PYTHON AUTOMATION

Beginner to Advance Guide

index

INTRODUCTION

▯ What is Automation?

Let’s start with a simple idea — **what if your computer could do boring tasks for you, automatically?**

Automation means writing code that performs repetitive tasks for you.
Instead of doing the same thing again and again — like sending a message, renaming files, or checking your email — you let **Python do it for you.**

Think of it like building your own robot assistant.

▯ Real-Life Examples of Automation

Here are things you can automate using Python:

✅ Sending WhatsApp messages to friends at a specific time
✅ Downloading Instagram posts or stories automatically
✅ Cleaning up your messy "Downloads" folder
✅ Sending personalized emails to 100+ people in one click
✅ Merging PDFs into one single document
✅ Getting a Telegram bot to reply to messages automatically

🔥 Why Python for Automation?

There are many programming languages, but Python is the BEST for automation. Why?

✔ Simple and beginner-friendly syntax
✔ Huge library support (you’ll learn them soon)
✔ Cross-platform (Windows, Mac, Linux — all work)
✔ Huge community (You’ll never be stuck!)

Python’s power isn’t just in writing automation scripts — it’s in making your life easier.

💡 Example: A Life Without Python Automation

Imagine this: You need to send 30 birthday wishes on WhatsApp every month.

You’d have to:

- Open WhatsApp

- Search each contact
- Type the message
- Send it manually

OR… you could write a Python script that does this **automatically at 9 AM**, while you're sleeping. 😴
That's the power of automation.

🔧 What Will You Learn in This Module?

In this first module, we'll walk step-by-step through:

1. How to install and set up Python + Code Editor
2. How to install powerful libraries for automation
3. How to write your very first automation script
4. How to handle automation safely (delays, errors, etc.)

Setting Up Your Python Automation Environment

✅ Step 1: Installing Python

Python is the programming language we'll use to write automation scripts.
To install it:

1. Go to the official website:
   https://www.python.org/downloads
2. Download the latest version (usually at the top of the page).
3. Run the installer. On the first screen, **very important**:
   - ✅ Check the box that says **"Add Python to PATH"**
   - Then click **"Install Now"**

⚠️ If you don't check that box, Python won't work from the terminal.

Once installed, open your command prompt (Windows) or terminal (Mac/Linux) and type:

```
python --version
```

If you see something like:

```
Python 3.12.2
```

You’re good to go!

✅ Step 2: Installing VS Code (Code Editor)

We need a clean, fast code editor to write Python scripts.

1. Download Visual Studio Code from:
   https://code.visualstudio.com/
2. Install it using the default options.
3. Open it, then go to the **Extensions panel** (left sidebar icon that looks like blocks), and search for:

```
Python
```

4. Install the official Python extension (made by Microsoft).

This extension will help you run scripts, get smart suggestions, and fix errors easily.

✅ Step 3: Creating Your First Project Folder

Let’s organize your automation work.

1. On your Desktop (or any location), create a new folder named:

```
Python Automation Mastery
```

2. Open VS Code.
3. Click on **File** → **Open Folder** and select the folder you just created.

Now all your scripts and files will live inside this one folder. Neat and professional.

✅ Step 4: Creating a Virtual Environment (Very Important)

A virtual environment is like a clean space just for your project — it avoids conflicts with other Python tools on your system.

Inside VS Code terminal (press Ctrl + ~ to open), type:

```
python -m venv venv
```

This creates a folder named venv with an isolated environment.

To activate it:

- **On Windows:**

```
.\venv\Scripts\activate
```

- **On Mac/Linux:**

```
source venv/bin/activate
```

You’ll see your terminal change to:

```
(venv) YourFolderName$
```

Now you’re working inside your project’s own environment. 🔒

✅ Step 5: Installing Required Libraries

Let’s install all the important libraries we’ll use in later chapters.

In the terminal, type:

```
pip install pyautogui pywhatkit instabot python-telegram-bot
pypdf2 fpdf smtplib schedule
```

Each of these tools has a specific use:

- pyautogui → for controlling mouse/keyboard
- pywhatkit → for sending WhatsApp messages
- instabot → to automate Instagram
- python-telegram-bot → to create Telegram bots
- pypdf2 and fpdf → for PDF reading/writing
- smtplib → for email automation
- schedule → for setting daily/weekly tasks

▯ How to Check if a Library Installed Successfully

Just type:

```
pip list
```

This will show all installed packages. If you see the names listed above, you’re ready!

