**FastAPI Parameters: Detailed Explanation**

Parameters are the data sent by the client (e.g., browser, mobile app, or another tool) to the server. In FastAPI, this data can come in different ways, and each method has its specific use case. I'll explain the three main types of parameters: **Path Parameters**, **Query Parameters**, and **Request Body**.

FastAPI is a modern Python framework used for building APIs. It's extremely fast and developer-friendly. The biggest advantage of FastAPI is that it leverages Python's type hints, which automatically validate data and generate interactive documentation (like Swagger UI). This means the APIs you create come with built-in testing and documentation.

**Example**: If you want to create an API to fetch user details, FastAPI makes it very easy. You just write the code, and FastAPI handles validation and presentation.

---

**1. Path Parameters**

**Path Parameters** are variables included directly in the URL path. They are typically used to identify specific resources (e.g., a user, product, or item).

**Real-World Example**:
Suppose you're building an API for an e-commerce website. You need to fetch details for a specific product with ID `456`. The URL would look like:
```
/products/456
```
Here, `456` is the path parameter.

**Code Example**:
```python
from fastapi import FastAPI, Path

app = FastAPI()

@app.get("/products/{product_id}")
async def read_product(
    product_id: int = Path(
        ...,  # Required parameter
        title="Product ID",
        description="A unique identifier for the product",
        ge=1,  # Must be >= 1
        le=10000  # Must be <= 10000
    )
):
    return {"product_id": product_id, "message": f"Fetching details for product {product_id}"}
```

**Detailed Explanation**:

- **`@app.get("/products/{product_id}")`**: This is a GET endpoint that handles URLs like `/products/456`. `{product_id}` is a placeholder that gets replaced by the actual value (e.g., `456`).
- **`product_id: int`**: This specifies that `product_id` must be an integer. If the client sends a non-integer value (e.g., `/products/abc`), FastAPI automatically returns a 422 error.
- **`Path(...)`**: The `Path` function is used for validation and metadata:
  - `...` (ellipsis): Indicates that `product_id` is required. If the client doesn't provide it, an error is raised.
  - `title` and `description`: Metadata for documentation, visible in Swagger UI.
  - `ge=1`: Validates that `product_id` must be greater than or equal to 1.
  - `le=10000`: Ensures `product_id` is not greater than 10000.

- **Response**: If you hit `/products/456`, the response will be:
  ```json
  {
    "product_id": 456,
    "message": "Fetching details for product 456"
  }
  ```
- **Error Handling**: If you provide invalid values, like `/products/0` or `/products/15000`, FastAPI returns a 422 Unprocessable Entity error, explaining which validation rule was violated.

**Practical Use**:
- Path parameters are used when you need to fetch or update a specific resource, such as:
  - `/users/123` (details for user ID 123)
  - `/orders/789` (status of order ID 789)
- **Swagger UI**: When you open `http://localhost:8000/docs`, this endpoint will be displayed. You can input `product_id` and test it by clicking the "Execute" button. If validation fails, the error message will be clearly shown.

***Try This**:
- Run the server and try `/products/10` and `/products/0` in the browser. Observe how FastAPI handles validation errors.
- Test the endpoint in Swagger UI and check the validation rules.

---

**2. Query Parameters**

**Query Parameters** are key-value pairs that appear in the URL after a `?`. They are optional and used for filtering, sorting, or pagination.

**Real-World Example**:
You're building a search API to find products. The client can send a search query, page number, and limit:

```
/products?search=phone&skip=10&limit=20
```
Here:
- `search=phone`: Search query
- `skip=10`: Skip 10 products (for pagination)
- `limit=20`: Return 20 products

**Code Example**:
```python
from fastapi import FastAPI, Query

app = FastAPI()

@app.get("/products/")
async def search_products(
    search: str | None = Query(
        None,  # Optional parameter
        title="Search Query",
        description="Search term to filter products",
        min_length=3,
        max_length=100,
        pattern="^[a-zA-Z0-9 ]+$"  # Only alphanumeric and spaces
    ),
    skip: int = Query(0, ge=0, description="Number of products to skip"),
    limit: int = Query(10, ge=1, le=100, description="Max number of products to return")
):
    return {
        "search": search,
        "skip": skip,
        "limit": limit,
        "message": f"Searching for '{search}' with skip={skip} and limit={limit}"
    }
```

