PYDANTIC

Type hinting:- In python type hinting does not raise any errors, it just suggest

Pydantic solves two major problem in Python :-

1) Is Type hinting type safe
2) Data validation

Pydantic :

1. Define a pydantic model that represents the ideal schema of the data,
   - This includes the expect fields, their types and any validation constraints (e.g., gt=0 for positive numbers).
2) Instantiate the model with raw input data (usually a dictionary or JSON-like structure)
   - Pydantic will automatically validate the data and coerce it into the correct python types. (If possible).
   - If the data doesn't meet the model's requirements, pydantic raise a validation Error
3) Pass the validated model object to functions or use it throughout your codebase.
   - This ensures that every part of your program works with clean, type safe, and logically valid data.

Two level Data validation

we use here

from typing import List
from typing " Dict

"   "   "   List, Dict

class patient (BaseModel):

allergies : List [str]

contact_details : Dict[str, str]

here the point is we cannot use normal list, dict here is pydantic base model, because we want here two level Data validation. like List [str] this syntax mean in allergies the Data which store in List format, and also inner list all datas are in string format, not int, not float, by this method we can achive two level Data validation (1) whole data is list
(2) inner list data is string

if we use there normal list like allergies : list there we can only add one level Data validation like whole Data is list but inner data in list we dont know/not validate. same goes for contact_details also

Two level Data validation

we use here

```
from typing import List
from typing import Dict
```

** we " " " List, Dict

```
class patient (BaseModel):

allergies : List [str]
contact_details : Dict[str, str]
```

here the point is we cannot use normal [list], [dict] here is pydantic base model, because we want here two level Data validation. like [List [str]] this syntax mean in allergies the Data which store in List format, and also inner list all datas are in string format, not int, not float, by this method we can achive two level Data validation (1) whole data is list
(2) inner list data is string

if we use there normal list like [allergies : list] there we can only add one level Data validation like whole Data is list but inner data in list we dont know/ not validate. so same goes for contact_details also

whole Data are in Dictionary, are ~~we~~ inner to dictionary ~~all data~~ key and value are in string format.

★★★ you have to add all fields in your user data, which you add in your pydantic Basemodel, ~~ob~~ if not its throw a error. ~~but~~ | until you ~~add~~ we ~~use~~ optional[] until you set default values, |

★★★ From typing import Optional

=> Some times we can face a issue like a user does not fill the data and its ~~have to be~~ not necessary for everyone like

married : Optional[bool] = None
variable

here we can ~~see~~ see we use Optional here its not necessary now, you have to use third bracket after this and also you have to add a default value = None, if user dont put data its automatically sends None value,

★★★ you can also set default value in require fields. if user dont put value it sends default data.

Data validation in python (Pydantic)

```
from pydantic import BaseModel, EmailStr
```

↳ it is a built in data validation type for validate Email.

```
email : EmailStr
variable
```

by using this syntax you can use it and you use this without coding ~~this with out~~ Email validation code.

```
from pydantic import BaseModel, AnyUrl
```

↓

it is a built in data validation type ~~for~~ validate Url.

```
Url : Url
variable
```

Now you dont have to worry about Url data validation.

Uncommon Data validation (custom Data validation)

```
from pydantic import BaseModel, Field
```

↓

we can use our custom constraints. like `gt, ge, lt, le`

```
weight : float = Field (gt=0)
variable   Data type   constraints (custom)
```

Whole Data are in Dictionary, and we validate it inner dictionary key and value are in both strings in this scenario

Attach Meta data using Field and Annotated

```
from pydantic import BaseModel, Field
from typing import Annotated
```

most important

- use in custom data validation
- " " " set to meta data
- " " " " " default value
- " " " overwrite normal type conversion

Actually we use this many times

example :- 1

```
weight : Annotated[float, Field(gt=0, lt=120, strict=True)]
```

weight → variable
float → Data type
gt=0, lt=120 → custom constraints
strict=True → now it's does not change data type automatic.

example :- 2

```
name : Annotated[str, Field(max_length=50, title="...", description="...", examples=['nitish', 'amit'])]
```

here we can see we can also set description, title, examples. these all are meta data.

Field Validator in Pydantic

it is use for custom data validation in one specific data

- class method
- value
- mode = 'before' , mode = 'after' (default)
- type coherson

```
from pydantic import BaseModel, field_validator

class patient (BaseModel):
    email: EmailStr
indention

    @field_validator ('email')
    @ classmethod
    def email_validator (cls, value): # value for checking field value in email
        valid_domains = ['hdfc.com', 'icici.com']
        domain_name = value.split ('@')[-1]
        if domain_name not in valid_domains:
            raise ValueError ('Not a valid domain')
        return value
```

Model Validator

it is use for custom data validation in more than one field. so you have to use model validator not field.

```
from pydantic import BaseModel, model_validator
class patient (Basemodel):
    age: int
    weight: float
    @ model_validator (mode = "after") # here is a case we dont use field here because this is for model not single field
```

Computed fields

Here we calculate the new field by existing field's data, here for this field we dont receive new user's data, we receive existing field's data which we user put it other fields.

here we use two new decorators

```
@computed_field
@property
```

```
from pydantic import BaseModel, computed_field

class patient (BaseModel):
    weight : float #kg
    height : float #meter

    @computed_field
    @property
    def bmi (self) -> float :
        bmi = round (self.weight/(self.height**2),2)
        return bmi
```

here noticable thing is we set function name bmi here, actually now when you print the data its now (patient.bmi), its taking function name as a new computed_field's variable here.

Nested Models

here we use nested model like

```
class Address (BaseModel):
    city : str
    State : str
    Pin : str
class Patient (BaseModel):
    name : str
    gender : str
    age : int
    address: Address
```

Serialization

Convert ~~python~~ Pydantic model into ~~pydantic~~ Python dictionary or JSON

- .model_dump
- .model_dump_json
  - (include)
  - (exclude)
  - exclude_unset=True