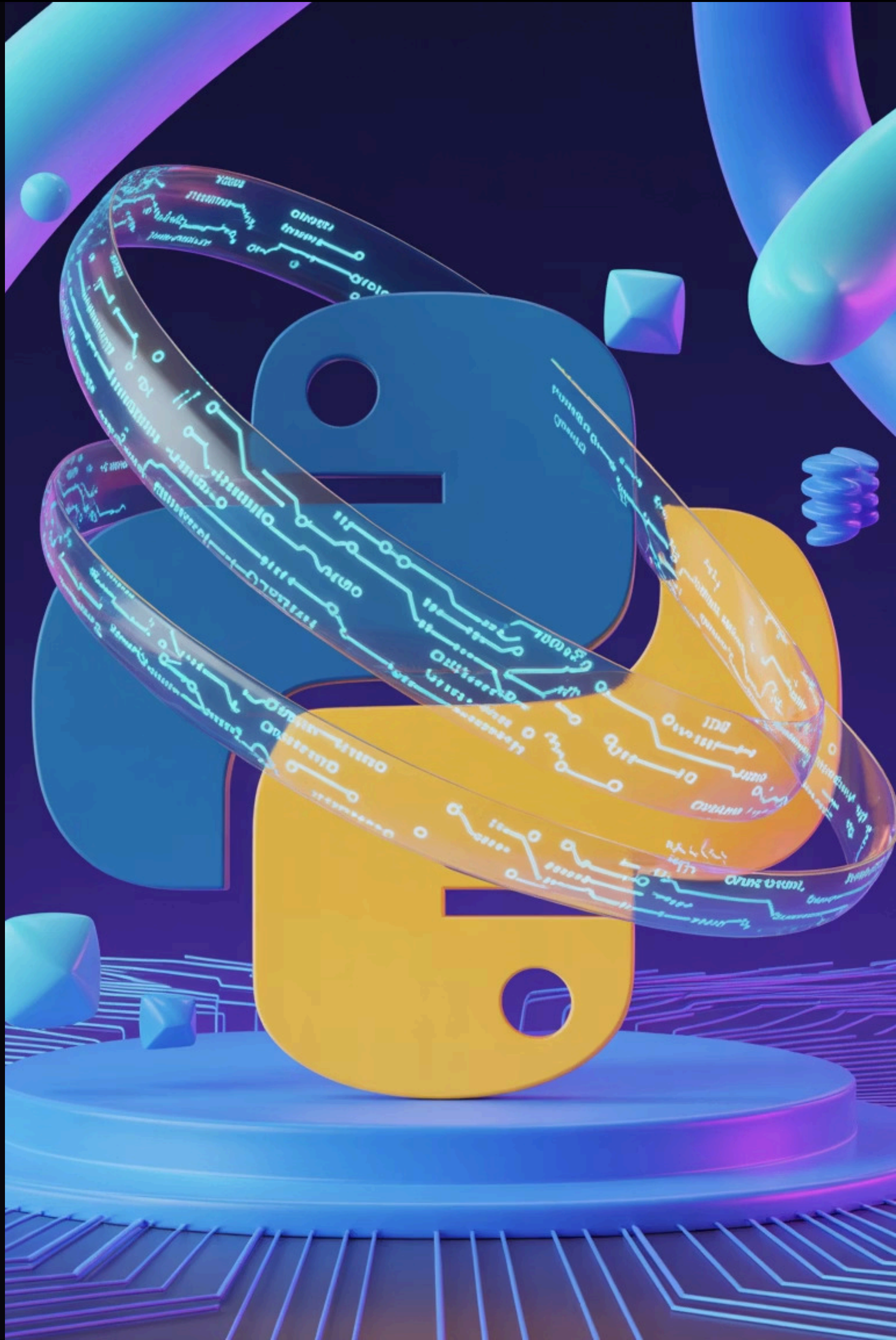
Python Meets AI: Automating Everyday Tasks with Just a Few Lines of Code

A practical guide to simplifying your workflow using Python and artificial intelligence



The "Boring Task" Problem

Ever spent hours sorting through hundreds of images to find specific content? Or manually transcribing meeting recordings? Perhaps you've faced the tedium of extracting and summarising information from dozens of PDFs?

These repetitive tasks drain your productivity and creative energy—but there's a better way.

What You'll Learn Today

01

Why Python + AI Is the Perfect Automation Combination

Understanding the power of combining Python's simplicity with modern AI capabilities

02

Three Practical Automation Examples

Ready-to-implement solutions for PDF summarisation, voice transcription, and image sorting

03

Tips for Creating Advanced Workflows

How to combine multiple automations and schedule them for maximum efficiency

04

Resources to Continue Your Journey

Links to GitHub repositories, additional libraries, and community resources

Why AI + Python Is Perfect for Automation

Python's Strengths

- Readable, approachable syntax
- Vast ecosystem of libraries
- Strong community support
- Cross-platform compatibility

AI Capabilities

- Natural language understanding
- Image recognition and classification
- Speech-to-text conversion
- Pattern recognition in data

Together, they create a powerful toolkit that lets developers of any skill level build sophisticated automation solutions without deep AI expertise.

