📁Title: Basic Python Operations for Beginners

Overview:

This project comprises four beginner-level Python scripts that demonstrate essential programming concepts, including conditional logic, data structures, file handling, and user interaction. These programs are designed to teach fundamental programming concepts using:

* Control Structures (if-else)
* Data Structures (dictionary)
* Functions
* File input/output
* Error Handling

🔢 1. Grade Checker

Description:

This script accepts a numerical score from the user as input and prints the corresponding grade based on standard grading criteria.

Features:

* Input validation for numeric input
* Modularized using functions

**Logic:**

# Function to get a valid score from the user

def get\_score():

while True:

try:

# Prompt the user to enter a score

score = int(input("Enter the score (0-100): "))

# Check if the score is within the valid range

if 0 <= score <= 100:

return score # Return the valid score

else:

print("Score must be between 0 and 100.") # Validation failed

except ValueError:

# Handle non-integer inputs

print("Invalid input. Please enter a valid number.")

# Function to determine grade based on score

def determine\_grade(score):

if score >= 90:

return "A"

elif score >= 80:

return "B"

elif score >= 70:

return "C"

elif score >= 60:

return "D"

else:

return "F"

# Main function to run the program

def main():

score = get\_score() # Get a valid score from the user

grade = determine\_grade(score) # Determine the grade using the score

print(f"Grade: {grade}") # Print the grade

# Entry point of the program

if \_\_name\_\_ == "\_\_main\_\_":

main()

## **🧠 Code Explanation (Step-by-Step)**

### **🔹 get\_score() — Input Validation**

* **Purpose**: Ensures the user enters a number between 0 and 100.
* **Loop**: Uses a “while True” loop to repeatedly prompt the user until valid input is given.
* **Error Handling**: Uses try-except to catch and respond to invalid (non-integer) input using ValueError.
* **Range Check**: Ensures the score is within the range 0–100.

### **🔹 determine\_grade(score) — Grade Decision Logic**

* **Purpose**: Determines the letter grade based on the input score.
* **Logic**:
  + 90+: A
  + 80–89: B
  + 70–79: C
  + 60–69: D
  + <60: F
* Uses clean if-elif-else statements for readability and logic hierarchy.

### **🔹 main() — Orchestrator**

* **Purpose**: Acts as the main controller of the program.
* **Calls**:
  + Calls get\_score() to receive valid user input.
  + Calls determine\_grade(score) to calculate the grade.
  + Prints the final result.

### **🔹 if \_\_name\_\_ == "\_\_main\_\_":**

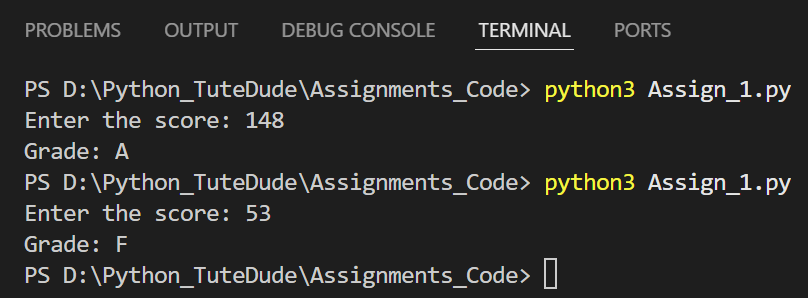
* **Purpose**: This is a Python best practice.
* Ensures that the main() function only runs when the file is executed directly (not when imported as a module).

## **📝 Example Run**

Enter the score (0-100): 85

Grade: B

If the user enters something like abc or a number like -5, it will prompt again until a valid number is entered.



👩‍🎓 2. Student Grades Manager

Description:

A menu-driven program to manage student grades using a dictionary with persistent file storage.

Features:

* Add, update, and view grades
* Saves/loads data from a text file
* Input validation
* File error handling

Logic:

import json # For file-based JSON storage

import os # To check file existence

# JSON file name where grades will be stored

FILENAME = "grades.json"

# Load student grades from file (if exists), otherwise return empty dictionary

def load\_grades():

if os.path.exists(FILENAME): # Check if file exists

try:

with open(FILENAME, "r") as f:

return json.load(f) # Load and return data from JSON

except json.JSONDecodeError:

# If the file is corrupted or unreadable, start fresh

print("Error reading grade data. Starting with an empty record.")

return {}

return {} # File doesn't exist; start with empty grades

# Save the student grades dictionary to the JSON file

def save\_grades(grades):

with open(FILENAME, "w") as f:

json.dump(grades, f) # Convert dictionary to JSON and save

# Add a new student and grade

def add\_student(grades):

name = input("Enter student name: ").strip()

if name in grades:

print(f"{name} already exists.")

else:

grade = input("Enter grade: ").strip().upper()

grades[name] = grade

print(f"Added {name} with grade {grade}.")

