Team Alpha

PDF PROCESSOR REPORT

Prepared By
Mansoor Bukhari

Presented To
Sir Muhammad Rizwan

1. Project Overview

Objective: The objective of this project is to create a Python-based application that facilitates the conversion of Jupyter Notebooks to PDF files and processes existing PDF files by adding borders and page numbers. The application uses web automation for conversion and PDF manipulation libraries for processing.

Features:

- Conversion of Jupyter Notebooks (`.ipynb` files) to PDF format.
- Processing of existing PDF files to add borders and page numbers.
- Customizable border colors for processed PDFs.
- User-friendly interface with console-based interaction.

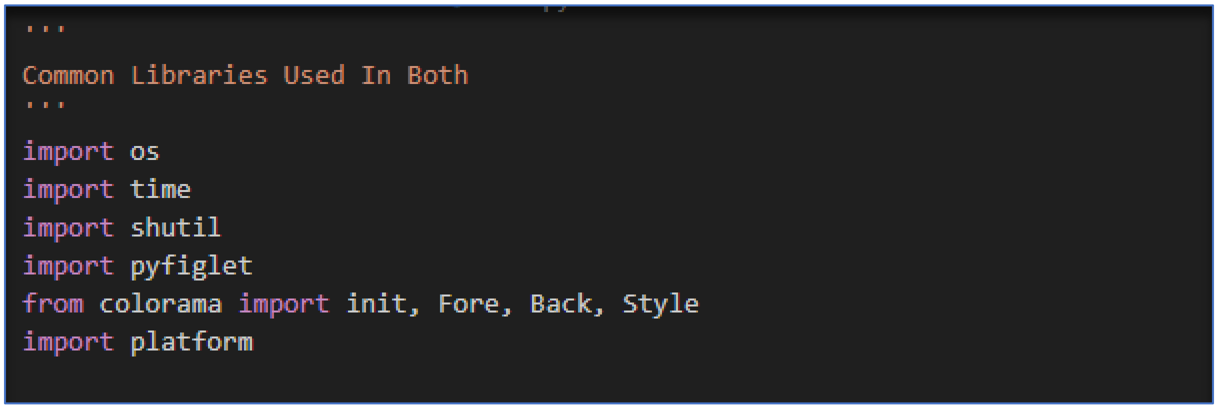
Tools and Libraries Used:

- Web Automation: Selenium
- PDF Processing: PyPDF2, ReportLab
- Banner and Console Output: pyfiglet, colorama
- File and Path Operations: os, shutil
- Web Browser Interaction: Chrome WebDriver
- PDF Conversion: Ploomber Convert Service

2. Project Components

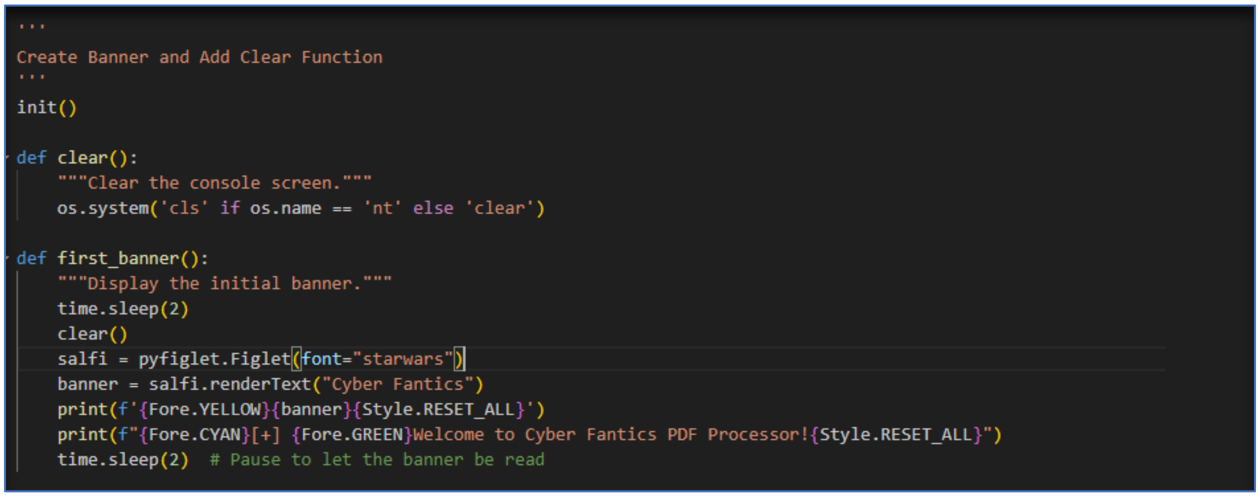
2.1. Banner Creation and Console Clearing

The project starts by creating a banner and providing functionality to clear the console screen.



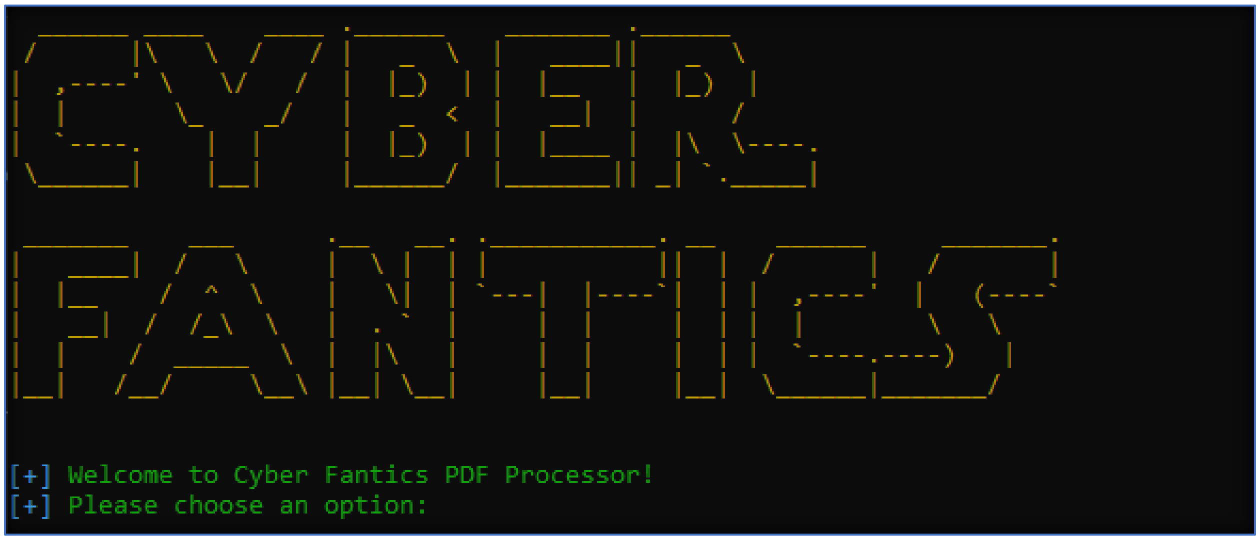
```
'''
Common Libraries Used In Both
'''
import os
import time
import shutil
import pyfiglet
from colorama import init, Fore, Back, Style
import platform
```

