

Perfect! Let's start with the first **two in-depth chapters** of your **Python Scripting Notes**. These include real-world examples and clear explanations.

Chapter 1: Introduction to Python Scripting

◇ What is Scripting?

Scripting means writing a small piece of code to automate tasks that could otherwise be done manually. It's generally simpler and faster to write than full-blown applications.

Examples:

- Renaming multiple files in a folder
- Automatically sending emails
- Logging system activity
- Web scraping data from a site

◇ Difference Between Programming and Scripting

Feature	Programming	Scripting
Purpose	Build complete apps/systems	Automate small tasks
Compilation	Compiled (e.g., Java, C++)	Interpreted (e.g., Python, JS)
Execution time	Generally faster	Slightly slower (interpreted)
Examples	C++, Java	Python, JavaScript, Bash

◇ Why Python for Scripting?

Advantage	Explanation
Simple Syntax	Easy to read, write, and understand
Huge Libraries	os, sys, requests, csv, sqlite3, etc.
Cross-platform	Runs on Windows, Linux, macOS
Community	Vast tutorials, documentation, and
Support	forums

◇ Types of Tasks You Can Automate with Python

- File/folder operations
- API calls
- System monitoring
- Excel sheet handling
- Web scraping and browser automation
- Sending emails or notifications

◇ Example: Basic Script to Rename Files

```
import os

folder_path = 'C:/Users/You/Documents/Photos'
for i, filename in enumerate(os.listdir(folder_path)):
    new_name = f"image_{i+1}.jpg"
    src = os.path.join(folder_path, filename)
    dst = os.path.join(folder_path, new_name)
    os.rename(src, dst)
```

✅ This script renames all files in a folder to image_1.jpg, image_2.jpg, etc.

Chapter 2: Python Basics Refresher (For Scripting)

◇ Variables and Data Types

```
name = "Alice"  
age = 25  
height = 5.4  
is_admin = True
```

Data Type	Example
String	"hello"
Integer	42
Float	3.14
Boolean	True/False
List	[1, 2, 3]
Tuple	(1, 2)
Dictionary	{"a": 1}

◇ Control Structures

If-Else

```
age = 20  
if age >= 18:  
    print("Adult")  
else:  
    print("Minor")
```

Loops

```
for i in range(5):  
    print(i)
```

```
x = 0  
while x < 5:  
    print(x)  
    x += 1
```

◆ Functions

```
def greet(name):  
    return f"Hello, {name}"  
  
print(greet("Alice"))
```

Default and Keyword Arguments

```
def add(a=0, b=0):  
    return a + b  
  
print(add(b=3, a=2))
```

◆ Data Structures: List, Tuple, Dictionary

List

```
fruits = ["apple", "banana", "cherry"]  
fruits.append("orange")  
print(fruits[0])
```

Tuple (Immutable)

```
coordinates = (10, 20)
print(coordinates[1])
```

Dictionary

```
user = {"name": "Bob", "age": 25}
print(user["name"])
```

◆ Exception Handling

```
try:
    number = int(input("Enter a number: "))
    print(100 / number)
except ValueError:
    print("Invalid number!")
except ZeroDivisionError:
    print("Cannot divide by zero!")
finally:
    print("End of script.")
```

Great! Here are the next two chapters with detailed explanations and practical code examples.

Chapter 3: File and Directory Handling in Python

◇ Why File Handling?

Python scripts often interact with files—for reading logs, saving output, or modifying documents.

◇ Opening a File

```
file = open("example.txt", "r") # modes: 'r', 'w', 'a', 'b', 'x'
content = file.read()
file.close()
```

Better way (with with block):

```
with open("example.txt", "r") as file:
    content = file.read()
    print(content)
```

◇ Writing to a File

```
with open("output.txt", "w") as file:
    file.write("This is a new file.\n")
    file.write("Hello, Python scripting!")
```

- 'w' mode overwrites
- 'a' mode appends

◇ Reading Line by Line

```
with open("example.txt", "r") as file:
    for line in file:
```

```
print(line.strip())
```

◆ File Modes Table

Mode	Meaning
'r'	Read
'w'	Write (overwrites)
'a'	Append
'x'	Exclusive creation
'b'	Binary mode
't'	Text mode (default)

