- Initial VS Code setup instructions.
- Detailed slide content for each topic.
- 10 exercises per slide.
- 2 real-world use cases for each section, with code and solutions.

SECTION 2 – Scripting – Automation Angle

Initial Setup with VS Code for Automation Projects

1. Install VS Code from https://code.visualstudio.com.
2. Install Python (□3.9) from https://python.org/downloads.
3. Install Git from https://git-scm.com (optional but recommended).
4. Add Extensions in VS Code:
   - Python (Microsoft)
   - Pylance
   - Code Runner (optional for quick runs)
   - GitLens (optional for Git integration)
5. Setup Python Interpreter:
   - Press `Ctrl+Shift+P` → "Python: Select Interpreter" → Choose installed Python version.
6. Create Virtual Environment:

   ```
   python -m venv venv
   ```

   Activate it:

   - Windows: `venv\Scripts\activate`
   - Mac/Linux: `source venv/bin/activate`
7. Install Required Libraries:

   ```
   pip install requests schedule selenium pytest
   ```
8. Configure Integrated Terminal:
   - 'Ctrl+" → Choose bash or PowerShell.
9. Enable Auto Save & Linting in Settings.
10. Test Setup:

    ```
    print("VS Code + Python setup complete!")
    ```

Slide 8 – Shell Scripting Basics

Content:

- Running commands via `.sh` files.
- Variables, loops, conditionals.
- File operations.
- Scheduling with `cron` (Linux/Mac) or Task Scheduler (Windows).

10 Exercises:

1. Write a shell script to print “Hello, Automation”.
2. List all `.txt` files in a directory.
3. Create a script to backup a folder to another location.
4. Write a loop that prints numbers 1 to 10.
5. Create a script that checks disk space and logs warnings.
6. Script that pings a website and logs status.
7. Find and replace a word in all `.txt` files.
8. Write a script to count lines in a file.
9. Create a daily cron job to compress logs.
10. Schedule a script to delete files older than 7 days.

Slide 9 – Python Automation – Files

Content:

- Reading/writing text, CSV, JSON.
- File manipulation with `os` & `shutil`.
- Error handling with `try/except`.

10 Exercises:

1. Read a file and print each line.
2. Count words in a file.
3. Copy a file to another folder.
4. Merge contents of two files.
5. Convert CSV to JSON.
6. Append text to an existing file.
7. Delete a file safely.
8. Rename files in bulk.
9. Search for a keyword in files.
10. Track file changes over time.

Slide 10 – Python Automation – Regex

Content:

- Pattern matching with `re`.
- Extracting emails, phone numbers, dates.
- Replacing patterns.

10 Exercises:

1. Extract all numbers from a string.
2. Validate an email format.
3. Find all dates in `DD/MM/YYYY` format.
4. Mask phone numbers.
5. Replace all spaces with underscores.
6. Extract hashtags from a tweet.
7. Validate strong password pattern.
8. Extract all words starting with capital letters.
9. Find repeated words in a text.
10. Remove all HTML tags from a string.

Slide 11 – Python Automation – Scheduling

Content:

- Scheduling with `schedule` module.
- Automating recurring tasks.
- Integrating with cron.

10 Exercises:

1. Schedule a function to print time every minute.
2. Automate daily log cleanup.
3. Schedule a script to send emails every morning.
4. Automate file backup every Sunday.
5. Run a web scraper every 2 hours.
6. Schedule database cleanup.
7. Automate report generation at midnight.
8. Schedule an API call every 10 minutes.
9. Run a function every working day at 9 am.
10. Schedule script termination after 1 hour.

Section 2 – Use Cases

Use Case 1: Daily Log Archiving

```python
import schedule, shutil, datetime

def archive_logs():
    date_str = datetime.datetime.now().strftime("%Y-%m-%d")
    shutil.make_archive(f"logs_backup_{date_str}", 'zip', 'logs')

schedule.every().day.at("23:59").do(archive_logs)

while True:
    schedule.run_pending()
```

✅ Solution: Keeps logs compressed daily, reducing disk usage.