Figure 1 - Import Libraries for Creating Banner and Clearing Screen



```
'''
Create Banner and Add Clear Function
'''
init()

def clear():
    """Clear the console screen."""
    os.system('cls' if os.name == 'nt' else 'clear')

def first_banner():
    """Display the initial banner."""
    time.sleep(2)
    clear()
    salfi = pyfiglet.Figlet(font="starwars")
    banner = salfi.renderText("Cyber Fantics")
    print(f'{Fore.YELLOW}{banner}{Style.RESET_ALL}')
    print(f"{Fore.CYAN}[+] {Fore.GREEN}Welcome to Cyber Fantics PDF Processor!{Style.RESET_ALL}")
    time.sleep(2)  # Pause to let the banner be read
```

Figure 2 - Clear Screen Function and Banner



```
[+] Welcome to Cyber Fantics PDF Processor!
[+] Please choose an option:
```

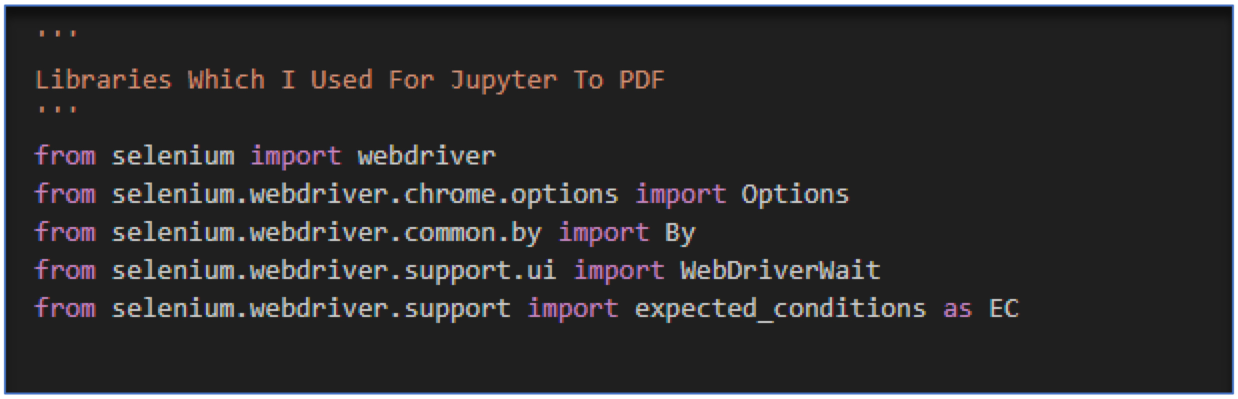
Figure 3 - Banner Output

2.2. Jupyter Notebook to PDF Conversion

The function `download_jupyter` handles the conversion of Jupyter Notebooks to PDFs using a web automation approach with Selenium.

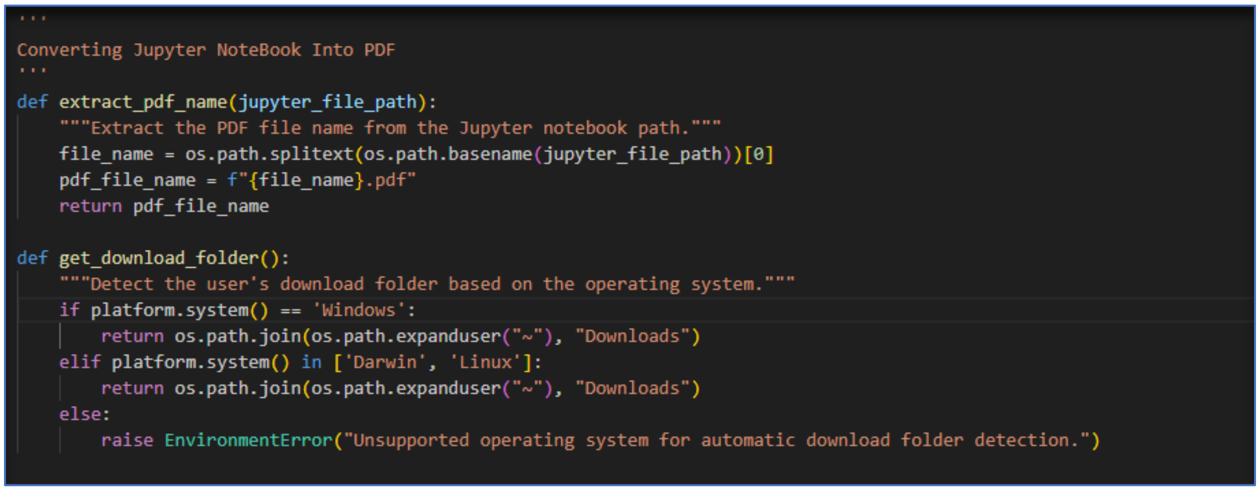
- ❖ **Extracting PDF Name:** Extracts the PDF filename from the Jupyter Notebook path.
- ❖ **Download Folder Detection:** Detects the download folder based on the operating system.
- ❖ **File Upload and Conversion:** Uploads the Jupyter Notebook to the Ploomber Convert Service and waits for the download of the resulting PDF.
- ❖ **Download File:** After conversion jupyter file into PDF format download it into Download Folder.

- **Move Into Main Folder:** Move file into main folder in which script is running, so that further process on pdf can be applied.
- **Libraries Used:** Selenium library is used for automation purpose but other libraries used for banner and clearing screen are also used to detect folder and files.



