Batch: B1 Roll No.: 1711083

Experiment No. 9

Title: Implementation of CAPTCHA for Security of systems

Objective: To Implement of randomised and innovative CAPTCHA for Security of systems.

Expected Outcome of Experiment:

CO	Outcome
CO1	Understand the concept and need of captcha for security systems.
CO2	Implement randomise captcha.
CO3	Implement innovative captcha.

Books/ Journals/ Websites referred:

1. https://piratefsh.github.io/projects/2015/04/30/battle-of-the-captchas.html
2. https://www.edureka.co/blog/tkinter-tutorial/
3. https://www.youtube.com/watch?v=bfKwizfuuOU
4. https://www.pandasecurity.com/mediacenter/panda-security/what-is-captcha/
5. https://www.imperva.com/learn/application-security/what-is-captcha/
6. https://www.letsnurture.com/blog/8-widely-used-captcha-examples.html

Abstract:-

In this experiment we are gonna implement two things one is randomised captcha and innovative captcha. Randomised captcha is very simple; it's done just using a simple if - else statement. My idea for innovation is using math + color + shape basically asking the user to enter the shape and color in which the expression is right.

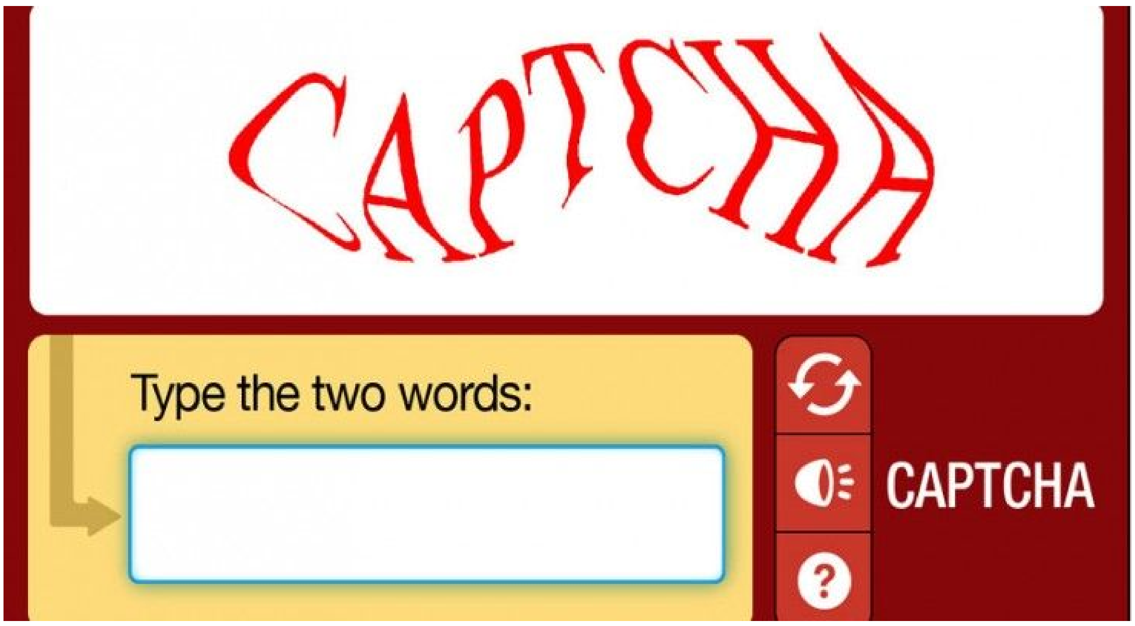
Related Theory: -

What is captcha?

CAPTCHA stands for Completely Automated Public Turing test to tell Computers and Humans Apart. In other words, CAPTCHA determines whether the user is real or a spam robot. CAPTCHAs stretch or manipulate letters and numbers, and rely on human ability to determine which symbols they are.

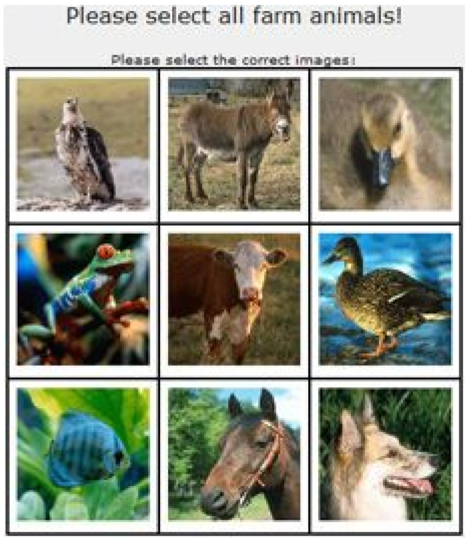
Examples of commonly used captcha :

