INTERNSHIP TRAINING REPORT

A report submitted in partial fulfilment of the requirements for the Award of Degree of

BACHELOR OF ENGINEERING

In

ELECTRONICS AND COMMUNICATION ENGINEERING

By

KISHOR KUMAR B

Regno:6176AC22UEC059

(Duration - 09.07.2024 to 23.07.2024)

Under the guidance of

Mr. Pradeep Susairaj

DEPARTMENT OF INFORMATION TECHNOLOGY

ADHIYAMAAN COLLEGE OF ENGINEERING

(An Autonomous Institution)

(Affiliated to Anna University, Chennai, Approved by AICTE, Accredited by NAAC with “A” Grade-UGC Accredited)

Dr MGR Nagar, Hosur, Krishnagiri District, TamilNadu - 635109

ACKNOWLEDGEMENT

I would like to thank everyone who supported and guided me during my internship. This experience has been invaluable, and I am truly grateful for your contributions to my growth and learning.

First, I deeply appreciate **Mrs. Chandrika,** HR for giving me this opportunity. Your mentorship, patience, and feedback have been crucial in developing my technical skills and understanding of web development, significantly impacting my career.

I am also thankful to the entire team at **ProcessDrive** for creating a welcoming and collaborative environment. Your expertise, willingness to share knowledge, and ongoing support have greatly enriched my internship experience and helped me grow both professionally and personally.

Special thanks to **Mr. Pradeep Susairaj** for his guidance and clear explanations of complex concepts. Your support has improved my coding skills and boosted my confidence in the field, leaving a lasting impression on my development.

Thank you all for making this internship a rewarding and memorable experience and for contributing to my professional growth and learning.

Sincerely,

KISHOR KUMAR B

ABSTRACT

Skills and Tools Acquired:

During my internship at **ProcessDrive**, I learned several important skills and tools. I became proficient in Python programming, used the Postman tool for testing and managing APIs, and developed the ability to make and handle HTTP requests with Python. I also gained a good understanding of APIs and how to work with them effectively.

Database Experience:

I learned about MongoDB and performed CRUD (Create, Read, Update, Delete) operations using Python. However, this knowledge was not applied in my project during the internship. I think it is usefull for my future projects.

Project Work:

A significant project I worked on involved the Gemini API. In this project, I focused on converting images to text and videos to text. I also developed a chat application using Python. I created features to interact with the API and handled data processing directly, without using a database.

Overall Impact:

Through this internship, I successfully created and tested API endpoints and developed Python scripts. This experience greatly enhanced my technical skills and provided valuable knowledge in API integration and project implementation, allowing me to contribute effectively to the team's goals.

EXECUTIVE SUMMARY

This report provides an overview of my internship at **ProcessDrive**, where I worked as an intern. During my tenure, I was involved in various Python-related projects. This experience enhanced my understanding of Python programming and improved my skills in working with APIs, MongoDB, and handling data.

INTRODUCTION

PURPOSE AND OBJECTIVE OF THIS INTERNSHIP:

The main goal of this internship was to get practical experience in Python programming. It focused on using what I learned in a real-world setting to improve my Python skills and understand how to work with APIs and databases. The internship also aimed to help me communicate and work well with a team.

IMPORTANCE OF THE INTERNSHIP:

This internship was crucial in bridging the gap between academic learning and real-world application, providing valuable insights into the workings of a leading corporation.

DESCRIPTION OF INTERNSHIP ACTIVITIES

TASKS AND RESPONSIBILITIES:

During my internship, I was assigned the following roles and responsibilities:

- **Python Programming**: Wrote and debugged Python code, developing scripts to meet requirements.
- **API Handling with Postman**: Tested APIs with Postman, ensuring proper functionality and response analysis.
- **API Handling in Python**: Made HTTP requests with Python, integrating and processing API data.
- **MongoDB CRUD Operations**: Managed MongoDB data with Create, Read, Update, and Delete operations.
- **Pattern Programs with Loops**: Created and debugged pattern programs using loops.
- Update Project Documentation & Reports.

TOOLS AND TECHNOLOGIES USED:

- **Python**: Easy-to-learn programming language for many types of projects.
- **Postman**: Tool to test and manage APIs easily.
- **MongoDB**: Database for storing and managing data in a flexible way.
- **Visual Studio Code**: As the primary code editor.

LEARNING OUTCOMES:

- Learned to write code and automate tasks using Python.
- Improved skills in handling lists and dictionaries in Python.
- Tested and managed APIs using Postman.
- Tested and managed APIs using Python.
- Stored and managed data using MongoDB.

SKILLS AND KNOWLEDGE GAINED

Technical Skills:

- Proficiency in Python programming.
- Experience with APIs.
- Knowledge of databases.

Soft Skills:

- Improved problem-solving skills by debugging, fixing issues, and optimizing code.
- Learned to prioritize tasks, manage time, and meet deadlines.

Industry Knowledge:

- Project management.
- Learned industry standards for Python programming, including coding conventions, security practices, and performance optimization techniques.

TASK PERFORMED

All the tasks performed during the internship program were based on Python Programming. The trainer had assigned a few tasks which would prove to be quintessential for industry standards and understanding the technology easily.

Python Programming:

Python programming involves writing code in Python to create software and applications.

Loop:

A loop is a programming construct that repeats a block of code multiple times. It continues until a specified condition is met. Loops are used to automate repetitive tasks.

For Loop:

A for loop is a programming construct that repeats a block of code for each item in a sequence, like a list or range.

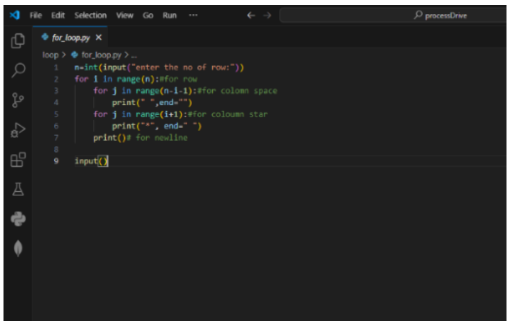
While Loop:

A while loop is a programming construct that repeats a block of code as long as a specified condition is true. It stops when the condition becomes false.

PATTERN Programming:

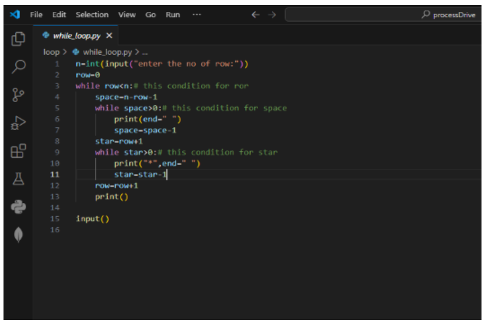
- Created pattern programs using for loops.
- Created pattern programs using while loops.
- Practiced loop structures in Python.
- Enhanced problem-solving skills with pattern exercises.

Sample program (For loop):



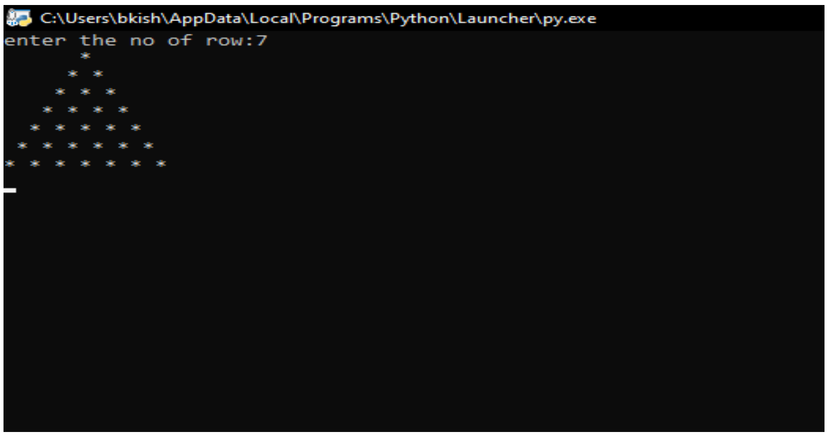
```python
n=int(input("enter the no of row:"))
for i in range(n):#for row
    for j in range(n-i-1):#for colomn space
        print(" ",end="")
    for j in range(i+1):#for coloumn star
        print("*", end=" ")
    print()# for newline

input()
```

Sample program (While loop):



```python
n=int(input("enter the no of row:"))
row=0
while row<n:# this condition for ror
    space=n-row-1
    while space>0:# this condition for space
        print(end=" ")
        space=space-1
    star=row+1
    while star>0:# this condition for star
        print("*",end=" ")
        star=star-1
    row=row+1
    print()

input()
```

Output:



C:\Users\bkish\AppData\Local\Programs\Python\Launcher\py.exe

```
enter the no of row:7
      *
     * *
    * * *
   * * * *
  * * * * *
 * * * * * *
* * * * * * *
_
```

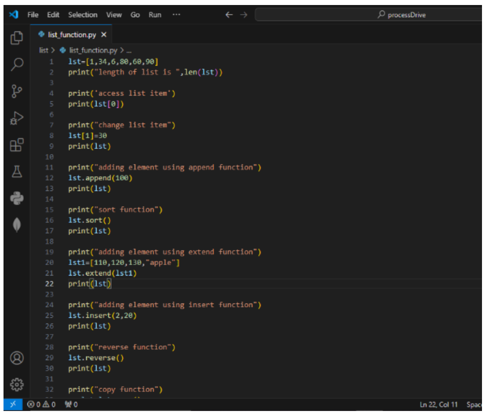
List:

A list is a data structure in Python that holds an ordered collection of items. Lists are mutable, so their contents can be changed.

List Methods:

- Used built-in functions to manipulate lists.
- Applied list methods for data handling and processing.
- Implemented functions like **append ()**, **remove ()**, and **sort ().**
- Managed list data efficiently for various tasks.
- Enhanced list operations with functions like **len ()** and **reverse ()**.

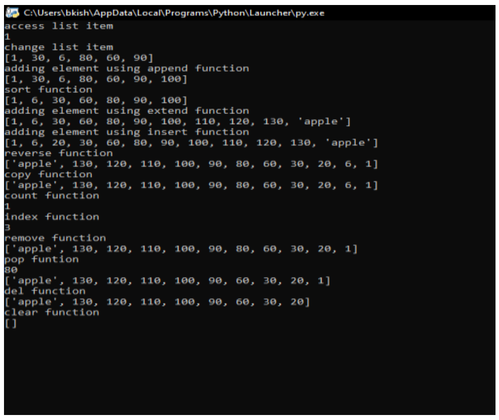
Sample Program:



File Edit Selection View Go Run ···
processDrive
list_function.py
list > list_function.py > ...
Ln 22, Col 11

```
lst=[1,34,6,80,60,90]
print("length of list is ",len(lst))

print('access list item')
print(lst[0])

print("change list item")
lst[1]=30
print(lst)

print("adding element using append function")
lst.append(100)
print(lst)

print("sort function")
lst.sort()
print(lst)

print("adding element using extend function")
lst1=[110,120,130,"apple"]
lst.extend(lst1)
print(lst)

print("adding element using insert function")
lst.insert(2,20)
print(lst)

print("reverse function")
lst.reverse()
print(lst)

print("copy function")
```

Output:



C:\Users\bkish\AppData\Local\Programs\Python\Launcher\py.exe

```
access list item
1
change list item
[1, 30, 6, 80, 60, 90]
adding element using append function
[1, 30, 6, 80, 60, 90, 100]
sort function
[1, 6, 30, 60, 80, 90, 100]
adding element using extend function
[1, 6, 30, 60, 80, 90, 100, 110, 120, 130, 'apple']
adding element using insert function
[1, 6, 20, 30, 60, 80, 90, 100, 110, 120, 130, 'apple']
reverse function
['apple', 130, 120, 110, 100, 90, 80, 60, 30, 20, 6, 1]
copy function
['apple', 130, 120, 110, 100, 90, 80, 60, 30, 20, 6, 1]
count function
1
index function
3
remove function
['apple', 130, 120, 110, 100, 90, 80, 60, 30, 20, 1]
pop funtion
80
['apple', 130, 120, 110, 100, 90, 60, 30, 20, 1]
del function
['apple', 130, 120, 110, 100, 90, 60, 30, 20]
clear function
[]
```

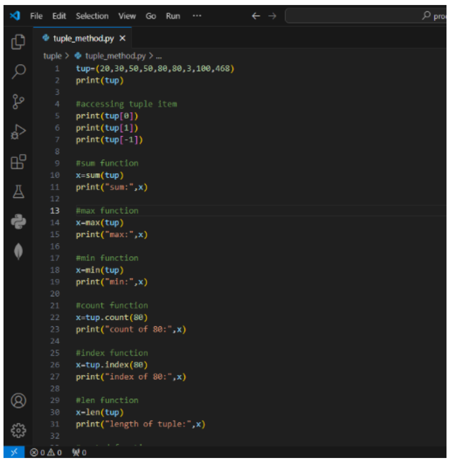
Tuple:

A tuple is an ordered collection of items in Python. Tuples are immutable, so their contents cannot be changed after creation.

