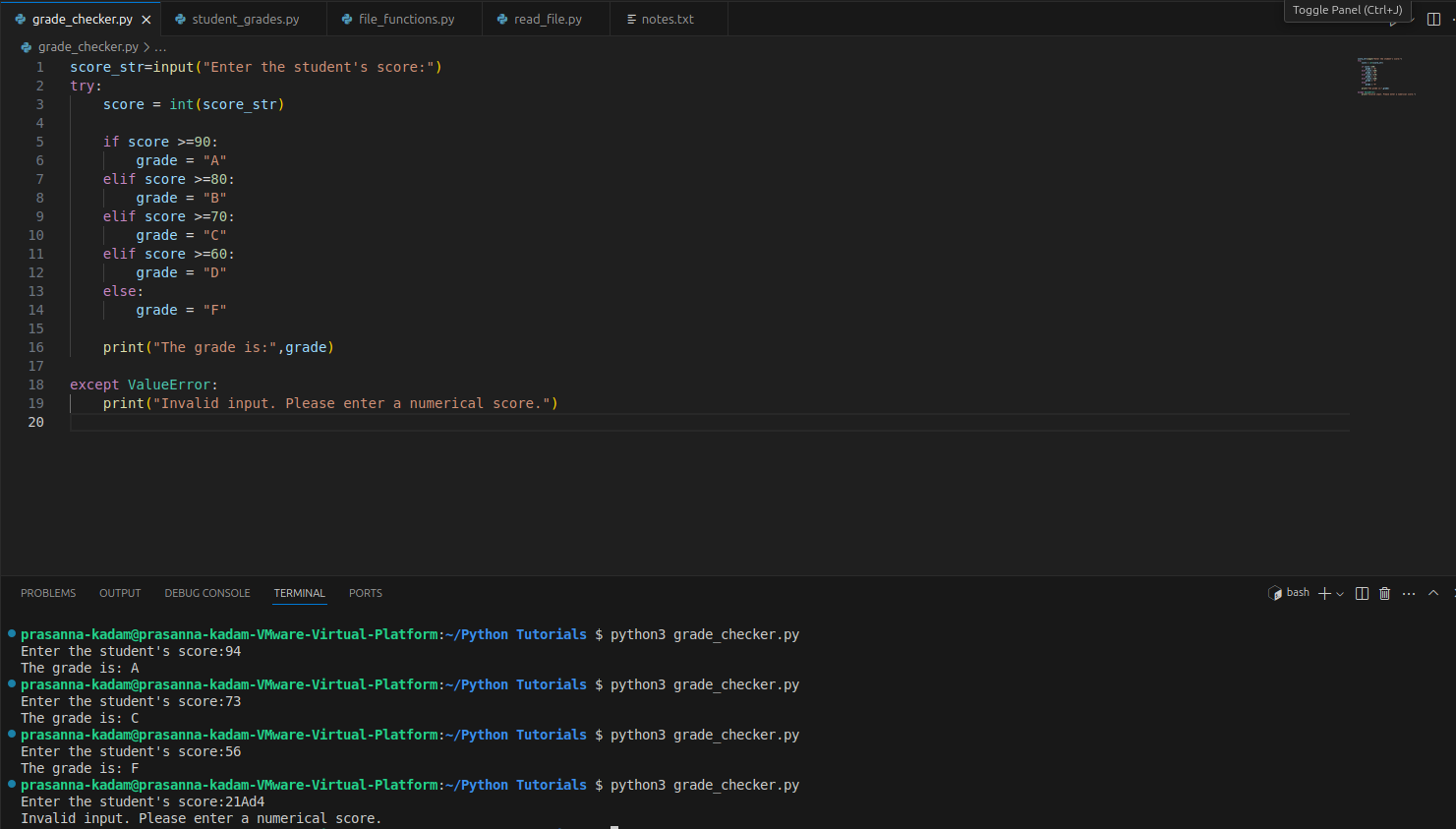
1. Grade Checker.

