ATTENTION

JOKER'S DEV CHALLENGE

Collaborate in teams of 1-4 to design, implement and test an API to interact with the Trio Triumph Dealer.

PYTHON API MENTORSHIP

Learn API design and integration. Learn game mechanics implementation. Practice testing and debugging techniques.

CHALLENGE ON 27TH MAR

19:00-02:00

VENUE - C101, LHC

MENTORSHIP WILL TAKE PLACE 1 DAY PRIOR

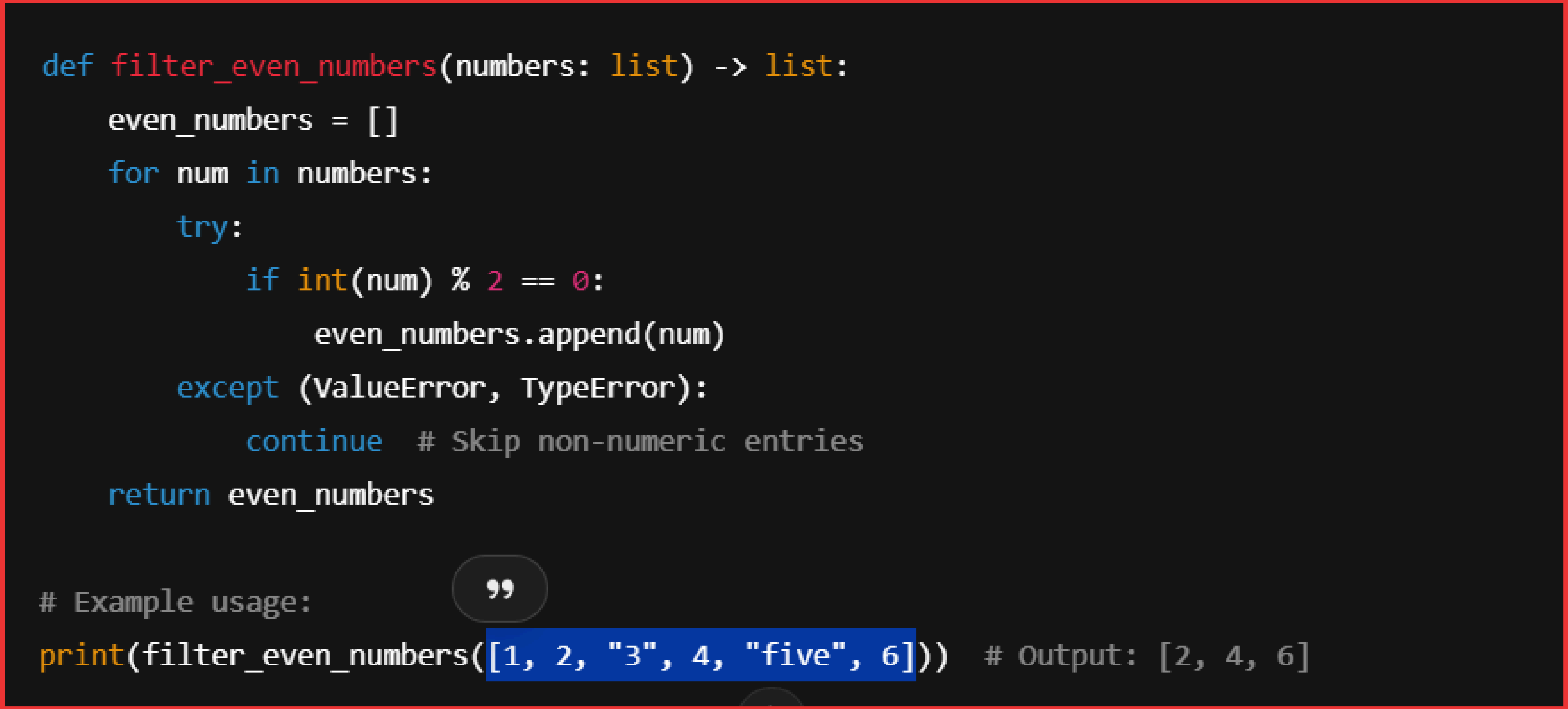
CASH PRIZES | CERTIFCATES | REFRESHMENTS

BASIC PYTHON PRACTICE

Write a function filter_even_numbers(numbers: list) -> list that returns only even numbers from the input list.

