Name: Syed Ali Mosavi

Intern ID: TN/IN02/PY/028

Email ID: alimonster105@gmail.com

Task week: 02

Internship Domain: Python Development

Instructor Name: Mr. Hassan Ali

Tasks – Lists

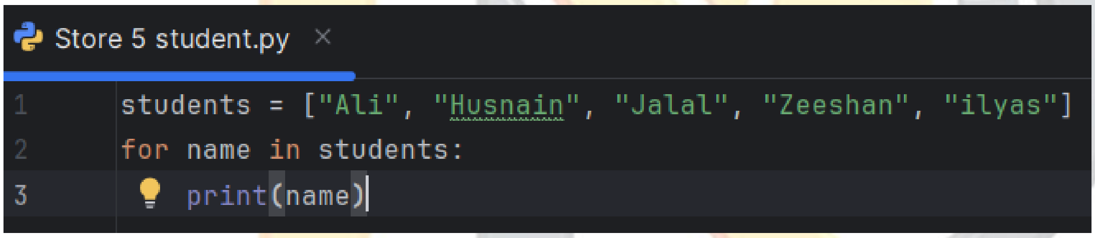
Task 01:

Store 5 student names & print each

Step-by-Step Instructions

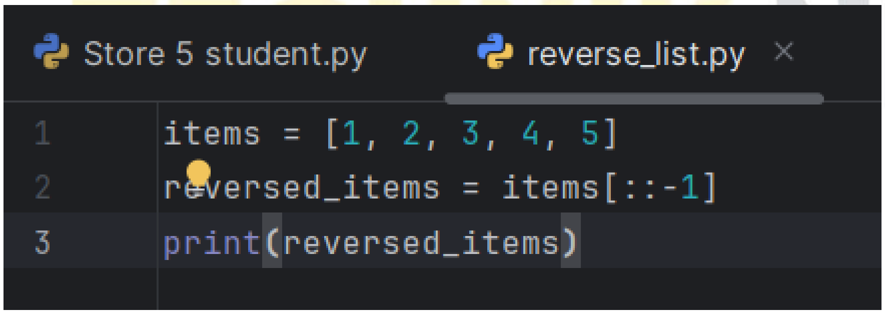
1. I created a file named student_list.py.
2. I defined a list students containing five names.
3. I used a for loop to iterate over students and print each name.

Code Snippet:



```python
students = ["Ali", "Husnain", "Jalal", "Zeeshan", "ilyas"]
for name in students:
    print(name)
```

Output Snippet:



```python
items = [1, 2, 3, 4, 5]
reversed_items = items[::-1]
print(reversed_items)
```

Learning and Challenges

- I practiced creating and using Python lists to hold multiple items.
- I reinforced how a for loop walks through each element in a list.

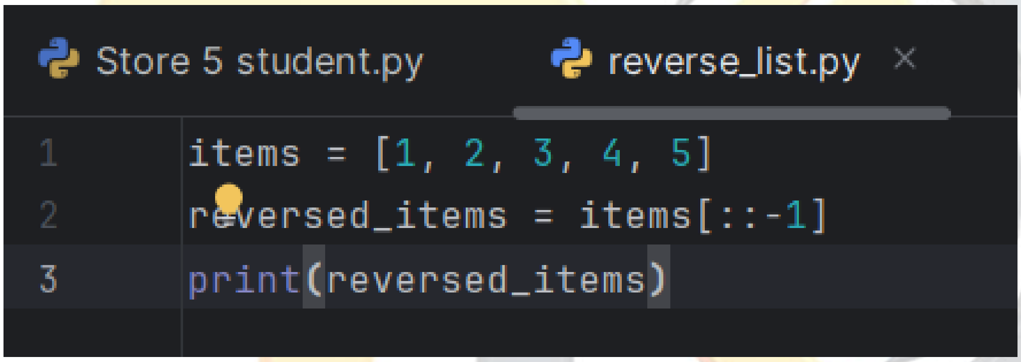
<u>Task 02:</u>

Reverse list without reverse().

Step-by-Step Instructions

1. I created a file named reverse_list.py.
2. I defined a list items with several elements.
3. I used list slicing items[::-1] to produce a new list in reverse order.
4. I printed the reversed list.

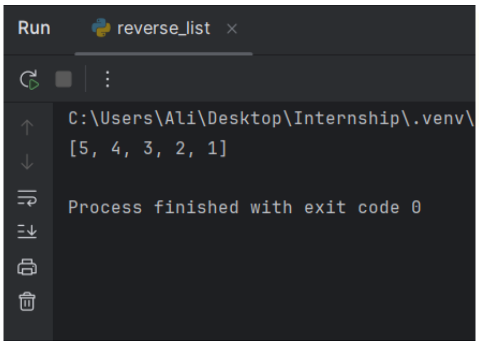
Code Snippet:



Store 5 student.py   reverse_list.py

```python
items = [1, 2, 3, 4, 5]
reversed_items = items[::-1]
print(reversed_items)
```

Output Snippet:



Run   reverse_list

```
C:\Users\Ali\Desktop\Internship\.venv\
[5, 4, 3, 2, 1]

Process finished with exit code 0
```

Learning and Challenges

- I learned that **slicing** with [::-1] creates a reversed copy of a list.
- This avoids modifying the original list and doesn't rely on built-in methods like reverse().

Tasks – Tuples

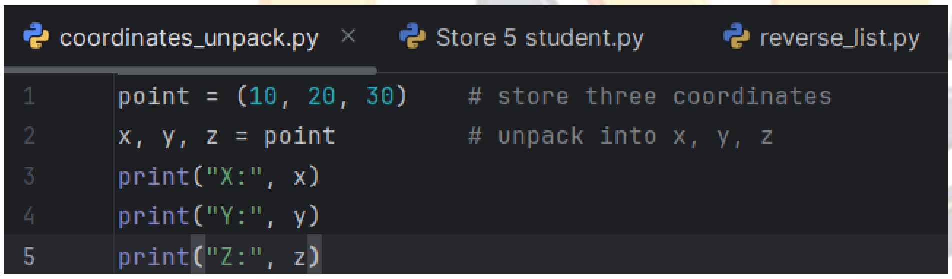
Task 03:

Store 3 coordinates & unpack.

Step-by-Step Instructions

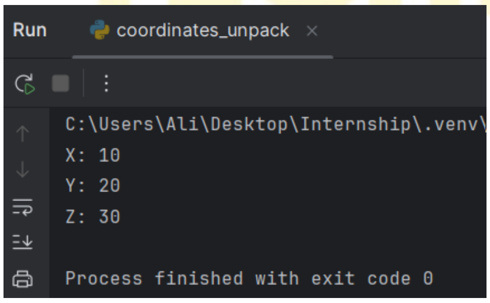
1. I created a file named coordinates_unpack.py.
2. I defined a tuple point with three values (x, y, z).
3. I unpacked the tuple into three separate variables x, y, and z.
4. I printed each coordinate.

Code Snippet:



```python
point = (10, 20, 30)    # store three coordinates
x, y, z = point         # unpack into x, y, z
print("X:", x)
print("Y:", y)
print("Z:", z)
```

Output Snippet:



```
C:\Users\Ali\Desktop\Internship\.venv\
X: 10
Y: 20
Z: 30

Process finished with exit code 0
```

Learning and Challenges

- I practiced using **tuples** to hold fixed-size groups of values.

- I learned how **tuple unpacking** assigns each element to a separate variable in one line.

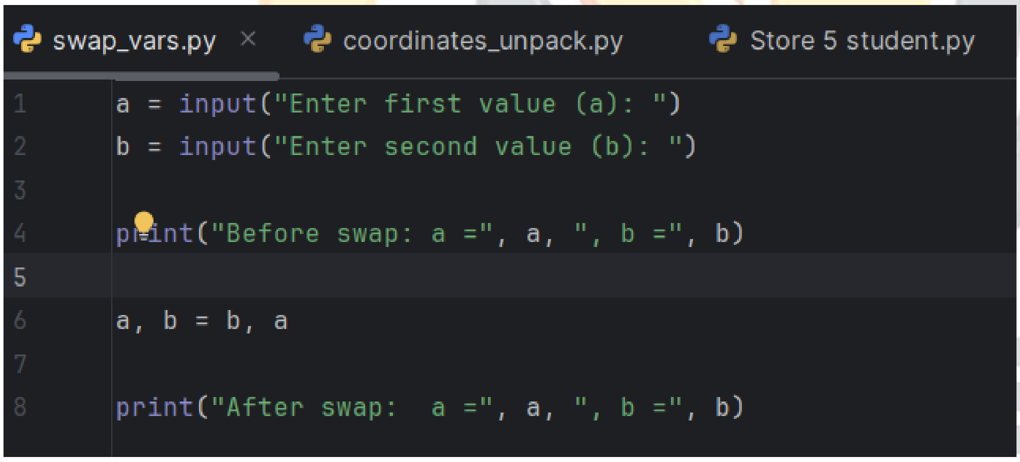
Task 04:

Swap Two Variables Using Tuple Assignment.

