#3

Data Platform Development

FastAPI & Pydantic

"Konsumera, Ta emot och skicka data med FastAPI."

Table of Contents


Overview

Moduler:

- Installation
- FastAPI & unik syntax - basics
- Pydantic & Datastrukturer
- Postman
- Exempel: Konsumering av randomFox API

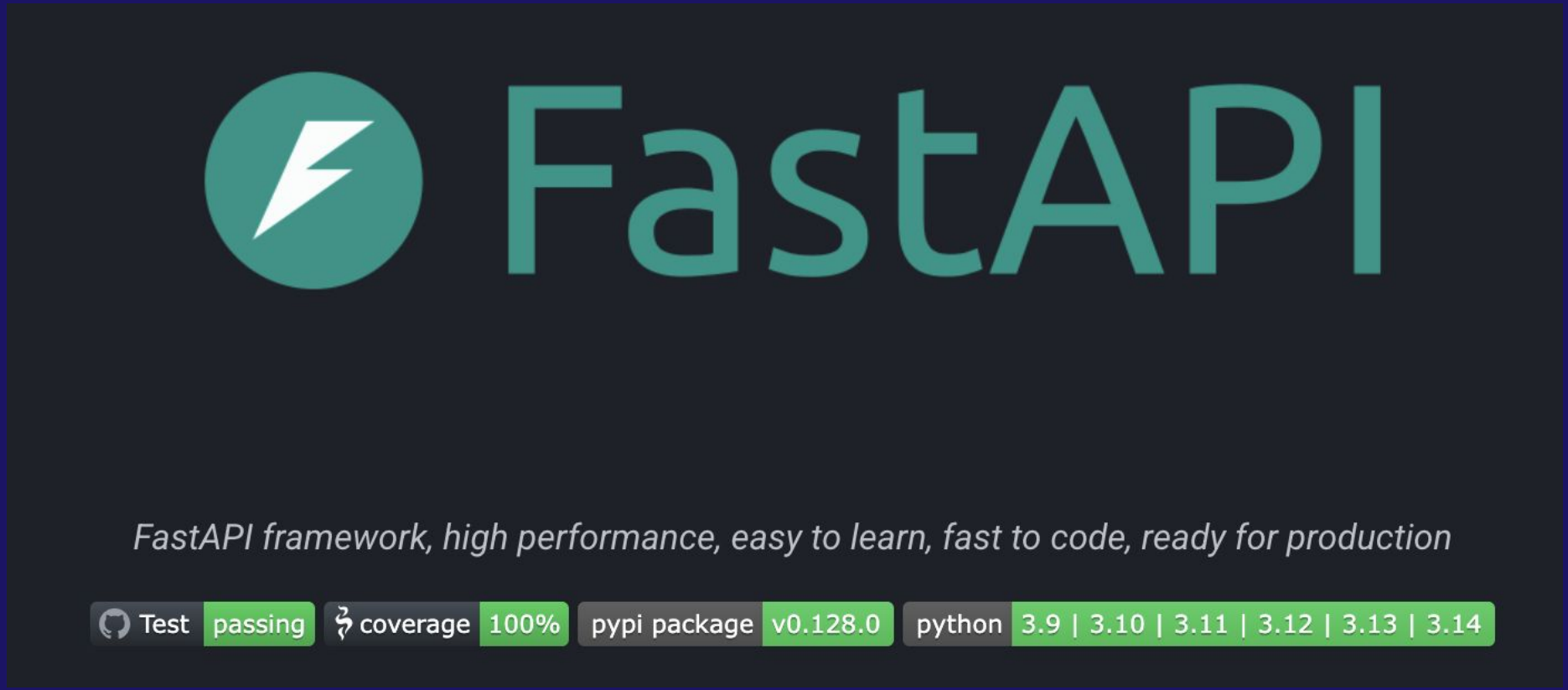
Utbildningsmoment

- Dataplatfformar, bakgrund och syfte
- Git och github i teamkontext
- Komponenter och teknologier i en data platform ✅
- ETL vs ELT
- Utveckling av mjukvara mot databaser ✅
- Använda Python mot relationsdatabaser och andra datakällor såsom csv, http xml/json
- Använda Python mot realtidsdataströmmar såsom message queues och/eller event streaming platforms
- Använda Python och för att rensa, validera och transformera data
- Workflow processer ✅

02

Project Setup, Dependency Installation & FastAPI


FastAPI
Open System
Interconnection



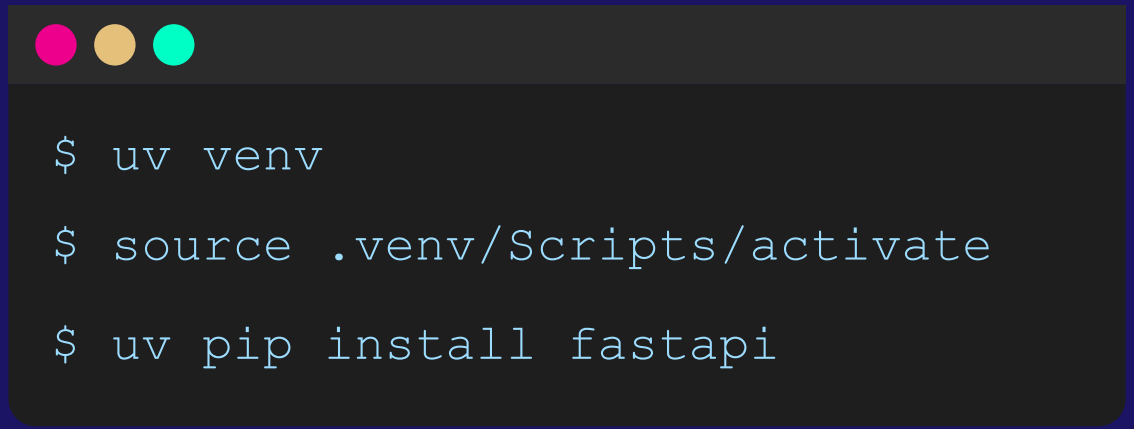
Source: https://fastapi.tiangolo.com/

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

The key features are:

