Neetu Upadhyay
May 31, 2023
Foundations of Programming, Python
Assignment 07
GitHubURL :- https://github.com/upd-neetu/IntroToProg-Python-Mod07
GitHub Webpage :- https://github.com/upd-neetu/ITFnd100-Mod07

Basic Writing and Executing a Python Script

Introduction

In this module, we will provide a brief overview of key topics in Python programming. During this introduction, we will cover the benefits of using functions to encapsulate code, the advantages of structured error handling, the distinctions between text and binary files, the usage of the Exception class, creating custom exception classes, an introduction to Markdown, and utilizing Markdown on GitHub webpages. Our exploration of these topics aims to enhance your understanding of Python and related concepts.

Benefits of Putting Built-in Python Commands into Functions

Putting built-in Python commands into functions offers several benefits:

1. Reusability: By encapsulating a command or a set of commands into a function, you can reuse that functionality multiple times throughout your code. This promotes code reusability and helps avoid repetition.

2. Modularity: Functions allow you to break down your code into smaller, modular units. This makes your code more organized and easier to understand, maintain, and debug. Each function can focus on a specific task or operation, enhancing code readability.

3. Abstraction: Functions provide a level of abstraction, allowing you to hide the implementation details of a command or a series of commands. This abstraction simplifies the usage of complex commands by providing a high-level interface.

4. Code organization: By placing related commands within a function, you can group them together based on their purpose or functionality. This improves code organization and makes it easier to navigate and locate specific sections of code.

5. Code maintainability: Functions make it easier to update or modify a specific command or set of commands. Instead of making changes throughout your code, you can focus on modifying the function implementation. This reduces the chances of introducing errors and makes code maintenance more efficient.

6. Overall, putting built-in Python commands into functions promotes code reusability, modularity, abstraction, code organization, and maintainability, leading to more efficient and readable code.

Differences Between Text Files and Binary Files

Text files and binary files are two different types of file formats with distinct characteristics. Here are the key differences between them:

1. Content Representation: Text files store data as plain text using character encoding schemes like ASCII or Unicode. They primarily contain human-readable text, including letters, numbers, symbols, and control characters. On the other hand, binary files store data in a binary format, which represents information as a sequence of 0s and 1s. Binary files can contain any type of data, including text, numbers, images, audio, video, and executable code.

2. Data Encoding: In a text file, characters are encoded using specific character encoding schemes like UTF-8 or UTF-16, which assign numerical values to each character. Each character is represented by a specific code point. In contrast, binary files store data in their raw binary form, without any specific character encoding. The binary data can represent complex structures, such as numeric values, byte arrays, or custom data structures.

3. Readability: Text files are human-readable, and their content can be easily viewed and edited using a text editor. Each character is interpreted as a specific character based on the encoding scheme. Binary files, however, are not intended for direct human consumption. They contain encoded binary data that may not be understandable or interpretable by humans without specialized tools or knowledge.

4. Portability: Text files are generally more portable across different platforms and systems. They can be opened and read by any text editor or word processor, regardless of the underlying operating system. Binary files, on the other hand, may have platform-specific formats or binary data representations, making them less portable and requiring specific software or tools to interpret their contents accurately.

5. File Size: Text files tend to have larger file sizes compared to binary files when storing the same information. This is because text files often include additional formatting characters, line breaks, and metadata. Binary files are more compact since they store data directly in binary form without any extra formatting.

6. Editing and Manipulation: Text files can be easily modified or manipulated using standard text editing operations. Binary files, however, require specialized software or programming techniques to modify their contents accurately. Modifying binary files typically involves working with the binary data structures and understanding the specific file format.

Understanding the differences between text files and binary files is crucial when working with different types of data and choosing the appropriate file format for storing and processing information.

Using the Exception Class in Python

Using the Exception class allows you to handle and manage exceptional or error situations in your Python code. Exceptions are raised when an error occurs, and by using the Exception class, you can catch and handle these exceptions gracefully. The Exception class is the base class for all built-in exceptions in Python, and you can also create custom exception classes by deriving from it. It provides methods and attributes that allow you to gather information about the exception and take appropriate actions in response to it. By using the Exception class effectively, you can improve the reliability and robustness of your Python programs.

