102 Workshop: Web API Development using Python (Day 1)

Presented by:
Nobel Wong
TM Technology Sdn Bhd

< / > TM Technology
Software | Cloud | Automation

Content

Background

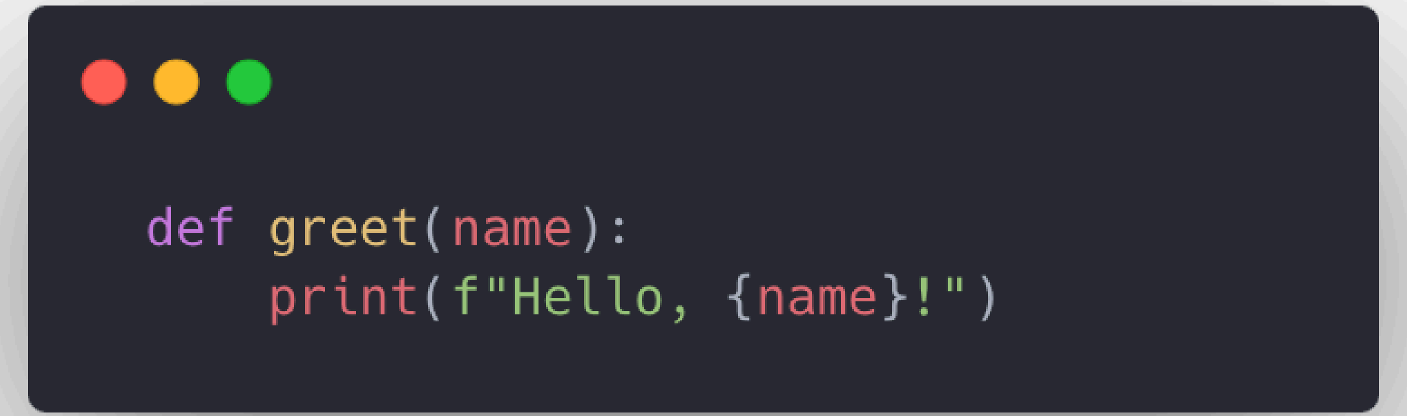
- Technical Director and Software Engineer for TM Technology Sdn Bhd.
- Bachelor and Masters of Engineering from University of Melbourne.
- 8+ years experience programming
  - Data processing
  - Cloud architecture & computing
  - Web & Mobile app development
  - DevOps
- US & AUS software patent co-author

Welcome & Setup

- Theory and Practical workshop about the building a Web API using Python. FastAPI framework.
- By the end of this workshop, we will understand the basic of Web APIs and have a very simple python task manager API running locally.
- VSCode - Free, cross-platform code editor by Microsoft.
- Python 3.13.x - Download from Python Org
- PostMan - Free, cross-platform API platform

Functions

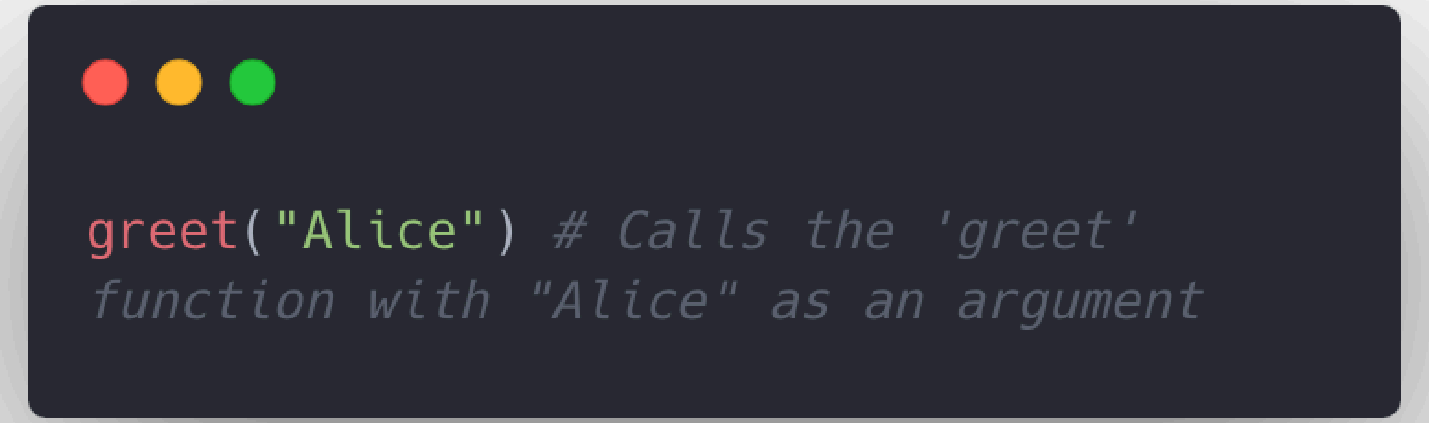
- Python functions are named, reusable blocks of code designed to perform specific tasks.
- They are fundamental to organizing code, promoting modularity, and enhancing readability and maintainability.
- Definition: Functions are defined using the def keyword, followed by the function name, parentheses (which may contain parameters), and a colon. The function's body is indented.



```python
def greet(name):
    print(f"Hello, {name}!")
```

Functions

- Calling: Functions are executed by calling them by their name followed by parentheses, potentially including arguments if the function has parameters.
- Parameters and Arguments:
  - Parameters are placeholders defined in the function signature, receiving input values.
  - Arguments are the actual values passed to the function when it is called.