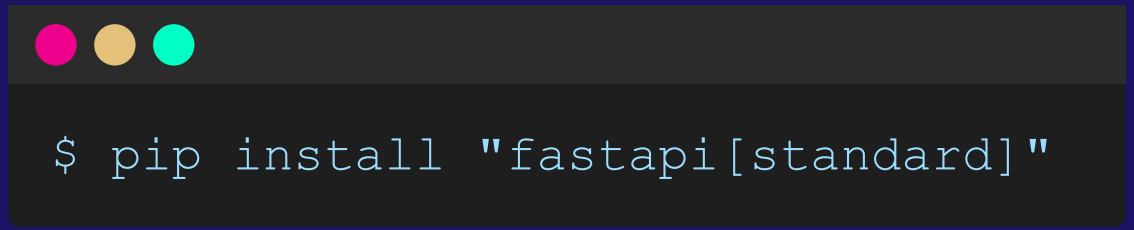
- **Fast**: Very high performance, on par with **NodeJS** and **Go** (thanks to Starlette and Pydantic). One of the fastest Python frameworks available.
- **Fast to code**: Increase the speed to develop features by about 200% to 300%. *
- **Fewer bugs**: Reduce about 40% of human (developer) induced errors. *
- **Intuitive**: Great editor support. Completion everywhere. Less time debugging.
- **Easy**: Designed to be easy to use and learn. Less time reading docs.
- **Short**: Minimize code duplication. Multiple features from each parameter declaration. Fewer bugs.
- **Robust**: Get production-ready code. With automatic interactive documentation.

Install FastAPI
(UV)



If you run using ‘uv venv’ do this ^

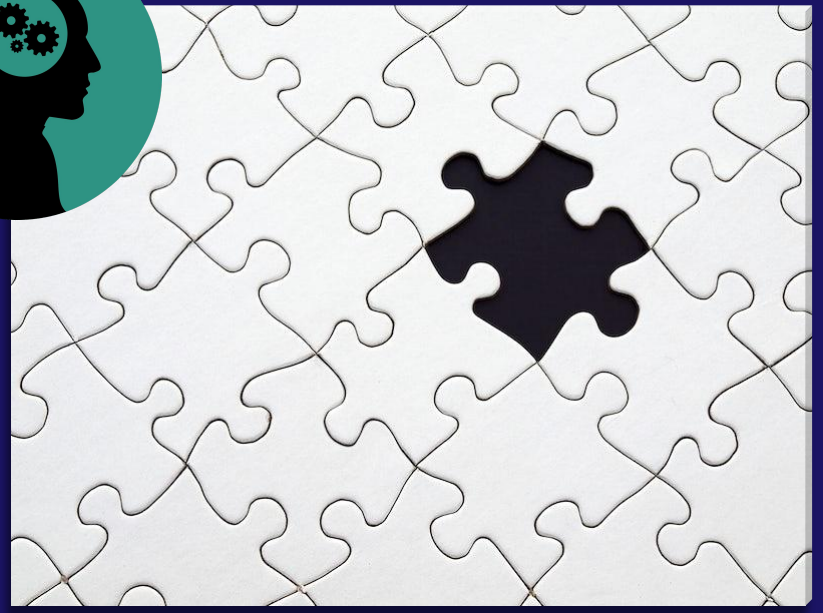
Install FastAPI (Terminal)



Install fastAPI

[standard] includes:
(Uvicorn, pydantic, python-multipart)
This also ensures that it works on all terminals!

Frågor?




FastAPI
Setup

FastAPI
(app & title)

```python
from typing import Union

from fastapi import FastAPI

app = FastAPI(title="My First API")
```

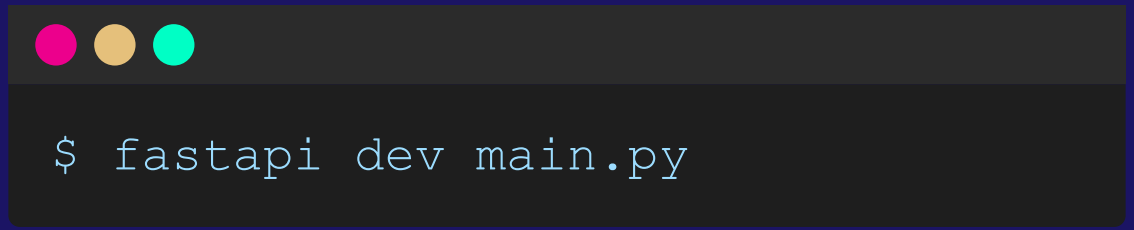
This ensures our app can be started from the Terminal, with the name of “My First API”

FastAPI
(HTTP-GET endpoint)

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

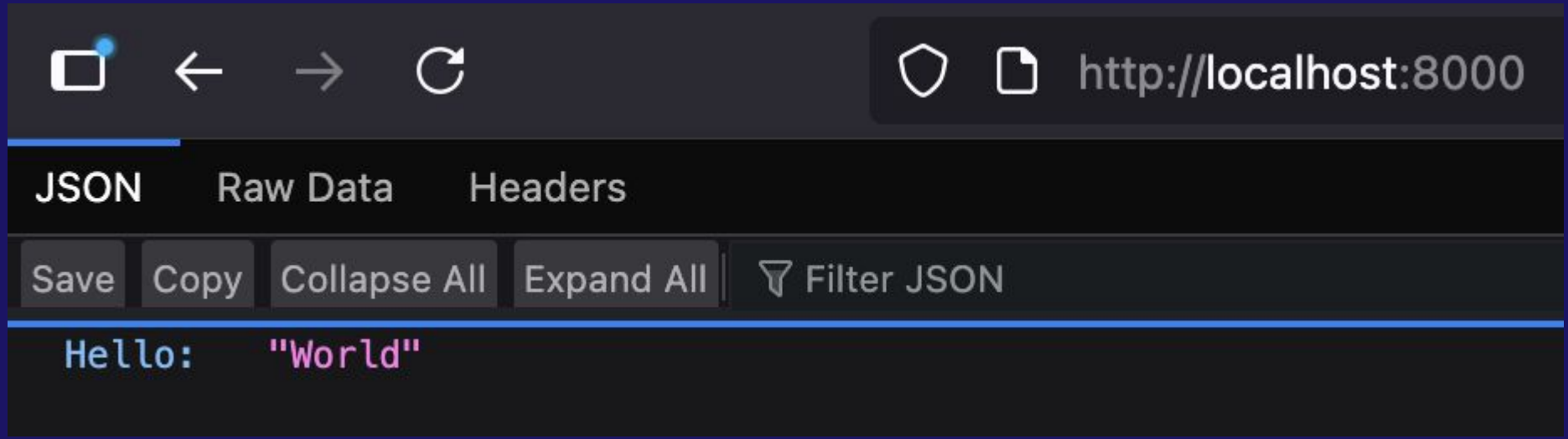
Create a path '/' that when visited, displays JSON format translated from Python Dictionary data.
'Root' is just a name.