Deriving a Custom Exception Class from the Exception Class

To derive a new class from the Exception class in most programming languages, including Python, you need to create a new class that inherits from the Exception class. This new class will inherit all the properties and methods of the Exception class and can have additional features specific to your needs.

Here's an example in Python:

```
class CustomException(Exception):
    pass
```

In the above code, CustomException is the new class that derives from the built-in Exception class. The pass statement is used to indicate that the class has no additional properties or methods defined at the moment. However, you can customize the CustomException class by adding your own properties and methods as needed.

Here's an example of adding a custom message to the CustomException class:

```
class CustomException(Exception):
    def __init__(self, message):
        self.message = message
        super().__init__(self.message)
```

In this updated code, we define an __init__ method that takes a message parameter. The message parameter is assigned to the self.message attribute, which allows you to provide a custom message when creating an instance of the CustomException class. Additionally, we call the __init__ method of the base Exception class using super().__init__(self.message) to ensure that the base class is properly initialized.

By creating your own custom exception class, you can handle specific types of errors or exceptional situations in your code, making it easier to identify and handle those situations differently from other exceptions.

Reasons for Creating a Custom Exception Class Derived from the Exception Class

You might create a class derived from the Exception class in several scenarios. Here are a few common situations where creating a custom exception class can be useful:

1. Specific error handling: When you want to handle a specific type of error or exceptional situation in your code, creating a custom exception class allows you to differentiate it from other exceptions. For example, if you're building a file handling module, you could create a FileNotFoundError class derived from Exception to handle cases where a requested file cannot be found.

2. Custom error messages: By deriving from the Exception class, you can add additional properties or methods to your custom exception class. This enables you to include specific information or error messages relevant to the exceptional situation you're handling. For instance, you could create a InvalidInputException class with a custom message indicating the specific input that caused the exception.

3. Domain-specific exceptions: In certain domains or projects, it can be beneficial to define your own exception hierarchy to reflect the specific requirements and challenges of your application. By creating custom exception classes derived from Exception, you can structure the exception hierarchy in a way that makes sense for your application's logic and workflows.

4. Granular exception handling: By creating custom exception classes, you can catch and handle exceptions at different levels of granularity. For example, if you have different layers in your application architecture (such as presentation layer, business logic layer, and data access layer), you can define specific exception classes for each layer to handle errors more precisely and take appropriate actions.

Creating custom exception classes provides you with more control over error handling, improves code readability, and allows for better organization and maintenance of your codebase. It helps you distinguish and handle different types of exceptional situations in a structured manner, leading to more robust and manageable code.

Introduction to Markdown: A Lightweight Markup Language for Structured Content

Markdown is a lightweight markup language that provides a simple and readable syntax for formatting plain text. It was created by John Gruber and Aaron Swartz in 2004 with the goal of making it easy to write and read structured content that can be converted into HTML.

Markdown is widely used for writing documentation, README files, forum posts, blog articles, and other types of content where formatting and structure are important but a complex markup language like HTML would be cumbersome.

The syntax of Markdown is designed to be intuitive and human-readable, using plain text with special characters and formatting conventions to indicate various elements and styles. For example, using asterisks or underscores around a word or phrase will render it as italicized or bold in the converted output.

Here are some common features of Markdown:

- Headers: Using one to six hash (#) symbols at the beginning of a line to create different levels of headers.
- Lists: Creating ordered or unordered lists using numbers or bullets.
- Links: Adding hyperlinks with descriptive text.
- Images: Embedding images with alt text and a URL.
- Emphasis: Applying italics or bold to text.
- Code blocks: Displaying code snippets or blocks of code.
- Blockquotes: Quoting text with indentation.
- Tables: Creating simple tables with rows and columns.

One of the advantages of Markdown is its portability and ease of conversion. Markdown files can be easily converted to other formats, such as HTML, PDF, or even Word documents, using various tools and libraries.

Overall, Markdown provides a convenient way to create structured and formatted content without the need for complex HTML tags or extensive knowledge of markup languages. It allows writers and content creators to focus on the content itself, while still maintaining a visually appealing and well-formatted output.

