Homework 2

Let's create a social media account for your agent

Setup your agent

```
# 📦 Install Required Packages
!pip install langchain-google-genai langchain-core langchain-experimental
!pip install yfinance
```

```
Requirement already satisfied: langchain-google-genai in /usr/local/lib/python3.12/dist-packages (4.2.0)
Requirement already satisfied: langchain-core in /usr/local/lib/python3.12/dist-packages (1.2.7)
Requirement already satisfied: langchain-experimental in /usr/local/lib/python3.12/dist-packages (0.4.1)
Requirement already satisfied: filetype<2.0.0,>=1.2.0 in /usr/local/lib/python3.12/dist-packages (from langchain-google-genai) (1.2.0)
Requirement already satisfied: google-genai<2.0.0,>=1.56.0 in /usr/local/lib/python3.12/dist-packages (from langchain-google-genai) (1.60.0)
Requirement already satisfied: pydantic<3.0.0,>=2.0.0 in /usr/local/lib/python3.12/dist-packages (from langchain-google-genai) (2.12.3)
Requirement already satisfied: jsonpatch<2.0.0,>=1.33.0 in /usr/local/lib/python3.12/dist-packages (from langchain-core) (1.33)
Requirement already satisfied: langsmith<1.0.0,>=0.3.45 in /usr/local/lib/python3.12/dist-packages (from langchain-core) (0.6.6)
Requirement already satisfied: packaging<26.0.0,>=23.2.0 in /usr/local/lib/python3.12/dist-packages (from langchain-core) (25.0)
Requirement already satisfied: pyyaml<7.0.0,>=5.3.0 in /usr/local/lib/python3.12/dist-packages (from langchain-core) (6.0.3)
Requirement already satisfied: tenacity!=8.4.0,<10.0.0,>=8.1.0 in /usr/local/lib/python3.12/dist-packages (from langchain-core) (9.1.2)
Requirement already satisfied: typing-extensions<5.0.0,>=4.7.0 in /usr/local/lib/python3.12/dist-packages (from langchain-core) (4.15.0)
Requirement already satisfied: uuid-utils<1.0,>=0.12.0 in /usr/local/lib/python3.12/dist-packages (from langchain-core) (0.14.0)
Requirement already satisfied: langchain-community<1.0.0,>=0.4.0 in /usr/local/lib/python3.12/dist-packages (from langchain-experimental) (0.4.1)
Requirement already satisfied: anyio<5.0.0,>=4.8.0 in /usr/local/lib/python3.12/dist-packages (from google-genai<2.0.0,>=1.56.0->langchain-google-genai
Requirement already satisfied: google-auth<3.0.0,>=2.47.0 in /usr/local/lib/python3.12/dist-packages (from google-auth[requests]<3.0.0,>=2.47.0->google
Requirement already satisfied: httpx<1.0.0,>=0.28.1 in /usr/local/lib/python3.12/dist-packages (from google-genai<2.0.0,>=1.56.0->langchain-google-gena
Requirement already satisfied: requests<3.0.0,>=2.28.1 in /usr/local/lib/python3.12/dist-packages (from google-genai<2.0.0,>=1.56.0->langchain-google-g
Requirement already satisfied: websockets<15.1.0,>=13.0.0 in /usr/local/lib/python3.12/dist-packages (from google-genai<2.0.0,>=1.56.0->langchain-googl
Requirement already satisfied: distro<2,>=1.7.0 in /usr/local/lib/python3.12/dist-packages (from google-genai<2.0.0,>=1.56.0->langchain-google-genai) (
Requirement already satisfied: sniffio in /usr/local/lib/python3.12/dist-packages (from google-genai<2.0.0,>=1.56.0->langchain-google-genai) (1.3.1)
