🧠 FastAPI Parameters & Request Body – Full Deep Dive 🚀

FastAPI makes parameter handling intuitive, type-safe, and well-documented. Let's break down everything!

🔹 1. Path & Query Parameters

✅ Path Parameters

Defined in the URL path like `/items/{item_id}`.

```python
@app.get("/items/{item_id}")
def read_item(item_id: int):
    return {"item_id": item_id}
```

- `item_id` is **mandatory**
- Automatically converted to `int`

✅ Query Parameters

Passed via the URL like `/items?name=phone`.

```python
@app.get("/items/")
def read_item(name: str, price: float = 0.0):
    return {"name": name, "price": price}
```

- **Optional** if default value is provided
- FastAPI infers types and validates them

🔹 2. Request Body 🧾

Used to send JSON, Form data, etc.

✅ Example with Pydantic Model:

```python
from pydantic import BaseModel

class Item(BaseModel):
    name: str
    price: float
    in_stock: bool
```

```
@app.post("/items/")
def create_item(item: Item):
    return {"received": item}
```

- Sends JSON like:

  ```
  {
    "name": "Phone",
    "price": 299.99,
    "in_stock": true
  }
  ```

- Automatic validation + documentation

◇ 3. Parameter Metadata 📎

Use `Query()`, `Path()`, and `Body()` to add **validation and metadata**.

```
from fastapi import Query, Path

@app.get("/products/{product_id}")
def get_product(
    product_id: int = Path(..., title="The ID of the product", gt=0),
    name: str = Query(None, max_length=50)
):
    return {"product_id": product_id, "name": name}
```

🏷 Common metadata options:

- `title`: Adds a title in Swagger UI
- `description`: Tooltip in docs
- `alias`: Alternate key
- `deprecated=True`: Warns in docs
- `example`: Shown as input sample

◇ 4. Validators 🛡

Validation happens **automatically** via:

- Type hints (`str`, `int`)
- `Pydantic` models
- `Query`, `Path`, `Body` metadata

You can also **custom validate** inside models:

```python
from pydantic import BaseModel, validator

class Product(BaseModel):
    name: str
    price: float

    @validator("price")
    def price_positive(cls, value):
        if value <= 0:
            raise ValueError("Price must be positive")
        return value
```

◇ 5. Multiple Values 🗐

Query parameters can accept multiple values (like a list):

```python
from typing import List

@app.get("/search/")
def search_items(tags: List[str] = Query([])):
    return {"tags": tags}
```

👉 Accessed via `/search?tags=tech&tags=fastapi`

◇ 6. Number Validators 🔢

Use built-in constraints for validation:

```python
@app.get("/range/")
def get_range(
    num: int = Query(..., gt=10, lt=100)
):
    return {"num": num}
```

🔢 Available:

- `gt` / `ge`: Greater than / Greater or equal
- `lt` / `le`: Less than / Less or equal
- `multiple_of`: Must be divisible

◇ 7. Complex Subtypes 🧬

Nested models, deeply structured request bodies:

```
class Features(BaseModel):
    size: str
    color: str

class Product(BaseModel):
    name: str
    price: float
    features: Features

@app.post("/products/")
def create_product(product: Product):
    return {"product": product}
```

☑ Swagger UI automatically nests the structure.

☑ Bonus: Mixing All Three – Path, Query, and Body

```
@app.put("/products/{product_id}")
def update_product(
    product_id: int = Path(...),
    available: bool = Query(...),
    product: Product = Body(...)
):
    return {
        "id": product_id,
        "available": available,
        "details": product
    }
```

🧠 Summary Table

Feature	Example	Tool Used
Path Param	`/items/{id}`	`Path()`
Query Param	`/items?name=x`	`Query()`
Request Body	JSON payload	`Body()`, Model
Metadata	Titles, defaults, constraints	`Path()`, `Query()`
List Values	`/tags?tag=x&tag=y`	`List[str]`
Validators	price > 0, name length, etc.	`Pydantic`, `Query()`
Number Constraints	gt, lt, multiple_of	`Query()`, `Path()`
Nested Structures	Models inside Models	Pydantic

💧 Pro Tips

- Use `Optional[]` or default `None` to make parameters optional.
- Use `Body(embed=True)` if you want request body as a single object like: `{ "item": { ... } }`
- Use `Enum` for fixed choices in query/path params.
- Use `response_model=YourModel` to enforce response types.

✍️ Author

Darshan Vasani 🚀 Website | 🔗 Linktree | 🧑‍🏫 Topmate
