Schedule

Sidheswar Routray
Department of Computer Science & Engineering
School of Technology, PDEU

Introduction to the schedule

- A Python library for scheduling and automating tasks at specific intervals.
- Enables developers to create and manage automated tasks without manual intervention.
- Task Scheduling: Set up tasks to run at specific times or intervals.
- Recurring Jobs: Define tasks that repeat regularly (daily, hourly, etc.).
- Flexible Syntax: Specify task schedules using a human-readable syntax.
- Lightweight: A simple and easy-to-use library with minimal dependencies.

Why Use schedule?

- **Automation:** Automate routine tasks and processes in your Python programs.
- **Timed Actions:** Execute functions at specific times without manual triggering.
- **Simplified Cron Jobs:** Provides an alternative to traditional cron jobs.

Functionality

- **Creating Scheduled Tasks:**
  - Use the schedule.every() function to define tasks.
  - Set the interval for the task using methods like .minutes(), .hours(), .days().
- Ex:
  - Run a function every hour:
    - def my_task():
    -     print("Task executed!")
    - schedule.every().hour.do(my_task)
  - Run a function daily at a specific time:
    - schedule.every().day.at("15:30").do(my_task)

Use the schedule library to run a Python function (task) at different intervals and specific times

```
import schedule
import time

def task():
    print("Executing Job...")

# Execute this task every 5 seconds
schedule.every(5).seconds.do(task)
# Execute this task every 5 minutes
schedule.every(5).minutes.do(task)
# Execute this task every 5 hours
schedule.every(5).hours.do(task)
# Execute this task every 5 days
schedule.every(5).days.do(task)
# Execute this task every 5 weeks
schedule.every(5).weeks.do(task)
# Execute task every Monday
schedule.every().monday.do(task)
```

```
# Execute task on Wednesday at 11:45:20
schedule.every().wednesday.at("11:45:20").do(task)

# Keep the script running to execute the tasks
# For demonstration, we'll run it for a few seconds to show the output
end_time = time.time() + 20  # Run for 20 seconds
while time.time() < end_time:
    schedule.run_pending()
    time.sleep(1)
```

Use the schedule library to run a Python function (task) at different intervals and specific times

- end_time = time.time() + 20:This line calculates the time at which the loop should stop running.
- time.time() returns the current time in seconds since the epoch, and adding 20 means the loop will run for 20 more seconds.
- while time.time() < end_time:This loop runs as long as the current time is less than end_time. In this case, the loop will continue running for 20 seconds.
- schedule.run_pending():This function checks if any scheduled tasks are due to be run and executes them.
- time.sleep(1):This line pauses the loop for 1 second on each iteration to prevent it from consuming too much CPU by constantly checking for pending tasks.

Functionality

- Checking and Running Tasks:
    - Call **schedule.run_pending()** in a loop to check and execute pending tasks.
    - Optionally, use **schedule.run_all()** to force execution of all due tasks.
- Pausing and Canceling Tasks:
    - Pause a task with **.pause()** and resume with **.resume().**
    - Cancel a task with **.cancel()**.

Continue

- Use Cases:
  - Data Scraping: Automate web scraping tasks at regular intervals.
  - Notifications: Schedule sending automated alerts or reminders.
  - Data Processing: Automate data processing and transformation.
  - Regular Maintenance: Perform routine tasks such as backups and cleanup.
- Limitations and Considerations:
  - May not be suitable for extremely complex scheduling needs.
  - Consider using external tools for more advanced scheduling requirements.

File backup task

- Create a scheduler to perform backup everyday 6:30pm. In backup function you will pass source and destination folder.

```
import schedule
import time
import shutil
import os
def backup(source_folder, destination_folder):
    """
    Function to back up files from the source folder to the
destination folder.
    """
    try:
        if not os.path.exists(destination_folder):
            os.makedirs(destination_folder)

        for filename in os.listdir(source_folder):
            source_file = os.path.join(source_folder, filename)
            destination_file = os.path.join(destination_folder,
filename)
```

```
 if os.path.isfile(source_file):
                shutil.copy2(source_file, destination_file)
                print(f"Backed up: {filename}")
    except Exception as e:
        print(f"Error during backup: {e}")

# Define the source and destination folders
source_folder = "/path/to/source_folder"
destination_folder = "/path/to/destination_folder"

# Schedule the backup task to run every day at 6:30 PM
schedule.every().day.at("18:30").do(backup, source_folder,
destination_folder)

# Keep the script running to execute the scheduled task
while True:
    schedule.run_pending()
    time.sleep(1)
```

File backup task

backup Function:The backup function takes source_folder and destination_folder as arguments.

- It copies files from the source folder to the destination folder.
- If the destination folder doesn't exist, it creates it.
- It uses shutil.copy2() to copy each file, preserving metadata.

Scheduling:schedule.every().day.at("18:30").do(backup, source_folder, destination_folder) schedules the backup function to run every day at 6:30 PM.

- Running the Scheduler:The while True loop keeps the script running and continuously checks for any pending tasks that need to be executed.
- schedule.run_pending() runs any tasks that are due.
- time.sleep(1) pauses the loop for one second to avoid excessive CPU usage.

pyautogui

Introduction to the pyautogui

- A Python library for automating GUI interactions by simulating mouse and keyboard actions.
- Enables programmatic control of the mouse, keyboard, and screen.
- Automate User Interactions: Mimic human actions like clicking, typing, scrolling, and moving the mouse.
- Screen Capture: Capture screenshots and record screen activities.
- Platform Independent: Works on Windows, macOS, and Linux.

