FastAPI Basics: Essential Concepts & Setup

1. What is FastAPI?

FastAPI is a modern, high-performance web framework for building APIs with Python, based on standard Python type hints.

- **Key Features:** Extremely fast performance, automatic data validation, serialization, and interactive documentation (OpenAPI/Swagger UI).
- **Foundation:** Built on **Starlette** (for routing and networking) and **Pydantic** (for data handling).

2. Initial Setup

Before starting, you need Python and two packages: `fastapi` and an ASGI server like `uvicorn`.

Bash

```
# 1. Install FastAPI and standard dependencies (includes Pydantic and Uvicorn)
pip install "fastapi[standard]"

# 2. Run the application (assuming your file is main.py)
uvicorn main:app --reload
```

---

3. The Minimal Working Application

Every FastAPI application starts with importing the class, creating an instance, and defining a **Path Operation**.

File: `main.py`	Concept
`from fastapi import FastAPI`	**Import:** Import the main `FastAPI` class.
`app = FastAPI()`	**Instance:** Create the main application object.
`@app.get("/")`	**Decorator/Route:** Defines a **Path Operation** that handles **GET** requests to the root path (`/`).
`def read_root():`	**Function:** The code that runs when the request is received.

File: `main.py`	Concept
`return {"message": "Hello, World"}`	**Response:** Returns a Python dictionary, which FastAPI automatically converts to a JSON response.

Code Example:

Python

```python
# main.py
from fastapi import FastAPI

app = FastAPI()

@app.get("/")
def read_root():
    return {"message": "Hello, FastAPI"}
```

---

4. Path and Query Parameters (Reading Data)

FastAPI uses Python **type hints** to automatically validate and process data coming from the request.

A. Path Parameters

Values embedded directly in the URL path.

Concept	Explanation
Syntax	Use curly braces in the path: `/items/{item_id}`
Type Hint	Use standard Python types (e.g., `item_id: int`) to ensure the input is valid.

Code Example: Path Parameter

Python

```python
from fastapi import FastAPI
app = FastAPI()

@app.get("/items/{item_id}")
def read_item(item_id: int): # Automatically validates item_id as an integer
    return {"item_id": item_id, "type": type(item_id).__name__}
```

B. Query Parameters

Optional key-value pairs appended to the URL after a `?` (e.g., `?limit=10`).

Concept	Explanation
Syntax	Defined as function parameters that *do not* appear in the path.
Optional	Set a default value of `None` or use the `typing.Union` helper.
Required	Do not assign a default value.

Code Example: Query Parameter

Python

```python
from typing import Union
from fastapi import FastAPI
app = FastAPI()

@app.get("/search/")
def search_products(query: str, limit: Union[int, None] = 10):
    # 'query' is required (no default value)
    # 'limit' is optional with a default of 10
```

```
    return {"query": query, "limit": limit}

# Example URL: http://127.0.0.1:8000/search/?query=shoes&limit=5
```

---

5. Request Body and Pydantic (Writing Data)

For data like a new user or item, you use a **Request Body** (typically JSON) sent with **POST** or **PUT** requests. FastAPI uses **Pydantic** models to define the shape and types of the body data.

A. Define the Pydantic Model

Create a class that inherits from `pydantic.BaseModel` to define your data structure.

Python

```
from pydantic import BaseModel

# Defines the expected data structure for an Item
class Item(BaseModel):
    name: str             # Required string
    price: float          # Required float
    description: Union[str, None] = None # Optional string with a default of None
```

B. Use the Model in a Path Operation

Use the Pydantic model as a type hint in your function.

Concept	Explanation
Decorator	Use `@app.post()` for creating new data.
Argument	The function argument (`item: Item`) tells FastAPI to expect the request body to match the `Item` model.
Validation	If the request body doesn't match the model (e.g., missing `name` or `price` is not a number), FastAPI automatically returns a clear error.

Code Example: Request Body

Python

```python
# main.py (continued)
from fastapi import FastAPI
from pydantic import BaseModel

class Item(BaseModel):
    name: str
    price: float
    description: Union[str, None] = None

app = FastAPI()

# Handles POST requests to create a new item
@app.post("/items/")
def create_item(item: Item):
    # The 'item' object is an instance of the Pydantic Item class
    return {"message": "Item created successfully", "item_name": item.name}

# Example Request (JSON Body for POST /items/):
# {
#   "name": "Laptop",
#   "price": 1200.50,
#   "description": "Powerful machine."
# }
```