Report

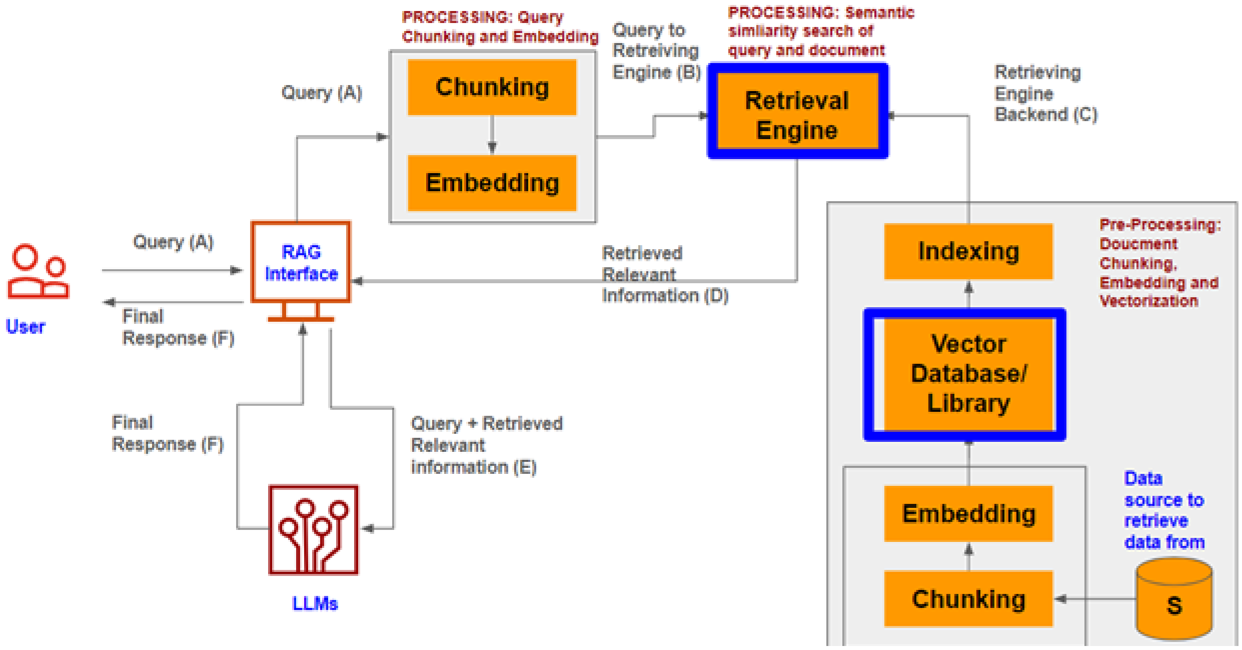
Overview:

- Combine Retrieval-Augmented Generation (RAG) with a vector database.
- Use GPT API for real-time queries.
- Store and retrieve data from a vector database.

Technologies:

- Frontend: Streamlit
- Backend: Python, FastAPI.
- Vector Database: Weaviate
- LLM: GPT 4.o (Azure OpenAI)

Approach:


PROCESSING: Query Chunking and Embedding
Query to Retreiving Engine (B)
PROCESSING: Semantic simliarity search of query and document
Query (A)
Chunking
Embedding
Retrieval Engine
Retrieving Engine Backend (C)
Query (A)
RAG Interface
User
Final Response (F)
Retrieved Relevant Information (D)
Indexing
Pre-Processing: Doucment Chunking, Embedding and Vectorization
Vector Database/ Library
Final Response (F)
Query + Retrieved Relevant information (E)
LLMs
Embedding
Data source to retrieve data from
Chunking
S

1. **User Query (A)**:

   - The user sends a query to the system.

2. **RAG Interface**:

   - The query (A) is received by the RAG Interface.

3. **Processing: Query Chunking and Embedding**:

   - The query is chunked and embedded to prepare it for the retrieval process.

4. **Retrieval Engine**:

   - The embedded query is sent to the Retrieval Engine for semantic similarity search against the indexed documents.

5. **Retrieving Engine Backend (C)**:

   - The Retrieval Engine queries the backend, which includes a Vector Database/Library.
   - The backend is pre-processed with chunking, embedding, and vectorization of documents.

6. **Vector Database/Library**:

   - This database stores the embedded documents and indexes them for efficient retrieval.

7. **Indexing**:

   - Documents are indexed to facilitate quick search and retrieval.

8. **Retrieval Process**:

   - The Retrieval Engine retrieves the relevant information (D) based on the similarity search.

9. **Response Compilation**:

   - The RAG Interface combines the user query (A) and the retrieved relevant information (E).

10. **LLMs (Large Language Models)**:

- The combined query and retrieved information are processed by LLMs to generate a final response (F).

11. **Final Response**:

- The final response (F) is sent back to the user.

The pipeline utilizes both traditional information retrieval techniques and advanced language models to provide accurate and contextually relevant responses to user queries.

Challenges:

No challenges were faced. Everything was clear and straight forward

Prompt Outputs

Creativity:

Extended Documentation Features:

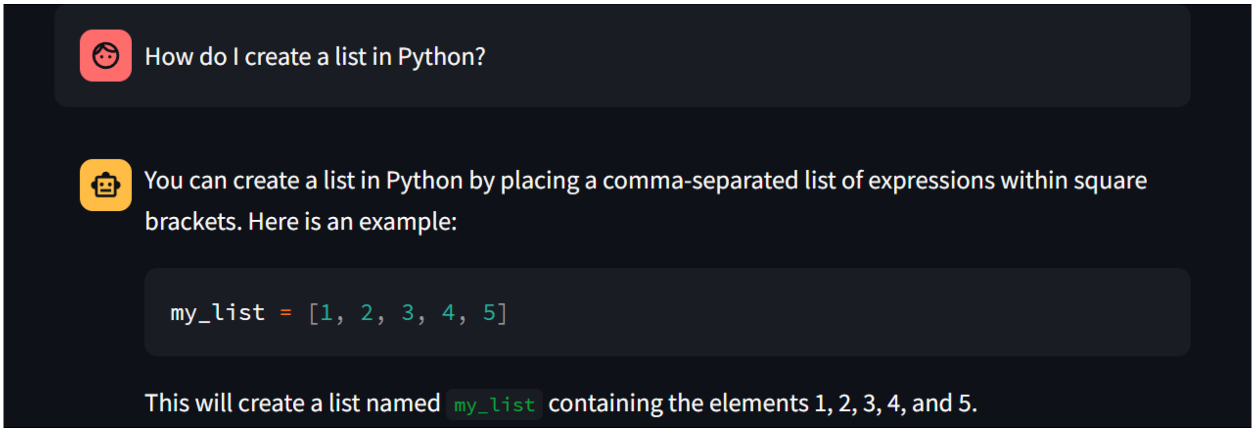
Encourage the bot to extend its responses to include more features from the Python documentation. Examples include detailed explanations of functions, usage examples, related modules, best practices, and potential pitfalls.

Nuanced Understanding:

The bot's ability to understand and process nuanced programming queries. This includes recognizing complex code requirements and constraints, such as specific libraries, performance considerations, or compatibility issues. Examples include:

Simple Query Examples:

- "How do I create a list in Python?"