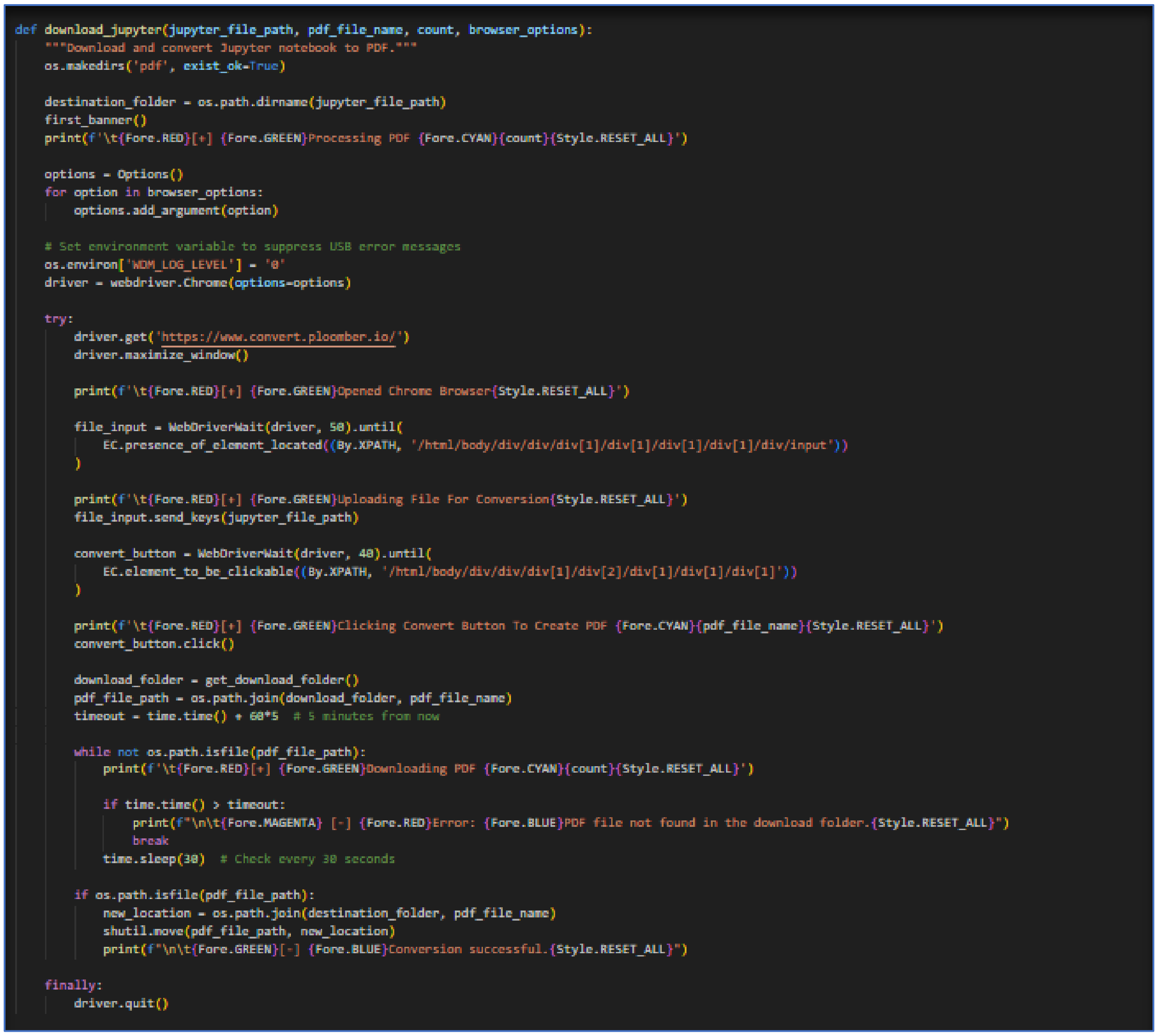
```
'''
Libraries Which I Used For Jupyter To PDF
'''
from selenium import webdriver
from selenium.webdriver.chrome.options import Options
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
```

Figure 4 - Selenium Libraries



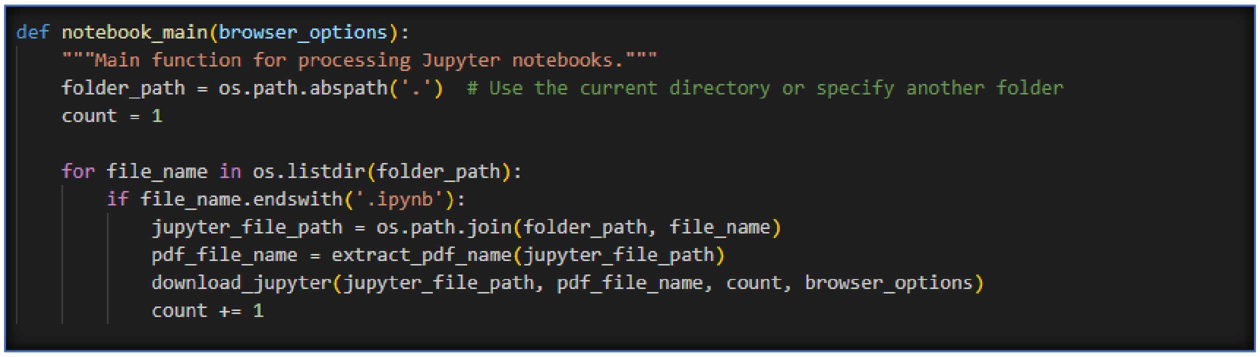
```
'''
Converting Jupyter NoteBook Into PDF
'''
def extract_pdf_name(jupyter_file_path):
    """Extract the PDF file name from the Jupyter notebook path."""
    file_name = os.path.splitext(os.path.basename(jupyter_file_path))[0]
    pdf_file_name = f"{file_name}.pdf"
    return pdf_file_name

def get_download_folder():
    """Detect the user's download folder based on the operating system."""
    if platform.system() == 'Windows':
        return os.path.join(os.path.expanduser("~"), "Downloads")
    elif platform.system() in ['Darwin', 'Linux']:
        return os.path.join(os.path.expanduser("~"), "Downloads")
    else:
        raise EnvironmentError("Unsupported operating system for automatic download folder detection.")
```

