You can download the `requirements.txt` for this course from the workspace of this lab. `File --> Open...`

L2: Create Agents to Research and Write an Article

In this lesson, you will be introduced to the foundational concepts of multi-agent systems and get an overview of the crewAI framework.

The libraries are already installed in the classroom. If you're running this notebook on your own machine, you can install the following:

```
!pip install crewai==0.28.8 crewai_tools==0.1.6
langchain_community==0.0.29
```

In [ ]:
```
# Warning control
import warnings
warnings.filterwarnings('ignore')
```

- Import from the crewAI libray.

In [ ]:
```
from crewai import Agent, Task, Crew
```

- As a LLM for your agents, you'll be using OpenAI's `gpt-3.5-turbo`.

Optional Note: crewAI also allow other popular models to be used as a LLM for your Agents. You can see some of the examples at the bottom of the notebook.

In [ ]:
```
import os
from utils import get_openai_api_key

openai_api_key = get_openai_api_key()
os.environ["OPENAI_MODEL_NAME"] = 'gpt-3.5-turbo'
```

Creating Agents

- Define your Agents, and provide them a `role`, `goal` and `backstory`.
- It has been seen that LLMs perform better when they are role playing.

Agent: Planner

Note: The benefit of using *multiple strings* :

```
varname = "line 1 of text"
          "line 2 of text"
```

versus the *triple quote docstring*:

```
varname = """line 1 of text
             line 2 of text
          """
```

is that it can avoid adding those whitespaces and newline characters, making it better formatted to be passed to the LLM.

In [ ]:
```
planner = Agent(
    role="Content Planner",
    goal="Plan engaging and factually accurate content on {topic}",
    backstory="You're working on planning a blog article "
              "about the topic: {topic}."
              "You collect information that helps the "
              "audience learn something "
              "and make informed decisions. "
              "Your work is the basis for "
              "the Content Writer to write an article on this topic.",
    allow_delegation=False,
        verbose=True
)
```

Agent: Writer

In [ ]:
```
writer = Agent(
    role="Content Writer",
    goal="Write insightful and factually accurate "
         "opinion piece about the topic: {topic}",
    backstory="You're working on a writing "
              "a new opinion piece about the topic: {topic}. "
              "You base your writing on the work of "
              "the Content Planner, who provides an outline "
              "and relevant context about the topic. "
              "You follow the main objectives and "
              "direction of the outline, "
              "as provide by the Content Planner. "
              "You also provide objective and impartial insights "
              "and back them up with information "
              "provide by the Content Planner. "
              "You acknowledge in your opinion piece "
              "when your statements are opinions "
              "as opposed to objective statements.",
    allow_delegation=False,
    verbose=True
)
```

Agent: Editor

In [ ]:
```
editor = Agent(
    role="Editor",
    goal="Edit a given blog post to align with "
         "the writing style of the organization. ",
    backstory="You are an editor who receives a blog post "
              "from the Content Writer. "
              "Your goal is to review the blog post "
              "to ensure that it follows journalistic best practices,"
              "provides balanced viewpoints "
              "when providing opinions or assertions, "
              "and also avoids major controversial topics "
              "or opinions when possible.",
    allow_delegation=False,
    verbose=True
)
```

Creating Tasks

- Define your Tasks, and provide them a `description`, `expected_output` and `agent`.

Task: Plan

In [ ]:
```
plan = Task(
    description=(
        "1. Prioritize the latest trends, key players, "
            "and noteworthy news on {topic}.\n"
        "2. Identify the target audience, considering "
            "their interests and pain points.\n"
        "3. Develop a detailed content outline including "
            "an introduction, key points, and a call to action.\n"
        "4. Include SEO keywords and relevant data or sources."
    ),
    expected_output="A comprehensive content plan document "
        "with an outline, audience analysis, "
        "SEO keywords, and resources.",
    agent=planner,
)
```

Task: Write

In [ ]:
```
write = Task(
    description=(
        "1. Use the content plan to craft a compelling "
            "blog post on {topic}.\n"
        "2. Incorporate SEO keywords naturally.\n"
                "3. Sections/Subtitles are properly named "
            "in an engaging manner.\n"
        "4. Ensure the post is structured with an "
            "engaging introduction, insightful body, "
            "and a summarizing conclusion.\n"
        "5. Proofread for grammatical errors and "
            "alignment with the brand's voice.\n"
    ),
    expected_output="A well-written blog post "
        "in markdown format, ready for publication, "
        "each section should have 2 or 3 paragraphs.",
    agent=writer,
)
```

Task: Edit

In [ ]:
```
edit = Task(
    description=("Proofread the given blog post for "
                 "grammatical errors and "
                 "alignment with the brand's voice."),
    expected_output="A well-written blog post in markdown format, "
                    "ready for publication, "
                    "each section should have 2 or 3 paragraphs.",
    agent=editor
)
```

Creating the Crew

- Create your crew of Agents
- Pass the tasks to be performed by those agents.
    - **Note**: *For this simple example*, the tasks will be performed sequentially (i.e they are dependent on each other), so the *order* of the task in the list *matters*.
- `verbose=2` allows you to see all the logs of the execution.

In [ ]:
```
crew = Crew(
    agents=[planner, writer, editor],
    tasks=[plan, write, edit],
    verbose=2
)
```

Running the Crew

Note: LLMs can provide different outputs for they same input, so what you get might be different than what you see in the video.

In [ ]:
```
result = crew.kickoff(inputs={"topic": "Artificial Intelligence"})
```

- Display the results of your execution as markdown in the notebook.

In [ ]:
```
from IPython.display import Markdown
Markdown(result)
```

Try it Yourself

- Pass in a topic of your choice and see what the agents come up with!

In [ ]:
```
topic = "YOUR TOPIC HERE"
result = crew.kickoff(inputs={"topic": topic})
```

In [ ]:
```
Markdown(result)
```

Other Popular Models as LLM for your Agents

Hugging Face (HuggingFaceHub endpoint)

```
from langchain_community.llms import HuggingFaceHub

llm = HuggingFaceHub(
    repo_id="HuggingFaceH4/zephyr-7b-beta",
    huggingfacehub_api_token="<HF_TOKEN_HERE>",
    task="text-generation",
)

### you will pass "llm" to your agent function
```

Mistral API

```
OPENAI_API_KEY=your-mistral-api-key
OPENAI_API_BASE=https://api.mistral.ai/v1
OPENAI_MODEL_NAME="mistral-small"
```

Cohere

```
from langchain_community.chat_models import ChatCohere
# Initialize language model
os.environ["COHERE_API_KEY"] = "your-cohere-api-key"
llm = ChatCohere()

### you will pass "llm" to your agent function
```

For using Llama locally with Ollama and more, checkout the crewAI documentation on Connecting to any LLM.

In [ ]:

In [ ]:

In [ ]: