**CHAPTER 1**

**1.INRODUCTION**

OMR image scanning system using Python OpenCV and NumPy is the perfect solution for organizations looking to automate their document scanning process. This system uses powerful computer vision algorithms to quickly and accurately scan documents, eliminating the need for manual scanning. With this system, organizations can save time and money while ensuring accuracy in their document scanning process. Additionally, this system provides a secure way to store scanned documents, making it ideal for sensitive data handling.

**CHAPTER 2**

**2.PROBLEM DEFINITION**

OMR scanner is a complex task that requires a clear understanding of the problem. A comprehensive problem definition for an OMR scanner should include the types of documents to be scanned, the accuracy and speed requirements, and any other special requirements that must be met. This problem definition will provide a foundation for the development team to build upon as they design and develop a solution to meet their customer’s needs.The Existing system There is no need to man power,can be get mutible images same time.

**CHAPTER 3**

**3.SYSTEM STUDY**

**3.1 EXISTING SYSTEM**

Hard to scan the marking if the shades on the OMR sheet are not dark enough.

 Answer sheets must be placed properly in the tray otherwise, it would un-number the scanning.

OMR scanning hardware is costly.

If the answer is incomplete, the scanner won’t be able to read it.

**3.1.1.DISADVANTAGES**

In the project only can process the image file such a jpeg,jpg,png ect.

so there need manual effort for tack as picture the omr sheet.

**3.2 PROPOSED SYSTEM**

A single OMR solution helps scan and read OMR sheets in less time.

It can even read images scanned in tilted position.

 It can scan multiple pages in quick time.

Installation is easy. Even novice users can operate the system with ease.

**3.2.1.ADVANTAGES**

To avoid lot of manual errors such as counting mistake during evaluation.

To evaluate multiple answer sheets and generate a single report.

**CHAPTER 4**

**4.SYSTEM RQUIREMENT SPECIFICATION**

**4.1 HARDWARE REQUIREMENT**

Processor                           :Dual Core processor 2.6.0 GHZ.

RAM                                  :1GB.

Hard disk                           : 160GB.

Compact disk                     :650MB.

Keyboard                           :Standard Keyboard.

Monitor                              :15 inch colour monitor.

**4.2 SOFTWARE REQUIREMENTS**

Operating system           : Windows OS

Programming language  : python

IDLE                              : VS code

Modules                          : Open-CV ,numpy

**CHAPTER 5**

**5.1 DATA FLOW DIAGRAM**

**CHAPTER 6**

**6.SYSTEM IMPLEMENTATION**

**6.1 MODULES**

Module Loads Input Image

Module Normalizes Image

Module Finds Contours

Module Finds final

**6.2 MODULE DESCRIPTIONS**

**AUTHENTICATION MODULE:**

**Module Loads Input**

* first getting input image from user.
* such a png,jpg,jepg formeats.
* then store and converting the image to gray scale.

**Module Normalizes Image**

* In This Process Adjusd The Ratio Of The Image.
* Only If Image Is Not Proper Ratio.

**Module Contours**

* this stage is to finding the contors in the images.
* Finds corners among all contours.

**Module Final**

* Scans each line for the marked alternative.
* Appliespers pective transform to get a bird's eye view
* Finds 'outmost' points of all corners.

**CHAPTER 7**

**7.SOFTWARE ENVIRONMENT**

**PYTHON**

The software environment for a Python project generally refers to the tools, libraries, and dependencies that are required to run and develop a specific Python project. It may vary depending on the specific requirements of the project, but here are some common components of a typical Python project software environment:

**Python Interpreter**: The Python interpreter is the core component of the Python environment and is required to execute Python code. You can choose the appropriate version of Python based on the project requirements and install it on your development environment.

**Virtual Environment:** A virtual environment is a self-contained Python environment that allows you to isolate project-specific dependencies and avoid conflicts with other Python projects. You can create a virtual environment for your project using tools like virtualenv or conda, and install project-specific dependencies within the virtual environment.

**Code Editor or Integrated Development Environment (IDE):** A code editor or an IDE is a software tool that provides features like syntax highlighting, code completion, and debugging to help you write and manage your Python code. Popular Python IDEs include PyCharm, VSCode, and Jupyter Notebook, among others.

**Package Manager:** A package manager is used to install, manage, and update Python libraries and dependencies required by the project. The most commonly used package manager for Python is pip, which allows you to install packages from the Python Package Index (PyPI). You can define the project's dependencies in a requirements.txt or a Pipfile, which lists all the required packages and their versions.

**Version Control System:** Version control is essential for managing changes in your project's source code and collaborating with other developers. Git is a popular version control system used in the Python community. You can use Git to track changes, create branches, and merge changes from multiple contributors.

