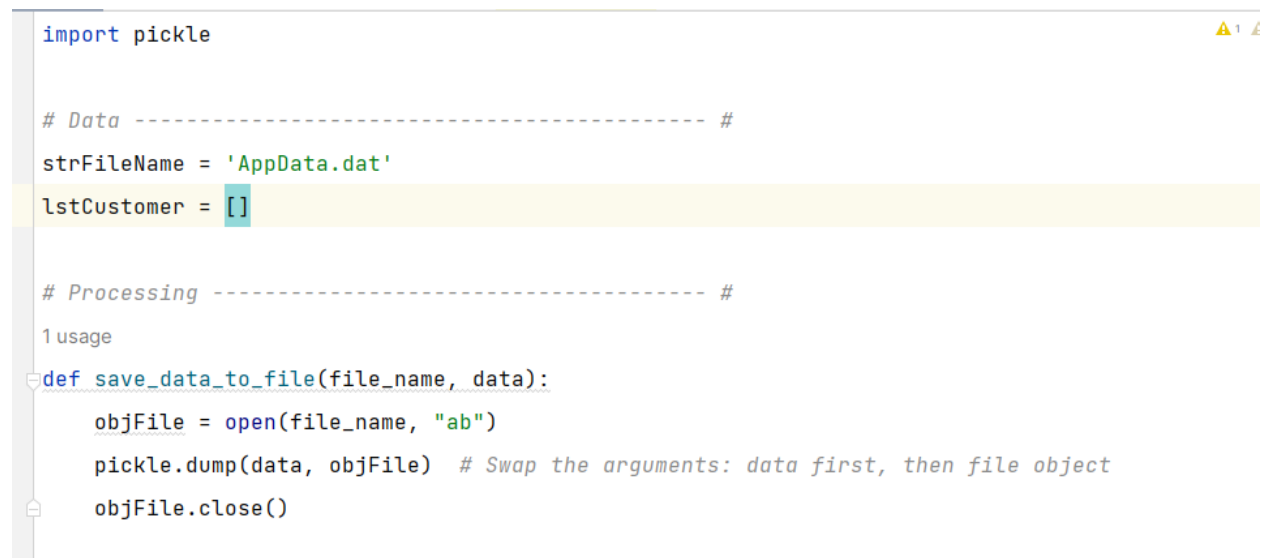
Introduction.

In module seven we begin by learning to work with files using pickling and structured exception handling. We will also be learning how to create more complex GitHub web pages. This new web page will include basic commands in a language called “markdown”.

Week 7 Labs

Lab 7.1

This lab was relatively easy to do. Create some data, save data to file, and read the data using pickling. First, we create a string and list for creating the data. Next save the data to the file by appending the binary (that is what “ab” stands for). `pickle.dump` is used to serialize(convert into a byte stream) Python objects and write them to a file or a binary stream. Serialization is the process of converting complex data structures, like dictionaries or objects, into a format that can be stored and later reconstructed. You can see each one of these steps in the example below Figure 6.1.

A screenshot of a code editor with a light blue background. The code is written in Python and is used to save data to a file using the pickle module. The code includes comments for data creation and processing. The list 'lstCustomer' is highlighted in yellow. The function 'save_data_to_file' is defined with a docstring and a usage example. The code is as follows:

```
import pickle

# Data ----- #
strFileName = 'AppData.dat'
lstCustomer = []

# Processing ----- #
1 usage
def save_data_to_file(file_name, data):
    objFile = open(file_name, "ab")
    pickle.dump(data, objFile) # Swap the arguments: data first, then file object
    objFile.close()
```

(Figure 6.1)

Next we have the script read the data from the file using pickling again. We simply have it open the file and read the binary text by inputting “rb”. Shown in figure 6.2.

```
def read_data_from_file(file_name):  
    file = open(file_name, "rb")  
    data = pickle.load(file)  
    file.close()  
    return data  
  
# Presentation ----- #  
intId = int(input("Enter an Id: "))  
strName = input("Enter a Name: ")  
lstCustomer = [intId, strName]  
  
# Save the list object into a binary file  
save_data_to_file(strFileName, lstCustomer)
```