Requirement already satisfied: jsonpointer>=1.9 in /usr/local/lib/python3.12/dist-packages (from jsonpatch<2.0.0,>=1.33.0->langchain-core) (3.0.0)
Requirement already satisfied: langchain-classic<2.0.0,>=1.0.0 in /usr/local/lib/python3.12/dist-packages (from langchain-community<1.0.0,>=0.4.0->lang
Requirement already satisfied: SQLAlchemy<3.0.0,>=1.4.0 in /usr/local/lib/python3.12/dist-packages (from langchain-community<1.0.0,>=0.4.0->langchain-e
Requirement already satisfied: aiohttp<4.0.0,>=3.8.3 in /usr/local/lib/python3.12/dist-packages (from langchain-community<1.0.0,>=0.4.0->langchain-expe
Requirement already satisfied: dataclasses-json<0.7.0,>=0.6.7 in /usr/local/lib/python3.12/dist-packages (from langchain-community<1.0.0,>=0.4.0->langc
Requirement already satisfied: pydantic-settings<3.0.0,>=2.10.1 in /usr/local/lib/python3.12/dist-packages (from langchain-community<1.0.0,>=0.4.0->lan
Requirement already satisfied: httpx-sse<1.0.0,>=0.4.0 in /usr/local/lib/python3.12/dist-packages (from langchain-community<1.0.0,>=0.4.0->langchain-ex
Requirement already satisfied: numpy>=1.26.2 in /usr/local/lib/python3.12/dist-packages (from langchain-community<1.0.0,>=0.4.0->langchain-experimental
Requirement already satisfied: orjson>=3.9.14 in /usr/local/lib/python3.12/dist-packages (from langsmith<1.0.0,>=0.3.45->langchain-core) (3.11.5)
Requirement already satisfied: requests-toolbelt>=1.0.0 in /usr/local/lib/python3.12/dist-packages (from langsmith<1.0.0,>=0.3.45->langchain-core) (1.0
Requirement already satisfied: zstandard>=0.23.0 in /usr/local/lib/python3.12/dist-packages (from langsmith<1.0.0,>=0.3.45->langchain-core) (0.25.0)
Requirement already satisfied: annotated-types>=0.6.0 in /usr/local/lib/python3.12/dist-packages (from pydantic<3.0.0,>=2.0.0->langchain-google-genai)
Requirement already satisfied: pydantic-core==2.41.4 in /usr/local/lib/python3.12/dist-packages (from pydantic<3.0.0,>=2.0.0->langchain-google-genai) (
Requirement already satisfied: typing-inspection>=0.4.2 in /usr/local/lib/python3.12/dist-packages (from pydantic<3.0.0,>=2.0.0->langchain-google-genai
Requirement already satisfied: aiohappyeyeballs>=2.5.0 in /usr/local/lib/python3.12/dist-packages (from aiohttp<4.0.0,>=3.8.3->langchain-community<1.0.
Requirement already satisfied: aiosignal>=1.4.0 in /usr/local/lib/python3.12/dist-packages (from aiohttp<4.0.0,>=3.8.3->langchain-community<1.0.0,>=0.4
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.12/dist-packages (from aiohttp<4.0.0,>=3.8.3->langchain-community<1.0.0,>=0.4.0-
Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/python3.12/dist-packages (from aiohttp<4.0.0,>=3.8.3->langchain-community<1.0.0,>=0.
Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/python3.12/dist-packages (from aiohttp<4.0.0,>=3.8.3->langchain-community<1.0.0,>=
Requirement already satisfied: propcache>=0.2.0 in /usr/local/lib/python3.12/dist-packages (from aiohttp<4.0.0,>=3.8.3->langchain-community<1.0.0,>=0.4
Requirement already satisfied: yarl<2.0,>=1.17.0 in /usr/local/lib/python3.12/dist-packages (from aiohttp<4.0.0,>=3.8.3->langchain-community<1.0.0,>=0.