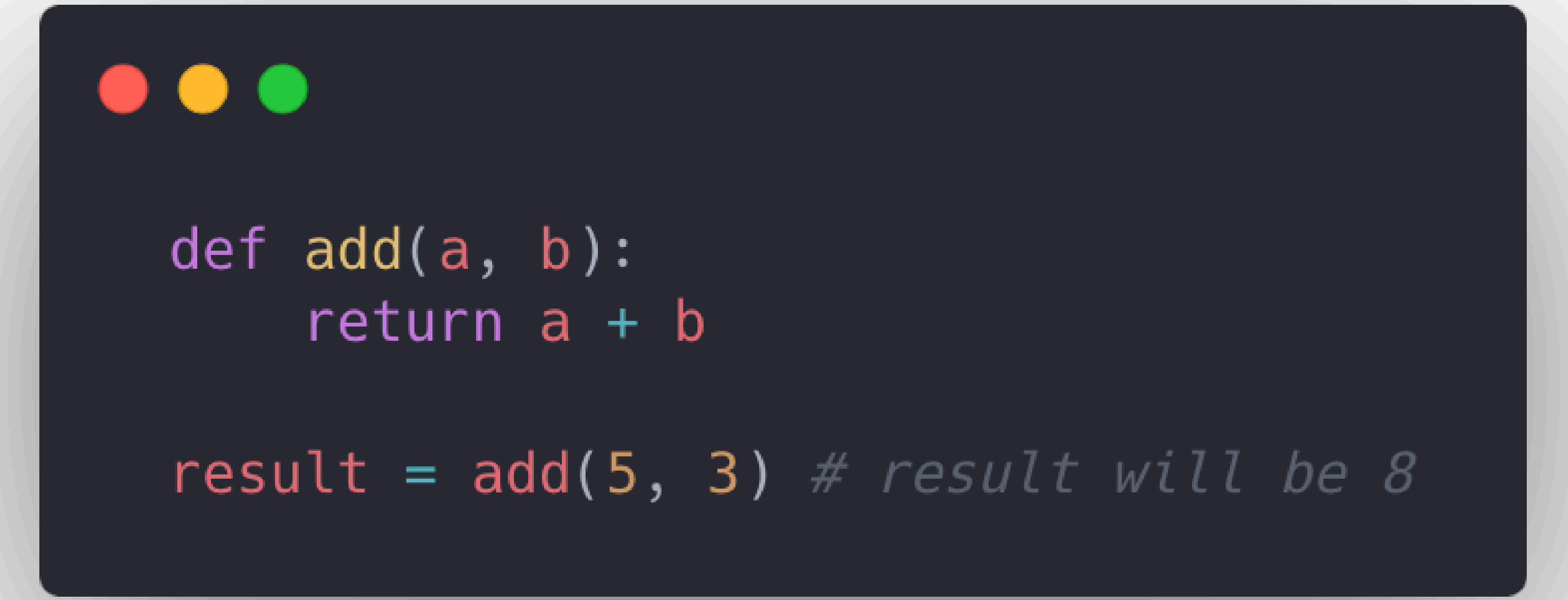


```
greet("Alice") # Calls the 'greet'
function with "Alice" as an argument
```

</> TM Technology

Function Return

- Functions can optionally return a value using the return keyword. If no return statement is present, the function implicitly returns None.



```python
def add(a, b):
    return a + b

result = add(5, 3) # result will be 8
```

</> TM Technology

Functions

- **Types of Functions:**
  - **Built-in Functions**: Provided by Python's standard library (e.g., print(), len(), sum()).
  - **User-defined Functions**: Created by the programmer to perform custom tasks.
  - **Functions from Modules**: Functions defined within modules that need to be imported before use (e.g., math.sqrt() after import math).

Functions

- **Benefits:**
  - Code Reusability: Avoids repeating the same code block multiple times.
  - Modularity: Breaks down complex programs into smaller, manageable units.
  - Readability: Well-named functions make code easier to understand.
  - Maintainability: Changes to a specific task can be isolated within a function.