**Documentation and Testing Tools:** Documentation is crucial for maintaining and understanding the project codebase. Tools like Sphinx or MkDocs can be used to generate project documentation. Additionally, testing is essential for ensuring the quality and reliability of your code. Python has built-in testing frameworks like unittest and third-party libraries like pytest that can be used for writing and running tests.

**Project-specific Libraries and Dependencies:** Depending on the requirements of your project, you may need to install and manage additional Python libraries or dependencies. These may include libraries for data analysis, machine learning, web development, or any other specific functionality required by your project.

These are some of the common components of a Python project software environment. It's important to tailor the software environment to the specific requirements of your project and follow best practices for managing dependencies, version control, documentation, and testing to ensure a smooth development process.

**Need of PYTHON:**

Python, a powerful high-level programming language, is utilized in a wide range of applications and industries due to its versatility, simplicity, and extensive ecosystem of libraries. Here are some common use cases and advantages of using Python:

Web Development: Python is commonly used for web development, with popular frameworks such as Django, Flask, and Pyramid providing robust tools for building dynamic websites, APIs, and web applications.

Data Science and Analytics: Python is widely used in the field of data science and analytics due to its rich ecosystem of libraries, including NumPy, Pandas, Matplotlib, and scikit-learn, which enable data manipulation, analysis, visualization, and machine learning.

Scientific Computing: Python is a preferred choice for scientific computing, including numerical simulations, computational modeling, and data analysis in various domains such as physics, biology, and chemistry, due to its extensive libraries like SciPy and NumPy.

**Comments in PYTHON**

In Python, comments are used to add notes or explanations in your code, and they are ignored by the Python interpreter when the code is executed. Comments are useful for providing documentation, explaining complex logic, and making your code more readable for others or for yourself in the future.

There are two types of comments in Python:

**1.Single-line comments:** Single-line comments start with the '#' character and continue until the end of the line. Anything after the '#' character on the same line is considered a comment.

# This is a single-line comment

x = 5 # This is a comment about assigning the value 5 to variable x

**2.Multi-line comments:** Multi-line comments are enclosed in triple quotes (''' ''') or double quotes (""" """) and can span across multiple lines. They are often used for longer comments or documentation.

'''

This is a multi-line comment

It can span across multiple lines

'''

"""

This is also a multi-line comment

It can be enclosed in double quotes as well

"""

It's good practice to use comments in your Python code to provide explanations, clarify your code, and make it more understandable to others and yourself. However, it's important to avoid over-commenting and keep your comments concise and relevant to the code.

**PYTHON Variables**

In Python, variables are used to store data values. A variable is a named location in the computer's memory where a value can be stored and retrieved. Here are some key points about variables in Python:

**Variable declaration:** Variables are declared by assigning a value to a name using the '=' operator. The name of a variable should be a combination of letters (lowercase or uppercase), numbers, and underscore (\_), and it should start with a letter or an underscore

x = 5 # Assigns the value 5 to the variable x

y = "Hello" # Assigns the string "Hello" to the variable y

**Data types:** Python is a dynamically typed language, which means that the data type of a variable is determined automatically based on the value assigned to it. Some common data types in Python include integers, floats, booleans, and strings.

**# Integers**

num1 = 10

num2 = -5

**# Floats**

num3 = 3.14

num4 = -0.5

**# Booleans**

is\_true = True

is\_false = False

**# Strings**

text1 = "Hello, world!"

text2 = 'Python is awesome!'

**Variable assignment:** You can assign a new value to a variable at any point in your code, and the variable will store the new value.

x = 5 # Assigns the value 5 to x

x = x + 3 # Updates the value of x to 8

**Variable usage:** Once a value is assigned to a variable, you can use the variable in expressions or statements throughout your code.

x = 5

y = 3

sum = x + y # Adds the values of x and y and stores the result in the variable sum

print(sum) # Prints the value of sum, which is 8

**PYTHON openCV and numyp**

OpenCV (Open Source Computer Vision) and NumPy (Numeric Python) are two popular Python libraries used for image processing, computer vision, and numerical computing tasks. Let's take a brief look at each of these libraries:

**7.2 openCV**

**OpenCV:** OpenCV is a powerful open-source library that provides a wide range of functions for image processing, computer vision, and machine learning. It is widely used in various applications such as image and video processing, object detection, facial recognition, and robotics

import cv2

# Load an image from file

image = cv2.imread("image.jpg")

# Display the loaded image

cv2.imshow("Image", image)

# Perform image processing operations

gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

edges = cv2.Canny(gray, 50, 150)

# Display the processed image

cv2.imshow("Edges", edges)

# Wait for a key event and close all windows

cv2.waitKey(0)

cv2.destroyAllWindows()

OpenCV provides a large number of functions for tasks such as image manipulation, filtering, feature extraction, object tracking, and camera calibration, making it a valuable tool for computer vision and image processing tasks.

