```
!pip install -U yt-dlp faster-whisper transformers accelerate sentencepiece
```

```
Requirement already satisfied: yt-dlp in /usr/local/lib/python3.12/dist-packages (2026.2.4)
Requirement already satisfied: faster-whisper in /usr/local/lib/python3.12/dist-packages (1.2.1)
Requirement already satisfied: transformers in /usr/local/lib/python3.12/dist-packages (5.1.0)
Requirement already satisfied: accelerate in /usr/local/lib/python3.12/dist-packages (1.12.0)
Requirement already satisfied: sentencepiece in /usr/local/lib/python3.12/dist-packages (0.2.1)
Requirement already satisfied: ctranslate2<5,>=4.0 in /usr/local/lib/python3.12/dist-packages (from faster-whisper) (4.7.1)
Requirement already satisfied: huggingface-hub>=0.21 in /usr/local/lib/python3.12/dist-packages (from faster-whisper) (1.4.0
Requirement already satisfied: tokenizers<1,>=0.13 in /usr/local/lib/python3.12/dist-packages (from faster-whisper) (0.22.2)
Requirement already satisfied: onnxruntime<2,>=1.14 in /usr/local/lib/python3.12/dist-packages (from faster-whisper) (1.24.1
Requirement already satisfied: av>=11 in /usr/local/lib/python3.12/dist-packages (from faster-whisper) (16.1.0)
Requirement already satisfied: tqdm in /usr/local/lib/python3.12/dist-packages (from faster-whisper) (4.67.3)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.12/dist-packages (from transformers) (2.0.2)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.12/dist-packages (from transformers) (26.0)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.12/dist-packages (from transformers) (6.0.3)
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.12/dist-packages (from transformers) (2025.11.3)
Requirement already satisfied: typer-slim in /usr/local/lib/python3.12/dist-packages (from transformers) (0.21.1)
Requirement already satisfied: safetensors>=0.4.3 in /usr/local/lib/python3.12/dist-packages (from transformers) (0.7.0)
Requirement already satisfied: psutil in /usr/local/lib/python3.12/dist-packages (from accelerate) (5.9.5)
Requirement already satisfied: torch>=2.0.0 in /usr/local/lib/python3.12/dist-packages (from accelerate) (2.9.0+cu128)
Requirement already satisfied: setuptools in /usr/local/lib/python3.12/dist-packages (from ctranslate2<5,>=4.0->faster-whisp
Requirement already satisfied: filelock in /usr/local/lib/python3.12/dist-packages (from huggingface-hub>=0.21->faster-whisp
Requirement already satisfied: fsspec>=2023.5.0 in /usr/local/lib/python3.12/dist-packages (from huggingface-hub>=0.21->fast
Requirement already satisfied: hf-xet<2.0.0,>=1.2.0 in /usr/local/lib/python3.12/dist-packages (from huggingface-hub>=0.21->
Requirement already satisfied: httpx<1,>=0.23.0 in /usr/local/lib/python3.12/dist-packages (from huggingface-hub>=0.21->fast
Requirement already satisfied: shellingham in /usr/local/lib/python3.12/dist-packages (from huggingface-hub>=0.21->faster-wh
Requirement already satisfied: typing-extensions>=4.1.0 in /usr/local/lib/python3.12/dist-packages (from huggingface-hub>=0.
Requirement already satisfied: flatbuffers in /usr/local/lib/python3.12/dist-packages (from onnxruntime<2,>=1.14->faster-whi
Requirement already satisfied: protobuf in /usr/local/lib/python3.12/dist-packages (from onnxruntime<2,>=1.14->faster-whispe
Requirement already satisfied: sympy in /usr/local/lib/python3.12/dist-packages (from onnxruntime<2,>=1.14->faster-whisper)
Requirement already satisfied: networkx>=2.5.1 in /usr/local/lib/python3.12/dist-packages (from torch>=2.0.0->accelerate) (3
Requirement already satisfied: jinja2 in /usr/local/lib/python3.12/dist-packages (from torch>=2.0.0->accelerate) (3.1.6)
Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.8.93 in /usr/local/lib/python3.12/dist-packages (from torch>=2.0.0
Requirement already satisfied: nvidia-cuda-runtime-cu12==12.8.90 in /usr/local/lib/python3.12/dist-packages (from torch>=2.0
