Pydantic

💡 pip install pydantic

What is Pydantic?

Pydantic is a **Python library** that helps you:

- **Validate data** (check if data is correct and in the right format).
- **Parse data** (convert data from one type to another, e.g., strings to integers).
- **Create data models** (like a blueprint for your data, similar to a form).

> **It is mainly used in FastAPI, but it works standalone too.**

Think of Pydantic as a **data checker**—it makes sure your data is clean and matches the structure you expect.

Why use Pydantic?

- **Automatic type conversion**

  If you say an age must be an integer but you get `"25"` (string), Pydantic converts it to `25` (integer).

- **Error messages**

  If the data is invalid, it shows clear error messages.

- **Works like a guard**

  You define rules, and Pydantic enforces them.

Type Hinting:

```
def greet(name: str) → str:
    return f"Hello, {name}!"
```

- `name: str` means the function expects a string.
- `> str` means it returns a string.

> This does not raise an error if you pass `int`

Steps

Step 1: Define a Pydantic Model (class)

```
from pydantic import BaseModel

class Patient(BaseModel):
    name: str
    age: int
```

Step 2: Make an Object

```
patient_info = {'name':'John','age': '30'}

patient1= Patient(**patient_info)
```

- We made an object of the `Patient` class & passed dictionary
- `**` → Unpacks the dictionary

> 💡 **Pydantic converted `'30'` → 30 (int)**

```
def insert_data(patient: Patient):
    print(patient.name)
    print(patient.age)
```

```
insert_data(patient1)
```

```
John
30
```

> 💡 **Pydantic will auto-validate the data**
> **If the data does not meet the req, it will raise a** `ValidationError`

Advanced Type validation

- For list of strings → `List [str]`

```
from typing import List
from pydantic import BaseModel

class Temp(BaseModel):
    name: str
    allengies: List [str]
```

> 💡 We write `List [str]` **and not just** `list` **because of 2 level validation.**

Dictionary:

`Dict[str, str]`

```
from typing import List, Dict
from pydantic import BaseModel

class Temp(BaseModel):
    name: str
    allengies: List [str]
    contact: Dict[str, str]
```

Optional Fields

`Optional[List [str]]= None`

```
from pydantic import BaseModel
from typing import List, Dict, Optional

class Patient(BaseModel):
    name: str
    allengies: Optional[List [str]] = None
    contact: Dict[str, str]

info= {"name": "MJF", "contact": {"phone": "6546351"}}

patient1= Patient(**info)
patient1
```

```
Patient(name='MJF', allengies=None, contact={'phone': '6546351'})
```

> Make sure to write `None` when you use `Optional`

- You can provide any default value in place of `None`

💡 Just like `Optional`, you can set default value for any field.

Data Validation

- eg. email

`email: EmailStr`

```
from pydantic import BaseModel, EmailStr
from typing import List, Dict, Optional

class Patient(BaseModel):
    name: str
    allengies: Optional[List [str]] = None
    email: EmailStr

info= {"name": "randy", "email": "rko@wwe.com"}

patient1= Patient(**info)
patient1
```

```
Patient(name='randy', allengies=None, email='rko@wwe.com')
```

```
from pydantic import BaseModel, EmailStr
from typing import List, Dict, Optional

class Patient(BaseModel):
    name: str
    allengies: Optional[List [str]] = None
    email: EmailStr

info= {"name": "randy", "email": "rkowwe.com"} #Improper Email Format

patient1= Patient(**info)
patient1
```

```
---------------------------------------------------------------------------
ValidationError                           Traceback (most recent call last)
Cell In[9], line 11
      7     email: EmailStr
      9 info= {"name": "randy", "email": "rkowwe.com"}
---> 11 patient1= Patient(**info)
     12 patient1

File d:\Python_Env\LangChain\venv\lib\site-packages\pydantic\main.py:253, in BaseModel.__init__(self, **data)
    251 # `__tracebackhide__` tells pytest and some other tools to omit this function from tracebacks
    252 __tracebackhide__ = True
--> 253 validated_self = self.__pydantic_validator__.validate_python(data, self_instance=self)
    254 if self is not validated_self:
    255     warnings.warn(
    256         'A custom validator is returning a value other than `self`.\n'
    257         "Returning anything other than `self` from a top level model validator isn't supported when validating via `__init__`.\n"
    258         'See the `model_validator` docs (https://docs.pydantic.dev/latest/concepts/validators/#model-validators) for more details.',
    259         stacklevel=2,
    260     )

ValidationError: 1 validation error for Patient
email
  value is not a valid email address: An email address must have an @-sign. [type=value_error, input_value='rkowwe.com', input_type=str]
```

