# Create a Screen recorder using Python

#### **About Screen recorder:**

To record our screen for presentation, we cannot rely on cameras for it. This is attributed to the quality and shakiness observed whilst recording. So what is the best alternative? A Screen Recorder. A screen recorder records the contents or anything visible on the screen and saves it for later use. Now the advantages are plenty; quality of the recording, less hassle and you can record it again at ease

#### **Python Screen recorder:**

Here, we create a simple screen recorder using tkinter and opencv.

#### **Project Prerequisites:**

The project is an introduction to Pillow and OpenCV. Hence understanding the syntactic meaning should suffice. The modules necessary are opencv, tkinter and pillow. Tkinter is already available in Python. Import to see, if it is available.

import tkinter

If it does not show any error, the package is available. If not, you can install it using the command

sudo apt-get install python3-tkinter, for linux users and windows users can reinstall python or follow the steps given in [Tkinter installation on windows](https://tkdocs.com/tutorial/install.html#install-win-python). The other packages can be installed using pip as follows

* opencv: pip install opencv-python
* pillow: pip install pillow

#### **Download Screen recorder code:**

You can download the source code for the Screen recorder in the given link: <add link>

#### **Project File Structure:**

Below is the flow of the project.

1. Importing libraries
2. Creating the screen recording function
3. Defining the GUI
4. Create the components and the button

##### **Importing libraries:**

#PythonGeeks Guide to create a simple screen recorder

#Import necessary modules

from PIL import ImageGrab

import cv2

import numpy as np

from tkinter import \*

**Code Explanation:**

* **from tkinter import \*:** We use Tkinter to build the user interface of the application. Hence we import it to make use of the widgets and define our application.
* **from PIL import ImageGrab:** Take screenshot of the screen using ImageGrab
* **import cv2:** To write the captured screen to a video, we use opencv
* **import numpy as np:** Convert images to arrays and vice versa

##### **Creating the screen recording function:**

def record\_screen():

#Obtain image dimensions

#Screen capture

image = ImageGrab.grab()

#Convert the object to numpy array

img\_np\_arr = np.array(image)

#Extract and print shape of array

shape = img\_np\_arr.shape

print(shape)

#Create a video writer

screen\_cap\_writer = cv2.VideoWriter('screen\_recorded.avi',cv2.VideoWriter\_fourcc('M','J','P','G'), 50, (shape[1], shape[0]))

#To View the screen recording in a separate window (OPTIONAL)

#This is optional. Use the aspect ratio scaling if you wish to view the screen recording simultaneously

#Low scale\_by\_percent implies smaller window

scale\_by\_percent = 50

width = int(shape[1] \* scale\_by\_percent / 100)

height = int(shape[0] \* scale\_by\_percent / 100)

new\_dim = (width, height)

#Record the screen

#Condition to keep recording as a video

while True:

#Capture screen

image = ImageGrab.grab()

#Convert to array

img\_np\_arr = np.array(image)

#OpenCV follows BGR and not RGB, hence we convert

final\_img = cv2.cvtColor(img\_np\_arr, cv2.COLOR\_RGB2BGR)

#Write to video

screen\_cap\_writer.write(final\_img)

#OPTIONAL: To view your screen recording in a separate window, resize and use imshow()

'''

If you choose to view the screen recording simultaneously,

It will be displayed and also recorded in your video.

'''

image = cv2.resize(final\_img, (new\_dim))

cv2.imshow("image", image)

#Stop and exit screen recording if user presses 'e' (You can put any letter)

if cv2.waitKey(1) == ord('e'):

break

#Release the created the objects

screen\_cap\_writer.release()

cv2.destroyAllWindows()

**Code explanation:**

* **def record\_screen():** Declare the function to initiate screen recording
* **image = ImageGrab.grab():** Takes a screenshot of the entire screen. If you wish to record only a certain part of the screen or an app, use the parameter bbox(). It takes the parameters left, top, right and bottom (clockwise from left to remember).
* **img\_np\_arr = np.array(image)**: Convert the image to numpy arrays. The array will contain pixel values corresponding to RGB values.
* **shape = img\_np\_arr.shape:** To obtain the dimensions or shape of the array, which in turn will be the shape of the image.
* **screen\_cap\_writer**: To write or save the screen capture to video format, we create a VideoWriter() with the parameters: output\_file\_name.avi, VideoWriter\_fourcc - a 4 character code used to compress the frames, frames per second, (width, height) of the video frames.