# Update an existing student's grade

def update\_grade(grades):

name = input("Enter student name to update: ").strip()

if name in grades:

grade = input("Enter new grade: ").strip().upper()

grades[name] = grade

print(f"Updated {name}'s grade to {grade}.")

else:

print(f"{name} does not exist.")

# Display all students and their grades

def print\_grades(grades):

if grades:

print("\nAll Student Grades:")

for name, grade in grades.items():

print(f"{name}: {grade}")

else:

print("No student records found.")

# Menu system to interact with the user

def menu():

grades = load\_grades() # Load existing data

while True:

print("\n1. Add Student")

print("2. Update Grade")

print("3. View Grades")

print("4. Exit")

choice = input("Enter choice (1-4): ")

if choice == "1":

add\_student(grades)

elif choice == "2":

update\_grade(grades)

elif choice == "3":

print\_grades(grades)

elif choice == "4":

save\_grades(grades) # Save data before exiting

print("Grades saved. Exiting.")

break

else:

print("Invalid choice.")

# Entry point of the program

if \_\_name\_\_ == "\_\_main\_\_":

menu()

## **🧠 Code Explanation (Step-by-Step)**

### **🔸 import json, import os**

* json: Used to **save/load** grades from a file in JSON format.
* os: Used to check if the file already exists (os.path.exists()).

### **🔸 FILENAME = "grades.json":** Defines the file name used to store student grades.

### **🔸 load\_grades()**

* Loads the grades from the file if it exists.
* Uses json.load() to parse the JSON.
* Handles errors like:
  + File not found: returns an empty dictionary.
  + File is corrupted/unreadable: prints a message and starts with an empty dictionary.

### **🔸 save\_grades(grades)**

* Takes the grades dictionary and saves it to grades.json.
* Uses json.dump() for writing structured data.

### **🔸 add\_student(grades)**

* Prompts the user for a name.
* Checks if the student already exists.
* If not, it gets a grade and adds the new entry.

### **🔸 update\_grade(grades)**

* Allows the user to update an existing student's grade.
* Verifies that the student exists before updating.

### **🔸 print\_grades(grades)**

* Displays all students and their grades from the dictionary.
* If the dictionary is empty, it notifies the user.

### **🔸 menu()**

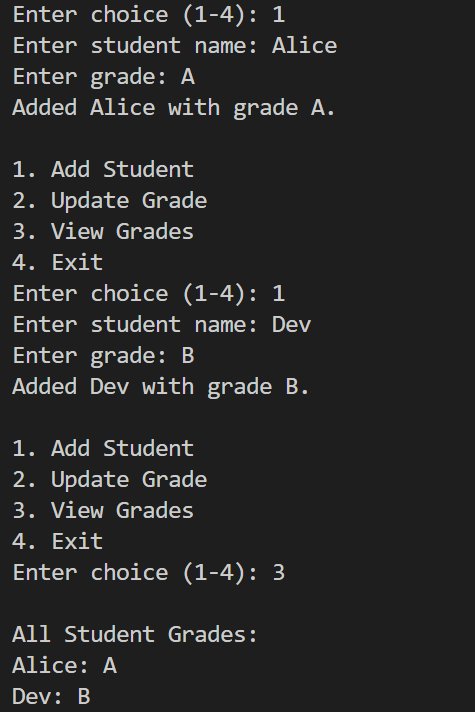
* Acts as the main loop to interact with the user.
* Presents a menu to choose options:
  1. Add Student
  2. Update Grade
  3. View Grades
  4. Exit and Save

### **🔸 if \_\_name\_\_ == "\_\_main\_\_":**

* Ensures that the program starts by calling menu() only when run directly.
* If this file is imported into another script, it won't execute automatically.

## **✅ Benefits of This Implementation**

| **Feature** | **Description** |
| --- | --- |
| **Persistent Storage** | Stores student grades in a JSON file (grades.json) |
| **Modular Code** | Organized into reusable, testable functions |
| **Input Validation** | Handles existing names and normalizes input with .strip() and .upper() |
| **Exception Handling** | Safely manages missing or corrupted files |
| **Scalable** | Can be extended to store more fields (e.g., subject, marks, etc.) |



📝 3. Write to a File

Description:

Writes multiline user input to a specified file. Uses functions and exception handling for reliability.