Test Case → [1, 2, "3", 4, “5”, 6]

BASIC PYTHON PRACTICE



```python
def filter_even_numbers(numbers: list) -> list:
    even_numbers = []
    for num in numbers:
        try:
            if int(num) % 2 == 0:
                even_numbers.append(num)
        except (ValueError, TypeError):
            continue  # Skip non-numeric entries
    return even_numbers

# Example usage:
print(filter_even_numbers([1, 2, "3", 4, "five", 6]))  # Output: [2, 4, 6]
```

INTERMEDIATE PYTHON PRACTICE

Write a function find_pairs(numbers: list, target: int) -> list that returns unique pairs (as tuples) that add up to the target sum.

Test Case → ([1, 2, 3, 4, 5, 6], 7)

INTERMEDIATE PYTHON PRACTICE

```python
def find_pairs(numbers: list, target: int) -> list:
    seen = set()
    pairs = set()
    for num in numbers:
        complement = target - num
        if complement in seen:
            pairs.add(tuple(sorted((num, complement))))
        seen.add(num)
    return list(pairs)

# Example usage:
print(find_pairs([1, 2, 3, 4, 5, 6], 7))  # Expected Output: [(1, 6), (2, 5), (3, 4)]
```

WHAT IS AN API?

An API (Application Programming Interface) is a set of rules that allows different software systems to communicate.

APIs expose endpoints (functions) that can be called remotely.

They separate the implementation details from the interface used by clients.

REAL WORLD EXAMPLES

- Weather apps
- Social Media apps
- Anything with a backend

- Anything that provides software services
- OpenAI
- Google Cloud

TYPES OF API FETCHES

HTTP Methods:

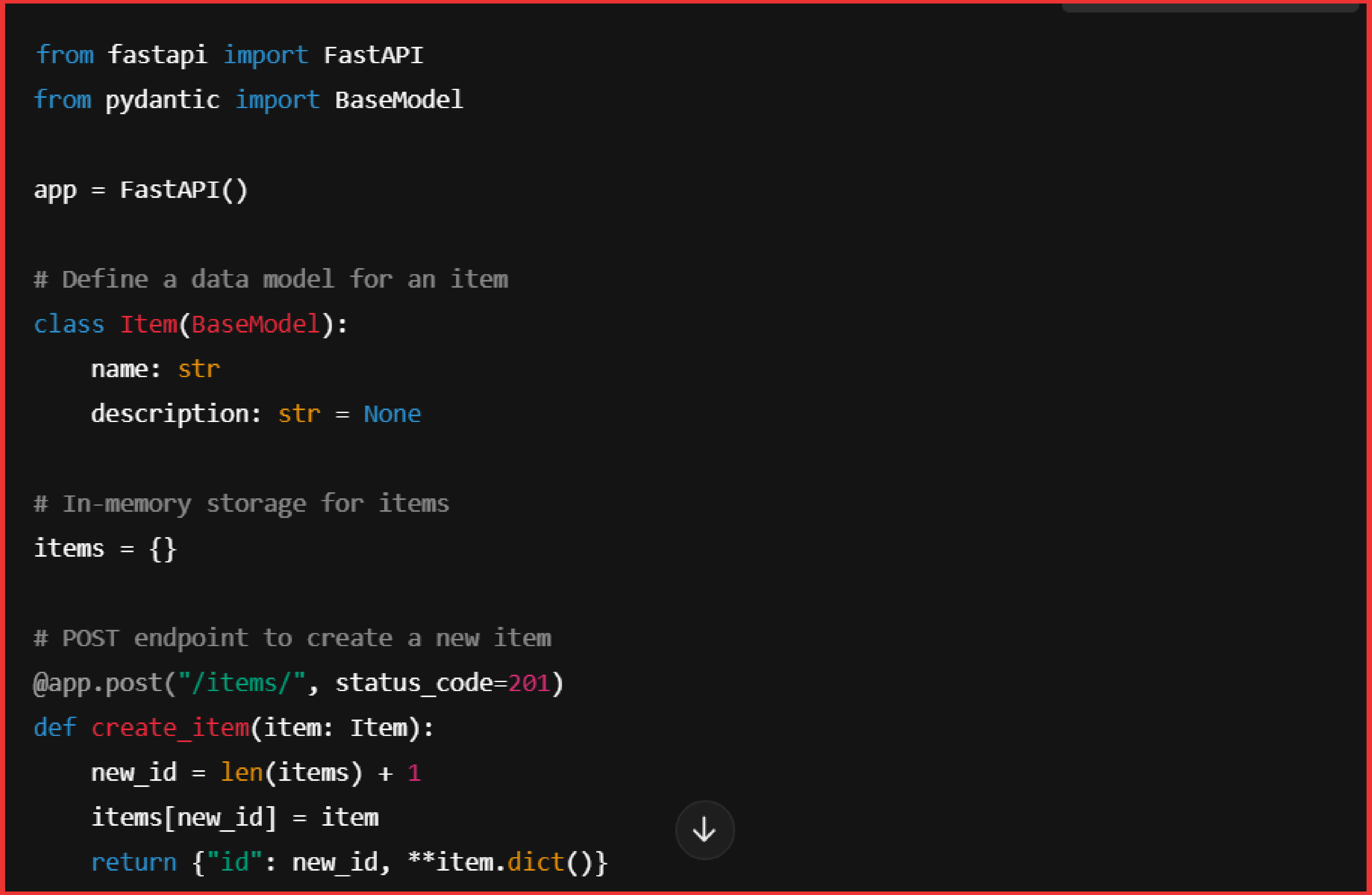
- **GET:**
  - **Retrieve data, e.g., fetching a list of products from an online store.**
- **POST:**
  - **Send data to create a resource, e.g., submitting a new blog post.**
- **PUT/PATCH:**
  - **Update existing data, e.g., modifying a user's profile information.**
- **DELETE:**
  - **Remove data, e.g., deleting an outdated advertisement.**

