# **LangChain for LLM Application Development**

## Chapter 1: Model, Prompts and Parsers

* **Models** refers to the language models often be reusable models where default is ChatGPT, or technically the model, GPT 3.5 Turbo.
* **Prompts** refers to the style of creating inputs to pass into the models generally created using f-string with the instructions. For example, translate the text that is delimited by triple backticks into style. LLM can translate the text to different languages or same language in a calmer and respectful tone or changing the dialect like calm pirate would speaks, etc. based on the style provided in the prompt.
* **Chain of Thought Reasoning** It is type of method to provide input to LLM using a React framework which gives an LLM space to think, which can often get to more accurate conclusions.
  + **Thought** is what LLM is thinking
  + **Action** as a keyword to carry the specific action that LLM performs
  + **Observation** to show what it learned from that action
* **Helper function** gets the completion result by calling the model.
* **LangChain** gives an easy set of abstractions for repeat prompt to a model and get parsed output. Prerequisite is to install OpenAI python library and OpenAI secret key that can be generated from [this link](https://platform.openai.com/account/api-keys).
* **Model Temperature** default 0.7, setting temperature to 0 makes the output a little bit less random.
* **ChatOpenAI** is used to create chat object providing model and it’s temperature. Typically this is abstraction for the ChatGPT API endpoint.
* **ChatPromptTemplate** used for repeatedly use input prompts.
  + LangChain provides built-in prompts for some common operations, such as summarization, or question answering, or connecting to SQL databases, or connecting to different APIs.
  + This can also be custom built using f-string where inputs can be identified using backticks. This gives advantage over f-string by wrapping general structure from the f-string typically for sophisticated application with long prompts asking the LLM to first solve the problem, and then have the output in a certain format.
* **Parsers - langchain.output\_parsers** involves taking the output provided by models and parsing it into a more usable structured format. It involves multiple classes and functions that instructs the LLM to use these specific keywords, thought, action, and observation, then this prompt can be coupled with a parser to extract out the text that has been tagged with these specific keywords.
  + **ResponseSchema** specifying details on what to parse from output text i.e. name and description of each output tag. has a function
  + **StructuredOutputParser** instruct the LLM to generate its output in a certain format, such as using specific keywords mentioned through ResponseSchema Objects. For example, output in form of JSON/Dictionary with tags specified in the input instruction for classification.
  + **get\_format\_instructions()** used to get the prompt for output parsing after ResponseSchema is specified.
  + **parse(output\_text)** used to get output in specific format