(Figure 6.2)

The final step of this lab saves the list object to a binary file. Then the script loads the data by using the `read_data_from_file` that I defined in the first block of the script and I present that data back to the user by simply making a print command for the loaded data. In figure 6.3 you can see how this was executed. This concludes lab 7.1.

```
# Save the list object into a binary file  
save_data_to_file(strFileName, lstCustomer)  
  
# Read the data from the file into a new list object and display the contents  
loaded_data = read_data_from_file(strFileName)  
print("Loaded Data:", loaded_data)
```

(Figure 6.3)

Exception handling

An exception in python is an unexpected event that occurs during program execution. Errors that occur at runtime are called **exceptions** or **logical errors**. These occur when; try to open a file that does not exist, try to divide a number by zero, or try to import a module that does not exist.

- The try...except block is used to handle exceptions in python.

```
try:
    # code that may cause exception
except:
    # code to run when exception occurs
```

For each try block, there can be zero or more except blocks. Multiple except blocks allow us to handle each exception differently. The argument type of each except block indicates the type of exception that can be handled by it. For example,

try:

```
even_numbers = [2,4,6,8]
```

```
print(even_numbers[5])
```

except ZeroDivisionError:

```
print("Denominator cannot be 0.")
```

except IndexError:

```
print("Index Out of Bound.")
```

Output: Index Out of Bound

Here, we are trying to access a value to the index 5. Hence, index error exception occurs

```
Index Out of Bound.
```

The ZeroDivisionError exception is skipped and the IndexError is executed.

- **Try with else clause.**

In some situations you might want to run a certain block of code if the code block inside try runs without any errors.

For these cases, you can use the optional else keyword with the try statement.

Here's an example:

```
try:
    num = int(input("Enter a number: "))
    assert num % 2 == 0
except:
    print("Not an even number!")
else:
    reciprocal = 1/num
    print(reciprocal)
```

Output: Index Out of Bound

If we pass an odd number.

```
Enter a number: 1
Not an even number!
```

If we pass an even number, the reciprocal is computed and displayed.

```
Enter a number: 4
0.25
```

However, if we pass a 0, we get the ZeroDivisionError as the code block inside the else is not handled by preceding.

```
Enter a number: 0
Traceback (most recent call last):
  File "C:\Python class\Assignment07\Exception errors.py", line 7, in <module>
    reciprocal = 1/num
                ~^~~~
ZeroDivisionError: division by zero
```

- **Python try...finally**

In Python, the finally block is always executed no matter whether there is an exception or not . The finally block is optional. For each try block, there can be only one finally block. Lets see an example,

```
try:
    numerator = 10
    denominator = 0

    result = numerator / denominator

    print(result)
except:
    print("Error: Denominator cannot be 0.")

finally:
    print("This is finally block.")
```

Output

```
C:\Users\Gunner\AppData\Local\Microsoft\W
Error: Denominator cannot be 0.
This is finally block.
```

In this example, we are dividing a number by 0 inside the try block. Here, this code generates an exception. The exception is caught by the except block, then the finally block is executed.

To learn more on exception handling visit: [Python Exception Handling \(With Examples\)](https://programiz.com/python/python-exception-handling/)
(programiz.com)

Pickling

Pickling is the process whereby a python object hierarchy is converted into a byte stream. Unpickling is the process by which original Python objects are retrieved from the string representation i.e., from the pickle file. It's a process that involves serializing (converting) a python object into a format that can be stored or transmitted, and later deserialized (reconstructed) to obtain the original object.

The term “pickling” comes from the analogy with the process of preserving vegetables by pickling them in a jar. In python, pickling allows you to preserve the state of complex data structures, including dictionaries, lists, and classes, so they can be saved to afile, database, or storage medium.

Python provides the built-in “pickle” module to facilitate pickling and unpickling.

Here I provided some script as an example of pickling:

```
1  import pickle
2
3  # Data to be pickled
4  data = {
5      'name': 'Gunner Santana',
6      'age': 31,
7      'city': 'Seattle'
8  }
9
10 # Pickle the data
11 with open('data.pkl', 'wb') as file:
12     pickle.dump(data, file)
13
14 # Unpickle the data
15 with open('data.pkl', 'rb') as file:
16     loaded_data = pickle.load(file)
17
18 print(loaded_data)
```

Assignment 07

In assignment seven the task was to demonstrate how pickling and structured error handling work. I kept my code really simple so it would be clear on how exactly each process works.

