

PydanticAI: the AI Agent Framework Winner



ArjanCodes
287K subscribers

Join

Subscribe

1.9K



Share

Ask

Download



31,881 views Aug 29, 2025 #python #softwaredesign #arjancodes
Check out <https://www.squarespace.com/arjancodes> to save 10% off your first purchase of a website or domain using code ARJANCODES.

Pydantic AI lets you integrate large language models like GPT-5 **directly** into your Python applications. In this video, I'll show you how to build a real AI-powered agent using **Pydantic AI** and **Python**. The agent acts as a **healthcare triage assistant**, dynamically pulling patient data, assessing urgency, and responding intelligently—with fully validated, structured outputs.

- 0:00 Intro
- 0:25 What is Pydantic AI?
- 1:27 Example Healthcare Triage Assistant
- 3:10 Agent Scaffolding
- 5:31 Sponsored section
- 7:11 Creating the Agent

- 8:51 Running the Agent
- 10:18 Adding Context with Dynamic Prompts
- 12:22 Defining Tools
- 13:47 Other Pydantic Features
- 15:15 Final Thoughts

ArjanCodes / examples

Type to search

Code Issues 3 Pull requests 2 Actions Projects Security Insights

Files

main

Go to file

github

.vscode

2022

2023

2024

2025

examples / 2025 / pydanticai /

egges Added code example e415018 · last month History

Name	Last commit message	Last commit date
..		
main.py	Added code example	last month
pyproject.toml	Added code example	last month
uv.lock	Added code example	last month

```
recording > main.py > main
1 import asyncio
2 from dataclasses import dataclass
3 from typing import Any
4
5 from dotenv import load_dotenv
6
7 # Load environment variables from .env
8 load_dotenv()
9
10
11 # Mock database
12 @dataclass
13 class Patient:
```



```
recording > main.py > Patient
12 @dataclass
13 class Patient:
14     id: int
15     name: str
16     vitals: dict[str, Any]
17
18
19 PATIENT_DB = {
20     42: Patient(
21         id=42, name="John Doe", vitals={"heart_rate": 72, "b
22     },
23     43: Patient(
24         id=43, name="Jane Smith", vitals={"heart_rate": 65,
25     },
26 }
27
```



```

20
21 ie="John Doe", vitals={"heart_rate": 72, "blood_pressure": "1
22
23
24 ie="Jane Smith", vitals={"heart_rate": 65, "blood_pressure":
25

```



```

recording > main.py > DatabaseConn > patient_name
28
29 class DatabaseConn:
30     async def patient_name(self, id: int) -> str:
31         patient = PATIENT_DB.get(id)
32         return patient.name if patient else "Unknown Patient"
33
34     async def latest_vitals(self, id: int) -> dict[str, Any]:
35         patient = PATIENT_DB.get(id)
36         return patient.vitals if patient else {"heart_rate":
37

```



```

37
38
39 async def main() -> None:
40     db = DatabaseConn()
41     patient_name = await db.patient_name(42)
42     latest_vitals = await db.latest_vitals(42)
43
44     print(f"Patient Name: {patient_name}")
45     print(f"Latest Vitals: {latest_vitals}")
46

```



```

zsh x main.py
• → recording uv run main.py
Patient Name: John Doe
Latest Vitals: {'heart_rate': 72, 'blood_pressure': '120/80'}
○ → recording

```

This works. Next, let us set up some agent scaffolding by using a class

```

recording > main.py > TriageDependencies
29 class DatabaseConn:
30     async def patient_name(self, id: int) -> str:
31
32         return patient.name if patient else "Unknown Patient"
33
34     async def latest_vitals(self, id: int) -> dict[str, Any]:
35         patient = PATIENT_DB.get(id)
36         return patient.vitals if patient else {"heart_rate": 0, "blood_pre
37
38 @dataclass
39 class TriageDependencies:
40     patient_id: int
41     db: DatabaseConn
42
43 async def main() -> None:
44     db = DatabaseConn()
45 → patient_name = await db.patient_name(42)
46     latest_vitals = await db.latest_vitals(42)