Requirement already satisfied: nvidia-cuda-cupti-cu12==12.8.90 in /usr/local/lib/python3.12/dist-packages (from torch>=2.0.0
Requirement already satisfied: nvidia-cudnn-cu12==9.10.2.21 in /usr/local/lib/python3.12/dist-packages (from torch>=2.0.0->a
Requirement already satisfied: nvidia-cublas-cu12==12.8.4.1 in /usr/local/lib/python3.12/dist-packages (from torch>=2.0.0->a
Requirement already satisfied: nvidia-cufft-cu12==11.3.3.83 in /usr/local/lib/python3.12/dist-packages (from torch>=2.0.0->a
Requirement already satisfied: nvidia-curand-cu12==10.3.9.90 in /usr/local/lib/python3.12/dist-packages (from torch>=2.0.0->
Requirement already satisfied: nvidia-cusolver-cu12==11.7.3.90 in /usr/local/lib/python3.12/dist-packages (from torch>=2.0.0
Requirement already satisfied: nvidia-cusparse-cu12==12.5.8.93 in /usr/local/lib/python3.12/dist-packages (from torch>=2.0.0
Requirement already satisfied: nvidia-cusparselt-cu12==0.7.1 in /usr/local/lib/python3.12/dist-packages (from torch>=2.0.0->
Requirement already satisfied: nvidia-nccl-cu12==2.27.5 in /usr/local/lib/python3.12/dist-packages (from torch>=2.0.0->accel
Requirement already satisfied: nvidia-nvshmem-cu12==3.3.20 in /usr/local/lib/python3.12/dist-packages (from torch>=2.0.0->ac
Requirement already satisfied: nvidia-nvtx-cu12==12.8.90 in /usr/local/lib/python3.12/dist-packages (from torch>=2.0.0->acce
Requirement already satisfied: nvidia-nvjitlink-cu12==12.8.93 in /usr/local/lib/python3.12/dist-packages (from torch>=2.0.0-
Requirement already satisfied: nvidia-cufile-cu12==1.13.1.3 in /usr/local/lib/python3.12/dist-packages (from torch>=2.0.0->a
Requirement already satisfied: triton==3.5.0 in /usr/local/lib/python3.12/dist-packages (from torch>=2.0.0->accelerate) (3.5
Requirement already satisfied: click>=8.0.0 in /usr/local/lib/python3.12/dist-packages (from typer-slim->transformers) (8.3.
Requirement already satisfied: anyio in /usr/local/lib/python3.12/dist-packages (from httpx<1,>=0.23.0->huggingface-hub>=0.2
Requirement already satisfied: certifi in /usr/local/lib/python3.12/dist-packages (from httpx<1,>=0.23.0->huggingface-hub>=0
Requirement already satisfied: httpcore==1.* in /usr/local/lib/python3.12/dist-packages (from httpx<1,>=0.23.0->huggingface-
Requirement already satisfied: idna in /usr/local/lib/python3.12/dist-packages (from httpx<1,>=0.23.0->huggingface-hub>=0.21
Requirement already satisfied: h11>=0.16 in /usr/local/lib/python3.12/dist-packages (from httpcore==1.*->httpx<1,>=0.23.0->h
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.12/dist-packages (from sympy->onnxruntime<2,>=1.
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.12/dist-packages (from jinja2->torch>=2.0.0->accele
```

```
import os
import yt_dlp
import torch
from faster_whisper import WhisperModel

device = "cuda" if torch.cuda.is_available() else "cpu"
compute_type = "float16" if device == "cuda" else "int8"

print("Loading Whisper model into VRAM...")
whisper_model = WhisperModel("turbo", device=device, compute_type=compute_type)
print("Whisper model loaded!")

def get_transcript(video_url):
    """Downloads audio and returns timestamped segments."""
    temp_audio = "temp_audio"

    ydl_opts = {
        'format': 'bestaudio/best',
        'postprocessors': [{
            'key': 'FFmpegExtractAudio',
            'preferredcodec': 'mp3',
            'preferredquality': '128',
        }],
        'outtmpl': temp_audio,
        'quiet': True,
        'no_warnings': True,
```

```
    }

    print("   -> Downloading audio track...")
    try:
        with yt_dlp.YoutubeDL(ydl_opts) as ydl:
            ydl.download([video_url])
    except Exception as e:
        return f"Error downloading audio: {str(e)}"

    print("   -> Transcribing audio with Faster-Whisper...")
    try:
        # Transcribe the downloaded mp3
