Assignment 20

**PYDANTIC AND GENERICS**

**RESEARCH WORK**

**Q1: What are the advantages of using Pydantic over ordinary data classes?**

**Ans: Pydantic** has several advantages over ordinary dataclasses such as data validation in runtime, automatic nested models, automatic smart conversion such as str -> int, float -> int etc. JSON serialization. But in ordinary dataclasses these features have to be done manually.

**Q2: How does Pydantic handle data coercion and validation?**

**Ans: Pydantic** automatically validates data types in the runtime. It causes an error if correct data type is not entered. It also does type coercion (smart conversion) automatically in runtime e.g. str -> int, list -> set etc. If a number is given in quotes, it is a string but pydantic converts it to int or float automatically in runtime.

**EXAMPLE:**class User(BaseModel):

  name: str = Field(..., description="Name of the user", example="Ali")

  age: int = Field(ge=18, le=99, description="Age of the user",example=23)

  email: str = Field(..., description="Email of the user", example="Ali@example.com")

u1 = User(name='Ali', age=’23’, email="Ali@example.com")

**Q3: What is a GenericModel in Pydantic and why is it beneficial?**

**Ans: GenericModel** in pydantic let you create generic classes that can deal with data of different types with just one model. It also ensures type validation using pydantic. It ensures type safety, flexibility and reusability in code.

**EXAMPLE:**from typing import Generic, TypeVar

from pydantic.generics import GenericModel

T = TypeVar("T")

class ApiResponse(GenericModel, Generic[T]):

    status: str

    data: T

res1 = ApiResponse[int](status="success", data=42)

print(res1)

res2 = ApiResponse[str](status="success", data="Hello")

print(res2)

**Q4: How do Python Generics improve code reusability and type safety?**

**Ans: Python Generics** improve code reusability by letting you write one function that works with any data type or types restricted by the user. It also ensures type safety before the code runs. If you enter a wrong data type it will flag it before the code runs.

**EXAMPLE:**

from typing import TypeVar

T = TypeVar("T", int, float)

def double(value: T) -> T:

    return value \* 2

print(double(5))

print(double(3.14))

print(double("Hi"))

**Q5: When would you choose Pydantic validation in production over manual validation?**

**Ans:** I would choose Pydantic validation in production when I need automatic validation and type checking in my code, also when I want to add constraints or conditions in the data validation.

**EXAMPLE:**

class User(BaseModel):

  name: str = Field(..., description="Name of the user", example="Ali")

  age: int = Field(ge=18, le=99, description="Age of the user",example=23)

  email: str = Field(..., description="Email of the user", example="Ali@example.com")