This program implements a conditional logic structure to assign letter grades based on a given numerical score. The program takes an integer representing a student's score as input and evaluates it against predefined grading criteria using a series of sequential conditional statements.

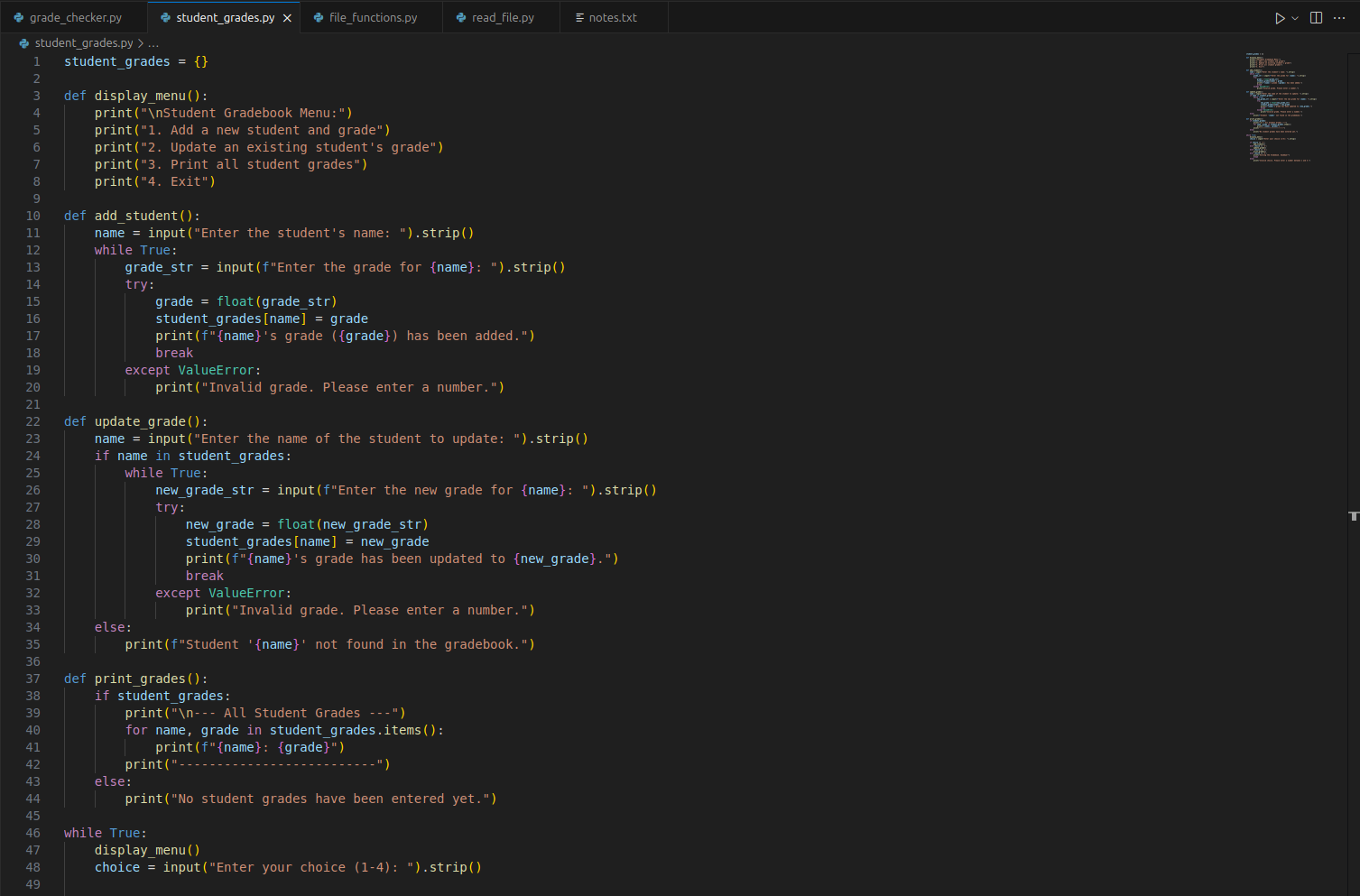
* Input Acquisition: The program begins by acquiring a numerical score from the user. This input is parsed as an integer to facilitate numerical comparisons.
* Sequential Conditional Evaluation: The core logic employs a series of if-elif-else statements to determine the appropriate grade. The score is evaluated against descending thresholds:
* An if statement checks if the score is greater than or equal to 90, assigning "A" if true.
* Subsequent elif (else if) statements evaluate progressively lower score ranges (80-89 for "B", 70-79 for "C", 60-69 for "D"). These conditions are only evaluated if the preceding if or elif conditions are false, ensuring a mutually exclusive assignment of grades.
* A final else statement captures any score below 60, assigning the grade "F".
* Output Generation: Upon determining the grade based on the evaluated conditions, the program outputs the resulting letter grade to the console.