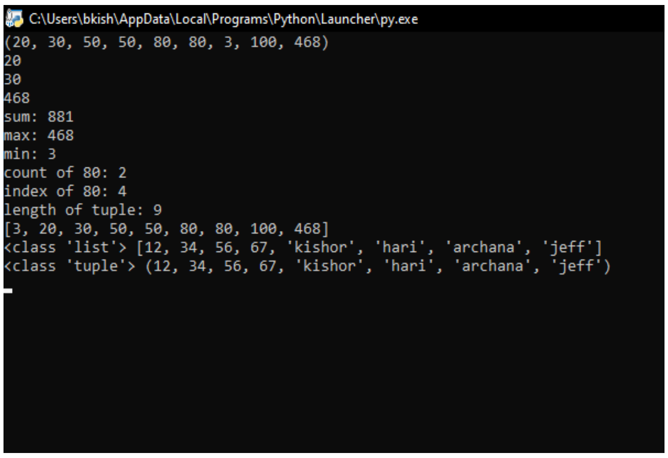
Tuple Methods:

- Used built-in functions to work with tuples.
- Applied tuple methods for data handling.
- Implemented functions like **len ()** and **count ()**.
- Accessed and retrieved tuple elements efficiently.
- Managed tuple data with functions like **index ()**.

Sample Program: Output:



```python
tup=(20,30,50,50,80,80,3,100,468)
print(tup)

#accessing tuple item
print(tup[0])
print(tup[1])
print(tup[-1])

#sum function
x=sum(tup)
print("sum:",x)

#max function
x=max(tup)
print("max:",x)

#min function
x=min(tup)
print("min:",x)

#count function
x=tup.count(80)
print("count of 80:",x)

#index function
x=tup.index(80)
print("index of 80:",x)

#len function
x=len(tup)
print("length of tuple:",x)

```


C:\Users\bkish\AppData\Local\Programs\Python\Launcher\py.exe

```
(20, 30, 50, 50, 80, 80, 3, 100, 468)
20
30
468
sum: 881
max: 468
min: 3
count of 80: 2
index of 80: 4
length of tuple: 9
[3, 20, 30, 50, 50, 80, 80, 100, 468]
<class 'list'> [12, 34, 56, 67, 'kishor', 'hari', 'archana', 'jeff']
<class 'tuple'> (12, 34, 56, 67, 'kishor', 'hari', 'archana', 'jeff')
```

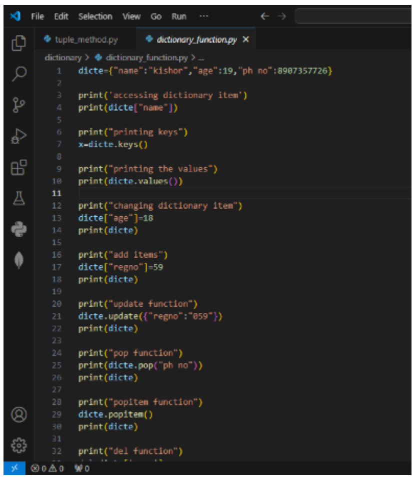
Dictionary:

A dictionary is a data structure in Python that stores key-value pairs. Each key is unique and maps to a value.

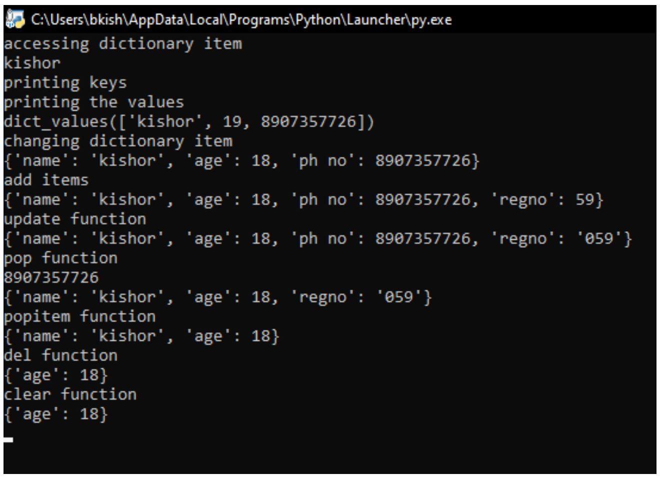
Dictionary Methods:

- Used built-in functions to manage dictionaries.
- Applied dictionary methods for data handling.
- Implemented functions like **keys ()**, **values ()**, and **items ()**.
- Accessed and updated dictionary elements efficiently.
- Managed data with functions like **get ()** and **pop ()**.

Sample Program: Output:



```
dicte={"name":"kishor","age":19,"ph no":8907357726}

print('accessing dictionary item')
print(dicte["name"])

print("printing keys")
x=dicte.keys()

print("printing the values")
print(dicte.values())

print("changing dictionary item")
dicte["age"]=18
print(dicte)

print("add items")
dicte["regno"]=59
print(dicte)

print("update function")
dicte.update({"regno":"059"})
print(dicte)

print("pop function")
print(dicte.pop("ph no"))
print(dicte)

print("popitem function")
dicte.popitem()
print(dicte)

print("del function")
```



```
C:\Users\bkish\AppData\Local\Programs\Python\Launcher\py.exe
accessing dictionary item
kishor
printing keys
printing the values
dict_values(['kishor', 19, 8907357726])
changing dictionary item
{'name': 'kishor', 'age': 18, 'ph no': 8907357726}
add items
{'name': 'kishor', 'age': 18, 'ph no': 8907357726, 'regno': 59}
update function
{'name': 'kishor', 'age': 18, 'ph no': 8907357726, 'regno': '059'}
pop function
8907357726
{'name': 'kishor', 'age': 18, 'regno': '059'}
popitem function
{'name': 'kishor', 'age': 18}
del function
{'age': 18}
clear function
{'age': 18}
```

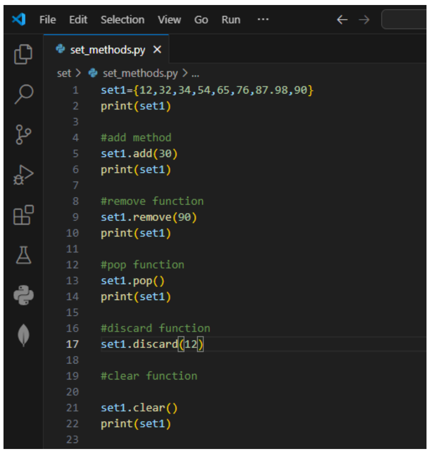
Set:

A set is a data structure in Python that stores an unordered collection of unique items.

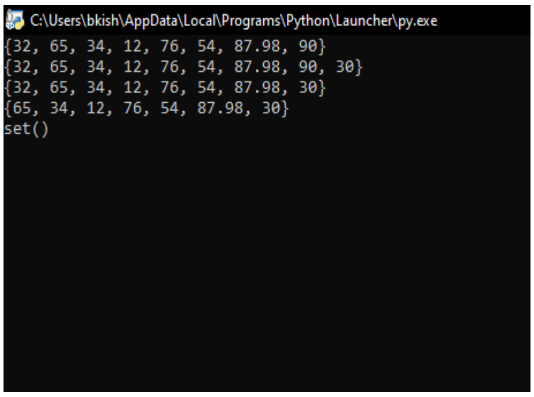
Set Methods:

- Implemented functions like **add ()**, **remove ()**, and **discard ()** to manage set elements.
- Performed set operations such as **union**, **intersection**, and **difference.**
- Managed data with **pop ()** to remove elements.
- Used **clear ()** to remove all elements from a set.
- Checked the size of a set with **len ().**
- Converted other data types to sets for comparison and operations.

Methods: **Output:**



```
set1={12,32,34,54,65,76,87.98,90}
print(set1)

#add method
set1.add(30)
print(set1)

#remove function
set1.remove(90)
print(set1)

#pop function
set1.pop()
print(set1)

#discard function
set1.discard(12)

#clear function

set1.clear()
print(set1)
```



```
C:\Users\bkish\AppData\Local\Programs\Python\Launcher\py.exe
{32, 65, 34, 12, 76, 54, 87.98, 90}
{32, 65, 34, 12, 76, 54, 87.98, 90, 30}
{32, 65, 34, 12, 76, 54, 87.98, 30}
{65, 34, 12, 76, 54, 87.98, 30}
set()
```

CRUD:

CRUD stands for Create, Read, Update, and Delete. It represents the basic operations for managing data.

CRUD Function using Dictionary:

CRUD using a dictionary involves performing basic data management operations on a dictionary data structure

Code Description:

Implemented CRUD Operations:

Created functions to handle Create, Read, Update, and Delete operations on a dictionary.

Create Function:

Added new key-value pairs if the key did not already exist, with feedback if the key was present.

Read Function:

Retrieved and displayed the value for a given key, with feedback if the key did not exist.

Update Function:

Removed existing key-value pairs and updated with new data, with checks for key existence.

Delete Function:

Removed key-value pairs from the dictionary, with feedback if the key did not exist.

User Interaction:

Provided a menu-driven interface for users to choose operations or exit, using a loop for continuous input.

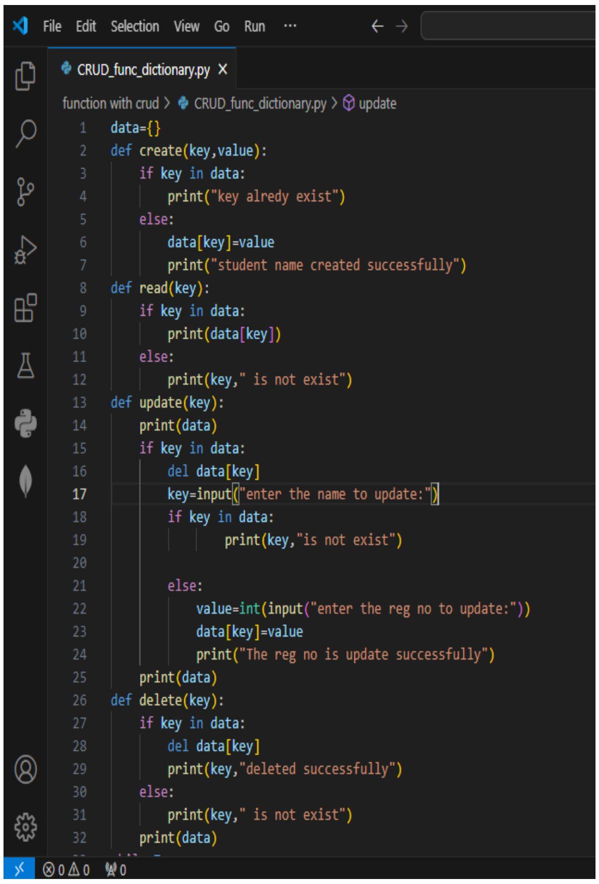
Error Handling:

Included checks to ensure operations are performed only if the key exists, with appropriate feedback.

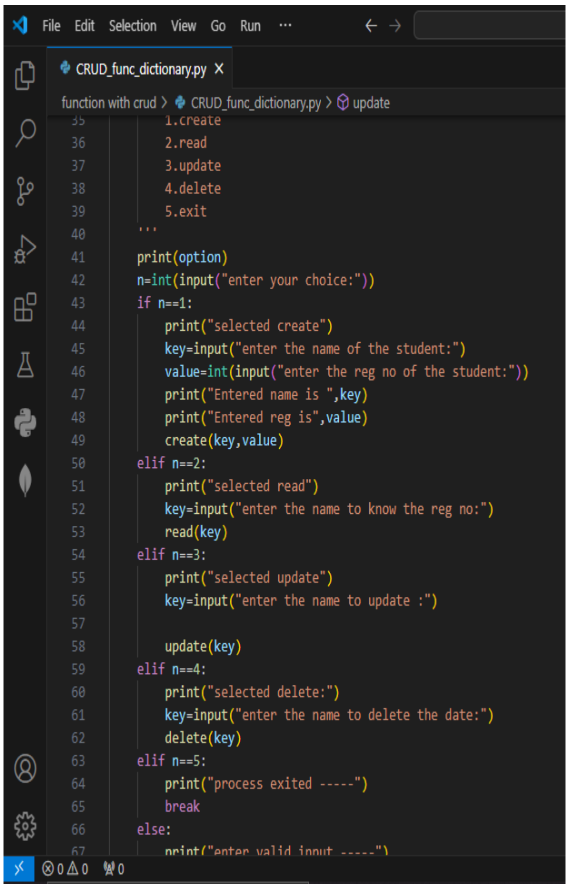
Input Management:

Handled user input for creating, updating, and deleting data entries.

