Guardrails

Guardrails run *in parallel* to your agents, enabling you to do checks and validations of user input. For example, imagine you have an agent that uses a very smart (and hence slow/expensive) model to help with customer requests. You wouldn't want malicious users to ask the model to help them with their math homework. So, you can run a guardrail with a fast/cheap model. If the guardrail detects malicious usage, it can immediately raise an error, which stops the expensive model from running and saves you time/money.

There are two kinds of guardrails:

1. Input guardrails run on the initial user input
2. Output guardrails run on the final agent output

Input guardrails

Input guardrails run in 3 steps:

1. First, the guardrail receives the same input passed to the agent.
2. Next, the guardrail function runs to produce a `GuardrailFunctionOutput`, which is then wrapped in an `InputGuardrailResult`
3. Finally, we check if `.tripwire_triggered` is true. If true, an `InputGuardrailTripwireTriggered` exception is raised, so you can appropriately respond to the user or handle the exception.

Note

Input guardrails are intended to run on user input, so an agent's guardrails only run if the agent is the *first* agent. You might wonder, why is the `guardrails` property on the agent instead of passed to `Runner.run`? It's because guardrails tend to be related to the actual Agent - you'd run different guardrails for different agents, so colocating the code is useful for readability.

Output guardrails

Output guardrails run in 3 steps:

1. First, the guardrail receives the same input passed to the agent.

2. Next, the guardrail function runs to produce a `GuardrailFunctionOutput`, which is then wrapped in an `OutputGuardrailResult`
3. Finally, we check if `.tripwire_triggered` is true. If true, an `OutputGuardrailTripwireTriggered` exception is raised, so you can appropriately respond to the user or handle the exception.

Note

Output guardrails are intended to run on the final agent output, so an agent's guardrails only run if the agent is the *last* agent. Similar to the input guardrails, we do this because guardrails tend to be related to the actual Agent - you'd run different guardrails for different agents, so colocating the code is useful for readability.

Tripwires

If the input or output fails the guardrail, the Guardrail can signal this with a tripwire. As soon as we see a guardrail that has triggered the tripwires, we immediately raise a `{Input,Output}GuardrailTripwireTriggered` exception and halt the Agent execution.

Implementing a guardrail

You need to provide a function that receives input, and returns a `GuardrailFunctionOutput`. In this example, we'll do this by running an Agent under the hood.

```python
from pydantic import BaseModel
from agents import (
    Agent,
    GuardrailFunctionOutput,
    InputGuardrailTripwireTriggered,
    RunContextWrapper,
    Runner,
    TResponseInputItem,
    input_guardrail,
)

class MathHomeworkOutput(BaseModel):
    is_math_homework: bool
    reasoning: str

guardrail_agent = Agent(  1
    name="Guardrail check",
    instructions="Check if the user is asking you to do their math homework.",
    output_type=MathHomeworkOutput,
)
```

```
@input_guardrail
async def math_guardrail( (2)
    ctx: RunContextWrapper[None], agent: Agent, input: str |
list[TResponseInputItem]
) -> GuardrailFunctionOutput:
    result = await Runner.run(guardrail_agent, input, context=ctx.context)

    return GuardrailFunctionOutput(
        output_info=result.final_output, (3)
        tripwire_triggered=result.final_output.is_math_homework,
    )


agent = Agent(  (4)
    name="Customer support agent",
    instructions="You are a customer support agent. You help customers with their
questions.",
    input_guardrails=[math_guardrail],
)

async def main():
    # This should trip the guardrail
    try:
        await Runner.run(agent, "Hello, can you help me solve for x: 2x + 3 =
11?")
        print("Guardrail didn't trip - this is unexpected")

    except InputGuardrailTripwireTriggered:
        print("Math homework guardrail tripped")
```

1. We'll use this agent in our guardrail function.
2. This is the guardrail function that receives the agent's input/context, and returns the result.
3. We can include extra information in the guardrail result.
4. This is the actual agent that defines the workflow.

Output guardrails are similar.

```
from pydantic import BaseModel
from agents import (
    Agent,
    GuardrailFunctionOutput,
    OutputGuardrailTripwireTriggered,
    RunContextWrapper,
    Runner,
    output_guardrail,
)
class MessageOutput(BaseModel): (1)
    response: str
```

```python
class MathOutput(BaseModel): (2)
    reasoning: str
    is_math: bool

guardrail_agent = Agent(
    name="Guardrail check",
    instructions="Check if the output includes any math.",
    output_type=MathOutput,
)

@output_guardrail
async def math_guardrail( (3)
    ctx: RunContextWrapper, agent: Agent, output: MessageOutput
) -> GuardrailFunctionOutput:
    result = await Runner.run(guardrail_agent, output.response,
context=ctx.context)

    return GuardrailFunctionOutput(
        output_info=result.final_output,
        tripwire_triggered=result.final_output.is_math,
    )

agent = Agent( (4)
    name="Customer support agent",
    instructions="You are a customer support agent. You help customers with their
questions.",
    output_guardrails=[math_guardrail],
    output_type=MessageOutput,
)

async def main():
    # This should trip the guardrail
    try:
        await Runner.run(agent, "Hello, can you help me solve for x: 2x + 3 =
11?")
        print("Guardrail didn't trip - this is unexpected")

    except OutputGuardrailTripwireTriggered:
        print("Math output guardrail tripped")
```

1. This is the actual agent's output type.
2. This is the guardrail's output type.
3. This is the guardrail function that receives the agent's output, and returns the result.
4. This is the actual agent that defines the workflow.