Your First Python Automation Script

🐭: Controlling Mouse and Keyboard

▯ What is PyAutoGUI?

PyAutoGUI is one of the coolest Python libraries.
It lets you control your **mouse, keyboard, and screen** just like a human.

You can:

- Move the mouse automatically
- Click or double-click on screen
- Type messages
- Press special keys (Enter, Ctrl, etc.)
- Take screenshots

Basically, you can "remote control" your computer using Python.

◈ Installing PyAutoGUI

If you haven't installed it yet, open your terminal and run:

```
pip install pyautogui
```

✅ After installation, you're ready to use it.

◈ Writing Your First Automation Script

Let's write a simple script that **moves the mouse** automatically.

1. In your project folder (Python Automation Mastery), create a new file:

```
mouse_move.py
```

2. Open it, and write this code:

```
import pyautogui
import time

# Wait for 5 seconds before starting
time.sleep(5)
```

```
# Move the mouse to x=100, y=100 over 2 seconds
pyautogui.moveTo(100, 100, duration=2)

# Move mouse relative to its current position
pyautogui.moveRel(200, 0, duration=2)
```

🚀 **How This Works:**

- time.sleep(5) → Gives you 5 seconds to switch to any window you want
- moveTo(x, y, duration) → Moves mouse to specific (x, y) position
- moveRel(xOffset, yOffset, duration) → Moves mouse *relative* to its current position

---

▯ Testing It Out

1. Run the script:

```
python mouse_move.py
```

2. Quickly switch to any window. After 5 seconds, you'll see your mouse moving **magically**!

🔥 Automating Clicks

You can also automate clicks:

```
# Click at current mouse location
pyautogui.click()

# Double click
pyautogui.doubleClick()

# Right click
pyautogui.rightClick()
```

You can even **click at a specific position**:

```
# Move and click at (500, 500)
pyautogui.moveTo(500, 500, duration=1)
pyautogui.click()
```

🔥 Automating Keyboard Typing

Typing automatically is super easy:

```
# Type a message automatically
pyautogui.write('Hello! This is automation.', interval=0.1)
```

- interval=0.1 → means 0.1 second delay between each character

You can also **press special keys**:

```
# Press the 'Enter' key
pyautogui.press('enter')
```

Or **hold down** keys:

```
# Hold down the 'Ctrl' key
pyautogui.keyDown('ctrl')

# Press 'c' to copy
pyautogui.press('c')

# Release 'Ctrl'
pyautogui.keyUp('ctrl')
```

⚡ Pro Tip: Finding Mouse Coordinates

You need exact (x, y) positions to automate perfectly.
PyAutoGUI has a cool way to find your mouse position:

Create a new Python file called:

```
position_finder.py
```

Add this code:

```
import pyautogui
import time

time.sleep(5)
print(pyautogui.position())
```

✅ Run the script → Move your mouse anywhere → After 5 seconds, it will print your mouse coordinates!

📌 Important Safety Tip

When automating mouse/keyboard, sometimes your script might go crazy 🤪
If you ever get stuck, immediately **move your mouse cursor to any corner of the screen**.
This will **stop** PyAutoGUI automatically.

🛠️ **Small Practice Task**

Create a script that:

- Moves the mouse to open Notepad
- Types a simple message
- Saves the file automatically

✅ Now you’ve learned how to **control your mouse and keyboard like a boss** with Python!
In the next chapter, we’ll dive into **sending WhatsApp messages automatically using PyWhatKit** — pure magic coming up 🚀

WhatsApp Message Automation

WhatsApp Message Automation using Python (PyWhatKit)

🔷 What is PyWhatKit?

PyWhatKit is a powerful library that lets us:

- Send WhatsApp messages automatically
- Perform Google searches
- Convert text to handwriting
- Even play YouTube videos!

But today, we'll focus on the **WhatsApp automation feature**.