Figure 5 - Function for Getting PDF name and Download Folder



```python
def download_jupyter(jupyter_file_path, pdf_file_name, count, browser_options):
    """Download and convert Jupyter notebook to PDF."""
    os.makedirs('pdf', exist_ok=True)

    destination_folder = os.path.dirname(jupyter_file_path)
    first_banner()
    print(f'\t{Fore.RED}[+] {Fore.GREEN}Processing PDF {Fore.CYAN}{count}{Style.RESET_ALL}')

    options = Options()
    for option in browser_options:
        options.add_argument(option)

    # Set environment variable to suppress USB error messages
    os.environ['WDM_LOG_LEVEL'] = '0'
    driver = webdriver.Chrome(options=options)

    try:
        driver.get('https://www.convert.ploomber.io/')
        driver.maximize_window()

        print(f'\t{Fore.RED}[+] {Fore.GREEN}Opened Chrome Browser{Style.RESET_ALL}')

        file_input = WebDriverWait(driver, 50).until(
            EC.presence_of_element_located((By.XPATH, '/html/body/div/div/div[1]/div[1]/div[1]/div[1]/div/input'))
        )

        print(f'\t{Fore.RED}[+] {Fore.GREEN}Uploading File For Conversion{Style.RESET_ALL}')
        file_input.send_keys(jupyter_file_path)

        convert_button = WebDriverWait(driver, 40).until(
            EC.element_to_be_clickable((By.XPATH, '/html/body/div/div/div[1]/div[2]/div[1]/div[1]/div[1]'))
        )

        print(f'\t{Fore.RED}[+] {Fore.GREEN}Clicking Convert Button To Create PDF {Fore.CYAN}{pdf_file_name}{Style.RESET_ALL}')
        convert_button.click()

        download_folder = get_download_folder()
        pdf_file_path = os.path.join(download_folder, pdf_file_name)
        timeout = time.time() + 60*5  # 5 minutes from now

        while not os.path.isfile(pdf_file_path):
            print(f'\t{Fore.RED}[+] {Fore.GREEN}Downloading PDF {Fore.CYAN}{count}{Style.RESET_ALL}')

            if time.time() > timeout:
                print(f"\n\t{Fore.MAGENTA} [-] {Fore.RED}Error: {Fore.BLUE}PDF file not found in the download folder.{Style.RESET_ALL}")
                break
            time.sleep(30)  # Check every 30 seconds

        if os.path.isfile(pdf_file_path):
            new_location = os.path.join(destination_folder, pdf_file_name)
            shutil.move(pdf_file_path, new_location)
            print(f"\n\t{Fore.GREEN}[-] {Fore.BLUE}Conversion successful.{Style.RESET_ALL}")

    finally:
        driver.quit()
```

Figure 6 - Function to convert Jupyter into PDF



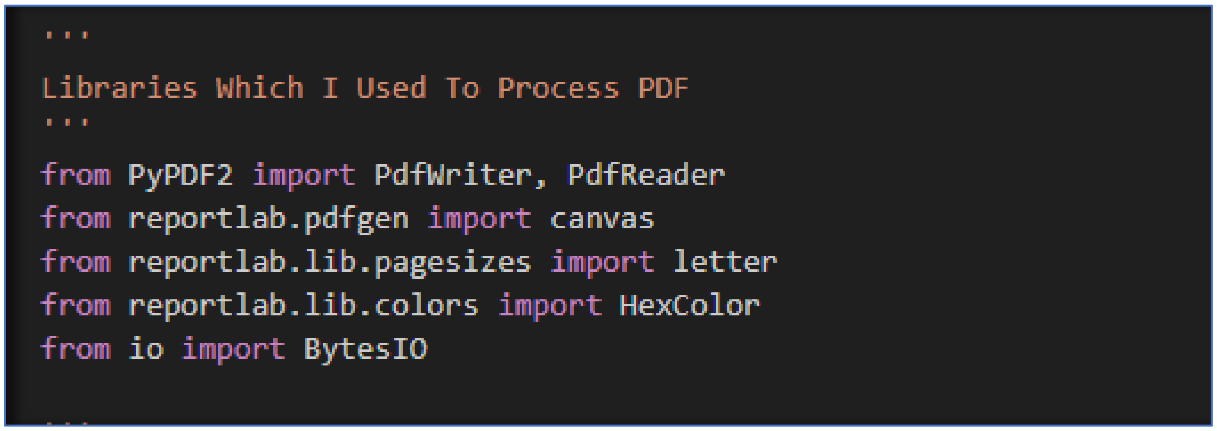
```python
def notebook_main(browser_options):
    """Main function for processing Jupyter notebooks."""
    folder_path = os.path.abspath('.')  # Use the current directory or specify another folder
    count = 1

    for file_name in os.listdir(folder_path):
        if file_name.endswith('.ipynb'):
            jupyter_file_path = os.path.join(folder_path, file_name)
            pdf_file_name = extract_pdf_name(jupyter_file_path)
            download_jupyter(jupyter_file_path, pdf_file_name, count, browser_options)
            count += 1
```

Figure 7 - Notebook main function to detect jupyter notebooks and loop them

2.3. PDF Processing

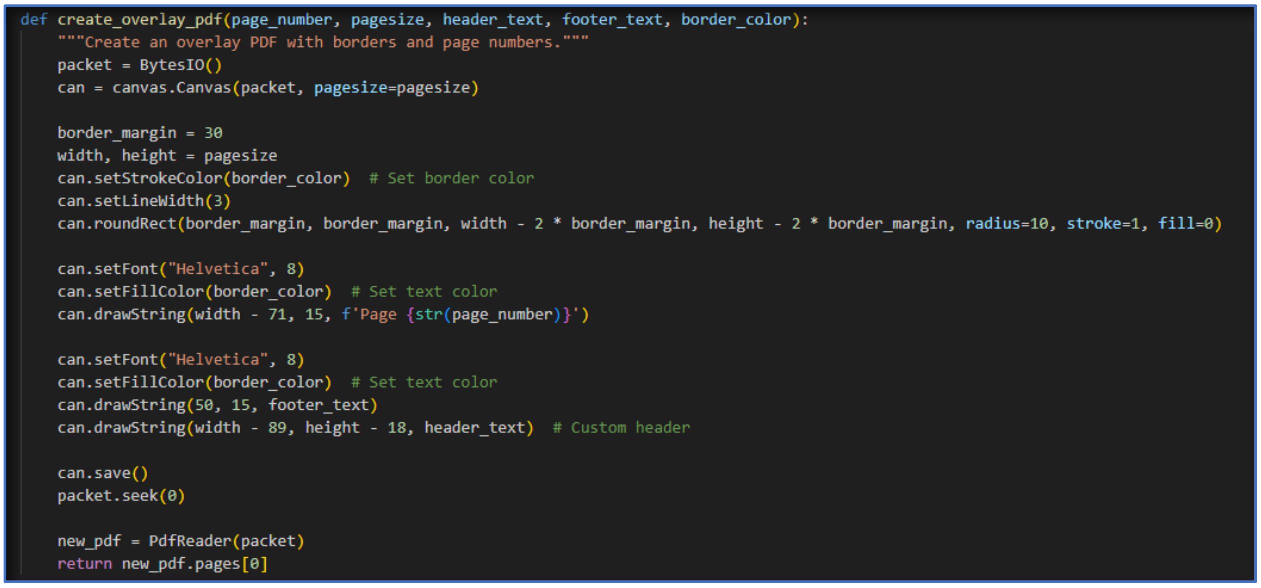
The `process_pdfs` function handles the addition of borders and page numbers to existing PDF files.

