AUTOMATION

Python Automation

1.Intro

- create and activate virtual environments to avoid python version clashes.
- install modules in that venv
- python -m venv <env_name>
- cd <env_name>
- scripts\activate in the location --> cmnd prompt changes
- pip freeze
- pip install <module_name> --> install modules & dependencies
- every module has an attribute "_ _ name _ _" where its value is "_ _ main_ _"
  - in animal.py module --> "_ _ name _ _" = "_ _ main_ _" #only inside module
  - on importing animal.py in test.py :
    - test.py : import animal.py
    - print(_ _ name _ _) --> "_ _ animal_ _"
  - To run a part of code in module or test a module, you can run the module with main
    - if "_ _ name _ _" = "_ _ main_ _" :
      return func()
- class --> imported from module
- module name starts with lowercase
- import - os , pandas, re , requests , random, queue , sqlite3, shutil, time, statistics

1. Excel Automation

- openpyxl
  - third party module
  - create excel
  - add sheets, data
  - read data
- pip install openpyxl
- import openpyxl
- wb = openpyxl.Workbook()

2.Web automation

- Response 200 from url – success
- beautiful soup – static website extraction
- selenium – dynamic website extraction (req. Chrome driver – depends on chrome version and browser)
- To install webdriver - https://googlechromelabs.github.io/chrome-for-testing/
- html – recursive datastructure , xml doc, xpath for elements access
- pat = "https?:[^\s]+"
- pythonanywhere.com – free cloud for website by ANACONDA

3.Test Automation

- Pytest
  - import pytest
  - Set up & teardown steps.
  - Testcases defined as a function, func_name starting with 'test' - def test_addition():
  - Assert – check if True
  - log levels – info, warnings, debug, error, fatal, trace, verbose
  - Run – pytest <filename>
    - pytest -v <filename> #entire testcase
    - pytest -v <filename>::<testcasename> #single testcase
    - pytest -v <filename> -m <groupedname> #mark & group testcases
      @pytest.mark.<group_name> before testcase name - @pytest.mark.Negetive

  @pytest.mark.wifitestcase
  def test_addition

@pytest.mark.skip (Not yest implemented), @pytest.mark.parameterize
- pytest.ini file – to create custom markers included in pytest
- html report – pytest_html
  pytest -v filename _ _html = report.html
- whitebox testing --> pytest, unittest library

Robot framework

1.Intro

- Keyword driven , setup & teardown
- pip install robotframework
- Pycharm – plugins, hyper framework support
- Test scipt Components:
  - *Settings (***Settings***) - documentation, metadata, Library
    import Robot file --> Res //df//d///
    import python --> Lib //df///
  - *Variables – global variables for the test suite :
  - ${var_name} var_value
    @{my_list} 1 2 3 4 5
    &{my_dict} name = Tessolve Place = Bangalore
  - *Keywords – Custom test function in Robot framework, written in py
    Just Add
    [arguments] ${a} ${b}
    ${result} Evaluate
  - *Testcases
- List ops
  - Append to list ${my_list} 8
  - Log to console ${my_list}
  - remove from list ${my_list} 2
  - ${value} Get from list ${my_list} 3
  - Log to console value: ${value}
- Dict ops
  - Set to dictionary ${my_dict} nation = India
  - Log to console ${my_dict}
  - Get from dictionary ${my_dict} place

Automation Scenarios

1. File Monitoring & Auto-Processing :
   - `You are asked to monitor a directory. Whenever a new .csv file is dropped, the script should automatically process it, extract data, and move it to an archive folder.`

2. Web Automation with Error Handling :
   - `Automate a login to a website using Selenium, but the website sometimes loads slowly, causing failures.`

3. API Automation with Retry Mechanism
   - `You are consuming an API that sometimes fails with a 503 Service Unavailable. You need to retry automatically with exponential backoff.`

4. Parallel Test Execution
   - `You need to run multiple test cases in parallel to reduce execution time.`

5. Log Parsing & Alerting
   - `You have a log file that updates continuously. You must detect if "ERROR" appears and immediately alert.`

Solutions

1)

```
import time
import shutil
import os
import pandas as pd

WATCH_DIR = "incoming_files"
ARCHIVE_DIR = "archive"

def process_file(file_path):
    df = pd.read_csv(file_path)
    print("Processed rows:", len(df))
    # Add data processing logic here...

def monitor_folder():
    seen_files = set()
    while True:
        for filename in os.listdir(WATCH_DIR):
            if filename.endswith(".csv") and filename not in seen_files:
                file_path = os.path.join(WATCH_DIR, filename)
                process_file(file_path)
                shutil.move(file_path, os.path.join(ARCHIVE_DIR,
filename))
                seen_files.add(filename)
        time.sleep(2)

monitor_folder()
```

2)

```
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC

driver = webdriver.Chrome()
driver.get("https://example.com/login")

try:
    username = WebDriverWait(driver, 10).until(
        EC.presence_of_element_located((By.ID, "username"))
    )
    username.send_keys("my_user")

    password = driver.find_element(By.ID, "password")
    password.send_keys("my_password")

    login_button = driver.find_element(By.ID, "loginBtn")
    login_button.click()
except Exception as e:
    print("Error during login:", e)
finally:
    driver.quit()
```

3)

```
import requests
import time

def fetch_with_retry(url, retries=5, backoff=2):
    for i in range(retries):
        try:
            response = requests.get(url, timeout=5)
```

```
            if response.status_code == 200:
                return response.json()
        except requests.exceptions.RequestException:
            pass
        print(f"Retrying in {backoff**i} seconds...")
        time.sleep(backoff**i)
    raise Exception("API request failed after retries")

data = fetch_with_retry("https://api.example.com/data")
print(data)
```

4)

```
import multiprocessing
import time

def run_test(test_id):
    print(f"Running test {test_id}")
    time.sleep(2)
    print(f"Test {test_id} finished")

if __name__ == "__main__":
    tests = [1, 2, 3, 4, 5]
    with multiprocessing.Pool(processes=3) as pool:
        pool.map(run_test, tests)
```

5)

```
import time

def monitor_log(file_path):
    with open(file_path, "r") as f:
        f.seek(0, 2)  # Move to end of file
        while True:
            line = f.readline()
            if not line:
                time.sleep(1)
                continue
            if "ERROR" in line:
                print("ALERT! Error detected:", line.strip())

monitor_log("application.log")
```