🔧 Installing PyWhatKit

If you haven't already, install it using:

bash

CopyEdit

pip install pywhatkit

It may also install some supporting libraries like pyautogui, pymsgbox, etc.

🔐 Important Note Before You Start

To send WhatsApp messages through Python:

- You must have **WhatsApp Web logged in** on your browser (Google Chrome)
- Your system's **default browser must be Chrome**
- Internet connection should be ON

✨ Sending a WhatsApp Message Automatically

Here's your first automation script:

1. Create a new file in your project folder:

```
send_whatsapp.py
```

2. Paste this code:

```
import pywhatkit as kit

# Send a WhatsApp message
kit.sendwhatmsg("+911234567890", "Hello! This is an automated
message from Python 😎", 15, 30)
```

This line means:

```
sendwhatmsg(phone_no, message, hour, minute)
```

So it will send the message at **15:30 (3:30 PM)** to the given phone number.

Make sure the number is in **international format** with +91 for India.

⏱️ What Happens When You Run It?

1. The browser will automatically open WhatsApp Web
2. It will wait until 15:30
3. Then paste and send your message — hands-free!

If you want to test quickly, schedule the message for 1–2 minutes in the future. Like:

```
kit.sendwhatmsg("+911234567890", "Test message", 14, 45)
```

🕒 Pro tip: Use datetime module to automate this even more.

⌛ What If You Want Instant Sending?

By default, sendwhatmsg() waits for the scheduled time. But for **instant messages**, use:

```
kit.sendwhatmsg_instantly("+911234567890", "Instant message",
wait_time=10, tab_close=True)
```

- wait_time: seconds to wait before sending
- tab_close=True: closes the browser tab after sending

⚠️ Use this carefully, as WhatsApp may block you if spammed too much.

🛠️ Mini Practice Task

Try this:

Create a script that asks user for:

- Phone number
- Message
- Time to send
  Then use sendwhatmsg() to send it.

😱 Can I Send to Multiple People?

No direct "bulk messaging" is supported for privacy reasons. But you can loop through numbers one by one (we'll cover this in a later chapter 😉)

📌 Important Tips

- Always keep your **browser and WhatsApp Web logged in**
- Don't use this to spam — WhatsApp may detect it
- Use for reminders, updates, birthday wishes, etc.

✅ That's it — you just **automated WhatsApp using Python**.

Automating Instagram

📸 Introduction

Instagram is one of the most popular platforms today.
What if you could **automate** things like:

- Auto login to Instagram
- Auto follow users
- Auto unfollow users
- Auto like posts
- Even send automated messages?

With Python, it's possible — and surprisingly easy!

In this chapter, we'll automate Instagram step-by-step using an amazing library called **Instabot**.

🧩 What is Instabot?

Instabot is a Python library that can:

- Automatically login to your Instagram account
- Follow/unfollow users
- Like posts
- Comment on posts
- Send direct messages

Important: Always use automation **carefully** on Instagram.
Spamming may lead to **account restrictions or bans**.

We'll use this bot **safely** for basic tasks.

🔧 Installing Instabot

First, install the Instabot library.

In your terminal, type:

```
pip install instabot
```

✅ After installation, you are ready to create your bot!

🔐 Important Preparation

Before you automate Instagram:

- Make sure your Instagram account is **older than 1–2 months**.
- Avoid using **new accounts** — Instagram monitors them more strictly.
- Do not spam with automation — start with **small limits**.

🚀 Logging into Instagram Automatically

Let’s create your first InstaBot script:

1. Create a new file:

```
insta_login.py
```

2. Paste this code:

```
from instabot import Bot

bot = Bot()

# Login
bot.login(username="your_username", password="your_password")
```

Replace "your_username" and "your_password" with your real Instagram credentials.

✅ When you run it, the bot will automatically login to your Instagram account.

🔥 Automating Follow and Unfollow

After login, you can follow/unfollow people easily.

➡ Auto Follow:

```
bot.follow("friend_username")
```

This will follow the user with the given username.

➡ Auto Unfollow:

```
bot.unfollow("friend_username")
```

This will unfollow the user.

🔥 Automating Liking Posts

You can even like photos of a user automatically!

