**Pydantic Dataclasses vs BaseModel, Type Hints for Validation & Using Dataclasses as Agent Output**

**1. @pydantic.dataclasses.dataclass vs pydantic.BaseModel**

**Overview:**Pydantic offers BaseModel and a decorator for dataclasses to provide validation.BaseModel is feature-rich; dataclass decorator wraps standard dataclasses for validation.

Key differences and trade-offs:  
- Syntax & ergonomics  
- Validators and lifecycle hooks  
- JSON schema & serialization  
- Use cases

**Example — BaseModel:**

from pydantic import BaseModel, Field  
  
class User(BaseModel):  
 id: int  
 name: str = Field(..., min\_length=1)  
  
user = User(id='123', name='Alice')  
print(user.dict())

**Example — pydantic dataclass:**

from pydantic.dataclasses import dataclass  
  
@dataclass  
class UserDC:  
 id: int  
 name: str  
  
user = UserDC(id='123', name='Bob')  
print(user)

**2. Type Hints for Validation and Schema Definition**

**Overview:**  
Pydantic uses Python type hints to perform runtime validation and generate JSON schemas. Core patterns include basic types, Optional, nested models, Annotated/Field, and validators.

**Annotated and Field example:**

from typing import Annotated  
from pydantic import BaseModel, Field  
  
class Product(BaseModel):  
 name: str  
 price: Annotated[float, Field(gt=0, description='Price must be > 0')]  
p = Product(name='Widget', price=9.99)  
**Validator example:**

from pydantic import BaseModel, field\_validator  
  
class Model(BaseModel):  
 tags: list[str]  
  
 @field\_validator('tags', mode='before')  
 def ensure\_list(cls, v):  
 if not isinstance(v, list):  
 return [v]  
 return v

**3. Using Dataclasses as output\_type in OpenAI Agents SDK**

**Overview:**The OpenAI Agents SDK supports structured outputs via output\_type. Supported types include Pydantic models, dataclasses, TypedDict, and other types that Pydantic TypeAdapter can wrap.

**Example — dataclass as output\_type:**

from dataclasses import dataclass  
from agents import Agent  
  
@dataclass  
class Summary:  
 title: str  
 bullets: list[str]  
agent = Agent(  
 name='Summarizer',  
 instructions='Read the input and return a Summary dataclass',  
 output\_type=Summary,  
)

Notes:  
- The SDK uses Pydantic TypeAdapter to generate JSON schema. Some complex constructs may differ from BaseModel behavior.  
- Use AgentOutputSchema wrappers or strict\_json\_schema flags for custom behavior.

**References**

OpenAI Agents SDK docs: <https://openai.github.io/openai-agents-python/agents/>

OpenAI Agents function\_schema: <https://openai.github.io/openai-agents-python/ref/function_schema/>

OpenAI Agents GitHub: <https://github.com/openai/openai-agents-python>

Pydantic docs — Dataclasses & BaseModel: <https://docs.pydantic.dev/latest/usage/dataclasses/>

Pydantic validators: <https://docs.pydantic.dev/latest/concepts/validators/>