◆ OS Module – Directory Operations

```
import os

print(os.getcwd()) # Current working dir
os.mkdir("new_folder")
os.rename("old.txt", "new.txt")
os.remove("file_to_delete.txt")
```

◆ List Files in a Directory

```
files = os.listdir("C:/Users/YourName/Documents")
for file in files:
    print(file)
```

◇ Check If File or Folder Exists

```
import os

if os.path.exists("sample.txt"):
    print("File exists!")
```

◇ Shutil Module (Copy/Move Files)

```
import shutil

shutil.copy("source.txt", "backup.txt")
shutil.move("backup.txt", "folder/backup.txt")
```

◇ Example: Rename All .txt Files to .bak

```
import os

for filename in os.listdir():
    if filename.endswith(".txt"):
        base = os.path.splitext(filename)[0]
        os.rename(filename, base + ".bak")
```

Chapter 4: Automating Tasks with Python

Python is great for scripting **repetitive tasks**, such as:

- Opening apps
- Sending notifications
- System monitoring
- Scheduled jobs

◇ Running System Commands with subprocess

```
import subprocess

subprocess.run(["echo", "Hello from Python!"])
```

Example: Ping a Website

```
subprocess.run(["ping", "google.com"])
```

◇ Using time Module for Delays

```
import time

print("Waiting for 3 seconds...")
time.sleep(3)
print("Done!")
```

◇ Automating Time-based Jobs with schedule

Install:

```
pip install schedule
import schedule
import time
```

```
def greet():
    print("Good morning!")

schedule.every(10).seconds.do(greet)

while True:
    schedule.run_pending()
    time.sleep(1)
```

◇ Example: Rename Files Every Day at a Set Time

```
import schedule
import time
import os

def rename_files():
    for i, file in enumerate(os.listdir()):
        if file.endswith(".log"):
            os.rename(file, f"logfile_{i+1}.log")
    print("Files renamed")

schedule.every().day.at("18:00").do(rename_files)

while True:
    schedule.run_pending()
    time.sleep(1)
```


◇ Automate Email Sending (Simple Example)

```
import smtplib

sender = "youremail@gmail.com"
receiver = "friend@gmail.com"
password = "yourpassword"
```

```
message = """Subject: Test Email\nThis is an automated message from Python."""
```

```
with smtplib.SMTP_SSL("smtp.gmail.com", 465) as server:  
    server.login(sender, password)  
    server.sendmail(sender, receiver, message)
```

 **Note:** You may need to enable "less secure apps" in your Gmail settings or use App Passwords.

Awesome! Let's now dive into:

Chapter 5: Web Scraping with requests and BeautifulSoup

Chapter 6: JSON, CSV Handling and Working with APIs

Chapter 5: Web Scraping with Python

◇ What is Web Scraping?

Web scraping is the process of extracting data from websites automatically using code.

◆ Required Libraries

Install:

```
pip install requests
pip install BeautifulSoup4
```

◆ Step-by-Step Scraping Example

```
import requests
from bs4 import BeautifulSoup

url = "https://www.example.com"
response = requests.get(url)

soup = BeautifulSoup(response.text, 'html.parser')
print(soup.prettify()) # Prints entire HTML in a readable format
```

◆ Extract Specific Elements

```
title = soup.title.text
print("Page title:", title)

all_links = soup.find_all('a')
for link in all_links:
    print(link.get('href'))
```

◆ Find by Tag, ID, Class

```
# Find first heading tag
heading = soup.find('h1')
```

```
# Find by class
paragraphs = soup.find_all('p', class_='info')

# Find by id
info = soup.find(id='main')
```

◇ Example: Scrape Quotes from a Site

```
url = "http://quotes.toscrape.com"
res = requests.get(url)
soup = BeautifulSoup(res.text, 'html.parser')

quotes = soup.find_all('span', class_='text')
authors = soup.find_all('small', class_='author')

for q, a in zip(quotes, authors):
    print(f"{q.text} - {a.text}")
```

◇ Handling Headers / Pretending as Browser

```
headers = {
    "User-Agent": "Mozilla/5.0"
}
res = requests.get("https://example.com", headers=headers)
```

◇ Notes:

- Not all sites allow scraping. Check `robots.txt`.
- For dynamic sites, use Selenium (covered later).

