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.

https://github.com/ArjanCodes/examples/tree/main/2025/pydanticai

ArjanCodes / examples

Code | Issues 3 | Pull requests 2 | Actions | Projects | Security | Insights

Files

main

Go to file

.github
.vscode
2022
2023
2024
2025

examples / 2025 / pydanticai /

Add file

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

```python
import asyncio
from dataclasses import dataclass
from typing import Any

from dotenv import load_dotenv

# Load environment variables from .env
load_dotenv()


# Mock database
@dataclass
class Patient:
```

recording > main.py > Patient

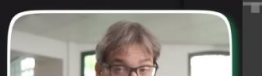
```python
@dataclass
class Patient:
    id: int
    name: str
    vitals: dict[str, Any]


PATIENT_DB = {
    42: Patient(
        id=42, name="John Doe", vitals={"heart_rate": 72, "b
    ),
    43: Patient(
        id=43, name="Jane Smith", vitals={"heart_rate": 65,
    ),
}
```

```
20
21  e="John Doe", vitals={"heart_rate": 72, "blood_pressure": "1
22
23
24  e="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 × 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:
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 > ...

```python
import asyncio
from dataclasses import dataclass
from typing import Any
from pydantic import BaseModel

```

recording > main.py > TriageOutput

```python
        return patient.vitals if patient else {"heart_rate": 0, "blood_pre

@dataclass
class TriageDependencies:
    patient_id: int
    db: DatabaseConn

class TriageOutput(BaseModel):
    response_text: str
    escalate: bool
    urgency: int

async def main() -> None:
    db = DatabaseConn()
    patient_name = await db.patient_name(42)
    latest_vitals = await db.latest_vitals(42)

```

zsh main.py 2

recording > main.py > ...

```python
import asyncio
from dataclasses import dataclass
from typing import Any
from pydantic import BaseModel, Field
import os

```

recording > main.py > TriageDependencies

```python
        return patient.vitals if patient else {"heart_rate":


@dataclass
class TriageDependencies:
    patient_id: int
    db: DatabaseConn


class TriageOutput(BaseModel):
    response_text: str = Field(description="Message to the patient")
    escalate: bool = Field(description="Whether to escalate the case to a
    urgency: int = Field(description="Urgency level of the case")


async def main() -> None:
    db = DatabaseConn()
```

```
utput(BaseModel):
text: str = Field(description="Message to the patient")
 bool = Field(description="Whether to escalate the case to a human nurse")
int = Field(description="Urgency level of the case")

n() -> None:
baseConn()
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 > ...
import asyncio
from dataclasses import dataclass
from typing import Any

from dotenv import load_dotenv
from pydantic import BaseModel, Field
from pydantic_ai import Agent

```

```
recording > main.py > ...
class TriageOutput(BaseModel):
    escalate: bool = Field(description="Whether to escalate the case to a
    urgency: int = Field(description="Urgency level of the case")

triage_agent = Agent(
    "openai:gpt-4o",
    deps_type=TriageDependencies,
    output_type=TriageOutput,
    system_prompt=(
        "You are a triage assistant helping patients."
        "Provide clear advice and assess urgency."
    )
)
async def main() -> None:
```

```
recording > main.py > ...
    deps_type=TriageDependencies,
    output_type=TriageOutput,
    system_prompt=(
        "You are a triage assistant helping patients."
        "Provide clear advice and assess urgency."
    )
)
async def main() -> None:
    db = DatabaseConn()
    patient_name = await db.patient_name(42)
    latest_vitals = await db.latest_vitals(42)

    print(f"Patient Name: {patient_name}")
    print(f"Latest Vitals: {latest_vitals}")


if __name__ == "__main__":
```

We can now update this main function as below

```python
recording > main.py > main
)


async def main() -> None:
    deps = TriageDependencies(patient_id=42, db=DatabaseConn())
    result = await triage_agent.run(
        "What should I do if I have a headache and a fever?", deps=deps
    )

    print(result.output)


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

```
→ 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

