

Take a scenario, let's say that we want to explore about Oracle Cloud Infrastructure AI Foundations course.

"Tell us about Oracle Cloud Infrastructure AI Foundations course."

"who should or can take this AI Foundations course?"

"List all the modules-- list all the modules of AI Foundations course."

" who are the instructors for the AI Foundations course?"

Now let us turn to some other course, say, let's turn to a **security professional course**.

"list all the modules of the OCI Security Professional course."

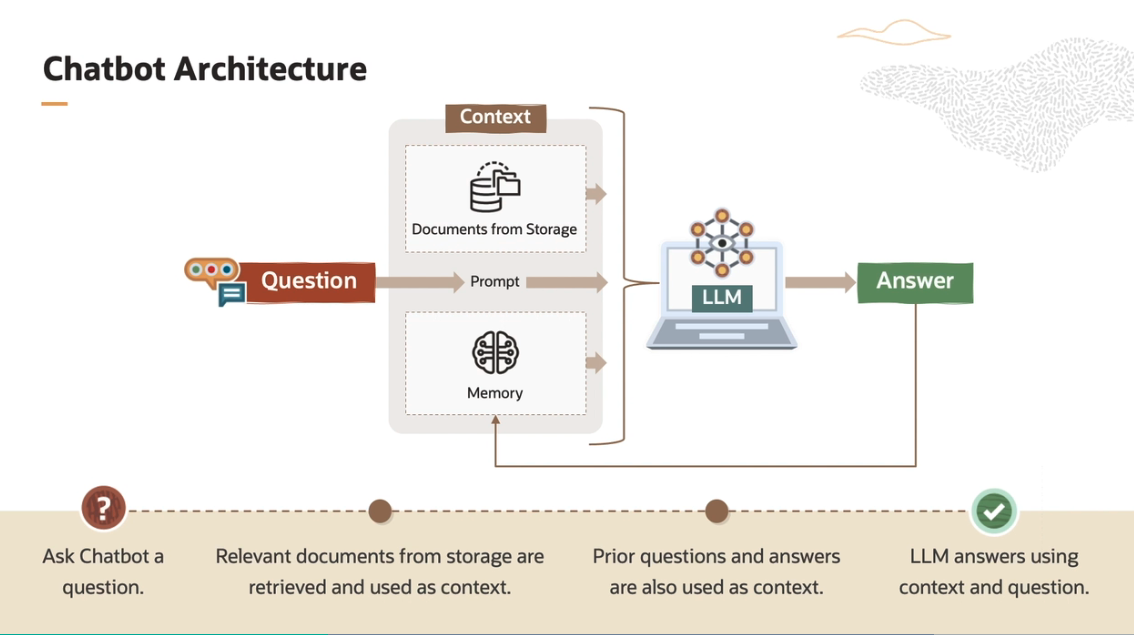
"what is the exam format of the OCI Security Professional Course?"

"What is the difference between OCI Load Balancer and OCI Web Firewall?"

"**Load Balancer** distributes network traffic across multiple servers, improving performance and preventing server overload. It allows for even distribution of workload and helps to maintain availability in the event of server failure. Whereas **Web Firewall** controls internet traffic going in and out of your web applications and prevents malicious traffic from reaching them."