1. **The standard word captcha with an audio option.**



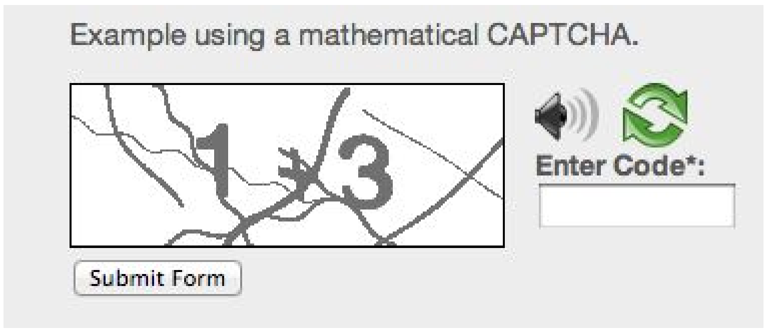
This is the standard captcha available whenever a security check-in is required, where you need to write the word which has been displayed. But some of the distorted word images are hard to solve. To get this pass through it allows you to use the option of **"Recaptcha"**, in order to receive a new one. There is also an audio option if you are unable to visually make out the word. These are the most commonly used while preparing a form in website development or app development.

2. Picture Identification of Captcha


Please select all farm animals!
Please select the correct images:

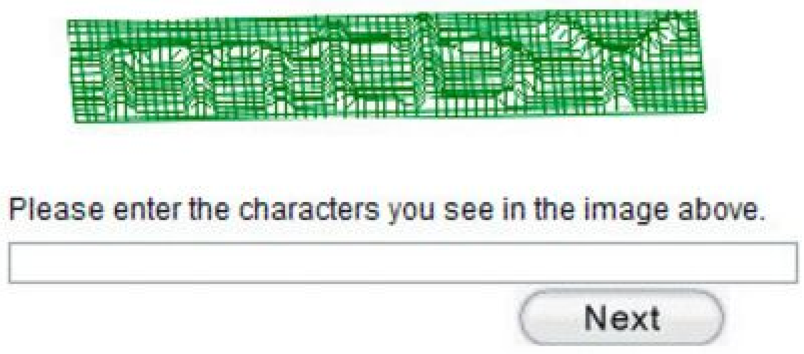
This captcha provides users for selecting the elementary choice of selecting the correct image that they are asked to identify. This type of captcha usually never gets harder than the basic images, so you do not have to worry about your users not being able to depict them.

3. Math Solution



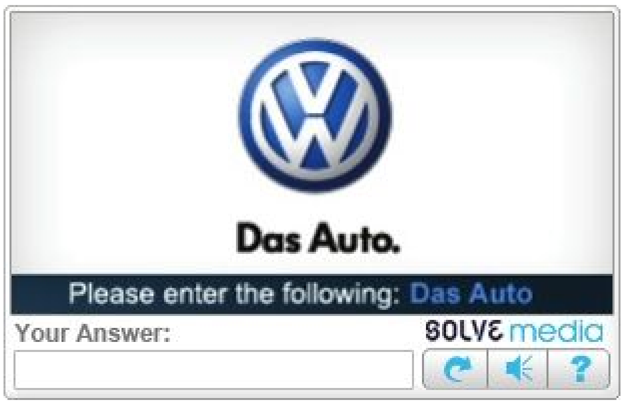
This type of captcha involves basic math problems and if your user cannot solve these basic questions then probably you do not want them to visit your website further. This provides easy to read numbers and must be solved to get through the captcha.

4. **3D Captcha**



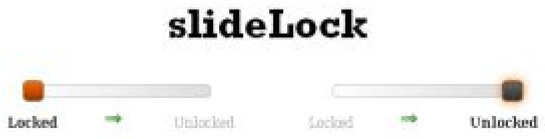
These types of captchas are called "Super Captcha" because there are several 3D images which include both images and words and thus it becomes hard for one to solve it.

