

<https://learn.deeplearning.ai/chatgpt-prompt-eng/lesson/2/guidelines>

note:

1. Get API Key from openAI to run scripts or api calls.

Basic setup:

```
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')


def get_completion(prompt, model="gpt-3.5-turbo", temperature=0):

    messages = [{"role": "user", "content": prompt}]

    response = openai.ChatCompletion.create(

        model=model,

        messages=messages,

        temperature=temperature,

    )

    return response.choices[0].message["content"]
```

Chapter 1: Guidelines:

Video link: https://dft3h5i221ap1.cloudfront.net/OpenAI/chatgpt-prompt-eng/video/prompt_eng_02_guidelines_v3.mp4

Principles of Prompting

Principle 1

- Write clear and specific instructions

clear \neq short

Principle 2

- Give the model time to think

Write clear prompt and not short version

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

Principles of Prompting

Principle 1

- Write clear and specific instructions

Principle 2

- Give the model time to think

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

Example : prompt with clear instructions:

```

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

```

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:  $100x + 250x + 100,000 + 100x = 450x + 100,000$

```

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

Model Limitations

Hallucination

Makes statements that sound plausible
but are not true

Example:

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


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.

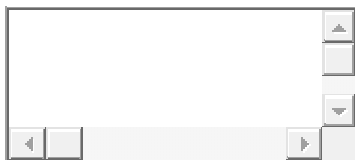
Lab - Guidelines for Prompting

In this lesson, you'll practice two prompting principles and their related tactics in order to write effective prompts for large language models.

Setup

Load the API key and relevant Python libraries.

In this course, we've provided some code that loads the OpenAI API key for you.



```
import openai
import os

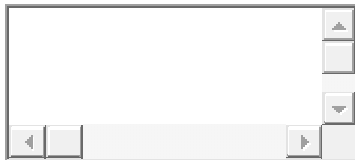
from dotenv import load_dotenv, find_dotenv
_ = load_dotenv(find_dotenv())
```

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

helper function

Throughout this course, we will use OpenAI's gpt-3.5-turbo model and the [chat completions endpoint](#).

This helper function will make it easier to use prompts and look at the generated outputs:



```
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 the model's  
output  
    )  
    return response.choices[0].message["content"]
```

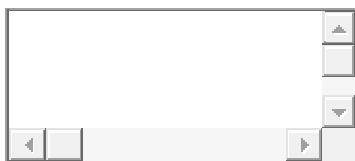
Prompting Principles

- **Principle 1: Write clear and specific instructions**
- **Principle 2: Give the model time to “think”**

Tactics

Tactic 1: Use delimiters to clearly indicate distinct parts of the input

- Delimiters can be anything like: ``, """, <>, <tag> </tag>, :



```
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 \  

```

```

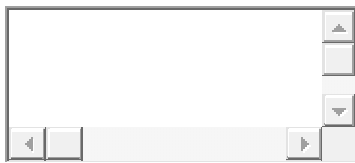
more detailed and relevant outputs.
"""
prompt = f"""
Summarize the text delimited by triple backticks \
into a single sentence.
```{text}```
"""

response = get_completion(prompt)
print(response)

```

### ***Tactic 2: Ask for a structured output***

- JSON, HTML



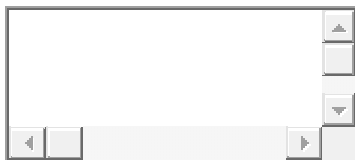
```

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)

```

### ***Tactic 3: Ask the model to check whether conditions are satisfied***



```

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, \n then simply write \"No steps provided.\\n\"

```
\\\"\\\"{text_1}\\\"\\\"\\n\"\\n\"
```

```
response = get_completion(prompt)
print("Completion for Text 1:")
print(response)
```



```
text_2 = f"""
The sun is shining brightly today, and the birds are \
singing. It's a beautiful day to go for a \
walk in the park. The flowers are blooming, and the \
trees are swaying gently in the breeze. People \
are out and about, enjoying the lovely weather. \
Some are having picnics, while others are playing \
games or simply relaxing on the grass. It's a \
perfect day to spend time outdoors and appreciate the \
beauty of nature.
"""

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:
```

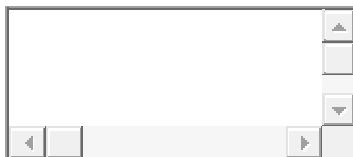
```
Step 1 - ...
Step 2 - ...
...
Step N - ...
```

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

```
\\\"\\\"{text_2}\\\"\\\"\\n\"\\n\"
```

```
response = get_completion(prompt)
print("Completion for Text 2:")
print(response)
```

#### ***Tactic 4: "Few-shot" prompting***



```
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.
```

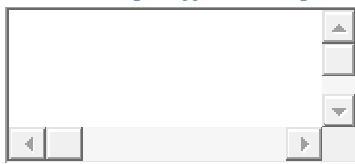
```
"""
```

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

```
print(response)
```

## Principle 2: Give the model time to “think”

### *Tactic 1: Specify the steps required to complete a task*



```
text = f"""
```

```
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)
```

### *Ask for output in a specified format*



```
prompt_2 = f"""
Your task is to perform the following actions:
1 - Summarize the following text delimited by
 <> 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.
```

```
Use the following format:
Text: <text to summarize>
Summary: <summary>
Translation: <summary translation>
Names: <list of names in Italian summary>
Output JSON: <json with summary and num_names>
```

```
Text: <{text}>
"""
```

```
response = get_completion(prompt_2)
print("\nCompletion for prompt 2:")
print(response)
```

### *Tactic 2: Instruct the model to work out its own solution before rushing to a conclusion*



```
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)

```

*Note that the student's solution is actually not correct.*

*We can fix this by instructing the model to work out its own solution first.*



```

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 \  
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
```

Actual solution:
"""
response = get_completion(prompt)
print(response)

```

Model Limitations: Hallucinations

- Boie is a real company, the product name is not real.



```

prompt = f"""
Tell me about AeroGlide UltraSlim Smart Toothbrush by Boie
"""

response = get_completion(prompt)
print(response)

```

Try experimenting on your own!



