

SESSION 7

TOC

1. Pyautogui
 - 1.1 Keyboard Functions
 - 1.2 Mouse Functions
2. Cursor movement using PyautoGui
3. OpenCV Library (opencv-python)
4. Python Imaging Library
5. Input device interface

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
```

```
In [ ]: !pip install pyautogui
```

```
Collecting pyautogui
  Using cached PyAutoGUI-0.9.54.tar.gz (61 kB)
  Installing build dependencies: started
  Installing build dependencies: finished with status 'done'
  Getting requirements to build wheel: started
  Getting requirements to build wheel: finished with status 'done'
  Preparing metadata (pyproject.toml): started
  Preparing metadata (pyproject.toml): finished with status 'done'
Collecting pytweneing>=1.0.4
  Using cached pytweneing-1.0.7.tar.gz (168 kB)
  Preparing metadata (setup.py): started
  Preparing metadata (setup.py): finished with status 'done'
Collecting pygetwindow>=0.0.5
  Using cached PyGetWindow-0.0.9.tar.gz (9.7 kB)
  Preparing metadata (setup.py): started
  Preparing metadata (setup.py): finished with status 'done'
Collecting mouseinfo
  Using cached MouseInfo-0.1.3.tar.gz (10 kB)
  Preparing metadata (setup.py): started
  Preparing metadata (setup.py): finished with status 'done'
Collecting pymsgbox
  Using cached PyMsgBox-1.0.9.tar.gz (18 kB)
  Installing build dependencies: started
  Installing build dependencies: finished with status 'done'
  Getting requirements to build wheel: started
  Getting requirements to build wheel: finished with status 'done'
  Preparing metadata (pyproject.toml): started
  Preparing metadata (pyproject.toml): finished with status 'done'
Collecting pyscreeze>=0.1.21
  Downloading PyScreeze-0.1.30.tar.gz (27 kB)
  Installing build dependencies: started
  Installing build dependencies: finished with status 'done'
  Getting requirements to build wheel: started
  Getting requirements to build wheel: finished with status 'done'
  Preparing metadata (pyproject.toml): started
  Preparing metadata (pyproject.toml): finished with status 'done'
Collecting pyrect
  Using cached PyRect-0.2.0.tar.gz (17 kB)
  Preparing metadata (setup.py): started
  Preparing metadata (setup.py): finished with status 'done'
Requirement already satisfied: Pillow>=9.2.0 in c:\users\omolp091\anaconda3\lib\site-packages (from pyscreeze>=0.1.21->pyautogui) (10.1.0)
Collecting pyperclip
  Using cached pyperclip-1.8.2.tar.gz (20 kB)
```