        segments, info = whisper_model.transcribe(temp_audio + ".mp3", beam_size=5)

        transcript_data = []
        for segment in segments:
            # Save both the start time and the text
            transcript_data.append({
                "start": segment.start,
                "text": segment.text.strip()
            })

        # Clean up
        if os.path.exists(temp_audio + ".mp3"):
            os.remove(temp_audio + ".mp3")

        return transcript_data
    except Exception as e:
        return f"Error transcribing audio: {str(e)}"
```

```
Loading Whisper model into VRAM...
Whisper model loaded!
```

```
import torch
from transformers import AutoTokenizer, AutoModelForCausalLM, pipeline
import transformers

transformers.logging.set_verbosity_error()

device = "cuda" if torch.cuda.is_available() else "cpu"

model_name = "Qwen/Qwen2.5-3B-Instruct"

print(f"Loading {model_name} on {device}... this might take a minute.")

tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForCausalLM.from_pretrained(
    model_name,
    torch_dtype=torch.float16,
    device_map="auto"
)

pipe = pipeline("text-generation", model=model, tokenizer=tokenizer)

print("Model loaded successfully!")
```

```
Loading Qwen/Qwen2.5-3B-Instruct on cuda... this might take a minute.
Loading weights: 100%                                                    434/434 [00:26<00:00, 15.49it/s, Materializing param=model.norm.weight]
Model loaded successfully!
```

```
def summarize_chunk(text_chunk, is_overview=False):
    if is_overview:
        system_prompt = "You are an expert technical educator. Your goal is to create a comprehensive, high-level course ov
        user_prompt = f"Summarize the following tutorial notes into a cohesive course overview:\n\n{text_chunk}"
        max_tokens = 350
    else:
        system_prompt = "You are an expert technical note-taker. Extract the key programming concepts from the text. You Ml
        user_prompt = f"Extract structured notes and code syntax from this transcript section:\n\n{text_chunk}"
        max_tokens = 250

    messages = [
        {"role": "system", "content": system_prompt},
        {"role": "user", "content": user_prompt},
    ]

    prompt = tokenizer.apply_chat_template(
        messages,
        tokenize=False,
        add_generation_prompt=True
    )
```

```
    outputs = pipe(
        prompt,
        max_new_tokens=max_tokens,
        max_length=None,
        do_sample=True,
        temperature=0.3,
        top_p=0.9,
    )

    generated_text = outputs[0]["generated_text"]
    response = generated_text[len(prompt):].strip()

    return response
```

```
def format_timestamp(seconds):
    """Converts raw seconds into a clean MM:SS format."""
    mins = int(seconds // 60)
    secs = int(seconds % 60)
    hours = int(mins // 60)
    if hours > 0:
        mins = mins % 60
        return f"{hours}:{mins:02d}:{secs:02d}"
    return f"{mins:02d}:{secs:02d}"

# SHRUNK CHUNK DURATION: 90 seconds for tech tutorials
def chunk_transcript_by_time(transcript_data, chunk_duration_sec=90):
    """Groups transcript segments into tight time-based blocks."""
    if isinstance(transcript_data, str):
        return []

    chunks = []
    current_chunk_text = ""

    if not transcript_data:
        return chunks

    current_chunk_start = transcript_data[0]["start"]

    for segment in transcript_data:
        if segment["start"] - current_chunk_start >= chunk_duration_sec:
            chunks.append({
                "timestamp": format_timestamp(current_chunk_start),
                "seconds": current_chunk_start,
                "text": current_chunk_text.strip()
            })
            current_chunk_text = segment["text"] + " "
            current_chunk_start = segment["start"]
        else:
            current_chunk_text += segment["text"] + " "

    if current_chunk_text:
        chunks.append({
