Quickstart

Create a project and virtual environment

You'll only need to do this once.

```
mkdir my_project
cd my_project
python -m venv .venv
```

Activate the virtual environment

Do this every time you start a new terminal session.

```
source .venv/bin/activate
```

Install the Agents SDK

```
pip install openai-agents # or `uv add openai-agents`, etc
```

Set an OpenAI API key

If you don't have one, follow these instructions to create an OpenAI API key.

```
export OPENAI_API_KEY=sk-...
```

Create your first agent

Agents are defined with instructions, a name, and optional config (such as `model_config`)

```
from agents import Agent

agent = Agent(
    name="Math Tutor",
    instructions="You provide help with math problems. Explain your reasoning at
each step and include examples",
)
```

Add a few more agents

Additional agents can be defined in the same way. `handoff_descriptions` provide additional context for determining handoff routing

```
from agents import Agent

history_tutor_agent = Agent(
    name="History Tutor",
    handoff_description="Specialist agent for historical questions",
    instructions="You provide assistance with historical queries. Explain
important events and context clearly.",
)

math_tutor_agent = Agent(
    name="Math Tutor",
    handoff_description="Specialist agent for math questions",
    instructions="You provide help with math problems. Explain your reasoning at
each step and include examples",
)
```

Define your handoffs

On each agent, you can define an inventory of outgoing handoff options that the agent can choose from to decide how to make progress on their task.

```
triage_agent = Agent(
    name="Triage Agent",
    instructions="You determine which agent to use based on the user's homework
question",
    handoffs=[history_tutor_agent, math_tutor_agent]
)
```

Run the agent orchestration

Let's check that the workflow runs and the triage agent correctly routes between the two specialist agents.

```
from agents import Runner

async def main():
    result = await Runner.run(triage_agent, "What is the capital of France?")
    print(result.final_output)
```

Add a guardrail

You can define custom guardrails to run on the input or output.

```
from agents import GuardrailFunctionOutput, Agent, Runner
from pydantic import BaseModel

class HomeworkOutput(BaseModel):
    is_homework: bool
    reasoning: str

guardrail_agent = Agent(
    name="Guardrail check",
    instructions="Check if the user is asking about homework.",
    output_type=HomeworkOutput,
)

async def homework_guardrail(ctx, agent, input_data):
    result = await Runner.run(guardrail_agent, input_data, context=ctx.context)
    final_output = result.final_output_as(HomeworkOutput)
    return GuardrailFunctionOutput(
        output_info=final_output,
        tripwire_triggered=not final_output.is_homework,
    )
```

Put it all together

Let's put it all together and run the entire workflow, using handoffs and the input guardrail.

```
from agents import Agent, InputGuardrail, GuardrailFunctionOutput, Runner
from pydantic import BaseModel
import asyncio

class HomeworkOutput(BaseModel):
    is_homework: bool
    reasoning: str

guardrail_agent = Agent(
    name="Guardrail check",
    instructions="Check if the user is asking about homework.",
    output_type=HomeworkOutput,
)

math_tutor_agent = Agent(
    name="Math Tutor",
    handoff_description="Specialist agent for math questions",
    instructions="You provide help with math problems. Explain your reasoning at
each step and include examples",
)

history_tutor_agent = Agent(
    name="History Tutor",
    handoff_description="Specialist agent for historical questions",
```

```
    instructions="You provide assistance with historical queries. Explain
important events and context clearly.",
)


async def homework_guardrail(ctx, agent, input_data):
    result = await Runner.run(guardrail_agent, input_data, context=ctx.context)
    final_output = result.final_output_as(HomeworkOutput)
    return GuardrailFunctionOutput(
        output_info=final_output,
        tripwire_triggered=not final_output.is_homework,
    )

triage_agent = Agent(
    name="Triage Agent",
    instructions="You determine which agent to use based on the user's homework
question",
    handoffs=[history_tutor_agent, math_tutor_agent],
    input_guardrails=[
        InputGuardrail(guardrail_function=homework_guardrail),
    ],
)

async def main():
    result = await Runner.run(triage_agent, "who was the first president of the
united states?")
    print(result.final_output)

    result = await Runner.run(triage_agent, "what is life")
    print(result.final_output)

if __name__ == "__main__":
    asyncio.run(main())
```

View your traces

To review what happened during your agent run, navigate to the Trace viewer in the OpenAI Dashboard to view traces of your agent runs.

Next steps

Learn how to build more complex agentic flows:

- Learn about how to configure Agents.
- Learn about running agents.
- Learn about tools, guardrails and models.