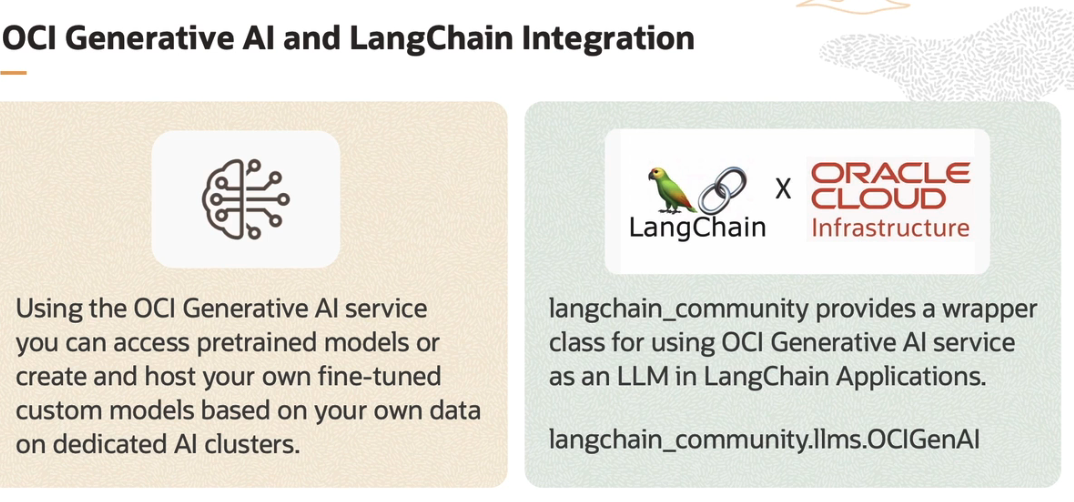
"One key difference is that the Load Balancer primarily focuses on performance optimization through traffic distribution, while the Web Firewall is primarily focused on securing web traffic against various attacks."

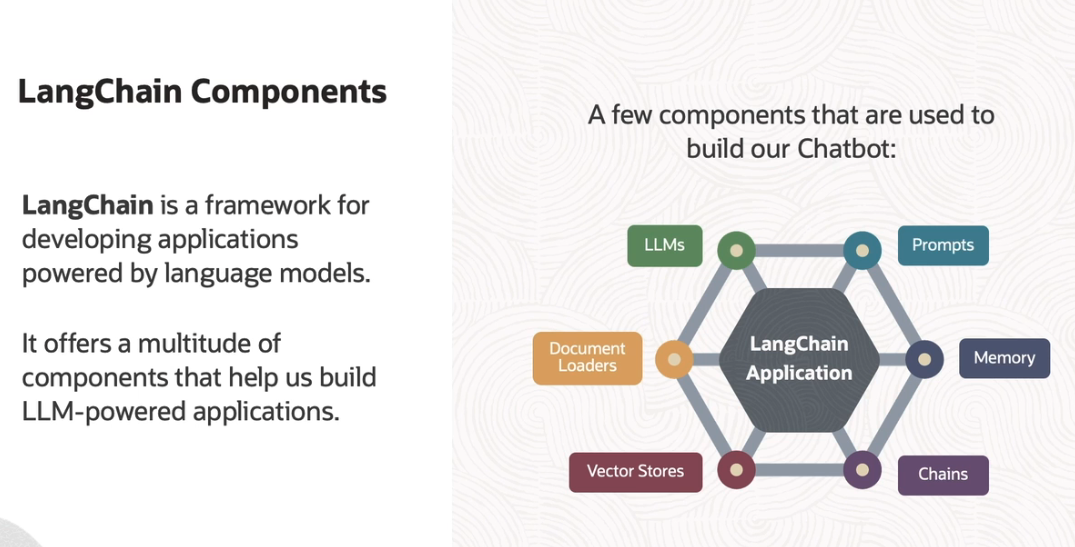




The workflow of a chatbot begins with a question from a user. Sometimes, question alone is not sufficient to arrive at the right answer. We may like to add information, or instructions, or both through a prompt. We may also provide additional context by fetching relevant documents to our question, and include these in our prompt. We may also consider including a set of prior questions and answers that we have stored in our memory.

We provide all these to LLM as input. LLM responds back with an answer, and we update memory with the last question and answer.