```python
recording > main.py > ...
import asyncio
from dataclasses import dataclass
from typing import Any

from dotenv import load_dotenv
from pydantic import BaseModel, Field
from pydantic_ai import Agent, RunContext
```

```python
recording > main.py > add_patient_name
    output_type=TriageOutput,
    system_prompt=(
        "You are a triage assistant helping patients."
        "Provide clear advice and assess urgency."
    ),
)

@triage_agent.system_prompt
async def add_patient_name(ctx: RunContext[TriageDependencies]) -> str:
    patient_name = await ctx.deps.db.patient_name(ctx.deps.patient_id)
    return f"The patient name is: {patient_name}."


async def main() -> None:
```

```
recording > main.py > ...
47  class TriageOutput(BaseModel):
50      urgency: int = Field(description="Urgency level of the ca
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
```

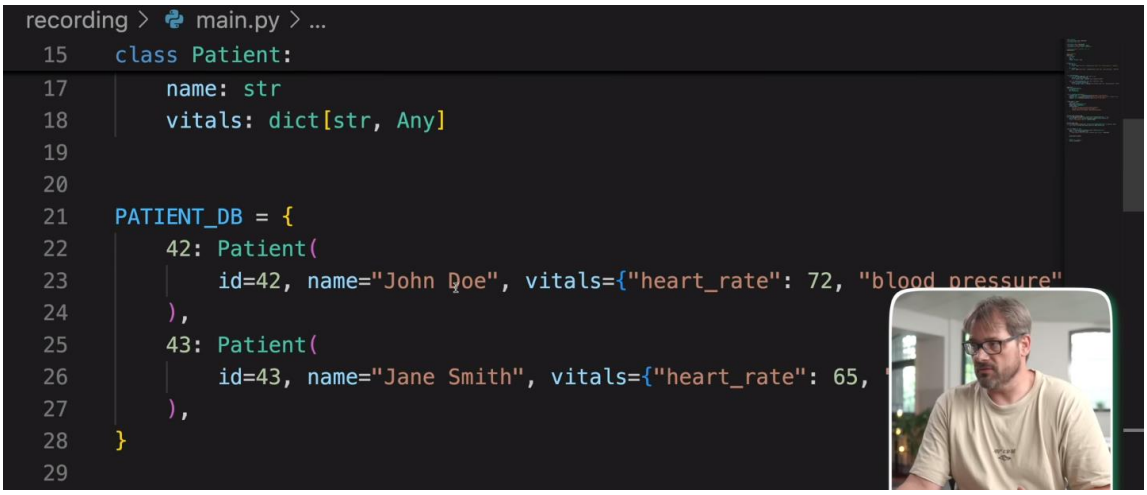
The agent response is now personalized to the patient name.