Step-by-Step Instructions

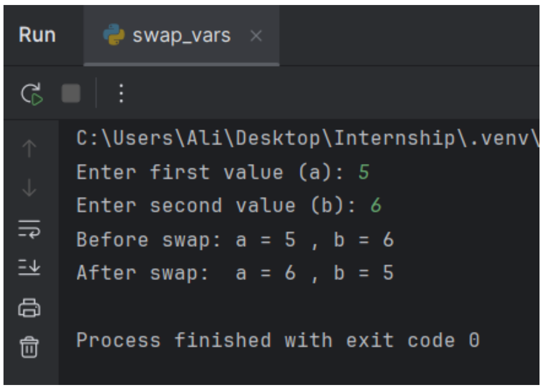
1. I created a file named swap_vars.py.
2. I used input() to read two values into variables a and b.
3. I printed their values before swapping.
4. I used tuple assignment (a, b = b, a) to swap them without a temporary variable.
5. I printed their values after swapping.

Code Snippet:



```python
a = input("Enter first value (a): ")
b = input("Enter second value (b): ")

print("Before swap: a =", a, ", b =", b)

a, b = b, a

print("After swap:  a =", a, ", b =", b)
```

Output Snippet:



```
C:\Users\Ali\Desktop\Internship\.venv\
Enter first value (a): 5
Enter second value (b): 6
Before swap: a = 5 , b = 6
After swap:  a = 6 , b = 5

Process finished with exit code 0
```

Learning and Challenges

- I learned that a, b = b, a lets Python swap values in one step without a temporary variable.
- Seeing the before/after printouts helps confirm the swap worked correctly.

Tasks – Sets

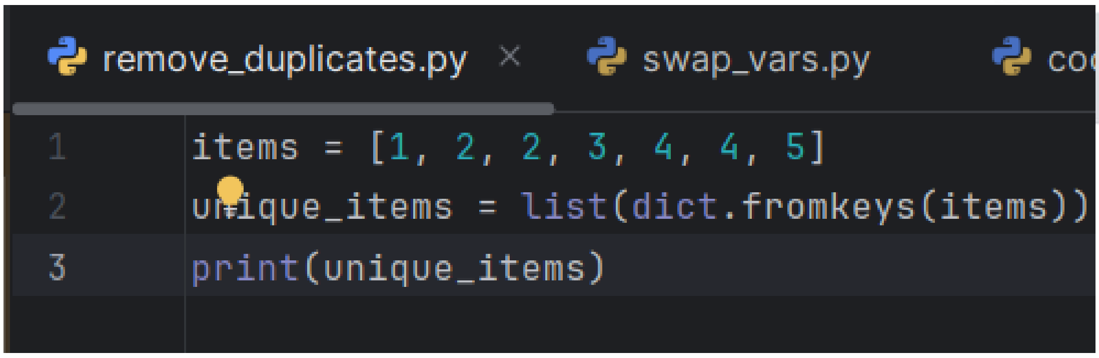
Task 05:

Remove Duplicates from a List.

Step-by-Step Instructions

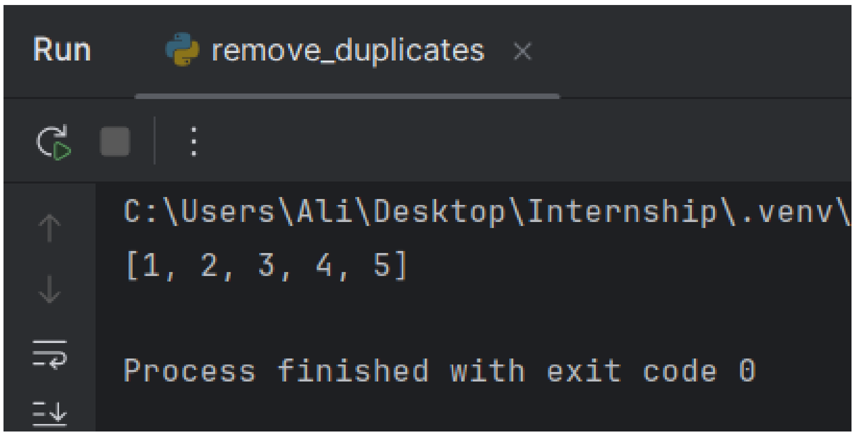
1. I created a file named remove_duplicates.py.
2. I defined a list items that contained some repeated elements.
3. I used dict.fromkeys() to eliminate duplicates while keeping the original order.
4. I converted the result back to a list and printed it.

Code Snippet:



```
items = [1, 2, 2, 3, 4, 4, 5]
unique_items = list(dict.fromkeys(items))
print(unique_items)
```

Output Snippet:



```
C:\Users\Ali\Desktop\Internship\.venv\
[1, 2, 3, 4, 5]

Process finished with exit code 0
```

Learning and Challenges

- I learned that dict.fromkeys(sequence) creates keys from the sequence, automatically dropping duplicates.
- Converting that dict's keys back to a list gives me a duplicate-free list in the original

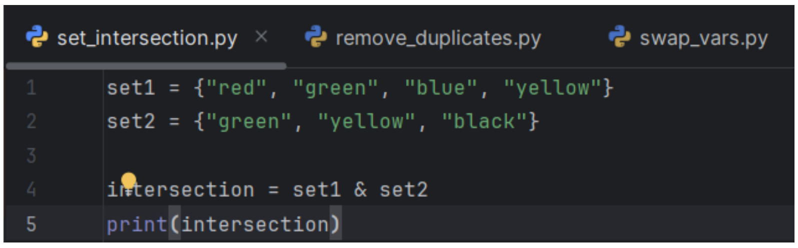
<u>Task 06:</u>

Find Intersection of Two Sets.

Step-by-Step Instructions

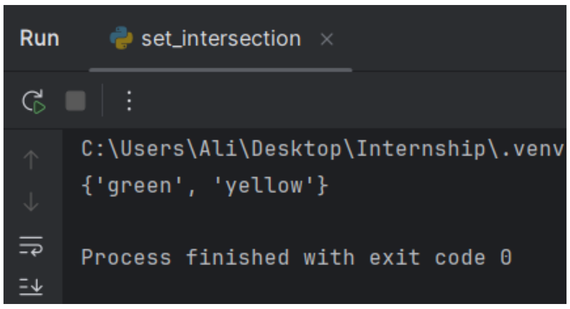
1. I created a file named set_intersection.py.
2. I defined two sets with some overlapping elements.
3. I used the & operator (or set1.intersection(set2)) to get their common items.
4. I printed the resulting intersection set.

Code Snippet:



```python
set1 = {"red", "green", "blue", "yellow"}
set2 = {"green", "yellow", "black"}

intersection = set1 & set2
print(intersection)
```

Output Snippet:



```
C:\Users\Ali\Desktop\Internship\.venv
{'green', 'yellow'}

Process finished with exit code 0
```

Learning and Challenges

- I learned that using & between two sets returns a new set of their shared elements.
- This operation is fast and concise for finding common data in Python sets.

Tasks – Dictionaries

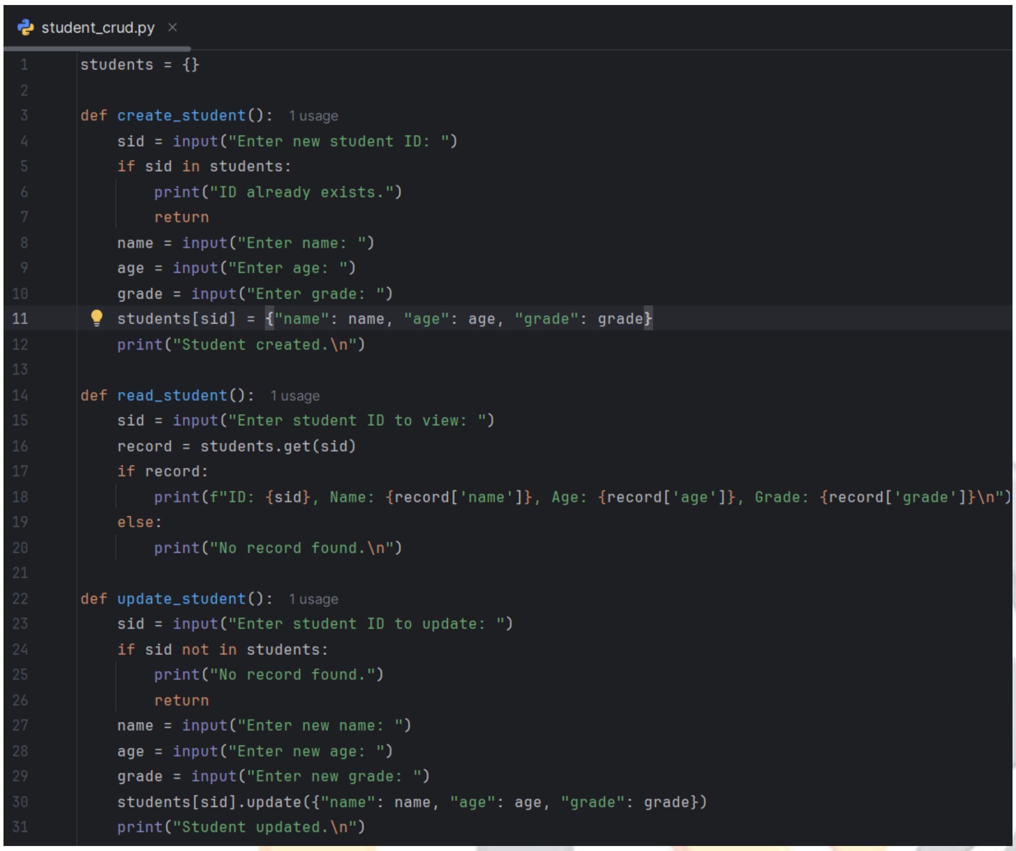
Task 07:

Read three numbers: output avg.

