# OMR SHEET EVALUATOR

*M Muddu Krishna Yadav*

# ABSTRACT

Optical mark recognition (OMR) is a technology that allows computers to read and interpret human-marked forms. OMR is used in a variety of applications, including surveys, exams, and ballots.

OMR evaluation is the process of determining whether a scanned OMR form has been correctly filled out. This is done by comparing the scanned image of the form to a template of the correctly filled out form.

There are a number of different methods that can be used for OMR evaluation. One common method is to use a template matching algorithm. This algorithm compares the scanned image of the form to the template and determines whether the two match.
Another method that can be used for OMR evaluation is to use a neural network. A neural network is a type of machine learning algorithm that can be trained to recognize patterns in data. In the case of OMR evaluation, a neural network can be trained to recognize the correct markings on an OMR form.
Here are some of the benefits of OMR evaluation:
It is a fast and efficient way to evaluate OMR forms.
It is accurate and reliable.
It can be used to evaluate a variety of different OMR forms.
Here are some of the challenges of OMR evaluation:
The quality of the scanned image can affect the accuracy of the evaluation.
The template must be accurate and up-to-date. The method that is used for evaluation must be appropriate for the specific application.

# DESCRIPTION

*OMR sheet scanner and test grader using OMR, Python, and OpenCV*

The goal of this blog post is to build a bubble sheet scanner and test grader using Python and OpenCV.

To accomplish this, our implementation will need to satisfy the following 7 steps.

OpenCV is the huge open-source library for the computer vision, machine learning, and image processing and now it plays a major role in real-time operation which is very important in today's systems. By using it, one can process images and videos to identify objects, faces, or even handwriting of a human. When it integrated with various libraries, such as NumPy, python is capable of processing the OpenCV array structure for analysis. To Identify image pattern and its various features we use vector space and perform mathematical operations on these features

Optical Mark Reader (OMR) is the technology of electronically extracting data from marked fields such as checkboxes or bubbles from pre-printed forms. OMR technology scans a printed form and reads from predefined positions. It records the data where marks are made on the form. This is widely used for processing the large number of hand-filled forms which have to be processed quickly and with great accuracy

The code first loads the answer key and student response images as grayscale images. Then, it performs image processing to extract the OMR bubbles. This is done by thresholding the images to convert them to binary images, and then finding contours in the images. The contours are sorted

from left to right, and then the selected answers are compared by comparing the centroid positions of the contours.

# ALGORITHM

1. Perform Image Pre-processing:
    a. Convert the image to grayscale.
    b. Apply Gaussian blurring to reduce noise and smoothen the image.
    c. Use erosion and dilation to remove noise near black marks. This can be achieved using morphological operations like cv2.erode() and cv2.dilate().

2. Crop the Image:
    a. Find the bounding boxes of the black marks using the contours (to be done in step 3).
    b. Crop the image based on the bounding boxes to extract the regions containing the black marks.

3. Find Contours and Centroids:
    a. Threshold the pre-processed image to create a binary image.
    b. Find contours in the binary image using cv2.findContours().
    c. Calculate the centroids of the contours using cv2.moments().

4. Find Angle of Rotation:
    a. Sort the contours based on their y-coordinate to identify the top and bottom black marks.
    b. Calculate the angle between the line connecting the centroids of the top and bottom marks and the horizontal axis.

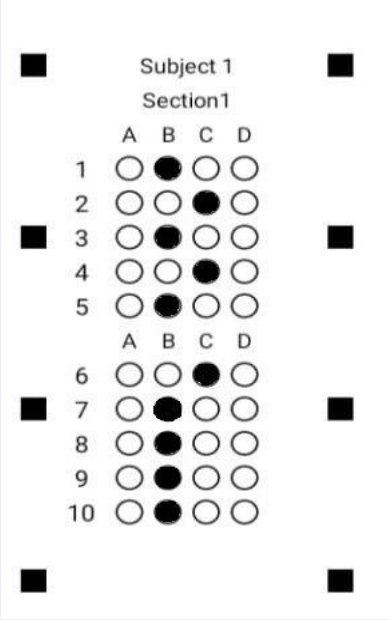
5. Rotate the Original Image:
    a. Rotate the original image by the calculated angle using cv2.warpAffine().