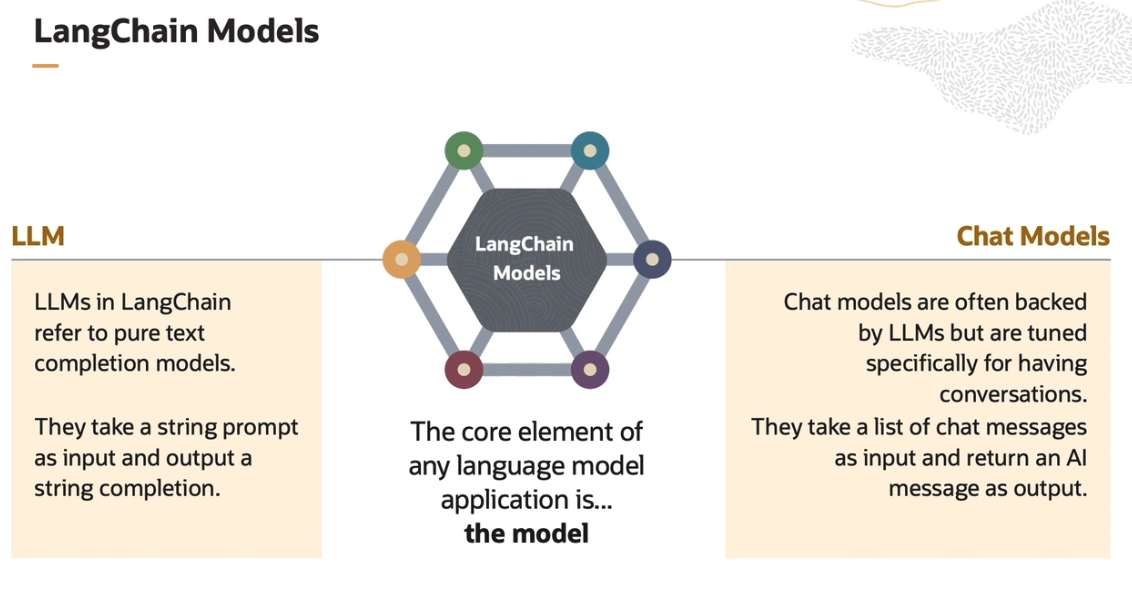




These components are easily exchangeable as well. For example, we can switch between, say, one LLM with another LLM with minimal code changes.



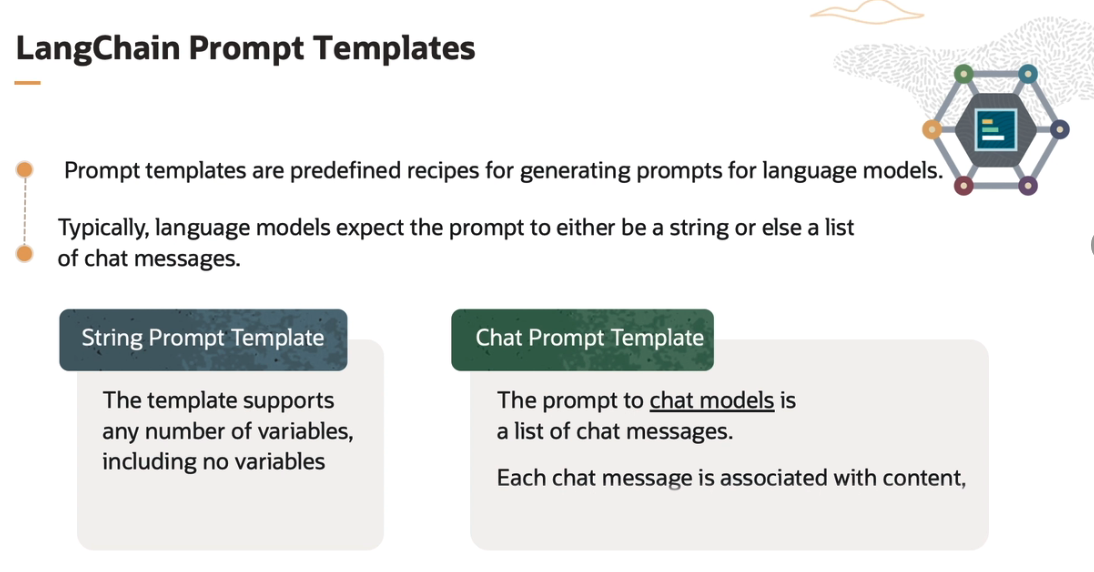
The heart of the LLM applications is the large language model itself. There are two main types of models that LangChain integrates with, LLMs and chat models.

