OPENAI AGENTS SDK

Quick Reference Cheatsheet

Agentic AI Framework | Python | pip install openai-agents

THE FOUR CORE PRIMITIVES

A	T	H	G
Agents	**Tools**	**Handoffs**	**Guardrails**
LLM + Instructions	Actions & Functions	Agent Delegation	Input/Output Validation

AGENTS

Basic Agent

```
from agents import Agent

agent = Agent(
    name="Math Tutor",
    instructions="Help with math.",
    model="gpt-4.1",
    tools=[calculate],
    output_type=MathResult
)
```

Agent Properties

- `name` - Unique identifier
- `instructions` - System prompt
- `model` - LLM (default: gpt-4.1)
- `tools` - Available actions
- `handoffs` - Delegate agents
- `output_type` - Pydantic model

TOOLS

Function Tool

```
@function_tool
def get_weather(city: str):
    """Get weather."""
    return f"Sunny in {city}"
```

Hosted Tools

- `WebSearchTool`
- `FileSearchTool`
- `CodeInterpreterTool`
- `ImageGenerationTool`

Agent as Tool

```
main = Agent(
    name="Manager",
    tools=[
      expert.as_tool()
    ]
)
```

HANDOFFS

Multi-Agent Handoffs

```
math = Agent(
    name="Math Expert",
    handoff_description="For math"
)
triage = Agent(
    name="Triage",
    instructions="Route to experts.",
    handoffs=[math, history]
)
```

Handoffs vs Tools

Handoffs: Specialist takes over entirely. LLM decides routing. Decentralized.

Agents-as-Tools: Main agent keeps control. Deterministic flow. Centralized.

RUNNING AGENTS

Sync

```
result = Runner.run_sync(
    agent, "Hello"
)
print(result.final_output)
```

Async

```
result = await Runner.run(
    agent, "Hello"
)
print(result.final_output)
```

Streaming

```
async for event in \
  Runner.run_streamed(
    agent, "Hello"
  ):
  print(event.data.delta)
```

```
Agent Loop:  Input → LLM Call → [Final Output? → DONE] | [Handoff? → Switch Agent → Loop] | [Tool Call? → Execute → Loop]
```

Sessions (Multi-turn Memory)

```
from agents.sessions import SQLiteSession
session = SQLiteSession("./db", "user_1")
result = await Runner.run(
    agent, "My name is Alice",
    session=session
)
```

`SQLiteSession` | `SQLAlchemySession` | `EncryptedSession`

Structured Outputs

```
class Result(BaseModel):
    answer: float
    explanation: str

agent = Agent(
    output_type=Result
)
# result.final_output.answer
```

GUARDRAILS

Guardrail Types

Type	When	Use Case
Input	Before agent	Block bad input
Output	After response	Validate quality

Guardrail Example

```
async def safety_check(ctx, agent, input):
    result = await Runner.run(checker, input)
    return GuardrailFunctionOutput(
        tripwire_triggered=not result.is_safe
    )
```

QUICK REFERENCE

Common Patterns

Pattern	Code	Use Case
Basic Run	`Runner.run_sync(agent, "input")`	Simple single turn
With Session	`Runner.run(agent, "input", session=s)`	Multi-turn chat
Function Tool	`@function_tool` decorator	Custom actions
Handoff	`handoffs=[agent1, agent2]`	Route to specialists
Structured	`output_type=PydanticModel`	Typed responses

INSTALLATION

`pip install openai-agents`

`export OPENAI_API_KEY=sk-...`