**Agentic AI Training — Assignments & Theory (Labs 1–11 , Week One)**

**Execution Flow**

User → Planner (JSON step/final) → Executor (tools) → Tools → Executor (EXEC\_RESULT) → Planner (next or done) → END.

**Lab 1 — Agent Foundations**

**Goal**

Run the baseline project and add one capability.

**Tasks**

1. Run the code and capture one successful run.
2. Add a pure Python helper (e.g., string\_cleaner or safe\_eval).
3. Add basic logging (timestamp, prompt, final answer) to a file.

**Deliverables**

* Screenshot/log excerpt of a successful run.
* README with run steps and log location.
* 3–5 bullets on file structure and data flow.

**Rubric**

* Runs cleanly (4)
* Helper function (3)
* Logging present (3)

**Stretch Ideas**

* Add a --trace flag for intermediate reasoning (mock is fine).

**Lab 2 — Tools 101**

**Goal**

Add two deterministic single‑input tools with validation.

**Tasks**

1. Implement two single‑input tools.
2. Validate inputs and sanitize outputs.
3. Write unit tests (success & failure).

**Deliverables**

* tools.py with docstrings for each @tool.
* tests/test\_tools.py with ≥4 tests.
* 1‑paragraph tool contract description.

**Rubric**

* Tools compile & run (4)
* Validation & clean outputs (3)
* Tests pass (3)

**Stretch Ideas**

* Use pydantic/TypedDict to validate inputs.

**Lab 3 — Chains (Prompt → Model → Output)**

**Goal**

Build and compare two chains (one LCEL).

**Tasks**

1. Create LLMChain to summarize 3 sentences into 1.
2. Create LCEL: PromptTemplate | llm | StrOutputParser.
3. Compare outputs & token usage on 3 prompts.

**Deliverables**

* lab3\_chains.py / notebook.
* Table: prompt | chain1 | chain2 | tokens (approx).
* 5–7 sentence reflection.

**Rubric**

* Two working chains (4)
* Comparison table (3)
* Reflection (3)

**Stretch Ideas**

* Vary temperature; discuss variance.

**Lab 4 — Agentic Lifecycle (Multi‑file)**

**Goal**

Document and harden the larger agent project.

**Tasks**

1. Draw lifecycle diagram (prompt→plan→tools→final→logging).
2. Add centralized error handler.
3. Add config (.env/yaml) changing behavior (e.g., model, log level).

**Deliverables**

* Diagram (image/ascii).
* Handler code & controlled failure screenshot.
* Config + rationale paragraph.

**Rubric**

* Clear lifecycle diagram (4)
* Error handling works (3)
* Configurable behavior (3)

**Stretch Ideas**

* Add a simple rate limiter/backoff.

**Lab 5 — Introduction to LangChain**

**Goal**

Move from custom code to LangChain primitives & LCEL.

**Tasks**

1. Wrap prompt→model with ChatOpenAI/PromptTemplate/LLMChain.
2. Build LCEL graph (prompt→model→parser→post‑processor).
3. Show parity against your custom pipeline.

**Deliverables**

* lab5\_langchain\_intro.py / notebook.
* Side‑by‑side outputs for 3 inputs.
* Note on maintainability benefits.

**Rubric**

* LCEL graph runs (4)
* Parity demo (3)
* Explanation quality (3)

**Stretch Ideas**

* Add RunnableMap to compute two outputs in parallel.

**Lab 6 — Tools & Agent (Tool‑Calling)**

**Goal**

Wrap tools and run a tool‑calling agent.

**Tasks**

1. Implement weather, mini\_wiki, suggest\_city\_activities, calculator (single‑input).
2. Bind tools to chat model and answer weather→activities composite question.
3. Handle parsing errors (retry/handler).

**Deliverables**

* lab6\_tools\_agent.py
* Execution trace with at least one tool call.
* Write‑up: why single‑input improves robustness.

**Rubric**

* Tools & binding work (5)
* Trace captured (2)
* Error handling (3)

**Stretch Ideas**

* Fallback answer for unknown cities.