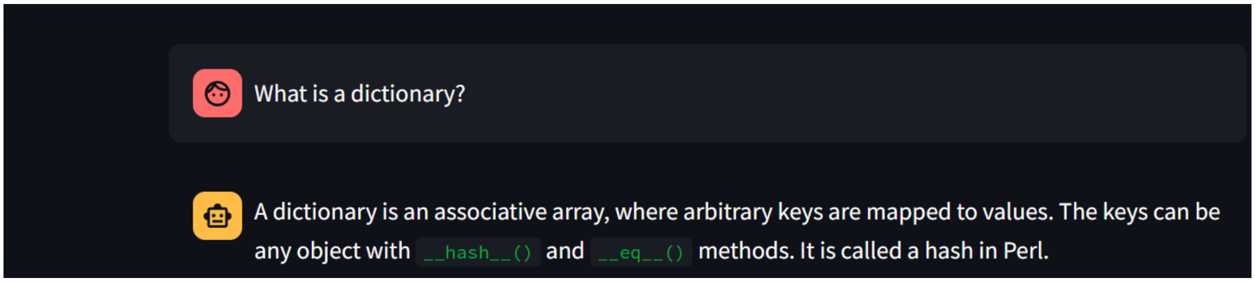
How do I create a list in Python?

You can create a list in Python by placing a comma-separated list of expressions within square brackets. Here is an example:

```
my_list = [1, 2, 3, 4, 5]
```

This will create a list named `my_list` containing the elements 1, 2, 3, 4, and 5.

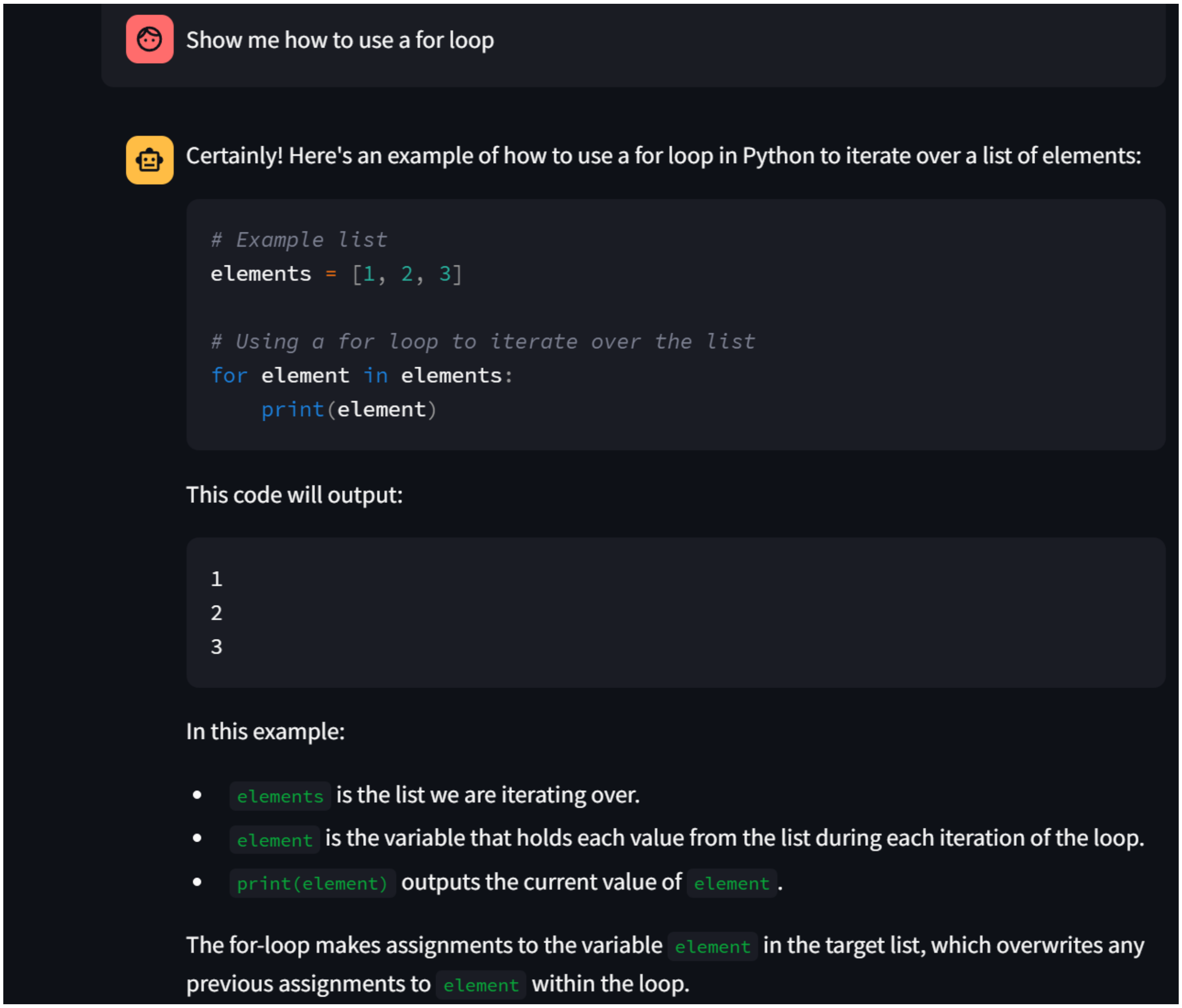
- "What is a dictionary?"



What is a dictionary?

A dictionary is an associative array, where arbitrary keys are mapped to values. The keys can be any object with `__hash__()` and `__eq__()` methods. It is called a hash in Perl.

- "Show me how to use a for loop."



Show me how to use a for loop

Certainly! Here's an example of how to use a for loop in Python to iterate over a list of elements:

```
# Example list
elements = [1, 2, 3]

# Using a for loop to iterate over the list
for element in elements:
    print(element)
```

This code will output:

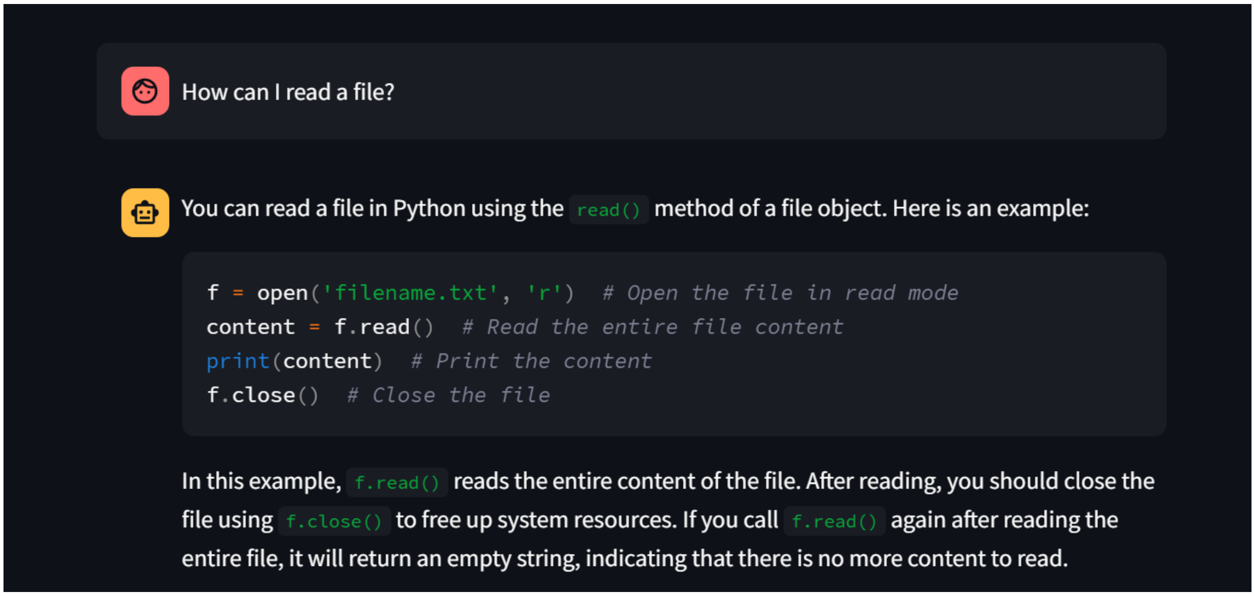
```
1
2
3
```

In this example:

- `elements` is the list we are iterating over.
- `element` is the variable that holds each value from the list during each iteration of the loop.
- `print(element)` outputs the current value of `element`.