Example:

```
bot.like_user("friend_username", amount=3)
```

This will like 3 posts of the given user.

You can also like posts by hashtags:

```
bot.like_hashtag("travel", amount=5)
```

✅ This likes 5 posts with the hashtag #travel.

🔥 Sending Direct Messages (DMs)

You can send messages automatically:

```
bot.send_message("Hello from Python!", ["friend_username"])
```

You can send the same message to multiple users too by giving a list:

```
bot.send_message("Good Morning!", ["friend1", "friend2",
"friend3"])
```

⚙️ Some More Useful Commands

- **Upload a photo** automatically:

```
bot.upload_photo("path_to_photo.jpg", caption="My new post via
Python 🚀")
```

- **Get your own followers:**

```
followers = bot.get_user_followers("your_username")
print(followers)
```

- **Get your following list:**

```
following = bot.get_user_following("your_username")
print(following)
```

✅ You can then automate follow-backs, unfollows, etc.

⚡ Pro Tips for Safe Automation

- **Limit actions:**
  Example: 20–30 likes, 10–15 follows/unfollows per hour
- **Randomize actions:**
  Do not perform 100 actions at once — take natural breaks using time.sleep()
- **Avoid bots on brand-new accounts:**
  Instagram easily flags new accounts.
- **Keep two-factor authentication OFF** when using Instabot (or it might cause login problems).

🛠 **Small Practice Task**

Create a Python script that:

- Logs into Instagram
- Follows 5 users (you choose)
- Likes 3 photos of each user
- Sends a “Hello!” DM to them

(This will make you **comfortable** with automation.)

✅ That’s it — now you have learned **how to automate Instagram** using Python easily!
In the next chapter, we will **build a real Telegram Bot** from scratch — and trust me, it's going to be super fun 🚀

Telegram Bot Creation

📱 Why Telegram Bots?

Telegram is not just a messaging app — it supports **powerful bots** that can:

- Reply to user messages
- Send custom replies, images, documents
- Work as mini-apps
- Handle tasks like reminders, APIs, notifications, and more

And the best part?

👉 You don't need to "hack" anything. Telegram **officially supports bots** via its **Bot API**.

🧩 What You'll Learn

In this chapter, you will learn to:

- Create your own Telegram Bot
- Set up your bot token
- Read user messages
- Reply to users
- Send files, images, and buttons

Let's go step by step 👇

🔧 Step 1: Create Your Telegram Bot

1. Open Telegram and search: **BotFather**
2. Start a chat with BotFather and type:

```
/newbot
```

3. It will ask:
   - Bot name → e.g. PythonAutomationBot
   - Bot username → must end with bot (e.g. python_auto_bot)

✅ Once done, it will give you a long **API Token** like this:

```
1234567890:AAHxK_jJfdfjHfiufHHdju34FjsdfKD
```

Copy this token. This is your bot's **password** to control it using Python.

🔧 Step 2: Install Required Library

We'll use a library called python-telegram-bot.

Install it using:

```
pip install python-telegram-bot --upgrade
```

This will help us connect to Telegram's API and handle bot messages easily.

🤖 Step 3: Create Your First Bot Script

Create a file my_bot.py and paste this code:

```
from telegram import Update
from telegram.ext import ApplicationBuilder, CommandHandler,
ContextTypes

async def start(update: Update, context:
ContextTypes.DEFAULT_TYPE):
    await update.message.reply_text("Hello! I am your Python
Automation Bot 🤖")

app = ApplicationBuilder().token("YOUR_BOT_TOKEN").build()

app.add_handler(CommandHandler("start", start))

app.run_polling()
```

📌 Replace "YOUR_BOT_TOKEN" with your actual token.

Now, run the file:

```
python my_bot.py
```

✅ Your bot is now live and listening!

Open your bot on Telegram (using its username) and type:

```
/start
```

You'll see the reply:

```
Hello! I am your Python Automation Bot 🤖
```

Boom! Bot created successfully 🚀

How It Works

- run_polling() keeps checking for new messages
- CommandHandler("start", start) listens for the /start command
- reply_text() sends message back to the user

We'll now build more features step by step.

💬 Replying to User Messages

Let's make the bot reply with a custom message when the user types anything.