**Lab 7 — Agent Types Overview**

**Goal**

Compare ReAct, Tool‑Using, and Conversational agents.

**Tasks**

1. Implement a demo for each agent type on the same prompt.
2. Log reasoning/steps/memory updates.
3. Write a 1‑page comparison (capabilities, failure modes, when to use).

**Deliverables**

* lab7\_agent\_types.py
* Table: agent | steps | tools | final quality (1–5).
* 1‑page comparison.

**Rubric**

* All three run (4)
* Trace comparison (3)
* Analysis depth (3)

**Stretch Ideas**

* Add a guardrail instruction and compare adherence.

**Lab 8 — Prompt Engineering for Agents**

**Goal**

Engineer prompts that reliably change behavior.

**Tasks**

1. Create three policies: JSON‑only; always use calculator for math; cost‑aware (limit tool calls).
2. For each policy, run 3 queries and show compliance.
3. Handle output parser errors via structured retries or constrained decoding.

**Deliverables**

* lab8\_prompt\_policies.py
* Compliance tables per policy.
* Error‑handling snippet + rationale.

**Rubric**

* Policies implemented (4)
* Evidence of compliance (3)
* Error handling (3)

**Stretch Ideas**

* Add few‑shot examples; measure adherence gains.

**Lab 9 — LangGraph ReAct with Persistence**

**Goal**

Put ReAct on a graph and fix tool‑message ordering.

**Tasks**

1. Implement LangGraph loop with Annotated[list[AnyMessage], add].
2. Add MemorySaver checkpointer; show multi‑turn persistence.
3. Show failing case without add, then fixed case with it.

**Deliverables**

* Lab9\_LangGraph\_ReAct.py
* Console excerpts: failing 400 error then fixed run.
* 6–8 sentence explanation of why add matters.

**Rubric**

* Graph runs w/ tools (4)
* Persistence demo (3)
* Root‑cause explanation (3)

**Stretch Ideas**

* Swap MemorySaver for SQLite/Redis for durability.

**Lab 10 — Memory + Multi‑Tool Orchestration**

**Goal**

Use memory + tool binding + loop for a travel assistant.

**Tasks**

1. Orchestrate weather → activities → (optional) mini\_wiki → calculator.
2. Persist user preferences across turns (same thread\_id).
3. Enforce output contract: 'Final Answer:' and calculator returns only the number.

**Deliverables**

* Lab10\_Memory\_MultiTool\_LangGraph.py / notebook.
* 5‑turn transcript showing preference recall & multi‑tool order.
* Validation snippet/regex enforcing contract.

**Rubric**

* Correct orchestration (5)
* Memory across turns (4)
* Output contract enforced (3)

**Stretch Ideas**

* Add a validator node to repair outputs until they pass.

**Lab 11 — Multi‑Agent: Planner + Executor (MAS)**

**Goal**

Separate planning from execution with two agents and centralized orchestration.

**Tasks**

1. Planner: strict JSON only (done/next\_step/final\_answer), no tools.
2. Executor: tool‑bound; executes next\_step and returns 'EXEC\_RESULT: ...'.
3. LangGraph: planner → executor → tools → executor → planner → … → END.
4. Run: weather → activities → one‑sentence city fact → finalize.

**Deliverables**

* Lab11\_MAS\_Planner\_Executor\_LangGraph.py (or your renamed version).
* Full run showing ≥2 planner cycles and ≥2 executor interactions.
* Diagram of MAS flow + 6–8 sentence rationale for role split.

**Rubric**

* Correct role separation (4)
* Tool routing via Executor (4)
* Planner JSON & finalization (4)

**Stretch Ideas**

* Add an approval node or a decentralized blackboard variant.
* ValueError: Function must have a docstring → add docstrings to each @tool or pass description=.
* OpenAI 400: 'tool message must follow tool\_calls' → use Annotated[list[AnyMessage], add] in state.
* ZeroShotAgent multi‑input error → classic agents expect single‑input tools.
* Model adds words to 'only number' → enforce with system prompt + validator or regex.