SESSION 7

TOC

1. Pyautogui

Introduction to Pyautogui library:

Pyautogui is a Python library that allows you to automate tasks on your computer. It can simulate keyboard and mouse movements and clicks, take screenshots, and even perform image recognition.

Installing Pyautogui:

To install Pyautogui, you can use pip, the Python package manager, by running the following command in your terminal or command prompt:

```
pip install pyautogui
```

Pyautogui functions and methods:

Once installed, you can use Pyautogui's functions and methods in your Python script. Here are some commonly used functions and methods:

pyautogui.moveTo(x, y, duration=0.0) - Moves the mouse cursor to the given (x, y) coordinates on the screen. You can also specify the duration of the movement in seconds.

pyautogui.click(x=None, y=None, clicks=1, interval=0.0, button='left') - Performs a mouse click at the given (x, y) coordinates on the screen. You can also specify the number of clicks, the interval between clicks, and the button to use (left, right, or middle).

pyautogui.typewrite(message, interval=0.0) - Types the given message using the keyboard. You can also specify the interval between keystrokes.

pyautogui.screenshot(filename=None, region=None, **kwargs) - Takes a screenshot of the screen and saves it to a file. You can also specify a region of the screen to capture.

pyautogui.locateOnScreen(image, **kwargs) - Searches the screen for the given image and returns the coordinates of the matching region.

pyautogui.press() and pyautogui.hotkey- The functions don't return anything but perform the job of simulating of pressing the enter key and simulates pressing the hotkey ctrl+a.

Pyautogui keyboard and mouse control:

You can use Pyautogui to control the keyboard and mouse on your computer. Here's an example of how to use Pyautogui to open a new Chrome window:

In [ ]:
```
import pyautogui
import time

# Press the Windows key and type "chrome"
pyautogui.press('win')
pyautogui.typewrite('chrome')
time.sleep(1)

# Press Enter to open Chrome
pyautogui.press('enter')
time.sleep(5)

# Open a new window
pyautogui.hotkey('ctrl', 'n')
```

Taking Screenshots:

Pyautogui provides the screenshot() function to take a screenshot of the current screen. The function returns a Pillow Image object, which can be saved or manipulated further. Here's an example:

In [ ]:
```
import pyautogui

# take a screenshot
screenshot = pyautogui.screenshot()

# save the screenshot as an image file
screenshot.save('Images/screenshot.png')
```

The screenshot() function can also take an optional region argument to specify a specific region of the screen to capture. Here's an example:

In [ ]:
```
import pyautogui

# specify the region to capture
left = 100
top = 100
width = 200
height = 200

# take a screenshot of the specified region
screenshot = pyautogui.screenshot(region=(left, top, width, height))

# save the screenshot as an image file
screenshot.save('Images/screenshot2.png')
```

Image Recognition:

Pyautogui also provides functions for image recognition, which allows us to locate images on the screen and perform actions based on their position. The locateOnScreen() function can be used to locate an image on the screen and return its position as a tuple of coordinates.

Here's an example:

In [ ]:
```
import pyautogui

# locate the 'firefox' image on the screen
position = pyautogui.locateOnScreen('Images/firefox.jpeg')

# # click on the center of the located image
# pyautogui.click(position.left + position.width / 2, position.top + position.height / 2)
```

The locateAllOnScreen() function can be used to locate all instances of an image on the screen and return their positions as a list of tuples.

Here's an example:

In [ ]:
```
import pyautogui

# locate all instances of the 'firefox' image on the screen
positions = list(pyautogui.locateAllOnScreen('Images/firefox.jpeg'))
```

```
# click on the center of each located image
for position in positions:
    pyautogui.click(position.left + position.width / 2, position.top + position.height / 2)
```

---

2. OpenCV Library (opencv-python)

"opencv-python" is a Python package that provides access to OpenCV (Open Source Computer Vision Library) functions and algorithms. OpenCV is a popular open-source library used for computer vision tasks such as image and video processing, object detection, and feature extraction.

To install "opencv-python" package, you can use the following command in your Python environment:

```
pip install opencv-python
```

Once the package is installed, you can import the necessary modules and start using OpenCV functions in your Python code.

Here is an example of how you can use "opencv-python" for basic image processing tasks:

In this code, we first import the "cv2" module from the "opencv-python" package. We then read an image from a file using the cv2.imread() function. Next, we convert the image to grayscale using the cv2.cvtColor() function. We apply edge detection using the cv2.Canny() function to detect the edges in the image. Finally, we display the original image and the edges using the cv2.imshow() function, and wait for a key press to close the windows using cv2.waitKey(0).