- **Overlay Creation:** Generates a PDF overlay with borders, page numbers, and custom text.
- **Merging Overlays:** Merges the overlay with each page of the original PDF.
- **File Cleanup:** Deletes the original PDF after processing.



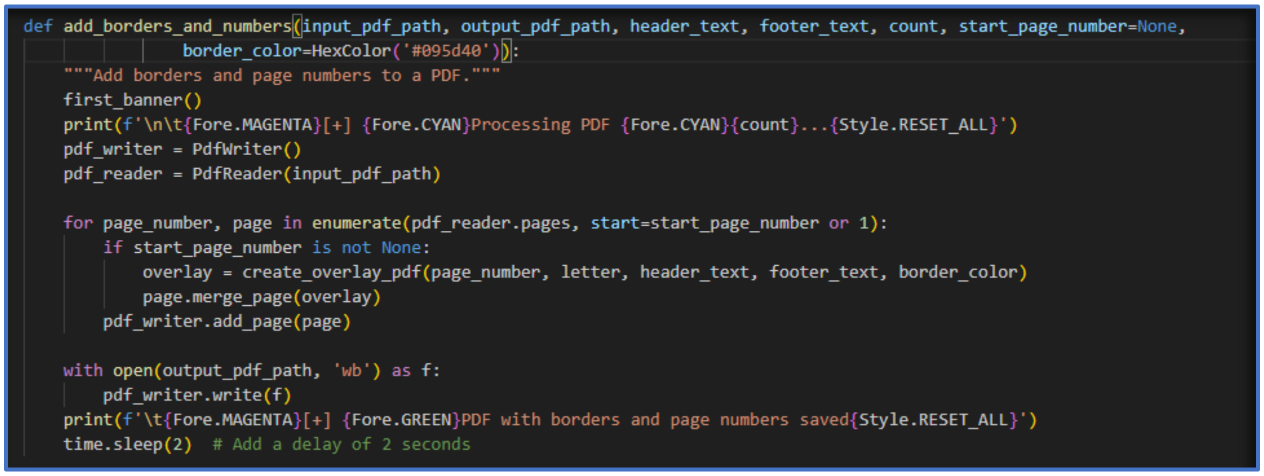
```python
'''
Libraries Which I Used To Process PDF
'''
from PyPDF2 import PdfWriter, PdfReader
from reportlab.pdfgen import canvas
from reportlab.lib.pagesizes import letter
from reportlab.lib.colors import HexColor
from io import BytesIO
```

Figure 8 - Import Necessary Libraries To Process PDF



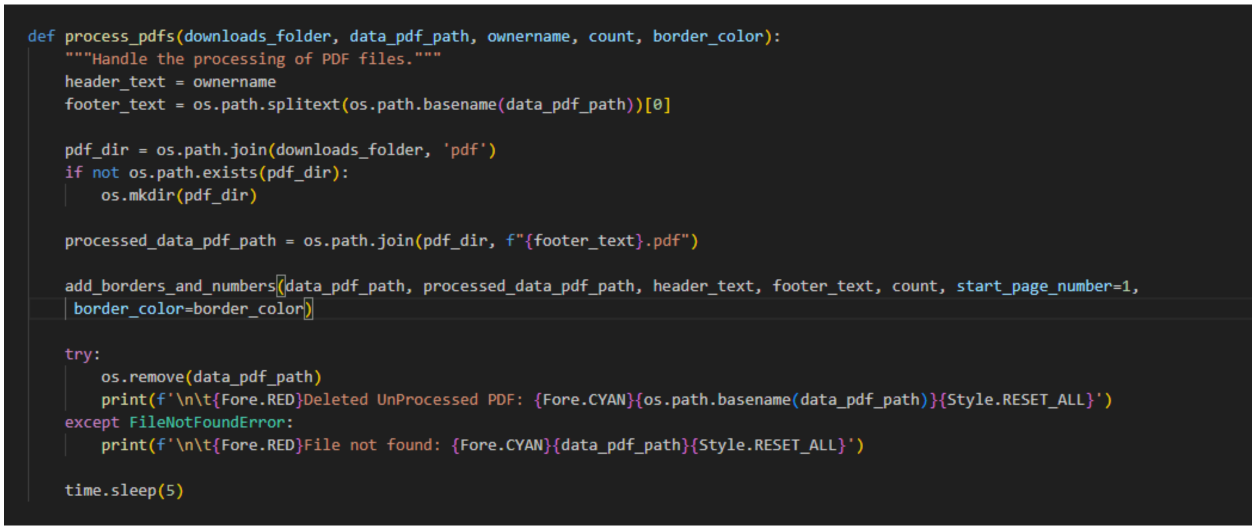
```python
def create_overlay_pdf(page_number, pagesize, header_text, footer_text, border_color):
    """Create an overlay PDF with borders and page numbers."""
    packet = BytesIO()
    can = canvas.Canvas(packet, pagesize=pagesize)

    border_margin = 30
    width, height = pagesize
    can.setStrokeColor(border_color)  # Set border color
    can.setLineWidth(3)
    can.roundRect(border_margin, border_margin, width - 2 * border_margin, height - 2 * border_margin, radius=10, stroke=1, fill=0)

    can.setFont("Helvetica", 8)
    can.setFillColor(border_color)  # Set text color
    can.drawString(width - 71, 15, f'Page {str(page_number)}')

    can.setFont("Helvetica", 8)
    can.setFillColor(border_color)  # Set text color
    can.drawString(50, 15, footer_text)
    can.drawString(width - 89, height - 18, header_text)  # Custom header

    can.save()
    packet.seek(0)

    new_pdf = PdfReader(packet)
    return new_pdf.pages[0]
```

Figure 9 - Function To Create an overlay for pdf.