Notes on using the OpenAI API outside of this classroom

To install the OpenAI Python library:

```
!pip install openai
```

The library needs to be configured with your account's secret key, which is available on the [website](#).

You can either set it as the `OPENAI_API_KEY` environment variable before using the library:


```
!export OPENAI_API_KEY='sk-...'
```

Or, set `openai.api_key` to its value:

```
import openai  
openai.api_key = "sk-..."
```

A note about the backslash

- In the course, we are using a backslash `\` to make the text fit on the screen without inserting newline `'\n'` characters.
- GPT-3 isn't really affected whether you insert newline characters or not. But when working with LLMs in general, you may consider whether newline characters in your prompt may affect the model's performance.

Cheat sheet from Guidelines:

Keywords to keep in mind:

1. Your task is to answer in a consistent style.
2. 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.\"
3. Provide them in a html format with following keys:
4. Summarize the given sentence with number words used, number of sentences and also with one line summary.
5. 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.
6. Use the following format:
 - Text: <text to summarize>
 - Summary: <summary>
 - Translation: <summary translation>
 - Names: <list of names in Italian summary>
 - Output JSON: <json with summary and num_names>

Instruct the model to work out its own solution before rushing to a conclusion

7. 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 \
- 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
- ```
- Actual solution
- """

Hallucinations

8. Tell me about AeroGlide UltraSlim Smart Toothbrush by Boie

Iterative prompt development

9. Provide different level for getting better details
 - Prompt 1:
 - prompt = f"""
 - Your task is to help a marketing team create a
 - description for a retail website of a product based
 - on a technical fact sheet.
 -
 - Write a product description based on the information
 - provided in the technical specifications delimited by
 - triple backticks.
 -
 - Technical specifications: ```{fact_sheet_chair}```
 - """
 - Prompt 2:
 - prompt = f"""
 - Your task is to help a marketing team create a
 - description for a retail website of a product based
 - on a technical fact sheet.
 -
 - Write a product description based on the information
 - provided in the technical specifications delimited by
 - triple backticks.
 -
 - Use at most 50 words.
 -
 - Technical specifications: ```{fact_sheet_chair}```
 - """
 - Prompt 3:
 - prompt = f"""
 - Your task is to help a marketing team create a

- description for a retail website of a product based
 - on a technical fact sheet.
 -
 - Write a product description based on the information
 - provided in the technical specifications delimited by
 - triple backticks.
 -
 - The description is intended for furniture retailers,
 - so should be technical in nature and focus on the
 - materials the product is constructed from.
 -
 - At the end of the description, include every 7-character
 - Product ID in the technical specification.
 -
 - Use at most 50 words.
 -
 - Technical specifications: ```\${fact_sheet_chair}```
 - """
- Prompt 4:
 - prompt = f"""
 - Your task is to help a marketing team create a
 - description for a retail website of a product based
 - on a technical fact sheet.
 -
 - Write a product description based on the information
 - provided in the technical specifications delimited by
 - triple backticks.
 -
 - The description is intended for furniture retailers,
 - so should be technical in nature and focus on the
 - materials the product is constructed from.
 -
 - At the end of the description, include every 7-character
 - Product ID in the technical specification.
 -
 - After the description, include a table that gives the
 - product's dimensions. The table should have two columns.
 - In the first column include the name of the dimension.
 - In the second column include the measurements in inches only.
 -
 - Give the table the title 'Product Dimensions'.