Features:

* Takes a filename and content as input
* Supports multiline input (stopped with END)
* Handles file write errors

Logic:

# Function to write content to a file

def write\_to\_file(filename, content):

try:

# Open the file in write mode

with open(filename, "w") as f:

f.write(content) # Write the user-provided content to the file

print(f"Content written to '{filename}' successfully.")

except Exception as e:

# Catch and display any error that occurs while writing

print(f"An error occurred: {e}")

# Function to get multiline input from the user

def get\_multiline\_input():

print("Enter your content (type 'END' on a new line to finish):")

lines = []

while True:

line = input()

if line.strip().upper() == "END":

break

lines.append(line)

return "\n".join(lines) # Combine all lines with newline characters

# Main program

def main():

filename = input("Enter the filename to write to (e.g., output.txt): ").strip()

content = get\_multiline\_input() # Get content from user

write\_to\_file(filename, content) # Save content to the specified file

# Entry point of the program

if \_\_name\_\_ == "\_\_main\_\_":

main()

## **🧠 Code Explanation (Step-by-Step):**

### **🔹 write\_to\_file(filename, content)**

* **Purpose**: Encapsulates the file-writing logic.
* **with open(filename, "w")**: Opens the file for writing. Creates the file if it doesn’t exist, or overwrites it if it does.
* **f.write(content)**: Writes the text to the file.
* **try...except**: Catches any unexpected errors (e.g., permission issues, invalid filename).

### **🔹 get\_multiline\_input()**

* **Purpose**: Allows the user to enter multiple lines of input until they type "END" on a new line.
* Uses a **loop to collect lines** and appends them to a list.
* Joins all lines into a single string using "\n".join(lines).

### **🔹 main()**

* **Gets the filename** from the user.
* Calls get\_multiline\_input() to capture the file content.
* Calls write\_to\_file() to save the content to disk.

### **🔹 if \_\_name\_\_ == "\_\_main\_\_":**

* This ensures the script runs only when executed directly (not when imported elsewhere).

## 

## 

## **✅ Features & Best Practices**

| **Feature** | **Description** |
| --- | --- |
| **Function-Based Design** | Code is reusable and easy to maintain |
| **Exception Handling** | Prevents the program from crashing during file operations |
| **Multiline Input** | Supports writing long texts or paragraphs |
| **User-Friendly** | Provides clear prompts and feedback to users |

## 

## 

## **💡 Example Output**

Enter the filename to write to (e.g., output.txt): mytext.txt

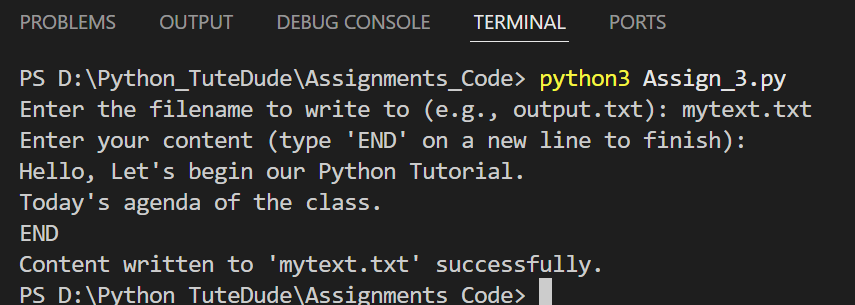
Enter your content (type 'END' on a new line to finish):

Hello, let’s begin our Python Tutorial.

Today’s agenda of the class.

END

Content written to 'notes.txt' successfully.



📖 4. Read from a File

Description:

This script reads and prints content from a specified file. Includes error handling for missing or unreadable files.

Features:

* Handles missing files
* Clean output format
* Function-based structure

Logic:

# Function to read and display content from a file

def read\_from\_file(filename):

try:

# Open the file in read mode

with open(filename, "r") as f:

content = f.read() # Read all content from the file

print("\nFile Content:")

print(content) # Print the read content

except FileNotFoundError:

# If the file does not exist

print(f"File '{filename}' not found.")

except Exception as e:

# Catch and display any other error that occurs

print(f"An error occurred: {e}")

# Main function to handle user interaction

def main():

filename = input("Enter the filename to read from: ").strip() # Ask user for filename

read\_from\_file(filename) # Call the function to read the file

# Entry point of the program

if \_\_name\_\_ == "\_\_main\_\_":

main()

## **🧠 Code Explanation (Step-by-Step):**

### **🔹 def read\_from\_file(filename):** This is the **core function** responsible for reading from a file.

* **with open(filename, "r") as f:**
  + Opens the file in **read mode**.
  + Automatically closes the file when the block ends.
* **f.read()**
  + Reads the **entire content** of the file as a string.
* **print(content)**
  + Displays the file’s content to the user.
* **Error Handling:**
  + **FileNotFoundError**: Catches the case where the file does not exist.
  + **Generic Exception**: Catches any other errors (e.g., permission errors, encoding errors).

### **🔹 def main():** Handles the **user interface**.

* Asks the user to **enter a file name**.
* Calls read\_from\_file() to attempt reading it.

### **🔹 if \_\_name\_\_ == "\_\_main\_\_":**

Ensures the script runs only when executed directly.