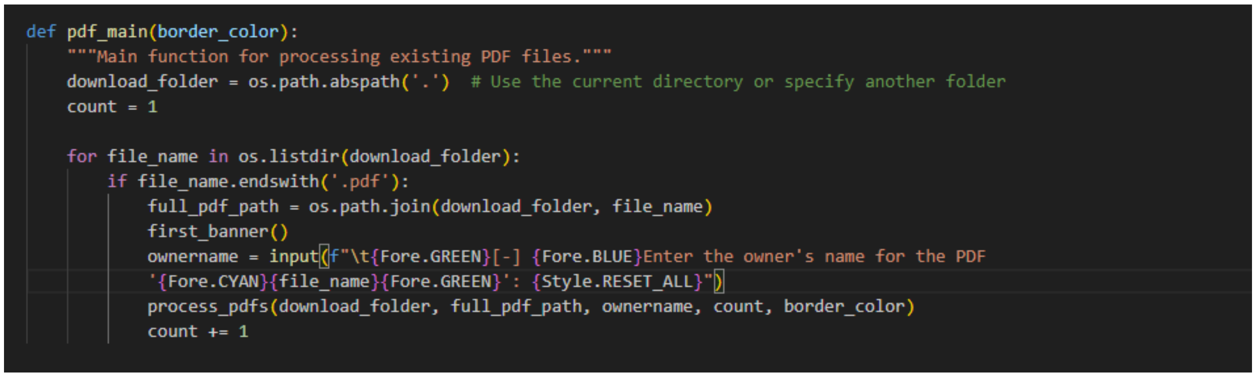
```python
def add_borders_and_numbers(input_pdf_path, output_pdf_path, header_text, footer_text, count, start_page_number=None,
                            border_color=HexColor('#095d40')):
    """Add borders and page numbers to a PDF."""
    first_banner()
    print(f'\n\t{Fore.MAGENTA}[+] {Fore.CYAN}Processing PDF {Fore.CYAN}{count}...{Style.RESET_ALL}')
    pdf_writer = PdfWriter()
    pdf_reader = PdfReader(input_pdf_path)

    for page_number, page in enumerate(pdf_reader.pages, start=start_page_number or 1):
        if start_page_number is not None:
            overlay = create_overlay_pdf(page_number, letter, header_text, footer_text, border_color)
            page.merge_page(overlay)
        pdf_writer.add_page(page)

    with open(output_pdf_path, 'wb') as f:
        pdf_writer.write(f)
    print(f'\t{Fore.MAGENTA}[+] {Fore.GREEN}PDF with borders and page numbers saved{Style.RESET_ALL}')
    time.sleep(2)  # Add a delay of 2 seconds
```

Figure 10 - Add Border and Caption



```python
def process_pdfs(downloads_folder, data_pdf_path, ownername, count, border_color):
    """Handle the processing of PDF files."""
    header_text = ownername
    footer_text = os.path.splitext(os.path.basename(data_pdf_path))[0]

    pdf_dir = os.path.join(downloads_folder, 'pdf')
    if not os.path.exists(pdf_dir):
        os.mkdir(pdf_dir)

    processed_data_pdf_path = os.path.join(pdf_dir, f"{footer_text}.pdf")

    add_borders_and_numbers(data_pdf_path, processed_data_pdf_path, header_text, footer_text, count, start_page_number=1,
    border_color=border_color)

    try:
        os.remove(data_pdf_path)
        print(f'\n\t{Fore.RED}Deleted UnProcessed PDF: {Fore.CYAN}{os.path.basename(data_pdf_path)}{Style.RESET_ALL}')
    except FileNotFoundError:
        print(f'\n\t{Fore.RED}File not found: {Fore.CYAN}{data_pdf_path}{Style.RESET_ALL}')

    time.sleep(5)
```

Figure 11 - Process PDF and Delete Original

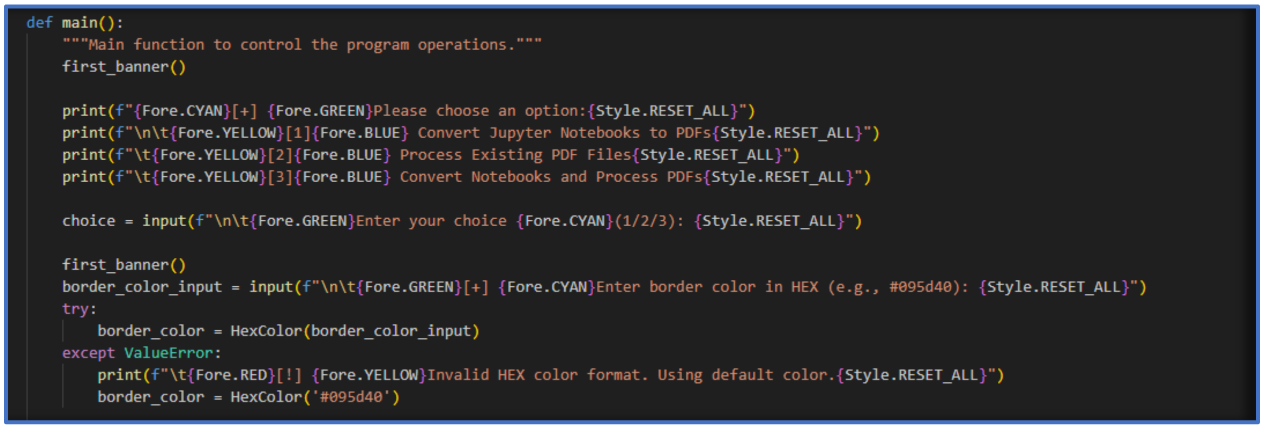


```python
def pdf_main(border_color):
    """Main function for processing existing PDF files."""
    download_folder = os.path.abspath('.')  # Use the current directory or specify another folder
    count = 1

    for file_name in os.listdir(download_folder):
        if file_name.endswith('.pdf'):
            full_pdf_path = os.path.join(download_folder, file_name)
            first_banner()
            ownername = input(f"\t{Fore.GREEN}[-] {Fore.BLUE}Enter the owner's name for the PDF
            '{Fore.CYAN}{file_name}{Fore.GREEN}': {Style.RESET_ALL}")
            process_pdfs(download_folder, full_pdf_path, ownername, count, border_color)
            count += 1
```