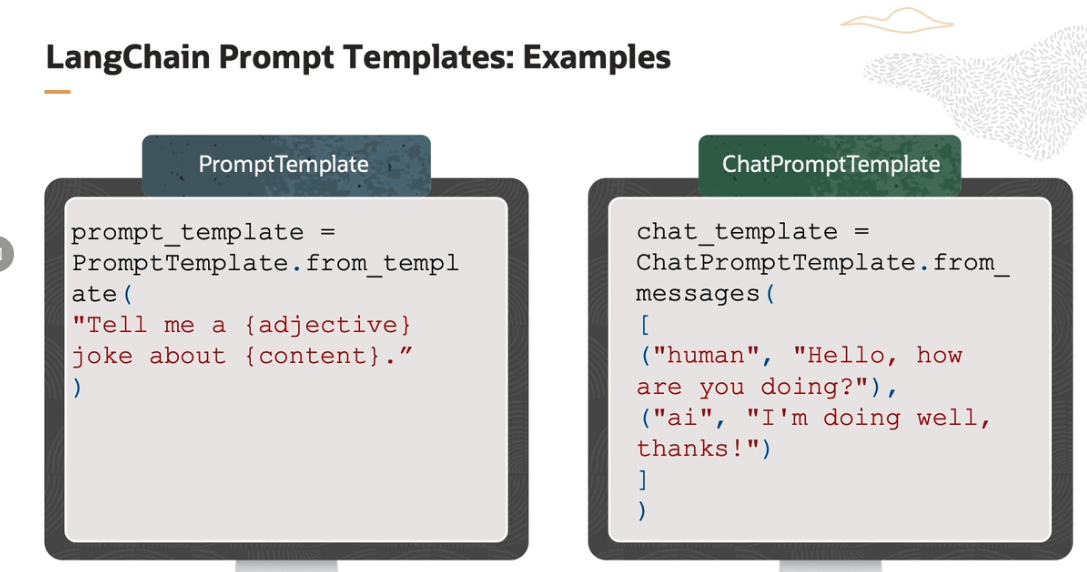


These are defined by their input and output types. LLMs in LangChain refer to pure text completion models. They take a string prompt as input and output a string completion. For example, we may input a question, which is a string, and LLM will return an answer that will also be a string.

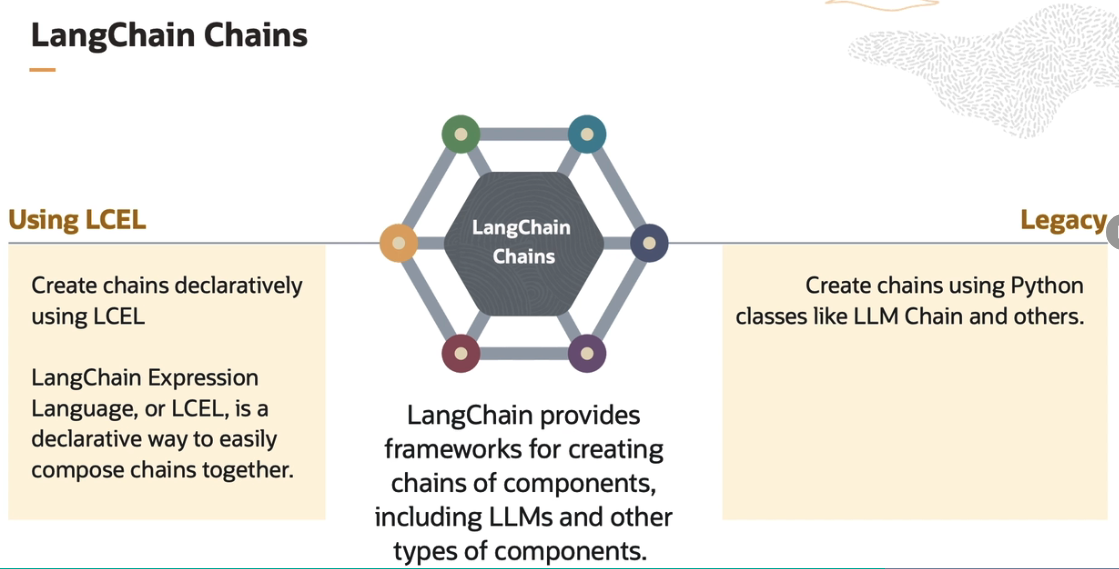
**Chat models** are often backed by LLMs but tuned specifically for having conversations. The conversation is an exchange of messages. For example, a question may go in as a user message, and LLM response can be an AI message. Chat models take a list of chat messages as input, and they return a message as output. For example, when we are asking a question to LLM, the input will be a list that includes the latest question and prior questions and answers as list elements.