```



recording > main.py > ...

```
1 import asyncio
2 from dataclasses import dataclass
3 from typing import Any
4 from pydantic import BaseModel
5
```

recording > main.py > TriageOutput

```
37     return patient.vitals if patient else {"heart_rate": 0, "blood_pre
38
39 @dataclass
40 class TriageDependencies:
41     patient_id: int
42     db: DatabaseConn
43
44 class TriageOutput(BaseModel):
45     response_text: str
46     escalate: bool
47     urgency: int
48
49 async def main() -> None:
50     db = DatabaseConn()
51     patient_name = await db.patient_name(42)
52     latest_vitals = await db.latest_vitals(42)
53
```



zsh

main.py 2

recording > main.py > ...

```
1 import asyncio
2 from dataclasses import dataclass
3 from typing import Any
4 from pydantic import BaseModel, Field
5 import os
6
```

recording > main.py > TriageDependencies

```
37     return patient.vitals if patient else {"heart_rate":
38
39
40 @dataclass
41 class TriageDependencies:
42     patient_id: int
43     db: DatabaseConn
44
45
46 class TriageOutput(BaseModel):
47     response_text: str = Field(description="Message to the patient")
48     escalate: bool = Field(description="Whether to escalate the case to a
49     urgency: int = Field(description="Urgency level of the case")
50
51
52 async def main() -> None:
53     db = DatabaseConn()
```



```

44  output(BaseModel):
45      text: str = Field(description="Message to the patient")
46      bool = Field(description="Whether to escalate the case to a human nurse")
47      int = Field(description="Urgency level of the case")
48
49  n() -> None:
50      baseConn()
51      ame = await db.patient_name(42)

```

Next, we will now define our agent that will return a structured output for our Triage agent

```

recording > main.py > ...
1  import asyncio
2  from dataclasses import dataclass
3  from typing import Any
4
5  from dotenv import load_dotenv
6  from pydantic import BaseModel, Field
7  from pydantic_ai import Agent
8

```

```

recording > main.py > ...
47  class TriageOutput(BaseModel):
49      escalate: bool = Field(description="Whether to escalate the case to a
50      urgency: int = Field(description="Urgency level of the case")
51
52  triage_agent = Agent(
53      "openai:gpt-4o",
54      deps_type=TriageDependencies,
55      output_type=TriageOutput,
56      system_prompt=[
57          "You are a triage assistant helping patients."
58          "Provide clear advice and assess urgency."
59      ]
60  )
61  async def main() -> None:

```



```

recording > main.py > ...
54      deps_type=TriageDependencies,
55      output_type=TriageOutput,
56      system_prompt=[
57          "You are a triage assistant helping patients."
58          "Provide clear advice and assess urgency."
59      ]
60  )
61  async def main() -> None:
62      db = DatabaseConn()
63      patient_name = await db.patient_name(42)
64      latest_vitals = await db.latest_vitals(42)
65
66      print(f"Patient Name: {patient_name}")
67      print(f"Latest Vitals: {latest_vitals}")
68
69
70  if __name__ == "__main__":


```




We can now update this main function as below


```
recording > main.py > main

61 )
62
63
64 async def main() -> None:
65     deps = TriageDependencies(patient_id=42, db=DatabaseConn())
66     result = await triage_agent.run(
67         "What should I do if I have a headache and a fever?", deps=deps
68     )
69
70     print(result.output)
71
72
73 if __name__ == "__main__":
74     asyncio.run(main())
75
```



```
• → recording uv run main.py
response_text='For a headache and fever, you can consider taking an over-the-counter
pain reliever like acetaminophen or ibuprofen to help reduce the fever and alleviate
the headache. Make sure to stay hydrated by drinking plenty of fluids and rest in a c
omfortable, cool environment. If your symptoms persist for more than a couple of days
or if you have other symptoms like a rash, severe headache, or neck stiffness, it ma
y be advisable to consult a healthcare professional. However, if your fever is very h
igh or if you have severe symptoms, you should seek medical attention promptly.' esca
late=False urgency=2
○ → recording
```




