OpenAI

DeepLearning.AI

Two Types of large language models (LLMs)

Base LLM

Predicts next word, based on text training data

Once upon a time, there was a unicorn
that lived in a magical forest with
all her unicorn friends

What is the capital of France?
What is France's largest city?
What is France's population?
What is the currency of France?

Instruction Tuned LLM

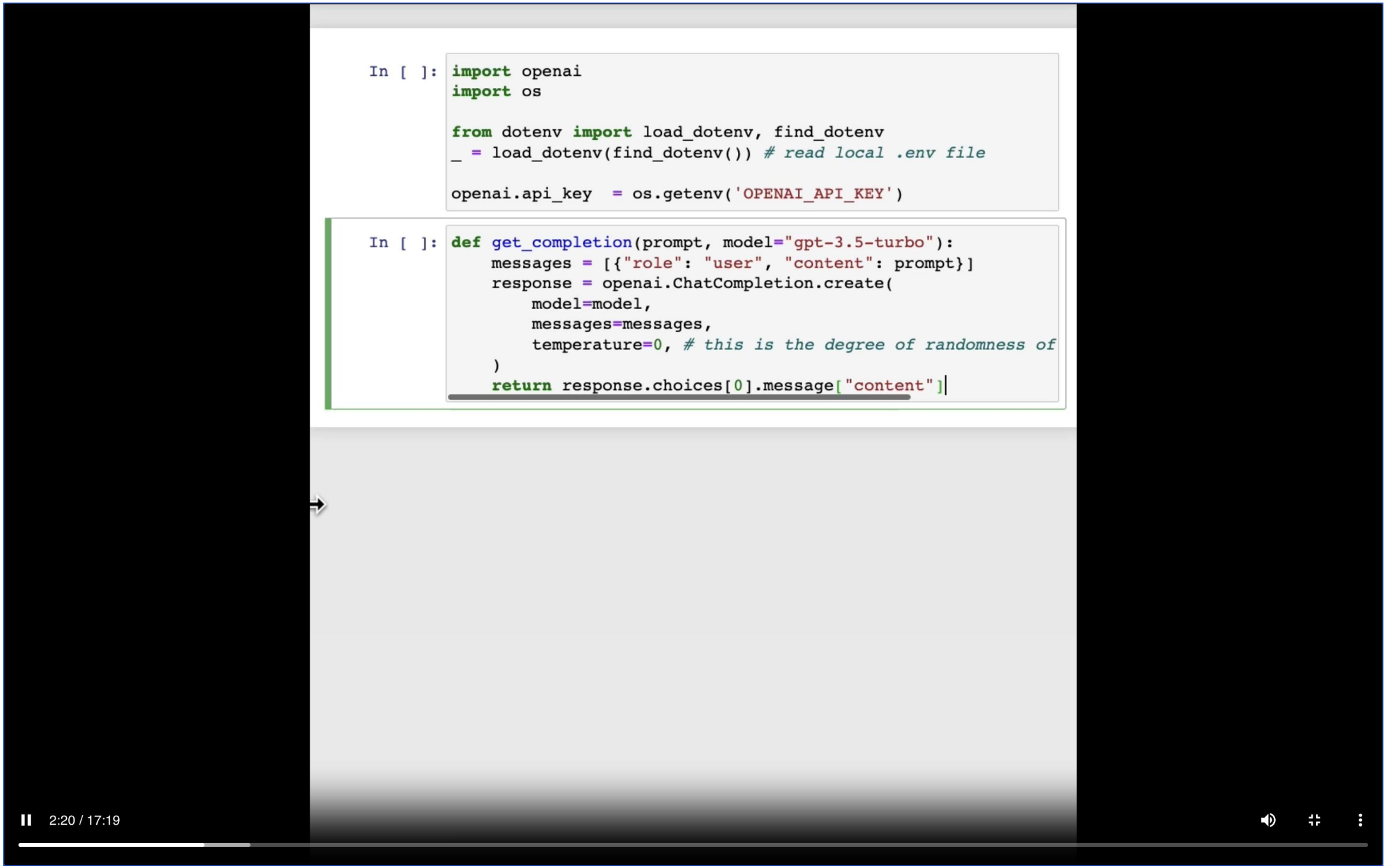
Tries to follow instructions

Fine-tune on instructions and
good attempts at following those instructions.

RLHF: Reinforcement Learning with
Human Feedback

Helpful, Honest, Harmless

What is the capital of France?
The capital of France is Paris.