\* LangChain has pre-built classes that we will use to create prompts. These are called as prompt templates.





First example is of a string prompt template. Here we replace a variable, say adjective and content with actual input. For example, we may say that, tell me a funny joke about politics. The other example is of a chart template. It is composed of a list of chat messages, each having a role and content. As you can see that there could be roles like human or AI and more.



LangChain provides framework for creating chains of component, including LLMs and other type of components. We can compose chains in two ways.

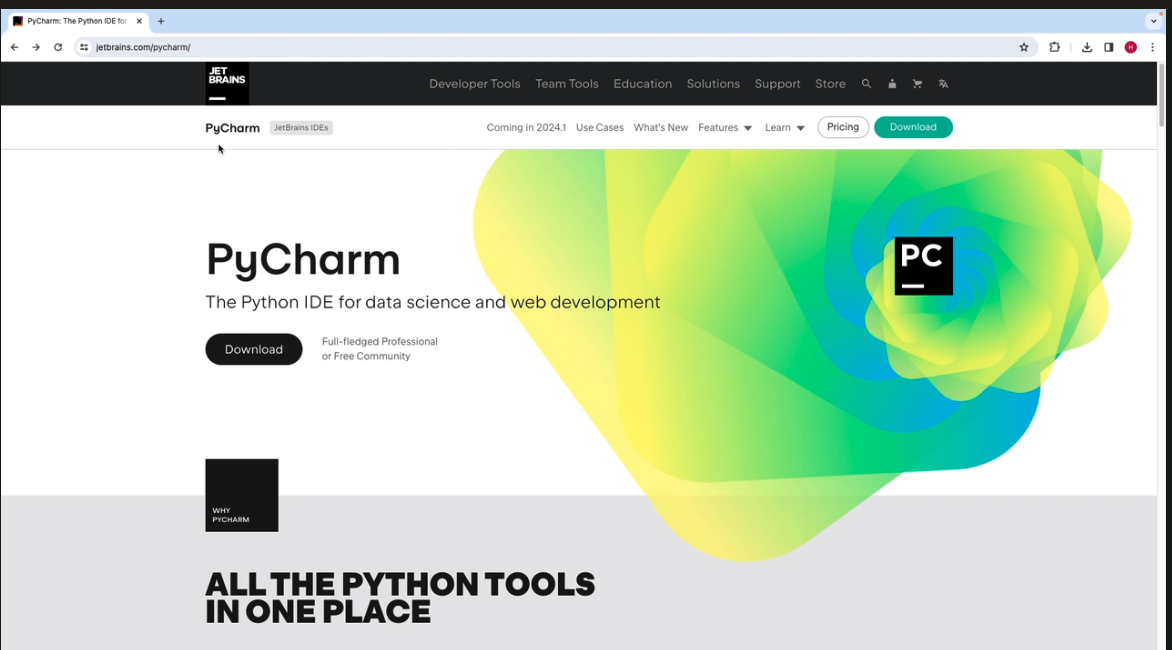
We can use LangChain Expression Language, or LCEL. It is a declarative and preferred way to create chains. Or we can create chains using LangChain Python classes like LLM Chain.



