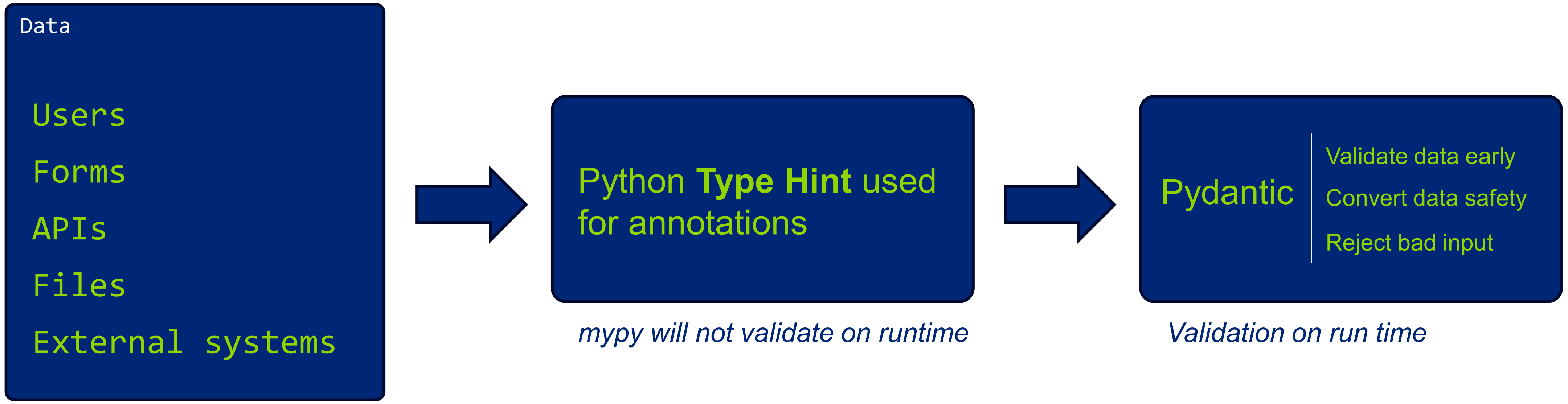
Pydantic

Python library for data validation and settings management

Why Pydantic Exists

Data can come anywhere



mypy will not validate on runtime

Validation on run time

What is Pydantic ?

- Pydantic is a data validation library.
- It uses Python type hints.
- Validation happens at runtime.
- Converts input data into clean Python objects.

```
Pydantic = Type hints + Runtime validation + Data parsing
```

Pydantic Syntax

Python library for data validation and settings management

Pydantic Syntax

Creating a Pydantic Model

- Inherits from BaseModel
- Fields are defined using type hints
- Validation happens during object creation

Behavior:

- Automatically converts types
- Raises errors if data is invalid
- Clean and readable error messages.

python

```python
from pydantic import BaseModel

class User(BaseModel):
        name: str
        age: int


# validate data
user = User(name="Amit", age=25)
```

Optional Fields, Default & Strictness

Required Fields

- All fields are required by default.

Optional Fields

- Use `Optional[T]`
- Allow None values

Strict Types

- Disable automatic type conversion
- Useful for critical fields

```python
from pydantic import BaseModel

class User(BaseModel):
        id: int
        name: str
        age: Optional[int] = None

# validate data
user = User(id=1, name="Amit", age=25)
```

RootModel

Data validation for variables, list, tuples, sets and other data structures

vedicskill.

RootModel

A **RootModel** is a special pydantic model where the entire model is just one value, instead of multiple named fields.

- A list
- A dictionary
- A single primitive (str, int, float etc)

python

```python
 from pydantic import RootModel

 class Numbers(RootModel[list[int]]):
        pass

# validate data
nums = Numbers.model_validate([1, 2, 3])
print(nums.root)
```

VEDICSKILL.

validate_call

Data validation for function arguments and return type

validate_call

A **validate_call** is a decorator function validates the input arguments and return type data types.

```python
from pydantic import validate_call

@validate_call(validate_return =True)
def add(a: int,  b: int) -> int:
        return a + b

# validate data
nums = add (2, 3)
print(nums)
```

VEDICSKILL.

Field

Validates that input data to specific value constraints.

Field Constraints

What are Field Constraints?

- Rules applied directly to fields
- Enforce limits without custom logic

Why Use Them?

- Keeps validation declarative
- Easy to read and maintain
- Self-documenting models

Common Constraints

- `gt, ge, lt, le`
- `min_length, max_length`

```python
from pydantic import BaseModel, Field

class User(BaseModel):
       id: int = Field(gt=0)
       name: str = Field(min_length=3, max_length=15)
       age: int = Field(ge=5, le=30)

# validate data
user = User(id=1, name="Amit", age=25)
```

VEDICSKILL.

Next

Async Programming

`async / await` keywords

vedicskill.