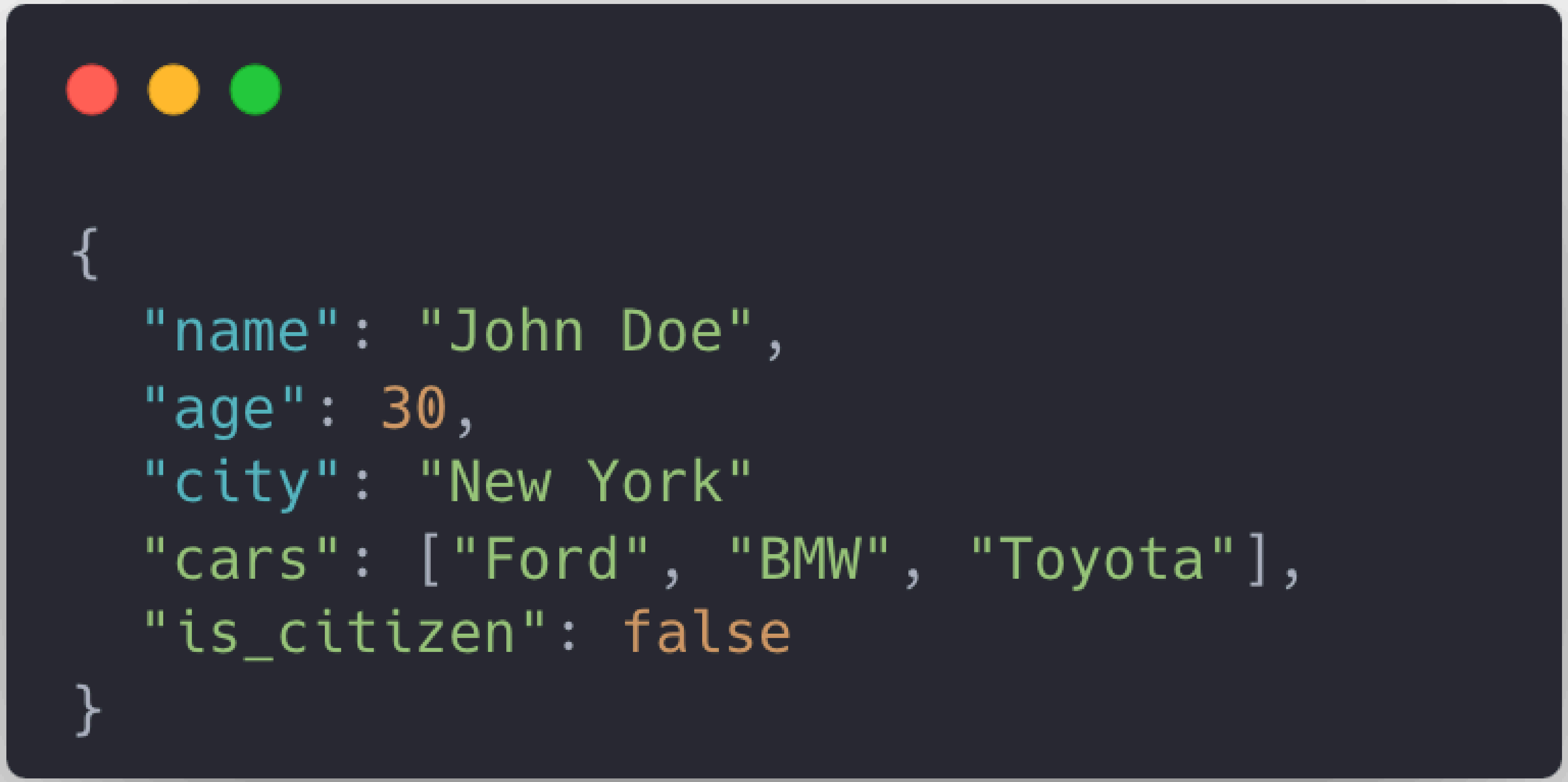
Task

- Create a function that find the square of a given number
- Create a function that find the maximum of three numbers
- Create a function that sum all numbers in a list
- Create a function that check if a number falls within a given range

JSON

- JSON (JavaScript Object Notation) is a lightweight format for storing and transporting data, often used for transmitting dta between web applications and servers.
- Text-based, human-readable format that is easy for both humans and machines to parse and generate.
- It can be used with any programming language.
- Can be stored as a variable or file.
- JSON data is typically structured as either an object (a collection of key-value pairs) or an array (an ordered list of values).

JSON



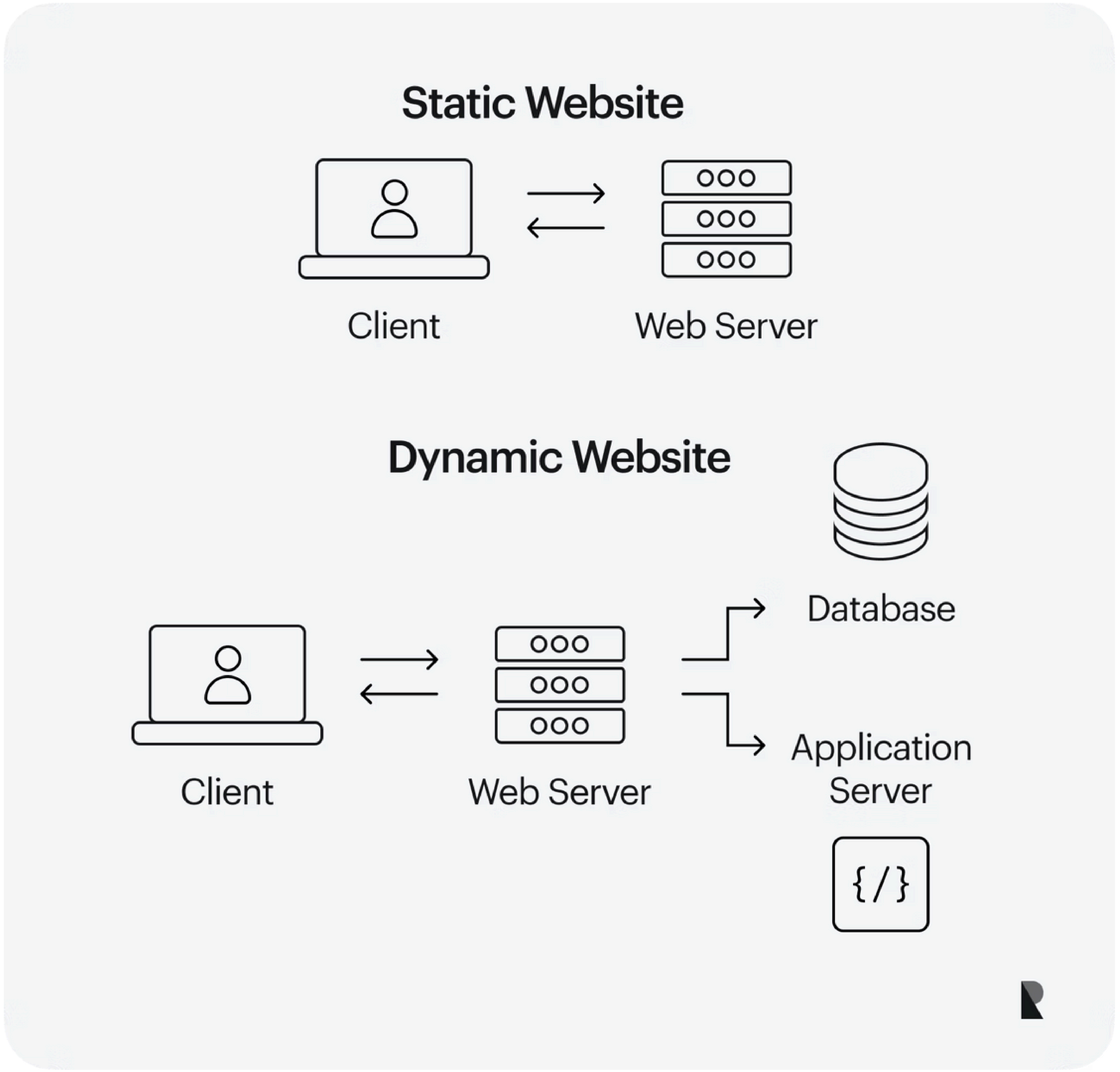
```
{
  "name": "John Doe",
  "age": 30,
  "city": "New York"
  "cars": ["Ford", "BMW", "Toyota"],
  "is_citizen": false
}
```

