FastAPI

```
pip install fastapi
```

```
pip install uvicorn
```

- This is the server to run FastAPI

FastAPI is a modern, fast (high-performance), web framework for building APIs with Python 3.7+ based on standard Python type hints. Here are the key basics:

Key Features

- **Fast**: Very high performance, on par with NodeJS and Go (thanks to Starlette and Pydantic)
- **Easy**: Designed to be easy to use and learn
- **Standards-based**: Based on (and fully compatible with) the open standards for APIs (OpenAPI and JSON Schema)
- **Automatic docs**: Automatic interactive API documentation (Swagger UI and ReDoc)
- **Type hints**: Uses Python type hints for data validation and editor support

⚡ Why Use FastAPI?

- ✅ **Fast**: Performance is comparable to Node.js and Go
- ✅ **Easy to use**: Write less code, more readable
- ✅ **Validation built-in**: You don't need to manually check inputs
- ✅ **Auto documentation**: Swagger UI and Redoc are built-in
- ✅ **Async support**: Runs non-blocking code easily

⚙️ How FastAPI Works Internally?

Here's a simple workflow:

1. You **define** your API using Python functions (just like writing a function in Python).
2. You **use decorators** (like `@app.get()` or `@app.post()`) to connect a URL with the function.
3. FastAPI automatically:
   - Validates data using **Pydantic**
   - Runs your logic
   - Converts output to JSON
   - Generates **interactive docs** (Swagger, Redoc)
4. You run the server using **Uvicorn** (an ASGI server).

FastAPI Code

```
from fastapi import FastAPI
```

Make an instance:

```
app=FastAPI()
```

```
@app.get("/hello")

async def hello():
    return "Welcome"
```

- **`@app.get("/hello")`**:

- This is a **decorator** that tells FastAPI to handle GET requests at the path `/hello`.
- When a client (like a browser or Postman) makes a GET request to `/hello`, this function will be called.

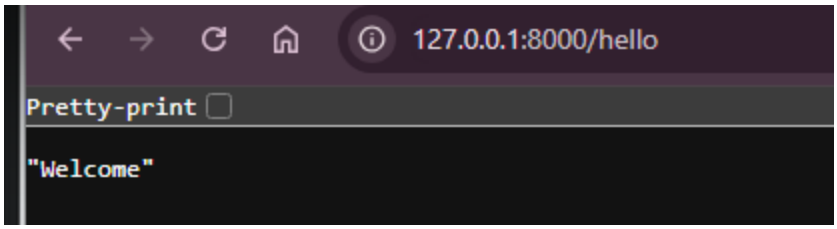
- `async def hello():`:
  - This defines an **asynchronous function** named `hello`.
  - Using `async def` means the function can handle asynchronous operations (like calling a database or another API) without blocking the application.
- `return "Welcome"`:
  - This returns a plain string `"Welcome"` as the response.
  - FastAPI will automatically convert this to a valid HTTP response (typically with content-type `application/json` or `text/plain` depending on how it's configured).

Run the code:

```
uvicorn main:app --reload
```

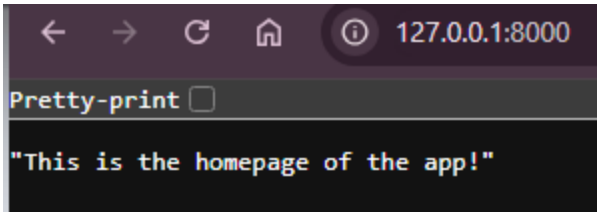
```
$ uvicorn main:app --reload
INFO:     Will watch for changes in these directories: ['D:\\Python Env\\venv']
INFO:     Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO:     Started reloader process [17976] using StatReload
INFO:     Started server process [25688]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
```

- Run the link → *http://127.0.0.1:8000/**hello***



You can replace /hello with anything.