We need to follow a few steps to set up our Python development environment. We have chosen to install JetBrains PyCharm Community Edition IDE, but you can choose any other Python IDE. Next, we create a new project in PyCharm. When we create a new project, we can choose to create a virtual environment. This helps us manage dependencies at a project level.

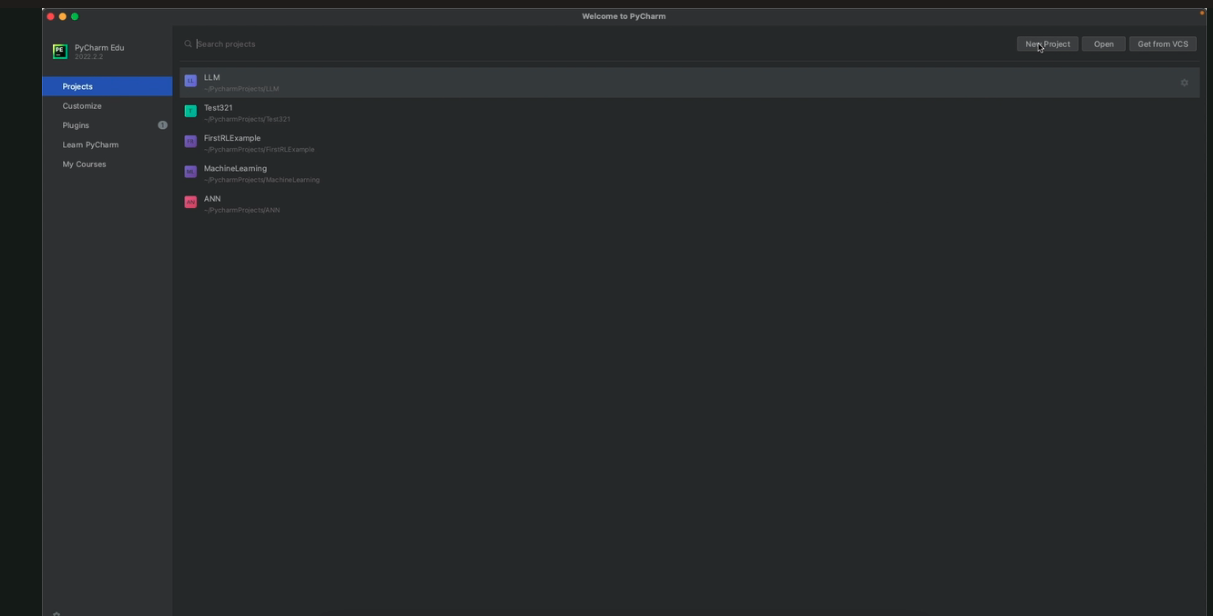
Next, we install necessary Python packages. In this case, we will install LangChain, OCI, Oracle IDEs, and a few others. Next, we generate config file and key file using OCI IAM and download these to our local machine, and keep this in [? .ocr ?] folder of the home directory. Finally, we write our code and run it.

**Setup of Python development environment**

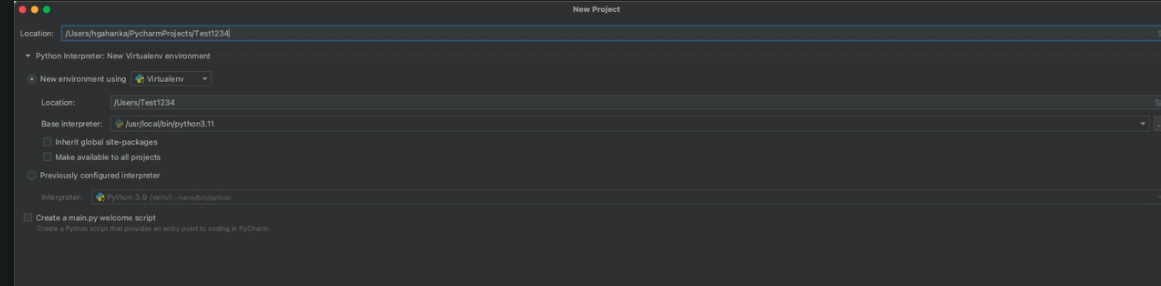


click on Download and choose to download either PyCharm Professional or PyCharm Community Edition, which is totally free.