</> TM Technology

Web Server

- A web server is a compute system that hosts websites and delivers web content to users over the internet.
- It processes requests from clients and sends back the requested web pages, along with associated files like images and scripts.
- Back-bone of the internet, making websites accessible to users worldwide.
- Single web server can host multiple websites.
- Requires hardware (specialized or general purpose) and Software (Apache, Nginx, FastAPI)

Web Server



</> TM Technology

APIs

- API (Application Programming Interface) is a set of rules and protocols that allows different software applications to communicate and interact with each other.
- APIs act as intermediaries, enabling applications to interact without needing to understand each other's internal workings.
- Various purposes
    - Accessing data, integrating services, and extending functionality.

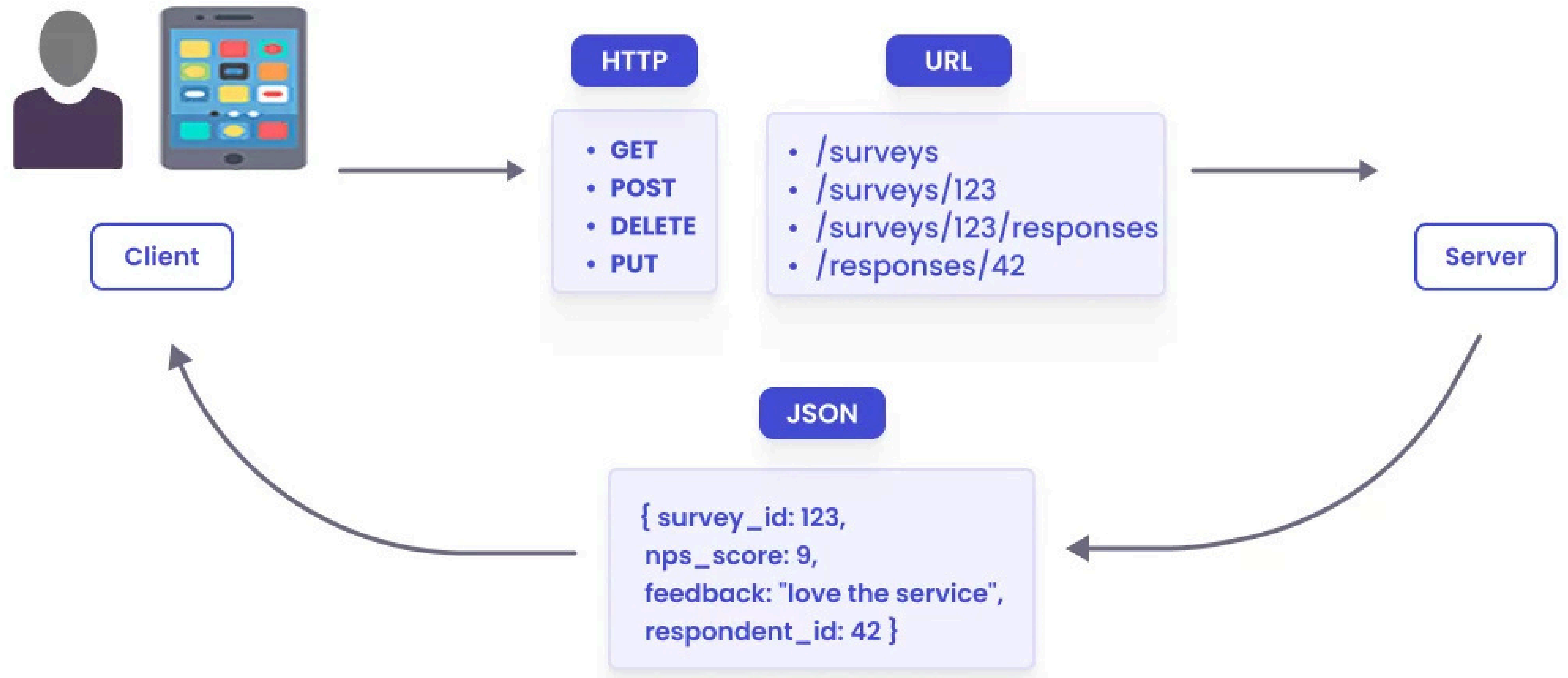
HTTP

- HTTP (Hypertext Text Transfer Protocol) is an application-layer protocol used for transmitting hypermedia documents, such as HTML.
- Foundation of data communication for the World Wide Web, enabling communication between web browsers and web servers.
- When a user requests a webpage, the browser sends an HTTP request to the server, and the server responds with the requetsed data using HTTP.