FastAPI
(Run the app)



You can then visit localhost:8000

FastAPI
(Running the app - JSON result)




FastAPI
Queries

FastAPI
(Query Theory)

```
Let's say you want to accept dynamic data:
Product price = 5

In a URL that would be
localhost:8000/product?price=500
@app.get("/product/{product_price}")
```

`/product/{}` <-- this is called a **Query Parameter** where the name inside becomes the **Variable Name**

FastAPI
(Query Parameter)

```python
@app.get("/items/{item_id}")
def get_item(item_id: int, query: Union[str, None] = None):
    return {"item_id": item_id, "query": query}
```

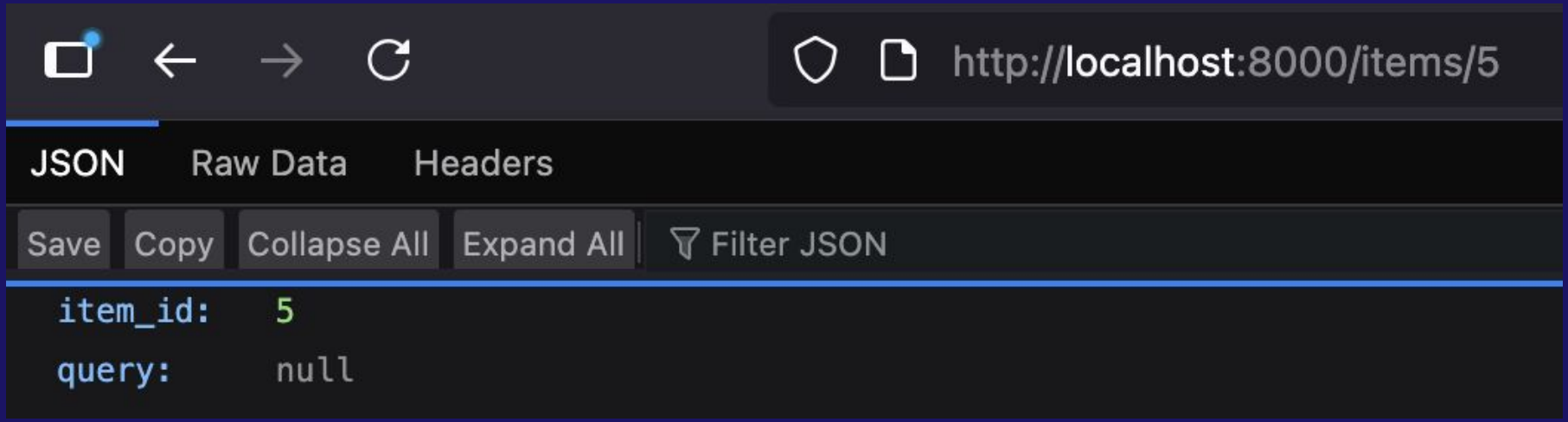
This ensures that we can write: `localhost:8000/items/5?query=bananas`
In other words... we now simulate that: `item with ID 5 = “bananas”`

FastAPI (bonus)
(Why Unions?)

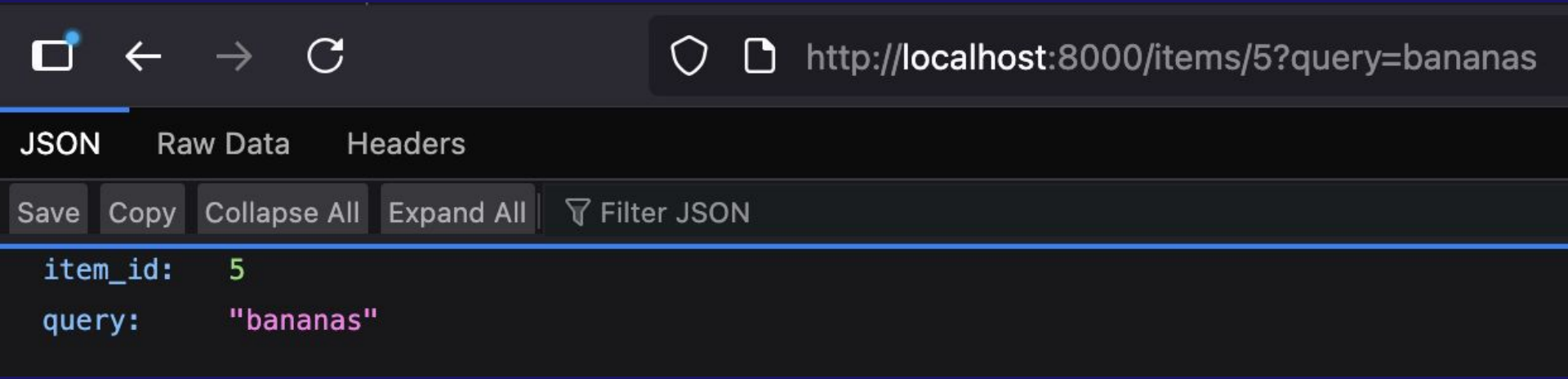
```python
@app.get("/items/{item_id}")
def get_item(item_id: int, query: Union[str, None] = None):
    return {"item_id": item_id, "query": query}
```

There are cases where users don't need to insert values. Therefore we have an Optional value.
Either it's a String or Nothing

FastAPI
(Why Unions - Result)



FastAPI
(Why Unions - Result #2)




The problem with
Dictionaries

Problems with Dictionaries
(Explained)

- Otypat
- Otydlig
- Implicit
- Ovaliderat
- Felbenäget

Struktur == viktigt

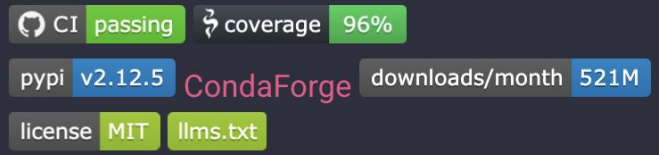
Problems with Dictionaries (Case)

```
product = {
    "name": "Apple",
    "price": "22",
    "inStock": True
}
```

Explanation: This works.. But what if the data was invalid or wrong datatype?
This bug would survive in silence through runtime and cause problems in production. Debugging will most likely be time consuming.

FastAPI includes Pydantic (Structure)

Pydantic Validation



Documentation for version: v2.12.5.

Pydantic is the most widely used data validation library for Python.

Fast and extensible, Pydantic plays nicely with your linters/IDE/brain. Define how data should be in pure, canonical Python 3.9+; validate it with Pydantic.