NOTE: Higher fps provides better results, but leads to heavier files. You can change the fourcc codes, but the output format ‘.avi’ must be checked for compatibility. You can play with other formats and fourcc codes to find the ideal combination.

* **scale\_by\_percent = 50:** Defines how much the frames must be scaled. This is optional. It is necessary to do this if you wish to view your screen recording in a separate window.
* **width, height, new\_dim:** When you scale, to maintain aspect ratio, you must reduce the width and height of the window or frames accordingly. Hence width and height, given by shape[1] and shape[0] respectively are multiplied by scale percentage/100. This new dimension is assigned to new\_dim
* **while True:** The screen recording goes on indefinitely until a keyboard interrupt is raised.
* **final\_img = cv2.cvtColor(img\_np\_arr, cv2.COLOR\_RGB2BGR):** Pillow uses RGB format whereas opencv uses BGR format. Hence we change the colour format. The first value is the numpy array and the second parameter tells the type of conversion. The other types of conversion are COLOR\_RGB2GRAY for grayscale conversion and so on.
* **screen\_cap\_writer.write(final\_img)**: Write the frame to the video.
* **image = cv2.resize(final\_img, (new\_dim))**: This is optional. To view the recording simultaneously while recording, we create a smaller window of the screen and display the images in it. Hence we resize the images by giving the image and the dimensions it must be resized to.
* **cv2.imshow("image", image):** View the resized image using imshow. The first parameter is the name of the window and the second parameter is the image.
* **if cv2.waitKey(1) == ord('e'):** cv2.waitkey(1) allows you to view the output as a video. It will freeze with the first frame if set to 0. If the user presses the key ‘e’, the ASCII value of it is taken and compared with waitkey(1). If yes ie. if ‘e’ is pressed and not other keys, the loop breaks and the recording terminates.
* **screen\_cap\_writer.release():** Release the video writer object.
* **cv2.destroyAllWindows():** This closes the imshow() windows that are open.

##### **Defining the GUI:**

#Define the user interface

screen\_recorder = Tk()

screen\_recorder.geometry("340x220")

screen\_recorder.title("PythonGeeks Screen Recorder")

bg\_img = PhotoImage(file = "/home/deepika/Downloads/image.png")

**Code explanation:**

* **screen\_recorder = Tk():** Assign the class object to screen\_recorder to use the widgets
* **screen\_recorder.geometry("340x220"):** Define the window size (width, height). Here the window is set to the dimensions of the image.
* **screen\_recorder.title("PythonGeeks Screen Recorder"):** Set a title for the application window.
* **bg\_img**: To set a background image, load the image using PhotoImage. This function is available in tkinter and it supports **only PNG** images.

##### **Create the components and the button:**

# Show image using label

label1 = Label( screen\_recorder, image = bg\_img, bd=0)

label1.pack()

#Create and place the components

title\_label = Label(screen\_recorder, text="PythonGeeks Screen Recorder", font=("Ubuntu Mono", 16), bg="#02b9e5")

title\_label.place(relx=0.5,rely=0.1, anchor=CENTER)

info\_label = Label(screen\_recorder, text="Enter 'e' to exit screen recording", bg="#02b9e5")

info\_label.place(relx=0.5,rely=0.3, anchor=CENTER)

screen\_button = Button(screen\_recorder, text="Record Screen", command=record\_screen, relief= RAISED)

screen\_button.place(relx=0.5,rely=0.6, anchor=CENTER)

