Give me python code which:

1. Creates a window for users to drag and drop a PDF similar to this code

class DragAndDropWidget(QWidget):  
 *"""  
 DragAndDropWidget class for the drag-and-drop GUI  
 """* def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 self.setWindowTitle("PDF Processor")  
 self.setGeometry(300, 300, 400, 200)  
 layout = QVBoxLayout()  
 self.label = QLabel("Drag and Drop PDF Here")  
 self.label.setAlignment(Qt.AlignCenter)  
 layout.addWidget(self.label)  
 self.setLayout(layout)  
  
 def dragEnterEvent(self, event: QDragEnterEvent):  
 if event.mimeData().hasUrls():  
 event.acceptProposedAction()  
  
 def dragMoveEvent(self, event: QMouseEvent):  
 event.accept()  
  
 def dropEvent(self, event: QDropEvent):  
 pdf\_processor = PDFProcessor()  
 for url in event.mimeData().urls():  
 filepath = url.toLocalFile()  
 filepath = filepath.strip('{}')  
 if filepath.endswith('.pdf'):  
 logging.info(f"Processing PDF at {filepath}")  
 pdf\_processor.process\_pdf(filepath)  
 else:  
 logging.warning(f"File {filepath} is not a PDF file")  
 event.acceptProposedAction()  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 app = QApplication(sys.argv)  
 window = DragAndDropWidget()  
 window.setAcceptDrops(True)  
 window.show()  
 sys.exit(app.exec\_())

1. Pulls job titles from a PDF similar to this

class PDFProcessor:  
 *"""  
 PDFProcessor class to process PDFs and extract job titles  
 """* def extract\_job\_titles(self, pdf\_path):  
 job\_titles = []  
 try:  
 with open(pdf\_path, 'rb') as file:  
 reader = PyPDF2.PdfReader(file)  
 catalog = reader.trailer["/Root"]  
 form = catalog["/AcroForm"]  
 if "/Fields" in form:  
 fields = form["/Fields"]  
 for field\_ref in fields:  
 field = field\_ref.get\_object()  
 field\_name = field['/T']  
 if field\_name.startswith("Job Title"):  
 field\_value = field.get('/V')  
 job\_titles.append(field\_value)  
 except Exception as e:  
 logging.error(f"Error extracting job titles: {e}")  
 return job\_titles  
  
 def process\_pdf(self, file\_path):  
 job\_titles = self.extract\_job\_titles(file\_path)  
 if job\_titles:  
 scraper = WebScraper()  
 job\_titles\_data = []  
 total\_jobs = len(job\_titles)  
 threads = []  
 for job\_title in job\_titles:  
 thread = threading.Thread(target=self.scrape\_and\_save\_data, args=(job\_title, scraper, job\_titles\_data,))  
 threads.append(thread)  
 thread.start()  
 if len(threads) > 10: # Throttle to maximum 10 threads  
 for thread in threads:  
 thread.join()  
 threads = []  
 # Make sure remaining threads are joined  
 for thread in threads:  
 thread.join()  
 output\_dir = os.path.dirname(file\_path)  
 scraper.create\_word\_document(job\_titles\_data, output\_dir)  
 show\_message\_box("PDF Processed", "Word document created successfully.")  
 else:  
 show\_message\_box("No Job Titles", "No job titles found in the PDF.")  
  
 def scrape\_and\_save\_data(self, job\_title, scraper, job\_titles\_data):  
 query = f'{job\_title} onetonline.org'  
 search\_results = search(query, lang="en", pause=2)  
 num\_results = 1  
 counter = 0  
 for result in search\_results:  
 if counter >= num\_results:  
 break  
 onet\_link = result  
 data = scraper.scrape\_data(job\_title, onet\_link)  
 job\_titles\_data.append((job\_title, data))  
 counter += 1  
 time.sleep(2)

1. Checks [www.onetonline.org](http://www.onetonline.org) for those job titles and selects the tasks associated with those job titles similar to this.

class WebScraper:  
 *"""  
 WebScraper class to scrape data and create a Word document  
 """* def scrape\_data(self, job\_title, onet\_link):  
 try:  
 response = requests.get(onet\_link)  
 response.raise\_for\_status()  
 soup = BeautifulSoup(response.text, 'html.parser')  
 div\_elements = soup.select('div.order-2.flex-grow-1')  
 data = []  
 for div in div\_elements:  
 if div.find('div', {'id': 'TechnologySkills'}) or div.find('b'):  
 break  
 text = div.text.strip()  
 data.append(text)  
 return data[:20]  
 except requests.exceptions.RequestException as e:  
 logging.error(f"Error scraping data for {job\_title}: {e}")  
 return []

1. Creates a Word document similar to this

def create\_word\_document(self, job\_titles\_data, output\_dir):  
 doc = Document()  
 for job\_title, data in job\_titles\_data:  
 doc.add\_heading(job\_title, level=1)  
 for item in data:  
 doc.add\_paragraph(item, style='List Bullet')  
 doc.add\_page\_break()  
 now = datetime.now()  
 date\_time\_str = now.strftime("%m-%d-%Y-%H%M")  
 doc.save(os.path.join(output\_dir, f'Skills List {date\_time\_str}.docx'))