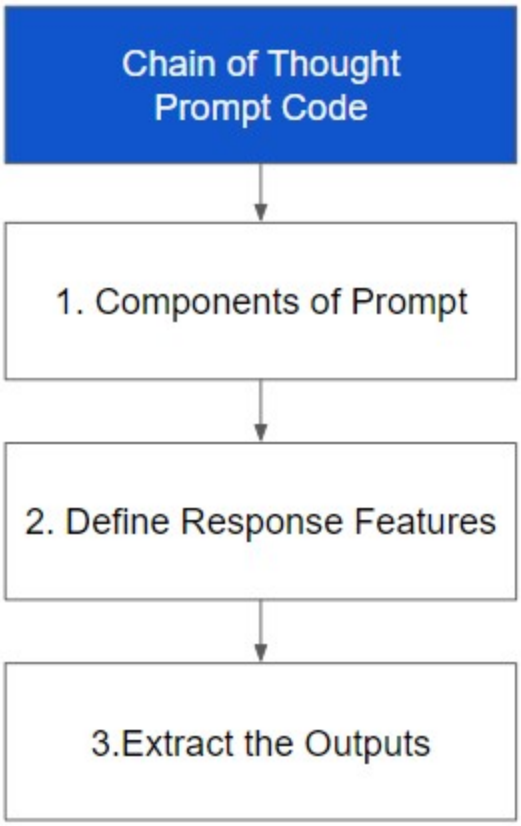
4. Chain of Thought Prompt Template



Section Overview: *In this section, we'll explore how to utilize Chain of Thought Promptin to generate responses. We'll achieve this by addressing a customer complaint step by step, extracting information sequentially. We'll craft the prompt accordingly and input it into the API function, which will generate the response for us.*

1. Define Components of Prompt

Section Overview: *Let us start by discussing components of prompt. CoT (Chain-of-Thought) prompting aims to encourage the model to generate more coherent and contextually relevant responses by guiding its thought process in a structured manner. Below is the code:*

Python ----------

```
le phone company to better understand customer complaints.
Customer complaints will be submitted as text delimited by triple backticks, that is, ```.
For each complaint, extract the following information and present it only in a JSON format:
1. phone_model: This is the name of the phone - if unknown, just say “UNKNOWN”
2. phone_price: The price in dollars - if unknown, assume it to be 1000 $
3. complaint_desc: A short description/summary of the complaint in less than 20 words
4. additional_charges: How much in dollars did the customer spend to fix the problem? - this should
be an integer
5. refund_expected: TRUE or FALSE - check if the customer explicitly mentioned the word “refund” to
tag as TRUE. If unknown, assume that the customer is not expecting a refund.
Take a step-by-step approach in your response, and give a detailed explanation before sharing your
final answer in the following JSON format:
{phone_model:, phone_price:, complaint_desc:, additional_charges:, refund_expected:}.
"""
user_message_template = "```{complaint}```"
```

```
customer_complaint = """
I am fuming with anger and regret over my purchase of the XUI890.
First, the price tag itself was exorbitant at 1500 $, making me expect exceptional quality.
Instead, it turned out to be a colossal disappointment.
The additional charges to fix its constant glitches and defects drained my wallet even more.
I spend 275 $ to get a new battery.
The final straw was when the phone's camera malfunctioned, and the repair cost was astronomical.
I demand a full refund and an apology for this abysmal product.
Returning it would be a relief, as this phone has become nothing but a money pit. Beware, fellow bu
yers!
"""
```

Python----------

- **Components of CoT Prompt:**
  - CoT Prompt has the same structure as a zero-shot prompt, consisting of two components: **system message** and **user message**.

- `system_message`: The system_message variable contains instructions for an assistant helping customer service representatives from a mobile phone company to extract specific information from customer complaints and present it in JSON format.

- **User Message Template:**
  - `user_message_template`: This is defined using string formatting. This template is structured with placeholders for the complaint text, enclosed within triple backticks. It allows the user to input their complaint in a specific format.
- **User Message (Customer Complaint)**
  - `customer_complaint`: An example customer complaint is provided as a multi-line string. This complaint contains details about the customer's feedback with their purchase of a specific phone model, along with associated issues and demands for a refund.

- **Prompt Structure:**
  - CoT prompts are structured to guide the model's thinking by presenting a series of interconnected steps, each with its own system and user messages.
  - This structured approach helps the model understand the context and build upon previous information to generate more relevant responses.