The for-loop makes assignments to the variable `element` in the target list, which overwrites any previous assignments to `element` within the loop.

- "How can I read a file?"



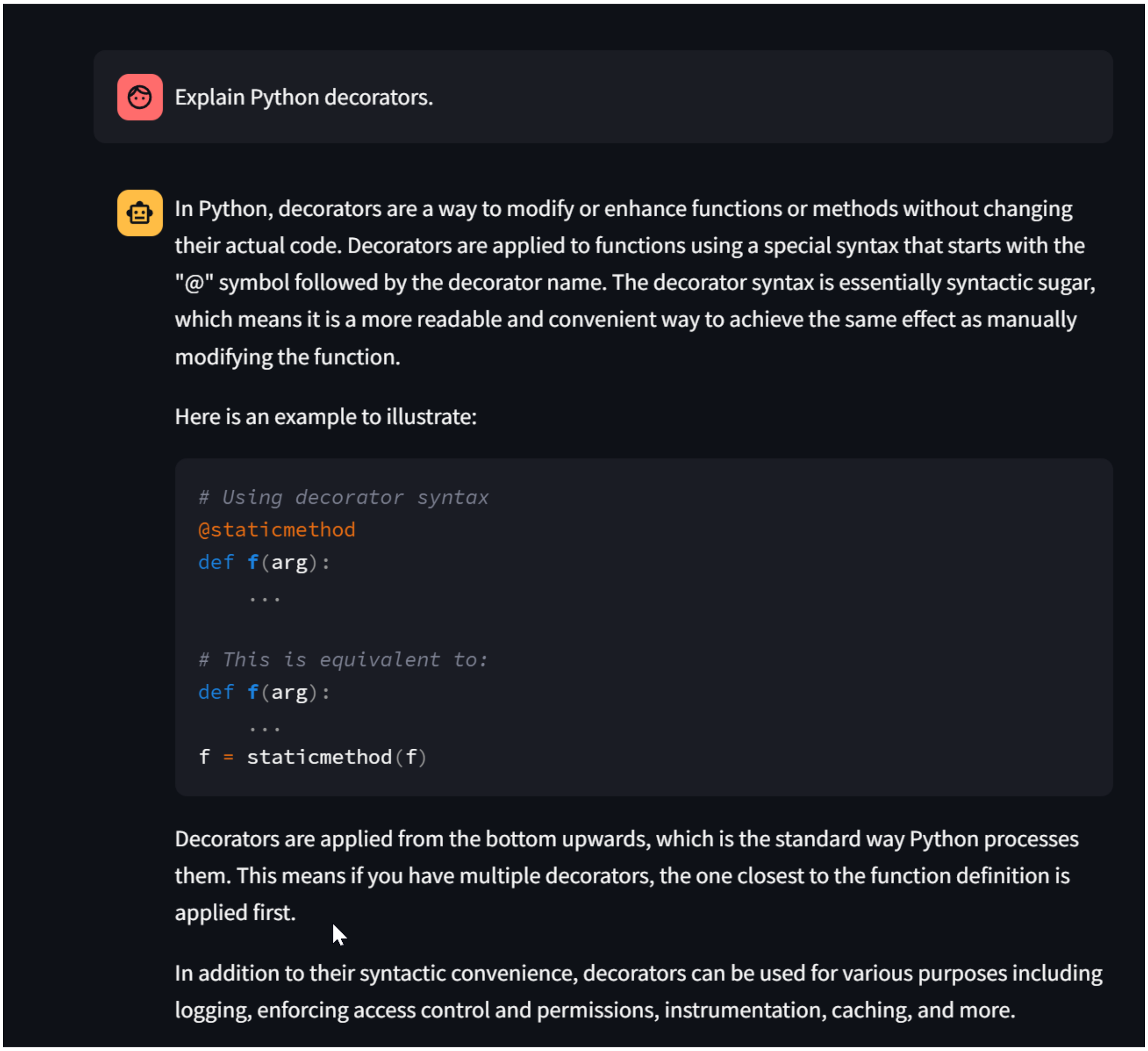
How can I read a file?

You can read a file in Python using the `read()` method of a file object. Here is an example:

```
f = open('filename.txt', 'r')  # Open the file in read mode
content = f.read()  # Read the entire file content
print(content)  # Print the content
f.close()  # Close the file
```

In this example, `f.read()` reads the entire content of the file. After reading, you should close the file using `f.close()` to free up system resources. If you call `f.read()` again after reading the entire file, it will return an empty string, indicating that there is no more content to read.

- "Explain Python decorators."



Explain Python decorators.

In Python, decorators are a way to modify or enhance functions or methods without changing their actual code. Decorators are applied to functions using a special syntax that starts with the "@" symbol followed by the decorator name. The decorator syntax is essentially syntactic sugar, which means it is a more readable and convenient way to achieve the same effect as manually modifying the function.

Here is an example to illustrate:

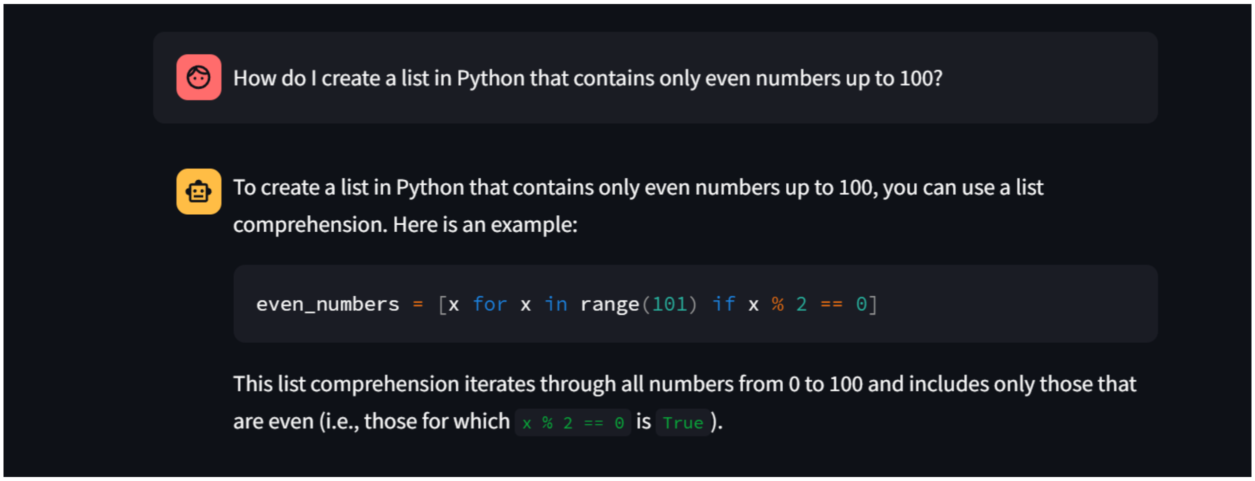
```
# Using decorator syntax
@staticmethod
def f(arg):
    ...

# This is equivalent to:
def f(arg):
    ...
f = staticmethod(f)
```

Decorators are applied from the bottom upwards, which is the standard way Python processes them. This means if you have multiple decorators, the one closest to the function definition is applied first.