```
Preparing metadata (setup.py): started
Preparing metadata (setup.py): finished with status 'done'
Building wheels for collected packages: pyautogui, pygetwindow, pyscreeze, pytweneing, mouseinfo, pymsgbox, pyperclip, pyrect
Building wheel for pyautogui (pyproject.toml): started
Building wheel for pyautogui (pyproject.toml): finished with status 'done'
Created wheel for pyautogui: filename=PyAutoGUI-0.9.54-py3-none-any.whl size=37597 sha256=d2f41f46a3ee1a2410ffe455faf4a5af807a1df155300798d5a9d1d02a421333
Stored in directory: c:\users\omolp091\appdata\local\pip\cache\wheels\27\ae\44\6ed17dd0eb2109b293dfd896e6b32176a662472e9ec276e911
Building wheel for pygetwindow (setup.py): started
Building wheel for pygetwindow (setup.py): finished with status 'done'
Created wheel for pygetwindow: filename=PyGetWindow-0.0.9-py3-none-any.whl size=11079 sha256=15b775682e3c4a9c513d2746648490eea1b1c83d55a292db571f67073b8a4262d
Stored in directory: c:\users\omolp091\appdata\local\pip\cache\wheels\b4\ba\41\c09cef95c23cdfed6af9df7328306fafbf27df2e77b14d25d7
Building wheel for pyscreeze (pyproject.toml): started
Building wheel for pyscreeze (pyproject.toml): finished with status 'done'
Created wheel for pyscreeze: filename=PyScreeze-0.1.30-py3-none-any.whl size=14400 sha256=bfb6023c4b77948c8612b360216f768d78ab4f657402085d4616b4465fdab326
Stored in directory: c:\users\omolp091\appdata\local\pip\cache\wheels\99\bf\0d\71d629ddfd949f89dabc0e9357f41254c723bbfb043a60d93c
Building wheel for pytweneing (setup.py): started
Building wheel for pytweneing (setup.py): finished with status 'done'
Created wheel for pytweneing: filename=pytweneing-1.0.7-py3-none-any.whl size=6214 sha256=ef110d1bedd715db9246f141039e99ad9533dbf52d77296df4360e23f3256377
Stored in directory: c:\users\omolp091\appdata\local\pip\cache\wheels\c6\4a\25\8e83858414a3d4f8d69cb04f86501b11160106b05fc9c8063b
Building wheel for mouseinfo (setup.py): started
Building wheel for mouseinfo (setup.py): finished with status 'done'
Created wheel for mouseinfo: filename=MouseInfo-0.1.3-py3-none-any.whl size=10906 sha256=4217fde488215c0a166fa4364b73a8eb5cd576b71834c2944b7039ffbd9af435
Stored in directory: c:\users\omolp091\appdata\local\pip\cache\wheels\52\e8\77\5f70ba97301c6d825754eea9f12f0ffd00e119fba64a68f4d2
Building wheel for pymsgbox (pyproject.toml): started
Building wheel for pymsgbox (pyproject.toml): finished with status 'done'
Created wheel for pymsgbox: filename=PyMsgBox-1.0.9-py3-none-any.whl size=7416 sha256=1cd3b319295035556f49b53e2f286ec64e7e345486022fcd73f5479033cef701
Stored in directory: c:\users\omolp091\appdata\local\pip\cache\wheels\97\61\22\1a0de2324b204d675c5bfb28b944101eb48dcf1a1fe87bce7
Building wheel for pyperclip (setup.py): started
Building wheel for pyperclip (setup.py): finished with status 'done'
Created wheel for pyperclip: filename=pyperclip-1.8.2-py3-none-any.whl size=11137 sha256=020c570edd8462ecbe016070cadd7a215ed9ed1db9c39b9c4bc21d569e6e0a1
Stored in directory: c:\users\omolp091\appdata\local\pip\cache\wheels\3c\77\81\aaa2802e9b0553585f2789c6f2756b50a09a01d2848423b
```

b15

```
Building wheel for pyrect (setup.py): started
```

```
Building wheel for pyrect (setup.py): finished with status 'done'
```

```
Created wheel for pyrect: filename=PyRect-0.2.0-py2.py3-none-any.whl size=11205 sha256=24244b64ada6ec79cd47b95776fcbe6b3d208378a0f86b9e4faf4aa1a44bf74f
```

```
Stored in directory: c:\users\omolp091\appdata\local\pip\cache\wheels\63\0e\55\adb1553ad973caacffc0bd779f74f4749f904da246674a26a1
```

```
Successfully built pyautogui pygetwindow pycreeze pytweneing mouseinfo pymsgbox pyperclip pyrect
```

```
Installing collected packages: pytweneing, pyrect, pyperclip, pymsgbox, pycreeze, pygetwindow, mouseinfo, pyautogui
```

```
Successfully installed mouseinfo-0.1.3 pyautogui-0.9.54 pygetwindow-0.0.9 pymsgbox-1.0.9 pyperclip-1.8.2 pyrect-0.2.0 pycreeze-0.1.30 pytweneing-1.0.7
```

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:

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.