 -

- Format everything as HTML that can be used in a website.
- Place the description in a <div> element.
-
- Technical specifications: ``{fact_sheet_chair}``
- ""

Summarization

10. Summarize with a word/sentence/character limit
 - Summarize the review below, delimited by triple backticks, in at most 30 words.
11. Summarize with a focus on shipping and delivery:
 - Summarize the review below, delimited by triple
 - backticks, in at most 30 words, and focusing on any aspects \
 - that mention shipping and delivery of the product.
12. Summarize with a focus on price and value
 - Summarize the review below, delimited by triple
 - backticks, in at most 30 words, and focusing on any aspects \
 - that are relevant to the price and perceived value.
13. Try "extract" instead of "summarize"
 - for i in range(len(reviews)):
 - prompt = f""
 - Your task is to generate a short summary of a product \
 - review from an ecommerce site.
 -
 - Summarize the review below, delimited by triple \
 - backticks in at most 20 words.
 -
 - Review: ``{reviews[i]}``
 - ""

Sentiment (positive/negative) and more:

Transforming:

1. Prompt 3: **Proof read and APA style guide:**
 - a. proofread and correct this review. Make it more compelling.
 - b. Ensure it follows APA style guide

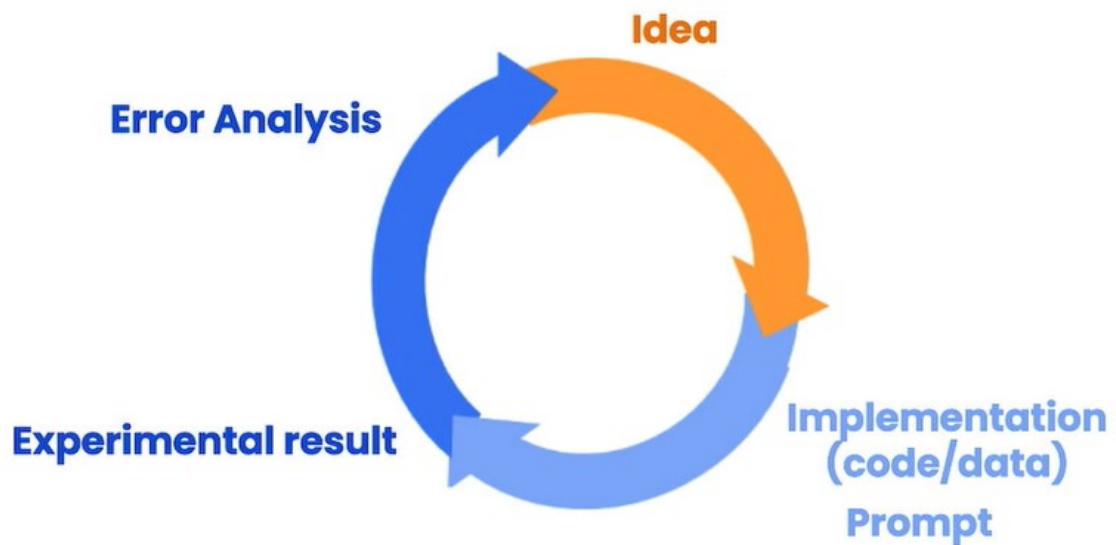
Expanding – Temperature property (value from 0 to 0.7/1):

1. temperature=0 to 0.7. Making sure the AI provide more random messages based on more temperatures.
 - a. prompt = f""
 - b. You are a customer service AI assistant.

- c. Your task is to send an email reply to a valued customer.
- d. Given the customer email delimited by ```\
- e. Generate a reply to thank the customer for their review.
- f. If the sentiment is positive or neutral, thank them for \
- g. their review.
- h. If the sentiment is negative, don't apologize and suggest that \
- i. they can reach out to customer service with their warrent card.
- j. Make sure to use specific details from the review.
- k. Write in a concise and professional tone.
- l. Sign the email as `AI customer agent`.
- m. Customer review: ```{review}```
- n. Review sentiment: {sentiment}
- o. """
- p. response = get_completion(prompt,temperature=0.7)
- q. print(response)

Chapter 2: Iterative Prompt Development

Iterative Prompt Development

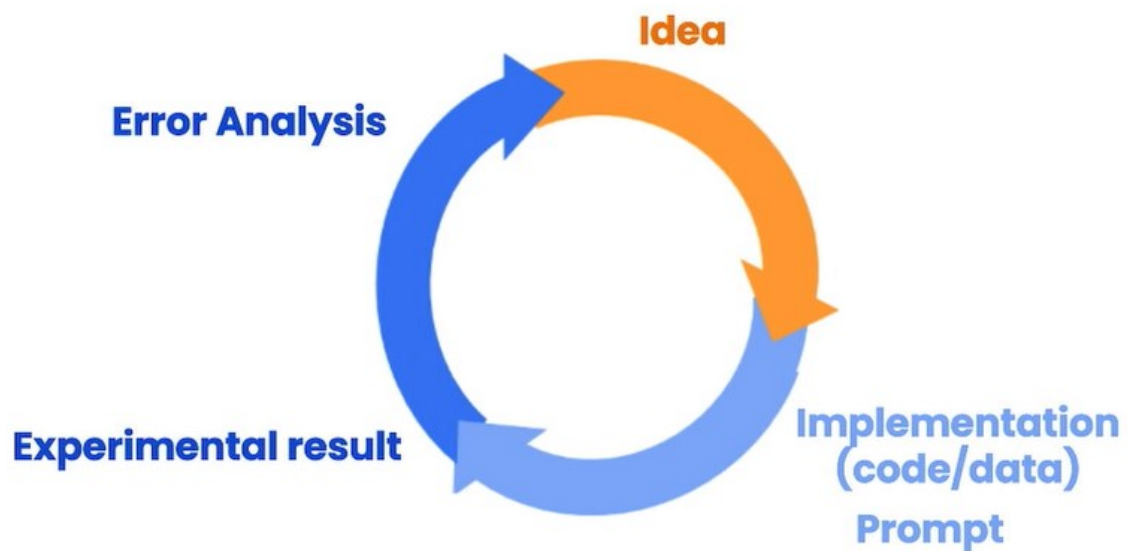


Prompt guidelines

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

Cheat sheet in the above list for iterative prompting.

Iterative Prompt Development



Iterative Process

- Try something
- Analyze where the result does not give what you want
- Clarify instructions, give more time to think
- Refine prompts with a batch of examples

Text focuses on the wrong details

Use at most 50 words.

Description needs a table of dimensions

At the end of the description, include every 7-character

Product ID in the technical specification.

After the description, include a table that gives the product's dimensions. The table should have two columns.

In the first column include the name of the dimension.

In the second column include the measurements in inches only.

Give the table the title 'Product Dimensions'.

Format everything as HTML that can be used in a website.

Place the description in a <div> element.

Summarize:

1. Summarize with a word/sentence/character limit
 - Summarize the review below, delimited by triple backticks, in at most 30 words.
2. Summarize with a focus on shipping and delivery:
 - Summarize the review below, delimited by triple backticks, in at most 30 words, and focusing on any aspects \
 - that mention shipping and delivery of the product.
3. Summarize with a focus on price and value
 - Summarize the review below, delimited by triple backticks, in at most 30 words, and focusing on any aspects \
 - that are relevant to the price and perceived value.
4. Try "extract" instead of "summarize"
 - for i in range(len(reviews)):
 - prompt = f"""
 - Your task is to generate a short summary of a product \
 - review from an ecommerce site.
 -