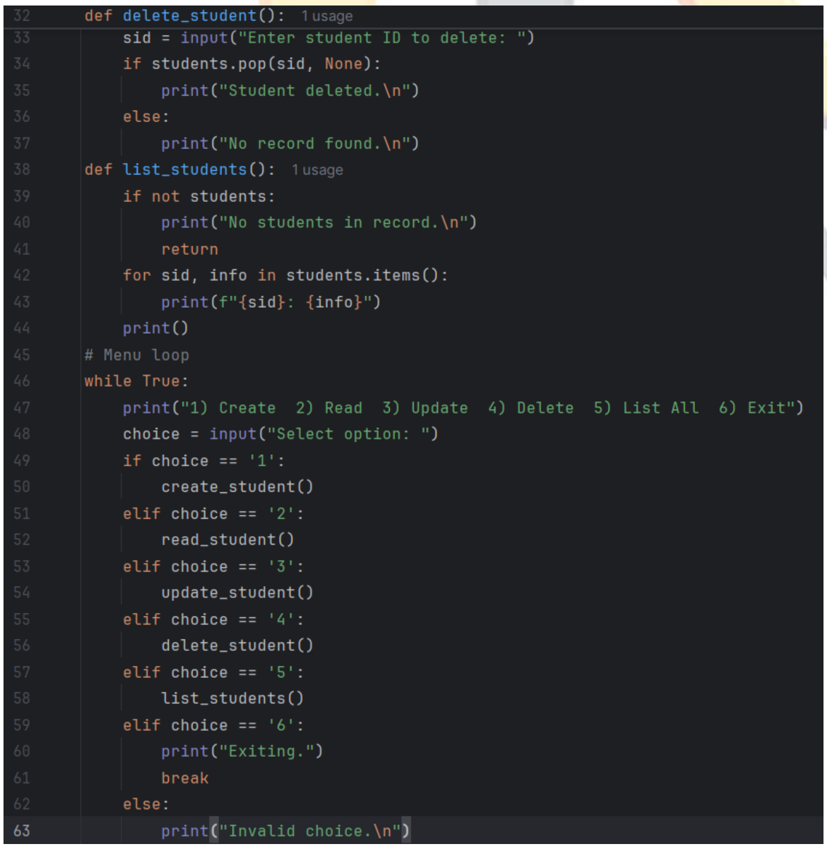
Step-by-Step Instructions

1. I created a file named student_crud.py.
2. I initialized an empty dictionary students = {} where each key is a student ID and each value is another dictionary holding that student's details.
3. I wrote functions to **Create**, **Read**, **Update**, and **Delete** records by manipulating students.
4. I built a while True menu loop to let the user choose an action until they exit.

Code Snippet:

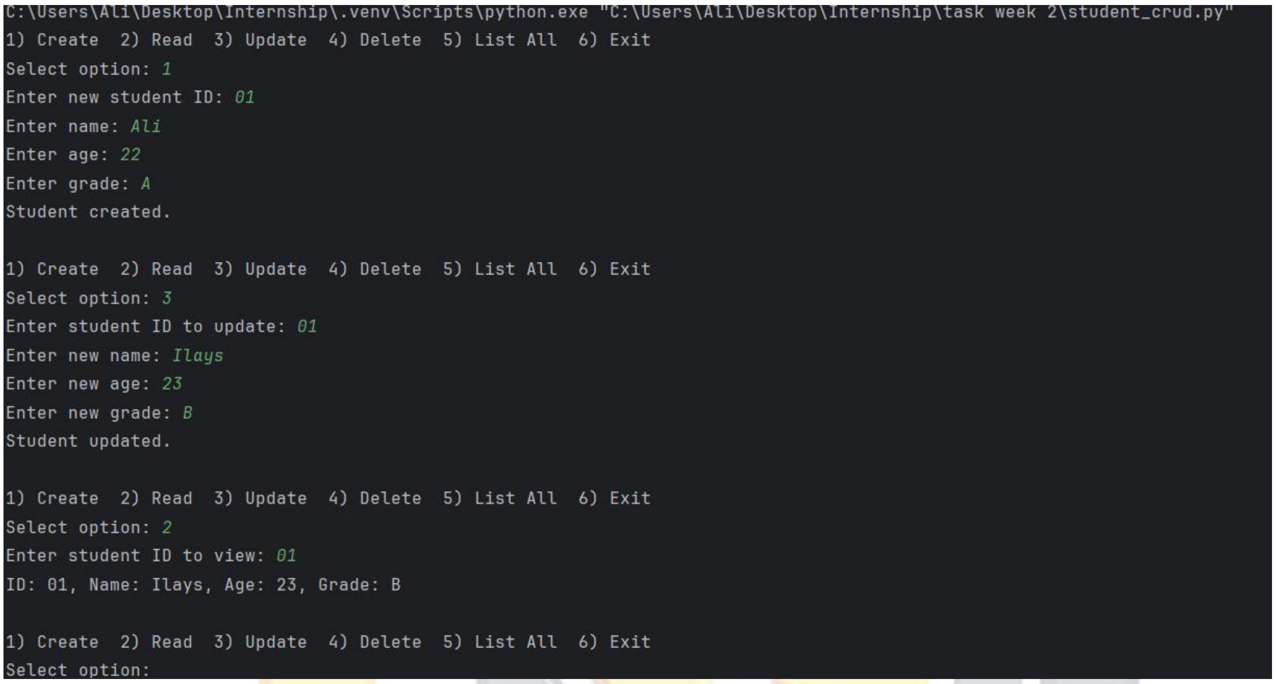


```python
students = {}

def create_student():
    sid = input("Enter new student ID: ")
    if sid in students:
        print("ID already exists.")
        return
    name = input("Enter name: ")
    age = input("Enter age: ")
    grade = input("Enter grade: ")
    students[sid] = {"name": name, "age": age, "grade": grade}
    print("Student created.\n")

def read_student():
    sid = input("Enter student ID to view: ")
    record = students.get(sid)
    if record:
        print(f"ID: {sid}, Name: {record['name']}, Age: {record['age']}, Grade: {record['grade']}\n")
    else:
        print("No record found.\n")

def update_student():
    sid = input("Enter student ID to update: ")
    if sid not in students:
        print("No record found.")
        return
    name = input("Enter new name: ")
    age = input("Enter new age: ")
    grade = input("Enter new grade: ")
    students[sid].update({"name": name, "age": age, "grade": grade})
    print("Student updated.\n")
```



```python
def delete_student():
    sid = input("Enter student ID to delete: ")
    if students.pop(sid, None):
        print("Student deleted.\n")
    else:
        print("No record found.\n")
def list_students():
    if not students:
        print("No students in record.\n")
        return
    for sid, info in students.items():
        print(f"{sid}: {info}")
    print()
# Menu loop
while True:
    print("1) Create  2) Read  3) Update  4) Delete  5) List All  6) Exit")
    choice = input("Select option: ")
    if choice == '1':
        create_student()
    elif choice == '2':
        read_student()
    elif choice == '3':
        update_student()
    elif choice == '4':
        delete_student()
    elif choice == '5':
        list_students()
    elif choice == '6':
        print("Exiting.")
        break
    else:
        print("Invalid choice.\n")
```

Output Snippet:



```
C:\Users\Ali\Desktop\Internship\.venv\Scripts\python.exe "C:\Users\Ali\Desktop\Internship\task week 2\student_crud.py"
1) Create  2) Read  3) Update  4) Delete  5) List All  6) Exit
Select option: 1
Enter new student ID: 01
Enter name: Ali
Enter age: 22
Enter grade: A
Student created.

1) Create  2) Read  3) Update  4) Delete  5) List All  6) Exit
Select option: 3
Enter student ID to update: 01
Enter new name: Ilays
Enter new age: 23
Enter new grade: B
Student updated.

1) Create  2) Read  3) Update  4) Delete  5) List All  6) Exit
Select option: 2
Enter student ID to view: 01
ID: 01, Name: Ilays, Age: 23, Grade: B

1) Create  2) Read  3) Update  4) Delete  5) List All  6) Exit
Select option:
```

Learning and Challenges

- I learned how to **structure a CRUD interface** using a dictionary of dictionaries.
- I practiced **menu-driven loops** and **function-based organization**, improving code readability and reuse.