**Detailed Explanation**:

- **`@app.get("/products/")`**: This endpoint handles query parameters.
- **`search: str | None`**: Indicates that `search` can be a string or `None` (optional). If the client doesn't send `search`, it defaults to `None`.
- **`Query(...)`**:
  - `None`: Default value is `None`, meaning optional.
  - `min_length=3`: If `search` is provided, it must be at least 3 characters long.
  - `max_length=100`: Maximum allowed length is 100 characters.
  - `pattern="^[a-zA-Z0-9 ]+$"`: This regex ensures `search` contains only letters, numbers, and spaces. Special characters (e.g., `@`, `#`) are not allowed.
- **`skip: int = Query(0, ge=0)`**: `skip` is an integer with a default value of 0 and cannot be negative.
- **`limit: int = Query(10, ge=1, le=100)`**: `limit` defaults to 10, must be at least 1, and cannot exceed 100.
- **Response**: If you hit `/products?search=phone&skip=10&limit=20`, the response will be:
  ```json
  {
    "search": "phone",
    "skip": 10,
    "limit": 20,
    "message": "Searching for 'phone' with skip=10 and limit=20"
  }
  ```
- **Error Handling**: If you provide invalid values, like `search=ab` (too short) or `limit=150` (too high), FastAPI returns a 422 error, specifying which rule was violated.

**Practical Use**:
- Query parameters are used for filtering or paginating data, such as:

  - Search APIs (`/products?category=electronics`)
  - Pagination (`/users?skip=20&limit=10`)
  - Sorting (`/items?sort=price&order=desc`)
- **Swagger UI**: In the docs, you'll see input fields for `search`, `skip`, and `limit`. Try invalid values to see how validation errors are displayed.

**Try This**:
- Run the server and try `/products?search=phone&skip=5&limit=15`.
- Test invalid inputs, like `search=a` (too short) or `limit=200` (too high), and check the error messages.
- Use Swagger UI to test regex validation (e.g., try `search=phone@123`).

---

**3. Request Body**

**Request Body** is used when the client needs to send complex data to the server, typically in JSON format. It's common with POST, PUT, or PATCH requests, such as creating or updating resources.

**Real-World Example**:
You're building an API to create a new product. The client sends JSON data:
```json
{
  "name": "Smartphone",
  "description": "Latest model with 5G support",
  "price": 699.99
}
```

#### **Code Example**:
```python
from fastapi import FastAPI, Path, Query, Body
```

```
from pydantic import BaseModel

class Product(BaseModel):
    name: str
    description: str | None = None
    price: float

app = FastAPI()

@app.post("/products/validated/{product_id}")
async def create_product(
    product_id: int = Path(..., title="Product ID", ge=1),
    search: str | None = Query(None, min_length=3,
description="Optional search term"),
    product: Product = Body(..., description="Product data (JSON
body)")
):
    result = {
        "product_id": product_id,
        "message": f"Created product with ID {product_id}"
    }
    if search:
        result.update({"search": search})
    if product:
        result.update({"product": product.model_dump()})
    return result
```

**Detailed Explanation**:

- **`class Product(BaseModel)`**: This Pydantic model defines the structure of the request body:
  - `name: str`: Product name, required field.
  - `description: str | None = None`: Description is optional.
  - `price: float`: Price is required and must be a float.
- **`@app.post("/products/validated/{product_id}")`**: This POST endpoint handles a path parameter (`product_id`), query parameter (`search`), and request body (`product`).

- **`product: Product = Body(...)`**: Specifies that the request body must contain JSON data matching the `Product` type and is required.
- **Request Example**: You can send this JSON via Swagger UI:
  ```json
  {
    "name": "Smartphone",
    "description": "Latest model with 5G support",
    "price": 699.99
  }
  ```
- **Response**: If you hit `/products/validated/123?search=phone`, the response will be:
  ```json
  {
    "product_id": 123,
    "search": "phone",
    "product": {
      "name": "Smartphone",
      "description": "Latest model with 5G support",
      "price": 699.99
    },
    "message": "Created product with ID 123"
  }
  ```
- **Error Handling**: If you send invalid JSON, like `price="abc"` (string instead of float) or omit `name`, FastAPI returns a 422 error, specifying which field is incorrect.