Next, let us have our agent use some of the patient's information to make its decision

```
recording > main.py > ...

1 import asyncio
2 from dataclasses import dataclass
3 from typing import Any
4
5 from dotenv import load_dotenv
6 from pydantic import BaseModel, Field
7 from pydantic_ai import Agent, RunContext
8
```

```
recording > main.py > add_patient_name

56     output_type=TriageOutput,
57     system_prompt=(
58         "You are a triage assistant helping patients."
59         "Provide clear advice and assess urgency."
60     ),
61 )
62
63 @triage_agent.system_prompt
64 async def add_patient_name(ctx: RunContext[TriageDependencies]) -> str:
65     patient_name = await ctx.deps.db.patient_name(ctx.deps.patient_id)
66     return f"The patient name is: {patient_name}."
67
68
69 async def main() -> None:
```



```
recording > main.py > ...
47 class TriageOutput(BaseModel):
50     urgency: int = Field(description="Urgency level of the case")
51
52
53 triage_agent = Agent(
54     "openai:gpt-4o",
55     deps_type=TriageDependencies,
56     output_type=TriageOutput,
57     system_prompt=[
58         "You are a triage assistant helping patients."
59         "Provide clear advice and assess urgency."
60         "Always mention the patient's name when available."
61     ],
62 )
63
```



```
• → recording uv run main.py
response_text="John, for your symptoms of headache and fever, it's important to first
assess the severity. \n\n- **Rest:** Take plenty of rest and avoid any strenuous act
ivities.\n- **Hydration:** Drink plenty of fluids such as water, herbal teas, or clea
r soups to stay hydrated.\n- **Medication:** You can take over-the-counter pain relie
vers such as acetaminophen or ibuprofen to help reduce fever and alleviate headache.\
\n\nIf your headache is severe, you have a high fever, or if these symptoms persist wi
thout improvement, I recommend seeking medical attention to rule out any underlying c
onditions." escalate=False urgency=2
○ → recording
```

The agent response is now personalized to the patient name.

```
recording > main.py > get_latest_vitals
60     "Always mention the patient's name when available."
61 ],
62 )
63
64
65 @triage_agent.system_prompt
66 async def add_patient_name(ctx: RunContext[TriageDependencies]) -> str:
67     patient_name = await ctx.deps.db.patient_name(ctx.deps.patient_id)
68     return f"The patient name is: {patient_name}."
69
70 @triage_agent.tool
71 async def get_latest_vitals(ctx: RunContext[TriageDependencies]) -> dict[str, str]:
72     return await ctx.deps.db.latest_vitals(ctx.deps.patient_id)
73
74
75 async def main() -> None:
76     deps = TriageDependencies(patient_id=42, db=DatabaseConn())
```



we can also add tools that does something as above like the `get_latest_vitals()` function

```
• → recording uv run main.py
response_text='John, based on your current vital signs which are wi
-heart rate at 72 bpm and a blood pressure of 120/80-you might cons
home care for your headache and fever. Some initial steps could inc
quiet, dark room, drinking plenty of fluids, and taking over-the-co
like acetaminophen or ibuprofen to reduce fever and relieve pain. H
mptoms worsen, if the fever persists for more than a couple of days
ence any new symptoms like a stiff neck or rash, please seek medica
ately, as these could be signs of a more serious condition.' escalate
○ → recording
```



It now mentions the vitals like the blood pressure

```
recording > main.py > ...
15 class Patient:
17     name: str
18     vitals: dict[str, Any]
19
20
21 PATIENT_DB = {
22     42: Patient(
23         id=42, name="John Doe", vitals={"heart_rate": 72, "blood_pressure": 120/80},
24     ),
25     43: Patient(
26         id=43, name="Jane Smith", vitals={"heart_rate": 65, "blood_pressure": 110/70},
27     ),
28 }
29
```