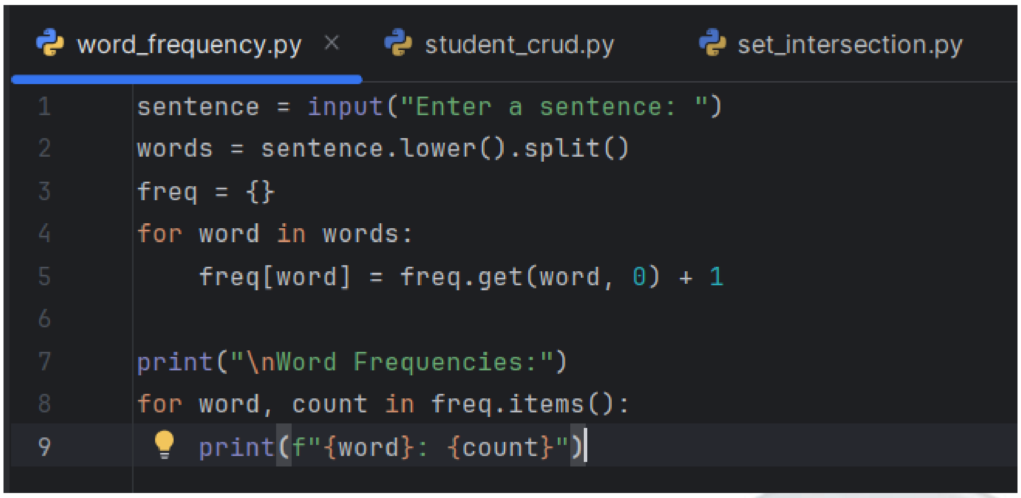
Task 08:

Count Word Frequency in a Sentence.

Step-by-Step Instructions

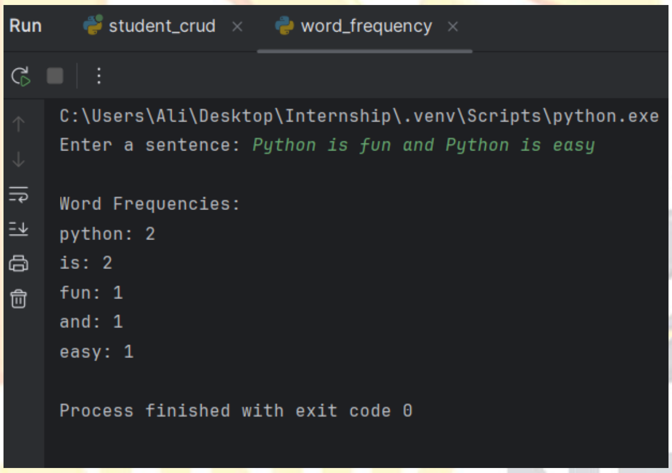
1. I created a file named word_frequency.py.
2. I used input() to read a full sentence from the user.
3. I split the sentence into words (by spaces) and normalized them to lowercase.
4. I used a dictionary to count how many times each word appears.
5. I printed each word alongside its frequency.

Code Snippet:



```python
sentence = input("Enter a sentence: ")
words = sentence.lower().split()
freq = {}
for word in words:
    freq[word] = freq.get(word, 0) + 1

print("\nWord Frequencies:")
for word, count in freq.items():
    print(f"{word}: {count}")
```

Output Snippet:



```
C:\Users\Ali\Desktop\Internship\.venv\Scripts\python.exe
Enter a sentence: Python is fun and Python is easy

Word Frequencies:
python: 2
is: 2
fun: 1
and: 1
easy: 1

Process finished with exit code 0
```

Learning and Challenges

- I learned to normalize text (lowercase) and split it into tokens with split().
- I practiced using a dictionary and get() to accumulate counts for each unique word.

Tasks – Functions

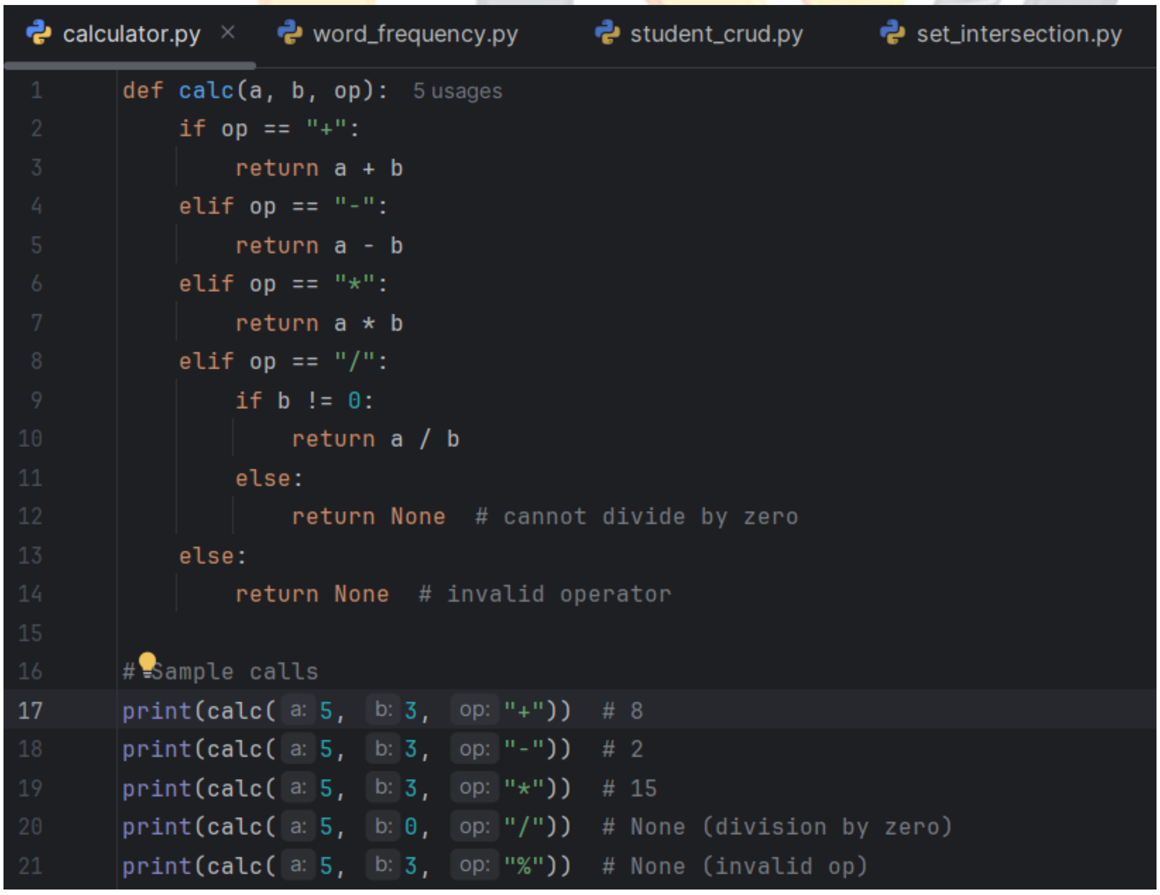
Task 09:

Write calc(a, b, op).

Step-by-Step Instructions

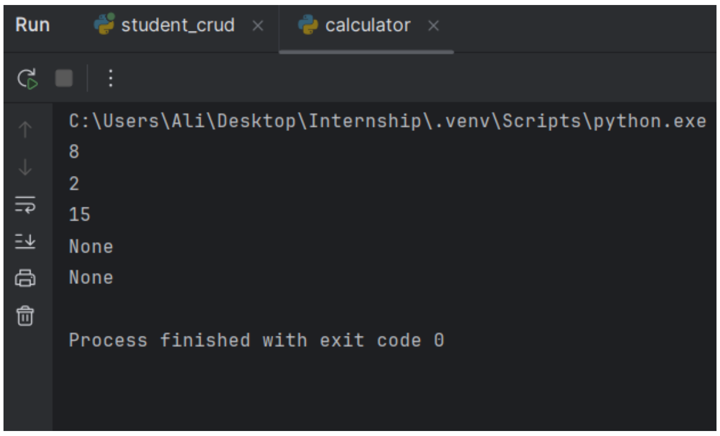
1. I created a file named calculator.py.
2. I defined a function calc(a, b, op) with three parameters.
3. Inside the function, I used if/elif to check if op is "+", "-", "*", or "/".
4. I computed and returned the appropriate result.
5. I added an else branch to handle invalid operators by returning None.
6. I called calc with sample values and printed the result.