GET Method Example: Reading an Item

```python
@app.get("/items/{item_id}")
def read_item(item_id: int):
    if item_id in items:
        return items[item_id]
    else:
        return {"error": "Item not found"}
```

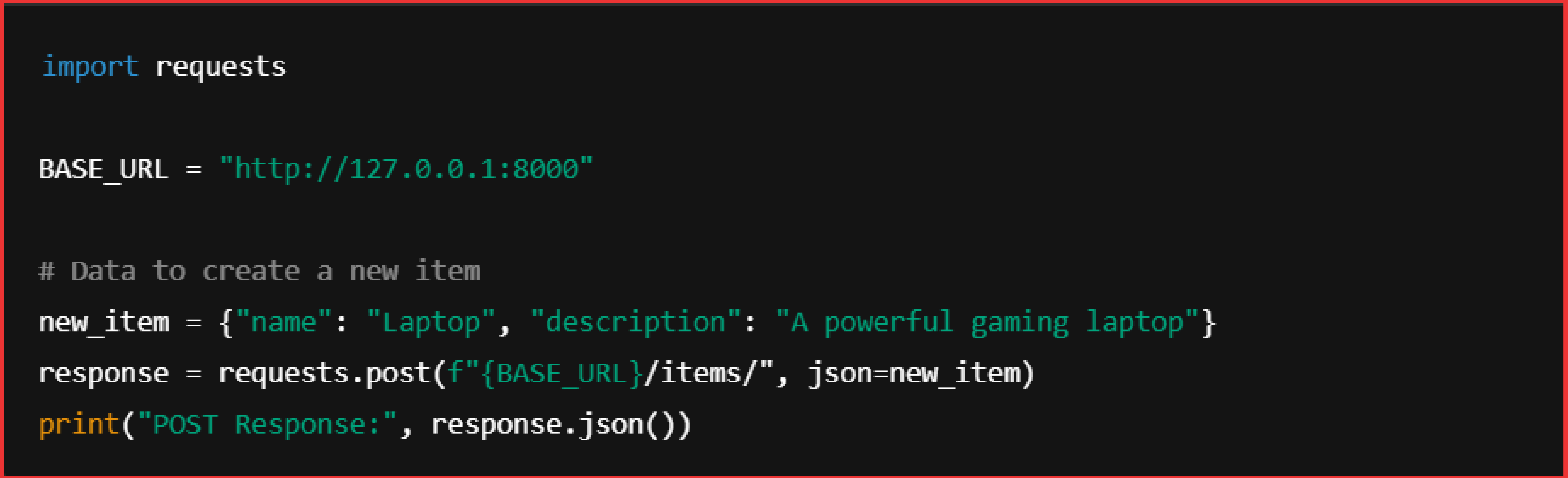
```python
import requests

BASE_URL = "http://127.0.0.1:8000"

response = requests.get(f"{BASE_URL}/items/1")
print("GET Response:", response.json())
```