Update your my_bot.py like this:

```
from telegram.ext import MessageHandler, filters

# New handler for text messages
async def reply_to_user(update: Update, context:
ContextTypes.DEFAULT_TYPE):
    user_message = update.message.text
    await update.message.reply_text(f"You said:
{user_message}")

# Add this handler below your existing code
app.add_handler(MessageHandler(filters.TEXT, reply_to_user))
```

✅ Now your bot will echo back whatever the user types.

Example:

```
User: Hello bot
Bot: You said: Hello bot
```

🖼 Sending Images

Let's make the bot send an image when someone types /photo.

1. Save any image in the same folder (example: cat.jpg)
2. Add this command:

```
from telegram import InputFile

async def send_photo(update: Update, context:
ContextTypes.DEFAULT_TYPE):
    photo = InputFile("cat.jpg")
    await update.message.reply_photo(photo, caption="Here’s a
cute cat 🐱")

app.add_handler(CommandHandler("photo", send_photo))
```

Now, type /photo to see your bot send an image with a caption.

📄 Sending Documents

Want to send a PDF, doc, or ZIP file? It’s super simple.

Add this command:

```
async def send_file(update: Update, context:
ContextTypes.DEFAULT_TYPE):
    doc = InputFile("my_file.pdf")  # Replace with your file
    await update.message.reply_document(doc, caption="Here’s
your file!")

app.add_handler(CommandHandler("file", send_file))
```

Now, send a PDF or ZIP by typing /file.

▯ Adding Buttons (Inline Keyboard)

Let’s make a menu with buttons. Example: Yes / No options.

Add this:

```
from telegram import InlineKeyboardButton,
InlineKeyboardMarkup

async def show_buttons(update: Update, context:
ContextTypes.DEFAULT_TYPE):
    keyboard = [
        [InlineKeyboardButton("Yes 👍", callback_data="yes"),
         InlineKeyboardButton("No 👎", callback_data="no")]
    ]
    reply_markup = InlineKeyboardMarkup(keyboard)
    await update.message.reply_text("Do you like this bot?",
reply_markup=reply_markup)

app.add_handler(CommandHandler("buttons", show_buttons))
```

Then add this callback handler:

```
from telegram.ext import CallbackQueryHandler
```

```
async def button_handler(update: Update, context:
ContextTypes.DEFAULT_TYPE):
    query = update.callback_query
    await query.answer()

    if query.data == "yes":
        await query.edit_message_text("Awesome! Thanks 🙆")
    else:
        await query.edit_message_text("No worries! I’ll
improve 😇")

app.add_handler(CallbackQueryHandler(button_handler))
```

✅ Now when you type /buttons, the bot shows two options, and replies based on your click.

🛠 **Practice Task**

Create a bot that does all this:

- Replies to user messages
- Sends a welcome message on /start
- Sends an image on /photo
- Sends a PDF on /file
- Shows buttons on /buttons

Try adding your own logic, like sending jokes or links!

✅ That’s a **fully working Telegram bot** — and this is just the beginning!
You can now use this bot to build **tools, reminders, chat assistants**, or even connect it to your own website/app.

Email Automation with Python

▯ Why Automate Emails?

Email automation is useful for:

- Sending reports or updates automatically
- Notifying users or clients
- Sending attachments (PDFs, invoices, certificates)
- Building email bots or campaigns

With Python, we can send emails in seconds — no manual typing, no delays!

🛠 What You'll Learn

- How to send emails using Python
- How to attach files (PDFs, images, ZIPs)
- How to send to multiple recipients
- How to connect with Gmail securely

Let's dive in!

🔧 Step 1: Enable Gmail Access (IMPORTANT)

If you're using Gmail, you need to allow **less secure apps** or use an **app password**.