In addition to their syntactic convenience, decorators can be used for various purposes including logging, enforcing access control and permissions, instrumentation, caching, and more.

Nuanced Query Examples:

- "How do I create a list in Python that contains only even numbers up to 100?"



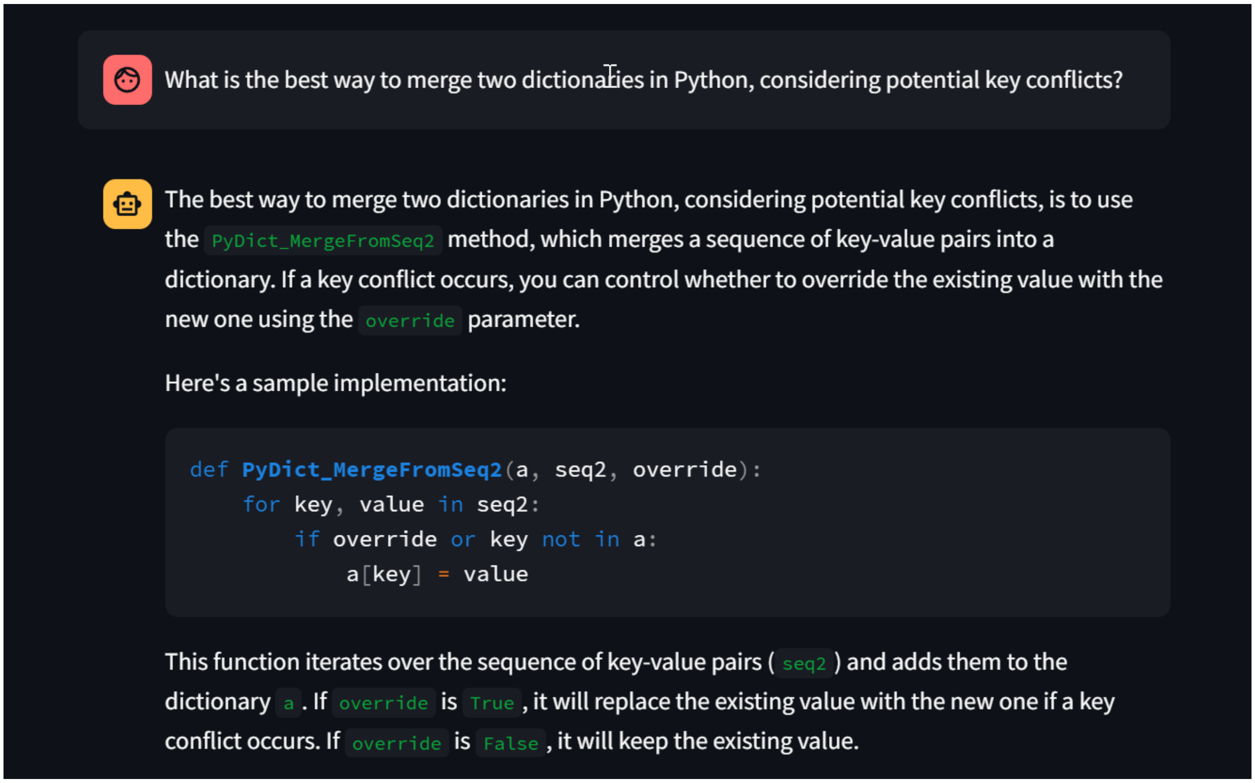
How do I create a list in Python that contains only even numbers up to 100?

To create a list in Python that contains only even numbers up to 100, you can use a list comprehension. Here is an example:

```
even_numbers = [x for x in range(101) if x % 2 == 0]
```

This list comprehension iterates through all numbers from 0 to 100 and includes only those that are even (i.e., those for which `x % 2 == 0` is `True`).

- "What is the best way to merge two dictionaries in Python, considering potential key conflicts?"



What is the best way to merge two dictionaries in Python, considering potential key conflicts?

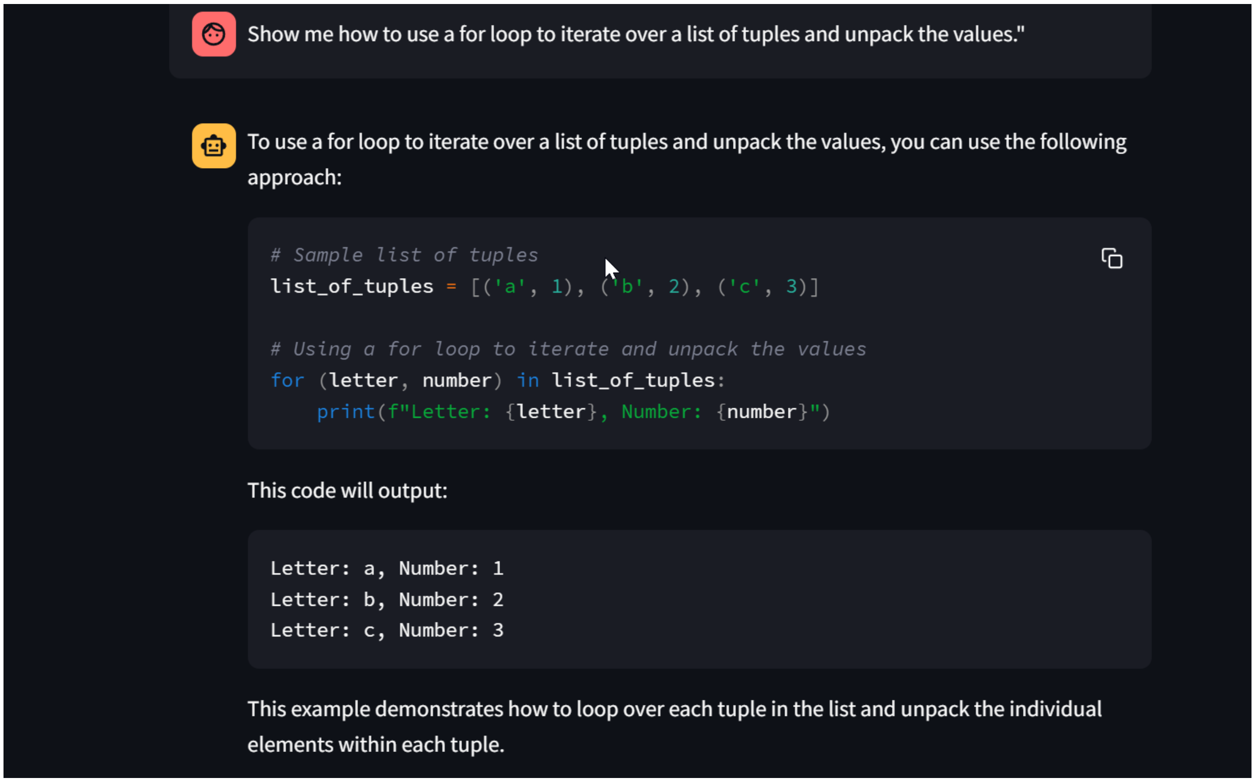
The best way to merge two dictionaries in Python, considering potential key conflicts, is to use the `PyDict_MergeFromSeq2` method, which merges a sequence of key-value pairs into a dictionary. If a key conflict occurs, you can control whether to override the existing value with the new one using the `override` parameter.