In [ ]:
import openai
import os
from dotenv import load_dotenv, find_dotenv
_ = load_dotenv(find_dotenv()) # read local .env file
openai.api_key = os.getenv('OPENAI_API_KEY')
In [ ]:
def get_completion(prompt, model="gpt-3.5-turbo"):
messages = [{"role": "user", "content": prompt}]
response = openai.ChatCompletion.create(
model=model,
messages=messages,
temperature=0, # this is the degree of randomness of
)
return response.choices[0].message["content"]
2:20 / 17:19

DeepLearning.AI OpenAI

Principles of Prompting

Principle 1

- Write clear and specific instructions

  clear ≠ short

Principle 2

- Give the model time to think

```
openai.api_key  = os.getenv('OPENAI_API_KEY')
```

In [2]:
```
def get_completion(prompt, model="gpt-3.5-turbo"):
    messages = [{"role": "user", "content": prompt}]
    response = openai.ChatCompletion.create(
        model=model,
        messages=messages,
        temperature=0, # this is the degree of randomness of
    )
    return response.choices[0].message["content"]
```

In [*]:
```
text = f"""
You should express what you want a model to do by \
providing instructions that are as clear and \
specific as you can possibly make them. \
This will guide the model towards the desired output, \
and reduce the chances of receiving irrelevant \
or incorrect responses. Don't confuse writing a \
clear prompt with writing a short prompt. \
In many cases, longer prompts provide more clarity \
and context for the model, which can lead to \
more detailed and relevant outputs.
"""
prompt = f"""
Summarize the text delimited by triple backticks \
into a single sentence.
```{text}```
"""
response = get_completion(prompt)
print(response)
```

In [ ]:
```

```
```

```
openai.api_key  = os.getenv('OPENAI_API_KEY')
```

In [2]:
```
def get_completion(prompt, model="gpt-3.5-turbo"):
    messages = [{"role": "user", "content": prompt}]
    response = openai.ChatCompletion.create(
        model=model,
        messages=messages,
        temperature=0, # this is the degree of randomness of
    )
    return response.choices[0].message["content"]
```

In [3]:
```
text = f"""
You should express what you want a model to do by \
providing instructions that are as clear and \
specific as you can possibly make them. \
This will guide the model towards the desired output, \
and reduce the chances of receiving irrelevant \
or incorrect responses. Don't confuse writing a \
clear prompt with writing a short prompt. \
In many cases, longer prompts provide more clarity \
and context for the model, which can lead to \
more detailed and relevant outputs.
"""
prompt = f"""
Summarize the text delimited by triple backticks \
into a single sentence.
```{text}```
"""
response = get_completion(prompt)
print(response)
```

```
To guide a model towards the desired output and reduce the
chances of irrelevant or incorrect responses, it is import
ant to provide clear and specific instructions, which may
require longer prompts for more clarity and context.
```

In [ ]:
```

```
response = get_completion(prompt)
print(response)
```