**Practical Use**:
- Request body is used for sending complex data, such as:
  - Creating a new user (`POST /users`)
  - Updating a product (`PUT /products/123`)
  - Form submission (`POST /forms`)
- **Swagger UI**: In the docs, you'll get a JSON editor to input `Product` data. Try invalid data (e.g., `price="invalid"`) and check the error messages.

**Try This**:
- Run the server and test the POST request via Swagger UI.
- Send valid and invalid JSON (e.g., missing `name` or incorrect `price`) and observe the responses.
- Compare the response and error messages.

---

**Complete Code**

Here's the complete `main.py` code combining all three examples:

```python
from fastapi import FastAPI, Path, Query, Body
from pydantic import BaseModel

app = FastAPI()

class Product(BaseModel):
    name: str
    description: str | None = None
    price: float

# Path Parameter Example
@app.get("/products/{product_id}")
async def read_product(
    product_id: int = Path(
        ..., title="Product ID", description="A unique identifier for the product", ge=1, le=10000
    )
):
    return {"product_id": product_id, "message": f"Fetching details for product {product_id}"}

# Query Parameter Example
@app.get("/products/")
```

```
async def search_products(
    search: str | None = Query(
        None, title="Search Query", description="Search term to filter
products",
        min_length=3, max_length=100, pattern="^[a-zA-Z0-9 ]+$"
    ),
    skip: int = Query(0, ge=0, description="Number of products to
skip"),
    limit: int = Query(10, ge=1, le=100, description="Max number of
products to return")
):
    return {
        "search": search,
        "skip": skip,
        "limit": limit,
        "message": f"Searching for '{search}' with skip={skip} and
limit={limit}"
    }

# Request Body Example
@app.post("/products/validated/{product_id}")
async def create_product(
    product_id: int = Path(..., title="Product ID", ge=1),
    search: str | None = Query(None, min_length=3,
description="Optional search term"),
    product: Product = Body(..., description="Product data (JSON
body)")
):
    result = {
        "product_id": product_id,
        "message": f"Created product with ID {product_id}"
    }
    if search:
        result.update({"search": search})
    if product:
        result.update({"product": product.model_dump()})
    return result
```

```

---

### **Running the Application**

1. **Setup**:
   ```bash
   uv init fastapi_p1
   cd fastapi_p1
   uv venv
   source .venv/bin/activate
   uv add "fastapi[standard]"
   ```

2. **Save Code**:
   Save the above code in `main.py`.

3. **Run Server**:
   ```bash
   fastapi dev main.py
   ```

4. **Test in Docs**:
   Open `http://localhost:8000/docs` in your browser and test the endpoints.

---

### **Key Points (Remember)**

1. **Path Parameters**:
   - Part of the URL path.
   - Use `Path()` for validation (e.g., `ge`, `le`, `pattern`).
   - Used for specific resources.
2. **Query Parameters**:
   - Appear after `?` in the URL.
   ```

**- Use `Query()` for validation (e.g., `min_length`, `max_length`, `regex`).**
**- Used for filtering, sorting, and pagination.**
**3. **Request Body**:**
**- Use Pydantic models for JSON data.**
**- Ideal for complex data (e.g., forms, objects).**
**4. **Validation**:**
**- FastAPI automatically returns 422 errors for validation failures.**
**- Error messages are detailed, helping clients understand issues.**
**5. **Swagger UI**:**
**- Test interactively at `http://localhost:8000/docs`.**
**- Each endpoint shows input fields and clear error messages.**

**---**

**### **Practical Tips for Projects****

**- **E-commerce API**: Use path parameters for product IDs (`/products/123`), query parameters for filtering (`/products?category=electronics`), and request body for creating new products (`POST /products`).**
**- **User Management**: Use path parameters for user profiles (`/users/456`), query parameters for search or pagination (`/users?role=admin&limit=10`), and request body for user creation (`POST /users`).**
**- **Error Handling**: Always define validations to prevent clients from sending incorrect data. This is crucial for security and reliability.**