FaceRecoMain.py

```python
import cv2 as cv
import  numpy as np
import FaceRecoApi as api
import os
import json

class NoKnownFaceException(Exception):
    pass

names = []
known_encoded_faces = []

# set font parameters
font = cv.FONT_HERSHEY_DUPLEX
font_scale = 0.6
font_color = (0, 0, 0)
font_thickness = 1

# the process count - this is a check to process the faces in a frame
# after a specific iteration
process_count = 0
n = 15

# set the named window
window_name = 'FaceReco'
cv.namedWindow(window_name, cv.WINDOW_NORMAL)

# set desired window size
window_width = 640
window_height = 480
cv.resizeWindow(window_name, window_width, window_height)

# capturing from webcam
capture_face_from_webcam = True
capture = cv.VideoCapture(0)

while capture_face_from_webcam:
    process_this_frame = True
    while process_this_frame:
        #taking one frame at a time
        status, frame = capture.read()

        # flipping the frame horizontally
        flipped_frame = cv.flip(frame,1)

        # increase the process count
        process_count += 1

        # checking if we process this frame or not
        if process_count % n == 0:
            # resize the frame to 1/4th size for faster search preocess
            resized_frame = cv.resize(flipped_frame, (0, 0), fx = 1, fy = 1)

            # convert the image from BGR to RGB
            rgb_image = cv.cvtColor(resized_frame, cv.COLOR_BGR2RGB)
            # convert the image from BGR to RGB
            detected_faces = api.detect_face(rgb_image)
            # print(faces)

            # if any face is detected
            if len(detected_faces) > 0:
                # Detecting faces
                landmarks = api.face_landmarks(rgb_image, detected_faces)

                # encoding faces and storing them encoded_faces list
                encoded_faces = np.array(api.face_encode(rgb_image, landmarks))

                # Comparing faces
                face_matches = []
                known_faces_index = []
                matched_faces,unmatched_faces, unmatched_encodings =
api.compare_faces(detected_faces,
                                                                encoded_faces)

                # show the matched faces and their names, unknown faces in the
frame
                known_face = []
                unknown_face = []
                if len(matched_faces) > 0:
                    for matched_name, matched_face in matched_faces.items():
                        known_name = matched_name
                        known_face = matched_face
                        cv.rectangle(flipped_frame,
                                    (known_face.left(), known_face.top()),
                                    (known_face.right(), known_face.bottom()),
                                    (0, 255, 0), 2)
                        cv.rectangle(flipped_frame,
                                    (known_face.left(), known_face.bottom()),
                                    (known_face.right(), known_face.bottom() +
50),

                                    (0, 255, 0), cv.FILLED)
                        cv.putText(flipped_frame, known_name, (known_face.left() +
5,
                                                        known_face.bottom() + 35),
                                    font, font_scale, font_color, font_thickness)
                elif len(unmatched_faces) > 0:
                    api.process_unknown_faces(flipped_frame, unmatched_faces,
unmatched_encodings)
                    for unknown_face in unmatched_faces:
                        cv.rectangle(flipped_frame,
                                    (unknown_face.left(), unknown_face.top()),
                                    (unknown_face.right(), unknown_face.bottom()),
                                    (0, 0, 255), 2)
                else:
                    for face in detected_faces:
                        cv.rectangle(flipped_frame, (face.left(), face.top()),
                                        (face.right(), face.bottom()),
                                        (255, 0, 0), 2)
                        cv.rectangle(flipped_frame,
                                            (face.left(), face.bottom()),
                                            (face.right(), face.bottom() + 50),
                                            (255, 0, 0), cv.FILLED)
                        cv.putText(flipped_frame, 'No Face Added', (face.left() +
5,
                                                                    face.bottom() +
35),
                                            font, font_scale, (255,255,255),
                                            font_thickness)

            process_this_frame = False

    # if the process iteration is complete reset the process count
    if process_count >= n:
        process_count = 0

    # show the processed or unprocessed frame
    cv.imshow(window_name, flipped_frame)

    # if key 'q' is pressed it will stop capturing from the webcam
    if cv.waitKey(1) & 0xFF == ord('q'):
        capture_face_from_webcam = False

#end of capture
capture.release()
cv.destroyAllWindows()
```

faces.py

```
import sys
import os
import json
import cv2
import numpy as np
from api import detect_face, face_landmarks, face_encode

# InvalidFlagError - a custom exception
class InvalidFlagError(Exception):
    pass
class InvalidInputError(Exception):
    pass
class NoFaceFoundError(Exception):
    pass

# get the path to the known_faces.json
def get_json_path(status):
    # Get the current directory of the script
    current_directory = os.path.dirname(os.path.abspath(__file__))
```

```python
    # Construct the path to "test.file"
    if status == 'k':
        path = os.path.join(current_directory, '..',
                                'assets', 'known_faces.json')
    elif status == 'u':
        path = os.path.join(current_directory, '..',
                                'assets', 'unknown_faces.json')

    return path

# list names in known_faces.json
def list_known_faces():
    status = 'k'
    path = get_json_path(status)

    with open(path, "r") as json_read:
        data = json.load(json_read)

    if len(data) == 0:
        raise NoFaceFoundError("No known faces")
    # print(data)
    print("Known Faces are: ")
    list = [
        value for item in data
        for key, value in item.items() if key == "Name"
        ]
    # print(list)
import sys
import os
import json
import cv2
import numpy as np
from FaceRecoApi.api import detect_face, face_landmarks, face_encode

# InvalidFlagError - a custom exception
class InvalidFlagError(Exception):
    pass
class InvalidInputError(Exception):
    pass
class NoFaceFoundError(Exception):
    pass

# get the path to the known_faces.json
def get_json_path(status):
    # Get the current directory of the script
    current_directory = os.path.dirname(os.path.abspath(__file__))

    # Construct the path to "test.file"
    if status == 'k':
        path = os.path.join(current_directory,
                                'assets', 'known_faces.json')
    elif status == 'u':
        path = os.path.join(current_directory,
                                'assets', 'unknown_faces.json')
    return path

# list names in known_faces.json
def list_known_faces():
    status = 'k'
    path = get_json_path(status)

    with open(path, "r") as json_read:
        data = json.load(json_read)

    if len(data) == 0:
        raise NoFaceFoundError("No known faces")
    # print(data)
    print("Known Faces are: ")
    list = [
        value for item in data
        for key, value in item.items() if key == "Name"
        ]
    # print(list)
    for name in list:
        print(name)


# Define functions for each flag

# add a face as known face
def add_face(param, status):
    """
        add_face expects one element in param
        - the path of the image that will be added
        this function can only add one person's picture at a time so
        - no directory will be accepted as input
        - if the image has multiple faces it will only add
          the first detected face, so specify the name carefully
    """

    if status == 'k':
        print("Adding face...")