 - Summarize the review below, delimited by triple \
 - backticks in at most 20 words.
 -
 - Review: ```{reviews[i]}``` """

Sentiment (positive/negative) and more:

1. Sentiment (positive/negative)

a. Prompt 1:

- i. prompt = f"""
- ii. What is the sentiment of the following product review,
- iii. which is delimited with triple backticks?
- iv.
- v. Review text: "{lamp_review}"
- vi. """

b. Prompt 2:

- i. prompt = f"""
- ii. What is the sentiment of the following product review,
- iii. which is delimited with triple backticks?
- iv.
- v. Give your answer as a single word, either "positive" \
- vi. or "negative".
- vii.
- viii. Review text: "{lamp_review}"
- ix. """

2. Identify types of emotions

- a. prompt = f"""
- b. Identify a list of emotions that the writer of the \
- c. following review is expressing. Include no more than \
- d. five items in the list. Format your answer as a list of \
- e. lower-case words separated by commas.
- f.
- g. Review text: "{lamp_review}"
- h. """

3. Identify anger

- a. prompt = f"""
- b. Is the writer of the following review expressing anger?\
- c. The review is delimited with triple backticks. \
- d. Give your answer as either yes or no.
- e.
- f. Review text: "{lamp_review}"
- g. """

4. Extract product and company name from customer reviews

- a. prompt = f"""
- b. Identify the following items from the review text:
- c. - Sentiment (positive or negative)

- d. - Is the reviewer expressing anger? (true or false)
 - e. - Item purchased by reviewer
 - f. - Company that made the item
 - g.
 - h. The review is delimited with triple backticks. \
 - i. Format your response as a JSON object with \
 - j. "Sentiment", "Anger", "Item" and "Brand" as the keys.
 - k. If the information isn't present, use "unknown" \
 - l. as the value.
 - m. Make your response as short as possible.
 - n. Format the Anger value as a boolean.
 - o.
 - p. Review text: "{lamp_review}"
 - q. ""
5. Inferring topics
- a. Prompt 1:
 - i. prompt = f""
 - ii. Determine five topics that are being discussed in the \
 - iii. following text, which is delimited by triple backticks.
 - iv.
 - v. Make each item one or two words long.
 - vi.
 - vii. Format your response as a list of items separated by commas.
 - viii.
 - ix. Text sample: "{story}"
 - x. ""
 - b. Prompt 2:
 - i. topic_list = [
 - ii. "nasa", "local government", "engineering",
 - iii. "employee satisfaction", "federal government"
 - iv.]
 - c. Prompt 3:
 - i. Give your answer as a single word, either "positive" \
 - ii. or "negative".
6. Make a news alert for certain topics
- a. Prompt 1:
 - i. prompt = f""
 - ii. Determine whether each item in the following list of \
 - iii. topics is a topic in the text below, which
 - iv. is delimited with triple backticks.
 - v.
 - vi. Give your answer as list with 0 or 1 for each topic.\

- vii.
- viii. List of topics: {"", ".join(topic_list)}
- ix.
- x. Text sample: ""{story}""
- xi. """"
- b. Prompt 2:
 - i. topic_dict = {i.split(':')[0]: int(i.split(':')[1]) for i in response.split(sep='\n')}
 - ii. if topic_dict['nasa'] == 1:
 - iii. print("ALERT: New NASA story!")

Transforming : translating to a different language and more:

Translation

1. Prompt 1: trasulate English to spanish
 - a. prompt = f""
 - b. Translate the following English text to Spanish: \
 - c. ``Hi, I would like to order a blender``
 - d. ""
2. Prompt 2 : which language
 - a. prompt = f""
 - b. Tell me which language this is:
 - c. ``Combien coûte le lampadaire?``
 - d. ""
3. Prompt 3: transulate to multiple languages
 - a. prompt = f""
 - b. Translate the following text to French and Spanish
 - c. and English pirate: \
 - d. ``I want to order a basketball``
 - e. ""
4. Prompt 4: formal or informal forms
 - a. prompt = f""
 - b. Translate the following text to Spanish in both the \
 - c. formal and informal forms:
 - d. 'Would you like to order a pillow?'
 - e. ""

Universal Translator:

1. Prompt 1: multiple language sentences
 - a. user_messages = [
 - "La performance du système est plus lente que d'habitude.", # System performance is slower than normal