Figure 12 - Main Function To Process Existing PDF Files

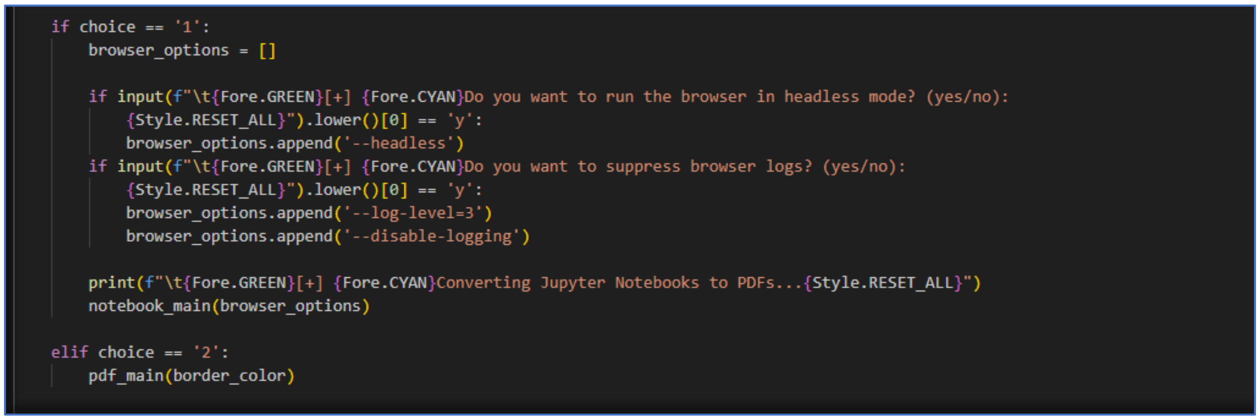
2.4. Main Control Flow

The `main` function manages user interaction and directs the flow of operations based on user input.



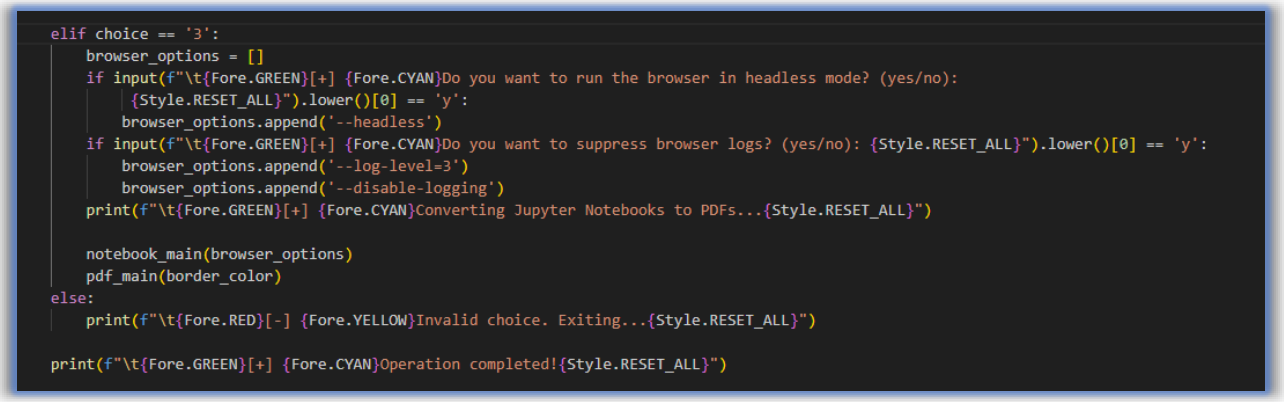
```
def main():
    """Main function to control the program operations."""
    first_banner()

    print(f"{Fore.CYAN}[+] {Fore.GREEN}Please choose an option:{Style.RESET_ALL}")
    print(f"\n\t{Fore.YELLOW}[1]{Fore.BLUE} Convert Jupyter Notebooks to PDFs{Style.RESET_ALL}")
    print(f"\t{Fore.YELLOW}[2]{Fore.BLUE} Process Existing PDF Files{Style.RESET_ALL}")
    print(f"\t{Fore.YELLOW}[3]{Fore.BLUE} Convert Notebooks and Process PDFs{Style.RESET_ALL}")

    choice = input(f"\n\t{Fore.GREEN}Enter your choice {Fore.CYAN}(1/2/3): {Style.RESET_ALL}")

    first_banner()
    border_color_input = input(f"\n\t{Fore.GREEN}[+] {Fore.CYAN}Enter border color in HEX (e.g., #095d40): {Style.RESET_ALL}")
    try:
        border_color = HexColor(border_color_input)
    except ValueError:
        print(f"\t{Fore.RED}[!] {Fore.YELLOW}Invalid HEX color format. Using default color.{Style.RESET_ALL}")
        border_color = HexColor('#095d40')
```

Figure 13 - Ask User Choices and Set Border



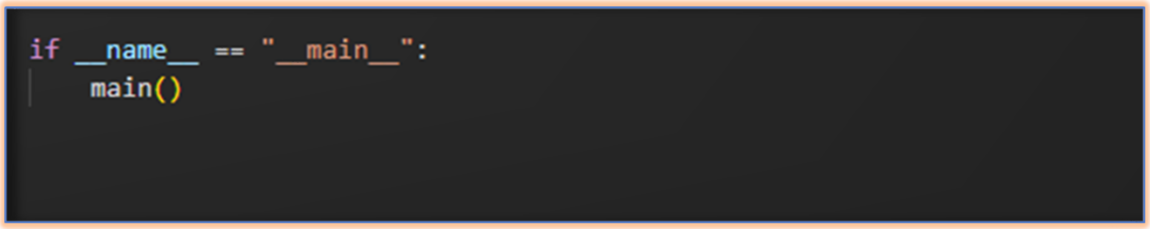
```
    if choice == '1':
        browser_options = []

        if input(f"\t{Fore.GREEN}[+] {Fore.CYAN}Do you want to run the browser in headless mode? (yes/no):
            {Style.RESET_ALL}").lower()[0] == 'y':
            browser_options.append('--headless')
        if input(f"\t{Fore.GREEN}[+] {Fore.CYAN}Do you want to suppress browser logs? (yes/no):
            {Style.RESET_ALL}").lower()[0] == 'y':
            browser_options.append('--log-level=3')
            browser_options.append('--disable-logging')

        print(f"\t{Fore.GREEN}[+] {Fore.CYAN}Converting Jupyter Notebooks to PDFs...{Style.RESET_ALL}")
        notebook_main(browser_options)

    elif choice == '2':
        pdf_main(border_color)
```