6. Check Key with Responses:
    a. Pre-process the rotated image for reading responses (if required).

7. Mark Correct, Incorrect, and Unanswered Responses:
    a. Compare the candidate's responses with the correct answers from the key.
    b. Mark the responses in the image with green for correct, red for wrong, and yellow for unanswered.

8. Count Correct, Incorrect, and Unanswered Responses:
    a. Loop through the responses and compare with the key.
    b. Keep a count of correct, incorrect, and unanswered responses.

INPUT:

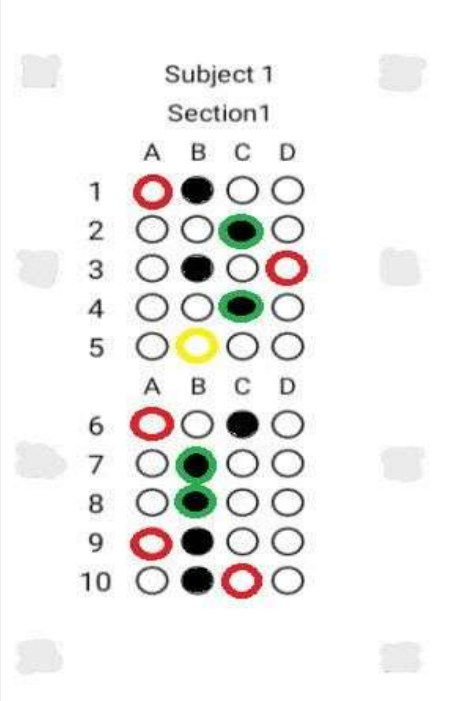


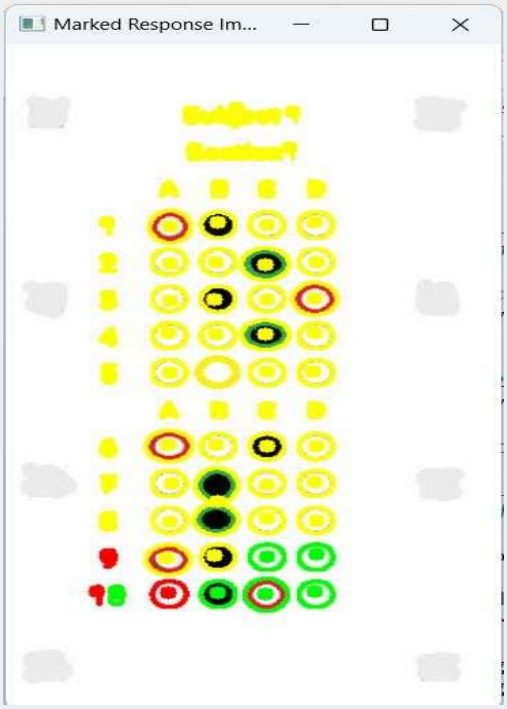
# *CODE FOR EVALUATION:*

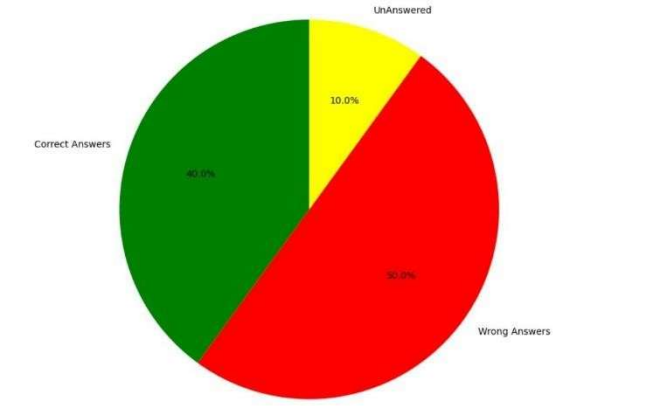
```
import cv2
import pytesseract

def preprocess_image(image_path):
    # Step 1: Image flre-processing
    img = cv2.imread("D:/omr/WhatsApp Image 2023-07-20 at 10.22.29.jpg",
cv2.IMREAD_GRAYSCALE)
    img = cv2.GaussianBlur(img, (5, 5), 0)
    kernel = cv2.getStructuringElement(cv2.MORflH_RECT, (5, 5))
    img = cv2.erode(img, kernel, iterations=1)
    img = cv2.dilate(img, kernel, iterations=1)
    return img

def crop_image(image, contours):
    # Step 2: Crop the Image
    cropped_images = []
    for contour in contours:
        x, y, w, h = cv2.boundingRect(contour)
        cropped_images.append(image[y:y + h, x:x + w])
    return cropped_images

def find_contours_and_centroids(image):
    # Step 3: Find Contours and Centroids
    _, binary_image = cv2.threshold(image, 0, 255, cv2.THRESH_BINARY_INV +
cv2.THRESH_OTSU)
    contours, _ = cv2.findContours(binary_image, cv2.RETR_EXTERNAL,
cv2.CHAIN_AflflROX_SIMflLE)
    centroids = []
    for contour in contours:
        M = cv2.moments(contour)
        cx = int(M['m10'] / M['m00'])
        cy = int(M['m01'] / M['m00'])
        centroids.append((cx, cy))
    return contours, centroids

def ocr_for_question_number(image):
    # Step 4: OCR to detect question number
    return pytesseract.image_to_string(image)

def compare_responses_with_key(responses, key):
    # Step 6: Check Key with Responses
    results = []
    for i, response in enumerate(responses):
        if response == key[i]:
            results.append("Correct")
        elif response == "":
            results.append("Unanswered")
```

```
            else:
                results.append("Incorrect")
        return results

def mark_responses_on_image(image, contours, results):
        # Step 7: Mark Correct, Incorrect, and Unanswered Responses