**7.2 NUMPY**

**NumPy:** NumPy is a powerful numerical computing library for Python that provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays. NumPy is widely used in scientific computing, data analysis, and machine learning applications.

import numpy as np

# Create a NumPy array

a = np.array([1, 2, 3, 4, 5])

# Perform array operations

b = a \* 2

c = np.sqrt(a)

# Perform matrix operations

matrix1 = np.array([[1, 2], [3, 4]])

matrix2 = np.array([[5, 6], [7, 8]])

product = np.dot(matrix1, matrix2)

# Perform mathematical functions

mean = np.mean(a)

std = np.std(a)

max\_value = np.max(a)

print(b)

print(c)

print(product)

print(mean)

print(std)

print(max\_value)

It provides efficient array operations, mathematical functions, and tools for working with arrays, making it a key tool for numerical computing in Python.

OpenCV and NumPy are often used together in computer vision and image processing tasks. NumPy provides efficient array operations and mathematical functions that can be used to manipulate images, and OpenCV provides a wide range of image processing functions to perform various tasks on images and videos. Together, these libraries provide a powerful toolset for working with images and performing computer vision tasks in Python.

**CHAPTER 8**

**8.SYSTEM TESTING**

**8.1 INTRODUCTION**

**8.2 STRATEGIC APPROACH TO SOFTWARE TESTING**

**8.3 Unit Testing**

**1. WHITE BOX TESTING**

**2. BASIC PATH TESTING**

**3. CONDITIONAL TESTING**

.

**4. DATA FLOW TESTING**

**5. LOOP TESTING**

**CHAPTER 9**

**9.CODING**

import argparse

import cv2

import numpy as np

TRANSF\_SIZE = 512

N\_QUESTIONS = 10

ANSWER\_SHEET\_WIDTH = 740

ANSWER\_SHEET\_HEIGHT = 1049

ANSWER\_PATCH\_HEIGHT = 50

ANSWER\_PATCH\_HEIGHT\_WITH\_MARGIN = 80

ANSWER\_PATCH\_LEFT\_MARGIN = 200

ANSWER\_PATCH\_RIGHT\_MARGIN = 90

FIRST\_ANSWER\_PATCH\_TOP\_Y = 200

ALTERNATIVE\_HEIGHT = 50

ALTERNATIVE\_WIDTH = 50

ALTERNATIVE\_WIDTH\_WITH\_MARGIN = 100

def calculate\_contour\_features(contour):

    moments = cv2.moments(contour)

    return cv2.HuMoments(moments)

def calculate\_corner\_features():

    corner\_img = cv2.imread('img/corner.png')

    corner\_img\_gray = cv2.cvtColor(corner\_img, cv2.COLOR\_BGR2GRAY)

    contours, hierarchy = cv2.findContours(

        corner\_img\_gray, cv2.RETR\_TREE, cv2.CHAIN\_APPROX\_SIMPLE)

    if len(contours) != 2:

        raise RuntimeError(

            'Did not find the expected contours when looking for the corner')

    corner\_contour = next(ct

                          for i, ct in enumerate(contours)

                          if hierarchy[0][i][3] != -1)

    return calculate\_contour\_features(corner\_contour)

def normalize(im):

    im\_gray = cv2.cvtColor(im, cv2.COLOR\_BGR2GRAY)

    blurred = cv2.GaussianBlur(im\_gray, (3, 3), 0)

    return cv2.adaptiveThreshold(

        blurred, 255, cv2.ADAPTIVE\_THRESH\_GAUSSIAN\_C, cv2.THRESH\_BINARY, 77, 10)

def get\_approx\_contour(contour, tol=.01):

    epsilon = tol \* cv2.arcLength(contour, True)

    return cv2.approxPolyDP(contour, epsilon, True)

def get\_contours(image\_gray):

    contours, \_ = cv2.findContours(

        image\_gray, cv2.RETR\_TREE, cv2.CHAIN\_APPROX\_SIMPLE)

    return map(get\_approx\_contour, contours)

def get\_corners(contours):

    corner\_features = calculate\_corner\_features()

    return sorted(

        contours,

        key=lambda c: features\_distance(

                corner\_features,

                calculate\_contour\_features(c)))[:4]

def get\_bounding\_rect(contour):

    rect = cv2.minAreaRect(contour)

    box = cv2.boxPoints(rect)

    return np.int0(box)

def features\_distance(f1, f2):

    return np.linalg.norm(np.array(f1) - np.array(f2))

def draw\_point(point, img, radius=5, color=(0, 0, 255)):

    cv2.circle(img, tuple(point), radius, color, -1)

def get\_centroid(contour):

    m = cv2.moments(contour)

    x = int(m["m10"] / m["m00"])