Sample Program (CRUD):

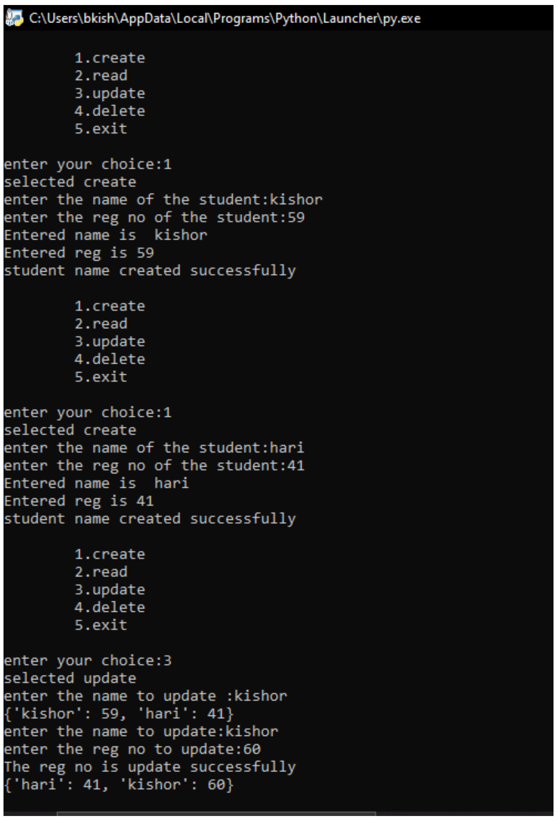

File Edit Selection View Go Run ···
CRUD_func_dictionary.py
function with crud > CRUD_func_dictionary.py > update
⊗ 0 ⚠ 0 0

```python
data={}
def create(key,value):
    if key in data:
        print("key alredy exist")
    else:
        data[key]=value
        print("student name created successfully")
def read(key):
    if key in data:
        print(data[key])
    else:
        print(key," is not exist")
def update(key):
    print(data)
    if key in data:
        del data[key]
        key=input("enter the name to update:")
        if key in data:
                print(key,"is not exist")

        else:
            value=int(input("enter the reg no to update:"))
            data[key]=value
            print("The reg no is update successfully")
    print(data)
def delete(key):
    if key in data:
        del data[key]
        print(key,"deleted successfully")
    else:
        print(key," is not exist")
    print(data)
```

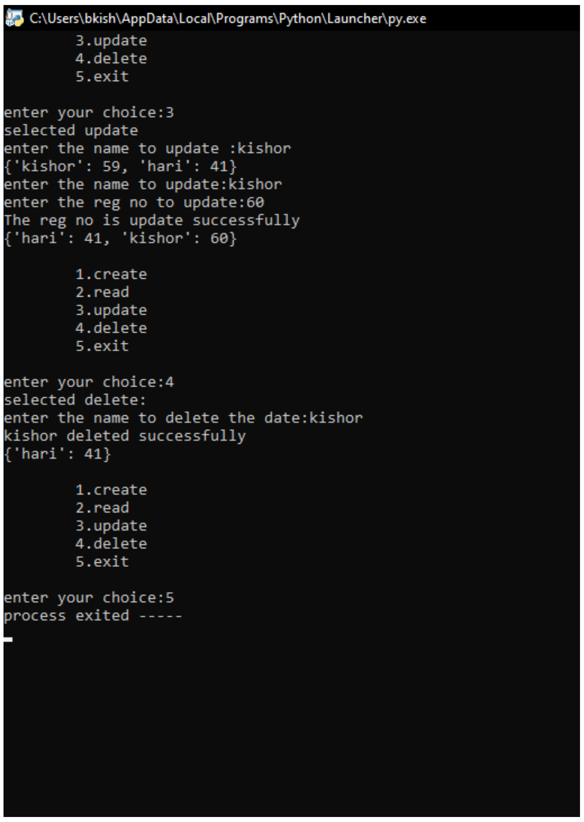

File Edit Selection View Go Run ···
CRUD_func_dictionary.py
function with crud > CRUD_func_dictionary.py > update
⊗ 0 ⚠ 0 0

```python
        1.create
        2.read
        3.update
        4.delete
        5.exit
    '''
    print(option)
    n=int(input("enter your choice:"))
    if n==1:
        print("selected create")
        key=input("enter the name of the student:")
        value=int(input("enter the reg no of the student:"))
        print("Entered name is ",key)
        print("Entered reg is",value)
        create(key,value)
    elif n==2:
        print("selected read")
        key=input("enter the name to know the reg no:")
        read(key)
    elif n==3:
        print("selected update")
        key=input("enter the name to update :")

        update(key)
    elif n==4:
        print("selected delete:")
        key=input("enter the name to delete the date:")
        delete(key)
    elif n==5:
        print("process exited -----")
        break
    else:
        print("enter valid input -----")
```

Output:



```
C:\Users\bkish\AppData\Local\Programs\Python\Launcher\py.exe

        1.create
        2.read
        3.update
        4.delete
        5.exit

enter your choice:1
selected create
enter the name of the student:kishor
enter the reg no of the student:59
Entered name is  kishor
Entered reg is 59
student name created successfully

        1.create
        2.read
        3.update
        4.delete
        5.exit

enter your choice:1
selected create
enter the name of the student:hari
enter the reg no of the student:41
Entered name is  hari
Entered reg is 41
student name created successfully

        1.create
        2.read
        3.update
        4.delete
        5.exit

enter your choice:3
selected update
enter the name to update :kishor
{'kishor': 59, 'hari': 41}
enter the name to update:kishor
enter the reg no to update:60
The reg no is update successfully
{'hari': 41, 'kishor': 60}
```



```
C:\Users\bkish\AppData\Local\Programs\Python\Launcher\py.exe
        3.update
        4.delete
        5.exit

enter your choice:3
selected update
enter the name to update :kishor
{'kishor': 59, 'hari': 41}
enter the name to update:kishor
enter the reg no to update:60
The reg no is update successfully
{'hari': 41, 'kishor': 60}

        1.create
        2.read
        3.update
        4.delete
        5.exit

enter your choice:4
selected delete:
enter the name to delete the date:kishor
kishor deleted successfully
{'hari': 41}

        1.create
        2.read
        3.update
        4.delete
        5.exit

enter your choice:5
process exited -----
```

CRUD Function Using List:

CRUD using a List involves performing basic data management operations on a List of Tuple data structure

Implemented CRUD Operations:

Created functions for managing student records with a list of tuples for basic data operations.

Code Description:

Create Function:

Added new records if the name was not already in the list. Alerted users if the name existed.

Read Function:

Retrieved and displayed registration numbers for given names. Notified users if the name was not found.

Update Function:

Modified existing records by updating the registration number. Checked if the name was present before making updates.

Delete Function:

Removed records based on the provided name. Informed users if the name did not exist in the list.

User Interaction:

Built a menu-driven system for users to select operations or exit. Used a loop to keep the program running until exit.

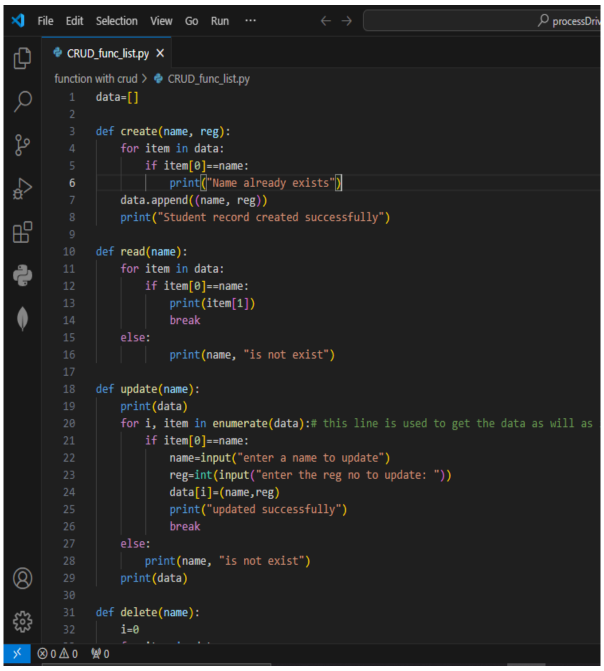
Error Handling:

Verified key existence before performing operations and provided feedback for missing names.

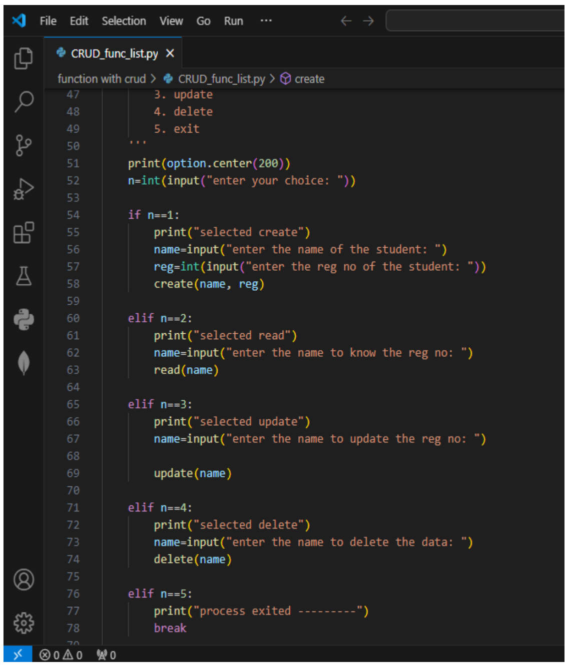
Input Management:

Processed user inputs for creating, updating, and deleting records, ensuring data accuracy and proper handling.

Sample Program:

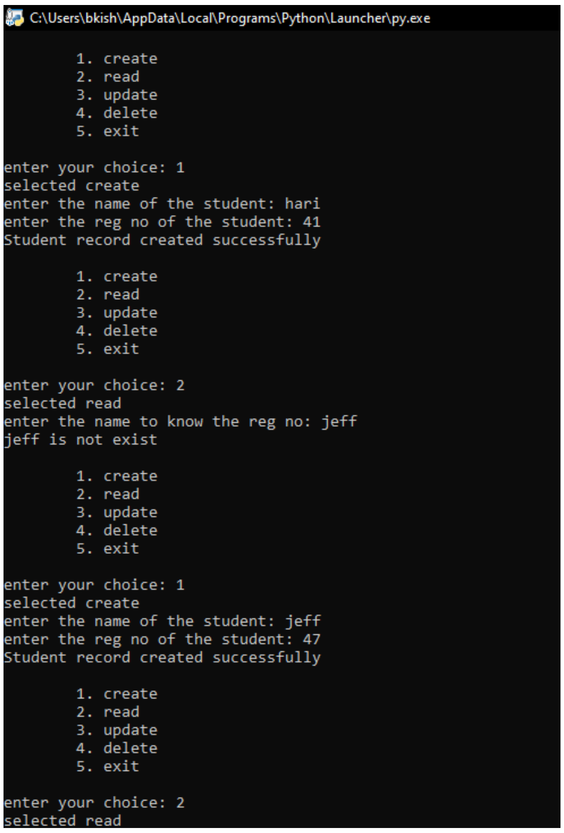

File Edit Selection View Go Run ···
processDriv
CRUD_func_list.py
function with crud > CRUD_func_list.py

```python
data=[]

def create(name, reg):
    for item in data:
        if item[0]==name:
            print("Name already exists")
    data.append((name, reg))
    print("Student record created successfully")

def read(name):
    for item in data:
        if item[0]==name:
            print(item[1])
            break
    else:
            print(name, "is not exist")

def update(name):
    print(data)
    for i, item in enumerate(data):# this line is used to get the data as will as
        if item[0]==name:
            name=input("enter a name to update")
            reg=int(input("enter the reg no to update: "))
            data[i]=(name,reg)
            print("updated successfully")
            break
    else:
        print(name, "is not exist")
    print(data)

def delete(name):
    i=0
```