POST Method Example: Creating an Item



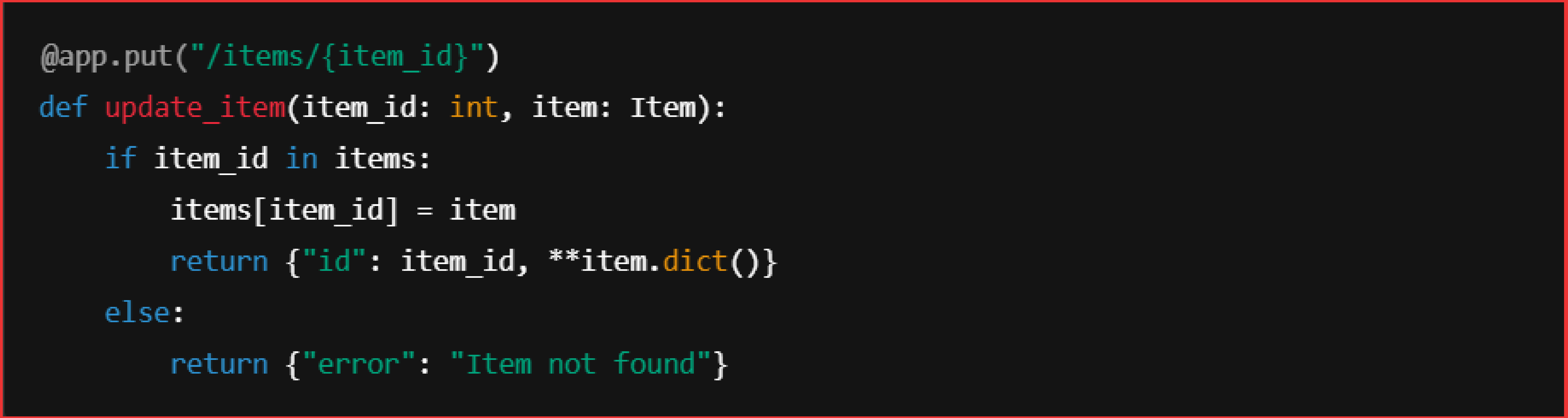
```python
from fastapi import FastAPI
from pydantic import BaseModel

app = FastAPI()

# Define a data model for an item
class Item(BaseModel):
    name: str
    description: str = None

# In-memory storage for items
items = {}

# POST endpoint to create a new item
@app.post("/items/", status_code=201)
def create_item(item: Item):
    new_id = len(items) + 1
    items[new_id] = item
    return {"id": new_id, **item.dict()}
```