5. **Ad - injected Captcha**



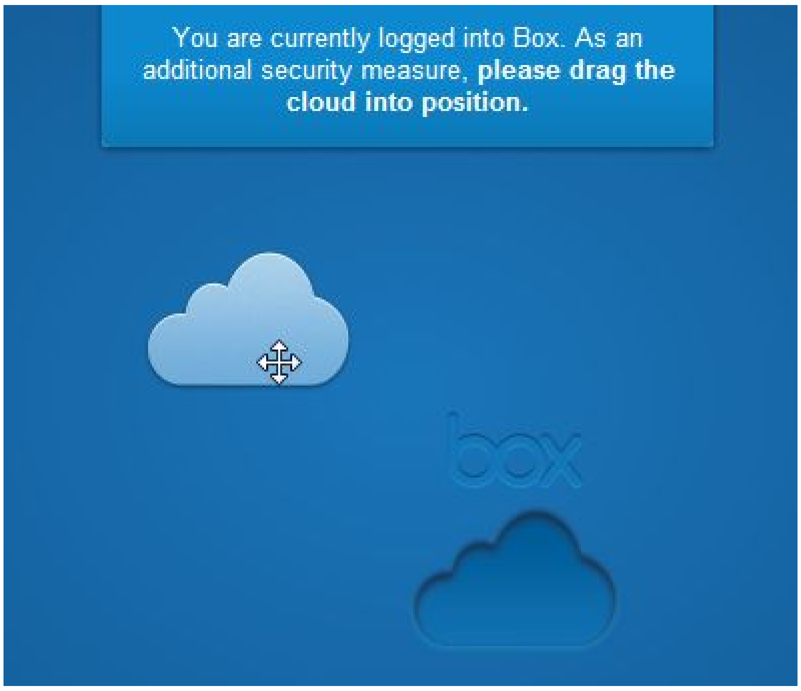
This type of captcha helps your websites to earn some extra cash by publishing it, which in turn also helps in terms of brand recognition.

6. **jQuery Slider Captcha**



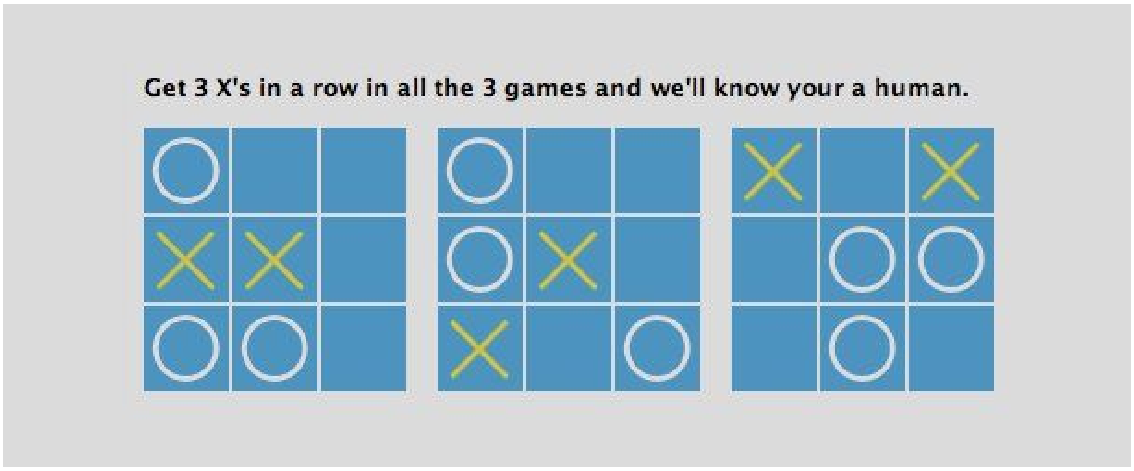
This is a plugin which gives you the ability to add captcha to your forms which are easy to use. This plugin is very useful to keep the spammers away. This plugin lock is disabled until a person slides it to enable it.

7. Drag and drop Captcha


You are currently logged into Box. As an additional security measure, **please drag the cloud into position.**

This is also one of the easy to use captchas. It is jQuery based which allows the user to drag the required object or shape to pass through the security gate.

8. Tic Tac Toe Captcha


Get 3 X's in a row in all the 3 games and we'll know your a human.

This captcha which involves gamification was designed for fun and an easy way to ensure that only humans can interact with your website. The captcha that does not hurt that much.

How Does a CAPTCHA Work?

CAPTCHAs were invented to block spam software from posting comments on pages or purchasing excess items at once. The most common form of CAPTCHA is an image with several distorted letters. It is also common to choose from a variety of images where you need to select a common theme.

The internet and computers are actually made up of a unique coding language. Computers find it difficult to understand languages because of the strange and intricate rules human languages take on, along with slang that humans use.

How CAPTCHA Prevents Scammers

CAPTCHA has a variety of applications for keeping websites and users secure. These include but are not limited to:

- Protecting email addresses from scammers
- Protect website registrations
- Protects online polling

- Protects against email worms/junk mail
- Prevents dictionary attacks
- Prevents comment spamming on blogs

What are CAPTCHAs Used for?