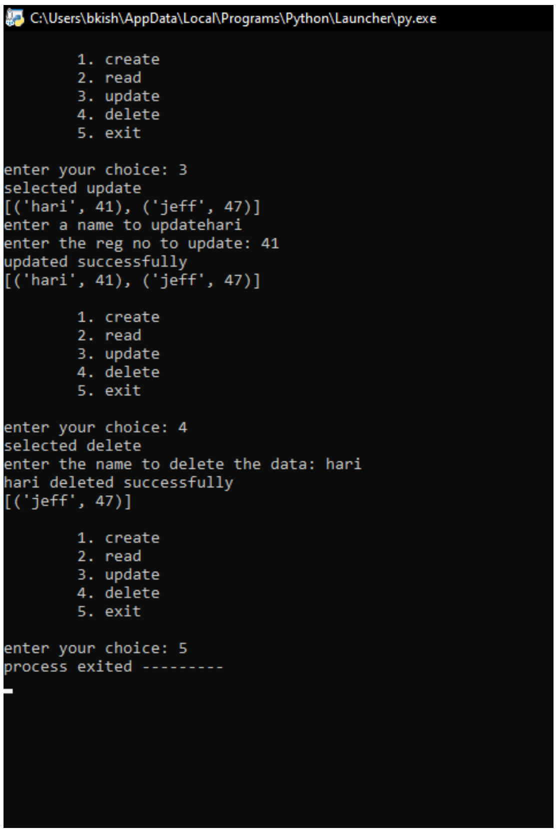

File Edit Selection View Go Run ···
CRUD_func_list.py
function with crud > CRUD_func_list.py > create

```python
        3. update
        4. delete
        5. exit
    '''
    print(option.center(200))
    n=int(input("enter your choice: "))

    if n==1:
        print("selected create")
        name=input("enter the name of the student: ")
        reg=int(input("enter the reg no of the student: "))
        create(name, reg)

    elif n==2:
        print("selected read")
        name=input("enter the name to know the reg no: ")
        read(name)

    elif n==3:
        print("selected update")
        name=input("enter the name to update the reg no: ")

        update(name)

    elif n==4:
        print("selected delete")
        name=input("enter the name to delete the data: ")
        delete(name)

    elif n==5:
        print("process exited ---------")
        break
```

Output:


C:\Users\bkish\AppData\Local\Programs\Python\Launcher\py.exe

```
        1. create
        2. read
        3. update
        4. delete
        5. exit

enter your choice: 1
selected create
enter the name of the student: hari
enter the reg no of the student: 41
Student record created successfully

        1. create
        2. read
        3. update
        4. delete
        5. exit

enter your choice: 2
selected read
enter the name to know the reg no: jeff
jeff is not exist

        1. create
        2. read
        3. update
        4. delete
        5. exit

enter your choice: 1
selected create
enter the name of the student: jeff
enter the reg no of the student: 47
Student record created successfully

        1. create
        2. read
        3. update
        4. delete
        5. exit

enter your choice: 2
selected read
```


C:\Users\bkish\AppData\Local\Programs\Python\Launcher\py.exe

```
        1. create
        2. read
        3. update
        4. delete
        5. exit

enter your choice: 3
selected update
[('hari', 41), ('jeff', 47)]
enter a name to updatehari
enter the reg no to update: 41
updated successfully
[('hari', 41), ('jeff', 47)]

        1. create
        2. read
        3. update
        4. delete
        5. exit

enter your choice: 4
selected delete
enter the name to delete the data: hari
hari deleted successfully
[('jeff', 47)]

        1. create
        2. read
        3. update
        4. delete
        5. exit

enter your choice: 5
process exited ---------
```

API (Application Programming Interface):

An API (Application Programming Interface) allows software systems to communicate by providing a set of rules for exchanging data and functionality. It enables integration and interaction between different applications.

Explored HTTP Methods with Postman Using API:

Installed Postman:

Set up Postman for testing HTTP methods.

Created Requests:

Configured API requests by selecting HTTP methods (GET, POST, PUT, DELETE) and entering endpoint URLs.

Added Headers:

Included required headers like Content-Type and Authorization.

Entered Body Data:

Provided request body data in JSON format for POST and PUT methods.

Sent Requests:

Executed API requests and examined the responses for accuracy and expected behavior.

Wrote Test Scripts:

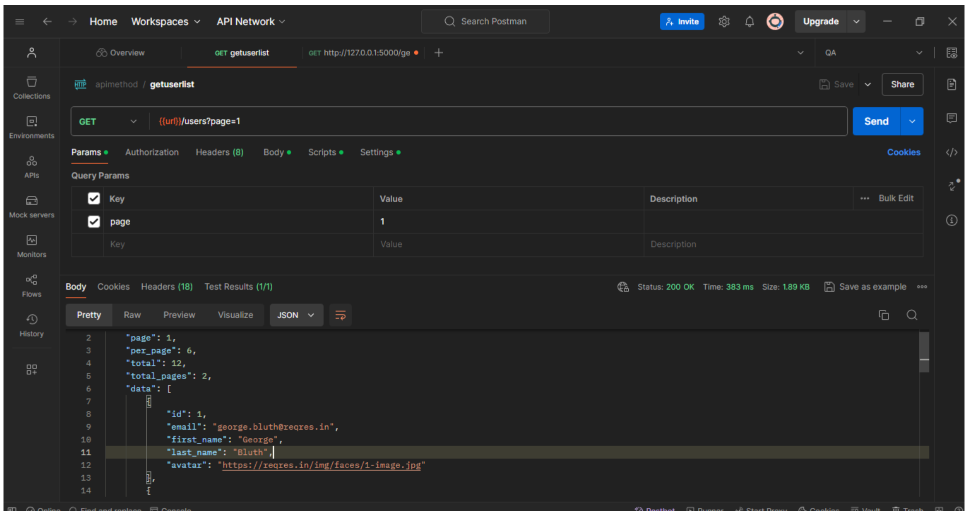
Automated API testing by writing test scripts in Postman's "Tests" tab using JavaScript.

HTTP Methods:

GET Method:

The GET method in HTTP is used to retrieve data from a specified resource. It requests information without altering the resource. The data is returned in the response body.

Example:



apimethod / getuserlist

GET {{url}}/users?page=1

Query Params

	Key	Value	Description
✓	page	1	

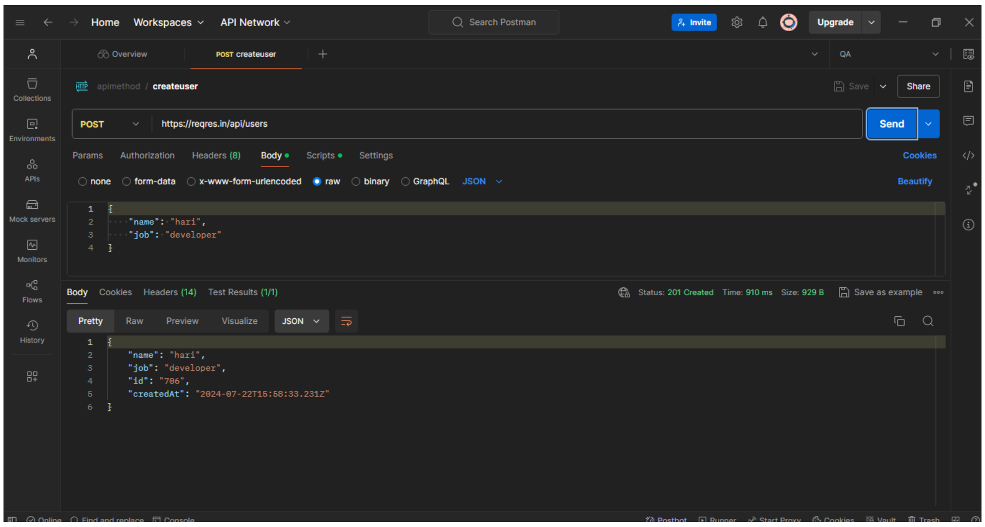
Status: 200 OK Time: 383 ms Size: 1.89 KB

```
    "page": 1,
    "per_page": 6,
    "total": 12,
    "total_pages": 2,
    "data": [
        {
            "id": 1,
            "email": "george.bluth@reqres.in",
            "first_name": "George",
            "last_name": "Bluth",
            "avatar": "https://reqres.in/img/faces/1-image.jpg"
        },
        {
```

Post Method:

The POST method in HTTP sends data to a server to create or update a resource. It includes data in the request body and can change the server's state.

Example:



apimethod / createuser

POST https://reqres.in/api/users

```
{
    "name": "hari",
    "job": "developer"
}
```

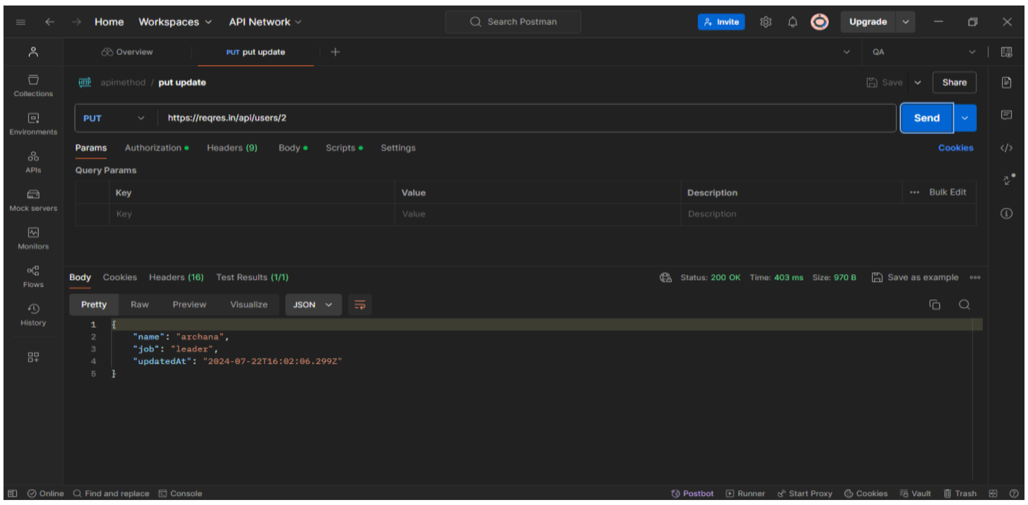
Status: 201 Created Time: 910 ms Size: 929 B

```
{
    "name": "hari",
    "job": "developer",
    "id": "706",
    "createdAt": "2024-07-22T15:58:33.231Z"
}
```

Put Method:

The PUT method in HTTP updates or replaces a resource on the server. It sends data to the server in the request body, often used to modify existing resources.

Explain:


PUT https://reqres.in/api/users/2

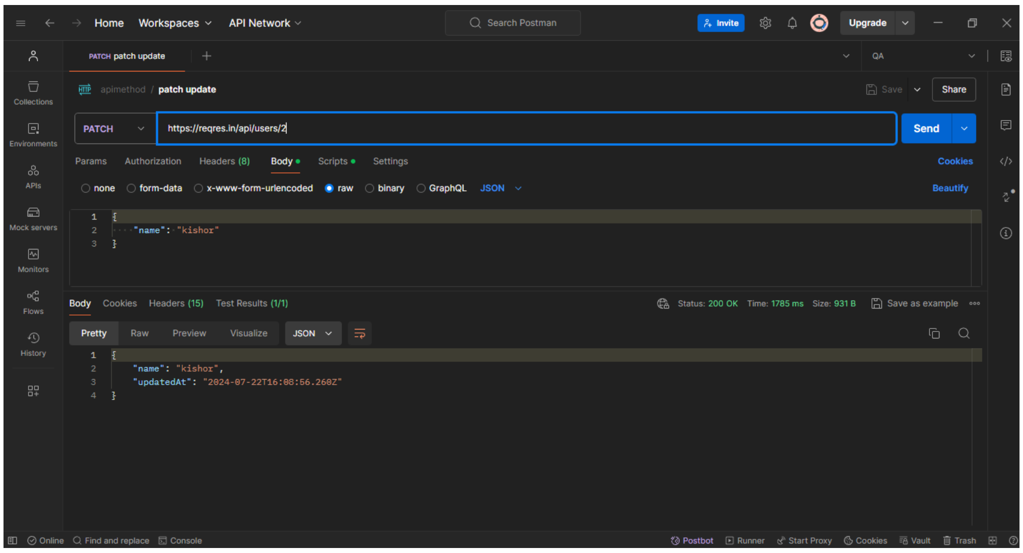
Status: 200 OK Time: 403 ms Size: 970 B

```
{
    "name": "archana",
    "job": "leader",
    "updatedAt": "2024-07-22T16:02:06.299Z"
}
```

Patch Method:

The PATCH method in HTTP partially updates a resource on the server. It sends only the data that needs to be updated, rather than replacing the entire resource.