```
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')
```

1.1 Keyboard Functions

1) PyAutoGUI provides several keyboard-related functions to simulate keyboard input. Here's an example code demonstrating some of the keyboard functions:

```
In [ ]: import pyautogui
import time

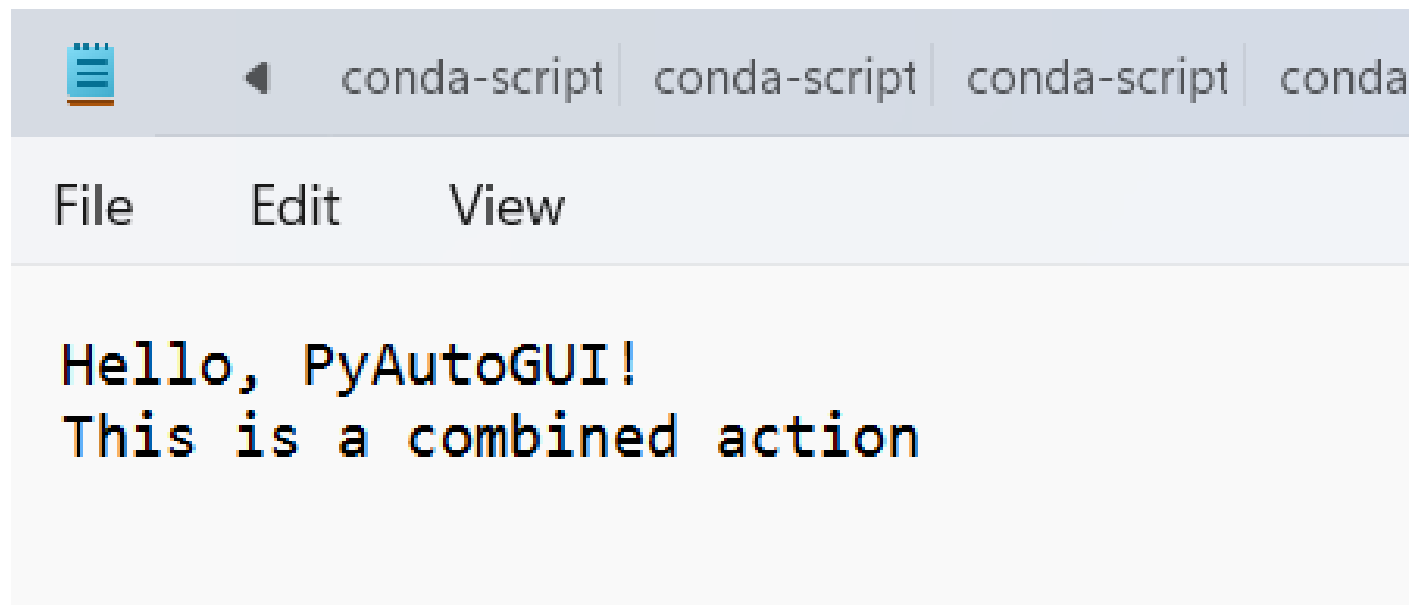
# Example: Typing text ( Open a empty notepad after execution begins )
time.sleep(5)
pyautogui.typewrite("Hello, PyAutoGUI!")

# Add a delay before the next action (in seconds)
time.sleep(1)

# Example: Pressing and releasing a key
pyautogui.press('enter')

# Add a delay before the next action (in seconds)
time.sleep(1)

# Example: Combining multiple keyboard actions
pyautogui.write('This is a combined action', interval=0.1)
```



2) PyAutoGUI provides several keyboard-related functions to simulate keyboard Copy Paste. Here's an example code demonstrating some of the keyboard functions:

```
In [ ]: import pyautogui
import time

# ( Open a empty notepad after execution begins )
# Example: Typing text
time.sleep(5)
pyautogui.typewrite("PyAutoGUI Continued!")