```

    "Mi monitor tiene píxeles que no se iluminan.",          # My monitor has pixels that are
not lighting
    "Il mio mouse non funziona",                            # My mouse is not working
    "Mój klawisz Ctrl jest zepsuty",                        # My keyboard has a broken control key
    "我的屏幕在闪烁"                                       # My screen is flashing
    ].
b. for issue in user_messages:
    prompt = f"Tell me what language this is: ``{issue}``"
    lang = get_completion(prompt)
    print(f"Original message ({lang}): {issue}")

    prompt = f"""
Translate the following text to English \
and Korean: ``{issue}``
"""
    response = get_completion(prompt)
    print(response, "\n")
c. Prompt3: Translate sentence to English:
    i. prompt = f"translate the following text to English and Tamil: 'ಡීಪ್ ಲರ್ಟಿಂಗ್
ಎಯಿ ಕು ಸ್ವಾಗತಂ'"
    response = get_completion(prompt)
    print(response, "\n")

```

2. Tone Transformation:

```

a. Prompt 1: general slang to business letter
    i. prompt = f"""
    ii. Translate the following from slang to a business letter:
    iii. 'Dude, This is Joe, check out this spec on this standing lamp.'
    iv. """
    v. response = get_completion(prompt)
    vi. print(response)
b. Prompt 2: transform HTML to JSON and viseversa
    i. data_json = { "restaurant employees" :[
    ii.     {"name":"Shyam", "email":"shyamjaiswal@gmail.com"},
    iii.     {"name":"Bob", "email":"bob32@gmail.com"},
    iv.     {"name":"Jai", "email":"jai87@gmail.com"}
    v.     ]}
    vi.
    vii. prompt = f"""
    viii. Translate the following python dictionary from JSON to an HTML \
    ix. table with column headers and title: {data_json}
    x. """

```

- xi. `response = get_completion(prompt)`
- xii. `print(response)`

3. spellcheck / grammer check:

2. Prompt 1: Proofread and correct

- a. `text = [`
- b. `"The girl with the black and white puppies have a ball.", # The girl has a ball.`
- c. `"Yolanda has her notebook.", # ok`
- d. `"Its going to be a long day. Does the car need it's oil changed?", # Homonyms`
- e. `"Their goes my freedom. There going to bring they're suitcases.", # Homonyms`
- f. `"Your going to need you're notebook.", # Homonyms`
- g. `"That medicine effects my ability to sleep. Have you heard of the butterfly affect?",`
`# Homonyms`
- h. `"This phrase is to cherck chatGPT for speling abilitty" # spelling`
- i. `]`
- j. `for t in text:`
- k. `prompt = f"""Proofread and correct the following text`
- l. `and rewrite the corrected version. If you don't find`
- m. `and errors, just say "No errors found". Don't use`
- n. `any punctuation around the text:`
- o. ````{t}```"""`
- p. `response = get_completion(prompt)`
- q. `print(response)`

3. Prompt 2: proofread and correct this review

- a. `text = f"""`
- b. `Got this for my daughter for her birthday cuz she keeps taking \`
- c. `mine from my room. Yes, adults also like pandas too. She takes \`
- d. `it everywhere with her, and it's super soft and cute. One of the \`
- e. `ears is a bit lower than the other, and I don't think that was \`
- f. `designed to be asymmetrical. It's a bit small for what I paid for it \`
- g. `though. I think there might be other options that are bigger for \`
- h. `the same price. It arrived a day earlier than expected, so I got \`
- i. `to play with it myself before I gave it to my daughter.`
- j. `"""`
- k. `prompt = f"proofread and correct this review: ```{text}```"`
- l. `response = get_completion(prompt)`
- m. `print(response)`

4. Python code to color red for differences:

- a. `from redlines import Redlines`
- b.
- c. `diff = Redlines(text,response)`
- d. `display(Markdown(diff.output_markdown))`

5. Prompt 3: **Proof read and APA style guide:**
 - a. proofread and correct this review. Make it more compelling.
 - b. Ensure it follows APA style guide

Expanding: you will generate customer service emails that are tailored to each customer's review.

2. Customize the automated reply to a customer email:
 - a. Prompt 1: AI email sent to customer for responding to the review.
 - i. `prompt = f"""`
 - ii. You are a customer service AI assistant.
 - iii. Your task is to send an email reply to a valued customer.
 - iv. Given the customer email delimited by `````, \
 - v. Generate a reply to thank the customer for their review.
 - vi. If the sentiment is positive or neutral, thank them for \
 - vii. their review.
 - viii. If the sentiment is negative, apologize and suggest that \
 - ix. they can reach out to customer service.
 - x. Make sure to use specific details from the review.
 - xi. Write in a concise and professional tone.
 - xii. Sign the email as ``AI customer agent``.
 - xiii. Customer review: ````{review}````
 - xiv. Review sentiment: `{sentiment}`
 - xv. `"""`
 - xvi. `response = get_completion(prompt)`
 - xvii. `print(response)`
 - b. Prompt 2: Remind the model to use details from customer email:
 - i. `prompt = f"""`
 - ii. You are a customer service AI assistant.
 - iii. Your task is to send an email reply to a valued customer.
 - iv. Given the customer email delimited by `````, \
 - v. Generate a reply to thank the customer for their review.
 - vi. If the sentiment is positive or neutral, thank them for \
 - vii. their review.
 - viii. If the sentiment is negative, apologize and suggest that \
 - ix. they can reach out to customer service.
 - x. Make sure to use specific details from the review.
 - xi. Write in a concise and professional tone.
 - xii. Sign the email as ``AI customer agent``.
 - xiii. Customer review: ````{review}````

- xiv. Review sentiment: {sentiment}
 - xv. """
 - xvi. response = get_completion(prompt, temperature=0.7)
 - xvii. print(response)
- 3. temperature=0 to 0.7. Making sure the AI provide more random messages based on more temperatures.
 - a. prompt = f"""
 - b. You are a customer service AI assistant.
 - c. Your task is to send an email reply to a valued customer.
 - d. Given the customer email delimited by ``` \
 - e. Generate a reply to thank the customer for their review.
 - f. If the sentiment is positive or neutral, thank them for \
 - g. their review.
 - h. If the sentiment is negative, don't apologize and suggest that \
 - i. they can reach out to customer service with their warrent card.
 - j. Make sure to use specific details from the review.
 - k. Write in a concise and professional tone.
 - l. Sign the email as `AI customer agent`.
 - m. Customer review: ```{review}```
 - n. Review sentiment: {sentiment}
 - o. """
 - p. response = get_completion(prompt,temperature=0.7)
 - q. print(response)

ChatBot

One of the exciting things about a large language model is you could use it to build a custom chatbot with only a modest amount of effort. ChatGPT, the web interface, is a way for you to have a conversational interface, a conversation via a large language model. But one of the cool things is you can also use a large language model to build your custom chatbot to maybe play the role of an AI customer service agent or an AI order taker for a restaurant.



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)
```

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)
```

Chat model

assistant



user



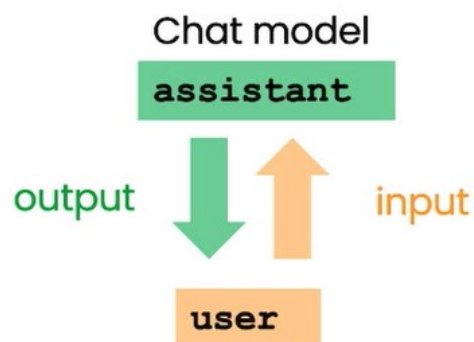

1:40 / 12:13



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)
```



```

    return response.choices[0].message["content"]

def get_completion_from_messages(messages, model="gpt-3.5-turbo", temperature=0):
    response = openai.ChatCompletion.create(
        model=model,
        messages=messages,
        temperature=temperature, # this is the degree of randomness of the model
    )
    return response.choices[0].message["content"]