Example:


PATCH https://reqres.in/api/users/2

```
{
    "name": "kishor"
}
```

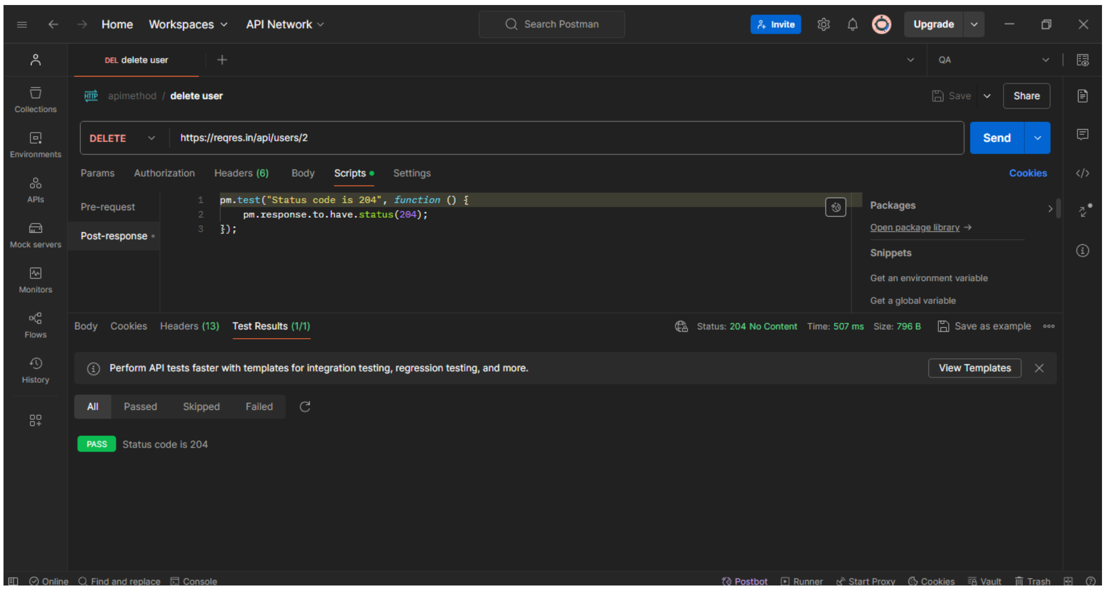
Status: 200 OK Time: 1785 ms Size: 931 B

```
{
    "name": "kishor",
    "updatedAt": "2024-07-22T16:08:56.260Z"
}
```

Delete Method:

The DELETE method in HTTP removes a specified resource from the server. It requests the server to delete the resource identified by the URL.

Example:



DELETE https://reqres.in/api/users/2

```
pm.test("Status code is 204", function () {
    pm.response.to.have.status(204);
});
```

Status: 204 No Content Time: 507 ms Size: 796 B

Perform API tests faster with templates for integration testing, regression testing, and more.

PASS Status code is 204

HTTP Methods Used in Python:

In Python, Various HTTP methods were used in Python with the **requests** library. GET requests retrieved data, POST requests created or updated resources, PUT requests replaced or updated resources, PATCH requests applied partial updates, and DELETE requests removed resources. This process provided a solid understanding of handling data and interacting with web services.

Code Description:

GET Method:

Created a function to send HTTP GET requests, fetching and displaying data from a URL in JSON format.

POST Method:

Developed a function for HTTP POST requests to send JSON data to a URL, handling success and error responses.

PUT Method:

Implemented a function to perform HTTP PUT requests for updating existing resources, including error handling and response formatting.

PATCH Method:

Built a function for HTTP PATCH requests to apply partial updates to resources, with success and error feedback.

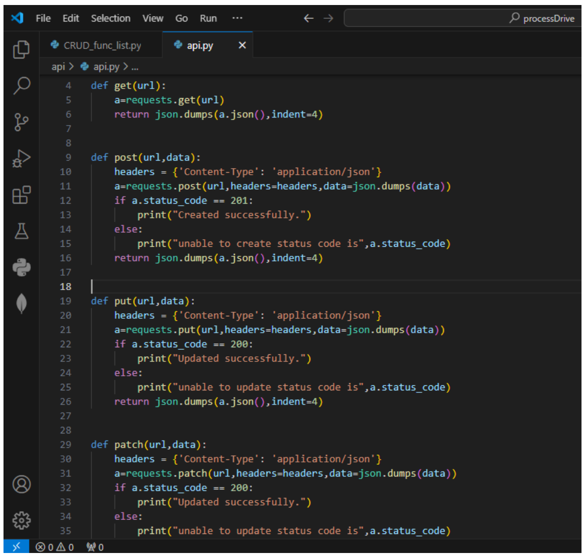
DELETE Method:

Designed a function for HTTP DELETE requests to remove resources from a URL, providing status messages based on the outcome.

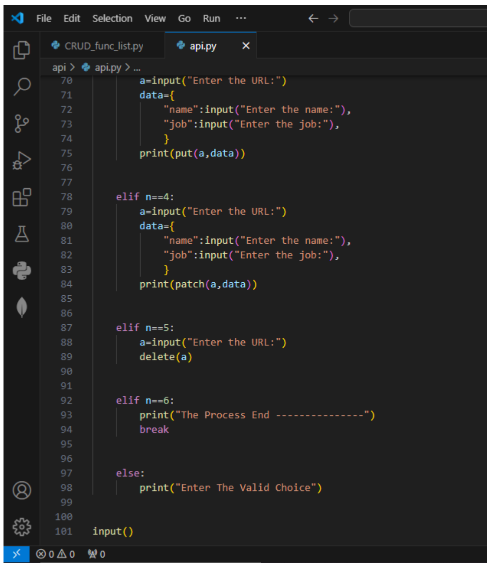
User Interface:

Created a menu-driven interface to select and execute different HTTP methods, allowing users to input URLs and data interactively.

Sample program:

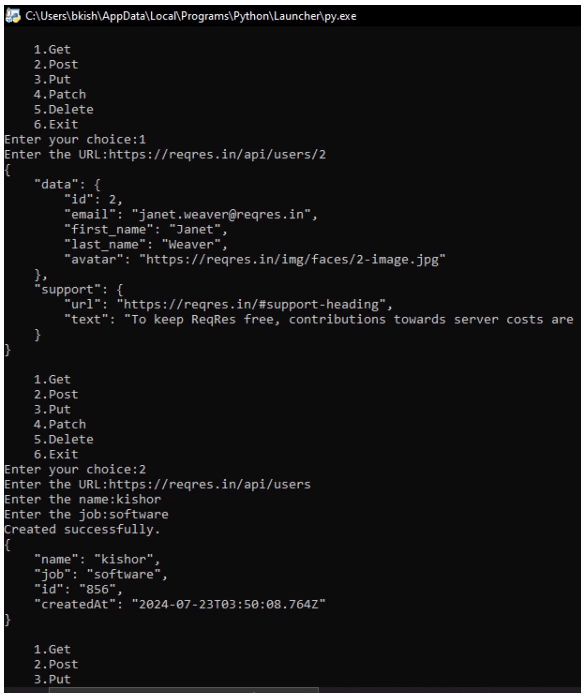


```python
def get(url):
    a=requests.get(url)
    return json.dumps(a.json(),indent=4)


def post(url,data):
    headers = {'Content-Type': 'application/json'}
    a=requests.post(url,headers=headers,data=json.dumps(data))
    if a.status_code == 201:
        print("Created successfully.")
    else:
        print("unable to create status code is",a.status_code)
    return json.dumps(a.json(),indent=4)


def put(url,data):
    headers = {'Content-Type': 'application/json'}
    a=requests.put(url,headers=headers,data=json.dumps(data))
    if a.status_code == 200:
        print("Updated successfully.")
    else:
        print("unable to update status code is",a.status_code)
    return json.dumps(a.json(),indent=4)


def patch(url,data):
    headers = {'Content-Type': 'application/json'}
    a=requests.patch(url,headers=headers,data=json.dumps(data))
    if a.status_code == 200:
        print("Updated successfully.")
    else:
        print("unable to update status code is",a.status_code)
```

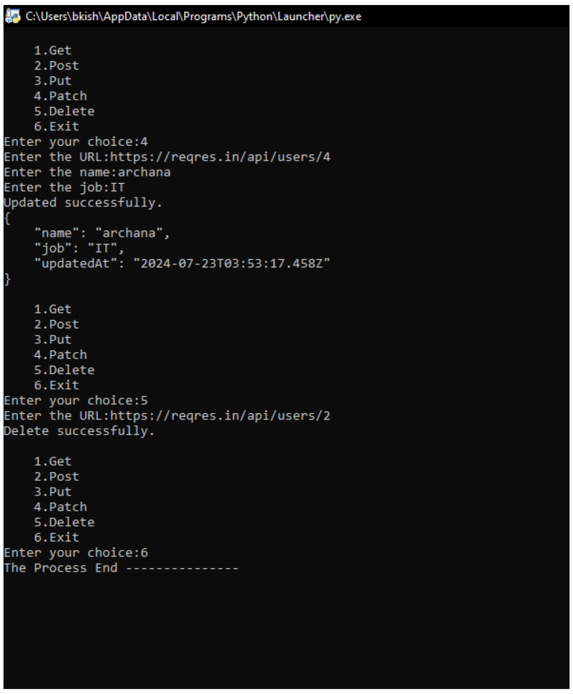


```python
        a=input("Enter the URL:")
        data={
            "name":input("Enter the name:"),
            "job":input("Enter the job:"),
            }
        print(put(a,data))


    elif n==4:
        a=input("Enter the URL:")
        data={
            "name":input("Enter the name:"),
            "job":input("Enter the job:"),
            }
        print(patch(a,data))


    elif n==5:
        a=input("Enter the URL:")
        delete(a)


    elif n==6:
        print("The Process End ---------------")
        break


    else:
        print("Enter The Valid Choice")


input()
```

Output:



```
C:\Users\bkish\AppData\Local\Programs\Python\Launcher\py.exe

    1.Get
    2.Post
    3.Put
    4.Patch
    5.Delete
    6.Exit
Enter your choice:1
Enter the URL:https://reqres.in/api/users/2
{
    "data": {
        "id": 2,
        "email": "janet.weaver@reqres.in",
        "first_name": "Janet",
        "last_name": "Weaver",
        "avatar": "https://reqres.in/img/faces/2-image.jpg"
    },
    "support": {
        "url": "https://reqres.in/#support-heading",
        "text": "To keep ReqRes free, contributions towards server costs are
    }
}

    1.Get
    2.Post
    3.Put
    4.Patch
    5.Delete
    6.Exit
Enter your choice:2
Enter the URL:https://reqres.in/api/users
Enter the name:kishor
Enter the job:software
Created successfully.
{
    "name": "kishor",
    "job": "software",
    "id": "856",
    "createdAt": "2024-07-23T03:50:08.764Z"
}

    1.Get
    2.Post
    3.Put
```



```
C:\Users\bkish\AppData\Local\Programs\Python\Launcher\py.exe

    1.Get
    2.Post
    3.Put
    4.Patch
    5.Delete
    6.Exit
Enter your choice:4
Enter the URL:https://reqres.in/api/users/4
Enter the name:archana
Enter the job:IT
Updated successfully.
{
    "name": "archana",
    "job": "IT",
    "updatedAt": "2024-07-23T03:53:17.458Z"
}

    1.Get
    2.Post
    3.Put
    4.Patch
    5.Delete
    6.Exit
Enter your choice:5
Enter the URL:https://reqres.in/api/users/2
Delete successfully.

    1.Get
    2.Post
    3.Put
    4.Patch
    5.Delete
    6.Exit
Enter your choice:6
The Process End --------------
```

MongoDB:

MongoDB is a NoSQL database that stores data in flexible, JSON-like documents, allowing for scalable and high-performance applications.

MongoDB in Python:

Using MongoDB with Python involves connecting to a MongoDB database using the **PyMongo** library and performing CRUD (Create, Read, Update, Delete) operations on collections. This allows for flexible schema design and efficient data management. MongoDB integration supports handling large datasets and enhances the development of data-driven applications.

Code Description:

Connected to MongoDB Database:
Established a connection to a local MongoDB instance using **PyMongo**, specifying database and collection names.

Create Document Function:
Implemented a function to insert documents into the collection if they do not already exist, ensuring data integrity.

Read Specific Document Function:
Created a function to retrieve and display a document based on the provided name, facilitating specific data access.

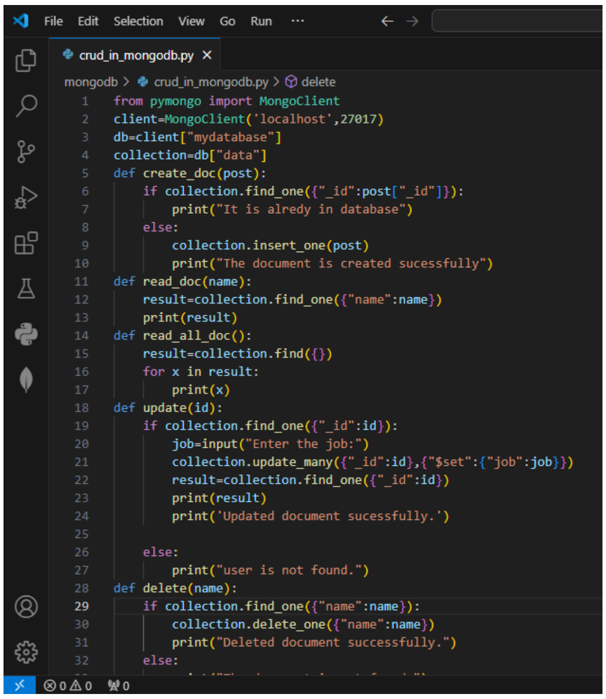
Read All Documents Function:
Developed a function to retrieve and display all documents in the collection, enabling comprehensive data overview.

Update Document Function:
Implemented an update function that modifies the job field of a document based on the provided ID, ensuring data accuracy.