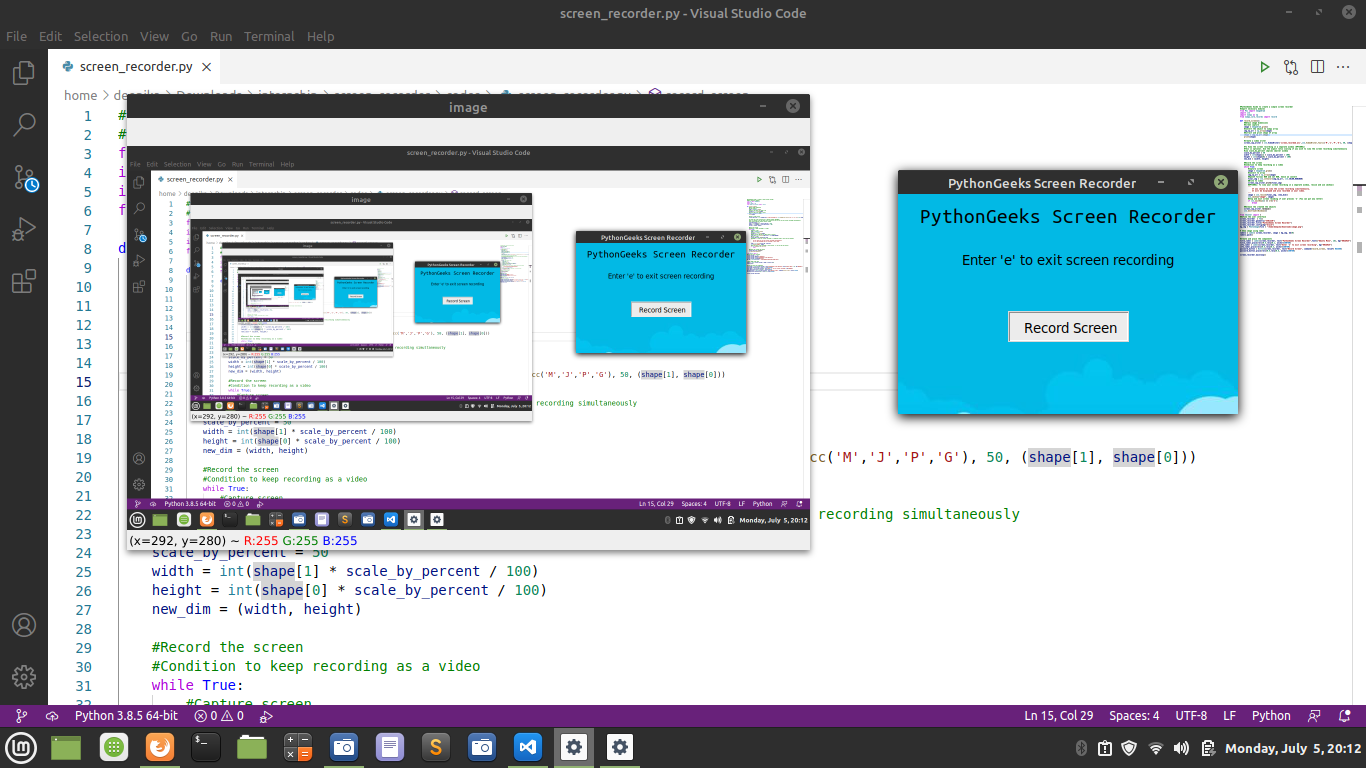
screen\_recorder.mainloop()

**Code explanation:**

* **label1:** To display the background image, use an empty label with no text attributes. The parameters are window of the application, image = loaded background image, bd=0:setting the border to 0 or no border.
* **label1.pack():** Place the label in the first row, immediately after the top margin
* **title\_label, info\_label:** Create a non editable text using Label. Here the additional parameters given are: text - text to display and font styling with bg. Font is a tuple containing the font name and the size. Specify the background colour either using the named colours available or use pick colour or eyedropper on firefox to get the value.
* **title\_label.place(), info\_lable.place():** To view the widgets, we position them. Similar to pack(), place() also positions the widget. Here we mention relx and rely which are the percentages of distance from the left and top respectively. Anchor=CENTER positions the element in the center.
* **screen\_button:** To call the declared function, we use a button. Button() creates a button with the parameters, window of the application, text: name of the button, command: the function to call, relief: styling of the button. RAISED creates a shadow effect for the button.
* **screen\_recorder.mainloop():** When the app is run, widgets and functions above mainloop() will be visible. The rest are simply ignored. Thus when a user selects exit, the control flows out of mainloop and the app terminates.

#### **Project output:**

Record the screen and view the output



#### **Summary**

Thus we created a simple Screen recorder from scratch. The project introduces opencv and a simple application of it. We also learnt the usage of pillow to capture screenshots and the difference in colour formats between RGB and BGR