CAPTCHAs are used by any website that wishes to restrict usage by bots. Specific uses include:

- **Maintaining poll accuracy** :- CAPTCHAs can prevent poll skewing by ensuring that each vote is entered by a human. Although this does not limit the overall number of votes that can be made, it makes the time required for each vote longer, discouraging multiple votes.
- **Limiting registration for services** :- Services can use CAPTCHAs to prevent bots from spamming registration systems to create fake accounts. Restricting account creation prevents waste of a service's resources and reduces opportunities for fraud.
- **Preventing ticket inflation** :- Ticketing systems can use CAPTCHA to limit scalpers from purchasing large numbers of tickets for resale. It can also be used to prevent false registrations to free events.
- **Preventing false comments** :- CAPTCHAs can prevent bots from spamming message boards, contact forms, or review sites.

The extra step required by a CAPTCHA can also play a role in reducing online harassment through inconvenience.

Drawbacks of Using CAPTCHA

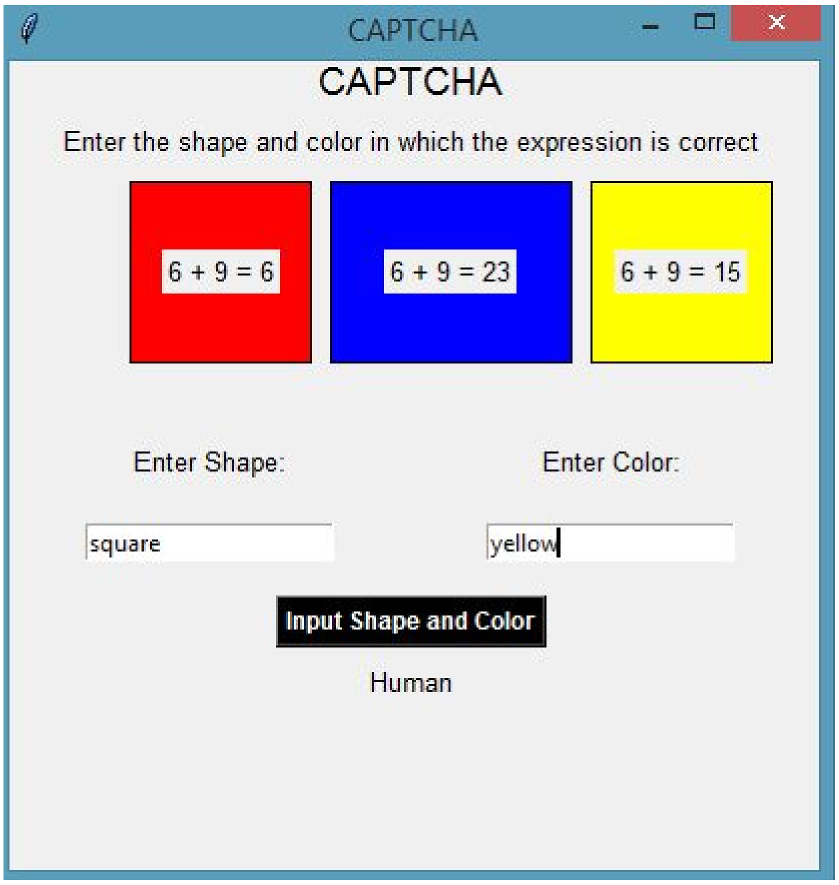
The overwhelming benefit of CAPTCHA is that it is highly effective against all but the most sophisticated bad bots. However, CAPTCHA mechanisms can negatively affect the user experience on your website:

- Disruptive and frustrating for users.
- May be difficult to understand or use for some audiences
- Some CAPTCHA types do not support all browsers.
- Some CAPTCHA types are not accessible to users who view a website using screen readers or assistive devices.

Implementation Details:

1. Enlist all the Steps followed and various options explored

While implementing randomised captcha I have gone through the video and have simply implemented it. For innovative ideas I have gone through many websites searching for different examples of captcha and I have combined 2 ideas together adding one idea of my own. The first idea was **Math Captcha** and the second idea was **Color Selection** and my idea was to add **Shape selection.**