Code Snippet:



```python
def calc(a, b, op):
    if op == "+":
        return a + b
    elif op == "-":
        return a - b
    elif op == "*":
        return a * b
    elif op == "/":
        if b != 0:
            return a / b
        else:
            return None  # cannot divide by zero
    else:
        return None  # invalid operator

# Sample calls
print(calc( a: 5,  b: 3,  op: "+"))  # 8
print(calc( a: 5,  b: 3,  op: "-"))  # 2
print(calc( a: 5,  b: 3,  op: "*"))  # 15
print(calc( a: 5,  b: 0,  op: "/"))  # None (division by zero)
print(calc( a: 5,  b: 3,  op: "%"))  # None (invalid op)
```

Output Snippet:



```
C:\Users\Ali\Desktop\Internship\.venv\Scripts\python.exe
8
2
15
None
None

Process finished with exit code 0
```

Learning and Challenges

- I learned how to write a **single function** that handles multiple operations via parameters.
- I practiced using **conditional branches** to select behavior and **edge checks** (like division by zero).

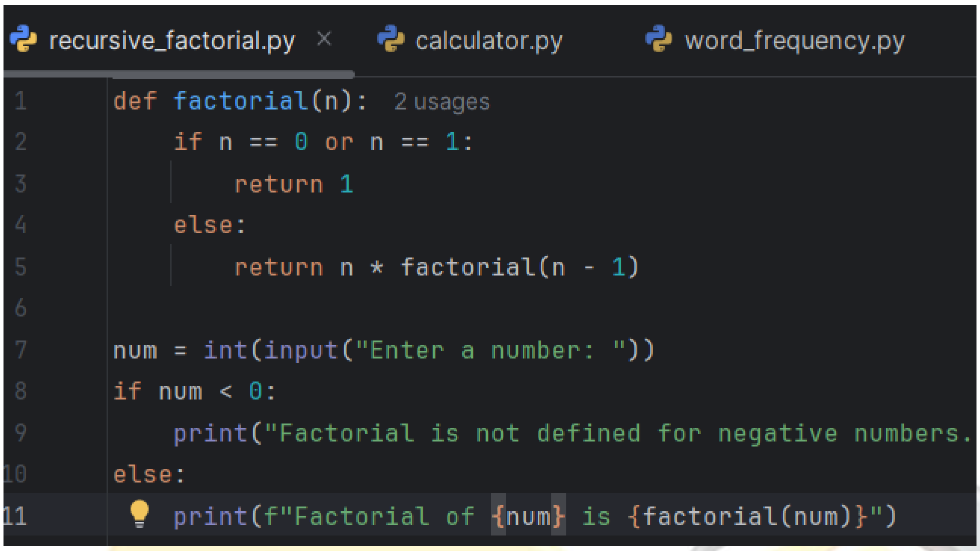
Task 10:

Write factorial(n) Using Recursion

Step-by-Step Instructions

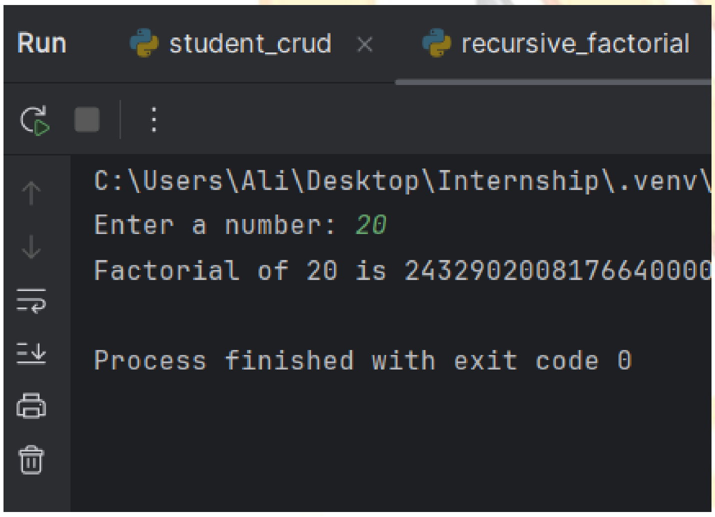
1. I created a file named recursive_factorial.py.
2. I defined a function factorial(n) that calls itself recursively.
3. If n is 0 or 1, it returns 1 (base case).
4. Otherwise, it returns n * factorial(n - 1) (recursive case).
5. I used input() to ask the user for a number and printed the factorial result.

Code Snippet:



```python
def factorial(n):
    if n == 0 or n == 1:
        return 1
    else:
        return n * factorial(n - 1)

num = int(input("Enter a number: "))
if num < 0:
    print("Factorial is not defined for negative numbers.
else:
    print(f"Factorial of {num} is {factorial(num)}")
```

Output Snippet:



```
C:\Users\Ali\Desktop\Internship\.venv\
Enter a number: 20
Factorial of 20 is 243290200817664000

Process finished with exit code 0
```

Learning and Challenges

- I learned how **recursion** works calling the same function from inside itself.
- I practiced writing a **base case** and handling edge conditions like negative input.

Tasks – Modules

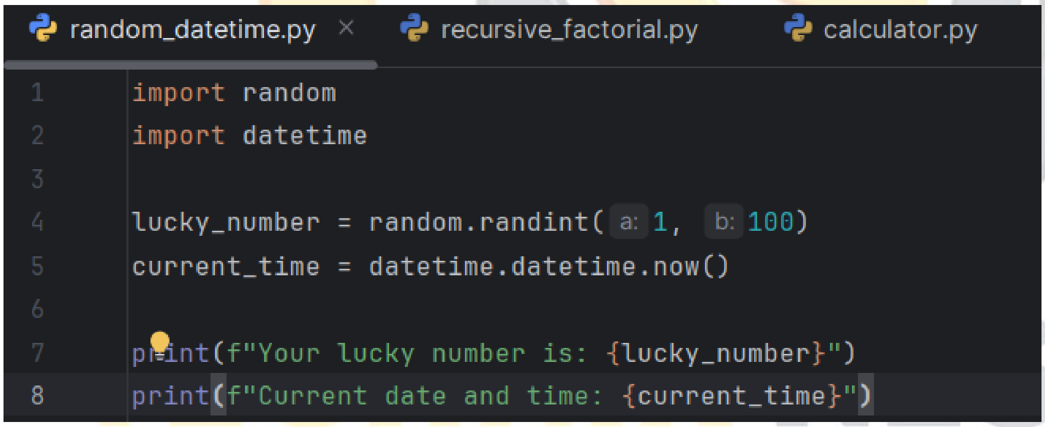
Task 11:

Use random & datetime in Script.

Step-by-Step Instructions

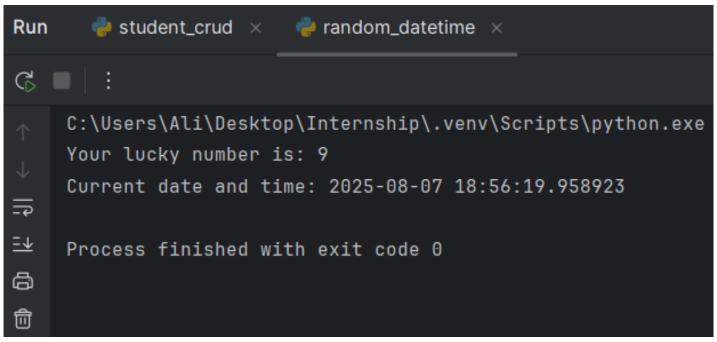
1. I created a file named random_datetime.py.
2. I imported the random and datetime modules.
3. I generated a random lucky number using random.randint().
4. I got the current date and time using datetime.datetime.now().
5. I printed both values in a simple message.

Code Snippet:



```python
import random
import datetime

lucky_number = random.randint( a: 1,  b: 100)
current_time = datetime.datetime.now()

print(f"Your lucky number is: {lucky_number}")
print(f"Current date and time: {current_time}")
```

Output Snippet:



```
C:\Users\Ali\Desktop\Internship\.venv\Scripts\python.exe
Your lucky number is: 9
Current date and time: 2025-08-07 18:56:19.958923

Process finished with exit code 0
```

Learning and Challenges

- I learned how to generate **random values** with the random module.
- I practiced using datetime.now() to display the **current date and time** in Python.

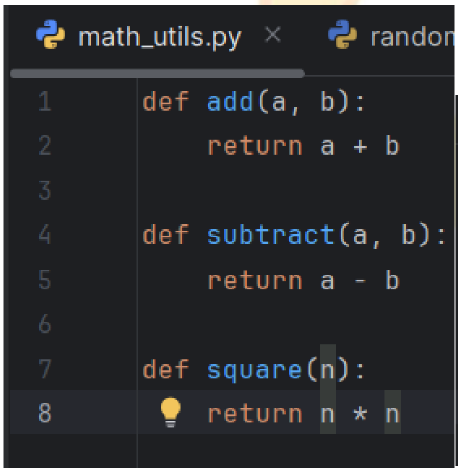
Task 12:

Create math_utils Module & Import It.