API and HTTP

- HTTP is a common protocol used for implementing APIs, known as HTTP API.
- API defines the specific endpoints (URLs) and the format of HTTP requests and responses that applications should use to intereact with the service.
- API is a broader concept of enabling software communication, HTTP is a specific protocol frequently used to facilitate that communication.

API and HTTP



</> TM Technology

API Demo

- https://jsonplaceholder.typicode.com/posts
- https://api.ipify.org?format=json
- https://ipinfo.io/{ip_address}/geo
- http://universities.hipolabs.com/search?country=United+States
- https://www.alphavantage.co/query?function=MARKET_STATUS&apikey=demo

Test using postman

Python FastAPI

- FastAPI is a modern, high-performance web framework for building APIs with Python based on standard Python type hints.
- It is designed for speed, ease of use, and automatic documentation generation.
- Good documentation and lots of online resources to build FastAPI projects.
- Alternatives are Flask and Django, both are Python API Frameworks but we will be using FastAPI in this workshop.
- Docs at https://fastapi.tiangolo.com/

Setting up FastAPI

- Check python -V in terminal to make sure Python is install correctly.
- Install FastAPI using pip install “fastapi[standard]”
- Follow https://fastapi.tiangolo.com/ to setup the API server
  - Create main.py file
  - Go to the example section and copy the main.py file content
- Run the FastAPI server using “fastapi dev main.py” with terminal in the project root directory.
- Open your browser at http://127.0.0.1:8000/items/5?q=somequery
- Now go to http://127.0.01:8000/docs
- Open PostMan and make a GET request to http://127.0.0.1:8000

Task

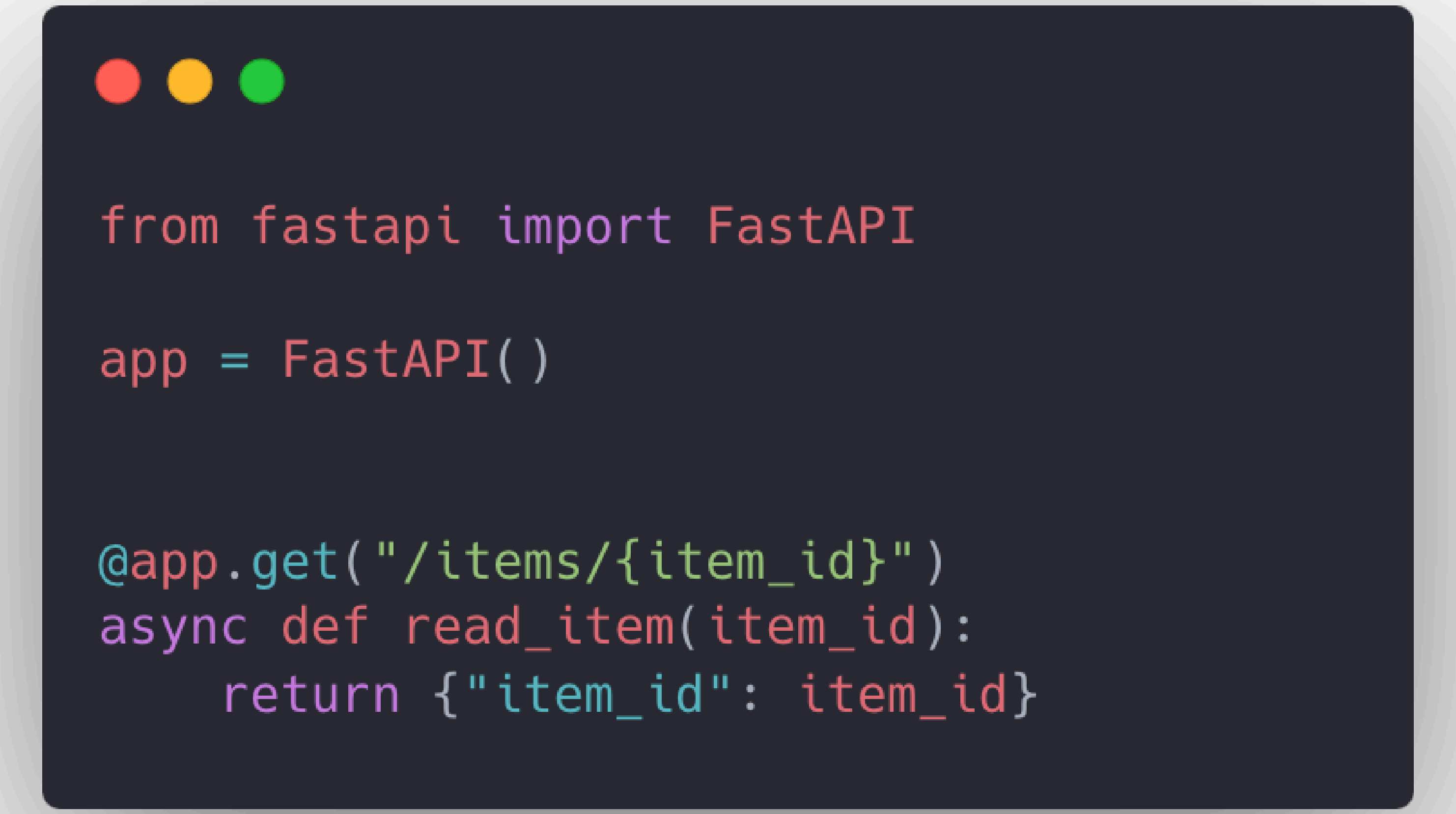
- Add GET routes at /hello and /about
  - Make them return some message
- Visit localhost:8000/docs to view api documentation and verify new routes has been added
- Make GET request to /hello and /about using PostMan