        for i, contour in enumerate(contours):
            color = (0, 255, 0)  # Green color for correct answer
            if results[i] == "Incorrect":
                color = (0, 0, 255)  # Red color for incorrect answer
            elif results[i] == "Unanswered":
                color = (0, 255, 255) # Yellow color for unanswered question
            cv2.drawContours(image, [contour], -1, color, 3)

def count_results(results):
        # Step 8: Count Correct, Incorrect, and Unanswered Responses
        correct_count = results.count("Correct")
        incorrect_count = results.count("Incorrect")
        unanswered_count = results.count("Unanswered")
        return correct_count, incorrect_count, unanswered_count

if _name_ == "_main_":
        image_path = "D:/omr/WhatsApp Image 2023-07-20 at 10.22.29.jpg"
        key = ["A", "B", "C", "D", "B", "A", "C", "D"]      # Replace this with the
correct key for your exam

        preprocessed_img = preprocess_image(image_path)
        contours, centroids = find_contours_and_centroids(preprocessed_img)
        cropped_images = crop_image(preprocessed_img, contours)

        responses = []
        for cropped_image in cropped_images:
            question_number = ocr_for_question_number(cropped_image)
            responses.append(question_number.strip())

        results = compare_responses_with_key(responses, key)

        image = cv2.imread(image_path)
        mark_responses_on_image(image, contours, results)

        correct_count, incorrect_count, unanswered_count =
count_results(results)
        print("Correct: ", correct_count)
        print("Incorrect: ", incorrect_count)
        print("Unanswered: ", unanswered_count)

        cv2.imshow("Marked Image", image)
        cv2.waitKey(0)
        cv2.destroyAllWindows()
```

*OUTPUT:*




# ERROR TYPE1:

### Installing Tesseract on Windows

Please note that the PyImageSearch team and I *do not* officially support Windows, except for customers who use our pre-configured Jupyter/Colab Notebooks, which you can find at **PyImageSearch University**. These notebooks run on all environments, including macOS, Linux, and Windows.

**We instead recommend using a Unix-based machine such as Linux/Ubuntu or macOS**, both of which are better suited for developing computer vision, deep learning, and OCR projects.

That said, if you wish to install Tesseract on Windows, we recommend that you follow the official Windows install instructions put together by the **Tesseract team**.