```

Role

```

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

```

system

assistant



user

Role

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

system

Sets behavior of assistant



assistant

Chat model



user

You



2:38 / 12:13



System message

```
context = [  
    {'role': 'system',  
      'content': """  
You are OrderBot, an automated service  
to collect orders for a pizza restaurant.  
You first greet the customer,  
then collect the order,  
and then ask if it's a pickup or delivery.  
You wait to collect the entire order,  
then summarize it and check for a final time  
if the customer wants to add anything else.  
If it's a delivery, you ask for an address.  
Finally you collect the payment.  
Make sure to clarify all options, extras and sizes  
to uniquely identify the item from the menu.  
You respond in a short, very conversational  
friendly style.  
The menu includes  
.  
"""  
}]
```

Setup

1. new method to collect multiple messages and responds based one conversation.

```
def get_completion_from_messages(messages, model="gpt-3.5-turbo", temperature=0):

    response = openai.ChatCompletion.create(

        model=model,

        messages=messages,

        temperature=temperature, # this is the degree of randomness of the model's output

    )

# print(str(response.choices[0].message))

return response.choices[0].message["content"]
```

Example 1: Simple group messages related to different rols (system, user, assistant)

```
messages = [

    {'role':'system', 'content':'You are an assistant that speaks like Shakespeare.'},

    {'role':'user', 'content':'tell me a joke'},

    {'role':'assistant', 'content':'Why did the chicken cross the road'},

    {'role':'user', 'content':'I don\'t know'} ]

response = get_completion_from_messages(messages, temperature=1)

print(response)
```

Prompt 1:

```
messages = [

    {'role':'system', 'content':'You are friendly chatbot.'},

    {'role':'user', 'content':'Yes, can you remind me, What is my name?'} ]

response = get_completion_from_messages(messages, temperature=1)

print(response)
```


Prompt 2:

```
messages = [  
    {'role':'system', 'content':'You are friendly chatbot.'},  
    {'role':'user', 'content':'Hi, my name is Isa'},  
    {'role':'assistant', 'content': "Hi Isa! It's nice to meet you. \n  
Is there anything I can help you with today?"},  
    {'role':'user', 'content':'Yes, you can remind me, What is my name?'} ]  
response = get_completion_from_messages(messages, temperature=1)  
print(response)
```

ORDER BOT

Part 1: method to collect messages

```
def collect_messages(_):  
    prompt = inp.value_input  
    inp.value = "  
    context.append({'role':'user', 'content':f"{prompt}"})  
    response = get_completion_from_messages(context)  
    context.append({'role':'assistant', 'content':f"{response}"})  
    panels.append(  
        pn.Row('User:', pn.pane.Markdown(prompt, width=600)))  
    panels.append(  
        pn.Row('Assistant:', pn.pane.Markdown(response, width=600, style={'background-color':  
        '#F6F6F6'})))  
  
    return pn.Column(*panels)
```

Part 2: create UI panel and process context and with chat button, process messages

```
import panel as pn # GUI
```

```
pn.extension()
```

```
panels = [] # collect display
```

```
context = [ {'role':'system', 'content':"""
```

```
You are OrderBot, an automated service to collect orders for a pizza restaurant. \
```

```
You first greet the customer, then collects the order, \
```

```
and then asks if it's a pickup or delivery. \
```

```
You wait to collect the entire order, then summarize it and check for a final \
```

```
time if the customer wants to add anything else. \
```

```
If it's a delivery, you ask for an address. \
```

```
Finally you collect the payment.\
```

```
Make sure to clarify all options, extras and sizes to uniquely \
```

```
identify the item from the menu.\
```

```
You respond in a short, very conversational friendly style. \
```

```
The menu includes \
```

```
pepperoni pizza 12.95, 10.00, 7.00 \
```

```
cheese pizza 10.95, 9.25, 6.50 \
```

```
eggplant pizza 11.95, 9.75, 6.75 \
```

```
fries 4.50, 3.50 \
```

```
greek salad 7.25 \
```

```
Toppings: \
```

```
extra cheese 2.00, \
```

```
mushrooms 1.50 \
```

```
sausage 3.00 \
canadian bacon 3.50 \
AI sauce 1.50 \
peppers 1.00 \
Drinks: \
coke 3.00, 2.00, 1.00 \
sprite 3.00, 2.00, 1.00 \
bottled water 5.00 \
"""} ] # accumulate messages
```

```
inp = pn.widgets.TextInput(value="Hi", placeholder='Enter text here...')
button_conversation = pn.widgets.Button(name="Chat!")

interactive_conversation = pn.bind(collect_messages, button_conversation)

dashboard = pn.Column(
    inp,
    pn.Row(button_conversation),
    pn.panel(interactive_conversation, loading_indicator=True, height=300),
)
```

Dashboard

```

dashboard = pn.Column(
    inp,
    pn.Row(button_conversation),
    pn.panel(interactive_conversation, loading_indicator=True, height=300)
)

dashboard

```

Enter text here...

Chat!

User:

Assistant: Hello! Welcome to our pizza restaurant. What can I get for you today?

User: hi

Assistant: Hi there! How can I assist you with your order today?

User: like to order veggitarian pizza

Assistant: Great choice! Would you like a cheese pizza or an eggplant pizza? And what size would you like? We have small, medium, and large.