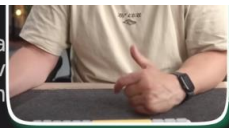


We can change John's vitals to give a high blood pressure and see what the agent does

```
22
23     name="John Doe", vitals={"heart_rate": 30, "blood_pressure": "80/30"}
24
25
26     name="Jane Smith", vitals={"heart_rate": 65, "blood_pressure": 110/70}
27
```



```
• → recording uv run main.py
response_text='John, based on your vitals, which indicate a critical
and blood pressure, along with your symptoms of a headache and fever,
I am seeking emergency medical attention immediately. These are signs of
serious underlying conditions.' escalate=True urgency=5
○ → recording
```



You can now combine LLM reasoning with the agent, validate using Pydantic and do more.

Pydantic AI

Search

pydantic/pydantic-ai v0.4.2 10.8k 1k

Pydantic AI

Troubleshooting

Upgrade Guide

Documentation

Agents

Models

Dependencies

Function Tools

Toolsets

Output

Messages and chat history

Unit testing

Pydantic Logfire Debugging and Monitoring

Multi-agent Applications

Graphs

Evals

Image, Audio, Video & Document Input

Thinking

Direct Model Requests

Common Tools

MCP

A2A

AG-UI

Multi-Agent Applications

There are roughly four levels of complexity when building applications with Pydantic AI:

1. Single agent workflows — what most of the `pydantic_ai` documentation covers

2. **Agent delegation** — agents using another agent via tools

3. **Programmatic agent hand-off** — one agent runs, then application code calls another agent

4. **Graph based control flow** — for the most complex cases, a graph-based state machine can be used to control the execution of multiple agents

Of course, you can combine multiple strategies in a single application.

Agent delegation

'Agent delegation' refers to the scenario where an agent delegates work to another agent, then takes back control when the delegate agent (the agent called from within a tool) finishes. If you want to hand off control to another agent completely, without coming back to the first agent, you can use an `output function`.

Since agents are stateless and designed to be global, you do not need to include the agent itself in agent `dependencies`.

You'll generally want to pass `ctx.usage` to the `usage` keyword argument of the delegate agent run so usage within that run counts towards the total usage of the parent agent run.

Table of contents

Agent delegation

Agent delegation and dependencies

Programmatic agent hand-off

Pydantic Graphs

Examples

Pydantic AI

Introduction

Installation

Getting Help

Contributing

Troubleshooting

Upgrade Guide

Documentation

Agents

Models

Dependencies

Function Tools

Toolsets

Output

Messages and chat history

Unit testing

Pydantic Logfire Debugging and Monitoring

Multi-agent Applications

Graphs

Evals

Image, Audio, Video & Document Input

Thinking

Direct Model Requests

Common Tools

Graphs

⚠ Don't use a nail gun unless you need a nail gun

If Pydantic AI **agents** are a hammer, and **multi-agent workflows** are a sledgehammer, then graphs are a nail gun:

- sure, nail guns look cooler than hammers
- but nail guns take a lot more setup than hammers
- and nail guns don't make you a better builder, they make you a builder with a nail gun
- Lastly, (and at the risk of torturing this metaphor), if you're a fan of medieval tools like mallets and untyped Python, you probably won't like nail guns or our approach to graphs. (But then again, if you're not a fan of type hints in Python, you've probably already bounced off Pydantic AI to use one of the toy agent frameworks — good luck, and feel free to borrow my sledgehammer when you realize you need it)

In short, graphs are a powerful tool, but they're not the right tool for every job. Please consider other **multi-agent approaches** before proceeding.

If you're not confident a graph-based approach is a good idea, it might be unnecessary.

Graphs and finite state machines (FSMs) are a powerful abstraction to model, execute, control and visualize complex workflows.

Alongside Pydantic AI, we've developed `pydantic-graph` — an async graph and state machine library for Python where nodes and edges are defined using type hints.

While this library is developed as part of Pydantic AI, it has no dependency on `pydantic-ai` and can be considered as a pure graph-based state machine library. You may find it useful whether or not you're using Pydantic AI or even building with GenAI.