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_20644/3358423611.py in <module>
      1 import cv2
      2 import numpy as np
----> 3 import pytesseract  # OCR library, you may need to install it using: pip install pytesseract
      4 from PIL import Image  # Python Imaging Library, you may need to install it using: pip install pillow
      5

ModuleNotFoundError: No module named 'pytesseract'
```

ERROR TYPE 2:

```
---------------------------------------------------------------------------
error                                     Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_21796/3137371848.py in <module>
     62 # Perform cropping based on your requirement
     63
---> 64 centroid1, centroid2, rotation_angle = find_centroids_and_rotation_angle(cropped_image)
     65
     66 rotated_image = rotate_image(image, rotation_angle)

~\AppData\Local\Temp/ipykernel_21796/3137371848.py in find_centroids_and_rotation_angle(cropped_image)
     21 def find_centroids_and_rotation_angle(cropped_image):
     22     # Find contours in the cropped image
---> 23     contours, _ = cv2.findContours(cropped_image, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
     24
     25     # Get the centroids of the two black marks

error: OpenCV(4.8.0) D:\a\opencv-python\opencv-python\opencv\modules\imgproc\src\contours.cpp:197: error: (-210:Unsupported for
mat or combination of formats) [Start]FindContours supports only CV_8UC1 images when mode != CV_RETR_FLOODFILL otherwise suppor
ts CV_32SC1 images only in function 'cvStartFindContours_Impl'
```

If we didn't use the "pytessract" module we may have this kind of error and shows the output of all marked with random yellow and green and red colors and we cant use this block without that package for detecting .

# *Output:*

```
Correct Responses: 4
Incorrect Responses: 5
Unanswered Responses: 1
total score: 40%
```

# OMR SHEET EVALUATION CODE:

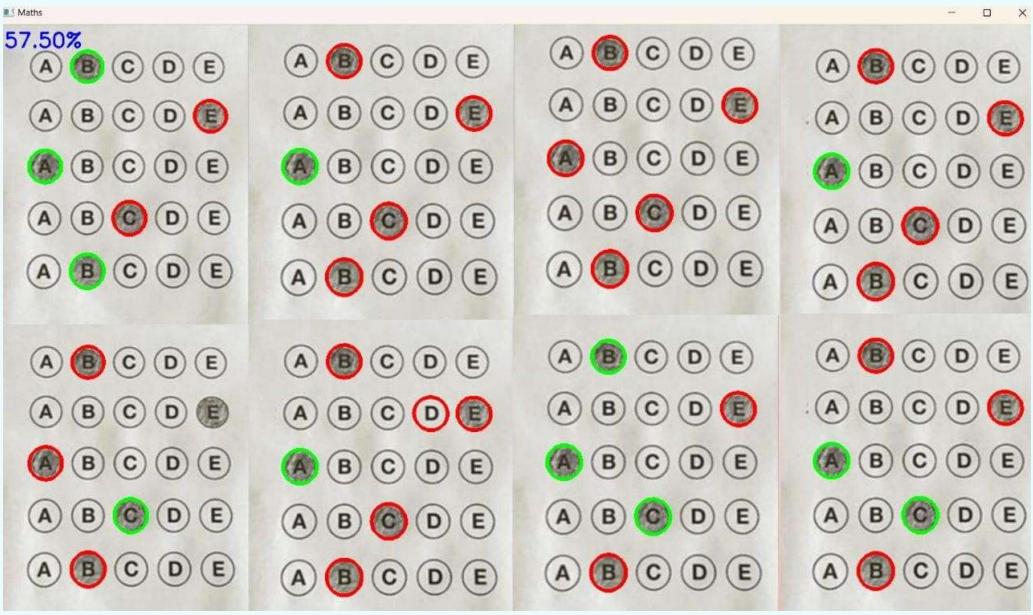
```
import numpy as np
import numpy as np
import tkinter as tk
imoort cv2
from tkinter import filedialog
import plotly.graph_objects as go

def main():
    name = input("fllease Enter student's name: ")