URL:

```
from pydantic import BaseModel, EmailStr, AnyUrl
from typing import List, Dict, Optional

class Patient(BaseModel):
    name: str
    allengies: Optional[List [str]] = None
    url: AnyUrl

info= {"name": "randy", "url": "https://www.wwe.com"}

patient1= Patient(**info)
patient1
```

```
Patient(name='randy', allengies=None, url=AnyUrl('https://www.wwe.com/'))
```

```
from pydantic import BaseModel, EmailStr, AnyUrl
from typing import List, Dict, Optional
```

```
class Patient(BaseModel):
    name: str
    allengies: Optional[List [str]] = None
    url: AnyUrl

info= {"name": "randy", "url": "wwe.com"}

patient1= Patient(**info)
patient1
```

```
---------------------------------------------------------------------------
ValidationError                           Traceback (most recent call last)
Cell In[13], line 11
      7     url: AnyUrl
      9 info= {"name": "randy", "url": "wwe.com"}
---> 11 patient1= Patient(**info)
     12 patient1

File d:\Python_Env\LangChain\venv\lib\site-packages\pydantic\main.py:253, in BaseModel.__init__(self, **data)
    251 # `__tracebackhide__` tells pytest and some other tools to omit this function from tracebacks
    252 __tracebackhide__ = True
--> 253 validated_self = self.__pydantic_validator__.validate_python(data, self_instance=self)
    254 if self is not validated_self:
    255     warnings.warn(
    256         'A custom validator is returning a value other than `self`.\n'
    257         "Returning anything other than `self` from a top level model validator isn't supported when validating via `__i
    258         'See the `model_validator` docs (https://docs.pydantic.dev/latest/concepts/validators/#model-validators) for mo
    259         stacklevel=2,
    260     )

ValidationError: 1 validation error for Patient
url
  Input should be a valid URL, relative URL without a base [type=url_parsing, input_value='wwe.com', input_type=str]
    For further information visit https://errors.pydantic.dev/2.11/v/url_parsing
```

💡 **It will raise an error even if we remove** `https://`

Custom Data Validation using `Field:`

Greater than, Less than:

```
from pydantic import BaseModel, EmailStr, AnyUrl, Field
from typing import List, Dict, Optional
```

```
class Patient(BaseModel):
    name: str
    age: int = Field(gt=0, lt=100)

info= {"name": "randy", "age": 52}

patient1= Patient(**info)
patient1
```

```
Patient(name='randy', age=52)
```

- The age must be between 1 and 99 (0<age<100)

Max Length:

```
name: str = Field(max_length=50)
```

- Name will be of max 50 characters

Attach Metadata using `Field`

- Useful in case of building APIs
- Use `Annotated` (from `typing`) & Field (from `Pydantic`)

```
# AGE VALIDATION

from pydantic import BaseModel, EmailStr, AnyUrl, Field
from typing import List, Dict, Optional, Annotated

class Patient(BaseModel):
    name: Annotated [str, Field(max_length=50, title="Name of the patient", des
cription= "Provide name of the patient", examples= ["Mat", "Jon"], default= "a
```

```
sldfikg")]
    age: int = Field(gt=0, lt=100)

info= {"name": "randy", "age": 52}

patient1= Patient(**info)
patient1
```

```
Patient(name='randy', age=52)
```

