1. Basic Concepts of an Agent

An **AI Agent** is a system that uses an LLM (Large Language Model) as a reasoning engine to determine **what actions to take and in what order**.[1] Unlike a standard LLM chatbot that simply outputs text based on input, an Agent "acts" in the world.

The Core Loop (The "Brain"):

1. **Perception:** Receives a task (e.g., "What is the stock price of Apple compared to Microsoft?").
2. **Reasoning:** The LLM decides it needs external data. It selects a "Tool" (e.g., a Search Tool or Stock API).[2]
3. **Action:** It executes the tool.[3]
4. **Observation:** It reads the output of the tool.[4]
5. **Response:** It repeats the loop until it has enough information to answer the user.[5]

---

2. Why do we need Agents?

Standard LLMs have limitations:

- **Static Knowledge:** They only know what they were trained on (cutoff dates).
- **No Math/Logic Guarantee:** They often hallucinate on complex calculations.
- **No External Access:** They cannot browse the live web, check your calendar, or query a database.

Agents solve this by giving LLMs "Hands" (Tools). We need them for complex, multi-step workflows where the path to the solution isn't known beforehand (e.g., "Book a flight for me on a day when it's sunny in London").

Common Implementation Strategies:

1. **ReAct (Reason + Act):** Interleaving thought and action.[6]
2. **Plan and Execute:** Planning all steps first, then executing them.[7]
3. **Function Calling (OpenAI):** Using fine-tuned models trained to output structured tool calls.[8]
4. **Multi-Agent Orchestration:** Different agents (Coder, Reviewer, Manager) working together (e.g., AutoGen, CrewAI).[9]

---

3. ReAct Prompting (Reason + Act)

Concept:

ReAct (Reasoning + Acting) is a strategy where the model generates a Thought (reasoning trace) followed by an Action (tool call), then receives an Observation (tool output).10 This loop continues until the task is done.

Advantages:

- **Reduced Hallucination:** The model grounds its answers in the observations it receives.[11]
- **Human Interpretability:** You can see exactly *why* the agent decided to call a specific tool.
- **Error Recovery:** If a tool fails (e.g., "API Error"), the agent sees this in the "Observation" and can "Think" of a new way to solve the problem.[12]

Code Example (Conceptual using LangChain):

```
Python

from langchain.agents import load_tools, initialize_agent, AgentType
from langchain.llms import OpenAI

# 1. Initialize the LLM
llm = OpenAI(temperature=0)

# 2. Load Tools (e.g., Google Search and Calculator)
tools = load_tools(["serpapi", "llm-math"], llm=llm)

# 3. Initialize the ReAct Agent
agent = initialize_agent(
    tools,
    llm,
    agent=AgentType.ZERO_SHOT_REACT_DESCRIPTION, # Uses ReAct framework
    verbose=True # Shows the Thought/Action/Observation loop
)

# 4. Run the Agent
agent.run("Who is the current CEO of Tesla and what is their age raised to the power of 0.2?")
```

What happens internally (The ReAct Loop):

Thought: I need to find the CEO of Tesla.

Action: Search [CEO of Tesla]

Observation: Elon Musk.

Thought: Now I need his age.

Action: Search [Elon Musk age]

Observation: 52 years old.

Thought: I need to calculate $52^{0.2}$.

Action: Calculator [52^0.2]

Observation: 2.20

Final Answer: Elon Musk is the CEO and his age to the power of 0.2 is 2.20.

---

4. Plan and Execute Strategy

Concept:

Instead of deciding the next step "on the fly" (like ReAct), this strategy separates the workflow into two distinct phases using two different "brains":

1. **The Planner:** An LLM analyzes the user request and generates a comprehensive, step-by-step plan.
2. **The Executor:** An Agent (usually ReAct) takes the plan and executes each step one by one.[13]

Why use it?

- **Complex Tasks:** Better for tasks that require long-term foresight (e.g., writing a software program) where ReAct might get "distracted" or lose context after too many steps.
- **Efficiency:** The Planner ensures the agent doesn't go down rabbit holes.

---

5. OpenAI Functions Strategy

Concept:

OpenAI models (like GPT-4o) are fine-tuned to detect when a function should be called.[14] Instead of outputting free text like "I should search for weather," the model outputs a structured JSON object containing the function name and arguments. This is not a prompting trick (like ReAct); it is a native model capability.

Code Example (OpenAI SDK):

```
Python

import openai
import json

# 1. Define the tool (Function)
tools = [
    {
        "type": "function",
        "function": {
            "name": "get_weather",
            "description": "Get the current weather in a given location",
            "parameters": {
                "type": "object",
                "properties": {
                    "location": {"type": "string", "description": "The city and state, e.g. San Francisco,
CA"},
                    "unit": {"type": "string", "enum": ["celsius", "fahrenheit"]}
                },
                "required": ["location"]
            }
        }
    }
]

# 2. Call the model with the tool definition
response = openai.chat.completions.create(
    model="gpt-4-turbo",
    messages=[{"role": "user", "content": "What's the weather like in Boston?"}],
    tools=tools,
    tool_choice="auto"  # Let the model decide whether to call the function
)

# 3. Check if the model wants to call a function
tool_call = response.choices[0].message.tool_calls[0]
if tool_call:
    print(f"Function Name: {tool_call.function.name}")
    print(f"Arguments: {tool_call.function.arguments}")
    # Output: Function Name: get_weather, Arguments: {"location": "Boston, MA"}
```

---

6. OpenAI Functions vs. LangChain Agents

It is important to understand that these are not mutually exclusive; LangChain often *uses* OpenAI Functions.[15]

Feature	OpenAI Functions	LangChain Agents
Nature	**Model Capability.** A fine-tuning feature of the GPT models to output reliable JSON.	**Framework.** A code library that orchestrates the loop of calling LLMs and executing tools.
Reliability	**High.** The model is trained to output strict structured data (JSON). Rarely generates invalid arguments.	**Variable.** Depends on the prompting strategy. Standard ReAct (text-based) can sometimes output malformed actions that code cannot parse.
Flexibility	**Lower.** Tied specifically to OpenAI's models and API structure.	**High.** Can switch between ReAct, Plan-and-Execute, or OpenAI Functions. Can swap the LLM (e.g., use Claude or Llama).
Role	Provides the "Instruction" (The JSON payload).	Provides the "Loop" (Parses the JSON, executes the Python function, feeds the result back to the LLM).

Summary:

- Use **OpenAI Functions** when you want the most reliable way to get structured data out of an OpenAI model.
- Use **LangChain Agents** (specifically the `OpenAIFunctionsAgent`) to build the actual application that *executes* those functions and manages the conversation history and memory.