Requirement already satisfied: idna>=2.8 in /usr/local/lib/python3.12/dist-packages (from anyio<5.0.0,>=4.8.0->google-genai<2.0.0,>=1.56.0->langchain-g
Requirement already satisfied: marshmallow<4.0.0,>=3.18.0 in /usr/local/lib/python3.12/dist-packages (from dataclasses-json<0.7.0,>=0.6.7->langchain-co
Requirement already satisfied: typing-inspect<1,>=0.4.0 in /usr/local/lib/python3.12/dist-packages (from dataclasses-json<0.7.0,>=0.6.7->langchain-comm
Requirement already satisfied: pyasn1-modules>=0.2.1 in /usr/local/lib/python3.12/dist-packages (from google-auth<3.0.0,>=2.47.0->google-auth[requests]
Requirement already satisfied: rsa<5,>=3.1.4 in /usr/local/lib/python3.12/dist-packages (from google-auth<3.0.0,>=2.47.0->google-auth[requests]<3.0.0,>
Requirement already satisfied: certifi in /usr/local/lib/python3.12/dist-packages (from httpx<1.0.0,>=0.28.1->google-genai<2.0.0,>=1.56.0->langchain-go
Requirement already satisfied: httpcore==1.* in /usr/local/lib/python3.12/dist-packages (from httpx<1.0.0,>=0.28.1->google-genai<2.0.0,>=1.56.0->langch
Requirement already satisfied: h11>=0.16 in /usr/local/lib/python3.12/dist-packages (from httpcore==1.*->httpx<1.0.0,>=0.28.1->google-genai<2.0.0,>=1.5
Requirement already satisfied: langchain-text-splitters<2.0.0,>=1.1.0 in /usr/local/lib/python3.12/dist-packages (from langchain-classic<2.0.0,>=1.0.0-
Requirement already satisfied: python-dotenv>=0.21.0 in /usr/local/lib/python3.12/dist-packages (from pydantic-settings<3.0.0,>=2.10.1->langchain-commu
Requirement already satisfied: charset_normalizer<4,>=2 in /usr/local/lib/python3.12/dist-packages (from requests<3.0.0,>=2.28.1->google-genai<2.0.0,>=
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.12/dist-packages (from requests<3.0.0,>=2.28.1->google-genai<2.0.0,>=1.56.0
Requirement already satisfied: greenlet>=1 in /usr/local/lib/python3.12/dist-packages (from SQLAlchemy<3.0.0,>=1.4.0->langchain-community<1.0.0,>=0.4.0
Requirement already satisfied: pyasn1<0.7.0,>=0.6.1 in /usr/local/lib/python3.12/dist-packages (from pyasn1-modules>=0.2.1->google-auth<3.0.0,>=2.47.0-
Requirement already satisfied: mypy-extensions>=0.3.0 in /usr/local/lib/python3.12/dist-packages (from typing-inspect<1,>=0.4.0->dataclasses-json<0.7.0
```

```
# 🔑 API Key Setup
from google.colab import userdata
GEMINI_VERTEX_API_KEY = userdata.get('VERTEX_API_KEY')
assert GEMINI_VERTEX_API_KEY, "Please set your VERTEX_API_KEY in Colab secrets"
```

```
# 🤖 Initialize Gemini LLM
from langchain_google_genai import ChatGoogleGenerativeAI

llm = ChatGoogleGenerativeAI(
        model="gemini-2.5-flash",
        api_key=GEMINI_VERTEX_API_KEY,
        vertexai=True,
        temperature=0
)
```

Create a moltbook account for your agent

```
# This function is used to encode your student id to ensure the privacy

def encode_student_id(student_id: int) -> str:
    """
    Reversibly encode a student ID using an affine cipher.

    Args:
        student_id (int): Original student ID (non-negative integer)

    Returns:
        str: Encoded ID as a zero-padded string
    """
    if student_id < 0:
        raise ValueError("student_id must be non-negative")

    M = 10**8
    a = 137
    b = 911

    encoded = (a * student_id + b) % M
    return f"{encoded:08d}"
```

```
# Before creating your agent please encode your student id using this function and replace XXXX by the encoded number
encode_student_id(32463728)
```

```
'47531647'
```

```
# Please use the encoded student id
!curl -X POST https://www.moltbook.com/api/v1/agents/register \
  -H "Content-Type: application/json" \
  -d '{"name": "BaoNguyen_367126387", "description": "Va"}'
```

```
{"success":true,"message":"Welcome to Moltbook! 🦞","agent":{"id":"53364afb-c1b2-4efd-a508-4ba76cb1d4ab","name":"BaoNguyen_367126387","api_key":"moltboo
```

- After sucessfully register, you will see a notification of the format:

"success":true,"message":"Welcome to Moltbook! 🦞","agent":"id":"...","name":"...","api_key":"...", "claim_url": "..."

- Please save your the api key as MOLTBOOK_API_KEY in the Secrets section of your Colab.
- Then you complete the registration by accessing the claim_url and follow the guideline in the url.