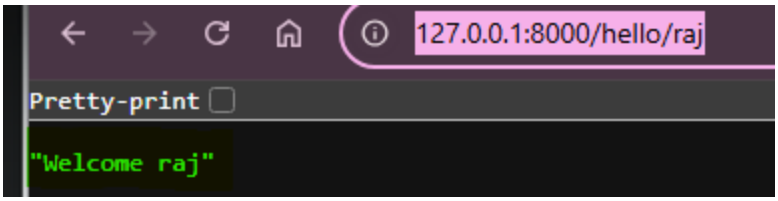
```
@app.get("/")

def read_root():
    return "This is the homepage of the app!"
```

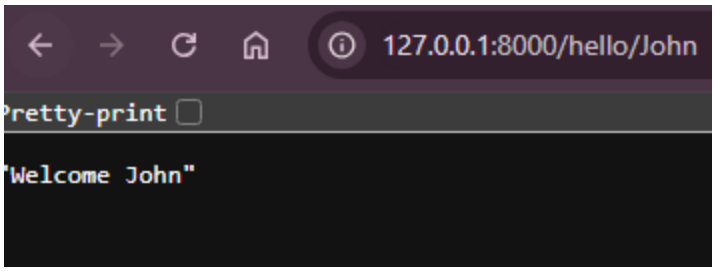


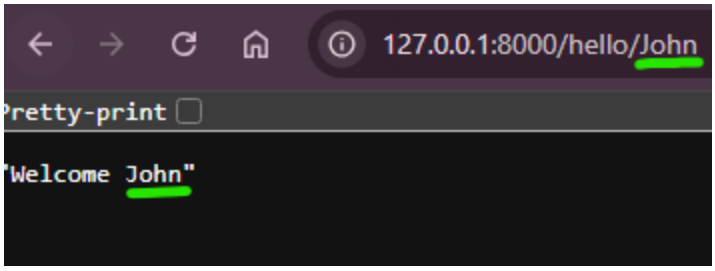
Add a custom name:

```
@app.get("/hello/{name}")

async def hello(name):
    return f"Welcome {name}"
```

run → ***http://127.0.0.1:8000/hello/raj***

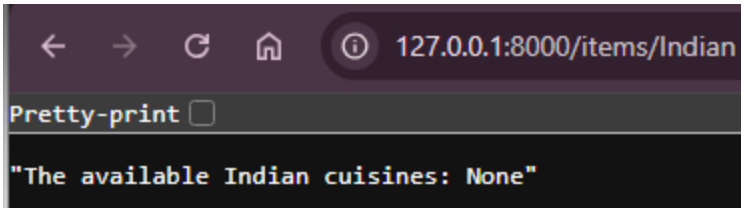




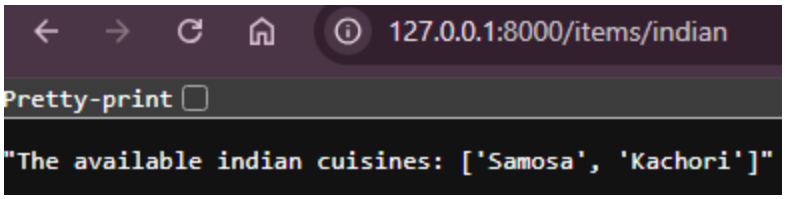



```
food= {
    "indian": ["Samosa","Kachori"],
    "italian": ["Pizza","Pasta"],
    "chinese": "Snakes"
}

@app.get("/items/{cuisine}")

def get_cuisine(cuisine):
    return f"The available {cuisine} cuisines: {food.get(cuisine)}"
```