Here's a sample implementation:

```
def PyDict_MergeFromSeq2(a, seq2, override):
    for key, value in seq2:
        if override or key not in a:
            a[key] = value
```

This function iterates over the sequence of key-value pairs (`seq2`) and adds them to the dictionary `a` . If `override` is `True` , it will replace the existing value with the new one if a key conflict occurs. If `override` is `False` , it will keep the existing value.

- "Show me how to use a for loop to iterate over a list of tuples and unpack the values."



Show me how to use a for loop to iterate over a list of tuples and unpack the values."

To use a for loop to iterate over a list of tuples and unpack the values, you can use the following approach:

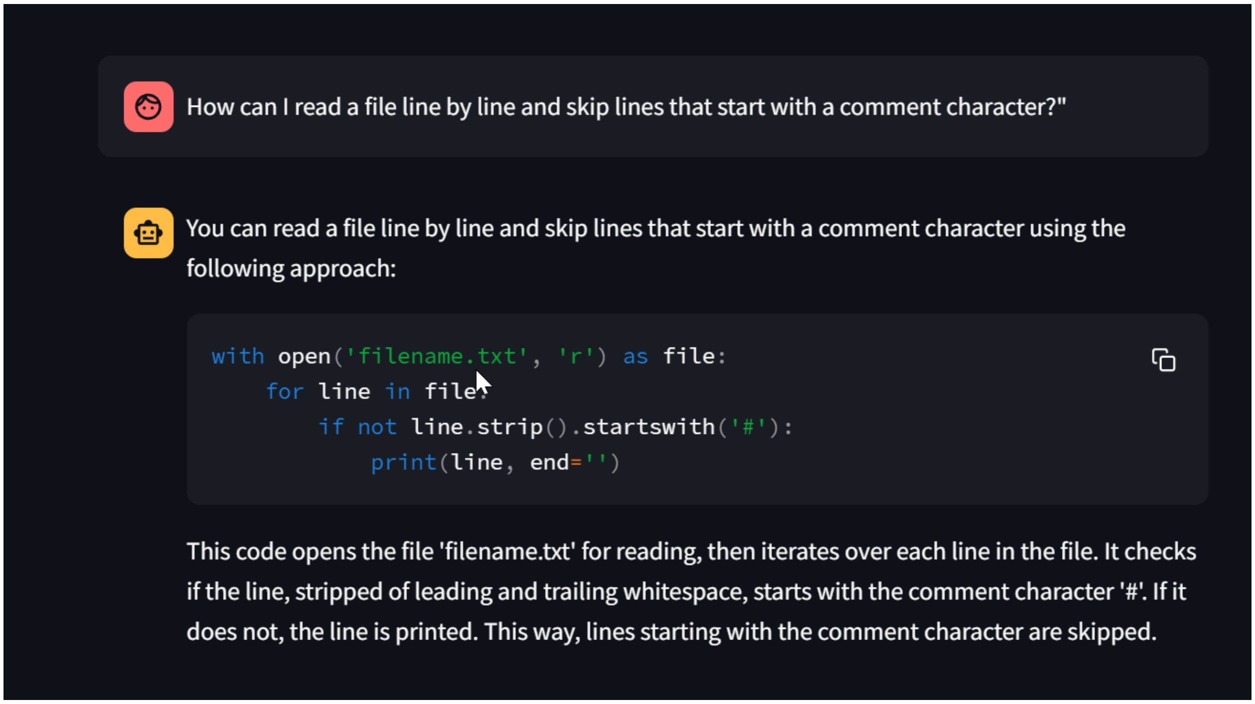
```
# Sample list of tuples
list_of_tuples = [('a', 1), ('b', 2), ('c', 3)]

# Using a for loop to iterate and unpack the values
for (letter, number) in list_of_tuples:
    print(f"Letter: {letter}, Number: {number}")
```

This code will output:

```
Letter: a, Number: 1
Letter: b, Number: 2
Letter: c, Number: 3
```

This example demonstrates how to loop over each tuple in the list and unpack the individual elements within each tuple.

- "How can I read a file line by line and skip lines that start with a comment character?"



How can I read a file line by line and skip lines that start with a comment character?"

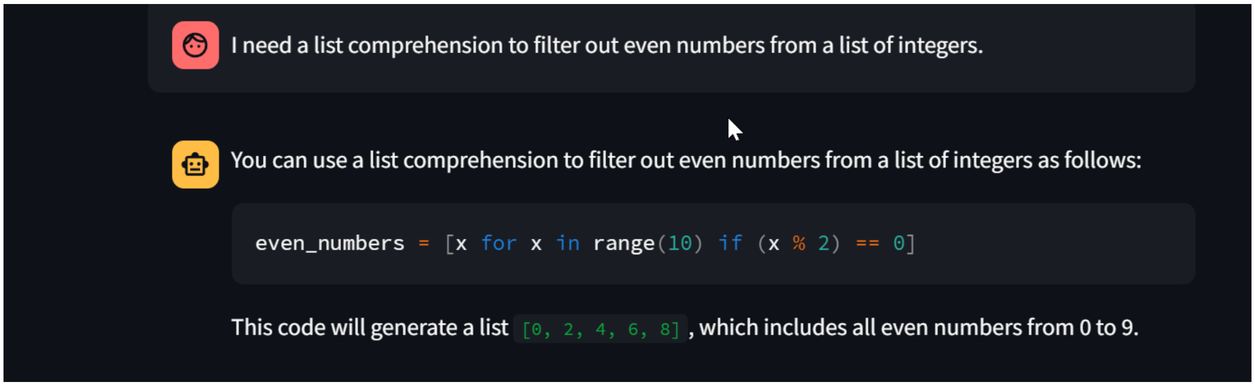
You can read a file line by line and skip lines that start with a comment character using the following approach:

```
with open('filename.txt', 'r') as file:
    for line in file:
        if not line.strip().startswith('#'):
            print(line, end='')
```

This code opens the file 'filename.txt' for reading, then iterates over each line in the file. It checks if the line, stripped of leading and trailing whitespace, starts with the comment character '#'. If it does not, the line is printed. This way, lines starting with the comment character are skipped.

Sensible and Accurate Responses (10%):
The bot should provide responses that are sensible in real-life programming scenarios based on the constraints provided by the user. Responses should adhere to constraints like:

- "I need a list comprehension to filter out even numbers from a list of integers."