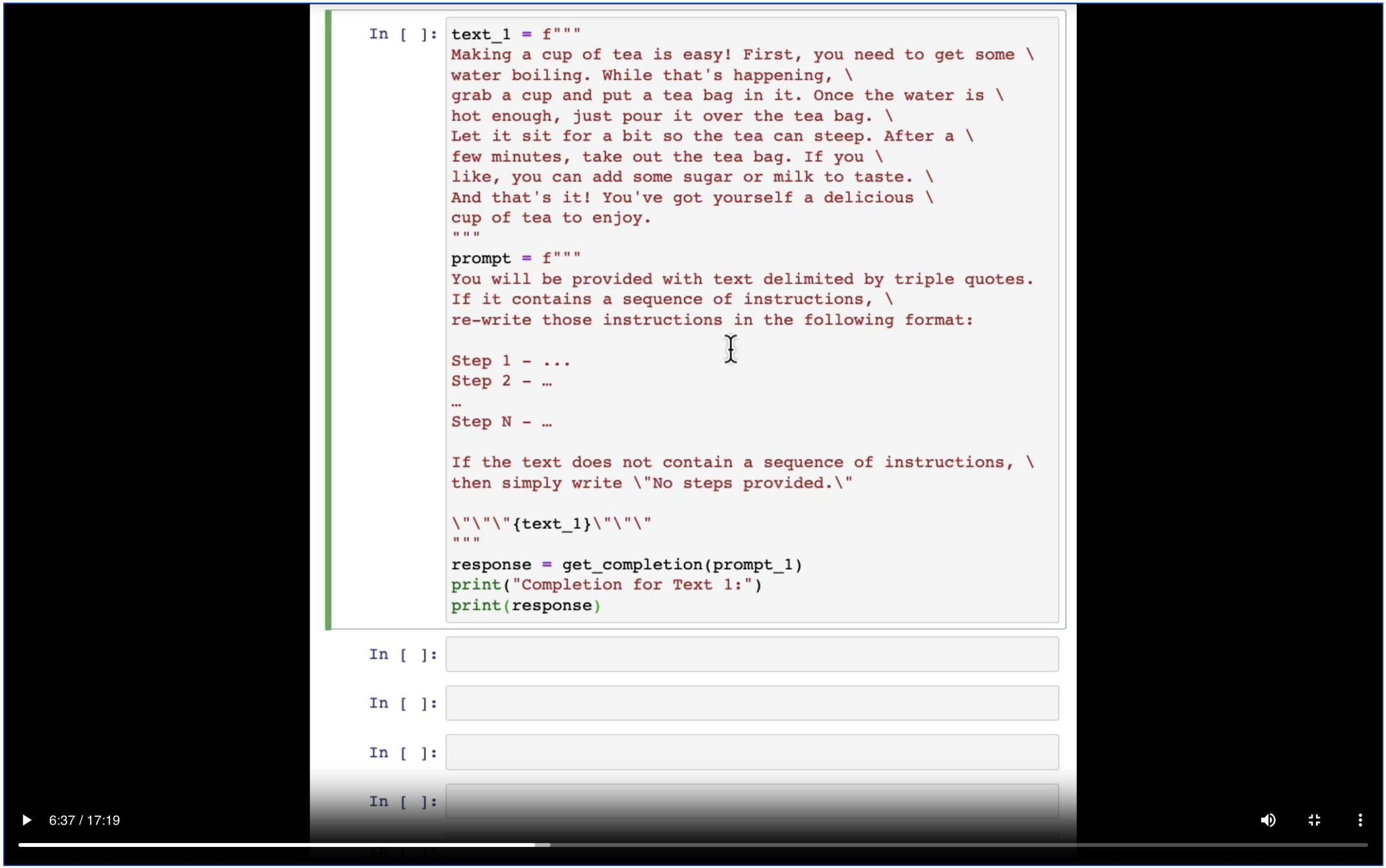
```
To guide a model towards the desired output and reduce the
chances of irrelevant or incorrect responses, it is import
ant to provide clear and specific instructions, which may
require longer prompts for more clarity and context.
```

In [4]:
```
prompt = f"""
Generate a list of three made-up book titles along \
with their authors and genres.
Provide them in JSON format with the following keys:
book_id, title, author, genre.
"""
response = get_completion(prompt)
print(response)
```

```
[
  {
    "book_id": 1,
    "title": "The Lost City of Zorath",
    "author": "Aria Blackwood",
    "genre": "Fantasy"
  },
  {
    "book_id": 2,
    "title": "The Last Survivors",
    "author": "Ethan Stone",
    "genre": "Science Fiction"
  },
  {
    "book_id": 3,
    "title": "The Secret of the Haunted Mansion",
    "author": "Lila Rose",
    "genre": "Mystery"
  }
]
```

In [ ]:

In [ ]:


In [ ]:
text_1 = f"""
Making a cup of tea is easy! First, you need to get some \
water boiling. While that's happening, \
grab a cup and put a tea bag in it. Once the water is \
hot enough, just pour it over the tea bag. \
Let it sit for a bit so the tea can steep. After a \
few minutes, take out the tea bag. If you \
like, you can add some sugar or milk to taste. \
And that's it! You've got yourself a delicious \
cup of tea to enjoy.
"""
prompt = f"""
You will be provided with text delimited by triple quotes.
If it contains a sequence of instructions, \
re-write those instructions in the following format:
Step 1 - ...
Step 2 - …
…
Step N - …
If the text does not contain a sequence of instructions, \
then simply write \"No steps provided.\"
\"\"\"{text_1}\"\"\"
"""
response = get_completion(prompt_1)
print("Completion for Text 1:")
print(response)
In [ ]:
In [ ]:
In [ ]:
In [ ]:
6:37 / 17:19