# Add a delay before the next action (in seconds)
time.sleep(1)

# Example: Pressing and releasing a key
pyautogui.press('enter')

# Add a delay before the next action (in seconds)
time.sleep(1)

# Example: Hotkey combination (Ctrl+C)
pyautogui.hotkey('ctrl', 'c')
```

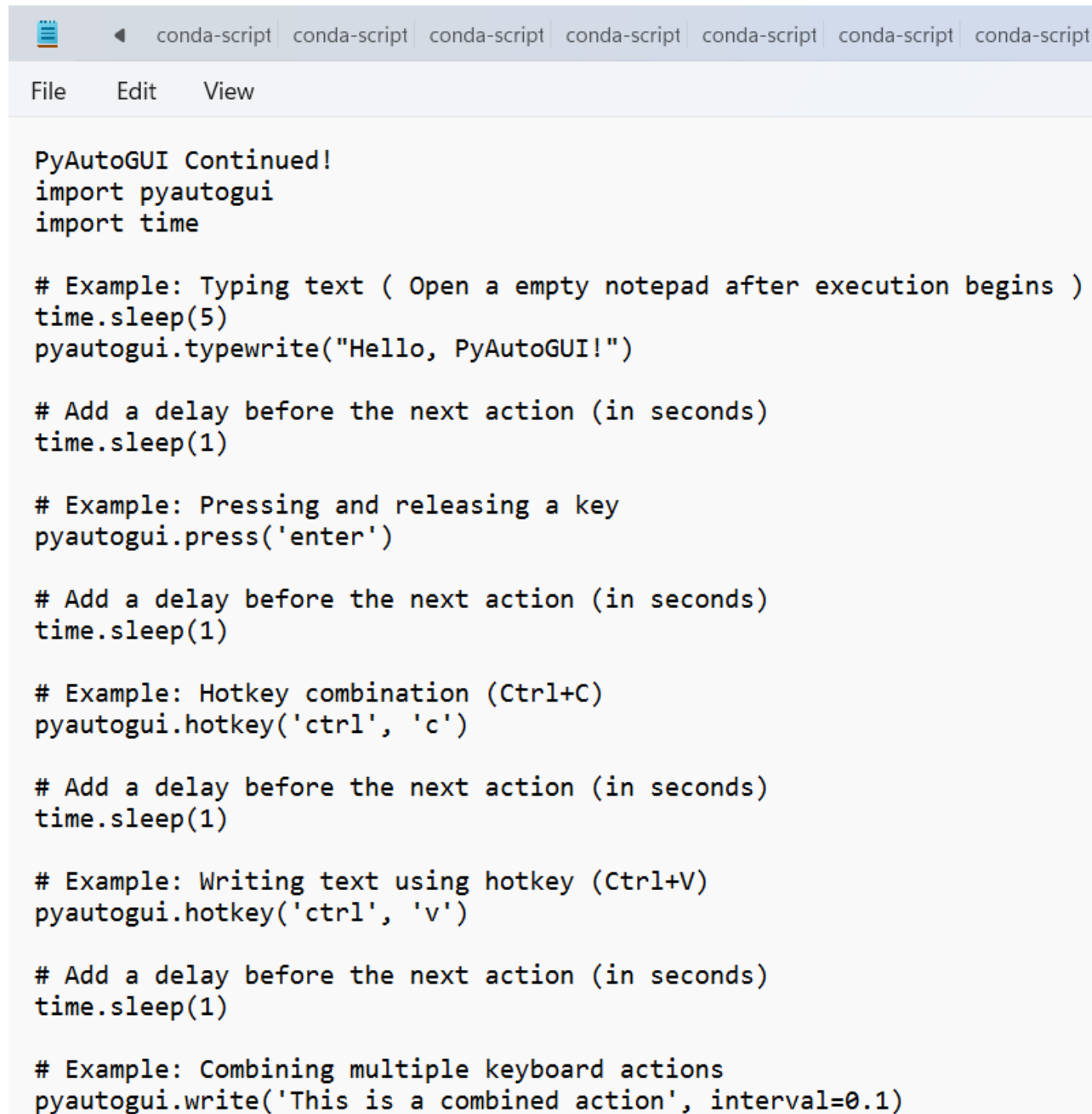
```
# Add a delay before the next action (in seconds)
time.sleep(1)

# Example: Writing text using hotkey (Ctrl+V)
pyautogui.hotkey('ctrl', 'v')

# Add a delay before the next action (in seconds)
time.sleep(1)

# Example: Combining multiple keyboard actions
pyautogui.write('This is a combined action', interval=0.1)

# Add a delay before the next action (in seconds)
time.sleep(1)
```

```
PyAutoGUI Continued!
import pyautogui
import time

# Example: Typing text ( Open a empty notepad after execution begins )
time.sleep(5)
pyautogui.typewrite("Hello, PyAutoGUI!")