Real-World Use Cases

Content Management

- Automatic document categorisation
- Content summarisation and extraction
- Media organisation and tagging

Communication

- Meeting transcription and minutes
- Email categorisation and response drafting
- Multilingual translation

Data Processing

- Report generation from raw data
- Data cleaning and normalisation
- Anomaly detection in datasets

Getting Started: Essential Libraries

Transformers

State-of-the-art NLP models from Hugging Face for text processing, summarisation, and generation

```
pip install transformers
```

PyTorch/TorchVision

Powerful deep learning frameworks for computer vision and image processing tasks

```
pip install torch torchvision
```

Whisper

OpenAI's robust speech recognition system for accurate audio transcription

```
pip install openai-whisper
```

Example 1: Summarising PDFs

Let's turn hours of reading into minutes with AI

The PDF Summarisation Challenge

Imagine facing a folder with dozens of research papers, reports, or legal documents. Reading each one thoroughly could take days or weeks.

With Python and AI, we can build a tool that:

- Extracts text from multiple PDFs
- Processes the content with a language model
- Generates concise, accurate summaries
- Outputs results in an organised format

PDF Summarisation: The Code

```
import PyPDF2
from transformers import pipeline

# Extract text from PDF
def extract_text(pdf_path):
    with open(pdf_path, 'rb') as file:
        reader = PyPDF2.PdfReader(file)
        text = ""
        for page in reader.pages:
            text += page.extract_text()
    return text

# Summarise with transformers
def summarise(text):
    summarizer = pipeline("summarization",
                          model="facebook/bart-large-cnn")
    summary = summarizer(text,
                         max_length=150,
                         min_length=40,
                         do_sample=False)
    return summary[0]['summary_text']

# Process a PDF file
pdf_text = extract_text("research_paper.pdf")
summary = summarise(pdf_text)
print(summary)
```

Key Components:

- **PyPDF2** handles PDF file operations
- **Transformers** provides the BART model for summarisation
- The **pipeline** abstraction simplifies complex AI operations
- **Parameters** control summary length and style

This solution works with any PDF containing extractable text and can be easily expanded to process entire directories of documents.

Example 2: Voice-to-Text Meeting Notes

Never miss an important detail in meetings again

The Meeting Transcription Challenge

Remote and hybrid work has led to more recorded meetings than ever before. But who has time to re-listen to hours of recordings to find key information?

Our Python + AI solution will:

- Convert speech recordings to text with high accuracy
- Handle multiple speakers and accents
- Identify and highlight key discussion points
- Create searchable, shareable transcripts

Voice-to-Text: The Code

```
import whisper
import datetime

# Load Whisper model (choose size based on needs)
model = whisper.load_model("medium")

# Transcribe audio file
def transcribe_meeting(audio_file):
    result = model.transcribe(audio_file)
    return result["text"]

# Process transcript to extract key points
def extract_key_points(transcript):
    # Use NLP to identify important segments
    # This is simplified - you would use a model
    # trained to extract meeting highlights
    keywords = ["action item", "decision",
               "follow up", "deadline"]

    lines = transcript.split('.')
    key_points = []

    for line in lines:
        if any(kw in line.lower() for kw in keywords):
            key_points.append(line.strip())

    return key_points

# Main process
audio_file = "weekly_meeting.mp3"
transcript = transcribe_meeting(audio_file)
key_points = extract_key_points(transcript)

# Save results
timestamp = datetime.datetime.now().strftime("%Y%m%d")
with open(f"meeting_notes_{timestamp}.txt", "w") as f:
    f.write(transcript)
    f.write("\n\nKEY POINTS:\n")
    for point in key_points:
        f.write(f"• {point}\n")
```

Why This Works:

- **Whisper** provides state-of-the-art speech recognition accuracy
- The **medium model** balances speed and accuracy
- **Keyword extraction** helps identify important meeting segments
- **Formatting** makes the output immediately useful

This can be enhanced with speaker diarization (identifying who said what) using additional libraries like pyannote.audio.

Example 3: Image Sorting by Content

Let AI organize your visual content automatically

The Image Organisation Challenge

Whether you're a photographer with thousands of images, a research team with countless microscopy samples, or simply managing family photos, manual sorting is incredibly time-consuming.