    ANSWER_KEY_MATHS = {
       0: 0, 1: 0, 2: 2, 3: 1, 4: 1, 5: 1, 6: 0, 7: 0, 8: 3, 9: 0, 10: 0,
11: 0, 12: 1, 13: 1, 14: 0, 15: 1, 16: 3, 17: 1, 18: 3, 19: 2, 20: 0, 21:
1, 22: 3, 23: 2, 24: 2, 25: 1, 26: 1, 27: 2, 28: 0, 29: 0, 30: 3, 31: 0,
32: 2, 33: 2, 34: 2, 35: 0, 36: 3, 37: 0, 38: 3, 39: 0, 40: 2, 41: 2, 42:
1, 43: 0, 44: 0, 45: 1, 46: 2, 47: 0, 48: 1, 49: 0, 50: 3, 51: 2, 52: 3,
53: 2, 54: 1, 55: 2, 56: 2, 57: 1, 58: 2, 59: 0, 60: 1, 61: 1, 62: 1, 63:
3, 64: 3, 65: 2, 66: 1, 67: 2, 68: 0, 69: 1, 70: 0, 71: 2, 72: 3, 73: 2,
74: 0, 75: 1, 76: 2, 77: 1, 78: 3, 79: 3, 80: 3, 81: 1, 82: 2, 83: 1, 84:
1, 85: 2, 86: 3, 87: 3, 88: 2, 89: 1, 90: 1, 91: 3, 92: 0, 93: 0, 94: 0,
95: 1, 96: 1, 97: 3, 98: 0, 99: 3, 100: 3, 101: 3, 102: 2, 103: 0, 104: 0,
105: 2, 106: 2, 107: 0, 108: 2, 109: 3, 110: 1, 111: 3, 112: 0, 113: 2,
114: 1, 115: 0, 116: 2, 117: 1, 118: 1, 119: 2, 120: 1, 121: 0, 122: 2,
123: 3, 124: 0, 125: 0, 126: 2, 127: 2, 128: 1, 129: 0, 130: 1, 131: 0,
132: 0, 133: 3, 134: 0, 135: 1, 136: 1, 137: 3, 138: 1, 139: 2, 140: 3,
141: 1, 142: 1, 143: 2, 144: 3, 145: 2, 146: 3, 147: 3, 148: 0, 149: 1,
150: 1, 151: 0, 152: 2, 153: 0, 154: 1, 155: 1, 156: 0, 157: 0, 158: 0,
159: 1, 160: 0, 161: 0, 162: 2, 163: 0, 164: 1, 165: 2, 166: 3, 167: 1,
168: 1, 169: 3, 170: 1, 171: 3, 172: 3, 173: 1, 174: 0, 175: 0, 176: 3,
177: 2, 178: 3, 179: 3, 180: 3, 181: 0, 182: 0, 183: 0, 184: 3, 185: 2,
186: 0, 187: 1, 188: 1, 189: 1, 190: 3, 191: 1, 192: 3, 193: 1, 194: 2,
195: 3, 196: 1, 197: 0, 198: 3, 199: 0   }

    print("Choose OMR")
    root = tk.Tk()
    root.withdraw()
    img_path = filedialog.askopenfilename(filetypes=(("ImageFiles",
("*.jpg", "*.png", "*.jpeg")), ("All Files", "*")), initialdir="D:/omr",
title="fllease select an OMR scanned image")

    if not img_path:
        return

    image = cv2.imread(img_path)
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    blurred = cv2.GaussianBlur(gray, (5, 5), 0)
    edged = cv2.Canny(blurred, 75, 200)
```

```
    cnts = cv2.findContours(edged.copy(), cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
    cnts = cnts[0] if len(cnts) == 2 else cnts[1]
    docCnt = None

    correct_m = 0
    if len(cnts) > 0:
        cnts = sorted(cnts, key=cv2.contourArea, reverse=True)

        for c in cnts:
            peri = cv2.arcLength(c, True)
            approx = cv2.approxPolyDP(c, 0.02 * peri, True)

            if len(approx) == 4:
                docCnt = approx
                break

    thresh = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY_INV |
cv2.THRESH_OTSU)[1]

    cnts = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
    cnts = cnts[0] if len(cnts) == 2 else cnts[1]
    questionCnts = []

    for c in cnts:
        (x, y, w, h) = cv2.boundingRect(c)
        ar = w / float(h)

        if w >= 20 and h >= 20 and ar >= 0.9 and ar <= 1.1:
            questionCnts.append(c)

    questionCnts = sorted(questionCnts, key=lambda x:
cv2.boundingRect(x)[1])