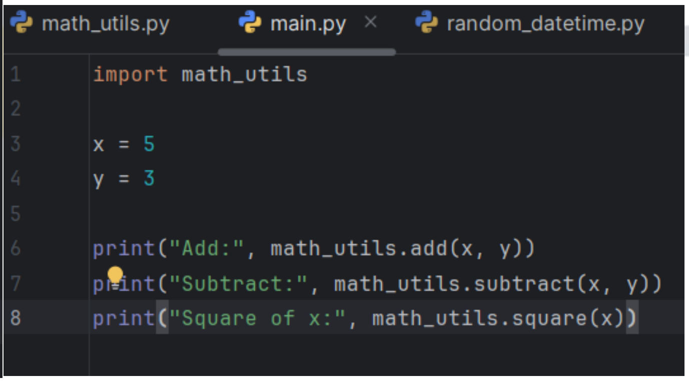
Step-by-Step Instructions

1. I created a file named math_utils.py which works like a module.
2. Inside math_utils.py, I defined some math functions like add, subtract, and square.
3. I created another file named main.py where I **imported** this module.
4. I called the functions from math_utils using the module_name.function_name() format.

Code Snippet:

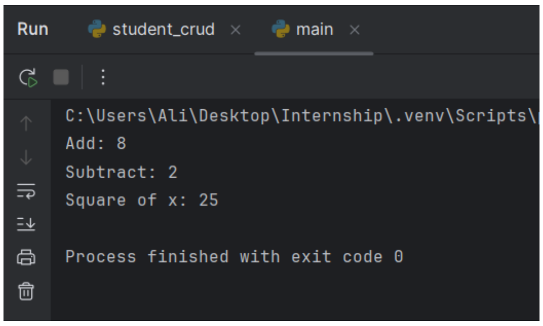


```python
def add(a, b):
    return a + b

def subtract(a, b):
    return a - b

def square(n):
    return n * n
```



```python
import math_utils

x = 5
y = 3

print("Add:", math_utils.add(x, y))
print("Subtract:", math_utils.subtract(x, y))
print("Square of x:", math_utils.square(x))
```

Output Snippet:



```
C:\Users\Ali\Desktop\Internship\.venv\Scripts\
Add: 8
Subtract: 2
Square of x: 25

Process finished with exit code 0
```

Learning and Challenges

- I learned how to **create reusable Python modules** using .py files.
- I practiced **importing custom modules** into other scripts using import and calling their functions.

<u>Tasks – Exceptions</u>

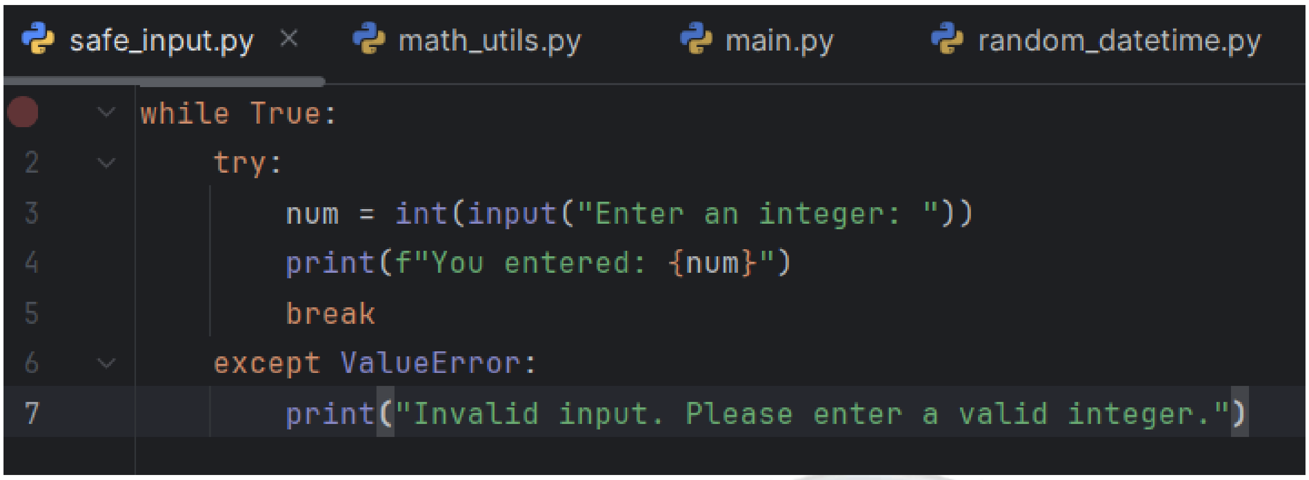
<u>Task 13:</u>

Grade calculator.

Step-by-Step Instructions

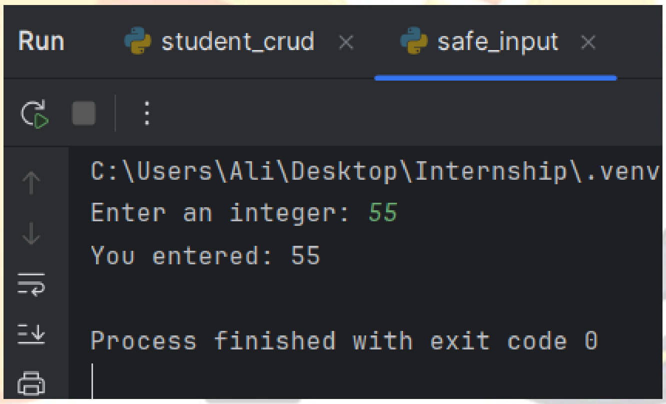
1. I created a file named safe_input.py.
2. I used a while True loop to ask the user to enter an integer.
3. I used a try-except block to catch ValueError if the input was not a valid number.
4. If input was valid, I printed it and broke the loop.
5. Otherwise, I displayed an error message and asked again.

Code Snippet:



```python
while True:
    try:
        num = int(input("Enter an integer: "))
        print(f"You entered: {num}")
        break
    except ValueError:
        print("Invalid input. Please enter a valid integer.")
```

Output Snippet:



```
C:\Users\Ali\Desktop\Internship\.venv
Enter an integer: 55
You entered: 55

Process finished with exit code 0
```

Learning and Challenges

- I learned how to **handle invalid input** safely using try-except.
- This loop ensures the program doesn't crash and keeps asking until valid input is given.

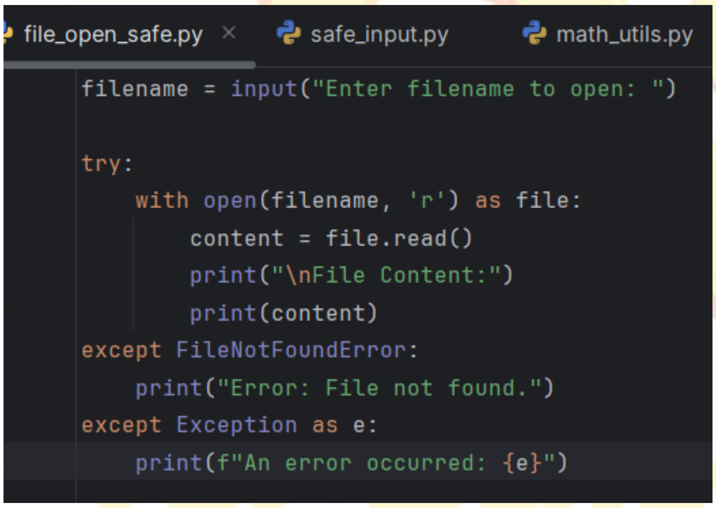
Task 14:

Open File with Error Message

Step-by-Step Instructions

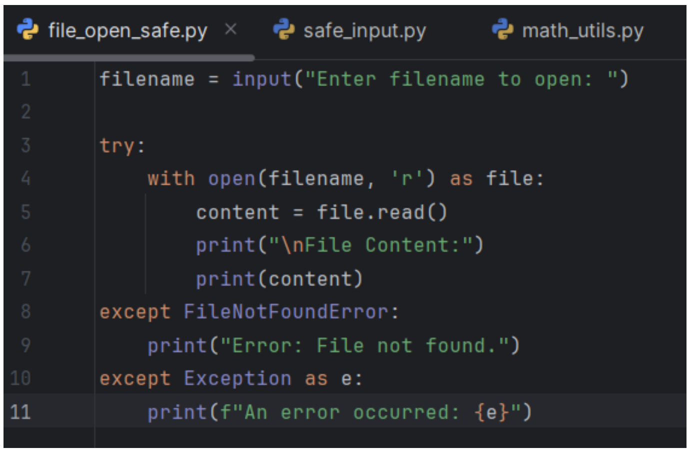
1. I created a file named file_open_safe.py.
2. I asked the user to enter a file name using input().
3. I used a try-except block to attempt opening the file in read mode.
4. If the file doesn’t exist or fails to open, an error message is shown.
5. If successful, the file content is printed.