http://127.0.0.1:8000/items/Indian

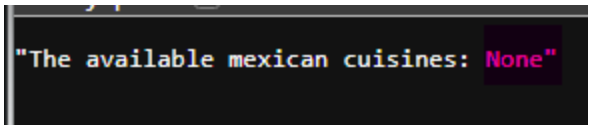


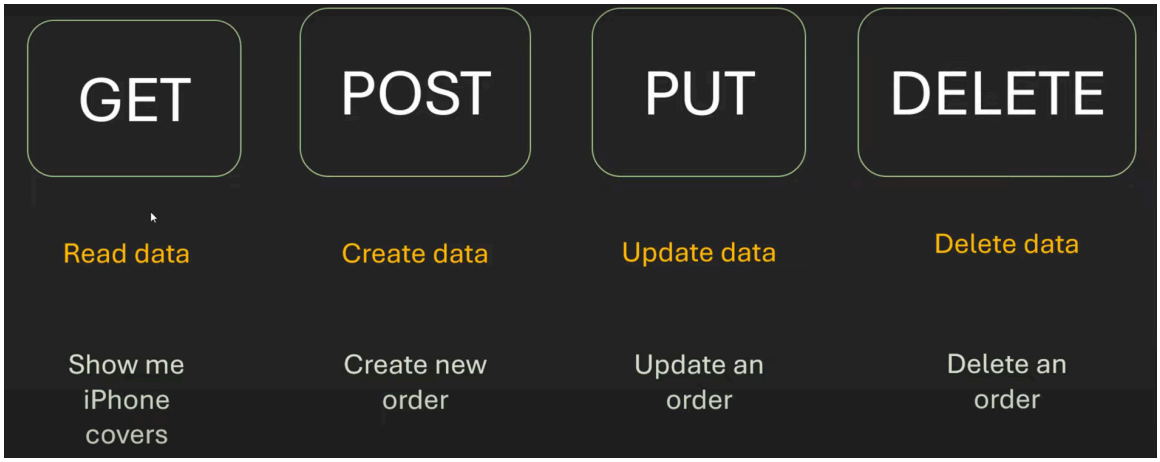
http://127.0.0.1:8000/items/indian



💡 **URLs are case sensitive.**

http://127.0.0.1:8000/items/**mexican**





Handle Error in FastAPI

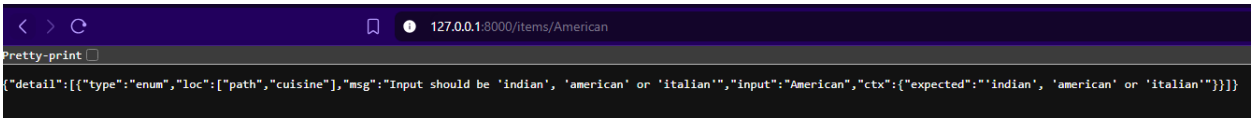
```
from enum import Enum
```

```
class AvailableCuisines(str, Enum):
    indian = "indian"
```

```
    american = "american"
    italian = "italian"
```

```
@app.get("/items/{cuisine}")

async def get_cuisine(cuisine: AvailableCuisines):
    return f"The available {cuisine} cuisines: {food.get(cuisine)}"
```

http://127.0.0.1:8000/items/**American**



- If the item is not in the list, we get this error instead of just “none”

`async def` vs `def`

🧱 Synchronous (`def`)

- When a request comes to your API, FastAPI sends it to the `def` function.
- If this function takes **5 seconds**, **no other request can be served** by that process during that time.
- Imagine a restaurant with **one chef** – if the chef is busy cooking one dish, others must wait.

⚙️ Asynchronous (`async def`)

- When a request is sent to an `async def` function:
  - FastAPI can **pause** the function when it reaches an `await` point.
  - Meanwhile, it can **start handling other requests**.
- It's like a chef with many kitchen assistants. While one dish is baking, the chef starts another.

```
import asyncio
from fastapi import FastAPI

app = FastAPI()

@app.get("/sync")
def slow_sync():
    import time
    time.sleep(5)  # This blocks everything
    return {"message": "Sync done"}

@app.get("/async")
async def slow_async():
    await asyncio.sleep(5)  # Non-blocking delay
    return {"message": "Async done"}
```

🔗 When Should You Use `async def`?

✅ Use `async def` when:

- Talking to **databases** using async drivers (like `asyncpg`, `encode/databases`)
- Calling **external APIs** using `httpx` or `aiohttp`
- Doing anything that takes time but doesn't use CPU (I/O-bound)

❌ Avoid `async def` if:

- You're doing **CPU-heavy** tasks (math, machine learning). Use `def` + background tasks/threading.
- Your libraries don't support async (e.g., many standard libraries)

🔍 Summary: `async def` vs `def` in FastAPI

Feature	`def` (Synchronous)	`async def` (Asynchronous)
Blocking	Yes – blocks the server while running	No – doesn't block, allows other tasks to run

Feature	`def` (Synchronous)	`async def` (Asynchronous)
Speed (for I/O operations)	Slower for concurrent tasks	Much faster for concurrent I/O (e.g., DB, API)
Use Case	Simple logic, CPU-heavy tasks	I/O tasks like DB queries, HTTP requests
Requires `await`?	❌ Cannot use `await` inside	✅ Can use `await` with async libraries
Supported by FastAPI?	✅ Yes	✅ Yes

Documentation

FastAPI automatically generates documentation:

- Interactive docs: http://127.0.0.1:8000/docs (Swagger UI)
- Alternative docs: http://127.0.0.1:8000/redoc