```
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[s
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())
```

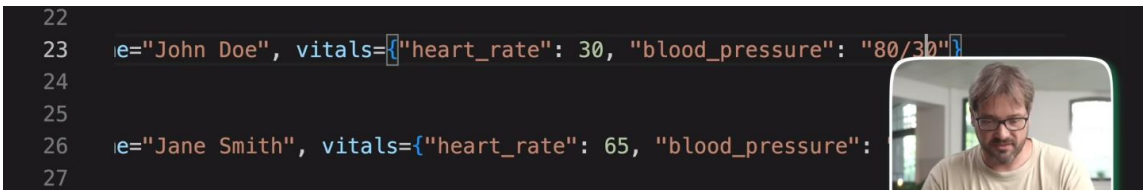
we can also add tools that does something as above like the **get_latest_vitals()** function

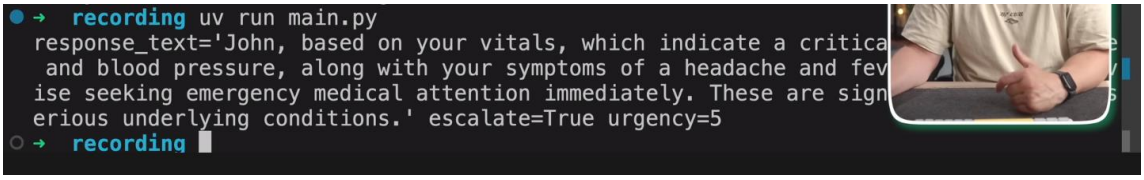
```
→  recording uv run main.py
response_text='John, based on your current vital signs which are wi
-heart rate at 72 bpm and a blood pressure of 120/80-you might cons
home care for your headache and fever. Some initial steps could inc
quiet, dark room, drinking plenty of fluids, and taking over-the-co
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.' escalat
→  recording
```

It now mentions the vitals like the blood pressure

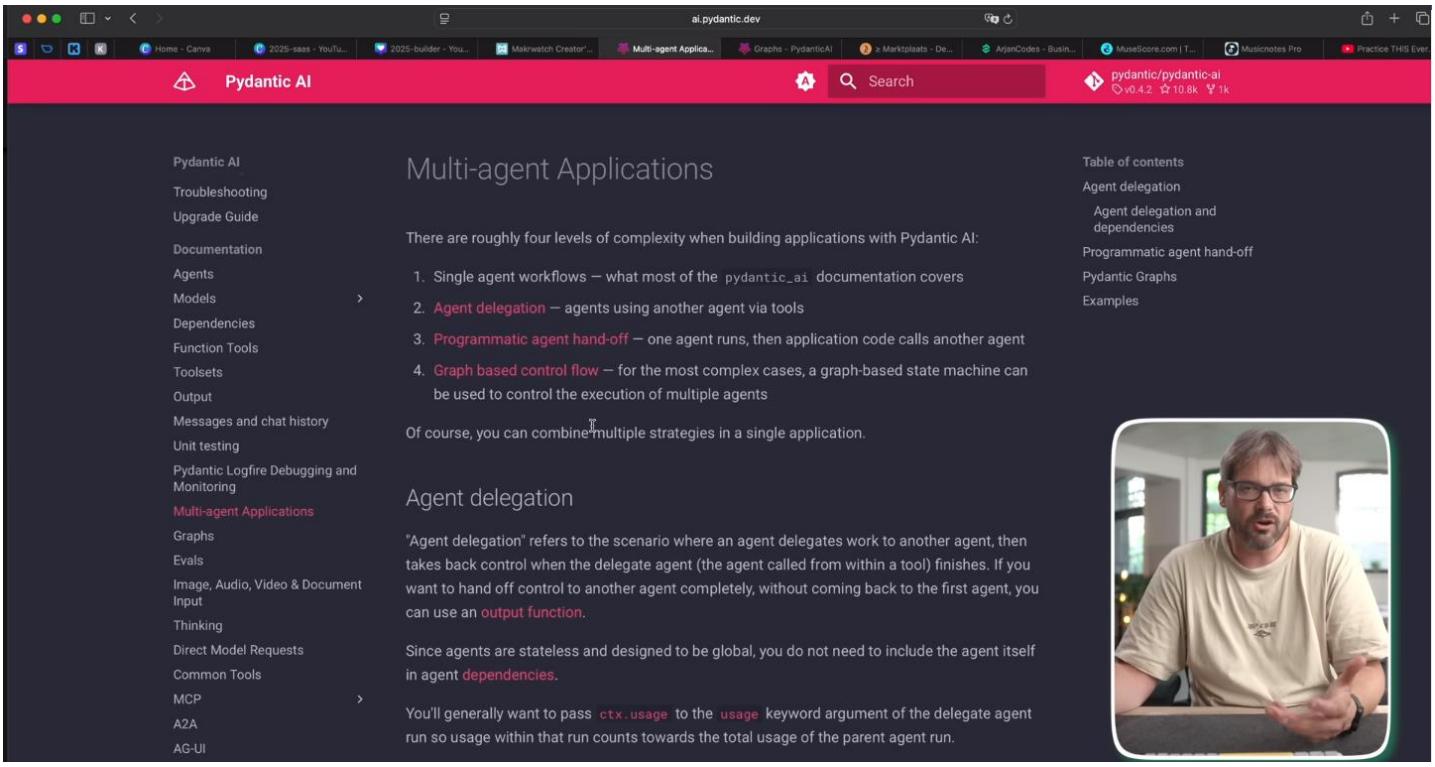


We can change John's vitals to give a high blood pressure and see what the agent does





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



ai.pydantic.dev

Pydantic AI

Search

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

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.

ai.pydantic.dev

Graphs

Search

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

graph_example.py