        # check if path in the param exists
        if not os.path.exists(param):
            raise FileNotFoundError
        file = cv2.imread(param)

        # ask to enter the name
        face_name = None
        face_name = input("Enter name of the person:")
        if not face_name.strip():
            raise InvalidInputError("Please specify a name")

        # detect face in the image
        face = detect_face(file)

        # find facial landmarks
        landmarks = face_landmarks(file, face)
        # encode the face
        encoded_face = np.array(face_encode(file, landmarks))
        encoded_face = encoded_face.tolist()

        # if no face is detected, no face will be encoded
        if len(encoded_face) == 0:
            raise NoFaceFoundError("No face found")

        print("Face encoded successfully")

        # add the encoding in json
        data = {
            "Name": face_name,
            "Encoding": encoded_face[0]
        }
    elif status == 'u':
        data = {
            "Encoding": param
        }

    # get file path of the json file

    path = get_json_path(status)

    # open json files
    json_append = open(path, "a")
    json_truncate = open(path, "ab")

    # check if json file is empty
    if os.stat(path).st_size == 0:
        json_append.write("[\n")
    if os.stat(path).st_size == 2:
        json_truncate.seek(-1, 2)
        json_truncate.truncate()
        json_truncate.truncate()
    else:
        json_truncate.seek(-1, 2)
        json_truncate.truncate()
        json_append.write(",\n")

    # serialize face data in json file
    json.dump(data, json_append, indent=4, separators=(',', ':'))
    json_append.write("\n]")
    if status == 'k':
        print("Face data added successfully.")
        print("Person recognized as: ", face_name)

    # close json files
    json_append.close()
    json_truncate.close()


# delete a known face
def delete_face(name, status):
    """
        delete_face expects one element in param
        - the exact name of the person whose data will be deleted

    """

    print("Deleting face", name,"...")

    # get file path of the json file
    path = get_json_path(status)

    if os.stat(path).st_size == 2:
        print("No faces to delete")
        return

    with open(path, "r") as json_read:
        data = json.load(json_read)

    new_data = [item for item in data if item.get("Name") != name]
    if new_data == data:
        raise NoFaceFoundError("No face found named", name)

    with open(path, "w") as json_write:
        json.dump(new_data, json_write, indent=4)
        print("Deleted face successfully")

# execute flag action
def execute_flag(flag, param):
    # Define flag-function mapping
    flag_actions = {
        '-a': lambda: add_face(param, 'k'),
        '-ak': lambda: add_face(param, 'k'),
        '-au': lambda: add_face(param, 'u'),
        '-d': lambda: delete_face(param, 'k'),
        '-l': list_known_faces
    }

    if flag in flag_actions:
        flag_actions[flag]()
    elif flag == "-ad":
        raise InvalidFlagError("You can do one task at a time")
    elif flag == None:
        raise InvalidFlagError(
            "Please enter a flag: -a to add faces, -d to delete faces"
            )
    else:
        raise InvalidFlagError("Invalid Flag")

# Process the arguments - separate the flag and specified path/name
def process_arguments(arguments):
    for arg in arguments:
        if arg.startswith('-'):
            flag = arg
        else:
            param = arg
```

```python
    # if just list the known faces no parameter is required
    if flag == '-l':
        param = " "

    return flag, param

def main(arguments):
    try:
        flag = None
        param = None
        flag, param = process_arguments(arguments)
        execute_flag(flag, param)
    except InvalidFlagError as e:
        print(str(e))
    except FileNotFoundError as e:
        print(str(e))
    except InvalidInputError as e:
        print(str(e))
    except NoFaceFoundError as e:
        print(str(e))


if __name__ == "__main__":
    arguments = sys.argv[1:]
    main(arguments)
```