Note on Pydantic models

compiled by D.Gueorguiev, 6/24/25

One of the primary ways of defining schema in Pydantic is via models. Models are simply classes which inherit from BaseModel and define fields as annotated attributes.

One can think of models as similar to structs in languages like C, or as the requirements of a single endpoint in an API.

Models share many similarities with Python’s dataclasses, but have been designed with some subtle-yet-important differences that streamline certain workflows related to validation, serialization, and JSON schema generation.

Untrusted data can be passed to a model and , after parsing and validation, Pydantic guarantees that the fields of the resultant model instance will conform to the field types defined on the model.

**Note**: The term *validation* refers to the process of instantiating a model (or other type) that adheres to specified types and constraints. However, Pydantic’s primary focus does not align precisely with the dictionary definition of *validation*. Pydantic guarantees the types and constraints of the output, not the input data. This distinction becomes apparent when considering that Pydantic’s ValidationError is raised when data cannot be successfully parsed into a model instance.

While this distinction may initially seem subtle, it holds practical significance. In some cases *validation* goes beyond just model creation, and can include the copying and coercion of data. This can involve copying arguments passed to the constructor in order to perform coercion to the new type without mutating the original input data.

The primary goal of Pydantic is to assure that the resulting structure post-processing (referred to as *validation*) precisely conforms to the applied type hints.

Basic model usage

from pydantic import BaseModel, ConfigDict

class User(BaseModel):

id: int

name: str = ‘Jane Doe’

model\_config = ConfigDict(str\_max\_length=10)

Fields can be customized using the Field() function. Field() behaves the same way as the standard library field() function for dataclasses.

# Appendix

## field() function in python dataclasses

The field() function in Python's dataclasses module is used to provide more granular control over individual fields within a dataclass. While basic dataclass fields can be defined simply with type annotations and optional default values, field() allows for customization of how these fields behave in generated methods like \_\_init\_\_, \_\_repr\_\_, \_\_eq\_\_, and \_\_hash\_\_.

Key parameters and their purpose:

default:

Specifies a default value for the field.

default\_factory:

Provides a zero-argument function that will be called to generate a default value for the field. This is crucial for handling mutable default values (like lists or dictionaries) to prevent shared state between instances.

init:

A boolean indicating whether the field should be included as a parameter in the generated \_\_init\_\_() method. (Default is True).

repr:

A boolean indicating whether the field should be included in the string returned by the generated \_\_repr\_\_() method. (Default is True).

compare:

A boolean indicating whether the field should be included in the generated equality and comparison methods (\_\_eq\_\_, \_\_gt\_\_, etc.). (Default is True).

hash:

A boolean or None indicating whether the field should be included in the generated \_\_hash\_\_() method. If None (the default), it uses the value of compare.

metadata:

A mapping (usually a dictionary) to store arbitrary metadata about the field. This metadata is not directly used by dataclasses but can be accessed by other tools or libraries.

kw\_only:

(Introduced in Python 3.10) A boolean indicating that the field can only be specified using a keyword argument in the \_\_init\_\_ method, not as a positional argument.

from dataclasses import dataclass, field

@dataclass

class Book:

title: str

author: str

pages: int = field(default=0) # Field with a default value

tags: list[str] = field(default\_factory=list) # Field with a mutable default, using default\_factory

isbn: str = field(init=False, repr=False) # Field not included in init or repr

notes: str = field(compare=False, default="") # Field not included in comparisons