Code Snippet:



```python
filename = input("Enter filename to open: ")

try:
    with open(filename, 'r') as file:
        content = file.read()
        print("\nFile Content:")
        print(content)
except FileNotFoundError:
    print("Error: File not found.")
except Exception as e:
    print(f"An error occurred: {e}")
```

Output Snippet:



```python
filename = input("Enter filename to open: ")

try:
    with open(filename, 'r') as file:
        content = file.read()
        print("\nFile Content:")
        print(content)
except FileNotFoundError:
    print("Error: File not found.")
except Exception as e:
    print(f"An error occurred: {e}")
```

Learning and Challenges

- I learned how to use try-except to **safely handle file errors**.
- I practiced using with open() to read a file without risking a crash if the file doesn't exist.

<u>Tasks – Loops</u>

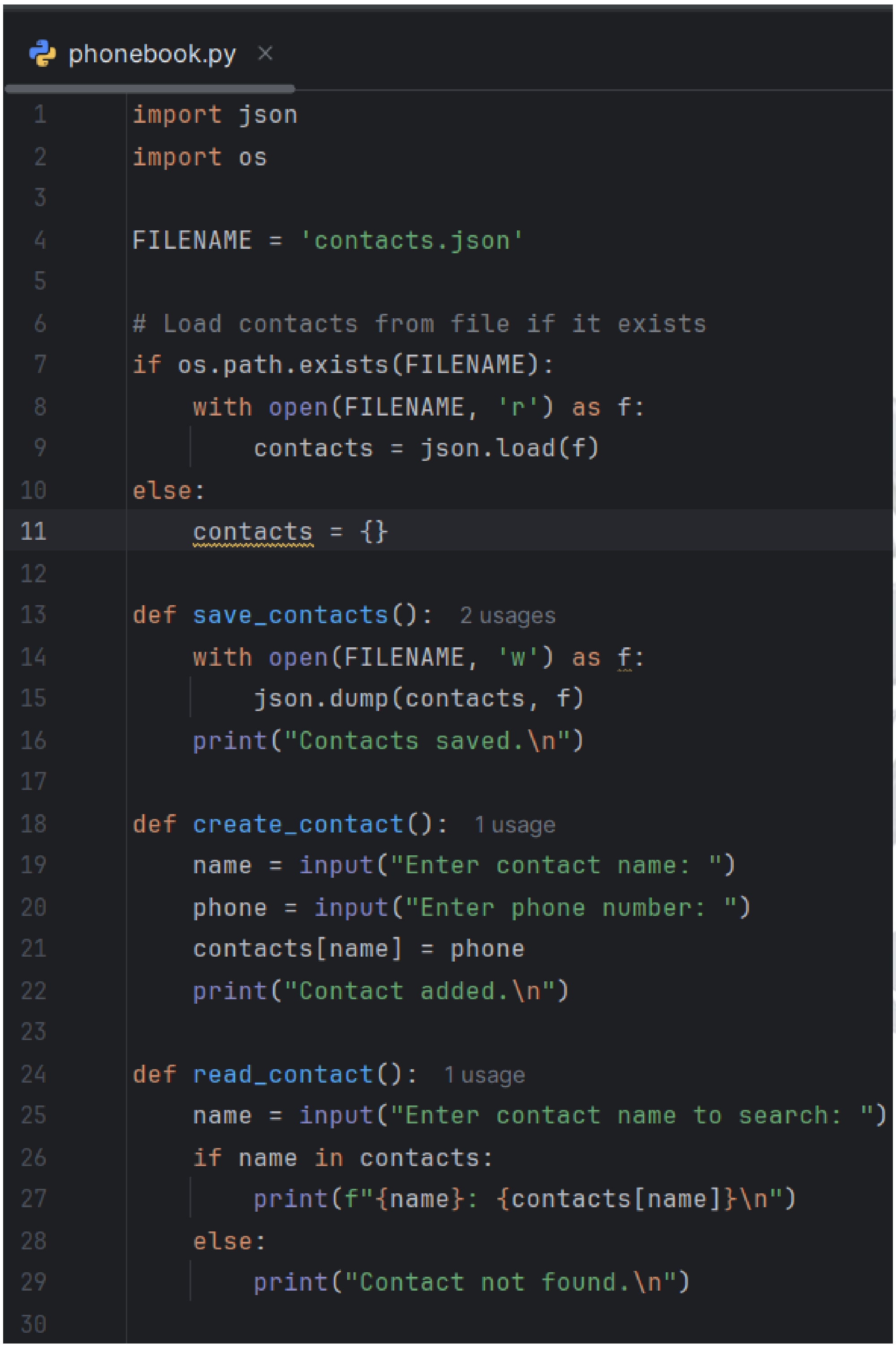
<u>Task 15:</u>

Phonebook App – CRUD Contacts with JSON File Storage.

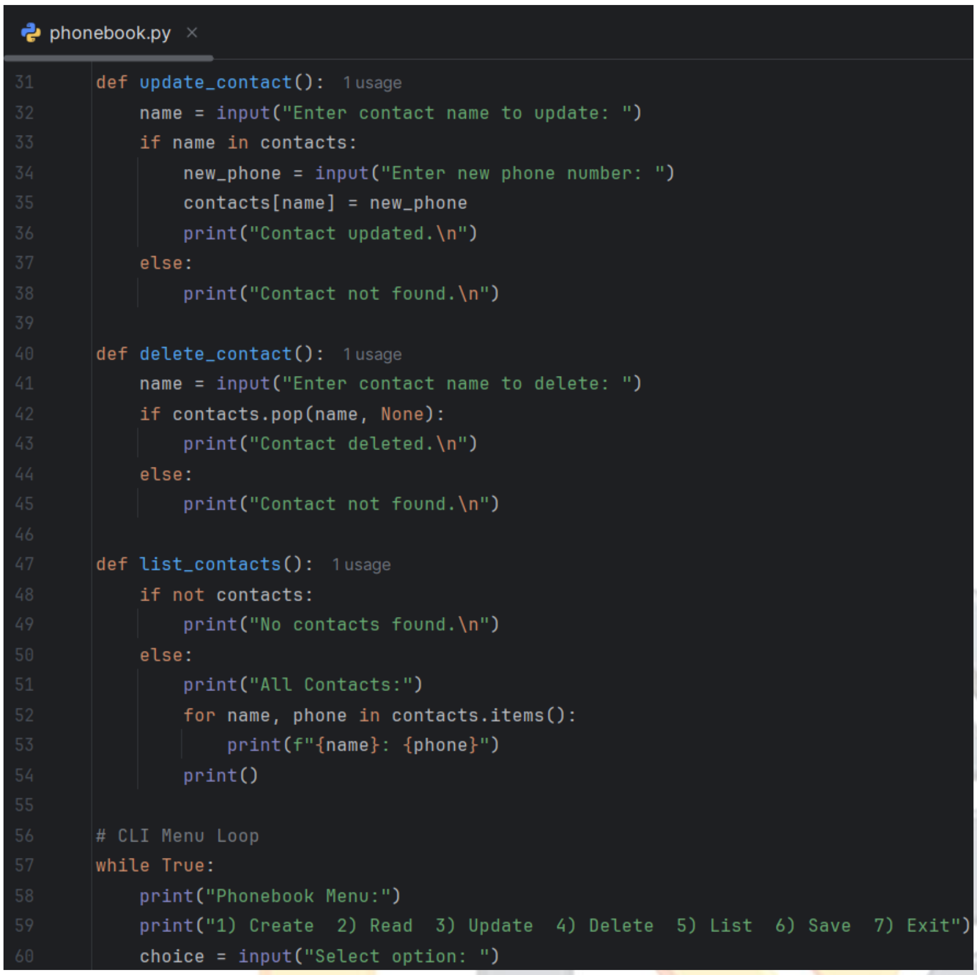
Step-by-Step Instructions

1. I created a file phonebook.py.
2. I used a dictionary to store contacts (name: phone_number).
3. I used the json module to save/load the contacts to/from a .json file.
4. I added options to **create**, **read**, **update**, **delete**, and **save** contacts.
5. All data is saved in a file called contacts.json for permanent storage.