Table of contents

Installation

Graph Types

GraphRunContext

End

Nodes

Graph

Stateful Graphs

GenAI Example

Iterating Over a Graph

Using Graph.iter for async for iteration

Using Graph.iter for sync iteration

Stateful Graphs

GenAI Example

Iterating Over a Graph

Using Graph.iter for async for iteration

Using Graph.iter for sync iteration

pydantic/pydantic-ai

v0.4.2

10.8k

1k

Pydantic AI

Upgrade Guide

Documentation

Agents

Models

Dependencies

Function Tools

Toolsets

Output

Messages and chat history

Unit testing

Pydantic Logfire Debugging and Monitoring

Multi-agent Applications

Graphs

Evals

Image, Audio, Video & Document Input

Thinking

Direct Model Requests

Common Tools

MCP

A2A

AG-UI

Command Line Interface (CLI)

Examples

Agent User Interaction (AG-UI)

Graphs

```
graph_example.py

from __future__ import annotations
from dataclasses import dataclass
from pydantic_graph import BaseNode, End, Graph, GraphRunContext

@dataclass
class DivisibleBy5(BaseNode[None, None, int]):
    foo: int

    async def run(
        self,
        ctx: GraphRunContext,
    ) -> Increment | End[int]:
        if self.foo % 5 == 0:
            return End(self.foo)
        else:
            return Increment(self.foo)

@dataclass
class Increment(BaseNode):
    foo: int

    async def run(self, ctx: GraphRunContext) -> DivisibleBy5:
        return DivisibleBy5(self.foo + 1)

fives_graph = Graph(nodes=[DivisibleBy5, Increment])
result = fives_graph.run_sync(DivisibleBy5(4))
print(result.output)
#> 5
```

(This example is complete, it can be run "as is" with Python 3.10+)

Table of contents

Installation

Graph Types

GraphRunContext

End

Nodes

Graph

Stateful Graphs

GenAI Example

Iterating Over a Graph

Using Graph.iter for async for iteration

Using Graph.iter for sync iteration

Stateful Graphs

GenAI Example

Iterating Over a Graph

Using Graph.iter for async for iteration

Using Graph.iter for sync iteration

pydantic/pydantic-ai

v0.4.2

10.8k

1k

Files

main

Go to file

- efficient-python-dockerfile
- functions
- gitbranch
- go
- kale
- libraries
- map
- mcp-server
- pydanticai
 - main.py
 - pyproject.toml
 - uv.lock
- pytips
- sdk
- serverless
- simple-pytest
- simple
- standard
- testtips
- typehints
- typescript
- .gitignore
- .pylintrc

examples / 2025 / pydanticai / main.py

egges Added code example

e415018 · last month History

Code Blame 101 lines (76 loc) · 2.71 KB

```
1 from dataclasses import dataclass
2 from typing import Any
3 import asyncio
4
5 from pydantic import BaseModel, Field
6 from pydantic_ai import Agent, RunContext
7
8 from dotenv import load_dotenv
9
10
11 # Load environment variables from .env
12 load_dotenv()
13
14
15 # Mock database
16 @dataclass
17 class Patient:
18     id: int
19     name: str
20     vitals: dict[str, Any]
21
22 PATIENT_DB = {
23     42: Patient(id=42, name="John Doe", vitals={"heart_rate": 72, "blood_pressure": "120/80"}),
24     43: Patient(id=43, name="Jane Smith", vitals={"heart_rate": 65, "blood_pressure": "110/70"}),
25 }
26
27 class DatabaseConn:
28     async def patient_name(self, id: int) -> str:
29         patient = PATIENT_DB.get(id)
30         return patient.name if patient else "Unknown Patient"
31
32     async def latest_vitals(self, id: int) -> dict[str, Any]:
33         patient = PATIENT_DB.get(id)
34         return patient.vitals if patient else {"heart_rate": 0, "blood_pressure": "N/A"}
35
```