POST Method Example: Creating an Item



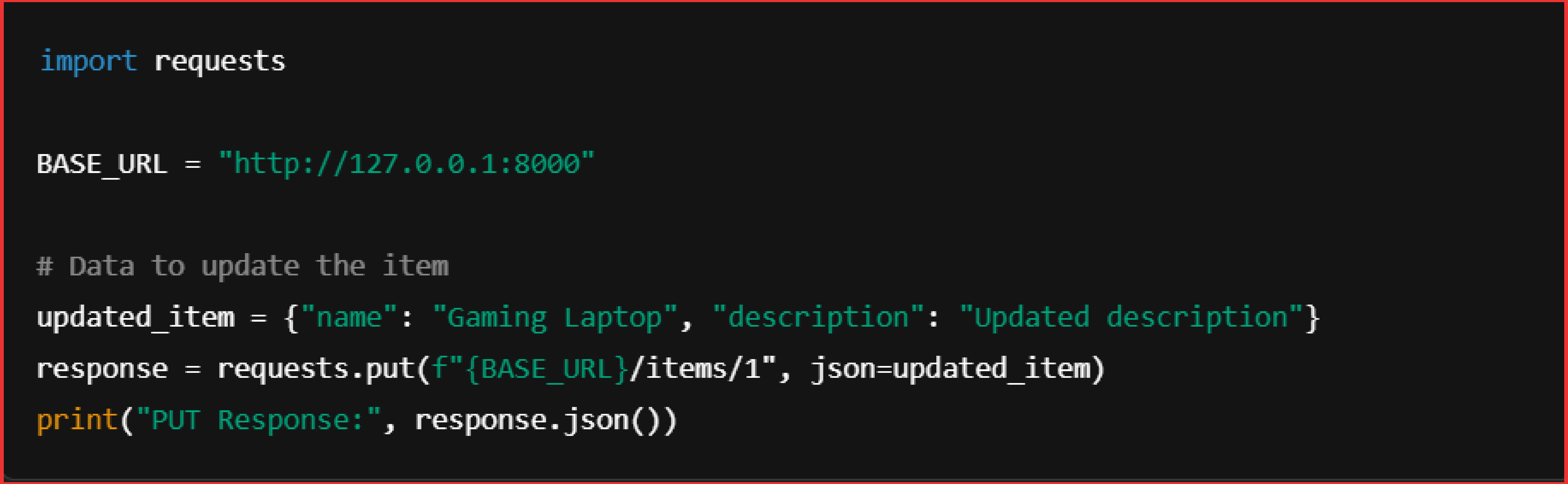
```python
import requests

BASE_URL = "http://127.0.0.1:8000"

# Data to create a new item
new_item = {"name": "Laptop", "description": "A powerful gaming laptop"}
response = requests.post(f"{BASE_URL}/items/", json=new_item)
print("POST Response:", response.json())
```

PUT Method Example: Updating an Item



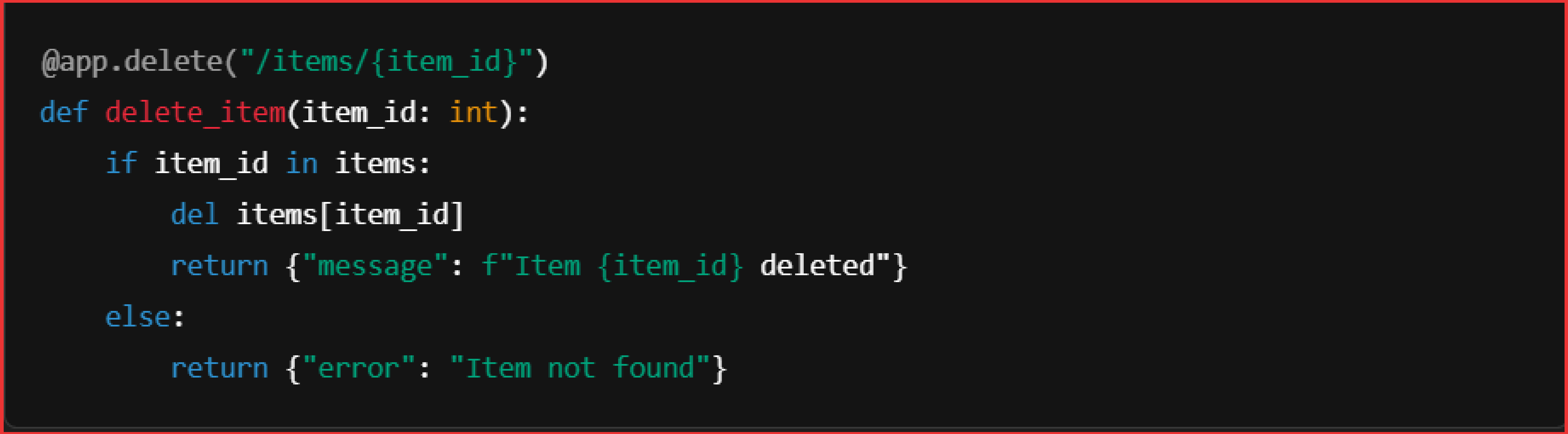
```python
@app.put("/items/{item_id}")
def update_item(item_id: int, item: Item):
    if item_id in items:
        items[item_id] = item
        return {"id": item_id, **item.dict()}
    else:
        return {"error": "Item not found"}
```