Suppress Type Coercing

- Pydantic smartly converts `'30'` → `30`
- But this ain't good all the time
- We can suppress this with the help of `Field()` → **`strict=True`**

```
# Suppress Type Coercing

from pydantic import BaseModel, EmailStr, AnyUrl, Field
from typing import List, Dict, Optional, Annotated

class Patient(BaseModel):
    name: Annotated [str, Field(max_length=50)]
    age: int = Field(gt=0, lt=100, strict=True)

info= {"name": "randy", "age": '52'} # We are passing str instead of int

patient1= Patient(**info)
patient1
```

```
---------------------------------------------------------------------------
ValidationError                           Traceback (most recent call last)
Cell In[30], line 12
      8     age: int = Field(gt=0, lt=100, strict=True)
     10 info= {"name": "randy", "age": '52'}
---> 12 patient1= Patient(**info)
     13 patient1

File d:\Python_Env\LangChain\venv\lib\site-packages\pydantic\main.py:253, in BaseModel.__init__(self, **data)
    251 # `__tracebackhide__` tells pytest and some other tools to omit this function from tracebacks
    252 __tracebackhide__ = True
--> 253 validated_self = self.__pydantic_validator__.validate_python(data, self_instance=self)
    254 if self is not validated_self:
    255     warnings.warn(
    256         'A custom validator is returning a value other than `self`.\n'
    257         "Returning anything other than `self` from a top level model validator isn't supported when validating via `__init__`.\n"
    258         'See the `model_validator` docs (https://docs.pydantic.dev/latest/concepts/validators/#model-validators) for more details.',
    259         stacklevel=2,
    260     )

ValidationError: 1 validation error for Patient
age
  Input should be a valid integer [type=int_type, input_value='52', input_type=str]
    For further information visit https://errors.pydantic.dev/2.11/v/int_type
```

```
# Suppress Type Coercing

from pydantic import BaseModel, EmailStr, AnyUrl, Field
from typing import List, Dict, Optional, Annotated

class Patient(BaseModel):
    name: Annotated [str, Field(max_length=50)]
    age: Annotated [int, Field(gt=0, lt=100, strict=True)]

info= {"name": "randy", "age": 45}

patient1= Patient(**info)
patient1
```

```
Patient(name='randy', age=45)
```

Field Validator

`from` `pydantic` `import` `field_validator`

Check the email domain → hdfc.com

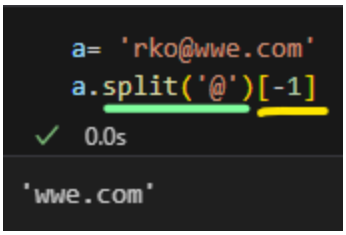
- Make transformation → Capitalize the name

- Make a method with decorator `@field_validator` & `@classmethod`

- We check if `hdfc.com` is present in the `value`

Extract the part after `@` :

- **Split the string with** `@`
- Check the last part




```
# Field Validator- Check email domain

from pydantic import BaseModel, EmailStr, AnyUrl, Field, field_validator
from typing import List, Dict, Optional, Annotated

class Patient(BaseModel):
    name: str
    age: int
    email: str

    @field_validator('email')
    @classmethod

    def email_validator(cls, value):
        valid_domains= ['hdfc.com', 'bank.com']
        domain_name = value.split('@')[-1]
```

```
        if domain_name not in valid_domains:
            raise ValueError('Not a valid domain')
        return value

info= {"name": "randy", "age": 45, 'email': 'rko@wwe.com'} #email domain is not valid

patient1= Patient(**info)
patient1
```

```
---------------------------------------------------------------------------
ValidationError                           Traceback (most recent call last)
Cell In[39], line 24
     20         return value
     22 info= {"name": "randy", "age": 45, 'email': 'rko@wwe.com'}
---> 24 patient1= Patient(**info)
     25 patient1

File d:\Python_Env\LangChain\venv\lib\site-packages\pydantic\main.py:253, in BaseModel.__init__(self, **data)
    251 # `__tracebackhide__` tells pytest and some other tools to omit this function from tracebacks
    252 __tracebackhide__ = True
--> 253 validated_self = self.__pydantic_validator__.validate_python(data, self_instance=self)
    254 if self is not validated_self:
    255     warnings.warn(
    256         'A custom validator is returning a value other than `self`.\n'
    257         "Returning anything other than `self` from a top level model validator isn't supported when validating via `__init__`.\n"
    258         'See the `model_validator` docs (https://docs.pydantic.dev/latest/concepts/validators/#model-validators) for more details.',
    259         stacklevel=2,
    260     )

ValidationError: 1 validation error for Patient
email
  Value error, Not a valid domain [type=value_error, input_value='rko@wwe.com', input_type=str]
    For further information visit https://errors.pydantic.dev/2.11/v/value_error
```

```
a= 'rko@wwe.com'
a.split('@')[-1]
```