            "timestamp": format_timestamp(current_chunk_start),
            "seconds": current_chunk_start,
            "text": current_chunk_text.strip()
        })

    return chunks
```

```
import base64
from IPython.display import display, Markdown, HTML

def format_time_url(url, seconds):
    joiner = "&" if "?" in url else "?"
    return f"{url}{joiner}t={int(seconds)}s"

def export_and_display_markdown(overview, detailed_notes, video_url):
    md = f"# 📝 AI Generated Tutorial Notes\n\n"
    md += f"**Source Video:** {video_url}\n\n"

    md += f"## 📌 Course Overview\n\n{overview}\n\n"
    md += f"## ⏱️ Detailed Timestamped Notes\n\n"

    for note in detailed_notes:
        clickable_url = format_time_url(video_url, note['seconds'])
        md += f"### [{note['timestamp']}]({clickable_url})\n\n"
        md += f"{note['text']}\n\n"

    filename = "tutorial_notes.md"

    with open(filename, "w", encoding="utf-8") as f:
```

```
        f.write(md)

    b64_md = base64.b64encode(md.encode('utf-8')).decode('utf-8')
    download_html = f'''
        <div style="margin: 20px 0;">
            <a href="data:text/markdown;base64,{b64_md}" download="{filename}"
               style="background-color: #4CAF50; color: white; padding: 10px 20px; text-decoration: none; border-radius: 5p
                📥 Download Notes (.md)
            </a>
        </div>
    '''

    print("\n✅ Notes successfully generated!")
    display(HTML(download_html))
    display(Markdown(md))

def generate_video_notes(video_url):
    print(f"\n🎬 Processing video: {video_url}")

    print("🎧 Fetching and transcribing audio...")
    transcript_data = get_transcript(video_url)

    if isinstance(transcript_data, str) and transcript_data.startswith("Error"):
        print(transcript_data)
        return

    print("🔪 Chunking transcript by timestamps...")

    chunks = chunk_transcript_by_time(transcript_data, chunk_duration_sec=90)
    print(f"   -> {len(chunks)} time-blocks created.")

    print("🧠 Generating detailed timestamped notes...")
    all_notes = []
    summary_texts = []

    for i, chunk in enumerate(chunks):
        print(f"   Writing section {i+1}/{len(chunks)}...")

        summary = summarize_chunk(chunk["text"], is_overview=False)

        all_notes.append({
            "timestamp": chunk["timestamp"],
            "seconds": chunk["seconds"],
            "text": summary
        })
        summary_texts.append(summary)

    print("✨ Generating high-level course overview...")
    combined_notes = "\n\n".join(summary_texts)


    overview = summarize_chunk(combined_notes, is_overview=True)

    print("\n" + "="*60)
    print("🎨 RENDERING FINAL OUTPUT")
    print("="*60)

    export_and_display_markdown(overview, all_notes, video_url)

if __name__ == "__main__":
    url = input("Paste YouTube URL: ")
    generate_video_notes(url)
```

```
Paste YouTube URL: https://youtu.be/WEm3EUdicDg?si=IxUdcRjMMM1-Fhdf

🎬 Processing video: https://youtu.be/WEm3EUdicDg?si=IxUdcRjMMM1-Fhdf
🎧 Fetching and transcribing audio...
   -> Downloading audio track...
   -> Transcribing audio with Faster-Whisper...
🔪 Chunking transcript by timestamps...
   -> 14 time-blocks created.
🧠 Generating detailed timestamped notes...
   Writing section 1/14...
   Writing section 2/14...
   Writing section 3/14...
   Writing section 4/14...
   Writing section 5/14...
   Writing section 6/14...
   Writing section 7/14...
   Writing section 8/14...
   Writing section 9/14...
   Writing section 10/14...
   Writing section 11/14...
   Writing section 12/14...
   Writing section 13/14...
   Writing section 14/14...
✨ Generating high-level course overview...

============================================================
🎨 RENDERING FINAL OUTPUT
============================================================

✅ Notes successfully generated!

   📥 Download Notes (.md)
```