User: eggplant with medium size

Assistant: Excellent! An eggplant pizza in medium size. Would you like to add any topping? We have extra cheese, mushrooms, sausage, Canadian bacon, AI sauce, and

User: sausage

Step 3: print total list of order and price to kichen.

```
messages = context.copy()
```

```
messages.append(
```

```
{'role':'system', 'content':'create a json summary of the previous food order. Itemize the price for each item\
```

```
The fields should be 1) pizza, include size 2) list of toppings 3) list of drinks, include size 4) list of sides include size 5)total price '},
```

```
)
```

```
#The fields should be 1) pizza, price 2) list of toppings 3) list of drinks, include size include price 4) list of sides include size include price, 5)total price '},
```

```
response = get_completion_from_messages(messages, temperature=0)
```

```
print(response)
```

```
In [11]: messages = context.copy()
messages.append(
    {'role': 'system', 'content': 'create a json summary of the previous food
    The fields should be 1) pizza, include size 2) list of toppings 3) list of drinks
    #The fields should be 1) pizza, price 2) list of toppings 3) list of drinks'}
response = get_completion_from_messages(messages, temperature=0)
print(response)
```

Here's a JSON summary of the previous food order:

```
...
{
  "pizza": {
    "type": "eggplant",
    "size": "medium",
    "toppings": [
      "sausage"
    ],
    "price": 11.95
  },
  "sides": [
    {
      "type": "fries",
      "size": "medium",
      "price": 3.50
    }
  ],
  "drinks": [],
  "total_price": 15.45
}
...
```

Note that the drinks field is empty since the customer did not order any drinks.

Tried one example with southIndian restaurant menu:

```
import panel as pn # GUI
```

```
pn.extension()
```

```
panels = [] # collect display
```

```
context = [ {'role':'system', 'content':""
```

```
You are OrderBot, an automated service to collect orders for a southIndian food restaurant. \
```

```
You first greet the customer, then collects the order, \
```

```
and then asks if it's a pickup or delivery. \
```

```
You wait to collect the entire order, then summarize it and check for a final \
```

```
time if the customer wants to add anything else. \
```

```
If it's a delivery, you ask for an address. \
```

```
Finally you collect the payment.\
```

```
Make sure to clarify all options, extras and sizes to uniquely \
```

```
identify the item from the menu.\
```

```
You respond in a short, very conversational friendly style. \
```

```
The menu includes \
```

```
dosa 8.00 \
```

```
masala dosa 10.00 \
```

```
onion dosa 9.00 \
```

```
onion masala dosa 11.00 \
```

```
idle (3 per plate) 4.00 \
```

```
ghee idle 5.00 \
```

```
masala ghee idle 8.00 \
```

```
upma 5.00 \
```

masala tea 1.00 \

tea 1.00 \

Drinks: \

coke 3.00, 2.00, 1.00 \

sprite 3.00, 2.00, 1.00 \

bottled water 5.00 \

"""}] # accumulate messages

inp = pn.widgets.TextInput(value="Hi", placeholder='Enter text here...')

button_conversation = pn.widgets.Button(name="Chat!")

interactive_conversation = pn.bind(collect_messages, button_conversation)

dashboard = pn.Column(

inp,

pn.Row(button_conversation),

pn.panel(interactive_conversation, loading_indicator=True, height=300),

)

dashboard

```

You respond in a short, very conversational friendly style. \
The menu includes \
dosa 8.00 \
masala dosa 10.00 \
onion dosa 9.00 \
onion masala dosa 11.00 \
idle (3 per plate) 4.00 \
ghee idle 5.00 \
masala ghee idle 8.00 \
upma 5.00 \
masala tea 1.00 \
tea 1.00 \
Drinks: \
coke 3.00, 2.00, 1.00 \
sprite 3.00, 2.00, 1.00 \
bottled water 5.00 \
"""} ] # accumulate messages

inp = pn.widgets.TextInput(value="Hi", placeholder='Enter text here...')
button_conversation = pn.widgets.Button(name="Chat!")

interactive_conversation = pn.bind(collect_messages, button_conversation)

dashboard = pn.Column(
    inp,
    pn.Row(button_conversation),
    pn.panel(interactive_conversation, loading_indicator=True, height=300)
)

dashboard

```

User: and one ghee idle and masala tea

Assistant: Sounds good! Would you like the Masala Ghee Idle or just the regular Ghee Idle for pickup or delivery?

User: regular ghee idle

Assistant: Got it! And for the Masala Tea, would you like it spicy or mild? And is this for pickup or delivery?

User: spicy

Assistant: Great! So to confirm, you'd like one plate of Dosa, one Ghee Idle, and one Spicy Masala Tea that correct? And is this for pickup or delivery?

User: pickup and provide total cost

Assistant: Sure, your total cost for the order will be \$14.00. You can pick up your order at our restaurant location. Is there anything else you'd like to add to your order?

User: nope i am good thank you

Assistant: You're welcome! Your order is confirmed for one plate of Dosa, one Ghee Idle, and one Spicy Masala Tea for a total of \$14.00. Please come to our restaurant location for pickup.