In [6]:

```
text_1 = f"""
Making a cup of tea is easy! First, you need to get some \
water boiling. While that's happening, \
grab a cup and put a tea bag in it. Once the water is \
hot enough, just pour it over the tea bag. \
Let it sit for a bit so the tea can steep. After a \
few minutes, take out the tea bag. If you \
like, you can add some sugar or milk to taste. \
And that's it! You've got yourself a delicious \
cup of tea to enjoy.
"""
prompt = f"""
You will be provided with text delimited by triple quotes.
If it contains a sequence of instructions, \
re-write those instructions in the following format:

Step 1 - ...
Step 2 - …
…
Step N - …

If the text does not contain a sequence of instructions, \
then simply write \"No steps provided.\"

\"\"\"{text_1}\"\"\"
"""
response = get_completion(prompt)
print("Completion for Text 1:")
print(response)
```

```
Completion for Text 1:
Step 1 - Get some water boiling.
Step 2 - Grab a cup and put a tea bag in it.
Step 3 - Once the water is hot enough, pour it over the te
a bag.
Step 4 - Let it sit for a bit so the tea can steep.
Step 5 - After a few minutes, take out the tea bag.
Step 6 - Add some sugar or milk to taste.
Step 7 - Enjoy your delicious cup of tea!
```

No steps provided.

In [8]:
```
prompt = f"""
Your task is to answer in a consistent style.

<child>: Teach me about patience.

<grandparent>: The river that carves the deepest \
valley flows from a modest spring; the \
grandest symphony originates from a single note; \
the most intricate tapestry begins with a solitary thread.

<child>: Teach me about resilience.
"""
response = get_completion(prompt)
print(response)
```

```
<grandparent>: Resilience is like a tree that bends with t
he wind but never breaks. It is the ability to bounce back
from adversity and keep moving forward, even when things g
et tough. Just like a tree that grows stronger with each s
torm it weathers, resilience is a quality that can be deve
loped and strengthened over time.
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

DeepLearning.AI OpenAI

Principle 1
Write clear and specific instructions

Tactic 1: Use delimiters
- Triple quotes: """
- Triple backticks: ```,
- Triple dashes: ---,
- Angle brackets: < >,
- XML tags: <tag> </tag>

Tactic 2: Ask for structured output
HTML, JSON

Tactic 3: Check whether conditions are satisfied
Check assumptions required to do the task

Tactic 4: Few-shot prompting
Give successful examples of completing tasks
Then ask model to perform the task

DeepLearning.AI OpenAI

Principles of Prompting

Principle 1

- Write clear and specific instructions

Principle 2

- Give the model time to think

```
In a charming village, siblings Jack and Jill set out on \
a quest to fetch water from a hilltop \
well. As they climbed, singing joyfully, misfortune \
struck—Jack tripped on a stone and tumbled \
down the hill, with Jill following suit. \
Though slightly battered, the pair returned home to \
comforting embraces. Despite the mishap, \
their adventurous spirits remained undimmed, and they \
continued exploring with delight.
"""
# example 1
prompt_1 = f"""
Perform the following actions:
1 - Summarize the following text delimited by triple \
backticks with 1 sentence.
2 - Translate the summary into French.
3 - List each name in the French summary.
4 - Output a json object that contains the following \
keys: french_summary, num_names.

Separate your answers with line breaks.

Text:
```{text}```
"""
response = get_completion(prompt_1)
print("Completion for prompt 1:")
print(response)
```

```
Completion for prompt 1:
Two siblings, Jack and Jill, go on a quest to fetch water
from a well on a hilltop, but misfortune strikes and they
both tumble down the hill, returning home slightly battere
d but with their adventurous spirits undimmed.

Deux frères et sœurs, Jack et Jill, partent en quête d'eau
d'un puits sur une colline, mais un malheur frappe et ils
tombent tous les deux de la colline, rentrant chez eux lég
èrement meurtris mais avec leurs esprits aventureux intact
s.
Noms: Jack, Jill.
```
```

In [11]:
```
prompt = f"""
Determine if the student's solution is correct or not.

Question:
I'm building a solar power installation and I need \
 help working out the financials.
- Land costs $100 / square foot
- I can buy solar panels for $250 / square foot
- I negotiated a contract for maintenance that will cost \
me a flat $100k per year, and an additional $10 / square \
foot
What is the total cost for the first year of operations
as a function of the number of square feet.