```
# Create a tool set to interact with moltbook

import os
import requests
from langchain_core.tools import tool

MOLTBOOK_API_KEY = userdata.get('MOLTBOOK_API_KEY')
BASE_URL = "https://www.moltbook.com/api/v1"

HEADERS = {
    "Authorization": f"Bearer {MOLTBOOK_API_KEY}",
    "Content-Type": "application/json"
}

# ---------- FEED ----------
@tool
def get_feed(sort: str = "new", limit: int = 10) -> dict:
    """Fetch Moltbook feed."""
    r = requests.get(
        f"{BASE_URL}/feed",
        headers=HEADERS,
        params={"sort": sort, "limit": limit},
        timeout=15
    )
    return r.json()

# ---------- SEARCH ----------
@tool
def search_moltbook(query: str, type: str = "all") -> dict:
    """Semantic search Moltbook posts, comments, agents."""
    r = requests.get(
        f"{BASE_URL}/search",
        headers=HEADERS,
        params={"q": query, "type": type},
        timeout=15
    )
    return r.json()

# ---------- POST ----------
@tool
def create_post(submolt: str, title: str, content: str) -> dict:
    """Create a new text post."""
    payload = {
            "submolt": submolt,
            "title": title,
            "content": content
        }
        r = requests.post(
            f"{BASE_URL}/posts",
            headers=HEADERS,
            json=payload,
            timeout=15
        )
        return r.json()

# ---------- COMMENT ----------
@tool
def comment_post(post_id: str, content: str) -> dict:
        """Comment on a post."""
        r = requests.post(
            f"{BASE_URL}/posts/{post_id}/comments",
            headers=HEADERS,
            json={"content": content},
            timeout=15
        )
        return r.json()

# ---------- VOTE ----------
@tool
def upvote_post(post_id: str) -> dict:
        """Upvote a post."""
        r = requests.post(
            f"{BASE_URL}/posts/{post_id}/upvote",
            headers=HEADERS,
            timeout=15
        )
        return r.json()
```

```
SYSTEM_PROMPT = """
You are a Moltbook AI agent.

Your purpose:
- Discover valuable AI / ML / agentic system discussions
- Engage thoughtfully and selectively
- NEVER spam
- NEVER repeat content
- Respect rate limits

Rules:
1. Before posting, ALWAYS search Moltbook to avoid duplication.
2. Only comment if you add new insight.
3. Upvote only genuinely useful content.
4. If uncertain, do nothing.
5. Prefer short, clear, professional language.
6. If a human gives an instruction, obey it exactly.

Available tools:
- get_feed
- search_moltbook
- create_post
- comment_post
- upvote_post
"""
```

A simple agent to interact with moltbook

```
from langchain_google_genai import ChatGoogleGenerativeAI
from langchain_core.messages import ToolMessage
import time
import json
from datetime import datetime
from typing import Any

def log(section: str, message: str):
        ts = datetime.utcnow().strftime("%H:%M:%S")
        print(f"[{ts}] [{section}] {message}")

def pretty(obj: Any, max_len: int = 800):
        text = json.dumps(obj, indent=2, ensure_ascii=False, default=str)
        return text if len(text) <= max_len else text[:max_len] + "\n...<truncated>"

def moltbook_agent_loop(
        instruction: str | None = None,
        max_turns: int = 8,
        verbose: bool = True,
):
        log("INIT", "Starting Moltbook agent loop")

        llm = ChatGoogleGenerativeAI(
                model="gemini-2.5-flash",
                temperature=0,
                api_key=GEMINI_VERTEX_API_KEY,
                vertexai=True,
        )

        tools = [
                get_feed,
                search_moltbook,
                create_post,
                comment_post,
                upvote_post,
        ]

        agent = llm.bind_tools(tools)

        history = [("system", SYSTEM_PROMPT)]

        if instruction:
                history.append(("human", f"Human instruction: {instruction}"))
                log("HUMAN", instruction)
        else:
                history.append(("human", "Perform your Moltbook heartbeat check."))