📝 AI Generated Tutorial Notes

Source Video: https://youtu.be/WEm3EUdicDg?si=IxUdcRjMMM1-Fhdf

📌 Course Overview

Course Overview

Course Title:

Introduction to Python Programming

Course Duration:

Approximately 4 weeks

Prerequisites:

Basic understanding of computer operations and a desire to learn a modern programming language.

Course Description:

This course provides a comprehensive introduction to Python programming, covering fundamental programming concepts, syntax, and practical applications. The curriculum is designed to build a strong foundation in Python, preparing learners for more advanced topics and real-world projects.

Main Topics Covered:

1. **Programming Basics**
   - **Comments**: Understanding how to use comments in Python to document code.
   - **Variables**: Introduction to creating and manipulating variables, including different data types such as numbers, strings, and booleans.
   - **Operators**: Exploring arithmetic, comparison, and logical operators.
   - **Loops**: Learning about `for` and `while` loops and their applications.
   - **Functions**: Discovering how to define and call functions in Python.
   - **Classes and Objects**: Introduction to object-oriented programming concepts, including classes, objects, and methods.
2. **Syntax and Basic Constructs**
   - **Syntax Examples**: Detailed examples of basic syntax, including comments, variable declarations, and printing values.
   - **Data Types**: Understanding built-in data types like `int`, `float`, `str`, and `bool`.
   - **String Literals**: Working with strings, including multi-line strings and string indexing.
   - **Boolean Values**: Introduction to boolean logic and conditional statements.
   - **Conditional Statements**: Using `if`, `elif`, and `else` to control program flow.
   - **Iteration**: Exploring `for` and `while` loops for iterating through sequences.
3. **Advanced

⏱️ Detailed Timestamped Notes

00:00

- Key programming concepts:
  - Comments: # symbol to start a comment, ignores when running script
  - Variables: Created by assigning a value to a name (variable)
  - Data types: Numbers, strings, booleans
  - Operators
  - Loops
  - Functions
  - Classes
  - Objects

- Syntax examples:
  - Comment: `# This is a comment`
  - Variable assignment: `my_variable = "Hello World"`
  - Triple quotes for multi-line comments: `'''This is a multi-line comment'''`

01:31

- Variables declaration: `x = 10`
- Printing variable values: `print(x)`
- Assigning different types to variables:
  - `y = "coding"`
  - `z = 3`
  - `z = 3.0`
- Changing variable type:
  - Casting: `x = str(10)`
- String declaration:
  - Using double quotes: `"coding"`
  - Using single quotes: `'coding'`

03:03

- Variable naming: Case sensitivity (e.g., `A` vs. `a`)
- Code example: Assigning values (`Python = A`, `variables = a`)
- Printing variables: `print(A)` and `print(variables)`
- Data types:
  - Built-in types: `str`, `int`, `float`, `list`, `tuple`, `range`, `bool`
  - Getting data type: Using `type()` function
    ```
    x = 5
    print(type(x))
    ```
  - Numeric types: `int` and `float`
- Creating numeric variables: By assigning values