    y = int(m["m01"] / m["m00"])

    return (x, y)

def sort\_points\_counter\_clockwise(points):

    origin = np.mean(points, axis=0)

    def positive\_angle(p):

        x, y = p - origin

        ang = np.arctan2(y, x)

        return 2 \* np.pi + ang if ang < 0 else ang

    return sorted(points, key=positive\_angle)

def get\_outmost\_points(contours):

    all\_points = np.concatenate(contours)

    return get\_bounding\_rect(all\_points)

def perspective\_transform(img, points):

    source = np.array(

        points,

        dtype="float32")

    dest = np.array([

        [TRANSF\_SIZE, TRANSF\_SIZE],

        [0, TRANSF\_SIZE],

        [0, 0],

        [TRANSF\_SIZE, 0]],

        dtype="float32")

    transf = cv2.getPerspectiveTransform(source, dest)

    warped = cv2.warpPerspective(img, transf, (TRANSF\_SIZE, TRANSF\_SIZE))

    return warped

def sheet\_coord\_to\_transf\_coord(x, y):

    return list(map(lambda n: int(np.round(n)), (

        TRANSF\_SIZE \* x / ANSWER\_SHEET\_WIDTH,

        TRANSF\_SIZE \* y / ANSWER\_SHEET\_HEIGHT

    )))

def get\_question\_patch(transf, question\_index):

    tl = sheet\_coord\_to\_transf\_coord(

        ANSWER\_PATCH\_LEFT\_MARGIN,

        FIRST\_ANSWER\_PATCH\_TOP\_Y + ANSWER\_PATCH\_HEIGHT\_WITH\_MARGIN \* question\_index

    )

    br = sheet\_coord\_to\_transf\_coord(

        ANSWER\_SHEET\_WIDTH - ANSWER\_PATCH\_RIGHT\_MARGIN,

        FIRST\_ANSWER\_PATCH\_TOP\_Y +

        ANSWER\_PATCH\_HEIGHT +

        ANSWER\_PATCH\_HEIGHT\_WITH\_MARGIN \* question\_index

    )

    return transf[tl[1]:br[1], tl[0]:br[0]]

def get\_question\_patches(transf):

    for i in range(N\_QUESTIONS):

        yield get\_question\_patch(transf, i)

def get\_alternative\_patches(question\_patch):

    for i in range(5):

        x0, \_ = sheet\_coord\_to\_transf\_coord(ALTERNATIVE\_WIDTH\_WITH\_MARGIN \* i, 0)

        x1, \_ = sheet\_coord\_to\_transf\_coord(ALTERNATIVE\_WIDTH +

                                            ALTERNATIVE\_WIDTH\_WITH\_MARGIN \* i, 0)

        yield question\_patch[:, x0:x1]

def draw\_marked\_alternative(question\_patch, index):

    cx, cy = sheet\_coord\_to\_transf\_coord(

        ALTERNATIVE\_WIDTH \* (2 \* index + .5),

        ALTERNATIVE\_HEIGHT / 2)

    draw\_point((cx, cy), question\_patch, radius=5, color=(255, 0, 0))

def get\_marked\_alternative(alternative\_patches):

    means = list(map(np.mean, alternative\_patches))

    sorted\_means = sorted(means)

    # Simple heuristic

    if sorted\_means[0]/sorted\_means[1] > .7:

        return None

    return np.argmin(means)

def get\_letter(alt\_index):

    return ["A", "B", "C", "D", "E"][alt\_index] if alt\_index is not None else "N/A"

def get\_answers(source\_file):

    im\_orig = cv2.imread(source\_file)

    im\_normalized = normalize(im\_orig)

    contours = get\_contours(im\_normalized)

    corners = get\_corners(contours)

    cv2.drawContours(im\_orig, corners, -1, (0, 255, 0), 3)

    outmost = sort\_points\_counter\_clockwise(get\_outmost\_points(corners))

    color\_transf = perspective\_transform(im\_orig, outmost)

    normalized\_transf = perspective\_transform(im\_normalized, outmost)

    answers = []

    for i, q\_patch in enumerate(get\_question\_patches(normalized\_transf)):

        alt\_index = get\_marked\_alternative(get\_alternative\_patches(q\_patch))

        if alt\_index is not None:

            color\_q\_patch = get\_question\_patch(color\_transf, i)

            draw\_marked\_alternative(color\_q\_patch, alt\_index)

        answers.append(get\_letter(alt\_index))

    return answers, color\_transf

def main():

    parser = argparse.ArgumentParser()

    parser.add\_argument(

        "--input",

        help="Input image filename",

        required=True,

        type=str)

    parser.add\_argument(

        "--output",

        help="Output image filename",

        type=str)

    parser.add\_argument(

        "--show",

        action="store\_true",

        help="Displays annotated image")

    args = parser.parse\_args()

    answers, im = get\_answers(args.input)

    for i, answer in enumerate(answers):

        print("Q{}: {}".format(i + 1, answer))

    if args.output:

        cv2.imwrite(args.output, im)

        print('Wrote image to {}.'.format(args.output))

    if args.show:

        cv2.imshow('Annotated image', im)