Order Summary

| Tiffen | Qty | Price |
|-----------|-----|--------|
| Dosa | 1 | \$8.00 |
| Ghee Idle | 1 | \$5.00 |

Toppings

None

| Drinks | Size | Price |
|------------------|---------|--------|
| Spicy Masala Tea | Regular | \$1.00 |

Sides

None

Total Price: \$14.00

```
In [14]: messages = context.copy()
messages.append(
    {'role': 'system', 'content': 'create a json summary of the previous food
    Print the bill with html version\
    The fields should be 1) tiffen name, include qty 2) list of toppings 3
    )
    #The fields should be 1) pizza, price 2) list of toppings 3) list of d
    response = get_completion_from_messages(messages, temperature=0)
    print(response)
```

Here's a JSON summary of the previous food order:

```
...
{
  "tiffen": [
    {
      "name": "Dosa",
      "qty": 1,
      "price": 8.0
    },
  ],
}
```

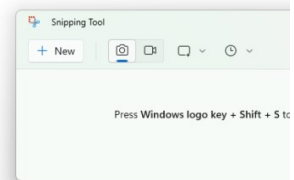
Summary

Conclusion



Summary

- Principles:
 - Write clear and specific instructions
 - Give the model time to “think”
- Iterative prompt development
- Capabilities: Summarizing, Inferring, Transforming, Expanding
- Building a chatbot



Jobs requirements for Prompt engineer:

Job 1:

https://www.indeed.com/q-Chatgpt-Prompt-Engineer-jobs.html?vjk=269f9bdbebaca9f2&aceid=&gclid=CjwKCAjw67ajBhAVEiwA2g_jEL6Uu_3cGloKs9t-RSVpGsqtC_lckrWVoHZGQe8OyXbUrFOzyrOmcxoCt_IQAvD_BwE&gclsrc=aw.ds

Klarity (YC S18) uses Natural Language Processing (NLP) to automate the enterprise contracts review and information extraction process. Our customers include companies like Coupa, Okta, RingCentral, Optimizely, and Lattice and they use Klarity to (1) save time and cost, (2) reduce compliance risk, and (3) speed up manual contract review.

We have spent years working on our core NLP platform and found strong Product Market Fit in 2020. We've since experienced hypergrowth and raised an \$18m Series A in Dec 2021. We are looking for a highly motivated individual who is passionate about our mission, and can accelerate deployment of our ML platform to our customer base.

We are looking for a brilliant Engineer to explore cutting-edge LLM (incl. GPT, ChatGPT, Dall E) applications within Klarity. These applications will be wide ranging; tabular data extraction, identifying esoteric non-standard legal terms, estimating prediction confidence and error, OCR enhancements to name a few.

We have strong conviction that LLMs will revolutionize the world and that we are uniquely positioned to take advantage of this technology given our years of experience shipping NLP products to large enterprises for document review. You will be the first dedicated subject matter expert on LLMs within Klarity and we will lean on you to pioneer LLM applications.

You will work closely with Klarity's CTO, VP Eng, as a member of our world-class ML team.

Location: San Francisco Bay Area (although daily office attendance not required)

What this role is:

- Moving at breakneck speed to prototype new LLM applications.
- Working with several cross functional teams across the company.
- Developing a keen ability to condition (prompt, finetune, chat with) LLMs and understand how to produce the best output from them.
- Staying on top of emerging trends in LLMs.
- Being an LLM product thinker in the sense of reformulating existing problem statements to a form that LLMs can tackle.

Klarity is fundamentally an applied ML company and this is an applied ML role. For clarity's sake (pun intended), here is **what this role is not**:

- Performing deep research on new model architectures. We'll leave GPT-4 to OpenAI.
- Publishing papers.
- Spending vast amounts of time trying to replicate results from academia and attending conferences.
- Pursuing SoTA on different shared tasks (PoS, NER, SquaD etc).

Requirements:

- Experience with natural language processing (NLP)
- Experience with large language models, such as BERT, GPT, or Transformers
- Strong programming skills, with experience in Python and ML libraries
- Proven track record of ideation, designing and building prototypes and demo'ing, documenting and sharing the learnings
- 3-5 years of experience with building and deploying ML applications
- Strong problem-solving and communication skills
- Bachelor's or Master's degree in Computer Science or a related field

Compensation Range: \$130K - \$230K

Job 2:

<https://prompt-engineering-jobs.com/jobs/prompt-engineer-highp9/>

Position Overview:

Here at ShyftLabs we are in search of an experienced and motivated Prompt Engineer to join our team. In this role, you will be responsible for developing, designing, and maintaining our cutting-edge AI-based conversation systems, ensuring smooth and engaging user experiences.

ShyftLabs is a growing data product company that was founded in early 2020, and works primarily with Fortune 500 companies. We deliver digital solutions built to help accelerate the growth of businesses in various industries, by focusing on creating value through innovation.

Job Responsibilities:

Design, develop, and maintain robust, scalable, and efficient AI-driven conversation systems for a diverse range of applications and industries.

Collaborate with cross-functional teams, including product managers, UX designers, and data scientists, to understand user requirements and create user-centric solutions.

Optimize and fine-tune conversation algorithms to ensure optimal user experiences and reduce latency.

Analyze and address performance bottlenecks in the AI engine.

Stay up-to-date with the latest industry trends, technologies, and advancements in AI-driven conversation systems and apply this knowledge to improve existing systems and develop new features.

Participate in code reviews, providing and receiving constructive feedback to improve overall code quality.

Qualifications:

Bachelor's or Master's degree in Computer Science, Engineering, or a related field.

Minimum of 3 years of experience in software development, with a focus on AI-driven conversation systems, natural language processing, or machine learning.

Strong programming skills in one or more of the following languages: Python or Scala.

Solid understanding of AI concepts, LLM, LangChain, natural language processing, and machine learning algorithms.

Strong problem-solving skills and the ability to troubleshoot and debug complex software systems.

Excellent communication and collaboration skills, with the ability to work effectively in a fast-paced, multidisciplinary team environment.

Detail-oriented and highly organized, with strong time management skills and the ability to prioritize tasks effectively.

Experience with AI-driven conversation and integration technologies such as GPT, Hugging Face, AutoGPT, etc.

We are proud to offer a competitive salary alongside a strong healthcare insurance and benefits package. The role is preferably hybrid, with 2 days per week spent in office. We pride ourselves on the growth of our employees, offering extensive learning and development resources.

Job 3:

<https://prompt-engineering-jobs.com/jobs/ai-prompt-engineering-intern-fc9rnm/>