CAPTCHA

CAPTCHA

Enter the shape and color in which the expression is correct

6 + 9 = 6

6 + 9 = 23

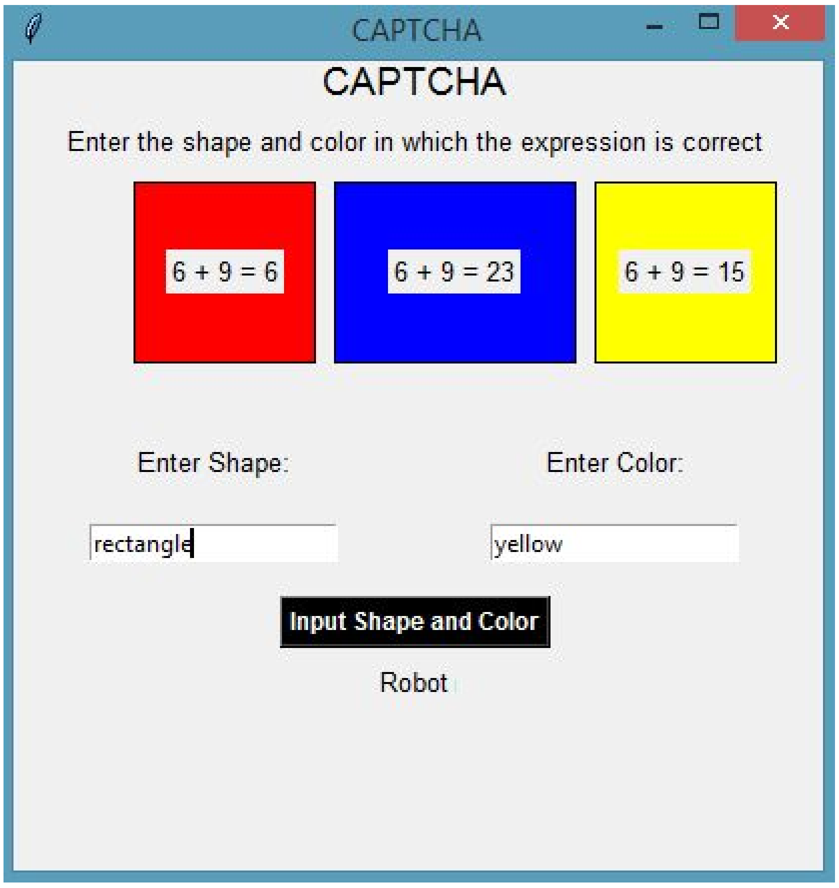
6 + 9 = 15

Enter Shape: square

Enter Color: yellow

Input Shape and Color

Human



CAPTCHA

CAPTCHA

Enter the shape and color in which the expression is correct

6 + 9 = 6

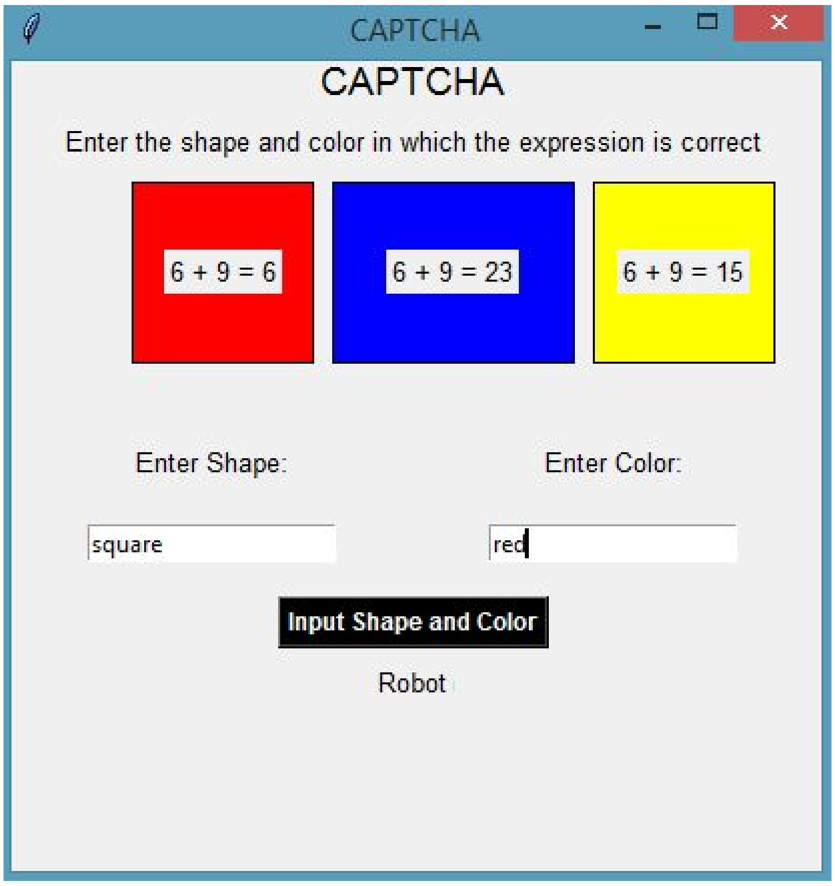
6 + 9 = 23

6 + 9 = 15

Enter Shape: rectangle

Enter Color: yellow

Robot



2. Explain your program logic, classes and methods used.

I have used tkinter for my GUI. GUI contains 9 labels, 3 shapes, 2 text boxes and a button for taking the input. The following code contains the important logic. I'm not explaining the GUI part.