04:34

- Variables assigned: `x = 1`, `y = 2.8`
- Printing types: `print(type(x))`, `print(type(y))`
- Data types:
  - `int` (integer): `x`
  - `float` (floating point number): `y`
- String literals:
  - Single quoted: `'Hello'`
  - Double quoted: `"Hello"`
  - Multi-line string assignment: `a = """some sample text in multi-lines"""`
- String properties:
  - Can span multiple lines using triple quotes (`'''` or `"""`)
  - Accessing individual characters using square brackets: `a[0]` gives 'h'
- Unicode representation: Strings in Python are represented as sequences of bytes

06:04

- String indexing: `print(a[1])` - Accesses second character (index 1)
- String length: `len(a)` - Returns length of string `a`
- Boolean values: `True`, `False`
- Comparison operators: `is greater than`, `equals to`, `is smaller than`
- Conditional statements: `if` statement
- Example `if` statement:

```
a = 100
b = 50
if b > a:
    print("b is greater than a")
else:
    print("b is not greater than a")
```

07:35

- Boolean function evaluation: `boolean`
- Examples:
  - `print(boolean "hello")` returns `true`
  - `print(boolean 15)` returns `true`
- Evaluation rules:
  - Non-empty strings (`"abc"`, `"123"`) are `true`
  - Non-zero numbers (`15`, `123`) are `true`
  - Non-empty lists (`["apple", "cherry", "banana"]`) are `true`
  - Empty values (`false`, `None`, `0`, `""`, `[]`, `{}`) are `false`
- Specific falsy values:
  - `false`
  - `0`
  - `None`
  - Empty strings (`""`)
  - Empty lists (`[]`)
  - Empty dictionaries (`{}`)

09:06

- Operators:
  - Addition: `print(10 + 10)`
  - Subtraction: -

    - Multiplication: *
    - Modulus: %
    - Assignment operators: +=, -=, etc.
    - Comparison operators: ==, >=, <=
- Python Lists:
    - Created using square brackets: `[apple, banana, oranges]`
    - Accessed via index: `list[0]`
    - Ordered, mutable, allows duplicates
    - Example usage: `print(list)`

10:40

- Lists:
    - Indexed starting from 0
    - Order is fixed unless manually changed
    - New items added to the end
    - Multiple items with same value possible
    - Length checked using `len(list)`
    - Can contain various data types (strings, integers, booleans)
    - Example: `print(len(my_list))`
- Tuples:
    - Ordered collection of items
    - Unchangeable (immutable)
    - Written with parentheses `()`
    - Similar to lists but cannot be modified after creation
    - Example: `my_tuple = ('apple', 'banana', 'orange')`
    - Indexing starts from 0

12:12

- Key concepts: Tuples, Sets, Dictionaries
- Tuple creation:
    - Example: `my_tuple = ("apple",)`
    - Note: A trailing comma is required to denote a tuple with a single element.
- Set creation:
    - Example: `my_set = {"apple", "banana", "orange"}`
    - Note: Sets are unordered and cannot contain duplicate values.
- Dictionary creation:
    - Example: `my_dict = {"key1": "value1", "key2": "value2"}`

13:43

- Curly brackets used to define a dictionary
- Keys and values within dictionary (e.g., `brand: "forward", model: "focused", year: 2010`)
- Dictionary items are ordered in Python 3.7+
- Dictionaries are mutable and do not allow duplicates
- Print dictionary items by referencing keys (e.g., `print(dictionary["brand"])`)
- Conditional statements using `if` and `else`
- Example conditional statement: `if B > A: print("B is greater than A") else: print("A is greater than B")`
- Importance of indentation in Python (`if` block requires consistent indentation)