Files

main

Go to file

- efficient-python-dockerfile
- functions
- gitbranch
- go
- kale
- libraries
- map
- mcp-server
- pydanticai
 - main.py
 - pyproject.toml
 - uv.lock
- pytips
- sdk
- serverless
- simple-pytest
- simple
- standard
- testtips
- typehints
- typescript
- .gitignore
- .pylintrc
- LICENSE

examples / 2025 / pydanticai / main.py

Code Blame 101 lines (76 loc) · 2.71 KB

```
36
37 @dataclass
38 class TriageDependencies:
39     patient_id: int
40     db: DatabaseConn
41
42
43 class TriageOutput(BaseModel):
44     response_text: str = Field(description="Message to the patient")
45     escalate: bool = Field(description="Should escalate to a human nurse")
46     urgency: int = Field(description="Urgency level from 0 to 10", ge=0, le=10)
47
48
49 triage_agent = Agent(
50     "openai:gpt-4o",
51     deps_type=TriageDependencies,
52     output_type=TriageOutput,
53     system_prompt=(
54         "You are a triage assistant helping patients. "
55         "Provide clear advice and assess urgency."
56     ),
57 )
58
59
60 @triage_agent.system_prompt
61 async def add_patient_name(ctx: RunContext[TriageDependencies]) -> str:
62     patient_name = await ctx.deps.db.patient_name(id=ctx.deps.patient_id)
63     return f"The patient's name is {patient_name}."
64
65
66 @triage_agent.tool
67 async def latest_vitals(ctx: RunContext[TriageDependencies]) -> dict[str, Any]:
68     """Returns the patient's latest vital signs."""
69     return await ctx.deps.db.latest_vitals(id=ctx.deps.patient_id)
70
71
72 async def main() -> None:
73     deps = TriageDependencies(patient_id=43, db=DatabaseConn())
74
75     result = await triage_agent.run(
76         "I have chest pain and trouble breathing.",

```

Files

main

Go to file

efficient-python-dockerfile

functions

gitbranch

go

kale

libraries

map

mcp-server

pydanticai

main.py

pyproject.toml

uv.lock

pytips

sdk

serverless

simple-pytest

simple

standard

testtips

typehints

typescript

.gitignore

.pylintrc

examples / 2025 / pydanticai / main.py

CodeBlame101 lines (76 loc) · 2.71 KB

Raw

```
63         return f"The patient's name is {patient_name!r}."
64
65
66     @trriage_agent.tool
67     async def latest_vitals(ctx: RunContext[TriageDependencies]) -> dict[str, Any]:
68         """Returns the patient's latest vital signs."""
69         return await ctx.deps.db.latest_vitals(id=ctx.deps.patient_id)
70
71
72     async def main() -> None:
73         deps = TriageDependencies(patient_id=43, db=DatabaseConn())
74
75         result = await triage_agent.run(
76             "I have chest pain and trouble breathing.",
77             deps=deps,
78         )
79         print(result.output)
80         """
81         Example output:
82         response_text='Your symptoms are serious. Please call emergency services immediately. A nurse will contact you shortly.'
83         escalate=True
84         urgency=10
85         """
86
87         result = await triage_agent.run(
88             "I have a mild headache since yesterday.",
89             deps=deps,
90         )
91         print(result.output)
92         """
93         Example output:
94         response_text='It sounds like your headache is not severe, but monitor it closely. If it worsens or you develop new symptoms, contact your doctor.'
95         escalate=False
96         urgency=3
97         """
98
99
100     if __name__ == "__main__":
101         asyncio.run(main())
```

Files

main

Go to file

functions

gitbranch

go

kale

libraries

map

mcp-server

pydanticai

main.py

pyproject.toml

uv.lock

examples / 2025 / pydanticai / pyproject.toml

CodeBlame10 lines (10 loc) · 209 Bytes

Raw

```
1 [project]
2 name = "pydanticai"
3 version = "0.1.0"
4 description = "Pydantic AI tutorial"
5 requires-python = ">=3.13"
6 dependencies = [
7     "pydantic>=2.11.7",
8     "pydantic-ai>=0.4.2",
9     "python-dotenv>=1.1.1",
10 ]
```

egges Added code example

e415018 · last month History