# Fast API

- External Python library
- Installed as: **pip install "fastapi[standard]"**
- Modern, Fast web framework for building APIs with Python
- Designed to be easy for developers

Analogy

- Think of your ML model as a **chef** who prepares a dish (prediction).
- FastAPI is the **waiter**: it takes orders (requests), passes them to the kitchen (model), and returns the dish (response) to the customer.
- Pydantic is the menu order form — it ensures orders are valid before they reach the chef.

# Terminologies

1. API ➔ Web applications using the HTTP protocol to transmit structured data
2. Web Application ➔ Application that serves traffic over the web
3. Web Framework ➔ Software framework that helps build web applications

- FastAPI is a web framework that helps to create API
- Other web frameworks: Flask, Django

# Notes

1. Can't run the FastAPI server with the "Run this code" button
2. Define server code in the Python editor as main.py
3. Run it from the terminal as **fastapi dev main.py**
4. Verify that the logs shows Application startup complete.
5. Stop the live server by pressing Control + C in the same terminal

# Sample

```
from fastapi import FastAPI

app = FastAPI()

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

# Now run this file as: **fastapi dev main.py**

# GET Operation

- HTTP protocol has several type of operation depending upon either we need to get, add, modify or delete something
- GET is the most common request
- Example: https://www.google.com:443/search?q=fastapi

Key Parts in above API request

- Host: www.google.com
- Port: 443
- Path: /search
- Query String: ?q=fastapi

# Query Param

- Path: /hello
- Query Parameter: “name” (default value: Rabindra)
- Request can be sent from web browser

```
@app.get("/hello")
def hello(name: str = "Rabindra"):
    return {"message": f"Hello {name}"}

@app.get("/hello_v2")
def hello(name: str = "Rabindra", addr: str = "Kathmandu"):
    return {"message": f"Hello {name} from {addr}"}

student_info = {1: {"Name": "A", "Addr": "KTM"}, 2:{"Name": "B", "Addr": "POK"}}

@app.get("/student_info/{student_id}")
def hello(student_id: int):
    return student_info.get(student_id)
```

http://127.0.0.1:8000/hello
http://127.0.0.1:8000/hello_v2?name=rasf&addr=Nep

# From Code

## Code

```
import requests

base_url = "http://127.0.0.1:8000/hello_v2"

params = {"name": "ABC", "addr": "KTM",}

response = requests.get(base_url, params=params)
response.raise_for_status()
data = response.json()
print("API Response:")
print(data)
```

# Post Operation

- **POST** is used for creating or adding new record
- Requires application to invoke post operation
- Data that is going to be added is passed as json

# Sample Code

## Code

```
@app.post("/student_info")
def add_student_info(data: dict):
    name = data.get("name")
    address = data.get("address")
    last_id = max(student_info.keys())
    current_data = {last_id + 1: {"Name": name, "Addr": address}}
    student_info.update(current_data)
    return {"message": "Record Created Successfully"}
```

## Uses in app

```
base_url = "http://127.0.0.1:8000/student_info"
response = request.post(base_url, json={"name":"New User", "address": "BKT"})
print(response.json())
```

# Put Operation

- **PUT** is used for updating existing record
- Requires application to invoke post operation
- Data that is going to be updated is passed as json

# Sample Code

## Code

```
@app.put("/student_info")
def add_student_info(data: dict):
    id = data.get("id")
    if id is None:
        return {"message": "ID required to update existing record"}
    name = data.get("name")
    address = data.get("address")
    student_info[id] = {"Name": name, "Addr": address}
    return {"message": "Record Updated Successfully"}
```

## Uses in app

```
base_url = "http://127.0.0.1:8000/student_info"
response = request.put(base_url, json={"id" : 3, "name":"New User", "address": "BKT"})
print(response.json())
```

# Delete Operation

- **DELETE** is used to remove existing record
- Requires application to invoke post operation

# Sample Code

## Code

```
@app.delete("/student_info/{student_id}")
def add_student_info(student_id: int):
    if student_id not in student_info:
        return {"message": "ID not found in existing record"}

    deleted_record = student_info.pop(student_id)
    return {"message": "Record Deleted Successfully", "deleted": deleted_record}
```

## Uses in app