Source: https://docs.pydantic.dev/latest/

Why Pydantic?
(Explanation)

Why use Pydantic?

- **Powered by type hints** — with Pydantic, schema validation and serialization are controlled by type annotations; less to learn, less code to write, and integration with your IDE and static analysis tools. Learn more…
- **Speed** — Pydantic's core validation logic is written in Rust. As a result, Pydantic is among the fastest data validation libraries for Python. Learn more…
- **JSON Schema** — Pydantic models can emit JSON Schema, allowing for easy integration with other tools. Learn more…
- **Strict** and **Lax** mode — Pydantic can run in either strict mode (where data is not converted) or lax mode where Pydantic tries to coerce data to the correct type where appropriate. Learn more…
- **Dataclasses**, **TypedDicts** and more — Pydantic supports validation of many standard library types including `dataclass` and `TypedDict`. Learn more…

Why Pydantic?
(Data → Schema)

```python
class User(BaseModel):
    id: int
    name: str = 'John Doe'
    signup_ts: datetime | None
    tastes: dict[str, PositiveInt]

external_data = {
    'id': 123,
    'signup_ts': '2019-06-01 12:22',
    'tastes': {
        'wine': 9,
        b'cheese': 7,
        'cabbage': '1',
    },
}

user = User(**external_data)
```

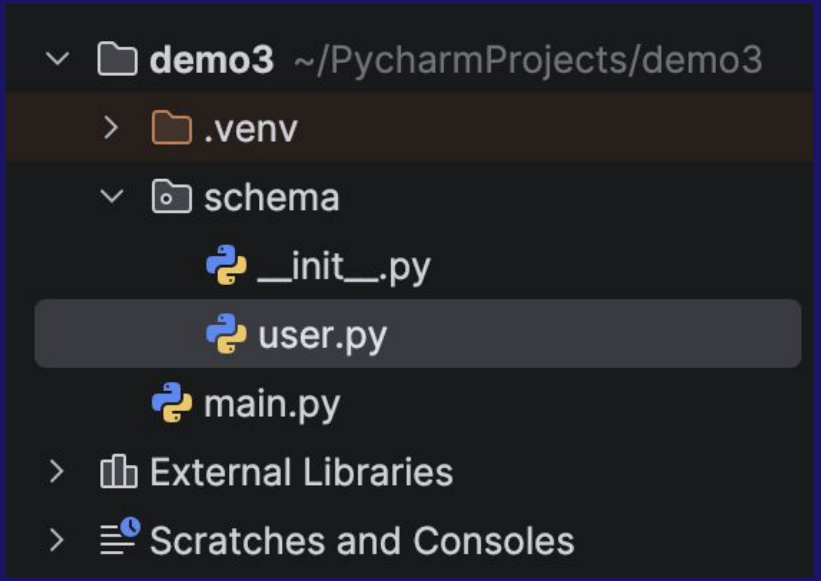
Vi kan göra om JSON ->
Python tolkade klasser.

03

Pydantic


Pydantic
Step by Step

Project Structure
(Folder)



NOTE:
__init__ is necessary for the class to be correctly imported into other .py files

What Data can a USER hold?
(User Schema)

username
password
is_enabled
is_authenticated
is_account_expired
is_credentials_expired
is_account_locked

id
email
first_name
last_name
is_subscribed_newsletter
role

Start easy - begin with username & password!

User Schema (BaseModel)

```python
from pydantic import BaseModel

class UserSchema(BaseModel):
    username: str
    password: str
```

By defining a **class**, we can then import and reuse it later.
BaseModel performs data validation behind the scenes.

Create new User (In-memory)

```python
# in-memory fake database
userList: list[UserSchema] = []
```

This **list** is intended to contain **only UserSchema** objects and as a result becomes **Type-hinted** and **IDE-assisted**.

Create new User (In-memory)

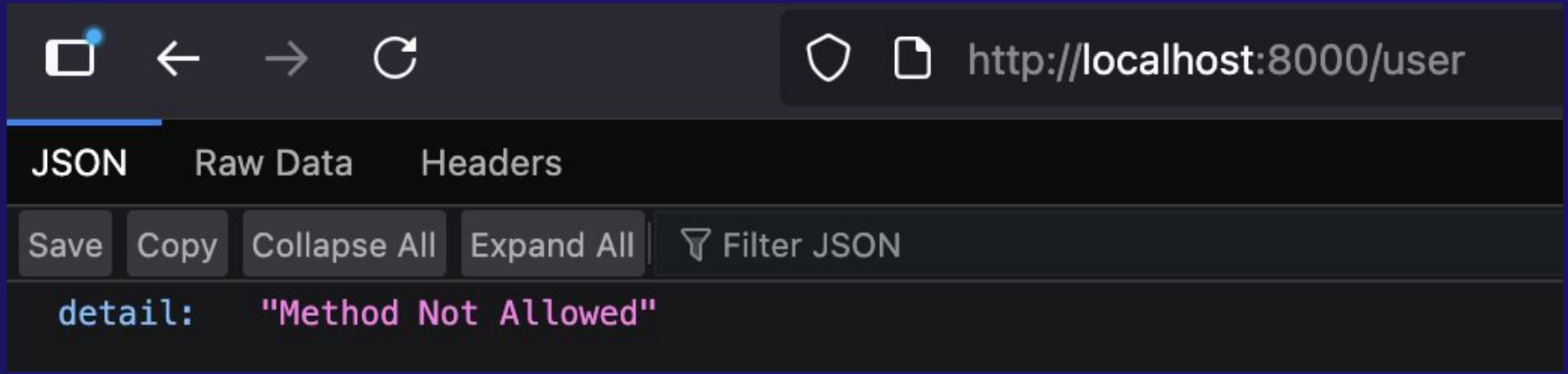
```python
@app.post("/users")
def create_user(user: UserSchema):
    userList.append(user)
    return {"user" : user}
```

This **list** is intended to contain **only UserSchema** objects and as a result becomes **Type-hinted** and **IDE-assisted**.