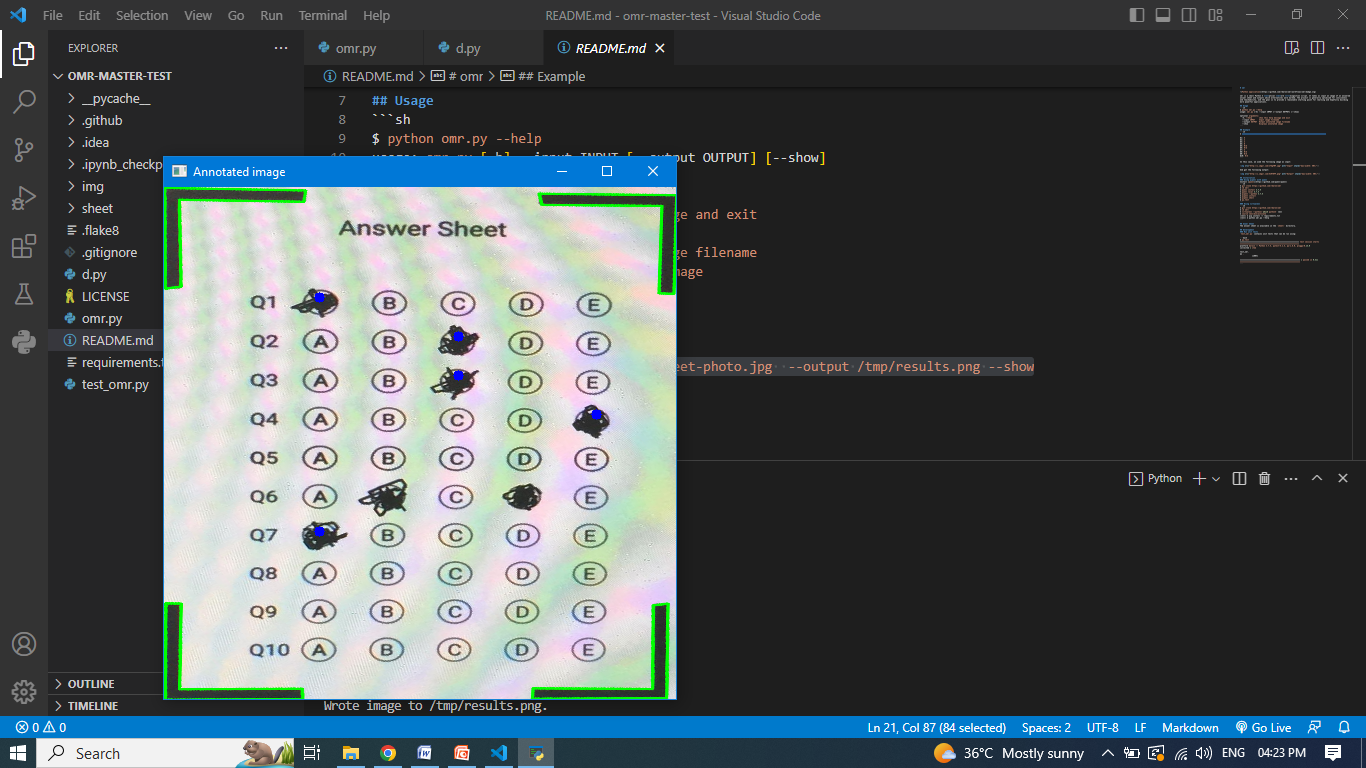
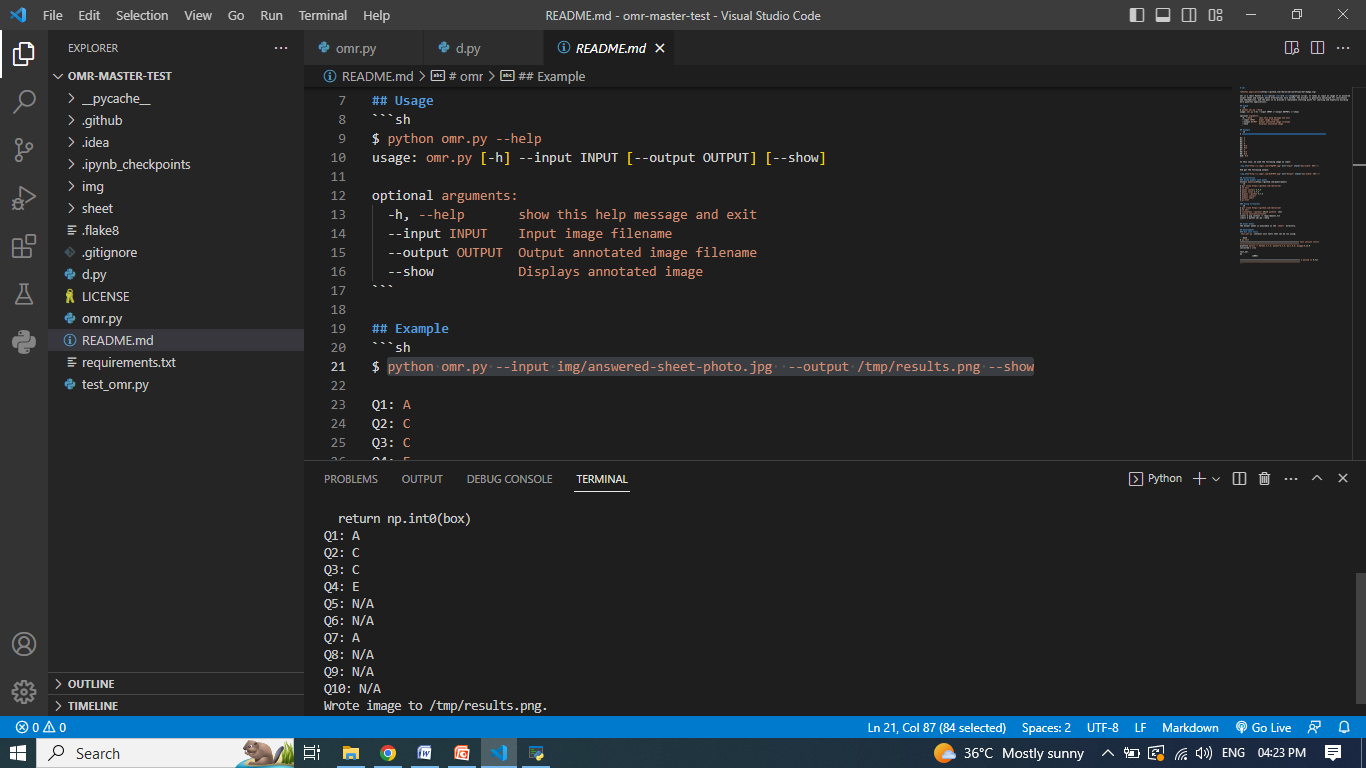