I have used two functions:

1. Used for generating a list of random unique numbers.

```
def createRandomUniqueList(num, start, end):
    arr = []
    tmp = r.randint(start, end)
    for x in range(num):
        while tmp in arr:
            tmp = r.randint(start, end)
        arr.append(tmp)
    return arr
```

Input : num -> number of total elements in the list, start -> start number, end -> end number

Output : returns the list of random unique numbers.

Here I am simply using two loops inner is while and outer is for and a temp variable which contains the random number between the range (start to end) and in the inner loop just checking whether it has been already there in the arr list if there then again generating the random number and storing it in temp variable and doing it till it’s unique. The outer loop will iterate till the end is reached.

2. This function gets triggered when someone clicks on the button.

```
def checkShapeColor():

    shape_input = shape_text_box.get()
    color_input = color_text_box.get()

    right_ans = index_list.index(0)
    # print(right_ans)
             x1,    y1,    x2,    y2    =    coordinates[right_ans][0],
coordinates[right_ans][1],                 coordinates[right_ans][2],
coordinates[right_ans][3]
    c = color[color_list[right_ans]]
    # print(x1, y1, x2, y2, c)
    if x1 - x2 == y1 - y2:
        s = 'square'
    else:
        s = 'rectangle'
    if shape_input == s and color_input == c :

        human = tk.Label(root, text = 'Enter into the website',font
= ('helvetica', 10))
        canvas1.create_window(200, 310, window = human)

    else:

            robot = tk.Label(root, text = "Can't enter into the
website",font = ('helvetica', 10))
        canvas1.create_window(200, 310, window = robot)
```

Here I am simply backtracking to the expression which is right and then retrieving the color and shape from the list stored and comparing it with the imputed answer and accordingly adding the label.

Logic for randomizing the expression and color :

```
coordinates = [[60, 60, 150, 150], [160, 60, 280, 150], [290, 60,
380, 150]]
color = ['red', 'green', 'blue', 'yellow', 'orange']
color_list = createRandomUniqueList(5, 0, 4)
index_list = createRandomUniqueList(3, 0, 2)

canvas1.create_rectangle(coordinates[index_list[0]][0],
coordinates[index_list[0]][1],       coordinates[index_list[0]][2],
coordinates[index_list[0]][3],                fill              =
color[color_list[index_list[0]]])
canvas1.create_rectangle(coordinates[index_list[1]][0],
coordinates[index_list[1]][1],       coordinates[index_list[1]][2],
coordinates[index_list[1]][3],                fill              =
color[color_list[index_list[1]]])
canvas1.create_rectangle(coordinates[index_list[2]][0],
coordinates[index_list[2]][1],       coordinates[index_list[2]][2],
coordinates[index_list[2]][3],                fill            =
color[color_list[index_list[2]]])
```

In the above code snippet I'm initializing three list co - ordinates and color. Then using the `createRandomUniqueList(num, start, end):` function I am randomly placing the colors.

```
operators = ["+","-","*"]
num1,  num2,  operation  =  r.randint(1,  10),  r.randint(1,  10),
r.choice(operators)
val = eval(str(num1) + operation + str(num2))
exp  =  [str(num1)  +  ' '  + operation +  ' '   + str(num2)  +  ' = '  +
str(val), str(num1) + ' ' + operation + ' '  + str(num2) + ' = ' +
str(val + r.randint(1, 10)), str(num1) + ' ' + operation + ' '  +
str(num2) + ' = ' + str(val - r.randint(1, 10))]
```

In the above code snippet I'm initializing a list operator and then randomly selecting it using the `r.randint(1, 10)` function. Storing it in the num1, num2 and operation. The val contains the answer of the expression and it's done by the `eval(str(num1) + operation + str(num2))` function. Then simply storing it in the exp var as string and then adding in the GUI as the label which you can see in the code below.