```python
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+)

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 Raw

```python
from dataclasses import dataclass
from typing import Any
import asyncio

from pydantic import BaseModel, Field
from pydantic_ai import Agent, RunContext

from dotenv import load_dotenv


# Load environment variables from .env
load_dotenv()


# Mock database
@dataclass
class Patient:
    id: int
    name: str
    vitals: dict[str, Any]

PATIENT_DB = {
    42: Patient(id=42, name="John Doe", vitals={"heart_rate": 72, "blood_pressure": "120/80"}),
    43: Patient(id=43, name="Jane Smith", vitals={"heart_rate": 65, "blood_pressure": "110/70"}),
}

class DatabaseConn:
    async def patient_name(self, id: int) -> str:
        patient = PATIENT_DB.get(id)
        return patient.name if patient else "Unknown Patient"

    async def latest_vitals(self, id: int) -> dict[str, Any]:
        patient = PATIENT_DB.get(id)
        return patient.vitals if patient else {"heart_rate": 0, "blood_pressure": "N/A"}

```

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

Top

Code Blame 101 lines (76 loc) · 2.71 KB Raw

```python

@dataclass
class TriageDependencies:
    patient_id: int
    db: DatabaseConn


class TriageOutput(BaseModel):
    response_text: str = Field(description="Message to the patient")
    escalate: bool = Field(description="Should escalate to a human nurse")
    urgency: int = Field(description="Urgency level from 0 to 10", ge=0, le=10)


triage_agent = Agent(
    "openai:gpt-4o",
    deps_type=TriageDependencies,
    output_type=TriageOutput,
    system_prompt=(
        "You are a triage assistant helping patients. "
        "Provide clear advice and assess urgency."
    ),
)


@triage_agent.system_prompt
async def add_patient_name(ctx: RunContext[TriageDependencies]) -> str:
    patient_name = await ctx.deps.db.patient_name(id=ctx.deps.patient_id)
    return f"The patient's name is {patient_name!r}."


@triage_agent.tool
async def latest_vitals(ctx: RunContext[TriageDependencies]) -> dict[str, Any]:
    """Returns the patient's latest vital signs."""
    return await ctx.deps.db.latest_vitals(id=ctx.deps.patient_id)


async def main() -> None:
    deps = TriageDependencies(patient_id=43, db=DatabaseConn())

    result = await triage_agent.run(
        "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

Top

Code Blame 101 lines (76 loc) · 2.71 KB

Raw

```python
    return f"The patient's name is {patient_name!r}."


@triage_agent.tool
async def latest_vitals(ctx: RunContext[TriageDependencies]) -> dict[str, Any]:
    """Returns the patient's latest vital signs."""
    return await ctx.deps.db.latest_vitals(id=ctx.deps.patient_id)


async def main() -> None:
    deps = TriageDependencies(patient_id=43, db=DatabaseConn())

    result = await triage_agent.run(
        "I have chest pain and trouble breathing.",
        deps=deps,
    )
    print(result.output)
    """
    Example output:
    response_text='Your symptoms are serious. Please call emergency services immediately. A nurse will contact you shortly.'
    escalate=True
    urgency=10
    """

    result = await triage_agent.run(
        "I have a mild headache since yesterday.",
        deps=deps,
    )
    print(result.output)
    """
    Example output:
    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.'
    escalate=False
    urgency=3
    """


if __name__ == "__main__":
    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

egges Added code example

e415018 · last month History

Code Blame 10 lines (10 loc) · 209 Bytes

Raw

```toml
[project]
name = "pydanticai"
version = "0.1.0"
description = "Pydantic AI tutorial"
requires-python = ">=3.13"
dependencies = [
    "pydantic>=2.11.7",
    "pydantic-ai>=0.4.2",
    "python-dotenv>=1.1.1",
]
```