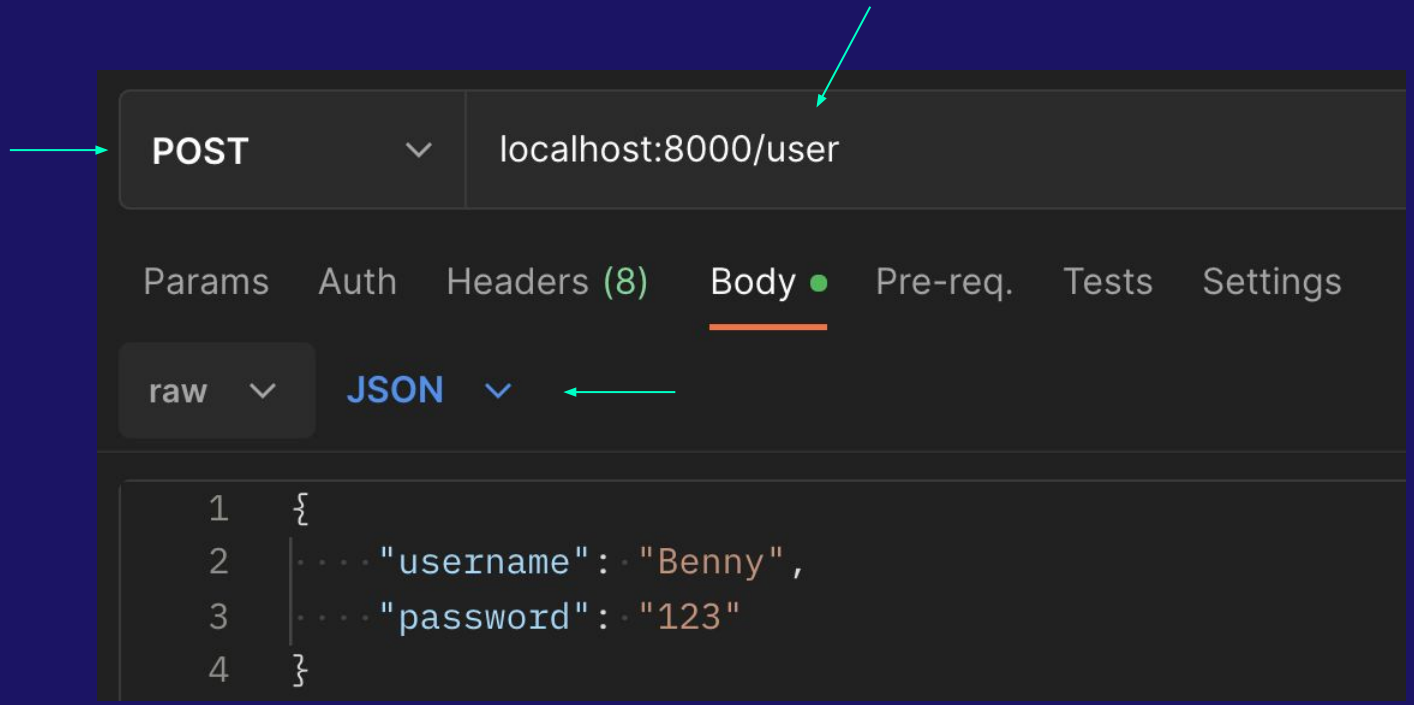
IMPORTANT: We are intentionally writing bad code here in the return statement. Ignore for now!

Posting in Browser
(Problem)

You can only perform GET requests in browsers *(bonus: check the network inspection tab)*

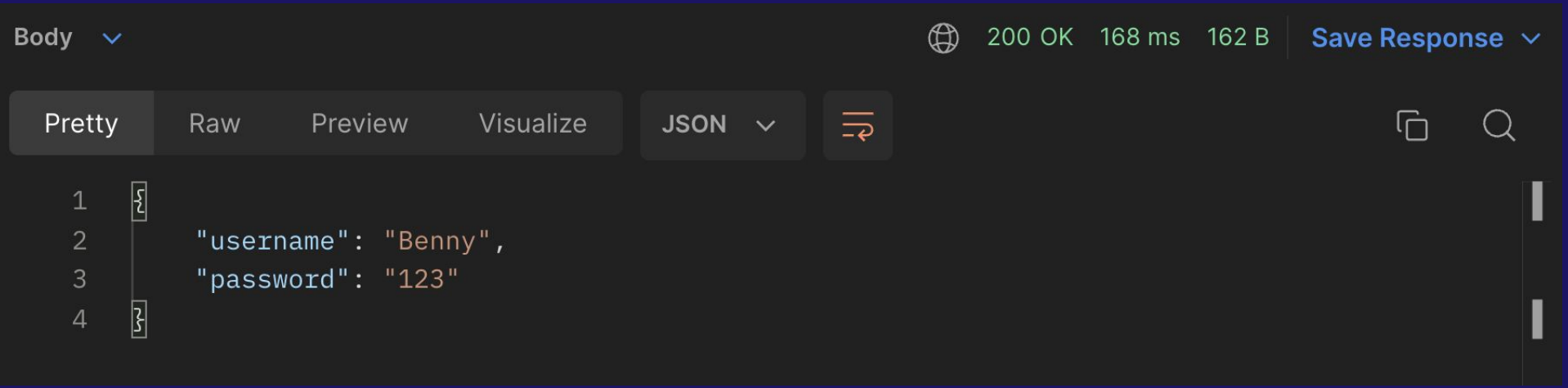


Postman
(Alternatives: Thunder Client or Curl)



200 OK
(Unclear Status code)

"200 ok" works, but we can make it clearer



201 Created (Explained)

201 Created

The HTTP **`201 Created`** successful response status code indicates that the HTTP request has led to the creation of a resource. This status code is commonly sent as the result of a `POST` request.

The new resource, or a description and link to the new resource, is created before the response is returned. The newly-created items can be returned in the body of the response message, but must be locatable by the **URL of the initiating request** or by the URL in the value of the `Location` header provided with the response.

Source: https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Status/201

Status Code (Implementation)

```python
from fastapi import FastAPI, status

@app.post("/users", status_code=status.HTTP_201_CREATED)
def create_user(user: UserSchema):
    userList.append(user)
    return {"user": user}
```

Our POST function's purpose and readability have improved dramatically

Debugging Exercise
(Problem)

```
from fastapi import FastAPI, status

@app.post("/users", status_code=status.HTTP_201_CREATED)
def create_user(user: UserSchema):
    userList.append(user)
    return {user}
```

Since our 'user' is an object, we don't need a dictionary... Notice, no hinting, no problems.

Debugging Exercise
(Returning expectations)

```python
@app.post("/users", status_code=status.HTTP_201_CREATED)
def create_user(user: UserSchema) -> UserSchema:
    userList.append(user)
    return {user}
```

This **helps showcasing problems**, very **helpful** in **debugging**.
This is considered a **best practice** and works very well **with type-safety** in **Python** using **FastApi**!