Why Use pyautogui?

- Task Automation: Automate repetitive GUI tasks to save time and effort.
- UI Testing: Automate user interface testing by simulating user actions.
- Data Entry: Simulate keyboard input for data entry tasks.

Functionality

- Mouse Interactions:
  - Simulate mouse actions: click(), doubleClick(), rightClick(), moveTo(), etc.
  - Get the current mouse position with position().
  - Drag and drop using dragTo() and drag().
- Keyboard Interactions:
  - Simulate key presses: press(), keyDown(), keyUp().
  - Type text using typewrite().
  - Simulate key combinations like ctrl, alt, and shift.
- Screen Capture and Recognition:
  - Capture the screen or a specific region using screenshot().
  - Locate an image on the screen using locateOnScreen().

Basic Mouse Control

```
# returns the monitor size
import pyautogui
screenWidth, screenHeight = pyautogui.size()
print("The Screen Width is: ", screenWidth)
print("The Screen Height is: ", screenHeight)
```

Move the Mouse to a Specific Location:

```
import pyautogui
# Move the mouse to the center of the screen
screenWidth, screenHeight = pyautogui.size()
pyautogui.moveTo(screenWidth / 2, screenHeight / 2)
```

Basic Mouse Control

```
# Perform a left-click
pyautogui.click()
# Perform a right-click
pyautogui.rightClick()


# Move the mouse to the top-left corner and drag to the right
pyautogui.moveTo(100, 100)
pyautogui.dragTo(400, 100, duration=1)  # drag to (400, 100) over 1 second
```

Keyboard Automation

```
import pyautogui
import time
# Open Notepad
pyautogui.press('win')
pyautogui.write('Notepad')
pyautogui.press('enter')
time.sleep(1)
# Type a message and save it
pyautogui.write("Hello, world!")
pyautogui.hotkey('ctrl', 's')
```

Taking Screenshots

```
import pyautogui
# Take a screenshot
screenshot = pyautogui.screenshot()
# Save the screenshot
screenshot.save('screenshot.png')

# Take a screenshot of a specific region (x, y, width, height)
region_screenshot = pyautogui.screenshot(region=(0, 0, 300, 400))
region_screenshot.save('region_screenshot.png')
```

Finding an Image on the Screen

```
import pyautogui
# Locate an image on the screen
location = pyautogui.locateOnScreen('example.png')
if location:
    print(f"Image found at: {location}")
# Move the mouse to the center of the located image
    pyautogui.moveTo(location.left + location.width / 2, location.top +
location.height / 2)
else:
    print("Image not found!")

# If the image is found, click on it
if location:
    pyautogui.click(location)
```

Combining Keyboard and Mouse Automation

```
import pyautogui
import time
# Open the default web browser (works on Windows)
pyautogui.hotkey('win', 'r')
pyautogui.write('chrome')
pyautogui.press('enter')
time.sleep(2)
# Search for 'PyAutoGUI'
pyautogui.write('PyAutoGUI')
pyautogui.press('enter')
time.sleep(2)
# Take a screenshot of the results
pyautogui.screenshot('search_results.png')
```

Continue

- Safety Measures:
  - Use delay functions (time.sleep()) to prevent accidental actions.
  - Always monitor the script to avoid unintended consequences.
- Use Cases:
  - Data Entry: Automate repetitive typing tasks.
  - GUI Testing: Simulate user interactions for testing UI.
  - Repetitive Tasks: Automate tasks involving clicking and scrolling.
- Safety Warnings:
  - Use with care: Mistakes can result in unintended actions.
  - Avoid running scripts unattended or without proper testing.

```
import pyautogui
screenWidth, screenHeight = pyautogui.size() # Get the size of the primary monitor.
print(screenWidth, screenHeight) # (2560, 1440)
currentMouseX, currentMouseY = pyautogui.position() # Get the XY position of the mouse.
print(currentMouseX, currentMouseY) #(1314, 345)
pyautogui.moveTo(100, 150) # Move the mouse to XY coordinates.
pyautogui.click() # Click the mouse.
pyautogui.click(100, 200) # Move the mouse to XY coordinates and click it.
pyautogui.click('button.png') # Find where button.png appears on the screen and click it.
pyautogui.move(400, 0) # Move the mouse 400 pixels to the right of its current position.
pyautogui.doubleClick() # Double click the mouse.
pyautogui.moveTo(500, 500, duration=2, tween=pyautogui.easeInOutQuad) # Use tweening/easing function to
move mouse over 2 seconds.
pyautogui.write('Hello world!', interval=0.25) # type with quarter-second pause in between each key
pyautogui.press('esc') # Press the Esc key. All key names are in pyautogui.KEY_NAMES
with pyautogui.hold('shift'): # Press the Shift key down and hold it.
        pyautogui.press(['left', 'left', 'left', 'left']) # Press the left arrow key 4 times. # Shift key is released
automatically.
pyautogui.hotkey('ctrl', 'c') # Press the Ctrl-C hotkey combination.
pyautogui.alert('This is the message to display.') # Make an alert box appear and pause the program until OK is
clicked.
```