Code:

```
import tkinter as tk
import random as r

root = tk.Tk()

root.title("CAPTCHA")

canvas1  =  tk.Canvas(root,  width  =  400,  height  =  400,   relief  =
'raised')
canvas1.pack()
```

```
heading = tk.Label(root, text = 'CAPTCHA')
heading.config(font = ('helvetica', 14))
canvas1.create_window(200, 10, window = heading)

instructions = tk.Label(root, text = 'Enter the shape and color in
which the expression is correct')
instructions.config(font = ('helvetica', 10))
canvas1.create_window(200, 40, window = instructions)

def createRandomUniqueList(num, start, end):
    arr = []
    tmp = r.randint(start, end)
    for x in range(num):
        while tmp in arr:
            tmp = r.randint(start, end)
        arr.append(tmp)
    return arr

coordinates = [[60, 60, 150, 150], [160, 60, 280, 150], [290, 60,
380, 150]]
color = ['red', 'green', 'blue', 'yellow', 'orange']
color_list = createRandomUniqueList(5, 0, 4)
index_list = createRandomUniqueList(3, 0, 2)

canvas1.create_rectangle(coordinates[index_list[0]][0],
coordinates[index_list[0]][1],       coordinates[index_list[0]][2],
coordinates[index_list[0]][3],                fill                =
color[color_list[index_list[0]]])
canvas1.create_rectangle(coordinates[index_list[1]][0],
coordinates[index_list[1]][1],       coordinates[index_list[1]][2],
coordinates[index_list[1]][3],                fill                =
color[color_list[index_list[1]]])
canvas1.create_rectangle(coordinates[index_list[2]][0],
coordinates[index_list[2]][1],       coordinates[index_list[2]][2],
coordinates[index_list[2]][3],                fill                =
color[color_list[index_list[2]]])

operators = ["+","-","*"]
```

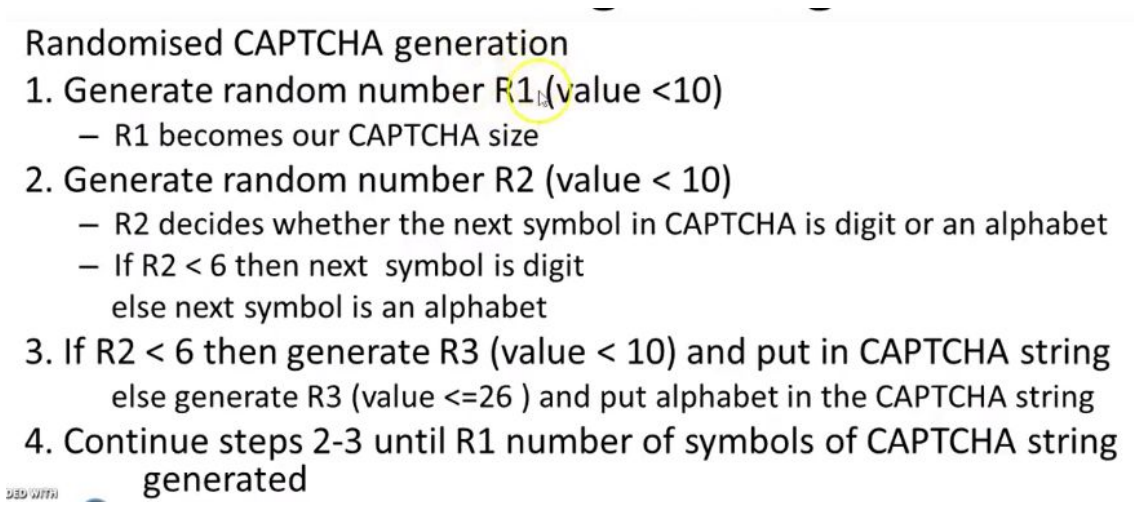
```
num1,  num2,  operation  =  r.randint(1,  10),  r.randint(1,  10),
r.choice(operators)
val = eval(str(num1) + operation + str(num2))
exp = [str(num1) + ' ' + operation + ' '  + str(num2) + ' = ' +
str(val), str(num1) + ' ' + operation + ' '  + str(num2) + ' = ' +
str(val + r.randint(1, 10)), str(num1) + ' ' + operation + ' '  +
str(num2) + ' = ' + str(val - r.randint(1, 10))]

# print(index_list)
# print(index_list.index(0))
# print(coordinates[index_list.index(0)][0])
# print(color)
# print(color_list)
# print(color[index_list.index(0)])
#        print(color[index_list[0]],        color[index_list[1]],
color[index_list[2]])

exp1 = tk.Label(root, text = exp[index_list[0]])
exp1.config(font = ('helvetica', 10))
canvas1.create_window(105, 105, window = exp1)
exp2 = tk.Label(root, text = exp[index_list[1]])
exp2.config(font = ('helvetica', 10))
canvas1.create_window(220, 105, window = exp2)
exp3 = tk.Label(root, text = exp[index_list[2]])
exp3.config(font = ('helvetica', 10))
canvas1.create_window(335, 105, window = exp3)

shape_text = tk.Label(root, text = 'Enter Shape:')
shape_text.config(font = ('helvetica', 10))
canvas1.create_window(100, 200, window = shape_text)

shape_text_box = tk.Entry (root)
canvas1.create_window(100, 240, window = shape_text_box)

color_text = tk.Label(root, text = 'Enter Color:')
color_text.config(font = ('helvetica', 10))
canvas1.create_window(300, 200, window = color_text)

color_text_box = tk.Entry (root)
```

```
canvas1.create_window(300, 240, window = color_text_box)

def checkShapeColor():

    shape_input = shape_text_box.get()
    color_input = color_text_box.get()

    right_ans = index_list.index(0)
    # print(right_ans)
             x1,   y1,   x2,   y2   =   coordinates[right_ans][0],
coordinates[right_ans][1],                  coordinates[right_ans][2],
coordinates[right_ans][3]
    c = color[color_list[right_ans]]
    # print(x1, y1, x2, y2, c)
    if x1 - x2 == y1 - y2:
        s = 'square'
    else:
        s = 'rectangle'

    if shape_input == s and color_input == c :

         human = tk.Label(root, text = 'Human',font = ('helvetica',
10))
        canvas1.create_window(200, 310, window = human)

    else:

         robot = tk.Label(root, text = "Robot",font = ('helvetica',
10))
        canvas1.create_window(200, 310, window = robot)

shape_color_input  =  tk.Button(text  =  'Input  Shape  and  Color',
command  =  checkShapeColor,  bg  =  'black',  fg  =  'white',  font  =
('helvetica', 9, 'bold'))
canvas1.create_window(200, 280, window = shape_color_input)

root.mainloop()
```

