week1 EXERCISE

January 11, 2025

1 End of week 1 exercise

To demonstrate your familiarity with OpenAI API, and also Ollama, build a tool that takes a technical question,
and responds with an explanation. This is a tool that you will be able to use yourself during the course!

[1]:
```
# Imports
import ollama
import os
from dotenv import load_dotenv
from IPython.display import Markdown, display, update_display
from openai import OpenAI
```

[2]:
```
# Constants
MODEL_GPT = 'gpt-4o-mini'
MODEL_LLAMA = 'llama3.2'
```

[3]:
```
# Set up environment
load_dotenv()
```

[3]:
```
True
```

[4]:
```
# Clients
openai = OpenAI()
```

[5]:
```
# Question (Change this to whatever is on your mind)
question = """
Please explain what this code does and why:
yield from {book.get("author") for book in books if book.get("author")}
"""
```

[6]:
```
# Prompts
system_prompt = "You are an expert that helps customers with their technical␣
 ↪questions. \
You are given a question to which you must respond with an easy to understand␣
 ↪explanation, \
without requiring previous knowledge on the topic from the customer. \
```

```
Respond in markdown."

user_prompt = question
```

[7]:
```
# Messages
messages = [
    {"role": "system", "content": system_prompt},
    {"role": "user", "content": user_prompt}
]
```

[8]:
```
# Payloads
payload_gpt = {
    "model": MODEL_GPT,
    "messages": messages,
    "stream": True
}

payload_llama = {
    "model": MODEL_LLAMA,
    "messages": messages
}
```

[9]:
```
# Get gpt-4o-mini to answer, with streaming
stream = openai.chat.completions.create(**payload_gpt)

response = ""
display_handle = display(Markdown(""), display_id=True)
for chunk in stream:
    response += chunk.choices[0].delta.content or ''
    response = response.replace("```","").replace("markdown", "")
    update_display(Markdown(response), display_id=display_handle.display_id)
```

Sure! Let's break down the code you provided:

1.0.1 What the Code Does

1. **`{... for book in books if book.get("author")}`**:
   - This part is creating a **set** (denoted by the curly braces `{}`) that contains the unique authors from a list of `books`.
   - `books` is expected to be a list (or another iterable) where each `book` is likely a dictionary.
   - `book.get("author")` is used to access the value associated with the key `"author"` in each `book`. If the `book` has an author, it returns their name; if not, it returns `None`.
   - The `if book.get("author")` condition ensures that only books with a valid author are included in the set.
2. **`yield from`**:
   - This keyword is used in generator functions in Python. It allows you to yield values from another iterator (in this case, the set of authors).
   - When the generator reaches `yield from`, it will yield all the items from the set one by

one.

1.0.2 Why the Code is Written This Way

- **Unique Authors**: By using a set, the code automatically filters out duplicate authors. Even if several books have the same author, they will appear only once in the final result.
- **Efficiency**: Using `book.get("author")` instead of `book["author"]` prevents errors if an author does not exist for a book. If you tried to access a non-existent key directly, the code would raise a KeyError.
- **Generator**: By using `yield from`, this code can efficiently generate a list of authors without needing to create and return a whole list at once. This is useful for saving memory if you are processing a large number of books.

1.0.3 Example

Suppose you have the following list of books:

python books = [{“title”: “Book 1”, “author”: “Author A”}, {“title”: “Book 2”, “author”: “Author B”}, {“title”: “Book 3”, “author”: “Author A”}, {“title”: “Book 4”}, # No author]

When you run your code, it will create a set containing `{"Author A", "Author B"}`, and `yield from` will then allow you to iterate over these authors. Each time you ask for the next author, you will get “Author A” and then “Author B”.

1.0.4 Conclusion

In summary, the code efficiently collects unique authors from a list of books while handling missing authors gracefully, and it uses a generator to yield these authors one at a time. This makes it both robust and memory-efficient!

[10]:
```
# Get Llama 3.2 to answer, without streaming
response = ollama.chat(**payload_llama)
```

[11]:
```
display(Markdown(response['message']['content']))
```

2 Unpacking and Yielding Data

The given code snippet is written in Python and utilizes the `yield from` keyword, which was introduced in Python 3.3.

```
# Assume we have two dictionaries: 'book' and 'books'
# book = {"author": "John Doe", ...}
# books = [{"author": "Jane Doe", ...}, {"author": "Bob Smith", ...}]
```

Let’s break down what the code does:

2.0.1 `for` Loop with `yield from`

The expression `{book.get("author") for book in books if book.get("author")}` is a **generator expression**. It generates a sequence of values on-the-fly, without storing them in memory.

The `for` loop iterates over each dictionary in the `books` list. The `if` condition filters out any dictionaries that don't have an "author" key.

2.0.2 `yield from`

The `yield from` keyword is used to delegate a sub-generator's iteration to the current generator's frame. In this case, it allows us to yield values from the generator expression directly without having to use another loop or function.

Here's how it works:

1. The outer generator (the one using `for`) yields each dictionary in the `books` list.
2. For each dictionary, the inner generator (`book.get("author") for book in ...`) generates a sequence of values (in this case, just the author name).
3. The `yield from` keyword delegates these generated values to the outer generator.

Result

The resulting iterator yields the authors' names one by one, filtered by the presence of an "author" key in each dictionary.

2.0.3 Example Use Case

```
books = [{"author": "Jane Doe", ...}, {"author": "Bob Smith", ...}]
authors = yield from {book.get("author") for book in books if book.get("author")}
print(authors)  # Output: ['Jane Doe', 'Bob Smith']
```

This code can be useful when working with large datasets and you need to process specific data points (in this case, author names).