2. Define Response Features

Section Overview: *Now, let's delve into understanding all the input parameters required for the API function to generate the response.*

Python----------

CODE:

```
response = client.chat.completions.create(
 model=deployment_name,
 messages=[
 {"role": "system", "content": system_message},
 {"role": "user", "content": user_message_template.format(complaint=customer_complaint)},
 ]
```

Python----------

- **Chat Completion API:**
  - `client.chat.completions.create`: This is the function call to create a new chat completion using the OpenAI API.We will provide the input and all the parameters to help us get results to this compltion function.Let us look at the parameters.

- **Parameters**:
  - `model=deployment_name`: This parameter specifies the name of the model to use for the chat completion. The name of the model is stored in the variable deployment_name.
  - `messages`: This is a list of dictionaries that contains the system message and user message Note that unlike zero shot and few shot case, system message and user message of the CoT prompt are directly provided as input to the completion function .They are included within a list passed to the messages parameter of the function.
  - `{"role": "system", "content": system_message}`: This is the system message that provided to the model. It contains the instructions for the model to follow. Here role defines that this message is for system and content is the text in that message.
  - `{"role": "user", "content": user_message_template.format(complaint=customer_complaint)}`: It contains the customer complaint text. Here role defines that this is the user message and content is the text of user message.
  - `temperature=0`: This parameter specifies the randomness of the generated response. A temperature of 0 means that the model will generate a deterministic response.The higher temperature gives more creative responses and can move away from instructions sometimes
  - Parameters such as **deployment_name** and **temperature** remain consistent with previous examples, maintaining coherence with the zero-shot prompt approach.

3. Extract the Outputs

Section Overview: *We will now extract both the response and the token count for both the input prompt and the generated response.*

Python----------

CODE:

OUTPUT:

Number of tokens in prompt: 383

Number of tokens in completion: 194

CODE:

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

OUTPUT:

To extract the required information from the complaint, we can follow these steps:

1. Identify the phone model: In this complaint, the phone model is mentioned as "XUI890".
2. Identify the phone price: The complaint mentions that the price of the phone was "exorbitant at 1500 $".
3. Identify the complaint description: The complaint describes the phone as a "colossal disappointment".
4. Identify the additional charges: The complaint mentions that the customer spent "$275 to get a new battery".
5. Identify if a refund is expected: The complaint explicitly states that the customer "demand[s] a ful refund".

Based on the above analysis, we can present the information in the following JSON format:

{

"phone_model": "XUI890",

"phone_price": 1500,

"complaint_desc": "Colossal disappointment",

"additional_charges": 275,

"refund_expected": true

}

Python----------

- **Print Prompt Token Count from Response Object:**

- response.usage.prompt_tokens: The response object is the response from the Azure OpenAI Service's Chat Completion API. It contains various properties, including **usage**, which provide information about the resource usage of the model. The **prompt_tokens** value in the usage object is the number of tokens used in the prompt that was provided to the model.
- This helps you stay within the context length limit of the model you are using. As there is a maximum context of 4096 token length which gpt-35-turbo model can hold (including input and output token), above which the model gives error.

- **Print Completion Token Count:**
  - `print(f"Number of tokens in completion: response.usage.completion_tokens}")`: This prints the number of tokens in the completion generated by the chat model.
  - This helps us gauge the token length consumption by the model's response.Here total outpu token usage is **194**.
  - response.usage.completion_tokens: In the provided code, the response object is the respons from the Azure OpenAI Service's Chat Completion API. It contains various properties, includir usage, which provides information about the resource usage of the model. The completion_tokens value in the usage object is the number of tokens used in the completion that was generated by the model.
  - Same as input token case, here also it is important to keep record of token count as the overall input and out token count should not go above 4096 for the gpt-35-turbo model.

- **Retrieve Actual Text Output:**
  - `response.choices[0].message.content`: This retrieve the actual text output generated by the model based on the provided prompt and parameters.
  - In the provided code, the response object is the response from the Azure OpenAI Service's Chat Completion API. It contains various properties, including 'choices', which provides a list of possible completions generated by the model. The 'message' field within each 'choice' represents the generated text that the model has produced, and the 'content' field within the 'message' represents the actual text of the generated response.

One problem with CoT prompts as evidenced in the generated response is the portions of the instructic might not be followed. To avoid such a situation, we can add a reiteration of the task summary at the end. This will help us improve the results.