Code for Randomised Captcha:

```python
import random
def captcha():
  captcha = ""
  r1 = random.randint(0, 10)
  if r1 < 10:
    len = r1
  for i in range(0, len):
    r2 = random.randint(0, 10)
    # r2 = 7
    if r2 < 6:
      r3 = random.randint(0, 10)
      # if r3 < 10:
      captcha += str(r3)
    else:
      r3 = random.randint(0, 25)
      captcha += chr(r3 + 97)
  return captcha
print(captcha())
```

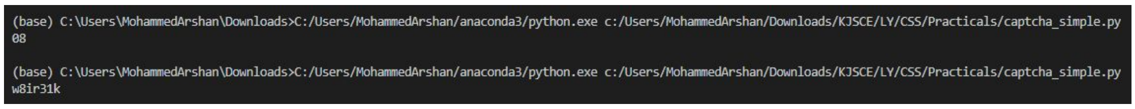
Logic:



Randomised CAPTCHA generation

1. Generate random number R1 (value <10)
   - R1 becomes our CAPTCHA size
2. Generate random number R2 (value < 10)
   - R2 decides whether the next symbol in CAPTCHA is digit or an alphabet
   - If R2 < 6 then next symbol is digit
     else next symbol is an alphabet
3. If R2 < 6 then generate R3 (value < 10) and put in CAPTCHA string
   else generate R3 (value <=26) and put alphabet in the CAPTCHA string
4. Continue steps 2-3 until R1 number of symbols of CAPTCHA string
   generated

Output:



```
(base) C:\Users\MohammedArshan\Downloads>C:/Users/MohammedArshan/anaconda3/python.exe c:/Users/MohammedArshan/Downloads/KJSCE/LY/CSS/Practicals/captcha_simple.py
08

(base) C:\Users\MohammedArshan\Downloads>C:/Users/MohammedArshan/anaconda3/python.exe c:/Users/MohammedArshan/Downloads/KJSCE/LY/CSS/Practicals/captcha_simple.py
w8ir31k
```

3. Explain the Importance of the approach followed by you

Since I have used randomized mathematical expressions, shape and color, so it becomes difficult for the bot to do three things first it's difficult to evaluate which expression is correct, second and third is to get the shape and color in which the expression is right. It doesn't pass the color-blindness test, but I figured adding the keyword for position (third) would compensate for that. It's simple, and not totally foolproof (you can always get bots to detect the hex color and do some simple math to determine if it's reddish), but good enough for a super-duper lightweight solution and since I have also added shape and math and I am also randomizing them so it will be difficult to get the right answer.

Conclusion:- Successfully implement the randomised and innovative CAPTCHA for Security of systems.