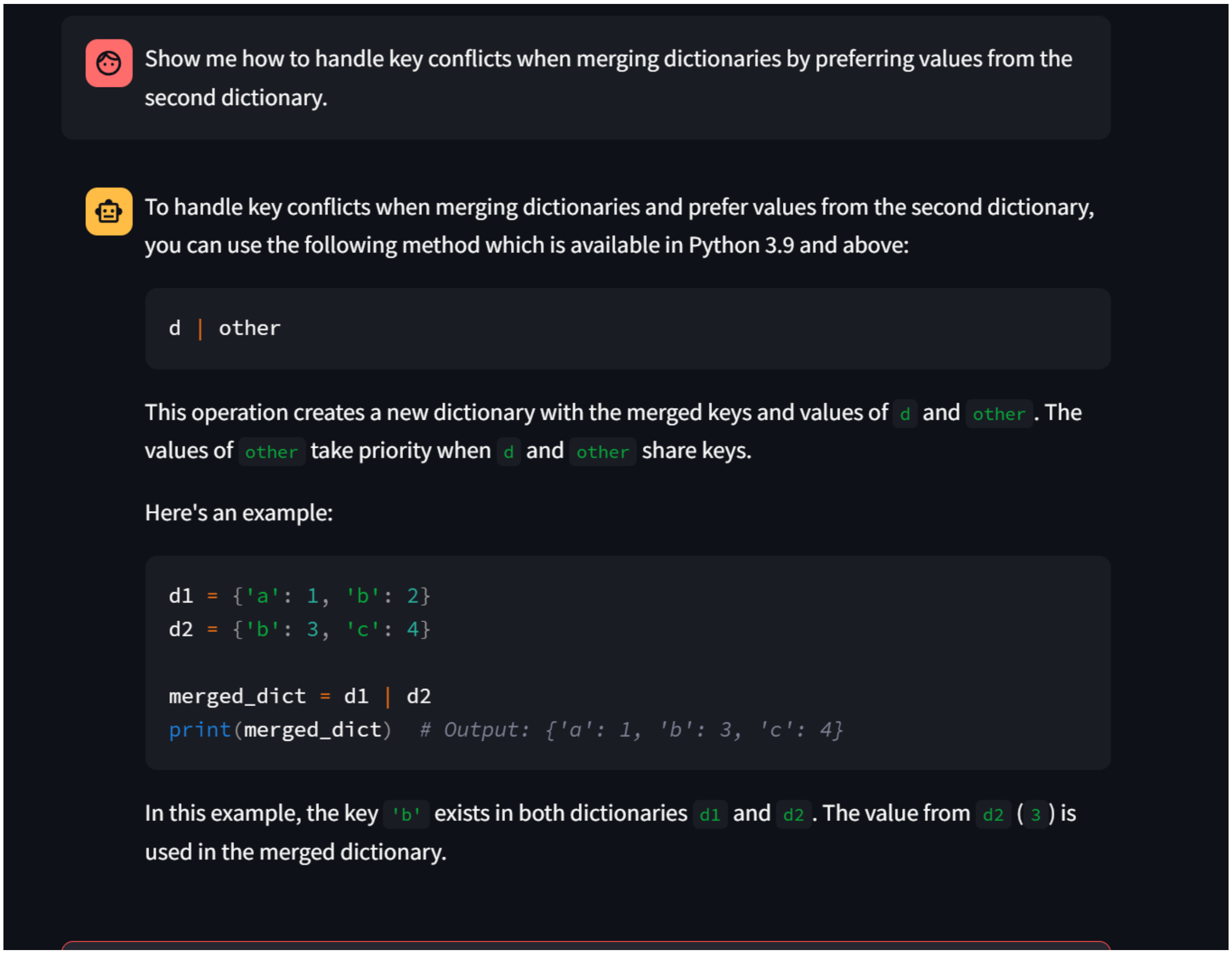
I need a list comprehension to filter out even numbers from a list of integers.

You can use a list comprehension to filter out even numbers from a list of integers as follows:

```
even_numbers = [x for x in range(10) if (x % 2) == 0]
```

This code will generate a list `[0, 2, 4, 6, 8]`, which includes all even numbers from 0 to 9.

- "Show me how to handle key conflicts when merging dictionaries by preferring values from the second dictionary."



Show me how to handle key conflicts when merging dictionaries by preferring values from the second dictionary.

To handle key conflicts when merging dictionaries and prefer values from the second dictionary, you can use the following method which is available in Python 3.9 and above:

```
d | other
```

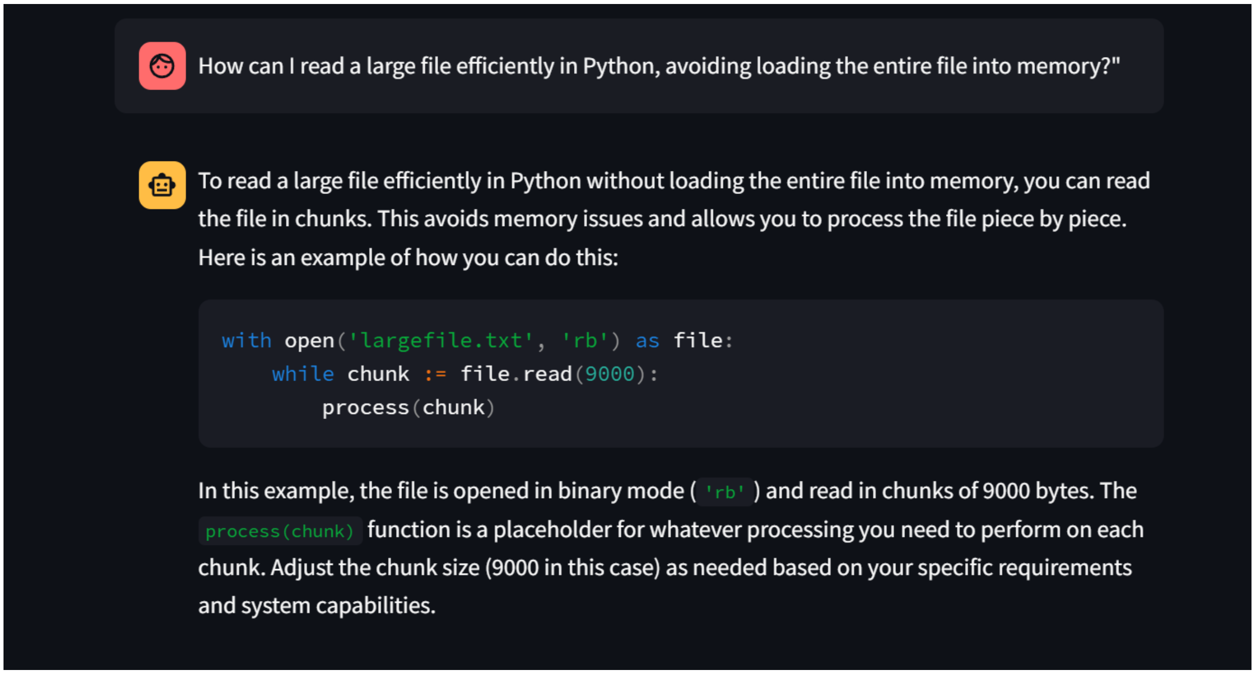
This operation creates a new dictionary with the merged keys and values of `d` and `other`. The values of `other` take priority when `d` and `other` share keys.