# Add a delay before the next action (in seconds)
time.sleep(1)

# Example: Pressing and releasing a key
pyautogui.press('enter')

# Add a delay before the next action (in seconds)
time.sleep(1)

# Example: Hotkey combination (Ctrl+C)
pyautogui.hotkey('ctrl', 'c')

# Add a delay before the next action (in seconds)
time.sleep(1)

# Example: Writing text using hotkey (Ctrl+V)
pyautogui.hotkey('ctrl', 'v')

# Add a delay before the next action (in seconds)
time.sleep(1)

# Example: Combining multiple keyboard actions
pyautogui.write('This is a combined action', interval=0.1)
```

```
# Add a delay before the next action (in seconds)
time.sleep(1)

# Example: Pressing and holding a key (e.g., Shift)
pyautogui.keyDown('shift')
```

1.2 Mouse Functions

PyAutoGUI provides various functions to simulate mouse actions. Here's an example code demonstrating some of the mouse functions:

```
In [ ]: import pyautogui
import time

#( Open a empty notepad after execution begins )
# Example: Move the mouse to specific coordinates
time.sleep(5)
pyautogui.moveTo(100, 100, duration=1)

# Add a delay before the next action (in seconds)
time.sleep(1)

# Example: Move the mouse relative to the current position
pyautogui.move(50, 50, duration=1)

# Add a delay before the next action (in seconds)
time.sleep(1)

# Example: Click the left mouse button at the current mouse position
pyautogui.click()

# Add a delay before the next action (in seconds)
time.sleep(1)

# Example: Double-click the left mouse button at the current mouse position
pyautogui.doubleClick()

# Add a delay before the next action (in seconds)
time.sleep(1)
```

```
# Example: Right-click at the current mouse position
pyautogui.rightClick()

# Add a delay before the next action (in seconds)
time.sleep(1)

# Example: Scroll up
pyautogui.scroll(10)

# Add a delay before the next action (in seconds)
time.sleep(1)

# Example: Scroll down
pyautogui.scroll(-10)

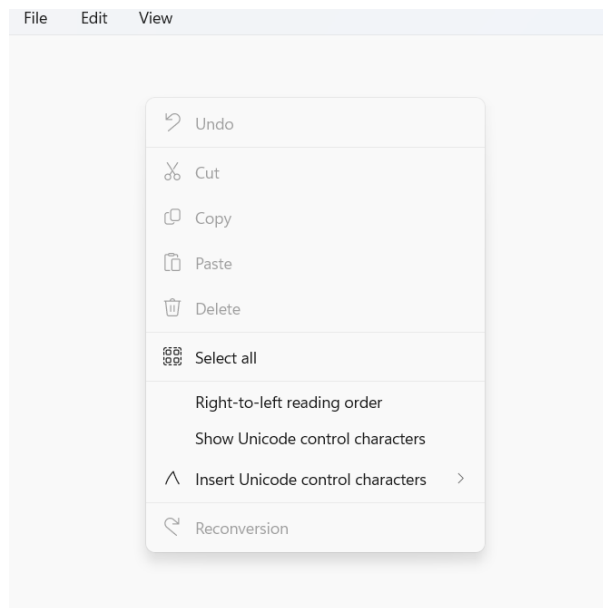
# Add a delay before the next action (in seconds)
time.sleep(1)

# Example: Drag the mouse (move to a location while holding the left button)
pyautogui.dragTo(300, 300, duration=1, button='left')

# Add a delay before the next action (in seconds)
time.sleep(1)

# Example: Drag the mouse (relative movement while holding the left button)
pyautogui.drag(50, 50, duration=1, button='left')