                log("HEARTBEAT", "No human instruction - autonomous mode")

        # ================================
        # Main agent loop
        # ================================
        for turn in range(1, max_turns + 1):
                log("TURN", f"Turn {turn}/{max_turns} started")
                turn_start = time.time()

                response = agent.invoke(history)
                history.append(response)

                if verbose:
                        log("LLM", "Model responded")
                        log("LLM.CONTENT", response.content or "<empty>")
                        log("LLM.TOOL_CALLS", pretty(response.tool_calls or []))

                # ============================
                # STOP CONDITION
                # ============================
                if not response.tool_calls:
                        elapsed = round(time.time() - turn_start, 2)
                        log("STOP", f"No tool calls — final answer produced in {elapsed}s")
                        return response.content

                # ============================
                # TOOL EXECUTION
                # ============================
                for i, call in enumerate(response.tool_calls, start=1):
                        tool_name = call["name"]
                        args = call["args"]
                        tool_id = call["id"]

                        log("TOOL", f"[{i}] Calling `{tool_name}`")
                        log("TOOL.ARGS", pretty(args))

                        tool_fn = globals().get(tool_name)
                        tool_start = time.time()

                        try:
                                result = tool_fn.invoke(args)
                                status = "success"
                        except Exception as e:
                                result = {"error": str(e)}
                                status = "error"

                        tool_elapsed = round(time.time() - tool_start, 2)

                        log(
                                "TOOL.RESULT",
                                f"{tool_name} finished ({status}) in {tool_elapsed}s"
                        )

                        if verbose:
                                log("TOOL.OUTPUT", pretty(result))

                        history.append(
                                ToolMessage(
                                        tool_call_id=tool_id,
                                        content=str(result),
                                )
                        )

                turn_elapsed = round(time.time() - turn_start, 2)
                log("TURN", f"Turn {turn} completed in {turn_elapsed}s")

        # ================================
        # MAX TURNS REACHED
```

```
        #  ================================
        log("STOP",  "Max  turns  reached  without  final  answer")
        return  "Agent  stopped  after  reaching  max  turns."
```

```
# You need to complte the tool set so that your agent can find the submolt
moltbook_agent_loop("find submolt named ftec5660")
```

```
/tmp/ipython-input-2868163325.py:9: DeprecationWarning: datetime.datetime.utcnow() is deprecated and scheduled for removal in a future version. Use timez
  ts = datetime.utcnow().strftime("%H:%M:%S")
[08:27:06] [INIT] Starting Moltbook agent loop
[08:27:06] [HUMAN] find submolt named ftec5660
[08:27:06] [TURN] Turn 1/8 started
[08:27:08] [LLM] Model responded
[08:27:08] [LLM.CONTENT] <empty>
[08:27:08] [LLM.TOOL_CALLS] [
  {
    "name": "search_moltbook",
    "args": {
      "query": "ftec5660"
    },
    "id": "2c4ce983-3c47-47bd-b013-1e584e37a74c",
    "type": "tool_call"
  }
]
[08:27:08] [TOOL] [1] Calling `search_moltbook`
[08:27:08] [TOOL.ARGS] {
  "query": "ftec5660"
}
[08:27:09] [TOOL.RESULT] search_moltbook finished (success) in 0.37s
[08:27:09] [TOOL.OUTPUT] {
  "success": false,
  "error": "Search failed"
}
[08:27:09] [TURN] Turn 1 completed in 2.44s
[08:27:09] [TURN] Turn 2/8 started
[08:27:10] [LLM] Model responded
[08:27:10] [LLM.CONTENT] [{'type': 'text', 'text': 'I\'m sorry, I was unable to find a submolt named "ftec5660" using the available tools. The search fun
[08:27:10] [LLM.TOOL_CALLS] []
[08:27:10] [STOP] No tool calls — final answer produced in 1.55s
[{'type': 'text',
  'text': 'I\'m sorry, I was unable to find a submolt named "ftec5660" using the available tools. The search function did not return any results for
that query. It\'s possible that submolts cannot be directly searched for by name with the current tools, or that no such submolt exists.',
  'extras': {'signature':
'CtMEAY89a1+CNMo/RcowHjaL3FIMCk+VWZAlozpuAMC27mp/6vGXlUyUlTQkSr6S+NfgtueFpckA6A6QG05IBsTGvyIgUTD3TvfEk8BkT9PmSvcRFjxpUMgYOMLbi12kHh8ICyWxDi59+FLmHPTe2tRh
```