Here's an example:

```
d1 = {'a': 1, 'b': 2}
d2 = {'b': 3, 'c': 4}

merged_dict = d1 | d2
print(merged_dict)  # Output: {'a': 1, 'b': 3, 'c': 4}
```

In this example, the key `'b'` exists in both dictionaries `d1` and `d2`. The value from `d2` (`3`) is used in the merged dictionary.

- "How can I read a large file efficiently in Python, avoiding loading the entire file into memory?"



How can I read a large file efficiently in Python, avoiding loading the entire file into memory?"

To read a large file efficiently in Python without loading the entire file into memory, you can read the file in chunks. This avoids memory issues and allows you to process the file piece by piece. Here is an example of how you can do this:

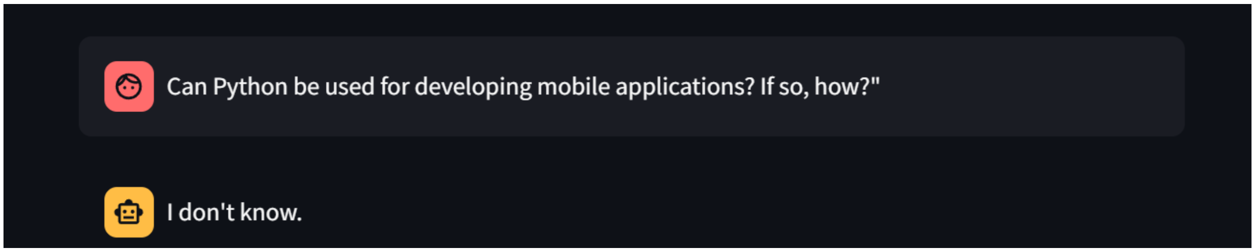
```
with open('largefile.txt', 'rb') as file:
    while chunk := file.read(9000):
        process(chunk)
```

In this example, the file is opened in binary mode (`'rb'`) and read in chunks of 9000 bytes. The `process(chunk)` function is a placeholder for whatever processing you need to perform on each chunk. Adjust the chunk size (9000 in this case) as needed based on your specific requirements and system capabilities.

Out-of-Context Python Questions:

The bot's ability to handle out-of-context questions that go beyond the standard Python documentation. Examples include:

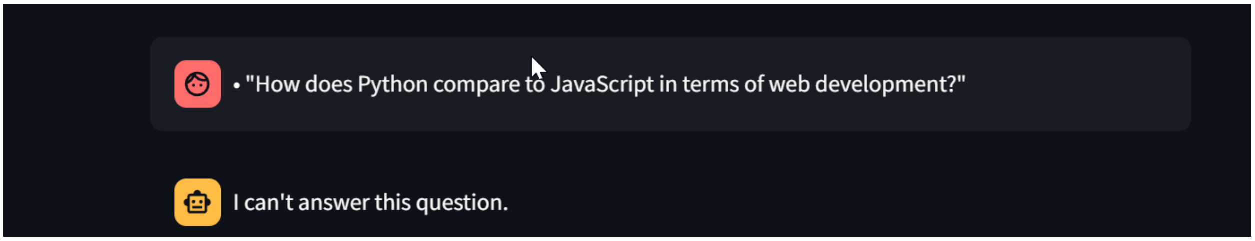
- "Can Python be used for developing mobile applications? If so, how?"



Can Python be used for developing mobile applications? If so, how?"

I don't know.

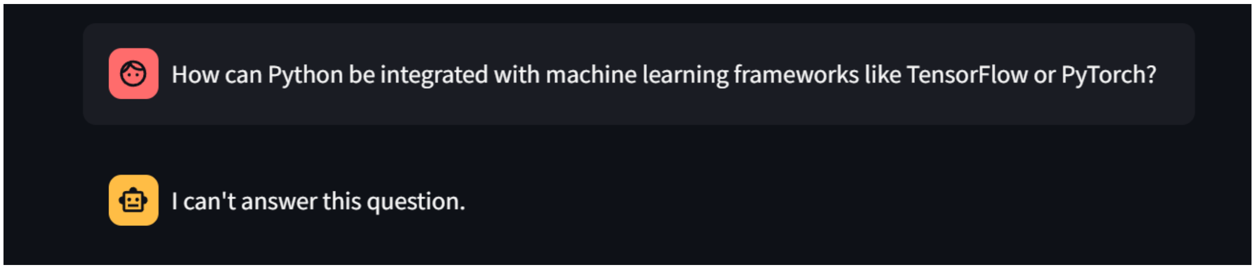
- "How does Python compare to JavaScript in terms of web development?"



• "How does Python compare to JavaScript in terms of web development?"

I can't answer this question.

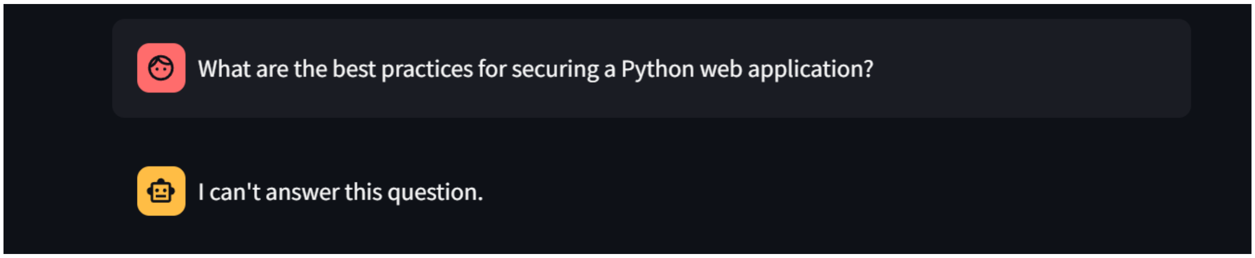
- "How can Python be integrated with machine learning frameworks like TensorFlow or PyTorch?"



How can Python be integrated with machine learning frameworks like TensorFlow or PyTorch?

I can't answer this question.

- "What are the best practices for securing a Python web application?"



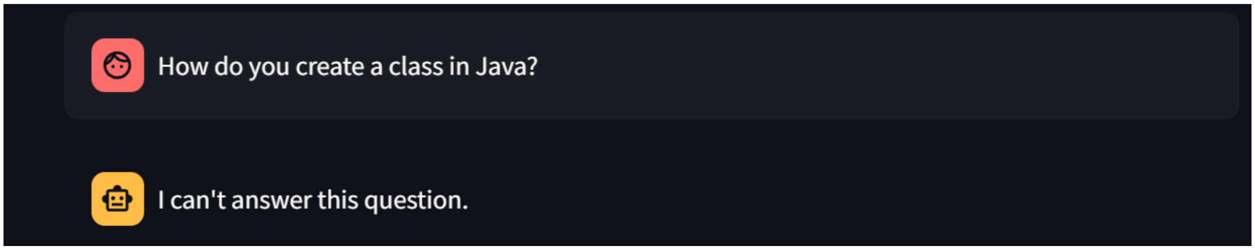
What are the best practices for securing a Python web application?

I can't answer this question.