API Endpoints

- An API endpoint is a specific network location, typically a Uniform Resource Locator (URL), where an API receives requests and sends responses.
- It represents a distinct resource or function that a client application can interact with.
- https://www.instagram.com/innovationlab.bn/p/DM7WmFDyWw1/
- The “paths” is the part of the URI after the domain name (e.g., /users)

Path Parameters

- You can declare path “parameters” or “variables” with the same syntax used by Python format strings:
- The value of the path parameter item_id will be passed to your function as the argument item_id.



```python
from fastapi import FastAPI

app = FastAPI()


@app.get("/items/{item_id}")
async def read_item(item_id):
    return {"item_id": item_id}
```

</> TM Technology

HTTP Methods

- API routing, in its most basic form, defines how an application responds to client requests at specific network endpoints.
- Typically identified by a Uniform Resource Identifier (URI) or path and an HTTP method
  - GET, POST, PUT, DELETE
- GET retrieves data, POST creates new data, PUT updates existing data, and DELETE removes data.
- A route combines a specific path and an HTTP method.
- When a client sends a request matching a defined route, the associated server-side function or handler is executed.

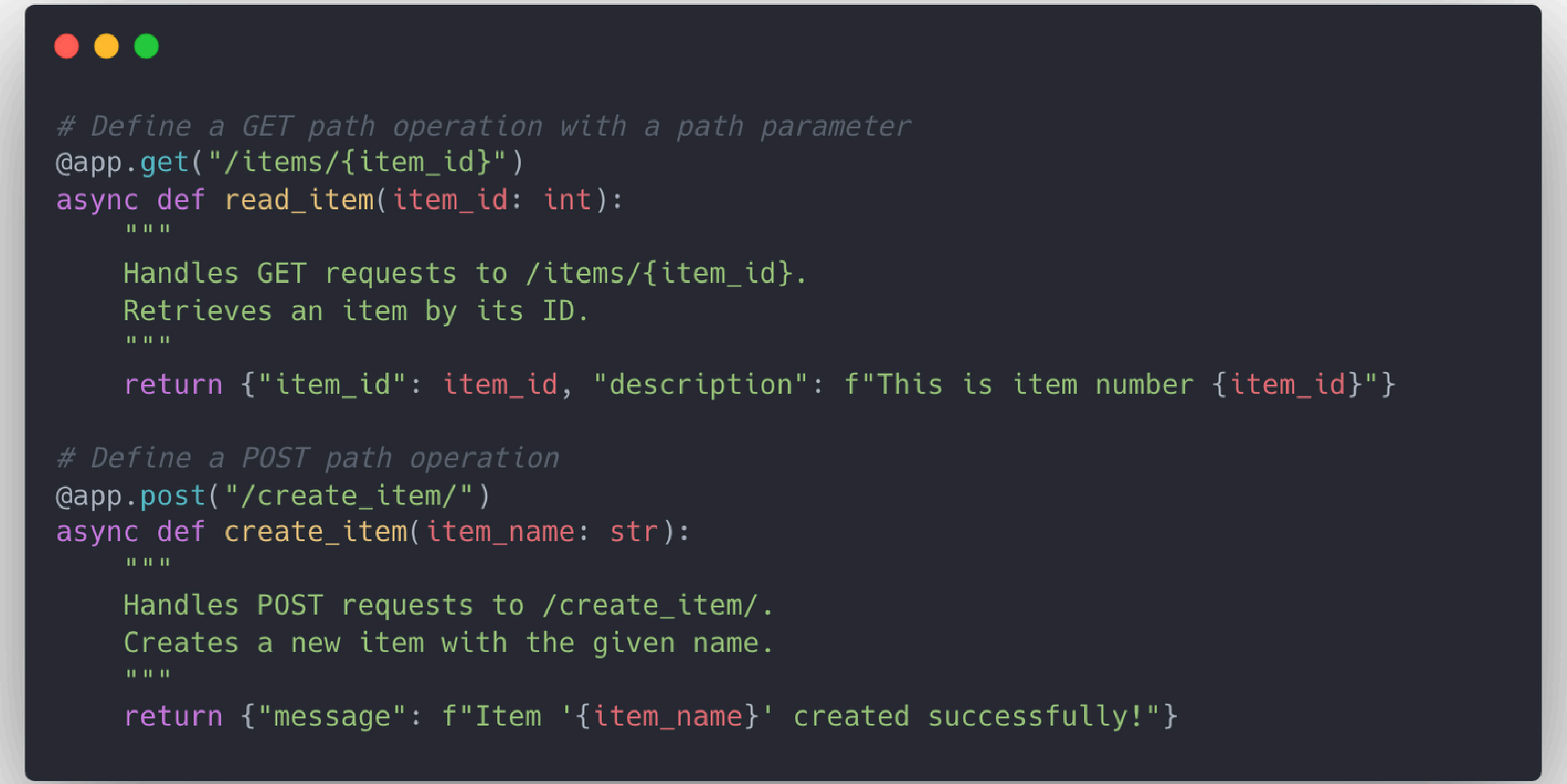
HTTP Status Code

- HTTP status codes are three-digit numerical codes returned by a server in response to an HTTP request made by a client.
- These code indicate the outcome of the request and provide information about whether the request was successful, redirected, or encountered an error.
- 1xx (Informational): 100 Continue
- 2xx (Success): 200 OK, 201 Created
- 3xx (Redirection): 301 Moved Permanently, 302 Found
- 4xx (Client Error): 400 Bad Request, 401 Unauthorised, 404 Not Found
- 5xx (Server Error): 500 Internal Server Error