    for q in range(0, len(questionCnts), 5):
        cnts = sorted(questionCnts[q:q + 5], key=lambda x:
cv2.boundingRect(x)[0])
        bubbled = None

        for (j, c) in enumerate(cnts):
            mask = np.zeros(thresh.shape, dtype="uint8")
            cv2.drawContours(mask, [c], -1, 255, -1)

            mask = cv2.bitwise_and(thresh, thresh, mask=mask)
            total = cv2.countNonZero(mask)

            if bubbled is None or total > bubbled[0]:
                bubbled = (total, j)
```

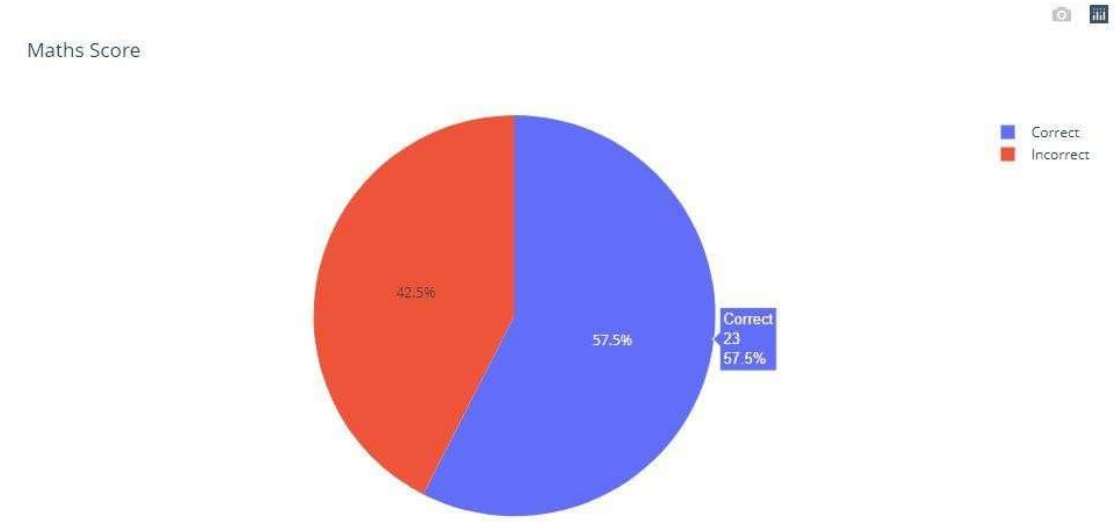
```
        color = (0, 0, 255)

        k = q // 5    # Convert 'q' to the corresponding index in
ANSWER_KEY_MATHS
        if k in ANSWER_KEY_MATHS and ANSWER_KEY_MATHS[k] == bubbled[1]:
            color = (0, 255, 0)
            correct_m += 1

        cv2.drawContours(image, [cnts[bubbled[1]]], -1, color, 3)

    print("\n\nMATHS:-\n")
    score_m = (correct_m / 40.0) * 100# Assuming there are 40 questions
in the 200-bit test
    print("Correct Answers = {}".format(correct_m))
    print("Incorrect Answers = {}".format(40 - correct_m))
    print("Score: {:.2f}%".format(score_m))
    cv2.putText(image, "{:.2f}%".format(score_m), (10, 30),
cv2.FONT_HERSHEY_SIMPLEX, 0.9, (255, 0, 0), 2)
    cv2.imshow("Maths", image)
    cv2.waitKey(0)

    tot = correct_m
    avg = (tot / 200) * 100

    print("\nTotal correct answers = {} ".format(tot))
    print("Total percentage = {} ".format(avg))

    labels = ['Correct', 'Incorrect']
    values = [correct_m, 40 - correct_m]

    fig = go.Figure(data=[go.pie(labels=labels, values=values)])
    fig.update_layout(title='Maths Score')
    fig.show()

    with open("D:/omr", "a+") as f:
        temp = str(name) + "\t" + str(correct_m)
        f.write(temp)

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

## *INPUT:*

## *OUTPUT:*



```
Please Enter student's name: gggggg
Choose OMR


MATHS:-

Correct Answers = 23
Incorrect Answers = 17
Score: 57.50%

Total correct answers = 23
Total percentage = 11.5
```




# THE END