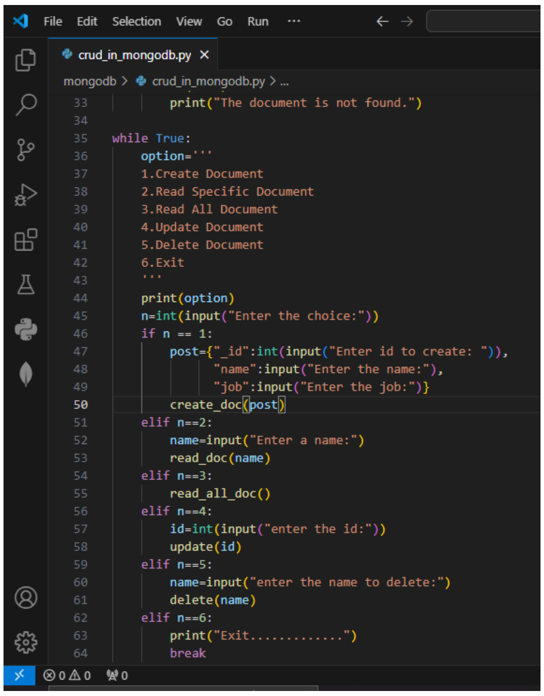
Delete Document Function:
Created a function to delete documents from the collection based on the provided name, maintaining data relevance.

User Interaction:
Designed a menu-driven interface allowing users to perform CRUD operations interactively, ensuring ease of use and functionality.

Sample program:

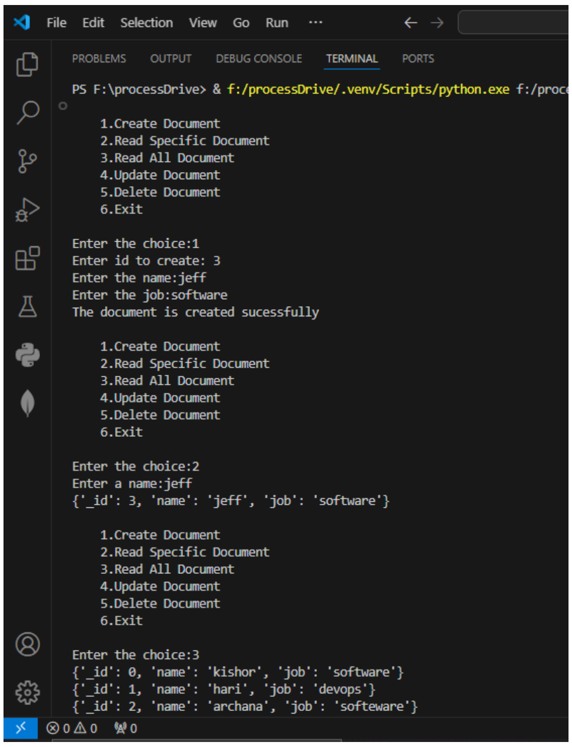


```python
from pymongo import MongoClient
client=MongoClient('localhost',27017)
db=client["mydatabase"]
collection=db["data"]
def create_doc(post):
    if collection.find_one({"_id":post["_id"]}):
        print("It is alredy in database")
    else:
        collection.insert_one(post)
        print("The document is created sucessfully")
def read_doc(name):
    result=collection.find_one({"name":name})
    print(result)
def read_all_doc():
    result=collection.find({})
    for x in result:
        print(x)
def update(id):
    if collection.find_one({"_id":id}):
        job=input("Enter the job:")
        collection.update_many({"_id":id},{"$set":{"job":job}})
        result=collection.find_one({"_id":id})
        print(result)
        print('Updated document sucessfully.')

    else:
        print("user is not found.")
def delete(name):
    if collection.find_one({"name":name}):
        collection.delete_one({"name":name})
        print("Deleted document successfully.")
    else:
```

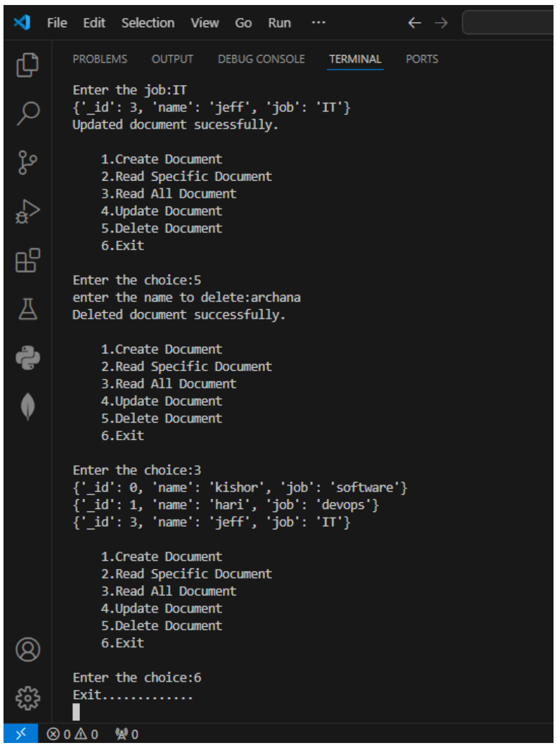


```python
            print("The document is not found.")

while True:
    option='''
    1.Create Document
    2.Read Specific Document
    3.Read All Document
    4.Update Document
    5.Delete Document
    6.Exit
    '''
    print(option)
    n=int(input("Enter the choice:"))
    if n == 1:
        post={"_id":int(input("Enter id to create: ")),
              "name":input("Enter the name:"),
              "job":input("Enter the job:")}
        create_doc(post)
    elif n==2:
        name=input("Enter a name:")
        read_doc(name)
    elif n==3:
        read_all_doc()
    elif n==4:
        id=int(input("enter the id:"))
        update(id)
    elif n==5:
        name=input("enter the name to delete:")
        delete(name)
    elif n==6:
        print("Exit............")
        break
```

Output:

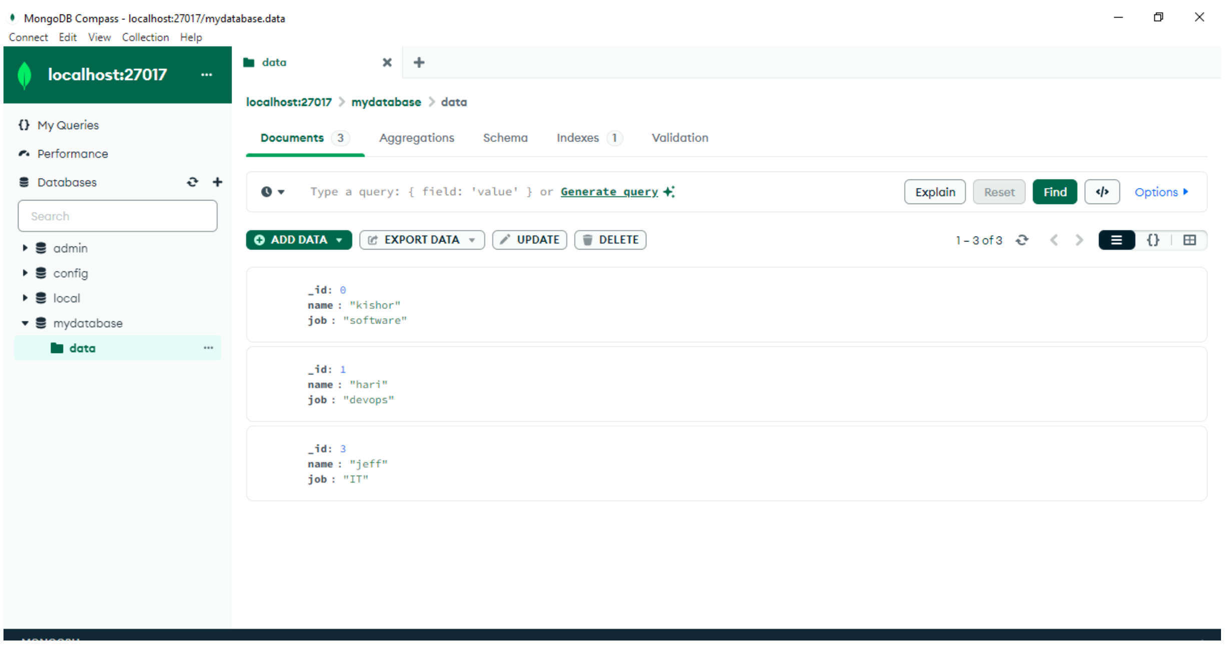


```
PS F:\processDrive> & f:/processDrive/.venv/Scripts/python.exe f:/proce

    1.Create Document
    2.Read Specific Document
    3.Read All Document
    4.Update Document
    5.Delete Document
    6.Exit

Enter the choice:1
Enter id to create: 3
Enter the name:jeff
Enter the job:software
The document is created sucessfully

    1.Create Document
    2.Read Specific Document
    3.Read All Document
    4.Update Document
    5.Delete Document
    6.Exit

Enter the choice:2
Enter a name:jeff
{'_id': 3, 'name': 'jeff', 'job': 'software'}

    1.Create Document
    2.Read Specific Document
    3.Read All Document
    4.Update Document
    5.Delete Document
    6.Exit

Enter the choice:3
{'_id': 0, 'name': 'kishor', 'job': 'software'}
{'_id': 1, 'name': 'hari', 'job': 'devops'}
{'_id': 2, 'name': 'archana', 'job': 'softeware'}
```



```
Enter the job:IT
{'_id': 3, 'name': 'jeff', 'job': 'IT'}
Updated document sucessfully.

    1.Create Document
    2.Read Specific Document
    3.Read All Document
    4.Update Document
    5.Delete Document
    6.Exit

Enter the choice:5
enter the name to delete:archana
Deleted document successfully.

    1.Create Document
    2.Read Specific Document
    3.Read All Document
    4.Update Document
    5.Delete Document
    6.Exit

Enter the choice:3
{'_id': 0, 'name': 'kishor', 'job': 'software'}
{'_id': 1, 'name': 'hari', 'job': 'devops'}
{'_id': 3, 'name': 'jeff', 'job': 'IT'}

    1.Create Document
    2.Read Specific Document
    3.Read All Document
    4.Update Document
    5.Delete Document
    6.Exit

Enter the choice:6
Exit............
```

Database in MongoDB:



```
_id: 0
name : "kishor"
job : "software"

_id: 1
name : "hari"
job : "devops"

_id: 3
name : "jeff"
job : "IT"
```

Project:

I worked on a project using the Gemini API, which included an image-to-text converter, a video-to-text converter, and a fun chat application. The image-to-text converter extracted text from images, while the video-to-text converter transcribed spoken words from videos into text. The fun chat application offered an engaging, interactive user experience. This project enhanced my skills in API integration and data processing.

Project 1:

Fun Chat Using Gemini API:

I worked on a project called FunChat using the Gemini API. FunChat is an interactive chat application that integrates various features to enhance user experience. Using the Gemini API, the application can process and understand user inputs, providing relevant and engaging responses. This project improved my skills in API integration, natural language processing, and real-time communication.

Tools Used:

- Google Generative AI (Gemini API)
- Python Programming Language
- requests Library
- json Library

Code Description:

API Integration:

Configured and utilized the Gemini API for generating conversational responses, including setting up the API key and selecting the appropriate model.

Chat Initialization:

Implemented the '**start_conversation'** function to initiate a new chat session using the Gemini API.

Response Handling:

Developed the '**get_response'** function to send user input to the model and receive a generated response. Included logic to limit response length to 200 words for display.

User Interaction:

Created a command-line interface for users to interact with the chat application, allowing input and receiving responses from the Gemini model.

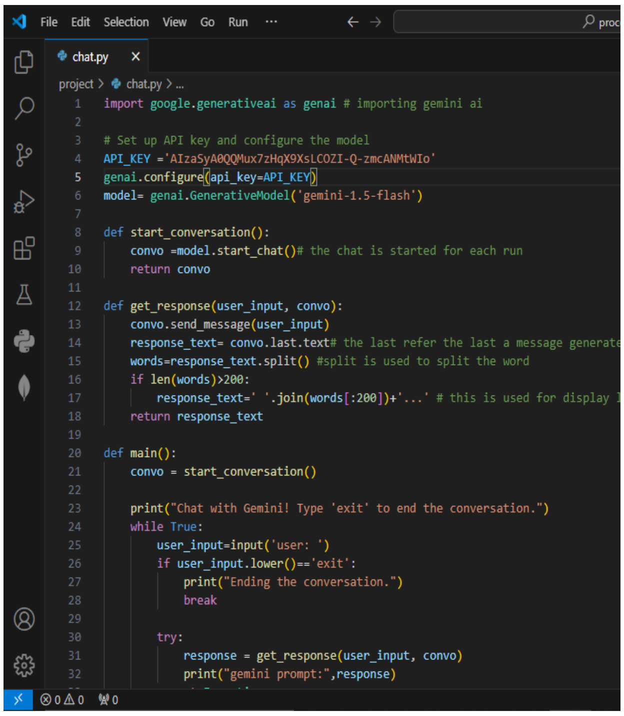
Error Management:

Incorporated exception handling to manage potential errors during interaction with the API, ensuring robust operation and user-friendly error messages.