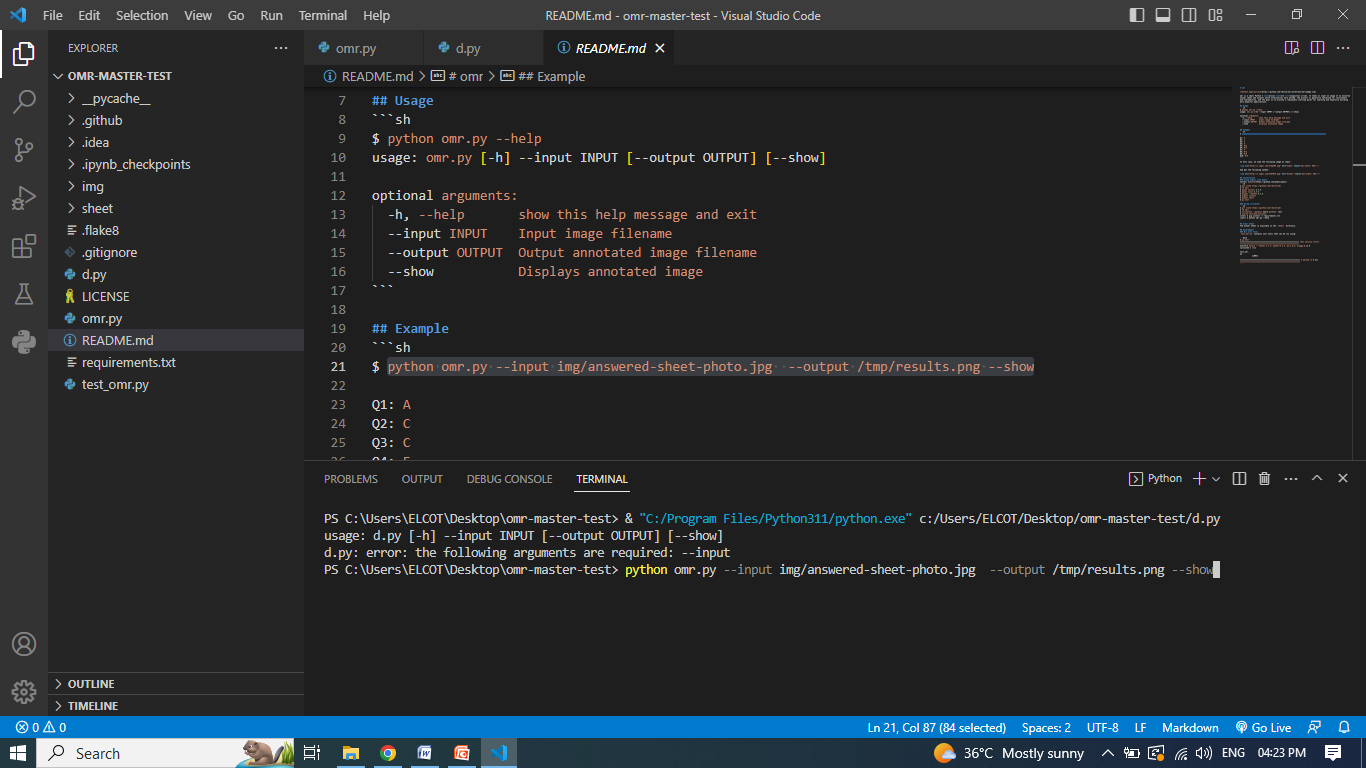
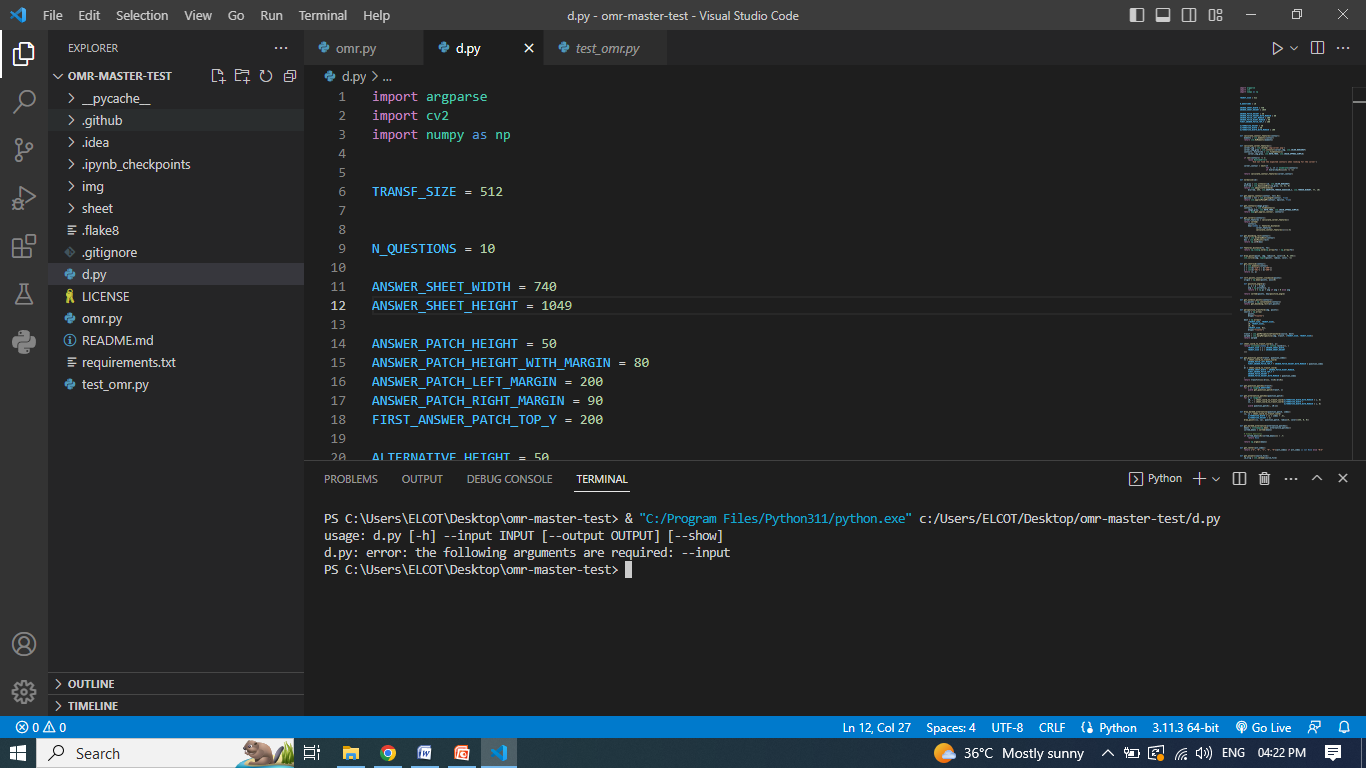