Using Markdown on a GitHub Webpage: Creating Structured and Formatted Content

To use Markdown on a GitHub webpage, follow these steps:

1. **Create a new file**: Start by creating a new file in your GitHub repository. You can create a new file by navigating to the desired folder in your repository and clicking on the "Create new file" button.

2. **Name the file**: Give the file a name with the **.md** extension. This extension indicates that the file should be treated as a Markdown file.

3. **Edit the file**: Once you've created the Markdown file, click on it to open the file editor. In the editor, you can start writing your content using Markdown syntax.

4. **Format your content**: Use Markdown syntax to format your content. For example, you can add headers using '**#**', create lists using **'-'** or **'1'**., add links using **'[text](url)',** include images using **'![alt text](image_url)'**, and apply emphasis using ***'*italic*'*** or **bold**.

5. **Preview your changes**: To see a preview of how your Markdown will be rendered, you can click on the "Preview changes" tab in the GitHub editor. This allows you to check how your content will appear when viewed as a webpage.

6. **Commit changes**: Once you are satisfied with your Markdown content, scroll down to the "Commit changes" section. Provide a commit message to describe your changes, and then click on the "Commit changes" button to save your file to the repository.

7. **View the rendered webpage**: After committing your changes, GitHub will automatically render the Markdown file as an HTML webpage. You can access the rendered webpage by clicking on the file in your repository's file list.

GitHub natively supports Markdown rendering, allowing you to create rich and well-formatted content directly in your repository. By using Markdown, you can easily create headings, lists, links, and other elements to enhance the readability and structure of your webpage.

Python Script

Writing my 7th Python script. To do this, I opened pycharm, which is a Python environment for writing and running scripts. I watched some videos recommended by Professor Root to learn how to complete my tasks.

Here's a simplified step-by-step description of the code screenshot is attached please see figure 1, 2 & 3:

Code Description:

The code demonstrates a simple example of storing and reading data in a binary file using the pickle module. Here's a breakdown of the code:

1. The code starts with a brief title, description, and change log information.

2. The pickle module is imported to enable serialization and deserialization of Python objects.

3. The script initializes the variables strFileName and lstCustomer. strFileName represents the name of the binary file to be created or read, and lstCustomer is an empty list to store customer data.

4. Two functions are defined:

- save_data_to_file(file_name, list_of_data): This function takes a file name and a list of data as input and saves the data to a binary file using pickle.dump().
- read_data_from_file(file_name): This function takes a file name as input and reads the data from the binary file using pickle.load(). It returns the deserialized data.

5. The code prompts the user to enter a customer ID and name, creates a dictionary (customer) with this information, and appends it to the lstCustomer list.

6. The save_data_to_file() function is called to save the lstCustomer data to the binary file specified by strFileName.

7. A message is displayed to indicate that the data has been saved to the file.

8. The read_data_from_file() function is called to read the data from the binary file specified by strFileName. The deserialized data is stored in the new_list variable.