☑ Chapter 6: Working with JSON, CSV and APIs

◇ JSON in Python

```
import json

# Reading JSON from file
with open('data.json') as f:
    data = json.load(f)
    print(data['name'])

# Writing JSON
sample = {"name": "Alice", "age": 30}
with open('output.json', 'w') as f:
    json.dump(sample, f)
```

◇ JSON from Web APIs

```
import requests

res = requests.get("https://api.github.com/users/octocat")
user_data = res.json()
print(user_data['login'], user_data['public_repos'])
```

◇ CSV Handling in Python

```
import csv

# Reading CSV
```

```

with open("data.csv") as f:
    reader = csv.reader(f)
    for row in reader:
        print(row)

# Writing CSV
data = [{"name", "age"}, ["Alice", 25], ["Bob", 30]]
with open("output.csv", "w", newline='') as f:
    writer = csv.writer(f)
    writer.writerows(data)

```

◆ CSV with Dicts

```

# Writing dict-style rows
data = [
    {"name": "Alice", "age": 24},
    {"name": "Bob", "age": 30}
]

with open("dict_output.csv", "w", newline='') as f:
    writer = csv.DictWriter(f, fieldnames=["name", "age"])
    writer.writeheader()
    writer.writerows(data)

```

◆ API Project Example: Get Weather Info

```

import requests

city = "Chennai"
api_key = "your_api_key" # Use OpenWeatherMap API
url =
f"http://api.openweathermap.org/data/2.5/weather?q={city}&appid={api_k
ey}"

```

```
response = requests.get(url)
weather = response.json()

if weather["cod"] == 200:
    print("Temperature:", weather["main"]["temp"])
    print("Condition:", weather["weather"][0]["description"])
else:
    print("City not found.")
```

Great! Let's now move forward with:

Chapter 7: GUI Automation with pyautogui

Chapter 8: Web Automation with Selenium

Chapter 7: GUI Automation with pyautogui

What is GUI Automation?

GUI automation allows Python to control your keyboard and mouse—useful for automating manual tasks like form filling, button clicking, etc.

◆ Install pyautogui

```
pip install pyautogui
```

◆ Basic Mouse Control

```
import pyautogui

pyautogui.moveTo(100, 100, duration=1)
pyautogui.click()
pyautogui.rightClick()
pyautogui.doubleClick()
```

◆ Get Mouse Position

```
x, y = pyautogui.position()
print(f"Current mouse position: ({x}, {y})")
```

◆ Typing Automatically

```
pyautogui.write("Hello, this is automated typing!", interval=0.05)
pyautogui.press("enter")
```

◆ Keyboard Shortcuts

```
pyautogui.hotkey("ctrl", "s") # Save command
pyautogui.hotkey("ctrl", "c") # Copy
```

◆ Taking Screenshots

```
screenshot = pyautogui.screenshot()  
screenshot.save("screenshot.png")
```

◆ Locate Button on Screen (Image Matching)

```
button_location = pyautogui.locateOnScreen('submit.png')  
if button_location:  
    pyautogui.click(button_location)
```

✦ Note: Screenshot image must be *exactly* the same for reliable detection.

◆ Fail-safe

If your automation goes out of control, move the mouse to the **top-left corner** to stop the script:

```
pyautogui.FAILSAFE = True
```

☑ Chapter 8: Web Automation with Selenium

◆ What is Selenium?

Selenium allows Python to control a real browser like Chrome or Firefox. It's useful for automating tasks on websites like:

- Logging in
- Filling forms
- Clicking buttons
- Scraping dynamic content

◆ Install Selenium

```
pip install selenium
```

And download the corresponding [ChromeDriver](#) for your browser version.

◆ Basic Script to Open Browser

```
from selenium import webdriver

driver = webdriver.Chrome()
driver.get("https://www.google.com")
```

◆ Searching in Google

```
from selenium import webdriver
from selenium.webdriver.common.by import By

driver = webdriver.Chrome()
driver.get("https://www.google.com")

search_box = driver.find_element(By.NAME, "q")
search_box.send_keys("Python automation")
search_box.submit()
```

◆ Clicking Buttons and Links

```
button = driver.find_element(By.ID, "submit")
button.click()
```

◆ Filling Forms

```
driver.find_element(By.NAME, "username").send_keys("admin")
driver.find_element(By.NAME, "password").send_keys("1234")
driver.find_element(By.NAME, "login").click()
```