```
base_url = "http://127.0.0.1:8000/student_info/2"
response = request.delete(base_url)
print(response.json())
```

# HTTP Status Codes

- Enables API to provide status in response. Success, failure, error etc.
- Specific code defined in HTTP protocol
- Range: 100 – 599
- Categorized by first number (1 – 5)

Responses

1. Informational Response: 100 – 199
2. Successful Response: 200 – 299
3. Redirection message: 300 – 399
4. Client error responses: 400 – 499
5. Server error response: 500 - 599

# Common Status Codes

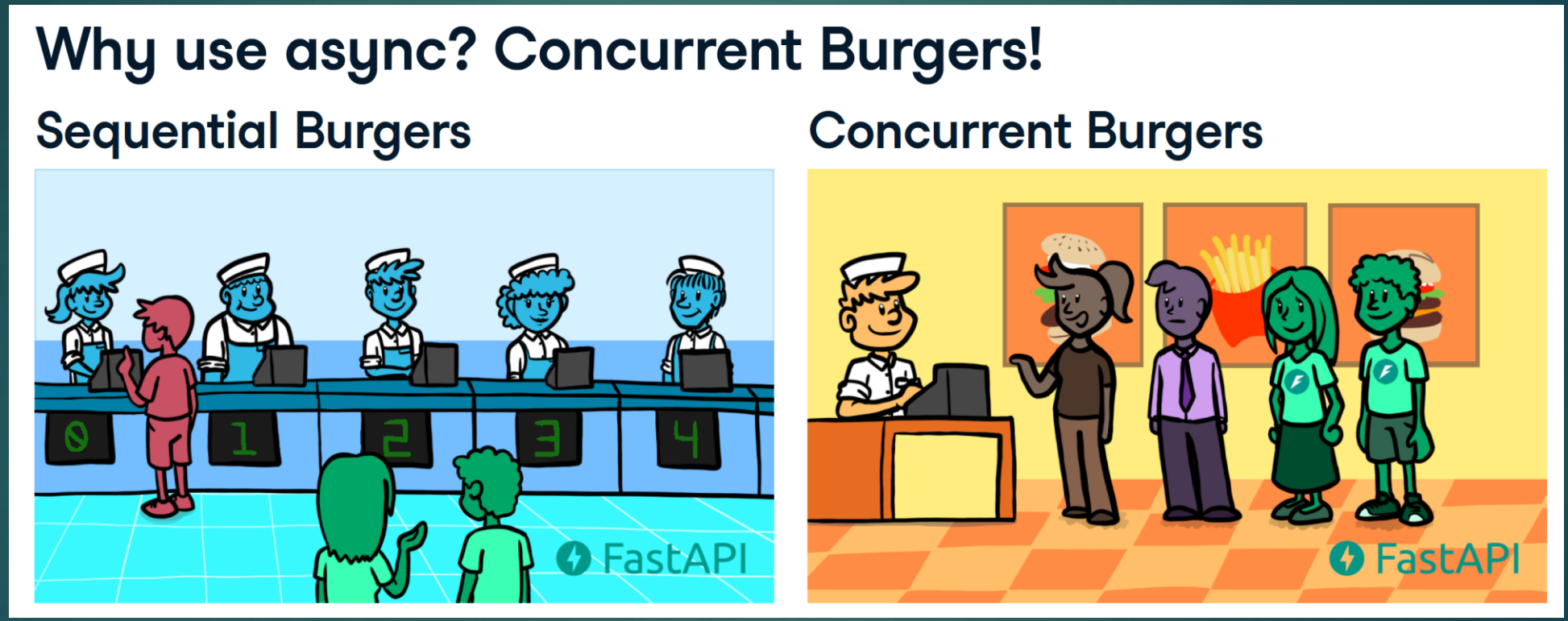
Success ( 200 – 299)

- 200 OK ➔ Default successes response
- 201 Created ➔ Specific to POST operation
- 202 Accepted ➔ Noncommittal. "Working on it"
- 204 No Content ➔ Success! Nothing more to say

Other Responses

- 301 Moved Permanently ➔ URI change permanently
- 400 Bad Request ➔ Client Error
- 404 Not Found ➔ Server cannot find the requested resource
- 500 Internal Server Error ➔ Server has encountered error

# Async for concurrent work



# Asynchronous Processing

- We can write asynchronous function using **async def**
- Allows Fast API to handle multiple concurrent request
- Synchronous handles one request at a time. If process is slow its blocked until server completes the processing

Advantages

- Better Performance: For **I/O heavy tasks** (query, file read, API call)
- Scalability: Handle many simultaneous users
- Responsiveness: App feels better as it doesn't block long running task

# Analogy

Imagine a chef in a restaurant

- **Synchronous (blocking)**: The chef cooks one dish at a time. If pasta is boiling, the chef just stands and waits until it's done before starting the next dish.

- **Asynchronous (non-blocking)**: While pasta is boiling, the chef starts chopping vegetables for another dish. The chef switches between tasks efficiently, so more meals are prepared at once.

# Synchronous

```
import datetime
import time

def task(name, duration):
    print(f"{datetime.datetime.now()}: Task {name} started")
    time.sleep(duration)  # Simulate a time-consuming task
    print(f"Task {name} finished after {duration} seconds")
    return f"Result of {name}"

def main():
    print("Synchronous Processing Demo\n")

    # Task 1
    result1 = task("A", 2)

    # Task 2
    result2 = task("B", 3)

    # Task 3
    result3 = task("C", 1)

    print("\nAll tasks finished")
    print(result1, result2, result3)

if __name__ == "__main__":
        main()
```

# Asynchronous

```
import asyncio
import datetime

async def task(name, duration):
    print(f"{datetime.datetime.now()}: Task {name} started")
    await asyncio.sleep(duration)  # Simulate a time-consuming task asynchronously
    print(f"Task {name} finished after {duration} seconds")
    return f"Result of {name}"

async def main():
    print("Asynchronous Processing Demo\n")

    # Schedule all tasks concurrently
    tasks = [
        asyncio.create_task(task("A", 2)),
        asyncio.create_task(task("B", 3)),
        asyncio.create_task(task("C", 1)), ]

    # Wait for all tasks to complete and collect results
    results = await asyncio.gather(*tasks)

    print("\nAll tasks finished")
    print(*results)

if __name__ == "__main__":
    asyncio.run(main())
```

# Asynchronous

## File Read

```
import asyncio
import aiofiles

async def read_file(filename):
    async with aiofiles.open(filename, mode='r') as f:
        content = await f.read()
    print(f"{filename} content length: {len(content)}")
    return content
```

## API Call

```
import aiohttp

async def fetch_url(url):
    async with aiohttp.ClientSession() as session:
        async with session.get(url) as response:
            data = await response.text()
            print(f"Fetched {len(data)} characters from {url}")
            return data
```