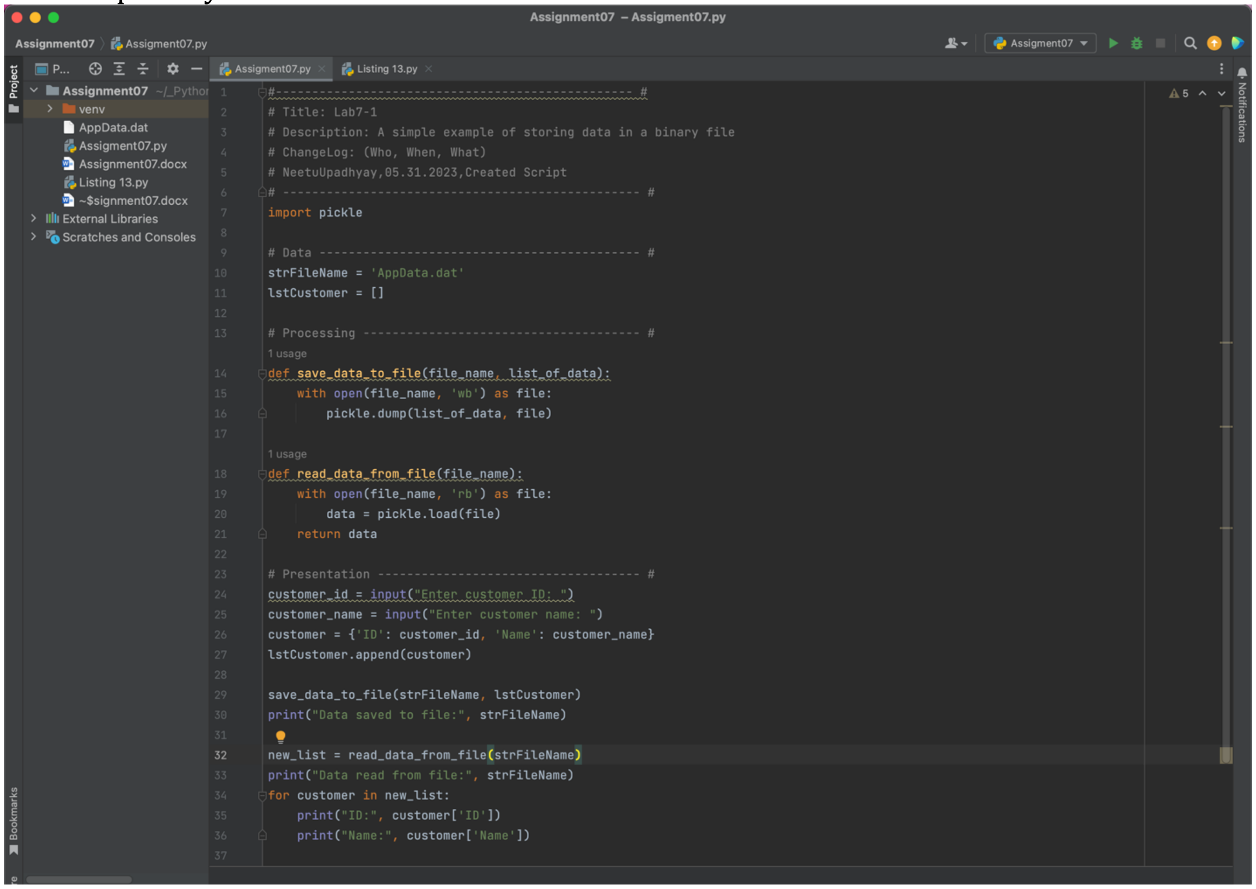
9. A message is displayed to indicate that the data has been read from the file.

10. Finally, a loop iterates over the new_list and prints the ID and name of each customer.

Error Handling:

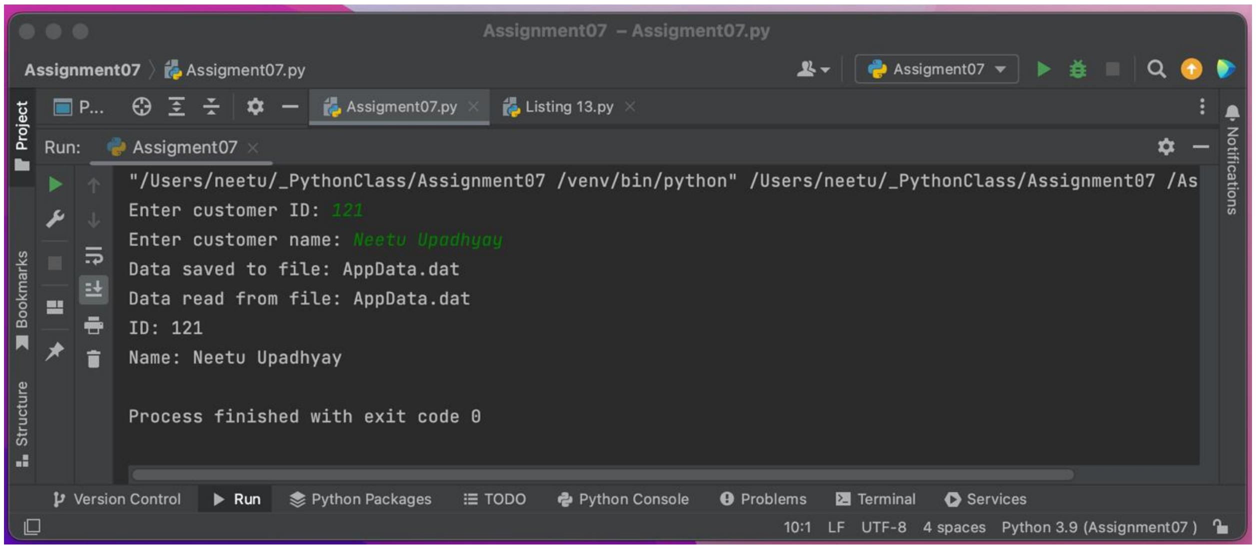
The provided code does not explicitly handle errors related to file operations or user inputs. If any error occurs during file operations (e.g., file not found, permission denied), an exception will be raised, potentially causing the program to terminate with an error message. It would be beneficial to include appropriate error handling mechanisms such as try-except blocks to handle and gracefully manage such errors.