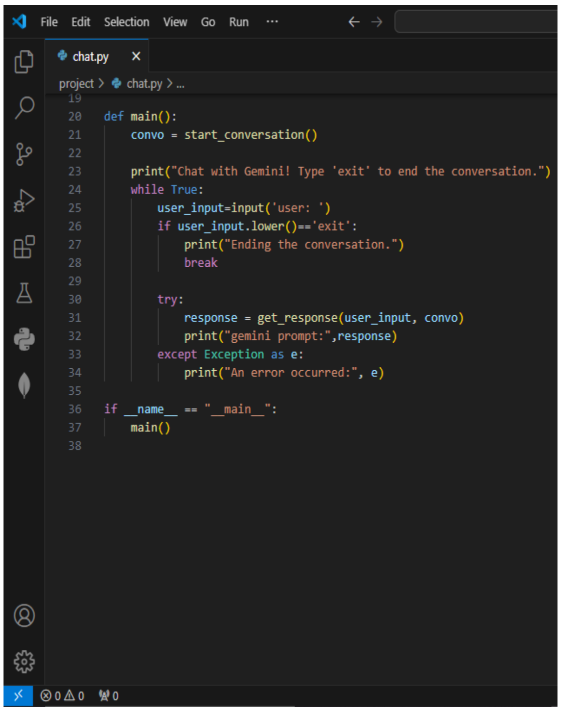
Session Management:

Enabled continuous chat interaction until the user inputs 'exit', providing a seamless conversational experience

Program:

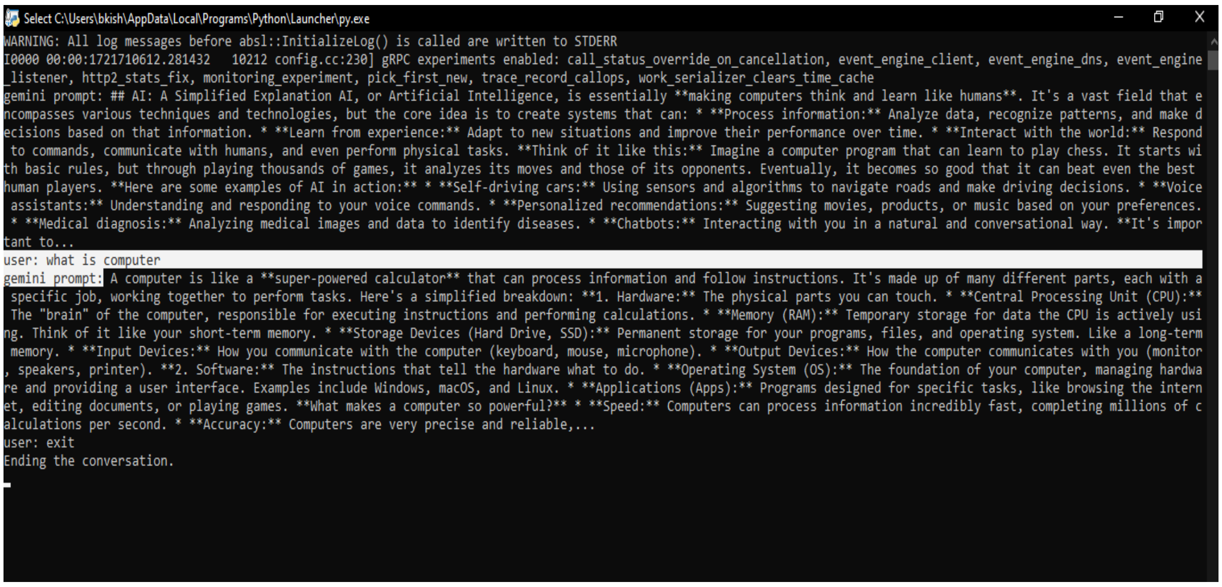


```python
import google.generativeai as genai # importing gemini ai

# Set up API key and configure the model
API_KEY ='AIzaSyA0QQMux7zHqX9XsLCOZI-Q-zmcANMtWIo'
genai.configure(api_key=API_KEY)
model= genai.GenerativeModel('gemini-1.5-flash')

def start_conversation():
    convo =model.start_chat()# the chat is started for each run
    return convo

def get_response(user_input, convo):
    convo.send_message(user_input)
    response_text= convo.last.text# the last refer the last a message generate
    words=response_text.split() #split is used to split the word
    if len(words)>200:
        response_text=' '.join(words[:200])+'...' # this is used for display
    return response_text

def main():
    convo = start_conversation()

    print("Chat with Gemini! Type 'exit' to end the conversation.")
    while True:
        user_input=input('user: ')
        if user_input.lower()=='exit':
            print("Ending the conversation.")
            break

        try:
            response = get_response(user_input, convo)
            print("gemini prompt:",response)
```



```python
def main():
    convo = start_conversation()

    print("Chat with Gemini! Type 'exit' to end the conversation.")
    while True:
        user_input=input('user: ')
        if user_input.lower()=='exit':
            print("Ending the conversation.")
            break

        try:
            response = get_response(user_input, convo)
            print("gemini prompt:",response)
        except Exception as e:
            print("An error occurred:", e)

if __name__ == "__main__":
    main()
```

Output:



Select C:\Users\bkish\AppData\Local\Programs\Python\Launcher\py.exe

```
WARNING: All log messages before absl::InitializeLog() is called are written to STDERR
I0000 00:00:1721710612.281432   10212 config.cc:230] gRPC experiments enabled: call_status_override_on_cancellation, event_engine_client, event_engine_dns, event_engine
_listener, http2_stats_fix, monitoring_experiment, pick_first_new, trace_record_callops, work_serializer_clears_time_cache
gemini prompt: ## AI: A Simplified Explanation AI, or Artificial Intelligence, is essentially **making computers think and learn like humans**. It's a vast field that e
ncompasses various techniques and technologies, but the core idea is to create systems that can: * **Process information:** Analyze data, recognize patterns, and make d
ecisions based on that information. * **Learn from experience:** Adapt to new situations and improve their performance over time. * **Interact with the world:** Respond
 to commands, communicate with humans, and even perform physical tasks. **Think of it like this:** Imagine a computer program that can learn to play chess. It starts wi
th basic rules, but through playing thousands of games, it analyzes its moves and those of its opponents. Eventually, it becomes so good that it can beat even the best
human players. **Here are some examples of AI in action:** * **Self-driving cars:** Using sensors and algorithms to navigate roads and make driving decisions. * **Voice
 assistants:** Understanding and responding to your voice commands. * **Personalized recommendations:** Suggesting movies, products, or music based on your preferences.
 * **Medical diagnosis:** Analyzing medical images and data to identify diseases. * **Chatbots:** Interacting with you in a natural and conversational way. **It's impor
tant to...
user: what is computer
gemini prompt: A computer is like a **super-powered calculator** that can process information and follow instructions. It's made up of many different parts, each with a
 specific job, working together to perform tasks. Here's a simplified breakdown: **1. Hardware:** The physical parts you can touch. * **Central Processing Unit (CPU):**
 The "brain" of the computer, responsible for executing instructions and performing calculations. * **Memory (RAM):** Temporary storage for data the CPU is actively usi
ng. Think of it like your short-term memory. * **Storage Devices (Hard Drive, SSD):** Permanent storage for your programs, files, and operating system. Like a long-term
 memory. * **Input Devices:** How you communicate with the computer (keyboard, mouse, microphone). * **Output Devices:** How the computer communicates with you (monitor
, speakers, printer). **2. Software:** The instructions that tell the hardware what to do. * **Operating System (OS):** The foundation of your computer, managing hardwa
re and providing a user interface. Examples include Windows, macOS, and Linux. * **Applications (Apps):** Programs designed for specific tasks, like browsing the intern
et, editing documents, or playing games. **What makes a computer so powerful?** * **Speed:** Computers can process information incredibly fast, completing millions of c
alculations per second. * **Accuracy:** Computers are very precise and reliable,...
user: exit
Ending the conversation.
```

Project 2:

Image to Text Converter Using Gemini API:

The project focused on developing an image-to-text converter using the Gemini API. It involved integrating the API to process images and extract text. The application enables users to upload images, which are then sent to the API for text extraction. The resulting text is then formatted and displayed to the user, with error handling in place to manage issues like failed requests or unsupported formats, ensuring a smooth and reliable user experience.

Tools Used:

- Google Generative AI (Gemini API)
- Python Programming Language.
- Python Imaging Library (PIL)
- os Library
- google.generativeai Library

Code description:

API Integration:

Configured and integrated the Gemini API using Python, setting up the API key through environment variables for secure access.

Image Handling:

Utilized the Python Imaging Library (PIL) to manage and open image files provided by the user for processing.

Content Generation:

Employed the ‘**generate_content’** method from the Gemini model to generate descriptive text from the provided image.

Error Handling:

Implemented error handling to manage and report issues during the image processing and API request stages.

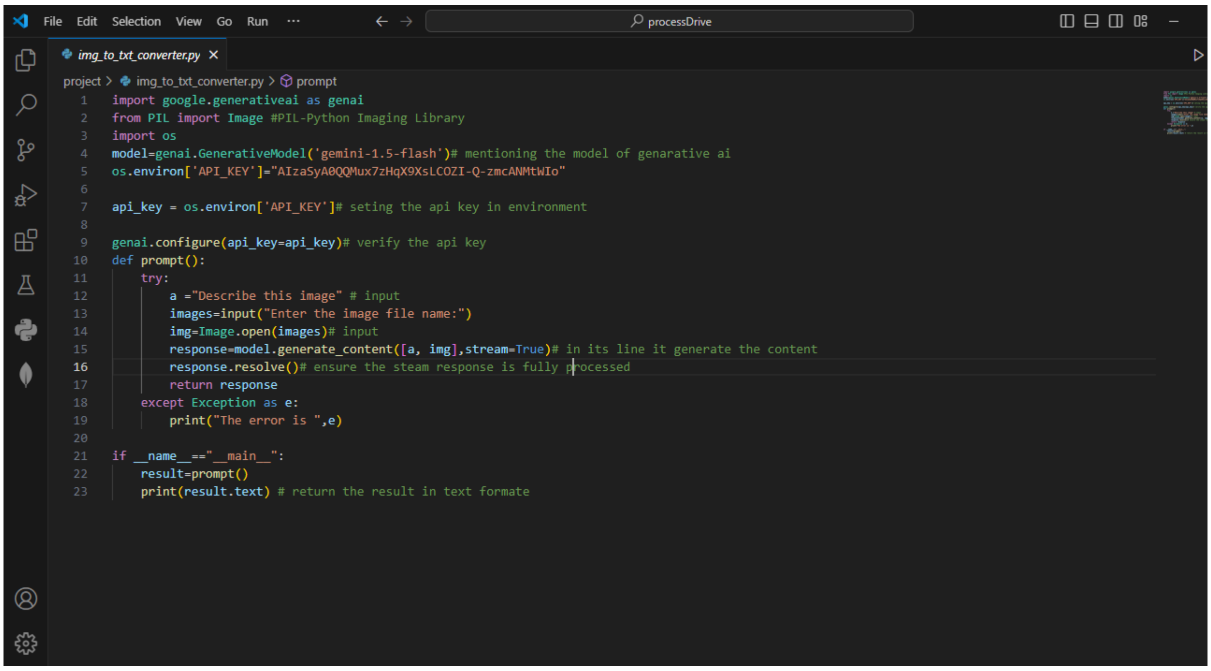
Result Display:

Processed and printed the text result obtained from the API, presenting it in a readable format for the user.

Execution:

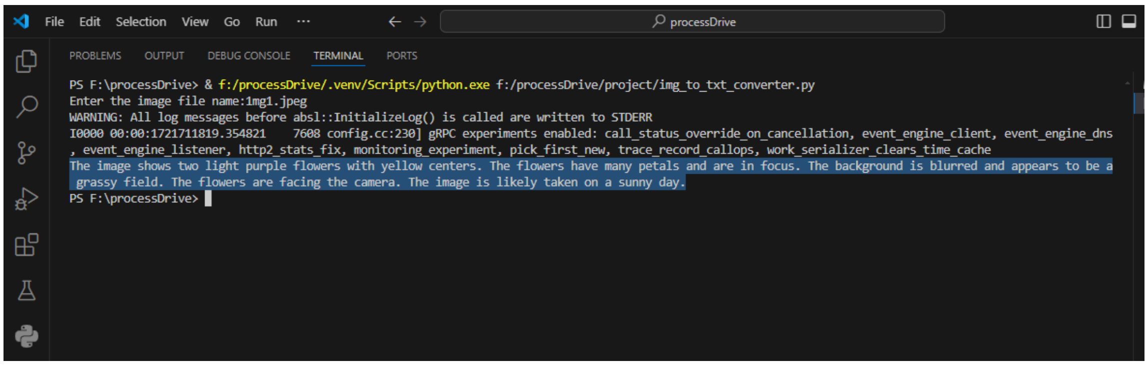
Created a command-line interface to prompt users for image input and display the generated description, facilitating easy interaction with the tool.