✓ 0.0s

```
'wwe.com'
```

```
# Field Validator- Check email domain

from pydantic import BaseModel, EmailStr, AnyUrl, Field, field_validator
from typing import List, Dict, Optional, Annotated

class Patient(BaseModel):
    name: str
    age: int
    email: EmailStr
```

```
    @field_validator('email')
    @classmethod

    def email_validator(cls, value):
        valid_domains= ['hdfc.com', 'bank.com']
        domain_name = value.split('@')[-1]

        if domain_name not in valid_domains:
            raise ValueError('Not a valid domain')
        return value

info= {"name": "randy", "age": 45, 'email': 'rko@hdfc.com'} #valid domain

patient1= Patient(**info)
patient1
```

```
Patient(name='randy', age=45, email='rko@hdfc.com')
```

- No error

Transformation → Capitalize the name

```
# Field Validator- Capitalize the name

from pydantic import BaseModel, EmailStr, AnyUrl, Field, field_validator
from typing import List, Dict, Optional, Annotated

class Patient(BaseModel):
    name: str
    age: int
    email: EmailStr

    @field_validator('email')
```

```
    @classmethod

    def email_validator(cls, value):
        valid_domains= ['hdfc.com', 'bank.com']
        domain_name = value.split('@')[-1]

        if domain_name not in valid_domains:
            raise ValueError('Not a valid domain')
        return value

    @field_validator('name')
    @classmethod

    def captitalize(cls, value):
        return value.capitalize()

info= {"name": "randy", "age": 45, 'email': 'rko@hdfc.com'} #name passed in small letters

patient1= Patient(**info)
patient1
```

```
Patient(name='Randy', age=45, email='rko@hdfc.com')
```

💡 The `value` **returned is the value before type coercion. (** `mode = 'after'` **) (Default).**
If you want the value before type coercion , set → `mode ='before'`

Model Validator

`from` `pydantic` `import` `model_validator`

- **Validation for more than 1 field**

Validate emergency contact for patients with 60+ age:

```
# Model validator- Check emergency contact
from pydantic import BaseModel, model_validator
from typing import Dict

class Patient (BaseModel):
    name: str
    age: int
    contact: Dict[str, str]

    @model_validator(mode='after')
    def validate_emergency_contact(cls, model):
        if model.age >60 and 'emergency' not  in model.contact:
                raise ValueError
        return model

info= {"name": "randy", "age": 30, 'contact': {'phone': '654654'} } # no error

patient1 = Patient(**info)
patient1
```

```
Patient(name='randy', age=30, contact={'phone': '654654'})
```

💡 !!write `model.age` **(not** `model['age']` **) and** `model.contact` **(not** `model['contact']` **)**

```
# Model validator- Check emergency contact

from pydantic import BaseModel, model_validator
from typing import Dict
```

```python
class Patient (BaseModel):
    name: str
    age: int
    contact: Dict[str, str]

    @model_validator(mode='after')
    def validate_emergency_contact(cls, model):
        if model.age >60 and 'emergency' not  in model.contact:
                raise ValueError
        return model

info= {"name": "randy", "age": 62, 'contact': {'phone': '654654'} } # error

patient1 = Patient(**info)
patient1
```

```
---------------------------------------------------------------------------
ValidationError                           Traceback (most recent call last)
Cell In[87], line 19
     15         return model
     17 info= {"name": "randy", "age": 62, 'contact': {'phone': '654654'} }
---> 19 patient1 = Patient(**info)
     20 patient1

File d:\Python_Env\LangChain\venv\lib\site-packages\pydantic\main.py:253, in BaseModel.__init__(self, **data)
    251 # `__tracebackhide__` tells pytest and some other tools to omit this function from tracebacks
    252 __tracebackhide__ = True
--> 253 validated_self = self.__pydantic_validator__.validate_python(data, self_instance=self)
    254 if self is not validated_self:
    255     warnings.warn(
    256         'A custom validator is returning a value other than `self`.\n'
    257         "Returning anything other than `self` from a top level model validator isn't supported when validating via `__init__`.\n"
    258         'See the `model_validator` docs (https://docs.pydantic.dev/latest/concepts/validators/#model-validators) for more details.',
    259         stacklevel=2,
    260     )

ValidationError: 1 validation error for Patient
  Value error,  [type=value_error, input_value={'name': 'randy', 'age': ...t': {'phone': '654654'}}, input_type=dict]
    For further information visit https://errors.pydantic.dev/2.11/v/value_error
```