Use Case 2: Automated Email Report

```
import smtplib
from email.mime.text import MIMEText
import schedule

def send_report():
    msg = MIMEText("Daily report content here.")
    msg['Subject'] = "Daily Report"
    msg['From'] = "me@example.com"
    msg['To'] = "team@example.com"
    with smtplib.SMTP('smtp.example.com') as server:
        server.login("me@example.com", "password")
        server.send_message(msg)

schedule.every().day.at("08:00").do(send_report)

while True:
    schedule.run_pending()
```

✅ Solution: Sends automated updates without manual intervention.

SECTION 3 – Software Testing Basics

Initial Setup with VS Code for Testing

1. Install Python & VS Code as above.
2. Install `pytest`, `selenium`, and `unittest`:

   `pip install pytest selenium`
3. Install ChromeDriver or GeckoDriver for Selenium.
4. Create a `tests/` folder for test files.
5. Configure VS Code Python testing:
   - `Ctrl+Shift+P` → "Python: Configure Tests" → pytest/unittest.
6. Enable test explorer in VS Code for visual execution.

Slide 12 – Manual Testing

10 Exercises:

1. Write test cases for login functionality.
2. Test a signup form with invalid inputs.
3. Verify error messages appear when fields are empty.
4. Test password reset functionality.
5. Test search results filtering.
6. Test a shopping cart checkout process.
7. Verify UI alignment on multiple devices.
8. Check language translation on a multilingual site.
9. Test file upload feature.
10. Perform regression testing after code changes.

Slide 13 – Selenium Intro

10 Exercises:

1. Open a webpage in Chrome using Selenium.
2. Locate an element by ID.
3. Locate an element by XPath.
4. Click a button using Selenium.
5. Enter text into a form field.
6. Take a screenshot of a page.
7. Extract all links from a webpage.
8. Test login functionality.
9. Navigate between multiple browser tabs.
10. Scroll to bottom of the page.

Slide 14 – Test Cases

10 Exercises:

1. Write positive test cases for a search feature.
2. Write negative test cases for login.
3. Write test cases for form field validation.
4. Test cases for API response validation.
5. Test cases for session timeout.
6. Test cases for data export.
7. Test cases for pagination.
8. Test cases for sorting.

9. Test cases for user roles/permissions.
10. Test cases for 404 and error pages.

Slide 15 – JUnit

(Since JUnit is Java-based, mention Python equivalent: `unittest`.) 10 Exercises:

1. Create a unittest test case class.
2. Write `setUp()` and `tearDown()` methods.
3. Test addition function.
4. Test subtraction function.
5. Test API call returns 200.
6. Test file read function.
7. Test database insert.
8. Test exception handling.
9. Test object equality.
10. Test string contains substring.

Slide 16 – Unit Testing

10 Exercises:

1. Unit test a function that reverses strings.
2. Unit test a function that sums numbers.
3. Unit test a factorial function.
4. Test a prime number checker.
5. Test a palindrome checker.
6. Test a sorting function.
7. Test a function that removes duplicates.
8. Test an API parser function.
9. Test a JSON to CSV converter.
10. Test a simple authentication function.

Section 3 – Use Cases

Use Case 1: Automated Web Login Test (Selenium)

```
from selenium import webdriver

driver = webdriver.Chrome()
driver.get("https://example.com/login")
driver.find_element("id", "username").send_keys("testuser")
driver.find_element("id", "password").send_keys("password123")
driver.find_element("id", "loginBtn").click()

assert "Dashboard" in driver.title
driver.quit()
```

✅ Solution: Ensures login works after updates.

Use Case 2: Unit Test for Data Cleaning Function

```
import unittest

def clean_data(data):
    return [d.strip().lower() for d in data if d]

class TestCleanData(unittest.TestCase):
    def test_clean(self):
        self.assertEqual(clean_data([" A ", None, "B "]), ["a", "b"])

if __name__ == "__main__":
    unittest.main()
```

✅ Solution: Verifies data cleaning logic works with various inputs.