Code Snippet:

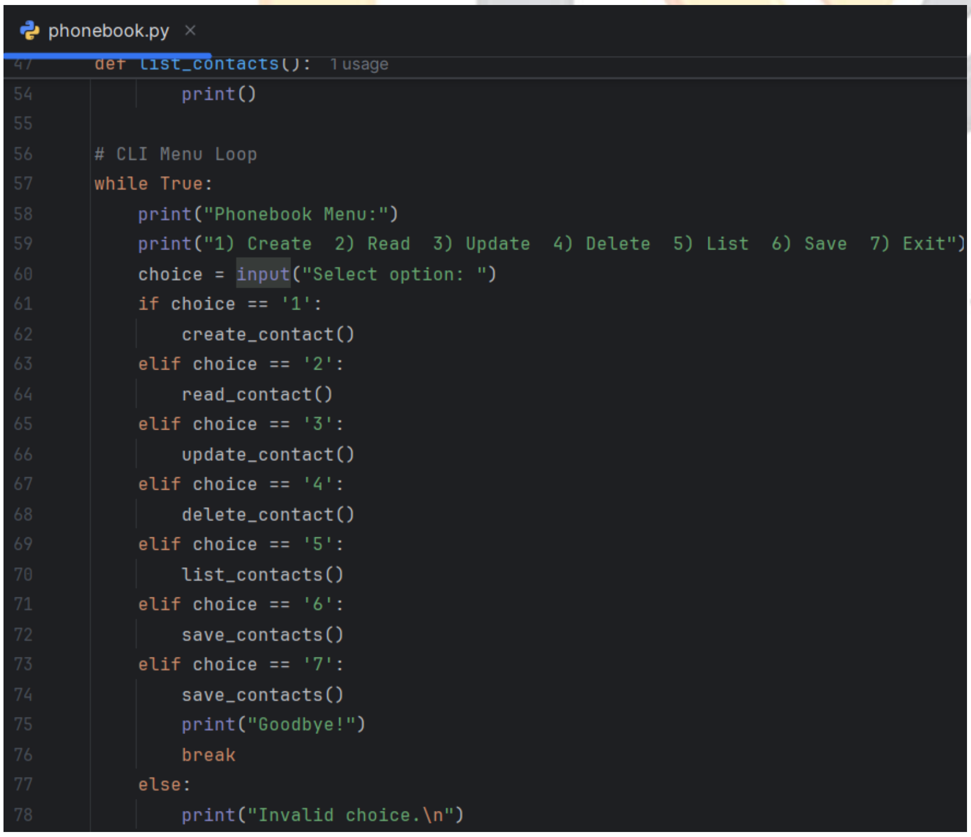

phonebook.py

```
import json
import os

FILENAME = 'contacts.json'

# Load contacts from file if it exists
if os.path.exists(FILENAME):
    with open(FILENAME, 'r') as f:
        contacts = json.load(f)
else:
    contacts = {}

def save_contacts():  2 usages
    with open(FILENAME, 'w') as f:
        json.dump(contacts, f)
    print("Contacts saved.\n")

def create_contact():  1 usage
    name = input("Enter contact name: ")
    phone = input("Enter phone number: ")
    contacts[name] = phone
    print("Contact added.\n")

def read_contact():  1 usage
    name = input("Enter contact name to search: ")
    if name in contacts:
        print(f"{name}: {contacts[name]}\n")
    else:
        print("Contact not found.\n")

```



phonebook.py

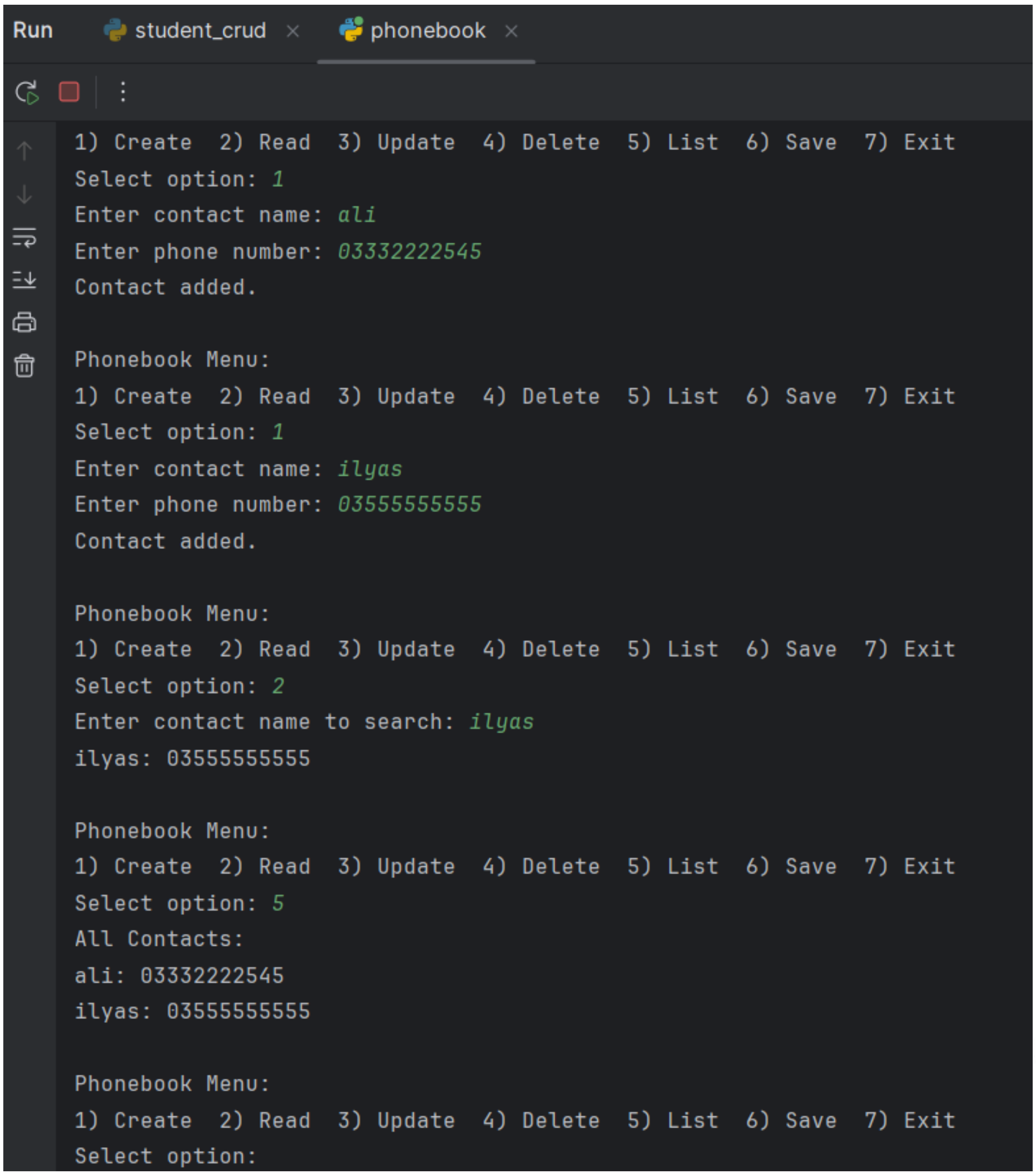
```python
def update_contact():  1 usage
    name = input("Enter contact name to update: ")
    if name in contacts:
        new_phone = input("Enter new phone number: ")
        contacts[name] = new_phone
        print("Contact updated.\n")
    else:
        print("Contact not found.\n")

def delete_contact():  1 usage
    name = input("Enter contact name to delete: ")
    if contacts.pop(name, None):
        print("Contact deleted.\n")
    else:
        print("Contact not found.\n")

def list_contacts():  1 usage
    if not contacts:
        print("No contacts found.\n")
    else:
        print("All Contacts:")
        for name, phone in contacts.items():
            print(f"{name}: {phone}")
        print()

# CLI Menu Loop
while True:
    print("Phonebook Menu:")
    print("1) Create  2) Read  3) Update  4) Delete  5) List  6) Save  7) Exit")
    choice = input("Select option: ")
```



phonebook.py

```python
def list_contacts():  1 usage
        print()

# CLI Menu Loop
while True:
    print("Phonebook Menu:")
    print("1) Create  2) Read  3) Update  4) Delete  5) List  6) Save  7) Exit")
    choice = input("Select option: ")
    if choice == '1':
        create_contact()
    elif choice == '2':
        read_contact()
    elif choice == '3':
        update_contact()
    elif choice == '4':
        delete_contact()
    elif choice == '5':
        list_contacts()
    elif choice == '6':
        save_contacts()
    elif choice == '7':
        save_contacts()
        print("Goodbye!")
        break
    else:
        print("Invalid choice.\n")
```

Output Snippet:



```
1) Create  2) Read  3) Update  4) Delete  5) List  6) Save  7) Exit
Select option: 1
Enter contact name: ali
Enter phone number: 03332222545
Contact added.

Phonebook Menu:
1) Create  2) Read  3) Update  4) Delete  5) List  6) Save  7) Exit
Select option: 1
Enter contact name: ilyas
Enter phone number: 03555555555
Contact added.

Phonebook Menu:
1) Create  2) Read  3) Update  4) Delete  5) List  6) Save  7) Exit
Select option: 2
Enter contact name to search: ilyas
ilyas: 03555555555

Phonebook Menu:
1) Create  2) Read  3) Update  4) Delete  5) List  6) Save  7) Exit
Select option: 5
All Contacts:
ali: 03332222545
ilyas: 03555555555

Phonebook Menu:
1) Create  2) Read  3) Update  4) Delete  5) List  6) Save  7) Exit
Select option:
```

Learning and Challenges

- I learned to use the **json module** for reading/writing data to a file.
- I practiced full **CRUD logic** in a text-based app using a dictionary and saved it for later use.