Done
(Return Schema only)

```
@app.post("/users", status code=status.HTTP 201 CREATED)
def create_user(user: UserSchema) -> UserSchema:
    userList.append(user)
    return user
```


GET
All Users

GET
(All users)

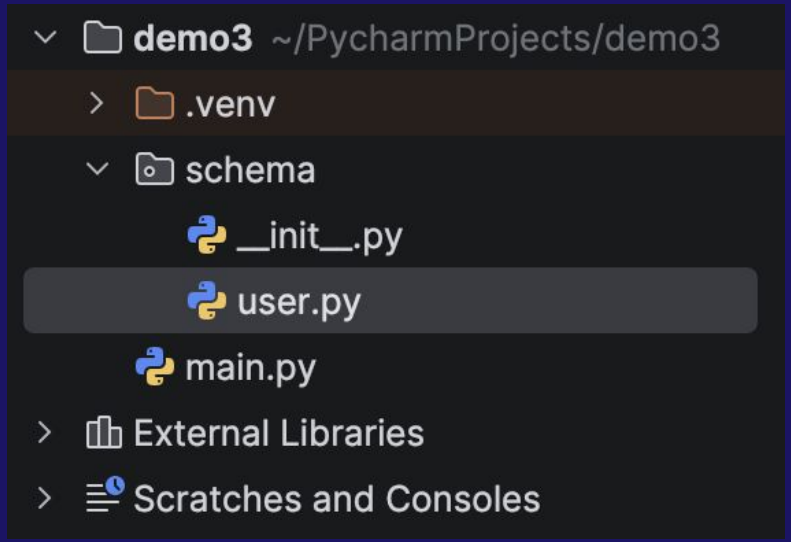
```python
@app.get("/users")
def get_users():
    return userList
```

Note: This works however, when you think about it.. are we actually returning safe data back to the client?

Result (bonus)
(The danger)

- **Leak-risk** (password)
- **Refactor-fragile**
- **Type-weak.**

UserSchemaOut (bonus)
(Excluding Data)




```python
class UserSchemaOut(BaseModel):
    username: str
```

Instead (bonus)
(Use Response_Model)

```python
@app.get("/users", response_model=list[UserSchemaOut])
```

- **contract enforcement** (if SQL changes, you'll notice)
- **automatic docs** (Swagger shows the shape)
- **type validation** (e.g. id is int, email is str)
- **safer refactors**

Debugging (bonus)
(What are we returning?)

```python
@app.get("/users", response model=UserSchemaOut)
def get_users() -> UserSchemaOut:
    result = []

    for user in userList:
        result.append(UserSchemaOut(username=user.username))

    return result
```

Notice what's expected in 'def' vs what's actually returned

Done (bonus)
(Return Schema only)

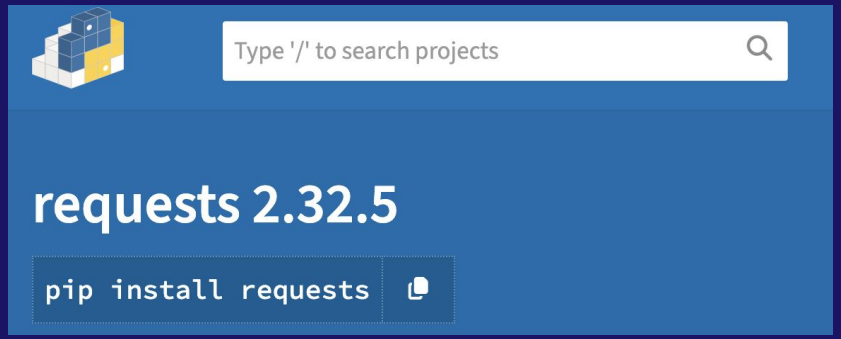
```python
@app.get("/users", response_model=list[UserSchemaOut])
def get_users() -> list[UserSchemaOut]:
    result: list[UserSchemaOut] = []

    for user in userList:
        result.append(UserSchemaOut(username=user.username))

    return result
```


Consuming API
Example: fox

Step #1
(Fox API)

Requests Library *(Dependency)*
https://pypi.org/project/requests/

API
https://randomfox.ca/floof/



Install
(requests dependency)

To stop server from running press
CTRL + C

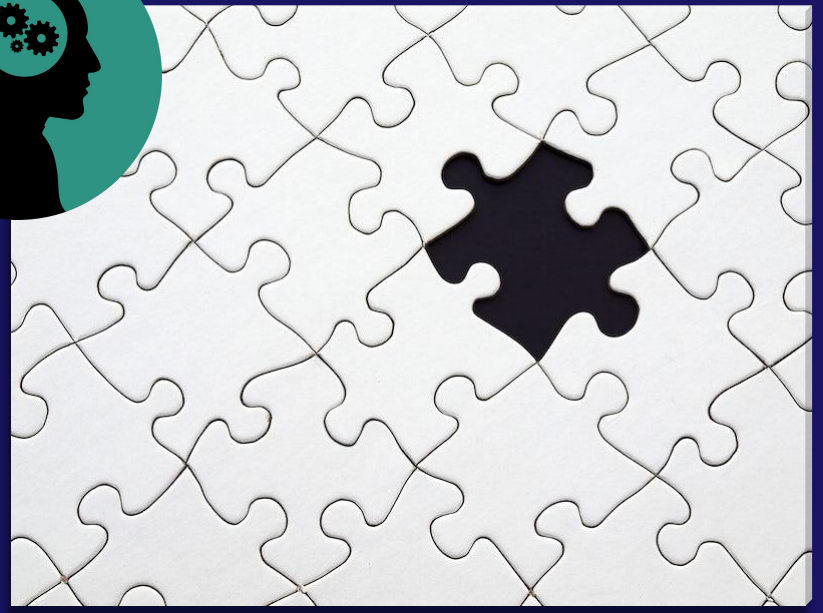
Step #2 - fetch from API (requests.get())

```python
@app.get("/fox", response model=FoxSchema)
def get_fox() -> FoxSchema:
    response = requests.get("https://randomfox.ca/floof/")
```

Step #3 - DONE
(JSON & Respond)

```python
@app.get("/fox", response_model=FoxSchema)
def get_fox() -> FoxSchema:
    response = requests.get("https://randomfox.ca/floof/")
    response_json = response.json()

    print(response)
    fox = FoxSchema(**response_json)

    return fox
```

Frågor?