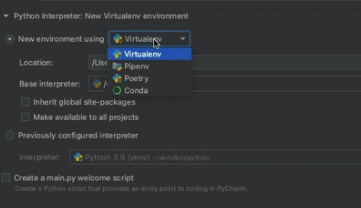
open it



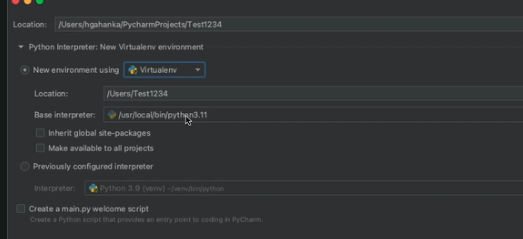
create a new project, give it a name for example, our Test1234.

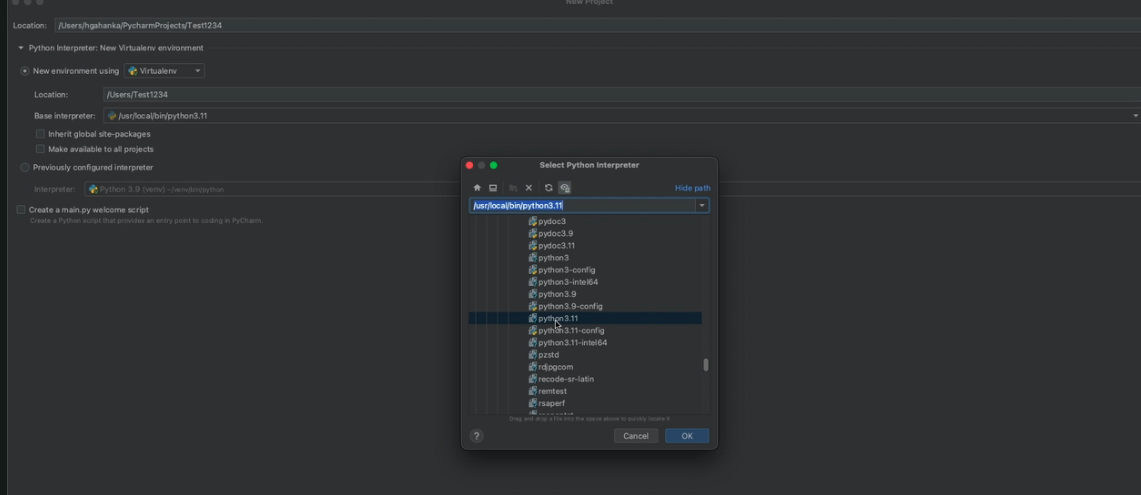


We can select a virtual environment also. So there are multiple options available over here for choosing the virtual environment.