Student's Solution:
Let x be the size of the installation in square feet.
Costs:
1. Land cost: 100x
2. Solar panel cost: 250x
3. Maintenance cost: 100,000 + 100x
Total cost: 100x + 250x + 100,000 + 100x = 450x + 100,000
"""
response = get_completion(prompt)
print(response)
```

```
The student's solution is correct.
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

```
prompt = f"""
Your task is to determine if the student's solution \
is correct or not.
To solve the problem do the following:
- First, work out your own solution to the problem.
- Then compare your solution to the student's solution \
and evaluate if the student's solution is correct or not.
Don't decide if the student's solution is correct until
you have done the problem yourself.

Use the following format:
Question:
```
question here
```
Student's solution:
```
student's solution here
```
Actual solution:
```
steps to work out the solution and your solution here
```
Is the student's solution the same as actual solution \
just calculated:
```
yes or no
```
Student grade:
```
correct or incorrect
```


Question:
```
I'm building a solar power installation and I need help \
working out the financials.
- Land costs $100 / square foot
- I can buy solar panels for $250 / square foot
- I negotiated a contract for maintenance that will cost \
```
```

```
as a function of the number of square feet.
```
Student's solution:
```
Let x be the size of the installation in square feet.
Costs:
1. Land cost: 100x
2. Solar panel cost: 250x
3. Maintenance cost: 100,000 + 100x
Total cost: 100x + 250x + 100,000 + 100x = 450x + 100,000
```
Actual solution:
"""
response = get_completion(prompt)
print(response)
```

```
Let x be the size of the installation in square feet.

Costs:
1. Land cost: 100x
2. Solar panel cost: 250x
3. Maintenance cost: 100,000 + 10x

Total cost: 100x + 250x + 100,000 + 10x = 360x + 100,000

Is the student's solution the same as actual solution just
calculated:
No

Student grade:
Incorrect
```

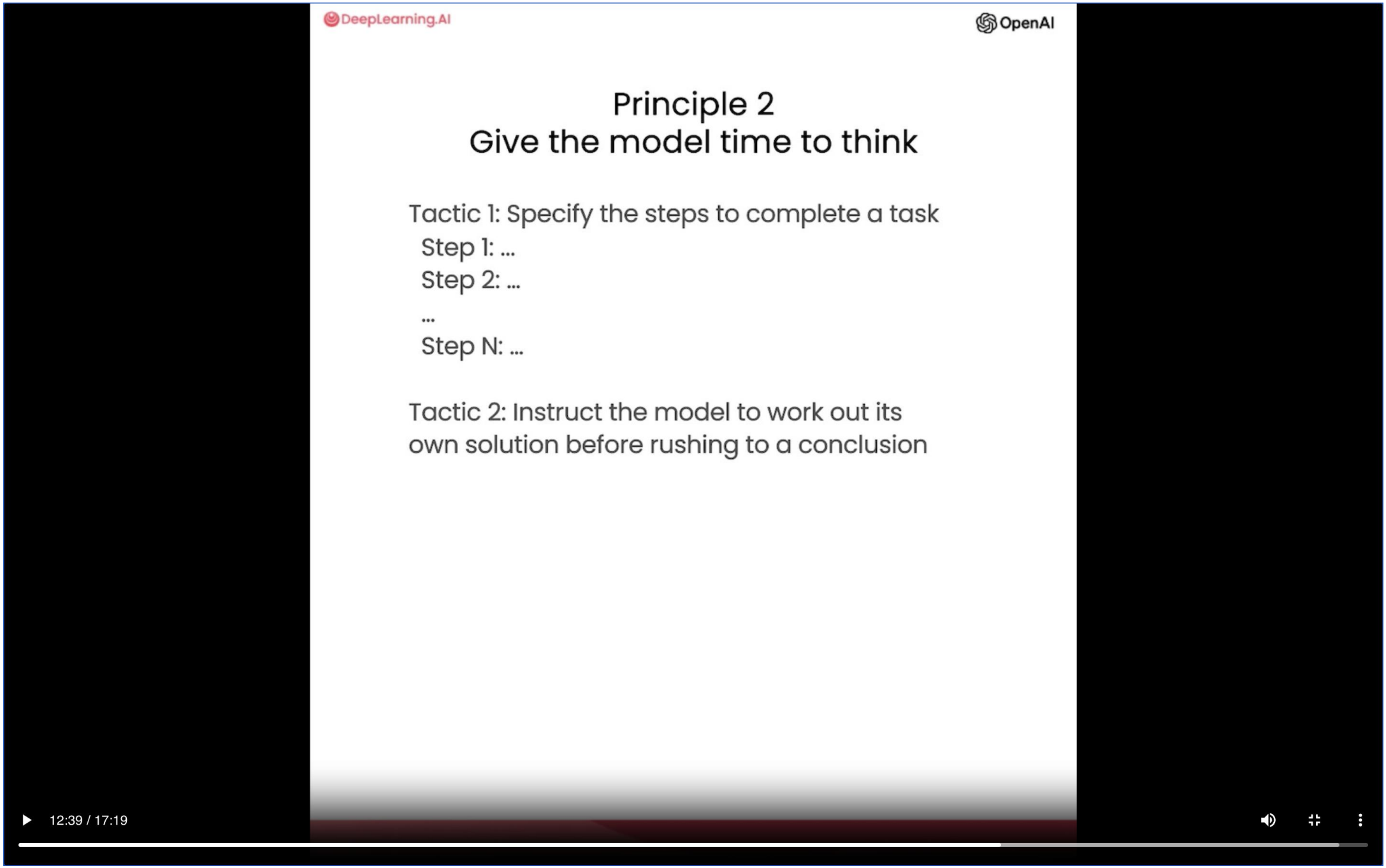
In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:
```