04

Uppgifter & Eget Arbete

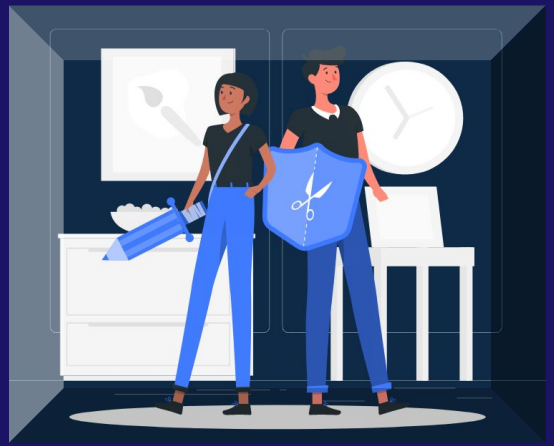
Uppgifter

Välkommen till första uppgiften!

Uppgifterna är till för att testa dina färdigheter och kunskaper för att både öva och repetera på det vi har arbetat med under föreläsningarna.

Dessa är **INTE** obligatoriska.

Men är ämnen ni kommer testas mot.



MINNS DU?

```
/*

    Förklara följande:
      • Mapping
      • Endpoint
      • HTTP-method
      • Query-parameter
      • Status Code

*/
```

Theory - Delete Mapping

```
/*
    På nästa sida presenteras kod som
    du ska analysera.

    Därefter får du ytterligare
    information om hur det presenterade
    problemet kan lösas.
*/
```

Theory - Delete Mapping

```python
@app.delete("/users", status_code=status.HTTP_200_OK)
def delete_user(username: str) -> dict[str, str]:
    for user in userList:
        if user.username == username:
            userList.remove(user)
            return {"message": "User deleted"}

    return {"message": "User not found"}


# Notera att nu kommer vi ALLTID att returnera 200 ok även om
# resultatet ej hittas…
```

Theory - Solution

```python
@app.delete("/users", status_code=status.HTTP_200_OK)
def delete_user(username: str) -> dict[str, str]:
    for user in userList:
        if user.username == username:
            userList.remove(user)
            return {"message": "User deleted"}

    raise HTTPException(
        status_code=status.HTTP_404_NOT_FOUND,
        detail="User not found",
    )

# Vi kan grena ut med HTTPException (glöm inte importera bara)
# 'raise' är likt 'throw' inom Java, C#, javascript/typescript, kotlin
```

```
1   // -Uppgift #1- //
2
3   /* INSTRUCTIONS
4
5       Skapa ett helt nytt projekt.
6       Installera rätt bibliotek:
7       https://fastapi.tiangolo.com/#create-it
8
9       Försök hitta hur en installerar!
10
11  */
12
13  // HINT & Examples
14  hint("Leta efter terminal kommandon")
15  hint("Installation inkluderar pydantic")
16
17
18
19
20
21
22
23
```

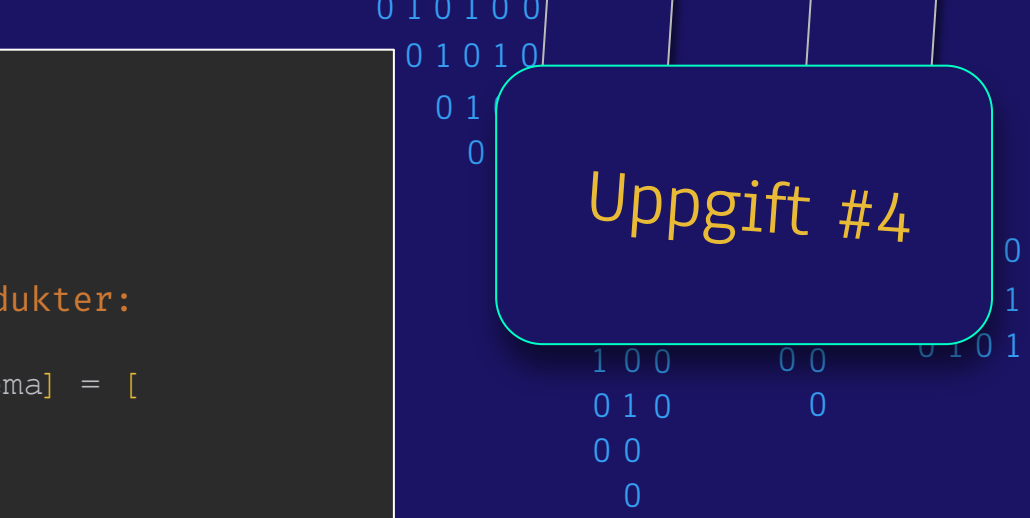
Kom igång enkelt med uppgift #1

```
1          // -Uppgift #2- //
2
3  /* INSTRUCTIONS
4
5      Skapa en 'app' variabel:
6
7      from fastapi import FastAPI
8      app = FastAPI(title="My First API")
9
10
11     Skapa nu en enkel 'Hello World'
12     GET-Mapping som använder sig av en
13     'Dictionary'
14
15 */
16
17 hint("@")
18 hint("app")
19 hint("get()")
20 hint("return {}")
21
22
23
```

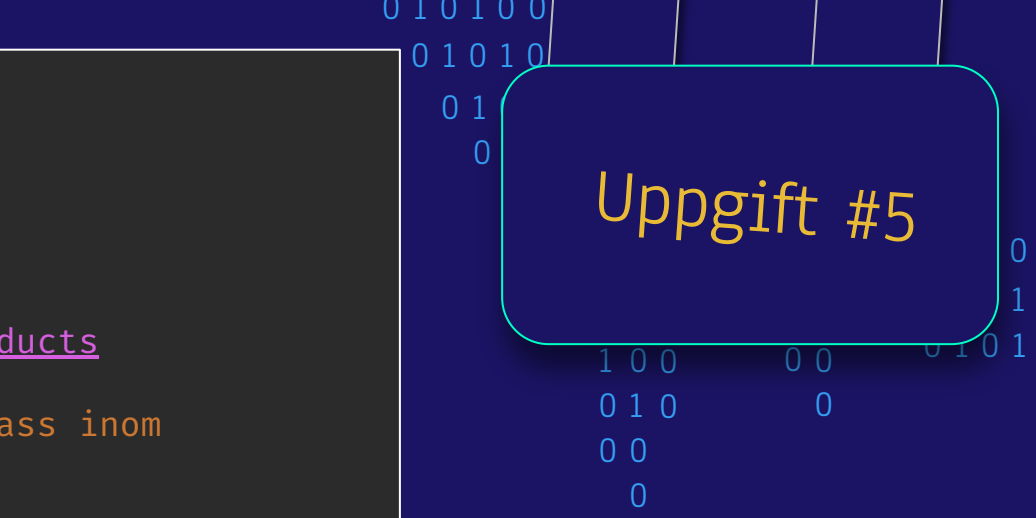
Uppgift #3

```
1   // -Uppgift #3- //
2
3   /* INSTRUCTIONS
4
5       Skapa ett nytt 'package'
6       Döp den till 'schema'
7
8       Skapa en ny .py fil: Product
9       class ProductSchema(BaseModel):
10          TBD: tbd
11
12      Inkludera
13        • id,
14        • title,
15        • price,
16        • description,
17        • category,
18        • image
19  */
20
21  // HINT & Examples
22  hint("Glöm inte data typer: str, int etc…")
23
```

```
1          // -Uppgift #4- //
2
3   /* INSTRUCTIONS
4
5        Inom main.py
6        Skapa en enkel array med produkter:
7
8        productList: list[ProductSchema] = [
9           ProductSchema(...),
10          ProductSchema(...),
11          ProductSchema(...),
12          ProductSchema(...),
13          ProductSchema(...),
14       ]
15
16       Hämta ut alla produkter inom en
17       'getProducts mapping'
18  */
19
20  // HINT & Examples
21  hint("Glöm inte att returnera listan inom 'def'
22  också för bättre struktur")
23
```