◆ Wait for Elements to Load

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

wait = WebDriverWait(driver, 10)
element = wait.until(EC.presence_of_element_located((By.ID,
"element_id")))
```

◆ Scraping Text

```
element = driver.find_element(By.CLASS_NAME, "product-title")
print(element.text)
```

◆ Close Browser

```
driver.quit()
```

🔧 Mini Project Idea (Selenium):

Auto-login to a website

- Open login page
- Fill username & password
- Submit form
- Capture success/failure message

Perfect! Let's now cover the final two chapters:

☑ Chapter 9: Working with Databases in Python

☑ Chapter 10: Packaging Scripts & Command-Line Tools

☑ Chapter 9: Working with Databases in Python

Python can connect to various databases—SQLite (built-in), MySQL, PostgreSQL, etc. For scripting, **SQLite** is a great choice due to its simplicity and no setup requirements.

◆ SQLite – Built-in Database Engine

```
import sqlite3

# Connect to database (creates if not exists)
conn = sqlite3.connect('mydb.sqlite')

# Create a cursor object to execute SQL
cursor = conn.cursor()
```

◆ Creating a Table

```
cursor.execute('''
    CREATE TABLE IF NOT EXISTS users (
        id INTEGER PRIMARY KEY,
        name TEXT,
        email TEXT
    )
''')
```

◆ Inserting Data

```
cursor.execute("INSERT INTO users (name, email) VALUES (?, ?)",
               ("Alice", "alice@example.com"))
conn.commit()
```

◆ Reading Data

```
cursor.execute("SELECT * FROM users")
rows = cursor.fetchall()
```

```
for row in rows:
    print(row)
```

◆ Updating and Deleting Data

```
cursor.execute("UPDATE users SET name = ? WHERE id = ?", ("Bob", 1))
cursor.execute("DELETE FROM users WHERE id = ?", (1,))
conn.commit()
```

◆ Closing Connection

```
conn.close()
```

◆ Mini Script: Create and Query Users

```
import sqlite3

conn = sqlite3.connect('data.sqlite')
cur = conn.cursor()

cur.execute("CREATE TABLE IF NOT EXISTS users (id INTEGER PRIMARY KEY,
name TEXT)")
cur.execute("INSERT INTO users (name) VALUES (?)", ("Suriya",))
conn.commit()

cur.execute("SELECT * FROM users")
print(cur.fetchall())

conn.close()
```

☑ Chapter 10: Packaging Python Scripts & CLI Tools

◇ Making a Python Script Executable

For Linux/Mac:

1. Save file as `myscript.py`
2. Add shebang line at top:

```
#!/usr/bin/env python3
```

3. Make it executable:

```
chmod +x myscript.py  
./myscript.py
```

For Windows:

Just run with `python myscript.py`

◇ Creating Command-Line Interfaces (CLI)

Use `argparse` to handle arguments from terminal.

```
import argparse  
  
parser = argparse.ArgumentParser(description="Simple Calculator")  
parser.add_argument("--a", type=int, required=True)  
parser.add_argument("--b", type=int, required=True)  
args = parser.parse_args()  
  
print("Sum:", args.a + args.b)
```


Run from terminal:

```
python calc.py --a 3 --b 7
```

◇ Converting Python Script to Executable (Optional)

Install:

```
pip install pyinstaller
```

Create executable:

```
pyinstaller --onefile myscript.py
```

Output will be in dist/ folder.

◇ Packaging as Module (Basic)

1. Create a folder structure:

```
my_script/  
├── my_module/  
│   ├── __init__.py  
│   └── core.py  
└── setup.py
```

2. setup.py:

```
from setuptools import setup
```

```
setup(  
    name="my_script",  
    version="1.0",  
    packages=["my_module"],
```

```
entry_points={
    "console_scripts": [
        "myscript = my_module.core:main"
    ]
}
```

3. Install locally:

```
pip install .
```

Now run `myscript` directly from terminal 🎉

☑ Summary of What You Learned

Chapter	Topic
1	Scripting Basics
2	Python Refresher
3	File Handling
4	Task Automation
5	Web Scraping
6	JSON, CSV, APIs
7	GUI Automation
8	Selenium Web Automation
9	Databases (SQLite)
10	Packaging & Command-Line Tools