Computed Field

- Use does not provide its value
- The value is calculated with the use of other fields

Calculate BMI with weight & height:

- BMI will be computed field

```
# Calculate BMI

from pydantic import BaseModel, model_validator, computed_field

class Patient(BaseModel):
    name: str
    weight: float     #kg
    height: float     #meter

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

info= {"name": "aj", "weight": 85, "height": 1.72}

patient1 = Patient(**info)
patient1
```

```
Patient(name='aj', weight=85.0, height=1.72, bmi=28.73)
```

`@property` decorator, meaning `bmi` acts like an attribute (you can access it like `patient1.bmi`

`@computed_field`

This is a decorator specific to the Pydantic library. Its purpose is to explicitly tell Pydantic that the decorated method is a "computed field." A computed field is a value that isn't provided when the model is initialized, but is instead calculated from other fields within the model.

Nested Models

```
from pydantic import BaseModel

class Address(BaseModel):
    city: str
    state: str
    pin: int

class Patient(BaseModel):
    name: str
    gender: str
    address: Address

address_dict = {'city': 'pune', 'state': 'MH', 'pin': 400751}

add1= Address(**address_dict)
patient_dict = {'name': 'seth', 'gender': 'm', 'address': add1}

patient1= Patient(**patient_dict)
patient1
```

```
Patient(name='seth', gender='m', address=Address(city='pune', state='MH', pin=400751))
```

- `address` field is explicitly typed as an `Address` object, which is the Pydantic model we defined earlier.

```
patient1.address.pin
```

```
40075
```

Export Pydantic models as `Dict` or `JSON`:

Dictionary:

`.model_dump()`

```
from pydantic import BaseModel

class Address(BaseModel):
    city: str
    state: str
    pin: int

class Patient(BaseModel):
    name: str
    gender: str
    address: Address

address_dict = {'city': 'pune', 'state': 'MH', 'pin': 40075}

add1= Address(**address_dict)
patient_dict = {'name': 'seth', 'gender': 'm', 'address': add1}

patient1= Patient(**patient_dict)
patient1

temp= patient1.model_dump()
temp
```

```
{'name': 'seth',
 'gender': 'm',
 'address': {'city': 'pune', 'state': 'MH', 'pin': 40075}}
```

JSON:

```
# Model → JSON

from pydantic import BaseModel

class Address(BaseModel):
    city: str
    state: str
    pin: int

class Patient(BaseModel):
    name: str
    gender: str
    address: Address

address_dict = {'city': 'pune', 'state': 'MH', 'pin': 40075}

add1= Address(**address_dict)
patient_dict = {'name': 'seth', 'gender': 'm', 'address': add1}

patient1= Patient(**patient_dict)
patient1

temp= patient1.model_dump_json()
temp
```

```
'{"name":"seth","gender":"m","address":{"city":"pune","state":"MH","pin":40075}}'
```

Chose specific fields to export:

include= []

```
patient1.model_dump(include=['name', 'gender'])
```

```
{'name': 'seth', 'gender': 'm'}
```

`exclude=[]`

```
patient1.model_dump(exclude=['name', 'gender'])
```

```
{'address': {'city': 'pune', 'state': 'MH', 'pin': 40075}}
```

Exclude state from address dictionary:

```
patient1.model_dump(exclude={'address': ['state']})
```

```
{'name': 'seth', 'gender': 'm', 'address': {'city': 'pune', 'pin': 40075}}
```

exclude_unset

- The things which are not set while creating the object will not be exported
- **Used to exclude default values.**

```
from pydantic import BaseModel

class Address(BaseModel):
    city: str
    state: str ='Maharashtra' # default
    pin: int

class Patient(BaseModel):
    name: str
    gender: str
    address: Address
```

```
address_dict = {'city': 'pune', 'pin': 40075} # State is not passed

add1= Address(**address_dict)
patient_dict = {'name': 'seth', 'gender': 'm', 'address': add1}

patient1= Patient(**patient_dict)
patient1

temp= patient1.model_dump_json(exclude_unset=True)
temp
```

```
'{"name":"seth","gender":"m","address":{"city":"pune","pin":40075}}'
```

- We set Maharashtra as default state
- BuWe do not pass the state while creating the object
- If we set `exclude_unset=True`**, it will not display the state as we haven't explicitly mentioned it while creating the object.**