Task

- Mini task manager API with hardcoded data (15-30 mins)
- Use a Python list/dictionary to store and access items data.
- Add a GET endpoint at “/tasks/{task_id}”
- Add a POST endpoint at “/task/” - create_task
- Add a PUT endpoint at “/task/{task_id}” - update_task
- Add a DELETE endpoint at “/task/{task_id}” - delete_task

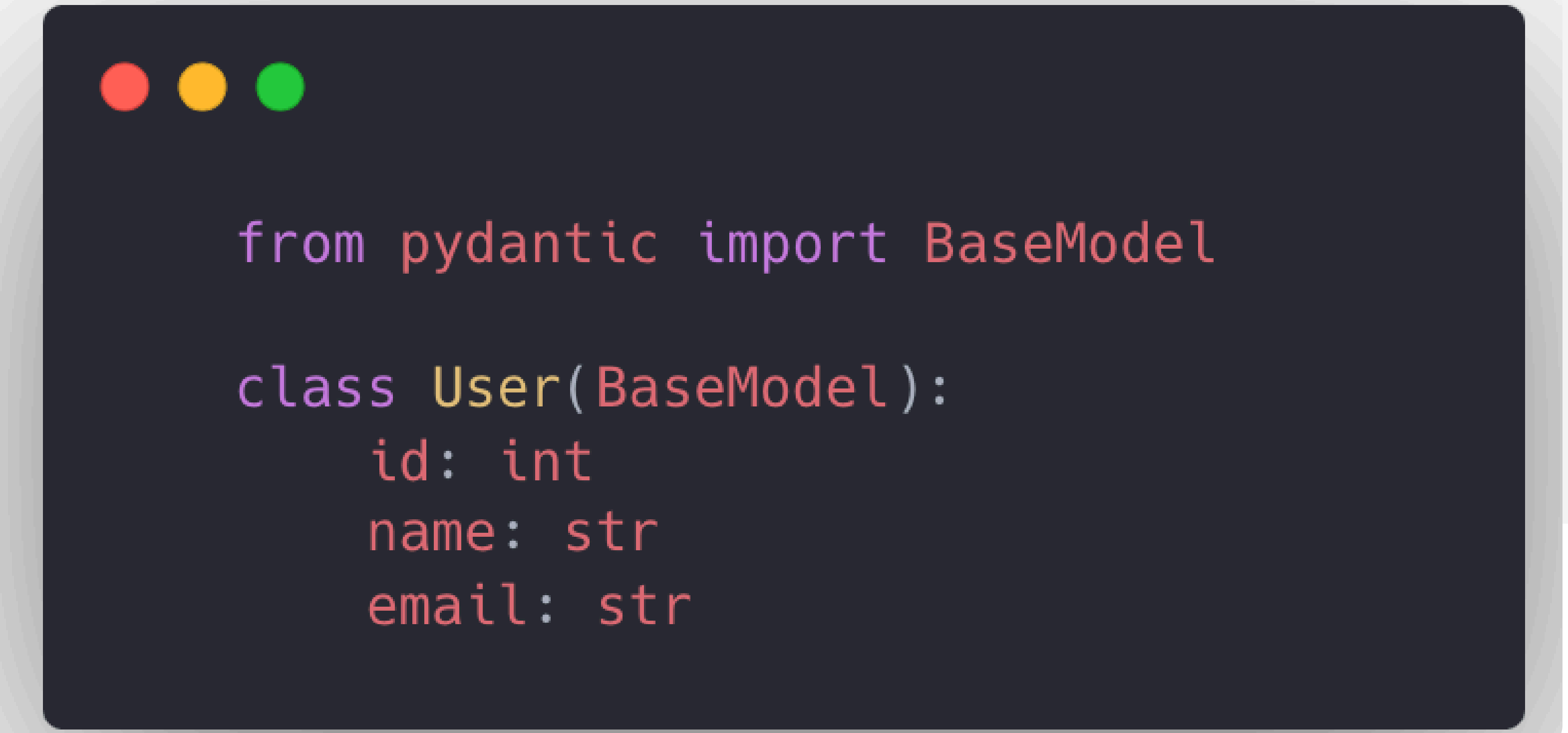
Task



```python
# Define a GET path operation with a path parameter
@app.get("/items/{item_id}")
async def read_item(item_id: int):
    """
    Handles GET requests to /items/{item_id}.
    Retrieves an item by its ID.
    """
    return {"item_id": item_id, "description": f"This is item number {item_id}"}

# Define a POST path operation
@app.post("/create_item/")
async def create_item(item_name: str):
    """
    Handles POST requests to /create_item/.
    Creates a new item with the given name.
    """
    return {"message": f"Item '{item_name}' created successfully!"}
```

Data Models with Pydantic

- Pydantic is a Python library used for data validation and parsing, primarily for defining data models using Python's type annotations.
- It enforces data integrity and simplifies the creation of data models with automatic type checking and validation.
- BaseModel: The foundation for creating Pydantic models. Classes that inherit from BaseModel automatically gain data validation, parsing, and serialization capabilities.