1. The first step is to define my data structure. I made some simple data consisting of name, age, and city you live in. You can see this below in figure 7.1.

```
1 import pickle
2
3 # Define a sample data structure
4 data = {
5     'name': 'John Doe',
6     'age': 30,
7     'city': 'Seattle'
8 }
```

(Figure 7.1)

2. Next, I write the data to a file using the pickle method. When the file opens it will write binary, hence the 'wb'. The 'w' opens the file in write mode and the 'b' indicates the file should be treated as a binary file. See figure 7.2 for an example. I begin to add in some error handling, often referred to as "try-except" blocks.

```
# Pickling: Writing data to a file using pickle

1 usage

def save_data_to_file(filename, data):
    try:
        with open(filename, 'wb') as file:
            pickle.dump(data, file)
            print(f"Data saved to '{filename}' successfully.")
    except IOError as e:
        print(f"Error saving data: {e}")
    except Exception as e:
        print(f"An unexpected error occurred: {e}")
```

(Figure 7.2)

3. The third step is to unpickle and read the binary file. Similar to when you open and write a binary file, this time it will open and read the binary final. That is what the 'rb' stands for in the script below in figure 7.3. Again, I use structured error handling in the form of "try-except" block.

Gunner Santana

08/19/2023

IT FDN 110A

Assignment 07

<https://gsantana667.github.io/IntroToProg-Python-Mod06/>

```
# Unpickling: Reading data from a file using pickle
1 usage
def load_data_from_file(filename):
    try:
        with open(filename, 'rb') as file: # reads binary file
            loaded_data = pickle.load(file)
            print("Loaded data:", loaded_data)
    except FileNotFoundError:
        print(f'{filename} not found.')
    except pickle.PickleError as pe:
        print(f"Error loading data: {pe}")
    except Exception as e:
        print(f"An unexpected error occurred: {e}")
```

(Figure 7.3)

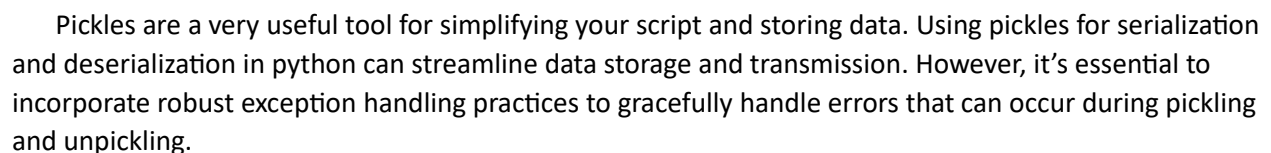
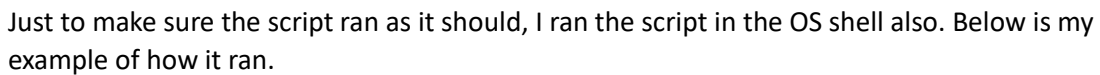
4. The final section of the code encapsulates the main logic of the script. It initializes the filename, saves data to a file using pickling, and then loads data from the file using unpickling. The code inside this block will only execute when the script is run directly, ensuring that these operations are performed when the script is intended to be used as a standalone program. If this script is imported into another script, this block won't be executed, allowing the functions and data to be used in a modular way. See figure 7.4 for example.

```
30 # Demonstrate pickling and structured error handling
31 if __name__ == "__main__":
32     filename = "data.pkl"
33
34     # Saving data to a file
35     save_data_to_file(filename, data)
36
37     # Loading data from a file
38     load_data_from_file(filename)
```

(Figure 7.4)

Here is what the text file looks like when writing binary code using pickling.

<https://gsantana667.github.io/IntroToProg-Python-Mod06/>



Gunner Santana

08/19/2023

IT FDN 110A

Assignment 07

<https://gsantana667.github.io/IntroToProg-Python-Mod06/>