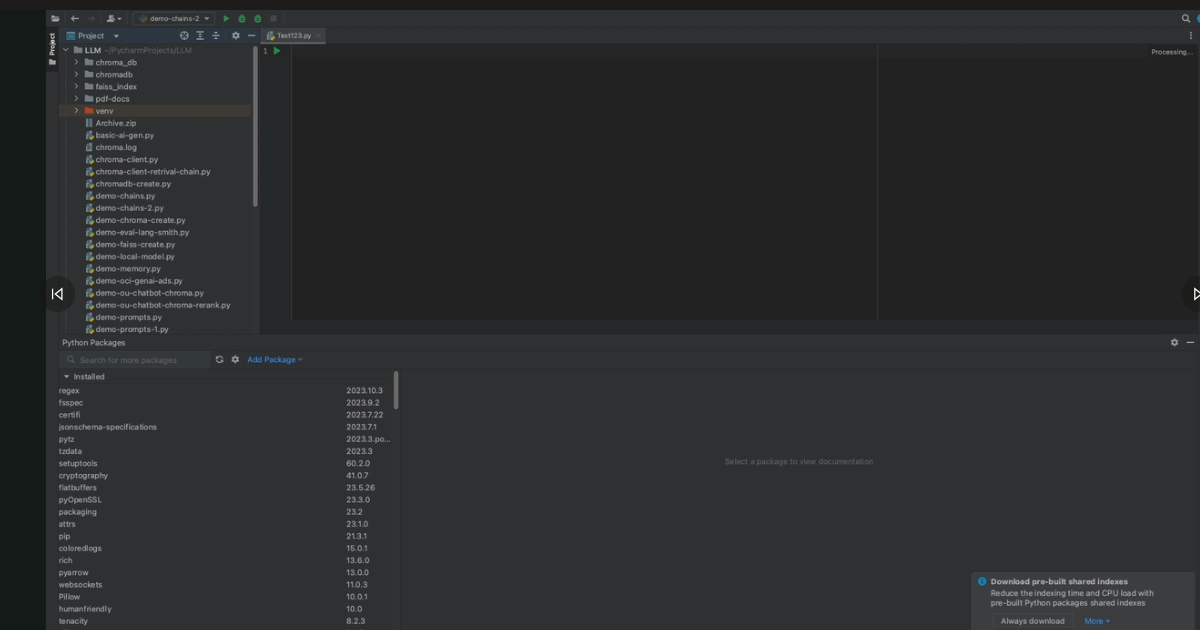


We can choose the Python interpreter. So you should have a Python interpreter installed on your local machine, and it can be chosen by clicking on this particular button over here.



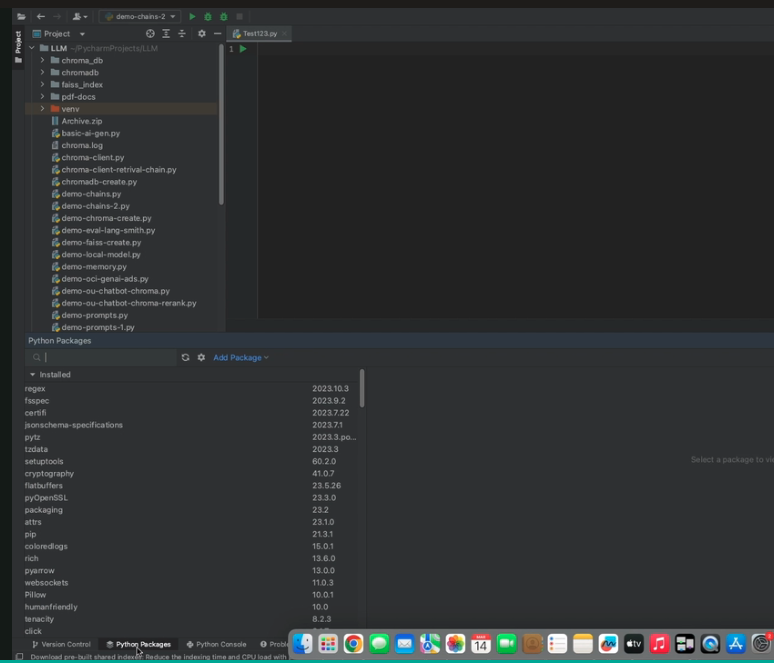


finally, what we can do is we can just go and create this particular project. click create button at the bottom.



You can click on that tab. And whatever packages are required for this particular project, they could be installed by searching for it. So in our case, we will require packages OCI, Oracle IDEs, LangChain, LangSmith, Chroma DB, Wise, Pydantic, Streamlit, and PyPDF.

we can install all those packages from this particular tab, which is called as Python packages.



once we have installed all the necessary packages, we can start creating the actual code files.