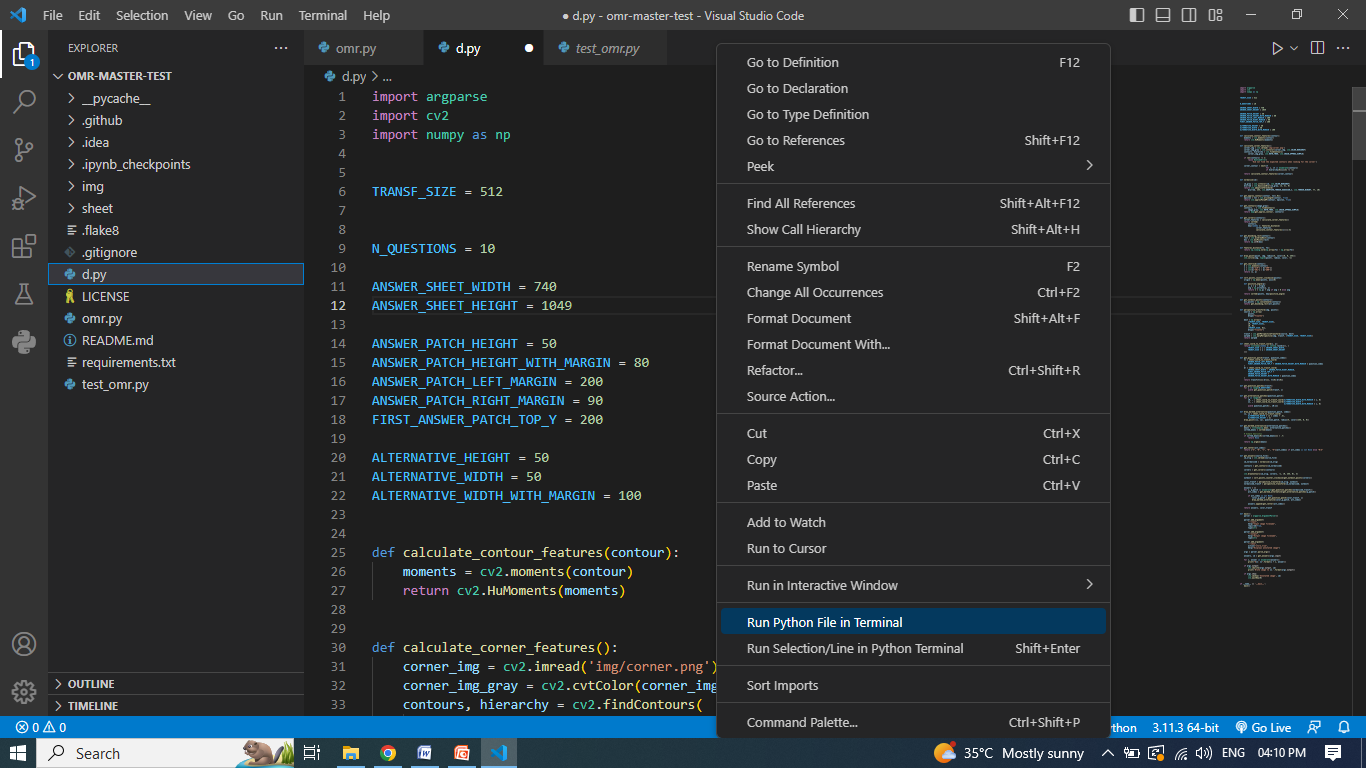
        cv2.waitKey(0)

