Running agents

You can run agents via the `Runner` class. You have 3 options:

1. `Runner.run()`, which runs async and returns a `RunResult`.
2. `Runner.run_sync()`, which is a sync method and just runs `.run()` under the hood.
3. `Runner.run_streamed()`, which runs async and returns a `RunResultStreaming`. It calls the LLM in streaming mode, and streams those events to you as they are received.

```
from agents import Agent, Runner

async def main():
    agent = Agent(name="Assistant", instructions="You are a helpful assistant")

    result = await Runner.run(agent, "Write a haiku about recursion in
programming.")
    print(result.final_output)
    # Code within the code,
    # Functions calling themselves,
    # Infinite loop's dance.
```

Read more in the results guide.

The agent loop

When you use the run method in `Runner`, you pass in a starting agent and input. The input can either be a string (which is considered a user message), or a list of input items, which are the items in the OpenAI Responses API.

The runner then runs a loop:

1. We call the LLM for the current agent, with the current input.
2. The LLM produces its output.
    a. If the LLM returns a `final_output`, the loop ends and we return the result.
    b. If the LLM does a handoff, we update the current agent and input, and re-run the loop.
    c. If the LLM produces tool calls, we run those tool calls, append the results, and re-run the loop.
3. If we exceed the `max_turns` passed, we raise a `MaxTurnsExceeded` exception.

> **Note**
>
> The rule for whether the LLM output is considered as a "final output" is that it produces text output with the desired type, and there are no tool calls.

Streaming

Streaming allows you to additionally receive streaming events as the LLM runs. Once the stream is done, the `RunResultStreaming` will contain the complete information about the run, including all the new outputs produces. You can call `.stream_events()` for the streaming events. Read more in the streaming guide.

Run config

The `run_config` parameter lets you configure some global settings for the agent run:

- `model`: Allows setting a global LLM model to use, irrespective of what `model` each Agent has.
- `model_provider`: A model provider for looking up model names, which defaults to OpenAI.
- `model_settings`: Overrides agent-specific settings. For example, you can set a global `temperature` or `top_p`.
- `input_guardrails`, `output_guardrails`: A list of input or output guardrails to include on all runs.
- `handoff_input_filter`: A global input filter to apply to all handoffs, if the handoff doesn't already have one. The input filter allows you to edit the inputs that are sent to the new agent. See the documentation in `Handoff.input_filter` for more details.
- `tracing_disabled`: Allows you to disable tracing for the entire run.
- `trace_include_sensitive_data`: Configures whether traces will include potentially sensitive data, such as LLM and tool call inputs/outputs.
- `workflow_name`, `trace_id`, `group_id`: Sets the tracing workflow name, trace ID and trace group ID for the run. We recommend at least setting `workflow_name`. The group ID is an optional field that lets you link traces across multiple runs.
- `trace_metadata`: Metadata to include on all traces.

Conversations/chat threads

Calling any of the run methods can result in one or more agents running (and hence one or more LLM calls), but it represents a single logical turn in a chat conversation. For example:

1. User turn: user enter text
2. Runner run: first agent calls LLM, runs tools, does a handoff to a second agent, second agent runs more tools, and then produces an output.

At the end of the agent run, you can choose what to show to the user. For example, you might show the user every new item generated by the agents, or just the final output. Either way, the user might then ask a followup question, in which case you can call the run method again.

You can use the base `RunResultBase.to_input_list()` method to get the inputs for the next turn.

```
async def main():
    agent = Agent(name="Assistant", instructions="Reply very concisely.")

    with trace(workflow_name="Conversation", group_id=thread_id):
        # First turn
        result = await Runner.run(agent, "What city is the Golden Gate Bridge
in?")
        print(result.final_output)
        # San Francisco

        # Second turn
        new_input = result.to_input_list() + [{"role": "user", "content": "What
state is it in?"}]
        result = await Runner.run(agent, new_input)
        print(result.final_output)
        # California
```

Exceptions

The SDK raises exceptions in certain cases. The full list is in `agents.exceptions`. As an overview:

- `AgentsException` is the base class for all exceptions raised in the SDK.
- `MaxTurnsExceeded` is raised when the run exceeds the `max_turns` passed to the run methods.
- `ModelBehaviorError` is raised when the model produces invalid outputs, e.g. malformed JSON or using non-existent tools.
- `UserError` is raised when you (the person writing code using the SDK) make an error using the SDK.
- `InputGuardrailTripwireTriggered`, `OutputGuardrailTripwireTriggered` is raised when a guardrail is tripped.