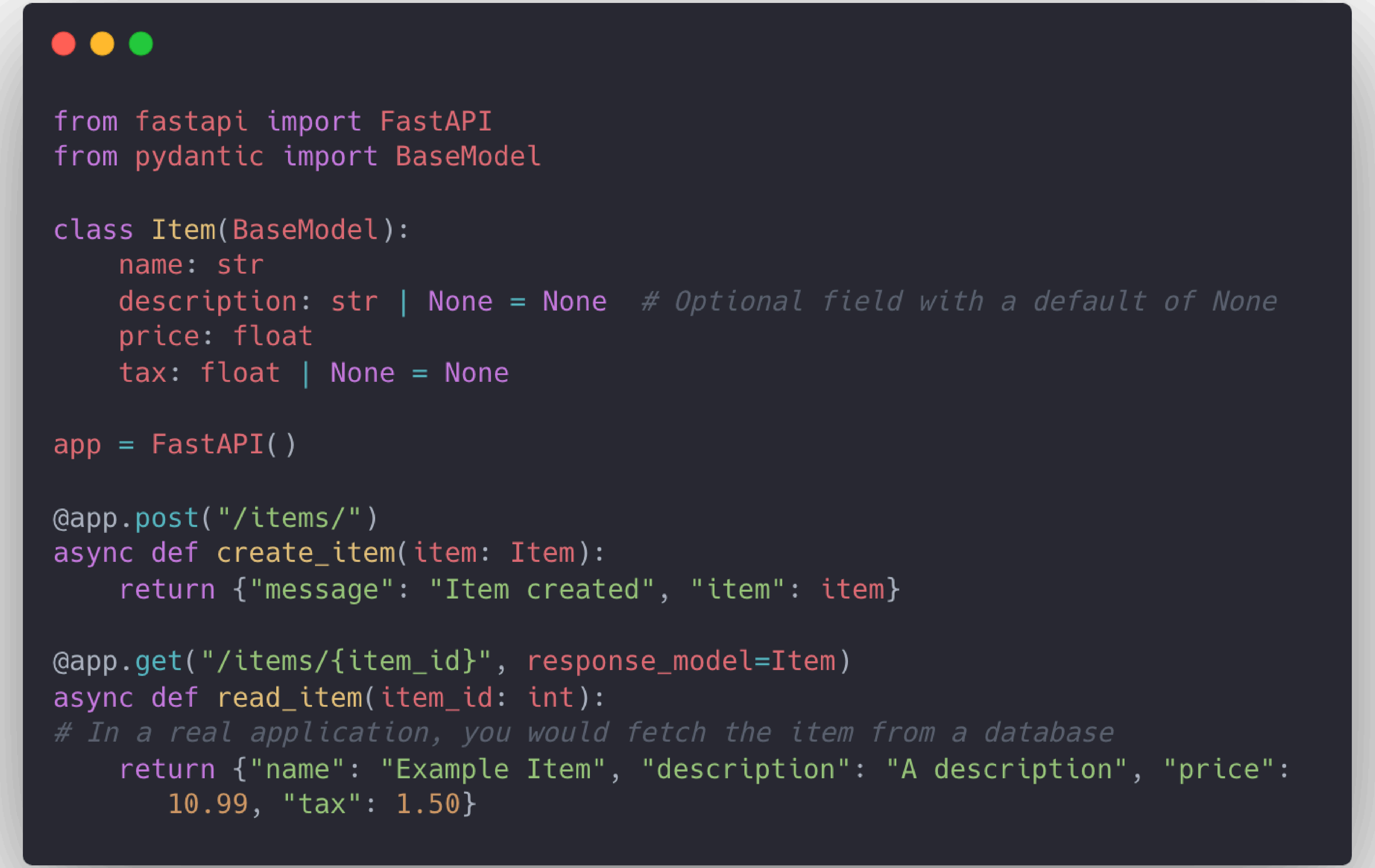
```python
from pydantic import BaseModel

class User(BaseModel):
    id: int
    name: str
    email: str
```

</> TM Technology

Data Models with Pydantic

- Using data models with API endpoints is essential for ensuring consistency, reliability, and maintainability in API design and development.
- The reasons for this necessity include:
  - Defining Data Structure and Format
  - Enabling Validation and Error Handling
  - Facilitating Code Generation and SDKs
  - Improving Documentation and Discoverability
  - Promoting Consistency and Standardization
  - Simplifying Data Transformation and Mapping

Data Models with Pydantic



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

class Item(BaseModel):
    name: str
    description: str | None = None  # Optional field with a default of None
    price: float
    tax: float | None = None

app = FastAPI()

@app.post("/items/")
async def create_item(item: Item):
    return {"message": "Item created", "item": item}

@app.get("/items/{item_id}", response_model=Item)
async def read_item(item_id: int):
# In a real application, you would fetch the item from a database
    return {"name": "Example Item", "description": "A description", "price":
       10.99, "tax": 1.50}
```

</> TM Technology

Task

- Add Pydantic model for items and validate input
- Items should have an id, name and description, extend using BaseModel
    - id should be integer
    - Name should be string
    - Description should be string

Wrap-Up

- Functions
- JSON data structure
- Webserver
- API and API endpoints
- Path Parameters
- HTTP methods and status code
- FastAPI
- Data Models for Input Validation
- Pydantic and FastAPI

Q&A Session

Thank you

Presented by:
Nobel Wong
TM Technology Sdn Bhd

</> TM Technology