Our solution leverages computer vision to:

- Automatically classify images by content
- Sort files into appropriate folders
- Handle large batches efficiently
- Work with minimal human supervision

Image Sorting: The Code

```python
import os
import shutil
from PIL import Image
import torch
from torchvision import transforms
from torchvision.models import resnet50

# Setup model
def setup_model():
    model = resnet50(pretrained=True)
    model.eval()
    return model

# Image preprocessing
preprocess = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406],
                         [0.229, 0.224, 0.225])
])

# Classify an image
def classify_image(img_path, model):
    img = Image.open(img_path).convert('RGB')
    img_t = preprocess(img)
    batch_t = torch.unsqueeze(img_t, 0)

    with torch.no_grad():
        output = model(batch_t)

    # For simplicity, return class index with highest score
    # Real implementation would map to class names
    return torch.argmax(output, dim=1).item()

# Sort images in a directory
def sort_images(source_dir, output_dir, model):
    os.makedirs(output_dir, exist_ok=True)

    for img_file in os.listdir(source_dir):
        if img_file.lower().endswith(('.png', '.jpg', '.jpeg')):
            img_path = os.path.join(source_dir, img_file)
            class_id = classify_image(img_path, model)

            # Create category directory if it doesn't exist
            category_dir = os.path.join(output_dir, f"category_{class_id}")
            os.makedirs(category_dir, exist_ok=True)

            # Copy the image to its category directory
            shutil.copy(img_path, os.path.join(category_dir, img_file))
            print(f"Sorted {img_file} to {category_dir}")
```

Advanced Implementation:

- Replace ResNet with a custom-trained model for your specific categories
- Use descriptive folder names instead of numeric IDs
- Add confidence thresholds to create an "uncertain" category for manual review
- Implement metadata extraction for additional sorting criteria

This approach works with any image classification model, including custom models trained on your specific image types.

Going Further: Building Complete Workflows

Connect Multiple Automations

Chain your scripts together to create end-to-end solutions. For example, transcribe a meeting, summarise the transcript, and email the summary to participants.

Add APIs and Web Services

Integrate with cloud services, messaging platforms, or database systems to extend functionality beyond your local environment.

Schedule Regular Execution

Use tools like cron (Linux/Mac) or Windows Task Scheduler to run your automations on a schedule without manual intervention.

Scheduling Your Automations

Using APScheduler

```
from apscheduler.schedulers.blocking import BlockingScheduler
from datetime import datetime

# Create scheduler
scheduler = BlockingScheduler()

# Schedule PDF processing every weekday at 9 AM
@scheduler.scheduled_job('cron', day_of_week='mon-fri', hour=9)
def process_daily_reports():
    print(f"Processing reports at {datetime.now()}")
    # Call your PDF processing function here
    process_pdf_directory("daily_reports")

# Schedule meeting transcription every Monday at 3 PM
@scheduler.scheduled_job('cron', day_of_week='mon', hour=15)
def transcribe_weekly_meeting():
    print(f"Transcribing meeting at {datetime.now()}")
    # Call your transcription function here
    transcribe_meeting("weekly_meeting.mp3")

# Start the scheduler
scheduler.start()
```

Benefits of Scheduled Automation:

- **Consistency:** Tasks run reliably without human error
- **Time optimization:** Schedule resource-intensive tasks during off-hours
- **Integration:** Align with business processes (run after daily reports are generated)
- **Scalability:** Easy to add new tasks to the schedule

APScheduler provides flexible scheduling options including interval-based (every X minutes), cron-based (specific times), and one-off delayed execution.

Best Practices for Python AI Automation

Start with Pre-trained Models

Unless you have specific requirements, leverage existing models to save time and resources. Fine-tune only when necessary.

Handle Errors Gracefully

Automated systems should fail safely. Implement proper error handling, logging, and notification systems.

Consider Computational Resources

AI operations can be resource-intensive. Choose appropriate model sizes and implement batching for efficiency.

Test with Real-World Data

Ensure your automation works with the messy, inconsistent data you'll encounter in production, not just with clean examples.

Resources to Continue Your Journey

Libraries and Tools

- **Hugging Face Transformers:** State-of-the-art NLP models
- **Streamlit:** Create web interfaces for your automation tools
- **FastAPI:** Build APIs around your Python automations
- **Langchain:** Framework for building applications with LLMs
- **Pydantic:** Data validation for more robust automations

Learning Resources

- **Hugging Face Course:** Free NLP and transformers tutorials
- **PyTorch Vision Tutorial:** Computer vision fundamentals
- **Real Python:** Practical Python programming tutorials
- **Papers with Code:** Latest AI research with implementations
- **GitHub Repository:** Complete code examples from this presentation

Key Takeaways

1. **Start Small**

   Begin with a single repetitive task and build from there

2. **Leverage Existing Tools**

   Pre-trained models and Python libraries provide tremendous power with minimal effort

3. **Think in Workflows**

   Connect individual automations to create comprehensive solutions that transform your productivity

4. **Share Your Creations**

   Contribute to the community by sharing your code, approaches, and lessons learned as you develop your own automation solutions

The combination of Python and AI has democratised automation capabilities that were once accessible only to large organisations with dedicated teams.