# Add a delay before exiting (in seconds)
time.sleep(2)
```



2. 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.

3. 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 [ ]: !pip install matplotlib
```

```
Requirement already satisfied: matplotlib in c:\users\omolp091\anaconda3\lib\site-packages (3.7.0)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\omolp091\anaconda3\lib\site-packages (from matplotlib) (2.8.2)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\omolp091\anaconda3\lib\site-packages (from matplotlib) (1.4.4)
Requirement already satisfied: packaging>=20.0 in c:\users\omolp091\anaconda3\lib\site-packages (from matplotlib) (22.0)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\omolp091\anaconda3\lib\site-packages (from matplotlib) (4.25.0)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\omolp091\anaconda3\lib\site-packages (from matplotlib) (1.0.5)
Requirement already satisfied: pillow>=6.2.0 in c:\users\omolp091\anaconda3\lib\site-packages (from matplotlib) (10.1.0)
Requirement already satisfied: numpy>=1.20 in c:\users\omolp091\anaconda3\lib\site-packages (from matplotlib) (1.23.5)
Requirement already satisfied: pyparsing>=2.3.1 in c:\users\omolp091\anaconda3\lib\site-packages (from matplotlib) (3.0.9)
Requirement already satisfied: cycycler>=0.10 in c:\users\omolp091\anaconda3\lib\site-packages (from matplotlib) (0.11.0)
Requirement already satisfied: six>=1.5 in c:\users\omolp091\anaconda3\lib\site-packages (from python-dateutil>=2.7->matplotlib) (1.16.0)
```

```
In [ ]: !pip install opencv-python
```

```
Requirement already satisfied: opencv-python in c:\users\omolp091\anaconda3\lib\site-packages (4.8.1.78)
Requirement already satisfied: numpy>=1.17.0 in c:\users\omolp091\anaconda3\lib\site-packages (from opencv-python) (1.23.5)
```

```
In [ ]: import cv2
import matplotlib.pyplot as plt

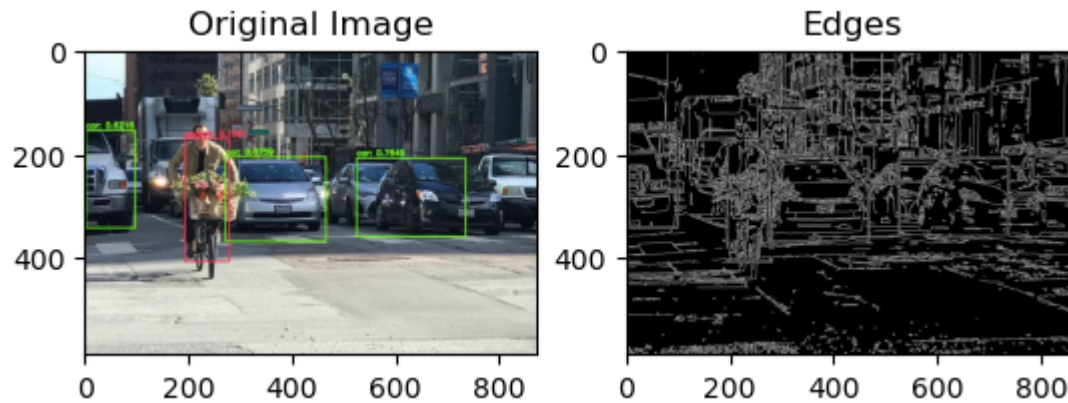
# Read an image from file
image = cv2.imread('Images/image.jpg')

# 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 using matplotlib
plt.subplot(121), plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB)), plt.title('Original Image')
plt.subplot(122), plt.imshow(edges, cmap='gray'), plt.title('Edges')

# Show the plots
plt.show()
```



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.

4. 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.

```
In [ ]: !pip install Pillow
```

```
Requirement already satisfied: Pillow in c:\users\omolp091\anaconda3\lib\site-packages (10.1.0)
```

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.

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

# Open a empty notepad to test this example
# Add a delay before the next action (in 10 seconds)
time.sleep(10)

# Type 'Hello World!' as input to the sensor

pyautogui.typewrite('I have mastered using PyAutoGUI Library !!!')
```