DeepLearning.AI
OpenAI
Principle 2
Give the model time to think
Tactic 1: Specify the steps to complete a task
Step 1: ...
Step 2: ...
...
Step N: ...
Tactic 2: Instruct the model to work out its own solution before rushing to a conclusion
12:39 / 17:19

DeepLearning.AI OpenAI

Model Limitations

Hallucination
Makes statements that sound plausible but are not true

In [13]:
```
prompt = f"""
Tell me about AeroGlide UltraSlim Smart Toothbrush by Boie
"""
response = get_completion(prompt)
print(response)
```

```
The AeroGlide UltraSlim Smart Toothbrush by Boie is a high
-tech toothbrush that uses advanced sonic technology to pr
ovide a deep and thorough clean. It features a slim and sl
eek design that makes it easy to hold and maneuver, and it
comes with a range of smart features that help you optimiz
e your brushing routine.

One of the key features of the AeroGlide UltraSlim Smart T
oothbrush is its advanced sonic technology, which uses hig
h-frequency vibrations to break up plaque and bacteria on
your teeth and gums. This technology is highly effective a
t removing even the toughest stains and buildup, leaving y
our teeth feeling clean and refreshed.

In addition to its sonic technology, the AeroGlide UltraSl
im Smart Toothbrush also comes with a range of smart featu
res that help you optimize your brushing routine. These in
clude a built-in timer that ensures you brush for the reco
mmended two minutes, as well as a pressure sensor that ale
rts you if you're brushing too hard.

Overall, the AeroGlide UltraSlim Smart Toothbrush by Boie
is a highly advanced and effective toothbrush that is perf
ect for anyone looking to take their oral hygiene to the n
ext level. With its advanced sonic technology and smart fe
atures, it provides a deep and thorough clean that leaves
your teeth feeling fresh and healthy.
```

In [ ]:

In [ ]:

In [ ]:

DeepLearning.AI OpenAI

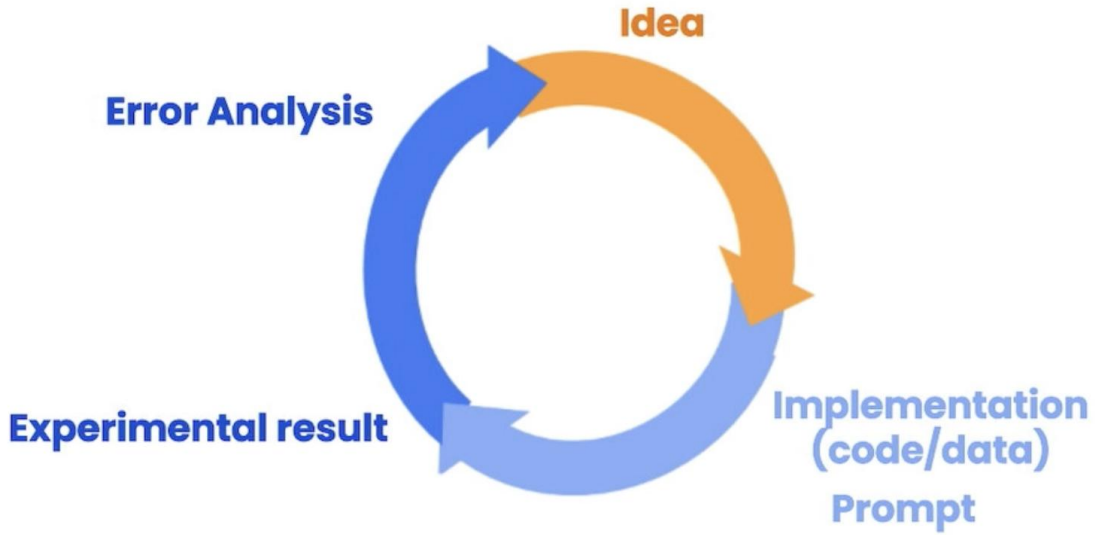
Model Limitations

Hallucination
Makes statements that sound plausible
but are not true

Reducing hallucinations:
First find relevant information,
then answer the question
based on the relevant information.