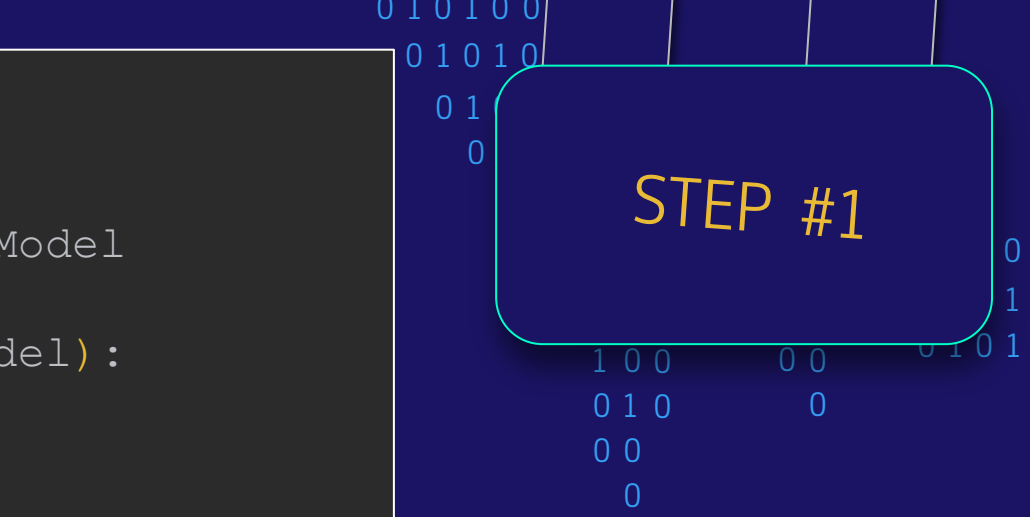


```
1   // -Uppgift #5- //
2
3   /* INSTRUCTIONS
4
5       Kolla på URL'n
6       Fakestore API
7       https://fakestoreapi.com/products
8
9       Skapa nu en till 'Schema' klass inom
10      product.py
11
12      Vi har redan definierat de 6 första värden…
13  */
14
15  // HINT & Examples
16  hint("Ignorera index elementen. Detta är enbart en
17  lista med objekt.
18  Fokusera enbart på objekt")
19  hint("rating")
20
21
22
23
```

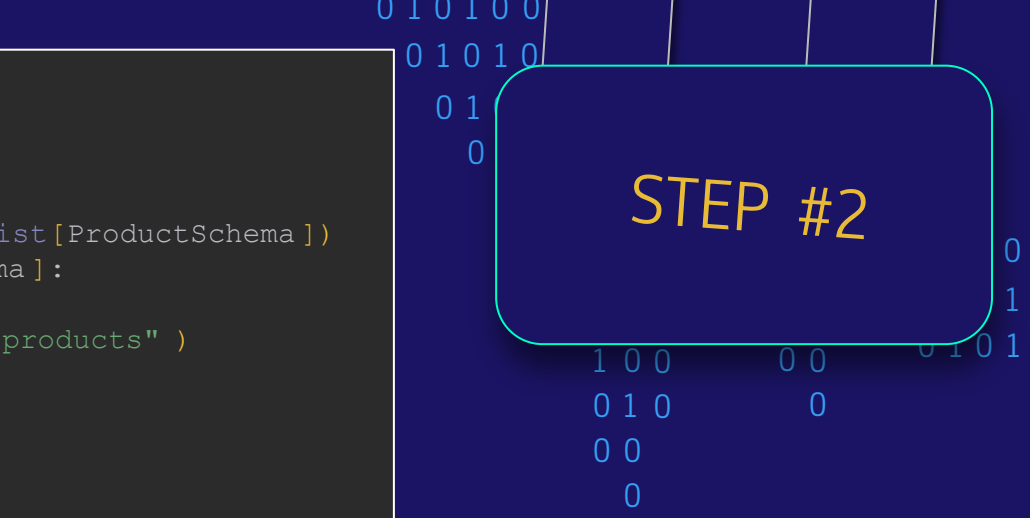


```
1   // -Uppgift #6- //
2      Tough nut
3   /* INSTRUCTIONS
4
5        Konsumera API:et
6        https://fakestoreapi.com/products
7
8        Använd datan för att visa upp alla produkter.
9        (Notera att allt är en enda stor array)
10
11       FACIT finns på nästkommande sidor!
12  */
13
14  // HINT & Examples
15  hint("List + For loop")
16
17  hint("Denna uppgift är svår, ta gärna hjälp av
18  google eller andra externa resurser")
19
20
21
22
23
```

```python
        # -FACIT- #
           Tough nut


from pydantic import BaseModel

class RatingSchema(BaseModel):
    rate: float
    count: int

class ProductSchema(BaseModel):
    id: int
    title: str
    price: float
    description: str
    category: str
    image: str
    rating: RatingSchema

```



```
1   // -FACIT- //
2       Tough nut
3
4
5   @app.get("/products", response_model=list[ProductSchema])
6   def get_products() -> list[ProductSchema]:
7       result =
8   requests.get("https://fakestoreapi.com/products")
9       response_json = result.json()
10
11      products: list[ProductSchema] = []
12
13      for item in response_json:
14          product = ProductSchema(**item)
15          products.append(product)
16
17      return list(products)
18
19
20
21
22
23
```



THANKS !

Do you have any questions?
kristoffer.johansson@sti.se