Option A: If 2-Step Verification is OFF

1. Visit: https://myaccount.google.com/lesssecureapps
2. Turn ON access

Option B: If 2-Step Verification is ON (recommended)

1. Visit: https://myaccount.google.com/apppasswords
2. Generate a new App Password for "Mail"
3. Copy the 16-digit password — you'll use this instead of your normal password

---

▯ Step 2: Send Your First Email

Install the required library (for attachments):

```
pip install secure-smtplib
```

Now, create a file send_email.py:

```
import smtplib
from email.mime.text import MIMEText
from email.mime.multipart import MIMEMultipart

# Email setup
sender_email = "youremail@gmail.com"
receiver_email = "receiver@gmail.com"
password = "your_app_password"

# Create the email
message = MIMEMultipart("alternative")
message["Subject"] = "Hello from Python!"
message["From"] = sender_email
message["To"] = receiver_email

text = "Hi there! This email was sent using Python 😎"
part = MIMEText(text, "plain")

message.attach(part)

# Send the email
with smtplib.SMTP_SSL("smtp.gmail.com", 465) as server:
    server.login(sender_email, password)
    server.sendmail(sender_email, receiver_email,
message.as_string())

print("✅ Email sent successfully!")
```

📌 Replace youremail, receiver, and password with your own values.

📎 Sending Attachments (PDF, ZIP, etc.)

Let's send a PDF file with the email:

```
from email.mime.application import MIMEApplication

filename = "sample.pdf"  # Your file name

with open(filename, "rb") as f:
    attachment = MIMEApplication(f.read(), _subtype="pdf")
    attachment.add_header("Content-Disposition", "attachment",
filename=filename)
    message.attach(attachment)
```

💡 You can attach multiple files using the same method.

🧑‍🤝‍🧑 Sending to Multiple Recipients

Want to send one email to multiple people?

```
receiver_emails = ["person1@gmail.com", "person2@gmail.com",
"person3@gmail.com"]

for email in receiver_emails:
    message["To"] = email
    server.sendmail(sender_email, email, message.as_string())
```

✅ Great for reports, notifications, or email campaigns.

✨ Make it Dynamic (Using Input or CSV)

You can automate this further by:

- Reading emails from a .csv file
- Customizing the name/message for each person
- Creating loops to send personalized emails

Sending Beautiful HTML Emails

Plain text is fine, but HTML emails look better.

Let’s send a styled email with bold text, links, and colors:

```
html = """
<html>
  <body>
    <h2 style='color: teal;'>Hello from Python 🚀</h2>
    <p>This is a <b>test email</b> sent with <i>HTML
formatting</i>.</p>
    <p>Check out our <a
href='https://yourwebsite.com'>Website</a></p>
  </body>
</html>
"""

part2 = MIMEText(html, "html")
```

```
message.attach(part2)
```

✅ Now your email will have both plain text and styled HTML.

📊 Read Email Contacts from a CSV File

Instead of typing emails one by one, save them in a .csv file like this:

contacts.csv

```
name,email
Raj,raj@example.com
Aisha,aisha@gmail.com
John,john@yahoo.com
```

Now read them using Python:

```
import csv

with open("contacts.csv", "r") as file:
    reader = csv.DictReader(file)
    contacts = [row for row in reader]

for contact in contacts:
    receiver = contact["email"]
    name = contact["name"]

    # Customize email
    text = f"Hi {name}, this is a personalized email from 
Python!"
    message = MIMEMultipart("alternative")
    message["Subject"] = "Personalized Email"
    message["From"] = sender_email
    message["To"] = receiver
    message.attach(MIMEText(text, "plain"))

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

✅ Now each contact gets their own custom message.

🔁 Build a Mini Campaign System (Optional)

You can combine all this into a mini system that:

- Sends bulk emails from a contact list
- Attaches PDFs automatically
- Logs successful sends

You can even schedule it using **Python's schedule** or **Windows Task Scheduler** for daily/weekly reports.

🧠 Practice Task

Build a Python script that:

- Sends an HTML email with one attachment
- Reads contacts from a CSV file
- Greets each user by name
- Logs which emails were sent successfully

🔥 You've now mastered **Email Automation** with Python!
From simple sends to dynamic campaigns — you can build **tools, alerts, reminders, or marketing systems**.

PDF and Document Automation

Why Automate PDFs?

Automating PDFs is useful for:

- **Extracting data** from large PDFs (e.g., invoices, reports)
- **Merging** multiple files into one
- **Modifying** documents (e.g., adding watermarks, editing text)
- **Generating** PDFs dynamically (e.g., receipts, certificates)

Python makes all of this super easy with libraries like **PyPDF2** and **ReportLab**.

Step 1: Install PyPDF2

We'll use **PyPDF2** for basic PDF tasks. Install it first:

```
pip install pypdf2
```

Extract Text from a PDF

Let's start with extracting text from an existing PDF.

Create a script extract_text.py:

```
import PyPDF2

# Open the PDF file
with open("document.pdf", "rb") as file:
    reader = PyPDF2.PdfReader(file)
    text = ""

    # Extract text from all pages
    for page in reader.pages:
        text += page.extract_text()

print(text)
```

What it does: It reads a PDF file, extracts text, and prints it to the console.

Merge Multiple PDFs

You can merge multiple PDFs into one using PyPDF2. Let's say you have two PDFs: file1.pdf and file2.pdf.

```
import PyPDF2

# Create a PdfWriter object
pdf_writer = PyPDF2.PdfWriter()

# Read the first PDF
with open("file1.pdf", "rb") as file1:
    reader = PyPDF2.PdfReader(file1)
    pdf_writer.add_page(reader.pages[0])  # Add first page of
file1

# Read the second PDF
with open("file2.pdf", "rb") as file2:
    reader = PyPDF2.PdfReader(file2)
    pdf_writer.add_page(reader.pages[0])  # Add first page of
file2

# Write the merged PDF to a new file
with open("merged_output.pdf", "wb") as output_file:
    pdf_writer.write(output_file)

print("✅ PDFs merged successfully!")
```

Now you'll have a new PDF called merged_output.pdf with the first page from each of the original PDFs.

✂️ Split a PDF into Multiple Pages

If you want to split a PDF into individual pages, here's how:

```
import PyPDF2

# Open the source PDF
with open("document.pdf", "rb") as file:
    reader = PyPDF2.PdfReader(file)
    pdf_writer = PyPDF2.PdfWriter()

    # Extract pages and save them as separate files
    for i in range(len(reader.pages)):
        pdf_writer.add_page(reader.pages[i])
        with open(f"page_{i + 1}.pdf", "wb") as output_file:
            pdf_writer.write(output_file)

    print("✅ PDF split into individual pages!")
```

Now each page of the original PDF will be saved as a separate file (e.g., page_1.pdf, page_2.pdf).

💧 Add a Watermark to a PDF

Let’s add a watermark to a PDF page. First, you need a watermark PDF (e.g., watermark.pdf).

```
import PyPDF2

# Open the original PDF and the watermark PDF
with open("document.pdf", "rb") as original_file,
open("watermark.pdf", "rb") as watermark_file:
    reader = PyPDF2.PdfReader(original_file)
    watermark = PyPDF2.PdfReader(watermark_file)

    # Apply watermark to each page
    pdf_writer = PyPDF2.PdfWriter()
    for page in reader.pages:
        page.merge_page(watermark.pages[0])  # Merge watermark
        pdf_writer.add_page(page)

    # Save the watermarked PDF
    with open("watermarked_document.pdf", "wb") as
output_file:
        pdf_writer.write(output_file)

print("✅ Watermark added to PDF!")
```

This will overlay the watermark on every page of the document.

📝 Generate a PDF from Scratch

Finally, let’s create a brand new PDF. We’ll use **ReportLab** for this.

First, install ReportLab:

```
pip install reportlab
```

Now, create a simple PDF with a title and some text:

```
from reportlab.lib.pagesizes import letter
from reportlab.pdfgen import canvas

# Create a new PDF
```

```
c = canvas.Canvas("generated_document.pdf", pagesize=letter)

# Add some text
c.setFont("Helvetica", 12)
c.drawString(100, 750, "Hello, this is a generated PDF with
Python!")

# Save the PDF
c.save()

print("✅ PDF generated!")
```

This will create a new PDF called generated_document.pdf with some basic text.

▯ Practice Task

Create a Python script that:

1. Extracts text from a PDF
2. Merges it with another PDF
3. Adds a watermark
4. Generates a new PDF with custom content

Now you know how to manipulate PDFs in Python! You can automate everything from text extraction to PDF generation and modification.