PUT Method Example: Updating an Item



```python
import requests

BASE_URL = "http://127.0.0.1:8000"

# Data to update the item
updated_item = {"name": "Gaming Laptop", "description": "Updated description"}
response = requests.put(f"{BASE_URL}/items/1", json=updated_item)
print("PUT Response:", response.json())
```

DELETE Method Example: Deleting an Item



```python
@app.delete("/items/{item_id}")
def delete_item(item_id: int):
    if item_id in items:
        del items[item_id]
        return {"message": f"Item {item_id} deleted"}
    else:
        return {"error": "Item not found"}
```

ADVANCE PYTHON PRACTICE

Given an array arr, answer multiple queries (L, R), returning the sum of elements from index L to R (inclusive).

Test Case →
arr = [1, 2, 3, 4, 5]
queries = [(1, 3), (2, 4), (1, 5)]

ADVANCE PYTHON PRACTICE

```python
def range_sum_naive(arr, queries):
    result = []
    for L, R in queries:
        result.append(sum(arr[L-1:R]))  # Slicing the array for each query
    return result
```

ADVANCE PYTHON PRACTICE

```python
def compute_prefix_sum(arr):
    prefix_sum = [0] * (len(arr) + 1)
    for i in range(1, len(arr) + 1):
        prefix_sum[i] = prefix_sum[i - 1] + arr[i - 1]
    return prefix_sum
```

Data Exchange Formats

- JSON:
  - Standard for modern APIs; human-readable and easily parsed.
  - Real-World Example: Weather data (temperature, humidity) in JSON.
- XML:
  - Used in legacy systems.

Information Exchanged

- Requests:
  - Include URL parameters, query strings, and JSON bodies.
  - Example: A client sending a JSON body with a user's login details.
- Responses:
  - Return JSON objects with keys like message, data, and error.
  - Example: A server returning a list of tasks with their statuses.

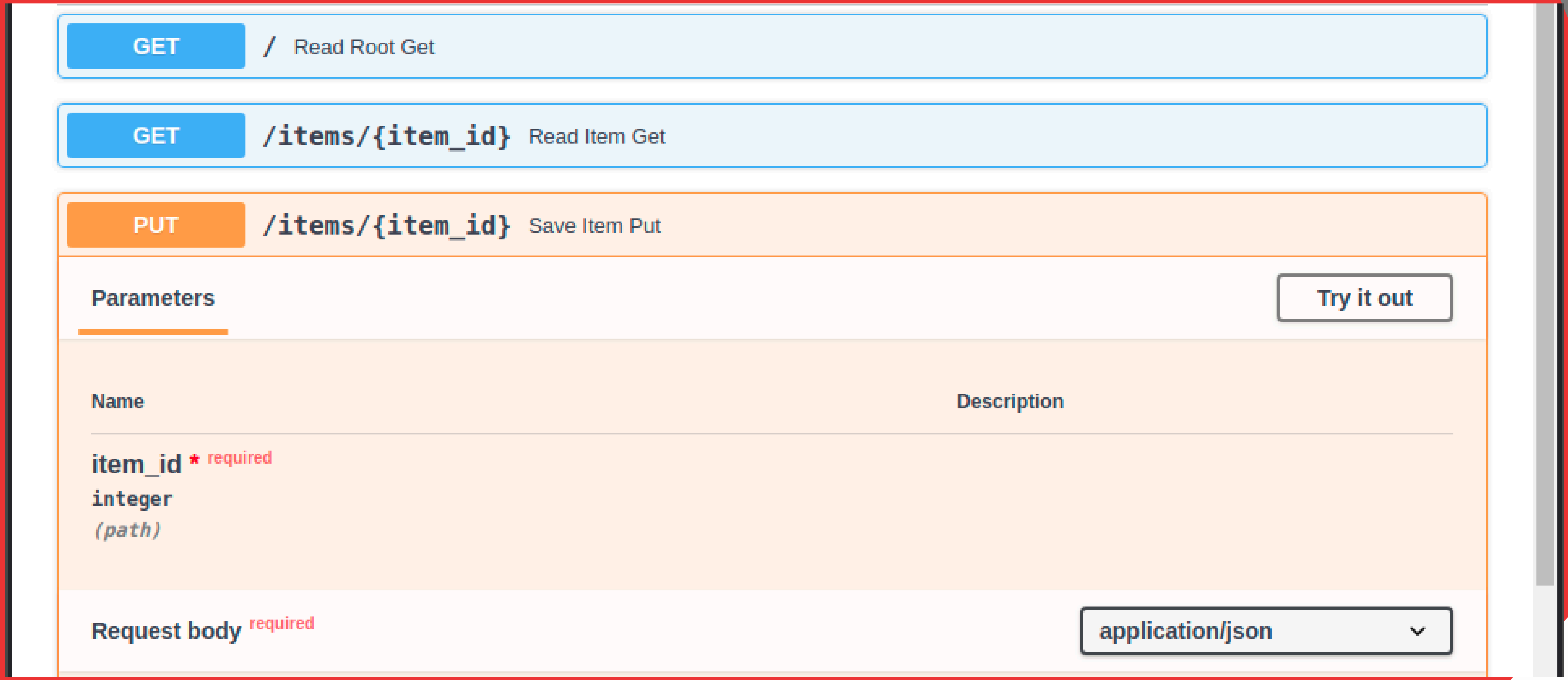
Extra Note:

These methods and formats are standard across industries—from retail APIs to transportation systems.

Setting Up

Installation: pip install fastapi uvicorn

Access interactive API documentation at http://127.0.0.1:8000/docs



GET / Read Root Get

GET /items/{item_id} Read Item Get

PUT /items/{item_id} Save Item Put

Parameters | Try it out

Name	Description
item_id * required integer (path)	

Request body required | application/json

Create an endpoint

- Create a file called main.py.

```
from fastapi import FastAPI
app = FastAPI()
@app.get("/")
def home():
    return {"message": "Welcome to FastAPI!"}
```

Add some functionality

```
tasks = []
@app.post("/tasks/")
def create_task(task: str):
    tasks.append(task)
    return {"message": "Task added!", "tasks": tasks}
@app.get("/tasks/")
def get_tasks():
    return {"tasks": tasks}
```

- Run Server: uvicorn main:app --reload

API Calls from the client

An example of how to interact with our tokenisation endpoint from a client perspective. We're using Python's requests library

```python
def call_tokenise_api(text: str):
    url = f"{BASE_URL}/tokenise/"
    response = requests.post(url, json={"text": text})
    if response.status_code == 200:
        return response.json()
    else:
        return {"error": response.status_code, "message": "Failed to tokenise"}
```

Run in production

- pip install gunicorn
- gunicorn -w 4 -k uvicorn.workers.UvicornWorker main:ap

In production, we want better performance and scalability.

Gunicorn along with Uvicorn workers. Gunicorn is a WSGI server that can handle multiple worker processes, and Uvicorn is our ASGI server for FastAPI.

THE CHALLENGE

EndPoints Provided

Expected Files

- Init.py
- playapi.py

init.py

```
PLAYER_NAME = "Player1"
PLAYER_API_URL = "http://127.0.0.1:8001"  # Adjust if needed
DEALER_API_URL = "http://127.0.0.1:8000"

if __name__ == "__main__":
    logging.info("Starting Player API...")
    uvicorn.run(app, host="127.0.0.1", port=8001)
```

playapi.py

Create an post which returns

- a post with a json with

1. type of turn → bet, fold, show

Your game logic based on -

apis available

Ping API (GET /ping)
Input: No input required.
Output: Returns a simple JSON response with "message": "pong".
Use: Primarily used to check if the server is running ar
responsive. Often used for health checks.

apis available

Show Cards API (GET /show_cards)
Input: Typically requires authentication or player identification (though not explicitly mentioned here).
Output: Returns the cards of the requesting player in a JSON format.
Use: Allows a player to view their current hand of cards during the game.

apis available

Show Pot API (GET /show_pot)
Input: No input required.
Output: Returns the current total pot amount in JSON format.
Use: Provides visibility into the total bet amount collected in the ongoing game round. Helpful for playe to make betting decisions.

Database info

Models.py Shared