Run script in Pycharm



```python
#-----------------------------------------------#
# Title: Lab7-1
# Description: A simple example of storing data in a binary file
# ChangeLog: (Who, When, What)
# NeetuUpadhyay,05.31.2023,Created Script
# ---------------------------------------------- #
import pickle

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

# Processing ------------------------------------ #
def save_data_to_file(file_name, list_of_data):
    with open(file_name, 'wb') as file:
        pickle.dump(list_of_data, file)

def read_data_from_file(file_name):
    with open(file_name, 'rb') as file:
        data = pickle.load(file)
    return data

# Presentation ---------------------------------- #
customer_id = input("Enter customer ID: ")
customer_name = input("Enter customer name: ")
customer = {'ID': customer_id, 'Name': customer_name}
lstCustomer.append(customer)

save_data_to_file(strFileName, lstCustomer)
print("Data saved to file:", strFileName)

new_list = read_data_from_file(strFileName)
print("Data read from file:", strFileName)
for customer in new_list:
    print("ID:", customer['ID'])
    print("Name:", customer['Name'])
```

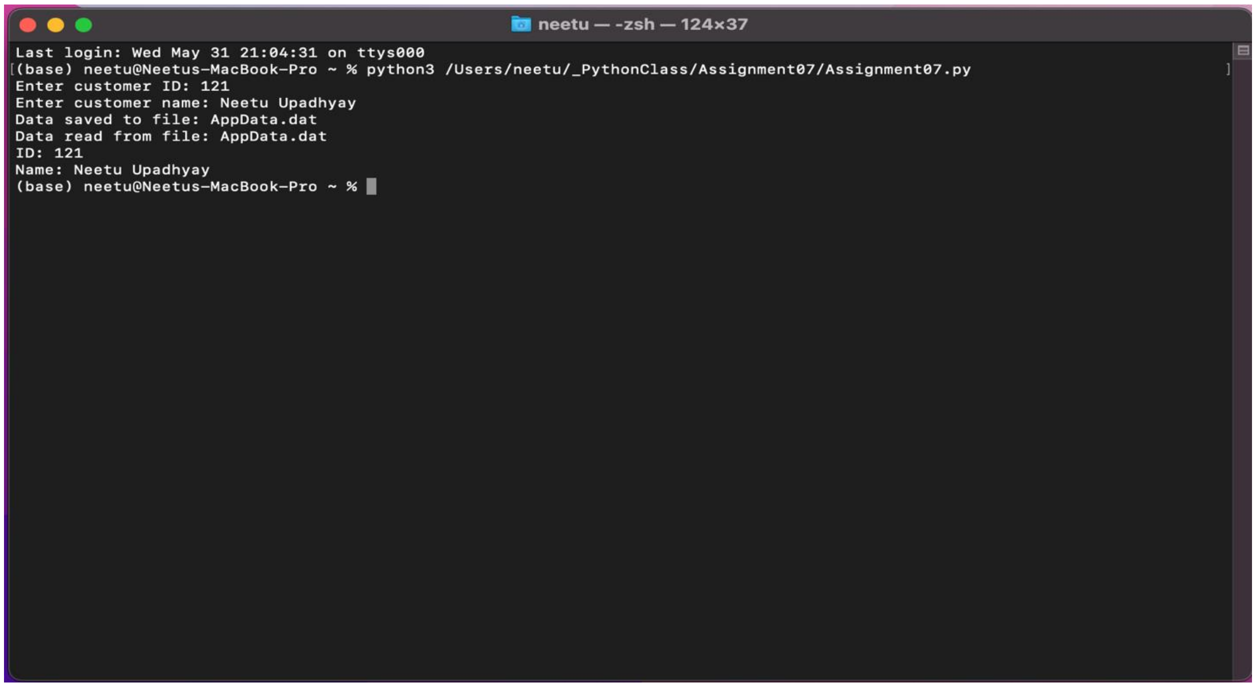
Figure 1



```
"/Users/neetu/_PythonClass/Assignment07 /venv/bin/python" /Users/neetu/_PythonClass/Assignment07 /As
Enter customer ID: 121
Enter customer name: Neetu Upadhyay
Data saved to file: AppData.dat
Data read from file: AppData.dat
ID: 121
Name: Neetu Upadhyay

Process finished with exit code 0
```