Irrelevant Question Testing Related to Other Programming Languages:

The bot's ability to recognize and address questions related to other programming languages, even if they are not directly related to Python. Examples include:

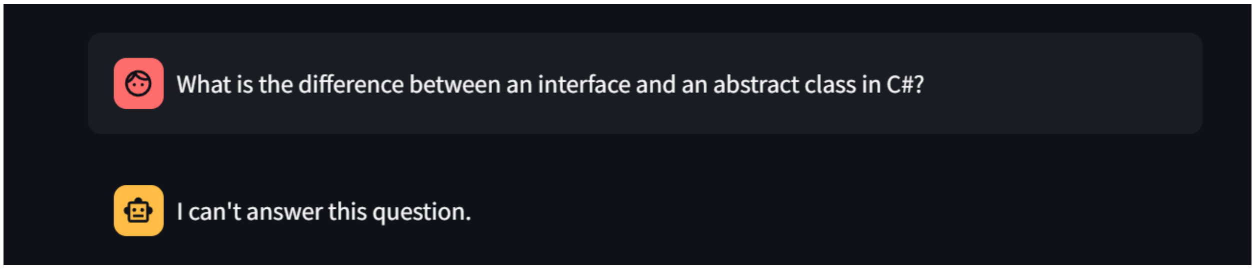
- "How do you create a class in Java?"



How do you create a class in Java?

I can't answer this question.

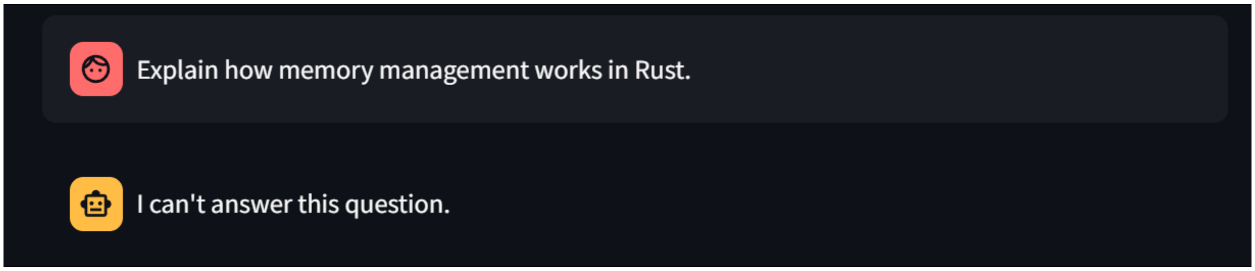
- "What is the difference between an interface and an abstract class in C#?"



What is the difference between an interface and an abstract class in C#?

I can't answer this question.

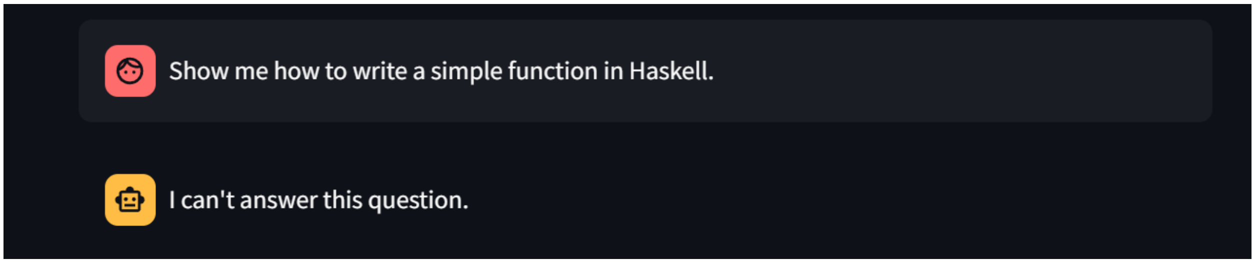
- "Explain how memory management works in Rust."



Explain how memory management works in Rust.

I can't answer this question.

- "Show me how to write a simple function in Haskell."



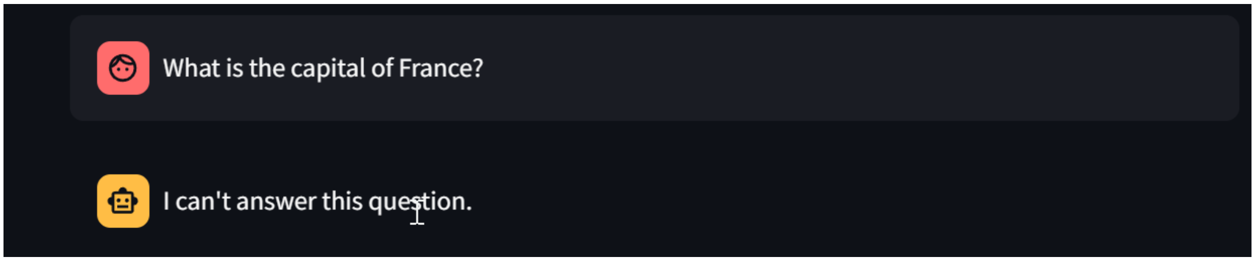
Show me how to write a simple function in Haskell.

I can't answer this question.

Totally Irrelevant Questions:

The bot's ability to recognize and appropriately handle totally irrelevant questions that are unrelated to programming. Examples include:

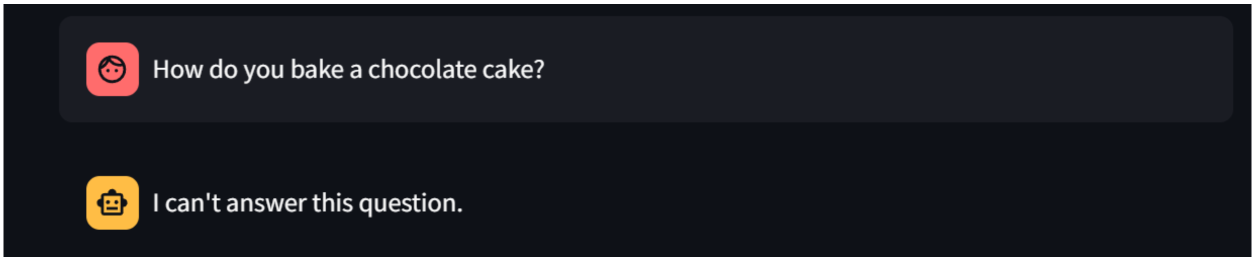
- "What is the capital of France?"



What is the capital of France?

I can't answer this question.

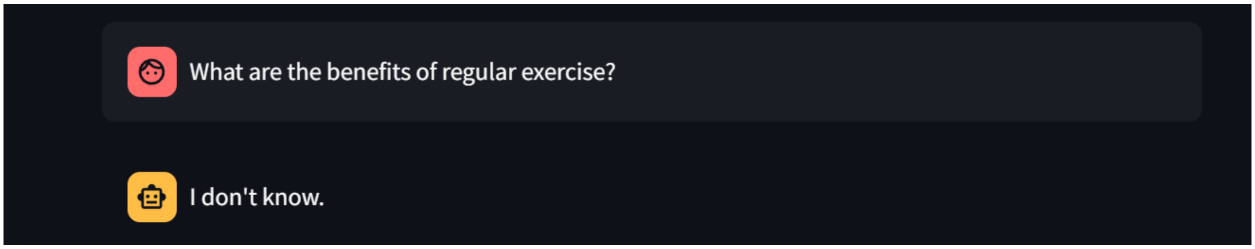
- "How do you bake a chocolate cake?"



How do you bake a chocolate cake?

I can't answer this question.

- "What are the benefits of regular exercise?"



What are the benefits of regular exercise?

I don't know.