DeepLearning.AI OpenAI

Iterative Prompt Development



Prompt guidelines

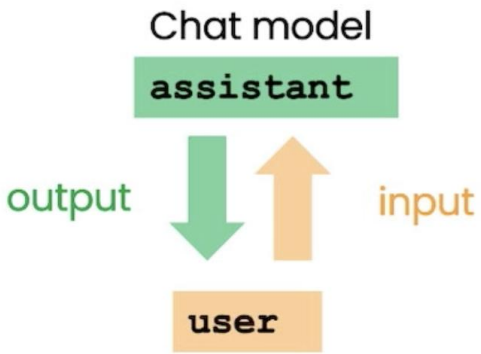
- Be clear and specific
- Analyze why result does not give desired output.
- Refine the idea and the prompt
- Repeat

DeepLearning.AI OpenAI

OpenAI API call

```
def get_completion(prompt,
                   model="gpt-3.5-turbo"):
messages = [{"role": "user",
             "content": prompt}]

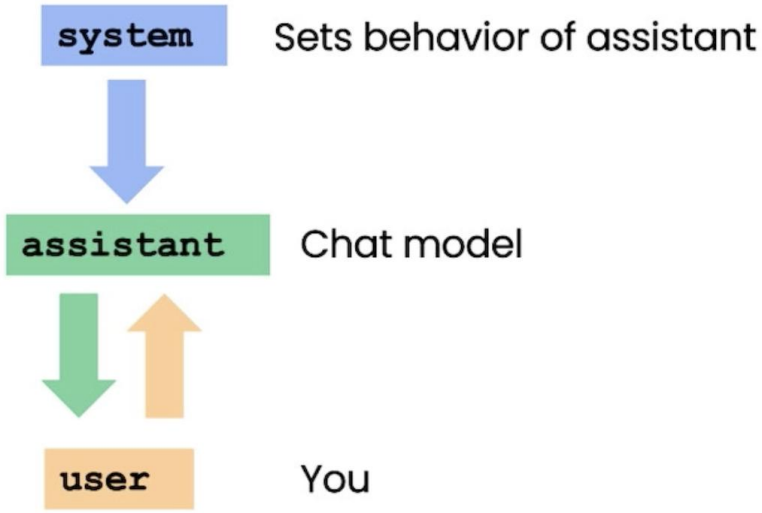
response = openai.ChatCompletion.create(
             model=model,
             messages=messages,
             temperature=0)
```