In [ ]:

```
import cv2

# Read an image from file
image = cv2.imread('Images/image.jpg')

# Convert the image to grayscale
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Apply edge detection
edges = cv2.Canny(gray_image, 100, 200)

# Display the original image and the edges
cv2.imshow('Original Image', image)
cv2.imshow('Edges', edges)
```

```
# Wait for a key press and then close the windows
cv2.waitKey(0)
cv2.destroyAllWindows()
```

This "opencv-python" provides a wide range of functions and algorithms for various computer vision tasks. You can explore the official OpenCV documentation (https://docs.opencv.org/) for more detailed information on the available functions and their usage.

---

3. Python Imaging Library

"PIL" (Python Imaging Library) is a powerful library for opening, manipulating, and saving many different image file formats in Python. It provides a wide range of image processing capabilities, such as resizing, cropping, rotating, filtering, and more. The library is widely used in various applications that involve working with images.

To use "PIL" in your Python code, you need to install the package. You can install it using the following command:

```
pip install Pillow
```

Once installed, you can import the necessary modules and start using "PIL" functions in your code.

Here is an example that demonstrates some basic image processing operations using "PIL":

In this code, we first import the Image module from the "PIL" package. We then open an image file using the Image.open() function. We can access various properties of the image, such as its format, size, and mode.

Next, we perform some basic image processing operations. We resize the image using the resize() method, passing in the desired width and height. We convert the image to grayscale using the convert() method and specifying the mode as 'L'. We rotate the image by 45 degrees using the rotate() method.

Finally, we save the processed images using the save() method, specifying the file names. The images are saved in the specified formats, which can be JPEG, PNG, or other supported formats, depending on the file extension.

In [ ]:

```
from PIL import Image

# Open an image file
image = Image.open('Images/image.jpg')
```

```
# Display image properties
print("Image format:", image.format)
print("Image size:", image.size)
print("Image mode:", image.mode)

# Resize the image
resized_image = image.resize((500, 500))

# Convert the image to grayscale
grayscale_image = image.convert('L')

# Rotate the image
rotated_image = image.rotate(45)

# Save the processed images
resized_image.save('Images/resized_image.jpg')
grayscale_image.save('Images/grayscale_image.jpg')
rotated_image.save('Images/rotated_image.jpg')
```

```
Image format: JPEG
Image size: (872, 586)
Image mode: RGB
```

"PIL" provides many more functions and capabilities for working with images. You can refer to the official "PIL" documentation (https://pillow.readthedocs.io/) for detailed information on the available functions and their usage.

---

4. Cursor movement using PyautoGui

Cursor movement is a fundamental part of any graphical user interface (GUI). With Pyautogui, you can control the mouse cursor's movement and automate tasks that involve moving the cursor. The following Python code shows how to move the mouse cursor to a specific location on the screen:

In [ ]:

```
import pyautogui

# Move the cursor to x=100, y=100 on the screen
pyautogui.moveTo(100, 100)
```

You can also specify the duration of the cursor movement using the duration parameter:

In [ ]:
```
# Move the cursor to x=100, y=100 on the screen over 2 seconds
pyautogui.moveTo(100, 100, duration=2)
```

To move the cursor relative to its current position, you can use the move function:

In [ ]:
```
# Move the cursor 100 pixels to the right and 50 pixels down
pyautogui.move(100, 50)
```

To optimize cursor movement, you can adjust the cursor speed and acceleration using the pyautogui.PAUSE and pyautogui.MINIMUM_DURATION constants:

In [ ]:
```
# Set the delay between each PyAutoGUI function call to 0.1 seconds
pyautogui.PAUSE = 0.1

# Set the minimum duration for any PyAutoGUI function call to 0.01 seconds
pyautogui.MINIMUM_DURATION = 0.01
```

With these optimizations, Pyautogui can move the cursor quickly and accurately, reducing the time it takes to complete automation tasks.

---

5. Input device interface

Pyautogui provides various functions and methods to interact with different input devices like touchpads, touchscreens, sensors, etc. Let's see how we can use Pyautogui for input device interface:

Touchpad and touchscreen interface with Pyautogui:
To interact with touchpad and touchscreen, we can use the Pyautogui.moveTo() and Pyautogui.click() functions. The code snippet below moves the cursor to a specified location on the screen and clicks the left button on a touchpad or touchscreen:

In [ ]:
```
import pyautogui

# Move the cursor to position (x=100, y=100) and click the left button
pyautogui.moveTo(100, 100)
pyautogui.click(button='left')
```

Sensor interface with Pyautogui:

To interact with sensors, we can use the Pyautogui.typewrite() function. The code snippet below types the string 'Hello World!' as input to a sensor:

In [ ]:
```
import pyautogui

# Type 'Hello World!' as input to the sensor
pyautogui.typewrite('Hello World!')
```