if \_\_name\_\_ == '\_\_main\_\_':

    main()

**CHAPTER 10**

**10.SCREENSHOTS**



**CHAPTER 11**

**11.CONCLUSION**

OMR scanners are increasingly becoming popular in exams and other tests due to their ability to accurately and quickly scan large volumes of documents. However, there is still a need for a PROBLEM DEFINITION for Thus a high efficient semi-automated OMR scanning system that could help examiners evaluate the scanned documents more efficiently. With this, it is possible to create an automated system that can accurately and quickly scan large volumes of documents with minimal human intervention. This would further reduce the time taken by examiners in evaluating the scanned documents and ensure accuracy in results.

**CHAPTER 12**

**12.FUTURE ENHANCEMEN**

As technology advances, more and more businesses are turning to mobile apps to help them stay competitive. This system could be developed as an Android mobile app that uses the camera to take images and feed them into the system for image processing. This would enable businesses to quickly and efficiently process images, allowing them to make decisions quickly based on the data gathered from the images. Furthermore, this system could be used for various use cases such as automated facial recognition, object recognition, and motion detection. With its potential applications in various industries, this system could revolutionize how businesses operate in the future.

**CHAPTER 13**

**13.BIBLOGRAPHY**

* OMR Software - A list of popular OMR software options, including free and paid options, as well as user reviews and ratings.
* OMR Blog - A blog dedicated to discussing various aspects of OMR technology, including best practices, tips and tricks, and industry news.