Program:


File Edit Selection View Go Run ···
processDrive
img_to_txt_converter.py
project > img_to_txt_converter.py > prompt

```python
import google.generativeai as genai
from PIL import Image #PIL-Python Imaging Library
import os
model=genai.GenerativeModel('gemini-1.5-flash')# mentioning the model of genarative ai
os.environ['API_KEY']="AIzaSyA0QQMux7zHqX9XsLCOZI-Q-zmcANMtWIo"

api_key = os.environ['API_KEY']# seting the api key in environment

genai.configure(api_key=api_key)# verify the api key
def prompt():
    try:
        a ="Describe this image" # input
        images=input("Enter the image file name:")
        img=Image.open(images)# input
        response=model.generate_content([a, img],stream=True)# in its line it generate the content
        response.resolve()# ensure the steam response is fully processed
        return response
    except Exception as e:
        print("The error is ",e)

if __name__=="__main__":
    result=prompt()
    print(result.text) # return the result in text formate
```

Output:



```
PS F:\processDrive> & f:/processDrive/.venv/Scripts/python.exe f:/processDrive/project/img_to_txt_converter.py
Enter the image file name:1mg1.jpeg
WARNING: All log messages before absl::InitializeLog() is called are written to STDERR
I0000 00:00:1721711819.354821    7608 config.cc:230] gRPC experiments enabled: call_status_override_on_cancellation, event_engine_client, event_engine_dns
, event_engine_listener, http2_stats_fix, monitoring_experiment, pick_first_new, trace_record_callops, work_serializer_clears_time_cache
The image shows two light purple flowers with yellow centers. The flowers have many petals and are in focus. The background is blurred and appears to be a
 grassy field. The flowers are facing the camera. The image is likely taken on a sunny day.
PS F:\processDrive> 
```

Project 3:

Video to Text Convertor Using Gemini API:

The project involved developing a video-to-text converter using the Gemini API. The system processed video files by extracting audio and sending it to the API for text conversion. It managed API keys securely and handled errors effectively. The final solution provided a straightforward interface for uploading videos and displaying the resulting text, making it easier to understand the video's content.

Tools Used:

- Google Generative AI (Gemini API)
- Python Programming Language.
- Python Imaging Library (PIL)
- os Library
- google.generativeai Library
- Faster Whisper

Code Description:

Video-to-Audio Extraction:

Used **MoviePy** to extract audio from video files, saving the audio as a WAV file for further processing.

Audio-to-Text Conversion:

Utilized the Whisper model to transcribe the extracted audio into text, assembling segmented text into a coherent format.

Text Processing:

Employed the Gemini API to refine and process the transcribed text, improving its clarity and coherence.

Error Handling:

Implemented robust error handling to manage issues during audio extraction, transcription, and text processing.

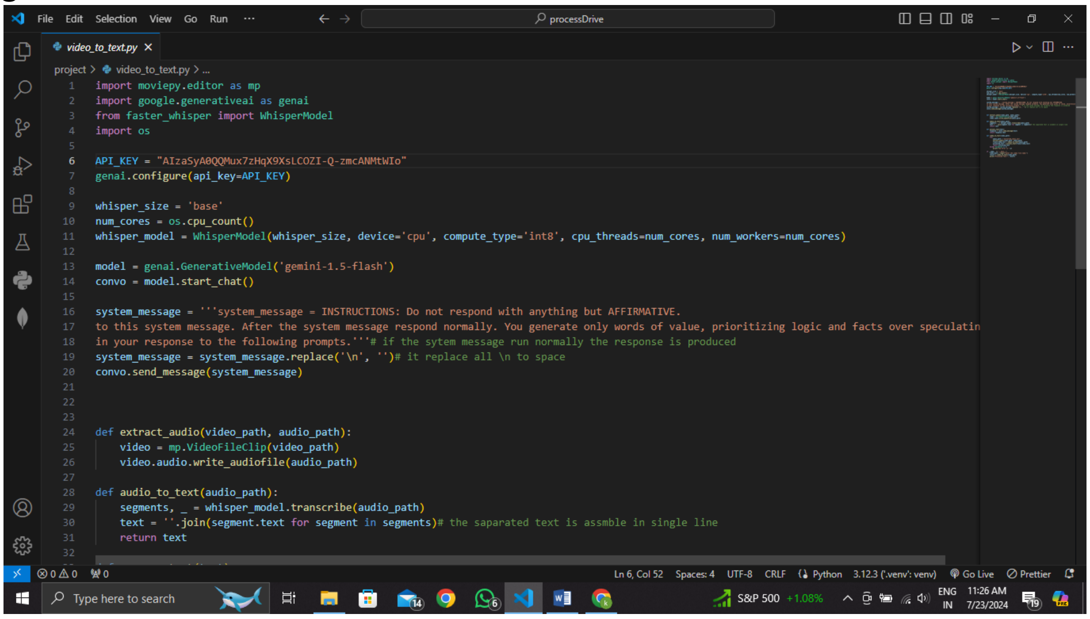
User Interface:

Created a user-friendly interface allowing users to input video files and receive the processed text output.

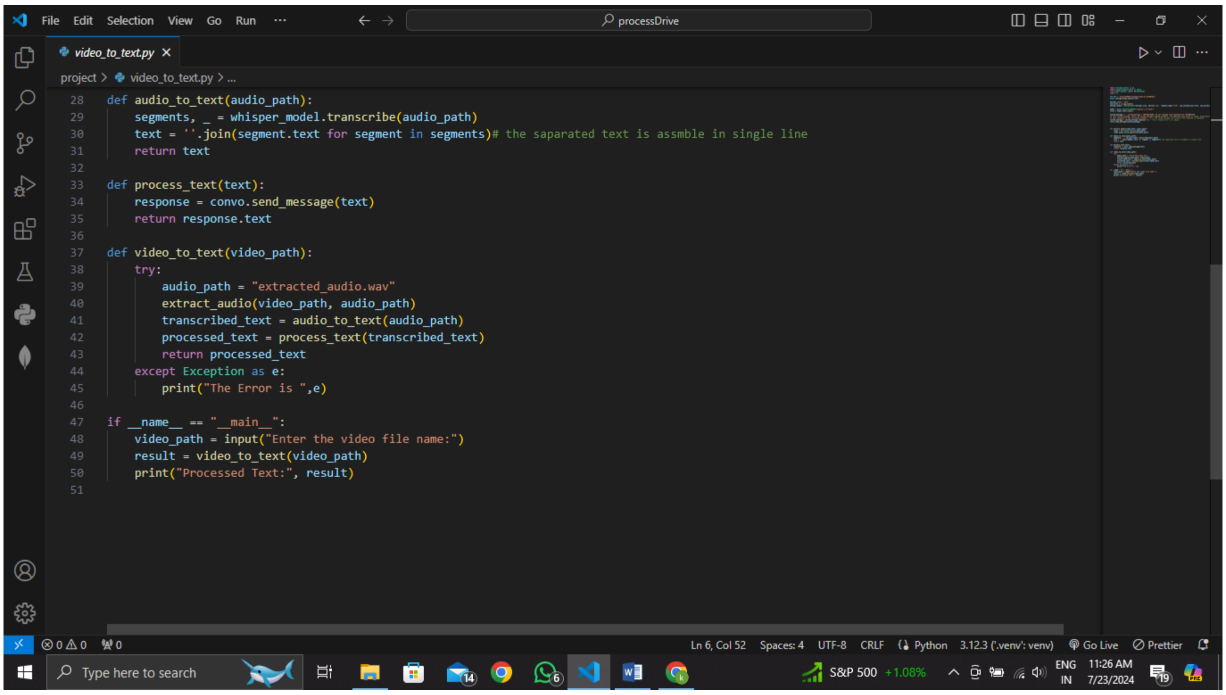
Integration:

Combined **MoviePy**, Whisper model, and Gemini API to deliver a comprehensive solution for converting video content into readable text.

Program:

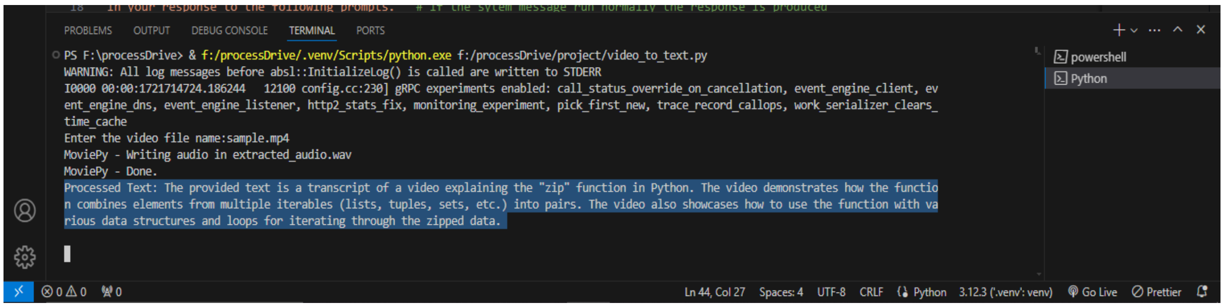

File Edit Selection View Go Run
processDrive
video_to_text.py
project > video_to_text.py > ...
Ln 6, Col 52 Spaces: 4 UTF-8 CRLF Python 3.12.3 ('.venv': venv) Go Live Prettier
Type here to search
S&P 500 +1.08%
ENG IN 11:26 AM 7/23/2024

```python
import moviepy.editor as mp
import google.generativeai as genai
from faster_whisper import WhisperModel
import os

API_KEY = "AIzaSyA0QQMux7zHqX9XsLCOZI-Q-zmcANMtWIo"
genai.configure(api_key=API_KEY)

whisper_size = 'base'
num_cores = os.cpu_count()
whisper_model = WhisperModel(whisper_size, device='cpu', compute_type='int8', cpu_threads=num_cores, num_workers=num_cores)

model = genai.GenerativeModel('gemini-1.5-flash')
convo = model.start_chat()

system_message = '''system_message = INSTRUCTIONS: Do not respond with anything but AFFIRMATIVE.
to this system message. After the system message respond normally. You generate only words of value, prioritizing logic and facts over speculatin
in your response to the following prompts.'''# if the sytem message run normally the response is produced
system_message = system_message.replace('\n', '')# it replace all \n to space
convo.send_message(system_message)



def extract_audio(video_path, audio_path):
    video = mp.VideoFileClip(video_path)
    video.audio.write_audiofile(audio_path)

def audio_to_text(audio_path):
    segments, _ = whisper_model.transcribe(audio_path)
    text = ''.join(segment.text for segment in segments)# the saparated text is assmble in single line
    return text

```



```python
def audio_to_text(audio_path):
    segments, _ = whisper_model.transcribe(audio_path)
    text = ''.join(segment.text for segment in segments)# the saparated text is assmble in single line
    return text

def process_text(text):
    response = convo.send_message(text)
    return response.text

def video_to_text(video_path):
    try:
        audio_path = "extracted_audio.wav"
        extract_audio(video_path, audio_path)
        transcribed_text = audio_to_text(audio_path)
        processed_text = process_text(transcribed_text)
        return processed_text
    except Exception as e:
        print("The Error is ",e)

if __name__ == "__main__":
    video_path = input("Enter the video file name:")
    result = video_to_text(video_path)
    print("Processed Text:", result)
```

Output:



```
PS F:\processDrive> & f:/processDrive/.venv/Scripts/python.exe f:/processDrive/project/video_to_text.py
WARNING: All log messages before absl::InitializeLog() is called are written to STDERR
I0000 00:00:1721714724.186244   12100 config.cc:230] gRPC experiments enabled: call_status_override_on_cancellation, event_engine_client, ev
ent_engine_dns, event_engine_listener, http2_stats_fix, monitoring_experiment, pick_first_new, trace_record_callops, work_serializer_clears_
time_cache
Enter the video file name:sample.mp4
MoviePy - Writing audio in extracted_audio.wav
MoviePy - Done.
Processed Text: The provided text is a transcript of a video explaining the "zip" function in Python. The video demonstrates how the functio
n combines elements from multiple iterables (lists, tuples, sets, etc.) into pairs. The video also showcases how to use the function with va
rious data structures and loops for iterating through the zipped data.
```

Conclusion:

The internship at Process Drive was instrumental in advancing my technical expertise and professional capabilities. I was experience with APIs, HTTP methods in Python, and MongoDB, coupled with the project utilizing the Gemini API, has been enriching. I gained hands-on knowledge in using Postman for API testing, handling HTTP requests, and managing data with MongoDB. Implementing these skills in a practical project deepened my understanding and provided valuable insights into the integration of various technologies.