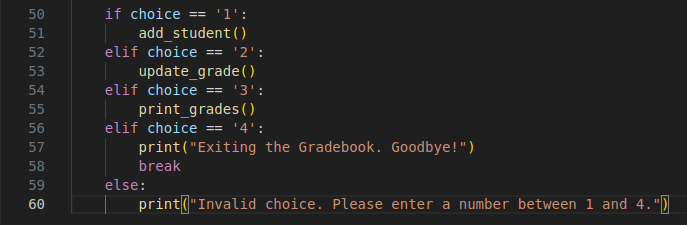


2. Student Grades.

This program implements a basic student grade management system utilizing a dictionary data structure. The system provides a menu-driven interface enabling users to perform fundamental data management operations on student records, specifically adding new records, updating existing ones, and displaying all current records.

* Data Structure Initialization: An empty dictionary, student\_grades, is initialized. This dictionary serves as the primary data store, with student names acting as unique keys and their corresponding grades as the associated values. The dictionary structure facilitates efficient data retrieval and manipulation based on student names.
* User Interface Presentation: The display\_menu() function presents a clear set of operational choices to the user: adding a new student and grade, updating an existing grade, printing all stored grades, and exiting the application.
* Adding New Records (add\_student()):
* The program prompts the user to input the name of the new student.
* Subsequently, it prompts for the student's grade. Input validation, typically employing a try-except block to handle non-numeric input, ensures data integrity.
* The provided student name and grade are then added as a new key-value pair to the student\_grades dictionary.
* Updating Existing Records (update\_grade()):
* The user is prompted to enter the name of the student whose grade requires modification.
* The program checks for the existence of the entered student name as a key within the student\_grades dictionary.
* If the student record exists, the user is prompted to input the updated grade. Input validation may be applied here as well.
* The grade associated with the student's name in the dictionary is updated with the new value.
* If the student name is not found, an informative message is displayed to the user.
* Displaying All Records (print\_grades()):
* The program first checks if the student\_grades dictionary contains any entries.
* If records exist, it iterates through the dictionary's key-value pairs using the .items() method.
* For each student, the program formats and prints their name and corresponding grade to the console.
* If the dictionary is empty, a message indicating the absence of student grade data is displayed.
* Main Program Loop Implementation: A while True loop forms the application's main execution cycle. It continuously presents the user menu and processes the selected option. Based on the user's input, the appropriate function (add\_student(), update\_grade(), print\_grades()) is invoked. The loop terminates only when the user selects the "Exit" option.