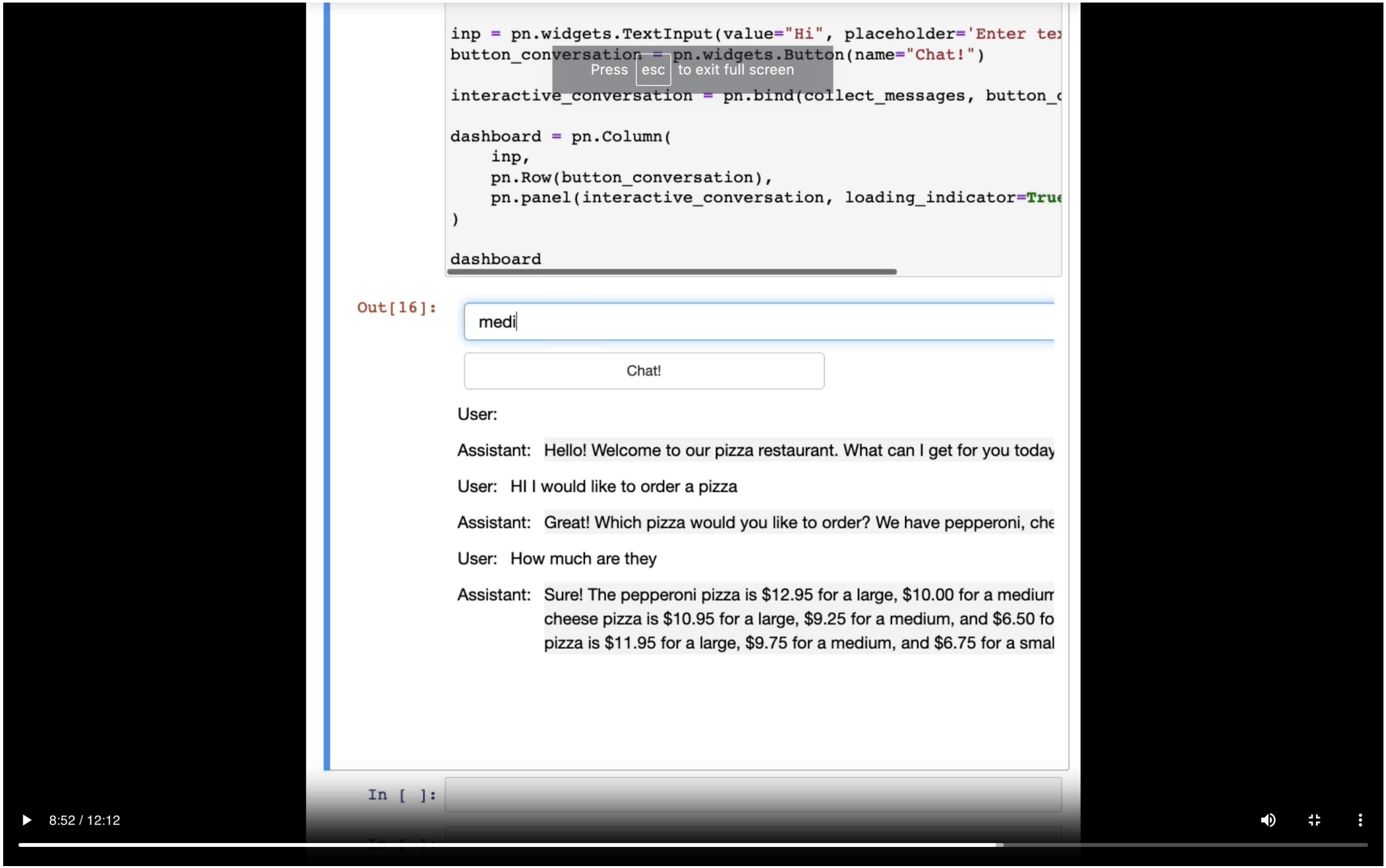
DeepLearning.AI OpenAI

Role

```
messages =
[
 {"role": "system",
  "content": "You are an assistant… "},
 {"role": "user",
  "content": "tell me a joke "},
 {"role": "assistant",
  "content": "Why did the chicken… "},
 …
]
```



So here's an example of a list of messages and so the first
message is a system message which kind of
gives an overall instruction and then after this
message we have turns between the user and
the assistant and this will kind of continue
to go on. And if you've ever used ChatGPT, the web interface, then your
messages are the user messages and then ChatGPT's
messages are the assistant messages. So the system message helps to
kind of set the behavior and persona of the assistant and it acts as kind of
a high-level instruction for the conversation. So you can kind of think of
it as whispering in the assistant's ear and kind of guiding its responses
without the user being aware of the system
message. So as the user, if you've ever used
ChatGPT, you probably don't know what's in ChatGPT's system message.
The benefit of the system message is that it provides you, the
developer, with a way to kind of frame the conversation without
making the request itself part of the conversation. So you can
kind of guide the assistant and
kind of whisper in its ear and guide its responses without making the user aware.


inp = pn.widgets.TextInput(value="Hi", placeholder='Enter tex
button_conversation = pn.widgets.Button(name="Chat!")
Press esc to exit full screen
interactive_conversation = pn.bind(collect_messages, button_c
dashboard = pn.Column(
inp,
pn.Row(button_conversation),
pn.panel(interactive_conversation, loading_indicator=True
)
dashboard
Out[16]:
medi
Chat!
User:
Assistant: Hello! Welcome to our pizza restaurant. What can I get for you today
User: HI I would like to order a pizza
Assistant: Great! Which pizza would you like to order? We have pepperoni, che
User: How much are they
Assistant: Sure! The pepperoni pizza is $12.95 for a large, $10.00 for a mediun
cheese pizza is $10.95 for a large, $9.25 for a medium, and $6.50 fo
pizza is $11.95 for a large, $9.75 for a medium, and $6.75 for a smal
In [ ]:
8:52 / 12:12