Figure 2

Run script in Terminal



```
Last login: Wed May 31 21:04:31 on ttys000
(base) neetu@Neetus-MacBook-Pro ~ % python3 /Users/neetu/_PythonClass/Assignment07/Assignment07.py
Enter customer ID: 121
Enter customer name: Neetu Upadhyay
Data saved to file: AppData.dat
Data read from file: AppData.dat
ID: 121
Name: Neetu Upadhyay
(base) neetu@Neetus-MacBook-Pro ~ %
```

Figure 3

Facing and Correcting a File Path Error

During the execution of my code, I encountered an error related to the file path. Here's the error message I received:

```
Last login: Wed May 31 21:01:11 on ttys000
(base) neetu@Neetus-MacBook-Pro ~ % python3 /Users/neetu/_PythonClass/Assignment07 /Assignment07.py
python3: can't open file '/Users/neetu/_PythonClass/Assignment07': [Errno 2] No such file or directory
(base) neetu@Neetus-MacBook-Pro ~ %
```

This error indicated that the file or directory /Users/neetu/_PythonClass/Assignment07 could not be found. To address this issue, I took the following steps:

1. I double-checked the file path I provided and realized that I had mistakenly added an extra slash before the script name.

2. I corrected the file path by removing the extra slash and ensuring that the correct path to the directory was specified.

Here is the corrected command:

```
python3 /Users/neetu/_PythonClass/Assignment07/Assignment07.py
```

By removing the extra slash and specifying the correct path to the Assignment07.py script, I resolved the file path error. I then successfully executed the code and achieved the desired outcome.

Summary:

In the INTRO TO PROGRAMMING (PYTHON) ASSIGNMENT 07, we covered various topics related to Python programming. Here's a brief overview of what we did:

The provided document discusses several topics related to programming in Python. It covers the benefits of using functions to encapsulate code, the differences between text files and binary files, using the Exception class for error handling in Python, an introduction to Markdown as a lightweight markup language, and using Markdown on a GitHub webpage. The document also includes a description of a Python script that demonstrates storing and reading data in a binary file using the pickle module, along with error handling considerations.

The benefits of putting built-in Python commands into functions are highlighted, including reusability, modularity, abstraction, code organization, and maintainability. Text files and binary files are compared in terms of content representation, data encoding, readability, portability, file size, and editing/manipulation capabilities. The usage of the Exception class in Python is explained, along with an example of creating custom exception classes derived from the Exception class to handle specific types of errors. Markdown is introduced as a lightweight markup language for structured content, offering simplicity and portability. The process of using Markdown on a GitHub webpage is described step-by-step, from creating a new file with the .md extension to previewing and committing changes.

The Python script described in the document focuses on storing and reading data in a binary file using the pickle module. It includes functions for saving data to a file and reading data from a file, along with prompts for user input and error handling considerations. The script demonstrates how to append customer data to a list, save it to a binary file, read the data from the file, and print the customer information.

A file path error encountered during the execution of the script is mentioned, along with the steps taken to correct it. The error message indicated that the specified file or directory could not be found, and the correction involved double-checking the file path and removing an extra slash before the script name.