Figure 14 - Convert Notebook or Process PDF



```
    elif choice == '3':
        browser_options = []
        if input(f"\t{Fore.GREEN}[+] {Fore.CYAN}Do you want to run the browser in headless mode? (yes/no):
            {Style.RESET_ALL}").lower()[0] == 'y':
            browser_options.append('--headless')
        if input(f"\t{Fore.GREEN}[+] {Fore.CYAN}Do you want to suppress browser logs? (yes/no): {Style.RESET_ALL}").lower()[0] == 'y':
            browser_options.append('--log-level=3')
            browser_options.append('--disable-logging')
        print(f"\t{Fore.GREEN}[+] {Fore.CYAN}Converting Jupyter Notebooks to PDFs...{Style.RESET_ALL}")

        notebook_main(browser_options)
        pdf_main(border_color)
    else:
        print(f"\t{Fore.RED}[-] {Fore.YELLOW}Invalid choice. Exiting...{Style.RESET_ALL}")

    print(f"\t{Fore.GREEN}[+] {Fore.CYAN}Operation completed!{Style.RESET_ALL}")
```

Figure 15 - Convert Notebook and Process PDF



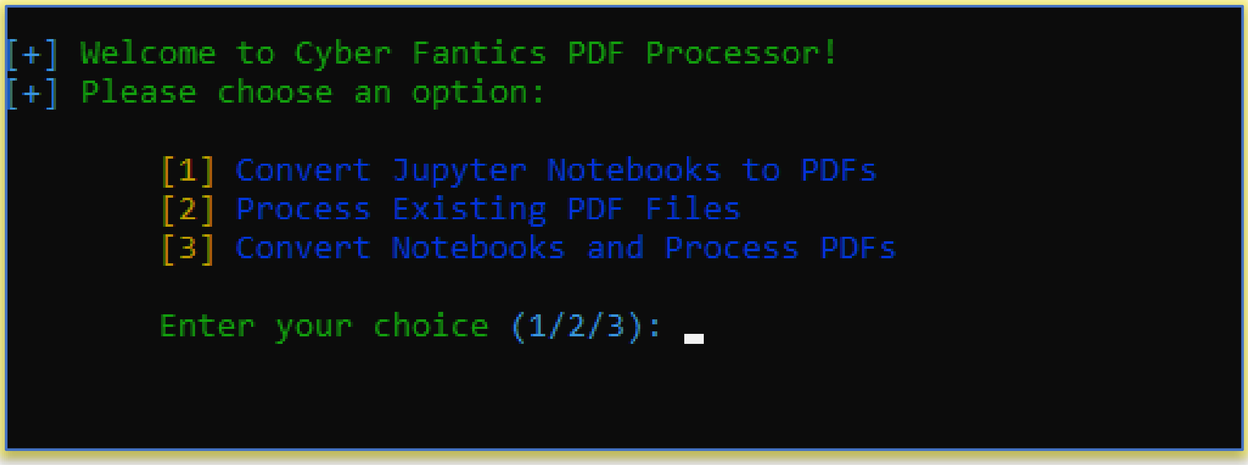
```
if __name__ == "__main__":
    main()
```

Figure 16 - Run main()

3. User Instructions

Running the Application:

1. Ensure all dependencies are installed using `pip install -r requirements.txt`.
2. Place Jupyter Notebooks or PDF files in the current directory or specify a different directory in the code.
3. Execute the script. You will be prompted to choose one of the following options:
   - ✓ **Convert Jupyter Notebooks to PDFs:** Converts `.ipynb` files in the directory to PDF format.
   - ✓ **Process Existing PDF Files:** Adds borders and page numbers to existing PDF files.
   - ✓ **Convert Notebooks and Process PDFs:** Performs both operations in sequence.

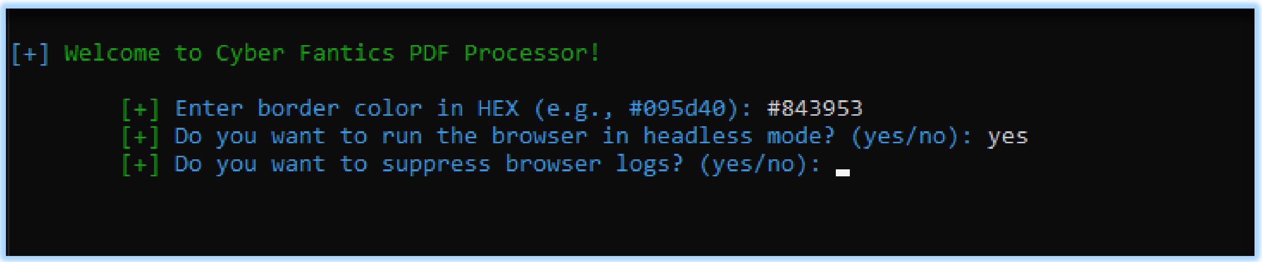


```
[+] Welcome to Cyber Fantics PDF Processor!
[+] Please choose an option:

        [1] Convert Jupyter Notebooks to PDFs
        [2] Process Existing PDF Files
        [3] Convert Notebooks and Process PDFs

        Enter your choice (1/2/3):
```

Figure 17 - Choice Prompt

Customizing Options:

- ❖ You can specify if you want the browser to run in headless mode and if you want to suppress browser logs.
- ❖ Set the border color for the processed PDFs using a HEX color code.



```
[+] Welcome to Cyber Fantics PDF Processor!

        [+] Enter border color in HEX (e.g., #095d40): #843953
        [+] Do you want to run the browser in headless mode? (yes/no): yes
        [+] Do you want to suppress browser logs? (yes/no):
```

Figure 18 - Customizing Options

Error Handling:

- Ensure that the path to files and folders is correct.
- Verify that the browser automation service is accessible and functioning.

4. Future Enhancements

- **GUI Interface:** Develop a graphical user interface for more intuitive interactions.
- **Error Handling Improvements:** Implement more robust error handling and logging mechanisms.
- **Support for Additional Formats:** Extend functionality to support additional file formats and conversions.
- **Optimization:** Enhance performance and efficiency of file processing and browser automation.

About the Developer

I am Syed Mansoor ul Hassan Bukhari, a passionate Python developer and AI enthusiast currently enrolled in an Advanced Python course at Corvit Muzaffarabad. Under the guidance of Sir Muhammad Rizwan, I am honing my skills in artificial intelligence, data processing, and Python development. I am dedicated to creating innovative Python-based tools and contributing to the developer community.

You can explore more of my projects on my GitHub profile: Cyber Fantics.

For this specific project, feel free to review the code and contribute on GitHub: Jupyter-to-PDF Processor.