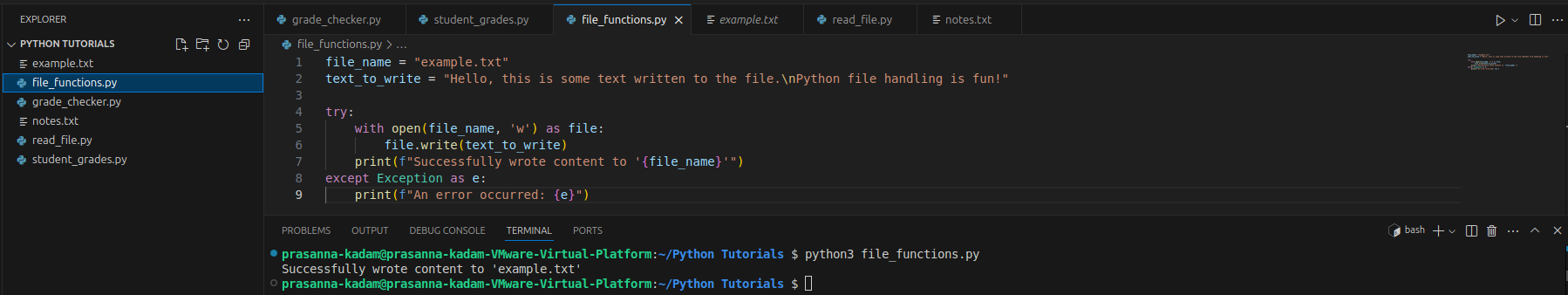




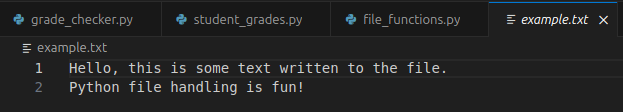
3. Write to a File.

This program demonstrates the fundamental operation of creating and writing content to a text file using Python's file handling mechanisms. It utilizes the open() function in write mode ('w') to establish a file object and the write() method to persist textual data to the file system.

* File Name Definition: The program begins by defining a string variable representing the desired name of the text file to be created or modified (e.g., file\_name = "output.txt").
* Content Specification: The textual content intended for writing to the file is defined as a string variable. This string may encompass single or multiple lines, with newline characters (\n) embedded to denote line breaks within the file.
* File Opening in Write Mode: The open() function is invoked with the specified file name and the mode specifier 'w'. This action attempts to open the file in write mode. The behavior of this mode is as follows:
* If the specified file does not exist, a new file with that name is created.
* If the file already exists, its entire contents are truncated (deleted) before the new content is written.
* The with open(...) as file: construct ensures robust file handling by automatically closing the file object upon exiting the with block, even in the event of exceptions. The opened file object is assigned to the variable file.
* Data Persistence via write(): The write() method of the file object (file.write(content\_to\_write)) is then used to write the defined string content into the opened file. The write() method writes the string exactly as provided, without any implicit formatting.
* Implicit File Closure: The with statement guarantees that the file is properly closed after the writing operation is completed, ensuring data integrity and releasing system resources.
* (Optional) Error Handling: A try-except block can be incorporated to manage potential exceptions that may arise during file operations, such as file system permission errors.



Output:



4. Read from a File.

This program demonstrates the process of reading the entire content of an existing text file using Python's file handling capabilities. It utilizes the open() function in read mode ('r') to establish a connection to the file and the read() method of the file object to retrieve the file's contents as a single string.

Step-by-Step Breakdown:

* File Name Specification: The program begins by defining a string variable containing the name of the text file to be read (e.g., file\_name\_to\_read = "input.txt"). It is assumed that this file exists in the program's execution environment.
* File Opening in Read Mode: The open() function is invoked with the specified file name and the mode specifier 'r'. This action attempts to open the file in read mode, which allows for non-destructive access to the file's contents. If no mode is explicitly specified, 'r' is the default. The with open(...) as file: construct ensures that the file object is automatically closed after the reading operation is complete. The opened file object is assigned to the variable file.
* Content Retrieval via read(): The read() method of the file object (file.read()) is called to read the entire contents of the file from the current file pointer position to the end of the file. The retrieved content is returned as a single string.
* Content Display: The string containing the file's content is then printed to the console, making the information accessible to the user.
* Implicit File Closure: The with statement ensures that the file is automatically closed after the reading operation, even if exceptions occur during the process.
* (Optional) Error Handling: A try-except block is often employed to handle potential FileNotFoundError exceptions that may occur if the specified file does not exist in the expected location, as well as other potential file access errors.

