# 1)What is Langchain?

* LangChain is a framework for developing applications powered by language models.
* Based on 2 main principles:

1. Be data-aware: connect a language model to other sources of data
2. Be agentic: Allow a language model to interact with its environment

# Value proposition of langchain:

1. Components: LangChain provides modular abstractions for the components neccessary to work with language models. LangChain also has collections of implementations for all these abstractions. The components are designed to be easy to use, regardless of whether you are using the rest of the LangChain framework or not.
2. Use-Case Specific Chains: Chains can be thought of as assembling these components in particular ways in order to best accomplish a particular use case. These are intended to be a higher level interface through which people can easily get started with a specific use case. These chains are also designed to be customizable.

# Main components of LangChain:

1. Schema
2. Models
3. Prompts
4. Indexes
5. Memory
6. Chains
7. Agents

# 1)Schema:

When working with LLMs the main way of interacting with them is giving them a message then them giving a response back in the form of text so that will be the main mode of interaction with them. LC is built around this by simplifying a lot of interfaces to be “text in,text out”.

**ChatMessages Schema:**

A user facing LLM is expected to be presented as a chatbot so the interface has to be chat based.

A lot of LLMs have the following subcategories of messages:

SystemChatMessages: The messages which are meant to be instructions to the chatbot before the conversation begins. For example: “You are a pirate from the 1800s, only communicate with the user in pirate speak.”

HumanChatMessages: The messages sent from the user to the chatbot.

AIChatMessages: The messages sent from the chatbot back to the user.

**Document Schema:**

A piece of unstructed data which can be loaded as an object. Usually has 2 parts: page\_content and metadata.

# 2) Models:

3 main types of models provided by LangChain:

a)LLMs

b)Chat Models

c) Text Embedding models

LLMs-> Text input in, Text input out. One time message thing

Chat Models -> ChatMessages as input and return a chatmessage. Usually have more structured APIs.

Text Embedding models-> Converts text into floats(embeddings)

# 3) Prompts:

A "prompt" refers to the input to the model. This input is rarely hard coded, but rather is often constructed from multiple components. A PromptTemplate is responsible for the construction of this input. LangChain provides several classes and functions to make constructing and working with prompts easy.

**PromptValue**

The class representing an input to a model.

**Prompt Templates**

The class in charge of constructing a PromptValue.

**Example Selectors**

Often times it is useful to include examples in prompts. These examples can be hardcoded, but it is often more powerful if they are dynamically selected.

**Output Parsers**

Language models (and Chat Models) output text. But many times you may want to get more structured information than just text back. This is where output parsers come in. Output Parsers are responsible for (1) instructing the model how output should be formatted, (2) parsing output into the desired formatting (including retrying if necessary).

Example:

A screen shot of a computer

Description automatically generated

# 4)Indexes:

Indexes refer to ways to structure documents so that LLMs can best interact with them.

The most common way that indexes are used in chains is in a "retrieval" step. This step refers to taking a user's query and returning the most relevant documents. We draw this distinction because an index can be used for other things besides retrieval, and retrieval can use other logic besides an index to find relevant documents. We therefor have a concept of a "Retriever" interface - this is the interface that most chains work with.

Most of the time when we talk about indexes and retrieval we are talking about indexing and retrieving unstructured data (like text documents).  For interacting with structured data (SQL tables, etc) or APIs, please see the corresponding use case sections for links to relevant functionality. The primary index and retrieval types supported by LangChain are currently centered around vector databases, and therefore a lot of the functionality we dive deep on those topics.

**Document Loaders**

Classes responsible for loading documents from various sources. Example: PDFLoaders

**Text Splitters**

Classes responsible for splitting text into smaller chunks.

**VectorStores**

The most common type of index. One that relies on embeddings.Example: FAISS,ChromaDB

**Retrievers**

Interface for fetching relevant documents to combine with language models.

# 5)Memory:

Memory is the concept of storing and retrieving data in the process of a conversation. There are two main methods:

1. Based on input, fetch any relevant pieces of data
2. Based on the input and output, update state accordingly

There are two main types of memory: short term and long term.

Short term memory generally refers to how to pass data in the context of a singular conversation (generally is previous ChatMessages or summaries of them).

Long term memory deals with how to fetch and update information between conversations.

# 6) Chains

Chains is an incredibly generic concept which returns to a sequence of modular components (or other chains) combined in a particular way to accomplish a common use case.

The most commonly used type of chain is an LLMChain, which combines a PromptTemplate, a Model, and Guardrails to take user input, format it accordingly, pass it to the model and get a response, and then validate and fix (if necessary) the model output.

7)Agents

Some applications will require not just a predetermined chain of calls to LLMs/other tools, but potentially an unknown chain that depends on the user's input. In these types of chains, there is a “agent” which has access to a suite of tools. Depending on the user input, the agent can then decide which, if any, of these tools to call.

We split the documentation into the following sections:

**Tools**

How language models interact with other resources.

**Agents**

The language model that drives decision making.

**Toolkits**

Sets of tools that when used together can accomplish a specific task.

**Agent Executor**

The logic for running agents with tools.